Oracle® Application Development Framework

チュートリアル

10*g* リリース 3（10.1.3）

部品番号 : B28911-01

2006 年 6 月

ORACLE®

Oracle Application Development Framework チュートリアル , 10*g* リリース 3（10.1.3）

部品番号 : B28911-01

原本名 : Oracle Application Development Framework Tutorial, 10*g* Release 3 (10.1.3)

原本著者 : Jeff Gallus, Gary Williams, Kate Heap

原本協力者 : Lynn Munsinger, Duncan Mills, Frank Nimphius

目次

はじめに v

対象読者 vi
このマニュアルの構成 vi
関連ドキュメント vii
サポートおよびサービス vii

1 チュートリアルを始める前に

チュートリアルの流れ : 概要 1-2
業務上の問題 1-2
業務目標 1-2
業務上の解決策 1-2
デザイン・パターンおよびアーキテクチャ・フレームワーク 1-3
このチュートリアルの使用方法 1-4
環境の設定 1-4
スキーマのインストールの準備 1-5
サービス・リクエスト・スキーマのインストール 1-6
Oracle JDeveloper の起動 1-7
SRDEMO スキーマにアクセスするためのデータベース接続の作成 1-7
JDeveloper でのアプリケーションとそのプロジェクトの定義 1-10
まとめ 1-12

2 データ・モデルの開発

概要 2-2
Oracle TopLink を使用したデータ・モデルの作成 2-2
データベース・オブジェクトに対する TopLink マッピングの作成 2-2
TopLink 定義の調整 2-5
デフォルト値を管理するためコードを追加 2-6
TopLink 名前付き問合せの作成 2-7
名前付き問合せの作成 2-8
問合せ式の定義 2-9
追加の名前付き問合せの作成 2-10
TopLink セッションの作成 2-12
EJB セッション Bean の作成 2-13
セッション Bean のカスタマイズ 2-15
ADF データ・コントロールの作成 2-16
アプリケーションの作成 2-17

まとめ 2-17

3 ページ・フローおよびナビゲーションの定義

概要 3-2
JSF ナビゲーション・モデルの作成 3-2
ダイアグラムでのページの作成 3-4
ページのリンク 3-6
グローバル・ナビゲーション・ルールの定義 3-8
まとめ 3-9

4 アプリケーション標準の開発

概要 4-2
コードの再利用 4-2
ユーザー・インタフェースの翻訳の実行 4-3
状態ホルダーの提供 4-4
標準のルック・アンド・フィールの作成 4-6
ページへのコンポーネントの追加 4-7
まとめ 4-10

5 単純な表示ページの開発

概要 5-2
ページ・アウトラインの作成 5-4
ページへのユーザー・インタフェース・コンポーネントの追加 5-5
「Edit」ボタンの組込み 5-6
「View」ボタンの組込み 5-9
リフレッシュ動作の定義 5-9
ページへのメニュー・バーの追加 5-11
ドリルダウン・リンクの追加 5-13
ページの実行 5-15
まとめ 5-15

6 ログイン・セキュリティの実装

概要 6-2
セキュリティを制御するクラスの作成 6-3
ロールを管理するクラスの作成 6-3
認証を与えるクラスの作成 6-4
ユーザーのアプリケーションへの統合 6-6
ユーザーを管理する名前付き問合せの作成 6-6
ユーザーの電子メール ID のアプリケーションへの公開 6-7
コンテナ・セキュリティの設定 6-9
アプリケーションへのアクセスの設定 6-10
まとめ 6-12

7 検索ページの開発

概要 7-2
検索ページの作成 7-2
名前付き問合せの作成 7-4

問合せ式の作成 7-5
DataControl の再作成 7-6
検索ページへのデータ・コンポーネントの追加 7-7
データ・コンポーネントの追加 7-8
問合せのデフォルトの動作の変更 7-9
リフレッシュ条件の追加 7-10
「Edit」ボタンの組込み 7-10
バッキング Bean コードの追加 7-11
「View」ボタンの組込み 7-12
まとめ 7-13

8 マスター / ディテール・ページの開発

概要 8-2
基本の UI の開発 8-2
サービス・リクエスト・コンポーネントの追加 8-4
データベース対応コンポーネントの追加 8-5
レイアウトの調整 8-6
ノート・パネルの追加 8-7
ノートの追加セクションの作成 8-8
コードの値の導出 8-9
サービス履歴パネルの追加 8-11
まとめ 8-12

9 トランザクション機能の実装

概要 9-2
作成ページの開発 9-3
作成ページの変更 9-4
ページへのデータ・コンポーネントの追加 9-7
ユーザーが入力した問題の説明の保存 9-9
再利用可能なメソッドの作成 9-11
マネージド Bean へのバインディングの割当て 9-12
確認ページの作成 9-12
確認ページの変更 9-14
表示の調整 9-14
ProductId の表示 9-15
製品の説明の表示 9-16
トランザクションの制御 9-17
ボタンへのメソッドのバインディング 9-18
データのコミットおよびサービス・リクエスト ID のリターン 9-20
完了ページの作成 9-22
完了ページの変更 9-23
FAQ ページの作成 9-25
FAQ ページの変更 9-25
プロセス・トレインの組込み 9-27
ページの実行 9-30
まとめ 9-30

10 編集ページの開発

概要 10-2
ページ・アウトラインの作成 10-4
ページへの UI 要素の追加 10-5
createdBy 名および assignedTo 名を取得するルックアップの作成 10-7
入力サービス・リクエスト ID パラメータの組込み 10-8
status 属性用のドロップダウン・リストの追加 10-9
「Cancel」ボタンの組込み 10-10
「Save」ボタンの組込み 10-11
クローズドのサービス・リクエストに対する入力フィールドの無効化 10-11
ページの実行 10-12
アプリケーションのルック・アンド・フィールの変更 10-13
まとめ 10-14

11 Oracle Application Server 10*g* へのアプリケーションのデプロイ

概要 11-2
Oracle Application Server 10*g* への接続の作成 11-3
J2EE（OC4J）インスタンスの OracleAS コンテナの起動 11-4
OC4J への接続の作成 11-5
Enterprise Manager の起動 11-6
デプロイメントの準備 11-7
デプロイメント・プロファイルの作成 11-9
UserInterface デプロイメント・プロファイルの作成 11-10
アーカイブの作成 11-11
接続プールおよびデータソースの作成 11-12
接続プールの作成 11-13
アプリケーションのデプロイ 11-14
アプリケーションのテスト 11-14
Enterprise Manager を使用したアプリケーション・サーバーの参照 11-15
まとめ 11-16

はじめに

ここでは、『Oracle Application Development Framework チュートリアル』の内容のおよび対象読者について概要を説明します。

内容は次のとおりです。

- 対象読者
- このマニュアルの構成
- 関連ドキュメント
- サポートおよびサービス

対象読者

このチュートリアルは、Oracle Application Development Framework（ADF）を使用して Web アプリケーションを作成する J2EE 開発者を対象としています。

このマニュアルの構成

チュートリアルは、次の章で構成されています。

第 1 章「チュートリアルを始める前に」
この章では、サービス・リクエストの流れおよびスキーマのインストールについて説明します。

第 2 章「データ・モデルの開発」
この章では、Oracle TopLink および Oracle ADF を使用して、アプリケーションのデータ・モデルを作成する方法について説明します。

第 3 章「ページ・フローおよびナビゲーションの定義」
この章では、JSF アプリケーション内のスケルトン・ページの作成方法、およびページ間のナビゲーションの定義方法について説明します。

第 4 章「アプリケーション標準の開発」
この章では、アプリケーション開発における標準の役割について説明し、標準の SRDemo アプリケーションへの実装方法を示します。

第 5 章「単純な表示ページの開発」
この章では、SRDemo アプリケーションの中心となる単純な表示ページの作成方法について説明します。このページを使用するとサービス・リクエストの情報を表示できます。

第 6 章「ログイン・セキュリティの実装」
この章では、SRDemo アプリケーションのセキュリティを作成する方法について説明します。

第 7 章「検索ページの開発」
この章では、検索ページの作成方法について説明します。ページには 2 つのセクションがあり、一方は問合せ基準の指定に、もう一方は結果の表示に使用されます。

第 8 章「マスター / ディテール・ページの開発」
この章では、サービス・リクエストおよびサービス・リクエスト履歴行を表示する、マスター / ディテール・ページを開発します。

第 9 章「トランザクション機能の実装」
この章では、サービス・リクエストを作成するページの作成方法について説明します。サービス・リクエストのプロセスは、3 つの主なページ、つまり製品と問題を指定するページ、値を確認するページ、サービス・リクエスト ID をコミットおよび表示するページで構成されています。4 つ目のページを作成することもできます。このページには、一般的な製品の問題の解決に関する FAQ を表示します。

第 10 章「編集ページの開発」
この章では、マネージャおよび技術者がサービス・リクエストを編集できるページを作成する方法について説明します。

第 11 章「Oracle Application Server 10g へのアプリケーションのデプロイ」
この章では、JDeveloper を使用して、アプリケーションおよび必要なデプロイメント・ディスクリプタを含む、デプロイ可能なパッケージを作成します。次に、パッケージをデプロイします。

関連ドキュメント

Oracle ADF を使用したアプリケーション作成の詳細は、次のドキュメントを参照してください。

- 『Oracle Application Development Framework 開発者ガイド』
- 『Oracle Application Development Framework Developer's Guide for Forms/4GL Developers』

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。

オラクル社カスタマ・サポート・センター

オラクル製品サポートの購入方法、およびオラクル社カスタマ・サポート・センター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

チュートリアルを始める前に

このチュートリアルでは、Oracle JDeveloper、Oracle ADF および Oracle TopLink を使用したエンドツーエンドの J2EE Web アプリケーションの作成方法について説明します。アプリケーションでは、Enterprise JavaBeans（EJB）や JavaServer Faces（JSF）などの様々な J2EE テクノロジが使用されます。このチュートリアルでは、アプリケーションのユーザー・インタフェース、およびアプリケーションのナビゲーション制御のための JSF の使用方法を学びます。

この章の内容は次のとおりです。

- チュートリアルの流れ : 概要
- このチュートリアルの使用方法
- Oracle JDeveloper の起動
- SRDEMO スキーマにアクセスするためのデータベース接続の作成
- JDeveloper でのアプリケーションとそのプロジェクトの定義
- まとめ

チュートリアルの流れ : 概要

ServiceCompany は、大型の家庭用電化製品（食器洗い機、洗濯機、電子レンジなど）のサービス・サポートを行う会社です。多様な電化製品のサポートを行っており、Web 上でサービス・リクエストに対応することで、大部分の顧客の問題を解決しようとしています。

ServiceCompany は長年の経験から、適切な情報さえあればほとんどの問題は顧客が解決できることを知りました。この手法により、会社と顧客両方の時間とお金を節約できることがわかりました。サービス・リクエストは、顧客、技術者またはマネージャがリクエストするときに作成できます。

従業員がオープンしたサービス・リクエストは、ある製品に関連するいろいろなタイプの社内情報（製品のリコール、製品固有の問題など）になります。

業務上の問題

ServiceCompany では、過去 2 四半期、サービス・リクエストが増加しているにもかかわらず、その解決は減少しています。その結果、顧客にとってより便利なシステム、およびより効率的で迅速なリクエスト解決サービスを導入することを決定しました。

ServiceCompany は、サービス・リクエストの経路やサービス・リクエストのタイプにかかわらず、すべての顧客に同じサービスとレスポンスを提供できることを目標としています。

業務目標

ServiceCompany の目標は次のとおりです。

- 製品に関連するサービス・リクエストを記録および追跡する。
- サービス・リクエストをスムーズに効率的かつ迅速に解決する。
- 割当てを管理し、すべてのサービス・リクエストの進捗状況を記録する。
- サービス・リクエストの処理を完全に自動化する。
- マネージャがサービス・リクエストを適切な技術者に割り当てられるようにする。
- 顧客と技術者がサービス・リクエストを記録できるようにする。
- 技術者の製品の専門分野を追跡する。

業務上の解決策

ServiceCompany は、Oracle Application Development Framework（ADF）を使用して作成する新しい、完全自動化されたシステムを導入することを決定しました。これにより、標準ベースの J2EE アプリケーション構造の開発の生産性が高くなります。アプリケーション・サーバーは Oracle Application Server 10*g* です。

新しいアプリケーションの主要な構成要素は次のようになります。

- あらゆるユーザー（顧客、技術者およびマネージャ）がサービス・リクエストのステータスを追加、更新および確認できる顧客用のインタフェース。
- 会社がサービス・リクエストを作成、更新および管理できるユーザー・インタフェース。これには、適切な技術者へのリクエストの割当て、および累積履歴情報の収集が含まれます。
- サービス・リクエストを適時解決できるようにする様々なレポート・ツール。
- 技術者が自身の製品専門分野を更新できるユーザー・インタフェース。

次のプロセスは、顧客が生成するサービス・リクエストの計画フローです。

1. 顧客は Web インタフェースを使用してリクエストを発行します。
2. マネージャはリクエストを技術者に割り当てます。
3. 技術者はリクエストを確認し、解決法を提供するか、詳細な情報を顧客に尋ねます。
4. 顧客はリクエストを確認し、リクエストをクローズするか、詳細な情報を顧客に尋ねます。
5. マネージャは既存のリクエストについて技術者に確認し、（必要な場合は）別の技術者にそのリクエストを再割当てします。
6. 技術者は専門分野の製品を特定します。マネージャはこの情報を使用してサービス・リクエストを割り当てられます。

アプリケーションの作成に使用されるテクノロジは、次のとおりです。

- 使用されるテクノロジは Oracle ADF です。
- データは Oracle Database 10*g* に格納されます。
- データ・モデルおよびビジネス・ロジックは、Oracle TopLink および Enterprise JavaBeans を使用して実装されます。
- データバインディング（クライアント・コンポーネントとビジネス・ロジックの間のマッピング）は Oracle ADF によって提供されます。
- Web クライアント・レイヤーは、JSF ページおよび ADF Faces コンポーネントを使用して作成されます。
- 認可は、J2EE コンテナ・セキュリティに基づきます。
- アプリケーションは Oracle Application Server 10*g* にデプロイします。

デザイン・パターンおよびアーキテクチャ・フレームワーク

アプリケーション開発時のよい方法は、デザイン・パターンを使用することです。デザイン・パターンは、アプリケーション間、および開発者間でオブジェクト指向概念を再利用する便利な方法です。デザイン・パターンの考え方は簡単で、オブジェクト間の共通動作パターンのドキュメントおよびカタログです。開発者は、再作成せずにこれらのパターンを使用することができます。

デザイン・パターンの他に、開発者は標準的な方法で実行するアプリケーションの作成にアーキテクチャ・フレームワークを使用することがよくあります。頻繁に使用されるアーキテクチャ・パターンの 1 つが、Model-View-Controller（MVC）パターンです。

MVC アーキテクチャでは、ユーザー入力、ビジネス・ロジックおよびユーザーへのビジュアル・フィードバックが 3 種類のオブジェクトによって明示的に分類され、処理されます。各オブジェクトは、アプリケーション内で特定の役割を持ちます。

- **ビュー**は、ユーザーに対するアプリケーション出力のプレゼンテーションを管理します。
- **コントローラ**は、ユーザーのマウスおよびキーボード入力を解釈し、必要に応じて変更するようモデルまたはビュー、あるいはその両方に命令をします。
- **モデル**は、アプリケーション・ドメインのデータを管理し、その状態の情報についてのリクエスト（通常ビューから）に対して応答し、状態を変更する指示（通常コントローラから）に対して応答します。

これら 3 つのコンポーネントを明確に分類することがよい設計の重要なポイントです。

J2EE アプリケーションのその他の一般的なデザイン・パターンは、セッション・ファサード・パターンです。セッション・ファサード・パターンは、クライアントのビューからのアプリケーション・コンポーネント間の複雑な相互作用を隠します。アプリケーション・ワークフローに含まれるビジネス・ロジックをカプセル化するので、アプリケーション・コンポーネント間の相互作用を単純化します。セッション Bean（セッション・ファサードを示す）は、ビジネス・オブジェクト間の関係を管理します。セッション Bean は、アプリケーションの必要に応じて関係するオブジェクトを作成、検索、変更および削除することで、そのライフ・サイクルも管理します。

このチュートリアルでは、アプリケーションのモデルでセッション・ファサード・パターンを使用し、Oracle ADF および JavaServer Faces を使用する際に MVC アーキテクチャを使用します。

このチュートリアルの使用方法

このチュートリアルは、章構成になっており、各章はその前の章に基づいて記載されています。チュートリアルに記載された順に各章を実行してください。

この項では、アプリケーションの作成を始める前に行っておく必要がある前提条件となるすべての手順について説明します。

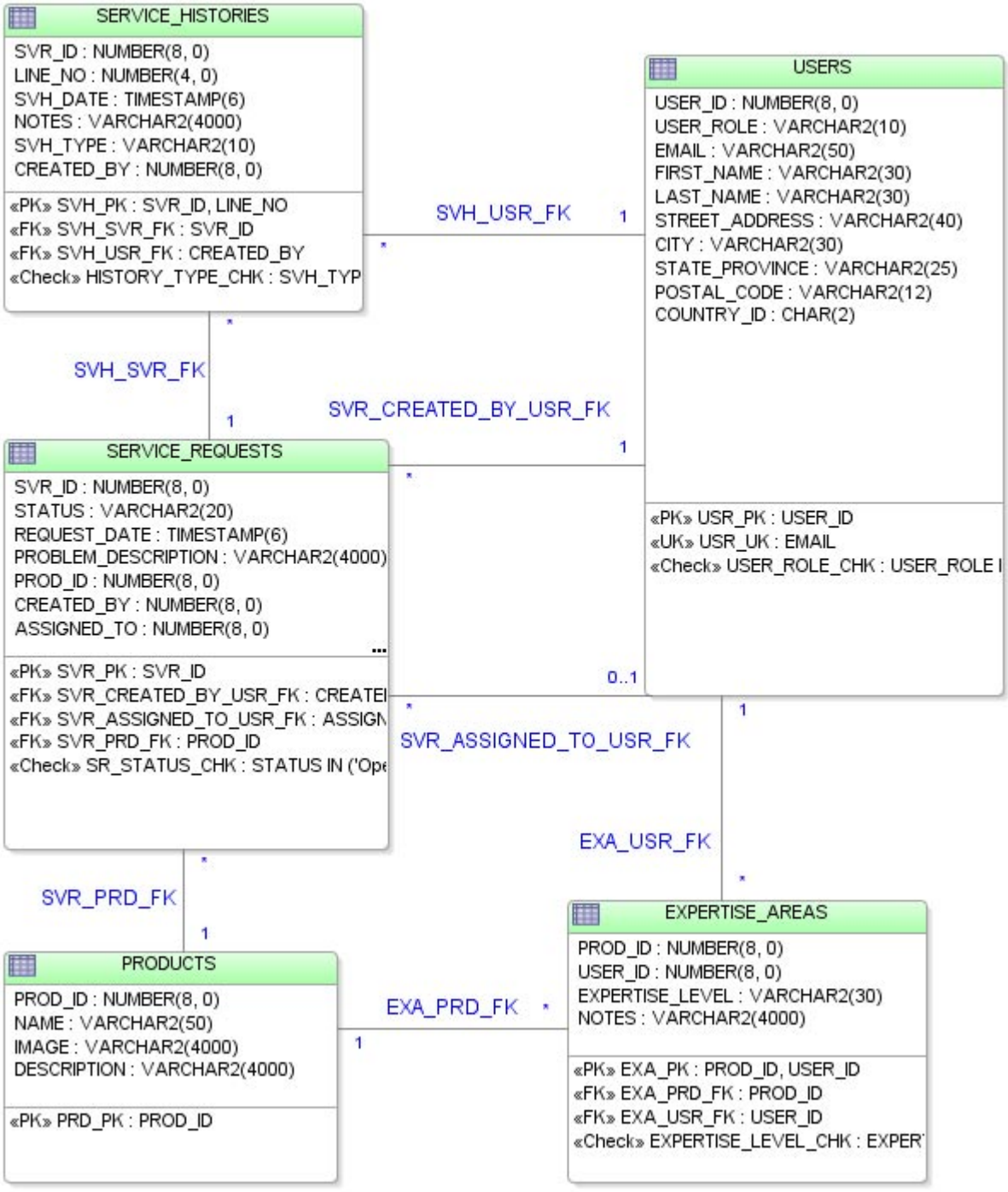
環境の設定

チュートリアルのアプリケーションをサポートするよう作業環境を準備する必要があります。これには、次の主な作業を実行します。

- スキーマのインストールの準備
- `SRDEMO` スキーマ所有者の作成およびサービス・リクエスト・スキーマのインストール
- Oracle JDeveloper 10*g* リリース 3 の起動
- JDeveloper データベース接続の作成
- チュートリアル用のアプリケーションおよびプロジェクトの定義

スキーマのインストールの準備

スキーマは、5 つの表と 3 つのデータベース順序で構成されています。表は次のように図示されます。



5 つの表は、サービス・リクエストの作成、および適切な技術者への割当てを示します。

表

USERS: この表は、システムと対話するすべてのユーザー（顧客、技術者、マネージャなど）を格納します。各ユーザーの電子メール・アドレス、姓名、番地、都市名、州、郵便番号および国情報が格納されます。ID はユーザーを一意に識別します。

SERVICE_REQUESTS: この表は、特定の製品に対するアクティビティについての内部および外部リクエスト両方を示します。常に、製品の問題の解決に対してリクエストがあります。サービス・リクエストが作成されると、リクエストの日付、それをオープンした人の名前、および関連製品すべてが記録されます。問題の概略も格納されます。リクエストが技術者に割り当てられると、技術者の名前および割当ての日付も記録されます。人為的 ID がすべてのサービス・リクエストを一意に識別します。

SERVICE_HISTORIES: 各サービス・リクエストに対して、多くのイベントが記録されている場合があります。リクエストが作成された日付、それを作成した人の名前、およびイベント固有の注意すべてが記録されます。サービス・リクエストに関連する内部通信も記録されます。サービス・リクエストとその順序番号は、各サービス履歴を一意に識別します。

PRODUCTS: この表は、会社で取り扱っているすべての製品を格納します。各製品について、名前および説明が記録されます。製品のイメージがある場合、それも格納されます。人為的 ID がすべての製品を一意に識別します。

EXPERTISE_AREAS: 技術者をリクエストにより適切に割り当てるため、各技術者の専門分野が定義されています。

順序

USERS_SEQ: 新しいユーザーに ID を指定します。

PRODUCTS_SEQ: 各製品に ID を指定します。

SERVICE_REQUESTS_SEQ: 新しい各サービス・リクエストに ID を指定します。

1. **ADF_tutorial_setup.zip** ファイルを次の場所から取得します。
   http://download.oracle.com/otndocs/products/jdev/1013/ADF_tutorial_setup.zip
2. ファイルを一時ディレクトリ（例：`C:¥temp¥ADFTutorialSetup¥`）に解凍し、Web ページ用の 3 つのイメージを作成するために使用されるファイルを公開します。チュートリアルではこの後、このディレクトリを <tutorial_install> と呼びます。

サービス・リクエスト・スキーマのインストール

SRDEMO ユーザーは、アプリケーションに表示されるデータを所有します。ユーザー・アカウントの作成および適切な権限の割当てには、Oracle SYSTEM ユーザーまたは相当するものへのアクセスが必要です。`createSchema.sql` ファイルには、データベース・ユーザーの作成に必要なすべてのコマンドが含まれています。`createSchemaObjects.sql` ファイルは SRDEMO ユーザーとして接続し、このチュートリアル用のすべての表、制約およびデータベース順序を作成します。また、`populateSchemasTables.sql` ファイルは、チュートリアルで使用するサンプル・データを表に挿入します。

注意： セキュリティ上の理由から、ADF チュートリアル・スキーマを本番データベースにインストールすることはお薦めしません。ユーザーを作成する権限を持つアカウントにアクセスするには、DBA の許可が必要な場合があります。

---

1. SQL*Plus を起動し、SYSTEM または別の DBA レベル・ユーザーとしてログインします。DBA にアカウントを要求するか、スクリプトを実行する必要がある場合があります。

2. SQL*Plus ウィンドウで、build.sql スクリプトを解凍したディレクトリから起動します。次に例を示します。

```
SQLPLUS>Start <tutorial_install>¥scripts¥build.sql
```

build.sql スクリプトに制御が戻ると、作成されたオブジェクトのリストが無効となる可能性のあるオブジェクトとともに表示されます。これらのスクリプトの実行は、30 秒未満で終わります。build.sql スクリプトを再実行して SRDEMO 所有者およびオブジェクトを削除し、再作成する場合があります。

Oracle JDeveloper の起動

JDeveloper Studio を準備するには、次の手順を実行します。

> **注意：** JDeveloper 10*g* リリース 3 をまだインストールしていない場合、次のステップに進む前にインストールをしてください。

1. Windows Explorer で、JDeveloper がインストールされているディレクトリに移動します。ルート・ディレクトリで、**JDeveloper.exe** アイコンをダブルクリックし、JDeveloper を起動します。JDeveloper を初めて実行する場合、ユーザー設定の移行ウィンドウが表示されます。
2. **「いいえ」**をクリックして続行します。アプリケーション全体を最初から作成します。
3. 起動時、今日のヒント・ウィンドウが表示されます。これらのヒントは、開発をより生産的にするためにできることです。ヒントを見終わったら、**「閉じる」**をクリックします。

SRDEMO スキーマにアクセスするためのデータベース接続の作成

SRDEMO ユーザーを使用してサービス・リクエスト・スキーマへの新しいデータベース接続を作成するには、次の手順を実行します。

> **注意：** このチュートリアルでは、データベース接続は **SRDemo** という名前です。接続名はチュートリアルを完了するための機能には影響ありません。ただし、すべてのステップで記述されているネーミング規則を使用することをお薦めします。そうすることで、指示が理解しやすくなります。

1. JDeveloper で、**「表示」**→**「接続ナビゲータ」**を選択します。
2. **「データベース」**ノードを右クリックし、**「データベース接続の作成」**を選択します。
3. 「ようこそ」ページで**「次へ」**をクリックします。
4. 「接続名」フィールドで、接続名として **SRDemo** と入力します。**「次へ」**をクリックします。
5. 「認証」ページで、次の表に示されている値を入力します。**「次へ」**をクリックします。

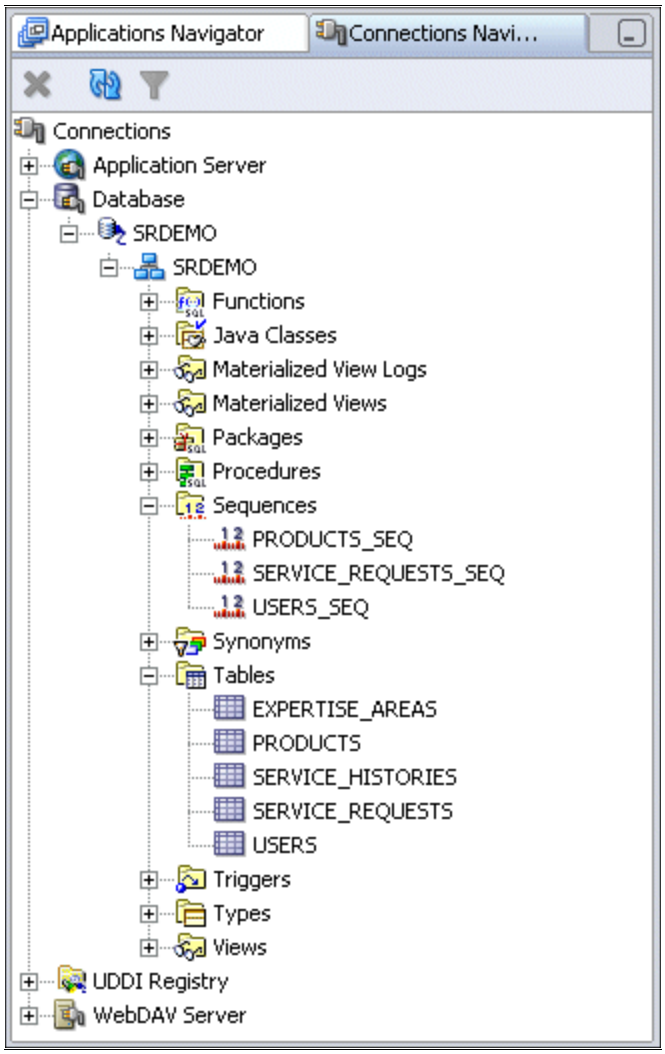
フィールド	値
ユーザー名	SRDEMO
パスワード	Oracle
パスワードを配布	チェック・ボックスを選択

6. 「接続」ページで、次の値を入力します。**「次へ」**をクリックします。

フィールド	値
ホスト名	`localhost` これは、データベースが JDeveloper と同じマシン上にある場合のデフォルトのホスト名です。データベースが別のマシンにある場合、データベースがあるコンピュータの名前（または IP アドレス）を入力します。
JDBC ポート	`1521` これは、データベースへのアクセスに使用するポートのデフォルト値です。この値がわからない場合、データベース管理者に問い合せてください。
SID	`ORCL` これは、データベースへの接続に使用する SID のデフォルト値です。この値がわからない場合、データベース管理者に問い合せてください。

7. **「接続のテスト」**をクリックします。データベースが使用可能で、接続の詳細が正しければ、続行します。そうでない場合、**「戻る」**ボタンをクリックし、値を確認します。
8. **「終了」**をクリックします。これで、接続が接続ナビゲータの「データベース接続」ノードの下に表示されます。

9. これでスキーマを JDeveloper で調べることができます。接続ナビゲータで、**「データベース」→「SRDemo」**を開きます。スキーマのデータベース要素を参照し、前述のスキーマ定義と一致しているかを確認します。



JDeveloper でのアプリケーションとそのプロジェクトの定義

JDeveloper では、アプリケーションに含まれるプロジェクト内を処理します。

アプリケーションは、制御構造内の最上位レベルで、アプリケーションのすべてのサブパートのコントローラとして機能します。JDeveloper を開くと、最後に JDeveloper を閉じたときに開いていたアプリケーションが、デフォルトで開きます。

JDeveloper プロジェクトは、関連ファイルを論理的にグループ化するために使用される組織構造です。J2EE アプリケーションでは、プロジェクトは通常、データ・モデルやクライアントの一部などの、アプリケーション構造の一部を表します。

複数のプロジェクトをアプリケーションに追加し、ソース・コードを簡単に編成、アクセス、変更および再利用できます。

アプリケーション・コンポーネントを作成する前に、アプリケーションとそのプロジェクトを最初に作成する必要があります。新しい SRDemo アプリケーションとそのプロジェクトを作成するには、次の手順を実行します。

注意： プロジェクト名、アクティビティ名または要素名に特殊文字（ピリオドなど）を使用しないでください。特殊文字を使用した場合、プロジェクトをコンパイルしようとするとエラーが発生します。

---

1. アプリケーションを作成するには、**アプリケーション・ナビゲータ**内をクリックし、**「アプリケーション」**ノードを右クリックして、ショートカット・メニューから**「新規アプリケーション」**を選択します。

フィールド	値
アプリケーション名	`SRDemo`
ディレクトリ名	`<jdev_install>¥jdev¥mywork¥SRDemo` デフォルト値のままにします。デフォルトのディレクトリ構造を使用した場合、パスはこの値と一致するはずです。ディレクトリは指定したパスに作成され、アプリケーションを示します。
アプリケーション・パッケージの接頭辞	`oracle.srdemo` この値は、Java パッケージ名すべての接頭辞になります。必要に応じて後でオーバーライドできます。
アプリケーション・テンプレート	`No Template [All Technologies]` このチュートリアルでは、すべての JDeveloper のテクノロジにアクセスします。開発中に使用可能なテクノロジを制限するため、新しいテンプレートを作成および追加できます。

2. アプリケーションの作成ウィンドウで、次の値を入力します。
3. **「OK」**をクリックします。
4. 「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。次のように値を設定します。

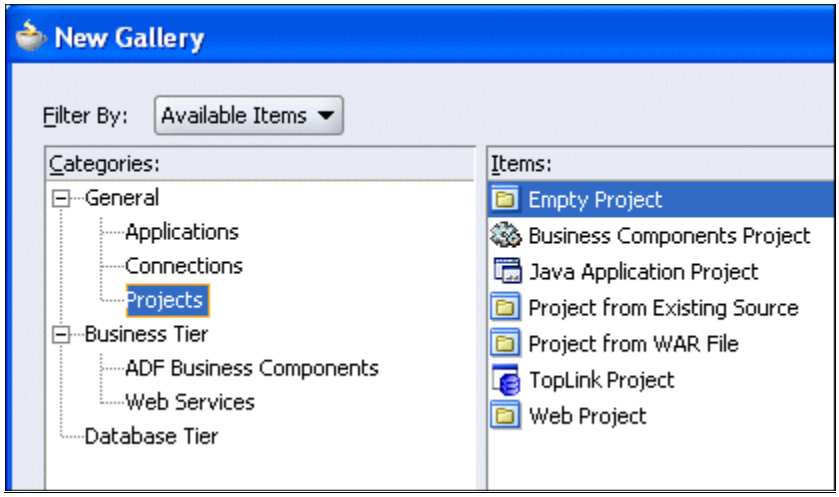
フィールド	値
プロジェクト名	`DataModel`
ディレクトリ名	`<jdev_install>¥jdev¥mywork¥SRDemo¥DataModel` デフォルト値のままにします。推奨ディレクトリ構造を使用した場合、パスはこの値と一致するはずです。プロジェクトのファイルを含むよう、指定したパスにディレクトリは作成されます。

5. 「**OK**」をクリックします。
DataModel プロジェクトは、アプリケーションのデータ・モデル層を示します。次の章で、Oracle TopLink を使用してこのプロジェクトのコンポーネントを作成します。

6. DataModel プロジェクトと同じレベルに別のプロジェクトを作成します。アプリケーション・ナビゲータで、「**SRDemo**」ノードを右クリックし、「**新規プロジェクト**」を選択します。

7. 新規ギャラリが起動します。「**空のプロジェクト**」が選択されていることを確認します。「**OK**」をクリックします。

   新しいプロジェクトは、選択したアプリケーション・ノードの子になります。プロジェクトのタイプについて詳細を調べる場合、および使用する必要がある場合、プロジェクト・カテゴリについて、「ヘルプ」を参照します。



8. 「プロジェクトの作成」ペインで、次のように値を設定します。

フィールド	値
プロジェクト名	`UserInterface`
ディレクトリ名	`<jdev_install>¥jdev¥mywork¥SRDemo¥UserInterface` デフォルト値のままにします。推奨ディレクトリ構造を使用した場合、パスはこの文字列と一致するはずです。

9. 「**OK**」をクリックします。UserInterface プロジェクトは、アプリケーションの残りの部分を示し、後述の章でユーザー・インタフェース用に作成するファイルを含みます。

10. 新しい **UserInterface** プロジェクトをダブルクリックし、「**依存性**」ノードを選択します。

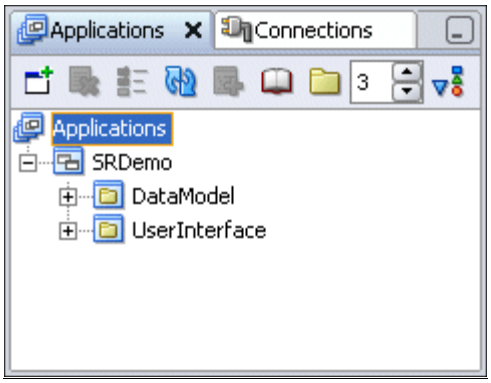
11. **DataModel** プロジェクトに関連付けられたチェック・ボックスを選択します。これにより、UserInterface プロジェクトが DataModel プロジェクトで作成されたオブジェクトにアクセスできます。

12. 「**プロジェクト・コンテンツ**」ノードを選択します。パネルの下部で、デフォルト・パッケージを **`oracle.srdemo.userinterface`** に設定します。「**OK**」を押します。これにより、クラスおよびファイルをより効率的に管理できるようになります。

13. **DataModel** プロジェクトをダブルクリックし、「**プロジェクト・コンテンツ**」ノードを選択します。

14. パネルの下部で、デフォルト・パッケージを **oracle.srdemo.datamodel** に設定します。「**OK**」を押します。これにより、クラスおよびファイルをより効率的に管理できるようになります。

    アプリケーション・ナビゲータは、次のスクリーンショットのように表示されます。これで、チュートリアル用のアプリケーション・コンポーネントを作成する準備ができました。



まとめ

この章では、アプリケーションの作成を始める前に行っておく必要がある前提条件となるすべての手順を実行しました。次の主な作業を実行しました。

- チュートリアルのスキーマ設定の準備
- サービス・リクエスト・スキーマのインストール
- JDeveloper の起動
- JDeveloper データベース接続の作成
- JDeveloper でのアプリケーションとそのプロジェクトの定義

2

データ・モデルの開発

この章では、Oracle TopLink および Oracle ADF を使用して、アプリケーションのデータ・モデルを作成する方法について説明します。

この章の内容は次のとおりです。

- 概要
- Oracle TopLink を使用したデータ・モデルの作成
- TopLink 定義の調整
- TopLink 名前付き問合せの作成
- TopLink セッションの作成
- EJB セッション Bean の作成
- ADF データ・コントロールの作成
- まとめ

概要

TopLink は、Java のオブジェクトとリレーショナル間の永続性を提供し、これによりリレーショナル・データのアクセスおよび永続化のために Java オブジェクトを作成できます。Oracle ADF を使用すると、ADF データ・コントロールを介してユーザー・インタフェースでこれらの TopLink オブジェクトを使用できます。これらのデータ・コントロールにより、基礎となるテクノロジの選択（この場合は Oracle TopLink）を考慮せずに、クライアント・アプリケーションでデータを使用できます。

この章では、次の主なタスクを実行します。

- データベース・オブジェクト用の TopLink POJO（Java オブジェクト）の作成
- TopLink セッションの作成
- TopLink 名前付き問合せの作成
- EJB セッション Bean の作成
- EJB セッション Bean POJO 用の ADF データ・コントロールの作成

Oracle TopLink を使用したデータ・モデルの作成

アプリケーションで実行されるすべてのデータ・アクセスは、データ・モデルを介して行われます。この項では、データベースの表を TopLink のクラスにマップする方法を示します。

データベース・オブジェクトに対する TopLink マッピングの作成

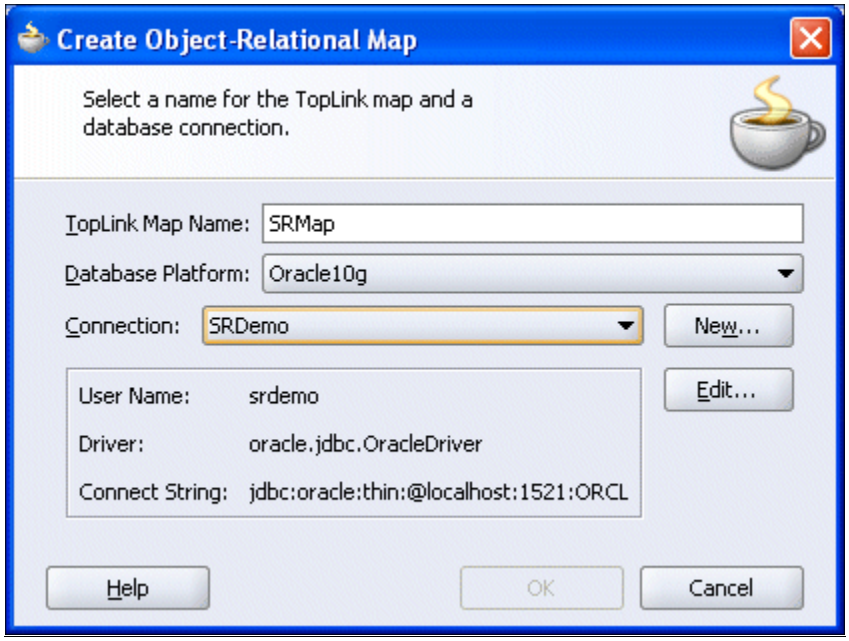
> **注意：** 基礎となるデータベース・スキーマは、TopLink マッピングが作成された後、変わる場合があります。そのような場合、オブジェクトとマッピングを削除し、再作成できます。ただし、TopLink オブジェクトを作成する前に一致したデータベース・スキーマを持つ方が適切です。

このステップでは、TopLink Java オブジェクトを SRDEMO スキーマの既存のデータベース表からリバース・エンジニアリングします。

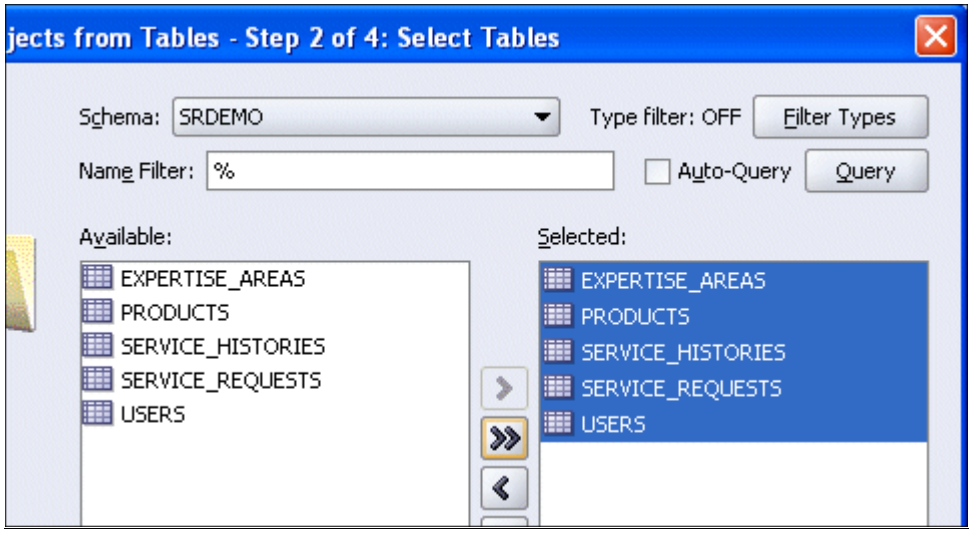
1. アプリケーション・ナビゲータで、**DataModel** プロジェクトを右クリックし、**「新規」**を選択します。
2. 新規ギャラリで、**「Business Tier」**ノードを開き、**「TopLink」**を選択し、「項目」リストから**「表からの Java オブジェクト」**を選択します。
3. **「OK」**をクリックします。

   TopLink では、各プロジェクトに対して TopLink マップ・ファイルが必要です。このファイルには、クラスのデータベース表へのマップ方法の情報が含まれます。TopLink オブジェクトを作成した後、このファイルを編集してデフォルトのマッピングに変更ができます。プロジェクトに対して新しいマップを作成する必要があります。
4. 表からの Java オブジェクトの作成ウィザードのステップ 1 で、TopLink マップ・プロパティに対して**「新規」**ボタンをクリックします。

5. 「オブジェクト・リレーショナル・マップの作成」ダイアログ・ボックスで、「TopLink マップ名」を **SRMap** に設定し、接続が **SRDemo** に設定されていることを確認します。



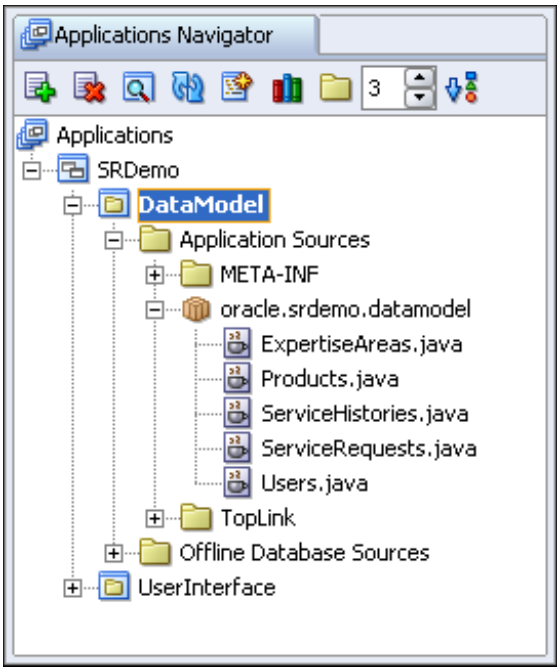
6. 「**OK**」をクリックします。
7. ウィザードのステップ 1 に戻り、「**次へ**」をクリックします。
8. ウィザードのステップ 2 で、TopLink オブジェクトとして示す表を選択します。「**問合せ**」ボタンをクリックして使用可能な表を表示します。「**>>**」ボタンをクリックして使用可能な表すべてを「選択」側に移動します。



9. 「**次へ**」をクリックします。
10. ウィザードのステップ 3 で、パッケージ名が **oracle.srdemo.datamodel** であることを確認します。
    違う場合、変更する必要があります。
11. 「**次へ**」をクリックします。

12. ウィザードのステップ 4 で、各データベース表の完全修飾 Java クラス名を確認します。**「終了」**をクリックしてウィザードを完了します。

13. ツールバーの**「すべて保存」**ボタンをクリックするか、メニューから**「ファイル」→「すべて保存」**を選択して作業を保存します。**「アプリケーション・ソース」**を開きます。ここでは、次のように表示されます。



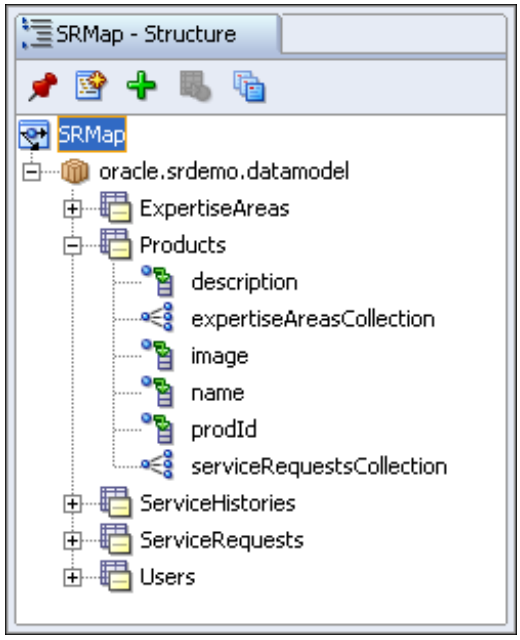
SRDEMO スキーマの 5 つの表それぞれに対して TopLink POJO が作成されました。各 `.java` ファイルには、属性定義、コンストラクタ、ゲッターおよびセッターのコードが含まれます。

14. **`Products.java`** ファイルをダブルクリックし、内容を確認します。表の各列に対して変数が作成されていることに注意します。下へスクロールして、コンストラクタ、ゲッターおよびセッターを表示します。

    TopLink マッピング・ファイルには、オブジェクトと表間のマッピングが含まれます。構造ウィンドウで、各オブジェクトに移動し、それをエディタに表示できます。

15. アプリケーション・ナビゲータで、**「SRMap」**をクリックして選択します。構造ウィンドウで、**oracle.srdemo.datamodel** ノードを開きます。各ノードは、作成されたオブジェクトを表します。

16. 「**Products**」ノードを開き、マップされた属性およびコレクションを表示します。次のスクリーンショットのように表示されます。



複数の属性およびコレクションが示されます。コレクションは、モデル内の選択したオブジェクトと他のオブジェクトの関係を表します。たとえば、`expertiseAreasCollection` コレクションは `ExpertiseAreas` オブジェクトとの親子関係を表します。

TopLink 定義の調整

この項では、作成した TopLink オブジェクトの名前を調整し、主キー値のシーケンス・ジェネレータを追加して、いくつかのデフォルト値を指定します。

場合によっては、TopLink によって割り当てられるデフォルト名よりも意味のあるオブジェクト名を定義する場合があります。オブジェクトの名前は簡単に変更できます。

`ServiceHistories.java` ファイルには `svrId` の整数およびサービス・リクエストの `ValueHolderInterface` 両方が含まれます。`svrId` 整数は、オブジェクト指向以外のアプローチで使用されるので、その整数を削除し、サービス・リクエストのコレクションのみを保持します。`ExpertiseAreas.java` ファイルにも同様の変更をします。

1. アプリケーション・ナビゲータで、「**`ServiceHistories.java`**」ファイルをダブルクリックします。これによって、コード・エディタでファイルが開きます。エディタで、コード **`private Integer svrId`** の行を削除します。変数 `svrId` は、このクラスのセッター・メソッドおよびゲッター・メソッドで使用されます。変数を削除すると、コード・エディタの右余白に赤いバーが表示されます。これは変数は使用されていたが、もう使用できないことを示します。これらのメソッドも削除する必要があります。
2. これらの赤いバーの 2 番目および 3 番目の場所でゲッター（**`getSvrId`**）およびセッター（**`setSvrId`**）を削除します（赤いバーをクリックすると、特定のメソッドに移動します）。
3. **`ExpertiseAreas.java`** ファイルに対して、前述の 2 つのステップを繰り返します。**`private Integer prodId`** および **`private Integer userId`** を削除し、次にそれらのスカラー属性に対するゲッターおよびセッターを削除します。6 つの削除操作があります。2 つは属性に対するもの、2 つはゲッター（`getProdId` および `getUserId`）に対するもの、および 2 つはセッター（`setProdId` および `setUserId`）に対するものです。
4. 作業内容を保存します。

デフォルト値を管理するためコードを追加

属性に対してデフォルト値を指定する必要がある場合、Java クラスでセッター・メソッドに直接そのコードを追加できます。

次の手順で、ServiceRequests の POJO にデフォルト値を含めます。

1. 「**ServiceRequests.java**」をダブルクリックして、コード・エディタでこれを開きます。
2. setReequestDate(Timestamp) メソッドを検索し、次の変更をします。(このコードでは、requestDate がそのデフォルトとして現在の時間と日付を使用するよう設定します)

```
public void setRequestDate(Timestamp requestDate) {
   this.requestDate = (requestDate==null)
?new Timestamp(System.currentTimeMillis()):requestDate;
}
```

3. **setStatus** がそのデフォルトとして **Open** を使用するよう変更します。

```
public void setStatus(String status) {
   this.status = (status==null)?"Open":status;
}
```

作業内容を保存し、追加のビジネス・ロジックを含め、ServiceHistories の POJO のデフォルト値を設定して行数を増やします。ServiceHistories の POJO に対して次の変更を行います。

1. **svhDate** がそのデフォルトとして現在の時間と日付を使用するよう設定します。現在の割当てをコメント・アウトします。

```
public void setSvhDate(Timestamp svhDate) {
   this.svhDate = (svhDate==null)
?new Timestamp(System.currentTimeMillis()):svhDate;
}
```

2. ユーザーがサービス履歴のノートを追加する場合、次の行番号を判断し、自動的に使用します。番号を判断するには、現在の ServiceHistories コレクション内を反復して最大値を検索し、1 増分された値を戻すメソッドを追加します。メソッドは、新しいサービス履歴記録のための次の行番号を計算するために使用されます。既存の ServiceHistories 内を反復し、最大値となる行番号を検索します。次のようにメソッドを追加します。

```
public Integer getNextLineItem() {
       int maxLineNo = 0;
       for (ServiceHistories
            svh:getServiceRequests().getServiceHistoriesCollection()){
            if (svh.getLineNo() !=null) {
               int testLineNo = svh.getLineNo().intValue();
                if (testLineNo > maxLineNo){
                   maxLineNo = testLineNo;
                }
            }
       }
return ++maxLineNo;
}
```

3. getNextLineItem メソッドから setLineNo 値を取得します。

```
public void setLineNo(Integer lineNo) {
   this.lineNo = (lineNo==null)?getNextLineItem():lineNo;
}
```

4. アプリケーション・ナビゲータで、「**DataModel**」プロジェクトを選択します。コンテキスト・メニューから、「**再ビルド**」を選択し、クラスをコンパイルします。問題を修正してから、次に進みます。

Toplink を使用し、データベース順序を宣言的に使用して列に移入します。次の手順で、ServiceRequests 表の主キーを選択し、データベース順序を使用するよう設定します。

1. アプリケーション・ナビゲータで、「**TopLink**」ノードを開きの「**SRMap**」を選択します。構造ウィンドウで、「SRMap」の下の「**oracle.srdemo.datamodel**」ノードを開きます。

   シーケンス・ジェネレータによって、SERVICE_REQUESTS.SVR_ID 列に値が提供され、その列が ServiceRequest マッピングにアタッチされます。

2. 「**SRMap**」をダブルクリックし、順序で**ネイティブ順序付け**が使用されるようラジオ・ボタンを設定します。

3. 「**ServiceRequests**」ノードをダブルクリックしてサービス・リクエストのマッピングの詳細を表示します。ページの中ほどの「順序付けの使用」領域で、次のように値を設定します。

フィールド	値
順序付けの使用	チェック・ボックスを選択
名前	SERVICE_REQUESTS_SEQ
表	SRDEMO.SERVICE_REQUESTS (デフォルトになっています)
フィールド	SRDEMO.SERVICE_REQUESTS.SVR_ID (デフォルトになっています)

   これで、データ・モデルに対するビジネス・ロジックの変更が終了しました。次の項では、ページに対するいくつかの特別なメソッドを宣言的に作成します。

4. 作業内容を保存します。

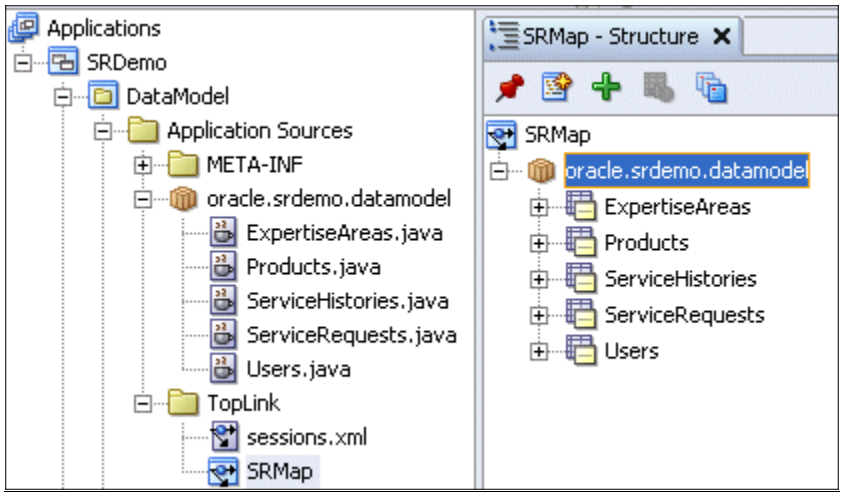
TopLink 名前付き問合せの作成

名前付き問合せでは、データベースにアクセスする複雑なまたは一般的に使用される問合せの定義方法を提供します。名前付き問合せにはいくつかの主な利点があります。名前付き問合せを使用すると、問合せを集中管理することによって作業が向上し、メンテナンスが簡単になるよう TopLink の内部パフォーマンスを最適化することができます。

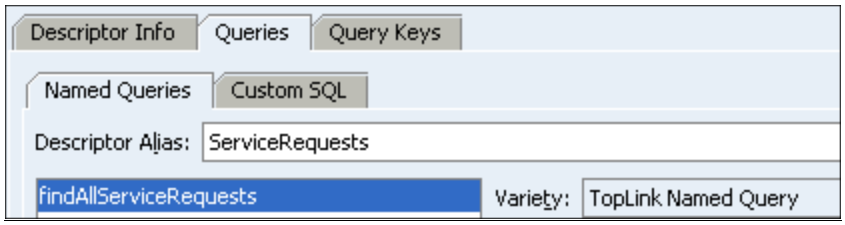
このチュートリアルでは、名前付き問合せを使用して Web クライアントと TopLink の間で値を渡す必要があります。これらの各名前付き問合せには、パラメータと、受信パラメータを問合せに適用する式が含まれます。次の手順では問合せとパラメータ式の両方を定義します。

名前付き問合せの作成

1. アプリケーション・ナビゲータで、「TopLink」ノードの下の**「SRMap」**ノードを選択します。構造ウィンドウで、**「SRMap」**ノードをダブルクリックし、「oracle.srdemo.datamodel」ノードを開きます。各ノードは、作成されたオブジェクトを表します。



2. **「ServiceRequests」**ノードをクリックして、**「問合せ」**タブをクリックします。**「追加」**ボタンをクリックして、新規の名前付き問合せを作成します。



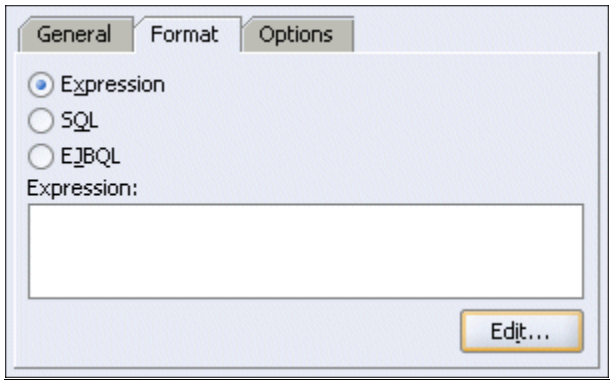
3. 「TopLink 名前付き問合せの追加」ペインで、新規の問合せの名前に、**`findServiceRequests`** と入力します。**「OK」**をクリックして続行します。
4. 名前付き問合せ `findServiceRequests` が選択された状態で、**「一般」**タブをクリックします。エディタの「パラメータ」領域で、**「追加」**ボタンをクリックします。
5. クラス・ブラウザ・ウィンドウで、**「検索」**タブをクリックします。ここでパラメータ・タイプを定義します。
6. ただ **`Integer`** と入力すると、**`Integer (java.lang)`** と表示されます。一致するクラスの `java.lang.Integer` がハイライトされたら、**「OK」**をクリックしてタイプを定義します。

7. 「パラメータ」領域で、パラメータ名を **filedBy** に変更します。
8. java.lang.String の 2 つ目のパラメータを作成し、**status** という名前にします。

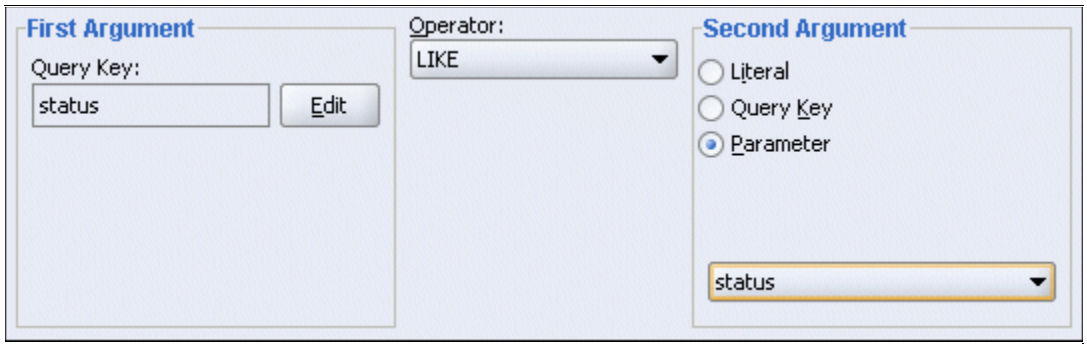
問合せ式の定義

2 つのパラメータ（filedBy および status）を定義したので、パラメータを使用する式を定義する必要があります。この式では、名前付き問合せで、どのようにパラメータが属性に関連付けられるかを定義します。式は実行時に評価され、名前付き問合せによって戻される行を決定します。

1. 名前付き問合せ findServiceRequests が選択された状態で、**「書式」**タブをクリックして、エディタの「パラメータ」領域で**「編集」**ボタンをクリックします。



2. 式ビルダーで、**「追加」**をクリックして、新しい式を作成します。
3. 式の「第 1 引数」領域で、**「編集」**をクリックします。「問合せキーの選択」ペインで、**「status」**を選択します。
4. **「OK」**をクリックして続行します。
5. 式ビルダーに戻り、「演算子」を **Like** に設定し、「第 2 引数」を先に作成した status パラメータに設定します。次のスクリーンショットのように表示されます。



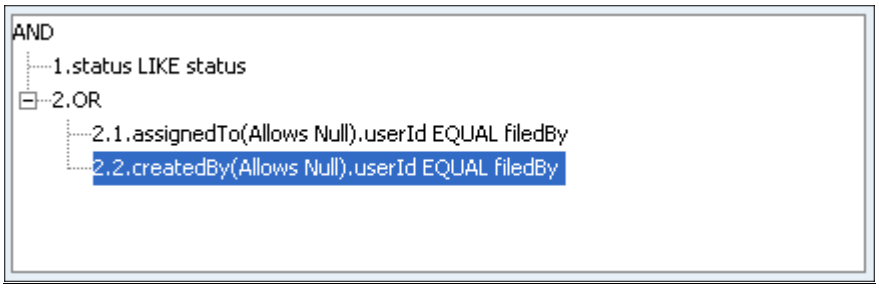
6. 式の最初のコンポーネントが完了したら、それを選択して**「ネストの追加」**ボタンをクリックします。次に**「2.AND」**ノードを選択して「論理演算子」を **OR** に変更します。
7. ネストされた式のコンポーネントを次の表の値に設定します。式に対して**「NULL を許容」**チェック・ボックスを選択します。

フィールド	第 1 引数	演算子	第 2 引数（パラメータ）
第 2 のコンポーネント	AssignedTo(AllowsNull).userId	EQUAL	filedBy

8. 前述のコンポーネントと同じレベルに 2 番目のネストされた式のコンポーネントを作成します。「**追加**」ボタンをクリックして行います（「ネストの追加」ボタンではありません）。値を次の表の値に設定します。式に対して「**NULL を許容**」チェック・ボックスを選択します。

フィールド	第 1 引数	演算子	第 2 引数（パラメータ）
第 3 のコンポーネント	`CreatedBy(AllowsNulls).userId`	`EQUAL`	`filedBy`

完了すると、式は次のように表示されます。結果を確認して、必要に応じて変更を行い、完了したら「**OK**」をクリックします。



追加の名前付き問合せの作成

1. 次の表で指定されている値を使用して ServiceRequests の POJO に別の名前付き問合せを作成します。「**タイプ**」を **ReadObjectQuery** に設定します。

フィールド	値
名前	`findServiceRequestById`
パラメータ型	`java.lang.Integer`
パラメータ名	`findSvrId`

2. 次の表の値を使用して名前付き問合せの式を作成します。

フィールド	値
第 1 引数	`svrId`
演算子	`EQUAL`
第 2 引数	`findSvrId`

3. ここで、その他の POJO にいくつかの名前付き問合せを作成します。次の表で指定されている値を使用して Users に対して名前付き問合せを作成します。「**タイプ**」を **ReadObjectQuery** に設定します。

フィールド	値
名前	`findUserById`
パラメータ型	`java.lang.Integer`
パラメータ名	`createdBy`

4. 次の表の値を使用して名前付き問合せの式を追加します。

フィールド	値
第 1 引数	userId
演算子	EQUAL
第 2 引数	createdBy

5. 最後に、次の表で指定されている値を使用して Products に対して名前付き問合せを作成します。**「タイプ」**を **ReadObjectQuery** に設定します。

フィールド	値
名前	findProductById
クラス名	java.lang.Integer
パラメータ名	prodId

6. 次の表の値を使用して名前付き問合せの式を作成します。

フィールド	値
第 1 引数	prodId
演算子	EQUAL
第 2 引数	prodId

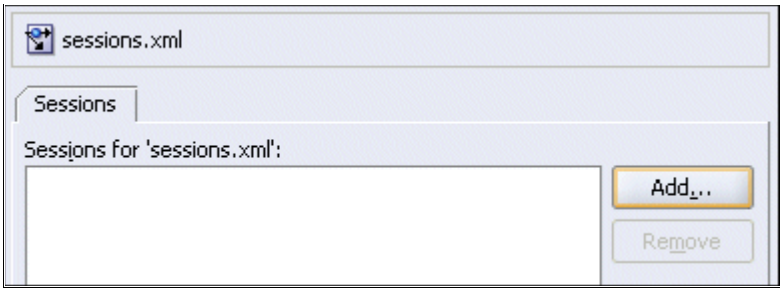
7. 作業内容をすべて保存します。すべてのステップが正しく実行されたかを確認するには、**「SRMap」**を右クリックしてコンテキスト・メニューから**「マッピング・ステータス・レポートの生成」**を選択します。
8. メッセージ領域で、「マッピング・ステータス」の「ログ」では、SRMap にはエラーがないことが示されます。エラーがあった場合、この項の手順中に入力した値を再確認します。

TopLink セッションの作成

ここでは TopLink セッションを作成します。これには、最初にセッション構成ファイルを定義します。TopLink セッション構成は、実行時に各実行セッションの状態情報を保持するために使用されます。また、セッションのデータベース接続情報を提供します。アプリケーションへのアクティブな各接続は、一意のセッション・オブジェクトを受け取ります。

デフォルトでは、セッションは **default** という名前で作成されます。次の手順では、定義する名前で新しいセッションを作成します。

1. アプリケーション・ナビゲータで、**「sessions.xml」**ファイルを選択します。構造ウィンドウで、**「default」**セッションを選択して削除します。
2. 構造ウィンドウで**「sessions.xml」**ファイルをダブルクリックして、エディタで**「追加」**ボタンをクリックします。

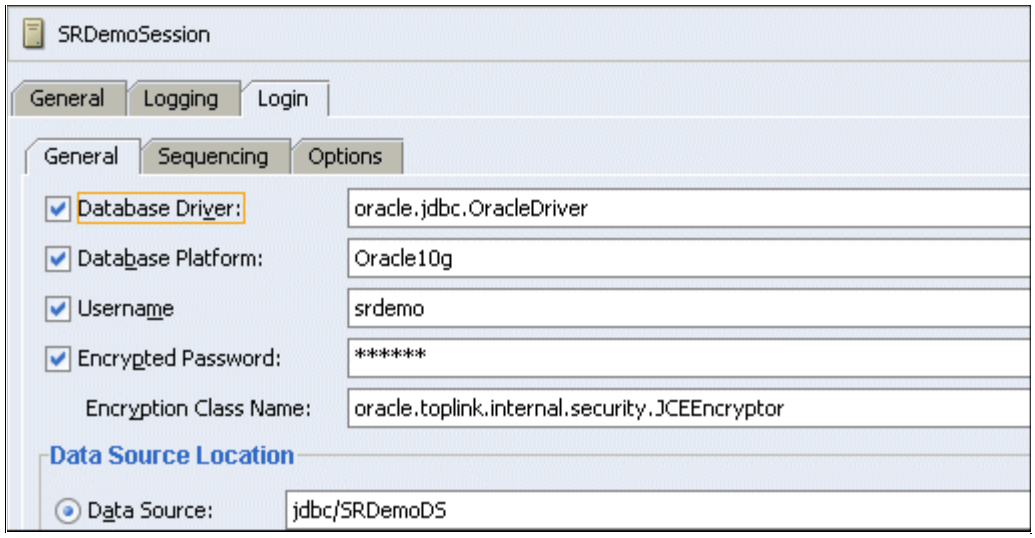


3. 新しいセッションに **SRDemoSession** と名前を付けて、**「OK」**をクリックします。残りのプロパティ設定はデフォルトのままにします。
4. **「すべて保存」**をクリックして作業内容を保存します。

次の手順で、アプリケーションの実行の詳細な結果を表示できるよう、ロギングを有効にします。デバッグが必要な場合、役に立ちます。また、データソースの場所を設定し、直接データベース接続のかわりにデータソース定義を使用します。これによって、このアプリケーション開発のデプロイ・フェーズ中に役立つことがわかります。次に、アプリケーション内のコードまたは設定に影響しないよう接続の詳細を変更できます。

1. 構造ウィンドウで、**「SRDemoSession」**セッションを選択します。エディタには、セッション名と「一般」、「ロギング」および「ログイン」の 3 つのタブが表示されます。
2. **「ロギング」**タブをクリックし、「ロギングの有効化」プロパティを **True** に設定します。

3. 「ログイン」タブで、**「データ ソース」**オプションを選択し、その値を **`jdbc/SRDemoDS`** に設定します。この値は大 / 小文字が区別されるので、接続名で使用したものと大 / 小文字が同じになるようにしてください。



4. **「すべて保存」**をクリックして作業内容を保存します。

接続情報を指定する以外に、セッションに対してロギング機能をオンまたはオフに切り替えることもできます。ロギング機能によって、選択したレベルでセッション情報が記録されます。たとえば、デバッグ情報、アプリケーション例外、または両方を記録するよう選択できます。

次の手順では、`oracle.srdemo.datamodel` パッケージに `SRPublicFacade` という名前のセッション Bean を作成します。

EJB セッション Bean の作成

この項では、EJB セッション Bean を作成してアプリケーションのアクセス・ポイントを提供します。クライアント・アプリケーションでは、このセッション Bean はデータ・モデルへのアクセスに使用されます。このアプローチは、アプリケーションを構成する際のセッション・ファサード・デザイン・パターンに従っています。

セッション Bean によって、必要なインタフェースの公開中にビジネス・ロジックおよびビジネス・データがカプセル化されます。したがって、クライアント層ではモデル内の配布済サービスをその複雑さを考慮せずに使用できます。セッション Bean は次の 2 つのファイルで構成されています。

- **`SRPublicFacadeBean`** には Bean コードが含まれます。
- **`SRPublicFacadeLocal`** はセッション Bean のローカル・インタフェースです。

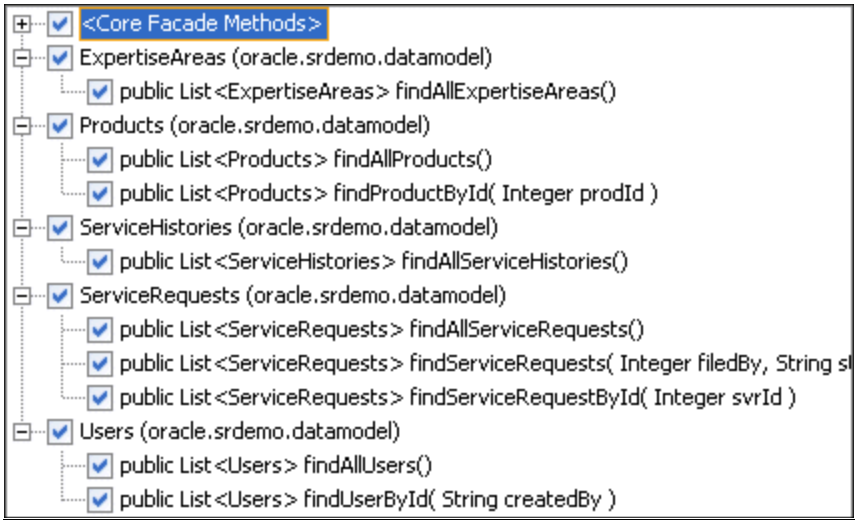
次の手順では、TopLink の POJO を表すセッション Bean とインタフェースを作成します。

1. `oracle.srdemo.datamodel` パッケージに **`SRPublicFacade`** という名前のセッション Bean を作成します。アプリケーション・ナビゲータで、**「DataModel」**プロジェクトを選択しコンテキスト・メニューから**「新規」**を選択します。
2. 新規ギャラリで、**「Business Tier」**を開き、**「EJB」**ノードをクリックします。「項目」ペインで、**「セッション Bean」**を選択して**「OK」**をクリックします。

3. セッション Bean 作成ウィザードのステップ 1、「EJB の名前とオプション」で、次のように値を設定して**「次へ」**をクリックします。

フィールド	値
EJB 名	`SRPublicFacade`
セッション・タイプ	`Stateless`
トランザクション・タイプ	`Container`
セッション・ファサード・メソッドの生成	チェック・ボックスを選択
エンティティの実装	「TopLink POJO」オプションを選択

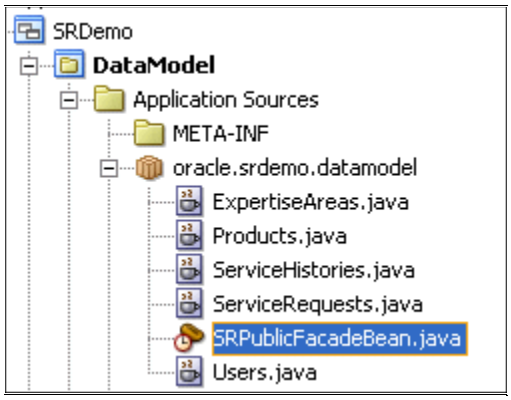
ステップ 2、「セッション・ファサード - サービス・メソッドの選択」で、セッション・ファサード・パターンにすべてのメソッドと名前付き問合せを組み込みます。**「`ServiceRequests`」**ノードを開き、宣言的に作成した名前付き問合せを公開します。すべてのチェック・ボックスが選択されていることを確認し、**「次へ」**をクリックします。



4. ステップ 3、「クラス定義」で、Bean クラスが **`oracle.srdemo.datamodel.SRPublicFacadeBean`** に設定されていることを確認します。デフォルトのディレクトリを受け入れることができます。「次へ」をクリックします。

5. 最後のウィザード・ステップで、**「リモート・インタフェースの実装」**チェック・ボックスの選択を解除し、**「ローカル・インタフェースの実装」**チェック・ボックスは選択したままにします。ここで作業している場合同様、Web クライアントで作業をしている場合はこの選択が必要です。

6. **「終了」**をクリックしてプロセスを完了します。これで、アプリケーション・ナビゲータには（次のスクリーンショットで示すように）新しい SRPublicFacadeBean クラスが表示されます。



セッション Bean のカスタマイズ

作成するアプリケーションは、サービス・リクエストの作成および保持を主に行います。新しい行を作成する最良の方法は、パラメータを受け入れ、そのパラメータに基づいて新しい行を作成するメソッドをセッション Bean に追加することです。これはセッション Bean 内のメソッドなので、単純であろうと複雑であろうと、どんなコードが必要な場合でも追加できます。なお、サービス・リクエスト作成メソッドのコードは、このチュートリアルの設定ファイルで提供されています。

次の手順で、このコードをセッション Bean にコピーします。

1. **「SRPublicFacadeBean」**を選択し、構造ウィンドウで**「Sources」**ノードを開きます。**「SRPublicFacadeLocal」**ノードをダブルクリックして、エディタでファイルを開きます。
2. 入力する必要があるコードは、`<tutorial_install>¥files¥` ディレクトリのファイルにあります。Windows Explorer で、**`CreateInterface.txt`** ファイルを検索し、開きます。コードをコピーし、`SRPublicFacadeLocal` ファイルの下部（閉じの中カッコの前）に貼り付けます。
3. 詳細なメソッドを SRPublicFacadeBean に追加します。**「SRPublicFacadeBean」**をダブルクリックして、エディタでこれを起動します。`<tutorial_install>¥files¥` ディレクトリの **`CreateSession.txt`** ファイルを開きます。すべてのコードをコピーし、`SRPublicFacadeBean.java` に貼り付けます。
4. 作業内容を保存します。

ADF データ・コントロールの作成

TopLink の POJO および SRPublicFacadeBean を使用するには、ADF データ・コントロールを作成する必要があります。

データ・コントロールによって、データ・モデルをユーザー・クライアント・アプリケーションにバインドするインタフェースが提供されます。つまり、後でクライアント・アプリケーションを開発する際に、データ・モデルの作成に使用された実際のテクノロジ（この場合、TopLink および EJB セッション Bean）を考慮することなくデータ・コントロールをページ・コンポーネントの基礎とすることができることを意味します。

ここで、次のように TopLink マップ（SRMap）で定義されているオブジェクトへのアクセスを提供するデータ・コントロールを作成します。

アプリケーション・ナビゲータで、**「SRPublicFacadeBean.java」**ノードを右クリックして、コンテキスト・メニューから**「データ・コントロールの作成」**を選択します。

データ・コントロールの詳細は、いくつかのファイルに保持されています。これには、各オブジェクトに対して組込み操作が可能かを判断する XML ファイルが含まれます。ここでは、データ・コントロール用に作成されるファイルの概略を示します。

ファイル名（または例）	ファイル・タイプ
Users.xml、ServiceRequests.xml など	**個々の POJO（それぞれのうちの 1 つ）を表し、その属性、アクセッサおよびパラメータについて説明します。**
UpdateableSingleValue.xml	**1 つの値を持つオブジェクトに対する操作を制御します。**
UpdateableCollection.xml	**複数の値を持つオブジェクトに対する操作を制御します。**
DataControls.dcx（oracle.srdemo パッケージ内）	**データ・コントロールに対して使用されるインタフェースを定義します。**

データ・コントロールのランタイム動作を変更するプロパティを Bean に追加することもできます。最も一般的なプロパティの一部は、次のとおりです。

- **コントロール・ヒント :** ラベル・テキスト、ツールヒント・テキストおよび書式設定規則を提供します。
- **属性規則 :** 主キー定義および問合せ規則を有効にします。
- **宣言検証規則**（値の比較、リスト検証、範囲検証、式検証など）

Bean プロパティ・エディタによって、Bean クラス内の Java コードを必要とせずに Bean の動作をカスタマイズするツールが提供されます。このエディタを使用して行った変更は、各 POJO の XML ファイル（Users.xml など）に格納されます。

Bean プロパティ・エディタには、構造ウィンドウからアクセスできます。

1. アプリケーション・ナビゲータで、**「Users.xml」**ファイルをクリックします。
2. 構造ウィンドウで**「Users」**（最上位レベル・オブジェクト）を右クリックし、コンテキスト・メニューから**「プロパティ」**を選択し、Bean プロパティ・エディタを開きます。これで、特定の JavaBean に対する宣言規則を定義する複数のオプションを参照できます。

アプリケーションの作成

この章の最後に、アプリケーションをコンパイルして作業した内容すべてを確認します。

1. **「SRDemo」**を右クリックし、コンテキスト・メニューから**「再ビルド」**を選択します。
2. ログ・ウィンドウをチェックして、コンパイル・エラー（またはその他のエラー）がないことを確認します。

注意： チュートリアルの設定ファイルの一部として、ステップを正常に完了した各章用の ZIP されたアプリケーションが含まれています。

アプリケーションが正常にコンパイルされなかったり、その他のエラーがあった場合、このアプリケーションを次の章の開始時のアプリケーションとして使用できます。各章の最初で、これらの開始用のアプリケーションの使用方法について説明しています。

まとめ

この章では、アプリケーション用のデータ・モデルを作成しました。これを実現するために、次の主なタスクを実行しました。

- データベース・オブジェクト用の TopLink POJO（Java オブジェクト）の作成
- TopLink 定義の調整
- TopLink セッションの作成
- EJB セッション Bean の作成
- TopLink マップで定義する各オブジェクト用の ADF データ・コントロールの生成
- データ・モデルに対する TopLink 名前付き問合せの作成

3

ページ・フローおよびナビゲーションの定義

この章では、JSF アプリケーション内の各ページのアウトライン定義の作成方法、およびページ間のナビゲーションの指定方法について説明します。これには、JDeveloper のダイアグラマを使用します。

この章の内容は次のとおりです。

- 概要
- JSF ナビゲーション・モデルの作成
- ダイアグラムでのページの作成
- ページのリンク
- まとめ

概要

前述の章では、サービス・リクエスト・アプリケーション用のデータ・モデルを作成しました。その章に記載されているように、この章を開始する前にデータ・モデルが正常に作成されていることを確認してください。

この章では、ユーザー・インタフェースでの作業を開始します。JDeveloper の JSF ナビゲーション・モデラーを使用し、アプリケーションのページとそのページ間のナビゲーションをダイアグラムを使用して計画および作成します。次の主な作業を実行します。

- ページ・フロー・ダイアグラムの作成
- ダイアグラムでのページ・プレースホルダの作成
- ページ間のナビゲーション・ルールおよびナビゲーション・ケースの定義

注意： 第 2 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **`Chapter3`** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、`<jdev_install>¥jdev¥mywork¥Chapter3` になります。
2. **`<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter2.zip`** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「`<jdev_install>¥jdev¥mywork¥Chapter3¥SRDemo¥SRDemo.jws`」**を選択します。これによって、この章の初期アプリケーションが開きます。

第 2 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

JSF ナビゲーション・モデルの作成

JSF ナビゲーション・モデラーを使用すると、全体を見ながら視覚的にアプリケーションを設計できます。次の手順では、空の新規ダイアグラムを開き、それにタイトルを追加します。

1. アプリケーション・ナビゲータで、「UserInterface」ノードを右クリックし、ショート・カット・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」ペインで、**「Web Tier」**ノードを開き、**「JSF」**を選択します。
3. 「項目」ペインから**「JSF ページ・フローと構成 (faces-config.xml)」**を選択します。**「OK」**をクリックします。

   この選択により、JSF 構成ファイルがプロジェクトに追加されます。JSF ナビゲーション・モデラーを使用して JSF 構成ファイルを編集し、アプリケーションのページ間のナビゲーション・ルールを指定します。
4. JSF 構成ファイルの作成ダイアログ・ボックスで、デフォルト名の **`faces-config.xml`** を受け入れ、デフォルト・ディレクトリの場所に作成します。**「OK」**をクリックして続行します。

   空のダイアグラムが開きます。コンポーネント・パレットがダイアグラム・エディタの右にあります。これを使用して JSF ナビゲーション・モデルのコンポーネントを作成します。

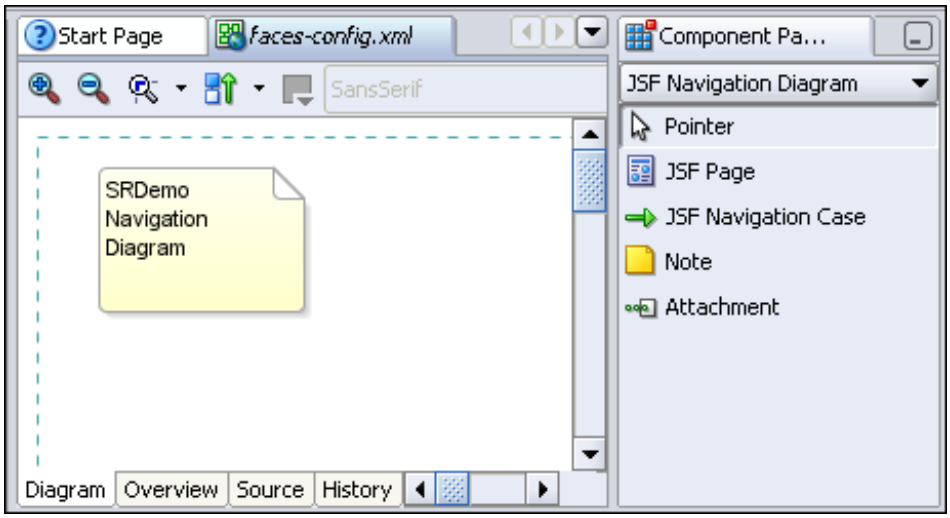
   ダイアグラム・エディタ画面の下部には 4 つのタブがあります。デフォルトのビューは「ダイアグラム」ビューで、ここではアプリケーションのページをモデル化および作成できます。「概要」タブをクリックすると、コンソールタイプのインタフェースが表示され、あらゆるタイプの構成（マネージド Bean、ナビゲーション・ルール、およびカスタム・バリデータ、コンバータなどのその他の項目を含む）を `faces-config` ファイルに登録できます。「ソース」タブを使用すると、生成された XML コードを直接編集できます。

JDeveloper では、JSF ナビゲーションの異なるビューが自動的に同期化されます。「履歴」タブでは、最近の変更履歴が表示されます。

5. タイトルをダイアグラムに追加します。JSF ナビゲーション・ダイアグラムの選択候補がリストされているコンポーネント・パレットから **Note** を選択します。次にそれをダイアグラムの左上にドラッグします。

   これはダイアグラムに注釈を追加するよい方法で、一般的な Note コンポーネントはこのために役立ちます。

6. ダイアグラムのボックス内に `SRDemo Navigation Diagram` と入力し、ダイアグラム内をクリックして、ノートを作成します。ノートはダイアグラムのタイトルになります。



注意： この項の手順では、新しいページ・フロー・ダイアグラムを明示的に作成する方法を示しました。Web アプリケーションを作成し始めたばかりで、まだページを作成していないので、この作業が必要です。`faces-config.xml` ファイルを明示的に作成しない場合、プロジェクトに JSF ページを追加した際に JDeveloper によって自動的に作成されます。これによって、ナビゲータでそれをダブルクリックするだけで、ダイアグラム・エディタで開きます。

ダイアグラムでのページの作成

ページ・フロー・ダイアグラムにページを追加する方法は 2 つあります。JSF ページをいくつかすでに作成している場合は、それをナビゲータからダイアグラムにドラッグできます。あるいは、ダイアグラムで直接ページを作成できます。

SRDemo アプリケーション用のページはまだ作成していません。次の手順で、アプリケーション内にページのプレースホルダを作成します。（このチュートリアルの後の章で、ページを完全に定義します。）

1. コンポーネント・パレットの **JSF Page** を選択し、ダイアグラム内でページを表示する場所をクリックします。

   ページのアイコンがダイアグラムに表示されます。この段階では、警告を表す黄色いアイコンが表示されており、完全に定義されたページではなく単なるプレースホルダであることを示します。

2. アイコン・ラベルをクリックし、ページ名として **`/app/SRList.jspx`** と入力します。

   SRList ページはアプリケーションの中心です。このページで、すべてのユーザー（顧客、技術者およびマネージャ）が既存のサービス・リクエストを参照できます。

   実行できるよう、ページ名の最初にはスラッシュが必要です。名前を入力するときにスラッシュを忘れた場合、スラッシュが追加されます。詳細なページを繰返し作成することも（ページ・フロー・ダイアグラムの開発同様）、後で作成することもできます。

   このチュートリアルでは、アプリケーションのすべてのページのプレースホルダをこの段階で作成し、後の章で詳細なページを作成します。

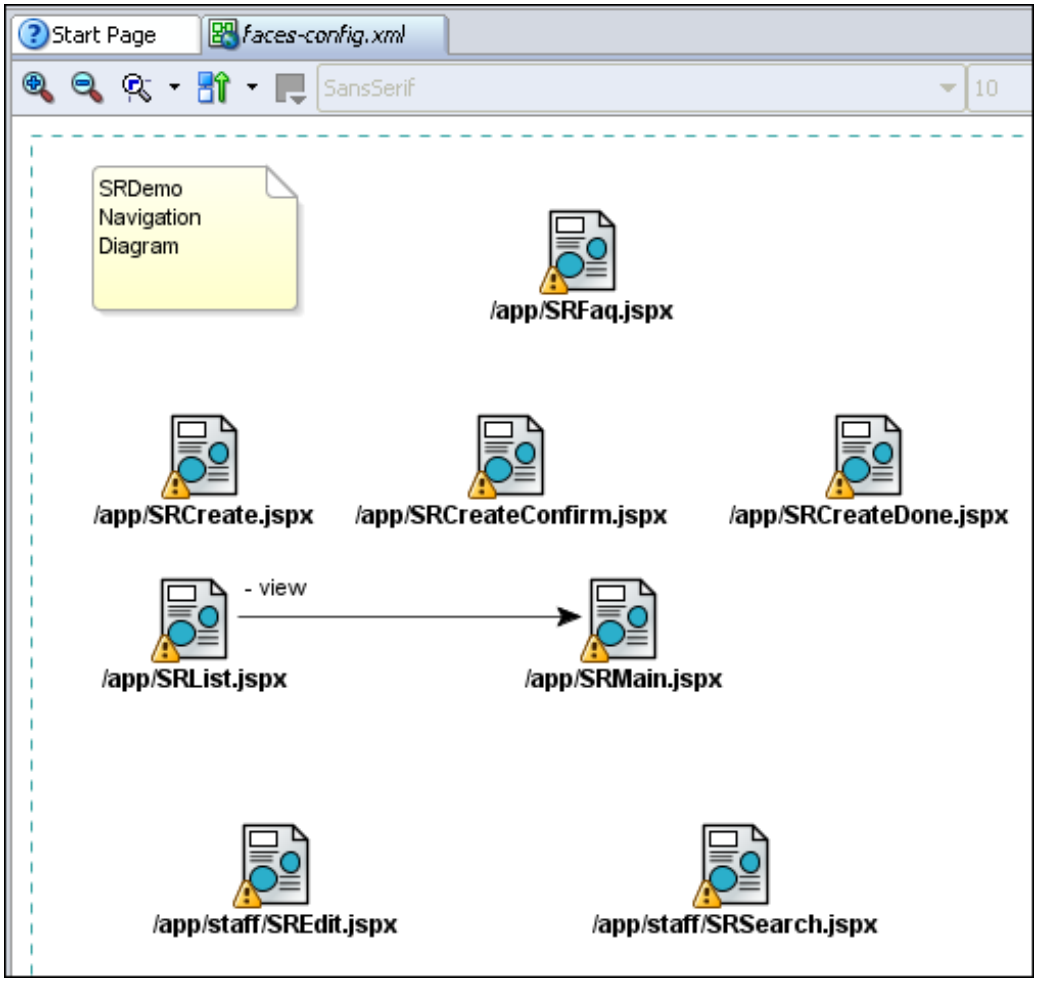
3. 手順 1 と 2 を繰り返し、/app ディレクトリにあと 5 つのページ・プレースホルダを作成します。

ページ名	ページの目的
`/app/SRMain.jspx`	ユーザーが既存のサービス・リクエストに追加情報を追加できるようにします。
`/app/SRCreate.jspx`	ユーザーが新しいサービス・リクエストを作成できるようにします。
`/app/SRCreateConfirm.jspx`	新しいリクエストの確認画面
`/app/SRCreateDone.jspx`	新しいリクエスト作成の最後の画面
`/app/SRFaq.jspx`	ユーザーが FAQ を表示できるようにします。

4. 次の 2 つのスタッフ固有のページを `/app/staff` ディレクトリに作成します。

ページ名	ページの目的
`/app/staff/SREdit.jspx`	マネージャおよび技術者がサービス・リクエストを修正できるようにします。
`/app/staff/SRSearch.jspx`	ユーザーがサービス・リクエストを検索できるようにします。

これらの画面で SRDemo アプリケーションは構成されます。終了すると、ダイアグラムは次のスクリーンショットのように表示されます。違う場合、ダイアグラムが同じになるようページをドラックできます。



ページのリンク

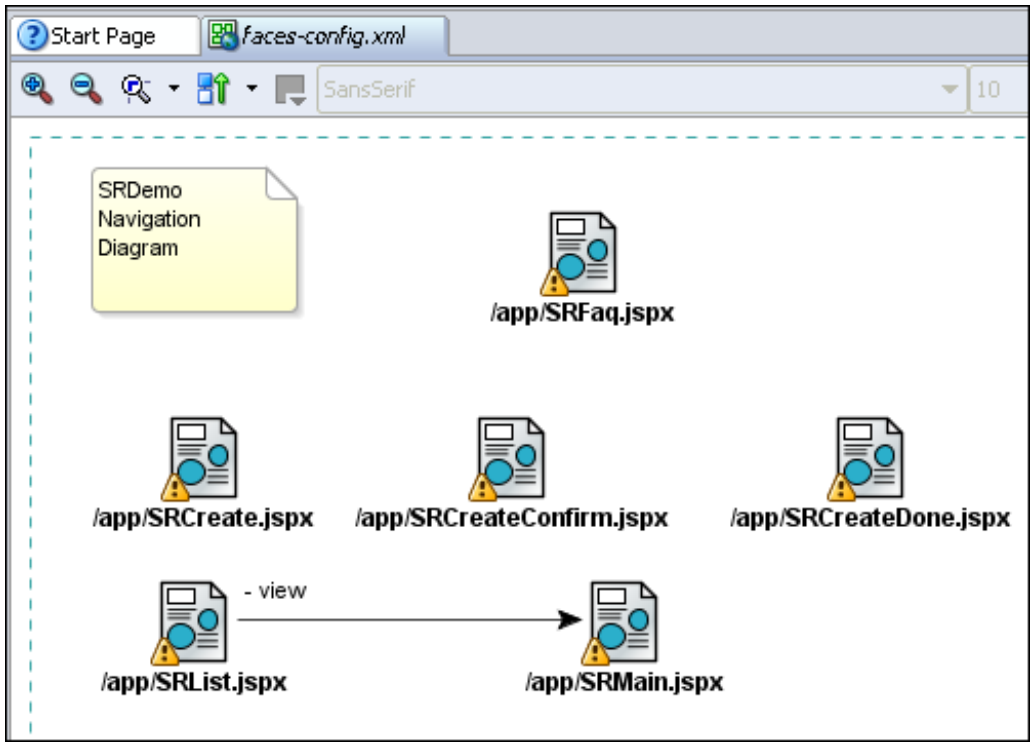
アプリケーションのページのプレースホルダを作成したので、ユーザーのページ間の移動方法を定義する必要があります。JSF ナビゲーションは、ユーザーが UI コンポーネント（たとえば、コマンド・ボタン）をクリックしたときに表示される次のページを選択する一連のルールによって定義されます。これらのルールは、JSF 構成ファイルで定義されます。この構成ファイルはダイアグラムの作成時に作成されます。ユーザーがページから移動する方法はいくつかあり、各方法はそれぞれのナビゲーション・ケースによって表されます。

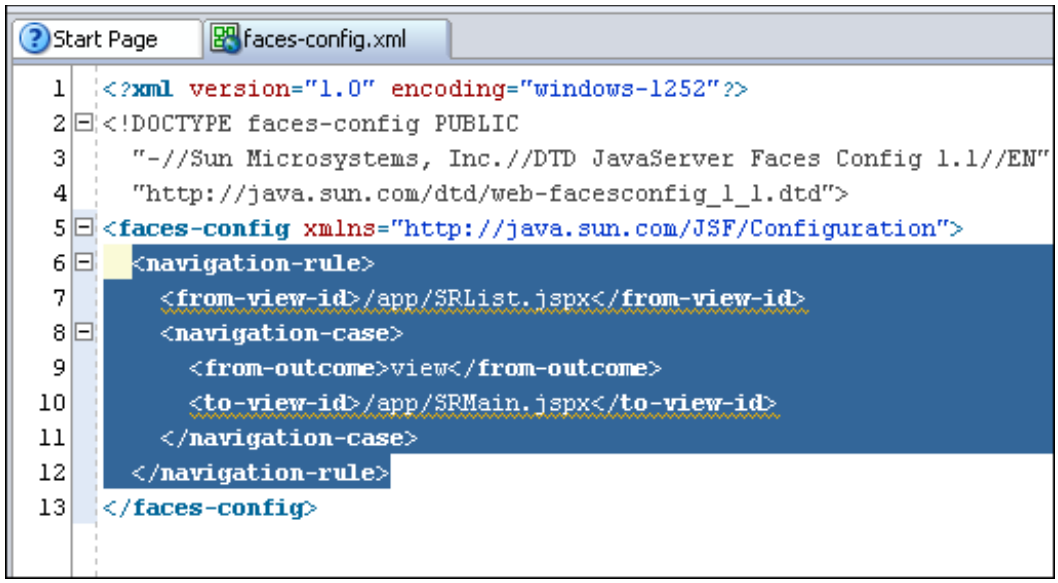
SRDemo アプリケーションの場合、ページ・フロー・ダイアグラムに単純なナビゲーション・ケースを描くことから始めます。

リンクを追加すると、ユーザーは SRList ページから SRMain ページに移動できます。

1. コンポーネント・パレットで、**JSF Navigation Case** を選択します。
2. ナビゲーション・ケース用に、ソース JSF ページ（**SRList**）のアイコンをクリックし、次にリンク先 JSF ページ（**SRMain**）のアイコンをクリックします。

   ナビゲーション・ケースはダイアグラム上に実線で示され、デフォルトのラベル（success）がナビゲーション・ケースの名前として示されます。
3. デフォルトのラベルをクリックして **view** と上書きして変更します。SRList ページには「View」ボタンがあります。このボタンをクリックして SRMain ページに移動します。
4. 画面下部の**「概要」**タブをクリックします。左側の表の**「ナビゲーション・ルール」**をクリックします。ダイアグラムにルールを作成すると、この表にリストされます。
5. 作成したナビゲーション・ルールの構文を理解するには、**「ソース」**タブをクリックしてルールの XML コードを表示します。次の画像では、ダイアグラム上およびソース内のルールを示します。





<from-view-id> タグはソース・ページを識別し、<to-view-id> タグはリンク先ページを識別します。ページ名の下の波線は、ページがまだ作成されていないことを示します。

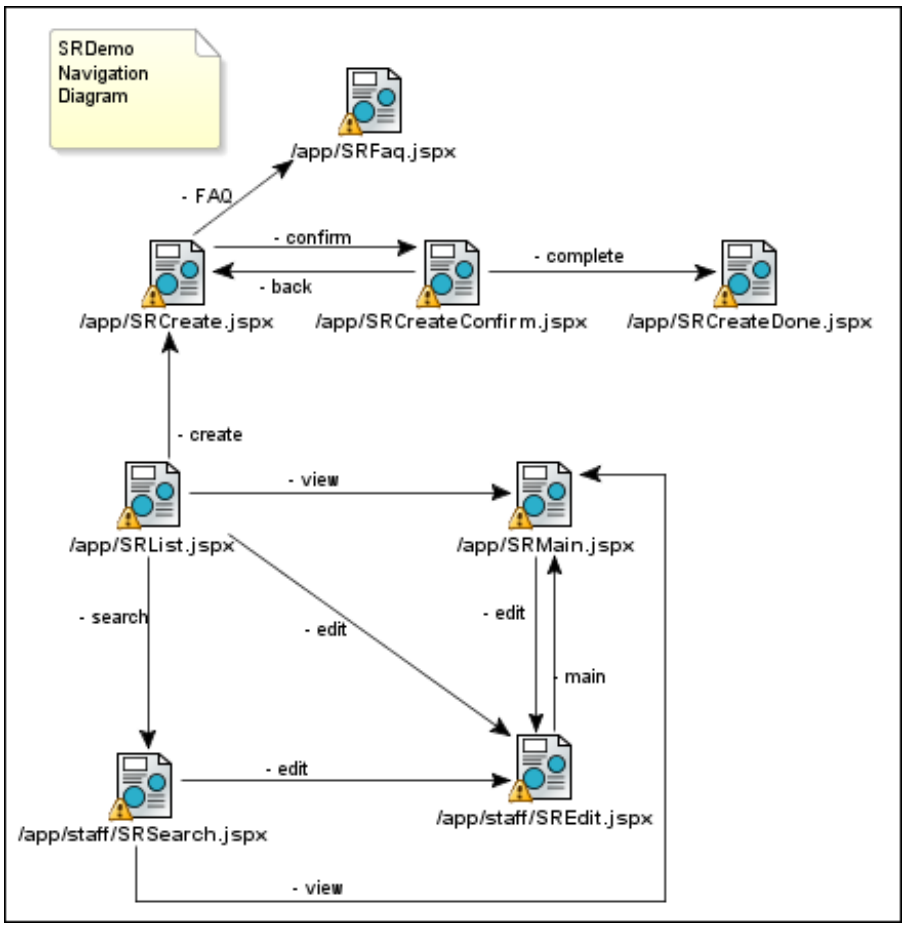
6. この手順を繰り返して、次の表で定義するその他のナビゲーション・ケースをダイアグラムに作成します。

表には、示されているページ間をユーザーが移動する際の経路が示されています。たとえば、ユーザーは SRList ページの「View」ボタンをクリックして SRMain ページに移動し、「Edit」ボタンをクリックして SREdit ページに移動します。

ソース	リンク先	結果
SRList	SRMain	view
SRList	SREdit	edit
SRList	SRSearch	search
SRMain	SREdit	edit
SREdit	SRMain	main
SRSearch	SRMain	view
SRSearch	SREdit	edit
SRList	SRCreate	create
SRCreate	SRFaq	FAQ
SRCreate	SRCreateConfirm	confirm
SRCreateConfirm	SRCreate	back
SRCreateConfirm	SRCreateDone	complete

注意： ナビゲーション・ケースを表す線を曲げる場合、方向を変える位置を1度クリックします。ナビゲーション・ケースは、何か所でも曲げることができます。

7. 「**保存**」をクリックしてダイアグラムを保存します。ダイアグラムは次のスクリーンショットのように表示されます。



グローバル・ナビゲーション・ルールの定義

グローバル・ナビゲーション・ルールは、（通常、Help ボタンや Logout アイコンを使用することによって）すべてのページから利用できるナビゲーション・パスを定義します。これらのパスを定義する際、ダイアグラムは使用しません。かわりに「概要」ページ（または構成エディタ）を使用します。

グローバル・ナビゲーション・ルールを定義するには、次の手順を実行します。

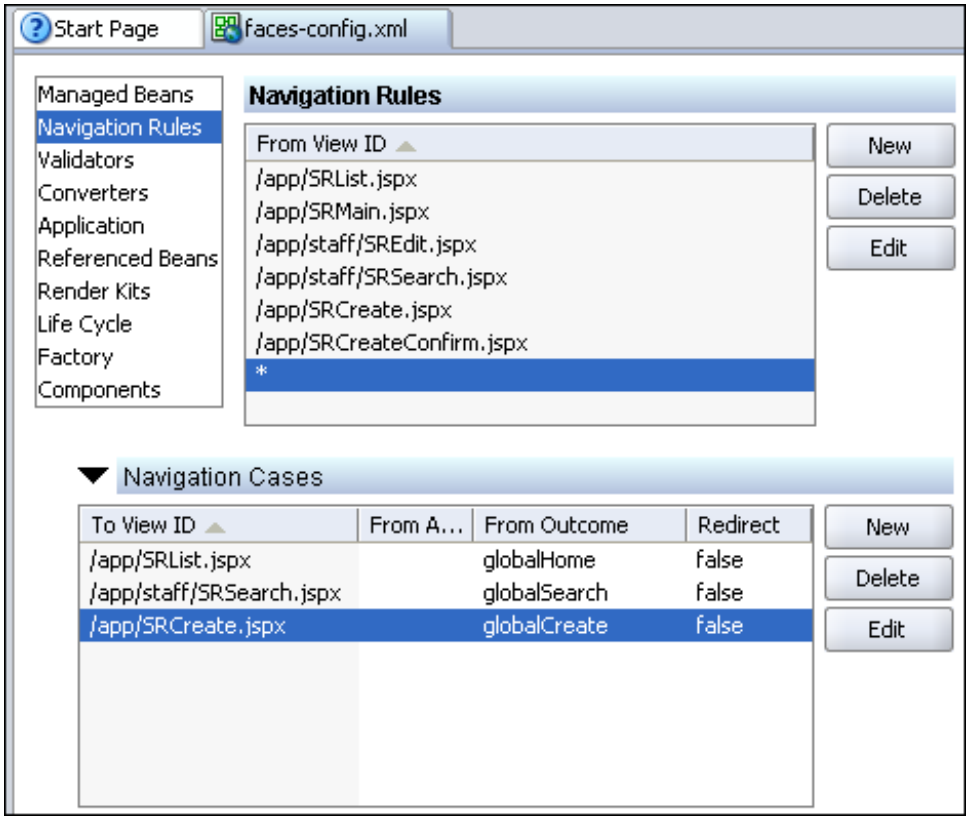
1. ダイアグラム・ページ下部の「**概要**」タブをクリックします。
2. ページの左上の「**ナビゲーション・ルール**」を選択します。「ナビゲーション・ルール」ボックスには、ダイアグラムで作成したルールが表示されます。
3. 右側の「**新規**」ボタンをクリックします。
4. 「ナビゲーション・ルールの作成」ダイアログ・ボックスに、ワイルドカード記号 * を入力し、「**OK**」をクリックします。
5. 「ナビゲーション・ケース」ボックスの右にある「**新規**」ボタンをクリックします。
6. 「ビュー ID(至)」フィールドに **`/app/SRList.jspx`** と入力します（ドロップダウン・リストは、まだページを作成していないので空です）。「結果 (自)」フィールドに **`globalHome`** と入力します。「**OK**」をクリックします。これにより、SRList ページがホームページとして識別されます。ユーザーはアプリケーションのすべてのページからこのホームページに戻ることができます。

7. 次の表で定義されている 2 つのグローバル・ルールも作成します。

ビュー ID(至)	結果 (自)
/app/staff/SRSearch.jspx	globalSearch
/app/SRCreate.jspx	globalCreate

8. 作業内容を保存します。

完了した「ナビゲーション・ルール」ページは、次のスクリーンショットのように表示されます。



まとめ

この章では、アプリケーションのページ、およびユーザーがページ間を移動する際に必要なリンクを示す、ページ・フロー・ダイアグラムを作成しました。これを実現するために、次の主なタスクを実行しました。

- 新しいページ・フロー・ダイアグラムの作成
- ダイアグラムでのアウトライン・ページの作成
- ページ間のナビゲーション・リンクの作成
- グローバル・ナビゲーション・ルールの定義

4

アプリケーション標準の開発

この章では、アプリケーション開発における標準の役割について説明し、標準の SRDemo アプリケーションへの実装方法を示します。

この章の内容は次のとおりです。

- 概要
- コードの再利用
- ユーザー・インタフェースの翻訳の実行
- 状態ホルダーの提供
- 標準のルック・アンド・フィールの作成
- まとめ

概要

プロジェクトがどんなに小さくても、個々のプロジェクト内で定義する必要がある標準の様々なセットがあります。これには、ユーザー・インタフェースの共通のルック・アンド・フィールの確保、可能性のあるコードの重複の回避、およびその他の言語への翻訳の簡易化などが含まれます。この章では、チュートリアルとの関連でこれらのタイプの標準について考慮します。

次の主な作業を実行します。

- アプリケーションの中でコードの再利用を容易にするユーティリティ・ファイルの作成
- ユーザー・インタフェースの複数言語への翻訳の実行
- アプリケーションの標準のルック・アンド・フィールを提供する基本テンプレート・ページの作成

注意： 第 3 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **`Chapter4`** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、**`<jdev_install>¥jdev¥mywork¥Chapter4`** になります。
2. **`<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter3.zip`** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「`<jdev_install>¥jdev¥mywork¥Chapter4¥SRDemo¥SRDemo.jws`」**を選択します。これによって、この章の初期アプリケーションが開きます。

第 3 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

コードの再利用

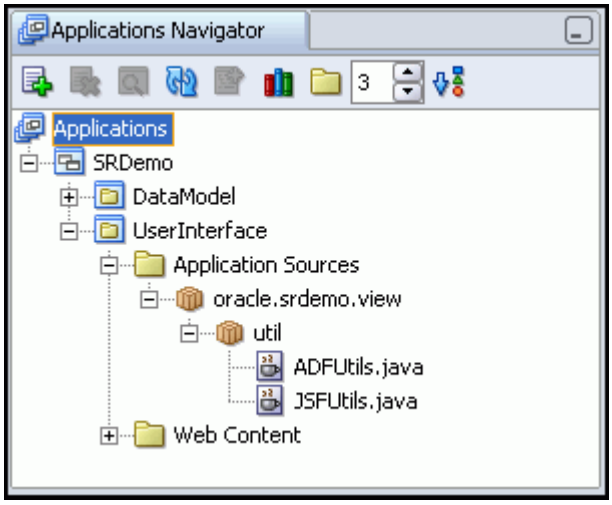
コードを一度作成し、それを再利用するのは、理にかなったことです。SRDemo での再利用に役立つよう、`ADFUtils.java` および `JSFUtils.java` ファイルが作成されており、アプリケーション内の様々なポイントで必要になる可能性のある有用なユーティリティが保持されています。このファイルは両方とも、チュートリアルで提供されています。

アプリケーション内に Java ファイルを作成してこのコードを保持するには、次の手順を実行します。

1. ナビゲータで、**「UserInterface」**プロジェクトを選択し、コンテキスト・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」ペインで、**「General」**ノードを選択し（まだ選択されていない場合）、「項目」ペインで**「Java クラス」**を選択します。「OK」をクリックします。
3. クラス名として **`ADFUtils`** と入力し、パッケージ名として **`oracle.srdemo.view.util`** を指定します。「拡張対象」フィールドが **`java.lang.Object`** に設定されており、その他のフィールドがデフォルトのままになっていることを確認します。**「OK」**をクリックします。コード・エディタが開き、スケルトン・クラスのコードが表示されます。
4. Windows Explorer（別のオペレーティング・システムを使用している場合は同等のもの）で、チュートリアル設定ファイルを解凍したディレクトリに移動します。**`ADFUtils.java`** ファイル（`<tutorial_install>¥files` ディレクトリにあります）を開きます。
5. ファイル内のすべてのテキストを選択し、コピーします。

6. JDeveloper で、`ADFUtils.java` ファイルからスケルトン・コードを削除し、次にクリップボードの内容をファイルのその場所に貼り付けます。ファイルを保存します。
7. ファイルの内容を確認します。ADF バインディングを処理する一連の便利な関数を保持します。ファイル内のコードの行の下に赤の波線がある場合、ADF Faces ランタイム・ライブラリがプロジェクトから欠落していることを示します。このライブラリについては、この章の後で追加します。
8. これらの手順を繰り返して、2 つ目のユーティリティ・ファイル **`JSFUtils.java`** を作成します。このファイルにはいくつかの一般的な static ユーティリティが含まれています。

次のスクリーンショットは、ナビゲータに 2 つのファイルがどのように表示されるかを示しています。



ユーザー・インタフェースの翻訳の実行

JavaServer Faces（JSF）は、ユーザー・インタフェースの複数言語への比較的直接的な翻訳処理を行います。SRDemo のユーザー・インタフェースで使用されているほとんどの文字列は、`UIResources.properties` という 1 つのファイルに定義されています。これは名前と値のペアを含むフラット・テキスト・ファイルで、名前はリソースの抽象識別子で、値は実行時に表示される実際の値です。たとえば、srdemo.browserTitle=SRDemo Sample Application のようになります。

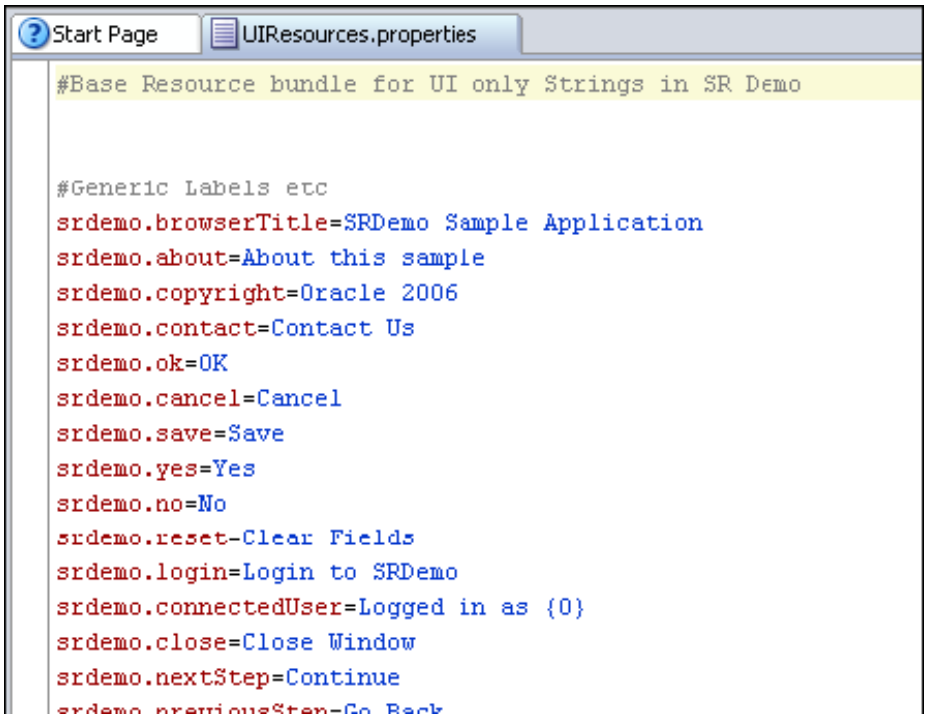
このようなプロパティ・ファイルによって UI の管理が簡単になり、翻訳に非常に役立ちます。アプリケーションの翻訳に必要なのは、「ファイル」→「別名保存」を選択して適切な接尾辞を持つファイル名（たとえば、ドイツ語バージョンの場合は `UIResources_de`）でプロパティ・ファイルを複製することだけです。次に値を翻訳します（リソースの名前は同じにしておきます）。JSF は、実行時にブラウザのロケール情報を調べ、自動的に適切なバンドルをロードします。リソースがロケール固有のバンドルになかった場合、JSF はかわりに基本バンドルを使用します。

便宜上、このチュートリアルでは `UIResources` ファイルが提供されています。ファイルを JDeveloper プロジェクトにインポートするには、次の手順を実行します。

1. ナビゲータで、**「UserInterface」**プロジェクトを選択し、コンテキスト・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」ペインで、**「General」**ノードを選択し（まだ選択されていない場合）、「項目」ペインで**「ファイル」**を選択します。**「OK」**をクリックします。
3. ファイルに **`UIResources.properties`** と名前を付けて、**「OK」**をクリックします。

   JDeveloper では、`UserInterface` プロジェクトの下に「リソース」ノードが作成され、そこにファイルが置かれます。

4. Windows Explorer（または使用しているオペレーティング・システムで同等のもの）で、設定ファイルを解凍したディレクトリに移動します。**UIResources.properties** ファイル（<tutorial_install>¥files ディレクトリにあります）を開き、ファイル内のすべてのテキストを選択してクリップボードにコピーします。
5. JDeveloper で、クリップボードの内容をファイルに貼り付け、保存します。
6. 前述した名前と値のペアをいくつか調べます。全ページでこれらを使用します。次のスクリーンショットは、いくつかの例を示しています。



状態ホルダーの提供

アプリケーション内のページ間を移動する際、いくつかの値を追跡する必要があります。これらの値は、SRList ページから SRMain または SREdit ページのいずれかに移動する際の現在のサービス・リクエスト ID のようなものを表します。問合せ記録を編集しようとしている場合、SRSearch ページでその記録に対するユーザーを追跡する必要があります。

この項では、アプリケーションの状態の追跡に必要なすべての属性を管理するクラスを作成します。次に、マネージド Bean を作成してメソッドをページに公開します。この Bean は、後の章でアプリケーションの状態を表す値を設定または取得するために使用されます。

属性を定義し、アプリケーションの状態を表す値を保持する Java クラスを作成するには、次の手順を実行します。

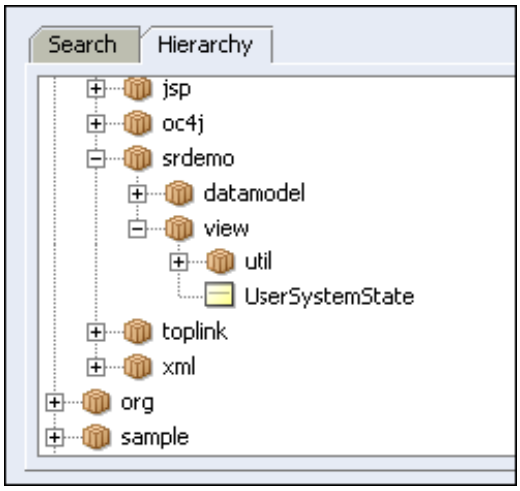
1. 「**UserInterface**」プロジェクトを選択します。コンテキスト・メニューから、「**新規**」を選択します。
2. 新規ギャラリで、「**General**」カテゴリを選択し（まだ選択されていない場合）、「項目」ペインで「**Java クラス**」を選択します。「**OK**」をクリックします。
3. クラス名として **UserSystemState** と入力し、パッケージとして **oracle.srdemo.view** と入力します。その他のすべてのフィールドはデフォルトのままとします。
4. 基本クラスが作成され、コーディング用に準備されます。時間を節約するには、**UserSystemState.txt** ファイル（<tutorial_install>¥files ディレクトリまたは設定ファイルを解凍した場所にあります）を開き、そのコードをコピーします。新しく作成したクラスにそれを貼り付けます。作業内容を保存します。

このクラスのメソッドは、アプリケーション内を移動しているユーザー固有の情報を判断するのに使用されます。前述したように、情報にはユーザー ID、ユーザー・ステータス、現在のサービス・リクエストなどが含まれます。クラスのメソッドを、アプリケーションで簡単に使用できるよう公開する必要があります。これには、次の手順で説明するよう、マネージド Bean を作成します。

1. `faces-config.xml` ファイルがまだ開いていない場合、アプリケーション・ナビゲータで**「UserInterface」→「Web コンテンツ」→「WEB-INF」**を選択し、ファイルを開きます。
2. **「概要」**タブをクリックし、「Managed Bean」ページにアクセスします。
3. **「Managed Bean」**カテゴリが選択された状態で、**「新規」**をクリックします。
4. 「マネージド Bean の作成」ペインで、プロパティ値を次の表の値に設定します。

フィールド	値
名前	`userState`
クラス	`oracle.srdemo.view.UserSystemState` 値を入力するか、「参照」ボタンをクリックして「階層」タブを開き、値を選択することもできます。
有効範囲	`session`
クラスが存在しない場合は生成	チェック・ボックスの選択を解除。

5. 次のスクリーンショットは、マネージド Bean クラスを定義する際にクラス・ブラウザを使用した場合の「階層」タブの使用を示します。



6. **「OK」**をクリックして続行します。

これで、アプリケーションの状態の判断に必要なすべての値を設定または取得する準備ができました。

標準のルック・アンド・フィールの作成

Web アプリケーションを開発する場合、一般的なルック・アンド・フィール（同じ色使い、同じ画面設計およびレイアウト）を維持し、（より重要なこととして）同じような相互作用の動作を作成することで、ユーザーに変わらない環境を提供できます。アプリケーションの各ページの基礎として使用できる簡単なテンプレートを作成できます。これにより、作成する各ページについて同じことを繰り返す必要がないので、開発の生産性が向上します。

SRDemo アプリケーションのすべてのページを開発する際の基礎となるテンプレートを作成するには、次の手順を実行します。

1. ナビゲータで、**「UserInterface」**プロジェクトを選択し、コンテキスト・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」ペインで、**「Web Tier」**ノードを開き（まだ開いていない場合）、**「JSF」**を選択します。
3. 「項目」ペインで、**「JSF」**を選択して**「OK」**をクリックします。JSF JSP の作成ウィザードが起動します。
4. 次の値を使用して、ウィザードの最初の 3 つのステップを完了します。

ウィザード・ステップ 1: JSP ファイル	
フィールド	**値**
ファイル名	SRDemoTemplate.jspx
ディレクトリ名	ファイルが保存される場所。<jdev_install>¥SRDemo¥UserInterface¥public_html¥**Template** フォルダにページを作成してください。
タイプ	JSP Document
モバイル	チェック・ボックスの選択を解除。

5. **「次へ」**をクリックします。

ウィザード・ステップ 2: コンポーネント・バインディング	
フィールド	**値**
マネージド Bean で UI コンポーネントを自動公開しない	このオプションが選択されていることを確認します。（これは後で開発するページの単なるアウトラインなので、ここでは UI コンポーネントのためのマネージド Bean を作成しないよう選択します。後で、詳細な個々のページを作成する際、ページの UI コンポーネントのためにマネージド Bean を作成します。）

6. **「次へ」**をクリックします。

ウィザード・ステップ 3: タグ・ライブラリ	
フィールド	**値**
選択済のライブラリ	ADF Faces Components ADF Faces HTML JSF Core JSF HTML

7. **「終了」**をクリックします。

   ビジュアル・エディタに、空のページが表示されます。

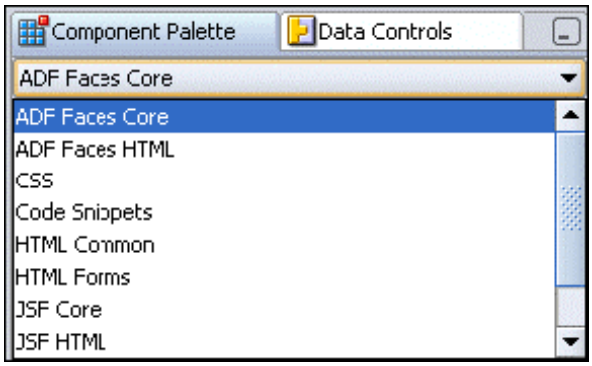
ページへのコンポーネントの追加

次の手順では、テンプレート・ページへのコンポーネントの追加方法を示します。コンポーネント・パレットは、異なるビジュアル・エディタでコンポーネントを作成する際に使用されます。

コンポーネント・パレットには、多くのパレット・ページが用意されています。各パレット・ページには、コンポーネントの論理グループが含まれます。たとえば、JSF ナビゲーション・ダイアグラムを作成したとき、パレット・ページには JSF Page、JSF Navigation Case、Note および Attachment が表示されていました。これらは、JSF ナビゲーション・ダイアグラムに適した項目です。

複数のコンポーネント・セットが使用可能な場合もあります。このような場合、使用するコンポーネントのグループを、コンポーネント・パレットでパレット・ページのリストをクリックして選択します。

次のスクリーンショットは、JSF ページの作成に使用するパレット・ページの一部を示しています。



コンポーネント・パレットが表示されていない場合、**「表示」→「コンポーネント・パレット」**を選択します。

1. コンポーネント・パレットで **ADF Faces Core** ページを選択し、リストで**「PanelPage」**コンポーネントを検索します。
2. **PanelPage** をビジュアル・エディタのテンプレートにドラッグします。
3. プロパティ・インスペクタで、Title プロパティに **`Change me`** と入力します。このタイトルは、後でテンプレート・ページから開発するページに適切に対応するよう変更されます。
4. 次のようにページにブランドを追加します。**`images`** というディレクトリを `<tutorial_install>¥files` ディレクトリで検索します。Windows Explorer（または使用しているオペレーティング・システムで同等のもの）で、それを `<jdev_install>¥jdev¥mywork¥SRDemo¥UserInterface¥public_html` にコピーします。
5. アプリケーション・ナビゲータで、**「Web コンテンツ」**フォルダを選択し、「表示」メニューから**「リフレッシュ」**を選択します。
6. **`images`** ノードを検索し、開いていない場合は開いて、テンプレート・ページの左上のブランド・ファセットに **`SRBranding.gif`** をドラッグします。ポップアップ・ウィンドウで、作成するコンポーネントのタイプとして**「`GraphicImage`」**を選択します。

次の手順で、LoadBundle タグを追加します。このタグは、アプリケーションを国際化する場合に翻訳プロセスで役立つよう使用されます。loadBundle タグは、アプリケーションの jspx ページで使用されるリソース・バンドルを識別します（つまり、前述の項でコピーした UIResources.properties ファイル）。

1. 「**JSF Core**」コンポーネント・パレット・ページを選択します。構造ウィンドウに **LoadBundle** コンポーネントをドラッグし、<>afh:head の上にドロップします。
2. LoadBundle の挿入ポップアップ・ウィンドウで、ベース名を **UIResources** に設定（または「**...**」を使用）し、「プロパティ・ファイル」ラジオ・ボタンを選択して、プロパティ・ファイルを参照します。ファイルが <jdev_install>¥jdev¥mywork¥SRDemo¥UserInterface¥ にあることを覚えておきます。Var プロパティを **res**（res はリソース・バンドルのページ有効範囲別名で、全ページの式言語で使用できます）に設定します。「**OK**」をクリックします。
3. 次のようにコピーライト情報を追加します。ビジュアル・エディタ（または構造ウィンドウ）で、**appCopyright** ファセットを選択します。ファセットは、ページ上で特別な視覚効果を与えるために UI コンポーネントを置くことができる panelPage 内のスロットです。ファセットを右クリックし、「**AppCopyright の中に挿入**」を選択して、コンテキスト・メニューから「**OutputText**」を選択します。
4. 構造ウィンドウで af:outputText が選択された状態で、コンテキスト・メニューから「**プロパティ**」を選択します。「値」フィールドの右の「**バインド**」をクリックします。
5. 「データにバインド」ダイアログ・ボックスの「変数」ツリーで、「**JSP Objects**」ノード、次に「**res**」ノードを開きます。
6. 下にスクロールして **srdemo.copyright** を検索します。それを選択して「>」をクリックし、「式」ペインに移動します。「**OK**」をクリックし、再度「**OK**」をクリックします。（かわりに、プロパティ・インスペクタの Value プロパティに直接 **#{res['srdemo.copyright']}** と入力できます。）

   「ユーザー・インタフェースの翻訳の実行」のスクリーンショットは、UIResources ファイル内の名前と値のペアの 1 つとしてこれを示しています。
7. Escape プロパティを **false** に変更し、リソース文字列内の © マークアップが正しいコピーライト記号として印刷されるようにします。
8. 次のように About リンクを追加してアプリケーションの情報を提供します。ビジュアル・エディタで **appAbout** ファセットを選択します。コンテキスト・メニューから、「**appAbout の中に挿入**」を選択し、次に「**CommandLink**」を選択します。
9. 構造ウィンドウで commandLink を選択し、コンテキスト・メニューから「**プロパティ**」を選択します。
10. 「CommandLink のプロパティ」ダイアログ・ボックスで、Text プロパティの右の「**バインド**」をクリックします。
11. 「データにバインド」ダイアログの「変数」ツリーで、「**JSP Objects**」、次に「**res**」を開きます。
12. スクロールして **srdemo.about** を検索し、「**>**」をクリックして「式」ペインに移動します。「**OK**」をクリックし、再度「**OK**」をクリックします。（かわりに、プロパティ・インスペクタの Text プロパティに直接 **#{res['srdemo.about']}** と入力できます。）
13. Immediate プロパティを **true** に設定します。

    Help リンク（ここでは About ボタン）のように画面内のデータを適用する必要がない場合、Immediate プロパティを使用して JSF ライフサイクルをショートカットします。
14. 次のようにラベルを追加して現在ログインしているユーザーの情報を表示します。構造ウィンドウで「**InfoUser**」ファセットを選択し（「**PanelPage facets**」ノードを開いて検索）、コンテキスト・メニューから「**infoUser の中に挿入**」を選択し、次に「**JSF HTML**」を選択します。「JSF HTML 項目の挿入」ダイアログ・ボックスから、「**Output Format**」を選択します。「**OK**」をクリックします。

    「Output Format」は標準の JSF HTML タグで、ローカライズされたメッセージの表示に使用されます。

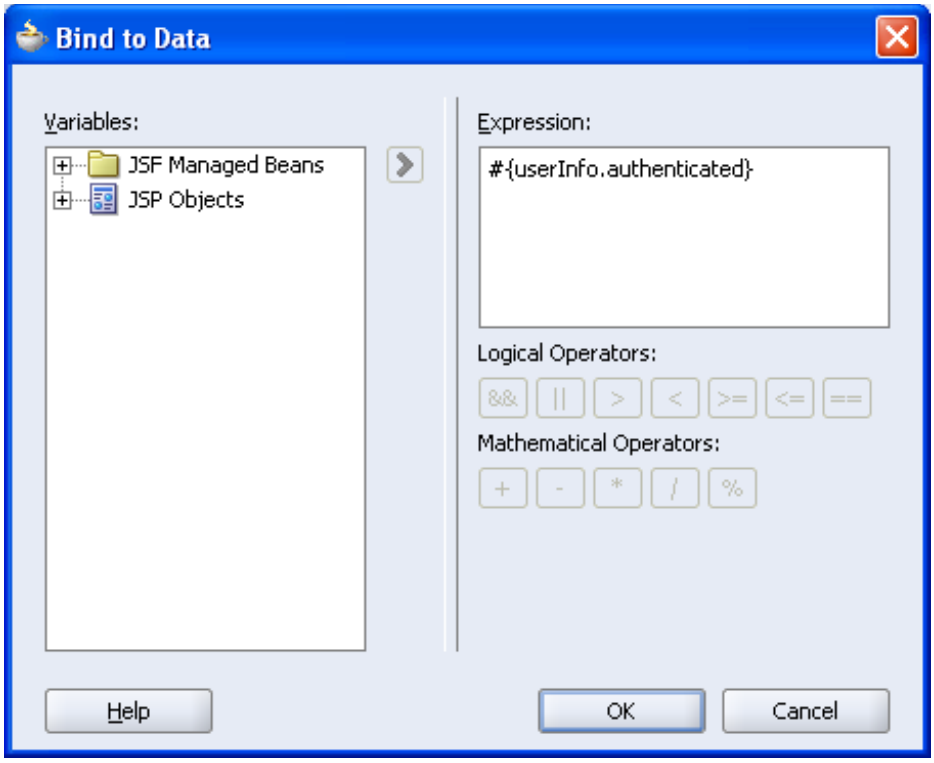
15. 構造ウィンドウで、**「h:outputFormat」**を右クリックして、コンテキスト・メニューから**「プロパティ」**を選択します。「Output Format のプロパティ」ダイアログ・ボックスで、**Value** プロパティ右の**「Bind」**ボタンをクリックします。

16. 「データにバインド」ダイアログ・ボックスの「変数」ツリーで、**「JSP Objects」**ノード、次に**「res」**ノードを開きます。

17. リストをスクロールして **srdemo.connectedUser** を検索します。それを選択して**「>」**をクリックし、「式」ペインに移動します。**「OK」**をクリックします。

18. 「Output Format のプロパティ」ウィンドウで、**Rendered** プロパティ右の**「Bind」**ボタンをクリックし、「式」フィールドで **`#{userInfo.authenticated}`** と入力します。**「OK」**をクリックし、再度**「OK」**をクリックします。

    userInfo マネージド Bean は、コンテナ・セキュリティから取得されたセキュリティ情報へのアクセスを提供します。ここで、現在のセッションが実際に認証されているかを確認します。まだ userInfo マネージド Bean は作成していません。後の章で作成します。

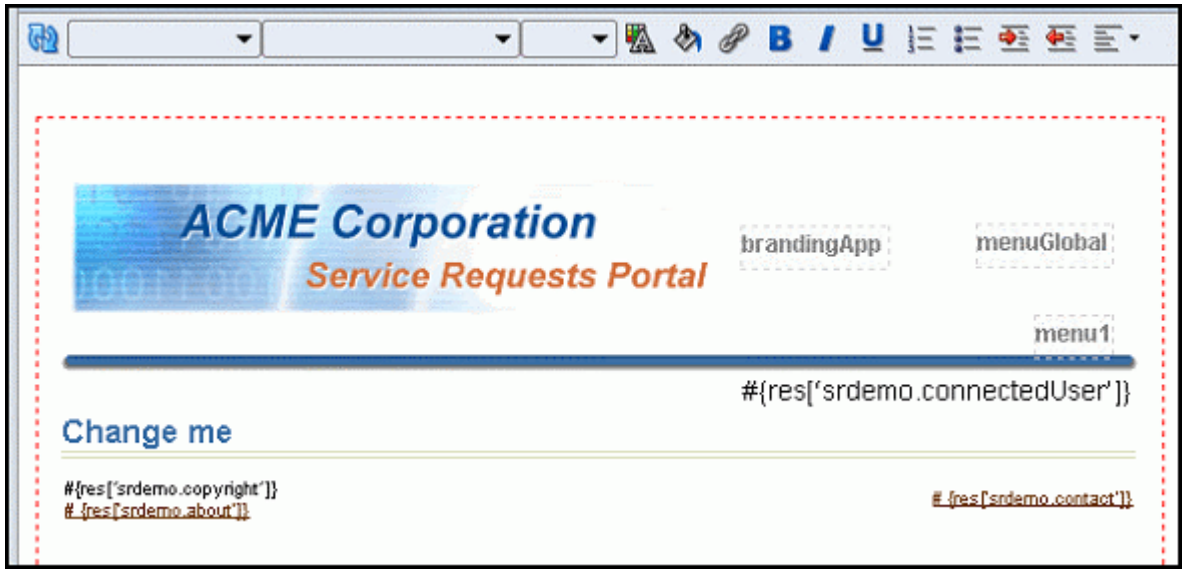


19. 次のようにパラメータを定義してログインしているユーザーの名前に渡します。構造ウィンドウで**「`h:outputFormat`」**を右クリックし、コンテキスト・メニューから**「h:outputFormat の中に挿入」**→**「Param」**を選択します。「Param の挿入」ダイアログ・ボックスで、「値」フィールドに **`#{userInfo.userName}`** と入力します。**「OK」**をクリックします。

20. 次のように連絡先情報を追加します。構造ウィンドウで、**appPrivacy** ファセットを検索します。次に右クリックして、コンテキスト・メニューから**「appPrivacy の中に挿入」**→**「CommandLink」**を選択します。

21. プロパティ・インスペクタで、Text プロパティに **`#{res['srdemo.contact']}`** と入力し、Action プロパティに **`dialog:globalContact`** と入力します。この形式（`dialog:`）は ADF Faces の機能で、JSF 仕様では提供されません。

22. SRDemoTemplate を保存します。テンプレート・ページは、次のスクリーンショットのように表示されます。



まとめ

この章では、SRDemo アプリケーションにいくつかの標準を組み込みました。これを実現するために、次の主なタスクを実行しました。

- 一般的に使用されるユーティリティを保持するようファイルを作成
- アプリケーションの UI の翻訳が簡単になるようリソース・ファイルを作成
- アプリケーション・ページの標準のルック・アンド・フィールを提供するようテンプレート・ページを作成

5

単純な表示ページの開発

この章では、SRDemo アプリケーションの中心となる単純な表示ページの SRList ページの作成方法について説明します。このページを使用するとサービス・リクエストの情報を表示できます。

この章の内容は次のとおりです。

- 概要
- ページ・アウトラインの作成
- ページへのユーザー・インタフェース・コンポーネントの追加
- 「Edit」ボタンの組込み
- 「View」ボタンの組込み
- リフレッシュ動作の定義
- ページへのメニュー・バーの追加
- ドリルダウン・リンクの追加
- ページの実行
- まとめ

概要

SRList ページは SRDemo アプリケーションの中心です。ユーザーがログインした後に最初に見るページで、このページからアプリケーション内のその他のすべてのページに移動できます。

注意： 第 4 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **`Chapter5`** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、**`<jdev_install>¥jdev¥mywork¥Chapter5`** になります。
2. **`<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter4.zip`** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「`<jdev_install>¥jdev¥mywork¥Chapter5¥SRDemo¥SRDemo.jws`」**を選択します。これによって、この章の初期アプリケーションが開きます。

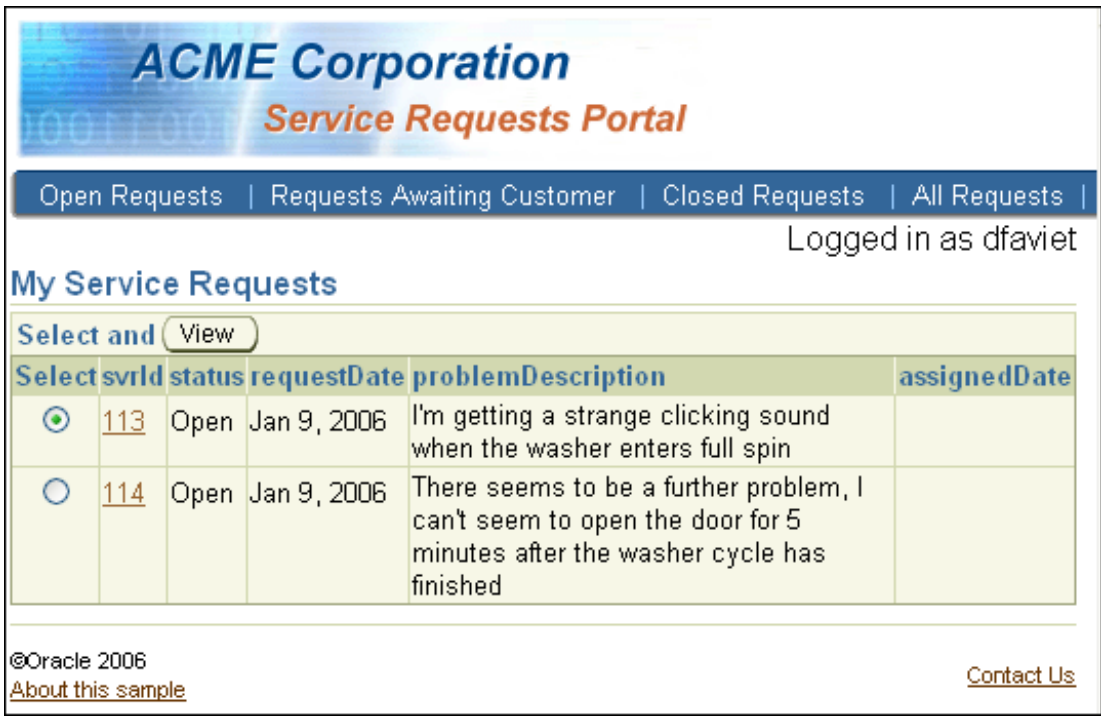
第 4 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

ナビゲータで「`faces-config.xml`」ファイルをダブルクリックして、作成したページ・フロー・ダイアグラムに再度移動し、SRList ページと他のページとの関連を参照します。

ここでは、SRList ページに関して注意する点をいくつか示します。

- ページには、既存の各サービス・リクエストとそのステータスが表示されます。
- すべてのユーザー（顧客、技術者またはマネージャ）がページにアクセスできます。
- 顧客がログインすると、その顧客に関するすべてのサービス・リクエストが表示されます。「View」ボタンを使用できます。
- 技術者がログインすると、その技術者に割り当てられたすべてのサービス・リクエストが表示されます。「View」および「Edit」ボタンを使用できます。
- 表示されるリクエストのリストを、2 番目のレベルのメニューをクリックして、現在オープンしているすべてのリクエスト、ステータスがクローズドのリクエストのみ、またはステータスに関係なくすべてのリクエスト、のいずれかを表示することで、フィルタ処理できます。デフォルトは開かれているリクエストです。
- すべてのユーザーがサービス・リクエストを選択することができ、「View」ボタンをクリックすることで SRMain ページに移動し、そのリクエストの履歴を更新できます。また、技術者は「Edit」ボタンをクリックして SREdit ページに移動し、ここでリクエストを編集できます。
- 新しいリクエストを追加するには、「Create New Service Request」メニュー・オプションを選択して SRCreate ページに移動できます。
- 技術者は「Advanced Search」メニュー・タブを使用して「Search」ページ（SRSearch）に移動できます。
- マネージャはすべてのサービス・リクエストを参照できます。マネージャはすべてのメニュー・オプションを使用できます。

次のスクリーンショットは、終了した SRList ページがどのように表示されるかを示しています。



前述のリストで説明した機能およびルック・アンド・フィールを備えた SRList ページを作成するには、ここで次の主なタスクを実行します。

- 第 4 章で作成したテンプレート・ページを基にしたページ・アウトラインの作成
- ページへのユーザー・インタフェース・コンポーネントの追加
- 「View」ボタンおよび「Edit」ボタンの組込み
- リフレッシュ動作の定義
- ユーザーが異なるステータスのサービス・リクエストを表示したり、新しいサービス・リクエストを作成できるよう、メニュー・バーの作成
- ユーザーが表内の行を選択し、SRMain ページに移動して、選択されたサービス・リクエストの情報を追加できるよう、ドリルダウン機能の作成

ページ・アウトラインの作成

この項では、SRList ページを作成し、テンプレートを追加して適切なルック・アンド・フィールを適用します。

1. 開いていない場合は、**「faces-config.xml」**ファイルをダブルクリックして、ページ・フロー・ダイアグラムを表示します。
2. **SRList** ページをダブルクリックして、JSF ページ・ウィザードを起動します。
3. 次の値を使用して、ウィザードの最初の 3 つのステップを完了します。

ウィザード・ステップ 1: JSP ファイル	
フィールド	**値**
ファイル名	SRList.jspx
ディレクトリ名	ファイルが保存される場所。¥SRDemo¥UserInterface¥public_html¥app フォルダにページを作成してください。
タイプ	JSP Document
	チェック・ボックスの選択を解除。

4. **「次へ」**をクリックします。

ウィザード・ステップ 2: コンポーネント・バインディング	
新規マネージド Bean での UI コンポーネントの自動公開	このオプションが選択されていることを確認します。
名前	backing_app_SRList
クラス	SRList
パッケージ	oracle.srdemo.userinterface.backing.app

5. **「終了」**をクリックして、ページ詳細を作成します。ビジュアル・エディタに、新規の SRList ページが表示されます。
6. まだ開かれていない場合は、**SRDemoTemplate** ファイルを開きます。構造ウィンドウで、**「afh:html」**ノードを閉じて選択します。コンテキスト・メニューから、**「コピー」**を選択します。
7. タブをクリックして SRList ページに戻り、構造ウィンドウで、**「f:view」**ノードを開きます。
8. **「html」**ノードを削除します。**「f:view」**を右クリックして、コンテキスト・メニューから**「貼付け」**を選択します。先に作成したルック・アンド・フィールが新しいページに適用されます。

ページへのユーザー・インタフェース・コンポーネントの追加

ページにユーザー・インタフェース要素を追加するには、次の手順を実行します。

1. 次のようにタイトルをページに追加します。ビジュアル・エディタで「page」をクリックして選択します。（構造ウィンドウで、**「af:panelPage」**を選択することもできます。）構造ウィンドウで、コンテキスト・メニューから**「プロパティ」**を選択します。
2. 「PanelPage のプロパティ」ダイアログ・ボックスで、Title プロパティで**「バインド」**をクリックします。
3. 「データにバインド」ダイアログ・ボックスの「変数」ツリーで、**「JSP Objects」**ノード、次に**「res」**ノードを開きます。
4. スクロールして **srlist.pageTitle** を検索します。それを選択して「式」ペインに移動します。**「OK」**をクリックし、再度**「OK」**をクリックします。

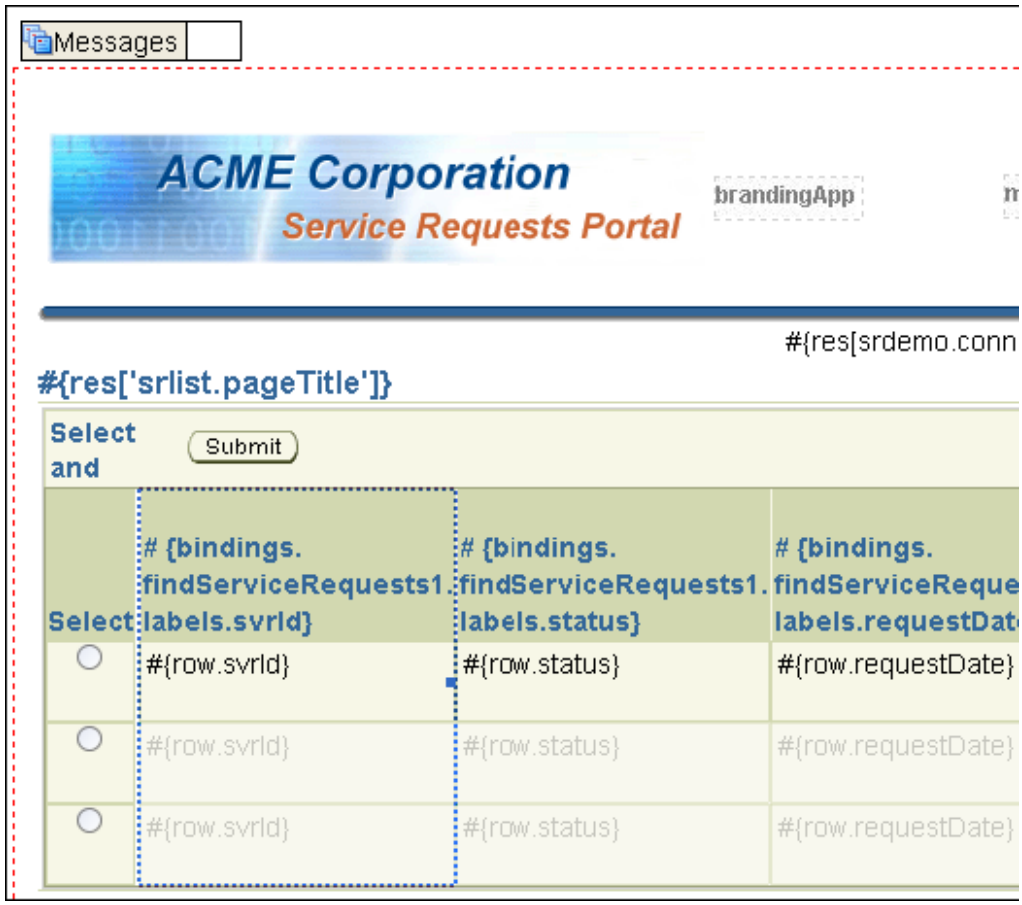
   これは、前述の章で作成した `UIResources.properties` ファイルで定義されたページ・タイトル・リソースの名前です。実行時に表示される値は、ページの実際のタイトルです。
5. 次のようにページの実行時にブラウザ・タイトルに表示されるヘッダーを追加します。構造ウィンドウで**「afh:head」**を選択します。プロパティ・インスペクタで、Title プロパティを **`#{res['srdemo.browserTitle']}`** に設定します。

   これらの値は、「プロパティ」ページ、次に「データにバインド」ダイアログ・ボックスを起動し、前述のステップで行ったように「変数」ツリーから選択して設定できます。（プロパティ・インスペクタで、値を入力することもできます。）
6. ここで、データをページに追加します。データは、findServiceRequests データ・コレクションに基づいた読取り専用表の形式です。最初に、データ・コントロール・パレットがビジュアル・エディタの右に表示されていることを確認します。ない場合、**「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。
7. **「SRPublicFacadeLocal」**ノードを開き、データ・コレクションのリストから**「findServiceRequests(Integer, String)」→「ServiceRequests」**を選択します。
8. コレクションを構造ウィンドウにドラッグし、af:panelPage の上にドロップします。「作成」ポップアップ・メニューから、**「表」→「ADF 読取り専用表」**を選択します。
9. アクション・バインディング・エディタで、`filedBy` パラメータの値を **`${userInfo.userId }`** に、`status` パラメータの値を **`${userState.listMode }`** に設定します。**「OK」**をクリックします。

   > **注意：** まだ `userInfo` クラスおよびマネージド Bean は作成していません。次の章で作成します。

10. 「表の列の編集」ダイアログ・ボックスで、列を **srvId** がリストのトップになり、**status**、**requestDate**、**problemDescription** および **assignedDate** が続くよう列を並べ替えます。これにより、ページ上の列の順序が決定されます。「使用するコンポーネント」列に、すべての列に対して**「ADF 出力テキスト」**が設定されていることを確認します。**「選択を有効にする」**チェック・ボックスを選択して、オプション選択列を表に追加し、**「OK」**をクリックします。

11. ビジュアル・エディタで表が選択された状態で、プロパティ・インスペクタの Id プロパティを **srtable** に変更します。ページを保存します。ここで、ページは次のスクリーンショットのように表示されます。



「Edit」ボタンの組込み

SRList ページは、顧客や技術者が最初に見るページになります。このページから、ユーザーは編集ページまたは表示ページのいずれかに移動できます。「Edit」ボタンは、技術者またはマネージャのログイン時のみ使用可能です。このボタンをクリックすると、ユーザーは SREdit ページに移動して、現在選択しているサービス・リクエストの詳細を変更できます。

デフォルト送信ボタンを「Edit」ボタンに変更するには、次の手順を実行します。

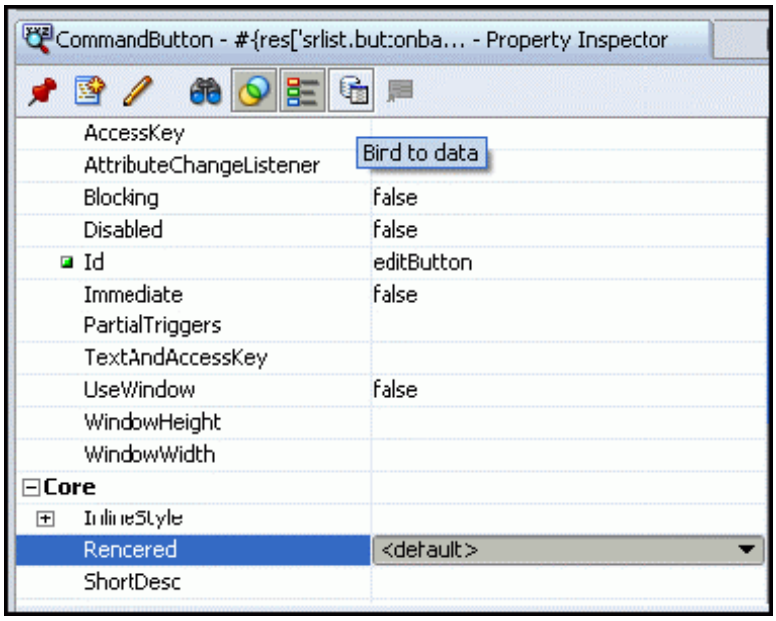
1. 「**Submit**」ボタンをクリックします。プロパティ・インスペクタで、Text プロパティを **#{res['srlist.buttonbar.edit']}** に変更します。

   これは、`UIResources.properties` ファイルで定義されたボタン・リソースの名前です。実行時に表示される値は Edit です。

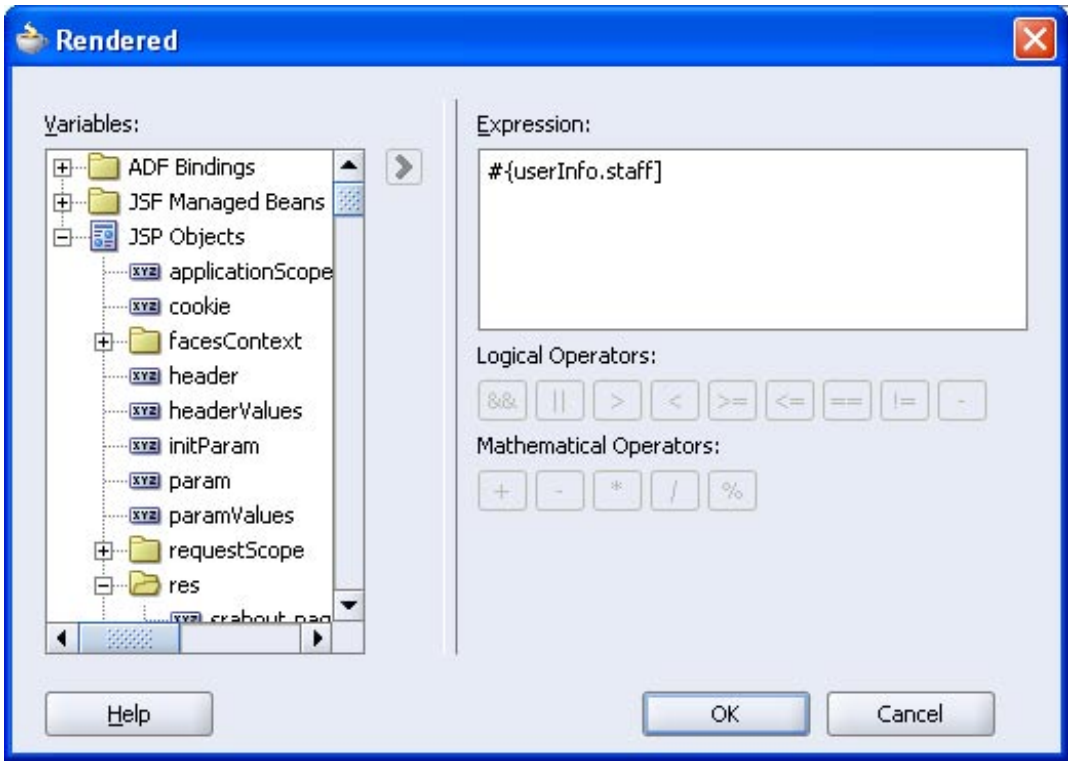
2. また、プロパティ・インスペクタで、Id プロパティを **editButton** に設定します。

   JDeveloper は、このボタンの元の ID を使用するため、すべてのソース・コードを検索します。それらのリファレンスすべてを自動的に新しい名前に変更します。

3. 現在ログインしているユーザーがスタッフ・メンバーの場合のみ、「Edit」ボタンを表示するよう指定します。**Rendered** プロパティを選択して、プロパティ・インスペクタのツールバーで**「データにバインド」**ボタン（右から 2 番目のボタン。ツールチップ・ラベルを使用して、必要なボタンを確認します）をクリックします。



4. Rendered ダイアログ・ボックスで、「式」フィールドに **`#{userInfo.staff}`** と入力します。**「OK」**をクリックします。（まだこのクラスおよびマネージド Bean は作成していません。次の章で作成します。）これにより、ログインしているユーザーがスタッフのメンバーであることを確認します。



5. 現在選択済のサービス・リクエストの ID を編集ページに渡す、バッキング Bean のメソッドを作成します。これにより、該当するレコードが取得および表示できます。ビジュアル・エディタで、**「Edit」**ボタンをダブルクリックして、バッキング Bean を設定するダイアログを起動します。

6. 「バインド Action プロパティ」ダイアログ・ボックスで、**「OK」**をクリックして、`editButton_action` メソッドをバッキング Bean に追加します。

7. `editButton_action` で、コードを追加してサービス・リクエスト ID を現在の行に設定し、使用する JSF ナビゲーション・ケースの名前を戻す必要があります。ナビゲーション・ケース名への戻りを設定すると、JSF はその値をとり、ユーザーをそのナビゲーション・ケースの最後のページに移動させます。次のコードを `editButton_action` メソッドに追加します。

```
setCurrentSvrIdFromRow();
return "edit";
```

8. `setCurrentSvrIdFromRow` メソッドはまだ存在しないので、JDeveloper はコード・エラーとしてそれにフラグを付けます。**「CodeAssist」**アイコン（左マージンの電球）をクリックしてメソッドを作成します。

9. 次のコードを作成したばかりのメソッドに追加して、`setCurrentSvrIdFromRow` メソッドを実装します。（コード・アシストにより要求された場合は、[Alt] を押しながら [Enter] を押して、適切なパッケージをインポートします。）

   このコードでは、次の 2 つのことを行います。

   - 現在の行サービス・リクエスト ID を取得し、UserState マネージド Bean に保存します。
   - 編集の終了時に SRList ページに戻るためのナビゲーション・パスを設定します。

```
FacesContext ctx = FacesContext.getCurrentInstance();
JUCtrlValueBindingRef tableRowRef =
   (JUCtrlValueBindingRef) this.getSrtable().getRowData();
Integer svrId =
   (Integer)tableRowRef.getRow().getAttribute("svrId");

UserSystemState.storeCurrentSvrID(svrId);

//Store away where we want to come back to
UserSystemState.storeReturnNavigationRule
  ("globalHome");
```

10. バッキング Bean に、さらに新規のクラス変数を次のように追加します。

```
private BindingContainer bindings;
```

    （コード・アシストにより要求された場合は、[Alt] を押しながら [Enter] を押して、`oracle.binding` パッケージをインポートします。）

11. 自動バインドをオフにしてページを作成しているので、このステップでバインディングを手動で追加する必要があります。**「bindings」**を右クリックして、コンテキスト・メニューから**「アクセッサの生成」**を選択します。「アクセッサの生成」メニューで、**「OK」**をクリックして **setBindings** メソッドと **getBindings** メソッドを生成します。

12. **SRList.java** ファイルを保存します。

13. ページをクリックし、コンテキスト・メニューから**「実行」**を選択してページを実行します。ただし、表に「no rows yet」と表示されます。これは、ページが特定のログインしたユーザーに対するサービス・リクエストを基にしているためです。まだアプリケーションがどんなタイプのログインでも使用するよう設定していないので、ページから行は戻されません。

「View」ボタンの組込み

「View」ボタンはアプリケーションのすべてのユーザーから使用可能なので、SRMain ページに移動して、選択済のリクエストの履歴を更新できます。

次の手順を実行して、「View」ボタンの機能を指定します。

1. ビジュアル・エディタで **SRList.jspx** に戻ります。
2. 作成した**「Edit」**ボタンを選択します。
3. 右クリックして、コンテキスト・メニューから**「Command Button の後ろに挿入」→「Command Button」**を選択します。
4. プロパティ・インスペクタで、Text プロパティを **#{res['srlist.buttonbar.view']}** に、Id プロパティを **viewButton** に設定します。
5. 「Edit」ボタンと同様に、「View」ボタンのクリック時に発生する必要があることを指定する、バッキング Bean のメソッドを作成する必要があります。**「View」**ボタンをダブルクリックして、ページのバッキング Bean を起動します。「バインド Action プロパティ」ダイアログ・ボックスで、**「OK」**をクリックして、viewButton_action メソッドをバッキング Bean に追加します。
6. SRList.java ファイルで、次のコードを viewButton_action() メソッドに追加します。

```
return drillDown_action();
```

7. drillDown_action メソッドはまだ存在しないので、**「コード・アシスト」**アイコンをクリックして作成します。

   このメソッドでは、前述の項で作成した setCurrentSvrIdFromRow メソッドを再度使用します。現在選択済のサービス・リクエストの ID を SRMain ページに渡します。これにより、該当するレコードが取得および表示できます。このメソッドは、サービス・リクエストをドリルダウンするページで、後で再度使用します。

8. メソッドに次のコードを追加して、この動作を指定します。

```
setCurrentSvrIdFromRow();
return "view";
```

9. ファイルを保存します。

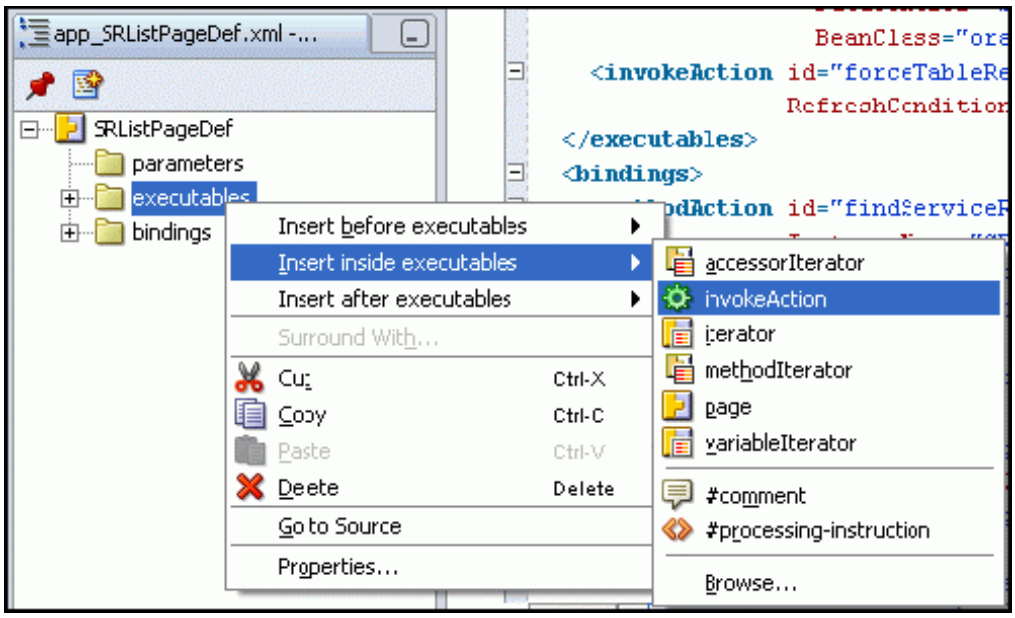
リフレッシュ動作の定義

ユーザーは、アプリケーションのどこからでも SRList ページに戻れます。次の手順では、userState.refresh パラメータを作成し、別のページから戻った際に「List」ページをリフレッシュする必要があるかを決定します。

ページをリフレッシュする場合（たとえば、データが更新されたとき）、UserState Bean のこのパラメータを True に設定し、その結果、式言語がこれを選択して「List」ページに対して問合せを実行します。リフレッシュの例外は、ページが JSF ポストバックから起動される場合です。ポストバックは、ページに対するユーザーの操作によりページがリフレッシュされた場合に発生します。発生した場合は、ページを強制的にリフレッシュする必要はありません。

ページのリフレッシュ動作を指定するには、次の手順を実行します。

1. SRList ページに戻ります。構造ウィンドウで、「**af:panelPage**」を右クリックして、コンテキスト・メニューから「**ページ定義に移動**」を選択します。
2. 構造ウィンドウで、「**SRListPageDef**」ノードを開きます（まだ開いていない場合）。「**executables**」を右クリックして、コンテキスト・メニューから「**executables の中に挿入**」→「**invokeAction**」を選択します。



3. id を **forceTableRefresh** に設定します。
4. 「Binds」フィールドで、矢印をクリックして「**findServiceRequests**」を選択します。
5. 「**アドバンスト・プロパティ**」タブをクリックして、「**RefreshCondition**」プロパティで「**...**」をクリックします。
6. Advanced Editor で、次のコード行を式にコピーします。

   ```
   ${(userState.refresh) and (!adfFacesContext.postback)}
   ```

   この式は、この同じページからのポストバックの一部としてページがコールされた場合という条件を追加します。その結果、リフレッシュは行われません。
7. 「**OK**」をクリックし、再度「**OK**」をクリックします。
8. ページを保存します。

ページへのメニュー・バーの追加

SRList ページはメニュー・オプションのセットがあります。これにより、ユーザーはステータスがオープン、クローズドまたはペンディングのサービス・リクエストを表示するよう選択できたり、ステータスに関係なくすべてのリクエストを表示できます。新しいサービス・リクエストを作成するリンクもあります。

ページにメニュー・オプションを追加するには、次の手順を実行します。

1. 最初に、2 番目のレベルのメニュー・バーを作成し、メニュー・オプションを保持します。構造ウィンドウで、**「PanelPage facets」**ノードを開き、**「menu2」**にスクロール・ダウンします。右クリックして、メニューから**「menu2 の中に挿入」**→**「MenuBar」**を選択します。

2. **データ・コントロール・パレット**で、**「SRPublicFacadeLocal」**ノードを開いて**「findServiceRequests(Integer, String)」**を選択します。構造ウィンドウの **af:menuBar** にドラッグします。「作成」ポップアップ・メニューで、**「メソッド」**→**「ADF コマンド・リンク」**を選択します。

3. プロパティ・インスペクタで、Title プロパティに **`#{res['srlist.menubar.openLink']}`** と入力します。

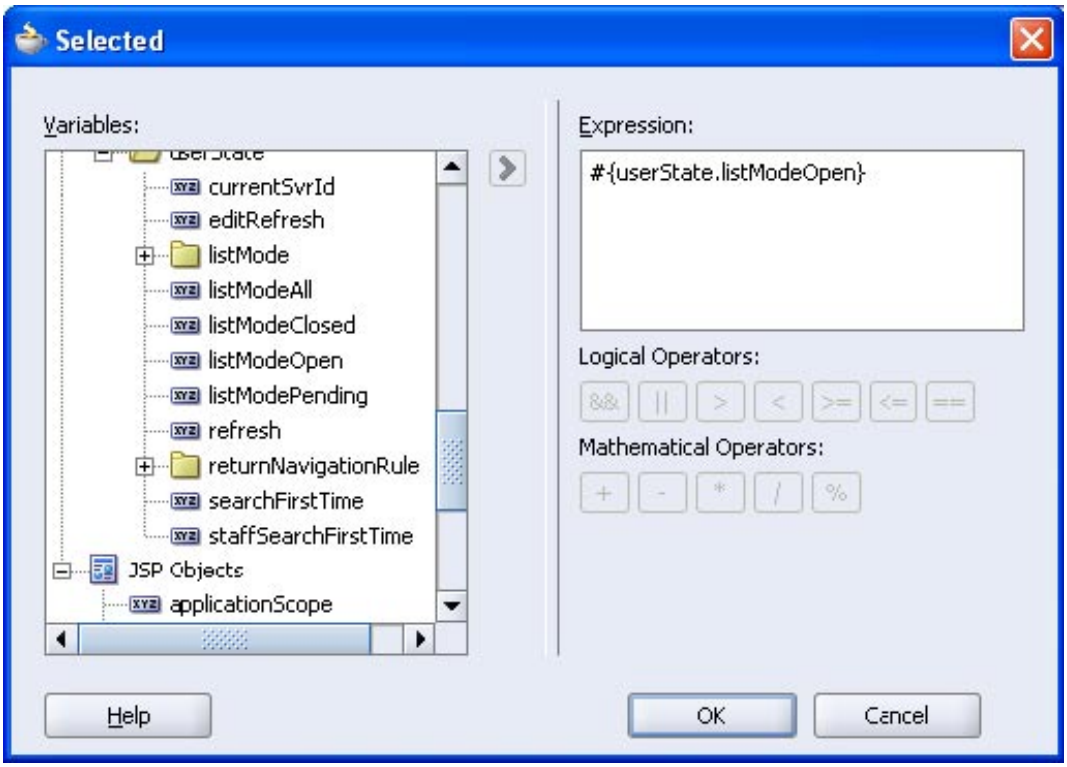
4. 手順 2 をさらに 3 回繰り返し、全部で 4 つのメニュー・オプションを作成します。各リンクについて Text プロパティを次のように設定します。

   `#{res['srlist.menubar.pendingLink']}` **`#{res['srlist.menubar.closedLink']}`**
   `#{res['srlist.menubar.allRequests']}`

5. 各コマンド・リンクがメニュー・バーのオプションとして表示されるよう、次のようにコマンド・メニュー項目に変換します。各コマンド・リンクを右クリックし、メニューから**「変換」**を選択します。「CommandLink の変換」ダイアログ・ボックスで、**「CommandMenuItem」**を選択して**「OK」**をクリックし、これらの要素をメニュー項目に変更します。

   選択したタブがハイライト表示されるようにするには、式言語を使用してリンクに使用されるステータス・パラメータを確認します。前述のレッスンで作成した userState マネージド Bean では便利な関数（`isListModeOpen`）が定義されています。

   ステータスがオープン（デフォルト）のリクエストに対してこれを行うには、openLink メニュー項目について **Selected** プロパティをクリックし、プロパティ・インスペクタのツールバーで**「データにバインド」**ボタン（右から 2 番目）をクリックします。「Selected」ダイアログ・ボックスで、**「JSF マネージド Bean」**ノード、次に**「userState」**ノードを開きます。リストから、**「listModeOpen」**を選択します。**「>」**をクリックし、「式」ペインに移動します。**「OK」**をクリックします。



6. その他のメニュー項目に対しても同じことをし、次の表で定義されている値を選択します。

メニュー項目	式
Pending Requests	`#{userState.listModePending}`
Closed Requests	`#{userState.listModeClosed}`
All Requests	`#{userState.listModeAll}`

Open コマンド・リンクに actionListener を設定します。actionListener は、イベントが発生した際（たとえば、ユーザーがリンクをクリックした場合）起動します。actionListener が Open（手順 6 で定義）に設定した `userState.listMode` パラメータで起動した場合、Open のステータスを持つサービス・リクエストが表示されます。

7. 構造ウィンドウで、**最初のコマンド・メニュー項目**を右クリックし、コンテキスト・メニューから**「af:commandMenuItem の中に挿入」→「ADF Faces Core」**を選択します。「ADF Faces Core 項目の挿入」ダイアログ・ボックスで、**「setActionListener」**を選択します。**「OK」**をクリックします。

8. 「SetActionListener の挿入」ダイアログ・ボックスで、**「From*」**を**「#{'Open'}」**に、**「To*」**を**「#{userState.listMode}」**に設定します。

   （値を入力したり、**「バインド」**ボタンをクリックして「データにバインド」ダイアログ・ボックスで**「JSF マネージド Bean」**ノード、次に**「userState」**ノードを開くことができます。リストから**「listMode」**を選択し、「式」ペインに移動します。**「OK」**をクリックします。）

9. 「**OK**」をクリックします。その他の 3 つのコマンド・メニュー項目それぞれの中に同じ方法で actionListeners を設定し、次の表で定義されている値を選択します。

メニュー項目	From*	To*
Pending Requests	`#{'Pending'}`	`#{userState.listMode}`
Closed Requests	`#{'Closed'}`	`#{userState.listMode}`
All Requests	`#{'%'}`	`#{userState.listMode}`

10. 次のように commandMenuItem をメニュー・バーに追加し、ユーザーが新しいサービス・リクエストを作成できるようにします。コンポーネント・パレットで **ADF Faces Core** ページを選択し、次にメニュー・バーに **CommandMenuItem** をドラッグします。

11. プロパティ・インスペクタで、Text プロパティを **`#{res['srlist.menubar.newLink']}`** に、Action プロパティを `create` に設定します。

12. ページを保存します。

ドリルダウン・リンクの追加

ここで、最初の列にリンクを追加し、ユーザーがサービス・リクエストでドリルダウンし、SRMain ページに移動できるようにします。このページでは、ユーザーはサービス・リクエストに新しい情報を追加できます。

ページにドリルダウン・リンクを追加するには、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、「**findServiceRequests(Integer, String)**」ノード、次に「**ServiceRequests**」ノードを開きます。下にスクロールして「**操作**」を検索し、同じようにそのノードを開きます。

2. リストで、「**setCurrentRowWithKey**」を選択して、表の **svrId** 列で既存のテキストの次にドラッグします。ポップアップ・メニューから、「**操作**」→「**ADF コマンド・リンク**」を選択します。

3. アクション・バインディング・エディタで、リンクへのパラメータの値を `${bindings.setCurrentRowWithKey}` から **`#{row.rowKeyStr}`** に変更します。「**OK**」をクリックします。

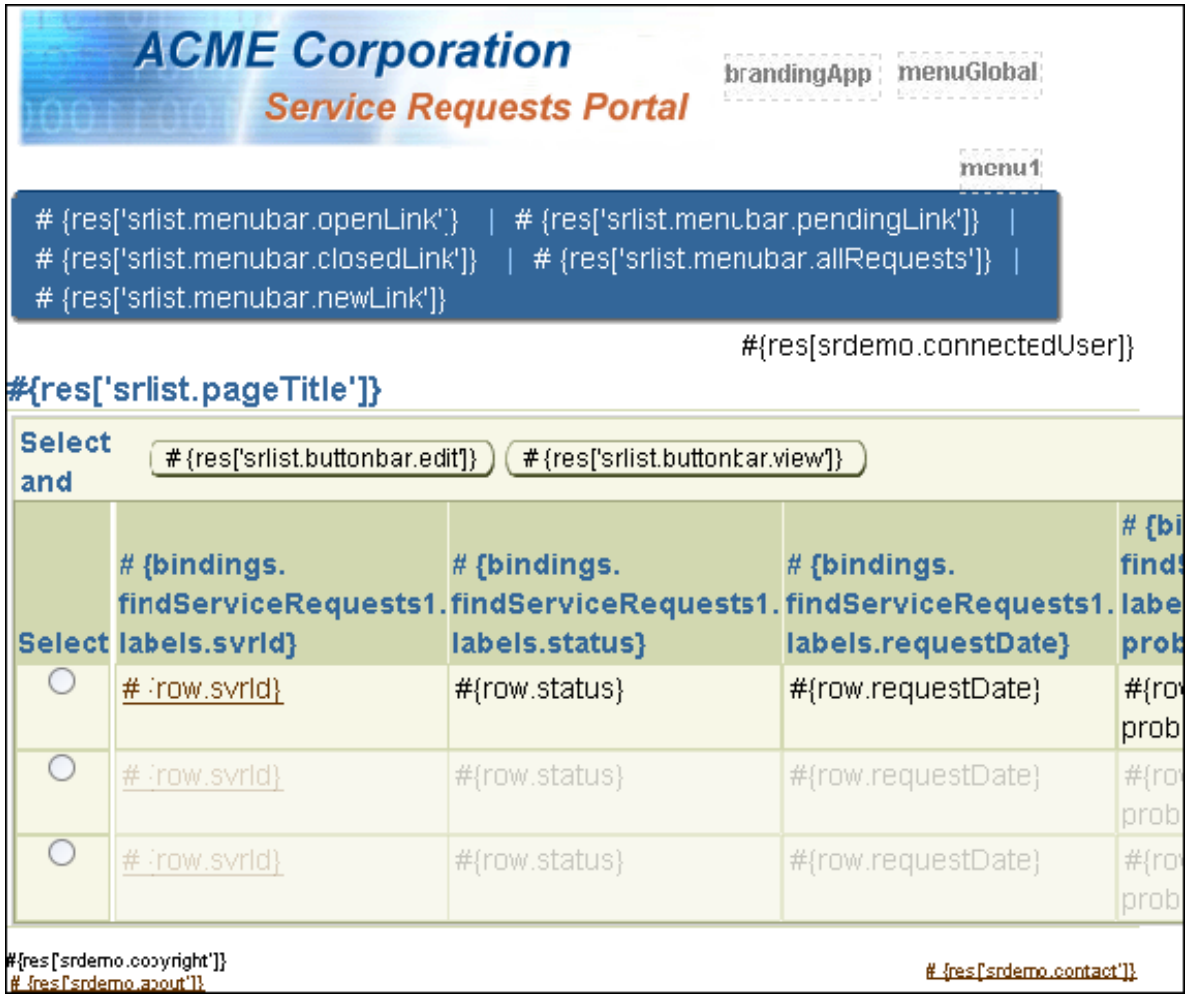
   ここで、`row` は現在選択されている行の索引です。したがって、その行のキーを取得できます。その結果、このキー値は `setCurrentRowWithKey` メソッドに渡されます。したがって、ユーザーがリンクをクリックして選択するキー値は現在の行の設定に使用されます。これによって、この行は、ユーザーが SRMain ページへのリンクによって移動させられるときに表示される行になります。

4. コマンド・リンクの Text プロパティを **`#{row.svrId}`** に設定し、Action プロパティをこの章の前で作成した **`drillDown_action()`** メソッドを使用するよう設定します。

5. 構造ウィンドウで、ラベルが #{row.svrId} の元の **`af:outputText`** を削除します。

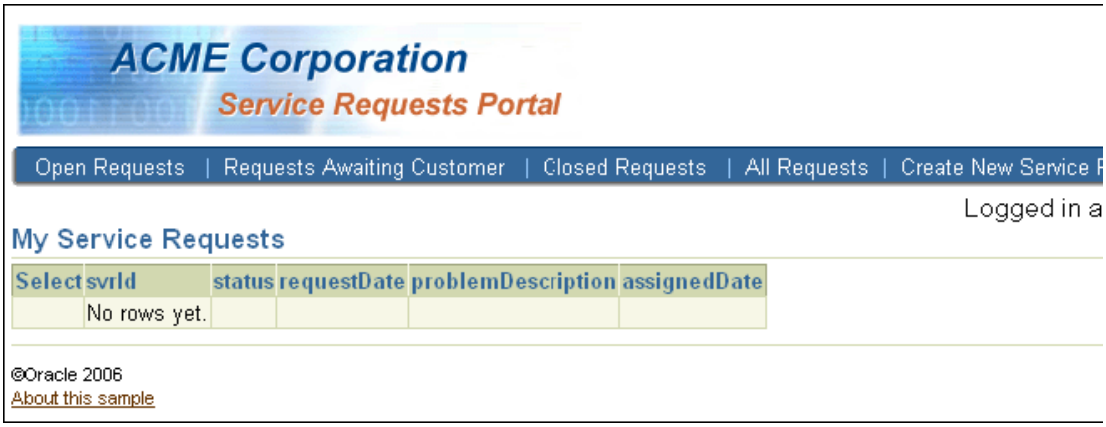
6. ページを保存します。

   メニュー・バーとドリルダウン・リンクが適切な位置にある場合、ページは、次のスクリーンショットのように表示されます。



ページの実行

ページが完成したので、ページを実行し、どのように表示されるかを確認できます。ビジュアル・エディタで SRList ページを開いた状態で、右クリックして**「実行」**を選択します。ページが表示されると、次のスクリーンショットのように表示されます。



最初、データは表示されません。これは、SRList ページを直接実行しているためです。ページが全体のアプリケーションの一部として実行される場合、ログインしたユーザーに適切なサービス・リクエストを表示できるよう、ユーザー ID はログインからページに渡されます。ログイン機能は、後述の章で作成します。

ページに追加した様々な UI コンポーネントに注目してください。値は `UIResources` ファイルから取得されています。

定義したメニュー・タブは、ページの上部にあるメニュー・バーに表示されます。選択したリクエスト・タブにハイライトされたテキストを追加しました。オープンがデフォルトのリクエストのステータスなので、「Open Requests」タブが太字です。

「View」および「Edit」ボタンは表示されていません。これらは表にデータが表示されると表示されます。

まとめ

この章では、SRDemo アプリケーションのユーザーがサービス・リクエストの情報を表示できる表示ページを作成しました。これを実現するために、次の主なタスクを実行しました。

- 第 4 章で定義したテンプレート・ページを基にしたアウトライン・ページの作成
- サービス・リクエスト情報を表示するページへのユーザー・インタフェース・コンポーネントの追加
- アプリケーション内のその他のページに移動する「View」および「Edit」ボタンの追加
- ユーザーが別のページから戻るときにサービス・リクエストを表示できるリフレッシュ動作の指定
- ユーザーがステータスによってサービス・リクエストを選択できるメニューの追加
- ユーザーが表内の行を選択し、SRMain ページに移動できる、ドリルダウン機能の作成

6

ログイン・セキュリティの実装

この章では、SRDemo アプリケーションのセキュリティを作成する方法について説明します。まず認証を追加し、デフォルト・ユーザーとしてログインし、ページのデータを参照できるようにします。次に、他のユーザーが自身のユーザー ID とパスワードでアプリケーションにアクセスできるようにします。

この章の内容は次のとおりです。

- 概要
- セキュリティを制御するクラスの作成
- ロールを管理するクラスの作成
- 認証を与えるクラスの作成
- ユーザーのアプリケーションへの統合
- コンテナ・セキュリティの設定
- アプリケーションへのアクセスの設定
- まとめ

概要

SRDemo アプリケーションが実行される環境で認証および認可を実装する必要があります。認証はアプリケーションにアクセスするユーザーを決定します。これは各ユーザーに割り当てられるロールによって制御されます。認可により、ユーザーがアプリケーションに入った後に実行することを許可される動作のタイプを制御します。

SRDemo のセキュリティは、J2EE コンテナ・セキュリティに基づいています。アプリケーションで使用可能なロールは、user、technician および manager （すべて小文字）です。

セキュリティ・コンテナでは、`remoteUser` 値（コンテナ `userid`）は、アプリケーションのユーザー表の電子メール・パスワードと一致します。主要なセキュリティ・アーチファクトは `UserInfo` Bean です。このクラスは、最初に参照されたときにコンテナ・セキュリティ属性を読み取り、式言語を使用してアクセスできる形式で、主要な情報を入手できるようにします。

開発のため、ユーザー名およびロールのテスト値を `faces-config.xml` ファイルのマネージド・プロパティを使用して `userInfo` オブジェクトに入れることができます。これらの設定は、セキュリティが有効になっているコンテナにアプリケーションがデプロイされている場合は無視されます。この場合、コンテナ・セキュリティ情報が戻されます。

この章では、次のようなアプリケーション用の認証および認可を作成します。

- ログインおよびユーザーのロールを管理するクラス
- ユーザーに認証を与えるクラス
- ユーザーを電子メール ID によって検索する名前付き問合せ
- アプリケーションへのアクセスが許可されているユーザーの登録

注意： 第 5 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **`Chapter6`** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、**`<jdev_install>¥jdev¥mywork¥Chapter6`** になります。
2. **`<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter5.zip`** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「`<jdev_install>¥jdev¥mywork¥Chapter6¥SRDemo¥SRDemo.jws`」**を選択します。これによって、この章の初期アプリケーションが開きます。

第 5 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

セキュリティを制御するクラスの作成

SRDemo アプリケーションでは、誰がどんなタイプの機能を実行しているかを判断するのに、3 つのコア・ロールを使用します。各ユーザーは、user、technician または manager の 3 つのロールのうちの 1 つで分類されている必要があります。すべてのユーザーのデフォルトのパスワードは `welcome` です。これらの基準はすべて、Oracle Application Server 10*g* またはその他のアプリケーション・サーバーで提供されるコンテナ管理の BASIC 認証を使用して実装されます。

`UserInfo` Bean は、#{userInfo.userName} などの式をサポートする `userInfo` というマネージド Bean として登録されます。ログイン ID または「`Not Authenticated`」という文字列を戻します。次の表では、`UserInfo` クラスで使用される式とその値を示します。

式	値
`#{userInfo.userRole}`	現在のユーザーのロールを文字列で戻します（たとえば、manager）。
`#{userInfo.staff}`	ユーザーが技術者またはマネージャの場合は true を戻します。
`#{userInfo.customer}`	ユーザーに user ロールが割り当てられている場合は true を戻します。
`#{userInfo.technician}`	ユーザーに technician ロールが割り当てられている場合は true を戻します。
`#{userInfo.manager}`	ユーザーに manager ロールが割り当てられている場合は true を戻します。

最初のタスク・セットにより、必要なコンテナが作成され、テスト・ユーザーおよびロールが移入されます。この方法で、アプリケーションをテストできます。2 番目のタスク・セットでは、アプリケーション・サーバーでユーザーとロールを定義し、ユーザーが自身のユーザー ID でログインできるようにします。

ロールを管理するクラスの作成

最初のタスクは、ユーザーの検証に必要なコードを含むクラスの作成、および使用可能なロールの判断です。

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。**「アプリケーション・ソース」**ノードを右クリックして、**「新規」**を選択します。
2. 新規ギャラリで、「General」カテゴリから**「Java クラス」**を選択します。
3. 次の表を使用して、Java クラスの作成ペインで値を移入します。

フィールド	値
名前	`UserInfo`
パッケージ	`oracle.srdemo.view`
拡張対象	`Java.lang.Object`
Public（属性）	チェック・ボックスを選択
デフォルトのコンストラクタを生成	チェック・ボックスを選択
main メソッドの生成	チェック・ボックスの選択を解除

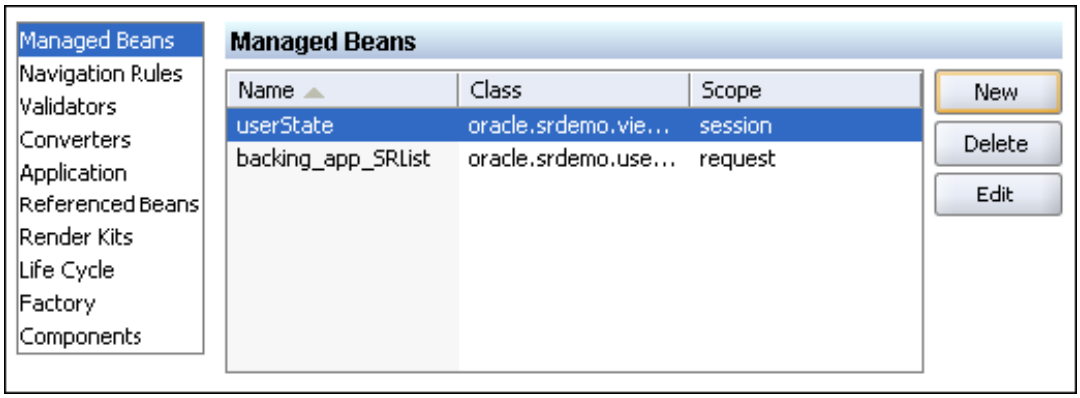
4. 基本クラスが作成され、コーディング用に準備されます。時間を節約するには、**`UserInfo.java`** ファイル（`<tutorial_install>¥files¥` ディレクトリまたは ADFTutorialSetup ファイルを解凍した場所にあります）を開き、そのコードをコピーします。新しく作成したクラスにそれを貼り付け、生成されたコードを置き換えます。

5. 「**保存**」をクリックしてファイルを保存します。

認証を与えるクラスの作成

次のタスクは、マネージド Bean の作成および移入です。この項では、`userInfo` マネージド Bean を作成します。これにより、JSF ページで `UserInfo.java` クラスにアクセスできます。

1. `faces-config.xml` ファイルがまだ開いていない場合、「**UserInterface**」を右クリックし、「**JSF ナビゲーションを開く**」を選択します。
2. 「**概要**」タブをクリックし、すべてのマネージド Bean のリストを公開します。作成した各ページに対するマネージド Bean が表示されます。
3. 「**Managed Bean**」カテゴリが選択された状態で、「**新規**」をクリックします。



4. 「マネージド Bean の作成」ペインで、プロパティ値を次の表の値に設定します。

フィールド	値
名前	`userInfo`
クラス	`oracle.srdemo.view.UserInfo` 値を入力するか、省略記号（...）をクリックして「階層」タブを開き、値を選択することができます。
有効範囲	`session`
クラスが存在しない場合は生成	チェック・ボックスを選択

5. 「**OK**」をクリックして続行します。
6. 構造ウィンドウで、新しい **userInfo** マネージド Bean を右クリックします。「**マネージド Bean の中に挿入**」→「**管理プロパティ**」を選択します。このステップにより、テスト中にデフォルトのユーザー名でログインできます。次の情報を使用して、必要な値を移入します。

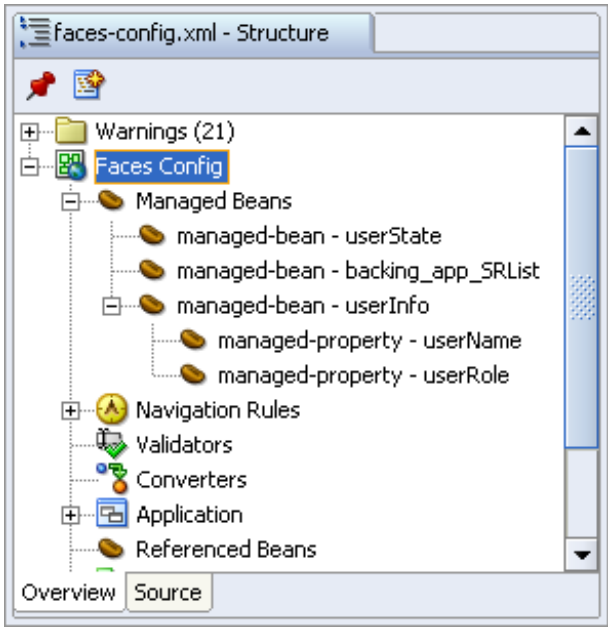
フィールド	値
名前	`userName`
クラス	`java.lang.String` 値を入力するか、省略記号（...）をクリックして「階層」タブを開き、値を選択することができます。

7. 「**OK**」をクリックして続行します。

8. **userInfo** マネージド Bean の中に別の管理プロパティを作成します。次の情報を使用して、必要な値を移入します。

フィールド	値
名前	`userRole`
クラス	`java.lang.String` 値を入力するか、省略記号（...）をクリックして「階層」タブを開き、値を選択することができます。

9. 「**OK**」をクリックして続行します。構造ウィンドウは、次のスクリーンショットのように表示されます。



10. テストのため、**userName** 管理プロパティをダブルクリックして Value プロパティを **`sking`** に設定します。「**OK**」をクリックします。
11. テストのため、**userRole** 管理プロパティをダブルクリックして Value プロパティを **`manager`** に設定します。「**OK**」をクリックします。

12. 「ソース」タブをクリックして、新しいコードを調べます。管理プロパティには指定した値が移入されます。

```
<managed-bean>
  <managed-bean-name>userInfo</managed-bean-name>
  <managed-bean-class>oracle.srdemo.view.UserInfo</managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
  <managed-property>
    <property-name>userName</property-name>
    <property-class>java.lang.String</property-class>
    <valuc>sking</valuc>
  </managed-property>
  <managed-property>
    <property-name>userRole</property-name>
    <property-class>java.lang.String</property-class>
    <value>manager</value>
  </managed-property>
</managed-bean>
```

ユーザーを追加したので、ページを実行し、データを移入することができます。テストのためにはこれで十分です。ただし、最終的にはアプリケーション用にセキュリティを提供する必要があります。

ユーザーのアプリケーションへの統合

アプリケーションをデプロイする際、ユーザーが自身の情報を提供することができ、その情報が記録されるようにします。この場合、アプリケーションは、どんなメニューや機能を提供するか決定できるよう、誰がログインしたかを認識する必要があります。ユーザーの電子メール ID は、セキュリティ・コンテナからアプリケーションに渡され、ユーザーが利用できるアクセス・タイプ（たとえば、SRList および SRCreate ページへのアクセス）を制御します。

ユーザーを管理する名前付き問合せの作成

名前付き問合せを使用してアプリケーションにログインしているユーザーを識別できます。問合せは読取り専用で、電子メール ID を含む String パラメータをとります。この問合せは、コンテナ・セキュリティから受け取った特定の電子メール ID のユーザー・オブジェクトを戻します。`DataModel` → `TopLink` → `SRMap` ファイルの User ディスクリプタに名前付き問合せを作成します。

1. User ディスクリプタに名前付き問合せを作成します。問合せは電子メール ID を基にし、セキュリティ・コンテナからその値を受け取ります。次の表の値を使用します。（問合せを作成する詳細な手順は、第 2 章を参照。）

フィールド	値
名前付き問合せの名前	`findUserByEmail`
タイプ	`ReadObjectQuery`
パラメータ型	`java.lang.String`
パラメータ名	`emailParam`

2. 名前付き問合せに対する式を作成します。次の表の値を使用して、定義を完成させます。

フィールド	値
第 1 引数問合せキー	email
演算子	EQUAL
第 2 引数 - パラメータ	emailParam

名前付き問合せの宣言的な作成が終了しました。ここで、セッション Bean のメソッドを作成する必要があります。メソッドを含めて電子メール ID を取得します。

3. 名前付き問合せをメソッドに変換し、値をアプリケーションに公開します。**「DataModel」→「アプリケーション・ソース」→「oracle.srdemo.datamodel」**ノードを開きます。

4. EJB セッション Bean を右クリックし、**「セッション・ファサードの編集」**を選択して、新しい問合せを Bean に追加します。**「User」**ノードを開き、**public User findUserByEmail (String emailParam)** の前のチェック・ボックスを選択します。**「OK」**をクリックします。

5. **「すべて保存」**をクリックして作業内容を保存します。

6. アプリケーション・ナビゲータで、**「SRPublicFacadeBean.java」**ノードを選択して、コンテキスト・メニューから**「データ・コントロールの作成」**を選択します。

これにより、現在の EJB に処理が転送され、既存のデータ・コントロールが更新されます。データ・コントロール・パレットでは、新しく追加されたメソッドにコントロールが含まれるよう更新されます。

ユーザーの電子メール ID のアプリケーションへの公開

ここでは、TopLink Java オブジェクトを外部に公開するメソッドを作成します。値が公開された後は、アプリケーション内のどこででも使用できます。XML ファイルを作成し、すべての JSF ページからユーザー ID にアクセスできるようにします。

1. XML ファイルを作成してメソッドを含めます。**「UserInterface」→「アプリケーション・ソース」→「oracle.srdemo.userInterface」→「pageDef」**ノードを開き、右クリックして**「新規」**を選択します。

2. 「General」カテゴリから**「XML」**を選択し、「項目」ペインから**「XML 文書」**を選択します。

3. ポップアップ・メニューで、ファイルに **userInfo.xml** と名前を付け、ディレクトリにページ定義パス（...SRDemo¥UserInterface¥src¥oracle¥srdemo¥userInterface¥pageDefs）が含まれていることを確認します。

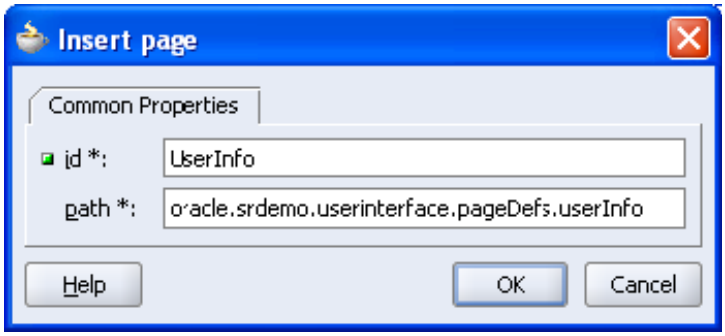
4. **userInfo.xml** ファイルにコードをコピーします。このファイルは <tutorial_install>¥files ディレクトリにあります。既存のコードをファイルのコードに置き換えます。UserInfo.xml ファイルを定義するコードは、他のページ定義パッケージとともに保存されており、SRPublicFacade セッション Bean の findUserByEmail メソッドを使用します。

5. JDeveloper を閉じて再度起動し、JDeveloper にページ定義として userInfo.xml を認識させます。

XML 定義とは別に、DataBindings.cpx から名前（UserInfo）でその定義にアクセスできる必要があります。DataBindings.cpx ファイルは、HTML ビジュアル・エディタのページ・フロー・ダイアグラムから Web ページを初めて開いたときに作成されます。.cpx ファイルは、アプリケーション全体に対する Oracle ADF バインディング・コンテキストを定義します。.cpx ファイルは、実行時に Oracle ADF バインディング・オブジェクトが作成される際に元となるメタデータを提供します。バインディング・コンテキストにより、アプリケーション全体のバインディングへのアクセスが提供されます。プロパティ・インスペクタでこのファイルを編集し、パラメータを追加または削除したり、バイ

ンディング・コンテナ設定を変更したりできます。これは、データ・バインディング・ファイルのある構造ウィンドウから実行できます。

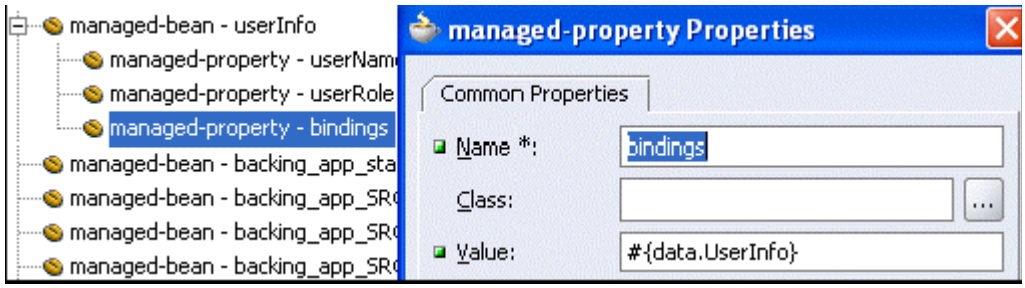
6. 「**UserInterface**」→「**アプリケーション・ソース**」→「**oracle.srdemo.userInteface**」ノードを開き、「**DataBinding.cpx**」ノードを選択します。
7. 構造ウィンドウで、「**pageDefintionUsages**」ノードを選択して、「**pageDefinitionUsages の中に挿入**」→「**page**」を選択します。
8. 「挿入」ポップアップ・メニューで、ID を **`UserInfo`** に設定し、パスを **`oracle.srdemo.userinterface.pageDefs.userInfo`** に設定します。「**OK**」をクリックして続行します。



実行時、この段階で `data.UserInfo` への参照はこのバインディング定義に解決され、ページは UserInfo クラスにアクセスできるようになります。

次の手順で、管理プロパティに値を割り当て、作成した `userInfo.xml` ファイルを指すようにします。これにより、マネージド Bean は `findUserByEmail` 名前付き問合せの戻りからの値を受け入れることができます。

9. ビジュアル・エディタで、**`faces-config.xml`** ファイルを開きます。「**概要**」サブタブをクリックし、**`userInfo`** マネージド Bean を選択します。**`bindings`** という新しい管理プロパティを作成します。バインディングにより、参照で正しい userInfo を見つけることができます。
10. 作成後、**`bindings`** プロパティをダブルクリックし、値プロパティを **`#{data.UserInfo}`** に設定します。次のスクリーンショットは、userInfo Bean のすべての管理プロパティ、および bindings プロパティの値を示します。



コンテナ・セキュリティの設定

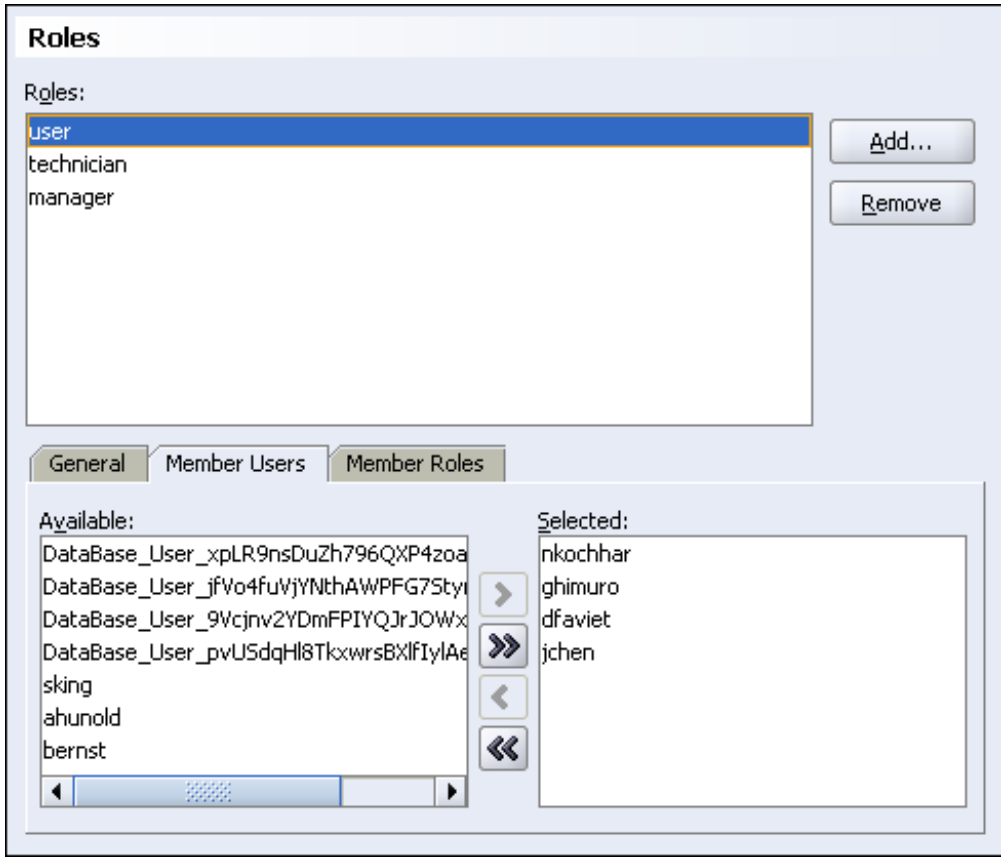
ユーザーが認証されている場合、J2EE コンテナの JAZN ベース・セキュリティ・システムに登録する必要があります。jazn-data.xml ファイルにロール、ユーザー名および資格証明を登録するには、次の手順を実行します。

1. 実行されている場合、**「実行」→「終了」→「埋込み OC4J サーバー」**を選択して OC4J サーバーを停止します。
2. **「ツール」→「埋込み OC4J サーバーの設定」**に移動します。
3. **「現在のワークスペース (SRDemo)」→「認証 (JAZN)」→「レルム」**の順にドリルダウンします。
4. jazn.com レルムが存在しない場合、**「新規」**ボタンをクリックして、新しいレルムに **jazn.com** と名前を付けます。

   レルムは、セキュリティの一般的なコンポーネントです。後で追加される SRDemo コンポーネントは、レルムを使用する方法です。この方法では、多くの異なるアプリケーションに対して同じレルムを使用できます。
5. **「ロール」**ノードを選択して、**「追加」**ボタンをクリックします。
6. **user**、**technician** および **manager** の 3 つのロールを作成します。
7. 次の表の値でユーザーを作成します。全体に資格証明に **welcome** を使用し、ユーザー名はすべて小文字にします。ロールを選択して**「メンバー・ユーザー」**タブをクリックして、ロールにユーザー名を割り当てます。各ユーザーを対応するロールに割り当てるため、「選択済」側にユーザーを移動します。

ユーザー名	ロール
sking	manager
nkochhar	user
ghimuro	user
ahunold	technician
bernst	technician
dfaviet	user
jchen	user

次のスクリーンショットは、user ロールに割り当てられたメンバーを示しています。

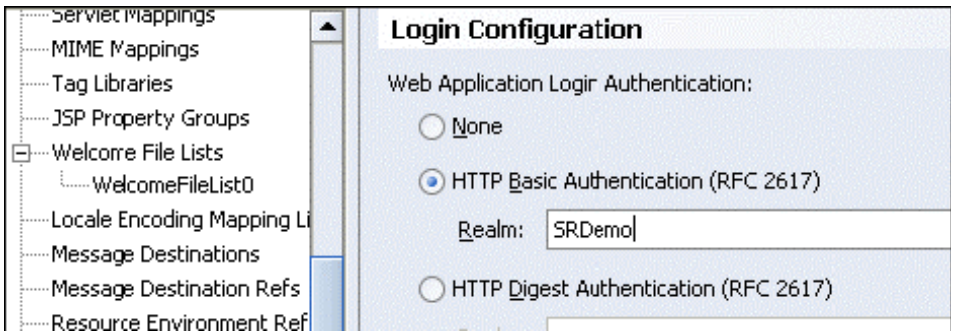


入力したすべてのデータは、アプリケーションのルート・ディレクトリにあるファイルに保存されます。ファイルの名前付けに使用される規則は、`<applicationName>-jazn-data.xml` です。このチュートリアルでは、ファイル名は `SRDemo-jazn-data.xml` です。

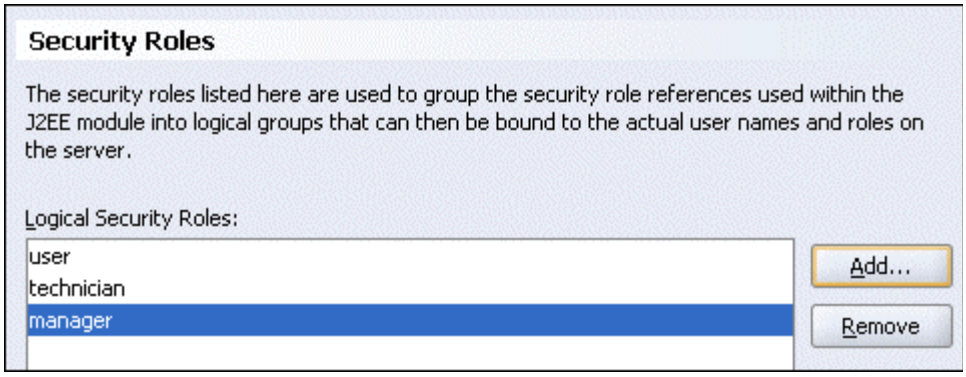
アプリケーションへのアクセスの設定

この項では、どのロールがアプリケーション内のどのディレクトリ構造にアクセスできるかを定義します。

1. アプリケーション・ナビゲータで、**「UserInterface」→「Web コンテンツ」→「WEB-INF」**ノードを開き、**web.xml** を選択します。コンテキスト・メニューから**「プロパティ」**を選択します。
2. 左のペインを下へスクロールして、**「ログイン構成」**ノードを選択します。**「HTTP Basic 認証」**オプションを選択し、「レルム」プロパティを **SRDemo** に設定します。



3. **「セキュリティ・ロール」**ノードを選択し、前述した **user**、**technician** および **manager** の 3 つのロールを追加します。（追加する必要がある各ロールについて「追加」をクリックします。）



セキュリティ・ロールを定義したら、セキュリティ制約に割り当て、認可を行う必要があります。

4. **「セキュリティ制約」**ノードを選択し、**「新規」**をクリックします。「Web リソース」タブで、**「追加」**をクリックして Web リソース名として **AllManagers** を指定します。**「OK」**をクリックします。

5. **AllManagers** コレクションが選択された状態で、「URL パターン」タブの右下のペインで **「追加」**ボタンをクリックします。「URL パターンの作成」ポップアップ・ウィンドウで、**faces/app/management/*** と入力します。これにより、すべてのマネージャが管理ディレクトリのファイルにアクセスできます。

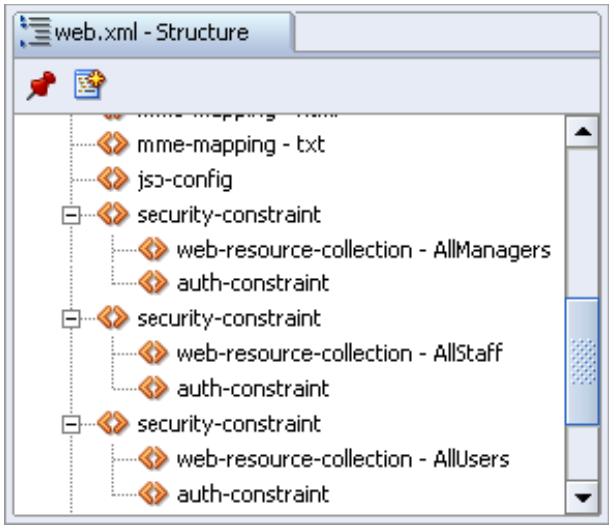
6. **AllManagers** コレクションが選択された状態で、**「認可」**タブをクリックし、**manager** チェック・ボックスを選択します。

7. 次の表の値を使用してセキュリティ制約をあと 2 つ作成します。

Web リソース・コレクション	URL パターン	認可
AllStaff	`faces/app/staff/*`	`technician` および `manager`
AllUsers	`faces/app/*`	`user`、`technician` および `manager`

8. 「**OK**」をクリックし、「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・ボックスを閉じます。

   このステップを終了すると、3 つの制約のエントリが構造ウィンドウの「セキュリティ制約」ノードの下に表示されます。各制約のエントリは、前述の表のコレクションの 1 つと対応します。`web.xml` ファイルがエディタで開いた状態では、次のスクリーンショットのように構造ウィンドウにすべての制約が表示されます。



9. ビジュアル・エディタで SRList ページを開いた状態で、右クリックして「**実行**」を選択します。入力が求められたら、ユーザー名に **`bernst`** と入力し、パスワードに **`welcome`** と入力します。リストにサービス・リクエストが表示されます。オープン、ペンディングおよびクローズド・リクエストについて、いくつかのメニュー・オプションを選択します。

まとめ

この章では、SRDemo アプリケーションにセキュリティを指定しました。これを実現するために、次の主なタスクを実行しました。

- セキュリティを制御するコンテナの作成
- ロールを管理するクラスの作成
- 認証を与えるクラスの作成
- ユーザーのアプリケーションへの統合
- ユーザーの電子メール ID のアプリケーションへの公開
- コンテナ・セキュリティの設定
- アプリケーションへのアクセスの設定

7

検索ページの開発

この章では、JavaServer Faces および ADF コンポーネントを使用して、検索ページを作成する方法について説明します。ページには、問合せ基準の指定および結果の表示という 2 つのセクションがあります。ユーザーが、レコードの選択およびレコードの表示または編集を行うことができるボタンを作成します。

この章の内容は次のとおりです。

- 概要
- 検索ページの作成
- 検索ページへのデータ・コンポーネントの追加
- 「Edit」ボタンの組込み
- 「View」ボタンの組込み
- まとめ

概要

SRSearch ページには、技術者およびマネージャがサービス・リクエストの全リストを検索する QBE の画面が表示されます。ページは、ページ上部の問合せ領域（常に問合せモード）および表形式の結果領域（最新の検索結果を表示）の 2 つの領域に分かれています。

注意： 第 6 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. `Chapter7` というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、`<jdev_install>¥jdev¥mywork¥Chapter7` になります。
2. `<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter6.zip` をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、「`<jdev_install>¥jdev¥mywork¥Chapter7¥SRDemo¥SRDemo.jws`」を選択します。この章の初期アプリケーションが開きます。

第 6 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

ページ・フロー・ダイアグラムを使用して、検索ページをアプリケーションに構成する方法を表示します。ここでは、ページに関して次の点に注意します。

- 戻されたレコードは、問合せ領域と同じページに表示されます。
- ページは常に問合せモードです。
- 戻されたレコードの選択と表示または編集を行うことができます。
- マネージャのみすべてのレコードを表示できます。マネージャ以外は自分のレコードのみ表示できます。

検索ページの作成

検索ページを作成するには、最初に、ADF コンポーネントを使用して構造を作成し、データ・モデルからデータ・コンポーネントを追加します。

注意： チュートリアルの先の手順において、ページ・フロー・ダイアグラムでページのアウトラインを作成しました。ここで、ページを完了し、第 4 章で作成したテンプレートを適用します。JSF JSP 項目を使用して、新規ギャラリからページを作成することもできます。

次の手順を実行して、SRSearch ページを作成し、先に作成したテンプレートを使用します。

1. 開いていない場合は、**「faces-config.xml」**ファイルをダブルクリックして、ページ・フロー・ダイアグラムを表示します。
2. **「SRSearch」**ページをダブルクリックして、JSF ページ・ウィザードを起動します。

3. 次の値を使用して、ウィザードを完了します。

ウィザード・ステップ 1: JSP ファイル	
フィールド	**値**
ファイル名	SRSearch.jspx
ディレクトリ名	ファイルが保存される場所。**¥app¥staff** をデフォルト値に付加して、各サブディレクトリにファイルを置きます。ディレクトリは、<jdev_install>¥jdev¥mywork¥SRDemo¥UserInterface¥public_html¥app¥staff になります。
タイプ	JSP Document
モバイル	チェック・ボックスの選択を解除。

4. 「次へ」をクリックして続行します。

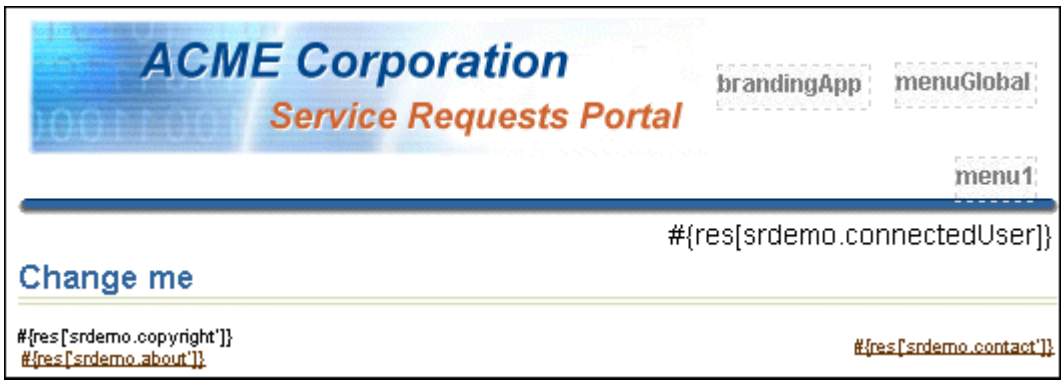
ウィザード・ステップ 2: コンポーネント・バインディング	
フィールド	**値**
新規マネージド Bean での UI コンポーネントの自動公開	このオプションが選択されていることを確認します。
名前	backing_app_staff_SRSearch
クラス	SRSearch
パッケージ	oracle.srdemo.userinterface.backing.app.staff

5. 「次へ」をクリックして続行します。

ウィザード・ステップ 3: タグ・ライブラリ	
フィールド	**値**
選択済のライブラリ	ADF Faces Components ADF Faces HTML JSF Core JSF HTML

6. **「終了」**をクリックして、ページ詳細を作成します。ビジュアル・エディタに、新規の SRSearch ページが表示されます。
7. まだ開かれていない場合は、**SRDemoTemplate** ファイルを開きます。構造ウィンドウで、**「afh:html」**ノードを右クリックして、ショートカット・メニューから**「コピー」**を選択します。
8. タブをクリックして、SRSearch ページに戻ります。構造ウィンドウで、**「f:view」**ノードを開きます。
9. **「html」**ノードを削除します。**「f:view」**を右クリックして、ショートカット・メニューから**「貼付け」**を選択します。
10. メイン・イメージが表示されない場合は、パスを変更する必要があります。イメージを選択し、プロパティ・インスペクタで URL プロパティを **/images/SRBranding** にリセットします。
11. **「afh:head」**をダブルクリックして、Title プロパティを **SRDemo Search** に設定します。このプロパティの値がブラウザのタイトルになります。

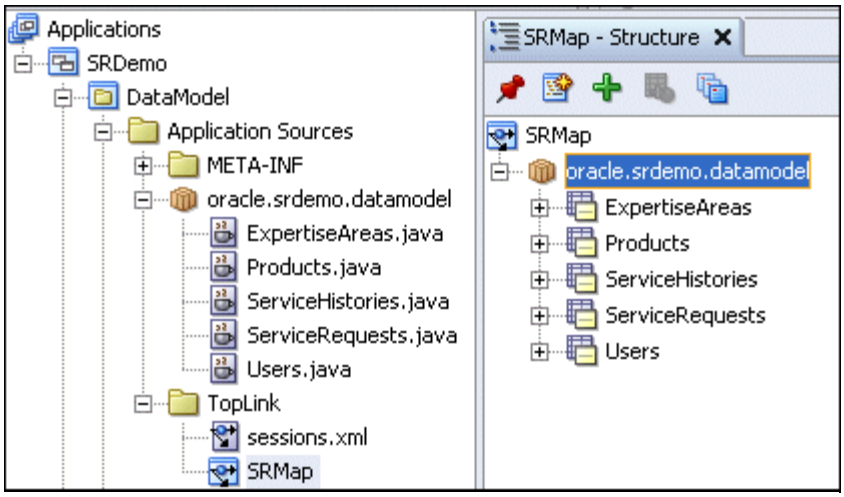
12. ビジュアル・エディタにより、他のページのルック・アンド・フィールで、SRSearch ページが次のように表示されます。



名前付き問合せの作成

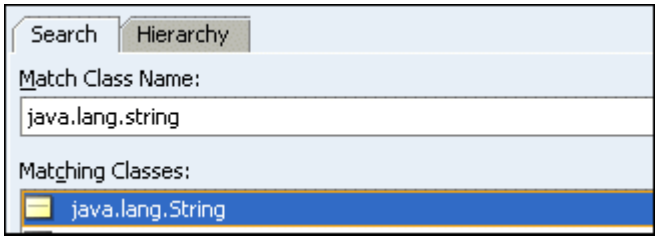
検索フォームでは、ServiceRequest に基づいた名前付き問合せを使用して、2 つのパラメータを受け取ります。

1. アプリケーション・ナビゲータで、DataModel プロジェクトの「**SRMap**」ノードを選択します。SRMap の構造が、構造ウィンドウに表示されます。



2. 「**ServiceRequests**」ノードをダブルクリックして、「**問合せ**」タブをクリックします。「**追加**」ボタンをクリックして、新規の名前付き問合せを作成します。
3. 「TopLink 名前付き問合せの追加」ペインで、新規の問合せの名前に、**`searchServiceRequests`** と入力します。「**OK**」をクリックして続行します。
4. 名前付き問合せ「`searchServicesRequests`」が選択された状態で、「**一般**」タブをクリックします。エディタの「パラメータ」領域で、「**追加**」ボタンをクリックします。
5. クラス・ブラウザ・ウィンドウで、「**検索**」タブをクリックします。ここでパラメータ・タイプを定義します。

6. 「クラス名を一致」プロパティに、**`java.lang.String`** と入力します。入力すると、リストがさらに絞り込まれます。一致するクラスの java.lang.String がハイライトされたら、**「OK」**をクリックしてタイプを定義します。

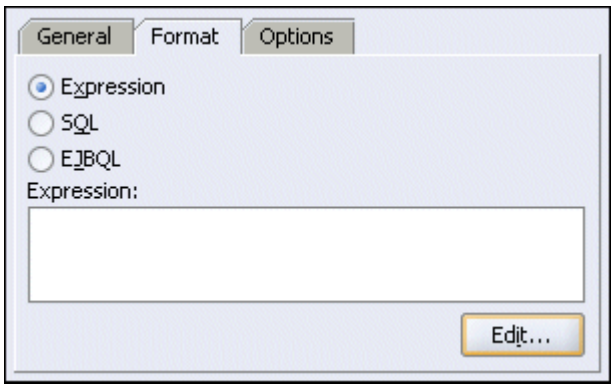


7. 「パラメータ」領域で、パラメータ名を **`descr`** に変更します。
8. `java.lang.String` の 2 つ目のパラメータを作成し、**`status`** という名前にします。

問合せ式の作成

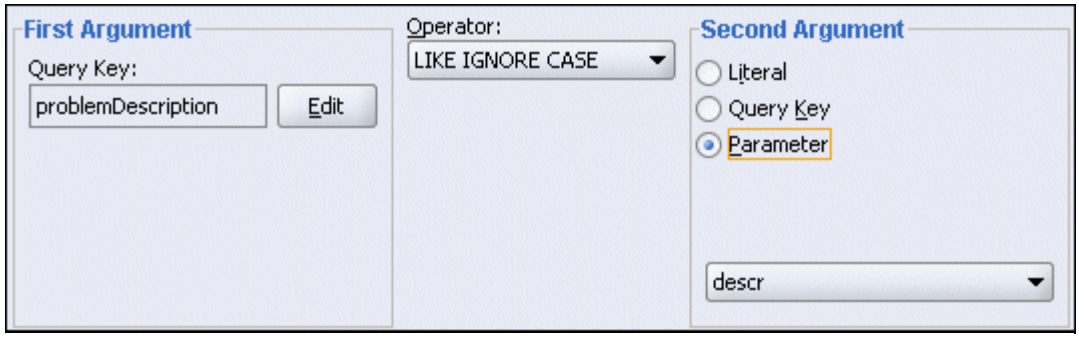
2 つのパラメータ（`descr` および `status`）を定義したので、パラメータを使用する式を定義する必要があります。この式では、名前付き問合せで、どのようにパラメータが属性に関連付けられるかを定義します。式は実行時に評価され、名前付き問合せによって戻される行を決定します。

1. 名前付き問合せ**「searchServiceRequests」**が選択された状態で、**「書式」**タブをクリックして、エディタの「パラメータ」領域で**「編集」**ボタンをクリックします。



2. 式ビルダーで、**「追加」**をクリックして、新しい式を作成します。
3. 式の「第 1 引数」領域で、**「編集」**をクリックします。「問合せキーの選択」ペインで、**「`problemDescription`」**を選択します。
4. **「OK」**をクリックして続行します。

5. 式ビルダーに戻り、演算子を **LIKE IGNORE CASE** に設定し、第 2 引数を先に作成した **descr** パラメータに設定します。次のスクリーンショットのように表示されます。**「OK」**をクリックして続行します。

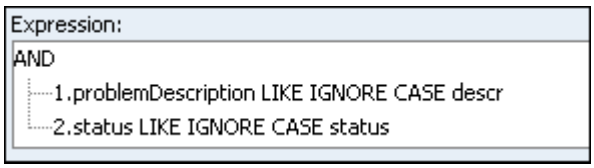


6. 次に、2 つ目の式を追加します。**「追加」**ボタンをクリックして、新しい式を追加します。式のコンポーネントを次の表の値に設定します。

フィールド	第 1 引数	演算子	第 2 引数（パラメータ）
第 2 のコンポーネント	**status**	**LIKE IGNORE CASE**	**status**

完了すると、式は次のように表示されます。結果を確認して、必要に応じて変更を行い、完了したら**「OK」**をクリックします。

7. 作業内容を保存します。



DataControl の再作成

この名前付き問合せの作成の最後のステップとして、これをデータ・コントロールに追加します。コードをセッション Bean メソッドの一部に追加したため、データ・コントロールを再作成する際に、上書きしないことを確認する必要があります。

1. アプリケーション・ナビゲータで、**「SRPublicFacadeBean.java」**を右クリックして、コンテキスト・メニューから**「セッション・ファサードの編集」**を選択します。
2. **「ServiceRequests」**を開いて、新規の名前付き問合せ**「searchServiceRequests」**が選択されていることを確認します。

3. **「OK」**をクリックします。

4. アプリケーション・ナビゲータで、**「SRPublicFacadeBean.java」**を右クリックして、コンテキスト・メニューから**「データ・コントロールの作成」**を選択します。
5. 「データ・コントロール」ペインに、`searchServiceRequests(String, String)` が表示されます。

検索ページへのデータ・コンポーネントの追加

この項では、SRSearch ページを変更して、別のタイトルの表示およびデータ・バインドされたコントロールの組込みを行います。タイトルおよびプロンプトを定義する場合、多くはページにハードコーディングするのではなく、データモデルに定義された値を参照します。その結果、モデルのプロンプトまたはタイトルを変更する場合、更新された値が実行時に使用されます。

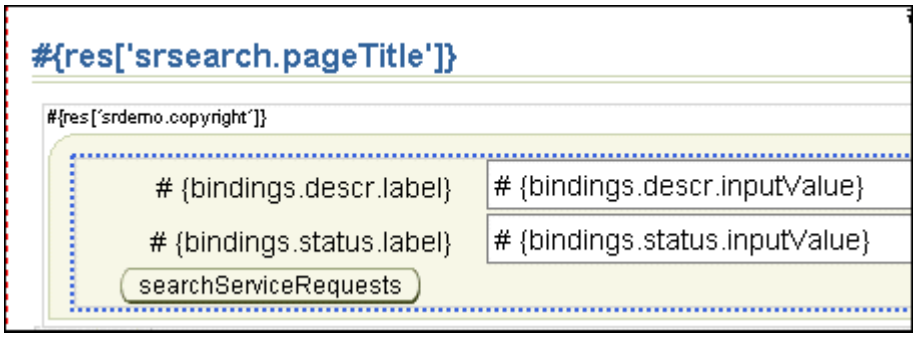
1. **「SRSearch.jspx」**タブをクリックし、構造ウィンドウで、**「f:view」**→**「afh.html」**→**「afh.body」**→**「h:form」**ノードを開いて、`af:panelPage` を表示します。
2. **「`af:panelPage`」**をダブルクリックして、プロパティの Title フィールドを **`#{res['srsearch.pageTitle']}`** に設定します。

   この値はテンプレートに設定されるので、テンプレートを最初に使用する場合またはテンプレートを使用する各ページで `res` 変数を参照できます。このプロパティをタイトル用の文字テキストに設定できます。プロパティを、複数のプロパティと値が組になったリストを含む、`UIResources.properties` ファイルの値に設定します。実行時に、このファイルが読み込まれ、ページ上で値が置き換えられます。チュートリアル全体で、ボタン名、ページ・タイトルおよびフィールド・プロンプトに対してこの規則を使用します。実行時に、値は「Find a Service Request」になります。
3. エディタで**「ソース」**タブをクリックすると、コードが表示されます。次のスクリーンショットは、この手順のソース・コードを示しています。

```
<af:panelPage title="#{res['srsearch.pageTitle']}"
              binding="#{backing_app_staff_SRSearch.panelPage1}"
              id="panelPage1">
```

4. **「設計」**タブをクリックして、変更内容を表示します。
5. コンポーネント・パレットで、ADF Faces Core ページを開いて、構造ウィンドウで、**PanelBox** を **`af:panelPage`** にドラッグします。この操作により、ページの問合せコンポーネント用のプレースホルダが作成されます。

   名前付き問合せ searchServiceRequests を使用するパラメータ・フォームを追加します。「データ・コントロール」タブで、**「`SRPublicFacadeLocal`」**コントロールを開いて、**「searchServiceRequests(String, String)」**を検索します。
6. **「`searchServiceRequests(String, String)`」**を選択して、これを af:panelBox にドラッグします。コンテキスト・メニューから、**「パラメータ」**→**「ADF パラメータ・フォーム」**を選択します。「フォーム・フィールドの編集」ダイアログ・ボックスで**「OK」**をクリックして、デフォルトを受け入れます。この操作により、2 つのフィールドと検索ボタンのあるフォームが追加されます。



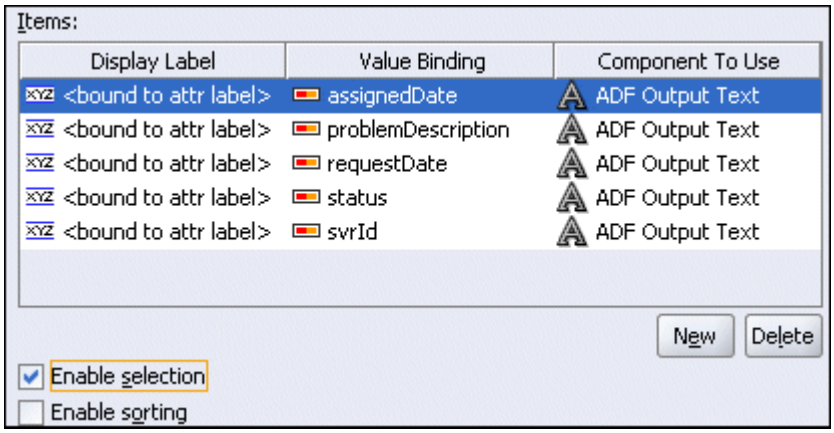
7. 「searchServiceRequests」ボタンの Text プロパティを変更して、リソース **`#{res['srsearch.searchLabel']}`** を使用します。
8. 説明の `af:inputText` コンポーネントの Label プロパティを**「Description:」**に変更します。
9. ステータスの `af:inputText` コンポーネントの Label プロパティを**「Status:」**に変更します。

データ・コンポーネントの追加

次の手順で、領域およびデータ・コンポーネントを作成し、問合せ結果を表示します。

1. データ・コントロール・パレットで、**「searchServiceRequests」**を開いて、サブノード**「ServiceRequests」**を **`af:panelPage`** にドラッグします。
2. ポップアップ・メニューで、**「表」→「ADF 読取り専用表」**を選択します。

   この表には検索結果が表示されます。表の列の編集ウィンドウが表示されます。ここで、表示用の表示ラベル、バインディング値およびコンポーネントのタイプを変更できます。デフォルト値はデータ・モデルで定義された値に基づきます。
3. **「選択を有効にする」**チェック・ボックスを選択して、デフォルトを受け入れます。**「OK」**をクリックして続行します。



4. 構造ウィンドウで、**「af:table」**ノードを開いて、表の列を **srvId**、**problemDescription**、**status**、**requestDate**、**assignedDate** の順に並べ替えます。

   構造ウィンドウでノードを選択すると、ビジュアル・エディタにより列が表示されるので、完全な列定義を表示できます。
5. 構造ウィンドウで、**「af:table」→「Table facets」→「selection」**を開いて、**「af:tableSelectOne」**ノードをダブルクリックします。ハードコーディングされたデフォルトの Text プロパティをリソース文字列 **`#{res['srsearch.resultsTable.prefix']}`** に変更します。この操作により、列のラベルは `UIResources.properties` ファイルに定義された結果に変更されます。
6. 「af:table」が選択されている状態で、構造ウィンドウで **`Id`** プロパティを **`srtable`** に変更します。
7. **「Submit」**ボタンを削除します。
8. ページを保存します。
9. コンポーネント・パレットの ADF Faces Core ドロップダウン・メニューから、**「ADF Faces Core」→「CommandButtons」**を `af:tableSelectOne` ノードに 2 回ドラッグします。
10. **「af:commandButton」**をそれぞれダブルクリックして、Text プロパティを次の表の値に変更します。

11. この操作により、各ボタンは実行する 2 つの機能（編集または表示）のいずれかに関連付けられます。次の表の値は両方とも UIResources.properties ファイルに定義されています。

コマンド・ボタン	値
1	#{res['srsearch.resultsTable.edit']}
2	#{res['srsearch.resultsTable.view']}

問合せのデフォルトの動作の変更

パラメータ・フォームのデフォルトの動作は、単にユーザーが入力する値を受け取り、問合せを実行することです。つまり、結果が戻されるには、問合せに対応する各フィールドに値またはワイルドカードを入力する必要があります。2 つのフィールドがあるフォームであればおそらく問題ありませんが、それ以上になるとユーザーにとって使いやすさの面で問題が発生する可能性があります。

この項では、一部の式言語（EL）を、ユーザーがフィールドに値を入力しない場合にワイルドカードを挿入するページ定義パラメータに追加します。このコードをパラメータごとに追加するので、ユーザーは一方または両方のパラメータを入力できます。

1. アプリケーション・ナビゲータで、**SRSearch** ページ用のページ定義を選択します。
2. 構造ウィンドウで、**「バインディング」**→**「searchServiceRequests」**を開きます。
3. **descr** プロパティをダブルクリックします。
4. 「NamedData のプロパティ」ダイアログ・ボックスの NDValue で、**「...」**ボタン（参照 / 編集）をクリックします。

   式は、実行用の問合せに渡される前にパラメータを移入するものです。デフォルトでは、パラメータは、パラメータ・フォームの関連するフィールドから移入されます。パラメータが NULL または空白の場合、追加しようとしているコードにより値がチェックされ、ワイルドカード（「%」）が戻されます。
5. descr 用のデフォルトの式を次のコードに置き換えます。

   ```
   ${((bindings.searchServiceRequests_descr == null) ||
   (bindings.searchServiceRequests_descr == '')) ?'%' :
   bindings.searchServiceRequests_descr }
   ```

   前述のコードは、NULL および空白（''）に関して descr をテストします。いずれかが true の場合、式はワイルドカード（「%」）を戻します。descr が NULL ではない場合、式はユーザーが入力した値を戻します。
6. **「OK」**をクリックして、変更内容を受け入れます。再度**「OK」**をクリックして、ダイアログ・ボックスを閉じます。
7. status プロパティをダブルクリックします。
8. 「NamedData のプロパティ」ダイアログ・ボックスの **NDValue** で、**「...」**ボタン（参照 / 編集）をクリックします。
9. status 用のデフォルトの式を次のコードに置き換えます。

   ```
   ${((bindings.searchServiceRequests_status == null) ||
   (bindings.searchServiceRequests_status == '')) ?'%' :
   bindings.searchServiceRequests_status }
   ```
10. ページの任意の場所で右クリックして、**「実行」**を選択します。入力が求められたら、ユーザー名に **sking** と入力し、パスワードに **welcome** と入力します。
11. いずれのフィールドにも値を入力しないで「search」ボタンをクリックします。結果には、サービス・リクエストがすべて表示されます。
12. status フィールドで **open** と入力して、「search」をクリックします。結果には、オープンのサービス・リクエストがすべて表示されます。

13. description フィールドに値を入力し（たとえば、"**%wash%**"）、「search」をクリックします。ステータスが**オープン**で、説明内に「wash」を含む行が表示されます。
14. **status** フィールドをクリアして、「search」をクリックします。説明内に「wash」を含む行がすべて表示されます。

リフレッシュ条件の追加

最初にページを実行したときには、ページのロード時に問合せが実行されました。コードを追加して NULL 値のかわりにワイルドカードを使用したので、最初の問合せでは、すべての行が戻されて表示されます。

最後の手順として、ユーザーが「search」ボタンをクリックするまで、問合せを実行しない状態にフォームを保持するリフレッシュ条件を追加します。

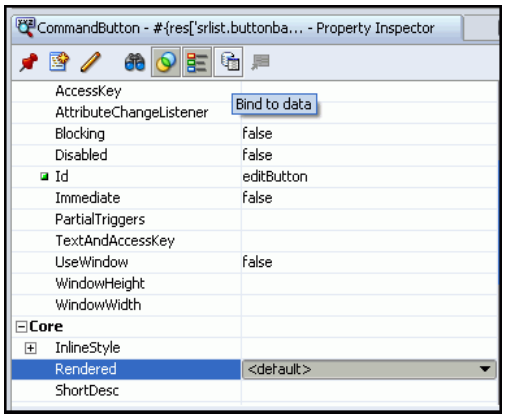
1. アプリケーション・ナビゲータで、**SRSearch** ページ用のページ定義を選択します。
2. 構造ウィンドウで、**「executables」**を開きます。
3. **「searchServiceRequestsIter」**をクリックします。
4. プロパティ・インスペクタで、RefreshCondition を **`${adfFacesContext.postback}`** に変更します。
5. ページの任意の場所で右クリックして、**「実行」**を選択します。
6. 「search」をクリックするまで、ディテール表には行が表示されないことに注意してください。

「Edit」ボタンの組込み

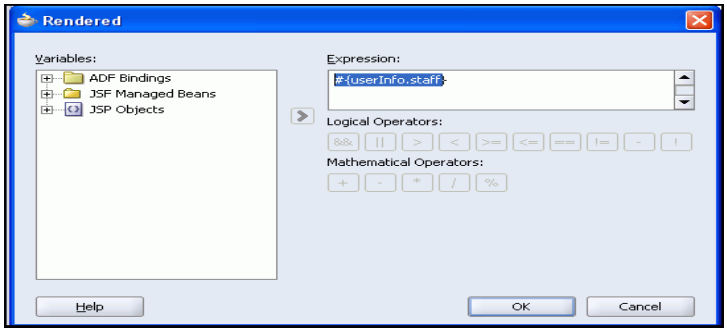
次の 2 つの手順は、SRList ページを作成した際の手順と同じです。「Edit」ボタンは、技術者またはマネージャのログイン時のみ使用可能です。ボタンをクリックすると、ユーザーは SREdit 画面に移動して、サービス・リクエストの詳細を変更できます。

次の手順を実行して、「Edit」ボタンの機能を指定します。

1. **「edit」**ボタンをクリックします。プロパティ・インスペクタで、Id プロパティを **editButton** に設定します。
2. 現在ログインしているユーザーがスタッフのメンバーである場合のみ、「Edit」ボタンを表示するよう指定します。**Rendered** プロパティを選択して、プロパティ・インスペクタのツールバーで**「データにバインド」**ボタン（右から 2 番目のボタン。ツールチップ・ラベルを使用して、必要なボタンを確認します）をクリックします。



3. Rendered ダイアログ・ボックスで、「式」フィールドに **#{userInfo.staff}** と入力します。**「OK」**をクリックします。これにより、ログインしているユーザーがスタッフのメンバーであることを確認します。



バッキング Bean コードの追加

現在選択済のサービス・リクエストの ID を編集ページに渡す、バッキング Bean のメソッドを作成します。これにより、該当するレコードが取得および表示できます。

1. ビジュアル・エディタで、**「Edit」**ボタンをダブルクリックして、バッキング Bean を設定するダイアログを起動します。
2. 「バインド Action プロパティ」ダイアログ・ボックスで、**「OK」**をクリックして、`editButton_action` メソッドをバッキング Bean に追加します。バッキング Bean ファイルで、`editButtonAction` メソッドに生成されたコードを次のコードに置き換えて、ナビゲーション・パスを指定します。

```
setCurrentSvrIdFromRow();
return "edit";
```

3. `setCurrentSvrIdFromRow` メソッドはまだ存在しません。**「CodeAssist」**アイコン（左マージンの電球）をクリックして作成します。
4. 次のコードを追加して、`setCurrentSvrIdFromRow` メソッドを実装します。

```
FacesContext ctx = FacesContext.getCurrentInstance();
JUCtrlValueBindingRef tableRowRef =
 (JUCtrlValueBindingRef)this.getSrtable().getRowData();

Integer svrId =
(Integer)tableRowRef.getRow().getAttribute("svrId");

UserSystemState.storeCurrentSvrID(svrId);

//Store away where we want to come back to
UserSystemState.storeReturnNavigationRule
("GlobalHome");
```

このコードでは、次の 2 つのことを行います。

- サービス・リクエスト ID を抽出し、`UserState` マネージド Bean に保存します。
- 編集の終了時に SRList ページに戻るためのナビゲーション・パスを設定します。

5. バッキング Bean に、さらに新規のクラス変数を次のように追加します。

```
private BindingContainer bindings;
```

（コード・アシストにより入力を求められた場合は、[Alt] を押しながら [Enter] を押して、パッケージ `oracle.binding` をインポートします。）

6. **「バインディング」**を右クリックして、コンテキスト・メニューから**「アクセッサの生成」**を選択します。「アクセッサの生成」メニューで**「OK」**をクリックして、setBindings メソッドと getBindings メソッドを生成します。
7. **SRSearch.java** ファイルを保存します。

「View」ボタンの組込み

「View」ボタンはアプリケーションのすべてのユーザーから使用可能なので、SRMain ページに移動して、選択済のリクエストの履歴を更新できます。

次の手順を実行して、「View」ボタンの機能を指定します。

1. ビジュアル・エディタで SRSearch.jspx に戻り、2 番目のコマンド・ボタン（**「View」**ボタン）を選択します。
2. プロパティ・インスペクタで、Id プロパティを **viewButton** に設定します。
3. 「Edit」ボタンと同様に、「View」ボタンのクリック時に発生する必要があることを指定する、バッキング Bean のメソッドを作成する必要があります。**「View」**ボタンをダブルクリックして、ページのバッキング Bean を起動します。「バインド Action プロパティ」ダイアログ・ボックスで、**「OK」**をクリックして、**viewButton_action** メソッドをバッキング Bean に追加します。
4. SRList.java ファイルで、viewButton_action() メソッドに生成されたコードを次のコードに置き換えます。

   ```
   return drillDown_action();
   ```

5. 前述と同様に、**「コード・アシスト」**ボタンをクリックして、drillDown_action メソッドを作成します。

   このメソッドでは、前述の項で作成した setCurrentSvrIdFromRow メソッドを再度使用します。現在選択済のサービス・リクエストの ID を SRMain ページに渡します。これにより、該当するレコードが取得および表示できます。
6. メソッドの下部に次のコードを追加して、この動作を指定します。

   ```
   setCurrentSvrIdFromRow();
   return "view";
   ```

7. ファイルを保存します。

   ページが完成したので、ページを実行して機能を確認できます。
8. SRSearch ページを開いた状態で、ビジュアル・エディタで右クリックして、**「実行」**を選択します。入力が求められたら、ユーザー名に **bernst** と入力し、パスワードに **welcome** と入力します。検索機能をテストします。（**注意：**後の章で SRMain ページおよび SREdit ページを完成するまで、「View」ボタンおよび「Edit」ボタンを使用できません。）

ページは次のように表示されます。

まとめ

この章では、JavaServer Faces および ADF コンポーネントを使用して検索ページを作成しました。これを実現するために、次の主なタスクを実行しました。

- 検索ページの作成
- 検索ページへのデータ・コンポーネントの追加
- 「View」ボタンおよび「Edit」ボタンの組込み
- 問合せのデフォルトの動作の変更

8

マスター / ディテール・ページの開発

この章では、サービス・リクエストおよびサービス・リクエスト履歴行を表示する、マスター / ディテール・ページを開発します。このページから、ユーザーはサービス・リクエストのスコープおよび履歴を表示できます。詳細なノートをサービス・リクエストに追加することもできます。

この章の内容は次のとおりです。

- 概要
- サービス・リクエスト・コンポーネントの追加
- ノート・パネルの追加
- サービス履歴パネルの追加
- まとめ

概要

SRMain ページには、サービス・リクエストおよびサービス・リクエストの履歴行のマスター / ディテール表示が表示されます。ページには、読取り専用のサービス・リクエスト・フォーム、ノート入力領域、およびサービス・リクエスト履歴表の 3 つのコンポーネント領域があります。

この章では、次の主なタスクを実行します。

- サービス・リクエスト読取り専用フォームの作成
- ノート入力フォームの作成およびパラメータからデータの値をプログラム的に設定するコードの追加
- サービス・リクエスト履歴表の追加

注意： 第 7 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **Chapter8** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、**<jdev_install>¥jdev¥mywork¥Chapter8** になります。
2. **<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter7.zip** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「<jdev_install>¥jdev¥mywork¥Chapter8¥SRDemo¥SRDemo.jws」**を選択します。この章の初期アプリケーションが開きます。

第 7 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

基本の UI の開発

この章の最初に、SRMain ページの作成および基本の UI コンポーネントの追加を行います。これは、後から追加するデータベース対応のコンポーネントを保持するために使用するレイアウト・コンポーネントです。

次の手順を実行して SRMain ページを作成し、先に作成したテンプレートをこのページにコピーします。

1. 開いていない場合は、**「faces-config.xml」**ファイルをダブルクリックして、ページ・フロー・ダイアグラムを表示します。
2. **「/app/SRMain.jspx」**ページをダブルクリックして、JSF ページ・ウィザードを起動します。
3. ウィザードの最初の 3 つのステップに対する値が、次の表の値と一致していることを確認します。

ウィザード・ステップ 1: JSP ファイル

フィールド	値
ファイル名	SRMain.jspx
ディレクトリ名	ディレクトリ名は、「…public_html¥app」になります。
タイプ	JSP Document
モバイル	チェック・ボックスの選択を解除。

4. 「次へ」をクリックして続行します。

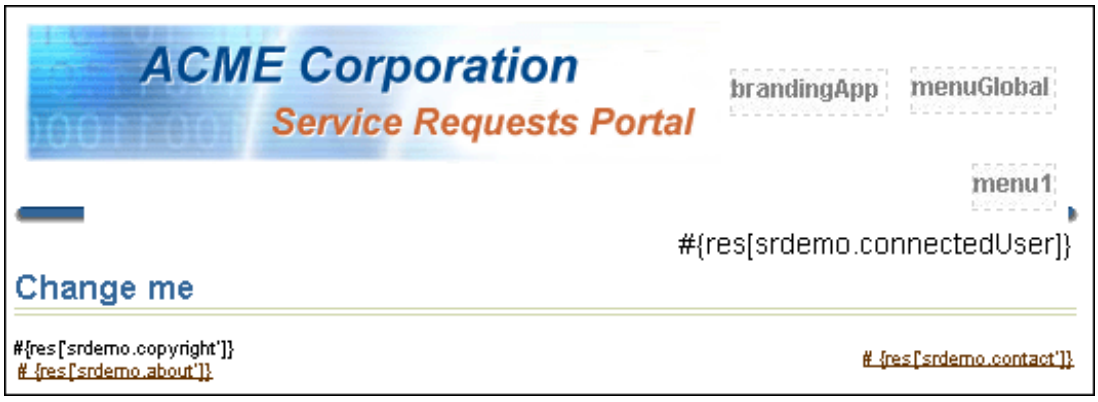
ウィザード・ステップ 2: コンポーネント・バインディング	
フィールド	**値**
新規マネージド Bean での UI コンポーネントの自動公開	このオプションが選択されていることを確認します。
名前	`backing_app_SRMain`
クラス	`SRMain`
パッケージ	`oracle.srdemo.userinterface.backing.app`

5. 「次へ」をクリックして続行します。

ウィザード・ステップ 3: タグ・ライブラリ	
フィールド	**値**
選択済のライブラリ	`ADF Faces Components`
	`ADF Faces HTML`
	`JSF Core`
	`JSF HTML`

6. **「終了」**をクリックして、ページ詳細を作成します。ビジュアル・エディタに、新規の SRMain ページが表示されます。
7. （まだ開かれていない場合は、）ビジュアル・エディタで**「SRDemoTemplate.jspx」**を開きます。構造ウィンドウで、**「afh:html」**ノードを閉じて選択します。ショートカット・メニューから、**「コピー」**を選択します。
8. タブをクリックして SRMain ページに戻り、構造ウィンドウで**「f:view」**ノードを選択します。
9. **「html」**ノードを削除します。**「f:view」**を右クリックして、ショートカット・メニューから**「貼付け」**を選択します。
   先に作成したルック・アンド・フィールが新しいページに適用されます。

   ページは次のように表示されます。



サービス・リクエスト・コンポーネントの追加

基本のページを作成したので、サービス・リクエスト・データを表示する、データベース対応のコンポーネントの追加を開始できます。

最初に、ページの位置合せおよびフォーマットに役立つ、ADF Layout コンポーネントを追加する必要があります。3 つの rowLayout コンポーネントを含む ADF tableLayout コンポーネントを追加します。各 rowLayout コンポーネントにはこのページの 1 セクションが保持されます。

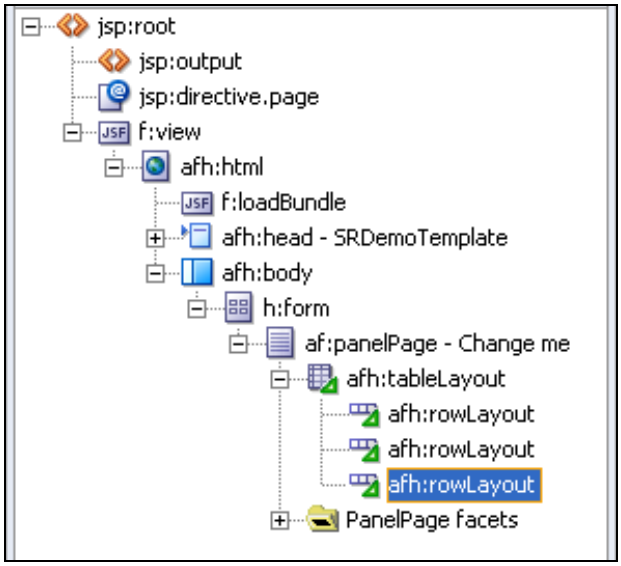
1. 「**ADF Faces HTML**」→「**TableLayout**」コンポーネントを選択して、これを「構造」ペインの **`af:panelPage`** にドラッグします。

   このコンポーネントは、バッキング Bean を使用してプログラム的に操作できる以外は HTML 表に類似しています。たとえば、`Rendered` プロパティを false に設定し、表および表の全コンポーネントの表示を無効にできます。

2. 「**ADF Faces HTML**」→「**RowLayout**」コンポーネントを追加して、これを直前に追加した **`afh:tableLayout`** コンポーネントにドラッグします。このコンポーネントはレイアウト・オブジェクトを提供し、(tableLayout コンポーネントと同様に) プログラム的に変更できます。

3. 前述の手順を繰り返し、2 番目と 3 番目の **rowLayout** コンポーネントを追加します。

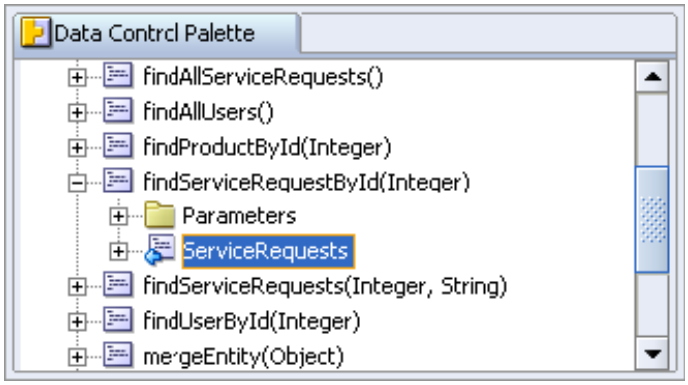
   構造ウィンドウは、次のスクリーンショットのように表示されます。



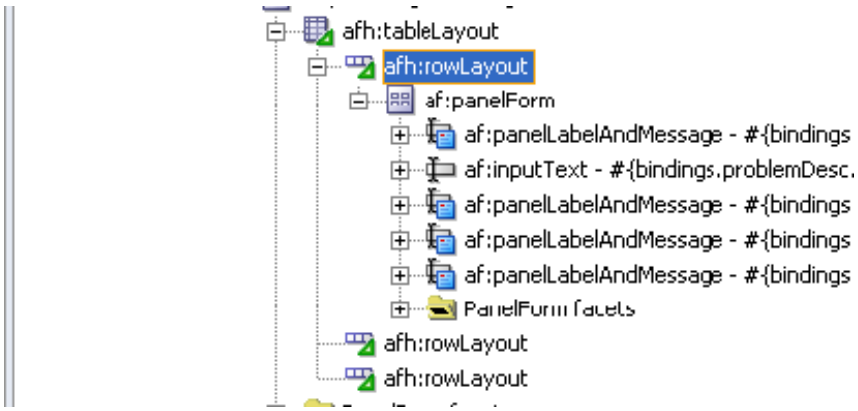
データベース対応コンポーネントの追加

次の手順を実行して、サービス・リクエストを表示するために ADF データベース対応コンポーネントを追加します。

1. ビジュアル・エディタで**「SRMain.jspx」**タブをクリックして、ページを再度開きます。
2. **「データ・コントロール・パレット」**を選択して、「SRPublicFacadeLocal」ノードを開きます。
3. **ADF 読取り専用フォーム**として、**「findServiceRequestById(Integer)」**の結果コレクションを、構造ウィンドウの最初の「`afh:rowLayout`」にドラッグします。コレクションはメソッド名の下のノードです。



4. フォーム・フィールドの編集フィールド・ダイアログ・ボックスで、**problemDescription** 列でラベル付 ADF 入力テキストを使用するように変更します。problemDescription の隣の**使用するコンポーネント**列で**ラベル付 ADF 入力テキスト**を選択します。problemDescription フィールドは、ブラウザのテキスト入力ウィジェットに変更されます。後で高さおよび幅などの表示プロパティを変更できます。**「OK」**をクリックします。
5. アクション・バインディング・エディタで、findSvrID パラメータ用のデフォルト値を入力します。Value プロパティにより、パラメータのソースが決定されます。EL ブラウザ（...）を使用して、「JSF マネージド Bean」ノードから**「`${userState.currentSvrId}`」**を選択します。**「OK」**をクリックします。
6. 構造ウィンドウをチェックして、次のように表示されることを確認します。



レイアウトの調整

次の手順を実行し、構造ウィンドウを使用して、コンポーネントを追加および配置します。この方法では、レイアウト・コンポーネントが他のコンポーネントに対して適切な位置にあることを簡単に確認できます。

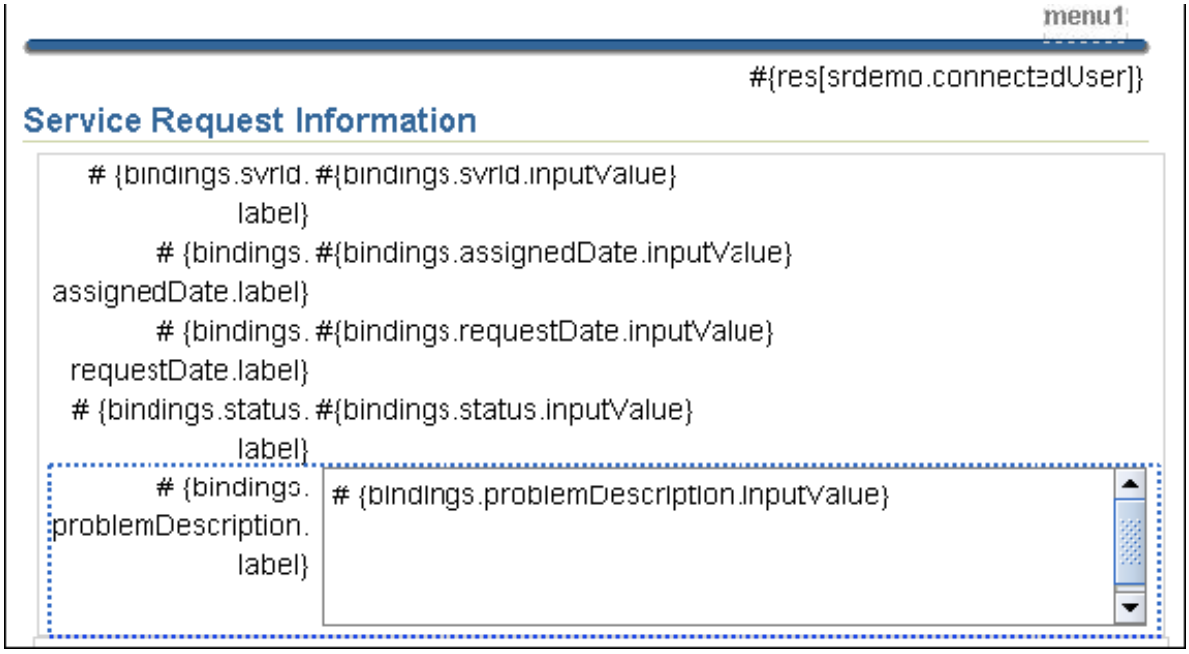
1. **af:panelLabelAndMessage** コンポーネントを再配置して、**svrID** コンポーネントが最初で、**problemDescription** が最後になるようにします。これを実行するには、構造ウィンドウを使用して、コンポーネントを対応する適切な場所にドラッグします。
2. panelPage のタイトルを、Change Me から **Service Request Information** に変更します。（「構造」ペインで「**af:panelPage**」を選択し、プロパティ・インスペクタで Title プロパティを変更します。）
3. 構造ウィンドウおよびプロパティ・インスペクタを再度使用して、problemDescription が表示される行数を変更します。「**problemDescription**」を選択して、Rows プロパティを **4** に設定します。

   problemDescription はマルチライン・フィールドであるため、複数の説明を表示できます。Rows プロパティを変更して、自動的にスクロール機能をウィジェットに追加することもできます。
4. **Columns** プロパティを **35** に変更します。

   フィールド幅が 35 文字になります。このプロパティを空白のままにすることができます。JSF ではデフォルト値を使用できますが、さらにきめ細かなレイアウトの制御が可能になります。
5. **ReadOnly** プロパティを **true** に変更します。

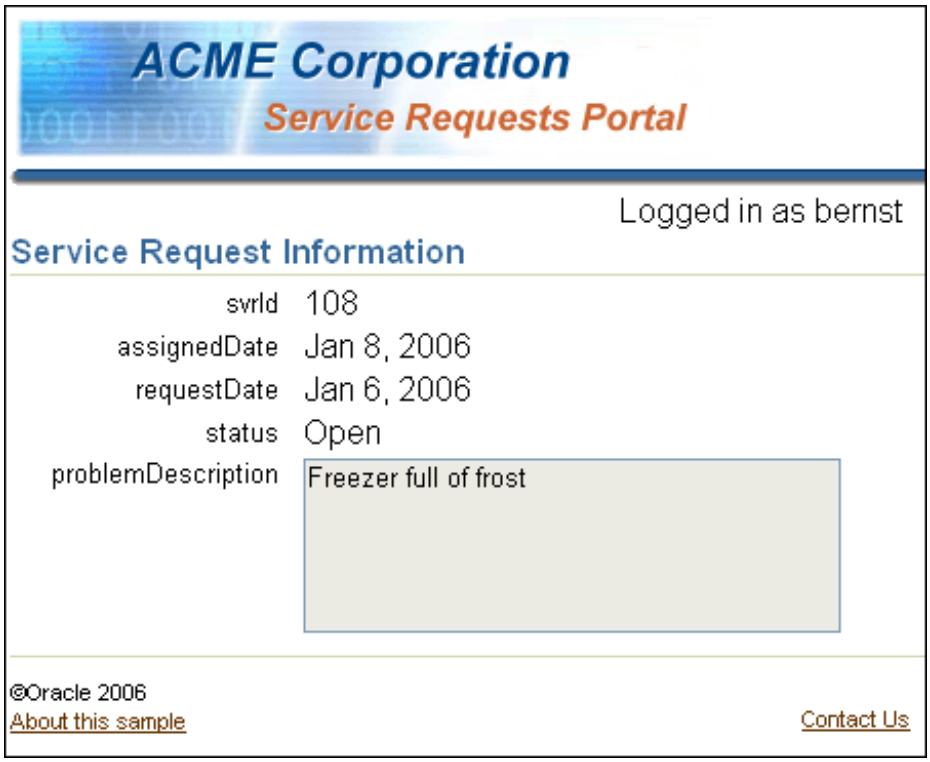
   テキスト入力コンポーネントのデフォルトは更新可能ですが、フォームのこの部分は読取り専用なので、このフィールドを readOnly に変更する必要があります。

   ページは、次のスクリーンショットのように表示されます。



6. SRMain ページは、userState.currentSvrId からサービス・リクエスト・コンテキストを取得します。このページをスタンドアロンとして実行できますが、SRList ページからコールされないと、データは表示されません。ページを実行してここまでの結果を表示するには、「**SRList**」ページで右クリックして、コンテキスト・メニューから「**実行**」を選択します。

   ユーザー名 **bernst** およびパスワード **welcome** で、アプリケーションにサインオンします。SRList ページには、現在のユーザーに対するサービス・リクエストがすべて表示されます。任意のサービス・リクエストを選択して、「**View**」ボタンをクリックします。選択したサービス・リクエストを使用して、SRMain ページが表示されます。ページは次のように表示されます。



ノート・パネルの追加

このページの 2 番目のセクションは、ユーザーが新しい行を ServiceRequest 履歴に追加できるエントリ・フォームです。フォームには、ノート・フィールドおよびボタンのみ表示されます。

ノート・パネルは、serviceHistories カスタム・コンストラクタに基づいています。ページのロード時に、コンストラクタがコールされます。フォームから値を受け入れるイテレータで行が作成され、EL を使用してアクセス可能になります。コンストラクタは、現在のサービス・リクエストおよび現在のユーザー・オブジェクトの 2 つの引数を受け入れます。

次の手順を実行して、カスタム・コンストラクタを作成します。

1. アプリケーション・ナビゲータで、「**DataModel**」→「**アプリケーション・ソース**」→「**oracle.srdemo.datamodel**」を開きます。
2. 「**ServiceHistories.java**」をダブルクリックして、コード・エディタでこれを開きます。

3. デフォルト・コンストラクタのすぐ下に、次のコードを追加します。デフォルトのコンストラクタは、

```
public ServiceHistories() {
  …
}
```

として定義されています。

```
public ServiceHistories(ServiceRequests sr, Users user) {
     this();
     sr.addServiceHistories(this);
     setUsers(user);
     setLineNo(null);
     setSvhDate(null);
}
```

4. コンストラクタの作成後に、データ・コントロールを再作成します。**「SRPublicFacadeBean.java」**を右クリックして、コンテキスト・メニューから**「データ・コントロールの作成」**を選択します。

ノートの追加セクションの作成

カスタム・コンストラクタを作成し、データ・コントロールに含めたので、フォームを追加して、サービス履歴行を作成できます。フォームを追加する場合、コンストラクタに対する 2 つの引数の入力を求められます。最初の引数は現在のサービス・リクエストで、findServiceRequestById バインディングから取得します。2 番目の引数はユーザー・オブジェクトで、userInfo Bean から取得します。

1. ビジュアル・エディタで、**「SRMain.jspx」**タブをクリックします。
2. 「データ・コントロール・パレット」を選択し、**「SRPublicFacadeLocal」→「コンストラクタ」→「oracle.srdemo.datamodel.ServiceHistories」→「ServiceHistories(ServiceRequests, Users)」**メソッドを、構造ウィンドウの 2 番目の **`afh:rowLayout`** にドラッグします。
3. 「フォーム・フィールドの編集」ダイアログ・ボックスで、**「送信ボタンを含める」**を選択して、**「OK」**をクリックし、デフォルトのフィールド値を受け入れます。
4. アクション・バインディング・エディタで、**「sr」**パラメータを選択して、**「value」**プロパティをダブルクリックします。**「参照」**ボタンをクリックして、EL ピッカーを開きます。
5. **「ADF バインディング」→「バインディング」→「findServiceRequestByIdIter」→「currentRow」**を開きます。**「dataProvider」**をダブルクリックして、式を作成します。**「OK」**をクリックします。コードは次のように表示されます。

   `${bindings.findServiceRequestByIdIter.currentRow.dataProvider}`

6. **user** パラメータの「value」プロパティをダブルクリックし、「参照」ボタンをクリックして EL ピッカーを開きます。
7. **「JSF マネージド Bean」→「userInfo」**を選択し、**「userobject」**をダブルクリックしてこれを式に追加します。**「OK」**をクリックします。コードは次のように表示されます。

   `${userInfo.userobject}`

8. **「OK」**をクリックして、アクション・バインディング・エディタを閉じます。

   ユーザーがこのパネルを使用してノートを追加する場合に、使用する必要があるフィールドは「ノート」フィールドのみです。次の手順では、「構造」ペインおよびプロパティ・インスペクタで false である「ノート」フィールド以外の、すべてのフィールドの rendered プロパティを変更します。

9. 「構造」ペインで**「af:inputText」**項目を選択し、**rendered** プロパティを notes.inputValue 以外の属性ごとに **false** に変更します。

   注意 : [Ctrl] を押しながら [Click] を押して、すべての属性を複数選択し、プロパティ・インスペクタで共通の rendered プロパティを変更できます。

10. **notes.inputValue** コンポーネントの **Rows** プロパティを **4** に変更します。このフィールドをテキスト領域タイプのフィールドにします。

11. Columns プロパティを **35** に変更します。

12. Label プロパティを **SR Notes** に変更します。

コードの値の導出

最後に、このパネルに対して、カスタム・コードを、データを追加して履歴行を保持する「Add a Note」ボタンに追加する必要があります。ユーザーが値をそれぞれ入力するのではなく、現在のサービス・リクエストおよび現在のユーザーから値を導出します。

1. コンポーネント・パレットで、**「SRPublicFacadeLocal」→「persistEntity(Object)」**を選択して、これをビジュアル・エディタの「Submit」ボタンにドラッグします。コンテキスト・メニューから**「既存の CommandButton のバインド」**を選択します。

2. アクション・バインディング・エディタで、value プロパティを **${bindings.ServiceHistories.result}** に設定します。**「OK」**をクリックして続行します。

3. **「persistEntity」**ボタンを選択して、次のプロパティを変更します。

フィールド	値
Text	`Add a Note`
Id	`addNoteButton`

   後でデータをリフレッシュできるのは、このボタンがクリックされる場合のみです。ADF には、ボタンのクリック時のみ実行される setActionListener コンポーネントが用意されています。setActionListener は、from 句および to 句の 2 つの引数を受け入れます。リフレッシュ条件では、to 句の値をチェックして、ボタンが押されたかどうかを確認します。

4. 構造ウィンドウで、**「af:commandButton（Add a Note)」**を右クリックします。コンテキスト・メニューから、**「af:commandButton の中に挿入」→「ADF Faces Core」→「SetActionListener」**を選択します。

5. 「SetActionListener の挿入」ダイアログ・ボックスで次の値を入力して、**「OK」**をクリックします。

フィールド	値
From	`#{true}`
To	`#{requestScope.createNewSH}`

6. **「Add a Note」**ボタンをダブルクリックします。「バインド Action プロパティ」ダイアログ・ボックスが開きます。**「ADF バインディング・コードの生成」**チェック・ボックスが選択されていることを確認し、**「OK」**をクリックして、メソッド名のデフォルト値を受け入れます。

   「バインド Action プロパティ」ダイアログ・ボックスが閉じて、SRMain.java ファイルの addNoteButton_action() メソッドに移動します。ここで、svhType 引数の設定、エンティティの保持およびサービス・リクエストの再問合せを行うカスタム・コードを追加します。

ユーザーが「**Add a Note**」ボタンをクリックすると、次のコードが実行されます。ユーザーは、フォームにノート説明を入力するのみです。コードは、ユーザーに基づくノートのタイプを判別します。サインオンしているユーザーが顧客の場合、値には Customer が設定され、ユーザーがスタッフの場合、タイプには Technician が設定されます。

7. 次のように svhType を判別します。次のコードを `AddNoteButton_action()` メソッドに、メソッドの最初の行としてコピーします。

```
// START CUSTOM CODE TO SET SVHTYPE

FacesContext ctx = FacesContext.getCurrentInstance();
String callType = "Customer";
UserInfo user =
    (UserInfo)JSFUtils.getManagedBeanValue(ctx, "userInfo");
    if (user.isStaff()) {
        callType = "Technician";
    }
ADFUtils.setPageBoundAttributeValue(getBindings(),"svhType", callType);
```

8. このコードをメソッドにコピーする場合、JDeveloper では必要なクラスをインポートするよう求められます。次のインポートを受け入れます。

```
javax.faces.context.FacesContext
oracle.srdemo.view.UserInfo
oracle.srdemo.view.util.JSFUtils
oracle.srdemo.view.util.ADFUtils
```

9. 次のようにエンティティを保持します。JDeveloper では、「submit」ボタンに `persistEntity()` メソッドをドロップした際にエンティティを保持するコードが作成されます。これは、次の内容を実行するコードです。`addNoteButton_action()` メソッドにはすでに設定されているので、変更する必要はありません。

```
BindingContainer bindings = getBindings();
OperationBinding operationBinding = bindings.getOperationBinding("persistEntity");
Object result = operationBinding.execute();
```

10. 次のようにサービス・リクエスト・イテレータをリフレッシュします。ユーザーがノートを追加し、エンティティを保持する場合、（次の項で追加する）ページの下部にある表をリフレッシュする必要があります。メソッドの OperationBinding を取得し、これを実行することによって、イテレータをリフレッシュします。エンティティを保持するコードの後に、次のコードを追加します。

```
…
Object result = operationBinding.execute();
//now re-execute the iterator to refresh the screen
OperationBinding requery = bindings.getOperationBinding("findServiceRequestById");
requery.execute();
```

これは、addNoteButton_action() メソッドのすべてのコードです。

最後に、`ServiceHistoriesIterator` の自動リフレッシュ時間および回数を指定する必要があります。

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

12. 構造ウィンドウで、「**ServiceHistoriesIter**」を選択します。

13. プロパティ・インスペクタで、RefreshCondition を次のように変更します。

```
${(!adfFacesContext.postback || requestScope.createNewSH) and
         empty bindings.exceptionsList}
```

Faces ポストバックのためページ・リフレッシュが実行されない場合または createNewSH が TRUE の場合に、自動リフレッシュの実行を確認することをこの条件によりチェックします。これは、setActionListener イベントで追加したコードです。例外リストでエラーをバインディングしている場合、リフレッシュを行わないようにすることもできます。

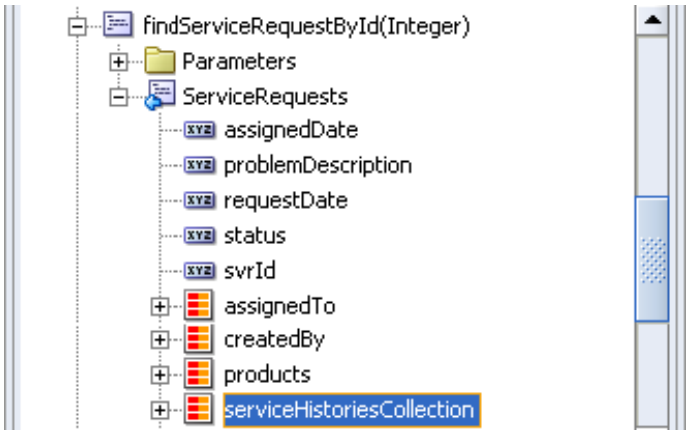
データベースにレコードをコミットしているので、ローカル・トランザクションとしてトランザクションを管理するよう、OC4J のプリファレンスを設定する必要があります。OC4J のプリファレンスでこれを行います。

14. OC4J サーバーが稼働していないことを確認してください。メニューから、**「実行」→「OC4J の終了」**を選択します。
15. アプリケーション・ナビゲータで**「DataModel」**プロジェクトをクリックし、メニューから**「ツール」→「埋込み OC4J の設定」**を選択します。
16. **「Current Workspace (SRDemo)」→「DataSources」**を開きます。
17. **「jdev-connection-managed-SRDemo <Managed Data Sources>」**をクリックします。
18. トランザクション・レベルを **Local** に設定し、**「OK」**をクリックします。

サービス履歴パネルの追加

SRMain ページに必要な、最後の UI コンポーネントは、現在のサービス・リクエストに対するサービス・リクエスト履歴レコードを表示する読取り専用パネルです。

1. ビジュアル・エディタで、**「SRMain.jspx」**タブをクリックします。
2. コンポーネント・パレットで、**「findServiceRequestById(Integer)」→「ServiceRequests」**を開きます。これは、このページ（サービス・リクエストを表示するページ）の上位パネルに使用するコレクションです。



3. データ・コントロール・パレットから、**「serviceHistoriesCollection」**を構造ウィンドウの最後の（3 番目の）`afh:rowLayout` に **ADF 読取り専用表**としてドラッグします。
4. 「表の列の編集」ダイアログ・ボックスで、**「選択を有効にする」**チェック・ボックスの選択が解除されていることを確認します。**「OK」**をクリックして、デフォルトを受け入れます。
5. ビジュアル・エディタで、**「Notes」**列、**「svhType」**列および**「svhDate」**列を除く、すべての列を削除します。（列でクリックして [Delete] を押すか、右クリックしてコンテキスト・メニューから「削除」を選択します。）
6. 構造ウィンドウで、列を次の順番に再配置します。
   svhDate
   svhType
   notes

   SRList ページからページをテストできます。SRList ページで選択した同じ列がページに表示されることを確認します。
7. アプリケーション・ナビゲータで**「SRList.jspx」**を右クリックして、**「実行」**を選択します。サインオンしているユーザーとして、ユーザーに属するサービス・リクエストのいずれかを表示するよう選択することもできます。ノートを任意のサービス・リクエストに追加します。ページの下部にあるディテール表に表示されます。テストが終了したら、ブラウザを閉じます。

まとめ

この章では、ADF Faces コンポーネントを使用して、マスター / ディテール・ページを作成しました。このコンポーネントを使用すると、ページの 2 つのパネル間で連携されたデータを表示できます。コードを追加しないと、履歴パネルには、上位パネルに表示されたサービス・リクエストに関連付けられた行のみ表示されます。

コール側のページから送信されたパラメータに基づいたページを、リフレッシュする処理を追加することもできます。これは、複数ページ全体で連携されたデータが SRMain ページに表示されることで確認しました。

この章で実行した主なタスクは次のとおりです。

- サービス・リクエスト読取り専用フォームの作成
- 新規のサービス履歴行を作成するカスタム・コンストラクタの作成
- ノート入力フォームの作成
- サービス・リクエスト履歴表の追加

9

トランザクション機能の実装

この章では、サービス・リクエストを作成するページの作成方法について説明します。サービス・リクエストのプロセスは、3 つの主なページ、つまり製品と問題を指定するページ、値を確認するページ、サービス・リクエスト ID をコミットおよび表示するページで構成されています。4 つ目のページを作成することもできます。このページには、一般的な製品の問題の解決に関する FAQ を表示します。

この章の内容は次のとおりです。

- 概要
- 作成ページの開発
- 確認ページの作成
- 完了ページの作成
- FAQ ページの作成
- ページの実行
- まとめ

概要

レコードの作成および削除のトランザクション操作はプロセスに類似しています。レコードの作成時には、ユーザーは情報を入力し、作成ボタンをクリックして、作成の確認を受信します。レコードの削除時には、ユーザーはレコードを選択し、削除ボタンをクリックして、削除の確認を受信します。どちらの場合も、ユーザー処理に間違いがないかどうかユーザーに尋ねるステップを追加することができます。

この章では、次の 4 つの画面を作成します。

- サービス・リクエスト・レコード（SRCreate）を作成または削除するページ
- 新規作成されたサービス・リクエスト値（SRCreateConfirm）を含む、処理を確認するページ
- 作成の成功（SRCreateDone）を確認するページ
- 可能性のあるサービス・リクエストの解決策（SRFaq）を説明する静的テキストを表示するページ

この章では、次の主なタスクを実行します。

- 作成、確認および完了ページの作成
- UIResources.properties から値を取得するプロンプトの変更
- データ・バインドされたオブジェクトに対するコンポーネントの追加
- トランザクションの制御用コマンドの追加
- トランザクションの値の管理
- ページ間のパラメータの渡しおよび移動
- 作成プロセスの進行状況を表示するプロセス・トレインの定義およびアクティブ化
- FAQ の作成およびリンク

注意： 第 8 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **`Chapter9`** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、**`<jdev_install>¥jdev¥mywork¥Chapter9`** になります。
2. **`<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter8.zip`** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「`<jdev_install>¥jdev¥mywork¥Chapter9¥SRDemo¥SRDemo.jws`」**を選択します。この章の初期アプリケーションが開きます。

第 8 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

作成ページの開発

SRCreate ページを使用すると、ユーザーはサービス・リクエストを新規作成できます。SRList ページのメイン・メニューはこのページをコールできます。すべてのページで使用可能なグローバル・メニューの「サービス・リクエストの新規作成」をコールすることもできます。

SRCreate を使用すると、ユーザーは、すべての電化製品のリストから選択し、説明を入力できます。ユーザーは、説明の入力後に「Continue」ボタンをクリックすると、確認ページにアクセスできます（SRCreateConfirm の詳細を参照）。

ただし、ユーザーが情報を入力する前に、FAQ ページを表示するオプションがあります。このページには、クリック時に、対応する回答とともに FAQ 一式を表示するモデル・ページが表示されます。SRCreate ページの別の重要な機能として、「Cancel」ボタンをクリックすると、サービス・リクエストの新規作成を取り消すオプションがあります。「Cancel」をクリックすると、すべてのフォームの検証を省略して、SRList ページに戻ります。

注意： チュートリアルの先の手順において、ページ・フロー・ダイアグラムでページのアウトラインを作成しました。この章では、ページを完了し、第 4 章で作成したテンプレートを適用します。

次の手順を実行して、作成ページを作成し、先に作成したテンプレートを使用します。

1. 開いていない場合は、**「faces-config.xml」**ファイルをダブルクリックして、ページ・フロー・ダイアグラムを表示します。
2. **「SRCreate」**ページをダブルクリックして、JSF ページ・ウィザードを起動します。
3. ウィザードの最初の 3 つのステップに対する値が、次の表の値と一致していることを確認します。

ウィザード・ステップ 1: JSP ファイル	
フィールド	**値**
ファイル名	SRCreate.jspx
ディレクトリ名	ファイルが保存される場所。デフォルト値で問題ありません。
タイプ	JSP Document
モバイル	チェック・ボックスの選択を解除。

4. **「次へ」**をクリックして続行します。

ウィザード・ステップ 2: コンポーネント・バインディング	
フィールド	**値**
新規マネージド Bean での UI コンポーネントの自動公開	このオプションが選択されていることを確認します。
名前	backing_app_SRCreate
クラス	SRCreate
パッケージ	oracle.srdemo.userinterface.backing.app

5. 「次へ」をクリックして続行します。

ウィザード・ステップ 3: タグ・ライブラリ	
フィールド	**値**
選択済のライブラリ	ADF Faces Components ADF Faces HTML JSF Core JSF HTML

6. 「終了」をクリックして、ページ詳細を作成します。ビジュアル・エディタに、新規の SRCreate ページが表示されます。
7. まだ開かれていない場合は、**SRDemoTemplate** ファイルを開きます。構造ウィンドウで「**afh:html**」ノードを右クリックして、ショートカット・メニューから「**コピー**」を選択します。
8. タブをクリックして **SRCreate** ページに戻り、構造ウィンドウで、「**f:view**」ノードを開きます。
9. 「**html**」ノードを削除します。「**f:view**」を右クリックして、ショートカット・メニューから「**貼付け**」を選択します。
10. 構造ウィンドウで、「**afh:html**」ノードをダブルクリックして、Title プロパティを **`SRDemo Create`** に変更します。
11. 「**OK**」をクリックします。ビジュアル・エディタにより、他のページのルック・アンド・フィールで、SRCreate ページが表示されます。

作成ページの変更

この項では、SRCreate ページ構造を変更して、別のタイトルの表示、レイアウトの変更および出力項目の組込みを行います。サービス・リクエストの作成は一種のプロセスなので、そのプロセスを追跡することは役に立ちます。

1. 構造ウィンドウで、「**f:view**」→「**afh.html**」→「**afh.body**」→「**h:form**」ノードを開いて、af:panelPage を表示します。
2. 「**`af:panelPage`**」をダブルクリックして、Title プロパティを **`#{res['srcreate.pageTitle']}`** に変更します。

   このプロパティをタイトル用に選択した文字テキストに設定できます。プロパティを、複数のプロパティと値が組になったリストを含む、`UIResources.properties` ファイルの値に設定します。実行時に、このファイルが読み込まれ、ページ上で値が置換されます。ページ・タイトルは、「Create a New Service Request」になります。
3. プロセス・トレインを作成して、サービス・リクエストの作成プロセスを追跡します。ADF Faces Core カテゴリから、**ProcessTrain** を af:panelPage コンポーネントにドラッグします。すべてのページを通じて、作成プロセスの進行状況を視覚的に表示できます。

4. 「**af:processTrain**」を選択します。プロパティ・インスペクタで、プロパティを次の表の値に設定します。

フィールド	値
Value	`#{createTrainMenuModel.model}`
Var	`train`
Id	`createStepsTrain`

5. 構造ウィンドウで、「**af:processTrain**」→「**ProcessTrain**」**ファセット**を開いて、**nodeStamp** を表示します。ADF Faces Core カテゴリから、**CommandMenuItem** を nodeStamp にドラッグします。

6. **af:commandMenuItem** プロパティを選択および変更して、次の表と一致するようにします。

フィールド	値
Text	`#{train.label}`
Action	`#{train.getOutcome}`
Id	`trainNode`

7. 「**af:commandMenuItem**」を選択した状態で、「**ソース**」タブをクリックして、もう 1 つプロパティ **`readOnly="#{createTrainMenuModel.model.readOnly}"`** を設定します。

```
<af:commandMenuItem text="#{train.label}"
                    binding="#{backing_jag_SRCreate.trainNode}"
                    id="trainNode"
                    action="#{train.getOutcome}"
                    readOnly="#{createTrainMenuModel.model.readOnly}"/>
```

8. 「**設計**」タブをクリックします。構造ウィンドウで、ADF Faces Core カテゴリから **ObjectSpacer** をドロップします。af:processTrain の下に表示されます。

   これを行うには、**ObjectSpacer** を af:processTrain の右下の af:panelPage にドラッグします。別の場所に移動する必要がある場合、別のノードにドラッグすると、そのノードのリストの最後に追加されます。**Width** プロパティをクリアします。

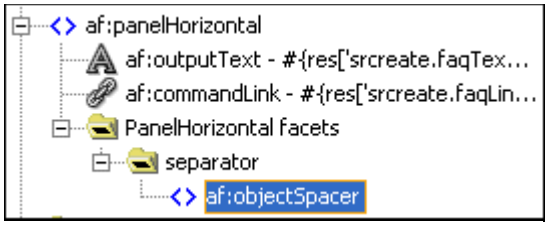
9. ページに余白を追加して項目用の場所を作成するには、**PanelHorizontal** コンポーネントを af:panelPage にドロップします。PanelHorizontal を af:panelPage にドロップすると、リストの最後で af:objectSpacer の下に追加されます。

10. 「**af:panelHorizontal**」ノードを開きます。ADF Faces Core カテゴリから、**OutputText** 項目および **CommandLink** を af:panelHorizontal ノード内にドロップします。出力テキストおよびコマンド・リンク項目により、問題を解決するための FAQ ページの確認に関する情報が表示されます。

11. af:outputText の Value プロパティを **`#{res['srcreate.faqText']}`** に設定します。af:commandLink プロパティを次の表の値に設定します。

フィールド	値
Text	`#{res['srcreate.faqLink']}`
Action	`FAQ`
PartialSubmit	`true`

12. テキストとリンクの間に余白を作成します。af:panelHorizontal ノードで、**ObjectSpacer** を「PanelHorizontal」ファセット→「セパレータ」ノードにドロップします。spacer の幅を **4** に設定します。構造ウィンドウは、次のスクリーンショットのように表示されます。



このページで実行およびテストする内容について、説明テキストを追加します。

13. 「**af:panelHorizontal**」ノードを閉じて、af:panelHorizontal の下に別の **ObjectSpacer** を追加します。これを行うには、**ObjectSpacer** を af:panelPage にドロップします。**Width** プロパティをクリアします。

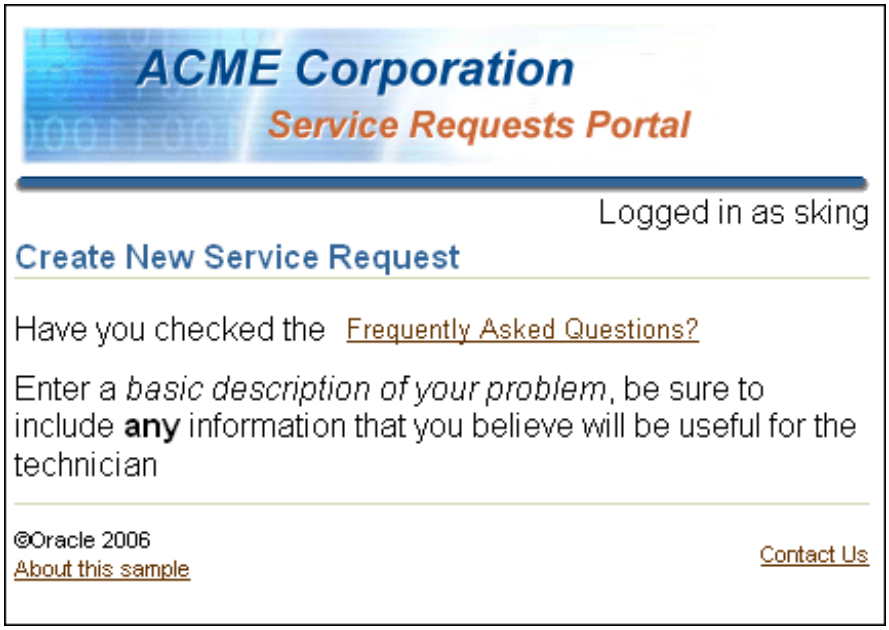
14. af:objectSpacer の下に、ADF Faces Core カテゴリから **OutputFormatted** 項目を追加します。

15. 「**af:outputFormated**」項目をダブルクリックして、Value プロパティを **`#{res['srcreate.explainText']}`** に設定します。これにより、ページにサービス・リクエストに関する情報の入力方法を説明するテキストが表示されます。

16. af:outputFormatted 項目の下に、別の **ObjectSpacer** を追加します。**Width** プロパティをクリアします。

17. ビジュアル・エディタで、コンテキスト・メニューから**「実行」**を選択します。ページが実行されると、`UIResources.properties` ファイルから導出されたヘッダーおよびラベルを配置したフォームの構造が表示されます。

18. 入力が求められたら、ユーザー名に **`sking`** を使用し、パスワードに **`welcome`** を使用します。



ページへのデータ・コンポーネントの追加

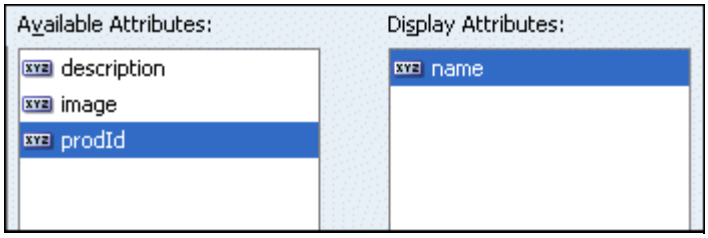
この項では、サービス・リクエストを作成するために、データ・コンポーネントを組み込みます。SRCreate の入力フォームを使用すると、ユーザーは、製造表から移入された電化製品リストから選択できます。問題の説明を入力するためのテキスト領域もいっしょに表示されます。「Continue」ボタンをクリックすると、確認のアクションが実行され、SRCreate の確認ページに移動します。このページでは、新規のサービス・リクエストをデータベースにコミットできます。

注意： コンポーネントをノードの最後に追加するときには、常に、親ノードにドロップします。自動的にリストの最後に追加されます。追加後に、そのコンポーネントを移動できます。

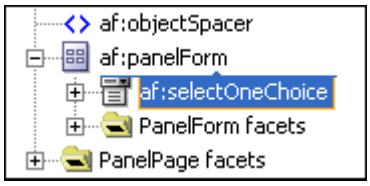
---

1. 最初に、データ・コンポーネントを構成するパネルが必要です。ADF Faces Core カテゴリで、**PanelForm** を構造ウィンドウにドラッグしてドロップし、最後の af:objectSpacer の下に表示されるようにします。これを行うには、PanelForm を af:panelPage にドロップします。
2. **「データ・コントロール・パレット」**タブをクリックします。**「SRPublicFacadeLocal」**→**「findAllProducts()」**ノードを開いて、戻り値ノードの**「Products」**を選択します。

3. このコンポーネントを af:panelForm にドラッグします。ポップアップ・メニューで、**「移動」**→**「ADF ナビゲーション・リスト」**を選択します。
4. リスト・バインディング・エディタで、**name** 属性のみが「属性の表示」リストにあることを確認してください。**「OK」**をクリックして続行します。



5. リストは af:panelGroup に作成されますが、af:panelGroup は必要ありません。構造ウィンドウで、**af:selectOneChoice** を af:panelForm にドラッグします。**「af:panelGroup」**を選択して削除します。



この操作では、af:panelGroup で Details のラベルが付いた、ネストされた af:panelHeader も削除されます。

6. ドロップダウン・リストではなくリストとして、Products のリストを表示します。「**af:selectOneChoice**」を再度選択して、コンテキスト・メニューから「**変換**」を選択します。下へスクロールして、ダイアログ・ボックスから、「**SelectOneListbox**」を選択して、「**OK**」をクリックします。

7. プロパティ・インスペクタで、af:selectOneListbox 項目に対して、Label プロパティを、UIResources.properties から取得される **#{res['srcreate.info.1']}** に設定します。また、AutoSubmit プロパティを **false** に設定します。このプロパティは、ある値を選択する際に発生する内容を制御します。true の場合、値を選択すると即座に発行されます。false の場合、明示的に発行の操作を実行するまで発行されません。

8. ADF Faces Core カテゴリから、**InputText** を af:panelForm にドラッグします。af:selectOneListbox の後に表示されます。この値を次の表の値に設定します。「**OK**」をクリックします。

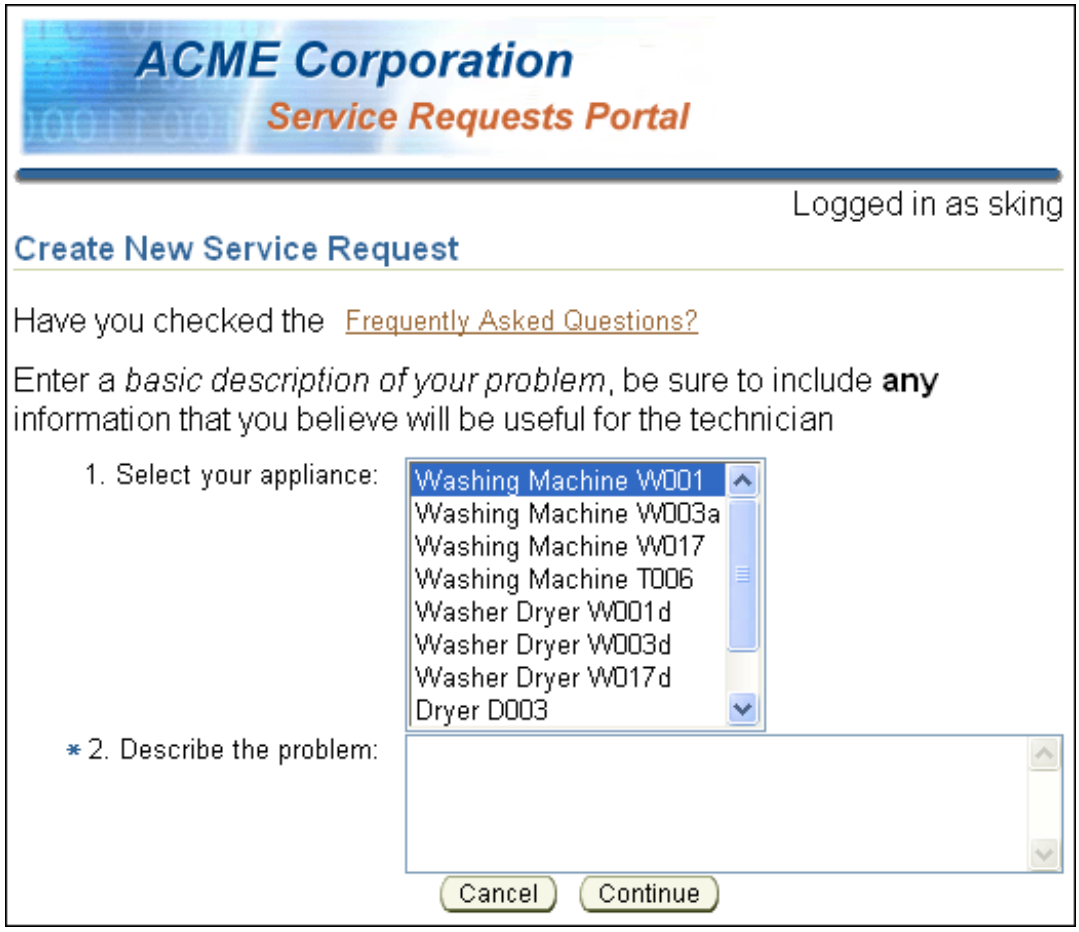
フィールド	値
Label	`#{res['srcreate.info.2']}`
Columns	`50`
Rows	`4`
Required	`true`

9. **PanelButtonBar** を「af:panelForm」→「PanelForm Facet」→「footer」にドロップします。

10. **CommandButton** を af:panelButtonBar にドロップします。ユーザーがサービス・リクエストの作成を取り消すためのものです。Text プロパティを **#{res['srdemo.cancel']}** に、Immediate プロパティを **true** に、Id プロパティを **cancelButton** にそれぞれ設定します。

11. 2 番目の **CommandButton** を af:commandButtonBar にドロップします。ユーザーがサービス・リクエストの作成を続行するためのものです。Text プロパティを **#{res['srdemo.nextStep']}** に、Action プロパティを **confirm** にそれぞれ設定します。

12. 「保存」をクリックし、「実行」をクリックして、ページを実行します。

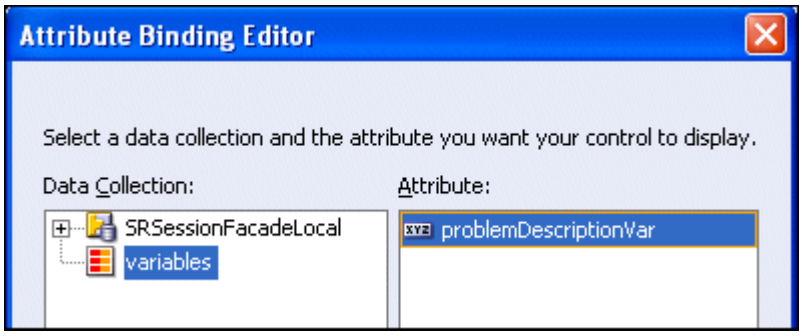


ユーザーが入力した問題の説明の保存

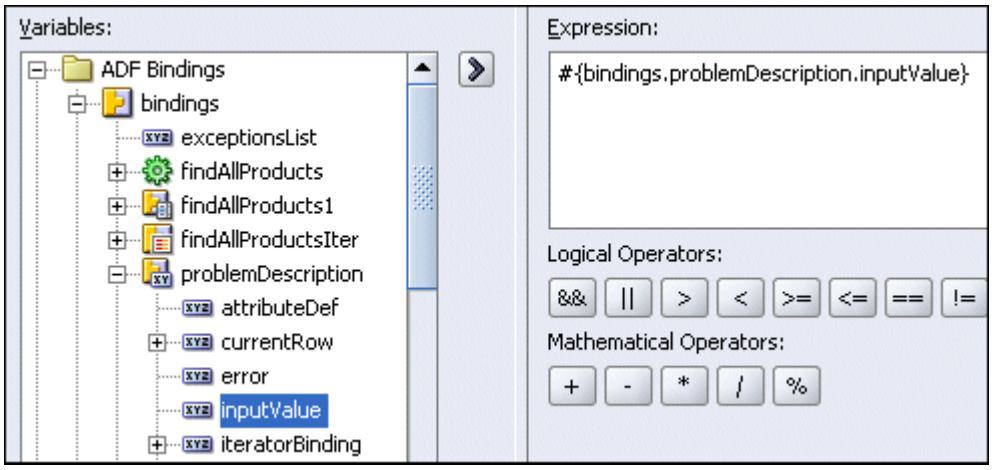
このページでは、ユーザーは、基本的に、リストからの製品の選択および問題の説明の入力という 2 つのことを行います。結果コレクションの現在の行を使用して、現在選択されている製品を記録します。これは、データ・コントロール・パレットから項目を作成したために実行されます。ただし、問題の説明フィールドはバインドされていません。変数を使用して、値を保存できます。

1. SRCreate ページ用のページ定義ファイルに変数を保存します。ビジュアル・エディタから、「**ページ定義に移動**」コンテキスト・メニューを使用して、ファイルに移動します。
2. 「**executables**」ノードで右クリックして、「**executables の中に挿入**」→「**variableIterator**」を選択します。
3. 構造ウィンドウで、「**executables**」ノードを開きます。「**variables**」ノードを選択して、コンテキスト・メニューから、「**variables の中に挿入**」→「**variable**」を選択します。変数の名前を **`problemDescriptionVar`** に、型を **`java.lang.String`** に設定します。「**OK**」をクリックします。
4. 構造ウィンドウの「**バインディング**」および「**Bindings の中に挿入**」→「**attributeValues**」を選択します。

5. 属性バインディング・エディタで、**「変数」**ノードを選択すると、「属性」パネルに problemDescriptionVar が表示されます。これを選択して、ページの下部にある「イテレータの選択」ドロップ・ダウンに、変数が表示されることを確認します。**「OK」**をクリックします。



6. 構造ウィンドウで、**「バインディング」→「problemDescriptionVar」**を選択します。ID プロパティを、`problemDescriptionVar` から **`problemDescription`** に変更します。
7. 作業内容を保存します。
8. ビジュアル・エディタで、**「SRCreate.jspx」**タブをクリックして、ページに戻ります。problemDescription の **「af:inputText -#{res['srcreate.info.2']}」**フィールドを選択します。プロパティ・インスペクタで、EL 式ピッカーを使用して、Value プロパティを **#{bindings.problemDescription.inputValue}** に設定します。



再利用可能なメソッドの作成

「Cancel」ボタンをクリックすると、（GlobalHome ナビゲーション・ルールを通じて）SRList ページに戻ります。SRList ページに戻る際に、製品リストを初期値にリセットして、problemDescription フィールドをクリアすることもできます。ページは、別のサービス・リクエストを作成できる状態になっています。

1. ボタンを組み込むには、ビジュアル・エディタで、フッターにある最初のボタン（**#res['srdemo.cancel']**）をダブルクリックして、アクションを定義します。**「バインド Action プロパティ」**ペインで、（**cancelButton_action** となる）デフォルトの**メソッド**名を受け入れて、**「OK」**をクリックします。

2. **SRCreate.java** ファイルに移動します。次のコードをイベントに追加します。import 文をファイルの先頭に追加します。

```
import oracle.binding.BindingContainer;
```

3. このコードの最初のセクションは、変数の宣言の後で、クラスの最初のメソッドの前にあります。BindingContainer の変数および BindingContainer の get メソッドと set メソッドをそれぞれ定義します。

```
    private BindingContainer bindings;
    public BindingContainer getBindings() {
        return this.bindings;}
    public void setBindings(BindingContainer bindings) {
        this.bindings = bindings;}
```

4. コードの次の部分が、生成された cancelButton_action() メソッドに取ってかわります。コードの各セクションが行う内容を説明するインライン・コメントがあります。既存のメソッドを次の内容に置き換えます。

```
public String cancelButton_action() {
/*
This action is actually reused from two pages, so we just need to ensure that we
use the correct binding container reference.
*/
DCBindingContainer dcBindings =
(DCBindingContainer)ADFUtils.findBindingContainer(getBindings(),
   "app_SRCreatePageDef");
/*
Reset the product list to the first item:
*/
DCIteratorBinding productsIter =
dcBindings.findIteratorBinding("findAllProductsIter");
productsIter.setCurrentRowIndexInRange(0);
/*
Clean out the description field:
*/
AttributeBinding problem =
    (AttributeBinding)dcBindings.getControlBinding
("problemDescription");
problem.setInputValue(null);
/*
Navigate back to the list page
*/
return "GlobalHome";
}
```

5. **[Alt]** を押しながら **[Enter]** を押して、ファイルにすべての import 文を組み込みます。AttributeBinding パッケージの場合、**oracle.adf.model** をインポートします。

6. 作業内容を保存します。

マネージド Bean へのバインディングの割当て

ページに追加されるデータ・コンポーネントは、手動でデータソースにバインドされました。ここでは、SRCreate マネージド Bean に対するバインディングを作成する必要があります。マネージド Bean を使用すると、UI コンポーネントはデータ・モデルと通信できます。ADF によりマネージド Bean は作成されたので、プロパティを定義して、バインディングを管理し手動でバインドする必要があります。

1. **faces-config.xml** ファイルを開いて、**「概要」**タブをクリックします。
2. **「app_SRCreate」**マネージド Bean を選択します。「管理プロパティ」ペインで、**「新規」**ボタンをクリックします。
3. プロパティに **bindings** と名前を付けて、**「OK」**をクリックします。
4. 「編集」ボタンをクリックして、Value プロパティを **#{bindings}** に設定します。**「OK」**をクリックすると、プロセスが完了します。

確認ページの作成

SRCreateConfirm ページを使用すると、ユーザーは、新規作成したサービス・リクエストの確認、およびサービス・リクエストのデータベースへのコミットを行うことができます。SRCreate ページで、ユーザーが「Continue」ボタンをクリックするときにコールされます。新規のサービス・リクエスト情報が表示され、「Cancel」、「Go Back」、「Submit Request」という 3 つのボタンがあります。「Cancel」をクリックすると、すべての新規サービス・リクエスト・エントリが取り消されて、SRList ページに移動します。「Go Back」をクリックすると、SRCreate ページに戻りますが、仮の新規サービス・リクエスト・エントリが保持されています。

注意： フィールドのラベルが、フィールドに対してバインディングから継承されるのではなく、一般的なリソース・バンドルから取り込まれる場合があります。「#{res[...]}」を含むラベルはすべて、コンポーネントの属性を参照します。

プロセスの作成をサポートするページの開発を続行します。プロセスの 2 番目のページは SRCreateConfirm ページで、先に作成したテンプレートも使用できます。

1. **faces-config.xml** ファイルを開く場合は、**「SRCreateConfirm」**ページをダブルクリックして、JSF ページ・ウィザードを起動します。
2. ウィザードの最初の 3 つのステップに対する値が、次の表の値と一致していることを確認します。

ウィザード・ステップ 1: JSP ファイル

フィールド	値
ファイル名	SRCreateConfirm.jspx
ディレクトリ名	ファイルが保存される場所。最後に **¥app** を付加した値を設定するので、デフォルト値にはすでに含まれています。**¥app** が含まれていない場合は追加します。
タイプ	JSP Document
モバイル	チェック・ボックスの選択を解除。

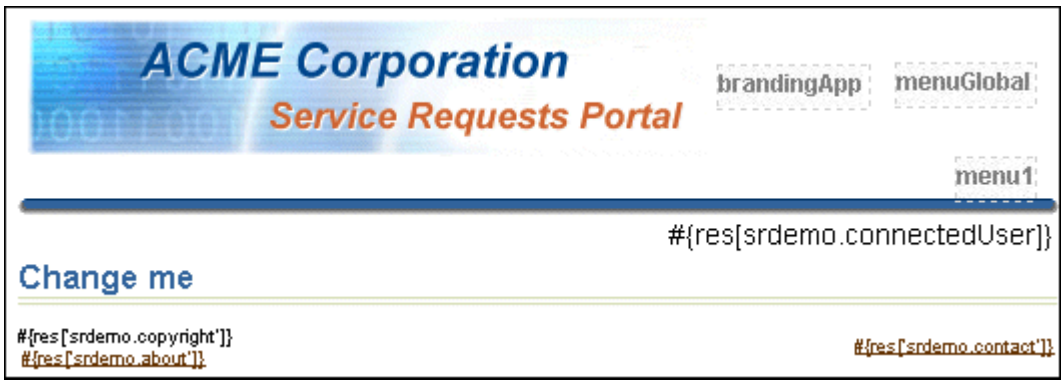
3. 「次へ」をクリックして続行します。

ウィザード・ステップ 2: コンポーネント・バインディング	
フィールド	**値**
新規マネージド Bean での UI コンポーネントの自動公開	このオプションが選択されていることを確認します。
名前	`backing_app_SRCreateConfirm`
クラス	`SRCreateConfirm`
パッケージ	`oracle.srdemo.userinterface.backing.app`

4. 「次へ」をクリックして続行します。

ウィザード・ステップ 3: タグ・ライブラリ	
フィールド	**値**
選択済のライブラリ	`ADF Faces Components`
	`ADF Faces HTML`
	`JSF Core`
	`JSF HTML`

5. **「終了」**をクリックして、ページ詳細を作成します。ビジュアル・エディタに、新規の **SRCreateConfirm** ページが表示されます。
6. まだ開かれていない場合は、**SRDemoTemplate** ファイルを開きます。構造ウィンドウで**「afh:html」**ノードを右クリックして、ショートカット・メニューから「コピー」を選択します。
7. タブをクリックして **SRCreateConfirm** ページに戻り、構造ウィンドウで、**「f:view」**ノードを開きます。
8. **「html」**ノードを削除します。**「f:view」**を右クリックして、ショートカット・メニューから**「貼付け」**を選択します。
9. 構造ウィンドウで、**「afh:html」**ノードをダブルクリックし、**Title** プロパティを **`SRDemo Confirm`** に変更します。
10. **「終了」**をクリックします。ビジュアル・エディタにより、他のページのルック・アンド・フィールで、**SRCreateConfirm** ページが表示されます。



確認ページの変更

この項では、SRCreateConfirm ページを変更して、別のタイトルの表示およびデータ・バインドされたコントロールの組込みを行います。ページには、作成プロセス全体の進行状況を視覚的に表示するプロセス・トレインが組み込まれています。最初のページでは、最初のプロセス・トレインの最初のサークルは表示されますが、2 番目サークルは表示されません。SRCreateConfirm ページでは、両方のサークルが表示されます。最初のいくつかの手順は、プロセス・トレイン機能の有効化に焦点を合せています。

1. 構造ウィンドウで、**「f:view」→「afh.html」→「afh.body」→「h:form」**ノードを開いて、**af:panelPage** を表示します。
2. 構造ウィンドウで、**「af:panelPage」**をダブルクリックして、**Title** プロパティを **#{res['srcreate.pageTitle']}** に設定します。チュートリアルの他の全ページと同様に、この値は実行時に、UIResources プロパティから提供される値に置換されます。

   次に、SRCreate ページに対して実行したように、プロセス・トレインを作成する必要があります。手動でプロセス・トレインを作成するのではなく、SRCreate ページ用に作成した af:processTrain のコピーを使用できます。プロセス・トレインは、両方のページで同じ値を使用します。
3. **「SRCreate」**ページを開いて、**af:panelPage** を表示します。
4. **「af:processTrain」**を選択して右クリックします。**「コピー」**を選択します。
5. **SRCreateConfirm** ページをクリックして、構造ウィンドウで、**af:panelPage** ノードを表示します。
6. **「af:panelPage」**を選択して、コンテキスト・メニューから、**「貼付け」**を選択します。af:processTrain が、af:panelPage の子として追加されます。
7. **「設計」**タブをクリックして、構造ウィンドウで、**ObjectSpacer** を af:panelPage にドラッグします。**af:processTrain** の下に追加されます。高さを **20** に、幅を **<null>** にそれぞれ設定します。

表示の調整

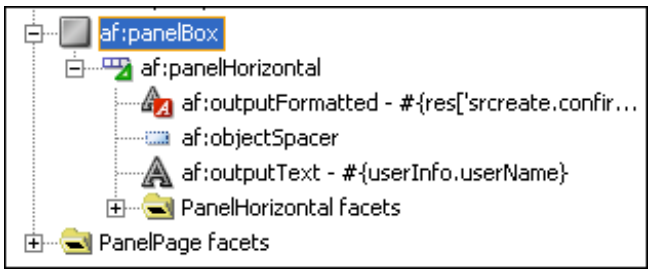
プロセス・トレインは、作成プロセスの進行状況を表示するよう、構成されています。ここで、3 つのデータの値、つまりサービス・リクエスト、製品、問題の説明をログに記録した各ユーザーを表示する必要があります。

次の手順では、確認テキストおよびサービス・リクエストを作成するユーザーに関する情報を追加します。

1. **OutputFormated** コンポーネントを af:panelPage にドラッグします。そうすると、af:objectSpacer の下に表示されます。Value プロパティを **#{res['srcreate.confirmText']}** に設定します。実行時に、製品および問題の値を確認する場合、このテキストが表示されます。
2. 別の **ObjectSpacer** をドラッグして、af:panelPage にドロップします。af:outputFormated コンポーネントの下に表示されます。高さを **20** に、幅を **<null>** にそれぞれ設定します。
3. コンポーネント・パレットから、**PanelBox** を af:panelPage にドロップします。af:objectSpacer の後に表示されます。PanelBox の Width プロパティを **100%** に設定します。
4. **PanelHorizontal** コンポーネントを af:panelBox にドラッグ・アンド・ドロップして、子ノードにします。
5. **OutputFormatted** コンポーネントを af:panelHorizontal にドラッグして、その値を **#{res['srcreate.confirmLine.1']}** に設定します。
6. **ObjectSpacer** コンポーネントを af:panelHorizontal ノードにドラッグします。そうすると、af:outputFormated コンポーネントの下に表示されます。幅を **<null>** に、高さを **20** にそれぞれ設定します。

7. **OutputText** コンポーネントを af:panelHorizontal にドロップします。af:objectSpacer の下に表示されます。この outputText により、サービス・リクエストをログに記録しているユーザーに関する情報が表示されます。このコンポーネントでユーザー名およびユーザー ID を表示します。Value プロパティを **`#{userInfo.userName}`** **`#{userInfo.userId}`** に設定します。

構造ウィンドウは、次のスクリーンショットのように表示されます。これで、リクエストをログに記録したユーザーに関する情報を追加するタスクは完了です。

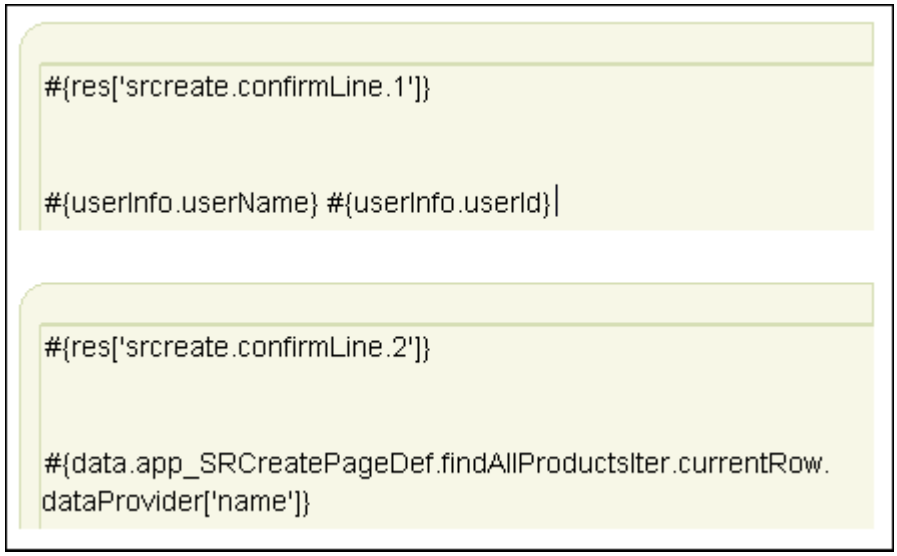


ProductId の表示

製品 ID をページに追加します。構造ウィンドウで、ADF コンポーネントをある場所から別の場所にコピーして貼り付けます。製品 ID を追加する次の手順は、完了済のユーザーの追加手順に類似しています。

1. **ObjectSpacer** コンポーネントを af:panelPage にドロップします。そうすると、af:panelBox コンポーネントの下に表示されます。高さを **`20`** に、幅を **`<null>`** にそれぞれ設定します。
2. 前述の手順で作成した「**af:panelBox**」を選択します。コンテキスト・メニューから、「**コピー**」を選択します。
3. 「**af:panelPage**」を選択して、コンテキスト・メニューから、「**貼付け**」を選択します。新規の af:panelBox が、直前に追加した af:objectsSpacer の下に表示されます。
4. 「**af:panelBox**」→「**af:panelHorizontal**」を開き、次の手順に従って、新規のコンポーネントのプロパティを変更します。
5. af:outputFormatted コンポーネントを **`#{res['srcreate.confirmLine.2']}`** に設定します。
6. af:outputText の Value プロパティを **`#{data.app_SRCreatePageDef.findAllProductsIter.currentRow.dataProvider['name']}`** に設定します。

ページに直前に作成した 2 つの領域は、次のスクリーンショットのように表示されます。



製品の説明の表示

最後に、製品の説明を追加します。ADF コンポーネントを前述の場所の 1 つからコピーして貼り付け、プロパティを更新します。説明を追加する次の手順は、完了済の製品 ID の追加手順に類似しています。

1. **ObjectSpacer** コンポーネントを af:panelPage にドロップします。af:panelBox コンポーネントの下に表示されます。高さを **20** に、幅を **<null>** にそれぞれ設定します。
2. 前述の手順で作成した「**af:panelBox**」を選択します。コンテキスト・メニューから、「**コピー**」を選択します。
3. 「**af:panelPage**」を選択して、コンテキスト・メニューから、「**貼付け**」を選択します。af:panelBox が、直前に追加した af:objectsSpacer の下に表示されます。
4. af:panelHorizontal で、af:outputFormatted コンポーネントを **#{res['srcreate.confirmLine.3']}** に設定します。
5. af:outputText の Value プロパティを **#{data.app_SRCreatePageDef.problemDescription.inputvalue}** に設定します。

6. 高さ **20** で幅 **<null>** の 3 番目の af:panelBox の下に、最後にもう一度 **ObjectSpacer** を追加します。ページに対して、3 つの表示項目を作成しました。ビジュアル・エディタは、次のスクリーンショットのように表示されます。

トランザクションの制御

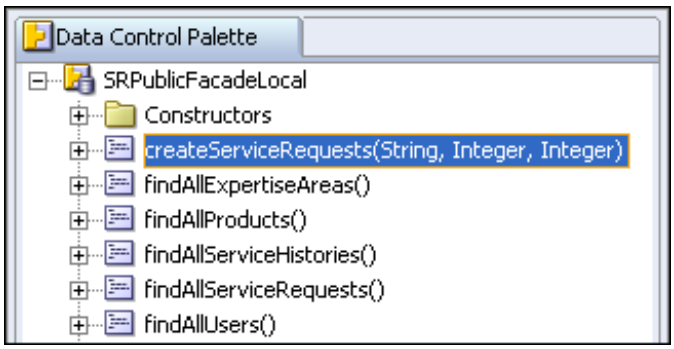
ここでは、サービス・リクエスト、製品および問題の説明をログに記録した各ユーザーに関する情報を表示するコンポーネントを組み込みます。トランザクションを制御するボタンを追加します。

1. 「**SRCreateConfirm.jspx**」タブをクリックして、ページに戻ります。
2. 構造ウィンドウを使用して、**PanelButtonBar** を af:PanelPage にドロップして、これを最後の af:objectSpacer の後に移動します。
3. af:panelButtonBar 内に、2 つの **CommandButton** コンポーネントをドロップします。
4. 最初のボタンで、Text プロパティを **#{res['srdemo.cancel']}** に、Action プロパティを **#{backing_app_SRCreate.cancelButton_action}** に、それぞれ設定します。(SRCreate ページと同じ取消しのアクションにコールを戻す方法に注意してください。)
5. 2 番目のボタンで、Text プロパティを **#{res['srdemo.previousStep']}** に、Action プロパティを **back** に、それぞれ設定します。このボタンにより、作成ページに移動します。

ボタンへのメソッドのバインディング

3 番目のボタンをクリックすると、いくつかのことが実行される必要があります。最初に、バインドされる値を収集し、実行するメソッドのパラメータを移入します。セッション Bean からのメソッドはすべて、データ・コントロール・メソッドから使用可能です。次の手順で、3 番目のボタンの作成およびバインドを行います。

1. 「データ・コントロール・パレット」を開きます。**`createServiceRequests(String,Integer,Integer)`** メソッドを af:panelButtonBar にドラッグします。ポップアップ・メニューで、**「メソッド」→「ADF コマンド・ボタン」**を選択します。

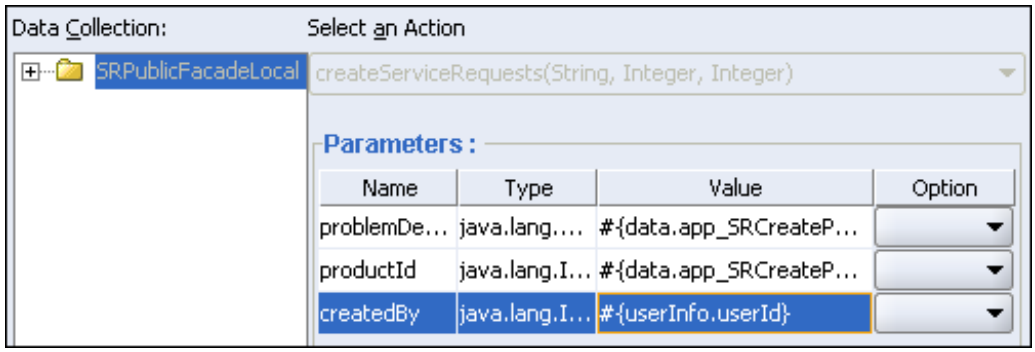


アクション・エディタを使用して、パラメータをソースに割り当てます。製品 ID と説明に対するバインディングの値およびユーザーの値を設定します。

2. 問題の説明は、createServiceRequest メソッドの最初のパラメータを移入する必要があります。problemDescription の値を **`#{data.app_SRCreatePageDef.problemDescription.inputValue}`** に設定します。

3. 2 番目のパラメータは、1 つを選択するリストから選択した製品の製品 ID である必要があります。productId の値に **`#{data.app_SRCreatePageDef.findAllProductsIter.currentRow.dataProvider['prodId']}`** を設定します。EL ピッカーを使用して、「prodId」以外のすべてを追加できます。最後の部分は手動で追加する必要があります。

4. 3 番目の値は、ログインした人のユーザー ID です。createdBy の値を **`#{userInfo.userId}`** に設定します。アクション・エディタは、次のスクリーンショットのように表示されます。

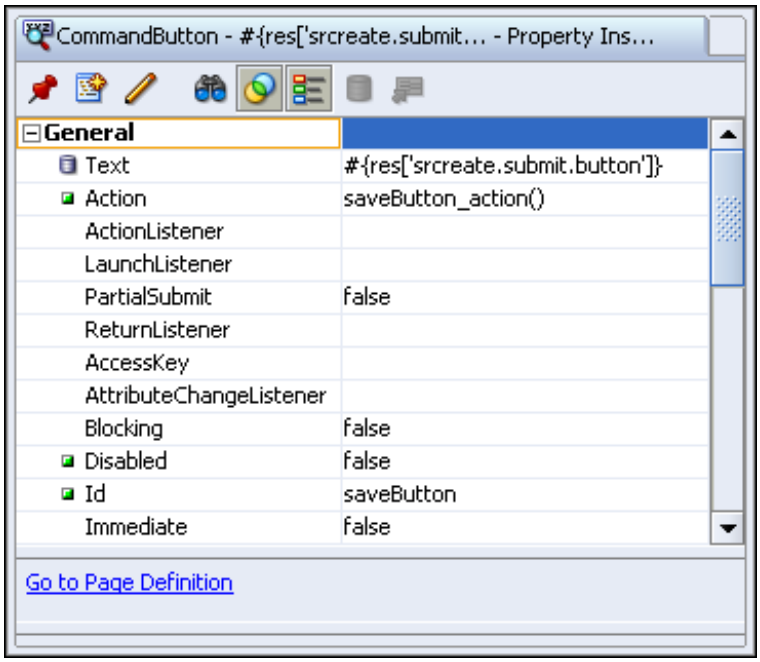


5. **「OK」**をクリックして続行します。

6. 次の表を使用して、3 番目の af:commandButton 用のプロパティ値を設定します。

フィールド	値
Text	`#{res['srcreate.submit.button']}`
Action	`saveButton_action()`
Id	`saveButton`
Disabled	`false`

プロパティ・インスペクタは、次のスクリーンショットのように表示されます。

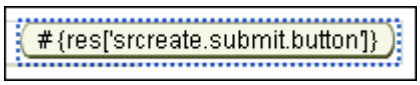


7. 作業内容を保存します。構造ウィンドウには、次のように 3 つのボタンが含まれています。

データのコミットおよびサービス・リクエスト ID のリターン

現在、createServiceRequests メソッドのパラメータはバインドされています。ボタンのクリック時に、2 つのアクションを実行する必要があります。データの値をデータベースにコミットする必要があります。次に、サービス・リクエスト ID を SRDone ページに確認として表示するための属性に戻す必要があります。サービス・リクエスト ID は、ネイティブ・シーケンス・ジェネレータによって生成され、Toplink サービス・リクエスト定義にマッピングされます。次の一連のステップで、コードをボタンに追加して、このアクションを完成します。

1. ビジュアル・エディタで、作成済の 3 番目のボタンをダブルクリックして、バッキング Bean にアクションを定義します。



2. ポップアップする「バインド Action プロパティ」ダイアログ・ボックスで、**メソッド**名のデフォルト値を受け入れます。**「ADF バインディング・コードの生成」**チェック・ボックスが選択されていることを確認します。このチェック・ボックスを選択すると、import 文およびすべての ADF バインディングのゲッターとセッターが追加されます。**「OK」**をクリックして続行します。

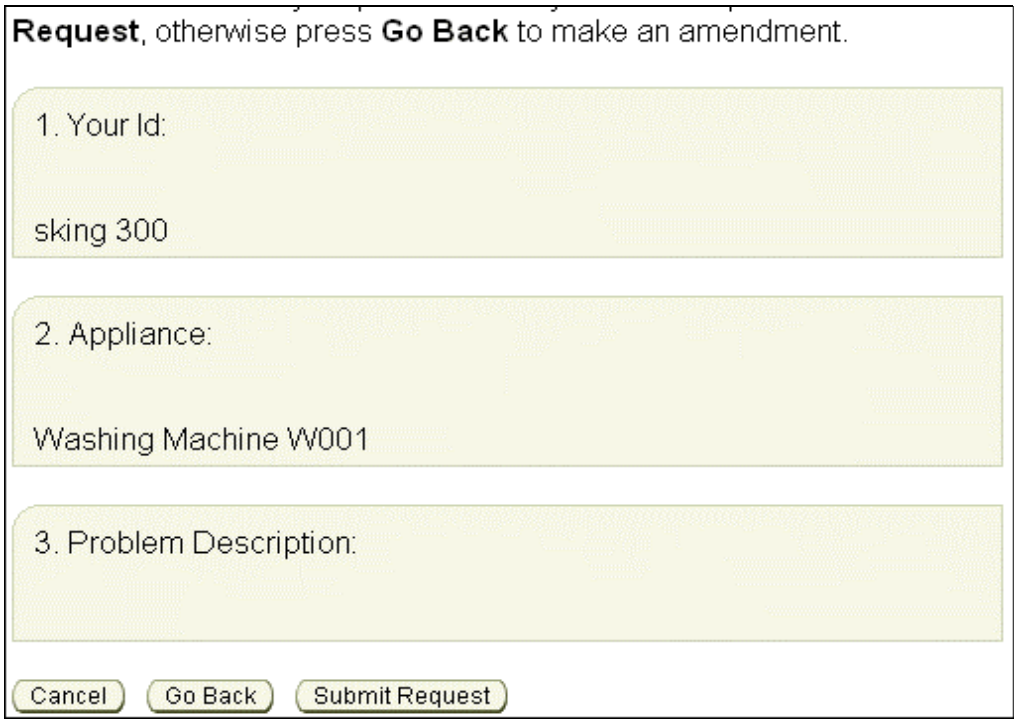
3. **SRCreateConfirm.java** ファイルに移動します。

4. 発生する両方のアクションを取得するには、saveButton_action() メソッドに生成されるコードの一部を次のコードに置き換えます。OperationsBinding 行の下にコードを貼り付けます。

```
OperationBinding operationBinding =
       bindings.getOperationBinding("createServiceRequests");
//Add this code
ServiceRequests result = (ServiceRequests)operationBinding.execute();
Integer svrId = result.getSvrId();
ExternalContext ectx =
     FacesContext.getCurrentInstance().getExternalContext();
HttpServletRequest request = (HttpServletRequest)ectx.getRequest();
request.setAttribute("SRDEMO_CREATED_SVRID",svrId);
return "complete";
}
```

5. **[Alt]** を押しながら **[Enter]** を押して、推奨されるパッケージをすべてインポートします。

6. 作業内容を保存して、ページを実行します。SRCreate ページからコールされずにページが実行されるため、製品はデフォルトでデータベースから取得された最初の製品になり、問題の説明は空白になります。次のスクリーンショットは、ページを表示しています。入力が求められたら、ユーザー名に **`sking`** を使用し、パスワードに **`welcome`** を使用します。



完了ページの作成

SRDone ページでは、技術者によってサービス・リクエストが割り当てられ、処理されていることをユーザーに通知します。ユーザーのレコードに対するサービス・リクエスト ID を表示することもできます。

注意： フィールドのラベルが、フィールドに対してバインディングから継承されるのではなく、一般的なリソース・バンドルから取り込まれる場合があります。

SRCreateDone ページでは、チュートリアルで先に作成したテンプレートを使用することもできます。

1. `faces-config.xml` ファイルで、**「SRCreateDone」**ページをダブルクリックして、JSF ページ・ウィザードを起動します。
2. ウィザードの最初の 3 つのステップに対する値が、次の表の値と一致していることを確認します。

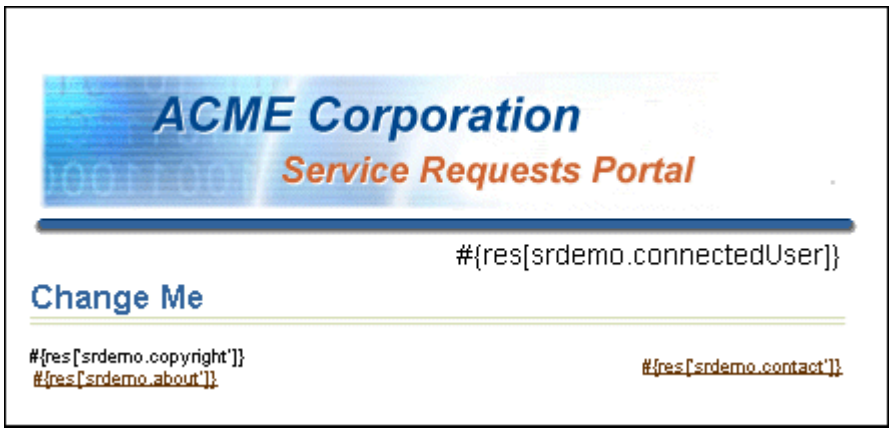
ウィザード・ステップ 1: JSP ファイル	
フィールド	**値**
ファイル名	`SRCreateDone.jspx`
ディレクトリ名	ファイルが保存される場所。最後に `¥app` を付加した値を設定するので、デフォルト値にはすでに含まれています。`¥app` が含まれていない場合は追加します。
タイプ	`JSP Document`
モバイル	チェック・ボックスの選択を解除。

3. 「次へ」をクリックして続行します。

ウィザード・ステップ 3: タグ・ライブラリ	
フィールド	**値**
選択済のライブラリ	`ADF Faces Components`
	`ADF Faces HTML`
	`JSF Core`
	`JSF HTML`

4. **「終了」**をクリックして、ページ詳細を作成します。ビジュアル・エディタに、新規の SRCreateDone ページが表示されます。
5. まだ開かれていない場合は、**SRDemoTemplate** ファイルを開きます。構造ウィンドウで**「afh:html」**ノードを右クリックして、ショートカット・メニューから**「コピー」**を選択します。
6. タブをクリックして SRCreateDone ページに戻り、構造ウィンドウで、**「f:view」**ノードを開きます。
7. **「html」**ノードを削除します。**「f:view」**を右クリックして、ショートカット・メニューから**「貼付け」**を選択します。
8. 構造ウィンドウで、**「afh:html」**ノードをダブルクリックし、Title プロパティを **`SRDemo Done`** に変更します。

9. **「終了」**をクリックします。ビジュアル・エディタにより、他のページのルック・アンド・フィールで、SRCreateDone ページが表示されます。



完了ページの変更

この項では、SRCreateDone ページを変更して、別のタイトルの表示、確認メッセージの組込みおよびホームページに戻るボタンの表示を行います。このページは、新規の SvrId 番号を保持する HTTP リクエストから値を使用しますが、データ・バインドされていません。

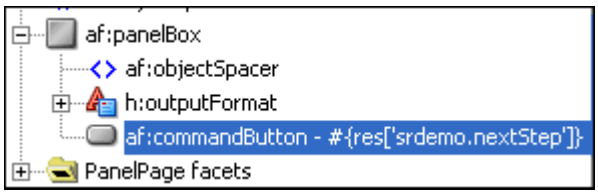
1. 構造ウィンドウで、**「f:view」→「afh.html」→「afh.body」→「h:form」**ノードを開いて、**af:panelPage** を表示します。
2. **「af:panelPage」**をダブルクリックして、**Text** プロパティを **`#{res['srcreate.pageTitle']}`** に変更します。**「OK」**をクリックします。ビジュアル・エディタに、新規のパネル・タイトルが表示されます。
3. 構造ウィンドウで、**ObjectSpacer** を af:panelPage にドラッグします。幅を **`<null>`** に設定します。
4. **PanelBox** をコンポーネント・パレットから af:panelPage にドラッグして、幅を **`100%`** に設定します。
5. **ObjectSpacer** を af:panelBox にドラッグします。
6. コンポーネント・パレットから、**「JSF HTML」→「OutputFormat」**コンポーネントを af:panelBox にドラッグします。

   次の手順で、技術者へのサービス・リクエストの割当てに関するメッセージ、および終了時に移動する場所の定義に関するメッセージを表示します。
7. **「h:outputFormat」**コンポーネントを選択して、**「ソース」**タブをクリックします。既存のコードを次のコードに置き換えます。このコードには、サービス・リクエストを作成したユーザーに対する JSF パラメータが含まれています。

```
<h:outputFormat value="#{res['srcreate.confirmPanel.message']}" escape="false">
<f:param value="#{requestScope.SRDEMO_CREATED_SVRID}"/>
</h:outputFormat>
```

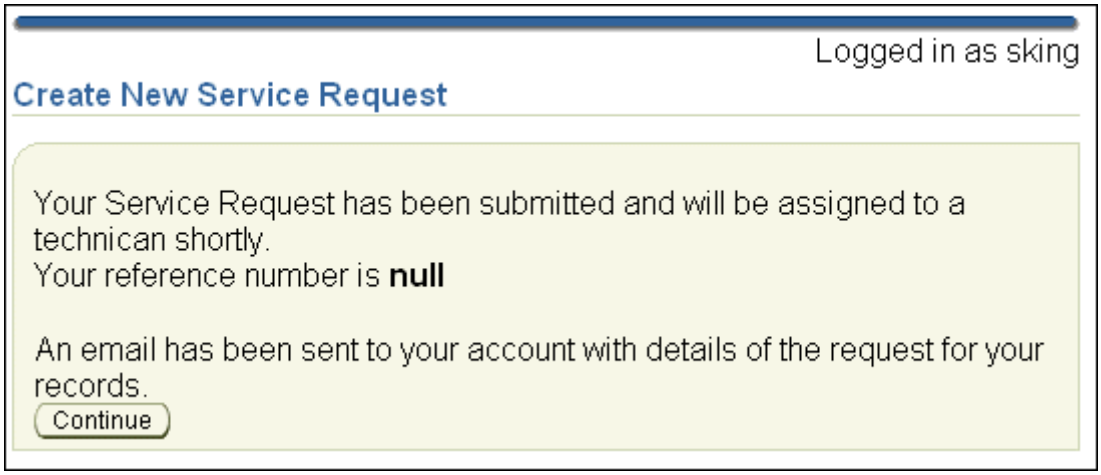
8. **「設計」**タブをクリックして、**「ADF Faces Core」→「CommandButton」**を af:panelBox にドラッグします。**h:outputFormat** コンポーネントの下に表示されます。

9. ボタンのテキストを **`#{res['srdemo.nextStep']}`** に、アクションを **`globalHome`** にそれぞれ設定します。ボタンは次のスクリーンショットのように表示されます。



10. 空白を増やす場合には、**ObjectSpacer** を af:panelPage にドラッグします。af:panelBox の後に表示されます。ビジュアル・エディタで、af:panelBox は次のスクリーンショットのように表示されます。

11. 作業内容を保存して、ページを実行します。ページは最初の 2 ページ以降実行されないので、サービス・リクエスト ID がありません。これは、ページを実行する際に、参照番号を参照しているテキストが NULL を表示しているためです。順序の 3 番目としてページを実行すると、ある値のサービス・リクエスト ID が表示されます。ページは、次のスクリーンショットのように表示されます。入力が求められたら、ユーザー名に **`sking`** を使用し、パスワードに **`welcome`** を使用します。



FAQ ページの作成

この項では、SRCreate ページから起動可能な SRFaq ページを開発します。このページには、ユーザーの問題解決に役立つ FAQ が表示されます。テキストの値は UIResources.properties ファイルから取得されます。

1. faces-config.xml ファイルで、**「SRFaq」**ページをダブルクリックして、JSF ページ・ウィザードを起動します。値が、ウィザードの最初のステップに対して適切である必要があります。**「次へ」**をクリックします。
2. ウィザードの 2 番目のステップで、**「マネージド Bean で UI コンポーネントを自動公開しない」**ラジオ・ボタンを選択します。このページで実行されるのは静的テキストを表示する機能のみなので、バッキング Bean は必要ありません。
3. **「終了」**をクリックします。ページが作成されます。テンプレートおよびテキスト参照の追加のみ必要です。
4. まだ開かれていない場合は、**SRDemoTemplate** ファイルを開きます。構造ウィンドウで**「afh:html」**ノードを右クリックして、ショートカット・メニューから**「コピー」**を選択します。
5. タブをクリックして SRFaq ページに戻り、構造ウィンドウで、**「f:view」**ノードを開きます。
6. **「html」**ノードを削除します。**「f:view」**を右クリックして、ショートカット・メニューから**「貼付け」**を選択します。
7. 構造ウィンドウで、**「afh:html」**ノードをダブルクリックし、Title プロパティを **SRDemo FAQ** に変更します。
8. **「終了」**をクリックします。ビジュアル・エディタにより、他のページのルック・アンド・フィールで、SRFaq ページが表示されます。

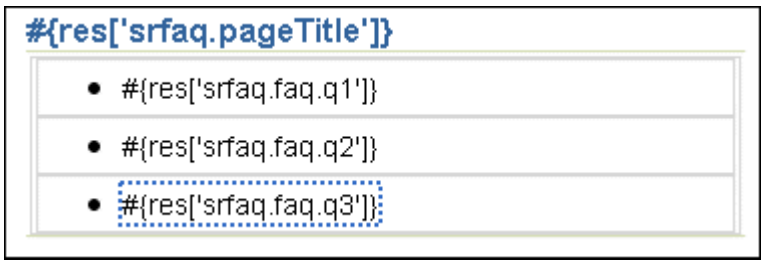
FAQ ページの変更

この項では、SRFaq ページを変更して、3 つの一般的なサービス・リクエスト問題の説明とそれぞれの解決策を表示します。テキストは UIResources.properties ファイルから取得され、その場で変更できます。このページには、テキストにより、3 つのサービス・リクエストの問題とそれぞれの解決策が表示されます。

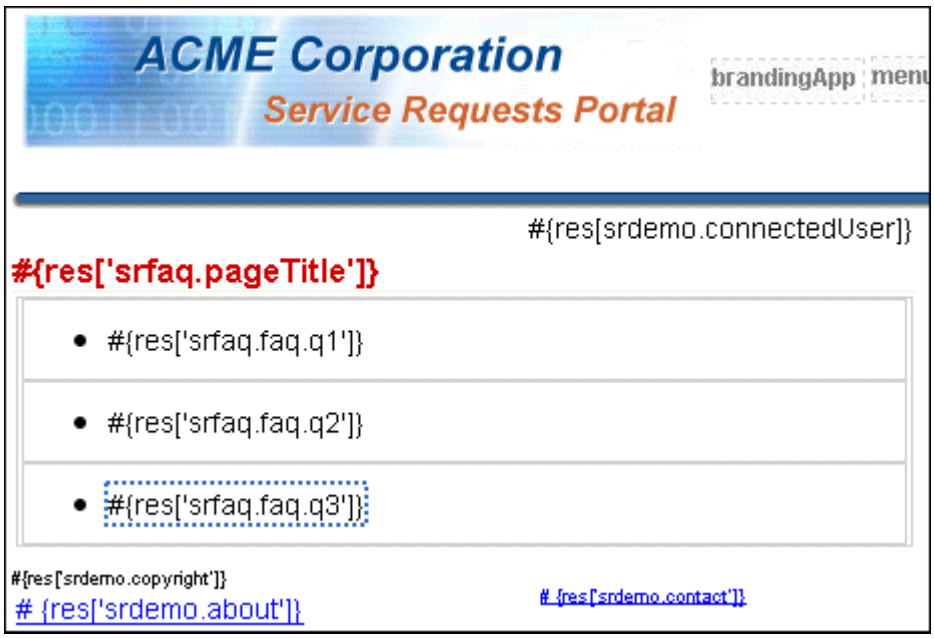
1. 構造ウィンドウで、**「f:view」**→**「afh.html」**→**「afh.body」**→**「h:form」**ノードを開いて、**af:panelPage** を表示します。
2. **「af:panelPage」**をダブルクリックして、**Text** プロパティを **#{res['srfaq.pageTitle']}** に変更します。**「OK」**をクリックします。ビジュアル・エディタに、新規のパネル・タイトルが表示されます。
3. 構造ウィンドウで、**「ADF Faces Core」**→**「PanelGroup」**を **af:panelPage** 内にドラッグします。
4. **af:panelGroup** 内に、3 つの**「ADF Faces Core」**→**「PanelLists」**を追加します。このパネル・リストは、表示するテキスト用のプレースホルダとして使用されます。
5. 各 **af:panelList** コンポーネントで、**「ADF Faces Core」**→**「OutputFormatted」**コンポーネントを追加します。各 **Value** プロパティを次の表の値に設定します。この各フィールドは、UIResources.properties ファイルに存在するサービス・リクエストの解決策に対応しています。

出力フォーマット済項目	値
1 番目	#{res['srfaq.faq.q1']}
2 番目	#{res['srfaq.faq.q2']}
3 番目	#{res['srfaq.faq.q3']}

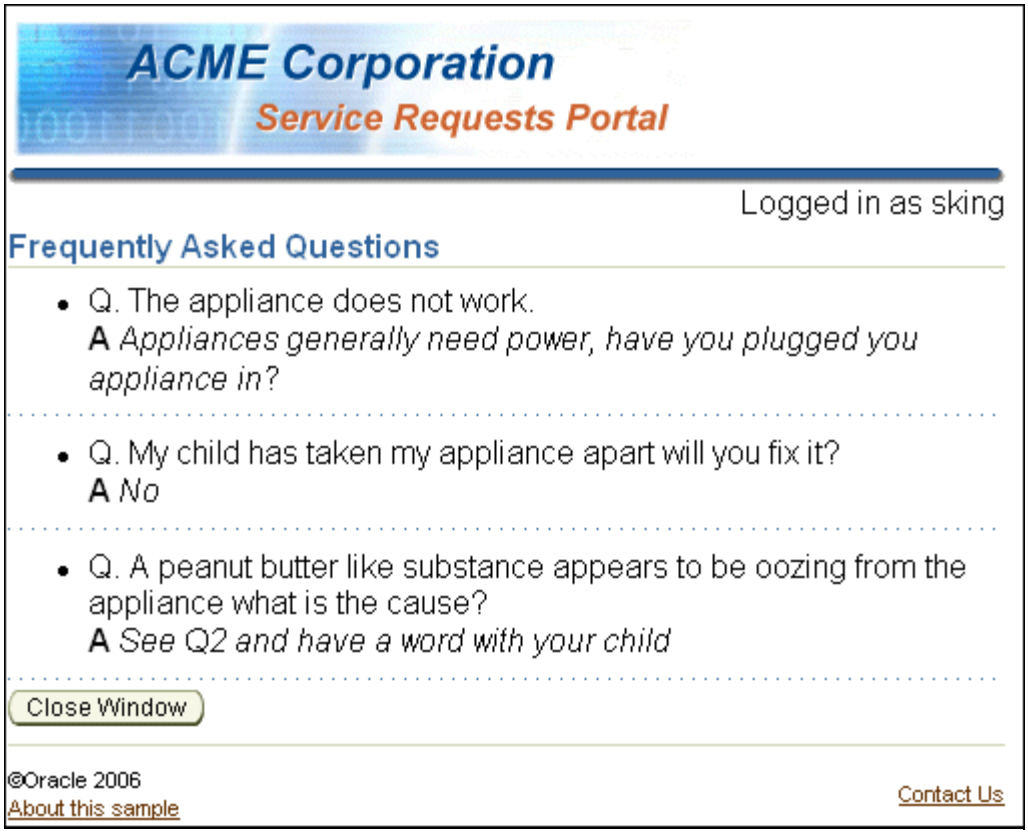
直前に追加されたページ・コンポーネントにより、ページは、次のスクリーンショットのように表示されます。



6. 「**af:panelGroup**」→「**PanelGroup Facet**」→「**セパレータ**」ノードを開いて、「セパレータ」ノード内に **ObjectSeperator** を追加します。この操作により、パネルに余白を作成するために、各 af:panelist コンポーネント間にセパレータが追加されます。終了したページは、次のスクリーンショットのように表示されます。



7. 作業内容を保存して、ページを実行します。ページは、次のスクリーンショットのように表示されます。入力が求められたら、ユーザー名に **sking** を使用し、パスワードに **welcome** を使用します。

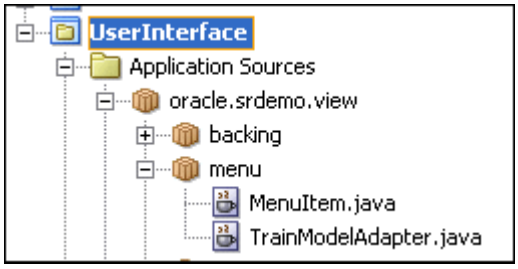


プロセス・トレインの組込み

SRCreate ページおよび SRCreateConfirm ページの両方に、作成プロセス全体の進行状況を表示するプロセス・トレインを追加しました。ページの開発中に、プロセス・トレイン用の表示を作成しました。この項では、2 つのクラスを作成し、次に 4 つのマネージド Bean を作成して、それらをプロセス・トレインにリンクします。

1. アプリケーション・ナビゲータで、「**UserInterface**」→「**アプリケーション・ソース**」ノードを開きます。
2. 「**oracle.srdemo.view**」ノードを選択して、コンテキスト・メニューから「**新規**」を選択します。
3. 新規ギャラリで、「**General**」を選択して、「**Java クラス**」を選択します。「**OK**」をクリックします。
4. クラスに **MenuItem** と名前を付けて、パッケージを **oracle.srdemo.view.menu** に設定します。次に「**OK**」をクリックします。
5. Windows Explorer で、セットアップ・ファイルを解凍した場所に移動します。**MenuItem.txt** ファイルを開いて、ファイルのすべてを直前に作成した新規クラスにコピーします。
6. 2 番目の Java クラスを作成し、**TrainModelAdapter** と名前を付けて、パッケージ値を **oracle.srdemo.view.menu** に設定します。
7. Windows Explorer で、セットアップ・ファイルを解凍した場所に移動します。**TrainModelAdapter.txt** ファイルを開いて、ファイルのすべてを直前に作成した新規クラスにコピーします。

8. アプリケーション・ナビゲータは、次のスクリーンショットのように表示されます。



9. アプリケーション・ナビゲータで、**「faces-config.xml」**ファイルをダブルクリックして、**「概要」**タブをクリックします。

10. 次の表の値を使用して、プロセス・トレインの 4 つのマネージド Bean を作成します。それぞれ Scope プロパティを **application** に設定し、**クラスが存在しない場合は作成**チェック・ボックスの選択を解除します。

名前	クラス
createTrain_Step1	`oracle.srdemo.view.menu.MenuItem`
createTrain_Step2	`oracle.srdemo.view.menu.MenuItem`
createTrainNodes	`java.util.ArrayList`
createTrainMenuModel	`oracle.srdemo.view.menu.TrainModelAdapter`

各 Bean は別の機能を実行します。2 つの createTrain_Step Bean は、項目の表示方法を示す単一の Bean に結び付きます。createTrainMenuModel は、プロセス・トレインのノードの保持を行うクラスに結び付きます。

各マネージド Bean には、現在のプロセス・ステップに関する情報が含まれています。ここには、ページ名と現在のプロセス・ステップが含まれます。次の手順で、管理プロパティを作成し、値を割り当てます。

11. 3 つのマネージド Bean に対して管理プロパティを作成します。次の表および「faces-config.xml」→「概要」ページを使用して、この手順を完了します。管理プロパティの場合はすべて Class プロパティを **<null>** に設定したままにします。

表にリストされた管理プロパティを作成した後に、各プロパティをダブルクリックして、その値を表の 2 列目の値に設定します。

12. **「createTrain_Step1」**マネージド Bean を選択して、次の管理プロパティを作成します。

管理プロパティ	値
label	`#{resources['srcreate.train.step1']}`
viewId	`/app/SRCreate.jspx`
outcome	`GlobalCreate`

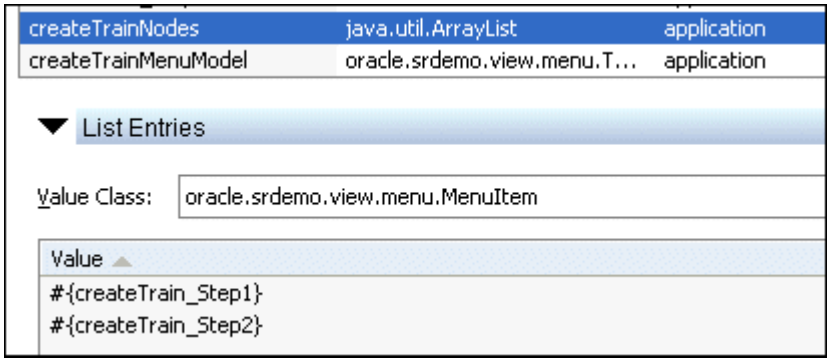
13. **「createTrain_Step2」**マネージド Bean を選択して、次の管理プロパティを作成します。

管理プロパティ	値
label	`#{resources['srcreate.train.step2']}`
viewId	`/app/SRCreateConfirm.jspx`
outcome	`Confirm`

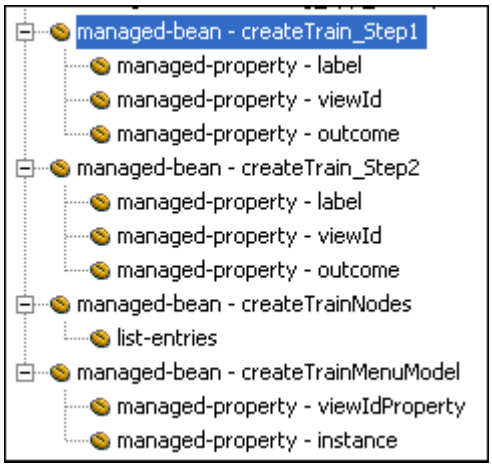
14. 「**CreateTrainMenuModel**」マネージド Bean を選択して、次の管理プロパティを作成します。

管理プロパティ	値
viewIdProperty	`viewId`
instance	`#{createTrainNodes}`

15. 残りの「**createTrainNodes**」マネージド Bean を選択します。「リスト・エントリ」領域で、「値クラス」を **`oracle.srdemo.view.menu.MenuItem`** に設定し、2 つの新しい値 **`#{createTrain_Step1}`** および **`#{createTrain_Step2}`** を追加します。完了すると、領域は次のスクリーンショットのように表示されます。



4 つのマネージド Bean の作成がすべて完了すると、構造ウィンドウは次のスクリーンショットのように表示されます。



ページの実行

ページが完成したので、ページを実行して機能を確認できます。ここでは、次の点に注意します。

- SRCreate ページを使用すると、製品の選択および問題の定義を行うことができます。
- プロセス・トレインは、プロセスの進行状況に応じてハイライトされます。
- 製品名および問題の説明の値は、SRCreateConfirm ページに引き継がれます。
- ユーザー ID およびユーザー名は、ログイン名から取得されて表示されます。
- 「Submit Request」ボタンにより、SRDone ページに移動して、新規のサービス・リクエスト ID を表示します。
- 「Continue」ボタンにより、グローバル・ホーム（ここでは SRList ページ）に移動します。
- 新規のサービス・リクエスト・レコードがリストに表示されます。

まとめ

この章では、JavaServer Faces および ADF コンポーネントを使用して検索ページを作成しました。これを実現するために、次の主なタスクを実行しました。

- サービス・リクエストを指定する作成ページの開発
- サービス・リクエストの値を表示する確認ページの作成
- 新規のサービス・リクエストを表示する完了ページの作成
- サービス・リクエストの解決策を表示する FAQ ページの作成

10

編集ページの開発

この章では、SREdit ページを作成する方法について説明します。SREdit ページは SRDemo アプリケーションのページで、このページを使用すると、マネージャおよび技術者はサービス・リクエストを編集できます。

この章の内容は次のとおりです。

- 概要
- ページ・アウトラインの作成
- ページへの UI 要素の追加
- createdBy 名および assignedTo 名を取得するルックアップの作成
- 入力サービス・リクエスト ID パラメータの組込み
- status 属性用のドロップダウン・リストの追加
- 「Cancel」ボタンの組込み
- 「Save」ボタンの組込み
- クローズドのサービス・リクエストに対する入力フィールドの無効化
- ページの実行
- アプリケーションのルック・アンド・フィールの変更
- まとめ

概要

SREdit ページは、顧客ではなく企業のスタッフによる使用を目的とする 3 つの画面の 1 つです。マネージャおよび技術者は、このページを使用してサービス・リクエストの情報を更新します。

注意： 第 9 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **`Chapter10`** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、**`<jdev_install>¥jdev¥mywork¥Chapter10`** になります。
2. **`<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter9.zip`** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「`<jdev_install>¥jdev¥mywork¥Chapter10¥SRDemo¥SRDemo.jws`」**を選択します。この章の初期アプリケーションが開きます。

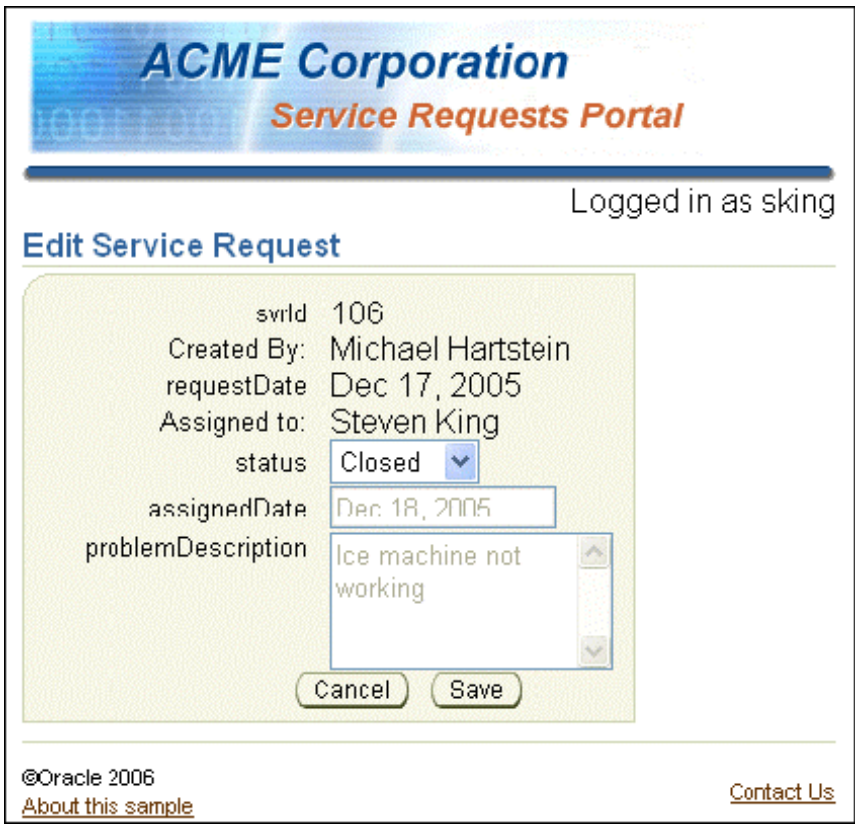
第 9 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

アプリケーション・ナビゲータで「`faces-config.xml`」ファイルをダブルクリックして、ページ・フロー・ダイアグラムに再度移動し、SREdit ページと他のページとの関連を調査します。

ここでは、SREdit ページに関して次の主な点に注意します。

- このページは、アプリケーション内の 3 つの異なる場所、つまり SRList ページ、SRSearch ページおよび SRMain からコールできます。異なるコンテキストからページを再利用するためにパラメータ化されています。コール側のページは、問い合せるサービス・リクエストおよび編集完了後に戻る場所について、SREdit ページに通知するパラメータを設定することを求められます。
- このページを使用すると、マネージャまたは技術者はサービス・リクエストを編集できます。
- ユーザーはリクエストのステータスを変更できます。
- リクエストに対して、別のユーザーまたはマネージャを割り当てられます。
- 「Problem Description」フィールドおよび「Assigned Date」フィールドは、ステータスが「Closed」のリクエストに対して無効化されます。

次のスクリーンショットは、終了した SREdit ページが（マネージャのログインから）どのように表示されるかを示しています。



前述のリストで説明した機能およびルック・アンド・フィールを備えた SREdit ページを作成するには、次の主なタスクを実行します。

- （第 4 章で作成したテンプレート・ページをベースにした）ページ・アウトラインの作成
- ページへの UI 要素の追加
- `createdBy` 名および `assignedTo` 名を取得するルックアップの作成
- 入力サービス・リクエスト ID パラメータの組込み
- `status` 属性用のドロップダウン・リストの追加
- 「Cancel」ボタンの組込み
- 「Save」ボタンの組込み
- クローズドのサービス・リクエストに対する入力フィールドの無効化

ページ・アウトラインの作成

次の手順を実行して SREdit ページを作成し、テンプレートを追加して該当するルック・アンド・フィールを適用します。

1. 開いていない場合は、**「faces-config.xml」**ファイルをダブルクリックして、**ページ・フロー・ダイアグラム**を表示します。
2. **「SREdit」**ページをダブルクリックして、JSF ページ・ウィザードを起動します。
3. 次の表の値を使用して、ウィザードの最初の 3 つのステップを完了します。

ウィザード・ステップ 1: JSP ファイル	
フィールド	**値**
ファイル名	SREdit.jspx
ディレクトリ名	ファイルが保存される場所。 ¥SRDemo¥UserInterface¥public_html¥app¥staff フォルダにページを作成してください。
タイプ	JSP Document
モバイル	チェック・ボックスの選択を解除。

4. **「次へ」**をクリックします。

ウィザード・ステップ 2: コンポーネント・バインディング	
フィールド	**値**
新規マネージド Bean での UI コンポーネントの自動公開	このラジオ・ボタンが選択されていることを確認します。
名前	backing_app_staff_SREdit
クラス	SREdit
パッケージ	oracle.srdemo.view.backing.app.staff

5. **「次へ」**をクリックします。

ウィザード・ステップ 3: タグ・ライブラリ	
フィールド	**値**
選択済のライブラリ	ADF Faces Components
	ADF Faces HTML
	JSF Core
	JSF HTML

6. **「終了」**をクリックして、ページ詳細を作成します。ビジュアル・エディタに、新規の SREdit ページが表示されます。
7. まだ開かれていない場合は、**SRDemoTemplate** ファイルを開きます。構造ウィンドウで、**「afh:html」**ノードを閉じて選択します。コンテキスト・メニューから、**「コピー」**を選択します。
8. タブをクリックして、SREdit ページに戻ります。構造ウィンドウで、**「f:view」**ノードを開きます。
9. **「html」**ノードを削除します。**「f:view」**を右クリックして、コンテキスト・メニューから**「貼付け」**を選択します。

10. メイン・イメージが表示されない場合は、パスを変更する必要があります。イメージを選択し、プロパティ・インスペクタで URL プロパティを **/images/SRBranding** にリセットします。

    先に作成したルック・アンド・フィールが新しいページに適用されます。

ページへの UI 要素の追加

次の手順を実行して、ページへのユーザー・インタフェース要素の追加を開始します。`findServiceRequestById` メソッドからデータ・コンポーネントをすべて組み込み、保存および取消しアクションを制御するコマンド・ボタンを追加する必要があります。

1. 次のようにタイトルをページに追加します。ビジュアル・エディタで「panelPage」をクリックして選択します。（構造ウィンドウで、**「af:panelPage」**を選択することもできます。）プロパティ・インスペクタで、Title プロパティに **#{res['sredit.pageTitle']}** と入力します。

   または、このチュートリアルの先の章で、これまで数回行ったように、「PanelPage のプロパティ」ダイアログを起動し、Title プロパティで**「バインド」**をクリックしてノードから**「sredit.pageTitle」**を選択し、これを「式」ペインに移動できます。

   ページのタイトル名は `UIResources.properties` ファイルに定義されたリソースから取得されます。

2. ページの実行時に、ブラウザ・タイトルに表示されるヘッダーを次のように追加します。構造ウィンドウで**「afh:head」**を選択します。Title プロパティを **#{res['srdemo.browserTitle']}** に設定します。

3. コンポーネント・パレットで、**ADF Faces Core** ページを選択します。リストをスクロールして**「PanelBox」**を見つけて、これを panelPage にドラッグします。ビジュアル・エディタで、タイトルの下に表示されます。panelBox にはページのコンテンツ領域が定義されます。

   次の手順で、データ・コンポーネントの作成、フィールド表示のサイズ変更および 2 つのコマンド・ボタンの組込みを行います。

4. データをページに追加します。「データ・コントロール・パレット」を選択して、**「SRPublicFacadeLocal」**を開きます。下にスクロールして**「findServiceRequestById(Integer)」**→**「ServiceRequests」**を選択し、これを panelBox 内のページにドラッグします。

5. ポップアップ・メニューで、**「フォーム」**→**「ADF フォーム」**を選択します。表示される「フォーム・フィールドの編集」ダイアログ・ボックスで、列を **srvId**、**requestDate**、**status**、**assignedDate**、**problemDescription** の順に並べ替えます。ダイアログ・ボックスはまだ閉じないでください。

6. ダイアログ・ボックスで、svrId の右側にある**「使用するコンポーネント」**フィールドをクリックして、ドロップダウン・リストから**「ラベル付 ADF 出力テキスト」**を選択します。**「requestDate」**フィールドも同様に処理します。**「OK」**をクリックします。

7. アクション・バインディング・エディタが表示されます。ダイアログ・ボックスで、**findSvrId** をここでは空のままにしておきます。後からページ・パラメータをこれに割り当てます。**「OK」**をクリックします。

   ページにフォームが表示されます。次の手順で詳細を追加します。

8. **「problemDescription」**フィールドを選択し、プロパティ・インスペクタで **Rows** プロパティを **4** に設定して、マルチライン項目を作成します。これにより、ユーザーは電化製品の問題に関するテキストの説明を入力できます。

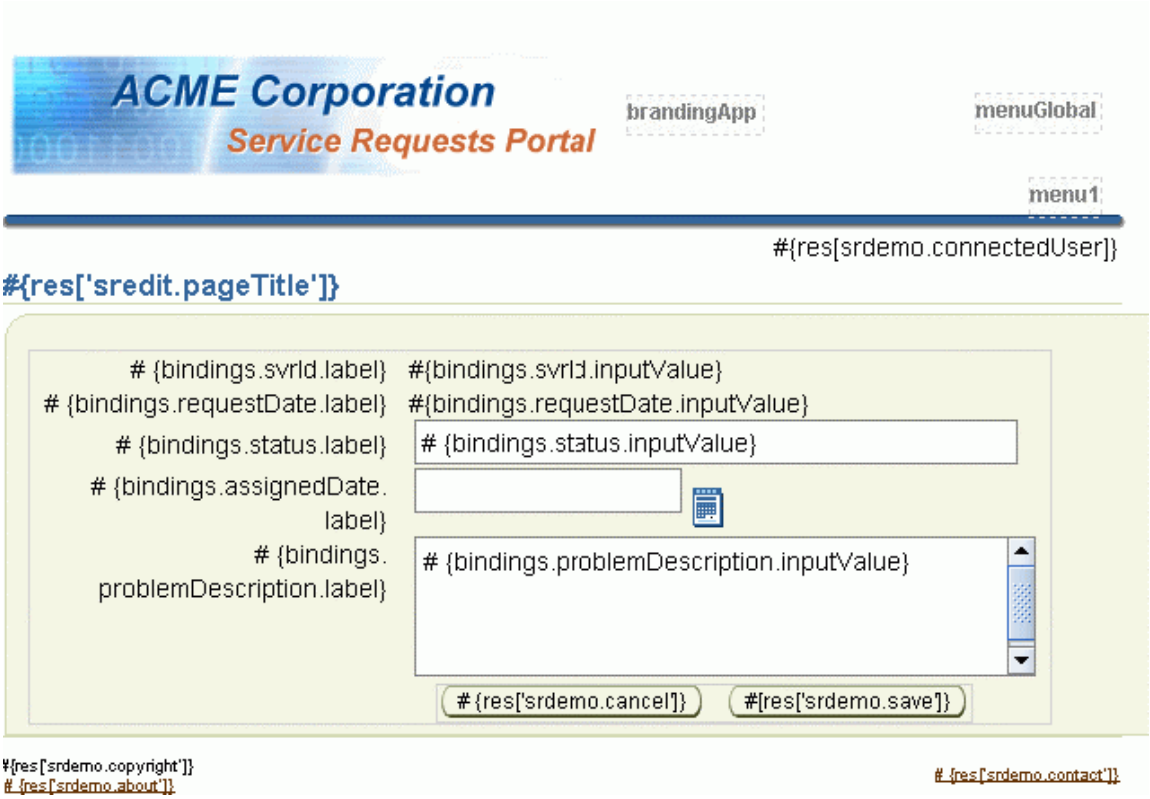
9. 「PanelForm」ファセット・ノードを開きます。ADF Faces Core コンポーネント・パレットで**「panelButtonBar」**を選択して、これを構造ウィンドウの PanelForm の「footer」ファセットにドラッグします。

   次のように、ページの「Save」ボタンと「Cancel」ボタンを作成します。

10. ADF Faces Core コンポーネント・パレットから、2 つの **CommandButtons** を panelButtonBar にドラッグします。プロパティ・インスペクタで、各プロパティを次の表の値に設定します。

ID	テキスト
cancelButton	`#{res['srdemo.cancel']}`
commitButton	`#{res['srdemo.save']}`

ここで、ページは次のスクリーンショットのように表示されます。



createdBy 名および assignedTo 名を取得するルックアップの作成

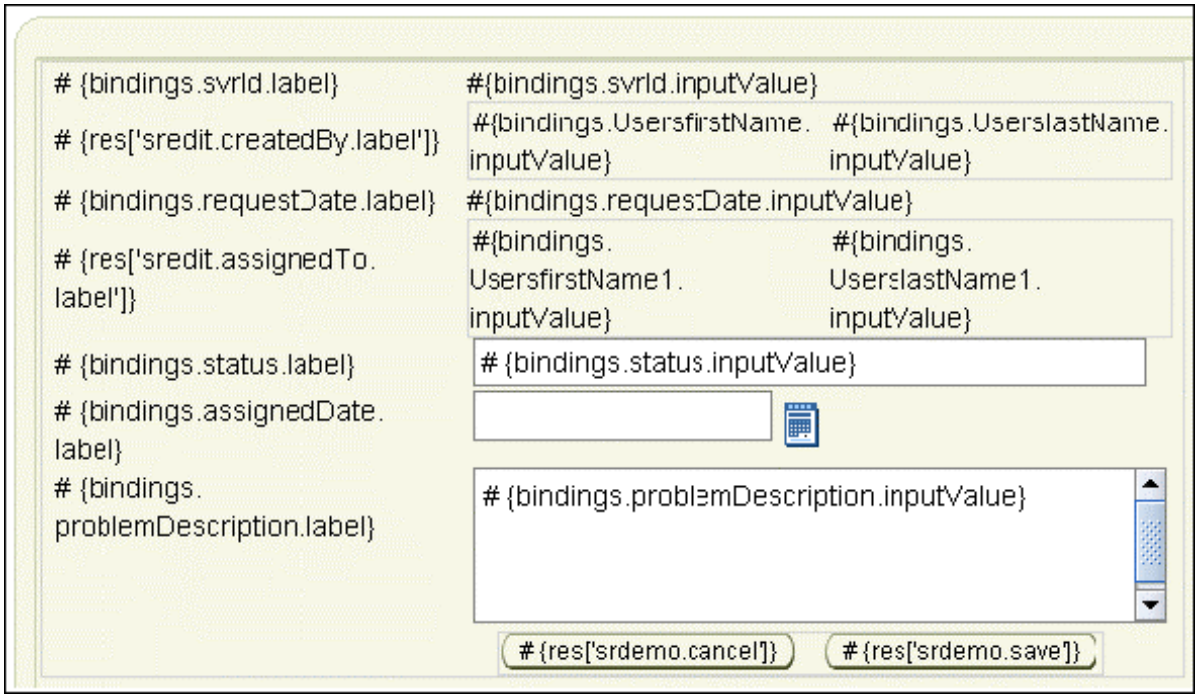
createdBy 属性および assignedTo 属性により、ユーザーは、サービス・リクエストの作成者の名前またはサービス・リクエストが割り当てられた人の名前を取得できます。

ServiceRequest オブジェクトの createdBy 属性および assignedTo 属性は、実際には子オブジェクト・アクセッサです。次の手順で、この子オブジェクトから relevant name 属性をページにバインドします。

1. 最初のコンポーネントには、サービス・リクエストを作成したユーザーの姓と名が含まれています。ADF Faces Core コンポーネント・パレットで**「PanelLabelAndMessage」**を選択して、ビジュアル・エディタで、これを svrId フィールドの下のページにドラッグします。
2. 2 番目のコンポーネントには、サービス・リクエストが割り当てられた技術者の姓と名が含まれています。2 番目の **PanelLabelAndMessage** コンポーネントを requestDate フィールドの下にドラッグ・アンド・ドロップします。
3. **PanelHorizontal** を各 af:panelLabelAndMessage コンポーネント内にドロップします。firstName 属性および lastName 属性が垂直方向ではなく横に並べて表示されることを確認します。

   次の手順で、作成したユーザーおよび割り当てられたユーザーの姓と名に対するデータ・コンポーネントを追加し、これをユーザー・インタフェースにバインドします。
4. データ・コントロール・パレットで、**「findServiceRequestById」→「ServiceRequests」→「createdBy」**ノードを開いて、**firstName** 属性を選択します。これを最初の panelHorizontals にドラッグして、ADF 出力テキストとしてドロップします。
5. **lastName** 属性を使用して前述の手順を繰り返し、同じ af:panelHorizontal に追加します。
6. ADF Faces Core コンポーネント・パレットで**「ObjectSpacer」**を選択します。これを構造ウィンドウの「PanelHorizontal」の「セパレータ」ファセットにドラッグ・アンド・ドロップします（必要に応じて、「PanelHorizontal」ファセット・ノードを開きます）。Width プロパティを **4** に設定します。
7. データ・コントロール・パレットで、**「findServiceRequestById」→「ServiceRequests」→「assignedTo」**ノードを開いて、**firstName** 属性を選択します。これを 2 番目の panelHorizontals にドラッグして、ADF 出力テキストとしてドロップします。
8. **lastName** 属性を使用して前述の手順を繰り返し、af:panelHorizontal コンポーネントに置きます。
9. 先に実行したように、**ObjectSpacer** をこの**「PanelHorizontal」**の「セパレータ」ファセットに追加します。再度、幅を **4** に設定します。
10. 最初の**「panelLabelAndMessage」**コンポーネントを選択します。Label プロパティを **#{res['sredit.createdBy.label']}** に設定します。
11. 2 番目の panelLabelAndMessage コンポーネントの Label プロパティを **#{res['sredit.assignedTo.label']}** に設定します。

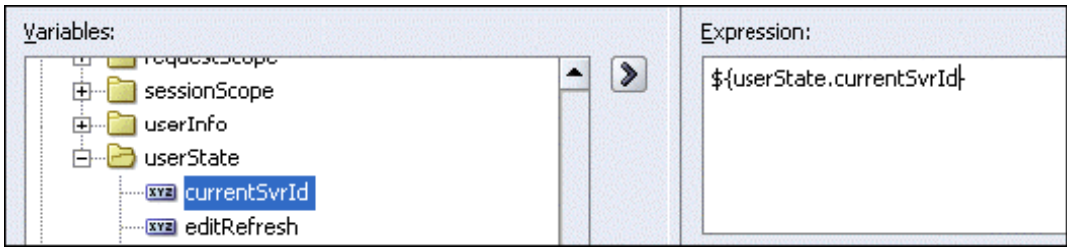
12. ページを保存します。ここで、ページは次のスクリーンショットのように表示されます。



入力サービス・リクエスト ID パラメータの組込み

次の手順では、ユーザーがページに入力する際に、SREdit ページに表示される必要があるサービス・リクエストの識別および組込みを行います。

1. ビジュアル・エディタで右クリックして、コンテキスト・メニューから、**「ページ定義に移動」**を選択します。
2. 構造ウィンドウで、**「バインディング」**ノードを開いて、次に**「findServiceRequestById」**ノードを開きます。メソッドの引数を表示する**「findSvrId」**をダブルクリックします。
3. 「NamedData のプロパティ」ダイアログ・ボックスで、**「...」**ボタンをクリックして Advanced Editor に移動します。「変数」リストで、**「JSF Managed Beans」**ノード、次に**「userState」**ノードを開いて、下へスクロールして**「currentSvrId」**を見つけます。
4. **「currentSvrId」**を選択し、「>」矢印をクリックして「式」ボックスに移動します。**「OK」**をクリックし、再度**「OK」**をクリックします。ページ定義ファイルを保存します。

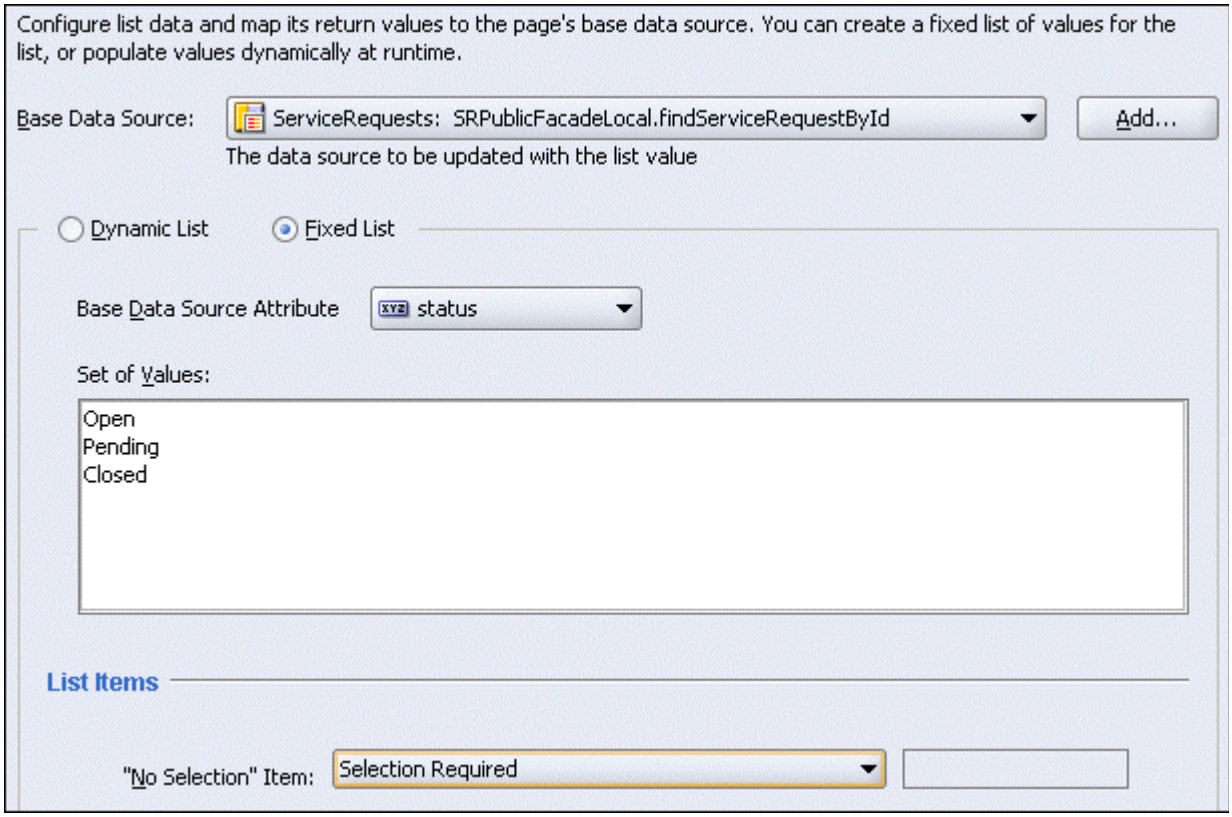


status 属性用のドロップダウン・リストの追加

status フィールドでは、ユーザーが各種ステータスのドロップダウン・リストを使用できるようにする必要があります。次の手順を実行して、現在の status フィールドをリストに変換します。

1. 「**SREdit.jspz**」タブをクリックして、ビジュアル・エディタで、既存の status フィールドのラベルと入力テキストの両方をページから削除します。
2. データ・コントロール・パレットで、「**findServiceRequestById**」→「**ServiceRequests**」を開いて、**status** 属性を選択します。これをページの元の場所にドラッグして、「単一選択」→「ADF 選択肢を 1 つ選択」としてドロップします。

   selectOneChoice コンポーネントでは、ユーザーが項目リストから単一の値を選択できる、メニュー形式のコンポーネントが作成されます。
3. リスト・バインディング・エディタで、「**固定リスト**」オプションを選択します。「ベース・データソース属性」を **status** に設定します。
4. 「一連の値」ボックスで、新規の行にそれぞれ **Open**、**Pending**、**Closed** と入力します。この値が実行時に表示されます。
5. 選択なし項目フィールドを「**選択必須**」に設定します。「**OK**」をクリックします。



6. プロパティ・インスペクタで、新規の af:selectOneChoice に対して、新規リスト・コンポーネントの ID プロパティを **statusSelection** に設定します。
7. ページを保存します。

「Cancel」ボタンの組込み

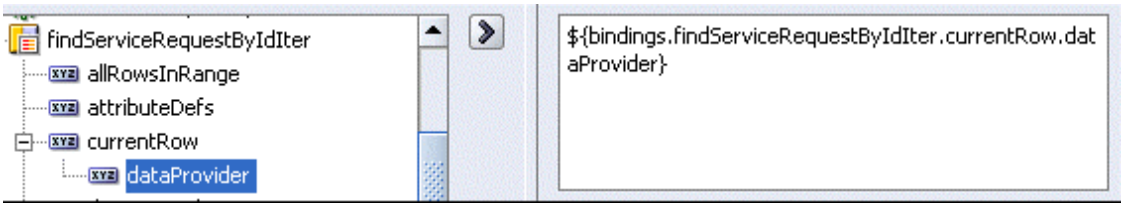
ユーザーが編集画面で「Cancel」ボタンをクリックすると、変更内容を保存せずに破棄します。変更を行う前の場所に、ユーザーを戻す必要があります。これを行うには、コール側のページからパラメータを渡します。また、ユーザーがページに戻って同じサービス・リクエストを表示する場合は、問合せが再実行されることを確認します。再実行されないと、編集前の状態が表示されます。

1. ビジュアル・エディタで右クリックして、コンテキスト・メニューから、**「ページ定義に移動」**を選択します。
2. 構造ウィンドウで、**「executables」**ノードを右クリックして、**「executables の中に挿入」**→**「invokeAction」**を選択します。
3. **「invokeAction の挿入」**ダイアログ・ボックスで **id** に **`explicitRefresh`** と入力して、**「バインド」**フィールドで、ドロップダウン・リストから**「findServiceRequestById」**を選択します。ダイアログ・ボックスを終了しないでください。
4. **「アドバンスト・プロパティ」**タブをクリックして、Refresh プロパティが **IfNeeded** に設定されていることを確認します。これは、リフレッシュを行う必要があるかどうかを制御します。
5. RefreshCondition プロパティで**「...」**をクリックします。「変数」ペインで、**「JSF Managed Beans」**ノードおよび**「userState」**ノードを開きます。
6. **「refresh」**に移動して選択し、**「>」**をクリックして**「式」**ペインに移動します。**「OK」**をクリックしてエディタを閉じて、再度**「OK」**をクリックしてダイアログ・ボックスを終了します。
7. ページ定義ファイルを保存します。
8. ユーザーが「Cancel」ボタンをクリックしたときに、actionListener を fire に設定して、リフレッシュをトリガーします。
   SREdit ページで**「Cancel」**ボタンを右クリックして、コンテキスト・メニューから、**「af:commandButton の中に挿入」**→**「ADF Faces Core」**→**「SetActionListener」**を選択します。
9. 「SetActionListener の挿入」ダイアログ・ボックスで、From* フィールドに **`#{true}`** と入力し、To* フィールドに **`#{userState.refresh}`** と入力します。起動された場合、このリスナーにより #{true} の値が user state refresh プロパティに移入されます。**「OK」**をクリックします。

「Save」ボタンの組込み

編集されたレコードをデータベースに保存する必要があります。次の手順では、サービス・ファサードで汎用 `mergeEntity` メソッドを使用してこれを実行する方法を示します。

1. データ・コントロール・パレットで、**`mergeEntity(Object)`** メソッドを見つけます。これをビジュアル・エディタにドラッグして、この章で先に作成した「Save」ボタンにドロップします。ポップアップ・メニューから、**「既存の CommandButton のバインド」**を選択します。
2. **アクション・バインディング・エディタ**で、**「値」**フィールドでクリックして、表示される**「...」**をクリックします。**「変数」**ダイアログ・ボックスで、**「ADF バインディング」**ノード、次に**「バインディング」**ノードを開きます。**「findServiceRequestById Iter」**ノードを見つけて開き、次に**「currentRow」**ノードを開きます。**「dataProvider」**を選択して、「式」ペインに移動します。**「OK」**をクリックし、再度**「OK」**をクリックします。



3. プロパティ・インスペクタで、「Save」ボタンに対して、Action プロパティを **`#{userState.retrieveReturnNavigationRule}`** に設定します。このメソッドは、変更内容が保存された後に戻るページを識別します。
4. 前述の項と同様に、「Save」ボタンがクリックされたときに、actionListener を fire に設定して、リフレッシュをトリガーします。「Save」ボタンを右クリックして、コンテキスト・メニューから、**「af:commandButton の中に挿入」**→**「ADF Faces Core」**→**「SetActionListener」**を選択します。
5. 「SetActionListener の挿入」ダイアログ・ボックスで、From* フィールドに **`#{true}`** と入力し、To* フィールドに **`#{userState.refresh}`** と入力します。**「OK」**をクリックします。
6. ページを保存します。

クローズドのサービス・リクエストに対する入力フィールドの無効化

サービス・リクエストのステータスがクローズドの場合、problemDescription フィールドおよび assignedDate フィールドが無効化される必要があります。次の手順にこの方法を示します。

1. ステータス・リストを選択して、AutoSubmit プロパティおよび Immediate プロパティを **`true`** に設定します。

   status 属性に対して AutoSubmit を true に設定すると、status 属性の更新時にフォームの一部が連動します。

2. problemDescription フィールドと assignedDate フィールドの両方に対して、Disabled プロパティを **`#{backing_app_staff_SREdit.statusSelection.value=='2'}`** に設定します。

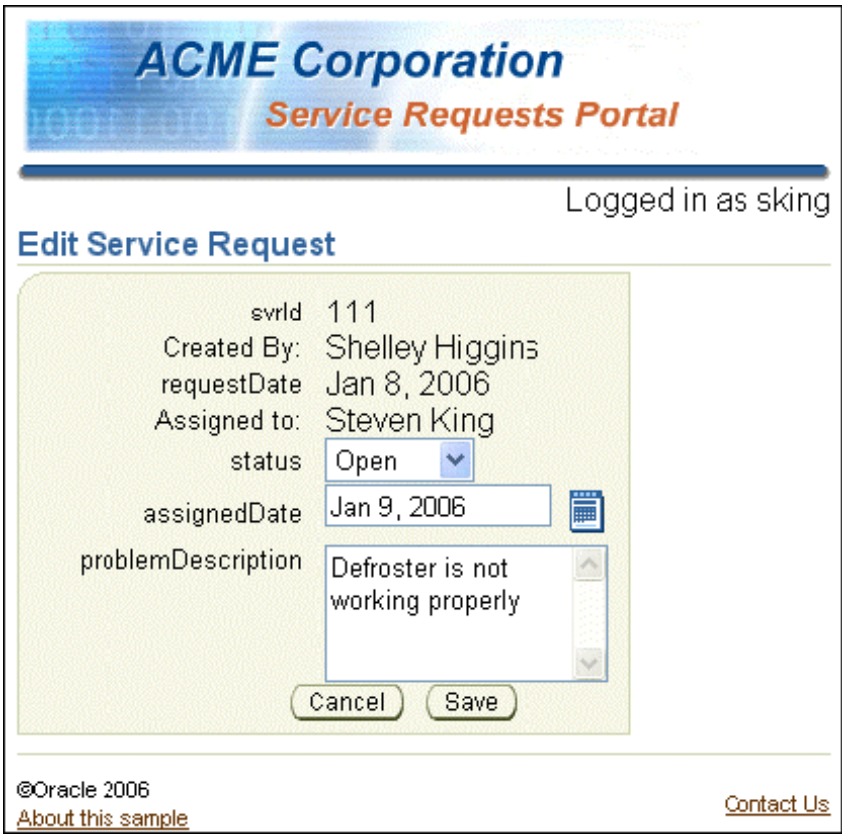
   プロパティ・インスペクタで **Disabled** プロパティをクリックして、ツールバーの**「データにバインド」**アイコンをクリックします。「Disabled」ダイアログ・ボックスで、**「JSF マネージド Bean」**ノード、次に**「backing_app_staff_SREdit」**ノードを開きます。下にスクロールして**「statusSelection」**を見つけて開きます。リストをスクロールして、**「value」**を選択します。これを「式」ペインに移動します。「式」ペインで、値 **2** を追加するよう式を編集します。**「OK」**をクリックします。

リスト・バインディング・エディタで作成した列挙リストはゼロ・インデックス・ベースであり、Closed は 3 番目のエントリなので、値は 2 になります。

3. 2 つのフィールドに対して PartialTriggers プロパティを **`statusSelection`** に設定します。先の場合と同様に、status 属性の値が変更されると、フィールドがリフレッシュされます。
4. ページを保存します。

ページの実行

SREdit ページは SRList ページから編集するレコードを受け取ります。SRList を実行します。入力が求められたら、ユーザー名に **`sking`** を使用し、パスワードに **`welcome`** を使用します。**`id111`** のサービス・リクエストを選択して、「**Edit**」ボタンをクリックします。SREdit ページに、レコードが編集できる状態で表示されます。ページは、次のスクリーンショットのように表示されます。



1. ページの「Status」フィールドでドロップダウン・リストのテストを行います。
2. 日付ピッカー・フィールドが正常に動作することを確認します。
3. リクエストのステータスを「Closed」に設定すると、「Problem Description」フィールドと「Assigned Date」フィールドが無効化されることを確認します。
4. 「**Save**」ボタンをクリックして、List ページに戻ることを確認します。
5. 変更を行い「**Cancel**」をクリックします。変更が行われていない必要があります。

アプリケーションのルック・アンド・フィールの変更

スキンを変更して、アプリケーションのルック・アンド・フィールを変更できます。スキンとは、アプリケーション全体に対して 1 箇所でのみ設定する必要があるグローバルなスタイル・シートのことです。アプリケーション開発者は、実行時に取得されるスキンにコードおよび変更を追加する必要はありません。スキンは Cascading Style Sheet 仕様に基づいています。

デフォルトでは、ADF Faces アプリケーションは Oracle skin を使用します。これは、SRDemo アプリケーションで作成したページに適用されるルック・アンド・フィールです。ADF Faces では、他の 2 つのスキン、つまり最小スキン（一部の基本の書式設定を含む）とシンプル・スキン（特別な書式設定をほとんど含まない）も提供されています。独自のアプリケーション向けに、特別にカスタム・スキンを作成することもできます。作成方法の詳細は JDeveloper の Help ページを参照してください。

次の手順を実行して、最小スキンを SRDemo アプリケーションに適用します。

1. ナビゲータで、「**UserInterface**」→「**WEB-INF**」→「**adf-faces-config.xml**」ファイルを見つけて、これをダブルクリックして開きます。
2. <skin-family> の値を、**<skin-family>oracle</skin-family>** から **`<skin-family>minimal</skin-family>`** に変更します。
3. ファイルを保存します。
4. 新規のルック・アンド・フィールを表示するには、**`bernst`** としてアプリケーションを実行します。**SRList** ページを開きます。

   ビジュアル・エディタのコンポーネントでは、新規のスタイルが表示されます。
5. **SRList** ページを実行します。最小スキンが適用されたページは、次のスクリーンショットのように表示されます。ボタンが四角になり、メニュー・バーが簡略化され、ラベルの色が緑になります。

まとめ

この章では、マネージャおよび技術者がサービス・リクエストを変更できる編集ページを作成しました。これを実現するために、次の主なタスクを実行しました。

- 第 4 章で定義したテンプレート・ページをベースにしたフレームワーク・ページの作成
- サービス・リクエスト情報を表示するページへの UI 要素の追加
- `createdBy` 名および `assignedTo` 名用のルックアップの作成
- `status` 属性用のドロップダウン・リストの追加
- 「Save」ボタンおよび「Cancel」ボタンに対する特別な動作の定義
- ステータスがクローズドのリクエストに対する入力フィールドの無効化
- 別のスキンの適用によるアプリケーションの外観の変更

11

Oracle Application Server 10*g* へのアプリケーションのデプロイ

この章では、JDeveloper を使用して、アプリケーションおよび必要なデプロイメント・ディスクリプタを含む、デプロイ可能な J2EE Web アーカイブを作成します。JDeveloper のデプロイ機能を使用して、アプリケーションを Oracle Application Server 10*g* にデプロイします。Oracle Enterprise Manager を使用して、アプリケーションのテストおよびパフォーマンスの表示を行うことができます。

この章の内容は次のとおりです。

- 概要
- Oracle Application Server 10g への接続の作成
- J2EE（OC4J）インスタンスの OracleAS コンテナの起動
- OC4J への接続の作成
- Enterprise Manager の起動
- デプロイメントの準備
- デプロイメント・プロファイルの作成
- 接続プールおよびデータソースの作成
- アプリケーションのデプロイ
- アプリケーションのテスト
- Enterprise Manager を使用したアプリケーション・サーバーの参照
- まとめ

概要

J2EE アプリケーションのデプロイは、アプリケーション開発の最終ステップです。アプリケーションが完成し計画どおりに動作したら、次のステップは、アプリケーションを顧客が使用できる場所にデプロイすることです。

JDeveloper には、アプリケーションを多くのアプリケーション・サーバーにデプロイするビルトイン機能があります。この章では、アプリケーションを Oracle Application Server 10*g* にデプロイします。

この章では、次の主なタスクを実行します。

- **既存の Oracle Application Server 10*g* への接続の作成 :** JDeveloper は、ターゲット・アプリケーション・サーバーへの接続を確立する必要があります。これにより、正確なデプロイメント・プロファイルの作成、および完成したファイルのサーバーへの送信が可能になります。
- **デプロイメント・プロファイルの作成 :** このプロファイルは、ターゲット・アプリケーション・サーバーへのデプロイメントに必要なファイルの内容およびタイプを管理します。
- **Enterprise Manager（EM）の起動 :** EM の場合、アプリケーションを監視または再デプロイもできます。

  注意 : Oracle Application Server 10*g* にアクセスしない場合は、OC4J インスタンス（JDeveloper 内からは実行されない OC4J のインスタンス）を起動します。
- **アプリケーションのデプロイ :** プロファイルが作成されると、JDeveloper 内からアプリケーションをデプロイできます。
- **デプロイのテスト :** アプリケーションがデプロイされると、Web ブラウザから実行してアプリケーションが期待どおりに動作することを確認できます。
- **Enterprise Manager を使用したアプリケーション・サーバーの参照 :** Enterprise Manager を使用すると、アプリケーション・サーバーの全コンポーネントの詳細な表示が可能です。アプリケーション・パラメータの監視と変更、およびアプリケーション・サーバーのパフォーマンスの微調整を行うことができます。

注意 : 第 10 章を正常に完了しなかった場合は、チュートリアル設定に含まれる章終了時のアプリケーションを使用できます。

1. **`Chapter11`** というサブディレクトリを作成して、初期アプリケーションを保存します。デフォルト設定を使用した場合は、**`<jdev_install>¥jdev¥mywork¥Chapter11`** になります。
2. **`<tutorial_install>¥starterApplications¥SRDemo-EndOfChapter10.zip`** をこの新規ディレクトリに解凍します。別の新規ディレクトリを使用して、この初期アプリケーションとこれまでの作業を別々に保存します。
3. JDeveloper で、使用している SRDemo アプリケーション・ワークスペースを閉じます。
4. **「ファイル」→「開く」**を選択して、**「`<jdev_install>¥jdev¥mywork¥Chapter11¥SRDemo¥SRDemo.jws`」**を選択します。この章の初期アプリケーションが開きます。

第 10 章のステップの内容をすべて実装したアプリケーションを使用して、このチュートリアルを続行できます。

Oracle Application Server 10*g* への接続の作成

JDeveloper は、アプリケーション・サーバー接続を経由した、様々な本番アプリケーション・サーバーへのアプリケーションのデプロイをサポートします。JDeveloper を使用してアプリケーションをデプロイする最初のステップは、ターゲット・アプリケーション・サーバーへの接続を作成することです。

1. JDeveloper で**「接続」**タブをクリックします。

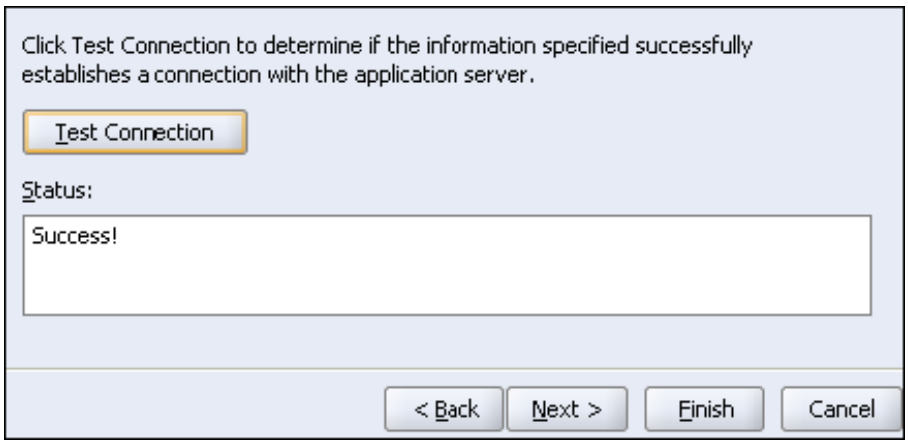
   「接続」タブが表示されていない場合は、JDeveloper のメニュー・バーから**「表示」→「接続ナビゲータ」**を選択できます。([Ctrl] を押しながら、[Shift] と [O] を押すこともできます。)

2. 接続ウィンドウで**「アプリケーション・サーバー」**を右クリックして、コンテキスト・メニューから**「アプリケーション・サーバー接続の作成」**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で次の値を入力します。

フィールド	値
接続名	`OracleAS10g`
接続タイプ	`Oracle Application Server 10g 10.1.3`
ユーザー名	`oc4jadmin`
パスワード	インスタンスの管理者パスワードを入力。
ホスト名	`localhost`

4. ウィザードの最後のページで**「接続のテスト」**をクリックします。成功のメッセージが表示されます。



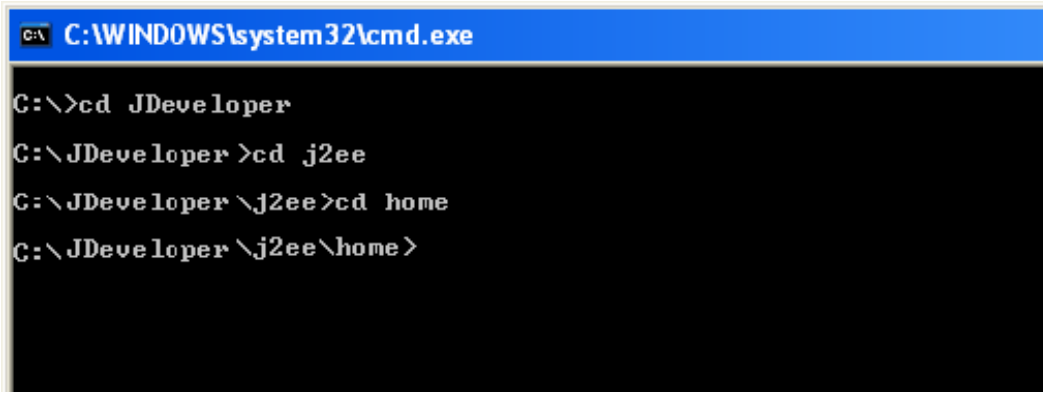
5. テストが成功したら、**「終了」**をクリックして接続を作成します。

注意： Oracle Application Server 10*g* へのアクセスを作成していない場合は、次の項に従って OC4J のインスタンスおよび JDeveloper の接続を作成します。

J2EE（OC4J）インスタンスの OracleAS コンテナの起動

JDeveloper には OC4J のインストールが含まれています。この項では、oc4j.jar ファイルが保存されているディレクトリに移動し、OC4J を起動します。

1. 次のように Windows のコマンド・ウィンドウを開きます。Windows のスタート・バーの**「スタート」**をクリックして、**「ファイル名を指定して実行」**を選択します。
2. ダイアログ・ボックスに **cmd** と入力して、**「OK」**をクリックします。
3. コマンド・ウィンドウで、カレント・ディレクトリを **<jdev_install>¥j2ee¥home** に変更します。



4. 次のコマンドを入力して、JDK を使用して OC4J を起動します。
   <jdk_install>¥jdk¥bin¥java -jar oc4j.jar

5. 初めて OC4J を実行する場合は、自動的にインストールされ、管理者アカウントのパスワードの入力を求められます。管理者アカウントは oc4jadmin です。パスワード **admin** を入力します。インストールでは、パスワードを 2 回入力して、パスワードを確認するよう求められます。
6. インストールおよび起動が完了すると、次のメッセージが表示されます。
   `Oracle Containers for J2EE 10g <10.1.3.0.0> initialized`

   OC4J は稼働中で、使用できる状態になっています。

OC4J への接続の作成

アプリケーション・サーバーへの接続を作成する一般的なステップは、基本的には同じです。違いが発生するのは、サーバーの特別な接続要件がある場合です。Oracle Application Server 10*g* への接続の作成で発生する相違点は、基本的に OC4J への接続の作成と同じです。指定する引数のいくつかの違いのみです。

1. JDeveloper で**「接続」**タブをクリックします。

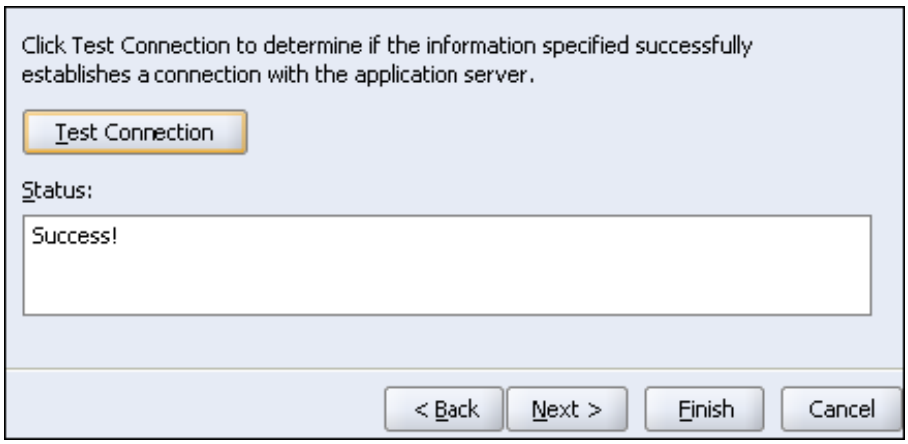
   「接続」タブが表示されていない場合は、JDeveloper のメニュー・バーから**「表示」→「接続ナビゲータ」**を選択できます。(**[Ctrl]** を押しながら、**[Shift]** と **[O]** を押すこともできます。)

2. 接続ウィンドウで**「アプリケーション・サーバー」**を右クリックして、コンテキスト・メニューから**「アプリケーション・サーバー接続の作成」**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で次の値を入力します。

フィールド	値
接続名	`OC4J`
接続タイプ	`Standalone OC4J 10g 10.1.3`
ユーザー名	`oc4jadmin`
パスワード	`admin`
ホスト名	`localhost`

4. ウィザードの最後のページで**「接続のテスト」**をクリックします。成功のメッセージが表示されます。



5. テストが成功したら、**「終了」**をクリックして接続を作成します。

Enterprise Manager の起動

Oracle Application Server 10*g* には、ブラウザ・ベースの Enterprise Manager（EM）が同梱されています。この EM のインタフェースを通じて、アクティビティおよびアプリケーション・サーバーにデプロイしたアプリケーションを監視できます。アプリケーション・サーバーの稼働後に、ブラウザを使用して EM に接続できます。次のステップにより、このインタフェースをオープンして、アプリケーション・サーバーを簡単に参照します。

JDeveloper を同梱しているスタンドアロン OC4J にも Enterprise Manager が含まれています。

1. ご使用のブラウザ（Firefox、Internet Explorer または別のブラウザ）を開いて、次のアドレス（`http://127.0.0.1:7777/em`）を入力します。スタンドアロン OC4J を使用する場合、アドレスは `http://127.0.0.1:8888/em` になります。
2. Enterprise Manager 10*g* では、ユーザー名およびパスワードを求められます。ユーザー名は **`oc4jadmin`** で、パスワードに **`admin`** のパスワード（または管理者のパスワード）を使用します。「ログイン」をクリックして、EM と入力します。
3. ログインに成功すると、ブラウザは次のように表示されます。

アプリケーション、Web サービスおよびその他の Oracle Application Server 10*g* のコンポーネントを参照できます。

デプロイメントの準備

アプリケーションのすべての要素を編成して、きちんと分類しておくには、アプリケーションのすべてのデプロイメント・コンポーネントを保持するプロジェクトを作成します。このプロジェクトには、デプロイメント・プロファイルおよびデプロイメント・ファイル（.ear、.war、.jar）が保持されます。

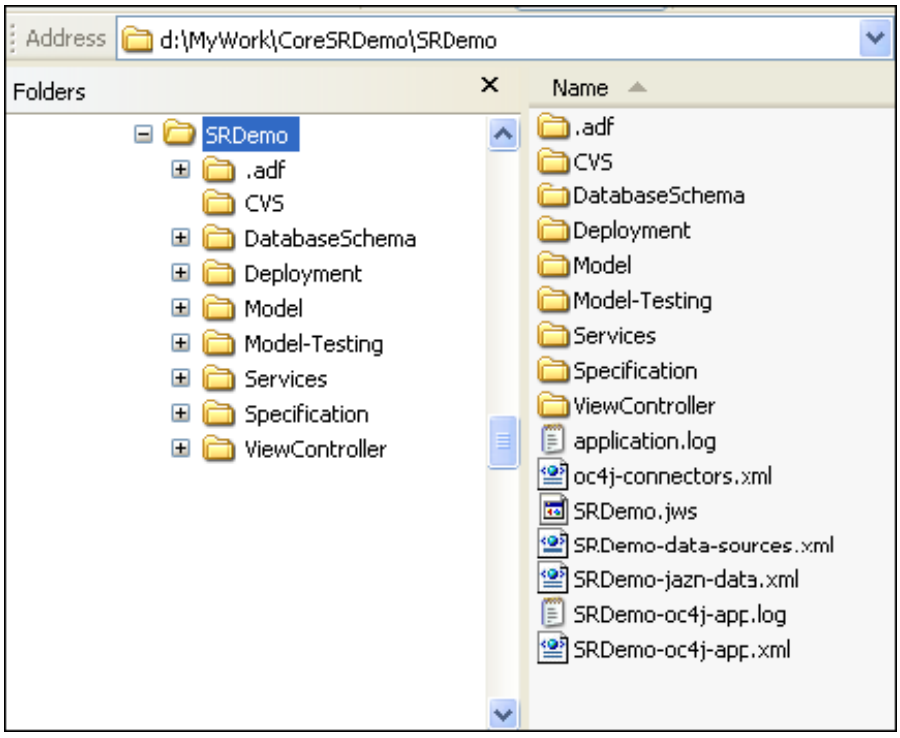
1. アプリケーション・ナビゲータで**「SRDemo」**を右クリックして、コンテキスト・メニューから**「新規プロジェクト」**を選択します。
2. 新規ギャラリで**「空のプロジェクト」**を選択して、**「OK」**をクリックします。
3. プロジェクトに **Deployment** と名前を付けます。

アプリケーションのデプロイを管理するために使用するプロジェクトができました。

サービス・リクエスト・アプリケーションを作成していたときに、SRDemo-jazn-data.xml というファイルにユーザーおよびロールを作成しました。これは、JDeveloper がアプリケーション別にファイルを分類するために使用するデフォルトの名前です。デフォルトでは、最上位レベルのディレクトリにファイルを置きます。

このことは、JDeveloper の開発環境内でテストする際に役立ちます。ただし、ファイルの名前およびディレクトリ構造は、直接 J2EE サーバーにデプロイできません。J2EE サーバーの中には、デプロイされるファイルの名前および場所に関して非常に特殊な要件を必要とするものがあります。次の手順で、ファイルを移動して、標準の J2EE デプロイメント名に名前を変更します。

1. Windows Explorer（または使用しているオペレーティング・システムで同等のもの）を開いて、アプリケーションのルート・ディレクトリに移動します。



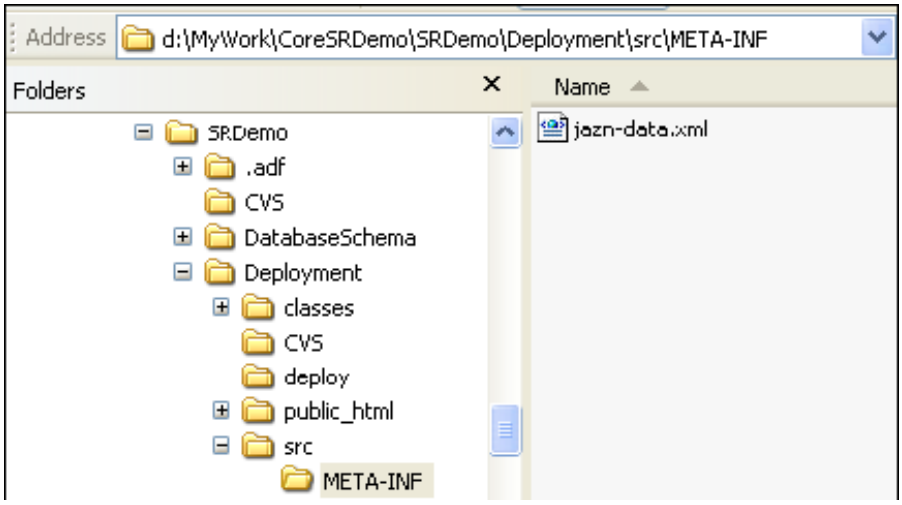
2. 「Deployment」ノードを開いて、次のようにサブディレクトリ構造を作成します。

   ..¥SRDemo¥Deployment¥src¥META-INF

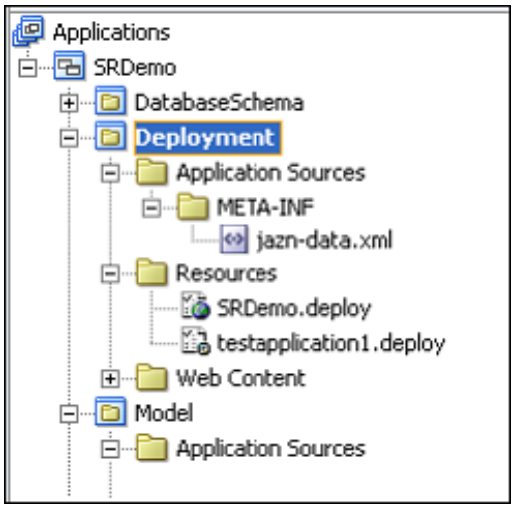
3. これは、jazn-data.xml ファイルおよび data-sources.xml ファイルの J2EE アプリケーション・サーバーに必要な構造です。ディレクトリを作成します。

   Windows を使用している場合は、左側のウィンドウで親を選択して、右側のウィンドウで右クリックし、コンテキスト・メニューから「新規作成」→「フォルダ」を選択します。

4. SRDemo-jazn-data.xml ファイルを新規ディレクトリにコピーします。
5. ファイルの名前を変更して、名前から `SRDemo-` を削除します。結果として、ファイル名は `jazn-data.xml` になります。ディレクトリ構造は、次のスクリーンショットのように表示されます。



6. JDeveloper のアプリケーション・ナビゲータで**「リフレッシュ」**ボタン（  ）をクリックします。
7. プロジェクト構造は次のように表示されます。



これで、ファイル名およびディレクトリ構造はデプロイできる状態になりました。デプロイ前に最後に行う必要があることとして、デプロイメント・プロファイルが TopLink トランザクション・コントローラを使用することを確認します。正しいトランザクション・クラスを指す必要がある、セッション・レベルでのパラメータがあります。

次の手順で、正しいトランザクション管理クラスを使用するよう `sessions.xml` ファイルを変更します。

1. アプリケーション・ナビゲータで、**「DataModel」→「アプリケーション・ソース」→「TopLink」**を選択します。
2. **「sessions.xml」**を選択します。「構造」ペインで**「SRDemoSession」**をダブルクリックします。

3. **「一般」**ページで、**「オプション」**の**「外部トランザクション・コントローラ」**チェック・ボックスを選択します。
4. 次のトランザクション・クラスを入力します。

   ```
   oracle.toplink.transaction.oc4j.Oc4jTransactionController
   ```

5. **「ログイン」**タブをクリックして、**「オプション」**タブをクリックします。
6. 「外部接続プール」を **True** に設定します。
7. 「外部トランザクション・コントローラ」を **True** に設定します。
8. JDeveloper メニューから**「ツール」**→**「設定」**を選択します。
9. **「デプロイ」**をクリックします。
10. **「配布中にデフォルトの data-sources.xml をバンドル」**オプションをクリアします。
11. **「OK」**をクリックします。

これで、アプリケーションはデプロイできる状態になりました。

デプロイメント・プロファイルの作成

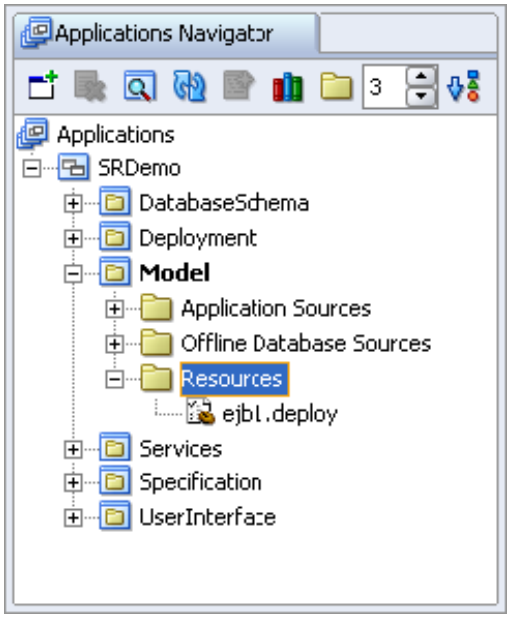
デプロイメント・プロファイルは、アプリケーションのデプロイを管理するプロジェクト・コンポーネントです。デプロイメント・プロファイルには、デプロイメント・パッケージに含まれるソース・ファイル、デプロイメント・ディスクリプタ（必要に応じて）およびその他の補助ファイルがリストされます。

サービス・リクエスト・アプリケーション用のデプロイメント・パッケージには Model プロジェクト（`.jar`）、UserInterface プロジェクト（`.war`）および Deployment プロジェクト（`.ear`）の 3 つのファイルが含まれます。チュートリアルのこの項では、3 つのものそれぞれにデプロイメント・プロファイルを作成します。

最初に作成するデプロイメント・プロファイルは Model プロジェクト用です。このプロジェクトの内容は、主にアプリケーションのデータ・モデル部分を構成する Java クラスです。このプロジェクトのデプロイメント・タイプは EJB JAR（Java アーカイブ）ファイルです。

1. アプリケーション・ナビゲータで **Model** プロジェクトを右クリックして、コンテキスト・メニューから**「新規」**を選択します。
2. 新規ギャラリで**「Deployment Profiles」**→**「EJB JAR ファイル」**を選択して、**「OK」**をクリックします。
3. **「OK」**をクリックして、デフォルトのプロファイル名を受け入れます。
4. **「OK」**をクリックして、「EJB JAR デプロイメント・プロファイルのプロパティ」ダイアログ・ボックスのデフォルトの値を受け入れます。
5. 作業内容を保存します。

6. アプリケーション・ナビゲータは、次のスクリーンショットのように表示されます。

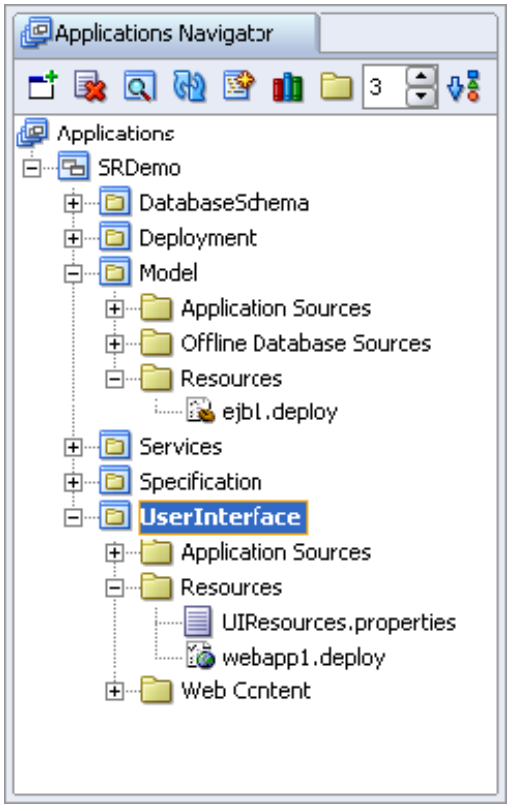


UserInterface デプロイメント・プロファイルの作成

ここでは UserInterface プロジェクト用のデプロイメント・プロファイルを作成します。このプロジェクトは、アプリケーションのユーザー・インタフェース・コンポーネントを作成した場所です。このプロジェクト用のデプロイメント・ファイルは `.war` ファイル（Web アーカイブ、Web コンポーネント用）です。

1. アプリケーション・ナビゲータで「**UserInterface**」プロジェクトを右クリックして、コンテキスト・メニューから「**新規**」を選択します。
2. 新規ギャラリで「**Deployment Profiles**」→「**WAR ファイル**」を選択して、「**OK**」をクリックします。
3. 「**OK**」をクリックして、デフォルトのプロファイル名を受け入れます。
4. Web アプリケーションのコンテキスト・ルートを **SRDemo** に変更します。これは、顧客がアプリケーションへのアクセス用に使用する URL の一部になります。
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・ボックスのデフォルトの値を受け入れます。
6. 作業内容を保存します。

7. アプリケーション・ナビゲータは、次のスクリーンショットのように表示されます。



アーカイブの作成

`.jar` ファイルおよび `.war`（Web アーカイブ）ファイルを作成したので、アプリケーションをデプロイ可能なパッケージにアセンブルできます。

デプロイを構成する部分に、アプリケーションに必要なすべての `.jar` ファイルと `.war` ファイルおよびその他の必要なサーバー構成ファイルを含むデプロイメント・プロファイルを作成します。すでに確認したように、このファイルの中には、`data-sources.xml` ファイルおよび `jazn-data.xml` ファイルが含まれています。

1. アプリケーション・ナビゲータで「**Deployment**」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「**Deployment Profiles**」→「**EAR ファイル**」を選択して、「**OK**」をクリックします。
3. プロファイル名を **SRDemoApplication** に変更して、「**OK**」をクリックします。
4. 「**アプリケーション・アセンブリ**」カテゴリをクリックします。
5. 次の J2EE モジュールを選択して、`.ear` ファイルに含めます。

   「**UserInterface.jpr**」→「**webapp1.deploy**」

   「**Model.jpr**」→「**ejb1.deploy**」
6. 「**OK**」をクリックして、他のデフォルトを受け入れます。

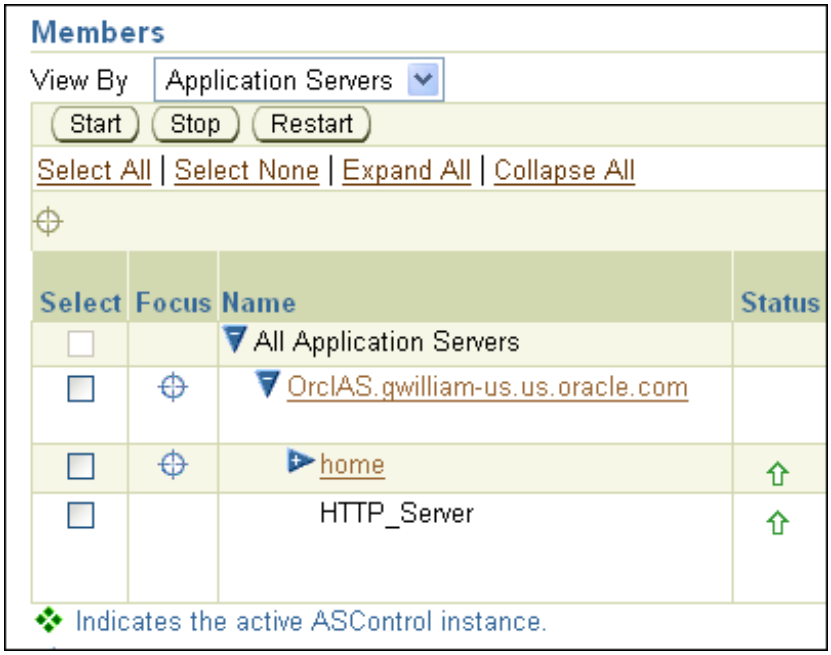
これで、アプリケーションを J2EE サーバーに正常にデプロイするために必要なデプロイメント・プロファイルを作成できました。次の項でアプリケーションのデプロイおよびテストを行います。

接続プールおよびデータソースの作成

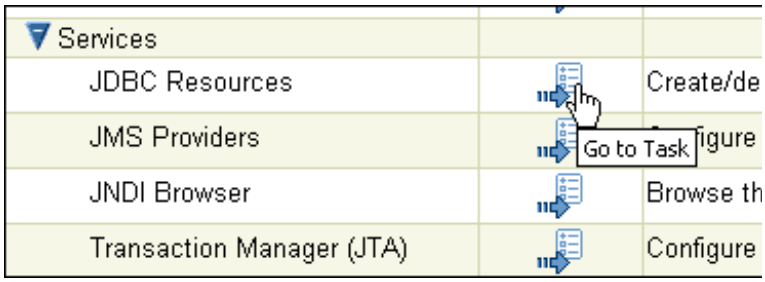
Oracle Application Server 10*g* にデプロイされたアプリケーションは、データソースおよび接続プールを使用してデータベース・アクセスを管理します。アプリケーションを開発しているときに、JDeveloper では `data-sources.xml` ファイルが作成されます。開発中および内部テスト中にこのファイルを使用します。アプリケーションをデプロイするときには、データソースを直接アプリケーション・サーバーに作成します。

この項では、アプリケーション用の接続プールおよびデータソースを作成します。アプリケーションに使用されるデータソース名は SRDemoDS です。EM を使用して作成するデータソースは、開発中に使用したデータソースと同じ名前にする必要があります。

1. まだ完了していない場合は、ブラウザを開いて Enterprise Manager（EM）に接続します。
2. EM でアプリケーション・サーバーの下の**「ホーム」**をクリックします。



3. **「管理」**リンクをクリックします。
4. 「サービス」の下に既存のデータソースおよび接続プールが表示されます。ここで**「JDBC リソース」**行の**「タスクに移動」**をクリックします。



5. **「接続プール」**表の**「作成」**ボタンをクリックします。

6. 次の情報を使用して、接続プールを作成します。

フィールド	値
アプリケーション	Default
接続プール・タイプ	新規接続プール
名前	SRDemoDSPool
コネクション・ファクトリ	oracle.jdbc.pool.OracleDataSource （デフォルト値）
JDBC URL	jdbc:oracle:thin:@//localhost:1521/orcl （ご使用のデータベース）
資格証明 : ユーザー名	srdemo
資格証明 : パスワード	oracle

7. **「終了」**をクリックして、接続プールを作成します。
8. **「接続テスト」**をクリックします。テスト・ページに移動します。**「テスト」**をクリックします。ページ上部の「確認」領域に次のメッセージが表示されます。

   "SRDemoDSPool" への接続が正常に確立されました。

接続プールの作成

次に、SRDemoDSPool 接続プールを使用するデータソースを作成します。

1. 「データソース」表で、**「作成」**ボタンをクリックします。
2. 次の情報を使用して、データソースを作成します。

フィールド	値
アプリケーション	Default
データ・ソース・タイプ	マネージド・データソース
名前	SRDemoDS
JNDI ロケーション	jdbc/SRDemoDS
トランザクション・レベル	グローバルおよびローカル・トランザクション（デフォルト）
接続プール	SRDemoDSPool
ログイン・タイムアウト	0 （デフォルト）

3. **「終了」**をクリックします。
4. **「接続のテスト」**をクリックします。
5. **「接続テスト」**ページで、**「テスト」**ボタンをクリックします。
6. 次のメッセージが表示されます。

   "SRDemoDS" への接続が正常に確立されました。

アプリケーションのデプロイ

JDeveloper では、アプリケーションをアプリケーション・サーバーにデプロイするために、シングルクリック・オプションが提供されています。アプリケーションを EAR ファイルにアセンブルした後に、デプロイメント・プロファイルを右クリックしてターゲット・アプリケーション・サーバーを選択できます。

1. 「**SRDemoApplication.deploy**」デプロイメント・プロファイルを右クリックします。
2. コンテキスト・メニューから、「**配布先**」→「**OracleAS10*g***」を選択します。OracleAS10*g* は、このチュートリアルで先に作成した Oracle Application Server 10*g* への接続名です。

   > **注意：** Oracle Application Server 10*g* へのアクセスの作成を行わず OC4J を使用しなかった場合は、OC4J 用に作成した接続名（OC4J になります）を使用してください。

   デプロイ中に、JDeveloper では `.jar` ファイルおよび `.war` ファイルを再作成して、次に `.ear` ファイルをアセンブルします。ファイルがアセンブルされた後に、JDeveloper では、ターゲットの環境に応じて、ファイルがデプロイされて、アプリケーション・サーバーのディレクトリに解凍されます。
3. 「**OK**」をクリックして、アプリケーション設定を受け入れてデプロイを開始します。
4. デプロイが完了すると、次のメッセージが JDeveloper のログ・ウィンドウに表示されます。

```
Initializing Servlet: javax.faces.webapp.FacesServlet for web
Binding web application(s) to site default-web-site ends...
Applicaticn Deployer for SRDemoApplication COMPLETES. Operati
Elapsed time for deployment:  58 seconds
----  Deployment finished.  ----    Nov 30, 2005 11:20:07 PM
```

アプリケーションのテスト

ここでは、アプリケーションを OC4J のスタンドアロン・インスタンスにデプロイします。アプリケーション用に指定したコンテキスト・ルートを使用して、Web ブラウザでアプリケーションをテストできます。

1. Web ブラウザを開きます。
2. URL として **http://localhost:7777/SRDemo/SRList.jspx** と入力します。

   > **注意：** OC4J を使用している場合、URL はポート番号以外を同じにします。`7777` を **`8888`** に置き換えます。

3. アプリケーションのログイン・ページに移動します。ログインの資格証明として、次の内容を使用します。

フィールド	値
ユーザー名	`sking`
パスワード	`welcome`

4. アプリケーションを参照する場合は、ブラウザを開いたままにします。

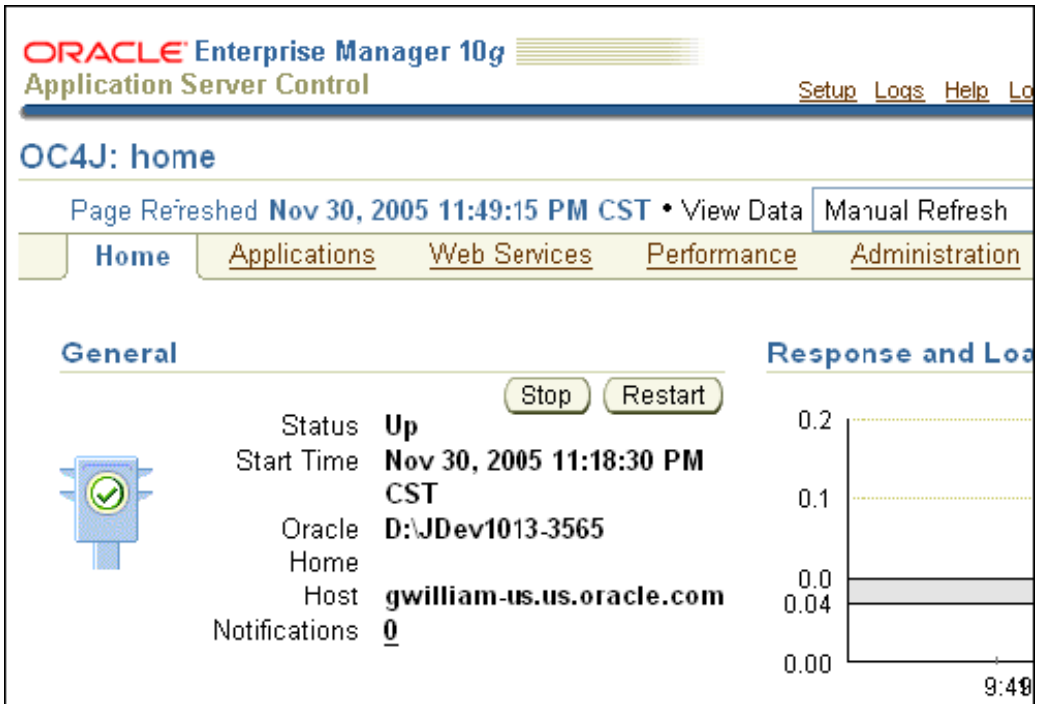
Enterprise Manager を使用したアプリケーション・サーバーの参照

OC4J は、デプロイされたアプリケーションを監視および管理するために使用可能な Web ベースの Enterprise Manager（EM）といっしょに提供されます。この最後の項では、EM を使用してアプリケーションおよびアプリケーション・サーバーを参照します。

1. 2 つ目の Web ブラウザを開きます。
2. URL として **`http://localhost:7777/em`**（OC4J の場合、**`8888`**）と入力します。
3. 次の資格証明を使用して EM にログインします。

フィールド	値
ユーザー名	`oc4jadmin`
パスワード	`admin`

4. ログインすると、EM のホームページに移動します。このページは次のスクリーンショットのように表示されます。



次のように、EM を使用してアプリケーションに関する多くの状況を参照できます。

5. **「アプリケーション」**タブをクリックし、次に**「SRDemoApplication」**をクリックして、直前にデプロイしたアプリケーションの詳細を表示します。
6. **「パフォーマンス」**リンクをクリックして、アプリケーション・パフォーマンスのグラフを表示します。
7. **「管理」**リンクをクリックして、JAZN セキュリティ実装の詳細など、アプリケーションのデプロイ状況をすべて表示します。

まとめ

この章では、JDeveloper を使用して、アプリケーションを OC4J にデプロイする方法を行いました。JDeveloper では、デプロイメント・プロファイルおよびアプリケーション・サーバー接続を使用して、アプリケーションを多くの様々なアプリケーション・サーバーに簡単にデプロイできます。

Enterprise Manager は、Oracle Application Server およびデプロイされたアプリケーションを管理するための、使いやすく強力なインタフェースです。

この章で実行した主なタスクは次のとおりです。

- 既存の Oracle Application Server 10*g* への接続の作成
- デプロイメント・プロファイルの作成
- Enterprise Manager の起動
- アプリケーションのデプロイ
- デプロイのテスト
- Enterprise Manager を使用したアプリケーション・サーバーの参照