IBM® WebSphere® Commerce

プログラマーズ・ガイド

バージョン 5.4

GC88-9273-00
(英文原典：GC09-4951-00)

IBM® WebSphere® Commerce

プログラマーズ・ガイド

バージョン 5.4

GC88-9273-00
(英文原典：GC09-4951-00)

ご注意!

本書および本書で紹介する製品をご使用になる前に、『特記事項』に記載されている情報をお読みください。

本書は以下の製品に適用されます。

- IBM® WebSphere® Commerce Business Edition for Windows NT® and Windows® 2000 バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Business Edition for AIX® バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Business Edition for Solaris オペレーティング環境ソフトウェア バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Business Edition for Linux バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Studio, Business Developer Edition for Windows NT and Windows 2000 バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Professional Edition for Windows NT and Windows 2000 バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Professional Edition for AIX バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Professional Edition for Solaris オペレーティング環境ソフトウェア バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Professional Edition for Linux バージョン 5.4 (プログラム 5724-A18)
- IBM WebSphere Commerce Studio, Professional Developer Edition for Windows NT and Windows 2000 バージョン 5.4 (プログラム 5724-A18)

および、新版で特に指定のない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。製品のレベルにあった版を使用していることをご確認ください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは `http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。 (URL は、変更になる場合があります)

原 典 :	GC09-4951-00 IBM WebSphere Commerce Programmer's Guide Version 5.4
発 行 :	日本アイ・ビー・エム株式会社
担 当 :	ナショナル・ランゲージ・サポート

第1刷 2002.6

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この (書体*) は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、平成角ゴシック体™W7

はじめに

IBM WebSphere Commerce プログラマーズ・ガイド は、 WebSphere Commerce のアーキテクチャーとプログラミング・モデルについて説明します。特に、以下のトピックについて詳細に説明します。

- コンポーネント間の相互作用
- デザイン・パターン
- 永続オブジェクト・モデル
- アクセス制御
- エラー処理とメッセージ
- コマンドのインプリメンテーション
- 開発ツール
- カスタマイズ・コードのデプロイメント

さらに、本書には以下のチュートリアルが含まれています。

ビジネス・ロジックの新規作成
このチュートリアルでは、WebSphere Commerce プログラミングに従って、新しいコマンド、データ Bean、およびエンタープライズ Bean を作成する方法を示します。また、ロジックを既存のストアに統合し、コードをターゲットの WebSphere Commerce Server にデプロイメントする方法も示します。

既存のビジネス・ロジックの変更および拡張
このチュートリアルには 2 つのセクションがあります。最初のセクションでは、既存のコントローラー・コマンドに新規ロジックを追加する方法を示します。 2 番目のセクションでは、既存のタスク・コマンドと WebSphere Commerce エンティティー Bean を修正する方法を示します。また、変更を既存のストアに統合し、コードをターゲットの WebSphere Commerce Server にデプロイメントする方法も示します。

本書の表記規則

本書では、以下のような強調表示の規則を使用しています。

太文字は、コマンドまたはグラフィカル・ユーザー・インターフェース (GUI) のコントロール (フィールド、ボタン、またはメニュー選択項目の名前など) を表します。

`モノスペース (Monospace)` は、ユーザーが表示どおりに入力するテキストの例、およびディレクトリーのパスを表します。

イタリック は、強調のため、およびユーザーが値を置き換える変数を表すために使用されます。

このアイコンはヒントを表します。これは、タスクを完了するのに役立つ可能性のある追加情報です。

Windows は、WebSphere Commerce for Windows NT および Windows 2000 に固有の情報を示します。

AIX は、WebSphere Commerce for AIX に固有の情報を示します。

Solaris は、WebSphere Commerce for Solaris™ オペレーティング環境ソフトウェアに固有の情報を示します。

400 は、WebSphere Commerce for the IBM @server iSeries™ 400® (以前の AS/400®) に固有の情報を示します。

Linux は、WebSphere Commerce for Linux に固有の情報を示します。

DB2 は、DB2® ユニバーサル・データベースに固有の情報を示します。

Oracle は、Oracle® に固有の情報を示します。

Professional WebSphere Commerce Professional Edition に固有の情報を示します。

Business WebSphere Commerce Business Edition に固有の情報を示します。

知識に関する条件

本書の対象読者は、 WebSphere Commerce アプリケーションのカスタマイズ方法を理解する必要のある、ストア開発者です。プログラムによる拡張を施す場合は、以下に関する知識が必要です。

- Java™
- Enterprise JavaBeans (EJB) コンポーネント・アーキテクチャー
- JavaServer Pages テクノロジー
- HTML
- データベース・テクノロジー
- VisualAge® for Java、エンタープライズ版 バージョン 3.5

追加情報の入手先

本書は、将来、更新される可能性があります。以下の WebSphere Commerce Web サイトで更新情報をチェックしてください。

Business

http://www.ibm.com/software/webservers/commerce/wc_be/lit-tech-general.html

Professional

http://www.ibm.com/software/webservers/commerce/wc_pe/lit-tech-general.html

更新内容には、新しい情報、追加または更新されたチュートリアル、およびチュートリアルに関連するコードが含まれます。

目次

第 1 部 概念とアーキテクチャー

第 1 章 概要

WebSphere Commerce ソフトウェア・コンポーネント

WebSphere Commerce Server がどのように機能するかを調べる前に、カスタマイズ・プロセスに関連のあるソフトウェア・コンポーネントについて概観することが役に立ちます。以下の図は、これらのソフトウェア・プロダクトを単純化して表示したものです。

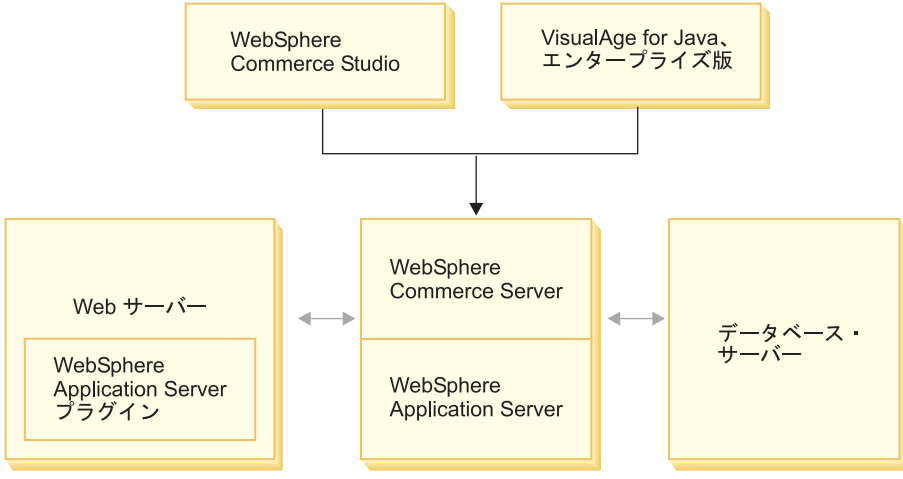


図 1.

Web サーバーは、 e-commerce アプリケーションに対する着信 HTTP 要求のための最初の接触点です。 WebSphere Application Server と効率的にインターフェースをとるために、同サーバーは WebSphere Application Server プラグインを使用します。

WebSphere Commerce Server は WebSphere Application Server 内で稼働して、アプリケーション・サーバーのフィーチャーの多くを利用します。データベース・サーバーは、商品およびショッパーのデータを含め、アプリケーションのデータの大部分を保持します。一般に、アプリケーションに対する拡張は、 WebSphere Commerce Server のコードの修正、または拡張を施すことによって行われます。加えて、データベース内の WebSphere Commerce データベース・スキーマのレルムの外に属するデータを保管する必要もあります。

開発者はカスタマイズされたビジネス・ロジックを作成するための 2 つの主要なツールを使用することができます。それは WebSphere Commerce Studio と VisualAge for Java エンタープライズ版です。 WebSphere Commerce Studio は、ストア・フロント資産 (JSP テンプレートなど) の作成と管理のために使用されます。 VisualAge for Java エン

ターブライズ版は、新しいビジネス・ロジックを Java で作成するために使用されます。その際、既存の機能性を拡張するか、全く新規の機能を作成するかのいずれかになります。アプリケーションがデータベース・スキーマの拡張を必要とする場合、データベース開発者は、任意のデータベース開発ツールを使用して新しいテーブルを作成する必要があります。

WebSphere Commerce アプリケーション・アーキテクチャー

ここまでで、カスタマイズ用に関連付けられた各種のソフトウェア・コンポーネントが組み合わされる仕組みを見てきましたが、次に、アプリケーション・アーキテクチャーを理解することが重要となります。これにより、基礎層を成すパーツと、ユーザーが変更できるパーツとを区別できるようになります。以下の図は、アプリケーション・アーキテクチャーを構成する各種の層を示しています。

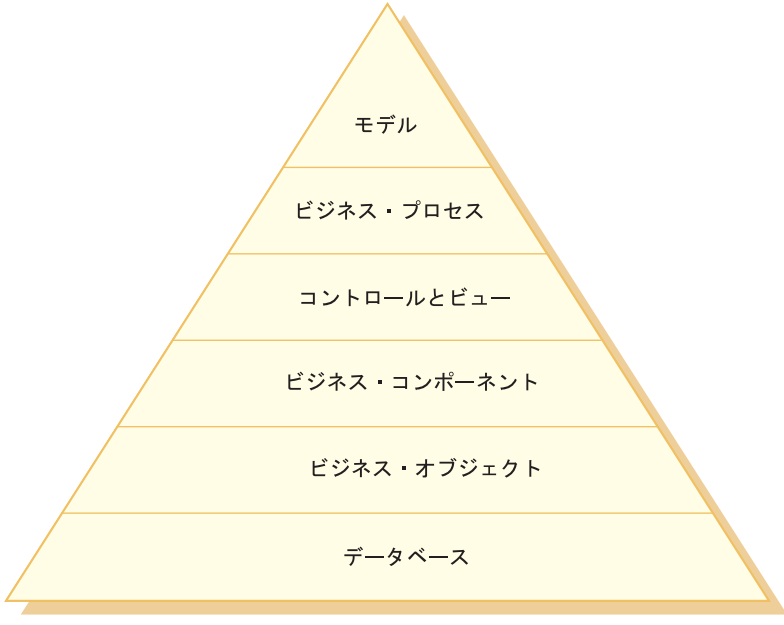


図 2.

アプリケーション・アーキテクチャーの各層について、以下に説明します。

データベース

WebSphere Commerce では、e-commerce アプリケーションとそのデータ要件向けに特に設計されたデータベース・スキーマを使用します。このスキーマにおけるテーブルの例を以下に示します。

- User (ユーザー)
- Order (オーダー)

- Product (商品)

ビジネス・オブジェクト

ビジネス・オブジェクトはコマース・ドメイン内のエンティティーを表しており、データベース内に含まれる情報の抽出や解釈に必要なデータ中心のロジックをカプセル化したものです。これらのエンティティーは Enterprise JavaBeans 仕様に準拠しています。

これらのエンティティー Bean はコマース・アプリケーションとデータベースとのインターフェースとして働きます。加えて、エンティティー Bean は、データベース・テーブルの列同士の複雑な関係に比べて、理解するのが容易です。

ビジネス・コンポーネント

ビジネス・コンポーネントはビジネス・ロジックのそれぞれの単位です。それらは、大きく分けられた手順上のビジネス・ロジックを実行します。そのロジックは、コントローラー・コマンドおよびタスク・コマンドの WebSphere Commerce モデルを使用してインプリメントされます。この種のコンポーネントの一例は OrderProcess コントローラー・コマンドです。この特定コマンドは、典型的なオーダーの処理に必要なすべてのビジネス・ロジックをカプセル化します。 e-commerce アプリケーションが OrderProcess コマンドを呼び出すと、今度はそのコマンドがいくつかのタスク・コマンドを呼び出して、個々の作業単位を実行します。たとえば、個々のタスク・コマンドは、オーダーの要件にかなう十分な在庫があることを確認し、支払いを処理し、オーダーの状況を更新し、処理が完了したら、該当する量だけ在庫を減らします。

コントロールとビュー

Web コントローラーは、該当するコントローラー・コマンドのインプリメンテーションと、使用されるビューを判別します。インプリメンテーションはストアごとに固有にすることができます。
ビューは、コマンドおよびユーザー処置の結果を表示します。ビューは JSP テンプレートを使用してインプリメントされます。ビューの例としては、ProductDisplay (ショッパーが選択した商品に関係のある情報を表示する商品ページを戻します) と、 OrderPrepare (適切なオーダー情報を送信するためのフォームをショッパーに提示します) が挙げられます。

ビジネス・プロセス

ビジネス・コンポーネントとビューのセットが一緒になって、ビジネス・プロセスとして知られる、ワークフローおよびサイトフローのプロセスが作成されます。ビジネス・プロセスの例には、以下のものがあります。

ユーザー登録

このビジネス・プロセスには、ビジネス・コンポーネント (たとえば、新規ユーザーの登録レコードを作成する UserRegistrationAdd コマンド) と、ユーザーの登録のプロセスに関与するすべてのステップに関連したビューが含まれます。

カタログ・ナビゲーション
このビジネス・プロセスには、ビジネス・コンポーネント (たとえば、ストアのカタログを表示する StoreCatalogDisplay コマンド、およびカタログの中のカテゴリーを示す CategoryDisplay コマンド) と、カタログの中をナビゲートするプロセスに関与するすべてのステップに関連したビューが含まれます。

モデル 図のこれより下の層が一緒になって、e-commerce ビジネス・モデルを形成します。 e-commerce ビジネス・モデルの一例としては、 InFashion サンプル・ストアで使用されるビジネス対顧客モデルがあります。別の例としては、ToolTech サンプル・ストアで使用される企業間取引モデルがあります。

WebSphere Commerce ランタイム・アーキテクチャー

ここまでのセクションではアプリケーション・アーキテクチャーを紹介し、ビジネス・アプリケーションの観点から、WebSphere Commerce アプリケーションの各層を説明しました。このセクションでは、ランタイム・アーキテクチャーをインプリメントする方法を説明します。

WebSphere Commerce ランタイム・アーキテクチャーの主なコンポーネントは以下のとおりです。

- サーブレット・エンジン
- プロトコル・リスナー
- アダプター・マネージャー
- アダプター
- Web コントローラー
- コマンド
- エンティティー Bean
- データ Bean
- データ Bean マネージャー
- 表示ページ
- XML ファイル

WebSphere Commerce コンポーネント間の相互作用が以下の図に示されています。各コンポーネントの詳細情報については、以下のセクションを参照してください。

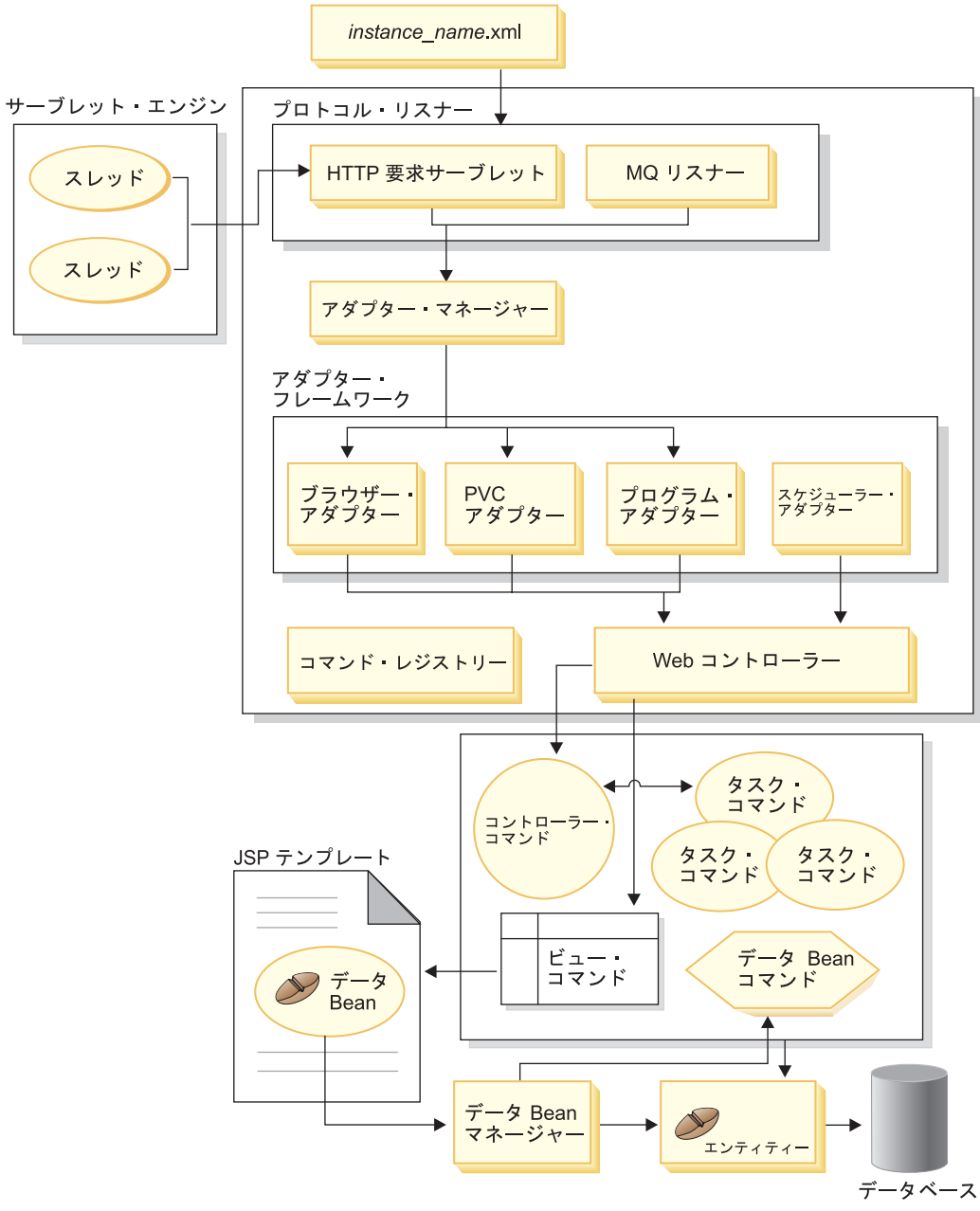

instance_name.xml
サーブレット・エンジン
スレッド
スレッド
プロトコル・リスナー
HTTP 要求サーブレット
MQ リスナー
アダプター・マネージャー
アダプター・
フレームワーク
ブラウザー・
アダプター
PVC
アダプター
プログラム・
アダプター
スケジューラー・
アダプター
コマンド・レジストリー
Web コントローラー
コントローラー・
コマンド
タスク・
コマンド
タスク・
コマンド
タスク・
コマンド
JSP テンプレート
データ
Bean
ビュー・
コマンド
データ Bean
コマンド
データ Bean
マネージャー
エンティティー
データベース

図 3.

サーブレット・エンジン

サーブレット・エンジンは WebSphere Application Server ランタイム環境の一部であり、インバウンド URL 要求の要求ディスパッチャーとして働きます。サーブレット・エンジンは、要求を処理するスレッドのプールを管理します。各インバウンド要求は、それぞれ別個のスレッドで実行されます。

以前のバージョンの WebSphere Commerce では、 URL 要求のための独自のタスク・ディスパッチャーをインプリメントする C++ アプリケーション・サーバーを使用していました。それは事前割り当てされたシステム・プロセスのセットを保持するという方法を採っていました。システム・プロセスを使用するこの古いモデルは、 Java スレッドを使用する新しいモデルに比べて、リソースに負担がかかります。新しい WebSphere Commerce ランタイム・アーキテクチャーでは、サーブレット・エンジンを活用することにより、拡張の容易なコマース・ソリューションが提供できるようになりました。

アダプター・マネージャー

アダプター・マネージャーは、要求を処理できるアダプターを判別し、そのアダプターに要求を転送します。

プロトコル・リスナー

WebSphere Commerce のコマンドは、さまざまな装置から呼び出すことができます。コマンドを呼び出すことができる装置の例には、以下のものがあります。

- 標準的なインターネット・ブラウザー
- インターネット・ブラウザーを使った携帯電話
- MQSeries® を使用して XML メッセージを送信する、ビジネス間アプリケーション
- HTTP を介する XML を使用して要求を送信する調達システム
- バックグラウンド・ジョブを実行する、WebSphere Commerce スケジューラー

装置の使用する通信プロトコルは多岐に渡ります。プロトコル・リスナーは、トランスポートからインバウンド要求を受け取り、それらの要求を、使用されているプロトコルに基づいて適切なアダプターにディスパッチする、ランタイム・コンポーネントです。プロトコル・リスナーには、以下のものが含まれます。

- 要求サーブレット
- MQSeries リスナー

要求サーブレットは、サーブレット・エンジンから URL 要求を受け取ると、この要求をアダプター・マネージャーに渡します。すると、アダプター・マネージャーは、アダプターのタイプを照会し、要求を処理できるアダプターを判別します。いったん特定のアダプターが判別されると、要求はそのアダプターに渡されます。

要求サーブレットは、初期化されると、 *instance_name*.xml 構成ファイルを読み取ります。 XML ファイル内の構成ブロックの 1 つでは、すべてのアダプターを定義しています。要求サーブレットの init() メソッドは、定義されたすべてのアダプターを初期化します。

MQSeries リスナーは、リモート・プログラムから XML ベースの MQSeries メッセージを受け取り、要求を非 HTTP アダプター・マネージャーにディスパッチします。

Job Scheduler には、プロトコル・リスナーは不要です。

アダプター

WebSphere Commerce アダプターは、要求を Web コントローラーに渡す前に処理機能を実行する、装置固有のコンポーネントです。アダプターが実行する処理の例には、以下のものがあります。

- Web コントローラーに対し、要求を装置のタイプに固有の方法で処理するよう指示する。たとえば、パーベイシブ・コンピューティング (PvC) 装置アダプターは、 Web コントローラーに対し、元の要求に含まれている HTTPS 検査を無視するよう指示することがあります。
- インバウンド要求のメッセージ・フォーマットを、WebSphere Commerce コマンドが解析できるプロパティーのセットに変換する。
- 装置固有のセッション持続性を指定する。

以下の図は、WebSphere Commerce アダプター・フレームワークのインプリメンテーション・クラス階層を示しています。

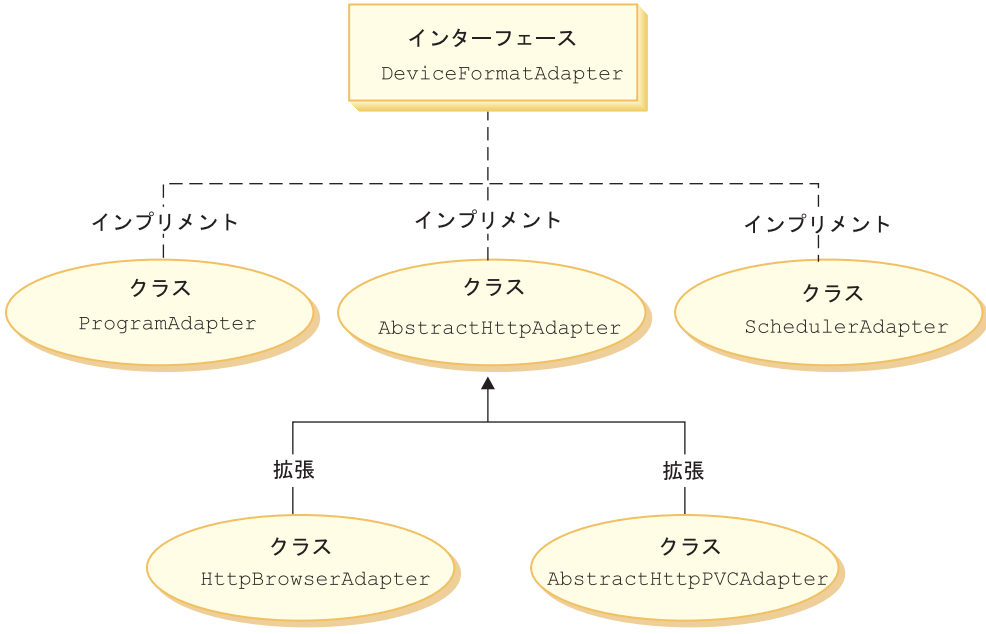


図 4.

上記の図のとおり、すべてのアダプターが DeviceFormatAdapter インターフェースをインプリメントします。WebSphere Commerce ランタイム環境で使用されるアダプターは、以下のとおりです。

プログラム・アダプター

プログラム・アダプターは、リモート・プログラムが WebSphere Commerce コマンドを実行するためのサポートを提供します。プログラム・アダプターが要求を受け取り、メッセージ・マッパーを使用して要求を CommandProperty オブジェクトに変換します。変換後、プログラム・アダプターは CommandProperty オブジェクトを使用して要求を実行します。

スケジューラー・アダプター

スケジューラー・アダプターは、バックグラウンド・ジョブとして実行される WebSphere Commerce コマンドのためのサポートを提供します。

HTTP ブラウザー・アダプター

HTTP ブラウザー・アダプターは、 HTTP ブラウザーから受け取る、WebSphere Commerce コマンド呼び出し要求のためのサポートを提供します。

HTTP PvC アダプター

これは、特定の PvC 装置アダプターを開発するために使用することのできる、抽象アダプター・クラスです。たとえば、特定のセル電話アプリケーション用のアダプターを開発する必要がある場合は、このアダプターを拡張します。

アダプターのフレームワークは、必要に応じて、以下の 2 つの方法で拡張できます。

- 特定の PvC 装置用のアダプターを作成する (たとえば、 HttpIModePVCAdapterImpl クラスを作成すると i モード装置用のサポートが提供されます)。このタイプのアダプターは、AbstractHttpAdapterImpl クラスを拡張する必要があります。
- 新しいプロトコル・リスナーに接続する、新しいアダプターを作成する。この新しいアダプターは、 DeviceFormatAdapter インターフェースをインプリメントしなければなりません。

Web コントローラー

WebSphere Commerce Web コントローラーは、EJB コンテナーと類似したデザイン・パターンに従う、アプリケーション・コンテナーです。このコンテナーは、セッション管理 (アダプターが確立したセッション持続性に基づく)、トランザクション制御、アクセス制御、および認証などのサービスを提供することにより、コマンドの役割を単純化します。

Web コントローラーは、コマース・アプリケーション用のプログラミング・モデルを守らせる上でも、役割を果たします。たとえば、プログラミング・モデルは、アプリケーションが作成すべきコマンドのタイプを定義します。コマンドのタイプは、それぞれ特有の目的を果たします。ビジネス・ロジックはコントローラー・コマンドによってインプリメントされなければならず、ビュー・ロジックはビュー・コマンドによってインプリメントされなければなりません。 Web コントローラーは、コントローラー・コマンドがビュー名を戻すものと予期します。ビュー名が戻されないと、例外がスローされます。

HTTP 要求の場合、Web コントローラーは以下のタスクを実行します。

- javax.transaction パッケージの UserTransaction インターフェースを使用して、トランザクションを開始する。
- セッション・データをアダプターから取得する。
- コマンドの呼び出しの前に、ユーザーのログオンが必要かどうかを決定する。必要な場合は、ユーザーのブラウザーをログオン URL にリダイレクトします。
- URL にセキュア HTTPS が必要かどうかを検査する。必要な場合に、現在の要求が HTTPS を使用していなければ、Web ブラウザーを HTTPS URL にリダイレクトします。
- コントローラー・コマンドを呼び出し、コマンド・コンテキストと入力プロパティー・オブジェクトを渡す。
- トランザクション・ロールバック例外が発生し、コントローラー・コマンドが再試行可能である場合に、コントローラー・コマンドを再試行する。
- クライアントに戻すべきビュー・コマンドがある場合、コントローラーは普通、ビュー名を戻す。 Web コントローラーは、対応するビューのためにビュー・コマンドを呼び出します。応答ビューを形成する方法はいくつかあります。その中には、別の

URL へリダイレクトする方法や、JSP テンプレートへ転送する方法、あるいは応答オブジェクトに対する HTML 文書を作成する方法があります。

- セッション・データを保管する。
- セッション・データをコミットする。
- 現行のトランザクションが成功した場合にコミットする。
- 現行のトランザクションが失敗した場合にロールバックする (状況による)。

コマンド

WebSphere Commerce コマンドは、特定の要求の処理に関連したプログラミング・ロジックを含む bean です。

WebSphere Commerce コマンドには 4 つの主なタイプがあります。

コントローラー・コマンド

コントローラー・コマンドは、特定のビジネス処理に関係するロジックをカプセル化します。コントローラー・コマンドの例としては、注文処理用の *OrderProcessCmd* コマンドや、新規登録ユーザー作成用の *UserRegistrationAddCmd* コマンドがあります。一般に、コントローラー・コマンドには制御ステートメント (if、then、else など) が含まれており、ビジネス処理の中で個々のタスクを実行するためにタスク・コマンドを呼び出します。完了すると、コントローラー・コマンドはビュー名を戻します。そして、Web コントローラーは、ビュー・コマンド用の適切なインプリメンテーション・クラスを判別して、ビュー・コマンドを実行します。

タスク・コマンド

タスク・コマンドは特定のアプリケーション・ロジックの単位をインプリメントします。一般に、コントローラー・コマンドと一式のタスク・コマンドが一緒になって、 URL 要求のためのアプリケーション・ロジックをインプリメントします。タスク・コマンドはコントローラー・コマンドと同じコンテナーで実行されます。

データ Bean コマンド

データ Bean コマンドは、データ Bean がインスタンス化されるときに JSP テンプレートによって呼び出されます。データ Bean コマンドの主な機能は、データ Bean にデータを取り込むことです。

ビュー・コマンド

ビュー・コマンドは、クライアント要求への応答としてビューを構成します。ビュー・コマンドには 3 つのタイプがあります。

Redirect View コマンド

このビュー・コマンドは、 URL リダイレクトなどのリダイレクト・プロトコルを使用してビューを送信します。コントローラー・コマンドは、リダイレクト・プロトコルを使用してビューを戻すために、このビュー・タイプのビュー・コマンドを戻す必要があります。リダイ

レクト・プロトコルが使用されると、それはブラウザー内の URL スタックを変更します。再ロード鍵が入力されると、元の URL の代わりに、リダイレクトされた URL が実行されます。

Direct View コマンド
このビュー・コマンドは、クライアントに直接、応答ビューを送信します。

Forward View コマンド
このビュー・コマンドは他の Web コンポーネント (JSP テンプレートなど) にビュー要求を転送します。

ビュー・コマンドを呼び出すことのできる 3 つの方法があります。

- 要求が正常完了したときに、コントローラー・コマンドでビュー・コマンド名を指定する。
- クライアントが直接にビューを要求する。
- コマンドがエラーを検出し、エラーを処理するにはエラー・タスクを実行しなければならない場合。コマンドは、ビュー・コマンド名を伴って例外をスローします。例外が Web コントローラーに伝搬されると、エラー・ビュー・コマンドが実行され、クライアントに応答が戻されます。

WebSphere Commerce エンティティー Bean

エンティティー Bean は、 WebSphere Commerce が提供する、永続的なトランザクションのコマース・オブジェクトです。コマースの分野に精通した方にとって、エンティティー Bean は WebSphere Commerce データを直観的に表現したものです。つまり、データベース・スキーマ全体を理解しなくても、コマースの分野の概念とオブジェクトをより綿密にモデル化したエンティティー Bean からデータにアクセスできます。既存のエンティティー Bean を拡張することができます。さらに、ユーザーのアプリケーション固有のビジネス要件に合わせて、完全に新しいエンティティー Bean のデプロイメントを実行することもできます。

エンティティー Bean は、Enterprise JavaBeans (EJB) コンポーネント・モデルに合わせてインプリメントされています。

エンティティー Bean の詳細については、49 ページの 『WebSphere Commerce エンティティー Bean のインプリメンテーション』を参照してください。

データ Beans

データ Bean は、主に Web 設計者が使用する Java bean です。一般的に、WebSphere Commerce エンティティーへのアクセスを提供します。 Web 設計者はこれらの bean を JSP テンプレート上に配置することができ、表示の際にページ上に動的な情報を取り込むことが可能になります。 Web 設計者は、bean が提供できるデータは何か、また入

力として bean が必要とするデータは何かを理解するだけでよいのです。ビジネス・ロジックから表示を分離するという課題と一致して、 Web 設計者が bean の働きを理解しなくてもよいようになっています。

データ Bean マネージャー

WebSphere Studio Page Designer を使用して WebSphere Commerce データ Bean を JSP テンプレートに挿入すると、ランタイムにデータ Bean マネージャーを呼び出してデータ Bean にデータを取り込むコードの行が生成されます。

Page Designer からのサンプルのコードを以下に示します。

```
com.ibm.commerce.beans.DataBeanManager.activate(data_bean, request)
```

JavaServer Pages テンプレート

JSP テンプレートは、通常は表示目的で使用される、特化されたサーブレットです。URL 要求が完了すると、Web コントローラーはビュー・コマンドを呼び出し、そのコマンドが JSP テンプレートを呼び出すことになります。クライアントは、関連したコマンドを使わずに、ブラウザーから直接 JSP テンプレートを呼び出すこともできます。この場合、JSP テンプレートの URL は、要求サーブレットをそのパスに含んでいなければなりません。そうすることで、JSP テンプレートが必要とするすべてのデータ Bean を単一トランザクション内で活動化することができます。要求サーブレットは URL 要求を JSP テンプレートへ転送し、単一トランザクション内で JSP テンプレートを実行することができます。

データ Bean マネージャーは、パスに要求サーブレットが含まれていない JSP テンプレートの URL をすべて拒否します。 JSP テンプレートと他のリソースの保護の詳細については、 91 ページの『第 4 章 アクセス制御』を参照してください。

Instance_name.xml 構成ファイル

Instance_name.xml 構成ファイルは、インスタンスの構成情報を設定します。このファイルは、要求サーブレットが初期化されるときに読み取られます。

要求の要約

このセクションでは、要求に対する応答を形成するときのコンポーネント間の相互作用フローを要約します。

各ステップの説明は、図に従っています。

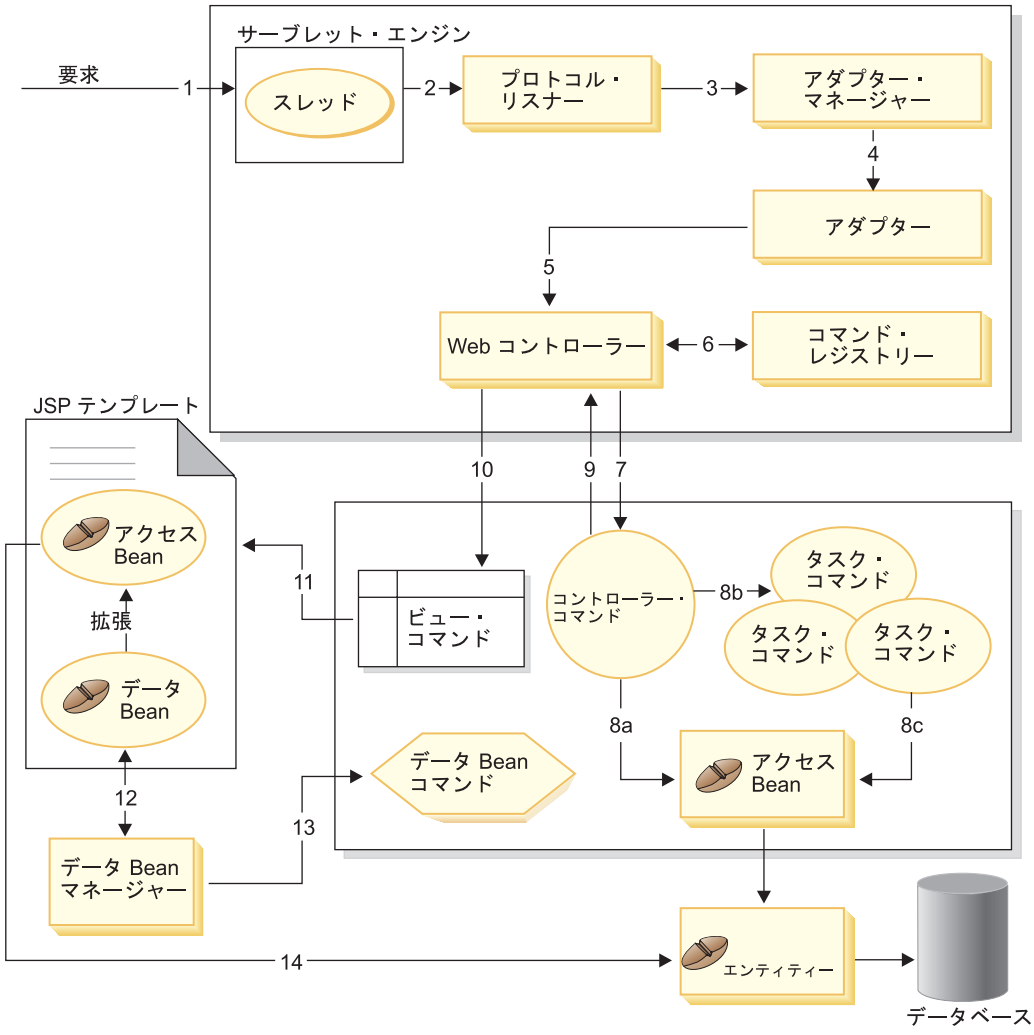


図 5.

以下の情報は、上記の図に対応しています。

1. 要求が、WebSphere Application Server プラグインによって、サーブレット・エンジンに送られます。
2. 要求は独自のスレッドで実行されます。サーブレット・エンジンは、要求をプロトコル・リスナーにディスパッチします。プロトコル・リスナーは HTTP 要求サーブレットと MQ リスナーのどちらでもかまいません。
3. プロトコル・リスナーは、要求をアダプター・マネージャーに渡します。

4. アダプター・マネージャーは、要求を処理できるアダプターを判別し、適切なアダプターに要求を転送します。たとえば、インターネット・ブラウザーからの要求であれば、アダプター・マネージャーが要求を HTTP ブラウザー・アダプターに転送します。
5. アダプターは、要求を Web コントローラーに渡します。
6. Web コントローラーは、コマンド・レジストリーにクエリーすることにより、どのコマンドを呼び出すかを判別します。
7. 要求が、コントローラー・コマンドの使用を必要としているとします。この場合、Web コントローラーは、適切なコントローラー・コマンドを呼び出します。
8. コントローラー・コマンドがいったん実行を開始した後、たどる可能性のあるパスがいくつかあります。
   a. コントローラー・コマンドは、アクセス Bean とそれに対応するエンティティー Bean を使用して、データベースにアクセスすることができます。
   b. コントローラー・コマンドは、1 つ以上のタスク・コマンドを呼び出すことができます。それから、タスク・コマンドは、アクセス Bean とそれに対応するエンティティー Bean を使用して、データベースにアクセスすることができます (8(c) に示されています)。
9. 完了すると、コントローラー・コマンドは Web コントローラーにビュー名を戻します。
10. Web コントローラーは、VIEWREG テーブルでビュー名を検索します。そして、リクエスターの装置タイプに登録されている、ビュー・コマンドのインプリメンテーションを呼び出します。
11. ビュー・コマンドは、要求を JSP テンプレートに転送します。
12. JSP テンプレートの中では、データベースから動的な情報を検索するために、データ Bean が必要とされます。データ Bean マネージャーがデータ Bean をアクティブにします。
13. 必要であれば、データ Bean マネージャーがデータ Bean コマンドを呼び出します。
14. データ Bean の拡張元であるアクセス Bean が、それに対応するエンティティー Bean を使用して、データベースにアクセスします。

カスタマイズに関する、前のリリースとの主な相違点

WebSphere Commerce Suite バージョン 5.1 以降では、 WebSphere Commerce Server はすべて Java で作成されています。したがって、機能をカスタマイズするには Java を使用する必要があります。これは、WebSphere Commerce Suite、バージョン 4.1 (および Net.Commerce™ の初期のバージョン) で使用されていたモデルとは非常に異なっています。それまでは、カスタマイズに C++ および Net.Data® マクロが使用されていました。

以下の表は、主な変更点を要約したものです。

	C++ で作成された バージョン 4.1 (およびそれ以前)	**Java で作成された バージョン 5.1 (およびそれ以降のバージョン)**
アプリケーション・モデル	コマンド	コントローラー・コマンド
	タスク	タスク・コマンド
	指定変更可能機能	タスク・コマンド
	ビュー・タスク	ビュー・コマンド
	エラー・タスク	ビュー・コマンド
表示モデル	Net.Data マクロ	JSP テンプレート、データ Bean、データ Bean コマンド、およびビュー・コマンド
永続モデル	Net.Data および ODBC	エンティティー Bean
URL ディスパッチャー	WebSphere Commerce C++ URL ディスパッチャー	WebSphere サーブレット・エンジン
タスク・モデル	システム・プロセス	Java スレッド
コマンド・アダプター	なし	Web コントローラーおよびアダプター

本書では、マイグレーション作業については説明していません。マイグレーションについての詳細は、*WebSphere Commerce* マイグレーション・ガイド を参照してください。

第 2 部 プログラミング・モデル

第 2 章 デザイン・パターン

WebSphere Commerce フレームワークを開発するには、さまざまなデザイン・パターンと機構が使用されます。WebSphere Commerce には、各 WebSphere Commerce アプリケーションが従うべき高レベルのデザイン・パターンが用意されています。この章では、以下のデザイン・パターンについて説明します。

- モデル・ビュー・コントローラー・デザイン・パターン
- コマンド・デザイン・パターン
- 表示デザイン・パターン

モデル・ビュー・コントローラー・デザイン・パターン

モデル・ビュー・コントローラー (MVC) デザイン・パターンの仕様では、アプリケーションは、データ・モデル、プレゼンテーション情報、および制御情報から構成されます。このパターンでは、上記の各要素がそれぞれ別個のオブジェクトに分離されることが必要です。

モデル (データ情報など) には、純粋なアプリケーション・データだけが含まれます。どのようにデータをユーザーに提示するかを記述したロジックは、まったく含まれません。

ビュー (プレゼンテーション情報など) は、モデルのデータをユーザーに提示します。ビューは、モデルのデータにアクセスする方法は知っていますが、このデータが何を意味するかということや、どうすればユーザーはこれを操作できるかということは知りません。

最後のコントローラー (制御情報など) は、ビューとモデルの間に存在しています。コントローラーは、ビュー (または別の外部ソース) が起動するイベントを listen し、それらのイベントに対して適切な反応を実行します。たいていの場合、反応は、モデルのメソッドを呼び出すことです。ビューとモデルは通知機構で接続されているため、このアクションの結果は自動的にビューに反映されます。

最近のアプリケーションの大部分がこのパターンに従っているものの、その多くは若干の変化を伴っています。たとえば、あるアプリケーションでは、ビューとコントローラーがかねてから緊密に連結しているため、これらが 1 つのクラスに組み合わされています。こうした変化はすべて、データとデータの表示を分離するよう、強く促すものとなります。これにより、アプリケーションの構造が単純になるだけでなく、コードの再利用も可能になります。

このパターンについて説明した資料やサンプルが多数存在するので、本書ではあまり詳しく説明しません。

以下の図では、 MVC デザイン・パターンがどのように WebSphere Commerce に適用されているかが示されています。

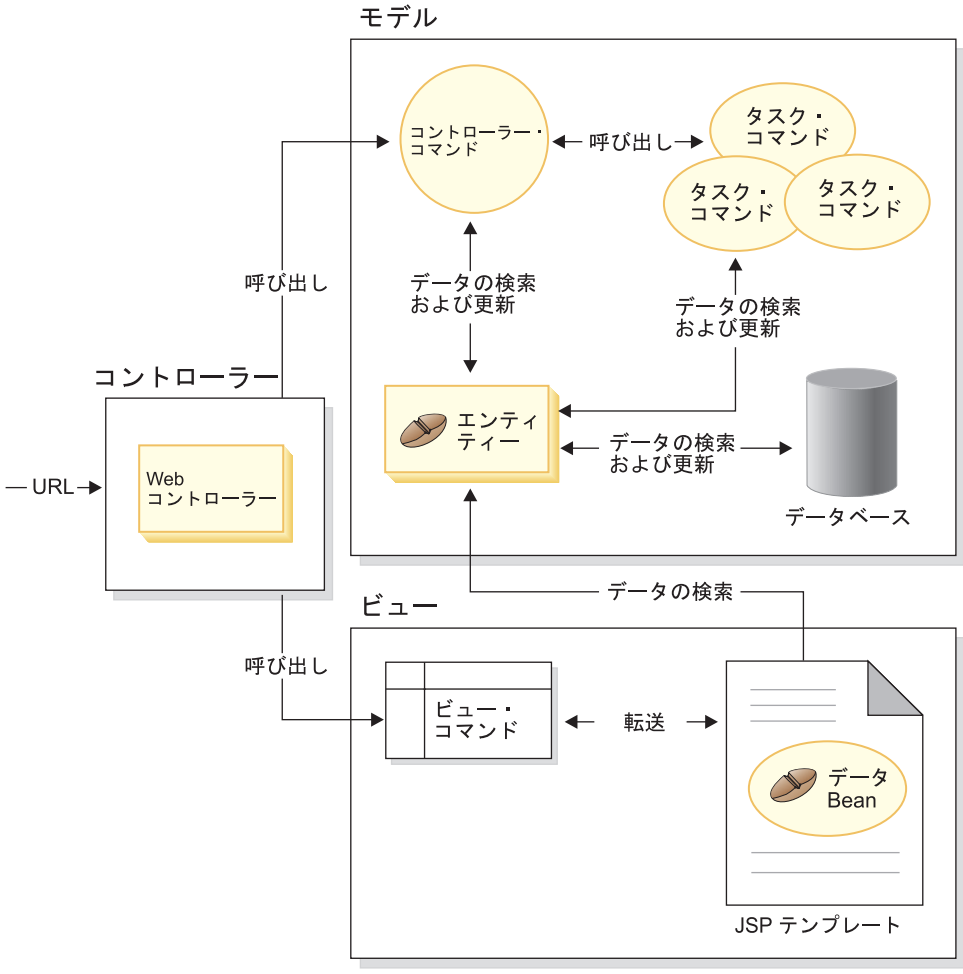


図 6.

コマンド・デザイン・パターン

WebSphere Commerce Server は、他のリモート・アプリケーションからの要求と同様に、ブラウザー・ベースのシン・クライアント・アプリケーションからの要求を受け入れます。たとえば、リモート調達システムからの要求の場合も、別のコマース・サーバーからの要求の場合もあります。

さまざまなフォーマットによる要求はすべて、アダプター・フレームワークを構成するアダプターによって共通フォーマットに変換されます。要求が共通フォーマットになれば、 WebSphere Commerce コマンドがこれを理解できるようになります。

コマンドは、ビジネス・ロジックを実行する bean です。コマンドは、手続き型のロジックを、高水準の処理ロジック・タスク、または離散的なビジネス・ロジック・タスクのいずれかの形で表したものです。処理ベースのコマンドが、複数のエンティティーと他のコマンドの橋渡しをするコントローラーの役割を果たす一方で、タスク・コマンドは特定のタスクを実行し、単一のオブジェクトにだけアクセスできます。

コマンド・フレームワーク

コマンド bean は特定のデザイン・パターンに従います。すべてのコマンドは、インターフェース・クラス (`CategoryDisplayCmd` など) とインプリメンテーション・クラス (`CategoryDisplayCmdImpl` など) の両方を含みます。呼び出し側の見地からすると、呼び出しのロジックには、入力プロパティーの設定、`execute()` メソッドの呼び出し、および出力プロパティーの検索が含まれます。

コマンドをインプリメントする側から見ると、コマンドは WebSphere のコマンド・フレームワークに従います。このフレームワークは、標準のコマンド・デザイン・パターンをインプリメントし、呼び出し側とインプリメンテーションの間の間接性をあるレベルまで可能にします。このレベルの間接性で使用できる主な機構には、以下のものがあります。

1. ユーザーがコマンドの呼び出しを許可されているかどうかを判別する、アクセス制御ポリシー・マネージャーを呼び出す能力。
2. ストアの ID に基づいて、異なるストア向けの異なるコマンドのインプリメンテーションを実行する能力。
3. リクエスターの装置タイプに基づいて、異なるビュー・インプリメンテーションを実行する能力。

以下の図は、主な 4 種類のコマンドのインターフェースの概念的な概要を示したものです。

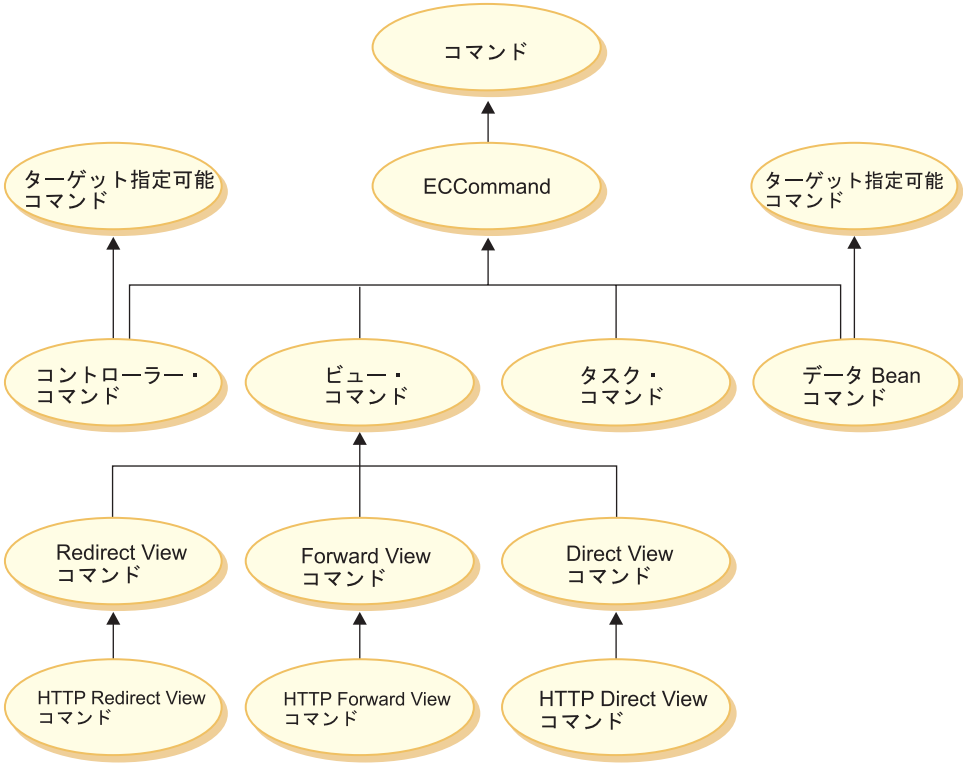


図 7.

コントローラー・コマンド

コントローラー・コマンドは、特定のビジネス処理に関係するロジックをカプセル化します。コントローラー・コマンドの例としては、注文処理用の *OrderProcessCmd* コマンドや、新規登録ユーザー作成用の *UserRegistrationAddCmd* コマンドがあります。一般に、コントローラー・コマンドには制御ステートメント (if、then、else など) が含まれており、ビジネス処理の中で個々のタスクを実行するためにタスク・コマンドを呼び出します。完了すると、コントローラー・コマンドはビュー名を戻します。そして、Web コントローラーは、ビュー・タスク用の適切なインプリメンテーション・クラスを判別して、ビュー・タスクを実行します。

コントローラー・コマンドはターゲット指定可能コマンドですが、ローカルなターゲットしかサポートしません。

タスク・コマンド

タスク・コマンドは特定のアプリケーション・ロジックの単位をインプリメントします。一般に、コントローラー・コマンドと一式のタスク・コマンドが一

緒になって、 URL 要求のためのアプリケーション・ロジックをインプリメントします。タスク・コマンドはコントローラー・コマンドと同じコンテナーで実行されます。

データ Bean コマンド

データ Bean コマンドは、データ Bean がインスタンス化されるときに JSP ページによって呼び出されます。データ Bean コマンドの主な機能は、データ Bean のフィールドにデータを取り込むことです。

データ Bean コマンドはターゲット指定可能コマンドですが、ローカルなターゲットしかサポートしません。

ビュー・コマンド

ビュー・コマンドは、クライアント要求への応答としてビューを構成します。ビュー・コマンドを呼び出すことのできる 3 つの方法があります。

- 要求の正常終了時に、コントローラー・コマンドでビュー・コマンド名を指定する方法。
- クライアントから直接にビューを要求する方法。
- コントローラー・コマンドまたはタスク・コマンドがエラーを検出し、そのエラーを処理するためにエラー・タスクを実行する必要があると判断し、ビュー・コマンド名を伴って例外をスローするという方法。例外が Web コントローラーに伝搬されると、ビュー・コマンドが実行され、クライアントに応答が戻されます。

ビュー・コマンドには 3 つのタイプがあります。

Redirect View コマンド

このビュー・コマンドは、 URL リダイレクトなどのリダイレクト・プロトコルを使用してビューを送信します。コントローラー・コマンドは、リダイレクト・プロトコルが必要な場合に、このビュー・タイプのビュー・コマンドを戻す必要があります。リダイレクト・プロトコルが使用されると、それはブラウザー内の URL スタックを変更します。再ロード鍵が入力されると、元の URL の代わりに、リダイレクトされた URL が実行されます。

Direct View コマンド

このビュー・コマンドは、クライアントに直接、応答ビューを送信します。

Forward View コマンド

このビュー・コマンドは他の Web コンポーネント (JSP テンプレートなど) にビュー要求を転送します。

コマンド・ファクトリー

新しいコマンド・オブジェクトを作成するために、コマンドの呼び出し側はコマンド・ファクトリー を使用します。コマンド・ファクトリーは、コマンドをインスタンス化するために使用される bean です。コマンド・ファクトリーは、ファクトリー・デザイン・パターンに従っています。このデザイン・パターンは、呼び出し元のクラスから離れたオブジェクトのインスタンス化を、ファクトリー・クラスにゆだねます。このクラスは、どのインプリメンテーション・クラスをインスタンス化するかを理解しています。

ファクトリーは、新規オブジェクトをインスタンス化するためのスマートな方法を提供します。この場合、コマンド・ファクトリーは、新しいコマンド・オブジェクトを個々のストアに基づいて作成するときに、正しいインプリメンテーション・クラスを判別するための方法を提供します。コマンド・インターフェース名と、特定のストアの ID が、インスタンス化の時点で新規コマンド・オブジェクトに渡されます。

コマンドのインプリメンテーション・クラスを指定する方法は 2 つあります。デフォルトのインプリメンテーション・クラスは、 `defaultCommandClassName` 変数を使用して、コマンド・インターフェースのコードの中に直接に指定することができます。たとえば、 CategoryDisplayCmd インターフェースには以下のコードが存在します。

```
String defaultCommandClassName =
    "com.ibm.commerce.catalog.commands.CategoryDisplayCmdImpl"
```

インプリメンテーション・クラスを指定する 2 番目の方法は、 WebSphere Commerce コマンド・レジストリーを使用するというものです。インプリメンテーション・クラスがストアごとに異なる場合は、必ずコマンド・レジストリーを使用するべきです。コマンド・レジストリーの詳細については、 28ページに記載されています。

デフォルトのインプリメンテーション・クラスがインターフェースのコードに指定されており、コマンド・レジストリーには異なるインプリメンテーション・クラスが指定されている場合、コマンド・レジストリーが優先されます。

コマンド・ファクトリーを使用するための構文は、以下のとおりです。

```
cmd = CommandFactory.createCommand(interfaceName, commandContext.getStoreId())
```

ここで、*interfaceName* は新規コマンド bean のインターフェース名です。 getStoreId メソッドは、コマンドが使用されることになるストアを決定します。

注: コマンド・ファクトリーを使用してビジネス・ポリシー・コマンドを作成するための構文は、ここまでのコードの断片とは異なります。コマンド・ファクトリーを使用するビジネス・ポリシー・コマンドの作成については、 180 ページの 『新規のビジネス・ポリシーの呼び出し』を参照してください。

コマンドのフロー

このセクションでは、コマンドと WebSphere Commerce データベースの間の論理フローの概要を示します。以下の図と説明は、このフローを表現したものです。

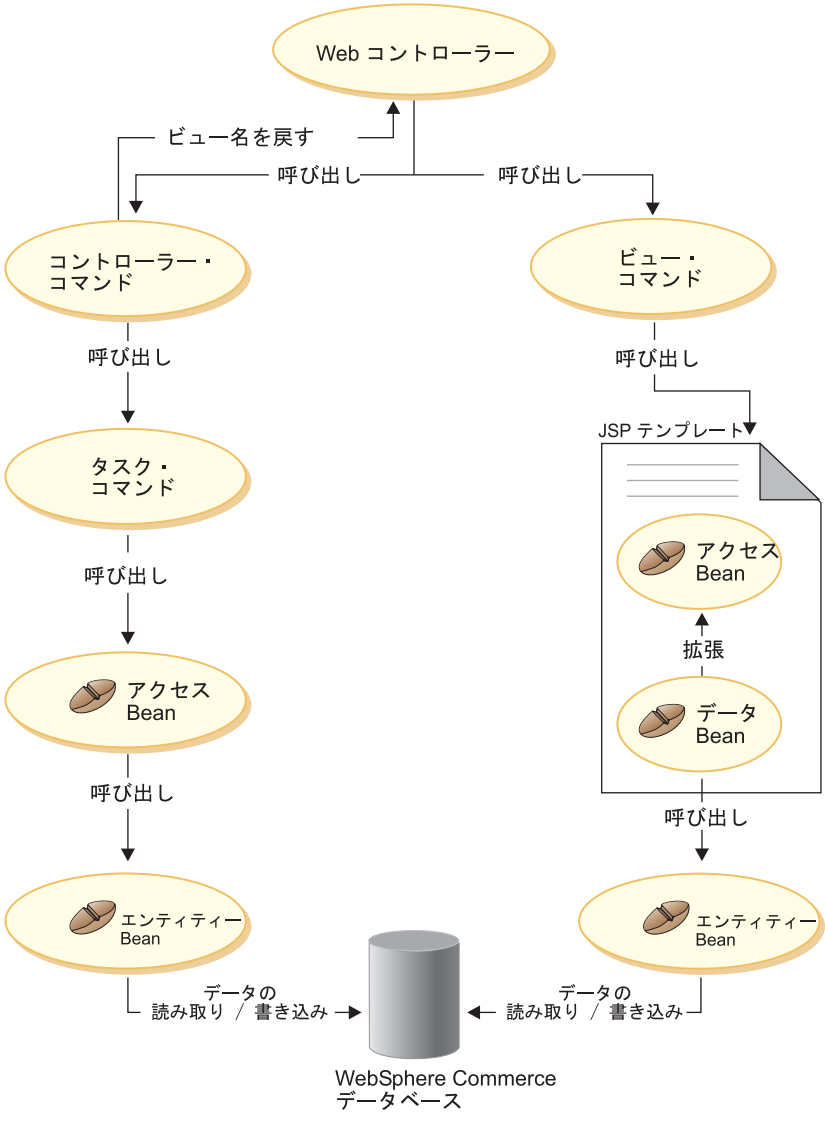


図 8.

Web コントローラーは、要求を受け取ると、この要求にはコントローラー・コマンドとビュー・コマンドのどちらを呼び出すことが必要かを判別します。どちらの場合でも、Web コントローラーはコマンドのインプリメンテーション・クラスも判別し、コマンドを呼び出します。

まず、図の左側を調べてみましょう。コントローラー・コマンドは、ビジネス処理のロジックをカプセル化しているので、ビジネス処理の中の特定の作業単位を実行するため、個々のタスク・コマンドを頻繁に呼び出します。データベースの情報の検索または更新が必要な場合、アクセス Bean が呼び出されます。タスク・コマンドもコントローラー・コマンドも、アクセス Bean を呼び出すことができます。要求は、アクセス Bean からエンティティー Bean にフローしていきます。エンティティー Bean は、WebSphere Commerce データベースを読み書きすることができます。

今度は、図の右側を調べてみましょう。コントローラー・コマンドが処理を完了して、呼び出すべきビュー・コマンドの名前を戻した場合、またはエラーが発生して、エラー・ビューを表示しなければならなくなった場合のいずれかに、ビュー・コマンドが Web コントローラーによって呼び出されます。

ビュー・コマンドは、クライアントに応答を表示するために、 JSP テンプレートを呼び出すのが一般的です。 JSP テンプレートの中では、ページに動的な情報を取り込むためにデータ Bean が使用されます。データ Bean はデータ Bean マネージャーによってアクティブにされます。データ Bean (アクセス Bean から拡張されたもの) は、それに対応するエンティティー Bean を呼び出します。 JSP テンプレートから間接的にアクセスされると、エンティティー Bean は (データベースに情報を書き込むよりも) データベースから情報を検索するのが普通です。

コマンド登録フレームワーク

WebSphere Commerce コントローラー・コマンドおよびタスク・コマンドは、コマンド・レジストリーに登録されます。コマンド・レジストリーは以下の 3 つのテーブルから構成されます。

- URLREG
- CMDREG
- VIEWREG

注: Business このセクションは、ビジネス・ポリシー・コマンドの登録には適用されません。新規のビジネス・ポリシー・コマンドの登録については、 164 ページの『新規のビジネス・ポリシーとビジネス・ポリシー・コマンドの登録』を参照してください。

URLREG テーブル

URLREG テーブルは、 URI (Universal Resource Indicator) をコントローラー・コマンド・インターフェースにマップします。 URI は、リソース識別のための、単純で拡張

可能な機構を提供します。 URI は、抽象リソースまたは物理リソースを識別するために使用される、比較的に短い文字ストリングです。 WebSphere Commerce の場合、URI にはコマンド情報だけが含まれます。以下の URL の、太字で示した部分が URI です。

```
http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
     storeId=store_Id&catalogId=catalog_Id&langId=-1
```

URI とインターフェース名の間には 1 対 1 のマッピングが存在しますが、各ストアは、コマンドに必要なのは HTTPS と AUTHENTICATION のどちらであるかを指定することができます。インバウンド URL 要求を受け取るたびに、 Web コントローラーはコントローラー・コマンドのインターフェース名を検索します。そして、その名前を使用して、CMDREG テーブルに登録されているとおりに、正しいインプリメンテーション・クラスを判別します。

URLREG データベース・テーブルに含まれる情報を、以下の表で説明します。

列名	説明	注釈
URL	URI 名	`MyNewCommand` または `ProductDisplay` など。
STOREENT_ID	ストアのエンティティー ID	すべてのストアで使用するコマンドの場合は `0`、特定のストアでしか使用しないコマンドの場合は固有のストア ID に設定できます。
INTERFACENAME	コントローラー・コマンド・インターフェース名	たとえば、`com.ibm.commerce.catalog.commands.GetProductDisplay.TemplateCmd` など。
HTTPS	この URL 要求にセキュア HTTP が必要かどうか	HTTPS が必要な場合は `1`、不要な場合は `0` を使用します。
DESCRIPTION	URI の説明	`This command is used for testing purposes` など。
AUTHENTICATED	この URL 要求にユーザー・ログオンが必要かどうか	認証が必要な場合は `1`、不要な場合は `0` を使用します。
INTERNAL	コマンドが WebSphere Commerce の内部的なものかどうか	コマンドが内部的なものであれば `1`、外部的なものであれば `0` を使用します。

Web コントローラーは、URL 要求を受け取ると、要求されたコントローラー・コマンドのインターフェース名を検索します。次いで、その名前を使用して、CMDREG テー

ブルからインプリメンテーション・クラス名を検索します。また、URLREG テーブルの HTTPS 列を検査して、この URL 要求に HTTPS が必要かどうかも判別します。

URLREG テーブルに登録する必要があるのは、 URL 要求を介して呼び出されるコマンドだけです。したがって、ここに登録されるのはコントローラー・コマンドだけです。タスク・コマンドやビュー・コマンドは登録されません。

以下の SQL ステートメントは、MyNewControllerCommand のエントリーを作成します。このコマンドは特定のストアで使用されます (ストア ID は 5)。

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCommand',
5,'com.ibm.commerce.commands.MyNewControllerCommand',0,
'This is a test command.',null)
```

挿入ステートメントの汎用的な構文は、以下のとおりです。

```
insert into table_name (column_name1,column_name2, ... ,column_namen)
values (column1_value,column2_value,...,columnn_value)
```

ストリング値は、単一引用符で囲んでください。

CMDREG テーブル

CMDREG はコマンド登録テーブルです。このテーブルは、コマンド・インターフェースをインプリメンテーション・クラスにマッピングする機構を提供します。 1 つのインターフェースがインプリメンテーションを複数持つことができるので、ストアごとにコマンドのカスタマイズを行うことが可能です。

CMDREG テーブルに登録されるのは、コントローラー・コマンドとタスク・コマンドだけです。ビュー・コマンドは VIEWREG テーブルに登録されます。

CMDREG データベース・テーブルに含まれる情報を以下に説明します。

列名	説明	注釈
STOREENT_ID	ストアのエンティティー ID	すべてのストアで使用するコマンドの場合は 0、特定のストアでしか使用しないコマンドの場合は固有のストア ID に設定できます。
INTERFACENAME	コマンド・インターフェース名	インターフェースを定義します。 URLREG テーブルで使用したのと同じ名前を使用してください。
DESCRIPTION	コマンドの説明	`This command is used for testing purposes` など。

列名	説明	注釈
CLASSNAME	コマンド・インプリメンテーション・クラス名	インターフェース名の末尾に ″Impl″ を追加するのが一般的です。
PROPERTIES	コマンドへの入力プロパティーとして設定する、名前と値のデフォルトの対	形式は URL クエリー・ストリングと同じです。たとえば、 ″parm1=val1&parm2=val2″ など。
LASTUPDATE	このコマンド・エントリーの最終更新日付	
TARGET	コマンドのターゲット名。コマンドが実際に実行される場所です。	ローカル・ターゲットしかサポートされていません。

一般に、新しくコントローラー・コマンドまたはタスク・コマンドを作成したら、それに対応するエントリーを CMDREG テーブルに作成する必要があります。たとえば、以下の SQL ステートメントは MyNewCommand のエントリーを作成します。このコマンドは特定のストアで使用されます (ストア ID は 5)。

```
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION, CLASSNAME,
PROPERTIES, LASTUPDATE, TARGET) values
(5,'com.ibm.commerce.catalog.commands.MyNewCommand', 'This is a test command',
'com.ibm.commerce.catalog.commands.MyNewCommandImpl',
'myDefaulParm1=myDefaultVal1', '0000-12-01', 'Local')
```

挿入ステートメントの汎用的な構文は、以下のとおりです。

```
 insert into table_name (column_name1,column_name2, ... ,column_namen)
values (column1_value,column2_value,...,columnn_value)
```

ストリング値は、単一引用符で囲んでください。

作成するコマンドがいつも同じインプリメンテーション・クラスを使用するのであれば、コマンドを CMDREG テーブルに登録する必要はありません。この場合は、インターフェースの中で defaultCommandClassName 属性を使用して、インプリメンテーション・クラスを指定することができます。たとえば、インターフェースのコードに以下の内容を含めることができます。

```
String defaultCommandClassName =
     "com.ibm.commerce.command.MyNewCommandImpl"
```

この方法でインプリメンテーション・クラスを指定した場合、インプリメンテーション・クラスにデフォルトのプロパティーを渡すことはできず、すべてのストアで同じインプリメンテーション・クラスを使用しなければなりません。

登録済みのコントローラー・コマンドの例

サイトに 2 つのストア (StoreA と StoreB) があるというシナリオを考えてみましょう。ストアごとに、MyUrl コントローラー・コマンドのセキュリティー要件と、コマンドのインプリメンテーションが異なるものとします。ここでは、このカスタマイズを可能にするために、コマンド・レジストリーをどのように使用するかを示します。

URLREG テーブルの StoreA と StoreB のエントリーを、以下の表に示します。

列名	StoreA のエントリー	StoreB のエントリー
URL	MyUrl	MyUrl
STOREENT_ID	11	22
INTERFACENAME	com.ibm.commerce. mycommands.myUrl	com.ibm.commerce. mycommands.myUrl
HTTPS	1	1
DESCRIPTION	URLREG テーブルのエントリー例	URLREG テーブルのエントリー例
AUTHENTICATED	1	0
INTERNAL	ヌル	ヌル

注: INTERFACENAME の値にスペースが含まれているのは、表示上の都合に過ぎません。実際には、それぞれの値は連続したストリングです。

Web コントローラーは、URLREG テーブルのエントリーに基づき、 MyUrl のインターフェース名が com.ibm.commerce.mycommands.MyUrl であると判別します。また、StoreA では HTTPS と認証の両方を使用してコマンドを実行する必要がある一方、StoreB では HTTPS だけが必要であるということも判別します。 HTTPS の値と認証の値は、Web コントローラーによって使用されます。インターフェースによって使用されることはありません。

以下の図は、このフローを示しています。

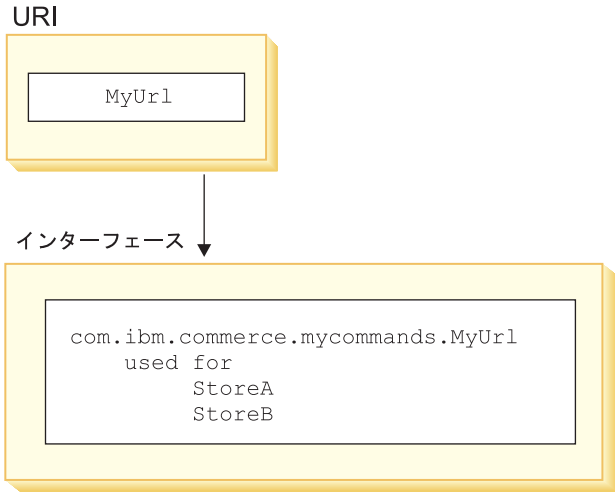


図 9.

以下の表は、CMDREG テーブルのエントリーを示しています。この例の目的に必要な列だけを示します。

列名	StoreA のエントリー	StoreB のエントリー
STOREENT_ID	11	22
INTERFACENAME	com.ibm.commerce. mycommands.myUrl	com.ibm.commerce. mycommands.myUrl
CLASSNAME	com.ibm.commerce. mycommands. myUrlStoreAImpl	com.ibm.commerce. mycommands.myUrlStoreBImpl

注: INTERFACENAME と CLASSNAME の値にスペースが含まれているのは、表示上の都合に過ぎません。実際には、それぞれの値は連続したストリングです。

Web コントローラーは、CMDREG テーブルのエントリーに基づき、StoreA では、com.ibm.commerce.mycommands.MyUrl インターフェースのインプリメンテーション・クラスが com.ibm.commerce.mycommands.MyUrlStoreAImpl であると判別します。同様に、StoreB では、同じインターフェースのインプリメンテーション・クラスが com.ibm.commerce.mycommands.MyUrlStoreBImpl であることを判別します。以下の図は、このフローを示しています。

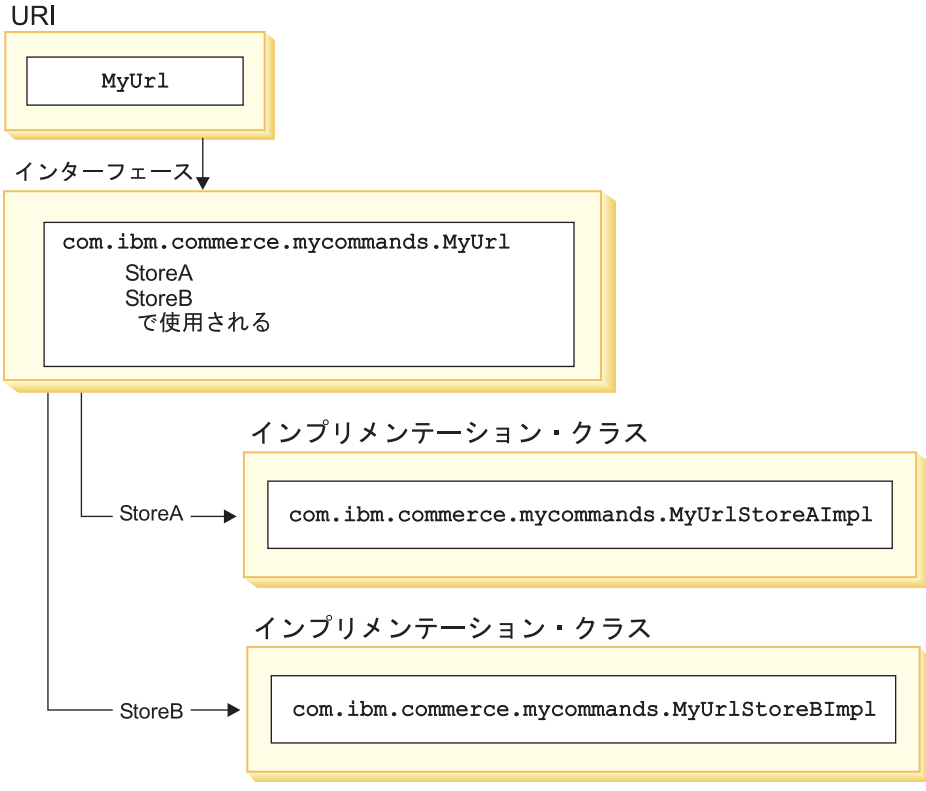


図 10.

VIEWREG テーブル

VIEWREG テーブルにより、装置固有のビュー・コマンドのインプリメンテーションを登録することが可能になります。このテーブルを使うと、ビューのインプリメンテーションを複数登録することができます。こうして、コマンド・フレームワークが、様々なクライアントに別々のビューを戻せるようになります。

ビュー・コマンド名がコントローラー・コマンドから戻された場合、あるいは例外にその名前が指定された場合、Web コントローラーは、ビュー・コマンド・クラスを VIEWREG テーブルから判別します。複数のビュー・コマンド名を同一のインプリメンテーション・クラスにマップすることができます。

列名	説明	注釈
VIEWNAME	ビュー名	AddressForm など。

列名	説明	注釈
DEVICEFMT_ID	装置タイプ ID	使用可能なオプションには以下のものがあります。 • BROWSER (デフォルト値) • I_MODE • E-mail • MQXML • MQNC
STOREENT_ID	ストアのエンティティー ID	すべてのストアで使用するコマンドの場合は 0、特定のストアでしか使用しないコマンドの場合は固有のストア ID に設定できます。
INTERFACENAME	ビュー・コマンド・インターフェース名	デフォルト・オプションは、ForwardView、DirectView、および RedirectView です。
CLASSNAME	ビュー・コマンド・インプリメンテーション・クラス名	デフォルトのインプリメンテーションを使用できます。
PROPERTIES	コマンドへの入力プロパティーとして設定する、名前と値のデフォルトの対	常に同じページが表示される場合は、このプロパティーに JSP ファイル名を設定します (docname=*jsp_name*.jsp)。 すべてのストアで同じ JSP テンプレートを使用する場合は、 storeDir=no と設定して、ストア固有のディレクトリーが使用されることのないようにします。 一般ユーザーがコマンドを呼び出すことができる場合は、isGeneric=true と設定します。
DESCRIPTION	コマンドの説明	
HTTPS	この URL 要求にセキュア HTTP が必要かどうか	HTTPS が必要な場合は 1、不要な場合は 0 を使用します。
LASTUPDATE	このエントリーの最終更新日付	

列名	説明	注釈
INTERNAL	コマンドが WebSphere Commerce の内部的なものかどうか	コマンドが内部的なものであれば 1、外部的なものであれば 0 を使用します。

新しいビュー・コマンドを作成したら、それに対応するエントリーを VIEWREG テーブルに作成する必要があるかもしれません。以下の条件の 1 つが満たされているなら、VIEWREG テーブルにビュー・コマンドを登録する必要があります。

- ビュー・コマンドをアクセス制御のもとで実行する場合
- ビュー・コマンドのインプリメンテーションが複数ある場合
- PROPERTIES 列にプロパティーを設定する場合

登録されたビュー・コマンドは、ビュー名を使用してコマンド・レジストリーを介してアクセスすることもできますし、実際のディスプレイ・ファイルを使用して直接にアクセスすることもできます。 VIEWREG テーブルに登録されていないビューは、クライアントが実際のディスプレイ・ファイル名を使用した場合にしかアクセスできません。

MyView というビューを例にして考えてみましょう。このビューの VIEWREG エントリーは以下のとおりです。

列名	エントリー
VIEWNAME	MyView
DEVICEFMT_ID	BROWSER
STOREENT_ID	0
INTERFACENAME	com.ibm.commerce.commands.ForwardViewCommand
CLASSNAME	com.ibm.commerce.commands.HTTPForwardViewCommandImp
PROPERTIES	docname=MyView.jsp
DESCRIPTION	ビュー名を使用して、または URL から直接 JSP テンプレートを呼び出すための例
HTTPS	0
LASTUPDATE	2000-11-30
INTERNAL	0

MyView は登録済みビューなので、クライアントは、コマンド名を使用してビューにアクセスすることもできますし、コマンド名の代わりに実際のディスプレイ・ファイル名を使用してこれにアクセスすることもできます。ビュー名を使用する場合のサンプルの URL は、以下のとおりです。

```
http://hostname.com/webapp/wcs/stores/servlet/MyView
```

ファイル名を使用する場合のサンプルの URL は、以下のとおりです。

```
http://hostname.com/webapp/wcs/stores/servlet/MyView.jsp
```

クライアントが直接に (ディスプレイ・ファイルを使用して) 登録済みのビューを呼び出す可能性がある場合は、この例に示されているとおり、実際のディスプレイ・ファイル名と同じビュー名を使用してコマンドを登録しなければなりません (MyView と MyView.jsp)。

テーブルに登録されていないビューは、ディスプレイ・ファイル名を使用して呼び出すことしかできません。したがって、ファイル MyUnregisteredView.jsp を使用する未登録のビューがある場合、このビューにアクセスするための URL は以下のようになります。

```
http://hostname.com/webapp/wcs/stores/servlet/MyUnregisteredView.jsp
```

以下の SQL ステートメントの例は、*MyNewViewCommand* のエントリーを作成します。このコマンドは特定のストアで使用されます。

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
CLASSNAME, PROPERTIES, DESCRIPTION,HTTPS, LASTUPDATE, INTERNAL) values
('MyNewViewCommand', 'BROWSER', 5,
'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commmerce.command.HttpForwardViewCommandImpl',
'docname=MyNewViewCommand.jsp', 'A test view command.', 0, '0000-12-01', 0)
```

以下の表は、VIEWREG テーブルの別のサンプルです。主要な情報は以下のとおりです。

COMMAND NAME	DEVICE - FMT_ID	INTERFACE - NAME	CLASSNAME	PROPERTIES
ProductDisplayView	BROWSER	Forward View コマンド	HttpForwardView CommandImpl	
InterestItemAddView	BROWSER	Redirect View コマンド	HttpRedirectView CommandImpl	docname =item.jsp
InterestItemDeleteView	BROWSER	Redirect View コマンド	HttpRedirectView CommandImpl	docname =item.jsp
GenericApplication Error	BROWSER	Redirect View コマンド	HttpRedirectView CommandImpl	docname =usererr.jsp

COMMAND NAME	DEVICE - FMT_ID	INTERFACE - NAME	CLASSNAME	PROPERTIES
GenericSystemError	BROWSER	Redirect View コマンド	HttpRedirectView CommandImpl	docname =syserr.jsp
Logon	BROWSER	Forward View コマンド	HttpForwardView CommandImpl	docname= logon.jsp & storeDir=no

注: COMMANDNAME、INTERFACENAME、CLASSNAME、および PROPERTIES の値にスペースが含まれているのは、いずれも表示上の都合に過ぎません。実際には、それぞれの値は連続したストリングです。列名のハイフンも、表示上の都合によるものです。

上記の表は、以下のシナリオを示すものです。

- コントローラー・コマンド (この場合は ProductDisplay) によって、 Web コントローラーに *ProductDisplayView* ビュー名が戻されます。 Web コントローラーは、ProductDisplayView ビュー・コマンド名と、その装置 ID を使用して、ビュー・コマンド・インターフェースとクラス名を判別します。ビュー・コマンドは、ストアおよび装置 ID ごとに異なるインプリメンテーション・クラスを持つことができます。しかし、インターフェース名は、ビュー・コマンドのタイプを定義するため、同一でなければなりません。
- InterestItemAdd コマンドと InterestItemDelete コマンドにより、 Web コントローラーに、それぞれ *InterestItemAddView* と *InterestItemDeleteView* というビュー名が戻されます。これらのコマンドは両方とも、ビューのリダイレクトを必要とします。したがって、ビュー・コマンド・インターフェース名は、どちらのビューでも RedirectViewCommand になります。どちらのビューでも、登録されている JSP テンプレートは共通です。 Web コントローラーはプロパティー (docname=item.jsp) を取り出して、ビュー・コマンド (HttpRedirectViewCommandImpl) に渡します。
- ユーザー・パラメーターの間違いで、コントローラー・コマンドまたはタスク・コマンドが ECApplication 例外をスローした場合、以下の事柄が生じます。
  - アプリケーション例外の場合に呼び出されるコントローラー・コマンドにビューが指定されている場合、そのビューのエントリーが VIEWREG テーブルから検索され、それに合わせて処理されます。
  - ビューが指定されていない場合は、 GenericApplicationError コマンドが呼び出され、データベースに登録されている JSP テンプレートが表示されます。上記のテーブルを例とした場合、 `usererr.jsp` テンプレートが表示されることになります。
- システム例外のために、コントローラー・コマンドまたはタスク・コマンドが ECSystem 例外をスローした場合、以下の事柄が生じます。

- システム例外の場合に呼び出されるコントローラー・コマンドにビューが指定されている場合、そのビューのエントリーが VIEWREG テーブルから検索され、それに合わせて処理されます。
- ビューが指定されていない場合は、 GenericSystemError コマンドが呼び出され、データベースに登録されている JSP テンプレートが表示されます。上記のテーブルを例とした場合、 `syserr.jsp` テンプレートが表示されることになります。

• ブラウザー・クライアントは、ログオン URL を入力することによって、ログオン・ページを呼び出すことができます。 storeDir プロパティーが “no” に設定されているので、ストア固有の情報は、JSP テンプレートのパスに含まれません。したがって、すべてのストアで同じログオン・ページが顧客に表示されます。

表示デザイン・パターン

表示ページは、クライアントに応答を戻します。表示ページは JSP テンプレートとしてインプリメントされるのが一般的ですが (この方法をお勧めします)、サーブレットとして直接に作成することもできます。

複数の装置タイプをサポートするためには、ビュー・コマンドへの URL アクセスで、実際の JSP ファイルの名前ではなく、ビュー名を使用する必要があります。

このレベルの間接性が備わっている根本的な理由は、 JSP テンプレートがビューを表すということにあります。適切なビューを (ロケール、装置タイプ、または要求コンテキストの他のデータなどに基づいて) 選択できるということは、非常に望ましいことです。単一の要求に対して戻される可能性のあるビューがしばしば複数存在することを考えると、特にそう言えます。 2 人のショッパーがストアのホーム・ページを要求するにあたり、一方は典型的な Web ブラウザーを使い、もう一方がセル電話を使うという例を考えてみてください。当然、各ショッパーに同じホーム・ページが表示されるべきではありません。要求を受け入れ、コマンド登録フレームワークの情報に基づいて、各ショッパーが受け取るビューを判別するのは、Web コントローラーの責任です。

JSP テンプレートとデータ Bean

データ Bean は、動的なコンテンツを提供するために JSP テンプレートの中で使用される Java Bean です。通常、データ Bean は、WebSphere Commerce エンティティー Bean の単純な表現を提供します。データ Bean は、エンティティー Bean から検索したり、エンティティー Bean の中で設定したりすることのできるプロパティーをカプセル化します。そのようにして、データ Bean は、JSP テンプレートに動的データを取り込むタスクを単純化します。

データ Bean には *BeanInfo* クラスがあります。これは、表示ページで使用できるプロパティーを定義するものです。また、BeanInfo クラスでは、プロパティー名が WebSphere Commerce のすべてのサポート言語で提供されているので、データ Bean を複数文化対応型のサイトで使用することも可能です。

データ Bean は、以下の呼び出しでアクティブになります。

```
com.ibm.commerce.beans.DataBeanManager.activate(DataBean, HTTPServletRequest)
```

WebSphere Commerce データ Bean を JSP テンプレートに挿入すると、上記のコード行が WebSphere Studio Page Designer によって自動的に生成されます。

ストア開発者は、JSP テンプレートを開発するときには、ストアのプロパティーと、複数文化への対応の問題を考慮しなければなりません。複数文化への対応の詳細については、 WebSphere Commerce のオンライン・ヘルプを参照してください。

JSP テンプレートとデータ Bean のセキュリティーに関する考慮事項

JSP テンプレートとデータ Bean で特定のコーディング方法を実践すれば、不正なユーザーが許可されない方法でデータベースにアクセスする可能性を最小限にとどめることができます。 SQL ステートメントの挿入、選択、更新、および削除部分の作成は、開発時に行うべきです。ランタイムの入力情報を収集するには、パラメーター挿入を使います。

ランタイムの入力情報を収集するためにパラメーター挿入を使う例は、以下のとおりです。

```
select * from Order where owner =?
```

これとは対照的に、SQL ステートメントの作成に入力ストリングを使うことは避けるべきです。入力ストリングを使用する例は、以下のとおりです。

```
select * from Order where owner = "input_string"
```

データ Bean のタイプ

データ Bean は、主に JSP テンプレートに動的なデータを提供するために使用される Java Bean です。データ Bean には、スマート・データ Bean とコマンド・データ Bean の 2 種類があります。

スマート・データ Bean は、自分自身のデータを検索するために、*遅延フェッチ* メソッドを使います。このタイプのデータ Bean が良好なパフォーマンスを発揮するのは、アクセス Bean のすべてのデータが必要ではない場合です。というのは、このデータ Bean は、必要な場合にだけデータを検索するためです。データベースにアクセスすることが必要なスマート・データ Bean は、対応するエンティティー Bean 用のアクセス Bean から拡張されなければならず、 com.ibm.commerce.SmartDataBean インターフェースをインプリメントしなければなりません。たとえば、 ProductData データ Bean は ProductAccessBean アクセス Bean を拡張したものであり、このアクセス Bean は Product エンティティー Bean に対応します。

データベース・アクセスを必要としないスマート・データ Bean もあります。たとえば、PropertyResource スマート・データ Bean は、データベースからではなく、リソー

ス・バンドルからデータを検索します。データベース・アクセスが不要な場合、スマート・データ Bean は SmartDataBeanImpl クラスを拡張しなければなりません。

コマンド・データ Bean は、コマンドに依存して自分のデータを検索します。これは、より軽量のデータ Bean です。コマンドは、JSP テンプレートが必要とするかどうかにかかわりなく、データ Bean のすべての属性を一度に検索します。その結果、データ Bean の属性の選ばれた部分だけを使用する JSP テンプレートの場合、コマンド・データ Bean はパフォーマンス時間の点で高価なものとなります。属性の大部分、またはそのすべてを必要とする JSP テンプレートの場合、コマンド・データ Bean は大変都合の良いものとなります。

コマンド・データ Bean も自分に対応するアクセス Bean から拡張することができ、com.ibm.commerce.CommandDataBean インターフェースをインプリメントします。

データ Bean のインターフェース

データ Bean は、以下の Java インターフェースのうちの 1 つ、または全部をインプリメントします。

- com.ibm.commerce.SmartDataBean
- com.ibm.commerce.CommandDataBean
- com.ibm.commerce.InputDataBean (オプション)

各 Java インターフェースは、データ Bean のデータ取り込み元のデータ・ソースを記述します。複数のインターフェースをインプリメントすることにより、データ Bean はさまざまなソースのデータにアクセスできるようになります。各インターフェースの詳細については、以下の部分で説明します。

SmartDataBean インターフェース: SmartDataBean インターフェースをインプリメントしたデータ Bean は、関連するデータ Bean コマンドを使用せずに、自分自身のデータを検索できます。スマート・データ Bean は、通常、対応するエンティティー Bean のアクセス Bean から拡張されます。スマート・データ Bean がアクティブになると、データ Bean マネージャーがデータ Bean の取り込みメソッドを起動します。取り込みメソッドを使うことにより、データ Bean はすべての属性を検索することができます。ただし、関連オブジェクトからの属性は例外です。たとえば、データ Bean がエンティティー Bean のアクセス Bean クラスから拡張されたものである場合、データ Bean は refreshCopyHelper メソッドを呼び出します。対応するエンティティー Bean の属性は、すべてスマート・データ Bean に自動的に取り込まれます。ただし、エンティティー Bean に関連オブジェクトがある場合、それらのオブジェクトの属性は検索されません。スマート・データ Bean を使用することの主な利点は、以下のとおりです。

- インプリメンテーションが単純であり、データ Bean コマンドを作成する必要がない。
- エンティティー Bean に新しいフィールドが追加されても、データ Bean に変更を加える必要がない。エンティティー Bean を変更したら、その後、アクセス Bean を再

生成する必要があります (VisualAge for Java のツールを使用)。アクセス Bean が再生成され次第、自動的にすべての新しい属性がスマート・データ Bean で使用できるようになります。

- エンティティー Bean には、しばしば関連オブジェクトを表す属性が含まれています。パフォーマンス上の理由から、スマート・データ Bean は自動的にそれらの属性を検索することがありません。むしろ、以下の図に示すように、それらの属性が必要になるまで検索を遅らせる方が望ましいです。

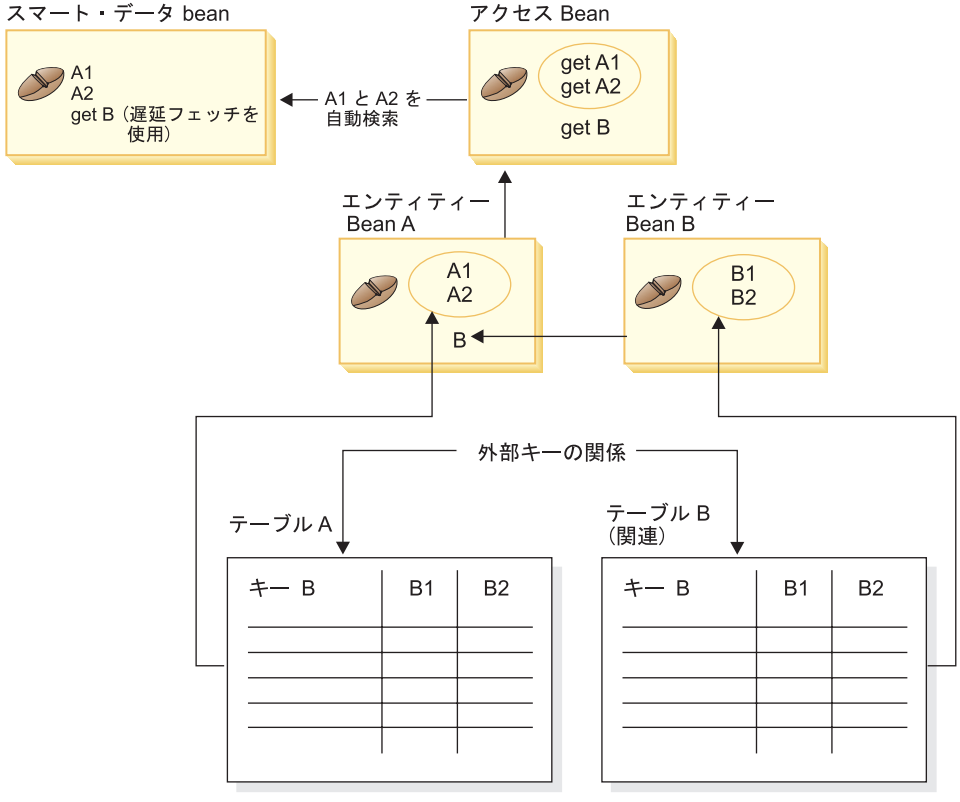


図 11.

遅延フェッチ検索のインプリメントについては、 44 ページの『遅延フェッチ・データ検索』を参照してください。

CommandDataBean インターフェース: CommandDataBean インターフェースをインプリメントしたデータ Bean は、データ Bean コマンドからデータを検索します。このタイプのデータ Bean は軽量オブジェクトであり、データの取り込みにあたって、データ Bean コマンドに依存します。データ Bean は、データ Bean コマンドのインタ

ーフェース名を戻す getCommandInterfaceName() メソッドを (com.ibm.commerce.CommandDataBean インターフェースに定義されているとおりに) インプリメントしなければなりません。

InputDataBean インターフェース: InputDataBean インターフェースをインプリメントしたデータ Bean は、 URL パラメーターから、またはビュー・コマンドによって設定された属性からデータを検索します。

このインターフェースに定義された属性は、追加のデータを取り出すための基本キーとして使用することができます。 JSP テンプレートが呼び出されると、生成された JSP サーブレット・コードは URL パラメーターに一致するすべての属性を取り込み、次いでデータ Bean をデータ Bean マネージャーに渡すことによって、データ Bean をアクティブにします。それから、データ Bean マネージャーは、データ Bean の setRequestProperties() メソッドを (com.ibm.commerce.InputDataBean インターフェースに定義されているとおりに) 呼び出して、ビュー・コマンドによって設定されているすべての属性を渡します。 Page Designer を使用してページに挿入されるデータ Bean のそれぞれについて、 WebSphere Studio が以下のコードを生成することに注意してください。

```
com.ibm.commerce.beans.DataBeanManager.activate(DataBean, HTTPServletRequest);
```

BeanInfo クラス

BeanInfo クラスなしでは、データ Bean は完全ではありません。このクラスは、java.lang.Object.BeanInfo インターフェースをインプリメントします。 BeanInfo クラスは、データ Bean のメソッドとプロパティーに関する明示的な情報を提供するために使用されます。このクラスは、データ Bean インプリメンテーション・クラスのパブリック・ランタイム・メソッドを Web 設計者から隠蔽するため、またはデータ Bean の属性のそれぞれに適切な表示ストリングを設定するために使用できます。

BeanInfo クラスのインプリメントの詳細については、Sun Microsystems の JavaBeans の仕様書を参照してください。

データ Bean の活動化

データ Bean は、 com.ibm.commerce.beans.DataBeanManager クラスの activate メソッド、または silentActivate メソッドのどちらかを使用してアクティブにすることができます。 activate メソッドは完全な活動化メソッドであり、活動化イベントが成功するのは、すべての属性が使用可能な場合だけです。属性が 1 つでも使用不能であると、活動化の処理全体に関して例外がスローされます。

silentActivate メソッドは、個々の属性が使用不能であっても、例外をスローすることはありません。

JSP テンプレートからのコントローラー・コマンドの呼び出し

JSP テンプレートからのコントローラー・コマンドの呼び出しとディスプレイからのロジックの分離に整合性はありませんが、 JSP テンプレートからのコントローラー・コマンドの呼び出しが必要な状態になることがあります。その場合は、`ControllerCommandInvokerDataBean` をこのために使用できます。

このデータ Bean を使用すると、呼び出すコマンドのインターフェース名を指定するか、呼び出すコマンド名を直接設定できます。コマンドの要求プロパティーも設定できます。

データ Bean がデータ Bean マネージャーによって活動化されると、コントローラー・コマンドが実行され、応答プロパティーが JSP テンプレートで使用可能になります。

一度コントローラー・コマンドが実行されれば、ビューを実行できます。

遅延フェッチ・データ検索

活動化されたデータ Bean には、データ Bean コマンドやデータ Bean の populate() メソッドにより、データを取り込むことができます。属性は、データ Bean の対応するエンティティー Bean から検索されます。エンティティー Bean は、関連オブジェクトを持つこともあります。これらのオブジェクト自身はいくつかの属性を持ちます。

活動化したときにすべての関連オブジェクトの属性が自動的に検索されると、パフォーマンス上の問題が発生する可能性があります。パフォーマンスは、関連オブジェクトの数が増えるとともに低下します。

多数のクロスセル、アップセルまたはアクセサリー製品 (関連オブジェクト) を含む製品データ Bean について考えてください。製品データ Bean が活動化されるとすぐにすべての関連オブジェクトにデータを取り込むことができます。しかしながら、このようにデータを取り込むには、複数のデータベース・クエリーが必要になる場合があります。ページですべての属性が必要でないのであれば、複数のデータベース・クエリーは非効率的となる可能性があります。

一般に、ページですべての属性が必要になることはないので、以下のような遅延フェッチを実行する方が良い設計パターンと言えます。

```
getCrossSellProducts () {
   if (crossSellDataBeans == null)
     crossSellDataBeans= getCrossSellDataBeans();
   return crossSellDataBean;
    }
```

JSP 属性の設定 - 概要

WebSphere Commerce プログラム・モデルは、MVC 設計パターンを促進しています。この設計パターンでは、URL 要求の結果表示がコントローラーおよびタスク・コマンドから分離されます。これらのコマンドは装置に依存せず、クライアントに関する情報がなくても、ビジネス・ロジックをインプリメントし、クライアントに戻されるデータを生成します。逆に、ビュー・コマンドは装置に固有です。

コントローラーおよびタスク・コマンドはビューを直接に構成するのではなく、ビューに情報を渡します。情報がビューに渡される方法を理解することは重要です。以下の図は、 Web コントローラー、コマンド・レジストリー、コントローラー・コマンド、およびビュー・コマンドの間でプロパティーが渡される方法を示しています。

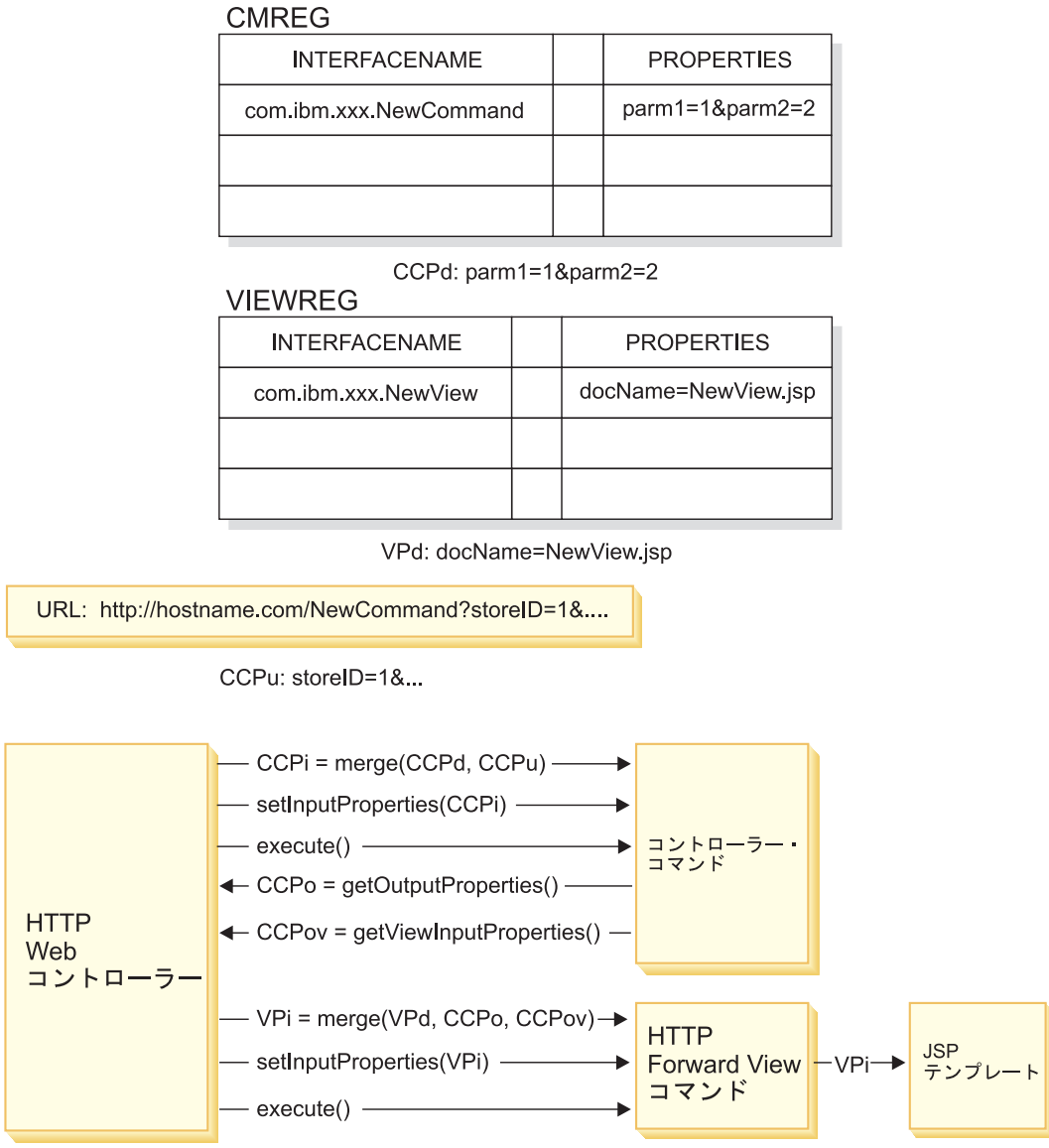


図 12.

上の図は、以下の相互作用を示しています。

- Web コントローラーは、URL パラメーターの入力プロパティー (CCPu) と、CMDREG テーブル内にあるコントローラー・コマンドのエントリー (CCPd) をマージします。これにより、CCPi が作成されます。
- Web コントローラーは、マージしたプロパティー (CCPi) をコントローラー・コマンドに渡し、コントローラー・コマンドを実行します。

- コントローラーは、出力プロパティー CCPo を設定します。これらは、コマンド自体によって生成される出力プロパティーです。出力プロパティーの 1 つである viewCommandName は、適切なビュー・コマンド名に設定されます。これらのプロパティーは、Web コントローラーによって get メソッドを使用して検索されます。
- コントローラー・コマンドは、別の出力プロパティー・セット CCPov を設定します。デフォルトでは、これらは元のマージ済み入力プロパティー (CCPi) に設定されます。これらのプロパティーをカスタマイズすることができます。たとえば、すべての入力パラメーターをビュー・コマンドに渡すことは必ずしも必要ではありません。
- Web コントローラーは、 3 つのプロパティー・セット CCPo、CCPov、および VPd (VIEWREG テーブルで登録されているプロパティー) を、ビュー・コマンドの入力プロパティー (VPi) にマージします。
- Web コントローラーは、マージしたプロパティー VPi を設定し、ビュー・コマンドを実行します。
- ビュー・コマンドは、属性を入力プロパティーの JSP テンプレートに設定します。

新規コマンドを作成する場合、プロパティーのマージを明示的に実行する必要はありません。抽象コマンド・クラスには、mergeProperties メソッドが含まれています。このメソッドについて詳しくは、WebSphere Commerce のオンライン・ヘルプの『Reference』セクションを参照してください。

必要なプロパティー設定

コントローラー・コマンドでは、ビュー・コマンドの各タイプについて以下のプロパティーを設定する必要があります。コマンドで設定されない場合、これらのプロパティーは VIEWREG テーブルで定義されていなければなりません。

- ForwardView コマンドを使用する場合は、`docname = view_file_name` (*view_file_name* はディスプレイ・テンプレートの名前) を設定します。たとえば、`docname = productDisplay.jsp` です。
- DirectView コマンドを使用する場合は、以下のいずれかを行います。
  - `textDocument = xxx` (*xxx* は、文書をテキスト・フォームで含む java.io.InputStream オブジェクト) を設定する
  - `rawDocument = yyy` (*yyy* は、文書をバイナリー・フォームで含む java.io.InputStream オブジェクト) を設定する

  DirectView コマンドを使用する場合は、オプションで `contentType = ttt` (*ttt* は文書コンテンツ・タイプ) を設定することもできます。
- RedirectView コマンドを使用する場合は、`url = uuu` (*uuu* はリダイレクト URL) を設定します。

第 3 章 永続オブジェクト・モデル

WebSphere Commerce は、大量の永続データを処理します。現在のデータベース・スキーマでは、520 以上のテーブルが定義されています。この広範囲なスキーマを使用するとしても、特定のビジネス・ニーズに合わせてデータベース・スキーマの拡張やカスタマイズが必要になることがあります。

WebSphere Commerce は、永続オブジェクト層として、 Enterprise JavaBeans (EJBs) コンポーネント・アーキテクチャーに基づくエンティティー Bean を使用します。これらのエンティティー Bean は、コマース・ドメイン内の概念やオブジェクトをモデル化するように、 WebSphere Commerce データを表します。この永続層により、拡張可能なフレームワークが提供されています。

VisualAge for Java エンタープライズ版は、このフレームワークの開発をサポートする、洗練された EJB ツールと単体テスト環境を備えています。

WebSphere Commerce エンティティー Bean のインプリメンテーション

WebSphere Commerce エンティティー Bean - 概要

前述のとおり、 WebSphere Commerce アーキテクチャー内の永続層は、 EJB コンポーネント・アーキテクチャーに従ってインプリメントされています。EJB アーキテクチャーは、 2 つのタイプのエンタープライズ Bean (エンティティー Bean とセッション Bean) を定義しています。エンティティー Bean-managed はさらに、コンテナー管理永続 (CMP) bean と bean 管理永続 (BMP) bean に分けられます。

ほとんどの WebSphere Commerce エンティティー Bean は、CMP エンティティー Bean です。少数のステートレス・セッション Bean は、集中的なデータベース操作 (たとえば、特定列にあるすべての行の合計の実行) を処理するために使用されます。 CMP エンティティー Bean を使用する 1 つの利点は、 VisualAge for Java エンタープライズ版で提供されている EJB ツールをデベロッパーが使用できることです。これらのツールにより、デベロッパーは、 Java オブジェクトとそのデータベース・テーブル・マッピングを定義することができます。これらのツールは、エンティティー Bean で必要な persister を自動的に生成します。 persister とは、 Java フィールドをデータベースに対して永続化し、 Java フィールドにデータベースからデータを取り込む Java オブジェクトのことです。

VisualAge for Java では、現在の EJB 仕様が 2 つの点 (EJB 継承とアソシエーション) で拡張されています。 EJB 継承により、エンタープライズ Bean は、同じグループに存在する別のエンタープライズ Bean から、プロパティー、メソッド、およびメソッ

ド・レベルの制御記述子属性を継承することができます。アソシエーションとは、2 つの CMP エンティティー Bean の間に存在する関係のことです。

EJB 継承機能は、一部の WebSphere Commerce エンティティー Bean で利用されます。 VisualAge for Java が提供するアソシエーション機能は、WebSphere Commerce エンティティー Bean では使用されません。独自のエンティティー Bean を開発する際は、 VisualAge for Java のアソシエーション機能を使用しないことをお勧めします。これは、オブジェクト・モデルができるだけ複雑にならないようにするためです。 VisualAge for Java が提供するアソシエーション機能を使用する代わりに、明示的な getter メソッドをエンタープライズ Bean 内で設定することにより、複数のエンタープライズ Bean の間でオブジェクト関係を設定することができます。

WebSphere Commerce は、2 種類のエンタープライズ Bean セット (private と public) を備えています。private エンタープライズ Bean は、WebSphere Commerce ランタイム環境およびツールで使用されます。これらの bean は決して 使用または変更しないでください。

一方、public エンタープライズ Bean は、商取引アプリケーションで使用され、使用することも拡張することも可能です。 public エンタープライズ Bean は、以下の EJB グループに編成されています。

- WCSActrlEJBGroup
- WCSApproval
- WCSAuction
- WCSCatalog
- WCSCommon
- WCSContract
- WCSCoupon
- WCSFulfillment
- WCSInventory
- WCSMessageExtensions
- WCSOrder
- WCSOrderManagement
- WCSOrderStatus
- WCSPayment
- WCSPVCDevices
- WCSTaxation
- WCSUserTraffic
- WCSUser
- WCSUTF

上記の EJB グループの一部には、セッション Bean が含まれています。将来のマイグレーションを単純化するため、セッション Bean クラスは変更しないでください。必要な場合は、新規 EJB グループ内に新規セッション Bean を作成することができます。新規セッション Bean の作成について詳しくは、 77 ページの『新規セッション Bean の作成』を参照してください。

WebSphere Commerce エンタープライズ Bean のデプロイメント記述子

デプロイメント記述子は、シリアライズされていて、エンタープライズ Bean に関するランタイム設定を含む、特殊なクラスです。 WebSphere Commerce エンタープライズ Bean のデプロイメント記述子は、特定の方法で設定され、変更することはできません。

新規エンタープライズ Bean (エンティティーまたはセッション Bean) を作成する際、(VisualAge for Java の) EJB Development Environment 内でデプロイメント記述子を設定するには、 bean を右クリックして**「プロパティー」**を選択します。 新規エンタープライズ Bean のデプロイメント記述子は、 WebSphere Commerce エンタープライズ Bean と同じ規則に従っていなければなりません。特に、属性を以下のように設定してください。

属性	値
Transaction Attribute	TX_REQUIRED
Isolation Level	TRANSACTION_READ_COMMITTED
Run-As Mode	SYSTEM_IDENTITY または CLIENT_IDENTITY
Reentrant	これは選択しないでください。

エンタープライズ Bean には、データベースから情報を読み取るだけで、データベース更新は実行しないメソッドがしばしば含まれます。これらのメソッドのことを、*読み取り専用* メソッドと言います。すべての読み取り専用メソッドは、そのようなものとして明示的にマークされている必要があります (メソッドを右クリックして、**「EJB メソッドの属性」**>**「読み取り専用メソッド」**を選択します)。読み取り専用メソッドがこのようにしてマークされていないと、 EJB コンテナーは、トランザクションの最後にデータベースを不必要に更新しようとして、読み取り専用トランザクション内にトランザクション・ロールバック・エラーを引き起こします。これにより、パフォーマンス上の問題が生じます。

分離レベル

VisualAge for Java の内部の WebSphere Commerce エンタープライズ Bean で使用されるトランザクション分離レベルは TRANSACTION_READ_COMMITTED です。 DB2 の JDBC ドライバーと Oracle の JDBC ドライバーでは、この分離レベルのインプリメンテーションに違いがあることに注意してください。 WebSphere Application Server 環境でデプロイメントを実行するときに使用されるトランザクション分離レベルが、使用されるデータベースによって異なるのと同様です。

WebSphere Test Environment の外で操作する際に DB2 データベースで使用される分離レベルは、 TRANSACTION_REPEATABLE_READ です。 WebSphere Test Environment の外で操作する際に Oracle データベースで使用される分離レベルは、 TRANSACTION_READ_COMMITTED です。

DB2 データベースでデプロイメントを実行する場合は、トランザクション分離レベルを手動で変更する必要はありません。デプロイメントのステップ中、 modifyIsolationLevel コマンドを出すときに変更されます。

次のテーブルは、 DB2 と Oracle のトランザクション分離レベルの、対応する JDBC トランザクション分離レベルへのマッピングを表しています。

JDBC	DB2	Oracle
Read Uncommitted	Uncommitted Read	Read Uncommitted
Read Committed	Cursor Stability	(適用外)
Repeatable Read	Read Stability	Read Committed
Serializable	Repeatable Read	Serializable
なし	(適用外)	(適用外)

DB2 での Cursor Stability トランザクション分離レベルでは、指定されたトランザクション中に更新された行だけが排他的にロックされます。指定された行のどの列も更新されない場合は、 SQL ステートメントから戻された結果セットの一部であっても、カーソルが別の行に移動した時点でその特定の行の行ロックが解除されます。在庫の更新などの状況では、これは望ましい振る舞いではありません。したがって、 DB2 でトランザクション分離レベルを Read Stability に変更すれば、並行性に多少のトレードオフはあってもデータ保全性は非常に向上します。

Oracle では Read Committed トランザクション分離レベル、または JDBC の Repeatable Read に相当するトランザクション分離レベルしか使用できないので、実際のインプリメンテーションは、 DB2 の Repeatable Read トランザクション分離レベルに一番似た振る舞いをする方法で行われます。

WebSphere Commerce オブジェクト・モデルの拡張

WebSphere Commerce オブジェクト・モデルは、以下の方法で拡張することができます。

- WebSphere Commerce のパブリック・エンタープライズ Bean を拡張する
- 新規エンティティー Bean を作成する
- 新規ステートレス・セッション Bean を作成する

以降のセクションでは、これらの拡張を実行する方法について詳しく説明されています。

オブジェクト・モデルの拡張方法

アプリケーション要件によっては、既存の WebSphere Commerce オブジェクト・モデルの拡張が必要になることもあります。そのような要件の一例は、アプリケーションに属性を追加することです。そのためには、以下のいずれかの方法があります。

既存の WebSphere Commerce パブリック・エンティティー Bean を変更しない方法
新しいデータベース・テーブルを作成してから、そのテーブル用の新しいエンティティー Bean を作成します。必要に応じて、新しい属性を操作するためのフィールドやメソッドをそのエンティティー Bean に追加します。新しいエンティティー Bean 用のデプロイメント・コードとアクセス Bean を生成します。アプリケーションは、その新しい属性を必要とするようになった時点で、アクセス Bean オブジェクトをインスタンス化して、その属性の取り込み、設定、操作のためにそのメソッドを使用します。

既存の WebSphere Commerce パブリック・エンティティー Bean を変更する方法
新しいデータベース・テーブルを作成してから、変更対象の既存のエンタープライズ Bean に対応する既存のテーブルとその新しいテーブルとの間で、テーブル結合を作成します。既存の WebSphere Commerce パブリック・エンティティー Bean に新しいフィールドを作成してから、2 次テーブル・マップを使用して、そのフィールドを新しいテーブルの対応する列にマッピングします。必要に応じて、メソッドを追加します。既存のエンティティー Bean 用のデプロイメント・コードとアクセス Bean を生成します。アプリケーションがアクセス Bean オブジェクトをインスタンス化した時点で、新しい属性が使用可能になります。

上記の 2 つの方法には、それぞれ利点と欠点があります。基本的には、パフォーマンスやコードの保守作業に関連した利点や欠点です。

拡張の例: 一例として、顧客の住居のタイプを取り込まなければならないようなアプリケーションについて考えてみましょう。まず、顧客 ID と住居タイプを保管するための USERRES テーブルを作成します。住居タイプ (resType) としては、自己保有の家、マンション、アパートなどがあるでしょう。この種の情報は個人情報なので、Commerce Suite の既存の USERDEMO テーブルと関連があります。 WebSphere Commerce のコード・リポジトリーを調べてみると、 WCSUser EJB グループに ″Demographics″ というエンタープライズ Bean があります。この bean には、USERDEMO テーブルに保管されている個人情報を操作するための getter と setter が用意されています。

カスタマイズを実行するには、 2 つの方法があります。 USERRES テーブルを操作するための新しいエンティティー Bean を作成する方法と、 Demographics bean に新しいフィールド (および適切な getter メソッドと setter メソッド) を追加する方法です。

最初の方法 (まったく新しいコードを作成する方法) の場合は、新しい Userres エンティティー Bean を作成してから、そのフィールドを USERRES テーブルの列にマッピングします。アプリケーションは、顧客の住居タイプ情報を必要とするようになった時点

で、 Userres アクセス Bean オブジェクトをインスタンス化して、データを取り出すことになります。そのときに他の個人情報も必要になった場合は、 Demographics アクセス Bean オブジェクトもインスタンス化して、他の必要な属性も取り出す必要があります。アプリケーション・ロジックの中で、顧客のすべての個人情報を取り出そうとする部分については、元のアクセス Bean だけでなく、新しいアクセス Bean もインスタンス化するように変更する必要があります。以下の図は、この方法でオブジェクト・モデルを拡張する手順を示したものです。

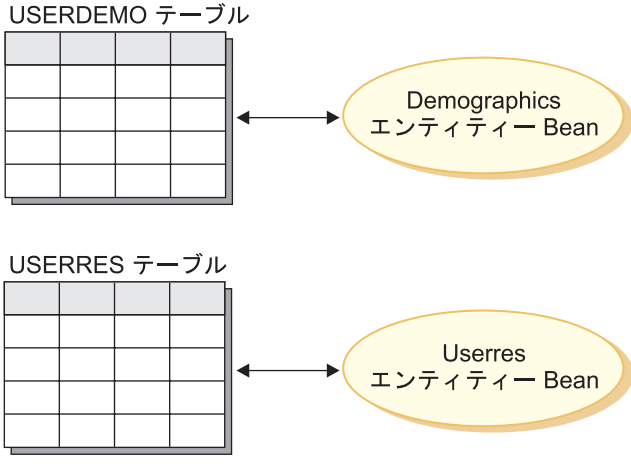


図 13.

表示テンプレートの観点からすると、データ Bean もその新しい属性にアクセスする機能を必要とします。そうであれば、その情報が JSP テンプレートで使用できるからです。 JSP テンプレートを作成する Web 開発者に統一的なビューを提示するには、元の (既存の) エンティティー Bean のアクセス Bean を拡張した新しいデータ Bean を作成するのが望ましいでしょう。そのデータ Bean では、新しいアクセス Bean から属性を取り込むために代行操作を使用するようにします。以下の図は、このデータ Bean のインプリメンテーション・シナリオを示したものです。

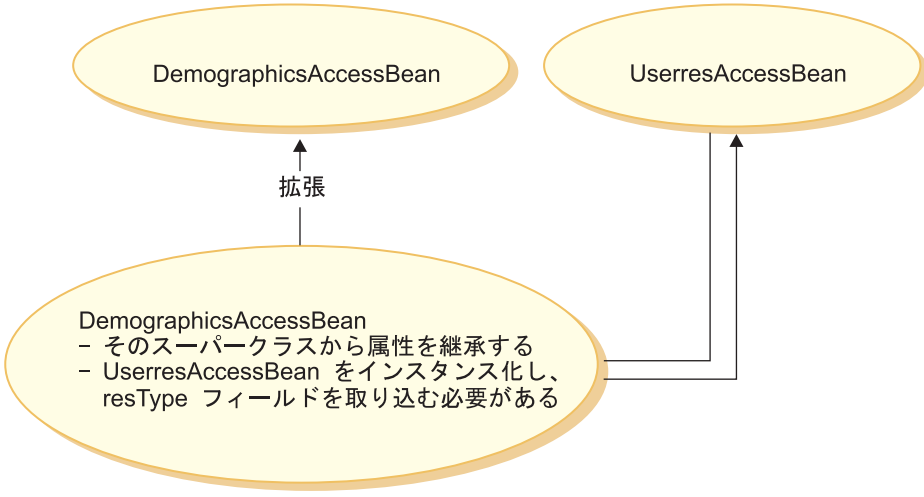


図 14.

2 番目の方法 (既存のコードを変更する方法) の場合は、 Demographics エンティティー Bean に新しいフィールドを追加してから、その新しいフィールドと USERRES テーブルの適切な列との間に 2 次テーブル・マップを作成します。アプリケーションは、顧客の住居タイプ情報を必要とするようになった時点で、 Demographics アクセス Bean オブジェクトをインスタンス化して、住居タイプ情報を取り出すことになります。アプリケーションがその顧客の他の個人情報も必要とする場合は、その bean に対する同じ呼び出しで、そうした情報を入手することができます。以下の図は、この方法でエンタープライズ Bean を変更する手順を示したものです。

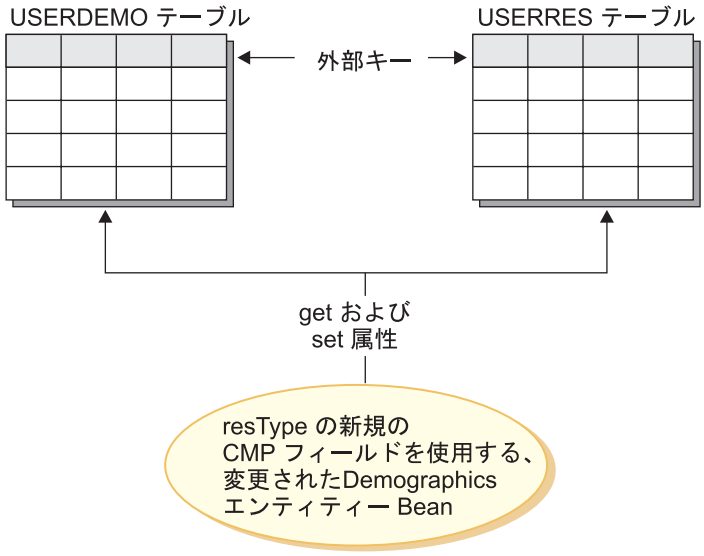


図 15.

表示テンプレートの観点からすると、 DemographicsAccessBean が生成し直された時点で、新しい属性 (resType) はデータ Bean でも自動的に使用できるようになります。

ただし、オブジェクト・モデルを拡張する場合は、既存の WebSphere Commerce データベース・テーブルに新しい列を追加することは、*絶対に* 避けてください。新しい属性用に新しいテーブルを作成する必要があります。既存のテーブルに新しい列を追加した場合は、 WebSphere Commerce の将来のリリースにマイグレーションする時点で、新しい属性が失われてしまいます。

パフォーマンスとコードの保守作業に関する注意点: ランタイムのパフォーマンスが優れているのは、2 番目の方法です。 2 番目の方法の場合は、新しい属性の取得や設定のために、 1 つのエンティティー Bean のインスタンス化だけが必要であり、 1 つの取り出し操作だけですべての必要な属性を取り出すことができるからです。

2 番目の方法は、既存の WebSphere Commerce コードを変更しているので、新しい WebSphere Commerce コード・リポジトリーがリリースされたときに、マイグレーションの問題が発生します。新しいコードとカスタマイズ・コードをマージする必要がありますが、新しいコード・リポジトリーをインポートするときに、エンタープライズ Bean に追加した新しいフィールドと新しいテーブルとの間のマッピング情報は保存されません。したがって、新しいリリースの WebSphere Commerce コード・リポジトリーにマイグレーションするときには、以下の手順を実行する必要があります。

1. カスタマイズした EJB コードのバージョンを設定します。
2. 新しいバージョンの WebSphere Commerce コードをインポートします。

3. VisualAge for Java のツールを使用して、カスタマイズ・バージョンのコードと新しいリリースの WebSphere Commerce コードを比較します。カスタマイズ・コードをワークスペースに戻します。
4. WebSphere Commerce パブリック・エンタープライズ Beans に追加した属性とデータベース内の適切な列との間のマッピングを手作業で設定し直します。
5. ステップ 4 で変更したエンタープライズ Bean 用のデプロイメント・コードとアクセス Bean を再生成します。

このマイグレーション作業を簡略化するために、開発時にオブジェクト・モデルの拡張方法をしっかりと文書化しておくのは大切なことです。

オブジェクト・モデルを大幅に拡張する場合は、上記の 2 つの方法を組み合わせることもあるでしょう。最初の方法は、システムの中でもパフォーマンスの低下があまり起きない部分に使用し、 2 番目の方法は、パフォーマンスを重視しなければならない部分に使用できます。このようにして、将来のマイグレーション作業を最小限に抑えるとともに、システム・パフォーマンスを高いレベルに維持できます。

推奨されるセッション Bean の使用法

WebSphere Commerce の強みの 1 つに、コンテナー管理パーシスタンス (CMP) エンティティー Bean を利用できる機能があります。 CMP エンティティー Bean は、VisualAge for Java 内で提供されるツールによって生成される、分散した永続的な、トランザクションについてのサーバー側の Java コンポーネントです。多くの場合、オブジェクトの永続性については CMP エンティティー Bean は非常によい選択です。また、CMP エンティティー Bean は、オブジェクトと関係のマッピングを行う他のオプションと少なくとも同じくらい効率的であり、場合によってはより効率的です。これらの理由から、WebSphere Commerce はコア・コマース・オブジェクトをインプリメントするのに CMP エンティティー Bean を使用しています。

ただし、セッション Bean JDBC helper の使用をお勧めする場合もあります。これには以下のような状況があります。

- クエリーが大規模な結果セットを戻す場合。これは *大規模な結果セット* の事例といいます。
- クエリーがさまざまなテーブルからデータを検索する場合。これは *集合エンティティー* の事例といいます。
- SQL ステートメントがデータベース集中操作を実行する場合。これは *任意の SQL* の事例といいます。

詳細について以下に説明します。

セッション Bean を JDBC ラッパーとして使用してデータベースから情報の検索を行う場合、リソース・レベルのアクセス制御をインプリメントすることはさらに難しくなります。このようにしてセッション Bean を使用する場合、セッション Bean の開発者は

適切な “where” 文節を “select” ステートメントに追加して、無許可のユーザーがリソースにアクセスできないようにしなければなりません。

大規模な結果セットの事例: クエリーが大規模な結果セットを戻し、検索されたデータが主に読み取りまたは表示の目的である場合があります。この場合はステートレス・セッション Bean を使用し、そのセッション Bean 内で、エンティティー Bean でのファインダー・メソッドと同じ機能を実行するファインダー・メソッドを作成するのがよりよい方法です。つまり、ステートレス・セッション Bean でのファインダー・メソッドは以下を行う必要があります。

- SQL select ステートメントを実行する
- 取り出される各行について、アクセス Bean をインスタンス化する
- 検索される各列について、アクセス Bean 内に対応する属性を設定する

アクセス Bean が戻されるとき、アクセス Bean がセッション Bean 内のファインダー・メソッドによって戻されたのか、エンティティー Bean 内のファインダー・メソッドから戻されたのかを、コマンドは判別できません。つまり、セッション Bean 内でファインダー・メソッドを使用してもプログラミング・モデルにいかなる変更も起こりません。呼び出しコマンドだけが、セッション Bean 内のファインダー・メソッドを呼び出しているのか、エンティティー Bean 内のファインダー・メソッドを呼び出しているのかを判別します。これはプログラミング・モデルのその他すべてのパーツに透過的です。

集合エンティティーの事例: この事例では、1 つのビューが複数のオブジェクトの部分を結合し、複数のデータベース・テーブルからの情報の断片が単一の表示ページに取り込まれます。たとえば “My Account” の概念について考えてみます。これは、顧客情報のテーブルからの情報 (たとえば、顧客の名前、年齢、および顧客 ID) およびアドレス・テーブルからの情報 (たとえば、市区町村と番地からなる住所) によって構成できます。

SQL 結合を実行することで、単純な SQL ステートメントを構成してさまざまなテーブルからすべての情報を検索することが可能です。これは “ディープ・フェッチ” を実行するといいます。以下に、“My Account” の例での SQL select ステートメントの例を示します。ここで、CUSTOMER テーブルは T1 で、ADDRESS テーブルは T2 です。

```
select T1.NAME, T1.AGE, T2.STREET, T2.CITY
  from CUSTOMER T1, ADDRESS T2
  where (T1.ID=? and T1.ID=T2.ID)
```

VisualAge for Java における EJB ツールは、ディープ・フェッチという概念をサポートしていません。その代わりに遅延フェッチを行い、これは結果的に各関連オブジェクトについて SQL select を行います。これは、このタイプの情報の検索には望ましいメソッドではありません。

ディープ・フェッチを実行するには、セッション Bean の使用をお勧めします。そのセッション Bean 内に、必要な情報を検索するためのファインダー・メソッドを作成してください。ファインダー・メソッドは以下を行う必要があります。

- ディープ・フェッチについて SQL select ステートメントを実行する
- 主テーブルの各行について、各関連オブジェクトと同様に、アクセス Bean をインスタンス化する
- 取り出される各列および取り出される各関連オブジェクトについて、アクセス Bean 内に対応する属性を設定する

アクセス Bean は、例外を送る getter メソッドをキャッシュに入れないことに注意してください。この場合は、以下のパターンを使用して、アクセス Bean について単純なラッパー・クラスを作成してください。

```
public class CustomerAccessBeanCopy extends CustomerAccessBean {
   private AddressAccessBean address=null;

   /* The following method overrides the getAddress method in
      the CustomerAccessBean.
   */
   public AddressAccessBean getAddress() {
      if (address == null)
         address = super.getAddress();
      return address;
      }

    /* The following method sets the address to the copy. */

    public void _setAddress(AddressAccessBean aBean) {
       address = aBean;
      }
}
```

CUSTOMER および ADDRESS の例に続けて、セッション Bean のファインダー・メソッドは CUSTOMER テーブルの各行について CustomerAccessBean をインスタンス化し、 ADDRESS テーブルの対応する各行について AddressAccessBean をインスタンス化します。それから、ADDRESS テーブルの各列について、 AddressAccessBean (市区町村と番地) に属性を設定し、 ADDRESS テーブルの各列について、 CustomerAccessBean (名前、年齢および住所) に属性を設定します。これを以下の図に示します。

図 16.

任意の SQL の事例: この事例には、データベース集中操作を実行する、一連の任意の SQL ステートメントがあります。たとえば、テーブル内のすべての行を合計する操作は、データベース集中操作と考えられます。選択されたすべての行が、永続モデルのエンティティー Bean に対応するとは限りません。

任意の SQL ステートメントを作成する結果となる例としては、顧客が非常に大規模なデータ集合をブラウズしようとするときがあります。たとえば、オンラインの金物屋のすべてのファスナーを検査したい場合や、オンラインの洋服店のすべての服を検査したい場合などです。この場合は非常に大規模な結果セットが作成されますが、ほとんどの場合、この結果セットの各行で必要とされるフィールドは少しだけです。つまり、最初には、アイテムの名前、写真、および価格を表示する要約だけを顧客に示せば足りるかもしれません。

この場合は、セッション Bean の helper メソッドを作成してください。このセッション Bean の helper メソッドは、読み取りと書き込みのいずれかの操作を実行します。読み取り操作を実行するときは、表示の目的で使用される読み取り専用の値をもつオブジェクトを戻します。

適切なデータ・モデルを使用することで、任意の SQL ステートメントを使用する事例の数を通常は最小にすることができます。

パブリック・エンティティー Bean の拡張

このセクションでは、WebSphere Commerce のパブリック・エンタープライズ Bean の設計パターンについて説明します。この設計パターンにより、新規の永続フィールド、新規のビジネス・メソッド、または新規のファインダー・メソッドを追加するなどの拡張を行うことができます。

以下の図は、Catalog エンティティー Bean のインプリメンテーション・クラスを示しています。

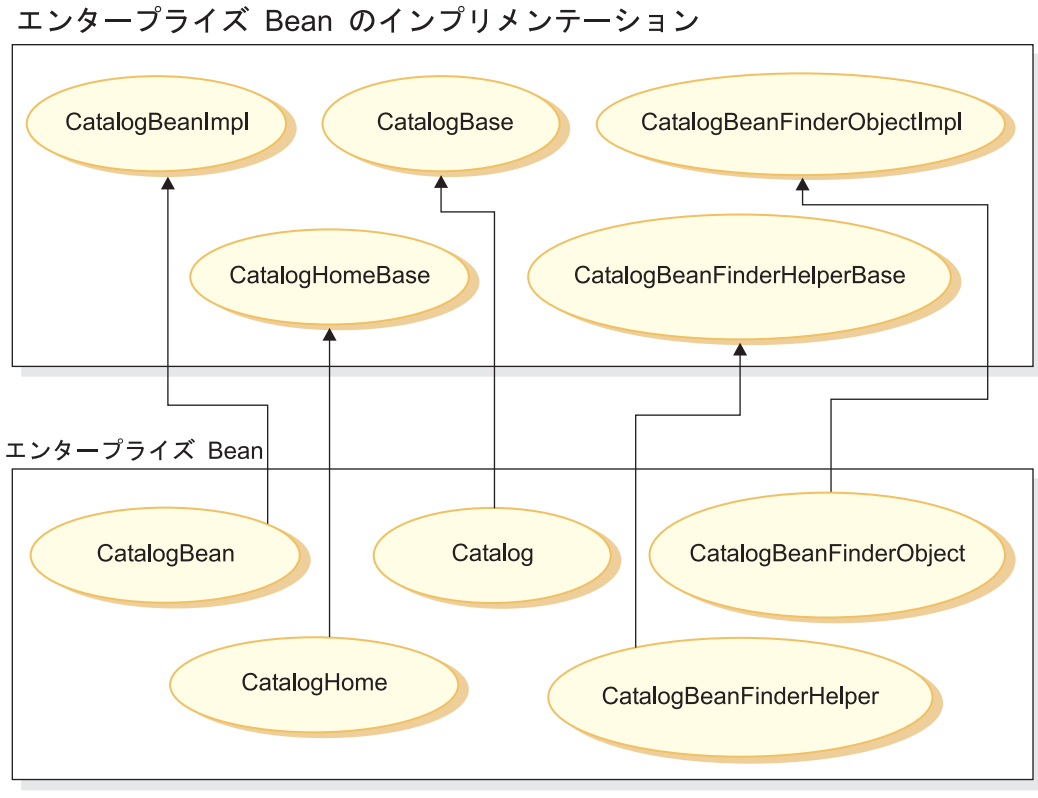


図 17.

他のエンティティー Bean も同様の方法で構造化され、同じ命名規則に従うので、上記の図は他のエンティティー Bean にも適用されます。この図を別のエンティティー Bean に適用するには、 “Catalog” をそのエンティティー Bean の名前に置き換えます。たとえば、InterestItemBean クラスは InterestItemBeanImpl クラスを拡張し、InterestItem インターフェースは InterestItemBase インターフェースを拡張します。

この図は、パブリック・エンタープライズ Bean のインプリメンテーション・クラスまたはインターフェースが、 Java 継承を用いて 2 つのパーツに分離されていることを示しています。スーパークラスまたはインターフェースには、 WebSphere Commerce インプリメンテーション・コードが含まれています。これらのすべてのスーパークラスとインターフェースは、子クラスおよびインターフェースとは別個のパッケージおよびプロジェクトで定義されています。

WebSphere Commerce コード・リポジトリーには、 *bean_name*BeanFinderHelperBase および *bean_name*BeanFinderObjectImpl クラスを除いて、これらのすべてのスーパークラスとインターフェースのバイナリー・コードが含まれています。*bean_name*BeanFinderHelperBase および *bean_name*BeanFinderObjectImpl クラスについ

ては、ソース・コードが含まれているので、各ファインダーの SQL ステートメントがどのように定義されているかが分かります。これらのクラスは、好きなように変更することができます。

CatalogBeanFinderHelperBase および CatalogBeanFinderObjectImpl のソース・コードを調べるには、 com.ibm.commerce.catalog.objsrc パッケージをオープンします。他のパッケージも同様の命名規則を使用しています。

子クラスおよびインターフェースは、変更することができます。

新しいファインダー・メソッドをパブリック・エンタープライズ Bean に追加する場合、メソッドの特定の命名規則に従う必要があります。新しいメソッドには、findX*a_description* という名前を付けます (*a_description* は任意の説明です)。たとえば、findXByOwnerId や findXByOrderStatus のような名前にします。このような命名規則を使用すれば、WebSphere Commerce のファインダー・メソッドとの名前の衝突 (重複名) の可能性を避けることができます。

既存の WebSphere Commerce パブリック・エンティティー Bean を変更する方法に、追加フィールドを追加する方法があります。この場合、新規のフィールドを追加してから、 bean の各ファインダー・メソッド検査する必要があります。ファインダー・メソッドの where 文節部分にデータベース別名 (たとえば T1. や T2.) が含まれる場合は、その別名を除去しなければなりません。

CMP エンタープライズ Bean の新規作成

WebSphere Commerce オブジェクト・モデルに新しい属性を追加する必要がある場合は、その必須属性の列を含むデータベース・テーブルを新しく作成できます。作成後、この属性をエンタープライズ Bean にも組み込んで、 WebSphere Commerce コマンドがその情報にアクセスできるようにしなければなりません。

新しい属性を WebSphere Commerce オブジェクト・モデルに統合する方法の 1 つとして、新しい CMP エンタープライズ Bean を作成するという方法があります。作成後、この bean 中に、新しいデータベース・テーブル中の属性に対応するフィールドを作成します。

新しい CMP エンタープライズ Bean を作成するには、 VisualAge for Java で以下のステップを実行しなければなりません。

1. ワークスペースの所有者が WCS 開発者に設定されていることを確認します。
2. この bean 用の新規 EJB グループを作成します。
3. 「Create Enterprise Bean SmartGuide (エンタープライズ Bean の作成 SmartGuide)」ツールを使用して、新規 CMP エンタープライズ Bean を作成します。
4. 対応するデータベース・テーブルの各列について、新規の CMP フィールドを bean に追加します。

5. 必要に応じて、作成された各 CMP フィールドに getter および setter メソッドの対を作成します。
6. 必要に応じて、FinderHelper インターフェース中に FinderHelper フィールドを定義して、新規の FinderHelper メソッドを追加します。
7. 必要に応じて、新規の ejbCreate メソッドを作成し、ejbCreate メソッドを、エンタープライズ Bean のホーム・インターフェースにプロモートします。このステップは、新規エンタープライズ Bean が、対応するデータベース・テーブルに新規エントリーを作成しなければならない場合に必要です。
8. エンタープライズ Bean 中のフィールドを、データベース・テーブル中の列にマップします。
9. アクセス Bean を生成し、エンタープライズ Bean についてコードをデプロイメントします。

上記の各ステップの詳細について、以下のセクションで説明します。

注: 新規のエンタープライズ Bean が WebSphere Commerce アクセス制御システムによって保護される場合は、詳細について 91 ページの 『第 4 章 アクセス制御』を参照してください。アクセス制御は bean を作成してから追加できます。

USERRES という名前のテーブルを新しく作成し、ユーザーの住居タイプに関する情報を指定したとします。このテーブルには、3 つの列が含まれているとします。 USERID 列、HOME 列 (家のタイプを指定)、および ROOMS 列 (住居の中の寝室の数を指定) です。

新規 EJB グループの作成: 新規エンティティー Bean を作成する際は、WebSphere Commerce EJB グループとは別個の EJB グループ内に作成する必要があります。VisualAge for Java でエンタープライズ Bean を処理するには、ワークスペース内で「EJB」タブに切り替えなければなりません。次いで**「EJB」**メニューから**「追加」**>**「EJB グループ」**を選択すると、「Add EJB Group SmartGuide (EJB グループの追加 SmartGuide)」を立ち上げることができます。

EJB グループを作成する際に指定しなければならない重要なアイテムが 2 つあります。それは、EJB コードを保管するプロジェクトと、EJB グループの名前です。

EJB プロジェクトはワークスペース内の「プロジェクト」タブで表示できます。新しい EJB グループを作成する際には、 WebSphere Commerce プロジェクトとは別のプロジェクトを指定しなければなりません。たとえば、EJB グループで MyCustomEJB プロジェクトが使用されるように選択できます。このプロジェクトは VisualAge for Java によって自動的に作成できるので、グループを作成する前にこのプロジェクトを作成する必要はありません。このプロジェクトは EJB コード専用にする必要があります。いかなるコマンドやデータ Bean コードにも使用できません。このコード・タイプの分離は、デプロイメントのために必要です。独自のカスタマイズ済みコードを WebSphere

Commerce コードから分離することにより、将来のリリースへのマイグレーションの影響をできるだけ小さくすることができます。

プロジェクト名と EJB グループ名の両方とも、ご使用のアプリケーションの該当する命名規則に従っているか確認してください。

CMP エンタープライズ Bean の新規作成: 新規 CMP エンタープライズ Bean を作成するには、「Create Enterprise Bean SmartGuide (エンタープライズ Bean の作成 SmartGuide)」ツールを使用できます。このツールを立ち上げるには、新しい bean を追加する EJB グループを右マウス・ボタン・クリックし、**「追加」>「エンタープライズ Bean」**を選択します。

新しいエンタープライズ Bean を選択して作成し、 bean の名前を指定してください。エンタープライズ Bean に関する WebSphere Commerce の命名規則では、 bean に対応するテーブルと同じ名前がその bean に付けられます。たとえば、新しいデータベース・テーブルの名前が USERRES の場合は、エンタープライズ bean の名前は UserRes になります。

自動的に「プロジェクト」フィールドにプロジェクトの名前が取り込まれます。コードの保管先にする、プロジェクト内のパッケージ名を指定しなければなりません。パッケージに保管するコードの例としては、エンタープライズ Bean 用に作成するアクセス Bean のコードがあります。パッケージを命名する際にも、ご使用のアプリケーションの該当する命名規則に従っているか確認してください。パッケージ名の例としては、`com.mycompany.mycustombeans` などが挙げられます。

bean クラスで、VisualAge for Java は `EntityContext` と呼ばれる専用フィールドを作成します。 WebSphere Commerce は ECEntityBean に独自のエンティティー・コンテキスト・フィールドを提供します。新しいエンティティー Bean では、ユーザー自身のクラスで生成されるフィールドではなく、このフィールドを使用してください。このため、新しいエンティティー Bean から、生成された `EntityContext` を除去してください。

新しいエンタープライズ Bean には `serialVersionUID` フィールドが含まれていなければなりません。新規 bean を作成するために SmartGuide を使用する場合は、VisualAge for Java がこのフィールドを生成します。bean の作成にツールを使用しない場合はこのフィールドを追加しなければなりません。

「スーパークラス」フィールドで、`com.ibm.commerce.base.objects.ECEntityBean` クラスを指定しなければなりません。以下のコード例は、このスーパークラスが提供する機能を示しています。

```
public class myEJB extends com.ibm.commerce.base.objects.ECEntityBean {
   public void ejbLoad()throws java.rmi.RemoteException {
      super.ejbLoad();--the super method will add EJB trace
      --your logic --
   }
```

```
   public void ejbStore()throws java.rmi.RemoteException {
      super.ejbStore();--the super method will add EJB trace
      --your logic --
   }
}
```

また、引き数を取らない ejbCreate() メソッドと ejbPostCreate() メソッドもすべて除去してください。エンティティー Bean にこれらのメソッドを残しておくと、ランタイム・エラーの原因となります。

データベース・テーブルの列ごとに、 bean 中に新しい CMP フィールドを作成しなければなりません (bean オプションに対する「CMP の追加」フィールドを表示するには、 SmartGuide で「次へ」をクリックしなければなりません)。フィールドごとにフィールド名とフィールドのデータ・タイプを指定してください。基本キーまたは基本キーの一部である列の場合は、「鍵フィールド」チェック・ボックスを使用可能にしてください。他のすべての列の場合は、リモート・インターフェースのチェック・ボックスに対して Promote getter および setter メソッドを使用可能にしてください。

すべてのフィールドに記入し終えたら、「終了」をクリックすると、 VisualAge for Java により新しい CMP エンタープライズ Bean が作成されます。

Bean_NameFinderHelper インターフェース中に FinderHelper フィールドを新規作成する: *Bean_Name*FinderHelper インターフェースには、 findByPrimaryKey メソッド以外のすべての FinderHelper メソッドに対応する SQL 検索文節が含まれています。

新しいエンタープライズ Bean は、FinderHelper メソッドとともに、その bean の FinderHelper インターフェースで定義された findXBy*ArgName* メソッド (*ArgName* は引き数名) を使用して、SQL クエリーを構成します。 “findXBy” 命名規則をフィールド名に使用して、フィールド名が常に WebSphere Commerce フィールド名とは異なる固有名になるようにしてください。

bean 中に新しい FinderHelper フィールドを作成するには、 *Bean_Name*_FinderHelper インターフェースを選択して、ソース・コードを修正し、 Select ステートメントのフォームを確立しなければなりません。以下に例を示します。

```
public interface UserResFinderHelper {
public static final String findXByHomeAndRoomsWhereClause = "(T1.HOME = ?
      and T1.ROOMS = ?)";
}
```

この時点で、ホーム・インターフェースには findXByHomeAndRooms というメソッドが必要になります。このメソッドには、HOME および ROOMS ごとに、? 文字で表される値を取り込む入力パラメーターがあります。このようなクエリー構造のタイプを、*パラメーター挿入* といいます。

入力パラメーターが ″detached″ および ″3″ であるとすると、以下の SQL ステートメントが生成されます。

```
select * from USERRES where HOME=detached and ROOMS=3
```

セキュリティー上の理由で、新しいエンティティー Bean の FinderHelper メソッドを作成するときには、パラメーター挿入を使用する必要があります。こうすれば、クエリーが他者によって変更されるのを避けることができます。これに代わる方法として、以下のような構造体の使用も考えられます。

```
public static final String
     findXByOwnerIdWhereClause = " (T1.OWNERID = input_string) ";
```

ここで、*input_string* は URL から渡されるストリング値です。これは望ましい方法ではありません。不正なユーザーが “‘123’ OR 1=1” のような値を入力する可能性があり、その結果、SQL ステートメントが変更されてしまうからです。ユーザーが SQL ステートメントを変更できるのであれば、データへの無許可アクセスも可能になる場合があります。このような理由で、パラメーター挿入を使用することをお勧めします。

パラメーター挿入を使用することができず、 SQL ステートメントの作成に入力ストリングを使用しなければならない場合には、入力ストリングのパラメーター検査を必ず実行して、入力パラメーターがデータへの不正なアクセスの試みでないことを確かめる必要があります。

***Bean_Name*Home インターフェース中に FinderHelper メソッドを新規作成する:** *Bean_Name*FinderHelper インターフェース中に指定した FinderHelper フィールドごとに、 bean のホーム・インターフェース中に FinderHelper メソッドを作成しなければなりません。 FinderHelper メソッドの名前は、 FinderHelper フィールド名から ″WhereClause″ を除いたものに正確に一致していなければなりません。つまり、フィールド名 findXByHomeAndRoomsWhereClause を例にすると、対応するメソッド名は findXByHomeAndRooms になります。

新しい FinderHelper メソッドを作成するには、以下のようにします。

1. *Bean_Name***Home** インターフェースを右マウス・ボタン・クリックして、**「追加」** > **「メソッド」**を選択します。
   「Create Method SmartGuide (メソッドの作成 SmartGuide)」がオープンします。
2. **「新規メソッドの作成」**を選択し、**「次へ」**をクリックします。
3. **「メソッド名」**フィールドに、 FinderHelper メソッドの名前を入力します。この FinderHelper メソッドの名前は、 FinderHelper フィールド名から ″WhereClause″ の部分を除いたものに正確に一致していなければなりません。たとえば findXByHomeAndRooms と入力します。
4. **「戻りタイプ」**フィールドに、以下のいずれかを入力します。

- FinderHelper メソッドが基本キーを使用してデータベースをクエリーし、固有のレコードを戻す必要がある場合は、戻りタイプとして EJB オブジェクトを指定します。 たとえば UserRes と入力します。
- FinderHelper メソッドが固有のレコードではなく結果セットを戻す場合は、戻りタイプとして java.util.Enumeration を指定します。

5. **「このメソッドが持つべきパラメーター」**の隣の**「追加」**ボタンをクリックして、該当するパラメーターを追加します。たとえば、String タイプの argHome を追加して住居タイプを保持したり、 byte タイプの argRooms を追加して部屋数を保持したりすることができます。
6. すべてのパラメーターを追加し終えたら、**「次へ」**をクリックします。
7. **「このメッセージが送った例外」**の隣の**「追加」**ボタンをクリックして、以下の例外を追加します。
   - java.rmi.RemoteException
   - javax.ejb.FinderException

   注: 上記の例外は、ユーザーのメソッドが送る必要がある最低限の例外です。ユーザー独自のコードに応じて、その他の例外を指定する必要があります。
8. **「終了」**をクリックします。

ejbCreate メソッドの新規作成: エンタープライズ Bean を作成する際に、ejbCreate メソッドは自動的に作成されます。作成後、このメソッドはリモート・インターフェースにプロモートされ、アクセス Bean で使用可能になります。デフォルトの ejbCreate メソッドには、基本キーまたは基本キーの一部であるパラメーターしか含まれていません。したがって、インスタンス化の際には、これらの値だけがインスタンス化されます。

エンタープライズ Bean に基本キーの一部ではないヌル不可フィールドが含まれている場合は、新しい ejbCreate メソッドを作成し、その中のこれらのフィールドを特別にインスタンス化しなければなりません。この処理を行うと、新しいレコードを作成するたびに、すべてのヌル不可フィールドに該当するデータが取り込まれます。

新しい ejbCreate メソッドを作成するには、以下のようにします。

1. 「タイプ」ペインで、***Bean_Name*Bean** クラスを拡張表示します。たとえば、**UserResBean** を選択します。
2. 既存の ejbCreate メソッドをクリックして、ソース・コードを表示します。(エンタープライズ Bean の基本キーに応じて、ejbCreate(String)、ejbCreate(String, int)、または他の入力パラメーターが含まれるので注意してください。)
3. ソース・コードを修正し、個々の CMP フィールドが入力パラメーターとして組み込まれるようにして、個々の CMP フィールドが該当する値でインスタンス化されるようにしなければなりません。 UserRes の例では、UserId が基本キーなので、最初の時点ではソース・コードは以下のとおりです。

```
public void ejbCreate(int argUserId)
   throws javax.ejb.CreateException, java.rmi.RemoteExeption {
   _initLinks();
   userId = argUserId;
}
```

しかし、部屋数と家のタイプを両方とも確実に初期化したいとします。この場合、コードに以下のような変更を加えます。

```
public void ejbCreate(int argUserId, String argHome, byte Rooms)
   throws javax.ejb.CreateException, java.rmi.RemoteExeption {
   _initLinks();
   // All CMP fields should be initialized here
   userId = argUserId;
   home = argHome;
   rooms = argRooms;
}
```

注: システムが生成する基本キーを使用したい場合、詳細については 82 ページの『基本キー』を参照してください。

4. 変更を加えたメソッドを保管します。コードを保管する際に、 VisualAge for Java により新しいパラメーターを含む新しい ejbCreate メソッドが作成されます。オリジナルの ejbCreate メソッドも保存されています。
5. オリジナルの ejbCreate メソッド (引き数のないもの) を削除します。
6. 新規 ejbCreate メソッドを右マウス・ボタン・クリックして、**「追加」**>**「EJB ホーム・インターフェース」**を選択します。
7. 対応する ejbPostCreate メソッドを作成します。(このメソッドをホーム・インターフェースに追加する必要はありません。)

データベース・テーブルの新規エンタープライズ Bean へのマッピング: 新しいエンタープライズ Bean を作成し終えたら、 bean 中の CMP フィールドとデータベース・テーブル中の列との間のマッピングを作成しなければなりません。このマッピングのことをスキーマといいます。 VisualAge for Java にはこのタスクを単純化できるツールが備えられています。

スキーマを作成するには、以下のようにします。

1. **「EJB」**メニューから、**「オープン」**>**「データベース・スキーマ」**を選択します。スキーマ・ブラウザーがオープンします。
2. **「スキーマ」**メニューから、**「インポート / エクスポート・スキーマ」**>**「データベースからスキーマをインポートする」** を選択します。
3. 新しいスキーマの名前を入力して、**「OK」**をクリックします。

4. 「データベース接続」ウィンドウで、情報を以下のように入力します。

属性	DB2 値	Oracle 値
接続タイプ	COM.ibm.db2.jdbc.app. DB2Driver	Oracle.jdbc.driver. OracleDriver
データ・ソース	jdbc:db2:wc_database_name	jdbc:oracle:thin@*hostname*: *port*:*Oracle_SID*
ユーザー名	*wc_db_user_name*	*wc_db_user_name*
パスワード	*wc_db_password*	*wc_db_password*

値は次のように置き換えられます。

- DB2 *wc_database_name* は WebSphere Commerce データベースの名前です。
- Oracle *hostname* は Oracle サーバーのホスト名です。
- Oracle *port* は Oracle データベースのポート番号です。
- Oracle *Oracle_SID* は Oracle のインスタンス ID です。
- *wc_db_user_name* はデータベースのユーザー名です。
- *wc_db_password* はデータベースのパスワードです。

5. **「修飾子」**リストから、ご使用のデータベースの修飾子 (データベース・ユーザー名の場合があります) を選択します。
6. **「テーブル・リストの構築」**をクリックします。
7. 生成されたリストから「*your_new_table*」を選択して、「OK」をクリックし、スキーマを生成します。
8. スキーマが生成されたら、「スキーマ」ペインの「*your_new_table*」をクリックします。
9. 「テーブル」ペインの「*your_new_table*」を強調表示してからダブルクリックします。
   「テーブル・エディター」ウィンドウがオープンします。
10. **「修飾子」**フィールド中の情報を除去します。この処理により、エンタープライズ Bean を他のマシンに取り込むことができるようになります。
11. **「スキーマ」**メニューから、**「スキーマの保管」**を選択します。該当するプロジェクト、パッケージ、およびクラス名を入力します。

次に、スキーマ・マップを作成しなければなりません。スキーマ・マップとは、エンタープライズ Bean でのデータベースの列とフィールド間のマッピングです。

スキーマ・マップを作成するには、以下のようにします。

1. **「EJB」**メニューで、**「オープン」>「スキーマ・マップ」**を選択します。「データ・ストア・マップ」ウィンドウがオープンします。
2. マップの名前を入力します。推奨される新規マップの命名については、85 ページの『データベース・スキーマ・オブジェクト命名に関する考慮事項』を参照してください。
3. EJB グループとスキーマを選択します。推奨される新規スキーマの命名については、 85 ページの 『データベース・スキーマ・オブジェクト命名に関する考慮事項』 を参照してください。
4. 「データ・ストア・マップ」ペインで、ご使用のマップを選択します。
5. 「永続クラス」ペインで、ご使用のクラスを選択します。
6. **「テーブル・マップ」**メニューから、**「新規テーブル・マップ」>「テーブル・マップを継承なしで追加」**を選択します。
7. **「テーブル」**ドロップダウン・リストで、ご使用のテーブルを選択し、**「OK」**をクリックします。
8. 「テーブル・マップ」パネルで、ご使用のテーブル・マップを強調表示してから右マウス・ボタン・クリックし、**「プロパティー・マップの編集」**を選択します。プロパティー・マップ・エディターがオープンします。
9. クラス属性 (bean 中の CMP フィールド) ごとに、マップのタイプと、マップ先のデータベース列を指定する必要があります。たとえば、マップ・タイプ ″Simple″ を使用して、データベース・テーブル中の USERID 列に、UserId クラス属性をマップできます。
10. すべてのフィールドを対応するデータベース列にマップし終えたら、スキーマ・マップを保管しなければなりません。**「データベース・マップ」**メニューから、**「データ・ストア・マップの保管」**を選択します。該当するプロジェクト、パッケージ、およびクラス名を入力して、マップを保管します。**「終了」**をクリックします。

アクセス Bean の作成およびデプロイメント・コードの生成: アクセス Bean は、エンタープライズ Bean のラッパーの働きをし、他のコンポーネントとエンタープライズ Bean との対話を単純化します。新しいエンタープライズ Bean のアクセス Bean を作成しなければなりません。

デプロイメント・コードを生成するとき、 VisualAge for Java のツールが bean を分析して、EJB サーバーに特定の規則と同様に、 Sun Microsystems EJB 仕様で指定される規則が満たされるようにします。

新しいエンタープライズ Bean 用のアクセス Bean を作成するには、以下のようにします。

1. 「エンタープライズ Bean」ペインで、新規エンタープライズ Bean を右マウス・ボタン・クリックして、**「追加」>「アクセス Bean」**を選択します。 (bean を表示するには、新規 bean が含まれる EJB グループを最初に拡張表示する必要が生じるこ

とがあります。)
「Create Access Bean SmartGuide (アクセス Bean の作成 SmartGuide)」がオープンします。

2. **「EJB グループ」**、**「エンタープライズ Bean」**および**「アクセス Bean 名」**フィールドで、該当する EJB グループ、bean 名、およびアクセス Bean 名を指定します。
3. **「アクセス Bean タイプ」**として**「エンティティー Bean 用のコピー・ヘルパー」**を選択して、**「次へ」**をクリックします。
4. **「ゼロ引き数コンストラクターのホーム・メソッドを選択する」**ドロップダウン・リストから、 **findByPrimaryKey** を選択します。
5. **「コンバーター」**ドロップダウン・リストから、最初のプロパティーに **WCSStringConverter** を選択し、**「次へ」**をクリックします。
6. 「コピー・ヘルパー用に bean プロパティーを選択してカスタマイズする」ウィンドウで、各フィールドについて **WCSStringConverter** を選択します。
7. **「終了」**をクリックします。

デプロイメント・コードを生成するには、以下のようにします。

1. 「エンタープライズ Bean」ペインで、新規エンタープライズ Bean を右マウス・ボタン・クリックして、**「デプロイメント・コードの生成」**を選択します。

このツールを使って生成されるデプロイメント・コードは EJB 1.0 仕様に準拠しており、 VisualAge for Java 内でエンタープライズ Bean を実行する場合に限り使用されます。後で、エンタープライズ Bean を WebSphere Application Server V4.0 内で実行する WebSphere Commerce アプリケーションにデプロイする場合、 EJB 1.1 仕様に準拠したデプロイメント・コードを含む JAR ファイルを生成する必要があります。この EJB 1.1 Export JAR ファイルの詳細については、194 ページの 『EJB デプロイメント・コードに関する情報』および 363 ページの 『デプロイメント・コードの生成』を参照してください。

エンタープライズ Bean をテストするテスト・クライアントの使用:

VisualAge for Java に付属のテスト・クライアントを使用して、エンタープライズ Bean をテストすることができます。テスト・クライアントを使用して新規 bean をテストするには、以下のようにします。

1. テストするエンタープライズ Bean を含む EJB サーバーを開始します。
2. エンタープライズ Bean を右クリックして**「テスト・クライアントの実行」**を選択します。
「EJB テスト・クライアント」および「EJB 検索」ウィンドウがオープンします。
3. **「JNDI 名」**フィールドでエンタープライズ Bean の JNDI 名を入力して、**「検索」**をクリックします。
4. 引き数が充てんされている **findByPrimaryKey** メソッドを右クリックして、**「呼び出し」**を選択します。

コーディングの慣例: 以下のエンタープライズ Bean のコーディングの慣例に注意してください。

- BLOB データ・タイプも CLOB データ・タイプも使用しないでください。
- LONG データ・タイプ (LONG VARCHAR ともいう) の CMP フィールドは、エンタープライズ Bean についての CMP フィールド・リストの最初または最後のメンバーにしないでください。リストを検査するには、`EJSJDBCPersister._hydrate()` メソッドをチェックして、リストの最初または最後のエレメントが LONG VARCHAR タイプかどうかを検査してください。最初のフィールドがこのタイプである場合は、次のようにしてください。
  1. 最初のフィールドを設定解除します。これを `fieldA` と呼びます。
  2. LONG VARCHAR タイプ でない 別のフィールドを設定解除します。これを `fieldB` と呼びます。
  3. `fieldA` をリセットします。
  4. `fieldB` をリセットします。
  5. スキーマ・マップ・ブラウザーをオープンし、これらの変更を反映するようにテーブル・マップを編集します。マップを保管します。
  6. デプロイメント・コードを再生成します。
- エンタープライズ Bean コードは、エンタープライズ Bean パッケージ以外のどのようなものも参照してはなりません。たとえば、エンタープライズ Bean コード内でコマンドやデータ Bean を参照してはなりません。
- ユーザーのエンタープライズ Bean のアクセス制御を使用可能にするには、com.ibm.commerce.security.Protectable インターフェースと com.ibm.commerce.security.Groupable インターフェースのどちらか、または両方をエンタープライズ Bean のリモート・インターフェースに追加します。これらのインターフェースを追加してから、bean のデプロイメント・コードおよびアクセス Bean を再生成します。また、objsrc パッケージにアクセス・ヘルパー・クラス・オブジェクトを作成しなければなりません。

シンプルなデータ Bean の作成

データ Bean は、エンタープライズ Bean から情報を検索するのに JSP テンプレートで使用される bean です。シンプルなデータ Bean は、対応するアクセス Bean を拡張し、 SmartDataBean インターフェースをインプリメントしたものです。データ Bean のほとんどのコードは、VisualAge for Java によって自動的に生成されます。

シンプルなデータ Bean を作成するには、以下のステップを実行しなければなりません。

1. データ Bean コードを保管するプロジェクトとパッケージを作成します。
2. 対応するアクセス Bean を拡張し、該当するデータ Bean インターフェースをインプリメントして、データ Bean を作成します。
3. データ Bean 用の set メソッドを作成します。

4. データ Bean 用の get メソッドを作成します。

データ Bean コード用のプロジェクトとパッケージの作成: プロジェクトとパッケージを作成すると、データ Bean コードを保管できる場所が作成されます。

新しいプロジェクトを作成するには、以下のようにします。

1. **「プロジェクト」**タブを選択します。
2. **「選択済み」**メニューから、**「追加」>「プロジェクト」**を選択します。
   「Add Project SmartGuide (プロジェクトの追加 SmartGuide)」がオープンします。
3. **「名前を付けた新規プロジェクトの作成」**が選択されていることを確認して、新しいプロジェクトの名前を入力します。たとえば、`My Data Beans` と入力します。
4. **「終了」**をクリックします。

新しいパッケージを作成するには、以下のようにします。

1. データ Bean コードのために作成したプロジェクトを右クリックして、**「追加」>「パッケージ」**を選択します。たとえば、**My Data Beans** を右クリックして、**「追加」>「パッケージ」**を選択します。
   「Add Package SmartGuide (パッケージの追加 SmartGuide)」がオープンします。
2. **「名前を付けた新規パッケージの作成」**が選択されていることを確認して、データ Bean パッケージに適した名前を入力します。たとえば `com.mycompany.mydatabeans` と入力します。
3. **「終了」**をクリックします。

データ Bean の作成: データ Bean は、動的なコンテンツをページに提供するために JSP テンプレートの中で使用される Java Bean です。データ Bean は、アクセス Bean を拡張して、エンティティー Bean を (間接的に) シンプルに表現します。データ Bean は、エンティティー Bean から検索したり、エンティティー Bean の中で設定したりすることのできるプロパティーをカプセル化します。

データ Bean を作成するには、以下のようにします。

1. データ Bean を保管するパッケージを右マウス・ボタン・クリックし、**「追加」>「クラス」**を選択します。
   「Create Class SmartGuide (クラスの作成 SmartGuide)」がオープンします。
2. プロジェクト名とパッケージ名のフィールドがすでに取り込まれています。
3. **「新規クラスの作成」**が選択されていることを確認して、**「次へ」**をクリックします。
4. **「クラス名」**フィールドで、新規データ Bean の名前を入力します。たとえば、UserResAccessBean を拡張するデータ Bean を作成するには、 `UserResDataBean` と入力します。

5. スーパークラスを指定するために、**「ブラウズ」**をクリックし、パターン・フィールドに対応するアクセス Bean を入力します。たとえば、UserResAccessBean と入力して、**「OK」**をクリックします。
6. **「次へ」**をクリックします。
7. データ Bean がインプリメントするインターフェースを指定するために、**「追加」**をクリックします。「インターフェース」ウィンドウで、以下のようにします。
   a. **「パターン」**フィールドに com.ibm.commerce.beans.SmartDataBean と入力し、**「追加」**をクリックします。
   b. **「パターン」**フィールドに com.ibm.commerce.beans.InputDataBean と入力し、**「追加」**をクリックします。
   c. **「クローズ」**をクリックします。
8. **「終了」**をクリックします。

データ Bean への必須フィールドの追加: ここでは新しいデータ Bean に必須フィールドを追加する方法について説明します。

iCommandContext フィールドを追加するには、以下のようにします。

1. 新しいデータ Bean (たとえば UserResDataBean) を右クリックし、「追加」> 「フィールド」を選択します。
   「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。
2. **「フィールド名」**フィールドに、 iCommandContext と入力します。
3. **「ブラウズ」**をクリックしてフィールド・タイプを追加し、com.ibm.commerce.command.CommandContext と入力します。**「OK」**をクリックします。
4. アクセス修飾子について**「保護」**を選択します。
5. **「終了」**をクリックします。

iRequestProperties フィールドを追加するには、以下のようにします。

1. 新しいデータ Bean (たとえば UserResDataBean) を右クリックし、「追加」> 「フィールド」を選択します。
   「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。
2. **「フィールド名」**フィールドに、 iRequestProperties と入力します。
3. **「ブラウズ」**をクリックしてフィールド・タイプを追加し、com.ibm.commerce.datatype.TypedProperty と入力します。**「OK」**をクリックします。
4. アクセス修飾子について**「保護」**を選択します。
5. **「終了」**をクリックします。

データ Bean の set メソッドの変更: データ Bean を作成してから、生成された set メソッドのいくつかでコードを変更しなければなりません。

set メソッドを更新するには、以下のようにします。

1. 新しいデータ Bean を拡張表示して、フィールドとメソッドを表示します。
2. **setCommandContext(CommandContext)** メソッドを選択して、そのソース・コードを表示します。
   「ソース」ペインに以下のようなソース・コードが表示されます。

```
public void setCommandContext(com.ibm.commerce.comand.CommandContext arg1)
 {
 }
```

3. メソッドが以下のように表示されるように、ソース・コードを変更します。

```
public void setCommandContext(com.ibm.commerce.comand.CommandContext arg1)
{
   iCommandContext = arg1;
}
```

   ここまでの作業を保管します (Ctrl + S)。

4. **setRequestProperties(TypedProperty)** メソッドを選択して、そのソース・コードを表示します。
   「ソース」ペインに以下のようなソース・コードが表示されます。

```
public void setRequestProperties(
com.ibm.commerce.datatype.TypedProperty arg1)
   throws Exception
   {
   }
```

5. 対応するアクセス Bean の基本キーを取り込めるようにソース・コードを変更したいかもしれません。これには、データ Bean マネージャーを使用してこの値を間接的に設定することをお勧めします。この間接的な方法は、URL プロパティーから取られる基本キー値が、すでに基本キーが設定されている場合にこれをオーバーライドしないようにするためです。ユーザーの setRequestProperties メソッドをこのモデルに従わせるには、以下のコードの断片と同じような方法でコーディングしてください。以下の例では、基本キーがユーザー ID であることに注意してください。これは状況によって異なることがあります (したがって、以下のコードはユーザーのアプリケーションで即時にコンパイルしない場合があります)。

```
public void setRequestProperties(
   com.ibm.commerce.datatype.TypedProperty arg1)
   throws Exception
   {
      iRequestProperties = arg1;
      try {
          if (// check for nulls
              getDataBeanKeyUserId() == null)
             {
                super.setInitKey_UserId(aUserId);
             }
```

```
        } catch (com.ibm.commerce.exception.ParameterNotFoundException e)
            {
            }
    }
```

アクセス Bean の基本キーを設定するには、他に 2 つの方法があります。たとえば JSP テンプレートなど、データ Bean の外部で行うことができます。この場合は、JSP テンプレートでデータ Bean を活動化する前に、基本キーについてデータ Bean の set メソッドを明示的に呼び出してください。たとえば、 JSP には以下に似たコードを組み込むことができます (ここで db はデータ Bean オブジェクトです)。

```
db.setInitKey_UserId(/*input parameter*/)
db.activate();
```

また、基本キーを直接設定することもできます。つまり、JSP テンプレートは db.activate メソッドしか含まず、データ Bean マネージャーがアクセス Bean で基本キーを明示的に設定します。たとえば、データ Bean の setRequestProperties メソッドのコードは、以下に似たものになります。

```
public void setRequestProperties(
   com.ibm.commerce.datatype.TypedProperty arg1)
   throws Exception
      {
      iRequestProperties = arg1;
         try
             {
                super.setInitKey_UserId(aUserId);
             }
       } catch (com.ibm.commerce.exception.ParameterNotFoundException e)
           {
           }
      }
```

基本キーの設定でお勧めする手順は、ステップ 5 に示した間接的な方法であることに注意してください。

データ Bean の get メソッドの変更: データ Bean を作成してから、生成された get メソッドのいくつかでコードを変更しなければなりません。

get メソッドを更新するには、以下のようにします。

1. 新しいデータ Bean を拡張表示して、フィールドとメソッドを表示します。
2. **getCommandContext()** メソッドを選択して、そのソース・コードを表示します。「ソース」ペインに以下のようなソース・コードが表示されます。

   ```
   public com.ibm.commerce.comand.CommandContext getCommandContext ()
   {
      return null;
   }
   ```

3. メソッドが以下のように表示されるように、ソース・コードを変更します。

```
public com.ibm.commerce.comand.CommandContext getCommandContext ()
{
   return iCommandContext;
}
```

ここまでの作業を保管します (Ctrl + S)。

4. **getRequestProperties()** メソッドを選択して、そのソース・コードを表示します。「ソース」ペインに以下のようなソース・コードが表示されます。

```
public com.ibm.commerce.datatype.TypedProperty setRequestProperties()
{
   return null;
}
```

5. ソース・コードを以下のように変更します。

```
public com.ibm.commerce.datatype.TypedProperty setRequestProperties()
{
   return iRequestProperties;
}
```

ここまでの作業を保管します (Ctrl + S)。

populate() メソッドの変更: 以下のようにして、populate メソッドを変更しなければなりません。

1. 新しいデータ Bean を拡張表示して、フィールドとメソッドを表示します。
2. **populate()** メソッドを選択して、そのソース・コードを表示します。「ソース」ペインに以下のようなソース・コードが表示されます。

```
public void populate () throws Exception {}
```

3. メソッドが以下のように表示されるように、ソース・コードを変更します。

```
public void populate () throws Exception {
   super.refreshCopyHelper();
}
```

ここまでの作業を保管します (Ctrl + S)。

新規セッション Bean の作成

新規セッション Bean を作成する際は、 WebSphere Commerce EJB グループとは別個の EJB グループ内に作成する必要があります。この新規 EJB グループは、 WebSphere Commerce プロジェクトとは別個の新規プロジェクトに保管してください。たとえば、MyCustomBeans EJB グループを、com.mycompany.mycustomcode パッケージ内に作成することができます。独自のカスタマイズ済みコードを WebSphere Commerce コードから分離することにより、将来のリリースへのマイグレーションの影響をできるだけ小さくすることができます。

新規セッション Bean は、com.ibm.commerce.base.helpers.BaseJDBCHelper クラスを拡張するものでなければなりません。このスーパークラスが備えているメソッドを使用すれ

ば、コマース・サーバーが使用するデータ・ソース・オブジェクトから JDBC 接続オブジェクトを取得することができます。その結果、セッション Bean は他のエンティティー Bean と同じトランザクションに参加します。以下のコード例は、このスーパークラスが提供する機能を示しています。

```
public class mySessionBean extends com.ibm.commerce.base.helpers.BaseJDBCHelper
   implements SessionBean {

   public Object myMethod () throws javax.naming.NamingException,
      java.rmi.RemoteException, SQLException {

       ////////////////////////////////////////////////
      //  -- your logic, such as initialization --    //
      ////////////////////////////////////////////////

      try {
            // get a connection from the WebSphere Commerce data source
            makeConnection();
            PreparedStatement stmt = getPreparedStatement(
               "your sql string");
            ////////////////////////////////////////////////////////////
            // -- your logic such as set parameter into the prepared  //
            // statement --                                           //
            ////////////////////////////////////////////////////////////
            ResultSet rs = executeQuery(stmt, false);

            ////////////////////////////////////////////////
            // -- your logic to process the result set -- //
           ////////////////////////////////////////////////

               }
         finally {
             // return the connection to the WebSphere Commerce data source
            closeConnection();
          }

       ////////////////////////////////////////////////////
      // -- your logic to return the result ---          //
      ////////////////////////////////////////////////////

   }

}
```

前述のコード例では、executeQuery メソッドは 2 つの入力パラメーターをとっています。最初のものは準備済みステートメントで、 2 番目のものはキャッシュ・フラッシュ操作に関係したブール・フラグです。クエリーを実行する前にコンテナーがキャッシュから現行トランザクション用のすべてのエンティティー・オブジェクトをフラッシュする必要がある場合は、このフラグを true に設定します。これが必要とされるのは、一部のエンティティー・オブジェクトに対して更新を実行しており、それらの更新済みオ

ブジェクトを検索するためのクエリーが必要な場合です。このフラグを false に設定した場合、エンティティー・オブジェクトに対する更新はトランザクションが終了するまでデータベースに書き込まれません。

このフラッシュ操作の使用は制限する必要があり、本当に必要とされる場合以外には、一般にこのフラグは false に設定してください。フラッシュ操作はリソース集中操作です。

オブジェクトのライフ・サイクル

オブジェクト・モデルの中のエンタープライズ Bean には、*独立* オブジェクトと*従属*オブジェクトの両方が含まれます。独立オブジェクトには独自のライフ・サイクルがあり、呼び出し元のビジネス・ロジックの作成要求または除去要求によって直接制御されます。従属オブジェクトには、*所有者オブジェクト* と呼ばれる他のオブジェクトとのつながりのあるライフサイクルがあります。 (所有者オブジェクトも従属オブジェクトである場合がありますが、アソシエーション階層を登れば、独立オブジェクトが存在するはずです。) 所有者オブジェクトが削除されると、すべての従属オブジェクトは削除されます。実際の削除は、データベース内のカスケード削除仕様によって制御されます。

たとえば、住所録オブジェクトを戻すユーザー・オブジェクトとオーダー・オブジェクトのリストが与えられた場合、ユーザー・オブジェクトが削除されると、(住所録がユーザーに所有されるため) その住所録オブジェクトも削除されます。また、住所録の中のすべての住所オブジェクトも (住所は住所録に所有されるため) 削除されます。しかし、オーダー・オブジェクトは削除されません。なぜなら、オーダーの所有者はストア・オブジェクトであってユーザー・オブジェクトではないからです。

従属オブジェクトの作成には、特定の設計パターンが使用されます。従属オブジェクトの作成メソッドは、所有者オブジェクトへの参照を提供しなければなりません。したがって、従属オブジェクトが作成される前に、まず所有者オブジェクトが存在しなければなりません。

トランザクション

Enterprise JavaBeans V1.0 アーキテクチャーでは、インスタンス状態に関して 3 つのコミット時オプションが指定されています。仕様文書では、これらはオプション A、B、および C として説明されています。これらのオプションに関する完全な詳細については、 Sun Microystem 社の Enterprise JavaBeans V1.0 仕様文書を参照してください。

WebSphere Application Server はオプション A および C をインプリメントしていますが、オプション A はデータベースが共用でないと見なします。

オプション C では、エンタープライズ Bean コンテナーは、「作動可能」インスタンスをトランザクション間でキャッシュに入れません。トランザクションが完了すると、インスタンスはすぐに使用可能インスタンスのプールに戻されます。 WebSphere Commerce では、データベースが複数の WebSphere Commerce アプリケーション間で共

用されるので、オプション C が使用されます。このインプリメンテーションでは、各トランザクションの最初にコンテナーがエンティティー Bean の永続データをロードし、エンティティー Bean はトランザクションの期間だけキャッシュに入れられます。コンテナーは、エンティティーがアクセスされるトランザクションごとに 1 つずつ、エンティティー Bean の複数インスタンスを活動化します。トランザクション同期化はデータベースによって実行されます。

各エンタープライズ Bean のトランザクション属性は TX_REQUIRED に設定されます。Web コントローラーは、 (対応するアクセス Bean を介して) エンタープライズ Bean にアクセスするコマンドを実行する前にトランザクションを開始するので、エンタープライズ Bean のビジネス・メソッドはこのトランザクションのコンテキスト内で呼び出されます。

エンティティー Bean に関するその他の考慮事項

Find for Update

行を更新する目的でデータベースの同じ行に複数のアプリケーションがアクセスできるような状態を、*並行更新* といいます。並行更新が可能な場合もあれば、並行更新がまったく望ましくない場合もあります。

データベースの更新が上書きであれば、新しい値はデータベースの現在の値と関係がないため、並行更新は可能です。並行更新が可能であり、複数のアプリケーションがデータベースの同じ行を更新しようとした場合には、最後の試行が実際のデータベース更新になります。

データベースの更新操作が、データベースの現行値に依存するような場合には、並行更新は望ましくありません。たとえば、アプリケーションが商品在庫を更新する場合、一度に 1 つのアプリケーションのみが更新できるようにする必要があります。

並行更新が可能かどうかに影響を与える要因には、データベース・ロック、およびエンタープライズ Bean 分離レベルがあります。

2 番目のアプリケーションが並行して行を更新するのを防ぐには、最初に行にアクセスするアプリケーションが、 “find for update” オプションを使用して行を取り出す必要があります。 “find for update” オプションが使用されると、*書き込みロック* (または排他ロックともいう) がその行に適用されます。この書き込みロックが行に適用されると、“find for update” を使用してその行にアクセスしようとするアプリケーションはすべてブロックされます。

アプリケーションが並行更新を許可すると、行をロックせずに単にデータを取り出すだけです。

UpdateInventory によってオーダーに含まれているすべての製品を見付け、在庫を適切に更新しなければならない、 OrderProcess シナリオを考えてみましょう。同じ製品が他の

多くのオーダーに含まれていることが考えられるので、 *find for update* を使用する必要があります。デッドロックの可能性を減らすため、トランザクションの有効範囲内でできるだけ早期に使用する必要があります。したがって、 UpdateInventory のアルゴリズムは、以下のような疑似コードで表すことができます。

```
UpdateInventory
 find all the order items in the order
 for each order item
  fetch its inventory using "find for update"
  ...
```

長期間実行されるビジネス間商取引シナリオでは、オーダーに多くのアイテムが含まれる可能性があるため、 find for update をできるだけ早期に使用する必要があります。このロジックは以下のようになります。

```
find for update the inventory of all the products in an order
for each product
 if (total quantity ordered for that product < inventory)
        deduct quantity from inventory
 else
        error
```

フラッシュ・リモート・メソッド

WebSphere Application Server は、エンティティー Bean に加えられた変更内容を、トランザクションのコミット時までデータベースに書き込まないので、データベースは、エンティティー Bean のコンテナーでキャッシュに入れられているデータと一時的に同期しなくなることがあります。

(com.ibm.commerce.base.helpers.BaseJDBCHelper クラス内に) 提供されているフラッシュ・リモート・メソッドは、すべてのトランザクションでコミットされたすべての変更内容を書き込み (つまり、エンタープライズ Bean キャッシュから情報を取得し)、データベースを更新します。このリモート・メソッドは、コマンドによって呼び出すことができます。このメソッドは絶対に必要な場合にのみ使用してください。オーバーヘッド・リソースの点で消費が大きいため、パフォーマンスによくない影響を与えます。

以下のコードの断片を含むログオン・コマンドについて考えてみましょう。

```
UserAccessBean uab = ...;
 uab.setRegisteredTimestamp(currentTimestamp);
 uab.commitCopyHelper();
```

トランザクションがコミットされる前、 USER テーブル内の REGISTEREDSTAMP は、現在のタイム・スタンプで更新されていません。更新は、トランザクションのコミット時まで起こりません。そのため、フラッシュ・メソッドを使用して、 (同じトランザクション内の) 直接 JDBC クエリー (たとえば、 *select from user where registeredstamp ...*) が、ユーザーに指定した登録タイム・スタンプを戻すようにする必要があります。

エンタープライズ Bean のセキュリティー確保

エンタープライズ Bean のセキュリティーを確保するために WebSphere Application Server を使用している場合には、WebSphere Application Server 管理コンソールを使用して、すべての新しいエンタープライズ Bean のメソッドをセキュリティー・メソッド・グループ WCSMethodGroup に割り当てる必要があります。このステップは、新しいエンタープライズ Bean のデプロイメントを実行するときに実行してください。さらに、既存の WebSphere Commerce エンティティー Bean を変更する場合は、影響を受ける EJB グループ内のすべてのエンティティー Bean のメソッドを WCSMethodGroup セキュリティー・メソッド・グループに割り当てる必要があります。カスタマイズ・コードのデプロイメント・プロセスについては、193 ページの『コードのデプロイメント』を参照してください。

基本キー

基本キーとは、テーブルの定義の一部になる固有鍵のことです。これを使用して、レコード同士を区別できます。すべてのレコードに基本キーがなければなりません。テーブル中に新しいレコードを作成する際には、そのレコード用に固有の基本キーを生成する必要が生じることがあります。

WebSphere Commerce プログラミング・モデルでは、永続層に、データベースと対話するエンティティー bean が組み込まれます。そのため、エンティティー Bean がインスタンス化される際に、データベース・レコードが作成される場合があります。したがって、新しいレコード用に基本キーを生成するロジックを組み込むには、エンティティー Bean のインスタンス化を行う ejbCreate メソッドが必要になる場合があります。

アプリケーションでデータベースからの情報が必要になる場合は、エンティティー Bean の対応するアクセス Bean がインスタンス化されてから、さまざまなフィールドの get や set が行われることにより、間接的にエンティティー Bean が使用されます。データベース中の特定レコード (特定ユーザー・プロファイルなど) に関するアクセス Bean がインスタンス化され、基本キーを使用してデータベースから正しい情報が選択されます。

固有の基本キーを作成する方法と、基本キーによって選択する方法について、以下に説明します。

基本キーの作成: エンティティー Bean の新規インスタンスをインスタンス化するには、 ejbCreate メソッドを使用します。このメソッドは自動的に生成されますが、生成されたメソッドには、基本キーを静的な値に初期化するロジックしか組み込まれていません。

基本キーが新しく、固有値であることを確認する必要が生じることがあります。この場合、 ejbCreate メソッドを以下のコードの断片のようにすることができます。

```
Public void ejbCreate(int argMyOtherValue)
   throws javax.ejb.CreateException,
   java.rmi.RemoteException {
```

```
        //Initialize CMP fields
        MyKeyValue = com.ibm.commerce.key.ECKeyManager.
           singleton().getNextKey("table_name");
        MyOtherValue = argMyOtherValue;
}
```

上記のコードの断片で、 getNextKey メソッドにより基本キーの固有の整数が生成されます。このメソッドの *table_name* 入力パラメーターは、 KEYS テーブル中に定義されている TABLENAME 値と正確に一致していなければなりません。文字と、その文字が大文字小文字のどちらであるかが正確に一致しているか確認してください。

ejbCreate メソッドに上記のコードを組み込むのに加えて、 KEYS テーブルにもエントリーを作成しなければなりません。以下は、KEYS テーブルにエントリーを作成するための SQL ステートメントの例です。

```
insert into KEYS (TABLENAME, COUNTER, KEYS_ID)
   values ("table_name", 0, 1)
```

上記の SQL ステートメントで、 KEYS テーブル内の他の列のデフォルト値が受け入れられることに注意してください。 COUNTER の値は、カウントが開始する値を示します。 KEYS_ID の値は正の値にしてください。

基本キーを長データ・タイプ (DB2 の場合は BIGINT、 Oracle の場合は NUMBER) として定義する場合は、getNextKeyAsLong メソッドを使用してください。

基本キーによる選択: アクセス Bean 中で、基本キーを使用して該当するデータベース・レコードを選択しなければなりません。以下のコードの断片は、この選択を実行する方法を示しています。また、追加のロジックも含まれています。これについては後で説明します。

```
UserProfileAccessBean abUserProfile = new UserProfileAccessBean();
abUserProfile.setInitKey_UserId(getUserId().toString());
abUserProfile.refreshCopyHelper();
```

上記のコードの断片の先頭行は、 ″abUserProfile″ という新しい UserProfileAccessBean をインスタンス化します。 2 行目は、アクセス Bean 中に基本キーを設定します。 VisualAge for Java で setInitKey_*xxx* (*xxx* は基本キーのフィールド名) という命名規則が使用され、基本キーの set メソッドに名前が付けられます。アクセス Bean をインスタンス化する際には、 setInitKey_xxx メソッドによって設定されたすべてのフィールドが初期化されていることを確認してから、 refreshCopyHelper メソッドを使用する必要があります。 setInitKey_xxx メソッドが呼び出される順序は重要ではありません。

setInitKey_xxx メソッドがすべて呼び出されたら、すべての必須フィールドが初期化されているので、 refreshCopyHelper メソッドを使用してデータベースから情報を検索できます。

アクセス Bean のローカル・キャッシュ中の値を更新する場合は、 commitCopyHelper 呼び出しも組み込んで、更新した情報でデータベースも更新しなければなりません。た

とえば、refreshCopyHelper メソッドを使用してデータを検索した後で、顧客の名前を更新した (名前値を設定して) 場合は、 `abUserProfile.commitCopyHelper()` を呼び出して、新しい情報でデータベースを更新してください。

エンティティー Bean の使用

エンタープライズ Bean を使用するプログラムは、エンタープライズ Bean のホームおよびリモート・インターフェースに加えて、 Java Naming and Directory Interface (JNDI) を扱う必要があります。プログラミング・モデルを単純化するため、各エンタープライズ Bean ごとにアクセス Bean が生成されます。独自のエンタープライズ Bean を作成する際は、 VisualAge for Java のツールを使用してこのアクセス Bean を生成してください。

WebSphere Commerce コマンドは、エンティティー Bean と直接対話する代わりに、アクセス Bean と対話します。図が示しているように、アクセス Bean を使用することには以下のような利点があります。

- より単純なプログラミング・インターフェース。アクセス Bean は Java bean と同じように動作し、エンタープライズ Bean 固有のプログラミング・インターフェース (JNDI、クライアントからのホームおよびリモート・インターフェースなど) をすべて隠します。
- 実行時には、アクセス Bean はエンタープライズ Bean をキャッシュに入れます。これは、ホーム・オブジェクトの参照が時間やリソース使用の観点ではコスト高であるためです。
- アクセス Bean がインプリメントする copyHelper オブジェクトは、コマンドがエンタープライズ Bean 属性を取得したり設定したりするときにエンタープライズ Bean への呼び出しの数を減らします。したがって、必要なエンタープライズ Bean への呼び出しは 1 つだけになり、これによって複数のエンタープライズ Bean 属性の読み書きを行えます。

以下の図は、コマンド、アクセス Bean、エンティティー Bean およびデータベースの間の対話を示しています。

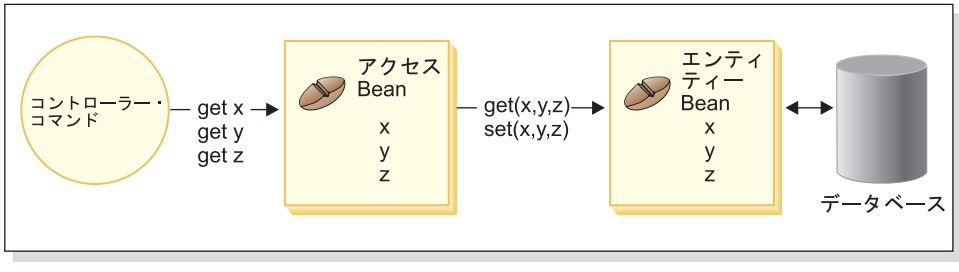


図 18.

データベースに関する考慮事項

e-commerce アプリケーションをカスタマイズするときには、新しいデータベース・テーブルを作成することができます。これらのテーブルを作成する場合、 WebSphere Commerce テーブルと整合性のある方法でテーブルを作成するために、一連の規則に従うことをお勧めします。

データベース・スキーマ・オブジェクト命名に関する考慮事項

以下のセクションでは、データベース・スキーマ・オブジェクトの命名に関する指針を示します。

テーブルおよびビューの命名規則

以下のリストは、新しいテーブルおよびビューを命名する際の指針です。

- 将来のリリースの WebSphere Commerce テーブル名およびビュー名との衝突 (重複名) を避けるために、テーブル名またはビュー名の最初の文字を X としてください (たとえば XMYTABLE)。
- テーブル名またはビュー名の長さは 10 文字を超えてはなりません。希望の名前がこの制限を超える場合には、名前の末尾の方から母音を削除していき、10 文字まで短縮してください。
- テーブル名またはビュー名には、“_”、“+”、“$”、“%”、またはブランク・スペースなどの特殊文字を含めないでください。
- データベース予約語をテーブル名またはビュー名に使用しないでください。
- ビュー名の末尾は VW にしてください。
- テーブル名およびビュー名は単数名詞にしてください。

列の命名規則

以下のリストは、新しいテーブル内の列を命名する際の指針です。

- 将来のリリースの WebSphere Commerce テーブルの列名との衝突 (重複名) を避けるために、列名の最初の文字を X としてください (たとえば XMYCOLUMN)。
- 列名の長さは 18 文字を超えてはなりません。希望の名前がこの制限を超える場合には、名前の末尾の方から母音を削除していき、18 文字まで短縮してください。
- 列名 (外部キー以外) には、“_”、“+”、“$”、“%”、またはブランク・スペースなどの特殊文字を含めないでください。
- データベース予約語を列名に使用しないでください。
- アクティブな音声の組み合わせを使用して、結合された語が列名として使用されることがあります。たとえば COMBINERESULT です。
- 生成された基本キー列は、*table*_id という名前にします。たとえば、USERS テーブルの基本キーは USERS_ID とします。

- 生成された外部鍵列の名前は、変更してはなりません。
- 将来のカスタマイズのためにいくつかの列を予約する場合には、それらの列を field*x* という名前にします (*x* は 1 から始まる数字)。

索引の命名規則

以下のリストは、新しいテーブル内の索引を命名する際の指針です。

- 索引名の長さは 18 文字を超えてはなりません。
- 索引名にブランク・スペースを含めることはできません。
- 索引名にデータベース予約語を含めることはできません。
- 非固有索引の名前は I_*tablex* とします。ここで、*table* はテーブル名、*x* は 1 から始まる数字です。たとえば、USERS テーブルの非固有索引は I_USERS1 という名前にします。
- 固有索引の名前は UI_*tablex* とします。ここで、*table* はテーブル名、 *x* は 1 から始まる数字です。たとえば、USERS テーブルの固有索引は UI_USERS1 という名前にします。
- 索引の合計サイズは 254 バイトを超えてはなりません。
- 索引名はデータベース・スキーマ全体において固有でなければなりません。

基本キーの命名規則

以下のリストは、新しいテーブル内の基本キーを命名する際の指針です。

- 基本キー名の長さは 18 文字を超えてはなりません。
- 基本キー名にブランク・スペースを含めることはできません。
- 基本キー名にデータベース予約語を含めることはできません。
- 基本キーの名前は P_*table* とします (*table* はテーブルの名前)。たとえば、USERS テーブルの基本キーは P_USERS とします。
- 基本キー名はデータベース・スキーマ全体において固有でなければなりません。

外部キーの命名規則

以下のリストは、新しいテーブル内の外部キーを命名する際の指針です。

- 外部鍵名の長さは 18 文字を超えてはなりません。
- 外部鍵名にブランク・スペースを含めることはできません。
- 外部鍵名にデータベース予約語を含めることはできません。
- 外部鍵の名前は F_*table* とします (*table* はテーブルの名前)。たとえば、USERS テーブルの外部鍵は F_USERS1 とします。
- 外部鍵名はデータベース・スキーマ全体において固有でなければなりません。

データベース・トリガーの命名規則

以下のリストは、データベース・トリガーを命名する際の指針です。

- データベース・トリガー名の長さは 18 文字を超えてはなりません。

- データベース・トリガー名にブランク・スペースを含めることはできません。
- データベース・トリガー名にデータベース予約語を含めることはできません。
- データベース・トリガーの名前は `T_table` とします (*table* はテーブルの名前)。たとえば、USERS テーブルのデータベース・トリガーの名前は `T_USERS1` とします。
- データベース・トリガー名はデータベース・スキーマ全体において固有でなければなりません。

データベース列のデータ・タイプに関する考慮事項

このセクションでは、新しいテーブルを作成する際の列のデータ・タイプについて説明します。以下のさまざまなデータ・タイプに関する説明では、 DB2 の用語を使用します。それ以外のデータベースを使う場合の相違点は、 88 ページの『さまざまなデータベース間でのデータ・タイプの違い』で示しています。

BIGINT 64 ビット符号付き整数で、 -9223372036854775807 ～ 9223372036854775807 の範囲。なお、INTEGER は BIGINT の半分のサイズしかありません。

INTEGER

32 ビット符号付き整数で、-2147483647 ～ 2147483647 までの範囲。一般に、BIGINT ではなく、INTEGER がデフォルトの有限数値データ・タイプでなければなりません。 BIGINT を使用するビジネス上の強い理由がない限り、パフォーマンス上の理由から、数値データ・タイプとして INTEGER を使用する方が有利です。 BIGINT データ・タイプは、一般にシステム生成鍵によって使用されます。

SMALLINT または SHORT データ・タイプは絶対に使用しないでください。これらのデータ・タイプは非オブジェクト Java データ・タイプにマップされ、それらの非オブジェクトのデータ・タイプがエンタープライズ Bean オブジェクトのインスタンス化で問題を起こすことがあります。

TIMESTAMP

7 つの部分 (年、月、日、時、分、秒、およびマイクロ秒) からなる値で、日付と時刻を指定します。ただし時刻の指定はマイクロ秒単位で断続的です。タイム・スタンプの内部表記は 10 バイトのストリングで、それぞれには 2 つのパック 10 進数が含まれています。最初の 4 バイトは日付、次の 3 バイトは時刻、最後の 3 バイトはマイクロ秒をそれぞれ表します。

CHAR 固定長の文字ストリングで、長さ INTEGER の範囲は 1 文字から 254 文字までです。長さの指定を省略した場合、1 文字の長さが想定されます。 CHAR は固定長データベース列であるため、末尾の未使用の文字スペースは空白文字に変換されます。 CHAR データ・タイプは柔軟でなく、後で長さ変えることができないため、パフォーマンス上の理由がある場合を除いては、CHAR の使用は推奨されません。経験則としては、長さ 64 文字未満のストリング列で、頻繁に検索または更新されるような場合には、 CHAR を使用するとパフォーマンスが改善されます。

VARCHAR

可変長の文字ストリングで、最大長は 1 ～ 32672 の範囲の整数です。ただし、列データがテーブルとともに保管される CHAR とは異なり、 VARCHAR は内部的にはデータベース・ページ内の参照ポインターとして表されます。したがって、VARCHAR 列の長さは、作成後にいつでも変更することができます。

LONG VARCHAR

可変長の文字ストリングで、同じデータベース・ページ内で VARCHAR を作成できない場合に使用することができます。 LONG VARCHAR は、複数のデータベース・ページにまたがることが可能である点を除いて、VARCHAR とよく似ています。 LONG VARCHAR オブジェクトは一般にパフォーマンスの点でコストがかかるので、どうしても必要な場合を除いて、LONG VARCHAR データ・タイプは使用しないでください。

CLOB これも可変長の文字ストリングで、 LONG VARCHAR の制限 32 KB を超える長さの列が必要な場合に、これを使用できます。 CLOB オブジェクトの長さは、データベース構成を変えずに最大 1 GB まで指定することができます。CLOB として保管されるテキスト・データは、異なるシステム間で移動する場合、適切に変換されます。

BLOB 非構造化データをデータベース内に保管する可変長バイナリー・文字ストリングです。 BLOB オブジェクトは、4 GB までのバイナリー・データを保管することができます。どうしても必要な場合を除き、通常は列のデータ・タイプとして BLOB を使用しないでください。パフォーマンスの点では、BLOB オブジェクトはどのデータベースでも最もコストのかかるオブジェクトの 1 つです。

DECIMAL(20,5)

このデータ・タイプは、(たとえば通貨などの) ほとんどの固定小数点数値を扱うために特別に定義されています。その他の浮動小数点数値には、代わりに FLOAT を使用することができます。

さまざまなデータベース間でのデータ・タイプの違い

以下の行では、WebSphere Commerce データベース・スキーマのデータ・タイプを一覧表示し、さまざまなデータベース・インプリメンテーションでそれに対応するデータ・タイプを示しています。

JDBC オブジェクト	Windows AIX Solaris Linux DB2	Windows AIX Solaris Oracle®	400 DB2
ハッシュ・テーブル	BLOB()	BLOB	BLOB()
タイム・スタンプ	TIMESTAMP	DATE	TIMESTAMP
整数	INTEGER	INTEGER	INTEGER
BigDecimal	DECIMAL(,)	DECIMAL(,)	DECIMAL(,)
Long	BIGINT	NUMBER	BIGINT
Double	FLOAT	NUMBER	FLOAT
ストリング	CHAR()	VARCHAR2()	GRAPHIC() CCSID 13488
byte[]	CHAR() (ビット・データ用)	RAW()	CHAR() (ビット・データ用)
ストリング	VARCHAR()	VARCHAR2()	VARGRAPHIC() CCSID 13488
ストリング	LONG VARCHAR	VARCHAR2() (詳しくは、表の下の注を参照。)	VARGRAPHIC(4000) ALLOCATE() CCSID 13488
byte[]	LONG VARCHAR (ビット・データ用)	LONG RAW	VARCHAR(8000) ALLOCATE() (ビット・データ用)
ストリング	CLOB()	CLOB()	DBCLOB() CCSID 13488

注:

Oracle JDBC ドライバーが LONG データ・タイプの情報を処理するときの成功率に矛盾があるため、できる限り LONG データ・タイプの使用を避けることをお勧めします。この状況で一番多く報告されるエラーは、「ストリームがすでにクローズされています (Stream has already been closed)」というエラーです。

このデータ・タイプを使用する必要がある場合は、 LONG タイプを使用するデータベース・テーブルごとに 1 列しか使用できません。また、Select ステートメント

を組み立てるとき、 Select の最初または最後のエレメントとして LONG 列を入れないでください。負荷が重い場合の操作の暫定措置として、エンティティー Bean の CMP フィールドへのこの特定の列のマッピングを避けるという方法もあります。この列で検索や更新を実行するには、代わりに セッション Bean を使用してください。

第 4 章 アクセス制御

アクセス制御の理解

WebSphere Commerce アプリケーションのアクセス制御モデルには 3 つの主な概念、すなわちユーザー、アクション、およびリソースがあります。ユーザーは、システムを使用する人間です。リソースは、アプリケーション内で、またはアプリケーションによって保守されるエンティティーです。たとえばリソースには、商品、文書、オーダーなどがあります。人間を表すユーザー・プロファイルもリソースです。アクションは、ユーザーがリソースで実行できるアクティビティーです。アクセス制御とは、特定のユーザーが特定のリソースで特定のアクションを実行できるかどうかを決定する、 e-commerce アプリケーションのコンポーネントです。

WebSphere Commerce アプリケーションでは、主要な 2 つのレベルのアクセス制御があります。アクセス制御の第 1 レベルは WebSphere Application Server によって実行されます。ここでは、WebSphere Commerce が WebSphere Application Server を使用してエンタープライズ Bean およびサーブレットを保護します。アクセス制御の 2 次レベルは、 WebSphere Commerce のきめ細かいアクセス制御システムです。

WebSphere Commerce アクセス制御フレームワークでは、アクセス制御ポリシーを使用して、特定のユーザーが特定のリソースで特定のアクションの実行を許可されているかどうかを判別します。このアクセス制御のフレームワークでは、きめ細かいアクセス制御が提供されます。 WebSphere Application Server によって提供されるアクセス制御とともに作業しますが、これに代わるものではありません。

WebSphere Application Server でのリソース保護の概要

以下の WebSphere Commerce リソースは、WebSphere Application Server によるアクセス制御の下で保護されます。

- エンティティー Bean
  これらの bean は、e-commerce アプリケーション内のオブジェクトをモデル化します。これらは、リモート・クライアントからアクセスできる分散オブジェクトです。
- JSP テンプレート
  WebSphere Commerce は、表示ページに JSP テンプレートを使用します。各 JSP テンプレートには、データをエンティティー Bean から検索する 1 つまたは複数のデータ Bean を含めることができます。クライアントは、URL 要求を構成することによって JSP ページを要求することができます。
- コントローラーおよびビュー・コマンド
  クライアントは、URL 要求を構成することによってコントローラーおよびビュー・コマンドを要求することができます。加えて、 VIEWREG テーブルで登録されてい

る JSP ファイル名またはビュー名を使用することにより、 1 つの表示ページに他の表示ページへのリンクを含めることができます。

通常、WebSphere Commerce Server は、以下の Web パスを使用するように構成されています。

- `/webapp/wcs/stores/servlet/*`
  これは、要求サーブレットの要求に使用します。
- `/webapp/wcs/stores/*.jsp`
  これは、JSP サーブレットの要求に使用します。

以下の図は、上記の Web パス構成の場合に要求が WebSphere Commerce リソースにアクセスする際に潜在的にたどる可能性のある経路を示しています。

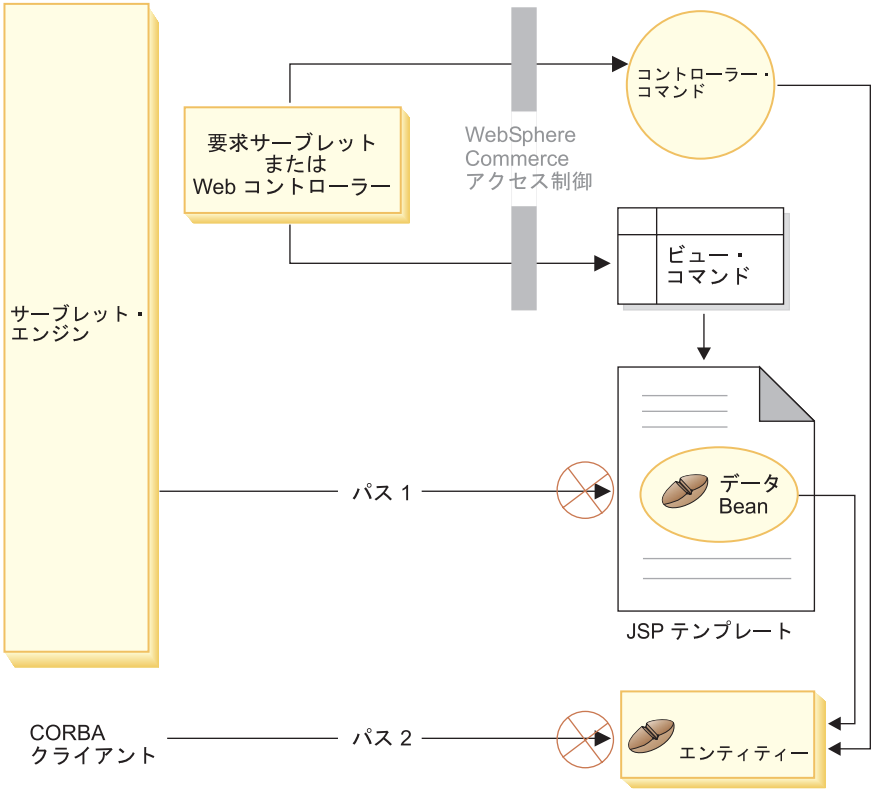


図 19.

正当な要求はすべて、要求サーブレットに送られる必要があります。次いで、要求サーブレットは、これを Web コントローラーに送ります。 Web コントローラーは、コントローラー・コマンドおよびビューのためのアクセス制御をインプリメントしています。しかしながら、上記の Web パスでは、悪質なユーザーが JSP テンプレート (パス

1) やエンティティー Bean (パス 2) に直接アクセスすることができます。これらの悪質なアタックが成功しないようにするため、実行時に拒否することが必要です。

JSP テンプレートとエンティティー Bean への直接アクセスは、以下のいずれかの方法によって防止することができます。

WebSphere Application Server セキュリティー

WebSphere Application Server はセキュリティー機能を備えています。この方法を使用すれば、すべてのエンタープライズ Bean メソッドと JSP テンプレートは、 System Identity だけが呼び出すように構成されます。これらの WebSphere Commerce リソースにアクセスするには、 URL 要求を、Web コントローラーに渡す前に、 System Identity を現行スレッドに設定する要求サーブレットに経路指定する必要があります。次いで、Web コントローラーは、要求を対応するコントローラー・コマンドまたはビューに渡す前に、呼び出し元が必要な許可を持っているかどうかを確かめます。 JSP テンプレートやエンティティー Bean に直接 (つまり、 Web コントローラーを使用せずに) アクセスしようとする試みはすべて、 WebSphere Application Server セキュリティー・コンポーネントによって拒否されます。

WebSphere Commerce リソースを保護するための WebSphere Application Server 構成については、 *WebSphere Commerce インストール・ガイド* を参照してください。 WebSphere Application Server 内のセキュリティーについては、 WebSphere Application Server 資料のシステム管理のトピックを参照してください。

カスタマイズされたエンタープライズ Bean でメソッドの WebSphere Application Server セキュリティーを構成することについては、 374 ページの『新しいエンタープライズ Bean のエンタープライズ・アプリケーションへのアセンブル』および 380 ページの 『変更されたエンタープライズ Bean のエンタープライズ・アプリケーションへのアセンブル』を参照してください。

ファイアウォール保護

WebSphere Commerce Server がファイアウォールの背後で稼働していると、インターネット・クライアントはエンティティー Bean に直接アクセスできません。この方法を使用する場合、 JSP テンプレートの保護は、ページに組み込まれたデータ Bean によって提供されます。データ Bean は、データ Bean マネージャーによって活動化されます。データ Bean マネージャーは、 JSP テンプレートがビュー・コマンドによって転送されたかどうかを調べます。これがビュー・コマンドによって転送されなかった場合は、例外が戻され、JSP テンプレートの要求は拒否されます。

WebSphere Commerce アクセス制御ポリシーの概要

WebSphere Commerce アクセス制御モデルは、アクセス制御ポリシーの制約に基づいています。アクセス制御ポリシーでは、アクセス制御規則をビジネス・ロジック・コード

から外部化することができるため、アクセス制御ステートメントをコードにハードコーディングする必要がなくなります。たとえば、以下のようなコードを組み込む必要はありません。

```
if (user.isAdministrator())
   then {}
```

アクセス制御ポリシーは、アクセス制御ポリシー・マネージャーによって実行されます。一般に、保護されたリソースにユーザーがアクセスしようとすると、アクセス制御ポリシー・マネージャーは最初に、その保護されたリソースに適用できるアクセス制御ポリシーを決定し、それからその適用できるアクセス制御ポリシーに基づいて、要求されたリソースへのユーザーのアクセスを許可するかどうかを決定します。

アクセス制御ポリシーは、 ACPOLICY テーブルに保管される 4 タプルのポリシーです。アクセス制御ポリシーはそれぞれ以下の形式をとります。

```
AccessControlPolicy [UserGroup, ActionGroup, ResourceGroup, Relationship]
```

4 タプルのアクセス制御ポリシー内のエレメントは、特定のユーザー・グループに属しているユーザーがリソースについて関係または関係グループで指定された条件を満たす限り、そのユーザーが指定されたリソース・グループに属しているリソースに対して、指定されたアクション・グループのアクションの実行を許可されることを指定します。たとえば、`[AllUsers, UpdateDoc, doc, creator]` は、文書の作成者であれば、すべてのユーザーが文書を更新できることを指定します。

ユーザー・グループは、MBRGRP データベース・テーブル内で定義される、特定のタイプのメンバー・グループです。ユーザー・グループは、メンバー・グループ・タイプ `-2` と関連している必要があります。 `-2` という値はアクセス・グループを表し、MBRGRPTYPE テーブルで定義されます。ユーザー・グループとメンバー・グループ・タイプのアソシエーションは、MBRGRPUSG テーブルに保管されます。

特定ユーザー・グループへのユーザーのメンバーシップは、明示的または暗黙的に記述されます。明示的に指定されるのは、ユーザーが特定のメンバー・グループに属していると MBRGRPMBR テーブルに記述される場合です。暗黙的に指定されるのは、MBRGRPCOND テーブルに記述された条件 (たとえば、プロダクト・マネージャーの役割を果たすすべてのユーザー) を、ユーザーが満たす場合です。複合条件 (たとえば、プロダクト・マネージャーの役割を果たし、最低 6 か月間はその役割にあったすべてのユーザー) または明示的な排他も使用できます。

ユーザーをユーザー・グループに組み込むための条件のほとんどは、特定の役割を果たすユーザーに基づきます。たとえば、プロダクト・マネージャーの役割を果たすすべてのユーザーにカタログ管理操作を実行する許可を与えるアクセス制御ポリシーがあるとします。この場合、MBRROLE テーブルでプロダクト・マネージャーの役割を割り当てられたユーザーはすべて、暗黙的にユーザー・グループに組み込まれます。

メンバー・グループのサブシステムについての詳細情報については、 WebSphere Commerce のオンライン・ヘルプを参照してください。

ActionGroup エレメントは ACACTGRP テーブルからとられます。アクション・グループは、明示的に指定されたアクションのグループを参照します。アクションのリストは ACACTION テーブルに保管され、各アクションとアクション・グループ (単数または複数) の関係は ACACTACTGP テーブルに保管されます。アクション・グループの例としては、 ″OrderWriteCommands″ アクション・グループがあります。このアクション・グループには、オーダーの更新に使用される以下のアクションが組み込まれています。

- com.ibm.commerce.order.commands.OrderDeleteCmd
- com.ibm.commerce.order.commands.OrderCancelCmd
- com.ibm.commerce.order.commands.OrderProfileUpateCmd
- com.ibm.commerce.order.commands.OrderUnlockCmd
- com.ibm.commerce.order.commands.OrderScheduleCmd
- com.ibm.commerce.order.commands.ScheduledOrderCancelCmd
- com.ibm.commerce.order.commands.ScheduledOrderProcessCmd
- com.ibm.commerce.order.commands.OrderItemAddCmd
- com.ibm.commerce.order.commands.OrderItemDeleteCmd
- com.ibm.commerce.order.commands.OrderItemUpdateCmd
- com.ibm.commerce.order.commands.PayResetPMCmd

リソース・グループは、特定のタイプのリソースをグループ化するメカニズムです。リソース・グループ内のリソースのメンバーシップは、次の 2 つの方法のいずれかで指定できます。

- ACRESGRP テーブルの条件列を使用する
- ACRESGPRES テーブルを使用する

ほとんどの場合、リソースをリソース・グループに関連付けるには、 ACRESGPRES テーブルを使用すれば十分です。この方法を使用すると、リソースは Java クラス名を使用して ACRESCGRY テーブルで定義されます。次に、これらのリソースは、ACRESGPRES アソシエーション・テーブルを使用して、適切なリソース・グループ (ACRESGRP テーブル) に関連付けられます。 Java クラス名だけではリソース・グループのメンバーを定義するのに十分でない場合 (たとえば、リソースの属性に基づいてこのクラスのオブジェクトをさらに限定する必要がある場合)、 ACRESGRP テーブルの条件列を使用してリソース・グループをすべて定義することができます。属性に基づいてこのようなリソースのグループ化を実行するには、リソースの Groupable インターフェースをインプリメントしていなければならないことに注意してください。

以下の図は、リソースのグループ化を指定する例を示しています。この例で、リソース・グループ 10023 には、 ACRESGPRES テーブル内でそれと関連付けられるすべて

のリソースが組み込まれています。リソース・グループ 10070 は、 ACRESGRP テーブルの条件フィールド列を使用して定義されています。このリソース・グループには Order リモート・インターフェースのインスタンスが組み込まれ、これには status = ″Z″ (共用要求リストを指定する) も含まれています。

注: ACRESGRP テーブルの条件列に関する XML 情報の詳細については、 *WebSphere アクセス・コントロール・ガイド* で説明されています。

ACRESGRP

AcResGrp_Id	GrpName	条件
10023	AccountRepresentatives CmdResourceGroup	ヌル
10070	SharedRequisitionList ResourceGroup	<profile> <andListCondition> <simpleCondition> <variable name="Status"/> <operator name="="/> <value data="Z"/> </simpleCondition> <simpleCondition> <variable name="classname"/> <operator name="="/> <value data="com.ibm.commerce.order.objects.Order"/> </simpleCondition> </andListCondition> </profile>

ACRESGRPES

AcResGrp_Id	AcResCgry_Id
10023	10246
10023	10247
10023	10248
10023	10249
10023	10250

ACRESCGRY

AcResCgry_Id	ResClassname
10246	com.ibm.commerce.contract.commands.ContractCreateCmd
10247	com.ibm.commerce.contract.commands.ContractCreateCmd
10248	com.ibm.commerce.contract.commands.ContractCreateCmd
10249	com.ibm.commerce.contract.commands.ContractCreateCmd
10250	com.ibm.commerce.contract.commands.ContractCreateCmd

図 20.

ACACTGRP、ACRESGRP、および ACRELGRP テーブルの MEMBER_ID 列の値は、 -2001 (ルート組織) でなければなりません。

アクセス制御ポリシーには、任意で 4 番目のエレメントとして Relationship または RelationshipGroup エレメントを含めることもできます。

アクセス制御ポリシーが Relationship エレメントを使用する場合、それは ACRELATION テーブルからとられます。一方、RelationshipGroup エレメントが含まれる場合、それは ACRELGRP テーブルからとられます。どちらも含める必要はありませんが、一方を含める場合には、他方を含めることはできないことに注意してください。ACRELGRP からとられる RelationshipGroup 仕様は、 ACRELATION テーブルからとられる Relationship より優先されます。

ACRELATION テーブルは、ユーザーとリソースの間に存在する関係のタイプを指定します。関係のタイプの例として、作成者、送信者、および所有者があります。relationship エレメントの使用例には、このエレメントを使用して、オーダーの作成者が常にオーダーを更新できるようにしておくことなどがあります。

ACRELGRP テーブルは、特定のリソースと関連付けることができる関係グループのタイプを指定します。関係グループは、1 つ以上の関係チェーンをグループ化したものです。関係チェーンとは、1 つ以上の関係の系列です。関係グループの例として、ユーザーがリソースの作成者でなければならないこと、さらにリソースで参照される購買組織エンティティーに属していなければならないことを指定することがあります。

関係グループ (または関係) の指定は、アクセス制御ポリシーの任意の部分です。これは、独自のコマンドを作成しており、それらのコマンドが特定の役割に制限されていない場合に、共通して使用されます。これらの場合、ユーザーとリソースの間の関係を強制することができます。一般に、コマンドが特定の役割に制限されている場合は、Relationship エレメントを使用するのではなく、アクセス制御ポリシーの UserGroup エレメントを使って実行されます。

アクセス制御ポリシーに関連するもう 1 つの重要な概念に、アクセス制御ポリシー*所有者* の概念があります。アクセス制御ポリシー所有者は、アクセス制御ポリシーを所有する、組織のエンティティーです。アクセス制御ポリシーは、アクセス制御ポリシー所有者が所有するリソースにしか適用できないため、アクセス制御ポリシー所有者を認識することが重要です。

問題となっているリソースごとに、アクセス制御ポリシー・マネージャーが、メンバー階層内の所有組織エンティティーまたはその上位の組織エンティティーによって所有されるアクセス制御ポリシーを適用します。これは、許可を与えるポリシーが見つかるか、またはすべてのポリシーが検査されてどれからも許可が得られないことが分かるまで続けられます。

メンバー階層を示す以下の図について考えてみます。

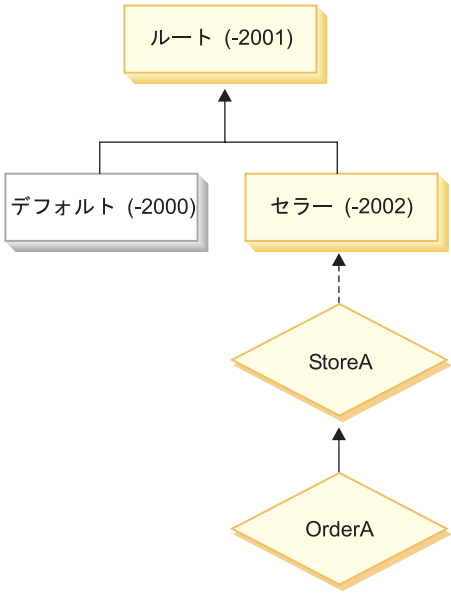


図 21.

リソース “OrderA” について、セラー組織またはルート組織が所有するすべてのアクセス制御ポリシーを適用できます。アクセス制御ポリシー・マネージャーは、これらの組織のどちらかによって所有された、(アクセス制御ポリシー内の 4 つのエレメントに基づいた) ユーザー許可を認可するポリシーを 1 つ検出した時点で、アクセス制御ポリシーの検索を即時に停止します。しかし、これらの組織によって所有されたすべてのアクセス制御ポリシーの中で、保護リソースでアクションを実行するためのユーザー許可を与えるものを検出できない場合は、アクセスは拒否されます。

関係グループ

関係グループを使用して、複数の関係を指定できます。関係は、ユーザーとリソースの間に直接指定することもできますし、ユーザーをリソースに間接的に関連付ける関係のチェーンにすることもできます。

注: 関係グループに関係した以下のセクションでは、 WebSphere Commerce Professional Edition で使用可能な組織だけが RootOrganization、DefaultOrganization、および SellerOrganization であることを認識することが重要です。他の組織を参照する例は、WebSphere Commerce Business Edition だけに適用されます。

関係と関係グループの比較: アクセス制御ポリシーでは、アクセスしているリソースについてユーザーが特定の関係を実現しなければならないことを指定できます。または、ユーザーが関係グループ内で指定されている条件を実現しなければならないことを指定できます。

たいていの場合、関係を指定する際には、ご使用のアプリケーションに該当するアクセス制御要件を満たしていなければなりません。ただし、ユーザーとリソースが直接結び付いていない関係を指定することを規定したポリシーの場合に、実際にはそのユーザーとリソースの間に一連の関係があれば、リソース・グループを使用する必要があります。

たとえば、ユーザーと購買組織の間のアソシエーションを指定しなければならない場合に、その関係においてユーザーがその組織の特定の役割を果たしていること、またはユーザーが購買組織のメンバーであることが必要であれば、関係グループおよび関係のチェーンを使用しなければなりません。

単にユーザーとリソースの間に直接存在するアソシエーションを強制すればよい場合には、単純な関係を使用することができます。たとえば、ユーザーがリソースの作成者でなければならないことを強制する必要がある場合などです。

複数の単純な関係を結合する場合、たとえばユーザーが作成者または 送信者でなければならない場合には、これが関係のチェーンになり、関係グループを作成する必要があります。こうした単純な関係の結合は、WebSphere Commerce Professional Edition または WebSphere Commerce Business Edition を使用する際に生じることがあります。

関係グループに関する一般情報: 関係チェーンとは、1 つ以上の関係の系列です。関係チェーンの長さは、そこに含まれる関係の数によって決定されます。これを決定するには、関係チェーンの XML 表記の `<parameter name="aName" value="aValue" />` エントリーの数を調べます。

最後の `<parameter name="Relationship" value="aValue" />` エレメントだけがリソースの fulfills() メソッドによって処理されなければなりません。残りはアクセス制御ポリシー・マネージャーによって内部的に処理されます。

関係チェーンの長さが 2 の場合、最初の `<parameter name="aName" value="aValue" />` エレメントはユーザーと組織エンティティーの間にあります。最後の `<parameter name="aName" value="aValue" />` エレメントは組織エンティティーとリソースの間にあります。

関係グループを定義する必要がある場合、 XML ファイル内で関係グループ情報を定義することによってそれを行わなければなりません。 `defaultAccessControlPolicies.xml` ファイルを変更するか、または独自の XML ファイルを作成することができます。このような XML ベースの情報を作成することについての詳細は、 *WebSphere Commerce アクセス・コントロール・ガイド* を参照してください。

以下のセクションでは、様々なタイプの関係グループの例を示します。

単一の関係チェーンで構成される関係グループ: Business アクセス制御ポリシーの一部として、ユーザーが組織エンティティー (リソースの BuyingOrganizationalEntity) に属していることを強制しなければならない場合があります。ここでは、長さが 2 である 1 つの関係チェーンで構成される関係グループを作成する必要があります。関係チェーンの長さが ″2″ と言えるのは、それが 2 つの別個の関係で成り立っているからです。最初の関係はユーザーとその親組織エンティティーの間にあります。その関係ではユーザーは「子」になります。 2 番目の関係の場合、アクセス制御ポリシー・マネージャーは、親組織エンティティーがリソースとの間で BuyingOrganizationalEntity 関係を強制しているかどうか調べます。言い換えれば、それがリソースの購買組織エンティティーである場合は、″true″ を戻します。

以下の XML 断片は defaultAccessControlPolicies.xml ファイルからとられており、このタイプの関係グループを定義する方法を示しています。

```
<RelationGroup Name="MemberOf->BuyerOrganizationalEntity"
     OwnerID="RootOrganization">
   <RelationCondition><![CDATA[
      <profile>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="HIERARCHY" value="child"/>
            <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
         </openCondition>
      </profile>
   ]]></RelationCondition>
</RelationGroup>
```

Business 別の例として、ユーザーがリソースの購買組織エンティティーである組織エンティティーのアカウント担当者の役割を持たなければならないことを強制します。ここでも、長さが 2 である 1 つの関係チェーンで構成される関係グループを使用します。チェーンの最初の部分ではユーザーがアカウント担当者の役割を持っているすべての組織エンティティーを見付けます。このような組織エンティティーのセットの場合、アクセス制御ポリシー・マネージャーは、それらの組織エンティティーの少なくとも 1 つがリソースとの間で BuyingOrganizationalEntity 関係を強制しているかどうか調べます。言い換えれば、それがリソースの購買組織エンティティーである場合は、true を戻します。

以下の XML 断片は defaultAccessControlPolicies.xml ファイルからとられており、このタイプの関係グループを定義する方法を示しています。

```
<RelationGroup Name="AccountRep->BuyerOrganizationalEntity"
      OwnerID="RootOrganization">
   <RelationCondition><![CDATA[
      <profile>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="ROLE" value="Account Representative"/>
            <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
```

```
        </openCondition>
      </profile>
   ]]></RelationCondition>
</RelationGroup>
```

複数の関係チェーンで構成される関係グループ: 関係グループに複数の関係チェーンが含まれるように、関係グループを構成することができます。これを行う際に、ユーザーがすべての関係チェーンを満たしていなければならないかどうか (つまり、それが *AND* シナリオであるか)、またはユーザーが少なくとも 1 つの関係チェーンを満たしていればよいか (つまり、それが *OR* シナリオであるか) を指定する必要があります。

Business このタイプの関係を示すために、以下の XML 断片が使用されます。ここでは、ユーザーがリソースの作成者でなければならないこと、さらにリソースで指定された BuyingOrganizationalEntity に属していなければならないことを強制します。最初のチェーンではユーザーがリソースの作成者でなければならず、そのチェーンの長さは 1 です。 2 番目のチェーンではユーザーがリソースで指定された BuyingOrganizationalEntity に属していなければならず、そのチェーンの長さは 2 です。

```
<RelationGroup Name="Creator_And_MemberOf->BuyerOrganizationalEntity"
      OwnerID="RootOrganization">
 <RelationCondition><![CDATA[
  <profile>
   <andListCondition>
    <openCondition name="RELATIONSHIP_CHAIN">
     <parameter name="RELATIONSHIP" value="creator" />
    </openCondition>
    <openCondition name="RELATIONSHIP_CHAIN">
     <parameter name="HIERARCHY" value="child"/>
     <parameter name="RELATIONSHIP" value="BuyingOrganizationalEntity"/>
    </openCondition>
   </andListCondition>
  </profile>
 ]]></RelationCondition>
</RelationGroup>
```

AND シナリオを使用するのではなく、ユーザーが 2 つの関係チェーンのどちらかを満たすように求める場合には、 `<andListCondition>` タグを `<orListCondition>` タグに変更します。

Professional Business WebSphere Commerce Professional Edition (WebSphere Commerce Business Edition と同様) で使用できる関係グループを示すために、ユーザーがリソースの作成者か送信者のどちらかでなければならないことを強制する関係グループについて考えます。これは以下の XML 断片に示されています。

```
<RelationGroup Name="Creator_Or_Submitter"
     OwnerID="RootOrganization">
   <RelationCondition><![CDATA [
   <profile>
```

```
      <orListCondition>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="RELATIONSHIP"value="creator"/>
         </openCondition>
         <openCondition name="RELATIONSHIP_CHAIN">
            <parameter name="RELATIONSHIP"value="submitter"/>
         </openCondition>
      </orListCondition>
   </profile>
   ]]></RelationCondition>
</RelationGroup>
```

アクセス制御のタイプ

アクセス制御のタイプには 2 つあります。コマンド・レベルのアクセス制御とリソース・レベルのアクセス制御で、両方ともポリシーに基づいています。

コマンド・レベル (“役割ベース” としても知られている) のアクセス制御は、幅広いタイプのポリシーを使用します。特定の役割をもつすべてのユーザーが、あるタイプのコマンドを実行できるように指定できます。たとえば、アカウント担当者の役割をもつユーザーが、 AccountRepresentativesCmdResourceGroup リソース・グループで任意のコマンドを実行できるように指定できます。あるいは以下の図で示すように、別のポリシーの例では、すべてのストア管理者が、 StoreAdminCmdResourceGrp によって指定される任意のリソースの ExecuteCommandActionGroup で、指定された任意のアクションを実行できるように指定できます。

注: MBRGRPCOND テーブルの条件列に関する XML 情報は、管理コンソールを使用してアクセス・グループをセットアップする際に生成されます。管理コンソールを使用してアクセス・グループをセットアップすることについての詳細は、WebSphere Commerce のオンライン・ヘルプを参照してください。

ACPOLICY

PolicyName	Member_Id	MbrGrp_Id	AcActGrp_id	AcResGrp_Id	AcRelGrp_Id
StoreAdministrators ExecuteStoreAdmin CmdResourceGroup	-2001	-8	10052	10018	ヌル

MBRGRP

MbrGrp_Id	MbrGrpName
-8	StoreAdministrators

MBRGRPCOND

MbrGrp_Id	条件
-8	<profile> <simpleCondition> <variable name="role"/> <operator name="="/> <value data="Store Administrator"/> </simpleCondition> </profile>

ACACTGRP

AcActGrp_Id	GroupName
10052	ExecuteCommandActionGroup

ACRESGRP

AcResGrp_Id	GrpName
10018	StoreAdminCmdResourceGroup

図 22.

コマンド・レベルのアクセス制御ポリシーは、コントローラー・コマンドのアクション・グループとして常に `ExecuteCommandActionGroup` をもちます。ビューについては、リソース・グループは常に `ViewCommandResourceGroup` です。

すべてのコントローラー・コマンドは、コマンド・レベルのアクセス制御によって保護されなければなりません。さらに、直接呼び出せるビュー、または別のコマンドからリダイレクトに起動できる (ビューへの転送によって起動される場合とは対照的に) ビューはすべて、コマンド・レベルのアクセス制御によって保護されなければなりません。

コマンド・レベルのアクセス制御は、コマンドが影響を及ぼすリソースのことを考えません。単に、ユーザーが特定のコマンドを実行できるかどうかを判別するだけです。ユ

ーザーが特定のコマンドを実行できる場合は、ユーザーが問題のリソースにアクセスできるかどうかを判別するために、後続のリソース・レベルのアクセス制御ポリシーが適用されます。

ストア管理者が管理用タスクの実行を試みる際のことを考えてみてください。アクセス制御検査の第 1 レベルでは、このユーザーが特定のストア管理コマンドの実行を許可されているかどうかを決定します。ユーザーがこれについて実際に (ストア管理者は storeAdminCmds グループでのコマンドの実行を許可されるので) 許可されていると判別されたら、リソース・レベルのアクセス制御ポリシーが呼び出されます。このポリシーには、自身がストア管理者とされている組織が所有するストアについてのみ、ストア管理者に実行が許可されていることが記述してあるかもしれません。

要約すると、コマンド・レベルのアクセス制御では、「リソース」がコマンドそのものであり、「アクション」は単にコマンドを実行する (言い換えれば、コマンド・オブジェクトをインスタンス化する) だけです。アクセス制御検査では、ユーザーにコマンドの実行が許可されているかどうかを決定します。これに対し、リソース・レベルのアクセス制御では、「リソース」はコマンドまたは bean がアクセスする保護可能な任意のリソースであり、「アクション」はコマンドそのものです。

アクセス制御の相互作用

このセクションでは、WebSphere Commerce アクセス制御ポリシーのフレームワークでアクセス制御がどのように動作するかを説明する、相互作用の図を示します。

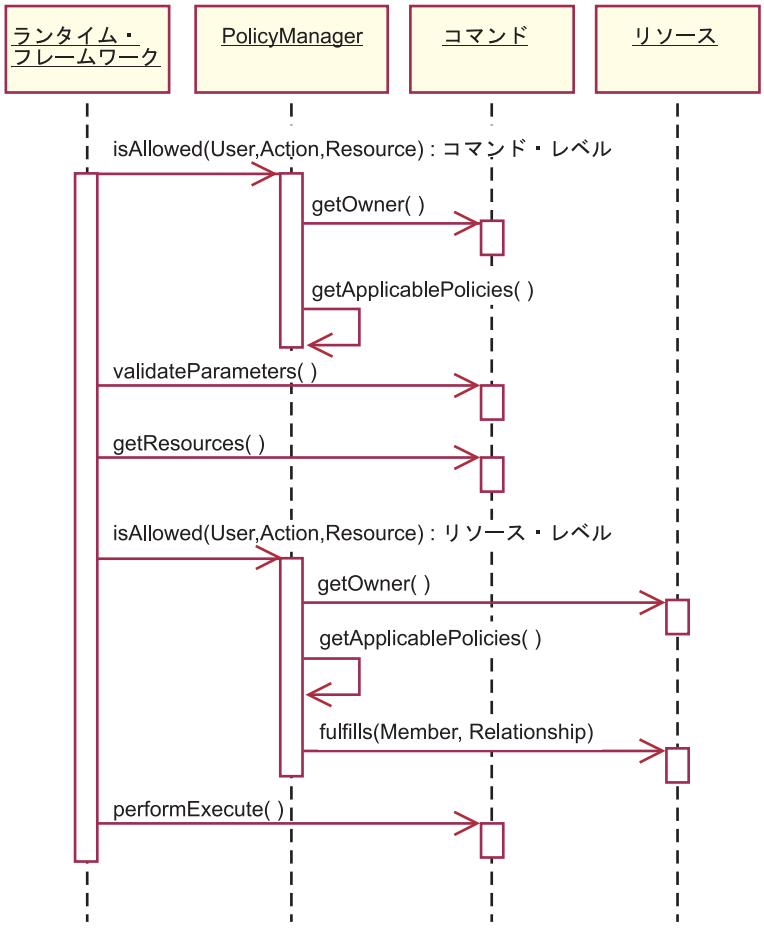


図 23.

上記の図は、アクセス制御ポリシー・マネージャー によって実行されるアクションを示しています。アクセス制御ポリシー・マネージャーは、現行ユーザーが指定されたリソースで指定されたアクションを実行することを許可されているかどうかを判別する、アクセス制御コンポーネントです。ポリシー・マネージャーは、リソースの所有者および祖先の組織が所有するポリシーを検索して、これを決定します。少なくとも 1 つのポリシーがアクセスを認可していれば、許可が与えられます。

以下に、上記の相互作用の図のアクションを説明します。図の上から下という順序で並べてあります。

1. `isAllowed()`
   ランタイム・コンポーネントが、コントローラー・コマンドまたはビューのいずれかについて、ユーザーがコマンド・レベルのアクセスをもつかどうかを判別します。

2. getOwner()
   アクセス制御ポリシー・マネージャーが、コマンド・レベル・リソースの所有者を決定します。デフォルトのインプリメンテーションでは、コマンド・コンテキスト内にあるストア (storeId) の所有者のメンバー ID (memberId) が戻されます。コマンド・コンテキストにストア ID がない場合は、ルート組織 (-2001) が戻されます。
3. getApplicablePolicies()
   アクセス制御ポリシー・マネージャーが、指定されたユーザー、アクション、およびリソースに基づいて、適当なポリシーを検索して処理します。
4. validateParameters()
   初期のパラメーター検査および解決です。
5. getResources()
   リソースとアクションの組みのベクトルであるアクセス・ベクトルを戻します。

   何も戻されない場合は、リソース・レベルのアクセス制御検査は行われていません。保護する必要のあるリソースがある場合は、 (リソースとアクションの組みからなる) アクセス・ベクトルが戻される必要があります。

   各リソース が保護可能なオブジェクト (com.ibm.commerce.security.Protectable インターフェースをインプリメントするオブジェクト) のインスタンスです。多くの場合、このリソースはアクセス Bean です。

   アクセス Bean は com.ibm.commerce.security.Protectable インターフェースをインプリメントしないことがありますが、 109 ページの 『エンタープライズ Bean でのアクセス制御のインプリメント』の情報に従って対応するエンタープライズ Bean が保護される限り、アクセス制御検査は発生します。

   アクション は、リソースで実行される操作を表すストリングです。ほとんどの場合、アクションはコマンドのインターフェース名です。
6. isAllowed()
   ランタイム・コンポーネントが、 getResources() によって指定されるすべてのリソースとアクションの組みについて、ユーザーがリソース・レベルでアクセスできるかどうかを判別します。
7. getOwner()
   リソースが、その所有者の memberId を戻します。これでどのポリシーが適用されるかが決定されます。リソース所有者と祖先の組織によって所有されたポリシーだけが適用されます。
8. getApplicablePolicies()
   アクセス制御ポリシー・マネージャーが、適用できるポリシーを検索して、それを適用します。リソースとアクションの組みについて、ユーザーのリソースへのアクセス権を認可するポリシーが少なくとも 1 つ検出されればアクセスは認可されますが、検出されない場合はアクセスは拒否されます。

9. `fulfills()`
   適用できるポリシーが関係グループを指定している場合、リソースについて、メンバーが指定された関係 (複数も可) を満たすかどうかがリソースで検査されます。
10. `performExecute()`
    コマンドのビジネス・ロジックです。

保護可能なインターフェース

WebSphere Commerce アクセス制御ポリシーによってリソースを保護させるための重要な要素は、リソースが `com.ibm.commerce.security.Protectable` インターフェースをインプリメントしなければならないことです。このインターフェースはエンタープライズ Bean および データ Bean で最もよく使用されますが、保護を必要とするこれらの bean でのみこのインターフェースをインプリメントしなければなりません。

Protectable インターフェースでは、リソースは次の 2 つの鍵メソッドを提供しなければなりません。それは、`getOwner()` と `fulfills(Long member, String relationship)` です。

アクセス制御ポリシーは、組織または組織エンティティーによって所有されます。`getOwner` メソッドは、保護可能なリソースの所有者の memberId を戻します。アクセス制御ポリシー・マネージャーは、リソースの所有者を判別してから、メンバー階層内の所有者の祖先それぞれの memberId も入手します。それから、オリジナルの getOwner 要求で戻された所有者に属する全アクセス制御ポリシーが、その所有者の任意の祖先に属する全アクセス制御ポリシーと同様に適用されます。

指定された所有者に適用されるアクセス制御ポリシーに加え、所有者のメンバーシップ階層内でより高位の祖先に適用されるアクセス制御ポリシーが適用されます。

指定されたメンバーがリソースの点で必要な関係を満たす場合は、 `fulfills` メソッドは true しか戻しません。一般にメンバーは単一のユーザーですが、組織であってもかまいません。アクセス制御ポリシーで関係グループを使用する場合、メンバーは組織になります。

Groupable インターフェース

アクセス制御ポリシーのアプリケーションは、リソースのグループに固有のものです。リソースのグループ化は、クラス名、オーダーの状態または storeId 値などの属性に基づいて行われます。

アクセス制御ポリシーを適用する目的で、クラス名以外の属性によってリソースをグループ化する場合、 `com.ibm.commerce.grouping.Groupable` インターフェースをインプリメントしなければなりません。

以下のコードの断片は Groupable インターフェースを表しています。

```
Groupable interface {
Object getGroupingAttributeValue (String attributeName, GroupContext context)
}
```

たとえば、保留状態 (status = P (保留)) になっているオーダーにしか適用されないポリシーをインプリメントするには、 Order エンティティーのリモート・インターフェースが Groupable インターフェースをサポートし、 attributeName の値が "status" に設定されます。

Groupable インターフェースを使用することはめったにありません。

アクセス制御についての情報の入手先

WebSphere Commerce アクセス制御モデルについての詳細は、 *WebSphere Commerce アクセス・コントロール・ガイド* を参照してください。この資料では、アクセス制御の概要について詳細に説明し、管理コンソールを、ポリシー、アクション・グループ、およびリソース・グループの作成または変更に使用する方法について説明します。

アクセス制御のインプリメント

ここでは、カスタマイズ・コードのアクセス制御をインプリメントする方法について説明します。

保護可能なリソースの識別

一般に、保護が必要なリソースはエンタープライズ Bean およびデータ Bean です。しかし、エンタープライズ Bean とデータ Bean すべてを保護するべきではありません。既存の WebSphere Commerce アプリケーションでは、保護が必要なリソースはすでに保護可能なインターフェースをインプリメントしています。何を保護すべきかという問題は、新規にエンタープライズ Bean およびデータ Bean を作成するときに発生します。どのリソースを保護するかは、ユーザーのアプリケーションに応じて決定します。

コマンドが getResources メソッドでエンタープライズ Bean を戻す場合は、エンタープライズ Bean を保護しなければなりません。これは、アクセス制御ポリシー・マネージャーがエンタープライズ Bean で getOwner メソッドを呼び出すためです。対応するリソース・レベルのアクセス制御ポリシーで関係が指定されている場合は、fulfills メソッドも呼び出されます。

ユーザー自身のエンタープライズ Bean およびデータ Bean すべてについて、保護可能なインターフェースをインプリメントしようとする場合 (したがって、リソースを保護下に置きたい場合) は、アプリケーションに多数のポリシーが必要です。ポリシーの数が増えると、パフォーマンスが悪くなる場合があり、ポリシー管理が難しくなります。

1 次リソースと従属リソースには理論上の違いがあります。 *1 次リソース* は自分だけで存在できます。*従属リソース* は、1 次リソースの存在に関係するときにのみ存在します。たとえば、WebSphere Commerce アプリケーション・コードでは、Order エンティ

ティー Bean は保護可能なリソースですが、 OrderItem エンティティー Bean は違います。この理由は、OrderItem の存在が Order に依存していることにあります。 Order は 1 次リソースで、OrderItem は従属リソースということです。もし、ユーザーが Order にアクセスできるなら、 Order 内のアイテム (OrderItem) にもアクセスできることになります。

同様に、User エンティティー Bean は保護可能なリソースですが、Address エンティティー Bean は違います。この場合は、アドレスの存在がユーザーに依存しているため、ユーザーにアクセスできるものはすべてアドレスへのアクセスもできることになります。

1 次リソースは保護しなくてはなりませんが、従属リソースには保護が不要である場合がよくあります。ユーザーが 1 次リソースへのアクセスを許可される場合、デフォルトでユーザーがその従属リソースへのアクセスも許可されていると理解できます。

エンタープライズ Bean でのアクセス制御のインプリメント

アクセス制御ポリシーによる保護が必要なエンタープライズ Bean を新規作成する場合は、以下を行ってください。

1. 新規のエンタープライズ Bean を作成し、それが `com.ibm.commerce.base.objects.ECEntityBean` から拡張していることを確認します。
2. その bean のリモート・インターフェースが `com.ibm.commerce.security.Protectable` インターフェースを拡張することを確認します。
3. bean が相互作用するリソースが、リソースの Java クラス名以外の属性によってグループ化される場合は、 bean のリモート・インターフェースも `com.ibm.commerce.grouping.Groupable` インターフェースを拡張しなければなりません。
4. エンタープライズ Bean クラスには、以下のメソッドについてのデフォルトのインプリメンテーションが含まれます。
   - `getOwner`
   - `fulfills`
   - `getGroupingAttributeValue`

   必要に応じてメソッドをオーバーライドします。 `getOwner` メソッドは必ずオーバーライドしてください。
   これらのメソッドのデフォルトのインプリメンテーションを以下のコードの断片に示します。

```
************************************************************************
public Long getOwner() throws Exception, java.rmi.RemoteException
{
```

```
      return null;
}
*****************************************************************************
*****************************************************************************
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
{
      return false;
}
*****************************************************************************
*****************************************************************************
public Object getGroupingAttributeValue(String attributeName,
   GroupingContext context) throws Exception, java.rmi.RemoteException
{
      return null;
}
*****************************************************************************
```

以下に、OrderBean bean に基づくこれらのメソッドのサンプル・インプリメンテーションを示します。

```
*****************************************************************************
public Long getOwner() throws Exception, java.rmi.RemoteException
{
   com.ibm.commerce.common.objects.StoreEntityAccessBean storeEntAB = new
   com.ibm.commerce.common.objects.StoreEntityAccessBean();
   storeEntAB.setInitKey_storeEntityId(getStoreEntityId().toString());
   return storeEntAB.getMemberIdInEJBType();
}
*****************************************************************************
*****************************************************************************
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
   {
      if (relationship.equalsIgnoreCase("creator"))
      {
         return member.equals(getMemberId());
      }
      else if (relationship.equalsIgnoreCase (
         com.ibm.commerce.base.helpers.EJBConstants.
         SAME_ORGANIZATIONAL_ENTITY_AS_CREATOR_RELATION)) {
            com.ibm.commerce.user.objects.UserAccessBean creator = new
               com.ibm.commerce.user.objects.UserAccessBean();
            creator.setInitKey_MemberId(getMemberId().toString());
            com.ibm.commerce.user.objects.UserAccessBean ab = new
               com.ibm.commerce.user.objects.UserAccessBean();
            ab.setInitKey_MemberId(member.toString());
            if (ab.getParentMemberId().equals(creator.getParentMemberId()))
               return true;
          }
      return false;
}
*****************************************************************************
```

```
*****************************************************************************
public Object getGroupingAttributeValue(String attributeName,
   GroupingContext context) throws Exception
   {
      if (attributeName.equalsIgnoreCase("Status"))
         return getStatus();
      return null;
   }
*****************************************************************************
```

5. エンタープライズ Bean のアクセス Bean および生成コードを作成 (または再作成) します。

データ Bean でのアクセス制御のインプリメント

データ Bean は、アクセス制御ポリシーによって直接または間接的に保護できます。データ Bean が直接保護される場合は、その特定のデータ Bean に適用されるアクセス制御ポリシーが存在します。データ Bean が間接的に保護される場合は、アクセス制御ポリシーが存在する別のデータ Bean に保護を代行させています。

アクセス制御ポリシーに直接保護される新規のデータ Bean を作成する場合は、データ Bean について以下のことを行わなくてはなりません。

1. `com.ibm.commerce.security.Protectable` インターフェースをインプリメントします。これにより、bean は `getOwner()` および `fulfills(Long member, String relationship)` メソッドをインプリメントする必要があります。これらは、bean のリモート・インターフェースでインプリメントしてください。

   データ Bean が Protectable インターフェースをインプリメントする際は、データ Bean マネージャーが `isAllowed` メソッドを呼び出して、現在のアクセス制御ポリシーに従って、ユーザーに適切なアクセス制御権限があるかどうかを決定します。`isAllowed` メソッドは以下のコードの断片によって記述されます。

   ```
   IsAllowed(Context, "Display", protectable_databean);
   ```

2. bean が相互作用するリソースが、リソースの Java クラス名以外の属性によってグループ化される場合は、bean が `com.ibm.commerce.grouping.Groupable` インターフェースをインプリメントしなければなりません。

3. `com.ibm.commerce.security.Delegator` インターフェースをインプリメントします。このインターフェースは以下のコードの断片によって記述されます。

   ```
   Interface Delegator {
   Protectable getDelegate();
   }
   ```

   注: 直接保護されるためには、 `getDelegate` メソッドがデータ Bean そのものを戻さなければなりません (つまり、データ Bean がアクセス制御の目的で自分に代行させます)。

直接保護されるべきデータ Bean と間接的に保護されるべきデータ Bean との違いは、1 次リソースと従属リソースとの違いに似ています。データ Bean オブジェクトが自分

だけで存在できる場合は、直接保護される必要があります。データ Bean の存在が別のデータ Bean の存在に依存する場合は、他のデータ Bean に保護を代行させるべきです。

直接保護されるデータ Bean の例としては、Order データ Bean があります。間接的に保護されるデータ Bean の例としては、OrderItem データ Bean があります。

アクセス制御ポリシーに間接的に保護される新規のデータ Bean を作成する場合は、データ Bean について以下のことを行わなくてはなりません。

1. com.ibm.commerce.security.Delegator インターフェースをインプリメントします。このインターフェースは以下のコードの断片によって記述されます。

```
Interface Delegator {
Protectable getDelegate();
}
```

   注: getDelegate データ Bean には、 Protectable インターフェースをインプリメントしなければなりません。

データ Bean が Delegator インターフェースをインプリメントしない場合は、アクセス制御ポリシーの保護なしで取り込まれます。

コントローラー・コマンドでのアクセス制御のインプリメント

新規のコントローラー・コマンドを作成すると、新規コマンドのインプリメンテーション・クラスは com.ibm.commerce.commands.ControllerCommandImpl クラスを拡張し、そのインターフェースは com.ibm.commerce.command.ControllerCommand インターフェースを拡張するはずです。インプリメントしなければなりません。

コントローラー・コマンドのコマンド・レベル・ポリシーの場合、そのコマンドのインターフェース名をリソースとして指定します。リソースが保護されるようにするには、保護可能なインターフェースをインプリメントする必要があります。 WebSphere Commerce プログラミング・モデルによれば、そのインプリメントの実現のためには、コマンドのインターフェースを com.ibm.commerce.command.ControllerCommand インターフェースから拡張し、コマンドのインプリメンテーションを com.ibm.commerce.commands.ControllerCommandImpl から拡張します。ControllerCommand インターフェースが com.ibm.commerce.command.AccCommand インターフェースに拡張された後、後者のインターフェースが Protectable を拡張します。コマンド・レベル・レベルのアクセス制御による保護を受けるためには、 AccCommand が、コマンドがインプリメントする必要のある最低限のインターフェースです。

コマンドが保護されるべきリソースにアクセスする場合は、 AccessVector タイプの専用インスタンス変数を作成して、リソースを保持します。それから、このメソッドのデフォルトのインプリメンテーションがヌル値を戻してから getResources メソッドをオーバーライドしてください。リソース検査は起こりません。

新規の getResources メソッドでは、コマンドが動作できるリソースの配列またはリソースとアクションの組みの配列を戻してください。アクションが明示的に指定されない場合、アクションのデフォルトは実行されるコマンドのインターフェース名になります。

さらに、メソッドがリソースをインスタンス化しなければならないのかどうか、またはリソースへの参照を持つ既存のインスタンス変数を使用できるかどうかを、メソッドが決定することをお勧めします。リソース・オブジェクトがすでに存在するかどうかを検査すると、システム・パフォーマンスが向上する可能性があります。必要に応じて、新規のコントローラー・コマンドの performExecute メソッドで、同じ getResources メソッドを使用できます。

以下は、getResources メソッドの例です。

```
private AccessVector resources = null;

public AccessVector getResources() throws ECException {

   if (resources == null) {
      OrderAccessBean orderAB = new OrderAccessBean();
      orderAB.setInitKey_orderId(getOrderId().toString());
      resouces = new AccessVector(orderAB);
      }
   return resources;
}
```

例として OrderItemUpdate コマンドについて考えてみます。このコマンドの getResources メソッドは、Order および User という保護可能なオブジェクトを戻します。アクションは指定されないので、デフォルトは OrderItemUpdate コマンドのインターフェースになります。

getResources メソッドによって複数のリソースが戻されることがあります。このような場合にアクションが実行されるためには、指定されたすべてのリソースについてユーザーにアクセスを許可するポリシーが検出されなければなりません。ユーザーが 3 つのリソースのうち 2 つについてアクセスをもっていてもアクションは進みません (3 つのうち 3 つが必要です)。

コントローラー・コマンドでさらにパラメーター検査またはパラメーターの解決を行う必要がある場合は、 validateParameters() メソッドを使用できます。これはオプションです。

追加のリソース・レベル検査

コントローラー・コマンドの getResources メソッドが呼び出されるときに、保護が必要なすべてのリソースを常に判別することができるとは限りません。

必要であれば、タスク・コマンドでも getResources メソッドをインプリメントして、コマンドを実行できるリソースのリストを戻すことができます。

リソース・レベル検査を呼び出すには、 checkIsAllowed(Object resource, String action) メソッドを使用して、アクセス制御ポリシー・マネージャーを直接呼び出す方法もあります。このメソッドは、
com.ibm.commerce.command.AbstractECTargetableCommand クラスから拡張されたすべてのクラスで使用することができます。たとえば、以下のクラスは
AbstractECTargetableCommand クラスから拡張しています。

- com.ibm.commerce.command.ControllerCommandImpl
- com.ibm.commerce.command.DataBeanCommandImpl

checkIsAllowed メソッドも、 com.ibm.commerce.command.AbstractECCommand クラスを拡張するクラスで使用することができます。たとえば、以下のクラスは
AbstractECCommand クラスから拡張しています。

- com.ibm.commerce.command.TaskCommandImpl

以下は、checkIsAllowed メソッドのシグニチャーを示しています。

```
void checkIsAllowed(Object resource, String action)
   throws ECException
```

このメソッドは、指定のリソースに対して指定のアクションを実行する許可が現在のユーザーに与えられていない場合に ECApplicationException をスローします。アクセスが認可された場合は、このメソッドは単に戻るだけです。

「作成」コマンドのアクセス制御

getResources メソッドはコマンド内で performExecute メソッドの前に呼び出されるので、まだ作成されていないリソースについては異なる方法のアクセス制御が必要です。たとえば、WidgetAddCmd がある場合、getResources メソッドはこれから作成されようとしているリソースを戻すことはできません。この場合、getResources メソッドはリソースの作成者を戻すはずです。たとえば、コマンドはコマンド・ファクトリーによって作成され、オーダーはストア内で作成され、ユーザーは組織内で作成されます。

コマンド・レベルのアクセス制御のデフォルトのインプリメンテーション

コマンド・レベルのアクセス制御では、storeId が指定されていれば、 getOwner() メソッドのデフォルトのインプリメンテーションがストア所有者の memberId を戻します。storeId が指定されていない場合は、ルート組織の memberId が戻されます (memberId = -2001)。

getResources() メソッドのデフォルトのインプリメンテーションは null を戻します。

validateParameters() のデフォルトのインプリメンテーションは何も行いません。

ビューでのアクセス制御ポリシーのインプリメント

ビューについてのリソース・レベルのアクセス制御は、データ Bean マネージャーによって実行されます。データ Bean マネージャーは以下の場合に呼び出されます。

1. JSP テンプレートに <useBean> タグが組み込まれていて、データ Bean が属性リストにない場合。
2. JSP テンプレートに以下の activate メソッドが組み込まれている場合。

```
DataBeanManager.activate(xyzDatabean, request);
```

注: (直接または間接的に) 保護されるデータ Bean は、 Delegator インターフェースをインプリメントしなければなりません。直接保護されるデータ Bean は自分に代行させるため、 Protectable インターフェースのインプリメントする必要があります。間接的に保護されるデータ Bean は、Protectable インプリメントをインターフェースしたデータ Bean に代行させる必要があります。

これは推奨されませんが、以下の場合にアクセス制御検査のバイパスが発生します。

1. JSP テンプレートがデータ Bean を使用せずに、アクセス Bean を直接呼び出す場合。
2. JSP テンプレートがデータ Bean の populate() メソッドを直接呼び出す場合。

コントローラー・コマンドの結果が (ForwardViewCommand を使用して) ビューに転送される場合、コマンド・レベルのアクセス制御はビューでは実行されません。さらに、コントローラー・コマンドが、(ビューで使用される) 取り込まれたデータ Bean を応答プロパティーの属性リストに置いてからビューに転送する場合、 JSP テンプレートは、データ Bean マネージャーを介さずにデータにアクセスできます。これには、<useBean> タグが JSP テンプレートで使用されている必要があります。これによって、ユーザーがコントローラー・コマンドを介してすでにアクセスを認可されているリソース (データ Bean) について、重複するリソース・レベルのアクセス制御検査をすべてバイパスできるので、 JSP テンプレートをより効率的にすることができます。

第 5 章 エラー処理とメッセージ

コマンド・エラー処理

WebSphere Commerce は、カスタマイズされたコードで使いやすい、良く定義されたコマンド・エラー処理フレームワークを使用します。設計では、このフレームワークは、複数文化対応のストアのサポートに対応した方法でエラーを処理します。以降のセクションでは、コマンドが戻す例外のタイプ、例外を処理する方法、メッセージ・テキストを保管して使用する方法、例外をログに記録する方法、および提供されているフレームワークを独自のコマンドで使用する方法について説明します。

例外のタイプ

コマンドは、以下の例外のいずれかを戻すことがあります。

ECApplicationException
この例外は、エラーがユーザーと関係している場合に戻されます。たとえば、ユーザーが無効なパラメーターを入力すると、 ECApplicationException が戻されます。この例外が戻されると、 Web コントローラーはコマンド (再試行可能なコマンドとして指定されたものを含む) を再試行しません。

ECSystemException
この例外は、ランタイム例外か WebSphere Commerce 構成エラーが検出された場合に戻されます。このタイプの例外には、NULL ポインター例外やトランザクション・ロールバック例外などがあります。このタイプの例外が戻されたとき、コマンドが再試行可能コマンドであり、例外がデータベース・デッドロックまたはデータベース・ロールバックによって発生した場合に、 Web コントローラーはコマンドを再試行します。

上記の例外はどちらも、 ECException クラス (com.ibm.commerce.exception パッケージにある) を拡張したクラスです。

これらの例外のいずれかを戻すには、以下の情報を指定する必要があります。

- エラー・ビュー名
  Web コントローラーはこの名前を VIEWREG テーブルで参照します。
- ECMessage オブジェクト
  この値は、プロパティー・ファイルに含まれているメッセージ・テキストに対応します。
- エラー・パラメーター
  これらの名前と値の対は、エラー・メッセージ内の情報を置換するために使用されます。たとえば、メッセージには、例外を戻したメソッドの名前を保持するパラメータ

ーを含むものがあります。このパラメーターは例外が戻されたときに設定され、エラー・メッセージがログに記録されるときには、ログ・ファイルに実際のメソッド名が含められます。

- エラー・データ
  これらは、エラー・データ Bean を介して JSP テンプレートで使用できる、オプションの属性です。

例外処理は、ロギング・システムに強く統合されています。戻された例外は、自動的にログに記録されます。

エラー・メッセージ・プロパティー・ファイル

エラー・メッセージの保守を単純化し、マルチリンガル・ストアをサポートするため、エラー・メッセージのテキストはプロパティー・ファイルに保管されています。WebSphere Commerce メッセージ・テキストは、ecServerMessages_*XX_XX*.properties ファイル (*_XX_XX* はロケール標識 (たとえば、_en_US)) に保管されています。

コマンド・コンテキストは、クライアントが使用している言語を示す ID を戻します。メッセージが必要になると、 Web コントローラーは、この言語 ID に基づいて使用するプロパティー・ファイルを決定します。

ecServerMessages*XX_XX*.properties ファイルでは、 2 つのタイプのメッセージ (ユーザー・メッセージとシステム・メッセージ) が定義されています。ユーザー・メッセージは、顧客のブラウザーに表示されます。システム・メッセージとユーザー・メッセージはどちらも、自動的にメッセージ・ログに取り込まれます。

エラーが戻される際、パラメーターの 1 つとしてメッセージ・オブジェクトが必要です。 ECSystemExceptions の場合、メッセージ・オブジェクトには 2 つのキーが含まれていなければなりません。 1 つはシステム・メッセージ用であり、もう 1 つはユーザー・メッセージ用です。 ECApplicationExceptions の場合、メッセージ・オブジェクトには、ユーザー・メッセージの鍵が含まれています (システム・メッセージは使用されません)。

すべてのシステム・メッセージは事前定義されています。固有のシステム・メッセージを作成することはできません。したがって、カスタマイズ・コードが ECSystemException を送信する場合、定義済みのシステム・メッセージの 1 つのメッセージ鍵を指定する必要があります。カスタマイズされたユーザー・メッセージは作成可能です。新規ユーザー・メッセージは、別個のプロパティー・ファイルに保管する必要があります。

例外処理のフロー

以下の図は、例外が検出されたときの情報のフローを示しています。各ステップの説明は以下のとおりです。

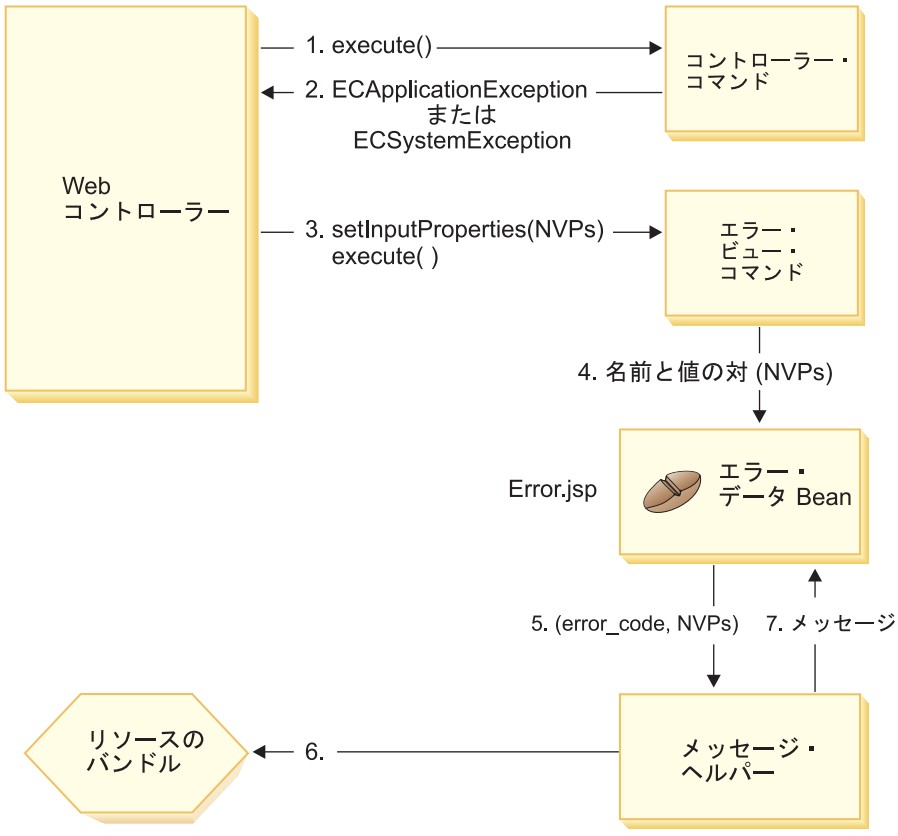


図 24.

1. Web コントローラーはコントローラー・コマンドを呼び出します。
2. コマンドは例外を戻し、その例外は Web コントローラーによって検出されます。この例外は、ECApplicationException または ECSystemException です。例外オブジェクトには以下の情報が含まれています。
   - エラー・ビュー名
   - ECMessage オブジェクト
   - エラー・パラメーター
   - (オプション) エラー・データ
3. Web コントローラーは、VIEWREG テーブルからエラー・ビュー名を判別し、指定したエラー・ビュー・コマンドを呼び出します。コマンドを呼び出す際、Web コントローラーは、 ECException オブジェクトからプロパティーのセットを構成し、ビュー・コマンドの setInputProperties メソッドを用いてそれをビュー・コマンドに設定します。

4. ビュー・コマンドはエラー JSP テンプレート (この場合は Error.jsp) を呼び出し、名前と値の対がその JSP テンプレートに渡されます。
5. ErrorDataBean は、エラー・パラメーターをメッセージ・ヘルパー・オブジェクトに渡します。
6. メッセージ・ヘルパー・オブジェクトは、 (メッセージ・オブジェクトとエラー・パラメーターを使用して) 必要なメッセージを適切なプロパティー・ファイルから取得します。
7. エラー・データ Bean は、メッセージを JSP テンプレートに戻します。

カスタマイズされたコードにおける例外処理

新規コマンドを作成する際は、適切な例外処理を含めることが重要です。例外検出時に必要な情報を指定することにより、 WebSphere Commerce が備えているエラー処理とメッセージング・フレームワークを利用することができます。

独自の例外処理ロジックを作成するには、以下のステップを行います。

1. 使用しているコマンドで、特殊な処理を必要とする例外を取り込む。
2. ECApplicationException または ECSystemException のいずれかを、取り込んだ例外のタイプに基づいて構築する。
3. ECApplicationException が新規メッセージを使用する場合、そのメッセージを新規プロパティー・ファイルで定義する。

例外の検出と構成

最初の 2 つのステップを示すため、以下のコードの断片はコマンド内でシステム例外を検出する例を示しています。

```
try {
// your business logic
}
catch(FinderException e) {
     throw new ECSystemException (ECMessage._ERR_FINDER_EXCEPTION,
          className, methodName, new Object [] {e.toString()}, e);
}
```

前述の _ERR_FINDER_EXCEPTION ECMessage オブジェクトは以下のように定義されています。

```
public static final ECMessage _ERR_FINDER_EXCEPTION =
new ECMessage (ECMessageSeverity.ERROR, ECMessageType.SYSTEM,
     ECMessageKey._ERR_FINDER_EXCEPTION);
```

_ERR_FINDER_EXCEPTION メッセージ・テキストは、以下のように ecServerMessages_*xx_XX*.properties ファイル (*_xx_XX* は *_en_US* などのロケール標識) 内で定義されています。

```
_ERR_FINDER_EXCEPTION  =
     The following Finder Exception occurred during processing: "{0}".
```

システム例外を検出する際は、事前定義されたメッセージのセットを使用できます。これらについて以下の表で説明します。

メッセージ・オブジェクト	説明
_ERR_FINDER_EXCEPTION	エラーが EJB ファインダー・メソッド呼び出しから戻されたときに戻されます。
_ERR_REMOTE_EXCEPTION	エラーが EJB リモート・メソッド呼び出しから戻されたときに戻されます。
_ERR_CREATE_EXCEPTION	エラーが EJB インスタンスの作成時に発生したときに戻されます。
_ERR_NAMING_EXCEPTION	エラーがネーム・サーバーから戻されたときに戻されます。
_ERR_GENERIC	予期しないシステム・エラーが発生したときに戻されます。たとえば、ヌル・ポインター例外などです。

アプリケーション例外を検出したときは、適切な ecServerMessages_xx_xx.properties ファイルで指定されている既存のメッセージを使用するか、新規プロパティー・ファイルに保管される新規メッセージを作成することができます。前述のように、ecServerMessages_xx_XX.properties ファイルは決して 変更しないでください。

以下のコードの断片は、コマンド内でのアプリケーション例外の検出の例を示しています。

```
try {
// your business logic

}
// catch some new type of application exception
catch(//your new exception)
 {
     throw new ECApplicationException (MyMessages._ERR_CUSTOMER_INVALID,
          className, methodName, errorTaskName, someNVPs);
}
```

前述の _ERR_CUSTOMER_INVALID ECMessage オブジェクトは以下のように定義されています。

```
public static final ECMessage _ERR_CUSTOMER_INVALID =
     new ECMessage (ECMessageSeverity.ERROR, ECMessageType.USER,
     MyMessagesKey._ERR_CUSTOMER_INVALID, "ecCustomerMessages");
```

新規ユーザー・メッセージの構築時には、それらに USER のタイプを、以下のように割り当てる必要があります。

```
ECMessageType.USER
```

_ERR_CUSTOMER_INVALID メッセージのテキストは、
ecCustomerMessages.properties ファイルに含まれています。このファイルは、クラスパスに含まれているディレクトリーに存在していなければなりません。このテキストは以下のように定義されています。

```
_ERR_CUSTOMER_INVALID = Invalid ID "{0}"
```

メッセージの作成

コマンドが新規メッセージを使用する ECApplicationException を戻す場合は、この新規メッセージを作成する必要があります。新規メッセージの作成には、以下のステップを行います。

1. メッセージ・キーを含む新規クラスを作成する。
2. ECMessage オブジェクトを含む新規クラスを作成する。
3. リソース・バンドルを作成する。
4. メッセージを単体テストする。

各ステップの詳細について以下に説明します。

メッセージ・キーのクラスの作成

新規ユーザー・メッセージの作成における最初のステップは、新規メッセージ・キーを含むクラスの作成です。メッセージ鍵は、ロギング・サービスによって使用される固有の標識で、リソース・バンドルで対応するメッセージ・テキストを位置決めします。この新規クラスは、独自のパッケージ内に作成し、 WebSphere Commerce プロジェクトとは別個のプロジェクト内に保管する必要があります。

MyNewMessages という例を考慮します。ここでは MyMessageKeys という新規クラスを作成し、それには _ERR_CUSTOMER および _ERR_CUSTOMER_INVALID_ID メッセージ・キーが含まれています。このクラスを com.mycompany.messages パッケージに置きます。この場合、クラス定義は以下のようになります。

```
public class MyMessageKeys
{
    public static String _ERR_CUSTOMER="_ERR_CUSTOMER";
    public static String _ERR_CUSTOMER_INVALID_ID="_ERR_CUSTOMER_INVALID_ID";
}
```

メッセージ・キーのストリング・ラッパーが提供されることにより、コンパイラーはその妥当性を検査することができます。

ECMessage オブジェクトのクラスの作成

メッセージ・キーのクラスを作成した同じパッケージ内では、 ECMessage オブジェクトを含む別のクラスを作成します。 ECMessage クラスは、メッセージ・オブジェクトの構造を定義します。これは、ロケール固有のテキスト・メッセージを検索して永続化するのに使用します。

メッセージ・オブジェクトには、重大度、タイプ、鍵、リソース・バンドル、および関連リソース・バンドルという属性があります。このクラスには、いくつかのコンストラクター・メソッドがあります。完全な詳細については、WebSphere Commerce のオンライン・ヘルプの『Reference』のセクションを参照してください。

MyNewMessages の例に続いて、MyMessages という新規クラスを、com.mycompany.messages パッケージ内に、以下のように作成します。

```
import com.ibm.commerce.ras.*;

public class MyMessages
    {
        static String myResourceBundle = "ecCustomerMessages";

        public static ECMessage _ERR_CUSTOMER = new ECMessage
             (ECMessageSeverity.ERROR,ECMessageType.USER,
                   MyMessageKeys._ERR_CUSTOMER, myResourceBundle);
        public static ECMessage _ERR_CUSTOMER_INVALID_ID = new ECMessage
             (ECMessageSeverity.ERROR, ECMessageType.USER,
                  MyMessageKeys._ERR_CUSTOMER_INVALID_ID,
                  myResourceBundle);

    }
```

上記の部分コードでは、ECMessage オブジェクトの作成には、インポート・ステートメントが必要です。オブジェクト MyMessage._ERR_CUSTOMER は、重大度 ERROR のユーザー・メッセージです。 MyMessageKeys._ERR_CUSTOMER は、 WebSphere Commerce ロギング・サービスによって使用され、 *ecCustomerMessages* プロパティー・ファイルに含まれているメッセージ・テキストを検索します。

メッセージ・リソース・バンドルの作成

メッセージ・キーが対応するメッセージ・テキストとともに保管される新規リソース・バンドルを作成する必要があります。このリソース・バンドルは、 Java オブジェクトまたはプロパティー・ファイルとしてインプリメントすることができます。プロパティー・ファイルの方が変換や保守が容易なので、こちらを使用することをお勧めします。プロパティー・ファイルは WebSphere Commerce メッセージに使用されます。

MyNewMessages の例を続行するには、 ecCustomerMessages.properties という名前でテキスト・ファイルを作成します。メッセージが単一のストア・サーブレットによって使用される場合は、このファイルを次のディレクトリーに入れてください。

drive:¥WebSphere¥AppServer¥installedApps¥WC_EnterpriseApp_*instanceName*.ear¥wcstores.war¥WEB-INF¥classes

メッセージが単一のツール・サーブレットによって使用される場合は、このファイルを次のディレクトリーに入れてください。

drive:¥WebSphere¥AppServer¥installedApps¥WC_EnterpriseApp_*instanceName*.ear¥wctools.war¥WEB-INF¥classes

エンタープライズ・アプリケーション内のあらゆるサーブレットによってメッセージがグローバルに使用される場合は、このファイルを次のディレクトリーに入れてください。

drive:¥WebSphere¥AppServer¥installedApps¥WC_EnterpriseApp_*instanceName*.ear¥properties

プロパティー・ファイルにはメッセージ・キーと対応するメッセージ・テキストの対が含まれるので、 ecCustomerMessages.properties ファイルには以下の行が含まれます。

```
_ERR_CUSTOMER_MESSAGE = The customer message "{0}".
_ERR_CUSTOMER_INVALID_ID = Invalid ID "{0}".
```

メッセージの単体テスト

メッセージ・キーを含むクラスと ECMessage オブジェクトを含むクラス、およびリソース・バンドルが作成されたら、新規メッセージの単体テストを行う必要があります。

これまでのセクションで説明した新規メッセージをテストするには、以下のようにします。

1. テスト目的に使用する新規クラスを作成します。この場合は、MyTestingClass を作成します。
2. 以下の import ステートメントをクラスに追加します。

   ```
   import com.ibm.commerce.ras.*;
   ```

3. main() メソッドをクラスに追加します。
4. 以下のコードの断片を調べ、独自の要件に従って変更し (たとえば、ディレクトリー・パスがシステム上の有効な構成ファイルを指すようにする)、main() メソッドに挿入します。

   ```
   // the fileName String variable should point
   // to a valid WebSphere Commerce configuration file:
     String fileName = "E:¥¥WebSphere¥¥CommerceServer¥¥instances
         ¥¥demo¥¥xml¥¥demo.xml";

      LogConfiguration config = LogConfiguration.getUniqueInstance ();
      config.initialize (fileName, "testClone");

      ECMessageLog.out (MyMessage._ERR_CUSTOMER,
           "MyTestingClass", "main", "Hello");
   ```

   コードの最初の 3 行は、WebSphere Commerce ロギング・サービスを初期化します。コードの最後の行は、メッセージ MyMessage._ERR_CUSTOMER をプリントアウトするように ECMessageLog に指示します。 MyTestingClass と main は、ログ・トレース・フォーマットの一部です。 Hello ストリングは、ecCustomerMessages.properties ファイルで定義されているメッセージの {0} プレースホルダーに入れられます。
5. クラスを実行します。ログ・ファイルでは、以下のようなメッセージ・トレースが出力されます。

```
==============
TimeStamp:      2000-11-29 16:41:42.5
Thread ID:     <main>
Class:          MyTestingClass
Method:         main
Severity:       1
Message Text:   The customer message "Hello".
```

同様のテストを実行すれば、2 番目のメッセージを見ることができます。

実行フローのトレース

WebSphere Commerce には、WebSphere Commerce Server で実行されているコンポーネントの実行フローのトレースに使う ECTrace クラスが組み込まれています。ECTrace クラスは、 com.ibm.commerce.ras パッケージの一部です。

新規のビジネス・ロジックを作成する際は、コードにトレースを挿入することにより、デバッグ目的でメソッドをトレースすることができます。トレースからの情報は、トレース・ログに取り込まれます。トレースには、エントリー・ポイントとエグジット・ポイントを指定することができます。加えて、これらの 2 つのポイントの間で特定のデータをトレースするように指定できます。

トレースを使用するには、トレースを実行するコンポーネントでトレースが使用可能になっている必要があります。特定のコンポーネントについてトレースを使用可能にするには、管理コンソールか構成マネージャーのどちらかを使用できます。

カスタマイズ・コードをトレースする際には、EXTERN コンポーネントを使用*しなければなりません*。構成マネージャーでは、これは *External* と呼ばれています。

コード内でトレースのエントリー・ポイントを設定するには、以下の構文を使用します。

```
ECTrace.entry (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName, myMethodName);
```

myClassName は、トレースされるメソッドを含むクラスのストリング表現です。このストリングは、トレース・ファイルの構文解析に使用できるため、完全修飾クラス名を含める必要があります。トレースするメソッドが静的メソッドである場合、 myClassName の宣言例は以下のようになります。

```
String myClassName = "com.mycompany.agrouping.MyTracedClass";
```

トレースするメソッドが静的メソッドでない場合、 myClassName の宣言例は以下のようになります。

```
String myClassName = this.getClass().getName();
```

メソッド内でデータをトレースするトレース・ポイントを設定するには、以下の構文を使用します。

```
ECTrace.trace (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName,
     myMethodName, myText);
```

myText は、トレース・ログに出力されるテキストです。

コード内でトレースのエグジット・ポイントを設定するには、以下の構文を使用します。

```
ECTrace.exit (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName,
     myMethodName);
```

トレースするメソッドから戻されたオブジェクトをトレースする必要がある場合は、エグジット・ポイントを以下のように設定します。

```
ECTrace.exit (ECTraceIdentifiers.COMPONENT_EXTERN, myClassName,
     myMethodName, returnedObject);
```

returnedObject は、メソッドから戻される Java オブジェクトを表します。

新規コントローラー・コマンド MyNewControllerCmd の performExecute メソッドをトレースしなければならない例を考えましょう。以下のコードの断片は、performExecute メソッド内で ECTrace メソッドを使用する方法を示しています。

```
public void performExecute() throws ECException {
     ECTrace.entry(ECTraceIdentifiers.COMPONENT_EXTERN,
               this.getClass().getName(), "performExecute");

         super.performExecute();


//////////////////////////////////////////
//   Some of your business logic          //
//////////////////////////////////////////


     ECTrace.trace(ECTraceIdentifiers.COMPONENT_EXTERN,
          this.getClass().getName(), "performExecute",
          "My code is great!");

//////////////////////////////////////////
//   Some more business logic             //
//////////////////////////////////////////


     ECTrace.exit(ECTraceIdentifiers.COMPONENT_EXTERN,
               this.getClass().getName(), "performExecute");

}
```

前述の performExecute メソッドが呼び出されると、トレース・ログ・ファイルに以下の情報が取り込まれます。

```
==============
TimeStamp:     2000-12-05 17:32:00.257
Thread ID:    <P=502832:O=0:CT>
Component:     EXTERN
Class:         com.mycompany.agrouping.MyNewControllerCmd
Method:        performExecute
Trace:          ENTRY POINT
==============
TimeStamp:     2000-12-05 17:32:00.257
Thread ID:    <P=502832:O=0:CT>
Component:     EXTERN
Class:         com.mycompany.agrouping.MyNewControllerCmd
Method:        performExecute
Trace:         My code is great!
==============
TimeStamp:     2000-12-05 17:32:00.258
Thread ID:    <P=502832:O=0:CT>
Component:     EXTERN
Class:         com.mycompany.agrouping.MyNewControllerCmd
Method:        performExecute
Trace:         EXIT POINT
```

トレースは、主要機能に対してのみ使用することをお勧めします。トレースは、ストア開発者が使用することを目的としているため、複数の言語に対応していません。メッセージはこれと対照的で、複数の言語に対応しています。これは、システム・メッセージは管理目的で使用されるものであり、ユーザー・メッセージは顧客に表示されるものであるためです。

JSP テンプレートのエラー処理

JSP テンプレートのエラー処理は、以下のようなさまざまな方法で実行することができます。

- ページ内からのエラー処理
  より複雑なエラー処理およびリカバリーが必要な JSP の場合には、データ Bean からのエラーを直接処理するファイルを作成することができます。この JSP ファイルは、データ Bean が活動化されたかどうかに応じて、データ Bean から戻された例外を検出するか、または各データ Bean 内でエラー・コード・セットを検査することができます。次いで、受け取ったエラーに基づき、適切なリカバリー・アクションを実行することができます。 JSP ファイルは、以降のエラー処理の有効範囲を任意で組み合わせて使用できます。
- ページ・レベルのエラー JSP
  JSP ファイルは、JSP エラー・タグを使用して、自身の内部で発生した例外から独自のデフォルト・エラー JSP テンプレートを指定することもできます。これにより、JSP プログラムは、独自のエラー処理を指定することができます。 JSP エラー・タグを指定しない JSP ファイルでは、エラーはアプリケーション・レベルの JSP エラ

ー・テンプレートになります。ページ・レベルのエラー JSP では、 JSP ヘルパー・クラス (com.ibm.server.JSPHelper) を呼び出して、現在のトランザクションをロールバックする必要があります。

- アプリケーション・レベルのエラー JSP
  WebSphere の舌で実行されているアプリケーションは、サーブレットや JSP ファイル内からの例外が発生したときにおけるデフォルト・エラー JSP テンプレートを指定できます。アプリケーション・レベルのエラー JSP テンプレートは、モール・レベルまたはストア・レベル (単一ストア・モデルの場合) のエラー・ハンドラーとして使用することができます。アプリケーション・レベルのエラー JSP テンプレートでは、サーブレット・ヘルパー・クラスを呼び出して、現在のトランザクションをロールバックする必要があります。これは、Web コントローラーが、トランザクションをロールバックする実行パスに存在しないためです。可能な場合は常に、先行する 2 つのタイプの JSP エラー処理を行う必要があります。アプリケーション・レベルのエラー処理ストラテジーは、必要な場合にのみ使用してください。

第 6 章 コマンドのインプリメンテーション

このセクションでは、新規のコントローラー・コマンド、タスク・コマンド、およびデータ Bean コマンドを作成する方法について説明します。また、既存のコントローラー・コマンド、タスク・コマンド、およびデータ Bean コマンドを拡張する方法についても説明します。

注: Business この章ではビジネス・ポリシー・コマンドについては説明しません。ビジネス・ポリシー・コマンドの情報については、 153 ページの 『第 7 章 取引の合意事項とビジネス・ポリシー (Business Edition)』を参照してください。

新規コマンド - 概要

WebSphere Commerce プログラミング・モデルは、コントローラー・コマンド、タスク・コマンド、ビュー・コマンド、およびデータ Bean コマンドの 4 種類のコマンドを定義しています。 e-commerce アプリケーション用の新しいビジネス・ロジックを作成するとき、コントローラー・コマンド、タスク・コマンド、およびデータ Bean コマンドを新しく作成しなければならない場合があります。ビュー・コマンドについては、新しく作成する必要はありません。ビュー・コマンドに関する詳細は、このセクションの後の部分で説明されます。

新規コマンドには、対応するインターフェースをインプリメントする必要があります (そのようなインターフェースは、既存のインターフェースから拡張します)。コマンドの記述を簡単にするために、WebSphere Commerce には、それぞれの種類のコマンドの抽象インプリメンテーション・クラスが含まれています。新規コマンドは、これらのクラスから拡張する必要があります。

以下の表は概要を示しています。新規コマンドの拡張元となるインプリメンテーション・クラスはどれか、また、どんなインターフェースをインプリメントするかを示しています。

コマンドのタイプ	コマンド名の例	拡張元	インプリメントするインターフェースの例
コントローラー・コマンド	MyControllerCmdImpl	com.ibm.commerce.command.ControllerCommandImpl	MyControllerCmd
タスク・コマンド	MyTaskCmdImpl	com.ibm.commerce.command.TaskCommandImpl	MyTaskCmd

コマンドのタイプ	コマンド名の例	拡張元	インプリメントするインターフェースの例
データ Bean コマンド	MyDataBeanCmdImpl	com.ibm.commerce.command.DataBeanCommandImpl	MyDataBean

注: インプリメンテーション・クラスの名前にはスペースが含まれていますが、これは単に説明のためです。

以下の図は、新規コントローラー・コマンドのインターフェースとインプリメンテーション・クラスの間の関係、および既存の抽象インプリメンテーション・クラスと既存のインターフェースを示しています。抽象クラスとインターフェースは、どちらも com.ibm.commerce.command パッケージに含まれています。

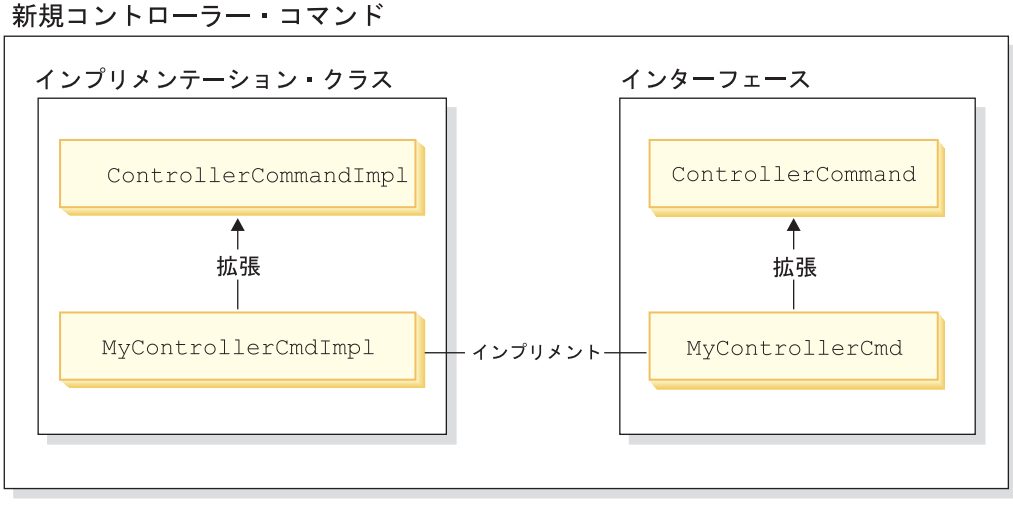


図 25.

以下の図は、新規タスク・コマンドのインターフェースとインプリメンテーション・クラスの間の関係、および既存の抽象インプリメンテーション・クラスと既存のインターフェースを示しています。抽象クラスとインターフェースは、どちらも com.ibm.commerce.command パッケージに含まれています。

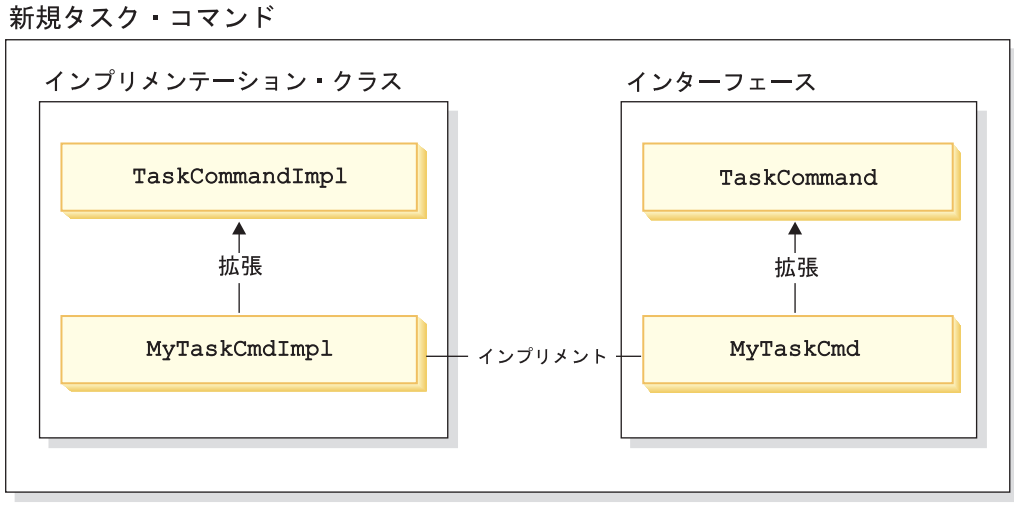


図 26.

以下の図は、新規データ Bean コマンドのインターフェースとインプリメンテーション・クラスの間の関係、および既存の抽象インプリメンテーション・クラスと既存のインターフェースを示しています。抽象クラスとインターフェースは、どちらも com.ibm.commerce.command パッケージに含まれています。

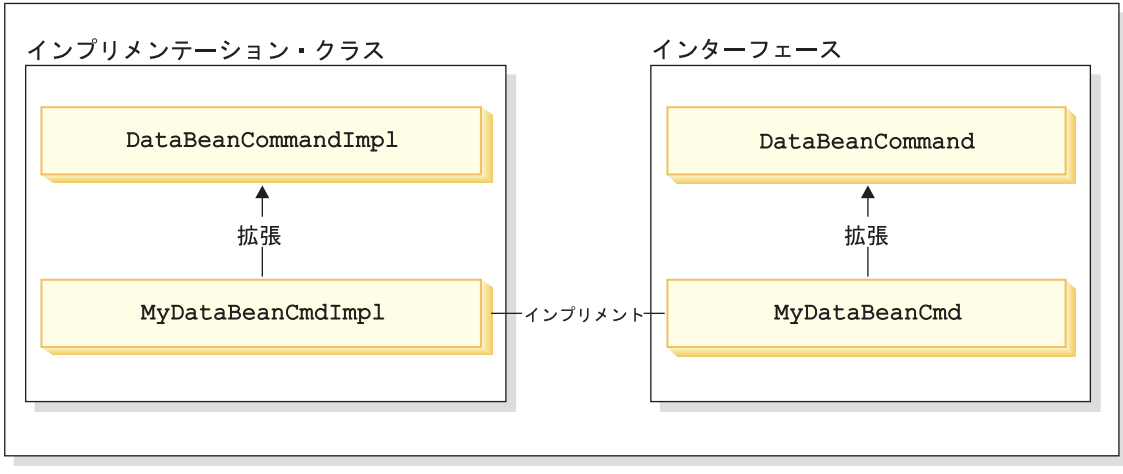


図 27.

ビュー・コマンドの主な 2 つの機能は、応答のフォーマット、およびクライアントへの応答の送信です。さまざまなプロトコルを使用してクライアントに応答を送信する、いくつかの汎用ビュー・コマンドが提供されています。フォーマット機能は、通常、 JSP

テンプレートを呼び出すビュー・コマンドによって実行されます。たとえば、RedirectViewCommand ビュー・コマンドは、応答を得るためにクライアントを URL に転送します (続いて、応答は指定された JSP テンプレートによってフォーマットされます)。 ForwardViewCommand ビュー・コマンドは、フォーマットを行うために要求を JSP テンプレートに転送し、ページがクライアントに表示されます。

このビュー・コマンド・モデルを使えば、新規 JSP テンプレートを作成することによって、新しい ビュー (クライアントへの応答) を作成できます。ただし、 JSP テンプレートは既存のビュー・コマンドのいずれかから呼び出さなければなりません。

カスタマイズ・コードのパッケージ

カスタマイズ・コードを作成するとき、特定のコード編成構造に従う必要があります。一般に、カスタマイズされたコードは、WebSphere Commerce に組み込まれているものとは別のパッケージおよびプロジェクトで保守されます。

新規コマンドを作成するとき、ビジネス要件からみて適切なパッケージ名の中に入れる必要があります。たとえば、特定のストアに適用されるコマンドの場合、そのストアに固有のパッケージの中に含めるようにします。複数のストアに適用されるコマンドの場合には、それに対応するパッケージに含めます。たとえば、以下のようなパッケージがあるとします。

- `com.bigbusiness.storeA.commands`
- `com.bigbusiness.storeB.commands`
- `com.bigbusiness.commands`

上記のようなパッケージ構造にすれば、さまざまなビジネス・ロジックをストア・レベルで区別することができます。さらに、これらのパッケージは、 WebSphere Commerce プロジェクトとは別のプロジェクトに保管する必要があります。たとえば、上記のパッケージを `BigBusinessCustomCode` というプロジェクトの中に入れることができます。

新しいデータ Bean を作成するときには、コマンド・ロジックとは異なるパッケージの中に bean を保管する必要があります。ただし、そのパッケージは、コマンド・パッケージと同じプロジェクトの中に入れることができます。上記の例に従えば、`com.bigbusiness.databeans` パッケージを `BigBusinessCustomCode` プロジェクトの中に含めることができます。

新しいエンティティー Bean を作成するときには、固有のプロジェクトの中に bean を保管する必要があります。したがって、`com.bigbusiness.objects` パッケージが含まれている `BigBusinessCustomEntityBeans` プロジェクトを持つことができます。

このようなパッケージの方針は、コードのデプロイメントのために必要です。

コマンド・コンテキスト

コマンド・コンテキストは Web コントローラーへのハンドルです。コマンド・コンテキストを使用して、コマンドは Web コントローラーから情報を得ることができます。入手できる情報には、たとえばユーザー ID、ユーザー・オブジェクト、言語 ID、ストア ID などがあります。

コマンドを記述するとき、そのコマンドのスーパークラスの getCommandContext() メソッドを呼び出すことによって、コマンド・コンテキストにアクセスします。 Web コントローラーによってコマンドが呼び出されるとき、コマンド・コンテキストはコントローラー・コマンドに設定されます。コントローラー・コマンドは、処理中に呼び出される任意のタスク・コマンドまたはコントローラー・コマンドにコマンド・コンテキストを伝搬します。コマンドは、以下の key 情報をコマンド・コンテキストから受け取る可能性があります。

getUserId() および getUser()

現在のユーザー ID またはユーザー・オブジェクトを取得します。現在のセッションのユーザー ID はセッション・コンテキストに保管されます。セッション・コンテキストは、 WebSphere Commerce cookie または WebSphere Application Server 永続セッション・オブジェクトのいずれかを使用して永続的に保持することができます。コマンド・コンテキストは、セッション管理の複雑さをコマンドの表面から隠します。

getStoreId()、getStore()、および getStore(storeId)

現在の要求に関連したストアを取得します。 Web コントローラーは URL のストア ID を戻します。 URL にストア ID が指定されていない場合には、前の要求で保管されたセッション・オブジェクトからストア ID が取得されます。 WebSphere Commerce ランタイム環境は、頻繁にアクセスされる一連のオブジェクトを保守します。たとえば、ストア・オブジェクトのセットを保守しています。コマンドは、コマンド・コンテキストからストア・オブジェクトをいつでも取得して、 Web コントローラー内のオブジェクト・キャッシュを利用することができます。コマンド・コンテキストから getStore() メソッドを呼び出して現在のストアを取得したり、 getStore(storeId) メソッドを呼び出して特定のストア・オブジェクトを取得することができます。

getLanguageId()

現在の要求で使用される言語 ID を戻します。 Web コントローラーはグローバリゼーション・フレームワークをインプリメントします。このフレームワークの背後にある概念は、ユーザーにとって好ましく、かつ、ストアによってサポートされる言語を判別することです。 URL に言語 ID が含まれる場合、この言語がストアによってサポートされるかどうかを Web コントローラーが判別します。サポートされる場合は、それが getLanguageId() メソッドによって戻される言語 ID となります。 URL に言語 ID が組み込まれていない場合は、 Web コントローラーが決定ツリーをたどって、現在のセッション・オブ

ジェクトまたはユーザーの登録済み設定に (ストアによってサポートされる) 言語 ID があるかどうかを判別します。判別できない場合は、最終的にストアのデフォルト言語 ID を戻します。

getCurrency()
現在の要求で使用される通貨を戻します。通貨はグローバリゼーション・フレームワークの一部なので、このメソッドの背後にあるロジックは getLanguageId() メソッドに似ています。

getCurrentTradingAgreements() および getTradingAgreement(tradingAgreementId)
現行セッションで使用される取引の合意事項のセットを戻します。このセットは、ユーザーにかかわるすべての取引の合意事項である場合もありますし、ContractSetInSession コマンドに定義されたサブセットである場合もあります。コマンドは、コマンド・コンテキストから取引の合意事項オブジェクトをいつでも取得して、 Web コントローラー内のオブジェクト・キャッシュを利用することができます。コマンド・コンテキストから getCurrentTradingAgreements() メソッドを呼び出して現在の取引の合意事項を取得したり、getTradingAgreement(tradingAgreementId) メソッドを呼び出して特定の取引の合意事項オブジェクトを取得することができます。

コマンド・コンテキストは、読み取り専用オブジェクトとして使う必要があります。setter メソッドを呼び出さないでください。 setter メソッドは、 WebSphere Commerce ランタイム環境が使用するためだけの目的で予約されています。将来のリリースでは使用できなくなるかもしれません。

コマンド・コンテキスト API (アプリケーション・プログラミング・インターフェース) の完全な詳細については、 WebSphere Commerce のオンライン・ヘルプの『Reference』のセクションを参照してください。

新規コントローラー・コマンド

すでに説明したように、新しいコントローラー・コマンドは抽象コントローラー・コマンド・クラス (com.ibm.commerce.command.ControllerCommandImpl) から拡張する必要があります。新しいコントローラー・コマンドを書くとき、以下のようなメソッドを抽象クラスからオーバーライドしてください。

- isGeneric()
- isRetriable()
- setRequestProperties(com.ibm.commerce.datatype.TypedProperty reqParms)
- validateParameters()
- getResources()
- performExecute()

前述のメソッドのそれぞれについて、以下のセクションで詳しく説明します。

isGeneric メソッド

標準的な WebSphere Commerce インプリメンテーションでは、これには、一般、ゲスト、および登録ユーザーがあります。登録ユーザーのグループの中には、顧客とアドミニストレーターが含まれます。

すべての一般ユーザーには、システム全体で使われる 1 つの共通のユーザー ID が割り当てられます。共通ユーザー ID は、システム・リソースの使用を最小にする方法で、サイトに対する一般的なブラウズをサポートします。一般的なブラウズにこの共通ユーザー ID を割り当てると、より効率的です。なぜなら、Web コントローラーは、一般ユーザーによって呼び出されるコマンド用にユーザー・オブジェクトを検索する必要がないからです。

isGeneric メソッドは、コマンドを一般ユーザーが呼び出せるかどうかを指定するブール値を戻します。コントローラー・コマンドのスーパークラスの isGeneric メソッドは、値を `false` に設定します (これは呼び出し側が登録ユーザーまたはゲスト・ユーザーのどちらかでなければならないことを意味します)。新規コントローラー・コマンドを一般ユーザーから呼び出し可能にするには、このメソッドが `true` を戻すようにオーバーライドしてください。

ユーザーに関連付けられたリソースを作成したり取り出したりしない新規コマンドの場合、このメソッドが `true` を戻すようにオーバーライドする必要があります。一般ユーザーによる呼び出しが可能なコマンドの例には、 ProductDisplay コマンドがあります。すべてのユーザーが商品を見られるようにすることは、道理にかなっています。一方、ゲスト・ユーザーまたは登録ユーザーでなければならない (つまり、 isGeneric が `false` を戻す) コマンドの例としては、 OrderItemAdd コマンドがあります。

isGeneric が `true` の値を戻すとき、 Web コントローラーは現在のセッションの新規ユーザー・オブジェクトを作成しません。したがって、一般ユーザーによる呼び出しが可能なコマンドの場合、 Web コントローラーはユーザー・オブジェクトを検索する必要がないため、より高速に実行されます。

一般ユーザーがコマンドを呼び出し可能にするためにこのメソッドを使う場合の構文は、以下のとおりです。

```
public boolean isGeneric()
{
   return true;
}
```

isRetriable メソッド

isRetriable メソッドは、コマンドをトランザクションのロールバック例外で再試行できるかどうかを指定するブール値を戻します。新規コントローラー・コマンドのスーパークラスの isRetriable メソッドは、 `false` の値を戻します。トランザクションのロールバック例外の際に再試行できるコマンドの場合、このメソッドをオーバーライドして `true` の値を戻すようにする必要があります。

トランザクションの例外の場合に再試行すべきでないコマンドの例には、 OrderProcess コマンドがあります。このコマンドは、サード・パーティーの決済与信プロセスを呼び出します。与信を取り消すことはできないため、このコマンドは再試行できません。再試行の可能なコマンドの例には、 ProductDisplay コマンドがあります。

トランザクションのロールバック例外においてこのコマンドを再試行可能にする構文は、以下のとおりです。

```
public boolean isRetriable()
{
   return true;
}
```

setRequestProperties メソッド

setRequestProperties メソッドは、すべての入力プロパティーをコントローラー・コマンドに渡すために Web コントローラーによって呼び出されます。このメソッドの内部では、コントローラー・コマンドが入力プロパティーを構文解析して、それぞれの個別のプロパティーを明示的に設定しなければなりません。コントローラー・コマンドによるこのようなプロパティーの明示的設定は、タイプ・セーフなプロパティーの概念を促進します。

このメソッドを使用するための構文は、以下のとおりです。

```
public void setRequestProperties(
   com.ibm.commerce.datatype.TypedProperty reqParms)
{

     // parse the input properties and explicitly set each parameter

}
```

validateParameters メソッド

validateParameters メソッドは、初期パラメーター検査や、必要なパラメーターの解決すべてを行うために使用されます。たとえば `orderId=*` を解決するために使用できます。このメソッドは、 getResources および performExecute の両方のメソッドの前に呼び出されます。この順序についての詳細情報は、 104 ページの 『アクセス制御の相互作用』を参照してください。

getResources メソッド

このメソッドは、リソース・レベルのアクセス制御をインプリメントするために使用されます。このメソッドは、コマンドが動作しようとするリソースとアクションの組みのベクトルを戻します。何も戻されない場合は、リソース・レベルのアクセス制御は行われていません。アクセス制御についての詳細は、 91 ページの 『第 4 章 アクセス制御』を参照してください。

performExecute メソッド

performExecute メソッドには、コマンドのビジネス・ロジックを含めます。新しいビジネス・ロジックを実行する前に、コマンドのスーパークラスの performExecute メソッドを呼び出す必要があります。また、最後にビュー名を戻す必要があります。

以下の例は、新規コントローラー・コマンドの中の performExecute メソッドの構文を示しています。この例の場合は応答に Redirect View コマンドを使用していますが、Direct View コマンドや Forward View コマンドを使用することもできます。

```
public void performExecute() throws ECException
{
         super.performExecute();

//////////////////////////////////////////////
//  your business logic                     //
//////////////////////////////////////////////

// Create a new TypedProperty for response properties.
TypedProperty rspProp = new TypedProperty();

// set response properties
rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");
//////////////////////////////////////////////////
//  The following line is optional.  The VIEWREG   //
//  table can specify the redirect URL.           //
//////////////////////////////////////////////////

rspProp.put(ECConstants.EC_REDIRECTURL, MyURL);

////////////////////////////////////////////////////////////////////////////
//   If you are using a forward view, you can set the                     //
//   response properties as follows:                                      //
//   TypedProperty rspProp = new TypedProperty();                         //
//   rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");                  //
//   rspProp.put(ECConstants.EC_DOCPATHNAME, "MyJSP.jsp");                //
//                                                                        //
//   Again, it is optional to explicitly set the name of the JSP template.//
//   The VIEWREG table can specify the JSP template.                      //
////////////////////////////////////////////////////////////////////////////

setResponseProperties(rspProp);
}
```

performExecute メソッドの中でリダイレクト URL を指定した場合、 VIEWREG テーブル内にエントリーがあれば、コードで指定した値の方が VIEWREG テーブルの値よりも優先されます。コードの中で JSP テンプレートを指定した場合も、同じ優先順位が適用されます。

長時間実行されるコントローラー・コマンド

コントローラー・コマンドの実行が長時間かかる場合、コマンドを 2 つのコマンドに分割することができます。 URL 要求の結果として実行される最初のコマンドは、単に 2

番目のコマンドをスケジューラーに追加し、それをバックグラウンド・ジョブとして実行するだけです。この仕組みが以下の図に示されています。

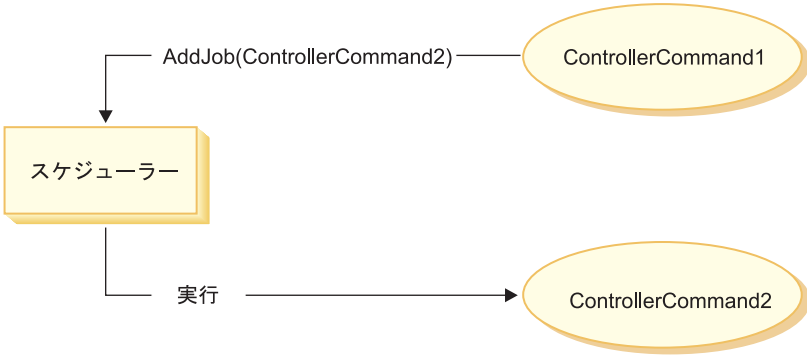


図 28.

上記の図に示されているフローは、以下のようになります。

1. ControllerCommand1 が、URL 要求の結果として実行される。
2. ControllerCommand1 はスケジューラーにジョブ (つまり ControllerCommand2) を追加する。ジョブをスケジューラーに追加するとただちに、ControllerCommand1 はビューを戻す。
3. スケジューラーは、 ControllerCommand2 をバックグラウンド・ジョブとして実行する。

このシナリオでは、クライアントは通常、 ControllerCommand2 から結果を受け取ります。 ControllerCommand2 はジョブの状態をデータベースに書き込む必要があります。

ビュー・コマンドに対する入力プロパティーのフォーマット設定

コントローラー・コマンドが完了すると、実行されるべきビューの名前が戻されます。場合によっては、このビューに複数の入力プロパティーを渡す必要があります。その入力パラメーターには、以下にリストする 3 つのソースがあります。

- CMDREG テーブルの PROPERTIES 列に保管されるデフォルトのプロパティー
- VIEWREG テーブルの PROPERTIES 列からのデフォルト・プロパティー
- URL からの入力プロパティー

これらのプロパティーがマージされて JSP テンプレートの属性に設定される方法について、詳しくは 45 ページの 『JSP 属性の設定 - 概要』を参照してください。このセクションでは、ビュー・コマンドに対する入力プロパティーをフォーマット設定する方法について説明します。

リダイレクト・ビュー・コマンドについては、2 つのトピックが検査されます。

- URL のリダイレクトをサポートするためのクエリー・ストリングのフラット化
- リダイレクト URL の長さの制限の処理

転送ビュー・コマンドについては、入力パラメーターを列挙して、それらを HttpServletRequestObject 内で属性として設定するトピックについて検査されます。

入力パラメーターの HttpRedirectView 用クエリー・ストリングへのフラット化

リダイレクト・ビュー・コマンドに渡される入力パラメーターは、すべて URL リダイレクト用のクエリー・ストリングにフラット化されます。たとえば、リダイレクト・ビュー・コマンドに対する入力に以下のプロパティーが含まれるとします。

```
URL = "MyView?p1=v1&p2=v2";
ip1 = "iv1";   // input to orginal controller command
ip2 = "iv2" ;  // input to original controller command
op1 = "ov1";
op2 = "ov2";
```

上記の入力パラメーターに基づいて、最終的な URL は以下のようになります。

```
MyView?p1=v1&p2=v2&ip1=iv1&ip2=iv2&op1=ov1&op2=ov2
```

コマンドが SSL を使用する場合は、パラメーターは暗号化され、最終的な URL は以下のようになることに注意してください。

```
MyView?krypto=encrypted_value_of"p1=v1&p2=v2&ip1=iv1&ip2=iv2&op1=ov1&op2=ov2"
```

長さが制限されたリダイレクト URL の処理

デフォルトでは、コントローラー・コマンドへの入力パラメーターはすべてリダイレクト・ビュー・コマンドに伝搬されます。リダイレクト URL の文字数に制限があると、問題が起きることがあります。長さが制限される例としては、クライアントが Internet Explorer ブラウザーを使用している場合があります。このブラウザーでは、URL は 2083 バイトを超えられません。 URL がこの制限を超えると、URL は切り捨てられます。このように、入力パラメーターの数が多いと問題が起こる可能性があります。また、一般に暗号化ストリングは非暗号化ストリングより 2 倍から 3 倍長いため、暗号化を使用している場合も問題が起こる可能性があります。

長さが制限されたリダイレクト URL を処理するには、2 つの方法があります。

1. コントローラー・コマンドで getViewInputProperties メソッドをオーバーライドして、リダイレクト・ビュー・コマンドに渡す必要のあるパラメーターのセットだけを戻すようにする。
2. 指定された特殊文字を URL パラメーターに使用して、入力パラメーター・ストリングから除去できるパラメーターを示す。

コントローラー・コマンドに対して以下の一連の入力パラメーターがあるとして、上記の各方法を説明します。

```
URL="MyView";
// All of the following are inputs to the original controller command.
ip1="ipv1";
ip2="ipv2";
ip3="ipv3";
iq1="iqv1";
iq2="iqv2";
ir1="ipr1";
ir2="ipr2";
is="isv";
```

getViewInputProperties メソッドをオーバーライドする場合は、新しいメソッドを書き込んで、以下のパラメーターだけをビュー・コマンドに渡すようにできます。

```
ir2="ipr2";
is="isv";
```

2 番目の方法を使用する場合は、特殊なパラメーターを使用してビュー・コマンドを呼び出して、特定の入力パラメーターを除去するように指示することができます。たとえば、URL パラメーターとして以下を指定することで、同じ結果が得られます。

```
URL="MyView?ip*=&iq*=&ir1="
```

この URL パラメーターは、 WebSphere Commerce ランタイム・フレームワークに以下を指示します。

- `ip*=` の指定は、名前が `ip` で始まるパラメーターをすべて除去するという指示です。
- `iq*=` の指定は、名前が `iq` で始まるパラメーターをすべて除去するという指示です。
- `ir1=` の指定は、`ir1` パラメーターを除去するという指示です。

HttpForwardView についての HttpServletRequest オブジェクトでの属性の設定

デフォルトの HttpForwardViewCommandImpl は、コマンドに渡されるパラメーターをすべて列挙し、 HttpServletRequest オブジェクト内で属性として設定します。

たとえば、転送ビュー・コマンドに渡された requestProperties オブジェクトに以下のパラメーターが含まれるとします。

```
p1="pv1";
p2="pv2";
p3=pv3;   // pv3 is an object
```

それから、request.setAttribute() メソッドを使用して、以下の属性が JSP テンプレートに渡されます。

```
request.setAttribute("p1", "pv1");
request.setAttribute("p2", "pv2");
request.setAttribute("p1", pv1);
request.setAttribute("RequestProperties", requestProperties);
request.setAttribute("CommandContext", commandContext);
```

ここで、requestProperties はコマンドに渡される TypedProperty オブジェクト、commandContext はコマンドに渡されるコマンド・コンテキスト・オブジェクト、 p1、p2、p3 は requestProperties オブジェクトで定義されるパラメーターです。

コントローラー・コマンド用のデータベース・コミットおよびロールバック

コントローラー・コマンドを実行し始めてから終わるまで、頻繁にデータが作成されたり更新されたりします。多くの場合、トランザクションの末尾で新しい情報でデータベースを更新しなければなりません。トランザクションは Web コントローラーによって管理されます。

Web コントローラーは、コントローラー・コマンドを呼び出す前に、トランザクションの先頭にマークを付けます。コントローラー・コマンドの実行が完了すると、コントローラー・コマンドは Web コントローラーにビュー名を戻します。 Web コントローラーは、トランザクションの末尾にマークを付けます。実際にトランザクションが終了する時点 (ビューを呼び出す前または後) は、使用するビューのタイプに応じて変わります。

ビュー・コマンドには 3 つのタイプがあります。

- Forward View コマンド
- Redirect View コマンド
- Direct View コマンド

Web コントローラーは、VIEWREG テーブル中でビュー名を検索して、ビューに使用するビュー・コマンドを判別します。

VIEWREG テーブル中のエントリーで ForwardViewCommand を使用するよう指定されている場合は、 Web コントローラーは、コントローラー・コマンドの結果を、対応する ForwardViewCommand インプリメンテーション・クラス (これも VIEWREG 中に指定されている) に転送します。ビュー・コマンドは現行トランザクションのコンテキスト中で実行されます。この場合、ビュー・コマンドが完了するまでデータベースのコミットやロールバックは行われません。

VIEWREG テーブル中のエントリーで RedirectViewCommand を使用するよう指定されている場合は、 Web コントローラーは、コントローラー・コマンドの結果を、対応する RedirectViewCommand インプリメンテーション・クラスに転送します。転送後、ビ

ュー・コマンドは現行トランザクションの有効範囲外で操作され、リダイレクト・ビュー・コマンドが呼び出される前にデータベースのコミットやロールバックが行われます。

VIEWREG テーブル中のエントリーで DirectViewCommand を使用するよう指定されている場合は、 Web コントローラーは、コントローラー・コマンドの結果を、対応する DirectViewCommand インプリメンテーション・クラスに転送します。ビュー・コマンドは現行トランザクションのコンテキスト中で実行されます。この場合、ビュー・コマンドが完了するまでデータベースのコミットやロールバックは行われません。(ForwardViewCommand と DirectViewCommand は似ていることに注意してください。ForwardViewCommand は、結果を JSP テンプレートに転送します。逆に、DirectViewCommand は結果を入力ストリームとして受け取り、出力ストリームとして渡します。その際、データをバイトとして処理する getRawDocument メソッドか、データをテキストとして処理する getTextDocument メソッドのどちらかを使用します。)

ビュー・コマンドがコントローラー・コマンドと同じトランザクション有効範囲内で実行されると、ビュー・コマンドでエラーが生じて、トランザクション全体のロールバックが行われます。この結果が望ましいかどうかは、ご使用のビジネス・ロジックによります。

コントローラー・コマンド使用時のトランザクション有効範囲の例

以下の例には、使用するビュー・コマンドのタイプが違うと、コントローラー・コマンドのトランザクション有効範囲に関してどのような違いが生じるか図示されています。

事例 1: コントローラー・コマンドのトランザクションの有効範囲内でビューを実行する

YourControllerCmdA という新しいコントローラー・コマンドを作成したとします。この場合、このコマンドの performExecute メソッドには以下のものが組み込まれています。

```
.
.
// Create a new TypedProperty object for output.
TypedProperty rspProp = new TypedProperty();

//////////////////////
// Business logic   //
//////////////////////

// Return the view
rspProp.put(ECConstants.EC_VIEWTASKNAME, "YourView");
SetResponseProperties(rspProp);
```

上記のコードの断片で、コントローラー・コマンドはビューとして “YourView” を戻します。 YourView は VIEWREG テーブルに登録されています。以下は、YourView を登録する insert ステートメントの例です。

```
insert into VIEWREG (ViewName, DeviceFmt_id, storeEnt_id, interfacename,
classname, properties)

values ('YourView',  -1, XX,'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl','docname=YourView.jsp');
```

XX はストア ID です。ビューが com.ibm.commerce.command.HttpForwardViewCommandImpl インプリメンテーション・クラスを使用するので、 Webコントローラーは汎用ビュー転送コマンドを使用します。

上記のコマンド登録に基づいて、 Web コントローラーはコントローラー・コマンドのトランザクションの有効範囲内で YourView.jsp ファイルを立ち上げます。YourView.jsp でエラーが生じると、トランザクションは失敗し、データベースのロールバックが行われます。その結果、コントローラー・コマンド全体が失敗します。

事例 2: コントローラー・コマンドのトランザクションの有効範囲外でビューを実行する

ビューでエラーが生じた場合でも、データベースに情報をコミットすることを選んだとします。コントローラー・コマンドのトランザクションの有効範囲外でビューを実行するには、ビューをリダイレクトとして実行しなければなりません。

コントローラー・コマンドの performExecute メソッドは、ビューをリダイレクトとして実行するために、以下の方法でビューを戻します。

```
.
.
// Create a new TypedProperty object for output.
TypedProperty rspProp = new TypedProperty();

//////////////////////
// Business logic   //
//////////////////////

// Return the view
rspProp.put(ECConstants.EC_VIEWTASKNAME, EC_GENERIC_REDIRECTVIEW);
rspProp.put(EC_Constants.EC_REDIRECTURL, "YourView2");
```

以下の例の SQL ステートメントは、リダイレクト・ストラテジーをサポートしています。

```
insert into VIEWREG (ViewName, DeviceFmt_id, storeEnt_id, interfacename,
classname, properties)

values ('YourView2',  -1, XX,'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl','docname=YourView2.jsp');
```

XX はストア ID です。

このコマンドは EC_GENERIC_REDIRECTVIEW 値を応答プロパティー・パラメーターとして渡すので、 Web コントローラーは汎用ビュー・リダイレクト・コマンドを使用します。この汎用ビュー・リダイレクト・コマンドは、以下の情報を指定され、VIEWREG テーブルに登録されます。

- ViewName = RedirectView
- DeviceFmt_Id = -1
- InterfaceName = com.ibm.commerce.command.RedirectViewCommand
- ClassName = com.ibm.commerce.command.HttpRedirectViewCommandImpl

Web コントローラーは汎用ビュー・リダイレクト・コマンドを呼び出しますが、このコマンドには入力プロパティーとしてリダイレクト URL があります。応答はリダイレクト URL にリダイレクトされます。リダイレクトされた後に、YourView2 が呼び出されます。これは汎用ビュー転送としてインプリメントされます。

新規タスク・コマンド

新しいタスク・コマンドは抽象タスク・コマンド・クラス (com.ibm.commerce.command.TaskCommandImpl) を拡張するとともに、com.ibm.commerce.TaskCommand インターフェースを拡張するインターフェースをインプリメントします。 131ページの図に示されるように、新規タスク・コマンドは以下のように定義される必要があります。

```
public class MyTaskCmdImpl extends com.ibm.commerce.command.TaskCommandImpl
     implements MyTaskCmd {

}
```

タスク・コマンドのすべての入出力プロパティーは、コマンド・インターフェース (たとえば MyTaskCmd) で定義する必要があります。呼び出し側は、タスク・コマンドのインプリメンテーション・クラスではなく、タスク・コマンド・インターフェースを呼び出すようにプログラミングします。こうすれば、タスク・コマンドの複数のインプリメンテーション (各ストアごとに 1 つ) が可能になり、呼び出し側はどのインプリメンテーション・クラスを呼び出すべきかを考慮する必要がありません。

インターフェースで定義されているすべてのメソッドを、インプリメンテーション・クラスにインプリメントしなければなりません。コマンド・コンテキストは呼び出し側 (コントローラー・コマンド) によって設定さるため、タスク・コマンドはコマンド・コンテキストを設定する必要がありません。しかし、タスク・コマンドは、コマンド・コンテキストを使用して、 Web コントローラーから情報を得ることができます。

タスク・コマンド・インターフェースで定義されているメソッドをインプリメントすることに加えて、 com.ibm.commerce.command.TaskCommandImpl クラスの performExecute メソッドをオーバーライドする必要があります。

performExecute メソッドには、タスク・コマンドが実行する特定の作業単位のビジネス・ロジックを含めます。ビジネス・ロジックを実行する前に、タスク・コマンドのスーパークラスの performExecute メソッドを呼び出す必要があります。以下のコードの断片は、タスク・コマンドの performExecute メソッドの例を示しています。

```
public void performExecute() throws ECException
{
         super.performExecute();

     // Include your business logic here.

     // Set output properties so the controller command
     // can retrieve the result from this task command.
}
```

ランタイム・フレームワークは、コントローラー・コマンドの getResources メソッドを呼び出して、このコマンドがアクセスする保護可能なリソースを決定します。コントローラー・コマンドの有効範囲中に実行されたタスク・コマンドが、そのコントローラー・コマンドの getResources メソッドによって戻されなかったリソースにアクセスしようとするケースも考えられます。この場合は、保護可能なリソースについてアクセス制御が確実に行われるように、タスク・コマンド自体が getResources メソッドをインプリメントできます。

デフォルトでは getResources はタスク・コマンドについて null を戻し、リソース・レベルのアクセス制御検査は実行されないことに注意してください。したがって、タスク・コマンドが保護可能なリソースにアクセスする場合は、 getResources をオーバーライドしなければなりません。

既存のコマンドのカスタマイズ

このセクションでは、既存のコントローラー・コマンド、タスク・コマンド、およびデータ Bean コマンドをカスタマイズするさまざまな方法について説明します。

既存のコントローラー・コマンドのカスタマイズ

コントローラー・コマンドは、ビジネス・プロセスのビジネス・ロジックをカプセル化します。ビジネス・プロセス内の個々の作業単位を、タスク・コマンドによって実行することがあります。したがって、コントローラー・コマンドはいくつかの方法でカスタマイズすることができ、タスク・コマンドのカスタマイズと関連する場合もあります。

コントローラー・コマンドをカスタマイズする際に、以下のことを行えます。

- 既存のコントローラー・コマンドに追加の処理とロジックを加えることができます。これは、既存のビジネス・ロジックの前、後、または前と後の両方に追加できます。
- 1 つまたは複数のコマンドを置き換えます。こうすれば、ビジネス・プロセス内の特定のステップの実行方法を変えることができます。

- コントローラー・コマンドによって呼び出されるビューを置き換えることができます。

上記の変更を加える方法の詳細について以下に説明します。

コントローラー・コマンドへの新規ビジネス・ロジックの追加

ExistingControllerCmd という既存の WebSphere Commerce コントローラー・コマンドがあるとします。 WebSphere Commerce の命名規則に従って、このコントローラー・コマンドには ExistingControllerCmd という名前のインターフェース・クラスと、ExistingControllerCmdImpl という名前のインプリメンテーション・クラスがあります。ビジネス要件が生じて、この既存のコマンドに新規のビジネス・ロジックを追加しなければならないとします。ロジックには、既存のコマンド・ロジックの前に実行しなければならない部分と、既存のコマンド・ロジックの後に実行しなければならない部分があります。

新しいビジネス・ロジックを追加するための最初のステップとして、オリジナルのインプリメンテーション・クラスを拡張する新しいインプリメンテーション・クラスを作成します。この例では、ExistingControllerCmdImpl クラスを拡張する新しい ModifiedControllerCmdImpl クラスを作成します。この新しいインプリメンテーション・クラスは、オリジナルのインターフェース (ExistingControllerCmd) をインプリメントする必要があります。

この新しいインプリメンテーション・クラスでは、新規の performExecute メソッドを作成して、既存のコマンドの performExecute をオーバーライドしなければなりません。新規の performExecute メソッドに新しいビジネス・ロジックを挿入する方法は 2 つあります。コントローラー・コマンドにコードを直接組み込む方法か、または新しいタスク・コマンドを作成して新しいビジネス・ロジックを実行する方法です。新しいタスク・コマンドを作成する場合は、コントローラー・コマンド中から新しいタスク・コマンド・オブジェクトをインスタンス化しなければなりません。

以下のコードの断片は、コントローラー・コマンド中にロジックを直接組み込むことにより、既存のコントローラー・コマンドの先頭および末尾に新しいビジネス・ロジックを追加する方法を示しています。

```
public class ModifiedControllerCmdImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

            /* Insert new business logic that must be
               executed before the original command.
            */

            // Execute the original command logic.
            super.performExecute();
```

```
            /* Insert new business logic that must be
               executed after the original command.
            */
          }
    }
```

以下のコードの断片は、コントローラー・コマンド中から新しいタスク・コマンドをインスタンス化することにより、既存のコントローラー・コマンドの先頭に新しいビジネス・ロジックを追加する方法を示しています。さらに、新しいタスク・コマンド・インターフェースとインプリメンテーション・クラスを作成し、コマンド・レジストリー中にタスク・コマンドを登録します。

```
// Import the package with the CommandFactory
import com.ibm.commerce.command.*;

public class ModifiedControllerCmdImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {

      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {
            MyNewTaskCmd cmd = null;
            cmd = (MyNewTaskCmd) CommandFactory.createCommand(
               "com.mycompany.mycommands.MyNewTaskCommand",
               getStoreId());

            /*
            Set task command's input parameters, call its
            execute method and retrieve output
            parameters, as required.
            */

            super.performExecute();
         }
   }
```

コントローラー・コマンドに新しいビジネス・ロジックを組み込むか、タスク・コマンドを作成してロジックを実行するかにかかわらず、 WebSphere Commerce コマンド・レジストリー中の CMDREG テーブルを更新して、新しいコントローラー・コマンドのインプリメンテーション・クラスを既存のコントローラー・コマンドのインターフェースに関連付けなければなりません。以下の SQL ステートメントは、更新の例を示しています。

```
update CMDREG
set CLASSNAME='ModifiedControllerCmdImpl'
where INTERFACENAME='ExistingControllerCmd'
```

コントローラー・コマンドによって呼び出されるタスク・コマンドの置換

コントローラー・コマンドは、個別のタスクを実行する複数のタスク・コマンドを呼び出すことがよくあります。これらのタスクは一まとまりとして、コントローラー・コマ

ンドによって表されるビジネス・プロセスを構成します。新しいビジネス・ロジックをコントローラー・コマンドの先頭や末尾に追加するのではなく、このプロセス中の特定のプロセスが実行される方法を変更する必要が生じることがあります。この場合は、オーバーライドしたいタスク・コマンドのインスタンス化を、ご希望の方法でタスクを実行する新しいタスク・コマンドのインスタンス化に置き換えなければなりません。

WebSphere Commerce プログラミング・モデルの設計では、新しいコントローラー・コマンドのインプリメンテーション・クラスを作成して、タスク・コマンドを置き換える必要はありません。コントローラー・コマンドは、コマンド・ファクトリーの createCommand メソッドを呼び出して、タスク・コマンドをインスタンス化します。コマンド・ファクトリーは、タスク・コマンドのインターフェース名を使用して、コマンド・レジストリーに基づいて正しいインプリメンテーション・クラスを判別します。したがって、インスタンス化されるタスク・コマンドを置き換えるには、新しいタスク・コマンドのインプリメンテーション・クラスを作成してから、コマンド・レジストリーを更新して、オリジナルのタスク・コマンドのインターフェースと新しいタスク・コマンドのインプリメンテーション・クラスを関連付けなければなりません。詳しくは、150 ページの『既存のタスク・コマンドのカスタマイズ』を参照してください。

コントローラー・コマンドによって呼び出されるビューの置換

コントローラー・コマンドによって呼び出されるビューを置換するには、コントローラー・コマンドの新しいインプリメンテーション・クラスを作成します。たとえば、 ExistingControllerCmdImpl を拡張し、 ExistingControllerCmd インターフェースをインプリメントする、新規の ModifiedControllerCmdImpl を作成します。

ModifiedControllerCmdImpl クラス中で performExecute メソッドをオーバーライドしてください。新規の performExecute メソッドで、 super.performExecute を呼び出して、すべてのコマンド処理が起こるようにします。コマンド・ロジックが実行されてから、応答プロパティーを使用して呼び出されたビューをオーバーライドできます。以下のコードの断片は、ビューがリダイレクトとして実行されるときにビューをオーバーライドする方法を示しています。

```
// Import the packages containing TypedProperty, and ECConstants.
import com.ibm.commerce.datatype.*;
import com.ibm.commerce.server.*;

public class ModifiedControllerCmdImplImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

            // Execute the original command logic.
            super.performExecute();

            // Create a new TypedProperty for response properties.
            TypedProperty rspProp = new TypedProperty();
```

```
            // set response properties
            rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");
            ////////////////////////////////////////////////////
           //  The following line is optional.  The VIEWREG   //
           //  table can specify the redirect URL.            //
           ////////////////////////////////////////////////////

            rspProp.put(ECConstants.EC_REDIRECTURL, MyURL);

            setResponseProperties(rspProp);

          }
   }
```

以下のコードの断片は、ビューが転送ビューとして実行されるときにビューをオーバーライドする方法を示しています。

```
// Import the packages containing TypedProperty, and ECConstants.
import com.ibm.commerce.datatype.*;
import com.ibm.commerce.server.*;

public class ModifiedControllerCmdImplImpl extends ExistingControllerCmdImpl
   implements ExistingControllerCmd
   {
      public void performExecute ()
         throws com.ibm.commerce.exception.ECException
         {

            // Execute the original command logic.
            super.performExecute();

            // Create a new TypedProperty for response properties.
            TypedProperty rspProp = new TypedProperty();

           // set response properties
            rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyView");

            ////////////////////////////////////////////////
           //  It is optional to explicitly set the name   //
           //  of the JSP template. The VIEWREG table can  //
           //  specify the JSP template.                   //
           ////////////////////////////////////////////////

            rspProp.put(ECConstants.EC_DOCPATHNAME, "MyJSP.jsp");

            setResponseProperties(rspProp);

          }
    }
```

既存のコントローラー・コマンドに使用されているビューを判別するには、 WebSphere Commerce のオンライン・ヘルプの『Reference』のセクションを参照してください。

既存のタスク・コマンドのカスタマイズ

既存の WebSphere Commerce タスク・コマンドに変更を加える標準的な方法は 2 つあります。これらの変更方法では、それぞれ以下のことを行えます。

- 既存のタスク・コマンドに追加の処理とロジックを加えることができます。これは、既存のビジネス・ロジックの前、後、または前と後の両方に追加できます。
- 既存のビジネス・ロジックを独自のビジネス・ロジックに完全に置き換えます。

上記のいずれかの変更を加えるには、新しいタスク・コマンド・インプリメンテーション・クラスを実際に作成します。詳細について以下に説明します。

タスク・コマンドへの新規ビジネス・ロジックの追加

ExistingTaskCmd という既存の WebSphere Commerce タスク・コマンドがあるとします。 WebSphere Commerce の命名規則に従って、タスク・コマンドには ExistingTaskCmd という名前のインターフェース・クラスと、 ExistingTaskCmdImpl という名前のインプリメンテーション・クラスがあります。ビジネス要件が生じて、この既存のコマンドに新規のビジネス・ロジックを追加しなければならないとします。ロジックには、既存のコマンド・ロジックの前に実行しなければならない部分と、既存のコマンド・ロジックの後に実行しなければならない部分があります。

新しいビジネス・ロジックを追加するための最初のステップとして、オリジナルのインプリメンテーション・クラスを拡張する新しいインプリメンテーション・クラスを作成します。この例では、ExistingTaskCmdImpl クラスを拡張する新しい ModifiedTaskCmdImpl クラスを作成します。この新しいインプリメンテーション・クラスは、オリジナルのインターフェース (ExistingTaskCmd) をインプリメントしている必要があります。

この新しいコマンドで、既存の performExecute メソッドをオーバーライドし、super.performExecute メソッドを呼び出す前と後に新しいロジックを組み込みます。

以下の疑似コードは、既存のタスク・コマンドに新しいビジネス・ロジックを追加する方法を示しています。

```
public class ModifiedTaskCmdImpl extends ExistingTaskCmdImpl
     implements ExistingTaskCmd {


         /* Insert new business logic that must be
            executed before the original command.
         */

         // Execute the original command logic.
         super.performExecute();

         /* Insert new business logic that must be
            executed after the original command.
         */
}
```

さらに、CMDREG テーブルを更新して、新しいインプリメンテーション・クラスを既存のインターフェースに関連付けなければなりません。以下の SQL ステートメントは、更新の例を示しています。

```
update CMDREG
set CLASSNAME='ModifiedTaskCmdImpl'
where INTERFACENAME='ExistingTaskCmd'
```

既存のタスク・コマンドのビジネス・ロジックの置換

既存のタスク・コマンドのビジネス・ロジックを置換するには、タスク・コマンドの新しいインプリメンテーション・クラスを作成しなければなりません。この新しいインプリメンテーション・クラスは、既存のタスク・コマンドから拡張しなければなりませんが、既存のインターフェースをインプリメントしてはなりません。さらに、新規のインプリメンテーション・クラスでは、スーパークラスの performExecute メソッドを呼び出さないでください。

置き換えようとしているコマンドそのものからの拡張は直感性に反するように見えるかもしれませんが、このアプローチをとる理由は、今後のバージョンの WebSphere Commerce のサポートとの関連性があります。このアプローチをとれば、今後のバージョンの WebSphere Commerce でコマンド・インターフェースに変更が加えられても、各自のコードを保護することができます。

例として、 OrderNotifyCmdImpl タスク・コマンドのビジネス・ロジックを置き換えようとしていると仮定します。この場合、CustomizedOrderNotifyCmdImpl という新規のタスク・コマンドを作成します。このコマンドは OrderNotifyCmdImpl を拡張します。新規の CustomizedOrderNotifyCmdImpl では、新規のビジネス・ロジックを作成することはできますが、スーパークラスから performExecute メソッドを呼び出すことはできません。将来のバージョンの WebSphere Commerce で、インターフェース内に newMethod という新規のメソッドが採用された場合、それに対応するバージョンの OrderNotifyCmdImpl コマンドには、 newMethod メソッドのデフォルト・インプリメンテーションが組み入れられます。するとその新規のコマンドは OrderNotifyCmdImpl から拡張されるので、コンパイラーは OrderNotifyCmdImpl コマンド内でその新規メソッドのデフォルト・インプリメンテーションを検索し、各自の新規コマンドはインターフェースの変更の影響を受けません。

新しいインプリメンテーション・クラスに既存のクラスと外から見て同じ動作をさせるには、 WebSphere Commerce のオンライン・ヘルプの『Reference』のセクションを参照してください。

データ Bean のカスタマイズ

データ Bean は、通常、アクセス Bean から拡張します。 (VisualAge for Java を使用して生成できる) アクセス Bean を使えば、エンティティー Bean 情報に簡単にアクセスできます。いずれかのエンティティー Bean を変更した場合 (たとえばフィールド、ビジネス・メソッド、または finder の新規追加など)、更新内容は、アクセス Bean が

再生成されるとただちにがアクセス Bean に反映されます。データ Bean はアクセス Bean からの拡張であるため、新しい属性を自動的に継承します。このような関係があるので、エンティティー Bean の新しい属性をデータ Bean で使用できるようにするためにコーディングする必要はありません。

エンティティー Bean から派生した属性ではない新しい属性をデータ Bean に追加する必要がある場合、 Java の継承を使用して、既存のデータ Bean を拡張することができます。たとえば、OrderDataBean に新規フィールドを追加したい場合、MyOrderDataBean を以下のように定義します。

```
public class MyOrderDataBean extends OrderDataBean
{
     public String myNewField () {
     //  implement the new field here
     }

}
```

新しいデータ Bean にも、BeanInfo クラスを含める必要があります。このクラスを宣言するには、たとえば以下のようにします。

```
public class MyOrderDataBeanInfo extends java.beans.SimpleBeanInfo
{

}
```

VisualAge for Java には、この BeanInfo クラスを生成するツールが提供されています。

第 7 章 取引の合意事項とビジネス・ポリシー (Business Edition)

この章は、WebSphere Commerce Business Edition にのみ適用されます。

概要

B2B (企業間取引) コマースのかぎとなるエレメントの 1 つに、関係の管理があります。取引の合意事項は、バイヤーとセラー組織間のビジネス関係の管理に使用されます。 WebSphere Commerce Business Edition で使用される取引の合意事項モデルは、契約および RFQ (見積依頼) などのさまざまなタイプの取引の合意事項をサポートします。

取引の合意事項の主なエレメントは、一連の使用条件です。各使用条件で、取引中に使用される特定のビジネスの規則を定義します。 WebSphere Commerce Business Edition を使用すると、一連の使用条件について、RFQ オンライン処理を使用して折衝をしたり、もしくはオフラインで折衝してから、WebSphere Commerce アクセラレーターのビジネス関係管理インターフェースを使用して取り込むことができます。

使用条件はいくつかあります。

- 価格表およびリターン・ポリシーのような、事前定義されたビジネス・ポリシーの 1 つを選択する使用条件。また、ユーザーが作成したビジネス・ポリシーも選択できます。 1 つの使用条件のオブジェクトが、複数のビジネス・ポリシー・オブジェクトを参照することもできます。
- 標準価格に対する調整など、ビジネス・ポリシーに対する特定の調整に適用する使用条件。
- ビジネス・プロセスを管理する一連のパラメーターを定義する使用条件。たとえば、特定の契約によって特定の配送センターが使用されることを指定できます。

契約は一連の使用条件からなります。これを以下の図に示します。

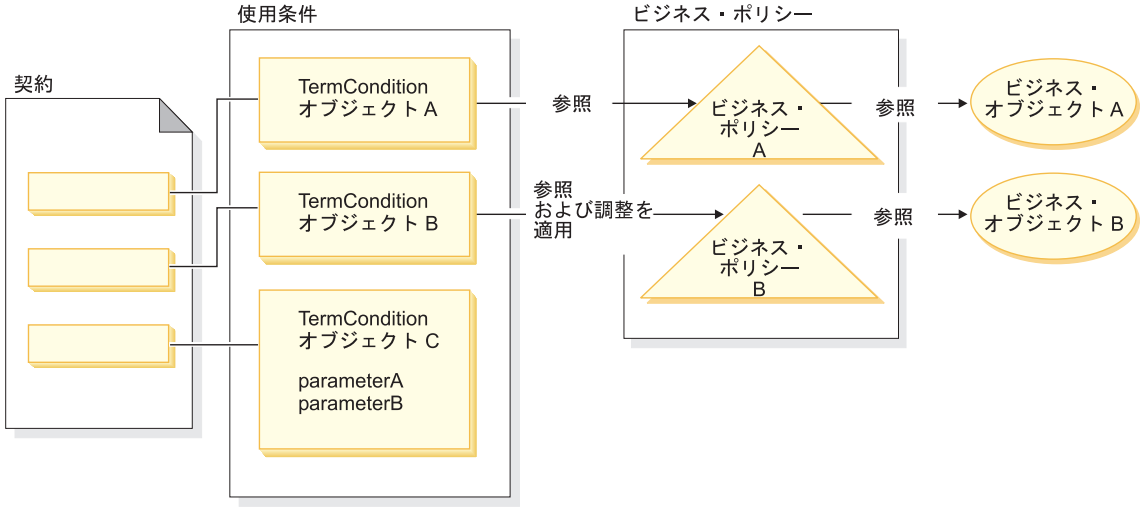


図 29.

上図では、以下の点に気を付けてください。

- 「調整」の語は、ビジネス・ポリシーの変更を指します。たとえば、標準価格に 10% の割引を適用するというように、ビジネス・ポリシーの処理結果に対して割引を適用する場合にこの語を使用することができます。また、一連のパラメーターを使ってビジネス・ポリシーに影響を与える場合にも使用することができます。
- たとえば、TermCondition オブジェクト A を使って、配送条件オブジェクトを表すことができます。この場合、ビジネス・ポリシー A は配送モードのビジネス・ポリシーを表し、ビジネス・オブジェクト A は運送会社 XYZ の配送モード “A3” を表すことができます。
- 別の例として、TermCondition オブジェクト B は、ビジネス・ポリシー B で定義されている価格に 50% の割引を適用する価格条件オブジェクトを表すとします。この場合、ビジネス・ポリシー B は価格ポリシーでありビジネス・オブジェクト B は、マスター・カテゴリー用の商取引配備を定義するオブジェクト位置コンテナーになります。

この章では、プログラマーに、新規のビジネス・ポリシーおよび新規の使用条件を作成する方法のガイドラインを示します。

ToolTech サンプル・ストアを使って、商取引の流れの中の配送条件オブジェクトと価格条件オブジェクトを説明します。この例をサポートする契約データの詳細は、156 ページの 『ToolTech のサンプル契約データ』を参照してください。

ビジネス・ポリシー・オブジェクトおよびコマンド

ビジネス・ポリシー・オブジェクトには以下の情報が含まれています。

- ポリシー ID
  ビジネス・ポリシー・オブジェクトの基本キーです。
- ポリシー・タイプ
  ビジネス・ポリシーのタイプを定義します。ポリシー・タイプの例としては `Price` および `ProductSet` があります。
- ポリシー名
  ビジネス・ポリシーにはそれぞれ固有の名前が必要です。
- ストア・エンティティー
  ビジネス・ポリシーがデプロイメントされるストアまたはストア・グループです。
- プロパティー
  ビジネス・ポリシー・コマンドに渡すことのできる、一連のデフォルト・プロパティーです。ビジネス・ポリシー・オブジェクトに関連したコマンドは BusinessPolicyCmd テーブルに保管されます。
- 有効期間
  ビジネス・ポリシー・オブジェクトが有効な期間です。
- ビジネス・ポリシー・コマンド
  ビジネス・ポリシーをインプリメントするゼロ以上のビジネス・ポリシー・コマンドです。通常、ビジネス・ポリシーは、ビジネス・プロセス (タスク・コマンドまたはコントローラー・コマンドのどちらでもかまいません) によって呼び出されます。たとえば、`getContractPrice()` コマンドは価格条件を取り込みます。その価格条件はある特定の価格ポリシー・コマンドを参照し、そしてその価格ポリシーは、価格の計算に使用されます。

複数のビジネス・ポリシー・コマンドを単一のビジネス・ポリシー・オブジェクトに関連付けることができます。各ビジネス・ポリシー・コマンドが、ビジネス・ポリシー・タイプ・オブジェクトによって定義される同じインターフェースをインプリメントしなければなりません。新規のビジネス・ポリシー・コマンドの構造について以下の図で説明します。

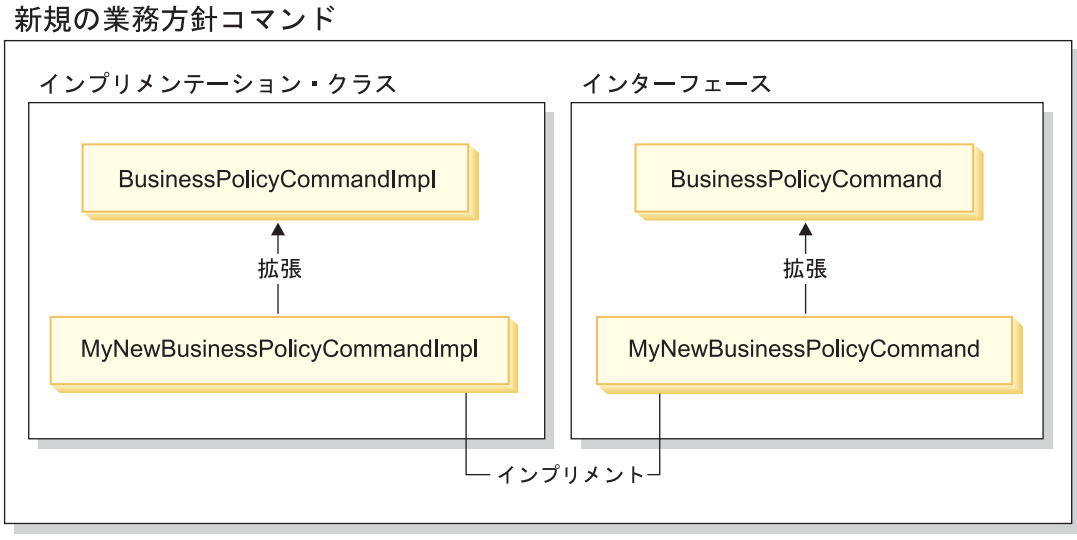


図 30.

上記の図に示されるように、新規のビジネス・ポリシー・コマンドを作成するには、WebSphere Commerce BusinessPolicyCmdImpl インプリメンテーション・クラスを拡張する新規のインプリメンテーション・クラスを作成します。 BusinessPolicyCmd インターフェースを拡張する新規のインターフェースも作成します。

ToolTech のサンプル契約データ

この項では、ToolTech サンプル・ストアで使用される契約データの一部を紹介します。

この後の項のサンプル・データは、データベース・テーブル別に編成されています。関連した行と列だけが示されています。また、サンプルをインストールしたときは、固有 ID (CONTRACT_ID など) がここに示されているものとは異なっている場合があることに注意してください。

CONTRACT テーブルのサンプル・データ

以下の表は、ToolTech の CONTRACT データベース・テーブルの関連サンプル・データを示しています。分かりやすく示すため、データベースの列見出しが最初の列に、テーブルのサンプル・データの列が 2 番目の列に示されていることに注意してください。

列名	サンプル・データ
CONTRACT_ID	10007
MAJORVERSION	1
MINORVERSION	0

列名	サンプル・データ
NAME	ToolTechContractNumber 4567
MEMBER_ID	-2001
ORIGIN	0
STATE	3
USAGE	1
MARKFORDELETE	0

TERMCOND テーブルのサンプル・データ

以下の表は、ToolTech の TERMCOND データベース・テーブルの関連サンプル・データを示しています。分かりやすく示すため、データベースの列見出しが最初の列に、テーブルのサンプル・データの行が 2 番目と 3 番目の列に示されていることに注意してください。

列名	サンプル・データ行 1	サンプル・データ行 2
TERMCOND_ID	10025	10030
TCSUBTYPE_ID	PriceTCPriceListWith SelectiveAdjustment	ShippingTCShippingMode
TRADING_ID	10007	10007
STRINGFILED1	ProductSet2	
INTEGERFIELD2	10002	
INTEGERFIELD3	1	
BIGINTFIELD1	10051	
FLOATFIELD1	-50.0	
SEQUENCE	1	6

POLICYTC テーブルのサンプル・データ

以下の表は、ToolTech の POLICYTC データベース・テーブルの関連サンプル・データを示しています。この表は、ポリシーおよび条件オブジェクトの関係を確立します。

	列名	
	POLICY_ID	TERMCOND_ID
サンプル・データ行 1	10053	10025
サンプル・データ行 2	10056	10030

POLICY テーブルのサンプル・データ

以下の表は、ToolTech の POLICY データベース・テーブルの関連サンプル・データを示しています。

列名	サンプル・データ行 1	サンプル・データ行 2
POLICY_ID	10053	10056
POLICYNAME	MasterCatalogPriceList	A3
POLICYTYPE_ID	Price	ShippingMode
STOREENT_ID	10051	10051
PROPERTIES	name=ToolTech& member_id=-2001	shippingMode=A3
STARTTIME	ヌル	ヌル
ENDTIME	ヌル	ヌル

TRADEPOSCN テーブルのサンプル・データ

以下の表は、ToolTech の TRADEPOSCN データベース・テーブルの関連サンプル・データを示しています。

	列名			
	READEPOSCN_ID	MEMBER_ID	NAME	TYPE
サンプル・データ行	10051	-2001	ToolTech	S

SHIPMODE テーブルのサンプル・データ

以下の表は、ToolTech の SHIPMODE データベース・テーブルの関連サンプル・データを示しています。

	列名			
	SHIPMODE_ID	STOREENTITY_ID	CODE	CARRIER
サンプル・データ行	10053	10051	A3	XYZ Carrier

既存の契約モデルの拡張

契約は、おのおのが 1 つのポリシーを参照する 1 つ以上の条件で構成されます。それをふまえて、この後の項では、新規のビジネス・ポリシーを作成してそれをビジネス・フローに統合するのに必要なステップについて説明します。

以下に、次のようなタスクを実行するための高レベルなステップの概要を述べます。

1. 新しいビジネス・ポリシーを作成します。
   次に示すタスクは、新しいビジネス・ポリシーの作成に関連したタスクです。
   a. 新しいビジネス・ポリシー・タイプの作成 (必要な場合)。
      いくつかのビジネス・ポリシー・タイプが用意されていますが、標準タイプのものが業務上の要件に適していない場合、新規のビジネス・ポリシー・タイプを作成してください。
   b. 新しいビジネス・ポリシー・コマンドの作成。
   c. 新しいビジネス・ポリシーとビジネス・ポリシー・コマンドの登録。
2. 新規のビジネス・ポリシーへの条件オブジェクトの関連付け。
   この関連付けを行うには、既存の条件オブジェクトを新規のビジネス・ポリシーに関連付けるか、または新しい条件オブジェクトを作成します。新しい条件オブジェクトを作成する場合は、次のようなステップを行わなければなりません。
   a. 新しい条件をデータベースに登録する。
   b. 新しい条件を契約 DTD (文書タイプ定義) に登録する。
   c. 条件用の新規エンタープライズ Bean を作成する。
   d. 新しい条件を反映するために WebSphere Commerce アクセラレーターを更新する。
3. ビジネス・フロー内での新規ビジネス・ポリシーの呼び出し。

新しいビジネス・ポリシーの作成

新しいビジネス・ポリシー・コマンドの作成には、一般に、固有のビジネス・ポリシーをデータベースに登録することに加えて、新規ビジネス・ポリシー・コマンドを作成することが関係しています。

新規のビジネス・ポリシー・コマンドの作成には、以下のような高水準のステップが関係しています。

1. 新規のビジネス・ポリシー・タイプの作成 (必要な場合)。
2. 新規のビジネス・ポリシー・コマンドの作成。
3. 新規のビジネス・ポリシーとビジネス・ポリシー・コマンドのデータベースへの登録。

以下のセクションで、これらのステップについて詳しく説明します。

新規のビジネス・ポリシー・タイプの作成

このセクションでは、新規のビジネス・ポリシー・タイプを作成する方法について説明します。ビジネス・ポリシー・タイプは、ポリシーが適用されるトランザクションのレルムを表します。ビジネス・ポリシー・タイプの例には、以下のものがあります。

- Price
- ProductSet
- ShippingMode
- ShippingCharge
- Payment
- ReturnCharge
- ReturnApproval
- ReturnPayment
- InvoiceFormat

既存のビジネス・ポリシー・タイプではユーザーのビジネス要件に合わない場合は、新規のビジネス・ポリシー・タイプを作成してください。新規のビジネス・ポリシー・タイプを作成するには、ビジネス・ポリシー・タイプを定義して登録します。

新規ポリシー・タイプを定義および更新するときは、以下のデータベース・テーブルを更新することが必要です。

- POLICYTYPE
- PLCYTYCMIF
- PLCYTYPDSC

POLICYTYPE テーブルは、作成するビジネス・ポリシーのタイプを指定します。これには単一の列として POLICYTYPE_ID が含まれます。これは基本キーです。値の例としては `Price` があります。新規のビジネス・ポリシー・タイプを作成する場合、固有の POLICYTYPE_ID を指定するようにしてください。

PLCYTYCMIF テーブルは、コマンド・インターフェース関係指定テーブルに対するビジネス・ポリシー・タイプです。つまり、各ビジネス・ポリシー・タイプについて、ビジネス・ポリシー・オブジェクトに関する Java コマンド・インターフェースを指定します。 1 つのビジネス・ポリシーをインプリメントするビジネス・ポリシー・コマンドがゼロ以上ある可能性がありますが、これらのビジネス・ポリシー・コマンドのそれぞれが、ここで指定したインターフェースをインプリメントする必要があります。

PLCYTYPDSC テーブルはビジネス・ポリシー・タイプの説明を指定します。これには、説明の言語 ID と、業務方針タイプの説明が含まれます。

新規のビジネス・ポリシー・タイプを作成するには、新規のビジネス・ポリシー・タイプについての各テーブルにエントリーを作成してください。以下の SQL ステートメントはその例です。

```
insert into POLICYTYPE (POLICYTYPE_ID) values ('MyNewPolicyType');
insert into PLCYTYCMIF (POLICYTYPE_ID, BUSINESSCMDIF)
   values ('MyNewPolicyType',
   'com.mycompany.mybusinesspolicycommands.MyNewPolicy');
insert into  PLCYTYPDSC (POLICYTYPE_ID, LANGUAGE_ID, DESCRIPTION)
   values ('MyNewPolicyType', -1,
   'My new policy type for example purposes.');
```

新規ビジネス・ポリシー・タイプの作成の最終ステップとして、 1 つ以上の新規ビジネス・ポリシー・タイプ・インターフェースをコード化することができます。次いで、それらのインターフェースは、このビジネス・ポリシー・タイプのレルムに入る何らかのビジネス・ポリシー・コマンドによってインプリメントされます。たとえば、ToolTech サンプル・ストアでは、価格 (Price) はビジネス・ポリシー・タイプとして定義されます。そのようなものとして com.ibm.commerce.price.commands.ResolvePriceListsCmd および com.ibm.commerce.price.commands.RetrievePricesCmd インターフェースがあり、これらはすべての価格関係のビジネス・ポリシー・コマンドによってインプリメントされます。

新規ビジネス・ポリシー・タイプで操作を実行するようなビジネス・ポリシー・コマンドを持つ予定がない場合は、新しいインターフェースを作成する必要はありません。そのようなことはまれなので、新規ビジネス・ポリシー・タイプを作成するほとんどのケースでは、新しいビジネス・ポリシー・タイプ・インターフェースも作成する必要があります。

ビジネス・ポリシー・タイプ・インターフェースを作成するときは、新しいインターフェースは com.ibm.commerce.command.BusinessPolicyCommand インターフェースを拡張するものでなければなりません。

新規のビジネス・ポリシー・コマンドの作成

新規ビジネス・ポリシー・コマンドを作成するためには、そのコマンドが関連するビジネス・ポリシー・タイプのインターフェースをインプリメントする、新しいコマンドを作成する必要があります。その新しいコマンドは com.ibm.commerce.command.BusinessPolicyCommandImpl インプリメンテーション・クラスを拡張するものであることも必要です。これは新規のコントローラーまたはタスク・コマンドの作成に非常によく似ています。

入力プロパティーをビジネス・ポリシー・コマンドに渡すには 2 つの方法があります。1 つ目は、POLICY テーブルの PROPERTIES 列にデフォルトの入力プロパティーを指定する方法です。このテーブルについての詳細は以下のセクションを参照してください。

2 つ目は、各入力プロパティーのコマンドに新規フィールドを作成する方法です。各フィールドごとに、新規の getter および setter メソッドの対を作成します。

ビジネス・ポリシー・コマンドでの requestProperties の設定

ビジネス・ポリシー・コマンド・オブジェクトに requestProperties を設定するには、2 つの方法があります。 1 番目の方法では、POLICY テーブルの PROPERTIES 列を使用して、デフォルト・プロパティーを設定します。これは setRequestProperties メソッドによって完了します。プロパティーを設定する 2 番目の方法では、ビジネス・ポリシー・コマンドを呼び出すコマンド (コントローラーまたはタスク) に、他の必要なプロパティーを明示的に設定させます。

新規ビジネス・ポリシー・コマンドを作成する際、デフォルトの setRequestProperties メソッドをオーバーライドして、requestProperties オブジェクトに組み込まれるそれぞれのパラメーターを明示的に設定するためのロジックを組み込むことが必要です。

インターフェース名 MyNewBusinessPolicyCmd とインプリメンテーション・クラス名 MyNewBusinessPolicyCmdImpl をもつ新規のビジネス・ポリシー・コマンドの例について考えてみます。

この新規のビジネス・ポリシー・コマンドに関する POLICY テーブルのエントリーで、 PROPERTIES 列に以下の値が組み込まれるとします。

- defaultProperty1=apple
- defaultProperty2=orange
- defaultProperty3=banana

この新規のビジネス・ポリシー・コマンドのインターフェースは、以下のように定義されます。

```
public interface MyNewBusinessPolicyCmd extends
   com.ibm.commerce.command.BusinessPolicyCmd {
      java.lang.String defaultCommandClassName =
         'com.mycompany.mycommands.MyNewBusinessPolicyCmdImpl';
      public void setProperty1();
      public void setProperty2();
}
```

この新規のビジネス・ポリシー・コマンドのインプリメンテーション・クラスは、以下のように定義されます。

```
public class MyNewBusinessPolicyCmdImpl extends
   com.ibm.commerce.command.BusinessPolicyCmdImpl
   implements com.mycompany.mycommands.MyNewBusinessPolicyCmd {
      // Establish default properties that are stored in the POLICY table

      private java.lang.String defaultProperty1;
      private java.lang.String defaultProperty2;
      private java.lang.String defaultProperty3;
      // Begin to establish properties that must be set
```

```
      // by the calling command.

       //  *** property1   ***
       private java.lang.String property1;
       public java.lang.String getProperty1() {
          return property1;
          }
       public void setProperty1(java.lang.String newProperty1) {
          property1 = newProperty1;
       }

       //  *** property2   ***
       private java.lang.String property2;
       public java.lang.String getProperty2() {
          return property2;
          }
       public void setProperty1(java.lang.String newProperty2) {
          property2 = newProperty2;
          }

       // End establishing properties that must be set
       // by the calling command.
       /* Upon instantiation the business policy command sets all
          default properties from the POLICY table into the
          requestProperties object.  The calling command
          is responsible for setting any other required properties.
       */

       public void setRequestProperties(com.ibm.commerce.datatype.TypedProperty
          requestProperties) {
          //  Get the default properties defined in the POLICY table
          setDefaultProperty1(requestProperties.get("defaultProperty1"));
          setDefaultProperty2(requestProperties.get("defaultProperty2"));
          setDefaultProperty3(requestProperties.get("defaultProperty3"));
          }
}
```

新規のビジネス・ポリシー・コマンドを呼び出すコマンドは、以下に似た方法で定義できます。

```
public class MyCallerCommandImpl
   extends com.ibm.commerce.command.TaskCommandImpl
   implements com.mycompany.mycommands.MyCallerCommand {

   /* Include all elements and processing required for the
      task command.
   */

   // Determine the policy ID and setPolicyId

   // Call the business policy command.

   cmd = (MyNewBusinessPolicyCmd) CommandFactory
            createPolicyCommand(policyId);
```

```
    // Set required properties

    cmd.setProperty1("Fruit salad");
    cmd.setProperty2("Favorite food");

cmd.execute();
}
```

新規のビジネス・ポリシーとビジネス・ポリシー・コマンドの登録

新規のビジネス・ポリシー・コマンドを作成してから、そのビジネス・ポリシーとビジネス・ポリシー・コマンドの両方をデータベースに登録する必要があります。

ビジネス・ポリシーは POLICY テーブルに登録されます。このテーブルには以下の列が含まれています。

- POLICY_ID
  基本キー。ポリシーの ID です。
- POLICYNAME
  固有のポリシー名です。
- POLICYTYPE_ID
  ポリシー・タイプの ID。 POLICYTYPE テーブルの外部鍵です。
- STOREENT_ID
  ポリシーが適用されるストアまたはストア・グループです。
- PROPERTIES
  ビジネス・ポリシー・コマンドに設定できるデフォルトのプロパティーです。
  `parm1=val1&parm2=val2` のように、名前と値の対で指定されます。
- STARTDATE
  ポリシーの開始日 (タイム・スタンプとして指定される) です。 NULL の場合、開始日は即日です。
- ENDDATE
  ポリシーの終了日 (タイム・スタンプとして指定される) です。 NULL の場合、終了日はありません。

新規のポリシーを POLICY テーブルに登録したら、そのポリシーと、ビジネス・ポリシーをインプリメントするビジネス・ポリシー・コマンドの関係を登録する必要があります。このためには POLICYCMD テーブルを使用します。 POLICYCMD テーブルには以下の列が含まれます。

- POLICY_ID
  POLICY テーブルに対する外部鍵の参照です。
- BUSINESSCMDCLASS
  ポリシーをインプリメントするビジネス・ポリシー・コマンドです。

- PROPERTIES
  ビジネス・ポリシー・コマンドに設定できるデフォルトのプロパティーです。
  `parm1=val1&parm2=val2` のように、名前と値の対で指定されます。

新規ビジネス・ポリシーへの条件オブジェクトの関連付け

WebSphere Commerce の契約やポリシーのフレームワークでは、使用条件 (*条件* とも呼びます) によって、バイヤーとセラー間の合意事項を記述する手段が提供されます。使用条件は、契約や RFQ (見積依頼) などのさまざまな取引の合意事項で使用できます。使用条件オブジェクトは、通常、オプショナルの調整を含むビジネス・ポリシーを示します。たとえば、価格の使用条件オブジェクトは、価格設定ポリシー・オブジェクトを 1 つ選択することで作成されます。価格条件では、アカウンティング・マネージャーが、以下のように、ストア標準価格に調整を加えることができます。

- 標準価格リストに対するパーセンテージ割引。
- 指定された商品のセットに対するパーセンテージ割引。

調整はそれぞれ使用条件として指定されます。

新たにビジネス・ポリシーを作成する場合に、そのポリシーを契約で使用する予定であれば、そのビジネス・ポリシーを参照する少なくとも 1 つの条件オブジェクトがなければなりません。既存の条件オブジェクトを新規のビジネス・ポリシーに関連付ける (それには、 `B2BTrading.dtd` ファイル内に既存の条件オブジェクトと新規のビジネス・ポリシーの関係を取り込みます) か、または、新しいビジネス・ポリシーに関連した新しい条件オブジェクトを作成することができます。

新規の使用条件の作成

WebSphere Commerce アーキテクチャーで、新規の使用条件オブジェクトは、以下のステップで作成されます。

1. データベース・スキーマを更新して、新規の使用条件を組み込みます。
2. `B2BTrading.dtd` ファイルを更新して、新規の使用条件を反映させます。
3. 使用条件のために新規のエンタープライズ Bean を作成します。
4. WebSphere Commerce アクセラレーターを更新して新規の条件を反映させるか、契約ロード・コマンドを使用して、新規の条件を使用する新規契約を作成します。

以下のセクションでは、`MyTC` の例が新規の使用条件オブジェクトです。

データベースへの新しい条件の登録

新規の使用条件オブジェクトを作成する際に、データベース・スキーマを更新してこのオブジェクトを組み込まなくてはなりません。更新する必要のあるデータベース・テーブルは、 TCTYPE と TCSUBTYPE です。

以下の SQL ステートメントは、スキーマを更新する方法の例を示しています。

```
insert into TCTYPE (TCTYPE_ID) values ('MyTC');
insert into TCSUBTYPE (TCSUBTYPE_ID, TCTYPE_ID, ACCESSBEANNAME,
   DEPLOYCOMMAND)
values ('MySubTC, 'MyTC ',
   'com.ibm.commerce.contract.objects.MySubTCAccessBean',
   'packagename.MySubTCDeployCmd');
```

契約タイプ定義への新しい条件の登録

B2BTrading.dtd ファイルは、ビジネス・ポリシーで使用できるさまざまな条件を指定する DTD (document type definition (文書タイプ定義)) ファイルです。新規の条件を契約中で使用できるようにするには、このファイルを更新してその新規の条件を組み込む必要があります。

新規の条件を作成するときは、その新条件を TermCondition 定義に追加して、その条件を記述する新規エレメントを作成する必要があります。

B2BTrading.dtd ファイルを更新するには、以下のようにします。

1. 以下のディレクトリーに移動します。
   - Windows *drive*:¥WebSphere¥CommerceServer¥xml¥trading
   - AIX /usr/WebSphere/CommerceServer/xml/trading
   - Solaris /opt/WebSphere/CommerceServer/xml/trading
   - Linux /opt/WebSphere/CommerceServer/xml/trading
   - 400 /QIBM/ProdData/WebCommerce/xml/trading
2. B2BTrading.dtd ファイルをオープンします。
3. 新規の使用条件を使用して TermCondition 定義を更新します。たとえば、この更新は以下の TermCondition 定義では太字で示されています。

   ```
   <!ELEMENT TermCondition (TermConditionDescription?,Participant*,
   CreateTime?,UpdateTime?,(PriceTC|ProductSetTC|ShippingTC|FulfillmentTC|
   PaymentTC|ReturnTC|InvoiceTC|RightToBuyTC|ObligationToBuyTC|
   PurchaseOrderTC|OrderApprovalTC|DisplayCustomizationTC|
   OrderTC|MyTC))>
   ```

   なお、改行がなされているのは、表示上の都合に過ぎません。
4. ここで B2BTrading.dtd ファイルに新規エレメントを追加します。たとえば、以下は MyTC エレメントを追加するための更新を示しています。それはビジネス・ポリシーを参照し、2 つの必須属性があります。

   ```
   <!ELEMENT MyTC (MySubTC)>
   <!ELEMENT MySubTC (PolicyReference)>
   <!ATTLIST MySubTC
      attr1 CDATA #REQUIRED
      attr2 CDATA #REQUIRED
   >
   ```

5. ファイルを保管します。

使用条件のための新規 CMP エンタープライズ Bean の作成

使用条件オブジェクトのために新しい CMP エンタープライズ Bean を作成する必要があります。 bean は使用条件サブタイプ用に作成します。

通常は、新規エンタープライズ Bean を作成するときには、 WebSphere Commerce エンティティー Bean を含む EJB グループのいずれかに bean を入れるのではなく、自分の EJB グループの中に bean を入れます。しかし、このケースでは、使用条件用のすべての新規エンティティー Bean は WebSphere Commerce TermCondition bean から継承する必要があるので、新規の使用条件 bean を WCS Contract EJB グループに入れる必要があります。

使用条件オブジェクトのために新しい CMP エンタープライズ Bean を作成するには、VisualAge for Java で以下のようにしてください。

1. 以下のようにして、ウィザードを使って新規エンタープライズ Bean を作成します。
   a. VisualAge for Java ワークベンチで、EJB タブを選択します。
   b. **WCS Contract** EJB グループを強調表示してから右クリックし、**「追加」>「Enterprise Bean with Inheritance (継承を伴うエンタープライズ Bean)」**を選択します。
      「Create Enterprise Bean with Inheritance (継承を伴うエンタープライズ Bean の作成)」SmartGuide がオープンします。
   c. SmartGuide では、ご自分の bean に合った情報を入力してください。以下の表に値を例を示します。

属性	値
Bean 名	MySubTC
継承元	TermCondition
パッケージ	com.ibm.commerce.contract.objects
Bean クラス	MySubTCBean
リモート・インターフェース	MySubTC
ホーム・インターフェース	MySubTCHome

   d. bean に CMP フィールドを追加するために**「追加」**をクリックし、必要に応じて bean 用の新しいフィールドを作成します。この例では、以下の情報を使用して、2 つの新しい CMP フィールドを作成します。

属性	値
フィールド名	attr1
フィールド・タイプ	String

属性	値
getter および setter メソッドを使ったアクセス	enable
getter および setter メソッドをリモート・インターフェースにプロモートする	enable

属性	値
フィールド名	attr2
フィールド・タイプ	Integer
getter および setter メソッドを使ったアクセス	enable
getter および setter メソッドをリモート・インターフェースにプロモートする	enable

e. **「終了」**をクリックします。

2. 次のステップは、新規 bean から TERMCOND テーブル内の列に、フィールドをマップすることです。このマッピング情報を作成するには、以下のようにします。

   a. **「EJB」**メニューで、**「オープン」>「スキーマ・マップ」**を選択します。マップ・ブラウザーがオープンします。

   b. マップ・ブラウザーの「Datastore Maps (データストア・マップ)」パネルで、**「WCS Contract (WCS 契約)」**をダブルクリックします。

   c. 「永続クラス」パネルで、**TermCondition** をダブルクリックしてから、**MySubTC** を選択します。

   d. 「テーブル・マップ」メニューで、**「新規テーブル・マップ」> 「Add Single Inheritance Table Map (単一継承テーブル・マップの追加)」**を選択します。「Single Inheritance Table Map Editor (単一継承テーブル・マップ・エディター)」がオープンします。

   e. **「Discriminator value (判別値)」**フィールドに、 TCSUBTYPE_ID 値を入力します。たとえば、このケースでは 'MySubTC' (引用符を含める) と入力して**「OK」**をクリックします。

   f. 「永続クラス」パネルで **MySubTC** がまだ選択されていることを確認します。「テーブル・マップ」パネルで、TERMCOND テーブルを強調表示して右クリックします。「Edit Property Maps (プロパティー・マップの編集)」を選択します。プロパティー・マップ・エディターがオープンします。

   g. プロパティー・マップ・エディターでは、以下のように属性を設定します。

クラス属性	マップ・タイプ	テーブル列
attr1	シンプル	STRINGFIELD2

クラス属性	マップ・タイプ	テーブル列
attr2	シンプル	INTEGERFIELD1

それから**「OK」**をクリックします。

h. 「データベース・マップ」メニューから、「データ・ストア・マップの保管」を選択します。マップを保管する際に、以下の情報を入力します。

属性	値
プロジェクト	IBM WCS Enterprise Beans
パッケージ	WCSContract EJB Reserved
クラス名	WCSContractMap

それから**「終了」**をクリックして、マップ・ブラウザーをクローズします。

3. 新規エンタープライズ Bean (つまり、MySubTCBean) で、以下のように、新規 ejbCreate(java.lang.Long argTradingId, org.w3c.dom.Element argElement) メソッドを作成します。

```
public void ejbCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException, javax.ejb.FinderException,
   java.rmi.RemoteException, javax.naming.NamingException,
   javax.ejb.RemoveException {
   _initLinks();
   super.ejbCreate (argTradingId, argElement);
   this.attr1= null;
   this.attr2= null;
}
```

4. 以下のようにして、新規の ejbPostCreate(java.lang.Long argTradingId, org.w3c.dom.Element argElement) メソッドを作成します。

```
public void ejbPostCreate(java.lang.Long argTradingId,
   org.w3c.dom.Element argElement)
   throws javax.ejb.CreateException, javax.ejb.FinderException,
   java.rmi.RemoteException, javax.naming.NamingException,
   javax.ejb.RemoveException
   {
   parseXMLElement(argElement);
   }
```

5. 以下のように、MySubTCBean の parseXMLElement(org.w3c.dom.Element argElement) メソッドをオーバーライドします。

```
public void parseXMLElement(org.w3c.dom.Element argElement) throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException,
   javax.ejb.RemoveException
   {
      super.parseXMLElement(argElement);
```

```
        if (argElement == null)
           return;

        String nodeName = argElement.getNodeName();
        if (nodeName.equals("TCCopy"))
           return;
        // get element "MyTC"
        Element eMyTC = ContractUtil.getElementByTag(argElement,"MyTC");

        // get element "MySubTC" from element "MyTC"
        Element eMySubTC = ContractUtil.getElementByTag(eMyTC ,"MySubTC");
        this.attr1 = eMySubTC .getAttribute("attr1").trim();
        this.attr2 = new Integer (eMySubTC .getAttribute("attr2").trim());

        // get element "PolicyReference" from "MySubTC"
        Element ePolicyReference = ContractUtil.getElementByTag(eMySubTC,
           "PolicyReference");
        parseElementPolicyReference(ePolicyReference);

    }
```

6. 以下のように、MySubTCBean の createNewVersion(Long argNewTradingId) メソッドをオーバーライドします。

```
    public Long createNewVersion(Long argNewTradingId) throws
       javax.ejb.CreateException,
       javax.ejb.FinderException,
       java.rmi.RemoteException,
       javax.naming.NamingException,
       javax.ejb.RemoveException,
       org.xml.sax.SAXException,
       java.io.IOException

       {

       // Contract a seqElement since tcSequence can not be null
       Element seqElement = ContractUtil.getSeqElementFromTCSequence(
          this.tcSequence);
       MySubTCAccessBean newTC = new MySubTCAccessBean(argNewTradingId,
         seqElement);
       Long newTCId = newTC.getReferenceNumberInEJBType();
       newTC.setInitKey_referenceNumber(newTCId.toString());
       newTC.setMandatoryFlag(this.mandatoryFlag);
       newTC.setChangeableFlag(this.changeableFlag);
       // set columns for this specific TC
       newTC.setAttr1(this.attr1);
       newTC.setAttr2(this.attr2);
       newTC.commitCopyHelper();
       return newTCId;
    }
```

7. 以下のように、MySubTCBean の getXMLString() メソッドをオーバーライドします。

```
public String getXMLString() throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException
   {
      String xmlTC = "      <MyTC>"  +
                     "%XML_POLICYREFERENCE%" +
                     "         <MySubTC attr1=¥"" + this.attr1 +
                     "¥"  attr2=¥"" + this.attr2.toString() + "¥"/>"
                     "         '>" +
                     "      <MYTC></MYTC>" ;
      String xmlPolicy = getXMLStringForElementPolicyReference(
         "ProductSet") ;
      xmlTC = ContractUtil.replace( xmlTC, "%XML_POLICYREFERENCE%",
         xmlPolicy );

      return xmlTC;
}
```

8. 以下のように、MySubTCBean の markForDelete() メソッドをオーバーライドします。

```
public void markForDelete() throws
   javax.ejb.CreateException,
   javax.ejb.FinderException,
   java.rmi.RemoteException,
   javax.naming.NamingException
   {
      // code: remove entries from associated tables which
      // cannot be deleted though delete cascade
}
```

9. ejbCreate メソッドがホーム・インターフェースに追加され、変更された他のすべてのメソッドがリモート・インターフェースに追加されたことを確認してください。
10. 次のステップは、以下のようにして、 MySubTC エンティティー Bean 用のアクセス Bean を作成することです。
    a. **MySubTC** エンティティー Bean を右クリックし、**「追加」> 「アクセス Bean」**を選択します。
       アクセス Bean の作成 SmartGuide がオープンします。
    b. 以下の情報が入力されていることを確認してください。

表 1.

属性	値
EJB グループ	WCSContract
Enterprise Bean	MySubTC
Access Bean 名	MySubTCAccessBean
Access Bean タイプ	Entity Bean 用のコピー・ヘルパー

そして**「次へ」**をクリックします。

c. **「ゼロ引き数コンストラクターのホーム・メソッドを選択する」**ドロップダウン・リストから、 **findByPrimaryKey(TermConditionKey)** を選択します。

d. **initKey_referenceNumber** (「初期プロパティー」列にある) では、**コンバーター** を `com.ibm.commerce.base.objects.WCStringConverter` に変更して、**「次へ」**をクリックします。

e. 追加したすべての新規フィールドについて、**CopyHelper** が選択されており、それぞれのコンバーター値が `com.ibm.commerce.base.objects.WCStringConverter` に設定されていることを確認してください。**「終了」**をクリックします。
コード生成が完了したら、新しいコードを見ることができます。そのためには、**「プロジェクト」**タブに切り替え、 **IBM WCS Enterprise Beans** プロジェクトを拡張表示し、次いで **com.ibm.commerce.contract.objects** パッケージを拡張表示します。

11. EJB タブに戻り、**MySubTC** enterprise bean を右クリックして、**「デプロイメント・コードの生成」**を選択します。

12. さらに、親 bean (TermCondition bean) およびすべての兄弟 bean (名前に “TC” が含まれる、 WCS Contract EJB グループ内の他のすべての bean) のためにデプロイされたコードを再生成する必要もあります。注意点として、新しいフィールドを追加した場合や、既存の TermCondition bean のリモート・インターフェースを変更した場合には、アクセス bean をそれ自身のため、またそのすべての子 bean のために再生成する必要もあります。
デプロイメント・コードを再生成するために、以下のようにします。

a. TermCondition bean と、名前に “TC” が含まれる他のすべての bean を強調表示します (たとえば、DisplayCustomizationTC、FulfillmentTC、および InvoiceTC は兄弟 bean の少数の例です)。

b. これらすべての bean を強調表示したまま、**「Generate Deployed Code (デプロイされたコードの生成)」**を右クリックして選択します。

13. 次のステップでは、`ValidateContractCmd` タスク・コマンドにあるメソッドをオーバーライドします。このコマンドでは、新規の使用条件オブジェクトをサポートするためにオーバーライドする必要のあるメソッドが 3 つあります。それは以下のとおりです。

- `validateTCType()`
  このメソッドは、契約の中に入れることができる条件のタイプを検査します。たとえば、InvoiceTC はアカウントに属するので、契約には表れません。
- `validateTCOccurrence()`
  このメソッドは、条件の出現を検査します。たとえば、このメソッドのデフォルトのインプリメンテーションでは、契約には少なくとも 1 つの PriceTC が必要です。
- `otherValidateCheck()`
  このメソッドのデフォルトのインプリメンテーションは空です。最初の 2 つのメソッドにあてはまらない任意の妥当性検査を追加することができます。

14. 使用条件をデプロイメントしなければならない場合は、新規のデプロイメント・コマンドを作成して、このコマンドをデータベースに登録しなければなりません。必要に応じて以下のようにします。

    a. この例では、新規のデプロイメント・コマンド・インターフェースは MySubTCDeployedCmd といい、インプリメンテーション・クラスは MySubTCDeployedCmdImpl といいます。さらに、コマンドは packagename パッケージにパッケージされます。このコマンドを登録するには、以下の SQL コマンドを実行します。

    ```
    insert into CMDREG (STOREENT_ID, INTERFACENAME, CLASSNAME, TARGET)
    values (0, 'packagename.MySubTCDeployCmd',
    'packagename.MySubTCDeployCmdImpl', 'Local');
    ```

    b. packagename パッケージで、新規の MySubTCDeployedCmd インターフェースを作成します。このインターフェースは com.ibm.commerce.contract.commands.DeployTCCmd コマンド・インターフェースを拡張しなければなりません。以下は新規コマンド・インターフェースの記述です。

    ```
    public interface MySubTCDeployCmd extends
       com.ibm.commerce.contract.commands.DeployTCCmd
       {
           // customized code
       }
    ```

    DeployTCCmd には、保護パラメーター abTC と、getTargetStoreId() というメソッドがあります。 abTC の値は MySubTCAccessBean であり、getTargetStoreId() メソッドは契約がデプロイメントされるストアの ID を戻します。

    c. 同じパッケージで、MySubTCDeployCmdImpl インプリメンテーション・クラスを作成します。このインプリメンテーション・クラスは com.ibm.commerce.contract.commands.DeployTCCmdImpl を拡張しなければなりません。以下は新規コマンド・インプリメンテーション・クラスの記述です。

    ```
    public class MySubTCDeployCmdImpl
       extends com.ibm.commerce.contract.commands.DeployTCCmdImpl
       implements MySubTCDeployCmd
       {
         // customer code
       }
    ```

新規の条件を使用するための WebSphere Commerce アクセラレーターの更新

新規の使用条件を作成したら、 WebSphere Commerce アクセラレーターを更新して、その新規の使用条件を組み込む新規の契約の作成に使用できます。この目的のために WebSphere Commerce アクセラレーターを更新するステップは以下のとおりです。

1. 新規の使用条件のための、新規の JavaScript ファイルを作成します。このセクションの例では、このファイルは Extensions.js といいます。

2. ユーザーが新規の使用条件についての必須情報を入力できる HTML セクションを組み込んだ、新規の JSP テンプレートを作成します。このセクションの例では、このファイルは ContractMyTC.jsp といいます。
3. 新規の使用条件のための、新規のデータ Bean を作成します。このセクションの例では、このファイルは MyTCDataBean といいます。
4. VIEWREG テーブルで新規ビューを登録します。
5. ContractRB_*locale*.properties ファイルを更新して、新規のリソースを組み込みます。
6. ContractNotebook.xml ファイルを編集して、新規のページを組み込みます。

以下のセクションで、これらのステップについて詳しく説明します。

新規の JavaScript ファイルの作成: 新規の使用条件を使用するために WebSphere Commerce アクセラレーターを更新する最初のステップは、これらのための新規の JavaScript ファイルを作成することです。以下のサンプル・ファイルを参照できます。

- Windows *drive*:¥WebSphere¥CommerceServer¥samples¥contract¥Extensions.js
- Windows *drive*:¥WebSphere¥CommerceServerDev¥samples¥contract¥Extensions.js
- AIX /usr/WebSphere/CommerceServer/samples/contract/Extensions.js
- Solaris /opt/WebSphere/CommerceServer/samples/contract/Extensions.js
- Linux /opt/WebSphere/CommerceServer/samples/contract/Extensions.js
- 400 /QIBM/ProdData/WebCommerce/samples/contract/Extensions.js

このサンプル・ファイルを使用するには、以下のディレクトリーにこのファイルをコピーしてください。

- Windows *drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wctools.war¥ javascript¥tools¥contract
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/ javascript/tools/contract
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/ javascript/tools/contract
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/ javascript/tools/contract
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wctools.war/javascript/tools/contract

この新規ファイルで、JavaScript オブジェクトを作成して、新規の使用条件についてのデータを保管しなければなりません。これは以下のコードの断片に示されています。

```
function ContractMyTCModel() {

   this.tcReferenceNumber = "";
   this.policyReferenceNumber = "";

   this.attr1 = "";
   this.attr2 = "";

   this.policyList = new Array();
   this.selectedPolicyIndex = "0";
}
```

新規の JavaScript オブジェクトを作成して、新規の使用条件の送信も行わなければなりません。これは B2BTrading.dtd ファイルに行った拡張と整合性のある方法で行ってください。これは以下のコードの断片に示されています。

```
function submitMyTC(termsAndConditions) {

   var tcModel = get("ContractMyTCModel");

   if (tcModel != null) {

      var myTC = new Object();
      myTC.MyTC = new Object();
      myTC.MyTC.MySubTC = new Object();
      myTC.MyTC.MySubTC.attr1 = tcModel.attr1;
      myTC.MyTC.MySubTC.attr2 = tcModel.attr2;

      myTC.MyTC.PolicyReference = new Object();
      myTC.MyTC.PolicyReference.policyName =
         tcModel.policyList[tcModel.selectedPolicyIndex].policyName;
      myTC.MyTC.PolicyReference.policyType = "ProductSet";
      myTC.MyTC.PolicyReference.storeIdentity =
         tcModel.policyList[tcModel.selectedPolicyIndex].storeIdentity;
      myTC.MyTC.PolicyReference.Member =
         tcModel.policyList[tcModel.selectedPolicyIndex].member;

      if (tcModel.tcReferenceNumber != "") {
         // Change the term and condition
         myTC.action = "update";
         myTC.referenceNumber = tcModel.tcReferenceNumber;
      }
      else {
         // Create a new term and condition
         myTC.action = "new";
      }

      termsAndConditions[termsAndConditions.length] = myTC;
   }

   return true;
}
```

新規の JSP テンプレートの作成: 次のステップは、新規の使用条件で必要とされる情報をユーザーが入力できる HTML セクションを組み込んだ、新規の JSP テンプレートの作成です。以下のサンプル・ファイルを参照できます。

- Windows *drive*:¥WebSphere¥CommerceServer¥samples¥contract¥ContractMyTC.jsp
- Windows *drive*:¥WebSphere¥CommerceServerDev¥samples¥contract¥ ContractMyTC.jsp
- AIX /usr/WebSphere/CommerceServer/samples/contract/ContractMyTC.jsp
- Solaris /opt/WebSphere/CommerceServer/samples/contract/ContractMyTC.jsp
- Linux /opt/WebSphere/CommerceServer/samples/contract/ ContractMyTC.jsp
- 400 /QIBM/ProdData/WebCommerce/samples/contract/ContractMyTC.jsp

このサンプル・ファイルを使用するには、以下のディレクトリーにこのファイルをコピーしてください。

- Windows *drive*:¥WebSphere¥AppServer¥installedApps¥ WC_Enterprise_App_*instanceName*.ear¥wctools.war¥tools¥contract
- AIX /usr/WebSphere/AppServer/installedApps/ WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- Solaris /opt/WebSphere/AppServer/installedApps/ WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- Linux /opt/WebSphere/AppServer/installedApps/ WC_Enterprise_App_*instanceName*.ear/wctools.war/tools/contract
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/ installedApps/WC_Enterprise_App_*instanceName*.ear/ wctools.war/tools/contract

以下のコードの断片は、MyTC のために使用できる JSP テンプレートの HTML セクションの例を示しています。

```
<!--
////////////////////////////////////
// HTML SECTION
////////////////////////////////////
-->

<BODY onLoad="onLoad()" class="content">

   <H1>
   <%= contractsRB.get("MyTCHeading") %>
   </H1>

<FORM NAME="MyTCForm">
```

```
   <%= contractsRB.get("MyTCAttr1Label") %>
   <BR>
   <INPUT type=text name=Attr1 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCAttr2Label") %>
   <BR>
   <INPUT type=text name=Attr2 value="" size=10 maxlength=10>
   <BR>

   <%= contractsRB.get("MyTCPolicyLabel") %>
   <BR>
   <SELECT NAME="PolicyList" SIZE="1">
   </SELECT>

</FORM>
```

新規のデータ Bean の作成: このステップでは、MySubTC アクセス Bean から必要なデータをロードする、新規のデータ Bean を作成します。関係のあるセクションのコードが、以下のコードの断片に示されています。

```
public class MyTCDataBean extends MySubTCAccessBean
   implements SmartDataBean, Delegator {
      private java.lang.Long contractId;
      private boolean hasMyTC = false;
      private CommandContext iCommandContext;


      /**
       * MyTCDataBean default constructor.
       */
      public MyTCDataBean() {
      }
      /**
       * MyTCDataBean constructor.
       */
      public MyTCDataBean(Long newContractId) {
       contractId = newContractId;
      }


      /*
       * populate the attributes from TermConditionAccessBean
       */
      public void populate() throws Exception {

            Enumeration myTCEnum = new TermConditionAccessBean().
                     findByTradingAndTCSubType(contractId, "MySubTC");
                  if (myTCEnum != null) {
                     // assume a contract only has one MyTC for this example

                     setEJBRef(((TermConditionAccessBean)
                           myTCEnum.nextElement()).getEJBRef());
                     refreshCopyHelper();
```

```
                    hasMyTC = true;
                }

}
```

VIEWREG テーブルへの新規ビューの登録: 新規に作成したビューを VIEWREG テーブルに登録しなければなりません。以下は、新規のビューを登録するための SQL ステートメントの例です。

```
insert into VIEWREG(VIEWNAME,DEVICEFMT_ID,STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, HTTPS, INTERNAL)
values ('ContractMyTCPanelView', -1, 0,
   'com.ibm.commerce.tools.command.ToolsForwardViewCommand',
   'com.ibm.commerce.tools.command.ToolsForwardViewCommandImpl',
   'docname=tools/contract/ContractMyTC.jsp', 1, 1)
```

ContractRB_locale.properties ファイルの更新: 新規の使用条件に特定の情報を使用して、以下のプロパティー・ファイルを更新しなければなりません。

- Windows *drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥properties¥com¥ibm¥commerce¥tools¥contract¥properties¥ContractRB_*locale*.properties
- AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ContractRB_*locale*.properties
- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ContractRB_*locale*.properties
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ ContractRB_*locale*.properties
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/properties/com/ibm/commerce/tools/contract/properties/ContractRB_*locale*.properties

以下は、ファイルに追加する情報の例です。

```
MyTCHeading=My TC
attr1Empty=Attribute One must be entered.
attr2Empty=Attribute Two must be entered.
attr1TooLong=Attribute One is too long.
attr2TooLong=Attribute Two is too long.
MyTCAttr1Label=Attribute One (required)
MyTCAttr2Label=Attribute Two (required)
MyTCPolicyLabel=Policy
```

ContractNotebook.xml ファイルの編集: WebSphere Commerce アクセラレーターに新規の使用条件を組み込むための最後のステップは、以下のファイルを更新して新規ページを組み込むことです。

- Windows *drive*:¥WebSphere¥CommerceServer¥xml¥tools¥contract¥ContractNotebook.xml
- AIX /usr/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- Solaris /opt/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- Linux /opt/WebSphere/CommerceServer/xml/tools/contract/ContractNotebook.xml
- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/xml/tools/contract/ContractNotebook.xml

以下は、この例で新規ページを組み込むために使用されるコードの断片の例です。

```
<panel name="MyTCHeading"
    url="ContractMyTCPanelView"
    parameters="contractId,accountId"
    helpKey="MC.contract.MyTCPanel.Help" />
```

新規の使用条件を使用する新規契約のインポート

新規の使用条件を使用するために WebSphere Commerce ツールを変更する代わりに、契約をインポートするコマンド (このコマンドについての情報は WebSphere Commerce のオンライン・ヘルプを参照) を使用して、この新規の使用条件が組み込まれる新規の契約をインポートすることができます。インポート後、 Contract.xml ファイル内の関係のあるセクションは以下のように表示されます。

```
<TermCondition>
   <MyTC>
      <MySubTC attr1="adc" attr2="123" />
         <PolicyReference policyName = "Product Set 1"
                          policyType = "ProductSet"
                          storeIdentity = "StoreGroup1" >
            <Member>
               <User distinguishName = "uid=wcsadmin,o=Root Organization"/>
            </Member>
         </PolicyReference>
      </MyTC>
 </TermCondition>
```

新規のビジネス・ポリシーの呼び出し

新たにビジネス・ポリシーを作成し、少なくとも 1 つ以上の条件オブジェクトにそのビジネス・ポリシーを関連付けし終わったら、新規のビジネス・ポリシー・コマンドを呼び出すためにアプリケーション・ロジックを更新する必要があります。

ビジネス・ポリシー・コマンドは、コントローラー・コマンドおよびタスク・コマンドから呼び出されます。

ビジネス・ポリシー・コマンドを呼び出すには、コマンド・ファクトリーを使用します。 2 つの create メソッドを使用して、ビジネス・ポリシー・コマンドを呼び出すことができます。 1 つ目のメソッドは、ビジネス・ポリシーに関連したビジネス・ポリシー・コマンドが 1 つしかないときに、ビジネス・ポリシー・コマンドを呼び出すために使用します。これは、以下のコードの断片のようになります。

```
CommandFactory createBusinessPolicyCommand(Long policyId);
```

2 つ目のメソッドは、ビジネス・ポリシーに関連したビジネス・ポリシー・コマンドが複数あるときに、ビジネス・ポリシー・コマンドを呼び出すために使用します。これは、以下のコードの断片のようになります。

```
CommandFactory createBusinessPolicyCommand(Long policyId, String cmdIfName);
```

上記の例で、cmdIfName は、作成されるビジネス・ポリシー・コマンドのインターフェース名を指定します。

コマンド・ファクトリーは、 POLICYCMD テーブル内でポリシー・オブジェクトを検索して、このポリシーをインプリメントするコマンドを判別します。また、このテーブルから任意のデフォルトのプロパティーを取り出して、ビジネス・ポリシー・コマンド内で requestProperties として設定します。

以下のコードの断片は、リファンド・ポリシーの呼び出しの例を示しています。

```
RefundPolicyCmd  cmd;

   ////////////////////////////////////////////////////////
   // Get the refund policy id from the refundTC object     //
   //  and use it to create the policy command.             //
   ////////////////////////////////////////////////////////
cmd = (RefundPolicyCmd) CommandFactory
         createPolicyCommand (refundTC.getRefundPolicy);

  cmd.execute()
```

契約の作成

次のステップでは、契約モデルに対する拡張を完全にビジネス・プロセスに統合するために、新しいビジネス・ポリシーを参照する条件を取り入れた契約を作成します。契約を作成するには、WebSphere Commerce アクセラレーターを使用するか、または契約 URL コマンド (ContractImportApprovedVersion および ContractImportDraftVersion) のうちのいずれかを使用します。契約の作成に関する詳細は、WebSphere Commerce のオンライン・ヘルプを参照してください。

契約のカスタマイズのシナリオ

このセクションでは、以下のカスタマイズのシナリオにかかわるステップの概要を述べます。

- リベートを活動化する。

リベートのシナリオ

このシナリオ例では、一定率のリベートを作成します。 ToolTech サンプル・ストアには、リベート・シナリオに一致する使用条件もポリシー・タイプもないので、それらを作成する必要があります。それに加えて、新規ビジネス・ポリシーと、リベートのコードを格納するデータベース・テーブルも作成する必要があります。

このリベート・シナリオの実装には、次のような高レベルのステップが含まれます。

1. XREBATECODE データベース・テーブルとそれに対応する XRebateCodeBean エンティティー Beanを作成します。このエンティティー Bean は、同テーブルから情報を利用するために使用されます。
2. 以下のサブタスクを実行することにより、新しい 5DollarRebate ビジネス・ポリシーを作成します。
   a. 対応する新規ビジネス・ポリシー・タイプを作成します。これにより、新規ビジネス・ポリシー・コマンドがインプリメントすることになるインターフェース (RebatePolicyCmd) が定義されます。
   b. 新しい CalculateRebateCmdImpl ビジネス・ポリシー・コマンドを作成します。
   c. 新規ビジネス・ポリシー・コマンドとビジネス・ポリシー・タイプをデータベースに登録します。
3. 以下のサブタスクを実行することにより、リベートのための使用条件 (RebateTC) を作成します。
   a. RebateTC 使用条件をデータベースに登録します。
   b. 新しい RebateTC を反映するように `B2BTrading.dtd` ファイルを更新します。
   c. RebateTC 用の新しいエンタープライズ Bean を作成します。
   d. 新しい RebateTC を反映するように WebSphere Commerce アクセラレーターを更新します。

4. RebateTC を使用する新しい使用条件を作成します。
5. 新規ビジネス・ポリシーをショッピング・フローに統合します。

以下のセクションでは、これらのステップをそれぞれ詳しく説明します。

ステップ 1: 新規テーブルおよびエンタープライズ Bean の作成

既存のデータベース・スキーマにはリベートの金額やコードの仕様が含まれていないので、新しいテーブルを作成する必要があります。一般に、新規テーブルを作成するときに、そのテーブル内に含まれる情報にアクセスする際に使用される新規エンティティー Bean も作成されます。

この例の目的上、以下の XREBATECODE データベース・テーブルが作成されることを想定します。

表 2. XREBATECODE データベース・テーブル

	列名		
	REBATECODE_ID	AMOUNT	CURRENCY
サンプル・データ	201	5	CAD
	202	10	CAD

加えて、新しい CMP エンティティー Bean (XRebateCodeBean) も作成できます。この bean の作成の詳細については、62 ページの 『CMP エンタープライズ Bean の新規作成』 を参照してください。

ステップ 2: “5DollarRebate” ビジネス・ポリシーの作成

この新しいビジネス・ポリシーを作成するには、以下のステップを実行する必要があります。

1. 新規ビジネス・ポリシー・タイプ・インターフェースを作成します。これは `RebatePolicyCmd` インターフェースで、 `CalculateRebateCmdImpl` がインプリメントすることになります。
2. 新しい `CalculateRebateCmdImpl` ビジネス・ポリシー・コマンドを作成します。
3. 新しいビジネス・ポリシーとビジネス・ポリシー・コマンドをデータベースに登録します。

“Rebate” ビジネス・ポリシー・タイプの作成: リベートに対応する既存のビジネス・ポリシー・タイプがないので、新規に作成する必要があります。新規ビジネス・ポリシー・タイプの作成には、ポリシー・タイプを定義してデータベースに登録することが関係しています。以下のテーブルを更新する必要があります。

- POLICYTYPE
- PLCYTYCMIF
- PLCYTYPDSC

このシナリオでは、新しい REBATE ポリシー・タイプを作成するために、以下の SQL ステートメントを使用します。

```
insert into POLICYTYPE (POLICYTYPE_ID) values ('Rebate');
insert into PLCYTYCMIF (POLICYTYPE_ID, BUSINESSCMDIF)
   values ('Rebate',
   'com.mycompany.mybusinesspolicycommands.RebatePolicyCmd');
insert into  PLCYTYPDSC (POLICYTYPE_ID, LANGUAGE_ID, DESCRIPTION)
   values ('Rebate', -1,
   'Rebate policy type.');
```

結果として、PLCYTYCMIF テーブルの関係のある列は、次の表のようになります。この表は、ポリシー・タイプと、それが関連付けられるビジネス・ポリシー・コマンドとの関係を示しています。

表 3. PLCYTYCMIF テーブルになされる更新

	列名	
	POLICYTYPE_ID	BUSINESSCMDIF
サンプル・データ	Rebate	com.mycompany.mybusinesspolicycommands.RebatePolicyCmd

新しい RebatePolicyCmd インターフェースをコード化する必要もあります。このインターフェースは、 com.ibm.commerce.command.BusinessPolicyCommand インターフェースを拡張したものでなければなりません。前述のテーブルに示唆されているように、このインターフェースを自分のパッケージに組み込んでください。

CalculateRebateCmdImpl ビジネス・ポリシー・コマンドの作成: 新しいビジネス・ポリシー・コマンドを作成するには、 CalculateRebateCmdImpl という新しいコマンドを作成する必要があります。これは com.ibm.commerce.command.BusinessPolicyCommandImpl インプリメンテーション・クラスを拡張したものです。このコマンドは、前のステップで作成された RebatePolicyCmd インターフェースをインプリメントする必要があります。

この例では、インターフェース名とコマンド名が似ていないのでご注意ください。これらの名前は意図的に選ばれたもので、ビジネス・ポリシーのリベート・タイプをインプリメントする多くのビジネス・ポリシー・コマンドがありうることを示しています。そして、各インプリメンテーション (すなわち、各ビジネス・ポリシー・コマンド) が、それぞれ固有の方法でリベートをインプリメントすることになります。

コマンドのロジックは、顧客が商品を選出する方法についての特定のインプリメンテーションに依存しています。加えて、この CalculateRebateCmdImpl は、別のコントローラーまたはアプリケーションのタスク・コマンドによって起動する必要があります。

新規のビジネス・ポリシーとビジネス・ポリシー・コマンドの登録: 新規ビジネス・ポリシーはデータベースに登録する必要があります。また、新規ビジネス・ポリシーと新規ビジネス・ポリシー・コマンドの関係も登録する必要があります。

この情報を登録するためには、`com.ibm.commerce.contract.commands.PolicyAddCmd` コマンドを使用することができます。このシナリオのための PolicyAdd コマンドの使用例を以下に示します。

```
http://localhost:8080/webapp/wcs/stores/servlet/PolicyAdd?
   type=Rebate&name=5DollarRebate&plcyStoreId=-1
   &cmd_1=com.mycompany.mybusinesspolicycommands.CalculateRebateCmdImpl
   &startDate=2002-05-08%2000:00:00&endDate=2003-05-09%2000:00:00
   &commonProps=rebatecode_id%3D501&URL=aRedirectURL
```

注意点として、URL 予約文字は入力プロパティーのための ASCII コードに置き換える必要があります。したがって、通常の = (等号) は “%3D” に置き換え、 & (アンパーサンド) は “%26” に置き換え、スペース文字は “%20” に置き換えます。前述の例で使用される日付形式は yyyy-mm-dd hh:mm:ss であり、 URL 予約文字を置き換える ASCII コードが使用されています。

以下の表は、更新の実行後に影響を受けるデータベース・テーブルの関係のある列を示しています。

表 4. POLICY テーブルになされる更新

	列名				
	POLICY _ID	POLICY NAME	POLICYTYPE _ID	STOREENT _ID	PROPERTIES
サンプル・データ	301	5Dollar Rebate	Rebate	-1	rebatecode_id= 201

開始日と終了日の値が `null` に設定されていることも想定されています。

表 5. POLICYCMD テーブルになされる更新

	列名		
	POLICY _ID	BUSINESS CMDCLASS	PROPERTIES
サンプル・データ	301	com.mycompany. mybusinesspolicycommands. CalculateRebateCmdImpl	ヌル

結果として、CalculateRebateCmd ビジネス・ポリシー・コマンドに関連する “5DollarRebate” という新しいビジネス・ポリシーができました。

ステップ 3: “RebateTC” 使用条件の作成

“RebateTC” 使用条件を作成するためには、以下のステップを実行する必要があります。

1. RebateTC 使用条件をデータベースに登録します。
2. 新しい RebateTC を反映するように B2BTrading.dtd ファイルを更新します。
3. RebateTC 用の新しいエンタープライズ Bean を作成します。
4. 新しい RebateTC を反映するように WebSphere Commerce アクセラレーターを更新します。

データベースへの “RebateTC” 条件の登録: 新規の使用条件オブジェクトを作成する際に、データベース・スキーマを更新してこのオブジェクトを組み込まなくてはなりません。更新する必要のあるデータベース・テーブルは、 TCTYPE と TCSUBTYPE です。

以下の SQL ステートメントは、データベースに RebateTC を登録する方法の例を示しています。

```
insert into TCTYPE (TCTYPE_ID) values ('RebateTC');
insert into TCSUBTYPE (TCSUBTYPE_ID, TCTYPE_ID, ACCESSBEANNAME,
   DEPLOYCOMMAND)
values ('RebateTC, 'RebateTC ',
   'com.ibm.commerce.contract.objects.RebateTCAccessBean',
    null);
```

以下の表は、TCTYPE および TCSUBTYPE テーブル内の関係のある列を抜き出したものです。

表 6. TCTYPE テーブルになされる更新

	列名
	TCTYPE_ID
サンプル・データ	RebateTC

表 7. TCSUBTYPE テーブルになされる更新

	列名			
	TCSUBTYPE _ID	TCTYPE _ID	ACCESSBEAN NAME	DEPLOY COMMAND
サンプル・データ	RebateTC	RebateTC	com.ibm.commerce. contract.objects. RebateTCAccessBean	ヌル

新しい RebateTC を反映するように B2BTrading.dtd ファイルを更新する:
契約で新しい使用条件を使用できるようにするには、 B2BTrading.dtd ファイルを更新して、新しい使用条件を組み込む必要があります。このファイルを更新する際に、新しい使用条件を TermCondition 定義に追加してから、その使用条件を記述する新規エレメントを作成する必要があります。

太字のテキストは、新規 RebateTC を TermCondition 定義に追加する方法を示す例です。

```
<!ELEMENT TermCondition (TermConditionDescription?,Participant*,
CreateTime?,UpdateTime?,(PriceTC|ProductSetTC|ShippingTC|FulfillmentTC|
PaymentTC|ReturnTC|InvoiceTC|RightToBuyTC|ObligationToBuyTC|
PurchaseOrderTC|OrderApprovalTC|DisplayCustomizationTC|
OrderTC|RebateTC))>
```

なお、改行がなされているのは、表示上の都合に過ぎません。

次に、この条件を記述する新しいスタンザをファイルに追加する必要があります。RebateTC の例を以下に示します。

```
<!ELEMENT RebateTC (PolicyReference?)>
```

RebateTC のための新規エンタープライズ Bean の作成: 新しい RebateTC のための新規エンタープライズ Bean を作成する必要があります。この新しい bean は WebSphere Commerce TermCondition bean から継承する必要があります。

使用条件用の新しいエンタープライズ Bean は、一般にサブタイプにちなんで命名されます。このケースでは使用条件サブタイプは使用条件タイプと同じなので、 bean の名前は使用条件タイプと同じになります。

以下の表では、作成する必要のある新規 bean に関する一般情報をいくつか示しています。 bean の詳細については、どのメソッドをオーバーライドするのかも含めて、 167 ページの 『使用条件のための新規 CMP エンタープライズ Bean の作成』 を参照してください。

表 8.

属性	値
Bean 名	RebateTC
継承元	TermCondition
パッケージ	com.ibm.commerce.contract.objects
Bean クラス	RebateTCBean
リモート・インターフェース	RebateTC
ホーム・インターフェース	RebateTCHome

RebateTC を組み込むように WebSphere Commerce アクセラレーターを更新する: 新規の使用条件を作成したら、 WebSphere Commerce アクセラレーターを更新して、その新規の使用条件を組み込む新規の契約の作成に使用できます。このツールの更新方法の詳細については、173 ページの 『新規の条件を使用するための WebSphere Commerce アクセラレーターの更新』 を参照してください。

ステップ 4: 新規契約の作成

“RebateTC” 使用条件を含み、 “5DollarRebate” ビジネス・ポリシーを参照する新規契約を作成する必要があります。 WebSphere Commerce アクセラレーターか XML のいずれかを使って、新規契約を作成することができます。新規契約の作成のためのこれらの方法については、WebSphere Commerce オンライン・ヘルプにそれぞれ説明があります。

以下の表は、契約が作成された後の、 TERMCOND および POLICYTC データベース・テーブルの関係のある列の更新を示しています。

表 9. TERMCOND テーブルになされる更新

	列名		
	TRADING _ID	TERMCOND _ID	TCSUBTYPE _ID
サンプル・データ	25	901	RebateTC

表 10. POLICYTC テーブルになされる更新

	列名	
	POLICY _ID	TERMCOND_ID
サンプル・データ	301	901

ステップ 5: ショッピング・フローへの新規ビジネス・ポリシーの統合

このシナリオでは、顧客がログオンし、払い戻しを請求できるよう、ストアに新しいページを追加することを想定します。顧客が払い戻しを請求するためにクリックすると、新しい `RebatePolicyCmd` インターフェースを呼び出すコマンドが起動されるようにします。たとえば、`RebatePolicyCmd` を呼び出す新しい ClaimRebateCmd コントローラー・コマンドが考えられます。次いで、正しいビジネス・ポリシーが検索され、 (このケースでは) “5DollarRebate” ビジネス・ポリシーが適用されます。

第 3 部 開発環境

第 8 章 開発ツールおよびデプロイメント

この章では、 WebSphere Commerce アプリケーションをカスタマイズするための主な開発ツールについて紹介します。また、カスタマイズ・コードの VisualAge for Java から WebSphere Commerce Server へのデプロイメントを実行する手順も説明します。さらに、JSP テンプレート開発時にカスタマイズ・コードを利用するために、コードの Commerce Studio へのデプロイメントを実行する方法についても説明します。

開発環境

WebSphere Commerce Business Edition で使用するカスタマイズ・コードを作成するための開発パッケージには、 WebSphere Commerce Studio, Business Developer Edition 製品が推奨されています。 WebSphere Commerce Professional Edition で使用するカスタマイズ・コードを作成するための開発パッケージには、 WebSphere Commerce Studio Professional Developer Edition 製品が推奨されています。これらのパッケージには、いずれも、カスタマイズ・コードの作成や Web 開発作業に必要なツールがすべて揃っています。

WebSphere Commerce Studio Business Developer Edition と WebSphere Commerce Studio Professional Developer Edition には、どちらにも、 VisualAge for Java の WebSphere Test Environment コンポーネントで使用されるサンプル WebSphere Commerce ストアを組み込むオプションがあります。これにより、開発者は WebSphere Commerce のツールを使用してストアを作成してからストア資産を開発環境に移動する必要がなくなるため、開発環境の構成が単純化されます。

環境を開発するためのもう 1 つの主要コンポーネントは、WebSphere Commerce コードのリポジトリーです。このリポジトリーは、製品のインストールが完了してから、VisualAge for Java ワークスペースにインポートする必要があります。このリポジトリーのインポートに続いて、開発者は WebSphere Test Environment に関連する構成ステップを実行しなければなりません。

すべてのインストールおよび構成作業が完了すると、開発者は、カスタマイズした WebSphere Commerce コードを作成し、テストすることができるスタンドアロンの開発マシンを使用できるようになります。開発者は、開発用のマシンに WebSphere Commerce をインストールする必要はありません。

WebSphere Commerce Studio

WebSphere Commerce Studio Business Developer Edition と WebSphere Commerce Studio Professional Developer Edition には、いずれも VisualAge for Java Enterprise Edition バージョン 4.0 が組み込まれています。このエディションの VisualAge for Java には、ストアフロント資産の開発を援助する、拡張 JSP テンプレートの開発およびデバッグ用の強力なツールなどのフィーチャーが組み込まれています。さらに、WebSphere Studio との拡張統合によって、コンテンツをそれらの JSP テンプレートに短時間で追加して、プログラマーおよび Web 開発者の生産性を向上させることができます。事務アプリケーション・コードの開発について、Enterprise JavaBeans テクノロジーのサポート、および CICS® TS、 MQSeries,、SAP R/3 などの他のシステムとの統合をサポートするコネクティビティー機能が組み込まれます。加えて、VisualAge for Java Enterprise Edition の WebSphere Test Environment との統合によって、開発者は VisualAge for Java を終了しなくても WebSphere Commerce 機能を実行することが可能になります。これは、カスタマイズした WebSphere Commerce コードを WebSphere Commerce Server にデプロイメントしなくてもテストできることを意味します。

注: VisualAge for Java Enterprise Edition バージョン 4.0 の WebSphere Test Environment コンポーネントは、 WebSphere Application Server V3.5.4 内でアプリケーションを実行します。しかし、WebSphere Commerce Business Edition と WebSphere Commerce Professional Edition はどちらも WebSphere Application Server V4.0 を使用するので注意が必要です。このバージョンの違いにより、新しいエンタープライズ Bean や変更したエンタープライズ Bean を WebSphere Application Server V4.0 で稼働する WebSphere Commerce インスタンスにデプロイメントするときには、WebSphere Application Server V4.0 に同梱されている EJBDeploy ツールを使用して、 VisualAge for Java の外部でデプロイメント・コードを生成することが必要になります。なお、デプロイメント・コードの生成は、WebSphere Application Server V4.0 がインストールされているマシンで行われなければなりません。詳しくは、194 ページの 『EJB デプロイメント・コードに関する情報』を参照してください。

205 ページの『第 4 部 チュートリアル』で説明されている参考例を実行するためには、 WebSphere Commerce Studio Business Developer Edition か WebSphere Commerce Studio Professional Developer Edition をインストールする必要があります。 Commerce Studio のインストールと VisualAge for Java の構成の詳細については、 *WebSphere Commerce Studio Business Developer Edition インストール・ガイド* または *WebSphere Commerce Studio Professional Developer Edition インストール・ガイド* を参照してください。

VisualAge for Java のフィーチャーと機能

このプログラマーズ・ガイド の目的は、 VisualAge for Java の使用方法を伝えることではありません。この製品については、大部分は、 WebSphere Commerce のための e-commerce アプリケーションを作成するプログラミング・タスクを完了する方法よりも、 VisualAge for Java でのタスクの実行方法について示しています。 VisualAge for Java の新規ユーザーは、本書に記載されているチュートリアルを実行して、 VisualAge for Java でのタスクの実行方法の確認から始めます。ただし、このツールの使用に熟練するためには、VisualAge for Java の資料、チュートリアルおよび研修を参照してください。

たとえば、(オプション) VisualAge for Java でのデバッガーの使用チュートリアル・トピックでは、コードの実行中にデバッガーを使用してタスク・コマンド内の変数の値を確認する方法について簡単に紹介します。 VisualAge for Java のデバッガー・コンポーネントはデバッグのための堅固なツールです。このフィーチャーと使用法についての完全な詳細情報は、VisualAge for Java 資料を参照してください。

WebSphere Commerce コード・リポジトリー

WebSphere Commerce アプリケーションのカスタマイズ・コードを作成するためには、WebSphere Commerce コードのリポジトリーを VisualAge for Java ワークスペースにインポートする必要があります。リポジトリーは、WebSphere Commerce Business Edition Disk 2 CD と WebSphere Commerce Professional Edition Disk 2 CD から入手できます。現在のリポジトリー (本書の出版時点でのリポジトリー) は、 `WC_54.dat` です。

コードのデプロイメント

e-commerce アプリケーションをカスタマイズする場合は、以下のいずれかを実行することができます。

- 新しいコマンド、データ Bean、またはエンティティー Bean を作成する
- 既存の WebSphere Commerce エンティティー Bean を拡張する
- 既存のコマンドまたはデータ Bean のロジックを変更する

VisualAge for Java でコードを開発する場合、 WebSphere Test Environment でコードをテストすることができます。その後、適切な時点で、開発環境の外にある WebSphere Commerce Server にコードのデプロイメントを実行する必要があります。

以下のセクションでは、ターゲットの *WebSphere Commerce Server* は、カスタマイズ・コードのデプロイメントの対象となる WebSphere Commerce Server を指します。テスト・シナリオによっては、VisualAge for Java と同じマシンで実行している WebSphere Commerce Server にデプロイメントを実行することができます。別の状況として、ターゲットの WebSphere Commerce Server は別のマシン上にあり、異なるプラットフォーム上で稼働している場合さえあります。

以降のいくつかのセクションでは、さまざまなタイプのカスタマイズ・コードのデプロイメントを実行する、ハイレベルなステップを説明しています。それらの説明を、デプロイメント・プロセスに関係したステップの理解に役立ててください。段階的な手順については、351 ページの『付録 B. デプロイメントに関する詳細情報』を参照してください。

カスタマイズされたコードのデプロイメントについて説明する以下のセクションに加えて、ユーザーの開発環境で新規のアクセス制御ポリシーを作成してある場合は、同じアクセス制御ポリシーがターゲットの WebSphere Commerce Server 上に作成されなければなりません。この手順の例については、チュートリアルの 284 ページの 『新規リソースのためのアクセス制御ポリシーのロード』を参照してください。

EJB デプロイメント・コードに関する情報

VisualAge for Java V4.0 の WebSphere Test Environment コンポーネントは WebSphere Application Server V3.5.4 を使用することを知っておく必要があります。このバージョンの WebSphere Application Server では、エンタープライズ Bean が Enterprise JavaBeans (EJB) V1.0 の仕様のレベルでサポートされます。そのため、WebSphere Test Environment 内で任意のエンタープライズ Bean を実行するためには、VisualAge for Java のツールを使用して、 EJB V1.0 の仕様に準拠したエンタープライズ Bean のデプロイメント・コードを生成します。

これに対して、WebSphere Commerce Business Edition と WebSphere Commerce Professional Edition ではどちらも WebSphere Application Server V4.0 が使用されます。WebSphere Application Server V4.0 は、EJB V1.1 の仕様をサポートしています。ですから、WebSphere Test Environment 内で稼働するエンタープライズ Bean のデプロイメント・コードと、 WebSphere Application Server V4.0 内で稼働するエンタープライズ Bean のデプロイメント・コードは異なることになります。

これは、開発やデプロイメントに次のような影響を及ぼします。

- WebSphere Test Environment 内でエンタープライズ Bean を実行するには、VisualAge for Java のツールを使用して、WebSphere Test Environment 内で使用されている WebSphere Application Server V3.5.4 の要件を満たすデプロイメント・コードを生成します。このコードを生成するには、エンタープライズ Bean を右クリックして、**「デプロイメント・コードの生成」**を選択します。これによって JAR ファイルは作成されないことに注意してください。
- WebSphere Application Server V4.0 にエンタープライズ Bean をデプロイメントするには、以下を行う必要があります。
  1. VisualAge for Java のツールを使用して、エンタープライズ Beans を、本書で *EJB 1.1 Export JAR ファイル* と呼んでいるものにエクスポートするこの JAR ファイルは、コードを EJBDeploy ツールで使用されるフォーマットにパッケージします。このファイルを作成するには、デプロイするエンタープライズ Bean が含まれる EJB グループを右クリックし、**「エクスポート」** >**「 EJB 1.1 JAR」**を

選択します。この JAR ファイルを作成するときには、コードをデプロイメントするマシンに合ったデータベース・タイプを選択する必要があることに注意してください。

2. この EJB 1.1 Export JAR ファイルを WebSphere Application Server V4.0 が稼働するターゲット・サーバーに転送する。
3. ターゲット・サーバーで EJBDeploy ツールを実行し、この EJB 1.1 Export JAR ファイルを、 Enterprise JavaBeans V1.1 の仕様に準拠したデプロイメント・コードの新規 JAR ファイルを作成するために入力として用いる。

エンタープライズ Bean のデプロイメントに関係したその他のステップもあります。詳細については、196 ページの 『新規エンティティー Bean のデプロイメント』および 198 ページの 『変更された WebSphere Commerce パブリック・エンティティー Bean のデプロイメント』 を参照してください。 EJBDeploy ツールの使用についての詳細は、363 ページの 『デプロイメント・コードの生成』を参照してください。

新規コマンドとデータ Bean のデプロイメント

新しいコマンドとデータ Bean を作成するとき、ユーザーのアプリケーション用として適切な名前の付いたパッケージにそれらを入れる必要があります。たとえば、`com.mycompany.mycommands` という名前の新しいパッケージを作成して、この中に新規コマンドを保管することができます。このパッケージは、WebSphere Commerce プロジェクト (IBM WC Commerce Server および IBM WC エンタープライズ Beans) とは別のプロジェクト内に保管する必要があります。たとえば、`My Project` という名前の新規プロジェクトを作成することができるでしょう。デプロイメントを成功させるには、コードを注意深く編成することが必要です。

新しいコマンドやデータ Bean を独自のプロジェクトに保管したら、おおよそ以下のようなステップに従ってデプロイメントを行います。

1. 開発マシンで、プロジェクトの JAR ファイルを作成します。 VisualAge for Java の「プロジェクト」ページから、プロジェクトを JAR ファイルにエクスポートするためのツールを使用します。 JAR ファイルの名前は、コードに応じた適切な名前 (たとえば `CustomCommands.jar`) にします。さらに、VisualAge for Java の外側にあるツールを使用して JAR ファイルを rejar して、 WebSphere Application Server へのデプロイメントについて、必要な命名情報がすべて組み込まれることを確認しなければなりません。詳しくは、351 ページの『カスタマイズされたコマンドおよびデータ Bean の JAR ファイル』を参照してください。
2. JAR ファイル、JSP テンプレート、その他のストア資産を、ターゲットの WebSphere Commerce Server の適切なディレクトリーにコピーします。詳しくは、358 ページの『ターゲットの WebSphere Commerce Server への資産の保管』を参照してください。
3. ターゲット WebSphere Commerce Server上のコマンド・レジストリーに新しいコマンドを登録します。詳しくは、28 ページの『コマンド登録フレームワーク』を参照してください。

4. WebSphere Application Server 管理コンソールを使用して、WebSphere Commerce エンタープライズ・アプリケーションを停止および再始動します。このアプリケーションの始動と停止の詳細については、該当する *WebSphere Commerce インストール・ガイド* を参照してください。

新規エンティティー Bean のデプロイメント

新規エンティティー Bean は、 WebSphere Commerce エンティティー Bean が含まれている EJB グループとは別の EJB グループに作成する必要があります。これには独自のプロジェクトを使用し、このプロジェクトに、何らかのコマンドやデータ Bean 用のコードが含まれることがないようにしてください。たとえば、`MyEntityBeans` という新規 EJB グループと、 `MyEntityBeansProject` という名前のプロジェクトを作成することができます。プロジェクトに新規エンティティー Bean のコード以外のコードが含まれている場合、デプロイメントは成功しない場合があります。

エンティティー Bean が WebSphere Test Environment で適切に機能することがわかったら、そのデプロイメントを実行する必要があります。デプロイメント・ステップの概要を以下に示します。

1. 開発マシンで、新しい EJB グループ用の新しい EJB 1.1 Export JAR ファイルを作成します。 VisualAge for Java の「EJB」ページで、その新しい EJB グループを右クリックして、 EJB 1.1 JAR ファイルにコードをエクスポートすることを選択します。ファイルには、コードに応じた適切な名前 (たとえば、`MyEntityBeans_DT.jar`) を付けてください。詳しくは、 353 ページの『新しいエンティティー Bean 用の JAR ファイルの作成』を参照してください。
2. 新しい EJB プロジェクトのインプリメンテーション JAR ファイルを新規作成します。この JAR ファイルを作成するには、「プロジェクト」ページを選択し、新しい EJB プロジェクトを右クリックして、コードを JAR ファイルにエクスポートするよう選択します。ファイルには、コードに応じた適切な名前 (たとえば、`MyEntityBeansImpl.jar`) を付けてください。さらに、 VisualAge for Java の外側にあるツールを使用してインプリメンテーション JAR ファイルを rejar して、WebSphere Application Server へのデプロイメントについて、必要な命名情報がすべて組み込まれることを確認しなければなりません。詳しくは、 353 ページの『新しいエンティティー Bean 用の JAR ファイルの作成』を参照してください。
3. JAR ファイルをターゲット上の WebSphere Commerce Server の適切なディレクトリーにコピーします。詳しくは、 358 ページの『ターゲットの WebSphere Commerce Server への資産の保管』を参照してください。
4. ターゲット WebSphere Commerce Server上で、WebSphere Application Server が提供している EJB デプロイメント・ツールを使用して、新規エンタープライズ Bean のデプロイメント・コードを生成します。このツールは、ステップ 1 で作成された JAR ファイルを入力として用い、 EJB グループ内のすべてのエンタープライズ Bean 用のデプロイメント・コードを含む、対応する JAR ファイルを作成します。詳しくは、 363 ページの『デプロイメント・コードの生成』を参照してください。

5. ターゲットの WebSphere Commerce Server 上で、デプロイメント・コードの JAR ファイルに含まれているエンタープライズ Bean のトランザクション分離レベルを変更します。 WebSphere Commerce の `modifyIsolationLevel` コマンド行ユーティリティーを使用して、データベースのタイプに合ったトランザクション分離レベルを設定してください。詳しくは、 366 ページの『エンティティー Bean のトランザクション分離レベルの変更』を参照してください。
6. ターゲットの WebSphere Commerce Server 上で、作成したすべての新規リソースのアクセス制御ポリシーをロードします。ポリシー情報のロードには、WebSphere Commerce の `acpload` コマンドと `acpnlsload` コマンドを使用します。新規リソースのアクセス制御ポリシーのロードの例については、 『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』チュートリアルで、 284 ページの『新規リソースのためのアクセス制御ポリシーのロード』のセクションを参照してください。
7. ターゲットの WebSphere Commerce Server 上で、WebSphere Application Server から現在の WebSphere Commerce エンタープライズ・アプリケーションをエクスポートします。エクスポートが完了すると、アプリケーション全体を含む .ear ファイルが作成されます。現在のアプリケーションをエクスポートするには、WebSphere Application Server 管理コンソールをオープンし、 WebSphere Commerce エンタープライズ・アプリケーションを選択して、そのエクスポート・オプションを選択します。エクスポートが完了すると、 `WC_Enterprise_App_`*`instanceName`*`.ear` ファイルが作成されます。詳しくは、 367 ページの『現在の WebSphere Commerce エンタープライズ・アプリケーションのエクスポート』を参照してください。
8. ターゲットの WebSphere Commerce Server 上で、 WebSphere Application Server 内で稼働する現在の WebSphere Commerce エンタープライズ・アプリケーションに含まれているエンタープライズ Bean の構成情報をエクスポートします。この情報は、WebSphere Application Server が提供している `XMLConfig` コマンド行ユーティリティーの `-export` オプションを使用して XML ファイルにエクスポートされます。生成される XML ファイル (ここでは `OutputFile.xml` ファイルとしておきます) には、エンタープライズ・アプリケーション内の各エンタープライズ Bean に関する構成情報のスタンザが含まれています。加えて、デプロイメントする新しいエンタープライズ Bean ごとに、それぞれこのファイルに新しいスタンザを追加する必要があります。詳しくは、 368 ページの『エンタープライズ Bean の構成情報のエクスポート』を参照してください。
9. WebSphere Application Server のアプリケーション・アセンブリー・ツールを使用して、エンタープライズ・アプリケーションに 1 つ以上の新しいエンタープライズ Bean を追加します。このツールでは、現在のアプリケーションの `WC_Enterprise_App_`*`instanceName`*`.ear` をオープンし、アプリケーションに新しいエンタープライズ Bean をインポートすることができます。さらに、新しいエンタープライズ Bean のクラスパスを設定して任意の従属 JAR ファイルを組み込み、インプリメンテーション JAR ファイルを 1 つのファイルとしてアプリケーションに追加します。最後に、このツールを使用して、エンタープライズ Bean に含まれるメソッドの WebSphere Application Server セキュリティーを構成します。これらのステップが完了したら、アプリケーションと、エンタープライズ・アプリケーション用に作成

した新しい .ear ファイルを保管します。詳しくは、 374 ページの『新しいエンタープライズ Bean のエンタープライズ・アプリケーションへのアセンブル』を参照してください。

10. 新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートします。このステップは、以下の 3 つのサブタスクから成っています。

    a. WebSphere Application Server 管理コンソールを使用して、オリジナルの WebSphere Commerce エンタープライズ・アプリケーションを停止し、除去します。詳しくは、 384 ページの『エンタープライズ・アプリケーションの停止と除去』を参照してください。

    b. XMLConfig コマンド行ユーティリティーの -import オプションを使用して、新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートします。詳しくは、 385 ページの『エンタープライズ・アプリケーションのインポート』を参照してください。

    c. WebSphere Application Server 管理コンソールを使用して、ビューを最新表示し、新しいエンタープライズ・アプリケーションを表示させてから新しいアプリケーションを開始します。詳しくは、 387 ページの『エンタープライズ・アプリケーションの開始』を参照してください。

既存のコマンドおよびデータ Bean からの拡張のデプロイメント

既存のコマンドを拡張するメソッドは、必要な変更のタイプに応じて異なります。拡張の方法は、 145 ページの『既存のコマンドのカスタマイズ』で説明されています。通常、既存のロジックを変更するには、カスタマイズ対象のクラスから継承する新しいクラスを作成します。スーパークラスのオーバーライド・メソッドは、必要に応じてロジックを置換または変更します。

データ Bean をカスタマイズする場合にも、既存のデータ Bean を拡張する新規クラスを作成します。新規クラスの中で、必要な変更を行います。

このような新規クラスを作成するとき、必ずユーザー独自のパッケージに保管してください。また、そのようなパッケージ自体も、ユーザー独自のプロジェクトに保管します。

サブクラス化によって拡張を効率的に行うことができるため、コマンドおよびデータ Bean の拡張のデプロイメントは、新規コマンドおよびデータ Bean のデプロイメントと同様です。詳しくは、 195 ページの『新規コマンドとデータ Bean のデプロイメント』を参照してください。

変更された WebSphere Commerce パブリック・エンティティー Bean のデプロイメント

WebSphere Commerce パブリック・エンタープライズ Bean を変更する場合は、WebSphere Commerce コードを変更します。このため、カスタマイズされたエンティテ

ィー Bean のデプロイメントを実行する方法は、新規エンティティー Bean のデプロイメントを実行する場合と少し異なります。デプロイメント・ステップの概要を以下に示します。

1. 変更されたエンティティー Bean が属している WebSphere Commerce EJB グループの EJB 1.1 Export JAR ファイルを作成します。 VisualAge for Java ワークベンチで「EJB」タブを選択して、適当な EJB グループを選択し、次いで EJB 1.1 Export JAR ファイルへのそのグループのエクスポートを選択します。 JAR ファイルの名前を付ける際には、 Cust_*EJBGroupName*-ejb_DT.jar 命名規則を使用します。ここで、*EJBGroupName* は変更された EJB グループの名前であり、 JAR ファイルの名前の末尾には _DT 接尾部を付けます。たとえば、WCSUser EJB グループの JAR ファイルに名前を付ける場合は、 Cust_WCSUser-ejb_DT.jar のような名前を使用できます。なお、この _DT 接尾部は、 EJB グループ内にある bean のデプロイメント・コードを生成するために、後でこの JAR ファイルを EJBDeploy ツールに渡す必要があることを思い出させるためだけのものですので、ご注意ください。詳しくは、356 ページの『カスタマイズされた WebSphere Commerce エンティティー Bean 用の JAR ファイルの作成』を参照してください。
2. すべての WebSphere Commerce EJB グループのクライアント・コードを含む新しいクライアント JAR ファイルを作成します。すべての WebSphere Commerce EJB グループ (WCS で始まる名前すべて) を選択した状態で、クライアント JAR ファイルへのエクスポートを選択します。詳しくは、356 ページの『カスタマイズされた WebSphere Commerce エンティティー Bean 用の JAR ファイルの作成』を参照してください。
3. JAR ファイルをターゲット上の WebSphere Commerce Server の適切なディレクトリーにコピーします。詳しくは、358 ページの『ターゲットの WebSphere Commerce Server への資産の保管』を参照してください。
4. ターゲットの WebSphere Commerce Server 上で、WebSphere Application Server が提供している EJBDeploy ツールを使用して、デプロイメントしている EJB グループに含まれているエンタープライズ Bean のデプロイメント・コードを生成します。このツールは、ステップ 1 で作成された JAR ファイルを入力として用い、 EJB グループ内のすべてのエンタープライズ Bean のためのデプロイメント・コードを含む、対応する JAR ファイルを作成します。詳しくは、363 ページの『デプロイメント・コードの生成』を参照してください。
5. ターゲットの WebSphere Commerce Server 上で、デプロイメント・コードの JAR ファイルに含まれているエンタープライズ Bean のトランザクション分離レベルを変更します。 WebSphere Commerce の modifyIsolationLevel コマンド行ユーティリティーを使用して、データベースのタイプに合ったトランザクション分離レベルを設定してください。詳しくは、366 ページの『エンティティー Bean のトランザクション分離レベルの変更』を参照してください。
6. ターゲットの WebSphere Commerce Server 上で、WebSphere Application Server から現在の WebSphere Commerce エンタープライズ・アプリケーションをエクスポートします。エクスポートが完了すると、アプリケーション全体を含む .ear ファイル

が作成されます。現在のアプリケーションをエクスポートするには、WebSphere Application Server 管理コンソールをオープンし、 WebSphere Commerce エンタープライズ・アプリケーションを選択して、そのエクスポート・オプションを選択します。エクスポートが完了すると、 WC_Enterprise_App_*instanceName*.ear ファイルが作成されます。詳しくは、 367 ページの『現在の WebSphere Commerce エンタープライズ・アプリケーションのエクスポート』を参照してください。

7. ターゲットの WebSphere Commerce Server 上で、 WebSphere Application Server 内で稼働する現在の WebSphere Commerce エンタープライズ・アプリケーションに含まれているエンタープライズ Bean の構成情報をエクスポートします。この情報は、WebSphere Application Server が提供している XMLConfig コマンド行ユーティリティーの -export オプションを使用して XML ファイルにエクスポートされます。生成される XML ファイル (ここでは OutputFile.xml ファイルとしておきます) には、エンタープライズ・アプリケーション内の各エンタープライズ Bean に関する構成情報のスタンザが含まれています。詳しくは、 368 ページの『エンタープライズ Bean の構成情報のエクスポート』を参照してください。このファイル内で、新しい Cust_*EJBGroupName*-ejb.jar ファイルを指すように変更されたエンタープライズ Bean について記述するスタンザを変更する必要があります。
8. ターゲットの WebSphere Commerce Server 上で、アプリケーション・アセンブリー・ツールを使用して、変更された EJB グループをエンタープライズ・アプリケーションに統合します。このツールで、現在のアプリケーションの .ear ファイルをオープンします。ファイルをオープンしたら、以下のタスクを実行してください。
   a. 変更された EJB グループのオリジナルのクラスパス情報をメモしておく。たとえば、WCSUser EJB グループの User bean を変更した場合は、 WCSUser グループのクラスパス情報をテキスト・ファイルにコピーしておきます。
   b. 変更前のオリジナルの EJB グループ (たとえば、WCSUser グループ) を削除する。
   c. 変更された新しいバージョンの EJB グループをインポートする。
   d. インポートしたばかりの EJB グループにオリジナルのクラスパス情報を適用する。
   e. 変更された EJB グループのすべてのエンタープライズ Bean に含まれているメソッドの WebSphere Application Server セキュリティーを構成する。
   f. アプリケーションを新しい .ear に保管する。

   詳しくは、 380 ページの『変更されたエンタープライズ Bean のエンタープライズ・アプリケーションへのアセンブル』を参照してください。
9. 新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートします。このステップは、以下の 3 つのサブタスクから成っています。
   a. WebSphere Application Server 管理コンソールを使用して、オリジナルの WebSphere Commerce エンタープライズ・アプリケーションを停止し、除去します。詳しくは、 384 ページの『エンタープライズ・アプリケーションの停止と除去』を参照してください。

b. XMLConfig コマンド行ユーティリティーの -import オプションを使用して、新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートします。詳しくは、 385 ページの『エンタープライズ・アプリケーションのインポート』を参照してください。

c. WebSphere Application Server 管理コンソールを使用して、ビューを最新表示し、新しいエンタープライズ・アプリケーションを表示させてから新しいアプリケーションを開始します。詳しくは、 387 ページの『エンタープライズ・アプリケーションの開始』を参照してください。

Commerce Studio で使用するための新規データ Bean のデプロイメント

JSP テンプレートの開発に Commerce Studio を使用している場合、新規データ Bean の Commerce Studio へのデプロイメントを実行する必要があります。つまり、データ Bean の JAR ファイルを作成しなければなりません。

この JAR ファイルを作成するには、データ Bean のパッケージを右クリックして、それを JAR ファイルにエクスポートするよう選択します。オプションで、以下のものを含めるように選択します。

- **クラス**
- **リソース**
- **bean**

さらに、データ Bean が必要とするコマンドとリソースを含めるために、**「参照タイプとリソースの選択」**を選択してください。

JAR ファイルをエクスポートしたら、Commerce Studio のクラスパスを更新して、この JAR を含むようにしなければなりません。

既存の WebSphere Commerce データ Bean を変更するとき、カスタマイズ対象のデータ Bean から拡張する新しいデータ Bean を作成しなければなりません。したがって、実質的には新規データ Bean を作成しているのと同じですから、このセクションで説明した方法に従ってデプロイメントを実行してください。

Commerce Studio で使用するためのパブリック・エンティティー Bean のデプロイメント

パブリック・エンティティー Bean を変更して、Commerce Studio を JSP テンプレートの開発に使用する場合、すべてのパブリック・エンティティー Bean 用に新規クライアント JAR ファイルを作成して、 Commerce Studio のクラスパスを変更し、新規 JAR ファイルの名前を元の JAR ファイルの名前の前に組み込む必要があります。 Page Designer ツールでデータ Bean が使われるとき、 Commerce Studio はクライアント JAR ファイルを使用します。

この JAR ファイルを作成するには VisualAge for Java のツールを使用します。VisualAge for Java の「EJB」ページから、 (変更したグループだけではなく) すべての WebSphere Commerce EJB グループを選択し、これらをクライアント JAR ファイルにエクスポートするように選択します。エクスポートが完了したら、 Commerce Studio のクラスパスの先頭にこの新規 JAR ファイルを必ず指定してください。

ログ・ファイル

製品のインストール、コード開発、およびコード・デプロイメントのステージの至るところでログ・ファイルが生成されます。このセクションでは、これらのステージの至るところで調べる可能性のあるログ・ファイルをいくつかリストしています。

Commerce Studio ログ・ファイル

Commerce Studio はインストール・プロセス時にログ・ファイルを作成します。これらのログにアクセスするには、以下のファイルをオープンしてください。

```
C:¥Winnt¥WCStudioInstall.log
```

WebSphere Test Environment についての Commerce Studio 構成のログ

バックエンド開発タスクを行うように選択する場合、 WebSphere Test Environment の構成ステップのいくつかが、 WebSphere Commerce Studio のインストール中に行われます。たとえば、 VisualAge for Java の WebSphere Test Environment コンポーネント内で実行するサンプル・ストアを組み込むように選択する場合は、大量のロード・プロセスおよびデータベース作成に関連するログ・ファイルが作成されます。これらのログにアクセスするには、以下のディレクトリーに移動してください。

drive:¥WebSphere¥CommerceServerDev¥instances¥*instance_name*¥logs

WebSphere Test Environment 内で実行中の WebSphere Commerce コンポーネントからのログ・ファイル

WebSphere Test Environment 内でストアを実行しているか、個々のコンポーネントをテストしているときに、トレース・ファイルとメッセージ・ファイルが生成されることがあります。これらのファイルのデフォルトの位置は、以下のディレクトリーにあります。

drive:¥WebSphere¥CommerceServerDev¥instances¥*instance_name*¥logs

実行中の modifyIsolationLevel コマンドからのログ

エンタープライズ Bean のデプロイメント時には、modifyIsolationLevel コマンドを使用して、 JAR ファイル内の各エンタープライズ Bean のトランザクション分離レベルが変更されます。生成されたログ・ファイルは、コマンドの実行中に指定するログ・ファイルに保管されます。詳しくは、366 ページの 『エンティティー Bean のトランザクション分離レベルの変更』を参照してください。

VisualAge for Java 内でのロギング

WebSphere Test Environment 内でのコードの実行中に、「コンソール」ウィンドウがアクティブ・ログとして実行します。さらに、EJB サーバーのトレースのレベルを変更して、「コンソール」ウィンドウ内で検索できる情報の量を増やすことができます。この情報は、 EJB サーバーのプロパティー内で、トレース・レベルとして指定されます。

テスト支払いメソッド

WebSphere Test Environment 内で実行している InFashion サンプル・ストアは、デフォルトでテスト支払いメソッドを使用します。このテスト支払いメソッドが組み込まれているため、ユーザーはリモートの Payment Manager の呼び出しを必要とせずに WebSphere Test Environment 内でのショッピング・フローを完了できます。この支払いメソッドを使用して発行されるオーダーは、 'M' という状況 (支払いが開始され、処理を待っていることを示す) になります。

このテスト支払いメソッドは、1 回の購入を完了させるだけであり、この支払いメソッドで送信されたオーダーは今後の処理で使用可能にすることはできません。したがって、テスト支払いメソッドは WebSphere Test Environment 内でのみ使用してください。

支払い関連コマンドについてのインプリメンテーション・クラスはビジネス・ポリシー・コマンドのタイプです。つまり、使用するインプリメンテーション・クラスの選択は、ビジネス・ポリシーによって決まります。デフォルトでは、WebSphere Test Environment で実行している InFashion サンプル・ストアには、以下の支払い関連コマンドのインプリメンテーションを使用するビジネス・ポリシー (TestPaymentMethod policy) が組み込まれています。

- com.ibm.commerce.payment.commands.DoPayment**Test**CmdImpl
- com.ibm.commerce.payment.commands.CheckPaymentAccept**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoCancel**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoDeposit**Test**CmdImpl
- com.ibm.commerce.payment.commands.DoRefund**Test**CmdImpl

これらのコマンドは、チェックアウト・プロセスをテストするためだけに提供されていることを強調しておきます。上記のコマンドの組は完成の目的で提供されていますが、DoDepositTestCmdImpl は呼び出されると例外をスローするコマンド・スタブです。このような順序でオーダー管理機能を使用しないでください。そのような他の機能をテストする機能が必要な場合は、 WebSphere Test Environment を構成してリモートの Payment Manager を使用できます。

ターゲットの WebSphere Commerce Server にコードをデプロイメントするときには、絶対にユーザーの開発マシンからターゲット・マシンにテスト支払いメソッドに関連し

たビジネス・ポリシーをコピーしないでください。さらに、ターゲットの WebSphere Commerce Server 上でテスト支払いメソッドに関連したコマンドを登録しないでください。

テスト支払いメソッドを使用不可にすることの詳細については、 *WebSphere Commerce Business Edition インストール・ガイド* または *WebSphere Commerce Professional Edition インストール・ガイド* の 『VisualAge for Java の構成』の章を参照してください。

リモートの Payment Manager の使用

ユーザーの開発作業に、たとえばオーダー管理プロセスでのステップに修正を加えるなど、一度発行されたオーダーの処理が含まれる場合は、リモートの Payment Manager を使用するように、WebSphere Test Environment で実行するストアを構成しなければなりません。構成ステップの詳細については、*WebSphere Commerce Studio, Business Developer Edition インストール・ガイド* を参照してください。この資料には、ユーザーのデータベースからテスト支払いメソッドを使用して発行されたオーダーの除去についての情報も記載されています。

第 4 部 チュートリアル

このセクションでは、e-commerce アプリケーションのカスタマイズに精通する助けとなるチュートリアルを記載します。以下のチュートリアルが提供されます。

- 新しいビジネス・ロジックの作成。
  このチュートリアルは新しいビジネス・ロジックを作成するための開発過程を示します。これには以下のサブタスクが含まれています。
  - サンプル・プロジェクトの準備
  - コマンドの新規記述
  - データ Bean の新規作成と、要求からのプロパティーの取得
  - エンティティー Bean の使用
  - エンタープライズ Bean の新規作成
- 既存のビジネス・ロジックの更新。
  このチュートリアルでは、既存の WebSphere Commerce ビジネス・ロジックを更新するための開発過程を示します。これには以下のサブタスクが含まれています。
  - 既存の WebSphere Commerce コントローラー・コマンドの拡張
  - 既存の WebSphere Commerce パブリック・エンティティー Bean の変更
  - 既存の WebSphere Commerce タスク・コマンドを拡張する新しいタスク・コマンドの作成

チュートリアルには、 VisualAge for Java に入力する必要のあるコードのセクションが含まれています。以下の Web サイトから、この資料の PDF 版を入手することをお勧めします。

Business

```
http://www.ibm.com/software/webservers/commerce/wc_be
   /lit-tech-general.html
```

Professional

```
http://www.ibm.com/software/webservers/commerce/wc_pe
   /lit-tech-general.html
```

PDF 版のプログラマーズ・ガイドから、コードの断片を切り貼りすることができます。

第 9 章 チュートリアル: ビジネス・ロジックの新規作成

チュートリアル環境

本書に含まれているチュートリアルでは、 WebSphere Commerce Business Edition V5.4 か WebSphere Commerce Professional Edition V5.4 のどちらかをインストール済みである必要があります。さらに、製品をインストールする際は、以下のオプションを選択しなければなりません。

- **WebSphere Studio を使用してストア・フロント資産を開発する**
- **VisualAge for Java を使用してストアのバックエンド・ロジックを開発する**
- **データベースを作成する**
- **サンプル・ストアを組み込む**

インストールおよび構成の完全な情報については、*WebSphere Commerce Studio, Business Developer Edition インストール・ガイド* または *WebSphere Commerce Studio Professional Developer Edition インストール・ガイド* に示されている指示に従ってください。

このチュートリアルを開始する前に、WebSphere Test Environment 内でサンプル・ストアを実行し、ストアで購入の操作ができるようにしておく必要があります。

チュートリアル・コードのデプロイメント・ステップ

本書に含まれているチュートリアルには、ターゲットの WebSphere Commerce Server にカスタマイズされたコードをデプロイメントする方法を説明する、オプションのステップがあります。ターゲットの WebSphere Commerce Server は、ユーザーの開発環境とは別のマシン上の Windows NT または Windows 2000 で実行しているとします。開発環境として WebSphere Commerce Studio Business Developer Edition を使用する場合は、WebSphere Commerce Business Edition にデプロイしなければならないことに注意してください。開発環境として WebSphere Commerce Studio Professional Developer Edition を使用する場合は、WebSphere Commerce Professional Edition にデプロイしなければならないことに注意してください。

さらに、ターゲットの WebSphere Commerce Server 上で、InFashion サンプル・ストアを基にしたストアを作成しておくことも必要です。そのストア内で、購入の操作を完了できなければなりません。支払い処理には、リモートまたはローカルいずれかの Payment Manager を使用できます。

開発環境と同じマシン上にある、または別のオペレーティング・システムで実行されているターゲットの WebSphere Commerce Server にデプロイメントを実行する場合、

191 ページの 『第 8 章 開発ツールおよびデプロイメント』および 351 ページの 『付録 B. デプロイメントに関する詳細情報』で、適切なデプロイメント情報を参照してください。

サンプル・プロジェクトの準備

このセクションでは、「ビジネス・ロジックの新規作成」チュートリアルの開始点を含む WC_SAMPLE_54.dat リポジトリーをインポートします。

環境を準備するため、以下のようにします。

1. WebSphere Commerce コードの WC_54.dat リポジトリーを使用していることを確認してください。これは、 WebSphere Commerce Business Edition V5.4 Disk 2 CD または WebSphere Commerce Professional Edition V5.4 Disk 2 のどちらかの CD に含まれています。
2. 以下のようにして、サンプル・プロジェクトをワークスペースにインポートします。
   a. 以下の CD をユーザーの開発マシンの CD ドライブに挿入します。
      - Business WebSphere Commerce Business Edition V5.4 Disk 2
      - Professional WebSphere Commerce Professional Edition V5.4 Disk 2
   b. VisualAge for Java をオープンします。
   c. **「ファイル」**メニューから、**「インポート」**を選択します。
      「Import SmartGuide (インポート SmartGuide)」がオープンします。
   d. **「リポジトリー」**をインポートするよう選択して、**「次へ」**をクリックします。
   e. 別のリポジトリーからのインポート・ウィンドウで、以下のようにします。
      1) **「ローカル・リポジトリー」**を選択します。
      2) **「リポジトリー名」**フィールドで、
         CD_drive:¥repository¥samples¥programguide¥WC_SAMPLE_54.dat
         と入力します。 *CD_drive* は CD ドライブです。
      3) **「プロジェクト」**を選択して**「詳細情報」**をクリックします。
         「_WCSamples」プロジェクトを選択します。**「WC Sample 5.4」**バージョンを選択して、**「OK」**をクリックします。
      4) **「最新のプロジェクト・エディションをワークスペースに追加」**が選択されていることを確認します。
      5) **「終了」**をクリックしてインポートを開始します。
         プロジェクトのインポートには数分かかる場合があります。
3. 以下のようにして、ワークスペースの所有者が WCS 開発者に設定されていることを確認します。
   a. **「ワークスペース」**メニューから、**「ワークスペース所有者」**を選択します。
   b. **「WCS 開発者」**を選択して**「OK」**をクリックします。

4. 演習用 JSP テンプレートを WebSphere Test Environment で使用できるようにするために、適切なディレクトリーにコピーします。これらのファイルをコピーするには、以下のようにします。

   a. 次のディレクトリーの Sample.jsp ファイルと Sample_All.jsp ファイルを探します。

      CD_drive:¥repository¥samples¥programguide¥

      ここで、*CD_drive* はそのディレクトリーの CD ドライブです。

   b. これら 2 つのファイルを以下のディレクトリーにコピーします。

      vaj_drive:¥VAJava¥ide¥project_resources¥IBM WebSphere Test Environment ¥hosts¥default_host¥default_app¥web

      ここで、*vaj_drive* は VisualAge for Java のインストール先のドライブです。

5. アクセス制御ポリシーのファイルを適切なディレクトリーにコピーします。これらのファイルは、チュートリアル中にユーザーが作成する、新規リソースについての新規のアクセス制御ポリシーのロードに使用されます。これらのファイルをコピーするには、以下のようにします。

   a. 以下のディレクトリーに移動します。

      CD_drive:¥repository¥samples¥programguide¥

   b. そのディレクトリーで以下のファイルを見付けます。

      - SampleCmdACPolicy.xml
        この XML ファイルは、新規のコントローラー・コマンドを作成するときに使用されるアクセス制御ポリシーを含みます。
      - SampleACPolicy.xml
        この XML ファイルは、新規のエンタープライズ Bean を作成するときに使用されるアクセス制御ポリシーを含みます。
      - SampleACPolicy_*locale*.xml
        locale は言語 ID です。この XML ファイルには、アクセス制御ポリシーの説明があります。

   c. 以上のファイルを以下のディレクトリーにコピーします。
      drive:¥WebSphere¥CommerceServerDev¥xml¥policies¥xml
      drive は WebSphere Commerce Studio のインストール先のドライブです。

6. チュートリアルを開始する準備ができたことを確認するため、以下の手順で環境をテストします。

   a. Persistent Name Server を開始する (349ページを参照)。

   b. EJB サーバーを開始する (350ページを参照)。

   c. サーブレット・エンジンを開始する (350ページを参照)。

   d. ブラウザーをオープンして、以下の URL を入力する。

```
http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
   storeId=store_ID&catalogId=catalog_Id&langId=-1
```

ここで、*store_ID* はユーザーのサンプル・ストアの ID (10001 は例の値) で、*catalog_Id* はユーザーのサンプル・ストアのカタログの ID (10001 は例の値) です。
サンプル・ストアのホーム・ページが表示される場合は、商品を選択して購入してください。ショッピング・フローの「オーダーの確認」ページを表示できなければなりません。

ストアの storeId 値を検査するために、 STOREENT テーブルを参照できます。

e. ブラウザーをすべてクローズして、WebSphere Test Environment を停止します。

これでチュートリアルを開始する準備ができました。

コマンドの作成

コントローラー・コマンドの作成

このセクションでは、新しいコントローラー・コマンドの書き方を示します。この演習を完了すると、*MyNewControllerCmd* という新規コマンドが使えるようになります。このコマンドはすべてのストアで使用することができ、各ストアはコマンドの同じインプリメンテーションを使用します。

新規コントローラー・コマンドの作成には、基本的に以下の 3 つのステップがあります。

1. コマンドをコマンド登録フレームワークに登録する
2. コマンドのインターフェースを作成する
3. コマンドのインプリメンテーション・クラスを作成する

コマンドについての詳細は、 23 ページの『コマンド・デザイン・パターン』を参照してください。

このチュートリアルを始める前に

208 ページの 『サンプル・プロジェクトの準備』のステップを完了している必要があります。

新しいコマンドを記述するには、以下のようにします。

1. コントローラー・コマンドを書く上での最初のステップは、コマンド用の名前を確立することです。この例では、コマンドは *MyNewControllerCmd* と呼ばれます。
2. コマンドはコマンド登録フレームワークに登録する必要があります。新規コントローラー・コマンドの登録プロセスには、 URLREG テーブルにエントリーを作成することが含まれます。

   DB2 DB2 データベースを使用している場合は、以下のようにしてコマンドを登録してください。

   a. DB2 コマンド・センターをオープンします (**「スタート」** > **「プログラム」** > **「IBM DB2」** > **「コマンド・センター」**)。
   b. **「ツール」**メニューから、**「ツール設定」**を選択します。
   c. **「ステートメント終了文字の使用」**チェック・ボックスを選択し、セミコロン (**;**) が文字として指定されていることを確認します。
   d. スクリプト・ウィンドウの「スクリプト」タブを選択し、スクリプト・ウィンドウで以下の情報を入力することによって、 URLREG テーブルに必要なエントリーを作成します。

      ```
      connect to your_database_name;
      insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
          DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
          'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
          'This is a new controller command for test/education purposes.',
          null)
      ```

      ここで、*your_database_name* はデータベースの名前です。続いて「実行」アイコンをクリックします。
      このコマンドはすべてのマーチャントによって使われます (STOREENT_ID の 0 の値がそれを示しています)。

   Oracle Oracle データベースを使用している場合は、以下のようにしてコマンドを登録してください。

   a. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。
   b. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。
   c. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
   d. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
   e. 「SQL Plus」ウィンドウで、以下を入力して URLREG テーブルに必要なエントリーを作成します。

      ```
      insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
          DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
          'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
          'This is a new controller command for test/education purposes.',
          null);
      ```

それから Enter を押して SQL ステートメントを実行します。

f. データベースの変更をコミットするには

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

注: この演習では、単純化のために、新規コマンドにはインプリメンテーション・クラスが 1 つしかありません。したがって、すべてのストアは同じインプリメンテーション・クラスを使用します。このインプリメンテーション・クラスは、インターフェースのコードの中で直接指定されています。したがって、 CMDREG テーブルでインターフェースとインプリメンテーション・クラスの間のマッピングを登録する必要はありません。チュートリアル以外の環境では、コントローラー・コマンドを CMDREG テーブルおよび URLREG テーブルで定義する必要があります。

3. コントローラー・コマンドはビューを戻すはずです。新しく作成するコントローラー・コマンドは、SampleViewTask ビューを戻します。 SampleViewTask ビューを VIEWREG テーブル中に登録しなければなりません。

DB2 DB2 データベースを使用している場合は、以下のようにしてビューを登録してください。

a. スクリプト・ウィンドウで以下の情報を入力することによって、 VIEWREG テーブルにエントリーを作成します (最初にウィンドウから直前の SQL ステートメントを消去する必要がある場合があります)。

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('SampleViewTask',-1, 0,
   'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=Sample.jsp','This is a sample view for the
    Bonus Point exercise', 0, null)
```

続いて「実行」アイコンをクリックします。

Oracle Oracle データベースを使用している場合は、以下のようにしてデータベースにビューを登録してください。

a. 「SQL Plus」ウィンドウで、以下の SQL ステートメントを入力します。

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('SampleViewTask',-1, 0,
   'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=Sample.jsp','This is a sample view for the
   Bonus Point exercise', 0, null);
```

それから Enter を押して SQL ステートメントを実行します。

b. データベースの変更をコミットするには

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

4. VisualAge for Java ワークベンチ・ウィンドウで **_WCSamples** プロジェクトを拡張表示します。
5. **com.ibm.commerce.sample.commands** パッケージを拡張表示して **MyNewControllerCmd** インターフェースを右クリックし、**「追加」>「フィールド」**を選択します。
「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。
6. 以下のようにして、インターフェースのデフォルト・インプリメンテーション・クラスを指定するフィールドを作成します。
   a. **「フィールド名」**フィールドに、 `defaultCommandClassName` と入力します。
   b. **「フィールド・タイプ」**ドロップダウン・リストから、 `String` を選択します。
   c. 「初期値」フィールドで、
   `"com.ibm.commerce.sample.commands.MyNewControllerCmdImpl"` と入力します。
   (二重引用符を必ず含めてください。)
   d. **「終了」**をクリックします。
   新規フィールドのコードが生成されます。

このチュートリアルのいずれかで、スーパークラスに存在するフィールドまたはメソッドをオーバーライドすると、新規のフィールドまたはフィールドが、継承されたフィールドまたはメソッドを非表示にすることを示す警告が表示されることがあります。その場合は、**「はい」**をクリックして続行します。

7. **MyNewControllerCmdImpl** クラスを拡張表示し、その中の **performExecute** メソッドを選択して、そのソース・コードを表示します。
8. performExecute メソッドのソース・コードの中で、「Section 1」および「Section 5」をコメント解除します。「Section 1」は、以下のコードをメソッドに導入します。

```
// Create a new TypedProperties for output.
  TypedProperty rspProp = new TypedProperty();
```

Section 5 は、以下のコードをメソッドに導入します。

```
// see how controller command call a JSP

rspProp.put(ECConstants.EC_VIEWTASKNAME, "SampleViewTask");
setResponseProperties(rspProp);
```

ここまでの作業を保管します (**Ctrl + S**)。上記のコードの断片部分は、コントローラー・コマンドによって戻されるビュー名を設定します。

9. この新規のコントローラー・コマンドについて、コマンド・レベルのアクセス制御を指定してください。この場合、コマンド・レベルのアクセス制御ポリシーは、すべてのユーザーがコマンドの実行を許可されていると指定します。このポリシーは SampleCmdACPolicy.xml ファイルに指定されます。新しいアクセス制御ポリシーをロードする方法は、以下のとおりです。
   a. コマンド・プロンプトで、以下のディレクトリーに移動します。
      drive:¥WebSphere¥CommerceServerDev¥bin
   b. 以下のフォームの acpload コマンドを出す必要があります。

      ```
      acpload db_name db_user db_password inputXMLFile
      ```

      ここで
      - *db_name* はユーザーのデータベースの名前
      - *db_user* はユーザーのデータベースのユーザー名
      - *db_password* はデータベース・パスワード
      - *inputXMLFile* はポリシーを含む XML ファイルの名前

      たとえば、以下のコマンドを出すとします。

      ```
      acpload VAJ_Demo user password SampleCmdACPolicy.xml
      ```

10. 以下のようにして、新規 _WCSamples プロジェクトを、サーブレット・エンジンのクラスパスに追加します。
    a. **「ワークスペース」**メニューから、**「ツール」**>**「WebSphere Test Environment」**と選択します。
       「WebSphere Test Environment Control Center」がオープンします。
    b. **「サーブレット・エンジン」**> **「クラスパスの編集」**をクリックします。
       「サーブレット・エンジン・クラスパス」ウィンドウから、**「すべて選択」** > **「OK」**の順にクリックします。
11. 以下のようにして、新しいコマンドをテストします。
    a. Persistent Name Server を開始する (349ページを参照)。
    b. EJB サーバーを開始する (350ページを参照)。
    c. サーブレット・エンジンを開始する (350ページを参照)。
    d. ブラウザーをオープンし、以下の URL を入力します。

       ```
       http://localhost:8080/webapp/wcs/stores/servlet/
          MyNewControllerCmd
       ```

       数分後、ブラウザーは以下のように “Sample JSP” と示されたページを表示します。

図 31.

12. 現在の状態のコードのバージョンを作成します。これによって、いつでもコードをこの状態に復元することができます。バージョンを作成するには、以下のようにします。

    a. **com.ibm.commerce.sample.commands** および **com.ibm.commerce.sample.databeans** パッケージを選択します (複数のパッケージを選択するには、`Ctrl` キーを押しながら強調表示をします)。右クリックして**「管理」**>**「オープン・エディションの作成」**と選択します。

    b. **_WCSamples** プロジェクトを右クリックして、**「管理」**>**「オープン・エディションの作成」**を選択します。

    c. **_WCSamples** プロジェクトを再度右クリックして、**「管理」**>**「バージョン」**を選択します。
    「選択項目のバージョン作成」ウィンドウがオープンします。

    d. **「1 つの名前」**ラジオ・ボタンを選択して、 `mySample 1.2 Completed` と入力し、**「OK」**をクリックします。

ここまでで、新しいコマンドを追加し、統合テスト環境で (簡単に) テストしました。

MyNewControllerCmd の変更

前のセクションでは、MyNewControllerCmd を作成しました。この演習では、独自のカスタマイズ・コントローラー・コマンドを作成する方法をさらによく理解するために、新しいコマンドの内容を詳細に検討します。

まず、ここまでで作成したコードの構造を調べてみましょう。 MyNewControllerCmd インターフェースは ControllerCommand インターフェースの拡張です。またそれは、デフォルトとして使用するインプリメンテーションも定義しています。このクラスは、コマンドが CMDREG テーブルに登録されていないか、またはインプリメンテーション・クラスがそのテーブルに指定されていない場合に使用されます。

さらに、MyNewControllerCmdImpl クラスも作成しました。このインプリメンテーション・クラスは、インプリメントしたいビジネス・ロジック (あるいは、個々のビジネス・タスクを実行するためのタスク・コマンドへの呼び出し) が最終的に入るクラスです。インプリメンテーション・クラスには、以下のメソッドが含まれています。

MyNewControllerCmdImpl 内のメソッド	**説明**
MyNewControllerCmdImpl()	コンストラクター・メソッド。
validateParameters()	コマンドの入力パラメーターのサーバー側の妥当性検査に使用します。
isGeneric()	一般ユーザーがコマンドを呼び出せるかどうかを判別します。
isRetriable()	データベースのロールバックの後にコマンドが再試行されるかどうかを判別します。
performExecute()	コマンドのためのビジネス・ロジックを含みます。

以下のセクションでは、新しいコントローラー・コマンドを更新する方法についての詳細を示します。

JSP テンプレートに変数を渡す

このセクションでは、MyNewControllerCmd を変更して、変数を JSP テンプレートを渡します。変数を表示するために、 DataBeanSampleBean という新しいデータ Bean を使用する必要があります。 JSP テンプレートで変数を表示可能にするには、以下のようにします。

1. VisualAge for Java ワークベンチ・ウィンドウで **_WCSamples** プロジェクトを拡張表示します。
2. **com.ibm.commerce.sample.commands** パッケージの **MyNewControllerCmdImpl** クラスを拡張表示し、その中の **performExecute** メソッドを選択します。
3. performExecute メソッド用のソース・コード内で、Section 2 をコメント解除します。これによって、以下のコードがメソッドに導入されます。

```
// see how controller command pass in variables to JSP

// Add additional parameters in controller command
```

```
// to rspProp for response
//
rspProp.put("ControllerParm1", "Hello world");
rspProp.put("ControllerParm2", "Have a nice day!");
```

上記の部分コードでは、JSP テンプレートに渡されるプロパティーに入れられる 2 つの新しいパラメーターを作成します。ここまでの作業を保管します (**Ctrl + S**)。

4. 変数を表示するために、 DataBeanSampleBean データ Bean が JSP テンプレートによって使用されます。この bean はあらかじめ作成されていますが、以下のようにしてソース・コードを変更してください。
   a. **com.ibm.commerce.sample.databeans** パッケージを拡張表示します。
   b. **DataBeanSampleBean** クラスを拡張表示して **setRequestProperties** メソッドを選択し、そのソース・コードを表示します。
   c. setRequestProperties メソッドのソース・コードの中で、「Section 1」をコメント解除します。これによって、以下のコードがメソッドに導入されます。

      ```
      // copy input TypedProperties to local

      requestProperties = aParam;
      ```

      作業内容を保管します上記の部分コードでは、aParam からの値 (スーパークラスで定義されている) をローカルにコピーします。このコードの断片は、要求オブジェクトからプロパティーを取得するために使われます。
5. 新しいデータ Bean を使用して変数を表示するために、以下のようにして Sample.jsp ファイルを更新します。
   a. テキスト・エディターで、 Sample.jsp ファイルと Sample_All.jsp ファイルをオープンします。これらのファイルは、以下のディレクトリーにあります。

      vaj_drive:¥VAJava¥ide¥project_resources¥IBM WebSphere Test Environment¥hosts¥default_host ¥default_app¥web
   b. Sample_All.jsp のコードの「Section 1」を、 Sample.jsp の <!-- SECTION 1 --> マーカーと <!-- END OF SECTION 1 --> マーカーの間にコピーします。これによって、以下のコードが JSP テンプレートに導入されます。

      ```
      <!-- SECTION 1 -->
      <%
      DataBeanSampleBean testBean = new DataBeanSampleBean ();
      com.ibm.commerce.beans.DataBeanManager.activate (testBean, request);
      %>
      <!-- END OF SECTION 1 -->
      ```

      このセクションのコードはデータ Bean をインスタンス化します。
   c. Sample_All.jsp の「Section 2」を、 Sample.jsp の <!--SECTION 2 --> マーカーと <!-- END OF SECTION 2 --> マーカーの間にコピーします。これによって、以下のコードが JSP テンプレートに導入されます。

```
<!-- SECTION 2 -->
<%
TypedProperty prop = testBean.getRequestProperties();
out.print("<B>List of name value pairs in TypedProperties
            object</B><P>");
// convert from request Properties to query string
for (Enumeration pns = prop.keys(); pns.hasMoreElements();) {
     String paramName = (String) pns.nextElement();
     // do not add the url parameter to the query string
     Object val = prop.get(paramName,null);
     if (val != null) {
          if (val.getClass().isArray()) {
          // flatten the array
         String[] oarray = (String[]) val;
          int len = java.lang.reflect.Array.getLength(val);
          for (int i = 0; i < len; i++) {
          out.print(paramName + "[" + i + "]=" + oarray[i] + "<br>");
          }
     } else {
     // assume that it is a String
     out.print(paramName + "=" + val.toString() + "<br>");
     }
   }
}
%>
<P>
<!-- END OF SECTION 2 -->
```

このファイルを保管します。
このコード・セクションは testBean データ Bean オブジェクトの getRequestProperties メソッドを使用します。プロパティーを循環的に取得して、それらをブラウザーに表示します。

6. 変更内容をテストするために、以下のようにして、変数が JSP テンプレートに表示されることを確認します。
   a. WebSphere Test Environment が稼働していることを確認します。
   b. ブラウザーで、以下の URL を入力します。

      ```
      http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd
      ```

      Sample JSP ファイルが表示されて、コントローラー・コマンドのプロパティーがリストされます。出力は以下のようになります。

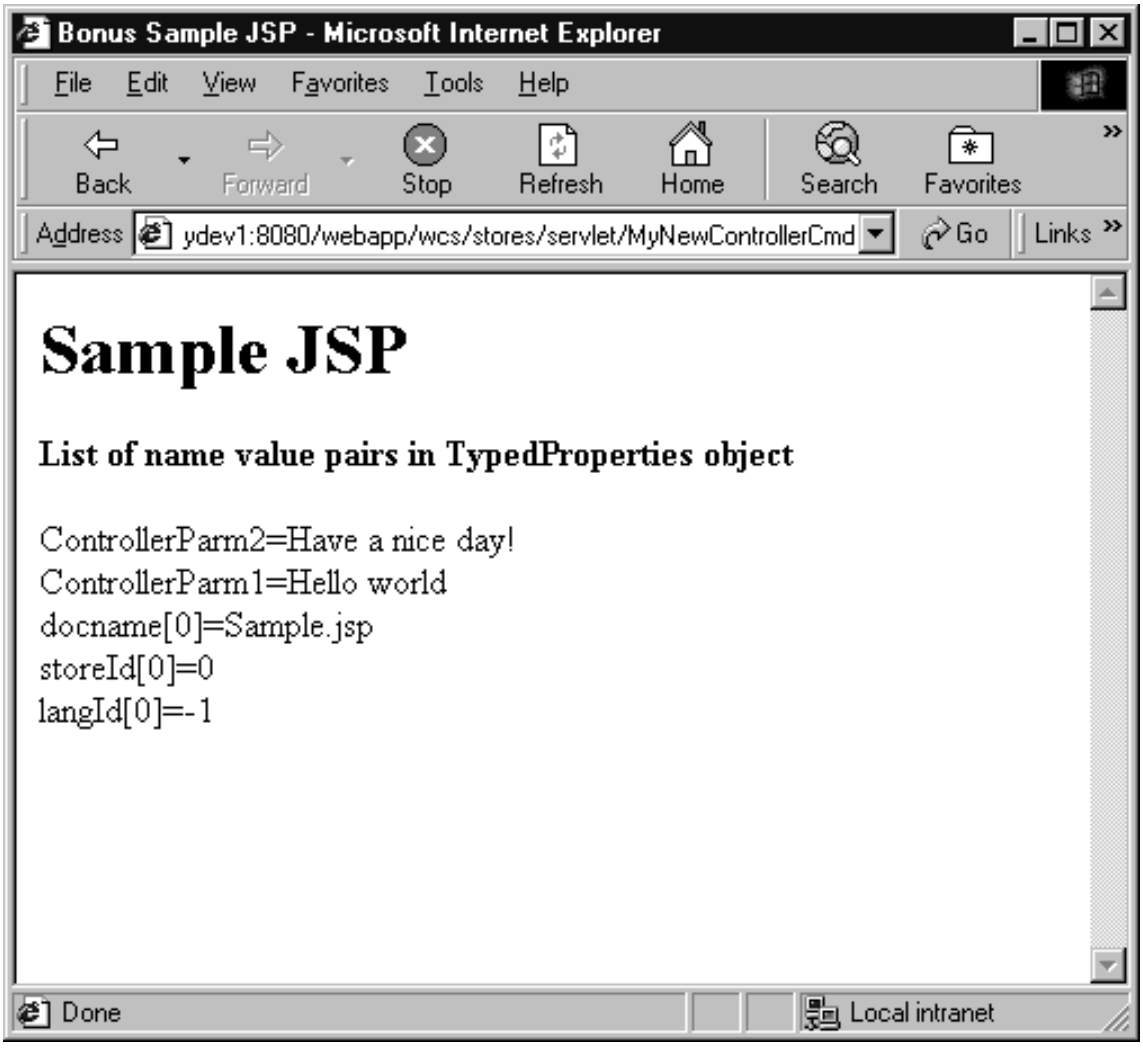


図 32.

7. 現在の状態のコードのバージョンを作成します。このバージョンに `mySample 1.3 Completed` と名前を付けます。コードのバージョン化についての詳細情報が必要な場合、ステップ 12 (215 ページ) を参照してください。

validateParameters メソッドの変更

現在のところ新しいコマンドに入る validateParameters メソッドは、実際のところコマンド・スタブにすぎません。それは以下のコードで構成されます。

```
public void validateParameters() throws ECException {
     }
```

このセクションでは、カスタマイズされた独自のパラメーター検査をコマンドに追加して、パラメーターを JSP テンプレートに渡します。

validateParameters メソッドを変更するとき、パラメーター用の新しいフィールドをクラスに追加します。

新規フィールドの作成: 既存のインターフェースまたはクラスに新しいフィールドを追加するとき、 VisualAge for Java の「Create Fields SmartGuide (フィールドの作成 SmartGuide)」を使用することができます。このセクションでは、新規フィールドを作成する一般的なステップを説明します。チュートリアルのこれ以降のセクションでインターフェースやクラスに新しいフィールドを追加する必要があるとき、以下の情報に従ってください。

新しいフィールドを作成するには、以下のようにします。

1. 新規フィールドを追加するインターフェースまたはクラスを右クリックして、**「追加」>「フィールド」**を選択します。
   「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。
2. **「フィールド名」**フィールドに、新しいフィールドの名前を入力します。
3. フィールド・タイプを指定する場合には、以下のいずれかを行います。
   - **「フィールド・タイプ」**ドロップダウン・リストから、フィールド・タイプを選択します。
   - 必要なフィールド・タイプがリストの中に含まれない場合は、**「ブラウズ」**をクリックします。**「パターン」**フィールドにフィールド・タイプの名前 (または名前の一部) を入力して、**「OK」**をクリックします。

   注: このチュートリアルでは、フィールド・タイプ `String` が指定されるときは必ず、`java.lang` パッケージからです。
4. **「初期値」**フィールドに、フィールドの初期値を入力します。ストリングの初期値を入力する場合には、必ず値を二重引用符 (“ ”) で囲んでください。
5. **「アクセス修飾子」**の値は、必要に応じて適切なラジオ・ボタン (パブリック (public)、保護 (protected)、なし (none)、またはプライベート (private)) を選択します。
6. フィールドの getter メソッドと setter メソッドを生成したい場合には、**「getter および setter メソッドを使ったアクセス」**チェック・ボックスを選択します。これを選択すると、 getter および setter メソッドのプロパティーを以下のようにして指定する必要があります。
   - getter メソッド用に、ラジオ・ボタン (パブリック (public)、保護 (protected)、プライベート (private)、またはなし (none)) のいずれか 1 つを選択します。
   - setter メソッド用に、ラジオ・ボタン (パブリック (public)、保護 (protected)、プライベート (private)、またはなし (none)) のいずれか 1 つを選択します。
7. **「終了」**をクリックします。

validateParameters メソッドに変更を加えるには、以下のようにします。

1. 2 つの新規フィールドを MyNewControllerCommandImpl クラスで作成する必要があります。 1 つは入力ストリング用、もう 1 つは入力整数用に使われます。これらのフィールドを作成するには、以下のようにします。
   a. **com.ibm.commerce.sample.commands** パッケージを拡張表示します。
   b. **MyNewControllerCmdImpl** クラスを選択します。
   c. 以下の値を使用して、フィールドをクラスに追加します。新規フィールドを作成する詳しい方法については、 220 ページの『新規フィールドの作成』を参照してください。

属性名	値
フィールド名	inputString
フィールド・タイプ	String
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

   d. 以下の値を使用して、入力整数用にもう 1 つのフィールドを作成します。

属性名	値
フィールド名	inputInteger
フィールド・タイプ	Integer **注: 「ブラウズ」**をクリックして、Integer と入力します。 int は選択しないでください。
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

2. MyNewControllerCmdImpl クラスの中で、 **performExecute** メソッドを選択します。
3. performExecute メソッドのソース・コードの中で、「Section 3」をコメント解除します。これによって、以下のコードがメソッドに導入されます。

```
// see how controller command pass in input variables to JSP
     rspProp.put("ControllerInput1", getInputString());
     rspProp.put("ControllerInput2", getInputInteger().toString());
```

このコードは、コントローラー・コマンドからデータ Bean へ変数を渡します。作業内容を保管します。

4. MyNewControllerCmdImpl クラスの中で、 **validateParameters** メソッドを選択します。
5. validateParameters ソース・コードの「Section 1」のコメントを解除して、以下のコードをメソッドに導入します。

```
// uncomment to check parameters

      TypedProperty prop = getRequestProperties();

// retrieve required parameters
 //
 try {
          setInputString(prop.getString("input1"));
 } catch (ParameterNotFoundException e) {
          throw new ECApplicationException(
               ECMessage._ERR_CMD_MISSING_PARAM,
               "MyControllerCmdImpl","validateParameters",
               ECMessageHelper.generateMsgParms(e.getParamName()));
 }

 // retrieve optional Integer
 / set input2 = 0 if no input value
 //
 setInputInteger(prop.getInteger("input2", 0));
```

作業内容を保管します。
上記の部分コードは、2 つの入力パラメーターを検査します。「try」ブロックは最初のパラメーターがあるかどうかを判別し、存在しない場合は例外を戻します。 2 番目のパラメーターはオプションであるため、このコードは、パラメーターが欠落している場合、または不適切なタイプである場合に、値を 0 に設定します。

6. 以下のようにして、コマンドをテストします。
   a. 永続ネーム・サーバー、EJB サーバー、およびサーブレット・エンジンが実行中であることを確認します。
   b. ご使用のブラウザーで、URL を入力して以下のことを確認します。
      - ケース 1: パラメーターを欠落させる
        以下のように入力します。

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd
```

コマンドにパラメーターが渡されないので、一般アプリケーション・エラーが、パラメーターが欠落していることを示します。結果は、以下の画面ショットのようになります。

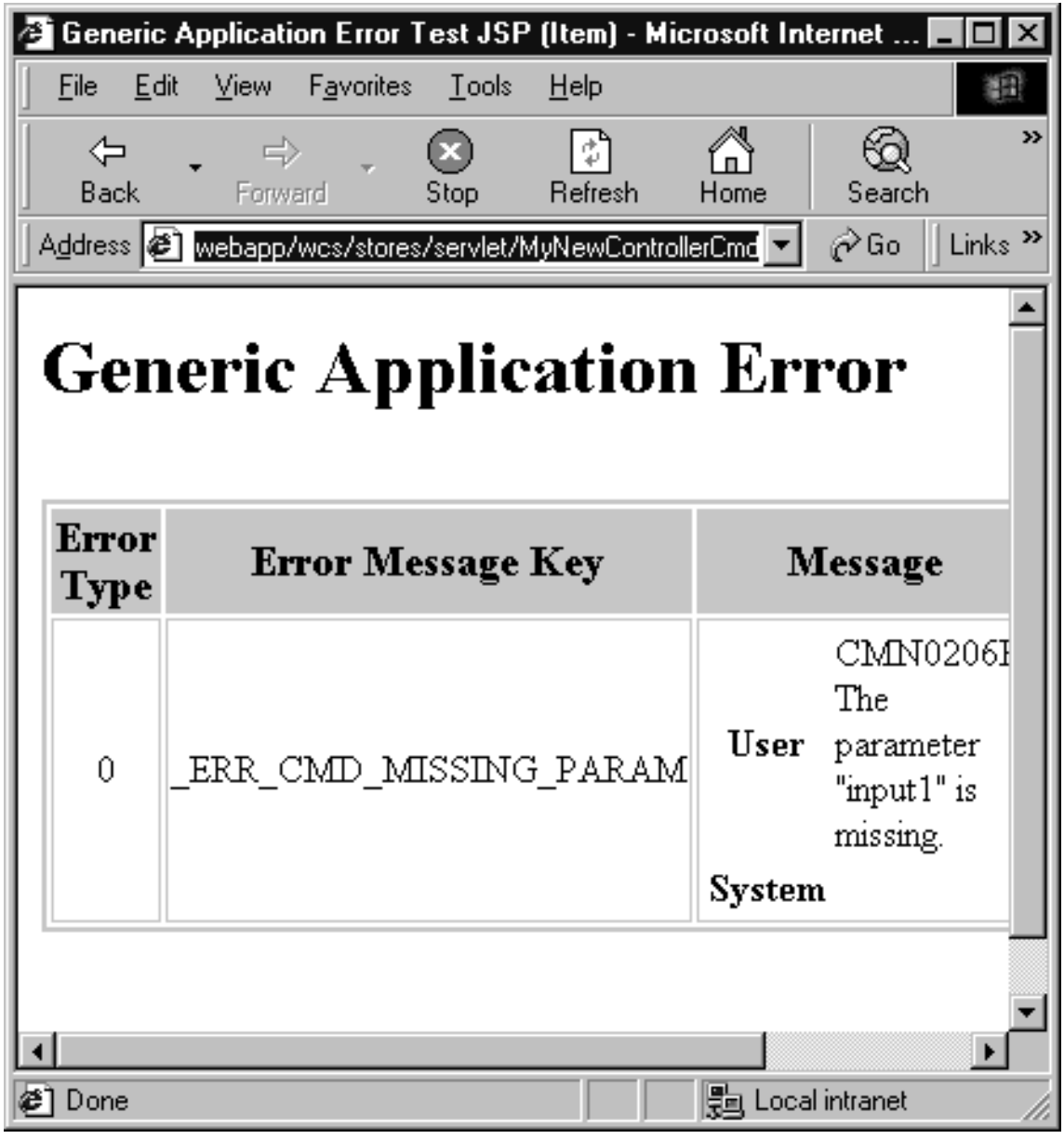


図 33.

注: “Page not found” エラーが表示される場合、サーブレット・エンジンが停止していることがあります。 WebSphere Test Environment Control Center で詳細を検査してください。「一般アプリケーション・エラー」ページの

代わりに「サンプル JSP」ページが表示される場合は、サーブレット・エンジンを停止して再始動する必要があります。あるいは、ブラウザーでページを再ロードしてください。

- ケース 2: 最初のパラメーターを正しく指定し、2 番目のパラメーターを欠落させる
  以下のように入力します。

  ```
  http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
        input1=abc
  ```

  このコマンドの結果として、2 番目のパラメーターが省略されていることに関係なく、「サンプル JSP」ページが表示されます。以下の画面ショットはこの結果を示しています。画面ショットに続いて、エラーが戻されなかった理由を説明しています。

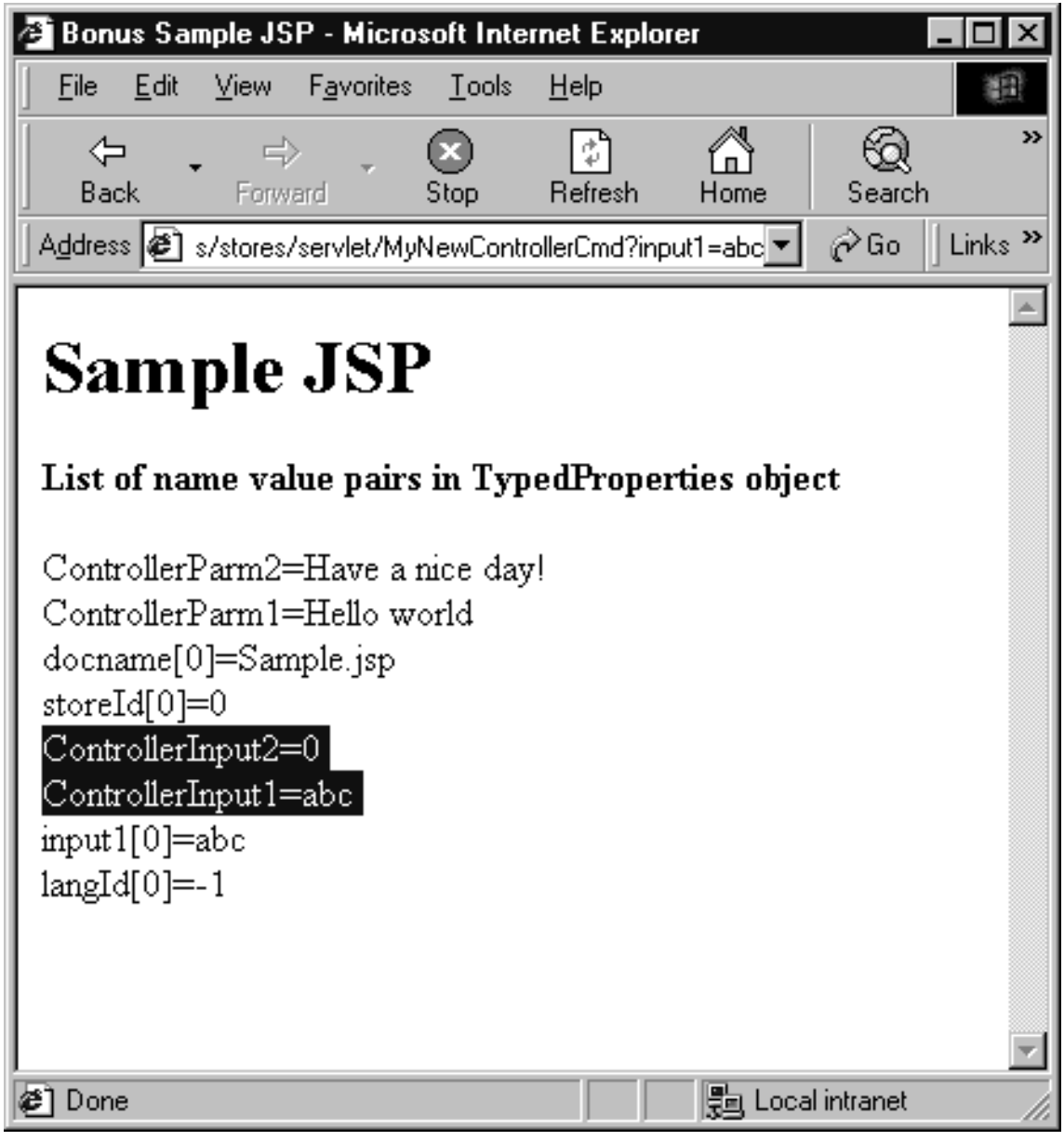


図 34.

エラーが戻されなかった理由は、`getInteger` メソッドの使用方法にあります。具体的には、 `setInputInteger(prop.getInteger("input2", 0))` というコードの行は、 input2 にデフォルト値の 0 を設定します。このデフォルト値は、パラメーターが欠落しているか、タイプが間違っている場合に使用されます。このパラメーターでタイプ検査を強制するためには、コードを

setInputInteger(prop.getInteger("input2")) に変更し、 URL を再度入力します (忘れずにブラウザーを最新表示してください)。一般アプリケーション・エラー・ページが表示されるはずです。

注: コードで setInputInteger(prop.getInteger("input2")) の変更をテストする場合は、次のテスト・ケースの前に、setInputInteger(prop.getInteger("input2", 0)) に戻します。

- ケース 3: 最初のパラメーターを正しく指定し、2 番目のパラメーターが無効である
  以下のように入力します。

  ```
  http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
       input1=abc&input2=abc
  ```

  このコマンドの結果として、「サンプル JSP」ページが表示され、 2 番目のパラメーター (整数) の値がデフォルト値 0 に設定されます。これは、前述のテスト・ケースで説明したとおり、使用した getInteger メソッドの結果です。結果は、以下の画面ショットのようになります。

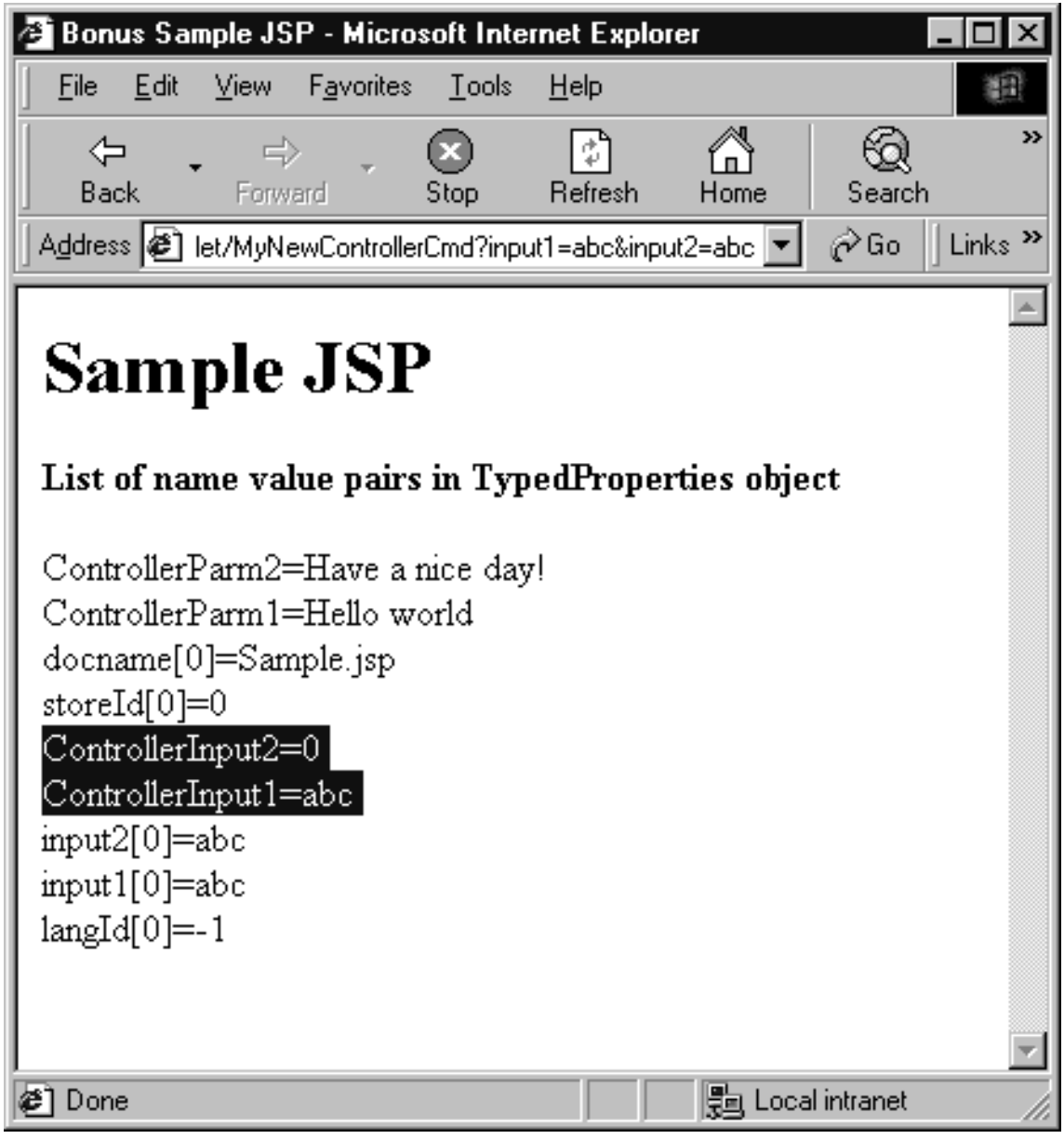


図 35.

- ケース 4: どちらのパラメーターも有効である
  以下のように入力します。

  ```
  http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
      input1=abc&input2=1000
  ```

このコマンドの結果、「サンプル JSP」ページが表示され、入力したとおりに両方の入力値が表示されます。結果は、以下の画面ショットのようになります。

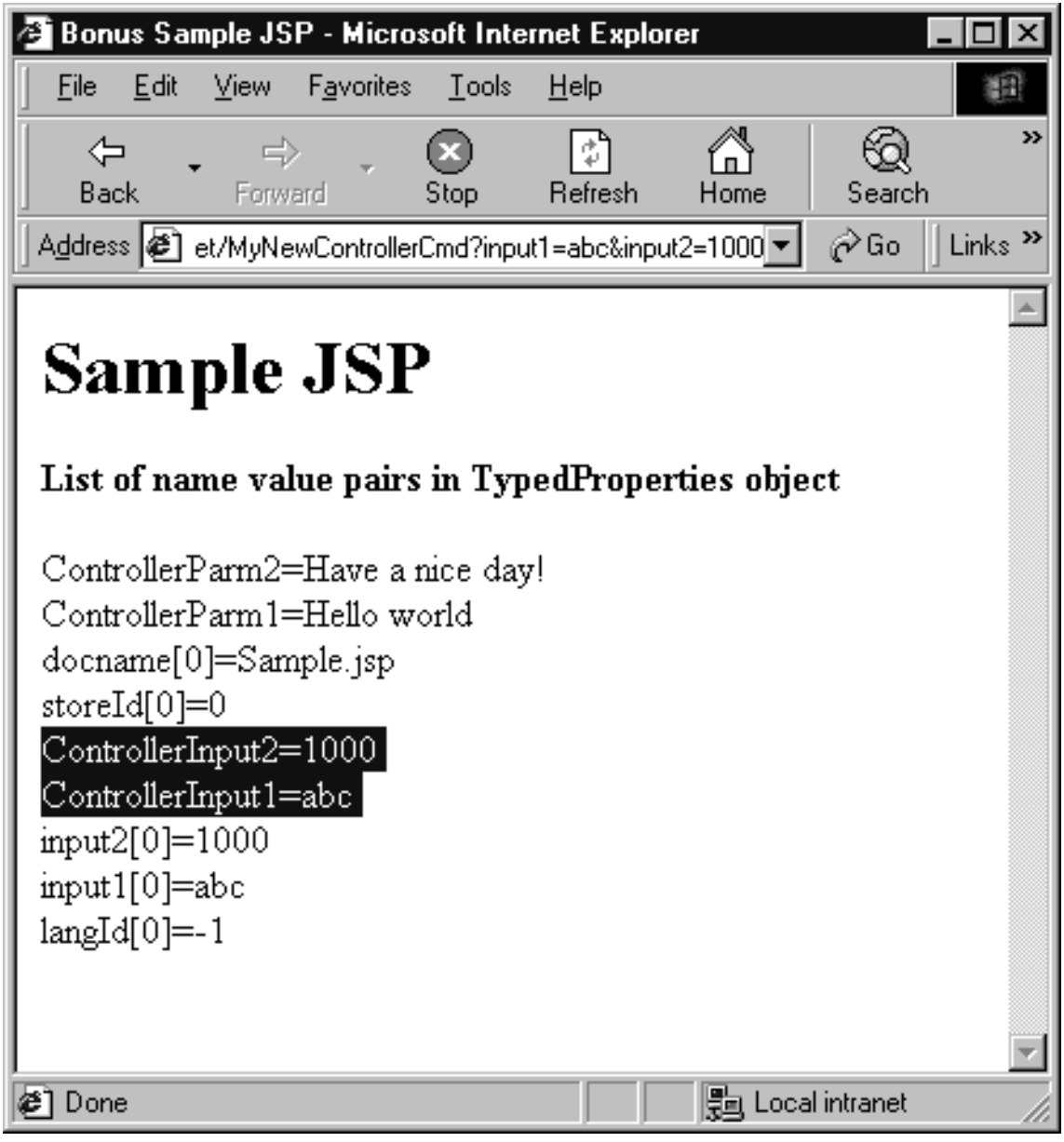


図 36.

7. 現在の状態のコードのバージョンを作成します。このバージョンに mySample 1.4 Completed と名前を付けます。コードのバージョン化についての詳細情報が必要な場合、ステップ 12 (215 ページ) を参照してください。

タスク・コマンドの作成

通常、コントローラー・コマンドはビジネス・プロセスまたは複合機能を表しています。たとえば、オーダーの処理に関連したすべてのビジネス・ロジックは、OrderProcessCmd コントローラー・コマンドの中にカプセル化されています。通常、 1 つのビジネス・プロセスをより小さな特定のタスクに分割することができます。たとえば、OrderProcessCmd コントローラー・コマンドの中には、呼び出されて個々の作業単位を実行するタスク・コマンドがいくつかあります。 OrderProcessCmd コントローラー・コマンド内部で呼び出されるタスク・コマンドの一例として、CalculateOrderTaxTotalCmd があります。

MyNewControllerCmdImpl は、現在のところ、どのタスク・コマンドも呼び出しません。この演習には 2 つのセクションがあります。最初のセクションでは、新しいタスク・コマンドを作成します。 2 番目のセクションでは、コントローラー・コマンドの performExecute メソッドに変更を加えて、タスク・コマンドを呼び出すようにします。

以下の部分コードは、すべてのコメントを除去した、 MyNewControllerCmdImpl クラスからの現在の performExecute メソッドを示しています。

```
public void performExecute() throws ECException {


     super.performExecute();

     TypedProperty rspProp = new TypedProperty();
     rspProp.put("ControllerParm1", "Hello world");
     rspProp.put("ControllerParm2", "Have a nice day!");

     rspProp.put("ControllerInput1", getInputString());
     rspProp.put("ControllerInput2", getInputInteger().toString());

     rspProp.put(ECConstants.EC_VIEWTASKNAME, "SampleViewTask");
     setResponseProperties(rspProp);
}
```

右タスク・コマンド・コードの記述: このセクションでは、新しいタスク・コマンドの書き方を示します。まったく新しいタスク・コマンドを作成するには、インターフェースとインプリメンテーション・クラスを作成する必要があります。タスク・コマンドを作成するときには、インターフェースを com.ibm.commerce.commands.TaskCommand に拡張する必要があります。インプリメンテーション・クラスは、com.ibm.commerce.command.TaskCommandImpl から拡張します。

この演習を完了すると、*MyNewTaskCmd* という新規コマンドが使えるようになります。このコマンドはすべてのストアで使用することができ、各ストアはコマンドの同じインプリメンテーションを使用します。

チュートリアルのこの部分では、新しいタスク・コマンドのインターフェースにフィールドおよびメソッドを追加します。すでに、 MyNewControllerCmdImpl クラス用の新しいフィールドを作成済みです。このインターフェースのフィールドを独自に作成してみてください。詳細情報が必要な場合は、 220 ページの『新規フィールドの作成』を参照してください。

メソッドの作成: このセクションでは、既存のクラスおよびインターフェースにメソッドを追加する一般的なステップを説明します。以下の説明を読んで、チュートリアルの他の部分でメソッドを新規作成する必要がある場合には、再びこれを参照してください。

新しいメソッドを作成するには、以下のようにします。

1. メソッドを追加するクラスまたはインターフェースを右クリックして、**「追加」** > **「メソッド」**を選択します。
   「Create Method SmartGuide (メソッドの作成 SmartGuide)」がオープンします。
2. **「新規メソッドの作成」**が選択されていることを確認して、**「次へ」**をクリックします。
3. **「メソッド名」**フィールドに、新しいメソッドの名前を入力します。
4. 以下のいずれかの方法で、メソッドの戻りタイプを指定します。
   - **「戻りタイプ」**ドロップダウン・リストから、適切な戻りタイプを選択します。たとえば `String` を選択します。
   - 必要な戻りタイプがリストの中に含まれない場合は、**「ブラウズ」**をクリックします。続いて**「パターン」**フィールドに戻りタイプを入力して、**「OK」**をクリックします。

   注: このチュートリアルでは、`String` が戻りタイプとして指定されるときは必ず、これは `java.lang` パッケージからとられます。
5. メソッドがパラメーターを入力とする場合には、**「追加」**をクリックします。「パラメーター」ウィンドウで、パラメーター名その他の必要情報を指定して、**「追加」**をクリックします。すべてのパラメーターを追加し終わったら、**「クローズ」**をクリックします。
6. **「次へ」**をクリックします。
   「属性」ウィンドウがオープンします。
7. メソッドが例外を返すようにする場合は、「属性」ウィンドウの**「追加」**をクリックします。**「パターン」**フィールドで、例外の名前を入力して、**「追加」**をクリックします。すべての例外を追加し終わったら、**「クローズ」**をクリックします。
8. **「終了」**をクリックします。
   メソッドのコードが生成されます。

MyNewTaskCmd コマンドを作成するには、以下のようにします。

1. **com.ibm.commerce.sample.commands** パッケージを拡張表示します。

2. **MyNewTaskCmd** インターフェースを右クリックして、**「追加」>「フィールド」**を選択します。
3. 「Create Field Smart Guide (フィールドの作成 SmartGuide)」を使用して、インターフェースによって使用されるデフォルトのインプリメンテーション・クラスを指定するフィールドを作成します。以下のテーブルの値を使用します。フィールドの作成の詳細情報がさらに必要な場合は、220 ページの 『新規フィールドの作成』を参照してください。

属性名	値
フィールド名	defaultCommandClassName
フィールド・タイプ	String
初期値	"com.ibm.commerce.sample.commands.MyNewTaskCmdImpl"

フィールドのコードが生成されると、以下のようになります。

```
java.lang.String defaultCommandClassName =
     "com.ibm.commerce.sample.commands.MyNewTaskCmdImpl";
```

サイト全体に対して同じインプリメンテーション・クラスが使用され、デフォルト・プロパティーがコマンドに渡されないので、デフォルトのインプリメンテーション権限をコード内に指定することができます。しかし、複数のインプリメンテーションを持つコマンドの場合、または (CMDREG テーブルに保管される) デフォルト・プロパティーを持つコマンドの場合には、そのコマンドを CMDREG テーブルに登録することによって、インターフェースとインプリメンテーション・クラスの間のマッピングを作成する必要があります。

4. 以下のようにして、 MyNewTaskCmd インターフェースに新しいメソッドを追加します。
   a. **MyNewTaskCmd** インターフェースを右クリックして、**「追加」>「メソッド」**を選択します。「Create Method SmartGuide (メソッドの作成 SmartGuide)」を使用して、以下のステップで指定した値を使用して新規メソッドを作成します。メソッドの作成についての詳細情報が必要な場合は、231 ページの 『メソッドの作成』を参照してください。
   b. 以下の値を使用して、顧客のボーナス・ポイントのバランスを取得する新しいメソッドを作成します。

属性名	値
メソッド名	getOldBonusPoint
戻りタイプ	String
パラメーター	なし
例外	なし

注: 直前に作成した継承抽象メソッドが MyNewTaskCmdImpl インプリメンテーション・クラスにインプリメントされないことを示すエラーが表示されることがあります。これは後続のステップで訂正されます。

c. タスク・コマンドからユーザー ID 出力を取得する新しいメソッドを作成します。このメソッドを作成するとき、以下の値を使用します。

属性名	値
メソッド名	getTask_output_userId
戻りタイプ	String
パラメーター	なし
例外	なし

d. タスク・コマンドから出力値を取得する新しいメソッドを作成します。このメソッドを作成するとき、以下の値を使用します。

属性名	値
メソッド名	getTask_output1
戻りタイプ	String
パラメーター	なし
例外	なし

e. タスク・コマンドの最初の入力値を設定する新しいメソッドを作成します。このメソッドを作成するとき、以下の値を使用します。

属性名	値
メソッド名	setTask_input1
戻りタイプ	void
パラメーター名	newTask_input1
参照タイプ	String
例外	なし

f. タスク・コマンドの 2 番目の入力値を設定する新しいメソッドを作成します。このメソッドを作成するとき、以下の値を使用します。

属性名	値
メソッド名	setTask_input2
戻りタイプ	void

属性名	値
パラメーター名	newTask_input2
プリミティブ・タイプ	int
例外	なし

5. 以下のようにして、MyNewTaskCmdImpl クラスに新しいフィールドを追加します。
   a. **MyNewTaskCmdImpl** クラスを右クリックして、**「追加」>「フィールド」**を選択します。「Create Field SmartGuide (フィールドの作成 SmartGuide)」を使用して、以下のステップで指定した値を使用して新規フィールドを作成します。新しいフィールドの作成に関する詳しい追加情報が必要であれば、 220 ページの『新規フィールドの作成』を参照してください。
   b. 以下の値を使用して、インプリメンテーション・クラスに新しいフィールドを作成します。

属性名	値
フィールド名	task_input1
フィールド・タイプ	String
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

   c. 以下の値を使用して、インプリメンテーション・クラスに新しいフィールドを作成します。

属性名	値
フィールド名	task_input2
フィールド・タイプ	int
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public

属性名	値
Setter	public

d. 以下の値を使用して、インプリメンテーション・クラスに新しいフィールドを作成します。

属性名	値
フィールド名	task_output_userId
フィールド・タイプ	String
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

e. 以下の値を使用して、インプリメンテーション・クラスに新しいフィールドを作成します。

属性名	値
フィールド名	oldBonusPoint
フィールド・タイプ	String
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

f. 以下の値を使用して、インプリメンテーション・クラスに新しいフィールドを作成します。

属性名	値
フィールド名	task_output1
フィールド・タイプ	String
初期値	ブランクのままにします。

属性名	値
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

6. **MyNewTaskCmdImpl** クラスから **performExecute** メソッドを選択して、そのソース・コードを表示します。
7. ソース・コードの中で「Section 1」をコメント解除して、以下のコードをメソッドに導入します。

```
// modify the task_input1 and see it in the NVP list

     setTask_output1( "Hello ! " + getTask_input1() );
```

作業内容を保管します
コードの先行セクションで、新規属性がコマンドからの出力として入手可能になります。

タスク・コマンドの呼び出し: タスク・コマンドを作成したら、コントローラー・コマンド内部から、タスク・コマンドを呼び出す必要があります。以下のステップでは、コントローラー・コマンドをそのように変更する方法を説明します。

1. ワークベンチで、 **MyNewControllerCmdImpl** クラスの **performExecute** メソッドを選択します。
2. 「ソース」ペインで、タスク・コマンドを呼び出すための「Section 4」をコメント解除します。これによって、以下のコードがメソッドに導入されます。

```
// see how controller command call a task command

     MyNewTaskCmd cmd = null;
     try {
          cmd = (MyNewTaskCmd) CommandFactory.createCommand(
               "com.ibm.commerce.sample.commands.MyNewTaskCmd",
               getStoreId());
          // Set input parameters to task command
          cmd.setTask_input1(getInputString());
          cmd.setTask_input2(getInputInteger().intValue());
          // This is required for all commands
          cmd.setCommandContext(getCommandContext());
          // Invoke the command's performExecute method
          cmd.execute();
          // retrieve output parameter from task command

          rspProp.put("task_output1", cmd.getTask_output1());
```

```
            if (cmd.getTask_output_userId() != null) {
                 rspProp.put("task_output_userId",
                 cmd.getTask_output_userId());
            }

            if (cmd.getOldBonusPoint() != null) {
                 rspProp.put("task_output_oldBonusPoint",
                 cmd.getOldBonusPoint());
            }

    } catch (ECException ex) {
            // throw the exception as is
            throw (ECException) ex;
    }
```

作業内容を保管します。

上記の部分コードは、コマンド・ファクトリーを使用して、新規のタスク・コマンドを作成します。その後、コマンド・コンテキストを設定し、タスク・コマンドの実行を呼び出して、出力パラメーターをタスク・コマンドから取得します。

3. 以下のように、コントローラー・コマンドの URL を入力することによって、コマンドをテストします。

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

Sample JSP は、要求されたオブジェクトの名前と値の対のリストを表示します。これにはタスク出力値が含まれています。以下のように表示されるはずです。

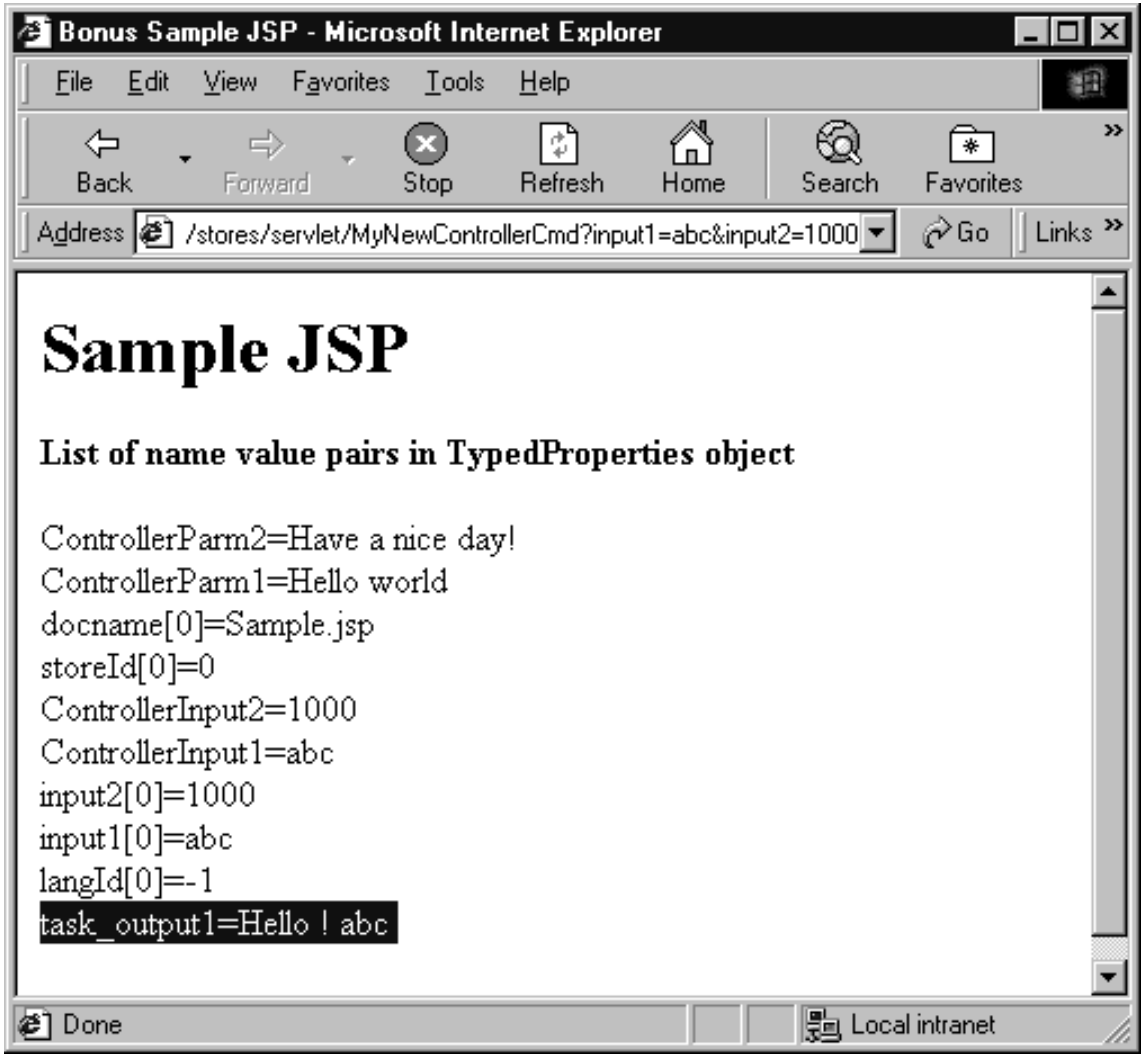


図 37.

4. 現在の状態のコードのバージョンを作成します。このバージョンに `mySample 1.5 Completed` と名前を付けます。コードのバージョン化についての詳細情報が必要な場合、ステップ 12 (215 ページ) を参照してください。

ユーザー ID の妥当性検査

次のステップは、 UserRegistryAccessBean を使用するようタスク・コマンドを変更することです。その目的は、ユーザーによって入力された値が登録ユーザーの値であるかどうかの妥当性検査です。タスク・コマンドの変更に加えて、 DataBeanSampleBean も変更する必要があります。

ユーザー ID 妥当性検査のための MyNewTaskCmdImpl の変更:

MyNewTaskCmdImpl クラスの中の performExecute メソッドを変更して、このメソッドが UserRegistryAccessBean を使用してユーザー ID の妥当性検査を行うようにする必要があります。この新しい機能を performExecute メソッドに追加するには、以下のようにします。

1. ワークベンチで、 **MyNewTaskCmdImpl** クラスの **performExecute** メソッドを選択します。
2. 「ソース」ペインで、 performExecute メソッドの Section 2 をコメント解除します。これによって、以下のコードがメソッドに導入されます。

```
// use UserRegistryAccessBean to check member reference number

     String refNum;
     UserRegistryAccessBean rrb = new UserRegistryAccessBean();

     try {
          rrb = rrb.findByUserLogonId(getTask_input1());
          refNum = rrb.getUserId();
     } catch (javax.ejb.FinderException e) {

     return;

     } catch (javax.naming.NamingException e) {
          throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
          this.getClass().getName(), "performExecute");
     } catch (java.rmi.RemoteException e) {
          throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
          this.getClass().getName(), "performExecute");
     } catch (javax.ejb.CreateException e) {
          throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
          this.getClass().getName(), "performExecute");
     }

      setTask_output_userId(refNum);
```

作業内容を保管します。

DataBeanSampleBean の変更: 定義済みの入力パラメーターを使用するためには、DataBeanSampleBean を変更する必要があります。この bean を変更するには、以下のようにします。

1. ワークベンチで、 **com.ibm.commerce.sample.databeans** パッケージを拡張表示します。
2. 以下のようにして、データ Bean に新しいフィールドを追加します。
   a. **DataBeanSampleBean** クラスを右クリックして、**「追加」>「フィールド」**を選択します。「Create Field SmartGuide (フィールドの作成 SmartGuide)」を使用して、以下のステップで指定した値を使用して新規フィールドを作成します。フィールドの作成に関する詳しい追加情報が必要であれば、 220 ページの『新規フィールドの作成』を参照してください。

b. 以下の値を使用して、データ Bean に新しいフィールドを追加します。

属性名	値
フィールド名	input1
フィールド・タイプ	String
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

c. 以下の値を使用して、データ Bean に新しいフィールドを追加します。

属性名	値
フィールド名	input2
フィールド・タイプ	int
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

d. 以下の値を使用して、データ Bean に新しいフィールドを追加します。

属性名	値
フィールド名	task_output_userId
フィールド・タイプ	String
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public

属性名	値
Setter	public

3. DataBeanSampleBean クラスから **populate** メソッドを選択して、そのソース・コードを表示します。
4. ソース・コードの中で、Section 1A と Section 1B をコメント解除します。これによって、以下のコードがメソッドに導入されます。

```
//// Section 1A //////////
// set additional data fields

try {
   setInput1( getRequestProperties().getString("input1"));
   setInput2( getRequestProperties().getIntValue("input2", 0)  );
   setTask_output_userId(getRequestProperties().getString
     ("task_output_userId"));

//// End of Section 1A /////

   //// Section 2 ////////////
   /*
   // instantiate databean to BonusAccessBean
   setTask_output_oldBonusPoint(getRequestProperties().getString(
      "task_output_oldBonusPoint"));
   */
   //// End of Section 2 ////

//// Section 1A ///////////
// instantiate databean to BonusAccessBean and
// set additional data fields

   }
catch (ParameterNotFoundException e){}

//// End of Section 1B ////
```

作業内容を保管します
上記の部分コードは、getRequestProperties メソッドを使用して、コントローラー・コマンドからデータ・フィールド値を入手します。

ユーザー ID 妥当性検査のための Sample.jsp の変更: ユーザー ID 妥当性検査を実行するためには、現在の JSP テンプレートを変更する必要があります。 Sample.jsp ファイルを変更するには、以下のようにします。

1. テキスト・エディターで、 Sample.jsp ファイルおよび Sample_All.jsp ファイルをオープンします。
2. Sample_All.jsp のコードの「Section 3」を、 Sample.jsp の <!-- SECTION 3 --> マーカーと <!-- END OF SECTION 3 --> マーカーの間にコピーします。以下のコードが JSP テンプレートに導入されます。

```
<!-- SECTION 3 -->

<B>Your first input is &lt; <%=testBean.getInput1()%> &gt;</B>

<%
String userId = testBean.getTask_output_userId();

if (userId == null) {

%>

<UL>
     <LI> This is not a registered user id.
</UL>

<%

} else {

%>

<B>
<UL>
     <LI> 'lt; <%=testBean.getInput1()%> &gt; is a registed user id.
     <LI> The member reference number of this user is <%=userId%>.
</UL>
</B>

<%

}

%>

<B>Your second input is < <%=testBean.getInput2()%> ></B> <P>

<!-- END OF SECTION 3 -->
```

Sample.jsp ファイルを保管します。

3. ブラウザーをオープンし、以下の URL を入力します。

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=abc&input2=1000
```

ブラウザーには Sample JSP ファイルが表示され、以下の画面ショットのように、input1 の値が有効なユーザー ID ではないことを示します。

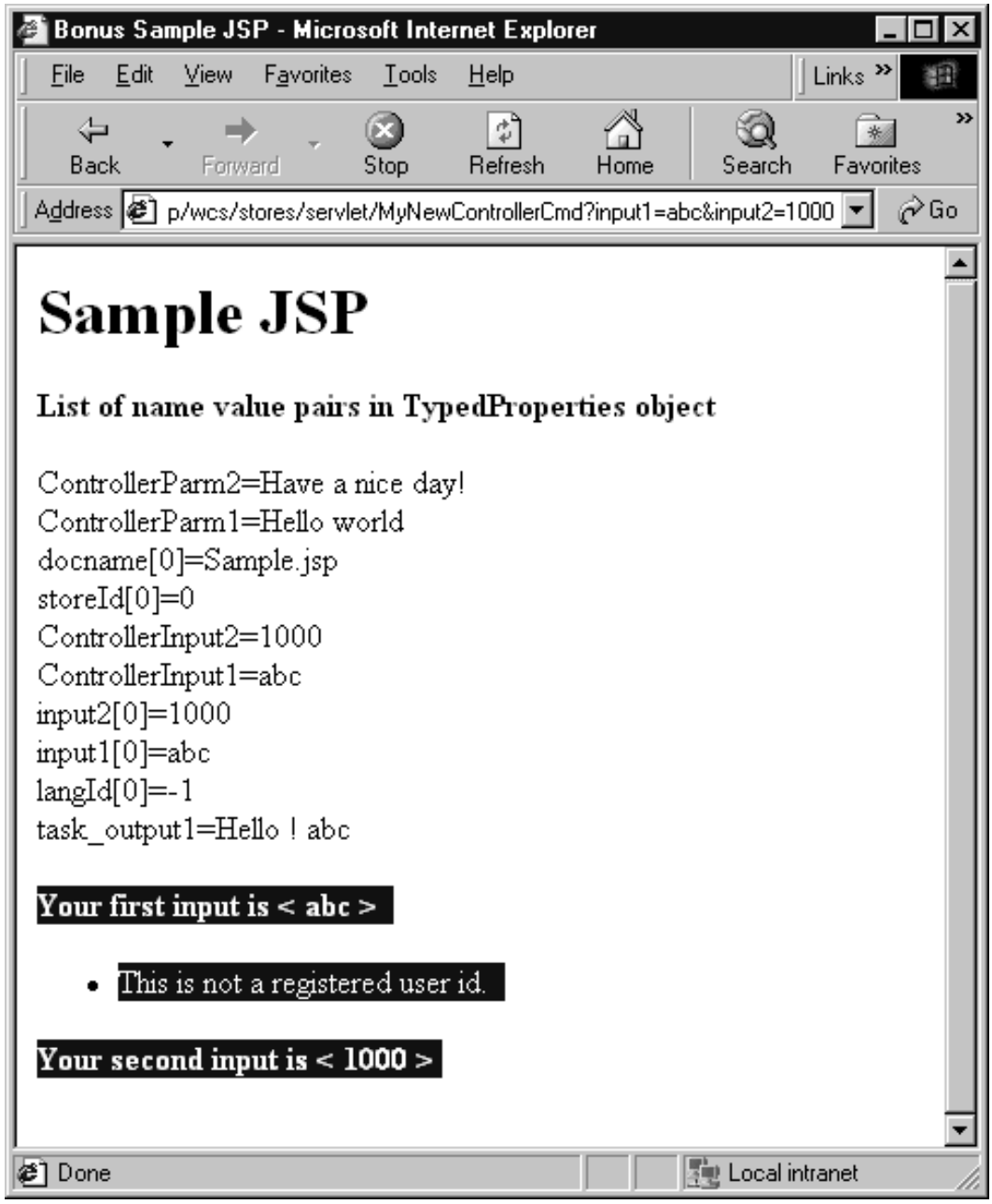


図 38.

4. input1 の値が有効なユーザー ID である場合の結果を見るには、以下の URL を入力します。

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
     input1=wcsadmin&input2=1000
```

結果は、以下の画面ショットのようになります。

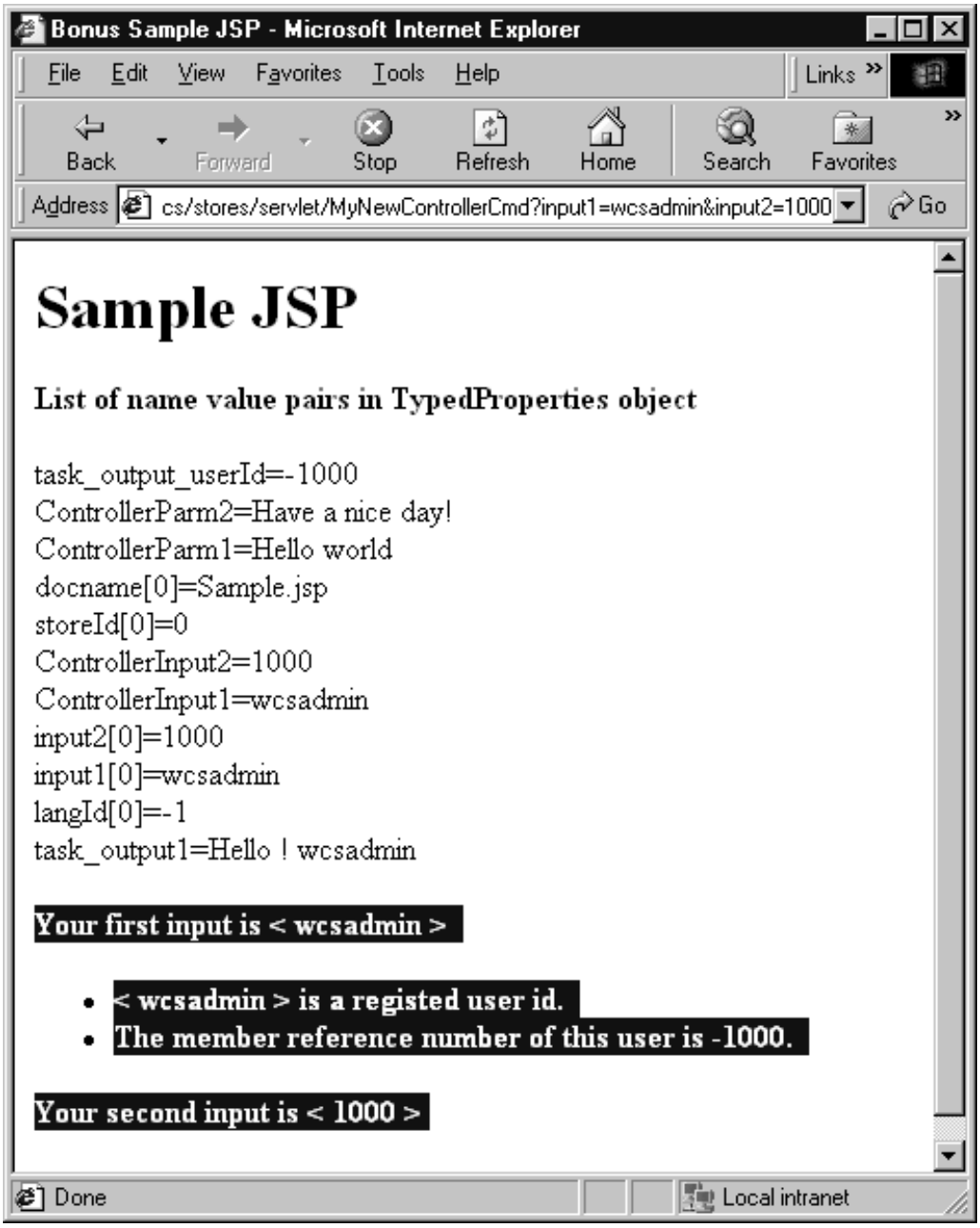


図 39.

5. 現在の状態のコードのバージョンを作成します。このバージョンに `mySample 1.6 Completed` と名前を付けます。コードのバージョン化についての詳細情報が必要な場合、ステップ 12 (215 ページ) を参照してください。

新規エンティティー Bean の作成

このセクションでは、VisualAge for Java を使用して新しいエンティティー Bean を作成する方法を説明します。このシナリオ例では、各ユーザーのボーナス・ポイントの得点をコマース・アプリケーションに組み込むという、ビジネス要件があるとします。WebSphere Commerce データベース・スキーマにはこの情報は入っていないので、この情報を保持するための新しいデータベース・テーブルを作る必要があります。WebSphere Commerce プログラミング・モデルと調和して、データベース・テーブルが一度作成されると、データにアクセスするエンティティー Bean (エンタープライズ Bean) を作成する必要があります。

この例では、VisualAge for Java SmartGuide を使用してそのエンティティー Bean を作成します。

新規データベース・テーブルの作成

エンティティー Bean の作成に備えて、まず新しいデータベース・テーブルを作成する必要があります。作成するテーブルの名前は Bonus とします。

DB2 DB2 データベースを使用している場合は、以下のようにしてテーブルを作成してください。

1. DB2 の「コマンド・センター」をオープンし (**「スタート」**>**「プログラム」**>**「IBM DB2」**>**「コマンド・センター」**)、**「スクリプト」**タブをクリックします。
2. 「スクリプト」ウィンドウで、以下を入力します。

```
connect to your_database_name;
CREATE TABLE Bonus (MEMBERID BIGINT NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade)
```

ここで *your_database_name* は、ご使用のデータベースの名前です。「実行」アイコンをクリックします。
これで、BONUS テーブルが作成されました。

注: 他のユーザーが以前にこのデータベースを使用してこの例を実行したことがある場合には、 Bonus テーブルを作成する前に、以下のコマンドを送出してください。

```
drop table Bonus
```

Oracle Oracle データベースを使用している場合は、以下のようにしてテーブルを作成してください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」**>**「プログラム」**>**「Oracle」**>**「アプリケーション開発」**>**「SQL Plus」**)。
2. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。

3. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで、以下の SQL ステートメントを入力します。

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade);
```

それから Enter を押して SQL ステートメントを実行します。 BONUS テーブルが作成されました。

注: 他のユーザーが以前にこのデータベースを使用してこの例を実行したことがある場合には、 Bonus テーブルを作成する前に、以下のコマンドを送出してください。

```
drop table Bonus;
```

6. データベースの変更をコミットするには

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

BonusBean エンティティー Bean の作成

データベースが作成されたら、新しいエンティティー Bean の作成を始めることができます。次のステップでは VisualAge for Java を使います。新しいエンティティー Bean を作成する方法は、以下のとおりです。

1. EJB グループを作成します。 EJB グループは、エンタープライズ Bean を編成するのを可能にする論理グループです。 EJB グループに対してグローバル操作を実行することができます。これは、そのグループに存在するすべてのエンタープライズ Bean に繰り返されます。たとえば、EJB グループを選択して EJB JAR ファイルにエクスポートする場合、そのグループ内のすべてのエンタープライズ Bean がエクスポートされます。
   このチュートリアルでは、EJB グループを作成して、 Bonus テーブルのカスタマイズに関係するすべてのエンタープライズ Bean を編成します。
   EJB グループを作成するには、以下のようにします。
   a. ワークベンチで、**「EJB」**タブをクリックします。
   b. **「EJB」**メニューから、**「追加」**>**「EJB グループ」**と選択します。
      「Add EJB Group SmartGuide (EJB グループの追加 SmartGuide)」がオープンします。
   c. **「プロジェクト」**フィールドに、`_WCSamplesEntityBeansProject` と入力します。

      注: エンティティー Bean コードは、デプロイメントを目的として、その固有のプロジェクト内に保管する必要があります。

d. **「名前指定した EJB グループの新規作成」**フィールドに WCSSamplesEntityBeans と入力し、**「終了」**をクリックします。

2. 新規エンティティー Bean を作成します。
新しいエンティティー Bean を作成するには、以下のようにします。

a. 「エンタープライズ Bean」ペインで、 **WCSSamplesEntityBeans** EJB グループを右クリックし、**「追加」>「エンタープライズ Bean」**と選択します。
「Create Enterprise Bean SmartGuide (エンタープライズ Bean の作成 SmartGuide)」がオープンします。

b. 以下の情報を入力します。

属性	値
Bean 名	Bonus **注:** 命名規則として、エンティティー Bean は、そのアクセス先のテーブルと同じ名前で呼ばれます。
Bean タイプ	コンテナー管理の永続性 (CMP) フィールドを持つエンティティー Bean
新規 bean クラスの作成	enable
プロジェクト	_WCSamplesEntityBeansProject
パッケージ	com.ibm.commerce.sample.objects
クラス名	BonusBean
スーパークラス	com.ibm.commerce.base.objects.ECEntityBean

そして、**「次へ」**をクリックします。

c. **「CMP フィールドを bean に追加」**テキスト・ボックスの隣の**「追加」**ボタンをクリックして、 BONUS テーブル内の MEMBERID 列にフィールドを追加します。
「Create CMP Fields SmartGuide (CMP フィールドの作成 SmartGuide)」がオープンします。

d. 以下の情報を入力します。

属性	値
フィールド名	memberId
フィールド・タイプ	Long **注:** *long* データ・タイプではなく、 *Long* データ・タイプを使用してください。
鍵フィールド	enable

それから**「終了」**をクリックします。

e. もう一度**「追加」**をクリックして、 BONUS テーブル内の BONUSPOINT 列にフィールドを追加します。

f. 以下の情報を使用して別のフィールドを作成します。

属性	値
フィールド名	bonusPoint
フィールド・タイプ	整数 **注:** *int* データ・タイプではなく、*Integer* データ・タイプを使用してください。
getter および setter メソッドを使ったアクセス	enable
getter および setter メソッドをリモート・インターフェースにプロモートする	enable

それから、「CMP フィールドの作成」ウィンドウで**「終了」**をクリックします。

g. 以下を行って、アクセス制御を使用してこのエンタープライズ Bean を保護します。

1) **「リモート・インターフェースが拡張するインターフェース」** の横の**「追加」**をクリックします。

2) **「パターン」**フィールドで `com.ibm.commerce.security.Protectable` と入力して**「追加」**をクリックします。**「クローズ」**をクリックしてウィンドウをクローズします。

h. **「終了」**をもう一度クリックします。

Bonus エンティティーがエンタープライズ Bean として作成されます。

3. エンティティー Bean の分離レベルを設定するには、以下のようにします。

a. **Bonus** bean を右クリックして、**「プロパティー」**を選択します。

b. **「分離レベル」** ドロップダウン・リストから、**TRANSACTION_READ_COMMITTED** を選択して、**「OK」**をクリックします。

4. 新規のエンタープライズ Bean を作成するとき、VisualAge for Java は、 bean 内の対応する getEntityContext() および setEntityContext(EntityContext) メソッドに加えて、 EntityContext フィールドを生成します。 WebSphere Commerce のプログラミング・モデルに従って、新しい bean は com.ibm.commerce.base.objects.ECEntityBean クラスを拡張し、 ECEntityBean はこのフィールドとこれらのメソッドの独自のインプリメンテーションを提供します。 EntityContext、getEntityContext および setEntityContext をオーバーライドしてはならないので、ここで bean から生成されたフィールドとメソッドを削除しなくてはなりません。
生成された EntityContext フィールドとその getter および setter メソッドを削除するには、以下を行います。

a. 「タイプ」ペインで**「BonusBean」** クラスを選択します。
「メンバー」ペインにこのクラスのフィールドとメソッドが表示されます。

注: 「タイプ」ペインが見えない場合は、「プロパティー」ペインで C/I アイコン (クラス / インターフェース) をクリックします。すると、「タイプ」ペインがオープンします。

b. 「メンバー」ペインで、以下のようにします。

1) **「entityContext」**フィールドを右クリックし、**「削除」**を選択します。

2) **「getEntityContext()」**メソッドを右クリックし、**「削除」**を選択します。

3) **「setEntityContext(EntityContext)」**メソッドを右クリックし、**「削除」**を選択します。

c. ここまでの作業を保管します (Ctrl + S)。

5. 以下のようにして、エンタープライズ Bean に新しい getMemberId を追加します。

a. **「BonusBean」**クラスを右クリックし、**「追加」>「メソッド」**を選択します。
「Create Method SmartGuide (メソッドの作成 SmartGuide)」がオープンします。

b. 以下のテーブルで指定される値を使用して新しいメソッドを作成してください。
新規メソッドの作成について詳細情報が必要な場合は、 231 ページの 『メソッドの作成』を参照してください。

表 11.

属性名	値
メソッド名	`getMemberId`
戻りタイプ	`Long`
パラメーター	なし
例外	なし

c. 新規メソッドが生成されるときに、ソース・コードを表示します。

d. デフォルトでは、メソッドに以下のコードが含まれています。

```
return null;
```

このコードを以下のコードに変更します。

```
return memberId;
```

e. **getMemberId** メソッドを右クリックし、**「追加先」>「EJB Remote Interface (EJB リモート・インターフェース)」**と選択して、新しいメソッドをリモート・インターフェースに追加します。

6. BonusBeanFinderHelper に新しい FinderHelper フィールドを追加します。
このインターフェースには、 FinderHelper メソッド (次のステップで作成される) に対応する検索文節があります。 FinderHelper フィールドを追加するには、以下のようにします。

a. 「タイプ」ペインで、**BonusBeanFinderHelper** インターフェースをクリックします。

b. 「ソース」ペインで、コードを以下のように変更します。

```
public interface BonusBeanFinderHelper {
     public static final String
                    findByMemberIdWhereClause = " (MEMBERID = ?) ";
          }
```

注: “WhereClause” の構文は非常に重要です。それは FinderHelper メソッドに使用されているメソッド名に一致していなければなりません。このケースでは、findByMemberIdWhereClause 内の “findByMemberId” は、次のステップで作成するメソッドの名前に完全に一致しています (findByMemberId)。

7. 新しい FinderHelper メソッドを BonusHome インターフェースに追加します。FinderHelper を新規追加するには、以下のようにします。

   a. 「タイプ」ペインで、**BonusHome** インターフェースを右クリックし、**「追加」>「メソッド」**と選択します。
   「Create Method SmartGuide (メソッドの作成 SmartGuide)」がオープンします。

   b. **「新規メソッドの作成」**を選択し、**「次へ」**をクリックします。

   c. **「メソッド名」**フィールドに、 findByMemberId と入力します。

   d. **「戻りタイプ」**フィールドに、 Bonus と入力します。

   e. **「このメソッドのパラメーター」**の隣の**「追加」**をクリックします。
   「パラメーター」ウィンドウがオープンします。

   f. **「名前」**フィールドに、 argMemberId と入力します。

   g. **「参照タイプ」**を選択して Long と入力します。**「追加」**に続いて**「クローズ」**をクリックします。

   h. **「次へ」**をクリックします。

   i. **「このメッセージが送った例外」**フィールドの横にある**「追加」**をクリックして、**「パターン」**フィールドに RemoteException と入力して、**「追加」**をクリックします。これによって「属性」ウィンドウで java.rmi.RemoteException 例外が追加されます。 (「属性」ウィンドウは、「例外」ウィンドウの後ろに位置することがあります。)

   j. 「例外」ウィンドウの**「パターン」**フィールドで、 FinderException と入力して、**「追加」**、次に**「クローズ」**を入力します。 javax.ejb.FinderException 例外が「属性」ウィンドウにリストされます。

   k. **「終了」**をクリックします。

8. 新規 ejbCreate メソッドを EJB に追加します。このメソッドはホーム・インターフェースにプロモートされ、生成されたアクセス Bean で使用可能になります。このメソッドを作成するには、以下のようにします。

   a. 「タイプ」ペインで、**BonusBean** クラスを選択します。

   b. 「メンバー」ペインで、**ejbCreate(Long)** をクリックします。

c. 以下の記述と一致するように、コードを変更します。

```
public void ejbCreate(java.lang.Long argMemberId,
   Integer argBonusPoint)
   throws javax.ejb.CreateException, java.rmi.RemoteException {
            _initLinks();
    // All CMP fields should be initialized here.
     memberId=argMemberId;
            bonusPoint=argBonusPoint;
    }
```

d. コードを保管すると、VisualAge for Java は **ejbCreate(Long, Integer)** という新しいメソッドを「メンバー」ペインに作成します。

e. 元の **ejbCreate(Long)** メソッドを右クリックして、**「削除」**を選択します。

9. 新しいメソッドをホーム・インターフェースに追加します。これにより、メソッドをアクセス Bean クラスで使用できるようになります。以下のようにして、これを行います。

a. BonusBean クラス内の **ejbCreate(Long, Integer)** メソッドを右クリックして、**「追加先」**>**「EJB ホーム・インターフェース」**を選択します。

10. 以下のようにして、getOwner メソッドを更新します。

a. 「タイプ」ペインで、**BonusBean** クラスを選択します。

b. 「メンバー」ペインで、**getOwner** メソッドをクリックします。

注: 継承されたメソッドを表示するように選択してある場合は、 2 つの getOwner() メソッドが表示されます。 1 つは ECEntityBean クラスから継承されたものです。これは、このステップで選択するものではありません。BonusBean クラスに特定の getOwner メソッドを選択してください。

c. getOwner メソッドのソース・コードは以下のように表示されます。

```
public Long getOwner()
  throws Exception, java.rmi.RemoteException
   {
      return null;
   }
```

メソッドが戻す値を変更しなければなりません。コードのうち変更しなければならない部分が、以下に太字で示されています。

```
public Long getOwner()
  throws Exception, java.rmi.RemoteException
   {
      return getMemberId();
   }
```

作業内容を保管します。

d. 「メンバー」ペインで、**fulfills(Long, String)** メソッドをクリックします。

注: 継承されたメソッドを表示するように選択してある場合は、2 つの fulfills(Long, String) メソッドが表示されます。 1 つは ECEntityBean クラスから継承されたものです。これは、このステップで選択するものではありません。 BonusBean クラスに特定の fulfills(Long, String) メソッドを選択してください。

e. fulfills(Long, String) メソッドのソース・コードは以下のように表示されます。

```
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
  {
      return false;
  }
```

ユーザーが実現しなければならない関係を指定してください。このためには、以下に太字で示される部分のコードを変更しなければなりません。

```
public boolean fulfills(Long member, String relationship)
  throws Exception, java.rmi.RemoteException
  {

      if (relationship.equalsIgnoreCase("creator"))
      {
         return member.equals(getMemberId());
      }      return false;
  }
```

作業内容を保管します。

11. BONUS データベース・テーブルを BonusBean にマップします。データベース・スキーマを BonusBean エンティティーにマップするための最初のステップには、VisualAge for Java のツールを使用してデータベース・スキーマを作成することが含まれます。以下のようにして、スキーマを作成します。

a. ワークスペース・ウィンドウで、**「EJB」**メニューから、**「オープン」**>**「データベース・スキーマ」**を選択します。

b. **「スキーマ」**メニューから、**「スキーマのインポート / エクスポート」**>**「データベースからスキーマをインポートする」**と選択します。
「必須情報」ウィンドウがオープンします。

c. **「スキーマ名」**フィールドに WCSSamples と入力して、**「OK」**をクリックします。
「データベース接続情報」ウィンドウがオープンします。

d. 以下の情報を記入します。

属性	DB2 **DB2 値**	Oracle **Oracle 値**
接続タイプ	COM.ibm.db2.jdbc.app.DB2Driver	Oracle.jdbc.driver.OracleDriver
データ・ソース	jdbc:db2:*wcs_db_name*	jdbc:oracle:thin:@*hostname*:*port*:*SID*

属性	DB2 DB2 値	Oracle Oracle 値
ユーザー名	*wcs_db_user_name*	*wcs_db_user_name*
パスワード	*wcs_db_password*	*wcs_db_password*

値は次のように置き換えられます。

- DB2 *wcs_database_name* は WebSphere Commerce データベースの名前です。
- Oracle *hostname* は Oracle のホスト名です。
- Oracle *port* は Oracle データベースのポート番号です (たとえば 1521)。
- *wcs_db_user_name* はデータベースのユーザー名です。
- *wcs_db_password* はデータベースのパスワードです。

「OK」をクリックします。
「テーブルの選択」ウィンドウがオープンします。

e. **「修飾子」**リストから、ご使用のデータベースの修飾子 (データベース・ユーザー名またはマシン名の場合があります) を選択し、**「テーブル・リストの構築」**をクリックします。使用可能なデータベース・テーブルのリストがロードされます。

f. 「テーブル」パネルから **Bonus** を選択して**「OK」**をクリックします。少しの間、待ちます。

g. 「スキーマ・ブラウザー」で、テーブルの両方の列が表示されるか確認するために、新たに追加されたテーブルをクリックします。

h. **Bonus** テーブルを右クリックして、**「テーブルの編集」**を選択します。
テーブル・エディターがオープンします。

i. **「修飾子」**フィールドのすべてのエントリーを除去します。修飾子情報を除去すれば、異なるデータベースを使用する他のマシンにコードのデプロイメントを実行できるようになります。

j. Oracle 列データのタイプを以下のように変更します。

1) **「MEMBERID」**列を選択してから**「編集」**を選択します。「タイプ」ドロップダウン・リストから、**BIGINT** を選択して**「OK」**をクリックします。

2) **「BONUSPOINT」**列を選択してから**「編集」**を選択します。「タイプ」ドロップダウン・リストから、**INTEGER** を選択して**「OK」**をクリックします。

k. **「OK」**をクリックしてテーブル・エディターを終了します。

l. **「スキーマ」**メニューから、**「スキーマの保管」**を選択します。
「スキーマの保管」ウィンドウがオープンします。

m. 以下の情報を入力します。

属性	値
プロジェクト	_WCSamplesEntityBeansProject
パッケージ	com.ibm.commerce.sample.objects
クラス名	WCSSamplesSchema

それから**「終了」**をクリックして、スキーマ・ブラウザーをクローズします。

12. スキーマが作成されたら、スキーマ・マップを作成することができます。 BONUS テーブルと BonusBean エンティティーとのマップを作成するには、以下のようにします。
    a. **「EJB」**メニューから、**「オープン」**>**「スキーマ・マップ」**と選択します。マップ・ブラウザーがオープンします。
    b. マップ・ブラウザーで、**「データ・ストア・マップ」**メニューから、**「新規 EJB グループのマップ」**を選択します。
    「新規データ・ストアのマップ」ウィンドウがオープンします。
    c. 以下の情報を記入します。

属性	値
名前	WCS Samples
EJB グループ	WCSSamplesEntityBeans
スキーマ	WCSSamples

それから**「OK」**をクリックします。

    d. 「データ・ストア・マップ」パネルで、**「WCS サンプル」**をクリックします。
    e. 「永続クラス」パネルで、**Bonus** をクリックします。
    f. **「テーブル・マップ」**メニューから、**「新規テーブル・マップ」**>**「テーブル・マップを継承なしで追加」**を選択します。
    g. **「テーブル」**ドロップダウン・リストから **Bonus** を選択して、**「OK」**をクリックします。
    h. 「テーブル・マップ」パネルで **Bonus** を選択してそれを右クリックし、**「プロパティー・マップの編集」**を選択します。
    プロパティー・マップ・エディターがオープンします。
    i. 以下のように属性を設定します。

クラス属性	マップ・タイプ	テーブル列
memberId	シンプル	MEMBERID
bonusPoint	シンプル	BONUSPOINT

それから**「OK」**をクリックします。

j. **「データ・ストア・マップ」**メニューから、**「データ・ストア・マップの保管」**を選択します。
「データ・ストア・マップの保管」がオープンします。

k. 以下の情報を入力します。

属性	値
プロジェクト	_WCSamplesEntityBeansProject
パッケージ	com.ibm.commerce.sample.objects
クラス名	WCSSamplesMap

それから**「終了」**をクリックして、マップ・ブラウザーをクローズします。

13. BonusBean エンティティーが作成され、スキーマが正しくマップされたら、エンティティー Bean 用のアクセス Bean を作成する必要があります。このアクセス Bean は、Bonus エンティティーに含まれている情報に、アプリケーションが容易にアクセスできるようにします。すでに作成したエンティティーに基づいて、このアクセス Bean を生成するために、 VisualAge for Java のツールが使用されます。(特に、リモート・インターフェースにプロモートされたメソッドだけが、アクセス Bean によって使用されることになります。) Bonus エンティティー Bean 用のアクセス Bean を作成するには、以下のようにします。

a. 「ワークベンチ」で、「EJB」タブを選択して、 **Bonus** エンタープライズ Bean を右クリックし、**「追加」>「アクセス Bean」**と選択します。
「Create Access Bean SmartGuide (アクセス Bean の作成 SmartGuide)」ウィンドウがオープンします (多少時間がかかることがあります)。

b. 以下の情報が入力されていることを確認してください。

属性	値
EJB グループ	WCSSamplesEntityBeans
エンタープライズ Bean	Bonus
アクセス Bean 名	BonusAccessBean
アクセス Bean タイプ	Copy Helper for an Entity Bean (エンティティー Bean 用のコピー・ヘルパー)

そして、**「次へ」**をクリックします。

c. **「ゼロ引き数コンストラクターのホーム・メソッドを選択する」**ドロップダウン・リストから、**findByMemberId(Long)** を選択します。

d. **init_argMemberId** (「初期プロパティー」列にある) では、**コンバーター**を com.ibm.commerce.base.objects.WCSStringConverter に変更して、**「次へ」**をクリックします。

e. bonusPoint では、**CopyHelper** が選択されていることを確認して、**コンバーター**の値を `com.ibm.commerce.base.objects.WCSStringConverter` に変更し、**「終了」**をクリックします。

注: memberId フィールドに変更を加える必要はありません。

f. “Code Generation Complete” というメッセージが表示されたら、**「OK」**をクリックします。

「プロジェクト」タブに切り替えてから **_WCSamplesEntityBeansProject** プロジェクトを拡張表示し、さらに **com.ibm.commerce.sample.objects** を拡張表示すると、新たに生成されたコードを表示することができます。 `BonusAccessBean` という新しいクラスが、パッケージの内側に表示されます。

14. 次のステップは、デプロイメントを実行されるコードを生成することです。
コード生成ユーティリティーは bean を解析して、 Sun Microsystems の EJB 仕様に合致していることを確認し、 EJB サーバーに固有の規則に従っていることを確認します。加えて、選択されたそれぞれのエンタープライズ Bean ごとに、コード生成ツールは、ホームおよび EJBObject (リモート) インプリメンテーションを生成し、ホームおよびリモート・インターフェース用のインプリメンテーション・クラス、さらには CMP bean 用の JDBC persister および finder クラスを生成します。またそれは、 Java ORB、スタブ、およびタイ・クラス (IIOP 上の RMI アクセスで必要とされる)、さらに、ホームおよびリモート・インターフェース用のスタブを生成します。
CMP エンタープライズ Bean を含む EJB グループを選択した場合、あるいは個々の CMP エンタープライズ Bean を選択した場合は、以下のアイテムも生成されます。
    - テーブル作成ストリング。これは persister クラス内に生成されます。
    - Persister インプリメンテーション。これはテーブルと相互にマップします。

    デプロイメント・コードを生成するには、以下のようにします。

    a. 「エンタープライズ Bean」パネルで EJB タブを選択して、 **Bonus** エンタープライズ Bean を右クリックし、**「デプロイメント・コードの生成」**を選択します。コードの生成は、数分を要します。

15. Bonus エンタープライズ Bean をテストする前に、すべての WebSphere Commerce EJB グループと新しい WCSSamplesEntityBeans EJB グループとを含んでいる新しい EJB サーバーを作成しなければなりません。トランザクション有効範囲を維持するには、これらのグループを同じサーバー上で実行しなければなりません。新しいサーバーを作成したら、それを開始する必要があります。
新しい EJB サーバーを作成または開始するには、以下のようにします。

    a. すべての EJB グループが縮小表示されていることを確認します。

    b. 「エンタープライズ Bean」ペインで、WebSphere Commerce EJB グループすべてと、新しい EJB グループを選択します (すなわち、`WCS` で始まるすべての EJB グループを選択します)。

c. これらの EJB グループが選択された状態で右クリックして、**「追加先」>「サーバー構成」**の順に選択します。
「EJB サーバー構成」ウィンドウがオープンします (多少時間がかかることがあります)。

d. 新しく作成した EJB サーバー (たとえば **EJB Server (server2)**) を右クリックし、**「プロパティー」**を選択して以下を入力します。

属性	DB2 **DB2 値**	Oracle **Oracle 値**
データ・ソース	WebSphere Commerce DB2 DataSource *instance_name*	WebSphere Commerce Oracle DataSource *instance_name*
接続タイプ	<DataSource>	<DataSource>
ユーザー名	*wcs_db_user_name*	*wcs_db_user_name*
パスワード	*wcs_db_password*	*wcs_db_password*
トランザクション・タイムアウト	1200	1200
トランザクション非活動タイムアウト	600000	600000

それから**「OK」**をクリックします。

注: データ・ソースの値は、 *instance_name*.xml ファイルに指定されるデータ・ソースの値と一致しなければなりません。

開発マシンのハードウェアによっては、(たとえば、プロセッサーのスピードなど) **トランザクション・タイムアウト**および**トランザクションの非アクティブ**の EJB サーバー・プロパティーの値を増やす必要があります。

e. WebSphere Test Environment を実行している場合はそれを停止し、さらに、 349 ページの『付録 A. WebSphere Test Environment の開始と停止』で説明されているように、永続ネーム・サーバーおよびその他の EJB サーバーも同様に停止します。

f. 349 ページの『永続ネーム・サーバーの開始と停止』で説明されているように、永続ネーム・サーバーを開始します。

g. 350 ページの『EJB サーバーの開始と停止』で説明されているように、新しい EJB サーバーを開始します。

16. EJB サーバーが開始されたら、テスト・クライアントを開始することができます。テスト・クライアントを使用して、データベース内に新しいレコードを作成します。テスト・クライアントを開始して、このレコードを作成するには、以下のようにします。

a. EJB タブを選択して、**Bonus** エンタープライズ Bean を右クリックし、**「テスト・クライアントの実行」**を選択します。

b. 「EJB 検索」ウィンドウで、**「検索」**をクリックします。
Bonus ウィンドウがオープンします。

c. **「create(Long, Integer)」**をクリックします。「詳細情報」ペインで、以下を充てんします。

属性	値
Long	-1000
Integer	100

それから「EJB クライアント・テスト」ウィンドウ内の「呼び出し」アイコンをクリックします。

注: 最初の属性は、いずれかの登録済みユーザーの memberId に一致していなければなりません。 USERS テーブルを調べることにより、memberId を判別することができます。

17. 直接データベースをクエリーして、データベース・レコードが正しく作成されたか検証します。

DB2 DB2 データベースを使用している場合は、以下のようにしてください。

a. DB2 の「コマンド・センター」をオープンします (**「スタート」**>**「プログラム」**>**「IBM DB2」**>**「コマンド・センター」**)。

b. **「インタラクティブ」**タブを選択します。

c. `connect to wcs_database_name` と入力して、「実行」アイコンをクリックします。

d. `SELECT * FROM Bonus` と入力して、「実行」アイコンをクリックします。
次のような値が戻されるはずです。

列	値
MEMBERID	-1000
BONUSPOINT	100

上記に示されているレコードは、EJB テスト・クライアントにより作成されたものです。

Oracle Oracle データベースを使用している場合は、以下のようにしてください。

a. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」**>**「プログラム」**>**「Oracle - OraHome81」**>**「アプリケーション開発」**>**「SQL Plus」**)。

b. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。

c. **「パスワード」** フィールドに、 Oracle パスワードを入力します。

d. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。

e. 「SQL Plus」ウィンドウで、以下を入力します。

```
select * from BONUS;
```

それから Enter をクリックして SQL ステートメントを実行します。
以下の値が戻されるはずです。

列	値
MEMBERID	-1000
BONUSPOINT	100

18. 以下のように、テスト・クライアントを使用して、 Bonus エンタープライズ Bean がデータベース・レコードに正常にアクセスできるか検証します。

a. Bonus ウィンドウで、**「ホーム」**タブを選択します。

b. 「メソッド」パネルで、**FindByMemberId(Long)** をクリックします。

c. **「Long」**フィールドで、-1000 と入力して「呼び出し」アイコンをクリックします。

d. **「リモート」**タブを選択し、**「メソッド」**を拡張表示して、 **getBonusPoint** をクリックしてから、「呼び出し」アイコンをクリックします。
「詳細情報」パネルには、整数の結果である 100 が表示されます。

e. Bonus ウィンドウと「EJB クライアント・テスト」ウィンドウをクローズします。

Bonus エンティティーと MyNewControllerCmd の統合

ここまでのセクションでは、VisualAge for Java 内で生成されたテスト・クライアントを使用して、新しい Bonus エンティティー Bean をテストしてきました。このセクションでは、Bonus エンティティー Bean と MyNewControllerCmd ロジックとを統合します。 Java コードがいったん更新されたら、Sample.jsp テンプレートが更新されて、ボーナス・ポイントのショッパー・バランスへの更新を許可するインターフェースが作成されます。

Bonus エンティティー Bean の統合には、以下のような高水準のステップが関係しています。

1. MyNewTaskCmdImpl クラスの performExecute メソッドに変更を加えて、新規ボーナス・ポイントを計算してそのポイントを BONUS テーブルに保管するようにします。

2. getResources メソッドを MyNewControllerCmdImpl クラスに追加して、コマンドが使用するリソースのリストを戻します。このメソッドは、アクセス制御の目的で組み込まれます。
3. 新規リソースのための新規アクセス制御ポリシーを作成します。
4. DataBeanSampleBean に変更を加えて、Bonus エンティティー Bean 用に、アクセス Bean から拡張するようにします。アクセス Bean から拡張されたデータ Bean を持つことによって、アクセス Bean からのすべての属性はデータ Bean に継承されます。
5. DataBeanSampleBean 内のメソッドに変更を加えます。
6. WebSphere Test Environment 内のサーブレット・エンジン用のクラスパスに変更を加えて、新しい _WCSamplesEntityBeansProject を組み込みます。
7. Sample.jsp テンプレートに変更を加えて、ユーザーがボーナス・ポイントを入力して結果を表示することを可能にします。

ボーナス・ポイント計算用の MyNewTaskCmdImpl の変更:

MyNewTaskCmdImpl は、 Bonus エンティティー Bean と MyNewControllerCmd との統合のポイントとして使用されます。(それは、 MyNewControllerCmd が MyNewTaskCmd を呼び出すからです。)

ボーナス・ポイントの計算を実行するよう MyNewTaskCmdImpl に変更を加えるには、以下のようにします。

1. VisualAge for Java ワークベンチ・ウィンドウで **_WCSamples** プロジェクトを拡張表示します。
2. **com.ibm.commerce.sample.commands** パッケージを拡張表示してから、**MyNewTaskCmdImpl** クラスを選択してそのソース・コードを表示します。
3. 以下のインポート・ステートメントをコメント解除します。

   ```
   import com.ibm.commerce.sample.objects.*;
   ```

   ここまでの作業を保管します (**Ctrl + S**)。
4. **MyNewTaskCmdImpl** クラスの **performExecute** メソッドを選択します。
5. performExecute メソッド用のソース・コード内で、Section 3 をコメント解除します。これによって、以下のコードがメソッドに導入されます。

   ```
   // use BonusAccessBean to update new bonus point

   String newBonusPoint = null;
   BonusAccessBean bb = new BonusAccessBean();
   try {
        if (refNum != null) {
             bb.setInit_argMemberId(refNum);
             bb.refreshCopyHelper();
             oldBonusPoint = bb.getBonusPoint();
        }
   ```

```
} catch (javax.ejb.FinderException e) {
     try {
          bb = new BonusAccessBean(new Long(refNum),new Integer(0));
          oldBonusPoint = "0";
     } catch (javax.ejb.CreateException ec) {
          throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (javax.naming.NamingException ec) {
          throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
               this.getClass().getName(), "performExecute");
     } catch (java.rmi.RemoteException ec) {
          throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
               this.getClass().getName(), "performExecute");
     }
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
          this.getClass().getName(), "performExecute");
} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
          this.getClass().getName(), "performExecute");
} catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
          this.getClass().getName(), "performExecute");
}


try {
     if (oldBonusPoint != null) {
          int newBP =  Integer.parseInt(oldBonusPoint) + getTask_input2();
          newBonusPoint = Integer.toString( newBP );
          bb.setBonusPoint( newBonusPoint )   ;
          newBonusPoint=bb.getBonusPoint();
          bb.commitCopyHelper();
     }
} catch (javax.ejb.FinderException e) {
     throw new ECSystemException(ECMessage._ERR_FINDER_EXCEPTION,
          this.getClass().getName(), "performExecute");
} catch (javax.naming.NamingException e) {
     throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
          this.getClass().getName(), "performExecute");
} catch (java.rmi.RemoteException e) {
     throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
          this.getClass().getName(), "performExecute");
} catch (javax.ejb.CreateException e) {
     throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
          this.getClass().getName(), "performExecute");
}
```

作業内容を保管します。

getResources メソッドの MyNewControllerCmdImpl クラスへの追加: このセクションでは、新規の getResources メソッドを MyNewControllerCmdImpl クラスに追加します。このメソッドは、処理中にコマンドが使用するリソースのリストを戻します。このメソッドは、リソース・レベルのアクセス制御のために必要です。

getResources メソッドを追加するには、以下のようにします。

1. **「プロジェクト」**タブを選択しておき、**_WCSamples** プロジェクトを選択します。
2. **com.ibm.commerce.sample.commands** パッケージを拡張表示します。
3. **MyNewControllerCmdImpl** クラスを選択して、そのソース・コードを表示します。
4. ソース・コードで、アクセス制御セクションをコメント解除します。このセクションは、以下のコードの断片に示されるようになります。

```
public AccessVector getResources() throws ECException{

   // use UserRegistryAccessBean to check member reference number

   String refNum;
   String methodName="getResources";

   com.ibm.commerce.user.objects.UserRegistryAccessBean rrb =
      new com.ibm.commerce.user.objects.UserRegistryAccessBean();

   try {
      rrb = rrb.findByUserLogonId(getInputString());
      refNum = rrb.getUserId();
   }
   catch (javax.ejb.FinderException e) {

      throw new ECSystemException(ECMessage._ERR_BAD_USER_NAME,
         this.getClass().getName(),methodName);

   }
   catch (javax.naming.NamingException e) {
      throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
         this.getClass().getName(), methodName);
   }
   catch (java.rmi.RemoteException e) {
      throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
         this.getClass().getName(), methodName);
   }
   catch (javax.ejb.CreateException e) {
      throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
      this.getClass().getName(), methodName);
   }


   //find the Bonus bean for this user
   String newBonusPoint = null;
   com.ibm.commerce.sample.objects.BonusAccessBean bb =
      new com.ibm.commerce.sample.objects.BonusAccessBean();
   try {
      if (refNum != null) {
         bb.setInit_argMemberId(refNum);
         bb.refreshCopyHelper();
      }
   }
catch (javax.ejb.FinderException e) {
```

```
    // The user doesn't have a Bonus object so return the container that
    // will hold the bonus object when it's created

    return new AccessVector(rrb);
    }
    catch (javax.naming.NamingException e) {
       throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
          this.getClass().getName(), methodName);
    }

    catch (java.rmi.RemoteException e) {
       throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
       this.getClass().getName(), methodName);
    }

    catch (javax.ejb.CreateException e) {
       throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
       this.getClass().getName(), methodName);
    }

    return new AccessVector(bb);

}
```

作業内容を保管します (Ctrl + S)。

上記のコード・セクションを保管すると、VisualAge for Java は、フィールドとアクセサーをこの特定のビューから取り除きます。その代わり、フィールドとアクセサーは MyNewControllerCmdImpl クラスの下に表示されるようになり、メソッドには M のマークが付きます。

注: このチュートリアルでは、単純化のためにリソース・オブジェクトはこの getResources メソッドで作成されます。実際のアプリケーションでは、リソース・オブジェクトを validateParameters メソッドで作成し、インスタンス変数として保管することをお勧めします。こうすると、getResources および performExecute メソッドによりオブジェクトが再利用されます。

新規リソースのためのアクセス制御ポリシーの設定: サンプルのアクセス制御ポリシーが用意されています。このポリシーは以下のアクセス制御オブジェクトを作成します。

アクション

作成されるアクションは `com.ibm.commerce.sample.commands.MyNewControllerCmd` です。

アクション・グループ

作成されるアクション・グループは `MyNewControllerCmdActionGroup` です。このアクション・グループには 1 つのアクションしか含まれていません。`com.ibm.commerce.sample.commands.MyNewControllerCmd` です。

リソース・カテゴリー
: 作成されるリソース・カテゴリーは com.ibm.commerce.sample.objects.BonusResourceCategory です。このリソース・カテゴリーは Bonus エンティティー Bean のためのものです。

リソース・グループ
: 作成されるリソース・グループは BonusResourceGroup です。このリソース・グループには上記のリソース・カテゴリーしか含まれていません。

ポリシー
: 作成されるポリシーは AllUsersUpdateBonusResourceGroup です。ユーザーがボーナス・オブジェクトの “所有者” である場合にのみ、ユーザーはこのポリシーを使用して MyNewControllerCmd アクションを実行できます。たとえば、ユーザーが wcsadmin ユーザーとしてログオンしている場合、このユーザーは wcsadmin のボーナス・ポイントとしか変更できません。

AllUsersUpdateBonusResourceGroup ポリシーを設定するステップは以下のとおりです。

1. SampleACPolicy.xml ファイルを、acpload コマンドを使用してロードします。
2. SampleACPolicy_***locale***.xml の説明を、acpnlsload コマンドを使用してロードします。
3. ポリシー・レジストリーを最新表示します。アクセス制御ポリシーがロードされるときにサーブレット・エンジンが実行中である場合にのみ、このステップが必要になります。

AllUsersUpdateBonusResourceGroup ポリシーをセットアップするには、以下のようにします。

1. コマンド・プロンプトで、以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServerDev¥bin
2. SampleACPolicy.xml ファイルをロードするには、以下のフォームの acpload コマンドを出す必要があります。

   acpload *db_name db_user db_password inputXMLFile*

   ここで
   - *db_name* はユーザーのデータベースの名前
   - *db_user* はユーザーのデータベースのユーザー名
   - *db_password* はデータベース・パスワード
   - *inputXMLFile* はポリシーを含む XML ファイルの名前です。

   たとえば、以下のコマンドを出すとします。

   acpload VAJ_Demo user password SampleACPolicy.xml
3. ポリシーの説明をロードするには、以下のフォームの acpnlsload コマンドを出す必要があります。

```
acpnlsload db_name db_user db_password inputXMLFile
```

たとえば、以下のコマンドを出すとします。

```
acpnlsload VAJ_Demo user password SampleACPolicy_en_US.xml
```

4. WebSphere Test Environment のサーブレット・エンジンが現在実行中である場合は、これを停止してから再始動して、ポリシー・レジストリーを最新表示します。

ボーナス・ポイント用の DataBeanSampleBean の変更: このセクションでは、以下を実行して、DataBeanSampleBean を変更して BonusAccessBean を拡張します。

1. ワークベンチで、 **com.ibm.commerce.sample.databeans** パッケージを拡張表示します。
2. 以下のようにして、データ Bean に新しいフィールドを追加します。
   a. **DataBeanSampleBean** クラスを右クリックして、**「追加」>「フィールド」**を選択します。「Create Field SmartGuide (フィールドの作成 SmartGuide)」を使用して、以下のステップに従って新規フィールドを作成します。フィールドの作成に関する詳しい追加情報が必要であれば、 220 ページの『新規フィールドの作成』を参照してください。
   b. 以下の値を使用して、データ Bean に新しいフィールドを追加します。

属性名	値
フィールド名	task_output_oldBonusPoint
フィールド・タイプ	String
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

「終了」をクリックします。

3. **DataBeanSampleBean** クラスをクリックして、そのソース・コードを表示します。ソース・コードで、以下のインポート・ステートメントをコメント解除します。

```
import com.ibm.commerce.sample.objects.*;
```

作業内容を保管します。

4. さらに DataBeanSampleBean クラス用のソース・コード内で、 Section 1 をコメント解除して、Section 2 をコメント化します。これにより bean は変更されて、BonusAccessBean から拡張することになります。これらの変更を加えた後、コードは以下のようになります。

```
/// Section 1 ////////////////////////////////////////////////

// Extend the databean to BonusAccessBean

public class DataBeanSampleBean
   extends com.ibm.commerce.sample.objects.BonusAccessBean
   implements SmartDataBean {


//////////////////////////////////////////////////////////////

/// Section 2 ////////////////////////////////////////////////
/*
// Extend the databean to BonusAccessBean

     public class DataBeanSampleBean implements SmartDataBean {
*/

//
//////////////////////////////////////////////////////////////
```

作業内容を保管します。

5. **setTask_output_userId(String)** メソッドを選択して、そのソース・コードを表示します。コードの中で、以下の行を見付けます。

```
public void setTask_output_userId(java.lang.String newTask_output_userId) {
task_output_userId = newTask_output_userId;
```

上記の行の後に以下のコードを入力して、新しい BonusAccessBean をインスタンス化します。

```
//////////////////////////////////////
// Section A : instantiate BonusAccessBean

if (task_output_userId != null)
     this.setInit_argMemberId(newTask_output_userId);

//////////////////////////////////////
```

作業内容を保管します。

6. **populate()** メソッドを選択して、そのソース・コードを表示します。 Section 2 をコメント解除して BonusAccessBean をインスタンス化します。これによって、以下のコードがメソッドに導入されます。

```
setTask_output_oldBonusPoint(getRequestProperties().getString(
    "task_output_oldBonusPoint"));
```

クラスパスの変更: WebSphere Test Environment 内の変更を加えたコマンドをテストする前に、新しい Bonus エンティティー Bean 用に作成された新しいプロジェクトを組み込むように、クラスパスを変更する必要があります。このクラスパスに変更を加えるには、以下のようにします。

1. VisualAge for Java の **「ワークスペース」**メニューから、**「ツール」**-> **「WebSphere Test Environment」**を選択します。
   WebSphere Test Environment Control Center がオープンします。
2. **「サーブレット・エンジン」**をクリックします。
3. サーブレット・エンジンが稼働している場合は、**「サーブレット・エンジンの停止」**をクリックしてから、**「クラスパスの編集」**をクリックします。
4. **「全選択」**をクリックしてから、**「OK」**をクリックします。

ボーナス・ポイント用の Sample.jsp テンプレートの変更: 表示テンプレートに変更を加えるには、以下のようにします。

1. テキスト・エディターで、 Sample.jsp ファイルおよび Sample_All.jsp ファイルをオープンします。
2. Sample_All.jsp のコードの「Section 4」を、 Sample.jsp の <!-- SECTION 4 --> マーカーと <!-- END OF SECTION 4 --> マーカーの間にコピーします。以下のコードが JSP テンプレートに導入されます。

```
<!-- SECTION 4 -->

<h1>
Bonus Administration
</h1>

<%
if (userId != null) {
%>

<B>
<UL>
   <LI> The bonus point before update is
      <%=testBean.getTask_output_oldBonusPoint()%>
   <LI> The bonus point after update is
      <%=testBean.getBonusPoint()%>
</UL>
</B>

<%
}
%>

<br>
<B>Input to command:</B><P>

<FORM NAME=Bonus ACTION="MyNewControllerCmd">
<TABLE>
```

```
    <TR>
         <TD>
              <B>Logon ID </B>
          </TD>
         <TD>
              <input type=text name=input1
                  value='<%=testBean.getInput1()%>'>
         </TD>
    </TR>
    <TR>
          <TD>
              <B>Bonus Point</B>
          </TD>
          <TD>
               <input type=text name=input2>
          </TD>
      </TR>
      <TR>
           <TD COLSPAN=2>
                <input type=submit>
           </TD>
      </TR>
</TABLE>
</FORM>

<!-- END OF SECTION 4 -->
```

Sample.jsp ファイルを保管します。

3. 新規の Bonus bean がアクセス制御の下で保護されており、ユーザーは自分が所有する bean 上の MyNewControllerCmd アクションしか実行できないので、ログインする必要があります。サンプル・ストアのログイン・フィーチャーを使用して、ユーザーはログインすることができます。ここでは、Sample.jsp ファイルを、ストアのディレクトリー構造にコピーしなければなりません。一度ストアにログインすると、Web コントローラーが Sample.jsp ファイルを探してストアのディレクトリー内を検索するためです。 Sample.jsp ファイルを、
   vaj_drive:¥VAJava¥Ide¥project_resources¥IBM WebSphere Test Environment ¥hosts¥default_host¥default_app¥web
   から
   vaj_drive:¥VAJava¥Ide¥project_resources¥IBM WebSphere Test Environment ¥hosts¥default_host¥default_app¥web¥*store_directory* にコピーします。

   注: サンプル・ストアでは、 *store_directory* の値は InFashion です。
4. WebSphere Test Environment が稼働していることを確認します (349 ページの『付録 A. WebSphere Test Environment の開始と停止』を参照)。
5. 以下のようにして、wcsadmin ユーザーとしてログインします。
   - ブラウザーに以下の URL を入力します。

     ```
     http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
          storeId=store_Id&catalogId=catalog_Id&langId=-1
     ```

- **「登録」**リンクをクリックします。
  「登録」または「ログイン」ページが表示されます。
- **「E メール・アドレス」**フィールドで wcsadmin と入力します。
- **「パスワード」**フィールドで wcsadmin ユーザーのためのパスワードを入力してから、**「ログイン」**をクリックします。

  注: オリジナルのパスワードは wcsadmin でしたが、インストール・プロセスの一部として contractPublish コマンドが実行されたときに変更されました。このステップについては、以下の資料で説明しています。

  - Business *WebSphere Commerce Studio Business Developer Edition インストール・ガイド*
  - Professional *WebSphere Commerce Studio Professional Developer Edition インストール・ガイド*

6. ログインが完了してから、同じブラウザーに以下の URL を入力します。

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
    input1=wcsadmin&input2=1000
```

前のすべての出力パラメーターと、ユーザーに対するボーナス・ポイント・バランスを更新できる新しいフォームを記載したページが表示されます。表示されるページは、以下のような画面です。

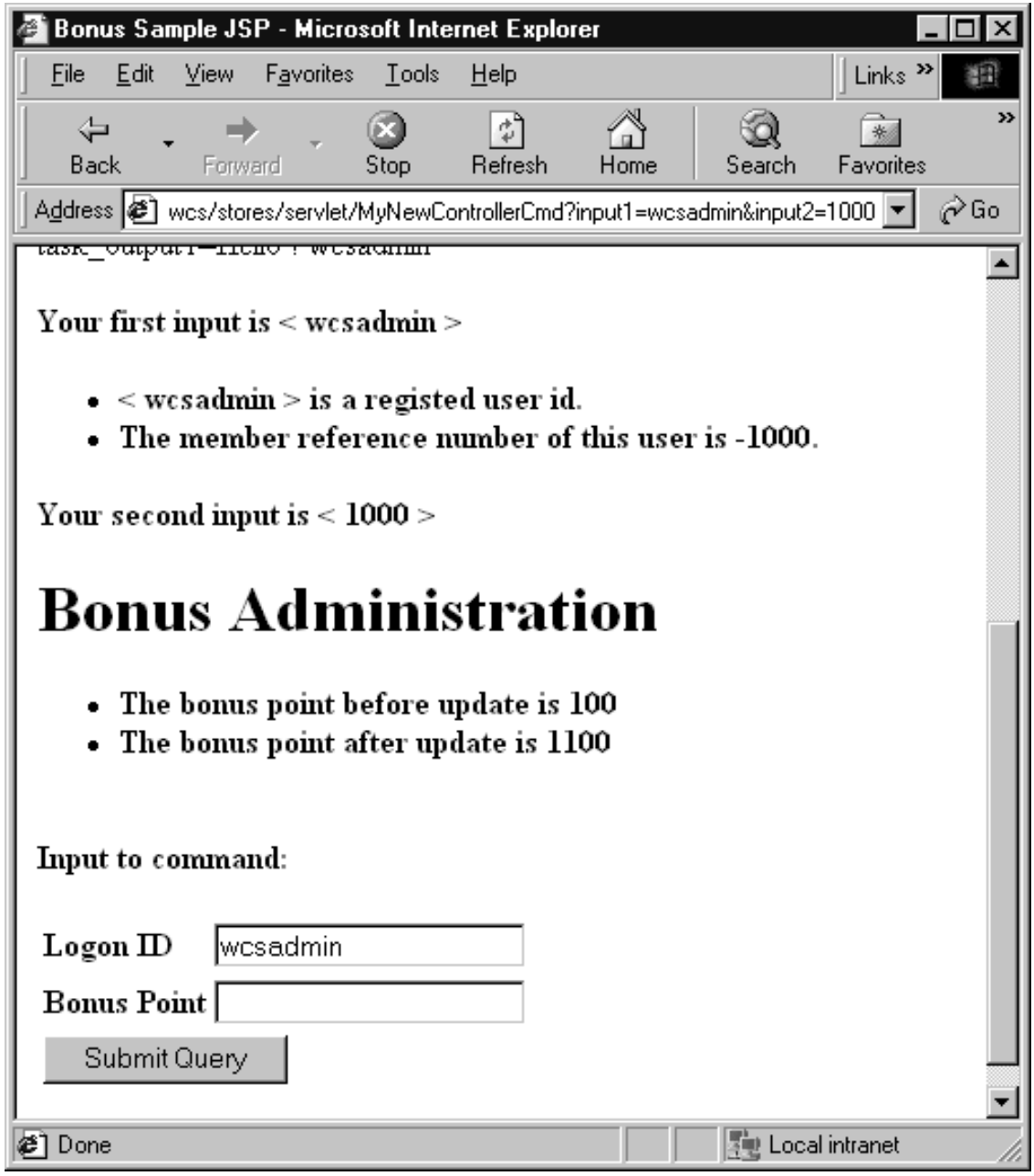


図 40.

7. 現在の状態のコードのバージョンを作成します。このバージョンに `mySample 1.7 Completed` と名前を付けます。コードをバージョン化する場合は、必ず両方のプロジ

ェクトを選択してください。コードのバージョン化についての詳細情報が必要な場合、ステップ 12 (215 ページ) を参照してください。

(オプション) VisualAge for Java でのデバッガーの使用

このセクションでは、ユーザーのコードに区切り点を追加し、 VisualAge for Java のデバッガー・コンポーネントを立ち上げる方法について説明します。このセクションはデバッガーを紹介するために組み込まれています。この優れたフィーチャーの詳細については、VisualAge for Java のオンライン・ヘルプを参照してください。

チュートリアルに必要な新しいコードを 1 つも導入しないため、このセクションはオプションとなります。その代わりに、区切り点を作成し、区切り点の変数の値を検証してから、区切り点を除去します。

ユーザーのコードへの区切り点の追加

ユーザーのコードに区切り点を追加するには、以下のようにします。

1. 以下のようにして、ワークスペースで区切り点を使用できるようにします。
   a. **「ウィンドウ」**メニューから**「デバッグ」**を選択します。**「グローバルな使用可能な区切り点」**が選択されていることを確認します。
2. **「プロジェクト」**タブを選択します。
3. **「_WCSamples」** プロジェクトを拡張表示します。
4. **com.ibm.commerce.sample.commands** パッケージを拡張表示します。
5. **MyNewTaskCmdImpl** クラスを拡張表示します。
6. **performExecute** メソッドを選択して、そのソース・コードを表示します。
7. 「ソース」ペインで、以下のコード行の先頭にカーソルを置き (それからクリックし) ます。

   ```
   setTask_output1( "Hello ! " + getTask_input1() );
   ```

   カーソル位置はこの位置のままにします。
8. **「編集」**メニューから**「区切り点」**を選択します。
9. 「構成: 区切り点 #1」ウィンドウで、**「選択済みのスレッド内」**を選択して**「OK」**をクリックします。
10. WebSphere Test Environment が稼働していることを確認します (349 ページの『付録 A. WebSphere Test Environment の開始と停止』を参照)。
11. 以下のようにして、wcsadmin ユーザーとしてログインします。
    - ブラウザーに以下の URL を入力します。

      ```
      http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
           storeId=store_Id&catalogId=catalog_Id&langId=-1
      ```

- **「登録」**リンクをクリックします。
  「登録」または「ログイン」ページが表示されます。
- **「E メール・アドレス」**フィールドで wcsadmin と入力します。
- **「パスワード」**フィールドで wcsadmin ユーザーのためのパスワードを入力してから、**「ログイン」**をクリックします。

12. ログイン・プロセスが完了してから、以下の URL を入力します。

```
http://localhost:8080/webapp/wcs/stores/servlet/MyNewControllerCmd?
    input1=wcsadmin&input2=1000
```

13. コード内の区切り点に到達すると、「デバッガー」ウィンドウがオープンします。

変数の値の検証

このセクションでは、コマンド実行中の様々な時点で、 task_input1 および task_output1 変数の値を検証する方法について説明します。

MyNewTaskCmdImpl の performExecute メソッド内の区切り点のために「デバッガー」がオープンする場合、このウィンドウに移動して以下を行います。

1. 「変数」ペインで **this** を拡張表示します。
2. **「task_input1」**をクリックします。
   「値」ペインに、実行中のこの時点でのこの変数の値が表示されます。 wcsadmin が表示されるはずです。

task_output1 変数の値が予定通りに設定されているか検証するには、コード内の変数が設定される点に到達するように、デバッガーに performExecute メソッドの実行を継続させなければなりません。これは以下のように行います。

1. ステップオーバー機能を使用して、コードの中を段階的に移動します。これは以下のいずれかの方法で行います。
   - **「選択済み」**メニューから**「ステップオーバー」**を選択します。
   - F6 をクリックします。
   - 「ステップオーバー」アイコンをクリックします。

   この例の目的の場合は、F6 を 4 回クリックします。これによって、task_output1 変数を設定するコードが実行されます。
2. 「変数」ペインで「task_output1」をクリックします。
   「値」ペインに、実行中のこの時点でのこの変数の値が表示されます。 Hello ! wcsadmin が表示されるはずです。
3. 「再開」アイコンをクリックして、コマンドの実行を終了できます。

区切り点の除去

ユーザーのコードから区切り点を除去するには、以下のようにします。

1. ワークベンチ・ウィンドウにスイッチバックします。

2. MyNewTaskCmdImpl クラスの performExecute メソッドについてのソース・コードを表示できることを確認します。
3. 「ソース」ペインで、以下のコード行に移動します。

   ```
   setTask_output1( "Hello ! " + getTask_input1() );
   ```

   ペインの左マージンに、区切り点をマークする青いドットがあることに注意してください。
4. その青いドットをダブルクリックして区切り点を除去します。

これで区切り点はユーザーのコードから除去されました。

WebSphere Test Environment 内のサンプル・ストアへの MyNewControllerCmd の統合

このセクションでは、 MyNewControllerCmd を呼び出すサンプル・ストアのホーム・ページへのリンクを追加します。この統合ステップを実行するには、以下のようにします。

1. テキスト・エディターで、 sidebar.jsp ファイルをオープンします。このファイルは、以下のディレクトリーにあります。

   vaj_drive:¥VAJava¥ide¥project_resources¥IBM WebSphere Test Environment ¥hosts¥default_host¥default_app¥web¥*store_directory*¥include

   vaj_drive は、VisualAge for Java のインストール先のドライブ、 *store_directory* は、サンプル・ストアのディレクトリーの名前です。
2. MyNewControllerCmd へのリンクがある別の行を、最終 </table> タグの前に以下のコードを挿入することによってテーブルに追加します。

   ```
   <tr>
   <td>
   <a href = "MyNewControllerCmd?input1=wcsadmin&input2=1000">
        MyNewControllerCmd</a>
   </td>
   </tr>
   ```

   作業内容を保管します。
3. WebSphere Test Environment が稼働していることを確認します。
4. 以下の URL ブラウザーに入力して、統合をテストします。

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

   「登録」リンクをクリックします。
   「登録」または「ログイン」ページが表示されます。
5. **「E メール・アドレス」**フィールドで wcsadmin と入力します。
6. **「パスワード」**フィールドで wcsadmin ユーザーのためのパスワードを入力してから、**「ログイン」**をクリックします。

7. ユーザーがログオンしてから、サイド・ナビゲーション・ペインで **MyNewControllerCmd** リンクをクリックします。サンプル JSP が表示されます。

   注: サイド・ナビゲーション・ペインに新しいリンクが表示されない場合は、おそらく sidebar.jsp がキャッシュされています。それをキャッシュから除去して、ページを再ロードしてください。コンパイルされた JSP ファイルの削除の情報については、 389 ページの 『付録 C. VisualAge for Java のヒント』 も参照してください。コンパイルされた JSP ファイルの削除後、サーブレット・エンジンを再始動する必要がある場合があります。

(オプション) 新しいビジネス・ロジックのリモート WebSphere Commerce Server へのデプロイメント

このセクションでは、新しいビジネス・ロジックを、リモート WebSphere Commerce Server で実行しているストアにデプロイメントする方法を説明します。これらのデプロイメント・ステップを開始する前に、リモートの WebSphere Commerce Server 上に (InFashion サンプル・ストアに基づいた) ストアを作成しておかなければなりません。

デプロイメント・プロセスには、ターゲットの WebSphere Commerce Server で実行されるステップと同様に、デプロイメント・マシン上で実行されるステップが組み込まれます。

新規コマンド・ロジック用の JAR ファイルの作成

新しいコマンドとデータ Bean のロジックの入った JAR ファイルを作成する必要があります。この JAR ファイルを作成するには、以下のようにします。

1. WebSphere Test Environment を停止します (349 ページの『付録 A. WebSphere Test Environment の開始と停止』を参照)。
2. 「プロジェクト」タブを選択しておき、**_WCSamples** プロジェクトを選択します。
3. そのプロジェクトを強調表示して右クリックし、**「エクスポート」**を選択します。「Export SmartGuide (エクスポート SmartGuide)」がオープンします。
4. **「JAR ファイル」**を選択し、**「次へ」**をクリックします。
5. 「Jar ファイル」フィールドで、以下のように入力します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp¥wcssamples_1.jar
   drive は Commerce Studio のインストール先のドライブです。
6. 属性を以下のように選択します。

属性	値
クラス	チェックする
Java	チェックしない
リソース	チェックする

属性	値
bean	チェックする
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

7. **「終了」**をクリックします。
8. プロンプトが出されたら、新規ディレクトリーの作成を確認します。

作成された JAR ファイルには完全なパッケージ命名情報が含まれていないため、(VisualAge for Java 以外の) 別のパッケージ・ユーティリティーを使用して JAR ファイルを再パッケージ化する必要があります。このファイルを再パッケージ化するには、以下のようにします。

1. コマンド・ウィンドウで、以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp
2. `mkdir temp1` と入力します。
3. `cd temp1` と入力します。
4. パスを以下のように設定します。
   `set PATH=%PATH%;`*drive*`:¥WebSphere¥WebSphereStudio4¥bin;`
   drive は、WebSphere Studio のインストール先のドライブです。
5. `jar xvf ../wcssamples_1.jar` と入力します。
6. `jar cvf ../wcssamples.jar *` と入力します (名前から `_1` が削除されていることに注意)。

新しい EJB グループ用の JAR ファイルの作成

新しい EJB グループ用の JAR ファイルを作成する必要があります。このファイルを作成するには、以下のようにします。

1. 「EJB」タブを選択した状態で、**WCSSamplesEntityBeans** EJB グループを右クリックして、**「エクスポート」**>**「EJB 1.1 JAR」**と選択します。
   「Export to an EJB 1.1 JAR File SmartGuide (EJB 1.1 JAR ファイルへのエクスポート SmartGuide)」がオープンします。
2. **「JAR ファイル」** フィールドに、
   drive:¥WebSphere¥CommerceServerDev¥mytemp¥sampleEntityBeans_DT.jar と入力します。
3. 属性を以下のように選択します。

属性	値
クラス	チェックする
Java	チェックしない

属性	値
リソース	チェックする
ターゲット・データベース	DB2 DB2 データベースにデプロイメントしている場合は、「**DB2 for NT V7.1**」を選択します。 Oracle Oracle データベースにデプロイメントしている場合は、「**Oracle V8**」を選択します。
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

4. 「**終了**」をクリックします。

これで、JAR ファイルが作成されます。

JAR ファイルの名前には接尾部 "_DT" が付きます。つまり、これを WebSphere Commerce アプリケーションにデプロイメントする前に、WebSphere Application Server 付属の EJB デプロイメント・ツールを使ってこの JAR ファイルを実行する必要があります。

新規エンタープライズ Bean 用のインプリメンテーション JAR ファイルの作成

Bonus エンタープライズ Bean 用のインプリメンテーション・コードの入った JAR ファイルを作成する必要があります。このファイルを作成するには、以下のようにします。

1. 「プロジェクト」タブを選択した状態で、「**_WCSSamplesEntityBeansProject**」を右クリックして、「**エクスポート**」を選択します。
   「Export SmartGuide (エクスポート SmartGuide)」がオープンします。
2. 「**JAR ファイル**」を選択し、「**次へ**」をクリックします。
3. 「**JAR ファイル**」フィールドに、
   drive:¥WebSphere¥CommerceServerDev¥mytemp¥sampleImpl_1.jar と入力します。
4. 属性を以下のように選択します。

属性	値
クラス	チェックする
Java	チェックしない
リソース	チェックする
bean	チェックする

属性	値
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

5. **「終了」**をクリックします。

これで、JAR ファイルが作成されます。 VisualAge for Java をクローズします。

作成された JAR ファイルには完全なパッケージ命名情報が含まれていないため、(VisualAge for Java 以外の) 別のパッケージ・ユーティリティーを使用して JAR ファイルを再パッケージ化する必要があります。このファイルを再パッケージ化するには、以下のようにします。

1. コマンド・ウィンドウで、以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp
2. mkdir temp2 と入力します。
3. cd temp2 と入力します。
4. パスを以下のように設定します。
   set PATH=%PATH%;*drive*:¥WebSphere¥WebSphereStudio4¥bin;
   drive は、WebSphere Studio のインストール先のドライブです。
5. jar xvf ../sampleImpl_1.jar と入力します。
6. jar cvf ../sampleImpl.jar * と入力します (名前から _1 が削除されていることに注意)。

JSP ファイルのターゲットの WebSphere Commerce Server へのコピー

Sample.jsp および更新済み sidebar.jsp ファイルを、コードをデプロイメントするストア用の適切なディレクトリーにコピーする必要があります。

更新された JSP テンプレートをターゲットの WebSphere Commerce Server にコピーする前に、そのマシンにオリジナルの JSP テンプレートのバックアップ・コピーを作成したい場合があります。その場合、既存のファイルを sidebar.jsp.bak とリネームしてください。

これらのファイルをコピーするには、以下のようにします。

1. 開発マシンで、以下のディレクトリーに移動します。
   vaj_drive:¥VAJava¥Ide¥project_resources¥IBM WebSphere Test Environment¥hosts¥default_host¥default_app¥web¥*store_directory vaj_drive* は、 VisualAge for Java のインストール先のドライブ、 *store_directory* は、ストアのディレクトリー名です。
   Sample.jsp をコピーします。

2. Sample.jsp を、ターゲットの WebSphere Commerce Server の以下のディレクトリーに貼り付けます。

   drive:¥WebSphere¥AppServer¥installedApps¥
   WC_Enterprise_App_*instance_name*.ear¥
   wcstores.war¥*store_directory*

   ただし *drive* は WebSphere Commerce がインストールされているドライブ、*store_directory* はストアのディレクトリー名、 *instance_name* は WebSphere Commerce インスタンスの名前です。

3. 開発マシンで、以下のディレクトリーに移動します。
   vaj_drive:¥VAJava¥Ide¥project_resources¥IBM WebSphere Test Environment¥hosts¥default_host¥default_app¥web¥*store_directory*¥include
   Copy the sidebar.jsp

4. sidebar.jsp を、ターゲットの WebSphere Commerce Server の以下のディレクトリーに貼り付けます。

   drive:¥WebSphere¥AppServer¥installedApps¥
   WC_Enterprise_App_*instance_name*.ear¥
   wcstores.war¥*store_directory*¥include

JAR ファイルのターゲットの WebSphere Commerce Server へのコピー

JAR ファイルを、開発マシンからターゲットの WebSphere Commerce Server の適切なディレクトリーにコピーしなければなりません。

このほか、新規 EJB グループを含む JAR ファイルに対して実行する必要のあるステップがさらに 2 つあります。第 1 に、 WebSphere Application Server からこのファイルに対して EJB デプロイメント・ツールを実行する必要があります。次に、このファイルに対して modifyIsolationLevel コマンドを実行しなければなりません。その結果、JAR ファイルをターゲットの WebSphere Commerce Server にコピーするこのステップでは、この特定の JAR ファイルが一時ディレクトリーに保管されて、その後のステップで使用できるようになります。

これらのファイルをコピーするには、以下のようにします。

1. 開発マシンで、ディレクトリー
   drive:¥WebSphere¥CommerceServerDev¥mytemp に移動し、以下のファイルを見付けてください。

   - wcssamples.jar
   - sampleImpl.jar
   - sampleEntityBeans_DT.jar

   ここで、*drive* は WebSphere Commerce Studio, Business Developer Edition のインストール先のドライブです。

上記のそれぞれのファイルを、ターゲットの WebSphere Commerce Server 上の特定のディレクトリーにコピーする必要があります。各ファイルを正しい場所に確実に保管するために、以下のステップを注意深くお読みください。

2. wcssamples.jar ファイルをターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。

   drive:¥WebSphere¥AppServer¥InstalledApps¥
   WC_Enterprise_App_*instance_name*.ear¥wcstores.war¥WEB-INF¥lib

   drive は WebSphere Commerce Business Edition のインストール先のドライブで、*instance_name* はインスタンスの名前です (たとえば demo)。

3. JAR ファイルを入れる一時ライブラリー・ディレクトリーを作成します。つまり、以下のディレクトリーを作成します。

   drive:¥WebSphere¥CommerceServer¥temp¥lib

4. sampleImpl.jar ファイルをターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。

   drive:¥WebSphere¥CommerceServer¥temp¥lib

5. sampleEntityBeans_DT.jar ファイルをターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。

   drive:¥WebSphere¥CommerceServer¥temp

EJB デプロイメント・ツールの実行

テスト・サーバーまたは実動サーバーでエンタープライズ Bean を実行する前に、そのエンタープライズ Bean のデプロイメント・コードを生成する必要があります。WebSphere Application Server に付属の Deployment Tool for Enterprise JavaBeans (EJB デプロイメント・ツールともいう) は、エンタープライズ Bean デプロイメント・コードの生成に使用できるコマンド行インターフェースを提供します。

このツールを実行するには、以下のようにします。

1. コマンド・プロンプトで、以下のディレクトリーに移動します。

   drive:¥WebSphere¥CommerceServer¥temp

2. 以下のコマンドを入力して、このツールをシステムに一時的に追加します。

   PATH=*drive*:¥WebSphere¥AppServer¥deploytool;%PATH%

3. 以下のように、ejbdeploy コマンドを入力します。

   ejbdeploy *EJBGroupJARFile WorkingDir OutputJARFile* -nowarn -keep -35 -cp
   ClassPathOfDepJARFiles

   ここで、

- *EJBGroupJARFile* は、デプロイメント・コードを生成したいエンタープライズ Bean の JAR ファイルの完全修飾名です。この場合は、*drive*:¥WebSphere¥CommerceServer¥temp¥sampleEntityBeans_DT.jar になります。
- *WorkingDir* は、コード生成に必要な一時ファイルが保管されているディレクトリーの名前です。
- *OutputJARFile* は出力 JAR ファイルの完全修飾名です。この場合は *drive*:¥WebSphere¥CommerceServer¥temp¥sampleEntityBeans.jar になります。
- -nowarn は、警告および情報メッセージを抑制するためのオプショナル・パラメーターです。
- -keep は、 ejbdeploy コマンドの実行後も作業ディレクトリーを保存するためのオプショナル・パラメーターです。
- -35 は必須パラメーターであり、 WebSphere Application Server バージョン 3.5 付属の EJB デプロイメント・ツールが使用したのと同じ CMP エンティティー Bean 用トップダウン・マッピング規則を使用します。
- -cp *ClassPathOfDepJARFiles* はすべての従属 JAR ファイルのクラスパスです。この場合は、以下のように入力します。

  ```
  "drive:¥WebSphere¥CommerceServer¥temp¥lib¥sampleImpl.jar;
  drive:¥WebSphere¥AppServer¥installedApps¥
     WC_Enterprise_App_instanceName.ear¥lib¥wcsejbimpl.jar"
  ```

  注: クラスパスの値は引用符 (“”) で囲む必要があります。

Bonus bean のトランザクション分離レベルの変更

このステップでは、modifyIsolationLevel コマンドを使用して、 Bonus bean のトランザクション分離レベルを変更します。さらにこのツールは、bean の分離レベルを、特定のタイプのデータベースに必要なレベルに設定します。

modifyIsolationLevel コマンドを実行するには、以下のようにします。

1. ターゲットの WebSphere Commerce Server で、コマンド・ウィンドウをオープンします。
2. 以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServer¥bin
3. 以下の一般構文をもつ modifyIsolationLevel コマンドを出す必要があります。

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType db_type
   ```

   ここで

   - *jar_file_name.jar* は、カスタマイズされたコードを含む JAR ファイルの名前です。
   - *log_file_name* は、情報がログに記録されるべき完全修飾ファイル名です。

- *db_type* はご使用のデータベースのタイプです。 DB2 または ORACLE のいずれかを入力します。

以下は、すべての値が指定された modifyIsolationLevel コマンドの例です。

```
modifyIsolationLevel -jarFile
   D:¥WebSphere¥CommerceServer¥temp¥sampleEntityBeans.jar
   -logFile D:¥WebSphere¥CommerceServer¥instances¥demo¥logs¥output.log
   -dbType DB2
```

コマンド・ウィンドウに例外が表示されない場合、コマンドは正常に実行されました。完了後は、デプロイメント JAR ファイルのタイム・スタンプが変更していることに注意してください。

注: パラメーターの名前はケース・センシティブです。つまり、jar**F**ile と jar**f**ile は違うものです。パラメーターの名前は正しく入力してください。

ターゲット・データベースの更新

WebSphere Test Environment で使用されているもの以外の別のデータベースを使用しているターゲットの WebSphere Commerce Server に新しいビジネス・ロジックをデプロイメントするので、コマンド・レジストリーに加えられた変更内容を反映させ、BONUS テーブルを作成するために、ターゲット・データベースを更新しなければなりません。

DB2 DB2 データベースを使用している場合は、以下のようにしてターゲット・データベースを更新してください。

1. DB2 コマンド・センターをオープンします (**「スタート」**>**「プログラム」**>**「IBM DB2」**>**「コマンド・センター」**)。
2. **「ツール」**メニューから、**「ツール設定」**を選択します。
3. **「ステートメント終了文字の使用」**チェック・ボックスを選択し、セミコロン (;) が文字として指定されていることを確認します。
4. スクリプト・ウィンドウの「スクリプト」タブを選択し、スクリプト・ウィンドウで以下の情報を入力することによって、 URLREG テーブルに必要なエントリーを作成します。

```
connect to your_database_name;
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
    DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
    'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
    'This is a new controller command for test/education purposes.',
    null)
```

ここで、*your_database_name* はデータベースの名前です。続いて「実行」アイコンをクリックします。
このコマンドはすべてのマーチャントによって使われます (STOREENT_ID の 0 の値がそれを示しています)。

5. スクリプト・ウィンドウで以下の情報を入力することによって、 VIEWREG テーブルにエントリーを作成します。

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE) values
   ('SampleViewTask',-1, 0, 'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=Sample.jsp','This is a sample view for the Bonus Point
   exercise', 0, null)
```

   続いて「実行」アイコンをクリックします。

6. スクリプト・ウィンドウで以下の情報を入力することによって、 BONUS テーブルを作成します。

```
CREATE TABLE Bonus (MEMBERID BIGINT NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade)
```

   「実行」アイコンをクリックします。

上記のステップに従うと、コマンド・レジストリー中に MyNewControllerCmd と SampleViewTask が登録され、 BONUS テーブルが作成されます。

Oracle Oracle データベースを使用している場合は、以下のようにしてターゲット・データベースを更新してください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。
2. **「ユーザー名」** フィールドに、ユーザーの Oracle ユーザー名を入力します。
3. **「パスワード」** フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」** フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで以下の情報を入力することによって、 URLREG テーブルに必要なエントリーを作成します。

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS,
    DESCRIPTION, AUTHENTICATED) values ('MyNewControllerCmd',0,
    'com.ibm.commerce.sample.commands.MyNewControllerCmd',0,
    'This is a new controller command for test/education purposes.',
    null);
```

   それから Enter を押して SQL ステートメントを実行します。

6. 「SQL Plus」ウィンドウで以下を入力することによって、 VIEWREG テーブルにエントリーを作成します。

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
   CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE) values
   ('SampleViewTask',-1, 0, 'com.ibm.commerce.command.ForwardViewCommand',
   'com.ibm.commerce.command.HttpForwardViewCommandImpl',
   'docname=Sample.jsp','This is a sample view for the Bonus Point
   exercise', 0, null);
```

それから Enter を押して SQL ステートメントを実行します。

7. 「SQL Plus」ウィンドウで以下を入力することによって、 BONUS テーブルを作成します。

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade);
```

それから Enter を押して SQL ステートメントを実行します。

8. データベースの変更をコミットするには

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

新規リソースのためのアクセス制御ポリシーのロード

このチュートリアルで、保護可能なリソースである新規のエンタープライズ Bean (Bonus bean) を作成しました。つまり、このリソースに関連するアクセス制御ポリシーがあります。また、すべてのユーザーが実行できる新規のコントローラー・コマンドも作成しました。開発マシン上で作業する間に、そのマシンにアクセス制御ポリシーの情報をロードしました。ここで、ターゲットの WebSphere Commerce Server にも同じアクセス制御ポリシーの情報をロードしなければなりません。

アクセス制御ポリシーをセットアップするには、以下のようにします。

1. 以下の CD を CD ドライブに挿入します。
   - Business WebSphere Commerce Business Edition V5.4 Disk 2 CD
   - Professional WebSphere Commerce Professional Edition V5.4 Disk 2 CD
2. 以下のディレクトリーに移動します。

   CD_drive:¥repository¥samples¥programguide¥
3. そのディレクトリーで以下のファイルを見付けます。
   - SampleCmdACPolicy.xml
     この XML ファイルは、新規のコントローラー・コマンドによって使用されるアクセス制御ポリシーを含みます。
   - SampleACPolicy.xml
     この XML ファイルは、新規のエンタープライズ Bean を作成するときに使用されるアクセス制御ポリシーを含みます。
   - SampleACPolicy_*locale*.xml
     locale は言語 ID です。この XML ファイルは、アクセス制御ポリシーの説明を含みます。
4. 以上の 3 つのファイルを以下のディレクトリーにコピーします。
   drive:¥WebSphere¥CommerceServer¥xml¥policies¥xml

5. SampleCmdACPolicy.xml ファイルをロードするには、コマンド・プロンプトを使用して以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServer¥bin
   以下のフォームの acpload コマンドを出す必要があります。

   ```
   acpload db_name db_user db_password inputXMLFile
   ```

   ここで

   - *db_name* はユーザーのデータベースの名前
   - *db_user* はユーザーのデータベースのユーザー名
   - *db_password* はデータベース・パスワード
   - *inputXMLFile* はポリシーを含む XML ファイルの名前

   たとえば、以下のコマンドを出すとします。

   ```
   acpload mall user password SampleCmdACPolicy.xml
   ```

6. SampleACPolicy.xml を入力ファイルとして指定する acpload コマンドを出して、SampleACPolicy.xml ファイルをロードします。たとえば、以下のコマンドを出すとします。

   ```
   acpload mall user password SampleACPolicy.xml
   ```

7. ポリシーの説明をロードするには、以下のフォームの acpnlsload コマンドを出す必要があります。

   ```
   acpnlsload db_name db_user db_password inputXMLFile
   ```

   たとえば、以下のコマンドを出すとします。

   ```
   acpnlsload mall user password SampleACPolicy_en_US.xml
   ```

ポリシーが有効になるためには、通常はレジストリーの最新表示が必要です。このケースでは、このステップを実行する必要はありません。これは、エンタープライズ Bean のデプロイメント・ステップの一部として、 WebSphere Application Server で WebSphere Commerce Server アプリケーションを停止し、再始動するためです。そうでない場合は、 WebSphere Commerce で管理コンソールを使用して、レジストリーを更新できます。管理コンソールについての詳細は、 WebSphere Commerce のオンライン・ヘルプを参照してください。

400 iSeries マシン上で実行する WebSphere Commerce インスタンスをデプロイした場合、アクセス制御ポリシー情報をロードするには別のコマンドを使用します。 acpload コマンドではなく LODWCSAC コマンドを使用し、acpnlsload コマンドではなく LODWCSACD コマンドを使用してください。

LODWCSAC コマンドの構文は、以下のとおりです。

```
LODWCSAC DATABASE(dbName) SCHEMA(schemaName)
PASSWD(instancePassword) INSTROOT('instanceRoot')
INFILE('inputFile')
```

ここで

- *dbName* は、 WRKRDBDIRE コマンドで定義されているリレーショナル・データベースの名前です。
- *schemaName* は、インスタンスのデータベース・スキーマの名前です (これはインスタンス名と同じ名前です)。
- *instancePassword* はインスタンスのパスワードです。
- *instanceRoot* はインスタンスのルートです。インスタンスのルートの例は次のとおりです。

  /QIBM/UserData/WebCommerce/instances/*instanceName*
- *inputFile* は、アクセス・ポリシーを持っている入力 XML ファイルの完全修飾名です。

LODWCSACD コマンドの構文は、以下のとおりです。

```
LODWCSACD DATABASE(dbName) SCHEMA(schemaName)
PASSWD(instancePassword) INSTROOT('instanceRoot')
INFILE('inputFile')
```

アクセス制御ポリシー用の XML ファイルは次のディレクトリーに保管できます。

/QIBM/UserData/WebCommerce/instances/*instanceName*

さらに、アクセス制御ポリシー用の XML ファイル内では、絶対パスを使用して制御 DTD にアクセスしなければなりません。アクセス制御ポリシー用の DTD は /QIBM/ProdData/WebCommerce/xml/policies/dtd ディレクトリーに保管されています。

たとえば、チュートリアル用のアクセス制御ポリシーを iSeries マシン上で実行する Websphere Commerce インスタンスにデプロイする場合、そのチュートリアル用のアクセス制御ポリシーのための XML ファイルの指定を以下のものから変更する必要があります。

```
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">
```

これを以下のように変更します。

```
<!DOCTYPE Policies SYSTEM "/QIBM/ProdData/WebCommerce/
xml/policies/dtd/accesscontrolpolicies.dtd">
```

WebSphere Commerce アクセス制御モデルについての詳細は、 *WebSphere Commerce アクセス・コントロール・ガイド* を参照してください。

現在のエンタープライズ・アプリケーションを WebSphere Application Server からエクスポートする

このステップでは、現在のエンタープライズ・アプリケーションを WebSphere Application Server からエクスポートして、アプリケーション・アセンブリー・ツールでオープンできるようにします。

現在のエンタープライズ・アプリケーションをエクスポートするには、以下のようにします。

1. 現在のエンタープライズ・アプリケーションのエクスポート先となるディレクトリーを作成します。ディレクトリー名は “temp” にしないでください。これは、カスタマイズされたコードがデプロイメントされた後、コードが適切に動作することを確認する前に、通常のシステム保守によってファイルが削除されてしまうのを防ぐためです。このディレクトリーを作成するには、コマンド・プロンプトで以下のようにします。
   a. 以下のディレクトリーに移動します。

      drive:¥WebSphere¥CommerceServer¥

   b. 以下のコマンドを入力します。

      ```
      mkdir working
      ```

      これによって、 *drive*:¥WebSphere¥CommerceServer¥working ディレクトリーが作成されます。
2. WebSphere Application Server 管理コンソールをオープンします。
3. **「WebSphere 管理可能ドメイン」**を拡張表示します。
4. **「Enterprise Applications (エンタープライズ・アプリケーション)」**を拡張表示します。
5. WebSphere Commerce アプリケーションを右クリックします。たとえば、**demo** アプリケーションを右クリックして、**「Export Application (アプリケーションのエクスポート)」**を選択します。
6. **「Export directory (エクスポート先ディレクトリー)」**フィールドに *drive*:¥WebSphere¥CommerceServer¥working と入力します。
   これによって、すべてのリソースを含むアプリケーション全体が WC_Enterprise_App_*instanceName*.ear ファイルにエクスポートされます (ここで、*instanceName* は WebSphere Commerce インスタンスの名前)。
7. **「OK」**をクリックします。アプリケーションのエクスポートには数分かかる場合があります。

エンタープライズ・アプリケーションの XML 構成情報のエクスポート

さらに、エンタープライズ・アプリケーションの XML 構成情報をエクスポートする必要もあります。この情報をエクスポートするには、 WebSphere Application Server に付属のコマンド行ユーティリティー XMLConfig を使用します。

この構成情報をエクスポートするには、以下のようにします。

1. ファイル was.export.app.xml を以下のディレクトリーからコピーします。

   drive:¥WebSphere¥CommerceServer¥xml¥config

   コピー先は以下のディレクトリーです。

   drive:¥WebSphere¥CommerceServer¥working

2. was.export.app.xml ファイルをテキスト・エディターでオープンします。このファイルの中で、すべての $Enterprise_Application_Name$ を以下に置換します。

   WebSphere Commerce Enterprise Application - *instanceName*

   ここで、 *instanceName* は WebSphere Commerce インスタンスの名前です (たとえば demo)。このファイルを保管します。

   注: 挿入する値は、 WebSphere Advanced 管理コンソール に表示されているインスタンスに関する情報と一致していなければなりません。

3. コマンド・プロンプトで、以下のディレクトリーに移動します。

   drive:¥WebSphere¥CommerceServer¥working

4. XMLConfig ツールを起動し、以下のコマンドを入力して、部分的なエクスポートを実行します。

   ```
   xmlConfig -export OutputFile.xml -partial was.export.app.xml
      -adminNodeName wasHostName
   ```

   ここで、*wasHostName* は、 WebSphere Application Server において現在のエンタープライズ・アプリケーションが入っているノードの名前です。さらに、OutputFile.xml はこのコマンドの実行結果として生成されるファイルの名前です。
   was.export.app.xml はステップ 2 で変更したファイルです。

現行のエンタープライズ・アプリケーション内にあるエンタープライズ Bean に関する情報をエクスポートし終えたら、 Bonus bean について記述する XML ファイルに新しいスタンザを追加する必要があります。

Bonus bean について記述する新しいスタンザを追加するには、以下のようにします。

1. 以下のディレクトリーに移動します。

   drive:¥WebSphere¥CommerceServer¥working

2. テキスト・エディターで OutputFile.xml ファイルをオープンします。
3. <ear-file-name> タグを検索して、値を以下のように置き換えます。

drive:¥WebSphere¥CommerceServer¥working¥
WC_Enterprise_App_*instanceName*.ear

4. さらに、以下の例のようにして、 Bonus bean 用の新しいスタンザを追加する必要があります。

```
<ejb-module name="WCSSamplesEntityBeans">
   <jar-file>sampleEntityBeans.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         WebSphere Commerce Server - demo/</application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/WebSphere Commerce DB2 DataSource demo</jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="Bonus_Binding">
         <jndi-name>democom/ibm/commerce/sample/objects/Bonus</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

where

- *user* はデータベースのユーザー名です。
- *password* はデータベースのユーザーのパスワードです。

注:

a. 上記の例にある改行は、表示目的に過ぎません。

b. **$hostName$** の値が現在の管理ノード・サーバー名と一致することを確認してください。さらに、この行に改行がないことも確認してください。

c. <application-server-full-name> の指定は複数行にまたがることができません。

d. Oracle データベースを使用している場合は、以下のようにしてデータ・ソース情報を変更する必要があります。前述のコード断片からとられた以下の行を変更します。

```
<jndi-name>jdbc/WebSphere Commerce DB2 DataSource demo</jndi-name>
```

以下のように変更します。

```
<jndi-name>jdbc/WebSphere Commerce Oracle DataSource demo</jndi-name>
```

e. 独自のアプリケーション (たとえば、このチュートリアルの対象外のもの) をデプロイする場合、 XML ファイルで指定されたご使用のエンタープライズ Bean の JNDI 名が、 VisualAge for Java で使用されている JNDI 名と一致すること (ただし、WebSphere Commerce インスタンス名がその前に付いています) を確認してください。

5. OutputFile.xml ファイルを保管します。

新しい EJB グループをエンタープライズ・アプリケーションにアセンブルする

このステップでは、エンタープライズ・アプリケーションをアプリケーション・アセンブラー・ツールでオープンします。このツール内でオープンしたら、以下の操作を行うことによって、新しい Bonus bean をエンタープライズ・アプリケーションに追加することができます。

1. 新しい Bonus bean をインポートします。新しい EJB グループの JAR ファイルは、エンタープライズ・アプリケーションの EJB Module セクション内に保管されます。
2. インプリメンテーション JAR ファイルを含むように、 Bonus bean のクラスパスを設定します。
3. インプリメンテーション JAR ファイルをアプリケーションに追加します。この JAR ファイルは、エンタープライズ・アプリケーションの Files セクション内に保管されます。
4. Bonus bean に含まれるメソッド用の WebSphere Application Server セキュリティーをセットアップします。

新しい EJB グループをエンタープライズ・アプリケーションにアセンブルするには、以下のようにします。

1. 以下のようにして、現在のエンタープライズ・アプリケーションのバックアップを取ります。
   a. コマンド・プロンプトで、以下のディレクトリーに移動します。

      ```
      drive:¥WebSphere¥CommerceServer¥working
      ```

   b. 以下のコマンドを入力します。

      ```
      copy WC_Enterprise_App_instanceName.ear
         WC_Enterprise_App_instanceName.ear.bak
      ```

2. WebSphere Application Server 管理コンソールをオープンします。
3. **「ファイル」**メニューから、**「ツール」**>**「Application Assembly Tool」**を選択します。
4. 「ウェルカム」ウィンドウがオープンしたら、**「キャンセル」**を選択して、このウィンドウをクローズします。
5. 以下のようにして、作業対象のエンタープライズ・アプリケーションをオープンします。
   a. **「ファイル」**メニューから、**「オープン」**を選択します。
   b. **「File name (ファイル名)」**フィールドで、以下を入力します。

      ```
      drive:¥WebSphere¥CommerceServer¥working¥
         WC_Enterprise_App_instanceName.ear
      ```

それから**「オープン」**をクリックします。アプリケーションがオープンするまで待って、次のステップに進みます。これには数分かかります。

6. **「EJB Modules (EJB モジュール)」**を右クリックして、**「Import (インポート)」**を選択します。
7. **「File name (ファイル名)」**フィールドで、以下を入力します。

   drive:¥WebSphere¥CommerceServer¥temp¥sampleEntityBeans.jar

   それから**「オープン」**をクリックします。「Confirm Values (値の確認)」ウィンドウで、**「OK」**をクリックします。
8. sampleEntityBeans.jar ファイルがインポートされたら、**WCSSamplesEntityBeans** EJB グループまでスクロールして、このグループを選択します。
   このグループに関する情報が右側の画面に表示されます。
9. 新しいエンタープライズ Bean の classpath フィールドに、すべての従属 JAR ファイルを入力します。この場合は、以下のように入力します。

   lib/sampleImpl.jar lib¥wcsejbimpl.jar
10. **「適用」**をクリックします。
11. 以下のようにして、 sampleImpl.jar ファイルをアプリケーションに追加します。
    a. エンタープライズ・アプリケーションの**「Files (ファイル)」**ノードを右クリックして、**「Add Files (ファイルの追加)」**を選択します。エンタープライズ・アプリケーションの**「Files (ファイル)」**ノードは、階層ツリーの下の方にあります。エンタープライズ・アプリケーション内には、コンポーネント用の他の「Files (ファイル)」ノードも存在しますが、必ずアプリケーション全体の「Files (ファイル)」ノードを選択してください。
    b. 「Add Files (ファイルの追加)」ウィンドウで、**「ブラウズ」**をクリックします。
    c. *drive*:¥WebSphere¥CommerceServer¥temp に移動します。
    d. このディレクトリーを強調表示して、**「選択」**をクリックします。
    e. 「Add Files (ファイルの追加)」ウィンドウに戻ります。
    drive:¥WebSphere¥CommerceServer¥temp ディレクトリーの内容が表示されることを確認します。 lib ディレクトリーを強調表示します。
    lib ディレクトリーの内容が右側の画面に表示されます。
    f. 右側の画面で、 sampleImpl.jar ファイルを選択して**「追加」**をクリックします。このファイルが、「Selected Files (選択済みファイル)」画面に表示されます。
    g. **「OK」**をクリックします。
12. 以下のようにして、Bonus bean のセキュリティーを構成します。
    a. 「EJB Modules (EJB モジュール)」ノードを拡張表示し、**WCSSamplesEntityBeans** ノードを見付けてこれを拡張表示します。

b. **「Entity Beans (エンティティー Bean)」**を拡張表示します。

c. **「Bonus」**を拡張表示します。

d. **「Method Extensions (メソッドの拡張機能)」**をクリックし、右側の画面で以下のようにします。

   1) **「Advanced (拡張)」**タブをクリックします。

   2) **「Security identity (セキュリティー ID)」**が選択されていることを確認します。

   3) 各メソッドごとに、**「 Use identity of EJB server (EJB サーバーの ID を使用する)」**が選択されていることを確認します。

   4) **「適用」**をクリックします (変更を加えた場合)。

e. 左側のナビゲーション画面で、WCSamplesEntityBeans EJB グループの**「Security Roles (セキュリティー役割)」**を右クリックし、**「新規」**を選択して、以下のようにします。

   1) **「名前」**フィールドに WCSecurityRole と入力して、**「適用」**をクリックします。この役割がすでに存在している場合、このステップを実行する必要はありません。

f. 左側のナビゲーション画面で、WCSamplesEntityBeans EJB グループの**「Method Permissions (メソッド許可)」**を右クリックし、**「新規」**を選択して、以下のようにします。

   1) **「Method Permission Name (メソッド許可名)」**フィールドで、WCMethodPermission と入力します。

   2) **「Methods (メソッド)」**選択領域で、**「追加」**をクリックします。「Add Methods (メソッドの追加)」ウィンドウがオープンします。

   3) **sampleEntityBeans.jar**、**「Bonus」**の順に拡張表示してから、メソッドの**「ホーム」**リストと**「リモート」**リストをそれぞれ拡張表示します。

   4) シフト・キーを押したまま、すべてのホーム・メソッドを選択して、**「OK」**をクリックします。

   5) メソッドの選択プロセスを繰り返し、リモート・メソッドを同様にして追加します (リモート・メソッドがある場合)。

   6) 「Roles (役割)」選択領域で、**「追加」**をクリックして、 WCSecurityRole を選択し、**「OK」**をクリックします。

   7) 更新ごとに**「適用」**をクリックします。

13. **「ファイル」**メニューから、**「Save (保管)」**を選択します。

14. アプリケーション・アセンブラー・ツールをクローズします。

このステップが完了したら、以前のすべてのビジネス・ロジックと新しいビジネス・ロジックを含む新しいエンタープライズ・アプリケーションが作成されました。これは、新しく変更した WC_Enterprise_App_*instanceName*.ear ファイルにすべて含まれます。

新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートする

新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートするための大まかなステップは、以下のとおりです。

1. WebSphere Application Server で現在実行されているエンタープライズ・アプリケーションを停止して、それを除去する。これらのステップは WebSphere Application Server 管理コンソールで実行します。
2. コマンド行ユーティリティー XMLConfig を使って新しいアプリケーションをインポートする。
3. WebSphere Application Server 管理コンソールを最新表示にして、新しいエンタープライズ・アプリケーションを開始する。

以下のセクションで、これらのステップについて詳しく説明します。

現在のエンタープライズ・アプリケーションの停止および除去

現在のエンタープライズ・アプリケーションを停止して WebSphere Application Server から除去するには、以下のようにします。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere 管理可能ドメイン」**を拡張表示します。
3. **「ノード」**を拡張表示します。
4. *「nodeName」* **を拡張表示します** (*nodeName* は、ご使用のノードの名前です)。
5. **「Application Servers (アプリケーション・サーバー)」**を拡張表示します。
6. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Server -** *instanceName***」**を右クリックして、**「停止」**を選択します。
7. **「Enterprise Applications (エンタープライズ・アプリケーション)」**を拡張表示します。
8. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Enterprise Application - demo」**アプリケーションを右クリックして、**「停止」**を選択します。
9. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Enterprise Application - demo」**アプリケーションを右クリックして、**「除去」**を選択します。
10. アプリケーションをエクスポートするかどうかの指示を求められたら、**「いいえ」**を選択します。

XMLConfig を使用して新しいエンタープライズ・アプリケーションをインポートする

XMLConfig コマンド行ユーティリティーを使用して新しいエンタープライズ・アプリケーションをインポートするには、以下のようにします。

1. 以下のディレクトリーに移動します。

   drive:¥WebSphere¥CommerceServer¥working

2. コマンド・プロンプトで以下のコマンドを入力して、エンタープライズ・アプリケーションを WebSphere Application Server にインポートします。

   ```
   xmlConfig -import OutputFile.xml -adminNodeName was_hostname
   ```

   ここで、 *was_hostname* は現在のアプリケーションを含んでいる WebSphere Application Server のノード名です。

注: 400 iSeries マシン上で実行する WebSphere Commerce インスタンスをデプロイした場合、アプリケーションをインポートした後で、ディレクトリー許可を変更するために追加のステップを実行する必要があります。これらの許可を変更する方法の詳細は、385 ページの 『エンタープライズ・アプリケーションのインポート』を参照してください。

新しいエンタープライズ・アプリケーションの開始

コマンド行ユーティリティー XMLConfig を使用して新しいエンタープライズ・アプリケーションをインポートした後、 WebSphere Application Server 管理コンソールを使って最新表示を実行し、それから新しいアプリケーションを開始することができます。

コンソールを最新表示して新しいアプリケーションを開始するには、以下のようにします。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere 管理可能ドメイン」**を拡張表示します。
3. **「ノード」**を強調表示します。
4. **「Refresh selected subtree (選択したサブツリーの最新表示)」**アイコンをクリックします。
5. 以下のようにして、WebSphere Commerce アプリケーションを開始します。
   - **「Application Servers (アプリケーション・サーバー)」**を拡張表示します。
   - WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Server -** *instanceName*」を右クリックして、**「開始」**を選択します。

MyNewControllerCmd のテスト

次のステップは、ストア内の新しいロジックを WebSphere Application Server 環境で実行してテストすることです。 MyNewControllerCmd をテストするには、以下のようにします。

1. 以下の URL ブラウザーに入力して、統合をテストします。

   ```
   http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

ここで、*store_Id* はストアの ID で、*catalog_Id* はストアのカタログの ID です。

2. **「登録」**リンクをクリックします。
「登録」または「ログイン」ページが表示されます。
3. **「E メール・アドレス」**フィールドで wcsadmin と入力します。
4. **「パスワード」**フィールドで、このサイトで使用される wcsadmin ID のためのパスワードを入力してから、**「ログイン」**をクリックします。
5. ユーザーがログオンしてから、サイド・ナビゲーション・ペインで **MyNewControllerCmd** リンクをクリックします。サンプル JSP が表示されます。

更新された sidebar.jsp ファイルが表示されない場合は、以下のようにしてキャッシュをクリアします。

1. キャッシュに入れられているファイルを以下のディレクトリーから削除します。
drive:¥WebSphere¥CommerceServer¥instances¥*instanceName*¥cache
2. キャッシュに入れられているすべての関連ファイルを以下のディレクトリーから削除します。

drive:¥WebSphere¥AppServer¥temp¥*hostName*¥
WebSphere_Commerce_Server_*instanceName*¥
WebSphere_Commerce_Enterprise_Application_-_*instanceName*¥
wcstores.war¥*storeName*

drive:¥WebSphere¥AppServer¥temp¥*hostName*¥
WebSphere_Commerce_Server_*instanceName*¥
WebSphere_Commerce_Enterprise_Application_-_*instanceName*¥
wcstores.war

3. ブラウザーのキャッシュをクリアします。

JSP ページのコンパイルに時間がかかりすぎる場合、ページが表示されないことがあります。その場合には、ページを再ロードしてください。

第 10 章 既存のビジネス・ロジックの変更および拡張

以下のチュートリアルでは、既存の WebSphere Commerce ビジネス・ロジックを拡張または変更する方法を示します。

既存のコントローラー・コマンドの拡張

以下で、既存の OrderProcess コントローラー・コマンドを拡張して、購入時のボーナス・ポイントの累計をオーダーの確認ページに表示できるようにします。

注: このチュートリアルの目的は、既存のコントローラー・コマンドを変更するプロセスを示すことです。このチュートリアルは、ショッピング・フローでのオーダー処理ステップを変更するための最善の方法を示すようには作られていません。実際は、WebSphere Commerce が、ショッピング・フローのオーダー処理ステップの変更に使用できる `ExtOrderProcess` タスク・コマンドを提供します。

このチュートリアルを始める前に

207 ページの『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』のステップを完了している必要があります。

以下のリストに、既存の OrderProcess コマンドの拡張に関係するステップを要約しています。

1. カスタマイズ・コードを保管する新しいパッケージを作成します。すべてのカスタマイズ・コード (コマンドおよびデータ Bean の両方) を、 WebSphere Commerce コードとは別のプロジェクトおよびパッケージに保管しなければならないことに注意してください。
2. 既存の `OrderProcessCmdImpl` コマンドを拡張した新しい `OrderProcessCmdBonusImpl` クラスを作成します。
3. フィールドとメソッドを `OrderProcessCmdBonusImpl` クラスに追加します。
4. `OrderProcessCmdBonusImpl` クラスを使用するように、コマンド・レジストリーに変更を加えます。
5. confirmation.jsp テンプレートに変更を加えて、新しいビジネス・ロジックを表示します。
6. WebSphere Test Environment 中で新しいビジネス・ロジックをテストします。
7. (オプション) 新しいビジネス・ロジックをリモート WebSphere Commerce Server のストアにデプロイメントします。

OrderProcessCmdBonusImpl のパッケージの新規作成

OrderProcessCmdBonusImpl コマンドの保管先の新しいパッケージを作成するには、以下のようにします。

1. 「VisualAge for Java ワークベンチ」ウィンドウで、「プロジェクト」タブが選択されていることを確認します。
2. **_WCSamples** プロジェクトを右クリックして、**「追加」**>**「パッケージ」**を選択します。
   「Add Package SmartGuide (パッケージの追加 SmartGuide)」がオープンします。
3. **「名前指定したパッケージの新規作成」**ラジオ・ボタンが使用可能になっていることを確認して、 com.ibm.commerce.sample.order を入力します。
4. **「終了」**をクリックします。

OrderProcessCmdBonusImpl クラスの作成

新しい OrderProcessCmdBonusImpl クラスを作成するには、以下のようにします。

1. com.ibm.commerce.sample.order パッケージを右クリックして、**「追加」**>**「クラス」**を選択します。
   「Create Class SmartGuide (クラスの作成 SmartGuide)」がオープンします。
2. **「新規クラスの作成」**ラジオ・ボタンが選択されていることを確認します。
3. **「クラス名」**フィールドに、 OrderProcessCmdBonusImpl と入力します。
4. スーパークラスを指定するために、**「ブラウズ」**をクリックし、**「パターン」**フィールドに com.ibm.commerce.order.commands.OrderProcessCmdImpl と入力して、**「OK」**をクリックします。
5. **「次へ」**をクリックします。
6. **「パッケージの追加」**をクリックして、インポートするパッケージを指定します。**「パターン」**フィールドに、以下のパッケージを入力します。
   - com.ibm.commerce.datatype と入力して**「追加」**をクリックする
   - com.ibm.commerce.exception と入力して**「追加」**をクリックする
   - com.ibm.commerce.order.commands と入力して**「追加」**をクリックする
   - com.ibm.commerce.order.objects と入力して**「追加」**をクリックする
   - com.ibm.commerce.ras と入力して**「追加」**をクリックする
   - com.ibm.commerce.server と入力して**「追加」**をクリックする
   - com.ibm.commerce.sample.objects と入力して**「追加」** をクリックする
   - javax.ejb と入力して**「追加」**をクリックする
   - java.io と入力して**「追加」**をクリックする
   - java.math と入力して、**「追加」**に続いて**「クローズ」**をクリックする。
7. クラスがインプリメントするインターフェースを指定するために、**「追加」**をクリックします。「パターン」フィールドに、以下のインターフェースを入力します。

- OrderProcessCmd と入力して、**「追加」**に続いて**「クローズ」**をクリックする。

8. **「終了」**をクリックします。

フィールドおよびメソッドの OrderProcessCmdBonusImpl への追加

2 つのフィールドと performExecute メソッドをクラスに追加しなければなりません。

theOrder フィールドを OrderProcessCmdBonusImpl クラスに追加するには、以下のようにします。

1. OrderProcessCmdBonusImpl クラスを右クリックし、**「追加」**>**「フィールド」**を選択します。
   「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。
2. 以下の属性を使用して、フィールドをクラスに追加します。新規フィールドを作成する詳しい方法については、220 ページの 『新規フィールドの作成』を参照してください。

属性名	値
フィールド名	theOrder
フィールド・タイプ	OrderAccessBean
初期値	ブランクのままにします。
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

それから**「終了」**をクリックします。

bonusPercentAmount フィールドを OrderProcessCmdBonusImpl クラスに追加するには、以下のようにします。

1. OrderProcessCmdBonusImpl クラスを右クリックして、**「追加」**>**「フィールド」**を再度選択します。
2. 以下の属性を使用して、フィールドをクラスに追加します。新規フィールドを作成する詳しい方法については、220 ページの 『新規フィールドの作成』を参照してください。

属性名	値
フィールド名	bonusPercentAmount
フィールド・タイプ	double

属性名	値
初期値	1000
アクセス修飾子	private
その他の修飾子	すべて未チェックのままにします。
getter および setter メソッドを使ったアクセス	チェックする
Getter	public
Setter	public

それから**「終了」**をクリックします。

performExecute メソッドを OrderProcessCmdBonusImpl クラスに追加するには、以下のようにします。

1. **OrderProcessCmdBonusImpl** クラスを右クリックし、**「追加」**>**「メソッド」**を選択します。
   「Method SmartGuide (メソッドの作成 SmartGuide)」がオープンします。
2. **「新規メソッドの作成」**ラジオ・ボタンが選択されていることを確認して、**「次へ」**をクリックします。
3. **「メソッド名」**フィールドに、 performExecute と入力します。
4. **「戻りタイプ」**ドロップダウン・リストから、 void を選択します。**「次へ」**をクリックします。
5. メソッドから出される例外を指定するために、**「追加」**を選択します。**「パターン」**フィールドに ECException と入力し、**「追加」**に続いて**「クローズ」**をクリックします。
6. **「終了」**をクリックします。
   メソッドが生成され、そのメソッドのソース・コードが表示されます。
7. ソース・コードに変更を加えなければなりません。 performExecute メソッドのソース・コードに以下の行を挿入します。

```
public void performExecute() throws
   com.ibm.commerce.exception.ECException {
```

上記の行の後に以下のコードを入力します。

PDF 版のプログラマーズ・ガイドから、このコードを切り貼りすることができます。まず VisualAge for Java の「Scrapbook」ウィンドウにコードをコピーして (詳しくは、VisualAge for Java のオンライン・ヘルプを参照)、切り貼り操作のときに文字が失われなかったかどうか検査することを推奨します。コードを検査した後、宛先にコピーしてください。別のエディターにテキストをコピーすると、変更される文字があることに注意してください。

```
final String methodName = "performExecute";
ECTrace.entry(ECTraceIdentifiers.COMPONENT_ORDER,
   this.getClass().toString(), methodName);

// do all order processing as normal
super.performExecute();

// *** updating Bonus Point Information ***

// fetch order info
theOrder = new OrderAccessBean();
theOrder.setInitKey_orderId(getOrderRn().toString());

int bonusPt;  // bonus points for this order
int bonusTotal;  // total bonus points
BigDecimal subtotal;  // subtotal
BigDecimal bonusdeter;  // bonus determinant
BigDecimal ans;

// determine bonus points = subtotal * bonus determinant
try {
     subtotal = theOrder.getTotalProductPriceInEJBType();
     bonusdeter = new BigDecimal(bonusPercentAmount);
     ans = subtotal.multiply(bonusdeter);
     bonusPt = Math.round(ans.floatValue());

     System.out.println("subtotal is: " + subtotal +
        " bonus deter is: " + bonusdeter + " ans is: " + ans);
     System.out.println("Bonus Percent amount = " +
        bonusPercentAmount);
     System.out.println("Bonus calculated is: "+ bonusPt);
    }


   // Various Exceptions
   catch (Exception ex) {
   throw new ECSystemException(ECMessage._ERR_GENERIC,
      this.getClass().toString(),methodName,
      ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
   }

// *** Updating bonus points in BONUS table using bean
//     created in previous example ***

BonusAccessBean bonusBean = new BonusAccessBean();
bonusBean.setInit_argMemberId(
   getCommandContext().getUserId().toString());
try {
    //new bonus value = this order bonus points + Old Bonus Points
    bonusTotal = bonusPt + Integer.parseInt(bonusBean.getBonusPoint());
    bonusBean.setBonusPoint(String.valueOf(bonusTotal));
    bonusBean.commitCopyHelper();

    System.out.println("In try, BonusTotal calculated is: "+
```

```
                bonusTotal);

            }
        // Various exceptions
        catch (FinderException e) // user does not have points setup yet
        {
           // create a row in table bonus
           bonusTotal = bonusPt;
           try {
               BonusAccessBean bonusBeanNew = new
                  BonusAccessBean(getCommandContext().getUserId(),
                     new Integer(bonusTotal));

                  System.out.println("In catch, BonusTotal calculated is: "+
                     bonusTotal);

        }
        catch (Exception ex) {
           throw new ECSystemException(ECMessage._ERR_GENERIC,
              this.getClass().toString(), methodName,
              ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
           }
     }
        catch (Exception ex) {
            throw new ECSystemException(ECMessage._ERR_GENERIC,
              this.getClass().toString(), methodName,
              ECMessageHelper.generateMsgParms(ex.getMessage()), ex);
        }
        // *** setting view details ***

        // Fetch setResponse properties and add bonus parameters
       //  needed by the JSP page
        TypedProperty resp = getResponseProperties();
        resp.put("bonus", new Integer(bonusPt).toString());
        setResponseProperties(resp);
        ECTrace.exit(ECTraceIdentifiers.COMPONENT_ORDER,
           this.getClass().toString(), methodName);
```

8. 作業内容を保管します。

OrderProcessCmdBonusImpl を使用するようにコマンド・レジストリーに変更を加える

この例で、オーダー処理が必要な場合に必ず新しいオーダー処理用インプリメンテーション・クラスを使用したいとします。そのためには、コマンド・レジストリーを更新して、オリジナルの OrderProcess インターフェースと新しい OrderProcessCmdBonusImpl インプリメンテーション・クラスを関連付けなければなりません。

DB2 DB2 データベースを使用している場合は、以下のようにしてコマンド・レジストリーを更新してください。

1. DB2 コマンド・センターをオープンします (**「スタート」**>**「プログラム」**>**「IBM DB2」**>**「コマンド・センター」**)。

2. スクリプト・ウィンドウの「スクリプト」タブを選択し、スクリプト・ウィンドウで以下の情報を入力することによって、 CMDREG テーブルに必要なエントリーを作成します。

```
connect to your_database_name;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0;
```

ここで、*your_database_name* はデータベースの名前です。続いて「実行」アイコンをクリックします。
このコマンドはすべてのマーチャントによって使われます (STOREENT_ID の 0 の値がそれを示しています)。

Oracle Oracle データベースを使用している場合は、以下のようにしてコマンド・レジストリーを更新してください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。
2. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。
3. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで以下の情報を入力することによって、 URLREG テーブルに必要なエントリーを作成します。

```
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0;
```

この SQL ステートメントを実行するには Enter を押します。
このコマンドはすべてのマーチャントによって使われます (STOREENT_ID の 0 の値がそれを示しています)。

6. データベースの変更をコミットするには

```
commit;
```

と入力し、それから Enter を押して SQL ステートメントを実行します。

confirmation.jsp テンプレートの変更

confirmation.jsp テンプレートに変更を加えて、オーダー処理ビジネス・プロセスに追加した新しいビジネス・ロジックを表示しなければなりません。表示テンプレートに変更を加えるには、以下のようにします。

1. 以下のディレクトリーに移動します。

   ```
   vaj_drive:¥VAJava¥ide¥project_resources¥IBM WebSphere Test
   Environment¥hosts¥default_host¥default_app¥web¥store_directory
   ```

2. confirmation.jsp のコピーを作成し、それを confirmation.jsp.bak と呼びます。
3. テキスト・エディターで confirmation.jsp ファイルをオープンします。
4. 既存の import ステートメントの後に、以下を追加します。

   ```
   <%@ page import="com.ibm.commerce.datatype.*"   %>
   ```

5. JSP テンプレートの以下の行の直後に、

   ```
   String orderRn = jhelper.getParameter("orderId");
   ```

   以下を追加します。

   ```
   String bonus = ((TypedProperty)request.getAttribute(
      ECConstants.EC_REQUESTPROPERTIES)).getString("bonus");
   ```

6. JSP テンプレート内で以下のセクションを見付けます。

   ```
   <tr>
   <td align="left" valign="middle">
   <font class="product"><%=infashiontext.getString("GRAND_TOTAL")%>
      </font></td>
   <td align="right" valign="middle">
   <font class="strongprice"><%=orderBean.getGrandTotal()  %></font></td>
   ```

   それから以下を追加します。

   ```
   </tr>
   <tr>
   <td align="left" valign="middle">
   <font class="text">Bonus Points</font></td>
   <td align="right" valign="middle">
   <font class="strongtext"><%=bonus  %></font></td>
   ```

7. 作業内容を保管します。

WebSphere Test Environment 中での OrderProcessCmdBonusImpl のテスト

この時点で、WebSphere Test Environment を使用して新しいビジネス・ロジックをテストできます。 OrderProcessCmdBonusImpl コマンドをテストするには、以下のようにします。

1. WebSphere Test Environment を開始します (349 ページの『付録 A. WebSphere Test Environment の開始と停止』を参照)。
2. ブラウザーをオープンし、ストアの URL を入力します。たとえば、以下の URL を入力します。

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=10001&catalogId=10001&langId=-1
   ```

3. 商品を選択して購入します。

4. 商品を購入し終えると、オーダーの確認に、オーダーの際に獲得したボーナス・ポイントの数が表示されます。

(オプション) カスタマイズされたビジネス・ロジックのリモート WebSphere Commerce Server へのデプロイメント

WebSphere Test Environment 中でビジネス・ロジックのテストを完了して、コードが満足のいく状態になった後、そのコードをリモート WebSphere Commerce Server のストアにデプロイメントすることができます。このチュートリアルでは、カスタマイズには以下のものが含まれます。

- 新規の OrderProcessCmdBonusImpl クラス
- 更新された confirmation.jsp テンプレート・ファイル
- 更新されたコマンド・レジストリー

したがって、コードのデプロイメントに含まれるステップは以下のとおりです。

1. VisualAge for Java 中のツールを使用して、コマンド・ロジック用の JAR ファイルを作成します。
2. JAR ファイルおよび JSP テンプレートを、ターゲットの WebSphere Commerce Server 上の該当するディレクトリーにコピーします。
3. ターゲットの WebSphere Commerce Server 上のコマンド・レジストリーを更新します。

テスト支払いメソッドについての注意

WebSphere Test Environment 内で実行しているサンプル・ストアは、デフォルトでテスト支払いメソッドを使用します。このテスト支払いメソッドが使用されるため、ユーザーは Payment Manager の呼び出しを必要とせずに WebSphere Test Environment 内でのショッピング・フローを完了できます。このテスト支払いメソッドは、1 回の購入を完了させるだけであり、この支払いメソッドで送信されたオーダーは今後の処理で使用可能にできません。したがって、テスト支払いメソッドは WebSphere Test Environment 内でのみ使用してください。

このカスタマイズ・コードをデプロイしているストアで購入を完了させてください。支払い処理は、ローカル Payment Manager かリモート Payment Manager を使って行えます。

テスト支払いメソッドについての詳細は、203 ページの 『テスト支払いメソッド』を参照してください。

コマンド・ロジック用の JAR ファイルの作成

コマンド・ロジックを JAR ファイルにパッケージして、ターゲットの WebSphere Commerce Server にデプロイメントできるようにしなければなりません。`OrderProcessCmdBonusImpl` は _WCSamples プロジェクトに保管されているので、このプロジェクト用の JAR ファイルを作成します。

JAR ファイルを作成するには、以下のようにします。

1. WebSphere Test Environment を停止します (349 ページの『付録 A. WebSphere Test Environment の開始と停止』を参照)。
2. **_WCSamples** プロジェクトを右クリックして、**「エクスポート」**を選択します。「Export SmartGuide (エクスポート SmartGuide)」がオープンします。
3. **「JAR ファイル」**を選択し、**「次へ」**をクリックします。
4. 「JAR ファイル」フィールドで、以下のように入力します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp_b¥wcssamplesb_1.jar
   drive は Commerce Studio のインストール先のドライブです。
5. 属性を以下のように選択します。

属性	値
クラス	チェックする
Java	チェックしない
リソース	チェックする
bean	チェックする
.class ファイルにデバッグ属性を組み込む	チェックする

   その他の属性については、デフォルト値を受け入れます。
6. **「終了」**をクリックします。

作成された JAR ファイルには完全なパッケージ命名情報が含まれていないため、(VisualAge for Java 以外の) 別のパッケージ・ユーティリティーを使用して JAR ファイルを再パッケージ化する必要があります。このファイルを再パッケージ化するには、以下のようにします。

1. コマンド・ウィンドウで、以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp_b
2. `mkdir temp1` と入力します。
3. `cd temp1` と入力します。
4. パスを以下のように設定します。
   `set PATH=%PATH%;`*drive*`:¥WebSphere¥WebSphereStudio4¥bin;`
   drive は、WebSphere Studio のインストール先のドライブです。
5. `jar xvf ../wcssamplesb_1.jar` と入力します。
6. `jar cvf ../wcssamplesb.jar *` と入力します。

ターゲットの WebSphere Commerce Server への資産の保管

コマンド・ロジック用の JAR ファイルと変更済みの confirmation.jsp テンプレートを、ターゲットの WebSphere Commerce Server 上の該当するディレクトリーに入れなければなりません。

リモート WebSphere Commerce Server 上の該当するディレクトリーに JAR ファイルを保管するには、以下のようにします。

1. 開発マシンで、コマンド・ウィンドウをオープンしてから以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp_b
   wcssamplesb.jar ファイルを見付けます。
2. このファイルをターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。

   drive:¥WebSphere¥AppServer¥installedApps¥
   WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥WEB-INF¥lib

ターゲットの WebSphere Commerce Server 上の適切なディレクトリーに confirmation.jsp テンプレートを保管するには、以下のようにします。

1. 開発マシンで、以下のディレクトリーに移動します。
   vaj_drive:¥VAJava¥ide¥project_resources¥IBM WebSphere Test Environment¥hosts¥default_host¥default_app¥web¥*store_directory*
2. ターゲットの WebSphere Commerce Server 上の以下のディレクトリーに、confirmation.jsp テンプレートをコピーします。

   drive:¥WebSphere¥AppServer¥installedApps¥
   WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥*storeDir*

コマンド・レジストリーの更新

OrderProcessCmdBonusImpl コマンドを、 WebSphere Test Environment で使用されていないデータベースを使用しているターゲットの WebSphere Commerce Server にデプロイメントする場合は、ターゲット・データベースを更新して、コマンド・レジストリーに加えられた変更内容を反映させなければなりません。

DB2 DB2 データベースを使用している場合は、以下のようにしてターゲットの WebSphere Commerce Server のデータベースを更新してください。

1. DB2 コマンド・センターをオープンします (**「スタート」**>**「プログラム」**>**「IBM DB2」**>**「コマンド・センター」**)。
2. スクリプト・ウィンドウの「スクリプト」タブを選択し、スクリプト・ウィンドウで以下の情報を入力することによって、 CMDREG テーブルに必要なエントリーを作成します。

```
connect to your_target_database_name;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
and storeent_Id=0
```

ここで、*your_target_database_name* はご使用のデータベースの名前です。「実行」アイコンをクリックします。

Oracle Oracle データベースを使用している場合は、以下のようにしてコマンド・レジストリーを更新してください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。
2. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。
3. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで以下の情報を入力することによって、 CMDREG テーブルに必要なエントリーを作成します。

   ```
   update CMDREG
   set CLASSNAME='com.ibm.commerce.sample.order.OrderProcessCmdBonusImpl'
   WHERE INTERFACENAME='com.ibm.commerce.order.commands.OrderProcessCmd'
   and storeent_Id=0;
   ```

   この SQL ステートメントを実行するには Enter を押します。
   このコマンドはすべてのマーチャントによって使われます (STOREENT_ID の 0 の値がそれを示しています)。
6. データベースの変更をコミットするには

   ```
   commit;
   ```

   と入力し、それから Enter を押して SQL ステートメントを実行します。

WebSphere Application Server でのエンタープライズ・アプリケーションの再始動

ファイル資産を適切なディレクトリーに入れ、コマンド・レジストリーを更新してコマンド・ロジックをエンタープライズ・アプリケーションに追加した後、変更内容を有効にするために、エンタープライズ・アプリケーションを停止して再始動する必要があります。

エンタープライズ・アプリケーションを停止して再始動するには、以下のようにします。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere 管理可能ドメイン」**を拡張表示します。

3. **「ノード」**を拡張表示します。
4. ***「nodeName」*を拡張表示します** (***nodeName*** は、ご使用のノードの名前です)。
5. **「Application Servers (アプリケーション・サーバー)」**を拡張表示します。
6. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Server - demo」**アプリケーションを右クリックして、**「停止」**を選択します。
7. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Server - demo」**アプリケーションを右クリックして、**「開始」**を選択します。

WebSphere Application Server 中で実行している InFashion 中の新規ロジックのテスト

この時点で、WebSphere Application Server 中で実行する InFashion ストアの新しいビジネス・ロジックを検証できます。

最終検査を実行するには、以下のようにします。

1. WebSphere Commerce Server インスタンスを開始したら、ブラウザーをオープンし、ストアのホーム・ページの URL を入力します。たとえば、以下の URL を入力します。

   ```
   http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

   ここで、*store_Id* はストアの ID で、*catalog_Id* はストアのカタログの ID です。
2. 商品を選択して購入します。
3. 商品を購入し終えると、オーダーの確認に、オーダーの際に獲得したボーナス・ポイントの数が表示されます。

既存のエンティティー Bean 変更と既存のタスク・コマンドの拡張

注: このチュートリアルの目的は、既存のエンティティー Bean の変更、既存のタスク・コマンドの拡張を行うプロセスを示すことです。商品価格を変更するための最良の方法を示すようには設計されていません。価格の割引についての情報は、*WebSphere Commerce 計算フレームワーク・ガイド* を参照してください。

『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』では、まったく新しいビジネス・ロジックのセットを作成しました。その作業には、新しいコントローラー・コマンド、新しい JSP テンプレート、新しいデータベース・テーブル、そのテーブルにアクセスするための新しいエンタープライズ Bean、それに対応するアクセス Bean、データ・アクセス Bean の作成が含まれています。このすべてのロジックを組み合わせてできるシンプルなボーナス・ポイント・アプリケーションでは、新しいコントローラー・

コマンドによって起動される JSP テンプレートを使用して、ユーザーのボーナス・ポイントのバランスを更新することができます。

『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』で説明された BONUS テーブルの使用方法が、以下の図に示されています。

「ビジネス・ロジックの新規作成」チュートリアルでの BONUS テーブルの使用

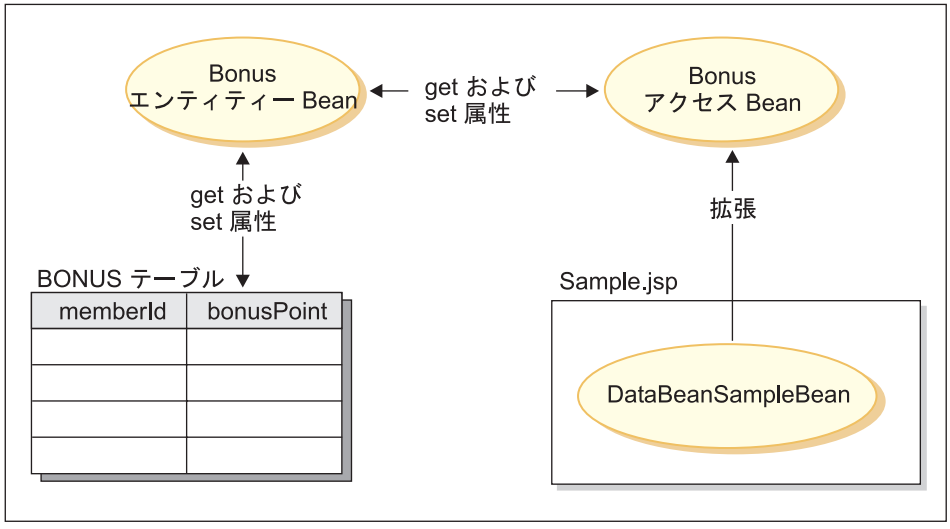


図 41.

次のチュートリアルでは、BONUS テーブルを別の方法で使用します。具体的には、User エンティティー Bean をカスタマイズして、アプリケーションからは、 BONUS テーブルの BONUSPOINT 列が USERS テーブルの実際の列として表示されるようにします。 USERS テーブルに新しいレコードを作成すると、対応するレコードが自動的に BONUS テーブルに挿入されます。

このテーブル結合 を作成するには、新しい CMP フィールドを User エンティティー Bean に追加する必要があります。この CMP フィールドは、VisualAge for Java マップ・ブラウザーの 2 次テーブル・マップ機能を使用して、 BONUS テーブルの BONUSPOINT 列にマッピングします。

変更済みの User エンティティー Bean を購入のフローの中に組み入れるためには、商品の新しい価格を作成します。この新しい価格は、ショッパーのボーナス・ポイントの現在のバランスを考慮に入れます。新しい価格を作成するには、GetProductContractUnitPriceCmd タスク・コマンドを拡張します。このコマンドを拡張するときには、GetProductContractUnitPriceCmd インターフェースを拡張した新しいインターフェースを作成します。その新しいインターフェースによって、追加の属性 (ボーナス価格の属性) が追加されます。さらに、GetContractUnitPriceCmdImpl インプリメンテーション・クラスを拡張した新しいインプリメンテーション・クラスも作成します。こ

の新しいインプリメンテーション・クラスは、“GetNewContractUnitPriceCmdImpl” と呼ばれます。このインプリメンテーション・クラスは、そのスーパークラスの performExecute メソッドを呼び出してから、新しいボーナス価格を決定するためのビジネス・ロジックを追加します。

以下の図は、次のチュートリアルで BONUS テーブルを使用する方法を示したものです。

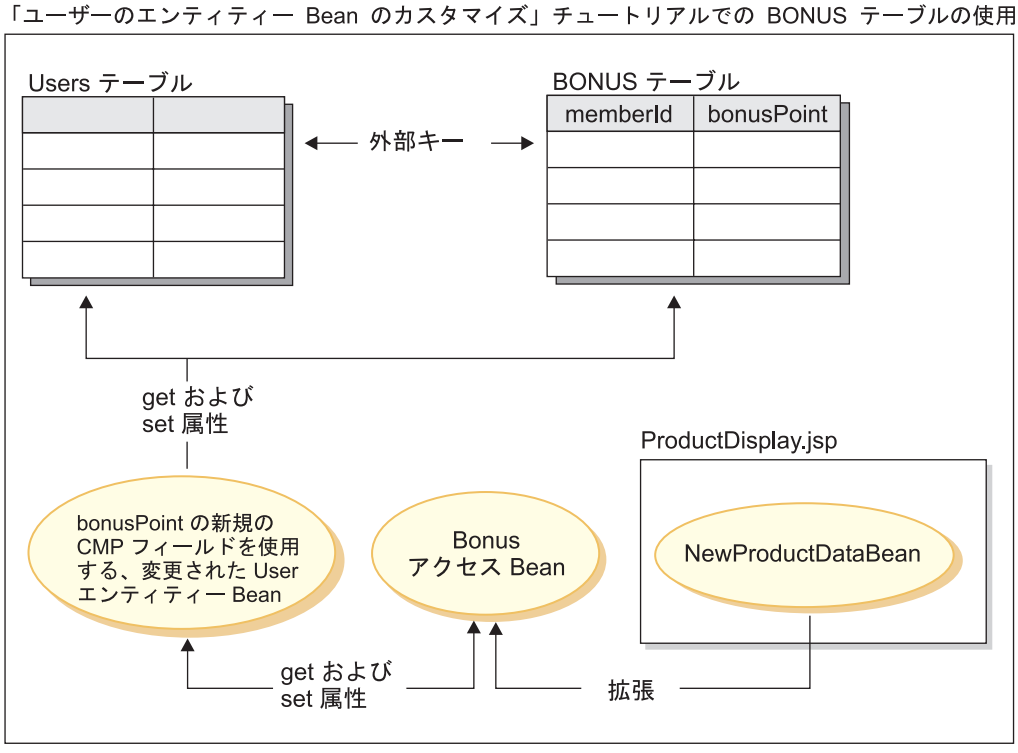


図 42.

このチュートリアルに含まれるステップは以下のとおりです。

1. 新しい CMP フィールドを User エンティティー Bean に追加します。
2. BONUS テーブルを作成して、値を取り込みます。
3. WCSUser データベース・スキーマとテーブル・マッピングを更新して、新しい BONUS テーブルを組み込みます。
4. BONUS テーブルと USER テーブルの間の外部鍵関連を作成します。
5. BONUS テーブル・マップを作成します。
6. User エンティティー Bean 用のデプロイメント・コードとアクセス Bean を生成します。

7. 更新済みのエンティティー Bean の予備検査用のテスト・クライアントを使用します。
8. 新しいタスク・コマンドのインターフェースとインプリメンテーション・クラスを作成します。新しいタスク・コマンドは、GetBaseUnitProductPriceCmd を拡張したものです。このコマンドの最も重要な新機能は、インプリメンテーション・クラスの performExecute() メソッドのロジックです。このメソッドは、商品の新しいボーナス*価格* を計算します。このボーナス価格は、計算価格の 20% を上限として、購買者のボーナス・ポイント・バランスに基づいて割引をした価格です。以下の表は、この割引価格の計算例を示したものです。

計算価格	ボーナス・ポイント・バランス	最大割引額 (計算価格の 20%)	新しいボーナス価格
$1000	100	$200	$900 ボーナス・ポイント・バランスが最大割引額よりも小さいので、表示価格からボーナス・ポイントを差し引いた額になります。
$1000	300	$200	$800 ボーナス・ポイント・バランスが最大割引額よりも大きいので、表示価格から最大割引額 $200 を差し引いた額になります。

9. ProductDataBean を拡張した NewProductDataBean を作成します。新しいボーナス価格を商品の表示ページで簡単に使用するための新しいメソッドをこの bean に追加します。
10. ボーナス価格を表示するために、 InFashion ストアの商品表示 JSP テンプレートを更新します。
11. WebSphere Test Environment 内で実行する InFashion ストアのビジネス・ロジックをテストします。
12. (オプション) 更新済みのビジネス・ロジックのリモート WebSphere Commerce Server へのデプロイメントを実行して、 WebSphere Test Environment の外でテストします。

このチュートリアルを始める前に

207 ページの『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』がまだ終わっていない場合は、このチュートリアルを始める前に、 208 ページの 『サンプル・プロジェクトの準備』で説明されているステップを実行する必要があります。

新しい bonusPoint フィールドの User エンティティー Bean への追加

このセクションでは、 VisualAge for Java の EJB ツールを使用して、エンティティー Bean に新しい CMP フィールドを追加します。この新しいフィールドは *bonusPoint* であり、最終的には BONUS テーブルの BONUSPOINT 列にマッピングされます。

User エンティティー Bean に新しい CMP フィールドを追加するには、以下のようにします。

1. サーブレット・エンジン、EJB サーバー、永続ネーム・サーバーが実行中であれば、 349 ページの『付録 A. WebSphere Test Environment の開始と停止』で説明されているように停止します。
2. ワークベンチで、**「EJB」**タブをクリックします。
3. **WCSUser** EJB グループを拡張表示します。
4. **User** bean を右クリックして、**「追加」**>**「CMP フィールド」**を選択します。「Create CMP Field SmartGuide (CMP フィールドの作成 SmartGuide)」がオープンします。
5. 以下のようにプロパティーを設定して、新しい CMP フィールドを作成します。

プロパティー	値
フィールド名	bonusPoint
フィールド・タイプ	int
初期値	0
getter および setter メソッドを使ったアクセス	enable
getter および setter メソッドをリモート・インターフェースにプロモートする	enable
Getter	public
Setter	public

それから**「終了」**をクリックします。

「プロパティー」ペインに bonusPoint フィールドが表示されます。いくつかの警告が表示される場合もありますが、エンティティー Bean のコードを再生成する時点で問題は解決されます。

BONUS テーブルの作成と値の取り込み

このチュートリアルでは、ユーザーのボーナス・ポイントを記録するテーブルを使用します。

207 ページの『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』を終了している場合は、このテーブルについてすでに習熟しているでしょう。その場合は、このテーブルを更新して、 USERS テーブル中のユーザーごとに行を追加しなければなりません。 207 ページの『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』をまだ終了していない場合は、このテーブルを作成して取り込まなければなりません。両方のシナリオごとの指示が用意されています。

DB2 DB2 データベースを使用している際に、 BONUS テーブルを*更新する* 必要がある場合は、以下のようにしてください。

1. DB2 コマンド・センターをオープンして (**「スタート」** > **「プログラム」** > **「IBM DB2」** > **「コマンド・センター」**)、**「スクリプト」**タブをクリックします。
2. **「コマンド」**フィールドに、以下のように入力します。

   ```
   connect to your_database_name
   ```

   ここで、*your_database_name* はご使用のデータベースの名前です。「実行」アイコンをクリックします。
3. **「コマンド」**フィールドに以下のように入力して、「実行」アイコンをクリックします。

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS))
   ```

   BONUS テーブルが更新されました。

DB2 DB2 データベースを使用している際に、 BONUS テーブルを*作成して取り込む* 必要がある場合は、以下のようにしてください。

1. DB2 コマンド・センターをオープンして (**「スタート」** > **「プログラム」** > **「IBM DB2」** > **「コマンド・センター」**)、**「スクリプト」**タブをクリックします。
2. **「コマンド」**フィールドに、以下のように入力します。

   ```
   connect to your_database_name
   ```

ここで、*your_database_name* はご使用のデータベースの名前です。「実行」アイコンをクリックします。

3. **「コマンド」**フィールドに以下のように入力して、「実行」アイコンをクリックします。

```
CREATE TABLE BONUS (MEMBERID BIGINT NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade)
```

BONUS テーブルが作成されました。

4. このテーブルに値を取り込むには、**「コマンド」**フィールドに以下のように入力して、「実行」アイコンをクリックします。

```
insert into BONUS (select USERS_ID, 0 from USERS)
```

5. DB2 コマンド・センターをクローズします。

Oracle Oracle データベースを使用している際に、 BONUS テーブルを*更新する* 必要がある場合は、以下のようにしてください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」**> **「プログラム」**> **「Oracle」**> **「アプリケーション開発」**> **「SQL Plus」**)。
2. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。
3. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで以下の情報を入力することによって、 BONUS テーブルを更新します。

```
INSERT INTO BONUS
(SELECT USERS_ID, 0
FROM USERS
WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS));
```

Enter を押して SQL ステートメントを実行します。
これで BONUS テーブルが更新されました。BONUS テーブルが更新されました。

6. データベースの変更をコミットするには

```
commit;
```

と入力し、それから Enter を押して SQL ステートメントを実行します。

Oracle Oracle データベースを使用している際に、 BONUS テーブルを*作成して取り込む* 必要がある場合は、以下のようにしてください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」**> **「プログラム」**> **「Oracle」**> **「アプリケーション開発」**> **「SQL Plus」**)。
2. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。

3. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで以下の情報を入力することによって、 BONUS テーブルを作成します。

```
CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
    constraint f_memberid foreign key (MEMBERID)
    references users (users_id) on delete cascade);
```

   Enter を押して SQL ステートメントを実行します。
   これで BONUS テーブルが作成されました。
6. 以下を入力することによって、BONUS テーブルを取り込みます。

```
insert into BONUS (select USERS_ID, 0 from USERS);
```

7. データベースの変更をコミットするには

```
commit;
```

   と入力し、それから Enter を押して SQL ステートメントを実行します。

スキーマとテーブル・マッピングの更新

ここからは、新しい BONUS テーブルによって WCSUser スキーマを更新し、その新しいテーブル用の外部鍵関係を作成し、 User エンティティー Bean のフィールドと BONUS テーブルの列との間のテーブル・マップを作成します。

BONUS テーブル・スキーマの作成

テーブル・スキーマを作成するには、以下のようにします。

1. **User** エンティティー Bean を右クリックして、**「オープン」**>**「データベース・スキーマ」**を選択します。
   「スキーマ・ブラウザー」ウィンドウがオープンします。
2. **「スキーマ」**リストで、**「WCS ユーザー」**を選択します。
3. **「テーブル」**メニューで、**「新規テーブル」**を選択します。
   テーブル・エディターがオープンします。
4. **「名前」**フィールドで、 BONUS と入力します。
5. **「修飾子」**フィールドと**「物理名」**フィールドはブランクにしておきます。
6. 「テーブル列」セクションで、**「新規」**をクリックします。列エディターがオープンします。列を以下のように作成します。

属性	値
名前	memberId
物理名	このフィールドはブランクのままにします。

属性	値
タイプ	BIGINT
タイプの詳細情報	VapConverter
ヌルの許容	チェックしない

それから**「OK」**をクリックします。

7. 「テーブル列」リストで、 **memberId** を選択し、**「>>」**をクリックして、この列を基本キーとして使用します。
8. 別の列 (**「新規」**を再びクリック) を以下のように作成します。

属性	値
名前	bonusPoint
物理名	このフィールドはブランクのままにします。
タイプ	INTEGER
タイプの詳細情報	VapConverter
ヌルの許容	チェックする

それから**「OK」**をクリックします。

9. テーブル・エディターで、**「OK」**をクリックします。
しばらくして、「スキーマ・ブラウザー」ウィンドウの**「テーブル」**リストに、**BONUS** が表示されます。
10. 念のために、**「テーブル」**リストの **BONUS** をクリックして、その 2 つの列が**「列」**リストに表示されることを確認してください。

外部鍵関係の作成

このセクションでは、BONUS テーブルと USERS テーブルの間に外部鍵関係を確立します。

外部鍵関係を作成するには、以下のようにします。

1. 「スキーマ・ブラウザー」ウィンドウで、**「WCS ユーザー」**スキーマをクリックします。
このスキーマのテーブルと外部鍵関係が表示されます。
2. **「外部鍵」**メニューで、**「新規外部鍵関係」**を選択します。
外部鍵関係エディターがオープンします。
3. **「名前」**フィールドで、 F_User_Bonus と入力します。
4. **「データベースに制約が存在する」**チェック・ボックスがチェックされていることを確認してください。
5. **「基本キー・テーブル」**ドロップダウン・リストで、 USERS を選択します。

6. **「外部鍵テーブル」**ドロップダウン・リストで、 BONUS を選択します。
7. **「外部鍵」**ヘッダーの下の空のフィールドをクリックして、ドロップダウン・リストを表示します。そのリストで、 **memberId** を選択して、**「OK」**をクリックします。
   外部鍵関係のリストに、 F_User_Bonus が表示されます。
8. **「スキーマ」**メニューから、**「スキーマの保管」**を選択して、**「終了」**をクリックします。
9. スキーマ・ブラウザーをクローズします。

BONUS テーブル・マップの作成

このセクションでは、 BONUS テーブルの BONUSPOINT 列と User エンティティー Bean の bonusPoint フィールドとの間にマッピングを作成します。

BONUS テーブル・マップを作成するには、以下のようにします。

1. ワークベンチがオープンしており、「EJB」タブが選択されていることを確認します。
2. **「EJB」**メニューで、**「オープン」**>**「スキーマ・マップ」**を選択します。
   マップ・ブラウザーがオープンします。
3. 「データ・ストア・マップ」リストで、**「WCS ユーザー」**を選択します。
4. **「永続クラス」**リストで、以下のようにします。
   a. **「メンバー」**をダブルクリックします。
   b. **「ユーザー」**を選択します (ステップ 4a で「メンバー」を拡張表示した後でないと、**「ユーザー」**は**「メンバー」**の下に表示されないことに注意してください)。
5. **「テーブル・マップ」**メニューで、**「新規テーブル・マップ」**>**「2 次テーブル・マップの追加」**を選択します。
   「2 次テーブル・マップ」ウィンドウがオープンします。
6. **「テーブル」**ドロップダウン・リストで、 **BONUS** を選択します。
7. 「外部鍵関係」ドロップダウン・リストで、 **F_User_Bonus** を選択して、**「OK」**をクリックします。
   しばらくして、「テーブル・マップ」リストに、「BONUS (2 次)」マップが表示されます。
8. **「BONUS (2 次)」**テーブル・マップを強調表示して右クリックし、**「プロパティー・マップの編集」**を選択します。
   プロパティー・マップ・エディターがオープンします。
9. bonusPoint クラス属性の行までスクロールダウンします。 **bonusPoint** を選択します。
10. **「マップ・タイプ」**列の対応する項目をクリックして、ドロップダウン・リストで**「シンプル」**を選択します。

11. **「テーブル列」**列の対応する項目をクリックして、ドロップダウン・リストで **bonusPoint** を選択します。
12. その他のフィールドはすべてそのままにして、**「OK」**をクリックします。
    新しいプロパティー・マップが生成されます。しばらくすると、

    ```
    (a) bonusPoint (bonusPoint)
    ```

    が、マップ・ブラウザーの**「プロパティー・マップ」**列に表示されます。
13. **「WCS ユーザー」**データ・ストア・マップを右クリックし、**「データ・ストア・マップの保管」**を選択して、**「終了」**をクリックします。
14. マップ・ブラウザーをクローズします。

この時点で、User bean には、一部の抽象クラスがインプリメントされていないことを示すエラーが含まれます。これらのエラーは、デプロイメント・コードが生成されるときに修正されます。

デプロイメント・コードとアクセス Bean の生成

User エンティティー Bean のコードを変更したので、そのデプロイメント・コードとアクセス Bean を再生成する必要があります。 VisualAge for Java のツールを使えば、このコード生成ステップが非常に簡単になります。

このステップを実行するには、以下のようにします。

1. ワークベンチがオープンしており、「EJB」タブが選択されていることを確認します。
2. **WCSUser** EJB グループを拡張表示します。
3. **User** bean を右クリックして、**「デプロイメント・コードの生成」**を選択します。
   パッケージの版を作成するかどうかを尋ねるメッセージ・ウィンドウが表示されたら、**「はい」**をクリックします。
   コードの生成には、数分かかります。
4. デプロイメント・コードが生成されたら、 **User** bean を再び右クリックして、**「追加」>「アクセス Bean」**を選択します。「Create Access Bean SmartGuide (アクセス Bean の作成 SmartGuide)」がオープンします。
5. 「アクセス Bean プロパティーの選択」ページのデフォルト値を受け入れて、**「次へ」**をクリックします。
6. 「ゼロ引き数コンストラクターの定義」ページのデフォルト値を受け入れて、**「次へ」**をクリックします。
7. 「コピー・ヘルパー用に bean プロパティーを選択してカスタマイズする」ページで、「エンタープライズ Bean」列の **bonusPoint** までスクロールダウンします。
   bean の**「コピー・ヘルパー」**をチェックして、コンバーターを `com.ibm.commerce.base.objects.WCSStringConverter` に設定します。
8. **「終了」**をクリックします。

9. “Code generation completed” というメッセージが表示されたら、**「OK」**をクリックします。

テスト・クライアントを使った変更のテスト

カスタマイズしたばかりの User エンティティー Bean をテストするには、テスト・クライアントを使います。テスト・クライアントを開始して、エンティティー Bean をテストするには、以下のようにします。

1. 349 ページの『永続ネーム・サーバーの開始と停止』で説明されているように、永続ネーム・サーバーを開始します。
2. WCSUser EJB グループのある EJB サーバーを開始します (350 ページの『EJB サーバーの開始と停止』を参照)。
3. 「エンタープライズ Bean」ペインで、 **WCSUser** EJB グループを拡張表示します。 **User** エンティティー Bean を右クリックして、**「テスト・クライアントの実行」**を選択します。
4. 「EJB 検索」ウィンドウで、**「検索」**をクリックします。
5. メソッドのリストで、 **findByPrimaryKey(MemberKey)** を選択します。
6. 「詳細情報」ペインで、 **<null>** をクリックしてから、引き数ボックスに `-1000` と入力して、「EJB クライアント・テスト」ウィンドウの「呼び出し」アイコンをクリックします。

   ここで入力する値は、 USERS テーブルの USERS_ID 列の値と一致いなければならないことに注意してください。

   このメソッドの実行が完了したら、 -1000 という ID を持ったユーザーの情報が「メソッド」ペインにツリー表示されます。この表示の中に、「メソッド」というノードがあります。
7. **「メソッド」**ノードを拡張表示します。
8. **getBonusPoint()** をクリックしてから、「呼び出し」アイコンをクリックします。このメソッドは顧客のボーナス・ポイントのバランスを取得します。現在のボーナス・ポイントのバランスが戻されます。
9. **setBonusPoint(int)** をクリックし、「詳細情報」ペインに `100` と入力して、「呼び出し」アイコンをクリックします。
10. **getBonusPoint()** をクリックして、「呼び出し」アイコンをクリックします。
    値 100 が戻されます。これは、データベースが User エンタープライズ Bean によって正常に更新されたことを示しています。
11. User ウィンドウと「EJB クライアント・テスト」ウィンドウをクローズします。

GetNewProductContractUnitPriceCmd インターフェースの作成

このカスタマイズ演習の一部として、顧客のボーナス・ポイントのバランスに基づいて割引価格を計算する新しいビジネス・ロジックを作成します。この計算は、新しいタスク・コマンドによって実行されます。このセクションでは、この新しいタスク・コマンドのインターフェースを作成します。

新しいタスク・コマンドのインターフェースを作成するには、以下のようにします。

1. ワークベンチで、「プロジェクト」タブを選択して、 **_WCSamples** プロジェクトを拡張表示します。
2. **com.ibm.commerce.sample.commands** パッケージを右クリックして、**「追加」**>**「インターフェース」**を選択します。
3. **「新規インターフェースの作成」**が選択されていることを確認して、**「インターフェース名」**フィールドに、 GetNewProductContractUnitPriceCmd と入力します。
4. **「追加」**をクリックして拡張したいインターフェースを選択し、続いて**「パターン」**フィールドに com.ibm.commerce.price.commands.GetProductContractUnitPriceCmd と入力して、**「追加」**に続いて**「クローズ」**をクリックします。
5. **「次へ」**をクリックします。
6. 適切なインポート・ステートメントを追加するために、**「パッケージの追加」**をクリックして、以下のようにします。
   a. **「パターン」**フィールドに、 com.ibm.commerce.price.utils と入力して、**「追加」**をクリックします。
   b. **「パターン」**フィールドに、 com.ibm.commerce.exception と入力して、**「追加」**をクリックします。
   c. **「クローズ」**をクリックします。
7. **「インターフェースを公開する」**が選択されていることを確認します。
8. **「終了」**をクリックします。
   新しいインターフェースのソース・コードが「ソース」ペインに表示されます。
9. getBonusPrice() メソッドのメソッド・シグニチャーを以下のようにして作成します。
   a. **GetNewProductContractUnitPriceCmd** インターフェースを右クリックして、**「追加」**>**「メソッド」**を選択します。
   「Create Method SmartGuide (メソッドの作成 SmartGuide)」がオープンします。
   b. **「新規メソッドの作成」**が選択されていることを確認して、**「次へ」**をクリックします。
   c. 「メソッド名」フィールドに、 getBonusPrice と入力します。
   d. メソッドの戻りタイプを選択するために、**「ブラウズ」**をクリックします。**「パターン」**フィールドに、 MonetaryAmount と入力して、**「OK」**をクリックして、**「次へ」**をクリックします。

e. メソッドから出される例外を選択するために、**「追加」**を選択します。**「パターン」**フィールドに、ECSystemException と入力して、**「追加」**をクリックして、**「クローズ」**をクリックします。

f. **「終了」**をクリックします。

10. コマンドのデフォルト・インプリメンテーション・クラスを指定する新しいフィールドを以下のようにしてインターフェースに追加します。

a. **GetNewProductContractUnitPriceCmd** インターフェースを右クリックして、**「追加」**>**「フィールド」**を選択します。
「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。

b. **「フィールド名」**フィールドに、 defaultCommandClassName と入力します。

c. **「フィールド・タイプ」**ドロップダウン・リストで、**「ストリング」**を選択します。

d. **「初期値」**フィールドに、以下のように入力します。

```
"com.ibm.commerce.sample.commands.
   GetNewContractUnitPriceCmdImpl"
```

それから**「終了」**をクリックします。

注:

1) この値を入力するときに、必ず二重引用符を付けてください。

2) この新しいフィールドを作成するとスーパークラスのフィールドが隠されることを示す警告メッセージが出されたら、**「はい」**をクリックして先に進みます。

11. コマンドの名前を指定する新しいフィールドを以下のようにしてインターフェースに追加します。

a. **GetNewProductContractUnitPriceCmd** インターフェースを右クリックして、**「追加」**>**「フィールド」**を選択します。
「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。

b. **「フィールド名」**フィールドに、 NAME と入力します。

c. **「フィールド・タイプ」**ドロップダウン・リストで、**「ストリング」**を選択します。

d. **「初期値」**フィールドに、以下のように入力します。

```
"com.ibm.commerce.sample.commands.
   GetNewProductContractUnitPriceCmd"
```

それから**「終了」**をクリックします。

GetNewContractUnitPriceCmdImpl インプリメンテーション・クラスの作成

新しいタスク・コマンドのインプリメンテーション・クラスを作成する必要があります。このインプリメンテーション・クラスは、作成したばかりのインターフェースをインプリメントしたものであり、そのタスク・コマンドのビジネス・ロジックが組み込まれています。

GetNewContractUnitPriceCmdImpl インプリメンテーション・クラスを作成するには、以下のようにします。

1. ワークベンチで、「プロジェクト」タブを選択して、 **_WCSamples** プロジェクトを拡張表示します。
2. **com.ibm.commerce.sample.commands** パッケージを右クリックして、**「追加」**>**「クラス」**と選択します。
   「Create Class SmartGuide (クラスの作成 SmartGuide)」がオープンします。
3. **「新規クラスの作成」**が選択されていることを確認して、以下のようにクラスを作成します。
   a. **「クラス名」**フィールドに、 GetNewContractUnitPriceCmdImpl と入力します。
   b. スーパークラスを指定するために、**「ブラウズ」**をクリックし、**「パターン」**フィールドに com.ibm.commerce.price.commands.GetContractUnitPriceCmdImpl と入力して、**「OK」**をクリックします。
   c. **「次へ」**をクリックします。
   d. **「パッケージの追加」**をクリックして、インポートするパッケージを指定します。**「パターン」**フィールドに、以下のパッケージを入力します。
      - com.ibm.commerce.command と入力して**「追加」**をクリックする
      - com.ibm.commerce.exception と入力して**「追加」**をクリックする
      - com.ibm.commerce.price.commands と入力して**「追加」**をクリックする
      - com.ibm.commerce.price.utils と入力して**「追加」**をクリックする
      - com.ibm.commerce.ras と入力して**「追加」**をクリックする
      - com.ibm.commerce.server と入力して**「追加」**をクリックする
      - java.math と入力して、**「追加」**に続いて**「クローズ」**をクリックする。
   e. クラスがインプリメントするインターフェースを指定するために、**「追加」**をクリックします。「パターン」フィールドに、以下のインターフェースを入力します。
      - GetContractSpecialPriceCmd と入力して**「追加」**をクリックする
      - GetContractUnitPriceCmd と入力して**「追加」**をクリックする
      - GetProductContractUnitPriceCmd と入力して**「追加」**をクリックする
      - GetNewProductContractUnitPriceCmd と入力して**「追加」**に続いて**「クローズ」**をクリックする。
   f. **「終了」**をクリックします。

4. **GetNewContractUnitPriceCmdImpl** クラスを右クリックして、**「追加」>「フィールド」**を選択します。「Create Field SmartGuide (フィールドの作成 SmartGuide)」がオープンします。以下のように入力します。
   a. **「フィールド名」**フィールドに、 `bonusPrice` と入力します。
   b. フィールド・タイプを選択するために、**「ブラウズ」**をクリックします。「パターン」フィールドに、 `MonetaryAmount` と入力して、**「OK」**をクリックします。
   c. **「終了」**をクリックします。
5. クラスに新しい `performExecute()` メソッドを追加します。このメソッドは、以下のことをします。
   - スーパークラス (GetContractUnitPriceCmdImpl) の `performExecute()` メソッドを呼び出します。
   - 例外処理メカニズムで使用する `thisClass` と `methodName` の値を設定します。
   - StoreAccessBean をインスタンス化します。
   - UserAccessBean をインスタンス化します。
   - 元の商品価格を取得します。
   - 該当する最大割引額を計算します。
   - 新しいボーナス価格を計算します (この価格は、 *double* タイプ)。
   - ストアの通貨タイプを取得します。
   - ボーナス価格を四捨五入して、その値を正しい通貨による金額として保管します。

   このメソッドを追加するために、 **GetNewContractUnitPriceCmdImpl** クラスを右クリックして、**「追加」>「メソッド」**を選択します。「Create Method SmartGuide (メソッドの作成 SmartGuide)」がオープンします。 以下のように入力します。
   a. **「新規メソッドの作成」**が選択されていることを確認して、**「次へ」**をクリックします。
   b. **「メソッド名」**フィールドに、 `performExecute` と入力します。
   c. **「戻りタイプ」**ドロップダウン・リストで、**「void」**を選択して、**「次へ」**をクリックします。
   d. メソッドから出される例外を選択するために、**「追加」**を選択します。**「パターン」**フィールドに、 `ECException` と入力して、**「追加」**をクリックして、**「クローズ」**をクリックします。
   e. **「終了」**をクリックします。
   f. ソース・コードの以下の行の後に、

```
public void performExecute()
   throws com.ibm.commerce.exception.ECException
{
```

以下のコードを追加します。

PDF 版のプログラマーズ・ガイドから、このコードを切り貼りすることができます。まず VisualAge for Java の「Scrapbook」ウィンドウにコードをコピーして (詳しくは VisualAge for Java のオンライン・ヘルプを参照)、切り貼り操作のときに文字が失われなかったかどうか検査することを推奨します。コードを検査した後、宛先にコピーしてください。別のエディターにテキストをコピーすると、変更される文字があることに注意してください。

```
super.performExecute();

// Get and set this class name and method
// for use when exceptions occur.
final String thisClass =
   GetContractUnitPriceCmdImpl.class.getName();
final String methodName = "performExecute";

//get the store access bean
Integer storeId = getStoreId();
com.ibm.commerce.common.objects.StoreAccessBean storeAB =
     getCommandContext().getStore(storeId);

//get the user access bean
com.ibm.commerce.user.objects.UserAccessBean bonusAB =
     new com.ibm.commerce.user.objects.UserAccessBean();

// get the calculated price from the GetContractUnitPriceCmdImpl
MonetaryAmount priceOrg = super.getPrice();
double dblPriceOrg = priceOrg.getValue().doubleValue();

//calculate the maximum bonus that can apply to this product
double dblBonusPrice; // = dblPriceOrg;
double dblMaxBonusPoint = 0;

try {
bonusAB.setInitKey_MemberId(super.getUserId().toString());
bonusAB.refreshCopyHelper();
double dblMaxDed = dblPriceOrg * 0.2;
dblMaxBonusPoint =
     (new java.math.BigDecimal(
        bonusAB.getBonusPoint()).doubleValue());
if (dblMaxBonusPoint > dblMaxDed) dblMaxBonusPoint = dblMaxDed;
} catch (javax.ejb.CreateException ex) {
throw new ECSystemException(
     ECMessage._ERR_CREATE_EXCEPTION, thisClass, methodName, ex);
} catch (javax.ejb.FinderException ex) {

} catch (javax.naming.NamingException ex) {
throw new ECSystemException(
     ECMessage._ERR_GENERIC, thisClass, methodName, ex);
} catch (java.rmi.RemoteException ex) {
```

```
throw new ECSystemException(
     ECMessage._ERR_REMOTE_EXCEPTION, thisClass, methodName, ex);
}


//apply the maximum applicable bonus to this product price
dblBonusPrice = dblPriceOrg - dblMaxBonusPoint ;

//get the currency of this store
CommandContext context = getCommandContext();
String requestedCurrency = Helper.getCurrency( context, storeAB );

//round off and return the bonus price in MonetoryAmount type
bonusPrice = new MonetaryAmount(
     new BigDecimal(dblBonusPrice), requestedCurrency);
CurrencyManager.getInstance().roundCustomized(bonusPrice, storeAB);
```

作業内容を保管します。

6. **GetNewContractUnitPriceCmdImpl** クラス内の **getBonusPrice()** メソッドを選択して、そのソース・コードを表示します。ソース・コードで、

   ```
   return null;
   ```

   これを以下のように変更します。

   ```
   return bonusPrice;
   ```

   作業内容を保管します。

NewProductDataBean データ Bean の作成

既存の WebSphere Commerce ProductDataBean に `getCalculatedBonusPrice()` メソッドを追加する必要があります。ProductDataBean のコードを実際に変更することはできないので、 ProductDataBean を拡張した新しいデータ Bean を作成して、その新しいデータ Bean にそのメソッドを追加しなければなりません。

新しいデータ Bean を作成するには、以下のようにします。

1. ワークベンチで、「プロジェクト」タブを選択して、 **_WCSamples** プロジェクトを拡張表示します。
2. **com.ibm.commerce.sample.databeans** パッケージを右クリックして、**「追加」** > **「クラス」**と選択します。「Create Class SmartGuide (クラスの作成 SmartGuide)」がオープンします。以下のように入力します。
   a. **「クラス名」**フィールドに、 `NewProductDataBean` と入力します。
   b. スーパークラスを指定するために、**「ブラウズ」**をクリックし、**「パターン」**に `com.ibm.commerce.catalog.beans.ProductDataBean` と入力します。**「OK」**に続いて**「次へ」**をクリックします。
   c. 適切なインポート・ステートメントをクラスに追加するために、**「パッケージの追加」**をクリックして以下の操作を行います。

- com.ibm.commerce.beans と入力して**「追加」**をクリックする
- com.ibm.commerce.catalog.objects と入力して**「追加」**をクリックする
- com.ibm.commerce.command と入力して**「追加」**をクリックする
- com.ibm.commerce.datatype と入力して**「追加」**をクリックする
- com.ibm.commerce.exception と入力して**「追加」**をクリックする
- com.ibm.commerce.price.beans と入力して**「追加」**をクリックする
- com.ibm.commerce.ras と入力して**「追加」**をクリックする
- com.ibm.commerce.sample.commands と入力して**「追加」**をクリックする
- com.ibm.commerce.server と入力して**「追加」**をクリックする
- java.util と入力して、**「追加」**に続いて**「クローズ」**をクリックする。

d. **「終了」**をクリックします。

3. **NewProductDataBean** クラスを右クリックし、**「追加」**>**「メソッド」**と選択します。
「Add Method SmartGuide (メソッドの追加 SmartGuide)」がオープンします。
4. **「新規メソッドの作成」**が選択されていることを確認して、**「次へ」**をクリックします。
5. **「メソッド名」**フィールドに、 getCalculatedBonusPrice と入力します。
6. 戻りタイプを選択するために、**「ブラウズ」**をクリックします。**「パターン」**フィールドに、 PriceDataBean と入力して、**「OK」**をクリックして、**「次へ」**をクリックします。
7. メソッドから出される例外を選択するために、**「追加」**を選択します。**「パターン」**フィールドに、ECSystemException と入力して、**「追加」**をクリックして、**「クローズ」**をクリックします。
8. **「終了」**をクリックします。
9. ソース・コードの中の return null; を以下に置き換えます。

```
PriceDataBean ibnPrice = null;
try {
     GetNewProductContractUnitPriceCmd comm =
        (GetNewProductContractUnitPriceCmd) CommandFactory.createCommand
        (GetNewProductContractUnitPriceCmd.NAME,
             getCommandContext().getStoreId());

     ECTrace.trace(ECTraceIdentifiers.COMPONENT_CATALOG,
     this.getClass().getName(), "getCalculatedBonusPrice",
     "Getting Price for CatalogEntry: " + getProductID());

     comm.setCatEntryId(new Long(getProductID()));
     comm.setCommandContext(getCommandContext());
     comm.execute();
     ibnPrice = new PriceDataBean(comm.getBonusPrice(),
          getCommandContext().getStore(),
          getCommandContext().getLanguageId());
```

```
} catch (Exception e) {
     throw new ECSystemException(ECMessage._ERR_RETRIEVE_PRICE,
          this.getClass().getName(), "getCalculatedBonusPrice",e);
}

return ibnPrice;
```

作業内容を保管します。

商品の表示テンプレートへの新しいボーナス価格の追加

次のステップは、商品の表示テンプレートに新しいボーナス価格を追加することです。そうすれば、ショッパーには、カスタマイズ価格が表示されるようになります。表示テンプレートを更新したら、新しい割引価格が表示されます。

サンプル・ストアは、 ProductDisplay.jsp テンプレートを使用して商品を表示します。したがって、新しい価格を表示するための情報でこのテンプレートを更新する必要があります。

表示テンプレートを更新するには、以下のようにします。

1. 以下のディレクトリーに移動します。
   vaj_drive:¥VAJava¥ide¥project_resources¥IBM WebSphere Test Environment¥hosts¥default_host¥default_app¥web¥*store_name*
2. ProductDisplay.jsp ファイルのコピーを作成して、 ProductDisplay.jsp.bak という名前を付けます。
3. テキスト・エディターで、 ProductDisplay.jsp をオープンします。
4. <%@ page import="com.ibm.commerce.common.beans.*" %> の後に、以下のインポート・ステートメントを追加します。

   ```
   <%@ page import="com.ibm.commerce.sample.commands.*" %>
   <%@ page import="com.ibm.commerce.sample.databeans.*" %>
   ```

5. ProductDataBean が出現するすべての個所を、 NewProductDataBean に置き換えます。
6. 以下の行を見付けます。

   ```
   <font class="price"><%=product.getCalculatedContractPrice()%></font>
      <br><br>
   ```

   この行の後に、商品のボーナス価格を取り込んで表示するための以下の行を挿入します。

   ```
   <font class="price"><%=product.getCalculatedBonusPrice()%>
         Bonus Price </font>
   <br><br>
   ```

7. このファイルを保管します。

注: PDF 版の *WebSphere Commerce プログラマーズ・ガイド* から表示テンプレートにセクションを切り貼りする場合は、このプロセス中にどの文字も変更されないようにしてください。

エンタープライズ Bean の拡張版のテスト

このセクションでは、 InFashion サンプル・ストアの商品を表示して、エンタープライズ Bean の拡張版をテストします。新しいボーナス価格が表示されます。

ただし、このサンプルでは、簡略化のためにボーナス価格がすべてのショッパーに (登録ショッパーだけでなくゲスト・ショッパーにも) 表示されます。ボーナス価格のないショッパーの場合は、通常価格と同じ額のボーナス価格が表示されることになります。

エンタープライズ Bean の拡張版をテストして、ボーナス価格を表示してみるには、以下のようにします。

1. `_WCSamples` プロジェクトがサーブレット・エンジンのパスに含まれていることを以下のようにして確認します。
   a. VisualAge for Java の**「ワークスペース」**メニューで、**「ツール」** >**「WebSphere Test Environment」**を選択します。
   WebSphere Test Environment Control Center がオープンします。
   b. **「サーブレット・エンジン」**をクリックします。
   c. サーブレット・エンジンが稼働している場合は、**「サーブレット・エンジンの停止」**をクリックしてから、**「クラスパスの編集」**をクリックします。
   d. **_WCSamples** がまだ選択されていない場合は、それを選択して、**「OK」**をクリックします。
2. WebSphere Test Environment を開始します (349 ページの『付録 A. WebSphere Test Environment の開始と停止』を参照)。永続ネーム・サーバーと EJB サーバーがすでに開始されている場合は、サーブレット・エンジンだけを開始します。
3. ブラウザーをオープンして、以下の URL を入力します。

   ```
   http://localhost:8080/webapp/wcs/stores/servlet/StoreCatalogDisplay?
       storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

4. 「サービス」ヘッダーの下の**「登録」**リンクをクリックしてから、「新規顧客」ヘッダーの下の**「登録」**をクリックします。
   `wctester@wc` という電子メール・アドレスと `wctester1` というパスワードを使用して、新しい顧客を登録します。それ以外のフィールドにはテスト値を入力して、**「送信」**をクリックします。ブラウザーはオープンしたままにしておきます。
5. DB2 DB2 コマンド・センターをオープンして、以下の操作を行います。
   a. **「インタラクティブ」**タブをクリックします。
   b. **「コマンド」**フィールドで、以下のようにします。
      1) 以下のように入力します。

```
connect to your_database_name
```

ここで、*your_database_name* はご使用の WebSphere Commerce データベースの名前です。「実行」アイコンをクリックします。

2) `select users_id from userreg where logonid = 'wctester@wc'` と入力して、「実行」アイコンをクリックします。

c. 「クエリー結果」タブに、ステップ 4 (329 ページ) で登録した顧客のエントリーが表示されます。顧客の USERS_ID の値をここにメモします。 _____

d. 新しく登録された顧客のボーナス・ポイントのバランスを更新します。「インタラクティブ」タブをクリックして、**「コマンド」**フィールドに、以下のように入力します。

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id
```

users_id は、ステップ 5c の値です。「実行」アイコンをクリックします。

6. Oracle 以下のようにして、テスト・ユーザーのボーナス・ポイントのバランスを更新してください。

a. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。

b. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。

c. **「パスワード」**フィールドに、 Oracle パスワードを入力します。

d. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。

e. `select users_id from userreg where logonid = 'wctester@wc';` と入力します。

f. ステップ 4 で登録した顧客のエントリーが表示されます。顧客の USERS_ID の値をここにメモします。 _____

g. 以下のように入力して、新規登録された顧客のボーナス・ポイントのバランスを更新します。

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id;
```

users_id は、ステップ 6f の値です。

h. データベースの変更をコミットするには

```
commit;
```

と入力し、それから Enter を押して SQL ステートメントを実行します。

7. ブラウザーで、**「メンズ」**をクリックしてストアの「メンズ」セクションを表示します。

8. 特集のリンクをクリックして、商品ページを表示します。このページは、正規価格と、顧客のボーナス・ポイントの残高に基づく割引価格を表示します。

注: ボーナス価格の代わりにスタック・トレースが表示されたら、キャッシュを使用不可にする必要が生じることがあります。そのためには、以下のように、*instance_name*.xml ファイル中の CacheDaemon コンポーネントの値を false に設定します。

```
<component compClassName=
      "com.ibm.commerce.cache.daemon.CacheDaemonComponent"
               enable="false"
               name="CacheDaemon" />
```

instance_name.xml ファイル中の値を変更し終えたら、 WebSphere Test Environment 中のサーブレット・エンジンを停止して再始動しなければなりません。

(オプション) カスタマイズされたビジネス・ロジックのリモート WebSphere Commerce Server へのデプロイメント

このセクションでは、変更されたエンティティー Bean と新しいタスク・コマンドを、WebSphere Test Environment の外で実行しているストアにデプロイメントする方法を示します。

デプロイメントを実行するには、 WebSphere Commerce パブリック・エンタープライズ Bean とコマンドとデータ Bean のロジックの JAR ファイルを作成し、 JAR ファイルをターゲット・サーバー上の適切なディレクトリーに入れ、 WebSphere Commerce インスタンスを停止し、クラスパスを変更し、 XMLConfig ユーティリティーを使用してエンタープライズ Bean のデプロイメントを実行して、インスタンスを再始動します。

新しい価格コマンドの JAR ファイルの作成

_WCSamples プロジェクトの JAR ファイルを作成して、新しいタスク・コマンドがデプロイメントされるようにする必要があります。この JAR ファイルを作成するには、開発マシンで以下のようにします。

1. WebSphere Test Environment を停止します (349 ページの『付録 A. WebSphere Test Environment の開始と停止』を参照)。
2. 「プロジェクト」タブを選択した状態で、**_WCSamples** プロジェクトを選択します。
3. そのプロジェクトを強調表示して右クリックし、**「エクスポート」**を選択します。「Export SmartGuide (エクスポート SmartGuide)」がオープンします。
4. **「JAR ファイル」**を選択し、**「次へ」**をクリックします。
5. **「JAR ファイル」**フィールドで、以下のように入力します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp_c¥wcssamplesc_1.jar
   drive は WebSphere Commerce のインストール先のドライブです。

6. 属性を以下のように選択します。

属性	値
クラス	チェックする
Java	チェックしない
リソース	チェックする
bean	チェックする
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

7. **「終了」**をクリックします。

作成された JAR ファイルには完全なパッケージ命名情報が含まれていないため、(VisualAge for Java 以外の) 別のパッケージ・ユーティリティーを使用して JAR ファイルを再パッケージ化する必要があります。このファイルを再パッケージ化するには、以下のようにします。

1. コマンド・ウィンドウで、以下のディレクトリーに移動します。
   drive:¥WebSphere¥CommerceServerDev¥mytemp_c
2. `mkdir temp3` と入力します。
3. `cd temp3` と入力します。
4. パスを以下のように設定します。
   `set PATH=%PATH%;`*drive*`:¥WebSphere¥WebSphereStudio4¥bin;`
   drive は、WebSphere Studio のインストール先のドライブです。
5. `jar xvf ../wcssamplesc_1.jar` と入力します。
6. `jar cvf ../wcssamplesc.jar *` と入力します (名前から _1 が削除されていることに注意)。

WCSUser EJB グループ用の JAR ファイルの作成

変更されたエンタープライズ Bean を含んでいる EJB グループの EJB 1.1 Export JAR ファイルを作成する必要があります。つまり、JAR ファイルを作成するときに、以下のグループを選択する必要があります。

- WCSUser

WCSUser EJB グループの JAR ファイルを作成するには、以下のようにします。

1. 「EJB」タブを選択し、**「WCSUser」**EJB グループを強調表示します。
2. **「WCSUser」** EJB グループを右クリックして、**「エクスポート」**>**「EJB 1.1 JAR」**を選択します。
   「Export to an EJB 1.1 JAR File SmartGuide (EJB 1.1 JAR ファイルへのエクスポート SmartGuide)」がオープンします。

3. **「JAR ファイル」**フィールドに、 *drive*:¥WebSphere¥CommerceServerDev¥mytemp_c¥CustomizedWCSUserDeployed_DT.jar と入力します。
4. 属性を以下のように選択します。

属性	値
クラス	チェックする
Java	チェックしない
リソース	チェックする
ターゲット・データベース	DB2 DB2 データベースにデプロイメントしている場合は、**「DB2 for NT V7.1」**を選択します。 Oracle Oracle データベースにデプロイメントしている場合は、**「Oracle V8」**を選択します。
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

5. **「終了」**をクリックします。

これで、JAR ファイルが作成されます。

JAR ファイルの名前には接尾部 "_DT" が付きます。つまり、これを WebSphere Commerce アプリケーションにデプロイメントする前に、WebSphere Application Server 付属の EJB デプロイメント・ツールを使ってこの JAR ファイルを実行する必要があります。

wcsejsclient.jar ファイルの作成

クライアント JAR ファイルを作成するには、以下のようにします。

1. 「EJB」タブを選択し、すべての WebSphere Commerce EJB グループ (名前が WCS で始まるもの) を強調表示します。 これらすべてのグループが強調表示された状態で右クリックし、**「エクスポート」**>**「Client JAR (クライアント JAR)」**を選択します。
   「Export SmartGuide (エクスポート SmartGuide)」がオープンします。
2. **「JAR ファイル」**フィールドに、*drive*:¥WebSphere¥CommerceServerDev¥mytemp_c¥wcsejsclient.jar と入力します。
3. 属性を以下のように選択します。

属性	値
bean	チェックする
クラス	チェックする

属性	値
Java	チェックしない
リソース	チェックする
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

4. **「終了」**をクリックします。

これで、JAR ファイルが作成されます。

更新済みの JSP テンプレートのターゲット・ストア・ディレクトリーへのコピー

328 ページの『商品の表示テンプレートへの新しいボーナス価格の追加』では、新しく作成された価格を反映させるために表示テンプレートを更新しました。このステップでは、更新済みの JSP テンプレートを、WebSphere Test Environment ディレクトリー構造から、 WebSphere Test Environment の外で実行しているストアが使用するディレクトリーにコピーします。

1. 開発マシンで、以下のディレクトリーに移動します。
   vaj_drive:¥VAJava¥Ide¥project_resources¥IBM WebSphere Test Environment ¥hosts¥default_host¥default_app¥web¥*store_directory*
   vaj_drive は VisualAge for Java のインストール先のドライブ、 *store_directory* は、サンプル・ストアのディレクトリーの名前です。
   ProductDisplay.jsp ファイルをコピーします。
2. ProductDisplay.jsp ファイルを以下のディレクトリーに貼り付けます。

   ```
   drive:¥WebSphere¥AppServer¥installedApps¥
      WC_Enterprise_App_instance_name.ear¥
      wcstores.war¥store_directory
   ```

   ここで *drive* は WebSphere Commerce がインスタンスされているドライブ、*store_directory* はストアのディレクトリー名、 *instance_name* は WebSphere Commerce インスタンスの名前です。

JAR ファイルのターゲットの WebSphere Commerce Server へのコピー

JAR ファイルを、開発マシンからターゲットの WebSphere Commerce Server の適切なディレクトリーにコピーしなければなりません。これらのファイルをコピーするには、以下のようにします。

1. 開発マシンで、ディレクトリー
   drive:¥WebSphere¥CommerceServerDev¥mytemp_c
   に移動し、以下のファイルを見付けます。
   - wcssamplesc.jar
   - CustomizedWCSUserDeployed_DT.jar

- wcsejsclient.jar

ここで、*drive* は WebSphere Commerce Studio, Business Developer Edition のインストール先のドライブです。

上記のそれぞれのファイルを、ターゲットの WebSphere Commerce Server 上の特定のディレクトリーにコピーする必要があります。各ファイルを正しい場所に確実に保管するために、以下のステップを注意深くお読みください。

2. wcssamplesc.jar ファイルをターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。

```
drive:¥WebSphere¥AppServer¥installedApps¥
   WC_Enterprise_App_instance_name.ear¥
   wcstores.war¥WEB-INF¥lib
```

drive は WebSphere Commerce Business Edition のインストール先のドライブで、*instance_name* はインスタンスの名前です (たとえば demo)。

3. wcsejsclient.jar ファイルをターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。

```
drive:¥WebSphere¥CommerceServer¥temp¥lib
```

4. CustomizedWCSUserDeployed_DT.jar ファイルをターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。

```
drive:¥WebSphere¥CommerceServer¥temp
```

EJB デプロイメント・ツールの実行

新しい EJB グループを含んでいる JAR ファイルに対して、 EJB デプロイメント・ツールを実行する必要があります。このツールは WebSphere Application Server に付属しています。

このツールを実行するには、以下のようにします。

1. コマンド・プロンプトで、以下のディレクトリーに移動します。

```
drive:¥WebSphere¥CommerceServer¥temp
```

2. 以下のコマンドを入力して、このツールをシステムに一時的に追加します。

```
PATH=drive:¥WebSphere¥AppServer¥deploytool;%PATH%
```

3. 以下のように、ejbdeploy コマンドを入力します。

```
ejbdeploy EJBGroupJARFile WorkingDir OutputJARFile -nowarn -keep -35 -cp
   ClassPathOfDepJARFiles
```

ここで、

- *EJBGroupJARFile* は EJB グループの JAR ファイルの名前です。この場合は CustomizedWCSUserDeployed_DT.jar です。
- *WorkingDir* は作業ディレクトリーです。

- *OutputJARFile* は出力 JAR ファイルの名前です。ここでは CustomizedWCSUserDeployed.jar と入力します。
- -nowarn は、警告および情報メッセージを抑制するためのオプショナル・パラメーターです。
- -keep は、 ejbdeploy コマンドの実行後も作業ディレクトリーを保存するためのオプショナル・パラメーターです。
- -35 は必須パラメーターであり、 WebSphere Application Server バージョン 3.5 付属の EJB デプロイメント・ツールが使用したのと同じ CMP エンティティー Bean 用トップダウン・マッピング規則を使用します。
- -cp *ClassPathOfDepJARFiles* はすべての従属 JAR ファイルのクラスパスです。既存の WebSphere Commerce エンタープライズ Bean を変更したときには、wcsejsclient.jar、wcsejbimpl.jar、および xml4j.jar ファイルを従属 JAR ファイルのクラスパスに含める必要があります。そうするには、以下のように入力します。

  ```
  "drive:¥WebSphere¥CommerceServer¥temp¥lib¥wcsejsclient.jar;
  drive:¥WebSphere¥AppServer¥InstalledApps¥
  WC_Enterprise_App_instanceName.ear¥lib¥wcsejbimpl.jar;
  drive:¥WebSphere¥AppServer¥InstalledApps¥
  WC_Enterprise_App_instanceName.ear¥lib¥xml4j.jar;"
  ```

エンティティー Bean のトランザクション分離レベルの変更

このステップでは、modifyIsolationLevel コマンドを使用して、エンティティー Bean のトランザクション分離レベルを、使用している特定のデータベースに必要とされるレベルに変更します。

modifyIsolationLevel コマンドを実行するには、以下のようにします。

1. ターゲットの WebSphere Commerce Server で、コマンド・プロンプトを使用して以下のディレクトリーに移動します。

   ```
   drive:¥WebSphere¥CommerceServer¥bin
   ```

2. 以下の一般構文をもつ modifyIsolationLevel コマンドを出す必要があります。

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType db_type
   ```

   where

   - *jar_file_name.jar* は、カスタマイズされたコードを含む JAR ファイルの名前です。
   - *log_file_name* は、情報がログに記録されるべき完全修飾ファイル名です。
   - *db_type* はご使用のデータベースのタイプです。 DB2 または ORACLE のいずれかを入力します。

   以下は、すべての値が指定された modifyIsolationLevel コマンドの例です。

```
modifyIsolationLevel -jarFile
   D:¥WebSphere¥CommerceServer¥temp¥CustomizedWCSUserDeployed.jar
   -logFile D:¥WebSphere¥CommerceServer¥instances¥demo¥logs¥output.log
   -dbType DB2
```

注: パラメーター名については、大文字小文字の区別をします。つまり、`jarFile` と `jarfile` は同じではありません。パラメーター名を正しく入力するように注意してください。

コマンド・ウィンドウに例外が表示されない場合は、コマンドが正常に実行されました。完了後は、デプロイメント JAR ファイルのタイム・スタンプが変更していることに注意してください。

ターゲット・データベースの更新

WebSphere Test Environment で使用されていないデータベースを使用しているターゲットの WebSphere Commerce Server にデプロイメントする場合は、以下のようにターゲット・データベースを更新しなければなりません。

- 207 ページの『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』を終了している場合は、このテーブルを更新して、 USERS テーブル中のユーザーごとに行を追加しなければなりません。
- 207 ページの『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』をまだ終了していない場合は、このテーブルを作成して取り込まなければなりません。

207 ページの『第 9 章 チュートリアル: ビジネス・ロジックの新規作成』をまだ終了していない場合は、このテーブルを作成して取り込まなければなりません。両方のシナリオごとの指示が用意されています。

DB2 DB2 データベースを使用している際に、 BONUS テーブルを*更新する* 必要がある場合は、以下のようにしてください。

1. DB2 コマンド・センターをオープンして (**「スタート」** > **「プログラム」** > **「IBM DB2」** > **「コマンド・センター」**)、**「スクリプト」**タブをクリックします。
2. **「スクリプト」**フィールドで、以下を入力します。

   ```
   connect to your_database_name
   ```

   ここで、*your_database_name* はご使用のデータベースの名前です。「実行」アイコンをクリックします。
3. **「スクリプト」**フィールドに以下を入力して、「実行」アイコンをクリックします。

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS))
   ```

   BONUS テーブルが更新されました。

DB2 DB2 データベースを使用している際に、 BONUS テーブルを*作成して取り込む* 必要がある場合は、以下のようにしてください。

1. DB2 コマンド・センターをオープンして (**「スタート」** > **「プログラム」** > **「IBM DB2」** > **「コマンド・センター」**)、**「スクリプト」**タブをクリックします。
2. **「スクリプト」**フィールドで、以下を入力します。

   ```
   connect to your_database_name
   ```

   ここで、*your_database_name* はご使用のデータベースの名前です。「実行」アイコンをクリックします。
3. **「スクリプト」**フィールドに以下を入力して、「実行」アイコンをクリックします。

   ```
   CREATE TABLE BONUS (MEMBERID BIGINT NOT NULL,
       BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
       constraint f_memberid foreign key (MEMBERID)
       references users (users_id) on delete cascade)
   ```

   BONUS テーブルが作成されました。
4. このテーブルに値を取り込むには、**「スクリプト」**フィールドに以下のように入力して、「実行」アイコンをクリックします。

   ```
   insert into BONUS (select USERS_ID, 0 from USERS)
   ```
5. DB2 コマンド・センターをクローズします。

Oracle Oracle データベースを使用している際に、 BONUS テーブルを*更新する* 必要がある場合は、以下のようにしてください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。
2. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。
3. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで以下の情報を入力することによって、 BONUS テーブルを更新します。

   ```
   INSERT INTO BONUS
   (SELECT USERS_ID, 0
   FROM USERS
   WHERE USERS_ID NOT IN (SELECT MEMBERID FROM BONUS));
   ```

   Enter を押して SQL ステートメントを実行します。
   これで BONUS テーブルが更新されました。BONUS テーブルが更新されました。
6. データベースの変更をコミットするには

   ```
   commit;
   ```

   と入力し、それから Enter を押して SQL ステートメントを実行します。

Oracle Oracle データベースを使用している際に、 BONUS テーブルを作成して取り込む 必要がある場合は、以下のようにしてください。

1. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。
2. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。
3. **「パスワード」**フィールドに、 Oracle パスワードを入力します。
4. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。
5. 「SQL Plus」ウィンドウで以下の情報を入力することによって、 BONUS テーブルを作成します。

   ```
   CREATE TABLE Bonus (MEMBERID NUMBER NOT NULL,
    BONUSPOINT INTEGER NOT NULL, constraint p_memberid primary key (MEMBERID),
       constraint f_memberid foreign key (MEMBERID)
       references users (users_id) on delete cascade);
   ```

   Enter を押して SQL ステートメントを実行します。
   これで BONUS テーブルが作成されました。
6. 以下を入力することによって、BONUS テーブルを取り込みます。

   ```
   insert into BONUS (select USERS_ID, 0 from USERS);
   ```
7. データベースの変更をコミットするには

   ```
   commit;
   ```

   と入力し、それから Enter を押して SQL ステートメントを実行します。

現在のエンタープライズ・アプリケーションを WebSphere Application Server からエクスポートする

このステップでは、現在のエンタープライズ・アプリケーションを WebSphere Application Server からエクスポートして、後でアプリケーション・アセンブリー・ツール からオープンできるようにします。

現在のエンタープライズ・アプリケーションを WebSphere Application Server からエクスポートするには、以下のようにします。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere 管理可能ドメイン」**を拡張表示します。
3. **「Enterprise Application (エンタープライズ・アプリケーション)」**を拡張表示します。
4. WebSphere Commerce アプリケーションを右クリックします。たとえば、**demo** アプリケーションを拡張表示して、**「Export Application (アプリケーションのエクスポート)」**を選択します。
5. **「Export directory (エクスポート先ディレクトリー)」**フィールドに *drive*:¥WebSphere¥CommerceServer¥working と入力します。
   これによって、すべてのリソースを含むアプリケーション全体が

WC_Enterprise_App_*instanceName*.ear ファイルにエクスポートされます (ここで、*instanceName* は WebSphere Commerce インスタンスの名前)。

注: 既存の WC_Enterprise_App_*instanceName*.ear ファイルがある場合、古いファイルの名前を変更するか、それを上書きすることができます。

エンタープライズ・アプリケーションの XML 構成情報のエクスポート

さらに、エンタープライズ・アプリケーションの XML 構成情報をエクスポートする必要もあります。この情報をエクスポートするには、 WebSphere Application Server に付属のコマンド行ユーティリティー XMLConfig を使用します。

この構成情報をエクスポートするには、以下のようにします。

1. コマンド・プロンプトで、以下のディレクトリーに移動します。

   drive:¥WebSphere¥CommerceServer¥working

2. XMLConfig ツールを起動し、以下のコマンドを入力して、部分的なエクスポートを実行します。

   ```
   xmlConfig -export OutputFileB.xml -partial was.export.app.xml
      -adminNodeName wasHostName
   ```

   ここで *wasHostName* は、WebSphere Application Server において、現在のエンタープライズ・アプリケーションが入っているノードの名前です。さらに、OutputFileB.xmlは、このコマンドの実行結果として生成されるファイルの名前です。

現行のエンタープライズ・アプリケーション内にあるエンタープライズ Bean に関する情報をエクスポートし終えたら、変更された User bean のコードを含む JAR ファイルを指すように OutputFileB.xml ファイルを更新する必要があります。

OutputFileB.xml ファイルを更新するには、以下のようにします。

1. 以下のディレクトリーに移動します。

   drive:¥WebSphere¥CommerceServer¥working

2. テキスト・エディターで OutputFileB.xml ファイルをオープンします。
3. <ear-file-name> タグを検索して、値を以下のように置き換えます。

   drive:¥WebSphere¥CommerceServer¥working¥
      WC_Enterprise_App_*instanceName*.ear

4. <ejb-module> スタンザで WCSUser EJB グループを見付け、 <jar-file> タグに含まれている値を CustomizedWCSUserDeployed.jar に変更します。
5. OutputFileB.xml ファイルを保管します。

変更された EJB グループをエンタープライズ・アプリケーションにアセンブルする

このステップでは、エンタープライズ・アプリケーションをアプリケーション・アセンブラー・ツールでオープンします。このツール内でオープンしたら、以下の操作を行うことによって、変更済み User bean をエンタープライズ・アプリケーションに組み込むことができます。

1. WCSUser EJB グループの既存のバージョンのクラスパスをコピーする。
2. WCSUser EJB グループの既存のバージョンを除去する。
3. WCSUser EJB グループの新しいバージョンをインポートする。新しい EJB グループの JAR ファイルは、エンタープライズ・アプリケーションの EJB Module セクション内に保管されます。
4. WCSUser EJB グループのクラスパスを設定する。

新しい EJB グループをエンタープライズ・アプリケーションにアセンブルするには、以下のようにします。

1. 以下のようにして、現在のエンタープライズ・アプリケーションのバックアップを取ります。
   a. コマンド・プロンプトで、以下のディレクトリーに移動します。

      ```
      drive:¥WebSphere¥CommerceServer¥working
      ```

   b. 以下のコマンドを入力します。

      ```
      copy WC_Enterprise_App_instanceName.ear
         WC_Enterprise_App_instanceName.ear.bak2
      ```

2. WebSphere Application Server 管理コンソールをオープンします。
3. **「ツール」**メニューから、**「Application Assembly Tool」**を選択します。 (「ウェルカム」ウィンドウがオープンしたら、**「キャンセル」**を選択して、コンソールにアクセスします。)
4. 以下のようにして、作業対象のエンタープライズ・アプリケーションをオープンします。
   a. **「ファイル」**メニューから、**「オープン」**を選択します。
   b. **「File name (ファイル名)」**フィールドで、以下を入力します。

      ```
      drive:¥WebSphere¥CommerceServer¥working¥
         WC_Enterprise_App_instanceName.ear
      ```

      それから**「オープン」**をクリックします。アプリケーションがオープンするまで待って、次のステップに進みます。これには数分かかります。
5. **「EJB Modules (EJB モジュール)」**をクリックします。右側の画面に、エンタープライズ・アプリケーション内の EJB モジュールが表示されます。
6. **WCSUser** EJB モジュールをクリックします。

7. 「General (一般)」タブをクリックして、既存の WCSUser EJB モジュールのクラスパス情報を表示します。この既存のクラスパス情報を、テキスト・ファイル (たとえば WCSUser_path.txt) にコピーします。
8. **WCSUser** EJB モジュールを右クリックして、**「削除」**を選択します。
9. **「EJB Modules (EJB モジュール)」**を右クリックして、**「Import (インポート)」**を選択します。
10. **「File name (ファイル名)」**フィールドで、以下を入力します。

    drive:¥WebSphere¥CommerceServer¥temp¥CustomizedWCSUserDeployed.jar

    それから**「オープン」**をクリックします。「Confirm Values (値の確認)」ウィンドウで、**「OK」**をクリックします。
11. CustomizedWCSUserDeployed.jar ファイルがインポートされたら、 **WCSUser** EJB グループまでスクロールして、このグループを選択します。
    このグループに関する情報が右側の画面に表示されます。
12. 以前のバージョンの WCSUser EJB グループのクラスパス情報を含んでいるテキスト・ファイルをオープンします。クラスパスを選択してコピーします。
13. 新しい WCSUser EJB グループの classpath フィールドに、このクラスパス情報を貼り付けます。
14. **「適用」**をクリックします。
15. **「ファイル」**メニューから、**「クローズ」**を選択します。
16. ファイルがクローズするのを待ち、**「ファイル」**メニューから**「オープン」**を選択して、 *drive*:¥WebSphere¥CommerceServer¥working¥WC_Enterprise_App_*instanceName*.ear ファイルを再オープンします。
17. 以下のようにして、User bean のセキュリティーを構成します。
    a. 「EJB Modules (EJB モジュール)」ノードを拡張表示し、 **WCSUser** ノードを見付けてこれを拡張表示します。
    b. **「Entity Beans (エンティティー Bean)」**を拡張表示します。
    c. **「ユーザー」**を拡張表示する。
    d. **「Method Extensions (メソッドの拡張機能)」**をクリックし、右側の画面で以下のようにします。
       1) **「Advanced (拡張)」**タブをクリックします。
       2) **「Security identity (セキュリティー ID)」**が選択されていることを確認します。
       3) 各メソッドごとに、**「Use identity of EJB server (EJB サーバーの ID を使用する)」**が選択されていることを確認します。
       4) **「適用」**をクリックします (変更を加えた場合)。

e. 左側のナビゲーション画面で、 WCSUser EJB グループの**「Security Roles (セキュリティー役割)」**を右クリックして**「新規」**を選択して、以下のようにします。

1) **「名前」**フィールドに WCSecurityRole と入力して、**「適用」**をクリックします。この役割がすでに存在している場合、このステップを実行する必要はありません。

f. 左側のナビゲーション画面で、WCSUser EJB グループの**「Method Permissions (メソッド許可)」**を右クリックし、**「New (新規)」**を選択して、以下のようにします。

1) **「Method Permission Name (メソッド許可名)」**フィールドで、WCMethodPermission と入力します。

2) **「Methods (メソッド)」**選択領域で**「追加」**をクリックします。「Add Methods (メソッドの追加)」ウィンドウがオープンします。

3) **CustomizedWCSUserDeployed.jar** を拡張表示して、すべてのエンタープライズ Bean を選択します (選択時にはシフト・キーを押したままにします)。**「OK」**をクリックします。「Enterprise bean (エンタープライズ Bean)」列の下にすべてのエンタープライズ Bean が表示され、「Types (タイプ)」列の下に**すべてのメソッド**が表示されます。

4) 「Roles (役割)」選択領域で、**「追加」**をクリックして、 WCSecurityRole を選択し、**「OK」**をクリックします。

5) **「適用」**をクリックしてから、**「OK」**をクリックします。

18. **「ファイル」**メニューから、**「Save (保管)」**を選択します。
19. アプリケーション・アセンブラー・ツールをクローズします。

このステップが完了したら、以前のすべてのビジネス・ロジックと新しいビジネス・ロジックを含む新しいエンタープライズ・アプリケーションが作成されました。これは、新しく変更した WC_Enterprise_App_*instanceName*.ear ファイルにすべて含まれます。

新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートする

新しいエンタープライズ・アプリケーションを WebSphere Application Server にインポートするための大まかなステップは、以下のとおりです。

1. WebSphere Application Server で現在実行されているエンタープライズ・アプリケーションを停止して、それを除去する。これらのステップは、WebSphere Application Server 管理コンソールで実行されます。
2. コマンド行ユーティリティー XMLConfig を使って新しいアプリケーションをインポートする。
3. WebSphere Application Server 管理コンソールを最新表示にして、新しいエンタープライズ・アプリケーションを開始する。

以下のセクションで、これらのステップについて詳しく説明します。

現在のエンタープライズ・アプリケーションの停止および除去: 現在のエンタープライズ・アプリケーションを停止して WebSphere Application Server から除去するには、以下のようにします。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere 管理可能ドメイン」**を拡張表示します。
3. **「ノード」**を拡張表示します。
4. *「nodeName」* **を拡張表示します** (*nodeName* は、ご使用のノードの名前です)。
5. **「Application Servers (アプリケーション・サーバー)」**を拡張表示します。
6. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Server -** *instanceName*」を右クリックして、**「停止」**を選択します。
7. **「Enterprise Applications (エンタープライズ・アプリケーション)」**を拡張表示します。
8. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Enterprise Server - demo」**アプリケーションを右クリックして、**「停止」**を選択します。
9. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Enterprise Server - demo」**アプリケーションを右クリックして、**「除去」**を選択します。
10. アプリケーションをエクスポートするかどうかの指示を求められたら、**「いいえ」**を選択します。

XMLConfig を使用して新しいエンタープライズ・アプリケーションをインポートする: XMLConfig コマンド行ユーティリティーを使用して新しいエンタープライズ・アプリケーションをインポートするには、以下のようにします。

1. 以下のディレクトリーに移動します。

   drive:¥WebSphere¥CommerceServer¥working

2. コマンド・プロンプトで以下のコマンドを入力して、エンタープライズ・アプリケーションを WebSphere Application Server にインポートします。

   ```
   xmlConfig -import OutputFileB.xml -adminNodeName was_hostname
   ```

   ここで、*was_hostname* は現在のアプリケーションを含んでいる WebSphere Application Server のノード名です。

注: ▶ 400 iSeries マシン上で実行する WebSphere Commerce インスタンスをデプロイした場合、アプリケーションをインポートした後で、ディレクトリー許可を変更するために追加のステップを実行する必要があります。これらの許可を変更する方法の詳細は、385 ページの 『エンタープライズ・アプリケーションのインポート』を参照してください。

新しいエンタープライズ・アプリケーションの開始: コマンド行ユーティリティー XMLConfig を使用して新しいエンタープライズ・アプリケーションをインポートした後、 WebSphere Application Server 管理コンソールを使って最新表示を実行し、それから新しいアプリケーションを開始することができます。

コンソールを最新表示して新しいアプリケーションを開始するには、以下のようにします。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere 管理可能ドメイン」**を拡張表示します。
3. **「ノード」**を強調表示します。
4. **「Refresh selected subtree (選択したサブツリーの最新表示)」**アイコンをクリックします。
5. 以下のようにして、WebSphere Commerce アプリケーションを開始します。
   - **「Application Servers (アプリケーション・サーバー)」**を拡張表示します。
   - WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Server -** *instanceName*」を右クリックして、**「開始」**を選択します。

ターゲット・ストアの新しいコードのテスト

WebSphere Application Server で実行しているストアで、変更済みのエンティティー Bean と新しいタスク・コマンドをテストするには、以下のようにします。

1. ブラウザーをオープンし、以下の URL を入力します。

   ```
   http://hostname/webapp/wcs/stores/servlet/StoreCatalogDisplay?
        storeId=store_Id&catalogId=catalog_Id&langId=-1
   ```

   ここで、*store_Id* はストアの ID で、*catalog_Id* はストアのカタログの ID です。

   注: エラー 500 が出される場合、エンタープライズ・アプリケーションを最新表示するために WebSphere Application Server を再始動する必要があります。
2. 「サービス」ヘッダーの下の**「登録」**リンクをクリックしてから、「新規顧客」ヘッダーの下の**「登録」**をクリックします。
   `wctester@wc` という電子メール・アドレスと `wctester1` というパスワードを使用して、新しい顧客を登録します。それ以外のフィールドにはテスト値を入力して、**「送信」**をクリックします。ブラウザーはオープンしたままにしておきます。
3. DB2 DB2 コマンド・センターをオープンして、以下の操作を行います。
   a. **「インタラクティブ」**タブをクリックします。
   b. **「コマンド」**フィールドで、以下のようにします。
      1) 以下のように入力します。

         ```
         connect to your_database_name
         ```

ここで、*your_database_name* はご使用の WebSphere Commerce データベースの名前です。「実行」アイコンをクリックします。

2) `select users_id from USERREG where LOGONID = 'wctester@wc'` と入力して、「実行」アイコンをクリックします。

c. 「クエリー結果」タブに、ステップ 2 で登録した顧客のエントリーが表示されます。顧客の USERS_ID の値をここにメモします。 _____

d. 新しく登録された顧客のボーナス・ポイントのバランスを更新します。「インタラクティブ」タブをクリックして、**「コマンド」**フィールドに、以下のように入力します。

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id
```

users_id は、ステップ 3c の値です。「実行」アイコンをクリックします。

4. Oracle 以下のようにして、テスト・ユーザーのボーナス・ポイントのバランスを更新してください。

a. 「Oracle SQL Plus」コマンド・ウィンドウをオープンします (**「スタート」** > **「プログラム」** > **「Oracle」** > **「アプリケーション開発」** > **「SQL Plus」**)。

b. **「ユーザー名」**フィールドに、ユーザーの Oracle ユーザー名を入力します。

c. **「パスワード」**フィールドに、 Oracle パスワードを入力します。

d. **「ホスト・ストリング」**フィールドに、接続ストリングを入力します。

e. `select users_id from USERREG where LOGONID = 'wctester@wc';` と入力して、テスト・ユーザーであることを判別します。

f. ステップ 2 で登録した顧客のエントリーが表示されます。顧客の USERS_ID の値をここにメモします。 _____

g. 以下のように入力して、新規登録された顧客のボーナス・ポイントのバランスを更新します。

```
update BONUS set BONUSPOINT = 1000 where MEMBERID = users_id;
```

users_id は、ステップ 4f の値です。

h. データベースの変更をコミットするには

```
commit;
```

と入力し、それから Enter を押して SQL ステートメントを実行します。

5. ブラウザーで、**「メンズ」**をクリックしてストアの「メンズ」セクションを表示します。

6. 特集のリンクをクリックして、商品ページを表示します。このページは、正規価格と、顧客のボーナス・ポイントの残高に基づく割引価格を表示します。

第 5 部 付録

付録 A. WebSphere Test Environment の開始と停止

このセクションでは、VisualAge for Java の中での WebSphere Test Environment の開始に関係するステップを説明します。一般的に、この環境を開始するには、以下のステップを実行する必要があります。

1. 永続ネーム・サーバーを開始する。
2. EJB サーバーを開始する。
3. サーブレット・エンジンを開始する。

これらのステップのおのおのについて、後続のセクションで説明します。

テスト環境を停止するには、以下のステップを逆順に行います。

注: WebSphere Test Environment のすべてのフィーチャーを利用するためには、WebSphere Test Environment を開始する前に、 WebSphere Application Server が稼働していないことを確認してください。 WebSphere Application Server を停止するための情報については、 *WebSphere Commerce インストール・ガイド* を参照してください。

永続ネーム・サーバーの開始と停止

永続ネーム・サーバーは、エンタープライズ Bean の検索要求をクライアントから受け取ります。すべてのエンタープライズ Bean は永続ネーム・サーバーで登録されます。

永続ネーム・サーバーを開始または停止するには、以下のようにします。

1. VisualAge for Java の**「ワークスペース」**メニューから、**「ツール」**>**「WebSphere Test Environment」**を選択します。
   WebSphere Test Environment Control Center がオープンします。
2. **「永続ネーム・サーバー」**をクリックして、以下のいずれかを実行します。
   - **「ネーム・サーバーの開始」**。
     “Server open for business” というメッセージがコンソール・ウィンドウに表示されるのを待ちます。このサーバーの開始には数分かかる場合があります。
   - **「ネーム・サーバーの停止」**。

EJB サーバーの開始と停止

EJB サーバーは、エンタープライズ Bean を使用するサーバー・アプリケーションの実行をサポートするための、ランタイム環境を提供する高水準プロセスまたはアプリケーションです。

EJB サーバーを開始または停止するには、以下のようにします。

1. EJB サーバー構成ウィンドウをオープンします (EJB タブをクリックして、**「EJB」**メニューから、**「オープン」**>**「サーバー構成」**を選択します)。
2. 開始または停止する EJB サーバー (**EJB Server {Server 1}** など) を右クリックして、以下のいずれかを実行します。
   - **「サーバーの開始」**を選択します。
     "Server open for business" というメッセージがコンソール・ウィンドウに表示されるのを待ちます (このサーバーの状況を見るには、コンソールで**「EJB サーバー」**を選択しなければならない場合もあります)。コンピューターにもよりますが、サーバーの開始には 10 分から 15 分かかることもあります。
   - **「サーバーの停止」**を選択します。

「コンソール」は IDE における Java プログラム用の標準入出力装置です。特定プログラムの出力を見るには、まず「すべてのプログラム」ペインでプログラムを選択します。すると、その出力が「出力」ペインに表示されます。

サーブレット・エンジンの開始と停止

サーブレット・エンジンは Web サーバーと WebSphere Application Server の機能性を統合して、テスト用の環境を作成します。

サーブレット・エンジンを開始または停止するには、以下のようにします。

1. VisualAge for Java の**「ワークスペース」**メニューから、**「ツール」**>**「WebSphere Test Environment」**を選択します。
   WebSphere Test Environment Control Center がオープンします。
2. **「サーブレット・エンジン」**をクリックして、以下のいずれかを実行します。
   - **「サーブレット・エンジンの開始」**
     サーブレット・エンジンが開始すると、コンソールに `***Servlet Engine is started ***` というメッセージが表示されます。
   - **「サーブレット・エンジンの停止」**

付録 B. デプロイメントに関する詳細情報

VisualAge for Java でカスタマイズ・コードを作成して WebSphere Test Environment 内でテストしたなら、今度はそれを WebSphere Test Environment の外で実行している WebSphere Commerce インスタンスにデプロイメントする必要があります。この WebSphere Commerce インスタンスは開発マシンでローカルに実行することもできますし、また、(同じオペレーティング・システムまたは別のオペレーティング・システムを使用して) 別のマシンで実行することもできます。

この付録では、カスタマイズしたコードを WebSphere Test Environment の外部で稼働する WebSphere Commerce インスタンスにデプロイメントするために必要なステップを説明します。デプロイメント・プロセスの高度なステップに関する理解を深めるには、193 ページの 『コードのデプロイメント』を、詳細情報についてはこの付録を参照してください。

統合ファイルシステムへのマッピング (iSeries)

400 このセクションは、ターゲットの WebSphere Commerce Server が iSeries プラットフォームで稼働する場合に限り適用されます。

ターゲットの WebSphere Commerce Server が iSeries プラットフォームで稼働している場合、開発マシン上のローカル・ドライブを iSeries サーバー上の統合ファイル・システム (IFS) にマップする必要があります。本書の続くセクションでは、IFS にマップされるローカル・ドライブを表すために *iSeries_drive* が使用されます。さらに、開発マシン (IFS にマップされていないもの) 上のローカル・ドライブを表すために *drive* が使用されます。

カスタマイズされたコマンドおよびデータ Bean の JAR ファイル

カスタマイズされたコマンドおよびデータ Bean のコードは、 WebSphere Commerce コードとは別のプロジェクトに保管する必要があります。 WebSphere Test Environment でのテストが完了したら、カスタマイズされたコマンドおよびデータ Bean のコードを含むプロジェクトの JAR ファイルを作成して、その JAR ファイルをターゲットの WebSphere Commerce Server の適切なディレクトリーに保管する必要があります。

カスタマイズされたコマンドおよびデータ Bean のコードを作成するには、以下のようにします。

1. VisualAge for Java のワークベンチで、**「プロジェクト」**タブを選択します。

2. カスタマイズされたコマンドおよびデータ Bean のコードを含んでいるプロジェクトを右クリックして、**「エクスポート」**を選択します。
「Export SmartGuide (エクスポート SmartGuide)」がオープンします。
3. **「JAR ファイル」**を選択し、**「次へ」**をクリックします。
4. **「Jar ファイル」**フィールドに、以下のように入力します。
drive:¥WebSphere¥CommerceServerDev¥*temp_directory*¥ *jar_file_name*_1.jar
drive:¥WebSphere¥CommerceServerDev¥*temp_directory*¥ は、 JAR ファイルを入れるための十分なフリー・スペースがある一時ディレクトリー、 *jar_file_name*_1.jar は JAR ファイルの名前に _1 が付加されたものを表します。
5. 以下のようにして、JAR ファイルの属性を選択します。

属性	値
クラス	チェックする
java	チェックしない
リソース	チェックする
beans	チェックする
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。
6. **「終了」**をクリックします。

作成されたインプリメンテーション JAR ファイルには完全なパッケージ命名情報が含まれていないため、 (VisualAge for Java 以外の) 別のパッケージ・ユーティリティーを使用して JAR ファイルを再パッケージ化する必要があります。このファイルを再パッケージ化するには、以下のようにします。

1. コマンド・ウィンドウで、前のステップで *jar_file_name*_1.jar を保管したディレクトリーに移動します。
2. mkdir temp1 と入力します。
3. cd temp1 と入力します。
4. パスを以下のように設定します。
set PATH=%PATH%;*drive*:¥WebSphere¥WebSphereStudio4¥bin;
drive は、WebSphere Studio のインストール先のドライブです。
5. jar xvf ../*jar_file_name*_1.jar と入力します。
6. jar cvf ../*jar_file_name*.jar * (名前から _1 を外すのを忘れないようにしてください) と入力します。
7. *drive*:¥WebSphere¥CommerceDev¥*temp_directory* ディレクトリーに移動します。
8. ローカルの WebSphere Commerce インスタンスにデプロイメントする場合は、*jar_file_name*.jar を以下のディレクトリーにコピーします。

drive:¥WebSphere¥AppServer¥installedApps¥
WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥WEB-INF¥lib

それ以外の場合は、すべてのファイル資産をターゲットの WebSphere Commerce Server に転送することが必要になるまで、 JAR ファイルを一時ディレクトリーに残しておきます。

新しいエンティティー Bean 用の JAR ファイルの作成

新しいエンティティー Bean を作成したら、すべての WebSphere Commerce コードとは別のプロジェクトにコードを保管する必要があります。さらに、そのプロジェクトは、カスタマイズされたコマンドおよびデータ Bean のすべてのプロジェクトとも別でなければなりません。新しいエンティティー Bean は、 WebSphere Commerce パブリック・エンティティー Bean がある EJB グループとは別の EJB グループに作成する必要があります。

新しいエンティティー Bean のデプロイメントを実行するには、2 つの JAR ファイルを作成します。最初の JAR ファイルは EJB 1.1 Export JAR で、「EJB」タブを選択しているときに有効なツールを使って作成されます。 2 番目の JAR ファイルにはエンティティー Bean のインプリメンテーション・コードが格納され、エンティティー Bean コードが入っているプロジェクトから作成されます。この 2 番目の JAR ファイルは、「VisualAge for Java ワークベンチ」で「プロジェクト」タブが選択されているときに有効なツールを使って作成されます。

EJB 1.1 Export JAR ファイルの作成

新しいエンティティー Bean の EJB 1.1Export JAR ファイルを作成するには、以下のようにします。

1. VisualAge for Java のワークベンチで、**「EJB」**タブを選択します。
2. カスタマイズしたエンティティー Bean がある EJB グループを右クリックし、**「エクスポート」>「EJB 1.1 JAR」**を選択します。
   「Export to an EJB 1.1 JAR file SmartGuide (EJB 1.1 JAR ファイルへのエクスポート SmartGuide)」がオープンします。
3. **「Jar ファイル」**フィールドに、
   drive:¥WebSphere¥CommerceServerDev¥temp_directory¥*jarFileName*_DT.jar
   と入力します。ここで、*drive*:¥WebSphere¥CommerceServerDev¥temp_directory は JAR ファイルを保管するための十分なフリー・スペースのある一時ディレクトリー、 *jarFileName*_DT.jar は JAR ファイル名に接尾部 _DT が付加されたものを表します。

JAR ファイルの名前には接尾部 “_DT” が付きます。つまり、これを WebSphere Commerce アプリケーションにデプロイメントする前に、WebSphere Application Server 付属の EJB デプロイメント・ツールを使ってこの JAR ファイルを実行する必要があります。

4. 以下のようにして、JAR ファイルの属性を選択します。

属性	値
クラス	チェックする
java	チェックしない
リソース	チェックする
ターゲット・データベース	DB2 DB2 データベースにデプロイメントしている場合は、以下のいずれかを選択します。 • Windows AIX Solaris Linux **DB2 for NT, V7.1** • 400 **DB2 for AS/400, V4** Oracle Oracle データベースにデプロイメントしている場合は、**「Oracle, V8」**を選択します。
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

5. **「終了」**をクリックします。

これで、JAR ファイルが作成されます。

インプリメンテーション JAR ファイルの作成

新しいエンティティー Bean のインプリメンテーション JAR ファイルを作成するには、以下のようにします。

1. VisualAge for Java のワークベンチで、**「プロジェクト」**タブを選択します。
2. カスタマイズされたエンティティー Bean コードを含んでいるプロジェクトを右クリックして、「**エクスポート**」を選択します。
   「Export SmartGuide (エクスポート SmartGuide)」がオープンします。
3. **「JAR ファイル」**を選択し、**「次へ」**をクリックします。
4. **「Jar ファイル」**フィールドに、
   drive:¥WebSphere¥CommerceServerDev¥*temp_directory*¥*jarFileName*_1.jar

と入力します。ここで、 *drive*:¥WebSphere¥CommerceServerDev¥*temp_directory* は JAR ファイルを保管するための十分なフリー・スペースのある一時ディレクトリー、 *jarFileName*_1.jar は JAR ファイル名に _1 が付加されたものを表します。

5. 以下のようにして、JAR ファイルの属性を選択します。

属性	値
クラス	チェックする
java	チェックしない
リソース	チェックする
beans	チェックする
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

6. **「終了」**をクリックします。

作成されたインプリメンテーション JAR ファイルには完全なパッケージ命名情報が含まれていないため、 (VisualAge for Java 以外の) 別のパッケージ・ユーティリティーを使用して JAR ファイルを再パッケージ化する必要があります。このファイルを再パッケージ化するには、以下のようにします。

1. コマンド・ウィンドウで、前のステップで *jarFileName*_1.jar を保管したディレクトリーに移動します。
2. mkdir temp1 と入力します。
3. cd temp1 と入力します。
4. パスを以下のように設定します。
   set PATH=%PATH%;*drive*:¥WebSphere¥WebSphereStudio4¥bin;
   drive は、WebSphere Studio のインストール先のドライブです。
5. jar xvf ../*jarFileName*_1.jar と入力します。
6. jar cvf ../*jarFileName*.jar * (名前から _1 を外すのを忘れないようにしてください) と入力します。
7. *drive*:¥WebSphere¥CommerceServerDev¥*temp_directory* ディレクトリーに移動します。
8. ローカルの WebSphere Commerce インスタンスにデプロイメントする場合は、 *jarFileName*_1.jar をディレクトリー
   drive:¥WebSphere¥CommerceServer¥temp¥lib
   にコピーしてください。そうでない場合は、すべてのファイル資産をターゲットの WebSphere Commerce Server に転送する必要が生じるときまで、 JAR ファイルを一時ディレクトリーに入れておいてください。

カスタマイズされた WebSphere Commerce エンティティー Bean 用の JAR ファイルの作成

パブリック WebSphere Commerce エンティティー Bean は、すべて拡張が可能です。これらの bean は以下の EJB グループにあります。

- WCSActrlEJBGroup
- WCSApproval
- WCSAuction
- WCSCatalog
- WCSCommon
- WCSContract
- WCSCoupon
- WCSFulfillment
- WCSInventory
- WCSMessageExtensions
- WCSOrder
- WCSOrderManagement
- WCSOrderStatus
- WCSPayment
- WCSPVCDevices
- WCSTaxation
- WCSUserTraffic
- WCSUser
- WCSUTF

いずれかのパブリック WebSphere Commerce エンティティー Bean を拡張する場合は、変更されたパブリック WebSphere Commerce エンティティー Bean が属している EJB グループの EJB 1.1 Export JAR ファイルを作成する必要があります。

EJB 1.1 Export JAR ファイルの作成

変更されたエンティティー Bean が属している EJB グループの EJB 1.1 Export JAR ファイルを作成するには、以下のようにします。

1. VisualAge for Java のワークベンチで、**「EJB」**タブを選択します。
2. カスタマイズしたエンティティー Bean がある EJB グループを右クリックし、**「エクスポート」>「EJB 1.1 JAR」**を選択します。
   「Export to an EJB 1.1 JAR file SmartGuide (EJB 1.1 JAR ファイルへのエクスポート SmartGuide)」がオープンします。

3. **「Jar ファイル」**フィールドに、
   drive:¥WebSphere¥CommerceServerDev¥*temp_directory*¥
   Cust_*EJBGroupName*-ejb_DT.jar
   と入力します。ここで、*drive*:¥WebSphere¥CommerceServerDev¥*temp_directory* は JAR ファイルを保管するための十分なフリー・スペースのある一時ディレクトリー、 Cust_*EJBGroupName*-ejb_DT.jar は JAR ファイル名に接尾部 _DT が付加されたものを表します。

JAR ファイルの名前には接尾部 "_DT" が付きます。つまり、これを WebSphere Commerce アプリケーションにデプロイメントする前に、WebSphere Application Server 付属の EJB デプロイメント・ツールを使ってこの JAR ファイルを実行する必要があります。

4. 以下のようにして、JAR ファイルの属性を選択します。

属性	値
クラス	チェックする
java	チェックしない
リソース	チェックする
ターゲット・データベース	DB2 DB2 データベースにデプロイメントしている場合は、以下のいずれかを選択します。 • Windows AIX Solaris Linux **DB2 for NT, V7.1** • 400 **DB2 for AS/400, V4** Oracle Oracle データベースにデプロイメントしている場合は、**「Oracle, V8」**を選択します。
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

5. **「終了」**をクリックします。

クライアント JAR ファイルの作成

クライアント JAR ファイルを作成するには、以下のようにします。

1. 「EJB」タブを選択し、すべての WebSphere Commerce EJB グループ (名前が WCS で始まるもの) を強調表示します。 これらすべてのグループが強調表示された状態で右クリックし、**「エクスポート」**>**「Client JAR (クライアント JAR)」**を選択し

ます。
「Export SmartGuide (エクスポート SmartGuide)」がオープンします。

2. **「JAR ファイル」**フィールドで、次のように入力します。
   drive:¥WebSphere¥CommerceServerDev¥*temp_directory*¥wcsejsclient.jar
3. 属性を以下のように選択します。

属性	値
beans	チェックする
クラス	チェックする
java	チェックしない
リソース	チェックする
.class ファイルにデバッグ属性を組み込む	チェックする

その他の属性については、デフォルト値を受け入れます。

4. **「終了」**をクリックします。

これで、JAR ファイルが作成されます。

ターゲットの WebSphere Commerce Server への資産の保管

カスタマイズ・コードに関連する資産を、ターゲットの WebSphere Commerce Server にコピーする必要があります。このような資産には、カスタマイズされたコマンド、データ Bean、およびエンティティー Bean の JAR ファイルが含まれます。また、カスタマイズをサポートする新しい JSP テンプレートやグラフィックスが含まれる場合もあります。

AIX Solaris Linux すべてのデプロイメント・ステップは、 *WebSphere Commerce インストール・ガイド* の「*インストール後スクリプトの実行*」 のセクションにあるステップの実行時に作成されたユーザーを使用して、ターゲット WebSphere Commerce Server 上で実行する必要があります。デフォルトでは、そのユーザーは `wasuser` です。さらに、ファイル資産 (たとえば、JAR ファイル) とそれらが置かれているディレクトリーに、このユーザーに付与されているファイルの読み取り、書き込み、および実行許可があることを確認する必要もあります。

400 `/QIBM/UserData/WebCommerce/instances/`*instanceName* および `/QIBM/UserData/WebCommerce/instances/`*instanceName*`/working` ディレクトリー用の権限にユーザー `QEJB` が含まれていることを確認します。両方のディレクトリーにこのユーザーを追加してください。その際、データ権限を `*RWX` に設定します。

次の表は、資産が保管されるターゲットの WebSphere Commerce Server (Windows NT または Windows 2000 が稼働する) 上の標準的なディレクトリーをまとめたものです。

表 12.

資産のタイプ	ターゲットの WebSphere Commerce Server 上でのディレクトリーの場所
コマンドおよびデータ Bean ロジック用の JAR ファイル	*drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥WEB-INF¥lib
VisualAge for Java を使用して作成された EJB 1.1 Export JAR	*drive*:¥WebSphere¥CommerceServer¥temp **注:** さらにこのファイルは、EJBDeploy ツールへの入力として渡され、 WebSphere Application Server V4.0 で使用できるデプロイメント・コードを生成します。
EJB クライアント・コードの JAR ファイル	*drive*:¥WebSphere¥CommerceServer¥temp¥lib **注:** このファイルは、EJBDeploy ツールではクラスパスでのみ使用されます。
EJB インプリメンテーション・コードの JAR ファイル	*drive*:¥WebSphere¥CommerceServer¥temp¥lib **注:** このファイルは、ファイル資産としてエンタープライズ・アプリケーションに追加されます。
JSP テンプレート	*drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥*storeDir*
イメージ	*drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥*storeDir*¥images

資産をターゲットの WebSphere Commerce Server に保管するには、以下のようにします。

1. コマンドとデータ Bean 用の JAR ファイルを見付けます。これらのファイルは、開発マシン上の以下のいずれかのディレクトリーにあるはずです。
   - Windows *drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥WEB-INF¥lib
   - Windows *drive*:¥WebSphere¥CommerceServerDev¥*temp_directory*
2. コマンドおよびデータ Bean 用の JAR ファイルを、ターゲットの WebSphere Commerce Server 上の以下のいずれかのディレクトリーにコピーします。
   - Windows *drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥WEB-INF¥lib
   - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib

- Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/WEB-INF/lib

3. 新規 EJB グループの EJB 1.1 Export JAR ファイルと、開発マシン上の以下のディレクトリーにある変更された WebSphere Commerce エンティティー Bean を見付けます。

   Windows *drive*:¥WebSphere¥CommerceServerDev¥*temp_directory*

4. EJB 1.1 Export JAR ファイルをターゲットの WebSphere Commerce Server 上の以下のいずれかのディレクトリーにコピーします。
   - Windows *drive*:¥WebSphere¥CommerceServer¥temp
   - AIX /usr/WebSphere/CommerceServer/temp
   - Solaris /opt/WebSphere/CommerceServer/temp
   - Linux /opt/WebSphere/CommerceServer/temp
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/temp
5. 新しいエンタープライズ Bean を作成した場合は、開発マシンの以下のディレクトリーにある、これらの bean のインプリメンテーション JAR ファイルを見付けてください。

   Windows *drive*:¥WebSphere¥CommerceServerDev¥*temp_directory*

6. この新しいエンタープライズ Bean 用のインプリメンテーション JAR ファイルを、ターゲットの WebSphere Commerce Server 上の以下のディレクトリーにコピーします。
   - Windows *drive*:¥WebSphere¥CommerceServer¥temp¥lib
   - AIX /usr/WebSphere/CommerceServer/temp/lib
   - Solaris /opt/WebSphere/CommerceServer/temp/lib
   - Linux /opt/WebSphere/CommerceServer/temp/lib
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/ temp/lib
7. 既存のg WebSphere Commerce エンティティー Beans に変更を加えた場合は、開発マシンの以下のディレクトリーにあるクライアント JAR ファイルを見付けてくだ

さい。

Windows *drive*:¥WebSphere¥CommerceServerDev¥*temp_directory*

8. このクライアント JAR ファイルを、ターゲットの WebSphere Commerce Server 上の以下のいずれかのディレクトリーにコピーします。
   - Windows *drive*:¥WebSphere¥CommerceServer¥temp¥lib
   - AIX /usr/WebSphere/CommerceServer/temp/lib
   - Solaris /opt/WebSphere/CommerceServer/temp/lib
   - Linux /opt/WebSphere/CommerceServer/temp/lib
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/ temp/lib
9. すべての JSP テンプレートを、ターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。
   - Windows *drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥*store_directory*
   - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*
   - 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*

   ここで、*store_directory* は保管用ディレクトリーです。
10. すべてのイメージを、ターゲットの WebSphere Commerce Server の以下のディレクトリーにコピーします。
    - Windows *drive*:¥WebSphere¥AppServer¥installedApps¥WC_Enterprise_App_*instanceName*.ear¥wcstores.war¥ *store_directory*¥images
    - AIX /usr/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/ *store_directory*/images
    - Solaris /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/ *store_directory*/images

- Linux /opt/WebSphere/AppServer/installedApps/WC_Enterprise_App_*instanceName*.ear/ wcstores.war/*store_directory*/images
- 400 /QIBM/UserData/WebASAdv4/*WAS_AdminInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear/wcstores.war/*store_directory*/images

ここで、*store_directory* は保管用ディレクトリーです。

ターゲット・データベースの更新

ターゲットの WebSphere Commerce Server が開発マシンと異なるデータベースを使用する場合、開発データベースに対して行ったすべての更新を、ターゲットの WebSphere Commerce Server で使用されるデータベース上で実行する必要があります。これには、新規コマンドまたは変更したコマンドの登録の更新、作成された追加テーブルの更新、および作成された新規リソース用のアクセス制御ポリシーの作成の更新が含まれます。

アクセス制御ポリシー (各種プラットフォーム用のコマンド構文およびディレクトリー許可要件を含む) のロードの詳細は、 *WebSphere Commerce アクセス・コントロール・ガイド* を参照してください。

400 SQL ステートメントを実行するためのユーティリティーは各自が用意してください。これを行う方法の 1 つは、 Client Access Express V5R1 (完全インストール) を使用することです。このユーティリティーをオープンするには、以下のようにします。

1. **「オペレーション・ナビゲーター」**をオープンします。
2. オペレーション・ナビゲーターがオープンしたら、特定のシステムにサインオンする必要があります。ターゲットの iSeries マシンを選択し、 WebSphere Commerce インスタンス・ユーザー・プロファイルおよびパスワードを選択してください。これにより、 WebSphere Commerce インスタンス・ユーザー・プロファイルが新しく作成されるすべてのテーブルを所有することになります。
3. パネルの左側でご使用のシステムをクリックしてから、**「DATABASE」**を右クリックし、ドロップダウン・リストから**「Run SQL Scripts (SQL スクリプトの実行)」**を選択します。
   「Run SQL Scripts (SQL スクリプトの実行)」ウィンドウがオープンします。このウィンドウを使用して、 SQL ステートメントで切り貼りを実行するか、SQL スクリプトをオープンします。「Connection/JDBC setup (接続/JDBC のセットアップ)」オプションを使用して、デフォルトのスキーマを設定することができます。

デプロイメント・コードの生成

このセクションでは、EJB デプロイメント・ツールを使用して、 EJB 1.1 Export JAR ファイルに含まれているエンタープライズ Bean のデプロイメント・コードを生成する方法を説明します。これは、WebSphere Application Server で提供されているツールです。

デプロイメント・コードを生成するには、以下のようにします。

1. ターゲットの WebSphere Commerce Server 上のコマンド・プロンプトで、以下のディレクトリーに移動します。
   - Windows *drive*:¥WebSphere¥CommerceServer¥temp
   - AIX /usr/WebSphere/CommerceServer/temp
   - Solaris /opt/WebSphere/CommerceServer/temp
   - Linux /opt/WebSphere/CommerceServer/temp
2. 400 ターゲットの WebSphere Commerce Server 上のコマンド・プロンプトで、以下のコマンドを入力します。

   ```
   STRQSH
   cd /QIBM/UserData/WebCommerce/instances/instanceName/temp
   ```
3. 以下のコマンドを入力して、このツールをシステムに一時的に追加します。
   - Windows PATH=*drive*:¥WebSphere¥AppServer¥deploytool;%PATH%
   - AIX PATH=/usr/WebSphere/AppServer/deploytool:$PATH
   - Solaris PATH=/opt/WebSphere/AppServer/deploytool:$PATH
   - Linux PATH=/opt/WebSphere/AppServer/deploytool:$PATH
   - 400 PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH

     CP=*ClassPathOfDepJARFiles*

     ClassPathOfDepJARFiles はすべての従属 JAR ファイルのクラスパスです。既存の WebSphere Commerce エンタープライズ Bean を変更したときには、wcsejsclient.jar、wcsejbimpl.jar、および xml4j.jar ファイルを従属 JAR ファイルのクラスパスに含める必要があります。たとえば、以下の例は、既存の WebSphere Commerce エンタープライズ Bean を変更した場合のクラスパスを示しています。

     ```
     CP=/QIBM/UserData/WebCommerce/instances/instanceName/temp/lib/
     wcsejsclient.jar:
     /QIBM/UserData/WebASAdv4/WAS_AdminInstanceName/installedApps/
     ```

```
WC_Enterprise_App_instanceName.ear/lib/wcsejbimpl.jar:
/QIBM/UserData/WebASAdv4/WAS_AdminInstanceName/installedApps/
WC_Enterprise_App_instanceName.ear/lib/xml4j.jar
```

注: 上記のクラスパスの例にある改行は、表示目的に過ぎません。クラスパス情報は 1 行に入力してください。

4. Solaris Linux コマンド行インターフェースの文字制限を超えないようにするには、スクリプトを使用して ejbdeploy コマンドを呼び出します。このスクリプトを作成してコマンドを呼び出すには、以下のようにします。

   a. 以下のディレクトリーに移動します。

      ```
      /opt/WebSphere/AppServer/bin
      ```

   b. 新規ファイルを作成し、それにスクリプト用の名前を適宜付けます。たとえば、myejbd.sh というファイル名にします。

   c. myejbd.sh ファイル内で以下の行を見付けます。

      ```
      #!/bin/sh
      ./ejbdeploy.sh EJBGroupJARFile WorkingDir OutputJARFile
      -nowarn -keep -35 -cp ClassPathOfDepJARFiles
      ```

      ここで、変数の定義はステップ 5 (ただし、このステップは実行しません) の定義と同じものです。以下の例は、スクリプト・ファイルの内容と、そこに含まれる変数の値を示しています。

      ```
      #!/bin/sh
      ./ejbdeploy.sh
      /opt/WebSphere/CommerceServer/temp/CustomizedWCSUserDeployed_DT.jar
      . /opt/WebSphere/CommerceServer/temp/CustomizedWCSUserDeployed.jar
      -nowarn -keep -35 -cp "/opt/WebSphere/CommerceServer/temp/lib/
      wcsejsclient.jar:/opt/WebSphere/AppServer/installedApps/
      WC_Enterprise_App_demo.ear/lib/wcsejbimpl.jar:/opt/WebSphere/
      AppServer/installedApps/WC_Enterprise_App_demo.ear/lib/xml4j.jar"
      ```

      注: ./ejbdeploy.sh コマンドの後で改行がなされているのは、すべて表示上の都合に過ぎません。ファイルでは、./ejbdeploy.sh コマンドの後に改行を入れないでください。

      スクリプトを保管し、それを実行可能にします。

   d. 以下のように入力して、スクリプトを呼び出します。

      ```
      ./myejbd.sh
      ```

5. Windows AIX 400 以下のように、ejbdeploy コマンドを入力します。

   Windows 400

   ```
   ejbdeploy EJBGroupJARFile_DT WorkingDir OutputJARFile -nowarn -keep -35 -cp
      ClassPathOfDepJARFiles
   ```

   AIX

```
/usr/WebSphere/AppServer/bin/ejbdeploy.sh EJBGroupJARFile_DT WorkingDir
OutputJARFile -nowarn -keep -35
-cp ClassPathOfDepJARFiles
```

ここで、

- *EJBGroupJARFile*_DT は、EJB グループの JAR ファイルの名前です。たとえば、WCSUser EJB グループで bean の変更を行う場合、 Windows ベースのプラットフォームでの使用例は次のようになります。

  ```
  drive:¥WebSphere¥CommerceServer¥temp¥CustomizedWCSUser_DT.jar
  ```

  400 iSeries プラットフォームの使用例は次のようになります。

  ```
  /QIBM/UserData/WebCommerce/instances/instanceName/temp/
  CustomizedWCSUser_DT.jar
  ```

- *WorkingDir* は、コード生成に必要な一時ファイルが保管されているディレクトリーの名前です。
- *OutputJARFile* は出力 JAR ファイルの完全修飾名です。たとえば、CustomizedWCSUserDeployed.jar のように入力できます。
- -nowarn は、警告および情報メッセージを抑制するためのオプショナル・パラメーターです。
- -keep は、 ejbdeploy コマンドの実行後も作業ディレクトリーを保存するためのオプショナル・パラメーターです。
- -35 は必須パラメーターであり、 WebSphere Application Server バージョン 3.5 付属の EJB デプロイメント・ツールが使用したのと同じ CMP エンティティー Bean 用トップダウン・マッピング規則を使用します。
- -cp *ClassPathOfDepJARFiles* はすべての従属 JAR ファイルのクラスパスです。既存の WebSphere Commerce エンタープライズ Bean を変更したときには、wcsejsclient.jar、wcsejbimpl.jar、および xml4j.jar ファイルを従属 JAR ファイルのクラスパスに含める必要があります。たとえば、以下の例は、既存の WebSphere Commerce エンタープライズ Bean を変更した場合のクラスパスを示しています。

  ```
  "drive:¥WebSphere¥CommerceServer¥temp¥lib¥wcsejsclient.jar;
  drive:¥WebSphere¥AppServer¥installedApps¥
  WC_Enterprise_App_instanceName.ear ¥lib¥wcsejbimpl.jar;
  drive:¥WebSphere¥AppServer¥installedApps¥
  WC_Enterprise_App_instanceName.ear¥lib¥xml4j.jar;"
  ```

  注: 400 クラスパスの値の場合は、-cp $CP を入力します。ここで、CP は適切なクラスパス・エントリーで事前に定義されています。さらに、引用符も不要です。

エンティティー Bean のトランザクション分離レベルの変更

このセクションでは、 modifyIsolationLevel コマンド行ユーティリティーを使用して、エンティティー Bean のトランザクション分離レベルをデータベースのタイプに合わせて設定する方法を説明します。

modifyIsolationLevel コマンドを実行するには、以下のようにします。

1. ターゲットの WebSphere Commerce Server で、コマンド・プロンプトを使用して以下のディレクトリーに移動します。
   - Windows *drive*:¥WebSphere¥CommerceServer¥bin
   - AIX /usr/WebSphere/CommerceServer/bin
   - Solaris /opt/WebSphere/CommerceServer/bin
   - Linux /opt/WebSphere/CommerceServer/bin
2. 400 以下のコマンドを入力します。

   ```
   STRQSH
   cd /QIBM/ProdData/WebCommerce/bin
   ```
3. modifyIsolationLevel コマンドを実行してください。このコマンドは、通常、以下のような構文を持ちます。

   Windows AIX 400

   ```
   modifyIsolationLevel -jarFile jar_file_name.jar
      -logFile log_file_name -dbType db_type
   ```

   Solaris Linux

   ```
   ./modifyIsolationLevel.sh -jarFile jar_file_name.jar
      -logFile log_file_name -dbType db_type
   ```

   ここで

   - *jar_file_name.jar* は、カスタマイズされたコードを含む JAR ファイルの名前です。
   - *log_file_name* は、情報がログに記録されるべき完全修飾ファイル名です。
   - *db_type* はご使用のデータベースのタイプです。 DB2 または ORACLE のいずれかを入力します。

   以下は、Windows プラットフォームで使用するためにすべての値が指定された modifyIsolationLevel コマンドの例です。

   ```
   modifyIsolationLevel -jarFile
      D:¥WebSphere¥CommerceServer¥temp¥CustomizedWCSUserDeployed.jar
      -logFile D:¥WebSphere¥CommerceServer¥instances¥demo¥logs¥output.log
      -dbType DB2
   ```

400 以下は、iSeries で使用するためにすべての値が指定された modifyIsolationLevel コマンドの例です。

```
modifyIsolationLevel -jarFile
   /QIBM/UserData/WebCommerce/instances/instanceName/temp/
   CustomizedWCSUserDeployed.jar -logFile /QIBM/UserData/WebCommerce/
   instances/instanceName/logs/output.log -dbType DB2
```

上記の例で改行がなされているのは、表示上の都合に過ぎません。

注: パラメーターの名前はケース・センシティブです。つまり、jarFile と jarfile は同じではありません。パラメーターの名前は正しく入力してください。

コマンド・ウィンドウに例外が表示されない場合は、コマンドが正常に実行されました。完了後は、デプロイメント JAR ファイルのタイム・スタンプが変更していることに注意してください。

現在の WebSphere Commerce エンタープライズ・アプリケーションのエクスポート

このセクションでは、WebSphere Application Server 管理コンソールを使用して、現在の WebSphere Commerce エンタープライズ・アプリケーションをエクスポートする方法を説明します。

現在のエンタープライズ・アプリケーションを WebSphere Application Server からエクスポートするには、以下のようにします。

1. ターゲットの WebSphere Commerce Server 上に以下のディレクトリーがあることを確認します。
   - Windows *drive*:¥WebSphere¥CommerceServer¥working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

   このディレクトリーが存在していない場合は作成してください。

   注: AIX Solaris Linux /working ディレクトリーの許可が、*WebSphere Commerce インストール・ガイド* の“インストール後スクリプトの実行”の項にあるステップの実行時に作成されたユーザーに設定されていることを確認します。

400 /QIBM/UserData/WebCommerce/instances/*instanceName* および /QIBM/UserData/WebCommerce/instances/*instanceName*/working ディレクトリー用の権限にユーザー QEJB が含まれていることを確認します。両方のディレクトリーにこのユーザーを追加してください。その際、データ権限を *RWX に設定します。

2. WebSphere Application Server 管理コンソールをオープンします。
3. 「**WebSphere 管理可能ドメイン**」を拡張表示します。
4. 「**Enterprise Application (エンタープライズ・アプリケーション)**」を拡張表示します。
5. WebSphere Commerce アプリケーションを右クリックします。たとえば、**demo** アプリケーションを拡張表示して、「**Export Application (アプリケーションのエクスポート)**」を選択します。
6. 「**Export directory (エクスポート先ディレクトリー)**」フィールドに、以下のように入力します。
   - Windows *drive*:¥WebSphere¥CommerceServer¥working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working

   これによって、すべてのリソースを含むアプリケーション全体が WC_Enterprise_App_*instanceName*.ear ファイルにエクスポートされます (ここで、*instanceName* は WebSphere Commerce インスタンスの名前)。

エンタープライズ Bean の構成情報のエクスポート

このセクションでは、 XMLConfig コマンド行ユーティリティーの -export オプションを使用して、既存のエンタープライズ・アプリケーションに含まれているエンタープライズ Bean の構成情報をエクスポートする方法を説明します。

既存のアプリケーションにある bean に関する情報をエクスポートした後、そのアプリケーションに追加する新規 bean に関する情報は手動で追加する必要があります。

この構成情報をエクスポートするには、以下のようにします。

1. was.export.app.xml ファイルをターゲットの WebSphere Commerce Server 上の以下のディレクトリーから
   - Windows *drive*:¥WebSphere¥CommerceServer¥xml¥config
   - AIX /usr/WebSphere/CommerceServer/xml/config

- Solaris /opt/WebSphere/CommerceServer/xml/config
- Linux /opt/WebSphere/CommerceServer/xml/config
- 400 /QIBM/ProdData/WebCommerce/xml/config

ターゲットの WebSphere Commerce Server 上の以下のディレクトリーにコピーします。

- Windows *drive*:¥WebSphere¥CommerceServer¥working
- AIX /usr/WebSphere/CommerceServer/working
- Solaris /opt/WebSphere/CommerceServer/working
- Linux /opt/WebSphere/CommerceServer/working
- 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/working
  - ここで、*instanceName* は WebSphere Commerce インスタンスの名前です。

2. Windows AIX Solaris Linux テキスト・エディターで was.export.app.xml ファイルをオープンします。このファイルの中で、すべての $Enterprise_Application_Name$ を以下に置換します。

   WebSphere Commerce Enterprise Application - *instanceName*

   ここで、*instanceName* は WebSphere Commerce インスタンスの名前です (たとえば demo)。このファイルを保管します。

   注: 挿入する値は、WebSphere Application Server 管理コンソールに表示されるインスタンスの情報と一致していなければなりません。

3. 400 テキスト・エディターで was.export.app.xml ファイルをオープンします。このファイルの中で、すべての $Enterprise_Application_Name$ を以下に置換します。

   instanceName - WebSphere Commerce Enterprise Application

   ここで、 *instanceName* は WebSphere Commerce インスタンスの名前です (たとえば demo)。このファイルを保管します。

   注: 挿入する値は、WebSphere Application Server 管理コンソールに表示されるインスタンスの情報と一致していなければなりません。

4. Windows AIX Solaris Linux コマンド・プロンプトで、以下のディレクトリーに移動します。

   - Windows *drive*:¥WebSphere¥CommerceServer¥working
   - AIX /usr/WebSphere/CommerceServer/working

- Solaris /opt/WebSphere/CommerceServer/working
- Linux /opt/WebSphere/CommerceServer/working

5. 400 コマンド・プロンプトで、以下を入力します。

```
STRQSH
PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH
cd /QIBM/UserData/WebCommerce/instances/instanceName/working
```

6. Windows AIX Solaris Linux XMLConfig ツールを起動し、以下のコマンドを入力して、部分的なエクスポートを実行します。

Windows

```
xmlConfig -export OutputFile.xml -partial was.export.app.xml
   -adminNodeName wasHostName
```

AIX

```
/usr/WebSphere/AppServer/bin/XMLConfig.sh  -export OutputFile.xml
   -partial was.export.app.xml -adminNodeName wasHostName
   -nameServiceHost wasHostName -nameServicePort wasAdminPort
```

Solaris Linux

```
/opt/WebSphere/AppServer/bin/XMLConfig.sh  -export OutputFile.xml
   -partial was.export.app.xml -adminNodeName wasHostName
   -nameServiceHost wasHostName -nameServicePort wasAdminPort
```

ここで

- *wasHostName* は、 WebSphere Application Server において現在のエンタープライズ・アプリケーションが入っているノードの名前です。
- OutputFile.xml は、このコマンドの実行結果として生成されるファイルの名前です。 was.export.app.xml は、ステップ 2 で変更したファイルです。
- *wasAdminPort* は、WebSphere Application Server 管理ポートです。

7. 400 XMLConfig ツールを起動し、以下のコマンドを入力して、部分的なエクスポートを実行します。

```
xmlConfig -export OutputFile.xml -partial was.export.app.xml
-adminNodeName wasHostName -nameServiceHost wasHostName
-nameServicePort wasAdminPort -instance wasInstanceName
```

ここで

- *wasHostName* は、 WebSphere Application Server において現在のエンタープライズ・アプリケーションが入っているノードの名前です。

注: *wasHostName* の値には大文字小文字の区別があり、 TCP/IP 構成での値と一致していなければなりません。 (コマンド行プロセッサーを使用して、CFGTCP オプション 12 にアクセスし、ホスト名を検査してください。)

- OutputFile.xml は、このコマンドの実行結果として生成されるファイルの名前です。 was.export.app.xml は、ステップ 3 で変更したファイルです。
- *wasAdminPort* は、WebSphere Application Server 管理ポートです。
- *wasInstanceName* は、WebSphere Application Server インスタンス名です。

現在のエンタープライズ・アプリケーションに含まれている各 bean の構成情報をエクスポートした後、アプリケーションに追加する各エンタープライズ Bean について記述した新しいスタンザを追加します。たとえば、“Bonus” という新しいエンティティーがある場合は、この Bonus bean について記述したスタンザを作成しなければなりません。加えて、構成ファイル内の変数を置き換えて、エンタープライズ・アプリケーションの .ear ファイルの正確な名前を指定する必要があります。

OutputFile.xml ファイルにこれらの更新を適用するには、以下のようにします。

1. ターゲットの WebSphere Commerce Server 上の以下のディレクトリーに移動します。
   - Windows *drive*:¥WebSphere¥CommerceServer¥working
   - AIX /usr/WebSphere/CommerceServer/working
   - Solaris /opt/WebSphere/CommerceServer/working
   - Linux /opt/WebSphere/CommerceServer/working
   - 400 /QIBM/UserData/WebCommerce/instances/ *instanceName*/working
2. テキスト・エディターで OutputFile.xml ファイルをオープンします。
3. <ear-file-name> タグを検索して、値を以下のように置き換えます。
   - Windows *drive*:¥WebSphere¥CommerceServer¥working¥ WC_Enterprise_App_*instanceName*.ear
   - AIX /usr/WebSphere/CommerceServer/working/ WC_Enterprise_App_*instanceName*.ear
   - Solaris /opt/WebSphere/CommerceServer/working/ WC_Enterprise_App_*instanceName*.ear
   - Linux /opt/WebSphere/CommerceServer/working/ WC_Enterprise_App_*instanceName*.ear
   - 400 /QIBM/UserData/WebCommerce/instances/*instanceName*/ working/WC_Enterprise_App_*instanceName*.ear

4. さらに、エンタープライズ・アプリケーションに追加するそれぞれの新規 bean に新しいスタンザを追加する必要があります。以下の例は、“Bonus” という新規の bean を追加する方法を示しています。

Windows AIX Solaris Linux

```
<ejb-module name="yourEJBGroup">
   <jar-file>yourDeployedJarFile.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         WebSphere Commerce Server - instanceName/
      </application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/WebSphere Commerce DB2 DataSource instanceName
            </jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="BeanBindingName">
         <jndi-name>instanceNameJNDINameOfBean</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

400

```
<ejb-module name="yourEJBGroup">
   <jar-file>yourDeployedJarFile.jar</jar-file>
   <module-install-info>
      <application-server-full-name>/NodeHome:$hostName$/EJBServerHome:
         instanceName - WebSphere Commerce Server/
      </application-server-full-name>
   </module-install-info>
   <ejb-module-binding>
      <data-source>
         <jndi-name>jdbc/instanceName WebSphere Commerce DB2 DataSource
            </jndi-name>
         <default-user>user</default-user>
         <default-password>password</default-password>
      </data-source>
      <enterprise-bean-binding name="BeanBindingName">
         <jndi-name>instanceNameJNDINameOfBean</jndi-name>
      </enterprise-bean-binding>
   </ejb-module-binding>
</ejb-module>
```

ここで

- *yourEJBGroup* は、エンタープライズ・アプリケーションに追加する bean が属する EJB グループの名前です。

- *yourDeployedJarFile* は、 EJB グループのデプロイメント・コードが含まれている JAR ファイルの名前です。
- *instanceName* は、コードのデプロイメント先となる WebSphere Commerce インスタンスの名前です。
- *user* はデータベースのユーザー名です。
- *password* はデータベースのユーザーのパスワードです。
- *BeanBindingName* は、エンタープライズ Bean のバインディング名です。たとえば、Bonus という bean の場合は Bonus_Binding になります。
- *instanceNameJNDINameOfBean* は、エンタープライズ Bean の JNDI 名の前に WebSphere Commerce インスタンスが付加されたものです。この JNDI 名は、エンタープライズ Bean の名前と正確に一致していなければなりません。たとえば、値は democom/ibm/commerce/sample/objects/Bonus のようになります。この例では、“demo” はインスタンス名で、
  “com/ibm/commerce/sample/objects/Bonus” はエンタープライズ Bean の JNDI 名です。この値は 1 行に入力しなければなりません。
  JNDI 名は、VisualAge for Java かアプリケーション・アセンブリー・ツールで調べることができます。
  VisualAge for Java を使用して JNDI 名を確認するには、以下のようにします。
  a. bean を右クリックし、**「プロパティー」**を選択します。
     JNDI 名が表示されます。

JNDI 名の検査にはアプリケーション・アセンブリー・ツールを使用することができます。しかし、それには新しいエンタープライズ Bean をエンタープライズ・アプリケーションにすでにアセンブルしている必要があります。アプリケーション・アセンブリー・ツールにアクセスするには、 WebSphere Application Server 管理コンソールの「ツール」メニューを使用します。

アプリケーション・アセンブリー・ツールを使用して JNDI 名を確認するには、以下のようにします。

a. bean が入っている .ear ファイルをオープンします。
b. bean が入っている EJB モジュールを拡張表示します。
c. bean を選択します。
d. **「Bindings (バインディング)」**タブをクリックします。
   JNDI 名が表示されます。

なお、このどちらの方法を使用して JNDI 名を表示する場合でも、XML ファイルに情報を追加するときには、先頭に WebSphere Commerce インスタンスの名前を付けるのを忘れないようにしてください。

注:

a. 上記のスタンザにある改行は、表示目的に過ぎません。

b. **$hostName$** の値は、必ず現在の管理ノード・サーバー名と一致させてください。さらに、この行に改行がないことも確認してください。

c. `<application-server-full-name>` の指定は複数行にまたがることができません。

d. Oracle データベースを使用している場合は、以下のようにしてデータ・ソース情報を変更する必要があります。前述のコード断片からとられた以下の行を変更します。

```
<jndi-name>jdbc/WebSphere Commerce DB2 DataSource instanceName
   </jndi-name>
```

以下のように変更します。

```
<jndi-name>jdbc/WebSphere Commerce Oracle DataSource instanceName
   </jndi-name>
```

5. 変更された WebSphere Commerce エンティティー Bean をでプロ委する場合、変更された bean のデプロイメント・コードを含む JAR ファイルの名前を反映するように、それらの bean のスタンザを更新する必要があります。

6. `OutputFile.xml` ファイルを保管します。

新しいエンタープライズ Bean のエンタープライズ・アプリケーションへのアセンブル

このセクションでは、アプリケーション・アセンブリー・ツールを使用して、新しいエンタープライズ Bean を既存のエンタープライズ・アプリケーションにアセンブルする方法を説明します。

400 アプリケーション・アセンブリー・ツールは Windows プラットフォームで実行するため、完全修飾パス名を入力するようにプロンプトが出されたら、iSeries IFS にマップしたドライブを参照する必要があります。これを *iSeries_drive* といいます。

AIX Solaris Linux `assembly.sh` ファイルのメモリーのヒープ・サイズを大きくし、新規または変更後の `.ear` ファイルを保管する際にメモリーが不足することのないようにしてください。

このファイルは以下のディレクトリーにあります。

- AIX `/usr/WebSphere/AppServer/bin`
- Solaris `/opt/WebSphere/AppServer/bin`
- Linux `/opt/WebSphere/AppServer/bin`

メモリーのヒープ・サイズを大きくするには、次の行を変更してください。

```
$JAVA_HOME/jre/bin/java
```

次のように表示されます。

```
$JAVA_HOME/jre/bin/java -mx512M
```

このステップでは、現在の WebSphere Commerce エンタープライズ・アプリケーションのエクスポートのセクションで作成した、エンタープライズ・アプリケーションの `.ear` ファイルをアプリケーション・アセンブラー・ツールでオープンします。このファイルをツールでオープンした後、以下のタスクを実行してエンタープライズ・アプリケーションに新しいエンティティー Bean を追加してください。

1. 新しいエンティティー Bean が含まれている EJB グループをインポートします。新しい EJB グループの JAR ファイルは、エンタープライズ・アプリケーションの EJB Module セクション内に保管されます。
2. インプリメンテーション JAR ファイルが含まれるように、新しいエンティティー Bean のクラスパスを設定します。
3. インプリメンテーション JAR ファイルをアプリケーションに追加します。この JAR ファイルは、エンタープライズ・アプリケーションの Files セクション内に保管されます。
4. 新しいエンティティー Bean に含まれるメソッド用の WebSphere Application Server セキュリティーをセットアップします。

新しい EJB グループをエンタープライズ・アプリケーションにアセンブルするには、以下のようにします。

1. Windows AIX Solaris Linux ターゲットの WebSphere Commerce Server 上で、以下のようにして現在のエンタープライズ・アプリケーションのバックアップを取ります。
   a. コマンド・プロンプトで、以下のディレクトリーに移動します。

- Windows *drive*:¥WebSphere¥CommerceServer¥working
- AIX /usr/WebSphere/CommerceServer/working
- Solaris /opt/WebSphere/CommerceServer/working
- Linux /opt/WebSphere/CommerceServer/working

b. 既存の WC_Enterprise_App_*instanceName*.ear ファイルのコピーを作成し、それに WC_Enterprise_App_*instanceName*.ear.bak という名前を付けます。

2. 400 以下のようにして、現在のエンタープライズ・アプリケーションのバックアップを取ります。

a. コマンド・プロンプトで、以下を入力します。

```
STRQSH
cd /QIBM/UserData/WebCommerce/instances/instanceName/working
cp WC_Enterprise_App_instanceName.ear
WC_Enterprise_App_instanceName.ear.bak
```

b. ローカル・クライアント・マシンと WebSphere Application Server を実行する iSeries マシンの間のデータ転送による待機時間が不必要に長引くのを避けるには、ローカル・クライアント・マシン上で以下のディレクトリーを作成してから、 WC_Enterprise_App_*instanceName*.ear ファイルをこのディレクトリーにコピーします。

drive:¥WebSphere¥CommerceServer¥working
ここで、*drive* はローカル・ドライブです。

注: ローカル・マシン上に *drive*:¥WebSphere¥CommerceServer¥working ディレクトリーがなければ、ここで作成してください。

3. WebSphere Application Server 管理コンソールをオープンします。
4. **「ツール」**メニューから、**「Application Assembly Tool」**を選択します。
5. 「ウェルカム」ウィンドウがオープンしたら、**「キャンセル」**を選択して、このウィンドウをクローズします。
6. 以下のようにして、作業対象のエンタープライズ・アプリケーションをオープンします。

a. **「ファイル」**メニューから、**「オープン」**を選択します。

b. **「File name (ファイル名)」**フィールドで、以下を入力します。

- Windows *drive*:¥WebSphere¥CommerceServer¥working¥WC_Enterprise_App_*instanceName*.ear
- AIX /usr/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear

- Solaris /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- Linux /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- 400 *drive*:¥WebSphere¥CommerceServer¥working¥WC_Enterprise_App_*instanceName*.ear

それから**「オープン」**をクリックします。アプリケーションがオープンするまで待って、次のステップに進みます。これには数分かかります。

7. **「EJB Modules (EJB モジュール)」**を右クリックして、**「Import (インポート)」**を選択します。
8. **「File name (ファイル名)」**フィールドで、以下を入力します。
   - Windows *drive*:¥WebSphere¥CommerceServer¥temp¥*yourDeployedJarFile*.jar
   - AIX /usr/WebSphere/CommerceServer/temp/*yourDeployedJarFile*.jar
   - Solaris /opt/WebSphere/CommerceServer/temp/*yourDeployedJarFile*.jar
   - Linux /opt/WebSphere/CommerceServer/temp/*yourDeployedJarFile*.jar
   - 400 *iSeries_drive*:¥QIBM¥UserData¥WebCommerce¥instances¥*instanceName*¥temp¥*yourDeployedJarFile*.jar

   ここで
   - *yourDeployedJarFile* は、 EJB グループのデプロイメント・コードが含まれている JAR ファイルの名前です。
   - *iSeries_drive* は、 iSeries IFS にマップされるローカル・ドライブです。

   「オープン」をクリックし、「Confirm Values (値の確認)」ウィンドウで**「OK」**をクリックします。
9. *yourDeployedJarFile*.jar ファイルがインポートされたら、 *yourEJBGroup* EJB グループ (*yourEJBGroup* は EJB グループの名前) までスクロールしてこのグループを選択します。
   すると、このグループに関する情報が右の画面に表示されます。
10. 新しいエンタープライズ Bean の classpath フィールドに、すべての従属 JAR ファイルを入力します。たとえば、対応するインプリメンテーション JAR ファイルと、 WebSphere Commerce エンティティー Bean のインプリメンテーション JAR を次のように入力することができます。

    lib/*yourImplJarFile*.jar lib/wcsejbimpl.jar
11. **「適用」**をクリックします。
12. 以下のようにして、EJB グループのインプリメンテーション JAR ファイルをアプリケーションに追加します。

a. エンタープライズ・アプリケーションの**「Files (ファイル)」**ノードを右クリックして、**「Add Files (ファイルの追加)」**を選択します。 (エンタープライズ・アプリケーションの**「Files (ファイル)」**ノードは、階層ツリーの下の方にあります。エンタープライズ・アプリケーション内には、コンポーネント用の他の「Files (ファイル)」ノードも存在しますが、必ずアプリケーション全体の「Files (ファイル)」ノードを選択してください。)

b. 「Add Files (ファイルの追加)」ウィンドウで、**「ブラウズ」**をクリックします。

c. 以下のディレクトリーに移動します。

- Windows *drive*:¥WebSphere¥CommerceServer¥temp
- AIX /usr/WebSphere/CommerceServer/temp
- Solaris /opt/WebSphere/CommerceServer/temp
- Linux /opt/WebSphere/CommerceServer/temp
- 400 *iSeries_drive*:¥QIBM¥UserData¥WebCommerce¥instances¥*instanceName*¥temp

d. このディレクトリーを強調表示して、**「選択」**をクリックします。

e. 「Add Files (ファイルの追加)」ウィンドウに戻ります。一時ディレクトリーの内容が表示されることを確認します。 lib ディレクトリーを強調表示します。lib ディレクトリーの内容が右側の画面に表示されます。

f. 右側の画面で *yourImplJarFile* ファイルを選択し、**「追加」**をクリックします。このファイルが、「Selected Files (選択済みファイル)」画面に表示されます。

g. **「OK」**をクリックします。

13. 以下のようにして、エンティティー Bean のセキュリティーを構成します。

a. 「EJB Modules (EJB モジュール)」ノードを拡張表示し、 *YourEJBGroup* ノードを見付けてこれを拡張表示します。

b. **「Entity Beans (エンティティー Bean)」**を拡張表示します。

c. *yourEntityBean* を拡張表示します (*yourEntityBean* はエンティティー Bean の名前)。

d. **「Method Extensions (メソッドの拡張機能)」**をクリックし、右側の画面で以下のようにします。

1) **「Advanced (拡張)」**タブをクリックします。

2) **「Security identity (セキュリティー ID)」**が選択されていることを確認します。

3) 各メソッドごとに、**「Use identity of EJB server (EJB サーバーの ID を使用する)」**が選択されていることを確認します。

4) **「適用」**をクリックします (変更を加えた場合)。

e. 左側のナビゲーション画面で、 **YourEJBGroup** EJB グループの**「Security Roles (セキュリティー役割)」**を右クリックして**「新規」**を選択し、以下のようにします。

1) **「名前」**フィールドに WCSecurityRole と入力して、**「適用」**をクリックします。この役割がすでに存在している場合、このステップを実行する必要はありません。

f. 左側のナビゲーション画面で、 *YourEJBGroup* EJB グループの**「Method Permissions (メソッド許可)」**を右クリックして**「新規」**を選択し、以下のようにします。

1) **「Method Permission Name (メソッド許可名)」**フィールドで、WCMethodPermission と入力します。

2) **「Methods (メソッド)」**選択領域で**「追加」**をクリックします。「Add Methods (メソッドの追加)」ウィンドウがオープンします。

3) *yourDeployedJarFile*.jar、**「Bonus」**の順に拡張表示してから、メソッドの**「ホーム」**リストと**「リモート」**リストをそれぞれ拡張表示します。

4) シフト鍵を押したまま、すべてのホーム・メソッドを選択して、**「OK」**をクリックします。

5) メソッドの選択プロセスを繰り返し、リモート・メソッドを同様にして追加します (リモート・メソッドがある場合)。

6) 「Roles (役割)」選択領域で、**「追加」**をクリックして、 WCSecurityRole を選択し、**「OK」**をクリックします。

7) **「適用」**をクリックします。

14. **「ファイル」**メニューから、**「Save (保管)」**を選択します。
15. アプリケーション・アセンブリー・ツールをクローズします。
16. 400 新しく変更された WC_Enterprise_App_*instanceName*.ear ファイルをローカル・マシンから WebSphere Application Server を実行する iSeries マシンにコピーします。つまり、以下のディレクトリーからファイルをコピーします。

    drive:¥WebSphere¥CommerceServer¥working

    コピー先は以下のディレクトリーです。

    iSeries_drive:¥QIBM¥UserData¥WebCommerce¥instances¥*instanceName*¥working

    ここで、*iSeries_drive* は、iSeries IFS にマップしたドライブ名です。

このステップが完了したら、以前のすべてのビジネス・ロジックと新しいビジネス・ロジックを含む新しいエンタープライズ・アプリケーションが作成されました。これは、新しく変更した WC_Enterprise_App_*instanceName*.ear ファイルにすべて含まれます。

変更されたエンタープライズ Bean のエンタープライズ・アプリケーションへのアセンブル

このセクションでは、変更された WebSphere Commerce エンタープライズ Bean をアプリケーション・アセンブリー・ツールでエンタープライズ・アプリケーションにアセンブルする方法を説明します。

このステップでは、エンタープライズ・アプリケーションをアプリケーション・アセンブラー・ツールでオープンします。アプリケーションをツール内でオープンした後、以下の操作を行って、変更された WebSphere Commerce エンタープライズ Bean をエンタープライズ・アプリケーションに組み込むことができます。

1. 変更前のバージョンの EJB グループのクラスパスをコピーする。
2. 変更前のバージョンの EJB グループを除去する。
3. 変更された新しいバージョンの EJB グループをインポートする。新しい EJB グループの JAR ファイルは、エンタープライズ・アプリケーションの EJB Module セクション内に保管されます。
4. 変更された EJB グループのクラスパスを設定する。
5. 変更されたエンティティー Bean に含まれるメソッド用の WebSphere Application Server セキュリティーをセットアップします。

AIX Solaris Linux `assembly.sh` ファイルのメモリーのヒープ・サイズを大きくし、新規または変更後の `.ear` ファイルを保管する際にメモリーが不足することのないようにしてください。

このファイルは以下のディレクトリーにあります。

- AIX `/usr/WebSphere/AppServer/bin`
- Solaris `/opt/WebSphere/AppServer/bin`
- Linux `/opt/WebSphere/AppServer/bin`

メモリーのヒープ・サイズを大きくするには、次の行を変更してください。

```
$JAVA_HOME/jre/bin/java
```

次のように表示されます。

```
$JAVA_HOME/jre/bin/java -mx512M
```

変更された EJB グループをエンタープライズ・アプリケーションにアセンブルするには、以下のようにします。

1. Windows AIX Solaris Linux ターゲットの WebSphere Commerce Server 上で、以下のようにして現在のエンタープライズ・アプリケーションのバックアップを取ります。
   a. コマンド・プロンプトで、以下のディレクトリーに移動します。
      - Windows *drive*:¥WebSphere¥CommerceServer¥working
      - AIX /usr/WebSphere/CommerceServer/working
      - Solaris /opt/WebSphere/CommerceServer/working
      - Linux /opt/WebSphere/CommerceServer/working
   b. 既存の WC_Enterprise_App_*instanceName*.ear ファイルのコピーを作成し、それに WC_Enterprise_App_*instanceName*.ear.bak という名前を付けます。
2. 400 以下のようにして、現在のエンタープライズ・アプリケーションのバックアップを取ります。
   a. コマンド・プロンプトで、以下を入力します。

      ```
      STRQSH
      cd /QIBM/UserData/WebCommerce/instances/instanceName/working
      cp WC_Enterprise_App_instanceName.ear
      WC_Enterprise_App_instanceName.ear.bak
      ```

   b. ローカル・クライアント・マシンと WebSphere Application Server を実行する iSeries マシンの間のデータ転送による待機時間が不必要に長引くのを避けるには、ローカル・クライアント・マシン上で以下のディレクトリーを作成してから、 WC_Enterprise_App_*instanceName*.ear ファイルをこのディレクトリーにコピーします。

      drive:¥WebSphere¥CommerceServer¥working
      ここで、*drive* はローカル・ドライブです。

      注: ローカル・マシン上に *drive*:¥WebSphere¥CommerceServer¥working ディレクトリーがなければ、ここで作成してください。
3. WebSphere Application Server 管理コンソールをオープンします。
4. **「ツール」**メニューから、**「Application Assembly Tool」**を選択します。
5. 以下のようにして、作業対象のエンタープライズ・アプリケーションをオープンします。
   a. **「ファイル」**メニューから、**「オープン」**を選択します。
   b. **「File name (ファイル名)」**フィールドで、以下を入力します。
      - Windows *drive*:¥WebSphere¥CommerceServer¥working¥
        WC_Enterprise_App_*instanceName*.ear

- AIX /usr/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- Solaris /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- Linux /opt/WebSphere/CommerceServer/working/WC_Enterprise_App_*instanceName*.ear
- 400 *drive*:¥WebSphere¥CommerceServer¥working¥WC_Enterprise_App_*instanceName*.ear

それから**「オープン」**をクリックします。アプリケーションがオープンするまで待って、次のステップに進みます。これには数分かかります。

6. **「EJB Modules (EJB モジュール)」**をクリックします。右側の画面に、エンタープライズ・アプリケーション内の EJB モジュールが表示されます。
7. 変更された EJB グループの EJB モジュールをクリックします。たとえば、WCSUser EJB グループ内の bean を変更した場合は、 **WCSUser** をクリックします。
8. 「General (一般)」タブをクリックして、クラスパス情報を表示します。この既存のクラスパス情報を、テキスト・ファイル (たとえば WCSUser_path.txt) にコピーします。
9. EJB モジュールを右クリックして、**「削除」**を選択します。
10. **「EJB Modules (EJB モジュール)」**を右クリックして、**「Import (インポート)」**を選択します。
11. **「File name (ファイル名)」**フィールドで、以下を入力します。
    - Windows *drive*:¥WebSphere¥CommerceServer¥temp¥ Cust_*EJBGroupName*-ejb.jar
    - AIX /usr/WebSphere/CommerceServer/temp/Cust_*EJBGroupName*-ejb.jar
    - Solaris /opt/WebSphere/CommerceServer/temp/Cust_*EJBGroupName*-ejb.jar
    - Linux /opt/WebSphere/CommerceServer/temp/Cust_*EJBGroupName*-ejb.jar
    - 400 *iSeries_drive*:¥QIBM¥UserData¥WebCommerce¥instances¥*instanceName*¥temp¥Cust_*EJBGroupName*-ejb.jar

    それから**「オープン」**をクリックします。「Confirm Values (値の確認)」ウィンドウで、**「OK」**をクリックします。
12. *EJBGroupJARFile*.jar ファイルがインポートされたら、変更した EJB グループまでスクロールして、このグループを選択します。
    このグループに関する情報が右側の画面に表示されます。
13. 変更前のバージョンの EJB グループのクラスパス情報が含まれているテキスト・ファイルをオープンします。クラスパスを選択してコピーします。

14. 変更された EJB グループの**「Classpath (クラスパス)」**フィールドに、このクラスパス情報を貼り付けます。
15. **「適用」**をクリックします。
16. **「ファイル」**メニューから、**「クローズ」**を選択します。
17. ファイルがクローズするのを待ち、**「ファイル」**メニューから**「オープン」**を選択して、 `WC_Enterprise_App_instanceName.ear` ファイルを再オープンします。
18. 以下のようにして、変更された bean のセキュリティーを構成します。
    a. 「EJB Modules (EJB モジュール)」ノードを拡張表示し、変更された EJB グループのノードを見付けてこれを拡張表示します。
    b. **「Entity Beans (エンティティー Bean)」**を拡張表示します。
    c. 変更された EJB グループを拡張表示します。
    d. **「Method Extensions (メソッドの拡張機能)」**をクリックし、右側の画面で以下のようにします。
        1) **「Advanced (拡張)」**タブをクリックします。
        2) **「Security identity (セキュリティー ID)」**が選択されていることを確認します。
        3) 各メソッドごとに、**「Use identity of EJB server (EJB サーバーの ID を使用する)」**が選択されていることを確認します。
        4) **「適用」**をクリックします (変更を加えた場合)。
    e. 左側のナビゲーション画面で、変更された EJB グループの**「Security Roles (セキュリティー役割)」**を右クリックし、**「New (新規)」**を選択して、以下のようにします。
        1) **「名前」**フィールドに `WCSecurityRole` と入力して、**「適用」**をクリックします。この役割がすでに存在している場合、このステップを実行する必要はありません。
    f. 左側のナビゲーション画面で、変更された EJB グループの**「Method Permissions (メソッド許可)」**を右クリックし、**「New (新規)」**を選択して、以下のようにします。
        1) **「Method Permission Name (メソッド許可名)」**フィールドで、`WCMethodPermission` と入力します。
        2) **「Methods (メソッド)」**選択領域で**「追加」**をクリックします。「Add Methods (メソッドの追加)」ウィンドウがオープンします。
        3) *modifiedEJBGroup* を拡張表示して、すべてのエンタープライズ Bean を選択します (選択時には Shift キーを押したままにします)。**「OK」**をクリックします。「Enterprise bean (エンタープライズ Bean)」列の下にすべてのエンタープライズ Bean が表示され、「Types (タイプ)」列の下にすべてのメソッドが示されます。

4) 「Roles (役割)」選択領域で、**「追加」**をクリックして、 WCSecurityRole を選択し、**「OK」**をクリックします。

5) **「適用」**をクリックします。

19. **「ファイル」**メニューから、**「Save (保管)」**を選択します。
20. アプリケーション・アセンブリー・ツールをクローズします。
21. 400 新しく変更された WC_Enterprise_App_*instanceName*.ear ファイルをローカル・マシンから WebSphere Application Server を実行する iSeries マシンにコピーします。つまり、以下のディレクトリーからファイルをコピーします。

    drive:¥WebSphere¥CommerceServer¥working

    コピー先は以下のディレクトリーです。

    iSeries_drive:¥QIBM¥UserData¥WebCommerce¥instances¥*instanceName*¥working

    ここで、*iSeries_drive* は、iSeries IFS にマップしたドライブ名です。

このステップが完了したら、以前のすべてのビジネス・ロジックと新しいビジネス・ロジックを含む新しいエンタープライズ・アプリケーションが作成されました。これは、新しく変更した WC_Enterprise_App_*instanceName*.ear ファイルにすべて含まれます。

エンタープライズ・アプリケーションの停止と除去

このセクションでは、WebSphere Application Server 管理コンソールを使用して、現在稼働しているエンタープライズ・アプリケーションを停止して除去する方法を説明します。

エンタープライズ・アプリケーションを停止して除去するには、以下のようにします。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere 管理可能ドメイン」**を拡張表示します。
3. **「ノード」**を拡張表示します。
4. ***「nodeName」*** を拡張表示します (***nodeName*** は、ご使用のノードの名前です)。
5. **「Application Servers (アプリケーション・サーバー)」**を拡張表示します。
6. Windows AIX Solaris Linux WebSphere Commerce アプリケーション・サーバーを右クリックします。たとえば、**「WebSphere Commerce Server - *nodeName*」**を右クリックして、**「停止」**を選択します。
7. 400 WebSphere Commerce アプリケーション・サーバーを右クリックします。たとえば、「***instanceName* - WebSphere Commerce Server**」を右クリックして、**「停止」**を選択します。
8. **「Enterprise Applications (エンタープライズ・アプリケーション)」**を拡張表示します。

9. Windows AIX Solaris Linux WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Enterprise Application - *instanceName*」**アプリケーションを右クリックして、**「停止」**を選択します。
10. 400 WebSphere Commerce アプリケーションを右クリックします。たとえば、**「*instanceName* - WebSphere Commerce Enterprise Application」**アプリケーションを右クリックして、**「停止」**を選択します。
11. WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Enterprise Application -*instanceName*」** (iSeries の場合は**「*instanceName* - WebSphere Commerce Enterprise Application」**) アプリケーションを右クリックして、**「除去」**を選択します。
12. アプリケーションをエクスポートするかどうかの指示を求められたら、**「いいえ」**を選択します。
13. アプリケーションを除去するかどうかの指示を求められたら、**「はい」**を選択します。

エンタープライズ・アプリケーションのインポート

このセクションでは、XMLConfig コマンド行ユーティリティーを使用してエンタープライズ・アプリケーションをインポートする方法を説明します。

新しいエンタープライズ・アプリケーションをインポートするには、以下のようにします。

1. Windows AIX Solaris Linux XMLConfig ツールを起動し、以下のコマンドを入力して、エンタープライズ・アプリケーションを WebSphere Application Server にインポートします。

   Windows

   ```
   xmlConfig -import OutputFile.xml -adminNodeName wasHostName
   ```

   AIX

   ```
   /usr/WebSphere/AppServer/bin/XMLConfig.sh -import OutputFile.xml
      -adminNodeName wasHostName
    -nameServiceHost wasHostName -nameServicePort wasAdminPort
   ```

   Solaris Linux

   ```
   /opt/WebSphere/AppServer/bin/XMLConfig.sh -import OutputFile.xml
      -adminNodeName wasHostName
    -nameServiceHost wasHostName -nameServicePort wasAdminPort
   ```

   ここで

- *wasHostName* は、 WebSphere Application Server において現在のエンタープライズ・アプリケーションが入っているノードの名前です。
- *OutputFile.xml* は、すべてのエンタープライズ Bean について記述する XML ファイルです。
- *wasAdminPort* は、WebSphere Application Server 管理ポートです。

2. 400 XMLConfig ツールを起動し、以下のコマンドを入力して、エンタープライズ・アプリケーションを WebSphere Application Server にインポートします。

```
STRQSH
PATH=/QIBM/ProdData/WebASAdv4/bin:$PATH
xmlConfig -import
/QIBM/UserData/WebCommerce/instances/instanceName/working/OutputFile.xml
-adminNodeName wasHostName
 -nameServiceHost wasHostName -nameServicePort wasAdminPort
-instance wasInstanceName
```

ここで

- *OutputFile.xml* は、すべてのエンタープライズ Bean について記述する XML ファイルの完全修飾名です。
- *wasHostName* は、 WebSphere Application Server において現在のエンタープライズ・アプリケーションが入っているノードの名前です。

  注: 400 *wasHostName* の値には大文字小文字の区別があり、 TCP/IP 構成での値と一致していなければなりません。 (コマンド行プロセッサーを使用して、CFGTCP オプション 12 にアクセスし、ホスト名を検査してください。)
- *wasAdminPort* は、WebSphere Application Server 管理ポートです。
- *wasInstanceName* は、WebSphere Application Server インスタンス名です。

400 XMLConfig -import コマンドを実行しようとしている間に、次のエラー・メッセージを受け取ることがあります。“Cannot expand the ear file under /QIBM/UserData/WebAsAdv4/*wasInstanceName*/installedApps/WC_Enterprise_App_*instanceName*.ear”. このメッセージを受け取る場合、前述のディレクトリーを除去するかまたは名前変更して、コマンドを再実行してください。

3. 400 エンタープライズ・アプリケーションをインポートし終えたら、ディレクトリー許可を変更するスクリプトを実行する必要があります。このスクリプトを実行するには、以下のようにします。
   - コマンド・プロンプトで、以下を入力します。

```
STRSQH
cd /QIBM/ProdData/WebCommerce/bin
changeAuthority wasAdminInstanceName instanceName
```

ここで

– *wasAdminInstanceName* は WebSphere Application Server 管理インスタンスの名前です。

– ここで、*instanceName* は WebSphere Commerce インスタンスの名前です。

エンタープライズ・アプリケーションの開始

このセクションでは、WebSphere Application Server 管理コンソールを使用してビューを最新表示し、エンタープライズ・アプリケーションを開始する方法を説明します。

1. WebSphere Application Server 管理コンソールをオープンします。
2. **「WebSphere Administrative Domain (WebSphere 管理可能ドメイン)」**、**「Nodes (ノード)」**、「*nodeName*」 の順に拡張表示します。
3. 「*nodeName*」ノードを強調表示します。
4. **「Refresh selected subtree (選択したサブツリーの最新表示)」**アイコンをクリックします。
5. 以下のようにして、WebSphere Commerce アプリケーションを開始します。
   - **「Application Servers (アプリケーション・サーバー)」**を拡張表示します。
   - Windows AIX Solaris Linux WebSphere Commerce アプリケーションを右クリックします。たとえば、**「WebSphere Commerce Server - *instanceName*」**を右クリックして、**「開始」**を選択します。
   - 400 WebSphere Commerce アプリケーションを右クリックします。たとえば、「*instanceName* **- WebSphere Commerce Server**」アプリケーションを右クリックして、**「開始」**を選択します。

付録 C. VisualAge for Java のヒント

このセクションでは、開発環境内でのトラブルシューティング、パフォーマンス向上、および単純化に関連したいくつかのヒントを説明します。

WebSphere Test Environment でのサーブレット・エンジンのプロパティーの変更

WebSphere Test Environment でのサーブレット・エンジンのプロパティーは default.servlet_engine プロパティー・ファイルによって制御されます。このファイルの中で、Web サーバー用の文書ルートに変更を加えたり、 WebSphere Test Environment によって使用されるポートを変更したりすることができます。

WebSphere Test Environment が機能を終えていて、リブートという手段を選択できない状況では、ポートを変更することが考えられます。ポートを変更するには、以下のようにします。

1. *VAJ_install_path*¥IDE¥ProjectResources¥IBM WebSphere Test Environment¥properties¥default.servlet_engine ファイルをテキスト・エディターでオープンします。
2. <transport> スタンザで、以下の行にある 8080 という値を、使用可能なポートに変更します。

   ```
   <arg name="port" value="8080"/>
   ```

3. 以下の行にある 8080 という値を、上記で指定したのと同じポートに変更します。

   ```
   <hostname-binding hostname="localhost:8080" servlethost="default_host"/>
      <hostname-binding hostname="127.0.0.1:8080" servlethost="default_host"/>
   ```

4. ファイルを保管します。
5. WebSphere Test Environment Control Center をオープンしてサーブレット・エンジンを開始します。

デフォルトでは、WebSphere Test Environment の文書ルートは *VAJ_install_path*¥IDE¥ProjectResources¥IBM WebSphere Test Environment¥hosts¥default_host¥default_app¥web です。これを別のディレクトリーに変更するには、以下のようにします。

1. WebSphere Test Environment Control Center をオープンしてサーブレット・エンジンを停止します。
2. *VAJ_install_path*¥IDE¥ProjectResources¥IBM WebSphere Test Environment¥properties¥default.servlet_engine ファイルをテキスト・エディターでオープンします。

3. `<websphere-webgroup name="default_app">` スタンザで、
   `<document-root>$approot$/web</document-root>` を以下のように変更します。

   ```
   <document-root>your_document_root</document-root>
   ```

   ここで、*your_document_root* は適切な文書ルートです。
4. 以下の行にある 8080 という値を、上記で指定したのと同じポートに変更します。

   ```
   <hostname-binding hostname="localhost:8080" servlethost="default_host"/>
      <hostname-binding hostname="127.0.0.1:8080" servlethost="default_host"/>
   ```
5. ファイルを保管します。
6. WebSphere Test Environment Control Center をオープンしてサーブレット・エンジンを開始します。

永続ネーム・サーバーの問題の解決

永続ネーム・サーバーで問題に直面した場合、永続ネーム・サーバー・データベースを除去してから再作成する必要があるかもしれません。このデータベースの作成の詳細については、 *Commerce Studio インストール・ガイド* を参照してください。

また、ポートが現在使用中であるというメッセージが出された場合は、 WebSphere Application Server が実行中でないことを確認してください。

コンパイルされた JSP ファイルの削除

VisualAge for Java が保守するプロジェクト・フォルダーには、パフォーマンス上の理由で、コンパイルされた JSP ファイルが保管されています。コンパイルされた JSP ファイルを削除することが必要な場合もあるでしょう。たとえば、JSP ファイルからデータ Bean を除去した場合、次回に JSP ファイルを呼び出すときにエラーになることがあります。そのような場合、コンパイルされた JSP ファイルを削除することができます。

コンパイルされた JSP ファイルを削除するには、以下のようにします。

1. VisualAge for Java で、「プロジェクト」タブを選択します。
2. **「JSP ページ・コンパイル生成コード」**プロジェクトまでスクロールダウンします。
3. (多くのコンパイルされた JSP ファイルを削除する必要がある場合) プロジェクト全体を選択するか、あるいはプロジェクトを拡張表示して、再コンパイルの必要なものだけを削除してください。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。 日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品、プログラムまたはサービスの操作性の評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む。) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。 実施権の許諾については、下記の宛先に書面にてご照会ください。

〒106-0032 東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。 国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書は定期的に見直され、必要な変更 (たとえば、技術的に不適切な表現や誤植など) は、本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。 それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Ltd.
Office of the Lab Director
8200 Warden Avenue,
Markham, Ontario
L6G 1C7
Canada

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。 お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

表示されている IBM の価格は IBM が小売価格として提示しているもので、現行価格であり、通知なしに変更されるものです。卸価格は、異なる場合があります。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。 お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。 したがって IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

以下は、IBM Corporation の商標です。

400	iSeries
AIX	MQSeries
AS/400	Net.Commerce
CICS	Net.Data
DB2	VisualAge
IBM	WebSphere

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

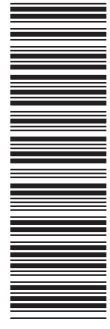
[カ行]

[サ行]

[タ行]

部品番号: CT024JA

Printed in Japan



(1P) P/N: CT024JA

GC88-9273-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

IBM® WebSphere® Commerce

プログラマーズ・ガイド

バージョン 5.4