Oracle® WebCenter Framework

WebCenter アプリケーションの構築 - ステップ・バイ・ステップ

10*g*（10.1.3.2.0）

部品番号 : E05617-02

2008 年 7 月

ORACLE®

Oracle WebCenter Framework WebCenter アプリケーションの構築 - ステップ・バイ・ステップ , 10*g* (10.1.3.2.0)

部品番号 : E05617-02

原本名 : Oracle WebCenter Framework Building a WebCenter Application Step by Step, 10*g* (10.1.3.2.0)

原本部品番号 : B31073-01

原本協力者 : Lalithashree Rajesh, Peter Lubbers, Promila Chitkara, Vanessa Wang, Barry Hiern, Candace Fender, Oliver Ricordel, Peter Moskovits, Philipp Weckerle, Sue Vickers

目次

はじめに v

対象読者 vi
ドキュメントのアクセシビリティについて vi
関連ドキュメント vi
表記規則 vii
サポートおよびサービス vii

1 WebCenter Suite の例の概要

Oracle WebCenter Suite の概要 1-2
Oracle WebCenter Framework 1-3
ポートレットの構築および使用 1-3
カスタマイズ可能コンポーネント 1-4
コンテンツ統合 1-4
アプリケーションの保護 1-4
ライフサイクル全体にわたるアプリケーションの管理 1-5
Oracle WebCenter Services 1-5
Oracle JDeveloper 1-6
この例で作成するもの 1-7
開始方法 1-9
完成したデモのナビゲート方法 1-10
デモの実行 1-10
顧客としてデモを表示する方法 1-11
技術者としてデモを表示する方法 1-11
管理者としてデモを表示する方法 1-11

2 ポートレットの構築

始める前に 2-2
ポートレット用のプロジェクトの作成 2-2
Oracle WebCenter Preconfigured OC4J へのアプリケーション・サーバー接続の作成 2-3
手順 1: Product Details ポートレット（JSR 168 ポートレット）の構築 2-5
JSR 168 ポートレットの作成 2-5
JSR 168 ポートレット用の Web サービス・プロキシの作成 2-8
JSR 168 ポートレットへのポートレット・ロジックの追加 2-10
アプリケーション・サーバーへの JSR 168 ポートレットのデプロイ 2-12
プロデューサの登録 2-15
手順 2: Service Request Status ポートレット（PDK-Java ポートレット）の構築 2-17
PDK-Java ポートレットおよびプロデューサの作成 2-17

PDK-Java ポートレットへのポートレット・ロジックの追加 2-23
アプリケーション・サーバーへの PDK-Java ポートレットのデプロイ 2-28
まとめ 2-30

3 スキンの設定

手順 1: スキンが登録されていることの確認 3-2
手順 2: アプリケーションが新しいスキンを使用するように構成されていることの確認 3-3
まとめ 3-3

4 コンテンツ・リポジトリの設定

手順 1: サンプル・コンテンツ用のコンテンツ・ディレクトリの設定 4-2
手順 2: コンテンツ・プロジェクトの作成 4-2
手順 3: JCR データ・コントロールの構成 4-4
まとめ 4-6

5 パブリックなようこそページの作成

手順 1: リッチ・テキスト・コンポーネントの追加 5-2
手順 2: 所定の場所でのファイルの表示 5-4
手順 3: グローバル・ナビゲーション・リンクの追加 5-6
手順 4: ようこそページのパブリック化 5-11
まとめ 5-12

6 ログイン・ページの作成

手順 1: ログイン・ページの作成 6-2
手順 2: ログイン・エラー・ページの作成 6-12
手順 3: ログイン・ページへのリッチ・テキスト・ポートレットの追加 6-15
手順 4: ログイン・ページ用の認可の編集 6-16
手順 5: ADF セキュリティおよびログイン・ページを使用するためのアプリケーションの構成 6-17
手順 6: 新しいログイン・ページをコールするための Web.xml の更新 6-19
手順 7: アプリケーションの実行 6-20
まとめ 6-20

7 ページの構築とコンポーネントの追加

手順 1: 「MyAcme」ページの作成 7-2
手順 2: OmniPortlet プロデューサの登録 7-5
手順 3: カスタマイズ・コンポーネントの追加 7-5
手順 4: ADF 表への SelectOneChoice の連結 7-13
手順 5: JavaServer Faces ドロップダウン・コンポーネントの追加 7-14
手順 6: Service Request History ポートレットの追加 7-14
手順 7: Service Request Status ポートレットの追加 7-16
手順 8: ポートレットへの表の連結 7-16
表への OmniPortlet の連結 7-17
Service Request 表への Service Request Status ポートレットと Service Request History ポートレットの連結 7-18
手順 9: Product Details ポートレットの追加 7-18
手順 10: フォルダの内容の表示 7-19
手順 11: Schedule On-Site Services ポートレットの追加 7-20
Web クリッピング・プロデューサの登録 7-20

Web クリッピング・ポートレットの追加 7-21
Web クリッピング・ポートレットに表示する Web ページの選択 7-21
手順 12: ユーザーに基づいた最新契約の追加 7-24
手順 13: 「My Acme」ページへのセキュリティの適用 7-25
手順 14: コンポーネントへのセキュリティの適用 7-26
まとめ 7-26

8 ダッシュボード・ページの構築

手順 1: ダッシュボード・ページの作成 8-2
手順 2: ページ・レイアウトの作成 8-4
手順 3: ページへの OmniPortlet のインスタンスの追加 8-9
手順 4: Service Request Volume ポートレット用の SelectOneChoice コンポーネントの追加 8-10
手順 5: 顧客契約の検索機能の追加 8-11
手順 6: Most Productive Employees ポートレットの定義 8-11
手順 7: Service Request Volume ポートレットの定義 8-12
手順 8: Most Requested Products ポートレットの定義 8-13
手順 9: Most Active Customers ポートレットの定義 8-13
手順 10: Customer Details ポートレットの定義 8-14
手順 11: ページ・コンテンツの連結 8-15
まとめ 8-16

9 サイト管理ページの構築

手順 1: サイト管理ページの作成 9-2
手順 2: ページへのスキン・セレクタの追加 9-3
手順 3: ログイン・ページのカスタマイズの可能化 9-8
手順 4: 「Management」ページのサブタブとしての「Site Administration」ページの追加 9-9
手順 5: カスタマイズのためのサイト管理ページの使用 9-11
アプリケーションのスキンの変更 9-11
ログイン・ページのカスタマイズ 9-12
まとめ 9-13

10 アプリケーションのデプロイ

手順 1: 汎用 EAR ファイルの作成 10-2
手順 2: ターゲット EAR ファイルの作成 10-4
手順 3: ユーザーおよびロールの設定 10-6
手順 4: サンプル・コンテンツの設定 10-7
手順 5: アプリケーションのデプロイ 10-7
手順 6: セキュリティ・ポリシーのデプロイ 10-9
手順 7: アプリケーションへのアクセス 10-10
まとめ 10-10

索引

はじめに

このマニュアルでは、Oracle WebCenter Framework を使用して WebCenter アプリケーションを構築する方法について説明します。このマニュアルで使用する例は、『Oracle Application Development Framework 開発者ガイド』に示されている ADF アプリケーションに基づいており、既存の Oracle ADF アプリケーションにポータルのような機能を追加する方法を示します。

対象読者

このマニュアルは、WebCenter アプリケーションを構築する WebCenter アプリケーション開発者または Oracle WebCenter Framework を使用してアプリケーションにカスタマイズ機能を追加するアプリケーション開発者を対象としています。したがって、開発者が次の事柄に精通していることを前提としています。

- Java
- Oracle JDeveloper
- Oracle Application Development Framework（Oracle ADF）
- Oracle ADF Faces
- 『Oracle WebCenter Framework チュートリアル』

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

詳細は、次のドキュメントを参照してください。

- 『Oracle WebCenter Framework Release Notes』
- 『Oracle WebCenter 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。
イタリック体	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。

1

WebCenter Suite の例の概要

この章では、Oracle WebCenter Suite の概要と機能およびこのマニュアルで使用されるシナリオの概要について説明します。

この章では、次の重要な項目について説明します。

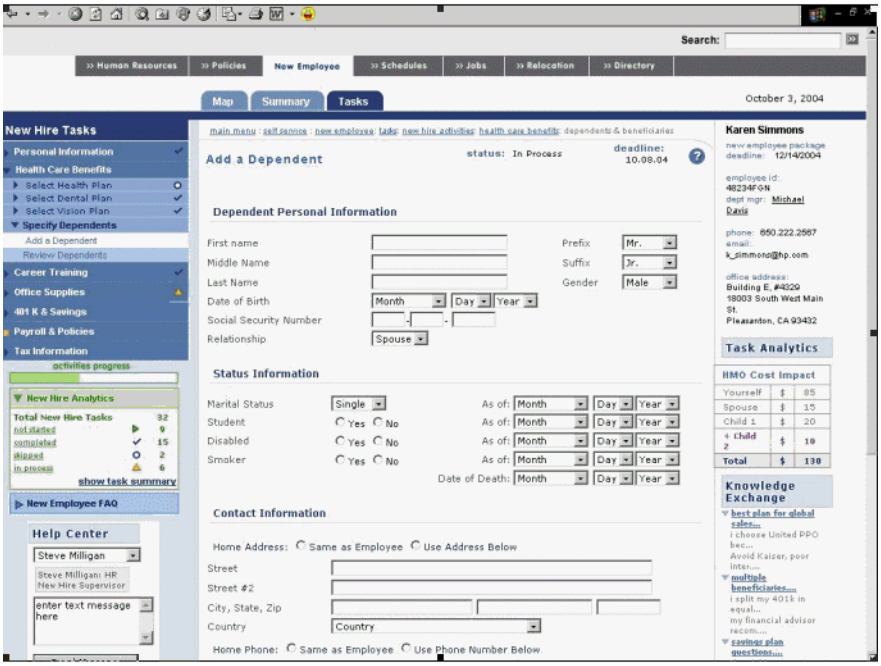
- Oracle WebCenter Suite の概要
- この例で作成するもの
- 開始方法
- 完成したデモのナビゲート方法

この章を読み終えたら、独自の WebCenter アプリケーションの作成を開始できます。

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につれて、トランザクションを簡略化するアプリケーションに対するユーザーの要求は顕著になってきています。トランザクションを簡略化する 1 つの方法は、特定の作業に対応するの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。図 1-1 に示す例で考えてみます。

図 1-1 サンプル・アプリケーション



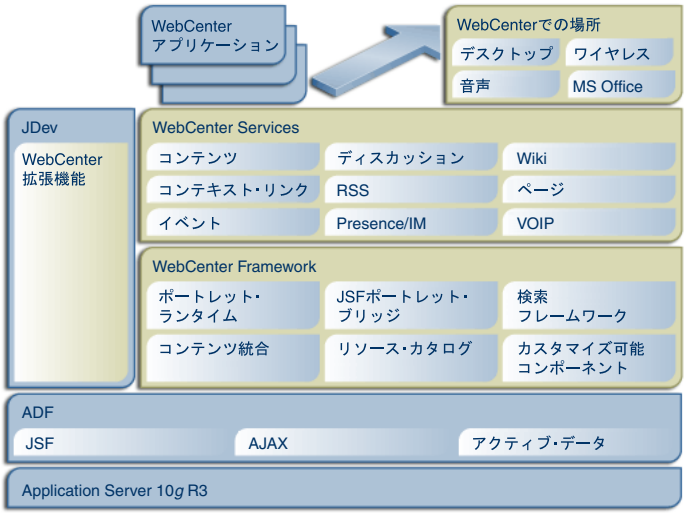
この例では、入社したばかりのユーザーが、会社の保険契約に扶養家族を追加できるアプリケーションを使用しています。トランザクションそのものの周囲に、次のようなユーザーの理解を助けるコンテキストが他にもあることに注意してください。

- 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。このようなプロセス編成により、ユーザーは複数の手順で構成されるフロー全体を素早く簡単にたどることができます。
- 作業およびプロセスの分析手法により、ユーザーはプロセスにおける自分の位置と、決定内容が自分に及ぼす影響を認識することができます。この例では、右側の「Task Analytics」に、これまで選択してきた給付内容の全体的な費用効果が示されます。
- 左下の「Help Center」には、一般的な質問にすぐにアクセスできる最新の FAQ と、FAQ には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、トランザクション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、ユーザーに該当する様々な受給者や扶養家族のシナリオについて詳細なアドバイスを提供します。

Oracle WebCenter Suite までは、この種のアプリケーションの構築は非常に冗長なプロセスでした。たとえば、アプリケーション・プログラミング・インタフェース（API）での必要に応じて、JCR 1.0 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にアクセスすることは可能でした。Oracle WebCenter Suite では、サービス指向アーキテクチャ（SOA）の概念を利用することで、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のに必要とされたフロントエンドの労力を軽減しています。

Oracle WebCenter Suite は、SOA の他に JCR 1.0 Java Content Repository（JSR 170）などの業界標準にも準拠しているため、多種多様なプラグ・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の構築が簡単になります。図 1-2 に、Oracle WebCenter Suite の構成を示します。[1]

図 1-2 Oracle WebCenter Suite



では、これらの構成要素を詳しく見ていきます。

Oracle WebCenter Framework

Oracle 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 Oracle Application Server Portal（OracleAS Portal）製品に組み込まれていた機能を JSF 環境の構造に直接統合します。これにより、ユーザーに対する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使用

ポートレットを使用すると、Web やデータベースなどからアプリケーションにデータを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使用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は、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で、あるいは同じページ内でも使用できます。

Preconfigured Oracle Containers for J2EE（OC4J）を介して使用できる、事前に構築された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で、ユーザーが使用できるようにすることができます。また、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可能です。

[1] 図に示されているコンポーネントの中には、Oracle WebCenter Suite の初期のリリースでは使用できないものがあります（Presence/IM、ディスカッションおよび Wiki）。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。

- 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。OmniPortlet では、スプレッドシート（文字で区切られた値）、XML、Web サービス、既存の Web ページからのアプリケーション・データなど、ほとんどのすべてのデータソースをベースとすることができます。データは、取得後に箇条書きリスト、グラフ、HTML などのレイアウトを使用して書式設定できます。

  開発者は、（従業員ディレクトリを作成する場合などに）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し、ユーザーが使用できるようにページ上に配置できます。配置されたポートレットは、JDeveloper のコンポーネント・パレットから使用可能になり、他のユーザーはそれぞれのアプリケーションで使用できます。

- Web クリッピング : 技術的な専門知識を一切必要としない、非常に使いやすいウィザード。ユーザーは、クリップする Web コンテンツを探し、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ると、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ブロードキャストを公開できるようにするツール。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。

カスタマイズ可能コンポーネント

WebCenter Framework には、新しい JSF コンポーネントが用意されており、開発者はアプリケーションをカスタマイズできるよう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、開発者はこのコンテナの中に別の 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、管理者はページ上のコンポーネントを最小化 / 最大化、非表示 / 表示または移動することで、ほとんどの JSF ページをカスタマイズできます。

コンテンツ統合

Oracle Content DB、OracleAS Portal などのコンテンツ管理システム（あるいはファイル・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できるようにするとします。WebCenter Framework には、そのコンテンツにアクセスするのに必要な 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 リポジトリの内容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 Oracle ADF 拡張機能を使用すると、アプリケーション全体、アプリケーション内のページ、カスタマイズ可能コンポーネントによって提供される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。このマニュアルでは、パブリックにアクセスされるログイン・ページの作成方法について説明します。このログイン・ページでは、ユーザーが資格証明を入力すると様々なページやページ内のコンポーネントにアクセスできるようになります。また、ログイン / ログアウト・リンクを作成する方法および権限を各種ロールに関連付ける方法については、例を示します。

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ことが望ましいです。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 では、次のような複数のツールを使用することで、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 Oracle 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用意されています。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 環境のいずれかに該当するオブジェクトをドラッグ・アンド・ドロップするだけです。テストおよびデバッグのフェーズを簡略化するために、WebCenter Framework には、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 J2EE コンテナ（付属のスタンドアロン OC4J など）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およびデバッグできます。
- 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により、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ます。また、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必ず指すようにします。デプロイ前ツールの作業が完了すると、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を使用すると、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り、管理者はきめ細やかに制御し、状態ステータス、パフォーマンスおよびアクセス数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）過去のパフォーマンスおよびステータスのレポートを経時的に取得するためのツールも提供されます。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。

Oracle WebCenter Services

Oracle 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ようにす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にアクセスすることで、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、コンテンツに付随する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情報テクノロジ（IT）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、非常に幅広く使用できる専門的で、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ものです。
- Oracle Secure Enterprise Search は、クローラベースのサービスで、多種多様なソース（構造化 / 非構造化データ、様々なファイル形式のデータ、索引付きデータ、リアルタイム・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検索する時間を短縮できます。

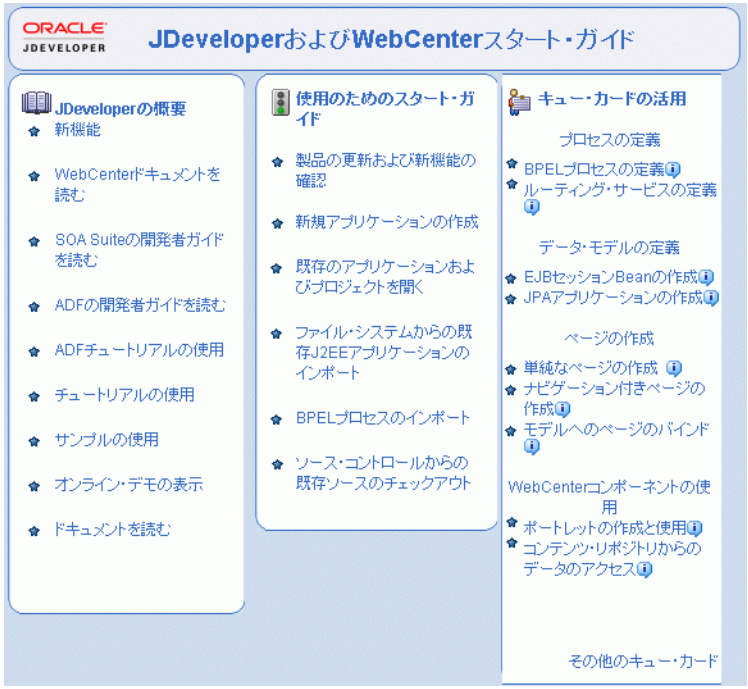
- 人と人とを接続しやすくし、通信を促進するための通信サービス。次のようなサービスがあります。
  - インスタント・メッセージング : オーディオおよびビデオ・フィード、ファイル交換、その他様々な機能を介して、ユーザーがアイデアを自由に交換できるようにします。
  - プレゼンス・サーバー : プレゼンスは、ある人が自分のステータスにサブスクライブしている個々人またはアプリケーションに対して使用可能であるかどうかの情報を提供します。チャットなどのリアルタイム・サービスは、関連付けられたユーザー・インタフェースから開始できます。
  - ディスカッション・フォーラム : 情報、質問およびコメン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機能により、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 ADF が連動して機能することにより、アプリケーション開発は簡略化され、日常的なコーディング作業が削減されます。たとえば、多くの標準的なユーザー・インタフェース・ウィジェット（ボタン、値リスト、ナビゲーション・バーなど）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。

このマニュアルを読み進むうちに、Oracle JDeveloper とそのメリットについての理解が深まっていきます。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）から、数多くある参考資料の 1 つにアクセスしてください。

図 1-3 Oracle JDeveloper の開始ページ



この例で作成するもの

このマニュアルの手順では、ポートレットの追加方法、既存のコンテンツの統合方法、アプリケーションへのカスタマイズの追加方法およびインタラクティビティの作成方法を示します。このようなインタラクティビティには、ポートレット対ポートレットの通信や ADF Faces コンポーネント対ポートレットのインタラクティビティがあります。また、実行時に管理者がページをカスタマイズできるようにする方法についても学びます。

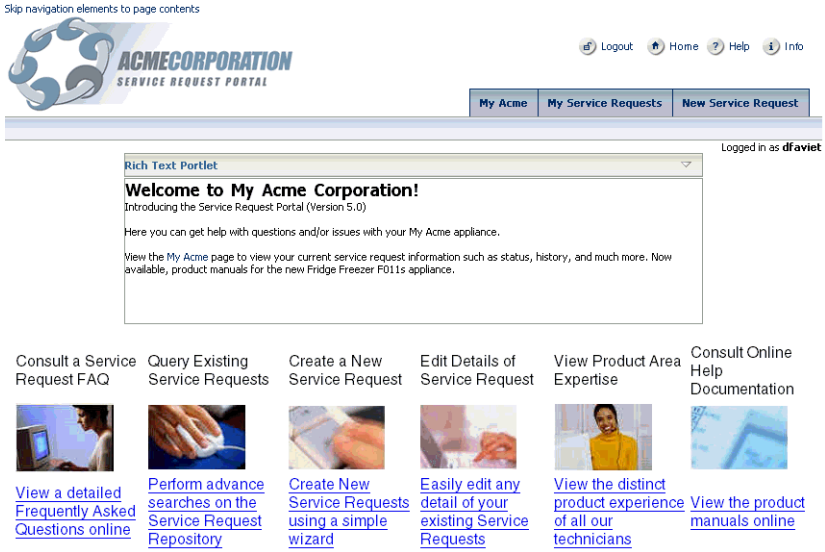
このマニュアルでは、顧客サービス・リクエストを追跡管理するための既存の ADF アプリケーションを利用し、既存のアプリケーションを変更せずにポータル機能を追加します。この Service Request アプリケーションを使用すると、顧客、技術者および管理者は、サービス・リクエストに関する情報をすべて同じインタフェースから確認できます。シナリオは次の 3 つです。

- 顧客（My Acme 社の顧客）
- 技術者（My Acme 社に対するサービス・リクエストを処理する技術者）
- 管理者（My Acme 社の Web サイトを管理し、技術者チームを指揮する管理者）

顧客

顧客がアプリケーションにログインすると、最新のお知らせおよび既存のサービス・リクエストとその詳細を表示できます（図 1-4 を参照）。また、これまで購入した製品に関する情報やサービスを提供している企業との現在の契約のリストも表示できます。既存のサービス・リクエストについて、フィードバックを送信することも可能です。

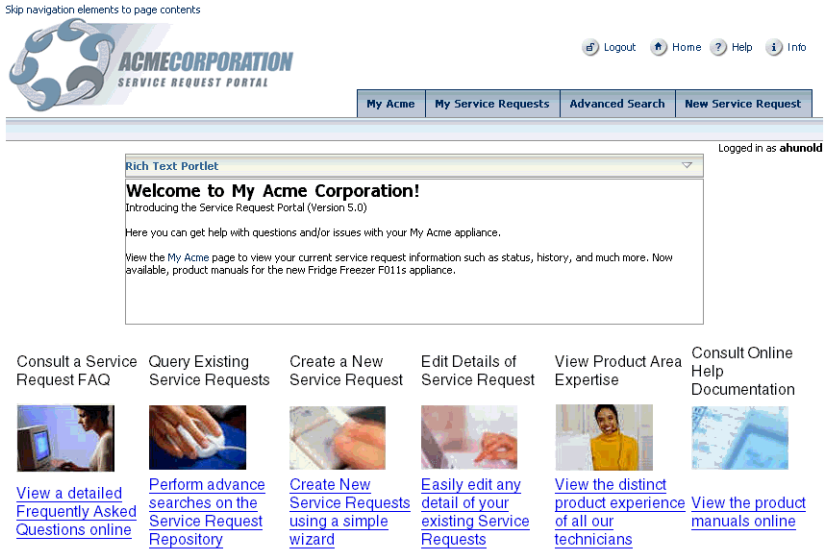
図 1-4 Service Request アプリケーションの顧客用の表示



技術者

技術者がアプリケーションにログインすると、自分に割り当てられたサービス・リクエストを表示して（図 1-5 を参照）、既存のサービス・リクエストを更新できます。

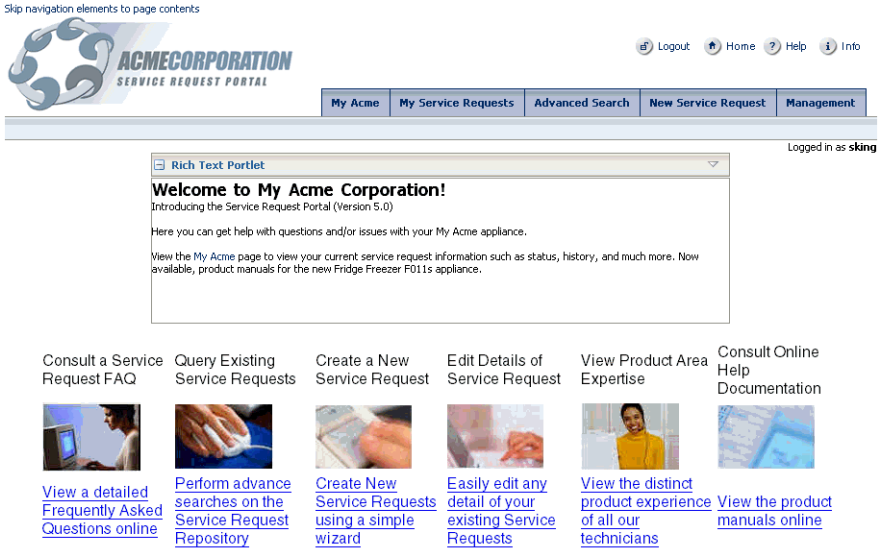
図 1-5 Service Request アプリケーションの技術者用の表示



管理者

管理者がアプリケーションにログインすると、実行時に、顧客に表示されるお知らせを更新できます。また、実行時に、コンテンツ・リポジトリ内のコンテンツを使用してページを変更できます。たとえば、現在、新規サービスを顧客が使用できる場合、管理者はこの新規サービスに関する情報を実行時に追加できます。顧客が返してきたフィードバックを検討して、管理者独自のメモを追加することもできます。さらに、ダッシュボード・ページを使用してサイトの統計を表示したり、このダッシュボード・ページをカスタマイズすることも可能です。ダッシュボード・ページには、現在のサービス・リクエストの量、最もアクティブ顧客などが表示されます。管理者は、ある 1 つのページからサイトの一般管理を実行することもできます。このページでは、スキンの切り替えによるルック・アンド・フィールの変更およびログインのカスタマイズが可能です。図 1-6 に、アプリケーションの管理者用の表示を示します。

図 1-6 Service Request アプリケーションの管理者用の表示



開始方法

始める前に、サンプル・アプリケーションとサポート・ファイルをダウンロードして設定します。このマニュアルに関連するサンプル・ファイルは、Oracle Technology Network（OTN）の Oracle WebCenter Suite の「Documentation」ページ（`http://www.oracle.com/technology/products/webcenter/documentation.html`）にあります。`SRDemo_App_Download.ZIP` ファイルをダウンロードして解凍し、`install.html` ファイルの指示に従ってください。ファイルの設定が終了したら、次の項の説明に従って完成したデモを表示できます。あるいは、第 2 章「ポートレットの構築」に進んでアプリケーションの構築を開始できます。

完成したデモのナビゲート方法

サンプル・ファイルのインストールおよび設定を終了すると、完成した作業バージョンのデモを表示できるため、後続の各章で構築するものを確認できます。この項では、デモとその実行時の使用方法について説明します。

デモの実行

デモを実行するには、次のようにします。

1. ブラウザで、次の URL に移動します。

   `http://localhost:8888/SRDemo/faces/app/SRWelcome.jspx`

   > **注意：** アプリケーションでは、すでにホスト名が指定されたポートレットを URL または DB 接続に使用します。アプリケーションが十分機能するよう、ホスト名を変更する必要があります。

2. 次のログイン情報を使用してアプリケーションにログインします。

   ユーザー ID: `sking`

   パスワード : `welcome`

3. 表示される最初のページは、Service Request アプリケーションのようこそページです。このページは、パブリック（非認証）ユーザーに表示され、「Help and About」ページが含まれます。ようこそページには 2 つのコンポーネント、すなわちリッチ・テキスト・ポートレットおよびコンテンツ・リポジトリの HTML コンテンツがあります。

   - リッチ・テキスト・ポートレットは、稼働中のアプリケーションから直接実行時に、サイト管理者が新しいお知らせやテキストなどの HTML の更新を簡単にできるようにします。ログインしていないユーザーは、このポートレットの表示およびリフレッシュしかできません。
   - ようこそページのイメージおよびテキストは、ファイルベースのコンテンツ・リポジトリ内にある HTML コンテンツです。

4. **「My Acme」**タブをクリックして、アプリケーション内の他のコンポーネントの一部を設定および表示します。

5. 「My Acme」ページで、**「Schedule On-Site Service」**サブタブをクリックして Web クリッピング・ポートレットにアクセスします。

6. **「Customize」**をクリックします。すると、「手順 11: Schedule On-Site Services ポートレットの追加」の手順に従ってポートレットの Web ページを選択できます。

7. 次に、OmniPortlet のデータベース接続をデモに設定し、実行できるようにする必要があります。**「SR Information」**サブタブをクリックします。

8. 「Service Request History」ポートレットの横の「Customize」をクリックします。

9. 「ソース」タブで、**「接続の編集」**をクリックし、データベース接続の詳細を入力します。データベース接続情報は、サンプル・ファイルのインストール時に入力した情報と同じである必要があります。

   > **注意：** デモで 1 つの OmniPortlet インスタンスの接続を編集すると、すべての OmniPortlet インスタンスで同じ接続情報を使用します（たとえば、この時点で「My Dashboard」ページ上のポートレットはすべて正常に機能しています）。同じ接続情報を使用していないことが判明した場合は、いつでもそれぞれの OmniPortlet をカスタマイズできます。

10. **「終了」**をクリックします。これで、アプリケーション内のポートレットを使用できます。

顧客としてデモを表示する方法

アプリケーションに作成されているユーザーには、顧客、技術者および管理者の 3 種類があります。顧客の観点からアプリケーションを表示するには、次の情報を使用してログインします。

- ユーザー ID: dfaviet
- パスワード : welcome

顧客は、サービス・リクエストのステータスおよび履歴を表示できます。「MyAcme」タブをクリックし、サービス・リクエストを選択して詳細を表示します。

技術者としてデモを表示する方法

顧客の観点からアプリケーションを表示するには、次の情報を使用してログインします。

- ユーザー ID: ahunold
- パスワード : welcome

技術者も、顧客のサービス・リクエストのステータスおよび履歴を表示できます。「My Acme」タブをクリックし、オープン・サービス・リクエストの 1 つを選択してそのサービス・リクエストに関する最新情報を表示します。そのサービス・リクエストは、「My Service Requests」タブで変更できます。

また、顧客が参照できない他のコンポーネントにアクセスできます。「My Acme」ページの「Product Information」サブタブの新規コンポーネント（JSR 168 ポートレット）を使用すると、特定の洗濯機や乾燥機などの製品の詳細を表示できます。

最後に、「Schedule On-Site Service」サブタブにアクセスできます。このサブタブでは、外部アプリケーションにより、スケジュール可能な次のオンサイト・サービスを表示できます。

管理者としてデモを表示する方法

顧客の観点からアプリケーションを表示するには、次の情報を使用してログインします。

- ユーザー ID: sking
- パスワード : welcome

管理者は、アプリケーション内で使用可能なすべてのタブを表示できます。「Management」タブをクリックし、「Dashboard」サブタブをクリックしてアプリケーション内のサービス・リクエストに関する情報を表示します。次の詳細情報を確認できます。

- 顧客契約の検索。たとえば、`df%` と入力して**「Search」**をクリックし、顧客 `dfaviet` の契約を表示します。
- 最もアクティブな顧客の詳細の表示。たとえば、**「Urman」**をクリックし、Jose Manul Urman に関する顧客の詳細を表示します。
- 一定期間におけるサービス量の表示。

管理者は、サイト管理者でもあります。再度**「Management」**タブをクリックし、**「Site Administration」**をクリックします。使用可能なスキンの 1 つを選択して、アプリケーションのスキンを変更できます。

2

ポートレットの構築

この章では、JSR 168 を使用する Java ポートレットと Oracle 固有の API を使用する PDK-Java ポートレットの 2 種類のポートレットを構築する方法について学びます。

この章の内容は次のとおりです。

- 始める前に
- 手順 1: Product Details ポートレット（JSR 168 ポートレット）の構築
- 手順 2: Service Request Status ポートレット（PDK-Java ポートレット）の構築

始める前に

この章の手順を始める前に、ダウンロードしたサンプル・ファイルに含まれる `install.html` ファイルの手順を必ず完了しておきます。次に、次の手順を実行します。

- ポートレット用のプロジェクトの作成
- Oracle WebCenter Preconfigured OC4J へのアプリケーション・サーバー接続の作成

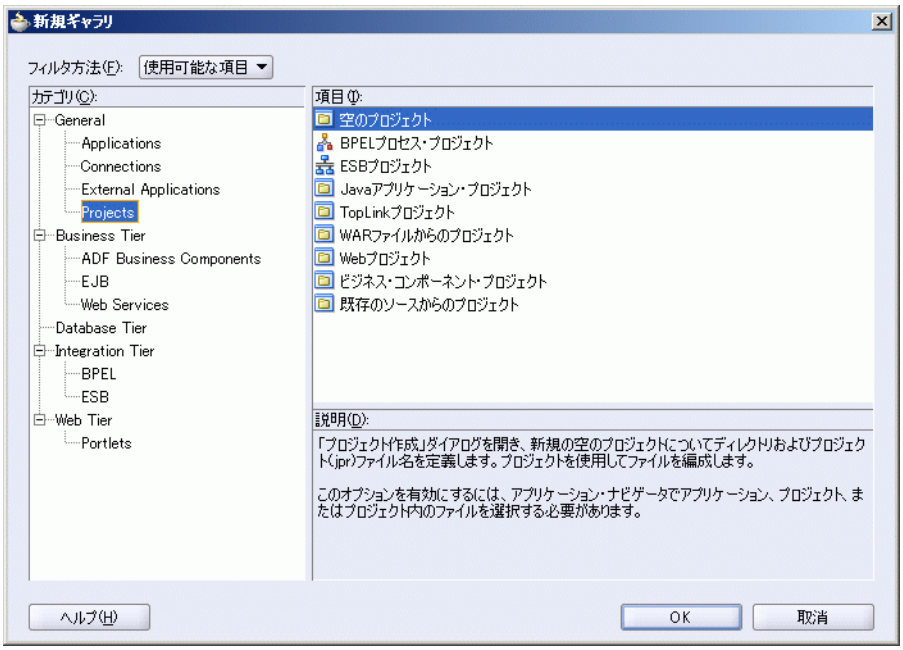
ポートレット用のプロジェクトの作成

アプリケーション内のものを見つけやすくするために、プロジェクトを作成してポートレットを格納することができます。

ポートレット用のプロジェクトを作成するには、次のようにします。

1. アプリケーション・ナビゲータで、**「SRDemoSample_Starter」**アプリケーションを右クリックし、**「新規プロジェクト」**を選択します。
2. 「項目」リスト（図 2-1）で**「空のプロジェクト」**を選択し、**「OK」**をクリックします。

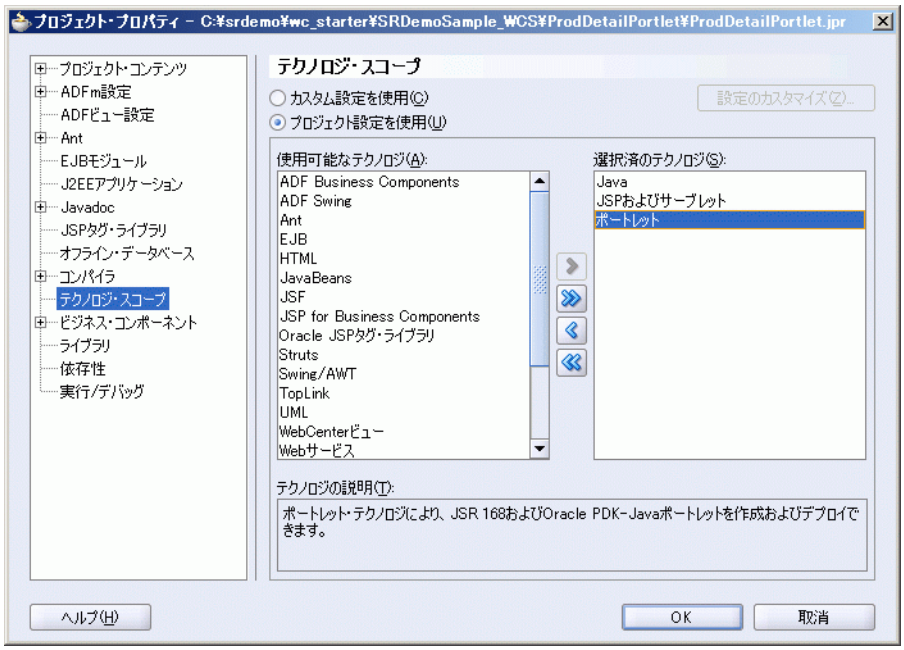
図 2-1 空のプロジェクトの作成



3. **「プロジェクト名」**フィールドに、`ProdDetailPortlet` と入力します。
4. デフォルトのディレクトリ名を使用し、**「OK」**をクリックしてプロジェクトを作成します。
5. プロジェクト・プロパティを表示するには、プロジェクト名（この場合は**「ProdDetailPortlet」**）を右クリックし、ポップアップ・メニューから**「プロジェクト・プロパティ」**を選択します。

6. プロジェクトのテクノロジ・スコープ（プロジェクトに含めるテクノロジ）、すなわち「Java」、「JSP およびサーブレット」、「ポートレット」を選択します（図 2-2 を参照）。

図 2-2 新規プロジェクトのテクノロジ・スコープ



7. 「**OK**」をクリックします。
8. 再度、手順 1 ～ 7 に従って 2 つ目のプロジェクトを作成し、`SRStatusPortlet` と名前を付けます。
9. アプリケーションを保存します。

Oracle WebCenter Preconfigured OC4J へのアプリケーション・サーバー接続の作成

ポートレットをアプリケーション・サーバーの Oracle JDeveloper にデプロイするには、まず、Oracle JDeveloper に埋め込まれている Oracle WebCenter Preconfigured OC4J への接続を確立する必要があります。

Preconfigured OC4J へのアプリケーション・サーバー接続を作成するには、次のようにします。

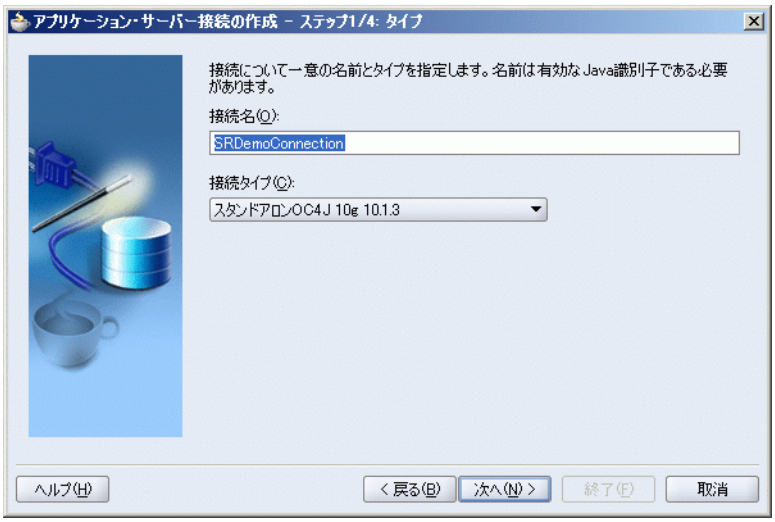
1. Oracle JDeveloper で、**「WebCenter Preconfigured OC4J の起動」**アイコンをクリックします。`readme.html` ファイルが表示されます。このファイルには、Preconfigured OC4J に関する接続情報が格納されています。
2. 接続ナビゲータで、**「接続」**を右クリックし、**「アプリケーション・サーバー接続の作成」**を選択します。

   ヒント： 接続ナビゲータが表示されない場合は、「表示」メニューから**「接続ナビゲータ」**を選択します。

3. 「アプリケーション・サーバー接続の作成」ウィザードの「ようこそ」ページが表示されたら、**「次へ」**をクリックして「タイプ」ページを表示します。

4. **「接続名」**フィールド（図 2-3）に、SRDemoConnection と入力します。
5. 「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 2-3 アプリケーション・サーバー接続の作成：タイプ



6. **「次へ」**をクリックして「認証」ページを表示します。
7. 関連する中間層に対する Application Server Control コンソール管理者のユーザー名およびパスワードを入力します。通常、管理者のユーザー名は oc4jadmin、パスワードは welcome です。
8. **「次へ」**をクリックして「接続」ページを表示します。
9. **「ホスト名」**フィールドに、localhost と入力します。
10. **「RMI ポート」**フィールドに、Remote Method Invocation（RMI）サーバーが OC4J サーバーにバインドするためのポート番号を入力します。Preconfigured OC4J のデフォルト値は、22667 です。
11. **「次へ」**をクリックして「テスト」ページを表示します。
12. **「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、成功メッセージが表示されます。テストが失敗した場合は、接続情報を確認する必要があります。
13. **「終了」**をクリックします。

手順 1: Product Details ポートレット（JSR 168 ポートレット）の構築

単純な JSR 168 ポートレットを構築するために、サンプルの Service Request Status ポートレットを構築する手順を実行します。このポートレットをデプロイすると、「My Acme」ページに追加できます。Ajax を使用してより高度にインタラクティブな JSR 168 ポートレットのライブ・デモ・バージョンを表示するには、Service Request デモの完成バージョンを実行するだけです。

JSR 168 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。

JSR 168 ポートレットを構築するには、次の作業を行います。

- JSR 168 ポートレットの作成
- JSR 168 ポートレットへのポートレット・ロジックの追加
- アプリケーション・サーバーへの JSR 168 ポートレットのデプロイ

JSR 168 ポートレットの作成

Oracle WebCenter Framework には、Java ポートレット・ウィザードが用意されているため、ポートレットのモードごとに単純な実装を迅速に作成できます。このウィザードを使用して、ポートレットの作成を開始します。

JSR 168 ポートレットを作成するには、次のようにします。

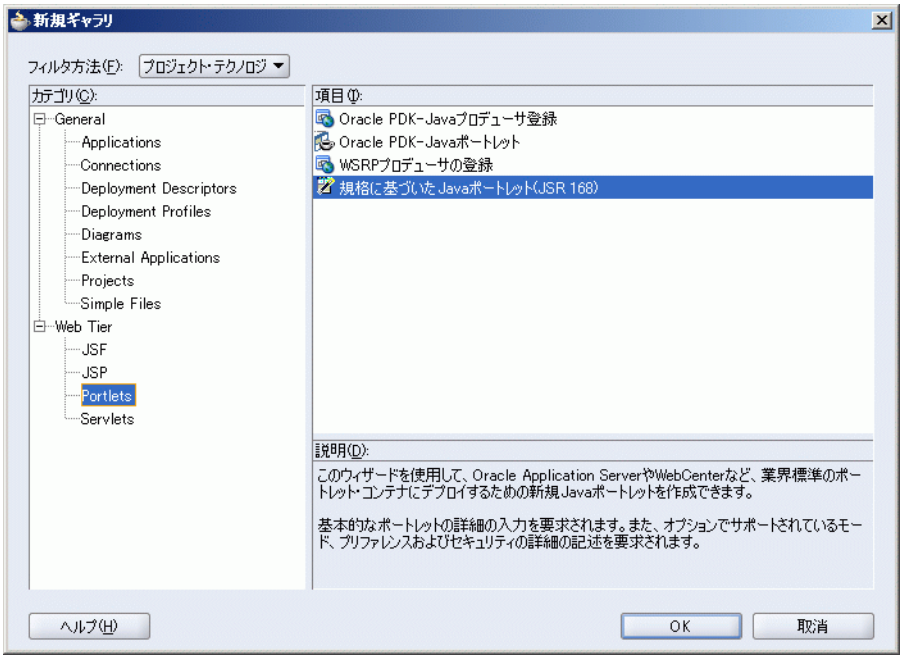
1. Oracle JDeveloper のアプリケーション・ナビゲータで、**「SRDemoSample_Starter」**アプリケーション内の**「ProdDetailsPortlet」**プロジェクトを右クリックし、**「新規」**を選択します。

   ヒント： このプロジェクトが存在しない場合は、この章の「ポートレット用のプロジェクトの作成」の項で説明した指示に従います。

2. 新規ギャラリで、**「Web Tier」**カテゴリを開き、**「Portlets」**を選択します。

3. 「項目」リスト（図 2-4）で、**「規格に基づいた Java ポートレット (JSR 168)」**を選択して JSR 168 ポートレットを構築します。

図 2-4 新規 JSR 168 ポートレットの作成



4. **「OK」**をクリックして JSR 168 Java ポートレット・ウィザードを表示します。
5. ウィザードの「ようこそ」ページが表示されたら、**「次へ」**をクリックして「Web アプリケーション」ページを表示します。
6. **「サーブレット 2.4¥¥JSP 2.0 (J2EE 1.4)」**を選択します。

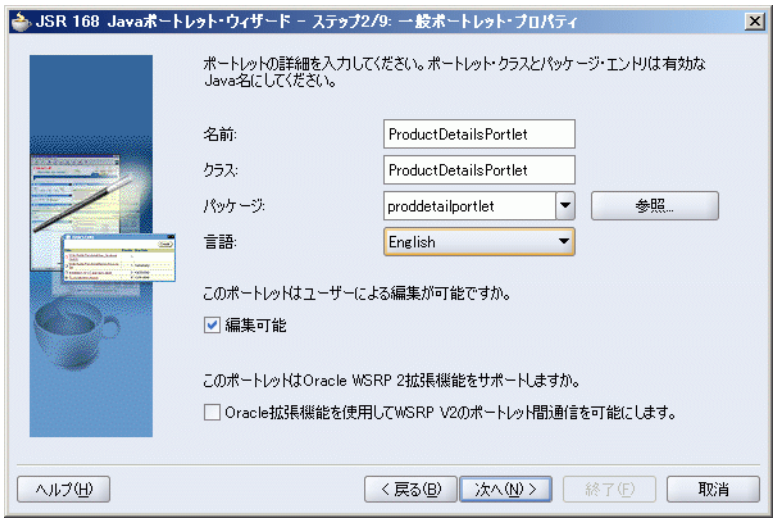
   ヒント： プロジェクトに対して Web アプリケーションがすでに作成されている場合は、ウィザードのこのページは表示されず、直接「一般ポートレット・プロパティ」ページが表示されることがあります。

7. **「次へ」**をクリックして「一般ポートレット・プロパティ」ページを表示します。

8. **「名前」**および**「クラス」**フィールド（図 2-5）に ProductDetailsPortlet と入力し、**「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。」**チェック・ボックスの選択を解除します。

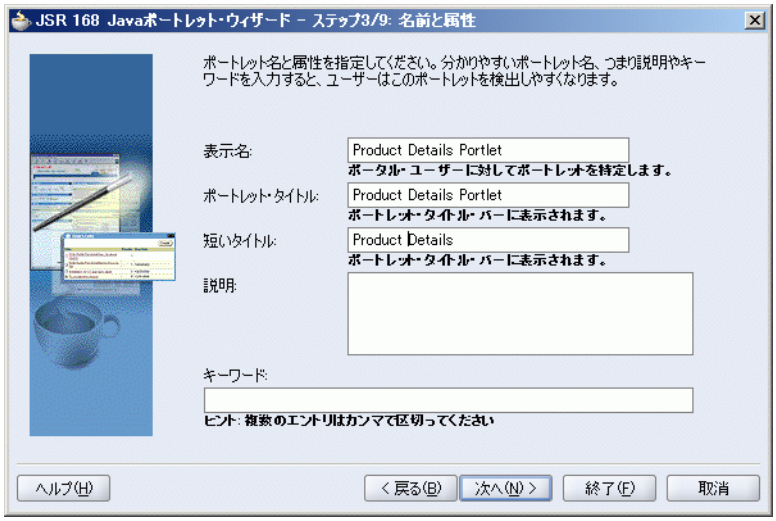
   ポートレットを表すこの名前でクラスが作成されます。

図 2-5 JSR 168 ポートレットの一般ポートレット・プロパティ



9. **「次へ」**をクリックします。
10. **「表示名」**フィールド（図 2-6）に、Product Details Portlet と入力します。
11. **「ポートレット・タイトル」**フィールドに、Product Details Portlet と入力します。
12. **「短いタイトル」**フィールドに、Product Details と入力します。

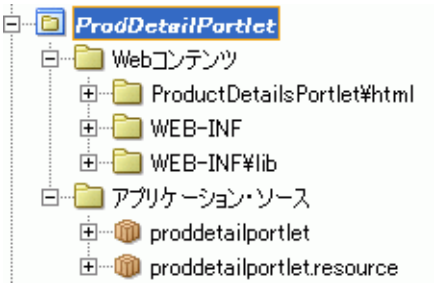
図 2-6 JSR 168 ポートレットの名前と属性



13. **「終了」**をクリックし、ポートレットの残りの属性についてデフォルト設定を受け入れます。

    これで、Product Details ポートレットがアプリケーション・ナビゲータに表示されます（図 2-7 を参照）。

図 2-7 アプリケーション・ナビゲータでの Product Details ポートレット



14. 作業内容を保存します。

JSR 168 ポートレット用の Web サービス・プロキシの作成

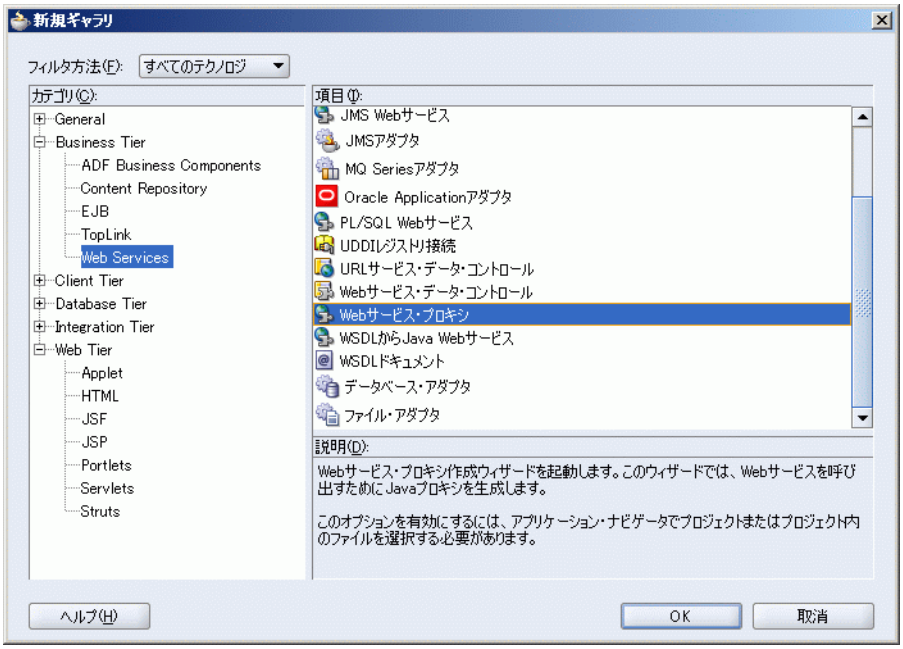
JSR 168 ポートレットでは、Web サービスをデータソースとして使用します。アプリケーション内のポートレットで Web サービスを使用するには、まず、ポートレット・プロジェクト用に Web サービス・プロキシを作成する必要があります。

Web サービス・プロキシを作成するには、次のようにします。

1. ProductDetailsPortlet が含まれる「ProdDetailPortlet」プロジェクトを右クリックし、**「新規」**を選択します。
2. 「Business Tier」ノード（図 2-8）の下の**「Web Services」**を選択し、**「Web サービス・プロキシ」**を選択します。

> **ヒント：** 新規ギャラリに「Business Tier」ノードが表示されない場合は、「フィルタ方法」ドロップダウン・リストから「すべてのテクノロジ」オプションを選択してフィルタ処理をする必要があります。

図 2-8 新規ギャラリ：Web サービス・プロキシ



3. 「**OK**」をクリックします。
4. ウィザードの「ようこそ」ページで、「**次へ**」をクリックします。
5. 「Web サービス記述」ページ（図 2-9）で、次のような Web サービスの WSDL を入力します。

```
http://localhost:8888/ProductDetailsWS/ProductDetailsWSSoapHttpPo
rt?WSDL
```

> **注意：** この WSDL では、localhost およびポートは、Web サービスをデプロイしたシステムを指しています。Web サービスは、SRDemo OC4J にデプロイされます。

図 2-9 Web サービス記述の設定

6. 「**次へ**」をクリックします。

> **注意：**「モデルを作成中」ダイアログ・ボックスが表示されない場合は、「**次へ**」を再度クリックする必要があります。

7. Web サービスのモデルが作成されると、Web サービスのエンドポイント URL を検証できるページ（図 2-10）が表示されます。

図 2-10 Web サービス・ポート・エンドポイントの検証

8. **「終了」**をクリックし、Web サービス・プロキシの残りのパラメータについてデフォルト値を受け入れます。

9. **ProductDetailsWSSSoapHttpPortClient.java** ファイルを閉じます。

JSR 168 ポートレットへのポートレット・ロジックの追加

この時点で、ポートレットはほとんど機能しません。実際には、ようこそメッセージを表示し、エンド・ユーザーがポートレット・タイトルをパーソナライズできるようにしているだけです。ポートレットが必要な特定の機能を実行するようにするには、必要な機能を実装するビジネス・ロジックでサンプル・コードを拡張する必要があります。

まず、ポートレットに製品タイプのリストを表示させます。このリストは Web サービスから取得します。その後で、ポートレットを拡張し、エンド・ユーザーが製品タイプを選んで特定の製品を選択できるようにします。製品が選択されると、詳細情報が簡単な説明およびイメージとともに表示されます。

JSR 168 ポートレットにポートレット・ロジックを追加するには、次のようにします。

1. アプリケーション・ナビゲータで、**「Web コンテンツ」**ノードを開いて**「view.jsp」**を右クリックし、**「開く」**を選択してビジュアル・エディタにファイルを開きます。

   ヒント： view.jsp を探すには、SRDemoSample_Starter アプリケーションの下で、**「Portlets」→「Web コンテンツ」→「ProductDetailsPortlet¥html」**の順にノードを開きます。

2. ビジュアル・エディタの下部で、**「ソース」**タブをクリックしてページのソース・コードを表示します。

3. 既存のコードを例 2-1 に示すコードで置き換えます。

例 2-1 view.jsp のコード

```
<%@ page contentType="text/html"
import="javax.portlet.*,java.util.*,portlets.ProductDetailsPortlet,portlets.resource.Pr
oductDetailsPortletBundle"%>

<%@ page contentType="text/html"
import="portlets.proxy.ProductDetailsWSSoapHttpPortClient,portlets.proxy.types.productd
etailsws.types.ProductDetailsBean"%>

<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>

<portlet:defineObjects/>

<%
    String prodType = renderRequest.getParameter("prodType");
%>

<table cellpadding="10%">
 <tr>
  <td>
   <font class="portlet-section-header">
      Product Types
   </font>
  </td>
  <td>
   <font class="portlet-section-header">
      Products
   </font>
  </td>
  <td>
   <font class="portlet-section-header">
      Details
   </font>
  </td>
 </tr>
 <tr>
   <td>
<%
   ProductDetailsWSSoapHttpPortClient proxy = new ProductDetailsWSSoapHttpPortClient();
   Vector productTypes =  proxy.getProductTypes();
       for (int i=0; i<productTypes.size(); i++)
       {
          PortletURL productURL = renderResponse.createRenderURL();
          productURL.setParameter ("prodType", productTypes.elementAt(i).toString());
%>

         <font class="portlet-font">

           <a href="<%=productURL%>">
             <%= productTypes.elementAt(i) %>
           </a></font><br>
<%
       }
%>

  </td>
 </tr>
</table>
```

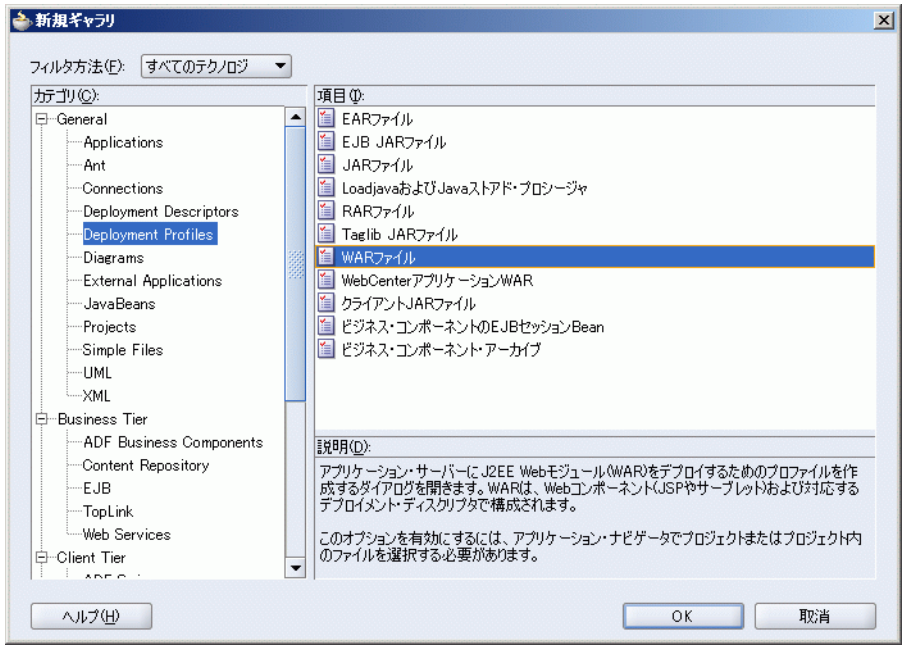
4. 「ファイル」メニューから、**「すべて保存」**を選択します。

アプリケーション・サーバーへの JSR 168 ポートレットのデプロイ

ポートレットの構築が終了したら、パッケージ化してポートレット・コンテナにデプロイする必要があります。

1. アプリケーション・ナビゲータで、**「ProdDetailPortlet」**プロジェクトを右クリックし、**「新規」**を選択します。
2. 新規ギャラリで、**「General」**カテゴリを開き、**「Deployment Profiles」**を選択します。
3. 「項目」リスト（図 2-11）で、**「WAR ファイル」**を選択します。

図 2-11 WAR デプロイメント・ファイルの作成

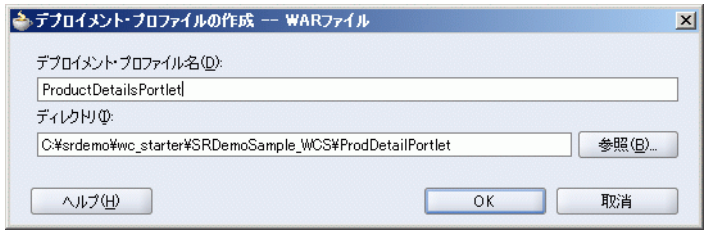


4. **「OK」**をクリックして「デプロイメント・プロファイルの作成 -- WAR ファイル」ダイアログ・ボックスを表示します。

   ヒント： このダイアログ・ボックスは、アプリケーション・ナビゲータで web.xml ファイルを右クリックし、「WAR デプロイメント・ファイルの作成」を選択しても表示できます。

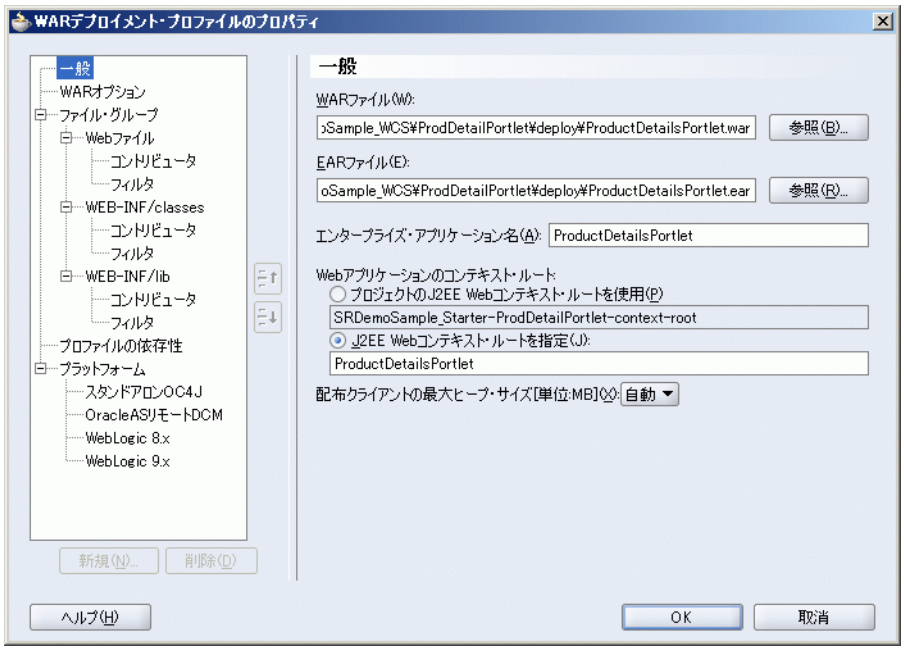
5. **「デプロイメント・プロファイル名」**フィールド（図 2-12）に、`ProductDetailsPortlet` と入力します。
6. デフォルト・ディレクトリを使用します。

図 2-12「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



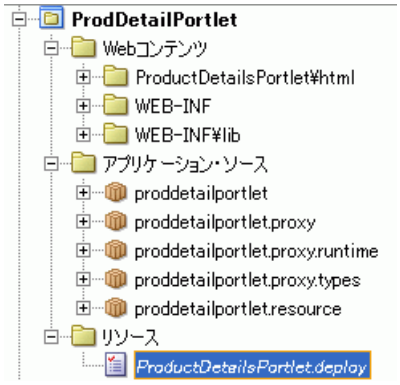
7. 「**OK**」をクリックして「WAR デプロイメント・プロファイルのプロパティ」ダイアログ・ボックスを表示します。
8. 「**J2EE Web コンテキスト・ルートを指定**」を選択し、対応するフィールドに `ProductDetailsPortlet` と入力します（図 2-13 を参照）。

図 2-13 WAR デプロイメント・プロファイルのプロパティ



9. 「**OK**」をクリックします。
10. アプリケーション・ナビゲータ（図 2-14）で、「**ProdDetailPortlet**」→「**リソース**」の順にノードを開き、デプロイメント・ファイルを確認します。

図 2-14 アプリケーション・ナビゲータでの ProductDetailsPortlet.deploy

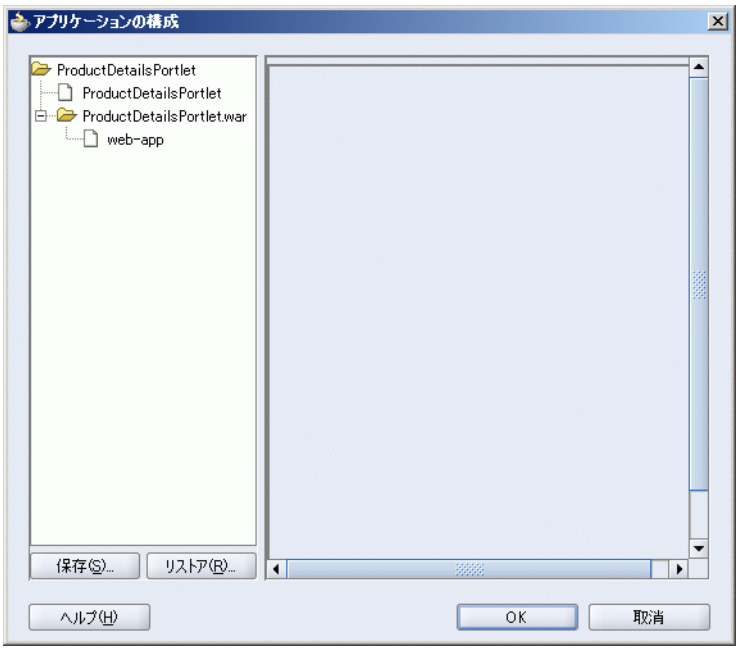


11. 「ファイル」メニューから、「**すべて保存**」を選択します。

12. 「**ProductDetailsPortlet.deploy**」を右クリックし、図 2-15 に示す「配布先」メニューから「**SRDemoConnection**」を選択します。

    「アプリケーションの構成」ダイアログ・ボックスが表示されます。

図 2-15「アプリケーションの構成」ダイアログ・ボックス



注意： メニューに「SRDemoConnection」が表示されない場合は、この章の「Oracle WebCenter Preconfigured OC4J へのアプリケーション・サーバー接続の作成」の項で示した手順に従います。

---

13. 「**OK**」をクリックします。
14. 「デプロイ - ログ」に「---- デプロイが終了 ----」メッセージが表示されたら、エラーが発生していないことを確認します。

プロデューサの登録

ポートレットは作成およびデプロイされたので、アプリケーションで使用する準備ができました。ポートレット・プロデューサをアプリケーションに登録して、アプリケーション開発者がポートレットをページに追加できるようにします。

プロデューサを登録するには、次のようにします。

1. ポートレット・プロデューサはアプリケーションに関連付けられるため、アプリケーションまたはアプリケーション内の任意のプロジェクトを右クリックして起動します。しかし、最も簡単な方法は、**UserInterface** プロジェクトから新規ギャラリを開くことです。UserInterface プロジェクトのテクノロジ・スコープは、プロデューサ登録ウィザードが新規ギャラリ内に表示されるように設定されています。
2. ポップアップ・メニューから**「新規」**を選択します。
3. **「Web Tier」**の**「Portlets」**で、**「WSRP プロデューサの登録」**を選択し、**「OK」**をクリックします。

   ヒント： 新規ギャラリに「Business Tier」ノードが表示されない場合は、「フィルタ方法」ドロップダウン・リストから「すべてのテクノロジ」オプションを選択してフィルタ処理をする必要があります。

4. 「ようこそ」ページで、**「次へ」**をクリックします。
5. 「名前」ページ（図 2-16）で、**「名前」**フィールドに `ProductDetailsProducer` と入力し、**「次へ」**をクリックします。

図 2-16 WSRP プロデューサ名

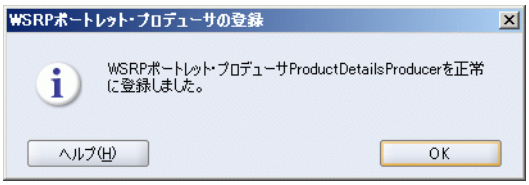
6. 「接続」ページ（図 2-17）で、次のような Preconfigured OC4J の WSRP コンテナの URL を入力します。

   ```
   http://localhost:6688/ProductDetailsPortlet/portlets/wsrp1?WSDL
   ```

図 2-17 WSRP プロデューサ URL

7. 「**次へ**」をクリックします。
8. 接続が作成されたら、「**終了**」をクリックします。成功メッセージが表示されたら、「**OK**」をクリックします。

図 2-18 WSRP プロデューサの登録の成功メッセージ



9. これで、ポートレットはページに追加できます。

手順 2: Service Request Status ポートレット（PDK-Java ポートレット）の構築

Service Request アプリケーションでは、サービス・リクエスト ID を指定すると、特定のサービス・リクエストの現行ステータスが表示されるようにします。このような情報を表示するポートレットを構築できます。

PDK-Java 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。

PDK-Java ポートレットを構築するには、次の作業を行います。

- PDK-Java ポートレットおよびプロデューサの作成
- PDK-Java ポートレットへのポートレット・ロジックの追加
- アプリケーション・サーバーへの PDK-Java ポートレットのデプロイ

PDK-Java ポートレットおよびプロデューサの作成

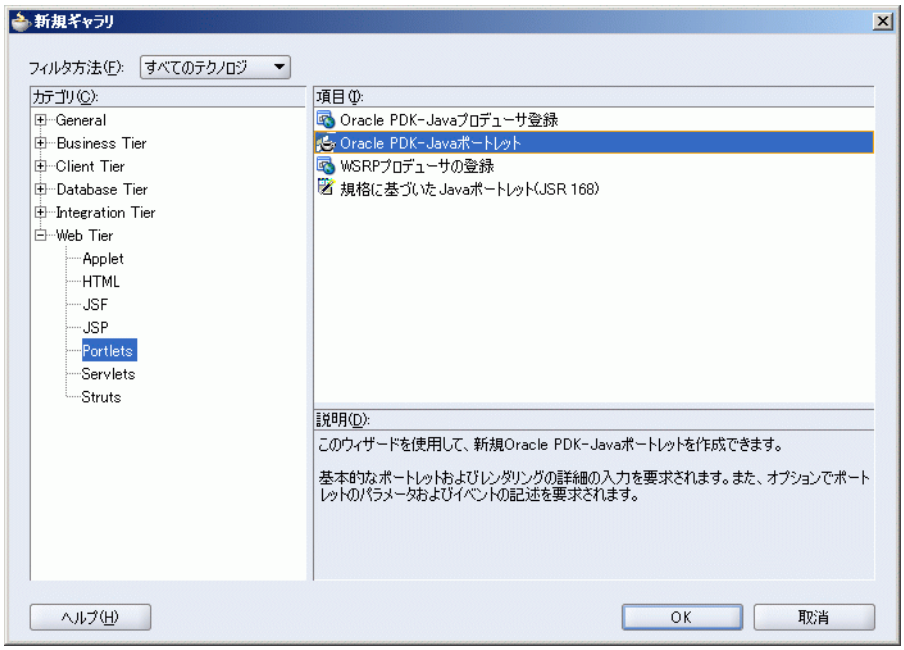
Oracle WebCenter Framework には、Java ポートレット・ウィザードが用意されているため、ポートレットのモードごとに単純な実装を迅速に作成できます。このウィザードを使用して、ポートレットの作成を開始します。

1. Oracle JDeveloper のアプリケーション・ナビゲータで、**「SRDemoSample_Starter」**アプリケーションを開きます。
2. **「SRStatusPortlet」**プロジェクトを右クリックし、**「新規」**を選択します。

   ヒント： このプロジェクトが存在しない場合は、この章の「ポートレット用のプロジェクトの作成」の項で説明した指示に従います。

3. 新規ギャラリで、**「Web Tier」**カテゴリを開き、**「Portlets」**を選択します。
4. 「項目」リストで、**「Oracle PDK-Java ポートレット」**を選択します（図 2-19 を参照）。

図 2-19 新規 PDK-Java ポートレットの作成



5. **「OK」**をクリックして Oracle PDK-Java ポートレット・ウィザードを表示します。

6. ウィザードの「ようこそ」ページが表示されたら、**「次へ」**をクリックして「Web アプリケーション」ページを表示します。
7. **「次へ」**をクリックして「ポートレットの説明」ページを表示します。
8. **「ポートレット名」**フィールドに、`SRStatusPortlet` と入力します。

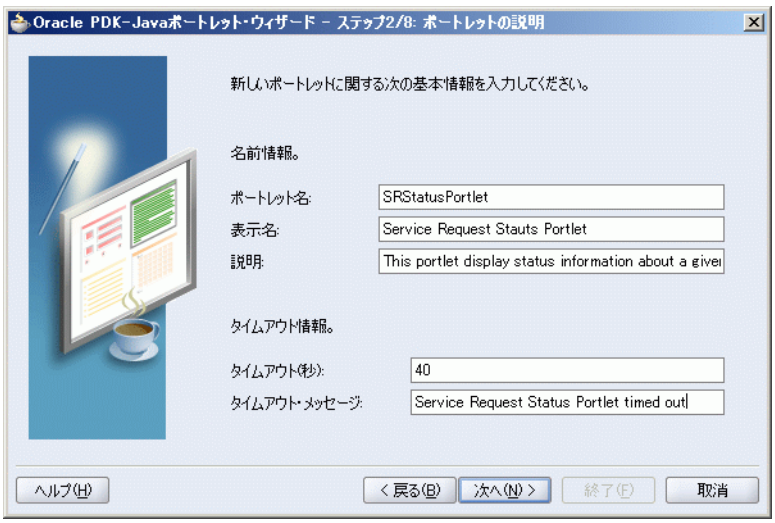
   これは内部名にすぎず、エンド・ユーザーには公開されません。
9. **「表示名」**フィールドに、`Service Request Status Portlet` と入力します。

   これは、ページに追加するポートレットを選択する Oracle JDeveloper のコンポーネント・パレットなどのポートレット選択リストに表示される名前です。
10. **「説明」**フィールドに、次のように入力します。

    ```
    This portlet displays status information about a given service
    request.
    ```
11. **「タイムアウト（秒）」**フィールドは、デフォルト値の 40 のままにしておきます。
12. **「タイムアウト・メッセージ」**フィールドに、次のように入力します。

    ```
    Service Request Status Portlet timed out
    ```

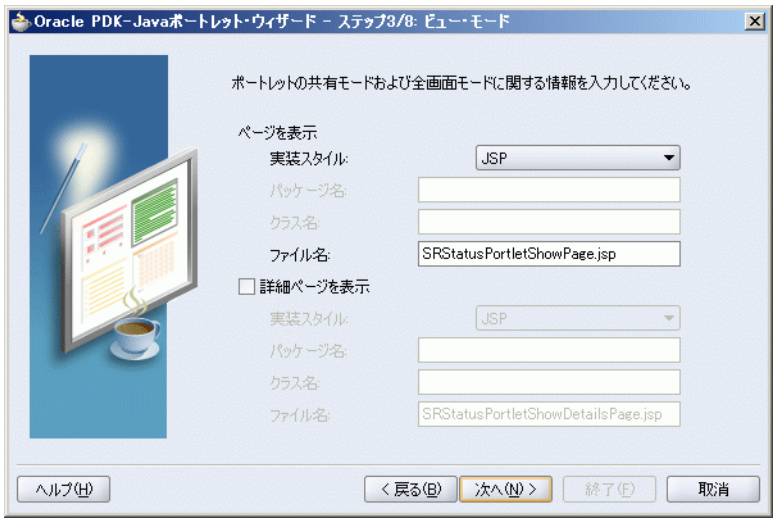
図 2-20「ポートレットの説明」ページ



13. **「次へ」**をクリックして「ビュー・モード」ページを表示します。
14. **「ページを表示」**チェック・ボックスがデフォルトで選択されています。表示モードには、次のように指定します。
    a. 「実装スタイル」リストでは、**「JSP」**が選択されていることを確認します。
    b. 「ファイル名」フィールドは、デフォルト値の `SRStatusPortletShowPage.jsp` のままにしておきます。

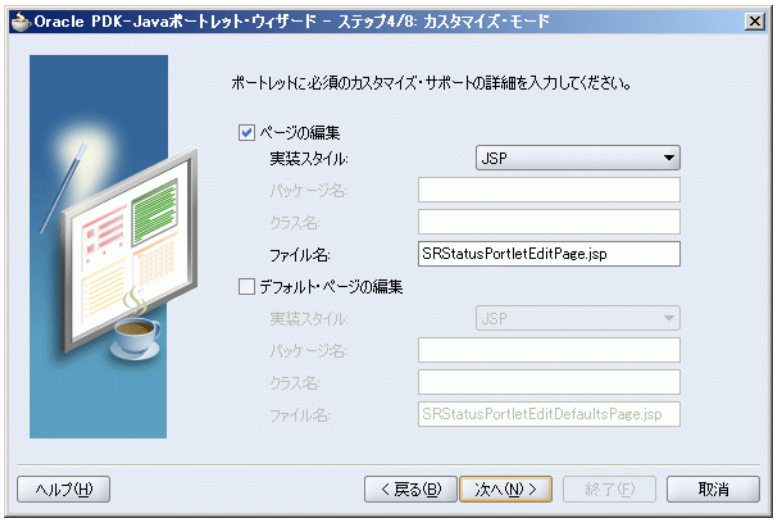
15. このポートレットには詳細表示モードを指定しないため、**「詳細ページを表示」**チェック・ボックスの選択は解除したままにしておきます。

図 2-21「ビュー・モード」ページ



16. **「次へ」**をクリックして「カスタマイズ・モード」ページを表示します。
17. 「ページの編集」チェック・ボックスがデフォルトで選択されています。編集モードには、次のように指定します。
    a. 「実装スタイル」リストでは、**「JSP」**が選択されていることを確認します。
    b. **「ファイル名」**フィールドは、デフォルト値の `SRStatusPortletEditPage.jsp` のままにしておきます。
18. このポートレットにはデフォルト編集モードを指定しないため、「デフォルト・ページの編集」チェック・ボックスの選択は解除したままにしておきます。

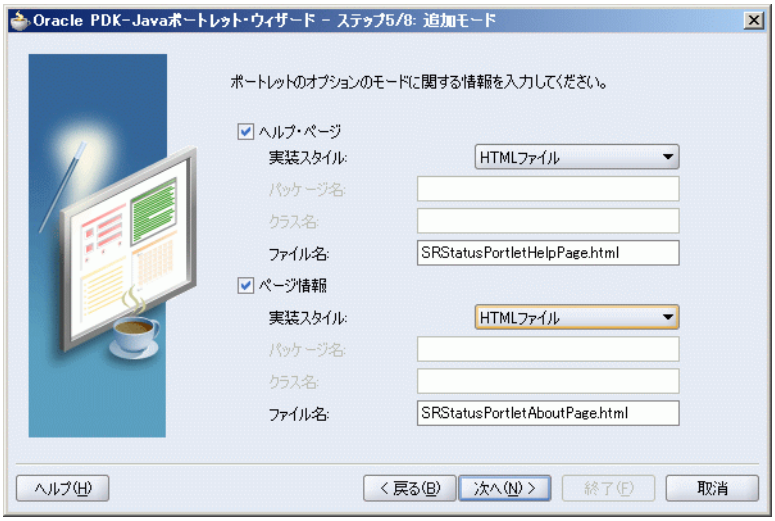
図 2-22「カスタマイズ・モード」ページ



19. **「次へ」**をクリックして「追加モード」ページを表示します。

20. **「ヘルプ・ページ」**チェック・ボックスを選択します。ヘルプ・モードには、次のように指定します。
    a. **「実装スタイル」**リストから、**「HTML ファイル」**を選択します。
    b. **「ファイル名」**フィールドは、デフォルト値の `SRStatusPortletHelpPage.html` のままにしておきます。
21. **「ページ情報」**チェック・ボックスを選択します。情報モードには、次のように指定します。
    a. 「実装スタイル」リストから、**「HTML ファイル」**を選択します。
    b. **「ファイル名」**フィールドは、デフォルト値の `SRStatusPortletAboutPage.html` のままにしておきます。

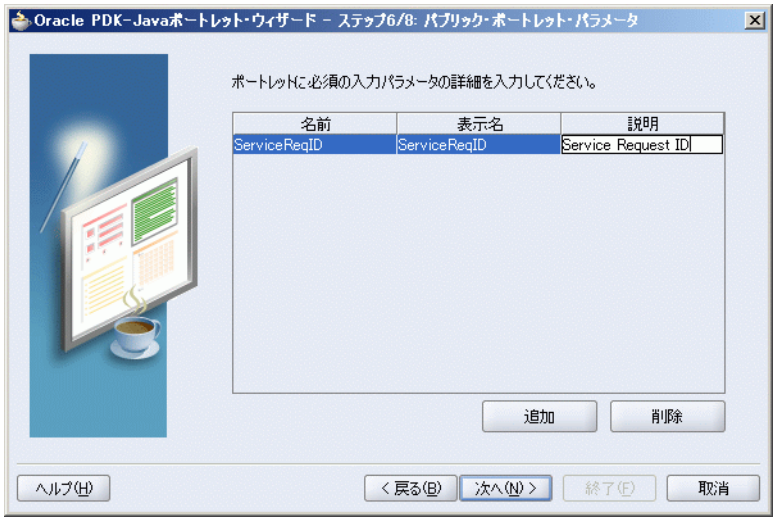
図 2-23「追加モード」ページ



22. **「次へ」**をクリックして「パブリック・ポートレット・パラメータ」ページを表示します。
23. 「パブリック・ポートレット・パラメータ」ページで、**「追加」**をクリックします。
24. 新規パラメータの名前を `ServiceReqID` に変更します。
25. 新規パラメータの表示名を `ServiceReqID` に変更します。

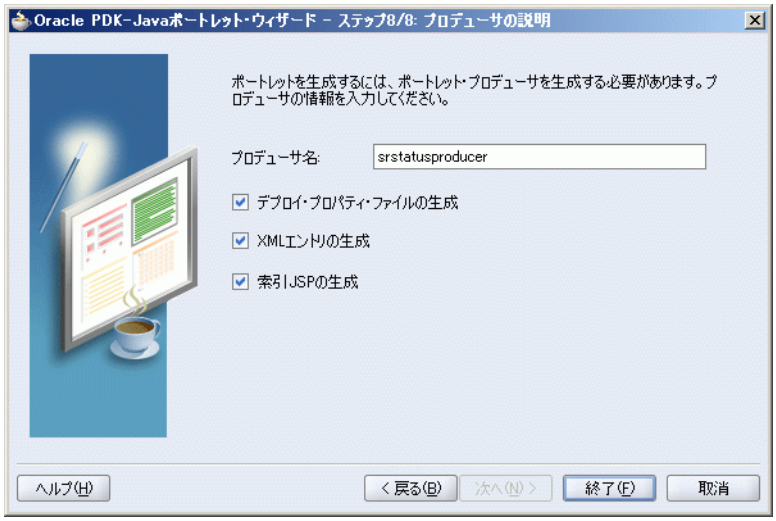
26. 新規パラメータの説明を `Service Request ID` に変更します。

図 2-24 パブリック・ポートレット・パラメータ



27. **「次へ」**をクリックして「パブリック・ポートレット・イベント」ページを表示します。
28. **「次へ」**をクリックして「プロデューサの説明」ページを表示します。
29. **「プロデューサ名」**フィールドに、`srstatusproducer` と入力します。
30. すべてのチェック・ボックス、すなわち**「デプロイ・プロパティ・ファイルの生成」**、**「XML エントリの生成」**、**「索引 JSP の生成」**が選択されていることを確認します。

図 2-25「プロデューサの説明」ページ

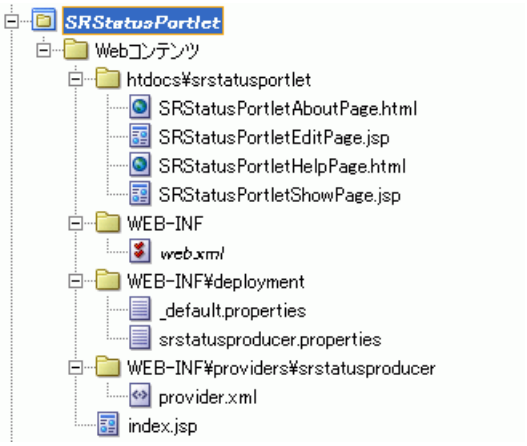


31. **「終了」**をクリックしてポートレット用のファイルを生成します。アプリケーション・ナビゲータで、「SRStatusPortlet」プロジェクトの下のすべてのノードを開くと、次のファイルが表示されます。

    - 選択したポートレット・モードごとのファイル
      - `SRStatusPortletAboutPage.html`
      - `SRStatusPortletEditPage.jsp`
      - `SRStatusPortletHelpPage.html`
      - `SRStatusPortletShowPage.jsp`
    - `web.xml`
    - `_default.properties`
    - `srstatusproducer.properties`
    - `provider.xml`
    - `index.jsp`

    これらのファイルはすべてポートレットを正常にデプロイおよび実行するのに必要です。ただし、テストのために Oracle JDeveloper で使用される `index.jsp` は除きます。

図 2-26 Service Request Status ポートレット用に生成されたファイル



32. 「ファイル」メニューから、**「すべて保存」**を選択します。

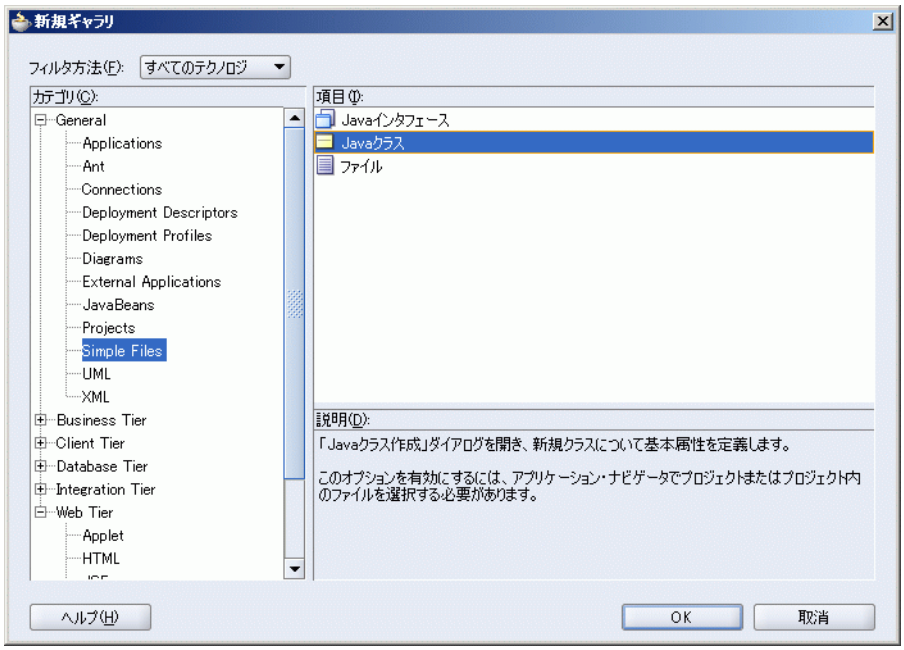
PDK-Java ポートレットへのポートレット・ロジックの追加

この時点で、ポートレットはほとんど機能しません。実際には、ようこそメッセージを表示し、エンド・ユーザーがポートレット・タイトルをパーソナライズできるようにしているだけです。ポートレットが必要な特定の機能を実行するようにするには、必要な機能を実装するビジネス・ロジックでサンプル・コードを拡張する必要があります。

Service Request ポートレットは、サービス・リクエスト ID が指定されると、SRDemo データベースに問い合せて特定のサービス・リクエストのステータスを検出し、その情報を表に表示する必要があります。

1. アプリケーション・ナビゲータで、**「SRStatusPortlet」**プロジェクトを右クリックし、**「新規」**を選択します。
2. 新規ギャラリで、**「General」**カテゴリを開き、**「Simple Files」**を選択します。
3. 「項目」リストで、**「Java クラス」**を選択します。

図 2-27 Java クラスの作成



4. **「OK」**をクリックして「Java クラスの作成」ダイアログ・ボックスを表示します。
5. **「名前」**フィールドに、`SRConnectionUtil` と入力します。

6. このダイアログ・ボックスの残りのオプションは、デフォルト値のままにしておきます。

図 2-28「Java クラスの作成」ダイアログ・ボックス

7. 「**OK**」をクリックします。
8. 生成されたコードを例 2-2 に示すコードで置き換えます。

例 2-2 SRDemo データベースへの接続を作成するコード

```
package srstatusportlet;

import java.sql.Connection;
import java.sql.SQLException;

import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.sql.DataSource;


public class SRConnectionUtil {
    public SRConnectionUtil() {
    }

    public static Connection getConnection ()
    {
      InitialContext ctx;
      DataSource ds;
      Connection conn = null;
      try
      {
        ctx = new InitialContext();
        ds = (DataSource) ctx.lookup("jdbc/SRDemoCoreDS");
        conn = ds.getConnection();
      }
      catch (NamingException ne)
      {
        conn = null;
      }
      catch (SQLException sqle)
      {
        conn = null;
      };
      return conn;
```

```
    }

}
```

このクラスは、ポートレット・コードでの問合せ実行対象となる SRDemo データベースへの接続を作成します。

9. ファイルを保存します。
10. まだ作成していない場合は、ダウンロードしたサンプル・ファイルに含まれる `install.html` ファイルの説明に従って、srdemo スキーマを指すデータベース接続を作成します。
11. アプリケーション・ナビゲータで、**「SRStatusPortletShowPage.jsp」**を右クリックし、**「開く」**を選択します。
12. 「ソース」タブを選択してページのソース・コードを表示します。
13. 既存のコードを選択し、例 2-3 に示すコードで置き換えます。

例 2-3 Service Request Status ポートレット用のコード

```
<%@page contentType="text/html; charset=windows-1252"
        import="oracle.portal.provider.v2.render.PortletRenderRequest"
        import="oracle.portal.provider.v2.http.HttpCommonConstants"
        import="oracle.portal.provider.v2.ParameterDefinition"
        import="srstatusportlet.SRConnectionUtil"
        import="java.sql.Connection"
        import="java.sql.Statement"
        import="java.sql.ResultSet"
        import="java.sql.SQLException"
%>

<%
   PortletRenderRequest pReq = (PortletRenderRequest)
      request.getAttribute(HttpCommonConstants.PORTLET_RENDER_REQUEST);

   // Reading the service request ID, as a parameter
   String SRID = pReq.getParameter("ServiceReqID");

   // Creating the database connection
   Connection conn = SRConnectionUtil.getConnection();

   // Testing if the parameter has been wired. If no parameter is passed
   // it is set to 104, by default.
   if (SRID == null)
   {
     %>
     Please map the service request ID to this portlet's input parameter.
     <%
     SRID = "104";
   }

   if (conn == null) {
    %>
    Couldn't connect to the database.
    <%
   }
   else
     try
     {
```

```
        Statement stmt = conn.createStatement();
        // Constructing the SELECT statement
        String query =
            "SELECT svr_id, status, " +
            "users1.first_name || ' ' || users1.last_name createdby, " +
            "users2.first_name || ' ' || users2.last_name assignedto, " +
            "TO_CHAR(assigned_date, 'Dy, Mon DD, YYYY') " +
            "FROM service_requests, users users1, users users2 " +
            "WHERE svr_id = " + SRID + " AND " +
            "users1.user_id = service_requests.created_by AND " +
            "users2.user_id = service_requests.assigned_to";

        // Executing the query.
        ResultSet rs = stmt.executeQuery(query);
        // Stepping through the result set.
        while (rs.next())
        {
%>
<table>
 <tr>
  <td>
   <font class="PortletText1">
     Service Request ID:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getInt(1)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Status:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(2)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Created on:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(5)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Created By:
   </font>
  </td>
```

```
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(3)%>
   </font>
  </td>
 </tr>
 <tr>
  <td>
   <font class="PortletText1">
     Assigned To:
   </font>
  </td>
  <td>
   <font class="PortletHeading1">
     <%=rs.getString(4)%>
   </font>
  </td>
 </tr>
</table>

<%
        } //while
      } // try
      catch (SQLException sqle)
      {
        System.out.println ("DB Connection established successfully but ran into an
issue while working with the DB.");
        System.out.println (sqle);
      }
      conn.close();
%>
```

このコードは、まず、先ほど作成した Java クラスをコールしてデータベースに接続します。次に、サービス・リクエスト ID を渡されたパラメータに基づいて設定します（パラメータが渡されない場合、サービス・リクエスト ID はデフォルト値の 100 に設定されます）。このサービス・リクエスト ID を使用して SRDemo データベースに問い合せ、サービス・リクエストのステータスを検出します。最後に、この問合せ結果が HTML 表に出力されます。

14. ファイルを保存します。
15. 「SRStatusPortletAboutPage.html」を右クリックし、**「開く」**を選択します。
16. 「設計」ビューで、次のテキストを入力します。

    ```
    Service Request Status Portlet

    MyAcme, Copyright
    ```

17. ファイルを保存します。
18. **「SRStatusPortletHelpPage.html」**を右クリックし、**「開く」**を選択します。
19. 「設計」ビューで、次のテキストを入力します。

    ```
    This portlet displays status information about a given service
    request, based on a parameter (service request ID) it receives. If
    the portlet does not work as expected, it may not have been wired
    properly.
    ```

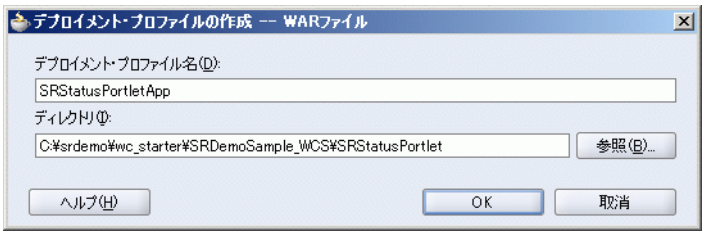
20. ファイルを保存します。

アプリケーション・サーバーへの PDK-Java ポートレットのデプロイ

ポートレットの構築が終了したので、Preconfigured OC4J にデプロイする準備ができました。

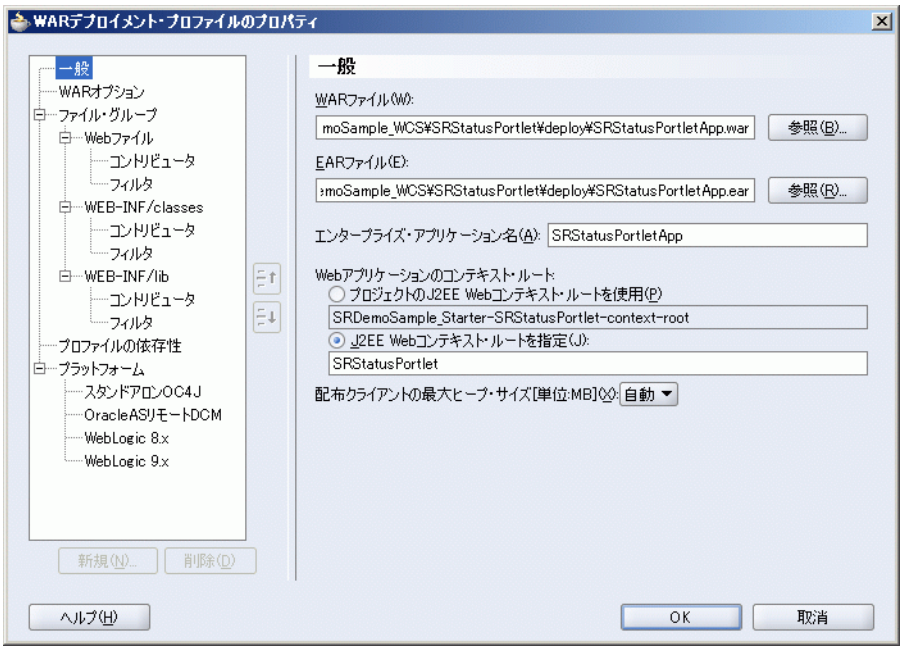
1. アプリケーション・ナビゲータで、**「SRStatusPortlet」→「Web コンテンツ」→「WEB-INF」**の下の **web.xml** ファイルを探します。
2. **「web.xml」**を右クリックし、**「WAR デプロイメント・プロファイルの作成」**を選択します。
3. **「デプロイメント・プロファイル名」**フィールドに、`SRStatusPortletApp` と入力します。
4. デフォルト・ディレクトリを使用します。

図 2-29「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



5. **「OK」**をクリックして「WAR デプロイメント・プロファイルのプロパティ」ダイアログ・ボックスを表示します。
6. **「J2EE Web コンテキスト・ルートを指定」**を選択し、対応するフィールドに `SRStatusPortlet` と入力します。

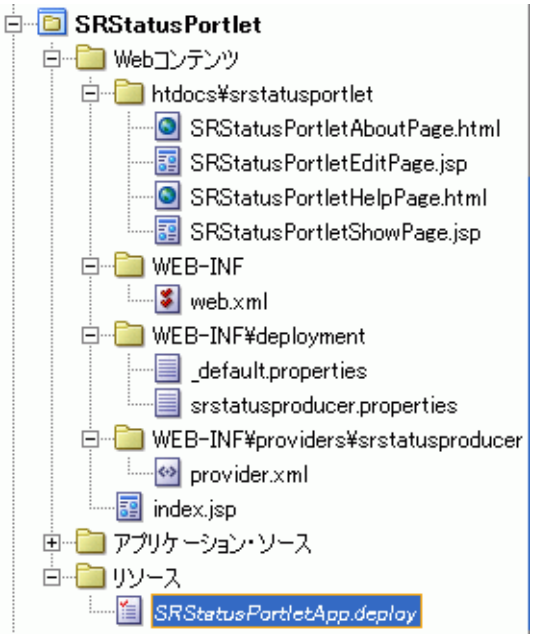
図 2-30「WAR デプロイメント・プロファイルのプロパティ」ダイアログ・ボックス



7. **「OK」**をクリックします。

8. アプリケーション・ナビゲータで、**「リソース」**ノードを開き、デプロイメント・ファイルを確認します。

図 2-31 SRStatusPortlet のデプロイメント・ファイル



9. 「ファイル」メニューから、**「すべて保存」**を選択します。

10. **「SRStatusPortletApp.deploy」**を右クリックし、「配布先」メニューから**「SRDemoConnection」**を選択します。

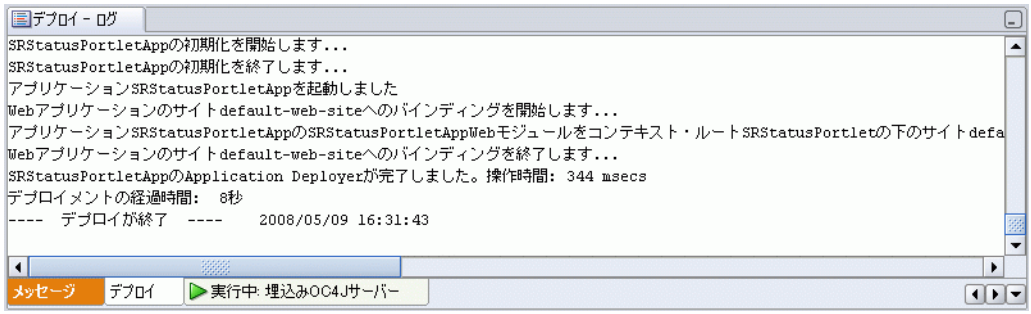
    「アプリケーションの構成」ダイアログ・ボックスが表示されます。

> **注意：** メニューに「SRDemoConnection」が表示されない場合は、この章の「Oracle WebCenter Preconfigured OC4J へのアプリケーション・サーバー接続の作成」の項で示した手順に従います。

11. ここでは、デフォルト値を受け入れることができるため、**「OK」**をクリックします。

12. 「デプロイ - ログ」に「---- デプロイが終了 ----」メッセージが表示されたら、エラーが発生していないことを確認します。

図 2-32 デプロイ - ログ



13. JSR 168 ポートレットの「プロデューサの登録」で使用したのと同じ手順に従って、次の URL を使用してプロデューサを登録します。

    ```
    http://localhost:6688/SRStatusPortlet/providers
    ```

    これで、作成した JSR 168 ポートレットと PDK-Java ポートレットは、アプリケーションに追加できます。

まとめ

この章では、JSR 168 ポートレットと PDK-Java ポートレットを構築する方法および WebCenter アプリケーションで使用するためにこれらのポートレットをデプロイする方法について学びました。この時点で、プロデューサをアプリケーションに登録し、ポートレットをページに追加できます。第 7 章「ページの構築とコンポーネントの追加」の作業を実行するだけです。しかし、ページの構築およびコンポーネントの追加に進む前に、第 3 章「スキンの設定」の作業を続けることができます。

3

スキンの設定

この章では、デモ・アプリケーションの一部としてダウンロードしたスキンを使用するようにアプリケーションを構成し、アプリケーションの実行時にスキンを選択できるようにします。

この章の作業を実行するには、`SRDemo_App_Download.ZIP` に含まれる `install.html` ドキュメントの手順を実行して、`skins.zip` ファイルから適切なフォルダにスキンを解凍しておく必要があります。この ZIP ファイルには、`myCompany` と `limerine` の 2 つのスキンと、対応する必要なイメージ・ファイルが含まれています。デフォルトでは、`srdemo` スキンが `SRDemoSample_Starter` アプリケーションの一部として使用できます。

この章の内容は次のとおりです。

- 手順 1: スキンが登録されていることの確認
- 手順 2: アプリケーションが新しいスキンを使用するように構成されていることの確認

手順 1: スキンが登録されていることの確認

新しいスキンの myCompany と limerine をアプリケーションに適用するには、まず、これらのスキンが WebCenter アプリケーションに登録されていることを確認する必要があります。そのためには、adf-faces-skins.xml ファイルに、これら 3 つのスキンのエントリがあることを確認する必要があります。このファイルには、アプリケーションで使用できるすべてのスキンに関する情報が格納されます。

スキンの新規作成方法など、スキンの詳細は、『Oracle Application Development Framework 開発者ガイド』を参照してください。

3 つの新しいスキンがアプリケーションに登録されていることを確認するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、**「UserInterface」**プロジェクトを開きます。
2. **「Web コンテンツ」**フォルダ、**「WEB-INF」**フォルダの順に開きます。
3. **「adf-faces-skins.xml」**を右クリックし、**「開く」**を選択します。
4. 例 3-1 で太字テキストになっているコードがファイル内に存在することを確認します。

例 3-1 adf-faces-skins.xml のスキンのエントリ

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<skins xmlns="http://xmlns.oracle.com/adf/view/faces/skin">

    <skin>
        <id>srdemo.desktop</id>
        <family>srdemo</family>
        <render-kit-id>oracle.adf.desktop</render-kit-id>
        <style-sheet-name>skins/srdemo/srdemo.css</style-sheet-name>
    </skin>
    <skin>
        <id>mycompany.desktop</id>
        <family>mycompany</family>
        <render-kit-id>oracle.adf.desktop</render-kit-id>
        <style-sheet-name>skins/mycompany/myCompanySkin.css</style-sheet-name>
    </skin>
    <skin>
        <id>limerine.desktop</id>
        <family>limerine</family>
        <render-kit-id>oracle.adf.desktop</render-kit-id>
        <style-sheet-name>skins/limerine/limerine.css</style-sheet-name>
    </skin>
</skins>
```

5. ファイルを閉じます。

手順 2: アプリケーションが新しいスキンを使用するように構成されていることの確認

別のスキンを使用するには、adf-faces-config.xml ファイルの <skin-family> タグを、目的のスキンのファミリ名で更新しておく必要があります。adf-faces-config.xml ファイルには、実行時に使用されるスキンに関する情報が格納されます。ほとんどの場合、スキン・ファミリ名は構成ファイルにハードコーディングされます。

しかし、このアプリケーションでは、管理者が実行時にスキンを選択できるようにします。そのためには、ハードコーディングするかわりに、adf-faces-config.xml ファイルのスキン・ファミリ名に式言語（EL）を使用する必要があります。この項では、それを確認する手順を説明します。

adf-faces-config.xml ファイルでスキン・ファミリ名に EL が使用されていることを確認するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、*<SRDemoSample>* アプリケーションを開きます。
2. **「UserInterface」**プロジェクトを開きます。
3. **「Web コンテンツ」**ノード、**「WEB-INF」**ノードの順に開きます。
4. **「adf-faces-config.xml」**を右クリックし、**「開く」**を選択します。
5. 「構造」ペインで、**「skin-family」**を選択します。
6. 「プロパティ・インスペクタ」で、skin-family が #{skinBean.currentSkin} に設定されていることを確認します。

まとめ

この章では、新しいスキンをアプリケーションに登録する方法と、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する方法について学びました。この時点で、ユーザーに、実行時にスキンを変更するためのオプションを提示できます。この種のカスタマイズを可能にする手順は、第 9 章「サイト管理ページの構築」を参照してください。

4

コンテンツ・リポジトリの設定

この章では、ローカル・ドライブにあるコンテンツ・ディレクトリからコンテンツ・リポジトリを設定する方法について学びます。後で、他の各章の手順を実行する際に、このリポジトリのコンテンツを構築した JSPX ページに統合します。

コンテンツ・リポジトリを設定するには、ファイル・システムの Java Content Repository（JCR）データ・コントロールとしてローカル・ディレクトリを構成します。このデータ・コントロールは、`search`、`advancedSearch`、`getURI` および `getItems` の各メソッドを提供します。`getURI` メソッドと `getItems` メソッドは、リンク、表および階層ツリーとしてコンテンツを追加できるようにします。`search` メソッドと `advancedSearch` メソッドは、追加したコンテンツに検索および拡張検索の機能を提供できるようにします。

この章では、次の手順を実行します。

- 手順 1: サンプル・コンテンツ用のコンテンツ・ディレクトリの設定
- 手順 2: コンテンツ・プロジェクトの作成
- 手順 3: JCR データ・コントロールの構成

手順 1: サンプル・コンテンツ用のコンテンツ・ディレクトリの設定

ファイル・システムのデータ・コントロールを構成する前に、ダウンロードしたサンプル・ファイルに含まれる `install.html` ファイルのサンプル・コンテンツの設定に関する項の手順を実行して、コンテンツ・ディレクトリをローカル・ドライブ上に設定します。

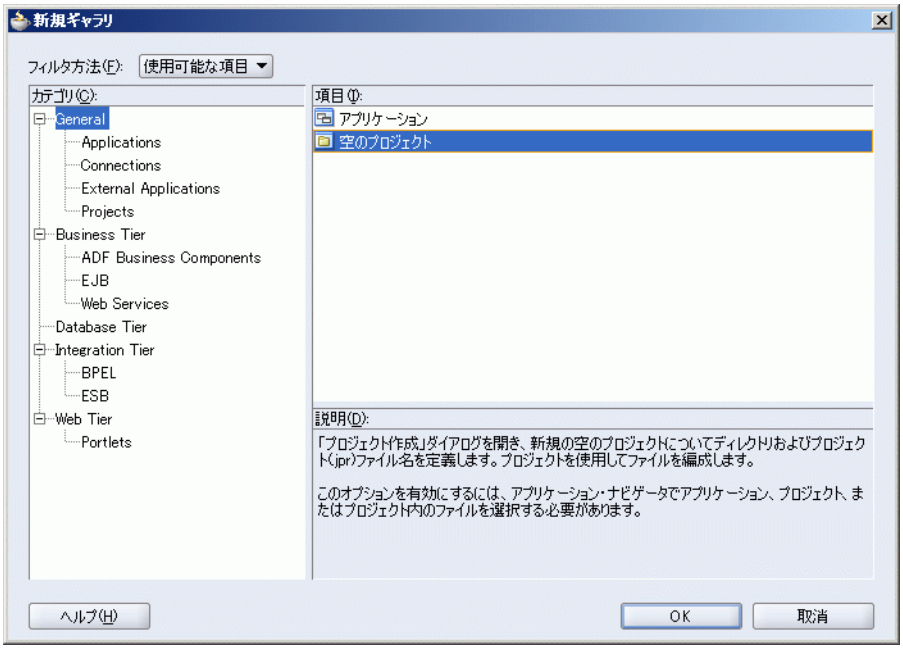
手順 2: コンテンツ・プロジェクトの作成

この項では、次の手順でデータ・コントロールを構成するプロジェクトを作成します。

プロジェクトを作成するには、次の手順を実行します。

1. Oracle JDeveloper で、アプリケーション・ナビゲータに移動します。「アプリケーション」の下で、**「SRDemoSample_Starter」**を右クリックし、**「新規」**を選択します。「新規ギャラリ」ダイアログ・ボックスが表示されます。
2. 「項目」で、**「空のプロジェクト」**を選択し（図 4-1 を参照）、**「OK」**をクリックします。「プロジェクトの作成」ダイアログ・ボックスが表示されます。

図 4-1「新規ギャラリ」ダイアログ・ボックス - 「空のプロジェクト」



3. 「プロジェクト名」フィールドに **Content** と入力し（図 4-2 を参照）、「**OK**」をクリックします。Content プロジェクトがアプリケーション・ナビゲータに表示されます（図 4-3 を参照）。

図 4-2 新規プロジェクト

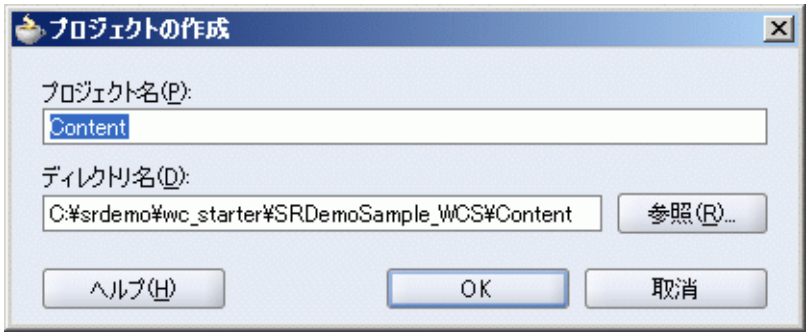
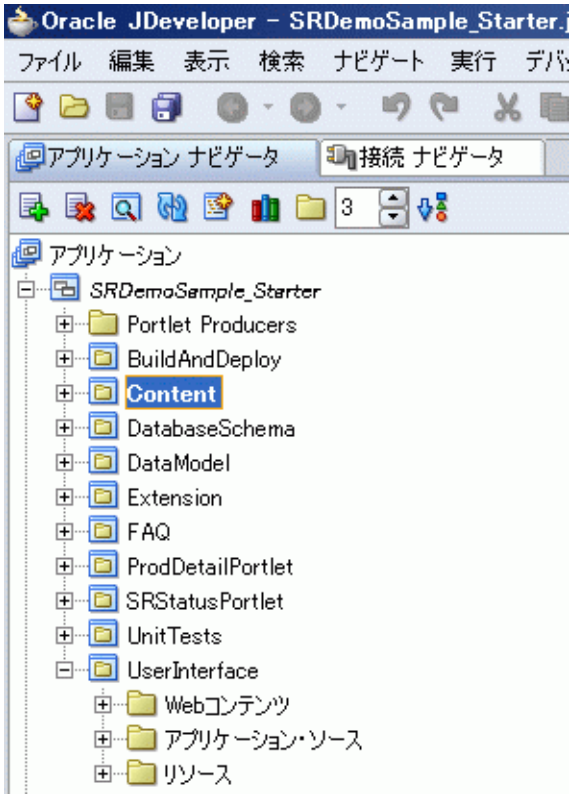


図 4-3 アプリケーション・ナビゲータ - Content プロジェクト



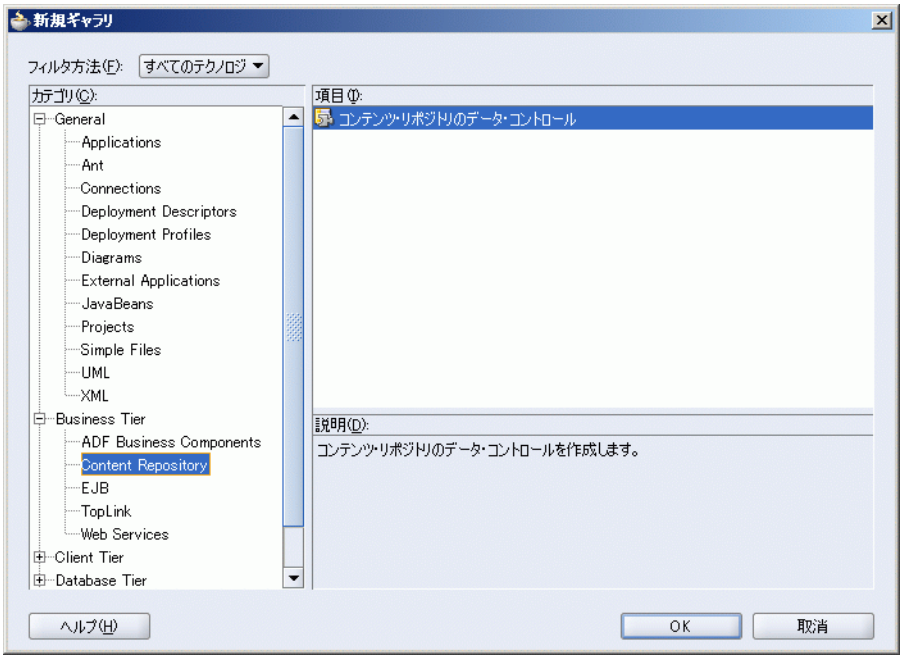
手順 3: JCR データ・コントロールの構成

この項では、ファイル・システムの JCR データ・コントロールを構成し、ローカル・ドライブにあるコンテンツをアクセスおよび公開できるようにします。

ファイル・システムのデータ・コントロールを構成するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、**「Content」**を右クリックし、**「新規」**を選択します。「新規ギャラリ」ダイアログ・ボックスが表示されます。
2. 「カテゴリ」で、**「Business Tier」**ノードを開き、**「Content Repository」**を選択します。次に、「項目」で**「コンテンツ・リポジトリのデータ・コントロール」**を選択し（図 4-4 を参照）、**「OK」**をクリックします。「データ・コントロールの作成」ダイアログ・ボックスが表示されます。

図 4-4「新規ギャラリ」ダイアログ・ボックス - 「コンテンツ・リポジトリのデータ・コントロール」



3. 「データ・コントロールの作成」ダイアログ・ボックスで、**「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. ステップ 1 で、データ・コントロールの名前として `SRContentRepository` と入力し、**「次へ」**をクリックします。
5. ステップ 2 で、**「リポジトリ・タイプ」**ボックスから**「ファイル・システム」**を選択します。
6. **「ベース・パス」**フィールドに、コンテンツが存在するフォルダへのパス、すなわち `C:¥srdemo¥SRContentRepository` を入力します。

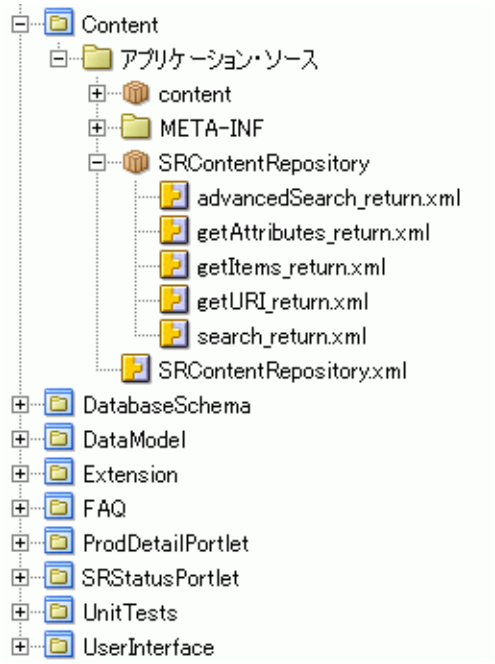
7. 「**テスト**」ボタンをクリックして接続を確認します。図 4-5 に示すような「成功」メッセージが表示されます。

図 4-5 SRContentRepository への接続のテスト

8. 「**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。

   アプリケーション・ナビゲータには、「Content」→「アプリケーション・ソース」の下に新しいエントリが表示されます（図 4-6 を参照）。

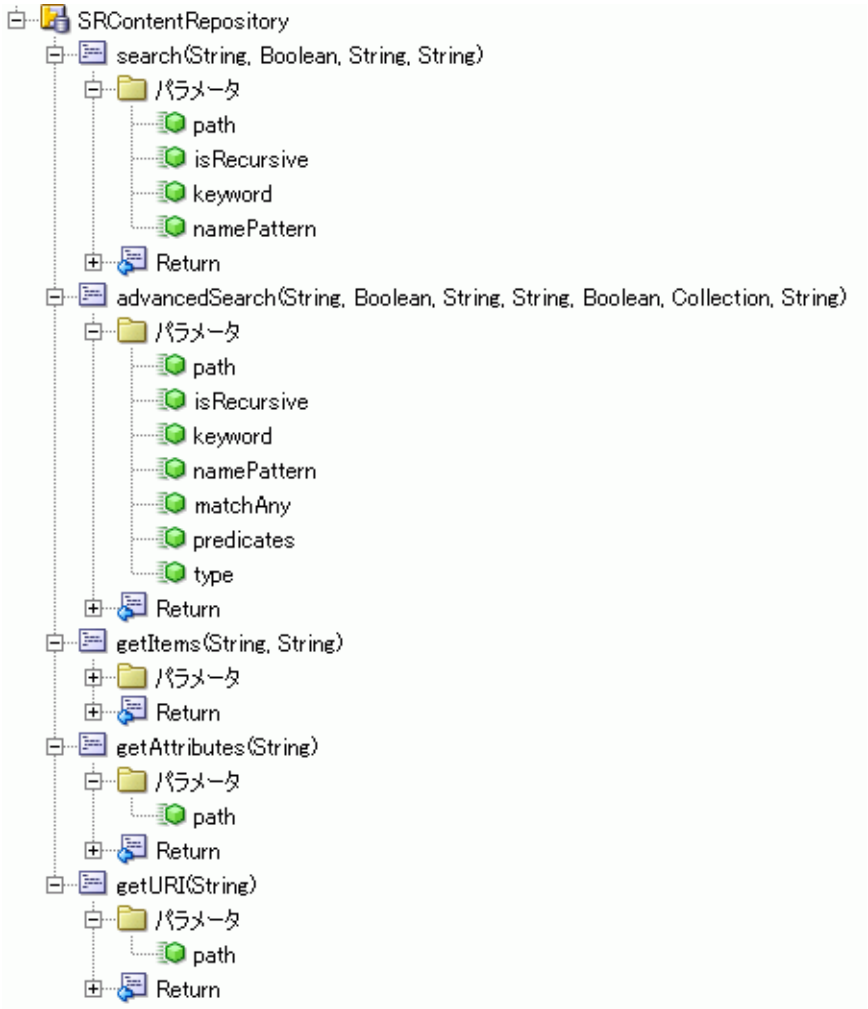
図 4-6 新しいデータ・コントロール用のファイル



9. SRContentRepository データ・コントロールのメソッドを表示するには、「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。

   SRContentRepository の下には、新しいデータ・コントロール用のメソッド、パラメータおよび演算子の階層リストが表示されます（図 4-7 を参照）。

図 4-7 データ・コントロール・パレット - SRContentRepository



まとめ

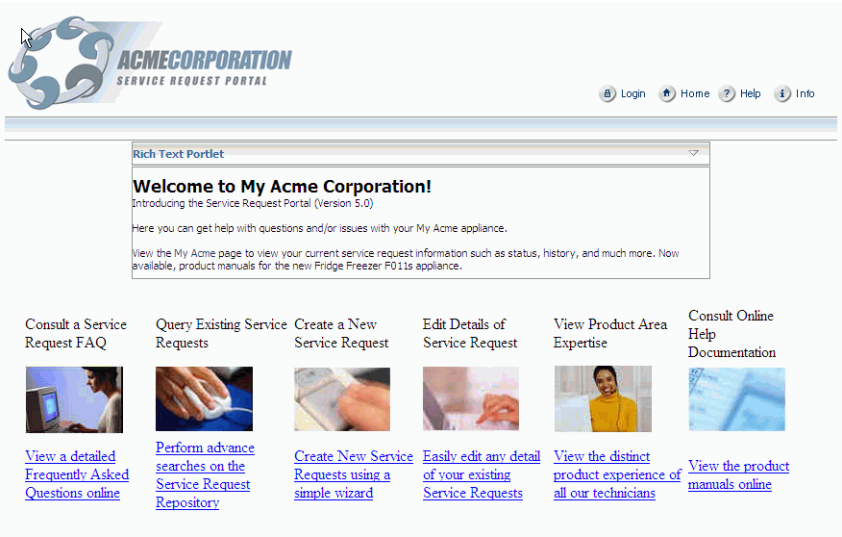
この章では、ローカル・ドライブにあるコンテンツ・ディレクトリからコンテンツ・リポジトリを構成する方法について学びました。このコンテンツ・リポジトリには、第 5 章「パブリックなようこそページの作成」でアクセスします。

5

パブリックなようこそページの作成

この章では、ようこそページを SRDemo アプリケーション内に作成し、ユーザー用にカスタマイズする方法について説明します。エンド・ユーザーにお知らせのテキストを表示するリッチ・テキスト・コンポーネントを追加し、第 4 章「コンテンツ・リポジトリの設定」で設定したサンプルのコンテンツ・リポジトリからファイルを所定の場所に表示します。次に、ようこそページをパブリックにしてユーザー全員が表示できるようにし、「Login」リンクを追加してユーザーがユーザー ID およびパスワードを入力すると専用のアプリケーションに情報を表示できるようにします。作成するようこそページは、図 5-1 のような外観です。

図 5-1 ようこそページの全体像



この章の内容は次のとおりです。

- 手順 1: リッチ・テキスト・コンポーネントの追加
- 手順 2: 所定の場所でのファイルの表示
- 手順 3: グローバル・ナビゲーション・リンクの追加
- 手順 4: ようこそページのパブリック化

手順 1: リッチ・テキスト・コンポーネントの追加

この項の手順では、リッチ・テキスト・コンポーネントをアプリケーション内の既存のようこそページに追加する方法を示します。このリッチ・テキスト・コンポーネントは、パブリック・ユーザーへのお知らせを表示します。会社の管理者は、サイト管理のアクセス権があるため、実行時にこのお知らせを更新できます。

リッチ・テキスト・コンポーネントを追加するには、次のようにします。

1. **SRDemoSample_Starter** アプリケーションから、`SRWelcome.jspx` ページの「設計」ビューを開きます。このページは、**「UserInterface」**、**「Web コンテンツ」**、**「app」**の下にあります。
2. イメージ `acmecenter.jpg` を探します。
3. **「ソース」**タブをクリックしてページのソース・コードを表示します。
4. 次のコードを削除して、ページからイメージを削除します。

```
<af:objectImage height="376"
        width="800"
        source="../images/acmecenter.jpg"/>
```

この時点で、ソース・コードのその部分は、図 5-2 のようになります。

図 5-2 acmecenter.jpg が削除された panelGroup

```
<af:panelGroup layout="vertical">
  <af:panelHorizontal halign="center">

  </af:panelHorizontal>
```

今度は、この場所にリッチ・テキスト・ポートレットを追加します。

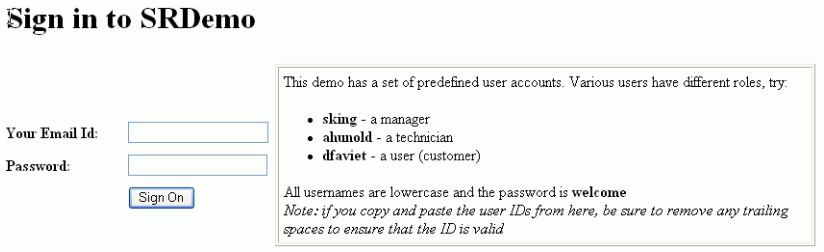
5. リッチ・テキスト・ポートレットを追加するには、まず、そのポートレットを登録します。しかし、リッチ・テキスト・ポートレット・プロデューサ用の登録 URL を検出するために、先に次の手順を実行します。
   a. Preconfigured OC4J を起動します。
   b. readme ページがデフォルトで開かない場合は、**「ヘルプ」**メニューから**「WebCenter Preconfigured OC4J README」**を選択します。
   c. readme ページの「Index Page」セクションで、**「index page」**リンクをクリックします。
   d. WebCenter Preconfigured OC4J の索引ページで、「WSRP Portlet Producers」の下の**「Rich Text Portlet Producer」**をクリックします。
   e. **「WSRP v2 WSDL」**をクリックします。
   f. ブラウザのアドレス・バーから URL を保存します。プロデューサの登録時に、この URL を使用します。
6. リッチ・テキスト・ポートレット・プロデューサを登録するには、**「UserInterface」**プロジェクトを右クリックし、**「新規」**を選択します。
7. 「新規ギャラリ」ダイアログ・ボックスで、「すべてのテクノロジ」でフィルタ処理し、**「Web Tier」**、**「Portlets」**の順に選択し、**「WSRP プロデューサの登録」**を選択します。
8. 「名前」フィールドに `RichTextPortletProducer` と入力し、**「次へ」**をクリックします。
9. 「URL エンドポイント」フィールドに、Preconfigured OC4J での WSRP v2 プロデューサの WSDL 用の URL を入力します。これは、手順 5 で保存した URL です。
10. **「次へ」**をクリックし、**「終了」**をクリックします。

11. 「構造」ペインで、元々イメージが含まれていた「panelHorizontal」コンポーネントをクリックして選択します。
12. コンポーネント・パレットから、**「RichTextPortletProducer」**を選択し、リストから**「Rich Text Portlet」**を選択します。新しいポートレットが、panelHorizontal コンポーネント内に表示されます。
13. **「Rich Text Portlet」**を選択し、プロパティ・インスペクタを開きます。
14. **AllModesSharedScreen** プロパティを true に設定します。
15. **Width** を 600 ピクセルに設定します。
16. **「SRWelcome.jspx」**を右クリックしてページ定義に移動します。
17. 認可を編集し、**Customize** 権限を manager に付与します。

    この作業は、管理者がページ上のリッチ・テキスト・ポートレットのコンテンツをカスタマイズできる必要があるため、実行する必要があります。
18. ページを保存します。
19. アプリケーション・ナビゲータで、**「SRWelcome.jspx」**を右クリックし、**「実行」**を選択します。

    すべての JSP および JSPX ファイルに対する標準的な J2EE セキュリティ制約のため、デフォルトの SRDemo ログイン・ページに自動的にリダイレクトされます。このログイン・ページは、静的 HTML を含む単純な JSP ページです。ahunold、sking および dfaviet の 3 種類のユーザーとしてログインできます（図 5-3 を参照）。

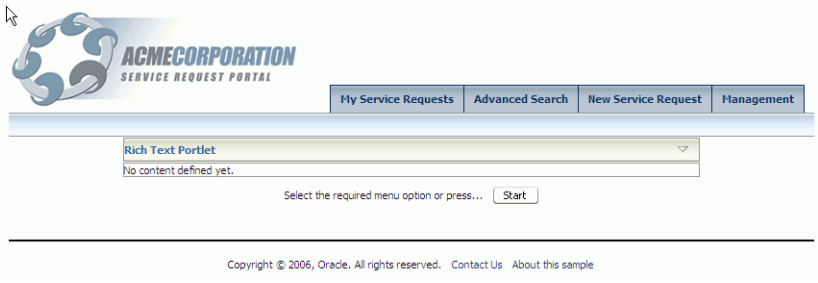
図 5-3 SRDemo ログイン・ページ



sking としてログインすると、次の図の右端に示されている管理タブを含め、すべてのタブが表示されます。

実行時のポートレットは、図 5-4 のような外観です。

図 5-4 管理者としてログインしたときに表示されるタブ



ahunold としてログインすると、管理タブは表示されません。これは、ahunold が管理者ではないためです。

20. ブラウザで、リッチ・テキスト・ポートレット上の矢印をクリックし、**「Customize」**を選択してポートレットのテキストをカスタマイズします。

21. ポートレットをカスタマイズして、次のテキストを組み込みます。

```
Welcome to My Acme Corporation!

Introducing the Service Request Portal (Version 5.0)

Here you can get help with questions and/or issues with your My Acme appliance.

View the My Acme page to view your current service request information such as
status, history, and much more. Now available, product manuals for the new
Fridge Freezer F011s appliance.
```

このテキストを書式設定するには、次のようにします。

注意： テキストの書式設定に関する問題を回避するには、まず、このテキストをメモ帳にコピーし、次にメモ帳からポートレットにテキストをコピーします。

---

a. 第 1 文を選択し、大きなフォントを選択して「太字」アイコンをクリックします。

b. テキストにハイパーリンクを作成するには、**My Acme** テキスト（サンプル・テキストの下線部）を選択し、リッチ・テキスト・エディタの**「ハイパーリンク」**アイコンをクリックします。次のように、選択した URL をフィールドに入力します。

`http://localhost:port/SRDemo/faces/app/SRWelcome.jspx`

手順 2: 所定の場所でのファイルの表示

コンテンツ管理システムを使用してコンテンツを表示する方法について、ユーザーには多くの選択肢があります。一般的な方法はコンテンツをリンクとして表示することですが、開発者は、書式設定などの理由で、テキストまたは HTML ファイルのコンテンツをアプリケーション内に直接表示することが必要な場合もあります。JCR を使用すると、ファイルを所定の場所に表示でき、コンテンツ管理システム内からもコンテンツを管理できます。

この項の手順では、コンテンツ・リポジトリのファイルを既存のページ上に表示する方法を示します。これにより、ページをレンダリングするアプリケーションを変更せずにページを編集できます。

ファイルを所定の場所に表示するには、次のようにします。

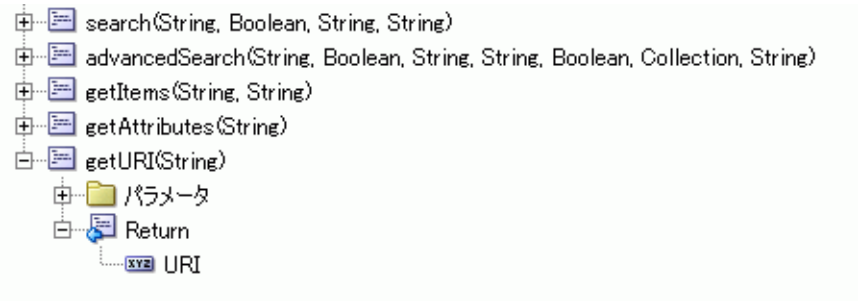
1. 既存の `SRWelcome.jspx` ページをまだ開いていない場合は、**「SRDemoStarter」**、**「UserInterface」**、**「Web コンテンツ」**、**「app」**から開きます。
2. 「ADF Faces Core」ライブラリから、「`panelHorizontal`」を選択して `panelPage` 上の、リッチ・テキスト・ポートレットが含まれる `panelHorizontal` の下にドラッグします。
3. panelHorizontal の **halign** を `center` に設定します。
4. データ・コントロール・パレットで、**「SRContentRepository」**を開きます。このデータ・コントロールは、第 4 章の「手順 3: JCR データ・コントロールの構成」ですでに作成しています。

ヒント： データ・コントロール・パレットが表示されない場合は、「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。

---

5. 「getURI(String)」の下の**「Return」**ノードを開き、**「URI」**（図 5-5）を panelHorizontal にドラッグします。

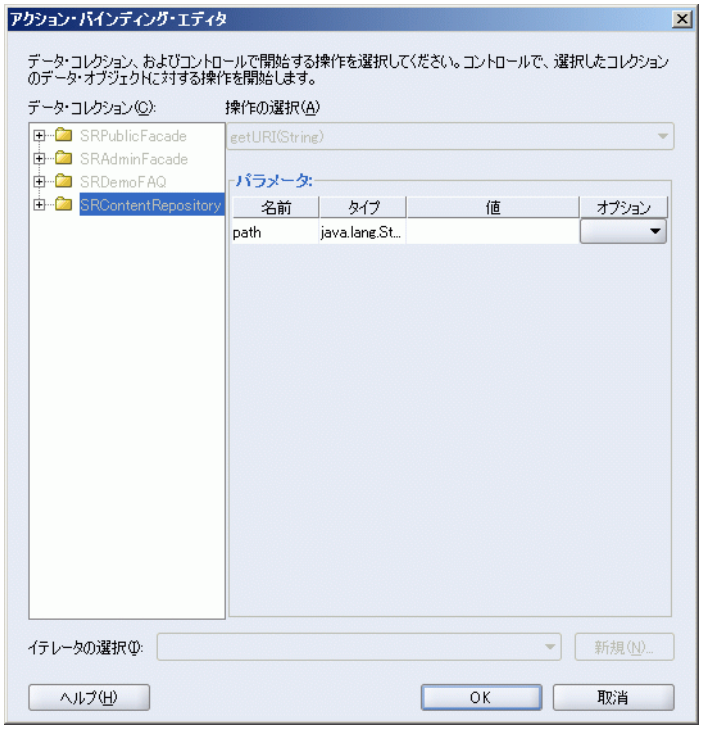
図 5-5 データ・コントロール・パレットでの getURI



6. 「作成」メニューから**「テキスト」**、**「ADF 出力テキスト」**の順に選択します。

   アクション・バインディング・エディタが表示されます（図 5-6 を参照）。

図 5-6 アクション・バインディング・エディタ



7. path パラメータの「値」フィールドに、/welcome.html と入力します。
8. **「OK」**をクリックします。
9. ページのソースを表示し、例 5-1 に示すコードを OutputText の下に追加します。

例 5-1 getURI のサンプル・コード

```
<f:verbatim>
   <iframe height="450" width="850"
src="${pageContext.request.contextPath}${bindings['getURI_returnURI'].inputValue}"
frameborder="0">
</iframe>
</f:verbatim>
```

10. ファイルを保存します。
11. SRWelcome.jspx を実行します。

パブリック・ユーザーによる getURI メソッドの実行の可能化

ようこそページはパブリック・ページであるため、パブリック・ユーザーが getURI メソッド（コンテンツ・リポジトリのファイルを表示するために使用）を実行できるように定義する必要があります。

アクセス権限を変更するには、次のようにします。

1. ページ定義を開きます。
2. 「構造」ペインで、「bindings」の下の**「getURI1」**を右クリックします。
3. ポップアップ・メニューから、**「認可の編集」**を選択します。

   ユーザー権限の設定の詳細は、『Oracle WebCenter Framework 開発者ガイド』の「WebCenter アプリケーションの保護」を参照してください。
4. ダイアログ・ボックスで、メソッド oracle.SRDemo.......getURI1_return の anyone ロールに対して **invoke** を選択します。

OutputText によって生成された URL とともにようこそページが表示されます。Oracle JDeveloper に戻って、「構造」ペインからこの OutputText を削除します。

手順 3: グローバル・ナビゲーション・リンクの追加

SRWelcome.jspx を実行すると、デフォルトの SRDemo アプリケーションのログイン・ページにリダイレクトされます。作業開始時のバージョンの SRDemo では、アプリケーション内の認可は J2EE セキュリティ・ロール・メンバーシップによって実行されます。

J2EE セキュリティは、ロールに基づいてパスを保護します。制約は特定の URL パターンに対して定義され、これらの URL パターンは制約にマップされ、制約はロールにマップされます。たとえば、URL パターン faces/app/* を使用可能なロールすべてにマップして、その特定のロールを持つユーザー全員がアプリケーション内のすべてのページにアクセスできようにすることが可能です。アプリケーションが認識するように制約を web.xml ファイルに構成できます。たとえば、例 5-2 に示すように（新しい制約は**太字**テキストで表示）、manager ロールを URL パターン faces/app/manager/* にマップする制約を構成できます。

例 5-2 web.xml ファイルでのセキュリティ制約の例

```
<!-- Security Constraints ================================================ -->
 <security-constraint>
   <web-resource-collection>
     <web-resource-name>J2EE-Secured-Application</web-resource-name>
     <url-pattern>faces/app/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>user</role-name>
      <role-name>staff</role-name>
      <role-name>technician</role-name>
      <role-name>manager</role-name>
   </auth-constraint>
 </security-constraint>
<security-constraint>
   <web-resource-collection>
     <web-resource-name>J2EE-Secured-Application</web-resource-name>
     <url-pattern>faces/app/manager/*</url-pattern>
   </web-resource-collection>
   <auth-constraint>
      <role-name>manager</role-name>
   </auth-constraint>
 </security-constraint>
```

セキュリティ制約の定義の詳細は、『Oracle WebCenter Framework 開発者ガイド』の「WebCenter アプリケーションの保護」を参照してください。

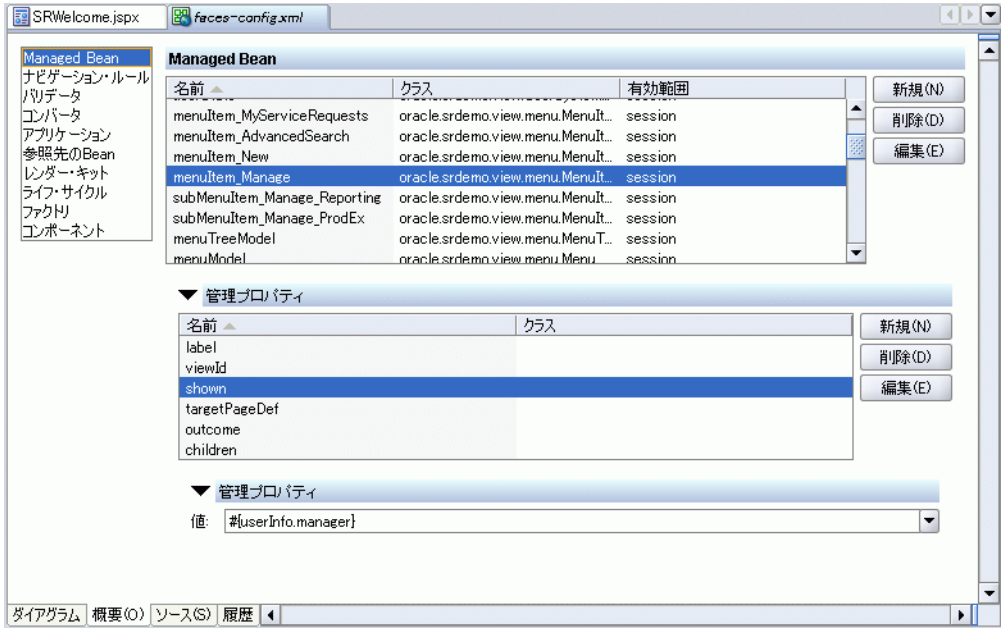
実行時、デプロイメント・ディスクリプタにより、ロールは ID ストアにマップされます。ポリシー（どのユーザーがどのロールに属し、何にアクセスできるかの定義）は、JAZN-XML や JAZN-LDAP などの JAZN リソース・プロバイダに指定されます。JAZN-XML は設定を `system-jazn-data.xml` ファイルに指定し、JAZN-LDAP は Oracle Internet Directory を使用できます。Oracle JDeveloper 開発環境では、`system-jazn-data.xml` ファイルを使用してこのポリシー情報を格納します（例 5-3 を参照）。

例 5-3 system-jazn-data.xml ファイルでのユーザーとロールのマッピング定義

```
<role>
  <name>manager</name>
  <members>
   <member>
      <type>user</type>
      <name>sking</name>
   </member>
  </members>
</role>
```

さらに、式言語をアプリケーション内で使用して、管理者にのみ表示される管理タブのように、特定のメニュー項目を表示 / 非表示にすることができます。図 5-7 に、manager ロールのメンバーシップに基づいて管理タブを表示または非表示にする方法を示します。

図 5-7 メンバーシップに基づいた UI 要素の表示



これは内部的に UserInfo クラスの isManager 関数にマップされます。この関数は、名前付きロールのメンバーシップを特定します（例 5-4 を参照）。

例 5-4 表示する UI 要素を定義する EL

```
/**
  * Function designed to be used from EL for rendering UI Features based on
  * membership of the "manager" role.
  * @return boolean
  */
 public boolean isManager() {
        return (checkIsUserInRole("manager"));
 }
```

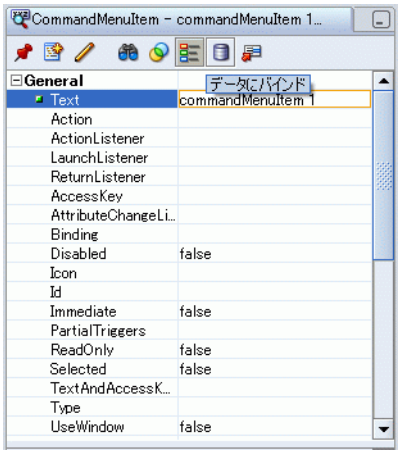
WebCenter バージョンの SRDemo では、ロール・メンバーシップではなく、特定のアクションを実行する能力に基づいた、よりきめ細かいセキュリティ・モデルを実装する必要があります。

URL に基づいた J2EE セキュリティとは異なり、JAAS ベースの ADF セキュリティを使用すると、特定のページに対して異なるアクセス・レベルを実装できます。たとえば、View、Customize および Edit があります。

ADF セキュリティベースのグローバル・ナビゲーション・リンクをページに追加するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、「SRWelcome.jspx」を開きます。
2. 「構造」ペインで、「**menuGlobal PanelPage facets**」に移動します。
3. 「設計」ビューで、menuButtons コンポーネントを menuGlobal の上にドラッグします。
4. 次に、3 つの CommandMenuItems をページ右上隅の menuGlobal ファセットにドラッグします。CommandMenuItems は、「構造」ペインの該当するノードにドロップしてもかまいません。
5. 1 つ目の commandMenuItem について、プロパティ・インスペクタで「**Text**」プロパティを選択し、「**データにバインド**」アイコンをクリックして式を Text プロパティに追加します（図 5-8 を参照）。「データにバインド」ダイアログ・ボックスが表示されます。

図 5-8 CommandMenuItem のプロパティ



6. 「JSP Objects」の下の「res」クラスを開き、srdemo.menu.home エントリまでスクロール・ダウンします。このエントリを選択し、右矢印をクリックして必要な式言語を表示し、「**OK**」をクリックします。

7. commandMenuItem の **Action** プロパティを GlobalWelcomeHome に設定します。

   使用する予定の各種スキン用のアイコンは、個々のスキン・イメージ・ディレクトリに格納されており、スキン Bean（第 4 章を参照）を使用して、EL にラップできる現行スキンに関連するディレクトリ構造を返すことができます。

8. 1 つ目の commandMenuItem の **Icon** プロパティを次のように設定します。

   ```
   /skins/#{skinBean.currentSkin}/skin_images/home.gif
   ```

   > **注意：** スキンのアイコンには、アクションを示すテキストが含まれているものもあれば、別のテキスト・ラベルを必要とするものもあります。スキン Bean には、どのスキンがアイコンのみを必要とし、あるいはアイコンおよびテキストを必要とするかを定義する機能があります。

9. デフォルト・スキンのアイコンにはすでにラベルがあるため、commandMenuItem の Text プロパティを次のように設定して、ナビゲーション・ボタンのテキスト・ラベルを使用しないようにする必要があります。

   ```
   text="#{skinBean.iconOnly ? null : res['srdemo.menu.home']}"
   ```

10. 他の 2 つの commandMenuItems について、同じ手順を繰り返し、次のようにプロパティを設定します。

    - 2 つ目の commandMenuItem について、**Action** プロパティには「GlobalHelp」を、**Text** プロパティには「srdemo.menu.help」を選択します。

      2 つ目の commandMenuItem の **Icon** プロパティを次のように設定します。

      ```
      /skins/#{skinBean.currentSkin}/skin_images/help.gif
      ```

    - 3 つ目の commandMenuItem について、**Action** プロパティには「GlobalAbout」を、**Text** プロパティには「srdemo.menu.info」を選択します。

      3 つ目の commandMenuItem の **Icon** プロパティを次のように設定します。

      ```
      /skins/#{skinBean.currentSkin}/skin_images/info.gif
      ```

    この時点で、「ソース」ビューには、commandMenuItem 定義が例 5-5 に示すように表示されます。

例 5-5 コマンド・ボタンのアクションに使用されるサンプル言語

```
<af:menuButtons>
  <af:commandMenuItem text="#{skinBean.iconOnly ? null : res['srdemo.menu.home']}"
                      action="GlobalWelcomeHome"
                      icon="/skins/#{skinBean.currentSkin}/skin_images/home.gif"/>
  <af:commandMenuItem text="#{skinBean.iconOnly ? null : res['srdemo.menu.help']}"
                      action="GlobalHelp"
                      icon="/skins/#{skinBean.currentSkin}/skin_images/help.gif"/>
  <af:commandMenuItem text="#{skinBean.iconOnly ? null : res['srdemo.menu.info']}"
                      action="GlobalAbout"
                      icon="/skins/#{skinBean.currentSkin}/skin_images/info.gif"/>
</af:menuButtons>
```

11. 「構造」ペインで、**goMenuItem** を 1 つ目の commandMenuItem の直前に追加します。

    a. プロパティ・インスペクタで**「Text」**プロパティを選択し、**「データにバインド」**アイコンをクリックして式を Text プロパティに追加します。JSF Managed Beans の下の authNLink エントリまでスクロール・ダウンします。**「label」**プロパティを選択し、右矢印をクリックして必要な式言語 text="#{authNLink.label }" を表示します。この式言語は、この後さらに更新します。

    b. **「OK」**をクリックします。

スキンのアイコンにはテキストがすでに含まれているものがあるため、Text プロパティを更新し、次のようにプロパティを設定してラベルを含むアイコンを使用できるようにする必要があります。

```
text="#{skinBean.iconOnly ? null : authNLink.label }"
```

c. 次のように、Destination プロパティを **JSF Managed Beans** の **authNLink URL** に設定します。

```
destination="#{authNLink.URL}"
```

d. 次のように、icon プロパティを状況依存スキン・アイコンに設定します。

```
/skins/#{skinBean.currentSkin}/skin_images/#{authNLink.icon}
```

authnLink は、ユーザーの認証状態を追跡管理し、ログイン / ログアウト・リンクと関連アイコンを自動的に設定するマネージド Bean です。詳細は、『Oracle WebCenter Framework 開発者ガイド』の「WebCenter アプリケーションの保護」を参照してください。

注意： SRDemo バージョンの authnLink ログイン・コンポーネントは、ブール管理プロパティ（`returnHomeOnLogout`）を定義するため、開発者は、アプリケーションのログアウト時にユーザーが現行ページにとどまるかわりに、アプリケーションのホームページ（authnLink の `homePage managed` プロパティに指定）に戻ることを指定できます。

---

12. ログインしているユーザーの名前を表示するには、コンポーネント・パレットを開き、**「JSF HTML」**を選択し、**「OutputFormat」**を選択します。このコンポーネントを「構造」ペインの `infoUser` の上にドラッグします。

13. Value プロパティを `JSP Objects/res` の下の `srdemo.connectedUser` にバインドします。

14. rendered プロパティを `JSF Managed Beans` の下の `#{authNLink.authenticated}` にバインドします（この値は、認証ステータスに基づいて `true` または `false` を戻すブールです）。

15. コンポーネント・パレットで、**「JSF Core」**を選択し、**「Param」**コンポーネントを選択します。このコンポーネントを「構造」ペインの `outputFormat` オブジェクトの上にドラッグし、次のプロパティを設定します。

- Name: `currentUserparam`
- Value: `#{userInfo.userName}`（現行ユーザーを表示）（JSF Managed Beans/userInfo の下）
- Set escape: `false`

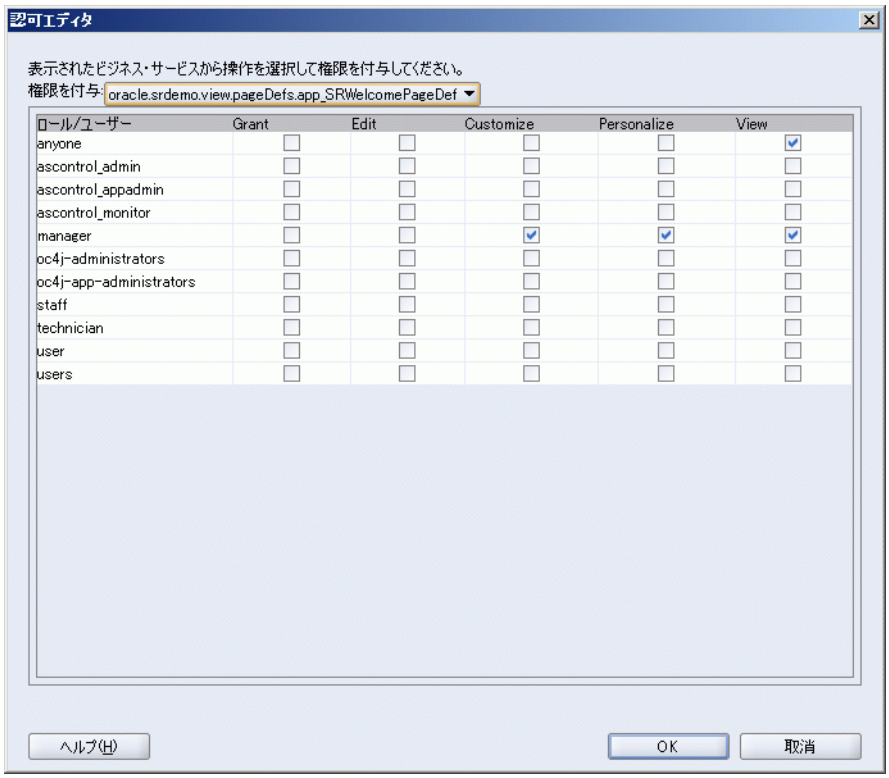
注意： `AuthnLink.currentUser` プロパティは、現行の認証済ユーザーの名前も返すため、ログイン・ユーザーの表示に使用できます。ただし、その場合、旧バージョンの SRDemo アプリケーションとの下位互換性には userInfo Bean を使用しています。

手順 4: ようこそページのパブリック化

ようこそページを最終的にパブリックにアクセスできるようにする（ADF セキュリティの構成後）には、認証済または匿名に関係なく（つまり、誰でも使用可能）、すべてのユーザーがページを表示できることを記述したアクセス・ポリシーを定義する必要があります。そのためには、次の手順を実行します。

1. 「**SRWelcome.jspx**」を右クリックしてページ定義に移動します。
2. 認可を編集して **View** 権限を anyone に付与します（図 5-9 を参照）。

図 5-9 認可エディタ



`system-jazn-data.xml` ファイルの anyone ロールに表示権限が追加されます。ADF セキュリティ（次の章で使用できるようにします）によって保護されるアプリケーションでは、各ユーザーは自動的にこの疑似ロールのメンバーになります（anyone ロール・プリンシパルはユーザーのサブジェクトに自動的に追加されます）。そのため、パブリック・ページは、誰でも使用できる特殊なタイプの保護ページです。これは、J2EE セキュリティとは異なります。J2EE セキュリティでは、パブリック・ページに対してセキュリティ制約を指定せずにそのページを定義します。そのため、ADF セキュリティ・モデルでは、パブリック・アクセスに対して保護されているページとセキュリティ保護が実装されていないページは区別されます。

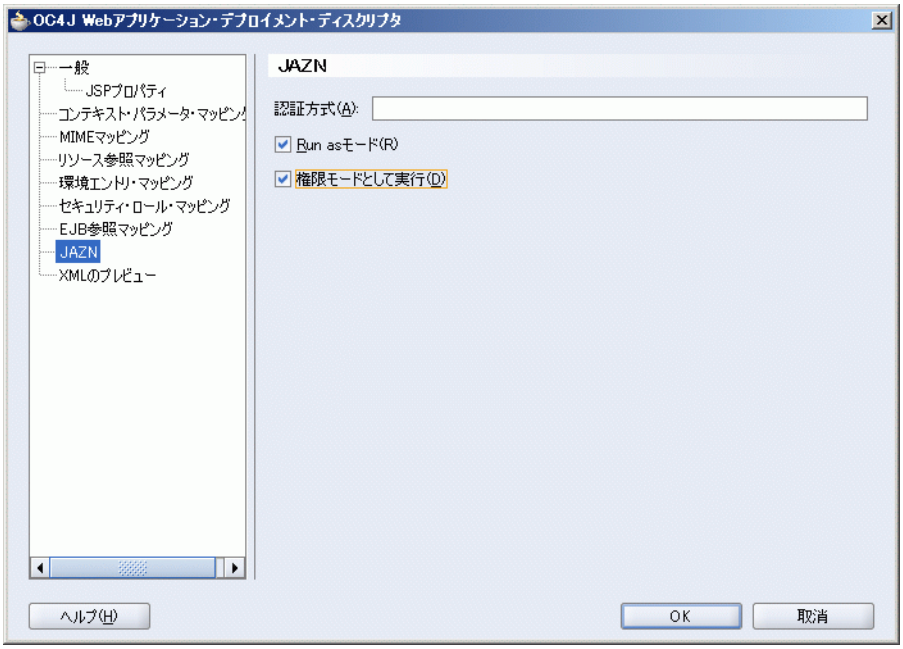
注意： 次の章では、ログイン・ページを構築して ADF セキュリティをアクティブ化してから、そのページを実行します。

---

3. アプリケーション・ナビゲータで、「Web コンテンツ」→「`WEB-INF`」フォルダの「`orion-web.xml`」ファイルを右クリックし、**「プロパティ」**を選択します。「OC4J Web アプリケーション・デプロイメント・ディスクリプタ」ダイアログ・ボックスが表示されます。

4. 「JAZN」の「Run as モード」オプションと「権限モードとして実行」オプションがすでに選択されていることを確認します（図 5-10 を参照）。

図 5-10 orion-web.xml のプロパティ



5. 「OK」をクリックします。

まとめ

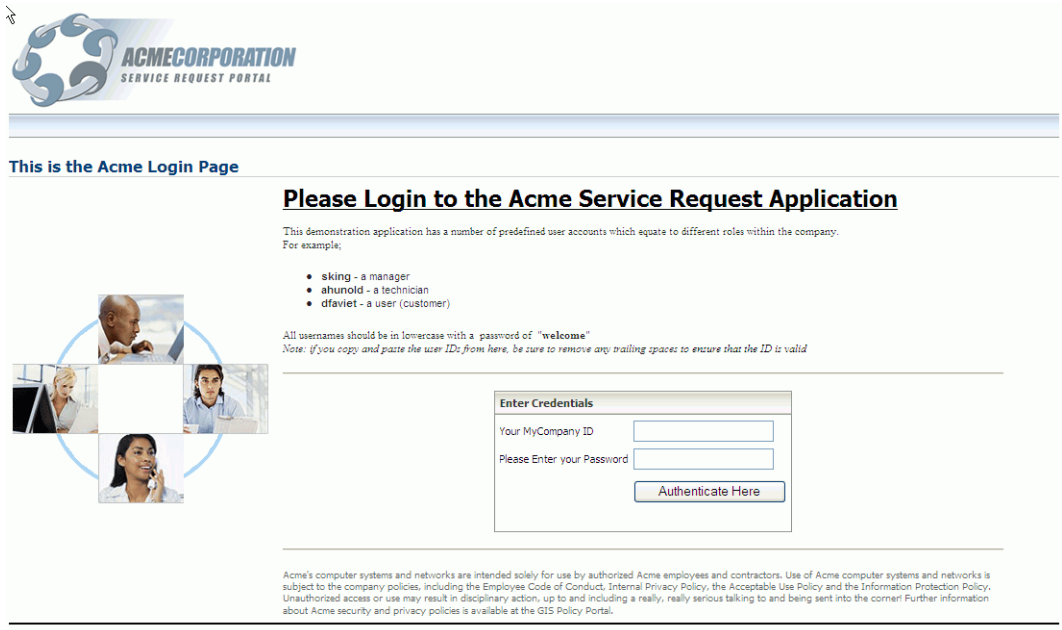
この章では、ようこそページを作成し、ポートレットおよびコンテンツ・リポジトリのコンテンツを追加する方法について学びました。また、ページをパブリックに使用できるようにし、資格証明を使用してユーザーが情報にアクセスするためのログイン・リンクを追加する方法についても学びました。

6

ログイン・ページの作成

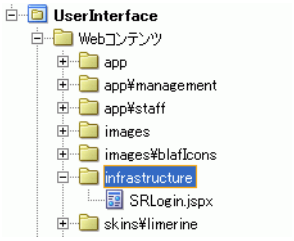
この章では、ユーザーが認証のためにリダイレクトされるログイン・ページを新規作成します。通常、WebCenter アプリケーションにはパブリック・ページの概念があり、明示的な認証も暗黙的な認証も可能です。つまり、ユーザーは、ログイン・リンクをクリックしてアプリケーションにログインしてから保護コンテンツに移動する（明示的）、あるいは保護ページに移動するとアプリケーションのログイン・ページにリダイレクトされる（暗黙的）ことが可能です。暗黙的および明示的な認証の詳細は、『Oracle WebCenter Framework 開発者ガイド』の「WebCenter アプリケーションの保護」を参照してください。図 6-1 に、この章で構築するサンプルのログイン・ページを示します。

図 6-1 SRDemo アプリケーションのログイン・ページ



SRDemoSample_Starter アプリケーションには、すでにログイン・ページ SRLogin.jspx が含まれています。このページは、図 6-2 に示す場所にあります。

図 6-2 元の SRDemo ログイン・ページ



ログイン・ページではログイン・フォームを使用するので、通常、ADF Faces ページはログイン・ページに使用しません。これは、ADF Faces ライフサイクルがログイン・フォームの送信アクションおよびフォームのフィールドを内部参照に再マップして、ログイン機能を中断するためです。しかし、ログイン・ページはアプリケーションの重要な部分です。このページには、ポートレットやデータ・コントロールなどのカスタマイズ可能な情報を含めることができます。また、ADF Faces ページで使用できるスキニング機能を利用することも可能です。また、ログイン・ポートレットを実装してもかまいません。この章では、SRDemo アプリケーション用に ADF Faces ベースのログイン・ページを構築する方法について学びます。このログイン・ページのフォームでは、J2EE セキュリティ・コンテナ・ログイン・メソッド j_security_check も使用します。この章の内容は次のとおりです。

- 手順 1: ログイン・ページの作成
- 手順 2: ログイン・エラー・ページの作成
- 手順 3: ログイン・ページへのリッチ・テキスト・ポートレットの追加
- 手順 4: ログイン・ページ用の認可の編集
- 手順 5: ADF セキュリティおよびログイン・ページを使用するためのアプリケーションの構成
- 手順 6: 新しいログイン・ページをコールするための Web.xml の更新
- 手順 7: アプリケーションの実行

手順 1: ログイン・ページの作成

ADF Faces ベースのログイン・ページを作成するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、必要に応じて、「UserInterface」プロジェクトの**「Web コンテンツ」**フォルダを開きます。
2. **「infrastructure」**フォルダ（図 6-2）を右クリックし、**「新規」**を選択します。
3. 「新規ギャラリ」ダイアログ・ボックスで、**「Web Tier」**ノードを開きます。
4. **「JSF」**を選択します。
5. 「項目」リストで、**「JSF JSP」**を選択します。
6. **「OK」**をクリックして「JSF JSP の作成」ダイアログ・ボックスを表示します。
7. ウィザードの「ようこそ」ページが表示されたら、**「次へ」**をクリックして「JSP ファイル」ページを表示します。
8. **「ファイル名」**フィールドに、SRLoginADF.jspx と入力します。
9. **「JSP ドキュメント (*.jspx)」**を選択します。
10. **「次へ」**をクリックして「コンポーネント・バインディング」ページを表示します。

11. **「新規マネージド Bean での UI コンポーネントの自動公開」**を選択します。
12. **「次へ」**をクリックして「タグ・ライブラリ」ページを表示し、**「すべてのライブラリ」**を選択します。
13. 次のライブラリが「選択済のライブラリ」に表示されていることを確認します。
    - JSF Core
    - JSF HTML
    - ADF Faces Components
    - ADF Faces HTML
    - ADF Portlet Components
    - Customizable Components Core
14. **「終了」**をクリックしてページを作成します。
15. ページを保存します。
16. `SRLoginADF.jspx` ページの「ソース」ビューで、例 6-1 に示すコードをコピーし、`f:view` タグの直前に貼り付けます。

例 6-1 SRDemo リソース・バンドルへの参照

```
<!-- Resource Bundle for Translatable Strings within Application =========== -->
<f:loadBundle basename="oracle.srdemo.view.resources.UIResources" var="res"/>
```

17. ファイルを保存します。
18. 「設計」ビューに切り替えます。
19. コンポーネント・パレットから、**「ADF Faces Core」**を選択します。
20. **「PanelPage」**コンポーネントを選択して「構造」ペインの `body` 要素の上にドラッグします。要求されても、フォームを追加しないでください。これは、後で独自のフォームを追加してデフォルトのフォーム要素を削除するためです。
21. `Title` プロパティを変更し、アプリケーションに付属するリソース・バンドルの文字列にバインドします。プロパティ・インスペクタで、**「Title」**をクリックします。
22. **「データにバインド」**アイコン（プロパティ・インスペクタの上部にあるデータベース・アイコン）をクリックします。「データにバインド」ダイアログ・ボックスが表示されます。
23. **「JSP objects」**の**「res」**を選択し、`srlogin.pageTitle` 変数を探します。
24. この変数を「式」フィールドに移動して、**「OK」**をクリックします。
25. ページ・タイトル・キー `srlogin.pageTitle` にファセットをいくつか追加します。そのためには、例 6-2 に示す既存のブランド・ファセット・コードを探します。

例 6-2 元のブランド・ファセット

```
<f:facet name="branding"/>
```

26. このブランド・イメージ・ファセット・コードを例 6-3 に示すファセット・コードで置き換えます。これにより、ブランド・ファセット内のロゴ・イメージに、選択したスキンの適切な企業ブランド・ロゴが反映されます。

例 6-3 新しいブランド・ファセット

```
<!-- Site Branding Section (Top Left of Page) ============================== -->
      <f:facet name="branding">
            <af:objectImage
source="/skins/#{skinBean.currentSkin}/skin_images/SRBranding.gif"/>
      </f:facet>
```

27. 次に、コピーライト・ファセットを新しいログイン・ページに追加します。SRLoginADF.jspx ページで、例 6-4 に示す既存のコピーライト・ファセット・コードを探します。

例 6-4 元のコピーライト・ファセット

```
<f:facet name="appCopyright"/>
```

28. このコピーライト・ファセットを例 6-5 に示すファセットで置き換えます。これにより、スキン Bean からブランド・イメージがロードされるため、スキンを変更すると、実行時にブランド・イメージも必ず変更されます。

例 6-5 コピーライト・ファセット

```
<!-- Copyright Message -->
     <f:facet name="appCopyright">
       <h:panelGroup>
         <af:outputText value="#{res['srdemo.copyright']}"/>
         <af:objectSpacer width="10" height="10"/>
       </h:panelGroup>
     </f:facet>
```

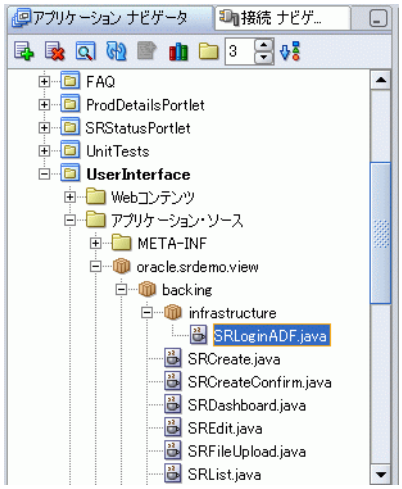
29. ログイン・ページを ADF Faces ページとして作成したので、コンポーネント・パレットのフォーム要素を使用してログイン・フォームを追加するわけにはいきません。その操作は、ADF Faces ライフサイクルによってフォーム要素が実行時にシリアライズされ、再マップされる原因となります。かわりに、アクティブな JSF コンポーネント（Faces ライフサイクルに送信されて戻ってくる）を囲むフォームにログイン・コンポーネントが依存しないようにページを設計します。これは JSF ページ内のタグをそのまま使用することで実現できますが、非常に複雑になり、利用できる柔軟性は制限されます。ログイン・コンポーネントを（一連のエスケープされた HTML タグではなく）ページをあちこち移動したり簡単にスキン化できる単一要素として処理できるようにするために、HTML フォームをユーザー・インタフェースと切り離し、実行時に動的に挿入することができます。これは、HTML フォームをバッキング Bean に組み込み、返された HTML をユーザー・インタフェースにレンダリングされる OutputText オブジェクトに公開することで実現できます。そのためには、次の手順を実行します。

注意： コードは、（ルート・ディレクトリの）開始時にダウンロードした ZIP ファイルに含まれる SRLoginADF_Backing_Code.txt ファイルから入手できます。

---

a. 「backing」→「infrastructure」フォルダ内にあるバッキング Bean の SRLoginADF.java（図 6-3 を参照）を開き、HTML 挿入コードをバッキング Bean に追加して実行時にページに挿入されるようにします。このコードは原則として、該当する HTML をページに直接生成します。そのため、様々な getter メソッドが適切にマークアップされた HTML を単純な文字列として返します。この動的に生成された HTML を使用するメリットは、スキン情報への参照を組み込むことができることです。Faces ページへの HTML の統合は、ページ内のタグおよび CDATA エスケープをそのまま使用しないことで簡略化されます。

図 6-3 アプリケーション・ナビゲータでの SRLoginADF.java の場所



b. 今度は、Java コードをファイルに追加します。そのためには、例 6-6 に示す import 文を追加します。

例 6-6 import 文

```
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpSession;
import oracle.adf.share.ADFContext;
import oracle.adf.view.faces.component.core.output.CoreMessages;
import oracle.srdemo.view.util.JSFUtils;
```

c. バッキング Bean のプロパティを SRLoginADF.java の先頭に追加します（例 6-7 を参照）。これらのプロパティの getter メソッドは、後でログイン・ページの式言語で参照されます。

例 6-7 バッキング Bean のプロパティ

```
private String  _loginStyleBlock   = null;
private String  _loginScriptBlock  = null;
private String  _loginFormBlock    = null;
private boolean _validLoginAttempt = true;
```

表 6-1 に、これらのプロパティの説明を示します。

表 6-1 SRLoginADF.java ファイルのバッキング Bean のプロパティ

変数	説明
loginStyleBlock	CSS クラスを定義するための HTML が含まれます。
loginScriptBlock	ログイン・フォームで使用される必須 JavaScript の HTML が含まれます。
loginFormBlock	ログイン・フォーム自体の HTML が含まれます。
validLoginAttempt	現行ユーザーがもう一度ログインを試行できるかどうかを示すブール値。ユーザーが有効なログイン試行回数を超過した場合、`false` に設定されます。

d. `getloginStyleBlock` getter メソッドをバッキング Bean に挿入します。例 6-8 を参照してください。

このメソッドは、認証プロセスが実行中であることを示すために、ログイン・フォームで使用されるフラッシュ・テキストを定義する単純なスタイルシートを生成します。このメソッドによって生成される HTML は表示できないため、ページの本体の外側に出力文字列をレンダリングできます。

例 6-8 getloginStyleBlock getter メソッド

```
/**
  * ============================================================================
  * Function to generate the CSS style class referenced in Login Form
  * ============================================================================
  * @return
  */
public String getloginStyleBlock()
{
  _loginStyleBlock = "\n\n"
                   + "<!--
========= CSS Style Block Generated in Backing Bean ========= -->\n"
                   + "<style type=\"text/css\">\n"
                   + ".VP_Blink {text-decoration: blink;}\n"
                   + "</style>\n"
                   + "<!--
 ============================================================ -->"
                   + "\n\n";
  return (_loginStyleBlock);
}
```

e. `getloginScriptBlock` getter メソッドをバッキング Bean に挿入します。例 6-9 を参照してください。

このメソッドは、ログイン・フォームで使用される JavaScript 関数を含む `<script>` ブロックを生成します。これらの関数には、フィールド検証アクションやフォーム送信アクションがあります。JavaScript は動的に生成されるため、現行スキンへの参照の他にリソース・バンドルの適切な文字列を組み込むことができます。たとえば、現行スキンが `myCompany` に設定されている場合、次の行は、JavaScript のアラート・メッセージをリソース・バンドル・キーに定義されている値（`SRLoginADF.myCompany.jsAlert`）になるように設定します。

```
String alertMsg    = JSFUtils.getStringFromBundle("SRLoginADF." +
currentSkin + ".jsAlert");
```

同様に、このメソッドは、認証中であることを示す適切なアイコンを特定し、Faces 外部コンテキストに基づいてそのアイコンへの URL を生成します。

例 6-9 getloginScriptBlock getter メソッド

```
/**
 * ============================================================================
 * Method getloginScriptBlock() to generate a JScript Submit form block.
 * This function implements the current skin and generates simple JavaScript
 * to validation the user input prior to the Authentication submit action.
 * ============================================================================
 * @return
 */
public String getloginScriptBlock()
{
 String currentSkin = (String)
 JSFUtils.getManagedBeanValue("skinBean.currentSkin");
 String alertMsg    = JSFUtils.getStringFromBundle("SRLoginADF."
                                                   + currentSkin +
                                                   ".jsAlert");
 String valPwdMsg   = JSFUtils.getStringFromBundle("SRLoginADF."
                                                   + currentSkin +
                                                   ".AuthMsg");
 String urlBaseRef  =
FacesContext.getCurrentInstance().getCurrentInstance().getExternalContext().getRequestC
ontextPath();
 String processIcon = urlBaseRef + "/skins/" + currentSkin +
"/skin_images/process_animation.gif";

 _loginScriptBlock = "\n\n"
                   + "<!--
======== JavaScript Block Generated in Backing Bean ========= -->\n"
                   + "<SCRIPT language=\"JavaScript\" type=\"text/JavaScript\">\n"
                   + "(new Image(32,32)).src=\"" + processIcon +"\";\n"
                   + "function SubmitForm(AuthFrm){\n"
                   + "var alertMsg =\""   + alertMsg  + "\";\n"
                   + "var valPwdMsg = \"" + valPwdMsg + "\";\n\n"
                   + "if (((AuthFrm.j_username.value == null) ||
(AuthFrm.j_username.value ==\"\")) ||\n"
                   + "    ((AuthFrm.j_password.value == null) ||
(AuthFrm.j_password.value ==\"\")))\n"
                   + " {\n  alert(alertMsg);\n  return false;\n }\n else\n {\n"
                   + " var divTag = document.getElementById(\"vp\");\n"
                   + " divTag.innerHTML = '<table width=\"50\" border=\"0\"><tr>' +\n"
                   + "                    '<td align=\"center\"><img src=\""
                                          + processIcon + "\" width=\"32\"
                                          height=\"32\"></td>' +\n"
                   + "                    '<td align=\"left\"><font size=\"-2\"
                                          class=\"VP_Blink\">' + valPwdMsg +
                                          '</font></td>' +\n"
                   + "                    '</tr></table>';\n"
                   + " return true;\n"
                   + " }\n}\n"
                   + "</SCRIPT>\n"
                   + "<!--
 ============================================================ -->"
                   + "\n\n";
return (_loginScriptBlock);
}
```

f. `getLoginFormBlock` getter メソッドをバッキング Bean に挿入します。例 6-10 を参照してください。

このメソッドは、ページ内にレンダリングされる HTML ログイン・フォームを定義する文字列を生成します。HTML は動的であるため、現行スキンへの参照の他にリソース・バンドルのキーも組み込むことができます。さらに、ログイン・ページはログイン後にカスタマイズできるように設計されているため、ADF セキュリティ・コンテキストの `isAuthenticated();` メソッドを使用して HTML の disabled プロパティが動的に設定されるので、HTML はユーザーの認証状態を敏感に反映します。

その場合、ユーザーが認証されてログイン・ページがアクセスされると、ログイン・ページは（レンダリング時に）非アクティブ化されてそれ以降のログイン試行はできなくなります。たとえば、サイト管理ページからページがアクセスされる場合（第 9 章「サイト管理ページの構築」を参照）がこれに該当します。

次の例では、ADF セキュリティ・コンテキストの `isAuthenticated` の状態に基づいて、文字列変数 `htmlDisable` が `"disabled="true""` または `null` のいずれかになることがわかります。

```
String htmlDisable
=(ADFContext.getCurrent().getSecurityContext().isAuthenticated() ?
"disabled=\"true\"" : null);
```

次のコードは、その後で、入力フィールドや「送信」ボタンなどのフォーム要素をアクティブ化または無効化するために、`htmlDisable` 文字列を使用する方法を示しています。

```
<input type=\"text\" name=\"j_username\" "+ htmlDisable + "/></td>\n"
```

例 6-10 getLoginFormBlock getter メソッド

```
/**
 * ============================================================================
 * Method getLoginFormBlock() generates the standard Credential Form used
 * by container security. The function implements the current skin values and
 * returns the required HTML which is subsequently output using an OutputText
 * object populated using EL. While this could also have been achieved by
 * directly using the Verbatim tags within the JSPX itself, this would require the
 * use of CDATA escapes and complicates the design of the page (the login form
 * is exposed through a faces component in this case.)
 *
 * Note as the Login Page contains customizable components, the HTML form elements
 * generated are disabled if the user is currently authenticated.
 * ============================================================================
 * @return
 */

public String getLoginFormBlock()
{
  String currentSkin       = (String)
JSFUtils.getManagedBeanValue("skinBean.currentSkin");
  String userNameLabel     = JSFUtils.getStringFromBundle("SRLoginADF."
                                                          + currentSkin +
                                                          ".userName");
  String passwordLabel     = JSFUtils.getStringFromBundle("SRLoginADF."
                                                          + currentSkin +
                                                          ".password");
  String submitButtonLabel = JSFUtils.getStringFromBundle("SRLoginADF." +
                                                           currentSkin +
                                                          ".buttonLabel");
  String htmlDisable       =
(ADFContext.getCurrent().getSecurityContext().isAuthenticated() ? "disabled=\"true\"" :
null);
```

```
  _loginFormBlock = "\n\n"
                  + "<!--
======== Login Form Block Generated in Backing Bean ========= -->\n"
                  + "<form name=\"LoginForm\" id=\"LoginForm\" \n"
                  + " action=\"j_security_check\" method=\"POST\"
                      onSubmit=\"return SubmitForm(this)\" >\n"
                  + "<table cellspacing=\"5\" cellpadding=\"0\" border=\"0\"
                     width=\"50%\" bgcolor=\"#FFFFFF\" width=\"240\" >\n"
                  + " <tr>\n"
                  + "  <td nowrap>" + userNameLabel + "</td>\n"
                  + "  <td nowrap><input type=\"text\" name=\"j_username\"
                    "  + htmlDisable + "/></td>\n"
                  + " </tr>\n"
                  + " <tr>\n"
                  + "  <td nowrap>" + passwordLabel + "</td>\n"
                  + "  <td nowrap><input type=\"password\" name=\"j_password\"
                    " + htmlDisable + "/></td>\n"
                  + " </tr>\n"
                  + " <tr>\n"
                  + "  <td nowrap height=\"34\"><DIV id=\"vp\"></DIV></td>\n"
                  + "  <td nowrap>\n"
                  + "    <input type=\"submit\" value=\"" + submitButtonLabel +
                    "\" " + htmlDisable + "/></td>\n"
                  + "  </td>\n"
                  + " </tr>\n"
                  + "</table>\n"
                  + "</form>\n"
                  + "<!--
 ============================================================ -->"
                  + "\n\n" ;
  return ( _loginFormBlock);
}
```

g. ファイルを保存します。

30. コンポーネント・パレットから、**「ADF Faces Core」**を選択します。

31. **「PanelHorizontal」**を選択して page コンポーネントの上にドラッグします。

32. プロパティ・インスペクタで Valign プロパティを middle に設定します。

33. ADF Faces Core コンポーネント・パレットから**「ObjectImage」**を選択して PanelHorizontal コンポーネントの上にドラッグします。source 属性を次の値に設定します。

```
/skins/#{skinBean.currentSkin}/skin_images/LoginSplash.gif
```

34. **「ObjectSpacer」**を選択して ObjectImage コンポーネントのすぐ下の page の上にドラッグします。

35. コンポーネント・パレットから、**「カスタマイズ可能コンポーネント・コア」**を選択します。

36. **「PanelCustomizable」**を選択し、「構造」ペインで追加したばかりの ObjectSpacer 要素の下にドラッグします。要求されても、フォームを追加しないでください。これは、後で独自のフォームを追加してデフォルトのフォーム要素を削除するためです。

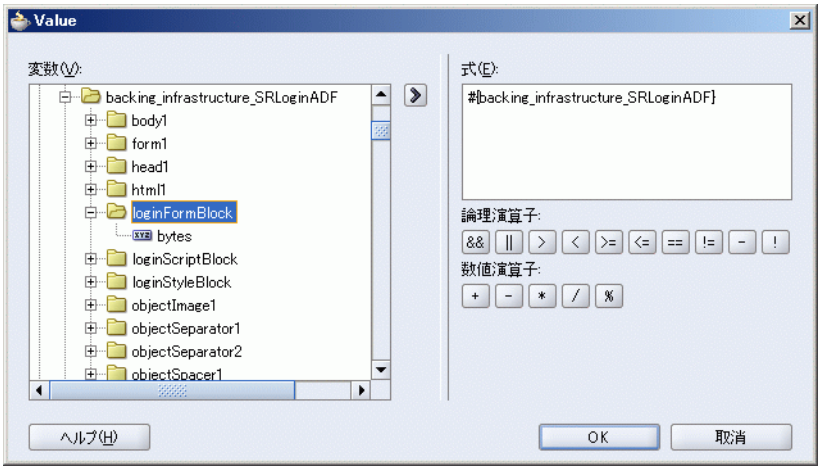
37. PanelCustomizable コンポーネントの Layout プロパティを vertical に設定します。

38. コンポーネント・パレットから、**「ADF Faces Core」**を選択します。

39. 今度は、次のものを PanelCustomizable コンポーネントに追加します。
    - objectSpacer を 1 つ
    - objectSeparator を 1 つ
    - objectSpacer を 1 つ
    - panelHorizontal を 1 つ

      Halign プロパティを center に設定します。
    - objectSpacer を 1 つ
    - objectSeparator を 1 つ
    - objectSpacer を 1 つ
40. 「**PanelBox**」を選択して PanelHorizontal コンポーネントの上にドラッグします。
41. 式言語（EL）を使用して、PanelBox の Text プロパティを、リソース・バンドルを参照することでログイン・コンポーネント・ヘッダーになるように設定します。そのためには、「**データにバインド**」アイコンをクリックします。「データにバインド」ダイアログ・ボックスが表示されます。
42. 「**JSP objects**」の「**res**」を選択し、SRLoginADF.credentialHeader 変数を探します。
43. ADF Faces Core コンポーネント・パレットから「**OutputText**」を選択して PanelBox コンポーネントの上にドラッグします。
44. プロパティ・インスペクタで、「**JSF Managed Beans**」、「**backing_infrastructure_SRLoginADF**」、「**loginFormBlock**」の順に選択して、OutputText コンポーネントの Value プロパティをマネージド Bean のデータにバインドします（図 6-4 を参照）。

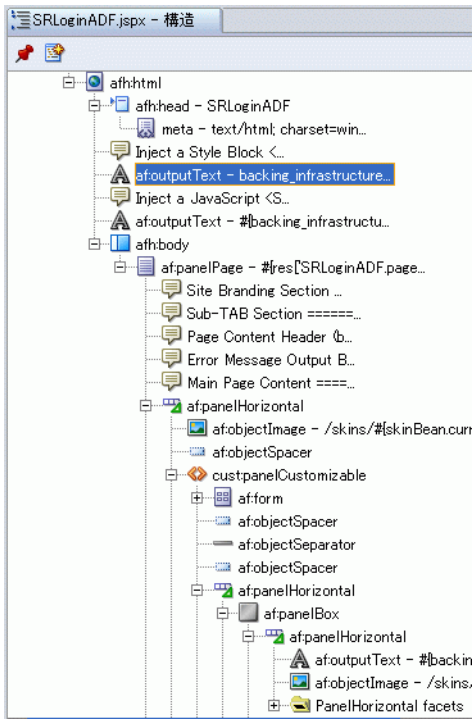
図 6-4 データへのバインド



45. OutputText コンポーネントの Escape プロパティを false に設定します。

46. 別の **OutputText** コンポーネントを「構造」ペインの body タグの上（ページの先頭のすぐ下）にドラッグします（図 6-5 を参照）。

図 6-5「構造」ペイン



47. 「**JSF Managed Beans**」、「**backing_infrastructure_SRLoginADF**」、「**loginStyleBlock**」の順に選択して `Value` プロパティをバッキング Bean のデータにバインドし、`Escape` プロパティを `false` に設定します。

48. 別の **OutputText** コンポーネントを、「構造」ペインの body タグのすぐ上にドラッグします。

49. 「**JSF Managed Beans**」、「**backing_infrastructure_SRLoginADF**」、「**loginScriptBlock**」の順に選択して `Value` プロパティをバッキング Bean のデータにバインドし、`Escape` プロパティを `false` に設定します。

50. `SRLoginADF.jspx` ページの「構造」ペインで、「**panelPag**」ノードを開き、**Messages** コンポーネントを `messages` ファセットの上にドラッグします。

51. ファイルを保存します。

手順 2: ログイン・エラー・ページの作成

ログイン・エラー・ページを作成するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、必要に応じて、「UserInterface」プロジェクトの**「Web コンテンツ」**ノードを開きます。
2. **「infrastructure」**フォルダ（図 6-2）を右クリックし、**「新規」**を選択します。
3. 「新規ギャラリ」ダイアログ・ボックスで、**「Web Tier」**ノードを開きます。
4. **「JSP」**を選択します。
5. 「項目」リストで、**「JSP」**を選択します。
6. **「OK」**をクリックして「JSP の作成」ダイアログ・ボックスを表示します。
7. ウィザードの「ようこそ」ページが表示されたら、**「次へ」**をクリックして「JSP の作成」ページを表示します。
8. **「ファイル名」**フィールドに、`SRLoginErrorPage.jsp` と入力します。
9. ウィザードの「エラー・ページ・オプション」ページで、**「このファイルの捕捉されない例外の処理にエラー・ページを使用しない」**を選択します。
10. 「タグ・ライブラリ」ページで、**「すべてのライブラリ」**を選択し、**「JSTL Core 1.1」**を選択して「選択済のライブラリ」リストに移動します。
11. **「終了」**をクリックします。
12. 例 6-11 に示す JSTL コードを `<html>` 要素の直前に追加して、ログイン試行回数を追跡管理します。

例 6-11 ログイン試行回数を追跡管理するための JSTL コード

```
<!-- ============================================================================
 For the SRDemo Demo we will track the number of Login Attempts against the file
 based repository (In a deployed system this would be handled by the IdM system).
 This counter is then used in the backing bean to define if the login attempt
 is valid (referenced in EL as "loginAttemptValid").
 ============================================================================ -->
<c:set var="LoginErrorCount"
       scope="session"
       value="${sessionScope.LoginErrorCount == null ? 1 :sessionScope.LoginErrorCount
+ 1}"/>
<c:redirect url="/faces/infrastructure/SRLoginADF.jspx"/>
```

この JSTL コードにより、（無効なログインによって）エラー・ページがコールされた回数が単純なセッション変数に格納され、ユーザーは実際の処理が行われている元のログイン・ページにリダイレクトされます。

13. `SRLoginErrorPage.jsp` ファイルを保存します。
14. バッキング Bean の「`SRLoginADF.java`」を開き、例 6-12 に示す `getMessages1()` メソッドを追加します。

次の 2 つのメソッドでは、Faces 外部コンテキストを使用してエラー・ページで JSTL によって設定されたセッション変数を返し、ユーザーがログインを試行した回数を特定します。

次のコードは、Faces コンテキストおよび関連付けられた HTTP セッションに基づいてセッション変数を返す方法を示しています。返されたオブジェクトは、ユーザーのログイン試行回数が有効か否かを評価するために、ローカル整数に変換されます。たとえば、試行回数上限 3 回は次のように適用できます。

```
FacesContext facesContext = FacesContext.getCurrentInstance();
HttpSession  session =
(HttpSession)facesContext.getExternalContext().getSession(true);
Object loginAttempts  = session.getAttribute("LoginErrorCount");
Integer loginAttemptCount = (loginAttempts == null)? 0 :
Integer.parseInt(loginAttempts.toString());
```

注意： getMessages1 メソッドは、PanelPage の message コンポーネント用に作成されるデフォルトの getter メソッドです。message コンポーネントが messages1 ではない場合、メソッドの名前を変更する必要があります。

現在のログイン試行回数が特定されると、getMessages1 メソッドは、（リソース・バンドルから返される）適切なエラー文字列を Faces の message コンポーネントに設定します。その場合、2 回以上の試行は無効なユーザー名とパスワードの組合せを示し、4 回以上の試行はユーザーが適正な上限を超過したことを示します。

例 6-12 getMessage1() メソッド

```
/**
     * ========================================================================
     * Function getMessages1() determines the current number of login failures and
     * sets the appropriate error message in the Faces Context
     * ========================================================================
     * @return
     */
    public CoreMessages getMessages1()
    {
     if (!ADFContext.getCurrent().getSecurityContext().isAuthenticated())
     {
       String       currentSkin       = (String)
       JSFUtils.getManagedBeanValue("skinBean.currentSkin");
       FacesContext facesContext      = FacesContext.getCurrentInstance();
       HttpSession  session           = (HttpSession)
       facesContext.getExternalContext().getSession(true);
       Object       loginAttempts     = session.getAttribute("LoginErrorCount");
       Integer      loginAttemptCount = (loginAttempts == null)? 0 :
       Integer.parseInt(loginAttempts.toString());
       String       loginErrorMessage = null;
       if (loginAttemptCount >0)
       {
        if (loginAttemptCount >2) {
         loginErrorMessage = JSFUtils.getStringFromBundle(
                             "SRLoginADF." + currentSkin +
                             ".tooManyLoginAttempts");
        }
        else {
         loginErrorMessage = JSFUtils.getStringFromBundle(
                    "SRLoginADF." + currentSkin + ".invalidLogin");
        }
        FacesMessage fm =  new FacesMessage(
                   FacesMessage.SEVERITY_INFO,loginErrorMessage,null);
        facesContext.addMessage(null, fm );
       }
     }
     return messages1;
    }
```

15. バッキング Bean の「SRLoginADF.java」を開き、例 6-13 に示す isValidLoginAttempt メソッドを追加します。

    isValidLoginAttempt メソッドから返されるブール値は、ログイン・フォームの HTML コードが含まれる outputtext コンポーネントの rendered プロパティで参照されます。その場合、有効な試行回数が 3 回を超えるとき、ログイン・フォームはページにレンダリングされず、ユーザーは再度ログインすることができなくなります。

例 6-13 isValidLoginAttempt メソッド

```
/**
 * ============================================================================
 * Method isValidLoginAttempt() evaluates the number of invalid login attempts
 * performed by the user and returns false if greater than 3. (used in EL to
 * show/hide the Login Page components.
 * ============================================================================
 * @return
 */
public boolean isValidLoginAttempt()
{
  _validLoginAttempt = true;
  if (!ADFContext.getCurrent().getSecurityContext().isAuthenticated()) {
     FacesContext facesContext      = FacesContext.getCurrentInstance();
     HttpSession  session           = (HttpSession)
     facesContext.getExternalContext().getSession(true);
     Object       loginAttempts     = session.getAttribute("LoginErrorCount");
     Integer      loginAttemptCount = (loginAttempts == null)? 0
    : Integer.parseInt(loginAttempts.toString());

     if (loginAttemptCount >2){ _validLoginAttempt = false; }
  }
  return (_validLoginAttempt);
}
```

16. ファイルを保存します。

17. 「SRLoginADF.jspx」を開きます。

18. **ObjectImage** を OutputText オブジェクトの下の PanelBox にドラッグし、source プロパティを次の値に設定します。

    source="/skins/#{skinBean.currentSkin}/skin_images/BadLoginAttempt.gif"

19. rendered プロパティをバッキング Bean の isValidLoginAttempt() メソッドの否定（反対）値にバインドします。rendered プロパティを次の値に設定します。

    #{!backing_infrastructure_SRLoginADF.validLoginAttempt}

    この設定により、イメージはログイン試行が無効の場合のみレンダリングされます。

20. OutputText オブジェクトの rendered プロパティを、次のような isValidLoginAttempt() メソッドの値に設定します。

    #{backing_infrastructure_SRLoginADF.validLoginAttempt}

    これで、ログイン・フォームはユーザーのログイン試行が有効な場合のみレンダリングされます。

手順 3: ログイン・ページへのリッチ・テキスト・ポートレットの追加

この項では、リッチ・テキスト・ポートレットをログイン・ページに追加します。そのためには、次の手順を実行します。

1. 作業を進める前に、リッチ・テキスト・ポートレットが登録されていることを確認します。詳細は、第 5 章「パブリックなようこそページの作成」の「手順 1: リッチ・テキスト・コンポーネントの追加」を参照してください。
2. コンポーネント・パレットから、**「RichTextPortletProducer」**を選択し、リストから**「リッチ・テキスト」**を選択します。「構造」ペインで、この**リッチ・テキスト・ポートレット**を `panelCustomizable` 内の先頭の `objectSpacer` の上にドラッグします。
3. リッチ・テキスト・ポートレットを選択し、プロパティ・インスペクタを開きます。
4. 次のプロパティを設定します。

プロパティ	値
AllModesSharedScreen	true
IsMinimizable	false
IsMaximizable	false
DisplayHeader	false

5. コンポーネント・パレットから、**「RichTextPortletProducer」**を選択し、リストから**「リッチ・テキスト」**を選択します。「構造」ペインで、この**リッチ・テキスト・ポートレット**を `panelCustomizable` 内の最後の `objectSpacer` の下にドラッグします。
6. リッチ・テキスト・ポートレットを選択し、プロパティ・インスペクタを開きます。
7. 次のプロパティを設定します。

プロパティ	値
AllModesSharedScreen	true
IsMinimizable	false
IsMaximizable	false
DisplayHeader	false

8. ページを保存します。

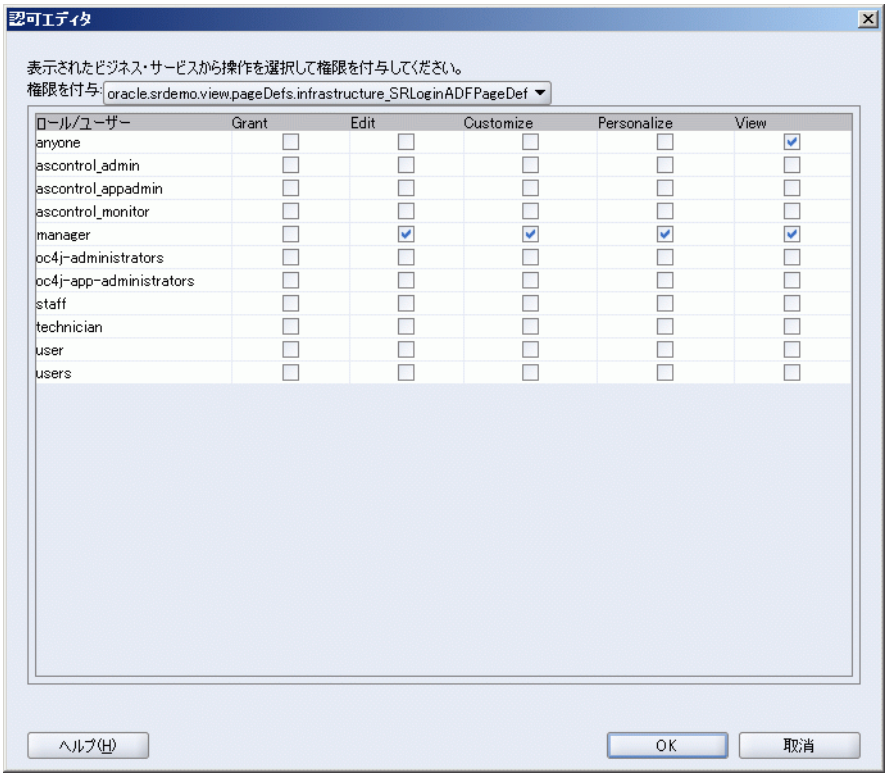
これで、ログイン後にログイン・ページに戻って、このリッチ・テキスト・ポートレットをカスタマイズできます。その実現方法は、第 9 章「サイト管理ページの構築」を参照してください。

手順 4: ログイン・ページ用の認可の編集

次に、適切な権限をログイン・ページに定義する必要があります。すべてのユーザーがページを表示できる必要があります。また、管理者は、リッチ・テキスト・ポートレットを更新できるように、ページの編集もできる必要があります。ログイン・ページにセキュリティを設定するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、**「SRLoginADF.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。
3. ページ定義を新規作成するように要求された場合は、**「はい」**をクリックします。「構造」ペインでページ定義ファイルが開きます。
4. 「Page Definition」ファイルを右クリックし、**「認可の編集」**を選択します。認可エディタが表示されます。
5. **View** 権限を *anyone* に、**Edit** 権限を *manager* に付与します（図 6-6 を参照）。

図 6-6 ログイン・ページにセキュリティを定義するための認可エディタ



手順 5: ADF セキュリティおよびログイン・ページを使用するためのアプリケーションの構成

この手順では、Oracle ADF セキュリティ・ウィザードを使用して SRDemo アプリケーションの認証設定を構成します。選択したオプションはすべて `web.xml` に記録されます。認証設定を構成するには、次の構成作業を実行する必要があります。

- Oracle ADF 認証を有効にします。
- 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としてフォームベースを指定します。
- 認証済ユーザー（`ValidUser`）に `adfAuthentication` サーブレットへのアクセス権を付与します。

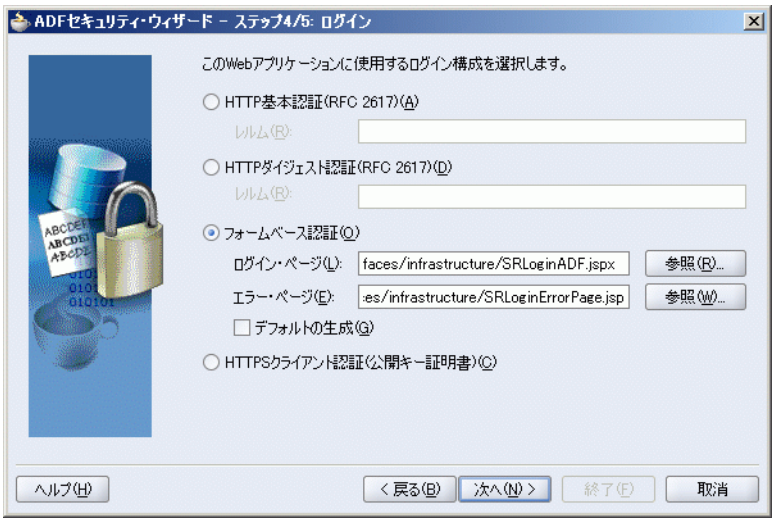
ADF セキュリティおよびログイン・ページを使用するように SRDemo アプリケーションを構成するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、**「UserInterface」**を開きます。
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・ウィザードにより、構成プロセスを誘導されます。
3. 必要に応じて、**「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」**が選択されていることを確認します。このオプションにより、`adfAuthentication` サーブレットが構成され、認可ルール（ページに対する現行ユーザーの権限をチェックできる適切なフィルタ）が構成されます。
5. **「次へ」**をクリックしてウィザードの次のページに移動します。
6. **「軽量 XML プロバイダ」**を選択します。Oracle ADF セキュリティでは、特定のリソース・プロバイダに対してユーザーを認証します。SRDemo の場合、ダウンロードしたサンプル・ファイルに含まれる `install.html` ファイルの手順に従ってコピーした、軽量リソース・プロバイダの `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. このページで、「場所」に**「アプリケーション・リポジトリ」**、「デフォルト・レルム」に**jazn.com**、「JAAS モード」に**doAsPrivileged** と設定し、**「次へ」**をクリックします。
9. 「ログイン」ページで、**「フォームベース認証」**を選択します（図 6-7）。これにより、SRDemo アプリケーションではフォームを使用して認証が円滑に行われます。

   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。これは、`SRLoginADF.jspx` 内にすでに構築したログイン・フォームを使用するためです。
10. 「ログイン・ページ」フィールドに、`faces/infrastructure/SRLoginADF.jspx` と入力します。

11. また、「エラー・ページ」フィールドに、`faces/infrastructure/SRLoginErrorPage.jsp` と入力できます（図 6-7 を参照）。

図 6-7 フォームベース認証の詳細



12. **「次へ」**をクリックしてウィザードの最後のページを表示します。

    このページは、保護対象とな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。`adfAuthentication` リソース（認証サーブレット）が定義されています。このリソースは、編集または削除できませんが、このリソースにアクセスできる一連のロールを指定できます。デフォルトで J2EE ロールが 1 つ（`oc4j-administrators`）選択されていますが、`adfAuthentication` リソースには任意の有効なユーザーがアクセスできるようにする必要があるため、`ValidUser` という名前の J2EE ロールを別に作成し、このロールにアクセス権を付与する必要があります。

13. Web リソースの **J2EE-Secured-Application** が表示されている場合は、これを削除します。

14. **「ロールの管理」**をクリックします。

15. **「追加」**をクリックし、名前の `ValidUser` を入力します。

    後で、この J2EE ロールを ID ストアのロールの 1 つ `users` にマップします。これは、このレッスンの始めに定義したロールです。このロールは、すべての有効なユーザーのリストを保持しています。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、認証されたパブリック・リソースが事実上定義されます。つまり、特定の権限を定義する必要なしにすべてのユーザーが使用できるようになります。

16. **「OK」**をクリックします。`ValidUser` ロールがリストに表示されます。

17. **「閉じる」**をクリックします。

18. 二重矢印（**「すべて追加」**）をクリックし、「使用可能なロール」リストのロールをすべて「選択したロール」リストに移動します。

19. ADF セキュリティ・ウィザードの設定はこれで完了したので、**「次へ」**をクリックし、**「終了」**をクリックします。

20. これで、アプリケーションの `SRWelcome.jspx` ページを実行でき、パブリックにアクセスできるようこそページが表示されます。

21. **「login」**リンクをクリックします。この章で構築したログイン・ページが表示されます。

22. 管理者（`sking`）としてログインします。RTP のコンテンツは、通常、サイト管理ページに定義されます。このページは後で構築します。

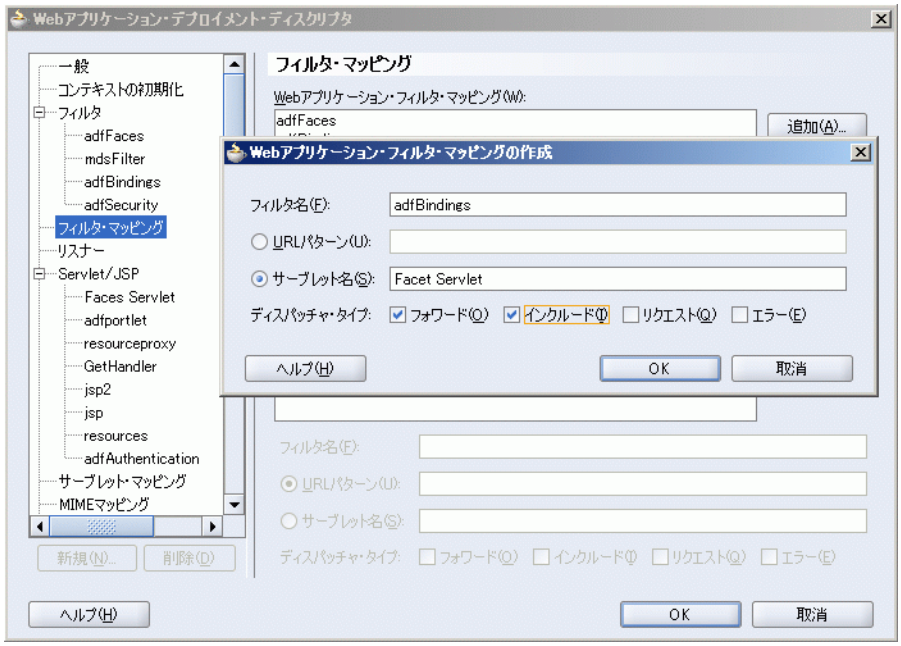
注意： サイト管理を作成する前に、ページに対して Customize 権限を持つユーザーとしてページ URL に直接アクセスすると、ログイン・ページをカスタマイズできます。

手順 6: 新しいログイン・ページをコールするための Web.xml の更新

新しいログイン・ページをコールするように web.xml を更新する手順は、次のとおりです。

1. アプリケーション・ナビゲータで、**「WEB-INF」**ノードを開き、「web.xml」プロパティ・パレットを開きます（ポップアップ・メニューから「プロパティ」を選択します）。
2. 左側のパネルで**「フィルタ・マッピング」**を選択し、新しいフィルタ・マッピングを adfBindings フィルタに追加します（図 6-8 を参照）。

図 6-8 adfBindings フィルタに対する新しいフィルタ・マッピング



必ず、「サーブレット名」に Faces Servlet と入力し、「フォワード」および「インクルード」の各ディスパッチャ・タイプを選択します。

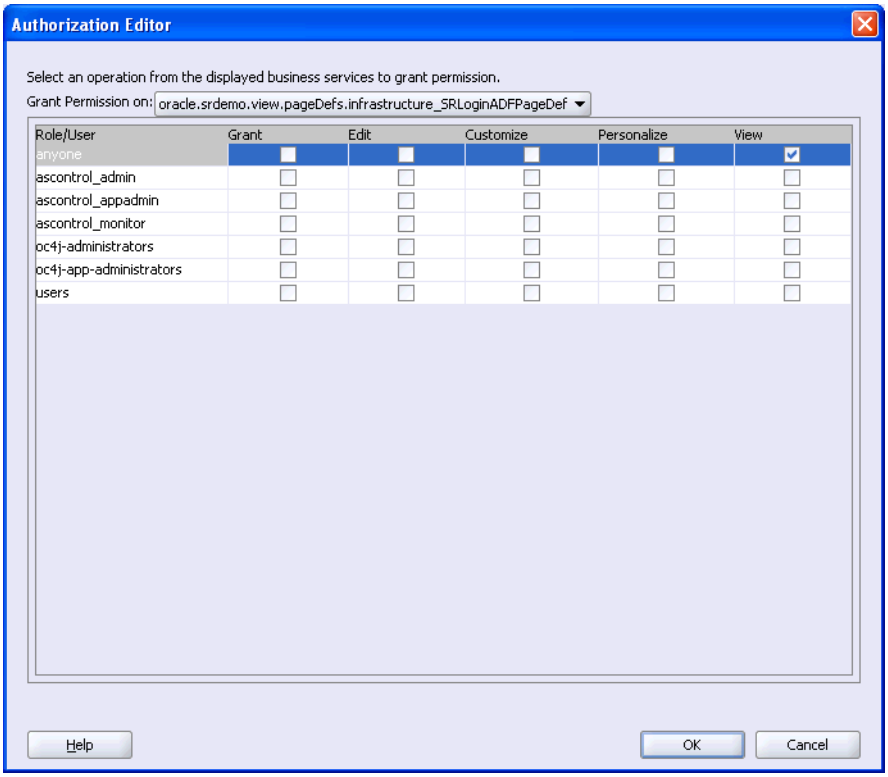
3. **「OK」**をクリックして web.xml への変更内容を保存します。

注意： ログイン・ページは JSPX ページでもある（よって ADF セキュリティによって保護されている）ため、パブリック・ページとして定義する必要があります。ページがパブリックとして定義されないと、コンテナは、このページへのアクセスを許可するまでずっと定義された認証ポイントにリダイレクトします（当然、このページは何度も繰り返されます）。

4. アプリケーション・ナビゲータで、「SRLoginADF.jspx」ページを右クリックし、**「ページ定義に移動」**を選択します。ページ定義ファイルが現在存在しない場合は、そのファイルの作成を確認します。
5. ページ定義ファイルを右クリックし、**「認可の編集」**を選択します。

6. **View** 権限を *anyone* ロールに付与します（図 6-9 を参照）。

図 6-9 認可エディタ



7. ログイン・ページには、カスタマイズ可能なポートレットが含まれているため、該当するロールにはページに対する Customize 権限が必要です。

手順 7: アプリケーションの実行

これで、アプリケーションを実行し、作成したログイン・ページとエラー・ページが正しく表示されるかどうかをテストできます。そのためには、次の手順を実行します。

1. `SRWelcome.jspx` ページを実行します。
2. 「**login**」リンクをクリックします。ログイン・ページが表示されます。
3. 有効なユーザー名およびパスワードを使用してログインします。

   デモ・アプリケーションに正常にログインします。
4. アプリケーションからログアウトします。
5. エラー・ページが正しく表示されるかどうかをテストするには、間違った資格証明を 3 回使用してログインします。エラー・ページが表示されます。

まとめ

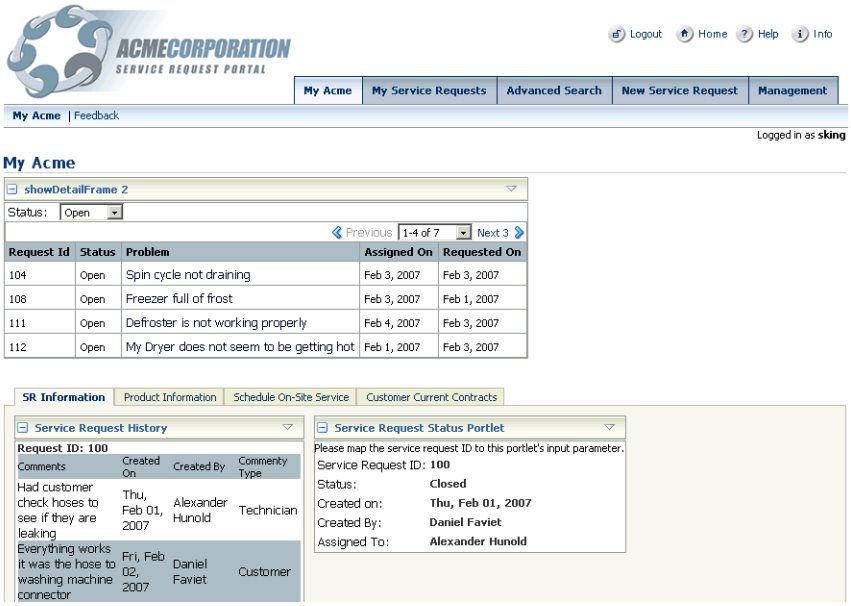
この章では、ADF Faces ベースのログイン・ページを構築し、そのログイン・ページを使用するように SRDemo アプリケーションを構成する方法について学びました。この時点で、管理者が実行時にログイン・ページをカスタマイズできるようにすることができます。この種のカスタマイズを可能にする手順は、第 9 章「サイト管理ページの構築」を参照してください。

7

ページの構築とコンポーネントの追加

この章では、ポートレット、ADF Faces コンポーネントおよびページ上のこれらのアイテム間のインタラクティビティで構成されている新しいページ（図 7-1 を参照）をアプリケーション用に構築する方法について説明します。

図 7-1 1 つに連結されたポートレットおよび ADF コンポーネントで構成されている「My Acme」ページ



注意： このページ上に要素を作成するには、第 2 章「ポートレットの構築」で説明した手順に従ってプロデューサを登録し、第 4 章「コンテンツ・リポジトリの設定」の説明に従ってコンテンツ・リポジトリを設定しておく必要があります。

この章の内容は次のとおりです。

- 手順 1: 「MyAcme」ページの作成
- 手順 2: OmniPortlet プロデューサの登録
- 手順 3: カスタマイズ・コンポーネントの追加
- 手順 4: ADF 表への SelectOneChoice の連結
- 手順 5: JavaServer Faces ドロップダウン・コンポーネントの追加
- 手順 6: Service Request History ポートレットの追加
- 手順 7: Service Request Status ポートレットの追加
- 手順 8: ポートレットへの表の連結
- 手順 9: Product Details ポートレットの追加
- 手順 10: フォルダの内容の表示
- 手順 11: Schedule On-Site Services ポートレットの追加
- 手順 12: ユーザーに基づいた最新契約の追加
- 手順 13: 「My Acme」ページへのセキュリティの適用
- 手順 14: コンポーネントへのセキュリティの適用

手順 1: 「MyAcme」ページの作成

コンポーネントを追加するページの構築を始めます。この項では、ページを作成し、サンプル・ファイルに用意されているテンプレートを適用します。

1. アプリケーション・ナビゲータで、必要に応じて、「UserInterface」プロジェクトの**「Web コンテンツ」**フォルダを開きます。
2. **「app」**フォルダを右クリックし、**「新規」**を選択します。
3. 新規ギャラリで、**「Web Tier」**ノードを開きます。
4. **「JSF」**を選択します。
5. 「項目」リストで、**「JSF JSP」**を選択します。
6. **「OK」**をクリックして「JSF JSP の作成」ダイアログ・ボックスを表示します。
7. ウィザードの「ようこそ」ページが表示されたら、**「次へ」**をクリックして「JSP ファイル」ページを表示します。
8. 「ファイル名」フィールドに、`SRMyAcme` と入力します。

9. 「JSP ドキュメント (*.jspx)」を選択します（図 7-2 を参照）。

図 7-2 JSF JSP の作成

10. **「次へ」**をクリックして「コンポーネント・バインディング」ページを表示します。

11. 「コンポーネント・バインディング」ページで、**「新規マネージド Bean での UI コンポーネントの自動公開」**オプションを選択します（図 7-3 を参照）。

図 7-3 JSF JSP の作成のコンポーネント・バインディング

12. **「次へ」**をクリックして「タグ・ライブラリ」ページを表示します。

13. 「フィルタ方法」を「プロジェクト・テクノロジ」にし、「使用可能なライブラリ」から「選択済のライブラリ」にすべてのライブラリを移動します（図 7-4 を参照）。

図 7-4 JSF JSP の作成のタグ・ライブラリのリスト

14. **「終了」**をクリックしてページを作成します。
15. 「WEB-INF」→「template」の順にフォルダを開きます。
16. **「SRDemoTemplate.jspx」**を右クリックし、**「開く」**を選択します。
17. SRDemoTemplate.jspx ファイルのソース・コードを選択してコピーし、新しい SRMyAcme.jspx ページの既存のコードを置き換えます。
18. SRMyAcme.jspx ページのソースで次の行を探します。

    ```
    <af:panelPage title="#{res['_PAGE_NAME_.pageTitle']}">
    ```

    この行を次の行で置き換えます。

    ```
    <af:panelPage title="#{res['SRMyAcmePage.pageTitle']}">
    ```

19. ファイルを保存します。
20. UIResources.properties ファイルを開きます。このファイルは、**「UserInterface」**→**「アプリケーション・ソース」**→**「oracle.srdemo.view」**→**「resources」**の下にあります。
21. ファイルの最後に、次の新規エントリを追加します。

    ```
    # ======================================================================
    # Resource Bundle Strings for new WebCenter version of SRDemo
    # ======================================================================

    # WC_SRMyAcmePage
    SRMyAcmePage.pageTitle=My Acme
    ```

22. ファイルを保存します。

手順 2: OmniPortlet プロデューサの登録

この章の後の手順で、「My Acme」ページに対して Service Request History ポートレットを作成します。OmniPortlet を使用してこのポートレットを構築しますが、そのためには、まず、OmniPortlet プロデューサを登録する必要があります。また、第 2 章「ポートレットの構築」で作成した PDK-Java ポートレット（SRStatus）をページに追加しますが、第 2 章の手順に従った場合はすでに SRStatus ポートレットはデプロイおよび登録されています。

注意： プロデューサの登録の詳細は、『Oracle WebCenter Framework 開発者ガイド』の第 3 章「ページへの移入」を参照してください。

OmniPortlet プロデューサを登録するには、次のようにします。

1. アプリケーション・ナビゲータで、**「UserInterface」**を右クリックし、ポップアップ・メニューから**「新規」**を選択します。
2. 新規ギャラリの「カテゴリ」で、**「Web Tier」**ノードを開き、**「Portlets」**を選択します。
3. 「項目」で、**「Oracle PDK-Java プロデューサ登録」**を選択し、**「OK」**をクリックします。
4. 「Oracle PDK ポートレット・プロデューサの登録」ウィザードのステップ 1/3 で、名前に `OmniPortlet Producer` と入力し、**「次へ」**をクリックします。
5. 「URL エンドポイント」フィールドに、例 7-1 のような OmniPortlet プロデューサの URL を入力します。

例 7-1 OmniPortlet プロデューサの URL エンドポイントのサンプル

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

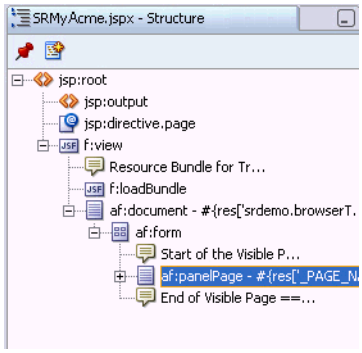
6. **「次へ」**をクリックします。
7. 作業環境に応じて残りのオプションを入力し、**「終了」**をクリックします。

手順 3: カスタマイズ・コンポーネントの追加

ポートレットおよびコンテンツを `SRMyAcme.jspx` ページに組み込むには、特定のカスタマイズ可能レイアウト・コンポーネントを追加して、これらのポートレットおよびコンテンツをユーザーに表示する方法を定義できるようにする必要があります。

`SRMyAcme.jspx` ページの「構造」ペインには、`af:panelPage` コンポーネントが表示されます（図 7-5 を参照）。

図 7-5 PanelPage コンポーネント



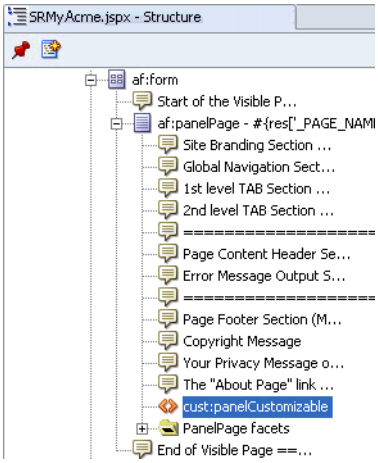
次のコンポーネントを、この PanelPage コンポーネント内に追加します。

1. PanelCustomizable コンポーネント

   PanelCustomizable コンポーネントを PanelPage コンポーネント内に追加して、子コンポーネントを表示 / 非表示にできるようにします。

   そのためには、「カスタマイズ可能コンポーネント・コア」ライブラリから「**PanelCustomizable**」を選択し、「構造」ペインの af:panelPage コンポーネントの上にドラッグします。Layout を horizontal に設定します。図 7-6 は、「構造」ペインでの cust:panelCustomizable コンポーネントの位置を示しています。

図 7-6 PanelCustomizable コンポーネント

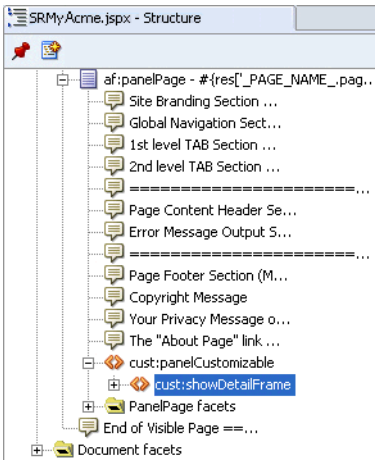


2. ShowDetailFrame コンポーネント

   ShowDetailFrame コンポーネントを PanelCustomizable コンポーネント内に追加して、コンテンツの表示レベルのカスタマイズができるようにします。たとえば、コンテンツの表示を移動、最小化または最大化するオプションを提供します。

   そのためには、「カスタマイズ可能コンポーネント・コア」ライブラリから「**ShowDetailFrame**」を選択し、「構造」ペインの cust:panelCustomizable コンポーネントの上にドラッグします。Text を My Service Requests に設定します。図 7-7 は、「構造」ペインでの cust:showDetailFrame を示しています。

図 7-7 ShowDetailFrame コンポーネント



3. `SelectOneChoice` コンポーネント

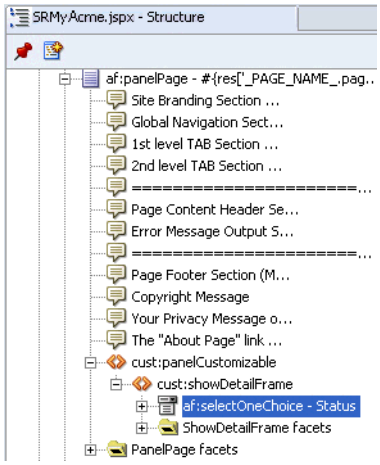
   `SelectOneChoice` コンポーネントをこの `ShowDetailFrame` コンポーネント内に追加して、表示する必要があるサービス・リクエストのステータスを選択できるようにします。

   そのためには、「ADF Faces Core」ライブラリから**「SelectOneChoice」**を選択し、「構造」ペインの `cust:showDetailFrame` コンポーネントの上にドラッグします。表示された「SelectOneChoice の挿入」ダイアログ・ボックスで、該当するタブをクリックし、いくつかの属性の値を次のように設定します。

   - `Label: Status`
   - `Value: Open`
   - `AutoSubmit: true`
   - `Binding: #{backing_app_SRMyAcme.statusPicker}`
   - `Id: statusPicker`

   図 7-8 は、「構造」ペインでの `af:selectOneChoice` コンポーネントを示しています。

図 7-8 SelectOneChoice コンポーネント



4. `SelectItem` コンポーネント

   3 つの `SelectItem` コンポーネントを `SelectOneChoice` コンポーネント内に追加して、`SelectOneChoice` コンポーネントの下にオプションとして表示されるコンポーネントを組み込みます。

   `SelectItem` コンポーネントを追加するには、「ADF Faces Core」ライブラリから**「SelectItem」**を選択し、「構造」ペインの `af:selectOneChoice` コンポーネントの上にドラッグします。

   新しいコンポーネントごとに、いくつかの属性の値を次のように設定します。

   1 つ目の `SelectItem` コンポーネント:

   - `Label: Open`
   - `Value: Open`
   - `Binding: #{backing_app_SRMyAcme.selectItem1}`
   - `Id: selectItem1`

2 つ目の SelectItem コンポーネント:

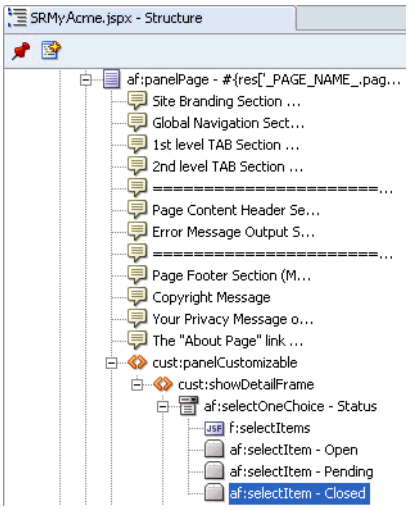
- Label: Pending
- Value: Pending
- Binding: #{backing_app_SRMyAcme.selectItem2}
- Id: selectItem2

3 つ目の SelectItem コンポーネント:

- Label: Closed
- Value: Closed
- Binding: #{backing_app_SRMyAcme.selectItem3}
- Id: selectItem3

図 7-9 は、「構造」ペインでの SelectItem コンポーネントを示しています。

図 7-9 SelectItem コンポーネント

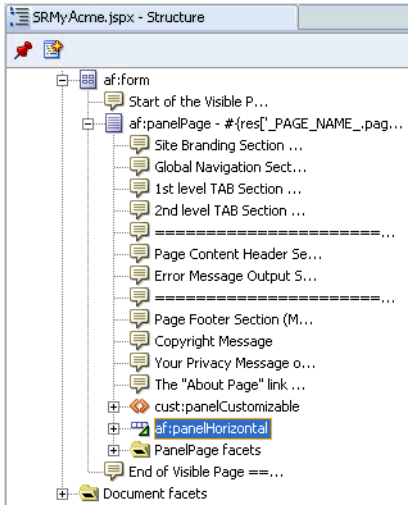


5. `PanelHorizontal` コンポーネント

   `PanelHorizontal` コンポーネントを `PanelPage` コンポーネント内に追加して、子要素を水平に配置します。

   そのためには、「ADF Faces Core」コンポーネント・ライブラリから**「PanelHorizontal」**を選択し、「構造」ペインの `af:panelPage` コンポーネントの上にドラッグします。「構造」ペインの `cust:panelCustomizable` コンポーネントの下に位置するようにします（図 7-10 を参照）。

図 7-10 PanelHorizontal コンポーネント



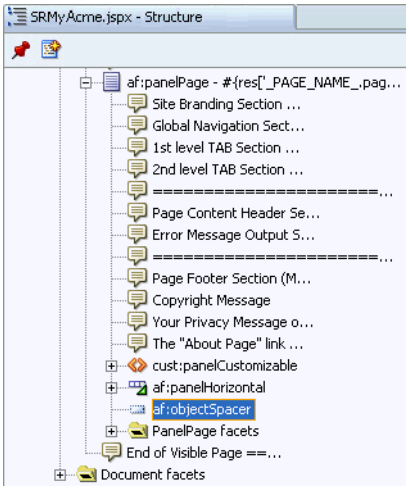
6. `ObjectSpacer` コンポーネント

   `ObjectSpacer` コンポーネント（高さを `30`、幅を `10` に設定）を `PanelHorizontal` コンポーネントの下に追加して、レイアウト内のコンポーネント間に固定幅のスペースを挿入します。

   そのためには、「ADF Faces Core」コンポーネント・ライブラリから**「ObjectSpacer」**を選択し、「構造」ペインの `af:panelPage` コンポーネントの上にドラッグします。「構造」ペインの `af:panelHorizontal` コンポーネントの下に位置するようにします（図 7-11 を参照）。次の属性を設定します。

   - `Height: 30`
   - `Width: 10`

図 7-11 ObjectSpacer コンポーネント



7. `ShowOneTab` コンポーネント

   `ShowOneTab` を `PanelPage` コンポーネント内に追加して、`ShowDetailItem` コンポーネントによって定義される一連のアイテムを組み込みます。

   そのためには、「ADF Faces Core」コンポーネント・ライブラリから**「ShowOneTab」**を選択し、「構造」ペインの `af:panelPage` コンポーネントの上にドラッグします。「構造」ペインの `af:objectSpacer` コンポーネントの下に位置するようにします。次に、ShowOneTab を選択し、`Position` プロパティを `above` に設定します。

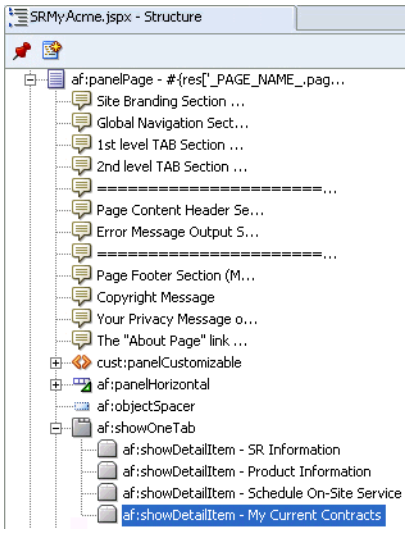
8. `ShowDetailItem` コンポーネント

   4 つの `ShowDetailItem` コンポーネントを `ShowOneTab` コンポーネント内に追加して、`ShowOneTab` コンポーネント内にタブとして表示されるコンポーネントを組み込みます。

   これらの `ShowDetailItem` コンポーネントの `Text` プロパティを、`SR Information`、`Product Information`、`Schedule On-Site Service`、`My Current Contracts` にそれぞれ設定します。後で、4 つのコンポーネントをこれらの `ShowDetailItem` コンポーネント内に追加します。

ShowDetailItem を追加するには、「ADF Faces Core」コンポーネント・ライブラリから**「ShowDetailItem」**を選択し、「構造」ペインの af:showOneTab コンポーネントの上にドラッグします。図 7-12 は、「構造」ペインでの 4 つの af:showDetailItem 子コンポーネントがある af:showOneTab コンポーネントを示しています。

図 7-12 ShowOneTab コンポーネント



9. PanelHorizontal コンポーネント

PanelHorizontal コンポーネントを、SR Information という ShowDetailItem コンポーネント内に追加して、子要素を水平に配置します。

そのためには、「ADF Faces Core」コンポーネント・ライブラリから**「PanelHorizontal」**を選択し、「構造」ペインの cust:showDetailItem component - SR Information コンポーネントの上にドラッグします。

後で、OmniPortlet および Oracle Java PDK ポートレットをこのコンポーネント内に追加し、サービス・リクエストの履歴およびステータスをそれぞれ表示できるようにします。

10. この項の前述の手順を実行して、PanelCustomizable コンポーネントを、Product Information という ShowDetailItem コンポーネント内に追加します。Layout を horizontal に設定します。図 7-13 は、「構造」ペインでの新しい cust:panelCustomizable コンポーネントを示しています。

図 7-13 ShowDetailItem コンポーネント群

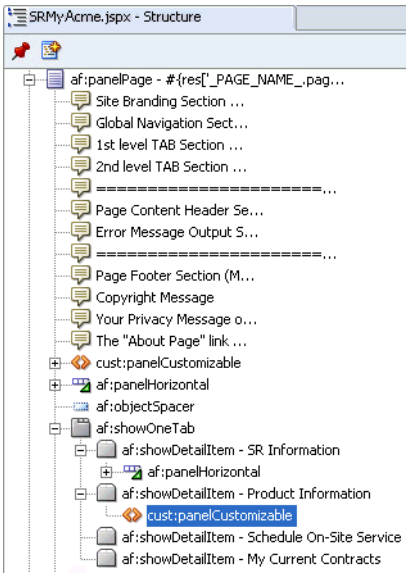
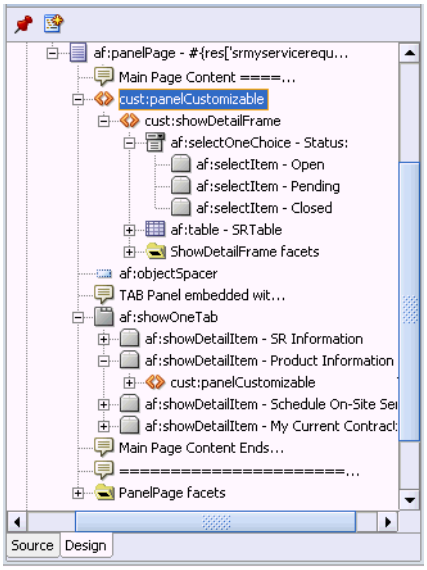


図 7-14 に、これらのコンポーネントが SRMyAcme.jspx の「構造」ペインにどのように表示されるかを示します。

図 7-14 SRMyAcme.jspx ページ内のレイアウト・コンポーネントを表示する「構造」ペイン



この章の次の手順では、これらのカスタマイズ・コンポーネント内にポートレットおよびコンテンツを追加する方法について説明します。

手順 4: ADF 表への SelectOneChoice の連結

特定のステータスであるサービス・リクエストのリストを表示するには、TopLink および EJB セッション Bean に基づいた ADF 読取り専用表を追加する必要があります。ユーザーがこの表のサービス・リクエスト番号をクリックすると、そのサービス・リクエストの履歴およびステータスがページ下部のポートレットに表示されます。

サービス・リクエストを表示する ADF 表を追加するには、次の手順を実行します。

1. データ・コントロール・パレットで、**findServiceRequestByStatus** を探して抽出します。

   ヒント： データ・コントロール・パレットが表示されない場合は、「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。

2. 「構造」ペインで、**ServiceRequest** を（前の項で追加した）`showDetailFrame` コンポーネントの上にドラッグします。
3. ポップアップ・メニューから、**「ADF 読取り専用表」**を選択します。アクション・バインディング・エディタが表示されます。クライアント・ライブラリが追加されていることに関する注意が表示された場合は、**「OK」**をクリックします。
4. `statusParam` パラメータの「値」フィールドに、`${"Open"}` と入力します。

   これで、サービス・リクエストのステータスは Open とハードコーディングされます。後で、動的な値を指定します。

5. **「OK」**をクリックします。

   「表の列の編集」ダイアログ・ボックスが表示されます。

6. 再配置および削除して、次のように列を配置します。
   - svrId
   - status
   - problemDescription
   - assignedDate
   - requestDate
7. **「ソートの有効化」**を選択します。
8. **「OK」**をクリックします。
9. **「SRMyAcme.jspx」**を右クリックし、**「実行」**を選択します。

   すべてのオープン・サービス・リクエストが表に表示されます。

10. 表示される行数を減らすには、ページ定義に移動して**太字**で示す変更を行います。

例 7-2 表示される行数に関するページ定義の編集

```
<methodIterator id="findServiceRequestByStatusIter"
                    Binds="findServiceRequestByStatus.result"
                    DataControl="SRPublicFacade" RangeSize="4"
                    BeanClass="oracle.srdemo.model.entities.ServiceRequest"/>
```

11. ページを再実行して、変更を確認します。

手順 5: JavaServer Faces ドロップダウン・コンポーネントの追加

前の項で、ステータスを Open とハードコーディングしました。その結果、オープン・サービス・リクエストのリストの表示しかできません。今度は、ハードコーディングされたこの値ではなく、エンド・ユーザーがサービス・リクエストのステータスをリストから選択できるようにするコンポーネントを追加します。この項では、ステータスの選択に使用できるリストを追加し、選択されたステータスであるサービス・リクエストをすべて表示する方法を示します。そのためには、次の手順を実行します。

1. アプリケーション・ナビゲータで、**「アプリケーション・ソース」**フォルダを開きます。
2. **「oracle.srdemo.view」**→**「pageDefs」**の順に開き、**「app_SRMyAcmePageDef.xml」**を選択します。
3. 「構造」ペインで、**「bindings」**ノードを開きます。
4. **「findServiceRequestByStatus」**ノードを開きます。
5. **「statusParam」**をダブルクリックして「NamedData のプロパティ」ダイアログ・ボックスを表示します。
6. 「NDValue」フィールドで、既存の値を次の値で置き換えます。

   ```
   ${(backing_app_SRMyAcme.statusPicker.value == null) ? "Open" :
   backing_app_SRMyAcme.statusPicker.value}
   ```

7. 常に statusPicker の最新の選択が表に反映されるようにするには、表の `partialTrigger` プロパティを `statusPicker` に設定します。そのためには、ページ上の表コンポーネントを選択します。

   リストから、**「statusPicker」**を選択します。
8. プロパティ・インスペクタで、表の Partial Trigger プロパティを探します。
9. **「編集」**ボタンをクリックして Partial Trigger ウィンドウを表示します。
10. **「新規」**をクリックし、リストから**「statusPicker」**を選択します。
11. ページを実行して、変更を確認します。
12. リストからステータスを選択し、表が変化する様子を確認します。

手順 6: Service Request History ポートレットの追加

この項の手順では、OmniPortlet を使用して、サービス・リクエストの履歴を表示するポートレットを作成する方法を示します。OmniPortlet を使用して、サービス・リクエストの履歴を表示するポートレットを作成するには、次の手順を実行します。

1. 「手順 2: OmniPortlet プロデューサの登録」の説明に従って、必ず OmniPortlet プロデューサを登録しておきます。
2. コンポーネント・パレットから、**「OmniPortlet Producer」**を選択し、**「OmniPortlet」**を選択します。このコンポーネントを「構造」ペインの「`panelHorizontal`」コンポーネント（「SR Information」タブ上にある）の上にドラッグします。
3. ページを実行して、ブラウザに表示します。

4. ブラウザで、OmniPortlet の**「定義」**をクリックして定義を変更します。
   a. 「タイプ」ページで、**「SQL」**を選択し、**「次へ」**をクリックします。
   b. 「ソース」タブで、例 7-3 に示すような問合せを入力します。

例 7-3 Service Request History ポートレット用の SQL 文

```
select svr_id, to_char(svh_date, 'Dy, Mon DD, YYYY') created_on, notes, svh_type,
users.first_name || ' ' || users.last_name createdby
from service_histories, users
where svr_id = substr ('##Param1##', 1, 3)
and users.user_id = service_histories.created_by
order by line_no
```

   c. 接続を編集し、データベースへの接続を定義します。
   d. **Param1** のデフォルト値として 104 を指定し、「表示」タブが表示されるまで**「次へ」**をクリックします。
   e. 「表示」タブで、次の変更を行います。
      – **タイトル** : Service Request History
      – **ヘッダー・テキスト** : Request ID: ##Param1##
      – **レイアウト・スタイル** : 表
   f. 「レイアウト」タブで、「表スタイル」を**「代替」**に設定し、表 7-1 に示すように「列のレイアウト」を指定します。

表 7-1 Service Request History OmniPortlet 用のレイアウト設定

列ラベル	列	表示内容
ID	SVR_ID	Hidden
Comments	NOTES	Text
Created by	CREATEDBY	Text
Created on	CREATED_ON	Text
Comment type	SVH_TYPE	Text

5. **「終了」**をクリックします。

注意 : OmniPortlet の詳細は、『Oracle WebCenter Framework 開発者ガイド』の第 13 章「OmniPortlet を使用したポートレットの作成」および『Oracle WebCenter Framework チュートリアル』の OmniPortlet を構築するための手順を参照してください。

手順 7: Service Request Status ポートレットの追加

第 2 章「ポートレットの構築」で、サービス・リクエスト ID を指定すると、特定のサービス・リクエストの現行ステータスが表示されるポートレットを作成しました。この項では、そのポートレットを「MyAcme」ページに追加します。

Service Request Status ポートレットをページに追加するには、次のようにします。

1. 第 2 章「ポートレットの構築」の説明に従って、必ず PDK-Java プロデューサを登録しておきます。

   ヒント： 第 2 章「ポートレットの構築」で登録したプロデューサは SRStatusProducer です。このプロデューサは、`http://host:port/SRStatusPortlet/providers` にあります。

2. コンポーネント・パレットで、リストから**「SRStatusProducer」**を選択します。
3. **Service Request Status Portlet** を、「`panelHorizontal`」コンポーネントのすでに追加した Service Request History Portlet の直下にドラッグします。
4. Service Request Status ポートレットでは、サービス・リクエスト ID を格納したパラメータを受け取ることを想定しています。作成中のページでは、ポートレットに、選択されたサービス・リクエストのステータスをページ上部の表に表示させます。サービス・リクエスト ID をポートレットに引き渡す手順は、「手順 8: ポートレットへの表の連結」を参照してください。

手順 8: ポートレットへの表の連結

コンポーネントおよびポートレットをすべてページに追加したので、1 つに連結してページ上にインタラクティビティを作成します。この項では、顧客または技術者（あるいはその両方）がページ上の表から選択した各サービス・リクエストのステータスおよび履歴を表示できるように、ページを設定します。また、部分ページ・リフレッシュを使用してコンポーネントを 1 つに連結し、ページ全体をリフレッシュしなくてもエンド・ユーザーがその情報を表示できるようにします。

この手順では、次の連結を実行します。

- 表への OmniPortlet の連結
- Service Request 表への Service Request Status ポートレットと Service Request History ポートレットの連結

表への OmniPortlet の連結

OmniPortlet は、これまでの手順で行ったカスタマイズに基づいてサービス・リクエストの履歴を表示できますが、ここでは、ユーザーが表で行った選択に基づいてポートレットに履歴を表示させます。OmniPortlet は、表から情報を受け取り、その情報を使用して特定のサービス・リクエストの履歴を表示できます。

1. 「手順 4: ADF 表への SelectOneChoice の連結」で作成した表で、commandLink を表の problemDescription 列に追加します。

   コンポーネント・パレットで、「ADF Faces Core」の下の**「commandLink」**を選択し、「構造」ペインの 3 つ目の af 列内の outputText の上にドラッグします。

2. この outputText を削除します。

3. commandLink のプロパティを次のように設定します。

   - Text: #{row.problemDescription}
   - Id: commandLink1
   - Action: #{backing_app_SRMyAcme.commandLink1_action}

4. 例 7-4 に示すコードを commandLink1_action() メソッドとして入力します。このメソッドは、**「アプリケーション・ソース」**→**「oracle.srdemo.view」**→**「backing.app.SRMyAcme.java」**の下にある backing.app.SRMyServiceRequests.java クラス内にあります。

例 7-4 commandLink_action () メソッド

```
public String commandLink1_action() {
        FacesContext context = FacesContext.getCurrentInstance();
        context.getExternalContext().getSessionMap().put("serviceID",
(Integer)outputText1.getValue());
        return "setCurrentRowWithKey";
    }
```

注意： outputText1 が未登録の場合（Oracle JDeveloper により下線表示されます）は、表の svrId outputText 列に outputText1 という ID があることを確認します。次に、バインディングが id="outputText1" binding="#{backing_app_SRMyAcme.outputText1 と定義されていることを確認します。

---

5. 「import」セクションを開き、次の行を Java クラスの先頭に追加します。

   ```
   import javax.faces.context.FacesContext;
   ```

6. 「構造」ペインで、ページ定義に移動し、**「executables」**→**「variables」**の順に移動して**「OmniPortlet1_1_Param1」**をクリックします。

7. プロパティ・インスペクタで、**DefaultValue** プロパティを次の値に設定します。

   ```
   ${sessionScope.serviceID}
   ```

8. サービス ID を受け取るための OmniPortlet 用のページ・パラメータを作成します。ページ定義の先頭には、次のタグがあります。

   ```
   <parameters/>
   ```

   このタグを次のコードで置き換え、serviceID というページ・パラメータを設定します。

   ```
   <parameters>
   <parameter id="pageParam" value="${sessionScope.serviceID}"/>
   </parameters>
   ```

9. ページを実行してポートレット・ワイヤリングをテストします。
10. 表の Problem 列のハイパーリンクをクリックして、テストします。ポートレットは、リンクをクリックするたびに更新されます。

Service Request 表への Service Request Status ポートレットと Service Request History ポートレットの連結

Service Request Status ポートレットと Service Request History ポートレットではいずれも、サービス・リクエスト ID をパラメータとして受け取ることを想定しています。そして、その ID で識別される特定のサービス・リクエストのステータスまたは履歴を表示します。サービス・リクエスト ID が、ページ上部の表で選択されたサービス・リクエストの ID になるように設定します。

Service Request ポートレットを Service Request 表に連結するには、次のようにします。

1. ページ定義に移動し、「構造」ペインで**「executables」→「variables」**の順に開き、**「OmniPortlet1_1_Param1」**をクリックします。
2. プロパティ・インスペクタで、**DefaultValue** プロパティを `${sessionScope.serviceID}` に設定します。
3. 「構造」ペインで、変数 **ServiceRequestStatusPortlet1_1_ServiceReqID** を探して選択します。

   ヒント： この変数は、**「executables」→「variables」→「ServiceRequestStatusPortlet1_1_ServiceReqID」**の順にノードを開くと見つかります。

4. プロパティ・インスペクタで、**DefaultValue** プロパティを `${sessionScope.serviceID}` に設定します。
5. ページを保存し、実行してブラウザに表示します。これで、表のサービス・リクエストをクリックしてポートレットをテストできます。それに応じて、Service Request History ポートレットと Service Request Status ポートレットは更新されます。

手順 9: Product Details ポートレットの追加

この項の手順では、JSR 168 ポートレットをページ上に表示する方法を示します。ここでは、第 2 章「ポートレットの構築」で作成した Product Details ポートレットを追加します。「Product Information」タブの `panelCustomizable` コンポーネント内に、次のとおり追加します。

1. 第 2 章「ポートレットの構築」の説明に従って、必ず PDK-Java プロデューサを登録しておきます。

   ヒント： 第 2 章「ポートレットの構築」で登録したプロデューサは ProductDetailsPortlet です。このプロデューサは、`http://host:port/ProductDetailsPortlet/portlets/wsrp1?WSDL` にあります。

2. コンポーネント・パレットで、リストから**「ProductDetailPortlet」**を選択します。
3. **「Product Details Portlet」**を `panelHorizontal` コンポーネントの上にドラッグします。
4. ページを保存し、実行してブラウザに表示して「Product Information」タブ上のポートレットをテストします。

手順 10: フォルダの内容の表示

この項の手順では、フォルダの内容をページ上に表示する方法を示します。「Product Information」タブの panelCustomizable コンポーネント内に、次のとおり追加します。

1. データ・コントロール・パレットで、**「SRContentRepository」**を開きます。このデータ・コントロールは、第 4 章「コンテンツ・リポジトリの設定」で作成しています。

   ヒント： データ・コントロール・パレットが表示されない場合は、「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。

2. **「getItems(String, String)」**を開きます。
3. panelBox を panelCustomizable に追加します。
4. **「Return」**ノードを panelBox にドラッグします。
5. メニューから**「ツリー」**、続いて**「ADF ツリー」**を選択します。

   アクション・バインディング・エディタが表示されます。

6. path パラメータの「値」フィールドに、/manuals と入力します。
7. type パラメータの「値」フィールドは空のままにしておきます。
8. **「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。

9. ツリー・バインディング・エディタで、右上のリストから**「Name」**、「URI」および**「Primary Type」**を選択します。
10. 「ブランチ・ルール・アクセッサ」リストで、**「Items」**を選択します。
11. **「新しい規則を追加」**をクリックします。
12. **「OK」**をクリックします。
13. **「OK」**をクリックします。
14. ページ・ソースに、次のようなコードが表示されます。

```
<af:panelBox>
<af:tree value="#{bindings.getItems1.treeModel}" var="node">
<f:facet name="nodeStamp">
<af:outputText value="#{node}"/>
</f:facet>
</af:tree>
</af:panelBox>
```

15. コンポーネント・パレットから、**「ADF Faces Core」**を選択します。
16. **「Switcher」**コンポーネントを <f:facet> タグの中にドラッグします。
17. facetName 属性を #{node.currentRow.dataProvider.primaryType} に設定します。
18. nt:file というファセットを 1 つと、nt:folder というファセットを 1 つ作成します。そのためには、**「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。
19. nt:folder には、生成された出力テキストを使用します。

    注意： 出力テキストが自動的に生成されない場合は、value 属性を設定するのではなく、出力テキストをファセット内に追加する必要があります。

20. nt:file について、goLink および objectImage を作成します。

21. switcher のコードは、例 7-5 のようになります。

例 7-5 switcher のコード

```
<af:switcher facetName="#{node.primaryType}">
<f:facet name="nt:file">
<h:panelGroup>
<af:objectImage source="/images/file.gif"/>
<af:goLink text="#{node.name}" destination="#{node.URI}" targetFrame="_blank"/>
</h:panelGroup>
</f:facet>
<f:facet name="nt:folder">
<af:outputText value="#{node.name}"/>
</f:facet>
</af:switcher>
```

22. `index.html` を実行します。
23. 「**Product Information**」タブをクリックします。
24. 表のリンクの 1 つをクリックします。
25. 製品カテゴリの 1 つを開いて、様々なモデルを表示します。
26. モデルの 1 つを開いて、そのモデルのマニュアルのリストを表示します。
27. マニュアルを 1 つ選択して表示します。

手順 11: Schedule On-Site Services ポートレットの追加

Schedule On-Site Services ポートレットを使用すると、アプリケーションでサービス・リクエストを作成した後、製品についてオンサイト・サービスをスケジュールできます。リクエストのサービス ID を指定し、適当な日時または自分で選んだ技術者を選択して、技術者が指定の場所に来て、問題に対処するようにスケジュールできます。

この項の内容は次のとおりです。

- Web クリッピング・プロデューサの登録
- Web クリッピング・ポートレットの追加
- Web クリッピング・ポートレットに表示する Web ページの選択

Web クリッピング・プロデューサの登録

Web クリッピング・プロデューサを登録するには、次のようにします。

1. アプリケーション・ナビゲータで、プロデューサを作成する「**SRDemoSample_Starter**」を右クリックし、ポップアップ・メニューから「**新規**」を選択します。
2. 新規ギャラリの「カテゴリ」で、「**Web Tier**」ノードを開き、「**Portlets**」を選択します。
3. 新規ギャラリの「項目」で、「**Oracle PDK-Java プロデューサ登録**」を選択します。
4. 「**OK**」をクリックします。
5. 「ようこそ」ページで、「**次へ**」をクリックします。

   必要に応じて、「次へ」をクリックする前に「**次回にこのページを表示しない**」を選択し、このウィザードの次回以降の使用時に「ようこそ」ページが表示されないようにします。「ようこそ」ページは、ウィザードの前回使用時にスキップするオプションが選択されると、表示されません。
6. 「**名前**」フィールドに、Web クリッピング・プロデューサの名前として `WebClipping Producer` と入力します。

   この名前は、アプリケーション内で一意である必要があります。文字、数字およびアンダースコアのみを使用してください。

7. **「次へ」**をクリックします。
8. **「URL エンドポイント」**フィールドに、例 7-6 に示すような Web クリッピング・プロデューサの URL を入力します。

例 7-6 Web クリッピング・プロデューサの URL エンドポイントのサンプル

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

9. **「終了」**をクリックし、PDK-Java ポートレット・プロデューサの登録を完了します。

Web クリッピング・ポートレットの追加

オンサイト Web クリッピング・サービスを追加するには、次の手順を実行します。

1. SRMyAcme.jspx ファイルを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「SRMyAcme.jspx」ファイルを右クリックし、ポップアップ・メニューから**「開く」**を選択します。
2. コンポーネント・パレットから、**「WebClipping Producer」**を選択します。
3. **「WebClippingPortlet」**を選択し、「構造」ペインの「Schedule On-Site Service」という「showDetailItem」コンポーネントの上にドラッグします。
4. 変更内容を保存します。
5. 「SRMyAcme.jspx」ファイルを右クリックし、ポップアップ・メニューから**「実行」**を選択します。

   埋込み OC4J サーバーが起動し、デフォルトのブラウザが起動して Web クリッピング・ポートレットが表示されます。結果のページで、WebCenter アプリケーションで公開する Web ページを選択します。次に、Web クリッピング・スタジオを使用して、組み込むための Web ページのセクションを選択します。

Web クリッピング・ポートレットに表示する Web ページの選択

この項では、ACME Technician Assignment System 外部アプリケーションからの情報を Web クリッピング・ポートレットに表示する方法を示します。この外部アプリケーションは、OC4J インスタンスにデプロイされ、SRDemo アプリケーションの一部として使用できます。この外部アプリケーションを使用して、製品に対するオンサイト・サービスをリクエストできます。

この項の手順を実行する前に、ページに対して適切な権限を付与する必要があります。権限の付与の手順は、「手順 13: 「My Acme」ページへのセキュリティの適用」を参照してください。

ACME Technician Assignment System アプリケーションを Web クリッピング・ポートレットに表示するには、次の手順を実行します。

1. Web クリッピング・ポートレットのヘッダーにある**「アクション」**アイコンをクリックし、**「カスタマイズ」**を選択してすべてのユーザーが使用できる Web ページを選択します。

   「Web クリッピングの検索」ページが表示されます。
2. **「URL ロケーション」**フィールドに、次の URL を入力します。

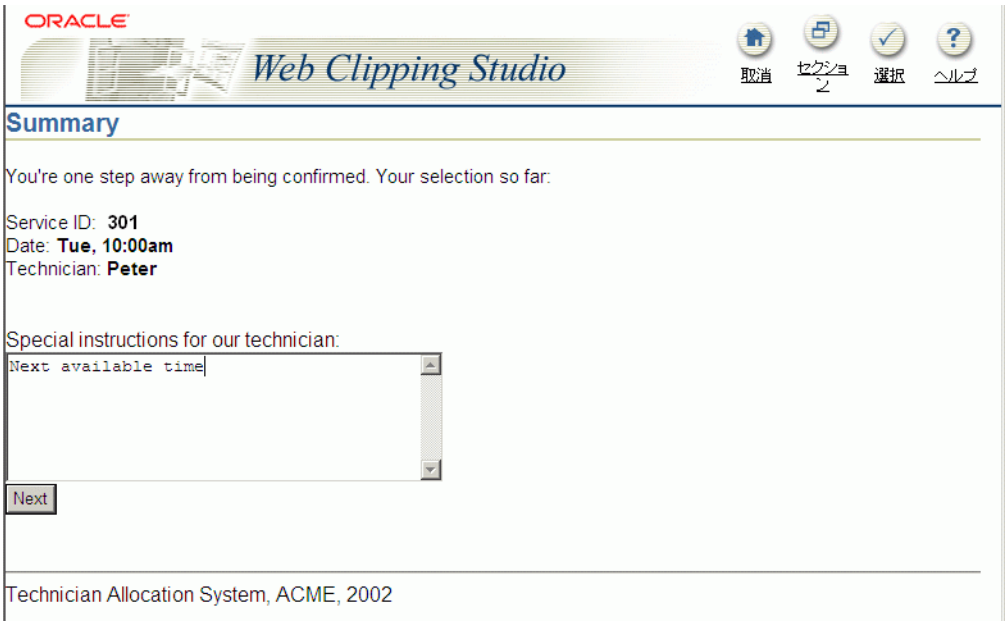
   ```
   http://host:port/ExternalApp/preConfirmation.jsp?date=Tue,%2010:00am&technician=Peter
   ```

   SRDemo アプリケーションをダウンロードしたシステムのホスト名およびポート番号を指定します。

3. **「開始」**をクリックします。

   指定したページが Web クリッピング・スタジオに表示されます（図 7-15 を参照）。

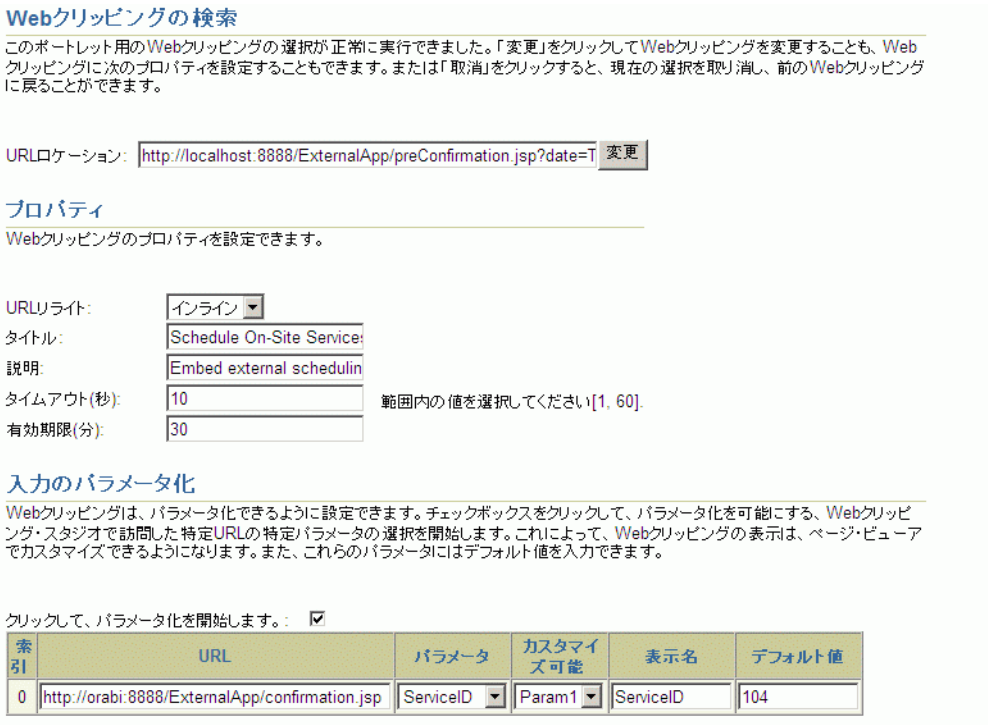
図 7-15 スタンドアロン・アプリケーションがクリップされている Web クリッピング・ポートレット



4. コメントとして `Next Available Time` と指定し、**「次へ」**をクリックします。
5. Web クリッピング・スタジオのバナーにある**「選択」**をクリックします。Web クリッピング・スタジオに、クリッピングのプロパティが表示された「Web クリッピングの検索」ページが表示されます。
6. 「Web クリッピングの検索」ページ（図 7-16）で、次のように指定します。
   - タイトル：`Schedule On-Site Services`
   - 説明：`Embed external scheduling application with the SR application`
   - URL リライト：インライン

注意： 外部アプリケーションと統合した場合またはクリップされたサイトにログインしている場合で、「URL リライト」に「インライン」を選択すると、クリップされたサイトへのセッションは参照中保持されます。

図 7-16「Web クリッピングの検索」ページ



7. 「入力のパラメータ化」セクションでは、次の値を指定します。
   - パラメータ : `ServiceID`
   - パーソナライズ可能 : `Param1`
   - 表示名 : `ServiceID`
   - デフォルト値 : `104`
8. 「**OK**」をクリックし、選択した Web クリッピングをページ上の Web クリッピング・ポートレットに表示します。

   図 7-17 に、Web クリッピング・ポートに追加されたコンテンツを示します。

図 7-17 クリップされたコンテンツが表示されている Web クリッピング・ポートレット

このポートレットを使用して、ユーザーは製品に対するオンサイト・サービスをスケジュールできます。

手順 12: ユーザーに基づいた最新契約の追加

この項の手順では、ページに最新契約コンポーネントを追加する方法を示します。

1. panelBox を「My Current Contracts」タブに追加します。
2. データ・コントロール・パレットで、**「SRContentRepository」**を開きます。

   ヒント： データ・コントロール・パレットが表示されない場合は、「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。

3. **「getItems(String, String)」**を開きます。
4. **「Return」**ノードを `panelBox` にドラッグします。
5. メニューから**「表」**、続いて**「ADF 読取り専用表」**を選択します。

   「表の列の編集」ダイアログ・ボックスが表示されます。

6. **name** 以外の列をすべて削除します。
7. **「OK」**をクリックします。
8. ページ定義ファイル（`app_SRMyAcmePageDef.xml`）の `executables` の下で、ID が `getItemster` の **methodIterator** タグを複製し、例 7-7 に示すようにコードを変更します。

例 7-7 app_SRMyAcmePageDef.xml ファイルの最初に変更されたコード

```
<methodIterator id="getContractsIter" Binds="getContracts.result"
DataControl="SRContentRepository"
RangeSize="10" BeanClass="SRContentRepository.getItems_return"/>
```

9. `Bindings` の下で、ID が `getItems` の `<methodAction>` を複製し、例 7-8 に示すようにコードを変更します。

例 7-8 app_SRMyAcmePageDef.xml ファイルの 2 番目に変更されたコード

```
<methodAction id="getContracts" InstanceName="SRContentRepository"
 DataControl="SRContentRepository" MethodName="getItems" RequiresUpdateModel="true"
 Action="999" ReturnName="SRContentRepository.methodResults.SRContentRepository_
 getContracts_result">
  <NamedData NDName="path" NDValue="/contracts/${userInfo.userName}"
 NDType="java.lang.String"/>
    <NamedData NDName="type" NDType="java.lang.String"/>
</methodAction>
```

10. `<table id="getItems2"...` を探し、`<table id="getItems2" IterBinding="getContractsIter">` に変更します。
11. ページのソースを表示します。
12. 表を探し、**inputText** 要素を見つけます。
13. コンポーネント・パレットから、**「ADF Core」**を選択します。
14. **GoLink** コンポーネントを、ページ上の表セル内の name の outputText の横にドラッグします。
15. 式 `#{row.name}` を outputText から **GoLink** の text 属性にコピーします。
16. destination 属性を GoLink に追加し、値を `#{row.URI}` とします。
17. targetFrame 属性を追加し、値を `_blank` とします。
18. この **outputText** 要素を削除します。

19. 次のようにして、アクセス権限を設定します。
    a. ページ定義を開きます。
    b. 「構造」ペインで、「bindings」の下の**「getItems2」**を右クリックします。
    c. ポップアップ・メニューから、**「認可の編集」**を選択します。
    d. ダイアログ・ボックスで、SRContentRepository.getItems.name およびリストのその他すべての値について、**anyone** のボックスを選択します。
20. **index.html** を実行します。

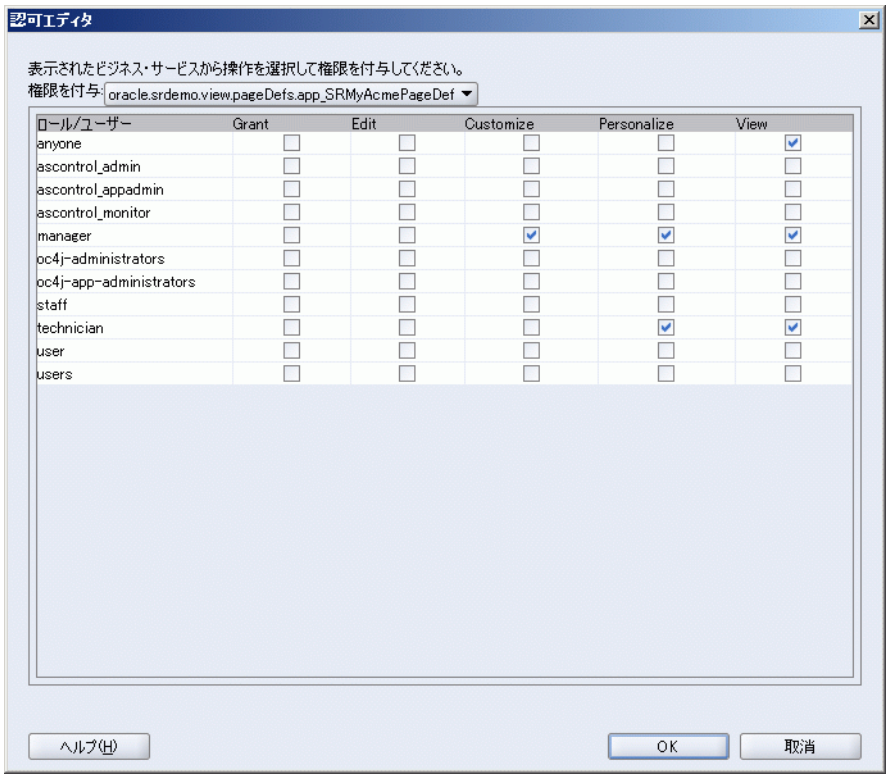
手順 13: 「My Acme」ページへのセキュリティの適用

ページの認可ポリシーは、ページ定義ファイルに対して定義されます。認可エディタは、ローカル・システムの JAZN ファイルからエンタープライズ・ロールを読み取るのに使用します。このエディタを使用すると、特定のリソース（ページ、メソッド、イテレータ、属性など）に対して設定できるアクションを定義できます。このエディタは、ポリシー・ストアに権限を書き込み、様々なロールに対してきめ細やかな JAAS 権限の宣言定義を可能にします。また、疑似ロール anybody に権限を付与するパブリック・ページの定義にも使用されます。

次の手順を実行して、SRMyAcme.jpsx ページに対する権限を設定します。

1. **SRMyAcme.jpsx** ページを右クリックし、**「ページ定義に移動」**を選択します。
2. 構造ウィンドウで、ページ定義を右クリックし、**「認可の編集」**を選択します。認可エディタが表示されます。
3. View 権限を anyone に、Customize 権限を manager に、Personalize 権限を technician にそれぞれ付与します（図 7-18 を参照）。

図 7-18「MyAcme」ページに対して権限を設定するための認可エディタ



手順 14: コンポーネントへのセキュリティの適用

SRMyAcme.jspx ページでは、技術者と管理者のみが「On-Site Service」タブおよび Product Details ポートレットを表示できるようにします。この項では、適切な権限を持つユーザーがページにアクセスしているときにのみレンダリングされるように、これらのコンポーネントを設定する方法について説明します。

「On-Site Service」タブおよび Product Details ポートレットにセキュリティを適用するには、次の手順を実行します。

1. 「SRMyAcme.jspx」ページを開きます。
2. Product Details ポートレットを選択します。
3. プロパティ・インスペクタで、Rendered プロパティを次の値に設定します。

   ```
   #{authNLink.authenticated &&
   bindings.permissionInfo['app_SRMyAcmePageDef'].allowsPersonalize}
   ```

4. 今度は「SRMyAcme.jspx」ページの「On-Site Service」タブを選択します。
5. このタブのプロパティ・インスペクタで、Rendered プロパティを次の値に設定します。

   ```
   #{authNLink.authenticated &&
   bindings.permissionInfo['app_SRMyAcmePageDef'].allowsPersonalize}
   ```

まとめ

この章では、各種コンポーネントをページに追加し、タブおよびサブタブに編成する方法について学びました。また、ページおよび各コンポーネントにセキュリティを適用する方法についても学びました。

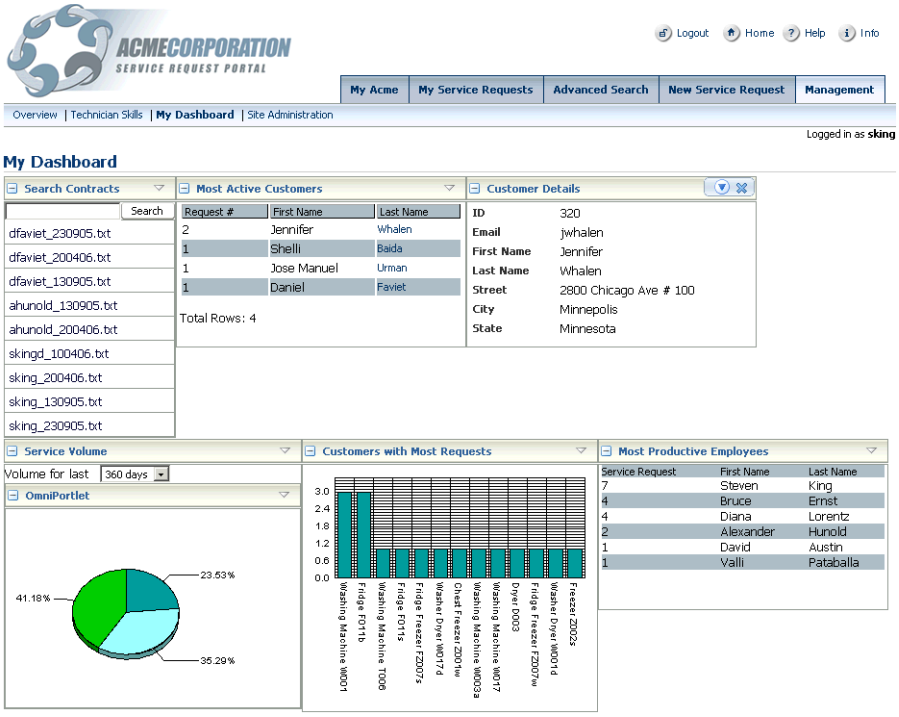
8

ダッシュボード・ページの構築

この章では、WebCenter アプリケーション用のダッシュボード・ページを構築します。ダッシュボード・ページは、ビジネス・アクティビティやビジネス・インテリジェンスに関するメトリックおよび主要なパフォーマンス・インジケータを体系化して表示する、わかりやすいユーザー・インタフェースです。

このサンプル・ダッシュボードでは、エンド・ユーザーはサイトの統計情報の表示、顧客契約の検索および毎日のサービス量の内訳の表示を行うことができます。また、最もアクティブな顧客、顧客の詳細、最も生産性の高い従業員および最もリクエストの多い製品に関する情報も表示できます（図 8-1 を参照）。

図 8-1 ダッシュボード・ページの全体像



この章では、ページを構築し、OmniPortlet を使用してダッシュボード・ページ（図 8-1）を作成する方法を示します。内容は次のとおりです。

- 手順 1: ダッシュボード・ページの作成
- 手順 2: ページ・レイアウトの作成
- 手順 3: ページへの OmniPortlet のインスタンスの追加
- 手順 4: Service Request Volume ポートレット用の SelectOneChoice コンポーネントの追加
- 手順 5: 顧客契約の検索機能の追加
- 手順 6: Most Productive Employees ポートレットの定義
- 手順 7: Service Request Volume ポートレットの定義
- 手順 8: Most Requested Products ポートレットの定義
- 手順 9: Most Active Customers ポートレットの定義
- 手順 10: Customer Details ポートレットの定義
- 手順 11: ページ・コンテンツの連結

手順 1: ダッシュボード・ページの作成

この項では、ポートレットを配置するページを構築して、ダッシュボード・ページを作成する方法を示します。始める前に、必要なサンプル・ファイルがあることを確認し、サンプル・ファイルに含まれる `install.html` ファイルの説明に従って Preconfigured OC4J を必ず初期化しておきます。

1. JSF JSP ページを新規作成します。「JSF JSP の作成」ウィザードで次のオプションを必ず選択してください。

注意： 「JSF JSP の作成」ウィザードにアクセスするには、**「ファイル」**メニューから**「新規」**を選択し、「新規ギャラリ」ダイアログ・ボックスで、**「Web Tier」**の下の**「JSF」**を選択します。

   - ステップ 1/4: ページ名を `SRDashboard.jspx` とし、ディレクトリ・パスを `UserInterface¥public_html¥app¥management¥` に設定し、**「JSP ドキュメント (*.jspx)」**タイプを選択します。
   - ステップ 2/4: **「新規マネージド Bean での UI コンポーネントの自動公開」**オプションを選択し、このページの残りのオプションはデフォルトのままにしておきます。

注意： 新規マネージド Bean の名前は、必ず `app_management_SRDashboard` にします。

   - ステップ 3/4: 次のライブラリが**「選択済のライブラリ」**リストに表示されていることを確認します。
     - JSF Core
     - JSF HTML
     - ADF Faces Components
     - ADF Faces HTML
     - ADF Portlet Components
     - Customizable Components Core

注意： ページの構築および移入の詳細は、『Oracle WebCenter Framework 開発者ガイド』の「ページへの移入」を参照してください。

2. ウィザードを完了してページを作成したら、アプリケーションの**「SRDemoSample_Starter」**、**「UserInterface」**、**「WEB-INF¥template」**の順にフォルダを開きます。
3. **「SRDemoTemplate.jspx」**を右クリックし、**「開く」**を選択します。
4. **「ソース」**タブをクリックし、ソース・コードを `SRDemoTemplate.jspx` ページからコピーして新しい `SRDashboard.jspx` ページのソースに貼り付け、新しいページの既存のコードをすべて置き換えます。
5. `_PAGE_NAME_` を探し、`srdashboard` で置き換えて、次のような `panelPage` タグにします。

   ```
   <af:panelPage title="#{res['srdashboard.pageTitle']}">
   ```

6. 翻訳可能なページ・タイトルを新しいページに指定するには、**「SRDemoSample_Starter」**、**「UserInterface」**、**「アプリケーション・ソース」**、**「oracle.srdemo.view」**、**「resources」**の順に開き、**「UIResources.properties」**ファイルを開きます。
7. `UIResources.properties` ファイルで、次の行をファイルの最後に追加します。

   ```
   #WC_DashboardPage
   srdashboard.pageTitle=Dashboard
   ```

8. 「`UserInterface`」プロジェクトの「`WEB-INF`」フォルダの「`faces-config.xml`」を開きます。
9. 必要に応じて、**「ソース」**タブをクリックします。
10. ファイルの最後の `</faces-config>` 終了タグの前に、マネージド Bean（例 8-1）を既存のコードに追加して、「Dashboard」サブタブのメニュー項目を管理する Bean をインスタンス化します。

例 8-1「Dashboard」サブタブ用のマネージド Bean

```
<managed-bean>
    <managed-bean-name>subMenuItem_Manage_Dashboard</managed-bean-name>
    <managed-bean-class>oracle.srdemo.view.menu.MenuItem</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
      <property-name>label</property-name>
      <value>#{resources['srdemo.menu.manage.dashboard']}</value>
    </managed-property>
    <managed-property>
      <property-name>shown</property-name>
      <value>#{userInfo.manager}</value>
    </managed-property>
    <managed-property>
      <property-name>viewId</property-name>
      <value>/app/management/SRDashboard.jspx</value>
    </managed-property>
    <managed-property>
      <property-name>outcome</property-name>
      <value>Dashboard</value>
    </managed-property>
<!-- ADF Authorization -->
    <managed-property>
      <property-name>targetPageDef</property-name>
      <value>app_management_SRDashboardPageDef</value>
    </managed-property>
<!-- End ADF Authorization -->
</managed-bean>
```

11. ファイルを保存します。
12. 次の行を UIResources.properties ファイルの最後に追加します。

```
srdemo.menu.manage.dashboard=My Dashboard
```

13. faces-config.xml ファイルを開き、<managed-bean-name>menuItem_Manage を探します。
14. サブタブを作成したので、それをメニュー項目として親メニューに追加する必要があります。そのためには、次のコードを list-entries タグに追加します。

例 8-2「Dashboard」サブタブを「Management」ページに追加するためのコード

```
<value>#{subMenuItem_Manage_Dashboard}</value>
```

15. 例 8-3 に示すコードをナビゲーション・ルールに追加して、新しいダッシュボード・メニュー項目を JSPX ページにリンクします。

例 8-3「Dashboard」サブタブを Dashboard.jspx ページにリンクするためのコード

```
<navigation-rule>
   <from-view-id>/app/management/SRManage.jspx</from-view-id>
   <navigation-case>
      <from-outcome>Dashboard</from-outcome>
      <to-view-id>/app/management/SRDashboard.jspx</to-view-id>
   </navigation-case>
</navigation-rule>
```

16. ページを保存します。

手順 2: ページ・レイアウトの作成

この項では、カスタマイズ可能コンポーネント（特に PanelCustomizable コンポーネント）を使用して、ページのレイアウトを設計する方法を示します。カスタマイズ可能コンポーネントの使用方法の詳細は、第 7 章「ページの構築とコンポーネントの追加」のカスタマイズ可能コンポーネントに関する項を参照してください。

完成したデモの SRDashboard.jspx ページを開いてソースを表示すると、完成したページのコードを確認できます。

1. ダッシュボードの 1 列目を追加するには、まず、SRDashboard.jspx ファイルのソースを開きます。
2. テキストの「Please Insert Main Page Content Here!」を探し、その行の下にカーソルを置きます。
3. コンポーネント・パレットから、**「カスタマイズ可能コンポーネント・コア」**を選択します。

4. 「カスタマイズ可能コンポーネント・コア」リストで、**「PanelCustomizable」**をクリックして PanelCustomizable コンポーネントをページに追加します。コードは、図 8-2 に示すようなコードになります。

図 8-2 1 つ目の PanelCustomizable の配置

```
<!--
============================================================
============================================================
            Please Insert Main Page Content Here!
============================================================
============================================================
-->
      <cust:panelCustomizable id="panelCustomizable1"
                        text="panelCustomizable 1"/>
      <!-- Page Footer Section (Messages and Links at bottom of the page)========= -->
```

5. 「構造」ペインで、追加したばかりの**「cust:panelCustomizable」**を選択し、プロパティ・インスペクタで IsMovable プロパティを false に設定します。
6. 2 つ目の PanelCustomizable を作成したばかりの PanelCustomizable の下に追加します。
7. 「構造」ペインで、追加したばかりの**「cust:panelCustomizable」**を選択し、プロパティ・インスペクタで次のプロパティを設定します。
   - DisplayHeader: true
   - Id: panelCustomizable2
   - Layout: horizontal
   - Text: Business Analysis
   - IsMovable: true
   - IsSeededInteraction: true

   この時点で、ページのソース・コードは、図 8-3 に示すようなコードになります。

図 8-3 ソース・コードでの PanelCustomizable

```
<cust:panelCustomizable id="panelCustomizable1"
                        text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
                        text="Business Analysis"
                        layout="horizontal"
                        isSeededInteractionAvailable="true"
                        displayHeader="true"/>
```

8. 「カスタマイズ可能コンポーネント・コア」コンポーネント・パレットから**「ShowDetailFrame」**を選択して、作成したばかりの PanelCustomizable の下に ShowDetailFrame コンポーネントを作成します。

9. 新しい ShowDetailFrame の Text プロパティを Customer Contracts に設定します。コードは、図 8-4 に示すようなコードになります。

図 8-4 ソース・コードでの ShowDetailFrame

```
<cust:panelCustomizable id="panelCustomizable1"
                        text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
                        text="Business Analysis"
                        layout="horizontal"
                        isSeededInteractionAvailable="true"
                        displayHeader="true"/>
<cust:showDetailFrame id="showDetailFrame1"
                      text="Customer Contracts"/>
```

10. 作成したばかりの ShowDetailFrame コンポーネントの後にスペーサを挿入します。そのためには、コンポーネント・パレットから「**ADF Faces Core**」を選択し、「**ObjectSpacer**」を選択します。コードは、図 8-5 に示すようなコードになります。

図 8-5 ソース・コードでの ObjectSpacer

```
<cust:panelCustomizable id="panelCustomizable1"
                        text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
                        text="Business Analysis"
                        layout="horizontal"
                        isSeededInteractionAvailable="true"
                        displayHeader="true"/>
<cust:showDetailFrame id="showDetailFrame1"
                      text="Customer Contracts"/>
<af:objectSpacer width="10" height="10"/>
```

11. 2 つ目の PanelCustomizable コンポーネントを作成して次のプロパティを設定し、2 列目をページに追加します。

プロパティ	値
DisplayHeader	`false`
Layout	`horizontal`
Text	`Service Requests Volume`
IsMovable	`true`
IsSeededInteractionAvailable	`true`

12. ShowDetailFrame 追加し、Text プロパティを Service Requests Volume に設定します。

    この時点で、ページのソース・コードは、図 8-6 に示すようなコードになります。

図 8-6 2 つのカスタマイズ可能コンポーネントがあるダッシュボード・ページのソース・コード

```
<cust:panelCustomizable id="panelCustomizable1"
                        text="panelCustomizable 1" isMovable="false"/>
<cust:panelCustomizable id="panelCustomizable2"
                        text="Business Analysis"
                        layout="horizontal"
                        isSeededInteractionAvailable="true"
                        displayHeader="true"/>
<cust:showDetailFrame id="showDetailFrame1"
                      text="Customer Contracts"/>
<af:objectSpacer width="10" height="10"/>
<cust:panelCustomizable id="panelCustomizable3"
                        text="Service Request Volume"
                        layout="horizontal" isEditable="false"
                        isMovable="true"
                        isSeededInteractionAvailable="true"/>
<cust:showDetailFrame id="showDetailFrame2"
                      text=" Service Requests Volume"/>
```

13. 次に、Customer Contracts という ShowDetailFrame 内に水平パネルを追加します。

    そのためには、ShowDetailFrame の後の行にカーソルを置きます。コンポーネント・パレットから、**「ADF Faces Core」**を選択し、リストから**「PanelHorizontal」**を選択します。

14. 次に、PanelHorizontal の中に入力テキスト・フィールドを追加します。

    そのためには、追加したばかりの PanelHorizontal の後の行にカーソルを置き、コンポーネント・パレットから**「InputText」**を選択します。

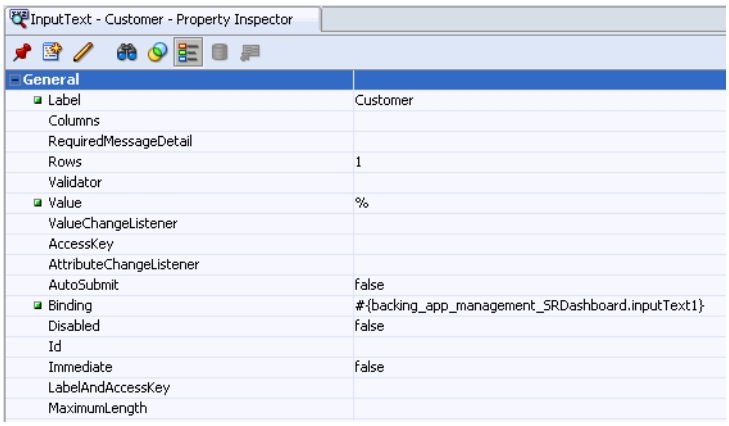
15. ソースで「InputText」コンポーネントを選択します。

16. プロパティ・インスペクタで、**Label** プロパティを Customer に設定します。

17. **Value** プロパティを % に設定します。

18. **Binding** プロパティを #{backing_app_management_SRDashboard.inputText1} に設定します。図 8-7 に、適切な値を設定したプロパティ・インスペクタを示します。

図 8-7 入力テキスト・フィールドのプロパティ・インスペクタ



結果のコードは、図 8-8 に示すソースのようになります。

図 8-8 入力テキスト・フィールドのソース・コード

```
<af:inputText label="Customer" value="%"
        binding="#{backing_app_management_SRDashboard.inputText1}"/>
```

19. 次に、ユーザーが顧客契約を検索できるようにするコマンド・ボタンを追加します。

    そのためには、InputText コンポーネントの下にカーソルを置き、コンポーネント・パレットから「**CommandButton**」を選択します。

20. Text プロパティを Search に設定します。

21. Action プロパティを #{backing_app_management_SRDashboard.commandButton_action} に設定します。

22. プロパティ・インスペクタで、Binding プロパティを #{backing_app_management_SRDashboard.commandButton1} に設定します。

23. 「ソース」で、Disabled プロパティを追加し、#{!bindings.search.enabled} と定義します。

    この時点で、コードは図 8-9 に示すようなコードになります。

図 8-9 入力テキスト・フィールドおよびコマンド・ボタンのソース

```
<cust:showDetailFrame id="showDetailFrame2"
                      text="Customer Contracts">
 <af:panelHorizontal>
  <af:inputText id="inputText1"
                binding="#{backing_app_management_SRDashboard.inputText1}"
                label="Customer"
                value="%"/>

  <af:commandButton id="commandButton1"
                    binding="#{backing_app_management_SRDashboard.commandButton1}"
                    text="search"
                    disabled="#{!bindings.search.enabled}"
                    action="#{backing_app_management_SRDashboard.commandButton_action}"/>
 </af:panelHorizontal>
```

24. InputText および CommandButton の下に表を追加して、各種ポートレットがページに水平に表示されるようにします。

手順 3: ページへの OmniPortlet のインスタンスの追加

この項では、OmniPortlet のインスタンスをページ・レイアウトに追加します。この章の後の手順ではポートレットを定義し、最終的にこれらのポートレットを 1 つに連結します。

1. レイアウトを作成したので、ポートレットをページに追加できます。

   ページのソースで、Customer Contracts という ShowDetailFrame の後、かつ作成した ObjectSpacer の前にカーソルを置き、次に「構造」ビューで、1 つ目の **PanelCustomizable** の上にカーソルを置きます。

2. コンポーネント・パレットから、**「OmniPortlet Producer」**を選択し、**「OmniPortlet」**をクリックします。この時点で、ページのソース・コードは、図 8-10 に示すようなコードになります。

図 8-10 ページのソース・コードでの OmniPortlet

```
<cust:panelCustomizable id="panelCustomizable1"
                        text="Business Analysis" isMovable="true"
                        layout="horizontal"
                        isSeededInteractionAvailable="true"/>
<cust:showDetailFrame id="showDetailFrame1" text="Customer Contracts"/>
<adfp:portlet value="#{bindings.portlet2}"
              portletType="/oracle/adf/portlet/OmniPortlet_Producer/applicationPortlets/Portlet100
<af:objectSpacer width="10" height="10"/>
```

注意： コンポーネント・パレットのドロップダウン・リストに OmniPortlet Producer が表示されない場合は、この例に関連するプロデューサを登録する必要があります。

---

3. 「OmniPortlet」を選択し、プロパティ・インスペクタで ID を customerContracts に設定します。

4. ページのソースで、追加したばかりの OmniPortlet の後にカーソルを置き、OmniPortlet の 2 つ目のインスタンスを追加して ID を `mostProductiveEmployees` に変更します。
5. 「構造」ペインで、3 つ目の「`PanelCustomizable`」の下の「`ShowDetailFrame`」を選択して OmniPortlet を追加し、ID を `serviceRequestVolume` に設定します。
6. 「構造」ペインで、3 つ目の「`PanelCustomizable`」を選択して別の OmniPortlet を追加し、ID を `mostRequestedProducts` に設定します。
7. 3 つ目の「`PanelCustomizable`」を再度選択して別の OmniPortlet を追加し、ID を `customerDetails` に設定します。

手順 4: Service Request Volume ポートレット用の SelectOneChoice コンポーネントの追加

この項では、Service Request Volume ポートレット用に `SelectOneChoice` コンポーネントを追加します。

1. 「構造」ビューで、3 つ目の「`PanelCustomizable`」の下の**「ShowDetailFrame」**を選択し、ADF Core コンポーネント・パレットから `SelectOneChoice` コンポーネントを追加します。
2. `SelectOneChoice` コンポーネントに、次のプロパティを設定します。
   - ID: `dayPicker`
   - label: `Volume for last`
   - default value: `360`
   - items: `1、2、3、5、10、30、60、90、180、360 days`
   - autoSubmit: `true`

   次のコードが JSPX に表示されます。

```
<af:selectOneChoice label="Label 1" value="360" autoSubmit="true"
id="dayPicker">
<af:selectItem label="1 day" value="1"/>
<af:selectItem label="2 days" value="2"/>
<af:selectItem label="3 days" value="3"/>
<af:selectItem label="5 days" value="5"/>
<af:selectItem label="10 days" value="10"/>
<af:selectItem label="30 days" value="30"/>
<af:selectItem label="60 days" value="60"/>
<af:selectItem label="90 days" value="90"/>
<af:selectItem label="180 days" value="180"/>
<af:selectItem label="360 days" value="360"/>
</af:selectOneChoice>
```

3. 「構造」ビューで、2 つ目の「`PanelCustomizable`」から**「ShowDetailFrame」**を選択します。
4. 最初のタブで、**「リストの作成」**を選択し、ドロップダウン・リストの内容を入力します。
5. **「アイテムの追加」**をクリックします。
6. 「アイテム・ラベル」フィールドに `1 day` と入力し、「アイテム値」フィールドに `1` と入力します。
7. 前の手順を、2、3、5、10、30、60、90、180 および 360 の各日数について繰り返します。
8. **「共通プロパティ」**タブをクリックします。
9. 「ラベル」フィールドに、`Volume for last` と入力します。
10. 「値」フィールドに、`360` と入力します。
11. 「アドバンスト・プロパティ」タブで、autoSubmit オプションに `true` と入力します。

12. **「OK」**をクリックしてコンポーネントを作成します。
13. 「プロパティ」ビューで、SelectOneChoice の id を dayPicker に更新します。
14. ShowDetailFrame 内で、**dayPicker** コンポーネントを OmniPortlet の上にドラッグ・アンド・ドロップします。

手順 5: 顧客契約の検索機能の追加

次の手順では、ユーザーが既存の顧客契約を検索できるようにするコンポーネントを追加します。

1. ページ・ソースで、Customer Contracts という ShowDetailFrame を探し、その ShowDetailFrame の下にある PanelHorizontal の下にカーソルを置きます。
2. データ・コントロール・パレットで、**「ContentRepository」**データ・コントロールを開き、**「Search」**を開きます。
3. Search メソッドを ADF コマンド・ボタンとして PanelHorizontal の下にドラッグ・アンド・ドロップします。
4. 「構造」ペインで、**「Search」**メソッドを選択します。
5. アクション・バインディング・エディタで、/contracts をパスとして指定します。
6. **isRecursive** を **true** に設定し、**「OK」**をクリックします。
7. データ・コントロール・パレットで、「Search」の下の**「Parameters」**を開きます。
8. **NamePattern** メソッドをラベル付きの入力テキスト・コンポーネントとして Search ボタンの前にドラッグ・アンド・ドロップします。
9. 「構造」ペインで、追加したばかりの**「InputText」**コンポーネントを選択し、**Label** プロパティに Name (use % as wildcard) と設定します。
10. データ・コントロール・パレットの「Search」の下で、**Return** メソッドを ADF 読取り専用表として PanelHorizontal の下にドラッグ・アンド・ドロップします。
11. ページを保存します。

手順 6: Most Productive Employees ポートレットの定義

次の手順では、複数の OmniPortlets を定義します。次に、この章の最後で、これらのポートレットを 1 つに連結してインタラクティビティを作成します。

この項では、組織で最も生産性の高い従業員を表示するポートレットを定義します。OmniPortlet インスタンスを定義するには、まず、JSPX ページを実行してブラウザに表示し、OmniPortlet ウィザードを使用してポートレットを定義します。

注意 : OmniPortlet ウィザードの使用方法の詳細は、『Oracle WebCenter Framework 開発者ガイド』の「OmniPortlet を使用したポートレットの作成」を参照してください。

---

1. SRDashboard.jspx ページを実行してブラウザに表示します。
2. 最初のポートレットの**「定義」**リンクをクリックし、データ・タイプとして**「SQL」**を選択して、**「次へ」**をクリックします。

3. 「ソース」タブで、例 8-4 に示すような SQL 文を入力します。

例 8-4 Most Productive Employees ポートレット用の SQL 文

```
select count(SH.CREATED_BY) as NB_REQ, SH.CREATED_BY,U.FIRST_NAME, U.LAST_NAME
FROM SERVICE_HISTORIES SH INNER JOIN USERS U ON U.USER_ID = SH.CREATED_BY WHERE
SH.SVH_TYPE = 'Technician' OR SH.SVH_TYPE = 'Hidden' GROUP BY SH.CREATED
BY,U.FIRST_NAME,U.LAST_NAME ORDER BY NB_REQ DESC
```

4. **「次へ」**をクリックし、再度**「次へ」**をクリックします。
5. ビュー・タブで、**「表」**を選択し、**「次へ」**をクリックします。
6. 「レイアウト」タブで、次のオプションを更新します。

   タイトル : `Most Productive Employees`

   レイアウト : Service Request (NB_REQ)

   First Name (FIRST_NAME)

   Last Name (LAST_NAME)
7. **「終了」**をクリックします。

手順 7: Service Request Volume ポートレットの定義

この項の手順では、サービス・リクエストに関する情報を円グラフで表示する方法を示します。

1. ブラウザで、`SRDashboard.jspx` ページの 2 つ目のポートレットの**「定義」**リンクをクリックします。
2. 「タイプ」タブで、**「SQL」**を選択し、**「次へ」**をクリックします。
3. 「ソース」タブの「SQL 文」フィールドに、次のように入力します。

例 8-5 Service Request Volume ポートレット用の SQL 文

```
select count(SVR_ID) as NB_REQ, STATUS FROM SERVICE_REQUESTS
WHERE REQUEST_DATE > (SYSDATE - ##Param1##)
GROUP BY STATUS ORDER BY NB_REQ
```

4. データベースへのグローバル接続を設定します。

   「接続」で、**「接続の編集」**をクリックします。
5. 「接続名」フィールドに、`SRDemo` と入力します。
6. 「ユーザー名」フィールドに、`hr` と入力します。
7. 「パスワード」フィールドに、`hr` と入力します。これらは、サンプル・ファイルのダウンロード時にインストールした SRDemo スキーマ用のユーザー名およびパスワードです。
8. 「接続文字列」フィールドに、データベースへのパス（`mydemo.mycompany.com:1522:XE` など）を入力し、**「OK」**をクリックします。これで、「ソース」タブの「接続」セクションに接続情報が表示されます。
9. `Param1` というポートレット・パレメータを作成し、デフォルト値を `360` に設定して、**「次へ」**をクリックします。
10. 「フィルタ」タブで、**「次へ」**をクリックします。
11. ビュー・タブで、**「チャート」**を選択し、**「次へ」**をクリックします。
12. 「レイアウト」タブで、**「円グラフ」**を選択し、「幅」には `300`、「高さ」には `200` と入力します。
13. 「凡例」を**「右揃え」**に設定し、**「3D 効果」**チェック・ボックスを選択します。
14. 「グループ」リストから、**「なし」**を選択します。

15. 「カテゴリ」リストから、**「ステータス」**を選択します。
16. 「値」リストから、**「NB_REQ」**を選択し、**「終了」**をクリックします。

手順 8: Most Requested Products ポートレットの定義

この項の手順では、SQL 文を使用してデータベースのデータにアクセスし、最もリクエストされた製品を表示するポートレットを定義する方法を示します。

1. ブラウザで、SRDashboard.jspx ページの 3 つ目のポートレットの**「定義」**リンクをクリックします。
2. 「タイプ」タブで、**「SQL」**を選択し、**「次へ」**をクリックします。
3. 「ソース」タブの「SQL 文」フィールドに、次のように入力します。

例 8-6 Most Requested Products ポートレット用の SQL 文

```
select count(SR.SVR_ID) as NB_REQ, SR.PROD_ID, P.NAME FROM
SERVICE_REQUESTS SR INNER JOIN PRODUCTS P ON SR.PROD_ID = P.PROD_ID
GROUP BY SR.PROD_ID, P.NAME ORDER BY NB_REQ DESC
```

4. **「次へ」**をクリックします。
5. 「フィルタ」タブで、**「次へ」**をクリックします。
6. ビュー・タブで、**「チャート」**を選択し、**「次へ」**をクリックします。
7. 「レイアウト」タブで、**「棒グラフ」**を選択し、次のオプションを設定します。

   幅 : 300

   高さ : 250

   グループ : Name

   カテゴリ : < なし >

   値 : NB_REQ
8. **「終了」**をクリックします。

手順 9: Most Active Customers ポートレットの定義

この項の手順では、最もアクティブな顧客を表示するポートレットを定義する方法を示します。

ページの「Customer Information」セクションに 2 つのポートレットを作成するには、次のようにします。

1. ブラウザで、SRDashboard.jspx ページの 4 つ目のポートレットの**「定義」**リンクをクリックします。
2. 「タイプ」タブで、**「SQL」**を選択し、**「次へ」**をクリックします。
3. 「ソース」タブで、次の SQL 文を入力し、**「次へ」**をクリックします。

例 8-7 Most Active Customers ポートレット用の SQL 文

```
select count(SH.CREATED_BY) as NB_REQ, SH.CREATED_BY , U.FIRST_NAME, U.LAST_NAME
FROM SERVICE_HISTORIES SH INNER JOIN USERS U ON U.USER_ID = SH.CREATED_BY WHERE
SH.SVH_TYPE = 'Customer' GROUP BY SH.CREATED_BY,U.FIRST_NAME,U.LAST_NAME ORDER BY
NB_REQ DESC
```

4. 「フィルタ」タブで、変更せずに**「次へ」**をクリックします。
5. ビュー・タブで、**「HTML レイアウト」**を選択してタイトルに Most Active Customers と入力し、**「次へ」**をクリックします。
6. 「レイアウト」タブで、**「ソート可能テンプレート」**を選択し、**「適用」**をクリックします。

7. ヘッダー・セクションのコードはデフォルトのままにしておきます。
8. 「繰返しセクション」で、次のようなコードに更新します。

```
<tr class="portlet-section-alternate opRowColorPI1146522302846_##OP_ROWNUM_MOD2##">
<td class=PortletText1>##NB_REQ##</td>
<td class=PortletText1>##CREATED_BY##</td>
<td class=PortletText1>##FIRST_NAME##</td>
<td class=PortletText1><a href="/SRDemo/faces/app/management/SRDashboard.jspx?
customerID=##CREATED_BY##" target="_top">##LAST_NAME##</a></td>
</tr>
```

9. フッター・セクションで、次のようなコードに更新します。

```
</tbody>
</table>
<font class=PortletText1>Total Rows: ##OP_ROWNUM##</font>
```

10. **「終了」**をクリックします。

手順 10: Customer Details ポートレットの定義

この項では、特定の顧客に関する詳細を表示するポートレットを定義します。

1. ブラウザで、SRDashboard.jspx ページの 5 つ目のポートレットの**「定義」**リンクをクリックします。
2. 「タイプ」タブで、**「SQL」**を選択し、**「次へ」**をクリックします。
3. 「ソース」タブで、次の SQL 文を入力します。

例 8-8 Customer Details ポートレット用の SQL 文

```
select * from USERS where USER_ID = '##Param1##')
```

4. **Param1** をデフォルト値の 320 に設定し、**「次へ」**をクリックします。
5. ビュー・タブで、**「HTML レイアウト」**を選択してタイトルに Customer Details と入力し、**「次へ」**をクリックします。
6. 「レイアウト」タブで、フィールドの値を削除します。
7. 「テンプレート」ドロップダウン・リストから**「ソート可能表」**を選択し、ヘッダー・セクションに次のコードを入力します。

```
<TABLE BORDER='0' WIDTH="100%">
```

8. 「繰返しセクション」のコードを次のコードで置き換えます。

```
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>ID</TD>
<TD>##USER_ID##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>Email</TD>
<TD>##EMAIL##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>First Name</TD>
<TD>##FIRST_NAME##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>Last Name</TD>
<TD>##LAST_NAME##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>Street</TD>
<TD>##STREET_ADDRESS##</TD>
```

```
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>City</TD>
<TD>##CITY##</TD>
</TR>
<TR CLASS='PortletText1'>
<TD CLASS='PortletHeading1'>State</TD>
<TD>##STATE_PROVINCE##</TD>
</TR>
```

9. フッター・セクションのコードを次のコードで置き換えます。

```
</tbody>
</table>
<font class=PortletText1>Total Rows: ##OP_ROWNUM##</font>
```

10. **「終了」**をクリックします。

手順 11: ページ・コンテンツの連結

この項の手順では、作成したばかりのポートレットにインタラクティビティを追加する方法を示します。この項では、1 つのページ・パラメータおよび複数のポートレット・パラメータを作成し、ページ上のポートレットを 1 つに連結します。

1. ページ・パラメータを作成するには、Oracle JDeveloper に戻り、**SRDashboardPageDef.xml** ファイルを開きます。
2. **custID** というページ・パラメータを作成します。

   そのためには、「構造」ペインで**「Parameters」**を右クリックし、新しいページ・パラメータを挿入して次のように設定します。

   ```
   <parameter id ="custID" value="$param.customerID)"/>
   ```

3. 作成した `SRDashboard.jspx` ページのソースを開きます。
4. ポートレット・パラメータを Customer Details ポートレットに追加します。
5. `Param1` の値には、bindings.custID からのページ・パラメータ値を設定します（デフォルト値を `320` に設定して、少なくとも 1 人の顧客が表示され、ページをアクティブにしますが、顧客は選択されないようにします）。

   ```
   <parameter name="Param1" value="${(bindings.custID == null || bindings.custID ==
   '') ? 320 : bindings.custID }"/>
   ```

6. Service Request Volume ポートレット用のパラメータを追加して、`SelectOneChoice` から値が選択されるとポートレットがリフレッシュされるようにします。

   「構造」ビューで、「`parameters`」を右クリックします。
7. **「パラメータの中に挿入」**を選択し、**「parameter」**を選択します。
8. **Param1** の値を `360` に設定します。デフォルトで 1 年または 360 日を使用して、すべての結果を表示します。
9. ページ定義の「executables」→「variables」内の**「OmniPortlet3_1_Param1」**を選択します。
10. プロパティ・インスペクタの「DefaultValue」フィールドに、次のように入力します。

    ```
    ${( app_management_SRDashboard.dayPicker.value == null) ? 360 :
    app_management_SRDashboard.dayPicker.value}
    ```

11. 「構造」ペインで、5 つ目のポートレット（Most Active Customers ポートレット）を選択します。
12. プロパティ・インスペクタで、`SubmitUrlParameters` プロパティを `true` に設定します。この設定により、ポートレットは URL を介してパラメータをページに送信できるようになります。

13. ページを保存します。

14. `SRDashboard.jspx` ページを実行してブラウザに表示し、追加したばかりのポートレットを確認します。

まとめ

この章の手順では、WebCenter アプリケーション用のダッシュボード・ページを構築する方法を示しました。カスタマイズ可能コンポーネントおよび ADF Faces Core コンポーネントを使用してレイアウトを作成し、ページを作成する方法を学びました。また、関連情報を表示するポートレットを作成し、ページ上で 1 つに連結してエンド・ユーザーが単一のエントリ・ポイントで顧客に関する情報を簡単に表示できるようにする方法も学びました。

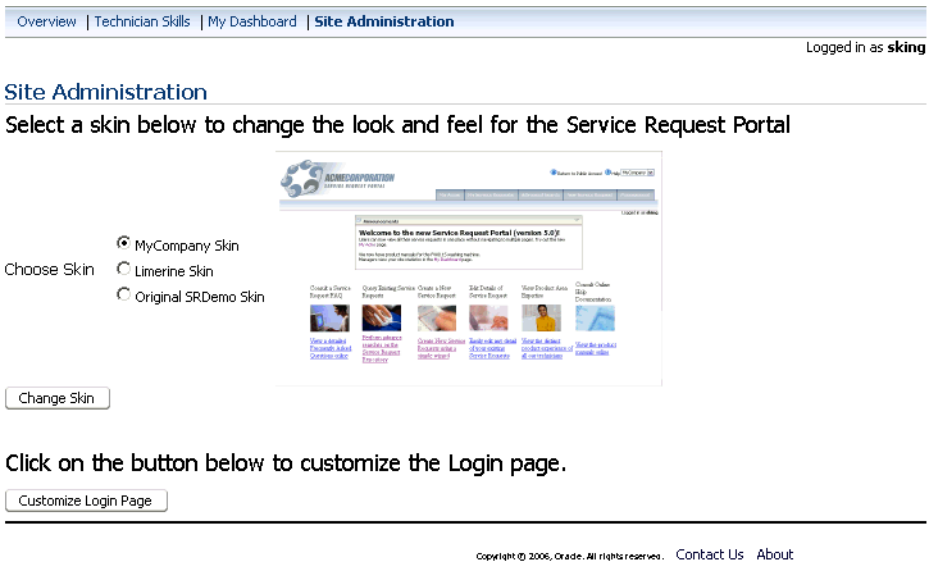
9

サイト管理ページの構築

WebCenter アプリケーションの作成はすでに完了しています。この章では、管理者がアプリケーションのルック・アンド・フィールの変更およびログイン・ページのカスタマイズに使用できるサイト管理ページを作成します。このサイト管理ページは、後で「Management」ページのサブタブとして追加されます。

SRDemo ZIP ファイルには、`myCompany`、`limerine` および `original` の 3 つのスキンと、対応する必要なイメージ・ファイルが含まれています。この章で作成するサイト管理ページを使用すると、エンド・ユーザーは実行時に 3 つのスキンから選択してアプリケーションのスキンを変更できます。また、アプリケーションのログイン・ページをカスタマイズすることも可能です。実行時のサイト管理ページは、図 9-1 のような外観です。

図 9-1 実行時のサイト管理ページ



この章の作業を実行するには、第 3 章「スキンの設定」の手順を実行してアプリケーション用のスキンを設定する必要があります。

この章の内容は次のとおりです。

- 手順 1: サイト管理ページの作成
- 手順 2: ページへのスキン・セレクタの追加
- 手順 3: ログイン・ページのカスタマイズの可能化
- 手順 4: 「Management」ページのサブタブとしての「Site Administration」ページの追加
- 手順 5: カスタマイズのためのサイト管理ページの使用

手順 1: サイト管理ページの作成

次の手順では、SRSiteAdmin.jspx ページを構築し、アプリケーションのルック・アンド・フィールの変更およびログイン・ページのカスタマイズを管理者ができるようにするコントロールを設定する方法を示します。

注意 : このページでは、すでに作成したスキン Bean で使用されるスキンの値を設定します。

サイト管理ページを作成するには、次の手順を実行します。

1. アプリケーション・ナビゲータで、必要に応じて、**「UserInterface」**ノードおよび**「Web コンテンツ」**ノードを開きます。
2. **「app」**ノードの下の**「management」**フォルダを右クリックし、**「新規」**を選択します。
3. 新規ギャラリで、**「Web Tier」**ノードを開きます。
4. **「JSF」**を選択します。
5. 「項目」リストで、**「JSF JSP」**を選択します。
6. **「OK」**をクリックして「JSF JSP の作成」ダイアログ・ボックスを表示します。
7. ウィザードの「ようこそ」ページが表示されたら、**「次へ」**をクリックして「JSP ファイル」ページを表示します。
8. 「ファイル名」フィールドに、SRSiteAdmin.jspx と入力します。
9. **「JSP ドキュメント (*.jspx)」**を選択します。
10. **「次へ」**をクリックして「コンポーネント・バインディング」ページを表示します。
11. **「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、「名前」フィールドに app_management_SRSiteAdmin と表示されていることを確認します。
12. **「次へ」**をクリックして「タグ・ライブラリ」ページを表示します。
13. 次のライブラリが「選択済のライブラリ」に表示されていることを確認します。
    - JSF Core
    - JSF HTML
    - ADF Faces Components
    - ADF Faces HTML
    - ADF Portlet Components
    - Customizable Components Core
14. **「終了」**をクリックしてページを作成します。
15. **「WEB-INF」**ノードおよび**「template」**ノードを開きます。
16. **「SRDemoTemplate.jspx」**を右クリックし、**「開く」**を選択します。
17. ソース・コードを SRDemoTemplate.jspx ページからコピーして新しい SRSiteAdmin.jspx ページのソースに貼り付け、新しいページの既存のコードをすべて置き換えます。
18. SRSiteAdmin.jspx ページのソースで次の行を探します。

    ```
    <af:panelPage title="#{res['_PAGE_NAME_.pageTitle']}">
    ```

    この行を次の行で置き換えます。

    ```
    <af:panelPage title="#{res['SRSiteAdmin.pageTitle']}">
    ```

19. ファイルを保存します。

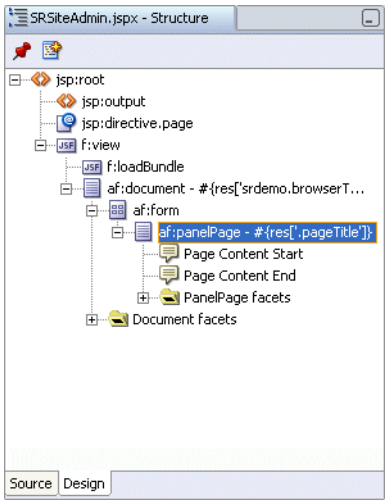
手順 2: ページへのスキン・セレクタの追加

今度は、アプリケーションに対して特定のスキンを選択できるようにする必要があります。そのためには、ラジオ・グループを作成し、管理者が実行時に使用するスキンを選択できるようにします。また、各スキンの外観をプレビューできるようにします。

スキン・セレクタをページに追加するには、次の手順を実行します。

1. 「構造」ペインで、「`f:view`」、「`af:document`」、「`af:form`」、「`af:panelPage`」の順にノードを開きます（図 9-2 を参照）。

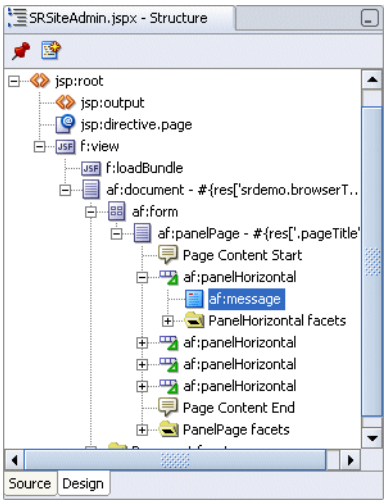
図 9-2 開かれた「構造」ペイン



2. コンポーネント・パレットから、「**ADF Faces Core**」を選択します。
3. **PanelHorizontal** コンポーネントを「構造」ペインにドラッグし、`Page Content Start` と `Page Content End` の間に表示されるようにします。
4. さらに 3 つの `panelHorizontal` コンポーネントを同じように追加し、全部同じレベルにして `panelPage` コンポーネントの子コンポーネントとします。

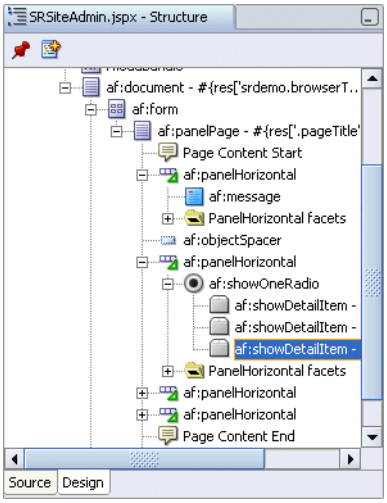
5. **Message** コンポーネントをコンポーネント・パレットから「構造」ペインにドラッグし、1 つ目の `panelHorizontal` コンポーネント内に表示されるようにします（図 9-3 を参照）。

図 9-3 ページの「構造」ペインでの Message コンポーネント



6. プロパティ・インスペクタで、`Message` コンポーネントのプロパティを次のように変更します。
   - `Message: Select a skin below to change the look and feel for the Service Request Portal`
   - `InlineStyle/font-size: large`
7. **ObjectSpacer** コンポーネントをコンポーネント・パレットから「構造」ペインにドラッグし、1 つ目と 2 つ目の `panelHorizontal` コンポーネントの間に表示されるようにします。
8. **ShowOneRadio** コンポーネントを 2 つ目の `panelHorizontal` コンポーネントにドラッグします。
9. 3 つの `showDetailItem` コンポーネントを `showOneRadio` コンポーネントの下に追加します（図 9-4 を参照）。

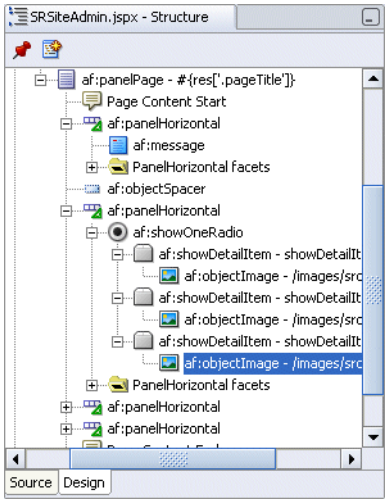
図 9-4 ページの「構造」ペインでの ShowOneRadio コンポーネント



10. **ObjectImage** コンポーネントを 1 つ目の showDetailItem コンポーネントにドラッグします。「ObjectImage の挿入」ダイアログ・ボックスが表示されます。
11. 「ソース」フィールドの横の**拡張エディタを表示する**ボタンをクリックします。
12. リストから**「srdemo_mycompany.gif」**を選択し、**「OK」**をクリックします。
13. **「OK」**をクリックします。
14. 他の 2 つの showDetailItem コンポーネントについて直前の 4 つの手順を繰り返し、次のイメージを追加します。
    - srdemo_limerine.gif
    - srdemo_original.gif

    これらのコンポーネントは、図 9-5 に示すように「構造」ペインに表示されます。

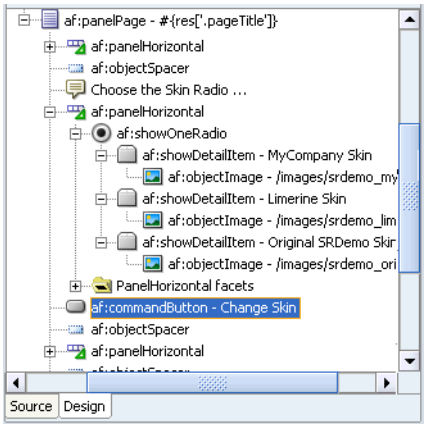
図 9-5 ラジオ・ボタンのイメージ



15. 1 つ目の showDetailItem コンポーネントを選択し、プロパティ・インスペクタで次のプロパティを設定します。
    - Binding: #{backing_SRSiteAdmin.showDetailItem1}
    - DisclosureListener: #{skinBean.processDisclosure}
    - Text: My Company Skin
16. 2 つ目の showDetailItem コンポーネントを選択し、次のプロパティを設定します。
    - Binding: #{backing_SRSiteAdmin.showDetailItem2}
    - DisclosureListener: #{skinBean.processDisclosure}
    - Text: Limerine Skin
17. 3 つ目の showDetailItem コンポーネントを選択し、次のプロパティを設定します。
    - Binding: #{backing_SRSiteAdmin.showDetailItem3}
    - DisclosureListener: #{skinBean.processDisclosure}
    - Text: Original SRDemo Skin

18. **CommandButton** コンポーネントを 2 つ目と 3 つ目の `panelHorizontal` コンポーネントの間にドラッグします（図 9-6 を参照）。

図 9-6「Change Skin」コマンド・ボタン

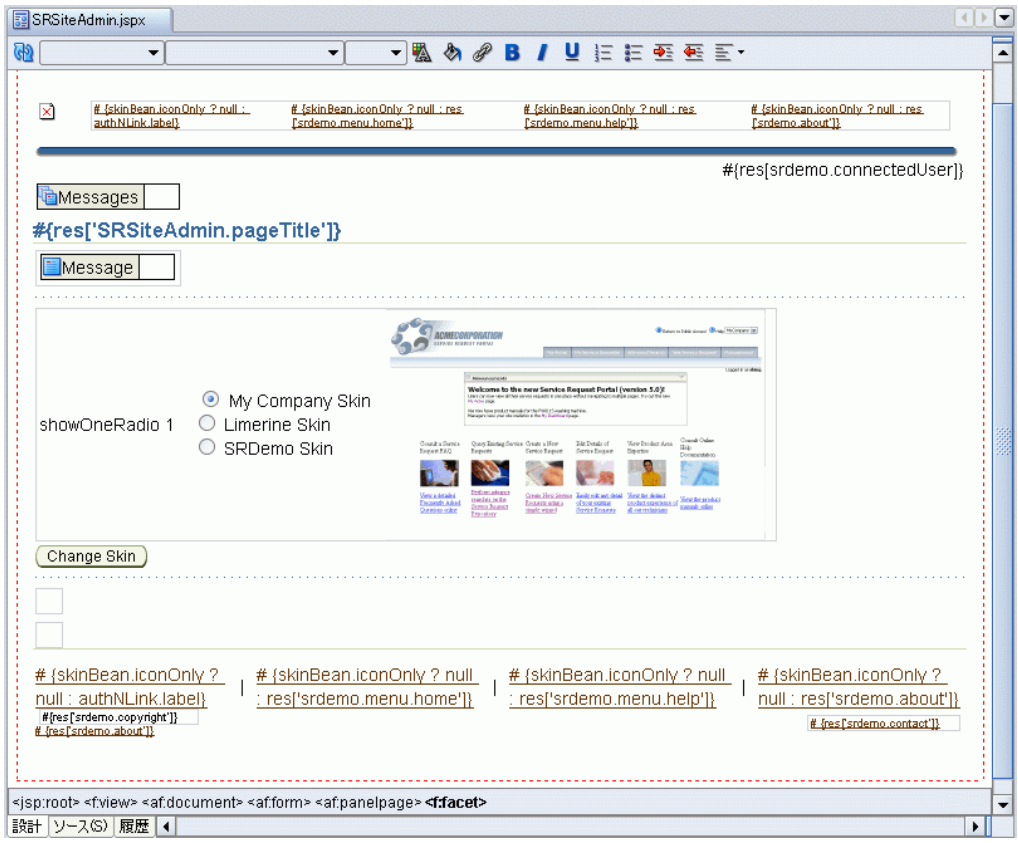


19. `commandButton` の `Text` プロパティを `Change Skin` に設定します。

20. **ObjectSpacer** コンポーネントをコンポーネント・パレットから「構造」ペインにドラッグし、1 つ目と 2 つ目の `panelHorizontal` コンポーネントの間に表示されるようにします。

21. ファイルを保存します。この時点で、サイト管理ページは、図 9-7 のような外観になります。

図 9-7 サイト管理ページ



手順 3: ログイン・ページのカスタマイズの可能化

ここでは、ログイン・ページをカスタマイズして、アプリケーションを再デプロイせずに、実行時に更新できる情報を組み込む方法を示します。この項の手順を実行すると、ログイン後にログイン・ページに戻って、第 6 章「ログイン・ページの作成」の「手順 3: ログイン・ページへのリッチ・テキスト・ポートレットの追加」で追加したリッチ・テキスト・ポートレットをカスタマイズできます。

ログイン・ページをカスタマイズできるようにするには、次の手順を実行します。

1. サイト管理ページからログイン・ページに移動するためのナビゲーション・ケースを作成します。そのためには、例 9-1 に示すコードを `faces-config.xml` ファイルに追加します。

例 9-1 サイト管理ページからログイン・ページに移動するためのナビゲーション・ルール

```
<navigation-rule>
  <from-view-id>/app/management/SRSiteAdmin.jspx</from-view-id>
  <navigation-case>
    <from-outcome>CustomizeLoginPage</from-outcome>
    <to-view-id>/infrastructure/SRLoginADF.jspx</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

2. ログイン・ページからサイト管理ページに移動するためのナビゲーション・ケースを作成します。そのためには、例 9-2 に示すコードを `faces-config.xml` ファイルに追加します。

例 9-2 ログイン・ページからサイト管理ページに移動するためのナビゲーション・ルール

```
<navigation-rule>
  <from-view-id>/infrastructure/SRLoginADF.jspx</from-view-id>
  <navigation-case>
    <from-outcome>ReturnToSiteAdmin</from-outcome>
    <to-view-id>/app/management/SRSiteAdmin.jspx</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

3. **ObjectSpacer** コンポーネントを `commandButton` と 3 つ目の `panelHorizontal` の間にドラッグします。

4. **OutputFormatted** コンポーネントを 3 つ目の `panelHorizontal` にドラッグし、プロパティ・インスペクタで次のプロパティを設定します。

   - `Value: #{res['srsiteadmin.editLoginPagePrompt']}`

     この EL は、リソース・バンドルから適切な文字列を返します。

   - `InlineStyle/font-size: large`

5. **commandButton** を 4 つ目の `panelHorizontal` コンポーネントにドラッグし、次のプロパティを設定します。

   - `Text: Customize Login Page`
   - `Action: CustomizeLoginPage`

   このボタンは、すでに定義した ADF Faces ナビゲーションを使用して、ユーザーをログイン・ページに移動します。

注意： パブリック・ユーザーが認証のためにログイン・ページにアクセスする場合とは異なり、サイト管理ページからこのページにアクセスする場合、ユーザーの ID はすでに定義されています。そのため、ログイン・ページのカスタマイズ時に次のログイン試行をさせないためには、画面のログイン・コンポーネントを非アクティブ化する必要があります。この非アクティブ化は、第 6 章「ログイン・ページの作成」の「手順 1: ログイン・ページの作成」で追加されたバッキング Bean コードで処理されます。

6. すべての管理者がログイン・ページをカスタマイズできないようにし、このページに対してカスタマイズ権限を持つユーザーにのみボタンを公開するには、commandButton の Rendered プロパティを次のように設定する必要があります。

   ```
   Render: #{bindings.permissionInfo['SRLoginADFPagePageDef'].allowsCustomize}
   ```

7. ファイルを保存します。

8. SRLogin.jspx ページを開き、**commandButton** をページの上にドラッグして次のプロパティを設定します。

   - Text: Return
   - Action: ReturnToSiteAdmin

   このボタンでは、すでに定義した ADF Faces ナビゲーションを使用して、必要なカスタマイズを実行した後にユーザーをサイト管理ページに戻します。

9. commandButton の Rendered プロパティを次のように設定します。

   ```
   Render: #{authNLink.authenticated}
   ```

10. ファイルを保存します。

手順 4: 「Management」 ページのサブタブとしての「Site Administration」 ページの追加

最後に、SRSiteAdmin.jspx ページを「Management」メイン・ページのサブタブとして追加する必要があります。そのためには、次の手順を実行します。

1. 「UserInterface」プロジェクトの「WEB-INF」フォルダで、**「faces-config.xml」**を右クリックし、**「開く」**を選択します。
2. 必要に応じて、**「ソース」**タブをクリックします。
3. 例 9-3 に示すマネージド Bean を、既存のコードの「Dashboard」サブタブ用のマネージド Bean の後に追加します。

例 9-3「Site Administration」サブタブ用のマネージド Bean

```
<managed-bean>
    <managed-bean-name>subMenuItem_Manage_SiteAdmin</managed-bean-name>
    <managed-bean-class>oracle.srdemo.view.menu.MenuItem</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
      <property-name>label</property-name>
      <value>#{resources['srdemo.menu.manage.siteadmin']}</value>
    </managed-property>
    <managed-property>
      <property-name>shown</property-name>
      <value>#{userInfo.manager}</value>
    </managed-property>
    <managed-property>
      <property-name>viewId</property-name>
      <value>/app/management/SRSiteAdmin.jspx</value>
    </managed-property>
```

```
    <managed-property>
      <property-name>outcome</property-name>
      <value>SRSiteAdmin</value>
    </managed-property>
<!-- ADF Authorization -->
    <managed-property>
      <property-name>targetPageDef</property-name>
      <value>app_management_SRSiteAdminPageDef</value>
    </managed-property>
<!-- End ADF Authorization -->
</managed-bean>
```

4. `UIResources.properties` ファイルを開きます。このファイルは、**「UserInterface」**→**「アプリケーション・ソース」**→**「oracle.srdemo.view」**→**「resources」**の下にあります。
5. 管理ページ領域を探し、次のコードを挿入します。

```
srdemo.menu.manage.siteadmin=Administration
```

6. `<managed-bean-name>menuItem_Manage` を探します。
7. 例 9-4 に示すコードを `list-entries` タグに追加します。

例 9-4 list-entries タグに追加するコード

```
<value>#{subMenuItem_Manage_SiteAdmin}</value>
```

8. `<from-view-id>/app/management/SRManage.jspx` を探します。
9. 例 9-5 に示すコードをナビゲーション・ルールに追加します。

例 9-5 navigation-case タグに追加するコード

```
<navigation-case>
  <from-outcome>SRSiteAdmin</from-outcome>
  <to-view-id>/app/management/SRSiteAdmin.jspx</to-view-id>
</navigation-case>
```

10. 例 9-6 に示すコードがすでに挿入されていることを確認します。

例 9-6 managed-bean タグに追加するコード

```
<managed-bean>
    <managed-bean-name>backing_SRSiteAdmin</managed-bean-name>
<managed-bean-class>oracle.srdemo.view.backing.app.management.SRSiteAdmin</managed-bean
-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

11. ファイルを保存します。
12. アプリケーション・ナビゲータの「UserInterface」プロジェクトで、**「index.jspx」**を右クリックし、**「実行」**を選択します。

別のページ上にサブタブとしてページを追加する方法の詳細は、第 8 章「ダッシュボード・ページの構築」を参照してください。

手順 5: カスタマイズのためのサイト管理ページの使用

ここでは、サイト管理ページを使用してアプリケーションをカスタマイズする方法を示します。このページでは、次のタスクを実行できます。

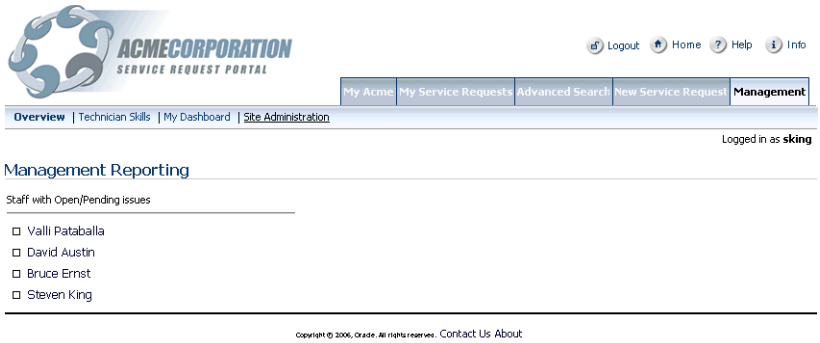
- アプリケーションのスキンの変更
- ログイン・ページのカスタマイズ

アプリケーションのスキンの変更

アプリケーションのスキンを変更するには、次の手順を実行します。

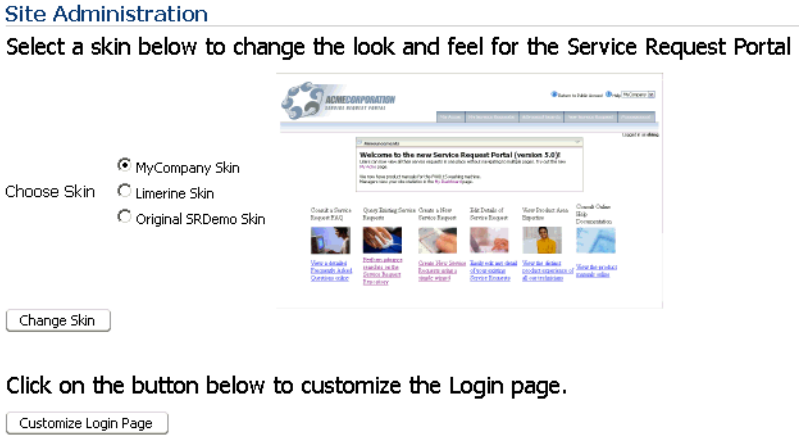
1. パスワードが `welcome` の `sking` として、Service Request ポータルにログインします。
2. 「**Management**」タブをクリックします。このとき、このページにはサイト管理ページのサブタブが組み込まれています（図 9-8 を参照）。

図 9-8 Service Request ポータル・ページの「Management」タブ



3. 「**Site Administration**」サブタブをクリックします。このページには、管理者がアプリケーションに使用するスキンを選択できるラジオ・グループがあります（図 9-9 を参照）。アプリケーションで使用されるデフォルトのスキンは、`MyCompany Skin` です。

図 9-9 カスタマイズ・オプションがある「Site Administration」タブ



4. 「**Limerine Skin**」を選択します。イメージが更新され、選択したスキンがプレビューされます。
5. 「**Change Skin**」をクリックします。ページの外観が変化し、`Limerine` スキンが使用されます。この変更は、アプリケーション全体に適用されます。

ログイン・ページのカスタマイズ

管理者である場合、ログイン・ページをカスタマイズし、そのページでユーザーに表示されるテキストを編集できます。そのためには、サイト管理ページの「Customize Login Page」ボタンを使用します。

注意：「Customize Login Page」ボタンは、ログイン・ページに対して customize 権限を持つ認証済ユーザーにのみ表示されます。第 6 章「ログイン・ページの作成」の「手順 4: ログイン・ページ用の認可の編集」の手順を実行したときに customize 権限を manager ロールに付与しているため、ログイン・ページをカスタマイズできるのは管理者のみです。

「Customize Login Page」ボタンは、ログイン・ページに対して view 権限しか持たないユーザーには表示されません。

アプリケーションのログイン・ページをカスタマイズするには、次の手順を実行します。

1. パスワードが welcome の sking として、Service Request ポータルにログインします。
2. **「Management」**タブをクリックします。このとき、このページにはサイト管理ページのサブタブが組み込まれています（図 9-8 を参照）。
3. **「Site Administration」**サブタブをクリックします。このページには、アプリケーションのスキンを変更するためのオプションとログイン・ページをカスタマイズするためのオプションがあります（図 9-9 を参照）。
4. **「Customize Login Page」**ボタンをクリックします。
5. ログイン・ページで、テキスト領域の上にマウスを移動し、表示されている**アクション・**ボタンをクリックします。表示されたリストから**「Customize」**を選択します。
6. 例 9-7 に示すテキストを、ページ上部のリッチ・テキスト・ポートレットに追加します。

例 9-7 ログイン・ページ上の 1 つ目のリッチ・テキスト・ポートレット

```
Please Log In to the Acme Service Request Application
This demonstration application has a number of predefined user accounts
which equate to different roles within the company.
For example:
   sking - a manager
   ahunold - a technician
   dfaviet - a user (customer)
All usernames should be in lowercase with a  password of  "welcome"
Note: If you copy and paste the user IDs from here, be sure to remove any trailing
spaces to ensure that the ID is valid
```

リッチ・テキスト・エディタの書式設定機能を使用し、図 9-10 に示すような表示になるようにこのテキストを書式設定します。

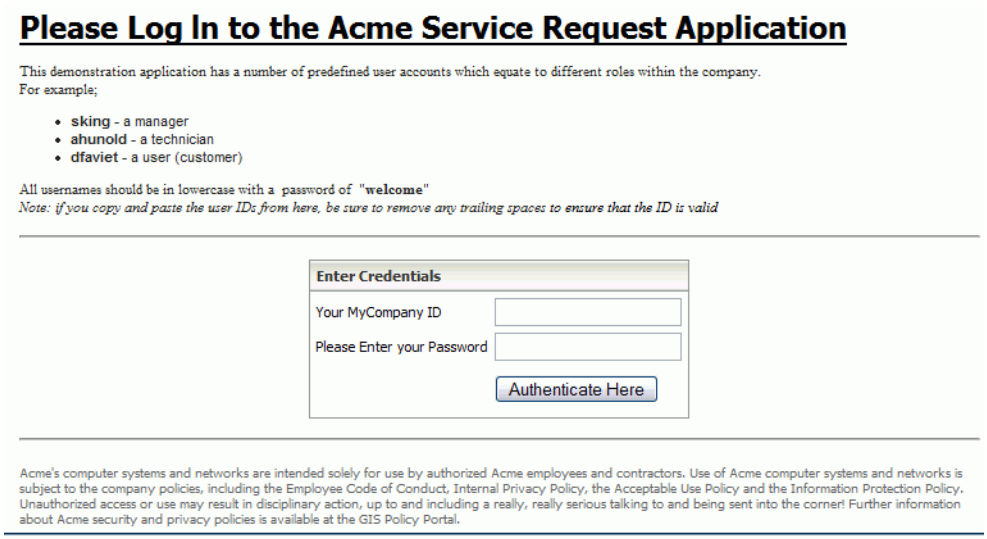
7. 例 9-8 に示すテキストを、ページ上の 2 つ目のリッチ・テキスト・ポートレットに追加します。

例 9-8 ログイン・ページ上の 2 つ目のリッチ・テキスト・ポートレット

```
Acme's computer systems and networks are intended solely for use by authorized
Acme employees and contractors. Use of Acme computer systems and networks is
subject to the company policies, including the Employee Code of Conduct, Internal
Privacy Policy, the Acceptable Use Policy and the Information Protection Policy.
Unauthorized access or use may result in disciplinary action, up to and including
a really, really serious talking to and being sent into the corner! Further
information about Acme security and privacy policies is available at the GIS
Policy Portal.to ensure that the ID is valid
```

このテキストのフォント・サイズを小さくし、色をライト・グレーに設定して、図 9-10 に示すようにテキストが表示されるようにします。

図 9-10 カスタマイズ後のログイン・ページ



8. 「**Return**」をクリックして、サイト管理ページに戻ります。

まとめ

この章では、サイト管理ページを作成し、「Management」ページのサブタブとして追加する方法について学びました。また、サイト管理ページを使用してアプリケーションのルック・アンド・フィールの変更およびログイン・ページのカスタマイズを行う方法についても学びました。

10

アプリケーションのデプロイ

この章の手順では、Application Server Control コンソールを使用してアプリケーションをデプロイする方法を示します。本番環境にアプリケーションを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。

SRDemo アプリケーションを Oracle Application Server またはスタンドアロン OC4J にデプロイするには、次の手順を実行します。

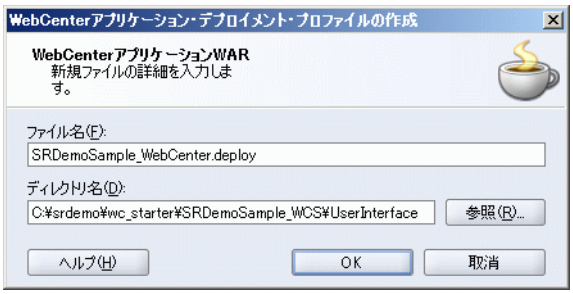
- 手順 1: 汎用 EAR ファイルの作成
- 手順 2: ターゲット EAR ファイルの作成
- 手順 3: ユーザーおよびロールの設定
- 手順 4: サンプル・コンテンツの設定
- 手順 5: アプリケーションのデプロイ
- 手順 6: セキュリティ・ポリシーのデプロイ
- 手順 7: アプリケーションへのアクセス

手順 1: 汎用 EAR ファイルの作成

汎用 EAR ファイルを作成するには、次の手順を実行します。

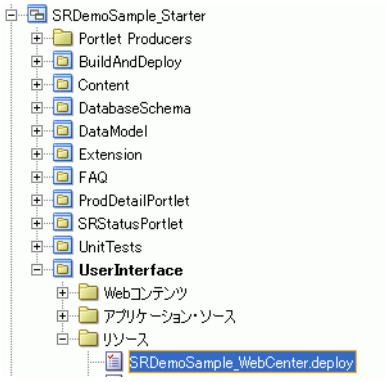
1. Oracle JDeveloper を開いてアプリケーション・ナビゲータに移動し、**「User Interface」**を右クリックし、**「新規」**を選択します。「新規ギャラリ」が表示されます。
2. **「General」**の下の**「Deployment Profiles」**を選択し、**「項目」**で**「WebCenter アプリケーション WAR」**を選択します。「WebCenter アプリケーション・デプロイメント・プロファイルの作成」ダイアログ・ボックスが表示されます。
3. **「ファイル名」**フィールドに、`SRDemoSample_WebCenter.deploy` と入力し（図 10-1 を参照）、**「OK」**をクリックします。

図 10-1 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



4. デプロイメント・プロファイルは、**「User Interface」**の下の**「リソース」**フォルダの中に作成されます（図 10-2 を参照）。

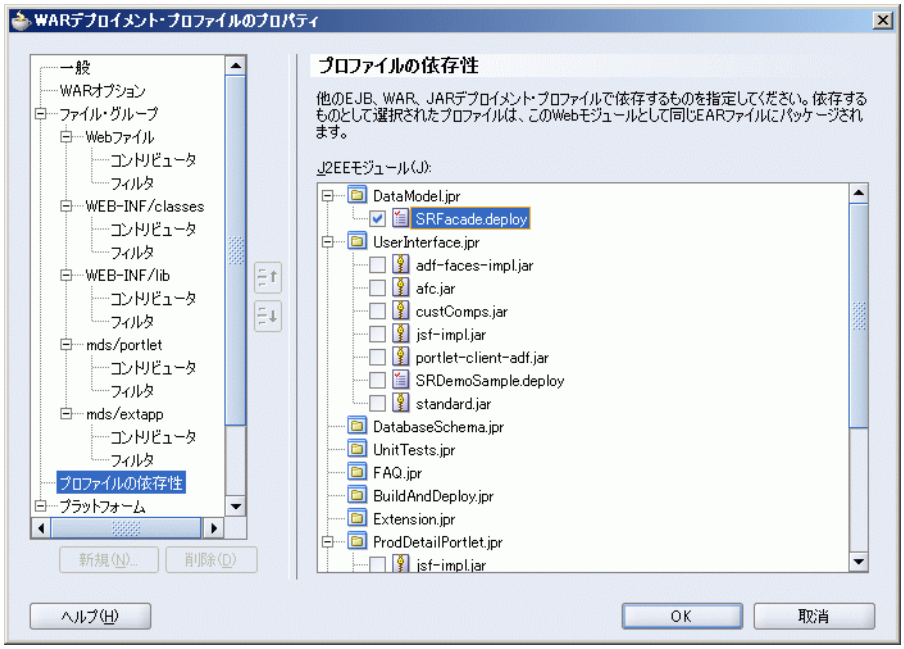
図 10-2 デプロイメント・プロファイル



5. デプロイメント・プロファイルを右クリックし、**「プロパティ」**を選択します。「WAR デプロイメント・プロファイルのプロパティ」ダイアログ・ボックスが表示されます。

6. 「**プロファイルの依存性**」を選択し、「**DataModel.jpr**」に移動して「**SRFacade.deploy**」チェック・ボックスを選択します（図 10-3 を参照）。

図 10-3 SRFacade.deploy



7. 「**OK**」をクリックします。

8. 「**リソース**」の下で、「**SRDemoSample_WebCenter.deploy**」を右クリックして「**EAR ファイルにデプロイ**」を選択します。これで、EAR ファイルが生成されます。「デプロイ - ログ」には、例 10-1 のような出力が表示されます。

例 10-1 デプロイ - ログ

```
----  Deployment started.  ----    May 29, 2007 1:24:32 PM
Target platform is Standard J2EE.
Exporting portlet metadata and customizations
Wrote EJB JAR file to
C:¥SRDemo¥wc_starter¥SRDemoSample_WCS¥DataModel¥deploy¥SRFacade.jar
Wrote WAR file to
C:¥SRDemo¥wc_starter¥SRDemoSample_WCS¥UserInterface¥deploy¥SRDemoSample_WebCenter.war
Wrote EAR file to
C:¥SRDemo¥wc_starter¥SRDemoSample_WCS¥UserInterface¥deploy¥SRDemoSample_WebCenter.ear
Elapsed time for deployment:  25 seconds
----  Deployment finished.  ----    May 29, 2007 1:24:57 PM
```

9. EAR ファイルを探します。

10. EAR ファイルを Oracle Application Server が稼働しているマシンにコピーします。これは、Windows のマップされたドライブを使用するか、FTP を使用して行うことができます。

手順 2: ターゲット EAR ファイルの作成

今度は、Oracle Application Server がインストールされているマシンに切り替えて、次の手順を実行します。

1. コマンド・プロンプトを開きます。
2. 次のように、デプロイ前ツールを実行してターゲット EAR ファイルを作成します。

```
C:¥java¥srdemo_oc4j>C:¥java¥jdev¥jdk¥bin¥java -jar
C:¥java¥srdemo_oc4j¥adfp¥lib¥portlet-client-deploy.jar -predeploy -source
C:¥Temp¥SRDemoSample_WebCenter.ear -target C:¥Temp¥SRDemoSample_WC_target.ear
```

> **注意：** ターゲット EAR を作成する構文は、1 行にする必要があります。この例では、読みやすくするために折り返しています。
>
> この構文の詳細は、『Oracle WebCenter Framework 開発者ガイド』の「WebCenter アプリケーションのデプロイ」を参照してください。

ソース・ファイルは、すでに作成した汎用 EAR ファイルです。

デプロイ前ツールを実行すると、様々な設定を指定するように要求されます。出力は、例 10-2 のようになります（ユーザー定義の設定は**太字**で表示）。

例 10-2 デプロイ前ツールの出力

```
Processing Arguments
Run Mode : 1
Source : C:¥Temp¥SRDemoSample_WebCenter.ear
Target : C:¥Temp¥SRDemoSample_WC_target.ear
Deployed App : null
Mapping File : null
Deployed App : null
Cleaning up C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥
May 16, 2007 12:25:33 PM oracle.adf.share.config.ADFConfigFactory findOrCreateADF
Config
INFO: oracle.adf.share.config.ADFConfigFactory No META-INF/adf-config.xml found
Processing source EAR file
Source EAR file processed
Processing adf-config.xml
adf-config.xml processed
Processing connections.xml
connections.xml processed

Creating a new Deployment Profile for : C:¥Temp¥SRDemoSample_WebCenter.ear

Development MDS Repository Path : ../../mds/;../../
Enter new MDS Repository Path :
C:¥java¥srdemo_oc4j¥mds¥SRDemo

Producer : OmniPortletProducer_11743434494566c52fb80-0111-1000-8001-a9fe02025106

Current Service URL :  http://pdtsdemo1.us.oracle.com:8888/portalTools/omniPortl
et/providers/omniPortlet
Current Proxy URL   :
Current Proxy Port   : 0
Do you want to modify this connection? (Y/N [default=N]) :
N
```

```
Validating producer
OmniPortletProducer_11743434494566c52fb80-0111-1000-8001-a9fe02025106
.
.
.
Do you wish to save this new Deployment Profile (Y/N [default=N]) :
y
Enter a name for this Deployment Profile (e.g. 'Production') :
srdemo_deploy
Enter the path for the Deployment Profile XML file (e.g. 'C:¥profile.xml'):
C:¥temp¥srdemo_deploy.xml
Saving to C:¥temp¥srdemo_deploy.xml
Cleaning up C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥
Extracting View Documents, MDS and connections.xml from source EAR file
Examining contents of SRDemoSample_WebCenter.war
Moving
C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥tmppkg¥views¥SRDemoSample_WebCenter¥WEB-IN
F to
C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥view¥UserInterface¥public_html¥WEB-INF
.
.
.
MDS extracted from source EAR file
Creating target EAR file
Processing source EAR file
.
.
.
Source EAR file processed
Processing adf-config.xml
adf/META-INF/adf-config.xml [Recreated]
adf-config.xml processed
Processing connections.xml
Validating producer
OmniPortletProducer_11743434494566c52fb80-0111-1000-8001-a9fe02025106
.
.
.
adf/META-INF/connections.xml [Recreated]
connections.xml processed
Processing SRDemoSample_WebCenter.war
WEB-INF/lib/META-INF/adf-config.xml [Recreated in WAR file]
WEB-INF/classes/META-INF/connections.xml [Recreated in WAR file]
WEB-INF/lib/META-INF/connections.xml [Recreated in WAR file]
SRDemoSample_WebCenter.war [Recreated]
WAR file processed
Creating target EAR file
WAR file processed
source MDS Path (temp) : C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥mds
Production MDS Path : C:¥java¥srdemo_oc4j¥mds¥SRDemo
Connections.xml Path : C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥connections.xml.new
Export ID : /export
May 16, 2007 12:25:58 PM oracle.mds
NOTIFICATION:
"oracle.portlet.client.persistenceimpl.mds.ImportListener"::"preOperation" is being
invoked.
.
.
.
NOTIFICATION: Transferring the document /UserInterface/public_html/app/staff/SRS
taffSearch.jspx.
NOTIFICATION: import is completed. Total number of documents successfully proces
sed : 24, total number of documents failed : 0.
```

```
Moving C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥SRDemoSample_WC_target.ear t
o C:¥Temp¥SRDemoSample_WC_target.ear
Target EAR C:¥Temp¥SRDemoSample_WC_target.ear created
Cleaning up C:¥DOCUME~1¥ADMINI~1¥LOCALS~1¥Temp¥predeploy¥
```

注意： 例 10-2 では、すべてのポートレット・プロデューサに対する既存の設定をそのまま変更せずにおいてあります。これらの設定は、Y と指定し、新しいプロデューサの場所への適切なパスを指定すると変更できます。

手順 3: ユーザーおよびロールの設定

SRDemo のユーザーおよびロールは、アプリケーションをデプロイする Oracle Application Server にまだ存在していないため、JAZN 移行ツールをレルム・モードで実行して、アプリケーション・サーバーまたはスタンドアロン OC4J の `system-jazn-data.xml` グローバル・ファイルを更新する必要があります。

注意： 開発環境ではこの手順をすでに実行していますが、Oracle Application Server のホスト・システムには Oracle JDeveloper がインストールされていない場合があるため、この手順では、JAZN 移行ツールを直接コールして行う方法を示します。

SRDemo のユーザーおよびロールを追加するには、次の手順を実行します。

1. `system-jazn-data-starter.xml` ファイルを開始時にダウンロードした ZIP ファイルで探します。このファイルには、ユーザーおよびロールが格納されています。
2. OC4J の `system-jazn-data.xml` ファイルをバックアップします。このファイルは、*ORACLE_HOME*`¥j2ee¥home¥config¥system-jazn-data.xml` にあります。
3. コマンド・プロンプトを開きます。
4. 次のようにクラスパスを設定します。

   ```
   set CLASSPATH = ORACLE_HOME/j2ee/home/jazn.jar;ORACLE_HOME/BC4J/lib/adfshare.jar
   ```

   たとえば、ORACLE_HOME が `C:¥java¥srdemo_oc4j` の場合、次のようにクラスパスを設定します。

   ```
   set CLASSPATH = C:¥java¥srdemo_oc4j/j2ee/home/jazn.jar; C:¥java¥srdemo_oc4j/BC4J/
   lib/adfshare.jar
   ```

5. JAZN 移行ツールを次のように実行します（先ほどと Oracle ホームの場所が同じであることが前提です）。

   ```
   java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com -dr jazn.com -st
   xml -dt xml -sf C:¥Temp¥system-jazn-data-starter.xml -df C:¥java¥srdemo_
   oc4j¥j2ee¥home¥config¥system-jazn-data.xml -m realm
   ```

手順 4: サンプル・コンテンツの設定

開発環境と同じ方法で、Oracle Application Server のホスト・システムにサンプル・コンテンツを設定する必要があります。そのためには、srcontentrepository.zip の内容を C:¥SDemo_App_Download から C:¥srdemo に抽出します。

注意： UNIX を使用している場合は、connections.xml ファイルをコンテンツ・リポジトリへのパスで更新する必要があります。connections.xml ファイルを開いて C:¥srdemo¥SRContentRepository を探し、パスを更新してサンプル・コンテンツを抽出したディレクトリを指すようにします。

Windows を使用し、推奨ディレクトリにコンテンツを抽出しなかった場合も、connections.xml ファイルを選択したパスで更新する必要があります。

手順 5: アプリケーションのデプロイ

ターゲット EAR ファイルの生成は済んでいるので、デプロイする準備ができました。デプロイは、次のように Application Server Control コンソールを使用して行います。

1. Application Server Control コンソールにアクセスするには、URL の http://*<host_name>.<domain>:<port>*/em. に移動します。

   たとえば、http://test.acme.com:8888/em などです。

   Application Server Control コンソールの正確の URL を見つけるには、readme.txt を参照してください。インストール後、このテキスト・ファイルは、次の Oracle Application Server の場所に保存されます。

   UNIX の場合 : *ORACLE_HOME*/install/readme.txt

   Windows の場合 : *ORACLE_HOME*¥install¥readme.txt

2. Application Server Control コンソールにログインします。「クラスタ・トポロジ」ページが表示されます。

3. 「クラスタ・トポロジ」ページで、アプリケーション・サーバーへのリンクをクリックします。

4. **「アプリケーション」**タブ（図 10-4）を選択し、**「デプロイ」**をクリックします。「デプロイ：アーカイブの選択」ページが表示されます。

図 10-4 Application Server Control コンソールの「アプリケーション」タブ

5. **「アーカイブはローカル・ホストに存在します。」**オプションを選択し、EAR ファイルの場所を参照します。次に、**「次へ」**をクリックします。
6. 「デプロイ：アプリケーション属性」ページで、Web アプリケーションのコンテキスト・ルートをアプリケーション名として入力するか、デプロイメント・プロファイルの作成時に構成したものを入力します。
7. 「デプロイ設定」ページが表示されます。必要に応じてこのページの値を変更し、**「デプロイ」**をクリックします。
8. デプロイの確認ページが表示され、図 10-3 のような出力が表示されます。

例 10-3 デプロイの確認の出力

```
[May 16, 2007 5:05:11 PM] Application Deployer for SRDemo STARTS.
[May 16, 2007 5:05:11 PM] Copy the archive to
C:¥java¥srdemo_oc4j¥j2ee¥home¥applications¥SRDemo.ear
[May 16, 2007 5:05:12 PM] Initialize C:¥java¥srdemo_
oc4j¥j2ee¥home¥applications¥SRDemo.ear begins...
[May 16, 2007 5:05:12 PM] Unpacking SRDemo.ear
[May 16, 2007 5:05:15 PM] Done unpacking SRDemo.ear
[May 16, 2007 5:05:15 PM] Unpacking SRDemoSample_WebCenter.war
[May 16, 2007 5:05:25 PM] Done unpacking SRDemoSample_WebCenter.war
[May 16, 2007 5:05:26 PM] Initialize C:¥java¥srdemo_
oc4j¥j2ee¥home¥applications¥SRDemo.ear ends...
[May 16, 2007 5:05:26 PM] Starting application : SRDemo
[May 16, 2007 5:05:26 PM] Initializing ClassLoader(s)
[May 16, 2007 5:05:26 PM] Initializing EJB container
[May 16, 2007 5:05:26 PM] Loading connector(s)
[May 16, 2007 5:05:27 PM] Starting up resource adapters
[May 16, 2007 5:05:27 PM] Processing EJB module: SRFacade.jar
[May 16, 2007 5:05:29 PM] Initializing EJB sessions
[May 16, 2007 5:05:29 PM] Committing ClassLoader(s)
[May 16, 2007 5:05:29 PM] Initialize SRDemoSample_WebCenter begins...
[May 16, 2007 5:05:29 PM] Initialize SRDemoSample_WebCenter ends...
[May 16, 2007 5:05:29 PM] Started application : SRDemo
[May 16, 2007 5:05:29 PM] Binding web application(s) to site default-web-site begins...
```

```
[May 16, 2007 5:05:29 PM] Binding SRDemoSample_WebCenter web-module for
application SRDemo to site default-web-site under context root SRDemo
[May 16, 2007 5:06:54 PM] Initializing Servlet: javax.faces.webapp.FacesServlet for web
application SRDemoSample_WebCenter
[May 16, 2007 5:06:55 PM] Initializing
Servlet:oracle.adfinternal.view.faces.renderkit.html.portlet.ADFPortletServlet for web
application SRDemoSample_WebCenter
[May 16, 2007 5:06:55 PM] Binding web application(s) to site default-web-site ends...
[May 16, 2007 5:06:55 PM] Application Deployer for SRDemo COMPLETES. Operation time:
103484 msecs
```

9. **「アプリケーション」**タブに移動し、アプリケーションを選択します。次に、**「管理」**をクリックします。
10. **「管理タスク」**で、**「クラスのロードの構成」**をクリックします。
11. **「Apache.commons.logging」**の選択を解除します。

これで、アプリケーションはデプロイされましたが、まだアクセスできません。まず、セキュリティ・ポリシーをデプロイする必要があります。

注意 : この例では、サンプルが依然として同じデータベースに存在すると想定しています。ターゲット EAR ファイルは、開発環境で構成したデータベース接続を使用するように構成されます。この接続が存在しない場合は、Application Server Control コンソールを使用して、開発環境で使用したのと同じ名前でデータベース接続を新規作成する必要があります。データベース・スクリプトの実行の詳細は、インストール・ファイルの完成したデモの実行に関する項を参照してください。

手順 6: セキュリティ・ポリシーのデプロイ

セキュリティ・ポリシーは、app-jazn-data.xml ファイル内のアプリケーションの EAR ファイルにバンドルされます。アプリケーションのポリシーをデプロイするには、次の手順を実行します。

1. app-jazn-data.xml ファイルを探します。ORACLE_HOME が C:¥java¥srdemo_oc4j である場合、このファイルは、C:¥java¥srdemo_oc4j¥j2ee¥home¥applications¥SRDemo¥adf¥META-INF¥app-jazn-data.xml にあります。
2. コマンド・プロンプトを開きます。
3. 次のようにクラスパスを設定します。

   ```
   set CLASSPATH = ORACLE_HOME/j2ee/home/jazn.jar;ORACLE_HOME/BC4J/lib/adfshare.jar
   ```

   たとえば、ORACLE_HOME が C:¥java¥srdemo_oc4j の場合、次のようにクラスパスを設定します。

   ```
   set CLASSPATH = C:¥java¥srdemo_oc4j/j2ee/home/jazn.jar; C:¥java¥srdemo_oc4j/BC4J/
   lib/adfshare.jar
   ```

4. JAZN 移行ツールを次のように実行します（C:¥java¥srdemo_oc4j が Oracle ホームの場所であることが前提です）。

   ```
   java oracle.security.jazn.tools.JAZNMigrationTool -sr jazn.com -dr jazn.com
   -st xml -dt xml -sf C:¥java¥srdemo_
   oc4j¥j2ee¥home¥applications¥SRDemo¥adf¥META-INF¥app-jazn-data.xml -df
   C:¥java¥srdemo_oc4j¥j2ee¥home¥config¥system-jazn-data.xml -m policy
   ```

5. OC4J を再起動します。

ポリシー情報が移行されます。これで、アプリケーションの実行準備ができました。

手順 7: アプリケーションへのアクセス

今度は、SRDemo にアクセスできます。そのためには、次の手順を実行します。

1. ブラウザを開き、次の URL に移動します。

   ```
   http://appserver_host_name:appserver_port_number/SRDemo/faces/app/SRWelcome.jspx
   ```

2. ログインして SRDemo を表示します。

まとめ

この章では、アプリケーションを取得してデプロイする方法を示しました。この時点で、アプリケーションを実行し、このマニュアルを通して作成したページおよびコンポーネントを表示できます。また、作成したアプリケーションとサンプル・ファイル内に付属する完成バージョンを比較することもできます。比較するには、install.html ファイルの指示に従って完成したデモを実行します。

索引

A

ADF，「Application Development Framework」を参照，1-6
ADF 表
　SelectOneChoice の連結，7-13
　ポートレットの連結，7-16
Application Development Framework（ADF），1-6

C

Content DB
　概要，1-5

J

JavaServer Faces ドロップダウン・コンポーネント
　ページへの追加，7-14
JAZN 移行ツール
　アプリケーションのデプロイ，10-6
JCR アダプタ，1-4
JCR データ・コントロール，1-4
JSR 168 ポートレット
　アプリケーション・サーバーへのデプロイ，2-12
　作成，2-5
　プロデューサの登録，2-15
　ロジックの追加，2-10

O

ObjectSpacer
　追加，7-9
OmniPortlet
　HTML レイアウトによる定義，8-13，8-14
　SQL データソースによる定義，7-14
　円グラフ・レイアウトによる定義，8-12，8-13
　説明，1-4
　表レイアウトによる定義，8-11
　プロデューサの登録，7-5
　ページへのインスタンスの追加，8-9
Oracle ADF セキュリティ・ウィザード，6-17
Oracle Content Database
　概要，1-5
Oracle JDeveloper
　ADF，1-6
Oracle WebCenter Suite
　内容，1-3

P

PanelCustomizable
　追加，7-6
PanelHorizontal
　使用，7-9
　追加，7-11
PDK-Java ポートレット
　アプリケーション・サーバーへのデプロイ，2-28
　構築，2-17
　ページへの追加，7-16
　ロジックの追加，2-23

S

Schedule On-Site Services ポートレット
　作成，7-20
SelectItem
　追加，7-7
SelectOneChoice
　ADF 表への連結，7-13
　追加，7-7
　ページへの追加，8-10
Service Request デモ
　説明，1-7
ShowDetailFrame
　追加，7-6
ShowDetailItem
　追加，7-10
ShowOneTab
　追加，7-10
SRLogin.jspx
　作成，6-2
SRMyAcme.jspx
　Web クリッピング・ポートレットの追加，7-20
　カスタマイズ・コンポーネントの追加，7-5
SRSiteAdmin.jspx
　作成，9-2

U

URL
　Web クリッピング用の指定，7-21

W

WebCenter Framework
　説明，1-3

WebCenter Services
概要，1-5
WebCenter アプリケーション
WAR，1-5
web.xml
更新，6-19
Web クリッピング・ポートレット
Web ページの選択，7-21
作成，7-20
説明，1-4
追加，7-21
登録，7-20
Web コンテンツ
指定，7-21

あ

アプリケーション
Application Server Control コンソールの使用，10-7
JAZN 移行ツールを使用したデプロイ，10-6
セキュリティ・ポリシーのデプロイ，10-9
デプロイ，10-1
デプロイ前ツールの使用，10-4
ユーザーおよびロールの設定，10-6
アプリケーション・サービス
接続，2-3

い

インタラクティビティ
ページへの追加，7-16

か

カスタマイズ
サイト管理ページ，9-11
カスタマイズ・コンポーネント
SRMyAcme.jspx ページへの追加，7-5
ページ・レイアウトの作成のための使用，8-4

け

検索
ページへの検索コンポーネントの追加，8-11

こ

コンテンツ・リポジトリ
JCR データ・コントロールの構成，4-4
所定の場所でのファイルの表示，5-4
設定，4-1
コンポーネント
ADF 表への SelectOneChoice の連結，7-13
JavaServer Faces ドロップダウン・コンポーネントの追加，7-14
SelectOneChoice の追加，8-10
セキュリティの追加，7-26
ページ上のコンポーネントの連結，8-15
ページへの ADF 表の追加，7-24
ページへの検索コンポーネントの追加，8-11

さ

サイト管理ページ
作成，9-1
スキンの変更，9-11
ログイン・ページのカスタマイズ，9-12

す

スキン
アプリケーションの構成，3-3
使用，9-11
選択，9-11
登録，3-2
スキン・セレクタ
追加，9-3

せ

セキュリティ
コンポーネントへの適用，7-26
ページへの追加，7-25
セキュリティ・ポリシー
デプロイ，10-9
接続
アプリケーション・サーバーへの接続の作成，2-3

た

ターゲット EAR ファイル
作成，10-4
ダッシュボード・ページ
概要，8-1
作成，8-2

て

データ・コントロール
構成，4-4
デプロイ
Application Server Control コンソールの使用，10-7
ターゲット EAR ファイルの作成，10-4
汎用 EAR ファイルの作成，10-2
デプロイ前ツール
使用，10-4

に

認可
ログイン・ページ用の編集，6-16

は

汎用 EAR ファイル
作成，10-2

ふ

プロデューサ
OmniPortlet プロデューサの登録，7-5
PDK-Java ポートレット・プロデューサの作成，2-17
Web クリッピング・プロデューサの登録，7-20
登録，2-15

へ

ページ
ADF 表の追加，7-24
OmniPortlet インスタンスの追加，8-9
PDK-Java ポートレットの追加，7-16
インタラクティビティの追加，7-16
カスタマイズ・コンポーネントの使用，8-4
サイト管理ページの作成，9-1
作成，7-2
セキュリティの追加，7-25
ダッシュボードの作成，8-2
ダッシュボード・ページの概要，8-1
ページ上のコンテンツの連結，8-15
ページ・レイアウト
作成，8-4

ほ

ポートレット
ADF 表への連結，7-16
HTML レイアウトによる OmniPortlet の定義，8-13，8-14
JSR 168 ポートレットの作成，2-5
JSR 168 ポートレットのデプロイ，2-12
JSR 168 ポートレット・プロデューサの登録，2-15
JSR 168 ポートレットへのロジックの追加，2-10
OmniPortlet の定義，7-14
PDK-Java ポートレットおよびプロデューサの作成，2-17
PDK-Java ポートレットの構築，2-17
PDK-Java ポートレットのデプロイ，2-28
PDK-Java ポートレットへのロジックの追加，2-23
Web クリッピング・ポートレットの追加，7-20
円グラフ・レイアウトによる OmniPortlet の定義，8-12，8-13
表レイアウトによる OmniPortlet の定義，8-11
プロジェクトの作成，2-2
ページへの OmniPortlet インスタンスの追加，8-9
ページへの PDK-Java ポートレットの追加，7-16

ゆ

ユーザーおよびロール
アプリケーション用の設定，10-6

よ

ようこそページ
グローバル・ナビゲーション・リンクの追加，5-8
作成，5-1
所定の場所でのファイルの表示，5-4
パブリックなアクセス化，5-11
リッチ・テキスト・コンポーネントの追加，5-2

ら

ライフサイクル・サポート
概要，1-5

り

リッチ・テキスト・コンポーネント
ようこそページへの追加，5-2
リッチ・テキスト・ポートレット
説明，1-4

れ

例
説明，1-7
レイアウト・コンポーネント
SRMyAcme.jspx ページへの追加，7-5

ろ

ロール
アプリケーション用の設定，10-6
ログイン・ページ
getLoginFormBlock getter メソッド，6-8
getloginScriptBlock getter メソッド，6-6
getloginStyleBlock getter メソッド，6-6
Oracle ADF セキュリティ・ウィザード，6-17
アプリケーションの構成，6-17
カスタマイズ，9-12
作成，6-1，6-2
認可の編集，6-16
認証設定，6-17
リッチ・テキスト・ポートレットの追加，6-15
ログイン・エラー・ページの作成，6-12