BEA WebLogic Platform™

WebLogic Platform サンプル アプリケーション ツアー

リリース 7.0 (Service Pack 1)
マニュアルの日付 : 2002 年 9 月

著作権

限定的権利条項

商標または登録商標

BEA WebLogic Platform サンプル アプリケーションツアー

パート番号	マニュアルの日付	ソフトウェアのバージョン
なし	2002 年 9 月	7.0 (Service Pack 1)

目次

4. Web サービス ツアー

このマニュアルの内容

このマニュアルでは、BEA WebLogic Platform サンプル アプリケーション ツアーについて説明します。

マニュアルの内容は以下のとおりです。

- 第 1 章の「はじめに」では、サンプルの機能の概要、および開始方法について説明します。
- 第 2 章の「企業消費者間 (B2C) ポータル ツアー」では、架空の企業 Avitek Digital Imaging のポータル Web サイトについて説明します。このサイトでは、WebLogic Portal のポータル フレームワーク、WebLogic Workshop に組み込まれた 2 つの Web サービスの使用、さらに WebLogic Integration に備わっている在庫確認と注文管理など、製品のさまざまな機能を組み合わせて表示します。
- 第 3 章の「Avitek の購入担当者とサプライヤとの接続」では、Avitek 購入担当者がサプライヤから見積もりを受け取り、注文書を発行し、受付通知を確認するための架空のイントラネット サイトについて説明します。ビジネス プロセスは、WebLogic Integration によって管理されます。
- 第 4 章の「Web サービス ツアー」では、2 つの Web サービスを WebLogic Workshop で構築する方法、およびポートレットで使用可能な Web サービス インタフェース コードを Portlet Wizard で生成する手順について説明します。

対象読者

このマニュアルは、WebLogic Platform ソフトウェアを評価または使用する製品評価担当者、アプリケーション開発者、システム管理者を対象に作成されています。このため、Web 技術、および Windows システムと UNIX システムの一般的な概念について読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のマニュアルは、BEA の Web サイトで入手できます。BEA ホーム ページから [製品のドキュメント] をクリックするか、または、「e-docs」の製品ドキュメント ページ (`http://edocs.beasys.co.jp/e-docs/index.html`) に直接アクセスしてください。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用して、Web ブラウザからこのマニュアルを一度に 1 章ずつ印刷できます。

このマニュアルの PDF 版は、WebLogic Platform の Web サイトで入手できます。また、ドキュメント CD にも収録されています。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体 (または一部) を書籍の形式で印刷できます。PDF を表示するには、WebLogic Platform ドキュメントのホーム ページを開き、[ドキュメントのダウンロード] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader をお持ちでない場合、Adobe Web サイト (`http://www.adobe.co.jp`) から無償で入手できます。

関連情報

BEA の Web サイトでは、WebLogic Platform の全マニュアルを提供しています。WebLogic Platform サンプル アプリケーションを使用する際に参考となる他のマニュアルは、以下のとおりです。

- *『BEA Weblogic Platform の紹介』*
- *『BEA WebLogic Server の紹介』*
- *『BEA WebLogic Workshop オンライン ヘルプ』*

- *『BEA WebLogic Integration 入門』*
- *『BEA WebLogic Portal 管理者ガイド』*
- *『BEA WebLogic Portal 開発者ガイド』*

サポート情報

WebLogic Platform のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@beasys.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用の WebLogic Platform マニュアルのリリースをお書き添えください。

本バージョンの BEA WebLogic Platform について不明な点がある場合、または BEA WebLogic Platform のインストールおよび動作に問題がある場合は、BEA WebSupport (http://www.bea.com) を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際にはあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファックス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	説明
太字テキスト	用語集で定義されている言葉を示す。
〔Ctrl〕+ 〔Tab〕	2 つ以上のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、Java のクラス、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

太字の等幅テキスト	コード内の重要な語を示す。 *例*: `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示す。 *例*: `String *CustomerName*;`
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 *例*: LPT1 BEA_HOME OR

表記法	説明
{ }	構文の中で複数の選択肢を示す。{ } 自体は、入力しない。

[ ]	構文の中で任意指定の項目を示す。[ ] 自体は、入力しない。
	例:
	`java utils.MulticastTest -n name -a address`
	`[-p portnumber] [-t timeout] [-s send]`

|	構文の中で相互に排他的な選択肢を区切る。| 自体は、入力しない。
	例:
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`password {application} {source}`

...	コマンド ラインで以下のいずれかを示す。
	■ 引数を複数回繰り返すことができる。
	■ 任意指定の引数が省略されている。
	■ パラメータや値などの情報を追加入力できる。
	... 自体は、入力しない。
	例:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	コード サンプルまたは構文で項目が省略されていることを示す。 垂直の省略記号自体は、入力しない。

1　はじめに

BEA WebLogic Platform では、標準に基づく*構築 - 統合*手法を採用し、アプリケーションの開発およびデプロイ、既存システムへの迅速な統合、ビジネス プロセスの自動化、取引相手への接続を可能にしています。

また、構築 - 統合手法をサポートし、WebLogic Platform の機能を相互に使用できるように、特別な統合エントリ ポイントを設け、プロセス レベルの通信や、フロントエンド Web アプリケーションと異機種間バックエンド システムとのデータ フローを容易にしています。*統合エントリ ポイント*は、相互使用が行われる場所を定義します。

WebLogic Platform サンプル アプリケーションでは、これらの多くの重要な統合ポイントが重点的に説明され、また、楽しく学習することもできます。

サンプルには、架空企業 Avitek Digital Imaging の企業消費者間取引 (B2C) ポータル Web サイトがあります。Avitek ポータルの訪問者は、カメラとアクセサリを購入できます。Web サイトには製品カタログがあり、2 つの Web サービスを使用したり、在庫確認、フル ショッピング カートおよび基礎となる処理、支払認可、注文管理を試すこともできます。

サンプルでは、企業間取引 (B2B) ポータル サイトについても説明されます。Avitek イントラネットを使用すると、購入担当者は、外部サプライヤから製品部品の見積もりを受け取り、その内 1 つを選択し、部品の注文書を生成し、選択したサプライヤと受取通知を交換できます。

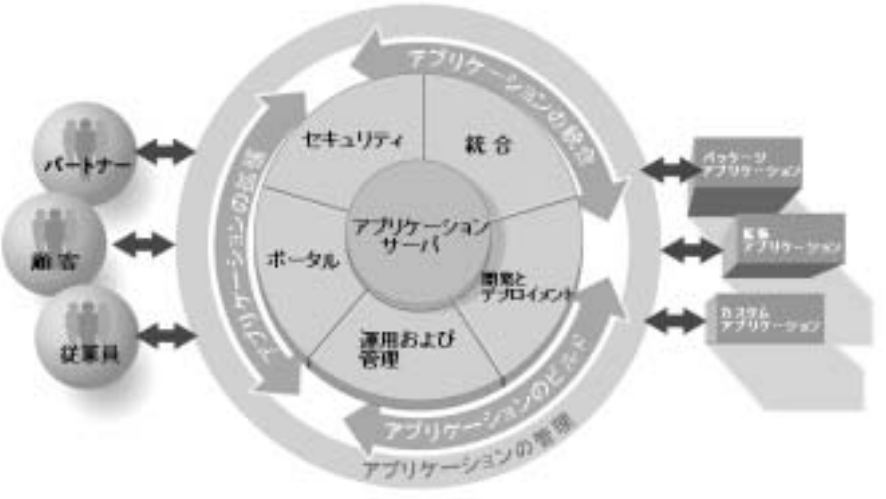
各ページおよびイベントのバックグラウンドにある技術を理解するため、B2C および B2B のサンプルには、ツアー ガイドも組み込まれています。ツアー ガイドに示される動的なマニュアル コンテンツは、実行している Web アプリケーションの最後のイベントに基づきます。ツアーのオンライン ブック バージョンは、アプリケーションのページが静的に表示されますが、内容はほとんど同じです。実行するアプリケーションとは別に説明を読みたい場合は、このオンライン ブックをご覧ください。

この章では、以下のトピックについて説明します。

- WebLogic Platform アーキテクチャ
- サンプルの開始
- サンプルの [はじめに] ページ
- 統合ポイント

WebLogic Platform アーキテクチャ

WebLogic Platform には、J2EE 準拠アプリケーション サーバ、および Platform のトップに構築される開発、ポータル、統合フレームワークなど、統合された機能があります。以下の図は、WebLogic Platform がどのように単一で、統合性の高いソリューションを提供するかを示します。



- アプリケーション サーバは、e ビジネス アプリケーションを迅速に開発、デプロイ、管理するため、BEA WebLogic Server を用いた基礎を提供します。
- 開発フレームワークは、エンタープライズクラス Web サービスのサポートなど、統合された開発環境およびランタイム環境を提供します。これには、

WebLogic Server API を使用した Web サービスのプログラミングや、BEA WebLogic Workshop を使用した Web サービスの作成が含まれます。

- ポータル フレームワークは、BEA WebLogic Portal を使用して、機能性の高い e ビジネス サイトを効率的に構築、開始、保守するためのサービスを提供します。このフレームワークを使用すると、複数のポータルを簡単に作成し、カスタマイズし、管理することができます。WebLogic Portal には、パーソナライズしたコンテンツ、Web ベースのマーケティング キャンペーン、製品カタログを提供するためのサービスや、ショッピング カート、支払サービス、税金サービスを含む注文記入システムなど、多くの関連機能があります。
- 統合フレームワークは、BEA WebLogic Integration を使用して、エンタープライズ リソース プランニング (ERP)、サプライ チェーン管理 (SCM)、人事 (HR)、顧客管理 (CRM)、カスタムおよび従来のアプリケーションなど、エンタープライズ情報システムを、標準に基づく統合技術を用いて統合します。このフレームワークによって、サプライヤと取引相手間の提携や、ビジネス プロセス ワークフローの自動化も可能になります。

詳細については、*『Weblogic Platform の紹介』*を参照してください。

サンプルの開始

WebLogic Platform のサンプルは、QuickStart アプリケーションから簡単に開始できます。QuickStart は、WebLogic Platform を初めて評価、学習、使用するユーザの手助けをします。QuickStart では、サンプルにすばやくアクセスできる他にも、特定の開発作業のためのツールや、オンライン ドキュメントへのリンクがあります。

QuickStart は、WebLogic Platform を [標準インストール] または [カスタム インストール] でインストールした後に、自動的に実行されます。サンプルには、動的な Web アプリケーションが含まれるため、WebLogic Platform の一部のみをインストールした場合は、サンプルを実行できません。インストール手順の詳細については、「WebLogic Platform のインストール」を参照してください。

以下の画面は、QuickStart のオプションを示しています。

[QuickStart] ページで、[Live Platform ツアー] リンクをクリックします。これによってサンプルの起動スクリプト `E2Estart.bat` (Windows) または `E2Estart.sh` が開始されます。スクリプト ファイルは、`BEA_HOME` の下にある以下のディレクトリにあります。

```
weblogic700/samples/platform/e2eApp/config
```

インストール後、QuickStart を以下のように開始します。

- Window システムの場合は、[スタート｜プログラム｜ BEA WebLogic Platform 7.0 ｜ QuickStart] を選択します。
- UNIX システムの場合は、次の手順を実行します。

a. 目的の UNIX システムにログインします。

b. コマンドライン シェルを開きます。

c. WebLogic Platform をインストールしたディレクトリにある /common/bin サブディレクトリに移動します。

   例 :

   ```
   cd /home/bea/weblogic700/common/bin
   ```

d. 以下のコマンドを入力します。

   ```
   sh quickstart.sh
   ```

QuickStart を使用した場合は、サーバ起動後、デフォルトのブラウザが自動的に開きます。サーバを起動し、「WebLogic Server started on...」という確認メッセージがコマンド ウィンドウに表示された後、E2Estart.bat または E2Estart.sh スクリプトを直接実行した場合は、サポート対象のブラウザを開き、次の URL を指定します。

```
http://<host>:<port>
```

たとえば、ローカル マシンでサーバを実行し、デフォルトのポート番号 7501 を使用する場合は、次のように URL を指定します。

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

サーバをリモート マシン (たとえば、blues) で実行し、デフォルトのポートを使用する場合は、次のように URL を指定します。

```
http://blues:7501
```

WebLogic Platform がサポートするブラウザについては、『サポート対象プラットフォーム』マニュアルを参照してください。

サンプルの [はじめに] ページ

サーバ起動後、前の節で説明したとおり、サンプル アプリケーションには [はじめに] ページが表示されます。このページは、大きく 5 つのセクションに分けられます。各セクションの概要は、次のとおりです。

セクション 01 紹介文

[はじめに] ページのセクション 01 は紹介文で、実行するサンプルへのリンクはありません。02 (B2C) および 03 (B2B) の Web アプリケーションで使用するツアー ガイド ポートレットのスクリーン ショットに説明が付けられて表示されます。たとえば、次のように表示されます。

B2C ポータルまたは B2B ポータルで、あるページから別のページに移動する際、ツアー ガイドにより、新しい内容や次の作業手順を理解できます。

ツアー ガイドの各ページには、以下の操作のためのリンクもあります。

- ポータル ページのポートレット コードを表示する
- 現在のページまたは直近に使用したポートレットに関する処理についての技術情報を読む
- BEA e-doc Web サイト上の関連マニュアルへのリンク一覧を取得する

[はじめに] ページのセクション 01 には、B2C ポータルおよび B2B ポータルで使用する 2 つのボタンのサンプルも表示されます。たとえば、次のように表示されます。

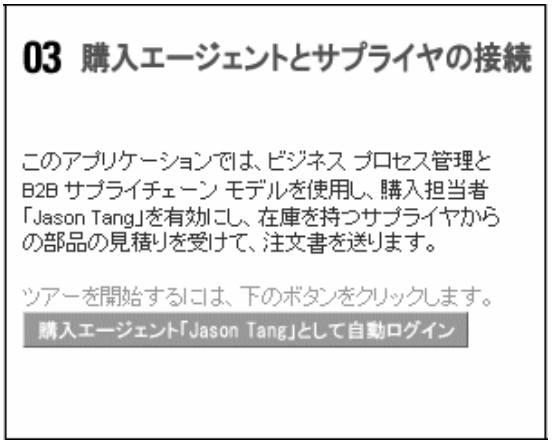
いずれかのボタンをクリックすると、B2C ポータルまたは B2B ポータルを終了し、サンプルの [はじめに] ページに戻ることができます。

セクション 02 B2C ポータルへのリンク

[はじめに] ページのセクション 02 では、B2C ポータルに「Rachel Adams」としてログインできます。Rachel は、架空の Avitek Digital Image Web サイトでの登録済みの顧客です。サンプル アプリケーションのこの部分には製品カタログがあり、2 つの Web サービスを使用したり、在庫確認、フル ショッピング カートおよび基礎となる処理、支払認可、XML 注文データの統合システムへの送信を含む注文管理を行うことができます。たとえば、次のように表示されます。

セクション 03 B2B ポータルへのリンク

[はじめに] ページのセクション 03 では、Avitek 購入担当者「Jason Tang」として、B2B ポータルにログインできます。Jason は、Avitek イントラネットを使用し、外部サプライヤから製品部品の見積もりを受け取り、その内 1 つを選択し、部品の注文書を生成し、選択したサプライヤと受取通知を交換します。



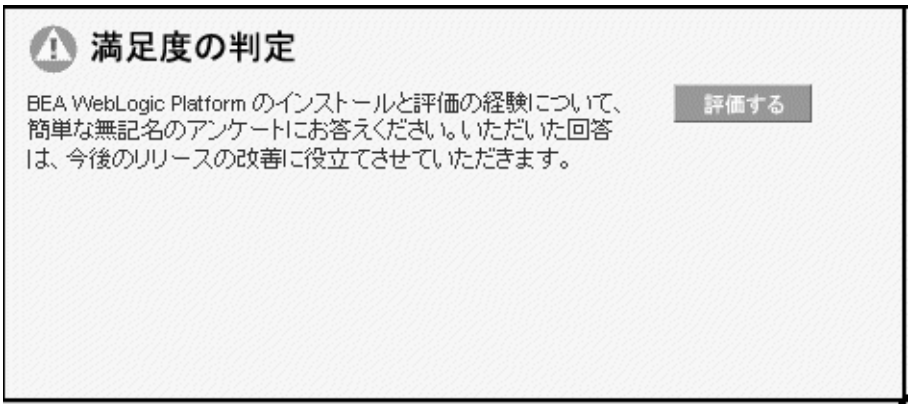
セクション 04 Web サービス技術ツアーへのリンク

[はじめに] ページのセクション 04 は、文書のみの Web サービス技術ツアーにリンクしています。たとえば、次のように表示されます。

このツアーでは、WebLogic Workshop を使用して、Web サービスの Product Evaluator および Payment をそれぞれ定義する方法について説明します。ツアーのこの部分では、Portlet Wizard での手順も説明します。Portlet Wizard を使用して、セクション 02 B2C ポータルに表示される商品評価ポートレット用の Web サービス インタフェース コードを生成します。

アンケート セクション

[はじめに] ページの最後の部分には、WebLogic Platform に対する満足度をお尋ねする簡単な匿名アンケートがあります。たとえば、次のように表示されます。



統合ポイント

サンプル アプリケーションには、WebLogic Platform を構成するコンポーネントが統合されて使用される領域も準備されています。統合ポイントは、以下のとおりです。

- Product Evaluator Web サービスおよび商品評価ポートレット
- Payment Web サービス
- WebLogic Portal で生成され、JMS キューおよび BPM を介して WebLogic Integration によって処理される注文
- WebLogic Integration AI によるリアルタイムの在庫確認

Product Evaluator Web サービスおよび商品評価ポートレット

企業消費者間取引 (B2C) サンプル ポータルには、選択した製品のレーティングを返す商品評価ポートレットがあります。カタログのカテゴリを移動していくと、[Products] タブのあるページの下の方にポートレットが表示されます。たとえば、次のように表示されます。

ポートレットには、WebLogic Workshop を使用して作成した Web サービス `productEvalWSC` を呼び出すコードがあります。

Web サービスは、アプリケーションがネットワークまたはインターネットを経由してアクセス可能な、言語独立、プラットフォーム独立、自己記述性コード モジュールです。アプリケーションは、サービスの場所をハードコード化することも、UDDI (Universal Description, Discovery, and Integration) を使用して指定することもできます。サービスは自己記述性のため、アプリケーションは使用する機能や、その呼び出し方法を決定できます。

開発では、WebLogic Workshop が提供するブラウザ ベースのテスト フォームを使用し、Web サービスから返される結果が予測どおりであるかを確認します。テスト フォームの機能およびポートレットについては、2-35 ページの「商品評価ポートレットと Web サービス」の節を参照してください。WebLogic Workshop、および WebLogic Portal に付属する Portlet Wizard での開発手順については、第 4 章の「Web サービス ツアー」も参照してください。

Payment Web サービス

支払の認可、取り込み、決済を実行する Web サービスを設計するために、WebLogic Workshop も使用しました。このポータル アプリケーションでは、ショッピング カートの [注文の送信] ボタンをクリックすると、ページのクレジット カード情報が Web サービスを介して認可されます。サンプルでは、事前定義されたクレジット カード データおよび Web サービスはローカルに保存されていますが、使用するコードは、必要な処理を行うために実際に動くものです。

Payment Web サービスは「対話形式」です。`authorize()` 呼び出しは会話を開始し、`capture()` 呼び出しは会話を継続し、`settle()` 呼び出しは会話を完了または終了します。

この Web サービスは、CajunBasedPaymentPC.java というパイプライン コンポーネントから呼び出されます。パイプライン コンポーネントは、Web サービスと相互作用するプロキシを使用します。支払の認可、取り込み、および決済の方法に問題が生じると、エラー コードが返されます。詳細については、Payment.jws ファイルを参照してください。使用するプロキシは、WebLogic Server clientgen タスクで生成されます。

Payment Web サービスおよび Portlet Wizard についての情報は、このサンプルの [はじめに] ページから表示可能な Web サービス技術ツアーをご覧ください。詳細については、第 4 章の「Web サービス ツアー」を参照してください。

WebLogic Portal で生成され、JMS キューおよび BPM を介して WebLogic Integration によって処理される注文

b2cPortal ツアーでは、ログインしたユーザが [注文の送信] ボタンをクリックした後に、WebLogic Portal で生成された注文をビジネス プロセスで使用できます。注文は、XML 表示に変換され、WebLogic Integration の Java Message Service (JMS) キューに置かれます。次に、WebLogic Integration のビジネス プロセス管理 (BPM) コンポーネントが、その注文を処理します。詳細については、第 2 章の「企業消費者間 (B2C) ポータル ツアー」を参照してください。

b2bPortal ツアーでは、Avitek 購入担当者は、外部サプライヤから部品の見積もりを受け取り、注文書を送信し、選択したサプライヤとデータの交換をします。この作業には、Query for Price and Availability (QPA) ビジネス プロセスと、注文書 (PO) ビジネス プロセスという異なる 2 つのビジネス プロセスが含まれます。WebLogic Integration は、取引相手とのビジネスに関する会話、および提携契約を管理し、バイヤとサプライヤ間のビジネス メッセージの交換を自動化します。ワークフローは、提携契約および会話で参照されます。詳細については、第 3 章の「Avitek の購入担当者とサプライヤとの接続」を参照してください。

WebLogic Integration AI によるリアルタイムの在庫確認

サンプル アプリケーションのデータベースには、在庫テーブルがあります。テーブルには、製品および部品に関する現在、最小、最大の在庫数についてのデータが保管されます。b2cPortal には、Web サイトで販売する商品を SKU で管理するカタログがあります。b2bPortal は、製品およびその部品の両方に対して SKU を使用します。

b2cPortal および b2bPortal のいずれの場合でも、在庫テーブルは、読み込み専用モードで、WebLogic Integration の Application Integration (AI) コンポーネントを介してアクセスされます。ユーザがショッピング カートに商品を追加しようとすると、その商品の在庫がチェックされ、注文が可能なことを確認します。たとえば、b2cPortal の [Products] ページにあるポートレットの [今すぐ購入] ボタンをクリックすると、この確認が実行されます。在庫確認は、ショッピング カートの `step1.jsp` ポートレットに新しい値を入力し、[再計算] ボタンをクリックしてショッピング カートにすでにある商品の数を変更するときにも実行されます。

2 企業消費者間 (B2C) ポータル ツアー

企業消費者間 (B2C) ポータル ツアーでは、Avitek Digital Imaging ポータルに実装された主な機能について説明します。このサイトでは、WebLogic Portal のポータル フレームワーク、WebLogic Workshop に組み込まれた 2 つの Web サービスの使用、さらに WebLogic Integration に備わっている在庫確認と注文管理など、製品のさまざまな機能を組み合わせて表示します。これらのコンポーネントはすべて、WebLogic Server の Web アプリケーション サーバ環境で動作します。

注意： このオンライン ブックに掲載されている情報は、アプリケーションの一部として実行される状況依存ツアー ガイド ポートレットでも入手できます。

この企業消費者間 (B2C) ポータル ツアーには、以下の節があります。

- 初期処理の概要
- [My Avitek] ページ
- [Products] ページ
- Category ポートレット
- Product Item ポートレット
- 商品評価ポートレットと Web サービス
- [今すぐ購入] ボタンと、WebLogic Integration AI を介した在庫チェック
- Search Results ポートレット
- [保留] ボタン

- My ショッピング カート ポートレット、Step1.jsp
  - [保存されているアイテム] リストの [カートに追加] ボタン
  - [保存されているアイテム] リストの [削除] ボタン
  - [現在のアイテム] リストの [削除] ボタン
  - [現在のアイテム] リストの [保留] ボタン
- チェックアウト ポートレット、Step2.jsp
- 注文の送信ポートレット、Step3.jsp
- 注文の確認ポートレット、Step4.jsp

初期処理の概要

WebLogic Platform のサンプル アプリケーションを起動したとき、サーバが起動し、[はじめに] すなわち「スプラッシュ」ページが最初に表示されたのは、どういう仕組みによるのでしょうか。まず、サンプル アプリケーションを呼び出すにはいくつかの方法があり、WebLogic Platform Quick Start Application または Windows の [スタート] メニューから起動する方法や、`startE2E` スクリプトの実行という直接的な方法がありました。

どのオプションを使用したかに関わらず、`startE2E.bat` (Windows) または `startE2E.sh` (UNIX) スクリプトが呼び出されました。これで、アプリケーションの WebLogic Server インスタンスが起動されましたが、このアプリケーションは、`e2eDomain` というドメインで実行されるものです。(「`e2e`」とは、エンドツー エンド (end-to-end) を略したもので、主要機能のすべての範囲を示すサンプルであることを意味しています)。「ドメイン」という言葉は、コンピュータ業界では、さまざまな意味を持っています。BEA 製品で**ドメイン**というとき、それは、サーバ、サービス、インタフェース、マシン、関連するリソース マネージャなど、すべて 1 つのコンフィグレーション ファイルで定義されるものの集合を表します。

`startE2E` スクリプトを実行すると、エンタープライズ アプリケーションの `config.xml` ファイルからコンフィグレーション情報が読み込まれます。デフォルトでは、このコンフィグレーション ファイルは、BEA 製品がインストールされている以下の `BEA_HOME` ディレクトリにあります。

```
weblogic700/samples/platform/e2eDomain
```

config.xml ファイルは以下のように定義され、そのドメインのデフォルト Web アプリケーションとして splashPage を設定しています。

```
<WebServer

      DefaultWebApp="splashPage"
      LogFileName="./logs/access.log"
      LoggingEnabled="true"
      Name="e2eServer"
   />
```

e2eServer が稼動した状態で、http://localhost:7501 などの URL をブラウザで指定すると、以下のディレクトリにある splashPage Web アプリケーションが起動します (Web アプリケーションは「Webapp」と略称されることもあります)。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\splashPage
```

splashPage Web アプリケーションの web.xml コンフィグレーション ファイルでは、<welcome-file-list> 定義の中で index.jsp を指定します。この web.xml は、以下のディレクトリにあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\splashPage\
WEB-INF
```

スプラッシュ ページで [「Rachel Adams」として自動ログイン] ボタンのグラフィックをクリックして、この [My Avitek] ページを表示しました。このアクションの結果、スプラッシュ ページから、ポータル アプリケーションの URL と定義済みのログイン資格情報が渡されることになりました。

スプラッシュ ページは、b2cPortal または b2bPortal に自動的にログインできるようにする要求を設定します。これは、このサンプル アプリケーションを簡素化するために行われます。JSP ページにユーザ名とパスワードを組み込むようなことは、普通は行いません。ログイン資格情報の定義にスクリプトレットを使用しましたが、実際の Web アプリケーションにこの手法を用いることはお勧めしません。

注意： WebLogic Portal は、その他にも、ログイン認証コードと、Web アプリケーションによるデモグラフィック情報収集のテクニックを示すサンプルを提供します。詳細については、WebLogic Portal のドキュメントを参照してください。

b2cPortal アプリケーションについて、ブラウザに URL が渡されたとき、[My Avitek] ページが最初に表示されたのはなぜでしょうか。このデフォルト ページは、WebLogic Portal Administration Tools で定義しました。この設定の詳細と [My Avitek] ページのその他の特性については、次の節を参照してください。

[My Avitek] ページ

B2C ポータル ツアーは [My Avitek] ページから始まります。この [My Avitek] ページには、ログイン ユーザである Rachel Adams 向けにパーソナライズされたコンテンツが表示されます。WebLogic Portal ツールは、一連の情報をポートレットにまとめるために使われました。このページ上のパーソナライズされたコンテンツには、Rachel の現在のショッピング カートと注文履歴があり、[ログアウト] ボタンの横に Rachel の名前が一覧表示されています。

これが初めてのツアーで、共有サーバ インスタンスに Rachel Adams としてログインしているユーザが他にいない場合、ショッピング カートと注文履歴は空白です。次の画面は、すでに 2 つの注文を完了したユーザの [My Avitek] ページです。

[My Avitek] ページの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

[My Avitek] ポータル ページには、ログイン ユーザがサイトのパーソナライズ表示をできるようにするためのポートレットが数多く用意されています。[My Avitek] ページは、b2cPortal で 3 つのタブから構成されるページの 1 つです。WebLogic Platform をインストールした `BEA_HOME` ディレクトリの以下の場所に、`e2eApp` を構成するファイルがあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\...
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\...
```

[My Avitek] ページには、以下のポートレットを追加することができます。

- **mybanner** ポートレット。一番上のバナーと 3 つのタブ (My Avitek、Products、Shopping Cart) のページを表示します。
- **login** ポートレット。認証済みのユーザがアプリケーションにログインする際に使用します。[ログアウト] ボタンおよび現在ログイン中のユーザ名を表示します。
- **search** ポートレット。キーワードベースの検索入力ボックスを表示します。
- **myavitek** ポートレット。[My Avitek] ページに、[写真貼り付けのヒント] をグラフィックとテキストで表示します (このサンプルに表示されるコンテンツは静的ですが、WebLogic Portal の機能を使えば、さまざまなユーザ向けにパーソナライズできます。アプリケーション コンテンツのパーソナライズの詳細については、*『開発者ガイド』* を参照してください)。
- **summarycart** ポートレット。ログイン ユーザのショッピング カートの簡単な一覧を表示します。
- **orderhistory** ポートレット。ログイン ユーザの注文履歴の簡単な一覧を表示します。
- **tourguide** ポートレット。アプリケーションの状況依存ドキュメントを表示します。このドキュメントには、実行中のアプリケーションの左側に小さく表示されるものと、現在示しているように [技術的詳細を参照]、[コードを表示]、[e-docs を参照] というポインタを含む最大化されたものの 2 つの形式があります。

[My Avitek] ページを構成するすべてのポートレットは、以下のファイルで定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\webapps\b2cPortal\b2cPortal.portal
```

例 :

```
<page-name>My Avitek</page-name>
...
  <portlet-pool>
      <portlet-name>login</portlet-name>
      <portlet-name>search</portlet-name>
      <portlet-name>tourguide</portlet-name>
      <portlet-name>subnav</portlet-name>
      <portlet-name>summarycart</portlet-name>
      <portlet-name>myavitek</portlet-name>
      <portlet-name>orderhistory</portlet-name>
      <portlet-name>mybanner</portlet-name>
```

```
        <portlet-name>anonUser</portlet-name>
    </portlet-pool>
```

これらのポートレットは、開発サイクル時に E-Business Control Center を使用して [My Avitek] ページに追加されました。EBCC は、Java クライアントベースのツールの集まりです。このツールには、ルールの定義、Webflow の編集、ポータルの作成と管理などの複雑な作業を簡単に行えるグラフィカル インタフェースが用意されています。E-Business Control Center のユーザは、ポイントアンドクリック方式のインタフェースで作業を行い、サーバと同期をとる XML ファイルを生成します。

EBCC の他にも、実行時のポータルの管理に、ブラウザベースの WebLogic Portal Administration Tools を使用しました。その処理の詳細については、*『管理者ガイド』* を参照してください。

開発者は、このサンプル アプリケーションを実行する際に、ブラウザに表示される前のポートレット JSP コードについて理解することが重要です。そのため、この節で使用されるコードの断片や [コードを表示] リンクでは、特定のポートレットのブラウザで表示する前の JSP ファイルについて説明します。

b2cPortal のポートレット JSP ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\...
```

この [My Avitek] ページでは、[コードを表示] リンクをクリックすると、`\orderhistory\content.jsp` ポートレット ファイルのソースが開きます。ここには、ログイン ユーザのパーソナライズされた注文履歴が表示されます。このポートレットのファイルは、上記のパスの下の orderhistory サブディレクトリにあります。

Order History ポートレットが `...\e2eApp\b2cPortal\portlets\orderhistory` ディレクトリの content.jsp を使用していることはどうしてわかるのでしょうか。これは WebLogic Platform の E-Business Control Center で指定されました。この情報は次のファイルにも定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\portlets\orderhistory.portlet
```

この XML ファイルでは以下が定義されています。

```
<portlet-name>orderhistory</portlet-name>
...
<content-url>/portlets/orderhistory/content.jsp</content-url>
```

b2cPortal Web アプリケーションのルート ディレクトリからコンテンツ URL への相対パスに注意してください。デフォルトでは、インストール先のディレクトリとして以下の場所が指定されます。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
```

デフォルト Webflow 内の位置

[My Avitek] ページのデフォルト Webflow について検討する前に、用語をいくつか定義しておきましょう。

Webflow とは、プレゼンテーション JSP で次ページの URL をハードコーディングせずに、Web サイトのフローを制御できるようにする開発者向けの BEA のテクノロジです。WebLogic Portal では、Webflow を使用して、アプリケーションがブラウザに表示する必要があるページ、および実行する必要があるビジネス ロジックを決定します。1 つまたは複数の Webflow ネームスペース ファイルを使用すると、特定の Web アプリケーションの Webflow をコンフィグレーションできます。

Webflow ネームスペース ファイルは、XML ファイルです。このファイルで、Web アプリケーションの Webflow のコンフィグレーション、サイトの Web ページが表示される順序の制御、およびこれらのページに関連付けられているビジネス ロジックの実行を開始します。Webflow ネームスペース ファイルは、BEA E-Business Control Center で入手できる Webflow エディタと Pipeline エディタを使って編集します。これらのエディタは、各 Web アプリケーションの Webflow を視覚的に作成、変更、検証するのに役立つグラフィカル ツールです。

Pipeline と入力プロセッサは、2 つの異なるタイプのプロセッサ ノードで、Webflow 実装のパッケージに同梱されています。

Pipeline とは、一連のサービスを 1 つの名前付きサービスにバインドできるようにする、開発者向けの BEA のテクノロジです。Pipeline を作成すると、ビジネス データの処理を管理できます。通常、Pipeline は、Webflow の結果として実行されるビジネス ロジックのフローを制御します。Pipeline は、トランザクション対応にすることもトランザクション非対応にすることもできます。たとえば、訪問者が Web サイトの別のページに移動してもその訪問者の情報をデータベースに維持したい場合は、Pipeline を使用します。Pipeline は、より規模の大きいエンタープライズ 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できるビジネス ロジックを含むため、エンタープライズ JavaBean (Enterprise JavaBeans: EJB) コンテナでロードされます。

Pipeline は、BEA E-Business Control Center で入手できる Webflow エディタと Pipeline エディタの Pipeline ノードとして表示されます。Webflow エディタと Pipline エディタによって、基本の Webflow ネームスペースと Pipeline ネームスペースの XML ファイルが生成されます。これは手動で編集しないでください。

b2cPortal アプリケーションでは、Webflow ネームスペース ファイルは以下のディレクトリにあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\application-sync\webapps\b2cPortal
```

入力プロセッサは、定義済みの特化した Java クラスで、Webflow メカニズムで呼び出されると、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 形式のデータを検証するため、または Web ページ内に条件ブランチを入れるために使われます。たとえば、日付が正しい形式で入力されているかどうかを検証するコードが、フォーム フィールドを表示する同一の JSP 内に埋め込まれるのではなく、入力プロセッサに含まれていることがあります。入力プロセッサには、Web アプリケーションに固有のロジックが含まれるため、Web アプリケーションのコンテナによってロードされます。

Webflow テクノロジと Pipeline テクノロジ、およびそれらのエディタの詳細については、*『開発者ガイド』*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。

ここには Webflow テクノロジの重要な定義も記載されています。それでは、[My Avitek] ページを見ていきましょう。[My Avitek] ページの 1 つ前のページで、以下のいずれかのアクションを行いました。

- サンプルの [はじめに] ページで、[「Rachel Adams」として自動ログイン] ボタンのグラフィックをクリック。このページで使われたフォームによって、定義済み値とログイン資格情報から URL 値が構成されました (サンプル アプリケーションを簡易化するためにこの方法で実装したもので、実際の作業にはお勧めできません)。ログイン認証コードの使い方、さらに Web アプリケーションのデモグラフィック情報収集のテクニックについては、WebLogic Portal のドキュメントを参照してください。
- [Products] ページまたは [Shopping Cart] ページから、[My Avitek] タブをクリック。

[My Avitek] ページのイベントの詳細については、次の節を参照してください。

[My Avitek] ページのイベント

顧客が JSP 上のリンクやボタンをクリックする度に、それが 1 つのイベントであると見なされます。イベントは、デフォルト Webflow 内の特定の応答をトリガし、顧客が処理を継続できるようにします。この応答から、別の JSP がロードされる場合もありますが、通常は、入力プロセッサや Pipeline が最初に呼び出された場合に行われます。

[Products] ページまたは [Shopping Cart] ページで、[My Avitek] タブをクリックすると、以下のような URL が表示されます (ここでは、見やすくするために何行かに改行してあります)。

```
http://<host>/<port>/b2cPortal/application?
origin=hnav_bar.jsp&event=bea.portal.framework.internal.refresh
&pageid=My+Avitek
```

更新イベントにより、最新データで更新されたページが再表示されます。[My Avitek] ページ上で、動的データ (My ショッピング カート、注文履歴、ツアー ガイド) が含まれるすべてのポートレットのデータが更新されます。

タブは、`hnav_bar.jsp` ファイルを介して提供されます。このファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\f
ramework
```

`hnav_bar.jsp` ファイルによって、以下の 2 つの JSP タグ ライブラリがインポートされます。

```
<%@ taglib uri="webflow.tld" prefix="wf" %>
<%@ taglib uri="portal.tld" prefix="ptl" %>
```

アプリケーション内の別のページから [My Abitek] タブをクリックした場合は、`hnav_bar.jsp` は対象となるタブへの以下のリンクで JSP タグを使用しました。

```
<a href="<ptl:createPortalPageChangeURL pageName='<%=
portalPageName %>'/>"><%=portalPageName%></a>
```

ポータルの `ptl:createPortalPageChangeURL` JSP タグは、ページ変更イベントの Webflow URL を生成します。

[My Avitek] ページでの動的データの表示

[My Avitek] ページで、動的ポートレットの 1 つ、`summarycart` を見てみます ([My Avitek] ページでは [My ショッピング カート] と表示されます)。

summarycart ポートレットの Webflow ネームスペース ファイルは、以下のとおりです。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\app
lication-sync\webapps\b2cPortal\summarycartportlet.wf
```

ここには、以下のコードが含まれます。

```
<presentation-origin node-name="step1" node-type="jsp">
    <node-processor-info page-name="step1.jsp"
         page-relative-path="/portlets/summarycart"/>
</presentation-origin>
```

step1.jsp ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\p
ortlets\summarycart\step1.jsp
```

step1.jsp ファイルでは、Pipeline セッションから現在のショッピング カートを取得する Webflow タグと保存されたショッピング カートを取得する Webflow タグなど、JSP タグを組み合わせて使用します。

```
<webflow:getProperty id="shoppingCart"
 property="<%=PipelineSessionConstants.SHOPPING_CART%>
 "type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"
 scope="session" namespace="portal" />

<webflow:getProperty id="savedShoppingCart"
 property="<%=PipelineSessionConstants.SAVED_SHOPPING_CART%>"
 type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart"
 scope="session" namespace="portal" />
```

デフォルトのポータル ページを指定する管理タスク

[My Avitek] の節で取り上げたプロパティのほとんどは、手動では編集せず、E-Business Control Center、および WebLogic Portal Administration Tools の [ポータル管理] 領域を使って定義しました。たとえば、[My Avitek] ページは、WebLogic Portal Administration Tools を使用することにより、b2cPortal アプリケーションのデフォルトの開始ページと指定されました。ここでは、このツールを起動する方法を説明し、デフォルトのページ設定のサンプル画面を示します。

1. 新しいブラウザのウィンドウで以下の URL をポイントすると、WebLogic Portal Administration Tools が起動します。

   ```
   http://<host>:<port>/e2eAppTools/index.jsp
   ```

   これらの手順は、e2eDomain のサーバがまだ稼動中であると想定しています。e2eAppTools Web アプリケーションとサンプル アプリケーションは同

時に実行できます。どちらの Web アプリケーションも同じ e2eApp エンタープライズ アプリケーションの一部であり、同じドメイン内で実行されます。

2. ローカル マシン名 (localhost) またはリモート ホスト名、および WebLogic Server がリスンしているポート番号を置き換えます。WebLogic Portal アプリケーションでは、デフォルトのポート番号は 7501 です。URL の e2eAppTools 部分では大文字と小文字が区別されます。

3. ユーザ名とパスワードの入力が要求されます。これらの値はサイトによって異なることがありますが、デフォルトは次のとおりです。

   ```
   Username: administrator
   Password: password
   ```

   ログイン値は、大文字と小文字が区別されます。これらの資格情報が機能しない場合は、管理者に問い合わせてください。サンプルのインストール後に、管理者アカウントのデフォルト値が変更された可能性があります。

4. WebLogic Portal Administration Tools のメイン ページで、[ポータル管理] バナーのアイコンをクリックします。

5. [ポータル管理ホーム] ページで b2cportal に対応する [Default Portal (Everyone)] をクリックします。

6. [グループ ポータル管理ホーム] ページで [ページ & ポートレット] をクリックします。

7. [ページおよびポートレット] ページで [ページの選択と順序設定] をクリックします。

   次の図はページの一部を示しています。

[My Avitek] ページが、どのようにしてデフォルトのホーム ページに指定されたかに留意してください。

WebLogic Portal Administration Tools の詳細については、*『管理者ガイド』*を参照してください。

WebLogic Portal Administration Tools を終了するには、ブラウザのウィンドウを閉じます。

次のステップ

ツアーを続けるには、バナーの [Products] タブを選択します。

[Products] ページ

[Products] タブをクリックすると、このページには中心となるポートレット `catalog.jsp` の製品カタログの最上位カテゴリがロードされます。カタログのテキストと画像は WebLogic Portal Administration Tools の [カタログ管理] 領域を使用して、すでに整理されています。ログイン ユーザの場合、[My ショッピング カート] と [注文履歴] の最新データが更新されました。

次の画面は、最初の [Products] ページのカテゴリを示しています。

[Products] ページの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

[Products] ページには [My Avitek] ページと同じポートレットの多くが組み込まれています。

- **mybanner** ポートレット。一番上のバナーと 3 つのタブ (My Avitek、Products、Shopping Cart) のページを表示します。
- **login** ポートレット。認証済みのユーザがアプリケーションにログインする際に使用します。[ログアウト] ボタンおよび現在ログイン中のユーザ名を表示します。
- **search** ポートレット。キーワードベースの検索入力ボックスを表示します。
- **summarycart** ポートレット。ログイン ユーザのショッピング カートの簡単な一覧を表示します。
- **orderhistory** ポートレット。ログイン ユーザの注文履歴の簡単な一覧を表示します。
- **tourguide** ポートレット。実行中のサンプルの状況依存ドキュメントを表示します。

さらに、この [Products] ページには `catalog.jsp` ポートレットが追加されています。このソース ファイルは、インストール済みディレクトリ `BEA_HOME` の下の、次のサブディレクトリにあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\portlets\catalog
```

`catalog.jsp` ポートレットには、発生したカタログ イベントに応じて、さまざまなインクルード ファイルの JSP コードが組み込まれます。このことは、これ以降のページのカタログで各種アイテムを選択したときに確認できます。

`catalog.jsp` ポートレットの他に、これ以降の [Products] ページに表示される追加ポートレットは、商品評価と呼ばれます。このポートレットは、カテゴリの製品アイテムを 1 つまたは複数含むカタログ ページに表示されます。商品評価ポートレットは、WebLogic Workshop で作成された Web サービスを使って、選

択された製品アイテムのコンシューマ レーティングを調べます。ポートレットのソース ファイル (evaluator.jsp、およびインクルード ファイル step1.jsp と step2.jsp) は、インストール済みディレクトリ BEA_HOME の下の、次のサブディレクトリにあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\portlets\evaluator
```

[Products] ページを構成するすべてのポートレットは、以下のファイルに定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\application-sync\webapps\b2cPortal\b2cPortal.portal
```

例：

```
    <page-name>Products</page-name
     ...
        <portlet-pool>
        <portlet-name>login</portlet-name>
        <portlet-name>search</portlet-name>
        <portlet-name>evaluator</portlet-name>
        <portlet-name>summarycart</portlet-name>
              <portlet-name>tourguide</portlet-name>
        <portlet-name>catalog</portlet-name>
        <portlet-name>subnav</portlet-name>
        <portlet-name>orderhistory</portlet-name>
   </portlet-pool>
```

デフォルト Webflow 内の位置

このアプリケーションの Webflow によって、[Products] タブを選択したときにこのカタログ ページが表示されました。

前の節で示したとおり、catalog.jsp ポートレットには、発生したカタログ イベントに応じて、さまざまなインクルード ファイルの JSP コードが組み込まれます。たとえば、以下の catalog.jsp の webflow:getProperty タグは、Pipeline セッションから最後のカタログ イベントを取得します。

```
<webflow:getProperty
    id="event"
    type="java.lang.String"
    property="<%= B2CPortalConstants.LAST_CATALOG_EVENT_ATTRIB %>"
    scope="session"
    namespace="portal" />
```

- カタログ イベントが NULL の場合は、以下の JSP が catalog.jsp ポートレットにインクルードされます。

  ```
  <jsp:include page="/portlets/catalog/catalog_index.jsp"
  flush="true" />
  ```

  Catalog_index.jsp によって、最初の [Products] ページに表示された最上位カテゴリのロード処理が実行されます。

- ユーザが特定のカテゴリの 1 つをクリックした場合、以下の JSP が catalog.jsp ポートレットにインクルードされます。

  ```
  <jsp:include page="/portlets/catalog/category.jsp" flush="true"
  />
  ```

  category.jsp 処理の詳細については、[カテゴリ] ページの [技術的詳細を参照] に記載されています。

- (最初の [Products] ページには表示されない) 特定のカテゴリから、ユーザが 1 つの製品アイテムをクリックした場合、以下の JSP が catalog.jsp ポートレットにインクルードされます。

  ```
  <jsp:include page="/portlets/catalog/item.jsp" flush="true" />
  ```

  item.jsp 処理の詳細については、2-31 ページの「Product Item ポートレット」を参照してください。

- ユーザが検索キーワードを入力して [Go] ボタンをクリックした場合、以下の JSP が catalog.jsp ポートレットにインクルードされます。

  ```
  <jsp:include page="/portlets/catalog/search_results.jsp"
  flush="true" />
  ```

  search_results.jsp 処理の詳細については、2-46 ページの「Search Results ポートレット」に記載されています。

動的データの表示

6 つの製品カテゴリをロードした最初の [Products] ページでは、カタログ イベントは NULL でした。その結果、catalog_index.jsp ファイルが catalog.jsp ポートレットにインクルードされました。必要に応じて、catalog_index.jsp のコードを調べてください。このコードは、インストール済みディレクトリ BEA_HOME の下の、次のサブディレクトリにあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\p
ortlets\catalog
```

catalog_index.jsp ファイルには、カタログを繰り返し表示し、各カテゴリのプロパティを取り出すビュー イテレータがあります。まず、次のように webflow:getProperty JSP タグを使って、Pipeline セッションからカタログ カテゴリのビュー イテレータを取得します。

```
<webflow:getProperty
    id="categories"
    type="com.beasys.commerce.ebusiness.catalog.ViewIterator"
    property="<%= PipelineSessionConstants.CATALOG_SEARCH_CATEGORIES%>"
    scope="session"
    namespace="portal" />
```

次にカテゴリの参照を繰り返し、各カテゴリを表示します。コーディングのサブセットを以下に示します。

```
<catalog:iterateViewIterator
    id="category"
    returnType="com.beasys.commerce.ebusiness.catalog.Category"
    iterator="<%=categories%>">

<catalog:getProperty
    id="categoryKey"
    returnType="com.beasys.commerce.ebusiness.catalog.CategoryKey"
    object="<%= category %>"
    propertyName="key" />

<catalog:getProperty
    id="largeImage"
    returnType="com.beasys.commerce.ebusiness.catalog.ImageInfo"
    object="<%= category %>"
    propertyName="image"
getterArgument="<%= new Integer(Category.LARGE_IMAGE_INDEX ) %>" />
...
```

catalog_index.jsp の同じイテレータを使っている状態で、各テーブルのセルにはページのプロパティがいくつか表示されます。コーディングのサブセットを以下に示します。

```
<a href="<portlet:createWebflowURL namespace="catalogportlet"
event="link.category"extraParams="<%=linkParams +
java.net.URLEncoder.encode( categoryKey.getIdentifier())
%>"/>"><img src="<webflow:createResourceURL
resource='<%=largeImage.getUrl()%>'/>" width="180" height="136"
alt="<%=shortDesc%>" border="0"></a>
```

この処理によって、catalog.jsp ポートレットの最初の [Products] ページにカテゴリが表示されます。上記の HTML を見れば、カテゴリの画像をクリックした場合、次のページへの移行のために、link.category というイベントが生成されることがわかります。しかし、この最初の [Products] ページが表示される前にどんなイベントが発生したでしょうか。詳細については、次の節を参照してください。

最初の [Products] ページでのイベント

[Products] ページまたは [Shopping Cart] ページが表示されているときに、[My Avitek] タブをクリックすると、以下のような URL が表示されます (ここでは、見やすくするために何行かに改行してあります)。

```
http://<host>/<port>/b2cPortal/application?
origin=hnav_bar.jsp&event=bea.portal.framework.internal.refresh
&pageid=Products
```

更新イベントにより、最新データで更新されたページが再表示されます。動的データを含む [Products] ページ上のすべてのポートレットのデータが更新されます。更新されたこの [Products] ページには上位 6 レベルのカテゴリを示す最初のページが含まれます。

タブは、`hnav_bar.jsp` ファイルを介して提供されます。このファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\framework
```

`hnav_bar.jsp` ファイルによって、以下の 2 つの JSP タグ ライブラリがインポートされます。

```
<%@ taglib uri="webflow.tld" prefix="wf" %>
<%@ taglib uri="portal.tld" prefix="ptl" %>
```

アプリケーション内の別のページから [Products] タブをクリックした場合は、`hnav_bar.jsp` は対象となるタブへの以下のリンク内で JSP タグを使用しました。

```
<a href="<ptl:createPortalPageChangeURL pageName='<%=
portalPageName %>'/>"><%=portalPageName%></a>
```

ポータルの `ptl:createPortalPageChangeURL` JSP タグは、ページ変更イベントの Webflow URL を生成します。

また、すでに示したとおり、この最初の [Products] ページでは、カタログ イベントは NULL です。この結果、`catalog_index.jsp` が `catalog.jsp` ポートレットにロードされます。`catalog.jsp` ポートレットのコーディングは次のとおりです。

```
<webflow:getProperty
    id="event"
    type="java.lang.String"
    property="<%= B2CPortalConstants.LAST_CATALOG_EVENT_ATTRIB %>"
    scope="session"
```

```
    namespace="portal" />
<%
    if ( event == null )
    {
%>
<jsp:include page="/portlets/catalog/catalog_index.jsp"
flush="true" />
```

カテゴリ管理タスク

このサンプル アプリケーションのカタログでは、単純な 2 つのレベルのカテゴリ階層が使われています。最初のレベルはデフォルトのルート カテゴリで、2 番目のレベルは最初の [Products] ページに表示された 6 つのカテゴリで構成されています。ただし、WebLogic Portal カタログ機能では、WebLogic Portal のドキュメントで説明しているとおり、複数レベルの階層カテゴリがサポートされています。

このサンプル アプリケーションの 6 つのカテゴリは、ブラウザベースの WebLogic Portal Administration Tools の [カタログ管理] 領域で、前に定義したものです。必要に応じて、この節の指示に従って既存の定義を確認できます。

以下の手順は、`e2eDomain` のサーバがまだ稼動中であると想定しています。`e2eAppTools` Web アプリケーションとサンプル アプリケーションは同時に実行できます。どちらの Web アプリケーションも同じ `e2eApp` エンタープライズ アプリケーションの一部であり、同じドメイン内で実行されます。

1. 新しいブラウザのウィンドウで以下の URL 形式を使うと、WebLogic Portal Administration Tools が起動します。

   ```
   http://<host>:<port>/e2eAppTools/index.jsp
   ```

   URL 内のローカル マシン名 (`localhost`) またはリモート ホスト名、および WebLogic Server がリスンしているポート番号を置き換えます。WebLogic Portal アプリケーションでは、デフォルトのポート番号は 7501 です。URL の `e2eAppTools` 部分では大文字と小文字が区別されます。

   例 :
   ```
   http://localhost:7501/e2eAppTools/index.jsp
   ```

2. ユーザ名とパスワードの入力が要求されます。これらの値はサイトによって異なることがありますが、デフォルトは次のとおりです。

   ```
   Username: administrator
   Password: password
   ```

ログイン値は、大文字と小文字が区別されます。これらの資格情報が機能しない場合は、管理者に問い合わせてください。サンプルのインストール後に、管理者アカウントのデフォルト値が変更された可能性があります。

3. WebLogic Portal Administration Tools のメイン ページで、[カタログ管理] バナーのアイコンをクリックします。

4. [カタログ管理] ページで、下線付きの [カテゴリ] をクリックします。

   以下の画面が表示されます。

下線付きのカテゴリを 1 つクリックします。たとえば、`AviPix (0avipix)` をクリックします (警告 : 赤い X アイコンはクリックしないでください。カタログからそのカテゴリが削除されてしまいます。サンプル カタログ データの管理ページを使っていることを忘れないでください)。

5. 表示された画面を下へスクロールして、そのカテゴリを表す画像の位置を確認します。

カタログ管理タスクの詳細については、WebLogic Portal のドキュメントを参照してください。

WebLogic Portal Administration Tools を終了するには、ブラウザのウィンドウを閉じます。

次のステップ

ツアーを続けるには、[AviPix Consumer Digital Cameras] カテゴリをクリックします。

Category ポートレット

この特定のカテゴリの製品アイテムはカタログからロードされ、このページに表示されます。たとえば、次のように表示されます。

[カテゴリ] ページの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

この特定のカテゴリの製品アイテムはカタログからロードされ、このページに表示されます。最上位のバナーには現在のカテゴリを強調表示したグラフィックがあります。ページの下部にある商品評価ポートレットには、ログイン ユーザ Rachel Adams がこのカテゴリの製品アイテムのレーティングを入手できるメニューがあります。

WebLogic イベントの処理によって、発生する link.category イベントが決定され、その結果、category.jsp ファイルが中心となる catalog.jsp ポートレットにロードされました。

現在のサンプル アプリケーションのページに結果として表示されたサンプルの URL は、以下のとおりです (ここでは、見やすくするために何行かに改行してあります)。

```
http://localhost:7501/b2cPortal/application
?origin=catalog_index.jsp
&event=bea.portal.framework.internal.portlet.event
&pageid=Products&portletid=catalog
&wfevent=link.category&wlcs_catalog_category_id=0avipix
```

サンプルの URL では、最初の状態は 2 番目、3 番目、4 番目の行に示されています。Webflow イベントは最後の行に示されています。link.category イベントが発生しています。このイベントの結果が、サンプル アプリケーションで表示されているページです。直前のページで選択されたカテゴリ 0avipix (この例の場合) の製品アイテムが表示されます。番号 0 は 6 つの製品カテゴリのインデックスです。

デフォルト Webflow 内の位置

このアプリケーションの Webflow により、直前のページで選択されたカテゴリのデータがロードされました。catalog.jsp ポートレットには、発生したカタログ イベントに応じて、さまざまなインクルード ファイルの JSP コードが組み込まれます。たとえば、以下の catalog.jsp の webflow:getProperty タグは、Pipeline セッションから最後のカタログ イベントを取得します。

```
<webflow:getProperty
    id="event"
    type="java.lang.String"
    property="<%= B2CPortalConstants.LAST_CATALOG_EVENT_ATTRIB %>"
    scope="session"
    namespace="portal" />
```

catalog.jsp ポートレットでは、返されたイベント属性が、以下のようにチェックされます。

```
...
<%
    }
    else if ( ( event.equals( "link.category" ) ) ||
              ( event.equals( "button.category.buy" ) ) ||
```

```
                ( event.equals( "button.category.save") )   )
    {
%>
...
<jsp:include page="/portlets/catalog/category.jsp" flush="true" />
```

この場合、link.category イベントが発生したために、category.jsp ファイルが catalog.jsp ポートレットに取り込まれます。

動的データの表示

category.jsp ファイルでは、直前のページで選択したカテゴリを Pipeline セッションから取得します。

```
<webflow:getProperty
    id="category"
    type="com.beasys.commerce.ebusiness.catalog.Category"
    property="<%= PipelineSessionConstants.CATALOG_CATEGORY %>"
    scope="session"
    namespace="portal" />
```

次に、一連の JSP タグを使って、現在のカテゴリについてのプロパティを取得します。

例：

```
<catalog:getProperty
    id="headerImage"
    returnType="com.beasys.commerce.ebusiness.catalog.ImageInfo"
    object="<%= category %>"
    propertyName="image"
    getterArgument="<%= new Integer( CatalogItem.SMALL_IMAGE_INDEX
    ) %>" />
```

次に、ビュー イテレータを使って、このカテゴリの製品アイテムを Pipeline セッションから取得します。

```
<webflow:getProperty
    id="items"
    type="com.beasys.commerce.ebusiness.catalog.ViewIterator"
    property="<%= PipelineSessionConstants.CATALOG_ITEMS %>"
    scope="session"
    namespace="portal" />
```

次に製品アイテムを繰り返し参照して、catalog.jsp ポートレット内のそれぞれのプロパティを表示します。コーディングのサブセットを以下に示します。

```
<catalog:iterateViewIterator
    id="item"
    returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
    iterator="<%= items %>">
```

```
<catalog:getProperty
    id="itemImage"
    returnType="com.beasys.commerce.ebusiness.catalog.ImageInfo"
    object="<%= item %>"
    propertyName="image"
    getterArgument="<%= new Integer( CatalogItem.SMALL_IMAGE_INDEX
    ) %>" />

<catalog:getProperty
    id="currentPrice"
    returnType="com.beasys.commerce.axiom.units.Money"
    object="<%= item %>"
    propertyName="currentPrice" />
        ...
```

category.jsp には、数多くのボタン ([詳細]、[今すぐ購入]、[保留]) があるため、その他の作業も行います。たとえば、[詳細] リンクを作成するには、HTTP 要求パラメータを設定します。

```
<%
    String linkParams =
       HttpRequestConstants.CATALOG_CATEGORY_ID + "=" +
           java.net.URLEncoder.encode( categoryKey.getIdentifier()
             ) + "&" +
       HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
           itemKey.getIdentifier();
%>
```

category.jsp 内の同じイテレータを使っている状態で、各テーブルのセルにはページのプロパティが表示されます。コーディングの小さなサブセットを以下に示します。追加コーディングについては、category.jsp 内のテーブルを参照してください。

```
<a href="<portlet:createWebflowURL namespace="catalogportlet" event="link.item"
extraParams="<%=linkParams%>" />"><img src="<webflow:createResourceURL
resource="<%=itemImage.getUrl()%>"/>" width="187" height="139" alt=""
border="0"></a>
```

上記に示したコードが使われるのは、ユーザがアイテムの画像または [詳細] ボタンのグラフィックをクリックした場合です。

上記の HTML サブセットを見れば、ユーザが製品アイテムの画像をクリックした場合、次のページへの移行のために、link.item というイベントが生成されることがわかります。しかし、この最初の [Products] ページが表示される前にどんなイベントが発生したでしょうか。詳細については、次の節を参照してください。

イベント

直前の [Products] ページでは、catalog_index.jsp ファイルが catalog.jsp ポートレットに取り込まれると、ポートレットの HTML テーブルに以下のコードがインクルードされます。

```
<a href="<portlet:createWebflowURL namespace="catalogportlet"
event="link.category"
extraParams="<%=linkParams + java.net.URLEncoder.encode(
categoryKey.getIdentifier() )%>"
/>"><img src="<webflow:createResourceURL resource='<%=largeImage.getUrl()%>'/>"
width="180" height="136" alt="<%=shortDesc%>" border="0"></a>
```

バナー グラフィックまたは (catalog_index.jsp ファイルを表示していた) catalog.jsp 内の特定のカテゴリをクリックすると、link.category Webflow イベントがトリガされました。これによって、category.jsp ファイルが catalog.jsp ポートレットにロードされ、次にこのポートレットに、選択したカテゴリを構成する製品アイテムが表示されました。

すべてのカタログ イベントは、catalog.jsp ポートレットの以下の Webflow ファイルに定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\app
lication-sync\webapps\b2cPortal\catalogportlet.wf
```

catalogportlet.wf に定義されているイベントの 1 つは次のとおりです。

```
  <event event-name="link.category">
     <destination namespace="catalogportlet"
        node-name="getCategoryIP" node-type="inputprocessor"/>
  </event>
                  ...
  <processor-origin node-name="getCategoryIP"
     node-type="inputprocessor"><node-processor-info
     class-name="examples.e2e.b2c.catalog.webflow.GetCategoryIP"/>
     <event-list>
         <event event-name="success">
              <destination namespace="catalogportlet"
                 node-name="getCategory" node-type="pipeline"/>
         </event>
     </event-list>
  </processor-origin>
...
```

link.category イベントでは、getCategoryIP という入力プロセッサが使用されます。これは Java プログラムで、HTTP 要求から Category ID String を取り出し、ID String を検証し、ID String に基づいて CategoryKey を作成し、これをセッション スコープ (有効範囲) の PipelineSession に追加します。この入力プロセッサのソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\WEB-INF\src\examples\e2e\b2c\catalog\webflow\GetCategoryIP.java
```

カテゴリへのアイテムの割り当て

サンプル アプリケーションでは、ブラウザベースの WebLogic Portal Administration Tools の [カタログ管理] を使って、製品アイテムがカタログに割り当てられました。この節では、その基本手順を概説します。詳細については、WebLogic Portal のドキュメントを参照してください。

注意 : カタログに何百というカテゴリと何千何万というアイテムがある場合、WebLogic Portal Administration Tools を使用した割り当ては、明らかに非現実的な作業です。このような場合は、WebLogic Portal Administration Tools の代わりに、WebLogic Portal の DBLoader ツールを使うことができます。

以下の手順は、e2eDomain のサーバがまだ稼動中であると想定しています。e2eAppTools Web アプリケーションとサンプル アプリケーションは同時に実行できます。どちらの Web アプリケーションも同じ e2eApp エンタープライズ アプリケーションの一部であり、同じドメイン内で実行されます。

1. 新しいブラウザのウィンドウで以下の URL 形式を使うと、WebLogic Portal Administration Tools が起動します。

   ```
   http://<host>:<port>/e2eAppTools/index.jsp
   ```

   URL 内のローカル マシン名 (localhost) またはリモート ホスト名、および WebLogic Server がリスンしているポート番号を置き換えます。WebLogic Portal アプリケーションでは、デフォルトのポート番号は 7501 です。URL の「e2eAppTools」部分では大文字と小文字が区別されます。

   例 :

   ```
   http://localhost:7501/e2eAppTools/index.jsp
   ```

2. ユーザ名とパスワードの入力が要求されます。これらの値はサイトによって異なることがありますが、デフォルトは次のとおりです。

```
Username: administrator
Password: password
```

ログイン値は、大文字と小文字が区別されます。これらの資格情報が機能しない場合は、管理者に問い合わせてください。サンプルのインストール後に、管理者アカウントのデフォルト値が変更された可能性があります。

3. WebLogic Portal Administration Tools のメイン ページで、[カタログ管理] バナーのアイコンをクリックします。
4. [カタログ管理] ページで、下線付きの [カテゴリ] をクリックします。
5. [カタログ] 階層の表示画面で、アイテムの追加と削除を行うカテゴリやサブカテゴリをクリックします (このサンプル アプリケーションのカテゴリにはサブカテゴリがありません)。
6. カテゴリが階層表示されたら、その下線付きのリンクをクリックします。たとえば、AviPro カテゴリをクリックしたとします。ページの上方に、以下のテキストが表示されていることがわかります。

```
カテゴリの編集： AviPro
  必要な情報を入力して、[save] アイコンをクリックしてください。
  このカテゴリに割り当てられているアイテムを変更するには、ここをクリックしてください。
```

このページのテキスト「... ここ」のリンクをクリックします。

以下のような画面が表示されます。

[カテゴリに割り当てられたアイテム] テキスト ボックスには、すでにこのカテゴリにあるアイテムが表示されます。追加または削除するアイテムの検索は、キーワード、クエリ、非カテゴリ化アイテム (カテゴリに分けられていないアイテム) という 3 つのモードで実行できます。検索結果は左側のテキスト ボックスに表示されます。カテゴリにアイテムを追加するには、右矢印をクリックして右側のテキスト ボックスにアイテムを移動します。

詳細については、WebLogic Portal のドキュメントを参照してください。

WebLogic Portal Administration Tools を終了するには、ブラウザのウィンドウを閉じます。

次のステップ

ツアーを続けるには、カテゴリの **AviPix 5000** 製品アイテムの画像をクリックします。

Product Item ポートレット

製品アイテムの詳細は、カタログからロードされます。ここまでのプロセスで、設計チームは管理者と連携し、カタログの各アイテムについてより大きな画像と詳しい説明を決定しました。アイテムの詳細の指定には、WebLogic Portal Administration Tools の [カタログ管理] 領域を使用しました。その結果、ポートレットの画面は以下のようになります。

Product Item ポートレットの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

WebLogic イベントの処理によって、発生する **`link.item`** イベントが決定され、その結果、`category.jsp` ファイルが中心となる `catalog.jsp` ポートレットにロードされました。

現在のサンプル アプリケーションのページに結果として表示されたサンプルの URL は、以下のとおりです (ここでは、見やすくするために何行かに改行してあります)。別の製品アイテムを選択した場合、サンプル ページの URL は異なります。

```
http://blues:7501/b2cPortal/application
?origin=category.jsp
&event=bea.portal.framework.internal.portlet.event
&pageid=Products&portletid=catalog
&wfevent=link.item
&wlcs_catalog_category_id=0avipix&wlcs_catalog_item_sku=pix5000
```

サンプルの URL では、最初の状態は 2 番目、3 番目、4 番目の行に示されています。Webflow イベントは最後の行に示されています。`link.item` イベントが発生しています。このイベントの結果が、サンプル アプリケーションで表示されているページです。直前のページで選択されたカテゴリ `0avipix` (この例の場合) の製品アイテムが表示されます。番号 0 は 6 つの製品カテゴリのインデックスです。この例では、選択された特定のアイテムは AviPix 5000 で、その在庫商品識別番号 (SKU) は `pix5000` です。

デフォルト Webflow 内の位置

このアプリケーションの Webflow により、直前のページで選択された特定の製品アイテムのデータがロードされました。`catalog.jsp` ポートレットには、発生したカタログ イベントに応じて、さまざまなインクルード ファイルの JSP コードが組み込まれます。たとえば、以下の `catalog.jsp` の `webflow:getProperty` タグは、Pipeline セッションから最後のカタログ イベントを取得します。

```
<webflow:getProperty
    id="event"
    type="java.lang.String"
    property="<%= B2CPortalConstants.LAST_CATALOG_EVENT_ATTRIB %>"
    scope="session"
    namespace="portal" />
```

catalog.jsp ポートレットでは、返されたイベント属性が、以下のようにチェックされます。

```
...
<%
    }
    else if ( ( event.equals( "link.item" ) ) ||
              ( event.equals( "button.item.buy" ) ) ||
              ( event.equals( "button.item.save") )   )
    {
%>
...
<jsp:include page="/portlets/catalog/item.jsp" flush="true" />
```

この場合、link.item イベントが発生したために、item.jsp ファイルが catalog.jsp ポートレットに取り込まれます。

動的データの表示

item.jsp ファイルでは、Pipeline セッションからアイテムのカテゴリを取得します。次に、一連の JSP タグを使って、現在のカテゴリとその現在のアイテムについてのプロパティを取得します。以下の例は、2 つの JSP タグを示しています。

```
<catalog:getProperty
  id="itemImage"
  returnType="com.beasys.commerce.ebusiness.catalog.ImageInfo"
  object="<%= item %>"
  propertyName="image"
  getterArgument="<%= new Integer( CatalogItem.LARGE_IMAGE_INDEX
  %>" />

<catalog:getProperty
  id="currentPrice"
  returnType="com.beasys.commerce.axiom.units.Money"
  object="<%= item %>"
  propertyName="currentPrice" />
```

item.jsp には、数多くのボタン ([今すぐ購入]、[保留]) があるため、その他の作業も行います。たとえば、[今すぐ購入] リンクを作成するには、そのグラフィックの HTTP 要求パラメータを設定します。

```
<catalog:getProperty
  id="itemKey"
```

```
  returnType="com.beasys.commerce.ebusiness.catalog.ProductItemKey"
  object="<%= item %>"
  propertyName="key" />

<%
  String linkParams =
     HttpRequestConstants.CATALOG_ITEM_SKU + "=" + itemKey.getIdentifier();
%>
```

item.jsp の以下のコードは、[今すぐ購入] ボタンのグラフィックをユーザがクリックすると、どのように URL が構成されるかを示しています。

```
<a href="<portlet:createWebflowURL namespace="catalogportlet"
   event="button.item.buy"
extraParams="<%= linkParams %>" />"><img src="<webflow:createResourceURL
resource='<%=imagesPath + "prod_step2_buynow.gif"%>'/>" width="63" height="18"
   alt=""
border="0"></a>
```

また、item.jsp ファイルでは、アイテムの価格表示を準備するときに、国際化 (I18N) JSP タグを使って、カタログに定義される通貨タイプを取得します。

```
<i18n:getMessage bundleName="/commerce/currency"
messageName="<%=currentPrice.getCurrency()
%>"/><%=WebflowJSPHelper.priceFormat(
currentPrice.getValue() )%>
```

イベント

直前の [Products] ページで、特定の製品アイテムをクリックすると、link.item Webflow イベントがトリガされます。その結果、item.jsp ファイルが catalog.jsp ポートレットにロードされます。

すべてのカタログ イベントは、catalog.jsp ポートレットの以下の Webflow ファイルに定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\webapps\b2cPortal\catalogportlet.wf
```

catalogportlet.wf に定義されているイベントの 1 つは次のとおりです。

```
<event event-name="link.item">
       <destination namespace="catalogportlet"
              node-name="getItemCategoryIP"
                     node-type="inputprocessor"/>
</event>
...
```

link.item イベントでは、getItemCategoryIP という入力プロセッサが使用されます。これは Java プログラムで、HTTP 要求から SKU 文字列を取り出し、SKU 文字列を検証し、SKU 文字列に基づいて ProductItemKey を作成し、これをセッション スコープ (有効範囲) の PipelineSession に追加します。この入力プロセッサのソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\
WEB-INF\src\examples\e2e\b2c\catalog\webflow\GetItemCategoryIP.ja
va
```

製品アイテム データの追加または変更

特定のアイテムのデータは、*カタログ データ*をカタログに追加する場合と同様の方法で、WebLogic Portal Administration Tools を使って追加します。この管理タスクをまだ読んでいない場合は、2-28 ページの「カテゴリへのアイテムの割り当て」を参照してください。

次のステップ

ツアーを続けるには、[Products] ページを下へスクロールして、商品評価ポートレットから **AviPix 5000** を選択します。[Go] ボタンをクリックすると、レーティングが Web サービスから返され、ポートレットに表示されます。次の節をお読みください。

商品評価ポートレットと Web サービス

商品評価ポートレットで製品アイテムを選択し、[Go] ボタンをクリックした後、製品レーティングの Web サービスの結果を反映してポートレットが更新されました (サンプルでは、結果を表示するために画面を下にスクロールする必要があります)。

アプリケーションの設計と開発のサイクル時には、BEA WebLogic Workshop を使用して製品レーティングの Web サービスを作成しました。次に、Portlet Wizard を使ってこの Web サービスのインタフェースを生成しました。

最後に、インタフェース コードを商品評価ポートレットにインクルードし、グラフィック アーティストによる初期段階の設計プロトタイプ作業に基づいて、表示コーディングを完了しました。また、第 4 章の「Web サービス ツアー」の説明にあるように、Web サービスを e2eApp 内の Web アプリケーションとしてパッケージ化しました。

商品評価ポートレットの技術詳細

商品評価ポートレット `evaluator.jsp` は、[Products] ページの下の方に表示されます。まず、ポートレットはインクルード ファイル `step1.jsp` を介して、割引広告を表示します。[Products] ページが更新され、商品評価ポートレットが具体的な製品アイテムがロードされたと判断すると、`step2.jsp` ファイルがインクルードされて、ログイン ユーザが製品レーティングを取得できるようになります。製品レーティングを検索するために WebLogic Workshop で以前作成した Web サービスを使うという点で、これは興味深いポートレットです。

WebLogic Workshop は視覚的な開発環境で、アプリケーション開発者も J2EE の専門家も、エンタープライズクラスの Web サービスの構築とデプロイを簡単に行えます。製品は次の 2 つの主要コンポーネントで構成されています。

1. 開発者が Java コードを記述して Web サービスを実装できる設計時ツール。
2. Web サービス インフラストラクチャ、アプリケーションのテスト、デバッグ、デプロイ環境を提供する実行時フレームワーク。

設計時ツールと実行時フレームワークの接点は、Java Web Service (JWS) ファイルと関連する制御 (CTRL) ファイルです。JWS ファイルは、追加機能を表すために (Javadoc 構文を使用して) 注釈が付けられた標準の Java ファイルです。注釈を使用すると、Web サービスとそのプロパティをグラフィカルに表示できます。また、フレームワークは注釈を使用して、EJB および J2EE のコードを生成し、Web サービスを実行します。通常、制御ファイルには、メソッド定義の集まりが含まれます、この定義によって、データベースや別の Web サービスなどのリソースに簡単にアクセスできます。

もう 1 つの重要なファイルは、Web サービスを説明する Web サービス記述言語 (Web Services Description Language: WSDL) ファイルです。WSDL ファイルは、Web サービスが利用可能なプロトコルの他、Web サービスがエクスポーズする

すべてのメソッドを (送受信可能な XML メッセージの形式で) 記述します。WSDL ファイルは、クライアント アプリケーションが Web サービスを使用するために必要な情報をすべて提供します。

開発では、WebLogic Workshop が提供するブラウザ ベースのテスト フォームを使用し、Web サービスから返される結果が予測どおりであるかを確認します。これをセット アップするために、**e2eApp** エンタープライズ アプリケーションの一部として、**workshop** Web アプリケーションをデプロイしました。Workshop の Web アプリケーションには、WebLogic Workshop で作成した `productEvalWSC` と `paymentWS` Web サービスが含まれます。結果的に、e2eDomain のサーバ インスタンスで、これらの Web サービスに WebLogic Workshop テスト ページを実行できます。テスト ページはブラウザベースです。たとえば、新しいブラウザ ウィンドウで、以下を開きます。

```
http://localhost:7501/workshop/productEvalWSC/EvalProduct.jws
```

URL の大文字と小文字は区別されます。Product Evaluator Web サービスの一連のテスト フォーム画面のツアーを完了するには、第 4 章の「Web サービス ツアー」を参照してください。以下のサンプル画面は、Web サービスと会話中のテスト フォームを 1 つだけ示しています。前回のテスト フォーム画面で、取得する評価データの `productId` (SKU) として、**pix1000** を指定しました。次の画面は、`getReliabilityRating` メソッドのテストを実行した直後のものです。

Web サービスが整数の **2** を返していることがわかります。商品評価ポートレットでは、この結果、最高 5 つの星のうち 2 つの星が表示されます。たとえば、次のように表示されます。

WebLogic Workshop で Web サービスを作成した後、WebLogic Workshop に付属の Portlet Wizard を使用して WSDL ファイルを指定し、クライアント インタフェースを生成できます。この処理の説明については、第 4 章の「Web サービス ツアー」を参照してください。この説明は、サンプル アプリケーションの「はじめに」のページにも記載されています。

表示アイテムの決定

商品評価ポートレットの JSP ファイルは、次のとおりです。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\evaluator\evaluator.jsp
```

まず、割引広告を表示する step1.jsp ファイルがインクルードされます。

[Products] ページ上の別のポートレットにあるイベントによって、商品評価ポートレットが具体的な製品アイテム データの存在を判断できる場合、商品評価ポートレットは製品レーティングを取得できる step2.jsp ファイルをインクルードします。step2.jsp ポートレットでは、最初にプルダウン メニューの項目が 1 つ以上表示されます。たとえば、次のように表示されます。

ポートレットのメニューからアイテムを選択し、[Go] ボタンをクリックすると、ポートレット内の step2.jsp ファイルは、Web サービスから返された結果で更新されます。たとえば、AviPix 5000 カメラを選択した場合、次のようになります。

evaluator.jsp ポートレットによって実行される最初のアクションは、Pipeline セッションから最後のカタログ イベントを取得することです。

```
<webflow:getProperty
  id="event"
  type="java.lang.String"
  property="<%= B2CPortalConstants.LAST_CATALOG_EVENT_ATTRIB %>"
  scope="session"
  namespace="portal" />
```

カタログ イベントが NULL の場合、静的な割引広告が表示されます。

```
<jsp:include page="/portlets/evaluator/step1.jsp" flush="true" />
```

この jsp:include タグは、最初の [Products] ページの下部に表示されていたグラフィックを取り込みます。

カタログ イベントが NULL でない場合、どのイベント タイプが発生したかはユーザが決定します。イベントが `link.category`、`button.category.buy`、`button.category.save` のいずれかの場合、`evalEvent` を「`browse`」に設定しました。イベントが `link.item`、`button.item.buy`、`button.item.save` のいずれかの場合、`evalEvent` を「`detail`」に設定しました。最後に、イベントが `button.search`、`button.search.buy`、`button.search.save` のいずれかの場合、`evalEvent` を「`search`」に設定しました。この値によって、どの項目 (複数可) を製品レーティングのプルダウン メニューに一覧表示するか、およびどのレーティング データを Web サービスから取得するかが決定されるため、ポートレットにはこの値が重要です。

`evalEvent` 値を Pipeline に渡すには、`webflow:setProperty` JSP タグを使います。

```
<webflow:setProperty
    property="B2CPortalConstants.PRODUCT_EVAL_ATTRIB"
        scope="request"
               value="<%=evalEvent%>"
               namespace="portal" />
```

Web サービスの使用

ここで、`Evaluator.jsp` が `step2.jsp` ポートレットをインクルードします。商品評価ポートレットが、自らに転記する場合、要求された製品アイテムを対象とした選択リストと共に `productEvalIP` 入力プロセッサの結果が表示されます。自らに転記しない場合は、製品アイテムの未選択リストが表示され、レーティング情報は (まだ) 表示されません。

step2.jsp ファイルには、次の Web サービスが含まれます。

```
<%@ page import="org.openuri.www.* "%>
<%@ page import="org.openuri.www.x2002.x04.soap.conversation.* "%>
<%@ page import="java.io.IOException" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="productEvalWSC.EvalProduct_Impl" %>
<%@ page import="productEvalWSC.EvalProductSoap" %>
```

Pipeline セッションから `evalEvent` 値を取得します。

```
<webflow:getProperty
    id="evalEvent"
        type="java.lang.String"
        property="B2CPortalConstants.PRODUCT_EVAL_ATTRIB"
        scope="request"
        namespace="portal" />
```

step2.jsp では、次にビュー イテレータを使用して、カテゴリを指定してアイテム (複数可) を取得するか、特定のアイテムを取得します。ユーザがアイテムを選択して [Go] ボタンをクリックした場合は、WebLogic Workshop で定義した EvalProduct.jws Web サービスを呼び出します。また、Portlet Wizard を使用して Web サービス インタフェース コードも作成しました。このコードには Web サービスの WSDL への参照が含まれています。

```
<%
  if (request.getParameter("origin").equals("step2.jsp") &&
      request.getParameter("wfevent").equals("button.evaluator.go"))
  {
         String productEvaluation = request.getParameter("productEvaluation");
%>
<!-- Portlet Wizard generated web services interfaces code -->

<%
         EvalProduct_Impl         m_Proxy = null;
         EvalProductSoap     m_ProxySoap = null;
         String m_conversationID = session.getId();
         String comments = "";
         int valueRating = 0;
         int reliabilityRating = 0;
         int overallRating = 0;
         boolean webServiceAvail = true;

         String serverNamePort = "http://" + request.getServerName() + ":"
            + request.getServerPort() + "/";

         try
         {
            m_Proxy = new EvalProduct_Impl(serverNamePort +
                      "workshop/productEvalWSC/EvalProduct.jws?WSDL");
         }
         catch (IOException ex)
         {
            ex.printStackTrace();
            webServiceAvail = false;
         }
            m_ProxySoap = m_Proxy.getEvalProductSoap();

         try
         {
```

```
        // Set up the header objects we'll need
        StartHeader startHeader =
              new StartHeader( m_conversationID,
                   serverNamePort
                   + "workshop/productEvalWSC/EvalProduct.jws");
                   ContinueHeader continueHeader =
                   new ContinueHeader( m_conversationID );
        // Start the conversation
        GetEvaluation getEvaluation = new GetEvaluation(productEvaluation);

        if (m_ProxySoap.getEvaluation(getEvaluation,
              startHeader).getGetEvaluationResult().equals("SUCCESS"))
        {
        // Continue the conversation
        comments =  m_ProxySoap.getComments(null,
                   continueHeader).getGetCommentsResult();

        // Continue the conversation
           reliabilityRating = m_ProxySoap.getReliabilityRating(null,
              continueHeader).getGetReliabilityRatingResult();

        // Continue the conversation
           overallRating = m_ProxySoap.getOverallRating(null,
              continueHeader).getGetOverallRatingResult();

        // Continue the conversation
           valueRating = m_ProxySoap.getValueRating(null,
              continueHeader).getGetValueRatingResult();
}
        ...
```

エラー処理などの追加コーディングについては、step2.jsp コードを参照してください。Web サービスから製品レーティングを受け取った後、その結果を step2.jsp の作業の一部としてポートレットに表示します。

商品評価ポートレットの開発中、WebLogic Workshop、および Portlet Wizard をどのように使用したかについては、「Web サービス ツアー」を参照してください。これは、このサンプルの「はじめに」ページからも参照できます。

次のステップ

ツアーを続けるには、製品アイテム AviPix 5000 の隣に表示されている [今すぐ購入] ボタンをクリックします。

[今すぐ購入] ボタンと、WebLogic Integration AI を介した在庫チェック

[今すぐ購入] ボタンをクリックすると、更新されたページの summarycart ポートレット (My ショッピングカート) で [現在のアイテム] リストにアイテムを追加した Webflow イベントがトリガされます。同じ製品アイテムに対して [今すぐ購入] を 2 回以上クリックすると、数量カラムが更新されます。価格と割引情報を含む完全なショッピング カートは、[チェックアウト] ボタンまたは [Shopping Cart] タブをクリックするまで表示されません。次のサンプル画面では、AviPrint 200 カメラがすでに入っているカートに、AviPix 5000 カメラが追加されました。サンプル アプリケーションでは、カートの [現在のアイテム] リストと [保存されているアイテム] リストが異なる場合があります。

[今すぐ購入] ボタンの技術詳細

[今すぐ購入] ボタンは、ユーザがカテゴリ ページ、あるいはアイテムの詳細ページまたは検索結果ページを参照しているときに、特定のアイテムに対して選択できます。イベント タイプは、ページによって異なりますが、catalogportlet.wf Webflow ファイルに定義されているとおり、以下のいずれかとなります。

- button.category.buy
- button.item.buy
- button.search.buy

[今すぐ購入] ボタンを選択すると、アイテムがユーザのショッピング カートに追加される前に、注文を確実に履行できるように在庫チェックが実行されます。データベースには、製品アイテムについて、現在、最小、最大の在庫に関するデータを保存した在庫テーブルがあります。在庫テーブルには、WebLogic Integration の Application Integration (AI) コンポーネントから、読み込み専用モードでアクセスします。

たとえば、この b2cPortal の [Products] ページで、複数のポートレットに対して [今すぐ購入] ボタンをクリックすると、このチェックが行われます。ショッピング カートにすでに入っているアイテムの数量を更新しようとして、ユーザがショッピング カートの ste1.jsp ポートレットに新しい値を入力して [再計算] ボタンをクリックした場合にも、在庫チェックが実行されます。

在庫チェックの実装について説明する前に、複数のポートレット ページで [今すぐ購入] リンクがセット アップされる方法を見てみましょう。たとえば、catalog.jsp ポートレットにインクルードされるアイテム jsp ファイルからは、次のように設定されます。

```
<a href="<portlet:createWebflowURL namespace="catalogportlet"
   event="button.item.buy"
extraParams="<%= linkParams %>" />"><img src="<webflow:createResourceURL
resource='<%=imagesPath + "prod_step2_buynow.gif"%>'/>" width="63" height="18"
   alt=""
border="0"></a>
```

上記の HTML サブセットでは、ユーザが [今すぐ購入] 画像をクリックした場合、次のページへの移行のために、button.item.buy というイベントが生成されます。linkParams パラメータには、特定のアイテムについての情報が含まれます。このデータは、一連の <catalog:getProperty...> JSP タグにより、item.jsp 処理ですでに収集されています。その後に以下の処理が行われます。

```
<%
    String linkParams =
        HttpRequestConstants.CATALOG_ITEM_SKU + "=" + itemKey.getIdentifier();
%>
```

在庫チェック

在庫チェックは、CheckInventoryPC Pipeline コンポーネントで実装されます。このソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\
examples\e2e\b2c\shoppingcart\pipeline\CheckInventoryPC.java
```

この Pipeline コンポーネントは、在庫をチェックし、追加の値を入れます。このサンプル アプリケーションでは、Pipeline コンポーネントはショッピング カートの内容が変更される度に呼び出されます。本番環境では、パフォーマンス上の理由からビジネス ロジックに基づいてこの動作を変更した方がよい場合があります。たとえば、在庫の実際の数量を保持し、在庫にある数量とショッピング カートにある数量との差に基づいてさまざまなメッセージを示すことで、アイテムがショッピング カートに初めて入れられたときにだけ在庫をチェックするようにします。

CheckInventoryPC Pipeline コンポーネントは、InventoryProvider SPI と連動します。このソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\common\inventory\spi\*.java
```

この SPI は checkInventory リモート メソッドを含むステートレス セッション Bean です。このメソッドは、次の処理を行います。

- Application Integration (AI) を使って、あらかじめコンフィグレーションされた AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルを照会するために使われます。
- XML ヘルパーを呼び出して、AI からの応答を解析する。
- 応答に基づいて、利用可能な在庫数 (int) を返す。

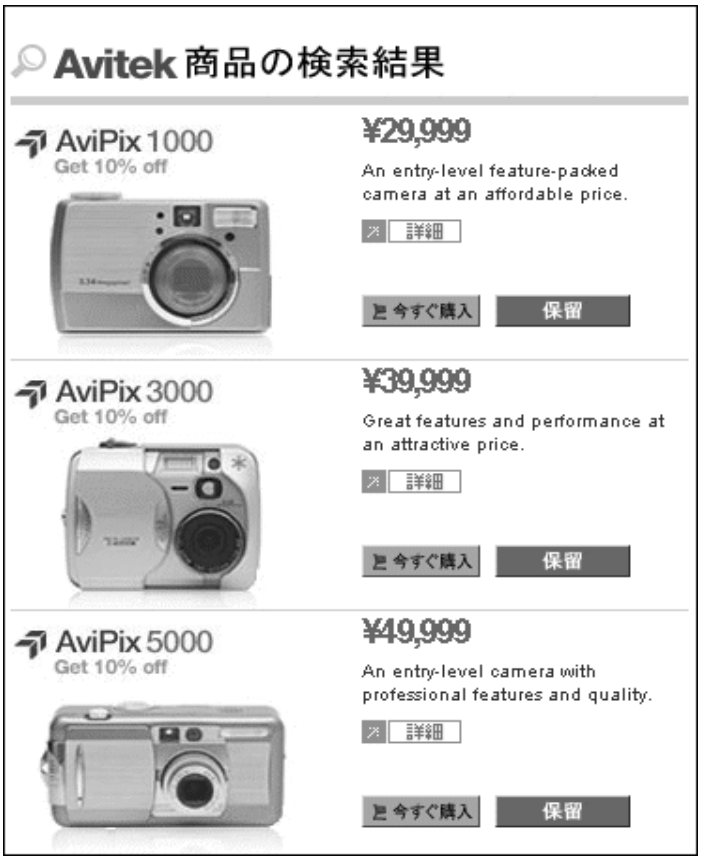
次のステップ

ツアーを続けるには、製品カタログをブラウズし、カートの [現在のアイテム] リストと [保存されているアイテム] リストに、製品アイテムを 4 ～ 5 個追加してください。カート内に多くのアイテムを入れると、[Shopping Cart] ページのさまざまなオプションを簡単に試すことができます。

その練習の一環として、Search ポートレットを使用してください。たとえば、キーワード **camera** を入力して、[Go] ボタンをクリックします。検索結果ページで、いくつかのアイテムを [現在のアイテム] リストと [保存されているアイテム] リストに追加します。次に、Search Results ポートレットについての以下の説明を読みます。

Search Results ポートレット

製品カタログに用意されている検索機能は、管理者が製品アイテムに割り当てたキーワードに基づいています。キーワードは WebLogic Portal Administration Tools の [カタログ管理] 領域を使って割り当てました。開発チームと管理者は協力して各アイテムに最も適したキーワードを決定できます。既存のキーワードについて、および検索の [Go] ボタンに関連付けられた Webflow イベントなどのイベントについては、この節をお読みください。次の画面は、「camera」と入力して [Go] をクリックした後の、Search Results ポートレットの表示の一部を示したものです。



Search Results ポートレットの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

search_results.jsp ポートレットには、製品アイテムについて、キーワードで検索した結果の情報が示されます。顧客は、この結果を参照できます。検索結果は、ポートレット、Webflow、カタログ、i18n (国際化) JSP タグの組み合わせでアセンブルされます。

一致が発生するには、すでに 1 つ以上の製品アイテムに関連付けられているキーワードを顧客が入力している必要があります。製品アイテムのキーワードは、Weblogic Portal Administration Tools の [カタログ管理] 領域で管理者が割り当てました。キーワードと WebLogic Portal Administration Tools の詳細については、後述します。

注意： WebLogic Portal コマース サービスでは、キーワード検索の他にクエリベースの検索を使用できます。このサンプル アプリケーションのスコープを簡素化するため、ここではキーワードベースの検索しか実装していません。クエリベースの検索の詳細については、WebLogic Portal のドキュメントを参照してください。

デフォルトでは、search_results.jsp ポートレットは以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\portlets\catalog
```

検索入力ボックス、[Go] ボタンのグラフィック、および処理を提供する searchform.jsp ポートレットは、デフォルトでは以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\portlets\search
```

開発者は、このサンプル アプリケーションを実行する際に、ブラウザに表示される前の JSP コードについて理解することが重要です。そのため、この節で使用されるコードの断片や [コードを表示] リンクでは、特定のポートレットのブラウザで表示する前の JSP ファイルについて説明し、これを示します。

デフォルト Webflow 内の位置

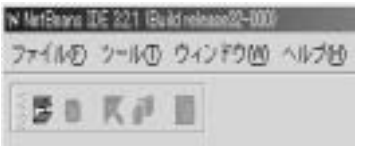
顧客が searchform.jsp ポートレットにキーワードを入力して [Go] ボタンをクリックすると、[Products] ページに search_results.jsp が表示されます。顧客は、search_results.jsp ポートレットから次の操作を実行できます。

- 結果リストに表示された特定のアイテムについての詳細を表示する (items.jsp ポートレットがロードされ、そこには itemdetails.jsp ポートレットの内容がインクルードされます)。
- [今すぐ購入] ボタンまたは [保留] ボタンをクリックして、検索結果リストに表示された製品アイテムを自分のショッピング カートに追加する。

WebLogic Platform は E-Business Control Center (EBCC) を備えています。これは、ルール定義、Webflow 編集、ポータルの作成および管理などの複雑なタスクを簡易化するグラフィック ツールです。EBCC の Webflow エディタと Pipeline エディタは、Webflow および Pipeline の XML コンフィグレーション ファイルの作成、変更、検証に役立つよう設計されています。必要に応じて、EBCC を起動し、`e2eApps` について、既存の Webflow 定義と Pipeline 定義を調べてください。

たとえば、以下の手順を行います。

1. Windows システムで、[スタート] メニューをクリックし、[プログラム | BEA WebLogic Platform 7.0 | WebLogic Portal 7.0 | E-Business Control Center] を選択します。
2. 画面の左上の [プロジェクトを開く] アイコンをクリックします。たとえば、次のように表示されます。



3. [プロジェクトを開く] ダイアログ ウィンドウから、以下のディレクトリに移動します。

   `weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project`

4. このディレクトリにインストールされているプロジェクト ファイル `e2eApp-project.eaprj` を選択して開きます。
5. [エクスプローラ] ウィンドウで、下方の [サイト インフラストラクチャ] タブを選択します。次に [Webflow/Pipelines] アイコンをクリックします。このアイコンを表示するために、下へスクロールしなければならない場合もあります。EBCC によって次のような画面が表示されます。

6. [b2cPortal] の横の正符号をクリックし、次に Webflow アイテムの searchportlet をダブルクリックします。searchportlet Webflow コンポーネントが、大きくグラフィック表示されます。コンポーネントを表示する画面スペースがもっと必要な場合は、隣接する [エクスプローラ] ウィンドウを閉じます。ここでは説明しませんが、Webflow エディタと Pipeline エディタには、多くのオプションがあります。*『開発者ガイド』*の「ポータル ナビゲーションのセットアップ」で、エディタの詳細説明をお読みください。

イベント

検索結果表示の直前のページでは、ユーザが検索入力ボックスにキーワードを入力し、[Go] ボタンをクリックしました。これは searchform.jsp ポートレットで行われました (別の Webflow シナリオは、別のページで [Products] タブをクリックしてこの [Products] ページに戻り、[Products] ページが最新のカタログ イベントを反映して更新され、それが検索結果になるというものでした。いずれのイベントでも、ここでは、元の検索関連イベントについてのみ説明します)。ポートレットに定義されているフォームには、以下が含まれます。

検索入力ボックスを含むテーブルのセル。

```
<input name="<%=HttpRequestConstants.CATALOG_SEARCH_STRING%>"
type="text" class="searchPortlet" size="27">
```

[Products] ページに結果を表示する追加 Webflow パラメータを定義し (検索は他のポータル ページからサブミットされるため)、検索ポートレットの Webflow を実行するスクリプトレット。

```
<%
String formParams=
  PortalAppflowConstants.PORTLET_WEBFLOW_EVENT_PARAMETER + "=" + "button.search"
    + "&" +
  PortalAppflowConstants.PAGE_PARAMETER + "=" +
  B2CPortalConstants.PRODUCTS_PAGE_NAME + "&" +
  PortalAppflowConstants.PORTLET_PARAMETER + "=" + "search" ;
%>
```

また、フォーム定義そのものは以下のとおりです。

```
<form method="POST" action="<webflow:createWebflowURL
event="bea.portal.framework.internal.portlet.event" origin="searchform.jsp"
extraParams="<%= formParams %>"/>" onsubmit="return ValidateSearchForm(this)" >
```

イベント **button.search** は、searchportlet ネームスペースの keywordSearchIP (入力プロセッサ) を使用して個別に定義されます。

これは、以下にある searchportlet.wf (Webflow) ファイルに格納されます。

```
BEA_HOME\weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-pr
oject\application-sync\webapps\b2cPortal
```

KeywordSearchIP.java は、HTTP 要求からキーワード検索 String を取り出し、その検索 String を検証し、これに基づいて KeywordQuery を作成し、これをセッション スコープの PipelineSession に追加します。そして以前の検索結果の PipelineSession をクリアします。キーワード検索 String が入力されていな

い場合、以前にキャッシュされた検索結果がないか PipelineSession が調べられます。以前にキャッシュされた有効な検索結果がある場合は、その結果が PipelineSession に保持されます。

この入力プロセッサのソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\WEB-INF\src\examples\e2e\b2c\catalog\webflow\KeywordSearchIP.jav
a
```

検索結果が使用可能になると、search_results.jsp ポートレット ページが表示されます。動的データ表示の詳細については、次の節を参照してください。

ここで、今後の処理のために search_results.jsp 内に定義されているイベントについて説明します。

- **link.item** イベントは、顧客が製品アイテムの画像 (AviPix 3000 カメラなど) またはアイテムの [詳細] ボタンの画像をクリックするとトリガされます。
- **button.search.buy** イベントは、顧客が検索結果の製品アイテムの隣に表示される [今すぐ購入] ボタンをクリックするとトリガされます。
- **button.search.save** イベントは、顧客が検索結果の製品アイテムの隣に表示される [保留] ボタンをクリックするとトリガされます。

以下の (あらかじめブラウザでレンダリングされた) search_results.jsp の行は、ポートレットの [詳細] ボタンに設定されたコーディングを示しています。

```
<a href="<portlet:createWebflowURL namespace="catalogportlet" event="link.item"
extraParams="<%=linkParams%
<" />"><img src="<webflow:createResourceURL resource='<%=imagesPath +
"prod_step2_details.gif"%>'/>
```

上記の例では、href リンクは、webflow:createResourceURL JSP タグを使用して構成されています。Webflow ネームスペースが catalogportlet であることがわかります。以下の場所にある catalogportlet.wf (Webflow) ファイルを見てみましょう。

```
BEA_HOME\weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-pr
oject\application-sync\webapps\b2cPortal
```

catalogportlet.wf では、link.item というイベントによって storeCatalogEventIP という入力プロセッサが識別されています。この入力プロセッサは、セッションスコープ化された Pipeline 属性として、現在のポートレット イベントを保存するために使用されます。ポートレットは、この情報を使用してカタログのどのビューを表示するかを決定します。

たとえば、Search Results ポートレット上で AviPix 1000 カメラの隣に表示されている [詳細] ボタンをクリックした場合、webflow:createResourceURL パラメータの値が、以下のコードでどのようになっているかに注目してください (ここでは、見やすくするために URL を何行かに改行してあります)。

```
http://localhost:7501/b2cPortal/application?origin=search_results
.jsp
&event=bea.portal.framework.internal.portlet.event
&pageid=Products&portletid=catalog
&wfevent=link.item&wlcs_catalog_item_sku=pix1000
```

動的データの表示

search_results.jsp では、一致した結果がある場合、その結果が Pipeline セッションから取り出されます。

例 :

```
<webflow:getProperty
  id="searchResults"
  type="com.beasys.commerce.ebusiness.catalog.ViewIterator"
  property="<%= PipelineSessionConstants.CATALOG_SEARCH_RESULTS%>"
  scope="session"
  namespace="portal" />
```

次のように、検索結果があるかどうかをチェックします。

```
<%
  if ( ( searchResults != null ) && ( searchResults.size() > 0 ) )
  {
%>
```

検索結果があれば、カタログを繰り返し参照して、各製品アイテムについてキーワードと一致する情報を取り出します。

例 :

```
<catalog:iterateViewIterator
    id="item"
    returnType="com.beasys.commerce.ebusiness.catalog.ProductItem"
    iterator="<%= searchResults %>">

    <catalog:getProperty
      id="itemImage"
```

```
returnType="com.beasys.commerce.ebusiness.catalog.ImageInfo"
object="<%= item %>"
propertyName="image"
getterArgument="<%= new Integer(CatalogItem.SMALL_IMAGE_INDEX
  ) %>" />
```

そのカタログで、以下に示す他のデータも製品アイテムごとに同じように取り出されます。currentPrice、(簡単な) description、productItemKey (SKU)。

次に、このアイテムの詳細リンク用に HTTP 要求パラメータを設定します。

一致した製品アイテムそれぞれのデータが、アイテムがなくなるまでビュー イテレータで繰り返されます。また、ポートレットが更新される場合は、検索結果のビュー イテレータをリセットします。

searchform.jsp に入力されたキーワードと一致するものが見つからない場合は、search_results.jsp で、Webflow Portal の i18n:getMessage JSP タグを使います。I18N とは国際化を略したもので、英語を母国語としない顧客向けにメッセージ テキストをカスタマイズするために、言語固有のファイルを追加または修正できるように、アプリケーション コードをセット アップするプロセスを指します。たとえば、search_results.jsp には、以下のコードが用意されています。

```
<i18n:getMessage messageName="noResults" bundleName="search_results" />
```

search_results.jsp と同じディレクトリにある search_results.properties ファイルには、以下が含まれます。

```
noResults=No matches found.
```

キーワードベース検索の管理タスク

このサンプル アプリケーションの検索機能では、WebLogic Portal Administration Tools の [カタログ管理] 領域で定義したキーワードが使われます。この節では、アイテムのキーワードを設定する方法について説明します。

以下の手順は、e2eDomain のサーバがまだ稼動中であると想定しています。e2eAppTools Web アプリケーションとサンプル アプリケーションは同時に実行できます。どちらの Web アプリケーションも同じ e2eApp エンタープライズ アプリケーションの一部であり、同じドメイン内で実行されます。

1. 新しいブラウザのウィンドウで以下の URL 形式を使うと、WebLogic Portal Administration Tools が起動します。

   ```
   http://<host>:<port>/e2eAppTools/index.jsp
   ```

URL 内のローカル マシン名 (`localhost`) またはリモート ホスト名、および WebLogic Server がリスンしているポート番号を置き換えます。WebLogic Portal アプリケーションでは、デフォルトのポート番号は 7501 です。URL の `e2eAppTools` 部分では大文字と小文字が区別されます。

例：

```
http://localhost:7501/e2eAppTools/index.jsp
```

2. ユーザ名とパスワードの入力が要求されます。これらの値はサイトによって異なることがありますが、デフォルトは次のとおりです。

   ```
   Username: administrator
   Password: password
   ```

   ログイン値は、大文字と小文字が区別されます。これらの資格情報が機能しない場合は、管理者に問い合わせてください。サンプルのインストール後に、管理者アカウントのデフォルト値が変更された可能性があります。

3. WebLogic Portal Administration Tools のメイン ページで、[カタログ管理] バナーのアイコンをクリックします。

4. [カタログ管理] ページで、下線付きの [アイテム] をクリックします。

5. [キーワード] 入力ボックスに、**camera** をキーワードとして入力します。

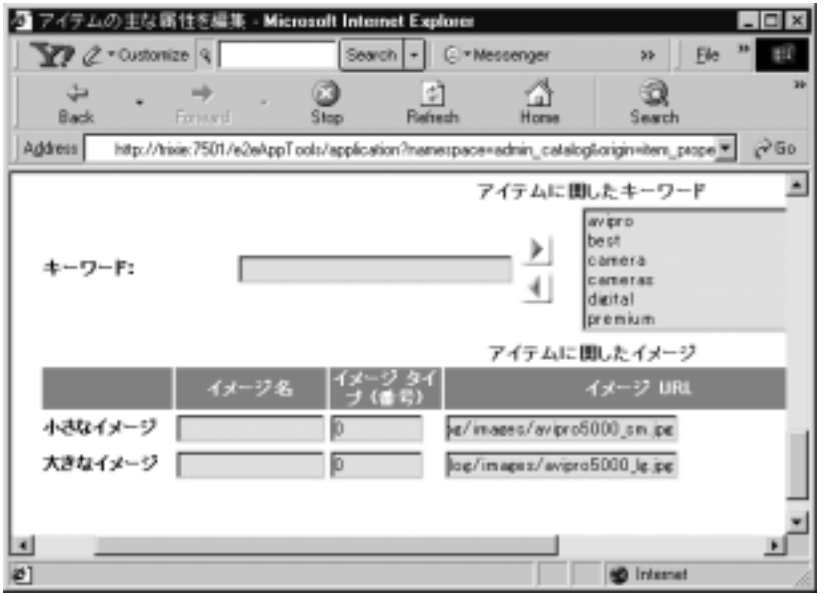
6. 下線付きの検索結果を 1 つクリックします。たとえば、AviPix 5000 をクリックします (警告 : **赤い X アイコン**はクリックしないでください。カタログからそのアイテムが削除されてしまいます。カタログ データの管理ページを使っていることを忘れないでください)。

7. 表示されたページで、「アイテムの主な属性」と同じ行にある [Edit] アイコンをクリックします。

サンプル アプリケーションのカタログに目を通していれば、[アイテム情報の編集] ページの値のいくつかは見慣れた値です。たとえば、簡単な説明や詳しい説明の値に注目してください。また、どのように大小グラフィックの場所が指定されたかにも注目してください (ページの下部)。まず、データ レコードはスクリプトから、または WebLogic Portal の DBloader プログラムを使ってロードされました。これらのカタログ管理ページには、値を操作するためのビューが用意されており、必要に応じて後で値を変更することができます (ほとんどの組織では、値を変更する際に、WebLogic Portal Administration Tools を使わずにカタログ データ スクリプトをもう一度実行します)。

検索キーワードは、この [アイテム情報の編集] ページで表示または変更できます。サンプル アプリケーションでは、カタログの各製品アイテムに対して一連のキーワードを定義しました。[キーワード] 入力ボックスと一連の既存のキーワードを確認するには、ページを下にスクロールします。

次のグラフィックはページの一部を示しています。



既存のキーワードは、affordable、avipix、camera、cameras、cheap、consumer、digital、pix です。

管理者は開発チームと協力して、サイトの訪問者が製品アイテムを見つけやすくなるように、キーワードをさらに決定して追加できます。

詳細については、WebLogic Portal のドキュメントを参照してください。

WebLogic Portal Administration Tools を終了するには、ブラウザのウィンドウを閉じます。

次のステップ

「一致するものが見つかりませんでした」が返された場合は、検索入力ボックスに camera など、既存の検索キーワードを 1 つ入力してください。ここでも、既存のキーワードは、affordable、avipix、camera、cameras、cheap、consumer、digital、pix です。次に [Go] ボタンをクリックします。

ツアーを続けるには、検索結果の中からアイテムをいくつかカートに追加します。次の節に進む前に、最後のイベントとして、アイテムの [保留] ボタンをクリックします。

[保留] ボタン

[保留] ボタンをクリックすると、更新されたページの `summarycart` ポートレット (My ショッピング カート) で [保存されているアイテム] リストにアイテムを追加した Webflow イベントがトリガされます。

[保留] ボタン イベントの技術説明

[保留] ボタンは、ユーザがカテゴリ ページ、アイテムの詳細ページ、検索結果ページのいずれかを参照しているときに、特定のアイテムに対して選択できます。イベント タイプは、ページによって異なりますが、catalogportlet.wf Webflow ファイルに定義されているとおり、以下のいずれかとなります。

- button.category.save
- button.item.save
- button.search.save

[保留] ボタンを選択すると、製品アイテムがユーザの [保存されているアイテム] リストに移されます。複数のポートレット ページでは、以下の方法で [保留] リンクが設定されます。たとえば、catalog.jsp ポートレットに含まれる item.jsp ファイルからは、次のように設定されます。

```
<a href="<a href="<portlet:createWebflowURL namespace="catalogportlet"
event="button.item.save" extraParams="<%= linkParams %>" />">
<img src="<webflow:createResourceURL resource='<%=imagesPath
+ "prod_step2_savelater.gif"%>'/>" width="72" height="18" alt="" border="0"></a>
```

上記の HTML サブセットでは、ユーザが [保留] 画像をクリックした場合、次のページへの移行のために、button.item.save というイベントが生成されます。linkParams パラメータには、特定のアイテムについての情報が含まれます。このデータは、一連の <catalog:getProperty...> JSP タグにより、item.jsp 処理ですでに収集されています。その後に以下の処理が行われます。

```
<%
    String linkParams =
        HttpRequestConstants.CATALOG_ITEM_SKU + "=" + itemKey.getIdentifier();
%>
```

[Products] ページで [保留] をクリックしても、在庫チェックは実行されません。ただし、現在 [Saved List] にあるアイテムについて [カートに追加] をクリックすると、ショッピング カートのページで在庫チェックが実行されます。

次のステップ

ツアーを続けるには、[チェックアウト] ボタンをクリックして My ショッピング カート `step1.jsp` ポートレットに進みます。

My ショッピング カート ポートレット、Step1.jsp

[Products] ページの `summarycart` ポートレットの [チェックアウト] ボタンをクリックした場合、ポートレットのアプリケーションの Webflow によって、[Shopping Cart] ページに `step1.jsp` チェックアウト ポートレットが表示されます。顧客が注文調達プロセスのどの段階にいるのか把握できるように、グラフィック `check_step1_header.gif` がロードされました。カート内にアイテムがある場合は、その注文の価格と割引情報が表示されます。表示されるのは、単価、単価からの 10% 割引 (AviPix Consumer Cameras のみ)、10,000 円を超える注文合計からの 15% 割引、合計額です。カートにアイテムがなければ、文字列「ショッピング カートに入っているアイテムはありません。」が表示されます。

Step1.jsp ポートレットの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

ショッピング カートのページには、顧客がチェックアウトし、注文を完了できる一連のポートレットが表示されます。チェックアウト処理中にユーザのショッピング カートが変更された場合は、在庫チェックが実行されます。この在庫チェックでは、`InventoryProvider` SPI と連動する `CheckInventoryPC` Pipeline コンポーネントが使われています。InventoryProvider SPI では、注文が確実に完了できるように WebLogic Integration の Application Integration (AI) コンポーネントが使用されます。

以降のショッピング カートのページでは、注文が発行されると、Pipeline コンポーネントが Payment Web サービスを呼び出して、クレジット カード購入を認可します。Web サービスは、WebLogic Workshop を使用してすでに作成済みです。

確定された注文がデータベースに保持された後で、別の Pipeline コンポーネントはその注文を XML 表示に変換し、これを Java Message Service (JMS) キューに置きます。WebLogic Integration Business Process Management (BPM) イベント プロセッサは、注文をキューから削除し、これを処理します。

ショッピング カートや チェックアウト ポートレットは以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\checkout\*
```

- **step1.jsp** は、現在のポートレットで、現在の保存されたアイテムを表す詳細な My ショッピングカート データを提供します。また、AviPix Consumer Camera カテゴリ アイテムの 10% 割引と 10,000 円を超える注文の合計金額からの 15% 割引も計算され、表示されます。step1.jsp 処理の詳細については、このページの「技術詳細」節に記載されています。

- **step2.jsp** は、ログイン ユーザ Rachel Adams のあらかじめ設定された請求と出荷情報を示すチェックアウト データを提供します。step2.jsp 処理の詳細については、2-73 ページの「チェックアウト ポートレット、Step2.jsp」に記載されています。

  注意： このサンプル アプリケーションのシンプルなスコープを維持するため、ユーザがクレジット カードの情報や出荷情報を入力したり、変更したりできるポートレットは除外しました。ただし、WebLogic Portal はこのタイプの処理をサポートしており、これを実演する別のサンプルが用意されています。WebLogic Portal のドキュメントの「コマースおよびキャンペーン ツアー」を参照してください。

- **step3.jsp** は、発行する注文の概要を示した (発行前の) [注文の送信] ページを提供します。ユーザが [注文の送信] をクリックすると、クレジット カードの認可が WebLogic Workshop で作成した Payment Web サービスから実行されます。step3.jsp 処理の詳細については、2-75 ページの「注文の送信ポートレット、Step3.jsp」節に記載されています。

- **step4.jsp** は、[注文の確認] ページを提供します。確定された注文は、Pipeline コンポーネントによって XML 表示に変換されます。Pipeline コンポーネントは、この注文のコピーを Java Message Service (JMS) キューに置きます。WebLogic Integration BPM イベント プロセッサは、注文をキューから削除し、これを処理します。step4.jsp 処理の詳細については、2-80 ページの「注文の確認ポートレット、Step4.jsp」節に記載されています。

在庫チェックと割引を含む Step1.jsp プロセス

step1.jsp My ショッピング カート ポートレットでは、JSP タグを使って Pipeline セッションからショッピング カート、保存されたショッピング カート、在庫数量を取得します。

```
<webflow:getProperty id="shoppingCart"
property="<%=PipelineSessionConstants.SHOPPING_CART%>"
type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart" scope="session"
namespace="portal" />
<webflow:getProperty id="savedShoppingCart"
property="<%=PipelineSessionConstants.SAVED_SHOPPING_CART%>"
type="com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart" scope="session"
namespace="portal" />
<webflow:getProperty id="inventoryCount"
property="<%=B2CPortalConstants.INVENTORY_CHECK%>" type="int[]"
scope="session" namespace="portal" />
```

ショッピング カートの各行を繰り返し参照した後で、不足気味の在庫についてのメッセージが必要かどうかを決定します。

例：

```
<% if ( (inventoryCount != null) && (inventoryCount[inventoryIndex]
<= shoppingCartLine.getQuantity() ) ) { outOfStock = true; %><%=
shoppingCartLine.getProductItem().getName() %> -
<i><i18n:getMessage bundleName="checkout"
messageName="over_inventory"/></i> - <% ;}
else { %><%= shoppingCartLine.getProductItem().getName() %> <% } %>
```

在庫チェックは、CheckInventoryPC Pipeline コンポーネントで実装されます。このソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src
\examples\e2e\b2c\shoppingcart\pipeline\CheckInventoryPC.java
```

この Pipleline コンポーネントは、在庫をチェックし、追加の値を入れます。このサンプル アプリケーションでは、Pipeline コンポーネントはショッピング カートの内容が変更される度に呼び出されます。たとえば、step1.jsp ページには、[再計算] ボタンが関連付けられた [数量] フィールドがあります。ユーザがこのポートレットの数量を変更すると、在庫が再びチェックされます。

注意： 本番環境では、パフォーマンス上の理由からビジネス ロジックに基づいてこの動作を変更した方がよい場合があります。たとえば、在庫の実際の数量を保持し、在庫にある数量とショッピング カートにある数量との

差に基づいてさまざまなメッセージを示すことで、アイテムがショッピング カートに初めて入れられたときにだけ在庫をチェックするようにします。

CheckInventoryPC Pipeline コンポーネントは、InventoryProvider SPI と連動します。このソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\common\inventory\spi\*.java
```

この SPI は checkInventory リモート メソッドを含むステートレス セッション Bean です。このメソッドは、次の処理を行います。

- WebLogic Integration Application Integration (AI) コンポーネントを使って、あらかじめコンフィグレーションされた AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルを照会するために使われます。
- XML ヘルパーを呼び出して、AI からの応答を解析する。
- 応答に基づいて、利用可能な在庫数 (int) を返す。

在庫チェックでは、over_inventory メッセージ名に注目してください。国際化 (I18N) への取り組みをサポートするために、これは以下のとおり別個のプロパティ ファイルである checkout.properties で定義されています。

```
# If the inventory cannot fulfill the quantity requested
over_inventory=out of stock
```

step1.jsp ポートレットでは、提示するラインアイテム割引の金額も計算されます。このサンプルでは、AviPix Customer Camera カテゴリのあらゆるカメラを 10% 割り引くことを選択しました。10,000 円を超える注文については、注文合計割引も計算されます。

step1.jsp ページでは、以下の多数のイベントが発生する可能性があります。

- button.updateShoppingCartQuantities。ユーザが数量フィールドを変更し、[再計算] ボタンをクリックすると発生します。
- button.deleteItemFromShoppingCart。ユーザが [現在のアイテム] リストの [削除] ボタンをクリックすると発生します。
- button.moveItemToSavedList。ユーザが [現在のアイテム] リストの [保留] ボタンをクリックすると発生します。
- button.deleteItemFromSavedList。ユーザが [保存されているアイテム] リストの [削除] ボタンをクリックすると発生します。

- button.moveItemToShopppingCart。ユーザが [保存されているアイテム] リストの [カートに追加] ボタンをクリックすると発生します。
- link.next。ユーザが [チェックアウト] ボタンをクリックすると発生します。

これらのイベントの詳細については、いずれかのボタンをクリックした後のイベントの技術詳細に関する節を参照してください。

注意： ユーザが step1.jsp ポートレットの [ショッピングを続ける] ボタンをクリックすると、以下の JSP タグを使ってこのユーザに [Products] ページが表示されます。

```
<a href="<portal:createPortalPageChangeURL  pagename="Products"
/>">
<img src="<webflow:createResourceURL
resource='<%=imagesPath + "check_step1_continue.gif"%>'/>"
width="94" height="18" alt="" border="0"></a>
```

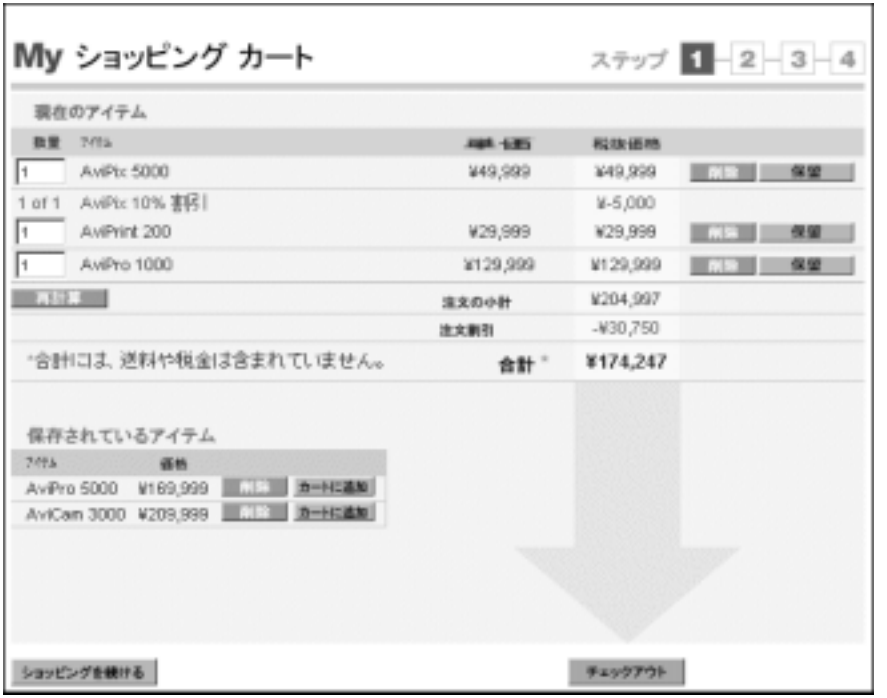
次のステップ

ツアーを続けるには、[保存されているアイテム] リストのアイテムの隣に表示される [カートに追加] ボタンをクリックします。

[保存されているアイテム] リストの [カートに追加] ボタン

ショッピング カート step1.jsp ポートレットの [保存されているアイテム] リストにある [現在のアイテム] リストには、カートに追加した製品アイテムが入っています。表示されるのは、単価、10,000 円を超えるアイテムの単価からの 10% 割引、10,000 円を超える注文合計からの 15% 割引、合計金額です。直前のページで選択したアイテムは、[保存されているアイテム] リストには表示されなくなります。

以下のサンプル画面に表示されているアイテムは、実際に表示されるカートの内容とは異なることがあります。



[保存されているアイテム] リストの [カートに追加] ボタンの技術詳細

`step1.jsp` [My ショッピング カート] ページには、`button.moveItemToShoppingCart` というイベントが含まれます。このイベントは、ユーザがショッピング カートの [保存されているアイテム] リストの [カートに追加] ボタンをクリックするとトリガされます。更新されたページでは、アイテムが [保存されているアイテム] リストから [現在のアイテム] リストに移されます。

step1.jsp ポートレット ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\checkout\*
```

`step1.jsp` からの例 :

```
<a href="<portlet:createWebflowURL namespace="checkoutportlet"
event="button.moveItemToShoppingCart"
extraParams="<%= extraParams %>" />">
<img src="<webflow:createResourceURL
resource='<%=imagesPath + "check_step1_add.gif"%>'/>"
width="52" height="13" alt="" border="0"></a>
```

extraParams パラメータには、step1.jsp で収集された製品アイテムについてのデータが含まれます。

```
<%
  extraParams = HttpRequestConstants.CATALOG_ITEM_SKU + "="
                shoppingCartLine.getProductItem().getKey().getIdentifier();
%>
```

button.moveItemToShoppingCart イベントは、以下のディレクトリにある checkoutportlet.wf Webflow ファイルで定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\webapps\b2cPortal
```

イベントは以下のとおりです。

```
<event event-name="button.moveItemToShoppingCart">
       <destination namespace="checkoutportlet"
       node-name="shoppingcart_MoveProductItemToShoppingCartIP"
       node-type="inputprocessor"/>
</event>
```

Pipeline コンポーネントは、MoveProductItemToShoppingCartPC です。その Java ソース ファイルは以下で確認できます。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src
\examples\e2e\b2c\shoppingcart\pipeline
```

MoveProductItemToShoppingCartPC は、顧客の保存されたアイテムのリストから ProductItem を削除し、これをショッピング カートに追加します。顧客のログイン ID は Pipeline セッションの PipelineSessionConstants.USER_NAME 属性で指定されます。移動するアイテムの SKU は、PipelineSessionConstants.CATALOG_ITEM_SKU Pipeline セッション属性で指定されます。PipelineSessionConstants.SAVED_SHOPPING_CART と PipelineSessionConstants.SHOPPING_CART という Pipeline セッションの属性、および WLCS_SAVED_ITEM_LIST テーブルは、その変更を反映して更新されます。

次のステップ

ツアーを続けるには、[保存されているアイテム] リストのアイテムの隣に表示される [削除] ボタンをクリックします。

[保存されているアイテム] リストの [削除] ボタン

ショッピング カート step1.jsp ポートレットの [保存されているアイテム] リストにある [削除] ボタンをクリックすると、そのページは同じポートレットに対する新しいデータを反映して更新されました。選択した製品アイテムは [保存されているアイテム] リストから削除されています。

[保存されているアイテム] リストの [削除] ボタンの技術詳細

step1.jsp [My ショッピング カート] ページには、button.deleteItemFromSavedList というイベントが含まれます。このイベントは、ユーザがショッピング カートの [保存されているアイテム] リストの [削除] ボタンをクリックするとトリガされます。更新されたページの [保存されているアイテム] リストにはアイテムが表示されなくなります。

step1.jsp ポートレット ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\checkout\*
```

step1.jsp からの例 :

```
<a href="<portlet:createWebflowURL namespace="checkoutportlet"
event="button.deleteItemFromSavedList"
extraParams="<%= extraParams %>" />">
<img src="<webflow:createResourceURL
resource='<%=imagesPath + "check_step1_remove.gif"%>'/>"
width="42" height="13" alt="" border="0"></a>
```

extraParams パラメータには、step1.jsp で収集された製品アイテムについてのデータが含まれます。

```
<%
   extraParams = HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
                 shoppingCartLine.getProductItem().getKey().getIdentifier();
%>
```

button.deleteItemFromSavedList イベントは、以下のディレクトリにある checkoutportlet.wf Webflow ファイルで定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\application-sync\webapps\b2cPortal
```

イベントは以下のとおりです。

```
<event event-name="button.deleteItemFromSavedList">
     <destination namespace="checkoutportlet"
     node-name="shoppingcart_DeleteProductItemFromSavedListIP"
     node-type="inputprocessor"/>
</event>
```

Pipeline コンポーネントは、DeleteProductItemFromSavedListPC です。その Java ソース ファイルは以下で確認できます。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\b2c\shoppingcart\pipeline
```

DeleteProductItemFromSavedListPC は、顧客の保存されたアイテム リストから ProductItem を削除します。顧客のログイン ID は Pipeline セッションの PipelineSessionConstants.USER_NAME 属性で指定されます。削除するアイテムの SKU は、PipelineSessionConstants.CATALOG_ITEM_SKU Pipeline セッション属性で指定されます。
PipelineSessionConstants.SAVED_SHOPPING_CART と PipelineSessionConstants.SHOPPING_CART という Pipeline セッションの属性、および WLCS_SAVED_ITEM_LIST テーブルは、その変更を反映して更新されます。

次のステップ

ツアーを続けるには、[現在のアイテム] リストのアイテムの隣に表示される [削除] ボタンをクリックします。

[現在のアイテム] リストの [削除] ボタン

ショッピング カート step1.jsp ポートレットの [現在のアイテム] リストにある [削除] ボタンをクリックすると、そのページは同じポートレットに対する新しいデータを反映して更新されました。製品アイテムとその価格はカートから削除されています。

[現在のアイテム] リストの [削除] ボタンの技術詳細

step1.jsp [My ショッピング カート] ページには、button.deleteItemFromShoppingCart というイベントが含まれます。このイベントは、ユーザがショッピング カートの [現在のアイテム] リストの [削除] ボタンをクリックするとトリガされます。更新されたページの [現在のアイテム] リストにはアイテムが表示されなくなります。

step1.jsp ポートレット ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\checkout\*
```

step1.jsp からの例 :

```
<a href="<portlet:createWebflowURL namespace="checkoutportlet"
event="button.deleteItemFromShoppingCart"
extraParams="<%= extraParams %>" />">
<img src="<webflow:createResourceURL
resource='<%=imagesPath + "check_step1_remove.gif"%>'/>"
width="42" height="13" alt="" border="0"></a>
```

extraParams パラメータには、step1.jsp で収集された製品アイテムについてのデータが含まれます。

```
<%
    extraParams = HttpRequestConstants.CATALOG_ITEM_SKU + "="+
                  shoppingCartLine.getProductItem().getKey().getIdentifier();
%>
```

button.deleteItemFromShoppingCart イベントは、以下のディレクトリにある checkoutportlet.wf Webflow ファイルで定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\webapps\b2cPortal
```

イベントは以下のとおりです。

```
<event event-name="button.deleteItemFromShoppingCart">
    <destination namespace="checkoutportlet"
    node-name="shoppingcart_DeleteProductItemFromShoppingCartIP"
    node-type="inputprocessor"/>
</event>
```

入力プロセッサは、DeleteProductItemFromShoppingCartIP です。その Java ソース ファイルは以下で確認できます。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\W
EB-INF\src\examples\e2e\b2c\shoppingcart\webflow
```

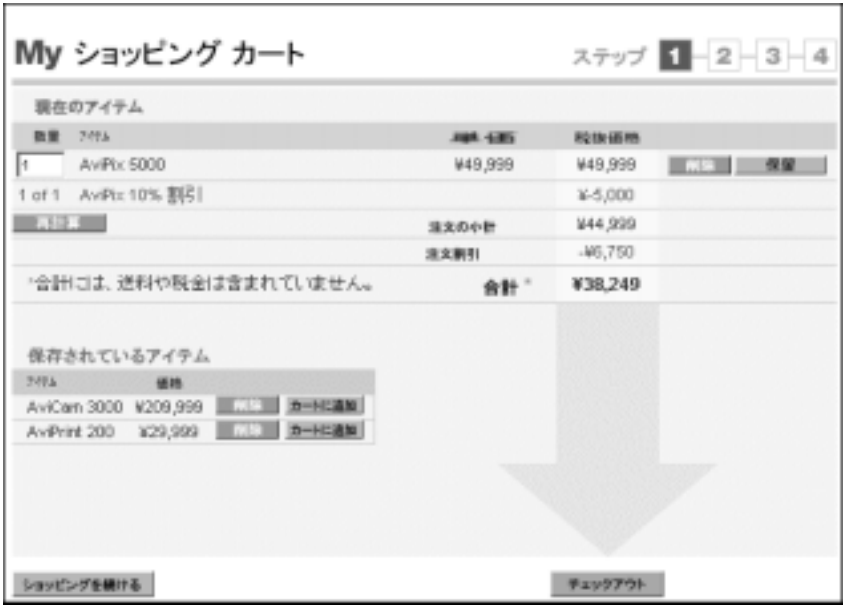
DeleteProductItemFromShoppingCartIP は、指定されたアイテムをショッピング カートから削除します。削除されるアイテムの SKU は要求オブジェクトの HttpRequestConstants.CATALOG_ITEM_SKU パラメータとして指定されます。ショッピング カートは、Pipeline セッションの PipelineSessionConstants.SHOPPING_CART 属性から取り出されます。アイテムは削除され、Pipeline セッションは変更されたショッピング カートで更新されます。

次のステップ

ツアーを続けるには、カートの [現在のアイテム] リストのアイテムの隣に表示される [保留] ボタンをクリックします。

[現在のアイテム] リストの [保留] ボタン

step1.jsp ショッピング カート ポートレットの [現在のアイテム] の部分にある [保留] ボタンをクリックすると、ページは同じポートレットに対する新しいデータを反映して更新されます。アイテムのステータスは変更され、カートの [保存されているアイテム] リストに移されました。



技術詳細

`step1.jsp` [My ショッピング カート] ページには、`button.moveItemToSaveList` というイベントが含まれます。このイベントは、ユーザがショッピング カートの [現在のアイテム] リストの [保留] ボタンをクリックするとトリガされます。更新されたページでは、アイテムが [保存されているアイテム] リストから [現在のアイテム] リストに移されています。

step1.jsp ポートレット ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\checkout\*
```

step1.jsp からの例：

```
<a href="<portlet:createWebflowURL namespace="checkoutportlet"
event="button.moveItemToSavedList" extraParams="<%= extraParams %>" />">
<img src="<webflow:createResourceURL
resource='<%=imagesPath + "check_step1_save.gif"%>'/>"
width="61" height="13" alt="" border="0"></a>
```

extraParams パラメータには、step1.jsp で収集された製品アイテムについてのデータが含まれます。

```
<%
    extraParams = HttpRequestConstants.CATALOG_ITEM_SKU + "=" +
                  shoppingCartLine.getProductItem().getKey().getIdentifier();
%>
```

button.moveItemToSavedList イベントは、以下のディレクトリにある checkoutportlet.wf Webflow ファイルで定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\app
lication-sync\webapps\b2cPortal
```

イベントは以下のとおりです。

```
<event event-name="button.moveItemToSavedList">
       <destination namespace="checkoutportlet"
       node-name="shoppingcart_MoveProductItemToSavedListIP"
       node-type="inputprocessor"/>
</event>
```

Pipeline コンポーネントは、MoveProductItemToSaveListPC です。その Java ソース ファイルは以下で確認できます。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\example
s\e2e\b2c\shoppingcart\pipeline
```

MoveProductItemToSavedListPC は、顧客のショッピング カートから ProductItem を削除し、これを顧客の保存されたアイテムのリストに追加します。顧客のログイン ID は Pipeline セッションの PipelineSessionConstants.USER_NAME 属性で指定されます。移動するアイテムの SKU は、PipelineSessionConstants.CATALOG_ITEM_SKU Pipeline セッション属性で指定されます。PipelineSessionConstants.SAVED_SHOPPING_CART と

PipelineSessionConstants.SHOPPING_CART という Pipeline セッションの属性、および WLCS_SAVED_ITEM_LIST テーブルは、その変更を反映して更新されます。

次のステップ

ツアーを続けるには、ショッピング カート step1.jsp ポートレットの下部にある [チェックアウト] ボタンをクリックします。

チェックアウト ポートレット、Step2.jsp

step1.jsp ショッピング カートの [チェックアウト] ボタンをクリックすると、ポータル アプリケーションの Webflow によって、[Shopping Cart] ページに step2.jsp チェックアウト ポートレットが表示されます。顧客が注文調達プロセスのどの段階にいるのか把握できるように、グラフィック check_step2_header.gif がロードされました。このサンプルを簡易化するために、ログイン ユーザのデータがページに表示されています。チェックアウト処理のステップ 2 ～ 4 では、注文データのセキュリティ暗号化にセキュア ソケット レイヤ (SSL) プロトコル (アプリケーションの URL の https://...) が使われていることに注意してください。

以下は、チェックアウト ポートレットです。スペースの理由からここでは 2 つに分けて示します。

下の図は、同じポートレットの下の部分です。

チェックアウト ポートレットの技術詳細

step2.jsp チェックアウト ポートレットには、ログイン ユーザ Rachel Adams のあらかじめ設定された請求と出荷情報が表示されます。このサンプル アプリケーションのシンプルなスコープを維持するため、ユーザがクレジット カードの情報や出荷情報を入力したり、変更したりできるポートレットは除外しました。ただし、WebLogic Portal はこのタイプの処理をサポートしており、これを実演する別のサンプルが用意されています。WebLogic Portal のドキュメントの「コマースおよびキャンペーン ツアー」を参照してください。

step2.jsp チェックアウト ポートレットは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\checkout\*
```

次のステップ

ツアーを続けるには、[注文を続ける] ボタンをクリックして step3.jsp の [注文の送信] ページに進みます。

注文の送信ポートレット、Step3.jsp

ショッピング カートの step2.jsp チェックアウト ポートレットで [注文を続ける] ボタンをクリックすると、ポータル アプリケーションの Webflow によって、この step3.jsp ポートレットが表示されます。概要情報が示されます。たとえば、次のように表示されます。

注文の送信ポートレットの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

`step3.jsp` チェックアウト ポートレットは、発行しようとする注文の概要を示した (発行前の) [注文の送信] ページを提供します。

このページでの注目すべき点とは、ユーザがページの下の方にある [注文の送信] ボタンをクリックした後で、何が起こるかということです。クレジット カードの認可は、`CajunBasedPaymentPC` という Pipeline コンポーネントを介して実行されます。これによって、WebLogic Workshop で作成した Payment Web サービスが呼び出されます。

注文が確認された後 (ステップ 4)、ConvertOrderRepPC という Pipeline コンポーネントによって、保持されている注文が XML 表示に変換され、これが Java Message Service (JMS) キューに置かれます。WebLogic Integration Business Process Management (BPM) イベント プロセッサは、注文をキューから削除し、これを処理します。そのプロセスの詳細については、2-80 ページの「注文の確認ポートレット、Step4.jsp」節を参照してください。

WebLogic Workshop Web サービスでの支払認可

step3.jsp ポートレットの [注文の送信] ボタンによって、link.next イベントが呼び出されます。checkoutportlet.wf Webflow ファイルには以下のコードが含まれています。

```
<presentation-origin node-name="step3" node-type="jsp">
    <node-processor-infopage-name="step3.jsp"page-relative-path="/portlets
        /checkout"/>
    <event-list>
        <event event-name="link.next">
            <destination namespace="checkoutportlet"
                node-name="Commit" node-type="inputprocessor"/>
```

この Webflow ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\webapps\b2cPortal
```

b2c_order.pln Pipeline ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\app
lication-sync\pipelines
```

このファイルには、以下のコードが含まれます。

```
<root-component component-name="CommitOrderPC"/>
  <component-branch-item>
    <source-component component-name="CommitOrderPC"/>
    <branch-success destination-component="CajunBasedPaymentPC"/>
  </component-branch-item>
```

クレジット カードの支払認可処理は、CajunBasedPaymentPC という Pipeline コンポーネントによって処理されます。これは、Pipeline コンポーネントが Payment Web サービスを呼び出すことを可能にする Java プロキシを呼び出します。Web サービスは、WebLogic Workshop を使用してすでに作成済みです。

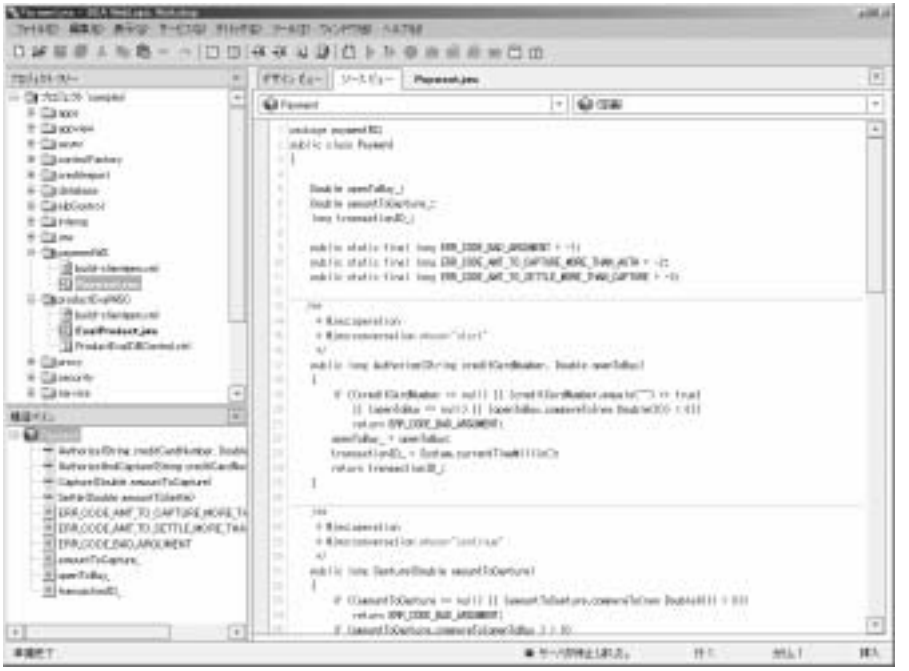
CajunBasedPaymentPC.java ファイルのソース ファイルは、以下の場所で確認できます。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\b2c\payment\pipeline
```

Payment Web サービスは、WebLogic Workshop フレームワークの対話的な機能を使用します。最初の呼び出しでクレジット カードを認可します。つまり、認可されるクレジット カード番号と金額を引数として渡します。クレジット カードの認可が終了すると、金額を取得するよう呼び出しが行われます。最後に、金額を決済する要求が行われます。

以下の画面は、Payment Web サービスの作成に使用した WebLogic Workshop Integrated Development Environment (IDE) の一部を示しています。最初の画面は、[デザイン ビュー] です。

WebLogic Workshop では、[デザイン ビュー] と以下の [ソース ビュー] を簡単に切り替えることができます。



Payment Web サービスから `CajunBasedPaymentPC` に返されるコードは、次のとおりです。

- -1 = Error (無効な引数が渡された)
- -2 = Error (取り込む金額がクレジットで認可されている金額を上回った)
- -3 = Error (決済金額がクレジット カードで取り込まれた金額を上回った)
- > 0 = Success

注意： Payment Web サービスは、サードパーティの支払サービスに接続および承認されている場合のように、常にエラーなく支払い情報を送信するわけではありません。Payment Web サービスを介した支払い処理は、本番環境向けに設計されていません。支払いを正確に処理するには、サードパーティ ベンダの支払いサービスと統合する必要があります。ただし、サンプルの Pipeline コンポーネントで表示されるコードは、エラーを適切に処理するよう設定されています。

CajunBasedPaymentPC Pipeline コンポーネントと Payment Web サービスの詳細については、第 4 章の「Web サービス ツアー」を参照してください。このサンプルの「はじめに」ページにも説明が記載されています。

[キャンセル] ボタンについて

step2.jsp または step3.jsp ショッピング カートのポートレットで [キャンセル] ボタンをクリックすると、アプリケーションの Webflow によってカートの step1.jsp に戻ります。カート内に現在あるアイテムおよび保存されているアイテムは、そのまま表示されています。キャンセルは、単にその注文のチェックアウト処理のステップ 2 または 3 を終了させるだけです。

次のステップ

ツアーを続けるには、[注文の送信] ボタンをクリックします。

注文の確認ポートレット、Step4.jsp

ショッピング カートの step3.jsp 注文の送信ポートレットで [注文の送信] ボタンをクリックすると、ポータル アプリケーションの Webflow によって、この step4.jsp ポートレットが表示されます。ここには、確認メッセージが一覧表示されます。保持されている注文は、JMS キューを介して、このポートレットとビジネス プロセス ワークフロー間の非同期通信を可能にする Pipeline コンポーネントによって発行されました。このインタラクションは、単一の WebLogic Server ドメイン インスタンスで稼動する WebLogic Portal と WebLogic Integration との間のアプリケーション統合を実証しています。

注文の確認ポートレットの技術詳細

注文がデータベースに保持されている場合は、同じ注文の XML 表示が注文管理用にキューに入れられます。作業は `ConvertOrderRepPC` Pipeline コンポーネントで開始されます。その Java ソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\b2c\order\pipeline
```

ConvertOrderRepPC Pipeline コンポーネントは、PurchaseManager SPI と連動します。このソース ファイルは、以下の場所にあります。

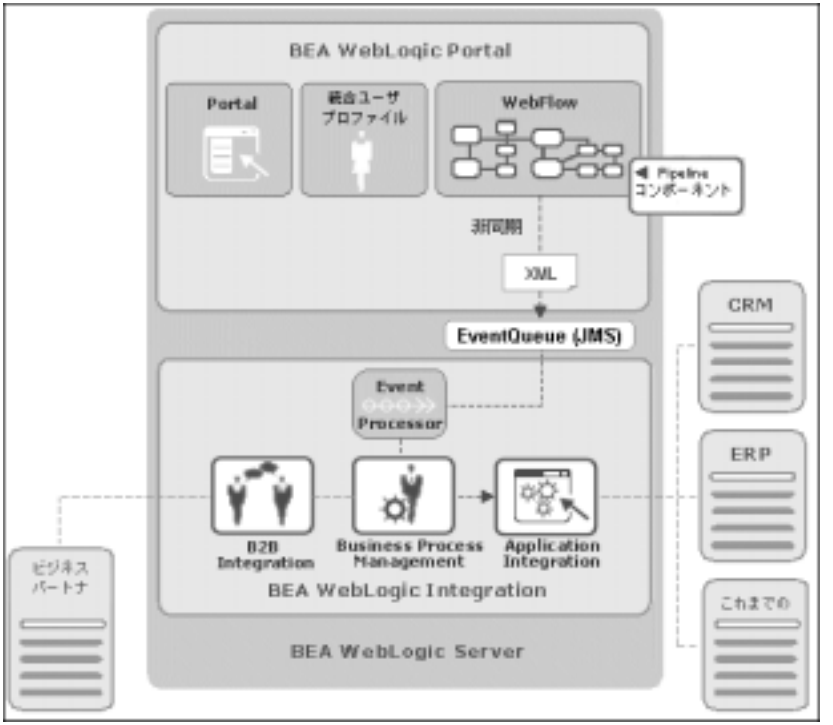
```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\common\purchase\spi\*.java
```

この SPI は、queueOrder リモート メソッドを含むステートレス セッション Bean です。

WebLogic Integration には、他のシステムとの通信に利用可能な以下のエントリポイントがあります。

- Application Integration (AI)— システムは、WebLogic Integration の AI エントリ ポイントを使って直接 J2EE Connection Architecture (J2EE CA) アダプタを呼び出すことができます。通常、このエントリ ポイントは、同期通信に使用します。
- 企業間統合 (B2Bi)— システムは、メッセージ API を通じて企業間統合 (B2Bi) を直接呼び出すことができます。このエントリ ポイントによって、開発者は WebLogic Integration の完全なインスタンスと通信するカスタム アプリケーションを記述することができます。また、B2Bi には、ホスト B2Bi イネーブラと通信できるシンクライアント インタフェースを構築するための JavaServer Pages (JSP) タグ ライブラリも用意されています。
- WebLogic Integration 処理エンジン — システムは、ネイティブ API への呼び出しを同期に利用して、または JMS を非同期に利用して WebLogic Integration 処理エンジンを呼び出すことができます。これらの呼び出しによって、データの送受信、プロセスの開始、ユーザ割り当てタスクの実行、エンタープライズ システム間での XML データの受け渡しなどが行われます。

このサンプル アプリケーションでは、JMS キューとの非同期通信を利用します。プロセスの流れを以下に図示します。



再び、`ConvertOrderRepPC` は、注文を JMS XML メッセージに変換し、これを BPM ワークフローのサブスクライブ先である JMS キュー (`com.bea.wlpi.EventQueue`) に送信します。次に BPM イベント リスナは、JMS キューからメッセージを取り出し、これを処理します。メッセージは、ワークフロー インスタンスを実行して、ワークフローを開始するか、リスンされるワークフロー イベントをトリガします。WebLogic Integration は、サンプル DBMS アダプタを使用して JMS キューから XML メッセージを取り出し、データをデータベースに転送します。サンプル DBMS アダプタは、WebLogic Integration でそのアプリケーション統合コンポーネントと共に使用するために提供されています。

XML メッセージは、ワークフローへの入力ドキュメントとして使われます。ワークフローは、次の 2 つのアクションに応答します。

Xpath を使って XML メッセージを解析し、すべての入力データをアプリケーション統合サービスに渡します。

ユーザが DBMS アダプタのアプリケーション ビューをデプロイするときに定義されるアプリケーション統合サービスを呼び出します。アプリケーション ビュー サービスはデータベースを更新します。

B2C ポータル ツアーの最終ステップ

以上で、注文は完了しました。[ショッピングを続ける] ボタンまたは [Products] タブをクリックすれば、オンライン ツアーのショッピングを続けることができます。または、[ログアウト] ボタンをクリックすると、このサンプルの [はじめに] ページに戻ることができます。ログアウトする場合、ショッピング カートのデータは次のセッションでは保持されないので注意してください。

3 Avitek の購入担当者とサプライヤとの接続

企業間取引 (B2B) ポータル ツアーでは、Avitek の購入担当者がサプライヤから見積もりを取り、注文書を発行してサプライヤから受付確認を受け取るための、架空のイントラネット サイトについて説明します。ビジネス プロセスは、WebLogic Integration によって管理されます。

注意： このオンライン ブックに掲載されている情報は、アプリケーションの一部として実行される状況依存ツアー ガイド ポートレットでも入手できます。

この企業間取引 (B2B) ポータル ツアーには、以下の節があります。

- Product Inventory ポートレット
- Product Parts Inventory ポートレット
- Query for Price and Availability (価格と在庫の照会) ポートレット
- Quotes for Price and Availability (価格と在庫の照会) ポートレットと QPA ビジネス プロセス
- Purchase Order for Review ポートレットと PO ビジネス プロセス
- Purchase Order History ポートレット

Product Inventory ポートレット

B2B ポータルのサンプル アプリケーションは、Avitek のイントラネットの [Inventory] ページから始まります。ログイン ユーザ Jason Tang がこのプロセスの手順をよりよく理解できるように、図 `processStep1.gif` がロードされています。初めてこのページを訪れた場合、Product Inventory ポートレットには、Avitek の販売商品の一部のデータがあらかじめロードされています。商品の在庫が最低水準を下回っている場合には、データが赤色で表示されます。

Product Inventory ポートレットの技術詳細

この節では、このページで行われる処理について詳しく説明します。

はじめに

b2bPortal は、架空の会社である Avitek Digital Imaging が使用するサンプルの企業間取引 (B2B) サイトです。Avitek の購入担当者は、このサイトを使って、外部のサプライヤから部品についての見積もりを取り寄せ、注文書を発行します。

b2bPortal は、e2eApp エンタープライズ アプリケーションの一部です。このエンタープライズ アプリケーションの名前には、e2e が含まれています。「e2e」とは、エンド ツー エンド (end-to-end) を略したもので、WebLogic Platform の主要機能のすべての範囲を示すサンプルであることを意味しています。

この [Inventory] ページの Product Inventory ポートレットには、Avitek の販売商品の一部のデータがあらかじめロードされています。商品の在庫が最低水準を下回っている場合には、データが赤色で表示されます。現在の在庫水準を確認するため、データベース内の在庫テーブルに、Application Integration (AI) フレームワークを介して読み込み専用モードでアクセスしました。AI フレームワークは、WebLogic Integration によって提供されます。

サンプル シナリオでは、Avitek Digital Imaging は、自動化、サイクル時間の短縮、在庫水準を下げることによって、サプライ チェーンを効率化し、コスト削減を図ろうとしていました。在庫水準を追跡して補充を行うために、Avitek は、ポータル ユーザ インタフェースを使って、販売側の B2B 交換、パートナとの提携、サプライ チェーン管理について、企業間取引ソリューションを実現しました。

このサンプル アプリケーションの B2C (企業消費者間取引) ポータル b2cPortal をすでにお使いの場合は、ポートレットと Webflow 処理についての説明が「技術詳細」の節に記載されていることをご存知でしょう。ポートレットと Webflow は b2bPortal でも重要な項目ですが、ここでは、Avitek 購入担当者用に実装されているサプライチェーン ソリューションに焦点をあてて説明します。ただし、この節でも、ポートレットと Webflow については一部詳細に説明します。

取引相手

この WebLogic Platform b2bPortal サンプル シナリオでは、バイヤ (Avitek Digital Imaging) 1 社とサプライヤ 2 社という取引関係を持つ 3 社が関わります。この 3 社それぞれに、取引相手が `BulkLoaderData.xml` ファイルでコンフィグレーションされます。サンプル用として、`E2E_Buyer`、`E2E_SupplierOne`、および `E2E_SupplierTwo` が取引相手として定義されています。

これらの取引相手間では、XOCP ビジネス プロトコルを使って通信を行うため、Avitek は、その WebLogic Integration システムをハブ & スポーク コンフィグレーションとして定義する必要があります。これを受けて、`BulkLoaderData.xml` ファイルで、第 4 の取引相手 `E2E_Hub` を定義します。B2B 統合のコンフィグレーションの詳細については、『B2B Integration 入門』の「B2B Integration の基礎」を参照してください。

この `E2E_Hub` という取引相手は、仲介者として機能します。スポークとなる `E2E_Buyer`、`E2E_SupplierOne`、および `E2E_SupplierTwo` という取引相手間のメッセージの仲介を担当します。`E2E_Hub` という取引相手は、業務メッセージの送信側でも受信側でもありませんが、トランザクション中は、必要に応じてプロキシ バイヤおよびプロキシ サプライヤとして機能します。

取引相手 3 社 (`E2E_Buyer`、`E2E_SupplierOne`、および `E2E_SupplierTwo`) はそれぞれ、取引相手 `E2E_Hub` と提携契約を結びます。取引相手 `E2E_Hub` は、提携契約を連結する役割を果たします。このような連結は、必要不可欠なものです。たとえば、1 つの提携契約の一部としてメッセージを受信し、そのメッセージを

他の提携契約の一部として別の取引相手に送る必要があります。同じ配信チャネル (取引相手 E2E_Hub に対して定義されているチャネル) を使用する提携契約が連結されます。

各取引相手要素は、さまざまな属性、サブ要素によって特徴付けられ、その中には、名前、電子メール、電話、FAX などの単純な識別情報も含まれます。

下の表は、各取引相手のロールをまとめたものです。

表 3-1 取引相手のロール

取引相手名	ロール
E2E_Hub	バイヤとサプライヤとの間の通信経路を管理し、企業間統合を提供する。
E2E_Buyer	ワークフロー テンプレートを使用して、サプライヤ間のビジネス プロセスと内部機能 (たとえば、バックエンド データベースの更新) の調整を行う。 アプリケーション ビューを使用して、バイヤのデータベース システムへの接続を提供する。 HTML ページや JSP ページを介して、データ表示とユーザ インタフェースを処理する。
E2E_SupplierOne	ワークフロー テンプレートを使用して、バイヤからの要求に応え、内部プログラム (データ変換およびデータ保持プログラム) を呼び出す。 データ変換を行って、アプリケーション間の情報交換の便宜を図る。
E2E_SupplierTwo	E2E_SupplierOne と同じロール。

WebLogic Integration Studio の使い方

WebLogic Integration Studio を使用すると、使い慣れたフローチャート図を使用して、新しいワークフローの設計、および実行中のワークフローの監視を行うことができます。この b2bPortal サンプルを実行する際に、WebLogic Integration Studio を実行する必要はありませんが、このサンプルのノードがどのように定義

され、コンフィグレーションされたかを学ぶために、いずれかのワークフローやワークフロー ノードの詳細を見るのは役に立つでしょう。また、サンプルを実行する際には、Studio を使用してそのワークフローを監視できます。

Windows システムで、[スタート ｜プログラム｜ BEA WebLogic Platform 7.0 ｜ WebLogic Integration 7.0 ｜ Studio] を選択します。

以下を入力して、Studio にログオンします。

- ユーザ : `admin`
- パスワード : `security`
- サーバ : `t3://localhost:7501`

例 :

注意 : http ではなく **t3** を使用します。

Studio でのワークフロー テンプレートの表示

Studio 内で、ワークフロー テンプレートとそのプロパティを表示するには、以下の手順に従います。

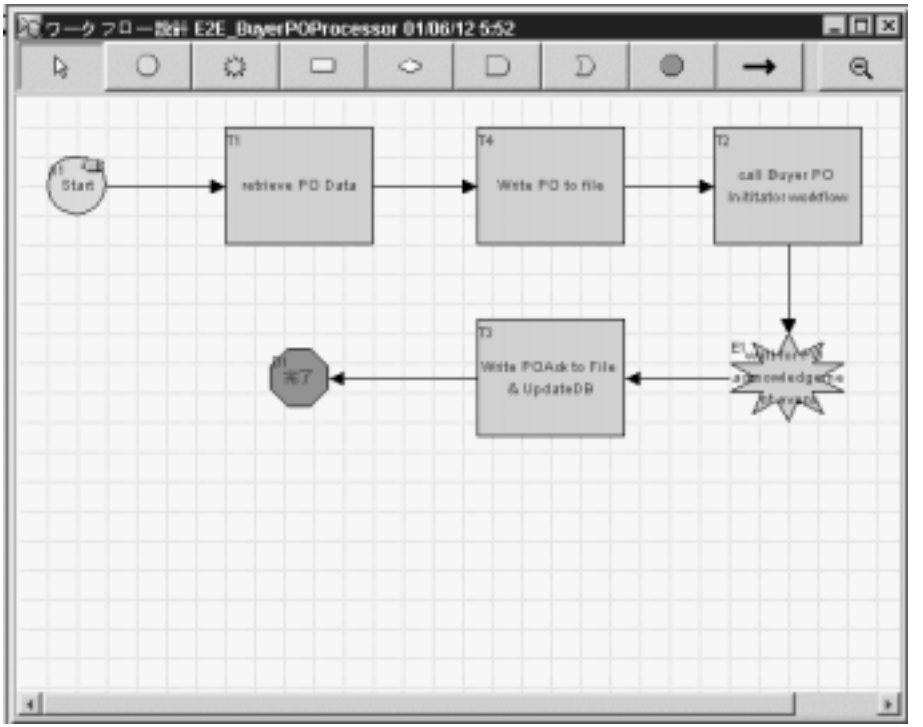
1. Studio の左ペインの [オーガニゼーション] フィールドで「ORG1」が選択されていることを確認します。

2. 左ペインで、テンプレート フォルダをダブルクリックし、ワークフロー テンプレートの一覧を表示します。
3. テンプレート フォルダを展開し、ワークフロー テンプレート定義の一覧を表示します。これらは、このサンプル用にあらかじめコンフィグレーションされている workflows.jar ファイルで定義されています。この worksflows.jar ファイルは、以下の場所にあります。

   ```
   weblogic700\samples\integration\samples\e2e\workflows
   ```

4. テンプレート定義を右クリックし、[開く] を選択して Studio 内のワークフロー テンプレートを開きます。たとえば、次のように表示されます。

この手順により、Studio 内にテンプレートのグラフィカルな表示が開きます。たとえば、次のように表示されます。



注意： また、特定のワークフロー テンプレート定義を展開して、そのワークフロー テンプレート定義用のタスク、Decisions、イベント、結合、開始、完了、変数などのフォルダを表示することもできます。

Studio 内のいずれかのノードをダブルクリックすると、そのノードの [プロパティ] ダイアログ ボックスが表示されます。Studio ツールと機能の詳細については、WebLogic Integration のドキュメントの『WebLogic Integration Studio ユーザーズ ガイド』を参照してください。

ビジネス プロセスとワークフローのモデル化

この節では、WebLogic Integration が提供するビジネス プロセス管理 (Business Process Management: BPM) を簡単に紹介します。

会話定義で取引相手に割り当てられたロールを実現するワークフローを *コラボレイティブ ワークフロー*といいます。

1 つのワークフロー テンプレートが 1 つのワークフローを表し、その実装のさまざまなワークフロー テンプレート定義 (バージョン) を組み合わせます。ワークフロー テンプレートの設計と編集は、WebLogic Integration Studio で行います。以下のような BPM プラグインにより、WebLogic Integration Studio の機能を拡張できます。

- B2B Integration プラグイン — B2B の統合をサポートします。つまり、コラボレイティブ ワークフローの設計と管理を行います。Studio により、ワークフローにプロパティを割り当てることができます。これらのプロパティにより、B2B 統合環境でワークフローが使用可能になります。
- Application Integration プラグイン — バックエンド システムと従来のエンタープライズ情報システム (Enterprise Information System: EIS) を統合するワークフローの設計を可能にします。
- Data Integration プラグイン — 異種の EIS アプリケーション間でデータを共有できるようにすることで、各種のデータ形式を統合するワークフローの設計を可能にします。

このサンプル シナリオでは、取引相手は、*プライベート ワークフローとコラボレイティブ ワークフロー*の両方を実装しています。プライベート ワークフローは、コラボレイティブ ワークフローと組み合わせて動作し、その取引相手に対するローカル処理を実装します。ローカル処理とプライベート処理は、会話定義で指定する必要はありません。たとえば、取引相手が会話を開始する際、その取引相手のプライベート ワークフローは、コラボレイティブ ワークフローを開始して、会話を開始することができます。

詳細については、WebLogic Integration のドキュメントを参照してください。このサンプルでは、続くポータル ページの「技術詳細」の節で、このサンプル アプリケーションで実装されている「価格と在庫の問い合わせ (Query for Price and Availability: QPA)」と「注文書 (Purchase Order: PO)」という 2 つのビジネス プロセスについて説明します。

在庫ページ ポートレット

[Inventory] ページは、b2bPortal 内の 3 つのタブから構成されるページの 1 つです。WebLogic Platform をインストールした `BEA_HOME` ディレクトリの以下の場所に、`e2eApp` を構成するファイルがあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\...
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\...
```

この [Inventory] ページには、以下のポートレットを追加することができます。

- **login** ポートレット。認証済みのユーザがアプリケーションにログインする際に使用します。[ログアウト] ボタンおよび現在ログイン中のユーザ名を表示します。
- **productinventory** ポートレット。購入担当者が管理する商品の在庫水準を特定します。
- **purchasingprocess** ポートレット。購入担当者に注文処理の流れを示すため、バナー内に適切なグラフィックを表示します。
- **b2b-tourguide** ポートレット。実行中のサンプルの状況依存ドキュメントを表示します。このドキュメントには、実行中のアプリケーションの左側に小さく表示されるものと、現在示しているように [技術的詳細を参照]、[コードを表示]、[e-docs を参照] というポインタを含む最大化されたものの 2 つの形式があります。

[Inventory] ページを構成するすべてのポートレットは、以下のファイルに定義されています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\webapps\b2bPortal\b2bPortal.portal
```

例：

```
<page-name>Inventory</page-name>
...
    <portlet-pool>
        <portlet-name>login</portlet-name>
        <portlet-name>partinventory</portlet-name>
        <portlet-name>productinventory</portlet-name>
        <portlet-name>purchasingprocess</portlet-name>
        <portlet-name>b2b-tourguide</portlet-name>
        <portlet-name>debug</portlet-name>
        <portlet-name>anonUser</portlet-name>
    </portlet-pool>
```

これらのポートレットは、開発サイクル時に E-Business Control Center を使用して [Inventory] ページに追加されました。EBCC は、Java クライアントベースのツールの集まりです。このツールには、ルールの定義、Webflow の編集、ポータルの作成と管理などの複雑な作業を簡単に行えるグラフィカル インタフェースが用意されています。E-Business Control Center のユーザは、ポイントアンド

クリック方式のインタフェースで作業を行い、サーバと同期をとる XML ファイルを生成します。EBCC の他にも、実行時のポータルの管理に、ブラウザベースの WebLogic Portal Administration Tools を使用しました。

ポータルの開発者は、主にブラウザに表示される前のポートレットの JSP コードに注目します。そのため、この節で使用されるコードの断片や [コードを表示] リンクでは、特定のポートレットのブラウザで表示する前の JSP ファイルについて説明し、これを示します。

b2bPortal のポートレット JSP ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2bPortal
\portlets\...
```

この [Inventory] ページでは、[コードを表示] リンクで `\productinventory\content.jsp` ポートレット ファイルのソースを開きます。これは Avitek Product Inventory (1) ポートレットを表します。この content.jsp は、前述のパスの下の productinventory サブディレクトリにあります。

初期ポータル処理の概要

サンプル アプリケーションを起動したとき、サーバが起動し、[はじめに]、すなわち「スプラッシュ」ページが最初に表示されたのは、どういう仕組みによるのでしょうか。まず、サンプル アプリケーションを呼び出す方法にはいくつかの方法があり、WebLogic Platform Quick Start Application から起動する方法や、`StartE2E` スクリプトの実行という直接的な方法がありました。

どのオプションを使用したかに関わらず、`startE2E.bat` (Windows) または startE2E.sh (UNIX) スクリプトが呼び出されました。これで、アプリケーションの WebLogic Server インスタンスが起動されましたが、このアプリケーションは、`e2eDomain` というドメインで実行されるものです。「ドメイン」という言葉は、コンピュータ業界では、さまざまな意味を持っています。BEA 製品でドメインというとき、それは、サーバ、サービス、インタフェース、マシン、関連するリソース マネージャなど、すべて 1 つのコンフィグレーション ファイルで定義されるものの集合を表します。

`startE2E` スクリプトを実行すると、エンタープライズ アプリケーションの `config.xml` ファイルからコンフィグレーション情報が読み込まれます。デフォルトでは、このコンフィグレーション ファイルは、BEA 製品がインストールされている以下の `BEA_HOME` ディレクトリにあります。

```
weblogic700/samples/platform/e2eApp/config
```

config.xml ファイルは以下のように定義され、そのドメインのデフォルト Web アプリケーションとして splashPage を設定しています。

```
<WebServer
       DefaultWebApp="splashPage"
       LogFileName="./logs/access.log"
       LoggingEnabled="true"
       Name="e2eServer"
       />
```

e2eServer が稼動した状態で、http://localhost:7501 などの URL をブラウザで指定すると、以下の場所にある splashPage Web アプリケーションが起動します (Web アプリケーションは「webapp」と略称されることもあります)。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\splashPage
```

splashPage Web アプリケーションの web.xml コンフィグレーション ファイルでは、<welcome-file-list> 定義の中で index.jsp を指定しました。この web.xml は、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\splashPage\
WEB-INF
```

スプラッシュ ページで、[購入エージェント「Jason Tang」として自動ログイン] ボタンのグラフィックをクリックして、この [Inventory] ページを表示しました。その結果、スプラッシュ ページから、ポータル アプリケーションの URL と定義済みのログイン資格情報が渡されることになりました。

スプラッシュ ページは、b2bPortal または b2cPortal に自動的にログインできるようにする要求を設定します。これは、このサンプル アプリケーションを簡素化するために行われます。JSP ページにユーザ名とパスワードを組み込むようなことは、普通は行いません。

フォームは、結果として以下のように構成された URL を開きます。

```
<%
       String b2bUrl = "http://" +
              request.getServerName() + ":" +
              request.getServerPort() + "/" +
              B2B_PORTAL_NAME + "/application";
...
```

この WebLogic Platform サンプル アプリケーションでは、サンプルを最も重要な機能に特化させるために、既存のユーザを自動的にログインすることを選択しました。WebLogic Portal は、その他にも、ログイン認証コードや、Web アプリケーションによるデモグラフィック情報収集テクニックを示すサンプルも提供します。詳細については、*『開発者ガイド』*を参照してください。

b2bPortal アプリケーションについて、ブラウザに URL が渡されたとき、[Inventory] ページが最初に表示されたのはなぜでしょうか。このプロパティは、WebLogic Portal Administration Tools で設定しました。このツールの [ポータル管理] セクションの [ページおよびポートレット] の下の [ページの選択と順序設定] 画面で b2bPortal のデフォルト ページを設定しました。この値は、データベースの `PORTAL_PAGE_P13N` テーブルの `INDEX_NUMBER` カラムに格納されます。

ページ切り替え Webflow イベント

顧客が JSP 上のリンクやボタンをクリックすると、それが 1 つのイベントであると見なされます。イベントは、デフォルト Webflow 内の特定の応答をトリガし、顧客が処理を継続できるようにします。この応答から別の JSP がロードされる場合もありますが、通常は、入力プロセッサや Pipeline が最初に呼び出された場合に行われます。

[Purchasing] ページまたは [Order History] ページで、[Inventory] タブをクリックすると、以下のような URL が表示されます (ここでは、見やすくするために何行かに改行してあります)。

```
http://<host>/<port>/b2bPortal/application?
origin=hnav_bar.jsp&event=bea.portal.framework.internal.refresh
&pageid=Inventory
```

更新イベントにより、最新データで更新されたページが再表示されます。Avitek Product Inventory ポートレットのデータが更新されます。

ページ タブは、`hnav_bar.jsp` ファイルを介して提供されます。このファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2bPortal
\framework
```

`hnav_bar.jsp` ファイルによって、以下の 2 つの JSP タグ ライブラリがインポートされます。

```
<%@ taglib uri="webflow.tld" prefix="wf" %>
<%@ taglib uri="portal.tld" prefix="ptl" %>
```

アプリケーション内の別のページから [Inventory] タブをクリックすると、`hnav_bar.jsp` は、対象となるタブへのリンクで以下の JSP タグを使用します。

```
<a href="<ptl:createPortalPageChangeURL
pageName='<%= portalPageName %>'/>"><%=portalPageName%></a>
```

ポータルの ptl:createPortalPageChangeURL JSP タグは、ページ変更イベントの Webflow URL を生成します。

ポートレットの動的表示と WebLogic Integration AI による在庫チェック

[Inventory] ページで、動的ポートレットの 1 つ、\productinventory\content.jsp を見てみましょう。このポートレットは、[Inventory] ページで、「1 Avitek Product Inventory」と表示されています。

このポートレットの Webflow ネームスペース ファイルは、以下のとおりです。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project\app
lication-sync\webapps\b2bPortal\product.wf
```

ここには、以下のコードが含まれます。

```
<presentation-origin node-name="product" node-type="jsp">
       <node-processor-info page-name="content.jsp"
       page-relative-path="/portlets/productinventory"/>
</presentation-origin>
```

content.jsp ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2bPortal
\portlets\productinventory\content.jsp
```

content.jsp ファイルには、以下の import 文があります。

```
<%@ page import="examples.e2e.common.Inventory" %>
```

content.jsp のスクリプトレットで、データベースの商品データを繰返し参照して、現在の在庫水準を取得します。その水準が定義されている最低水準より低い場合は、CSS ファイルから取得した CSS_INV_BELOW_MIN 設定を使用し、在庫に問題があることを赤色で表示します。

例 :

```
<%
       Iterator it = rState.getProducts().iterator();
       Inventory prod = null;
       String rowCssClass = null;
       String extraParams = null;
       int i = 0;
       for ( ; it.hasNext(); i++ ) {
       prod = (Inventory) it.next();
       if ( prod.isBelowMinimum() ) {
              rowCssClass = CSS_INV_BELOW_MIN;
```

```
        }
        else {
               rowCssClass = CSS_PRODUCT_ROW;
        }
        extraParams = PRODUCT_PARAM_EQUALS + prod.id();
        // skip the row divider the first time through
        if ( i != 0 ) {
%>
```

b2cPortal アプリケーションと、この b2bPortal アプリケーション用に、`InventoryProvider` という名前のサービス プロバイダ インタフェース (Service Provider Interface: SPI) を作成しました。これはステートレス セッション EJB として実装され、以下の 3 つのメソッドがあります。

`checkInventory()`

- WLI AI フレームワークを使って、コンフィグレーション済みの AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルを照会し、`E2E_PRODUCT_INV where sku = value` から数量を選択します。
- XML ヘルパーを呼び出して、AI からの応答を解析する。
- 応答に基づいて、利用可能な在庫数 (`int`) を返す。

`getProductInventory()`

- AI を使って、コンフィグレーション済みの AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルに照会するためのものです。

  例 :

  ```
  select sku, desc, minimum, maximum, quantity from
  E2E_PRODUCT_INV where parent_sku = NULL
  ```

- XML ヘルパーを呼び出して、AI からの応答を解析し、[List of Inventory] オブジェクトを作成する。

`getProductPartInventory()`

- AI を使って、コンフィグレーション済みの AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルに照会するためのものです。

  例 :

  ```
  select sku, desc, minimum, maximum, quantity from
  E2E_PRODUCT_INV where parent_sku <> NULL
  ```

- XML ヘルパーを呼び出して、AI からの応答を解析し、[List of Inventory] オブジェクトを作成する。

InventoryProvider SPI は、b2bPortal と b2cPortal に共通です。このソースファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\common\inventory\spi\*.java
```

b2bPortal の場合には、在庫チェックは以下によって実装されます。

- CheckInventoryAction
- GetProductInventoryAction
- GetProductPartInventoryAction

これらのプログラムの Java ソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\common\inventory\spi\*.java
```

これらのプログラムでは、EJB で提供されている getProductInventory() メソッドおよび getProductPartInventory() メソッドを使用します。このサンプル アプリケーションを簡略化するため、getProductInventory (List SKUs) メソッドを使用する際には、[List of SKU] が Pipeline コンポーネントの中にハードコード化されます。

次に、productinventory\content.jsp ポートレットで、返されたデータを表示します。

例：

```
<!-- model number -->
    <td width="187" class="<%= rowCssClass %>"><%= prod.id() %></td>

<!-- minimum # of units -->
    <td width="75" class="<%= rowCssClass %>"><%= prod.min() %></td>

<!-- maximum # of units -->
    <td width="75" class="<%= rowCssClass %>"><%= prod.max() %></td>

<!-- available # of units -->
    <td width="67" class="<%= rowCssClass %>"><%= prod.available() %></td>
```

次のステップ

ツアーを続けるには、**pix1000** カメラの [部品の在庫確認] ボタンをクリックしてください。

Product Parts Inventory ポートレット

Product Part Inventory ポートレットの全体を表示するには、スクロールしなければならない場合があります (ステップ 2)。このポートレットには、ステップ 1 で選択した Avitek 商品の構成部品の一覧が表示されます。部品ごとに在庫レベルが表示されます。在庫レベルが最低水準を下回った部品のデータは、赤色で表示されます。

Product Parts Inventory ポートレットの技術詳細

在庫処理のステップ 2 にあたる Product Part Inventory ポートレットの処理は、Product Inventory ポートレットの処理と似ています。商品の部品在庫が最低水準を下回った場合には、データが赤色で表示されます。現在の在庫水準を確認するため、データベース内の在庫テーブルに、Application Integration (AI) フレームワークを介して読み込み専用モードでアクセスしました。AI フレームワークは、WebLogic Integration によって提供されます。この場合、`InventoryProvider` SPI が提供する `getProductPartInventory()` メソッドが使用されました。

WebLogic Integration AI を介した商品の部品在庫のチェック

[Inventory] ページで、動的ポートレットの 1 つ、`\partinventory\content2.jsp` を見てみましょう。このポートレットは、[Inventory] ページで、「1 Avitek Product Inventory」と表示されています。

content2.jsp ファイルは、Product Inventory ポートレットで製品が選択されるとアクティブになります。\partinventory\content.jsp ポートレットは非アクティブのバージョンで、[Inventory] ページで製品がまだ選択されていないときに使用されます。

このポートレットの Webflow ネームスペース ファイルは、以下のとおりです。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project
\application-sync\webapps\b2bPortal\part.wf
```

ここには、以下のコードが含まれます。

```
<presentation-origin node-name="product" node-type="jsp">
       <node-processor-info page-name="content2.jsp"
       page-relative-path="/portlets/productinventory"/>
</presentation-origin>
```

content2.jsp ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2bPortal
\portlets\partinventory\content2.jsp
```

content2.jsp ファイルには、以下の import 文が含まれます。

```
<%@ page import="examples.e2e.common.Inventory" %>
```

content2.jsp のスクリプトレットで、データベース内の商品データおよび部品データを繰返し参照して、現在の在庫水準を取得します。その水準が定義されている最低水準より低い場合は、CSS ファイルから取得した CSS_INV_BELOW_MIN 設定を使用し、在庫に問題があることを赤色で表示します。

例：

```
<%
    String rowCssClass = null;;
    String extraParams = null;

      int i = 0;
      for ( ; parts.hasNext(); i++ ) {

          part = (Inventory) parts.next();

              if ( part.isBelowMinimum() ) {
                  rowCssClass = CSS_INV_BELOW_MIN;
              }
              else {

                    rowCssClass = CSS_PART_ROW;
              }

              extraParams = PART_PARAM_EQUALS + part.id();

%>
```

b2cPortal アプリケーションと、この b2bPortal アプリケーション用に、InventoryProvider という名前のサービス プロバイダ インタフェース (SPI) を作成しました。これはステートレス セッション EJB として実装され、以下の 3 つのメソッドがあります。

checkInventory()

- WLI AI フレームワークを使って、コンフィグレーション済みの AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルを照会します。

  例:

  ```
  select quantity from E2E_PRODUCT_INV where sku = value
  ```

- XML ヘルパーを呼び出して、AI からの応答を解析する。
- 応答に基づいて、利用可能な在庫数 (int) を返す。

getProductInventory()

- AI を使って、コンフィグレーション済みの AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルに照会するためのものです。

  例:

  ```
  select sku, desc, minimum, maximum, quantity from
  E2E_PRODUCT_INV where parent_sku = NULL
  ```

- XML ヘルパーを呼び出して、AI からの応答を解析し、[List of Inventory] オブジェクトを作成する。

getProductPartInventory()

- AI を使って、コンフィグレーション済みの AI サービスを介して呼び出しを行う。このサービスは、在庫テーブルに照会するためのものです。

  例:

  ```
  select sku, desc, minimum, maximum, quantity from
  E2E_PRODUCT_INV where parent_sku <> NULL
  ```

- XML ヘルパーを呼び出して、AI からの応答を解析し、[List of Inventory] オブジェクトを作成する。

`InventoryProvider` SPI は、b2bPortal と b2cPortal に共通です。このソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\example
s\e2e\common\inventory\spi\*.java
```

b2bPortal の場合には、在庫チェックは以下によって実装されます。

- `CheckInventoryAction`
- `GetProductInventoryAction`
- `GetProductPartInventoryAction`

これらのプログラムの Java ソース ファイルは、以下の場所にあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\common\inventory\spi\*.java
```

これらのプログラムでは、EJB で提供されている `getProductInventory()` メソッドおよび `getProductPartInventory()` メソッドを使用します。このサンプル アプリケーションを簡略化するため、`getProductInventory ( List SKUs )` メソッドを使用する際には、[List of SKU] が Pipeline コンポーネントの中にハードコード化されます。

次に、`partinventory\content2.jsp` ポートレットで、返されたデータを表示します。

例：

```
<td class="<%= rowCssClass %>" width="64"><%= part.id() %></td>
<td class="<%= rowCssClass %>" width="83"><%= part.description() %></td>
<td class="<%= rowCssClass %>" width="74"><%= part.min() %></td>
<td class="<%= rowCssClass %>" width="75"><%= part.max() %></td>
<td class="<%= rowCssClass %>" width="60"><%= part.available() %></td>
```

次のステップ

ツアーを続けるには、「2 Product Part Inventory」ポートレットで、赤色で表示されている部品の横の [見積の要求] ボタンをクリックしてください。

Query for Price and Availability (価格と在庫の照会) ポートレット

Query for Price and Availability (QPA) ポートレットを表示するには、下にスクロールしなければならない場合があります (ステップ 3)。このポートレットには、QPA を生成するためのフォームが用意されています。購入担当者が QPA 要求に必要なデータをすぐに取り出せるように、Product Inventory ポートレットと Product Part Inventory ポートレットは、Inventory Summary ポートレットに置き換えられています。このフォームに入力する必要のある情報については、必ず下記の「次のステップ」節を参照してください。

Query for Price and Availability (価格と在庫の照会) ポートレットの技術詳細

Query for Price and Availability (QPA) ポートレットには、QPA を生成するためのフォームが用意されています。QPA ビジネス プロセスは、このステップ 3 のポートレットで [QPA 要求の送信] ボタンをクリックすると開始されます。詳細については、3-22 ページの「Quotes for Price and Availability (価格と在庫の照会) ポートレットと QPA ビジネス プロセス」を参照してください。

この Query for Price and Availability ポートレットでは、購入担当者が QPA 要求に必要なデータをすぐに取り出せるように、Product Inventory ポートレットと Product Part Inventory ポートレットは、Inventory Summary ポートレットに置き換えられています。

サンプル アプリケーションに戻ったら、このツアー ガイドの「次のステップ」節の説明に必ず従ってください。ここでも説明しておきます。

Query for Price and Availability (価格と在庫の照会) ポートレットで、以下の操作を実行します。

- 注文を満たすのに必要な数量を入力してください。
- 次に、商品部品の単価を入力してください。たとえば、「50.00」のように入力します。
- また、商品を受け取りたい日付を入力してください。たとえば、今日から 1 週間後の日付を入力します。

次に、[QPA 要求の送信] ボタンをクリックします。

以下のエラーが表示される場合は、値を入力し直してから、[QPA 要求の送信] ボタンをクリックしてください。

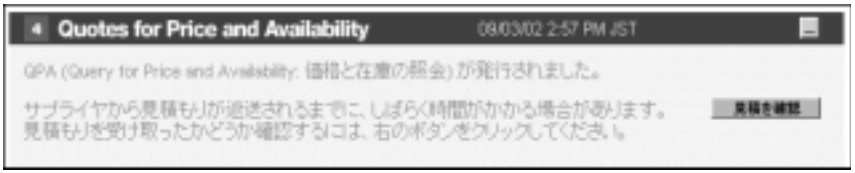
- 「数量フィールドには、1 以上の値を入力する必要があります。」
- 「単価フィールドには必ず値を入力してください。」
- 「今日より前の日付が入力されています。」

次のステップ

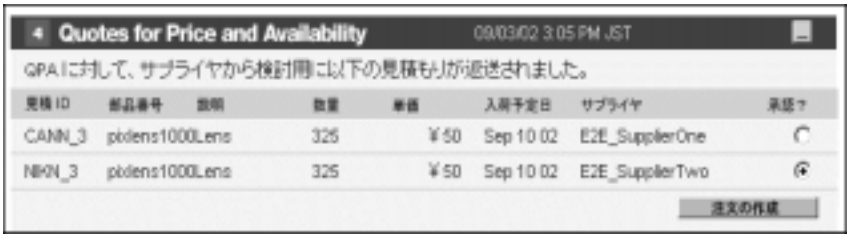
まだ注文を入力していない場合は、必要な数量を入力します。次に、製品部品の単価としてたとえば「50」と入力します。また、商品を受け取りたい日付を入力します。たとえば、今日から 1 週間後の日付を入力します。次に、[QPA 要求の送信] ボタンをクリックしてツアーを続けます。

Quotes for Price and Availability (価格と在庫の照会) ポートレットと QPA ビジネス プロセス

最初に、ステップ 4 のポートレットである Quotes for Price and Availabilty で、サプライヤから見積もりが届くまでにしばらく時間がかかる可能性があることを伝えるメッセージが表示されます。



[見積を確認] ボタンを何回かクリックすると、最後には、サプライヤからの見積もりがポートレットに表示されます。この処理は、WebLogic Portal のポータル フレームワークと、WebLogic Integration によって管理される企業間 (B2B) 対話との統合の例を示すものです。

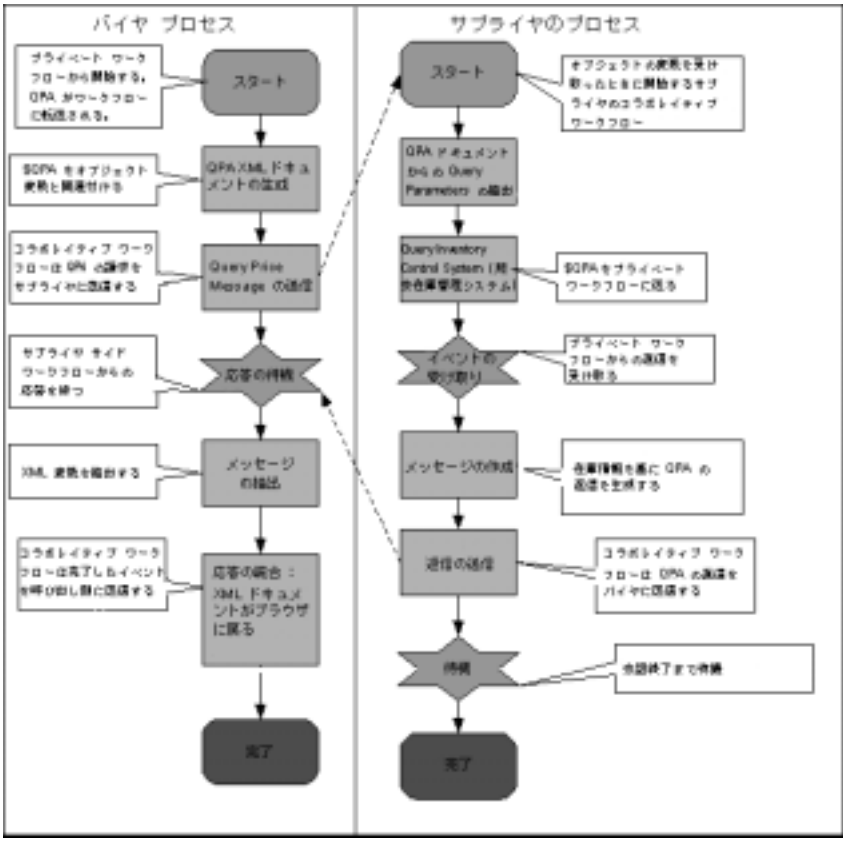


QPA ビジネス プロセスの技術詳細

この節では Query for Price and Availability (QPA) のビジネス プロセスに焦点をあてて説明します。このビジネス プロセスは、前のステップ 3 のポートレットで [QPA 要求の送信] ボタンをクリックすると開始されます。

pix1000 カメラの **pixlens1000** 部品が不足しているので、Avitek の購入担当者は、この部品の QPA メッセージを選択したサプライヤに送信します。次の図に、QPA ビジネス プロセスのイベント フローを示します。

QPA ビジネス プロセスのプロセス フロー



以下に、上の図で示したイベントを順に要約します。

1. バイヤが QPA を作成します。
2. QPA がサプライヤに送信されます。
3. サプライヤは QPA を処理し、応答を生成します。
4. バイヤは、サプライヤからの応答を収集し、結果をまとめます。

注意： 上の図では、高レベルから見た QPA ビジネス プロセスを示しています。両サイドのプロセスは、パブリック (コラボレイティブ) ワークフローとプライベート ワークフローによって実行されます。

以下の表に示したワークフロー テンプレートは、このサンプル QPA プロセスで使用されています。

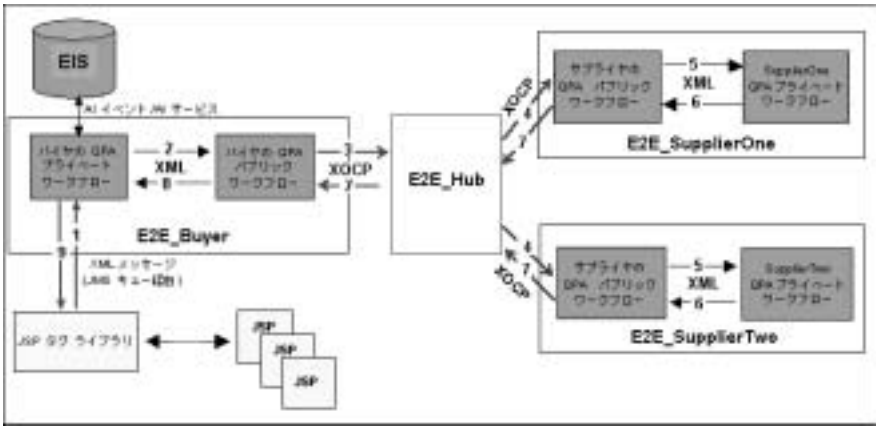
表 3-2 QPA プロセス ワークフローのテンプレート

ロール	パブリック / プライベート	ワークフロー名
バイヤ	プライベート	`E2E_BuyerQPAPrivate`
バイヤ	パブリック	`E2E_BuyerPOPublic`
サプライヤ	パブリック	`E2E_SupplierPOPublic` **注意：** シナリオに登場する両方のサプライヤは、同じパブリック ワークフローを使用します。
サプライヤ	プライベート	`E2E_SupplierOnePOPrivate`
サプライヤ	プライベート	`E2E_SupplierTwoPOPrivate`

WebLogic Integration は、取引相手とのビジネスに関する会話、および提携契約を管理し、バイヤとサプライヤ間のビジネス メッセージの交換を自動化します。ワークフローは、提携契約および会話で参照されます。

このサンプルでは、JSP と JSP タグ ライブラリを使用して QPA プロセスを開始し、QPA 要求と応答データを表示します。次の図に、QPA ビジネス トランザクションに関わる取引相手間のデータ フローを示します。

QPA ビジネス プロセスのデータ フロー



以下に、イベントを順に示し、取引相手間のデータ フローとワークフローについて要約します。

1. QPA フォームを含むポートレットは、QPA 要求を JMS キューに送信し、`E2E_BuyerQPAPrivate` ワークフローをトリガします。

2. `E2E_BuyerQPAPrivate` ワークフローは、`E2E_BuyerQPAPublic` ワークフローを呼び出し、QPA 要求 XML ドキュメントに渡します。次に、QPA 会話が開始します。

3. `E2E_Buyer` と `E2E_Hub` との間の提携契約に基づき、`E2E_BuyerQPAPublic` ワークフローは、QPA 要求 XML を XOCP メッセージにパックし、これを `E2E_Hub` に送信します。

   注意： `E2E_Hub` は、メッセージを受け取ると、提携契約のプロキシ サプライヤとして機能します。

4. `E2E_Hub` は、各サプライヤとの登録済み提携契約に基づいて、指定された取引相手、`E2E_SupplierOne` と `E2E_SupplierTwo` にメッセージを転送します。

   注意： この手順では、`E2E_Hub` はロールを変更し、サプライヤとの提携契約においてプロキシ バイヤとなります。

   各サプライヤのパブリック ワークフローは、XOCP メッセージを受け取ることによりトリガされます。このシナリオでは、`E2E_SupplierOne` と

E2E_SupplierTwo はパブリック ワークフロー (E2E_SupplierQPAPublic) を共有します。パブリック ワークフローは、メッセージから QPA 要求 XML ドキュメントを抽出します。

5. E2E_SupplierQPAPublic ワークフローは、各サプライヤに対するプライベート ワークフローを呼び出し、QPA 要求 XML ドキュメントをプライベート ワークフローに渡します。
6. 各サプライヤのプライベート ワークフローは、独自の QPA 応答 (XML ドキュメント) を作成し、これをパブリック ワークフローの戻り変数に添付します。
7. E2E_SupplierQPAPublic ワークフローは、QPA 応答 XML ドキュメントを抽出し、これを XOCP メッセージにパックしてバイヤに送信します。

   E2E_Hub が E2E_Buyer の送信プロキシとして機能していることを確認します。サプライヤが、(E2E_Hub と各サプライヤとの間の提携契約に基づいて) 応答メッセージを E2E_Hub に送信すると、E2E_Hub はプロキシ バイヤとして機能します。

   次に、E2E_Hub と E2E_Buyer との間の提携契約に基づき、E2E_Hub はロールをプロキシ サプライヤのロールに変更し、応答メッセージをバイヤ (E2E_Buyer) に転送します。
8. バイヤのパブリック ワークフロー (E2E_BuyerQPAPublic) は以下のとおりです。
   - 受信した XOCP メッセージから、QPA 応答 XML ドキュメントを抽出する。
   - 両方のサプライヤの応答ドキュメントを単一の XML ドキュメントに統合し、それを JMS キューを介してバイヤのプライベート ワークフロー (E2E_BuyerQPAPrivate) に送信します。
   - QPA 会話を終了し、サプライヤのパブリック ワークフロー (E2E_SupplierQPAPublic) に通知します。
9. バイヤのプライベート ワークフロー (E2E_BuyerQPAPrivate) は、統合された QPA 応答 XML ドキュメントを受け取り、それを XML ファイルに書き込みます。JSP は XML を解析し、統合された QPA 応答を Web ブラウザに表示します。

   この手順で、QPA ビジネス プロセスは終了です。

このプロセスの詳細については、WebLogic Integration のドキュメントを参照してください。

次のステップ

Quotes for Price and Availability ポートレットで、見積もりがサプライヤから返ってこない場合は、再度 [見積を確認] ボタンをクリックします。見積もりが返ってきたら、いずれかの見積もりを受理し、[注文の作成] ボタンをクリックしてツアーを続けます。

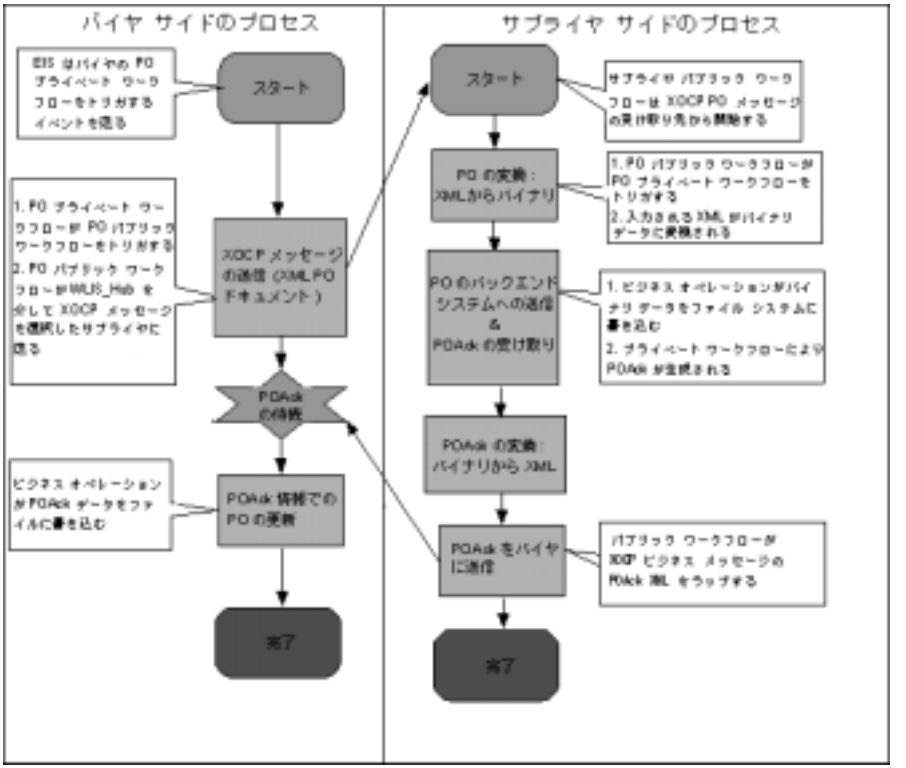
Purchase Order for Review ポートレットと PO ビジネス プロセス

Purchase Order for Review ポートレットを表示するには、下にスクロールしなければならない場合があります (ステップ 5)。このポートレットには、注文商品と購入商品の簡単な一覧が表示されます。

PO ビジネス プロセスの技術詳細

この節では、Purchase Order ビジネス プロセスに焦点をあてて説明します。このビジネス プロセスは、ユーザ (Avitek の購入担当者) が Purchase Order for Review ポートレットの [注文の送信] ボタンをクリックすると開始されます。このポートレットには、注文商品と購入商品の簡単な一覧が表示されます。

Purchase Order ビジネス プロセスのプロセス フロー



以下に、上の図で示したイベントを順に要約します。

1. バイヤは注文書 (PO) を作成します。
2. バイヤは選択したサプライヤにこの注文書 (PO) を送信します。

3. サプライヤは、PO を受け取り、XML ベースの PO をバイナリ データ ファイルに変換します。このサンプルでは、サプライヤの注文管理システムが、バイナリ ファイルを受け取って、注文を処理することを前提としています。
4. サプライヤは、別のバイナリ データ ファイルに基づいて、XML ベースの PO 確認書を生成します。
5. サプライヤはバイヤに PO 確認書を送信します。
6. バイヤは PO 確認に基づいて PO 情報を更新します。

注意： 上の図では、高レベルから見た PO ビジネス プロセスを表示しています。両サイドのプロセスは、パブリック ワークフローとプライベート ワークフローによって実行されます。

以下の表に示したワークフロー テンプレートは、このサンプル QPA プロセスで使用されています。

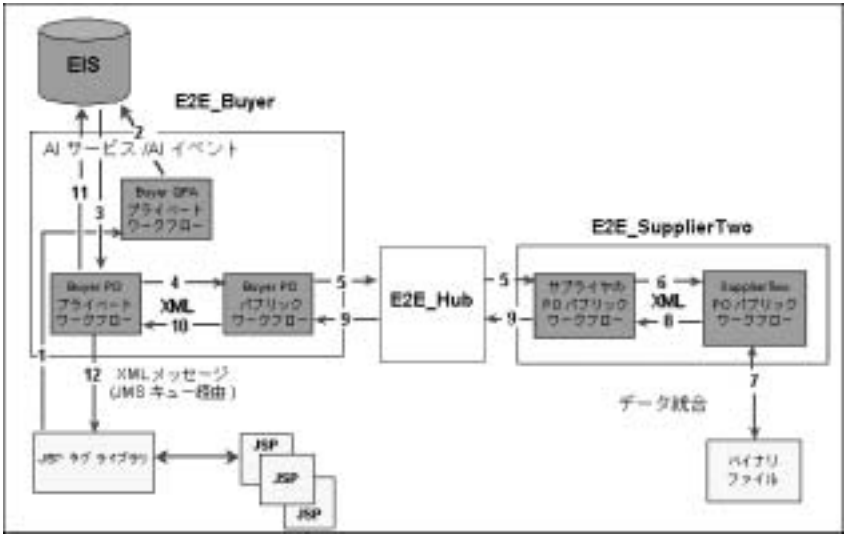
表 3-3 QPA プロセス ワークフローのテンプレート

ロール	パブリック / プライベート	ワークフロー名
バイヤ	プライベート	E2E_BuyerQPAPrivate
バイヤ	プライベート	E2E_BuyerPOPrivate
バイヤ	パブリック	E2E_BuyerPOPublic
サプライヤ	パブリック	E2E_SupplierPOPublic **注意：** シナリオに登場する両方のサプライヤは、同じパブリック ワークフローを使用します。
サプライヤ	プライベート	E2E_SupplierOnePOPrivate
サプライヤ	プライベート	E2E_SupplierTwoPOPrivate

このサンプルの PO 実装では、WebLogic Integration を使用してアプリケーション統合、データ統合、ビジネス プロセスの管理を行います。この節では、PO ワークフロー (バックエンド アプリケーションと異種データ形式との統合など) について説明します。

以下の図に、PO ビジネス プロセスに関わる取引相手間のデータ フローを示します。

PO ビジネス プロセスのデータ フロー



以下に、イベントを順に示し、取引相手間のデータ フロー、ワークフロー、およびバックエンドのシステムについて要約します。

1. PO ビジネス プロセスは、このシナリオのバイヤが Purchase Order for Review ポートレットの [注文の送信] ボタンをクリックすると開始されます。

   この PO は XML メッセージの形式で、BPM JMS キューに送信されます。

2. XML イベントにサブスクライブするバイヤのプライベート ワークフロー (`E2E_BuyerQPAPrivate`) が、イベント通知によって呼び出されます。

   `E2E_BuyerQPAPrivate` ワークフローは、`E2EAppView.sav` アプリケーション ビューの insertPO サービスを呼び出し、PO 情報をエンタープライズ情報システム (EIS) に挿入します。EIS は、このサンプルでは、RDBMS によってシミュレートされます。`E2EAppView.sav` アプリケーション ビュー、およびそのサービスとイベントは、サンプルをセット アップするときに、WebLogic Integration にコンフィグレーションおよびデプロイされます。

3. EIS は、PO 情報を含むイベントを WebLogic Integration プロセス エンジンに送信します。

   アプリケーション ビューのイベントは、PO プロセス (`E2E_BuyerPOPrivate`) のバイヤのプライベート ワークフローをトリガします。

4. `E2E_BuyerPOPrivate` ワークフローは、バイヤのパブリック ワークフロー (`E2E_BuyerPOPublic`) を開始します。

5. `E2E_BuyerPOPublic` は、サプライヤのパブリックワークフロー (`E2E_SupplierPOPublic`) に XOCP メッセージを送信します。この手順により PO 会話が開始されます。

   `E2E_Hub` は、`E2E_Buyer` との間の登録済み提携契約に基づいて、指定された取引相手、`E2E_Buyer` にメッセージを転送します。

   注意： この手順では、`E2E_Hub` はロールを変更し、バイヤとの提携契約においてプロキシ サプライヤとなります。

6. サプライヤ サイドでは、サプライヤのパブリック ワークフロー (`E2E_SupplierPOPublic`) のインスタンスは、サプライヤが XOCP メッセージを受け取ると起動します。

   `E2E_SupplierPOPublic` ワークフローは、選択したサプライヤのプライベート ワークフロー (`E2E_SupplierOnePOPrivate` または `E2E_SupplierTwoPOPrivate`) を開始します。

7. 選択したサプライヤのプライベート ワークフローは、以下のとおりです。

   - WebLogic Integration のデータ統合機能を使用して、受け取った XML PO データをバイナリ データに変換します。
   - バイナリ フォーマットで PO 確認書を生成します。
   - PO 確認書をバイナリ フォーマットから XML に変換します。

8. サプライヤのプライベート ワークフローは PO 確認書の XML データを `E2E_SupplierPOPublic` ワークフローに返します。

9. `E2E_SupplierPOPublic` ワークフローは、PO 確認情報を XOCP メッセージにラップし、それをバイヤのパブリック ワークフロー (`E2E_BuyerPOPublic`) に送ります。

10. E2E_BuyerPOPublic ワークフローは XOCP メッセージを受け取り、XML コンテンツを抽出して、XML イベントをバイヤのプライベート ワークフロー (E2E_BuyerPOPrivate) に送ります。

    この手順で、PO 会話は終了です。

11. E2E_BuyerPOPrivate ワークフローは、WebLogic Integration が提供するアプリケーション統合フレームワークを使用して、EIS 内の PO 情報を PO 確認書のデータに基づき更新します。また、ワークフローは、PO 確認情報を POAcknowledgement.xml ファイルに書き込みます。

12. 次に、WebLogic Portal は、POAcknowledgement.xml ファイルを読み込み、そのコンテンツを Order History ポートレットなどの適切なポートレットに表示します。

    この手順で、PO ビジネス プロセスは終了です。

このプロセスの詳細については、WebLogic Integration のドキュメントを参照してください。

次のステップ

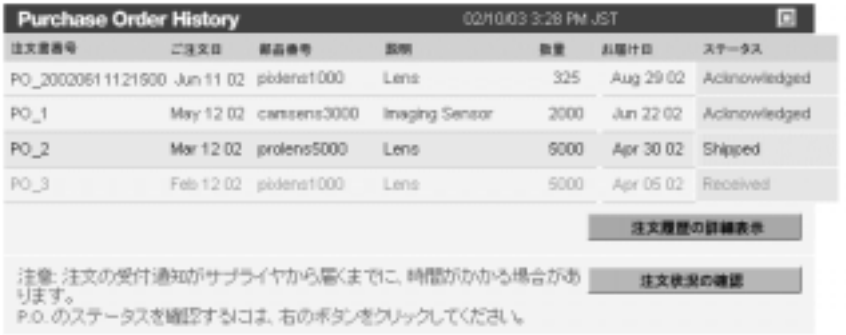
ツアーを続けるには、[注文の送信] ボタンをクリックして、注文をサプライヤに送信してください。

Purchase Order History ポートレット

[Order History] ページの最初の画面には、Purchase Order History ポートレットの要約ビューが表示されます。サンプル アプリケーションのデータには、既存の注文がいくつか含まれています。[注文履歴の詳細表示] ボタンをクリックすると、同じポートレットのさらに詳細なビューが表示されます。

[注文状況の確認] ボタンをクリックし、サプライヤから返された注文の受付確認をチェックできます。最初、画面は次のように表示されます。

[注文状況の確認] を数分後にクリックすると、画面が更新され、送信した注文とその配送日の 2002 年 8 月 29 日が以下のように表示されます。



Purchase Order History ポートレットの技術詳細

[Order History] ページの Purchase Order History ポートレットでは、各注文 (P.O.) に対する Acknowledged (受付確認済)、Shipped (配送済)、または Received (受領済) の各ステータスと関連情報が表示されます。送信したばかりの P.O. は、サプライヤから受付確認が返され、さらにユーザが [注文状況の確認] ボタンをクリックするまでは、このポートレットに表示されません。

注文書を送信したばかりで、Purchase Order History ポートレットにまだ表示されていない (ステータスが保留中の) 場合には、[注文状況の確認] ボタンをクリックしてください。

プロセスを高レベルで見ると、サプライヤから返された受付確認を受け取るのは、Purchase Order ビジネス プロセスでの最後から 2 番目のステップとなります。最後のステップは、バイヤ (Avitek) が P.O. 受付確認に基づいて P.O. 情報を更新するステップです。この Purchase Order ビジネス プロセスは、Purchase Order Review ポートレットの「技術詳細」の節で図を使って説明されています。

また、[Order History] ページでは、[注文履歴の詳細表示] ボタンをクリックして、注文とそのステータスの詳細を表示できます。ポートレットの詳細表示で、[注文状況の確認] ボタンをクリックすると、ステータスを確認できます。それでもまだ注文が保留中の場合は、ポートレットにより次のメッセージが返されます。「この注文の受付通知をまだ受け取っていません。」

詳細表示された Purchase Order History ポートレットで [注文履歴の概要に戻る] ボタンをクリックすると、ポートレットは一覧表示に戻ります。

企業間 (B2B) ポータル ツアーの完了

これで企業間 (B2B) ツアーは終了です。オンライン ツアーの場合、[ログアウト] ボタンをクリックすると、サンプルの [はじめに] ページに戻ることができます。ログアウトすると、このセッションで行った注文はまったくデータベースに保存されないことに注意してください。つまり、Jason Tang として再びログインしたときには、在庫水準は元の値にリセットされます。

4 Web サービス ツアー

サンプル アプリケーションには、WebLogic Workshop で作成した 2 つの Web サービスが含まれています。Avitek Digital Imaging の B2C (企業消費者間取引) サンプル ポータル (サンプルの「はじめに」ページのセクション 02) のツアーを終了すると、Web サービスから返された結果が表示されます。1 つ目の Web サービスは、サイトの商品評価ポートレットに表示される製品レーティングのルックアップを実行します。2 つ目の Web サービスは、クレジット カードの支払認可を実行します。

WebLogic Platform サンプルのこのツアーでは、Web サービスを作成する開発サイクルにおいて、BEA WebLogic Workshop がどのように機能しているかを説明します。さらに、Portlet Wizard を使用して、製品レーティング Web サービスの WSDL を参照し、ポートレット用のコードを生成する方法についても説明します。

注意： ここで示す情報は、サンプル アプリケーションの [はじめに] ページの [Web サービス技術ツアーの開始] ボタンをクリックして表示することもできます。

この章には次の節があります。

- WebLogic Workshop の起動
- WebLogic Workshop を使用した Web サービスの定義 — 概要
- Product Evaluator Web サービスの定義
- 支払認可 Web サービスの定義

WebLogic Workshop の起動

最初の手順は WebLogic Workshop のビジュアルな開発環境を起動することです。この節では、サポート対象プラットフォーム Microsoft Windows および UNIX での起動方法を説明します。

サポート対象プラットフォームについては、BEA e-docs Web サイトの「サポート対象プラットフォーム」ページを参照してください。

Microsoft Windows の [スタート] メニューから WebLogic Workshop を起動するには、次の順に選択します。

[プログラム | BEA WebLogic Platform 7.0 | WebLogic Workshop | WebLogic Workshop]

UNIX で WebLogic Workshop を起動するには、次の手順を実行します。

1. ファイル システム ブラウザまたはシェルを開きます。
2. `BEA_HOME` ディレクトリ構造内の Workshop.sh ファイルを検索します。

   例 :

   ```
   $HOME/bea/weblogic700/workshop/Workshop.sh
   ```

3. そのディレクトリに移動し、次のコマンドを入力します。

   ```
   sh Workshop.sh
   ```

4. または、ファイル システム ブラウザから Workshop.sh をダブルクリックします。

プロジェクトの場所について

このツアーでは、WebLogic Platform サンプル アプリケーションの WebLogic Workshop で定義された 2 つの Web サービスについて紹介します。この 2 つの Web サービスで構成されるプロジェクト ファイルは、BEA_HOME ディレクトリの下の次の場所にインストールされます。

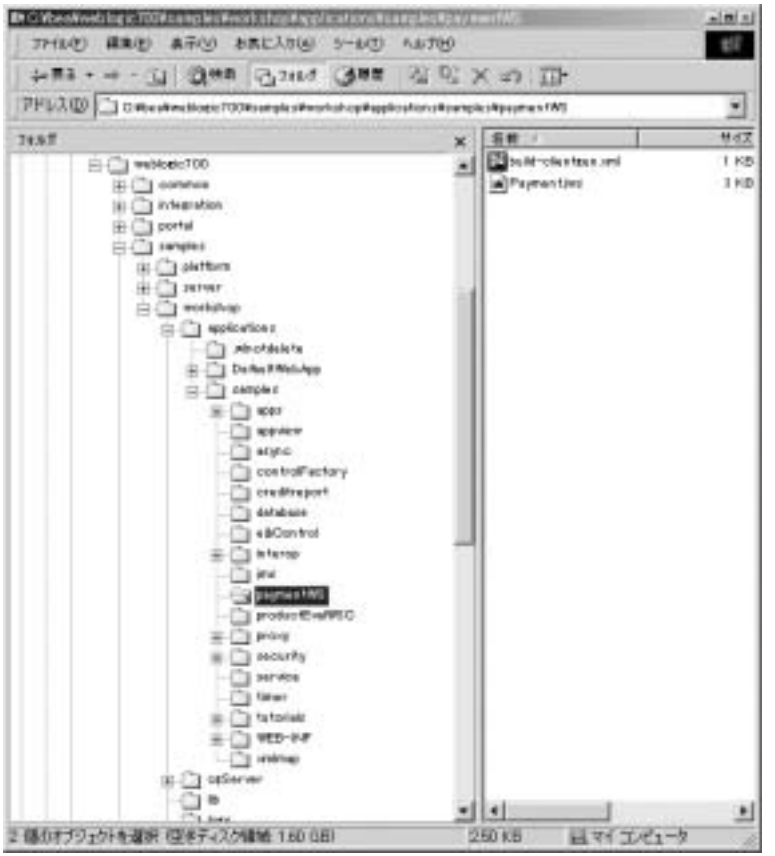
```
weblogic700\samples\platform\e2eDomain\beaApps\e2eWebServicesApp\workshop\productEvalWSC
```

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eWebServicesAp
p\workshop\paymentWS
```

これらの Web サービスを表示し、テストする最も簡単な方法は、太字で示されている上記の 2 つのフォルダを次のフォルダにコピーすることです。

```
weblogic700\samples\workshop\applications\samples
```

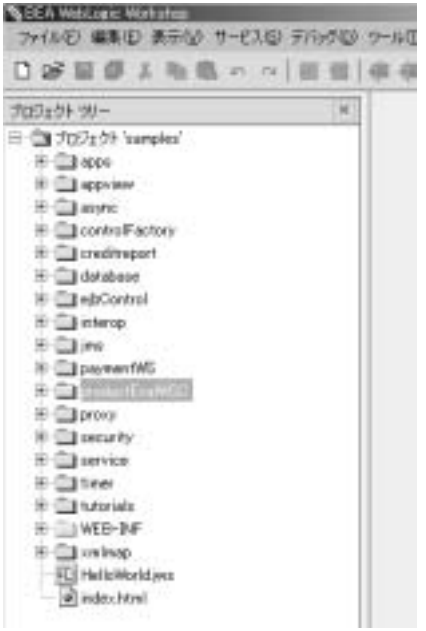
コピーが終了すると、このフォルダの階層構造は次のようになります。



次に、WebLogic Workshop で、[プロジェクトを開く] メニューのオプションを使用し、各 Web サービスの JWS ファイルに移動します。インストール後に初めて WebLogic Workshop を起動する場合は、新しいプロジェクトを作成するか、または新しいプロジェクトを開くよう求められます。その場合には、次の手順を実行します。

1. トップレベル メニューから [ファイル | プロジェクトを開く] をクリックします。
2. 次のように、**samples** プロジェクトを選択します。
3. [開く] ボタンをクリックします。

`productEvalWSC` および `paymentWS` の 2 つのフォルダを WebLogic Workshop の samples プロジェクト フォルダにコピーしているため、これらのフォルダはプロジェクト ツリーで次のように表示されます。



注意： WebLogic Workshop の各プロジェクトは J2EE Web アプリケーションに対応しており、これは、samples フォルダに `WEB-INF` フォルダがあることからもわかります。

たとえば、プロジェクト ツリーの `productEvalWSC` フォルダを展開し、`EvalProduct.jws` ファイルをダブルクリックすることができます。WebLogic Workshop のビジュアルな開発環境では、次のような画面が表示されます。

WebLogic Server インスタンスが稼動していないことに注意してください。サーバの起動オプションについては、次の節を参照してください。

サーバの起動オプション

Web サービスの定義を単に表示する場合は、WebLogic Workshop のビジュアルな開発環境で WebLogic Server インスタンスを起動する必要はありません。ただし、このツアーで Web サービスを実行し、テストする場合には、次の手順を実

行します。e2eDomain のサーバ インスタンスがすでに稼動していますが、このツアーでは別のサーバ インスタンスを起動して、次のディレクトリにある Web サービスを実行し、テストします。

```
weblogic700\samples\workshop\applications\samples
```

ここでは、4-2 ページの「プロジェクトの場所について」での説明に従って、paymentWS および productEvalWSC の 2 つのフォルダを Platform の samples 領域から Workshop の samples 領域にコピーしていることを前提としています。

Web サービスを実行し、テストするには、次の手順を実行します。

1. WebLogic Workshop のビジュアルな開発環境のトップレベル メニューから [ツール | プリファレンス] の順にクリックします。
2. [パス] タブをクリックします。
3. 起動するサーバ インスタンスのデフォルトの情報は、正しいものである必要があります。ドメインが「workshop」であることを確認します。
4. ブラウザのパスが正しいことを確認します。正しくない場合は、使用しているブラウザの実行ファイルの正しい場所を参照します。

   次の画面には、サンプルの値が表示されています。

5. [プリファレンス] 画面で [OK] をクリックします。
6. トップレベル メニューから [ツール | WebLogic Server の起動] をクリックします。

サーバが起動すると、画面の下に [サーバが実行中です。] ステータスが表示されます。

注意： WebLogic Workshop で作成し、cgDomain 以外のドメインにすでにデプロイしている Web サービスをテストすることができます。たとえば、e2eDomain には、次のディレクトリにデプロイされた paymentWS と productEvalWSC の 2 つの Web サービスがインストールされています。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eWebServicesApp\workshop\*
```

e2eDomain のサーバ インスタンスが稼動しているため、WebLogic Workshop のビジュアルな開発環境を開かなくても、ブラウザで次の場所を指定して、これらの Web サービスをテストできます。

```
http://localhost:7501/workshop/productEvalWSC/EvalProduct.jws
http://localhost:7501/workshop/paymentWS/Payment.jws
```

WebLogic Workshop からこのテスト ページを呼び出すかどうかにかかわらず、WebLogic Workshop には、Web サービスのテストに非常に有用なこれらのページが用意されています。

次の節では、WebLogic Workshop の機能と Web サービス定義のプロセスの概要を説明します。

WebLogic Workshop を使用した Web サービスの定義 — 概要

WebLogic Platform のサンプル アプリケーション用に作成された Product Evaluator および Payment の 2 つの Web サービスについて説明する前に、WebLogic Workshop を使用して Web サービスを定義する基本手順を説明します。また、この興味深い新製品、WebLogic Workshop についても簡単に紹介します。また、詳細情報を参照するためのオンライン ドキュメントへのリンクも載せています。

WebLogic Workshop は、新しいビジュアルな開発環境で、アプリケーション開発者も J2EE の専門家もエンタープライズクラスの Web サービスの構築とデプロイを簡単に行えます。製品は次の 2 つの主要コンポーネントで構成されています。

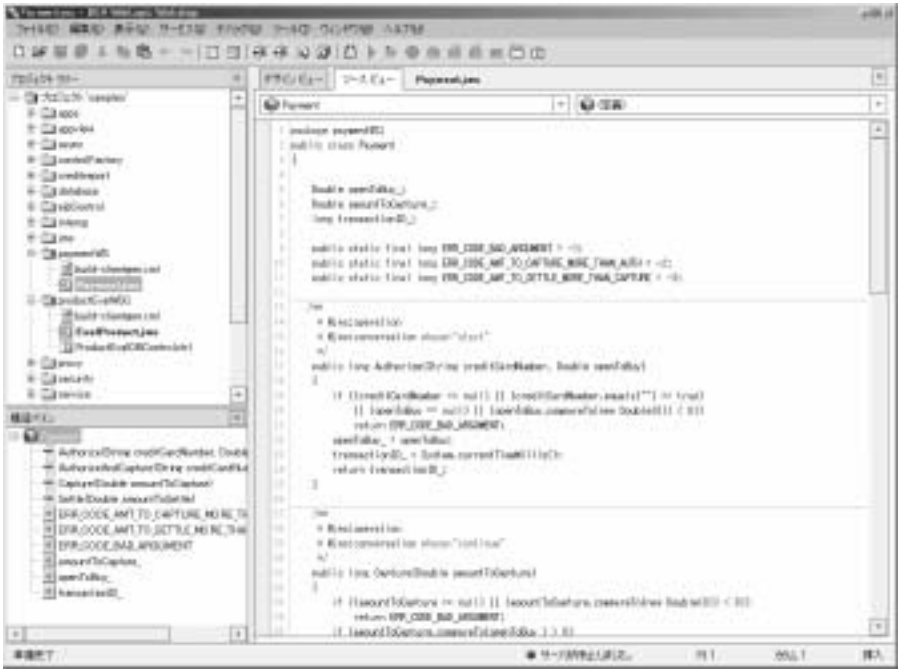
- 開発者が Java コードを記述して Web サービスを実装できる設計時ツール。
- Web サービス インフラストラクチャ、アプリケーションのテスト、デバッグ、デプロイ環境を提供する実行時フレームワーク。

WebLogic Workshop のビジュアルな開発環境

WebLogic Workshop のビジュアルな開発環境は、Web サービス アプリケーション開発に最適な環境です。プロジェクト管理、構文の強調表示、コード補完、および統合デバッグなどの標準機能もすべて含まれています。また、WebLogic Workshop は、Web サービスへのビジュアルで便利なアクセス方法を提供しています。例として、[デザイン ビュー] 画面を次に示します。

開発中の Payment Web サービスが画面中央に表示されます。この [デザイン ビュー] 画面により、ユーザは、新しいメソッドの作成、コントロールのプロパティ設定、および Web サービスの全体構造や外部との関係の指定をグラフィカルに行うことができます。この製品が目標とするのは、開発者が通常の Java プログラミングの手続きを意識することなく、ビジネス ロジック (一連の各メソッドに応答して実行されるコード) の記述に専念できるようにすることです。WebLogic Workshop は双方向編集をサポートしているため、グラフィカルな環境での変更はすぐにコードに反映され、またコードでの変更はグラフィカルな環境にすぐに反映されます。

WebLogic Workshop の [ソース ビュー] 画面には、標準の Java IDE 編集機能がすべて備わっており、Java Web サービス (JWS) の注釈が表示されます。たとえば、次のように表示されます。



WebLogic Workshop のコントロール

WebLogic Workshop のコントロールは主要な新機能で、エンタープライズ リソースおよび J2EE API の使用を簡素化します。WebLogic Workshop によって、複雑な API の使用が簡素化され、外部リソースにアクセスするためのオブジェクト指向プログラミングの量も軽減されます。たとえば、WebLogic Workshop には、JDBC API を簡素化するデータベース コントロールがあります。データベース管理者は、単純な宣言形式で、Java メソッドと SQL 文をリンクする再利用可能なデータベース コントロールを作成できます。次に、開発者は、これらのコンポーネントを使用して、単純な関数呼び出しでデータベース リソースにアクセスできます。

たとえば、productEvalWSC Web サービスでは、WebLogic Workshop を使用して、ProductEvalDBControl というデータベース コントロールを定義します。データベース コントロールの 4 つのメソッドのうちの 1 つは、次のような findRatingData です。

```
/**
 * @jws:sql statement="SELECT RELIABILITY_RATING,
       VALUE_RATING, OVERALL_RATING,
       COMMENTS FROM E2E_PRODUCT_EVAL
       WHERE SKU = {productId}"
 */

public Evaluation findRatingData(String productId)
        throws SQLException, ControlException;
```

SQL 文と Java メソッドが、WebLogic Workshop の特定の Javadoc 注釈によってどのように関連付けられているかに注目してください。このようにメソッドがコントロール上に置かれると、コントロールのユーザは単に関数を呼び出すだけで SQL コマンドを実行できます。

Java Web サービス ファイルとコントロール ファイル

設計時ツールと実行時フレームワークの接点は、Java Web Service (JWS) ファイルと関連する制御 (CTRL) ファイルです。

JWS ファイルは、追加機能を表すために (Javadoc 構文を使用して) 注釈が付けられた標準の Java ファイルです。注釈を使用すると、Web サービスとそのプロパティをグラフィカルに表示できます。また、フレームワークは注釈を使用して、EJB および J2EE のコードを生成し、Web サービスを実行します。コード生成をツールからフレームワークに移すことで、開発者は自らが記述していないコードの管理と保守を行わずに済みます。

拡張子 CTRL の付いたファイルは、WebLogic Workshop のコントロール ファイルです。通常、これらのファイルには、データベースや別の Web サービスなどのリソースへのアクセスを簡単にするメソッド定義の集合が格納されます。CTRL ファイルは次のようなコントロールを表すことができます。

- サービス コントロール。使用しているサービスから別の Web サービスと通信を行うために使用される。

- データベース コントロール。使用している Web サービスからデータベースにアクセスするために使用される。
- EJB コントロール。使用している Web サービスから既存の Enterprise Java Bean (EJB) にアクセスするために使用される。
- JMS コントロール。使用している Web サービスから既存の Java Message Service (JMS) キューまたはトピックにアクセスするために使用される。

WebLogic Workshop フレームワーク

Web サービスのビジネス ロジックをすべて含む JWS ファイルが作成されると、WebLogic Workshop フレームワークにより、JWS ファイルを実行するために必要な標準の EJB コードが生成されます。WebLogic Workshop フレームワークは、注釈を使用して、エンタープライズクラスの Web サービスを構築するため特別に設計された機能をエクスポーズします。WebLogic Workshop フレームワークにより次のことを実行できます。

- **会話のメタファを使用した非同期通信の管理**

  WebLogic Workshop フレームワークは、非同期メッセージの相関関係の管理、および会話でやりとりされるメッセージの状態管理を自動的に行います。つまり、ユーザは、メソッドに開始、終了、会話の継続といういずれかのマークを付けるだけで、残りの作業はフレームワークが実行します。会話を識別するための一意の ID が自動的に生成され、Web サービスで定義されているあらゆる状態 (クラスのメンバー変数) がエンティティ Java beans により一貫して管理されます。

- **XML マップと XML スクリプトの疎結合**

  WebLogic Workshop フレームワークは、単純な宣言的 XML マップを使用して、内部 Java コード、および Web サービス間で交換される XML メッセージをマップします。ユーザは使用するメソッドの構造を示し、XML フィールドを Java 変数に関連付けます。

- **JMS キューによる可用性の実現**

  高い負荷での可用性を確保するため、Web サービスはメッセージ バッファを利用します。ユーザはメソッドにバッファが必要であるとマークを付けます。WebLogic Workshop フレームワークによりキューが作成され、Web

サービスに書き込まれます。これにより、高度な J2EE 機能に 1 回のクリックでアクセスできます。

- **コントロール アーキテクチャのサポート**

  WebLogic Workshop フレームワークは、JWS が使用するあらゆるコントロールの実装をインスタンス化します。また、WebLogic Workshop フレームワークは、単純なネーミング規則を使用して、あらゆる非同期コールバックを関連付けます。

Web サービスの開発プロセスの概要

ツアーのこの部分では、Web サービスを構築するための基本手順を簡単に説明します。それぞれの基本手順の詳細については、WebLogic Workshop のオンライン ドキュメントを参照してください。チュートリアルから開始するのが最適です。チュートリアルは、トップレベル メニューから [ヘルプ | チュートリアル] をクリックすると、ビジュアルな開発環境で使用できます。または、BEA e-docs Web サイトの WebLogic Workshop のページでチュートリアルを開始して参照することもできます。「チュートリアル : 初めての Web サービス」というトピックを検索してください。

1. **WebLogic Workshop を起動し、メソッド、イベント、コントロールの定義を開始してテストを実行します。**
   - WebLogic Workshop を起動します。
   - 最初の Web サーバで、WebLogic Workshop に付属している Samples プロジェクトを開きます。
   - このプロジェクトで、Web サービスのサブディレクトリを作成します。ダイアログ ボックス ウィンドウには、Web サービスを作成中であることが表示されます。その Web サービスに名前を付けます。
   - WebLogic Server インスタンスを起動します。
   - グラフィカルな [デザイン ビュー] で、1 つまたは複数のメソッドとイベントを Web サービスに追加します。他のサービス、データベース、および EJB (Enterprise Java Beans) などのリソースを表すコントロールを追加することもできます。また、設計項目にプロパティを設定して、基本となるサーバの高い性能を十分に引き出すこともできます。

- [ソース ビュー] タブを選択し、[デザイン ビュー] での作業結果を表示します。ビジネス ロジックを追加できます。[ソース ビュー] でソース コードにメンバー変数を追加すると、その新しい変数が [デザイン ビュー] に表示されます。またその反対の動作も行われます。
- [デザイン ビュー] で、1 つまたは複数のパラメータを追加し、メソッドの値を返します。値を返すためのコードを追加します。
- Web サービスを構築し、テストし、デプロイします。WebLogic Workshop の [Test View] (Web サービスのメソッドを呼び出せるブラウザベース ツール) を使用します (サンプル画面がこのツアーの `paymentWS` および `productEvalWSC` の Web サービスのモジュールに表示されます)。

2. アプリケーションに応じて、非同期通信のサポートを追加します。

非同期通信でのクライアントと Web サービス間の通信では、Web サービスが応答を返している間は、通信が強制的に停止されることはありません。株価が特定の値になった場合に通知するなどのコールバックを追加することもできます。

3. データベース コントロールを追加します。

この手順では、データベース コントロールを Web サービスに追加します。データベース コントロールにより、Web サービスからデータベースにアクセスできます。コントロールは、Web サービスと他のデータ リソース (データベース、他の Web サービス、Java Message Services など) 間のインタフェースとして機能します。

4. アプリケーションに応じて、サービス コントロールを追加します。

Web サービスにサービス コントロールを追加して、別の外部 Web サービスを呼び出すことができます。

5. 必要に応じて、JMS および EJB などのコントロールを追加します。

`JMSControl` を使用することで、Java Message Service (JMS) を介して利用できるコンポーネントにアクセスできます。このコントロールにより、XML などさまざまな種類のメッセージを送受信できます。また、`EJBControl` を使用して、Enterprise Java Beans (EJB) にアクセスすることもできます。この `EJBControl` は、Web サービス設計に EJB インタフェースを表す単一のコンポーネントを提供して、EJB の使用を簡素化します。

6. **必要に応じて、マッピング用のスクリプト ファイルを追加します。**

   Web サービスと他のコンポーネント間で交換する特定の型の XML メッセージを処理、またはコントロールする必要がある場合は、XML マップを使用できます。XML マップにより、WebLogic Server の XML から Java への変換をカスタマイズすることができ、またその反対の操作も可能です。

7. **取消および例外を処理するサポートを追加します。**

   `TimerControl` を使用すると、Web サービスにタイマー機能を追加できます。これにより、処理の実行を指定した時間に限定したり、一定の間隔で処理を実行させることができます。また、`onException` コールバックを使用して、Web サービスの処理により生じた例外を処理することもできます。このコールバックによって要求されると、コードは、必要なクリーンアップを実行したり、クライントにメッセージを送信したりします。

8. **必要に応じて、Web サービスを変更し、ポーリング インタフェースをサポートします。**

   Web サービスを構築すると、WebLogic Workshop により、クライアント ソフトウェアで使用できるクラスが生成されます。これらのプロキシ クラスを使用して、クライアントはサービスのメソッドを呼び出すことができます。プロキシ クラスは [Test View] からダウンロードできます。

9. **セキュリティを設定し、Webapp の一部として Web サービスをデプロイします。**

   HTTP ではなく HTTPS プロトコルを介して Web サービスをエクスポーズすることで、Web サービスのセキュリティを高めることができます。これを実行するには、Web サービスのプロジェクトに関連付けられたコンフィグレーション ファイルを編集します。次に、`JwsCompile` コマンドを使用して、Web サービスをパッケージ化し、プロダクション サーバにデプロイできます。

これで基本手順の説明は終わりです。次の節では、WebLogic Platform サンプル アプリケーションの WebLogic Workshop で作成した `productEvalWSC` Web サービスについて説明します。

Product Evaluator Web サービスの定義

架空の会社、Avitek Digital Imaging の B2C (企業消費者間取引) ポータル Web サイトにの製品カタログを参照しているユーザ向けに、次のような商品評価ポートレットを用意しました。

このポートレットは、WebLogic Workshop で作成した Web サービスを使用して、選択したカタログ アイテムの製品レーティング情報を返します。ここでは、`productEvalWSC` という Web サービスについて説明します。また、Portlet Wizard で商品評価ポートレットを作成するための手順についても説明します。

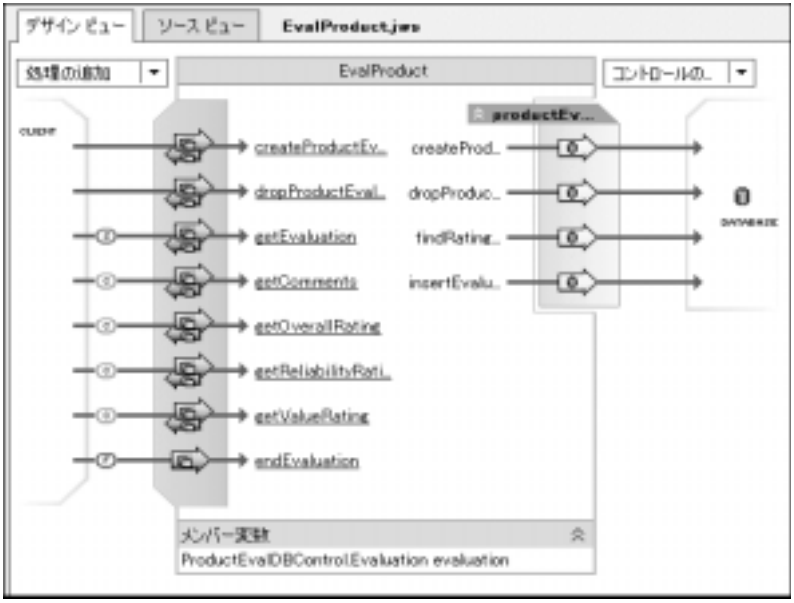
既存の productEvalWSC Web サービスの表示

WebLogic Workshop のプロジェクト ツリー ウィンドウで、次のディレクトリにコピーした `productEvalWSC` フォルダをダブルクリックします。

```
weblogic700\samples\workshop\applications\samples\productEvalWSC
```

この手順をツアーの前の部分で省略している場合は、4-2 ページの「プロジェクトの場所について」を参照してください。その後、このページに戻ってください。

展開した `productEvalWSC` プロジェクト フォルダ内の `EvalProduct.jws` という Java Web サービス ファイルをダブルクリックします。次の画面は、この Web サービスの [デザイン ビュー] 画面です。



WebLogic Workshop を使用して、`ProductEvalDBControl` とそのメソッドを定義しました。次のようなメソッドがあります。

- `createProductEvalTable`
- `dropProductEvalTable`
- `findRatingData`
- `insertEvaluationRecord`

次に、WebLogic Workshop を使用して、JWS の次のメソッドを定義しました。

- `getEvaluation`
- `getComments`
- `getOverallRating`
- `getReliabilityRating`
- `getValueRating`
- `endEvaluation`

[デザイン ビュー] 画面を見ると、小さな (s)、(c)、(F) の記号が JWS メソッドに付いていることがわかります。これらの記号は、どのメソッドが Web サービスの会話を**開始 (s)** (`getEvaluation`) し、どのメソッドが会話を続行 (c) (`getComments` から `getValueRating` まで) し、どのメソッドが会話を**終了 (F)** (`endEvaluation`) するかを示します。

注意： 本番環境の Web サービスでは、製品の評価コメント、総合的なレーティング、信頼性のレーティング、および価格のレーティングのそれぞれを返す個別の呼び出しは作成しません。個別の呼び出しではなく、Web サービスで複合オブジェクトを作成し、1 回の呼び出しですべてのレーティングを受け取ります。複合オブジェクトを使用し、個別に呼び出しを行わないことで、パフォーマンスが向上します。

WebLogic Workshop のビジュアルな開発環境でデータベース コントロールおよびそのメソッドを定義することは簡単です。このツアーでは、コントロールとメソッドを作成する手順は説明しません。これらの手順は、WebLogic Workshop のドキュメントに記載されています。このドキュメントは、ビジュアルな開発環境で参照できます。また、BEA e-docs Web サイトの WebLogic Workshop のページでも参照できます。

ここでは、`ProductEvalDBControl` データベース コントロールとそのメソッドの目的について簡単に説明します。開発の過程で、本稼動に移行するエンタープライズ アプリケーション用にデータベース スキーマを完了する前に、必要に応じてメタデータを作成したり削除するデータベース コントロール メソッドをテスト用に定義しておくことが開発に役立ちます。開発プロセスを簡潔にするため、まず `createProductEvaluatorTable` メソッドおよび `dropProductEvaluatorTable` メソッドを定義することから始めました。WebLogic Workshop では、各メソッドの矢印をダブルクリックして、その SQL 文を表示できます。たとえば、`createProductEvaluatorTable` の定義は次のようになります。

次に、`insertEvaluationRecord` メソッドおよび `findRatingData` メソッド (WebLogic Workshop のビジュアルな開発環境の SQL 文を参照) を定義しました。

これらの利用可能なデータベース コントロール メソッドを使用して、データベース コントロール メソッドを使用する JWS メソッドを定義しました。たとえば、`getEvaluation` メソッドは、Web サービスの会話を開始し、データベース コントロールから `findRatingData` メソッドを使用します。WebLogic Workshop では、[ソース ビュー] タブを選択するか、または `getEvaluation` メソッドの下線の付いたリンクをクリックしてそのメソッドを開始する行の近くに [ソースビュー] を表示して、JWS のコードを表示できます。たとえば、次のように表示されます。

```
* @jws:operation
* @jws:conversation phase="start"
*/

public String getEvaluation(String productId)
{
   try
   {
     evaluation = productEvalDB.findRatingData(productId);
        }
```

```
            catch (java.sql.SQLException e)
            {
                // will be interpreted as productId not found
            }
            if( evaluation != null )
            {
                return "SUCCESS";
            }
            else

        {
                return("NOT FOUND");
        }
    }
```

EvalProduct.jws、つまり productEvalWSC の JWS に定義したメソッドのコードを WebLogic Workshop で参照することができます。ビジュアルな開発環境におけるメソッド定義の詳細については、WebLogic Workshop のドキュメントを参照してください。ドキュメントはビジュアルな開発環境で参照できます。また、BEA e-docs Web サイトの WebLogic Workshop のページでも参照できます。

productEvalWSC Web サービスの構築とテスト

cgDomain の WebLogic Server インスタンスが稼動していない場合は、これを起動します。この手順をツアーの前の部分で省略している場合は、4-6 ページの「サーバの起動オプション」を参照してください。その後、この節に戻ってください。

サーバが稼動している場合は、WebLogic Workshop のビジュアルな開発環境の下部に次のようなステータス アイコンが表示されます。

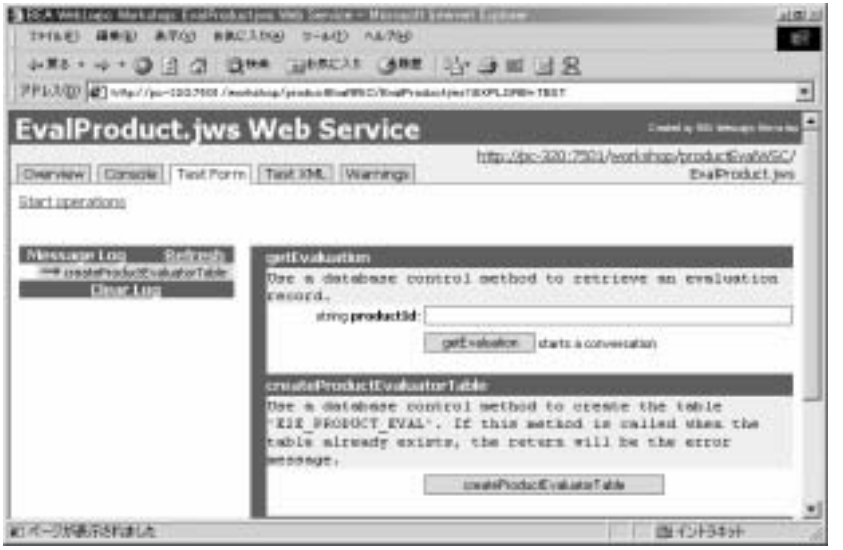
サーバが稼動した状態で、[デバック | ビルド] をクリックします。定義済みの Web サービスの構築が完了すると、WebLogic Workshop はコンパイルした JWS クラスを次のディレクトリに配置します。

```
weblogic700\samples\workshop\cgServer\.jwscompile\_jwsdir_samples
 \classes\productEvalWSC\*.class
```

サーバを稼動した状態で、[デバック | 開始] をクリックします。

テスト ページはブラウザベースです。最初に表示されるタブ付きのページは [Test Form] です。ここでは、[Overview] タブを選択します。次の画面は、`blues` というリモート マシン上でサーバを稼動している状態の [Overview] ページを示しています。

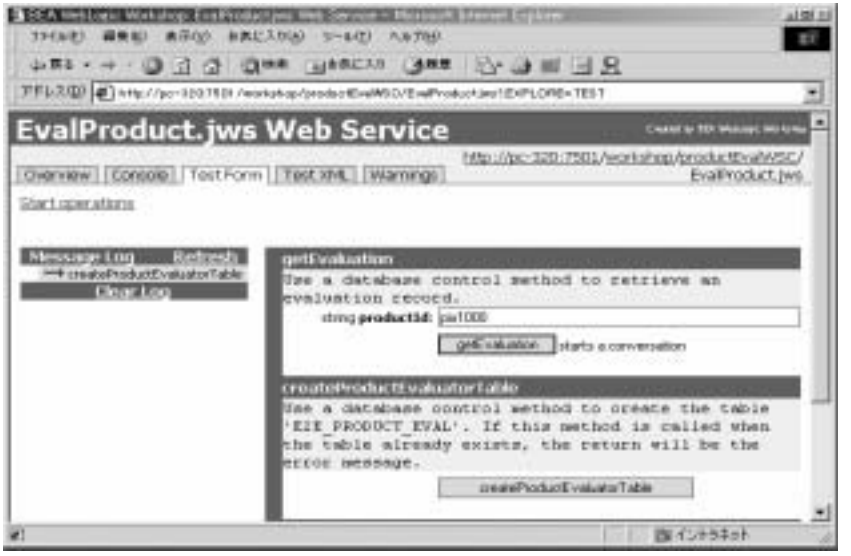
[Test Form] タブを選択します。サンプル データを入力すると、`productEvalWSC` Web サービスをテストできます。Web サービスを `cgDomain` (WebLogic Workshop で使用されるデフォルト サーバ) でテストしているため、`getEvaluation` メソッドを試行する前に、次の最初の [Test Form] ページにある **`CreateProductEvaluatorTable`** メソッドをまず始めに実行する必要があります (`CreateProductEvaluatorTable` メソッドを使用すると、ビジュアルな開発環境を使用する間だけ、テスト目的で簡単にデータベース テーブルを作成し、データを挿入できます。本稼動では、Web サービスはプロダクション データベースを使用します)。



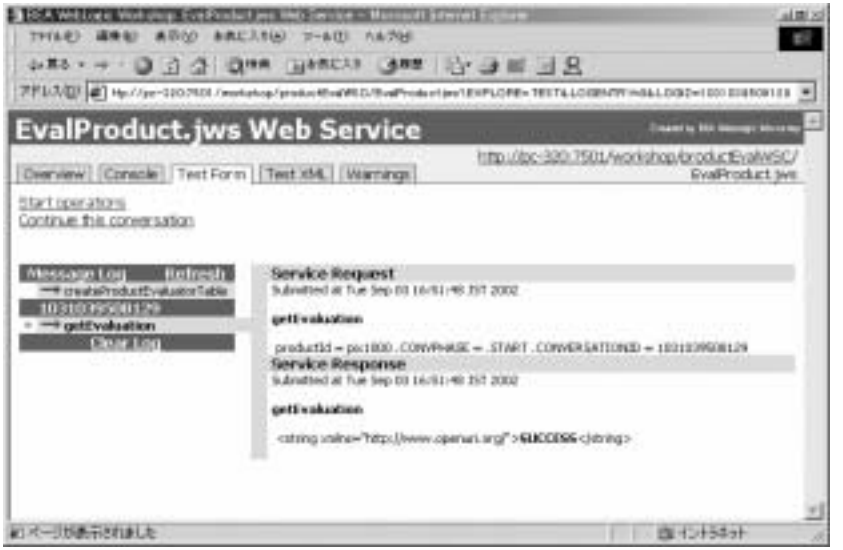
Web サービス メソッドは、「**Created**」メッセージを返します。たとえば、次のように表示されます。

次に、[Test Form] の [Start operations] リンクをクリックします。更新されたページで、Web サービスの getEvaluation メソッドのテストを開始します。たとえば、次のように [`productId`] フィールドに「**pix1000**」と入力します。次に、

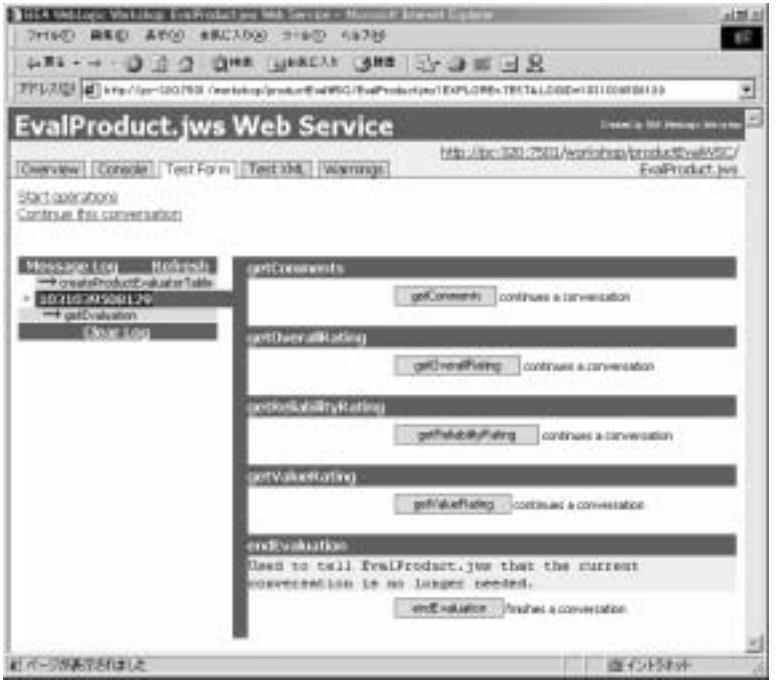
[getEvaluation] ボタンをクリックします。



Web サービスの会話が呼び出され、WebLogic Workshop のランタイム エンジンが次の画面で示すように **SUCCESS** ステータスを返します。



テストを続行するには、このページの [**Continue this conversation**] リンクをクリックします。WebLogic Workshop は、次の画面のように `productEvalWSC` Web サービスに定義されている続きのメソッドの一覧を返します。



これらのメソッドを 1 つクリックすると、この pix1000 カメラの製品レーティングを Web サービスが返します。たとえば、`getReliabilityRating` メソッド テストをクリックすると、WebLogic Workshop は次のように返します。

Web サービスが整数の **2** を返していることがわかります。商品評価ポートレットでは、この結果、最高 5 つの星のうち 2 つの星が表示されます。たとえば、次のように表示されます。

[Test Form] で、会話の他のメソッドを試行できます。前述したように、本番環境の Web サービスでは、製品の評価コメント、総合的なレーティング、信頼性のレーティング、および価格のレーティングのそれぞれを返す個別の呼び出しは作成しません。個別の呼び出しではなく、Web サービスで複合オブジェクトを作成し、1 回の呼び出しですべてのレーティングを受け取ります。

WebLogic Workshop を使用して、productEvalWSC の Java Web サービス (JWS) ファイルを作成してテストした後、Portlet Wizard を使用して、承認前のバージョンのポートレット (最終的に提示するコーディング、グラフィック、Webflow、およびパッケージ化の変更を含まない) を生成します。Portlet Wizard ツールは、WebLogic Portal が提供するグラフィカル ツール E-Business Control Center の一部です。Portlet Wizard を使用して、ポートレット用の Web サービス インタフェース コードを生成します。また、WebLogic Server を使用して、クライアント プロキシを生成します。

Web サービスに初めてアクセスすると、WebLogic Workshop がすべての EJB を自動的に生成するため、開発者が行うアクションや作業は何もないことがわかります。ここでは、最初にアクセスすると、productEvalWSC.EvalProductEJB.jar ファイルが次のディレクトリに作成されます。

```
weblogic700\samples\workshop\cgServer\.jwscompile\_jwsdir_samples
\EJB
```

Web サービスのクラス ファイルは次のディレクトリに作成されます。

```
weblogic700\samples\workshop\cgServer\.jwscompile\_jwsdir_samples
\classes\productEvalWSC\*.class
```

Web サービスの WSDL の保存

開発プロセス中はいつでも、Web サービスを記述する Web Service Description Language (WSDL) ファイルを WebLogic Server から利用できます。WSDL は、標準の XML ドキュメントで、World Wide Web Consortium (W3C。詳細は http://www.w3.org を参照) により管理されています。

WSDL ファイルは、Web サービスが利用可能なプロトコルの他、Web サービスがエクスポーズするすべてのメソッドを (送受信可能な XML メッセージの形式で) 記述します。WSDL ファイルは、クライアント アプリケーションが Web サービスを使用するために必要な情報をすべて提供します。

JWS ファイルに対応する WSDL ファイルを入手するには、次のような方法があります。

- ビジュアルな開発環境のプロジェクト ツリーで、WSDL ファイルを生成するための JWS ファイルを参照します。ここでは、EvalProduct.jws を参照します。プロジェクト ツリーで JWS ファイルを右クリックし、[JWS から

WSDL を生成] を選択します。同じディレクトリに、`EvalProductContract.wsdl` というファイルが作成されます。デフォルトでは、WSDL ファイルはファイルの作成元である JWS ファイルにリンクされます。つまり、JWS ファイルが変更されるたびに WSDL ファイルが再作成されます。このような理由により、この方法で WSDL を生成することをお勧めします。

- ブラウザで、`?WSDL` が付いた Web サービスの URL を参照します。たとえば、Web サービスが blues というマシンのデフォルトのサーバで稼動している場合は次のようになります。

  ```
  http://blues:7001/samples/productEvalWSC/EvalProduct.jws?WSDL
  ```

  ブラウザの [ファイル | 名前を付けて保存] 機能を使用して、WSDL ファイルを自分のローカル マシンに保存します。ブラウザによっては、保存したファイルの先頭および最後に HTML タグが含まれます。有効な WSDL ファイルを作成するには、これらのタグを削除する必要があります。必ず `.wsdl` のファイル タイプを指定してください。`<WebServiceName>Contract.wsdl` のネーミング規則に従って名前を付けることをお勧めします。たとえば、`EvalProductContract.wsdl` という名前を付けます。

- WebLogic Workshop のビジュアルな開発環境では、Web サービスをテストする際に [Overview] ページで [Complete WSDL] リンクをクリックして、WSDL を表示できます。次に、ブラウザの [名前を付けて保存] 機能を使用して、Web サービスのファイルを JWS ファイルと同じディレクトリに書き込みます (前述のようにネーミング規則に従ってください。また、ブラウザによっては、保存したファイルの先頭および最後に HTML タグが含まれることがあるので、これらのタグを必ず削除してください)。

Portlet Wizard を使用した、商品評価ポートレット用の最初のコードの生成

E-Business Control Center (EBCC) は、Java クライアント ベースのツールの集まりです。このツールには、ルールの定義、Webflow の編集、ポータルの作成と管理などの複雑な作業を簡単に行えるグラフィカル インタフェースが用意されています。E-Business Control Center のユーザは、ポイントアンドクリック方式のインタフェースで作業を行い、サーバと同期をとる XML ファイルを生成します。

EBCC の機能の 1 つに Portlet Wizard があります。Portlet Wizard を使用して、Web サービス用の WSDL を参照し、ポートレットに必要なコードを生成できます。次の一覧では、Portlet Wizard の一般的な手順について簡単に説明します。Portlet Wizard で作業する際には、既存の evaluator ポートレット ファイルを上書きしないようにしてください。

注意： インストールされた `evaluator.jsp` ポートレットで使用されるパッケージ化と変数は、次の手順で Portlet Wizard により作成される content.jsp ファイルとは異なります。この節の目的は、Portlet Wizard で何が最初に作成されるかを説明することです。

たとえば EBCC を Windows で起動するには、[スタート] メニューから次のようにクリックします。

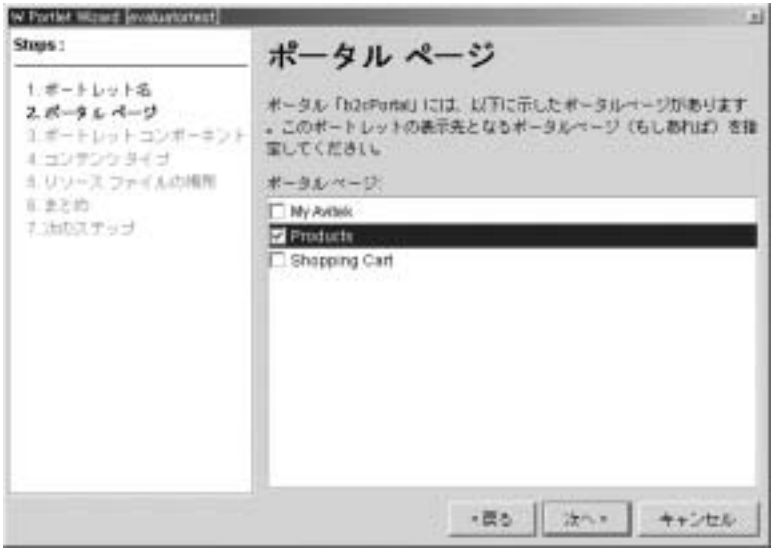
1. [プログラム | BEA WebLogic Platform 7.0 | WebLOgic Portal 7.0 | E-Business Control Center]
2. EBCC のトップレベル メニューから、[ファイル | プロジェクトを開く] をクリックします。BEA_HOME の下にある次のフォルダに移動します。

   `weblogic700\samples\platform\e2eDomain\beaApps\e2eApp-project`

3. フォルダの `e2eApp-project.eaprj` ファイルを選択し、[開く] ボタンをクリックします。
4. 次に、EBCC のトップレベル メニューから、[ファイル | 新規作成 | プレゼンテーション | ポートレット] をクリックします。
5. 次のようなオプション セットとともに最初に表示される [ポートレットの新規作成] 画面で、[OK] ボタンをクリックします。

6. [ポートレット名] 画面で、ポートレットに **evaluatortest** などの一意な名前を付けます。[ポータル] プルダウン メニューで、**[b2cPortal]** を選択します。次に、[次へ] ボタンをクリックします。

7. [ポータル ページ] 画面で、b2cPortal で利用可能なページ一覧から **[Products]** ページを選択します。次に、[次へ] ボタンをクリックします。



8. [ポートレット コンポーネント] 画面で、ポートレットに入れるコンポーネントを選択するか、またはこれらのオプションのチェックを外したままにします。次に、[次へ] ボタンをクリックします。

9. [コンテンツ タイプ] 画面で、[**Web** サービス **(** 複数可 **)**] オプションを選択します。次に、[次へ] ボタンをクリックします。

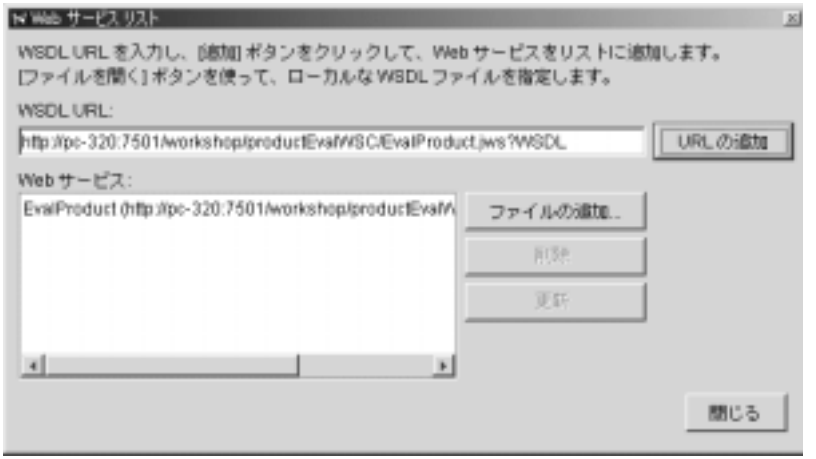
10. [サーバの場所] 画面で、WebLogic Server がインストールされている場所を指定します。次に、[次へ] ボタンをクリックします。

11. [生成されたコード タイプ] 画面で、[出力オプション] プルダウン メニューから [**Web** サービス インタフェース] を選択します。商品評価ポートレットは対話的であるため、Web サービスのインタフェース オプションが必要です。[フォーム] オプションと [呼び出し生成] オプションは、ポートレットの簡素化のために用意されています。選択が終了したら、[次へ] ボタンをクリックします。

12. Portlet Wizard を初めて使用し、現在の一覧に Web サービスが何も表示されていない場合は、[Web サービス] 画面で、[Web サービスの追加 ...] ボタンをクリックします。Web サービスが一覧に表示されていても、WebLogic Workshop で作成した `EvalProduct` がそこにない場合は、[リストの編集] ボタンをクリックします。[Web サービス リスト] 画面が表示されます。次のような WSDL URL を入力します。

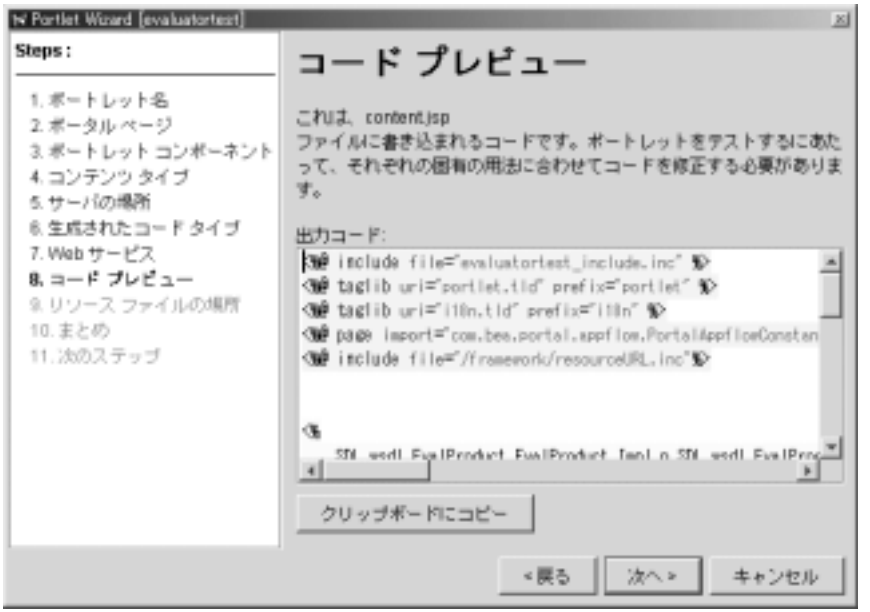
    ```
    http://localhost:7501/workshop/productEvalWSC/EvalProduct.jws?WSDL
    ```

    URL は大文字 / 小文字が区別されます。次の例では、e2eDomain アプリケーション (localhost:7501) の WSDL URL を使用します。入力ボックスに WSDL URL を入力した後、[URL の追加] ボタンをクリックします。Portlet Wizard により URL が確認され、有効であれば、[Web サービス] の一覧に追加されます。[Web サービス リスト] 画面で、[閉じる] ボタンをクリックします。



[Web サービス] 画面に戻り、一覧の **EvalProduct** Web サービスをクリックします。Portlet Wizard により WSDL URL に接続され、自己記述 Web サービス用に定義した処理が検索されます。この手順の間、進捗画面が表示されます。画面に「操作を取得しました。」のステータス メッセージが表示されたら、[次へ] ボタンをクリックします。

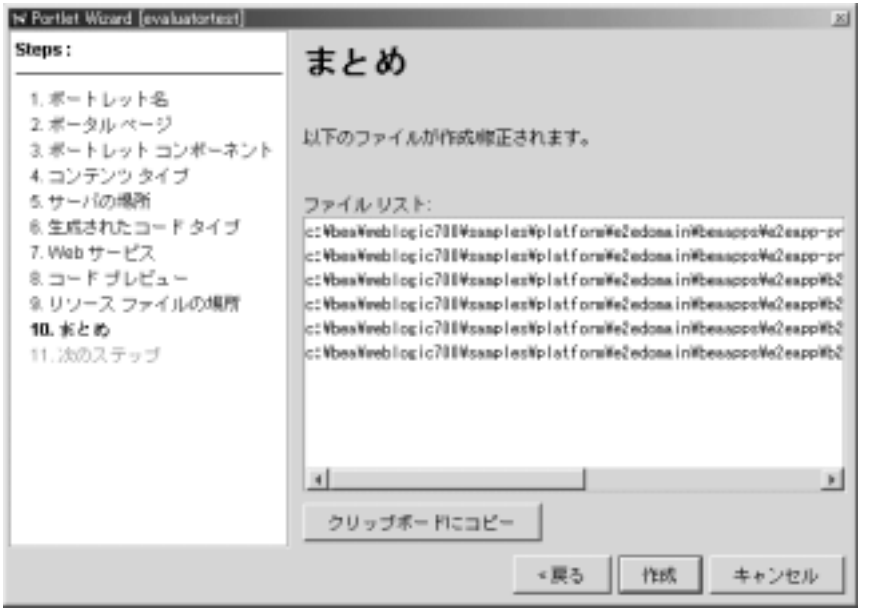
13. [コード プレビュー] 画面に、Portlet Wizard により生成されるコードが表示されます。必要に応じて、Portlet Wizard でコードを編集したり (変更内容は反映されるが検証は行われない)、クリップボードにコードをコピーして普段使用しているエディタに後で貼り付けることもできます。どちらの場合も、作成される Web サービスのインタフェース コードをプレビューした後に、[次へ] ボタンをクリックします。



14. [リソース ファイルの場所] 画面では、次のように、関連付けられている Web アプリケーションの「portlets」ディレクトリの下に、ポートレットの名前を使用したディレクトリが作成されることが表示されます。

作成するポートレット JSP ファイルの場所を指定した後、[次へ] ボタンをクリックします。

15. [まとめ] 画面では、作成または変更されるファイルの一覧が次のように表示されます。



この例では、weblogic700\samples\platform\e2eDomain\beaApps というディレクトリの下に、次のファイルが作成されます。

- `\e2eApp-project\application-sync\evaluatortest.portlet`
- `\e2eApp\b2cPortal\portlets\evaluatortest\content.jsp`
- `\e2eApp\b2cPortal\portlets\evaluatortest\evaluatortest_include.jsp`
- `\e2eApp\b2cPortal\portlets\evaluatortest\images\pt_image.gif`
- `\e2eApp\b2cPortal\WEB-INF\lib\sdl_wsdl_evalproduct.jar`

また、Portlet Wizard は、b2cPortal Web アプリケーションの利用可能なポートレット ([Products] ページではまだ有効になっていない) の一覧を示す次の XML ファイルを更新します。

- `\e2eApp-project\application-sync\webapps\b2cPortal\b2cPortal.portal`

処理を続行する場合は、[作成] ボタンをクリックします。Portlet Wizard のプロセスを単に確認したかった場合は、[キャンセル] ボタンをクリックします。

Portlet Wizard により、次のビルド コマンドが自動的に実行され、Web サービスのインタフェースが生成されます。

```
weblogic700\server\bin\ant.bat -f buildfile client-gen.xml
```

このコマンドにより、.class、.WSDL を含む JAR ファイルの他、Web サービスのインタフェースで構成されるファイルが作成され、これらのファイルがポートレットで使用できます。また、JAR ファイルには、.java のソース ファイルが含まれます。

ポートレットの初期設定をそのまま使用する場合は、最後の [次のステップ] 画面で編集オプションのチェックを外し、[閉じる] ボタンをクリックします。EBCC を使用してプロジェクトの同期をとる (ポータル定義をサーバに配置する) 前、および承認前のポートレットを [Product] ページに表示して利用できるようにする前に、ポートレットの JSP コードおよび関連ファイルを大幅に改良する必要がある場合もあります。次の節では、パッケージ化の相違点について説明します。EBCC のサーバ同期およびポートレットをポータル ページに表示し利用可能にする方法については、*『管理者ガイド』* を参照してください。

次のステップ

これで、Portlet Wizard 画面を使用して、ポートレットの最初の JSP およびインタフェース (JAR に含まれる `.class`、`.wsdl`、および `.java` のファイル) を生成する通常のプロセスを理解することができました。繰り返しますが、インストールされた商品評価ポートレットおよび `EvalProd` Web サービスで使用されるパッケージ化と変数は、Portlet Wizard を使用して作成したばかりのファイルとは異なります。JAR ファイルを作成した際には、Java ソースを抽出し、ポートレット JSP を大幅に変更し (2 つのインクルード JSP ファイルも追加)、作成したファイルを次のようにデプロイしました。注意 : Java ソースには変更を加える必要はありませんでした。単に抽出して、インストールされたサンプル ディレクトリに配置しています。

商品評価ポートレットの JSP ファイル

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal
\portlets\evaluator\*.jsp
```

主要な evaluator.jsp ポートレット ファイル、およびそこに含まれる step1.jsp ファイルと step2.jsp ファイルで、ポートレットを構成するプレゼンテーション グラフィックスを改善し、ポートレットと b2cPortal の Webflow で作業するためのスクリプトレットを追加しました。

WebLogic Workshop ランタイムにより使用される Web サービスのクライアント インタフェース、CLASS ファイル

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\WEB-INF\classes\productEvalWSC\*.class
```

Web サービスのクライアント インタフェース、JAVA ファイル

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\WEB-INF\src\productEvalWSC\*.java
```

サーバサイドの実装

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eWebServicesApp\workshop\productEvalWSC\*
```

これらのファイルには、デプロイされた EvalProduct.jws、および WebLogic Workshop 領域からコピーされ、workshop Web アプリケーションの一部としてデプロイされた ProductEvalDBControl.ctrl が含まれます。Workshop Web アプリケーションは、e2eWebServicesApp エンタープライズ アプリケーションの一部としてデプロイされています。当然のことながら、**workshop** Web アプリケーションには、*.xml ファイルのコンフィグレーション設定を含む WEB-INF サブディレクトリがあります。

クライアント生成タスクにより生成された SOAP 会話、JAVA ファイル

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\b2cPortal\WEB-INF\src\org\openuri\www\x2002\x04\soap\conversation
```

b2cPortal で使用される Payment Web サービスについては、次の節を参照してください。

支払認可 Web サービスの定義

Avitek Digital Imaging の B2C ポータルには、チェックアウト プロセスの一部として次のような [注文の送信] ページがあります。

ユーザが [注文の送信] ボタンをクリックすると、クレジット カードの支払認可プロセスが `CajunBasedPaymentPC` という Pipeline コンポーネントにより処理されます。この Pipeline コンポーネントは、Java プロキシを呼び出すことにより Payment Web サービスを呼び出します。Web サービスは、WebLogic Workshop を使用してすでに作成済みです。

Payment Web サービスは、WebLogic Workshop フレームワークの対話的な機能を使用します。最初の呼び出しでクレジット カードを認可します。つまり、認可されるクレジット カード番号と金額を引数として渡します。クレジット カードの認可が終了すると、金額を取得するよう呼び出しが行われます。最後に、金額を決済する要求が行われます。

Payment Web サービスから CajunBasedPaymentPC に返されるコードは次のとおりです。

- -1 = Error (無効な引数が渡された)

- -2 = Error (取得する金額がクレジット カードで認可されている金額を超えている)
- -3 = Error (決済金額が取得したクレジット カードの金額を超えている)
- > 0 = Success

注意： Payment Web サービスは、サードパーティの支払サービスに接続および承認されている場合のように、常にエラーなく支払い情報を送信するわけではありません。Payment サービスを介した支払処理は、本番環境向けに設計されていません。支払いを正確に処理するには、サードパーティ ベンダの支払いサービスと統合する必要があります。ただし、サンプルの Pipeline コンポーネントで表示されるコードは、エラーを適切に処理するよう設定されています。

CajunBasedPaymentPC Pipeline コンポーネントのロール

WebLogic Workshop の Payment Web サービスの定義を説明する前に、`CajunBasedPaymentPC.java` Pipeline コンポーネントの各部分を検証します。ソース ファイルは `BEA_HOME` の下の次のディレクトリにあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src\examples\e2e\b2c\payment\pipeline
```

このサンプルの Pipeline コンポーネントには、次のような import 文が多数含まれています。

```
import paymentWS.Payment_Impl;
import paymentWS.PaymentSoap;
import examples.e2e.b2c.util.B2CPortalConstants;

import com.bea.p13n.util.debug.Debug;
import org.openuri.www.*;
import org.openuri.www.x2002.x04.soap.conversation.*;

import com.beasys.commerce.axiom.units.Money;
import com.beasys.commerce.ebusiness.shoppingcart.ShoppingCart;
```

これらの文により、Payment プロキシ、生成された SOAP インタフェース、および支払タイプなどのサンプルの定数値セットのパッケージがインポートされ、アプリケーションが簡素化されます。ShoppingCart EJB もインポートされます。これは、コマース アプリケーション用のショッピング カート データを処理するものです。

b2cPortal アプリケーションにおける CajunBasedPaymentPC Pipeline コンポーネントの機能の一部として、Pipeline セッションから現在のショッピング カート データを受け取ることができます。

```
CreditCard cc = (CreditCard)
getSessionAttribute(PipelineSessionConstants.PAYMENT_CREDIT_CAR
D, namespace, pipelineSession);

ShoppingCart sc = (ShoppingCart)
getSessionAttribute(PipelineSessionConstants.SHOPPING_CART,
namespace, pipelineSession);

Money amt = (Money) sc.getTotal();
```

Web サービスのプロキシを次のようにインスタンス化します。

```
try {
       proxy = new Payment_Impl(connectString +
          "/workshop/paymentWS/Payment.jws?WSDL");
       if (DEBUG.ON) {
           DEBUG.out("proxy instantiated");
       }
```

Web サービスとの会話の準備としてヘッダー オブジェクトを設定した後、まずクレジット カード番号と値 (購入金額) を Pipeline セッションから取得し、会話を開始します。次に、認可データを Payment Web サービスに渡し、サービスにより結果が返されます。このエントリ ポイントでは、指定された金額に対する提供されたカードのクレジットを単に保存します。

```
Authorize auth = new Authorize(cc.getNumber(), amt.getValue());
    long result = proxySoap.authorize(auth, startHeader).getAuthorizeResult() ;
```

認可メソッドでエラーが返されなかった場合は、取得したデータで処理を続行します。支払トランザクションでは、「取得」という語は、クレジット カード保有者の利用可能なクレジット残高について言及する際に使用します。決済の合計金額は取得金額よりも少なくなる必要があります。

```
Capture capture = new Capture(amt.getValue());
result = proxySoap.capture(capture, continueHeader).getCaptureResult();
```

Web サービスから取得結果を受け取った後、決済データを使用して会話を終了します。

```
Settle settle = new Settle(amt.getValue());
result = proxySoap.settle(settle, continueHeader).getSettleResult();
```

エラー処理の詳細については、CajunBasedPaymentPC.java のソース ファイルを参照してください。前述のとおり、このファイルは次のディレクトリにあります。

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp\src
 \examples\e2e\b2c\payment\pipeline
```

既存の paymentWS Web サービスの表示

WebLogic Workshop のプロジェクト ツリー ウィンドウで、次のディレクトリにコピーした paymentWS フォルダをダブルクリックします。

```
weblogic700\samples\workshop\applications\samples\paymentWS
```

この手順をツアーの前の部分で省略している場合は、「プロジェクトの場所について」を参照してから、この節に戻ってください。

展開した paymentWS プロジェクト フォルダ内の Payment.jws という Java Web サービス ファイルをダブルクリックします。次の画面は、この Web サービスの [デザイン ビュー] 画面です。

WebLogic Workshop を使用して、Web サービス用の次のメソッドを定義しました。

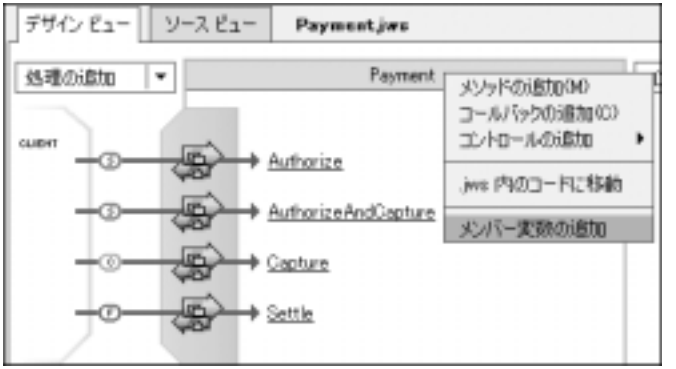
- `Authorize`
- `AuthorizeAndCapture`
- `Capture`
- `Settle`

[デザイン ビュー] 画面を見ると、小さな (s)、(c)、(F) の記号が JWS メソッドに付いていることがわかります。これらの記号は、どのメソッドが Web サービスの会話を開始 (`Authorize` または `AuthorizeAndCapture`) し、どのメソッドが会話を続行 (`Capture`) し、どのメソッドが会話を終了 (`Settle`) するかを示します。

成功した場合、Web サービス メソッドは一意の `transactionID` を返します。サンプルでは ID はトランザクションが発生した時間 (ミリ秒) で算出されます。このようにして ID を返すのは、クレジット カード認可アプリケーションでは典型的な方法です。

WebLogic Workshop でのメソッド定義は単純です。このツアーでは、コントロールとメソッドを作成する手順は説明しません。これらの手順は、WebLogic Workshop のドキュメントに記載されています。このドキュメントは、ビジュアルな開発環境で参照できます。また、BEA e-docs Web サイトの WebLogic Workshop のページでも参照できます。

Workshop では、[ソース ビュー] に切り替えて、Web サービス メソッドの定義を表示できます。これらの 4 つのメソッドを追加後、ビジュアルな開発環境に表示するビジネス ロジックを追加する前に、Web サービスに必要なメンバー変数を追加しました。メンバー変数を追加するには、ビジュアルな開発環境の [Payment] の見出しを右クリックし、次のようなプルダウン メニューから [メンバー変数の追加] を選択します。



Payment.jws に定義されたメンバー変数は次のとおりです。サフィックスのアンダースコアはインスタンス変数を示します。

- `openToBuy_`
- `amountToCapture_`
- `transactionID_`
- `ERR_CODE_BAD_ARGUMENT`
- `ERR_CODE_AMT_TO_CAPTURE_MORE_THAN_AUTH`
- `ERR_CODE_AMT_TO_SETTLE_MORE_THAN_CAPTURE`

支払モデルと定義されたメソッド

クレジット カードのトランザクションには、端末ベースとホストベースの 2 種類の支払モデルがあります。この 2 つの支払モデルの相違点は、トランザクション バッチの保存場所です。ホストベース モデルでは、トランザクション バッチは販売者サイトのローカル システムではなくホスト ネットワーク上に保存されます。通常、決済は 1 日の終わりに発生しますが、販売者は、決済プロセスを起動するために何らかの作業を行う必要がありません。

端末ベース モデルでは、トランザクション バッチは販売者サイトのローカル システム上にデータ ファイルとして保存されます。販売者は、各日の終わりに決済プロセスを起動して、資金を取引銀行の口座に送金する必要があります。

金融機関から割り当てられる端末ベースの支払モデルを次に示します。

- AUTO_MARK_AUTO_SETTLE

  この支払モデルは、非耐久消費財に使用されます。決済は認可が完了すると直ちに行われます。これは、非耐久消費財は購入時点ですぐに出荷されることが一般的であるためです。

- AUTO_MARK_MANUAL_SETTLE

  この支払モデルは、商品が認可の時点で出荷されたが、販売者が資金をその日ではなく後で送金する場合に使用します。

- MANUAL_MARK_AUTO_SETTLE

  この支払モデルは、商品を出荷したことを販売者が指定でき、その時点で決済を自動的に行うものです。

- MANUAL_MARK_MANUAL_SETTLE

  これは、最も柔軟な支払モデルで、商品の出荷時期と資金の送金時期を販売者が指定できるものです。マーク プロセスにより、販売者は商品が出荷されたことを指定します。決済プロセスにより、販売者は資金の送金を指定します。

割り当てられるホストベースの支払モデルを次に示します。

- HOST_AUTH_CAPTURE

  この支払モデルは、サービス、デジタル商品の販売、または受注後 24 時間以内に出荷される有形商品に使用されます。この場合、販売者は購入金額の認可のみを受け取る必要があります。バッチへの認可の取得とトランザクションの決済は、認可時に販売者に代わってプロセッサにより実行されます。

- HOST_POST_AUTH_CAPTURE

  受注後、注文を処理するまでに 2 日以上かかる場合、販売者は、認可とトランザクションの取得を別々に行う必要があります。この支払モデルでは、顧客が購入を決めた時点で認可が実行されます。取得は、販売者が注文を出荷する際に実行されます。プロセッサは、その日の指定された時刻にバッチ処理されたトランザクションを決済します。

前述したように、Authorize メソッドは端末ベースの支払モデルで使用されます。このエントリ ポイントでは、クレジット カード番号を検証し、そのカードのクレジットから指定された金額を確保します。検証が終わると、データベース テーブルに新しいエントリが作成されます。このエントリによって、インシデントが記録され、支払モデルに基づく状態が設定されます。トランザクションの金額が顧客のクレジット残高の openToBuy フィールドから差し引かれます。ただし、資金は決済時まで販売者には送金されません。

注意： 端末ベースのプロセッサを使用している販売者は、売上資金が口座に送金される前に、取得と決済の手順を実行する必要があります。この手順は、「Auto Mark/Auto Settle」のプロセッサ コンフィグレーションに応じて、Capture および Settle、またはそのどちらかを呼び出して実行します。

この `AuthorizeAndCapture` メソッドは、ホストベースの支払モデルで使用されます。このエントリ ポイントでは、クレジット カード番号を検証し、そのカードのクレジットから指定された金額を確保します。検証が終わると、データベース テーブルに新しいエントリが作成されます。このエントリによって、インシデントが記録され、支払モデルに基づく状態が設定されます。トランザクションの金額が顧客のクレジット残高の `openToBuy` フィールドから差し引かれます。ただし、資金は決済時まで販売者には送金されません。

注意： ホストベースの認可後取得プロセッサを使用している販売者は、売上資金が口座に送金される前に、取得と決済の手順を実行する必要があります。

Payment Web サービスのこの `Capture` メソッドにより、クレジット カード保有者の利用可能なクレジット残高を確認できます。決済の合計金額は、取得金額よりも少なくなる必要があります。次の Settle メソッドで確認されます。

```
* @jws:operation
* @jws:conversation phase="finish"
*/

public long Settle(Double amountToSettle)
{

if ((amountToSettle == null) ||
          (amountToSettle.compareTo(new Double(0)) < 0))
          return ERR_CODE_BAD_ARGUMENT;

if (amountToSettle.compareTo(amountToCapture_) > 0)
          return ERR_CODE_AMT_TO_SETTLE_MORE_THAN_CAPTURE;

return transactionID_;

}
```

ビジュアルな開発環境におけるメソッド定義の詳細については、WebLogic Workshop のドキュメントを参照してください。このドキュメントはビジュアルな開発環境で参照できます。また、BEA e-docs Web サイトの WebLogic Workshop のページでも参照できます。

Payment Web サービスの構築とテスト

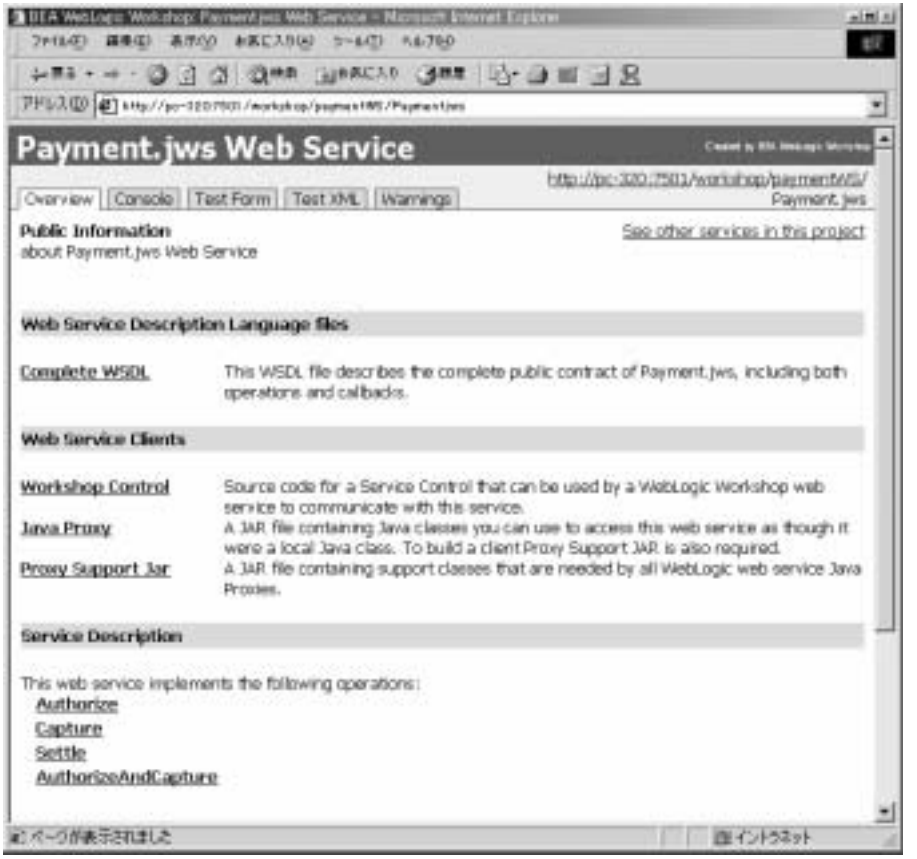
cgDomain の WebLogic Server インスタンスが稼動していない場合は、これを起動します。この手順をツアーの前の部分で省略している場合は、4-6 ページの「サーバの起動オプション」を参照してください。その後、この節に戻ってください。

サーバが稼動した状態で、[デバック | ビルド] をクリックします。「ビルドを開始しました。」というメッセージがビジュアルな開発環境の画面の左下隅に表示されます。定義済みの Web サービスの構築が完了すると、WebLogic Workshop はコンパイルした JWS クラスを次のディレクトリに配置します。

```
weblogic700\samples\workshop\cgServer
 \.jwscompile\_jwsdir_samples\classes\paymentWS\*.class
```

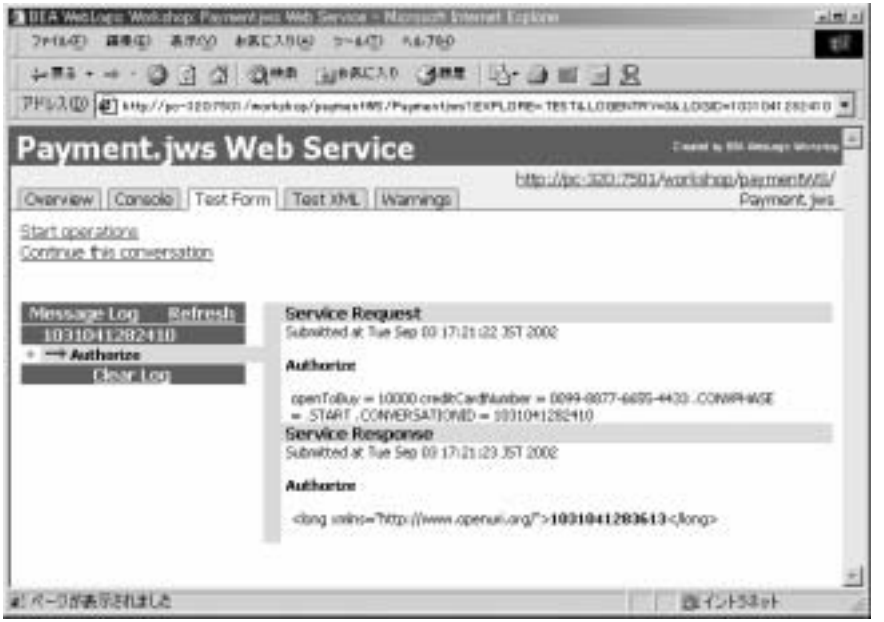
サーバを稼動した状態で、[デバック | 開始] をクリックします。

テスト ページはブラウザベースです。最初に表示されるタブ付きのページは [Test Form] です。ここでは、[Overview] タブを選択します。次の画面は、blues というリモート マシン上でサーバを稼動している状態の [Overview] ページを示しています。

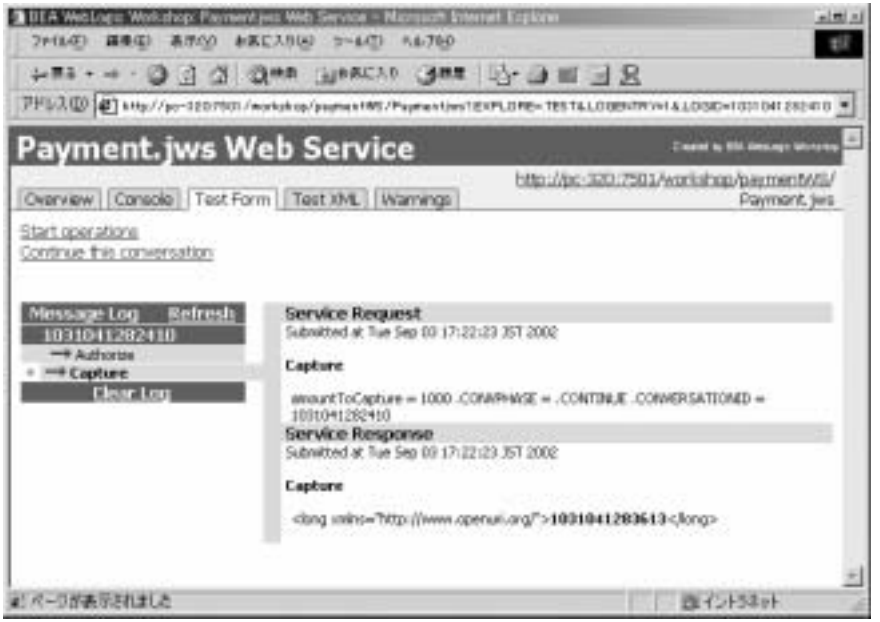


[Test Form] タブを選択します。サンプル データを入力すると、Payment Web サービスをテストできます。たとえば、クレジット カードの番号 `0099-8877-6655-4433` (ハイフンはなくても可) および `openToBuy` の認可金額 `10000` を入力できます。このテストには、次のように `Authorize` メソッドを使用します。

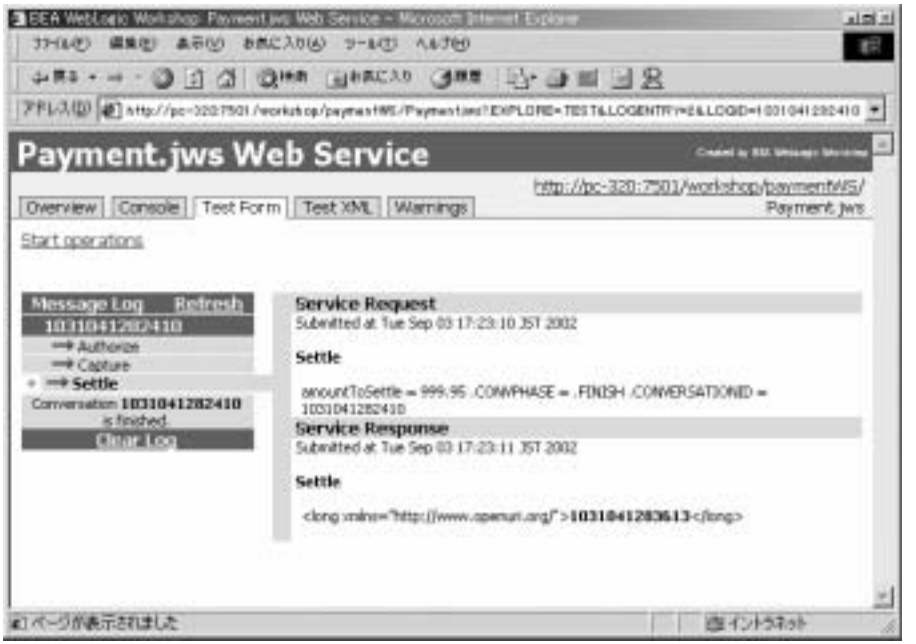
会話を開始するには、[Authorize] ボタンをクリックします。この Web サービス メソッドは、成功した場合に、一意の `transactionID` を返します。この ID は例で示されているとおり、トランザクションの発生した時間 (ミリ秒) で算出されます。



テストを続行するには、このページの [**Continue this conversation**] リンクをクリックします。WebLogic Workshop は、Payment Web サービスに定義されている続きのメソッドの一覧を返します。Capture メソッドの `amountToCapture` 変数に「1000」を入力します。次に、[Capture] ボタンをクリックします。取得する金額が認可金額よりも少ない場合 (成功した場合)、Web サービスは一意の `transactionID` を返します。



次に、[Continue this conversation] リンクを再度クリックします。Settle メソッドの `amountToSettle` 変数に「999.95」などの値を入力します。次に、[Settle] ボタンをクリックします。決済する金額が取得金額よりも少ない場合 (成功した場合)、Web サービスは一意の `transactionID` を返します。



[Test Form] では、他の値を入力してエラー処理を確認することができます。

WebLogic Workshop で Web サービスを構築すると、すべての EJB が自動的に生成されるため、開発者が行うアクションや作業は何もないことがわかります。ここでは、最初のアクセスで **paymentWS.PaymentEJB.jar** ファイルが次のディレクトリに作成されました。

```
weblogic700\samples\workshop\cgServer
 \.jwscompile\_jwsdir_samples\EJB
```

Web サービスのクラス ファイルは次のディレクトリに作成されます。

```
weblogic700\samples\workshop\cgServer
 \.jwscompile\_jwsdir_samples\classes\paymentWS\Payment.class
```

Web サービスの WSDL の保存

開発プロセス中はいつでも、Web サービスを記述する Web Service Description Language (WSDL) ファイルを WebLogic Server から利用できます。WSDL は、標準の XML ドキュメントで、World Wide Web Consortium (W3C。詳細は `http://www.w3.org` を参照) により管理されています。

WSDL ファイルは、Web サービスが利用可能なプロトコル、および Web サービスがエクスポーズするすべてのメソッドを記述 (送受信可能な XML メッセージの形で) します。WSDL ファイルは、クライアント アプリケーションが Web サービスを使用するために必要な情報をすべて提供します。

JWS ファイルに対応する WSDL ファイルを入手するには、次のような方法があります。

- WebLogic Workshop のプロジェクト ツリーで、WSDL ファイルを生成する JWS ファイルを参照します。ここでは、`Payment.jws` を参照します。プロジェクト ツリーで JWS ファイルを右クリックし、[JWS から WSDL を生成] を選択します。同じディレクトリに PaymentContract.wsdl というファイルが作成されます。デフォルトでは、WSDL ファイルはファイルの作成元である JWS ファイルにリンクされます。つまり、JWS ファイルが変更されるたびに WSDL ファイルが再作成されます。このような理由により、この方法で WSDL を生成することをお勧めします。
- ブラウザで、`?WSDL` が付いた Web サービスの URL を参照します。たとえば、Web サービスが `blues` というマシンのデフォルトのサーバで稼動している場合は次のようになります。

  ```
  http://blues:7001/samples/paymentWS/Payment.jws?WSDL
  ```

  ブラウザの [ファイル | 名前を付けて保存] 機能を使用して、WSDL ファイルを自分のローカル マシンに保存します。ブラウザによっては、保存したファイルの先頭および最後に HTML タグが含まれます。有効な WSDL ファイルを作成するには、これらのタグを削除する必要があります。必ず `.wsdl` のファイル タイプを指定してください。`<WebServiceName>Contract.wsdl` のネーミング規則に従って名前を付けることをお勧めします。たとえば、`PaymentContract.wsdl` という名前を付けます。
- WebLogic Workshop では、Web サービスをテストする際に [Overview] ページで [**Complete WSDL**] リンクをクリックして、WSDL を表示できます。次に、ブラウザの [名前を付けて保存] 機能を使用して、Web サービスのファ

イルを JWS ファイルと同じディレクトリに書き込みます (前述のようにネーミング規則に従ってください。また、ブラウザによっては、保存したファイルの先頭および最後に HTML タグが含まれることがあるので、これらのタグを必ず削除してください)。

e2eDomain で使用するための PaymentWS Web サービスのパッケージ化

このツアーでは、Payment Web サービスを構成するファイルを、インストールされた e2eDomain 用に次のようにパッケージ化しています。

WebLogic Workshop ランタイムにより使用される Web サービスのクライアント インタフェース、CLASS ファイル

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp
 \b2cPortal\WEB-INF\classes\paymentWS\*.class
```

Web サービスのクライアント インタフェース、JAVA ファイル

```
weblogic700\samples\platform\e2eDomain\beaApps\e2eApp
 \src\paymentWS\*.java
```

サーバサイドの実装

```
weblogic700\samples\platform\e2eDomain\beaApps
 \e2eWebServicesApp\workshop\paymentWS\Payment.jws
```

これは、WebLogic Workshop 領域からコピーされ、**workshop** Web アプリケーション (e2eWebServicesApp のエンタープライズ アプリケーションの一部としてデプロイされた) の一部としてデプロイされた Payment.jws ファイルです。当然のことながら、workshop Web アプリケーションには、*.xml ファイルのコンフィグレーション設定を含む WEB-INF サブディレクトリがあります。

paymentWS Web サービスにより使用される EJB の JAR ファイル

```
weblogic700\samples\platform\e2eDomain\e2eServer
 \.jwscompile\_jwsdir_workshop\EJB\paymentWS.PaymentEJB.jar
```

支払処理のための Webflow 入力プロセッサ、CLASS ファイル

```
weblogic700\samples\platform\e2eDomain\beaApps
 \e2eApp\b2cPortal\WEB-INF\classes\examples\e2e
 \b2c\payment\webflow\*IP.class
```

Web サービス ツアーの完了

これで、Web サービス ツアーは終了です。オンライン ツアーの場合は、[概要に戻る] ボタンをクリックして、サンプルの [はじめに] ページに戻ってください。