BEA WebLogic Portal

開発者ガイド

リリース 7.0 サービス パック 1
マニュアルの日付：2002 年 12 月

著作権

Copyright © 2002 BEA Systems, Inc. All Rights Reserved.

限定的権利条項

本ソフトウェアおよびマニュアルは、BEA Systems, Inc. 又は日本ビー・イー・エー・システムズ株式会社 (以下、「BEA」といいます) の使用許諾契約に基づいて提供され、その内容に同意する場合にのみ使用することができ、同契約の条項通りにのみ使用またはコピーすることができます。同契約で明示的に許可されている以外の方法で同ソフトウェアをコピーすることは法律に違反します。このマニュアルの一部または全部を、BEA からの書面による事前の同意なしに、複写、複製、翻訳、あるいはいかなる電子媒体または機械可読形式への変換も行うことはできません。

米国政府による使用、複製もしくは開示は、BEA の使用許諾契約、および FAR 52.227-19 の「Commercial Computer Software-Restricted Rights」条項のサブパラグラフ (c)(1)、DFARS 252.227-7013 の「Rights in Technical Data and Computer Software」条項のサブパラグラフ (c)(1)(ii) 、NASA FAR 補遺 16-52.227-86 の「Commercial Computer Software--Licensing」条項のサブパラグラフ (d)、もしくはそれらと同等の条項で定める制限の対象となります。

このマニュアルに記載されている内容は予告なく変更されることがあり、また BEA による責務を意味するものではありません。本ソフトウェアおよびマニュアルは「現状のまま」提供され、商品性や特定用途への適合性を始めとする (ただし、これらには限定されない) いかなる種類の保証も与えません。さらに、BEA は、正当性、正確さ、信頼性などについて、本ソフトウェアまたはマニュアルの使用もしくは使用結果に関していかなる確約、保証、あるいは表明も行いません。

商標または登録商標

BEA、Jolt、Tuxedo および WebLogic は BEA Systems, Inc. の登録商標です。BEA Builder、BEA Campaign Manager for WebLogic、BEA eLink、BEA Manager、BEA WebLogic Commerce Server、BEA WebLogic Enterprise、BEA WebLogic Enterprise Platform、BEA WebLogic Express、BEA WebLogic Integration、BEA WebLogic Personalization Server、BEA WebLogic Portal、BEA WebLogic Server および How Business Becomes E-Business は、BEA Systems, Inc. の商標です。

その他の商標はすべて、関係各社がその権利を有します。

開発者ガイド

マニュアルの版数	日付	ソフトウェアのバージョン
なし	2002 年 12 月	7.0 サービス パック 1

目次

第 4 章 ポータルのデプロイ

第 II 部 ポータルの拡張

第 5 章 カスタム テンプレートの作成

第 6 章 ユーザ プロファイルの実装

第 7 章　ポータルへのセキュリティの追加

第 8 章 ポータル コンテンツ管理

第 9 章 ポータル ナビゲーションのセットアップ

第 10 章 ルック アンド フィールの作成

第 11 章 ポートレットの拡張

第 12 章 パーソナライゼーションおよび対話管理のセットアップ

第 13 章 Campaign サービスのセットアップ

第 14 章 Commerce サービスのセットアップ

第 15 章 イベントおよび行動追跡

第 16 章 Expression パッケージの使用

付録 A イベント解説

索引

まえがき

『開発者ガイド』へようこそ。このマニュアルの他に、以下のリソースも活用されることをお勧めします。

オンライン マニュアルの参照　BEA 製品マニュアルは、BEA 社の Web サイトで公開しています。BEA Home ページで [製品のドキュメント] リンクをクリックするか、「e-docs」製品ドキュメント ページ (**http://edocs.beasys.co.jp/e-docs/**) に直接アクセスしてください。

マニュアルについてのフィードバック　BEA WebLogic Portal マニュアルについてのフィードバックをお寄せください。ご質問やコメントがあれば、電子メールで **docsupport-jp@bea.com** までお送りください。なお、お送りいただく電子メールには、WebLogic Portal **7.0** のマニュアルをお使いであることを明記してください。

BEA WebSUPPORT への連絡　このバージョンの WebLogic Portal について質問がある場合、または WebLogic Portal のインストールや実行に問題がある場合には、BEA WebSUPPORT (**http://support.bea.com/welcome.jsp**) を通じて BEA カスタマ サポートにご連絡ください。製品パッケージに同梱のカスタマ サポート カードに記載されている連絡先にお問い合わせいただいても結構です。

第 1 章　WebLogic Portal 開発入門

『WebLogic Portal 開発者ガイド』へようこそ。このガイドでは、ポータルとポートレットの開発とデプロイメントの方法、およびそれらの機能の拡張に必要なリソースの作成方法を紹介します。このガイドで説明するポータル開発作業は、ポータルのライフサイクルの初期フェーズに該当します。すなわち、ポータルとそのポータルの拡張に使用するリソースの作成です。ポータル開発が完了すれば、今度はポータル管理が主な関心事になります。管理作業については、『WebLogic Portal 管理者ガイド』で説明します。

この章では、以下の内容について説明します。

開発者のためのポータル入門

ポータルは、豊富な機能を備えた Web サイトです。ポータルは企業のデータおよびアプリケーションへの単一のアクセス ポイントとなり、そうした情報を統一 的かつパーソナライズされた形式で社員、顧客、およびビジネス パートナに見せることができます。

ポータルを利用すれば、複数の Web アプリケーションを単一の Web インタフェースで統一することができます。ポータルに表示される通常の Web コンテンツの他に、ポータルでは、ポートレット（自己完結型のアプリケーションまたはコンテンツ）をすべて単一の Web インタフェースで表示する機能も提供します。

ポータルでは、タブを使って複数のページを切り替えることができ、各ページには専用のコンテンツとポートレットが表示されています。

ポータルの機能

機能豊富なポータルでは、WebLogic Portal 利用時の体験を豊かなものにしてくれる数々の機能がポータル ユーザに提供される一方、ポータルおよびポータル リソースの開発を促進する開発機能も数多く用意されています。この節では、これらの機能をいくつか紹介します。

パーソナライゼーションと認可

WebLogic Portal には、堅牢な認証機能とパーソナライゼーション機能が用意されているので、管理者は、どのようなコンテンツを訪問者が操作できるか、そしてその情報が特定の訪問者にどう表示されるかを決定することができます。訪問者が自ら WebLogic Portal のパーソナライゼーション機能を利用して、自分専用のコンテンツを選択したり、独自のルック アンド フィール（見た目と使い心地）を作成することができます。そうした認可とパーソナライゼーションを実現するリソースを作成することが、ポータル開発プロセスの主要な要素になります。

グループ ポータル

ポータルには、個別ユーザ用のものとグループ用のものがあります。グループ ポータルを利用すれば、ポータルの委託管理をセットアップしたり、ポータルへのアクセスを特定のユーザに限定することができます。ポータル Web アプリケーション内にグループ ポータルを複数作成することができます。これらのグループ ポータルは、レイアウトやポートレットなどのポータル リソースを互いに共有することができますが、グループごとにコンフィグレーションを変えて、それぞれのニーズに個別に応えることもできます。ユーザはグループのメンバーとして一人一人指名されるので、グループ ポータルでは静的なパーソナライゼーション方式が用いられます。

JSP と JSP タグ

ポータル開発者は、JavaServer Pages (JSP) を用いて、既存のビジネス システムを生かした動的な Web ページを迅速に開発し容易に保守することができます。JSP を使用することで、プラットフォームに依存しない Web ベースのアプリケーションをすばやく開発することができます。JSP ではユーザ インタフェースとコンテンツ生成が分離されているので、基礎となる動的コンテンツを変更せずにページ レイアウト全体を変更することができます。

JSP の主要な構成要素は JSP タグというシンプルなコードで、これを使用すれば、Java コードをまったく使用せずに JSP をたやすく開発できるようになります。JSP タグは XML のようなタグと Java で書かれたスクリプトレットから成り、その中にはページのコンテンツを生成するロジックがカプセル化されています。WebLogic Portal には、Webflow と Pipeline の作成、プロダクト カタログの構築、キャンペーンの開発、コンテンツ管理システムとの統合といった作業に用いる膨大な量の JSP タグ ライブラリが付属しています。

EJB

Enterprise Java Beans (EJB) を使用すれば、サーバ上で動作するビジネス ロジックを実行するソフトウェア コンポーネントを作成することができます。EJB では、トランザクションと状態の管理、マルチスレッド処理、およびリソース プーリングは、サーバ実装に任されています。WebLogic Portal では、EJB は図 1-1 に示すエンタープライズ アプリケーション レイヤを構成し、Web アプリケーションへの Pipeline のロードなどの機能を実行します。

統合ユーザ プロファイル

WebLogic Portal では、ユーザはユーザ プロファイルで表されます。ユーザ プロファイルはユーザの ID を用いて、年齢や電子メール アドレスといった、そのユーザのプロパティにアクセスします。統合ユーザ プロファイルでは、LDAP サーバの他に、あるいは LDAP サーバの代わりに、既存のシステムやデータベースなどの外部データ ソースからユーザ データを取り込むので、ユーザは単一のプロファイルを通じて、さまざまなソースのデータにアクセスできます。ポータル開発時には、ポータル アプリケーションで多種多様な外部ソースからデータを取得できるように、こうしたプロファイルを作成することになります。

その他の有用な機能

WebLogic Portal には、この他に、ポータル開発を支援する以下の機能も用意されています。

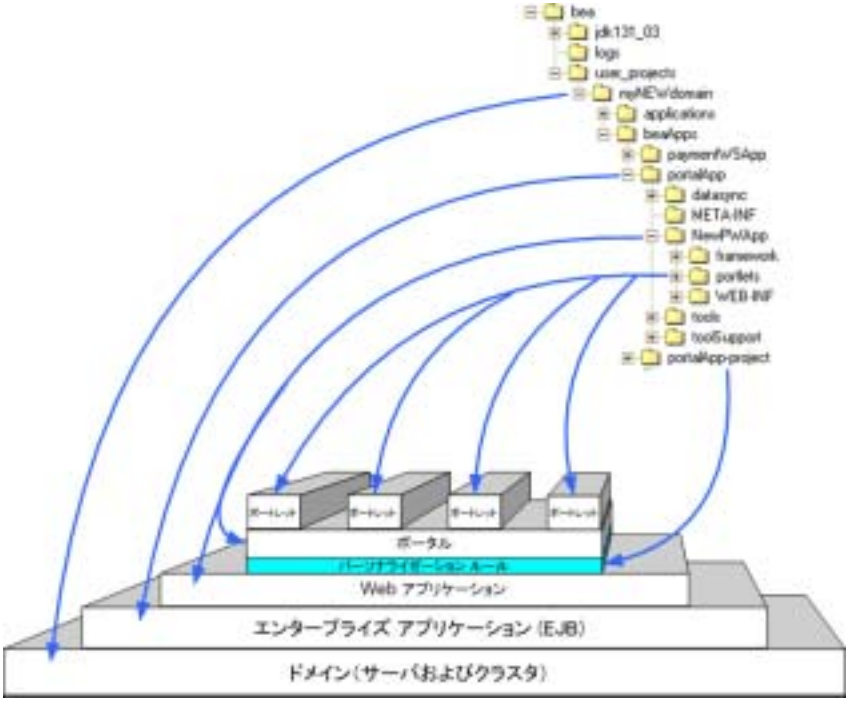
- 動的で対話的なパーソナライズされたコンテンツ向けのレイアウト パラダイム
- グリッド上に配置される、ポートレットと呼ばれるコンテンツ モジュール

- ポートレットをさまざまなレベルでパーソナライズできる機能
- 訪問者がカスタマイズしたレイアウト設定を Web サイトに保存できる機能
- 同じグループ ポータルに複数のビューを定義できる機能
- 配色（スキンと呼ばれる）を指定およびカスタマイズできる機能
- 複雑な分散型セキュリティ実装を可能にする委託管理

ポータル コンポーネント ファイルの格納場所

WebLogic Portal のインストール時に、開発環境を表すファイル構造が自動的に作成されます。そのファイル構造におけるさまざまなレベルのファイルの格納場所と汎用的なポータル アーキテクチャとの関係を図 1-1 に示します。

図 1-1 ポータル コンポーネント ファイルの格納場所



- すべてのものは *ドメイン*に基づいて構築され、ドメインはアプリケーション サーバ（たとえば、WebLogic Server）とクラスタ化されたサーバから成ります。完全なポータル アーキテクチャを構成するファイルとディレクトリはすべて、ドメイン フォルダ (**myNEWdomain**) に格納されます。
- *エンタープライズ アプリケーション*はドメインの上に構築されます。ドメインには、エンタープライズ アプリケーションが複数存在でき、それらは、実装することで Web アプリケーションおよびポータルに機能を作り込むことができる一連の EJB によって特徴づけられます。エンタープライズ アプリケーションを構成するファイルとディレクトリは、**portalApp** フォルダに格納されます。
- 各エンタープライズ アプリケーションは、複数の *Web アプリケーション*（ポータルの基礎）のホストになることができます。Web アプリケーションを構成するファイルには、デプロイメント記述子、コンフィグレーション ファイル、および Java Archive (`.jar`) ファイルがあり、`.jar` ファイルには、

その Web アプリケーションとそのサポート対象ポータルのロジックと書式化命令が格納されています。Web アプリケーションを構成するファイルとサブディレクトリは、Web アプリケーション フォルダ (**newPWApp**) に格納されます。

- Web アプリケーションには、特定の訪問者に望ましいコンテンツを表示するのに役立つ*パーソナライゼーション ルール*を管理および適用するためのロジックも含まれています。どのようなコンテンツを訪問者が操作できるか、そしてその情報が特定の訪問者にどう表示されるかは、これらのルールによって決まります。WebLogic Portal アーキテクチャのパーソナライゼーション レイヤを定義するファイルは、**portalApp-project** ディレクトリのサブディレクトリに格納されます。これらのファイルには、Webflow および Pipeline のコンフィグレーション ファイル、イベント、ポータルおよびポートレット、プロパティ、スキン、レイアウトがあり、これらは E-Business Control Center で使用されます。
- Web アプリケーション内に存在できる*ポータル*は 1 つだけです。このレイヤは WebLogic Portal のユーザ インタフェースになります。すなわち、そこには、訪問者が WebLogic Portal の利用時に実際に目にし操作するコンポーネントが入っています。ポータル アプリケーションを構成するファイルには、デプロイメント記述子、コンフィグレーション ファイル、および Java Archive (`.jar`) ファイルがあり、`.jar` ファイルには、そのポータル アプリケーションのロジックと書式化命令が格納されています。ポータル アプリケーションを構成するファイルとサブディレクトリは、Web アプリケーションと同じフォルダ (**newPWApp**) に格納されます。
- WebLogic Portal アーキテクチャの最上位レイヤは、ポータルの構成要素である*ポートレット*です。ポートレットを構成するファイルには、`.jsp` ファイルと、`.gif` ファイルなどの画像ファイルがあります。これらのファイルは、**portlets** フォルダに格納されます。

ポータル構築のガイドライン

この節では、ポータルおよびポータル リソースの作成に必要な作業について説明し、それらの作業をやり遂げる上で WebLogic Portal がどう役に立つかを示します。以下の問いに対する答えとなるでしょう。

ポータルをどう作成するか

構築対象のポータルが新規ドメイン用であろうと既存ドメイン用であろうと、WebLogic Portal を利用すれば、ポータルおよびポータル アプリケーションの開発は容易になります。BEA では、Java、XML、あるいは HTML の知識がなくてもポータルおよびポートレットを作成しコンフィグレーションできるような「ウィザード」（ポータルのコンフィグレーションとセットアップに関する情報の入力先となる GUI）を用意しています。これらのウィザードから要求されるデータを入力するだけで、ポータルとそれに必要なドメイン、エンタープライズ アプリケーション、Web アプリケーション、およびポートレットが作成されます。

たとえば、Domain Configuration Wizard で新規ドメイン用のポータルを作成する場合には、以下の手順に従います。

1. Web アプリケーションをサポートすることになるエンタープライズ アプリケーションを作成します。
2. ポータル アプリケーションのドメインを新規作成します。

そのあと、Portal Wizard を用いて以下の作業を行います。

3. ポータルをサポートする Web アプリケーションを作成します。
4. 新しいポータルを作成し、そのポータルにポートレットを追加します。
5. そのポータルとポートレットをデプロイします。

さらに、Portlet Wizard を用いて、ポータルにもっとポートレットを追加することができます。

これらのウィザードを利用すれば、動作可能なポータルを 1 時間以内に作成できます。

このガイドの第 1 部「ポータル開発チュートリアル」では、新規ドメインでポータルを新規作成したあと、そのポータルをデプロイする手順（概要は上記のとおり）を 1 つずつ順に説明します。さらに、既存ドメインを新規ポータルのホストにする方法も示します。

ポータルをどう拡張すればよいか

しかるべきポータルができあがったら、そこに機能を追加して企業に対する価値を高めることで、ポータルを拡張することができます。ポータルを拡張する方法には、以下のものがあります。

- ポートレットを追加する
- 静的コンテンツあるいは動的コンテンツを追加する
- 特定ユーザ向けにパーソナライズする
- Webflow と呼ばれるナビゲーション メカニズムを作成し、それに Pipeline を追加することで Webflow の機能を高める
- LDAP サーバや検索エンジンといった、サード パーティによるシステムおよびサービスと統合する
- 支払サービスや税額計算サービスなどのコマース サービスと統合する
- スキンを追加したりレイアウトを変えることで、デフォルトのルック アンド フォール（見た目と使い心地）を変更する

これらを初めとする数々のポータル拡張方法については、第 II 部「ポータルの拡張」で説明します。

作業に取りかかるには

この章で説明したポータルおよびポートレットについての基本知識があれば、今すぐポータルの作成に取りかかることができます。このガイドは、ポータルの開発にもその機能拡張にも役立つように構成されています。

このガイドに記載されている手順はポータルおよびポートレットの開発に必要な知識を示してくれますが、ポータルが自社を完全にサポートできるものになるには、さらに高度な立案作業も少し必要になります。

以下のリストは、ポータルを作成する前に検討しておくべき作業をいくつか示しています。このリストは新規ポータル用の包括的な立案ガイドではありませんが、それでも、作業に取りかかる上で十分役に立つはずです。

実際にポータルを開発する前に、以下の作業が必要になります。

- ポータルの作成、テスト、デバッグ、およびデプロイに用いるツールを決定する
- ポータルで扱うビジネス ニーズを決定し、ポータルの新規作成と既存ポータルの修正のどちらが必要かを決める
- ユーザとグループを定義することで、ポータル訪問者を特定する
- ポータル コンポーネント、すなわちポータルで利用可能になるものを特定する
- ポータル管理ロールおよび責務、すなわち管理者（SA、PA、GA）になるべきユーザと管理者が行うべき作業を特定する
- 新規ポータルを開発する場合には、ポータルおよびポートレットの骨組みを作成する
- ポータルおよびポートレットの HTML モックアップを作成して、望ましいルック アンド フィールをモデリングする
- 具体的なコンテンツを収集または特定し、ポータルおよびポートレットでそれを提供するためにどのようなプロセスが必要になるかを決定する

さあ、これでポータルの作成に取りかかる用意ができました。

第 I 部　ポータルの開発 — チュートリアル

第 I 部「ポータルの開発 — チュートリアル」では、ポータルを作成しデプロイする方法を示します。ここでは、新規ドメインと既存ドメイン (たとえば、別の Web アプリケーションと共用する) の両方の場合について、その中にポータルを作成しデプロイする方法を学びます。これらのチュートリアルを読み終える頃には、BEA から提供されるリソースを利用して、ポートレットと共に、ポータルをすばやく作成しデプロイできるようになっているでしょう。

第 I 部では、以下の内容について説明します。

- 新規ドメインへのポータルの新規作成
- 既存ドメインへのポータルの追加
- ポータルのデプロイ

ポータルを作成しデプロイした後は、それらのポータルの機能の拡張や追加ができるようになります。それらの手順については、このガイドの第 II 部「ポータルの拡張」を参照してください。

第 2 章 新規ドメインへのポータルの新規作成

ステップ 1: ドメインの新規作成

この節では、Domain Configuration Wizard を実行して、WebLogic Portal プラットフォームで提供される管理機能、コマース機能、パーソナライゼーション機能、およびポータル機能をすべて備えた完全なエンタープライズ アプリケーション一式を新規作成する方法を示します。

1. Windows プラットフォームでは、[スタート｜プログラム｜ BEA WebLogic Platform 7.0 ｜ Domain Configuration Wizard] を選択します。

   また、以下のディレクトリから `dmwiz.cmd`（UNIX では `dmwiz.sh`）を実行することで、Domain Configuration Wizard を起動することもできます。

   ```
   <BEA_HOME>\weblogic700\common\bin
   ```

2. 画面左側のドメイン テンプレートのリストから、図 2-1 に示すように **WLP Domain** を選択します。次に、ドメインの名前を入力します。この手順では、例として「**`myNewDomain`**」という名前を使用します。**[Next]** をクリックします。

図 2-1　WLP Domain の選択

3. [サーバ タイプを選択] ページで、図 2-2 に示すように、**[Single Server (StandAlone Server)]** が選択されていることを確かめ、**[Next]** をクリックします。

図 2-2 サーバ タイプの選択

4. [ドメインの場所を選択] ページに表示されているドメインの場所が正しいかどうか確かめます。この例の場合には、図 2-3 に示すように、<BEA_HOME>\user_projects でなければなりません。**[Next]** をクリックします。

図 2-3　ドメインの場所の選択

5. [スタンドアロン / 管理サーバのコンフィグレーション] ページに表示されるサーバ情報を確認します。WebLogic Portal をローカルに実行している場合には、図 2-4 に示すような情報になっているはずです。**[Next]** をクリックします。

図 2-4 サーバのコンフィグレーション

注意： Domain Configuration Wizard で指定するオプションの詳細については、『コンフィグレーション ウィザードの使い方』(http://edocs.beasys.co.jp/e-docs/platform/docs70/confgwiz/index.html) を参照してください。

6. 管理者のユーザ名とパスワードを入力して、管理ユーザを作成します。図 2-5 に示すように、`weblogic/weblogic` の組み合わせが典型的です。[Next] をクリックします。

図 2-5　管理ユーザの作成

7. 図 2-6 に示すように、[サーバの [スタート] メニュー エントリを作成] ページで **[はい]** を選択したあと、[Next] をクリックします。

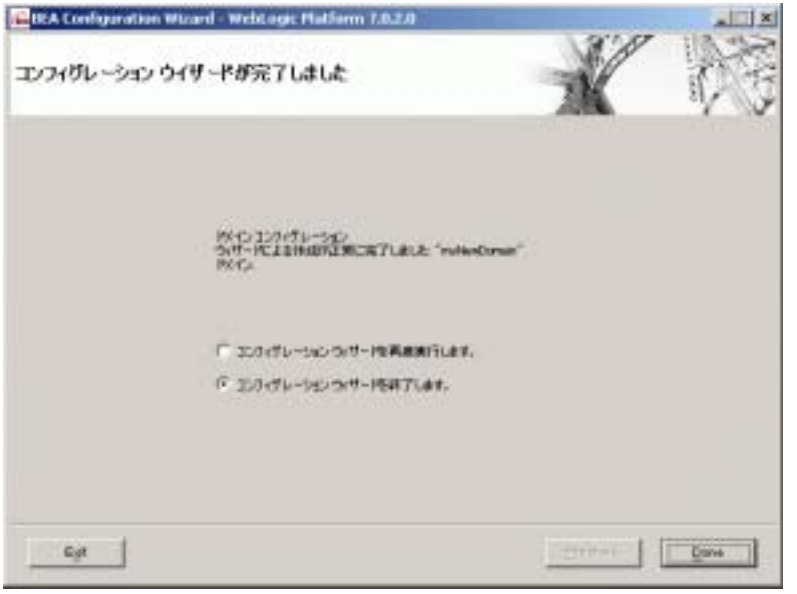
図 2-6 [スタート] メニュー エントリの作成

8. 図 2-7 に示すように [コンフィグレーションの概要] ウィンドウに表示される設定で正しいかどうか確認し、**[作成]** をクリックします。Wizard の実行がしばらく続き、ファイルが処理され作成されます。

図 2-7　コンフィグレーション概要の確認

9. [コンフィグレーション ウィザードが完了しました。] ページが表示されます。図 2-8 に示すように、**[コンフィグレーション ウィザードを終了します。]** が選択されていることを確かめ、**[完了]** をクリックします。

図 2-8　Configuration Wizard の完了

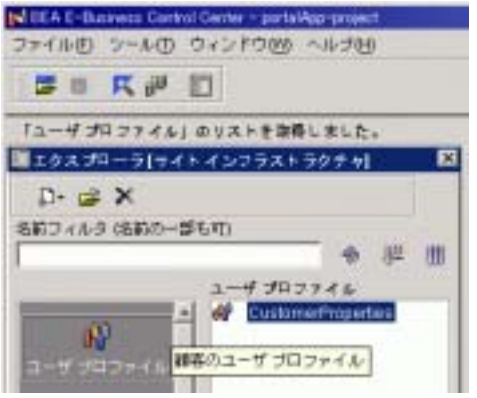


これで、新しいポータルとそれに関連付けられるポータル Web アプリケーションを作成する用意ができました。

上記で作成されたリソースに関する注意

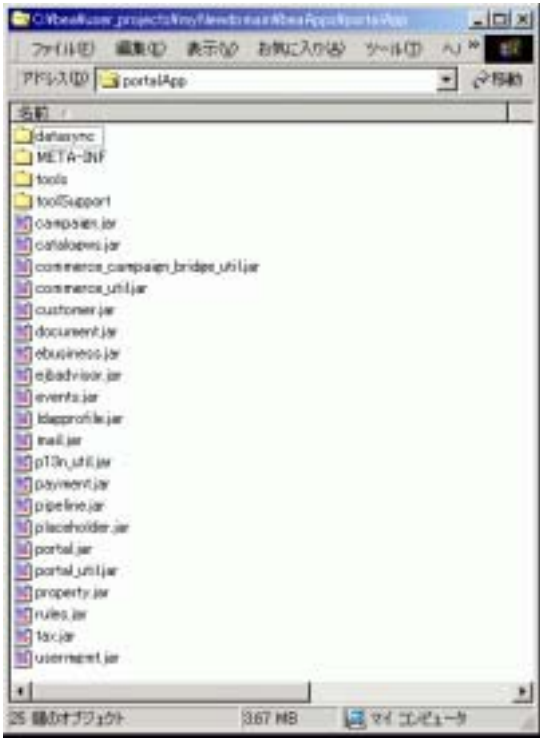
Configuration Wizard で提供されるサイト インフラストラクチャ：ドメインの新規作成の一環として、ドメイン ウィザードは、`portalApp` という完全なエンタープライズ アプリケーションも作成します。E-Business Control Center で `portalApp-project` ファイルを開いたうえで、[**サイト インフラストラクチャ**] タブをクリックし、図 2-9 に示すように **[ユーザ プロファイル]** を選択します。`CustomerProperties` というユーザ プロパティ セットがすでに用意されていることに注意してください。

図 2-9　ユーザ プロファイル



Configuration Wizard で提供される J2EE リソース : 新しいエンタープライズ アプリケーションは（今のところはまだ空のテンプレートですが）、Foundation サービスや、パーソナライゼーション、対話管理、インテリジェント管理、および統合の各サービスを最初からサポートしています。新たに作成されたディレクトリの中を調べれば、図 2-10 に示すように、これらのサービスを実装するための JAR が含まれていることがわかります。

図 2-10 新しいエンタープライズ アプリケーション ディレクトリに含まれている JAR



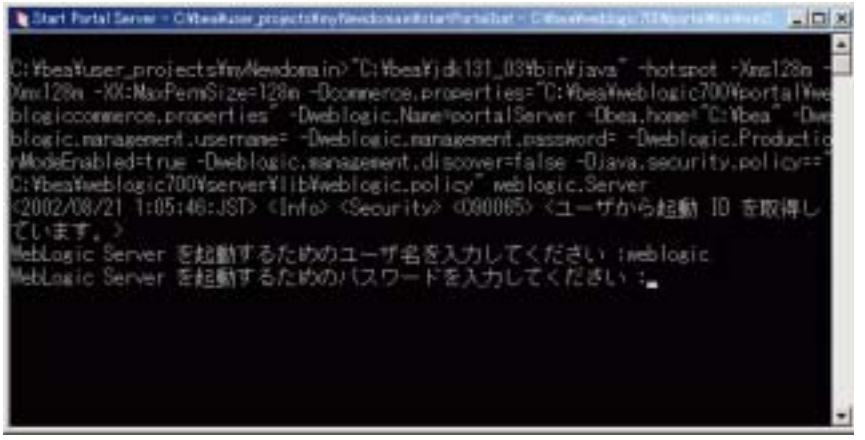
ステップ 2: ポータルの新規作成

サポート役のエンタープライズ アプリケーション リソースができあがったら、以下の手順に従って、新規ポータルを作成しデプロイします。

1. [スタート｜プログラム｜ BEA WebLogic Platform 7.0 ｜ User Projects ｜ < 新規ドメイン名 > ｜ Start Portal Server] を選択して、サーバを起動します（ここで紹介した手順に従っている場合には、< 新規ドメイン名 > は `myNewDomain` になります）。

2. 資格情報の入力を要求されたら、ステップ 1 の 6)（図 2-5）で作成したユーザ名とパスワードを入力します（たとえば、weblogic/weblogic）。ログインの様子を 図 2-11 に示します。

図 2-11　作成済みのドメイン用ユーザ名およびパスワードの入力



注意： **UNIX プラットフォームの場合：**デフォルトでは、新たに作成されたドメイン スクリプトにはどれも実行特権がないので、システム管理者に連絡して、それらに特権を付与してもらう必要があります。

3. [スタート｜プログラム｜ BEA WebLogic Platform 7.0 ｜ WebLogic Portal 7.0 ｜ E-Business Control Center] を選択して、E-Business Control Center を起動します。

4. E-Business Control Center が起動したら、図 2-12 および 図 2-13 に示すように、[ファイル｜プロジェクトを開く] を選択し、<BEA_HOME>user_projects\< 新規ドメイン名 >\beaApps\portalApp-project ディレクトリ内の portalApp-project.eaprj プロジェクト ファイルを開きます。

図 2-12　A: プロジェクト ファイルのオープン

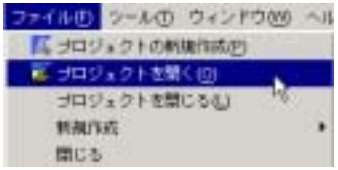
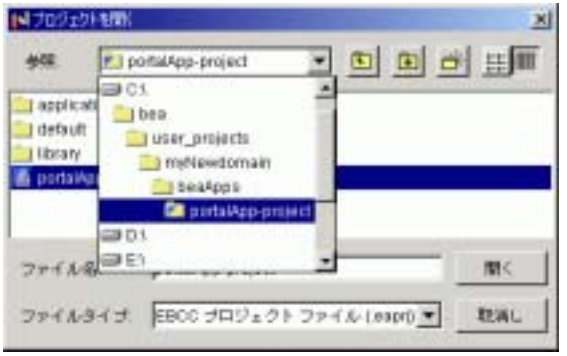
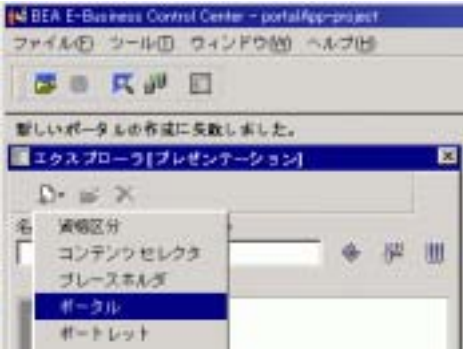


図 2-13　B: プロジェクト ファイルのオープン



5. E-Business Control Center の **[プレゼンテーション]** タブをクリックします。
6. [エクスプローラ] ツールバーの **[新規作成]** アイコンをクリックし、図 2-14 に示すように **[ポータル]** を選択します。

図 2-14　[ポータルの新規作成] ダイアログのオープン



7. 図 2-15 に示すように、**[Portal Wizard を使用して、新しいポータルを最初から作成する。]** オプションが選択されていることを確かめます。[OK] をクリックします。

図 2-15　Portal Wizard 起動画面

8. 図 2-16 に示すように、新規ポータルに名前を付けます。この手順では、例として「**ThisNewPortal**」というポータル名を使用します。

図 2-16　新規ポータルの命名

9. 図 2-17 に示すように、[**新規作成**] ボタンをクリックして、新しいポータル Web アプリケーションを作成します。

10. ポータル Web アプリケーション名を入力します。この手順では、例として「**NewPortalWebApp**」を使用します。**[OK]** をクリックします。

図 2-17　新規ポータル Web アプリケーションの命名

11. 図 2-18 に示すように、ポータル テンプレートを選択します。[**次へ**] をクリックします。

図 2-18　ポータル テンプレートの選択

12. [リソース ファイルの場所] ウィンドウ (図 2-19) で、新規ポータル Web アプリケーション用の J2EE リソースのロケーションが正しいことを確認します。

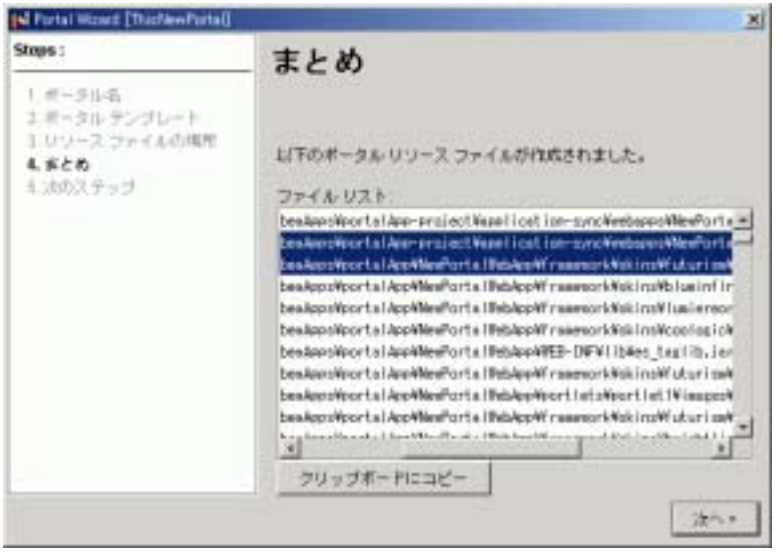
注意： サーバ マシンで E-Business Control Center を稼動している場合、デフォルトのロケーションで問題ないはずです。

13. [**作成**] をクリックします。

図 2-19　リソース ファイルの場所の選択

14. Portal Wizard によってファイルが作成され、図 2-20 に示すように、[まとめ] ページにその一覧が表示されます。[次へ] をクリックします。

図 2-20　ファイル一覧



注意： Portal Wizard によって自動的に作成される 2 種類のファイルの例を コード リスト 2-1 に示します。それは、`select_page_view.gif` などの J2EE リソースと、ポータル フレームワークで使用される `security.wf` などのメタデータです。これらのファイルを格納するのに別個のディレクトリが使用されることに注意してください。

コード リスト 2-1　異なるタイプのポータル リソース

```
\portalApp\NewPortalWebApp\framework\skins\futurism\images\select_page_view.gif
\portalApp-project\application-sync\webapps\NewPortalWebApp\security.wf
```

15. 新規ポータルをデプロイします。図 2-21 に示すように、**[はい、ホット デプロイします ...]** ラジオ ボタンを選択し、**[デプロイ]** をクリックします。

図 2-21　新規ポータルのホット デプロイ

16. 資格情報の入力を要求されたら、ステップ 1 の 6 （図 2-5）で作成した管理者のユーザ名とパスワード（たとえば、**weblogic**/**weblogic**）を入力します。

図 2-22　管理者のユーザ名とパスワードの入力

17. デプロイメント プロセスの実行がしばらく続き、その間、図 2-23 に示すようなウィンドウが表示されます。[詳細] をクリックすると、デプロイメント ログが表示されます。

図 2-23　ホット デプロイ中の処理メッセージ

18. 図 2-24 に示すように新規ポータルが正常にデプロイされたら、**[閉じる]** をクリックします。

図 2-24　ホット デプロイの正常終了メッセージ

19. 図 2-27 に示すように WebLogic Portal Administration Tools 内からこの新規ポータルが見えるかどうか確かめます。それには、以下の URL にアクセスします。

    ```
    http://<hostname>:<port>/portalAppTools/
    ```

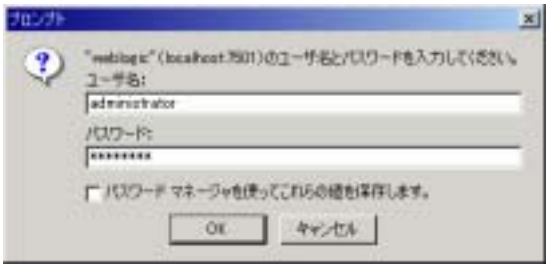
    管理ツールをローカルに実行している場合には、この URL は以下のものになるはずです。

    ```
    http://localhost:7501/portalAppTools/
    ```

20. 図 2-25 に示すように、ログイン画面が表示されます。`administrator/password` としてログインします。

注意： `weblogic/weblogic` を使ってはいけません 。

図 2-25　Administration Tools へのログイン



21. 図 2-26 に示すような、Administration Tools のメイン ウィンドウが表示されます。「**ポータル管理**」という見出しの右側にあるアイコンをクリックします。

図 2-26　[ポータル管理] アイコンのクリック

22. [ポータル管理] ページが表示され、図 2-27 に示すように、先ほど新規作成したポータル Web アプリケーションの名前が表示されます。

図 2-27　Portal Wizard で作成したポータルに関する情報（Administrator Tools での表示）

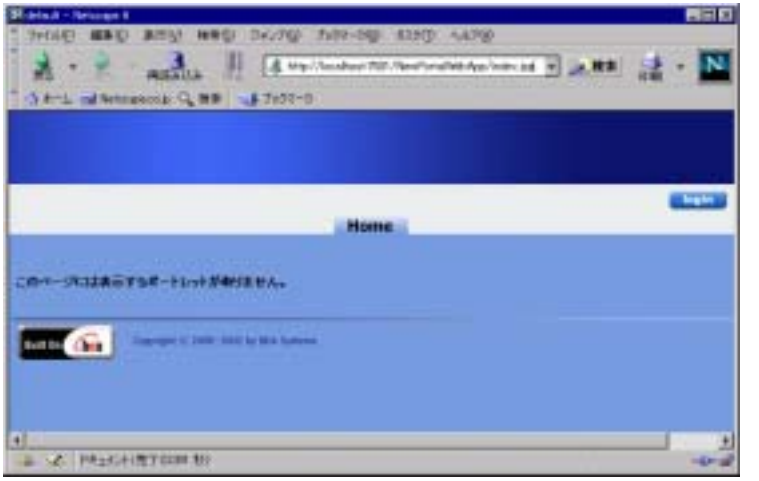
23. この空の新規ポータルはブラウザから見えるはずです。ただし、その中にはまだ何もコンテンツはありません。ブラウザに以下の URL を入力すると、ポータルが表示されます。

    ```
    http://<hostname>:<port>/<newportalwebappname/index.jsp
    ```

    ツールをローカルに実行していて、用意されているサンプル名を使用した場合には、この URL は以下のものになるはずです。

    ```
    http://localhost:7501/NewPortalWebApp/index.jsp
    ```

図 2-28　ブラウザに表示された新規ポータル

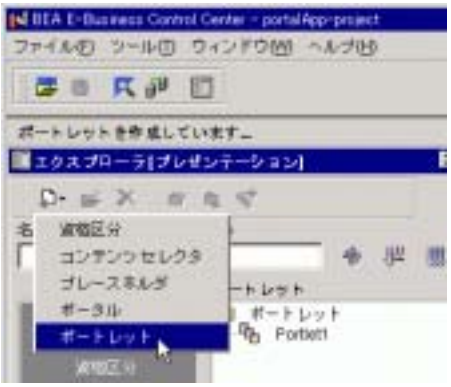


ステップ 3: ポートレットの追加

以上でポータルがデプロイされ動作するようになったので、次は、Portlet Wizard を使用してポータルに新しいポートレットを追加しましょう。

1. E-Business Control Center の [プレゼンテーション] タブで、図 2-29 に示すように、エクスプローラの [**新規作成**] アイコンをクリックし [**ポートレット**] を選択します。

図 2-29　ポートレットの新規作成の選択



2. 図 2-30 に示すように [**Portlet Wizard を使用して、新しいポートレットを最初から作成する。**] が選択されていることを確かめて、[OK] をクリックします。

図 2-30　Portlet Wizard 起動画面

3. 図 2-31 に示すように、新規ポートレットに名前を付けます。この手順では、例として**「BasicPortlet」**という名前を使用します。[**次へ**] をクリックします。

図 2-31 ポートレットの命名

4. 図 2-32 に示すように、ポートレットをポータル ページに関連付けます。この例では、表示される唯一のページである **home** ポータル ページを選択します。

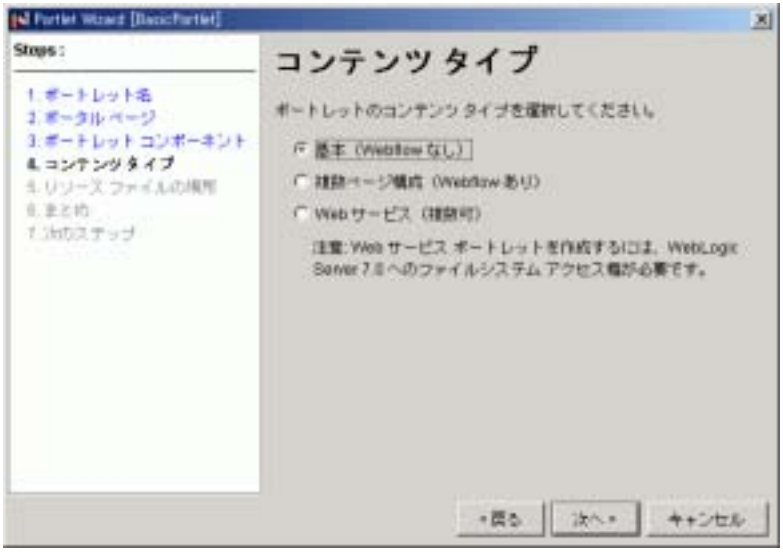
図 2-32　ポータル ページへのポートレットの関連付け

5. 図 2-33 に示す [ポートレット コンポーネント] ページには、ヘッダー、バナー、ヘルプ、フッターといった、ポートレットに追加できるコンポーネントが表示されます。この例では、追加コンポーネントを選択しないで、[**次へ**] をクリックします。

図 2-33　ポートレット コンポーネントの選択

6. 図 2-34 に示すように、ポートレットのコンテンツ タイプを選択します。この例では、[**基本（Webflow なし）**] を選択し、[**次へ**] をクリックします。

図 2-34　コンテンツ タイプの選択



7. ポートレット リソースのデフォルトの格納場所が表示されるので、それで正しいかどうか確かめます。これは、以下のようなパスになっているはずです。

   ```
   <BEA_HOME>\user_projects\<domainname>\beaApps\
   portalApp\<portalwebappname>\portlets
   ```

   この例では、図 2-35 に示されている以下の場所で正しいはずです。

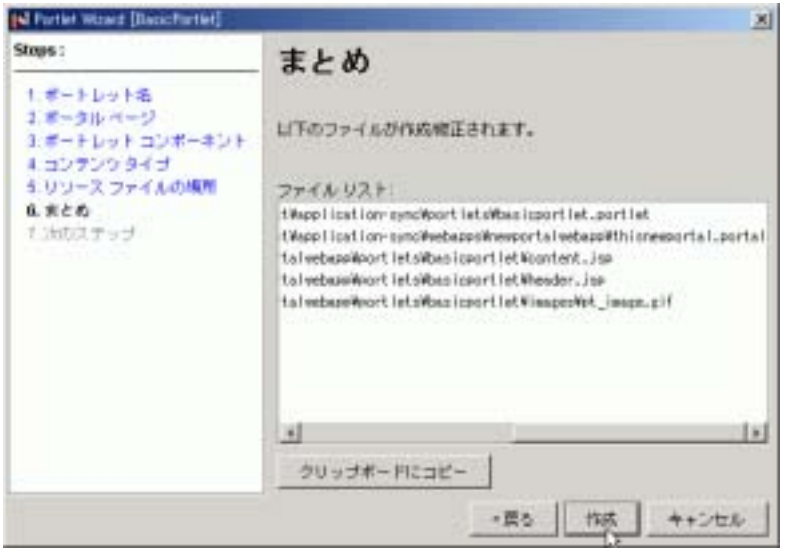
   ```
   C:\bea\user_projects\myNewdomain\beaApps\
   portalApp\NewPortalWebApp\portlets
   ```

   [次へ] をクリックします。

図 2-35　リソース ファイルの格納場所の選択

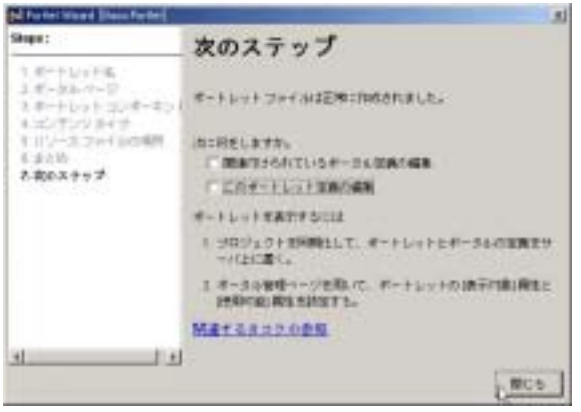
8. [まとめ] ページ (図 2-36 参照) に、Portlet Wizard によって作成されるファイルが示されます。[**作成**] をクリックします。

図 2-36　ファイル一覧の表示



9. 図 2-37 に示すような [次のステップ] ページが表示されたら、両方のチェックボックスのチェックをはずして、[**閉じる**] をクリックします。

図 2-37　オプションをチェックせずに [次のステップ] ウィンドウを閉じる



10. ポータル プロジェクトを同期化します。それには、図 2-38 に示すように、E-Business Control Center ツールバーの [**同期**] ボタンをクリックします。

図 2-38　E-Business Control Center の [同期] ボタン

注意： この例では、接続設定「Default」(`localhost:7501`) が使用されるはずです。

11. Portlet Wizard で作成したデータが E-Business Control Center によって同期化されたあと、図 2-39 に示すようなメッセージが表示されます。[**閉じる**] をクリックします。

図 2-39　同期の完了

12. キャンペーン状態のリセットを選択するウィンドウが表示されたら、[**キャンセル**] をクリックします。

このポートレットをブラウザに表示させるには、「ステップ 4: 新規ポートレットの可視化」に示すように、そのポートレットを表示対象かつ利用可能として指定する必要があります。

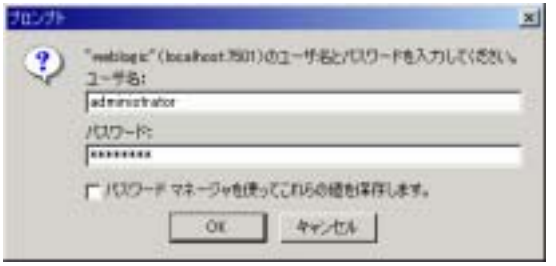
ステップ 4: 新規ポートレットの可視化

これで、新規ポートレットがサーバ上に配置されましたが、さらに、WebLogic Portal Administration Tools を使って、それを利用可能に設定する必要があります。

1. Web ブラウザで、`http://<hostname>:<port>/portalAppTools` という URL にアクセスします。この例では、`http://localhost:7501/portalAppTools` を使用します。

2. 図 2-40 に示すように、**administrator**/**password** としてログインします。

図 2-40　Administration Tools へのログイン



3. Administration Tools のメイン ページが表示されたら、図 2-41 に示すように **[ポータル管理]** をクリックします。

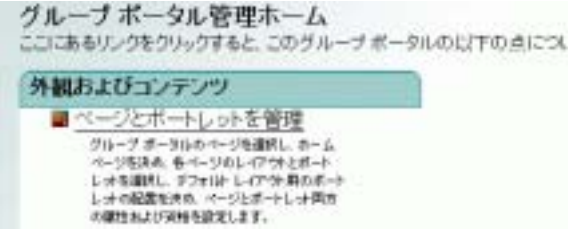
図 2-41　[ポータル管理] へのアクセス

4. 図 2-42 に示すように、[ポータル管理ホーム] ページでデフォルト ポータルをクリックします。

図 2-42　デフォルト グループ ポータルの選択

5. [グループ ポータル管理ホーム] ページで、図 2-43 に示す [ページとポートレットを管理] をクリックします。

図 2-43 [ページとポートレットを管理]



6. 次に、[ページおよびポートレット] ページで、図 2-44 に示すように [ポートレットの編集] をクリックします。

図 2-44 [ポートレットの編集] のクリック

7. [右記ページのポートレットの資格と属性の編集] ページで、ポートレット（この例では BasicPortlet）を選択してから、図 2-45 に示すように [**属性を設定**] ボタンをクリックします。

図 2-45 [属性を設定] の選択

8. 図 2-46 に示すように、ポートレットの属性を [**表示対象**] および [**利用可能**] に設定します。

図 2-46　ポートレット属性の設定

9. [**保存**] をクリックします。これで、ポートレット（この例では BasicPortlet）が表示されるように、属性が設定されました。

10. `http://<hostname>:<port>/<webappname>/index.jsp`（この例では、`http://localhost:7501/NewPortalWebApp/index.jsp`）にアクセスして、結果を確認します。図 2-47 に示すような結果になるはずです。

図 2-47　新規ポートレットの表示

第 3 章 既存ドメインへのポータルの追加

この章では、既存のドメインにポータル機能を追加する方法について説明します。以下に紹介する手順と作業は非常に専門的であり、理解するには WebLogic Server と J2EE アーキテクチャについての十分な知識が必要になります。

ドメインについての前提条件

この章では手順をざっと説明しますが、その手順全体では、既存ドメインに関する以下の事がらを前提としています。

- 既存ドメインはすでにバックアップされている。
- 既存ドメインは `<BEA_HOME>\user_projects\` ディレクトリにインストールされている。
- 既存ドメインには J2EE アプリケーションと少なくとも 1 つの Web アプリケーションが含まれているが、ポータル機能は組み込まれていない。
- 既存ドメイン（この章では `yourDomain` と呼ぶ）は保持されなければならない。すなわち、`portalDomain` に置き換えるわけではない。
- 作業を実行する担当者は、WebLogic Server プラットフォームと J2EE アーキテクチャ全般に精通している。

作業を始める前に

先に進む前に、以下の点について決定しておきましょう。

- 既存ドメインを保持するか置き換えるか

- 既存データベースを使用するか置き換えるか
- エンタープライズ アプリケーションを既存のものから選ぶか新たにインストールするか

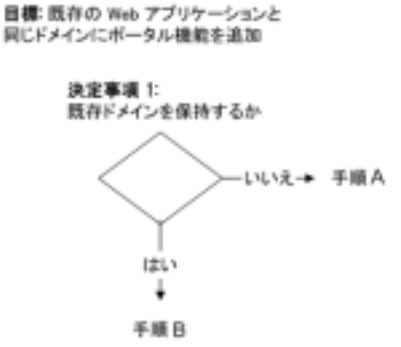
これらを決定してから、どの手順を実行するかを選択し作業を開始します。

既存ドメインを保持するか置き換えるか

最初に決定すべきことは、図 3-1 に示したように、既存の Web アプリケーションを含んでいるドメインに関することです。

- 既存ドメインを新規ポータル ドメインに置き換えてもよい場合には、3-2 ページの「手順 A」に従って、Web アプリケーション コンポーネントを新規ポータル ドメインに移動します。
- 既存ドメインを保持する必要がある場合には、3-3 ページの「手順 B」に従って、既存ドメイン内にポータル機能をインストールします。

図 3-1　決定事項 1



手順 A

1. 2-1 ページの「ステップ 1: ドメインの新規作成」で説明したように、Domain Configuration Wizard を使用して、新しいポータル ドメイン portalDomain を作成します。

2. 既存の Web アプリケーションを構成するオブジェクトを新規ポータル ドメインにインポートします。
3. 2-11 ページの「ステップ 2: ポータルの新規作成」で説明したように、Portal Wizard を使用して、ポータルとポータル Web アプリケーションを新規作成します。

以上の手順が完了したら、既存のポータル Web アプリケーションと完全なポータル Web アプリケーションを含んだ単一のドメインができあがります。

手順 B

1. 2-1 ページの「ステップ 1: ドメインの新規作成」で説明したように、Domain Configuration Wizard を使用して、partsDomain という新しい WLP ドメインを作成します。

注意： この `partsDomain` はテンプレートとして使用されますが、起動はされません。したがって、そのドメインへのリンクを [スタート] メニューに追加してはいけません。

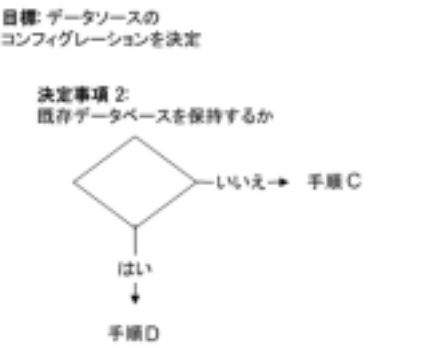
2. このあとの手順を続行し、手順の説明に従って、partsDomain を使用して既存ドメインにリソースをコピーします。

既存データベースを使用するか置き換えるか

もう 1 つの決定事項は、図 3-2 に示したように、どのようなデータベースを使用するかということです。

- 既存のデータベースを保持する必要がなければ、3-4 ページの「手順 C」に従って、Web アプリケーション コンポーネントを新規ポータル ドメインに移動します。
- 既存ドメインを保持する必要がある場合には、3-4 ページの「手順 D」に従って、既存ドメイン内にポータル機能をインストールします。

図 3-2　決定事項 2



手順 C

既存の Web アプリケーションを新規データベースでサポートできる場合には、既存の Web アプリケーションをサポートするデータベース オブジェクトを、新規ドメイン内のそのデータベースにロードします。

手順 D

既存の Web アプリケーションに関連付けられているデータベースを引き続き使用するには、以下のエントリをしかるべき場所に設定する必要があります。

JDBC エントリ

以下の JDBC エントリを既存ドメイン内の `config.xml` ファイルに追加する必要があります。

- commercePool: ポータル Web アプリケーションには、2 つの接続プールと、プールごとに 2 つのデータ ソース（コード リスト 3-1 に示す）が必要になる。

コード リスト 3-1　commercePool データソース エントリ

```
<JDBCDataSource
```

```
  JNDIName="weblogic.jdbc.pool.commercePool"
  Name="commercePool"
  PoolName="commercePool"
  Targets="portalServer"
/>
```

- commercePool データソース : コマース機能には、コード リスト 3-2 に示すような commercePool データソース エントリが必要になる。

コード リスト 3-2　commercePool データソース エントリ

```
<JDBCTxDataSource
  EnableTwoPhaseCommit="false"
  JNDIName="weblogic.jdbc.jts.commercePool"
  Name="commercePool"
  PoolName="commercePool"
  Targets="portalServer"
/>
<JDBCDataSource
  JNDIName="weblogic.jdbc.pool.commercePool"
  Name="commercePool"
  PoolName="commercePool"
  Targets="portalServer"
/>
```

- dataSyncPool: コード リスト 3-3 に示すような dataSyncPool エントリを追加する。

コード リスト 3-3　dataSyncPool エントリ

```
<JDBCTxDataSource
  EnableTwoPhaseCommit="false"
  JNDIName="weblogic.jdbc.jts.dataSyncPool"
  Name="dataSyncPool"
  PoolName="dataSyncPool"
  Targets="portalServer"
/>
```

- DataSync データ：コード リスト 3-4 に示すような DataSync データ エントリを追加する。

コード リスト 3-4　DataSync データソース エントリ

```
<JDBCTxDataSource
  EnableTwoPhaseCommit="false"
  JNDIName="weblogic.jdbc.jts.dataSyncPool"
  Name="dataSyncPool"
  PoolName="dataSyncPool"
  Targets="portalServer"
/>
<WebAppComponent
  Name="datasync"
  ServletReloadCheckSecs="300"
  Targets="portalServer"
  URI="datasync"
/>
```

レルムの選択（省略可能）

- 既存のデータベース レルムを使用するのに、特別なコンフィグレーションは不要。
- 新しい RDBMS レルムを使用するには、それを 1 つコンフィグレーションしてから、config.xml 内でそれを参照する必要がある。

KeyBootstrap（コマース機能を使用する場合）

ポータル アプリケーションでコマース クレジット カード機能を使用するには、KeyBootstrap クラスへの参照（コード リスト 3-5 に示す）が必要になります。

コード リスト 3-5　KeyBootstrap クラスへの参照

```
<StartupClass
  ClassName="com.beasys.commerce.ebusiness.security.KeyBootstrap"
  FailureIsFatal="false"
  Name="KeyBootstrap"
  Targets="portalServer"
/>
```

P13N Console

- config.xml ファイルに、コード リスト 3-6 に示すようなエントリを追加することで、Personalization Console をデプロイする。

コード リスト 3-6　Personalization Console デプロイメント エントリ

```
<Application
   Deployed="true"
   TwoPhase="true"
   StagedTargets="portalServer"
   Name="p13nConsoleApp"
```

```
   Path="<BEA_HOME>/weblogic700/portal/lib"
>
<WebAppComponent
   Name="p13nConsole"
   ServletReloadCheckSecs="300"
   Targets="portalServer"
   URI="p13nConsole.war"
/>
</Application>
```

WLPDocs サービス

- WebLogic Portal Administration Tools をオンライン ヘルプにリンクするには、WLPSDocs サービスが必要になる。

コード リスト 3-7　WLPSDocs サービスのエントリ

```
<Application
   TwoPhase="true"
   StagedTargets="portalServer"
   Deployed="true"
   Name="wlpDocsApp"
   Notes=""
   Path="<BEA_HOME>/weblogic700/portal/lib"
>
   <WebAppComponent
     IndexDirectoryEnabled="false"
     Name="wlpDocs"
     Targets="portalServer"
     URI="wlpDocs.war"
     ServletReloadCheckSecs="300"
```

```
   />
</Application>
```

Tax サービスと Payment サービス（省略可能）

既存ドメインにサンプルの Payment サービスおよび Tax サービスを追加するには、コード リスト 3-8 に示すエントリが必要になります。ポータル アプリケーションへのこれらのサービスの追加については、『管理者ガイド』の「Commerce サービスの管理」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/commerce.htm) を参照してください。

コード リスト 3-8　Tax サービスと Payment サービスのエントリ

```
<Application
  Deployed="true"
  TwoPhase="true"
  StagedTargets="portalServer"
  Name="paymentWSApp"
  Path="<BEA_HOME>/user_projects
/myPARTSdomain/beaApps/paymentWSApp"
>
 <EJBComponent
  Name="payment"
  URI="payment.jar"
  Targets="portalServer"
 />
 <WebAppComponent
  Name="payment-webservice"
  URI="pay-ws"
  Targets="portalServer"
/>
</Application>
```

```
<Application
  TwoPhase="true"
  StagedTargets="portalServer"
  Deployed="true"
  Name="taxWSApp"
  Path="<BEA_HOME>/user_projects/myPARTSdomain/beaApps/taxWSApp"
>
  <EJBComponent
    Name="tax"
    URI="tax.jar"
    Targets="portalServer"
  />
  <WebAppComponent
    Name="tax-webservice"
    URI="tax-ws"
    Targets="portalServer"
  />
</Application>
```

サーバ参照の検証

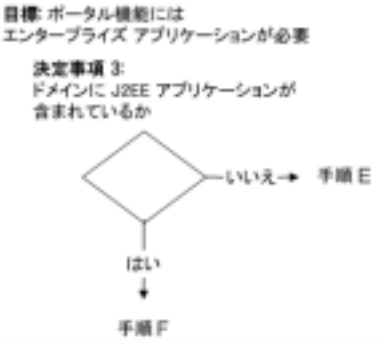
以上の編集が済んだら、config.xml ファイルを調べます。portalServer への参照をすべて検索し、それらを既存ドメインの対象サーバの名前に置き換えます。

これで、既存ドメイン内の config.xml ファイルには必要なエントリが設定されたので、WebLogic Portal サーバから、J2EE エンタープライズ ポータル アプリケーションを構成するすべてのコンポーネントに接続することができます。次のステップは、これらのコンポーネントがサーバ上のどのエンタープライズ アプリケーション内で動作できるかを特定することです。

エンタープライズ アプリケーションを既存のものから選ぶか新たにインストールするか

BEA WebLogic Platform 7.0 ドメインでは、ポータルには完全な J2EE アプリケーションが必要です。決定事項 3 では、図 3-3 に示すように、エンタープライズ アプリケーションを指定します。すなわち、3-11 ページの「手順 E」に従って、既存ドメインにエンタープライズ アプリケーションを追加するか、それとも、3-11 ページの「手順 F」に従って、既存ドメイン内にすでに存在するエンタープライズ アプリケーションにポータル機能を結合します。

図 3-3　決定事項 3



手順 E

1. <BEA_HOME>\user_projects\portalDomain\beaApps\portalApp ディレクトリの内容をそっくり既存ドメイン内にコピーします。
2. 既存の Web アプリケーションを構成するオブジェクトを、このエンタープライズ アプリケーション内に移動します。

手順 F

1. 既存ドメイン内の完全なエンタープライズ アプリケーションを作業の出発点とします。

2. 以下のディレクトリの内容およびサブフォルダをすべて、このアプリケーション ディレクトリ構造にコピーします。
   ```
   tools/
   DataSync/
   toolSupport/
   ```
3. 以下のディレクトリ内の `portalApp-project.eaprj` ファイルをコピーします。

```
bea\user_projects\portalDomain\beaApps\portalApp-project
```

4. BEA XML エディタを使って、既存ドメイン内と partsDomain 内の `META-INF/weblogic-application.xml` ファイルをマージします。これらのファイルのマージに当たっては、コード リスト 3-9 に示すエントリをコピーし、その中の「portalApp」という文字列を既存ドメイン内の J2EE アプリケーションの名前に置き換えます。たとえば、既存のエンタープライズ アプリケーションの名前が existingApp であったとすると、該当するエントリはそれぞれ、existingAppTools、existingAppDataSync、および existingAppTool となります。

コード リスト 3-9　weblogic-application.xml のコード

```
<module>
  <web>
    <web-uri>tools</web-uri>
    <context-root>portalAppTools</context-root>
  </web>
</module>
<module>
  <web>
    <web-uri>datasync</web-uri>
    <context-root>portalAppDataSync</context-root>
  </web>
</module>
<module>
  <web>
    <web-uri>toolSupport</web-uri>
```

```
    <context-root>portalAppTool</context-root>
  </web>
</module>
```

5. `application-config.xml` を既存のエンタープライズ アプリケーション内にコピーします。
6. ポータル エンタープライズ アプリケーションへの参照を `config.xml` に挿入するか、WebLogic Server Console を使ってこれらのモジュールをデプロイします。
7. 2-11 ページの「ステップ 2: ポータルの新規作成」で説明したように、Portal Wizard を使用して、ポータルとポータル Web アプリケーションを新規作成します。

   以上の手順を完了すれば、単一のドメインで、既存の Web アプリケーションと完全な WebLogic Portal インスタンスを両方とも実行できるようになっているはずです。

第 4 章 ポータルのデプロイ

この章では、ポータル アプリケーションを開発環境にデプロイするのに必要な手順の概要を説明します。

この章では、以下のトピックを扱います。

- Portal Wizard を用いたホット デプロイ
- Portal Wizard を用いないホット デプロイ
- ホット デプロイを用いないポータル デプロイメント
- ポータル デプロイメントについてのベスト プラクティス ガイドライン

詳細については、「デプロイメント」ページ (http://edocs.beasys.co.jp/e-docs/wls/docs70/deployment.html) を参照してください。また、クラスタや管理対象サーバなど、運用環境へのデプロイ方法については、『*コンフィグレーション ウィザードの使い方*』(http://edocs.beasys.co.jp/e-docs/platform/docs70/confgwiz/index.html) を参照してください。

Portal Wizard を用いたホット デプロイ

Portal Wizard を使用してポータルを作成する際には、ウィザードの最後の画面で、ポータル Web アプリケーションをすぐにホット デプロイすることを選択できます。これが、最も簡単なデプロイ方法です。

ポータルをいったんデプロイしたら、再度デプロイしなくても、そのポータルにポートレットを追加作成することができます。E-Business Control Center で同期化を実行して、デプロイ済みのポータルに新しいポートレットを追加すればよいだけです。

図 4-1　Portal Wizard の [次のステップ] ウィンドウでのホット デプロイ オプション

Portal Wizard を用いないホット デプロイ

Portal Wizard を用いないでデプロイするには、多少手作業が必要になります。

注意 : ダミー ポータルを作成しホット デプロイしておくと、手動デプロイ時に必要な作業のモデルとして非常に役に立つ場合があります。ダミーのドメイン、ポータル、ポータル Web アプリケーション、および関連する J2EE リソースの作成とデプロイについては、第 2 章「新規ドメインへのポータルの新規作成」を参照してください。

手動でデプロイするには、WebLogic Server の「デプロイメント」ページ (http://edocs.beasys.co.jp/e-docs/wls/docs70/deployment.html)、特に「WebLogic Server デプロイメント」(http://edocs.beasys.co.jp/e-docs/wls/docs70/programming/deploying.html) を参照してください。

デプロイメントに関するそれらの指示に従い、ダミー ポータルを参照用として使用する際には、以下の作業を必ず行ってください。

- ポータルの J2EE リソースをサーバ上に配置する
- E-Business Control Center を使用してメタデータを編集する (J2EE リソースの格納場所をサーバに知らせる)
- コンソールを使って、ポータル Web アプリケーションを WebLogic サーバに追加する
- ユーザ ドメインから WebLogic Portal サーバを起動する
- E-Business Control Center を使用して、メタデータを WebLogic Server に同期化する

ホット デプロイを用いないポータル デプロイメント

E-Business Control Center を用いてポータルとポータル Web アプリケーションを新規作成する場合に、何らかの理由でサーバへのホット デプロイを行わないことにした場合には、そのアプリケーションの J2EE リソースは自動的にはデプロイされません。ここでは、新しいポータル Web アプリケーションをデプロイする際のシナリオとして、以下の 2 つを紹介します。

- E-Business Control Center プロジェクト ファイルがターゲット サーバとは別のマシン上に存在する場合には、まず J2EE リソースをそのサーバに移してから WebLogic コンソールを用いてデプロイすることができる。これに該当する場合には、この節のステップ 1 ～ 3 を実行してください。
- E-Business Control Center がターゲット サーバ上で動作している場合には、J2EE ファイルはすでに適切な場所にあるので、あとは、WebLogic コンソールでアプリケーション コンポーネントをデプロイするだけである。これに該当する場合には、この節の最初のステップを飛ばしてください。

ポータル Web アプリケーションを手動でデプロイする

Portal Wizard のホット シンク機能を用いてデプロイされなかったポータルをデプロイするには、以下の手順に従う必要があります。

ステップ 1: J2EE リソースを移す

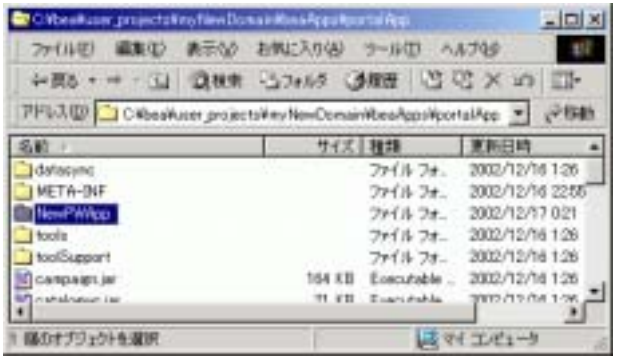
E-Business Control Center がターゲット サーバとは別のリモート サーバ上で動作している場合には、J2EE リソースを移す必要があります。リモート サーバ上には、図 4-2 に示すような完全なエンタープライズ ポータル アプリケーションが存在していなければなりません。

図 4-2　ポータルのないエンタープライズ ポータル ドメイン

J2EE リソースをローカル サーバからリモート サーバに移すには、以下の手順に従います。

1. 図 4-3 に示す NewPWApp ディレクトリをローカル サーバから、図 4-2 に示すポータル エンタープライズ アプリケーション ディレクトリにコピーします。

図 4-3 ポータル Web アプリケーションのコピー



2. ローカル サーバ上の application-sync ディレクトリ内の NewPWApp ディレクトリ (図 4-4 に示す) を、リモート サーバ上の application-sync\webapps ディレクトリ (図 4-5 に示す) にコピーします。

図 4-4 ローカルのメタデータ ディレクトリ

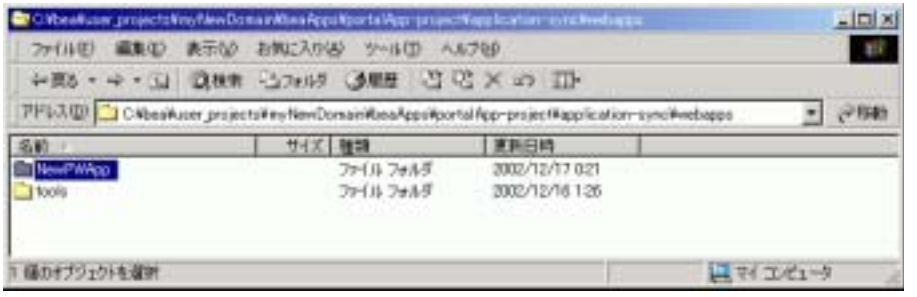


図 4-5　メタデータのコピー先となるリモート ディレクトリ

3. 引き続き、この手順の残りのステップを実行します。

注意： WebLogic プラットフォームで利用可能なサービスをすべてポータル Web アプリケーションに組み込むには、以下の構成要素が適切に配置されている必要があります。

- コード リスト 4-1 と コード リスト 4-2 に示す taglib JAR が以下のディレクトリに含まれていなければならない。

  `beaApps\portalApp\<yourPortalWebApp>\WEB-INF\lib`

- コード リスト 4-3 に示すエンタープライズ アプリケーション JAR が以下のディレクトリに含まれていなければならない。

  `beaApps\portalApp\<yourPortalWebApp>\`

コード リスト 4-1　baseportal インスタンスをサポートするのに必要な taglib JAR

```
weblogic700\common\templates\webapps\portal\baseportal\j2ee\WEB-INF\lib:
                util_taglib.jar
                webflow_servlet.jar
                ent_taglib.jar
                i18n_taglib.jar
                webflow_taglib.jar
                um_taglib.jar
                lic_taglib.jar
```

```
es_taglib.jar
ren_taglib.jar
portlet_taglib.jar
res_taglib.jar
p13n_servlet.jar
weblogic-tags.jar
portal_taglib.jar
portal_servlet.jar
visitor_taglib.jar
pz_taglib.jar
ph_taglib.jar
ps_taglib.jar
cm_taglib.jar
```

コード リスト 4-2　すべてのポータル サービスをサポートするのに必要な taglib JAR

```
/weblogic700/portal/lib/commerce/web/cat_taglib.jar
/weblogic700/portal/lib/commerce/web/eb_taglib.jar
/weblogic700/portal/lib/commerce/web/productTracking_taglib.jar
/weblogic700/portal/lib/p13n/web/ad_taglib.jar
/weblogic700/portal/lib/p13n/web/cm_taglib.jar
/weblogic700/portal/lib/p13n/web/ph_taglib.jar
/weblogic700/portal/lib/p13n/web/ps_taglib.jar
/weblogic700/portal/lib/p13n/web/pz_taglib.jar
/weblogic700/portal/lib/p13n/web/tracking_taglib.jar
```

コード リスト 4-3　ポータルに必要なエンタープライズ JAR

```
campaign.jar
```

```
catalogws.jar
commerce_campaign_bridge_util.jar
commerce_util.jar
customer.jar
document.jar
ebusiness.jar
ejbadvisor.jar
events.jar
ldapprofile.jar
mail.jar
p13n_util.jar
payment.jar
pipeline.jar
placeholder.jar
portal.jar
portal_util.jar
property.jar
rules.jar
tax.jar
usermgmt.jar
```

ステップ 2: メタデータを同期化する

E-Business Control Center でアプリケーションのプロジェクトを開き、同期ステップを実行します。このステップの詳細については、『管理者ガイド』の「アプリケーションへの新規ポータル データの同期」という節を参照してください。

注意： 新しいポータル Web アプリケーションをリモート サーバにデプロイする場合には、必ず、E-Business Control Center で新しいセッションをセットアップし、アプリケーション名を、Web アプリケーション名である NewPWApp ではなく、エンタープライズ アプリケーションである portalApp としてください。

ステップ 3: WebLogic Server Console でデプロイする

新しいポータル Web アプリケーションのデプロイ プロセスは、以下の 3 つのステップから成ります。

- 新しい Web モジュールの作成
- 新しい Web アプリケーションのコンフィグレーション
- デプロイメントの検証

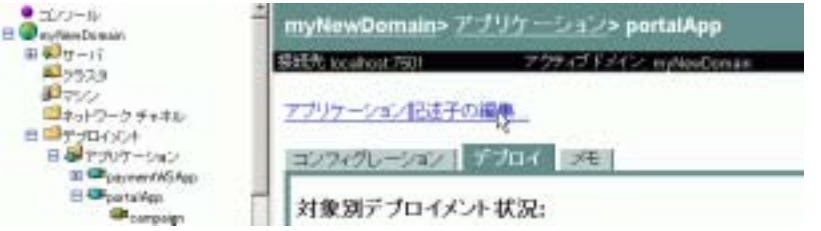
新しい Web モジュールの作成

1. 以下の URL で WebLogic Server Console にアクセスします。

   ```
   http://<yourserver>:<port>/console/
   ```

2. 左ペインで、[**myNewDomain** | **デプロイメント** | **アプリケーション** | **portalApp**] をクリックします。
3. 右ペインで、図 4-6 に示すように、[**アプリケーション記述子の編集 ...**] をクリックします。

図 4-6 アプリケーション記述子の編集



4. 新しいブラウザ ウィンドウが表示されたら、図 4-7 に示すように、左ペインで [Web Module] を右クリックします。

図 4-7 [Web Module] を右クリックしたところ

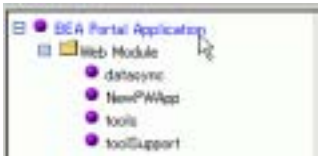
5. [新しい WebModule の作成] 画面が表示されたら、URI (ネイティブ ファイル システム上のエンタープライズ ディレクトリからの相対パス。たとえば、`beaApps\portalApp`) とコンテキスト ルート (新しいポータル Web アプリケーションの URL) を入力します。

6. 図 4-8 に示すように、画面右隅の [**作成**] を、クリックしたあと [**適用**] をクリックします。

図 4-8　モジュール URI とコンテキスト ルートの入力

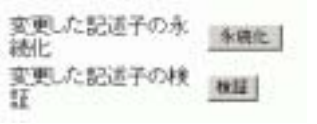
7. 図 4-9 に示すように、左ペイン上部の [BEA Portal Application] をクリックします。

図 4-9　[BEA Portal Application] の選択



8. 同じウィンドウの右ペインで、図 4-10 に示すように、[**永続化**] をクリックします。

図 4-10　[永続化] のクリック



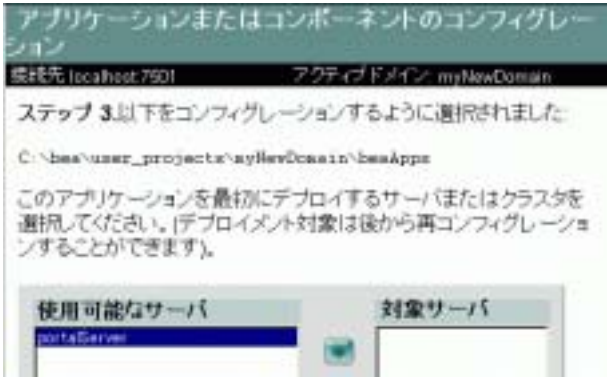
9. 「永続化は成功しました。」というメッセージが右ペインに表示されるはずです。ブラウザ ウィンドウを閉じます。

新しい Web アプリケーションのコンフィグレーション

1. メイン コンソール ウィンドウの表示を更新したあと、[**< ユーザ ドメイン >** | **デプロイメント** | **Web アプリケーション**] を選択し、右ペインの [**新しい Web Application のコンフィグレーション**] をクリックします。
2. 最初の [**コンフィグレーションするアプリケーションまたはコンポーネントの場所を指定**] ページが表示されたら、**beaApps** アプリケーションの左側の [**select**] をクリックします。
3. [**アプリケーションまたはコンポーネントのコンフィグレーション**] ページが表示されたら、図 4-11 に示すように、右ペインのリストからサーバを選択します。

注意 : リモート サーバ上のアプリケーションをコンフィグレーションしようとしている場合でも、図 4-11 に示す [**ステップ 3**] では、Web ブラウザが動作しているマシンではなくサーバそのものに対してローカルなドライブレターを用いてサーバが表記されます。この例で示したブラウザがリモート ホストに接続していたなら、(図 4-11 に示したような `localhost:7501` ではなく) そのホストの完全解決済み URL が右ペイン上部の黒いバーの 「**接続先**」という見出しの次に表示されていることでしょう。

図 4-11　サーバの選択

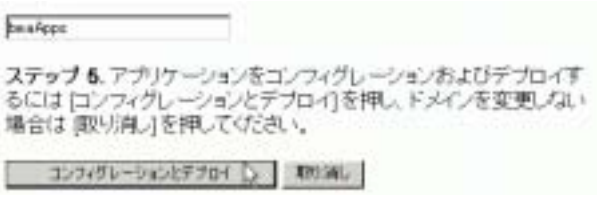


4. 図 4-12 の [ステップ 3] に示すように、2 つのリストの間にある右向き矢印をクリックして、サーバを右側の [対象サーバ] リストに移動します。

図 4-12　[対象サーバ] リストへのサーバの移動

5. 図 4-12 の [ステップ 4] に示すように、アプリケーション名を確認または編集します。
6. 図 4-13 の [ステップ 5] に示すように、[コンフィグレーションとデプロイ] をクリックします。

図 4-13　アプリケーションのコンフィグレーションとデプロイ



7. [**myNewDomain> アプリケーション > portalApp**] ページが表示されます。
8. デプロイメント プロセスの実行後、図 4-14 に示すように、ステータス メッセージがウィンドウ下部に表示され、デプロイメントに成功したことが通知されます。

図 4-14　デプロイメントの成功を示すメッセージ

9. [**対象別デプロイメント状況**] ページが表示されたら、新しいアプリケーションの [デプロイ] カラムに「**true**」というステータスが表示されていることを確認します。ステータスが「**false**」であれば、[**デプロイ**] をクリックし、プロセスが正常に完了するのを待ちます。

デプロイメントの検証

最後に、図 4-15 に示すように、`http:\\<hostname>:<port>\NewPWApp` にアクセスすることで、ポータルが正常にデプロイされたかどうかを確かめます。

注意 : この例では、WebLogic Portal Administration Tools を用いて表示対象に設定されたポートレットはないので、図 4-15 では、ポータルに何も表示されません。

図 4-15　デプロイされたポータル Web アプリケーションの表示

ポータル デプロイメントについてのベスト プラクティス ガイドライン

ポータル アプリケーションの開発、テスト、および公開には、多重環境モデルを用いることをお勧めします BEA Weblogic Platform には、デプロイメントを簡単にするためのツールやスクリプトが多数用意されています。これらのツールとベスト プラクティスについては、「WebLogic Server デプロイメント」(http://edocs.beasys.co.jp/e-docs/wls/docs70/programming/deploying.html) で詳しく説明されています。

以下の 3 段階モデルでポータルをデプロイすることをお勧めします。

第 1 段階 : ローカル マシン上のサーバにデプロイする

開発と単体テストは、このデプロイメント段階で行います 開発作業はすべてこの段階で済ませ、単体テスト終了時には次の段階へのデプロイを行うだけにすることを強くお勧めします。

注意 : 共有環境で開発と単体テストを行うことにした場合には、それぞれの開発者は各自のドメイン内で作業を行い、他の開発者用のデータベース情報を上書きしないようにする必要があります。

第 2 段階 : ローカル コンピュータからステージング サーバへデプロイする

開発と単体テストが完了したら、ステージング サーバにデプロイして例外テストを行います。問題のあるコードが見つかったら、ローカル マシンでそれらをすべて修正したあと、元のステージング サーバに再デプロイしなければなりません。

クラスタ テストが必要な場合は、ステージング サーバで行わなければなりません。

第 3 段階 : テスト環境から稼働中のプロダクション サーバにデプロイする

開発と例外テストがすべて完了したら、アプリケーションを稼働中のプロダクション サーバにデプロイすることができます。この時点でクラスタ デプロイメントを再度テストしたほうがよい場合もあります。

アプリケーションの追加開発分があれば、第 1 段階で説明したようにその開発と単体テストを行い、第 2 段階で説明したようにテストを行ったうえで、プロダクション サーバに再デプロイしなければなりません。

第 II 部 ポータルの拡張

ポータルがいったん動作するようになったら、機能を追加してその価値を高めることができます。ここでは、ポータルを拡張して、インタラクティブ、コンテンツが豊富、パーソナライゼーション可能という特徴を兼ね備えた、企業データおよびアプリケーションへの単一アクセス ポイントに変える方法を示します。

第 II 部では、以下の内容について説明します。

- カスタム テンプレートの作成
- ユーザ プロファイルの実装
- ポータルへのセキュリティの追加
- ポータル コンテンツ管理
- ポータル ナビゲーションのセットアップ
- ルック アンド フィールの作成
- ポートレットの拡張
- パーソナライゼーションおよび対話管理のセットアップ
- Campaign サービスのセットアップ
- Commerce サービスのセットアップ
- イベントおよび行動追跡
- Expression パッケージの使用
- イベント解説

第5章 カスタム テンプレートの作成

WebLogic Portal には、ドメイン、アプリケーション、ポータル、およびポートレットをすばやく効率的に作成するのに役立つウィザードとテンプレートが多数用意されています。この章では、独自のカスタム テンプレートを作成する方法の概要を説明します。ドメイン テンプレートとポータル テンプレートは Domain Wizard または Portal Wizard で使用され、グループ ポータル テンプレートは WebLogic Portal Administration Tools を用いて呼び出されます。

この章では、以下のトピックを扱います。

テンプレートの概要

BEA WebLogic Platform には、3 種類のテンプレートが組み込まれています。すなわち、ドメイン テンプレート、ポータル テンプレート、そして、グループ ポータル テンプレートです。ドメイン テンプレートとポータル テンプレートがウィザード コンポーネントとして作成されるのに対して、グループ ポータル テンプレートは WebLogic Portal Administration Tools で使用されます。

3 種類のテンプレート

WebLogic Platform 用テンプレートの作成方法を学ぶ前に、ポータル作成における各テンプレート タイプの役割について考えてみましょう。

ドメイン ウィザード テンプレート

定義済みのドメイン テンプレートは、JAR ファイルの形式で以下のディレクトリに格納されています。

```
<BEA_HOME>weblogic700\common\templates\domains
```

Domain Configuration Wizard を起動するには、[スタート｜プログラム｜ BEA WebLogic Platform 7.0] メニューから選択すればよく、その際にサーバが稼働している必要はありません。このウィザードは、一連の画面を通じてユーザから入力された情報を収集し、ディレクトリとファイルのコピーをいくつか実行したあと、作成対象となるドメイン内のコンフィグレーション ファイルと起動スクリプトに含まれているいくつかの文字列を置換します。そのため、ユーザが選択した設定はテンプレートに適用され、新しいドメイン ディレクトリ内にインスタンス化されます。

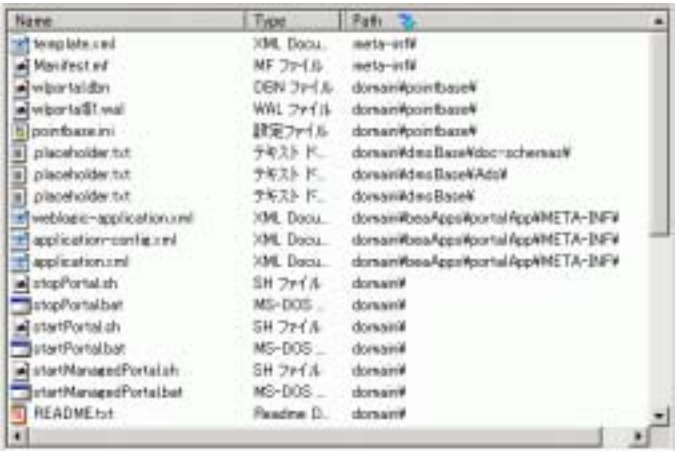
Domain Configuration Wizard でドメイン テンプレートがいくつか列挙されている様子を 図 5-1 に示します。

図 5-1　カスタム ドメインの選択

Domain Configuration Wizard では、template.xml ファイルの内容に基づいてテンプレートを認識します。そのファイルの編集に関しては、後で説明します。

図 5-2 は、株式ポータル用 WebLogic Portal ドメインの JAR を展開して、その中のファイルを一部示したものです。

図 5-2　WebLogic Portal ドメイン テンプレート ファイル



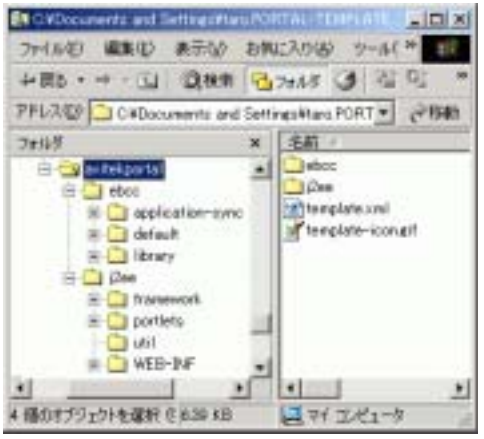
Domain Configuration Wizard では、`template.xml`、`config.xml`、および `application.xml` の各ファイルを用いて、リソースをコピーし、新しいドメイン内のコンフィグレーション ファイルとスクリプトの文字列置換を実行します。

ポータル ウィザード テンプレート

インストールされた WebLogic Portal には、定義済みのポータル テンプレートが 1 つ付属しています。図 5-3 に示すのはカスタム ポータル テンプレートの内容です。このテンプレートは以下のディレクトリに格納されているでしょう。

```
<BEA_HOME>\weblogic700\common\templates\webapps\portal
```

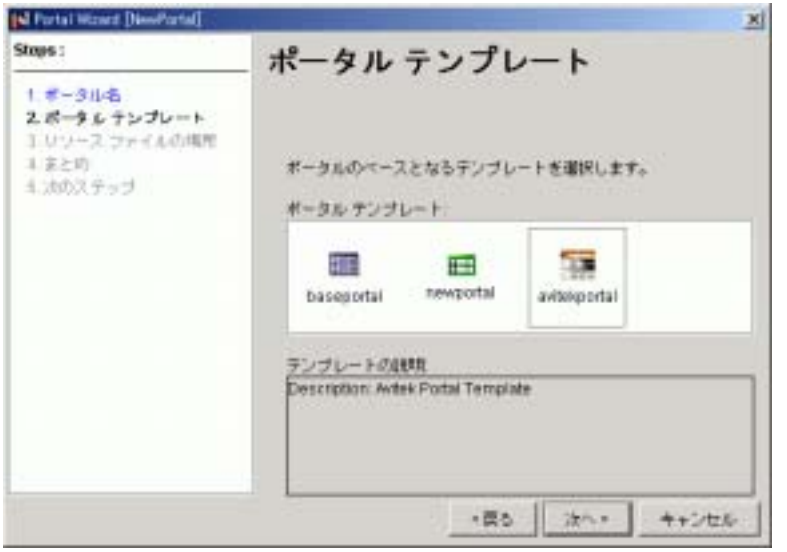
図 5-3　Avitek ポータル テンプレートのファイル構造



Domain Configuration Wizard と同様に、Portal Wizard でも `template.xml` というファイルを使用しますが、その他に、`template-icon.gif` も使用します。`template.xml` ファイルには、Portal Wizard で得られたユーザ入力を、このフォルダに含まれている J2EE リソースとメタデータ リソースに適用するためのディレクティブと変数が記述されています。

Portal Wizard では、図 5-4 に示すように、ポータル テンプレートは名前とアイコンで表示されます。

図 5-4　ポータル テンプレートの選択



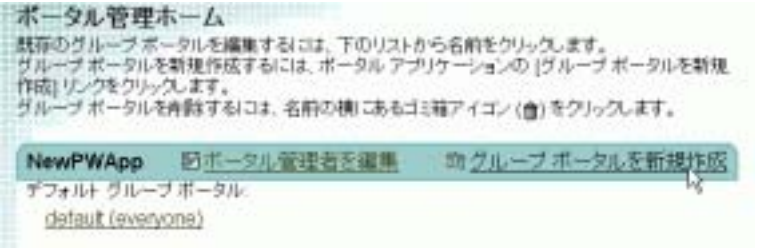
ポータル テンプレートを用いてポータルがインスタンス化されたあと、WebLogic Portal Administration Tools を用いて、そのポータルのページやポートレットといった構成要素の属性が設定されます。

グループ ポータル テンプレート

ドメイン テンプレートやポータル テンプレートとは異なり、グループ ポータル テンプレートはパッケージ化されたエンティティとして存在するわけではありません。テンプレートを用いてすべてのグループ ポータルを作成するためのメカニズムは、WebLogic Portal Administration Tools によって提供されるのです。あらかじめ用意されているのはデフォルト グループ ポータルだけで、これにはカスタマイズは施されていません。[ポータル管理ホーム] ページの [グループ ポータルを新規作成] リンクを 図 5-5 に示します。

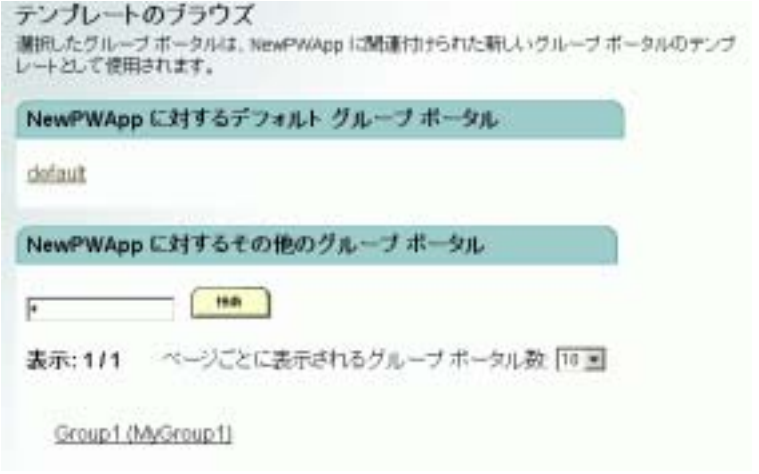
注意： グループ ポータル テンプレートと言っても、基本的にはグループ ポータルの再利用なので、このマニュアルではその作成については扱いません。グループ ポータルの作成方法の詳細については、「グループ ポータルの作成」というチュートリアルを参照してください。

図 5-5　グループ ポータルの作成



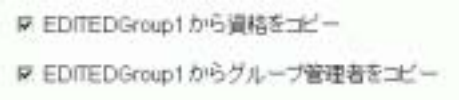
新しいグループ ポータルに名前を付け、ユーザ グループと関連付けたあとは、図 5-6 に示すように、グループ ポータル テンプレートを選択する必要があります。

図 5-6　グループ ポータル テンプレートの選択



こうして、どのようなグループ ポータルでも、新しいグループ ポータルのテンプレートになることができます。重要なのは、図 5-7 に示すように、資格と委託管理をグループ ポータル テンプレートからコピーできることです。

図 5-7　グループ ポータル テンプレートからの資格と委託管理のコピー



テンプレートの利用

ポータルのライフ サイクルには、およそ 3 通りのカスタマイズを取り入れることができます。すなわち、ポータルのレイアウトやスキンを作成することでルック＆フィールのカスタマイズを施すことができ、アプリケーション要素を追加することで機能をカスタマイズすることができ、そして、グループ ポータル テンプレートを用いてこれらのカスタム オブジェクトの一部を保存し普及させることができます。

各テンプレート タイプの利点は、以下のように、それぞれ異なります。

- カスタム ドメイン テンプレートを作成しておけば、組織内のすべての開発者にとって共通の出発点となる。こうしたものとしては、EJB などのエンタープライズ コンポーネント、スキンやロゴなどの CI (企業イメージ統一戦略) コンポーネント、さらには全社的に使用する企業ポートレットなどがあります。
- すでに存在する一連のポータル ページ、グループ ポータル、資格、ポートレット、スキン、およびレイアウトを複製する必要がある場合には、カスタム ポータル テンプレートを作成すればよい。
- ページ、ポートレット、資格、レイアウトなどのポータル リソースとユーザ グループとの管理上の関連を保持するには、グループ ポータル テンプレートを作成すればよい。

ドメイン テンプレートの作成

カスタム ポータル ドメイン テンプレートを作成する方法として最もよいのは、テンプレートを用いて作成された WebLogic Portal ドメインを出発点にすることです。アプリケーション機能の追加からポートレットへの資格の設定に至るまで、必要なカスタマイズはすべて実行し、所属組織において後任の開発者にカスタマイズを受け継いでいくメカニズムとしてドメイン テンプレートを使用します。

この節では、Domain Configuration Wizard 用のテンプレートを作成する基本手順を以下の順に説明します。

- ステップ 1: ポータル ドメインをインスタンス化する

ステップ 1: ポータル ドメインをインスタンス化する

まず、Domain Configuration Wizard を用いて WebLogic Portal ドメインをインスタンス化します。このステップの詳細については、「新規ドメインへのポータルの新規作成」というチュートリアルを参照してください。

ステップ 2: ポータル ドメインをカスタマイズする

Portal Wizard や Portlet Wizard を使用し、また、アプリケーション機能、ルック＆フィール コンポーネント、EJB などの J2EE コンポーネントを追加することで、ドメインをカスタマイズします。この節では、WebLogic Portal ドメインに機能を追加する方法を詳しく説明します。この節で説明するのは以下の手順ですが、これらはそれぞれ別々に適用することができます。

- 2 フェーズ デプロイメントのサポート
- ドメインへの全ポータル サービスの追加
- WebLogic Portal ドメインへの EJB の追加
- カスタム レイアウトをドメイン テンプレートに追加する
- カスタム スキンをドメイン テンプレートに追加する

2 フェーズ デプロイメントのサポート

テンプレートを使わずに独自のポータル Web アプリケーションを作成することにした場合には、コード リスト 5-1 に示す weblogic-application.xml ファイルを必ず以下のディレクトリに保存するようにしてください。

```
domain\beaApps\portalApp\META-INF\
```

2 フェーズ アプリケーション デプロイメントが正常に機能するには、このファイルが必要です。このファイルは、WebLogic Portal テンプレートを用いてアプリケーションを作成する際には自動的に生成されます。ウィザードを使わずにアプリケーションを作成する場合には、このファイルを手動で作成し、以下のエントリを追加する必要があります。

コード リスト 5-1 weblogic-application.xml

```
<!DOCTYPE weblogic-application PUBLIC "-//BEA Systems, Inc.//DTD WebLogic
Application 7.0.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic-application_1_0.dtd">

<weblogic-application>

  <application-param>

   <description>Required for deployment of portal

   applications</description>

   <param-name>weblogic.internal.listeners</param-name>

<param-value>com.bea.p13n.management.internal.lifecycle.J2EELifecycleListener</
param-value>

  </application-param>

</weblogic-application>
```

ドメインへの全ポータル サービスの追加

WebLogic Portal に付属しているテンプレートには、ポータル フレームワークで利用可能なアプリケーション機能のうち、コンテンツ管理サービス、パーソナライゼーション サービス、プレースホルダ サービス、および広告サービスを除くすべての機能が一部含まれています。

ドメインにこうした機能をすべて追加するには、以下の手順に従います。

1. Portal Wizard を使用して、新しいドメインにポータルとポータル Web アプリケーションを新規作成します。

2. コード リスト 5-2 に列挙したファイルを、新しいドメインの <webapp>\WEB-INF\lib ディレクトリにコピーします。
3. コード リスト 5-3 に示したエントリを <webapp>/WEB-INF 内の web.xml ファイルに挿入します。
4. コード リスト 5-4 に示したエントリを <webapp>/WEB-INF/weblogic.xml ファイルに追加します。

これで、このドメイン内のポータル Web アプリケーションは、WebLogic Platform に用意されているすべてのサービスをサポートすることになります。このドメインを Domain Configuration Wizard 用のテンプレートとして使用するには、このあと「ステップ 3: 全般的なコンフィグレーションを適用する」以降の手順に従います。

コード リスト 5-2　全ポータル サービスを追加するための JAR

```
BEA_HOME/weblogic700/portal/lib/commerce/web/cat_taglib.jar
BEA_HOME/weblogic700/portal/lib/commerce/web/eb_taglib.jar
BEA_HOME/weblogic700/portal/lib/commerce/web/productTracking_taglib.jar
BEA_HOME/weblogic700/portal/lib/p13n/web/ad_taglib.jar
BEA_HOME/weblogic700/portal/lib/p13n/web/cm_taglib.jar
BEA_HOME/weblogic700/portal/lib/p13n/web/ph_taglib.jar
BEA_HOME/weblogic700/portal/lib/p13n/web/ps_taglib.jar
BEA_HOME/weblogic700/portal/lib/p13n/web/pz_taglib.jar
BEA_HOME/weblogic700/portal/lib/p13n/web/tracking_taglib.jar
```

コード リスト 5-3　全ポータル サービスを追加するための web.xml 内エントリ

```
<!-- ファイル末尾の </web-app> エントリの直前に以下のテキストを追加する -->
<!-- クリック スルー イベントを発生させるためのフィルタ -->
<filter>
    <filter-name>ClickThroughEventFilter</filter-name>
```

```
<filter-class>com.bea.p13n.tracking.clickthrough.ClickThroughEventFilter</filte
r-class>
</filter>
<filter-mapping>
    <filter-name>ClickThroughEventFilter</filter-name>
    <url-pattern>/application/*</url-pattern>
</filter-mapping>
<!-- ShowDoc サーブレット -->
<servlet>
    <servlet-name>ShowDocServlet</servlet-name>
    <servlet-class>com.bea.p13n.content.servlets.ShowDocServlet</servlet-class>
    <!-- showdoc が常にローカルの ejb-ref DocumentManager を使用するように設定する -->
    <init-param>
        <param-name>contentHome</param-name>
        <param-value>java:comp/env/ejb/DocumentManager</param-value>
    </init-param>
</servlet>
<!-- AdClickThru サーブレット -->
<servlet>
    <servlet-name>adClickThru</servlet-name>
    <servlet-class>com.bea.p13n.ad.servlets.AdClickThruServlet</servlet-class>
</servlet>
<!-- ClickThrough サーブレット -->
<servlet>
    <servlet-name>clickThroughServlet</servlet-name>

<servlet-class>com.bea.p13n.tracking.clickthrough.ClickThroughServlet</servlet-
class>
```

```
</servlet>
<servlet-mapping>
    <servlet-name>ShowDocServlet</servlet-name>
    <url-pattern>/ShowDoc/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>adClickThru</servlet-name>
    <url-pattern>/adClickThru/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>adClickThru</servlet-name>
    <url-pattern>/AdClickThru/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>clickThroughServlet</servlet-name>
    <url-pattern>/clickThroughServlet/*</url-pattern>
</servlet-mapping>
<taglib>
    <taglib-uri>cat.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/cat_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>eb.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/eb_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>productTracking.tld</taglib-uri>
```

```
    <taglib-location>/WEB-INF/lib/productTracking_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>ad.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/ad_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>cm.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/cm_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>ph.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/ph_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>ps.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/ps_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>pz.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/pz_taglib.jar</taglib-location>
</taglib>
<taglib>
    <taglib-uri>tracking.tld</taglib-uri>
    <taglib-location>/WEB-INF/lib/tracking_taglib.jar</taglib-location>
</taglib>
<!-- これは、さまざまな <cm:> タグで使用される -->
```

```
<ejb-ref>
    <description>
The ContentManager EJB for this webapp
    </description>
    <ejb-ref-name>ejb/ContentManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.content.document.DocumentManagerHome</home>
    <remote>com.bea.p13n.content.document.DocumentManager</remote>
</ejb-ref>
<!-- これは ShowDocServlet で使用される -->
<ejb-ref>
    <description>
The DocumentManager for this webapp
    </description>
    <ejb-ref-name>ejb/DocumentManager</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.content.document.DocumentManagerHome</home>
    <remote>com.bea.p13n.content.document.DocumentManager</remote>
</ejb-ref>
<!-- これはプレースホルダ タグで使用される -->
<ejb-ref>
    <description>
The PlaceholderService Session EJB for the placeholder tag.
    </description>
    <ejb-ref-name>ejb/PlaceholderService</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.placeholder.PlaceholderServiceHome</home>
```

```
    <remote>com.bea.p13n.placeholder.PlaceholderService</remote>
</ejb-ref>
<!-- これは AdClickThruServlet と adTarget タグで使用される -->
<ejb-ref>
    <description>
The AdService for this webapp
    </description>
    <ejb-ref-name>ejb/AdService</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.ad.AdServiceHome</home>
    <remote>com.bea.p13n.ad.AdService</remote>
</ejb-ref>
<!-- これは AdClickThruServlet で使用される -->
<ejb-ref>
    <description>
The AdBucketService for this webapp
    </description>
    <ejb-ref-name>ejb/AdBucketService</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.ad.AdBucketServiceHome</home>
    <remote>com.bea.p13n.ad.AdBucketService</remote>
</ejb-ref>
<!-- これは、さまざまな <pz:> タグで使用される -->
<ejb-ref>
    <description>
The EjbAdvisor for this webapp
    </description>
```

```
    <ejb-ref-name>ejb/EjbAdvisor</ejb-ref-name>
    <ejb-ref-type>Session</ejb-ref-type>
    <home>com.bea.p13n.advisor.EjbAdvisorHome</home>
    <remote>com.bea.p13n.advisor.EjbAdvisor</remote>
</ejb-ref>
```

コード リスト 5-4　全ポータル サービスを追加するための weblogic.xml 内エントリ

```
<-- ファイルの冒頭付近にある <reference-descriptor> エントリの後に以下のテキストを追加する
-->
<ejb-reference-description>
    <ejb-ref-name>ejb/ContentManager</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.DocumentManager</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/DocumentManager</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.DocumentManager</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/PlaceholderService</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.PlaceholderService</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/AdService</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.AdService</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
```

```
    <ejb-ref-name>ejb/AdBucketService</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.AdBucketService</jndi-name>
</ejb-reference-description>
<ejb-reference-description>
    <ejb-ref-name>ejb/EjbAdvisor</ejb-ref-name>
    <jndi-name>${APPNAME}.BEA_personalization.EjbAdvisor</jndi-name>
</ejb-reference-description>
```

WebLogic Portal ドメインへの EJB の追加

たとえば、Enterprise JavaBeans (EJB) をドメインに追加するには、以下の手順に従います。

1. EJB を作成します。WebLogic Platform 用の EJB の作成方法については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』を参照してください。
2. EJB の JAR ファイルを以下のアプリケーション ディレクトリに格納します。

   ```
   <BEA_HOME>/yourdomain/BEAapps/portalApp/META-INF
   ```

3. この JAR ファイルへの参照を、以下の形式で `META-INF/application.xml` ファイルに挿入します。

   ```
   <module>
     <ejb>myEJB.jar</ejb>
   </module>
   ```

   ここで、`myEJB` は新しい EJB の名前です。

4. `domain/config.xml` のアプリケーション ノード内に、以下のようなエントリを追加します。

   ```
   <EJBComponent
        Name="myEJB"
        Targets="@TARGETS"
        URI="myEJB.jar"
   />
   ```

指定した EJB は、起動時にその他のすべてのアプリケーション機能と共にデプロイされることになります。

ステップ 3: 全般的なコンフィグレーションを適用する

エンタープライズ レベルまたは Web アプリケーション レベルでのアプリケーション機能の追加だけでなく、ポータル、ページ、ポートレット、さらにはスキンやレイアウトさえも、カスタマイズされた WebLogic Portal ドメイン テンプレートに挿入して、より全般的なカスタマイズを広く利用したい場合もあるでしょう。

ポータルとポートレットの作成方法の詳細については、『WebLogic Portal 開発者ガイド』を参照してください。グループ ポータルの作成方法の詳細については、『WebLogic Portal 管理者ガイド』の「チュートリアル - グループ ポータルの作成」という章を参照してください。

カスタム レイアウトおよびスキンの作成方法の詳細については、『WebLogic Portal 開発者ガイド』の「ルック アンド フィールの作成」という章を参照してください。この節では、ルック＆フィール コンポーネントを、カスタム WebLogic Portal ドメイン内にパッケージ化できるような形でドメインに追加する方法について説明します。

カスタム レイアウトをドメイン テンプレートに追加する

以下の例は、縦型 1 列配置の 「stack」というカスタム レイアウトを示します。このレイアウトは、コード リスト 5-5 に示す JSP ファイルと、図 5-8 に示すサムネイル画像から成ります。

図 5-8　stack レイアウト用の `thumbnail.gif`

コード リスト 5-5　stack レイアウトのテキスト

```
<%@ taglib uri='ren.tld' prefix='layout' %>
<layout:placePortletsinPlaceholder
placeholders="top,middle,bottom" />
<center>
    <table BORDER COLS="1" WIDTH="250" >
        <tr>
            <td>
                <layout:render section='top'/>
            </td>
        </tr>

        <tr>
            <td>
                <layout:render section='middle'/>
            </td>
        </tr>

        <tr>
            <td>
                <layout:render section='bottom'/>
            </td>
        </tr>
    </table>
</center>
```

カスタム レイアウトを作成したら、以下の手順に従って、それを新しいドメイン テンプレートで利用できるようにします。

1. このレイアウトをドメイン テンプレートに挿入するには、JSP ファイルを **template.jsp**、画像ファイルを **<yourlayoutName>.gif** として、以下のディレクトリに保存します。

   ```
   <BEA_HOME>\weblogic700\common\templates\domains\shared\bea\
   portal\projects\portalApp-project\library\portal\layouts\<y
   ourlayoutName>\
   ```

2. このカスタム レイアウトを E-Business Control Center から参照できるようにするには、JSP を **template.jsp**、画像ファイルを **thumbnail.gif** として、以下のディレクトリに保存します。

   ```
   <BEA_HOME>\weblogic700\common\templates\webapps\portal\newp
   ortal\j2ee\framework\layouts\<yourlayoutName>\
   ```

警告： レイアウト サムネイルのファイル名には一定の規則はありません。ファイル名が正しくなければ、E-Business Control Center または WebLogic Portal Administration Tools でプレビュー画像が表示されないことがあります。

カスタム スキンをドメイン テンプレートに追加する

カスタム スキンを作成したら、以下の手順に従って、そのカスタム スキンを新しいドメイン テンプレートで利用できるようにします。

1. ポータル上でスキンのスクリーンショットを撮り、それを 1 インチ幅に縮小して <yourlayoutName>.gif として保存します。

2. このサムネイルを以下のディレクトリに格納します。

   ```
   <BEA_HOME>\weblogic700\common\templates\domains\shared\bea\
   portal\projects\portalApp-project\library\portal\skins\<you
   rskinName>\
   ```

3. スキンの J2EE リソースを以下のディレクトリに格納します。

   ```
   <BEA_HOME>\weblogic700\common\templates\webapps\portal\newp
   ortal\j2ee\framework\skins\<yourskinName>\
   ```

これらの要素をドメインに追加したら、後は、Domain Configuration Wizard で使用してカスタム ドメインをインスタンス化できるような形に新しいドメインをパッケージ化するだけです。

ステップ 4: 新しいドメインをテンプレートとしてパッケージ化する

ドメインでのカスタマイズがすべて正常に追加されたことを確かめたら、その新しい機能をドメイン テンプレートの形で複製することができます。このテンプレートを用いれば、社内の開発者にとって作業の適切な基本線が与えられることになり、開発、デプロイメント、および保守の各プロセスのさまざまな局面で、作業の不必要な重複を減らすことができます。

これらの追加機能をサポートするためのエントリは、すでにいくつかのコンフィグレーション ファイルに作成されています。適切な J2EE リソースがテンプレートの JAR ファイルに正しくアーカイブされているとすれば、ウィザードでこれらのリソースを対象ドメインに組み込めるように、それらのメタデータを正しく入力することが非常に重要です。このため、`template.xml` ファイル内で、その他のコンフィグレーション ファイルや J2EE リソースの参照箇所に特に注意を払ってください。

注意： 新たに作成するドメイン テンプレートはどのようなものでも、すべてのポータル サービスをサポートしていなければなりません。このため、ドメインをテンプレート化する前に、「ステップ 2: ポータル ドメインをカスタマイズする」で示した手順に従うことを強くお勧めします。

カスタマイズされたドメインからテンプレートを作成するには、以下の手順に従います。

- template.xml ファイルを開く
- config.xml ファイルを編集する
- application.xml ファイルを編集する
- 文字列置換用のシェル スクリプトをチェックする
- アーカイブを作成する

template.xml ファイルを開く

以下の JAR ファイルをコピーすることで、`template.xml` ファイルのコピーを取得します。

```
<BEA_HOME>\weblogic700\common\templates\domains\portal.jar
```

このアーカイブを解凍し、template.xml ファイルをテキスト ブラウザで開きます。config.xml ファイルの編集時には、このファイルを参考にします。必要な編集をすべて行ったら、このファイルを (JAR 形式で) アーカイブして、\META-INF ディレクトリに抽出されるようにします。

config.xml ファイルを編集する

カスタマイズされたドメイン内の config.xml ファイルを開き、コード リスト 5-6 に示した置換を行います。設定済みの属性を template.xml で使用されている変数に置き換えることで、Domain Configuration Wizard では、ユーザ入力を自動的に config.xml ファイルの各フィールドに設定できるようになります。

コード リスト 5-6　congif.xml ファイルでのドメイン ノードの置換

```
<Server
    Name="portalServer"       "@SERVER_NAME" で置換
    ListenPort="7501"         "@LISTEN_PORT" で置換
    NativeIOEnabled="true"
    JavaCompiler="@JAVA_HOME/bin/javac"
    ServerVersion="7.0.1.0"
    StagingMode="nostage"
    TransactionLogFilePrefix="logs/"
>
<Log FileName="logs/weblogic.log"
    Name="portalServer"/      "@SERVER_NAME" で置換
>
<SSL Enabled="true"
   ListenPort="7502"          "@SSL_PORT" で置換
   Name="portalServer"       "@SERVER_NAME" で置換
   ServerCertificateChainFileName="ca.pem"
   ServerCertificateFileName="democert.pem"
   ServerKeyFileName="demokey.pem"/
```

```
>
<ServerStart Name="portalServer"/>  "@SERVER_NAME" で置換
 <WebServer
  DefaultWebApp="DefaultWebApp"
  LogFileName="access.log"
  LoggingEnabled="true"
  Name="portalServer"       "@SERVER_NAME" で置換
/>
```

注意： template.xml では、@TARGETS 値を用いることで、config.xml ファイルを修正せずに同じドメインをクラスタ、管理サーバ、あるいはスタンドアロン サーバにデプロイできるようになります。

```
 <change-pair name="TARGETS">
         <before string="@TARGETS" />
         <after string="$TARGET_NAMES$" />
       </change-pair>
```

Domain Wizard を実行してテンプレートを選択すると、コード リスト 5-7 に示すように変数エントリが置き換えられます。

コード リスト 5-7　config.xml ファイルでの変数の置換

```
ポータル テンプレート config.xml に元々記述されていたノード
<EJBComponent
    Name="campaign"
    Targets="@TARGETS"
    URI="campaign.jar"
/>
新しい config.xml ファイルでは変数に値が設定される
<EJBComponent
    Name="campaign"
    Targets="portalServer"
```

```
    URI="campaign.jar"
/>
```

application.xml ファイルを編集する

template.xml ファイルや config.xml ファイルの場合と同様に、まず、既存のテンプレートからサンプル ファイルを抽出するところから始めるのがよいでしょう。アーカイブ内のファイルとカスタマイズした application.xml ファイルを比較してください。サーバ起動時に自動的にデプロイしたい新規 Web アプリケーションごとに、application.xml の application ノードにエントリを 1 つ追加する必要があります。コード リスト 5-8 に示したのは、NewPWApp という Web モジュールを application ノードに挿入している例です。

コード リスト 5-8　application.xml へのモジュール リストの追加

```
<application>
 <module>
   <web>
     <web-uri>NewPWApp</web-uri>
     <context-root>NewPWApp</context-root>
   </web>
 </module>
...
</application>
```

文字列置換用のシェル スクリプトをチェックする

アプリケーションに特別な起動クラスや環境設定が必要な場合には、それらを必ずテンプレート内のシェル スクリプトに追加してください。コード リスト 5-9 に示したのは、定義済みのポータル ドメイン テンプレートに付属している startPortal.bat の内容です。

ドメインに起動コマンドのカスタマイズが必要な場合には、リテラル参照を追加してもスクリプトが保守しにくくならないようにしてください。

コード リスト 5-9 portal.jar 内の startPortal.bat

```
@ECHO OFF

SETLOCAL

REM #########################################################################

REM (c) 2002 BEA SYSTEMS INC. All rights reserved

REM

REM BEA WebLogic Portal Server 起動スクリプト

REM このスクリプトでは、ポータル ウィンドウ サービスのインストール / アンインストールを行うこともできる。それには、以下のコマンドライン引数を使用する。

REM -installService または -uninstallService

REM #########################################################################

REM #########################################################################

REM WLP のインストール ディレクトリ

REM #########################################################################

SET WLP_HOME=@BEA_PORTAL_HOME_BACK_SLASH@

REM #########################################################################

REM WebLogic サーバ名を設定する

REM #########################################################################

SET SERVER_NAME=@MANAGED_SERVER_REGISTERED_NAME_IN_ADMIN

IF "%SERVER_NAME%"=="" SET SERVER_NAME=@SERVER_NAME

REM #########################################################################

REM 管理対象ノードの場合には、WebLogic Admin Server の URL を設定する。

REM それ以外の場合には、次の変数は空欄のまま。

REM #########################################################################

set ADMIN_URL=@ADMIN_SERVER_URL

REM #########################################################################
```

```
REM データベース タイプを設定する
REM 有効な値は次のいずれか。POINTBASE、ORACLE_THIN、MSSQL、SYBASE_JCONNECT、DB2_TYPE2
REM 詳細については set-environment.bat を参照のこと。
REM ###########################################################################
SET DATABASE=@DATABASE@
REM db_settings.properties ファイルから取得を試みる
IF not exist .\db_settings.properties goto _setenv
SET DB_SETTINGS=.\db_settings.properties
FOR /F "eol=# tokens=1,2 delims==" %%i in (%DB_SETTINGS%) do  (
    if %%i == database SET DATABASE=%%j
)
:_setenv
REM ###########################################################################
REM 環境を設定する
REM 利用可能なパラメータの詳細については、set-environment.bat を参照のこと。
REM ###########################################################################
CALL "%WLP_HOME%\bin\win32\set-environment.bat"
REM ###########################################################################
REM 追加すべき CLASSPATH 情報があれば、ここで設定する。
REM ###########################################################################
SET
CLASSPATH=%CLASSPATH%;%P13N_DIR%\lib\commerce_system.jar;%P13N_DIR%\lib\campaig
n_system.jar
REM ###########################################################################
REM 上記パラメータで WebLogic を起動する
REM 利用可能なパラメータの詳細については、startWebLogic.cmd を参照のこと。
REM ###########################################################################
set MEM_ARGS=-Xms128m -Xmx128m -XX:MaxPermSize=128m
```

```
set JAVA_OPTIONS=-Dcommerce.properties="%WLP_HOME%\weblogiccommerce.properties"
if "%1" == "-installService" goto _installService
if "%1" == "-uninstallService" goto _uninstallService
:_startWebLogic
call "%P13N_DIR%\bin\win32\startWebLogic.cmd"
goto _the_end
:_installService
call "%P13N_DIR%\bin\win32\installWebLogicService.cmd"
goto _the_end
:_uninstallService
call "%P13N_DIR%\bin\win32\uninstallWebLogicService.cmd"
goto _the_end
:_the_end
ENDLOCAL
```

アーカイブを作成する

この段階で、ルック＆フィールのカスタマイズと機能はすべてポータル ドメインに追加されており、図 5-9 に示すようなディレクトリ構造になっています。

図 5-9　カスタム ポータル ドメインの展開表示

以下の手順に従って、アーカイブを作成します。

1. config.xml ファイルとシェル スクリプトがドメイン ディレクトリ内に存在し、application.xml ファイル、application-config.xml ファイル、および weblogic-application.xml ファイルが各エンタープライズ アプリケーション内の META-INF ディレクトリに含まれていることを確かめます。
2. 図 5-10 に示すように、アーカイブ フォルダのトップ レベルにファイルが存在しないことを確かめます。

図 5-10　アーカイブ化される前のドメイン テンプレート

3. アーカイブ フォルダでコマンド ラインから以下のコマンドを入力します。

```
jar -cfM ..\myportal.jar domain META-INF
```

4. このドメイン テンプレートを Domain Configuration Wizard で利用できるようにするには、JAR ファイルを以下のディレクトリに格納します。

```
<BEA_HOME>/weblogic700/common/templates/domains/
```

ポータル テンプレートの作成

ドメイン テンプレートの作成プロセスと同様に、ポータル テンプレートの作成も基本的には、ポータルの新しいインスタンスを作成してカスタマイズしたあと、リソースの格納場所が Portal Wizard にわかるような形でパッケージ化することです。

新規ポータルをインスタンス化する

Portal Wizard を用いたポータルの新規作成の詳細については、「ポータルの新規作成」の節を参照してください。

新規ポータルをカスタマイズする

ポータルの動作と外見をカスタマイズする人には、多くの情報が利用可能です。具体的な手順の概要とリンクについては、『WebLogic Portal 開発者ガイド』を参照してください。

基本コンフィグレーションを適用する

これで、グループ ポータル、ページ、ポートレット、資格、および委託管理設定の追加など、新規ポータルにコンフィグレーションを適用することができます。ポータルのコンフィグレーションの詳細については、『WebLogic Portal 管理者ガイド』を参照してください。

新規ポータルをテンプレートとしてパッケージ化する

Domain Configuration Wizard 用のテンプレートとは異なり、ポータル テンプレートは圧縮済みのアーカイブ ファイル形式で提供する必要はありません。ただし、新規ポータルのパッケージ化と言っても、`template.xml` ファイルの編集がほとんどです (このファイルの一例をコード リスト 5-12 に示します)。カスタマイズされたポータルを Portal Wizard 用のテンプレートとしてパッケージ化するには、以下の手順に従います。

ステップ 1: ステージング ディレクトリを作成する

`/myportal` というディレクトリを作成して、新しいポータル テンプレートのステージング フォルダにします。その中に 2 つのディレクトリを作成します。すなわち、`/j2ee` と `/ebcc` です。

ステップ 2: ソース ディレクトリを見つける

ソース ポータルに関連付けられている 2 つのディレクトリを見つけます。すなわち、図 5-11 に示すような J2EE リソースと、図 5-12 に示すようなメタデータ ディレクトリです。

- ドメイン内で、ポータル Web アプリケーションの名前を取って命名した Web アプリケーション ディレクトリを見つけます。この例では、Web アプリケーションの名前は NewPWApp です。
- それに対応するメタデータ ディレクトリの名前は portalApp-project です (エンタープライズ アプリケーションの名前を取って命名されています)。

図 5-11　NewPWApp 内の J2EE リソース ディレクトリ

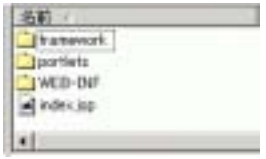


図 5-12　NewPWApp のメタデータ ディレクトリ

ステップ 3: ポータル リソースを移す

以下の手順で、リソースをテンプレート ステージング ディレクトリに移します。

- `NewPWApp` ディレクトリの内容を `myportal/j2ee` ディレクトリにコピーします。
- application-sync ディレクトリの内容を `myportal/ebcc` ディレクトリにコピーします。

注意： カスタマイズ時にポータルに追加したリソースのメタデータだけをコピーします。stockportal テンプレートの株式リソース メタデータ ファイルをコピーしないようにしてください。

ステップ 4: template.xml を編集する

以下の規則に従って、template.xml ファイルを編集します。

- コード リスト 5-10 に示すプロパティが Portal Wizard から渡される。
- コード リスト 5-11 に示すプロパティをポータルに合わせて設定する必要がある。
- カスタマイズされたポータル Web アプリケーションで必要な変更が他にあれば、それに合わせて `template.xml` ファイルを修正する。このファイルは ANT ビルド スクリプトです。コード リスト 5-12 に Portal Wizard 用の template.xml の全文を示します。

コード リスト 5-10　Portal Wizard から提供されるプロパティ

```
<property name="template.common.lib.root.dir" value="Path to
directory containing required jar files" />

<property name="template.ebcc.root.dir" value="Path to directory
containing application data directory" />

<property name="template.j2ee.webapp.root.dir" value="Path to web
application root directory" />

<property name="template.webapp.name" value="Name of the web
application"/>

<property name="template.portal.name" value="Portal web
application name"/>

<property name="template.portal.description" value="Description of
the portal application">
```

コード リスト 5-11　ポータル テンプレートに固有のプロパティ

```
<property name="template.name" value="baseportal" />

<property name="template.description" value="Description: Base
Portal Template" />
```

```
<property name="template.hyperlink.text" value="After your new
portal is deployed, follow these instructions..." />

<property name="template.hyperlink.url"
value="http://edocs.beasys.co.jp/e-docs/wlp/docs70/dev/newdom.htm
#1003370" />
```

ステップ 5: サムネイルを作成する

Portal Wizard に表示されるポータル テンプレートの外見を表すアイコンを作成します。そのファイルの名前を template-icon.gif とします。(このオプション ファイルがなければ、Portal Wizard では株式ポータル アイコンが表示されます。)

ステップ 6: アーカイブ ファイルを作成する

(省略可能) ポータル テンプレートの内容を圧縮します。すなわち、myportal ディレクトリから以下のコマンドを実行します。

```
jar -cfM ../myportal.jar *.*
```

ステップ 7: アーカイブを利用できるようにする

生成されたアーカイブ ファイルを以下のディレクトリに格納します。

```
<BEA_HOME>\weblogic700\common\templates\webapps\portal
```

コード リスト 5-12　ポータル用 template.xml

```
<?xml version="1.0"?>

<project name="Base Portal Template" default="main" basedir=".">

  <!--

    JAR またはディレクトリには、この template.xml ファイルが入っている必要があり、さらに、オプションとして

    template-icon.gif を中に入れることができる。 template-icon.gif が存在しなければ、

    デフォルト アイコンが表示される。
```

```
  -->


  <!-- 呼び出し側は以下のプロパティを渡さなければならない。

     <property name="template.common.lib.root.dir" value="Path to directory
containing required jar files" />

     <property name="template.ebcc.root.dir" value="Path to directory containing
application data directory" />

    <property name="template.j2ee.webapp.root.dir" value="Path to web application
root directory" />

     <property name="template.webapp.name" value="Name of the web application"/>

    <property name="template.portal.name" value="Portal web application name"/>

    <property name="template.portal.description" value="Description of the portal
application">

    <property name="template.hotdeploy.path" value="Path to directory containing
the portal for hot deploy"/>

     <property name="template.hotdeploy.user" value="User name for logging into
the server for hot deploy"/>

    <property name="template.hotdeploy.password" value="Password name for logging
into the server for hot deploy"/>


<property name="template.hotdeploy.adminurl" value="Server location for hot
deploy"/>

  -->


   <!-- テンプレート プロパティ -->

   <property name="template.name" value="baseportal" />

   <property name="template.version" value="1.0" />

   <property name="template.type" value="portal-webapp" />

   <property name="template.description" value="Description: Base Portal
Template" />

   <property name="template.hyperlink.text" value="After your new portal is
deployed, follow these instructions..." />
```

```
   <property name="template.hyperlink.url"
value="http://edocs.beasys.co.jp/e-docs/wlp/docs70/dev/newdom.htm#1003370" />


<!-- これは、.war ファイルかホット デプロイ用ディレクトリのどちらか。 -->

   <property name="template.hotdeploy.path"
value="${template.j2ee.webapp.root.dir}/${template.webapp.name}/"/>


   <target name="main" >

      <echo message="template.common.lib.root.dir (
${template.common.lib.root.dir} )"/>

      <echo message="template.ebcc.root.dir ( ${template.ebcc.root.dir} )"/>

     <echo message="template.j2ee.eapp.root.dir ( ${template.j2ee.eapp.root.dir}
)"/>

      <echo message="template.j2ee.webapp.root.dir (
${template.j2ee.webapp.root.dir} )"/>

      <echo message="template.appsync.dir ( ${template.appsync.dir} )"/>

      <echo message="template.webapp.name ( ${template.webapp.name} )"/>

      <echo message="template.portal.name ( ${template.portal.name} )"/>

     <echo message="template.hotdeploy.path ( ${template.hotdeploy.path} )"/>


      <!-- baseportal および tools Web アプリケーションを除くすべて -->


      <copy todir="${template.ebcc.root.dir}/"

            overwrite="no"

            preservelastmodified="yes"

            includeEmptyDirs="yes"

            filtering="no" >


         <fileset dir="ebcc/" >
```

```
            <include name="**" />
            <exclude name="application-sync/webapps/baseportal/" />
            <exclude name="application-sync/webapps/tools/" />
         </fileset>
      </copy>

      <!-- ここで、baseportal Web アプリケーション (baseportal.portal を除く ) をコピーし、ディレクトリの名前を任意の Web アプリケーション名に変更する。 -->
      <copy
todir="${template.ebcc.root.dir}/application-sync/webapps/${template.webapp.name}/"
            overwrite="no"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no" >

         <fileset dir="ebcc/application-sync/webapps/baseportal/" >
            <include name="**" />
            <exclude name="baseportal.portal" />
         </fileset>
      </copy>

      <filter token="template.portal.description"
value="${template.portal.description}" />

      <!-- ここで、baseportal.portal ファイルをコピーし、その名前を任意のポータル名に変更する。 -->
      <copy
tofile="${template.ebcc.root.dir}/application-sync/webapps/${template.webapp.name}/${template.portal.name}.portal"
```

```
            file="ebcc/application-sync/webapps/baseportal/baseportal.portal"
            overwrite="no"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no" >
      </copy>

      <!-- すべての J2EE リソースをコピーする -->
      <copy todir="${template.j2ee.webapp.root.dir}/${template.webapp.name}/"
            overwrite="no"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no" >

       <fileset dir="j2ee/" >
              <include name="**" />
              <exclude name="WEB-INF/weblogic.xml.stock"/>
              <exclude name="WEB-INF/web.xml.stock"/>
         </fileset>
      </copy>
     <filter token="template.portal.name" value="${template.portal.name}" />
      <filter token="template.webapp.name" value="${template.webapp.name}" />

      <copy
tofile="${template.j2ee.webapp.root.dir}/${template.webapp.name}/WEB-INF/weblog
ic.xml"
            overwrite="yes"
            preservelastmodified="yes"
```

```
            includeEmptyDirs="yes"
            filtering="no"
            file="j2ee/WEB-INF/weblogic.xml.stock">
      </copy>

      <copy
tofile="${template.j2ee.webapp.root.dir}/${template.webapp.name}/WEB-INF/web.xm
l"
            overwrite="yes"
            preservelastmodified="yes"
            includeEmptyDirs="yes"
            filtering="no"
            file="j2ee/WEB-INF/web.xml.stock">
      </copy>
   </target>
</project>
```

第 6 章　ユーザ プロファイルの実装

WebLogic Portal では、ユーザは*ユーザ プロファイル*で表されます。ユーザ プロファイルのおかげで、ユーザ属性の表現、格納、およびアクセスを柔軟に行えるようになります。WebLogic Portal には、基本的なユーザ プロファイルのサポートだけでなく、仮想エンタープライズ プロファイルの作成に利用できる*統合ユーザ プロファイル* (UUP) も用意されています。

この章では、以下の内容について説明します。

- 統合ユーザ プロファイルの作成
- プロパティ セット定義の作成

統合ユーザ プロファイルの作成

統合ユーザ プロファイルを用いれば、LDAP サーバ、レガシー システム、データベースなどの外部ソースに格納されているユーザ データを活用することができます。これによって、単一のプロファイルを使って、さまざまなソース内のユーザ データにアクセスできるようになります。

UUP を作成して外部ソースからユーザ データを取得するには、以下の作業を実行します。

1. 外部データを表現する EntityPropertyManager EJB を作成する
2. 新しい EntityPropertyManager を使用できる ProfileManager をデプロイする

外部データを表現する EntityPropertyManager EJB を作成する

外部ソースからデータを取り込むには、まず、com.bea.p13n.property.EntityPropertyManager リモート インタフェースのメソッドを実装するステートレス セッション Bean を作成する必要があります。EntityPropertyManager は、プロパティ データの永続性とプロファイル レコードの作成および削除を扱うセッション Bean のリモート インタフェースです。

さらに、このステートレス セッション Bean には、ホーム インタフェースと実装クラスが含まれていなければなりません。たとえば、以下のものです。

```
MyEntityPropertyManager
    extends com.bea.p13n.property.EntityPropertyManager

MyEntityPropertyManagerHome
    extends javax.ejb.EJBHome
```

実装クラスは、EntityPropertyManagerImpl クラスを拡張することができます。ただし、その実装クラスは、MyEntityPropertyManager リモート インタフェースの有効な実装であることが必要です。たとえば、以下のとおりです。

```
MyEntityPropertyManagerImpl extends
com.bea.p13n.property.internal.EntityPropertyManagerImpl
```

または

```
MyEntityPropertyManagerImpl extends
javax.ejb.SessionBean
```

推奨される EJB ガイドライン

新しい EJB についてのガイドラインとして、以下を推奨します。

- カスタム EntityPropertyManager はデフォルトの EntityPropertyManager ではありません。デフォルトの EntityPropertyManager は Portal スキーマ内のプロパティの取得 / 設定 / 削除に使用されます。カスタム EntityPropertyManager では、以下のメソッドをサポートする必要はありません。その代わりに、java.lang.UsupportedOperationException を送出することができます。

  getDynamicProperties

getEntityNames

getHomeName

getPropertyLocator

getUniqueId

- ポータル フレームワークとツールを使用して外部データ ストア内にユーザを作成または削除できるようにしたい場合には、createUniqueId() メソッドと removeEntity() メソッドをサポートする必要があります。ただし、カスタム EntityPropertyManager はデフォルトの EntityPropertyManager ではないので、用意する createUniqueId() メソッドはユニークな番号を返す必要はありません。外部データ ストアにユーザ エンティティを作成する必要がありますが、そのあとは、-1 などの任意の番号を返すことができます。

- EntityPropertyManager でサポートする必要があるメソッドについては、以下の推奨事項が当てはまります。

  - getProperty() – キャッシングを使用する。ユーザの全プロパティを一度に取得すると同時にそれらをキャッシングする getProperties メソッドをサポートしなければなりません。用意する getProperty メソッドでは、getProperties を使用しなければなりません。

  - setProperty() – キャッシングを使用する。

  - removeProperties()、removeProperty() – これらのメソッドが呼び出されたあとは、プロパティについて getProperty が呼び出されても、null を返さなければならない。キャッシュ内のプロパティも削除します。

- getProperty()、setProperty()、removeProperty()、および removeProperties() の各メソッドの実装には、外部システムへの接続に必要なロジックがすべて組み込まれていなければなりません。

- プロパティ データをキャッシュしたい場合には、上記の各メソッドでは、外部システムに適した形でプロファイル データをキャッシュできなければなりません（../javadoc/index.html にアクセスして com.bea.p13n.cache パッケージを参照のこと）。

- 外部システムに読み込み専用データが格納されている場合には、プロファイル データを変更するメソッドはすべて、java.lang.UnsupportedOperationException を送出する必要があります。さらに、外部データ ソースに、WebLogic Portal 以外のシステムで作成または削除されるユーザが含まれている場合には、createUniqueId メソッドと

removeEntity メソッドでは単に、UnsupportedOperationException を送出してかまいません。

- クラス ローダ依存関係の問題を回避するために、EJB を必ず専用のパッケージに入れるようにします。
- 保守しやすいように、カスタム EntityPropertyManager Bean のコンパイル済みクラスを独自の JAR ファイルに入れるようにします（既存の WebLogic Portal JAR ファイルを変更することはしない）。

JAR ファイルをデプロイする前に、次の節で示す手順に従います。

新しい EntityPropertyManager を使用できる ProfileManager をデプロイする

「ユーザ タイプ」とは、ProfileType 名から特定の ProfileManager へのマッピングのことです。このマッピングは、UserManagerEJB のデプロイメント記述子で行われます。

新しい EntityPropertyManager EJB 内のデータにアクセスするには、以下の作業の**いずれか 1 つ**を行う必要があります。

- ほとんどの場合には、ProfileManager のデフォルト デプロイメントである UserProfileManager を使用することができます。UserProfileManager のデプロイメント記述子を修正して、プロパティ セットとプロパティの両方あるいはどちらか一方をカスタム EntityPropertyManager にマップします。カスタム EntityPropertyManager で createUniqueId() メソッドと removeEntity() メソッドをサポートする場合には、WebLogic Portal Administration Tools を用いて、カスタム EntityPropertyManager を使ってプロパティを取得 / 設定できるプロファイルを持つ「User」というタイプのユーザを作成することができます。詳細については、6-5 ページの「既存の ProfileManager デプロイメントのコンフィグレーションを変更する」を参照してください。
- 時には、新たにコンフィグレーションされた ProfileManager をデプロイして、それを UserProfileManager の代わりに使用したほうがよい場合があります。こうした新しい ProfileManager は、UserManager のデプロイメント記述子で ProfileType にマップされます。カスタム EntityPropertyManager で createUniqueId() メソッドと removeEntity() メソッドをサポートする場合には、WebLogic Portal

Administration Tools（または API）を用いて、ProfileManager のカスタム デプロイメント（カスタム EntityPropertyManager を使用するようにコンフィグレーションされる）を使ってプロパティを取得 / 設定できる「MyUser」（任意の名前でよい）というタイプのユーザを作成することができます。詳細については、6-11 ページの「新しい ProfileManager のコンフィグレーションとデプロイ」を参照してください。

ProfileManager は、EntityPropertyManager EJB で取得されるプロファイル値へのアクセスを管理するステートレス セッション Bean です。この Bean は、自分のデプロイメント記述子に書かれた一連のマッピング文に基づいて、データを探します。たとえば、ProfileManager は DateOfBirth プロパティの値を要求するリクエストを受け取りますが、このプロパティは PersonalData プロパティ セットに入っています。ProfileManager は、自分のデプロイメント記述子に書かれているマッピング文を用いて、データがどの EntityPropertyManager EJB に格納されているかを判断します。

既存の ProfileManager デプロイメントのコンフィグレーションを変更する

既存の UserProfileManager デプロイメントを使用してユーザ プロファイルを管理する場合には、以下の手順を実行して、デプロイメントのコンフィグレーションを変更します。

ほとんどの状況では、この方法を UUP のデプロイに使用すべきです。この方法の例としては、カスタム EntityPropertyManager を LDAP プロパティ検索用にデプロイした LdapPropertyManager があります。LdapPropertyManager 用のクラスは ldapprofile.jar にパッケージ化されています。
UserProfileManager EJB のデプロイメント記述子は、「ldap」プロパティ セットを LdapPropertyManager にマップするようにコンフィグレーションされています。UserProfileManager は usermgmt.jar 内にデプロイされています。

1. エンタープライズ アプリケーションのルート ディレクトリ内の usermgmt.jar ファイルをバックアップします。
2. usermgmt.jar から META-INF/ejb-jar.xml を抽出し、編集用に開きます。
3. ejb-jar.xml 内で、コード リスト 6-1 に示すような <env-entry> 要素を見つけます。

コード リスト 6-1　<env-entry> Element

```
<!-- プロパティ セット ldap 内の全プロパティを ldap サーバにマップする
-->
<env-entry>
 <env-entry-name>PropertyMapping/ldap</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>LdapPropertyManager</env-entry-value>
</env-entry>
```

そして、この要素のあとに、コード リスト 6-2 に示すような <env-entry> 要素を追加して、プロパティ セットをカスタム EntityPropertyManager にマップします。

コード リスト 6-2　プロパティをマップする別の <env-entry> 要素の追加

```
<!-- UUPExample プロパティ セット内の全プロパティを
MyEntityPropertyManager にマップする -->
<env-entry>
  <env-entry-name>PropertyMapping/UUPExample</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

4. ejb-jar.xml 内で、コード リスト 6-3 に示すような <ejb-ref> 要素を見つけます。

コード リスト 6-3　<ejb-ref> 要素

```
<!-- ldap プロパティ マネージャ -->
<ejb-ref>
  <ejb-ref-name>ejb/LdapPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.bea.p13n.property.EntityPropertyManagerHome</home>
  <remote>com.bea.p13n.property.EntityPropertyManager</remote>
</ejb-ref>
```

そして、この要素のあとに、コード リスト 6-4 に示すような <ejb-ref> 要素を追加して、先ほどのステップで指定したマップ先の名前に ejb/ という接頭辞が付いた参照を EJB にマップします。

コード リスト 6-4　参照を EJB にマップするための <ejb-ref> 要素

```
<!-- プロパティ マネージャのサンプル -->
<ejb-ref>
  <ejb-ref-name>ejb/MyEntityPropertyManager</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>examples.usermgmt.MyEntityPropertyManagerHome</home>
  <remote>examples.usermgmt.MyEntityPropertyManager</remote>
</ejb-ref>
```

ホーム クラス名とリモート クラス名は、カスタム EntityPropertyManager の EJB JAR ファイルに入っているクラスと同じものです。

5. EntityPropertyManager 実装でプロファイル レコードの作成と削除を扱う場合には、Creator エントリと Remover エントリも追加する必要があります。たとえば、以下のようにします。

コード リスト 6-5　Creator エントリと Remover エントリを追加するための <env-entry> 要素

```
<env-entry>
  <env-entry-name>Creator/Creator1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>

<env-entry>
  <env-entry-name>Remover/Remover1</env-entry-name>
  <env-entry-type>java.lang.String</env-entry-type>
  <env-entry-value>MyEntityPropertyManager</env-entry-value>
</env-entry>
```

ここでは、ユーザ プロファイル レコードの作成時または削除時にはカスタム EntityPropertyManager を呼び出すように、UserProfileManager に指

示しています。「Creator1」および「Remover1」は任意の名前でかまいません。UserProfileManager によるユーザ プロファイルの作成時または削除時には、すべての Creator と Remover について反復処理が行われます。Creator と Remover の値は、ejb/ が先頭に付いていない点を除けば、カスタム EntityPropertyManager の ejb-ref-name 値と同じです。

6. usermgmt.jar から META-INF/weblogic-ejb-jar.xml を抽出し、編集用に開きます。
7. weblogic-ejb-jar.xml 内で、コード リスト 6-6 に示すような要素を見つけます。

コード リスト 6-6　weblogic-ejb-jar.xml 内の要素

```
<weblogic-enterprise-bean>
  <ejb-name>UserProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      EntityPropertyManager</jndi-name>
    </ejb-reference-description>
```

そして、カスタム EntityPropertyManager の ejb-ref を JNDI 名にマップするための ejb-reference-description を追加します。この JNDI 名は、カスタム EntityPropertyManager の JAR ファイル内の weblogic-ejb-jar.xml で割り当てた名前に一致しなければなりません。すなわち、コード リスト 6-7 に示す例のようになるはずです。

コード リスト 6-7　JNDI 名の指定

```
<weblogic-enterprise-bean>
  <ejb-name>UserProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      EntityPropertyManager</jndi-name>
    </ejb-reference-description>
    <ejb-reference-description>
```

```
      <ejb-ref-name>ejb/MyEntityPropertyManager
      </ejb-ref-name>
      <jndi-name>${APPNAME}.BEA_personalization.
      MyEntityPropertyManager</jndi-name>
    </ejb-reference-description>
```

${APPNAME} という文字列置換変数に注意してください。WebLogic EJB コンテナでは、この変数をエンタープライズ アプリケーション名に自動的に置き換えて、JNDI 名のスコープをそのアプリケーションに限定します。

8. デプロイメント記述子の変更に合わせて、usermgmt.jar を更新します。jar uf コマンドを使用して、META-INF/ 内のデプロイメント記述子の変更を JAR ファイルに反映させることができます。
9. エンタープライズ アプリケーションの META-INF/application.xml を編集して、カスタム EntityPropertyManager EJB モジュールのエントリ（コード リスト 6-8 に示す）を追加します。

コード リスト 6-8 カスタム EntityPropertyManager EJB モジュールのエントリの追加

```
<module>
  <ejb>UUPExample.jar</ejb>
</module>
```

10. アプリケーション全体のキャッシュを使用する場合には、以下のようなキャッシュ用の <Cache> タグをエンタープライズ アプリケーションの META-INF/application-config.xml デプロイメント記述子に追加すれば、Administration Console でキャッシュを管理することができます。

コード リスト 6-9 META-INF/application-config.xml への <Cache> タグの追加

```
<Cache
  Name="UUPExampleCache"
  TimeToLive="60000"
/>
```

11. 修正済みの usermgmt.jar とカスタム EntityPropertyManager EJB JAR アーカイブがエンタープライズ アプリケーションのルート ディレクトリ内にあることを確かめ、WebLogic Server を起動します。

12. WebLogic Server Administration Console を用いて、新規 EJB モジュールがエンタープライズ アプリケーションにデプロイされていることを確かめたあと、同コンソールを用いて、サーバをその EJB モジュールの ターゲットとして追加します。ターゲットを選択して、EJB モジュールをサーバにデプロイするようにドメインの config.xml ファイルを更新させる必要があります。

13. E-Business Control Center を用いて、（usermgmt.jar 内の）UserProfileManager の ejb-jar.xml でカスタム EntityPropertyManager にマップしたプロパティ セットの名前 に一致するユーザ プロファイル（プロパティ セット）を作成します。さらに、プロパティ セット内の特定のプロパティ名をカスタム EntityPropertyManager にマップすることさえできます。

    注意： プロパティ セットを作成したあとは、新しいデータを必ずサーバに同期化してください。

これで、新しい統合ユーザ プロファイル タイプがいつでも使用できるようになりました。WebLogic Portal Administration Tools を用いれば、「User」というタイプのユーザを作成することができ、そのユーザ タイプでは、「UUPExample」プロパティ セットの変更時に新しい UUP 実装が使用されることになります。UserManager に対して createUser("bob", "password") または createUser("bob", "password", null) を呼び出すと、以下の処理が行われます。

- 「bob」というユーザがセキュリティ レルム内に作成される。
- 「bob」の WebLogic Portal Server プロファイル レコードが WebLogic Portal RDBMS リポジトリに作成される。
- Creator マッピングをセットアップしてある場合には、UserManager がデフォルトの ProfileManager デプロイメント (UserProfileManager) を呼び出し、今度はそれがカスタム EntityPropertyManager を呼び出して、データ ソース内に Bob のレコードを作成する。

- Bob のプロファイルを取得する際には、デフォルトの ProfileManager デプロイメント (UserProfileManager) が使用され、「UUPExample」プロパティ セットに属するプロパティを要求すると、そのリクエストはカスタム EntityPropertyManager 実装に転送される。

新しい ProfileManager のコンフィグレーションとデプロイ

デフォルトの ProfileManager (UserProfileManager) を使うのではなく新たにコンフィグレーションした ProfileManager をデプロイして、ユーザ プロファイルを管理する場合には、以下の手順を実行してデプロイメントのコンフィグレーションを変更します。ほとんどの場合、このデプロイメント方法を用いる必要はありません。この方法を用いるのは、ProfileType に基づいてプロパティ セットをさまざまなカスタム EntityPropertyManager にマップできるようにするさまざまな ProfileManager デプロイメントを必要とする複数のユーザ タイプをサポートしなければならない場合だけです。

この方法の例としては、customer.jar 内のカスタム CustomerProfileManager のデプロイメントがあります。CustomerProfileManager は、「CustomerProperties」プロパティ セット内のプロパティに対してカスタム EntityPropertyManager (CustomerPropertyManager) を使用するようにコンフィグレーションされています。usermgmt.jar 内の UserManager EJB は、「WLCS_Customer」ProfileType を、ProfileManager のカスタム デプロイメントである CustomerProfileManager にマップするようにコンフィグレーションされています。

新しい ProfileManager をコンフィグレーションしデプロイするには、以下の手順に従います。

1. エンタープライズ アプリケーションのルート ディレクトリ内の usermgmt.jar ファイルをバックアップします。
2. usermgmt.jar から META-INF/ejb-jar.xml を抽出し、編集用に開きます。
3. ejb-jar.xml 内で、UserProfileManager の <session> タグ全体をコピーし、ProfileManager の新しいデプロイメントにカスタム実装クラスを使用するように、そのタグをコンフィグレーションします。

   さらに、UserProfileManager のホーム インタフェースとリモート インタフェースを再パッケージ化して独自のパッケージ化（たとえば、examples.usermgmt.MyProfileManagerHome、

examples.usermgmt.MyProfileManager）に合わせたい場合には、それらを独自のインタフェースで拡張することさえできます。

ただし、以下のようにして、Bean 実装クラスを入れ替えるだけで十分です。

<env-entry> 要素を作成して、プロパティ セットをカスタム EntityPropertyManager にマップする必要があります。さらに、<ejb-ref> 要素を作成して、プロパティ マッピング (PropertyMapping) で指定したマップ先の名前に ejb/ という接頭辞が付いた参照を EJB にマップする必要もあります。カスタム EntityPropertyManager のホーム クラス名とリモート クラス名は、カスタム EntityPropertyManager の EJB JAR ファイル内のクラスに一致します。

EntityPropertyManager 実装でプロファイル レコードの作成と削除を扱う場合には、Creator エントリと Remover エントリも追加する必要があります。これによって、ユーザ プロファイル レコードの作成時または削除時にはカスタム EntityPropertyManager を呼び出すように、新しい ProfileManager に指示することになります。

注意： Creator および Remover の後ろに付く「Creator1」および「Remover1」は任意の名前でかまいません。 ProfileManager によるユーザ プロファイルの作成時または削除時には、すべての Creator と Remover について反復処理が行われます。Creator と Remover の値は、ejb/ が先頭に付いていない点を除けば、カスタム EntityPropertyManager の <ejb-ref-name> 値と同じです。

4. ejb-jar.xml 内で、コード リスト 6-10 に示すように、UserManager EJB セクションに <ejb-ref> を追加して、ProfileType を ProfileManager の新しいデプロイメントにマップする必要があります。

コード リスト 6-10　UserManager EJB セクションへの <ejb-ref> の追加

```
<ejb-ref>
  <ejb-ref-name>ejb/ProfileType/UUPExampleUser</ejb-ref-name>
  <ejb-ref-type>Session</ejb-ref-type>
  <home>com.bea.p13n.usermgmt.profile.ProfileManagerHome</home>
  <remote>com.bea.p13n.usermgmt.profile.ProfileManager</remote>
</ejb-ref>
```

<ejb-ref-name> の先頭は ejb/ProfileType/ で、末尾は UserManager の createUser() メソッドの引数であるプロファイル タイプに使用したい名前でなければなりません。

5. usermgmt.jar から META-INF/weblogic-ejb-jar.xml を抽出し、編集用に開きます。
6. weblogic-ejb-jar.xml 内で、UserProfileManager の weblogic-enterprise-bean タグをコピーし、コード リスト 6-11 に示すように、それを新しい ProfileManager デプロイメント用にコンフィグレーションします。

コード リスト 6-11 新しい ProfileManager の <weblogic-enterprise-bean> タグ

```
<weblogic-enterprise-bean>
  <ejb-name>MyProfileManager</ejb-name>
  <reference-descriptor>
    <ejb-reference-description>
      <ejb-ref-name>ejb/EntityPropertyManager</ejb-ref-name>
        <jndi-name>${APPNAME}.BEA_personalization.
        EntityPropertyManager</jndi-name>
      </ejb-reference-description>
      <ejb-reference-description>
        <ejb-ref-name>ejb/PropertySetManager</ejb-ref-name>
        <jndi-name>${APPNAME}.BEA_personalization.
        PropertySetManager</jndi-name>
      </ejb-reference-description>
      <ejb-reference-description>
        <ejb-ref-name>ejb/MyEntityPropertyManager
        </ejb-ref-name>
        <jndi-name>${APPNAME}.BEA_personalization.
        MyEnitityPropertyManager</jndi-name>
      </ejb-reference-description>
  </reference-descriptor>
  <jndi-name>${APPNAME}.BEA_personalization.
  MyProfileManager</jndi-name>
</weblogic-enterprise-bean>
```

<ejb-reference-description> を作成して、カスタム EntityPropertyManager の <ejb-ref> を JNDI 名にマップする必要があります。この JNDI 名は、カスタム EntityPropertyManager の JAR ファイル

内の weblogic-ejb-jar.xml で割り当てた名前に一致しなければなりません。

${APPNAME} という文字列置換変数に注意してください。WebLogic Server EJB コンテナでは、この変数をエンタープライズ アプリケーション名に自動的に置き換えて、JNDI 名のスコープをそのアプリケーションに限定します。

7. weblogic-ejb-jar.xml 内で、UserProfileManager の <transaction-isolation> タグをコピーし、コード リスト 6-12 に示すように、それを新しい ProfileManager デプロイメント用にコンフィグレーションします。

コード リスト 6-12　新しい ProfileManager の <transaction-isolation> タグ

```
<transaction-isolation>
  <isolation-level>TRANSACTION_READ_COMMITTED
  </isolation-level>
  <method>
    <ejb-name>MyProfileManager</ejb-name>
    <method-name>*</method-name>
  </method>
</transaction-isolation>
```

8. 新しいデプロイメント記述子と新しい ProfileManager Bean 実装クラスを格納する一時的な usermgmt.jar を作成します。この一時的な EJB JAR アーカイブには、コンテナ クラスを入れてはいけません。ejbc を実行して、新しいコンテナ クラスを生成します。

9. エンタープライズ アプリケーションの META-INF/application.xml を編集して、カスタム EntityPropertyManager EJB モジュールのエントリ (コード リスト 6-13 に示す) を追加します。

コード リスト 6-13　カスタム EntityPropertyManager EJB モジュールのエントリの追加

```
<module>
  <ejb>UUPExample.jar</ejb>
</module>
```

10. アプリケーション全体のキャッシュを使用する場合には、コード リスト 6-14 に示すようなキャッシュ用の <Cache> タグをエンタープライズ アプリケーションの META-INF/application-config.xml デプロイメント記述子に追加すれば、WebLogic Server Administration Console でキャッシュを管理することができます。

コード リスト 6-14 META-INF/application-config.xml への <Cache> タグの追加

```
<Cache
  Name="UUPExampleCache"  TimeToLive="60000"
/>
```

11. 修正済みの usermgmt.jar とカスタム EntityPropertyManager EJB JAR アーカイブがエンタープライズ アプリケーションのルート ディレクトリ内にあることを確かめ、サーバを起動します。

12. WebLogic Server Administration Console を用いて、新規 EJB モジュールがエンタープライズ アプリケーションにデプロイされていることを確かめたあと、同コンソールを用いて、サーバをその EJB モジュールのターゲットとして追加します。ターゲットを選択して、EJB モジュールをサーバにデプロイするようにドメインの config.xml ファイルを更新させる必要があります。

13. E-Business Control Center を用いて、（usermgmt.jar 内の）UserProfileManager の ejb-jar.xml でカスタム EntityPropertyManager にマップしたプロパティ セットの名前 に一致するユーザ プロファイル（プロパティ セット）を作成します。さらに、プロパティ セット内の特定のプロパティ名をカスタム EntityPropertyManager にマップすることさえできます。

    注意： プロパティ セットを作成したあとは、新しいデータを必ずサーバに同期化してください。

14. これで、新しい統合ユーザ プロファイル タイプがいつでも使用できるようになりました。WebLogic Portal Administration Tools を用いれば、「UUPExampleUser」というタイプのユーザを作成することができ、そのユーザ タイプでは、「UUPExample」プロパティ セットの変更時に新しい UUP 実

装が使用されることになります。その理由は、UserManager デプロイメント記述子の中で、<ejb-ref> を使って ProfileType を「ejb/ProfileType/UUPExampleUser」とマップしたからです。

注意： WebLogic Portal Administration Tools でユーザを作成する際にこの新しいユーザ タイプを選択するように管理者に知らせてください。

ここで、UserManager に対して createUser("bob", "password", "UUPExampleUser") を呼び出すと、以下の処理が行われます。

- 「bob」というユーザがセキュリティ レルム内に作成される。
- 「bob」の WebLogic Portal Server プロファイル レコードが WebLogic Portal RDBMS リポジトリに作成される。
- Creator マッピングをセットアップしてある場合には、UserManager が新しい ProfileManager デプロイメントを呼び出し、今度はそれがカスタム EntityPropertyManager を呼び出して、データ ソース内に Bob のレコードを作成する。
- Bob のプロファイルを取得する際には、新しい ProfileManager デプロイメントが使用され、「UUPExample」プロパティ セットに属するプロパティを要求すると、そのリクエストはカスタム EntityPropertyManager 実装に転送される。

プロパティ セット定義の作成

プロパティ セットとは、パーソナライゼーション属性のスキーマのことです。プロパティ セットは、特定目的のプロパティ群に名前を付けるのに便利な手段です。たとえば、sampleportal-project の「Avitek」というユーザ プロファイルには、FirstName（名）、LastName（姓）、HomePhone（自宅電話番号）、Email（電子メール アドレス）、CustomerType（顧客種別）といった、e コマース顧客のプロパティを定義するプロパティ セットがあります。プロパティ セットの作成と、それらのプロパティ セットを構成するプロパティの定義には、E-Business Control Center を使用します。

この節では、カスタム ユーザ プロファイルの登録方法を説明します。

カスタム ユーザ プロファイルを登録する

プロパティ セット エディタは、すべてのプロパティ セットに対して同じように機能します。ここでは、E-Business Control Center を用いて、ユーザ プロファイル プロパティの作成と変更を行います。これらの例は、カスタム ユーザ プロファイルの登録に使用することができます。これと同じ手順に従って、イベント、HTTP リクエスト、HTTP セッション、およびカタログ構造のプロパティ セットを作成および変更することができます。

カスタム ユーザ プロファイルを登録するには、以下の手順を実行します。

1. E-Business Control Center を起動し、サーバに接続していることを確かめます。E-Business Control Center の起動とサーバへの接続については、『管理者ガイド』の「システム管理」**(**http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/sysadmin.htm**)** を参照してください。

   図 6-1 に示すような エクスプローラ ウィンドウが開きます。

図 6-1　E-Business Control Center ウィンドウ

2. 該当するプロジェクト ファイルを開きます。この手順では、例として、samples | portal | samplePortalDomain | beaApps | sampleportal-project を開きます。
3. 以下のようにして、イベント エディタを開きます。
   a. エクスプローラ ウィンドウで、[**ユーザ プロファイル**] アイコンを選択します。図 6-2 に示すように、ユーザ プロファイルのリストが [ユーザ プロファイル] フィールドに表示されます。

図 6-2 [ユーザ プロファイル] アイコン選択時の E-Business Control Center のエクスプローラ

b. [**新規作成**] アイコンをクリックして [**新規作成**] メニューを開き、コード リスト 6-3 に示すように、[**ユーザ プロファイル**] を選択します。

図 6-3 [新規作成] メニュー（[新規作成] アイコンをクリックすると開く）

図 6-4 に示すようなユーザ プロファイル エディタ ウィンドウが開きます。

図 6-4　ユーザ プロファイル エディタ ウィンドウ

4. [**新規作成**] をクリックします。

   [プロパティの編集] ウィンドウが表示されます（図 6-5）。

図 6-5　[プロパティの編集] ウィンドウ

5. [プロパティの編集] ウィンドウで、以下の手順を実行します。

   a. [**名前**] フィールドに、プロパティのユニークな名前を 100 文字以内で入力します（必須）。

      警告 : [名前] フィールドに「LDAP」と入力してはいけません。

b. [**概要**] フィールドに、プロパティの概要を 254 文字以内で入力します（省略可能）。

c. [**データ型**] リストで、プロパティのデータ型を選択します。データ型としてたとえば「ブール」を選択した場合には、[**選択モード**] と [**値の範囲**] は利用できません。ブール型のデフォルトは、「単値、制限付き」です。

d. [**選択モード**] リストで、[**単値**] か [**多値**] のいずれかを選択します。ここで選択した値によって、設定できるプロパティ値の数が決まります。すなわち、1 つ（**単値の場合**）か複数（**多値の場合**）です。

e. [値の範囲] リストで、値が [**制限付き**] なのか [**制限なし**] なのかを選択します。

f. [**値の追加**] をクリックします。

2 種類の [プロパティ値の入力] ウィンドウのいずれかが表示されます。どちらのタイプの [プロパティ値の入力 .] ウィンドウが表示されるかは、選択した値で決まります。その理由は、値の入力とデフォルト値の設定に必要な手順がデータ型によって異なるからです。以下のプロパティ カテゴリが利用できます。

- ブール型または単値かつ単一デフォルトのプロパティ
- 多値かつ単一デフォルト、複数デフォルト、全デフォルトのプロパティ
- 日時値を取るプロパティ

ブール型または単値かつ単一デフォルトのプロパティ

ブール型プロパティまたは、単値および単一デフォルト（制限なし）を持つプロパティのデフォルト値を入力するには、以下の手順を実行します。

1. 該当する [プロパティ値の入力] ウィンドウ（図 6-6 または図 6-7）で、以下のいずれかを実行します。
   - ブール型プロパティの場合には、[**True**] か [**False**] のどちらかを選択する。
   - 単値かつ単一デフォルトのプロパティの場合には、値を入力し、[**追加**] をクリックする。

図 6-6　[プロパティ値の入力] ウィンドウ（ブール値の場合）

図 6-7　[プロパティ値の入力] ウィンドウ（単値および単一デフォルトの場合）

2. [**OK**] をクリックします。

   [プロパティ値の入力] ウィンドウが閉じ、選択した値が、たとえばコードリスト 6-8 に示すように、[プロパティの編集] ウィンドウの [値] リストに表示されます。

図 6-8　テキスト値が選択された [プロパティの編集] ウィンドウ

3. [**OK**] をクリックします。

多値かつ単一デフォルト、複数デフォルト、全デフォルトのプロパティ

複数のプロパティ値を入力し任意の数のデフォルト（制限なし）を設定するには、以下の手順を実行します。

1. 該当する [プロパティ値の入力] ウィンドウで、値を入力したあと [**追加**] をクリックします。

   図 6-9、図 6-10、および 図 6-11 に示すように、新しい値が [値] リストボックスに表示されます。

図 6-9　[プロパティ値の入力] ウィンドウ（多値かつ単一デフォルトの場合）

図 6-10 [プロパティ値の入力] ウィンドウ（多値かつ制限付き複数デフォルトの場合）

図 6-11 [プロパティ値の入力] ウィンドウ（多値かつ制限なし複数デフォルトの場合）

2. 必要な値の入力がすべて済むまで、手順 1. を繰り返します。
3. 任意の数のデフォルト値を選択するには、以下のいずれかを実行します。
   - デフォルトを選択したくなければ、手順 5. に進む。
   - 多値かつ単一デフォルトの場合には、デフォルトとして設定したい値（ラジオ ボタン）を選択したあと [**OK**] をクリックする。

注意 : 多値かつ単一デフォルトのプロパティのデフォルト値を削除するには、[**すべて選択解除**] をクリックします。

- 多値かつ制限付き複数デフォルトの場合には、デフォルトとして設定したい値 (チェック ボックス) を選択したあと [**OK**] をクリックする。

注意 : 制限なしの多値（すなわち、[値の範囲] が [制限なし]）の場合には、デフォルトを選択する必要はありません。

4. [プロパティの編集] ウィンドウで [**OK**] をクリックします。

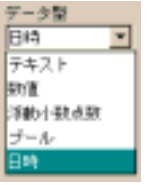
日時値を取るプロパティ

日時値を取るプロパティでは、[選択モード] と [値の範囲] のすべての設定が使えます。

日時値を入力し任意の数のデフォルトを設定するには、以下の手順を実行します。

1. [プロパティの編集] ウィンドウで、[**データ型**] ドロップダウン リストから [**日時**] を選択し（図 6-12 に示す）、[**値の追加**] を選択します。

図 6-12　[データ型] メニューでの [日時] の選択



日時値用の [プロパティ値の入力] ウィンドウが表示されます（図 6-13）。

図 6-13　日時の [プロパティ値の入力] ウィンドウ

2. [**日付**] リストのドロップダウン矢印をクリックします。

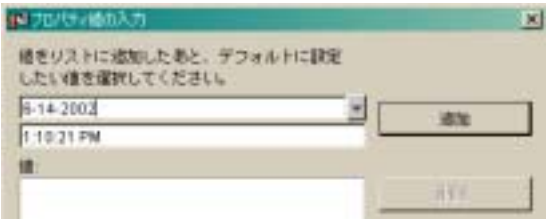
   図 6-14 に示すようなカレンダーが表示されます。

図 6-14　カレンダーが表示された [プロパティ値の入力] ウィンドウ

3. カレンダーから日付（たとえば 6 月 14 日）を選択します。

   カレンダーが閉じ、図 6-15 に示すように、選択した日付が日付編集ボックスに表示されます。

図 6-15　日付編集ボックスに表示される選択済みの日付



4. [**時刻**] フィールドに時刻を入力します。
5. [**追加**] をクリックします。

   図 6-16 に示すように、新しい日時が [値] リストに表示されます。

図 6-16　[値] リストに表示される日時

6. さらに日時を追加するには、必要な値の入力がすべて済むまで手順 1. から手順 5. を繰り返します。
7. 任意の数のデフォルト値を選択するには、以下のいずれかを実行します。
   - イベントの日時が単値かつ単一デフォルト（制限付き）の場合には、[**OK**] をクリックする。
   - イベントの日時が多値かつ単一デフォルト（制限付き）の場合には、デフォルトとして設定したい値（ラジオ ボタン）を選択したあと [**OK**] をクリックする。

- イベントの日時が多値かつ複数デフォルト（制限なし）の場合には、デフォルトとして設定したい値（チェック ボックス）を選択したあと [**OK**] をクリックする。

8. イベント用の [プロパティの編集] ウィンドウで、[**OK**] をクリックします。

登録済みカスタム イベントを更新する

カスタム イベントのコードに変更を加えるたびに、そのイベントの登録を更新しなければなりません。登録を更新すれば、カスタム イベントの変更が E-Business Control Center に認識され、キャンペーン開発者が E-Business Control Center を用いてそのイベントの参照元のシナリオ アクションをすべて修正するのに役立ちます。

カスタム イベントを更新するには、以下の手順を実行します。

1. E-Business Control Center を起動し、WebLogic Server に接続されていることを確かめます。

   エクスプローラ ウィンドウが開きます。

2. 正しいプロジェクト ファイルが開かれていることを確かめ、[**サイト インフラストラクチャ**] タブを選択します。

3. エクスプローラ ウィンドウで、[イベント] アイコンを選択します。図 6-17 に示すように、イベントのリストが [**イベント**] フィールドに表示されます。

   注意： 標準イベントは編集できません。

図 6-17　エクスプローラ ウィンドウ

4. 編集したいカスタム イベントをダブルクリックします。図 6-18 に示すようなイベント エディタ ウィンドウが開きます。[イベント プロパティ] フィールドに、既存プロパティのリストが表示されます。

図 6-18　イベント エディタ ウィンドウ

5. [**イベント プロパティ**] フィールドで、編集したいプロパティをダブルクリックします。

   図 6-19 に示すような [プロパティの編集] ウィンドウが開きます。

図 6-19 [プロパティの編集] ウィンドウ

6. [**データ型**]、[**選択モード**]、あるいは [**値の範囲**] を変更するには、該当するリスト ボックスから設定を選択します。

   注意 : [**データ型**]、[**選択モード**]、あるいは [**値の範囲**] の各プロパティ設定を変更すると、関連付けられている値は破棄されます。

7. 値を追加または変更するには、[**値の編集**] をクリックします。図 6-20 に示すような [プロパティ値の入力] ウィンドウが開きます。

図 6-20　[プロパティ値の入力] ウィンドウ

a. 値を削除するには、その値を選択してから [**削除**] をクリックします。

b. 値を追加するには、その値を入力してから [**追加**] をクリックします。

c. 値を変更するには、その値を選択し削除してから、新しい値を追加します。

d. 必要であれば、デフォルト値（複数の場合あり）を選択します。

e. 多値かつ単一デフォルトのプロパティのデフォルト値を削除するには、[**すべて選択解除**] をクリックします。

f. [**OK**] をクリックします。[プロパティ値の入力] ウィンドウが閉じます。

8. イベントのプロパティまたは値の更新が完了したら、イベントの [プロパティの編集] ウィンドウで [**OK**] をクリックします。

訪問者自主登録の有効化

Web サイトへの訪問者は多くの場合、引き続きサイトの機能を利用するには登録する必要があります。たとえば、オンライン書店では、訪問者は登録して初めて実際に書籍などの商品を買うことができるでしょう。登録は、それによって訪問者が Web サイトをより利用しやすくなるので有益な機能です。各取引に必要な訪問者関連情報（顧客プロファイルと呼ばれる）が格納され、取引のたびに訪

問者がそうした情報を再入力しなくても済むようになるからです。また、企業にとっても、訪問者データが格納されることで、自社のサービスを利用しそうな人々についての情報を維持管理できるようになるので便利です。

WebLogic Portal には、訪問者自主登録用の顧客プロファイルを作成するための JSP Webflow テンプレート一式が用意されています。これらのコンポーネントをそのまま使用することも、特定のニーズに合わせて修正することもできます。この節では、これらの JSP と Webflow コンポーネントについて説明し、その使用方法を解説します。

顧客プロファイル用 JSP を実装する

WebLogic Portal には、訪問者自主登録を機能させるのに使用できる 5 つの JSP テンプレートから成る Login and Registration サービスが用意されています。これらのテンプレートをそのまま使用することも、特定のニーズに合わせて修正することもできます。この節では、これらのテンプレートについて説明し、それらの実装方法を示します。

この節では、以下のテンプレートを取り上げます。

- login.jsp
- badlogin.jsp
- newuser.jsp
- newuserforward.jsp
- usercreationforward.jsp

login.jsp

`login.jsp` テンプレートは、まだサイトに登録していない顧客に、以後サイトを利用するための登録（新規顧客プロファイルの作成）を行うことのできるページへのエントリ ポイントを提供します。このページはチェックアウト プロセスへのエントリ ポイントなので、顧客がショッピングを続けられるようにするメカニズム（セッションなど）もこのページで設定されます。

解説

login.jsp テンプレートを用いて書式化された Web ページの例を図 6-21 に示します。

図 6-21　login.jsp を用いて書式化された Web ページ

login.jsp の動作の仕組み

未登録の顧客がポートレット内の [**Create**] をクリックした場合は、次にロードされるページ (newuser.jsp) で、顧客はプロファイルおよびユーザ名 / パスワードを作成できます。登録が済むと、顧客は自動的にログインし、続いて newusercreation.jsp テンプレートに移動します。そこでは顧客はショッピングを続けたり、ショッピング カートの内容を確認したり、チェックアウトすることができます。自動ログインに失敗すると、顧客にユーザ名とパスワードを入力させるために login.jsp テンプレートがロードされます。顧客がログインに失敗すると、badlogin.jsp がロードされます。

注意： チェックアウトに進むオプションは、顧客のショッピング カートに商品が入っている場合にのみ、newusercreation.jsp テンプレートに表示されます。

イベント

login.jsp テンプレートでは、顧客に 2 つのボタンが表示されますが、イベントと見なされるのは、そのうちの 1 つだけです。イベントが発生すると、デフォルト Webflow において特定の応答がトリガされて、顧客は操作を続けることができます。もう一方のボタンは HTML の標準の送信ボタンで、認証用に WebLogic Server にページを送り返すためのものです。イベントとそれによって呼び出されるビジネス ロジックを表 6-1 に示します。

表 6-1 login.jsp のイベント

イベント	Webflow の応答
button.createUser	ビジネス ロジックは不要。newuser.jsp をロードする。

[Log in] ボタンは Webflow の応答をトリガするイベントではありません。顧客がこのボタンをクリックすると、WebLogic Server（具体的には、WebLogic Portal の RDBMS レルム）に制御が移ります。WebLogic Server は HTTP リクエストを憶えておき、顧客のユーザ名とパスワードの組み合わせが正しいかどうかを判断したあと、そのリクエストを使って Webflow を再度呼び出します。

badlogin.jsp

badlogin.jsp テンプレート（図 6-22 に示す）は、ユーザ名とパスワードの無効な組み合わせが入力されたことを顧客に通知し、顧客がユーザ名とパスワードの有効な組み合わせを入力してサイトに再度ログインを試みることができるようにします。このテンプレートの動作は、エラー メッセージを除き、前述の login.jsp テンプレートとまったく同じです。

図 6-22　badlogin.jsp を用いて書式化された Web ページ

newuser.jsp

newuser.jsp テンプレート では、新規顧客が顧客プロファイルを作成して、e コマース サイトに登録できます。顧客プロファイルには、顧客の個人情報、届け先情報、支払い情報（省略可能）、およびアカウント情報が記載されます。

解説

newuser.jsp を用いて書式化された Web ページのブラウザでの表示例を示します。

図 6-23　newuser.jsp を用いて書式化された Web ページ — 個人情報

newuser.jsp の動作の仕組み

newuser.jsp は、顧客ログイン ページ (login.jsp) のあとに表示されるページです。顧客が最初のプロファイル情報を入力したときにエラーがなければ、顧客は自動的にログインしてウェルカム ページに移動します。顧客はそのページにあるさまざまなリンクを選択して、ショッピングを続けるかチェックアウトすることができます (newusercreation.jsp)。エラーがあった場合には、newuser.jsp が再ロードされ、無効なフォーム フィールドの脇に、そのエラーに対するメッセージが表示されます。

このテンプレートは、Webflow の sampleapp_user ネームスペースの一部です。

newuser.jsp にインクルードされる JSP テンプレート

newuser.jsp には、実装時に補助的 JSP テンプレートが 3 つインクルードされます。これらの JSP は、顧客に届け先住所、クレジット カード情報、およびデモグラフィック情報の入力を求めるすべての JSP テンプレートで行われるフォーム表示とエラー処理の両方についての標準フォーマットを提供します。以下では、これらのテンプレートについて 説明します。

newaddresstemplate.inc このテンプレートは、顧客に届け先住所の入力を求めるすべての JSP テンプレート（addaddress.jsp を除く）に組み込まれているフォーム フィールド表示とエラー処理の両方についての標準フォーマットを提供します。フォーム フィールドがテーブル形式で配置されており、フォームが送信されると、newaddresstemplate.inc テンプレートに関連付けられている入力プロセッサがフォームの有効性を検証して、すべての必須フィールドに値が入力されていることを確認します。エラーが検出された場合には、newaddresstemplate.inc テンプレートが再表示されます。このとき、画面の上部にエラー メッセージが表示され、無効なフィールドのラベルが赤で表示されます（本来のラベルの色は黒）。正しく入力された情報は、フォームにそのまま表示されます。

上記の動作は、newaddresstemplate.inc テンプレートで getValidatedValue JSP タグを用いて呼び出されます（コード リスト 6-15 に示す）。

コード リスト 6-15　newaddresstemplate.inc での getValidatedValue JSP タグの使い方

```
<!-- 顧客の届け先住所情報用のフィールドをまずテーブルに表示する  -->

<table width="90%" border="0">
   <tr>
      <td width="26%"><webflow:getValidatedValue
         fieldName="<%=HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1%>"
         fieldValue="customerShippingAddress1" fieldStatus="status"
         validColor="black" invalidColor="red" unspecifiedColor="black"
         fieldColor="fontColor" />
         <div class="tabletext"><font color=<%= fontColor %>><b>Address </b>
            </font>
         </div>
      </td>
      <td width="74%"> <input type="text"
         name="<%=HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1%>"
         value="<%=customerShippingAddress1%>" size="30" maxlength="30">*
```

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

newcctemplate.inc このテンプレートは、顧客にクレジット カード / 支払情報の入力を求めるすべての JSP テンプレートで行われるフォーム表示とエラー処理の両方についての標準フォーマットを提供します。フォーム フィールドがテーブル形式で配置されており、フォームが送信されると、newcctemplate.inc テンプレートに関連付けられている入力プロセッサがフォームの有効性を検証して、すべての必須フィールドに値が入力されていることを確認します。エラーが検出された場合には、newcctemplate.inc テンプレートが再表示されます。このとき、画面の上部にエラー メッセージが表示され、無効なフィールドのラベルが赤で表示されます（本来のラベルの色は黒）。正しく入力された情報は、フォームにそのまま表示されます。

上記の動作は、newcctemplate.inc テンプレートで getValidatedValue JSP タグを用いて呼び出されます（コード リスト 6-16 に示す）。

コード リスト 6-16　newcctemplate.inc での getValidatedValue JSP タグの使い方

```
<table>
.
.
.
   <td width="27%"><webflow:getValidatedValue
      fieldName="<%=HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER%>"
      fieldValue="customerCreditCardHolder" fieldStatus="status"
      validColor="black" invalidColor="red"
      unspecifiedColor="black"
      fieldColor="fontColor" />
      <div class="tabletext">
         <font color=<%= fontColor %>><b>Name on card</b>
         </font>
      </div>
   </td>
   <td width="73%"> <input type="text"
      name="<%=HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER%>"
```

```
      value="<%=customerCreditCardHolder%>" size="30" maxlength="50">*
   </td>
.
.
.
</table>
```

newdemographictemplate.inc　このテンプレートは、顧客にデモグラフィック情報の入力を求めるすべての JSP テンプレートで行われるフォーム表示とエラー処理の両方についての標準フォーマットを提供します。ラジオ ボタンがテーブル形式で配置されており、フォームが送信されると、newdemographictemplate.inc テンプレートに関連付けられている入力プロセッサがフォームの有効性を検証して、すべての必須フィールドに値が入力されていることを確認します。エラーが検出された場合には、newdemographictemplate.inc テンプレートが再表示されます。このとき、フォーム ページの上部にエラー メッセージが表示され、無効なフィールドのラベルが赤で表示されます（本来のラベルの色は黒）。正しく入力された情報は、フォームにそのまま表示されます。

上記の動作は、newdemographictemplate.inc テンプレートで getValidatedValue JSP タグを用いて呼び出されます（コード リスト 6-17 に示す）。

コード リスト 6-17　newdemographictemplate.inc での getValidatedValue JSP タグの使い方

```
<webflow:getValidatedValue fieldName="<%=HttpRequestConstants.CUSTOMER_GENDER%>"
   fieldDefaultValue="<%=(String)currentPropertyValue%>"
   fieldValue="genderValue" fieldStatus="status" validColor="black"
   invalidColor="red" unspecifiedColor="black" fieldColor="fontColor" />

   <td width="26%"><div class="tabletext"><b><font color=<%= fontColor %>>
      Gender*</font></b></div>
   </td>
   <td width="74%">

   <%// Gender ( 性別 ) のプロパティ値を取得する
   propertyBean.setPropertyName(GENDER);
   property = propertyBean.getPropertyObject();
   if(property == null || property.getRestrictedValues() == null)
```

```
arr = new Object[0];
else arr = property.getRestrictedValues().toArray();%>

<ps:getRestrictedPropertyValues propertySetName="Demographics"
   propertySetType="USER" propertyName="<%= GENDER %>" id="arr"
   result="foobar" />

<table width="100%" border="0" cellpadding="0"
   cellspacing="0"><es:forEachInArray id="valueObject" array="<%= arr %>"
      type="String">
   <tr>
      <td width="4%"><input type="radio" name="
      <%= HttpRequestConstants.CUSTOMER_GENDER %>" value="<%= valueObject %>"
      <% if ( valueObject.equals(genderValue) ) { %>CHECKED<% } %>></td>
      <td><%= valueObject %></td>
   </tr>
   </es:forEachInArray>
</table>
```

イベント

newuser.jsp テンプレートでは 2 つのボタンが顧客に表示され、そのそれぞれがイベントと見なされます。これらのイベントによってデフォルト Webflow の特定の応答がトリガされて、顧客は操作を続けることができます。この応答では、別の JSP がロードされることもありますが、通常は、入力プロセッサまたは Pipeline コンポーネントが先に呼び出されます。これらのイベントによって呼び出されるビジネス ロジックを 表 6-2 に示します。

表 6-2 newuser.jsp のイベント

イベント	Webflow の応答
button.cancel	GetCategoryIP GetTopCategories Pipeline
button.save	CustomerProfileIP CustomerProfile Pipeline

表 6-3 では、表 6-2 に示した各 Pipeline コンポーネントについて簡単に説明します。

表 6-3　newuser.jsp に関連付けられている Pipeline

Pipeline	解説
CustomerProfile	EncryptedCreditCardPC と RegisterUserPC から成り、トランザクション対応である。

newuser.jsp のフォーム フィールド

newuser.jsp テンプレートの主な目的は、HTML のさまざまなフォーム フィールドを用いて顧客からプロファイル情報を入力できるようにすることです。また、このテンプレートは、Webflow に必要な情報を渡すためにも使用されます。

newuser.jsp テンプレートで使用されるフォーム フィールド（大部分は他のテンプレートからインポートされる）と、そのそれぞれについての説明を表 6-4 にまとめます。

注意： フォーム フィールドが別のテンプレートからインポートされる場合には、解説欄にそう明記します。そうしたインポート情報の記載のないフォーム フィールドは、newuser.jsp テンプレート内にあります。

表 6-4　newuser.jsp のフォーム フィールド

パラメータ名	タイプ	解説
“event”	Hidden	どのイベントがトリガされたかを示す。次の処理を決定するために Webflow で使用。
“origin”	Hidden	現在のページ (newuser.jsp) の名前。Webflow で使用。
“namespace”	Hidden	JSP のネームスペース。この JSP では、sampleapp_user。
HttpRequestConstants. CUSTOMER_FIRST_NAME	Textbox	顧客のファーストネーム。
HttpRequestConstants. CUSTOMER_MIDDLE_NAME	Textbox	顧客のミドルネームのイニシャル。

表 6-4 newuser.jsp のフォーム フィールド （続き）

パラメータ名	タイプ	解説
HttpRequestConstants. CUSTOMER_LAST_NAME	Textbox	顧客の姓。
HttpRequestConstants. CUSTOMER_ADDRESS1	Textbox	顧客の住所の番地名以降の 1 行目。
HttpRequestConstants. CUSTOMER_ADDRESS2	Textbox	顧客の住所の番地名以降の 2 行目。
HttpRequestConstants. CUSTOMER_CITY	Textbox	顧客の住所の市区町村名。
HttpRequestConstants. CUSTOMER_STATE	Listbox	顧客の住所の州（都道府県）名。 states.inc からインポート。
HttpRequestConstants. CUSTOMER_ZIPCODE	Textbox	顧客の住所の郵便番号。
HttpRequestConstants. CUSTOMER_COUNTRY	Listbox	顧客の住所の国名。countries.inc からインポート。
HttpRequestConstants. CUSTOMER_HOME_PHONE	Textbox	顧客の自宅電話番号。
HttpRequestConstants. CUSTOMER_BUSINESS_PHONE	Textbox	顧客の勤務先電話番号。
HttpRequestConstants. CUSTOMER_EMAIL	Textbox	顧客の電子メール アドレス。
HttpRequestConstants. CUSTOMER_EMAIL_OPT_IN	Checkbox	顧客がプロモーション用電子メールの受信を希望していることを示す。
HttpRequestConstants. SAME_AS_ABOVE	Checkbox	顧客の届け先住所が連絡先住所と同じであることを示す。 newaddresstemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_SHIPPING_ADDRESS1	Textbox	顧客の届け先住所の番地名以降の 1 行目。 newaddresstemplate.inc からインポート。

表 6-4 newuser.jsp のフォーム フィールド （続き）

パラメータ名	タイプ	解説
HttpRequestConstants. CUSTOMER_SHIPPING_ADDRESS2	Textbox	顧客の届け先住所の番地名以降の 2 行目。newaddresstemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_SHIPPING_CITY	Textbox	顧客の届け先住所の市区町村名。newaddresstemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_SHIPPING_STATE	Listbox	顧客の届け先住所の州（都道府県）名。newaddresstemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_SHIPPING_ZIPCODE	Textbox	顧客の届け先住所の郵便番号。newaddresstemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_SHIPPING_COUNTRY	Listbox	顧客の届け先住所の国名。newaddresstemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_GENDER	Radio ボタン	顧客が男性か女性かを示す。newdemographictemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_DATE_OF_BIRTH	Textbox	顧客の生年月日。newdemographictemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_OCCUPATION	Radio ボタン	顧客の職業。newdemographictemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_EMPLOYMENT_STATUS	Radio ボタン	登録時に顧客が職に就いているかどうかを示す。 newdemographictemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_MARITAL_STATUS	Radio ボタン	顧客が結婚しているかどうかを示す。newdemographictemplate.inc からインポート。

表 6-4 newuser.jsp のフォーム フィールド （続き）

パラメータ名	タイプ	解説
HttpRequestConstants. CUSTOMER_EDUCATION_LEVEL	Radio ボタン	顧客の最終学歴を示す。 newdemographictemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_INCOME_RANGE	Radio ボタン	顧客の年収を示す。 newdemographictemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_QUALITY	Radio ボタン	商品使用に関する顧客のレベルを初心者から熟練者までの範囲で分類する。 newdemographictemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_TYPE	Listbox	顧客のクレジット カードの種類。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_HOLDER	Textbox	クレジット カードの名義人の氏名。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_NUMBER	Textbox	顧客のクレジット カード番号。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_MONTH	Listbox	顧客のクレジット カードの有効期限（月）。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_YEAR	Listbox	顧客のクレジット カードの有効期限（年）。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_ADDRESS1	Textbox	顧客の請求先住所の 1 行目。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_ ADDRESS2	Textbox	顧客の請求先住所の 2 行目。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_CITY	Textbox	顧客の請求先住所の市区町村名。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_STATE	Listbox	顧客の請求先住所の州（都道府県）名。 newcctemplate.inc からインポート。

表 6-4 newuser.jsp のフォーム フィールド （続き）

パラメータ名	タイプ	解説
HttpRequestConstants. CUSTOMER_CREDITCARD_ZIPCODE	Textbox	顧客の請求先住所の郵便番号。 newcctemplate.inc からインポート。
HttpRequestConstants. CUSTOMER_CREDITCARD_COUNTRY	Listbox	顧客の請求先住所の国名。 newcctemplate.inc からインポート。
HttpRequestConstants.USER_NAME	Textbox	顧客がログイン用に指定したユーザ名。
HttpRequestConstants.PASSWORD	Password	顧客がログイン用に指定したパスワード。
HttpRequestConstants. CONFIRM_PASSWORD	Password	顧客がログイン用に指定したパスワードの確認入力。

注意： JSP コード内でリテラルとして扱われるパラメータは、引用符で囲んで示されています。一方、リテラルでないパラメータを JSP 内で使用するには、スクリプトレット構文（たとえば、<%= HttpRequestConstants.USER_NAME %> など）に従う必要があります。

newusercreation.jsp

newusercreation.jsp テンプレートは、新しくユーザ プロファイルを作成した顧客に、ログインと登録が完了したことを知らせます。また、そこからショッピングに戻るためのいくつかのナビゲーション オプションも提供されます。

解説

newusercreation.jsp を用いて書式化された Web ページの例を 図 6-24 に示します。

図 6-24　newusercreation.jsp を用いて書式化された Web ページ

チェックアウトに進むオプションは、顧客のショッピング カートに商品が入っている場合にのみ、newusercreation.jsp テンプレートに表示されます。商品が入っていない場合には、図 6-25 に示すように、newusercreation.jsp テンプレートにこのオプションは表示されません。

図 6-25　newusercreation.jsp を用いて書式化された Web ページ（ショッピング カートが空の場合）

newusercreation.jsp の動作の仕組み

新規ユーザ プロファイルの作成が正常に終了し、JAAS (Java Authentication and Authorization Service) を用いた自動ログインが完了すると、顧客に対して newusercreation.jsp テンプレートが表示されます。顧客が新しくプロファイルを作成したあとで自動ログインが完了しなかった場合には、顧客に対して、newusercreation.jsp テンプレートではなく、login.jsp テンプレートが再び表示されます。手動ログインの場合には、顧客に対して main.jsp テンプレートが表示されます。

注意： 以前サイトを利用したときにプロファイルを作成してある顧客が login.jsp テンプレートを用いてログインした場合には、顧客がアクセスしようとしていた保護ページが表示されるだけです。

newusercreation.jsp テンプレートで顧客が行えることは、ショッピング カートに戻ること (shoppingcart.jsp)、ショッピングを続けること、チェックアウト プロセスに進むこと (shipping.jsp)、注文履歴を閲覧すること (orderhistory.jsp)、自分のプロファイルを閲覧すること (viewprofile.jsp)、支払履歴を閲覧すること (paymenthistory.jsp)、ログアウトすること、そしてメインのカタログ ページに戻ること (main.jsp) です。

注意： チェックアウトに進むオプションは、顧客のショッピング カートに商品が入っている場合にのみ、newusercreation.jsp テンプレートに表示されます。

このテンプレートは、Webflow の sampleapp_user ネームスペースの一部です。

イベント

顧客がボタンをクリックして注文の詳細を閲覧する操作は、すべてイベントと見なされます。イベントのたびに、デフォルト Webflow の特定の応答がトリガされて、顧客は操作を続けることができます。この応答では、別の JSP がロードされることもありますが、通常は、入力プロセッサと Pipeline の両方またはいずれか一方が先に起動されます。これらのイベントと、それによって呼び出されるビジネス ロジックを 表 6-5 に示します。

表 6-5　newusercreation.jsp のイベント

イベント	Webflow の応答
link.shoppingcart	InitShoppingCartIP
button.checkout	InitShippingMethodListIP
link.home	GetTopCategoriesIP GetTopCategories Pipeline

newuserforward.jsp

newuserforward.jsp テンプレートは、未登録のユーザが静的 URI の記述された広告プレースホルダをクリックしたときに、そのユーザを newuser.jsp に誘導するためのものです。これが必要なのは、プレースホルダで動的 URI がサポートされていないからです。newuserforward.jsp テンプレートによって、

ユーザには newuser.jsp が表示されます。さらに、newuserforward.jsp は、セキュリティの確保されていない接続からセキュリティの確保された接続への切り替え（.http から .https へ）の橋渡しも行います。

解説

このテンプレートは、Web ページなどの目に見える出力を表示しません。そのコードは、コード リスト 6-18 に示すとおりです。

コード リスト 6-18　newuserforward.jsp のコード

```
<% String s = com.bea.p13n.appflow.webflow.WebflowJSPHelper.
   createWebflowURL(pageContext, "sampleapp_main", "login.jsp",
   "button.createUser", true); %>

<% response.sendRedirect(s) ; %>
```

表 6-6 に主要なテンプレート コンポーネントを示します。

表 6-6 テンプレート コンポーネント

コンポーネントのタイプ	コンポーネント
インクルードされるテンプレート	なし
タグ ライブラリ	なし
インポートされる Java パッケージ	なし

newuserforward.jsp の動作の仕組み

newuserforward.jsp は、匿名ユーザからアクセスできる任意のページのあとに表示される可能性があります。ただし、このテンプレートが必要になるのは、未登録のユーザが広告プレースホルダをクリックし、登録するよう勧められた場合だけです。プレースホルダ内の静的 URI が newuserforward.jsp にアクセスし、今度はそのテンプレートによって、ユーザに newuser.jsp テンプレートが表示されます。

このテンプレートは、Webflow の sampleapp_main ネームスペースの一部です。

イベント

newuserforward.jsp テンプレートで扱われるイベントは 1 つで、このイベントによってデフォルト Webflow 内の特定の応答がトリガされ、顧客が処理を続行できるようになります。この応答では、別の JSP がロードされることもありますが、通常は、入力プロセッサまたは Pipeline が先に呼び出されます。このイベントとそれによって呼び出されるビジネス ロジックを表 6-7 に示します。

表 6-7　newuserforward.jsp のイベント

イベント	Webflow の応答
button.createUser	newuser.jsp

usercreationforward.jsp

usercreationforward.jsp テンプレートは、登録と、JAAS を用いた自動ログイン プロセスが Webflow で完了したあと、新規ユーザに newusercreation.jsp テンプレートを表示するためのものです。ユーザが作成されたらリクエストをフラッシュする必要がありますが、その処理は usercreationforward.jsp テンプレートで行われます。

解説

このテンプレートは、Web ページなどの目に見える出力を表示しません。そのコードは、コード リスト 6-19 に示すとおりです。

コード リスト 6-19　usercreationforward.jsp のコード

```
<% String s = WebflowJSPHelper.createWebflowURL(pageContext,
   "sampleapp_user", "usercreationforward.jsp",
   "forward.usercreation", true); %>

<% response.sendRedirect(s) ; %>
```

usercreationforward.jsp テンプレートでは、com.bea.p13n.appflow.webflow.WebflowJSPHelper パッケージ内の Java クラスを使用しており、以下のインポート文を含んでいる必要があります。

```
<%@ page import="com.bea.p13n.appflow.webflow.
   WebflowJSPHelper*" %>
```

usercreationforward.jsp の動作の仕組み

usercreationforward.jsp は、newuser.jsp テンプレートのあとに表示されるページです。新規ユーザは、自分のプロファイルを保存すると、JAAS により自動ログインします。ログインが正常に完了すると、以前のリクエストをフラッシュする必要があるので、ユーザを newusercreation.jsp テンプレートにリダイレクトするために usercreationforward.jsp が必要になります。

このテンプレートは、Webflow の sampleapp_user ネームスペースの一部です。

イベント

usercreationforward.jsp テンプレートで扱われるイベントは 1 つで、このイベントによってデフォルト Webflow 内の特定の応答がトリガされ、顧客が処理を続行できるようになります。この応答では、別の JSP がロードされることもありますが、通常は、入力プロセッサまたは Pipeline が先に呼び出されます。このイベントとそれによって呼び出されるビジネス ロジックを表 6-8 に示します。

表 6-8　usercreationforward.jsp のイベント

イベント	Webflow の応答
forward.usercreation	newusercreation.jsp

訪問者自主登録で用いられる Webflow コンポーネント

6-33 ページの「顧客プロファイル用 JSP を実装する」で説明したテンプレートでは、入力プロセッサおよび Pipeline と呼ばれる Webflow コンポーネントを用いて、訪問者自主登録を機能させるのに必要なビジネス ロジックの多くを実行します。この節では、実装されている主要な Webflow コンポーネントについて説明します。

以下のトピックスを扱います。

- 入力プロセッサ
- Pipeline コンポーネント

注意： Webflow の使用、作成、あるいは変更と、入力プロセッサおよび Pipeline コンポーネントの使用については、9-1 ページの「ポータル ナビゲーションのセットアップ」を参照してください。

入力プロセッサ

以下の入力プロセッサは、Webflow メカニズムから起動されたときに、より複雑な訪問者登録処理を実行するために呼び出される Java クラスを表します。

- CustomerProfileIP
- LoginCustomerIP

入力プロセッサの詳細については、9-3 ページの「ノードのタイプ」を参照してください。

CustomerProfileIP

`CustomerProfileIP`（入力プロセッサ名はすべて末尾に「IP」という文字が付く）は、`newuser.jsp` の入力を処理し、顧客が自分のプロファイルを保存できるようにします。さらに、`CustomerValue` オブジェクトを作成して Pipeline Processor セッションに格納します。

呼び出されるクラス	examples.wlcs.sampleapp.customer.webflow. CustomerProfileIP
HTTPServletRequest の必須パラメータ（個人情報）	HttpRequestConstants.CUSTOMER_FIRST_NAME HttpRequestConstants.CUSTOMER_MIDDLE_NAME HttpRequestConstants.CUSTOMER_LAST_NAME HttpRequestConstants.CUSTOMER_ADDRESS1 HttpRequestConstants.CUSTOMER_ADDRESS2 HttpRequestConstants.CUSTOMER_CITY HttpRequestConstants.CUSTOMER_STATE HttpRequestConstants.CUSTOMER_ZIPCODE HttpRequestConstants.CUSTOMER_COUNTRY HttpRequestConstants.CUSTOMER_HOME_PHONE HttpRequestConstants.CUSTOMER_BUSINESS_PHONE HttpRequestConstants.CUSTOMER_EMAIL HttpRequestConstants.CUSTOMER_EMAIL_OPT_IN （**コードの記載場所**：newuser.jsp テンプレート）
HTTPServletRequest の必須パラメータ（デモグラフィック情報）	HttpRequestConstants.CUSTOMER_INCOME_RANGE HttpRequestConstants.CUSTOMER_EDUCATION_LEVEL HttpRequestConstants.CUSTOMER_DATE_OF_BIRTH HttpRequestConstants.CUSTOMER_GENDER HttpRequestConstants.CUSTOMER_OCCUPATION HttpRequestConstants.CUSTOMER_MARITAL_STATUS HttpRequestConstants.CUSTOMER_EMPLOYMENT_STATUS HttpRequestConstants.CUSTOMER_QUALITY （**コードの記載場所**： newdemographictemplate.inc テンプレート）

HTTPServletRequest の必須パラメータ（届け先情報）	HttpRequestConstants.SAME_AS_ABOVE （**コードの記載場所**：newuser.jsp テンプレート）
	HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS1 HttpRequestConstants.CUSTOMER_SHIPPING_ADDRESS2 HttpRequestConstants.CUSTOMER_SHIPPING_CITY HttpRequestConstants.CUSTOMER_SHIPPING_STATE HttpRequestConstants.CUSTOMER_SHIPPING_ZIPCODE HttpRequestConstants.CUSTOMER_SHIPPING_COUNTRY HttpRequestConstants.DEFAULT_SHIPPING_ADDRESS （**コードの記載場所**：newaddresstemplate.inc テンプレート）
HTTPServletRequest のパラメータ（支払情報）	HttpRequestConstants.CUSTOMER_CREDITCARD_TYPE HttpRequestConstants.CUSTOMER_CREDITCARD_HOLDER HttpRequestConstants.CUSTOMER_CREDITCARD_NUMBER HttpRequestConstants.CUSTOMER_CREDITCARD_MONTH HttpRequestConstants.CUSTOMER_CREDITCARD_YEAR HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS1 HttpRequestConstants.CUSTOMER_CREDITCARD_ADDRESS2 HttpRequestConstants.CUSTOMER_CREDITCARD_CITY HttpRequestConstants.CUSTOMER_CREDITCARD_STATE HttpRequestConstants.CUSTOMER_CREDITCARD_ZIPCODE HttpRequestConstants.CUSTOMER_CREDITCARD_COUNTRY （**コードの記載場所**：newcctemplate.inc テンプレート）
HTTPServletRequest の必須パラメータ（アカウント情報）	HttpRequestConstants.USER_NAME HttpRequestConstants.PASSWORD HttpRequestConstants.CONFIRM_PASSWORD （**コードの記載場所**：newuser.jsp テンプレート）
必須の Pipeline セッション プロパティ	なし
更新される Pipeline セッション プロパティ	PipelineSessionConstants.CUSTOMER PipelineSessionConstants.PASSWORD PipelineSessionConstants.CREDITCARD_KEY（顧客がクレジット カード情報を更新する場合のみ）
削除される Pipeline セッション プロパティ	なし

検証	必須フィールドに値が入力されていること、およびクレジット カード番号が 16 桁未満（AMEX の場合は 15 桁未満）でないことを確認する。また、パスワード フィールドとパスワードの確認入力フィールドの値が一致することを確認する。
例外	必須フィールドに値が入力されていない場合、あるいはクレジット カード番号が 16 桁未満（AMEX の場合は 15 桁未満）の場合には、`InvalidInputException` が送出される。

LoginCustomerIP

`LoginCustomerIP` は、`login.jsp` の入力を処理し、サイトのセキュリティ保護付きページに顧客からアクセスできるようにします。さらに、`CustomerValue` オブジェクトを作成して Pipeline Processor セッションに格納します。

呼び出されるクラス	`examples.wlcs.sampleapp.customer.webflow.LoginCustomerIP`
`HTTPServletRequest` の必須パラメータ	なし
必須の Pipeline セッション プロパティ	PipelineSession`Constants.CUSTOMER` PipelineSession`Constants.PASSWORD` `PipelineSessionConstants.CREDITCARD_KEY`（顧客がクレジット カード情報を更新する場合のみ）
更新される Pipeline セッション プロパティ	なし
削除される Pipeline セッション プロパティ	PipelineSession`Constants.PASSWORD`
検証	ユーザ名とパスワードが正しいかどうか確認する。
例外	ユーザ名かパスワードが無効の場合には、`InvalidInputException` が送出される。 ユーザ名が無効か、認証を取得できない場合には、`ProcessingException` が送出される。

Pipeline コンポーネント

この節では、Customer Login and Registration サービスの JSP テンプレートに関連付けられている各 Pipeline コンポーネントについて簡単に説明します。これらの Pipeline は、訪問者登録に関係する特定タスクの実行を Webflow で開始する際に呼び出されるプロセッサ ノードです。

注意： Pipeline コンポーネントの中には、他の基本 Pipeline コンポーネントを拡張するものがあります。基本クラスの詳細については、『*Javadoc*』を参照してください。

Pipeline コンポーネントの詳細については、9-3 ページの「ノードのタイプ」を参照してください。

この節では、以下の Pipeline コンポーネントについて説明します。

- RegisterUserPC
- EncryptCreditCardPC

RegisterUserPC

`RegisterUserPC`（Pipeline コンポーネント名はすべて末尾に「PC」という文字が付く）は 、Pipeline Processor セッションから `CustomerValue` オブジェクトとパスワードを取得し、`CUSTOMER` 属性を作成します。

クラス名	`examples.wlcs.sampleapp.customer.`pipeline.`RegisterUserPC`
組み込み先	`CustomerProfile` Pipeline
必須の Pipeline セッション プロパティ	PipelineSession`Constants.CUSTOMER` PipelineSession`Constants.PASSWORD`
更新される Pipeline セッション プロパティ	なし
削除される Pipeline セッション プロパティ	PipelineSession`Constants.PASSWORD`
タイプ	Java クラス

JNDI 名	なし
例外	Pipeline コンポーネントでユーザを作成できない場合には、PipelineException が送出される。

EncryptCreditCardPC

EncryptCreditCardPC は、CREDITCARD_KEY オブジェクトを用いて顧客のクレジット カード情報を取得し、クレジット カード番号を暗号化したあと、修正済みのクレジット カード情報を PipelineSession CustomerValue 属性に追加し直します。

クラス名	examples.wlcs.sampleapp.customer.pipeline.EncryptCreditCardPC
解説	
組み込み先	CustomerProfile Pipeline
必須の Pipeline セッション プロパティ	PipelineSessionConstants.CREDITCARD_KEY
更新される Pipeline セッション プロパティ	PipelineSessionConstants.CUSTOMER
削除される Pipeline セッション プロパティ	PipelineSessionConstants.CREDITCARD_KEY
タイプ	Java クラス
JNDI 名	なし
例外	Pipeline コンポーネントが Pipeline Processor セッション内にユーザを見つけられないか、creditcard_key が無効か、あるいは暗号化が正常に終了しなかった場合には、PipelineException が送出される。

第 7 章 ポータルへのセキュリティの追加

Web サーバはユーザの認証を行い、ユーザがサーバ内のどのリソースを作成、アクセス、あるいは変更できるかを決定します。そのために、Web サーバはセキュリティ レルムを使用します。ユーザが特定のリソースにアクセスしようとすると、サーバは、アクセス制御リスト (ACL) とレルム内でそのユーザに割り当てられているパーミッションをチェックすることで、そのユーザの認証と認可を行おうとします。複数のセキュリティ レルムをセットアップできますが、Web サーバの各インスタンスでは 1 つのレルムしか使えません。サーバでは、Web サイト開発者用と訪問者用に同じセキュリティ レルムを使用します。

この章では、以下の内容について説明します。

ポータル セキュリティの実装

BEA から提供される RDBMS セキュリティ レルムの基本実装を使用することにした場合には、WebLogic Portal をインストールすれば、それが利用できるようになります。それ以上のコンフィグレーションは不要です。

注意： WebLogic Portal RDBMS レルムは、WebLogic Server RDBMS レルムとは異なる実装です。WebLogic Server RDBMS レルムはテスト用であり、運用環境での使用には適していません。WebLogic Portal の RDBMS レルムは、製品に付属しており

（BEA_HOME/weblogic700/portal/lib/p13n_system.jar ファイルに含まれている）、実装クラスは com.bea.p13n.security.realm.RDBMSRealm です。これは、config.xml ファイルの以下のエントリでコンフィグレーションされています。

```
RealmClassName="com.bea.p13n.security.realm.RDBMSRealm"
```

LDAP セキュリティ レルムとの統合

基本的な RDBMS セキュリティ レルムを使用したくない場合、代わりの方法としてよく行われるのは、LDAP (Lightweight Directory Access Protocol) サーバをセキュリティ レルムとして使用することです。この節では、LDAP サーバを WebLogic Portal に統合する方法について説明します。この節で紹介する手順では、LDAP 統合についての以下のシナリオを取り上げます。

- LDAP セキュリティ レルムと統合する
- LDAP セキュリティおよびユーザ プロファイルと統合する

サポート対象の LDAP サーバ

WebLogic Portal では、以下の LDAP サーバをサポートしています。

- Netscape Directory Server
- Microsoft Site Server
- Novell Directory Services
- Open LDAP Directory Services

これらの各サービスのテンプレートについては、7-4 ページの「サポート対象サーバ用のテンプレート」を参照してください。

LDAP セキュリティ レルムと統合する

この節では、WebLogic Portal をサードパーティ製の LDAP セキュリティ レルムと統合する方法を示します。セキュリティ レルムは、WebLogic Portal でユーザの認証に用いられる手段です。WebLogic Portal には RDBMS に基づいたデフォルトのユーザ ストアが用意されていますが、これを LDAP レルムに入れ替えることができ、LDAP レルムではセキュリティ情報の管理に LDAP サーバが使用されます。

LDAP サーバを統合用にコンフィグレーションする

LDAP セキュリティ レルムのコンフィグレーションには、WebLogic Server の LDAP セキュリティ レルムと LDAP サーバとの通信を可能にする属性と、LDAP ディレクトリへのユーザおよびグループの格納方法を記述する属性を定義することが必要になります。LDAP ツリーおよびスキーマは、LDAP サーバごとに異なります。

7-4 ページの「サポート対象サーバ用のテンプレート」には、サポート対象 LDAP サーバ用のテンプレートを示しています。これらのテンプレートは、サポート対象の各 LDAP サーバでユーザとグループを表現するのに用いられるデフォルトのコンフィグレーション情報を指定したものです。使用する LDAP サーバに該当するテンプレートを選択したあと、表 7-1 に示す属性に値を設定します。

注意： LDAP V1 では、[新しい LDAP Realm の作成] 画面の該当タブ（表 7-1 に明示）でこれらの属性をコンフィグレーションすることができます。

表 7-1 LDAP レルムのコンフィグレーション可能な属性

属性	解説
ユーザ DN	[ユーザ名属性] 属性と組み合わせて LDAP ユーザを一意に識別するための属性およびその属性値のリスト。 この属性は、[新しい LDAP Realm の作成] 画面の [ユーザ] タブでコンフィグレーションする。

表 7-1 LDAP レルムのコンフィグレーション可能な属性

属性	解説
グループ DN	[グループ名属性] 属性と組み合わせて LDAP ディレクトリ内のグループを一意に識別するための属性およびその属性値のリスト。 この属性は、[新しい LDAP Realm の作成] 画面の [グループ] タブでコンフィグレーションする。
プリンシパル	WebLogic Server で LDAP サーバへの接続に使用される LDAP ユーザの識別名（Distinguished Name: DN）。このユーザで LDAP ユーザおよびグループを列挙できる必要がある。 この属性は、[新しい LDAP Realm の作成] 画面の [LDAP レルム] タブでコンフィグレーションする。
資格	[プリンシパル] 属性に定義される LDAP ユーザを認証するためのパスワード。 この属性は、[新しい LDAP Realm の作成] 画面の [LDAP レルム] タブでコンフィグレーションする。

LDAP セキュリティ レルムのコンフィグレーション方法については、『WebLogic Server 管理者ガイド』の「LDAP セキュリティ レルムのコンフィグレーション」(http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/cnfgsec.html#338176) を参照してください。

サポート対象サーバ用のテンプレート

サポート対象 LDAP サーバのコンフィグレーションに使用できるテンプレートを、コード リスト 7-1 からコード リスト 7-4 に示します。ここで示したこれらのテンプレートを、アプリケーションの `config.xml` ファイルに直接コピーすることができます。

警告： 以下のサンプル コードの各行は 1 行で記述する必要があります。以下の例はこのマニュアルのマージンに合わせて整形されており、そのために分割されている行も一部あります。このテキストを `config.xml` ファイルにコピーする際には、分割されている行を必ず 1 行に連結してください。

コード リスト 7-1 デフォルトの Netscape カスタム セキュリティ レルム テンプレート

```
<CustomRealm
   Name="defaultLDAPRealmForNetscapeDirectoryServer"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=uid=
      admin,
   ou=Administrators, ou=TopologyManagement, o=NetscapeRoot;user.dn=ou=people,
   o=beasys.com;user.filter=(&amp;(uid=%u)(objectclass=person));group.dn=
      ou=groups,
   o=beasys.com;group.filter=(&amp;(cn=%g)(objectclass=groupofuniquenames));
      membership.filter=(&amp;(uniquemember=%M)(objectclass=
      groupofuniquenames));"
   Notes="This is provided as an example. Before enabling this Realm, you must
   edit the configuration parameters as appropriate for your environment."
/>
```

コード リスト 7-2 デフォルトの Microsoft カスタム セキュリティ レルム テンプレート

```
<CustomRealm
   Name="defaultLDAPRealmForMicrosoftSiteServer"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=cn=
      Administrator,
   ou=Members, o=ExampleMembershipDir;user.dn=ou=Members,
   o=ExampleMembershipDir;user.filter=(&amp;(cn=%u)(objectclass=member));
      group.dn=ou=Groups,
   o=ExampleMembershipDir;group.filter=(&amp;(cn=%g)(objectclass=mgroup));
      membership.scope.depth=1;microsoft.membership.scope=sub;membership.
      filter=(|(&amp;(memberobject=%M)(objectclass=memberof))
      (&amp;(groupobject=%M)(objectclass=groupmemberof)));"
   Notes="This is provided as an example. Before enabling this Realm,
      you must edit the configuration parameters as appropriate for your
      environment."
/>
```

コード リスト 7-3　デフォルトの Novell カスタム セキュリティ レルム テンプレート

```
<CustomRealm
   Name="defaultLDAPRealmForNovellDirectoryServices"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=cn=
      admin,
   o=example.com;user.dn=ou=people,
   o=example.com;user.filter=(&amp;(cn=%u)(objectclass=person));group.dn=ou=
      groups,
   o=example.com;group.filter=(&amp;(cn=%g)(objectclass=groupofuniquenames));
      membership.filter=(&amp;(member=%M)(objectclass=groupofuniquenames));"
   Notes="This is provided as an example. Before enabling this Realm, you must
      edit the configuration parameters as appropriate for your environment."
/>
```

コード リスト 7-4　デフォルトの OpenLDAP セキュリティ レルム テンプレート

```
<CustomRealm
   Name="defaultLDAPRealmForOpenLDAPDirectoryServices"
   RealmClassName="weblogic.security.ldaprealmv2.LDAPRealm"
   Password="*secret*"
   ConfigurationData="server.host=ldapserver.example.com;server.principal=cn=
      Manager,
   dc=example, dc=com;user.dn=ou=people, dc=example,
   dc=com;user.filter=(&amp;(uid=%u)(objectclass=person));group.dn=ou=groups,
   dc=example,
   c=com;group.filter=(&amp;(cn=%g)(objectclass=groupofuniquenames));membership.
      filter=(&amp;(uniquemember=%M)(objectclass=groupofuniquenames));"
   Notes="This is provided as an example. Before enabling this Realm, you must
      edit the configuration parameters as appropriate for your environment."
/>
```

LDAP セキュリティおよびユーザ プロファイルと統合する

この統合シナリオは、7-3 ページの「LDAP セキュリティ レルムと統合する」で示したものと同様ですが、セキュリティ レルムを統合するだけでなく、ユーザ プロファイル情報も統合する点が異なります。たとえば、LDAP サーバからユーザの電子メール属性を読み込み、それを統合ユーザ プロファイル (UUP) を通じてエクスポーズするのは、このシナリオの例でしょう。

あらかじめ必要な作業

ユーザ プロファイルを LDAP サーバから正常に取得するには、必ず以下を実行しておいてください。

1. WebLogic Portal インスタンスにアプリケーションをすでにデプロイしてある場合には、サーバを停止します。
2. `ldapprofile.jar` コンポーネントをアプリケーション内にデプロイします。
3. サーバを起動し、アプリケーションをデプロイします。
4. アクティブ ドメインに対して WebLogic Server Administration Console を起動します。

サーバを統合用にコンフィグレーションする

WebLogic サーバ を LDAP セキュリティ レルムおよびプロファイルとの統合用にコンフィグレーションするには、以下の 2 つのコンフィグレーション プロセスが必要になります。

- WebLogic Server の LDAP セキュリティ レルムと LDAP サーバとの通信を可能にする属性と、LDAP ディレクトリへのユーザおよびグループの格納方法を記述する属性を定義する。
- `ldapprofile.jar` ファイル内の接続変数をコンフィグレーションする。この Java アーカイブ ファイルによって、アプリケーションから、LDAP サーバに格納されているユーザおよびグループ プロファイル情報にアクセスできるようになります。コンフィグレーションの必要な変数を表 7-2 に示します。

これらの変数は、WebLogic Server Administration Console でコンフィグレーションすることができます。

表 7-2 コンフィグレーション可能な接続変数

接続変数	解説
config/serverURL	LDAP サーバの URL。
config/principal	LDAP サーバへの接続時に使用するプリンシパルの名前。通常は「admin」やディレクトリ管理者など。
config/principalCredential	上記プリンシパルのパスワード。
config/userDN	LDAP ディレクトリでのユーザの場所、すなわち識別名。
config/groupDN	LDAP ディレクトリでのグループの場所、すなわち識別名。
config/usernameAttribute	LDAP ディレクトリ内のユーザ名を記述するのに用いられる属性で、通常は「uid」。WLS ユーザ名から LDAP ユーザ名へのマッピングに使用される。
config/groupnameAttribute	LDAP ディレクトリ内のグループ名を記述するのに用いられる属性で、通常は「cn」。WLS グループ名から LDAP グループ名へのマッピングに使用される。

その他のサポート対象セキュリティ レルム

LDAP 以外に、WebLogic Server では以下のセキュリティ レルムがサポートされています。

- Windows NT セキュリティ レルム

  このセキュリティ レルムでは、Windows NT のアカウント情報を用いてユーザの認証を行います。Windows NT を通じて定義されたユーザとグループを Web アプリケーションで使用することができます。WebLogic Server

Administration Console を使用してこのレルムを参照することはできますが、ユーザやグループを定義するには Windows NT に用意されている機能を利用する必要があります。

- UNIX セキュリティ レルム

  UNIX セキュリティ レルムでは、ネイティブ プログラム `wlauth` を実行して、UNIX のログイン ID とパスワードでユーザとグループの認証を行います。UNIX プラットフォームによっては、プラグイン可能な認証モジュール (PAM: Pluggable Authentication Module) が `wlauth` で使用されます。このモジュールを使えば、UNIX プラットフォームにおける認証サービスを、それらのサービスを利用するアプリケーションを変更せずにコンフィグレーションすることができます。PAM が利用できない UNIX プラットフォームでは、シャドウ パスワード（サポートされている場合）などの標準のログイン メカニズムが `wlauth` で使用されます。Administration Console を使用してこのレルムを参照することはできますが、ユーザやグループを定義するには UNIX プラットフォームに用意されている機能を利用する必要があります。

- ファイル レルム

  サーバを起動すると、ファイル レルムは、WebLogic Server Administration Console を通じて定義されたプロパティに基づいて、ユーザ、グループ、および ACL の各オブジェクトを生成し、`fileRealm.properties` ファイルに格納します。

  注意： ファイル レルムは、10,000 ユーザ以内で使用するためのものです。10,000 ユーザを超える場合には、別のセキュリティ レルムを使用してください。

セキュア ソケット レイヤ セキュリティの有効化

WebLogic Portal の Commerce JSP テンプレートに関連付けられているプレゼンテーションとビジネス ロジックを制御する Webflow および Pipeline メカニズムでは、セキュア ソケット レイヤ (SSL) と宣言的転送のメカニズムが利用されます。保護された JSP ファイルを呼び出すリンクと、ある種の入力プロセッサおよび Pipeline には、HTTPS プロトコルでアクセスする必要があります。Commerce

(wlcs) Web アプリケーションの web.xml デプロイメント記述子には、これらのリンクがすでに多数定義されています。SSL に基づいたセキュリティ保護付き JSP テンプレートでも、転送の保証を示す設定が web.xml ファイルに必要になります。この保証の取り得る値は CONFIDENTIAL か INTEGRAL です。

- CONFIDENTIAL と設定すると、第三者から転送内容を見ることができなくなる
- INTEGRAL と設定すると、クライアント - サーバ間での転送中にデータを改ざんすることはできなくなる

Webflow および Pipeline については、第 9 章「ポータル ナビゲーションのセットアップ」を参照してください。

注意： SSL 接続が機能するためには、サーバ上にセットアップされた認証局から有効な SSL 証明書を取得しておく必要があります。

config.xml に必要な SSL 用の設定

Web アプリケーションで SSL を有効にするには、必ずドメインの config.xml ファイルに、SSL が有効になるように設定する必要があります（コード リスト 7-5 に示す）。

コード リスト 7-5　config.xml ファイルでの SSL の有効化

```
<server>
.
.
.
   <SSL Enabled="true" ListenPort="7502" Name="portalServer"
      ServerCertificateChainFileName="ca.pem"
      ServerCertificateFileName="democert.pem"
   ServerKeyFileName="demokey.pem"/>
</server>
```

SSL 属性には、必要な証明書ファイル名、サーバ キー ファイル名、および サーバ名も指定しなければなりません。

config.xml は、<BEA_HOME>/user_projects/<YOUR_DOMAIN> に格納されています。

ここで、<YOUR_DOMAIN> は Configuration Wizard の実行時に作成されたドメイン フォルダです。

web.xml に必要な SSL 用の設定

さらに、Web アプリケーションのデプロイメント記述子 (web.xml) に設定するセキュア リスン ポートが、コード リスト 7-6 に示すように、config.xml に設定したものと同じでなければなりません。

コード リスト 7-6　リスン ポートの指定

```
<context-param>
  <param-name>HTTPS_PORT</param-name>
  <param-value>7502</param-value>
</context-param>
```

シングル サインオンの有効化

シングル サインオンが有効になっていれば、訪問者は 1 回サインオンするだけで、複数の Web アプリケーションにアクセスすることができます（ただし、それらのアプリケーションが同一のサーバまたはクラスタ上で動作している場合に限ります）。シングル サインオンを有効にするには、以下の手順が必要です。

1. 各 Web アプリケーションに対するユーザのクッキーが同じになるようにします。
2. 各 Web アプリケーションに同じユーザ プロパティが使用されるようにします。

クッキー名を設定する

訪問者がシングル サインオンを利用してアクセスすることになる各アプリケーションのクッキー名を設定します。それには、該当する *アプリケーションごとに*、<BEA_HOME>weblogic700\samples\portal<PORTAL_DOMAIN>\beaApps\<PORTAL_APP>\<PORTAL>\WEB-INF\ 内の weblogic.xml ファイルを編集します。

1. 該当する weblogic.xml ファイルを開き、CookieName パラメータを設定する <session-param> 要素（コード リスト 7-7 に示す）を見つけます。

コード リスト 7-7

```
<session-param>
   <param-name>CookieName</param-name>
   <param-value>JSESSIONID_SAMPLEPORTAL</param-value>
</session-param>
```

2. <param-value> 値を適切なクッキー名に変更します。
3. Web アプリケーションごとに、ステップ 1 および 2 を繰り返します。

ユーザ プロパティを設定する

訪問者がシングル サインオンを利用してアクセスする各 Web アプリケーションでは、訪問者ごとに同じユーザ プロパティ セットを使用する必要があります。そのためには、該当するプロパティ セットを編集し、それらを一致させる必要があります。ユーザ プロパティ セットの編集の詳細については、6-16 ページの「プロパティ セット定義の作成」を参照してください。

第 8 章 ポータル コンテンツ管理

どのようなポータルであれ、重要なのはそのコンテンツです。WebLogic Portal では、コンテンツ マネージャを用いてコンテンツを提供します。コンテンツ マネージャには、パーソナライゼーション サービスでユーザに動的な Web コンテンツを提供するためのコンテンツおよびドキュメント管理機能が用意されています。コンテンツ マネージャが扱うのは、サードパーティ ベンダ製ツールで管理されるファイルやコンテンツです。ポータル リソースの開発時には、利用できる最も適切なコンテンツにユーザがアクセスできるように、コンテンツ マネージャのコンフィグレーションを行い、さまざまなコンテンツ関連タグを使用する必要があります。

この章では、以下のトピックを扱います。

- Bulk Loader を用いたコンテンツの追加
- コンテンツ セレクタ タグおよび関連する JSP タグの使い方
- 外部コンテンツ管理システムとの統合
- コンテンツ クエリの作成

Bulk Loader を用いたコンテンツの追加

ポータルにコンテンツを追加する最も簡単な方法は、製品に付属しているバルクローダ スクリプトを使用することです。この方法を実行するには、以下の手順に従います。

1. ファイル システム上のディレクトリに、指定された時間間隔でファイルをパブリッシュ（公開）します。該当するドメイン フォルダ内の `\dmsBase` ディレクトリのサブディレクトリにコンテンツをパブリッシュするのが理想的です。たとえば、広告であれば、以下の場所に格納します。

   ```
   <BEA_HOME>\weblogic700\samples\portal\<DOMAIN_NAME>\dmsBase\ads
   ```

2. ディレクトリ システム上のファイルにコンテンツをパブリッシュし終わったら、以下のスクリプトのいずれか一方を実行して BulkLoader を実行します。
   - テキスト ファイルや画像ファイルをロードする場合には、`loaddocs.bat`（UNIX ユーザの場合は `loaddocs.sh`）を実行する。
   - プレースホルダやキャンペーンの広告をロードする場合には、`loadads.bat`（UNIX ユーザの場合は `loadads.sh`）を実行する。

   このロード スクリプトは以下の場所にあります。

   ```
   <BEA_HOME>\weblogic700\samples\portal\<DOMAIN_NAME>\
   ```

   これらのスクリプトを実行するには、コマンド ライン（または、Windows NT/2000 では [**スタート**｜**ファイル名を指定して実行**] ）でそれらを入力するか、あるいは、Windows エクスプローラを開き、実行したいスクリプトを見つけ、ファイル リスト内でそれをダブルクリックします。コマンド ラインから BulkLoader を実行する場合には、表 8-1 に示した任意のコマンドライン スイッチを使用することもできます。

表 8-1 Bulk Loader のスイッチ設定

スイッチ設定	解説
`-verbose`	詳細なメッセージを表示する。
`+verbose`	実行時にメッセージを表示しない（デフォルト）。
`-recurse`	ディレクトリ内を再帰的に処理する（デフォルト）。
`+recurse`	ディレクトリ内を再帰的に処理しない。
`-delete`	データベースからドキュメントを削除する。
`+delete`	データベースにドキュメントを挿入する（デフォルト）。
`-metaparse`	HTML ファイルの `<meta>` タグを解析する（デフォルト）。
`+metaparse`	HTML ファイルの `<meta>` タグを解析しない。
`-cleanup`	指定された場合、`-d` 引数をドキュメント ベースとして、テーブルのクリーンアップだけを実行する（すべてのファイルはそのディレクトリ下に存在しなければならない）。

表 8-1 Bulk Loader のスイッチ設定

スイッチ設定	解説
+cleanup	テーブルのクリーンアップを実行せずに、ドキュメントをロードする（デフォルト）。
-hidden	隠しファイルおよび隠しディレクトリを無視するよう指定する（デフォルト）。
+hidden	隠しファイルおよび隠しディレクトリも処理の対象に含めるよう指定する。
-inheritProps	再帰処理時にメタデータ プロパティを継承するよう指定する（デフォルト）。
+inheritProps	再帰処理時にメタデータ プロパティを継承しないよう指定する。
-truncate	データ値がデータベースにとって大きすぎる（loader.properties で指定）場合、値の切り捨てを試みる。
+truncate	データ値の切り捨てを試みない（デフォルト）。
-ignoreErrors	ドキュメントのロード中は、エラーをすべて無視する（エラーの報告は行われる）。
+ignoreErrors	エラー発生時には処理を停止する（デフォルト）。
-htmlPat <pattern>	<meta> タグの解析を行うかどうかを判断する際にどのファイルが HTML ファイルなのかを判断するためのパターンを指定する。これは何度でも指定可能。何も指定されない場合には、*.htm と *.html が使用される。
-properties <name>	connectionPool が定義されている loaddocs.properties ファイルの位置を指定する。このファイルには、-columnMap 引数と同様の jdbc.column.<columnName>=<propname> という形式のエントリを記述してもよい。
-conPool <name>	BulkLoader による接続情報の取得先となるプロパティ ファイルに記述されている connectionPool 名 を指定する。

表 8-1 Bulk Loader のスイッチ設定

スイッチ設定	解説
-schema <name>	BulkLoader によって生成されるスキーマ ファイルのパスを指定する（デフォルトは document-schema.xml）。
+schema	指定された場合、スキーマ ファイルは作成されない。
-schemaName <name>	BulkLoader によって生成されるスキーマの名前を指定する。デフォルトは「LoadedData」。
-encoding <name>	使用するファイル エンコーディングを指定する。デフォルトは、システムのデフォルト エンコーディング（有効なエンコーディング名については JDK マニュアルを参照）。
-commitAfter <num>	この数のドキュメントがロードされたあとに JDBC トランザクションをコミットする。デフォルトでは、すべてをロードしたあとでのみコミットする。
-match <pattern>	BulkLoader の処理の対象となるファイル パターンを指定する。これは何度でも指定可能。何も指定されない場合には、すべてのファイルとディレクトリが対象となる。
-ignore <pattern>	BulkLoader の処理の対象とならないファイル パターンを指定する。これは何度でも指定可能。
-d <dir>	相対パスの基準となる docBase を指定する。指定されない場合には、「.」（カレント ディレクトリ）が使用される。
-mdext <ext>	メタデータ プロパティ ファイルのファイル名拡張子を指定する。必ず 「.」で始まる値を指定する（デフォルトは .md.properties）。
-filter <filter class>	ファイルを処理する際に用いる LoaderFilter のクラス名を指定する。これを複数回指定すれば、LoaderFilte のリストに追加することができる。
+filters	LoaderFilter の現在のリストをクリアする（デフォルト フィルタもクリアされる）。

表 8-1 Bulk Loader のスイッチ設定

スイッチ設定	解説
`--`	後続のものがすべてファイルまたはディレクトリとみなされる。
`-columnMap <file.properties>`	DOCUMENT テーブルの追加カラム（`-column` を参照）のリスト（各エントリは `jdbc.column.<columnName>=<propname,...>` の形式）が記載されているプロパティ ファイルを指定する。これは、標準カラムの動作をオーバーライドするために使用することはできない。
`-column <columnName>=<propName,...>`	DOCUMENT テーブルの追加カラムと、そのカラムにマップされるプロパティ名を指定する。これは、標準カラムの動作をオーバーライドするために使用することはできない。
`+columns`	コンフィグレーション済みの追加カラムをすべてクリアする。

3. アプリケーションの `META-INF/application-config.xml` に記述されている（または、WebLogic Server コンソールを通じて設定される）DocumentConnectionPool が正しいディレクトリを指すようにコンフィグレーションされていることを確認します。
4. docPool が XML スキーマ ファイルを再ロードするためには、再起動する必要があります。

この方法がうまくいくためには、BulkLoader でサポートされている以下の 3 通りの方法のいずれかで、CMS がドキュメントのメタデータをパブリッシュ（公開）できなければなりません。

- 対応する `md.properties` ファイルでパブリッシュする。このファイルは、ドキュメントのユーザ定義メタデータが記述された Java 形式の標準プロパティ ファイルでなければなりません。たとえば、`image.gif` の場合なら、このファイルは `image.gif.md.properties` になるでしょう。
- ドキュメント ファイルでパブリッシュする。HTML ファイルの場合には、メタデータは 、ドキュメント ファイル自体に含まれている標準の `<META name="…" content="…">` タグ内に記述することができます。

- LoaderFilter を作成してメタデータを収集できるような何か他の形態でパブリッシュする。

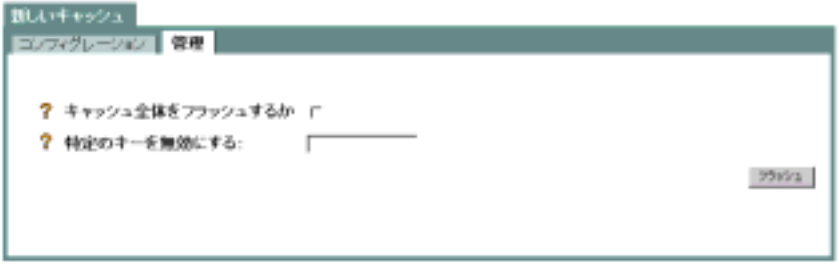
BulkLoader のパフォーマンス向上のヒント

以下の推奨事項に従えば、BulkLoader のパフォーマンスが向上します。

キャッシュをクリーンアップする

ページへの細かい更新を多数行った場合には、BulkLoader を実行する前に、キャッシュをクリーンアップしたほうがよいでしょう。Administration Console の [Cache Manager] ノードの [管理] タブで、図 8-1 に示すように [**キャッシュ全体をフラッシュするか**] を選択し、[**フラッシュ**] をクリックします。

図 8-1　キャッシュのフラッシュ



フラッシュは、以下のキャッシュに特に有効です。

- `DocumentMetaDataCache`
- `AdBucketServicesCache`
- `DocumentContentCache`
- `AdServicesCache`

Connection Pool を再起動する

E-Business Control Center を用いて新しいメタデータ プロパティを追加してある場合には、Connection Pool を再起動する必要があります。それには、以下を実行します。

1. Administration Console の左ペインで、以下のノードを開きます。

[<yourDomain> | デプロイメント | アプリケーション | <yourPortal> | Service Configuration | Document Connection Pool Service | default]

Document Connection Pool サービスの [default] タブ （図 8-2 に示す）が、Console の右ペインに表示されます。

図 8-2 Connection Pool の再起動

2. [**適用後再起動**] をクリックします。

コンテンツ マネージャのコンフィグレーション

サードパーティ製のコンテンツ管理システムを利用してポータルにコンテンツを供給する場合には、そのコンテンツ管理システムを、WebLogic Portal と連携するようにコンフィグレーションする必要があります。この節では、必要なコンフィグレーション手順について説明します。

コンテンツ マネージャは、タグと EJB を通じてコンテンツにアクセスできるようにする実行時サブシステムです。JSP を開発する際には、コンテンツ管理タグを使用すれば、検索式構文を用いてコンテンツ データベースに直接クエリを発行することでコンテンツ オブジェクトの一覧を取得できます。コンテンツ マ

ネージャ コンポーネントは、その他のコンポーネントと連携して、パーソナライズされたコンテンツを配信しますが、編集時カスタマイズに使用できる GUI ベースのツールは備えていません。

以下の節では、コテンツ マネージャのコンフィグレーションに必要な作業について説明します。以下のトピックを扱います。

- DocumentManager EJB デプロイメント記述子をコンフィグレーションする
- PropertySetManager EJB デプロイメント記述子をコンテンツ管理用にコンフィグレーションする
- DocumentManager MBean をコンフィグレーションする
- ドキュメント接続プールをセットアップする
- WebLogic Console で DocumentConnectionPool MBean を編集する
- Web アプリケーションをコンフィグレーションする

DocumentManager EJB デプロイメント記述子をコンフィグレーションする

`DocumentManager` EJB デプロイメント記述子は、コンテンツ管理コンポーネントのコンフィグレーションの EJB 部分を扱うもので、正しい環境設定を認識するようにコンフィグレーションする必要があります。`DocumentManager` EJB をコンフィグレーションするには、そのデプロイメント記述子に必ず、以下の環境設定が記載されていなければなりません。

- `DocumentManagerMBeanName` — この `DocumentManager` のコンフィグレーションに使用する `DocumentManager` MBean の名前を指定する
  - エンタープライズ アプリケーションの `application-config.xml` ファイルには、デプロイメント記述子に指定されている値と同一の `Name` 属性を持つ `<DocumentManager>` エントリが存在しなければなりません。それがデプロイメント記述子に指定されていない場合には、デフォルト値は「default」になります。
- `DocumentConnectionPoolName` — この `DocumentManager` で使用する `DocumentConnectionPool` MBean の名前を指定する

- アプリケーションの application-config.xml ファイルには、デプロイメント記述子に指定されている値と同一の Name 属性を持つ <DocumentConnectionPool> エントリが存在しなければなりません。
- MBean を使用するように DocumentManager の DocumentConnectionPoolName 属性がコンフィグレーションされている場合には、デプロイメント記述子内の値は無視されます。

■ jdbc/docPool — この DocumentManager でドキュメント接続プールへのアクセスに使用する javax.sql.DataSource への J2EE リソース参照を指定する

jdbc/docPool がデプロイメント記述子に指定されている場合には、

- DocumentConnectionPoolName は無視され、
- アプリケーションの application-config.xml ファイルに記述されているすべての DocumentConnectionPool MBean も無視されます。

■ PropertyCase — 入力されるプロパティ名が DocumentManager によってどう変更されるかを指定する

- 使用される DocumentManager MBean の PropertyCase 属性が設定される場合には、デプロイメント記述子内の値は無視されます。
- この属性が *lower* の場合には、すべてのプロパティ名は小文字に変換される。
- この属性が *upper* の場合には、すべてのプロパティ名は大文字に変換される。
- この属性が上記以外のものか未指定の場合には、プロパティ名は変更されない。

注意： *lower* と *upper* のどちらにするかは、使用するドキュメント接続プール実装によって決まります。ドキュメント参照実装の場合には、PropertyCase を指定してはいけない。

PropertySetManager EJB デプロイメント記述子をコンテンツ管理用にコンフィグレーションする

コンテンツ プロパティ セットがシステムにエクスポーズされるように、DocumentManager を PropertySetManager EJB デプロイメント記述子に統合する必要があります。PropertySetManager EJB デプロイメント記述子をコンテンツ管理用にコンフィグレーションするには、以下の環境設定を追加します。

- repository/CONTENT ― com.bea.p13n.property.PropertySetRepository 実装の完全修飾クラス名を指定する。コンテンツ管理コンポーネントと統合するには、com.bea.p13n.content.PropertySetRepositoryImpl を使用します。
- ejb/ContentManagers/[type] ― com.bea.p13n.content.PropertySetRepositoryImpl では、ejb/ContentManagers で始まるすべての環境エントリを探す。これらのエントリは、ContentManager（またはそのサブクラス）への J2EE EJB 参照であると想定されます。

  ContentManager または DocumentManager と PropertySetManager を統合するには、ここに EJB 参照を 1 つ追加します。たとえば、ejb/ContentManagers/Document は標準の DocumentManager にマップされます。

あるいは、DocumentManager MBean の JNDIName 属性を DocumentManager の JNDI ホーム名に設定することもできます。その値では ${APPNAME} 構文要素を使用することができ、それは現在の J2EE アプリケーション名に置き換えられます。com.bea.p13n.content.PropertySetRepositoryImpl では、それらの DocumentManager を自動的に見つけるので、J2EE EJB 参照は必要ありません。

DocumentManager MBean をコンフィグレーションする

DocumentManager 実装では、DocumentManager MBean を用いて、そのコンフィグレーションの保守を行います。デプロイされた DocumentManager は、どの DocumentManager MBean を使用すべきかを DocumentManagerMBeanName EJB

のデプロイメント記述子設定から知ることができます。それらの値が DocumentManagerMBeanName EJB デプロイメント記述子内の Name 属性に一致するように、アプリケーション内の DocumentManager MBean をコンフィグレーションする必要があります。

DocumentManager MBean をコンフィグレーションするには、アプリケーションの META-INF/application-config.xml ファイルを修正して、以下の XML（コード リスト 8-1 に示す）に追加または変更することができます。

コード リスト 8-1 アプリケーションの META-INF/application-config.xml ファイル内の <DocumentManager> 要素の修正

```
<DocumentManager
  Name="default"
  DocumentConnectionPoolName="default"
  PropertyCase="none"
  MetadataCaching="true"
  MetadataCacheName="documentMetadataCache"
  UserIdInCacheKey="false"
  ContentCaching="true"
  ContentCacheName="documentContentCache"
  MaxCachedContentSize="32768"
>
</DocumentManager>
```

application-config.xml ファイル内で直接これらの属性を変更しようとしてはいけません。その代わり、8-11 ページの「WebLogic Server Administration Console を用いて DocumentManager MBean を修正する」で説明しているように、WebLogic Server Administration Console を使用します。

WebLogic Server Administration Console を用いて DocumentManager MBean を修正する

WebLogic Server Administration Console を用いて DocumentManager MBeans を修正するには、以下の手順に従います。

1. WebLogic Server を起動し、Web ブラウザを開きます。
2. [アドレス] フィールドに以下の URL を入力して、WebLogic Server Administration Console を開きます。

```
http://<hostname>:<port>/console
```

たとえば、サーバ上でコンソールを起動する場合、デフォルト URL は以下のとおりです。

`http://localhost:7501/console`

3. 〔**Enter**〕を押します。

   WebLogic Server のサインイン画面が表示されます。

4. ユーザ名とパスワードを入力し、[**サインイン**] をクリックすることで、サインインします。デフォルトのユーザ名 / パスワードは、`system/weblogic` です。

5. 図 8-3 に示すような WebLogic Server Administration Console が表示されます。

図 8-3　WebLogic Server Administration Console

6. 左ペインで以下のノードを順に選択して、アプリケーションの DocumentManager MBean まで階層を下ります。

   `MyDomain` | デプロイメント | アプリケーション | `MyApplication` | `Service Configuration` | `Document Manager` | `MyMBean`

   ここで、

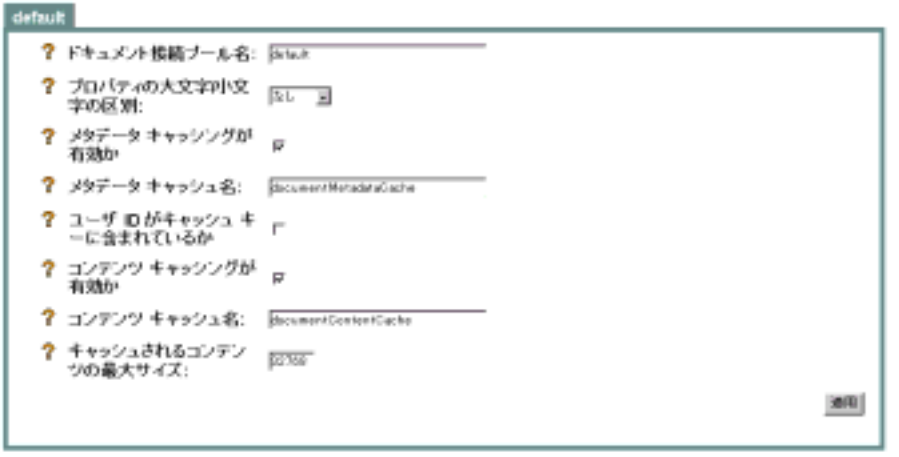
   - MyDomain はアプリケーションが存在するドメイン
   - MyApplication は当該 MBean を使用する Web アプリケーション
   - MyMBean はコンフィグレーション対象となる実際の MBean

   portalApp というアプリケーションの Document Manager のデフォルト MBean (default) まで階層を下る際に選択するノードの例を図 8-4 に示します。

図 8-4　MBean までのナビゲーション

MBean ノードを選択すると、その MBean の [Document Manager サービス] 画面（図 8-5 に示す）が右ペインに表示されます。MBean 名がタブに表示されていることに注意してください。

図 8-5 [Document Manager サービス] 画面



7. 属性を必要に応じて変更し、[**適用**] をクリックします。これらの属性を 表 8-2 に示します。

表 8-2 `DocumentManager` MBean の属性

属性	画面上の表示名 / コントロール タイプ	解説
`DocumentConnectionPoolName`	**[ドキュメント接続プール名]** テキスト ボックス	`DocumentManager` で使用する `DocumentConnectionPool` MBean の名前を指定する。

表 8-2 DocumentManager MBean の属性

属性	画面上の表示名 / コントロール タイプ	解説
PropertyCase	**[プロパティの大文字 / 小文字の区別]** ドロップダウン リスト	入力されるプロパティ名が DocumentManager によってどう変更されるかを指定する。 ♦ この属性が *lower* の場合には、すべてのプロパティ名は小文字に変換される。 ♦ この属性が *upper* の場合には、すべてのプロパティ名は大文字に変換される。 ♦ この属性が上記以外のものか未指定の場合には、プロパティ名は変更されない。 *lower* と *upper* のどちらにするかは、使用するドキュメント接続プール実装によって決まる。ドキュメント参照実装の場合には、PropertyCase を指定してはいけない。
MetadataCaching	**[メタデータ キャッシングが有効か]** チェックボックス	検索の結果得られる Document メタデータを DocumentManager でキャッシュするか否かを指定する。true の場合、DocumentManager では、MetadataCacheName で指定される com.bea.p13n.cache.Cache に検索結果がキャッシュされる。キャッシュしない場合には、false を指定する。デフォルトは true。
MetadataCacheName	**[メタデータ キャッシュ名]** テキスト ボックス	MetadataCaching が true に設定されている場合に使用する com.bea.p13n.cache.Cache の名前を指定する。デフォルトは documentMetadataCache。

表 8-2 DocumentManager MBean の属性

属性	画面上の表示名 / コントロール タイプ	解説
UserIdInCacheKey	**[ユーザ ID がキャッシュ キーに含まれているか]** チェックボックス	ドキュメントのメタデータまたはコンテンツをキャッシュする際にユーザの識別子をキーの一部として使用するか否かを指定する。デフォルトは true。WebLogic Portal の参照実装ドキュメント管理システムを使用する場合には、false に設定する。
ContentCaching	**[コンテンツ キャッシングが有効か]** チェックボックス	DocumentManager でドキュメント コンテンツ（すなわち、ドキュメントのバイト データ）をキャッシュするか否かを指定する。DocumentManager にドキュメント コンテンツのバイトデータをキャッシュさせる場合には true に設定し、そうでなければ false に設定する。デフォルトは true。
ContentCacheName	**[コンテンツ キャッシュ名]** テキスト ボックス	ContentCaching が true に設定されている場合に使用する com.bea.p13n.cache.Cache の名前を指定する。デフォルトは documentContentCache。
MaxCachedContent Size	**[キャッシュされるコンテンツの最大サイズ]** テキスト ボックス	ContentCaching が true の場合に DocumentManager でキャッシュされるドキュメント コンテンツの最大バイト数を指定する。デフォルトは 32768 (32K) バイト。
JNDIName	該当せず	この MBean に接続される DocumentManager EJB の JNDI ホーム名を指定する。その値では ${APPNAME} 構文要素を使用することができ、それは現在の J2EE アプリケーション名に置き換えられる。これは、com.bea.p13n.content.PropertySet RepositoryImpl において、ドキュメント プロパティ セット情報を PropertySetManager に結び付けるために使用される。

MBean を無効にする

WebLogic Server Administration Console から MBean を有効または無効にすることができます。それには、以下の手順に従います。

注意： この手順では、WebLogic Server に接続済みで Administartion Console が開かれていることを前提としています。そのための手順は、「WebLogic Server Administration Console を用いて DocumentManager MBean を修正する」の節で説明したとおりです。

1. 左ペインで [**Add or remove service configurations**] ノードを選択します。このノードは、図 8-6 に示すように、[Service Configuration] ノードの下にあります。

図 8-6　[Add or remove service configurations] ノードの選択

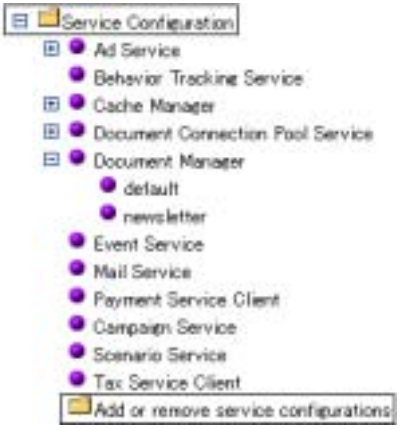
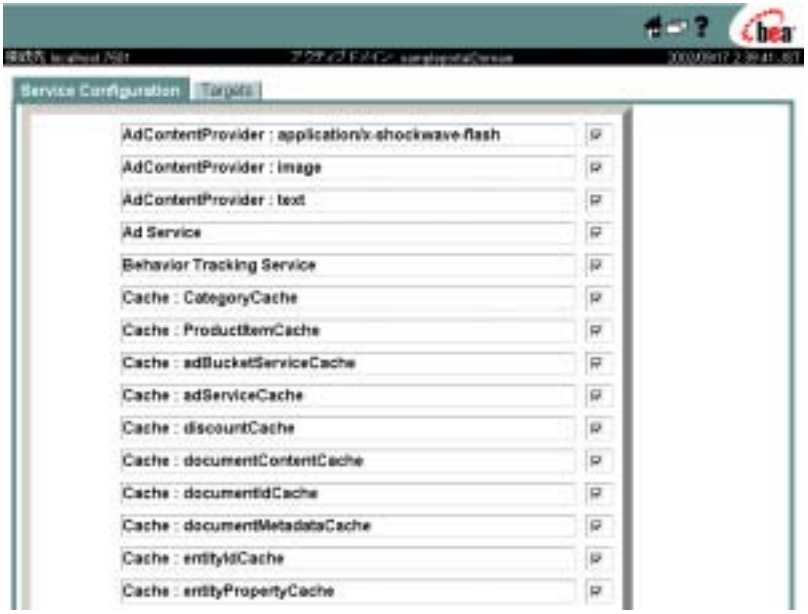


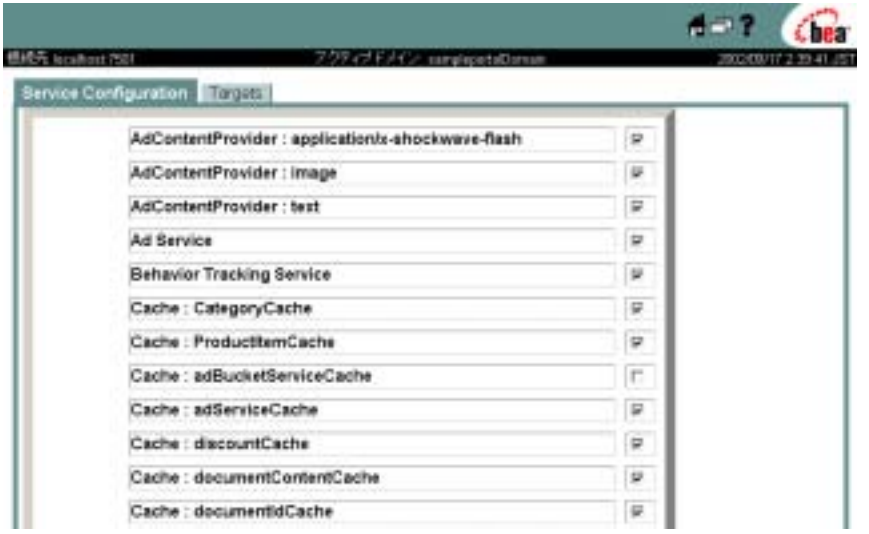
図 8-7 に示すような [Service Configuration] 画面が右ペインに表示されます。

図 8-7 [Service Configuration] 画面



2. 削除したい MBean をリストから見つけ、図 8-8 に示すように、該当するチェックボックスをチェックして選択を解除します。

図 8-8　MBean の選択解除を行った [Service Configuration] 画面



3. 無効にしたい MBean をそれぞれ選択解除します。作業が完了したら、[**Submit**] をクリックします。

   「これらの変更を反映させるには、アプリケーション [APP_NAME] を再デプロイする必要があります。」というメッセージが表示されます。

4. アプリケーションを再デプロイするか、サーバを再起動します。

無効にした MBean を再度有効にする

上記の手順で無効にした MBean を再度有効にするには、該当するチェックボックスをクリックして選択します。必要な MBean をすべて元どおり選択したら、やはり、[**Submit**] をクリックします。

ドキュメント接続プールをセットアップする

`DocumentManager` 実装では、専用の JDBC ドライバへの接続プールを使用して、コンテンツに関する検索を処理します。デプロイ済みの `DocumentManager` では、使用するドキュメント接続プールを、自分の `DocumentManager` MBean の

DocumentConnectionPoolName 属性か DocumentConnectionPoolName EJB デプロイメント記述子設定のいずれかを用いて見つけます。その値は、DocumentConnectionPool MBean に一致する必要があります。

DocumentConnectionPool MBean をコンフィグレーションするには、以下の XML（コード リスト 8-2 に示す）に追加または変更することで、アプリケーションの META-INF/application-config.xml ファイルを修正します。

コード リスト 8-2　アプリケーションの META-INF/application-config.xml ファイルの修正による DocumentConnectionPool MBean のコンフィグレーション

```
<DocumentConnectionPool

Name="default"DriverName="com.bea.p13n.content.document.
jdbc.Driver"URL="jdbc:beasys:docmgmt:com.bea.p13n.content
   document.ref.RefDocumentProvider"
Properties="jdbc.dataSource=weblogic.jdbc.pool.commercePool;
   schemaXML=D:/bea/user_projects/myNEWdomain/dmsBase/
   doc-schemas;docBase=D:/bea/user_projects/myNEWdomain/dmsBase"
InitialCapacity="20"
MaxCapacity="20"
CapacityIncrement="0"

/>
```

WebLogic Console で DocumentConnectionPool MBean を編集する

DocumentManager MBean の場合と同様に、DocumentConnectionPool MBean の修正には、WebLogic Server Administration Console を用いる必要があります。その方法については、8-11 ページの「WebLogic Server Administration Console を用いて DocumentManager MBean を修正する」の節で示した手順を参照してください。なお、DocumentConnectionPool MBean の場合には、[Document Connection Pool Service] ノードを選択し、[Document Connection Pool サービス] 画面（図 8-9 に示す）で変更を行う必要があることに注意してください。

図 8-9　[Document Connection Pool サービス] 画面

DocumentConnectionPool MBean の変更可能な属性を 表 8-3 に示します。

表 8-3　DocumentConnectionPool MBean の属性

属性	画面上のラベル / コントロール タイプ	解説
DriverName	**[ドライバ名]** テキスト ボックス	使用する JDBC ドライバ クラス名を指定する。これは、com.bea.p13n.content.document.jdbc.Driver に設定しなければならない。

表 8-3 DocumentConnectionPool MBean の属性

属性	画面上のラベル / コントロール タイプ	解説

URL	**[JDBC URL]**
	テキスト ボックス
		使用する JDBC URL を指定する。
		♦ WebLogic Portal の参照実装ドキュメント管理システムの場合には、以下のとおりに設定。
		`jdbc:beasys:docmgmt:com.bea.p13n.content.document.ref.RefDocumentProvider`
		♦ それとは別の Document Provider の場合には、以下を使用。
		`jdbc:beasys:docmgmt:<classname>`
		ここで、`<classname>` は `com.bea.p13n.content.document.spi.DocumentProvider` の実装の完全修飾クラス名。

Properties	**[JDBC プロパティ]** テキスト ボックス	プロパティの「名前 = 値」ペアをセミコロンで区切って並べたリストで、URL に指定された `DocumentProvider` にこれが渡される。参照実装で認識されるプロパティの一覧については、表 8-4 を参照のこと。
InitialCapacity	**[プールの初期容量]** テキスト ボックス	ドキュメント接続プールの起動時に作成される接続の初期数を指定する。
MaxCapcity	**[プールの最大容量]** テキスト ボックス	このプールで作成および維持管理される接続の最大数を指定する。

表 8-3 DocumentConnectionPool MBean の属性

属性	画面上のラベル / コントロール タイプ	解説
CapacityIncrement	**[容量増分]** テキスト ボックス	利用可能な接続を作成することが必要になるたびにプールで作成される接続の数を指定する。
LoginTimeout	該当せず	接続を待つ時間（この時間が過ぎたら、例外が送出される）を指定する。プールにタイムアウトさせない場合には、0 以下を使用する（これがデフォルト）。
ClassPath	該当せず	接続プールで Driver クラスと DocumentProvider クラスのロード時に使用すべき追加のディレクトリや JAR をセミコロンで区切って並べたリスト。指定されるパスはすべて、アプリケーション ディレクトリからの相対パスとみなされる。

DocumentConnectionPool MBean に設定できる有効な参照実装プロパティを表 8-4 に示します。

表 8-4 参照実装プロパティ

プロパティ	解説
jdbc.dataSource	データベース接続の取得に使用する javax.sql.DataSource の JNDI 名を指定する。このデータソースは、DOCUMENT テーブルと DOCUMENT_METADATA テーブルの入ったデータベースに接続しなければならない。
jdbc.url	接続先の JDBC URL を指定する。jdbc.dataSource が指定されている場合には、これは無視される。
jdbc.driver	ロードすべき JDBC ドライバ クラスを指定する。jdbc.dataSource が指定されている場合には、これは無視される。

表 8-4 参照実装プロパティ

プロパティ	解説
jdbc.isPooled	true の場合、jdbc.url が jdbc:weblogic:pool または jdbc:weblogic:jts で始まる場合、あるいは、jdbc.dataSource が指定されている場合には、接続はプールされておりキャッシュされないと仮定される。それ以外の場合には、接続はプールされておらず 1 つの接続のみを維持すると仮定される。
jdbc.supportsLikeEscapeClause	基礎となるデータベースで SQL の LIKE ESCAPE 句がサポートされているか否かを指定する。これが指定されない場合には、接続はクエリされる。
jdbc.docBase	ドキュメントがどのベース ディレクトリ下に格納されているかを指定する。データベースに格納されているパスはすべて、ここに指定したディレクトリからの相対パスとみなされる。
jdbc.schemaXML	プロパティ セット情報が記述された XML ファイル（ドキュメント スキーマ DTD に準拠）の格納先ディレクトリのパスを指定する。システムはこのディレクトリ内を再帰的に調べて、.xml で終わるファイルをすべてロードする。
jdbc.isolationLevel	データベース接続に設定するトランザクション アイソレーション レベルをコンフィグレーションする。指定可能な値は以下のいずれか。 READ_COMMITTED READ_UNCOMMITTED SERIALIZABLE REPEATABLE_READ NONE 未指定の場合には、デフォルトで SERIALIZABLE とみなされる。 詳細については、『Javadoc API マニュアル』内の java.sql.Connection を参照のこと。

表 8-4 参照実装プロパティ

プロパティ	解説
`jdbc.column.<colName>`	DOCUMENT テーブルの追加カラムを指定する。値は、そのカラムにマップされるプロパティ名をカンマで区切って並べたリスト。これは何度でも指定可能。BulkLoader の `-columnMap` 引数と `-column` 引数の両方あるいはいずれか一方と一緒に使用しなければならない。同じプロパティが複数のカラムにマップされる場合には、結果がどうなるかはわからない。

図 8-9 に示すように、WebLogic Server Administration Console を使用すれば、DocumentConnectionPool MBean を編集して、属性値やプロパティ値を必要に応じて変更することができます。

Web アプリケーションをコンフィグレーションする

コンテンツ管理サービスへのアクセスに必要な J2EE リソース（EJB、サーブレット、JSP タグ ライブラリなど）に Web アプリケーションからアクセスできるようにコンフィグレーションする必要があります。つまり、`ejb/ContentManager` と `ejb/DocumentManager` への EJB 参照をコンフィグレーションする必要があります。さらに、`com.bea.p13n.content.servlets.ShowDocServlet` を Web アプリケーションにマップしておくことが必要です。BEA では、コード リスト 8-3 に示すように、Web アプリケーションの `/ShowDoc/* URL` にマップすることをお勧めします。

コード リスト 8-3　ShowDocServlet のマッピング

```
<servlet>
  <servlet-name>ShowDocServlet</servlet-name>
  <servlet-class> com.bea.p13n.content.servlets.ShowDocServlet
  </servlet-class>

  <!-- showdoc が常にローカルの ejb-ref DocumentManager を使用するように
設定する -->
```

```
  <init-param>
    <param-name>contentHome</param-name>
    <param-value>java:comp/env/ejb/DocumentManager</param-value>
  </init-param>

</servlet>

...

<servlet-mapping>
  <servlet-name>ShowDocServlet</servlet-name>
  <url-pattern>/ShowDoc/*</url-pattern>
</servlet-mapping>
```

これによって、Web アプリケーションのコンテキスト ルート（たとえば /wlcs/ShowDoc）下の ShowDoc/ URI を ShowDocServlet に送ることができるようになります。初期化パラメータ contentHome（<init-param> タグ内）が上記のように設定されているので、その ShowDocServlet は常に ejb/DocumentManager EJB 参照を使用するようになります。この設定を削除すれば、ShowDocServlet は任意の contentHome リクエスト パラメータに従って動作できるようになります。

コンテンツ管理タグ ライブラリを利用するには、以下を実行する必要があります。

- cm_taglib.jar ファイルを Web アプリケーションの WEB-INF/lib ディレクトリにコピーする（この JAR ファイルは WL_PORTAL_HOME/lib/p13n/web からコピーできる）。
- Web アプリケーションの WEB-INF/web.xml ファイル内の <taglib> エントリで、cm.tld を必ず /WEB-INF/lib/cm_taglib.jar にマップする。

コンテンツ セレクタ タグおよび関連する JSP タグの使い方

コンテンツ セレクタは、コンテンツ管理システムからドキュメントを取得するためのメカニズムとして WebLogic Portal に用意されているものの 1 つです。コンテンツ セレクタ JSP タグなどの一連の JSP タグを使用すれば、コンテンツ セレクタで絞り込まれたコンテンツを取得し表示することができます。

この節では、コンテンツ セレクタ タグとそれらに関連する JSP タグを用いてコンテンツを管理する方法を説明します。以下のトピックスを扱います。

- <pz:contentSelector> タグの使い方
- コンテンツ セレクタを補助する関連タグ
- コンテンツ セレクタ タグと関連タグの使い方

WebLogic Portal のコンテンツ関連 JSP タグを WebLogic Portal のコンテンツ管理サービス プロバイダ インタフェース (SPI) へマップする方法については、http://edocs.beasys.co.jp/e-docs/wlp/docs70/jsp/p13njsp.htm にある『JavaServer Pages ガイド』の「パーソナライゼーション JSP タグ」を参照してください。

<pz:contentSelector> タグの使い方

<pz:contentSelector> セレクタ タグを用いて行えることは、以下のとおりです。

- コンテンツ セレクタの定義を識別する
- コンテンツ管理システムの JNDI ホームを識別する
- クエリ結果を格納する配列を定義する
- キャッシュを作成およびコンフィグレーションしてパフォーマンスを向上させる

コンテンツ セレクタの定義を識別する

E-Business Control Center で作成されるコンテンツ セレクタ定義では、コンテンツ セレクタがアクティブになる条件と、アクティブなコンテンツ セレクタで実行されるクエリを決定します。

この定義を参照するには、次のように rule 属性を使用します。

```
<pz:contentSelector rule= { definition-name | scriptlet } >
```

スクリプトレットを使用すれば、付加的な基準に基づいて rule 属性の値を決定することができます。たとえば、あるヘッダー JSP (heading.inc) でコンテンツ セレクタを使用し、その JSP を他の JSP にインクルードするとしましょう。このとき、heading.inc をインクルードするページごとに、異なるコンテンツ セレクタを作成することができます。

heading.inc でスクリプトレットを使用して、インクルードされた JSP ファイルが現在表示されているページに応じた属性値を設定します。コード リスト 8-4 に、その例を示します。

コード リスト 8-4　heading.inc でのスクリプトレットの使用

```
 String banner = (String)pageContext.getAttribute("bannerPh");
   banner = (banner == null) ? "cs_top_generic" : banner;
%>

<!-- ---------------------------------------------------------------
-->

<table width="100%" border="0" cellspacing="0" cellpadding="0"
   height="108">

  <tr><td rowspan="2" width="147" height="108">
  <pz:contentSelector rule="<%= banner %>" ... />

</td>
```

コンテンツ管理システムの JNDI ホームを識別する

コンテンツ セレクタ タグでは、contentHome 属性を用いてコンテンツ管理システムの JNDI ホームを指定する必要があります。参考版のコンテンツ管理システムを使用する場合や、サード パーティ製のコンテンツ管理システムと統合する場合には、スクリプトレットを用いてデフォルトのコンテンツ ホームを参照することができます。スクリプトレットでは ContentHelper クラスを使用するので、まず、以下のタグを用いてそのクラスを JSP にインポートする必要があります。

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
```

そのあと、コンテンツ セレクタ タグを使用する際には、コード リスト 8-5 に示すように contentHome を指定します。

コード リスト 8-5　コンテンツ セレクタ タグでの contentHome の指定

```
<pz:contentSelector
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
... />
```

独自のコンテンツ管理システムを構築する場合には、ContentHelper スクリプトレットを使用するのではなく、システムの JNDI ホームを指定する必要があります。さらに、その独自コンテンツ管理システムが JNDI ホームを提供する場合には、ContentHelper スクリプトレットを使用するのではなく、その JNDI ホームを指定することができます。

クエリ結果を格納する配列を定義する

表 8-5 に示した属性を用いて、コンテンツ セレクタでのクエリの結果を格納する配列をコンフィグレーションすることができます。

表 8-5 クエリ結果を格納する配列を定義する属性

属性	解説
id	配列の名前を指定する。この属性は必須。 たとえば、<pz:contentSelector id="docs" .../> と指定すると、docs という名前の配列にドキュメントが格納される。
max	コンテンツ セレクタ用の配列に格納されるドキュメントの数の上限を指定する。 たとえば、<pz:contentSelector max="10" .../> と指定すると、配列にドキュメントが 10 個格納された時点で、コンテンツ セレクタはドキュメントの取得を止める。 この属性は省略可能で、デフォルト値は -1（上限なし）。

表 8-5 クエリ結果を格納する配列を定義する属性

属性	解説
sortBy	配列内のドキュメントのソートに用いるドキュメント属性を任意の数だけ指定する。sortBy 属性の指定には、SQL の *order by 句*の構文を用いる。 この属性は省略可能。この属性を指定しない場合、コンテンツ セレクタは、コンテンツ管理システムから返された順にクエリ結果を返す。 たとえば、<pz:contentSelector sortBy="creationDate" .../> と指定すると、ドキュメントは作成日の早い順に配列に格納される。 また、<pz:contentSelector sortBy="creationDate ASC, title DESC" .../> というタグでは、ドキュメントは作成日の早い順に配列に格納され、同じ日に作成されたドキュメントが複数あれば、それらはタイトルの逆アルファベット順にソートされる。

キャッシュを作成およびコンフィグレーションしてパフォーマンスを向上させる

取得されたコンテンツによりアクセスしやすくなるように、また、パフォーマンスを向上させるために、コンテンツ セレクタ属性を用いて、配列の内容を格納するキャッシュを作成しコンフィグレーションすることができます。キャッシュを使用しなければ、コンテンツ セレクタ用の配列にアクセスできるのは、そのセレクタ タグを含む JSP ページ内だけであり、しかも、その配列を作成した顧客リクエストに限られます。さらに、コンテンツ セレクタ タグを含む JSP を顧客が要求するたびに、コンテンツ セレクタでクエリを実行しなければならず、そのため WebLogic Portal 全体のパフォーマンスが低下するおそれがあります。

配列の内容をキャッシュするには、表 8-6 に挙げた属性を使用します。

表 8-6 配列の内容をキャッシュするための属性

属性	解説
useCache	コンテンツ セレクタ用の配列をキャッシュに格納するかどうかを指定する。キャッシュを有効にするには、この属性を true に設定する。たとえば、<pz:contentSelector cache="true" ...> と指定する。 キャッシュを無効にするには、この属性を false に設定するか、この属性をタグに付けない。たとえば、以下の指定は機能的に等価。 <pz:contentSelector cache="false" .../> <pz:contentSelector ...>
cacheId	キャッシュの名前を指定する。この属性を指定しない場合、キャッシュ名には配列の名前（id 属性で指定する必要がある）が使用される。配列を作成した JSP またはユーザ セッション以外からキャッシュにアクセスする場合には、cacheId を指定する必要がある。
cacheTimeout	WebLogic Portal でキャッシュが保持される時間をミリ秒単位で指定する。コンテンツ セレクタは、指定された時間が経過するまでクエリを再実行しない。 たとえば、次のようにタグを記述するとしよう。 <pz:contentSelector cache="true" cacheTimeout=”300000” .../> 顧客がこのコンテンツ セレクタ タグを含むページを要求する。顧客はいったん別のページに移動し、2 分（120000 ミリ秒）後に再びこのページを要求する。このとき、コンテンツ セレクタでキャッシュのタイムアウト条件が評価されるが、コンテンツ セレクタがキャッシュを作成してから 120000 ミリ秒しか経過していないため、クエリは再実行されない。その代わり、キャッシュ内のドキュメントが表示される。

表 8-6 配列の内容をキャッシュするための属性

属性	解説
cacheScope	キャッシュへのアクセスを許可する範囲を指定する。この属性に指定できる値は以下のとおり。 ♦ application : 任意の顧客が要求する、Web アプリケーション内の任意の JSP ページからキャッシュにアクセスできる。 ♦ session（デフォルト）: 現在の顧客が要求する、Web アプリケーション内の任意の JSP からキャッシュにアクセスできる。 ♦ page : 任意の顧客が要求する現在の JSP からしかキャッシュにアクセスできない。 ♦ request : 現在のユーザ リクエストからしかキャッシュにアクセスできない。顧客がページを再び要求した場合、コンテンツ セレクタはクエリを再実行し、キャッシュを作成し直す。

コンテンツ セレクタを補助する関連タグ

コンテンツ セレクタを補助する JSP タグを表 8-7 に示します。

表 8-7　コンテンツ セレクタの機能を補助する JSP タグ

タグ	解説
<um:getProfile>	現在ページを閲覧している顧客のプロファイルを取得する。コンテンツ セレクタでは、その顧客プロファイルを用いて、顧客プロパティに絡む条件を評価する。 たとえば、「Gold Customer」という顧客セグメントに属するあらゆる顧客用のクエリを実行するコンテンツ セレクタを作成する場合、そのコンテンツ セレクタでは、顧客プロファイルにアクセスして、プロファイルがこの顧客セグメントに合致するかどうかを判定する必要がある。 コンテンツ セレクタでの条件評価に顧客プロファイルを現在使用していない場合でも、JSP に <um:getProfile> タグを組み込んでおくことを推奨する。このタグがパフォーマンスに及ぼす影響はごくわずかであり、またこのタグがあれば、開発者に JSP の修正を依頼しなくても、顧客プロファイルに関わる条件をコンテンツ セレクタに追加できる。 このタグは、コンテンツ セレクタ タグの近くではなく、JSP の冒頭付近に記載する必要がある。

表 8-7 コンテンツ セレクタの機能を補助する JSP タグ

タグ	解説
<es:forEachInArray>	コンテンツ セレクタでのクエリの結果を格納する配列内のデータを順に処理する。このタグを用いれば、配列内の各ドキュメントに対して以下の処理を実行できる。 ♦ System.out.println メソッドを使って、配列内の各要素を出力する。 ♦ <cm:getProperty> タグを使って、配列内の各ドキュメントの属性を任意の数だけ取得する。これらの属性を用いて、ブラウザでのドキュメントの表示に必要な HTML を作成できる。たとえば、<cm:getProperty> タグを用いて、ドキュメントの MIME-type 属性の値を決定するとしよう。配列内のドキュメントの MIME タイプが画像である場合には、適切な属性値を付けて HTML の <img> タグを出力する。 ♦ <pz:contentSelector> タグの属性（sortBy など）を用いて、配列内のドキュメントの属性を操作することもできる。 ♦ <cm:printProperty> タグを用いて、配列内の各ドキュメントの属性を任意の数だけ出力する。たとえば、このタグを使用すれば、コンテンツ セレクタで取得されたドキュメントのタイトル一覧を出力することができる。

コンテンツ セレクタ タグと関連タグの使い方

コンテンツ セレクタの定義、タグ属性、および関連する JSP タグを組み合わせれば、特定のコンテキストで顧客に合うドキュメントを用意するための強力なツール セットになります。ここでは、以下の作業例を通じて、コンテンツ セレクタとその関連タグの最も一般的な使い方を示します。

- テキスト タイプのドキュメントを取得し表示する
- 画像タイプのドキュメントを取得し表示する
- ドキュメントのリストを取得し表示する
- 別の JSP 上のコンテンツ セレクタ キャッシュにアクセスする

WebLogic Portal のコンテンツ関連 JSP タグを WebLogic Portal のコンテンツ管理サービス プロバイダ インタフェース (SPI) へマップする方法については、http://edocs.beasys.co.jp/e-docs/wlp/docs70/jsp/p13njsp.htm にある『JavaServer Pages ガイド』の「パーソナライゼーション JSP タグ」を参照してください。

テキスト タイプのドキュメントを取得し表示する

テキスト タイプのドキュメントを取得し表示するには、以下の手順に従います。

注意： この手順では、E-Business Control Center で作成されたコンテンツ セレクタ クエリに、テキスト ドキュメントだけを取得するフィルタが組み込まれているものとします。

1. JSP をテキスト エディタで開きます。
2. 必要なクラスおよびタグ ライブラリがまだ JSP に組み込まれていない場合には、コード リスト 8-6 に示す数行を JSP の冒頭付近に追加して、それらをインポートします。

コード リスト 8-6　クラスおよびタグ ライブラリをインポートするためのコード

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
```

3. 以下のタグがまだ JSP に含まれていない場合には、このタグを追加して顧客プロファイルを取得します。

```
<um:getProfile>
```

JSP で他の目的のために既にこのタグが使われている場合には、このタグにはおそらく何らかの属性が付いているでしょう。このタグは、<pz:contentSelector> タグ（次のステップで使用する）の近くではなく、必ず JSP の冒頭付近に記載してください。

4. コード リスト 8-7 に示す一連のタグを追加します。ここで、SpringSailing は、E-Business Control Center で作成されたコンテンツ セレクタの名前です。

コード リスト 8-7　コンテンツ セレクタ タグの例（その 1）

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="textDocs"/>
<es:forEachInArray array="<%=textDocs%>" id="aTextDoc"
type="com.bea.p13n.content.Content">
<p><cm:printDoc id="aTextDoc"/></p>
</es:forEachInArray>
```

注意： コンテンツ タイプが正しいかどうかを表示前に確認するには、<% "<P>" + aTextDoc + "</P>" %> というスクリプトレットを別のスクリプトレットで囲むことができます。コード リスト 8-8 に、その例を示します。

コード リスト 8-8　コンテンツ タイプの確認

```
<% if (aTextDoc.getMimeType().contains("text") != -1)
{
  %>
    <p><cm:printDoc id="aTextDoc"/></p>
<%
}
%>
```

5. JSP を保存します。Web アプリケーションを WAR ファイルとしてデプロイする場合には、Web アプリケーションを JAR アーカイブ化し直してから、デプロイします。

WebLogic Portal によって、変更内容がデプロイされます。Web アプリケーションのページ チェック率を指定してある場合には、WebLogic Portal はそのページ チェック間隔が経過するのを待ってから、変更内容をデプロイします。

画像タイプのドキュメントを取得し表示する

画像タイプのドキュメントを取得し表示するには、以下の手順に従います。

1. JSP をテキスト エディタで開きます。
2. 必要なクラスおよびタグ ライブラリがまだ JSP に組み込まれていない場合には、コード リスト 8-9 に示す数行を JSP の冒頭付近に追加して、それらをインポートします。

コード リスト 8-9　クラスおよびタグ ライブラリがまだ JSP に組み込まれていない場合にそれらをインポートするためのコード

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
```

3. 以下のタグがまだ JSP に含まれていない場合には、このタグを追加して顧客プロファイルを取得します。

   `<um:getProfile>`

   JSP で他の目的のために既にこのタグが使われている場合には、このタグにはおそらく何らかの属性が付いているでしょう。このタグは、`<pz:contentSelector>` タグ（次のステップで作成する）の近くではなく、必ず JSP の冒頭付近に記載してください。

4. コード リスト 8-10 に示す一連のタグを追加します。ここで、`SpringSailing` は、E-Business Control Center で作成されたコンテンツ セレクタの名前です。

コード リスト 8-10　コンテンツ セレクタ タグの例（その 2）

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="ImageDocs"/>

<es:forEachInArray array="<%=ImageDocs%>" id="anImageDoc"
type="com.bea.p13n.content.Content">

  <img src="ShowDoc/<cm:printProperty

  id="anImageDoc" name="identifier" encode="url"/>"

</es:forEachInArray>
```

注意： 上記のタグでは、E-Business Control Center で作成されたコンテンツ セレクタ クエリに、画像ドキュメントだけを取得するフィルタが組み込まれていることを前提としています。コンテンツ タイプが正しいかどうかを表示前に確認するには、<img> タグをスクリプトレットで囲むことができます。コード リスト 8-11 に、その例を示します。

コード リスト 8-11　スクリプトレットで囲まれた <img> タグ

```
<% if (anImageDoc .getMimeType().contains("image"))
{
%>
  <img src="ShowDoc/<cm:printProperty
  id="anImageDoc" name="identifier" encode="url"/>">
}
%>
```

5. JSP を保存します。Web アプリケーションを .war ファイルとしてデプロイする場合には、Web アプリケーションを JAR アーカイブ化し直してから、デプロイします。

   WebLogic Portal によって、変更内容がデプロイされます。Web アプリケーションのページ チェック率を指定してある場合には、WebLogic Portal はそのページ チェック間隔が経過するのを待ってから、変更内容をデプロイします。

ドキュメントのリストを取得し表示する

ドキュメントのリストを取得し表示するには、以下の手順に従います。

1. JSP をテキスト エディタで開きます。
2. 必要なクラスおよびタグ ライブラリがまだ JSP に組み込まれていない場合には、コード リスト 8-12 に示す数行を JSP の冒頭付近に追加して、それらをインポートします。

コード リスト 8-12　クラスおよびタグ ライブラリがまだ JSP に組み込まれていない場合にそれらをインポートするためのコード

```
<%@ page import="com.bea.p13n.content.ContentHelper"%> <%@
taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="pz.tld" prefix="pz" %>
<%@ taglib uri="um.tld" prefix="um" %>
```

3. 以下のタグがまだ JSP に含まれていない場合には、このタグを追加して顧客プロファイルを取得します。

   ```
   <um:getProfile>
   ```

   JSP で他の目的のために既にこのタグが使われている場合には、このタグにはおそらく何らかの属性が付いているでしょう。このタグは、<pz:contentSelector> タグ（次のステップで作成する）の近くではなく、必ず JSP の冒頭付近に記載してください。

4. コード リスト 8-13 に示す一連のタグを追加します。ここで、SpringSailing は、E-Business Control Center で作成されたコンテンツ セレクタの名前です。

コード リスト 8-13　コンテンツ セレクタ タグの例（その 3）

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs"/>
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
```

```
   <li>The document title is: <cm:printProperty id="aDoc"
   name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

5. JSP を保存します。Web アプリケーションを .war ファイルとしてデプロイする場合には、Web アプリケーションを JAR アーカイブ化し直してから、デプロイします。

   WebLogic Portal によって、変更内容がデプロイされます。Web アプリケーションのページ チェック率を指定してある場合には、WebLogic Portal はそのページ チェック間隔が経過するのを待ってから、変更内容をデプロイします。

別の JSP 上のコンテンツ セレクタ キャッシュにアクセスする

別の JSP 上のコンテンツ セレクタ キャッシュにアクセスするには、以下の手順に従います。

1. テキスト エディタで、コンテンツ セレクタ タグが含まれている JSP ページを開きます。たとえば、以下のタグの実行結果をキャッシュするとします。
   `<pz:contentSelector rule="SpringSailing" id="docs".../>`
2. コード リスト 8-14 に示すように、コンテンツ セレクタ タグに属性を追加します。

コード リスト 8-14　コンテンツ セレクタ タグの属性

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs"
useCache="true" cacheId="SpringSailingDocs"
cacheTimeout="120000"
cacheScope="application" />
```

これらの属性を指定することで作成されるキャッシュは、WebLogic Portal によって 2 分間（120000 ミリ秒間）保持され、Web アプリケーションの任意のページから任意のユーザによって SpringSailingDocs という名前でアクセスすることができます。cacheScope 属性の取り得る値の詳細については、8-31 ページの「キャッシュを作成およびコンフィグレーションしてパフォーマンスを向上させる」を参照してください。

3. JSP を保存し、デプロイします。
4. 作成したキャッシュにアクセスする JSP をテキスト エディタで開きます。
5. キャッシュへのアクセスには、ステップ 2 で作成したのと同一のコンテンツ セ レクタ タグを使用します。たとえば、現在開いている JSP に、コード リスト 8-15 に示すタグを追加 します。

コード リスト 8-15　同一タグの追加

```
<pz:contentSelector rule="SpringSailing"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
id="docs"
useCache="true" cacheId="SpringSailingDocs" cacheTimeout="120000"
cacheScope="application" />
```

6. JSP を保存し、デプロイします。

外部コンテンツ管理システムとの統合

大量のコンテンツがあり、コンテンツの発行とタグ付けに対するもっと強力な管理機能を必要とする顧客向けに、BEA では、サードパーティ ベンダと提携して WebLogic Portal の柔軟性を高めます。サードパーティ製のコンテンツ管理システムは堅牢なコンテンツ作成管理ソリューションを提供するのに対して、コンテンツ マネージャはコンテンツをパーソナライズしてエンド ユーザに配信します。

統合戦略

BEA では、サードパーティ製のコンテンツ管理システム (CMS) を WebLogic Portal に統合するための戦略として、以下の 3 つをお勧めします。

- CMS に、ドキュメントをファイル システム上にパブリッシュさせ、参照実装の BulkLoader を用いてそれらをデータベースにロードさせる。これは、8-1 ページの「Bulk Loader を用いたコンテンツの追加」の節で説明したのと同じプロセスです。ただし、サードパーティ製の CMS によってデータが適切な場所に定期的にロードされることを保証しなければならない点が異なります。詳細については、8-1 ページの「Bulk Loader を用いたコンテンツの追加」を参照してください。
- `DocumentProvider` インタフェースの実装を作成する。8-43 ページの「DocumentProvider インタフェースを実装することでコンテンツを追加する」を参照してください。
- CMS に、参照実装ドキュメント リポジトリ内にパブリッシュさせる。8-53 ページの「参照実装にパブリッシュする」を参照してください。

DocumentProvider インタフェースを実装することでコンテンツを追加する

DocumentProvider オブジェクトは、SPI 実装へのエントリ ポイントとなるものです。これは、基礎となるコンテンツ管理システムにアクセスするメソッドから成ります。DocumentProvider を開発する際には、トランザクション状態やスレッド セーフティ（スレッド保証）を気にかける必要はありません。DocumentProvider では書き込みアクションを実行する必要がないので、トランザクションの中で DocumentProvider にアクセスする必要はありません。

DocumentProvider インタフェースを実装するには、`com.beasys.p13n.content.document.spi` パッケージに含まれている Java インタフェースの実装を作成することが必要になります。これらのインタフェースは下記のとおりです。

- `DocumentProvider`
- `DocumentIterator`

- DocumentMetadataDef
- DocumentDef
- DocumentSchemaDef

以下の各節では、CMS を WebLogic Portal に統合するためのこれらのインタフェースの実装方法について説明します。

これらのインタフェースの詳細については、『Javadoc』を参照して com.beasys.p13n.content.document.spi を調べてください。

以下の手順では、DocumentProvider インタフェースの実装方法を大まかに示します。

ステップ 1: CMS が使用上の最低要件を満たしていることを確認する

コンテンツ管理システム (CMS) を WebLogic Portal にうまく統合するには、表 8-8 に挙げた機能を CMS がサポートしている必要があります。

表 8-8 CMS の使用上の最低要件

要件	解説
ユニークなドキュメント ID	ドキュメントをシステム内の他のすべてのドキュメントから一意に識別する単一のキーが存在しなければならず、そのキーは String として表現できなければならない。たとえば、コンテンツ管理システムの中には、ドキュメントにオブジェクト ID を割り当てるものもあれば、相対パスを用いるものもある（参照実装はこちらのほう）。
ドキュメント メタデータ	ドキュメントに関するメタデータをすべて取得する手段が存在しなければならない。メタデータは、WebLogic Portal の標準データ型（Boolean、Integer、Float、DateTime、String、多値）で構成されているか、あるいはそれらのタイプに変換できなければならない。このメタデータには、少なくともドキュメントのサイズ（バイト単位）と MIME タイプ（MIME 1.0 に準拠）が含まれていなければならない。
ドキュメント コンテンツの取得	ドキュメントの未処理のバイト データを取得する何らかの手段が存在しなければならない。

表 8-8 CMS の使用上の最低要件

要件	解説
検索	CMS には、ドキュメント メタデータに対するクエリに基づいてドキュメントを検索する何らかのメカニズムが用意されていなければならない（WebLogic Portal では全文検索は必須ではない）。
スキーマ	CMS には、ドキュメント メタデータ スキーマをエクスポーズするメカニズムが用意されていなければならない。スキーマには、メタデータ属性およびそれらのデータ型とドキュメント タイプとの関連付けが記述されている。ルール エディタはこのスキーマ情報を用いて、パーソナライゼーションを実現するコンテンツ セレクタ ルールの作成を可能にする（通常のドキュメント検索方法ではスキーマ情報は使用されないため、パーソナライズされていないドキュメントの取得はスキーマがなくても可能）。
Java からのアクセス	CMS では、Java コードからドキュメントにアクセスするのに利用できる何らかのメカニズムがサポートされていなければならない。これに該当すると思われるものには、Java クラス ライブラリ、JNI からアクセスできるネイティブ共有ライブラリ、ソケット ベースのアクセス（HTTP、DCOM、クライアント / サーバなど）、DBMS レベルのアクセス、ファイル ベースのアクセス、eLink 対応のアクセスなどがある（ただし、これらに限定されない）。なお、WebLogic Portal に用意されている参照実装内にパブリッシュする場合には、Java からのアクセスは必須ではない。

これらの要件のいずれかが何らかの形で満たされない場合には、CMS を WebLogic Portal に完全に統合することはできません。そのうえ、WebLogic Portal にはドキュメント作成 / 編集機能がないので、CMS には、ユーザがドキュメントを作成および編集するための何らかの手段が用意されていなければなりません。

ステップ 2: SPI 実装を作成する

次に、表 8-9 に示したインタフェースと、8-47 ページの「DefaultDocumentProvider」に挙げたその他のデフォルト クラスおよびヘルパー クラスを実装することで、SPI をコーディングする必要があります。これ

らのインタフェースでは、BEA オブジェクトを CMS 側で認識可能なオブジェクトに変換するので、Web アプリケーションと CMS の間の双方向通信が可能になります。

表 8-9　DocumentProvider のインタフェース

インタフェース	解説
`DocumentIterator`	DocumentIterator インタフェースは、`close()` メソッドを追加して java.util.Iterator インタフェースを拡張したもの。この `close()` メソッドは、DocumentIterator がもう使用されなくなったときに呼び出され、DocumentIterator に結び付けられているリソースをすべてクリアする。このインタフェースによって呼び出されるメソッドは null を返してはいけない。それらのメソッドは、例外が必要な場合には DocumentException を送出しなければならない。結果セットが空の場合には、空の DocumentIterator を返さなければならない。
`DocumentMetadataDef`	DocumentMetadataDef インタフェースは、ドキュメントのメタデータ属性を表す。このインタフェースには、明示的メタデータを取得するメソッドと暗黙的（CMS 定義）メタデータを取得するメソッドが両方とも含まれている。`getProperty(String name)` メソッドは暗黙的メタデータを取得するためのもので、明示的属性名（identifier、size、version、author、creationDate、lockedBy、modifiedDate、modifiedBy、description、comments、および mimeType）の入力に対してデータを返してはいけない。明示的プロパティを取得する際には、対応するメソッドがインフラストラクチャによって個別に呼び出される。

表 8-9 DocumentProvider のインタフェース

インタフェース	解説
`DocumentDef`	DocumentDef インタフェースは、ドキュメント コンテンツの未処理のバイト データを表す。このインタフェースでは 2 つのメソッドを実装する必要があるが、主に呼び出されるのは `openStream()` メソッドである。`openStream()` から返される InputStream では、`skip()` メソッドと `available()` メソッドを効率的な方法でサポートすることをお勧めする。`skip()` メソッドは、コンテンツのひとまとまりのバイト データを WebLogic Portal に返す際に用いられる。また、`available()` メソッドは、ストリームから入手できるバイト データの残量を決定するのに用いられる。
`DocumentSchemaDef`	DocumentSchemaDef は、基礎となる CMS で利用可能な単一のスキーマを表す。このインタフェースには、属性の名前、データ型、取り得る値、および概要を問い合わせるためのメソッドが 含まれている。

WebLogic Portal のコンテンツ関連 JSP タグを WebLogic Portal SPI へマップする方法については、http://edocs.beasys.co.jp/e-docs/wlp/docs70/jsp/p13njsp.htm にある『JavaServer Pages ガイド』の「パーソナライゼーション JSP タグ」を参照してください。

実装すべきその他の DocumentProvider 関連クラス

表 3-1 に挙げたインタフェースの他に、SPI の一部機能を実装するための基本クラスとして、`com.bea.p13n.content.document.ref` パッケージ内の抽象クラスの一部を用いることができます。それは以下のクラスです。

- `DefaultDocumentProvider`
- `DefaultDocumentIterator`
- `DefaultDocumentMetadata`
- `DefaultDocumentSchema`
- `DefaultDocument`
- `FileDocument`
- `URLDocument`

DocumentProvider 実装の開発に役立つクラスには、その他に以下のものがあります。

- `com.bea.p13n.content.document.ref.DocumentComparator`
- `com.bea.p13n.content.expression.SortCriteria`
- `com.bea.p13n.content.expression.ExpressionHelper`
- `com.bea.p13n.content.expression.ExpressionAdapter`
- `com.bea.p13n.content.MimeTypeHelper`
- `com.bea.p13n.util.DefaultEntityResolver`
- `com.bea.p13n.util.jdbc.JdbcHelper`
- `com.bea.p13n.util.WildCard`

注意： 各クラスの詳細については、WebLogic Portal の『Javadoc』を参照してください。

検索用およびスキーマ用のメソッドを実装する

一般に、サードパーティ製の CMS と統合するには、検索用のメソッドとスキーマ用のメソッドを両方とも実装する必要があります。

検索用メソッドは、ドキュメントに関するメタデータ、すなわち返されるデータの全体的イメージを特定するデータを返します。検索が機能するには、検索オブジェクトに渡される検索条件が、CMS に用意されている検索メソッドにどう対応するかを決定したうえで、そのマッピングを定義する必要があります。

スキーマ用メソッドは、「バイト データ」、すなわち Web アプリケーションで使用されるデータを表すものを返します。必要なデータがポーリング先の CMS に存在すると仮定すれば、これらのメソッドから返されるものは以下のとおりです。

- 指定されたスキーマ オブジェクト
- 全スキーマ名のリスト
- スキーマ名のマップ

ステップ 3: コードをアプリケーション内に配置する

SPI 実装の作成が完了したら、それを使用できるように WebLogic Portal をコンフィグレーションする必要があります。それには、以下のいずれかを行います。

- 既存の DocumentConnectionPool を修正する

または

- 新しい DocumentConnectionPool と DocumentManager をコンフィグレーションする

既存の DocumentConnectionPool を修正する

最初の方法は、アプリケーションの `META-INF/application-config.xml` に記述されている既存の DocumentConnectionPool を単に修正することです。DocumentManager 実装では、専用 JDBC ドライバへの接続プールを用いて検索を処理します。デプロイされた DocumentManager では、DocumentManager MBean の DocumentConnectionPoolName 属性か DocumentConnectionPoolName EJB のデプロイメント記述子設定のどちらかを参照して、使用するドキュメント接続プールを見つけます。

新しい DocumentConnectionPool と DocumentManager をコンフィグレーションする

WebLogic Portal をコンフィグレーションする 2 番目の方法は、新しい DocumentConnectionPool をアプリケーションの `META-INF/application-config.xml` ファイルにセットアップし、新しい DocumentManager EJB をアプリケーションにデプロイすることです。

まず、8-11 ページの「WebLogic Server Administration Console を用いて DocumentManager MBean を修正する」の節で説明した手順に従って、新しい接続プールを作成します。その DocumentConnectionPool には、必ずアプリケーション内でユニークな名前（たとえば、「`myConnectionPool`」）を付けます。

次に、以下の手順に従って、新しい EJB を作成 します。

1. `<application-directory>/document.jar` ファイルを一時ディレクトリに解凍します。
2. 作成済みの実装コードを適切なディレクトリに追加します。

3. META-INF/ejb-jar.xml で、DocumentManager のエントリを基に、新しい <session> エントリを作成します。必ず、<ejb-name> エントリをユニークな名前に変更します。たとえば、コード リスト 8-16 では、NewsletterDocumentManager としています。

コード リスト 8-16　<session> エントリの新規作成

```
<!--  The Newsletter DocumentManager  -->
<session>
<ejb-name>NewsletterDocumentManager</ejb-name>
   <home>com.bea.p13n.content.document.DocumentManagerHome</home>
   <remote>com.bea.p13n.content.document.DocumentManager</remote>
   <ejb-class>com.bea.p13n.content.document.internal.
      SPIFastDocumentManagerImpl</ejb-class>
   <session-type>Stateless</session-type>

   <transaction-type>Container</transaction-type>

   <!--
    これにより、このインスタンスが application-config.xml で
    どの DocumentManager MBean を探すかが決まる
   -->

   <env-entry>
      <env-entry-name>DocumentManagerMBeanName</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>newsletter</env-entry-value>
   </env-entry>
</session>
```

4. META-INF/ejb-jar.xml 内の新しい <session> エントリで、DocumentManagerMBeanName の <env-entry> に含まれている <env-entry-value> をユニークな名前（ここでは「NewsletterDocumentManager」）に変更します。この値は、application-config.xml ファイルで後ほど使用されます。

5. META-INF/ejb-jar.xml に、新しい <session> エントリ用の <assembly-descriptor> エントリと <container-transaction> エントリを追加します。既存のコードをコピーしてもかまいませんが、<ejb-name> を必ず上記の値（ここでは「NewsletterDocumentManager」）に変更してください。コード リスト 8-17 に例を示します。

コード リスト 8-17 <assembly-descriptor> エントリと <container-transaction> エントリの追加

```
<assembly-descriptor>
   <container-transaction>
      <method>
         <ejb-name>NewsletterDocumentManager</ejb-name>
         <method-name>*</method-name>
      </method>
      <trans-attribute>Required</trans-attribute>
   </container-transaction>
   .
   .
   .
</assembly-descriptor>
```

6. META-INF/weblogic-ejb-jar.xml を編集して、DocumentManager のエントリを基に、新しい <weblogic-enterprise-bean> エントリを作成します。<ejb-name> エントリを必ず、先ほど用いた名前（ここでは「NewsletterDocumentManager」）に変更してください。コード リスト 8-18 に例を示します。

コード リスト 8-18 <weblogic-enterprise-bean> エントリの新規作成

```
<weblogic-ejb-jar>
   <weblogic-enterprise-bean>
      <ejb-name>NewsletterDocumentManager</ejb-name>
      <entity-descriptor>
         <persistence>
            <persistence-type>
               <type-identifier>WebLogic_CMP_RDBMS
               </type-identifier>
               <type-version>7.0</type-version>
               <type-storage>META-INF/weblogic-cmp-rdbms-jar.xml
               </type-storage>
            </persistence-type>
            <persistence-use>
               <type-identifier>WebLogic_CMP_RDBMS
               </type-identifier>
               <type-version>6.0</type-version>
            </persistence-use>
         </persistence>
      </entity-descriptor>

   <jndi-name>${APPNAME}.BEA_portal_examples.
      NewsletterDocumentManage</jndi-name>
```

```
</weblogic-enterprise-bean>
```

7. META-INF/weblogic-ejb-jar.xml で、新しい <weblogic-enterprise-bean> エントリ内の <jndi-name> を然るべき値に変更します。この値が、新しい DocumentManager への EJB 参照で用いられる JNDI 名になります。たとえば、以下のように指定します。

   ```
   <jndi-name>${APPNAME}.BEA_portal_examples.
      NewsletterDocumentManage</jndi-name>
   ```

8. 一時ディレクトリを document.jar にアーカイブ化し直します。

ステップ 4: .jar ファイルにアプリケーションからアクセスできるようにする

次に、CMS にアクセスするアプリケーションから、作成した .jar ファイルにアクセスできるようにする必要があります。それには、以下のいずれかの方法を用います。

- .jar をアプリケーションにデプロイ済みの .jar ファイル内に配置する。
- .jar を、DocumentConnectionPool を初期化する .jar ファイル（たとえば、document.jar）に含まれている META-INF/Manifest.mf ファイルの Class-Path エントリで参照されている .jar ファイルに配置する。
- アプリケーションの META-INF/application-config.xml ファイル内で、DocumentConnectionPool MBean の classpath 属性に .jar 名を設定する。

ステップ 5: サーバを再起動する

これまで、コンフィグレーションとクラスパスに数々の変更を加えてきたので、ここでサーバを再起動したほうがよいでしょう。

ステップ 6: ポータルを適用する

統合を完了するには、最後に、CMS データの表示先となるポータルを作成する必要があります。ポータルやポートレットの作成方法については、2-23 ページの「ステップ 3: ポートレットの追加」を参照してください。

参照実装にパブリッシュする

この戦略では、WebLogic Portal の参照実装データベース テーブルおよび XML スキーマ ファイルに直接パブリッシュすることが必要になります。

この戦略を実行するには、以下の手順に従います。

1. ドキュメント エントリとドキュメント メタデータをデータベース テーブルに格納します。
2. ドキュメント メタデータ スキーマを WebLogic Portal の XML スキーマ ファイルに記述します。
3. ドキュメント ファイルをファイル システム上に配置します。

コンテンツ クエリの作成

この節では、コンテンツ管理システムへのクエリを作成するためのガイドラインを示します。以下のトピックを扱います。

- クエリを構造化する
- クエリ作成のための比較演算子の使い方
- Java を用いたクエリの作り方
- Document サーブレットの使い方

クエリを構造化する

WebLogic Portal のクエリは、SQL 文字列構文と構文的に似ていて、基本ブール型の比較表現 (括弧でネストされたクエリを含む) をサポートしています。概して、クエリには、メタデータ プロパティ名、比較演算子、およびリテラル値が含まれます。たとえば、以下のように指定します。

```
attribute_name comparison_operator literal_value
```

注意： クエリ構文の詳細については、『*Javadoc* API マニュアル』で `com.bea.p13n.content.expression.ExpressionHelper` を参照してください。

この構文を用いて作成するクエリに適用される制約は、以下のとおりです。

- 文字列リテラルは、必ず単一引用符で囲んでおくこと。
  - `‘WebLogic Server’`
  - `‘football’`
- 日付リテラルは、シンプルな `toDate` メソッドを用いて作成可能。このメソッドは、単一引用符で囲まれた `String` 引数を 1 つまたは 2 つ取ります。引数 2 つの場合、第 1 引数は `SimpleDateFormat` 形式の文字列で、第 2 引数は日付文字列です。引数が 1 つのみの場合は、‘MM/dd/yyyy HH:mm:ss z’ 形式の日付文字列です。
  - `toDate(‘EE dd MMM yyyy HH:mm:ss z’, ‘Thr 08 Nov 2001 16:56:00 MDT’)`

- `toDate('02/23/2005 13:57:43 MST')`

■ 空白その他の特殊文字を含む名前のプロパティを比較するには、`toProperty` メソッドを使用する。一般に、英数字を使用するという Java の変数名規約に従っていないプロパティ名に対して、`toProperty` を使用します。

- `toProperty ('My Property') = 'Content'`

■ プロパティ名にスコープを入れるには、`scope.propertyName` か、`toProperty` メソッド (引数 2 つ) のいずれかを使用する。

- `toProperty ('myScope', 'myProperty')`

注意 : 参照実装ドキュメント管理システムでは、プロパティのスコープは無視されます。

■ 文字列リテラル内に特殊文字を含んだエスケープ シーケンスを作成するには、「\」に続けてその文字を使用する。

- `toProperty ('My Property\'s Contents') = 'Content'`

■ また、文字列リテラル内に非 ASCII 文字を埋め込むには、Java 形式の Unicode　エスケープ シーケンスを使用する。

- `'*\u65e5\u672c\u8a9e*'` のような記述

注意 : クエリ構文に使用できる文字は、ASCII および拡張 ASCII 文字 (0-255) のみです。

クエリ内で使用可能な引用符またはエスケープコード付きの文字列リテラルに任意の文字列を変換するには、`ExpressionHelper.toStringLiteral` を使用します。

■ *now* キーワード (式のリテラル値側でのみ使用) は、現在の日時を参照する。

■ ブール型リテラルは、`true` または `false` のいずれか。

■ 数値リテラルは、引用符などのテキスト修飾を一切含まず数字のみで構成する。*1.24e4*、*1.24E-4* などの科学的記数法もサポートされる。

■ 感嘆符 (!) は、式を否定する目的で、開き括弧の前に使用できる。

- `!(keywords contains 'football') || (size >= 256)`

■ ブール and 演算子は、リテラル `&&` で表す。

- `author == 'james' && age < 55`

■ ブール演算子 or は、リテラル `||` で表す。

- `creationDate > now || expireDate < now`

以下に、式全体の例を示します。

例 1:

```
((color='red' && size <=1024) || (keywords contains 'red' &&
creationDate < now))
```

例 2:

```
creationDate > toDate ('MM/dd/yyyy HH:mm:ss', '2/22/2000 14:51:00')
&& expireDate <= now && mimetype like 'text/*'
```

クエリ作成のための比較演算子の使い方

高度な検索をサポートするため、WebLogic Portal では、比較演算子を組み込んだネストしたブール型クエリを作成することができます。表 8-10 はメタデータ型に利用可能な比較演算子をまとめたものです。

表 8-10 各メタデータ型に対して利用可能な比較演算子

演算子の種類	特徴
ブール (==, !=)	ブール属性では、`Boolean.TRUE` または `Boolean.FALSE` に対して一致するかどうかのチェックが可能。
数値 (==, !=, >, <, >=, <=)	数値 (Numeric) 属性では、`java.lang.Number` に対して、標準的な大小比較 (等しい、大きい、小さい) が可能。
テキスト (==, !=, >, <, >=, <=, like)	テキスト文字列では、標準的な同等性チェック (大文字 / 小文字の区別あり) に加え、辞書順での大小比較が可能。さらに、文字列では、ワイルドカード パターン マッチング (すなわち、*`like`* 演算子) を用いた比較も可能。SQL の「LIKE」演算子や DOS プロンプトのファイル名照合と同様。この場合、任意の文字列と照合するにはアスタリスク (*) を、任意の 1 文字と照合するには疑問符 (?) を用いる。間欠的な照合 (例えば [] を使うもの) はサポートされていない。「*」や「?」そのものと照合するには、バックスラッシュ (\) をエスケープ文字として使用する。
日時 (==, !=, >, <, >=, <=)	日時属性では、`java.sql.Timestamp` に対して、標準的な大小比較 (等しい、大きい、小さい) が可能。

表 8-10 各メタデータ型に対して利用可能な比較演算子（続き）

演算子の種類	特徴
多値の比較演算子 (contains, containsall)	多値属性では、*contains* 演算子がサポートされる。この演算子は、その属性のサブタイプのオブジェクトを 1 つ取り、それが属性値の中に含まれているかどうかをチェックする。さらに、多値属性では、*containsall* 演算子もサポートする。これは、属性のサブタイプのオブジェクトのコレクションを取り、それらのすべてが属性値の中に含まれているかどうかをチェックするものである。 多値属性に適用された単値の演算子は、属性の値のコレクション全体に適用されることになる。なんらかの値が演算子およびオペランドと一致すれば、true が返される。たとえば、多値のテキスト属性 *keywords* が、値として、*BEA*、*Computer* および *WebLogic* を持ち、オペランドが *BEA* の場合、< 演算子は true を返し (*BEA* は *Computer* よりも小さい)、> 演算子は false を返し (*BEA* よりも小さな値はない)、== 演算子は true を返す (*BEA* は *BEA* と等しい)。
ユーザ定義の比較演算子	現在、ユーザ定義の属性に適用できる演算子はない。

注意 : 検索パラメータと式オブジェクトでは、ビット フラグ (!) を用いた式の否定が利用できます。

参照実装コンテンツ管理システムは、単値のテキストおよび数値プロパティしか持てません。暗黙のプロパティはすべて単値テキストです。

Java を用いたクエリの作り方

Content Management コンポーネントで用意されているクエリ言語を使う代わりに Java 構文を使ってクエリを作るには、『*Javadoc* API マニュアル』で com.bea.p13n.content.expression.ExpressionHelper を参照してください。

ContentManager セッション Bean は、Content Management コンポーネントの機能への主要なインタフェースです。ContentManager インスタンスを使用すると、コンテンツは、埋め込まれた com.bea.p13n.expression.Expression (式ツリーを表す) と一緒に、com.bea.p13n.content.expression.Search オブジェクトに基づいて返されます。

ContentManager で有効な式ツリーとして使用するには、下記の注意事項に留意する必要があります。

- 各ブランチ ノードには、下記のタイプに限る。他のブランチ ノード タイプは無効になる。

com.bea.p13n.expression.operator.logical.LogicalAnd、
com.bea.p13n.expression.operator.logical.LogicalOr、
com.bea.p13n.expression.operator.logical.LogicalMulitAnd、または
com.bea.p13n.expression.operator.logical.LogicalMultiOr。

- 各リーフ ノードには、下記のタイプに限る。他のリーフ ノード タイプは無効になる。

com.bea.p13n.expression.operator.comparative.Equals、
com.bea.p13n.expression.operator.comparative.GreaterOrEquals、
com.bea.p13n.expression.operator.comparative.GreaterThan、
com.bea.p13n.expression.operator.comparative.LessOrEquals、
com.bea.p13n.expression.operator.comparative.LessThan、
com.bea.p13n.expression.operator.comparative.NotEquals、
com.bea.p13n.expression.operator.string.StringLike、
com.bea.p13n.expression.operator.collection.CollectionContains、または
com.bea.p13n.expression.operator.collection.CollectionsContainsAll

- すべての有効なブランチ ノードまたはリーフ ノードは、com.bea.p13n.expression.operator.logical.LogicalNot ノードに入れることができる。
- 各リーフ ノードでは、左側は常に com.bea.p13n.content.expression.PropertyRef ノードにし、必ず getPropertySet() および getPropertyName() 用の Strings を指定する。
- これらのリーフ ノードの右側には、java.util.Collection、Long、Double、String、または java.sql.Timestamp を指定できる。com.bea.p13n.expression.operator.comparative.Equals、com.bea.p13n.expression.operator.comparative.NotEquals、com.bea.p13n.expression.operator.comparative.GreaterOrEquals、com.bea.p13n.expression.operator.comparative.GreaterThan、com.bea.p13n.expression.operator.comparative.LessOrEquals、com.bea.p13n.expression.operator.comparative.LessThan、または com.bea.p13n.expression.operator.collection.CollectionContains
- com.bea.p13n.expression.operator.string.StringLike リーフ ノードの右側に使えるのは、String のみ。他のものはすべて無効。

- com.bea.p13n.expression.operator.collection.CollectionsContainsAll リーフ ノードの右側に使えるのは、java.util.Collection のみ。他のものはすべて無効。

Document サーブレットの使い方

Content Management コンポーネントには、Document オブジェクトのコンテンツを出力する能力を備えたサーブレットがあります。このサーブレットは、コンテンツ管理システム内に存在する画像コンテンツをストリーミングする時や、HTML リンクが選択された際にコンテンツ管理システム内に格納されているドキュメントのコンテンツをストリームする場合に役立ちます。このサーブレットでサポートされる Request/URL パラメータを、表 8-11 に示します。

表 8-11 Document サーブレットでサポートされるリクエスト パラメータ

リクエスト パラメータ	必須	解説
contentHome	場合による	*contentHome* 初期化パラメータが指定されない場合は、必須で、DocumentHome の JNDI 名として使用される。*contentHome* 初期化パラメータが指定されている場合は、これは無視される。
contentId	いいえ	取り出すための Document の文字列識別子。指定されない場合、サーブレットは PATH_INFO 内を探す。
blockSize	いいえ	読み出すデータ ブロックのサイズ。デフォルトは 8K。1 回の操作でブロック全体を読み出すには、0 以下を指定する。

サーブレットは、Document のみをサポートし、Content の他のサブクラスはサポートしません。*Content-Type* は Document の mimeType に設定され、*Content-Length* を Document のサイズに設定され、*Content-Disposition* を、ブラウザからのファイル保存時に正しいファイル名を表すべく、正しく設定されます。

例 1: JSP での使い方

この例は、夕方に表示するニュース項目を検索し、それらを箇条書きのリストにして表示します。

```
<cm:select sortBy="creationDate ASC, title ASC"

query=" type = ‘News’ && timeOfDay = ‘Evening’ && mimeType like
‘text/*’ "id="newsList"/>

<ul>

<es:forEachInArray array="<%=newsList%>" id="newsItem"
type="com.bea.p13n.content.Content">

     <li><a href="ShowDoc/<cm:printProperty id="newsItem"
     name="identifier" encode="url"/>"><cm:printProperty
     id="newsItem" name="title" encode="html"/></a>

  </es:forEachInArray>

</ul>
```

例 2: JSP での使い方

この例は、キーワードに「*bird*」が含まれている画像ファイルを検索し、その画像を箇条書きリストにして表示します。

```
<cm:select max="5" sortBy="name" id="list"

query=" KeyWords like ‘*birds*’ && mimeType like ‘image/*’ "

contentHome="java:comp/env/ejb/MyDocumentManager"/>

<ul>

<es:forEachInArray array="<%=list%>" id="img"
type=”com.bea.p13n.content.Content”>

   <li><img src="/ShowDoc/<cm:printProperty id="img"
   name="identifier"
   encode="url"/>?contentHome=<es:convertSpecialChars
   string="java:comp/env/ejb/MyDocumentManager"/>">

<es:forEachInArray>

</ul>
```

第 9 章 ポータル ナビゲーションのセットアップ

ポータルのナビゲーションは Webflow を用いて実現されます。この Webflow は、Web アプリケーションの構築を支援し、プレゼンテーション ロジックと基礎となるビジネス プロセスとを常に分離しておくのに役立つよう設計されたメカニズムです。訪問者が、ページ上で [**次へ**] をクリックするといったイベントを発生させると、Webflow は訪問者に次に表示するコンテンツを決定します。Webflow はまた、訪問者の操作の適切なタイミングを捉えて、Pipeline という定義済みの専用コンポーネントを呼び出すこともあります。Pipeline は、データの検証やバックエンド ビジネス プロセスの実行に用いられます。

Web サイトの訪問者に対するページの表示順序は Webflow の XML コンフィグレーション ファイルによって一元的に指定されるので、Webflow メカニズムを使用すると、Web サイトのフローを作成および変更するのに必要な労力を軽減することが可能です。

この章では、以下の内容について説明します。

- Webflow の作成
- Pipeline の作成と Webflow への追加
- アプリケーションへの Webflow の同期化
- 入力プロセッサの新規作成
- 拡張プレゼンテーション ノードと拡張プロセッサ ノードの作成による Webflow の拡張

Webflow の作成

この節では、必要なノードを Webflow に追加し、それらのノードを遷移で互いに接続することで、基本的な Webflow を作成する方法を示します。

Webflow を作成するには、以下の手順を実行する必要があります。

- ステップ 1: Webflow を作成する
- ステップ 2: Webflow キャンバスにノードを追加する
- ステップ 3: 開始ノードを指定する
- ステップ 4: ノード間の遷移を作成する

これらの手順を実行すれば、基本的な Webflow ができあがります。そのあとは、以降の節で示す作業を行うことで、Webflow 作成プロセスが完了します。

- Pipeline を作成して Webflow に追加する（9-20 ページの「Pipeline の作成と Webflow への追加」を参照）
- Webflow をアプリケーションに同期化する（9-35 ページの「アプリケーションへの Webflow の同期化」を参照）

注意： この章で説明している手順では、Webflow および Pipeline の特定の作成順序を念頭に置いていますが、これは説明の都合上そうしているだけであり、必ずそうしなければならないわけではありません。各自の開発ニーズに最も合う順序であれば何でもかまいません。たとえば、Webflow を実際に作成する前にまずその Webflow の Pipeline を作成したほうがよい場合もあるでしょう。ただし、Webflow が存在しないうちは、Webflow をアプリケーションに同期化することはできないので、くれぐれも注意してください。

Webflow コンポーネントの概要

Webflow の作成と実装を試みる前に、後述の手順で使用する Webflow コンポーネントのいくつかを理解しておくことが必要です。

ノードと遷移

ノードは、Webflow 内のある状態の機能をグラフィカルに表現したものです。ノード タイプに応じて、発生する可能性のある定義済み**イベント**（訪問者による Web ページ上のリンクのクリックなど）が多数用意されています。個々のイベントが発生すると、Webflow はどの後続ノード を呼び出してフローを続行するかを決定します。このプロセスを**遷移**と呼び、その概要を図 9-1 に示します。

図 9-1　一般的な Webflow 遷移

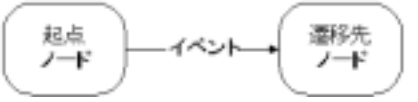


図 9-1 に示されているとおり、ノードは、遷移内の位置に応じて、起点ノードまたは遷移先ノードと呼ばれます。

ノードのタイプ

ノードの主要なタイプには、プレゼンテーション ノードとプロセッサ ノードの 2 つがあります。プレゼンテーション ノードとプロセッサ ノードはどちらも、Webflow 内で起点ノードまたは遷移先ノードとして使用することができます。

プレゼンテーション ノード

*プレゼンテーション ノード*は、Web アプリケーションを操作している人に Webflow が何かを表示する状態を表します。プレゼンテーションの形式には、以下のものがあります。

- HTML
- JavaServer Page (JSP)
- Java サーブレット

また、拡張（カスタム）プレゼンテーション ノードを作成して Webflow 内で使用することも可能です。拡張プレゼンテーション ノードの詳細については、9-39 ページの「拡張プレゼンテーション ノードの作成方法」を参照してください。

プロセッサ ノード

*プロセッサ ノード*は、Webflow が専用コンポーネントを呼び出して、フォーム検証や、サイトのプレゼンテーションを駆動するバックエンド ビジネス ロジックなどの処理を行う状態を表します。利用可能なプロセッサ ノードは、表 9-1 のとおりです。

表 9-1 Webflow プロセッサ ノードのタイプ

プロセッサ ノードのタイプ	解説
入力プロセッサ	入力プロセッサは定義済みの専用 Java クラスであり、Webflow メカニズムから呼び出されて、より複雑なタスクを実行する。入力プロセッサは通常、HTML フォーム データの検証や、Web ページ内での条件分岐の実現に用いられる。たとえば、入力された日付の形式が正しいかどうかを確かめるコードを、フォーム フィールドを表示するのと同じ JSP 内に埋め込むのでなく、入力プロセッサに入れておくことができる。入力プロセッサは Web アプリケーションに固有のロジックを含んでおり、そのため Web アプリケーションのコンテナによってロードされる。
Pipeline	Pipeline もプロセッサ ノードの一種で、Webflow から呼び出されることがある。Pipeline はビジネス プロセスに関連する特定タスクの実行を開始する。トランザクション対応の Pipeline とトランザクション非対応の Pipeline がある。たとえば、訪問者がサイト上の別のページに移動しようとしているが、その前に訪問者の情報をデータベースに保存しておきたいという場合には、Pipeline を使用することができる。Pipeline には、より大規模なエンタープライズ アプリケーション内の複数の Web アプリケーションに適用される可能性のあるビジネス ロジックが含まれており、そのため Pipeline は Enterprise JavaBeans（EJB）コンテナによってロードされる。 Pipeline はすべて、個々の Pipeline コンポーネントの集まりであり、それらのコンポーネントは Java オブジェクトやステートレス セッション Enterprise JavaBeans（EJB）として実装することができる。Pipeline コンポーネントは Pipeline の一部で、基礎となるビジネス ロジックに関連付けられているタスクを実際に遂行するものである。これらのタスクが複雑な場合には、Pipeline コンポーネントは外部サービス（他のビジネス オブジェクト）を呼び出すこともある。

表 9-1 Webflow プロセッサ ノードのタイプ

プロセッサ ノードのタイプ	解説
拡張プロセッサ ノード	Webflow で使用するための拡張（カスタム）プロセッサ ノード。拡張プロセッサ ノードの詳細については、「拡張プレゼンテーション ノードと拡張プロセッサ ノードの作成による Webflow の拡張」を参照のこと。

ワイルドカード ノード

Webflow が特定のプレゼンテーション ノードまたはプロセッサ ノードを見つけて遷移を完了できない場合には、*ワイルドカード* プレゼンテーション ノードまたは *ワイルドカード* プロセッサ ノードを検索して遷移先ノードとして使用します。そのため、ワイルドカード プレゼンテーション ノードとワイルドカード プロセッサ ノードは、Web アプリケーションのデフォルトの動作を実装しています。言い換えれば、ワイルドカード ノードを用いることで、共通の機能を抽出して Webflow 内の 1 箇所にまとめておくことができます。ワイルドカード ノードを用いるのは、遷移先ノードをまだ Webflow に明示的に定義していない場合に限ります。ネームスペースごとに、ワイルドカード プレゼンテーション ノードとワイルドカード プロセッサ ノードを 1 つずつ用意することができます。

ワイルドカード ノードの一例として、[**ヘルプ**] というリンク（どのページにも存在する）をクリックすると、ヘルプ情報が記載されている JSP に常に移動するようにしたい場合を考えてみましょう。それには、ワイルドカード プレゼンテーション起点を使用することができます。さらに、プロセッサ ノードから返された例外が、そのエラーに関する詳細な情報が記載された JSP に常に遷移するようにしたい場合もあります。これらの状況はどちらも、ワイルドカード プロセッサ ノードで処理することができます。

注意 : Webflow がワイルドカード ノードを検索する必要がある場合には、必要な処理が増えるため、パフォーマンスに少し影響が出ることがあります。

遷移のタイプ

遷移には、イベント遷移と例外遷移の 2 種類があります。

- イベント遷移は、2 つのノードのどちらか一方でイベントが発生し、そのイベントが正常に処理される場合に行われる、それら 2 つのノード間の処理ロジックを表す。
- 例外遷移は、プロセッサ ノードでの処理が失敗し、何らかの例外を送出する必要がある場合に行われる処理ロジックを表す。これらの遷移は、トランザクションの実行を制御するのが普通です。

遷移の原因となるイベントの詳細については、9-6 ページの「イベントのタイプ」を参照してください。

イベントのタイプ

Webflow 内の各ノードはイベントに応答し、その結果、遷移（すなわち、起点ノードから遷移先ノードへの移動）が発生します。ただし、ノードが応答するイベントのタイプは、ノードがプレゼンテーション ノードかプロセッサ ノードかによって異なります。

プレゼンテーション ノードは以下のイベントに応答します。

- リンク
- ボタン

すなわち、Web サイトの訪問者がリンクやボタンをクリックすると、Webflow はそのイベントに応答します。応答として考えられるのは、別のプレゼンテーション ノード（JSP など）への遷移や、プロセッサ ノード（訪問者から入力されたフォーム データを検証する入力プロセッサなど）への遷移でしょう。

一方、プロセッサ ノードは以下のイベントに応答します。

- 例外
- リターン オブジェクト

例外が発生するのは、入力プロセッサや Pipeline が正常に実行されず、エラー状態を示す場合です。そうでない場合には、これらのプロセッサ ノードはオブジェクトを返し、Webflow ではそれを用いて処理を続行できます。

注意： ポータル アプリケーションで用いられる Webflow では、上記より多くのイベントに応答する場合があります。

Webflow エディタのツールとボタン

Webflow の作成には、図 9-4 に示すような Webflow エディタを使用します。ほとんどの作業では、ツール ボタンかコマンド ボタンのどちらかを選択する必要があります。利用可能なツール ボタンについては表 9-2 で、コマンド ボタンについては表 9-3 で、それぞれ説明します。

表 9-2 Webflow エディタのツール

ツール	機能	解説
	[選択ツール]	ノード、イベント遷移、および例外遷移の選択や移動に用いる。遷移へのエルボーの追加にも用いる。Webflow エディタのデフォルト ツール。 **注意 :** このツールをいったん選択すると、別のツールを選択するまで、選択された状態のままになる。
	[イベント ツール]	2 つのノード間のイベント遷移や、自己参照イベント遷移を追加するのに用いる。 **注意 :** このツールをいったん選択すると、別のツールを選択するまで、選択された状態のままになる。
	[例外ツール]	2 つのノード間の例外遷移や、自己参照例外遷移を追加するのに用いる。 **注意 :** このツールをいったん選択すると、別のツールを選択するまで、選択された状態のままになる。
	[プレゼンテーション ノード]	エディタ キャンバスに新しいプレゼンテーション ノードを追加するのに用いる。
	[ワイルドカード プレゼンテーション ノード]	エディタ キャンバスに新しいワイルドカード プレゼンテーション ノードを追加するのに用いる。

表 9-2 Webflow エディタのツール（続き）

ツール	機能	解説
	[入力プロセッサ ノード]	エディタ キャンバスに新しい入力プロセッサ ノードを追加するのに用いる。
	[Pipeline ノード]	エディタ キャンバスに新しい Pipeline ノードを追加するのに用いる。
	[ワイルドカード プロセッサ ノード]	エディタ キャンバスに新しいワイルドカード プロセッサ ノードを追加するのに用いる。
	[拡張（カスタム）プロセッサ ノード]	エディタ キャンバスに新しい拡張（カスタム）プロセッサ ノードを追加するのに用いる。 **注意 :** このツールは、Webflow エディタが `webflow-extensions.wfx` ファイルに新しいノードを検出すると使用可能になる。
	[プロキシ ノード]	エディタ キャンバスにプロキシ ノードを追加するのに用いる。別のネームスペースに定義されているノードを参照するときには、必ずプロキシ ノードを作成しなければならない。

表 9-3 Webflow エディタのコマンド ボタン

ツール	機能	解説	キーボード ショートカット
	[印刷] ボタン	Webflow ネームスペースまたは Pipeline の全体をプリンタに出力するのに用いる。	〔Ctrl〕+〔P〕
	[削除] ボタン	選択されている Webflow コンポーネントを削除する。このボタンは、Webflow コンポーネントが選択されると使用可能になる。	〔Delete〕

表 9-3 Webflow エディタのコマンド ボタン（続き）

ツール	機能	解説	キーボード ショートカット
	[全体表示] ボタン	Webflow ネームスペースまたは Pipeline の全体を表示するのに用いる。	〔Ctrl〕+〔Z〕
	[選択したノードを検証] ボタン	選択されているノードに対して Webflow 検証機能を実行するのに用いる。このボタンは、Webflow コンポーネントが選択されると使用可能になる。	〔Ctrl〕+〔V〕
	[すべてを検証] ボタン	Webflow ネームスペース全体に対して Webflow エディタの検証機能を実行するか、Pipeline 全体に対して Pipeline エディタの検証機能を実行するのに用いる。	〔Alt〕+〔V〕
	[コンフィグレーション エラー ページ名のセットアップ] ボタン	コンフィグレーション エラー ページの名前とパスを指定するのに用いる。 **注意：** Webflow エディタでのみ利用可能。	なし
	[Pipeline コンポーネント エディタを起動] ボタン	Pipeline コンポーネント エディタを開く。このエディタを用いて、Pipeline コンポーネントを管理できる。 **注意：** Pipeline エディタでのみ利用可能。	なし

ステップ 1: Webflow を作成する

Webflow を作成するには、以下の手順に従います。

注意： この手順では、以下の事がらを前提としています。

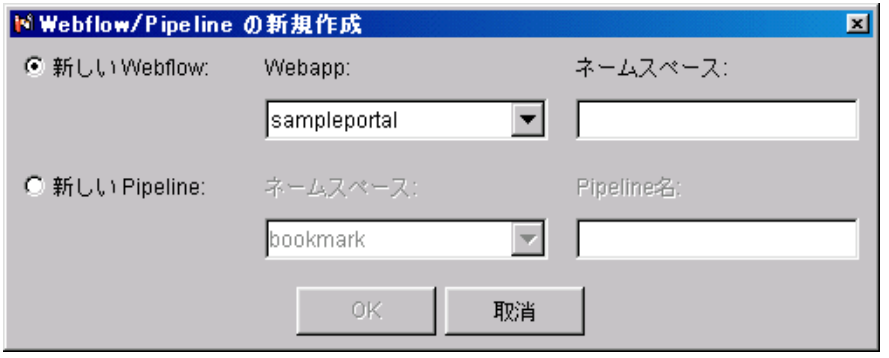
- 既存のプロジェクトを開いてあるか、もしくは新しいプロジェクトを作成してある。
- E-Business Control Center が動作している。
- [**サイト インフラストラクチャ**] タブが選択されている。

1. 図 9-2 に示すように、[新規作成] アイコンをクリックしてドロップダウン メニューを開き、[**Webflow/Pipeline**] を選択します。

図 9-2　E-Business Control Center で [新規作成] ドロップダウン メニューを開く

[Webflow/Pipeline の新規作成] ダイアログ ボックスが表示されます。

図 9-3　[Webflow/Pipeline の新規作成] ダイアログ ボックス



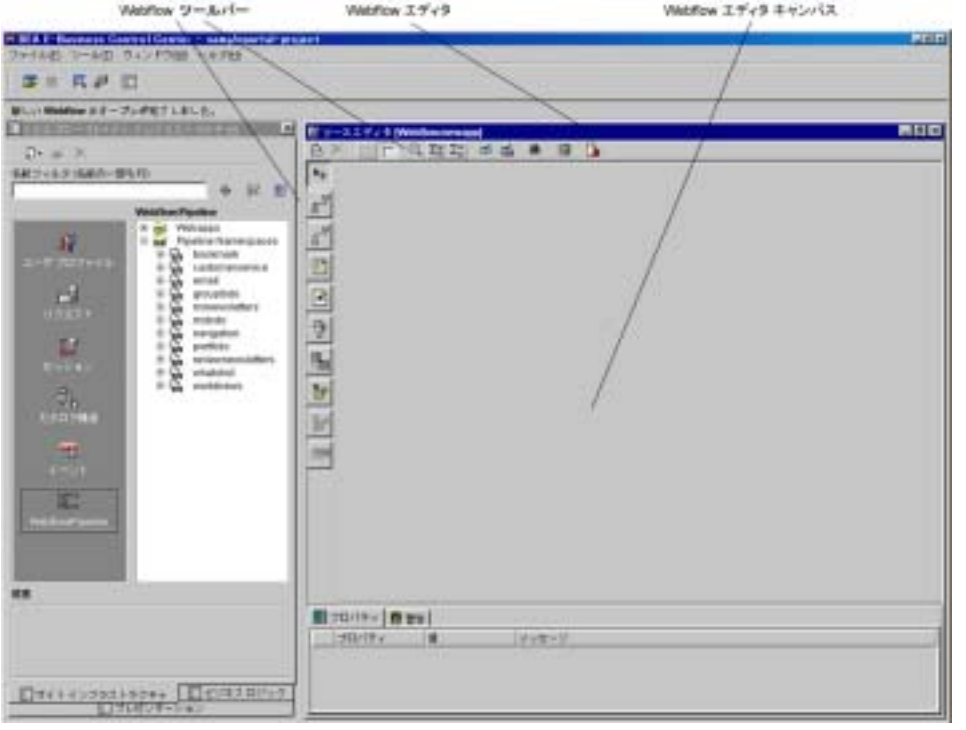
2. [**新しい Webflow**] が選択されていることを確認してから [**Webapp:**] ドロップダウン メニューを開き、Webflow の作成先となる Web アプリケーションを選択します（この値はすでに選択されているかもしれません）。この操作で Webflow のスコープ（有効範囲）がその Web アプリケーションに限定されるので、Web アプリケーションの選択は重要です。

3. [**ネームスペース**] 編集ボックスにネームスペース名を入力します。ネームスペースは、単一の Web アプリケーションで複数の Webflow を使用しても名前の衝突が発生しないように、Webflow のスコープを限定するのに用いられます。各 Web アプリケーションには複数のネームスペースが存在できることに注意してください。

4. [**OK**] をクリックします。

   E-Business Control Center の右ペインに Webflow エディタが表示されます。図 9-4 に例を示します。

図 9-4　Webflow エディタ



ステップ 2: Webflow キャンバスにノードを追加する

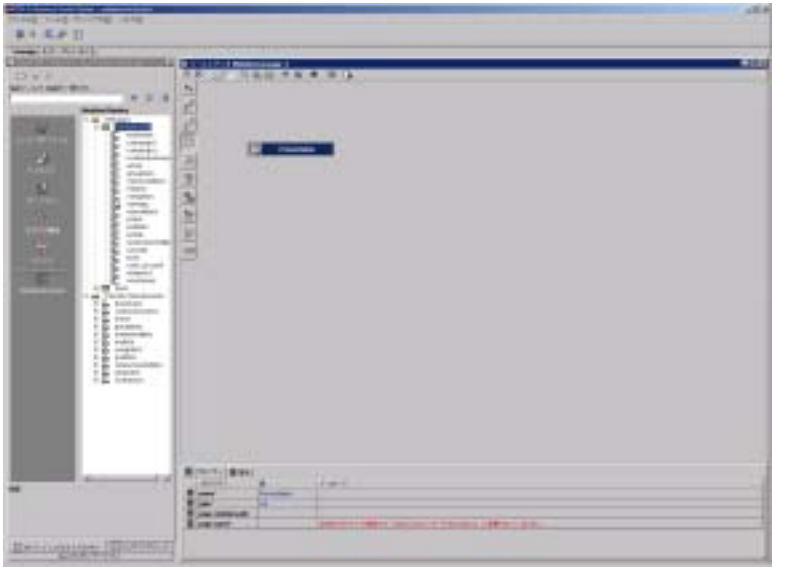
Webflow は 2 種類のノードから成ります。すなわち、プレゼンテーション ノードとプロセッサ ノードです。プレゼンテーション ノードとプロセッサ ノードはそれぞれ、Webflow 内の起点ノードまたは遷移先ノードに用いることができます。Webflow ノードの詳細については、9-2 ページの「Webflow コンポーネントの概要」を参照してください。

以下の手順に従って、Webflow エディタ キャンバスに最初のノードを追加します。

1. 適切なツールを選択し、キャンバス上の任意の場所にポインタ（十字線）を置いてクリックします。Webflow エディタ パレットに用意されている各ツールについては、表 9-2 で説明しています。

   キャンバス上のクリックした場所にノードが表示されます。たとえば、最初のノードをプレゼンテーション ノード（すなわち、JSP、HTML ファイルなどのプレゼンテーション ファイル）にする場合、[プレゼンテーション ノード] ツールを選択してからキャンバス上をクリックして、そこにノードを置くことになるでしょう。

図 9-5 プレゼンテーション ノードの配置



注意 : [プレゼンテーション ツール] は選択されたままになっています。これにより、ノードごとにツールを選択し直さなくても、プレゼンテーション ノードを必要なだけ追加できるようになります。また、ノードをキャンバスのどこに置くかを気にする必要はありません。ノードを選択して好ましい場所までドラッグすることで、ノードをどこにでも移動することができるからです。

ノードをキャンバスに置くと、そのノードのプロパティ エディタ（図 9-6 に示す）がキャンバスの下のペインに開きます。また、ノードそのものをクリックすることで、そのノードのプロパティ エディタを開くこともできます。

図 9-6 プレゼンテーション ノードのプロパティ エディタ

2. プロパティ エディタで、必要に応じて以下の情報を入力します。

表 9-4 プロパティ エディタ

プロパティ	解説
name	ノードの名前。この値がキャンバス上のノードに表示される。この値は、下記の **page-name** フィールドに入力すると自動入力される。
type	ノードの参照先ファイルのタイプ。プレゼンテーション ノードの場合には、有効なタイプは以下のとおり。 ♦ **portal** ♦ **jsp** ♦ **html/htm** ♦ **servlet**
page-relative-path	ノードの参照先ファイルの相対パス
page-name	ノードの参照先ファイル名。ここに入力される値は、**name** プロパティへの入力に使用される。ただし、ファイル拡張子は、選択されたタイプに基づいて自動的に設定されるので除外。たとえば、ここに「`pres01`」と入力し、タイプ (**type**) として「**jsp**」を選択する場合、このフィールドから移動すると、「`.jsp`」が page-name に付加され、**name** 値は「`pres01`」に変わる。

入力が完了したら、プロパティ エディタは、図 9-7 に示すサンプルのようになります。

図 9-7 入力の完了したプレゼンテーション ノード用プロパティ エディタのサンプル

3. Webflow に必要なノードがすべて追加されるまで、キャンバスへのノードの追加を続けます。

ステップ 3: 開始ノードを指定する

開始ノードは、訪問者のアプリケーション操作の出発点として用いられます。開始ノードは Webflow の最初のエントリ ポイントすなわち初期状態として指定されるもので、プレゼンテーション ノードまたはプロセッサ ノードに自動的に遷移します。開始ノードは通常、JSP 形式のプレゼンテーション ノードです。

URL で起点、ネームスペース、またはイベントが指定されていない場合には、Webflow メカニズムがデフォルト ネームスペース内で開始ノードを検索します。開始ノードは省略可能ですが、デフォルト ネームスペースに最低 1 つ定義しておくことをお勧めします。

ノードを開始ノードに指定するには、以下の手順に従います。

1. 開始ノードに指定したいノードを右クリックします。

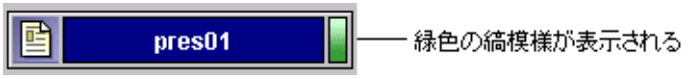
   図 9-8 に示すように、ノードのコンテキスト メニューが表示されます。

図 9-8　プレゼンテーション ノードのコンテキスト メニュー

2. [**開始ノードを設定**] を選択します。

   開始ノードに指定された印として、図 9-9 に示すように、ノード名の右側に緑色の縞模様が表示されます。

図 9-9　開始ノードの外見



注意： Webflow に存在できる開始ノードは 1 つだけです。

ステップ 4: ノード間の遷移を作成する

ノードをすべてキャンバス上に配置したら、それらの間の遷移を作成してノード同士を接続します。これらの遷移の作成には、表 9-2 に示すような [イベント ツール] と [例外ツール] を使用します。[例外ツール] は赤で、[イベント ツール] は黒で示されます。

イベント遷移を追加する

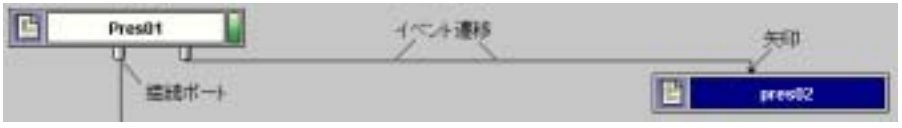
イベント遷移を追加するには、以下を実行します。

1. [イベント ツール] をクリックして、アクティブにします。
2. 遷移元ノードの外縁にポインタを持っていきます。

   接続が許される場合には、そのノードの外縁にオレンジ色の小さい四角形が表示されます。
3. 遷移先ノードまでポインタをドラッグします。

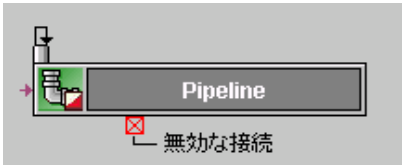
   灰色の接続ポートが遷移元ノードに表示され、接続が許される場合には、図 9-10 に示すように、遷移先ノードに矢印が表示されます。

図 9-10　ノード間のイベント遷移



接続が許されない場合（たとえば、プレゼンテーション ノードに例外リンクを作成しようとする場合など）には、図 9-11 に示すように、該当するノードの外縁に **X** 印を囲む赤い輪郭だけの四角形が表示されます。

図 9-11　無効な接続を示すマーク



4. イベント遷移の追加を必要なだけ続け、イベントに応答するノードをすべて接続します。

例外遷移を追加する

例外処理が必要と思われるすべてのノードを、その例外処理を行うノードに接続するには、[例外ツール] を使用します。たとえば、受け渡しの前に検証しておく必要があるデータを入力プロセッサが受け取るとしましょう。そのデータにエラーが含まれていたら、入力プロセッサは例外を送出する必要があるでしょう。[例外ツール] を使用すれば、例外を処理し、その結果などの情報を表示することができます。例外の場合には、ノード間の遷移リンクは赤で表示されることに注意してください。

例外遷移を追加するには、以下を実行します。

1. [例外ツール] をクリックして、アクティブにします。
2. 遷移元ノードの外縁にポインタを持っていきます。

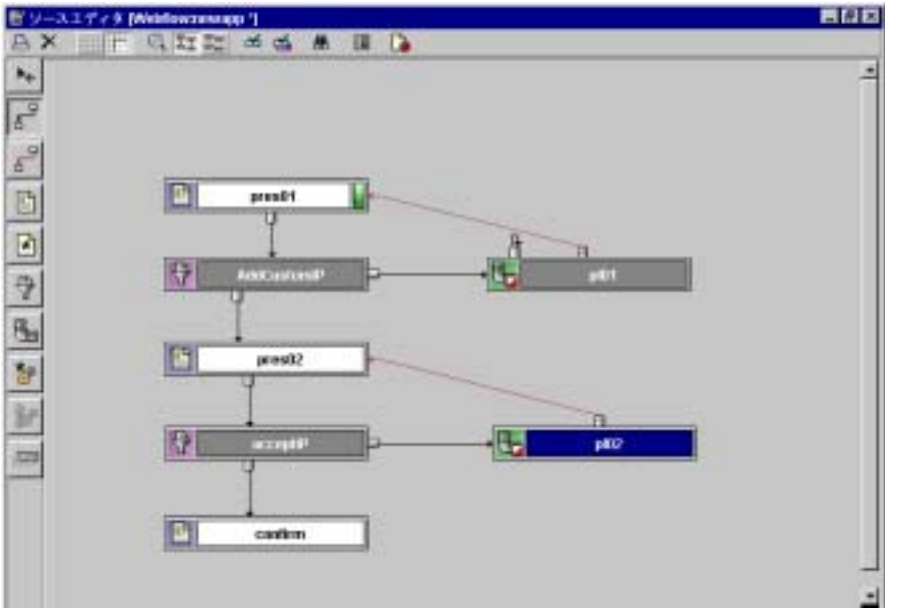
   接続が許される場合には、そのノードの外縁にオレンジ色の小さい四角形が表示されます。
3. 遷移先ノードまでポインタをドラッグします。

   灰色の接続ポートが遷移元ノードに表示され、接続が許される場合には、遷移先ノードに矢印が表示されます。

   接続が許されない場合（たとえば、プレゼンテーション ノードに例外リンクを作成しようとする場合など）には、図 9-11 に示すように、該当するノードの外縁に **X** 印を囲む赤い輪郭だけの四角形が表示されます。
4. 例外遷移の追加を必要なだけ続け、例外に応答するノードをすべて接続します。

ノードがすべて正しく配置され、適切な遷移で接続されたら、Webflow は 図 9-12 に示すようなものになるでしょう。

図 9-12　Webflow のサンプル レイアウト



遷移ツールの使い方

遷移の追加（9-16 ページの「イベント遷移を追加する」と 9-17 ページの「例外遷移を追加する」で説明) だけでなく、既存遷移の接続ポートを移動したり、遷移にエルボーを追加したり、あるいは遷移を削除することもできます。

接続ポートの移動：遷移を受け入れる接続ポートは入力接続ポートと呼ばれ、遷移の起点となる接続ポートは出力接続ポートと呼ばれます。場合によっては、ノードの接続ポートを移動すると有益なことがあります。ノード上の接続ポートの位置を変更するには、以下の手順に従います。

1. 遷移ツール（[イベント ツール] および [例外ツール] ）または [選択ツール] を選択します。
2. 接続ポート上でクリックし、マウス ボタンを押したまま、その接続ポートをノード上の目的の位置までドラッグします。
3. マウス ボタンを放すと、新しい位置に接続ポートが配置されます。

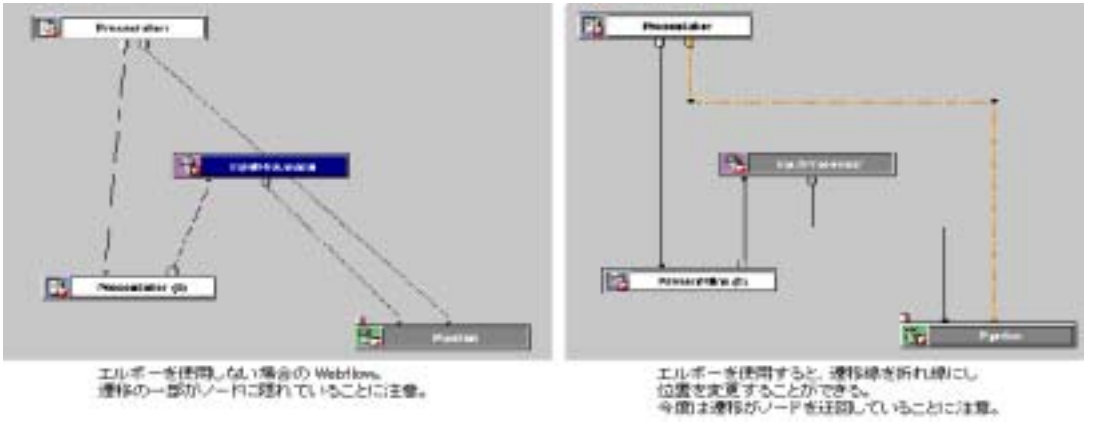
自己参照遷移に関連付けられている接続ポートの場合には、同じノードの外縁上にしか移動できません。

別ノードへの遷移の移動 : 遷移の終点（矢印で示される）をあるノードから別のノードへ移動することができます（接続が許される場合）。それには、以下を実行します。

1. 遷移ツールまたは [選択ツール] を選択します。
2. 移動したい終点（矢印）を選択します。
3. 左マウス ボタンを押したまま、新しい接続先ノードまで矢印をドラッグします。
4. マウス ボタンを放します。

遷移線でのエルボーの作成、移動、削除 : エルボーを移動、作成、または削除することにより、エディタ キャンバス上で遷移線の位置を変更することもできます。**エルボー**を利用すれば、図 9-13 に示すように遷移線を折れ線にして、フローをわかりやすくすることができます。

図 9-13　遷移線におけるエルボー



新しいエルボーを作成するには、以下の手順に従います。

1. 遷移をシングル クリックして既存のエルボーを表示させます。エルボーは黒い四角形で表示されます。

2. 遷移線の任意の箇所（既存エルボー上を除く）でクリックし、マウス ボタンを押したままドラッグすると、新しいエルボーが追加されます。選択されたエルボーは、オレンジ色の四角形で表示されます。
3. マウス ボタンを放すと、その位置にエルボーが作成されます。

既存の遷移内のエルボーを移動するには、以下の手順に従います。

1. 遷移をシングル クリックして既存のエルボーを表示させます。エルボーは黒い四角形で表示されます。
2. エルボーをクリックし、マウス ボタンを押したまま目的の位置までドラッグします。選択されたエルボーは、オレンジ色の四角形で表示されます。
3. マウス ボタンを放すと、そのエルボーが新しい位置に配置されます。

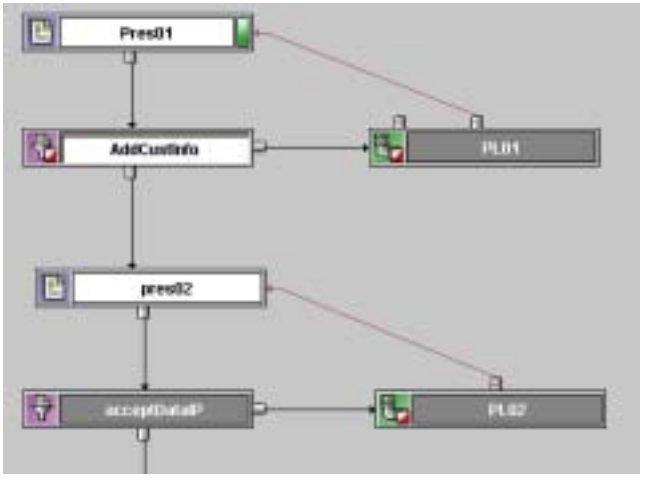
既存のエルボーを削除するには、以下の手順に従います。

1. 遷移をシングル クリックして既存のエルボーを表示させます。エルボーは黒い四角形で表示されます。
2. 削除したいエルボーをクリックして選択します。選択されたエルボーは、オレンジ色の四角形で表示されます。
3. エディタのツールバーの [削除] ボタンをクリックするか、〔Delete〕キーを押します。

Pipeline の作成と Webflow への追加

Webflow の詳細表示を図 9-14 に示します。この例では、PL01 および PL02 というラベルの付いた 2 つのノードが表示されています。これらのノードは Pipeline と呼ばれます。

図 9-14　Webflow の例 : Pipeline ノードを示す詳細表示



Pipeline はプロセッサ ノードの一種で、通常、Webflow でバックエンド ビジネス ロジックを実行するのに用いられます。各 Pipeline は、特定のタスクを実行する多数の Pipeline コンポーネントから成ります。WebLogic Portal 製品スイートには、新たに作成する Pipeline で再利用していただけるような Pipeline コンポーネントが多数付属しています。ただし、所属組織の特定のビジネス プロセスを実行する Pipeline コンポーネントを独自に作成したほうがよい場合もあります。

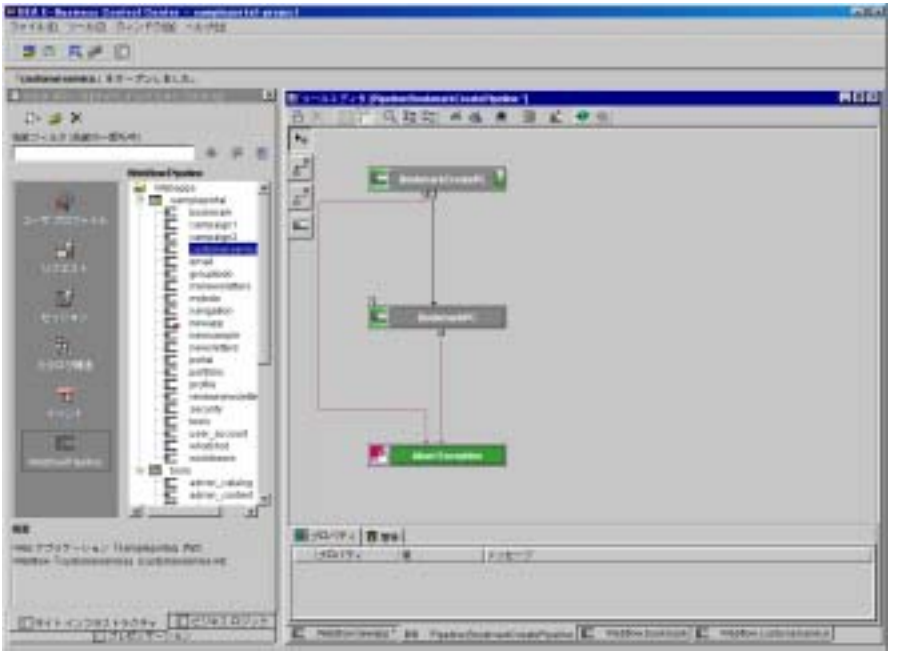
この節では、以下の手順に従って Pipeline を作成する方法を示します。

- ステップ 1: 新しい Pipeline コンポーネントを作成する
- ステップ 2: 新しい Pipeline コンポーネントを Webflow に追加する

Pipeline エディタの概要

Pipeline を作成し、そこに機能を追加するには、Webflow の場合と同様に、E-Business Control Center で Pipeline エディタ（図 9-15 および 図 9-18 を参照）を用います。

図 9-15　Pipeline エディタの表示された E-Business Control Center



Pipeline エディタの上部には、一連の表示および動作制御ボタンと一連のコマンド ボタンが用意されており、左端には一連のツール ボタンが用意されています。ツール ボタン、表示および動作制御ボタン、コマンド ボタンについては、それぞれ表 9-5、表 9-6、表 9-7 で説明します。

表 9-5　Pipeline エディタのツール

ツール	機能	解説
	[選択ツール]	Pipeline コンポーネント、イベント遷移、および例外遷移の選択や移動に用いる。遷移へのエルボーの追加にも用いる。Pipeline エディタのデフォルト ツール。 **注意：** このツールをいったん選択すると、別のツールを選択するまで、選択された状態のままになる。

表 9-5 Pipeline エディタのツール（続き）

ツール	機能	解説
	[イベント ツール]	2 つのノード間のイベント遷移を追加するのに用いる。 **注意：** このツールをいったん選択すると、別のツールを選択するまで、選択された状態のままになる。
	[例外ツール]	2 つのノード間の例外遷移や、自己参照例外遷移を追加するのに用いる。 **注意：** このツールをいったん選択すると、別のツールを選択するまで、選択された状態のままになる。
	[開始ノード]	エディタ キャンバス上の既存の Pipeline コンポーネントのいずれか 1 つを現在の Pipeline の開始ノードに指定するのに用いる。
	[Pipeline コンポーネント ノード]	エディタ キャンバスに新しい Pipeline コンポーネントを追加するのに用いる。

表 9-6 Pipeline エディタの表示および動作制御ボタン

ツール	機能	解説
	[背景のグリッド線を表示 / 非表示] ボタン	エディタ キャンバスにおける背景のグリッド線の表示 / 非表示を切り替えるのに用いる。
	[グリッドへの位置合わせを有効 / 無効にする] ボタン	マウス ボタンを放したとき、Webflow コンポーネントをエディタ キャンバス上の最も近いグリッド点に自動的に配置するかどうかを制御するのに用いる。
	[リンク最適化を有効 / 無効にする] ボタン	エディタ キャンバス上でノードを移動したとき、それに合わせて各ノード上のコネクタもノードの周囲に自動的に移動するかどうかを制御するのに用いる。

表 9-6 Pipeline エディタの表示および動作制御ボタン（続き）

ツール	機能	解説
	[例外イベントを表示 / 非表示] ボタン	エディタ キャンバスにおける例外遷移の表示 / 非表示を切り替えるのに用いる。
	[この Pipeline をトランザクション対応にする] ボタン	Pipeline をトランザクション対応にするかどうかを指定するのに用いる。 **注意：** Pipeline エディタでのみ利用可能。
	[Pipeline セッションをトランザクションに組み込む] ボタン	Pipeline セッションをトランザクションに組み込むかどうかを指定するのに用いる。[この Pipeline をトランザクション対応にする] ボタンがオンになっている場合のみ使用可能。 **注意：** Pipeline エディタでのみ利用可能。

Pipeline エディタでは、Webflow エディタで使用されるのと同じコマンド ボタン（表 9-3 を参照）と、表 9-7 に示すボタンが使用されます。

表 9-7 Pipeline エディタのコマンド ボタン

ツール	機能	解説
	[Pipeline コンポーネント エディタを起動] ボタン	Pipeline コンポーネント エディタを開く。このエディタを用いて、Pipeline コンポーネントを管理できる。 **注意：** Pipeline エディタでのみ利用可能。

これらのボタンの機能と使い方を理解すれば、Pipeline の作成を迅速かつ容易に行えるようになります。

ステップ 1: 新しい Pipeline コンポーネントを作成する

この節では、Webflow で使用する Pipeline コンポーネントを作成するのに必要な手順について説明します。

注意： この手順では、以下の事がらを前提としています。

- 既存のプロジェクトを開いてあるか、もしくは新しいプロジェクトを作成してある。
- E-Business Control Center が動作している。
- [**サイト インフラストラクチャ**] タブが選択されている。

1. エクスプローラの左ペインで、[Webflow/Pipeline] をクリックします。エクスプローラの右ペインに Webflow と Pipeline のリストが表示されます。
2. [Webflow/Pipeline] リストで、Pipeline Namespaces フォルダを展開します。
3. Pipeline のホストとなるネームスペースを開きます。Pipeline の場合には、ネームスペースは、単一のエンタープライズ アプリケーションで複数の Pipeline を使用しても名前の衝突が発生しないように、Pipeline のスコープを限定するのに用いられます。
4. [名前フィルタ] の上の [**新規作成**] アイコンをクリックして、図 9-16 に示すように [**新規作成**] メニューを開きます。

図 9-16　選択された Pipeline ネームスペースに対する [新規作成] メニュー

5. [**Webflow/Pipeline**] を選択します。

図 9-17 に示すような [Webflow/Pipeline の新規作成] ウィンドウが開きます。

図 9-17　[Webflow/Pipeline の新規作成] ウィンドウ

6. 以下を実行して、新しい Pipeline を作成します。

   a. [**新しい Pipeline**] ラジオ ボタンを選択します。

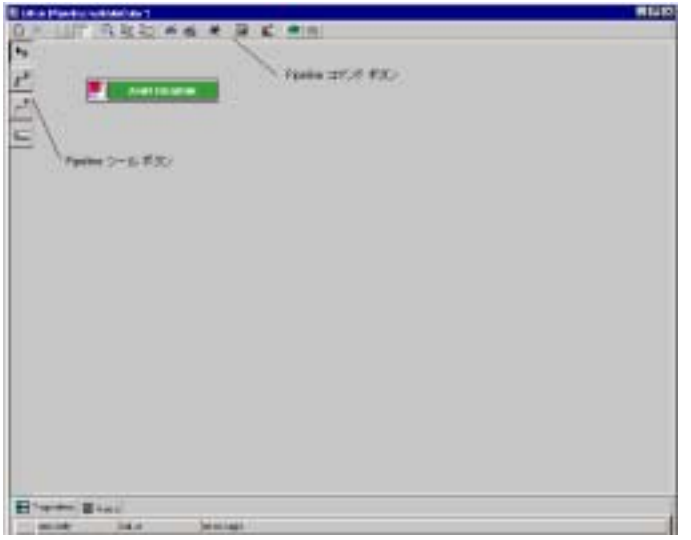
   b. [**ネームスペース**] リスト ボックスから、Pipeline のホストとなるネームスペース（たとえば、`bookmark`）を選択します。

   c. [**Pipeline 名**] フィールドに、Pipeline の名前（たとえば、「`validate Bookmark`」）を入力します。

   d. [**OK**] ボタンをクリックします。

      図 9-18 に示すような Pipeline エディタが表示されます。

図 9-18　Pipeline エディタ [Pipeline: validateData]



Pipeline エディタが開くと、アボート例外 (Abort Exception) ノードがキャンバス上に自動的に置かれることに注意してください。このノードは例外処理に用いられるもので、通常の処理で例外を解決できない場合に Webflow をアボートさせます。

7. 以下を実行して、最初の Pipeline コンポーネントを作成します。

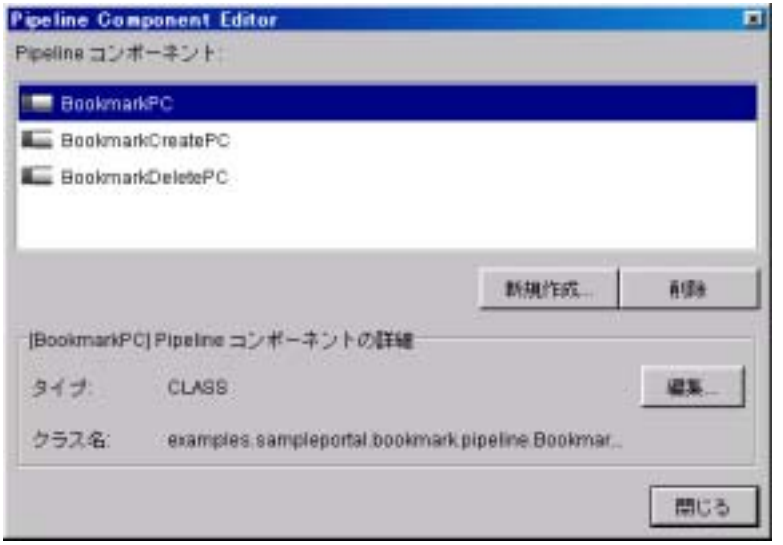
   a. Abort Exception ノードをクリックし 、マウス ボタンを押したまま、キャンバスの下部付近までドラッグします。

   b. [Pipeline コンポーネント ノード] ツール（表 9-5 を参照）を選択します。

   c. Pipeline エディタ キャンバスで、新しい Pipeline コンポーネントを置く場所を Abort Exception ノードより上のどこかに決めて、そこに十字線ポインタを持っていきます。そこでクリックして、新しい Pipeline コンポーネント ノードをエディタ キャンバス上に配置します。

   d. [Pipeline コンポーネント エディタを起動] ボタン（表 9-7 を参照）をクリックします。図 9-19 に示すような [Pipeline Component Editor] ウィンドウが開きます。

図 9-19　[Pipeline Component Editor]



e. [Pipeline Component Editor] ウィンドウで、[**新規作成**] をクリックします。図 9-20 に示すような [Pipeline コンポーネント作成ツール] ウィンドウが開きます。

図 9-20　[Pipeline コンポーネント作成ツール]

f. [**名前**] に、Pipeline コンポーネントの名前（たとえば「`validateBookmark`」）を入力します。

g. [**タイプ**] として [**Class**] を選択します。

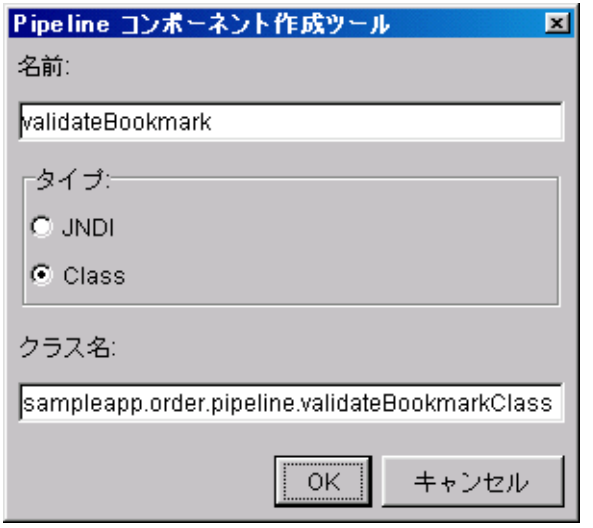
Pipeline コンポーネントは通常の Java クラスかセッション EJB のどちらかなので、[**タイプ**] 値は重要です。[**Class**] を選択すれば通常の Java クラスを実装することになるのに対して、[**JNDI**] を選択すればセッション EJB を実装することになります。

h. [**クラス名**] に、Pipeline コンポーネントの参照先クラスの完全パッケージ名を入力します。たとえば、以下のように入力します。

```
examples.wlcs.sampleapp.order.pipeline.ValidateBookmarkClass
```

入力の完了したダイアログ ボックスは、図 9-21 に示す例のようになるでしょう。

図 9-21 入力の完了した [Pipeline コンポーネント作成ツール] ダイアログ ボックス



i. [**OK**] をクリックして、[Pipeline コンポーネント作成ツール] ウィンドウを閉じます。

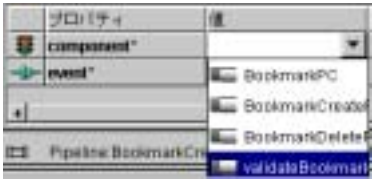
j. 新しい Pipeline コンポーネントが、[Pipeline Component Editor] ウィンドウの [**Pipeline コンポーネント**] リストに表示されます。

[Pipeline Component Editor] ウィンドウで、[**閉じる**] をクリックします。

k. 新しい Pipeline ノードを選択します。

l. キャンバスの下にあるプロパティ エディタで、図 9-22 に示すように、component*/[値] ポップアップ リストから、ステップ f で入力した Pipeline コンポーネント名を選択します。

図 9-22　プロパティ エディタ ― コンポーネントの選択



新しいコンポーネント名が [**component*/ 値**] セルに表示されます。

8. この例では、例外が発生すれば必ず Pipeline を終了するように設定します。そのため、新しいノードを `Abort Exception` ノードに接続する必要があります。以下の手順に従って、新しい Pipeline コンポーネントを `Abort Exception` ノードに接続します。

   a. [例外ツール] をクリックします。

   b. `validateBookmark` ノードの下縁にマウス カーソルを移動して、遷移の位置を決めます。オレンジ色で塗りつぶされた四角形が表示されることで、接続可能な位置であることが示されます。また、カーソルが遷移の追加を示すものに変わります。

   c. その位置でクリックし、マウス ボタンを押したまま、`Abort Exception` ノードまでドラッグします。マウス ボタンを放すと、遷移が `Abort Exception` ノードに接続されます（図 9-23）。

図 9-23　validateBookmark – Abort Exception 間の接続

d. その例外遷移を選択します。遷移線がオレンジ色に変わります。

e. プロパティ エディタで、第 2 列を選択し（図 9-24 を参照）、以下のように例外クラスのパッケージ名を入力します。

```
com.bea.p13n.appflow.exception.PipelineException
```

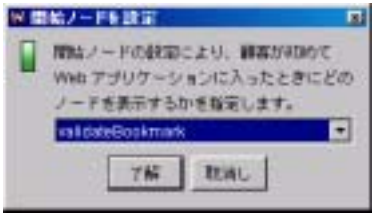
図 9-24　プロパティ エディタ ― 例外の値

9. 以下のようにして、`validateBookmark` ノードを開始ノードにします。

a. [開始ノード] ツール（9-23 ページの「[開始ノード]」を参照）をクリックします。

図 9-25 に示すように、開始ノードを作成しようとしていることを示すメッセージが表示されます。

図 9-25　開始ノード メッセージ ボックス



b. メッセージ ウィンドウで [**了解**] をクリックします。

c. validateBookmark が開始ノードになります。そのことは、図 9-26 に示すように、ノードの右縁に緑色のバーが表示されることでわかります。

図 9-26　開始ノードになった validateBookmark

10. E-Business Control Center のメイン ツールバーから [**ファイル**｜**保存**] を選択して、新しい Pipeline を保存します。

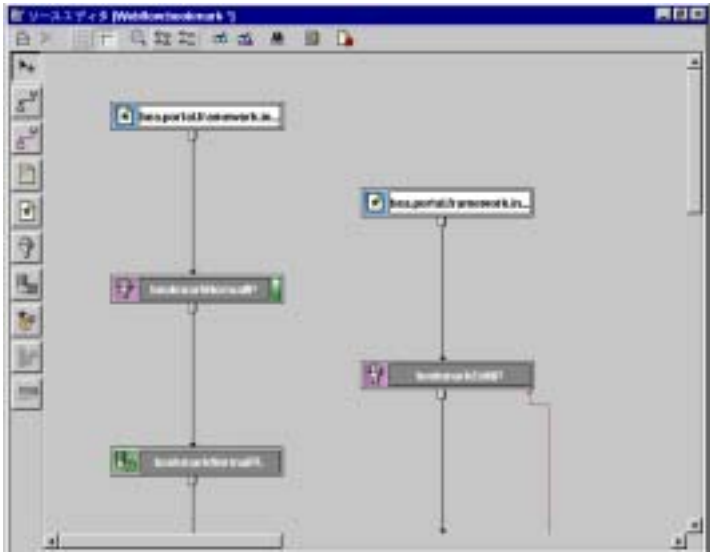
ステップ 2: 新しい Pipeline コンポーネントを Webflow に追加する

さて次は、新しい Pipeline コンポーネントを Webflow に追加する必要があります。それには、以下の手順に従います。

1. エクスプローラの右ペインで、Pipeline の追加先となる Web アプリケーションを開きます。その Web アプリケーションで、Pipeline の追加先となる Webflow をダブルクリックします。

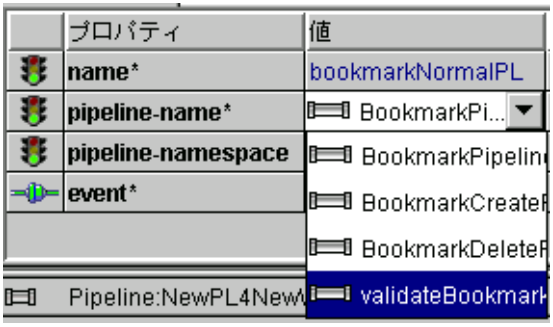
   図 9-27 に示すような Webflow エディタが開きます。

図 9-27　完成した Webflow を示す Webflow エディタ



2. `bea.portal.framework...` ノードと `bookmarkEditIP` ノードの間の遷移を選択し、〔**Delete**〕を押します。
3. 以下のようにして、`validateBookmark` Pipeline を Webflow に追加します。
   a. [Pipeline ノード] ツール（表 9-2 を参照）をクリックし、`bea.portal.framework...` ノードと `bookmarkEditIP` ノードの間にポインタを持っていきます。そこで 1 回クリックして、エディタ キャンバス上のその位置に新しい Pipeline ノードを追加します。
   b. キャンバスの下にあるプロパティ エディタで、**pipeline-name*** 行の第 2 列を選択したあと、図 9-28 に示すように、ドロップダウン リストから `validateBookmark` を選択します。

図 9-28　プロパティ エディタ ― コンポーネントの選択



4. 以下のようにして、validateBookmark Pipeline ノードを他の Pipeline ノードに接続します。

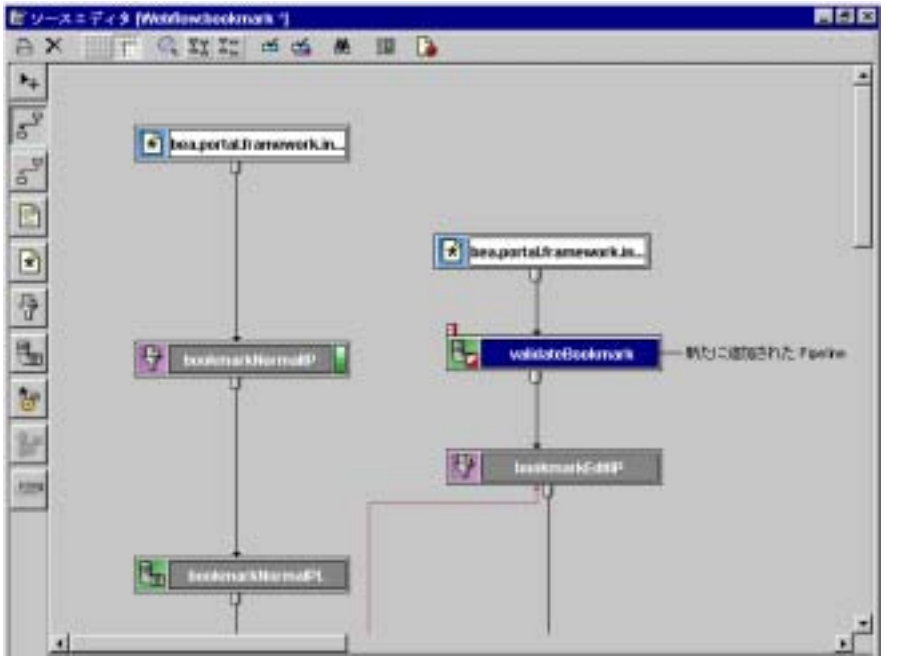
   a. [イベント ツール] （表 9-5 を参照）を用いて、validateBookmark ノードを bea.portal.framework... ノードに接続し、作成したイベント遷移をクリックしたあと、プロパティ エディタの [プロパティ] タブで、イベントの名前を「success」に変更してから〔Enter〕を押します。

   b. [イベント ツール] を用いて、validateBookmark ノードを bookmarkEditIP ノードに接続します。

   c. 作成したイベント遷移をクリックしたあと、プロパティ エディタの [プロパティ] タブで、（必要なら）イベントの名前を「success」に変更してから〔**Enter**〕を押します。

   上記の作業が完了したあとの Webflow エディタ キャンバスを図 9-29 に示します（図が見やすくなるようにノードを移動してあります）。

図 9-29 イベント遷移



5. 変更内容を保存するには、E-Business Control Center のメイン ツールバーから [**ファイル**｜**保存**] を選択します。

アプリケーションへの Webflow の同期化

上記で作成した Webflow が機能するためには、それを Web アプリケーションに同期化する必要があります。データの同期化によって、Webflow と Pipeline の XML 定義がデータベースとマスター データ リポジトリ（インメモリ データ ストア）にロードされます。

警告 : アプリケーション データはすべて、一斉に同期化されます。同時に複数の開発者が単一のエンタープライズ アプリケーションに対してデータを同期化すると、お互いの作業結果を上書きしたり、デバッグしにくい矛盾した変更を作り込んでしまうおそれがあります。そのような事態にな

らないようにするには、アプリケーションの別個のインスタンスに対して同期化を行います。

新しい Pipeline コンポーネントと変更済みの Webflow を同期化するには、以下の手順に従います。

1. WebLogic Server を起動し、E-Business Control Center を開きます。
2. [**ツール**｜**同期**] を選択します。

   数秒後に、同期進捗メーター（図 9-30）が表示されます。

図 9-30　同期進捗メーター

   このメーターは、同期が実行中であることを示します。処理が完了すると、進捗メーターにはそのことが表示され、[**停止**] が [**閉じる**] に変わります。

3. [**閉じる**] をクリックします。

これで、Webflow がアプリケーションに同期化されました。

入力プロセッサの新規作成

すでに述べたように、入力プロセッサは定義済みの専用 Java クラスであり、Webflow メカニズムから呼び出されて、より複雑なタスクを実行します。入力プロセッサは通常、HTML フォーム データの検証や、Web ページ内での条件分岐の実現に用いられます。WebLogic Portal には、開発済みの入力プロセッサが多数付属しています。開発するアプリケーションではこれらの入力プロセッサを再利用したほうがよい場合がある一方、独自の入力プロセッサを作成してアプリケーションの Webflow で使用するほうがよい場合もあるでしょう。

InputProcessor インタフェースを用いて入力プロセッサを作成する

新しい入力プロセッサを作成するには、process() メソッド（コード リスト 9-1 に示す）の詳細を用意することによって、com.bea.p13n.appflow.webflow.InputProcessor インタフェースを実装する必要があります。

コード リスト 9-1　InputProcessor インタフェースの Process() の実装

```
public java.lang.Object process(javax.servlet.http.HttpServletRequest req,
                                java.lang.Object requestContext)
                         throws ProcessingException
```

このインタフェースは、HttpSession 内に存在する HttpServletRequest や PipelineSession を処理します。リターン オブジェクトは何でもかまいませんが、そこに用意されている **toString()** の実装は意味のあるものでなければなりません。Webflow エグゼキュータは、返されたオブジェクトに対して toString() を呼び出して、プロセッサ用のイベントを生成します。

パラメータ	req - HttpServletRequest オブジェクト requestContext - リクエストを一意に識別する Object
戻り値	Object（toString() を意味のあるものに実装してあること）
送出する例外	ProcessingException または、そのサブクラス

入力プロセッサの命名規約

入力プロセッサの名前の末尾には、「IP」という接尾辞を付けなければなりません。たとえば、届け先住所の削除を担当する入力プロセッサには、DeleteShippingAddressIP という名前を付けるといった具合です。こうした命名規約に従うことで、入力プロセッサがより把握しやすくなるはずです。

入力プロセッサでのビジネス ロジックの実行

入力プロセッサ内では、通常、ビジネス（アプリケーション）ロジックは実行しない方がよいでしょう。特に、入力プロセッサでは、Enterprise JavaBeans（EJB）を呼び出したり、データベースへのアクセスを試みるべきではありません。そのようなロジックはすべて、Pipeline コンポーネントに実装すべきです。そのロジックを入力プロセッサ内で実行することも可能ですが、そうしたロジックはトランザクション対応でないことがあり、その場合には Webflow アーキテクチャの主たる目的を達成できなくなります。ビジネス ロジックとプレゼンテーション ロジックを分離することで、Web サイトの柔軟性が本質的に高くなります。機能の変更や追加は、Pipeline あるいは入力プロセッサを新たに作成してプラグインする程度の単純な作業になります。

InputProcessorSupport クラスを拡張する

あるいは、コード リスト 9-2 に示すような `com.bea.p13n.appflow.webflow.InputProcessorSupport` クラスを拡張する入力プロセッサを新たに作成することもできます。名前のとおり、この抽象クラスを使用すれば、入力プロセッサの働きを補助する静的ヘルパー メソッドを利用できるようになります。ただし、新しい入力プロセッサ クラスが何か他のクラスを拡張する必要がある場合には、`InputProcessorSupport` クラスを利用することはできません。

コード リスト 9-2　`InputProcessorSupport` クラスの拡張

```
public abstract class InputProcessorSupport
                        extends java.lang.Object
                implements InputProcessor, ValidatedFormConstants
```

注意： `InputProcessorSupport` クラスの実装の詳細については、`com.bea.p13n.appflow.webflow` の『Javadoc』を参照してください。

キャンバス上に配置した入力プロセッサ ノードのプロパティを、Webflow エディタを用いて指定する場合には、新たに作成した入力プロセッサのクラス名を該当するフィールドに入力すればよいだけです。入力プロセッサを既存の Webflow メカニズムと連携させるために必要な作業は、それ以外にはありません。

拡張プレゼンテーション ノードと拡張プロセッサ ノードの作成による Webflow の拡張

入力プロセッサと Pipeline コンポーネントを新たに作成して、製品にあらかじめ用意されているものに追加しても、必要を満たせない場合には、拡張（カスタム）プレゼンテーション ノードや拡張（カスタム）プロセッサ ノードとして使用できるクラスを作成して、Webflow メカニズムを拡張することにしてもかまいません。これらのノードに関連付けられるクラスを作成したら、新しいノードを `webflow-extensions.wfx` ファイルに登録する必要があります。この節では、これらの作業の実行方法を示します。

拡張プレゼンテーション ノードの作成方法

拡張（カスタム）プレゼンテーション ノードを作成するには、以下の手順に従います。

1. `com.bea.p13n.appflow.webflow.PresentationNodeHandler` インタフェースを実装するクラスを作成します。作成するクラスは、必ず `WebflowServlet` サーブレットの転送先の URL を返すようにしてください。
2. 作成した拡張ノードを `webflow-extensions.wfx` ファイルに登録して、Webflow エディタや Pipeline エディタで使用できるようにします。9-41 ページの「拡張プレゼンテーション ノードおよび拡張プロセッサ ノードを Webflow エディタや Pipeline エディタで使用可能にする」を参照してください。

WebLogic Portal では、`portal` という拡張（カスタム）プレゼンテーション ノードが使用されるので、このノードを一例と考えることができます。Portal では、この拡張ノードを用いて、ポートレットのコンテンツが変更されていない（つまり、最後の URL を表示すべきである）ことをポータル Webflow に知らせます。`portal` ノードの実装クラスは `LastContentUrlNodeHandler.java` です。

拡張プロセッサ ノードの作成方法

拡張（カスタム）プロセッサは、（BEA ではなく）開発者の所属組織がアプリケーションの Webflow 用に開発するプロセッサです。たとえば、入力プロセッサや Pipeline プロセッサと同じレベルで動作する拡張（カスタム）プロセッサを作成したいとしましょう。拡張プロセッサは、Webflow で現在サポートされていない処理を実行するのに用いることができます。ただし、拡張プロセッサの入出力フローは、やはり Webflow メカニズムによって制御されます。拡張プロセッサは Webflow エディタでは、入力プロセッサノードや Pipeline ノードによく似たノードとして表現されますが、表現が多少異なるので、容易に識別できます。

たとえば、BEA ルール エンジンと連携して、何らかの条件（顧客セグメントへの所属の有無など）に基づくさまざまな Webflow をサポートする拡張（カスタム）プロセッサを作成するといった場合もあるでしょう。また、もっと単純な例としては、ページ本文のコンテンツが与えられたときに JSP にヘッダーとフッターを自動的に付加するレイアウト マネージャ プロセッサがあります。実際に、そうしたプロセッサはすでに作成されています。

拡張（カスタム）プロセッサ ノードを作成するには、以下の手順に従います。

1. `com.bea.p13n.appflow.webflow.Processor` インタフェースを実装するクラスを作成して、拡張プロセッサを定義します。`Processor` インタフェースの典型的な実装をコード リスト 9-3 に示します。

コード リスト 9-3　`Processor` インタフェースの実装

```
public java.lang.Object process(java.lang.String webapp,
                                java.lang.String namespace,
                                javax.servlet.http.HttpServletRequest request,
                                java.lang.Object requestContext)
                         throws java.lang.Exception
```

このインタフェースは、リクエストで示されるプロセッサを実行します。リターン オブジェクトは何でもかまいませんが、そこに用意されている `toString()` の実装は意味のあるものでなければなりません。Webflow エグ

ゼキュータは、返されたオブジェクトに対して `toString()` を呼び出して、プロセッサ用のイベントを生成します。

パラメータ	`webapp` - Web アプリケーション名を格納する String `namespace` - ネームスペース名を格納する String `name` - プロセッサ名（`foo.inputprocessor`、`bar.pipeline`）を格納する String `request` - HttpServletRequest `requestContext` - 関連付けられているリクエストを一意に識別する Object オブジェクト
戻り値	結果を表す Object。何でもよいが、`toString()` を実装する必要がある。
送出する例外	`java.lang.Exception`（エラーが発生した場合）

2. 作成したプロセッサを `webflow-extensions.wfx` ファイルに登録して、Webflow エディタや Pipeline エディタで使用できるようにします。

注意： `webflow-extensions.wfx` ファイルに拡張ノードを登録する方法については、9-41 ページの「拡張プレゼンテーション ノードおよび拡張プロセッサ ノードを Webflow エディタや Pipeline エディタで使用可能にする」を参照してください。

拡張プレゼンテーション ノードおよび拡張プロセッサ ノードを Webflow エディタや Pipeline エディタで使用可能にする

拡張（カスタム）プレゼンテーション ノードや拡張（カスタム）プロセッサ ノードを作成したら、そのノードを `webflow-extensions.wfx` ファイルに登録して、チームの他の開発者から利用できるようにする必要があります。

注意： webflow-extensions.wfx ファイルは <BEA_HOME>/user_projects/myNEWDomain/beaApps/portalApp-project/default/webflow/ フォルダ内にあります（ここで、<BEA_HOME> は WebLogic Portal のインストール先ディレクトリです）。たとえば、以下のフォルダです。

```
bea/user_projects/myNEWDomain/beaApps/portalApp-project/
   default/webflow/
```

拡張（カスタム）プロセッサ ノードを登録すると、E-Business Control Center の再起動後に、Webflow エディタ パレット上の対応するツールが使用可能になります。

拡張プレゼンテーション ノードを登録する

拡張プレゼンテーション ノードを webflow-extensions.wfx ファイルに登録するには、以下の手順に従います。

1. webflow-extensions.xml ファイルを開きます。このファイルは、<BEA_HOME>/user-projects/myNEWDomain/beaApps/portalApp-project/application-synch/webapps/<APPLICATION> フォルダにあります（ここで、<BEA_HOME> は BEA 親ディレクトリ、<APPLICATION> は特定の Web アプリケーション用ディレクトリです）。
2. <end-node> 要素を <end-node-registration> リストに追加します。
3. name 属性でプレゼンテーション ノード名を指定し、node-handler 属性で、基礎となるノード実装のクラスを指定します。
4. 呼び出し時に上記クラスで想定される入力パラメータを、<node-processor-input> 要素を用いて定義します。各パラメータの name 属性を入力し、省略可能なパラメータの場合には、required 属性の値を false にします。

   注意： この情報が、Webflow エディタおよび Pipeline エディタのプロパティ エディタで使用されることになります。
5. webflow-extensions.wfx ファイルを保存し、E-Business Control Center を再起動します。

拡張プロセッサ ノードを登録する

拡張プロセッサ ノードを webflow-extensions.wfx ファイルに登録するには、以下の手順に従います。

1. webflow-extensions.xml ファイルを開きます。このファイルは、<BEA_HOME>/user-projects/myNEWDomain/beaApps/portalApp-project/application-synch/webapps/<APPLICATION> フォルダにあります（ここで、<APPLICATION> は特定の Web アプリケーション用ディレクトリです）。
2. <process> 要素を <process-registration> リストに追加します。
3. name 属性でプロセッサ名を指定し、executor 属性で、基礎となるプロセッサ実装のクラスを指定します。
4. 呼び出し時に上記クラスで想定される入力パラメータを、<node-processor-input> 要素を用いて定義します。各パラメータの name 属性を入力し、省略可能なパラメータの場合には、required 属性の値を false にします。

   注意： この情報が、Webflow エディタおよび Pipeline エディタのプロパティ エディタで使用されることになります。
5. webflow-extensions.wfx ファイルを保存し、E-Business Control Center を再起動します。

第 10 章ルック アンド フィールの作成

WebLogic Portal には、ポータルのルック アンド フィール（見た目と使い心地）を管理するための強力なメカニズムが 2 つ用意されています。すなわち、スキンとレイアウトです。スキンとは、ポータルのビジュアル スタイル（たとえば、色、フォント、アイコンなど）を定義するグラフィックスおよび HTML スタイル設定のことです。一方、レイアウトとは、ポートレットを配置できる HTML ベースのマトリクスのことです。WebLogic Portal プラットフォームには、これらのコンポーネントの作成、管理、およびさまざまなアプリケーションのユーザへの提供を行うための構造があらかじめ用意されています。

この章では、ポータルのスキンとレイアウトを作成する方法を示します。カスケーディング スタイル シート (CSS) の作成、グラフィックスの作成、および HTML テーブルの定義についての知識があらかじめ必要になります。

この章では、以下の内容について説明します。

- ポータルのルック アンド フィール構造
- スキンの使い方
- レイアウトの使い方

ポータルのルック アンド フィール構造

Web ブラウザでポータルを閲覧するとき、目に見えるものは統合された単一の Web ページですが、そのポータルは多数の JSP から成ります。たとえば、ポータルでは、ページ コンテンツだけでなく、ヘッダー、フッター、垂直および水平ナビゲーション バー用の JSP が個別に使用されることがあります。

ポータルの構造で用いられる各 JSP には、ポータル全体のルック アンド フィールの一部を実装するのに必要なコードが記述されています。たとえば、各 JSP には、ポータルのスキンを決定するための CSS およびグラフィックスへの参照を

記載することができ、また、ポータルのメイン コンテンツ ページには、そのポータル用に選ばれたレイアウトを実装するのに必要なロジックが記述されます。

E-Business Control Center で Portal Wizard を用いてポータルを作成する際には、**baseportal** というポータル テンプレートが製品にあらかじめ用意されています。このテンプレートの機能の 1 つは、ポータルの新規作成用のルック アンド フィール構造を定義する一連の定義済み JSP を使用することです。この章では、**baseportal** 構造を取り上げて、スキンおよびレイアウトの作成手順を示します。なお、この構造は、WebLogic Portal に付属している Avitek Financial ポータル サンプルにも使用されています。

スキンの使い方

*スキン*は、カスケーディング スタイル シート（.css ファイル）や画像ディレクトリなど、ポータルのルック アンド フィールを定義するファイルの集まりです。ボタン、バナー、ポートレット ヘッダー、背景色、およびフォント特性はすべて、CSS とグラフィックスで決まります。また、スキンにはそのスキンのサムネイル画像も含まれており、プレビュー目的に使用されます。

この節では、以下のトピックを扱います。

- スキンを作成する
- スキンを格納する
- スキンを利用可能にする

スキンを作成する

ポータルでスキンをシームレスに切り替えることができるようにするには、それぞれのスキンの CSS およびグラフィックスには、図 10-1 に示すように、他のスキンの CSS およびグラフィックスと同じ名前を付ける必要があります。

図 10-1　baseportal テンプレートを用いて作成されるポータルに適用されるさまざまなスキン

注意： 広告やポートレット内のグラフィックスなど、ページに表示されるコンテンツ グラフィックスはすべて、コンテンツ管理システム内か、ポートレット ディレクトリ下の画像サブディレクトリ内に格納されます。ただし、ポートレットのタイトルバー グラフィックスと背景色は、図 10-1 に示すように、スキンによって決まります。

スキンの作成は、CSS ファイルとグラフィックス ファイルを作成または修正し、他のスキンで使用されているのと同じ名前を付けて、別のディレクトリに格納するだけのことです。スキンの作成にはまた、プレビュー用のサムネイル画像の作成も必要になります。好みのグラフィックス ツールを使用して、スキンおよびサムネイルのグラフィックスを作成および修正します。

スキンのグラフィックスを作成する際には、画像が大きければ大きいほど、ページをブラウザにロードするのが遅くなることに注意してください。

定義済みのスキン

WebLogic Portal には、Portal Wizard でポータルを作成する際に用いられる一連の定義済みスキンが用意されています。これらのスキンは以下の場所にあります。

```
<BEA_HOME>\weblogic700\common\templates\webapps\portal\baseportal
\
j2ee\framework\skins
```

これらは、まったく新しいスキンを作成する際にテンプレートとして使用することができます。

スキンを格納する

次の節では、作成した新しいスキンの格納先となるディレクトリ構造について説明します。スキンをポータルで利用できるようにするには、ポータルのメタデータで参照されているディレクトリ構造に、CSS とグラフィックスを格納します。スキンを構成する必須要素をどこに格納すべきかを表 10-1 に示します。

スキンの構成要素のうち、スキン サムネイル画像だけは、サーバ上に格納されません。サムネイルは、選択されたスキンを E-Business Control Center でプレビューするためのものです。

表 10-1 スキン リソースの格納先

リソース	格納場所
スキン サムネイル画像	<BEA_HOME>\user_projects*your_domain*\beaApps\portalApp-project\library\portal\skins*skin_folder*

表 10-1 スキン リソースの格納先（続き）

リソース	格納場所
CSS ファイル	<BEA_HOME>\user_projects*your_domain*\beaApps\portalApp*your_portal*\framework\skins*skin_folder*\css
スキン グラフィックス	<BEA_HOME>\user_projects*your_domain*\beaApps\portalApp*your_portal*\framework\skins*skin_folder*\images

スキンを利用可能にする

スキンをポータルで利用できるようにするには、E-Business Control Center を用いてスキンをポータル定義に追加し、更新されたポータル定義をサーバに同期化したあと、WebLogic Portal Administration Tools を用いてそのスキンをポータルに適用することが必要になります。

管理目的でスキンを利用できるようにする手順については、『WebLogic Portal 管理者ガイド』に記載されています。「ポータルとポートレットの属性の管理」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/frmwork.htm) を参照してください。

レイアウトの使い方

レイアウトとは、ページ内の行と列を定義する HTML テーブルのことです。レイアウトは、HTML コード内に記述される特別定義のプレースホルダ タグに基づいて、ポータル ページへの Web コンテンツおよびポートレットの配置を決定するものです。

注意： レイアウトでは、フレームはサポートされません。

レイアウトは、HTML テーブル定義が記述された単一の JSP ファイルと、レイアウトのプレビュー用サムネイル画像から成ります。WebLogic Portal にあらかじめ用意されているレイアウトのサムネイル表現を図 10-2 に示します。

図 10-2　WebLogic Portal に用意されているデフォルト レイアウトのサムネイル

この節では、以下のトピックを扱います。

- レイアウトを作成する
- レイアウトを格納する
- レイアウトを利用可能にする

レイアウトを作成する

新しいレイアウトを作成して既存のポータルに追加するには、以下の手順に従います。

1. 既存のレイアウトを基に、作業を始めます。*your_portal*\framework\layouts\ ディレクトリで、既存のレイアウト フォルダのコピーを作成し、名前を変更します。その新しいディレクトリ内の template.jsp（図 10-3 に示す）をテキスト エディタで開きます。

図 10-3　3 列構成の template.jsp のソースコード

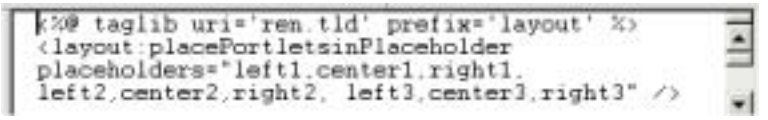
```
<%@ taglib uri='ren.tld' prefix='layout' %>
<layout:placePortletsinPlaceholder
placeholders="left,center,right" />

<table width="100%" border="0" cellspacing="0"
cellpadding="0">
  <TR>
    <TD WIDTH="33%" VALIGN="TOP">
      <layout:render section='left'/>
    </TD>
    <TD WIDTH="33%" VALIGN="TOP">
      <layout:render section='center'/>
    </TD>
    <TD WIDTH="33%" VALIGN="TOP">
      <layout:render section='right'/>
    </TD>
  </TR>
</TABLE>
```

layout:placePortletsinPlaceholder タグを用いたプレースホルダの定義に注目してください。また、テーブルの各セル内で layout:render タグが呼び出されていることにも注意してください。このタグで、プレースホルダを用いてポートレットのコンテンツがポータルに挿入されるのです。

2. 新しいプレースホルダ定義を追加します。図 10-4 に示すように、既存の template.jsp ファイル内の定義に新しいプレースホルダ名をいくつか挿入します。

図 10-4　新しいプレースホルダ名の追加



3. 図 10-5 に示すように、HTML でテーブルを作成し、各テーブル セルに新しいプレースホルダを配置します。

図 10-5　HTML コードへのプレースホルダの挿入

4. 新しいファイルを template.jsp として保存します。

5. 図 10-6 に示すように、グラフィック エディタを用いて、新しいレイアウトを表すように既存のサムネイルを編集し、それを 2 つの別個のファイル（thumbnail.gif と *layout_name*.gif）として保存します。ここで、*layout_name* はレイアウト フォルダの名前です。

図 10-6　レイアウト サムネイルの作成

WebLogic Portal には、Portal Wizard でポータルを作成する際に用いられる一連の定義済みレイアウトが用意されています。これらのレイアウトは以下の場所にあります。

```
<BEA_HOME>\weblogic700\common\templates\webapps\portal\
   baseportal\j2ee\framework\layouts
```

次の節では、新しいレイアウトの格納先となるディレクトリ構造について説明します。

レイアウトを格納する

レイアウトをポータルで利用できるようにするには、新しいレイアウトとサムネイル画像を、特定のディレクトリ構造内の 2 つの異なる場所に格納します。さらに、サムネイル画像には 2 つの異なる名前を使用します。これらのファイルの名前と格納場所を表 10-2 に示します。

表 10-2　スキン リソースの格納先

リソース	格納場所
レイアウト サムネイル画像 : *layout_name*.gif	<BEA_HOME>\user_projects*your_domain*\beaApps\portalApp-project\library\portal\layouts*layout_folder*
レイアウト サムネイル画像 : thumbnail.gif	<BEA_HOME>\user_projects*your_domain*\beaApps\portalApp*your_portal*\framework\layouts*layout_folder*

表 10-2 スキン リソースの格納先（続き）

リソース	格納場所
template.jsp	<BEA_HOME>\user_projects*your_domain*\beaApps\portalApp-project\library\portal\layouts*layout_folder* **および** <BEA_HOME>\user_projects*your_domain*\beaApps\portalApp*your_portal*\framework\layouts*layout_folder*

レイアウトを利用可能にする

レイアウトをポータルで利用できるようにするには、E-Business Control Center を用いてレイアウトをポータル定義に追加し、更新されたポータル定義をサーバに同期化したあと、WebLogic Portal Administration Tools を用いてそのレイアウトをポータルに適用することが必要になります。

レイアウトを利用できるようにする手順については、『WebLogic Portal 管理者ガイド』に記載されています。「ポータルとポートレットの属性の管理」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/frmwork.htm) を参照してください。

第 11 章 ポートレットの拡張

ポートレットの高度な機能を呼び出したり、それらのポートレットを管理者から利用できるようにするために、開発者はいくつかのツールと数々の手順を用います。この章では、以下のトピックを扱います。

- ポートレットの基本的なカスタマイズ
  - ポータル Web アプリケーション間でポートレットをコピーする
  - ドメイン間でポートレットをコピーする
  - ポートレットのカテゴリを作成する
- ポートレットとフレームワーク
  - 単純 JSP ポートレット
  - Webflow ポートレット
  - Web サービス ポートレット
- 既存の Web アプリケーションのポータル化
- パフォーマンス チューニング

ポートレットの基本的なカスタマイズ

最も基本的なカスタマイズの 1 つは、あるポータルから別のポータルへポートレットをコピーすることです。この手順を理解しておけば、この章で取り上げるその他の作業の多くを理解する上で役に立ちます。ここでは多くの手順とツールを紹介するからです。この節では、以下の基本的なポートレット カスタマイズを扱います。

- ポータル Web アプリケーション間でポートレットをコピーする
- ドメイン間でポートレットをコピーする
- ポートレットのカテゴリを作成する

ポータル Web アプリケーション間でポートレットをコピーする

この節では、一方のポータル Web アプリケーションからもう一方のポータル Web アプリケーションにポートレットをコピーするプロセスについて説明します。ここでは、どちらのポータルも正しくデプロイされているものとします。これら 2 つのポータルと、そのそれぞれに関連付けられている Web アプリケーションを 図 11-1 に示します。

図 11-1　ポータルとそれに関連付けられている Web アプリケーション

この例では、あるポータル Web アプリケーション (NewPWApp) から別のポータル Web アプリケーション (NextPWApp) に、`flowlet1` というポートレット (図 11-2 に示す) をコピーします。

図 11-2　NewWPApp 内の flowLet1 ポートレット

NextPWApp ポータルへ flowLet1 ポートレットを追加するには、以下の手順に従います。

ステップ 1: J2EE リソースを新しい Web アプリケーションにコピーする

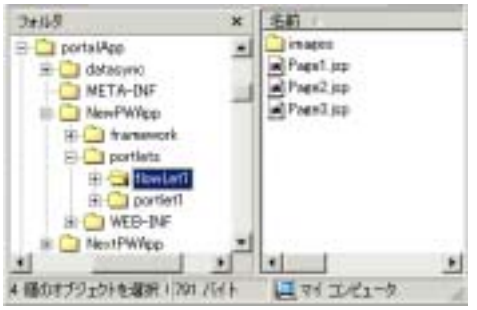
E-Business Control Center 内で flowLet1 ポートレットを開くと、それが利用可能なポートレットのリストに含まれているのがわかります。

元のポートレットではカスタム Webflow が使用されているので、コピー先のポータル Web アプリケーションでこのポートレットを使用するには、その Webflow ファイルをコピーする必要があります。NewPWApp の \webapps 内の flowLet1 Webflow ファイルを NextPWApp の \webapps ディレクトリにコピーする様子を 図 11-3 に示します。

図 11-3　ナビゲーション用 Webflow のコピー

また、ポートレット フォルダ flowLet1 を NewPWApp アプリケーション フォルダから NextPWApp Web アプリケーション フォルダにコピーする様子を 図 11-4 に示します。ポートレットを構成する JSP と画像は、このディレクトリ内に格納されます。

図 11-4　J2EE リソースのコピー



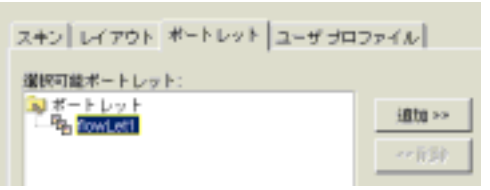
これで、J2EE リソースが新しい Web アプリケーションにコピーされたので、これらのリソースを参照するようにメタデータを編集することができます。

ステップ 2: ターゲット Web アプリケーションのメタデータを編集する

以下の手順に従って、E-Business Control Center を用いて flowLet1 ポートレットを `ThatNewPortal` ポータルに追加します。

1. E-Business Control Center の [プレゼンテーション] タブで、[**ポータル**] をクリックし、ThatNewPortal というポータルを選択します。
2. ポータル エディタが開いたら、右上の [全般] タブをクリックします。
3. [全般] ページの中ほどにある [ポートレット] タブをクリックし、図 11-5 に示すように、[選択可能ポートレット] のリストから flowLet1 ポートレットを選択してから、[**追加**] をクリックします。

図 11-5　ThatNewPortal への flowLet1 ポートレットの追加



4. [全般] タブを閉じ、[ページ] タブをクリックし、Home ポータル ページを選択してから、[**編集**] をクリックします。

5. Home ポータル ページの定義画面の左下にある [選択可能ポートレット] のリストから flowLet1 ポートレットを選択し、[**追加**] をクリックします。[**OK**] をクリックして、このウィンドウを閉じます。
6. このプロジェクトを保存します。

ステップ 3: プロジェクトを同期化する

以下の手順に従って、E-Business Control Center を用いてプロジェクトを同期化します。

注意 : E-Business Control Center の詳細については、『管理者ガイド』の「E-Business Control Center の設定」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/sysadmin.htm) を参照してください。

1. E-Business Control Center のツールバーで、図 11-6 に示すように、[**同期**] ボタンをクリックします。

図 11-6 [同期] ボタンのクリック

2. 同期が完了したことを知らせるメッセージが [アプリケーションの同期中] ウィンドウに表示されたら、[**閉じる**] をクリックします。

ステップ 4: 新しいポートレットを表示対象かつ利用可能にする

これで、flowLet1 という新しいポートレットがデプロイされましたが、さらに、WebLogic Portal Administration Tools を用いて、それをポータルで利用できるようにする必要があります。

1. Web ブラウザで、次の URL にアクセスします。すなわち、http://<hostname>:<port>/portalAppTools です。administrator/password の組み合わせでログインし、図 11-7 に示すように、[**ポータル管理**] をクリックします。

図 11-7　[ポータル管理] へのアクセス

2. [ポータル管理ホーム] ページで、図 11-8 に示すように、NextPWApp の下のデフォルト ポータルをクリックします。

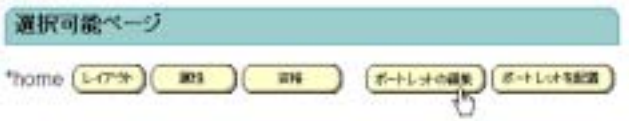
図 11-8　デフォルト グループ ポータルの選択

3. [グループ ポータル管理ホーム] ページで、図 11-9 に示す [**ページとポートレットを管理**] をクリックします。

図 11-9　[ページとポートレットを管理]

4. 図 11-10 に示すように、ポータル ページの行の [**ポートレットの編集**] をクリックします。

図 11-10 [ポートレットの編集] のクリック

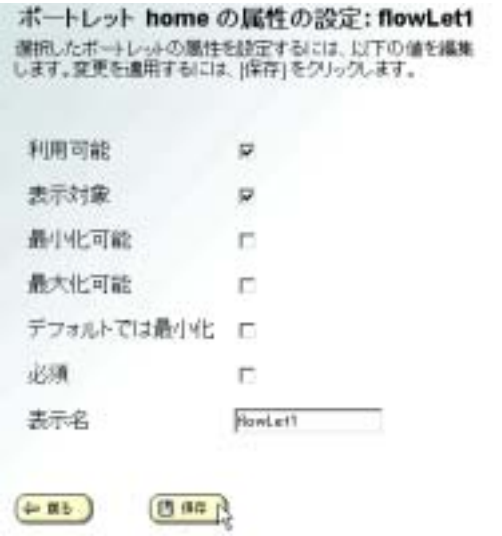


5. 図 11-11 に示すように、リストから flowLet1 ポートレットを選択し、[属性を設定] をクリックします。

図 11-11 flowLet1 ポートレットに対する [属性を設定] の選択

6. 図 11-12 に示すように、このポートレットの属性を [表示対象] および [利用可能] に設定し、[保存] をクリックします。

図 11-12　ポートレット属性の設定



7. `http://<hostname>:<port>/NextPWApp/index.jsp` にアクセスして、結果を確認します。図 11-13 に示すような結果になるはずです。

図 11-13　NextPWApp での flowLet1 の表示

ドメイン間でポートレットをコピーする

両方のドメインが WebLogic Portal の最新リリースのインスタンス上に正しくインストールされデプロイされているものとすると、そのうちの一方からもう一方にポートレットをコピーする手順は、ポータル Web アプリケーション間でポートレットをコピーする場合の手順とほとんど同じです。すなわち、

1. J2EE ファイルをコピーする際には、.portlet ファイルもターゲット サーバの `portalApp-project\portlets` ディレクトリにコピーします。コピー元のドメインの .portlet ファイルが格納されている場所 (`portalApp-project\portlets` ディレクトリ) を 図 11-14 に示します。
2. 「ポータル Web アプリケーション間でポートレットをコピーする」の節で示した手順の残りを完了します。

図 11-14 .portlet ファイル

欠如しているリソースをエラー メッセージを通じて見つける

必要なポートレット リソースがドメイン内になければ、サーバ起動時か、それらのリソースを用いるポータル ページにアクセスしようとしたときに、エラー メッセージが表示されます。(ポートレット リソースが欠如していても、サーバは起動されます)。

たとえば、404 エラーなどの HTTP セッション エラーや、「機能は現在利用できません。」といったレンダリング時のエラーがブラウザに表示される場合があります。メタデータの欠如によっても、エラーが発生することがあります。

サーバのコマンド ウィンドウでエラー メッセージを探して、どのリソースが欠けているのか判断します。たとえば、コード リスト 11-1 では、WebFlowPortlet というポートレットに header.jsp というファイルが足りないことを示しています。このメッセージは、このポートレットが含まれているポータル ページにアクセスしたときにコマンド ウィンドウに出力されたものです。

コード リスト 11-1　ポートレット リソースの欠如によって発生するエラー

```
<Jun 12, 2002 10:38:59 AM MDT> <Error> <HTTP> <101214> <Included
resource or file "/NewWebApp/portlets/WebflowPortlet/header.jsp"
not found from requested resource
"/NewWebApp/framework/portal.jsp".>
```

また、E-Business Control Center では、ポートレットの編集時にエラーがいくつか捕捉されます。たとえば、関連付けられている Webflow ファイルが編集中のポートレットに見つからない場合には、警告ダイアログ ボックスが表示されます。

ポートレットのカテゴリを作成する

E-Business Control Center を利用すれば、ポートレットをカテゴリに分類して、より容易に管理できるようになります。この節では、カテゴリの作成方法とカテゴリ内でのポートレットの管理方法を示します。

カテゴリを扱う準備をする

この節で示す手順に従うには、以下のものをあらかじめ準備しておく必要があります。

- WebLogic Portal 7.0 の作業用インスタンス
- 完全にデプロイされたポータル
- 稼働中の WebLogic Portal Server

警告： E-Business Control Center でポータルを定義する前に新しいカテゴリを作成しようとすると、エラーになります。

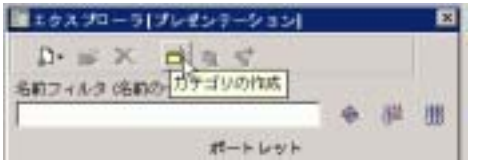
ポートレットとカテゴリを作成する

1. E-Business Control Center の [プレゼンテーション] タブから、第 2 章「新規ドメインへのポータルの新規作成」で示したように、Portlet Wizard を用いて汎用的なポートレットをいくつか作成します。この例に沿って、これらの新しいポートレットに以下のような名前を付けます。

- WallStreet
- MainStreet
- MidWest
- SouthEast
- LatinAmerica

2. E-Business Control Center の [プレゼンテーション] タブで、[**ポートレット**] アイコンをクリックし、図 11-15 に示すように [カテゴリの作成] アイコンをクリックします。

図 11-15　カテゴリの新規作成



3. [新規カテゴリ] 画面が表示されます。[名前] フィールドに「Business」と入力し、[**OK**] をクリックします。このステップを繰り返して、以下の新しいカテゴリを作成します。

- Education
- Regional
- International

その結果、図 11-16 に示すようなカテゴリ リストになるはずです。

図 11-16　新しいカテゴリ

注意： カテゴリの 1 つを選択すると、図 11-17 に示すように、エクスプローラ ツールバーの右側の 3 つのアイコンが使用可能になります。

図 11-17　エクスプローラ ツールバー内のカテゴリ関連アイコン

ポートレットとカテゴリを移動する

この節では、E-Business Control Center を用いてポートレットとカテゴリを移動する方法について説明します。

1. 図 11-18 に示すように、International カテゴリと Regional カテゴリを選択したあと、ツールバーの右端の [**移動**] アイコンをクリックします。

図 11-18　カテゴリの [移動] アイコンのクリック

2. [カテゴリ「International」の移動] 画面が表示されます。Education を選択し [**移動**] をクリックします。[カテゴリ「Regional」の移動] ウィンドウが表示されたら、同じことを行います。その結果、図 11-19 に示すようなカテゴリ構成になるはずです。

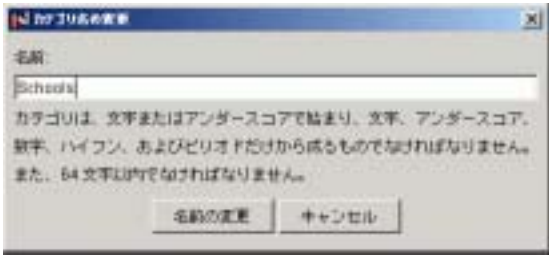
図 11-19　カテゴリとポートレット

3. Education カテゴリを選択したあと、図 11-20 に示すように、ツールバーの右から 2 番目の [**カテゴリ名の変更**] アイコンをクリックします。

図 11-20　カテゴリ選択後の [カテゴリ名の変更] のクリック

4. [カテゴリ名の変更] 画面が表示されます。図 11-21 に示すように「Schools」と入力し、[**名前の変更**] をクリックします。

図 11-21　カテゴリ名の変更



5. MidWest ポートレットと SouthEast ポートレットを Schools カテゴリに移動します。それには、図 11-22 に示すようにそれらのポートレットを選択してから [**移動**] アイコンをクリックし、図 11-23 に示すように移動先を選択します。

図 11-22　カテゴリへのポートレットの移動

図 11-23　移動先フォルダの選択

6. カテゴリ関連アイコンを用いて、MainStreet ポートレットと WallStreet ポートレットを Business カテゴリに、LatinAmerica ポートレットを International カテゴリに、そして、MidWest ポートレットと SouthEast ポートレットを Regional カテゴリに、それぞれ移動します。その結果、図 11-24 のようなカテゴリ構成になるはずです。

図 11-24　整理されたポートレットおよびカテゴリ

既存のカテゴリにポートレットを追加する

この節では、既存のカテゴリに新しいポートレットを追加する方法を説明します。この例では、以下のポートレットを作成します。

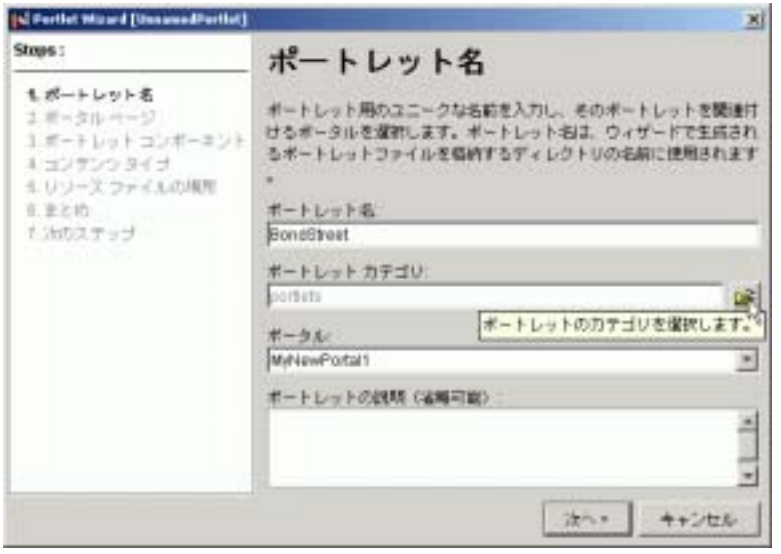
- BondStreet
- UK
- Asia

既存のカテゴリにポートレットを追加するには、以下の手順に従います。

1. E-Business Control Center の [プレゼンテーション] タブで、Portlet Wizard を用いて新しいポートレットを作成します。

注意： ポートレットに名前を付ける際には、図 11-25 に示すように、[**ポートレットのカテゴリを選択します。**] をクリックします。

図 11-25　ポートレット カテゴリの選択



2. 図 11-26 に示すような [ポートレット カテゴリの選択] 画面が表示されます。Business を選択し、[**OK**] をクリックします。

図 11-26　既存のカテゴリ内への新しいポートレットの追加

3. Portlet Wizard で残りの手順を続行し、BondStreet、UK、および Asia の各ポートレットについて同様の操作を繰り返します。
4. その結果、図 11-27 に示すようなポートレット構成になるはずです。

図 11-27　各カテゴリ内のポートレット

ポートレットとフレームワーク

BEA WebLogic プラットフォームでは、ポートレットのカスタマイズを幅広くサポートしており、その中には、ポートレットを構成する実際の JSP 内での最小限の Java スクリプト作成が必要にはなるものの堅牢な機能を提供できるように設計された JSP タグ ライブラリなどがあります。以下のポートレット タイプを作成する際には、この節の説明を参考にしてください。

- **単純 JSP ポートレット :** これらのポートレットにはスクリプトレットを組み込むことができ、パーソナライゼーション機能を呼び出すことができますが、個々の Webflow は使用されません。
- **Webflow ポートレット :** Portal Wizard を用いて、3 種類の異なる Webflow ポートレットを作成できます。Webflow ファイルをカスタマイズし個々のポートレットに関連付けることで、ナビゲーション シナリオやポートレット間シナリオを実現することもできます。
- **Web サービス ポートレット :** 他のプログラムとのさまざまなやり取りを、ローカルに、またはインターネット経由で呼び出すコードをポートレットに追加することができます。

この節ではこのあと、各ポートレット タイプについて説明し、それぞれの例をいくつか示すとともに、BEA WebLogic プラットフォームを最大限に活用するポートレットを作成するテクニックをいくつか紹介します。

単純 JSP ポートレット

WebLogic Portal プラットフォームに備わっている能力、柔軟性、および使いやすさを一部紹介するために、この節では、以下の例を扱います。

- scriptDemo ポートレット
- ポートレットから ActiveX コンポーネントを呼び出す

scriptDemo ポートレット

scriptDemo ポートレット (図 11-30 に示す) は、Portal Wizard を用いて作成されたサンプル ポータルの Home ページの左側に表示されます。

準備

これから紹介する例を実際に試してみるには、以下の準備が必要です。

1. 第 2 章「新規ドメインへのポータルの新規作成」の「ステップ 2: ポータルの新規作成」での説明に従って、「MyNewPortal1」という新規ポータルと「OldPortalWebApp」という新規ポータル Web アプリケーションをそれぞれ作成します。
2. 第 2 章「新規ドメインへのポータルの新規作成」の「ステップ 3: ポートレットの追加」での説明に従って、「scriptDemo」という新規ポートレットを作成します。
3. 第 2 章「新規ドメインへのポータルの新規作成」の「ステップ 4: 新規ポートレットの可視化」での説明に従って、「scriptDemo」ポートレットを表示対象にします。

続行する前に

ポータルは、ユーザがログインする前は図 11-28 のように、ユーザがログインしたあとは図 11-29 のように、それぞれ表示されることを確かめてください。

図 11-28　匿名ユーザ時の初期画面

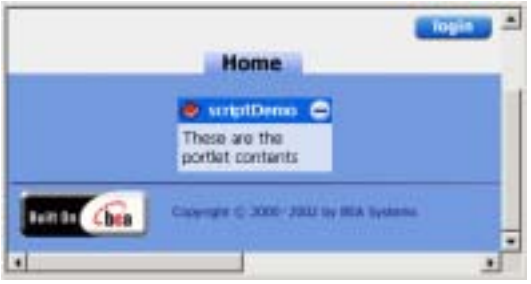


図 11-29　ユーザ ログイン後の初期画面

scriptDemo ポートレットのインストール

以上で素材が揃ったので、`portlet.jsp` を入れ替えて動作を変更することで、ポートレットを変えることができます。コード リスト 11-2 に非常に簡単なスクリプトレットを示しますが、これは、汎用的な Java スクリプトレットを用いてポートレット内でポータル フレームワークとやり取りする方法を示すものです。

コード リスト 11-2　scriptDemo ポートレット (バージョン 1)

```
<center> Test Portlet </center>
<%
<br><hr>
A Simple Java Scriptlet
<br>
/*
新しい String 変数を作成し、その値を空の文字列に設定する。
*/
  String userName = "";
/*

実際のユーザ名を取得するには、まず getUserPrincipal() メソッドを用いて
javax.servlet.http.HttpServletRequest オブジェクトから
java.security.Principal オブジェクトを取得する。 このリクエスト オブジェクトは JSP から直接利用できる。Principal オブジェクトについては、完全パッケージ名を使用すること。

*/

  java.security.Principal principal = request.getUserPrincipal();
/*

プリンシパル オブジェクトが null の場合には、ユーザはまだログインしていない。この例では、ログインしていない場合を無視する。それには、if 文を用いて、値が null でない場合にのみ値の処理を行うようにする。
```

```
*/
  if (principal != null)
  {
/*
*/
    userName = principal.getName();
  }
%>
<%--
```

スクリプトレットを用いて、userName 変数の値を表示する。スクリプトレットは HTML 内に埋め込まれ、<%= %> ブロックで示されることに注意。

```
--%>
The user name is : <%= userName %>
<br>
```

手順

この新しいコードを scriptDemo ポートレットに組み込むには、以下の手順に従います。

1. テキスト エディタで、コード リスト 11-2 に示すコードを portlet.jsp という名前で以下のディレクトリに保存します。

   ```
   <BEA_HOME>\portalDomain\beaApps\portalApp\
   <PortalWebApp>\portlets\scriptDemo\
   ```

2. ブラウザの表示を更新し、図 11-30 のような結果になることを確かめます。ポートレットのコンテンツが変更されたため、ユーザがログアウトしていることに注目してください。ポートレットは今度は匿名ユーザの状態で表示されるので、ユーザ名がポートレットに表示されないことがわかります。

図 11-30　匿名ユーザ時の scriptDemo ポートレット

3. ポータルの右上隅の [**login**] をクリックします。
4. [ポータル ログイン] ページが表示されたら、右側の [新規ユーザ] 列の下にある [**今すぐサインアップする**] をクリックします。
5. [新規ユーザの登録] ページが表示されたら、「bobjones」というユーザを新たに作成し、[パスワード] フィールドに「password」と入力したあと、[**送信**] をクリックします。

ユーザがログインすると scriptDemo ポートレット内のコードによってポータルがどのように表示されるかを、図 11-31 に示します。ユーザ名が表示されていることに注目してください。

図 11-31 ユーザ ログイン後の scriptDemo ポートレット

注意 : カスタム ポートレットでの JSP タグの使い方について詳しくは、以下の情報源を参照してください。

- JavaServer Pages ガイド
- WebLogic Portal *Javadoc* API マニュアル

ポートレットから ActiveX コンポーネントを呼び出す

ポートレット内から ActiveX コンポーネントを呼び出すには、コード リスト 11-3 および コード リスト 11-4 に示すように、HTML の <OBJECT> タグを使用します。

これらのサンプルには CODEBASE パラメータが含まれているため、ローカル マシンのレジストリにコンポーネントがインストールされていない場合には、コンポーネントを Microsoft からダウンロードすることができます。

注意： これらのサンプルでは、ポータルは HTML を介して ActiveX コンポーネントと通信するだけです。HTML はポータルからブラウザに返され、ActiveX コンポーネントをロードして実行するようブラウザに命令します。

コード リスト 11-3　ActiveX を用いた Outlook の呼び出し (予定表の例)

```
<OBJECT CLASSID="clsid:0006F063-0000-0000-C000-000000000046"
CODEBASE="http://activex.microsoft.com/activex/controls/office/
outlctlx.CAB#ver=9,0,0,3203"
    id=OVCtl1 width=100% height=200>
  <param name="Folder" value="Calendar">
  <param name="Namespace" value="MAPI">
  <param name="Restriction" value="">
  <param name="DeferUpdate" value="0">
</OBJECT>
```

コード リスト 11-4　ActiveX を用いた Outlook の呼び出し (受信トレイの例)

```
<OBJECT CLASSID="clsid:0006F063-0000-0000-C000-000000000046"
CODEBASE="http://activex.microsoft.com/activex/controls/office/ou
tlctlx.CAB#
ver=9,0,0,3203"
        id=OVCtl1 width=100% height=200>
  <param name="Folder" value="Inbox">
  <param name="Namespace" value="MAPI">
  <param name="Restriction" value="">
  <param name="DeferUpdate" value="0">
</OBJECT>
```

注意： ActiveX コンポーネントは、その実行が許可される適切なセキュリティ レベルに設定された Microsoft Internet Explorer 内でのみ動作します。

Webflow ポートレット

この節では、まず、Portlet Wizard で作成できる Webflow ポートレットのタイプを示します。次に、簡単なスクリプト作成の例を用いて、Webflow の基本的側面をいくつか説明します。この節で扱うトピックは以下のとおりです。

- 3 種類の Webflow ポートレット
- ポートレットでの表示更新イベントの処理方法
- ポートレットをカスタム イベントに応答させる
- ポートレット間で状態を共有する

3 種類の Webflow ポートレット

E-Business Control Center の Portlet Wizard では、現在、以下の 3 種類の Webflow ポートレットを生成することができます。

- ナビゲーション バー : セット内の各ページにはナビゲーションバーがあり、そこにはセット内の他の全ページへのリンクが付いています。
- 順次型 : [次へ] ボタンと [前へ] ボタンを使って、ページを順に移動できます。
- 階層型 : 親ページを作成し、それぞれの子ページへのリンクを持たせます。子ページには親ページに戻るリンクがありますが、子ページ間のリンクはありません。

準備

この節では、この章の随所で例として示しているように、アプリケーション `NewPWApp` に新しい Webflow ポートレットを追加する方法を説明します。これらの例を実際に試してみるには、ポータル サーバが稼働しており、E-Business Control Center でポータル プロジェクトが開かれていなければなりません。.

ナビゲーション バー Webflow ポートレットの作成

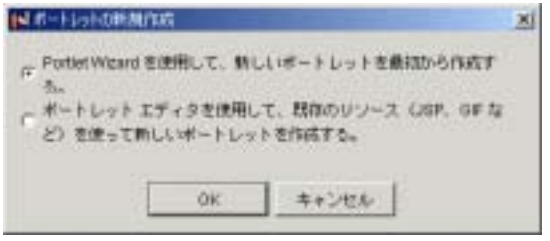
Portlet Wizard を用いてナビゲーション バー Webflow ポートレットを作成する手順を以下に示します。

1. E-Business Control Center の [プレゼンテーション] タブで、図 11-32 に示すように、[新規作成 | ポートレット] を選択します。

図 11-32　Portlet Wizard の起動

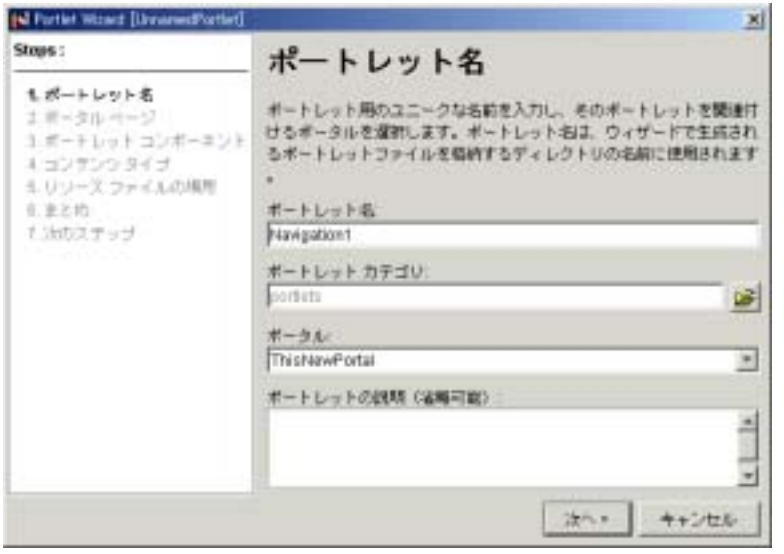
2. 図 11-33 に示すように、Portlet Wizard を使用することにし、[**OK**] をクリックします。

図 11-33　Portlet Wizard の使用を選択



3. 図 11-34 に示すように、このポートレットの名前を「Navigation1」にし、[**次へ**] をクリックします。

図 11-34　ナビゲーション ポートレットの命名



4. 図 11-35 に示すように、この新規ポートレットを Home ページに関連付け、[**次へ**] をクリックします。

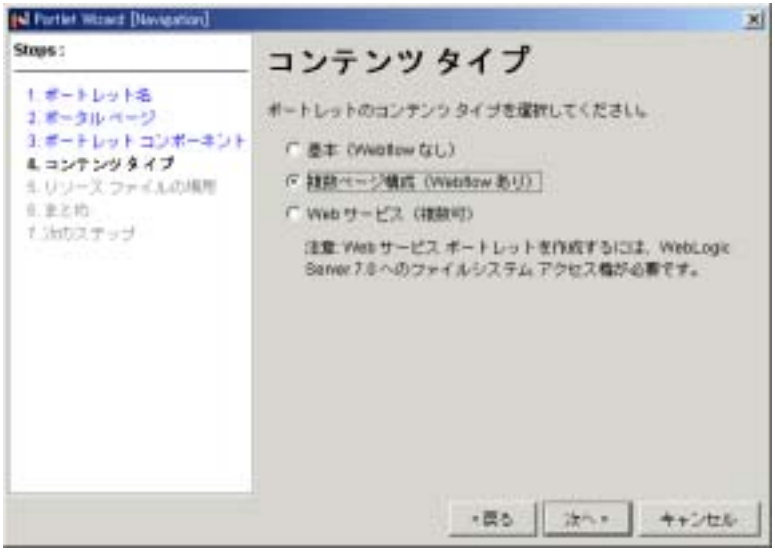
図 11-35　Home ページへのポートレットの関連付け

5. 図 11-36 に示すように、ポートレット コンポーネントを選択し、[**次へ**] をクリックします。

図 11-36　コンポーネントの選択

6. 図 11-37 に示すように、Webflow コンテンツ タイプを選択し、[**次へ**] をクリックします。

図 11-37　Webflow コンテンツ タイプの選択

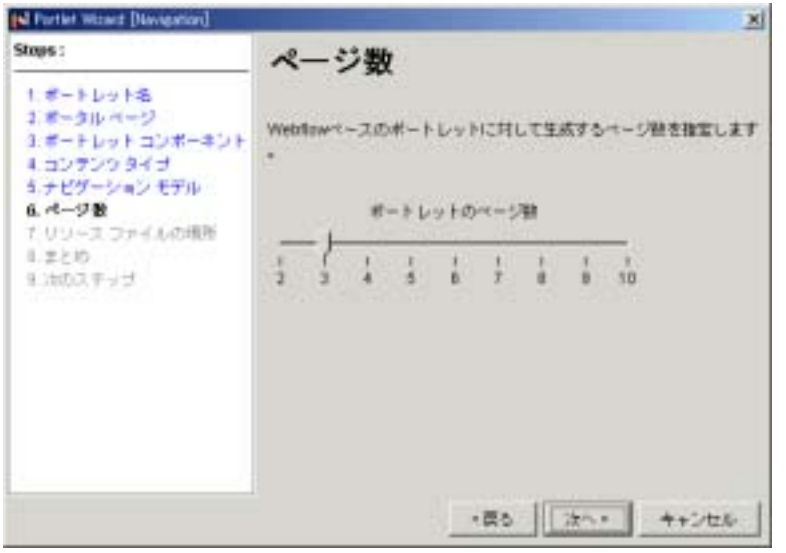


7. 図 11-38 に示すように、「ナビゲーション バー」モデルを選択し、[**次へ**] をクリックします。

図 11-38　「ナビゲーション バー」モデルの選択

8. 図 11-39 に示すように、ページ数を選択し、[**次へ**] をクリックします。

図 11-39　ページ数の選択

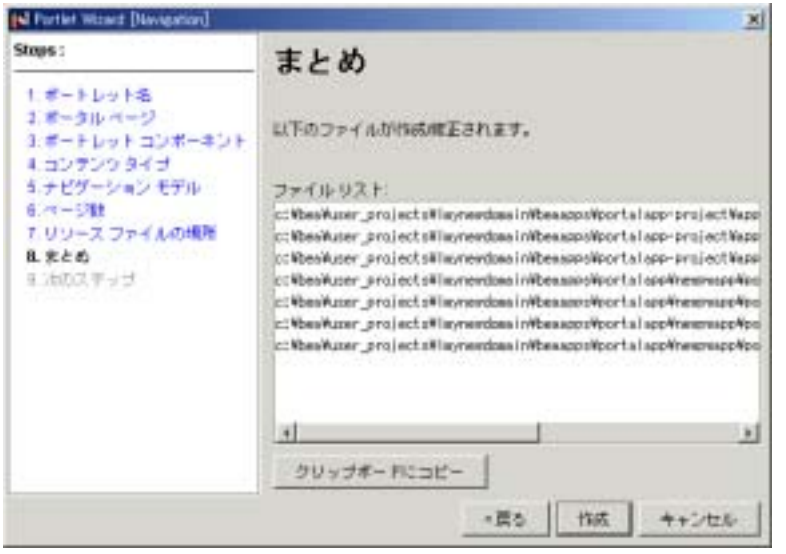


9. 図 11-40 に示すように、リソース ファイルの場所を確認し、[**次へ**] をクリックします。

図 11-40　リソース ファイルの格納場所の確認

10. 図 11-41 に示すように、作成対象ファイルのリストを確認し、[**作成**] をクリックします。

図 11-41　ファイル リストの確認



11. 図 11-42 に示すように、次のステップを確認し、[**閉じる**] をクリックします。

図 11-42　次のステップの確認

12. 図 11-43 に示すように、プロジェクトを同期化します。

図 11-43　プロジェクトの同期化

13. 11-5 ページの「ステップ 4: 新しいポートレットを表示対象かつ利用可能にする」の節で示したように、WebLogic Portal Administration Tools を用いて、このポートレットを表示対象かつ利用可能に設定します。
14. 図 11-44 に示すように、このポートレットを表示します。

図 11-44　ナビゲーション バー ポートレットの表示

順次型 Webflow ポートレットの作成

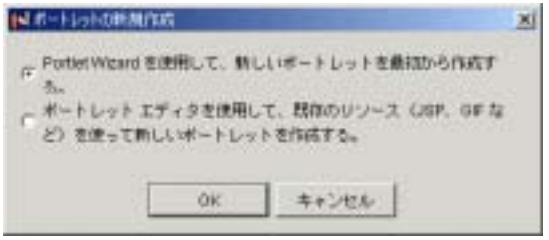
Portlet Wizard を用いて順次型 Webflow ポートレットを作成する手順を以下に示します。

1. E-Business Control Center の [プレゼンテーション] タブで、図 11-45 に示すように、[新規作成｜ポートレット] を選択します。

図 11-45　Portlet Wizard の起動

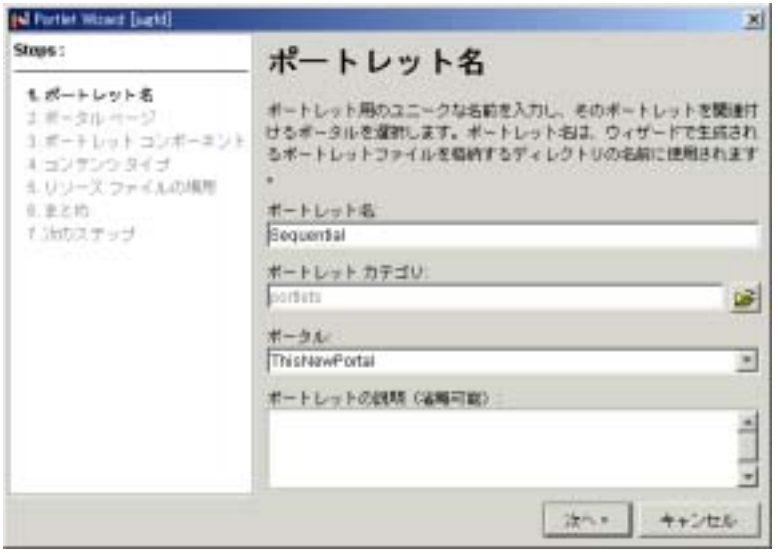
2. 図 11-46 に示すように、Portlet Wizard を使用することにし、[**OK**] をクリックします。

図 11-46　Portlet Wizard の使用を選択



3. 図 11-47 に示すように、このポートレットの名前を「Sequential」にしたあと、[**次へ**] をクリックします。

図 11-47　順次型 Webflow ポートレットの命名



4. 図 11-48 に示すように、ポートレットを Home ページに関連付け、[**次へ**] をクリックします。

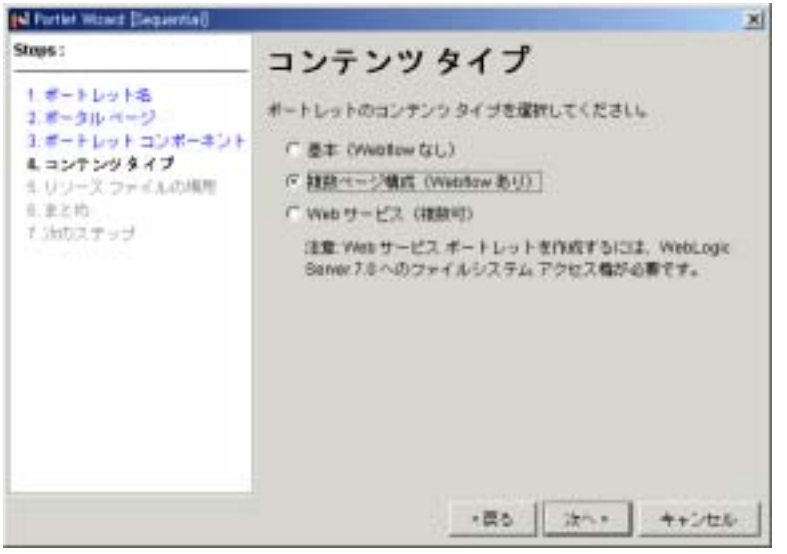
図 11-48　Home ページへのポートレットの関連付け

5. 図 11-49 に示すように、ポートレット コンポーネントを選択し、[**次へ**] をクリックします。

図 11-49　ポートレット コンポーネントの選択

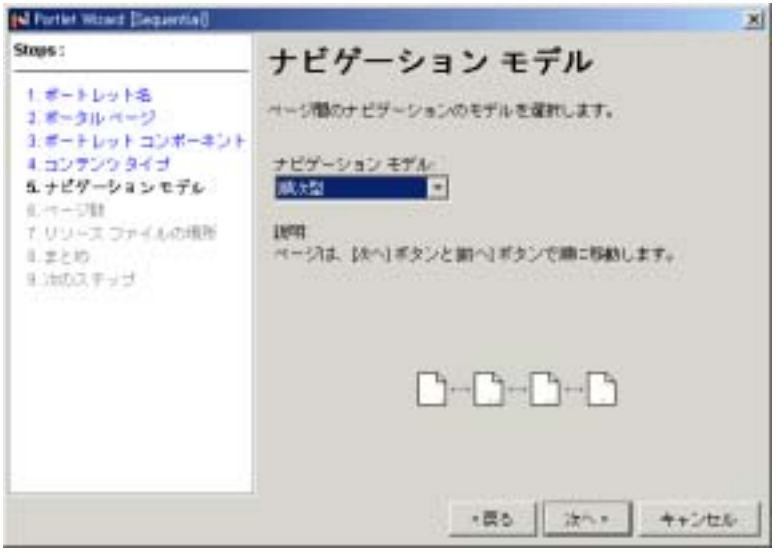
6. 図 11-50 に示すように、Webflow コンテンツ タイプを選択し、[**次へ**] をクリックします。

図 11-50　Webflow コンテンツ タイプの選択



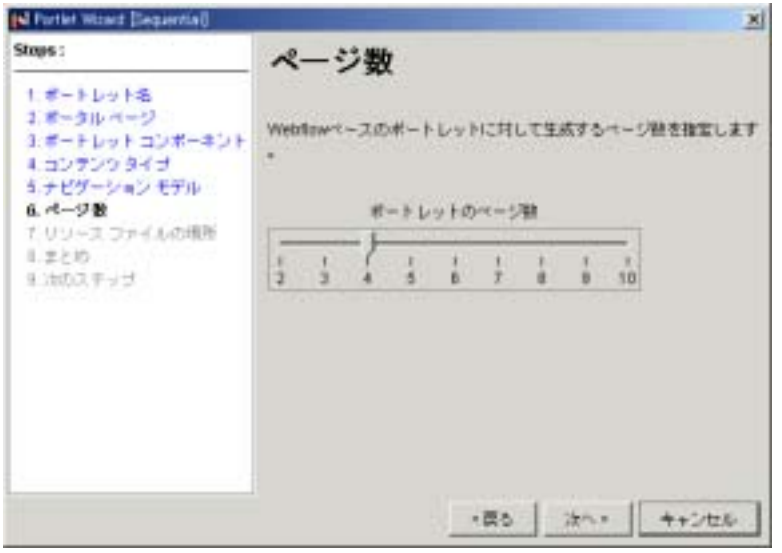
7. 図 11-51 に示すように、「順次型」モデルを選択し、[**次へ**] をクリックします。

図 11-51　順次型モデルの選択



8. 図 11-52 に示すように、ページ数を選択し、[**次へ**] をクリックします。

図 11-52　ページ数の選択

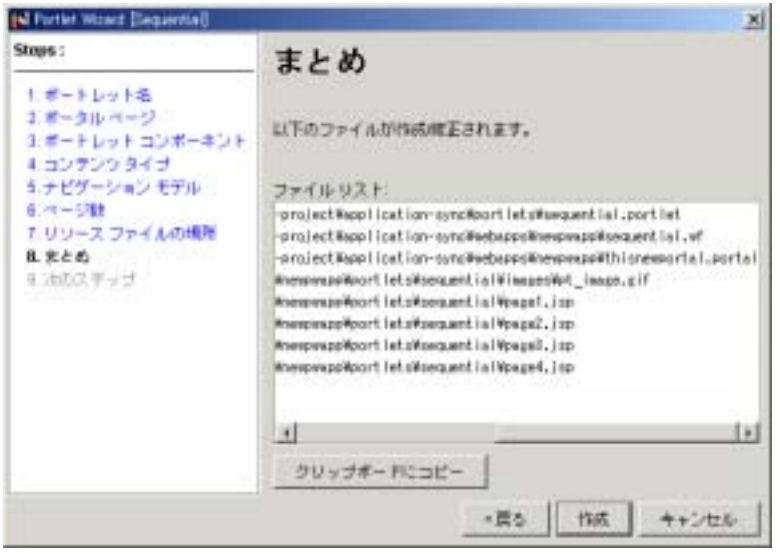


9. 図 11-53 に示すように、リソース ファイルの場所を確認し、[**次へ**] をクリックします。

図 11-53　リソース ファイルの格納場所の確認

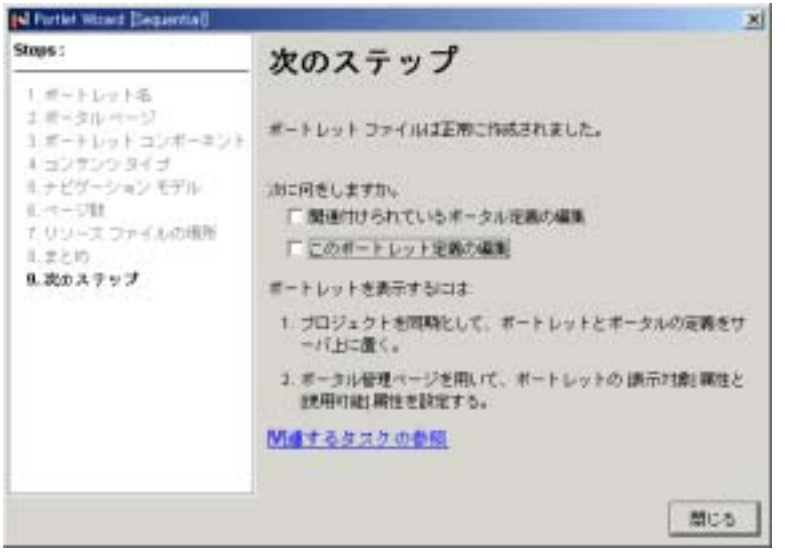
10. 図 11-54 に示すように、作成対象ファイルのリストを確認し、[**作成**] をクリックします。

図 11-54　ファイル リストの確認



11. 図 11-55 に示すように、次のステップを確認し、[**閉じる**] をクリックします。

図 11-55　次のステップの確認



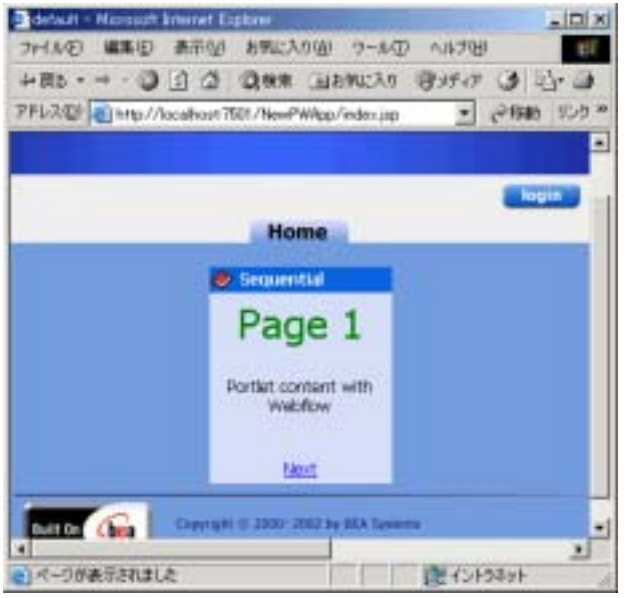
12. 図 11-56 に示すように、プロジェクトを同期化します。

図 11-56　プロジェクトの同期化

13. 11-5 ページの「ステップ 4: 新しいポートレットを表示対象かつ利用可能にする」の節で示したように、WebLogic Portal Administration Tools を用いて、このポートレットを表示対象かつ利用可能に設定します。

14. 図 11-57 に示すように、このポートレットを表示します。

図 11-57　順次型 Webflow ポートレットの表示



階層型 Webflow ポートレットの作成

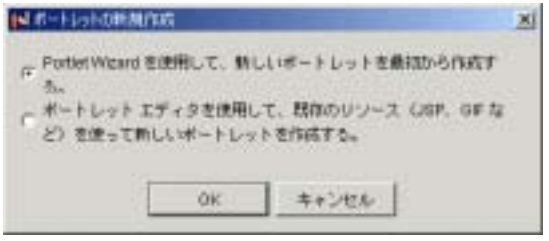
Portlet Wizard を用いて階層型 Webflow ポートレットを作成する手順を以下に示します。

1. E-Business Control Center の [プレゼンテーション] タブで、図 11-58 に示すように、[新規作成 | ポートレット] を選択します。

図 11-58　Portlet Wizard の起動

2. 図 11-59 に示すように、Portlet Wizard を使用することにし、[**OK**] をクリックします。

図 11-59　Portlet Wizard の使用を選択



3. 図 11-60 に示すように、このポートレットの名前を「Hierarchical」にしたあと、[**次へ**] をクリックします。

図 11-60　階層型ポートレットの命名

4. 図 11-61 に示すように、この新規ポートレットを Home ページに関連付け、[**次へ**] をクリックします。

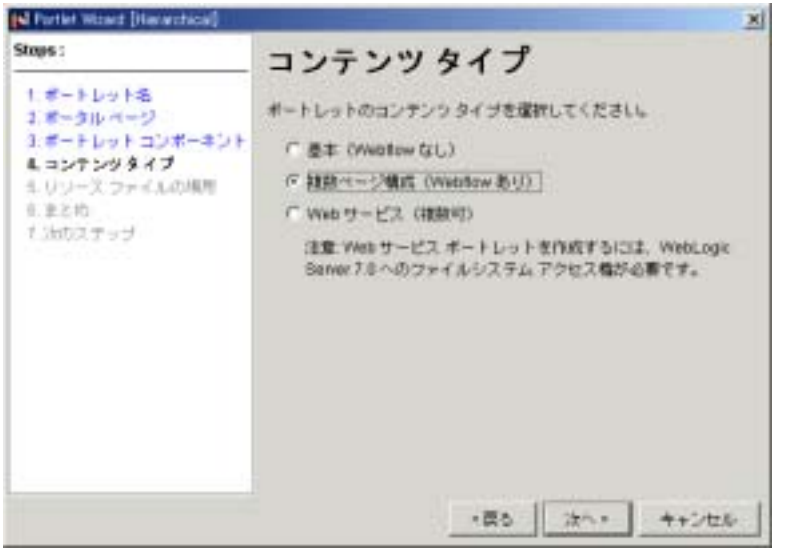
図 11-61　Home ページへのポートレットの関連付け

5. 図 11-62 に示すように、ポートレット コンポーネントを選択し、[**次へ**] をクリックします。

図 11-62　コンポーネントの選択

6. 図 11-63 に示すように、Webflow コンテンツ タイプを選択し、[**次へ**] をクリックします。

図 11-63　Webflow コンテンツ タイプの選択

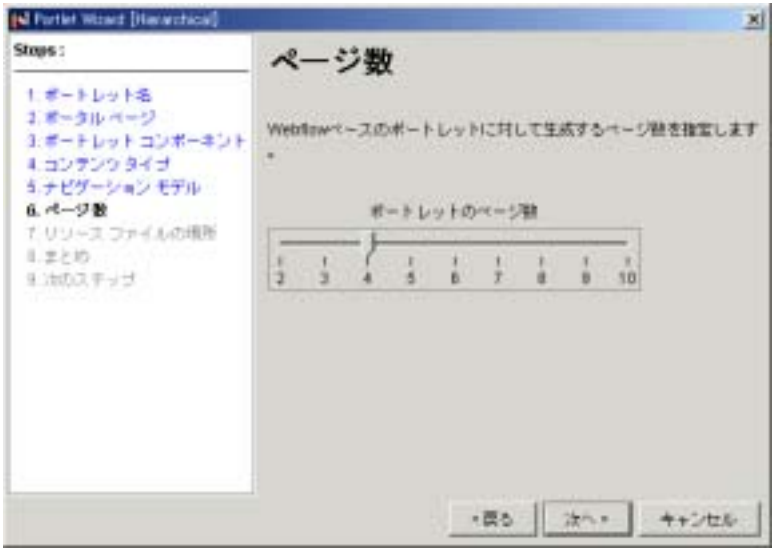


7. 図 11-64 に示すように、「階層型」モデルを選択し、[**次へ**] をクリックします。

図 11-64　階層型モデルの選択

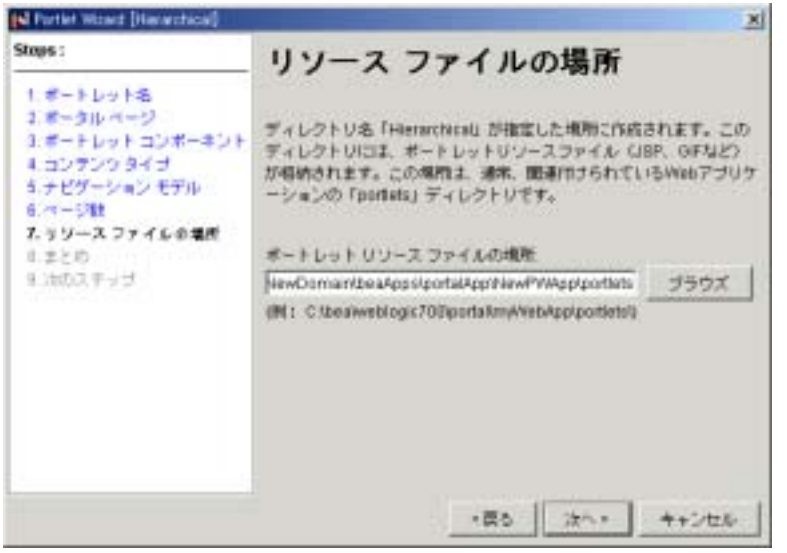
8. 図 11-65 に示すように、ページ数を選択し、[**次へ**] をクリックします。

図 11-65　ページ数の選択



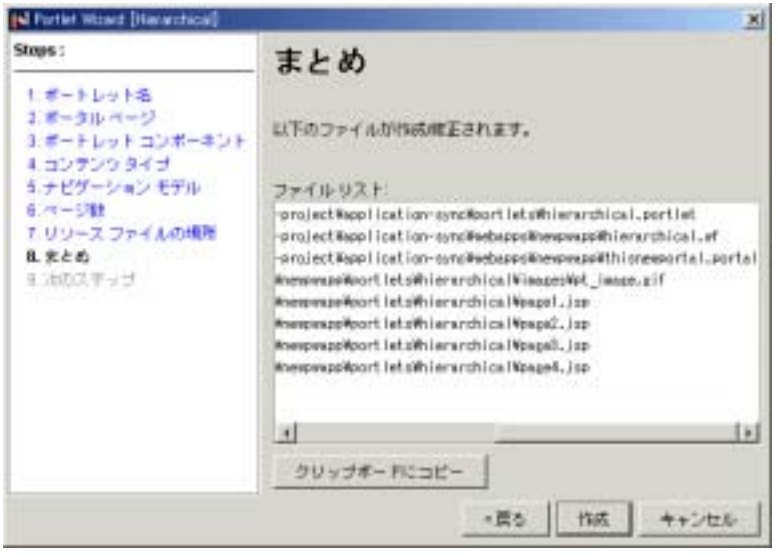
9. 図 11-66 に示すように、リソース ファイルの場所を確認し、[**次へ**] をクリックします。

図 11-66　リソース ファイルの格納場所の確認



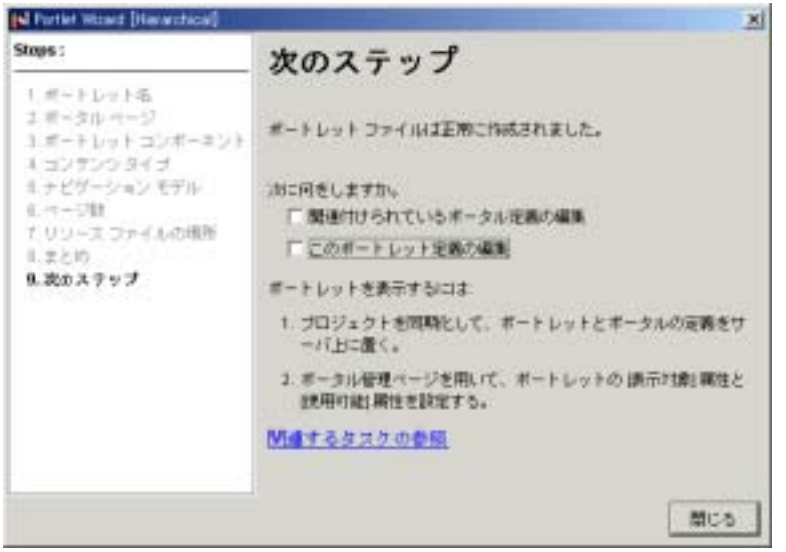
10. 図 11-67 に示すように、作成対象ファイルのリストを確認し、[**作成**] をクリックします。

図 11-67　ファイル リストの確認



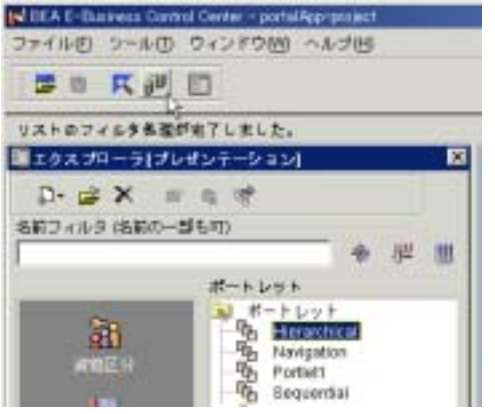
11. 図 11-68 に示すように、次のステップを確認し、[**閉じる**] をクリックします。

図 11-68　次のステップの確認



12. 図 11-69 に示すように、プロジェクトを同期化します。

図 11-69　プロジェクトの同期化



13. 11-5 ページの「ステップ 4: 新しいポートレットを表示対象かつ利用可能にする」の節で示したように、WebLogic Portal Administration Tools を用いて、このポートレットを表示対象かつ利用可能に設定します。

14. 図 11-70 に示すように、このポートレットを表示します。

図 11-70　階層型ポートレットの表示

ポートレットでの表示更新イベントの処理方法

ポートレットでの Webflow の使い方を示すために、コード リスト 11-5 内の太字で示したコードを例に、ポートレットで表示更新イベントがどう処理されるかを観察しましょう。

コード リスト 11-5　ポートレットへの表示更新通知の追加

```
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%
System.out.println("Calling refresh on flowLet1 portlet.");
%>
<center>
<font size="6" color="green">Portlet 1</font><BR>
<p>
Portlet content with Webflow
<p>
<a href="<portlet:createWebflowURL event="switch1"/>">Next
```

```
Page</a>
<p>
</center>
```

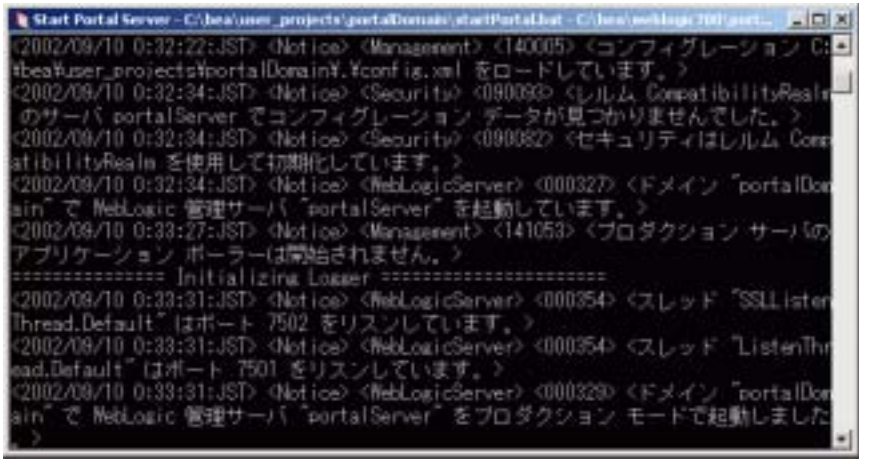
上記の例の続きとして、以下の手順に示すように、scriptDemo ポートレットのコードを編集します。

1. 図 11-71 に示すように、Windows のタスクバーからコンソールを選択して開きます。

図 11-71 コンソール ウィンドウのオープン

コンソールの表示は、図 11-72 のようになるはずです。

図 11-72 コンソール ウィンドウ



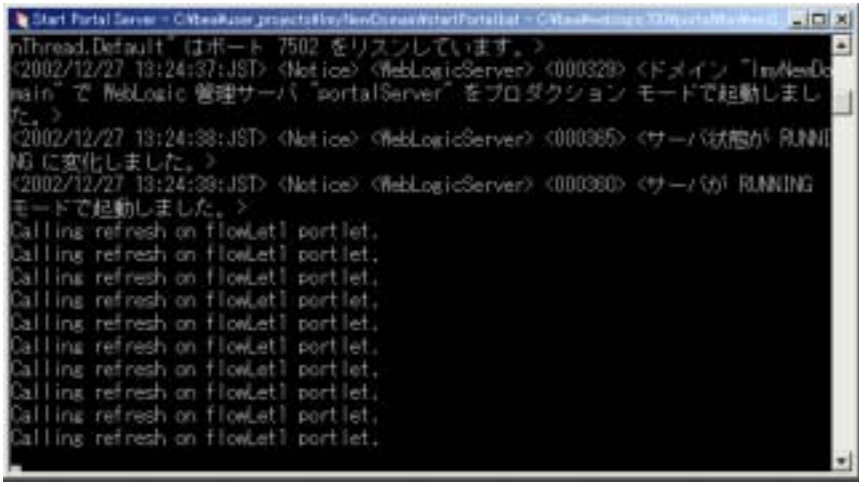
2. テキスト エディタで、コード リスト 11-2 に示すコードを `portlet.jsp` という名前で以下のディレクトリに保存します。

   ```
   <BEA_HOME>\portalDomain\beaApps\portalApp\<PortalWebApp>\
   portlets\scriptDemo\
   ```

3. ブラウザの表示を更新し、ポートレットに変化がないことを確かめます。各ポートレット内の Webflow をクリックし、ポートレット内のページ間を何度か行ったり来たりしてみます。

4. コンソールに戻り、flowLet1 からの出力 (図 11-73 に示す) を確認します。

図 11-73　コンソールに出力された表示更新メッセージ



ポートレットでの Webflow 表示更新イベントの概要

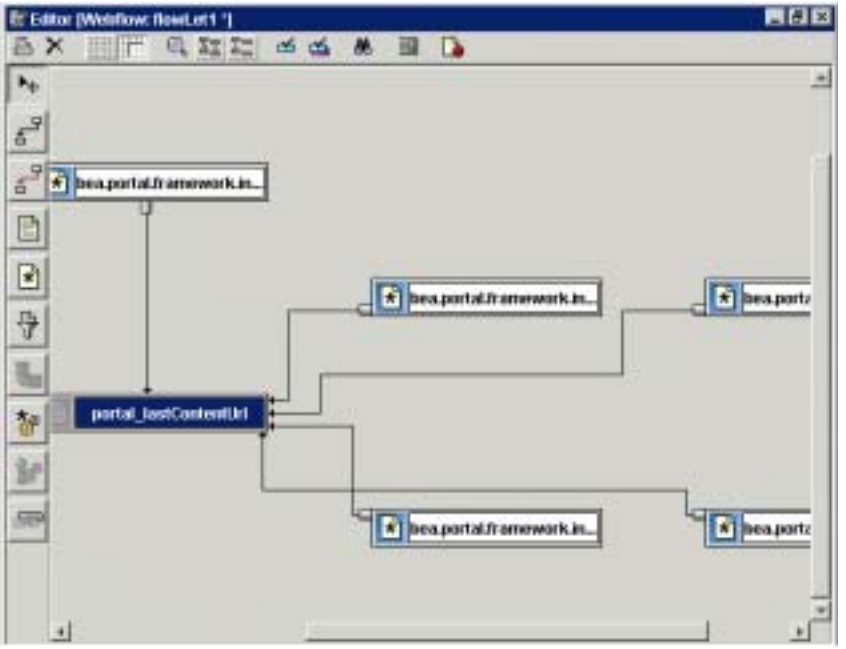
Webflow では、表示更新イベントが発生すると、結果的に lastContentUrl というエンティティが呼び出されることになり、これによって、ポートレット名を指定せずにポートレットに表示を更新させることができるようになります。この例のポータル内のさまざまなボタンやコントロールを操作しつつ、コンソール ウィンドウで表示更新メッセージを確認することで、特定のアクションによって表示更新イベントが発生するタイミングを調べることができます。flowLet1 に関連付けられている Webflow をもう少し詳しく調べるには、以下の手順に従います。

1. Webflow を調べるには、まず、E-Business Control Center の [サイト インフラストラクチャ] タブで、[**Webflow/Pipeline**] を選択します。
2. 図 11-74 に示すように、**NewPWApp1** をダブルクリックして開きます。

図 11-74　ポータル Web アプリケーション内の Webflow のオープン

3. **flowLet1** という Webflow をダブルクリックして開き、エディタ ウィンドウに表示されるダイアグラムを調べます。ナビゲーション マッピングの右側で、図 11-75 に示すように、最小化、最大化、最小化解除、および最大化解除を表す各ノードが `portal_lastContentUrl` にリンクされていることに注目してください。

図 11-75　portal_lastContentUrl にリンクされているノード



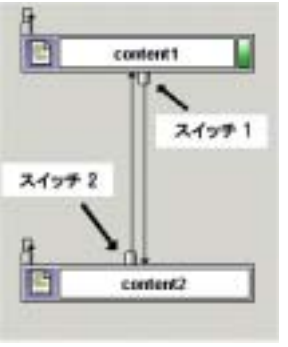
デフォルトでは、表示更新は常にポータル全体で呼び出され、すべてのポートレットに対して呼び出すことができます。専用の Webflow を備えたポートレットであれば、どのようなものでも、すべての表示更新イベントに応答させることができます。このデフォルト動作をポートレットに組み込むには、Webflow エディタを用いて、ポートレットの Webflow を開き、*.refresh プロキシ ノードを `lastContentUrl` に設定します。

Webflow の使い方の詳細については、第 9 章「ポータル ナビゲーションのセットアップ」を参照してください。

ポートレットをカスタム イベントに応答させる

カスタム イベントの説明のために、flowLet1 ポートレットの Webflow におけるナビゲーション マッピング (図 11-76 に示す) について考えてみましょう。

図 11-76　flowLet1 の Webflow におけるナビゲーション ノード



`content1.jsp` と `content2.jsp` の間のナビゲーションは、`switch1` および `switch2` という定義済みイベントを用いて実現されます。

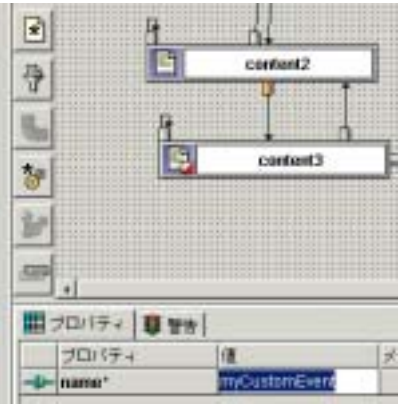
これらのコンテンツ ノードの一方を選択することで、flowLet1 ポートレットで content1.jsp と content2.jsp をリンクする Webflow のプロパティを調べることができます。プレゼンテーション ノード content2 のプロパティの詳細は、図 11-77 に示すとおりです。

図 11-77　content2 のマッピング

カスタム イベントの定義

新しいプレゼンテーション ノードにカスタム イベントを割り当てる方法を図 11-78 に示します。このイベントの名前は `myCustomEvent` であり、イベント コネクタ ツールを用いて、プレゼンテーション ノード `content2` とプレゼンテーション ノード `content3` をリンクします。

図 11-78　myCustomEvent の定義



カスタム イベントを呼び出す

ポートレット内でカスタム イベントを呼び出すには、JSP に以下のコードを追加します。

```
<a href="<portlet:createWebflowURL event="myCustomEvent"/>">Next
Page</a>
```

content3.jsp という新しいプレゼンテーション ノードを追加したあとの flowLet1 の 3 番目の画面を図 11-79 に示します。

図 11-79　flowLet1 ポートレットの 3 番目のページ

ポートレット間で状態を共有する

ポートレット間で状態を共有するには、入力プロセッサ (IP) を用いて HttpSession オブジェクト内に引数を設定します。たとえば、`portletA` で「foo」というフォームを用いる場合には、IP を用いて「foo」フォームのデータを抽出し、`portletB` では Pipeline セッションからそのフォーム データを取得することができます。

注意： 入力プロセッサの詳細については、第 9 章「ポータル ナビゲーションのセットアップ」を参照してください。

Web サービス ポートレット

Web サービスは、アプリケーション間の効率的な疎結合型の相互作用を実現する再利用可能なソフトウェア コンポーネントで、迅速かつ低コストのアプリケーション統合のために内部的に用いられることもあれば、インターネットを介して顧客、サプライヤ、あるいはパートナに公開されることもあります。内部的に、あるいはインターネットを介して、ポートレットで Web サービスを利用できるようになれば、さまざまなメリットが新たに生まれると同時に、開発上の新たな問題点もいくつか出てきます。

注意： 簡単のために、この節では、ローカル ホスト上の Web サービスに接続するポートレットの例を示します。

この節では、以下のトピックを扱います。

- Portlet Wizard を用いて Web サービス ポートレットを作成する
- 単純フォーム駆動型 Web サービス ポートレットを作成する
- 呼び出し生成型 Web サービス ポートレットを作成する
- Web サービス インタフェース型ポートレットを作成する
- Web サービス ポートレットをデプロイする
- Web サービス ポートレット内でのエラー処理
- Web サービスを非同期的に呼び出す

Portlet Wizard を用いて Web サービス ポートレットを作成する

Web サービスを利用するポートレットを Portlet Wizard で作成するには、以下の 3 通りの方法があります。

- **単純 Web サービス フォーム：** これは Web サービスとのやり取りのうち最も単純なモードで、そこでは、httpSession を用いて単純なパラメータが Web サービス クラスに送られ、生成される HTML にプリミティブな値が出力されます。
- **呼び出し生成：** もう少し複雑なのは呼び出し生成タイプのポートレットで、そこでは、Web サービスから提供されるクラスは WSDL (Web Services Definition Language) というオブジェクト内に列挙されます。ポートレット側では、それらのクラスへのリモート呼び出しを行い、返されるオブジェクトを使用します。Portlet Wizard では、[呼び出し生成] オプションを選ぶと、指定された Web サービスを呼び出すスタブ付きポートレットが生成されます。ユーザ プロファイル、リクエスト、セッションなどのソースから得られる出力を用いて、__REPLACE_ME__ 変数を設定する必要があります。
- **Web サービス インタフェース：** このオプションを用いると、複数の Web サービスを選んだり、インタフェース ドキュメントをポートレット自身に抽

出することができます。パラメータや戻り値に複合データ型が必要な場合には、このオプションを使用します。

このオプションは、クライアント ポーリングを通じて非同期的に Web サービスを呼び出すポートレットを作成するのに用いることができます。このタイプのポートレットの作成方法については、「Web サービスを非同期的に呼び出す」の節で説明します。

この節では、Portlet Wizard を用いて各タイプのポートレットを作成する方法を示します。

準備

[プログラム | BEA WebLogic Platform 7.0 | WebLogic Workshop | WebLogic Workshop Examples | Start Examples Server] を選択して、WebLogic Workshop のサンプル サーバを起動します。「scriptDemo ポートレット」の節で説明したサンプル ポータルを基に作業を始めます。

注意： 以下の例では、scriptDemo、flowLet1、および flowLet2 の各ポートレットは省略されていますが、以前作成したこれらのポートレットをポータルから削除せずに、これらの Web サービス ポートレット サンプルを完成させます。

ネームスペースの衝突を避ける

上記の呼び出し生成とインタフェースの例ではどちらも、同一クラスのインスタンスを作成し、それらは同じポータル ネームスペース内で動作します。ネームスペースの衝突を避けるために、各クラスのローカル インスタンスにはユニークな名前を付ける必要があります。2 番目と 3 番目のポートレットでは、サンプル コードの前に示すように実装クラス名を変更します。

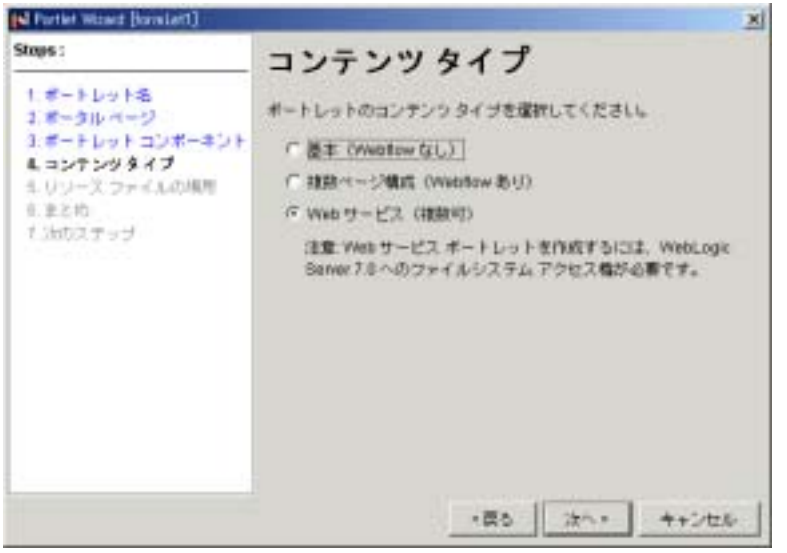
単純フォーム駆動型 Web サービス ポートレットを作成する

最初のポートレットでは、単純なフォーム入力を `AccountEJBClient` Web サービスに送信し、WebLogic Workshop のサンプル サーバをホストとする単純な銀行口座オブジェクトを作成します。

Portlet Wizard を用いて formLet というポートレットを作成する

1. 図 11-32 から 図 11-36 で示した手順に従って操作します。
2. [コンテンツ タイプ] 画面が表示されたら、図 11-80 に示すように、[**Web サービス**] を選択し、[**次へ**] をクリックします。

図 11-80　Web サービス コード タイプの選択

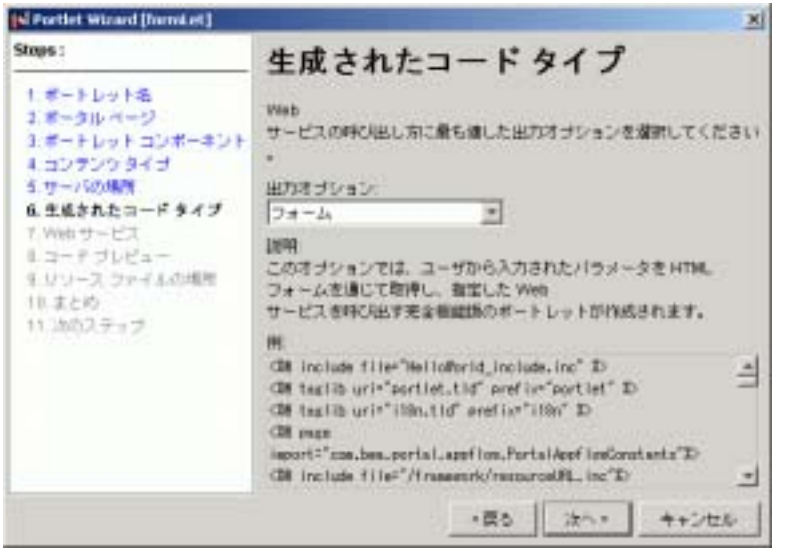


3. [サーバの場所] 画面が表示されたら、図 11-81 に示すように、WebLogic Server のインスタンスの場所を指定し、[**次へ**] をクリックします。

図 11-81　サーバの場所の選択

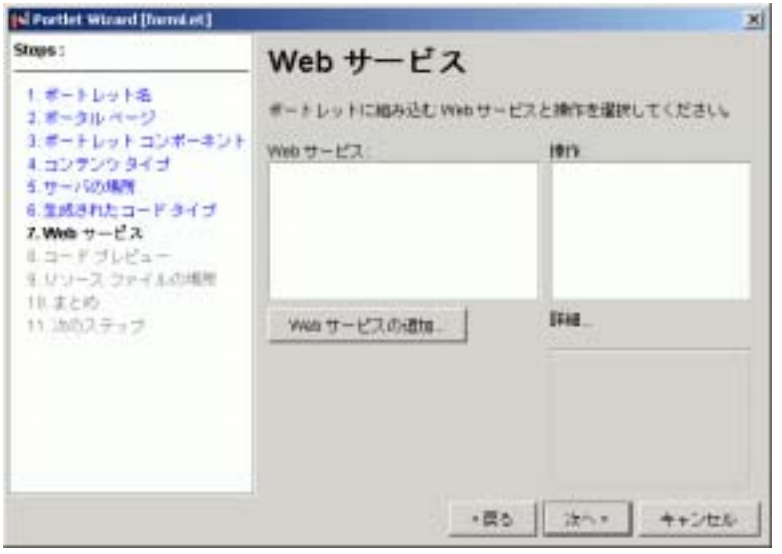
4. [生成されたコード タイプ] 画面が表示されたら、図 11-82 に示すように、[**フォーム**] を選択し、[**次へ**] をクリックします。

図 11-82　生成されるコード タイプの選択



5. [Web サービス] 画面が表示されたら、図 11-83 に示すように、[**Web サービスの追加**] をクリックします。

図 11-83　[Web サービスの追加] のクリック



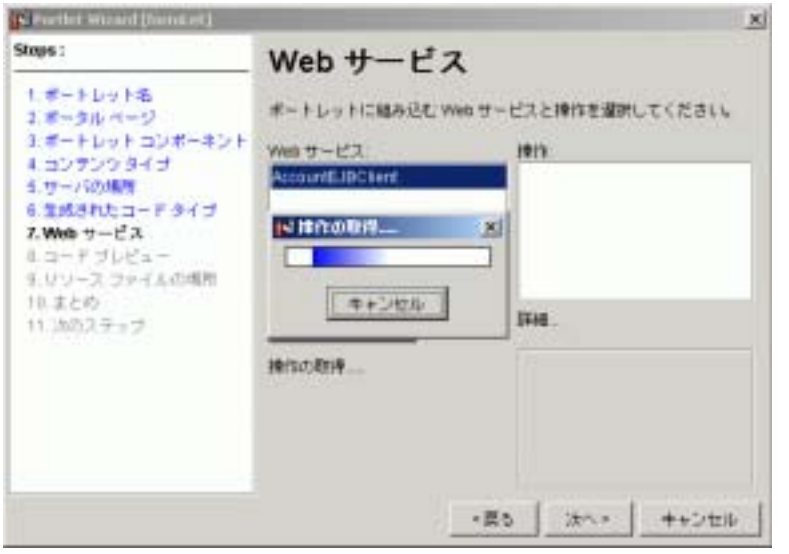
6. 図 11-84 に示すように、WSDL の URL を入力したあと、[**URL の追加**] をクリックします。

図 11-84　WSDL の URL の入力

7. `AccountEJBClient` Web サービスがリストに表示されたら、[**閉じる**] をクリックします。

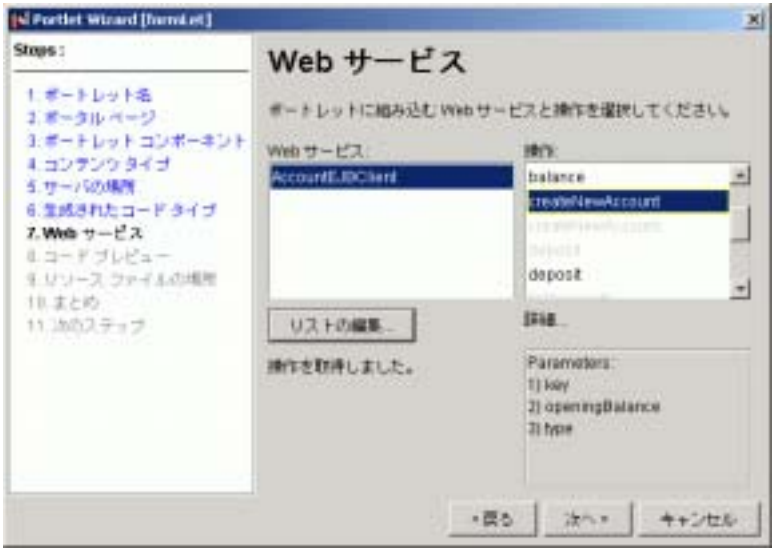
8. [Web サービス] 画面が再び表示されたら、新たに追加した Web サービスをクリックします。図 11-85 に示すように、[操作の取得] というタイトルの小さいウィンドウが [Web サービス] 画面の前面に表示されます。

図 11-85　操作の取得



9. [操作の取得] 画面が閉じたら、右側のリストボックスに操作のリストが表示されるはずです。図 11-86 に示すように、このリストから `CreateNewAccount` 操作を選択し、[**次へ**] をクリックします。

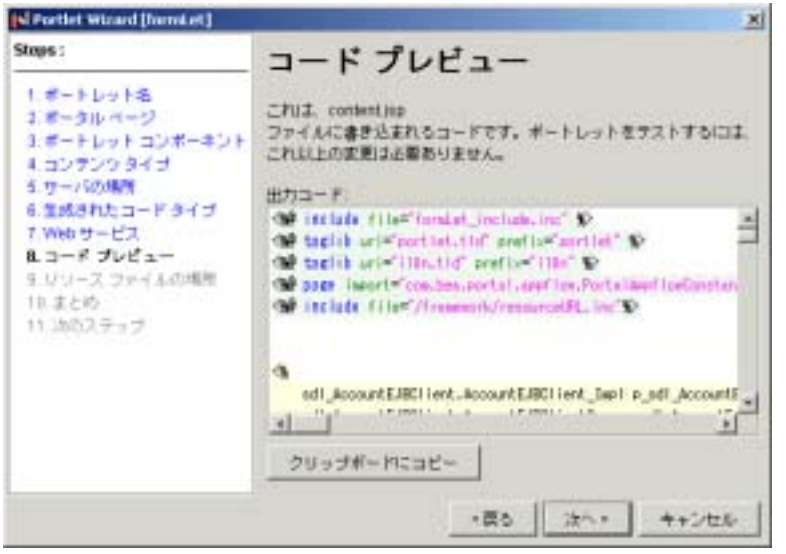
図 11-86　createNewAccount 操作の選択



10. [コード プレビュー] 画面が表示されたら、生成されたコードを下にスクロールして、taglib のインクルード、ポータルの表示更新イベント、そしてもちろん HelloWorld 自体を十分把握してください。[**クリップボードにコピー**] をクリックしエディタにコードを貼り付けて、内容を確認することもできます。

11. 図 11-87 に示すように、[**次へ**] をクリックします。

図 11-87　コードのプレビュー

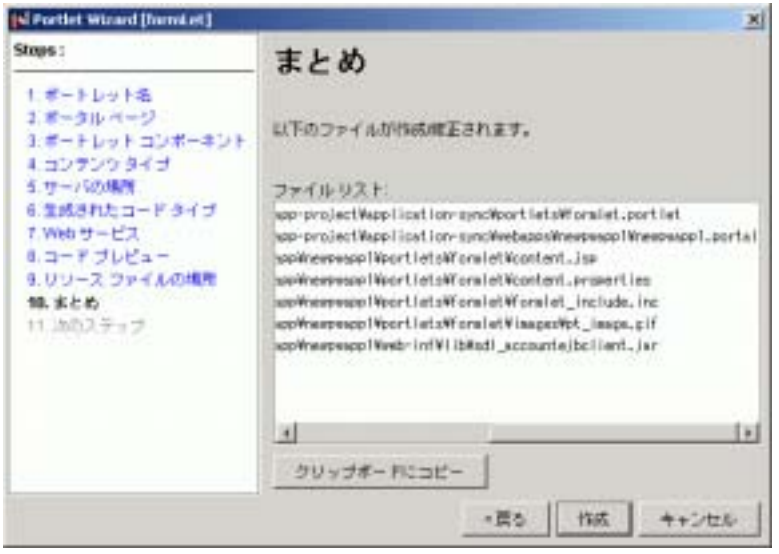


12. 図 11-88 に示すような [リソース ファイルの場所] 画面が表示されたら、ポートレット JSP の格納先ディレクトリが正しいことを確認したあと、[**次へ**] をクリックします。

図 11-88　リソース ファイルの格納場所の選択 / 確認

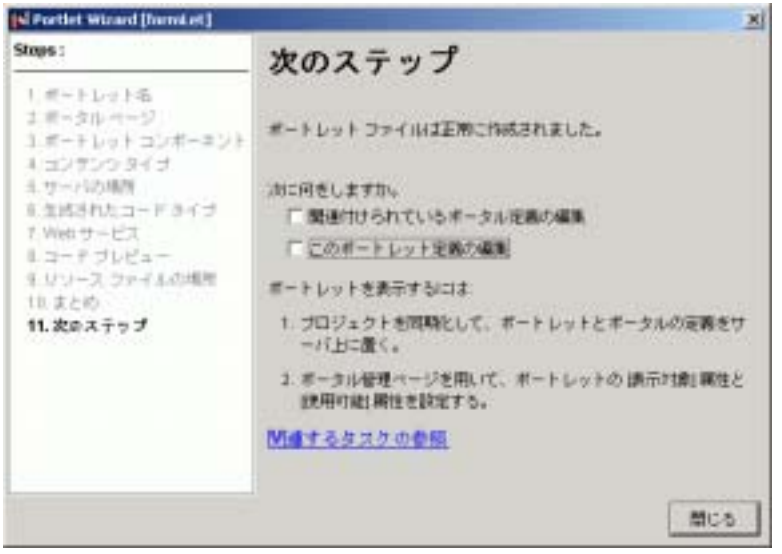
13. 図 11-89 に示すような [まとめ] ページが表示されたら、作成されるファイルを確認し、[**作成**] をクリックします。

図 11-89　ファイル一覧の確認



14. 図 11-90 に示すような [次のステップ] 画面が表示されたら、[**閉じる**] をクリックします。

図 11-90　次のステップの選択



注意： この例では、3 つのポートレットをまず全部作成してから、最後にそれらを一度にデプロイします。

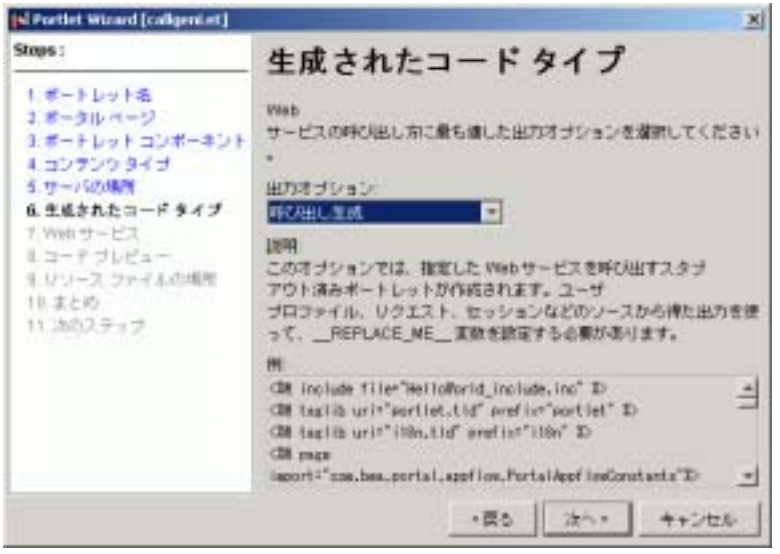
呼び出し生成型 Web サービス ポートレットを作成する

呼び出し生成型ポートレットを使用すれば、ユーザは、WebLogic Workshop サンプル サーバをホストとする Web サービス内に用意されているクラスをリモートで呼び出すことで、銀行口座に預金することができます。

Portlet Wizard を用いて callgenLet というポートレットを作成する

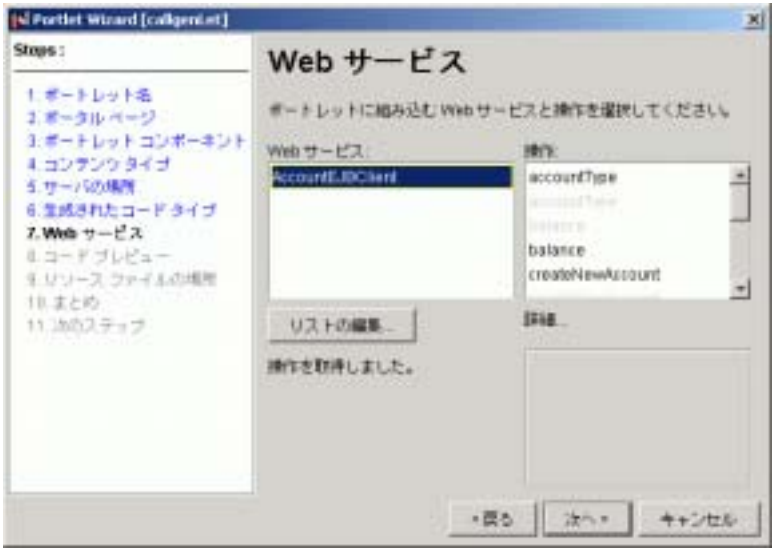
1. 「単純フォーム駆動型 Web サービス ポートレットを作成する」の節で説明した手順に従って、手順 4 まで実行します。
2. [生成されたコード タイプ] 画面が表示されたら、図 11-91 に示すように、[**呼び出し生成**] を選択し、[**次へ**] をクリックします。

図 11-91　Web サービス コード タイプの選択



3. [Web サービス] 画面が表示されたら、図 11-92 に示すように、[リストの編集] をクリックします。

図 11-92　[Web サービス] 画面



4. 図 11-93 に示すような [Web サービス リスト] 画面が表示されたら、以下の WSDL を入力し、[**URL の追加**] をクリックします。

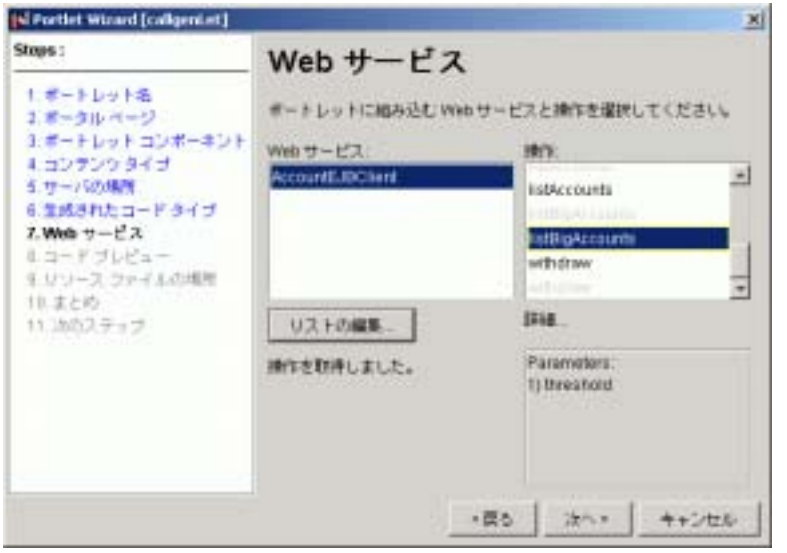
```
http://localhost:7001/samples/ejbControl/
AccountEJBClient.jws?WSDL
```

図 11-93　別の Web サービス WSDL の追加

5. 入力した URL が [Web サービス] リストに追加されたら、[**閉じる**] をクリックします。[Web サービス] 画面が再び表示され、新たに追加した Web サービスが左側のリストに、操作が右側のリストにそれぞれ表示されます。
6. `AccountEJBClient` Web サービスをクリックし、その操作が取得され右側のリストに表示されるのを待ちます。
7. 図 11-94 に示すように、`AccountEJBClient` の操作のリストで「**listBigAccounts**」をクリックし、[**次へ**] をクリックします。

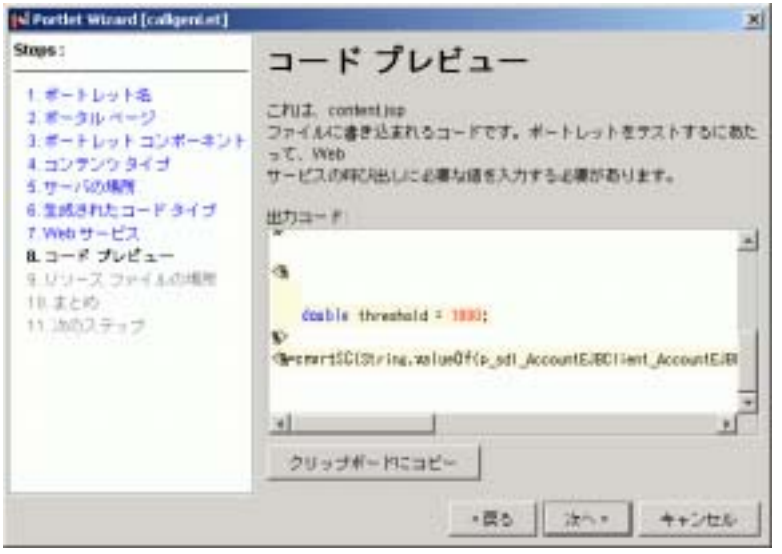
注意 : 操作の下に表示されるパラメータ リストに注目してください。

図 11-94 listBigAccounts 操作の選択



8. 図 11-95 に示すような [コード プレビュー] ページが表示されたら、`threshold` パラメータの値 (コード リスト 11-6 を参照) が **_REPLACE ME_** という文字列になっていることに注意してください。これらのパラメータ値を、コード リスト 11-6 で太字で示されている値に置き換えたあと、[**次へ**] をクリックします。

図 11-95　コードのプレビュー



注意： コード リスト 11-6 の中では、以下のエントリ (コード内では太字で示されている) は短縮表記されています。

```
callImpl was originally p_sdl_AccountEJBClient_AccountEJBClient_Impl
callSoap was originally p_sdl_AccountEJBClient_AccountEJBClientSoap
```

コード リスト 11-6　呼び出し生成型ポートレットのコード

```
<%@ include file="callgenlet1_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>
<%@ include file="/framework/resourceURL.inc"%>
<%
 sdl_AccountEJBClient.AccountEJBClient_Impl callImpl = new
sdl_AccountEJBClient.AccountEJBClient_Impl();
 sdl_AccountEJBClient.AccountEJBClientSoap callSoap =
callImpl.getAccountEJBClientSoap();
```

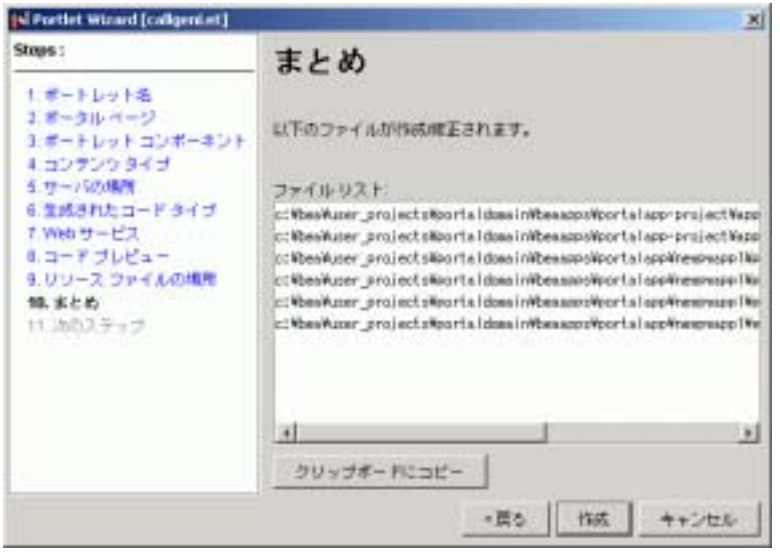
```
%>
<%
    double threshold = 1000;
%>
<%=cnvrtSC(String.valueOf(callSoap.listBigAccounts(threshold)))%>
```

9. 図 11-96 に示すような [リソース ファイルの場所] 画面が表示されたら、ファイルのインストール先ディレクトリが正しいことを確認したあと、[**次へ**] をクリックします。

図 11-96　リソース ファイルの格納場所の選択 / 確認

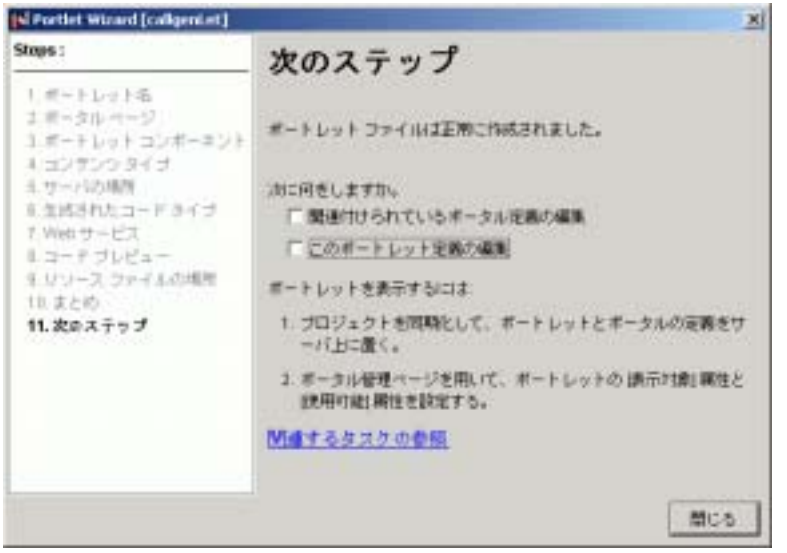
10. 図 11-97 に示すような [まとめ] 画面が表示されたら、作成されるファイルが正しいことを確認し、[**作成**] をクリックします。

図 11-97　ファイル一覧の確認



11. 図 11-98 に示すような [次のステップ] 画面が表示されたら、[**閉じる**] をクリックします。

図 11-98　次のステップの選択



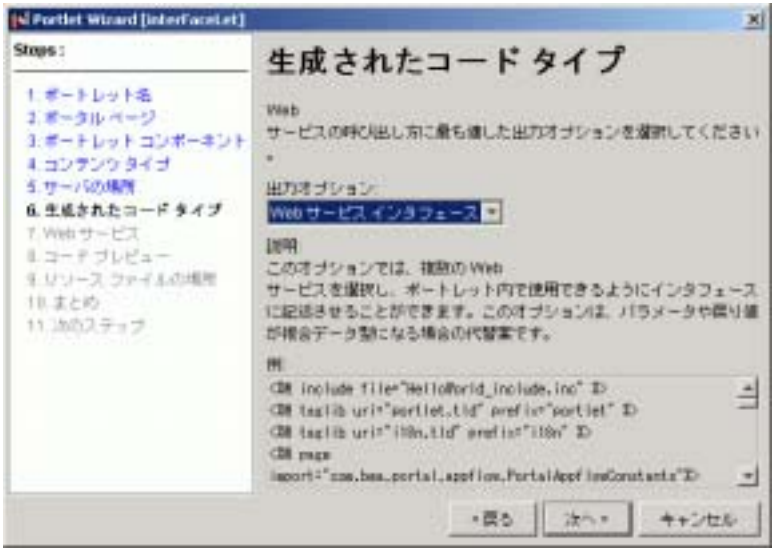
Web サービス インタフェース型ポートレットを作成する

このポートレットでは、ポートレット内に設定されたしきい値を上回る残高の口座を検索し、それらのリストをポートレットに表示します。ここで重要なのは、このポートレットと Web サービスとの通信方法です。このポートレットでは、`listBigAccounts` という自己記述型 WSDL に従って実装されるクラスを用います。

Portlet Wizard を用いて interFaceLet というポートレットを作成する

1. 「単純フォーム駆動型 Web サービス ポートレットを作成する」の節で説明した手順に従って、手順 4 まで実行します。
2. [生成されたコード タイプ] 画面が表示されたら、図 11-99 に示すように、[**Web サービス インタフェース**] を選択し、[**次へ**] をクリックします。

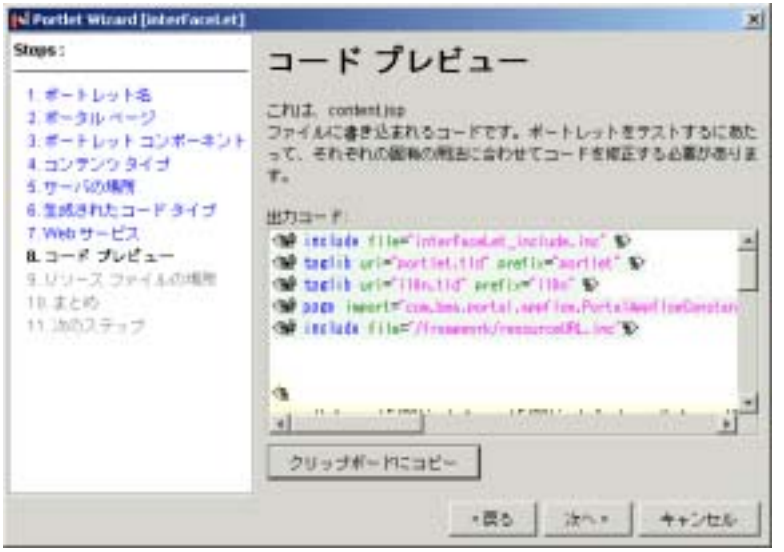
図 11-99　生成されるコード タイプの選択



注意： インタフェース生成型ポートレットの [Web サービス] 画面では、利用可能な操作が表示されないことに注意してください。その理由は、インタフェース記述を提供されたうえで、必要なメソッドを自分で実装する必要があるからです。

3. 図 11-100 に示すような [コード プレビュー] 画面でポートレット コードを編集し、[**次へ**] をクリックします。

図 11-100　interFaceLet の [コード プレビュー] 画面



4. しきい値を上回る残高の口座をリストアップするのに必要なインタフェースを、コード リスト 11-7 に示すように実装します。

   注意 : コード リスト 11-7 の中では、以下のエントリ (コード内では太字で示されている) は短縮表記されています。

   intImpl は本来は p_sdl_AccountEJBClient_AccountEJBClient_Impl

   intSoap は本来は sdl_AccountEJBClient.AccountEJBClientSoap

コード リスト 11-7　interFaceLet のコード

```
<%@ include file="interFaceLet_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>
<%@ include file="/framework/resourceURL.inc"%>
<%

sdl_AccountEJBClient.AccountEJBClient_Impl intImpl = new
    sdl_AccountEJBClient.AccountEJBClient_Impl();

sdl_AccountEJBClient.AccountEJBClientSoap intSoap =
    intImpl.getAccountEJBClientSoap();

...
```

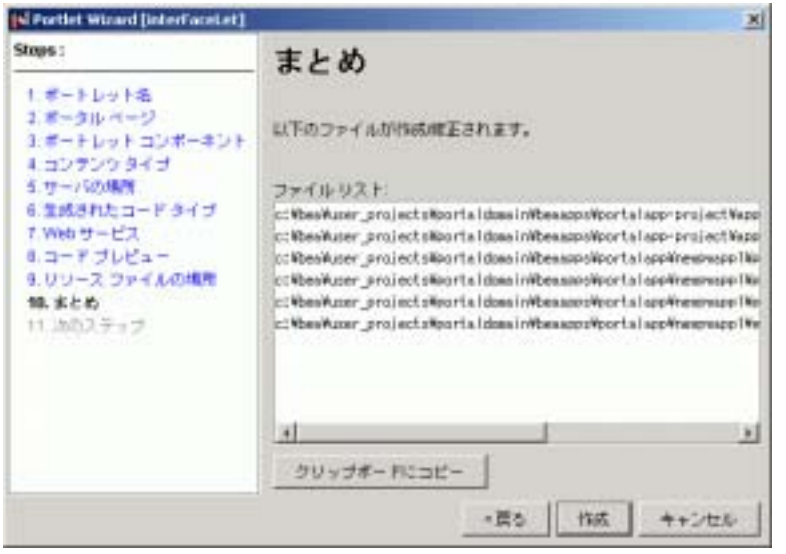
```
*/%>
```

5. 図 11-101 に示すような [リソース ファイルの場所] 画面が表示されたら、ファイルのインストール先ディレクトリが正しいことを確認したあと、[**次へ**] をクリックします。

図 11-101　リソース ファイルの格納場所の選択 / 確認

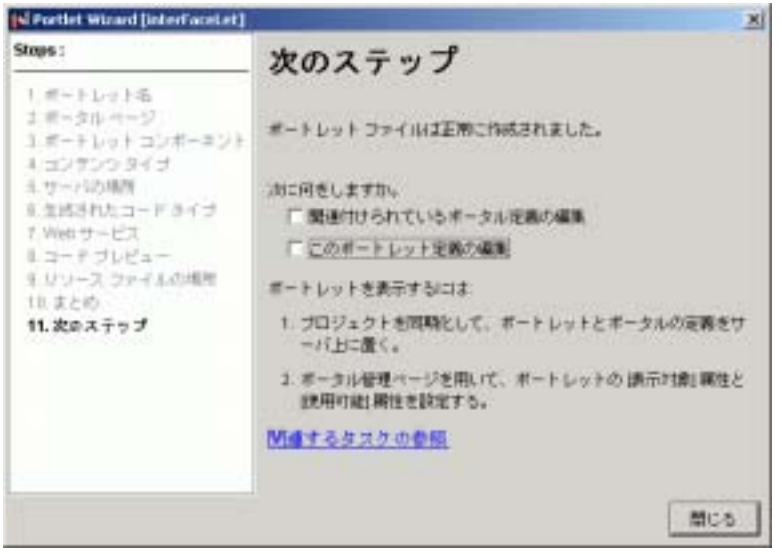
6. 図 11-102 に示すような [まとめ] 画面が表示されたら、作成されるファイルが正しいことを確認し、[**作成**] をクリックします。

図 11-102　ファイル一覧の確認



7. 図 11-103 に示すような [次のステップ] 画面が表示されたら、[**閉じる**] をクリックします。

図 11-103　次のステップの選択



8. E-Business Control Center のツールバーで、[**同期**] ボタンをクリックします。
9. 図 11-104 に示すように、同期が完了したことを知らせるメッセージが [アプリケーションの同期中] ウィンドウに表示されたら、[**閉じる**] をクリックします。

図 11-104　同期の完了

Web サービス ポートレットをデプロイする

これでポートレットがすべて作成されインストールされたので、あとは、WebLogic Server インスタンスでそれらを利用できるようにする必要があります。それには、これらのポートレットが動作するポータル Web アプリケーションを、以下の手順に従って再デプロイするだけです。

1. Web ブラウザで、以下の URL を入力して WebLogic Server コンソールにアクセスします。

   `http://<host>:<port>/console`

2. weblogic/weblogic という資格でログインします。

3. [デプロイメント | アプリケーション | NewPWApp] を選択し、図 11-105 に示すように、右ペインの [**デプロイ**] タブをクリックします。

図 11-105　コンソールの左タブ : [デプロイメント | アプリケーション | NewPWApp]

4. portalServer ターゲットの右側にある [**アンデプロイ**] ボタンをクリックして、NewPWApp をアンデプロイします。[デプロイメント アクティビティ] の状況が [完了しました。] に変わったら、[**デプロイ**] をクリックします。

5. [デプロイメント アクティビティ] の状況が [完了しました。] に変わったら、新しいポートレットの再デプロイは完了です。

6. WebLogic Portal Administration Tools を用いて、新しいポートレットを表示対象かつ利用可能にします。Web ブラウザで、次の URL にアクセスします。すなわち、`http://<hostname>:<port>/portalAppTools` です。

7. administrator/password の組み合わせでログインし、図 11-106 に示すように、[**ポータル管理**] をクリックします。

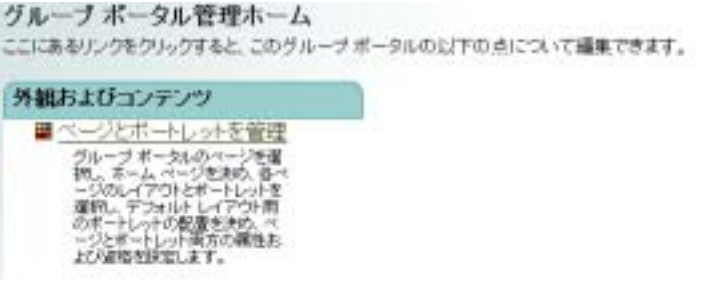
図 11-106　[ポータル管理] へのアクセス

8. [ポータル管理ホーム] ページで、図 11-107 に示すように、**デフォルト ポータル**をクリックします。

図 11-107　デフォルト グループ ポータルの選択

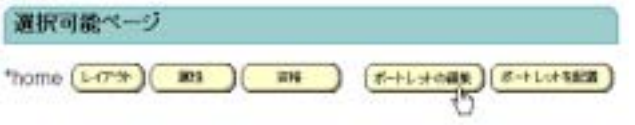
9. [グループ ポータル管理ホーム] ページで、図 11-108 に示す [**ページとポートレットを管理**] をクリックします。

図 11-108 [ページとポートレットを管理]



10. 図 11-109 に示すように、ポータル ページの行の [**ポートレットの編集**] をクリックします。

図 11-109 [ポートレットの編集] のクリック



11. 3 つのポートレットの属性を [表示対象] および [利用可能] に設定し、[**保存**] をクリックします。

12. **[保存] のクリック**：これによって、formLet、callGenLet、および interFaceLet の各ポートレットの属性が正しく設定されます。

Web サービス ポートレットを閲覧する

これで、ポートレットがすべてデプロイされ、同じポータル ページ内に配置されたので、以下の手順に従って、ポートレットから提供される機能を実際に確かめてみましょう。

1. 以下の URL を用いて、ポートレットにアクセスできることを確かめます。

```
http://<host>:<port>/NewPWApp/
```

図 11-110 に示すような結果になるはずです。

2. 有効なユーザおよび 2 つの新しい口座 (残高が 1001 の口座と、同じく 751 の口座) としてログオンします。しきい値を上回る残高の口座をリストアップする listBigAccounts ポートレット (interFaceLet) の表示に注目してください。
3. callGenlet ポートレットの content.jsp に記載されているしきい値を 750 に変更します。
4. 749 の残高の口座を新たに作成します。しきい値を上回る残高の口座をリストアップします。

図 11-110　口座を入力する前の Web サービス ポートレット

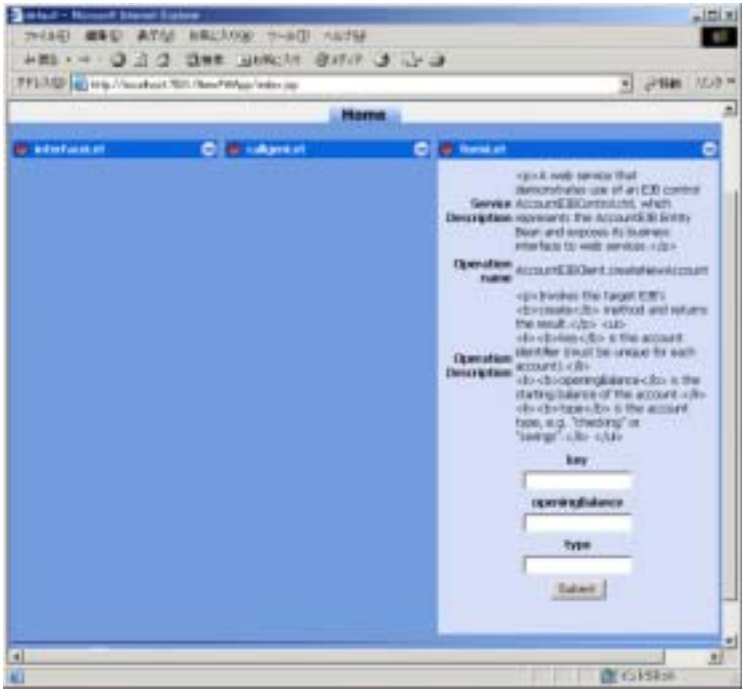
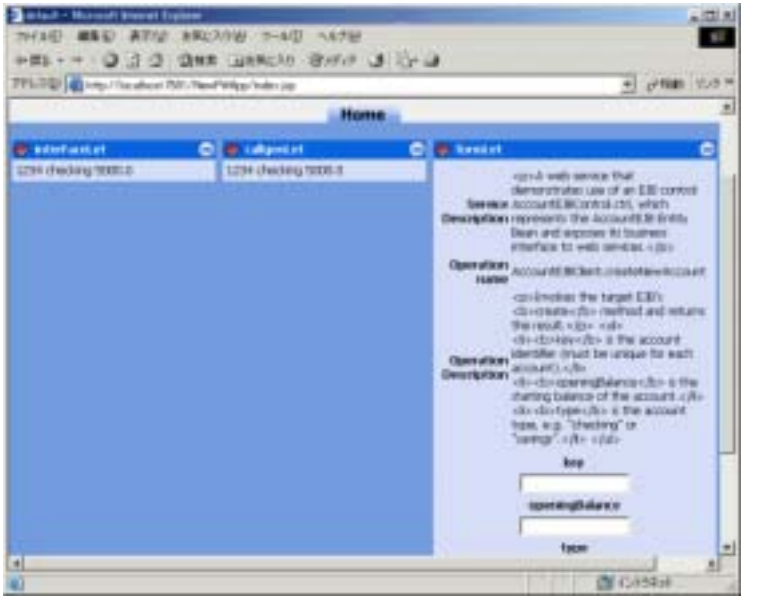


図 11-111　口座を入力したあとの Web サービス ポートレット



注意： Web サービスの詳細については、「WebLogic Platform での Web サービスの作成」(http://edocs.beasys.co.jp/e-docs/platform/docs70/interm/webserv.html) と「WebLogic Workshop の概要」(http://edocs.beasys.co.jp/e-docs/workshop/docs70/index.html) を参照してください。

Web サービスを非同期的に呼び出す

BEA WebLogic Portal 7.0 では、ポートレットは Web サービスとの間で会話などの非同期通信を行うことができます。以下の例では、ローカル サーバをホストとする Web サービスとやり取りする単純な会話ポートレットを作成する方法を示します。

図 11-112　会話 Web サービス ポートレット

会話ポートレットの概要

この例で作成するポートレットには、ボタンで起動される以下の 3 つのアクションが用意されます。

- [**Start**] ボタンをクリックすると、セッション ID が Web サービスに送信され、Web サービス側ではその ID を会話用のトークンとして保持する。
- [**Continue**] ボタンをクリックすると、トークンのステータスが要求される。
- [**Finish**] ボタンをクリックすると、会話が終了し、その結果、Web サービスはトークンを破棄し、それ以上メッセージを待つのをやめる。

準備

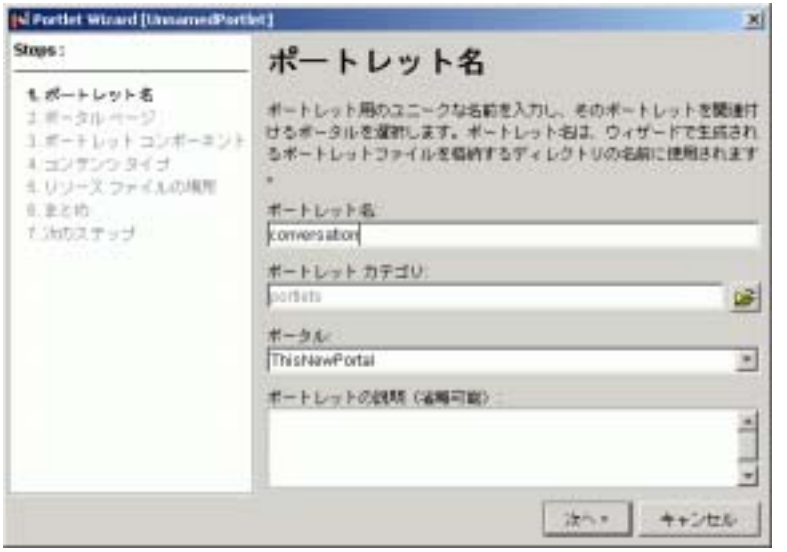
- [プログラム｜BEA WebLogic Platform 7.0｜WebLogic Workshop｜WebLogic Workshop Examples ｜ Start Examples Server] を選択して、WebLogic Workshop のサンプル サーバを起動します。「scriptDemo ポートレット」の節で説明したサンプル ポータルを基に作業を始めます。
- 使用するドメインのポータル サーバを起動します。それには、この例では、[プログラム｜ BEA WebLogic Platform 7.07.0 ｜ User Projects ｜ MyNewDomain ｜ Start Portal Server] を選択します。

会話ポートレットを作成する

サンプルの会話ポートレットを作成しデプロイするには、以下の手順に従います。

1. 「Web サービス インタフェース型ポートレットを作成する」の節で示したとおりに、Portlet Wizard を用いて Web サービス インタフェース ポートレットを作成します。図 11-113 に示すように、このポートレットの名前を「conversation」にし、[**次へ**] をクリックします。

図 11-113　会話ポートレット conversation の作成



2. この新しいポートレットを「home」というポータル ページに関連付け、[**次へ**] をクリックします。
3. ポートレット コンポーネント] ページが表示されたら、追加コンポーネントを指定せずに [**次へ**] をクリックします。
4. [コンテンツ タイプ] 画面が表示されたら、[Web サービス] を選択し、[**次へ**] をクリックします。
5. [サーバの場所] 画面が表示されたら、WebLogic Server のインスタンスの場所を指定し、[**次へ**] をクリックします。
6. [生成されたコード タイプ] 画面が表示されたら、[Web サービス インタフェース] を選択し、[**次へ**] をクリックします。
7. [Web サービス] 画面が表示されたら、[Web サービスの追加] をクリックし、以下の URL を入力します。
   `http://localhost:7001/samples/async/Conversation.jws?WSDL`
   [**URL の追加**] をクリックします。
8. Conversation Web サービスがリストに表示されたら、[**閉じる**] をクリックします。
9. 左側のリストから Conversation Web サービスを選択し、[**次へ**] をクリックします。
10. [コード プレビュー] 画面が表示されたら、[**次へ**] をクリックします。このコードは、後で置き換えられます。
11. [リソース ファイルの場所] 画面が表示されたら、ポートレット JSP の格納先ディレクトリが正しいことを確認したあと、[**次へ**] をクリックします。
12. [まとめ] ページが表示されたら、作成されるファイルを確認し、[**作成**] をクリックします。
13. [次のステップ] 画面が表示されたら、チェックボックスがすべて選択解除されていることを確かめてから、[**閉じる**] をクリックします。
14. テキスト エディタで、コード リスト 11-8 に示すコードを入力し、それを `content.jsp` という名前で以下のディレクトリに保存します。
    `myNewDomain\beaApps\portalApp\NewPWApp\portlets\conversation\`

コード リスト 11-8 Conversation Web サービス ポートレットの content.jsp

```
<%@ include file="Conversation_include.inc" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ page import="org.openuri.www.StartRequest"%>
<%@ page import="org.openuri.www.GetRequestStatusResponse"%>
<%@ page import="org.openuri.www.x2002.x04.soap.conversation.StartHeader"%>
<%@ page import="org.openuri.www.x2002.x04.soap.conversation.ContinueHeader"%>
<%@ page import="weblogic.xml.schema.binding.internal.builtin.VoidType"%>
<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>
<%@ page import="com.bea.portal.appflow.PortalAppflowConstants"%>
<%@ include file="/framework/resourceURL.inc"%>

<%
    DL_wsdl_Conversation.Conversation_Impl conversationImpl = new
DL_wsdl_Conversation.Conversation_Impl();
    DL_wsdl_Conversation.ConversationSoap soap =
conversationImpl.getConversationSoap();
%>

<%
    String target = request.getParameter("target");
    String conversationID = session.getId();
        if ( conversationID == null )
            conversationID = "";
%>

<portlet:form event="<%= PortalAppflowConstants.PORTLET_REFRESH %>">
    <table border="0" align="center">
        <tr>
            <td width="100%" align="center">
            <table border="0" align="left">
            <tr>
<%
                if ( target != null
                     && target.equals("start")
                     &&  true  )
                {
                    try
                    {
                     StartHeader startHeader = new StartHeader(conversationID,
"http://localhost:7001/samples/async/Conversation.jws");
            VoidType startResponse = new VoidType();

            StartRequest begin = new StartRequest(false);

            startResponse = soap.startRequest(begin, startHeader);
```

```
%>
            <td><%=cnvrtSC("Conversation started with ID: " +
String.valueOf(conversationID))%></td>
<%
              }
               catch (java.rmi.RemoteException e)
              {
%>
            <td><%=cnvrtSC("Duplicate conversation id for start: " +
String.valueOf(conversationID))%></td>
<%


            e.printStackTrace();
             }
                }
%>
            </tr>
            </table>
            </td>
        </tr>
        <tr>
            <td width="100%" align="center"><input type="submit" name="start"
value="Start"></td>
        </tr>
    </table>
    <br><br>
    <input type="hidden" name="target" value="start">
</portlet:form>

<portlet:form event="<%= PortalAppflowConstants.PORTLET_REFRESH %>">
    <table border="0" align="center">
        <tr>
            <td width="100%" align="center">
            <table border="0" align="left">
            <tr>
<%
                if ( target != null
                     && target.equals("continue")
                     &&  true  )
                {
                    try
                    {
            ContinueHeader continueHeader = new
ContinueHeader(conversationID);
            GetRequestStatusResponse status = soap.getRequestStatus(null,
continueHeader);
            String result = status.getGetRequestStatusResult();
%>
```

```
            <td><%=cnvrtSC("Response: " + String.valueOf(result))%></td>
<%
                }
                catch ( Exception e )
                {
                    e.printStackTrace();
                 }
             }
%>
            </tr>
            </table>
            </td>
        </tr>


        <tr>
            <td width="100%" align="center"><input type="submit" name="continue"
value="Continue"></td>
        </tr>
    </table>
    <br><br>
    <input type="hidden" name="target" value="continue">
</portlet:form>

<portlet:form event="<%= PortalAppflowConstants.PORTLET_REFRESH %>">
    <table border="0" align="center">
        <tr>
            <td width="100%" align="center">
            <table border="0" align="left">
            <tr>
<%
                if ( target != null
                     && target.equals("finish")
                     &&  true  )
                {
                    try
                    {
                     VoidType terminateResponse = new VoidType();
                     ContinueHeader finishHeader = new
ContinueHeader(conversationID);
                   terminateResponse = soap.terminateRequest(null, finishHeader);
%>
                <td><%=cnvrtSC("Conversation terminated.")%></td>
<%
                    }
                    catch ( java.rmi.RemoteException e )
                    {
%>
                <td><%=cnvrtSC("Conversation already terminated.")%></td>
```

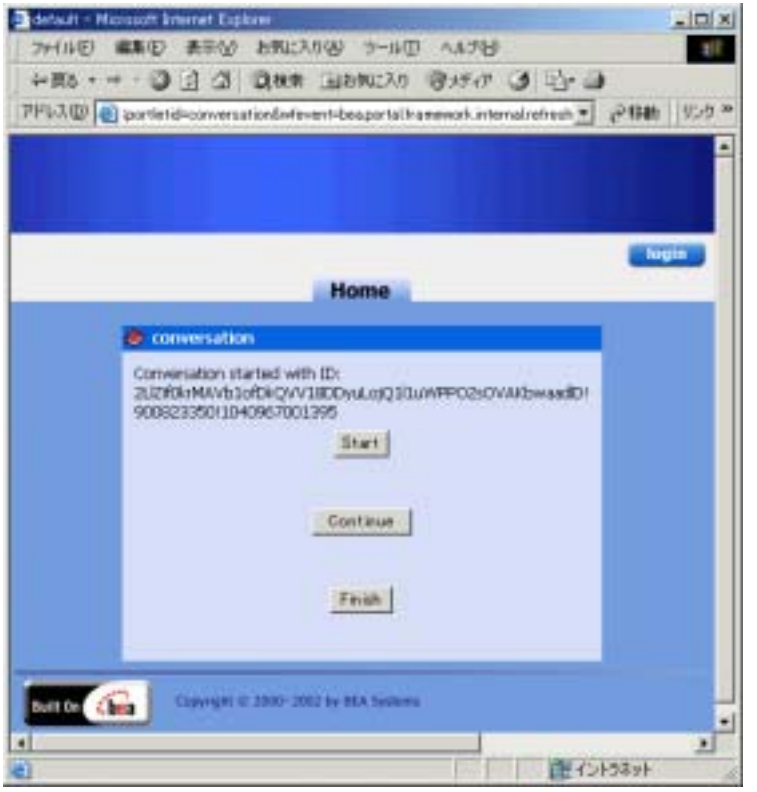
```
<%
                        e.printStackTrace();
                    }
                }
%>
            </tr>
            </table>
            </td>
        </tr>
        <tr>
            <td width="100%" align="center"><input type="submit" name="finish"
value="Finish"></td>
        </tr>
    </table>
    <br><br>


    <input type="hidden" name="target" value="finish">
</portlet:form>
```

15.「Web サービス ポートレットをデプロイする」の節の説明に従って、会話ポートレットをデプロイします。

16. 図 11-114 に示すように、それぞれのボタンをクリックし、その結果を確かめることで、新しいポートレットをテストします。

図 11-114　会話の開始



Web サービス ポートレット内でのエラー処理

実際の運用場面では、ポートレットの接続先の Web サービスは通常、他のマシンをホストにしており、利用する側の自由にはなりません。ポータル フレームワークでは、Web サービスに関する問題を処理するために特に用意された以下の 2 つのエラーをポートレットから発生させることができます。

JAXRPCException: 実行時に Web サービスが利用できない場合には、ポートレットは `javax.xml.rpc.JAXRPCException` を送出します。生成されるポートレットでこの例外を捕捉することで、JSP にエラー処理を追加します。

注意： JAXRPCException は、サービスの遅延が発生する場合ではなく、接続が拒否される場合に適用されます。

SOAPFaultException: Web サービス ポートレットの作成時に Portlet Wizard で生成された SOAP リクエストを Web サービスで処理できない場合には、javax.xml.rpc.soap.SOAPFaultException が送出されます。JSP 内でこの例外を捕捉して、コンパイルが失敗しないようにする必要があります。

既存の Web アプリケーションのポータル化

既存の非ポータル Web アプリケーションをポータル フレームワークに移行するには、一定の修正が必要です。この節では、インストール済みの WebLogic Platform 環境に用意されている例を用いて、このプロセスの概要を説明します。

はじめに

ポートレットに機能を追加する方法の 1 つは、既存の (非ポータル) Web アプリケーションの JSP コードを、ポートレットを構成する JSP に移植することです。このチュートリアルでは、製品に付属している国際化ポートレットのサンプル コードを手直しして用い、その機能をポートレット内に移植します。図 11-115 と 図 11-116 に示したのは、ユーザ入力に基づいて特定言語向けのコンテンツを表示する国際化サンプル アプリケーションです。

注意： このアプリケーションの起動と操作の詳細については、WebLogic Platform マニュアルの「Personalization サンプル」の章を参照してください。

図 11-115　国際化の入力

図 11-116　国際化の結果

必要条件

WebLogic Portal 7.0 (サービス パック 1) が正常にインストールされている必要があります。

プロセスの概要

このプロセスは、以下のステップから成ります。

ステップ 1: ポータル Web アプリケーションを作成する

新しいポータル Web アプリケーションを作成して、対象とする新しいアプリケーションの基本構造として使用する手順については、『WebLogic Portal 7.0 開発者ガイド』の「ポータルの新規作成」というチュートリアルを参照してください。

この例では、このポータル Web アプリケーションを `NewPWApp` と呼びます。

注意： アプリケーションでパーソナライゼーションや国際化などのポータル サービスを利用する場合には、Portal Wizard を用いて作成したポータルに、そうした機能のサポートを追加する必要があります。ポータルへのこれらの機能の追加について詳しくは、『WebLogic Portal 7.0 開発者ガイド』の「カスタム テンプレートの作成」の章の「ドメインへの全ポータル サービスの追加」という節を参照してください。

ステップ 2: 2 ページ構成の Webflow ポートレットを作成する

新しいドメインに作成したポータル サーバの稼働中に、E-Business Control Center を用いて Portlet Wizard を起動します。2 ページ構成の Webflow ポートレットを作成し、その名前を **i18n** とします。Webflow ポートレットのこうした作成方法の詳細については、「順次型 Webflow ポートレットの作成」の節を参照してください。

注意： 必ず、WebLogic Portal Administration Tools を用いて、ポートレットを表示対象かつ利用可能に設定してください。

ステップ 3: ポートレットのコードを編集する

このステップでは、ポートレット Webflow を使用しパーソナライゼーションを呼び出すように、JSP とプロパティ ファイルを編集します。

Portlet JSP を置き換える

まず、Portlet Wizard で生成された JSP を、パーソナライゼーション サービスを呼び出しコンテンツに応じた動作をする JSP に置き換える必要があります。コード リスト 11-9 と コード リスト 11-10 の内容を以下のディレクトリに保存します。

```
<BEA_HOME>beaApps\portalApp\NewPWApp\portlets\i18n
```

コード リスト 11-9 Page1.jsp

```
<%-------------------------------------------------------------
Copyright (c) 2000-2002  BEA Systems, Inc.  All rights reserved.
-------------------------------------------------------------%>


<%-------------------------------------------------------------
ファイル： Page1.jsp
```

```
目的： 国際化の対象言語を選択するためのフォーム入力を収集する。
------------------------------------------------------------%>


<%@ taglib uri="webflow.tld" prefix="webflow"%>
<%@ taglib uri="portlet.tld" prefix="portlet"%>


<%------------------------------------------------------------
有効および無効なフォーム エントリの HTML フォント スタイルを宣言する。
これらは、Webflow による検証が組み込まれたフォームで使用されることになる。
------------------------------------------------------------%>


<% String validStyle = "background: white; color: black;
font-family: Arial"; %>
<% String invalidStyle = "background: white; color: red;
font-style: italic"; %>


<center>
<%-------------------------------------------------------
ポートレットの検証機能付きフォームを使用する
------------------------------------------------------------%>
<portlet:validatedForm event="switch2" applyStyle="message"
     messageAlign="right" validStyle="<%= validStyle %>"
     invalidStyle="<%= invalidStyle %>" unspecifiedStyle="<%=
validStyle %>">


<table border="0" cellspacing="0" cellpadding="0" width="100%">
  <tr>
    <td>
      <table border="0" cellpadding="6" cellspacing="1"
width="100%">
        <tr class="header">
          <td colspan="2">
            Localization of Flickerstick Band Information
```

```
          </td>
        </tr>
        <tr class="tablerow1">
         <td align="right" valign="top" width="1%">Languages:</td>
          <td>
<%------------------------------------------------------------
HTML の標準の select および option の代わりに <webflow:select> と
<webflow:option> を使用することで、フォーム検証を可能にする。
------------------------------------------------------------%>
            <webflow:select name="language" size="5">
              <webflow:option value="en"/>English
              <webflow:option value="fr"/>French
              <webflow:option value="es"/>Spanish
            </webflow:select>
          </td>
        </tr>
        <tr class="tablerow1">
          <td align="right" valign="top" width="1%"> </td>
          <td>
            <input type="submit" name="Submit" value="Show Me!">
          </td>
        </tr>
        <tr class="tablerow2">
          <td class="label" colspan="2">
            Select the language in which you would like to view
Flickerstick information.
          </td>
        </tr>
      </table>


      <input type="hidden" name="resultFile" value="Page2.jsp">
      <input type="hidden" name="sample" value="<%=
request.getParameter("sample") %>">
```

```
    </td>
  </tr>
</table>
</portlet:validatedForm>
</center>
```

コード リスト 11-10　Page2.jsp

```
<%-------------------------------------------------------------
Copyright (c) 2000-2002  BEA Systems, Inc.  All rights reserved.
-------------------------------------------------------------%>
<%-------------------------------------------------------------
ファイル： Page2.jsp
目的： 国際化の対象言語を選択するためのフォーム入力を収集する。
-------------------------------------------------------------%>
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ page import="com.bea.p13n.content.Content" %>
<%@ taglib uri="cm.tld" prefix="cm" %>
<%@ taglib uri="es.tld" prefix="es" %>
<%@ taglib uri="i18n.tld" prefix="i18n" %>
<%@ taglib uri="portlet.tld" prefix="portlet" %>

<%-------------------------------------------------------------
クエリ文字列を合成する
例： isTrackIdentifier='true' && bandName='Flickerstick' &&
language='en'
-------------------------------------------------------------%>
<%
    StringBuffer queryStr = null;
    String language = request.getParameter("language");
```

```
    if (language != null)
    {
       // クエリ文字列を合成する
       queryStr = new StringBuffer();
       queryStr.append("isTrackIdentifier = 'true' && bandName =
'Flickerstick' && language = '");
       queryStr.append(language);
       queryStr.append("'");
    }

    //queryStr = new StringBuffer();
    //queryStr.append("bandName = 'Flickerstick'");

    if (queryStr != null)
    {
%>
<% System.out.println("\n\nqueryStr=" + queryStr +
"----------------------------------------\n\n"); %>
<br>Language is: <%= language %><br>
<br>Query String is: <%= queryStr %><br>
<br>ContentHelper.DEF_CONTENT_MANAGER_HOME is : <%=
ContentHelper.DEF_CONTENT_MANAGER_HOME %><br>
<%------------------------------------------------------------
選択された言語でページをローカライズする。今後、
このリクエストで i18n タグを呼び出すと、この言語がデフォルトになる。
------------------------------------------------------------%>
   <i18n:localize language="<%= language %>"/>

<%------------------------------------------------------------
合成されたクエリ文字列を使用すると、
Flickerstick の曲目が取得される。
------------------------------------------------------------%>
```

```
   <cm:select contentHome="<%=
ContentHelper.DEF_CONTENT_MANAGER_HOME %>"

      sortBy="trackNum" query="<%= queryStr.toString() %>"
id="contentArray" failOnError="true"/>


    <table border="0" cellspacing="0" cellpadding="0" width="100%">

      <tr>

        <td>

          <table border="0" cellspacing="1" cellpadding="6"
width="100%">

            <tr class="tableheader">

<%-------------------------------------------------------------

曲名と曲番号を示すローカライズ済みメッセージを取得する。

-------------------------------------------------------------%>

     <td><i18n:getMessage messageName="trackName"
bundleName="Page2"/></td>

     <td><i18n:getMessage messageName="trackNum"
bundleName="Page2"/></td>

            </tr>


            <% int row = 0; %>

       <br>contentArray length is : <%= contentArray.length %><br>

      <es:forEachInArray id="nextDoc" array="<%= contentArray %>"
type="Content">

    <tr class="<%= (row % 2 == 0) ? "tablerow1" : "tablerow2" %>">

<%---------------------------------------------------------------

cm:getProperty タグを用いて bandName プロパティを取得し、

それを用いて、Webflow に渡すパラメータを合成する。

-------------------------------------------------------------%>

          <td>

   <cm:printProperty id="nextDoc" name="trackName" encode="html"/>

         </td>

         <td>

   <cm:printProperty id="nextDoc" name="trackNum" encode="html"/>
```

```
            </td>
           </tr>
           <% row++; %>
              </es:forEachInArray>
            </table>
          </td>
        </tr>
    </table>
<%
    }
    else
    {
%>
        <b>Please specify one language in your request!</b>
<%
    }
%>
<center>
<a href="<portlet:createWebflowURL event="switch1"/>">Previous
Page</a>
</center>
```

国際化用のプロパティ ファイルを保存する

以下のコード リストの内容を次のディレクトリに保存します。

```
<BEA_HOME>beaApps\portalApp\NewPWApp\portlets\i18n
```

たとえば、コード リスト 11-11 であれば、`Page2_en.properties` として保存されるでしょう。

コード リスト 11-11　Page2_en.properties

```
trackName=Track Name
trackNum=Track Number
```

コード リスト 11-12　Page2_fr.properties

```
trackName=Nom de Piste
trackNum=Numero do Piste
```

コード リスト 11-13　Page2_sp.properties

```
trackName=Nombre de la canción
trackNum=Número de la canción
```

ステップ 4: コンテンツ リソースをロードする

このステップでは、コンテンツ リソースをパーソナライゼーション ドメインからインポートします。

1. <BEA_HOME>weblogic700\samples\portal\p13nDomain 内の dmsBase フォルダ全体 (その内容を含む) を以下のディレクトリにコピーすることで、ポータル ドメイン内の dmsBase フォルダを置き換えます。

   <BEA_HOME>\user_projects\myNewDomain

2. このコンテンツをポータル フレームワークで利用できるようにするには、メタデータをサーバにロードする必要があります。サーバ稼働中に、以下のディレクトリで loaddata スクリプトを実行します。

   <BEA_HOME>\user_projects\myNewDomain

ステップ 5: アプリケーションをテストする

さて、これで JSP が編集されたので、以下の手順に従って、新しいポートレットの機能を確認してみましょう。

1. 以下の URL を用いてポートレットにアクセスできることを確かめます。

```
http://<host>:<port>/NewPWApp/
```

図 11-117 に示すような結果になるはずです。

図 11-117　i18n ポートレットの検証

2. 言語を選択し、[**Show Me**] をクリックします。図 11-118 に示すような結果になるはずです。

図 11-118　i18n ポートレットの結果ページ

パフォーマンス チューニング

この節では、パフォーマンス上の問題のうち、JDBC およびスレッドの設定やいくつかのキャッシュ設定など、WebLogic Portal に特に関係するものを取り上げます。ポータル アプリケーションのパフォーマンスに影響を及ぼすさまざまな要因は、WebLogic Server に固有のものです。これらの調整については、『*WebLogic Server パフォーマンス チューニング ガイド*』(http://edocs.beasys.co.jp/e-docs/wls/docs70/perform/index.html) を参照してください。

キャッシュを用いてパフォーマンスをチューニングする

プロダクション Web サイトのキャッシングを調節するには、以下の要素について検討します。

コンテンツ管理のキャッシングを調節する

プロダクション Web サイトのコンテンツ管理性能を最適化するために、Content Manager では、キャッシング フレームワークを用いて以下のキャッシュのコンフィグレーションと管理を行います。

`documentContentCache`

`documentMetadataCache`

`documentIdCache`

コンテンツ管理用 JSP タグには、これら以外にも一連のキャッシュが用意されており、それらには以下の方法でアクセスすることができます。

`cm:select`、`cm:selectById`、`pz:contentQuery`、および `pz:contentSelector` の各 JSP タグの場合には、可能なかぎり `useCache` 属性を使用します。そうすることで、`DocumentManager` (`pz:ContentSelector` の場合には Rules Manager) を呼び出さなくてもよくなります。

ユーザ属性やドキュメント属性が変わったときに、キャッシュされているコンテンツをクリアするには、com.bea.p13n.content.ContentCache の remove メソッドを使用します。詳細については、『WebLogic Portal Javadoc』で com.bea.p13n.content.ContentCache を参照してください。

cm:select、cm:selectById、pz:contentQuery、および pz:contentSelector の各 JSP タグの場合には、可能なかぎり cacheScope 属性を application に設定します。この application スコープは、エンタープライズ アプリケーションではなく Web アプリケーションに適用されます。例をコード リスト 11-14 に示します。

コード リスト 11-14　cacheScope を *application* に設定

```
<cm:select id="myDocs" query="riskFactor = 'Low'"
useCache="true" cacheId="myDocs"
cacheScope="application"
max="10" cacheTimeout="300000" />
```

application というキャッシュ タイプは、ユーザ単位ではなくグローバルなものであり、それを設定すると DocumentManager EJB が呼び出されなくなるので、クエリの処理速度が上がるはずです。

pz:contentSelector の場合には、共有コンテンツを選択する場合にのみ、cacheScope 属性を application に設定します。たとえば、アプリケーション スコープのキャッシュを用いて非認証ユーザ用のコンテンツを選択するようなアプリケーションを作成するとしましょう。この場合、アプリケーション スコープを用いているので、すべての非認証ユーザに対して同じコンテンツが表示されます。アプリケーションでは、認証済ユーザに対しては、セッション スコープのキャッシュに切り換えることで、パーソナライズされたコンテンツを提供します。

ユーザが次に閲覧するドキュメントを、現在閲覧中のドキュメントに基づいて予測できる場合には、常に、ユーザが要求する前に次のドキュメントをキャッシュにロードしておきます。このような「先行キャッシング」を行うと、ユーザ リクエストに対する WebLogic Portal の応答速度は大幅に向上します (ただし、予測

が正しいと仮定した場合の話です。つまり、誰も要求しないドキュメントを先行キャッシングしても、パフォーマンスとスケーラビリティが低下するだけです)。

コード リスト 11-15 には JSP の一部を示しますが、この中には、ドキュメントの先行キャッシングの例が含まれています。

コード リスト 11-15 ドキュメントの先行キャッシング

```
<%-- 最初のコンテンツ セットを取得する --%>
<cm:select id="myDocs" query="riskFactor = 'Low'"
useCache="true" cacheId="myDocs"
cacheScope="application"
max="10" cacheTimeout="300000" />
<%-- 各コンテンツの relatedDocId からクエリを生成する --%>
<% String query = null; %>
<es:forEachInArray array="<%=myDocs%>" id="myDoc"
type="com.bea.p13n.content.Content">
<% String relId = (String)myDoc.getProperty("relatedDocId", null);
%>
<es:notNull item="<%=relId%>">
<%
if (query != null)
query += " || ";
else
query = "";
query += "identifier = '" +
ExpressionHelper.toStringLiteral(relId) + "'";
%>
</es:notNull>
</es:forEachInArray>
<%-- cm:select を通じて関係のあるコンテンツをキャッシュにロードする --%>
<es:notNull item="<%=query%>">
```

```
<cm:select query="<%=query%>" id="foo" useCache="true"
cacheId="relatedDocs"
cacheScope="session" max="10" cacheTimeout="300000" />
</es:notNull>
```

コンテンツ管理用 JSP タグの詳細については、『*JavaServer Pages ガイド*』の「パーソナライゼーション JSP タグ」の章 (http://edocs.beasys.co.jp/e-docs/wlp/docs70/jsp/p13njsp.htm) を参照してください。

クラスタ環境でのプロパティ キャッシング

ユーザ、グループといったプロパティ データへのアクセスに要する時間を短縮するために、WebLogic Server の Configurable Entity および Entity Property Manager では、キャッシング フレームワークを用いて以下のキャッシュのコンフィグレーションと管理を行います。

`ldapGroupCache`

`ldapUserCache`

`entityPropertyCache`

`entityIdCache`

`unifiedProfiletypeCache`

`propertyKeyIdCache`

注意： デフォルトでは、これらのプロパティ キャッシュは有効になっています。

クラスタ環境でプロパティ キャッシングが有効になっている場合には、クラスタ内の各サーバは独自にキャッシュを保持し、そのキャッシュは他のサーバにはレプリケートされません。このような環境では、あるサーバ上のキャッシュに格納されているプロパティが変化しても、別のサーバ上のキャッシュ内のプロパティがそれに合わせて変化するとはかぎりません。ほとんどの場合、別のサーバ上のプロパティに即座にないしは迅速にアクセスする必要はありません。ユーザセッションは単一のサーバに固定されており、キャッシングが有効になっていても、そのサーバ上でユーザが自分自身の設定に対して行った変更は、すぐに反映されます。

ユーザと管理者がクラスタ内の異なるサーバに固定的に割り当てられていると、管理者がユーザのプロパティを変更した場合、現在のセッションの間、ユーザにはその変更が見えない可能性があります。生存時間 (TTL) の設定に小さな値を指定することで、このような状況を緩和することができます。

クラスタ内の複数のサーバから変更済みのプロパティにすぐにアクセスできるようにする必要がある場合には、プロパティ キャッシングを無効にします。

Discount サービスのキャッシングを調節する

Order サービスと Shopping Cart サービスで注文と価格に関する情報 (割引など) の計算に要する時間を短縮するために、Discount サービスでは、キャッシング フレームワークを用いて以下のキャッシュの作成と管理を行います。

- discountCache: キャンペーン割引のデータが格納される。キャンペーン割引は、特定の顧客または顧客セグメントを対象としたもので、キャンペーン状況でのみ利用できます。
- globalDiscountCache: グローバル割引のデータが格納される。グローバル割引は、顧客プロパティまたは顧客セグメントに関係なく、すべての顧客に適用されます。

顧客がショッピング カートに商品を追加したり、ショッピング カートから商品を削除したり、チェック アウトを行ったり、あるいは注文を確認したりする際に、Pricing サービスは、カートに入っている商品の価格の決定を担当します。ショッピング カートに対する割引の影響を計算するために、Pricing サービスは、すべてのグローバル割引についての情報と、現在の顧客に適用されるキャンペーン割引についての情報を取得するよう、Discount サービスに要求します。

割引情報を求める最初のリクエストでは、適用される割引ごとにデータベースへの呼び出しを個別に行う必要があります。たとえば、グローバル割引が 1 つ定義されているほか、2 つのキャンペーン関連割引を受ける資格が顧客にある場合には、Discount サービスはデータベースへの呼び出しを 3 回行います。それ以降のリクエストに対する応答時間を短縮するために、Discount サービスではこれらのキャッシュを使用します。

discountCache を調節する

discountCache には、キャンペーン割引のデータが格納されます。最高のパフォーマンスを得るには、キャッシュの容量を、現在デプロイされているキャンペーン割引の数に設定します。容量を大きくすればするほど、メモリの消費量が増える可能性があります。

Discount サービスがこのキャッシュに情報を保持する時間 (ミリ秒単位) は、[生存時間] (TTL) プロパティによって決まります。キャッシュ内の値がタイムアウトしたあとでその値に対するリクエストを受け取ると、Discount サービスはデータベースを呼び出して情報を取得したあと、値をキャッシュに格納する必要があります。TTL を長くすると、キャッシュに格納されているオブジェクトが要求されたときに行われるデータベース呼び出しの回数が減ります。クラスタ環境では、TTL は、キャンペーン割引に対する変更がすべてのサーバ上で確実に利用可能になるまでに要する最大時間です。

globalDiscountCache を調節する

グローバル キャッシュの [最大エントリ数] プロパティの値を変更する必要はありません。

Discount サービスがこのグローバル割引用キャッシュに情報を保持する時間 (ミリ秒単位) は、[生存時間] プロパティによって決まります。その生存時間 (TTL) が経過したあとでグローバル割引情報に対するリクエストを受け取ると、Discount サービスはデータベースを呼び出して情報を取得したあと、値をキャッシュに格納する必要があります。TTL を長くすると、キャッシュに格納されているオブジェクトが要求されたときに行われるデータベース呼び出しの回数が減ります。クラスタ環境では、TTL は、グローバル割引に対する変更がすべてのサーバ上で確実に利用可能になるまでに要する最大時間です。

クラスタ環境および非クラスタ環境での Discount サービス用キャッシュ

どちらの環境 (クラスタ環境か非クラスタ環境) でも、割引の優先順位、終了日、あるいはアクティブ / 非アクティブ状態を変更すると、WebLogic Portal は該当するキャッシュから割引データをフラッシュします。キャンペーン割引を変

更した場合には、キャンペーン割引用キャッシュから特定の割引データだけがフラッシュされます。グローバル割引を変更した場合には、グローバル割引用キャッシュからすべての割引データがフラッシュされます。

たとえば、bread という名前の WebLogic Portal ホストにログインし、CampaignDiscount1 という名前のキャンペーン割引を無効にしたとしましょう。この場合、WebLogic Portal では、bread 上のキャンペーン割引用キャッシュから CampaignDiscount1 をフラッシュします。

クラスタ環境では、クラスタ内のほかのマシンは、キャッシュ内の割引データの TTL が経過するまでは、各マシン上のキャッシュに入っている割引データ (コピー) を引き続き使用します。

キャッシング レルムにおけるグループ メンバシップ TTL を調節する

WebLogic Server のキャッシング レルムには、成功した場合と失敗した場合の両方のレルム ルックアップの結果が格納されます。WebLogic Portal のキャッシング フレームワークは使用しません。

キャッシング レルムでは、ユーザ、グループ、パーミッション、ACL および認証の各リクエスト用に別個のキャッシュを管理します。ルックアップをキャッシュに格納し、それによって他のセキュリティ レルムに対する呼び出しの回数を減らすことで、WebLogic Server のパフォーマンスを向上させます。

WebLogic Portal では、キャッシング レルムはデフォルトで有効になっています。キャッシング レルム内のすべてのキャッシュがパフォーマンスの向上に役立つのに対して、特にグループ メンバシップ キャッシュの生存時間 (TTL) の値は、WebLogic Portal のパフォーマンスに影響を及ぼす可能性があります。

さらに、ユーザを先にグループから削除せずにシステムから削除すると、グループ メンバシップ キャッシュの TTL が経過するまでは、そのユーザは引き続きシステムから認識されることに注意してください。

グループ メンバシップ TTL の調節については、『*WebLogic Server 管理者ガイド*』(http://edocs.beasys.co.jp/e-docs/wls/docs70/adminguide/index.html) を参照してください。

JDBC におけるスレッド / 接続パラメータをチューニングする

ポータルで発生するパフォーマンス上のある種の問題は、config.xml 内のエントリを変更して、WLS 内のデフォルト実行キューのスレッド数を、commercePool に指定されている接続プール最大容量より少な目に設定することで、是正される場合があります。基本的には、接続プール内の接続の数がスレッド数 + 1 に等しくなるようにすべきです。

スレッドおよび接続プールの調節については、「JDBC 接続プールの最大サイズのチューニング」(http://edocs.beasys.co.jp/e-docs/wls/docs70/perform/WLSTuning.html#1117878) を参照してください。

第 12 章 パーソナライゼーションおよび対話管理のセットアップ

WebLogic Portal には、堅牢な認証機能とパーソナライゼーション機能が用意されており、それを利用することで、管理者は、どのようなコンテンツを訪問者が操作できるか、そしてその情報が特定の訪問者にどう表示されるかを決定することができます。また、訪問者が自ら WebLogic Portal のパーソナライゼーション機能を利用して、自分専用のコンテンツを選択したり、独自のルック アンド フィール（見た目と使い心地）を作成することもできます。Advisor、ルール フレームワーク、コンテンツ セレクタなどのツールを用いて、そうした認可とパーソナライゼーションを実現するリソースを作成することが、ポータル開発プロセスの主要な要素になります。

この章では、以下の内容について説明します。

- Advisor を用いたポータル アプリケーションのパーソナライズ
- ルール フレームワークの取り扱い
- コンテンツ セレクタを用いたパーソナライゼーション
- 編集 JSP を用いたポートレットのパーソナライズ
- プレースホルダを用いたポータルまたはポートレットのパーソナライズ

Advisor を用いたポータル アプリケーションのパーソナライズ

WebLogic Portal Advisor は、パーソナライズされたコンテンツ、ユーザのセグメント化、および基礎となるルール エンジンといったパーソナライゼーション サービスを利用するための、使いやすく柔軟性に富んだアクセス ポイントです。Advisor は、一連のルールとユーザ プロファイル情報に基づいて、パーソナライ

ズされたアプリケーションにコンテンツを配信します。また、ドキュメント管理システムから任意のタイプのコンテンツを取得し、JSP 内に表示することができます。

Advisor は、システム内のサービスとコンポーネントをすべて連携させ、パーソナライズされたコンテンツを配信します。Advisor コンポーネントとしては、WebLogic Portal の以下の中核的パーソナライゼーション サービスにアクセスする JSP タグ ライブラリと Advisor EJB（ステートレス セッション Bean）があります。

- ユーザ プロファイル管理
- ルール マネージャ
- コンテンツ管理
- パーソナライゼーション プラットフォーム

このタグ ライブラリとセッション Bean には、これらのサービスにアクセスしたり、パーソナライゼーション アクションの実行順序を決定したり、パーソナライズされたコンテンツをアプリケーションに返すためのパーソナライゼーション ロジックが組み込まれています。また、独自の Advisor プラグインを作成し、独自に作成した JSP タグを使ってそれらにアクセスすることも可能です。

このアーキテクチャによって、JSP 開発者は Advisor JSP タグを使ってパーソナライゼーション サービスを利用できるようになります。さらに Java 開発者は、Advisor Bean の公開インタフェースを通じて、基礎となる WebLogic Portal のパーソナライゼーション機能にアクセスすることができます。詳細については、『WebLogic Portal Javadoc API マニュアル』を参照してください。

Advisor の使い方は以下の 2 通りあります。

- **JSP タグを用いる。**典型的なページを作成する際には、JSP タグを用いるのが開発者にとっておそらく最も簡単でしょう。これらのタグを使用すれば、ユーザ分類に基づいてコンテンツの有効 / 無効を切り替えたり、静的クエリに基づいてコンテンツを返したり、さらにコンテンツ クエリを実行するルールに基づいてコンテンツをユーザに合わせることができます。これらのタスクを実行する JSP タグは、<pz:div>、<pz:contentSelector>、および <pz:contentQuery> です。
- **Advisor セッション Bean を用いる。**ページ制作者またはアプリケーション開発者は、必要であれば、タグの代わりに Advisor セッション Bean を直接使うこともできます。Advisor セッション Bean を使用すれば、ユーザ分類に

基づいてコンテンツの有効 / 無効を切り替えたり、静的クエリに基づいてコンテンツを返したり、さらにコンテンツ クエリを実行するルールに基づいてコンテンツをユーザに合わせることができます。

Advisor JSP タグを用いて、パーソナライズされたポータル アプリケーションを作成する

Advisor には、パーソナライズされたアプリケーションを開発者が作成するのに役立つ JSP タグが 3 つ用意されています。表 12-1 ではそれらについて説明します。これらのタグを用いれば、JSP から Advisor セッション Bean を利用することができ、開発者は Java ソース コードを書かなくても、パーソナライズされたデータを取得するページを作成できるようになります。

表 12-1 Advisor JSP タグ

タグ	解説
`<pz:div>`	ユーザから提供されるコンテンツの有効 / 無効を、分類ルールの実行結果に基づいて切り替える。分類ルールの結果が `true` の場合にはコンテンツが有効になり、`false` の場合にはコンテンツが無効になる。 ** システムでは、`<pz:div>` の開始タグと終了タグで囲まれたコンテンツを JSP コード内に挿入することで、コンテンツを有効にする。このコンテンツには、HTML タグ、他の JSP タグ、スクリプトレットなど、有効なあらゆる JSP コンテンツを組み込むことができる。分類ルールが `false` を返す場合には、システムは `<pz:div>` の開始タグと終了タグで囲まれたコンテンツを無視する。
`<pz:contentQuery>`	コンテンツ管理システム内のコンテンツを、コンテンツ属性に基づいて検索する。このタグは、開発者がさまざまな方法で扱える `Content` オブジェクトの配列を返す。

表 12-1 Advisor JSP タグ

タグ	解説
`<pz:contentSelector>`	コンテンツ セレクタ ルールの分類部にユーザが一致すれば、推奨コンテンツを返す。ユーザが条件に一致すると、パーソナライゼーション エンジンはルールに定義されているコンテンツ クエリを実行し、得られたコンテンツを JSP ページに返す。

パーソナライズされたアプリケーションを作成する手段としては、JSP タグを用いるだけでなく、Advisor Bean を直接扱うこともできます。Advisor Bean の使い方の詳細については、12-6 ページの「Advisor セッション Bean を用いて、パーソナライズされたアプリケーションを作成する」を参照してください。

<pz:div> JSP タグでユーザを分類する

`<pz:div>` タグは、ユーザから提供されるコンテンツの有効 / 無効を、分類ルールの実行結果に基づいて切り替えます。分類ルールの結果が `true` の場合にはコンテンツが有効になり、`false` の場合にはコンテンツが無効になります。

注意： ルールは、E-Business Control Center で作成します。E-Business Control Center を利用すれば、ユーザは独自の分類ルールを作成することができます。ユーザがルールの概念を知る機会はないため、E-Business Control Center では、分類ルールは「顧客セグメント」と呼ばれます。

コード リスト 12-1 では、`<pz:div>` タグを用いて *PremierCustomer* という分類ルールを実行する方法を示すとともに、主要顧客へのお知らせを HTML ページの出力に表示します。

コード リスト 12-1　<pz:dev> による分類ルールの実行

```
<%@ taglib URI="pz.tld" prefix="pz" %>
.
.
.
<pz:div
rule="PremierCustomer">
```

```
    <p>Please check out our new Premier Customer bonus program…</p>
</pz:div>
```

<pz:contentQuery> JSP タグを用いてコンテンツを選択する

<pz:contentQuery> タグを用いれば、コンテンツ管理システム内のコンテンツを、コンテンツ属性に基づいて検索することができます。このタグは、開発者がさまざまな方法で扱える Content オブジェクトの配列を返します。

コード リスト 12-2 に示すのは、コンテンツ管理システムに対してクエリを実行して、author 属性が「*Hemingway*」であるコンテンツをすべて検索したあと、見つかった Document のタイトルを表示する方法の例です。

コード リスト 12-2　CMS に対してクエリを実行して指定のコンテンツを見つける方法

```
<%@ page import="com.bea.p13n.content.ContentHelper"%>
<%@ taglib URI="pz.tld" prefix="pz" %>
.
.
.
<pz:contentQuery id="docs"
contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>"
query="author = 'Hemingway'" />

<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

<pz:contentSelector> JSP タグを用いてコンテンツをユーザに合わせる

<pz:contentSelector> タグは、コンテンツ セレクタ ルールの分類部にユーザが一致すれば、推奨コンテンツを返します。ルールの条件にユーザが一致すると、Advisor はそのルールに定義されているクエリを実行して、コンテンツを検索します。

コード リスト 12-3 に示す例では、主要顧客向けのコンテンツを Advisor に要求したあと、その結果としてドキュメント (Document) のタイトルを表示します。

コード リスト 12-3　Advisor に特定顧客向けのコンテンツを要求する方法

```
<%@ page import="com.bea.p13n.content.ContentHelper" %>
<%@ taglib URI="cm.tld" prefix="cm" %>
<%@ taglib URI="pz.tld" prefix="pz" %>
<%@ taglib URI="es.tld" prefix="es" %>
.
.
.
<pz:contentSelector id="docs"
    rule="PremierCustomerSpotlight"
    contentHome="<%=ContentHelper.DEF_DOCUMENT_MANAGER_HOME %>" />
<ul>
   <es:forEachInArray array="<%=docs%>" id="aDoc"
   type="com.bea.p13n.content.Content">
      <li>The document title is: <cm:printProperty id="aDoc"
      name="Title" encode="html" />
   </es:forEachInArray>
</ul>
```

Advisor セッション Bean を用いて、パーソナライズされたアプリケーションを作成する

Java 開発者は、一連の API を通じて Advisor Bean を直接操作して、パーソナライズされたアプリケーションを作成することができます。この方法は、JSP タグを用いて Advisor Bean を呼び出す方法の代わりとなるものです。

注意： このセッション Bean を用いてパーソナライズされたアプリケーションを作成する方法の詳細については、『WebLogic Portal Javadoc』を参照してください。

以下の手順は、アプリケーションで推奨コンテンツを Advisor から得る際に行われるプロセスの概要を示します。

1. Advisor セッション Bean のインスタンスをルックアップします。
2. AdvisorFactory の static メソッド `createAdviceRequest` を用いて AdviceRequest オブジェクトを作成します。

注意： このメソッドには、リクエストを表現する URI を渡す必要があります。Advisor では、その URI の接頭辞を用いて、どのアドバイズレットを呼び出すかを決定します。

3. AdviceRequest オブジェクトの必須属性と省略可能な属性を設定します。
4. Advisor の `getAdvice` メソッドを呼び出します。

   Advisor は最適なアドバイズレットを呼び出して、コンテンツに関するアドバイスを行います。そのアドバイズレットが推奨コンテンツを決定すると、その結果生成される `Advice` オブジェクトを Advisor がアプリケーションに返します。

   Advisor では、アドバイズレット レジストリを用いて、呼び出すべきアドバイズレットを選択します。
5. パーソナライズされたアプリケーションは、`Advice` オブジェクトから推奨コンテンツを取り出して、アプリケーションで使用します。

パーソナライズされたアプリケーションから Advisor にアドバイスを要求すると、Advisor Bean は、そのリクエストを処理できる登録済みアドバイズレットにそのリクエストを委託します。Advisor では、その URI 接頭辞を用いて、アドバイス リクエストを受信する登録済みアドバイズレットを決定します。選ばれたアドバイズレットはアドバイスを行い、`Advice` オブジェクトを Advisor に返します。この設計では、アドバイスに関するロジックがすべてアドバイズレットにカプセル化されているので、開発者は、より特化した目的に合うカスタム アドバイズレットを作成することができます。

属性オブジェクトはリクエストのパラメータの役目を果たします。属性オブジェクトは `AdviceRequest` オブジェクトに対して設定することができ、属性の名前を表す `String` オブジェクトに関連付けられます。

システムには、Classifier、ContentQuery、および ContentSelector の 3 つのアドバイズレットが用意されています。用意されているアドバイズレットに設定しなければならない属性の名前は、`AdviceRequestConstants` インタフェースに static String として定義されています。

アドバイス リクエストをアドバイズレットにマップする方法を決定するのに Advisor で用いられるロジックを表 12-2 に示します。

表 12-2 アドバイス リクエストからアドバイズレットへのマッピング

URI 接頭辞	対応するアドバイズレット
`classifier`	E-Business Control Center の顧客セグメント ツールを用いて作成されたルールを基に、ルールベース推論エンジンを用いてユーザを分類する。
`contentselector`	♦ ルールベース推論エンジンを用いて、ユーザを分類する。 ♦ ユーザが分類に一致するかどうかを決定する。 ♦ ルールベース推論エンジンを用いて、分類用のコンテンツ クエリを取得する。 ♦ 得られたコンテンツ クエリに基づいて、コンテンツを選択する。
`contentquery`	指定されたコンテンツ管理システム上で、コンテンツ属性検索を実行する。

以下の各節では、Advisor に直接アクセスして、JSP タグに用意されているのと同じ機能を実現する方法を示します。

Advisor セッション Bean を用いてユーザを分類する

分類に関する要求が JSP タグで対応できる範囲を越えている場合や、分類をサーブレットで用いる場合には、Advisor EJB を直接使用します。

Advisor に分類を要求するには、以下の手順に従います（API の詳細については、『Javadoc API マニュアル』を参照してください）。

注意： 特に断らないかぎり、ここで用いられるクラスはすべて、`com.bea.p13n.advisor` パッケージに入っています。

1. Advisor セッション Bean のインスタンスをルックアップし作成します。EJB Advisor ホーム インタフェースに定義されている `EJB_REF_NAME` 定数を、Advisor EJB ホームの JNDI 名として使ってもかまいません。

2. AdvisorFactory の static メソッド createAdviceRequest を用いて AdviceRequest オブジェクトを作成します。この場合には、URI 引数は “classifier://” にします。
3. AdviceRequest オブジェクトの必須属性を設定します（AdviceRequestConstants を参照）。それらの属性は以下のとおりです。
   - HTTP_REQUEST ― リクエスト オブジェクト（com.bea.p13n.httpRequest.createP13NRequest(HttpServletRequest) で取得される）
   - HTTP_SESSION ― セッション オブジェクト（com.bea.p13n.httpSession.createP13NSession(HttpServletRequest) で取得される）
   - USER ― ユーザ オブジェクト（com.bea.p13n.usermgmt.SessionHelper.getProfile(HttpServletRequest) で取得される）
   - TIME_INSTANT ― *現在*を表す java.sql.Timestamp オブジェクト
   - RULES_RULENAME_TO_FIRE （省略可能）― 起動すべきセグメント化ルールの名前
4. 新たに作成した AdviceRequest を指定して、Advisor の getAdvice メソッドを呼び出します。
5. Advisor から Advice のインスタンスが返されます。getResult メソッドを呼び出して、分類オブジェクトを取得します。分類オブジェクトが返された場合には、その分類は true とみなされます。戻り値が null の場合には、その分類は false とみなされます。

注意： AdviceRequest の省略可能パラメータ RULES_RULENAME_TO_FIRE を省略した場合には、そのユーザに関して複数の分類が返されることがあります。

Advisor セッション Bean を用いてコンテンツ管理システムにクエリを発行する

コンテンツ選択に関する要求が JSP タグで対応できる範囲を越えている場合や、コンテンツ選択をサーブレットで用いる場合には、開発者は Advisor EJB を直接使用することができます。

Advisor にコンテンツを要求するには、以下の手順に従います（API の詳細については、『Javadoc API マニュアル』を参照してください）。

注意： 特に断らないかぎり、ここで用いられるクラスはすべて、`com.bea.p13n.advisor` パッケージに入っています。

1. Advisor セッション Bean のインスタンスをルックアップし作成します。EJB Advisor ホーム インタフェースに定義されている `EJB_REF_NAME` 定数を、Advisor EJB ホームの JNDI 名として使ってもかまいません。
2. AdvisorFactory の static メソッド `createAdviceRequest` を用いて `AdviceRequest` オブジェクトを作成します。この場合には、URI 引数は “`contentquery://`” にします。
3. `AdviceRequest` オブジェクトの必須属性を設定します（`AdviceRequestConstants` を参照）。それらの属性は以下のとおりです。
   - `CONTENT_MANAGER_HOME` （必須）― コンテンツ マネージャ ホーム インタフェースを見つけるための JNDI 名。
   - `CONTENT_MANAGER` （省略可能）― 使用すべき ContentManager リモート インタフェースのインスタンス。これが設定されている場合には、`CONTENT_MANAGER_HOME` を設定する必要はない。
   - `CONTENT_QUERY_STRING` （必須）― システムに対して実行すべきクエリ。
   - `CONTENT_QUERY_SORT_BY` （省略可能）― 返される結果のソート順序。
   - `CONTENT_QUERY_MAX_ITEMS` （省略可能）― 返されるインスタンスの最大数。
   - `CONTENT_CONTEXT_PARAMS` （省略可能）― 生成される Search オブジェクトに入れて ContentManager に渡す名前 / 値ペアのマップ。
4. 新たに作成した `AdviceRequest` を指定して、Advisor の `getAdvise` メソッドを呼び出します。
5. Advisor から `Advice` のインスタンスが返されます。`getResult` メソッドを呼び出して、コンテンツ クエリの結果を表す `Content` オブジェクトの配列を取得します。

Advisor セッション Bean を用いてコンテンツをユーザに合わせる

コンテンツ選択に関する要求が JSP タグで対応できる範囲を越えている場合や、コンテンツ選択をサーブレットで用いる場合には、開発者は Advisor EJB を直接使用することができます。

Advisor にコンテンツを要求するには、以下の手順に従います（API の詳細については、『Javadoc API マニュアル』を参照してください）。

注意： 特に断らないかぎり、ここで用いられるクラスはすべて、`com.bea.p13n.advisor` パッケージに入っています。

1. Advisor セッション Bean のインスタンスをルックアップし作成します。EJB Advisor ホーム インタフェースに定義されている `EJB_REF_NAME` 定数を、Advisor EJB ホームの JNDI 名として使ってもかまいません。
2. AdvisorFactory の static メソッド `createAdviceRequest` を用いて `AdviceRequest` オブジェクトを作成します。この場合には、URI 引数は “`contentselector://`” にします。
3. `AdviceRequest` オブジェクトの必須属性を設定します（`AdviceRequestConstants` を参照）。それらの属性は以下のとおりです。
   - `HTTP_REQUEST` ― リクエスト オブジェクト（`com.bea.p13n.httpRequest.createP13NRequest(HttpServletRequest)` で取得される）。
   - `HTTP_SESSION` ― セッション オブジェクト（`com.bea.p13n.httpSession.createP13NSession(HttpServletRequest)` で取得される）。
   - `USER` ― ユーザ オブジェクト（`com.bea.p13n.usermgmt.SessionHelper.getProfile(HttpServletRequest)` で取得される）。
   - `TIME_INSTANT` ― *現在*を表す `java.sql.Timestamp` オブジェクト。
   - `RULES_RULENAME_TO_FIRE` （省略可能）― 起動すべきコンテンツ セレクタ ルールの名前。
   - `CONTENT_MANAGER_HOME` （必須）― コンテンツ マネージャ ホーム インタフェースを見つけるための JNDI 名。

- CONTENT_MANAGER （省略可能）― 使用すべき ContentManager リモート インタフェースのインスタンス。これが設定されている場合には、CONTENT_MANAGER_HOME を設定する必要はない。
- CONTENT_QUERY_STRING （必須）― システムに対して実行すべきクエリ。
- CONTENT_QUERY_SORT_BY （省略可能）― 返される結果のソート順序。
- CONTENT_QUERY_MAX_ITEMS （省略可能）― 返されるインスタンスの最大数。
- CONTENT_APPEND_QUERY_STRING （省略可能）― ルール エンジンで生成されるクエリに付加するクエリ。このクエリの先頭が「||」（縦棒が 2 つ）の場合には、ルールから生成されるクエリに対する OR（論理和）として扱われ、そうでない場合には、AND（論理積）として扱われる。
- CONTENT_CONTEXT_PARAMS （省略可能）― 生成される Search オブジェクトに入れて ContentManager に渡す名前 / 値ペアのマップ。

4. 新たに作成した AdviceRequest を指定して、Advisor の getAdvise メソッドを呼び出します。
5. Advisor から Advice のインスタンスが返されます。getResult メソッドを呼び 出して、推奨コンテンツを表す Content オブジェクトの配列を取得します。

ルール フレームワークの取り扱い

ルール管理は、ビジネス ルールを表現するための柔軟かつ強力なメカニズムを規定することで、パーソナライゼーション プロセスの主要な部分となります。これらのルールに組み込まれているビジネス ロジックのおかげで、それぞれのエンド ユーザ タイプ向けのパーソナライズされたコンテンツを確実に配信することが可能になります。

ルール フレームワークのさまざまなコンポーネントのコンフィグレーションは、rules.properties という外部コンフィグレーション ファイルを用いて行います。このファイルは、どの WebLogic Portal アプリケーションでも、そのルート ディレクトリ内の p13n_util.jar ファイル（com/bea/p13n/rules ディレクトリ内）に入っています。この節では、このファイルで設定できる各コンフィグレーション プロパティについて説明します。

rules.properties ファイルへの変更は、そのファイルを持つアプリケーションからのみ参照されます。すなわち、このコンフィグレーション ファイルのスコープはそのアプリケーションに限定されています。これによって、ルール フレームワークのコンフィグレーションをアプリケーションごとに変えることが可能になります。

ルール式を検証する

ルール エンジンにすべてのルール式（条件とアクションの両方）をちょうど 1 回だけ検証させたい場合には、rules.engine.expression.validation を true に設定します。開発時やテスト時には、コード リスト 12-4 に示すようにこのプロパティを true に設定して、新たに加えた式を検証することができます。

コード リスト 12-4 rules.engine.expression.validation プロパティの設定

```
##
# ルール エンジンによる式の検証：
#
# このプロパティが true に設定されている場合、
# ルール エンジンは、式が最初に実行されるときに
# それらを検証する。
##
rules.engine.expression.validation=true
```

ルール エンジンによるエラーの処理と報告

ルール エンジンの実行中にユーザに通知される例外のタイプは、rules.engine.throw.expression.exceptions プロパティと rules.engine.ignorable.exceptions プロパティによって決まります。

- 例外が通知されず、例外を発生させる条件式がすべて偽 (false) と評価されるようにするには、rules.engine.throw.expression.exceptions パラメータを false に設定する。
- すべての例外（rules.engine.ignorable.exceptions パラメータに列挙されているものを除く）をユーザに通知するには、

rules.engine.throw.expression.exceptions パラメータを true に設定する。

これらのパラメータの設定例をコード リスト 12-5 に示します。

コード リスト 12-5　ルール エンジンによるパターン式実行時のエラー処理

```
##
# ルール エンジンによるパターン式実行時のエラー処理：
#
# rules.engine.throw.expression.exceptions
#
# このプロパティが true に設定されている場合には、
# パターン式実行例外が送出される。それ以外の場合には、
# パターン式例外が発生するとパターン条件が偽（false）と
# 評価される。
#
# デフォルトは true。
#
# rules.engine.ignorable.exceptions （クラス名のリスト）
#
# 上記のプロパティが true に設定されていると、
# 列挙されたクラス以外のタイプの式例外および
# 組み込み例外が送出される。クラス タイプが何も
# 指定されていなければ、すべての式例外が送出される。
#
# デフォルトは、すべての例外クラス タイプ。
##
rules.engine.throw.expression.exceptions=true
rules.engine.ignorable.exceptions=java.lang.NullPointerException
```

コンテンツ セレクタを用いたパーソナライゼーション

コンテンツ セレクタは、コンテンツ管理システムからドキュメントを取得するためのメカニズムとして WebLogic Portal に用意されているものの 1 つです。コンテンツ セレクタを使用すれば、ドキュメントが WebLogic Portal で取得される際の条件を指定することで、ポータルやポートレットをパーソナライズすることができます。たとえば、日付の範囲、ユーザのステータス、ユーザの電子メール

アドレスなどの情報を指定することで、コンテンツ管理システム内のデータを表示するポートレットをパーソナライズすることができます。コンテンツ セレクタなら、選択条件に合うドキュメントだけを取得することになるでしょう。

注意： コンテンツ セレクタを扱うには、その前に顧客セグメントを作成しておく必要があります。セグメントの作成は管理作業であり、『管理者ガイド』で説明されています。詳細については、「顧客セグメントの作成」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/usrgrp.htm#1184110) を参照してください。

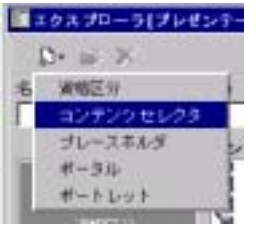
E-Business Control Center を用いて、コンテンツ セレクタを起動する条件を定義するとともに、コンテンツ セレクタでドキュメントの検索と取得に使用されるクエリを定義します。そのあと、コンテンツ セレクタ JSP タグとその他の一連の JSP タグを用いて、コンテンツ セレクタで絞り込まれたコンテンツを取得し表示します。

E-Business Control Center を用いて、コンテンツ セレクタを起動する条件と、クエリ条件を定義するには、以下の手順に従います。

1. E-Business Control Center を開き、[プレゼンテーション] タブを表示させます。
2. エクスプローラの左ペインの [**コンテンツ セレクタ**] アイコンをクリックします。

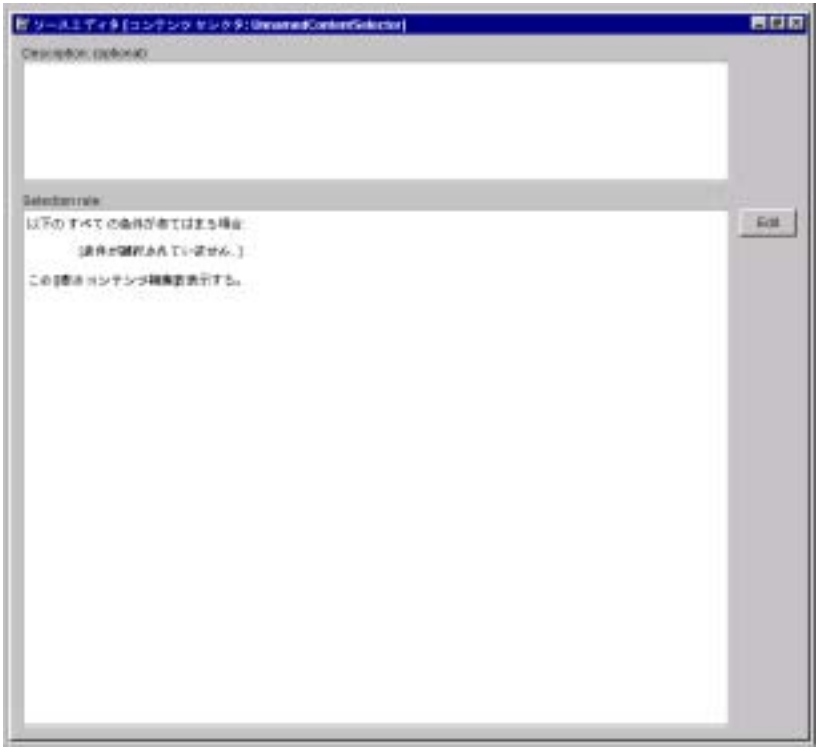
   エクスプローラの右ペインに、既存のコンテンツ セレクタがすべて表示されます。
3. [**新規作成**] を選択して [新規作成] メニューを表示し、[**コンテンツ セレクタ**] を選択します。

図 12-1 E-Business Control Center の [新規作成] メニューで [コンテンツ セレクタ] を選択



コンテンツ セレクタ エディタが表示されます。

図 12-2　コンテンツ セレクタ エディタ

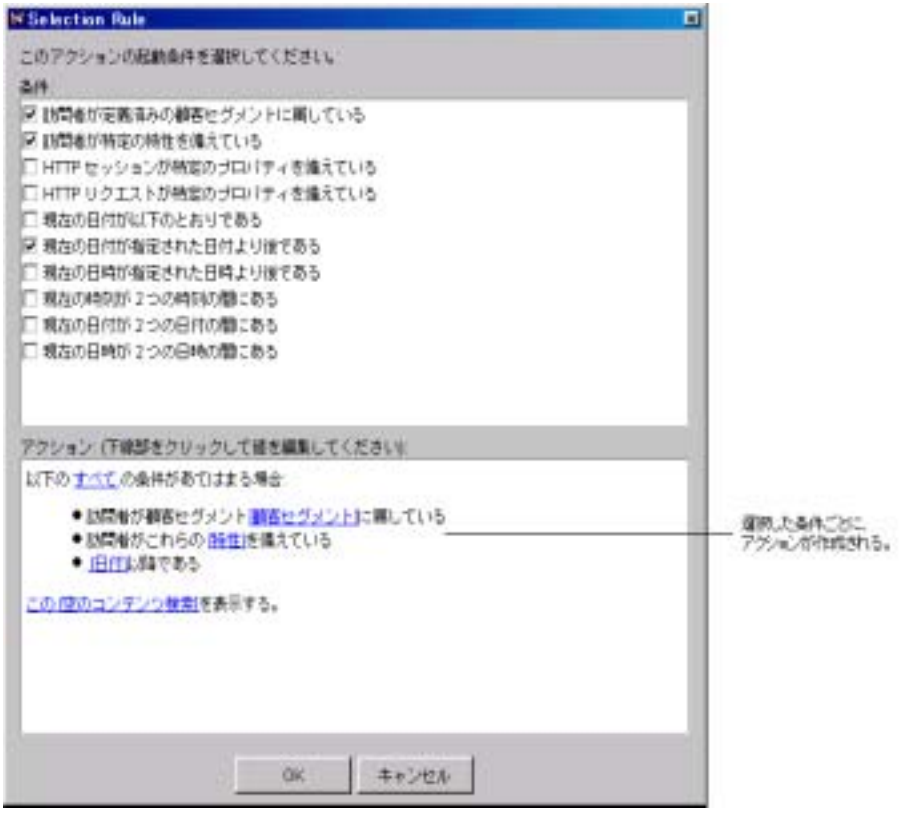


4. [**Selection rule**] ペインのどこかをダブルクリックして、以下のような [Selection Rule] エディタを開きます。

図 12-3　[Selection Rule] エディタ

5. コンテンツ セレクタ起動条件の横にあるチェックボックスをクリックします。条件を選択するたびに、それぞれに関係するアクションが [アクション] ペインに追加されます。

図 12-4　条件選択時の [Selection Rule] エディタ



6. [アクション] ペインで、以下を行います。

   a. 条件の適用方法を決定します。デフォルト値は [**すべて**] で、これは、すべての条件が満たされる場合にのみコンテンツ セレクタが起動されるという意味です。[**すべて**] というリンクをクリックして、値を [**いずれか**] に切り替えます。これは、少なくとも 1 つの条件が満たされればコンテンツ セレクタが起動されるという意味です。

   b. 次に、条件リスト内の下線付きのテキストをクリックして、条件ごとに値を設定します。たとえば、[**訪問者が定義済みの顧客セグメントに属している**] という条件を選択した場合には、[**訪問者が顧客セグメント [顧**

客セグメント] に属している] という条件が [**アクション**] ペインに表示されます。[**顧客セグメント**] をクリックして、[顧客セグメントの選択] ダイアログ ボックスを表示させます。

図 12-5 [顧客セグメントの選択] ダイアログ ボックス

c. 訪問者の所属先となる顧客セグメントを選択し、[**追加**] をクリックして [**選択済みセグメント**] リストに移動します。必要なセグメントをすべて追加したら、[**OK**] をクリックします。

d. 選択した条件ごとに、手順 b を繰り返します。

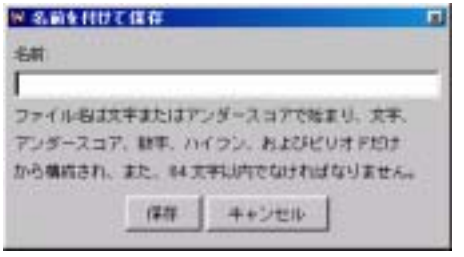
e. 選択したすべての条件の値が設定されたら、[**OK**] をクリックします。

[**Selection Rule**] ダイアログ ボックスが閉じます。

7. E-Business Control Center で、[**ファイル**] メニューを開き、[**名前を付けて保存**] を選択します。

[**名前を付けて保存**] ダイアログ ボックスが表示されます。

図 12-6 [名前を付けて保存] ダイアログ ボックス



8. コンテンツ セレクタの名前を [**名前**] に入力し、[**保存**] をクリックします。

   新しいコンテンツ セレクタが、エクスプローラの [コンテンツ セレクタ] リストに表示されます。

9. [**ツール**] メニューを開き、[**同期**] を選択します。

これで、新しいコンテンツ セレクタがいつでも使用できるようになりました。

与えられた JSP でコンテンツ セレクタ機能を使用するには、コンテンツ セレクタ JSP タグと一連の関連タグを呼び出す必要があります。詳細については、8-27 ページの「コンテンツ セレクタ タグおよび関連する JSP タグの使い方」を参照してください。

編集 JSP を用いたポートレットのパーソナライズ

訪問者は、必要なパーソナライゼーション属性や好みの設定を編集 JSP に記述することで、ポートレットをパーソナライズすることができます。編集 JSP とは、パーソナライゼーション データを収集し、それを用いて、要求されたデータをパーソナライズした形で表示する JSP のことです。

編集 JSP を用いて訪問者側からポートレットをパーソナライズできるようにするには、以下の 2 つのステップを実行します。

- ステップ 1: 編集 JSP を作成する
- ステップ 2: ポートレットの編集を有効にする

ステップ 1: 編集 JSP を作成する

他の JSP を作成するのと同じようにして、編集 JSP（Webflow が必要な場合には複数の JSP）を作成します。その名前は自由に付けることができ、ルック アンド フィールは、適切でさえあれば、どのようなものでもかまいません。この JSP の重要な機能は、パーソナライズされたコンテンツを訪問者側から入力したり検索できるようにする機能です。

たとえば、

- 株価ポートレットの編集 JSP であれば、訪問者が銘柄コードを入力できるテキスト ボックス（複数の場合あり）と、それぞれの銘柄の株価を検索するイベントを起動する [**OK**] ボタンまたは [**設定**] ボタンが付いているかもしれません。
- 電子メール ポートレットの編集 JSP であれば、サーバから電子メールを取得するためのユーザ名とパスワードを訪問者が設定する編集ボックスが付いているかもしれません。なお、このサーバも編集 JSP で指定されます。
- 列状に配列したデータを表示するポートレットの編集 JSP であれば、訪問者が列の有効 / 無効を切り替えたり、ソート設定を指定するのに使用できるチェックボックスが付いているかもしれません。

これでわかるように、訪問者パーソナライゼーションの要件を満たしているかぎり、編集 JSP に組み込むべき内容にはほとんど制限はありません。

ステップ 2: ポートレットの編集を有効にする

次に、訪問者側でポートレットを編集してパーソナライゼーション情報を追加できるようにする必要があります。それには、以下の手順に従います。

1. E-Business Control Center を起動します。E-Business Control Center の起動方法については、『管理者ガイド』の「E-Business Control Center の起動」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/admintro.htm#1185814) を参照してください。
2. エクスプローラ ウィンドウで、ウィンドウ下部の [**プレゼンテーション**] タブをクリックしたあと、[**ポートレット**] アイコンをクリックします。
3. エクスプローラのポートレット リストから、編集を有効にしたいポートレットをダブルクリックします。

   ポートレット エディタが表示されます。
4. 図 12-7 に示すように、[**編集を有効にする**] を選択します。

図 12-7　[編集を有効にする] チェックボックスが選択されているポートレット エディタ

5. このチェックボックスを選択することで、その下の編集ボックスも有効になることに注意してください。このボックスには、作成した編集 JSP の相対 URL を指定します。

ポートレット エディタの使い方の詳細については、『管理者ガイド』の「ポートレットの特性の変更」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/frmwork.htm#1199768) を参照してください。

プレースホルダを用いたポータルまたはポートレットのパーソナライズ

プレースホルダは、表示可能なコンテンツを定義する特定の条件が満たされたときにコンテンツが表示される、ポータルまたはポートレット内の領域を表す手段です。プレースホルダは、任意の数のクエリが組み込まれた名前付きエンティティです。プレースホルダ タグが含まれた JSP を訪問者が要求すると、プレースホルダは（通常、設定されたルールや顧客プロパティに基づいて）単一のクエリを選択して実行し、ブラウザでのクエリ結果の表示に必要になる HTML を生成します。

この節では、以下のトピックを扱います。

- プレースホルダの使い方
- プレースホルダ JSP タグ : <ph:placeholder>

プレースホルダの使い方

プレースホルダは、主にキャンペーンで使用され、訪問者の行動から訪問者が興味を持っていると判断される催しや商品などの情報に訪問者の注意を向けさせる働きをします。たとえば、オンラインのスポーツ用品店が、ある訪問者が特定のメーカーのルアー（釣りの擬似餌）を多数購入したことに気づいたとしましょう。あるルールが存在し、そこには、訪問者がそのメーカーの商品を一定額以上購入したときに、利用可能な割引についての情報をポータルに表示するよう指示してあります。その訪問者には、そのコンテンツのプレースホルダに指定された領域に、その情報が表示されることになります。また、別の訪問者が、キャンプ用具に興味を示しています（その訪問者が上記スポーツ用品店のカタログの「キャンプ」ページにアクセスした回数から判断される）。キャンプ好きの訪問者には、ルアーの割引ではなくキャンプ用具についての情報が、プレースホルダに指定された領域に表示されることになります。

上記の例は、情報を特定の訪問者向けにパーソナライズするための広告プレースホルダの使い方を示しています。同様に、広告以外のプレースホルダでも、同じレベルのパーソナライゼーションを提供することができます。たとえば、ある医師が製薬会社の Web サイトで薬品の調査を行っていて、その医師には特定の薬品群を調べる傾向があるとしましょう。その会社のポータルでは、その医師の行動を追跡して、医師がまだ調査していない関連薬品についての情報またはリンクをプレースホルダに表示することができます。

プレースホルダの作成は、JSP タグを使用し、E-Business Control Center でプレースホルダ クエリを定義することで行います。この節では、ポータルおよびポートレットのコンテンツをパーソナライズするためのプレースホルダを、これらの機能を用いてセットアップする方法を説明します。

プレースホルダ JSP タグ : <ph:placeholder>

プレースホルダ タグ <ph:placeholder> は、JSP 上にある、名前付きの位置です。このタグは、JSP へのプレースホルダを識別し、E-Business Control Center で設定された動作を記述するものです。E-Business Control Center を用いてプレースホルダの動作をセットアップする方法については、12-26 ページの「プレースホルダ ファイルを作成する」を参照してください。

注意 : 以下の表の「必須」列では、属性が必須と省略可能のどちらであるかを「はい」または「いいえ」で示します。「R/C」列では、「C」は属性がコンパイル (Compile) 時表現であることを、「R」は属性が要求 (Request) 時表現とコンパイル時表現のいずれかであることを示します。

<ph:placeholder> タグ（表 12-3）はプレースホルダを実装し、そのプレースホルダは、JSP ページ上のある位置での動作を記述します。

複数のプレースホルダ タグが同じプレースホルダを参照していてもかまいません。プレースホルダ タグの各インスタンスは、それぞれのプレースホルダ定義を個別に呼び出します。プレースホルダ定義で複数のクエリが指定されている場合には、たとえプレースホルダ タグの各インスタンスの定義が同じでも、インスタンスごとに異なる広告を表示することができます。

広告プレースホルダを含んだ JSP へのリクエストを WebLogic Portal が受信すると、プレースホルダ タグは AdService（ビジネス ロジックを呼び出して、表示すべき広告を決定するセッション EJB）に問い合わせます。

表 12-3 <ph:placeholder>

タグ属性	必須	データ型	解説	R/C
name	はい	String	プレースホルダ定義を参照する文字列。	R

表 12-3 <ph:placeholder>（続き）

タグ属性	必須	データ型	解説	R/C
height	いいえ	int	ブラウザでのドキュメント表示に必要な HTML を生成する際にプレースホルダで使用される高さをピクセル単位で指定する。 プレースホルダでこの値が用いられるのは、表示サイズに合ったコンテンツタイプの場合と、与えられたドキュメントのサイズが他の属性によってまだ定義されていない場合だけである。 この値を指定せず、他の属性が定義されていない場合には、ドキュメントの高さはブラウザの動作によって決まる。	R
width	いいえ	int	ブラウザでのドキュメント表示に必要な HTML を生成する際にプレースホルダで使用される幅をピクセル単位で指定する。 プレースホルダでこの値が用いられるのは、表示サイズに合ったコンテンツタイプの場合と、与えられたドキュメントのサイズが他の属性によってまだ定義されていない場合だけである。 この値を指定せず、他の属性が定義されていない場合には、ドキュメントの幅はブラウザの動作によって決まる。	R

例

この例では、MainPageBanner プレースホルダで指定された広告を表示します。

```
<%@ taglib uri="ph.tld" prefix="ph" %>
. . .
<ph:placeholder name="/placeholders/MainPageBanner.pla"/>
```

プレースホルダを実装する

プレースホルダを実装するには、以下の手順に従います。

1. プレースホルダの挿入先となる JSP テンプレート ファイルを開きます。JSP は個々のアプリケーションのポートレット フォルダ下のアプリケーション フォルダ内に入っています。たとえば、以下の場所です。

   ```
   <BEA_HOME>\weblogic700\samples\portal\<PORTAL_DOMAIN>\beaApps\
      <PORTAL_APP>\<PORTAL_APP>\portlets\campaigns
   ```

2. 以下のコードを JSP に追加することで、タグ ライブラリをインポートします。

   ```
   <%@ taglib uri="ph.tld" prefix="ph" %>
   ```

3. プレースホルダの設定先となる JSP 要素の中に <ph:placeholder> を追加します。コード リスト 12-6 に示すように、プレースホルダ タグ内に必ずプレースホルダ名を指定してください。

コード リスト 12-6　<ph:placeholder> タグ

```
<table class="homebackground" width="100%" height="100%"
   border="0" cellspacing="0" cellpadding="0">

   <tr>
      <td align="center">
         <ph:placeholder name="PrimaryCampaign"/>
      </td>
   </tr>

</table>
```

プレースホルダ ファイルを作成する

サイトの JSP に記述されているプレースホルダに合致するプレースホルダ ファイルを定義するには、E-Business Control Center を用います。以下の手順では、E-Business Control Center でプレースホルダ ファイルを作成する方法と、そのプレースホルダ内にデフォルト（キャンペーン以外）のクエリをセットアップする方法を示します。

注意： この手順を開始する前に、コンテンツ管理システム内のドキュメントの属性を定義しておく必要があります。

プレースホルダ ファイルを作成するには、以下の手順に従います。

1. WebLogic Server を起動し、E-Business Control Center を開きます。
2. 作業対象となるアプリケーションを開きます。
3. [**ファイル**｜**新規作成**｜**プレゼンテーション**｜**プレースホルダ**] を選択します。図 12-8 に示すように、新しいプレースホルダ ファイルが [エディタ] ウィンドウに開きます。

図 12-8　プレースホルダ エディタ

4. [概要] 領域にプレースホルダの説明を入力します。
5. [**新規作成**] をクリックして、デフォルト クエリの定義に取りかかります。デフォルト クエリが 1 つも含まれていなければ、プレースホルダ ファイルは不完全とみなされます（ただし、それでもプレースホルダ ファイルを保存することはできます）。

   注意： アクセスしようとしているコンテンツはサーバ上に格納されているため、[接続セットアップ] ウィンドウが表示されます。[表示名] フィールドに表示される既存の接続を選択し、ユーザ名とパスワード

を入力します。プレースホルダを扱う際には、ログインする必要があるのは、セッションごとに 1 回だけです。

このステップを繰り返すことで、デフォルト クエリを複数作成することができます。

デフォルト広告用の広告クエリを作成しない場合には、プレースホルダは、キャンペーン クエリで生成された広告しか表示しません。アクティブなキャンペーンが存在しない場合や、アクティブなキャンペーンのシナリオに、特定顧客向けの広告をトリガする広告アクションが含まれていない場合には、プレースホルダは顧客にとっては空のままです。

6. デフォルト クエリの優先順位を変更するには、そのクエリの [表示の優先順位] 列をクリックし、優先順位を選択します（図 12-8 を参照）。

   当該クエリがプレースホルダ内の他のすべてのクエリと比べて、実行される可能性がどの程度あるかは、[表示の優先順位] によって決まります。

7. キャンペーン広告クエリも含まれている場合に、広告プレースホルダでデフォルト クエリが使用されないようにするには、[**キャンペーンで出す広告を適用する場合には、デフォルト広告を表示しない**] オプションを選択します。

   プレースホルダにデフォルト広告クエリとキャンペーン広告クエリの中から選択させる場合には、[**キャンペーンで出す広告と一緒に、デフォルト広告もローテーションに入れておく**] オプションを選択します。これを選択すると、キャンペーンの一環として指定された広告がプレースホルダで表示される可能性は潜在的に少なくなります。

8. プレースホルダに名前を付けて保存します。使用するプレースホルダ名は、既存の名前にするか、これから JSP に記載する名前にしてください。

   プレースホルダのリストに、新しいファイルが表示されます。

第 13 章 Campaign サービスのセットアップ

キャンペーンでは、WebLogic Portal のいくつかのサービスを連携させて、e コマース Web サイトでのマーケティング目標の作成と追跡を行います。たとえば、マーケティング組織では、キャンペーンを利用して 6 月中に ACME 社ののこぎりを 100 個販売するといったことができます。この目標を達成するために、マーケティング部門では、一連の条件を満たす顧客（たとえば、以前 ACME 社の金物類をサイトから購入したことがある顧客など）を対象に、広告、電子メール、および商品価格の割引を実施することができます。

キャンペーン サービスのセットアップにおける開発者の責務は、キャンペーンを支援するインフラストラクチャを開発し、個々のキャンペーンの要件に合わせてそのインフラストラクチャに変更を加えることです。こうした作業としては、キャンペーン用プレースホルダの作成、表示およびクリックスルー動作の指定、コンテンツ管理システムへの広告のロード、パーソナライズされたキャンペーン用電子メールの作成、および見込み客と既存顧客へのメールの一括送信などがあります。

この章では、以下の内容について説明します。

注意： キャンペーンは、匿名ユーザに対しては使用できません。

Campaign サービスとは

キャンペーンでは、以下のサービスを連携利用します。

- イベントおよび行動追跡 (Events and Behavior Tracking) サービス：顧客とサイトのやり取りを識別します。デフォルトでは、顧客による特定の操作（イベント）の集合だけが WebLogic Portal で追跡されますが、Event サービスをカスタマイズすることで、この集合にイベントを追加することができます。イベントおよび行動追跡の詳細については、第 15 章「イベントおよび行動追跡」を参照してください。
- 顧客セグメント：顧客のプロファイルなどの動的データに記録された情報に基づいて顧客を分類します。顧客は、サイトにログインするために各自プロファイルを作成する必要があります。プロファイルには、顧客から入力される情報（届け先住所など）と、WebLogic Portal から提供される情報（サイトへの訪問回数やサイトで顧客が購入した商品の合計金額など）が記載されます。顧客セグメントは、E-Business Control Center で作成することができます。
- シナリオ：特定のイベントが発生した場合や特定の顧客が顧客セグメントに一致した場合に、アクションをトリガします。シナリオは、E-Business Control Center で作成することができます。

シナリオでは、以下のサービスをどれでも利用することができます。

- *広告プレースホルダ*：コンテンツ管理システムにクエリを発行して広告を検索し、クエリの結果を Web サイトに表示します。たとえば、顧客がログインし、顧客のプロファイルが顧客セグメント「SailingEnthusiast」（ヨット愛好家）に合致する場合には、シナリオによって、Web サイトのトップ バナーにヨットの広告が表示されることになります。
- *電子メール (E-mail) サービス*：JSP を用いて電子メールを生成するとともに、電子メールを一括送信するためのユーティリティを提供します。Mail サービスでは電子メールの生成に JSP が使用されるため、JSP タグを用いて電子メールをパーソナライズすることができます。
- **割引：**特定の商品または商品カテゴリの割引を提供します。割引は、E-Business Control Center で作成することができます。

キャンペーン用プレースホルダの作成

広告プレースホルダは、ブラウザでの広告コンテンツの表示に必要な HTML を生成し、それを JSP 内のプレースホルダ タグの位置に挿入する入れ物の働きをします。広告とは、コンテンツ管理システムに格納されているドキュメントのうち、広告プレースホルダによって表示されるものです。広告は、キャンペーンに不可欠な要素となることがあります。たとえば、キャンペーンでは、一定の広告クリックスルー数を記録することを目標として指定することができます。

属性を用いた表示およびクリックスルー動作の指定

以下の機能をサポートするために広告プレースホルダで用いられるドキュメント属性を、コンテンツ管理システム内に定義する必要があります。

- クエリの結果、複数のドキュメントが返される場合に、単一のドキュメントを選択する。
- 画像広告をクリック可能にする。
- Shockwave ファイルのムービー オプションを指定する。

ドキュメントへの属性の関連付けについては、コンテンツ管理システムのマニュアルを参照してください。BEA から提供される参考版のコンテンツ管理システムを使用する場合には、13-4 ページの「参考版のコンテンツ管理システムへの広告のロード」を参照してください。有効な属性の一覧を表 13-1、表 13-2、および 表 13-3 に示します。

コンテンツ管理システムへの広告のロード

広告プレースホルダに定義可能なクエリでは、コンテンツ管理システム内のドキュメントに関連付けられている属性が検索されます。WebLogic Portal では、広告を記述するのに用いられる属性群についての制限はありません。たとえば、ドキュメントで宣伝している商品の名前、広告スポンサーの名前、および e コマース商品カタログ内のカテゴリと一致する商品カテゴリを記述する属性を作成することができます。

コンテンツ管理システム (CMS) 内に広告をロードする方法については、CMS 側で指定されています。

- BEA から提供される参考版のコンテンツ管理システムを使用する場合には、この節で説明する手順に従います。
- サードパーティ製の CMS を使用する場合には、ベンダーから指示されるロード手順に従います。

参考版のコンテンツ管理システムへの広告のロード

WebLogic Portal には、コンテンツ管理ニーズが限られているサイト向けのコンコンテンツ管理システムがあらかじめ用意されています。この参考版のコンテンツ管理システムを使用する場合には、広告と広告属性を同時にロードする必要があります。既にロードされているドキュメントに属性を追加することはできません。

WebLogic Portal をインストールすると、参考版のコンテンツ管理システム（サンプル PointBase データベースを使用）には、一連のサンプル広告が格納されています。

参考版のコンテンツ管理システムに広告と広告属性をロードするには、以下の作業を行う必要があります。

- ステップ 1: HTML ドキュメントに属性をセットアップする
- ステップ 2: 画像および Shockwave ドキュメントの属性ファイルをセットアップする
- ステップ 3: dmsBase/Ads ディレクトリ ツリーにファイルを移動する

ステップ 1: HTML ドキュメントに属性をセットアップする

HTML だけで構成される広告の場合には、ドキュメントの <HEAD> 要素内の <META> タグにドキュメント属性を記述する必要があります。<META> タグでは、以下の構文に従います。

```
<META name="attribute-name" content="attribute-value">
```

ドキュメント属性ごとに別個の <META> タグを使用します。たとえば、次のように指定します。

```
<META name="attribute1-name" content="attribute1-value">
<META name="attribute2-name" content="attribute2-value">
<META name="attribute3-name" content="attribute3-value">
```

複数の属性を持つ簡単な広告が含まれた HTML ファイルの例を、コード リスト 13-1 に示します。

コード リスト 13-1　HTML 広告の属性

```
<HTML>
<HEAD>

<META name="adWeight" content="3">
<META name="productCategory" content="hardware">
<META name="productSubCategory" content="electic drill">
<META name="productName" content="Super Drill">
<META name="Manufacturer" content="ACME">

</HEAD>

<BODY>

<P>Buy our Super Drill. It'll get the job done!</P>

</BODY>

</HTML>
```

表 13-1 では adWeight 属性について説明しますが、これは、XHTML ドキュメント、画像、および Shockwave ドキュメントに関連付けることができる属性です。

表 13-1 すべてのドキュメント タイプに適用される属性

属性名	属性値のデータ型	解説および推奨事項
adWeight	Integer	クエリの結果、複数のドキュメントが返される場合にドキュメントを選択するのに使用される整数値を指定する。広告が選択される確率を高めるには、この属性値を大きくする。この属性のデフォルト値は 1。 **注意：** E-Business Control Center では、シナリオ アクションのクエリに「優先順位」を割り当てることができる。優先順位は adWeight 属性とは無関係で、プレースホルダでクエリが実行される確率を増減させるためのものである。adWeight 属性は、クエリが実行されたあとで広告を選択するために用いられる。

ステップ 2: 画像および Shockwave ドキュメントの属性ファイルをセットアップする

広告が画像または Shockwave ムービーの場合には、属性を別個のファイルに記述する必要があります。個々の画像ファイルまたは Shockwave ファイルには、以下の命名規則に従う名前のファイルが別途付いている必要があります。

filename.extension.md.properties

広告本体のファイルと付属ファイルは、どちらも同じディレクトリに存在しなければなりません。

たとえば、superDrill.jpg という画像ファイルの場合には、その属性を superDrill.jpg.md.properties というファイルに記述する必要があります。

filename.extension.md.properties ファイル内では、以下の構文に従って属性とその値を記述します。

attribute-name=attribute-value

画像広告の属性を記述したサンプル ファイルを、コード リスト 13-2 に示します。

コード リスト 13-2　属性ファイルの構文

```
adWeight=5
adTargetUrl=AcmeAds/saws.jpg
adAltText=Buy ACME and save!


productCategory=hardware
productSubCategory=electic drill
productName=Super Drill
Manufacturer=ACME
```

その他の画像ファイル属性

画像ファイルに関連付けることができる属性には、adWeight の他に、表 13-2 に示すものがあります。

表 13-2　画像ファイルの属性

属性名	属性値のデータ型	解説および推奨事項
adTargetUrl	String	画像をクリック可能にし、クリックスルーの遷移先を URL で指定する。クリックスルーは Event サービスによって記録される。 広告クリックスルーの遷移先の指定方法に応じて、adTargetUrl、adTargetContent、adMapName のいずれかの属性を使用する。
adTargetContent	String	画像をクリック可能にし、クリックスルーの遷移先をコンテンツ管理システムのコンテンツ ID で指定する。クリックスルーは Event サービスによって記録される。 広告クリックスルーの遷移先の指定方法に応じて、adTargetUrl、adTargetContent、adMapName のいずれかの属性を使用する。

表 13-2 画像ファイルの属性（続き）

属性名	属性値のデータ型	解説および推奨事項
adMapName	String	画像をクリック可能にし、イメージ マップを用いて任意の数の遷移先を指定する。 この属性の値は、以下の 2 箇所で使用される。 ♦ 画像をクリック可能にするアンカー タグ内： <img> ♦ マップ定義内：<map name=*value*> 広告クリックスルーの遷移先の指定方法に応じて、adTargetUrl、adTargetContent、adMapName のいずれかの属性を使用する。 adMapName の値を指定する場合には、adMap の値も指定する必要がある。
adMap	String	イメージ マップの XHTML 定義を指定する。 adMap の値を指定する場合には、adMapName の値も指定する必要がある。
adWinTarget	String	JavaScript を用いてポップアップを定義し、遷移先を新しいポップアップ ウィンドウに表示する。 この属性に指定できる値は newwindow だけである。
adWinClose	String	ポップアップ ウィンドウを閉じるリンクの名前を指定する。このリンクは、ウィンドウ コンテンツの最下部に表示される。 たとえば、この属性の値に「Close this window」を指定すると、ポップアップ ウィンドウの最後の行に「Close this window」というハイパーリンクが表示される。顧客がこのリンクをクリックすると、ウィンドウが閉じる。
adAltText	String	<img> タグの alt 属性のテキスト文字列を指定する。この属性を定義しない場合には、<img> タグの alt 属性は指定されない。
adBorder	Integer	<img> タグの border 属性の値を指定する。この属性を定義しない場合には、border 属性の値は "0" になる。

その他の Shockwave 属性

Shockwave ファイルに関連付けることができる属性には、adWeight の他に、表 13-3 に示すものがあります。広告プレースホルダと <ad:adTarget> タグでは、これらの値は <OBJECT> タグまたは <EMBED> タグの属性として書式化されます。Shockwave ファイルの表示には、Windows 上の Internet Explorer では <OBJECT> タグが用いられ、Netscape 互換プラグインをサポートするブラウザでは <EMBED> タグが用いられます。

これらの属性の詳細については、Shockwave 開発者用マニュアルを参照してください。

表 13-3 Shockwave ファイルの属性

属性名	属性値のデータ型	解説および推奨事項
swfLoop	String	ムービーを無限に繰り返す（true）か、最後のフレームに達した時点で停止する（false）かを指定する。 有効な値は true または false。この属性を定義しない場合には、true がデフォルト値になる。
swfQuality	String	表示する画像の品質を指定する。クライアント側のインターネット接続環境によっては、品質を下げることで再生が高速になる場合がある。 有効な値は low、high、autolow、autohigh、best のいずれか。
swfPlay	String	ムービーがブラウザにロードされた時点で直ちに再生を開始するかどうかを指定する。 有効な値は true か false。この属性を定義しない場合には、true がデフォルト値になる。
swfBGColor	String	ムービーの背景色を指定する。この属性は、HTML ページの背景色には影響を与えない。 値を指定するための有効な構文は *#RRGGBB*。
swfScale	String	HTML ページでムービー用に定義されている領域に対するムービーの相対的な大きさを指定する。 有効な値は showall、noborder、exact fit のいずれか。

表 13-3 Shockwave ファイルの属性（続き）

属性名	属性値のデータ型	解説および推奨事項
swfAlign	String	ブラウザ ウィンドウ内でのムービーの表示位置（中央、左、上、右、または下）を指定する。 値を指定しない場合、ムービーはブラウザ ウィンドウの中央に配置される。 有効な値は l、t、r、b のいずれか。
swfSAlign	String	ムービーの配置をブラウザ ウィンドウに対する相対位置で指定する。 有効な値は l、t、r、b、tl、tr、bl、br のいずれか。
swfBase	String	ムービーでの相対パス名の解決に用いられるディレクトリまたは URL を指定する。 有効な値は .（ピリオド）、*ディレクトリ名*、*URL* のいずれか。
swfMenu	String	ムービー プレーヤーに完全なメニューを表示するかどうかを指定する。 有効な値は true か false。

ステップ 3: dmsBase/Ads ディレクトリ ツリーにファイルを移動する

キャンペーンで広告を利用できるようにするには、HTML ファイル、画像ファイル、Shockwave ファイル、および属性ファイルをすべて、以下のパスの Ads ディレクトリ内に格納します。

<BEA_HOME>/user_projects/<YOUR-APPLICATIONDOMAIN>/dmsBase/Ads

ここで、<BEA_HOME> は BEA WebLogic Platform のインストール先ディレクトリ、<YOUR-APPLICATIONDOMAIN> は個々のドメインのディレクトリです。

ドキュメントを Ads ディレクトリのサブディレクトリに格納することもできますが、参考版のコンテンツ管理システムでは、ドキュメントの編成にサブディレクトリを使用しません。

サブディレクトリを用いてソース ファイルを管理する場合には、属性ファイルを、その記述対象である本体ファイルと同じディレクトリに格納する必要があります。たとえば、`superDrill.jpg` と `superDrill.jpg.md.properties` は同じディレクトリに存在しなければなりません。

ステップ 4: loadads スクリプトを実行する

`loadads` スクリプト（Windows の場合は `loadad.bat`、Unix の場合は `loadad.sh`）は、BulkLoader を実行して、`dmsBase/Ads` ディレクトリ内のドキュメントをコンテンツ管理システムにロードします。また、ドキュメントへの属性の関連付けも行います。

`loadads` を実行するには、以下のいずれかを実行します。

- コマンド ラインで（あるいは、Windows NT/2000 では [**スタート**｜**ファイル名を指定して実行**] を選択して）`loadads` と入力する。必ず、カレント ディレクトリを `<BEA_HOME>/user_projects/<YOUR-APPLICATIONDOMAIN>/dmsBase` ディレクトリにして実行すること。

または

- Windows エクスプローラを開き、`<BEA_HOME>/user_projects/<YOUR-APPLICATIONDOMAIN>/dmsBase` ディレクトリ内の `loadads` を探して、ファイル リスト内でダブルクリックする。

BulkLoader の実行方法の詳細については、8-1 ページの「Bulk Loader を用いたコンテンツの追加」を参照してください。

パーソナライズされたキャンペーン用電子メールの作成

電子メール サービスでは、JSP を用いて電子メールを生成するとともに、電子メールを一括送信するためのユーティリティを提供します。Mail サービスでは電子メールの生成に JSP が使用されるため、JSP タグを用いて電子メールをパーソナライズすることができます。この節では、パーソナライズされたキャンペーン用電子メールを以下の手順で作成する方法を示します。

ステップ 1: 電子メールのプロパティをコンフィグレーションする

キャンペーンで電子メールを送信するには、まず、Campaign サービスでメールの送受信に使用されるプロパティをコンフィグレーションしておく必要があります。クラスタ環境では、これらのプロパティは WebLogic Server によってクラスタ内の各ノードに伝達されます。

メール関係のプロパティをコンフィグレーションするには、以下を実行します。

1. E-Business Control Center で、アプリケーションを開きます。
2. E-Business Control Center の エクスプローラ ウィンドウで、[**サイト インフラストラクチャ**] タブをクリックします。
3. [**ユーザ プロファイル**] をクリックし、以下を確認します。
   - 顧客の電子メール アドレスを定義するプロパティ セットおよびプロパティの名前
   - キャンペーン関連電子メールの受信を希望するか否かに関する顧客の設定を記録するプロパティ セットおよびプロパティの名前。参照アプリケーションでは、この設定は Demographics プロパティ セットの `Email_Opt_In` プロパティに格納されます。

ステップ 2: ユーザ プロパティの名前を見つける

1. サーバを起動し、該当するドメインの WebLogic Server Administration Console にアクセスします。

2. WebLogic Server Administration Console の左ペインで、[**デプロイメント** | **アプリケーション** | **myApplication** | **Service Configuration** | **Campaign Service**] をクリックします。
3. [Campaign サービス] ページで、[**メール アクション**] タブをクリックします。
4. [**メール アクション**] タブで、以下の値を入力します。

表 13-4 [メール アクション] タブで指定する値

フィールド	入力する値
[**デフォルトの電子メール送信元アドレス**]	キャンペーンで送信する電子メールへの返信を受け取るデフォルトのアドレス。標準のメール ヘッダーでは、これは From アドレスである。 キャンペーン シナリオごとに専用の From アドレスを指定することができ、その指定はこのデフォルト プロパティよりも優先される。
[**電子メール アドレス プロパティ名**]	顧客の電子メール アドレスが格納されているプロパティの名前。
[**電子メール アドレス プロパティの所属先プロパティ セット名**]	顧客の電子メール プロパティの所属先プロパティ セットの名前。
[**電子メール オプトイン プロパティ名**]	顧客がキャンペーン関連の電子メールの受信を希望するかどうかを指定するプロパティの名前。参照アプリケーションでは、この設定は Demographics プロパティ セットの Email_Opt_In プロパティに格納される。
[**オプトイン プロパティの所属先プロパティ セット名**]	顧客のオプトイン（メール受信希望）プロパティの所属先プロパティ セットの名前。

5. [**適用**] をクリックします。

   以後、Campaign サービスで生成されるすべての電子メールには、これらの設定が適用されます。
6. 左ペインの [**Mail Service**] をクリックして、Mail サービスのデフォルト SMTP ホスト名をコンフィグレーションします。

注意： [Mail サービス] ページで行う変更は何であれ、WebLogic Portal を用いて送信されるすべての電子メール（Campaign サービスによって生成されるかどうかによらない）に影響を及ぼします。

ステップ 3: 電子メール JSP を作成する

Mail サービスでは、電子メールの内容および書式設定を JSP ファイルに記述する必要があります。この JSP では、WebLogic Portal における他の JSP で利用できる JSP タグおよび API をすべて使用することができます。

このステップでは、以下の事がらを説明します。

- 電子メール パラメータ
- セッション生成の無効化
- 電子メール JSP のサンプル
- 電子メール JSP の保存

電子メール パラメータ

シナリオ アクションは、電子メール JSP を要求する際に `userid` パラメータを渡して、そのシナリオ アクションをトリガした顧客のログイン名を指定します。`request.getParameter()` メソッドを使用すれば、ユーザ ID を取得し、それを電子メール JSP 内の JSP タグに渡すことができます。

さらに、シナリオは以下のパラメータも渡します（これらのパラメータを電子メール JSP 内の JSP タグに渡すこともできます）。

- `scenarioId`: 電子メールをトリガしたシナリオの ID を指定する
- `scenarioName`: 電子メールをトリガしたシナリオの名前を指定する
- `containerId`: シナリオの定義先キャンペーンの ID を指定する
- `containerName`: シナリオの定義先キャンペーンの名前を指定する

セッション生成の無効化

Campaign サービスで JSP から電子メールを生成するのに用いられる Java クラス `InternalRequestDispatcher` では、HTTPSession オブジェクトも生成します。通常は、電子メール JSP からこの HTTPSession オブジェクトを生成する必要はありません。HTTPSession オブジェクトは、顧客がサイトにアクセスしたときにアプリケーションによって生成済みだからです。

必要でない HTTPSession の生成を無効にするには、キャンペーン用電子メールの生成に用いる JSP の冒頭に以下のディレクティブを追加します。

```
<%@ page session="false" %>
```

このディレクティブの追加が必要なのは、顧客がサイトにアクセス（またはログイン）したときにアプリケーションによって HTTPSession オブジェクトが生成され、かつ、電子メールが InternalRequestDispatcher を通じて生成される場合だけです。

電子メール JSP のサンプル

サンプル Web アプリケーションの構成要素をなす電子メール JSP を、コード リスト 13-3 に示します。このファイルは、`<BEA_HOME>/weblogic700/samples/portal/wlcsDomain/beaApps/wlcsApp/wlcs/campaigns/emails/` 内にあります。

コード リスト 13-3　電子メール JSP のサンプル

```
Sample1.jsp:
<%@ page session="false" %>
<%@ page contentType="text/plain" %>

(このサンプル電子メールは、BEA Commerce Templates で新規ユーザ登録中にトリガされたサンプル キャンペーンの一環として自動的に送信されたものです。)

-------------------------------

拝啓 <%= request.getParameter("userId") %> 様

このたびは当サイトの会員にご登録いただき、誠にありがとうございます。ご登録後は、5,000 円以上のお買い物で 1,000 円の割引をご利用いただけます。

さらに、ご登録いただいたお客様には、以下のプレミアム サービスもご利用いただけます。
```

**「会員様限定」特別割引

** 新製品リリース情報をいち早くお届け

** お客様のご興味に合わせてカスタマイズされた各種サービス

ご登録ありがとうございました。

敬具

電子メール JSP の保存

E-Business Control Center ユーザがキャンペーン用電子メールを参照および選択できるように、電子メール JSP を Web アプリケーション内の特定のディレクトリに保存する必要があります。

デフォルトの保存先ディレクトリは `myApp/`*`myWebApp`*`/campaigns/emails` です。

（ここで、*myWebApp* は当該キャンペーンが定義されている Web アプリケーションの名前です。）

たとえば、Web アプリケーション `wlcs` のサンプル電子メール (`Sample1.jsp`) は、次のディレクトリに用意されています。

```
<BEA_HOME>/weblogic700/samples/portal/wlcsDomain/beaApps/wlcsApp/
   wlcs/campaigns/emails/
```

シナリオ アクションの一部としてこの電子メールを選択するには、以下を実行します。

1. E-Business Control Center で wlcsApp アプリケーションを開きます。
2. 電子メール アクションの作成時に、`<BEA_HOME>/weblogic700/samples/portal/wlcsDomain/beaApps/wlcsApp/wlcs/campaigns/emails` に保存されているすべての電子メール JSP をブラウズし、シナリオで用いる `Sample1.jsp` を選択します。

電子メール JSP の保存先のデフォルト位置を変更するには、以下を実行します。

1. WebLogic Server Administration Console の左ペインで、[**デプロイメント** | **アプリケーション** | **myApplication** | **Service Configuration** | **Campaign Service**] をクリックします。
2. [Campaign サービス] ページで、[**コンフィグレーション**] タブをクリックします。
3. [**電子メール ブラウズ開始ディレクトリ**] ボックスに、Web アプリケーションのルート ディレクトリからの相対パスでディレクトリ名を入力します。

メールの一括送信

コマンドを定期的に実行して、WebLogic Portal データ リポジトリに JSP によって格納されている電子メール バッチ（一括処理単位）を送信する必要があります。また、cron など、オペレーティング システムでサポートされている任意のスケジューラを利用して、メール送信コマンドを発行することもできます。

この節では、以下のトピックを扱います。

- リモート ホストから、あるいはクラスタ環境でメールを送信する
- 電子メールを一括送信する
- 電子メール一括配信をスケジューリングする
- 電子メール バッチを削除する

リモート ホストから、あるいはクラスタ環境でメールを送信する

メール送信ラッパー スクリプトでは、メール送信リクエストを処理する WebLogic Portal ホストの名前とリスン ポートを指定します。ラッパー スクリプトで指定されるホスト名およびリスン ポートは、デフォルトでは `localhost:7501` です。ただし、`localhost:7501` が有効なのは、シングル ノード環境内の WebLogic Portal ホストにログイン中にこのスクリプトを実行し、かつデフォルトのリスン ポートを変更していない場合だけです。

それ以外のコンフィグレーションからメール送信スクリプトを使用するには、以下のいずれかを行ってスクリプトを修正する必要があります。

- リモート ホストから実行できるようにメール送信スクリプトを修正する
- クラスタ環境で実行できるようにメール送信スクリプトを修正する

リモート ホストから実行できるようにメール送信スクリプトを修正する

メール送信スクリプトをリモート ホスト（すなわち、WebLogic Portal ホストでないコンピュータ）から実行する場合には、以下を実行します。

1. テキスト エディタで以下のファイルを開きます。

   `<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.bat`（Windows の場合）

   `<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.sh`（Unix の場合）

2. mailmanager スクリプトの `SET HOST=` 行で、localhost の部分を WebLogic Portal ホストの名前に置き換えます。
3. ホストで 7501 以外のリスン ポートを使用する場合には、`SET PORT=` 行の値 `7501` を、実際のリスン ポートの番号に置き換えます。
4. mailmanager スクリプトを保存します。

クラスタ環境で実行できるようにメール送信スクリプトを修正する

クラスタ環境で作業する場合には、クラスタ内のホストの名前を指定するようにメール送信ラッパー スクリプトを修正する必要があります。デフォルト値の localhost は、クラスタ環境で Mail サービスを利用する場合には無効です。

クラスタ環境でメール送信スクリプトを使用するには、スクリプトを実行する各ホスト上で以下を行います。

1. テキスト エディタで以下のファイルを開きます。

<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.bat（Windows の場合）

<BEA_HOME>\weblogic700\portal\bin\win32\mailmanager.sh （Unix の場合）

2. mailmanager スクリプトの SET HOST= 行で、localhost の部分を WebLogic Portal ホストの名前に置き換えます。電子メール メッセージを格納するデータ リポジトリにはクラスタ内の各ホストからアクセスできるため、クラスタ内の任意のホストの名前を指定することができます。
3. ホストで 7501 以外のリスン ポートを使用する場合には、SET PORT= 行の値 7501 を、実際のリスン ポートの番号に置き換えます。
4. mailmanager スクリプトを保存します。

電子メールを一括送信する

電子メールを一括送信するには、WebLogic Portal ホストにログインしているシェルから以下を実行します。

1. 以下のコマンドを入力して、データ リポジトリ内の電子メール バッチの名前と内容を調べます。

   mailmanager.bat *appName* list （Windows の場合）

   mailmanager.sh *appName* list （Unix の場合）

   ここで、*appName* は電子メール バッチを生成したエンタープライズ アプリケーションの名前です。コマンドの結果は標準出力に出力されます。シェル コマンドを用いて、出力をファイルにリダイレクトすることができます。

2. 電子メール バッチを送信後、データ リポジトリから削除するには、以下のコマンドを入力します。

   mailmanager.bat appName send-delete batch-name （Windows の場合）

   mailmanager.sh appName send-delete batch-name （Unix の場合）

電子メール一括配信をスケジューリングする

スケジューリング ユーティリティを利用して、データ リポジトリ内の電子メール バッチを送信することができます。mailmanager コマンドを用いてメールを送信する際にはバッチ名の指定が必要なため、キャンペーン シナリオごとにメール送信を個別にスケジューリングする必要があります。バッチの名前はシナリオのコンテナ ID に対応します。コンテナ ID については、13-14 ページの「電子メール パラメータ」を参照してください。

スケジューリング ユーティリティの使い方については、お使いのオペレーティング システムのマニュアルを参照してください。

電子メール バッチを削除する

「電子メールを一括送信する」で説明したように、電子メール バッチを送信と同時に削除することができます。それ以外にも、以下の方法で電子メール バッチを削除することができます。

1. 以下のコマンドを入力して、データ リポジトリ内の電子メール バッチの名前と内容を調べます。

   `mailmanager.bat` *`appName`* `list`（Windows の場合）

   `mailmanager.sh` *`appName`* `list`（Unix の場合）

   ここで、*appName* は電子メール バッチを生成したエンタープライズ アプリケーションの名前です。コマンドの結果は標準出力に出力されます。シェル コマンドを用いて、出力をファイルにリダイレクトすることができます。

2. 以下のコマンドを入力して、電子メール バッチを削除します。

   `mailmanager.bat` *`appName`* `delete batch-name`（Windows の場合）

   `mailmanager.sh` *`appName`* `delete batch-name`（Unix の場合）

第 14 章Commerce サービスのセットアップ

WebLogic Portal で利用できる重要なコマース サービスには、課税や支払などのビジネス トランザクション サービスと、プロダクト カタログ サービスがあります。これらのサービスに関連する開発作業としては、ローカルの課税サービスとサードパーティの課税サービスの統合、およびローカルの支払サービスとサードパーティの支払サービスの統合があります。また、プロダクト カタログ サービスについての開発作業には、カタログ データベースへのデータのロードと、カスタム カタログ サービスの作成および機能拡張があります。

この章では、以下の内容について説明します。

- ポータルとビジネス トランザクション サービスの統合
- プロダクト カタログのサポート

ポータルとビジネス トランザクション サービスの統合

WebLogic Portal は、課税サービスや支払サービスなどのビジネス トランザクション サービスと統合することができます。ポータルにこれらのサービスを付け加えると、外部サービスをポータルでローカルに利用できるようになるので、ポータルの機能が拡張されます。これらのサービスとの統合は開発任務の 1 つであり、そこでは、エンタープライズ アプリケーションに含まれている特定の EJB と、特定のコンフィグレーション ファイルに記述されている URL の更新が必要になります。ここでは、こうした修正について説明します。

この節では、以下のトピックを扱います。

- 課税サービスと統合する
  - サードパーティ ベンダが Web サービスのホストになる場合

課税サービスと統合する

WebLogic Portal にインストールされている Tax Web サービスは、デフォルトの `TaxCalculator EJB` から受け取った取引に関する税額計算を処理するためのデフォルトのフレームワークを提供します。ビジネス メソッドには、注文に対する課税を遂行するための標準ワークフローが実装されています （Tax Web サービスそのものは、Web サービス化するためのコードでステートレス セッション EJB をラップしたものです）。

エンタープライズ アプリケーションを Tax Web サービスと統合するには、`TaxCalculator EJB` か Tax Web サービスのどちらかを修正する必要があります。どちらを修正するかは、自らの組織とサードパーティ税額計算ベンダのどちらが Web サービスのホストになるかで決まります。

重要な注意：WebLogic Portal に付属しているデフォルトの Tax Web サービスでは、注文に対して自動的に 5% の課税を行います。こうしたデフォルトの税率適用は、実際の運用現場で用いるためのものではありません。サードパーティ ベンダの課税サービスと統合して、税額を適切に計算する必要があります。

サードパーティ ベンダが Web サービスのホストになる場合

サードパーティ ベンダが Tax Web サービスのホストになる場合には、そのベンダが、同 Web サービスをベンダ プログラムの API と統合し、Web サービス内部の `TaxWebService` EJB を修正します。この EJB の修正では、SOAP 呼び出し (エンタープライズ アプリケーションの `TaxCalculator` EJB から Web サービスに送信される SOAP 呼び出し) がベンダ API にとって解釈可能なメッセージに変換され、`TaxCalculator` EJB に返すための SOAP 呼び出しが適切に作成されるようにします。

ベンダをホストとする Web サービスに接続するには、以下の手順に従います。

1. Web サービスの tax.jar ファイル内の AVS*.class ファイルまたは Tax*.class ファイルがベンダによって変更されている場合には、変更されたファイルをエンタープライズ アプリケーションにコピーします。これらのクラスのソース コードは以下の場所にあります。

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

2. コマンド ラインから javac を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。

3. エンタープライズ アプリケーション内の TaxCalculator EJB の修正がベンダにとって必要な場合には、それを実行して、その EJB からベンダの TaxWebService EJB に適切な SOAP 呼び出しが行われるようにします。TaxCalculator EJB のソース コードは以下の場所にあります。

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

4. コマンド ラインから javac を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。

5. ソース コードのコンパイルが完了したら、生成されたクラス ファイルをエンタープライズ アプリケーション フォルダ内の wlcsSamples.jar に追加します。クラス ファイルを .jar ファイルに追加する際には、相対的なディレクトリ構造を変更しないようにしてください。

6. wlcsSample.jar ファイルに対して EJB コンパイラ (ejbc) を実行します。

7. アプリケーションの META-INF サブディレクトリ内の application-config.xml ファイルで、<TaxServiceClient> という要素を見つけ、ベンダのサーバ上の TaxWebService WSDL ファイルに接続するように TaxCalculatorWSDL 属性値の URL を変更します。

WebLogic Server は、起動時に application-config.xml ファイルを読み込むので、Web サービスの所在がわかります。

自らの組織が Web サービスのホストになる場合

自らの組織が Tax Web サービスのホストになる場合には、エンタープライズ アプリケーションが動作している JVM (Java Virtual Machine) とは別の JVM に Web サービスをデプロイします。こうすることで、Web サービスが停止して、その稼働元の JVM がフリーズするような事態になっても、エンタープライズ アプリケーションの JVM は稼働し続けることになります。

自らの組織をホストとする Tax Web サービスに接続するには、以下の手順に従います。

1. サードパーティ ベンダの税額計算ソフトウェアの API を入手します。
2. SOAP 呼び出しがサードパーティ ソフトウェア製品の API 記述言語に変換されるように、TaxWebService EJB (Web サービス内の EJB) を修正します。TaxWebService EJB のソース コードは、以下のディレクトリにあります。

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

3. コマンド ラインから javac を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。
4. ソース コードのコンパイルが完了したら、生成されたクラス ファイルを taxWSApp フォルダ内の tax.jar に追加します。クラス ファイルを .jar ファイルに追加する際には、相対的なディレクトリ構造を変更しないようにしてください。
5. tax.jar ファイルに対して Web サービス ジェネレータ (servicegen) を使用して、tax-webservice.war というファイルを作成します。

   servicegen の使い方については、『WebLogic Web サービス プログラマーズ ガイド』(http://edocs.beasys.co.jp/e-docs/wls/docs70/webserv/index.html) を参照してください。
6. TaxWebService EJB への適切な SOAP 呼び出しが行われるように、エンタープライズ アプリケーション内の TaxCalculator EJB を必要に応じて修正します。TaxCalculator EJB のソース コードは以下の場所にあります。

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\tax\
   ```

7. コマンド ラインから javac を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。

8. ソース コードのコンパイルが完了したら、生成されたクラス ファイルをエンタープライズ アプリケーション のルート フォルダ内の wlcsSamples.jar に追加します。クラス ファイルを .jar ファイルに追加する際には、相対的なディレクトリ構造を変更しないようにしてください。
9. wlcsSample.jar ファイルに対して EJB コンパイラ (ejbc) を実行します。
10. アプリケーションの META-INF サブディレクトリ内の application-config.xml ファイルで、<TaxServiceClient> という要素を見つけ、Web サービスのサーバ上の TaxWebService WSDL ファイルに接続するように TaxCalculatorWSDL 属性値の URL を変更します。

WebLogic Server は、起動時に application-config.xml ファイルを読み込むので、Web サービスの所在がわかります。

支払サービスと統合する

WebLogic Portal にインストールされている Credit Card Web サービスは、デフォルトの CreditCardService EJB から受け取ったクレジット カード取引の認可（承認)、売上計上、および決済を処理するためのデフォルトのフレームワークを提供します。ビジネス メソッドには、クレジット カード取引を遂行するための標準ワークフローが実装されています。取引の現在の状態が維持管理され、各アクションがログに記録されます （Credit Card Web サービスそのものは、Web サービス化するためのコードでステートレス セッション EJB をラップしたものです)。

エンタープライズ アプリケーションを Payment Web サービスと統合するには、CreditCardService EJB か Credit Card Web サービスのどちらかを修正する必要があります。どちらを修正するかは、自らの組織とサードパーティ支払処理ベンダのどちらが Web サービスのホストになるかで決まります。

重要な注意：WebLogic Portal に付属しているデフォルトの Payment Web サービスでは常に、あたかもサードパーティの支払サービスに接続しそこで承認を受けたかのように支払情報を送信し、エラーを発生しません。こうしたデフォルトの支払処理は、実際の運用現場で用いるためのものではありません。サードパーティ ベンダの支払サービスと統合して、支払処理を適切に行う必要があります。

サードパーティ ベンダが Web サービスのホストになる場合

サードパーティ ベンダが Credit Card Web サービスのホストになる場合には、そのベンダが、同 Web サービスをベンダ プログラムの API と統合し、Web サービス内部の `CreditCardWebService` EJB を修正します。この EJB の修正では、SOAP 呼び出し (エンタープライズ アプリケーションの CreditCardService EJB から Web サービスに送信される SOAP 呼び出し) がベンダ API にとって解釈可能なメッセージに変換され、CreditCardService EJB に返すための SOAP 呼び出しが適切に作成されるようにします。

ベンダをホストとする Credit Card Web サービスに接続するには、以下の手順に従います。

1. Web サービスの `payment.jar` ファイル内の `PS*.class` ファイルがベンダによって変更されている場合には、変更されたファイルをエンタープライズ アプリケーションにコピーします。これらのクラスのソース コードは以下の場所にあります。

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\payment
   ```

2. コマンド ラインから `javac` を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。

3. エンタープライズ アプリケーション内の `CreditCardService` EJB の修正がベンダにとって必要な場合には、それを実行して、その EJB からベンダの `CreditCardWebService` EJB に適切な SOAP 呼び出しが行われるようにします。`CreditCardService` EJB のソース コードは以下の場所にあります。

   ```
   <BEA_HOME>\Weblogic700\samples\portal\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\payment
   ```

4. コマンド ラインから `javac` を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。

5. ソース コードのコンパイルが完了したら、生成されたクラス ファイルをエンタープライズ アプリケーション フォルダ内の `wlcsSamples.jar` に追加します。クラス ファイルを `.jar` ファイルに追加する際には、相対的なディレクトリ構造を変更しないようにしてください。

6. `wlcsSample.jar` ファイルに対して EJB コンパイラ (`ejbc`) を実行します。

7. アプリケーションの META-INF サブディレクトリ内の `application-config.xml` ファイルで、`<PaymentServiceClient>` という要素を見つけ、ベンダのサーバ上の `CreditCardWebService WSDL` ファイルに接続するように `PaymentWebServiceWSDL` 属性値の URL を変更します。

WebLogic Server は、起動時に `application-config.xml` ファイルを読み込むので、Web サービスの所在がわかります。

セキュリティに関する重要な注意

Web サービスは SOAP（すなわち、XML over HTTP）に基づいているので、クレジット カード情報の受け渡しは Web を介してプレーン テキストで行われます。こうした情報は、盗聴中のハッカーに不正な目的で探知されるおそれがあります。そのため、クレジット カード処理サービス プロバイダへの専用線を設置し、やり取りされる HTTP 通信にセキュア ソケット レイヤ (SSL) を付け加えることを強くお勧めします。また、通信内容をすべて暗号化するクレジット カード プロバイダを選ぶことも必要です。

自らの組織が Web サービスのホストになる場合

自らの組織が Credit Card Web サービスのホストになる場合には、エンタープライズ アプリケーションが動作している JVM (Java Virtual Machine) とは別の JVM に Web サービスをデプロイすることを強くお勧めします。こうすることで、Web サービスが停止して、その稼働元の JVM がフリーズするような事態になっても、エンタープライズ アプリケーションの JVM は稼働し続けることになります。

自らの組織をホストとする Credit Card Web サービスに接続するには、以下の手順に従います。

1. サードパーティ ベンダの支払処理ソフトウェアの API を入手します。
2. SOAP 呼び出しがサードパーティ ソフトウェア製品の API 記述言語に変換されるように、`CreditCardWebService` EJB (Web サービス内部の EJB) を修正します。`CreditCardWebService` EJB のソース コードは、以下のディレクトリにあります。

```
<BEA_HOME>\Weblogic700\samples\samples\wlcsDomain\beaApps\
   wlcsApp\src\examples\wlcs\sampleapp\payment
```

3. コマンド ラインから javac を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。
4. ソース コードのコンパイルが完了したら、生成されたクラス ファイルを paymentWSApp フォルダ内の payment.jar に追加します。クラス ファイルを .jar ファイルに追加する際には、相対的なディレクトリ構造を変更しないようにしてください。
5. payment.jar ファイルに対して Web サービス ジェネレータ (servicegen) を使用して、payment-webservice.war というファイルを作成します。

   servicegen の使い方については、『WebLogic Web サービス プログラマーズ ガイド』(http://edocs.beasys.co.jp/e-docs/wls/docs70/webserv/index.html) を参照してください。
6. CreditCardWebService EJB への適切な SOAP 呼び出しが行われるように、エンタープライズ アプリケーション内の CreditCardService EJB を必要に応じて修正します。CreditCardService EJB のソース コードは以下の場所にあります。

   ```
   <BEA_HOME>\Weblogic700\samples\samples\wlcsDomain\beaApps\
      wlcsApp\src\examples\wlcs\sampleapp\payment
   ```
7. コマンド ラインから javac を実行するか、あるいは使用している Java エディタの指示に従って、上記ソース ファイルをコンパイルします。
8. ソース コードのコンパイルが完了したら、生成されたクラス ファイルをエンタープライズ アプリケーションのルート ディレクトリ内の wlcsSamples.jar に追加します。クラス ファイルを JAR に追加する際には、相対的なディレクトリ構造を変更しないようにしてください。
9. wlcsSample.jar ファイルに対して EJB コンパイラ (ejbc) を実行します。
10. アプリケーションの META-INF サブディレクトリ内の application-config.xml ファイルで、<PaymentServiceClient> という要素を見つけ、Web サービスのサーバ上の CreditCardWebService WSDL ファイルに接続するように PaymentWebServiceWSDL 属性値の URL を変更します。

WebLogic Server は、起動時に application-config.xml ファイルを読み込むので、Web サービスの所在がわかります。

Credit Card Web サービス EJB を修正する際のガイドライン

Payment サービス EJB は、クレジット カード取引の認可（承認）、売上計上、および決済に関係するサービスを提供するステートレス セッション Bean です。Credit Card Web サービス EJB は、さまざまな支払処理ソリューションとの統合を可能にするインタフェースの役目を果たします。各取引の現在の状態が維持管理され、各アクションがログに記録されます。取引の一般的特徴は以下のとおりです。

- 各取引は認可要求で開始されます。通常、この認可の結果、永続的な PaymentTransaction が生成されます。支払の状態とその PaymentTransaction のキーが、サービス固有の情報とともに、TransactionResponse に格納されて返されます。その PaymentTransaction のハンドルは、TransactionResponse から取得することができます。
- 支払認可サービスに接続できないために最初の認可が失敗した場合には、再認可メソッドを用いて認可を再度試みることができます。
- サービスのコンフィグレーションによっては、認可済み取引の売上計上または決済を行うことができます。

  法律によって、取引の売上計上を行えるのは、商品の発送が完了した場合に限られています。たとえば、ベンダからオンラインで書籍を購入し、その発送までに 2 日かかる場合には、ベンダは取引の認可だけはできますが、売上計上はできません。2 日後、商品が発送されたら、ベンダはその取引の売上計上を行うことができます。しかし、ソフトウェアをオンラインで購入し直ちにダウンロードするような場合には、ベンダはその場で取引の認可と売上計上を行うことができます。注文と実際の発送（ダウンロード）が数秒しかずれていないからです。
- 個々の取引全体を単一の AuthorizeAndCapture で完了することができます。
- トラフィックの大きいサイトでは、認可や売上計上の処理をオフラインで実行すべきです。この処理には通常 3 ～ 8 秒かかり、何千人ものユーザがいる場合には、サイトのパフォーマンスを低下させることにもなりかねません。サイトで認可や売上計上の処理をオフラインで実行する場合には、その処理に別個のマシンを使用して、処理の整合性を確保しなければなりません。

プロダクト カタログのサポート

この節では、プロダクト カタログのサポートに関連する開発作業について説明します。以下で取り上げる作業の中には、BEA から提供されるリソースを用いて作成されたプロダクト カタログに当てはまるものもあれば、適切なステートレス セッション EJB サービス API を実装することで作成できるカスタム カタログ サービスに当てはまるものもあります。前者については、『管理者ガイド』の「カタログの作成と管理」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/commerce.htm#1167188) を参照してください。さらに、JSP を用いてカタログを表示する方法と、カタログ サービスとカタログ キャッシュを統合する方法についても取り上げます。

この節では、以下のトピックを扱います。

- プロダクト カタログ データベース スキーマへの商品データのロード
- JSP を用いたカタログの表示
- ショッピング カートとカタログを接続する
- サービスとカタログ キャッシュを統合する

プロダクト カタログ データベース スキーマへの商品データのロード

最も重要な開発作業は、商品に関する情報を WebLogic にとって解釈可能な形式に変換することです。それには、DBLoader プログラムを使用します。DBLoader を使用すれば、任意のデータをデータベース内の任意のテーブルに一度にロードすることができます。

DBLoader を用いてプロダクト カタログを作成するには、以下の手順に従う必要があります。

- ステップ 1: DBLoader の使用準備をする
- ステップ 2: databaseload.properties ファイルを編集する
- ステップ 3: DBLoader プログラムを実行してデータをロードする

ステップ 1: DBLoader の使用準備をする

商品情報をデータベースに追加するには、まず、データベースに関する以下の問題点について検討し、入力ファイルを用意する必要があります。

データベースに関する重要事項を確認する

DBLoader プログラムを使用する際には、以下に示すように、データベースに関して念頭に置いておかなければならない重要なポイントがいくつかあります。

参照整合性と制約 : プロダクト カタログのスキーマでは、制約を用いて、テーブル間のデータ整合性および参照整合性を実現します。たとえば、`WLCS_PRODUCT` と `WLCS_CATEGORY` についての主キー制約や、`WLCS_PRODUCT_CATEGORY` についての外部キー制約などです。

主キーとユニークなインデックスがあれば、テーブルにエントリが重複して格納されるおそれはなくなります。外部キー制約では、親キーがすでに存在することを確認してからデータベースへの子レコードの書き込みを許可することで、参照整合性が確保されます。

注意 : `WLCS_PRODUCT` テーブルと `WLCS_CATEGORY` テーブルのどのエントリについても、対応するエントリを `CATALOG_ENTITY` テーブルにも作成する必要があります。

Java における String の取り扱い : Java では、すべての String が Unicode 2.0 文字の列として表されます (Unicode 2.0 は、世界中の主要な言語をサポートする 16 ビット文字エンコーディングです)。したがって、JVM (Java Virtual Machine) にテキストを読み込む際と、JVM からテキストを書き出す際には、エンコーディング体系を用いて、オペレーティング システムで使用される「ネイティブ」エンコーディングと Unicode 2.0 の間の変換を行う必要があります。テキスト ファイルに記述されたデータは、そのエンコーディングが JVM（およびオペレーティング システム）のデフォルトのファイル エンコーディングに一致する際には、Unicode 2.0 に自動的に変換されます。

プロダクト カタログ スキーマにロードする商品情報入力ファイルを用意する

この節で示すルールに従って、Web サイトで使用する商品情報を記載した入力ファイル（テキスト ファイル）を作成します。データベース テーブルごとに、別個のファイルを作成します。

入力ファイルのファイル構造の検証：入力データ ファイルは、デフォルトでは、パイプで値を区切ったテキスト ファイルです。この入力ファイルの構造は以下のとおりです。

- 第 1 行 : テーブル名が記載されたヘッダー
- 第 2 行 : そのテーブルの各カラム名
- 第 3 行 : 第 2 行に列挙したカラムのデータ型
- 第 4 ～*第 N* 行 : 入力データ

表 14-1 に入力ファイルの構造を示します。

表 14-1　入力ファイルの構造

行	内容
第 1 行	ファイルのヘッダーでは以下を指定する必要がある。 ♦ ロードするレコード数。DBLoader では、この数値を参考までに使用するだけである。この数値にかかわらず、ファイル内のすべてのレコードが処理される。 ♦ データのロード先となるデータベース テーブルの名前。
第 2 行	第 2 行では、データのロード先となるテーブル カラムの名前を指定する。入力ファイルには、主キー カラム（複数の場合あり）がなければならない。それぞれの主キー カラム名の先頭にはアスタリスク (*) を付ける。テーブル内の主キー カラム以外のカラムは、すべて NULL がデフォルト値になる。したがって、NULL が値として許容されるカラムについては、カラム名を省略でき、NULL 以外の値を持つカラムだけを指定する。
第 3 行	第 3 行では、ロード対象となる各カラムのデータ型を指定する。
第 4 ～ 第 N 行	入力データ ファイル内の以降の行はすべてデータ値である。

コード リスト 14-1 に、簡単な入力ファイルの例を示します。

コード リスト 14-1　簡単な入力ファイル

```
3|WLCS_PRODUCT
*SKU|NAME|IN_STOCK|EST_SHIP_TIME|SPECIAL_NOTES|CREATION_DATE
   VARCHAR|VARCHAR|VARCHAR|VARCHAR|VARCHAR|VARCHAR|DATE
P123|CoolKid|N|Out of stock until further notice|Special order
   only|02-Oct-2000
P124|FastKid|Y|One week|No special order|02-Oct-2000
P125|RadSneakers|Y||regular stock|02-Oct-2000
```

注意： 簡単な入力ファイルは以下の場所でも参照できます。
`<PORTAL_HOME>\db\data\sample\wlcs\hardware\PRODUCT.dat`

空の入力文字列：データ ファイル内に空の入力文字列があると、データベースには空の文字列として挿入されます。値を指定しないカラムが入力レコード内に存在する場合には、その対応する位置（第 2 行で列挙したカラムの位置に一致）に区切り文字（デフォルトでは、カンマ）を入れて、そのことを明示する必要があります。たとえば、以下のように指定します。

```
P125|RadSneakers|Y||regular stock|02-Oct-2000
```

非主キー フィールドの未指定値：上記の例では、第 4 カラム (`EST_SHIP_TIME`) の値が指定されていません。このカラムはデータベース レコードの主キーではないので、これでも問題ありません。このカラムの値は、空の文字列として格納されます。

注意： データベース内で主キー以外のカラムにヌル値を格納したい場合には、該当するレコード内のそのカラムの位置に `NULL` と入力しなければなりません。ただし、`NULL` を引用符で囲んではいけません。引用符で囲むと、カラムは文字列として格納されることになるからです。

ステップ 2: databaseload.properties ファイルを編集する

DBLoader では、`databaseload.properties` ファイルに記載された情報を用いて、データとロード処理についての情報 (たとえば、どのようなドライバ、データベース、あるいはログインを使用するかといったこと) を決定します。`<PORTAL_HOME>\db` ディレクトリにある `databaseload.properties` ファイルを

開きます。使いたいデータベース用の行についてコメントを外し、正しい設定を入力します。Pointbase 以外のデータベースを使用する場合は、必ず Pointbase の行をコメントアウトします。

databaseload.properties ファイルで、先頭に # 記号の付いた行はすべてコメント行です。ファイルには、コメント行と空白行のどちらも含めることができます。

表 14-2　databaseload.properties ファイルで設定可能な値

プロパティ名	デフォルト値	解説
jdbcdriver	databaseload.properties 参照。	データベースへの接続に使用する JDBC ドライバを指定する。デフォルトのドライバは、WebLogic に付属の Pointbase JDBC ドライバ。
connection	databaseload.properties 参照。	使用するデータベースに接続するためのドライバ用に必要なデータベース接続文字列。
dblogin	databaseload.properties 参照。	データベース ユーザ名。ログイン名は操作対象のテーブルに対する読み込み / 書き込み特権を持っていなければならない。
dbpassword	databaseload.properties 参照。	データベース ユーザのパスワード。
delimiter	|	入力データ ファイル内で値を区切るのに用いられる認識可能な区切り文字。これは変更可能。その場合には、区切り記号として別の文字、たとえばアクセント記号 (^) などを選ぶ。
dateformat	dateformat=mm-YY-dd	入力データ内の日付カラムで使われる書式を指定する。日付の書式はロケール固有である。databaseload.properties ファイル内の他の書式はコメントアウトする。

プロパティ名	デフォルト値	解説
timestamptable	WLCS_CATEGORY, WLCS_PRODUCT	DBLoader による更新追跡の対象となるデータベース テーブルを指定する。カラム名は Commerce サービスから提供されるスキーマで固定されている。ただし、他のテーブル（各 WLCS テーブル以外）に対して DBLoader を使用する場合には、その他の独自カラム名を指定できる。
timestampfield	MODIFIED_DATE	WLCS_CATEGORY テーブルと WLCS_PRODUCT テーブルにおいて当該レコードの最新変更日時を示すカラムを指定する。DBLoader では、timestamptable プロパティで指定されたテーブル内の各レコードに対する最新の更新日時を判断する際に、このカラムの値を用いる。カラム名は、Commerce サービスから提供されるスキーマで固定されている。ただし、他のテーブル（各 WLCS テーブル以外）に対して DBLoader を使用する場合には、その他の独自カラム名を指定できる。
commitTxn	50	何件のレコードをロードしてから更新をデータベースにコミットするかを設定する。この値が 1 以下の場合には、DBLoader は各レコードのロードが完了するたびにコミットする。
encoding	databaseload.properties ファイルには指定されない。そのため、デフォルト値は Java 2 SDK のプラットフォーム デフォルト値。	マルチバイト文字のエンコーディング タイプを設定する。指定可能なプロパティ値は UCS2 または UTF8。 データベースに対してデータの書き込みや読み込みを行う際には、システムで使用されるネイティブ文字エンコーディングと Unicode 2.0 は、Java によって透過的に変換される。これに対してユーザは、特に何もする必要はない。 ただし、システムのネイティブ エンコーディングとは異なるエンコーディングのデータベースに対してデータの書き込み / 読み込みを行う必要がある場合には、明示的に変換を行う必要がある。

ステップ 3: DBLoader プログラムを実行してデータをロードする

これで、入力ファイルの作成と databaseload.properties ファイルのセットアップが完了したので、databaseload スクリプトを実行して DBLoader プログラムを起動します。

- databaseload の基本事項を確認する
- 非 Windows 環境 – スクリプトの実行準備をする
- databaseload スクリプトを実行する

databaseload の基本事項を確認する

スクリプトは以下の場所にあります。

- <PORTAL_HOME>\db (Windows の場合)
- <PORTAL_HOME>/db (UNIX の場合)

注意： PORTAL_HOME は、WebLogic Portal のインストール先ディレクトリです。

databaseload スクリプトでは、以下のタスクを実行します。

- プログラム実行時の環境をコンフィグレーションする
- データ入力ファイルの位置を指定する
- DBLoader プログラムを起動する

非 Windows 環境 – スクリプトの実行準備をする

非 Windows 環境の場合、databaseload スクリプトを実行するには、その前にまず、databaseload.properties ファイルで指定しているのと同じデータベースを set-environment スクリプトにも指定していることを確認しておきます。set-environment スクリプトは、databaseload スクリプトと同じディレクトリに入っています。たとえば、databaseload.properties ファイルに ‘jdbc:pointbase:server://localhost:9092/wlportal’ 接続が指定されている場合には、set-environment スクリプト側では SET DATABASE=POINTBASE となっていなければなりません。

すでに述べたように、DBLoader の動作は WebLogic Portal サーバとは無関係です。したがって、ローダを実行しようとする場合にサーバを停止する必要はありません。

WebLogic Portal サーバを Oracle と連携して実行する場合には、データがデータベースにロードされる間はパフォーマンスが低下することがあります。

注意： DBLoader を実行する前に、更新対象のテーブルをバックアップしたほうがよいでしょう。DBLoader プログラムでは、データベース内に履歴レコードを保存しません。

databaseload スクリプトを実行する

スクリプトを実行するコマンドのフォーマットは、以下のとおりです。

```
>> databaseload { -insert | -update | -delete } input-file.dat
```

たとえば、以下のように入力します。

```
>> databaseload -update product_categories.dat
```

上の例では、DBLoader プログラムは、プロダクト カタログ データベース内の行のうち、`category.dat` 入力ファイルに指定されている主キーに一致するものを更新します。

操作のタイプの選択： 実行可能な 3 種類の操作、すなわち `-insert`、`-update`、または `-delete` の中から 1 つを選択する必要があります。

UNIX と特権： UNIX システムでは、`databaseload.sh` ファイルのデフォルトの保護設定に実行特権が含まれている必要があります。その設定には、通常、以下のコマンドを用います。

```
$ chmod +x databaseload.sh
```

複数のテーブルへのデータのロード： 複数テーブル内のデータの挿入、更新、削除を行うには、テーブルごとに、対応する入力ファイル名をパラメータとして指定して、`databaseload` スクリプトを別々に実行します。更新されるテーブルの順序については、他のすべての SQL 文の場合と同じデータ整合性ルールに従わなければなりません。たとえば、主キー制約のある親テーブルに行を挿入してから、外部キー制約のある子テーブルに行を挿入します。

ステップ 4: DBLoader ログ ファイルを用いてトラブルシューティングする

以下の 2 つの監査トレイル ログ ファイルを用いて、DBLoader 操作の実行時に発生したエラーなどの問題を明らかにすることができます。

- `dbloader.log`
- `dbloader.err`

この節では、以下のトピックを扱います。

ログ ファイルの検査のタイミングを決定する

各操作のあとすぐにログ ファイルを調べる必要があります。DBLoader の各操作によってログ ファイルが上書きされるからです。どちらのファイルも、`databaseload` スクリプトの実行時と同じディレクトリに作成されます。

dbloader.log ファイルを検査する

`dbloader.log` ファイルには、以下の情報が含まれています。

- 入力ファイル名と、実行されたアクション （挿入、更新、削除のいずれか）
- ロード操作時に処理されたレコード数
- データベース ロード処理の開始時刻と終了時刻

dbloader.err ファイルを検査する

データベースへのロード操作を実行中にエラーが発生した場合には、`dbloader.err` ファイルに以下の情報が書き込まれます。

- 入力ファイル名と、実行されたアクション （挿入、更新、削除のいずれか）
- レコードに関して障害または例外が発生したときのタイムスタンプ

- 入力ファイル内のデータ レコードのうち、ロードに失敗したレコードのインデックス
- 障害または例外の理由と、入力レコードの実際の値

DBLoader プログラムでは、ロードの影響を受けるカラムの数（入力データ ファイルの第 2 行に指定されている）と、各レコード内の入力カラムの数を照合します。カラム区切り記号はカンマ（デフォルト）であるので、この文字を入力カラムの文字列に含めることはできません。LONG_DESC（詳細な説明）カラム内の句読点など、余分なカンマをうっかり入れると、エラーが発生し、dbloader.err ファイルに記録されます。この種のエラーを避けるには、入力データのカラム値を区切るのに用いているカンマの数を注意深くチェックします。あるいは、別の区切り文字を選んで、databaseload.properties ファイル内に指定します。詳細については、「ステップ 2: databaseload.properties ファイルを編集する」を参照してください。

エラーと例外はすべて、DBLoader プログラムが動作しているコンソールに表示されます。エラーのあるレコードは読み飛ばされ、処理はファイルの最後まで続行します （エラーが発生しても、プログラムはトランザクションをロール バックしません）。

JSP を用いたカタログの表示

Commerce サービス に用意されている JavaServer Pages (JSP) テンプレートと JSP タグを利用すれば、プロダクト カタログのプレゼンテーション部分を容易に作成することができます。これらのテンプレートは、訪問者がカタログのカテゴリと商品アイテムを閲覧するためのメカニズムを提供します。すなわち、これらのテンプレート内の JSP タグには、そうした機能が実装されているのです。

JSP タグ ライブラリを使用すれば、プロダクト カタログに含まれている商品やカテゴリの属性を簡単に取得することができます。取得した属性はそのあと、HTML タグを用いて書式化することができます。カスタム レイアウトの作成には、どのような HTML エディタでも使用することができます。また、JSP 内にカスタム Java コードを組み込んで、カテゴリや商品を表示することもできます。

カタログ JSP タグを使用するには、JSP に以下のコードを追加することで、cat.tld タグ ライブラリを JSP ファイルにインポートする必要があります。

```
<%@ taglib uri=”cat.tld” prefix=”catalog” %>
```

BEA から提供されるデフォルトのプロダクト カタログを構成する JSP テンプレートでは、以下の 3 つの基本タグが使用されます。

- `<catalog:getProperty>`

  指定された ProductItem または Category から表示用のプロパティを取得する。明示的プロパティも暗黙的プロパティも取得することができます。

- `<catalog:iterateViewIterator>`

  指定された ViewIterator に対して反復処理を行う。ViewIterator に対する反復処理は、View 単位（反復ごとに 1 つの View）でも、View の中に含まれている Catalog アイテム単位（反復ごとに 1 つの ProductItem または Category）でも行うことができます。

- `<catalog:iterateThroughView>`

  指定された ViewIterator 内を順に調べ、指定された View 内に含まれている ProductItem または Category を反復処理する。

ビジネス ニーズに合わせ、必要に応じて、これら以外のタグを追加することができます。

<catalog:getProperty> タグの使い方

`ProductItem` または `Category` から表示用のプロパティを取得するには、`<catalog:getProperty>` タグ（表 14-3 参照）を用います。プロパティは、明示的プロパティ（Catalog アイテムに対して `get` メソッドを呼び出して取得できるプロパティ）でも、暗黙的プロパティ（Catalog アイテムに対して `ConfigurableEntity getProperty` メソッドを呼び出して取得できるプロパティ）でもかまいません。このタグは、まず、指定されたプロパティを明示的プロパティとして取得できるかどうかを調べます。それができない場合には、指定されたプロパティは暗黙的プロパティとして取得されます。

表 14-3 <catalog:getProperty> タグの属性

タグ属性	必須	データ型	解説	R/C
getterArgument	いいえ	String	明示的プロパティのゲッター メソッドへの引数として指定されるオブジェクトへの参照を示す。 これは、プロパティ セット フレームワークを用いて定義される暗黙的（すなわちカスタム）プロパティの取得にも使用されることがある。その場合には、プロパティ セットのスコープ名は getterArgument となる（以下に示すサンプル 2 を参照）。 オブジェクトは、<%= getterArgumentReference %> という形式で指定する必要があり、また、実行時表現でなければならない。	R
id	いいえ	String	id="newInstance" id 属性が指定された場合には、取得されたプロパティの値は、id に割り当てられた変数名で利用できるようになる。そうでない場合には、プロパティの値は、インラインに埋め込まれる。	C
object	はい	Catalog アイテム	ProductItem オブジェクトまたは Category オブジェクトへの参照を示す。これは、<%= objectReference %> の形式で指定する必要がある。	R
propertyName	はい	String	propertyName="propertyName" 取得するプロパティの名前。明示的プロパティの場合には、プロパティ名は表 14-4 に示す値のいずれか。	C
returnType	いいえ	String	returnType="returnType" id 属性が指定された場合には、id 属性で指定された変数のデータ型を宣言する。	C

表 14-4　propertyName に指定可能な値

プロパティ名	カタログ アイテム タイプ
“contributor | coverage | creationDate | creator | description | image | key | language | modifiedDate | name | publisher | relation | rights | source”	CatalogItem（共通プロパティ）
“jsp”	Category
“availability | currentPrice | format | jsp | msrp | shippingCode | taxCode | type | visible”	ProductItem

サンプル 1: アイテムからの JSP の詳細情報の取得 コード リスト 14-2 では、既存の ProductItem から JSP の詳細情報が取得されます。

コード リスト 14-2　アイテムからの JSP の詳細情報の取得

```
<%@ taglib uri=”cat.tld” prefix=”catalog” %>

<catalog:getProperty
object="<%= item %>"
   propertyName="Jsp"
   getterArgument=
   "<%= new Integer(ProductItem.DETAILED_DISPLAY_JSP_INDEX) %>"
   id="detailJspInfo"
returnType="com.beasys.commerce.ebusiness.catalog.JspInfo"
/>
```

サンプル 2: getterArgument 属性の使い方 コード リスト 14-3 では、getterArgument 属性を用いて、以下の特徴を持つプロパティ セット / スキーマの暗黙的（すなわちカスタム）プロパティを取得する方法を示しています。

- プロパティ セット名 : MyCatalog
- プロパティ名 : color

注意 : getterArgument は実行時表現でなければならないため、ここでは、MyCatalog を String 変数に代入し、その変数を getterArgument の値として用いています。

コード リスト 14-3　getterArgument 属性の使い方

```
<%@ taglib uri="cat.tld" prefix="catalog" %>

<%
String myPropertySetName = "MyCatalog";
ProductItem myProductItem = .....; // ProductItem への参照
%>
<catalog:getProperty
   object="<%=myProductItem%>
   propertyName="color"
   getterArgument="<%=myPropertySetName%>"
/>
```

<catalog:iterateViewIterator> タグの使い方

ViewIterator に対して反復処理を行うには、<catalog:iterateViewIterator> タグ（表 14-5 参照）を用います。ViewIterator は、固定サイズの一連の View に分割される潜在的に大量のリモート データ群に対するイテレータです。ViewIterator は、多数の ProductItem または Category から成る集合を返す可能性があるすべての Catalog サービス API メソッドから返されます。このタグを使用すれば、ViewIterator 内の要素を順に調べ、一度に 1 つのアイテム（ProductItem または Category）を、あるいは一度に 1 つの View 全体（一定数の ProductItem または Category から成る集合）を処理することができます。前者がデフォルトの動作です。このタグは、処理の完了時に ViewIterator の状態をリセットしないことに注意してください。

表 14-5 <catalog:iterateViewIterator> タグの属性

タグ属性	必須	データ型	解説	R/C
id	はい	String	id="newInstance" 現在、反復処理の対象になっているオブジェクトの値は、id に割り当てられた変数名で利用できるようになる。	C
iterator	はい	ViewIterator	ViewIterator オブジェクトへの参照を示す。 <%= iteratorReference %> という形式で指定する必要がある。	R

表 14-5 <catalog:iterateViewIterator> タグの属性（続き）

タグ属性	必須	データ型	解説	R/C
iterateByView	いいえ	String	iterateByView="{true|false}" ViewIterator に対する反復処理を View 単位で行うか、Catalog アイテム単位で行うかを指定する。指定されない場合、ViewIterator は Catalog アイテム単位で反復処理される。	C
returnType	いいえ	String	returnType="returnType" id 属性で指定された変数のデータ型を宣言する。デフォルトは java.lang.Object。 iterateByView が true の場合には、データ型は com.beasys.commerce.ebusiness.catalog.View と仮定される。	C

サンプル 1: ViewIterator 内に含まれるカテゴリのキーの表示 コード リスト 14-4 では、ViewIterator 内の全カテゴリのキーが表示されます。

コード リスト 14-4　ViewIterator 内に含まれるカテゴリのキーの表示

```
<%@ taglib uri="cat.tld" prefix="catalog" %>

<catalog:iterateViewIterator
   iterator="<%= myIterator %>"
   id="category"
   returnType="com.beasys.commerce.ebusiness.catalog.Category">
   <%= category.getKey().toString() %>
</catalog:iterateViewIterator>
```

サンプル 2: ViewIterator 内の全 View の表示 コード リスト 14-5 では、ViewIterator 内に含まれているすべての View が表示されます。

コード リスト 14-5　ViewIterator 内の全 View の表示

```
<%@ taglib uri="cat.tld" prefix="catalog" %>

<catalog:iterateViewIterator
   iterator="<%= myIterator %>"
   id="view"
   returnType="com.beasys.commerce.ebusiness.catalog.ViewIterator"
   iterateByView="true">
   <%= view.toString() %>
</catalog:iterateViewIterator>
```

<catalog:iterateThroughView> タグの使い方

<catalog:iterateThroughView> タグ (表 14-6 参照) は、指定された ViewIterator の View について反復処理を行います。このタグは、View の最後に達するまで、View 内の Catalog アイテムを一度に 1 つずつ反復処理します。反復対象となる特定の View を（インデックスによって）指定しない場合には、ViewIterator の現在の View が使用されます。このタグは、処理の完了時に ViewIterator の状態をリセットしないことに注意してください。

表 14-6 <catalog:iterateThroughView> タグの属性

タグ属性	必須	データ型	解説	R/C
id	はい	String	id="newInstance" 現在、反復処理の対象になっているオブジェクトの値は、id に割り当てられた変数名で利用できるようになる。	C
iterator	はい	ViewIterator	ViewIterator オブジェクトへの参照を示す。 <%= iteratorReference %> という形式で指定する必要がある。	R
returnType	いいえ	String	returnType="returnType" id 属性で指定された変数のデータ型を宣言する。デフォルトは java.lang.Object。	C

表 14-6 <catalog:iterateThroughView> タグの属性（続き）

タグ属性	必須	データ型	解説	R/C
viewIndex	いいえ	Integer	反復処理の対象となる View のインデックス（ViewIterator の先頭からの相対番号）を指定する。参照先のオブジェクトは、<%= viewIndexIntegerReference %> という形式で指定されなければならない。	R

サンプル 1: ViewIterator の現在の View に含まれている全 ProductItem のキーの表示 コード リスト 14-6 では、指定された ViewIterator の現在の View に含まれているすべての ProductItem のキーが表示されます。

コード リスト 14-6　ViewIterator の現在の View に含まれている全 ProductItem のキーの表示

```
<%@ taglib uri=”cat.tld” prefix=”catalog” %>

<catalog:iterateThroughView
   iterator="<%= myIterator %>"
   id="item"
   returnType="com.beasys.commerce.ebusiness.catalog.ProductItem">
<%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

サンプル 2: ViewIterator の先頭の View に含まれている全 ProductItem のキーの表示 コード リスト 14-7 では、指定された ViewIterator の先頭の View に含まれているすべての ProductItem のキーが表示されます。

コード リスト 14-7　ViewIterator の先頭の View に含まれている全 ProductItem のキーの表示

```
<%@ taglib uri=”cat.tld” prefix=”catalog” %>

<catalog:iterateThroughView
   iterator="<%= myIterator %>"
   id="item"
   returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
   viewIndex=”new Integer(0)”>
   <%= item.getKey().toString() %>
</catalog:iterateThroughView>
```

ショッピング カートとカタログを接続する

ショッピング カートとカタログを接続するには、カタログに shoppingcart.jsp テンプレートを実装する必要があります。このテンプレートには、ショッピング カート サービスを正常に実行するのに必要なビジネス ロジックとバックエンド処理の多くを管理するのに用いられる必須の入力プロセッサや Pipeline コンポーネントの実装など、ショッピング カート サービスを管理するコードが記述されています。このテンプレートをそのまま実装してもよいし、特定のニーズに合うように手を加えてもかまいません。

shoppingcart.jsp を実装する

shopppingcart.jsp を実装するには、まず、この JSP がアプリケーションの適切なポートレット フォルダに格納されていることを確かめます。そのあと、以下のいずれかを行います。

- 参照元のページからの URL を指定することで、shoppingcart.jsp を別の JSP に直接リンクする。参照元ページの機能しだいでは、転送イベントを起動するための [**カートの中を見る**] ボタンまたはアイコンを追加する必要があるかもしれません。

または

- 参照元ページのコードに、shoppingcart.jsp を呼び出す Webflow を組み込む。参照元ページの機能しだいでは、この Webflow を起動するための [**カートの中を見る**] ボタンまたはアイコンを追加する必要があるかもしれません。

shoppingcart.jsp の動作の仕組み

顧客は、任意の商品カタログ ページ内の [**カートの中を見る**] ボタン（あるいはそれと同等の機能を果たすもの）をクリックすることで、そのページから shoppingcart.jsp テンプレートに移動できます。shoppingcart.jsp テンプレートには、顧客のショッピング カートに現在入っている商品が表示されます。顧客がカート（現在の買物のアクティブな構成要素）に追加した商品ごとに、数量、商品名、定価、実売価格、割引、および小計が、shoppingcart.jsp テンプレートに表示されます。この情報に従って、注文の合計金額が表示されます。

商品の数量は編集可能なフィールドに表示され、顧客は、このフィールドに新しい数量を入力して [**Update**] ボタンをクリックするだけで、商品の数量を変更できます。顧客にとって便利なように、商品名は、プロダクト カタログ内の当該商品の説明にハイパーリンクされています。また、ショッピング カート内の商品ごとに、[**Remove**] ボタンと [**Buy later**] ボタンも用意されています。[**Remove**] ボタンをクリックすると、その商品はショッピング カートから削除され、一方、[**Buy later**] ボタンをクリックすると、その商品は [ショッピング カート] から [保存されているアイテム] リストに移されます。[保存されているアイテム] リストに表示されている商品ごとに、ハイパーリンク付きの商品名と商品概要が表示されます。さらに、このセクションの [**Remove**] ボタンや [**Add to cart**] ボタンを利用すると、顧客はその商品を完全に削除したり、アクティブな [ショッピング カート] に戻すことができます。

顧客が個々の商品アイテムのリンクをクリックして、その商品についての詳細情報を参照しようとすると、適切なプロダクト カタログ ページが次に表示されます。顧客が [**Update**]、[**Empty cart**]、[**Remove**]、[**Buy later**] のいずれかのボタンをクリックした場合には、適切な入力プロセッサや Pipeline が実行されて変更内容が保存されたあと、ショッピング カート ページ (`shoppingcart.jsp`) に戻ります。

顧客は、ショッピング カートに表示されている内容を見てこれでよいと思ったら、[**Check out**] ボタンをクリックしてチェックアウト プロセスを開始することができます。その場合には、配送情報のページ (`shipping.jsp`) が次に表示されます。

注意： 顧客がまだサイトにログインしていない状態で［**Check out**］ボタンをクリックした場合には、（`shipping.jsp` テンプレートがロードされる前に）`login.jsp` テンプレートが表示され、ログインするように促されます。

[保存されているアイテム] リストの機能を利用するには、顧客はまずログインしておく必要があります。

顧客のショッピング カートに何も商品が入っていない場合には、[Emptycart] ボタン、[Update] ボタン、および [Check out] ボタンは利用できません。

顧客は、ショッピング カートの内容を見てこれでよいと思ったら、[Check out] ボタンをクリックしてチェックアウト プロセスを開始することができます。

注意： 顧客が e コマース サイトにログインしていない場合には、チェックアウト プロセスの次のステップに進む前に、ログインを求められます。

顧客が最初からやり直したいと思った場合には、[Empty cart] ボタンをクリックすれば、ショッピング カートの中身（アクティブなアイテムも保留中のアイテムも）をそっくり空にすることができます。顧客が買い物を続けたい場合には、[Continue shopping] ボタンをクリックすれば、プロダクト カタログに戻ることができます。

解説

`shoppingcart.jsp` テンプレートに注釈を付けたものを、図 14-1 と図 14-2 に示します。図 14-1 に示すのは、まだログインしていない顧客に表示されるページです。

図 14-1 `shoppingcart.jsp` ー まだログインしていない訪問者用に書式化された Web ページ

図 14-2 に示すのは、ログイン済みの顧客に表示されるページです。

図 14-2　shoppingcart.jsp ー ログイン済みの訪問者用に書式化された Web ページ

このテンプレートのメイン コンテンツ領域には、動的に生成されるデータと静的なコンテンツが両方とも表示されています。shoppingcart.jsp の動的なコンテンツは、WebLogic Server JSP タグと Pipeline JSP タグを用いて生成されます。これらの JSP タグでは、アクティブなショッピング カート用のコンテンツも [保存されているアイテム] リスト用のコンテンツも取得し表示します。shoppingcart.jsp テンプレートの場合、フォーム ポストを行うものには、[Empty cart]、[Check out]、[Remove]、[Update]、および [Continue shopping] があります。

ユーザがログインする前後での変更点は、以下のとおりです。

1. [**ログイン**] リンクが [ログアウト] に変わります。
2. [ようこそ] セクションが表示され、その中に顧客の名前、顧客のプロファイルを参照するためのリンク、およびログアウトのためのリンクが示されます。
3. [履歴を見る] セクションが表示され、その中に顧客の注文履歴と支払履歴のリンクが示されます。

テンプレートの主要コンポーネントを表 14-7 に示します。

表 14-7 テンプレート コンポーネント

コンポーネントのタイプ	コンポーネント
インクルードされるテンプレート	♦ admin.inc ♦ stylesheet.inc ♦ header.inc ♦ leftside.inc ♦ footer.inc
タグ ライブラリ	<%@ taglib uri="weblogic.tld" prefix="wl" %> <%@ taglib uri="webflow.tld" prefix="webflow" %> <%@ taglib uri="i18n.tld" prefix="i18n" %>
インポートされる Java パッケージ	java.util.* java.text.* com.beasys.commerce.axiom.units.* com.beasys.commerce.ebusiness.shoppingcart.* com.bea.commerce.ebusiness.price.service.DiscountPresentation com.bea.commerce.ebusiness.price.quote.OrderAdjustment com.bea.commerce.ebusiness.price.quote.AdjustmentDetail com.beasys.commerce.webflow.HttpRequestConstants com.beasys.commerce.webflow.PipelineSessionConstants com.bea.p13n.appflow.webflow.WebflowJSPHelper

デフォルト Webflow での位置

顧客は、[カートの中を見る] ボタンをクリックすることで、任意の商品カタログ ページから shoppingcart.jsp テンプレートに移動できます。顧客は、このページに表示されているショッピング カートの内容を見てこれでよいと思ったら、[Check out] ボタンをクリックしてチェックアウト プロセスを開始することができます。その場合には、配送情報のページ (shipping.jsp) が次に表示されます。

注意： 顧客がまだサイトにログインしていない状態で [**Check out**] ボタンをクリックした場合には、（`shipping.jsp` テンプレートがロードされる前に）`login.jsp` テンプレートが表示され、ログインするように促されます。

顧客が個々の商品アイテムのリンクをクリックして、その商品についての詳細情報を参照しようとすると、適切なプロダクト カタログ ページが次に表示されます。顧客が [Update]、[Empty cart]、[Remove]、[Buy later] のいずれかのボタンをクリックした場合には、適切な入力プロセッサや Pipeline が実行されて変更内容が保存されたあと、ショッピング カート ページ (`shoppingcart.jsp`) に戻ります。

この JSP テンプレートは、`sampleapp_order` ネームスペース内にあります。

イベント

顧客がショッピング カートの内容を操作するボタンをクリックするたびに、その操作はイベントとみなされます。各イベントが発生すると、デフォルト Webflow 内の特定の応答がトリガされて、顧客は操作を続けることができます。この応答によって別の JSP がロードされることもありますが、通常は、入力プロセッサまたは Pipeline が先に呼び出されます。これらのイベントとそれによって呼び出されるビジネス ロジックを表 14-8 に示します。

表 14-8　`shoppingcart.jsp` で扱われるイベント

イベント	Webflow の応答
`--`	`InitShoppingCartIP`
`--`	`RefreshSavedList`
`button.checkout`	`InitShippingMethodListIP`
`button.deleteItemFromShoppingCart`	`DeleteProductItemFromShoppingCartIP`
`button.deleteItemFromSavedList`	`UpdateSkuIP` `DeleteProductItemFromSavedList`
`button.emptyShoppingCart`	`EmptyShoppingCartIP`
`button.moveItemToSavedList`	`UpdateSkuIP` `MoveProductItemToSavedList`

表 14-8 shoppingcart.jsp で扱われるイベント

イベント	Webflow の応答
button.moveItemToShoppingCart	UpdateSkuIP MoveProductItemToShoppingCart
button.updateShoppingCartQuantities	UpdateShoppingCartQuantitiesIP

表 14-9 では、表 14-8 に示した各 Pipeline について簡単に説明します。

表 14-9 ショッピング カートに関連付けられている Pipeline

Pipeline	解説
RefreshSavedList	RefreshSavedListPC を含み、トランザクション対応でない。
DeleteProductItemFromSavedList	DeleteProductItemFromSavedListPC と PriceShoppingCartPC を含み、トランザクション対応である。
MoveProductItemToSavedList	MoveProductItemToSavedListPC と PriceShoppingCartPC を含み、トランザクション対応である。
MoveProductItemToShoppingCart	MoveProductItemToShoppingCartPC と PriceShoppingCartPC を含み、トランザクション対応である。

注意： InitShoppingCartIP と RefreshSavedList Pipeline は shoppingcart.jsp テンプレートに関連付けられていますが、いずれも、このページで発生するイベントでトリガされるわけではありません。これらはどちらも、shoppingcart.jsp が表示される前に実行されるのです。InitShoppingCartIP 入力プロセッサが、顧客の買い物に備えて空のショッピング カートを作成するのに対して、RefreshSavedList Pipeline は、顧客がそれまでに保留にしたショッピング カート内商品のリストを取得します。

shoppingcart.jsp でのデータ表示の仕組み

shoppingcart.jsp テンプレートの目的の 1 つは、顧客の買い物に関するデータを表示して、顧客がそれを確認できるようにすることです。shoppingcart.jsp では、WebLogic Server JSP タグ、Pipeline JSP タグ、およびアクセサ メソッド / 属性を組み合わせて、これを実現しています。

まず、getProperty JSP タグによって、Pipeline セッションから SHOPPING_CART 属性と SAVED_SHOPPING_CART 属性が取得されます。表 14-10 に、これらの属性の詳細を示します。

表 14-10　shoppingcart.jsp Pipeline セッションの属性

属性	データ型	解説
PipelineSessionConstants .SAVED_SHOPPING_CART	com.beasys.commerce.ebusiness .shoppingcart.ShoppingCart	保留状態のショッピング カート（保留されている商品アイテムの発生元）
PipelineSessionConstants .SHOPPING_CART	com.beasys.commerce.ebusiness .shoppingcart.ShoppingCart	現在アクティブになっているショッピング カート

getProperty JSP タグを用いて Pipeline セッションからこれらの属性を取得する方法を、コード リスト 14-8 に示します。

コード リスト 14-8　ショッピング カート属性の取得

```
<webflow:getProperty id="shoppingCart"
   property="<%=PipelineSessionConstants.SHOPPING_CART%>"

type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"
   scope="session" namespace="sampleapp_main" />

<webflow:getProperty id="savedShoppingCart"
   property="<%=PipelineSessionConstants.SAVED_SHOPPING_CART%>"
   type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"
   scope="session" namespace="sampleapp_main" />
```

Pipeline セッション属性内に格納されているデータには、Java スクリプトレット内でアクセサ メソッド / 属性を用いてアクセスします。ShoppingCart（および savedShoppingCart）のアクセサ メソッド / 属性の詳細を表 14-11 に、また、ShoppingCartLine のアクセサ メソッド / 属性については表 14-12 に、それぞれ示します。

表 14-11 ShoppingCart のアクセサ メソッド / 属性

メソッド / 属性	解説
getShoppingCartLineCollection()	ショッピング カート内の個々の明細行（すなわち、ShoppingCartLine）のコレクション。
getTotal	この場合には、OrderConstants.LINE_TAX パラメータを指定して計算される合計税額。 **注意：** getTotal() メソッドでは、異なる合計タイプを組み合わせることもできる。詳細については、『*Javadoc*』を参照のこと。

getShoppingCartLineCollection() メソッドを使用すると、ショッピング カート内の個々の明細行のコレクションを取得できるので、各明細行に含まれている個々の情報を取り出すためのアクセサ メソッド / 属性も用意されています。これらのメソッドおよび属性を表 14-12 に示します。

表 14-12 ShoppingCartLine のアクセサ メソッド / 属性

メソッド / 属性	解説
getQuantity()	商品アイテムの数量。
getProductItem()	ショッピングカートの明細行に記載されている商品アイテム。
getUnitPrice()	ショッピング カートに追加された時点での商品の単価。MSRP（希望小売価格）とは異なる場合がある。
getBaseTotal(int totalType)	割引前の注文総額。

Java スクリプトレット内でのこれらのアクセサ メソッド / 属性の使用例を、コード リスト 14-9 に示します。

コード リスト 14-9　shoppingcart.jsp Java スクリプトレット内でのアクセサ メソッドの使い方

```
<<td align="right" valign="top" bgcolor="#CCCCFF"><div class="tabletext" nowrap>
<%-- i18n タグを用いることで、"currency.properties" ファイルでは、返される 3 文字の --%>
<%-- ISO4217 コード ("USD" など ) の代わりに "$" などの表示通貨を使用できるようになる。--%>

         <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getProductItem().getMsrp().getCurrency() %>"/>
         <%= WebflowJSPHelper.priceFormat(shoppingCartLine.getProductItem().
            getMsrp().getValue() ) %></div>

      </td>

      <td align="right" valign="top"><div class="tabletext" nowrap>

          <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getUnitPrice().getCurrency() %>"/>

         <%= WebflowJSPHelper.priceFormat( shoppingCartLine.getUnitPrice().
            getValue() ) %></div>

      </td>
      <td align="right" valign="top" bgcolor="#CCCCFF"><div class="tabletext"
         nowrap>

         <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getBaseSavings().getCurrency() %>"/>

         <%= WebflowJSPHelper.priceFormat( shoppingCartLine.getBaseSavings().
            getValue() ) %></div>

      </td>

      <td align="right" valign="top"><div class="tabletext" nowrap>

         <i18n:getMessage bundleName="/commerce/currency" messageName="<%=
            shoppingCartLine.getBaseTotal().getCurrency() %>"/>

         <%= WebflowJSPHelper.priceFormat( shoppingCartLine.getBaseTotal().
            getValue() ) %>

         </div>
      </td>
```

shoppingcart.jsp で用いられるフォーム フィールド

shoppingcart.jsp テンプレートのもう 1 つの目的は、さまざまな HTML フォーム フィールドを用いて、顧客が自分のショッピング カートの内容を変更できるようにすることです。また、これらのフォーム フィールドは、必要な情報を Webflow に渡すためにも使用されます。

shoppingcart.jsp テンプレートで使用されるフォーム フィールドとそれらの解説を、表 14-13 にまとめます。

表 14-13 shoppingcart.jsp のフォーム フィールド

パラメータ名	タイプ	解説
"event"	Hidden	どのイベントがトリガされたかを示す。次の処理を決定するために Webflow で使用。
"origin"	Hidden	現在のページ (shoppingcart.jsp) の名前。Webflow で使用。
HttpRequestConstants.CATALOG_ITEM_SKU	Hidden	イベントの作用を受ける商品の SKU。
NewQuantity_<SKU> ここで、<*SKU*> はショッピング カートの明細行に記載されている商品の SKU。	Textbox	ショッピング カート内の当該商品の新しい数量。このページで顧客からの入力が必要なのは、このフォーム フィールドのみ。

注意 : JSP コード内でリテラルとして扱われるパラメータは、引用符で囲んで示されています。一方、リテラルでないパラメータを JSP 内で使用するには、スクリプトレット構文 (たとえば、<%= HttpRequestConstants.CATALOG_ITEM_SKU %> など) に従う必要があります。

shoppingcart.jsp で用いられる入力プロセッサ

shoppingcart.jsp では、入力プロセッサおよび Pipeline と呼ばれる Webflow コンポーネントを用いて、必要なビジネス ロジックの多くを実行します。この節では、利用可能な主要入力プロセッサについて説明します。これらの入力プロセッサは、Webflow メカニズムから起動されたときに、より複雑なショッピング カート サービス タスクを実行するために呼び出される Java クラスを表します。

この節では、以下に示すこれらの入力プロセッサについて説明します。

- DeleteProductItemFromShoppingCartIP
- EmptyShoppingCartIP
- InitShoppingCartIP
- UpdateShoppingCartQuantitiesIP
- UpdateSkuIP

注意： Webflow の使用、作成、あるいは変更と、入力プロセッサの使用については、「ポータル ナビゲーションのセットアップ」を参照してください。

DeleteProductItemFromShoppingCartIP

この入力プロセッサ (入力プロセッサ名はすべて末尾に「IP」という文字が付く) は、ショッピング カートから商品アイテムを削除します。

呼び出されるクラス	`examples.wlcs.sampleapp.shoppingcart.webflow.DeleteProductItemFromShoppingCartIP`
HTTPServletRequest の必須パラメータ	`HttpRequestConstants.CATALOG_ITEM_SKU`
必須の Pipeline セッション 属性	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.CATALOG_ITEM`
更新される Pipeline セッション 属性	`PipelineSessionConstants.SHOPPING_CART`

削除される Pipeline セッション 属性	なし
検証	なし
例外	`ProcessingException`。必須のリクエスト パラメータや必須の Pipeline セッション属性が利用できない場合に送出される。

EmptyShoppingCartIP

この入力プロセッサは、新しいショッピング カートを作成し、それを Pipeline セッションに格納したあと、古いショッピング カートを破棄します。

呼び出されるクラス	`examples.wlcs.sampleapp.shoppingcart.webflow.EmptyShoppingCartIP`
`HTTPServletRequest` の必須パラメータ	なし
必須の Pipeline セッション 属性	なし

更新される Pipeline セッション 属性	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
	`PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`

削除される Pipeline セッション 属性	なし
検証	なし
例外	なし

InitShoppingCartIP

この入力プロセッサは、`shoppingcart.jsp` テンプレートをロードする前に、アクティブなショッピング カートを初期化します。ショッピング カートがすでに存在している場合には、この入力プロセッサは何もしません。

呼び出されるクラス	`examples.wlcs.sampleapp.shoppingcart.webflow.InitShoppingCartIP`
HTTPServletRequest の必須パラメータ	なし
必須の Pipeline セッション 属性	なし
更新される Pipeline セッション 属性	`PipelineSessionConstants.SHOPPING_CART` `PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
削除される Pipeline セッション 属性	なし
検証	なし
例外	なし

UpdateShoppingCartQuantitiesIP

この入力プロセッサは、ショッピング カートの各明細行の数量フィールドを検証し、それらの数量を設定します。数量が 0 の場合には、その商品アイテムをショッピング カートから削除します。

呼び出されるクラス	`examples.wlcs.sampleapp.shoppingcart.webflow.UpdateShoppingCartQuantitiesIP`
HTTPServletRequest の必須パラメータ	`NewQuantity_<SKU>` ここで、*<SKU>* はショッピング カートの明細行に記載されている商品の SKU。
必須の Pipeline セッション 属性	`PipelineSessionConstants.SHOPPING_CART`

更新される Pipeline セッション 属性	`PipelineSessionConstants.SHOPPING_CART`
	`PipelineSessionConstants.UPDATED_QUANTITY_DELTAS`
	`PipelineSessionConstants.UPDATED_PRODUCT_ITEMS`

削除される Pipeline セッション 属性	なし
検証	数量フィールドに正の整数だけが入力されているかどうかを確認する。
例外	ProcessingException。必須のリクエスト パラメータや必須の Pipeline セッション属性が利用できない場合に送出される。

UpdateSkuIP

この入力プロセッサは、HTTP リクエストから SKU を読み込んで、Pipeline セッション内に設定します。

呼び出されるクラス	examples.wlcs.sampleapp.shoppingcart.webflow. UpdateSkuIP
HTTPServletRequest の必須パラメータ	HttpRequestConstants.CATALOG_ITEM_SKU
必須の Pipeline セッション 属性	なし
更新される Pipeline セッション 属性	PipelineSessionConstants.CATALOG_ITEM_SKU
削除される Pipeline セッション 属性	なし
検証	なし
例外	ProcessingException。必須のリクエスト パラメータが利用できない場合に送出される。

shoppingcart.jsp で用いられる Pipeline コンポーネント

この節では、shoppingcart.jsp テンプレートに関連付けられている各 Pipeline コンポーネントについて簡単に説明します。これらの Pipeline は、Shopping Cart サービスに関係する特定タスクの実行を Webflow で開始する際に呼び出されるプロセッサ ノードです。

注意： Pipeline コンポーネントの中には、他の基本 Pipeline コンポーネントを拡張するものがあります。基本クラスの詳細については、『*Javadoc*』を参照してください。

Pipeline コンポーネントの詳細については、9-3 ページの「ノードのタイプ」を参照してください。

この節では、以下の Pipeline コンポーネントについて説明します。

- DeleteProductItemFromSavedListPC
- MoveProductItemToSavedListPC
- MoveProductItemToShoppingCartPC
- RefreshSavedListPC
- PriceShoppingCartPC
- AddToCartTrackerPC
- RemoveFromCartTrackerPC

注意： Webflow の使用、作成、あるいは変更と、Pipeline コンポーネントの使用については、「ポータル ナビゲーションのセットアップ」を参照してください。

DeleteProductItemFromSavedListPC

この Pipeline コンポーネント (Pipeline コンポーネント名はすべて末尾に「PC」という文字が付く) は、保留アイテム リストから商品を削除し、データベース内の `WLCS_SAVED_ITEM_LIST` テーブルを更新します。

呼び出されるクラス	`examples.wlcs.sampleapp.shoppingcart.pipeline.DeleteProductItemFromSavedListPC`

必須の Pipeline セッション 属性	`PipelineSessionConstants.CATALOG_ITEM_SKU`
	`PipelineSessionConstants.SAVED_SHOPPING_CART`
	`PipelineSessionConstants.USER_NAME`

更新される Pipeline セッション 属性	`PipelineSessionConstants.SAVED_SHOPPING_CART`

削除される Pipeline セッション 属性	なし
タイプ	セッション Bean
JNDI 名	examples.wlcs.sampleapp.shoppingcart.pipeline.DeleteProductItemFromSavedListPC
例外	PipelineException。必須の Pipeline セッション属性が利用できない場合に送出される。

MoveProductItemToSavedListPC

この Pipeline コンポーネントは、ショッピング カートから商品アイテムを削除し、それを保留アイテム リストに追加します。その後、データベース内の WLCS_SAVED_ITEM_LIST テーブルを更新します。

呼び出されるクラス	examples.wlcs.sampleapp.shoppingcart.pipeline.MoveProductItemToSavedListPC
必須の Pipeline セッション 属性	PipelineSessionConstants.CATALOG_ITEM_SKU PipelineSessionConstants.SAVED_SHOPPING_CART PipelineSessionConstants.SHOPPING_CART PipelineSessionConstants.USER_NAME
更新される Pipeline セッション 属性	PipelineSessionConstants.SAVED_SHOPPING_CART PipelineSessionConstants.SHOPPING_CART PipelineSessionConstants.CATALOG_ITEM PipelineSessionConstants.QUANTITY
削除される Pipeline セッション 属性	なし
タイプ	セッション Bean
JNDI 名	examples.wlcs.sampleapp.shoppingcart.pipeline.MoveProductItemToSavedListPC
例外	PipelineException。必須の Pipeline セッション属性が利用できない場合に送出される。

MoveProductItemToShoppingCartPC

この Pipeline コンポーネントは、保留アイテム リストから商品を削除し、それを数量 1 としてショッピング カートに追加します。その後、データベース内の WLCS_SAVED_ITEM_LIST テーブルを更新します。

呼び出されるクラス	examples.wlcs.sampleapp.shoppingcart.pipeline.MoveProductItemToShoppingCartPC
必須の Pipeline セッション 属性	PipelineSessionConstants.CATALOG_ITEM_SKU PipelineSessionConstants.SAVED_SHOPPING_CART PipelineSessionConstants.SHOPPING_CART PipelineSessionConstants.USER_NAME
更新される Pipeline セッション 属性	PipelineSessionConstants.SAVED_SHOPPING_CART PipelineSessionConstants.SHOPPING_CART PipelineSessionConstants.CATALOG_ITEM
削除される Pipeline セッション 属性	なし
タイプ	セッション Bean
JNDI 名	examples.wlcs.sampleapp.shoppingcart.pipeline.MoveProductItemToShoppingCartPC
例外	PipelineException。必須の Pipeline セッション属性が利用できない場合に送出される。

RefreshSavedListPC

この Pipeline コンポーネントは、WLCS_SAVED_ITEM_LIST テーブルにクエリを発行し、Pipeline セッション内の保留ショッピング カートをリフレッシュします。Pipeline セッション内に保留ショッピング カートが存在しない場合には、保留アイテム リストだけが更新されます。

呼び出されるクラス	examples.wlcs.sampleapp.shoppingcart.pipeline.RefreshSavedListPC
必須の Pipeline セッション 属性	PipelineSessionConstants.USER_NAME

更新される Pipeline セッション 属性	PipelineSessionConstants.SAVED_SHOPPING_CART
削除される Pipeline セッション 属性	なし
タイプ	セッション Bean
JNDI 名	examples.wlcs.sampleapp.shoppingcart.pipeline.RefreshSavedListPC
例外	PipelineException。必須の Pipeline セッション属性が利用できない場合に送出される。

PriceShoppingCartPC

この Pipeline コンポーネントは、Pricing サービスを呼び出して、各明細行の総額（小計）、割引額、ショッピング カート全体の総額、およびショッピング カート全体の割引総額を計算します。

呼び出されるクラス	examples.wlcs.sampleapp.shoppingcart.pipeline.PriceShoppingCartPC
必須の Pipeline セッション 属性	PipelineSessionConstants.SHOPPING_CART PipelineSessionConstants.USER_NAME
更新される Pipeline セッション 属性	PipelineSessionConstants.SHOPPING_CART
削除される Pipeline セッション 属性	なし
タイプ	Java オブジェクト
JNDI 名	なし
例外	PipelineException。Pricing サービスが何らかの点で正常に機能しない場合に送出される。

AddToCartTrackerPC

この Pipeline コンポーネントは、カートにどの商品アイテムが追加されたかを示す AddToCartEvent を発生させます。このイベントの詳細については、「イベントおよび行動追跡」を参照してください。

呼び出されるクラス	examples.wlcs.sampleapp.tracking.pipeline.AddToCartTrackerPC
解説	
必須の Pipeline セッション 属性	PipelineSessionConstants.CATALOG_ITEM PipelineSessionConstants.HTTP_SESSION_ID PipelineSessionConstants.USER_NAME PipelineSessionConstants.STOREFRONT PipelineSessionConstants.CUSTOM_REQUEST
更新される Pipeline セッション 属性	なし
削除される Pipeline セッション 属性	なし
タイプ	Java オブジェクト
JNDI 名	なし
例外	なし

RemoveFromCartTrackerPC

この Pipeline コンポーネントは、カートからどの商品アイテムが削除されたかを示す RemoveFromCartEvent を発生させます。このイベントの詳細については、「イベントおよび行動追跡」を参照してください。

呼び出されるクラス	examples.wlcs.sampleapp.tracking.pipeline.RemoveFromCartTrackerPC

必須の Pipeline セッション 属性	PipelineSessionConstants.CATALOG_ITEM PipelineSessionConstants.HTTP_SESSION_ID PipelineSessionConstants.USER_NAME PipelineSessionConstants.STOREFRONT PipelineSessionConstants.CUSTOM_REQUEST
更新される Pipeline セッション 属性	なし
削除される Pipeline セッション 属性	なし
タイプ	Java オブジェクト
JNDI 名	なし
例外	なし

UpdateShoppingCartQuantitiesTrackerPC

この Pipeline コンポーネントは、ショッピング カートの明細行ごとに以下のいずれかを実行します。

- 当該行で選択されている商品アイテムが増えた場合には、AddToCartEvent を発生させる。
- 当該行で選択されている商品アイテムが減った場合には、RemoveFromCartEvent を発生させる。
- 当該行で商品アイテムの数に増減がない場合には、イベントを発生させない。

呼び出されるクラス	examples.wlcs.sampleapp.tracking.pipeline. UpdateShoppingCartQuantitiesTrackerPC
必須の Pipeline セッション 属性	PipelineSessionConstants.UPDATED_PRODUCT_ITEMS PipelineSessionConstants.UPDATED_QUANTITY_DELTAS PipelineSessionConstants.HTTP_SESSION_ID PipelineSessionConstants.USER_NAME PipelineSessionConstants.STOREFRONT PipelineSessionConstants.CUSTOM_REQUEST

更新される Pipeline セッション 属性	なし
削除される Pipeline セッション 属性	なし
タイプ	Java オブジェクト
JNDI 名	なし
例外	なし

サービスとカタログ キャッシュを統合する

カタログ アーキテクチャには、プロダクト カタログ内の商品アイテムおよびカテゴリ用の強力なキャッシング メカニズムが用意されています。サービスとキャッシュの統合に当たっては、サービスをキャッシュのフロントエンドにするか、バックエンドにするかを選択することができます。現在、ProductItemManager と CategoryManager では、キャッシング アーキテクチャが活用されています。

CatalogManager のデプロイメント記述子で Bean の JNDI 名を変更することによって、キャッシュのフロントエンドにあるサービスが別のものに置き換わります。元のサービスは、独自のキャッシング メカニズムを実装するか、あるいはキャッシングの活用を諦めなければなりません。

あらかじめ定義されているサービス (CatalogManager のデプロイメント記述子で指定されている) では、商品アイテムとカテゴリにアクセスするためのキャッシュを実装しています。以下の Bean は、キャッシュに問い合わせ、キャッシュされたデータがあればそれを返します。そうでなければ、デプロイメント記述子で指定されている Bean に処理を委託します。

- `com.beasys.commerce.ebusiness.catalog.service.item.ProductItemManager`
- `com.beasys.commerce.ebusiness.catalog.service.category.CategoryManager`

ProductItemManager Bean と CategoryManager Bean のデプロイメント記述子を編集することで、キャッシュのバックエンドでもプロダクト カタログの機能を利用できるようにすることができます。それによって、キャッシング アーキテク

チャのことを気にせずに、カタログの永続性モデルに専念できるようになります。コード リスト 14-10 に示すように、ejb-jar.xml デプロイメント記述子で指定されている現在の代理サービス プロバイダ クラス (JdbcCategoryManager) を、CategoryManager インタフェースを実装する新しいセッション Bean の名前に置き換える必要があります。同様の変更を、weblogic-ejb-jar.xml ファイルに対しても行う必要があるでしょう。

コード リスト 14-10　CategoryManager のデプロイメント記述子 (ejb-jar.xml)

```
<session>
   <ejb-name>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManager
   </ejb-name>
   <home>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManagerHome
   </home>
   <remote>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManager
   </remote>
   <ejb-class>
      com.beasys.commerce.ebusiness.catalog.service.
         category.CategoryManagerImpl
   </ejb-class>
   <session-type>Stateless</session-type>
   <transaction-type>Container</transaction-type>

    <!--  delegateName を指定して、どの EJB に委託するかをブリッジ
  コンポーネント ( カタログ マネージャで使用されるもの ) に指示する。
    このように、env-entry を変更することで委託先を変更することができる ...
    -->

  <env-entry>
      <env-entry-name>delegateName</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>ejb/JdbcCategoryManager</env-entry-value>
   </env-entry>

   <ejb-ref>
      <ejb-ref-name>ejb/JdbcCategoryManager</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>
         com.beasys.commerce.ebusiness.catalog.service.category.
         JdbcCategoryManagerHome
      </home>
      <remote>
         com.beasys.commerce.ebusiness.catalog.service.
            category.JdbcCategoryManager
```

```
      </remote>
    </ejb-ref>

    .
    .
    .
</session>
```

ejb-jar.xml ファイルと weblogic-ejb-jar.xml ファイルは、ebusiness.jar ファイルにパッケージ化されています。この JAR ファイルは、<BEA_HOME>\weblogic700\common\templates\domains\shared\bea\portal\apps\jars ディレクトリ内にあります（<BEA_HOME> は、BEA アプリケーションのインストール先ディレクトリ）。

第 15 章 イベントおよび行動追跡

イベント システムを利用すれば、訪問者とポータルすなわち Web サイトとのやり取りを識別することができます。

顧客とのやり取り：イベントの主たる用途の 1 つには、プロモーションやキャンペーンなどでの顧客とのやり取りがあります。たとえば、顧客がショッピング カートに商品を入れたときに関連商品の広告表示をトリガするといったことは、キャンペーンにおけるイベントの利用法の簡単な例です。

行動追跡：イベントの主たる用途には、もう 1 つ、イベントを記録することで訪問者の行動を追跡することが挙げられます。イベントを記録すると言っても、単なるナビゲーション ロギングではありません。このようなロギングからは、どのようなページが訪問されたかがわかるだけです。これに対して、行動追跡を利用すれば、訪問者がページ上で何に注目し何に反応したかがわかります。あるいは、これも同じく重要ですが、ページ上で訪問者が何に「*注目しなかったか*」もわかります。データベースにイベントを記録することで、最先端の e 解析システムや e マーケティング システムでイベント データを用いてデータ マイニングを行うことができます。イベント情報を分析すると、サイトのコンテンツを各訪問者に合わせてカスタマイズするルールを作成または機能拡張したり、プロモーション用キャンペーンの有効性を評価するのに役立ちます。

カスタム イベント：イベント システムを利用すれば、独自のイベントを作成することができます。たとえば、各ポートレットへの訪問者とアクセス頻度を記録するイベントを作成することもできるでしょう。カスタム イベントを作成する場合、そのイベントは WebLogic Portal の永続性メカニズムを用いてデータベースに記録することもできますし、そうしないことも可能です。永続化されるイベントは行動追跡イベントと呼ばれます。永続化されないカスタム イベントの場合には、15-14 ページの「カスタム イベント クラスを作成する」での説明に従ってください。また、カスタム行動追跡イベントを作成する場合には、15-21 ページの「行動追跡イベント クラスを作成する」での説明に従ってください。

この章の内容は主として、J2EE に精通した人を対象にしています。扱うトピックは以下のとおりです。

- キャンペーンにおけるイベントの動作の仕組み — イベントとキャンペーンの関係についての概要

- Event サービスの動作の仕組み — イベントおよび行動追跡の機能をフルに活用できるように Event サービスの仕組みを解説
- 標準イベントの使い方 — WebLogic Portal に付属しているイベントの概要とそれらの生成方法
- カスタム イベントの作成 — イベントおよび行動追跡用のクラスとリスナの作成手順
- 行動追跡を有効にする方法 — 行動追跡をサービスとして有効にする方法
- Event サービスのデバッグ — イベントのデバッグを有効にする方法
- カスタム イベントの登録 — キャンペーンで使用できるようにカスタム イベントを登録する方法
- 行動追跡のアクティブ化 — 行動追跡をアクティブにする方法

キャンペーンにおけるイベントの動作の仕組み

イベントは、商品の閲覧などの訪問者のアクションによって発生します。Event サービスは、そのイベントを検出したことをすべてのイベント リスナに通知します。Campaign サービスのイベント リスナはそれを受けて、キャンペーン シナリオをアクティブにします。キャンペーンでは、ユーザに合ったコンテンツを選択する一連のルールを用いて、アクションを開始します。利用可能なアクションは、以下のとおりです。

- 広告などの特定のコンテンツを Web ページに表示する
- プロモーション用の電子メールを顧客に送る
- 顧客に割引を提供する

注意： キャンペーンの詳細については、E-Business Control Center のオンライン ヘルプを参照してください。

Event サービスの動作の仕組み

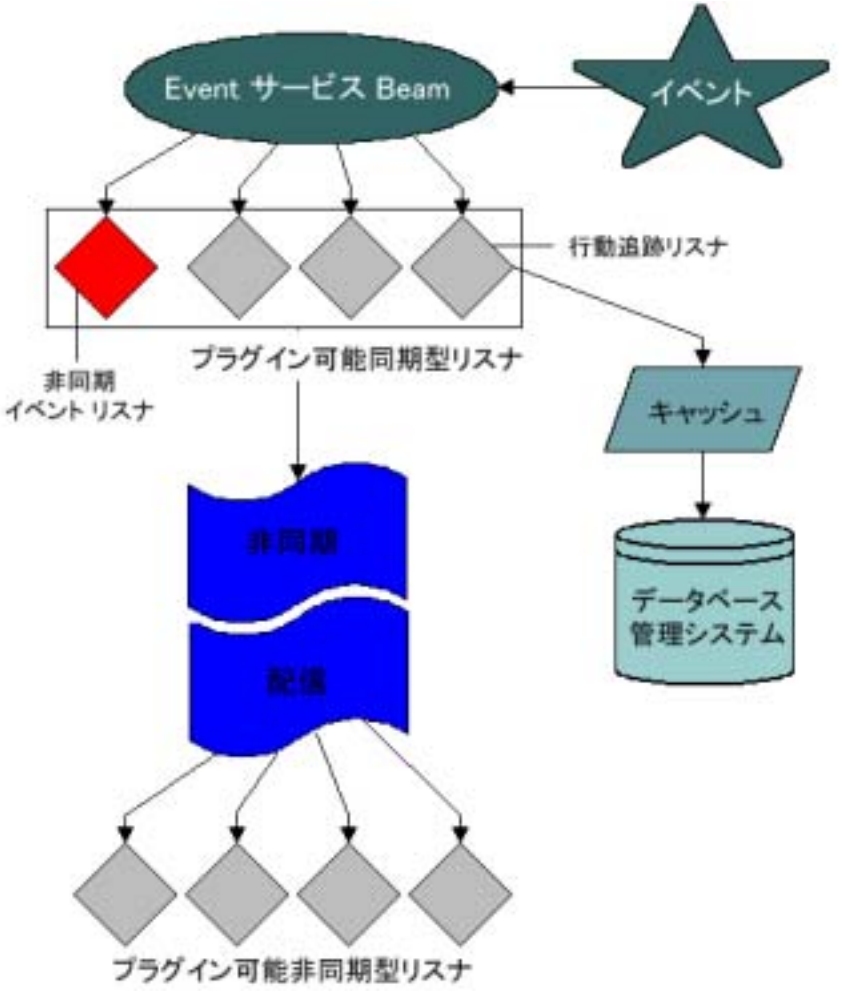
Event サービスの動作の仕組みを理解すれば、イベントを利用する上で役に立つとともに、イベントの生成とカスタム イベントの設計に必要な情報も得られます。

Event サービスの概要：Event サービスは、拡張可能な汎用のイベント作成および伝達システムです。図 15-1 に示すように、イベントは JSP タグなどのトリガによって生成されます。トリガはイベント オブジェクトを作成し、Event サービス Bean を見つけ、そして作成したイベント オブジェクトをその Event サービスに渡します。Event サービスはプラグイン リスナと連携して動作し、そのプラグイン リスナによって、当該イベントの受信登録を済ませているリスナにイベントが伝達されます。各イベント リスナは作成される際に、受信を希望するイベント タイプのリストを返します。Event サービスは、イベントを受信すると、そのイベントのタイプをチェックし、そのイベント タイプの受信をサブスクライブ（登録）しているすべてのリスナにそのイベントを送信します。

リスナ タイプ：Event サービスにプラグインされるリスナには、イベントに同期的に応答するものと非同期的に応答するものの 2 種類があります。同期型リスナでは、送られてきたイベントの作成および送信元の実行スレッドをそのまま用いて、そのイベントに応答するアクションを実行します。行動追跡リスナは、同期型リスナだけを使用します。一方、非同期型リスナでは、送られてきたイベントをその作成元のスレッドから受信し、その後いつか、別の実行スレッド内でイベントを処理します。非同期型サービスが用意されているのは、イベント ハンドラの実行に時間がかかる場合でも、Web サイト訪問者の目から見てアプリケーションの動作に遅延が生じないようにするためです。

個々のプラグイン リスナを Event サービスの同期側にインストールするか、非同期側にインストールするかは、アプリケーションの要件によって決まります。Event サービスのコンフィグレーションは、WebLogic Server Administration Console を用いて行われます。

図 15-1　イベントのメカニズム



イベント リスナは com.bea.p13n.events.EventListener インタフェースを実装します。このインタフェースには、以下の 2 つの public メソッドのシグネチャが定義されています。

- `public String[] getTypes()`
- `public void handleEvent( Event theEvent )`

最初のメソッドは、リスナで Event サービスから受信する対象となるイベント タイプのリストを返します。たとえば、リスナで *Foo* タイプのイベントを受信するようになっている場合には、そのリスナ オブジェクトに対する `getTypes()` メソッドの呼び出しで返される配列には、*Foo* が要素として含まれています。2 番目のメソッドは、イベントがリスナに渡されたときに呼び出されます。リスナには、自分が同期型であるか非同期型であるかはわかりません。

キャンペーン イベントだけを受信するリスナを作成するのであれば、WebLogic Server Administration Console で、そのリスナの完全修飾クラス名を追加します。作成されたリスナは `EventListener` インタフェースを実装し、その `getTypes()` メソッドが呼び出されたときに以下のイベント タイプを返します。

```
{“ClickCampaignEvent”,“DisplayCampaignEvent”,“CampaignUserActivityEvent” }
```

このリスナをインストールすると、これら 3 つのタイプのイベントが、リスナの `handleEvent( Event theEvent )` インタフェースに伝達されます。

図 15-1 に描かれた「非同期配信」の部分では、Event サービスの同期側から非同期的に送られてくるイベントが非同期型イベント ハンドラで受信されることを示しています。このイベント ハンドラはそのあと、各リスナでの受信対象として登録されているイベント タイプに基づいて、プラグイン可能非同期型リスナにイベントをディスパッチします。

イベント シーケンスの動作

一連のイベントが発生する様子（サンプル）を図 15-2 と 図 15-3 に示します。これらの図は、イベントの発生順序を直感的に理解していただくためのもので、イベントの順序付けを包括的に扱ったものではありません。これらを取り上げたのは、イベントによって Web サイトにおける訪問者のライフ サイクルについてどのような洞察が得られるのかということと、アプリケーションでイベントをいつどのように用いればよいかを示すためです。

図 15-2　イベント シーケンスのサンプル（その 1）

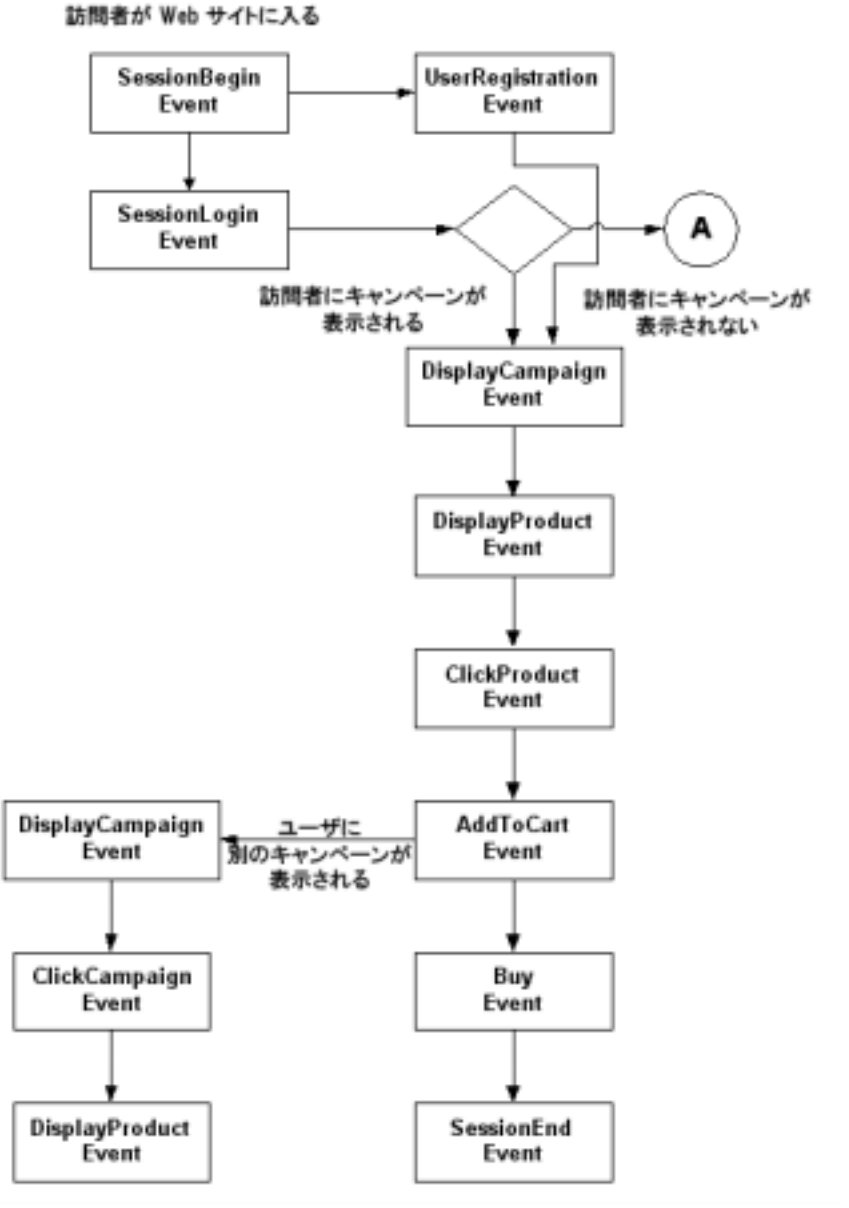
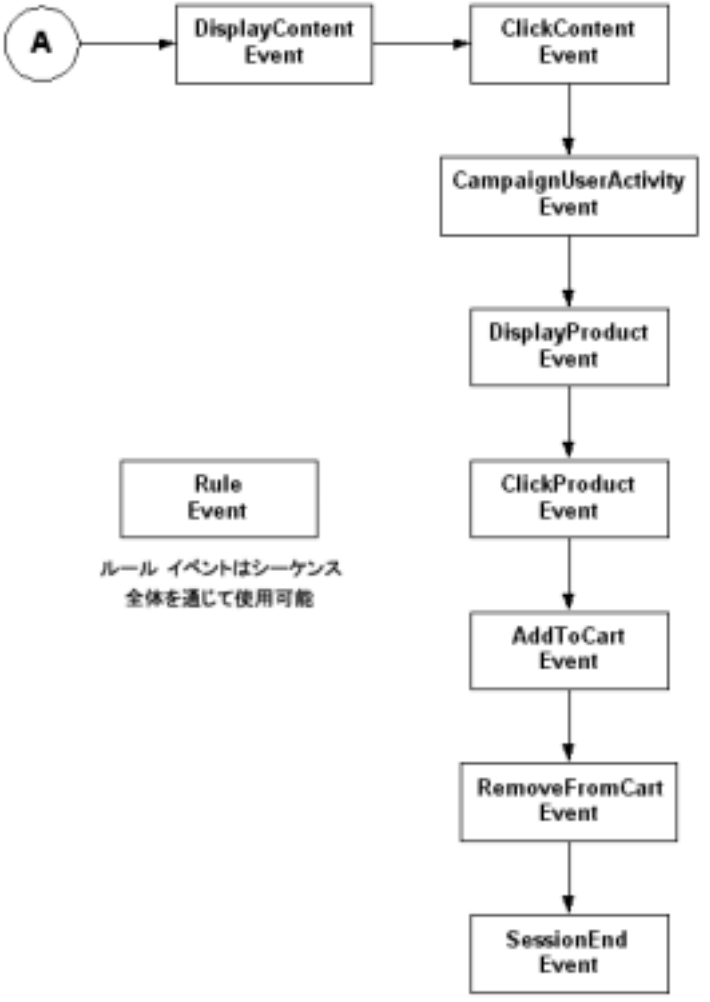


図 15-3　イベント シーケンスのサンプル（その 2）



標準イベントの使い方

この節では、WebLogic Portal にあらかじめ用意されている標準イベントの使い方について大まかに説明します。各イベントの詳細については、付録 A「イベント解説」を参照してください。付録 A には、各種イベントの解説、イベントの生成元（ジェネレータ）、イベント生成が行われるクラス、使用例、および各イベント オブジェクト内のデータのタイプが記載されています。

WebLogic Portal の標準イベントにはすべて、以下の基本情報が記載されています。

- イベントの発生元のアプリケーション
- イベントの発生時刻
- イベントのタイプ
- セッション ID
- ユーザ ID（訪問者がログインしていなければ null）
- イベント固有の情報

WebLogic Portal イベントはいくつかのカテゴリに分類されます。以下では、各タイプのイベント カテゴリと、イベントを発生させるアクションについての簡単な説明を示します。

- **セッション イベント**：訪問者のセッションの開始時、終了時、およびログイン時に発生。
  - SessionBeginEvent
  - SessionEndEvent
  - SessionLoginEvent
- **ユーザ登録イベント**：訪問者が Web サイトで登録すると発生。
  - UserRegistrationEvent
- **商品イベント**：訪問者が商品を提示されるか、提示された商品をクリック（選択）すると発生。
  - ClickProductEvent
  - DisplayProductEvent

- **コンテンツ イベント**: 訪問者が広告などの何らかのコンテンツを提示されるか、提示されたコンテンツをクリック（選択）すると発生。
  - ClickContentEvent
  - DisplayContentEvent
- **カート イベント**: 商品アイテムが訪問者のショッピング カートに追加、削除、あるいは更新されると発生。これらのイベントは注文品全体が購入されたときにも発生します。
  - AddToCartEvent
  - RemoveFromCartEvent
  - PurchaseCartEvent
- **購入イベント**: 訪問者が 1 つ以上の商品の購入を完了すると発生。
  - BuyEvent
- **ルール イベント**: Web サイトでの訪問者のナビゲーションに合わせて起動されるルール
  - RuleEvent
- **キャンペーン イベント**: キャンペーンとの関連で発生するイベント
  - CampaignUserActivityEvent
  - DisplayCampaignEvent
  - ClickCampaignEvent

サーブレット ライフサイクル イベントとサーブレット フィルタ イベント

以下のイベントは、サーブレット仕様 2.3 API のライフサイクル イベントの一部として定義されています。

- SessionBeginEvent
- SessionEndEvent

これらはセッションの Created() イベントおよび Destroyed() イベントのリスナで、この 2 つのイベントは、web.xml ファイルに定義されているサーブレットによって生成されます。web.xml ファイルは、アプリケーションごとに 1 つ存在します。たとえば、wlcsApp という e コマース アプリケーションの場合には、このファイルは以下の場所にあります。

```
<BEA_HOME>\weblogic700\portal\applications\wlcsApp\wlcs\WEB-INF
```

以下のイベントは JSP タグによって生成され、サーブレット仕様 2.3 の <filter> 要素によってフィルタ処理されます。

- ClickContentEvent
- ClickProductEvent
- ClickCampaignEvent

表示される Web ページごとに、Web アプリケーション サーブレットは HttpServletRequest 内にクリック イベントが存在するかどうかをチェックします。各ページ クリックはそのあと、サーブレット仕様 2.3 の <filter> 要素で定義されているとおりに、Web アプリケーション サーブレットによってフィルタ処理されます。クリック イベントは、サーブレットの呼び出しのたびに、<filter> 要素が呼び出されると自動的に生成されます。生成されるイベントのタイプの決定は、ClickThroughFilter によって HttpServletRequest 内のイベント タイプをチェックすることで行われます。有効なタイプは、以下のファイルに定義されています。

- <BEA_HOME>\weblogic700\portal\classes\campaign\campaign-app*.properties
- <BEA_HOME>\weblogic700\portal\classes\commerce\commerce-app*.properties
- <BEA_HOME>\weblogic700\portal\classes\p13n\p13n-app*.properties

ログイン イベントと作成イベントを生成する

この節では、ログイン イベントおよびユーザ登録イベントを生成するいくつかの異なる方法について説明します。

SessionLoginEvent: SessionLoginEvent は、以下のいずれかの方法で生成することができます。

- ログインの処理に <um:login> タグまたは weblogic.servlet.security.ServletAuthentication を手動で使用する場合には、com.bea.p13n.tracking.TrackingEventHelper.dispatchSessionLoginEvent() メソッドを用いる。
- j_security_check によるフォーム ベースのログインを直接使用する場合には、Web アプリケーションの WEB-INF\weblogic.xml ファイルで、<auth-filter> 要素として com.bea.p13n.servlets.P13NAuthFilter を登録する。JSP や Webflow プロセッサをコーディングする必要はありません。

UserRegistrationEvent: UserRegistrationEvent の生成には、com.bea.p13n.tracking.TrackingEventHelper.dispatchUserRegistrationEvent() メソッドを使用します。このイベントは、SessionLoginEvent（ユーザ作成中に発生するはず）のあとで生成しなければなりません。入力プロセッサを使用してもよいし、JSP に記述してもかまいません。

Webflow: ポータル Webflow フレームワークを使用する場合には、SessionLoginEvent と UserRegistrationEvent は、必要に応じてセキュリティ Webflow 内の com.bea.portal.appflow.processor.security.PostLoginProcessor から自動生成されます。

イベント ジェネレータを追加またはカスタマイズする

標準イベントは、e コマース サイトにおける重要な局面で生成されます。イベントを発生させるコンポーネントには、Java API、JSP タグ、JSP スクリプトレット、Webflow 入力プロセッサ、Pipeline コンポーネント、コンテンツ セレクタ、分類アドバイズレットなどがあります。以下の各イベントについては、ジェネレータ（生成元）の追加またはカスタマイズが可能です。

- DisplayContentEvent
- DisplayProductEvent
- ClickContentEvent

- ClickProductEvent

これらの各イベントは JSP タグによって生成されます。これらのイベントをトリガする JSP タグを用いて、どの商品やどのようなコンテンツによってこれらのイベントが生成されるかを指定することができます。たとえば、wlcsApp という e コマース アプリケーションでは、DisplayProductEvent に対応する JSP タグが details.jsp に記述されています。

このタグ（コード リスト 15-1 に示す）では、カタログの詳細ページに商品が表示されると、それに対応するイベントを生成します。ある特定の商品に対してイベントを生成したい場合には、その商品の SKU を手がかりに選択的処理を行うスクリプトレットを書くことができます。

コード リスト 15-1　JSP タグ

```
<%-- 商品が表示されたら、displayProductEvent を発生させる --%>
<productTracking:displayProductEvent documentId="<%= item.getName() %>"
        documentType="<%= DisplayProductEvent.ITEM_BROWSE %>"
                        sku="<%= item.getKey().getIdentifier() %>" />
```

イベント用の JSP タグを追加する際には、以下のような、タグ ライブラリ記述子への参照をコードに挿入しなければなりません。

```
<%@ taglib uri="productTracking.tld" prefix="productTracking" %>
```

details.jsp は以下のディレクトリに入っています。

```
<BEA_HOME>\weblogic700\portal\applications\wlcsApp\wlcs\commerce\catalog
```

カスタム イベントの作成

この節では、カスタム イベントの作成に必要な情報を提供します。任意の追跡対象についてのカスタム イベントを作成することができます。WebLogic Portal の永続性メカニズムを用いてイベントを記録させる場合には、15-21 ページの「行動追跡イベント クラスを作成する」で説明しているように、行動追跡イベントを作成します。

カスタム イベントのアイデア： 顧客ごとにどのページが表示されているかを知らせるイベントを作成することもできます。その場合には、その情報を用いて、セッションごとに閲覧された平均ページ数や最も人気があったページを決定することができます。さらに、マーケティング担当者は、特定ページの表示に基づいたプロモーション用キャンペーンを開発する際にこのイベントを利用することもできます。

カスタム イベントの作成方法を説明するために、ここでは簡単な例を示します。以下の各節では、この例を参照し、拡張します。

カスタム イベントの作成は、複数のステップから成るプロセスです。以下のリストでは、このプロセスの概要と、ここで扱われない情報の参照先を示します。

1. イベントを定義するコードを書きます。
2. イベント リスナを定義するコードを書きます。
3. リスナ クラスを Event サービスにインストールします。
4. JSP タグまたは API 呼び出しを用いて、イベントを生成するコードを書きます。
5. イベントを登録します（キャンペーンで使用される場合）。
6. イベント データを EVENT テーブルに記録するには、そのイベント用のエントリを EVENT_TYPE テーブルに作成します。詳細については、『管理者ガイド』の「行動追跡データの永続化」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/sysadmin.htm#1194894) を参照。

カスタム イベント クラスを作成する

カスタム イベントを作成するには、以下の手順に従います。

1. **イベント クラスを作成する**：このクラスは、リスナで受信したイベントを正しく解釈し処理するために必要なすべての情報をカプセル化したものです。

   すべてのカスタム イベントは、`com.bea.p13n.events.Event` クラスのサブクラスでなければなりません。この基本クラスでは、イベントのタイムスタンプとタイプの設定および取得を処理し、カスタム イベントの属性にアクセスする手段を提供します。`Event` クラスには、属性値を設定および取得する以下の 2 つのメソッドがあります。

   ```
   setAttribute( String theKey, Serializable theValue )
   getAttribute( String theKey )
   ```

   カスタム イベントのコンストラクタからこれらのメソッドを呼び出して、新しいイベントに固有の属性を設定することができます。なお、`Event` オブジェクト内に値として設定されるすべてのオブジェクトは、Java のシリアライズ可能なデータ型で*なければならない*ことを覚えておいてください。

   `getTimeStamp()` メソッドは、イベントの作成日時をミリ秒単位で返します。イベントのタイプには、`Event` クラスの `getType()` メソッドを用いてアクセスします。`Event` オブジェクト インスタンスのタイムスタンプとタイプの設定は、インスタンスの作成時に `Event` コンストラクタ内でのみ行うことができます。イベントのタイムスタンプは、明示的に指定されない場合には、イベントの作成時に自動的に設定されます。アプリケーション属性は、イベントの作成元のアプリケーションか Event サービス EJB (Enterprise JavaBean) アプリケーションのいずれかから自動的に設定されます。

   例：カスタム イベントの作成プロセスの例を示すために、ここでは `TestEvent` という簡単な例を取り上げます。この例は、イベント サブクラスの基本的な作成方法を示すものです。実際のカスタム イベントは、おそらくこれよりも複雑でしょう。

2. **タイプを作成する**：カスタム イベントを作成するには、まずイベント タイプを決定する必要があります。このイベント タイプは、スーパークラスのコンストラクタ（たとえば、`Event` クラスのコンストラクタ）に渡さなければなりません。このイベント タイプは、カスタム イベント オブジェクトのインスタンスに対して `getType()` メソッドを呼び出すことで返されます。イベント タイプの例を以下に示します。

```
/** イベント タイプ */
  public static final String TYPE = "TestEvent";
```

カスタム イベント オブジェクトの基本クラス Event を正しく初期化するために、値 TYPE が Event のコンストラクタに渡されます。すべてのイベントのタイプは、Java の単純な文字列オブジェクトでなければなりません。

3. **キーを定義する**：タイプを定義したら、カスタム イベントに格納されている属性にアクセスするためのキーを定義する必要があります。これらの属性には、コンストラクタで値を設定することができます。たとえば、TestEvent クラスには description と Zip Code の 2 つのプロパティがあります。description に関連付けられる値のデータ型は String で、Zip Code のデータ型は Integer です。キーは以下のように定義します。

```
/**
     * 1 つ目のユーザ定義プロパティのイベント属性キーの名前。
     * 属性値は String
 */
   public static final String DESCRIPTION = "description";

/**
 * 属性値はすべてシリアライズ可能でなければならないことに注意。
     * 2 つ目のユーザ定義プロパティのイベント属性キーの名前。
   * 属性値は Integer
 */
public static final String ZIP_CODE = "Zip Code";
```

最後に、イベント タイプと属性の設定に関する処理をコンストラクタにまとめ、そのコンストラクタでイベント オブジェクトを作成します。コンストラクタは以下のようになります。

```
/**
 * 新しい TestEvent を作成する
 *
 *
   * @param desc このイベントの説明
   * @param zip 郵便番号
 */
   public TestEvent( String desc, Integer zip )
 {
   /* このイベントのタイプを指定して Event クラスのコンストラクタを呼
び出す */
     super( TYPE );

   if( desc != null )
         setAttribute( DESCRIPTION, desc );
```

```
        if( zip != null )
              setAttribute( ZIP_CODE, zip );
    }
```

断片をすべて 1 つにまとめる：カスタム イベント クラス全体をコード リスト 15-2 に示します。

コード リスト 15-2　TestEvent クラス

```
/*TestEvent クラスの定義はここから */

public class TestEvent
  extends com.bea.p13n.events.Event
{
  /** イベント タイプ */
  public static final String TYPE = "TestEvent";

  /**
     * 1 つ目のユーザ定義プロパティのイベント属性キーの名前。
     * 属性値は String
   */
   public static final String DESCRIPTION = "description";

  /**
     * 2 つ目のユーザ定義プロパティのイベント属性キーの名前。
   * 属性値は Integer
   */
public static final String ZIP_CODE = "Zip Code";

  /**
   * 新しい TestEvent を作成する
   *
   *
   * @param desc このイベントの説明
   * @param zip 郵便番号
   */
   public TestEvent( String desc, Integer zip )
   {
   /* このイベントのタイプを指定して Event クラスのコンストラクタを呼び出す
     super( TYPE );

     if( descriptionValue != null )
         setAttribute( DESCRIPTION, desc );
```

```
     if( ZipCodeValue != null )
         setAttribute( ZIP_CODE, zip );
   }
}
/*TestEvent クラスの定義はここまで */
```

サンプルについての補足： コード リスト 15-2 の例では、基本クラス Event と Event サービスの基本機能の使い方を示しています。実際のカスタム イベントのコンストラクタは、おそらくこれよりも複雑でしょう。たとえば、コンストラクタでデフォルト値のチェックを行ったり、null 属性を排除するなどの処理を行うかもしれません。さらに、カスタム イベント オブジェクトに、もっと多くのメソッドやメンバー データが含まれる場合もあるでしょう。

注意： カスタム イベントをキャンペーンで使用するには、以下のオブジェクトを属性として追加する必要があります。

- Request（キーは “request”）– com.bea.p13n.http.Request 型のリクエスト。これは、HttpServletRequest をパラメータとして取るコンストラクタで作成しなければならない。
- Session（キーは “session”）– com.bea.p13n.http.Session 型のセッション。これは、HttpServletRequest をパラメータとして取るコンストラクタで作成される。
- UserId（キーは “userId”）– ユーザ名の格納された文字列。

ファイルを保存する： WebLogic Server で使用されるエンタープライズ アプリケーション クラスパスに含まれている場所であれば、どこにファイルを保存してもかまいません。

カスタム イベント リスナを作成する

イベントをリスンするには、イベント リスナを定義する必要があります。すべてのイベント リスナは com.bea.p13n.events.EventListener インタフェースを実装し、引数のないコンストラクタ（デフォルト コンストラクタ）を持つ必要があります。このインタフェースには、指定されたタイプのイベントを、配信先として登録済みのリスナに送信するのに不可欠な 2 つのメソッドの仕様が、以下のとおり定義されています。

```
public String[] getTypes()

public void handleEvent( Event ev )
```

最初のメソッドは、リスナで受信されるイベントのタイプを文字列配列で返します。Event サービスは、与えられたタイプのイベントを受信対象とする（上記のタイプ配列に含まれる）リスナに、そのイベントをディスパッチします。特定のリスナが現在のイベント タイプを受信対象として登録済みであると Event サービスが判断すると、そのタイプのイベントは、`handleEvent( Event ev )` の呼び出しを用いて、そのリスナにディスパッチされます。

イベント リスナの両メソッドを実装する：カスタム イベント リスナを作成する際には、`EventListener` インタフェースに定義されている 2 つのメソッドを両方とも実装する必要があります。引き続き `TestEvent` の例を取り上げると、Event サービスを通じて送信される `TestEvent` インスタンスは、`TestEventListener` でリスンされます。これは、以下のように記述することができます。

```
/** このリスナで受信するイベントのタイプ */
    private String[] eventTypes = {"TestEvent"};

/**
  このリスナに伝達するイベントのタイプを決定するために
 Event サービスによって呼び出されるメソッド。
 */
public String[] getTypes()
{
  return eventTypes;
}
```

受信したイベントを処理するには、`handleEvent( Event evt )` メソッドを以下のように実装します。

```
/**
 * Event サービスから送られたイベントを処理する
 */
public void handleEvent( Event ev )
{
  System.out.println("TestListener::handleEvent " +
                     " -> received an event" +
                     " of type: " + ev.getType() );

  /* ここで処理を行う */

}
```

これらの断片とコンストラクタをひとまとめにすると、コード リスト 15-3 に示すように、TestEvent オブジェクトの受信を登録する簡単なイベント リスナができあがります。

コード リスト 15-3 イベント リスナ

```
import com.bea.p13n.events.EventListener;
import com.bea.p13n.events.Event;

/**
 * リスナを行動追跡システムに容易にプラグインできる
 * ことを示す TestListener
 *
 * イベントを受信するには、このクラスをプロパティ eventService.listeners に
 * 追加しなければならない。 このプロパティには完全修飾クラス名を
 * 追加する必要がある。直前の行の終わりには必ず「,\」を追加すること。
 * 付け忘れると、プロパティの解析時に新しいクラス名が認識されない。
 *
 * リスナで受信するイベントのタイプは、String 配列 eventTypes に
 * 列挙される。 必要に応じて、イベント タイプの文字列を追加または削除する。
 *
 * @author Copyright (c) 2001 by BEA Systems, Inc. All Rights Reserved.
 */
public class TestListener
    implements EventListener
{

private String[] eventTypes = {"TestEvent"};

public TestListener()
{
}

public String[] getTypes()
{
    return eventTypes;
}

public void handleEvent( Event ev )
{
     System.out.println("TestListener::handleEvent -> received an event" +
                        " of type: " + ev.getType() );

     return;
```

```
    }
}
```

イベント リスナは汎用的でなければならない：単純なイベントを記述するのが簡単なように、単純な EventListener を作成するのも簡単な作業です。イベント リスナがどのようなものであれ、その内部の実装は汎用的なものにすべきです。TestEventListener の同一インスタンスで、すべての TestEvent オブジェクトが処理されるとはかぎりません。したがって、TestEventListener は完全にステートレスでなければならず、イベント オブジェクトまたは外部のデータベースに格納されているデータを操作するものでなければなりません。

注意： どのリスナでも、複数のインスタンスを同時に実行することができます。

Event サービスにリスナ クラスをインストールする

注意： この節では、サンプル ポータルにリスナ クラスを追加する方法を説明します。実際のアプリケーションの場合も、同様の手順に従うことになるでしょう。

Event サービスがアプリケーションのサービスとして存在しない場合には、WebLogic Server Administration Console を用いて追加します。

行動追跡を有効にするには、行動追跡リスナを同期型リスナとして Event サービスに追加する必要があります。

リスナを追加する：同期型リスナまたは非同期型リスナを追加するには、以下の手順に従います。

注意： 行動追跡リスナは、同期型リスナとしてのみ実装することができます。

1. WebLogic Server Administration Console で、以下のようにして、sampleportalDomain のノード ツリー内の [同期型リスナ] タブまたは [非同期型リスナ] タブを開きます。

   Web ブラウザで http://<hostname>:<port>/console にアクセスし、Administration Console の左ペインで [sampleportalDomain ｜デプロイメント ｜アプリケーション｜ sampleportal ｜ Service Configuration ｜ Event Service]

ノードを選択したあと、右ペインの [コンフィグレーション] タブから [同期型リスナ] タブまたは [非同期型リスナ] タブをクリックします。

2. 図 15-5 に示すように、該当するフィールドに同期型リスナまたは非同期型リスナを追加します。

図 15-4　WebLogic Server Administration Console ― Event サービス

行動追跡イベント クラスを作成する

行動追跡イベントは、訪問者と Web サイトとのやり取りを追跡する特別なタイプのイベントです。e 解析システムでは、行動追跡イベントから収集したデータを用いて、訪問者の行動を評価します。その評価結果は主に、キャンペーンの開発と、Web サイトで訪問者に提供するコンテンツの最適化に利用されます。

例：行動追跡イベントとそのリスナを作成する方法は、`TestEvent` クラスと `TestEventListener` の場合とよく似ています。ここでも、簡単な例を示します。ここで扱うサンプル追跡イベントを `TestTrackingEvent` とします。データベースに永続化（記録）され BEA Behavior Tracking で使用されるすべての行動追跡イベントは、`com.bea.p13n.tracking.listeners.BehaviorTrackingListener` によって処理されます。`BehaviorTrackingListener` は、`com.bea.p13n.events.EventListener` クラスを拡張したものです。

再デプロイ：`BehaviorTrackingListener` を Event サービスにおけるリスナとして定義したあと、そのリスナが行動追跡イベントを受信し永続化できるようになるには、アプリケーションを再デプロイする必要があります。

バッファについて：このリスナは Event サービスからイベントを受信し、バッファに格納します。バッファの内容は、データベース内のイベント用テーブルに断続的に永続化されます。イベントをバッファからスイープする（データベースにフラッシュする）頻度は、Behavior Tracking サービスに関する以下のプロパティで指定します。

- `MaxBufferSize` – イベント バッファの最大サイズを設定する。このプロパティを 0 に設定すると、すべてのイベントが受信時に永続化されることになる。
- `SweepInterval` – バッファをチェックして、バッファ内のイベントを永続化すべきかどうかを調べる間隔を秒単位で設定する。イベントが永続化されるのは、最大バッファ サイズ (`MaxBufferSize`) に達した場合か、バッファ内の最長滞留時間 (`SweepMaxTime`) を超過した場合である。
- `SweepMaxTime` – データベースへの強制フラッシュを行うまでの待ち時間を秒単位で設定する。これは、どのようなキャッシュであれ、イベントがキャッシュ内に存在できる最長の時間である。

最適化：これらのプロパティを調整して、パフォーマンスを最適化します。バッファ スイープの実行は、データベースへの書き込みに時間がかかりすぎるほど頻度が低くてもいけませんが、書き込み操作が無駄になるくらい頻繁であってもいけません。

イベント バッファ スイープのコンフィグレーション

注意： この節では、サンプル ポータルのバッファ スイープをコンフィグレーションする方法を説明します。実際のアプリケーションの場合も、同様の手順に従うことになるでしょう。

Event サービスがアプリケーションのサービスとして存在しない場合には、WebLogic Server Administration Console を用いて追加します。

イベント バッファのスイープをコンフィグレーションするには、以下の手順に従います。

1. WebLogic Server Administration Console で、以下のようにして、sampleportalDomain のノード ツリー内の [Behavior Tracking サービス] ページを開きます。

   Web ブラウザで http://<hostname>:<port>/console にアクセスし、Administration Console の左ペインで [sampleportalDomain ｜デプロイメント

｜アプリケーション｜ sampleportal ｜ Service Configuration ｜ Behavior Tracking Service] ノードを選択します。

2. 図 15-5 に示すように、該当するフィールドにバッファの新しい設定値を入力します。

図 15-5 WebLogic Server Administration Console ― Behavior Tracking サービス

オフライン処理を支援する

行動追跡イベントは、データベース内の EVENT テーブルに永続化（記録）されるように設計されています。行動追跡イベントのデータを記録するプロセスの一環としてデータの XML 表現が作成され、EVENT テーブルの xml_definition カラムに格納されます。ここでの説明は、BEA の行動追跡イベント永続性メカニズムを利用する場合には、この場所にイベントを永続化する*必要があります*。そのため、あらかじめ用意されている EVENT テーブルにイベントを永続化するには、カスタム イベントは、作成と永続化が適切に行われるように、この節の説明に従ったものでなければなりません。

XML-XSD スキーマ：行動追跡イベントに記載されているデータの仕様を形式的に記述するには、新しいイベントの XML-XSD スキーマを定義する必要があります。XSD は XML 作成の検証に内部的に使用されるわけではありませんが、作成される XML はデータベース内のイベント データを表現しています。イベント クラスが適切に作成され使用されれば、そのクラスは XML-XSD スキーマに準拠することになります。XSD ドキュメントを用いれば、行動追跡イベントのコン

ストラクタと属性キーの作成は容易になります。各標準イベントの具体的なデータ要素については、『管理者ガイド』の「XML_DEFINITION カラムのデータ要素」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/sysadmin.htm#1195110) を参照してください。

ファイル間の関連：行動追跡イベントを XML 表現に正しく変換するには、イベント タイプに関連付けられているスキーマの XML インスタンス ドキュメントを完全に記述するいくつかのメンバー データを、行動追跡イベントに定義する必要があります。このデータは、イベントに関連付けられているネームスペースと XSD ファイルを記述したものです。たとえば、コード リスト 15-4 と コード リスト 15-5 は、以下のファイル間の関連を示しています。

- com.bea.campaign.tracking.events.ClickCampaignEvent と
- /lib/schema/tracking-click-campaign-1_0_1.xsd (<BEA_HOME>\weblogic700\portal\\lib\campaign\ejb\campaign.jar 内)

その他の例については、既存の XSD ファイルを参照してください。

コード リスト 15-4　ClickCampaignEvent.java

```
/**
   キャンペーンのクリックを追跡するためのイベント
 */
public class ClickCampaignEvent
    extends ClickEvent
{
    /** イベント タイプ */
    public static final String TYPE = "ClickCampaignEvent";

    /**
      このイベントの XML ネームスペース
    */
private static final String XML_NAMESPACE=
       "http://www.bea.com/servers/commerce/xsd/tracking/click-campaign/1.01";

    /**
      このイベントのスキーマを記述した XSD ファイル
    */
    private static final String XSD_FILE = "tracking-click-campaign-1_0_1.xsd";

    /**
     * キャンペーン ID のイベント属性キー名。
```

```
     * 属性値は String
     */
    public static final String CAMPAIGN_ID = "campaign-id";

    /**
     * シナリオ ID のイベント属性キー名。
     * 属性値は String
     */
    public static final String SCENARIO_ID = "scenario-id";

    /**
     * 店舗（アプリケーション名）のイベント属性キー名。
     * 属性値は String
     */
    public static final String APPLICATION_NAME = "application-name";

    /**
     * 商品アイテム カテゴリ ID のイベント属性キー名。
     * 属性値は String
     */
     public static final String PLACEHOLDER_ID = "placeholder-id";

    /**
       コンストラクタに渡される documentType データへの入力例
       属性値は String
    */
    public static final String BANNER_AD_PROMOTION = "bannerAdPromotion";

    /**
       このオブジェクトを表現する XML 内に存在する
       要素のキーとそれらの順序。
    */
    private static final String localSchemaKeys[] =
    {
        APPLICATION, SESSION_ID, USER_ID, DOCUMENT_TYPE, DOCUMENT_ID,
        CAMPAIGN_ID, SCENARIO_ID, APPLICATION_NAME, PLACEHOLDER_ID
    };

    /**
     * 新しい ClickCampaignEvent を作成する。
     *


     * @param theSessionId HttpSession.getId() で取得
     * @param theUserId HttpServletRequest.getRemoteUser() または
     *          それに相当するメソッドで取得（不明の場合には null）
* @param theRequest HTTP サーブレット リクエスト オブジェクト
```

```
     * @param aDocumentType クリックされたコンテンツのドキュメント タイプ (null も可 )
     *
     * @param aDocumentId クリックされたコンテンツのドキュメント ID (null も可 )
     * @param aCampaignId アイテム クリック時のキャンペーンのキャンペーン ID
     *
     * @param aScenarioId アイテム クリック時の ( キャンペーン内 ) シナリオの
     * シナリオ ID
     * @param aApplicationName アプリケーション名 ( 店舗 ) (null も可 )
     *
     * @param aPlaceholderId プレースホルダ ID
     */
    public ClickCampaignEvent( String theSessionId,
                               String theUserId,
                               HttpServletRequest theRequest,
                               String aDocumentType,
                               String aDocumentId,
                               String aCampaignId,
                               String aScenarioId,
                               String aApplicationName,
                               String aPlaceholderId )
    {
        super( TYPE,
               theSessionId,
               theUserId,
               XML_NAMESPACE,
               XSD_FILE,
               localSchemaKeys,
               theRequest,
               aDocumentType,
               aDocumentId);

        if( aCampaignId != null ) setAttribute( CAMPAIGN_ID, aCampaignId );
        if( aScenarioId != null ) setAttribute( SCENARIO_ID, aScenarioId );
        if( aApplicationName != null ) setAttribute( APPLICATION_NAME,
            aApplicationName );
        if( aPlaceholderId != null ) setAttribute( PLACEHOLDER_ID,
            aPlaceholderId );
    }
}
```

イベントと XSD の間の相互参照：ClickCampaignEvent と XSD スキーマの間に相互参照があることに注目してください。この関連のおかげで、行動追跡データをデータベースに適切に記録することができるのです。

コード リスト 15-5　対応する XSD スキーマ

```
<xsd:schema

targetNamespace="http://www.bea.com/servers/commerce/xsd/tracking
/click-campaign/1.0.1"

xmlns="http://www.bea.com/servers/commerce/xsd/tracking/click-cam
paign/1.0.1"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
                        http://www.w3.org/2001/XMLSchema.xsd"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">
    <xsd:element name="ClickCampaignEvent">
        <xsd:complexType>
            <xsd:sequence>
               <xsd:element ref="application"/>
               <xsd:element ref="event-date"/>
               <xsd:element ref="event-type"/>
               <xsd:element ref="session-id"/>
               <xsd:element ref="user-id" minOccurs="0"/>
               <xsd:element ref="document-type" minOccurs="0"/>
               <xsd:element ref="document-id" minOccurs="0"/>
               <xsd:element ref="campaign-id"/>
               <xsd:element ref="scenario-id"/>
               <xsd:element ref="application-name" minOccurs="0"/>
               <xsd:element ref="placeholder-id" minOccurs="0"/>
            </xsd:sequence>
            <!-- types = banner-ad-promotion -->
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="application" type="xsd:string"/>
    <xsd:element name="event-date" type="xsd:string"/>
    <xsd:element name="event-type" type="xsd:string"/>
    <xsd:element name="session-id" type="xsd:string"/>
    <xsd:element name="user-id" type="xsd:string"/>
    <xsd:element name="document-type" type="xsd:string"/>
    <!-- types = banner-ad-promotion -->
    <xsd:element name="document-id" type="xsd:string"/>
    <xsd:element name="campaign-id" type="xsd:string"/>
    <xsd:element name="scenario-id" type="xsd:string"/>
    <xsd:element name="application-name" type="xsd:string"/>
    <xsd:element name="placeholder-id" type="xsd:string"/>
</xsd:schema>
```

キーの列挙：行動追跡イベントのソース コードには、イベント オブジェクトから XML インスタンス ドキュメントを作成するためのキーを、その順序どおりに列挙しておくことも必要です。たとえば、コード リスト 15-4 を参照してくださ

い。XSD ドキュメントの構造と、XML ネームスペースの詳細については、http://www.w3.org/XML/Schema を参照してください。BEA から提供される行動追跡イベントの XSD スキーマは、以下のファイル内の /lib/schema に収められています。

```
<BEA_HOME>\weblogic700\portal\lib\p13n\ejb\events.jar
```

ネームスペースとスキーマの指定：ネームスペースとスキーマは、以下のように指定します。

```
/**
      このイベントの XML ネームスペース
 */
private static final String XML_NAMESPACE=
    "http://<your URI>/testtracking";

/**
      このイベントのスキーマを記述した XSD ファイル
 */
private static final String XSD_FILE="TestTrackingEvent.xsd";
```

注意： これらの値は、インスタンス ドキュメントの作成時に、その中の各フィールドに値を設定するときに使用されます。

schemaKeys は、イベント クラスの getAttribute メソッドと setAttribute メソッドに渡されるキーとなる文字列のリストです。これらのキーは、行動追跡イベントを表現する XML インスタンス ドキュメント内の要素に設定されるデータを抽出するのに用いられます。キーは、文字列型のオブジェクトから成る配列内に列挙しなければなりません。キーの並びの順序は、XML インスタンス ドキュメントにおけるそれらのキーの出現順序を指定します。行動追跡システムによって生成される XSD ファイルでは、要素の順序は重要です。要素の順序が正しくないと、XSD ファイルでの XML ファイルの検証は失敗します。XML の numOccurs キーワードを使用し、その値をゼロに設定することで、要素を省略することができます。その方法の例については、BEA から提供される行動追跡イベントの XSD スキーマを参照してください。このスキーマは、以下のファイル内の /lib/schema に収められています。

```
<BEA_HOME>\weblogic700\portal\lib\p13n\ejb\events.jar
```

配列の定義：上記の行動追跡対応版 TestEvent の場合には、キーの配列は次のようになるでしょう。

```
/**
        このオブジェクトを表現する XML 内に存在する
```

```
        要素のキーとそれらの順序。
 */
private static final String localSchemaKeys[] =
{
    SESSION_ID, USER_ID, USER_PROPERTY_ONE_KEY,
        USER_PROPERTY_TWO_KEY
};
```

データ要素： SESSION_ID と USER_ID: は localSchemaKeys 配列内のデータ要素で、追跡イベントの実装に役立ちます。SESSION_ID は WebLogic Server のセッション ID で、セッション オブジェクトごとに作成されます（詳細については、http://edocs.beasys.co.jp/e-docs/wls/docs70/index.html にアクセスして WebLogic Server マニュアルを参照のこと）。USER_ID フィールド（null の場合もある）は、イベントの発生元のセッションに関連付けられている Web サイト訪問者のユーザ名です。一部のイベントでは、ユーザがイベントに関連付けられていない場合があります。その場合には、先述のように、XSD ファイル内の USER_ID フィールドの numOccurs の値をゼロにする必要があります。イベントを EVENT テーブルに永続化するには、SESSION_ID は null 以外のものでなければなりません。

その他の属性： すべての行動追跡イベントは、com.bea.p13n.tracking.events.TrackingEvent クラスを拡張する必要があります。このクラスには、どの追跡イベントの場合にも属性を設定するのに役立つ、以下の 3 つのキーが定義されています。

- TrackingEvent.SESSION_ID
- TrackingEvent.USER_ID
- TrackingEvent.REQUEST

これらのキーは、TrackingEvent コンストラクタにおいて SESSION_ID、USER_ID、および REQUEST（HttPServletRequest オブジェクト）の値をそれぞれ設定する際に行われる setAttribute の呼び出しで使用されます。また、これらのキーは、TrackingEvent を拡張するイベント オブジェクトに対して Event.getAttribute (String Key) メソッドを呼び出す際に、各キーに関連付けられている値の取得にも使用しなければなりません。

基本クラス TrackingEvent のコンストラクタを作成する

基本クラス TrackingEvent のコンストラクタは、Event クラスのコンストラクタよりも複雑です。Event のコンストラクタは、TrackingEvent のコンストラクタでの super(String eventType) の呼び出しによって呼び出されます。TrackingEvent のコンストラクタを、コード リスト 15-6 と コード リスト 15-7 に示します。

コード リスト 15-6　TrackingEvent のコンストラクタ — 例 1

```
/**
 * 新しい TrackingEvent を作成する
 *
 * @param theEventType イベントのタイプ
     * @param theSessionId HttpSession.getId() で取得
 *@param theUserId HttpServletRequest.getRemoteUser() または
 * それに相当するメソッドで取得 ( 不明の場合には null)
 * @param theXMLNamespace このイベント タイプの XML 表現のネームスペース

 * @param theXSDFile XML ファイル内のデータに対する型付けを指定し適用するスキーマを
 * 記述したファイル
* @param theSchemaKeys このイベントの XML に永続化されるデータを表すキーの
 * リスト ( 並びは XSD スキーマでの出現順序どおり )
      */
public TrackingEvent( String theEventType,
                      String theSessionId,
                      String theUserId,
                      String theXMLNamespace,
                      String theXSDFile,
                      String[] theSchemaKeys )
```

コード リスト 15-7 に示した TrackingEvent コンストラクタでは、HttpServletRequest オブジェクトを引数に取ります。

コード リスト 15-7　TrackingEvent コンストラクタ — 例 2

```
/**
 * 新しい TrackingEvent を作成する
```

```
 *
 * @param theEventType イベントのタイプ
     * @param theSessionId HttpSession.getId() で取得
 *@param theUserId HttpServletRequest.getRemoteUser() または
 * それに相当するメソッドで取得（不明の場合には null）
 * @param theXMLNamespace このイベント タイプの XML 表現のネームスペース

 * @param theXSDFile XML ファイル内のデータに対する型付けを指定し適用するスキーマを
 * 記述したファイル
* @param theSchemaKeys このイベントの XML に永続化されるデータを表すキーの
 * リスト（並びは XSD スキーマでの出現順序どおり）
 * @param theRequest HTTP サーブレット リクエスト オブジェクト
 */
public TrackingEvent( String theEventType,
                      String theSessionId,
                             String theUserId,
                      String theXMLNamespace,
                      String theXSDFile,
                      String[] theSchemaKeys,
                      HttpServletRequest theRequest )
```

コンストラクタについての補足：コード リスト 15-6 に示した最初のコンストラクタでは、null でもよいデータは theUerId だけで、それ以外のデータはすべて必須です。追跡イベントを EVENT テーブルに正しく永続化するためです。コード リスト 15-7 に示した 2 番目のコンストラクタでは、イベントの発生元で HttpServletRequest オブジェクトが利用可能な場合には、そこから、コンストラクタの引数として HttpServletRequest オブジェクトを渡すことができます。このオブジェクトには、イベント インスタンスに対してルールを適用するのに必要なデータが入っています。

注意： カスタム行動追跡イベントに対してルールを適用するには、HttpServletRequest と USER_ID が null 以外のものでなければなりません。USER_ID が null でないことは通常、訪問者が Web サイトにログイン済みであることを意味します。ユーザがいないときにトリガされたイベントに対して、ルールを適用することはできません。

TestTrackingEvent のコンストラクタをコード リスト 15-8 に示します。

コード リスト 15-8　TestTrackingEvent のコンストラクタ

```
/**
     * 新しい TestTrackingEvent を作成する
 *
     * @param theSessionId HttpSession.getId() で取得
 *@param theUserId HttpServletRequest.getRemoteUser() または
 * それに相当するメソッドで取得（不明の場合には null）
     * @param userPropertyOne String 型のユーザ定義プロパティ
     * @param userPropertyTwo Double 型の別のユーザ定義プロパティ
 */
    public TestTrackingEvent( String theSessionId,
                              String theUserId,
                              String userPropertyOneValue,
                              Double userPropertyTwoValue )
{
        super( TYPE, theSessionId, theUserId, XML_NAMESPACE, XSD_FILE,
               localSchemaKeys );

        if( userPropertyOneValue != null )
            setAttribute( USER_PROPERTY_ONE_KEY, userPropertyOneValue );

       if( userPropertyTwoValue != null )
           setAttribute( USER_PROPERTY_TWO_KEY, userPropertyTwoValue );

}
```

このコンストラクタは、TrackingEvent のコンストラクタを呼び出して必須属性に値を設定したあと、この特定の行動追跡イベント タイプに必要な属性を設定します。

TestTrackingEvent クラス全体をコード リスト 15-9 に示します。

コード リスト 15-9　TestTrackingEvent クラス

```
import com.bea.p13n.tracking.events.TrackingEvent;

/**
 * テスト目的のユーザ定義行動追跡イベント
 *
 * このイベントはデータベースへの永続化が可能
 *
```

```
*/
public class TestTrackingEvent
    extends TrackingEvent
{

    /** イベント タイプ */
    public static final String TYPE = "TestTrackingEvent";

    /**
      このイベントの XML ネームスペース
     */
    private static final String XML_NAMESPACE="http://<your URI>/testtracking";

    /**
      このイベントのスキーマを記述した XSD ファイル
     */
    private static final String XSD_FILE="TestTrackingEvent.xsd";

    /**
     * 1 つ目のユーザ定義プロパティのイベント属性キーの名前。
     * 属性値は String
     */
    public static final String USER_PROPERTY_ONE_KEY = "userPropertyOne";

    /**
     * 2 つ目のユーザ定義プロパティのイベント属性キーの名前。
     * 属性値は  Double
     */
    public static final String USER_PROPERTY_TWO_KEY = "userPropertyTwo";

    /**
       このオブジェクトを表現する XML 内に存在する
       要素のキーとそれらの順序。
     */
    private static final String localSchemaKeys[] =
    {
        SESSION_ID, USER_ID, USER_PROPERTY_ONE_KEY, USER_PROPERTY_TWO_KEY
    };

    /**
     * 新しい TestTrackingEvent を作成する
     *
     * @param theSessionId HttpSession.getId() で取得
 *@param theUserId HttpServletRequest.getRemoteUser() または
 * それに相当するメソッドで取得（不明の場合には null）
     * @param userPropertyOne String 型のユーザ定義プロパティ
     * @param userPropertyTwo Double 型の別のユーザ定義プロパティ
```

```
     */
    public TestTrackingEvent( String theSessionId,
                              String theUserId,
                              String userPropertyOneValue,
                              Double userPropertyTwoValue )
    {
        super( TYPE, theSessionId, theUserId, XML_NAMESPACE, XSD_FILE,
               localSchemaKeys );

        if( userPropertyOneValue != null )
            setAttribute( USER_PROPERTY_ONE_KEY, userPropertyOneValue );

       if( userPropertyTwoValue != null )
           setAttribute( USER_PROPERTY_TWO_KEY, userPropertyTwoValue );
    }
}
```

コード リスト 15-9 に示した TestTrackingEvent クラスでは、そのクラス独自の属性値を正しく設定するとともに、TrackingEvent をインスタンス化した部分の属性値を設定しています。これによって、このイベントを表現する XML インスタンス ドキュメントを作成する際に、ドキュメント内の属性値が正しく設定されるようになります。XML インスタンス ドキュメントは、データベースの EVENT テーブルに格納される TestTrackingEvent を表していることを思い出してください。

データベースへの永続化：カスタムの行動追跡イベント タイプをデータベースに永続化する場合には、application-config.xml ファイル内の behaviorTracking.persistToDatabase プロパティに、そのイベントを追加する必要があります。イベントを永続化しない場合には、イベント タイプをこのプロパティに追加する必要はありません。

行動追跡を有効にする方法

注意： Event サービスがアプリケーションのサービスとして存在しない場合には、WebLogic Server Administration Console を用いて追加します。

以下の手順では、行動追跡 (Behavior Tracking) をサンプル ポータルのサービスとして有効にする方法を示します。実際のアプリケーションの場合も、同様の手順に従うことになるでしょう。

1. WebLogic Server Administration Console で、以下のようにして、sampleportalDomain のノード ツリー内の [Behavior Tracking サービス] ページを開きます。

   Web ブラウザで http://<hostname>:<port>/console にアクセスし、Administration Console の左ペインで [sampleportalDomain ｜デプロイメント ｜アプリケーション｜ sampleportal ｜ Service Configuration ｜ Behavior Tracking Service] ノードを選択します。

2. 図 15-6 に示すように、[永続化イベント タイプ] フィールドにイベントの名前を入力します。

図 15-6 WebLogic Server Administration Console ― Behavior Tracking サービス

行動追跡イベントを XML に変換する

行動追跡イベントを EVENT テーブルに永続化する際には、イベント オブジェクトに格納されているデータの大半を XML に変換する必要があります。この XML ドキュメントは、XML インスタンス ドキュメントでの XML 要素の出現順序を指定する XML-XSD スキーマ（ユーザが作成する ）に準拠していなければなりません。さらに、このスキーマには、要素のデータ型とその多重度も定義する必要があります。イベント オブジェクトから XML ドキュメントを作成するプロセ

スは、行動追跡イベントのクラス ファイル内の変数と定数を利用するヘルパー クラスによって処理されます。すべてのスキーマ ドキュメントでは、ネームスペース「http://www.w3.org/2000/10/XMLSchema」が使用され、行動追跡スキーマのすべてのインスタンスでは、ネームスペース「http://www.w3.org/2000/10/XMLSchema-instance」が使用されます。作成される XML ドキュメント（コード リスト 15-10 に示す）は、XSD スキーマに準拠したものになります。

コード リスト 15-10　XSD ドキュメントの例

```
<xsd:schema
    targetNamespace="http://www.bea.com/servers/commerce/xsd/tracking/buy/1.0.1"
        xmlns="http://www.bea.com/servers/commerce/xsd/tracking/buy/1.0.1"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2001/XMLSchema
                            http://www.w3.org/2001/XMLSchema.xsd"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified">
        <xsd:element name="BuyEvent">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element ref="application"/>
                    <xsd:element ref="event-date"/>
                    <xsd:element ref="event-type"/>
                    <xsd:element ref="session-id"/>
                    <xsd:element ref="user-id" minOccurs="0"/>
                    <xsd:element ref="sku"/>
                    <xsd:element ref="quantity"/>
                    <xsd:element ref="unit-price"/>
                    <xsd:element ref="currency" minOccurs="0"/>
                    <xsd:element ref="application-name" minOccurs="0"/>
                    <xsd:element ref="order-line-id"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="application" type="xsd:string"/>
        <xsd:element name="event-date" type="xsd:string"/>
        <xsd:element name="event-type" type="xsd:string"/>
       <xsd:element name="session-id" type="xsd:string"/>
       <xsd:element name="user-id" type="xsd:string"/>
       <xsd:element name="sku" type="xsd:string"/>
       <xsd:element name="quantity" type="xsd:double"/>
       <xsd:element name="unit-price" type="xsd:double"/>
       <xsd:element name="currency" type="xsd:string"/>
```

```
      <xsd:element name="application-name" type="xsd:string"/>
      <xsd:element name="order-line-id" type="xsd:long"/>
</xsd:schema>
```

XML の作成：イベントの XML 表現の作成は、一般に、イベントのタイプに応じて行われます。そのため、正確な XML インスタンス ドキュメントを作成するには、各イベントで、ネームスペース、イベント タイプ、要素、および要素の出現順序を指定する必要があります。TestTrackingEvent の例では、TestTrackingEvent のインスタンスを表す XML は以下のように作成されるでしょう。

注意： ここでは、testTrackingEvent が TestTrackingEvent の整形式インスタンスであるものとします。

1. testTrackingEvent.getType() を呼び出して、イベントのタイプを取得します。
2. ((TrackingEvent)testTrackingEvent).getXMLNamespace() を呼び出して、イベントのネームスペースを取得します。
3. ((TrackingEvent)testTrackingEvent).getXSDFile() を呼び出して、イベントのネームスペースを取得します。

TestTrackingEvent クラスに定義されているスキーマ キーを用いて、XML ドキュメントに値が挿入されます。スキーマ キーと属性値のペアは、以下のような XML 要素に対応します。

```
<schema Key>value</schema Key>
```

行動追跡イベントの XML 表現を作成するヘルパー クラスでは、XML インスタンス ドキュメント内に挿入される要素のネストは深くないと仮定しています。さらに、Event クラスの getAttribute(String Key) メソッドの呼び出しを通じて取得される値オブジェクトの表現を作成するには、toString() メソッドが用いられます。値オブジェクトに対する toString() メソッドの呼び出しの結果返される文字列の内容は、イベントのスキーマ ドキュメントで指定されているデータ型に一致しなければなりません。TestTrackingEvent では、以下のキーを配列 schemaKeys で指定された順序で用いて、値を取得します。

- SESSION_ID
- USER_ID
- USER_PROPERTY_ONE_KEY

- USER_PROPERTY_TWO_KEY

これらのキーに対応する値は、testTrackingEvent.getAttribute(<schema Key>) の呼び出しで取得されます。XML 形式に書式化されたキーと値のペアがインスタンス ドキュメントに挿入される順序は、定数配列 schemaKeys で指定されます。この配列の定義と要素値の設定は、TestTrackingEvent クラスで行われます。

上記の手順によって作成された、TestTrackingEvent の XML インスタンス ドキュメントの例をコード リスト 15-11 に示します。

コード リスト 15-11　XML インスタンス ドキュメントの例

```
<TestTrackingEvent
    xmlns="http://<your URI>/testtracking"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
    xsi:schemaLocation="http://<your URI>/testtracking
TestTrackingEvent.xsd"
    >
<event_date>XML の時刻形式に書式化されたイベント発生日時 </event_date>
<event_type>TestTrackingEvent</event_type>
     <application>wlcsApp</application>
     <session_id>theSessionIdValue</session_id>
     <user_id>theUserIdValue</user_id>
     <userPropertyOne>userPropertyOneValue</userPropertyOne>
     <userPropertyTwo>userPropertyTwoValue</userPropertyTwo>
</TestTrackingEvent>
```

XML の作成は、イベントが com.bea.p13n.tracking.listeners.BehaviorTrackingListener に到着したときに自動的に実行され、それによって、WebLogic Portal における行動追跡が有効になります。行動追跡リスナは、application-config.xml ファイル内の <EventService Listeners="..."> プロパティに追加することでインストールされます。行動追跡リスナのインストール方法については、15-34 ページの「行動追跡を有効にする方法」を参照してください。

警告： カスタム行動追跡イベントが EVENT テーブルに永続化される場合には特に、そのイベント クラスにおけるネームスペース、XSD ドキュメント、およびスキーマ キー変数を定義する際に注意が必要です。上記で示した XML の作成および格納方法では、イベント クラスで指定されている変数と定数に厳密に従っています。XML を作成および格納するための方法を他にも自由に開発することができます。この節では、あらかじめ用意

されている EVENT テーブルに行動追跡イベントの XML 表現を永続化するプロセスだけに注目しています。

注意： イベントの日付は、Event クラスの getTimeStamp() メソッドを呼び出して取得されます。このメソッドは、Java の基本データ型である long 型の値を返します。この long 型の値は、XSD スキーマ ドキュメントで指定されている event_date 要素のデータ型に変換されなければなりません。この場合のデータ型は時刻です。BehaviorTrackingListener を通じて作成されるすべての XML インスタンス ドキュメントの最初の 2 つの要素は、イベントの日付とイベント タイプです。

カスタム行動追跡イベント リスナを作成する

カスタム行動追跡リスナを作成して、デフォルトの BehaviorTrackingListener の他に追加するか、あるいはその代わりに用いるには、15-17 ページの「カスタム イベント リスナを作成する」で示した例に従います。カスタム リスナの eventTypes 配列に新しいイベント タイプ（たとえば、TestTrackingEvent）を追加します。与えられたリスナは、行動追跡イベントであるかないかにかかわらず、任意の数のイベント タイプをリスンすることができます。カスタム行動追跡リスナは、同期型リスナか非同期型リスナとして（どちらか適切なほうを選ぶ）、Event サービスにインストールすることができます。

カスタム イベント ジェネレータを作成する

イベント クラスを作成したら、イベントを生成するためのメカニズムをアプリケーションにセットアップする必要があります。イベントの生成元になり得るのは、Pipeline コンポーネント、入力プロセッサ、JSP スクリプトレット、あるいは JSP タグです。行動追跡イベントの中には、WebLogic Portal ソフトウェア内部から生成されるものもあります。

イベント生成メカニズムを決定したら、`com.bea.p13n.tracking.TrackingEventHelper` クラスを用いて、行動追跡イベントをイベント システムに送信することができます。このクラスには、イベントを Event サービスに渡すヘルパー メソッドが定義されています。`TestTrackingEvent` の受け渡しの例をコード リスト 15-12 に示します。

コード リスト 15-12　イベントのディスパッチ

```
/*
 * イベントを作成する
 */
Event theEvent = new TestTrackingEvent( "<some session id>",
                                        "<some user id> ",
                                        new String("userPropertyOneValue"),
                                        new Double( 3.14 ) );

/*
    * イベントをディスパッチする
 */
EventService eventService = TrackingEventHelper.getEventService();
TrackingEventHelper.dispatchEvent( eventService, theEvent );
```

イベントのディスパッチ：Event サービスは EJB なので、イベントをディスパッチする前に、Event サービスを WebLogic Server インスタンスで稼働させておく必要があります。

複数のイベントをディスパッチする場合には、Event サービスのインスタンスを 1 つ取得し、以下のコードに示すように、アプリケーション クラス内の 1 つの属性として保存して再利用するのが一番よいでしょう。

```
/**
    * Event サービスにアクセスし、起動する。
*/
private EventService eventService =
com.bea.p13n.tracking.TrackingEventHelper.getEventService ( );
```

注意： イベント ディスパッチ用の API には 3 通りあります。そのうちのどれを用いればよいかについては、『Javadoc』(http://edocs.beasys.co.jp/e-docs/wlp/docs70/javadoc/index.html) を参照してください。

では、以下のように、Event サービスの上記インスタンスを用いてイベントをディスパッチしてみましょう。

```
/**
    * イベントをディスパッチする
*/
EventService eventService = TrackingEventHelper.getEventService();
TrackingEventHelper.dispatchEvent ( eventService, theEvent )
```

Event サービスのデバッグ

Event サービスをデバッグするには、以下のディレクトリに debug.properties ファイルを作成します。

```
<BEA_HOME>\weblogic700\portal\config\<YourDomain>\debug.properties
```

このファイルの内容は、コード リスト 15-13 に示すとおりです。

コード リスト 15-13　Event サービスのデバッグ

```
usePackageNames: on

# events 下のすべてのクラスについてデバッグを有効にする
com.bea.p13n.events: on
# com.bea.p13n.events.internal.EventServiceBean: on

# tracking 下のすべてのクラスについてデバッグを有効にする
com.bea.p13n.tracking: on

# あるいは、クラスを選んで、それらのデバッグを有効にする
com.bea.p13n.tracking.internal persistence: on
com.bea.p13n.mbeans.BehaviorTrackingListener: on
com.bea.p13n.tracking.listeners.BehaviorTrackingListener: on
com.bea.p13n.tracking.SessionEventListener: on
```

カスタム イベントの登録

この節では、カスタム イベントの登録（カスタム イベントについての背景的情報を含む）、BEA E-Business Control Center でイベント エディタを用いてイベントを登録する方法、およびカスタム イベントの修正時に必要な作業についての基本事項を説明します。

注意： WebLogic Portal にあらかじめ用意されている標準イベントは、どれも変更することはできません。

カスタム イベントの作成は、複数のステップから成るプロセスです。以下のリストは、このプロセスの概要を示したものです。

注意： ステップ 1 と 2 はすでに完了しているはずです。

1. イベントとイベント リスナを定義するコードを作成します。
2. JSP タグまたは API 呼び出しを用いて、イベントをトリガするコードを作成します。
3. この節での説明に従って、イベントを登録します。
4. 行動追跡情報の解析用にイベント データを記録するには、WebLogic Server Administration Console を用いてイベントを Event サービスに追加し、そのイベント用のエントリを EVENT_TYPE テーブルに作成します。

イベントを登録するタイミング

キャンペーンで用いるカスタム イベントを作成する際には、そのイベントを登録する必要があります。イベントをキャンペーンで使用しない場合には、登録の必要はありません。カスタム イベントを登録することで、そのカスタム イベントの存在が E-Business Control Center に認識されます。また、登録すれば、キャンペーン開発者が E-Business Control Center を用いて、そのイベントを参照するシナリオ アクションを作成できるようになります。さらに、登録によって、イベントのプロパティが識別されます。

警告： イベント コードを変更するたびに、イベント登録を更新する必要があります。逆に、イベント登録を変更するたびに、イベント コードも更新す

る必要があります。イベントを修正した結果、そのイベントのプロパティを参照するシナリオ アクションの修正が必要になる場合があります。

イベント プロパティ

E-Business Control Center のイベント エディタを利用すれば、カスタム イベントをたやすく登録することができます。イベントを登録する際には、イベント プロパティを名前と値のペアとみなすことができます。カスタム イベントの登録時には、イベントの名前、概要、および 1 つ以上のプロパティを指定します。各プロパティには、値の範囲、取り得る値のデータ型、およびデフォルト値があります。イベントの登録に必要な情報は、コマース ビジネス エンジニア (CBE) や Java 開発者から入手できるはずです。

カスタム イベントのプロパティには、以下の情報が含まれています。

- [**データ型**]: プロパティのデータ型を指定する。取り得る値は、テキスト、数値、浮動小数点数、ブール、および 日時のいずれか。
- [**選択モード**]: プロパティのデフォルト値が単数か複数かを指定する。
- [**値の範囲**] デフォルトが 1 つの特定値に限定されるのか、任意の数の特定値に限定されるのか、それとも任意の値でよいのかを指定する。

注意 : プロパティ値を設定する際には、プロパティが実行時にこれらの制限に従うかどうかは保証されません。`SchemaManager` では、イベントをチェックして、プロパティ スキーマに従っているかどうかを調べることはありません。そのため、イベント タイプ定義とイベント登録の同期を保つ必要があります。

上記リストからもわかるように、プロパティ値の組み合わせも可能です。プロパティの有効な組み合わせは、以下のとおりです。

- [**ブール**]: このタイプのプロパティの値は、True か False のどちらか。デフォルトを選ぶことができる。デフォルト値は、[プロパティの編集] ではなく [プロパティ値の入力] ウィンドウにのみ表示される。このデータ型が選択されると、[選択モード] と [値の範囲] は利用できない。
- [**単値**]、[**制限なし**]: このタイプのプロパティは値を 1 つだけ持ち、それがデフォルト値でもある。

- [**単値**]、[**制限付き**]: このタイプのプロパティは複数の値と単一のデフォルト値を持つ。どの値をデフォルト値にするか選択できる。
- [**多値**]、[**制限付き**]: このタイプのプロパティは複数の値を持つ。任意の数の値をデフォルト値として選択できる。
- [**多値**]、[**制限なし**]: このタイプのプロパティは複数の値を持つ。デフォルトの選択はできない。すなわち、すべての値がデフォルトになる。

カスタム イベントの登録手順

カスタム イベントを登録するには、以下の手順を実行します。

1. E-Business Control Center を起動します。図 15-7 に示すような エクスプローラ ウィンドウが開きます。

図 15-7　E-Business Control Center ウィンドウ

2. プロジェクトを開きます。詳細については、E-Business Control Center のオンライン ヘルプを参照してください。
3. エクスプローラ ウィンドウで、[サイト インフラストラクチャ] タブを選択したあと、[イベント] アイコンをクリックします。イベントのリストが [イベント] フィールドに表示されます。
4. [**新規作成**] アイコンをクリックしたあと、[イベント] を選択します。図 15-8 に示すようなイベント エディタ ウィンドウが表示されます。

図 15-8　イベント エディタ ウィンドウ

5. イベント エディタ ウィンドウで、[**新規作成**] ボタンをクリックします。図 15-9 に示すような [プロパティの編集] ウィンドウが開きます。

図 15-9　[プロパティの編集] ウィンドウ

6. [プロパティの編集] ウィンドウで、以下の手順を実行します。

   a. [名前] フィールドに、イベントのユニークな名前を 100 文字以内で入力します（必須）。

   b. [概要] フィールドに、イベントの概要を 254 文字以内で入力します（省略可能）。

   c. 各ドロップダウン リストからプロパティ値の [データ型]、[選択モード]、および [値の範囲] を選択します。

d. [**値の追加**] ボタンをクリックします。表示されるダイアログ ボックスはプロパティによって決まります。

e. 適切な値を入力し、（必要であれば）デフォルトを選択します。

f. イベントのプロパティ値の入力が完了したら、[**OK**] ボタンをクリックします。

7. イベントを保存します（E-Business Control Center のメイン メニューから [ファイル｜保存] を選択する）。

登録済みカスタム イベントを更新する

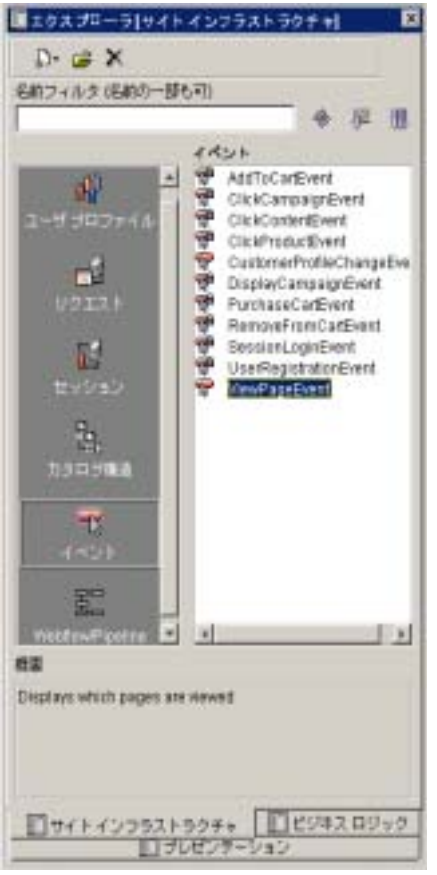
カスタム イベントのコードに変更を加えるたびに、そのイベントの登録を更新しなければなりません。登録を更新すれば、カスタム イベントの変更が E-Business Control Center に認識され、キャンペーン開発者が E-Business Control Center を用いて、そのイベントを参照するシナリオ アクション（キャンペーン内に定義）をすべて修正するのに役立ちます。

カスタム イベントを更新するには、以下の手順を実行します。

1. E-Business Control Center を起動します。エクスプローラ ウィンドウが開きます。
2. プロジェクトを開きます。詳細については、E-Business Control Center のオンライン ヘルプを参照してください。
3. エクスプローラ ウィンドウで、[サイト インフラストラクチャ] タブを選択したあと、[**イベント**] アイコンをクリックします。図 15-10 に示すように、イベントのリストが [イベント] フィールドに表示されます。

   注意： 標準イベントは編集できません。

図 15-10　エクスプローラ ウィンドウ



4. 編集したいカスタム イベントをダブルクリックします。図 15-11 に示すようなイベント エディタ ウィンドウが開きます。[イベント プロパティ] フィールドに、既存プロパティのリストが表示されます。

図 15-11　イベント エディタ ウィンドウ

5. 編集したいプロパティを選択したあと、[**編集**] ボタンをクリックします。図 15-12 に示すような [プロパティの編集] ウィンドウが開きます。

図 15-12　[プロパティの編集] ウィンドウ

6. 適切な変更を加えたあと、[**OK**] ボタンをクリックします。
7. イベントを保存します（E-Business Control Center のメイン メニューから [ファイル｜保存] を選択する）。

行動追跡のアクティブ化

オンライン訪問者が Web サイトとどのようにやり取りしているかを記録するために、イベント情報をデータベースに記録することができます。こうした種類のイベントは行動追跡イベントと呼ばれます。e 解析システムや e マーケティング システムでは、これらのイベントをオフラインで分析して、訪問者の行動とトランザクション データを評価することができます。

注意： イベント データを記録できるようにデータベースをコンフィグレーションする方法については、『管理者ガイド』の「行動追跡データの永続化」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/sysadmin.htm#1194894) を参照してください。

この節では、以下のトピックを扱います。

- 行動追跡をアクティブ化する手順
- WebLogic Server における Behavior Tracking サービスをコンフィグレーションする
- データ ソースをコンフィグレーションする

行動追跡をアクティブ化する手順

行動追跡イベントをデータベースに記録できるようにするには、まず、行動追跡リスナを有効にする必要があります。それには、リスナ クラスを追加します。

注意： Event サービスがアプリケーションのサービスとして存在しない場合には、WebLogic Server Administration Console を用いて追加します。

以下の手順では、リスナ クラスをサンプル ポータルに追加する方法を示します。実際のアプリケーションの場合も、同様の手順に従うことになるでしょう。

1. WebLogic Server Administration Console で、以下のようにして、sampleportalDomain のノード ツリー内の [同期型リスナ] タブまたは [非同期型リスナ] タブを開きます。

   Web ブラウザで http://<hostname>:<port>/console にアクセスし、Administration Console の左ペインで [sampleportalDomain ｜デプロイメ

ント｜アプリケーション｜ sampleportal ｜ Service Configuration ｜ Event Service] ノードを選択したあと、右ペインの [コンフィグレーション] タブから [同期型リスナ] タブをクリックします。

2. [追加すべきリスナ クラス] フィールドに行動追跡リスナ (`com.bea.p13n.tracking.listeners.BehaviorTrackingListener`) を追加したあと、[**追加**] ボタンをクリックします。図 15-13 を参照してください。

図 15-13 WebLogic Server Administration Console ― Event サービス

注意： 行動追跡をアクティブにする前に、データベースをコンフィグレーションする必要があります。その方法については、『管理者ガイド』の「行動追跡データの永続化」(http://edocs.beasys.co.jp/e-docs/wlp/docs70/admin/sysadmin.htm#1194894) を参照してください。

WebLogic Server における Behavior Tracking サービスをコンフィグレーションする

行動追跡イベントはいったんバッファに格納されたあと、データベース内のイベント用テーブルに断続的に永続化（記録）されます。データベースに格納されたイベントは、オフラインで分析することができます。非同期型サービスが使用されるのは、イベント ハンドラの実行に時間がかかる場合でも、Web サイト訪問者の目から見てアプリケーションの動作に遅延が生じないようにするためです。

注意： 行動追跡イベントの各プロパティは、WebLogic Server Administration Console でコンフィグレーションする必要があります。

接続プール：バッファに格納された行動追跡イベントは、データ接続プールを用いてデータベースにスイープされます。デフォルト データ ソースは `weblogic.jdbc.jts.commercePool` です。別のデータ ソースを使用することもできます。それには、新しいデータ ソースを作成およびコンフィグレーションし（15-53 ページの「データ ソースをコンフィグレーションする」を参照）、WebLogic Server Administration Console で、デフォルト データ ソースの名前を新しいデータ ソースの名前に置き換えます。

プロパティ：データベースに永続化される特定のイベントは、`PersistEventTypes` プロパティで指定します。WebLogic Server Administration Console で、永続化されるイベントのリストを確認したり変更することができます。このリスト内のタイプは、イベントで指定されたタイプと一致する必要があります。たとえば、`SessionBeginEvent` のタイプを表す文字列は “`SessionBeginEvent`” です。

パフォーマンスの最適化：バッファ内のイベントをスイープする頻度は、Behavior Tracking サービスの以下のプロパティで指定します。

- `MaxBufferSize`
- `SweepInterval`
- `SweepMaxTime`

これらのプロパティを調整して、パフォーマンスを最適化しなければなりません。バッファ スイープの実行は、データベースへの書き込みに時間がかかりすぎるほど頻度が低くてもいけませんが、書き込み操作が無駄になるくらい頻繁であってもいけません。

手順：Behavior Tracking サービスをコンフィグレーションするには、以下の手順に従います。

注意： 以下の手順では、サンプル　ポータルのパフォーマンスを最適化する方法を示します。実際のアプリケーションの場合も、同様の手順に従うことになるでしょう。

Event サービスがアプリケーションのサービスとして存在しない場合には、WebLogic Server Administration Console を用いて追加します。

注意： アプリケーションに Behavior Tracking サービスと Event サービスが用意されていない場合には、WebLogic Server Administration Console を用いて、それらを追加します。

1. WebLogic Server Administration Console で、以下のようにして、sampleportalDomain のノード ツリー内の [JDBC データソース ファクトリ] ページ（図 15-13 に示す）を開きます。

   Web ブラウザで http://<hostname>:<port>/console にアクセスし、Administration Console の左ペインで [sampleportalDomain ｜デプロイメント ｜アプリケーション｜ sampleportal ｜ Service Configuration ｜ Behavior Tracking Service] ノードを選択します。

図 15-14　WebLogic Server Administration Console ― Behavior Tracking サービス

2. データ ソースを変更するには、[データ ソースの JNDI 名] フィールドにデータ ソースの完全修飾名を入力します。
3. バッファからのイベント スイープの設定を変更するには、該当するフィールドにバッファの新しい設定値を入力します。
4. ある特定のイベントを永続化するかどうかを指定するには、[永続化イベント タイプ] リスト ボックスで、そのイベントを追加または削除します。

データ ソースをコンフィグレーションする

この節では、サンプル ポータルでイベントの永続化に用いられる接続プールの新しいデータ ソースのコンフィグレーションについて簡単に説明します。実際のアプリケーションの場合も、同様の手順に従うことになるでしょう。

新しいデータ ソースをコンフィグレーションするには、以下の手順に従います。

注意： WebLogic Server Administration Console の使い方の詳細については、WebLogic Server マニュアル (http://edocs.beasys.co.jp/e-docs/wls/docs70/index.html) を参照してください。

1. WebLogic Server Administration Console で、以下のようにして、sampleportalDomain のノード ツリー内の [JDBC データソース ファクトリ] ページ（図 15-13 に示す）を開きます。

   Web ブラウザで http://<hostname>:<port>/console にアクセスし、Administration Console の左ペインで [sampleportalDomain ｜サービス｜ JDBC ｜ JDBC データ ソース ファクトリ] ノードを選択します。

図 15-15　WebLogic Server Administration Console ― 新しい JDBCDataSourceFactory の作成

2. 右ペインで、[**新しい JDBCData Source Factory のコンフィグレーション**] をクリックします。
3. 該当するタブの該当するフィールドに新しいデータ ソースの適切な設定値を入力します。

第 16 章 Expression パッケージの使用

ここでは、Expression パッケージのサービスの使い方について説明します。Expression パッケージは、WebLogic Portal のパーソナライゼーションおよび対話管理機能の一部です。Expression パッケージを利用すると、計算、ビジネス ポリシー、決定木、その他の処理を Java コードの外に出すことができます。

この章では、以下の内容について説明します。

- Expression パッケージとは
- 式の組み立てと管理
- 式の扱い
- Expression パッケージの設定

Expression パッケージとは

前述のとおり、Expression パッケージを利用すると、ビジネス ロジックや数式を Java コードの外に出すことができます。Expression パッケージを使用すると、任意の数式、ブール式、関係式、または条件式を表現できます。Expression パッケージを使用して、独自のビジネス ロジックを動的に組み立てたり評価できます。

Expression パッケージの使用例としては、レンタカーの取次店が考えられます。頻繁に変更されるレンタル料を計算するために、Expression パッケージを使用できます。Java のステートメントを使用して計算する代わりに、計算部分を XML ドキュメントとして、Java コードの外に出し、実行時に解釈します。

WebLogic Portal では、式の例を提供しています。例を表示するには、次の手順を実行します。

1. 次のようにして、Personalization サーバを起動します。

[スタート｜プログラム｜ BEA WebLogic Platform 7.0 ｜ WebLogic Portal 7.0 ｜ Portal Examples ｜ Personalization Examples ｜ Launch Personalization Server]

2. Personalization サーバが起動したら、次のようにして、Personalization Examples を起動します。

   [スタート ｜ BEA WebLogic Platform 7.0 ｜ WebLogic Portal 7.0 ｜ Portal Examples ｜ Personalization Examples ｜ Start Personalization Examples]

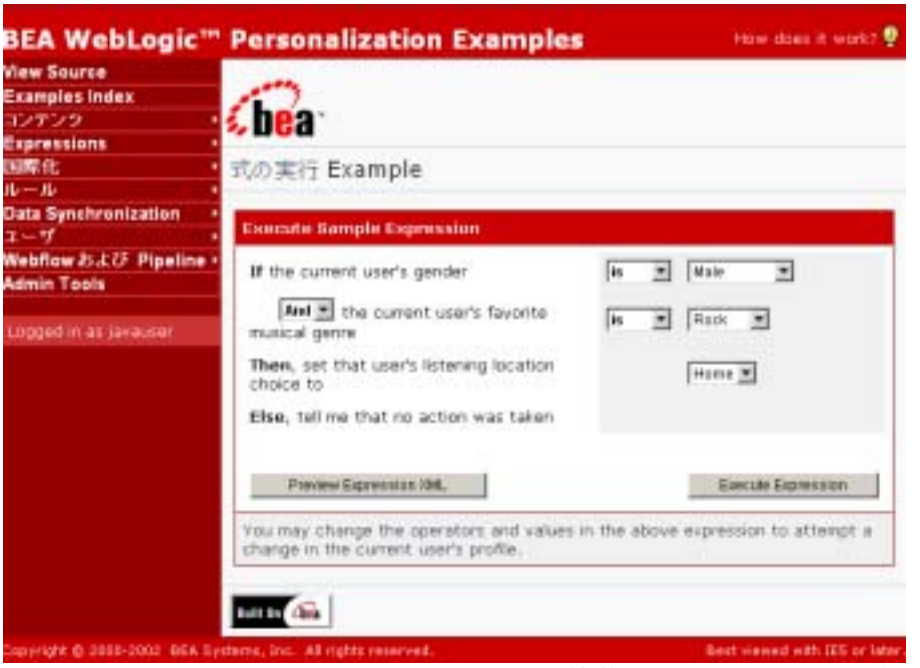
   ブラウザには、図 16-1 に示すような [Personalization Examples Index] が表示されます。

図 16-1　Personalization Examples Index

3. ログインしていない、またはユーザを作成していない場合は、**[Please visit the User Login example first]** をクリックします。ログイン、またはユーザ作成用の別ページに移動するためのページが表示されます。

4. ログイン、またはユーザ作成が完了したら、ページの左側のカラムから、[Expressions ｜式の実行] を選択します。図 16-8 に示すような [式の実行 Example] ページが表示されます。

図 16-2 [式の実行 Example]



このウィンドウには、Expression パッケージの簡単なアプリケーションが表示されています。パラメータは、ドロップ ダウン リストを使って設定できます。

5. 左側のカラムの **[View Source]** をクリックします。新しいページが開いて、図 16-3 に示すような JSP ソースが表示されます。

図 16-3　JSP ソース

このページには、Expression Example の JSP ソース (exec_expression.jsp) が表示されています。この式が実行される（および、パラメータが変更される）と、その結果が exec_expression_results.jsp に保存されます。どちらの JSP ファイルも <BEA_HOME>\weblogic700\samples\portal\p13nDomain\beaApps\p13nApp\p13n ディレクトリにあります。

次のステップでは、式がどのように動作するかについての情報を表示します。

6. **[式の実行 Example]** ページの **[How does it work?]** リンクをクリックします。新しいブラウザ ウィンドウが開き、図 16-4 に示すような説明が表示されます。

図 16-4　[How Does It Work?] 式の実行

このページは、Expression Example の動作の仕組みについての説明です。ここには、式に対する理解を深めるための練習も含まれています。

7. **[式の実行 Example]** ページの **[Preview Expression XML]** ボタンをクリックします。図 16-5 に示すような XML を表すページが表示されます。

図 16-5　[Preview Expression as XML]

このページには、実行前の Expression Example の XML が表示されています。

次の節では、Expression パッケージとルール フレームワークの違いについて説明します。

ルールまたは式の使用

ルール マネージャを適用する目的の 1 つは、ビジネス ルールを使用して、ユーザやグループを適切なコンテンツに結び付けることです。ルール マネージャは、Expressions パッケージと同様に、WebLogic Portal パーソナライゼーションおよび対話管理の一部です。

Expression パッケージとルール マネージャの最大の違いは、Expression パッケージが名前付きの変数を使用するのに対して、ルール エンジンは名前付きの変数を使用しない点です。さらに、ルール マネージャはルール セットを使用します。1 つのルールが別のルールを起動するというように、ルールを連結させることができます。

一般に、変数に値（通常は 1 つだけ）をバインドしたい場合は、式を使用します。パターンを検出して、変数に対して考えられるすべてのバインドを評価したい場合は、ルールを使用します。

ルール エンジンは、非常に強力なパターン マッチング機能と推論機能を持っています。ただし、これらの機能を使用すると、パフォーマンスが低下します。名前付きのルールを繰り返し実行していることに気付いた場合は、そのルールを式に変換することを検討してください。ルール エンジンの推論機能を利用しない場合や、考えられる変数と値のバインドを複数利用しない場合は、式を使用します。

式を使用して、UnificationList やカスタム Unifier によって明示的に変数と値をバインドする場合と、ルール エンジンを使用して、可能性のあるバインドをすべて探索する場合のパフォーマンスの違いを注意深く評価する必要があります。

表 16-1 は、ルールと式を使用する場合の例を示しています。

表 16-1 式とルールの比較

特徴	式	ルール
ビジネス ロジックを Java コードの外部に出す	はい	はい
アプリケーション コードと独立に、ビジネス ロジックを素早くデプロイできる	はい	はい
プログラマでなくても、JSP や Swing GUI を使用してビジネス ロジックを組み立てられる	はい	はい
推論機能	いいえ	はい 1 つのルールの発火によって、別のルールを発火させることができる。

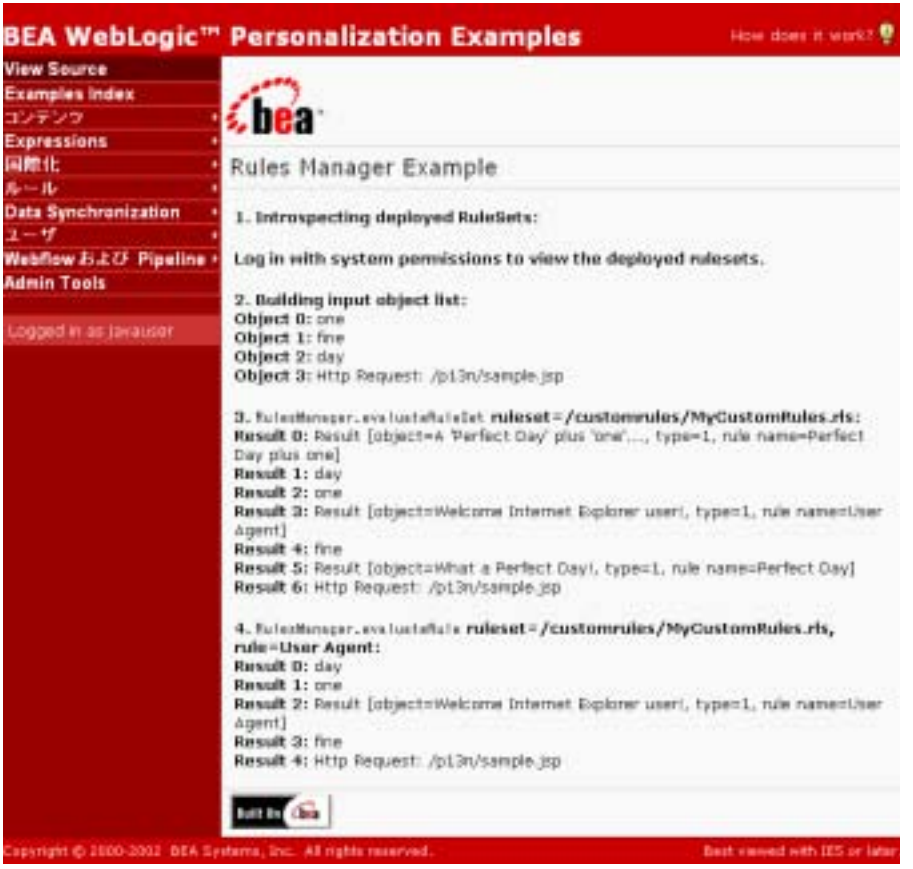
表 16-1 式とルールの比較

特徴	式	ルール
変数に値を明示的にバインドする	はい	いいえ 値は、クラス タイプを使用して変数にバインドされる。可能性のあるすべてのバインドが自動的にテストされる。
ビジネス ロジックの永続期間が長い	はい	はい ビジネス ロジックは、XML ドキュメントとして保持される。XML スキーマによって、Java コードからの独立性が保たれる。
ビジネス ロジックをプロセス間で渡すことができる	はい	はい ビジネス ロジックを定義した XML ドキュメントは、シリアライズまたは Web サービス間で渡すことができる。
XML 解析のキャッシュ	いいえ	はい ルール マネージャは、ルールセット ドキュメント用に TTL キャッシュを実装している。
式のキャッシュと最適化	はい	はい ルール エンジンは、内部で Expression パッケージを使用してるため、その最適化を利用している。

ルールの例を表示するには、次の手順を実行します。

1. [Personalization Examples] ウィンドウで、[ルール｜ Rules Manager] を選択します。図 16-6 に示すような [Rules Manager Example] が表示されます。

図 16-6 [Rules Manager Example]



このページには、ルールの一例が表示されます。この例は、あるルールのアクションが、別のルールの条件を満たすことを示しています。この機能は、Expression パッケージにはありません。

2. 詳細については、**[How does it work?]** リンクをクリックしてください。図 16-7 に示すような説明が表示されます。

図 16-7　[How Does It Work?] ルール マネージャ

Expression パッケージのクラス

Expression パッケージを利用すると、ユーザは、XML ベースの式を動的に組み立てて実行できます。このパッケージには、様々な種類の演算子を表す Java クラスが定義されています。また、これらの演算子のインスタンスで構成された式を評価するためのサービスも含まれています。

Expression パッケージには、基本 Expression クラス、Variable クラス、および式と変数を操作するための次のような演算子クラスが含まれています。

- 基本言語演算子（オブジェクトの生成、メソッド呼び出しなど）
- 論理演算子
- 比較演算子
- 集合演算子
- 算術演算子
- 文字列演算子

Expression パッケージには、式を操作するための次のようなサービスも含まれています。

- Unifier— 式を評価するための準備をする。
- Validator— 評価の前に、式の形式が正しいかどうか検証する。
- Optimizer— 評価の前に、式の構造を最適化する。
- Evaluator— 式を評価して、評価結果を返す。
- Executor— 統合、検証、評価のプロセスを組み合わせた集約サービス。

Java 内に直接記述され、Java プログラム内で実行される式とは異なり、Expression パッケージを利用すると、Java プログラム内から、式を動的に組み合わせたり変更できます。式は、評価の前でも後でも、何回でも変更できます。Expression パッケージを使用して式を組み立てると、式のキャッシュ、検証、最適化など、Expression パッケージのさらに高度な機能を利用できます。

Expression パッケージは、BEA ルール エンジンの基盤としての役割を果たしています。ルール エンジンは、ルールの条件式とアクション式を評価するために、このパッケージを利用しています。同様に、Expression パッケージを使用して、独自のビジネス ロジックを動的に組み立てたり評価できます。

Expression パッケージのパッケージ構成

Expression パッケージのインタフェースと抽象クラスは、com.bea.p13n.expression パッケージにあります。

Expression パッケージの演算子は、次のパッケージにまとめられています。

基本言語演算子 —com.bea.p13n.expression.operators
論理演算子 —com.bea.p13n.expression.operators.logical
文字列演算子 —com.bea.p13n.expression.operators.string
算術演算子 —com.bea.p13n.expression.operators.math
比較演算子 —com.bea.p13n.expression.operators.comparative
集合演算子 —com.bea.p13n.expression.operators.collection

Expression パッケージ関連のクラスは、p13n_util.jar アーカイブにまとめられています。

式の組み立てと管理

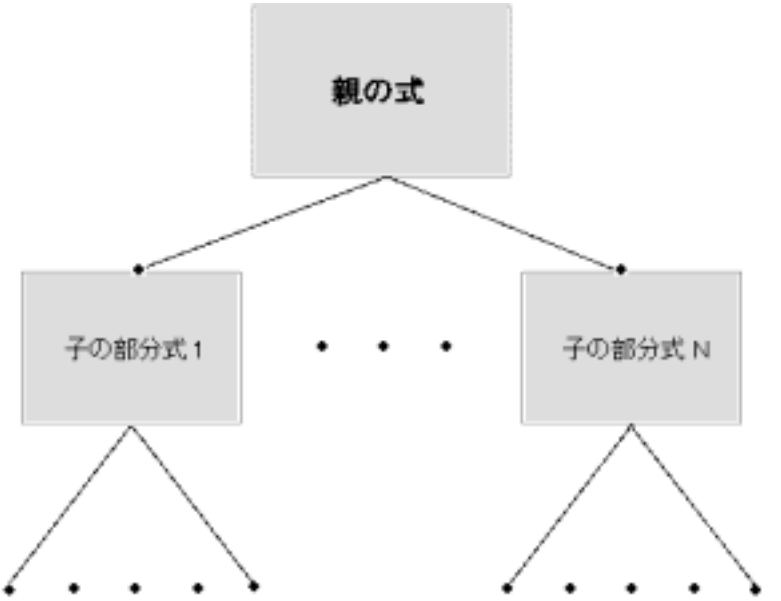
式を使い始める前に、Expression パッケージの様々な演算子クラスを使用して、式を体系的に組み立てる方法を学ぶ必要があります。

1 つの式は、1 つのツリーとして表現されます。このツリーの各ノードは、別の式、または単純な Java オブジェクトです。式を表すツリーは、ボトムアップに組み立てられます。最初に、子の式または Java オブジェクトが作成され、次に、親の式に追加されます。

図 16-8 は、式ツリーが構築されていく様子を表しています。

- 式を組み立てるプロセスの最初のステップは、1 つ以上の子の演算子または Java オブジェクトを作成することです。
- 次に、親の演算子が作成され、子の演算子または Java オブジェクトが親の演算子の構造に追加されます。
- 部分式を作成するこのプロセスは、式全体が組み立てられるまで継続されます。

図 16-8　式ツリーの構築



親子関係のメンテナンス

Expression パッケージで定義されている演算子クラスは、すべて共通の基本クラスを拡張したものです。この基本クラスには、親子関係をメンテナンスするために必要なロジックが含まれています。したがって、式の組み立て中に、親子関係のメンテナンスについて心配する必要はありません。ただし、式を作成した後で、式の構造を変更することができます。

表 16-2 は、1 つの式の中の部分式を追加、変更、削除するための Expression インタフェース内の操作を示しています。

表 16-2　式ツリーを構築するためのメソッド

Java メソッド	解説
addSubExpression	式オブジェクトに子（部分式になる）を追加する。

表 16-2 式ツリーを構築するためのメソッド

Java メソッド	解説
removeSubExpression	式オブジェクトからオブジェクト（部分式になる）を削除する。
setSubExpression	既存の（式の）オブジェクトを、与えられたオブジェクト（部分式になる）で置き換える。
getSubExpression	式オブジェクトの子にアクセスするために使用する。
getParent	式オブジェクトの親にアクセスするために使用する。

Expression インタフェースの詳細については、『*Javadoc* 』を参照してください。

Expression キャッシュの管理

Expression インタフェースには、結果のキャッシュを管理するメソッドも含まれています。式を評価した結果は、それぞれの式オブジェクト内にキャッシュされます。式に対するキャッシュが有効な場合は、同じ式を 2 回目に評価したときには、キャッシュされた値が返されます。

注意： デフォルトでは、キャッシュはオフになっています。MethodCall など、いくつかの演算子に対しては、キャッシュをオフのままにしておきたい場合があります。

表 16-3 は、結果のキャッシュを管理するための Expression インタフェース内のメソッドを示しています。

表 16-3 結果のキャッシュを管理するメソッド

Java メソッド	解説
setCacheEnabled	式のキャッシュを有効にしたり、無効にするために使用する。
isCacheEnabled	式のキャッシュが有効かどうかをチェックするために使用する。

表 16-3 結果のキャッシュを管理するメソッド

Java メソッド	解説
`isCached`	式の結果が現在キャッシュがされているかどうかをチェックするために使用する。
`getCachedValue`	式の評価結果として現在キャッシュされている値を取得するために使用する。

`Expression` インタフェースの詳細については、『*Javadoc* 』を参照してください。

式の扱い

式を組み立てたら、Expression パッケージの様々なサービスを使用して、式を扱うことができます。これらのサービスを利用すると、組み立てた式を評価するための準備、式の形式が正しいかどうかの検証、式の構造の最適化、そして最後に式の評価ができます。

この節では、以下のトピックを扱います。

- Expression Factory
- Expression パッケージのサービス
  - 統合サービス
  - 最適化サービス
  - 検証サービス
  - 評価サービス
  - 実行サービス

Expression Factory

ExpressionFactory は、Expression パッケージの様々なサービスや、そのサービスで使われるデータ構造を作成するメソッドを提供します。

たとえば、次のメソッドは、Validator サービスのインスタンスを作成します。

```
ExpressionFactory.createValidator(null);
```

Expression パッケージの様々なサービスの作成方法の詳細については、『*Javadoc* 』を参照してください。

Expression パッケージのサービス

Expression パッケージは、Expression パッケージ内の演算子を使用して構築された任意の式に対して使用できるサービスを提供しています。

統合サービス

Unifier は、式の中の変数を統合する（変数に値を割り当てる）ために使います。Unifier は、UnificationList と呼ばれるデータ構造を使用します。UnificationList には、変数名とその変数に対応する値が格納されています。Unifier と同様に、UnificationList のインスタンスは、ExpressionFactory によって作成されます。Unifier は、変数名をキーにして UnificationList を検索し、特定の変数の値を取得します。次に、取得した値をその変数にバインドします。

Unifier インタフェースと ExpressionFactory クラスの詳細については、『*Javadoc* 』を参照してください。

最適化サービス

Optimizer は、式を最適化するために使われます。Optimizer が使用するデフォルトの最適化アルゴリズムは次のとおりです。

- 式ツリーを走査し、すべての一意の部分式をリストに追加します。
- ある部分式が、このリスト内の式に等しい場合は、それを代理式で置き換えます。代理式は、元の式に処理を任せます。

Optimizer インタフェースと ExpressionFactory クラスの詳細については、『*Javadoc* 』を参照してください。

検証サービス

Validator は、式を検証するために使われます。Validator が使用するデフォルトの検証アルゴリズムは次のとおりです。

ある演算子のすべてのオペランドに対して

- オペランドに要求される型を取得します。
- オペランドが式の場合は、その式を評価し、その結果の型と要求される型を比較します。オペランドが式以外の場合は、そのオペランドが要求された型かどうかを検査します。
- オペランド式の評価中に、型が一致しなかったり、エラーが発生した場合は、Validator は、InvalidExpressionException を送出します。式の中の変数に値がバインドされていない場合は、UnboundVariableException を送出します。

Validator は、ステートレスまたはステートフル モードで使用できます。ステートレス モードの場合は、検証に必要なすべての式の評価は、ステートレス モードで実行されます。

ステートレスおよびステートフル評価モードの詳細については、次の節の「評価サービス」を参照してください。

Validator インタフェースと ExpressionFactory クラスの詳細については、『*Javadoc* 』を参照してください。

評価サービス

Evaluator は、式を評価するために使われます。式は、ステートフルまたはステートレス モードで評価できます。

ステートフル モード

このモードでは、式の中の各変数の値を、その変数内の値セットを取得することによって決定します。

つまり、ステートフル モードは、Unifier によって事前に統合されている式に依存します。

ステートフル モードで式が評価され、結果のキャッシュがオンの場合は、評価結果は式の内部にキャッシュされます。

ステートレス モード

このモードでは、式の中の各変数の値を、外部のデータ構造でその変数名にバインドされている値を検索することによって決定します。

つまり、評価プロセスは、式の状態に依存しません。したがって、評価の前に式を統合しておく必要がありません。

変数に対する名前と値のマッピングを含むデータ構造は、UnificationList と呼ばれ、Evaluator と関連付けられています。Evaluator と同様に、UnificationList のインスタンスは ExpressionFactory を使用して作成されます。

ステートレス モードの副作用は、式の評価の際に、結果のキャッシュが利用できないことです。

ステートフル モードは、シングル スレッドで式を評価する場合に使用できます。1 つの式をマルチスレッドで評価する場合は、ステートレス モードを使用*しなければなりません。*

注意： 式の中に変数が*含まれない*場合は、この 2 つの評価モードの間に違いはありません。

Evaluator インタフェースと ExpressionFactory クラスの詳細については、『*Javadoc* 』を参照してください。

実行サービス

Executor は、統合サービス、検証サービス、および 評価サービスを集約したものです。Executor の execute メソッドは、Unifier、Validator、および Evaluator を使用して、統合 - 検証 - 評価の一連のサイクルを実行します。

Executor が使用するデフォルトのアルゴリズムは次のとおりです。

統合

- Unifier が null でない場合は、式を統合します。
- Unifier が null の場合は、式を統合しません。

注意： Executor に渡される式が既に統合済みの場合や、式をステートレス モードで評価する場合は、Unifier を null にする必要があります。

検証

- Validator が null でない場合は、与えられた式を評価します。
- Validator が null の場合は、検証を無視します。

評価

- Evaluator が null でない場合は、（渡された Evaluator の種類に応じて）ステートフルまたはステートレス モードで式を評価します。
- Evaluator が null の場合は、Executor は、IllegalArgumentException を送出します。
- 結果を返します。

注意： 渡された Evaluator がステートレスの場合は、Unifier は null でなければなりません。

Executor インタフェースと ExpressionFactory クラスの詳細については、『*Javadoc*』を参照してください。

コード例

この節では、式を体系的に構築する方法と、Expression パッケージのサービスの使用方法を示す例を紹介します。

ここでは、以下のコード例を扱います。

- 単純な式のステートフル評価
- 変数を含む式のステートフル評価
- 変数を含む式のステートレスな検証と評価
- 変数を含む式のステートフルな検証と評価

単純な式のステートフル評価

論理式を作成して、ステートフル モードで実行します。この式には、変数が含まれません。

コード リスト 16-1　例

この式を作成して評価するためのソース コードを以下に示します。

```
Expression expression = new LogicalAnd(Boolean.TRUE,
Boolean.FALSE);

// Executor を作成するための準備として、ステートフルな
// Evaluator を作成します。この式には変数が含まれていないので、
// この例では、Validator や Unifier を使用しません。
// したがって、これらは作成しません。

// 環境 Map には null が渡されます。

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// 環境 Map には null が渡されます。

Executor executor = ExpressionFactory.createExecutor(null);

// Unifier パラメータと Validator パラメータの両方に null を渡して、
// 上の式を実行します。

Object result = executor.execute(expression, null, null,
evaluator);

// 結果は Boolean.FALSE になります。
```

変数を含む式のステートフル評価

変数を含む式を作成し、ステートフル モードで評価します。

コード リスト 16-2　例

式を作成して、ステートフル モードで実行するためのソース コードを以下に示します。

```
// Boolean 型のオブジェクトを格納する変数を作成します。
// 変数名は "booleanVariable" です。

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// これから、上のステップで作成した変数を使用します。

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);
```

```
// 次に、式の中のすべての変数をバインドし、式を統合します。
// 上の場合は、式の中に変数が 1 つあるので、
// この変数に値を割り当てる必要があります。
// その様子を以下に示します。

// UnificationList を作成し、変数名と値を
// キーと値のペアとして格納します。

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject("?booleanVariable", Boolean.FALSE);

// Unifier を作成します。

Unifier unifier = ExpressionFactory.createUnifier(null,
unificationList);

// Executor を作成するための準備として、ステートフルな
// evaluator. を作成します。この例では、Validator は使用しません。
// したがって、Unifier は作成しません。

// 環境 Map には null が渡されます。

Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// 環境 Map には null が渡されます。

Executor executor = ExpressionFactory.createExecutor(null);

// Unifier パラメータと、null の Validator を渡して、
// 上の式を実行します。

Object result = executor.execute(expression, unifier, null,
evaluator);

// 結果は Boolean.FALSE になります。
```

注意： execute メソッドを呼び出す前に、式を統合することもできます。それには、Unifier の unify メソッドを呼び出します。いったん式が統合された後は、execute メソッドに Unifier を渡す必要はありません。

変数を含む式のステートレスな検証と評価

変数を含む式を作成して、ステートレス モードで評価します。Validator サービスは、式を検証するために使われます。

コード リスト 16-3　例

式を作成して、ステートレス モードで実行するためのソース コードを以下に示します。

```
// Boolean 型のオブジェクトを格納する変数を作成します。
// 変数名は "booleanVariable" です。

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// これから、上のステップで作成した変数を使用します。

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// 次に、式の中のすべての変数をバインドし、式を統合します。
// 上の場合は、式の中に変数が 1 つあるので、
// この変数に値を割り当てる必要があります。
// その様子を以下に示します。

// UnificationList を作成し、変数名と値を
// キーと値のペアとして格納します。

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject(“?booleanVariable”, Boolean.FALSE);

// Executor を作成するための準備として、ステートレスな
// Evaluator を作成します。この例では、Unifier は使用しません。
// したがって、Unifier は作成しません。

// 環境 Map と UnificationList に null を渡して、
// ステートレスな Evaluator を作成します。

Evaluator evaluator = ExpressionFactory.createEvaluator(null,
unificationList);

// ステートレスな Validator を作成します。

Validator validator = ExpressionFactory.createValidator(null,
evaluator);

// Executor を作成します。

Executor executor = ExpressionFactory.createExecutor(null);

// Unifier には null を、Validator には null でない値を渡して、
// 上の式を実行します。

Object result = executor.execute(expression, null, validator,
evaluator)

// 結果は Boolean.FALSE になります。
```

```
// execute メソッドを呼び出した後は、上で使われたどのサービスも
// 与えられた式を変更できません。

// ステートレスな実行モードは、式を複数のスレッドで共有する場合に
// 役立ちます。
```

変数を含む式のステートフルな検証と評価

変数を含む式を作成し、ステートフル モードで評価します。Validator サービスは、式を検証するために使われます。

コード リスト 16-4　例

式を作成して、ステートフル モードで実行するためのソース コードを以下に示します。

```
// Boolean 型のオブジェクトを格納する変数を作成します。
// 変数名は "booleanVariable" です。

Variable booleanVariable = new Variable(“?booleanVariable”,
Boolean.class);

// これから、上のステップで作成した変数を使用します。

Expression expression = new LogicalAnd(Boolean.TRUE,
booleanVariable);

// 次に、式の中のすべての変数をバインドし、式を統合します。
//  上の場合には、式の中に変数が 1 つあるので、
// この変数に値を割り当てる必要があります。
//  その様子を以下に示します。

// UnificationList を作成し、変数名と値を
// キーと値のペアとして格納します。

UnificationList unificationList =
ExpressionFactory.createUnificationList(null);

UnificationList.addObject(“?booleanVariable”, Boolean.FALSE);

// Unifier を作成します。

Unifier unifier = ExpressionFactory.createUnifier(null,
unificationList);

// Executor を作成するための準備として、ステートフルな
// Evaluator と Validator を作成します。

// 環境 Map には null が渡されます。
```

```
Evaluator evaluator = ExpressionFactory.createEvaluator(null);

// 環境 Map には null が渡されます。

// Validator を作成します。

Validator validator = ExpressionFactory.createValidator(null);

// Executor を作成します。

Executor executor = ExpressionFactory.createExecutor(null);

// Unifier と、null でない Validator を渡して、
// 上の式を実行します。

Object result = executor.execute(expression, unifier, validator,
evaluator);

// 結果は Boolean.FALSE になります。
```

注意： execute メソッドを呼び出す前に、式を統合することもできます。それには、Unifier の unify メソッドを呼び出します。一旦式が統合された後は、Executor の execute メソッドに Unifier を渡す必要はありません。validate メソッドを呼び出すことによって、直接検証サービスを使用することもできます。与えられた式が無効な場合は、validate メソッドは、InvalidExpressionException を送出します。

Expression パッケージの設定

expression.properties ファイルには、Expression パッケージの設定が含まれています。このファイルは、注意深く変更しなければなりません。

このファイルは、com.bea.p13n.expression パッケージの下の p13n_util.jar アーカイブにまとめられています。

```
##
# 式 Comparator の null の扱い
#
# 次のプロパティを true に設定すると、
# 式 Comparator は、null でない値と null を比較したときに
# 実行された比較の種類に関わらず
# false を返します。
#
# デフォルトは true に設定されています。
##
```

```
expression.comparator.nullcheck=true

##

# 式 Comparator の等価比較のイプシロン
#
# 次のプロパティは、数値の等価比較の際の
# イプシロン値を決定します。
#

# デフォルトは 0 に設定されています。
##

expression.comparator.epsilon=0.00001

##
# 式 Introspector のメソッド配列のキャッシング
#
# 次のプロパティを true に設定すると、
# 式 Introspector は、Java クラスで実装された
# メソッドの配列をキャッシュします。
#

# デフォルトは true に設定されています。
##

expression.introspector.method.array.cache=true

##
# 式 Introspector のメソッドのキャッシング
#
# 次のプロパティを true に設定すると、
# 式 Introspector は、シグネチャによってメソッドをキャッシュします。
#
# デフォルトは true に設定されています。
##

expression.introspector.method.cache=true

##
# 式パーザのノード サポート クラス
#
# このプロパティは、基本 AST NodeSupport クラスを拡張したクラスの
# カンマ区切りのリストをサポートします。このようなクラスは、与えられた式インス
タンスの
# 中間的な AST 表現を作成するために必要な
# 式スキーマ用ネームスペースの作成をサポートします。
#
# NodeSupport のすべてのサブクラスは、
# 要求された CoreNodeSupport インスタンスと共存しなければなりません。
##

parser.node.support.list=\

com.bea.p13n.expression.internal.parser.expression.ExpressionNode
Support
```

```
##
# 式パーザの Transform Visitor クラス
#
# このプロパティは、ExpressionTranformVisitor または
# 中間的な AST-to-Expression 変換のために使われる
# サブクラスを指定します。
#
##

parser.transform=\

com.bea.p13n.expression.internal.parser.expression.ExpressionTran
sformVisitor
```

付録 A イベント解説

この付録には、WebLogic Portal が提供している標準イベントについての情報がまとめられています。具体的には、各イベントの解説、イベントの発生理由、イベントの生成元のクラス、使用例、および各イベント オブジェクト内のデータ型について説明しています。

WebLogic Portal のイベントは、次のカテゴリに分類されます。

- セッション イベント
- ユーザ登録イベント
- 商品イベント
- コンテンツ イベント
- カート イベント
- 購入イベント
- ルール イベント
- キャンペーン イベント

セッション イベント

セッション イベントは、開始時と終了時、実行中の場合は、訪問者のセッションのログイン時に発生します。

SessionBeginEvent

解説	訪問者が Web サイトやポータル サイトと対話を開始するときに発生する。

クラス	`com.bea.p13n.tracking.events.SessionBeginEvent`
生成元	15-9 ページの「サーブレット ライフサイクル イベントとサーブレット フィルタ イベント」を参照。

要素	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`

SessionEndEvent

解説	訪問者がサイトを去るとき、または訪問者のセッションがタイムアウトしたときに発生する。
クラス	`com.bea.p13n.tracking.events.SessionEndEvent`
生成元	15-9 ページの「サーブレット ライフサイクル イベントとサーブレット フィルタ イベント」を参照。

要素	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`

SessionLoginEvent

解説	訪問者が Web サイトやポータル サイトにログオンするときに発生する。
クラス	`com.bea.p13n.tracking.events.SessionLoginEvent`

生成元	TrackingEventHelper.dispatchSessionLoginEvent()、P13NAuthFilter、入力プロセッサのいずれか、またはすべて。15-10 ページの「ログイン イベントと作成イベントを生成する」を参照。
要素	application event-date event-type session-id user-id

ユーザ登録イベント

登録イベントは、1 つだけ存在します。次の表は、このイベントの説明です。

UserRegistrationEvent

解説	訪問者が Web サイトやポータル サイトに登録するときに発生する。
クラス	com.bea.p13n.tracking.events.UserRegistrationEvent
生成元	TrackingEventHelper.dispatchUserRegistrationEvent()、入力プロセッサのいずれか、または両方。
使用例	<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\wlcsApp\wlcs\WEB-INF\src の examples.wlcs.sampleapp.customer.webflow.LoginCustomerIP
要素	application event-date event-type session-id user-id

商品イベント

これらのイベントは、訪問者が商品を提示されるか、提示された商品をクリック（選択）したときに発生します。

ClickProductEvent

解説	訪問者が商品へのリンクをクリックしたときに発生する。
クラス	`com.bea.commerce.ebusiness.tracking.events.ClickProductEvent`
生成元	JSP タグ。15-9 ページの「サーブレット ライフサイクル イベントとサーブレット フィルタ イベント」を参照。

要素	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`sku`
	`category-id`
	`application-name`（ポータル アプリケーションではなく、店舗の名前）

DisplayProductEvent

解説	商品が訪問者に提示されたときに発生する。
クラス	`com.bea.commerce.ebusiness.tracking.events.DisplayProductEvent`

生成元	JSP タグ

要素	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`sku`
	`category-id`
	`application-name`（ポータル アプリケーションではなく、店舗の名前）

コンテンツ イベント

これらのイベントは、訪問者が広告などのコンテンツを提示されるか、提示されたコンテンツをクリックしたときに発生します。

ClickContentEvent

解説	顧客が、Web サイトのコンテンツ（リンク、バナーなど）をクリックしたときに発生する。
クラス	`com.bea.p13n.tracking.events.ClickContentEvent`
生成元	JSP タグ。15-9 ページの「サーブレット ライフサイクル イベントとサーブレット フィルタ イベント」を参照。

要素	application event-date event-type session-id user-id document-type document-id

DisplayContentEvent

解説	訪問者にコンテンツが提示されたときに発生する。コンテンツ管理システムからのすべてのコンテンツが対象。
クラス	com.bea.p13n.tracking.events
生成元	JSP タグ
要素	application event-date event-type session-id user-id document-type document-id

カート イベント

これらのイベントは、訪問者のショッピング カートに 1 つ以上の商品が追加されたり、削除されたことを表します。

AddToCartEvent

解説	訪問者のショッピング カートに 1 つの商品が追加されたときに発生する。
クラス	com.bea.commerce.ebusiness.tracking.events.AddToCartEvent
生成元	Pipeline コンポーネント。 <BEA_HOME>\weblogic700\portal\applications\wlcsApp-project\application-sync\pipelines に存在する。
使用例	<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\beaApps\wlcsApp\src の examples.wlcs.sampleapp.tracking.pipeline.AddToCartTrackerPC
要素	application event-date event-type session-id user-id sku quantity unit-list-price currency application-name（ポータル アプリケーションではなく、店舗の名前）

RemoveFromCartEvent

解説	訪問者のショッピング カートから 1 つの商品が削除されたときに発生する。
クラス	com.bea.commerce.ebusiness.tracking.events.RemoveFromCartEvent

生成元	Pipeline コンポーネント。 `<BEA_HOME>\weblogic700\portal\applications\wlcsApp-project\application-sync\pipelines` に存在する。
使用例	`<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\beaApps\wlcsApp\src` の `examples.wlcs.sampleapp.tracking.pipeline.RemoveFromCartTrackerPC`

要素	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`sku`
	`quantity`
	`unit-price`
	`currency`
	`application-name`（ポータル アプリケーションではなく、店舗の名前）

PurchaseCartEvent

解説	1 つの注文全体に対して 1 回だけ発生する。それに対して、`BuyEvent` は、明細項目ごとに発生します。このイベントは、キャンペーンのときに役立ちます。シナリオ アクションを記述して、訪問者の特徴的な購入行動（100 ドル以上の注文や、特定の商品の購入など）を知りたいときに、このイベントを利用できます。
クラス	`com.bea.commerce.ebusiness.tracking.events.PurchaseCartEvent`
生成元	Pipeline コンポーネント。 `<BEA_HOME>\weblogic700\portal\applications\wlcsApp-project\application-sync\pipelines` に存在する。

使用例	<BEA_HOME>\weblogic700\portal\samples\portal\wlcsDomain\beaApps\wlcsApp\src の examples.wlcs.sampleapp.tracking.pipeline.PurchaseTrackerPC
要素	application session-id user-id event-date event-type total-price order-id currency application-name （ポータル アプリケーションではなく、店舗の名前）

購入イベント

購入イベントは、1 つだけ存在します。次の表は、このイベントの説明です。

BuyEvent

解説	訪問者が購入を完了したときに発生する。BuyEvent は明細項目ごとに発生します。1 回の購入には、複数の明細項目が含まれます。1 個の明細項目には、複数の商品が含まれる場合があります。たとえば、ある明細項目の数量が 4 個でも、1 つの BuyEvent しか発生しません。
クラス	com.bea.commerce.ebusiness.tracking.events.BuyEvent
生成元	Pipeline コンポーネント

使用例	<BEA_HOME>\weblogic700\portal\applications\wlcsApp\src の examples.wlcs.sampleapp.tracking.pipeline.PurchaseTrackerPC
要素	application event-date event-type session-id user-id sku quantity unit-price currency application-name （ポータル アプリケーションではなく、店舗の名前） order-line-id

ルール イベント

ルール イベントは、1 つだけ存在します。次の表は、このイベントの説明です。

RuleEvent

解説	訪問者が Web サイトを移動するときにトリガしたルールを表す。
クラス	com.bea.p13n.tracking.events.RuleEvent
生成元	アドバイズレットから内部的に発生する。

要素	application event-date event-type session-id user-id ruleset-name rule-name

キャンペーン イベント

これらのイベントは、訪問者がキャンペーンに参加したときに発生します。

CampaignUserActivityEvent

解説	訪問者がキャンペーンに参加したときに発生する。具体的には、1つ以上のシナリオ アクションが真で、キャンペーン サービスがアクティブのときに、このイベントが発生します。このイベントが、特定のシナリオに対してだけ発生するように制限することもできます。このイベントは、分析ソフトウェアでの利用を目的としています。
クラス	com.bea.campaign.tracking.events.CampaignUserActivityEvent
生成元	キャンペーン サービスから内部的に発生する。
要素	application event-date event-type session-id user-id campaign-id scenario-id

DisplayCampaignEvent

解説	広告などのキャンペーン コンテンツが訪問者に提示されたときに発生する。具体的には、キャンペーンによって広告バケットに置かれた広告が、キャンペーン プレースホルダに提示されたときに、このイベントが発生します。別のキャンペーンを起動するために、このイベントを利用できます。分析ソフトウェアは、このイベントを利用して、キャンペーンの結果、訪問者が広告を見たかどうかを判断します。
クラス	`com.bea.campaign.tracking.events.CampaignUserActivityEvent`
生成元	キャンペーン サービスから内部的に発生する。

要素	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`campaign-id`
	`scenario-id`
	`application-name`（ポータル アプリケーションではなく、店舗の名前）
	`placeholder-id`

ClickCampaignEvent

解説	広告などのキャンペーン アイテムが訪問者によってクリックされたときに発生する。具体的には、訪問者が、キャンペーンによって広告バケットに置かれたキャンペーン広告をクリックしたときに、このイベントが発生します。別のキャンペーンを起動するために、このイベントを利用できます。分析ソフトウェアは、このイベントを利用して、キャンペーンの結果、訪問者が広告をクリックしたかどうかを判断します。
クラス	`com.bea.campaign.tracking.events.ClickCampaignEvent`
生成元	キャンペーン サービスから内部的に発生する。15-9 ページの「サーブレット ライフサイクル イベントとサーブレット フィルタ イベント」を参照。

要素	`application`
	`event-date`
	`event-type`
	`session-id`
	`user-id`
	`document-type`
	`document-id`
	`campaign-id`
	`scenario-id`
	`application-name`（ポータル アプリケーションではなく、店舗の名前）
	`placeholder-id`

索引

C

D

E

F

N

O

P

Q

R

S

T

U

V

W

X

ウ

エ

オ

カ

キ

ク

ケ

コ

サ

シ

ス

セ

ソ

ト

ナ

ニ

ネ

ノ

ハ

ヒ

フ

ヘ

ホ

マ

メ

ユ

ヨ

ラ

リ

ル

レ

ロ

ワ