

IBM WebSphere Commerce



プログラミング・ガイドとチュートリアル

バージョン 5.5

IBM WebSphere Commerce



プログラミング・ガイドとチュートリアル

バージョン 5.5

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、IBM WebSphere Commerce Business Edition バージョン 5.5、IBM WebSphere Commerce Professional Edition バージョン 5.5、および新しい版で明記されていない限り、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。

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)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原 典： IBM WebSphere Commerce
Programming Guide and Tutorials
Version 5.5

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 2003.7

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

© Copyright International Business Machines Corporation 2000, 2003. All rights reserved.

© Copyright IBM Japan 2003

はじめに

本書は 2003 年 5 月 30 日に更新されました。

「*WebSphere Commerce* プログラミング・ガイドとチュートリアル」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
- 既存のコントローラー・コマンドの変更
- オブジェクト・モデルの拡張および既存のタスク・コマンドの変更
- 既存の WebSphere Commerce Entity Bean の拡張

本書の変更点

本書のコピーおよび本書の更新版は、WebSphere Commerce Web サイトの Technical Library セクションから、PDF ファイルとして利用できます。

<http://www.ibm.com/software/commerce/library/>

その他のサポート情報は、WebSphere Commerce Support サイトを参照してください。

<http://www.ibm.com/software/commerce/support/>

本書の更新版は、次の Web サイトにある WebSphere Developer ドメインの WebSphere Commerce Zone から利用できます。

<http://www.ibm.com/websphere/developer/zones/commerce>

本書の規則

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® 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。

DB2 は、DB2 ユニバーサル・データベース (DB2 Universal Database™) 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。

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™
- EnterpriseJavaBeans 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
- WebSphere Studio Application Developer

パス変数

本書では、ディレクトリー・パスを表すために以下の変数を使用しています。

WC_installdir

これは、WebSphere Commerce のインストール・ディレクトリーです。以下に示すのは、さまざまなオペレーティング・システムでの WebSphere Commerce のデフォルト・インストール・ディレクトリーです。

-  Windows C:\Program Files\WebSphere\CommerceServer55
-  AIX /usr/WebSphere/CommerceServer55
-  Solaris /opt/WebSphere/CommerceServer55
-  Linux /opt/WebSphere/CommerceServer55
-  400 /QIBM/ProdData/CommerceServer55

WC_userdir

WebSphere Commerce によって使用されるすべてのデータ用のディレクトリーで、ユーザーが変更でき、ユーザーが構成する必要があるものです。

-  400 /QIBM/UserData/CommerceServer55


WAS_installdir

これは、WebSphere Application Server のインストール・ディレクトリーです。以下に示すのは、さまざまなオペレーティング・システムでの WebSphere Application Server のデフォルト・インストール・ディレクトリーです。

-  Windows C:\Program Files\WebSphere\AppServer
-  AIX /usr/WebSphere/AppServer
-  Solaris /opt/WebSphere/AppServer
-  Linux /opt/WebSphere/AppServer
-  400 /QIBM/ProdData/WebAs5/Base

WAS_userdir

WebSphere Application Server によって使用されるすべてのデータ用のディレクトリーで、ユーザーが変更でき、ユーザーが構成する必要があるものです。

-  400 QIBM/UserData/WebAS5/Base/WAS_instancename
および *WAS_instance_name* は、 WebSphere Commerce インスタンスが関連付けられる WebSphere Application Server の名前を表します。

WCStudio_installdir

WebSphere Commerce Studio のインストール・ディレクトリーです。次に示すのは、デフォルト・インストール・ディレクトリーです。

C:¥WebSphere¥CommerceStudio55

追加情報の入手先

WebSphere Commerce について詳しくは、以下の Web サイトを参照してください。

<http://www.ibm.com/software/commerce/library/>

目次

はじめに	iii
本書の変更点	iii
本書の規則	iv
知識に関する条件	iv
パス変数	v
追加情報の入手先	vi

第 1 部 概念とアーキテクチャー . . . 1

第 1 章 概要 3

WebSphere Commerce ソフトウェア・コンポー ネント	3
WebSphere Commerce アプリケーション・アー キテクチャー	4
WebSphere Commerce ランタイム・アーキテク チャー	7
サーブレット・エンジン	9
プロトコル・リスナー	9
アダプター・マネージャー	9
アダプター	10
Web コントローラー	11
コマンド	12
WebSphere Commerce Entity Bean	13
Data Bean	14
Data Bean マネージャー	14
JavaServer Pages テンプレート	14
instance_name.xml 構成ファイル	15
要求の要約	15

第 2 部 プログラミング・モデル 19

第 2 章 デザイン・パターン 21

モデル・ビュー・コントローラー・デザイ ン・パターン	21
コマンド・デザイン・パターン	23
コマンド・フレームワーク	23
コマンド・ファクトリー	26
コマンドのフロー	28
コマンド登録フレームワーク	30
表示デザイン・パターン	41
JSP テンプレートと Data Bean	41

Data Bean のタイプ	42
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	46
遅延フェッチ・データ検索	46
JSP 属性の設定 - 概要	47
必要なプロパティー設定	49

第 3 章 永続オブジェクト・モデル 51

WebSphere Commerce Entity Bean のインプリ メンテーション	51
WebSphere Commerce Entity Bean - 概要	51
WebSphere Commerce Enterprise Bean のデ プロイメント記述子	53
WebSphere Commerce オブジェクト・モデ ルの拡張	54
オブジェクトのライフ・サイクル	84
トランザクション	84
Entity Bean に関するその他の考慮事項	85
Entity Bean の使用	89
データベースに関する考慮事項	90
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	90
データベース列のデータ・タイプに関する 考慮事項	92
さまざまなデータベース間でのデータ・タ イプの違い	93

第 4 章 アクセス制御 97

アクセス制御の理解	97
WebSphere Application Server でのリソース 保護の概要	97
URL パラメーターに関するセキュリテ ー上の考慮事項	100
WebSphere Commerce アクセス制御ポリシ ーの概要	101
アクセス制御のタイプ	108
アクセス制御の相互作用	110
保護可能なインターフェース	113
Groupable インターフェース	114
アクセス制御についての情報の入手先	114
アクセス制御のインプリメント	114

保護可能なリソースの識別	114	新規コントローラー・コマンド	149
Enterprise Bean でのアクセス制御のインプリメント	115	isGeneric メソッド	149
Data Bean でのアクセス制御のインプリメント	118	isRetriable メソッド	150
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	119	setRequestProperties メソッド	150
ビューでのアクセス制御ポリシーのインプリメント	122	validateParameters メソッド	151
既存の WebSphere Commerce リソースでのアクセス制御の変更	123	getResources メソッド	151
既存の WebSphere Commerce Entity Bean への新規の関係の追加	123	performExecute メソッド	151
まだ保護されていない既存の WebSphere Commerce Entity Bean へのアクセス制御の追加	125	長時間実行されるコントローラー・コマンド	152
コントローラー・コマンド実行時のアクセス制御の影響の知識	126	ビュー・コマンドに対する入力プロパティのフォーマット設定	153
開発用のサンプル・アクセス制御ポリシー	128	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	154
新規ビューのサンプル・アクセス制御ポリシー	128	長さが制限されたりダイレクト URL の処理	154
新規コントローラー・コマンド用のサンプル・コマンド・レベル・アクセス制御ポリシー	129	HttpForwardView についての HttpServletRequest オブジェクトでの属性の設定	155
新規コマンドおよび Enterprise Bean のサンプル・リソース・レベル・アクセス制御ポリシー	130	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	156
第 5 章 エラー処理とメッセージ	133	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	157
コマンド・エラー処理	133	新規タスク・コマンド	159
例外のタイプ	133	既存のコマンドのカスタマイズ	160
エラー・メッセージ・プロパティ・ファイル	134	既存のコントローラー・コマンドのカスタマイズ	160
例外処理のフロー	134	既存のタスク・コマンドのカスタマイズ	164
カスタマイズされたコードにおける例外処理	136	Data Bean のカスタマイズ	166
メッセージの作成	138	第 7 章 取引の合意事項とビジネス・ポリシー (Business Edition)	169
実行フローのトレース	140	概要	169
JSP テンプレートのエラー処理	141	ビジネス・ポリシー・オブジェクトおよびコマンド	170
第 6 章 コマンドのインプリメンテーション 143		ToolTech のサンプル契約データ	172
新規コマンド - 概要	143	CONTRACT テーブルのサンプル・データ	172
カスタマイズ・コードのパッケージ	146	TERMCOND テーブルのサンプル・データ	173
コマンド・コンテキスト	147	POLICYTC テーブルのサンプル・データ	173
URL コマンドのコンテキスト情報に対する一時的な変更	148	POLICY テーブルのサンプル・データ	174
		TRADEPOSCN テーブルのサンプル・データ	174
		SHIPMODE テーブルのサンプル・データ	174
		既存の契約モデルの拡張	175
		新しいビジネス・ポリシーの作成	175
		新規のビジネス・ポリシー・タイプの作成	176

新規のビジネス・ポリシー・コマンドの作成	177
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	180
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	181
新規の使用条件の作成	181
新規のビジネス・ポリシーの呼び出し	197
契約の作成	198
契約のカスタマイズのシナリオ	199
リバートのシナリオ	199

第 3 部 開発環境 207

第 8 章 開発環境 209

代表的な開発環境	209
WebSphere Studio Application Developer	210
iSeries での開発環境	210
実稼働環境で Oracle データベースを使用するときの開発用ローカル DB2 データベースの使用	211
WebSphere Commerce Enterprise Bean 変換ツールの概要	211
開発環境内の支払いオプション	211

第 9 章 デプロイメントに関する詳細情報 213

デプロイメント・ステップのユーザー許可要件	213
増分デプロイメント	214
Enterprise Bean のデプロイメント	214
EJB JAR ファイルの作成	214
ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新	218
コマンドと Data Bean のデプロイメント	220
JAR ファイルの作成	221
ターゲット WebSphere Commerce Server 上の JAR ファイルの更新	221
ストア資産のデプロイメント	222
ストア資産のエクスポート	223
ストア資産の転送	223
ターゲット・データベースの更新	224
アクセス制御の更新	225

第 4 部 チュートリアル 227

第 10 章 チュートリアル: ビジネス・ロジックの新規作成

サンプル・コードの場所	230
ワークスペースの準備	230
新規ビューの作成	234
MyNewView の登録	234
チュートリアル用のプロパティ・ファイルの作成	236
MyNewJSPTemplate の作成	237
MyNewView のアクセス制御ポリシーの作成とロード	240
MyNewView のテスト	241
新規コントローラー・コマンドの作成	242
MyNewControllerCmd の登録	243
MyNewControllerCmd インターフェースの作成	244
MyNewControllerCmdImpl インプリメンテーション・クラスの作成	245
コマンドのアクセス制御ポリシーの作成とロード	246
MyNewControllerCmd のテスト	247
MyNewControllerCmd から MyNewView への情報の引き渡し	248
TypedProperties オブジェクトを使用した情報の引き渡し	248
Data Bean を使用した情報の引き渡し	251
MyNewControllerCmd 中の URL パラメータの構文解析と妥当性検査	257
MyNewControllerCmd への新規フィールドの追加	257
URL パラメータのビューへの引き渡し	259
欠落パラメータと妥当性検査値のキャッチ	259
MyNewDataBean への新規フィールドの追加	260
URL パラメータを表示するための MyNewJSPTemplate の変更	261
URL パラメータ値のテスト	262
新規タスク・コマンドの作成	265
MyNewTaskCmd の作成	266
タスク・コマンドの呼び出し	269
グリーンディング・メッセージを追加するための MyNewJSPTemplate の変更	270
MyNewTaskCmd のテスト	271
MyNewTaskCmd の変更	272

タスク・コマンド用のオブジェクトを作成するための MyNewControllerCmdImpl の変更	273
ユーザー名妥当性検査のための新規タスク・コマンドの変更	274
ユーザー名妥当性検査のための MyNewJSPTemplate の変更	275
ユーザー名妥当性検査のテスト	276
新規 Entity Bean の作成	278
XBONUS テーブルの作成	278
BonusBean Entity Bean の作成	279
Bonus エンティティーと MyNewControllerCmd の統合	289
ボーナス・ポイント・ロジックのデプロイメント	304
コマンドと Data Bean の JAR ファイルの作成	304
EJB JAR ファイルの作成	305
ストア資産のエクスポート	306
アクセス制御ポリシーのパッケージ化	307
ターゲット WebSphere Commerce Server への資産の転送	307
ターゲット WebSphere Commerce Server の停止	308
ターゲット WebSphere Commerce Server 上のデータベースの更新	308
ターゲット WebSphere Commerce Server 上のストア資産の更新	314
ターゲット WebSphere Commerce Server 上のコマンドと Data Bean の JAR ファイルの更新	314
ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新	314
ターゲット WebSphere Commerce Server 上のボーナス・ポイント・ロジックの検証	316

第 11 章 チュートリアル: 既存のコントローラー・コマンドの変更	319
前提条件	319
新規の MyOrderItemAddCmdImpl クラスの作成	320
メッセージ情報の作成	322
コマンド・レジストリーに変更を加える	324
MyOrderItemAddCmdImpl コマンドのテスト	325
MyOrderItemAddCmdImpl のデプロイメント	328
コマンド JAR ファイルの作成	329

メッセージ・プロパティー・ファイルのエクスポート	330
ターゲット WebSphere Commerce Server への資産の転送	330
ターゲット WebSphere Commerce Server の停止	331
ターゲット WebSphere Commerce Server 上のデータベースの更新	331
ターゲット WebSphere Commerce Server 上のコマンド JAR ファイルの更新	332
ターゲット WebSphere Commerce Server 上のメッセージ・プロパティーの更新	333
ターゲット WebSphere Commerce Server 上の MyOrderItemAddCmdImpl ロジックの検証	333

第 12 章 チュートリアル: オブジェクト・モデルの拡張および既存のタスク・コマンドの変更	337
前提条件	338
XORDGIFT テーブルの作成と値の取り込み	338
OrderGift Entity Bean の作成	339
OrderGift Entity Bean のショッピング・フローへの統合	353
OrderGiftDataBean の作成	353
MyExtOrderProcessCmdImpl クラスの作成	354
変更のコンパイル	357
ギフト・メッセージの表示ページの変更	357
新規ギフト・メッセージ機能のテスト	360
新規ギフト・メッセージ機能のデプロイメント	363
コマンドと Data Bean の JAR ファイルの作成	363
EJB JAR ファイルの作成	364
ストア資産のエクスポート	365
ターゲット WebSphere Commerce Server への資産の転送	366
ターゲット WebSphere Commerce Server の停止	366
ターゲット WebSphere Commerce Server 上のデータベースの更新	367
ターゲット WebSphere Commerce Server 上のストア資産の更新	369
ターゲット WebSphere Commerce Server 上のコマンドと Data Bean の JAR ファイルの更新	369

ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新	370
ターゲット WebSphere Commerce Server でのギフト・メッセージ機能の検査.	371
第 13 章 チュートリアル: 既存の WebSphere Commerce Entity Bean の拡張	
張	375
前提条件.	375
XHOUSING テーブルの作成と値の取り込み	375
新規フィールドの User Entity Bean への追加	377
スキーマとテーブル・マッピング情報の更新	378
XHOUSING テーブルのテーブル定義の作成	378
XHOUSING テーブル・マップの作成	380
マッピング・ファイルの更新	380
Access Bean およびデプロイメント・コードの生成.	381
MyPostUserRegistrationAddCmdImpl インプリメンテーションの作成	382
コマンド・レジストリーに変更を加える	384
ハウジング情報を収集して表示するための JSP テンプレートの変更	386
変更を加えたコードのテスト	388
ハウジング調査書ロジックのデプロイ	390
コマンド JAR ファイルの作成	390
EJB JAR ファイルの作成	391
ストア資産のエクスポート.	392
ターゲット WebSphere Commerce Server への資産の転送	393
ターゲット WebSphere Commerce Server の停止	393
ターゲット WebSphere Commerce Server 上のデータベースの更新	393
ターゲット WebSphere Commerce Server 上のストア資産の更新	396

ターゲット WebSphere Commerce Server 上のコマンド JAR ファイルの更新	396
ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新	396
ターゲット WebSphere Commerce Server 上のハウジング調査書ロジックの検証	397

第 5 部 付録 401

付録 A. WebSphere Commerce Studio での WebSphere Commerce コンポーネン ト・トレースの構成	403
出力ファイル	403

付録 B. 追加情報の入手先	405
WebSphere Commerce Studio 情報	405
WebSphere Commerce Studio オンライン ヘルプ	405
WebSphere Commerce Web サイト	406
WebSphere Developer Domain	406
IBM レッドブック	406
WebSphere Studio Application Developer 情報	406
WebSphere Studio Application Developer オンライン・ヘルプ	406
WebSphere Studio Application Developer Web サイト.	406
WebSphere Developer Domain	407
IBM レッドブック	407

特記事項	409
商標	411

索引	413
---------------------	------------

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、WebSphere Commerce 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。

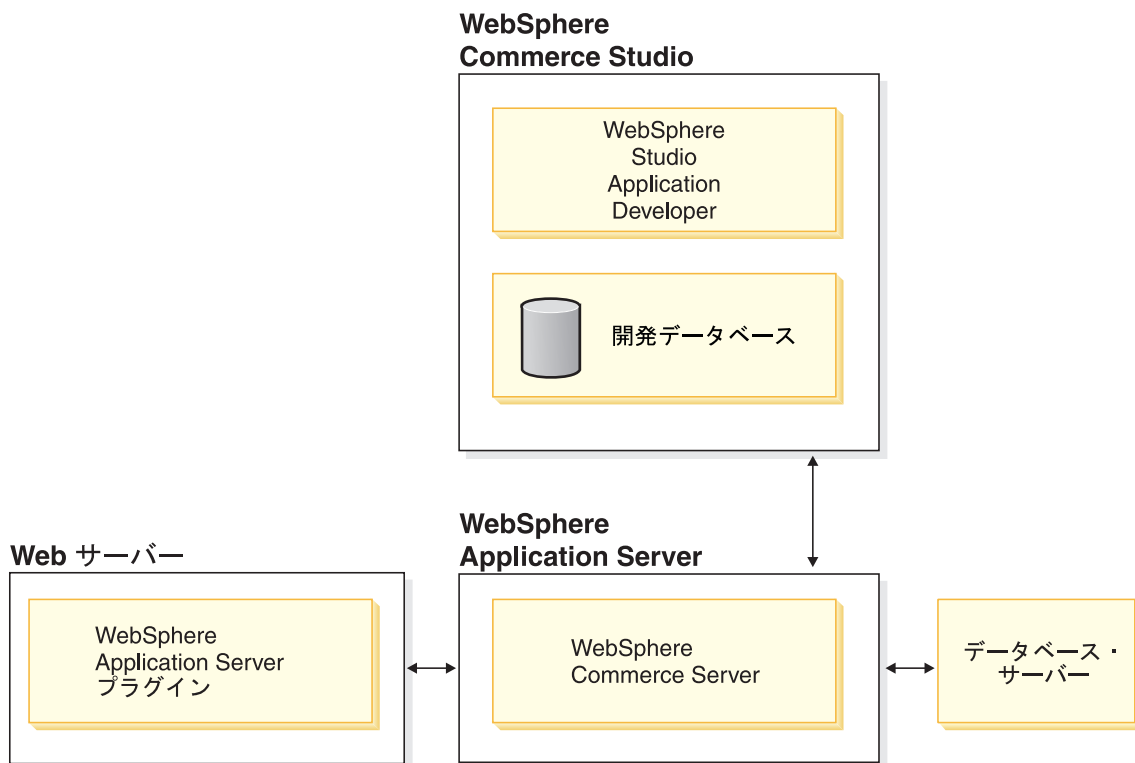


図 1.

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لم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 WebSphere Studio Application Developer を使用して以下のタスクを実行できます。

- JSP テンプレートや HTML ページなどのストアフロント資産の作成とカスタマイズ
- Java 形式の新規ビジネス・ロジックの作成
- Java 形式の既存ビジネス・ロジックの変更
- コードとストアフロント資産のテスト

WebSphere Commerce Studio は開発データベースを使用します。開発者は、ご希望のデータベース・ツール (WebSphere Studio Application Developer を含む) を使用して、データベースに変更を加えることができます。

WebSphere Commerce アプリケーション・アーキテクチャー

ここまでで、WebSphere Commerce 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。この説明により、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。

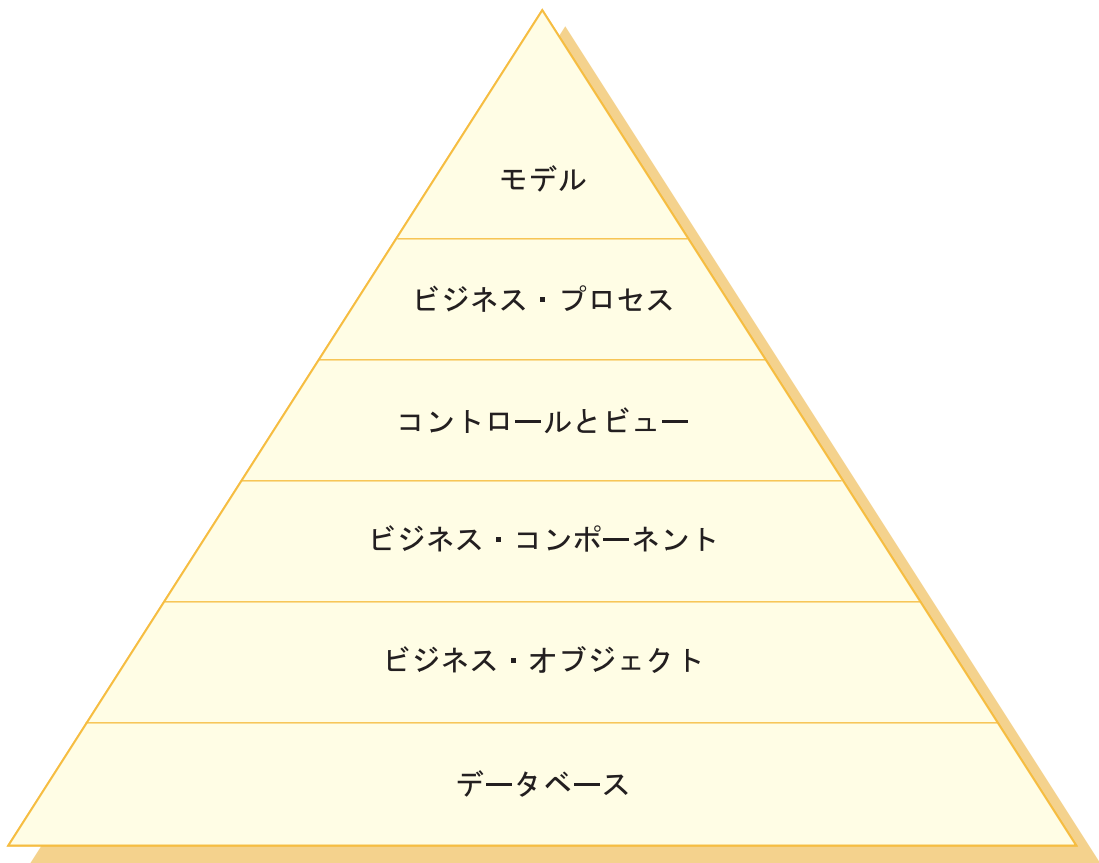


図2.

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。

- USERS
- ORDERS
- INVENTORY

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は Enterprise JavaBeans 仕様に準拠しています。

これらの Entity Bean はビジネス・コンポーネント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、Entity 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、ProductDisplayView (ショッピング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、 OrderCancelView 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。

E メール・キャンペーンの作成

このビジネス・プロセスには、ビジネス・コンポーネントと、E メール・キャンペーンの作成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。

オンライン・カタログの準備

このビジネス・プロセスには、ビジネス・コンポーネントと、オンライン・カタログの作成に関連したサブプロセスが含まれます。これには、カタログの設計、カタログ・データのロード、取引管理アソシエーションの作成、および価格情報の設定が含まれます。

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、FashionFlow サンプル・ストアで表示される消費者向けモデルがあります。別の例としては、ToolTech サンプル・ストアで表示される B2B 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
- Entity Bean
- Data Bean
- Data 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。

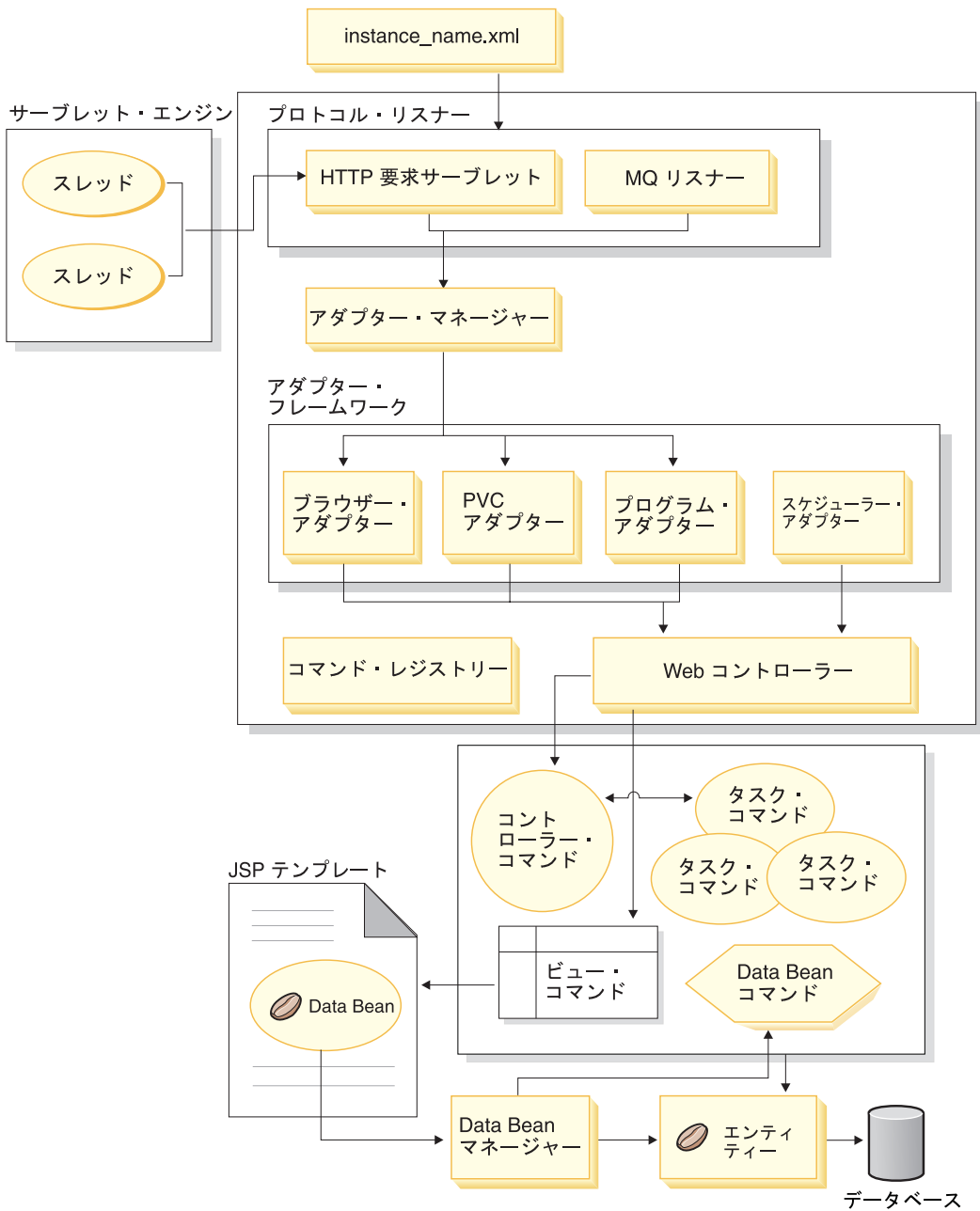


図 3.

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。

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
- インターネット・ブラウザを使用した携帯電話
- 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 (ここで *instance_name* は、WebSphere Commerce インスタンスの名前です)。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・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。

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。

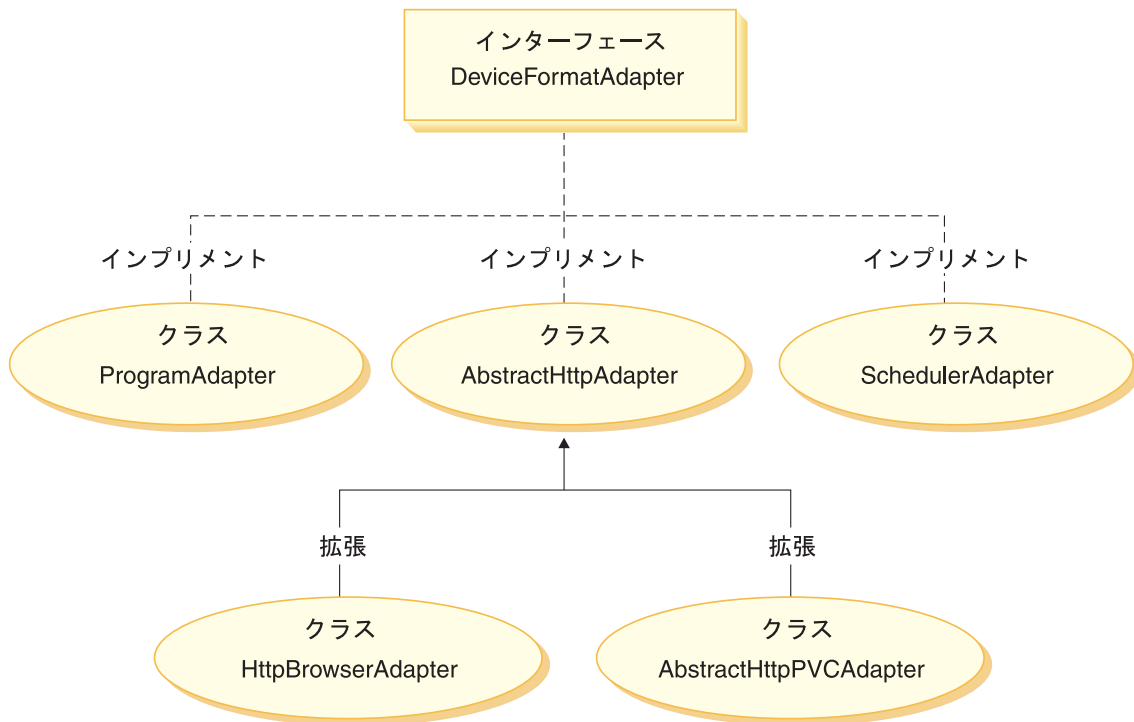


図 4.

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・マップ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・コンテナです。このコンテナは、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ない、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らない。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・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、オーダー処理用の *OrderProcessCmd* コマンドや、ユーザーがログオンできるようにする *LogonCmd* 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で実行されます。

Data Bean コマンド

Data Bean コマンドは、Data Bean がインスタンス化されるときに、Data Bean マネージャーによって呼び出されます。Data Bean コマンドの主な機能は、Data 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・キーが入力されると、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 Entity Bean

Entity Bean は、WebSphere Commerce が提供する、永続的なトランザクションのクラス・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、Entity 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 Entity Bean からデータにアクセスできます。既存の Entity 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、完全に新規 Entity Bean のデプロイメントを実行することもできます。

Entity 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。

Entity Bean の詳細については、51 ページの『WebSphere Commerce Entity Bean のインプリメンテーション』を参照してください。

Data Bean

Data 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。

Data Bean マネージャー

JSP テンプレートに挿入された WebSphere Commerce Data Bean により、ページに動的コンテンツを取り込めるようになります。Data Bean マネージャーは、次のコードの行がページに挿入されるときに Data Bean の値が取り込まれるように、その Data Bean をアクティブにします。

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

ここで、*data_bean* はアクティブにする Data Bean で、*request* は `HttpServletRequest` オブジェクトです。

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 Data Bean を単一トランザクション内でアクティブに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。

Data 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、97 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 (ここで、*instance_name* は WebSphere Commerce インスタンスの名前) は、WebSphere Commerce 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。

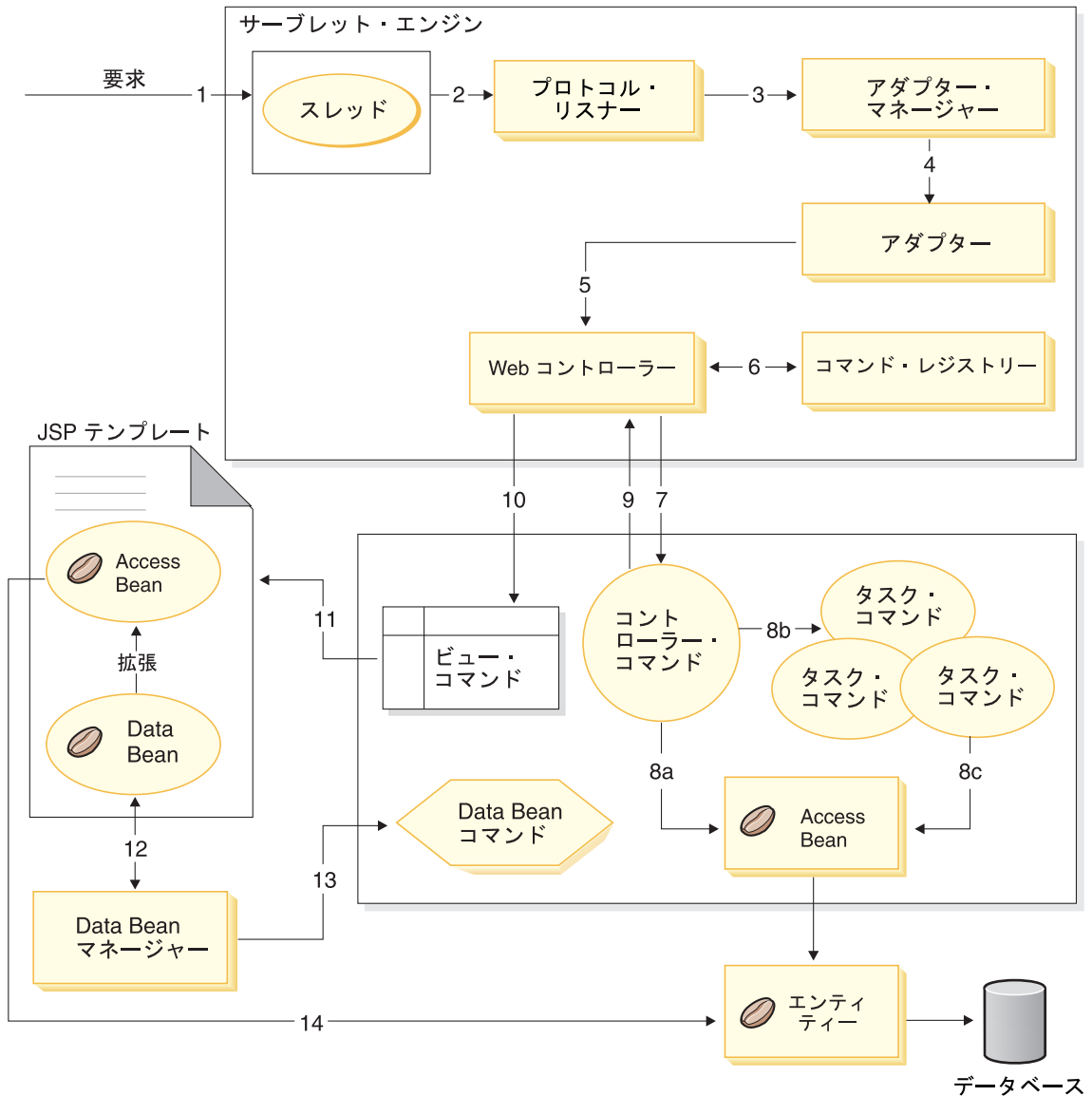


図 5.

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、Access Bean とそれに対応する Entity 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、Access Bean とそれに対応する Entity 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、リクエスト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、Data Bean が必要とされます。Data Bean マネージャーが Data Bean をアクティブにします。
13. 必要であれば、Data Bean マネージャーが Data Bean コマンドを呼び出します。
14. Data Bean の拡張元である Access Bean が、それに対応する Entity Bean を使用して、データベースにアクセスします。

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。

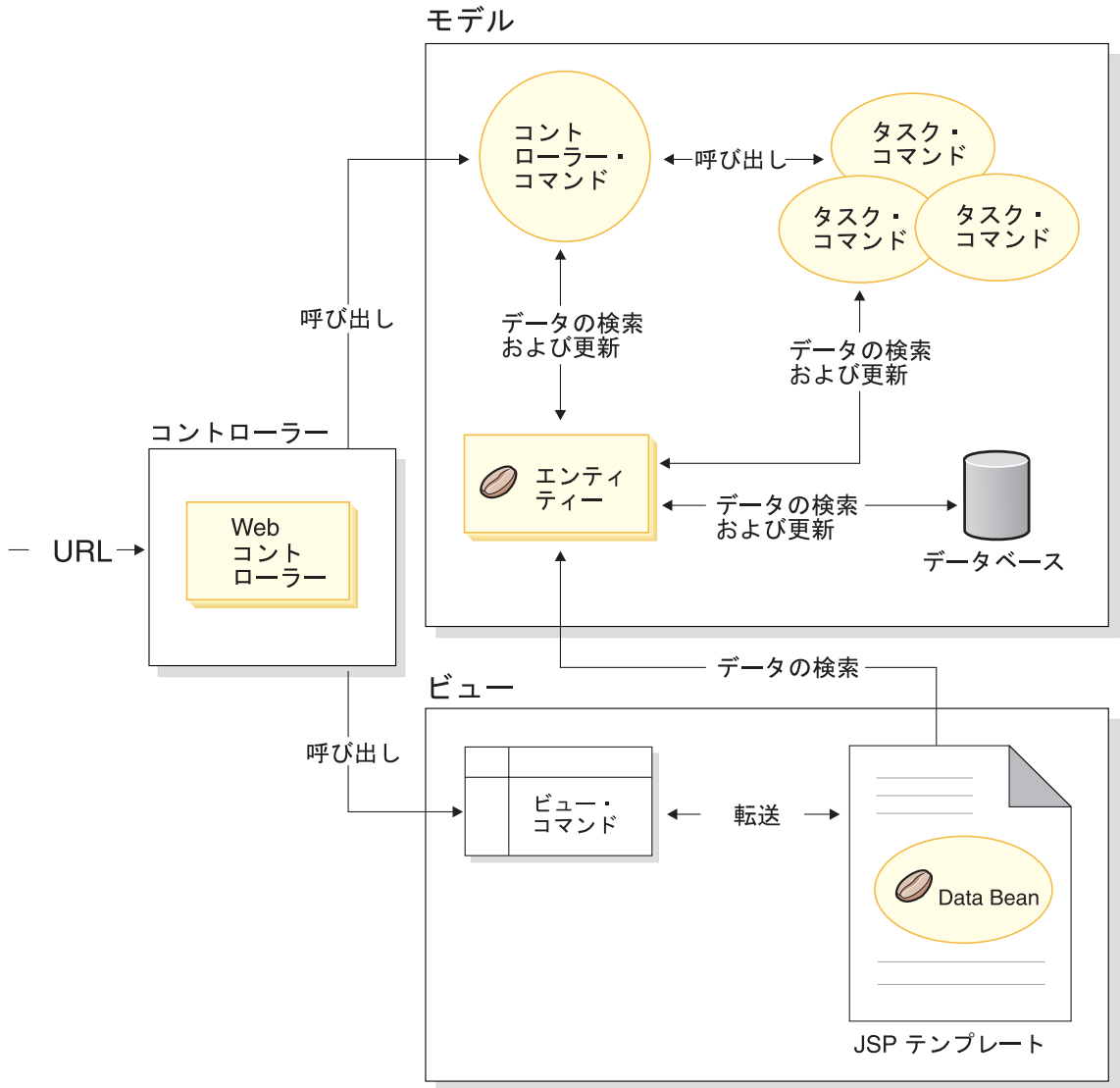


図 6.

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・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、インターフェイス・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、入力プロパティの設定、execute() メソッドの呼び出し、および出力プロパティ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の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. リクエスト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。

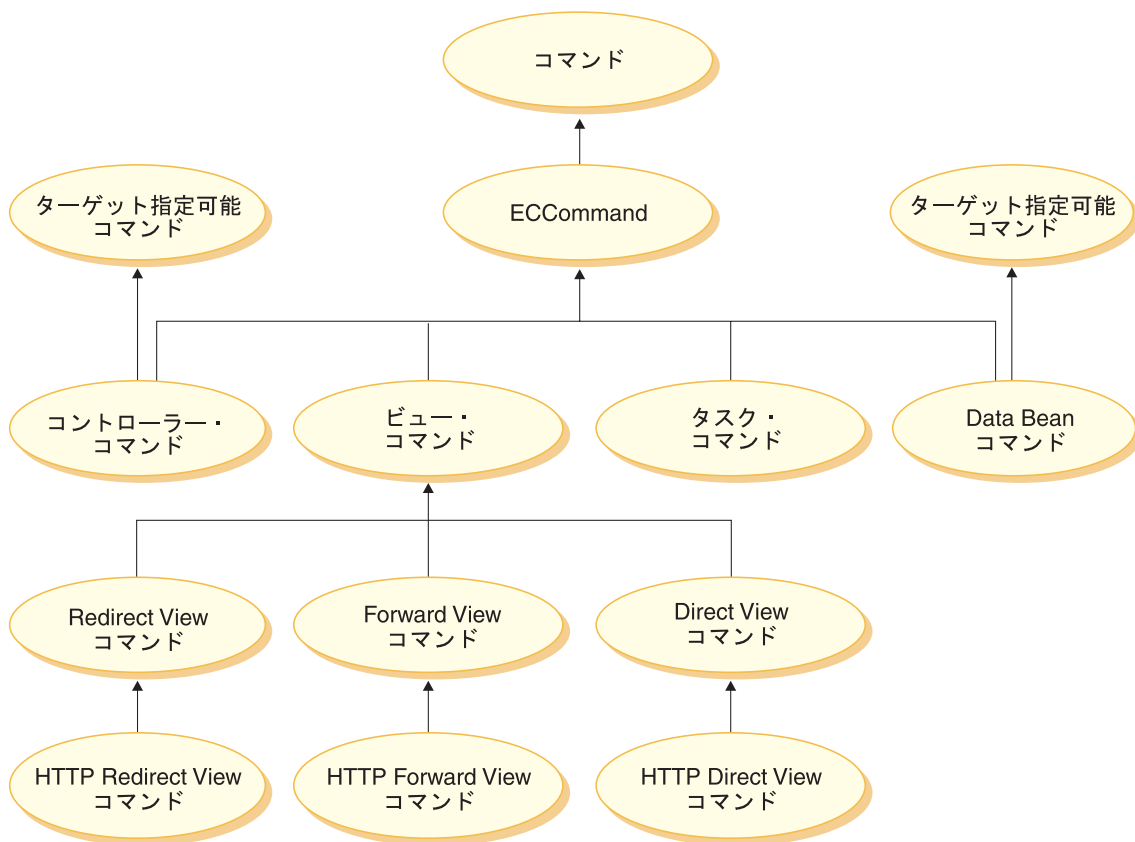


図 7.

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、オーダー処理用の *OrderProcessCmd* コマンドや、ユーザーがログオンできるようにする *LogonCmd* 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。ビュー名、ストア ID、および装置タイプに基づき、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。

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で実行されます。

Data Bean コマンド

Data Bean コマンドは、Data 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。Data Bean コマンドの主な機能は、Data Bean のフィールドにデータを取り込むことです。

Data 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、それはブラウザ内の URL スタックを変更します。再ロード・キーが入力されると、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せん。

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べきです。コマンド・レジストリーの詳細については、30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, storeId)
```

ここで、*interfaceName* は新規コマンド Bean のインターフェース名で、*storeId* はこのコマンドをインプリメントする必要のあるストアの ID です。一般に、ストア ID は、`commandContext.getStoreId()` メソッドを使用して検索できます。

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、197 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。

ネストされたコントローラー・コマンド

コマンド・ファクトリーは、ほとんどの場合、タスク・コマンドのインスタンスを作成するときに使用しますが、1つのコントローラー・コマンド内で使用して、別のコントローラー・コマンドのインスタンスを作成することも可能です。つまり、別のコントローラー・コマンド内に含まれる特定のコントローラー・コマンドを呼び出すときに使用するということです。

タスク・コマンドをインスタンス化するときの構文とコントローラー・コマンドをインスタンス化するときの構文は同じです。つまり、どちらのシナリオでも、コマンドのインターフェースの名前とストア ID を指定します。

あるコントローラー・コマンドを別のコントローラー・コマンド内にネストする場合、以下の点に注意してください。

- ネストされたコマンドをインスタンス化したら、その `setCommandContext` メソッドを呼び出し、現在のコマンド・コンテキストを渡します。ネストされたコマンドに異なる要求プロパティの設定を渡す場合で、これらのパラメーターがコマンド・コンテキストに影響する場合、ネストされたコマンドをインスタンス化する前に、コマンド・コンテキストを複製する必要があります。これで、外側のコマンド向けに、コマンド・コンテキスト情報が保持されます。
- 理想的には、ネストされたコマンドの `setRequestProperties` メソッドを呼び出し、入力プロパティを含む `TypedProperties` オブジェクトを渡します。そうしない場合、コマンドのインターフェースで定義された個々の設定メソッドを使用して、必要なプロパティを設定できます。
- 入力プロパティを設定したら、ネストされたコマンドの実行メソッドを呼び出します。
- コントローラー・コマンドはすべて、処理の完了時にビューを戻すことになっているため、外側のコマンドは、ネストされたコマンドによって戻されるビューに関係しません。
- ネストされたコマンドは、外側のコマンドのトランザクション範囲内で実行されます。

ネストされたコントローラー・コマンドの一例として、次のコード断片を考慮してください。この例は、外側のコマンドのメソッドと、そこでコマンド・ファクトリーを使用して、2番目のコントローラー・コマンドをインスタンス化する方法を示しています。

```
yourControllerCmd ctrlCmd = null;

public void processAndCallOtherCommand()
    throws ECEException
{
    ctrlCmd = (yourControllerCmd)CommandFactory.createCommand(
        com.yourcompany.commands.yourControllerCmd, this.getStoreId());
```

```

        ctrlCmd.setCommandContext(this.getCommandContext());
        ctrlCmd.setRequestProperties(this.getRequestProperties());
        ctrlCmd.execute();
    }

```

別の例として、ネストされたコマンドが、最初のストアとは異なるストアで実行されるケースを考慮してください。この場合、外側のコマンドからのコマンド・コンテキストは、内側のコマンドによって上書きされないよう、保持されなければなりません。

```

// Make a clone to preserve the command context of the outer command
CommandContext cloneCmdCtx = (CommandContext)this.getCommandContext().clone();

//Now pass in a new set of request properties to the cloned command context
cloneCmdCtx.setRequestProperties(reqProp);

yourControllerCmd ctrlCmd = null;

public void processAndCallOtherCommandForOtherStore(int aStoreId)
    throws ECEException
{
    ctrlCmd = (yourControllerCmd)CommandFactory.createCommand(
        com.yourcompany.commands.yourControllerCmd, aStoreId;
    ctrlCmd.setCommandContext(cloneCmdCtx);
    ctrlCmd.setRequestProperties(reqProp);
    ctrlCmd.execute();
}

```

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。

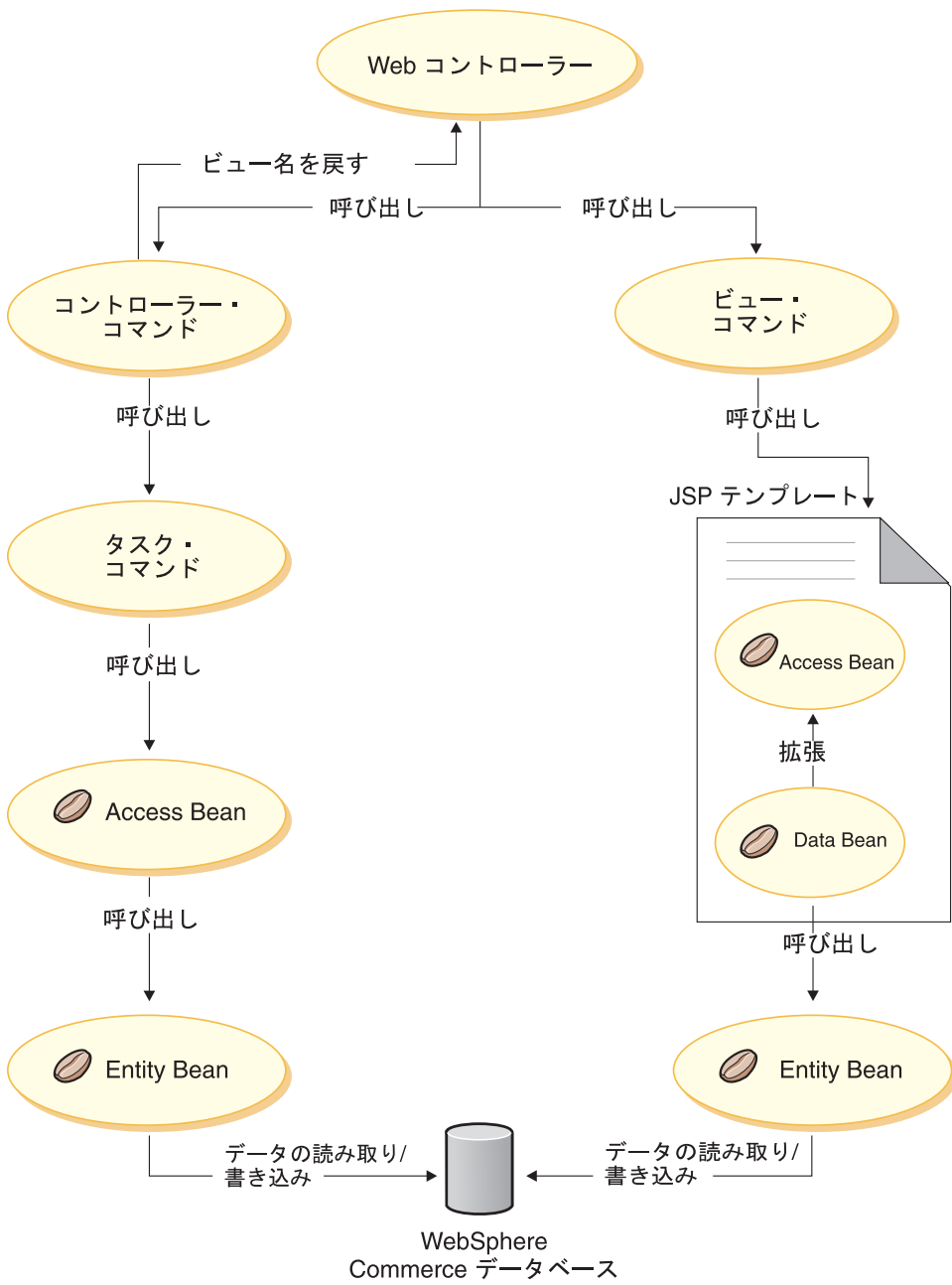


図 8.

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ため、ビジネス処理の中のある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、Access 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、Access Bean を呼び出すことができます。要求は、Access Bean から Entity Bean にフローしていきます。Entity 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 Data Bean が使用されます。Data Bean は Data Bean マネージャーによってアクティブにされます。Data Bean (Access Bean から拡張されたもの) は、それに対応する Entity 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、Entity 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:  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、180 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&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	たとえば、MyNewCommand または com.ibm.commerce.catalog.commands.ProductDisplayCmd
STOREENT_ID	ストアのエンティティ ID	すべてのストアで使用され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. ProductDisplayCmdImpl
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	たとえば、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 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。

列名	説明	注釈
DESCRIPTION	コマンドの説明	たとえば、This command is used for testing purposes など。
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.mycompany.commands.MyNewCommand', 'This is a test command',
'com.mycompany.commands.MyNewCommandImpl', 'myDefaultParm1=myDefaultVal1',
'0000-12-01', 'Local')
```

文字列値は、単一引用符で囲みます。

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。

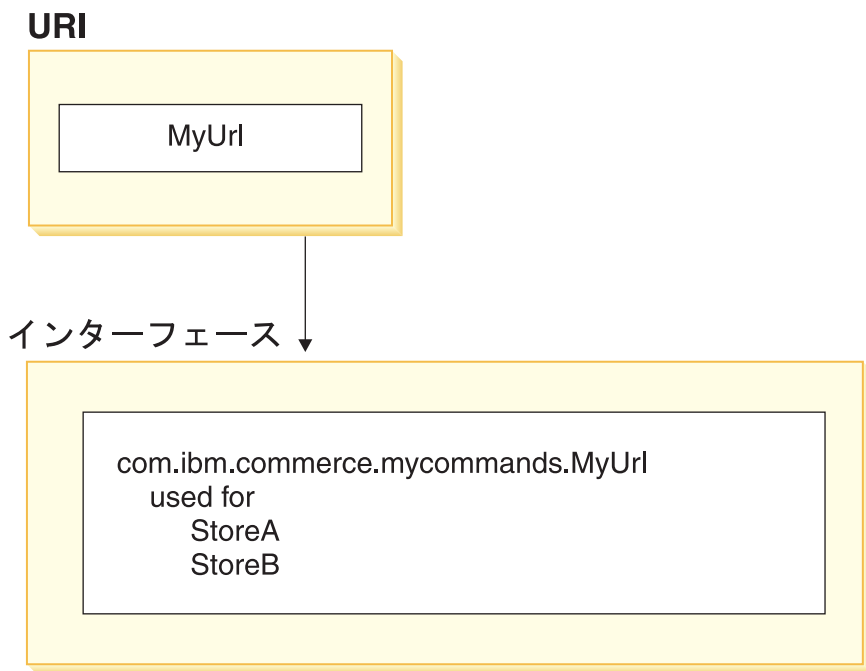


図9.

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。

URI

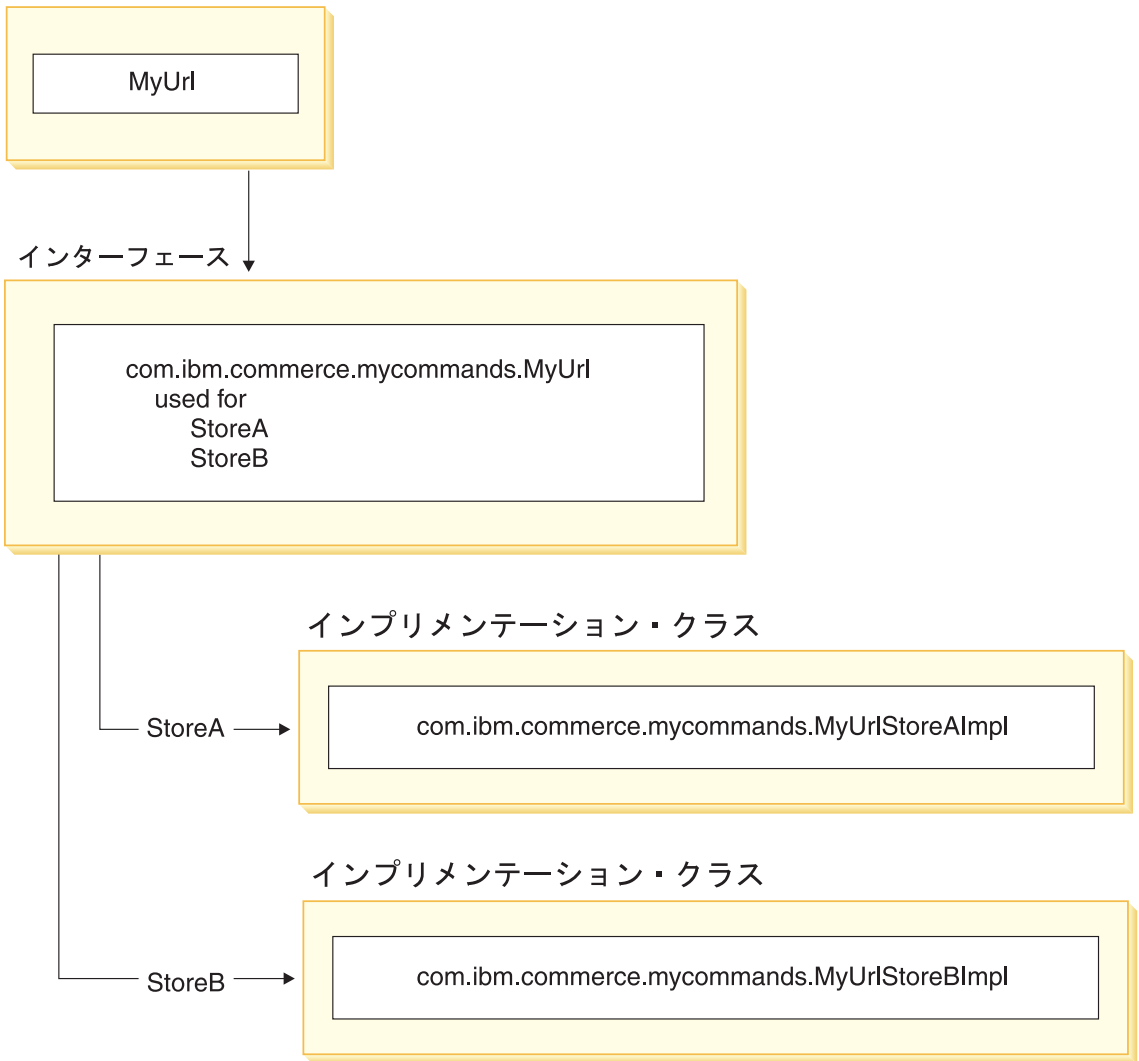


図 10.

VIEWREG テーブル

VIEWREG テーブルにより、装置固有およびストア固有のビュー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

セッ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、様々な装置に別々のビューを戻せるようになります。

ビュー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・インプリメンテーションを VIEWREG テーブルから判別します。複数のビュー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。
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。 <ul style="list-style-type: none"> • 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 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。

列名	説明	注釈
PROPERTIES	コマンドへの入力プロパティとして設定する、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	
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 テーブルにビューを登録する必要があります。

- ビュー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 列にプロパティを設定する場合

登録されたビューは、ビュー名を使用してビュー・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・ファイル名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、ビュー名を使用してビューにアクセスすることもできますし、ビュー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・ファイル名を使用して) 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ビュー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ができず。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、MyNewView 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
('MyNewView', -1, 5, 'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl',
'docname=MyNewView.jsp', 'A test view.', 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。

VIEW NAME	INTERFACE - NAME	CLASSNAME	PROPERTIES
ProductDisplay View	Forward View コマンド	HttpForwardView CommandImpl	docname= UserArea/ ServiceSection/ InterestItemList Subsection/ WishListDisplay.jsp
Generic Application Error	Forward View コマンド	HttpForwardView CommandImpl	docname =Generic Application Error.jsp&storeDir=no
GenericSystem Error	Forward View コマンド	HttpForwardView CommandImpl	docname = Generic System Error.jsp &storeDir=no
LogonForm	Forward View コマンド	HttpForwardView CommandImpl	docname = LoginForm.jsp &generic=true &storeDir=no

注: VIEWNAME、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。
- 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 *EApplication* 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、 `GenericApplicationError.jsp` テンプレートが表示されることとなります。
- 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 `ECSysSystem` 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、 `GenericSystemError.jsp` テンプレートが表示されることとなります。
- 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 Data Bean

Data 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、Data Bean は、WebSphere Commerce Entity Bean の単純な表現を提供します。Data Bean は、Entity Bean から検索したり、Entity Bean の中で設定

したりすることのできるプロパティをカプセル化します。そのようにして、Data 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。

Data 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を定義するものです。また、*BeanInfo* クラスでは、プロパティ名が *WebSphere Commerce* のすべてのサポート言語で提供されているので、Data Bean を多文化対応型のサイトで使用することも可能です。

Data Bean は、以下の呼び出しでアクティブにされます。

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

ここで、*data_bean* はアクティブにする Data Bean で、*request* は *HttpServletRequest* オブジェクトです。

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と、国際化問題を考慮しなければなりません。国際化についての詳細は、「*WebSphere Commerce* ストア開発者ガイド」を参照してください。

Data Bean のセキュリティに関する考慮事項

Data 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"
```

Data Bean のタイプ

Data 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。Data Bean には、Smart Data Bean とコマンド Data Bean の 2 種類があります。

Smart Data 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 Data Bean が良好なパフォーマンスを発揮するのは、Access 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 Data 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 Smart Data Bean は、対応する Entity Bean 用の Access Bean から拡張されなければならないので、`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 Data Bean は ProductAccessBean Access Bean を拡張したものであり、この Access Bean は Product Entity Bean に対応します。

データベース・アクセスを必要としない Smart Data Bean もあります。たとえば、PropertyResource Smart Data 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、Smart Data Bean は SmartDataBeanImpl クラスを拡張しなければなりません。

コマンド Data 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 Data 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らずなく、Data Bean のすべての属性を一度に検索します。その結果、Data 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 Data 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 Data Bean は大変都合の良いものとなります。

コマンド Data Bean も自分に対応する Access 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。

Data Bean のインターフェース

Data 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、Data 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、Data 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 Data Bean は、関連する Data 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。Smart Data Bean は、通常、対応する Entity Bean の Access Bean から拡張されます。Smart Data Bean がアクティブにされると、Data Bean マネージャーが Data 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、Data 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、Data Bean が Entity Bean の Access Bean クラスから拡張されたものである場合、Data Bean は refreshCopyHelper メソッドを呼び出します。対応する Entity Bean の属性は、すべて Smart Data Bean に自動的に取り込まれます。ただし、Entity 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。Smart Data 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、Data Bean コマンドを作成する必要がない。
- Entity Bean に新しいフィールドが追加されても、Data Bean に変更を加える必要がない。Entity Bean を変更したら、その後、Access Bean を再生成する必要があります (WebSphere Studio Application Developer のツールを使用)。Access Bean が再生成され次第、自動的にすべての新しい属性が Smart Data Bean で使用できるようになります。
- Entity 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、Smart Data 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。

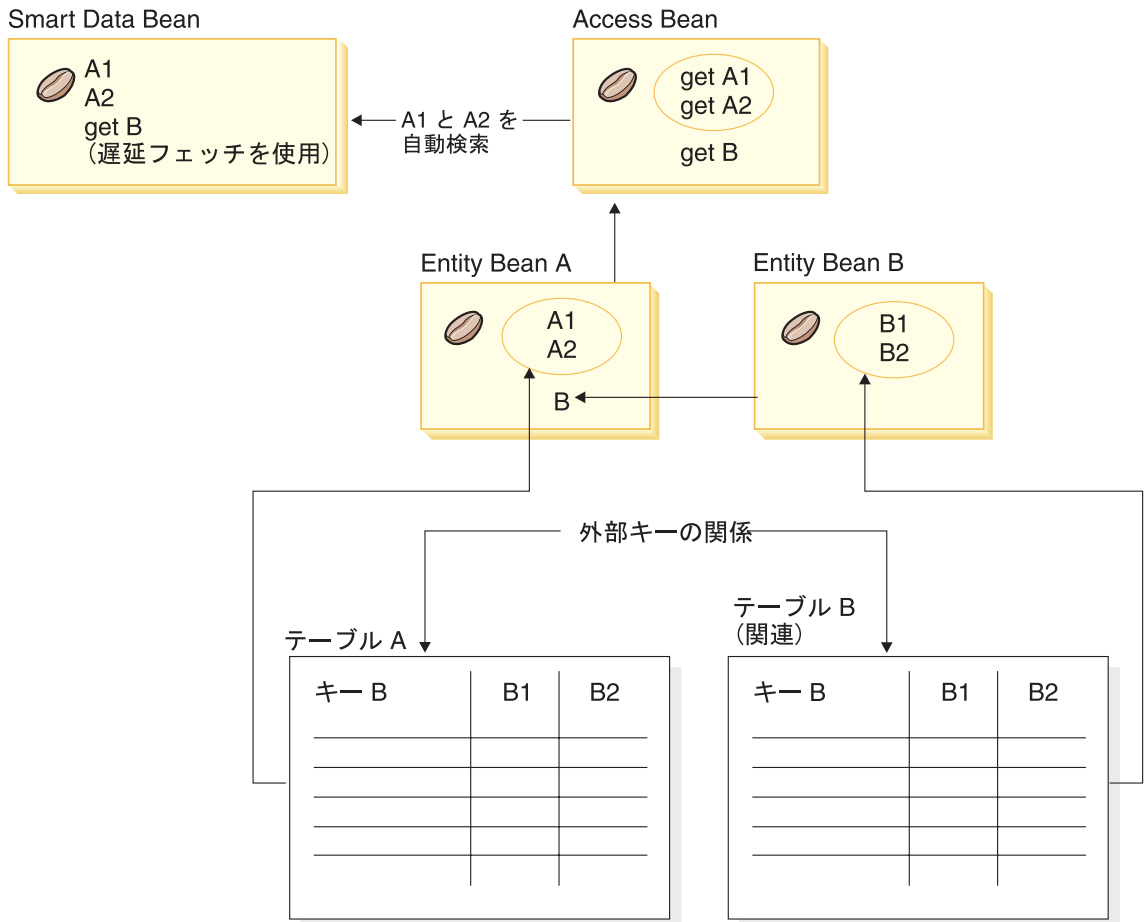


図 11.

遅延フェッチ検索のインプリメントについては、46 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 Data Bean は、Data Bean コマンドからデータを検索します。このタイプの Data 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、Data Bean コマンドに依存します。Data Bean は、Data 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 Data 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 Data Bean を Data Bean マネージャーに渡すことによって、Data Bean をアクティブにします。それから、Data Bean マネージャーは、Data 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。Data Bean をアクティブにするには、次のコードが必要になることに注意してください。

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

ここで、*data_bean* はアクティブにする Data Bean で、*request* は `HttpServletRequest` オブジェクトです。

BeanInfo クラス

BeanInfo クラスなしでは、Data 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、Data 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、Data 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 Data 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。

Data Bean の活動化

Data 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 テンプレート内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 `Data 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も設定できます。

`Data Bean` が `Data Bean` マネージャーによってアクティブにされると、コントローラー・コマンドが実行され、応答プロパティが JSP テンプレートで使用可能になります。

この `Data Bean` をアクティブにする前に `setRequestProperties` メソッドを使用しない場合、要求オブジェクトからのパラメーターは `Bean` に渡されるため、コントローラー・コマンドにも渡されます。しかし、この `Data Bean` をアクティブにする前に `setRequestProperties` メソッドを実際に呼び出す場合は、指定したプロパティ (`setRequestProperties` メソッドに渡されたもの) と、`CMDREG` テーブルの `PROPERTIES` 列で指定されたデフォルトのプロパティだけが、コマンドで使用できるようになります。

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。

同じ `Data Bean` のインスタンスには、前に使用したデータおよび状態情報が含まれるため、その同じインスタンスを再使用して他のコントローラー・コマンドを呼び出さないようにします。

遅延フェッチ・データ検索

アクティブにされた `Data Bean` には、`Data Bean` コマンドや `Data 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、`Data Bean` の対応する `Entity Bean` から検索されます。`Entity 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。

アクティブになる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。

多数の関連商品販売、上位商品販売またはアクセサリ製品 (関連オブジェクト) を含む製品 `Data Bean` について考えてください。製品 `Data Bean` がアクティブに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が渡される方法を示しています。

CMREG

INTERFACENAME	PROPERTIES
com.ibm.xxx.NewCommand	parm1=1&parm2=2

CCPd: parm1=1&parm2=2

VIEWREG

INTERFACENAME	PROPERTIES
com.ibm.command.ForwardViewCommand	docname=NewView.jsp

VPd: docName=NewView.jsp

URL: http://hostname/webapp/wcs/stores/servlet/NewCommand?storeId=1&...

CCPu: storeID=1&...

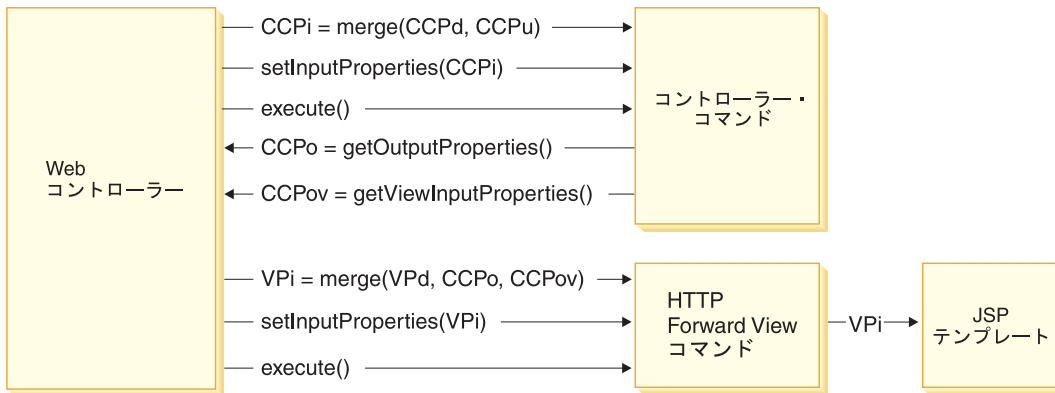


図 12.

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 (CCPu) と、CMREG テーブル内にあるコントローラー・コマンドのエントリ (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 (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 `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です。出力プロパティの 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・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 (`CCPi`) に設定されます。これらのプロパティ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・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) を、ビュー・コマンドの入力プロパティ (`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 `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。このメソッドの詳細は、WebSphere Commerce Production and Development オンライン・ヘルプの『[参照 \(Reference\)](#)』トピックを参照してください。

必要なプロパティ設定

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=SearchResult.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、たくさん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 (EJB) Version 1.1 コンポーネント・アーキテクチャーに基づく Entity Bean を使用します。これらの Entity 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。

WebSphere Studio Application Developer 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。

続くセクションは、EJB 1.1 仕様の WebSphere Commerce 永続オブジェクト・モデル・インプリメンテーションをインプリメントすることを前提とします。

WebSphere Commerce Entity Bean のインプリメンテーション

WebSphere Commerce Entity 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 Enterprise Bean (Entity Bean と Session Bean) を定義しています。Entity Bean-managed はさらに、コンテナ管理パーシスタンス (CMP) Bean と Bean 管理パーシスタンス (BMP) Bean に分けられます。

ほとんどの WebSphere Commerce Entity Bean は、CMP Entity Bean です。少数のステートレス Session 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 Entity Bean を使用する 1 つの利点は、WebSphere Studio Application Developer 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、Entity 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。

WebSphere Studio Application Developer では、EJB 1.1 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、Enterprise Bean は、同じグループに存在する別の Enterprise Bean から、プロパティ、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 Entity 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 Entity Bean で利用されます。WebSphere Studio Application Developer が提供するアソシエーション機能は、WebSphere Commerce Entity Bean では使用されません。独自の Entity Bean を開発する際は、WebSphere Studio Application Developer 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。WebSphere Studio Application Developer 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 Enterprise Bean 内で設定することにより、複数の Enterprise 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 Enterprise Bean セット (private と public) を備えています。private Enterprise 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 Enterprise 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 Enterprise Bean は、以下の EJB モジュールに編成されています。

- Catalog-ProductManagementData
- Enablement-RelationshipManagementData
- Marketing-CampaignsAndScenarioMarketingData
- Marketing-CustomerProfilingAndSegmentationData
- Member-MemberManagementData
- Merchandising-PromotionsAndDiscountsData
- Order-OrderCaptureData
- Order-OrderManagementData
- Trading-AuctionsAndRFQsData

上記のリストの EJB モジュールのいくつかには、Session 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、Session Bean クラスは変更しないでください。必要な場合は、WebSphereCommerceServerExtensionsData EJB モジュール内に新規 Session Bean を作成することができます。新規 Session Bean の作成については詳しくは、82 ページの『新規 Session Bean の作成』を参照してください。

WebSphere Commerce Enterprise Bean のデプロイメント記述子



EJB デプロイメント記述子には、Enterprise Bean のデプロイメント設定が含まれています。 WebSphere Studio Application Developer には、このデプロイメント情報を変更するために使用できる、 EJB デプロイメント記述子エディターが用意されています。

新規 Enterprise Bean (Entity または Session Bean) を作成する場合、デプロイメント記述子情報は、WebSphere Studio Application Developer の J2EE パースペクティブの「J2EE 階層 (J2EE Hierarchy)」ビュー内で設定されます。次のようにして、WebSphere Commerce Bean の EJB デプロイメント記述子を表示できます。

1. WebSphere Studio Application Developer をオープンし、J2EE パースペクティブに切り替えます。
2. 「J2EE 階層 (J2EE Hierarchy)」ビューを使用して、デプロイメント記述子情報を表示する EJB モジュールを探し出します。
3. *EJB_moduleName* EJB モジュールを右クリックして、「**オープンに使用 (Open With)**」 > 「**Deployment Descriptor Editor**」の順に選択します。
Deployment Descriptor Editor がオープンします。
4. 「Beans」タブを選択し、以下の点を確認します。
 - a. Bean のリストから Bean を 1 つ選択します。その Bean の情報が他のフィールドに取り込まれます。
 - b. その Bean は、Container Managed Entity 1.x Bean として表示されていなければなりません。
 - c. 「再入可能 (Reentrant)」チェック・ボックスはチェックしないようにします。
 - d. 「WebSphere バインディング (WebSphere Bindings)」セクションでは、JNDI 名のデフォルト値が使用されます。
 - e. 「WebSphere 拡張 (WebSphere Extensions)」セクションでは、並行性の制御用の「あいまいロックを使用可能にする (Enable optimistic locking)」が使用可能になっていません。
 - f. さらに、「ファインダー (Finders)」セクションで、Bean のファインダーを表示することも確認してください。
5. 「アセンブリー記述子 (Assembly Descriptor)」タブを選択し、以下の点を確認します。
 - a. 「メソッド許可 (Method Permissions)」セクションでは、WCSecurityRole が Enterprise Bean のすべてのメソッドに割り当てられます。
 - b. 「コンテナー・トランザクション (Container Transactions)」セクションでは、Enterprise Bean のすべてのメソッドに「必須 (Required)」が指定されます。
6. 「アクセス (Access)」タブを選択し、以下の点を確認します。
 - a. 「Entities 1.x のアクセス・インテント (Access Intent for Entities 1.x)」セクションでは、すべての読み取り専用メソッドが定義されています。たとえば、`_copyFromEJB()` および `handcoded getter` メソッドが、読み取り専用メソッドに割

り当てられています。独自の Entity Bean を作成する場合、該当するメソッドを読み取り専用にするようにします。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。

b. 「分離レベル (Isolation Level)」は、次のように設定します。

-  反復可能読み取り (Repeatable read)
-  コミット済み読み取り (Read committed)

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 public Enterprise Bean を拡張する
- 新規 Entity Bean を作成する
- 新規ステートレス Session 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 public Entity 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規 Entity 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 Entity Bean に追加します。新規 Entity Bean 用のデプロイメント・コードと Access 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、Access 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 public Entity 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 Enterprise 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 public Entity 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 Entity Bean 用のデプロイメント・コードと Access Bean

を生成します。アプリケーションが Access 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、自己所有の家、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・リポジトリを調べてみると、Member-MemberManagementData EJB モジュールに、“Demographics” という Enterprise 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規 Entity 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、新規 Userres Entity 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 Access 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 Access 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 Access Bean だけでなく、新しい Access 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。

USERDEMO テーブル

Demographics
Entity Bean

USERRES テーブル

Userres
Entity Bean

図 13.

表示テンプレートの観点からすると、Data 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) Entity Bean の Access Bean を拡張した新規 Data Bean を作成するのが望ましいでしょう。その Data Bean では、新しい Access 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 Data 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。

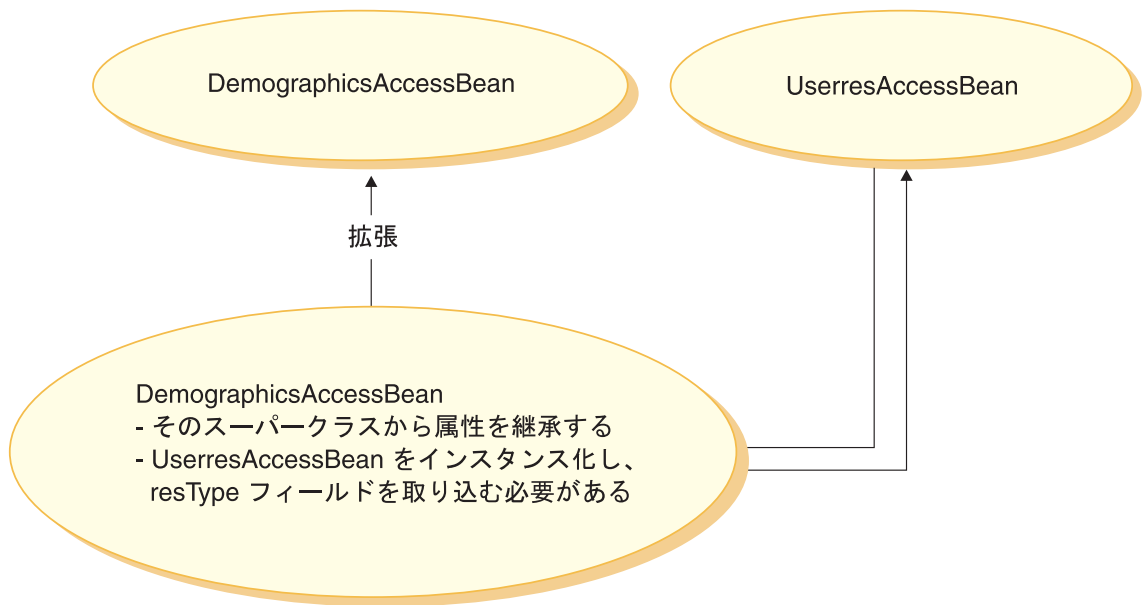


図 14.

2 番目の方法 (既存のコードを変更する方法) の場合は、Demographics Entity 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 Access 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となります。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 Enterprise Bean を変更する手順を示したものです。

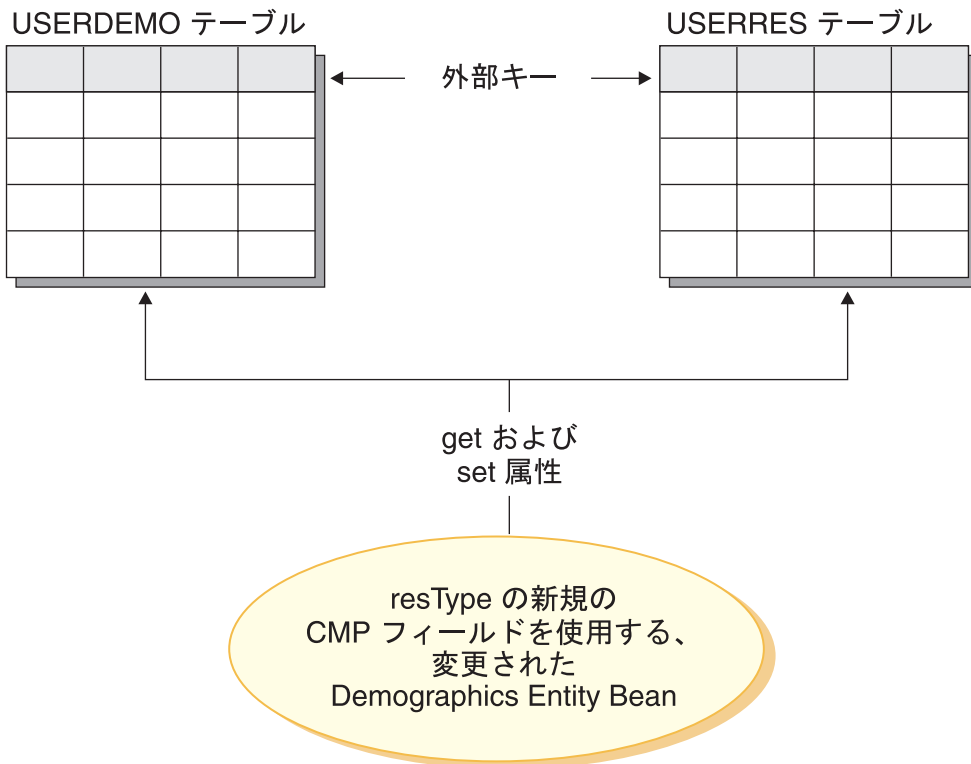


図 15.

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 Data 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 Entity 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ます。

2 番目の方法は、既存の WebSphere Commerce コードを変更しているため、新しいバージョンの WebSphere Commerce 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 WebSphere Commerce ワークスペースをインポートするときに、Enterprise 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. WebSphere Studio Application Developer 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 public Enterprise 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 Enterprise Bean 用のデプロイメント・コードと Access 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 Session Bean の使用法

WebSphere Commerce の強みの 1 つに、コンテナ管理パーシスタンス (CMP) Entity Bean を利用できる機能があります。CMP Entity Bean は、WebSphere Studio Application Developer に備えられてい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 Entity Bean は非常によい選択です。また、CMP Entity 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 Entity Bean を使用しています。

ただし、Session Bean JDBC helper の使用をお勧めする場合があります。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 の事例といえます。
- 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 の事例といえます。
- 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 の事例といえます。

詳細について以下に説明します。

Session 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 Session Bean を使用する場合、Session 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 Session Bean を使用し、その Session Bean 内で、Entity 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 Session 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、Access Bean をインスタンス化する
- 検索される各列について、Access Bean 内に対応する属性を設定する

Access Bean が戻される時、Access Bean が Session Bean 内のファインダー・メソッドによって戻されたのか、Entity 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、Session 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、Session Bean 内のファインダー・メソッドを呼び出しているのか、Entity 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)
```

EJB 1.1 仕様の Enterprise Bean 用 WebSphere Studio Application Developer に含まれるツールは、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、Session Bean の使用をお勧めします。その Session 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、Access 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、Access Bean 内に対応する属性を設定する

Access Bean は、例外を送る getter メソッドをキャッシュに入れ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、Access 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、Session 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 Entity 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、Session Bean の helper メソッドを作成してください。この Session 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。

public Entity Bean の拡張

このセクションでは、WebSphere Commerce の public Enterprise 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 Entity Bean のインプリメンテーション・クラスを示しています。

Enterprise Bean のインプリメンテーション

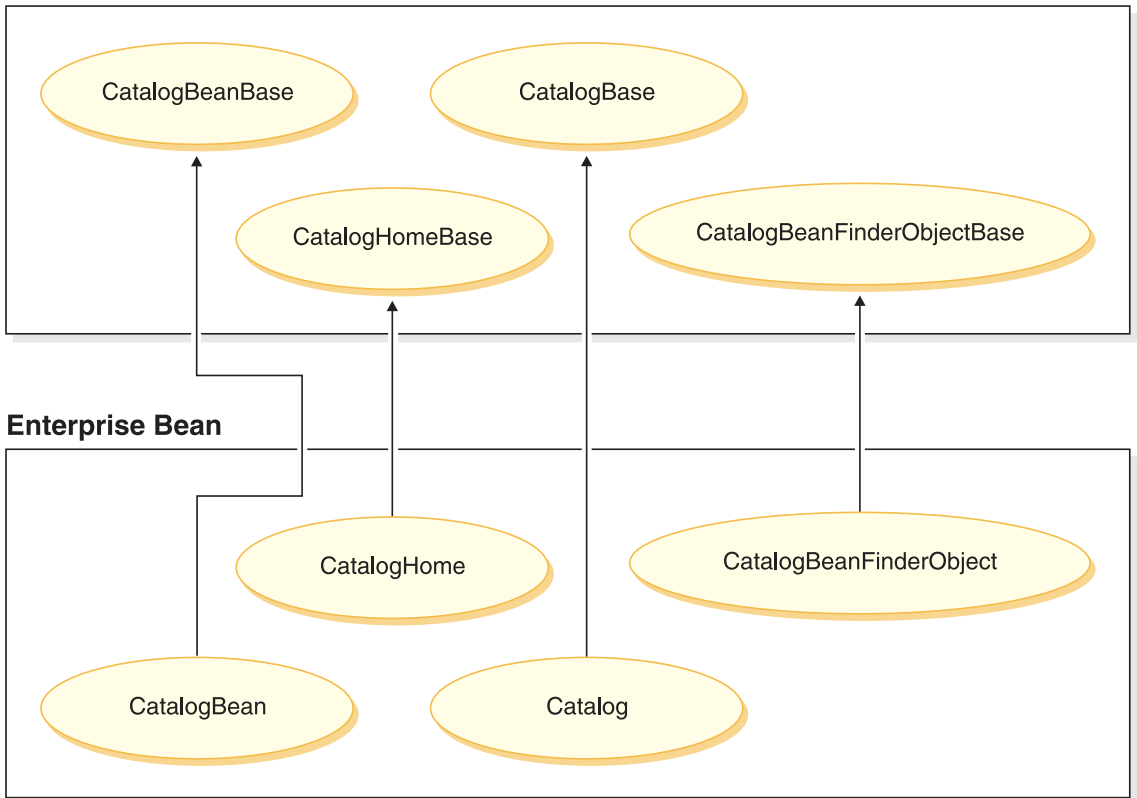


図 17.

他の Entity 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 Entity Bean にも適用されます。この図を別の Entity Bean に適用するには、“Catalog” をその Entity Bean の名前に置き換えます。たとえば、InterestItemBean クラスは InterestItemBeanBase 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、public Enterprise 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 Java パッケージで定義されています。

WebSphere Commerce ワークスペースには、これらのスーパークラスおよびインターフェースすべてのバイナリー・コードが含まれています。この子クラスおよびインターフ

エースに対して変更を加えることができます。一般に、変更は `com.ibm.commerce.xxx.objects` と、 `com.ibm.commerce.xxx.objsrc` パッケージで行えます (ここで、 `xxx` はコンポーネント名です)。

新しいファインダー・メソッドを `public Enterprise 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、`findXa_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。新しいファインダーを追加するときには、デプロイメント記述子エディターが使用されます。

既存の `WebSphere Commerce public Entity 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`、`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。

“findForUpdate” タイプのファインダーを含む Public Entity Bean:

`WebSphere Commerce public Entity Bean` に “findForUpdate” タイプのファインダーが含まれる場合、作成した新しいテーブルまでの 2 次マップを作成することでその `Bean` に新しいフィールドを追加することはできません。これは、実際に 2 次マップを作成する場合、生成される SQL ステートメントは無効になり、`Bean` は希望した機能を失くなるためです。そのような `Bean` によって表されるオブジェクト・モデルの一部を強化する場合、新規 `Entity Bean` を作成し、カスタマイズしたコードの中で、元の `Bean` とその新しい `Bean` を使用する必要があります。

CMP Enterprise 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 `Enterprise 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、新規 `CMP Enterprise 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。

`WebSphere Commerce` ワークスペースには、新規 `Enterprise Bean` のために事前定義された `EJB` プロジェクトがあるので、別の `EJB` プロジェクトを作成する必要はありません。新規 `Enterprise Bean` は、`WebSphereCommerceServerExtensionsData` `EJB` プロジェクトに置くようにします。後で、カスタマイズした `Bean` をデプロイするときに、`WebSphereCommerceServerExtensionsData.jar` `JAR` ファイルを作成し、`WebSphere Application Server` で実行している `WebSphere Commerce` エンタープライズ・アプリケ

ーションの既存の JAR ファイルを置き換えます。このパッケージ化の規則を使用することにより、デプロイメントは著しく単純化されます。

新規 CMP Enterprise Bean を作成するには、WebSphere Studio Application Developer で以下のステップを実行しなければなりません。

1. 「Enterprise Bean の作成 (Enterprise Bean Creation)」ウィザードを使用して、新規 CMP Enterprise Bean を作成します。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。
2. 新規 Bean のトランザクション分離レベルを設定します。
3. 新規 Bean のセキュリティ ID を設定します。
4. エンティティのコンテキスト・メソッドを変更します。
5. 必要であれば、EJB デプロイメント記述子エディターを使用して、新しいファインダーを定義します。
6. 必要に応じて、新規の ejbCreate メソッドを作成し、ejbCreate メソッドを、Enterprise 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 Enterprise 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。
7. Bean が WebSphere Commerce アクセス制御システムによって保護されている場合、Bean に必要なアクセス制御メソッドをインプリメントしてください。Enterprise Bean でのアクセス制御のインプリメントの詳細は、97 ページの『第 4 章 アクセス制御』を参照してください。さらに、自分の Access Bean を作成した後で、アクセス制御をインプリメントできます。
8. Enterprise 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. Enterprise Bean の対応する Access Bean を生成します。
10. Enterprise Bean のデプロイメント・コードを生成します。
11. WebSphere Studio Application Developer で、汎用テスト・クライアントを使用して Bean をテストします。

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。これらのセクションを読み進めながら、XUSERRES 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。

CMP Enterprise Bean の新規作成: 新規 CMP Enterprise Bean を作成するには、次のようにして、「Enterprise Bean の作成 (Enterprise Bean Creation)」ウィザードを使用できます。

1. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、「EJB モジュール (EJB Modules)」を拡張表示します。

2. **WebSphereCommerceServerExtensionsData** モジュールを右クリックして、「新規」>（「その他」>）「**Enterprise Bean**」を選択します。
「Enterprise Bean の作成 (Enterprise Bean Creation)」ウィザードがオープンします。
3. 「**EJB プロジェクト (EJB Project)**」ドロップダウン・リストで、「**WebSphereCommerceServerExtensionsData**」を選択して、「次へ」をクリックします。
4. 「Enterprise Bean の作成」ウィンドウで、以下のようにします。
 - a. 「**コンテナ管理パーシスタンス (CMP) フィールドを持つ Entity Bean**」を選択します。
 - b. 「**Bean 名 (Bean name)**」フィールドに、Bean の適切な名前を入力します。通常は、Bean 名は対応するデータベース・テーブルの名前と一致します。たとえば、Bean XUserRes には、XUSERRES テーブルに対応する名前を指定します。
 - c. 「**ソース・フォルダー (Source folder)**」フィールドを、指定されているデフォルト値 (ejbModule) のままにします。
 - d. 「**デフォルト・パッケージ (Default package)**」フィールドに、`com.mycompany.mycomponent.objects` と入力します。
 - e. 「次へ」をクリックします。
5. 「Enterprise Bean の詳細 (Enterprise Bean Details)」ウィンドウで、以下のようにします。
 - a. 「追加」をクリックして、データベース・テーブル中の列の新規 CMP 属性を追加します。
「CMP 属性の作成 (Create CMP Attribute)」ウィンドウがオープンします。このウィンドウで、以下のようにします。
 - 1) 「名前」フィールドで、新規 CMP フィールドに適切な名前を入力します。このフィールドをデータベース・テーブル内の対応する列にマッピングする場合で、後で「名前での一致 (Match by name)」機能を使用する場合、フィールドに列の名前を正確に指定してください (大文字小文字は区別されません)。
 - 2) 「タイプ」フィールドで、フィールドに適切なデータ・タイプを入力します。プリミティブ・データ・タイプには、ラッパー・クラスを使用する必要があるので注意してください (たとえば、`long` データ・タイプではなく、`java.lang.Long` データ・タイプを使用する)。
 - 3) フィールドが基本キーである場合、「**キー・フィールド (Key Field)**」チェック・ボックスをチェックし、「適用」をクリックします。
 - 4) フィールドが基本キーではない場合、「**getter および setter メソッドを使ったアクセス**」チェック・ボックスをチェックします。
 - 5) フィールドが基本キーではない場合、「**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**」チェック・ボックスのチェックを外します。「**getter を読み取り専用にする (Make getter read-only)**」チェック・ボックスが使用不能になりますので、「適用」をクリックします。



- 6) もう一度「追加」をクリックしてステップを繰り返し、CMP フィールドを必要とするデータベース・テーブルの列ごとに新規フィールドを追加します。
 - 7) 「クローズ」をクリックしてこのウィンドウをクローズします。
- b. 「鍵クラスに単一の鍵属性タイプを使用 (Use the single key attribute type for the key class)」チェック・ボックスをクリアしてから、「次へ」をクリックします。
6. 「EJB Java クラスの詳細 (EJB Java Class Details)」ウィンドウで、以下のようになります。
 - a. Bean のスーパークラスを選択するには、「ブラウズ」をクリックします。「タイプの選択 (Type Selection)」ウィンドウがオープンします。
 - b. 「使用するクラスの選択: (任意) (Select a class using: (any))」フィールドに、ECEntityBean と入力し、「OK」をクリックします。スーパークラスとして com.ibm.commerce.base.objects.ECEntityBean が選択されます。
 - c. 新規 Enterprise Bean を WebSphere Commerce アクセス制御フレームワークの下で保護する場合、「追加」をクリックして、リモート・インターフェースが拡張する必要のあるインターフェースを指定します。「タイプの選択 (Type Selection)」ウィンドウがオープンします。
 - d. 「使用するクラスの選択: (任意) (Select a class using: (any))」フィールドに、Protectable と入力し、「OK」をクリックします。com.ibm.commerce.security.Protectable が選択されます。アクセス制御下の新規リソースを保護するには、このインターフェースが必要です。
 - e. 「終了」をクリックします。

Bean クラスで、WebSphere Studio Application Developer 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・コンテキスト・フィールドを提供します。新規 Entity Bean では、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、新規 Entity Bean から、生成された EntityContext を除去してください。このことについては、続くセクションで説明します。

com.ibm.commerce.base.objects.ECEntityBean をスーパークラスとして指定すると、使用する Bean は特定の機能を継承します。以下のコード例は、そのような機能を示しています。

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

トランザクション分離レベルの設定: Bean のトランザクション分離レベルは、開発データベース・タイプに適した値に設定する必要があります。トランザクション分離レベルを設定するには、以下のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「**EJB モジュール (EJB Modules)**」を拡張表示します。
2. **WebSphereCommerceServerExtensionsData** プロジェクトをダブルクリックし、Deployment Descriptor Editor を使用してオープンします。
3. 「アクセス」タブをクリックします。
4. 「分離レベル (Isolation Level)」テキスト・ボックスの隣の「追加」をクリックします。
「分離レベルの追加 (Add Isolation Level)」ウィンドウがオープンします。
5.  「反復可能読み取り (**Repeatable Read**)」を選択してから、「次へ」をクリックします。
 「コミット済み読み取り (**Read Committed**)」を選択してから、「次へ」をクリックします。
6. 「検出された Bean (**Beans found**)」リストから *yourNewBean* Bean を選択してから、「次へ」をクリックします。
7. 「検出されたメソッド (**Methods found**)」リストから、*yourNewBean* を選択してそのメソッドをすべて選択し、「終了」をクリックします。
8. ここまでの作業を保管し (Ctrl + S)、エディターはオープンしたままにします。

Bean のセキュリティー ID の設定: 次に、以下のようにして、Bean のセキュリティー ID を設定します。

1. Deployment Descriptor Editor で、「アクセス」タブを選択してあることを確認します。
2. 「セキュリティー ID (Security Identity)」テキスト・ボックスの隣の「追加」をクリックします。
「セキュリティー ID の追加 (Add Security Identity)」ウィンドウがオープンします。
3. 「EJB サーバーの ID の使用 (**Use identity of EJB server**)」を選択してから、「次へ」をクリックします。
4. 「検出された Bean (**Beans found**)」リストから *yourNewBean* Bean を選択してから、「次へ」をクリックします。
5. 「検出されたメソッド (**Methods found**)」リストから、*yourNewBean* を選択してそのメソッドをすべて選択し、「終了」をクリックします。
6. ここまでの作業を保管します (Ctrl + S)。エディターはオープンしたままにしておきます。

Bean のセキュリティー役割の設定: 次に、以下のようにして、Bean 内のメソッドのセキュリティー役割を設定します。

1. Deployment Descriptor Editor で、「アセンブリー記述子 (Assembly Descriptor)」タブを選択します。
2. 「メソッド許可 (Method Permissions)」セクションで、「追加」をクリックします。
3. セキュリティー役割として **WCSecurityRole** を選択して、「次へ」をクリックします。
4. 見つかった Bean のリストから *yourNewBean* を選択し、「次へ」をクリックします。
5. 「メソッド・エレメント (Method elements)」ページで、「すべてに適用 (Apply to All)」をクリックし、「終了」をクリックします。
6. ここまでの作業を保管し (Ctrl + S)、Deployment Descriptor Editor をクローズします。

エンティティー・コンテキスト・フィールドおよびメソッドの削除: 次のステップとして、WebSphere Studio Application Developer が生成するエンティティー・コンテキストに関連したフィールドとメソッドの一部を除去します。これらのフィールドを削除する必要がある理由は、ECEntityBean の基本クラスにはこれらのメソッドの独自のインプリメンテーションがあるからです。生成されたエンティティー・コンテキストのフィールドとメソッドを削除するには、以下のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、**WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
2. *yourNewBean* Bean を拡張表示してから、*yourNewBean* **Bean** クラスをダブルクリックします。
3. 「概略 (Outline)」ビューで、以下のようにします。
 - a. 「**myEntityCtx**」フィールドを右クリックし、「削除」を選択します。
 - b. **getEntityContext()** メソッドを右クリックし、「削除」を選択します。
 - c. **setEntityContext(EntityContext)** メソッドを右クリックし、「削除」を選択します。
 - d. **unsetEntityContext()** メソッドを右クリックし、「削除」を選択します。
4. ここまでの作業を保管します (Ctrl + S)。

新規ファインダーの追加:

ファインダー処理の概要: ファインダー・ヘルパー・インターフェースの使用はお勧めできません。どのような開発作業でも、照会およびメソッド宣言を定義するときには、ファインダー・ヘルパー・インターフェースではなく、EJB デプロイメント記述子エディターを使用することが必要です。

ファインダー作成に EJB 照会言語を使用することの詳細、ファインダーに対して別の方法ではなくこの方法で取り組むことの利点、そして関連するツールの詳細については、WebSphere Studio Application Developer オンライン・ヘルプを参照してください。

新規 Bean への新規ファインダーの追加: Enterprise Bean に新規ファインダーを追加する必要がある場合、次のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「**EJB モジュール (EJB Modules)**」を拡張表示します。
2. **WebSphereCommerceServerExtensionsData** プロジェクトをダブルクリックし、EJB Deployment Descriptor Editor をオープンします。
3. 「**Bean**」タブをクリックします。
4. 「Bean」ペインで、*yourNewBean* Bean を選択してから、このペインの右側でスクロールダウンして、「**WebSphere 拡張版 (WebSphere Extensions)**」を拡張表示します。
5. 「**ファインダー (Finder)**」テキスト・ボックスの隣の「**追加**」をクリックします。「ファインダー記述子の追加 (Add Finder Descriptor)」ウィンドウがオープンします。
6. 「**新規**」を選択してから、「**名前**」フィールドで、*findByXyourArg* と入力します (ここで、*yourArg* は検索するときの引き数です)。“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。
7. 「**パラメーター**」テキスト・ボックスの隣の「**追加**」をクリックしてから、以下のようになります。
 - a. 「**名前**」フィールドで、*yourArg* と入力します。
 - b. 「**タイプ**」フィールドで、適切なデータ・タイプを入力します。
 - c. 「**OK**」をクリックします。
8. 「**戻りタイプ**」フィールドに、以下のいずれかを入力し、「**次へ**」をクリックします。
 - 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。
9. 「**ファインダーのタイプ (Finder type)**」ドロップダウン・リストで、「**WhereClauseFinderDescriptor**」を選択します。
10. 「**ファインダーのステートメント (Finder statement)**」フィールドで、適切なファインダーを入力します。たとえば、*T1.MEMBERID = ?* と入力してから、「**終了**」をクリックします。

11. ここまでの作業を保管してから、EJB Deployment Descriptor Editor をクローズします。

セキュリティ上の理由で、新規 Entity Bean の FinderHelper メソッドを作成するときには、前のステップで示されているように、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。

```
T1.MEMBERID = "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ます、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。

ejbCreate メソッドの新規作成: Enterprise 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、Access 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。

Enterprise 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、新規 *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。

新規 *ejbCreate* メソッドを作成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、***yourNewBean Bean*** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. ソース・コードを修正し、個々のヌル不可 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 {
    _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 {
    _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 87 ページの『基本キー』を参照してください。

3. ホーム・インターフェースに新規 `ejbCreate` メソッドを追加しなければなりません。これにより、生成された Access Bean でメソッドを使用できるようになります。ホーム・インターフェースにメソッドを追加するには、以下のようになります。
 - a. 「概略 (Outline)」ビューで `ejbCreate(yourParameters)` メソッドを右クリックして、「Enterprise Bean」>「ホーム・インターフェースへのプロモート (Promote to Home Interface)」を選択します。

ejbPostCreate メソッドの新規作成: 次に、新規 `ejbCreate` メソッドと同じ入力パラメーターを持つ、新規 `ejbPostCreate` メソッドを作成する必要があります。この新規メソッドを作成するには、以下のようになります。

1. **yourNewBean Bean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 新規 `ejbCreate` メソッドで使用したものと同じ入力パラメーターを持つ、新規 `ejbPostCreate` メソッドを作成します。ユーザーの住居についての例を続けるには、次のコードをクラスに挿入します。

```
public void ejbPostCreate(int argUserId,
    String argHome, byte Rooms)
{
}
```

コードの変更内容を保管します。

Bean へのアクセス制御メソッドの追加: 新規 Bean をアクセス制御の下で保護する場合、`getOwner` メソッドを追加する必要があります。アクセス制御の目的で任意に使用できる別のメソッドは、`fulfills` メソッドです。必須および任意のメソッドについての詳細は、115 ページの『Enterprise Bean でのアクセス制御のインプリメント』を参照してください。新規 Bean へアクセス制御メソッドを追加するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、**yourNewBean Bean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。

2. ソース・コードに、このリソースの所有者を戻すロジックを含む、 `getOwner` メソッドを追加します。たとえば、 `UserRes Bean` の所有者を戻す場合、ユーザーのメンバー ID を戻します。

```
public java.lang.Long getOwner()  
    throws java.lang.Exception {  
    return getMemberId();  
}
```

3. この例の目的では、このリソースに対して処理を行う前にユーザーが満たす必要のある関係を指定する、 `fulfills` メソッドを追加します。この場合、この `UserRes` オブジェクトの作成者だけが処理を行えることを指定します。つまり、各ユーザーは、自分自身の `UserRes` オブジェクトに対してのみ処理を行えるわけです。この関係要件は、次のコードの断片に示されています。

```
public boolean fulfills(Long member, String relationship)  
    throws java.lang.Exception {  
    if (relationship.equalsIgnoreCase("creator"))  
    {  
        return member.equals(getMemberId());  
    }  
    return false;  
}
```







4. 作業内容を保管します。

データベース・テーブルの新規 *Enterprise Bean* へのマッピング: 新規

Enterprise 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。 *Enterprise Bean* とそれに対応するデータベース・テーブルがどちらも存在する場合、“Meet-in-the-middle” タイプのマッピングが使用されます。 *WebSphere Studio Application Developer* 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。

マッピングを作成するには、以下のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、**WebSphereCommerceServerExtensionsData** を右クリックし、「生成 (Generate)」 > 「EJB 対 RDB のマッピング (EJB to RDB Mapping)」を選択します。
「EJB 対 RDB のマッピング(EJB to RDB Mapping)」ウィンドウがオープンします。
2. 「meet-in-the-middle (Meet In The Middle)」を選択し、「次へ」をクリックします。
3. 「データベース接続」ウィンドウで、以下のようにします。
 - a. 「接続名 (Connection name)」フィールドに、`WebSphereCommerceServerExtensionsData` と入力します。
 - b. 「データベース」フィールドに、 `developmentDB` と入力します。
 - c. 「ユーザー ID」フィールドに、 `dbuser` と入力します。

- d. 「パスワード」フィールドに、 `dbpassword` と入力します。
 - e. データベースのドロップダウン・リストから、ご使用の開発データベースのデータベース・メーカー・タイプを選択します。
 -  DB2 Universal Database 8.1
 -  Oracle 9i
 - f.  「ホスト」フィールドに、データベース・サーバーの完全修飾ホスト名を入力します。たとえば、`dbserver.yourcompany.com` と入力します。
 - g.  「クラスの場所 (Class Location)」フィールドに、 `classes12.zip` ファイルの場所を入力します。たとえば `D:\oracle\ora92\jdbc\lib\classes12.zip` と入力します。
 - h. 「次へ」をクリックします。接続を確立し終わると、データベース中のテーブルのリストが表示されます。また、「データ」パースペクティブの「データベース・サーバー」ビューを確認すると、後から接続文書を確認できます。
4. ***yourNewTable*** テーブルを選択して、「次へ」をクリックします。
 5. 「名前およびタイプ別に突き合わせ (Match By Name and Type)」を選択してから、「終了」をクリックします。これで、Mapping Editor がオープンします。
 6.  いずれかの列に「NUMBER」のデータ・タイプがある場合、データ・タイプを変更する必要があります。 ***yourNewTable*** テーブルを右クリックし、「テーブル・エディターのオープン (Open Table Editor)」を選択します。テーブル・エディターで、以下のようにします。
 - a. 「列 (Column)」タブを選択します。
 - b. 変更の必要なデータ・タイプが入れられた列を選択し、列タイプを NUMBER からさらに特定のタイプに変更します。たとえば、INTEGER に変更します。
 - c. 変更内容を保管します。
 7. 「Enterprise Bean」ペインで、 ***yourNewBean*** Bean を拡張表示します。「テーブル (Tables)」ペインで、 ***yourNewTable*** テーブルを拡張表示します。
 8. 以下のようにして、 ***yourNewBean*** Bean 中のフィールドを、 ***yourNewTable*** テーブル中の列にマップします。
 - a. ***yourNewBean*** Bean を右クリックして、「名前別に突き合わせ (Match By Name)」を選択します。
 9. Map.mapxmi ファイルに加えた変更内容を保管して、ファイルをクローズします。
 10.  テキスト・エディターを使用し、テーブル定義を以下のように編集する必要があります。
 - a. テキスト・エディターで ***yourNewBean.xmi*** ファイルをオープンします。
 - b. `SQLNumeric6` が出現するすべての箇所を、`SQLNumeric3` に置き換えます。
 - c. 変更内容を保管して、ファイルをクローズします。

スキーマ名の変更: 次のステップとして、スキーマ名を修正し、Bean を他のデータベースに移植できるようにします。このようにして Bean を移植可能にする特殊値は、NULLID です。スキーマ名を変更するには、以下のようにします。

1. J2EE パースペクティブで、「J2EE 階層 (J2EE Hierarchy)」ビューに切り替えます。
2. 「データベース (Databases)」を拡張表示してから、「WebSphereCommerceServerExtensionsData」を拡張表示します。
3. スキーマ・ノード (db2user など) を右クリックし、「名前変更」を選択します。
4. 値を NULLID に設定します。

Access Bean の作成: Access Bean は、Enterprise Bean のラッパーの働きをし、他のコンポーネントと Enterprise Bean との対話を単純化します。新規 Enterprise Bean の Access Bean を作成しなければなりません。すでに作成したエンティティに基づいて、この Access Bean を生成するために、WebSphere Studio Application Developer 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、Access Bean によって使用されることになります。)

Access Bean を作成するには、以下のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示してから、WebSphereCommerceServerExtensionsData を右クリックし、「新規」>「Access Bean」を選択します。
「Access Bean の追加 (Add an Access Bean)」ウィンドウがオープンします。
2. 「コピー・ヘルパー (Copy Helper)」を選択して、「次へ」をクリックします。
3. *yourNewBean* Bean を選択して、「次へ」をクリックします。
4. コンストラクター・メソッドのドロップダウン・リストから、コンストラクター・メソッドとして `findByPrimaryKey(yourPackageName. yourNewBeanKey)` を選択します。
5. 「属性ヘルパー (Attribute Helpers)」セクションで、すべての属性を選択します。
6. 「終了」をクリックします。
7. 作業内容を保管します。

デプロイメント・コードの生成: コード生成ユーティリティは 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 サーバーの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 Enterprise 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。

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. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示してから、**WebSphereCommerceServerExtensionsData** を右クリックし、「生成 (Generate)」>「デプロイメントおよび RMIC コード (Deploy and RMIC Code)」を選択します。

「デプロイメントおよび RMIC コード (Deploy and RMIC Code)」ウィンドウがオープンします。

2. **yourNewBean** Bean を選択して、「終了」をクリックします。

「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えると、新しく生成したコードを表示できます。以下のように表示されます。

表 1.

コードのタイプ	クラス名
コンテナ・インプリメンテーション生成コード	EJSCMPyourNewBeanHomeBean.java
	EJSRemoteCMPyourNewBean.java
	EJSRemoteCMPyourNewBeanHome.java
	EJSFinderyourNewBeanBean.java
JDBC アクセス・コード	EJSJDBCPersisterCMPyourNewBeanBean.java
RMI タイおよびスタブ・コード	_EJSRemoteCMPyourNewBean_Tie.java
	_yourNewBean_Stub.java
	_EJSRemoteCMPyourNewBeanHome_Tie.java
	_yourNewBeanHome_Stub.java

Enterprise Bean をテストするテスト・クライアントの使用: WebSphere Studio Application Developer に付属のテスト・クライアントを使用して、Enterprise 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. サーバー・パースペクティブに切り替えます。
2. **WebSphereCommerceServer** サーバーをダブルクリックして、「構成」タブをクリックします。
3. 「汎用テスト・クライアントの使用可能化 (Enable universal test client)」を選択します。変更内容を保管します。
4. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」を選択します。
5. **WebSphereCommerceServerExtensionsData** を右クリックして、「サーバー上で実行 (Run on server)」を選択します。
Web ブラウザーと汎用テスト・クライアントがオープンします。JNDI Explorer で、Enterprise Bean のテストを開始します。JNDI Explorer には、このサーバー上の EJB コンテナで実行される Bean の名前が表示されます。

6. 「JNDI Explorer」を右クリックします。
7. 次に、「cell」>「nodes」>「localhost」>「servers」>「server1」>「ejb」>「yourPackageStructure」の順で階層を拡張表示して、*yourNewBeanHome* インターフェースに移動しなければなりません。
8. *yourNewBeanHome* インターフェースをクリックします。「参照 (References)」ペインで、「EJB 参照 (EJB References)」>「*yourNewBean*」>「*yourNewBeanHome*」を選択します。create メソッドをクリックします。
9. create メソッドの必須入力パラメーターに対応する、右側のペインのフィールドに、適切な値を入力します。
10. 「呼び出し (Invoke)」をクリックすると、下部のペインに結果が表示されます。
11. 「オブジェクトの処理 (Work with Object)」をクリックして、リモート・インターフェースを「参照 (Reference)」ペインに追加すると、「オブジェクト参照 (Object Reference)」の下に入力した値が表示されます。新規テーブル中にレコードが新規作成されています。
12. 必要に応じて、他のメソッドもテストします。
13. テスト・クライアントをクローズして、サーバーを停止します。

コーディングの慣例: 以下の Enterprise 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。
- Enterprise Bean コードは、Enterprise Bean モジュール外のどのようなものも参照してはなりません。たとえば、Enterprise Bean コード内でコマンドや Data Bean を参照してはなりません。
- 前のセクションでは、初めて Bean を作成するときに、新規 Bean にアクセス制御を組み込む方法を記述しています。これは、`com.ibm.commerce.security.Protectable` インターフェースを追加することで、Bean を作成した後で追加することも可能です。必要であれば、Enterprise Bean のリモート・インターフェースに、`com.ibm.commerce.security.Groupable` インターフェースも追加します。Bean には、特定のメソッドもインプリメントする必要があります。これらのインターフェースを追加し、必要なメソッドを追加してから、Bean のデプロイメント・コードおよび Access Bean を再生成します。詳細は、115 ページの『Enterprise Bean でのアクセス制御のインプリメント』を参照してください。

シンプルな Data Bean の作成

Data Bean は、Enterprise 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 Data Bean は、対応する Access 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。Data Bean のほとんどのコードは、WebSphere Studio Application Developer によって自動的に生成されます。

新規 Data Bean は、WebSphereCommerceServerExtensionsLogic プロジェクトに保管されます。

シンプルな Data 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. Data Bean コードを保管するパッケージを作成します。
2. 対応する Access Bean を拡張し、該当する Data Bean インターフェースをインプリメントして、Data Bean を作成します。
3. Data Bean 用の set メソッドを作成します。
4. Data Bean 用の get メソッドを作成します。

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。

Data Bean コード用のパッケージの作成: パッケージを作成すると、Data Bean コードを保管できる場所が作成されます。

新しいパッケージを作成するには、以下のようになります。

1. WebSphere Studio Application Developer をオープンし、Java パースペクティブに切り替えます。
2. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックし、「新規」>「パッケージ」を選択します。「新規 Java パッケージ (New Java Package)」ウィザードがオープンします。
3. 「ソース・フォルダー (Source folder)」の値が、WebSphereCommerceServerExtensionsLogic/src に取り込まれています。この値をそのまま使用します。
4. 「名前」フィールドで、新規パッケージの適切な名前を入力します。たとえば com.mycompany.mydatabeans と入力します。
5. 「終了」をクリックします。

Data Bean の作成: Data 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。Data Bean は、Access Bean を拡張して、Entity Bean を (間接的に) シンプルに表現します。Data Bean は、Entity Bean から検索したり、Entity Bean の中で設定したりすることのできるプロパティをカプセル化します。

Data Bean を作成するには、以下のようになります。

1. Data Bean を保管するパッケージを右クリックし、「新規」>「クラス」を選択します。「新規 Java クラス (New Java Class)」ウィザードがオープンします。
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. 「名前」フィールドで、新規 Data Bean の名前を入力します。たとえば、UserResAccessBean を拡張する Data Bean を作成するには、UserResDataBean と入力します。

4. 「修飾子 (Modifiers)」リストで、 **public** を選択します。
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 Access 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. Data 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. 「OK」をクリックします。
7. 「終了」をクリックします。

Data Bean への必須フィールドの追加: ここでは新規 Data Bean で必須フィールドを変更する方法について説明します。以下のタイプの情報のために、2 つの必須フィールドがあります。

- コマンド・コンテキスト
- 要求プロパティ

iCommandContext フィールドを変更するには、以下のようにします。

1. 新規 Data Bean (たとえば、UserResDataBean) をダブルクリックし、ソース・コードを表示します。
2. getCommandContext メソッドを探し出します。次のように示されています。

```
public CommandContext getCommandContext() {  
    return null;  
}
```

ソース・コードに、次の部分を追加します。

```
private CommandContext iCommandContext = null;  
public com.ibm.commerce.command.CommandContext getCommandContext()  
{  
    return iCommandContext;  
}
```

3. setCommandContext メソッドを変更します。始めは次のように示されています。

```
public void setCommandContext(CommandContext arg0) {  
}
```

コードを以下のように変更します。

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

4. `getCommandContext` メソッドを変更します。始めは次のように示されています。

```
public CommandContext getCommandContext() {  
    return null;  
}
```

ソース・コードを以下のように変更します。

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

5. 作業内容を保管します。

`iRequestProperties` フィールドを変更するには、以下のようにします。

1. 新規 Data Bean (たとえば、`UserResDataBean`) をダブルクリックし、ソース・コードを表示します。

2. `getRequestProperties` メソッドを探し出します。始めは次のように示されています。

```
public TypedProperty getRequestProperties() {  
    return null;  
}
```

ソース・コードを以下のように変更します。

```
private com.ibm.commerce.datatype.TypedProperty  
requestProperties;
```

```
public TypedProperty getRequestProperties() {  
    return requestProperties;  
}
```

3. `setRequestProperties` メソッドを変更します。始めは次のように示されています。

```
public void setRequestProperties(TypedProperty arg0) throws Exception {  
}
```

ソース・コードを以下のように変更します。

```
public void setRequestProperties(com.ibm.commerce.datatype.TypedProperty  
    aParam)  
throws Exception  
{  
    // copy input TypedProperteis to local  
  
    requestProperties = aParam;  
}
```

4. `getRequestProperties` メソッドを変更します。始めは次のように示されています。

```
public TypedProperty getRequestProperties() {  
    return null;  
}
```

ソース・コードを以下のように変更します。

```

public TypedProperty getRequestProperties() {
    return requestProperties;
}

```

5. 作業内容を保管します。

対応する Access Bean の基本キーの取り込み: 対応する Access Bean の基本キーを取り込めるようにソース・コードを変更できるように留意してください。これには、Data 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)
    {
    }
}

```

Access 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、Data 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 Data Bean をアクティブにする前に、基本キーについて Data 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 Data 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、Data Bean マネージャーが Access Bean で基本キーを明示的に設定します。たとえば、Data 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、間接的な方法であることに注意してください。

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. 新規 Data 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. 「概略 (Outline)」ビューで、**populate()** メソッドを選択し、ソース・コードを表示します。
始めは次のようになっています。

```
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。

```

try {
    super.refreshCopyHelper();
} catch (javax.ejb.FinderException e) {
    throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
        "UserResDataBean", "populate");
}

```

ここまでの作業を保管します (Ctrl + S)。

新規 Data Bean が、インスタンス生成時に別の入力パラメーターを必要とする Access Bean を拡張している場合、それらの値も Data Bean の populate メソッドに設定する必要があります。

新規 Session Bean の作成

新規 Session Bean を作成するときには、WebSphereCommerceServerExtensionsData プロジェクト内で作成します。

新規 Session 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、WebSphere Commerce Server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、Session Bean は他の Entity 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,
        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・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・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・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 Enterprise 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、)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 バージョン 1.1 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 Microsystems 社の Enterprise JavaBeans バージョン 1.1 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で見なします。

オプション C では、Enterprise Bean コンテナは、「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が Entity Bean の永続データをロードし、Entity 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。コンテナは、エンティティがアクセスされるトランザクションごとに 1 つずつ、Entity Bean の複数インスタンスをアクティブにします。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せん。

各 Enterprise 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 Access Bean を介して) Enterprise 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、Enterprise 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。

Entity 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 Enterprise 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つけ、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、Entity 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、Entity Bean のコンテナ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、Enterprise 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、USERS テーブル内の REGISTRATIONUPDATE 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。

Enterprise Bean のセキュリティー確保

Enterprise 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、任意の新規 Enterprise Bean のメソッドに `WCSecurityRole` 役割を割り当てる必要があります。このタスクを実行するには、WebSphere Application Server Application Assembly Tool を使用します。このステップは、新規 Enterprise Bean のデプロイメントを実行するときに行ってください。さらに、既存の WebSphere Commerce Entity Bean を変更する場合は、変更した EJB プロジェクトの各 Entity Bean の各メソッドに、`WCSecurityRole` 役割を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、213 ページの『第 9 章 デプロイメントに関する詳細情報』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キーのことです。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 Entity Bean が組み込まれます。そのため、Entity 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、Entity 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、Entity Bean の対応する Access 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 Entity 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 Access 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: Entity 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 {
    //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(38, 0)) 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: Access 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」という新規 UserProfileAccessBean をインスタンス化します。2 行目は、Access Bean 中に基本キーを設定します。

WebSphere Studio Application Developer 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。

Access 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。

Access 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。

Entity Bean の使用

Enterprise Bean を使用するプログラムは、Enterprise 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 Enterprise Bean ごとに Access Bean が生成されます。独自の Enterprise Bean を作成する際は、WebSphere Studio Application Developer のツールを使用してこの Access Bean を生成してください。

WebSphere Commerce コマンドは、Entity Bean と直接対話する代わりに、Access Bean と対話します。図が示しているように、Access 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。Access Bean は Java Bean と同じように動作し、Enterprise 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、Access Bean は Enterprise 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。
- Access Bean がインプリメントする copyHelper オブジェクトは、コマンドが Enterprise Bean 属性を取得したり設定したりするときに Enterprise Bean への呼び出しの数を減らします。したがって、必要な Enterprise Bean への呼び出しは 1 つだけになり、これによって複数の Enterprise Bean 属性の読み書きを行えます。

以下の図は、コマンド、Access Bean、Entity Bean およびデータベースの間の対話を示しています。

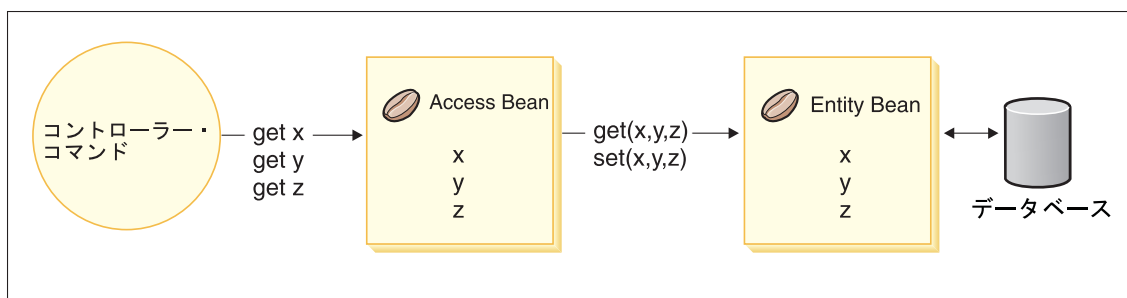


図 18.

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

一般には、前述のテーブル名の規則に従う新規テーブルを作成する場合、それらのテーブル内の列には、独自の命名規則をインプリメントできます。これは、SQL ステートメントで、必ず完全修飾列名を使用することを前提としています。ただし、既存の WebSphere Commerce テーブルとの結合を実行し、完全修飾列名を使用しない場合は、ここで説明されている列の命名規則に従う必要があります。

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。
- 生成された外部キー列の名前は、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 *fieldx* 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。

- 外部キー名の長さは 18 文字を超えてはなりません。
- 外部キー名にブランク・スペースを含めることはできません。
- 外部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。
- 外部キーの名前は *F_table* とします (*table* はテーブルの名前)。たとえば、USERS テーブルの外部キーは F_USERS1 とします。

- 外部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。

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、93 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キー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 Enterprise 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・ページにまたがるの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	    DB2	   Oracle®	 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(38,0)	BIGINT
Double	FLOAT	NUMBER(38,0)	FLOAT
文字列	CHAR()	VARCHAR2()	GRAPHIC() CCSID 13488
byte[]	CHAR() (ビット・データ用)	RAW()	CHAR() (ビット・データ用)
文字列	VARCHAR()	VARCHAR2()	VARGRAPHIC() CCSID 13488
文字列	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)
文字列	CLOB()	CLOB()	DBCLOB() CCSID 13488

注:

Oracle JDBC ドライバーが LONG データ・タイプ of 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 of 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、`Entity 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 `Session 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 Enterprise 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。

- Entity 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 Entity Bean から検索する 1 つまたは複数の Data 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。

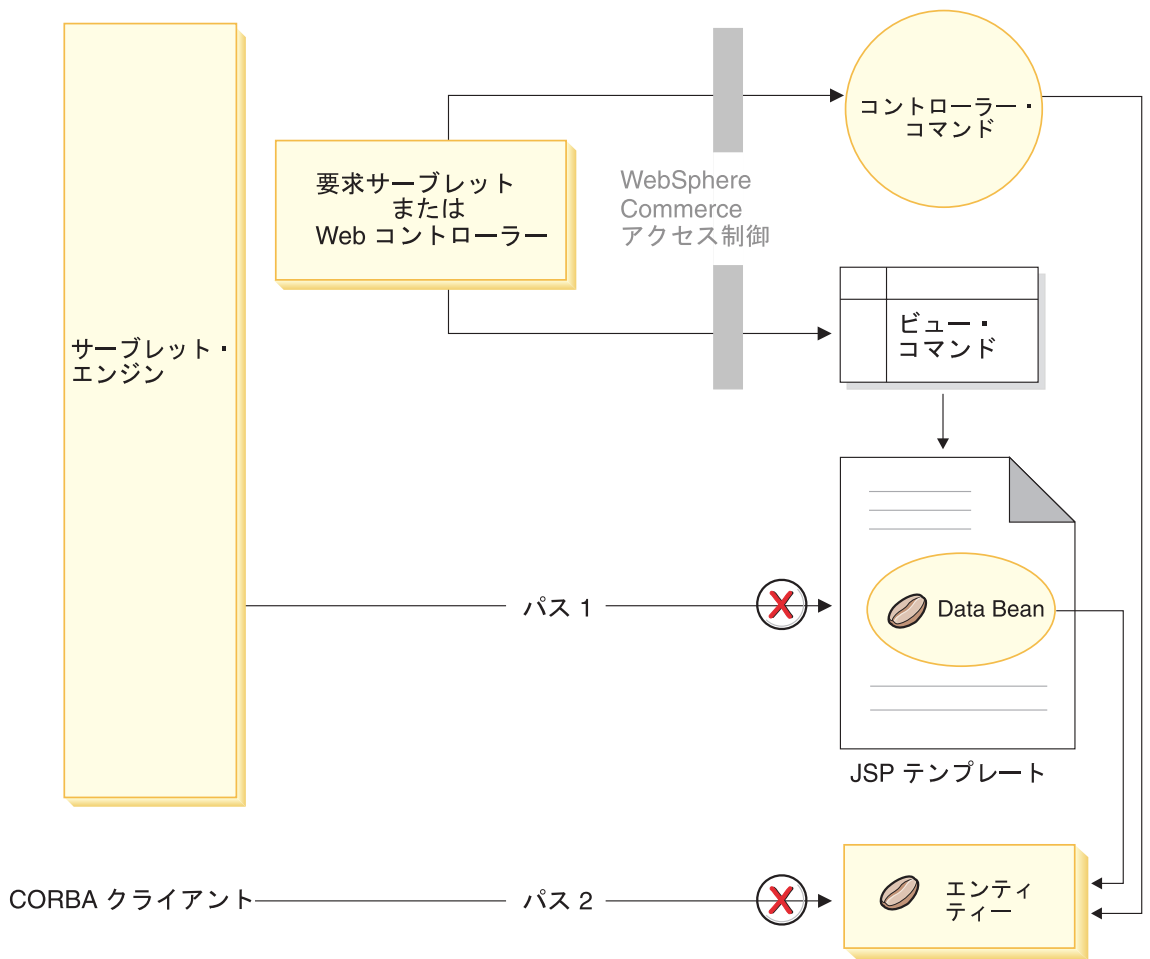


図 19.

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 Entity 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 Entity 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 は多くのセキュリティー機能を備えています。WebSphere Commerce では、このいずれかの機能を使用して、すべての

Enterprise Bean メソッドおよび JSP テンプレートが、選択した ID によってのみ呼び出されるように構成します。これらの WebSphere Commerce リソースにアクセスするには、URL 要求を、要求サーブレットに経路指定する必要があります。この要求サーブレットは、選択した ID を、Web コントローラーに渡す前に、現行スレッドで設定します。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認します。JSP テンプレートや Entity 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 セキュリティー・ガイド*」を参照してください。WebSphere Application Server 内のセキュリティについては、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。

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 Entity 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 Data Bean によって提供されます。Data Bean は、Data Bean マネージャーによってアクティブにされます。Data 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。

URL パラメーターに関するセキュリティ上の考慮事項

URL パラメーターには、要求固有の情報を渡すための便利な方法があります。悪意のあるユーザーが許可されない方法でデータベースへアクセスする機会を最小限に抑えるために、特定のコーディング方法に従う必要があります。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。パラメーター挿入は、ランタイムの入力情報を収集するために使用します。

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"
```

select を構成するのに入力ストリングを使用することを避けるのは、入力ストリングは操作されやすいためです。悪意のあるユーザーは、そのような入力ストリングを予期し

ない値に設定し、許可されていないアクセスを獲得するか、データベースに対して望ましくない操作を実行する可能性があります。

WebSphere Commerce アクセス制御ポリシーの概要

このセクションでは、WebSphere Commerce アクセス制御フレームワークの主要なコンポーネントの概要をごく簡単に説明します。これは、アクセス制御関連のプログラミング・タスクの一部を、正しい文脈で理解できるようにすることが目的で準備されています。実動システムでのアクセス制御の設定の詳細、またはアクセス制御フレームワークの詳細は、「*WebSphere Commerce セキュリティー・ガイド*」を参照してください。

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時、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 Production and Development オンライン・ヘルプ](#)を参照してください。

ActionGroup エレメントは AACTGRP 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.PayResetPMCcmd

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 Commerce セキュリティ・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	<pre><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></pre>

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.ContractUpdateCmd
10248	com.ibm.commerce.contract. commands.ContractDeleteCmd
10249	com.ibm.commerce.contract. commands.ContractCancelCmd
10250	com.ibm.commerce.contract. commands.ContractCloseCmd

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・グループの概念があります。WebSphere Commerce バージョン 5.5 ではさまざまなビジネス・モデルがサポートされており、それぞれのビジネス・モデルには、独自のアクセス制御ポリシーがあります。モデル内でポリシー群をグループ化するためには、ポリシー・グループが使用されます。ポリシーは、適切なポリシー・グループへ明示的に割り当てられるので、組織はそれらのポリシー・グループの 1 つ以上にサブスクライブできます。前のバージョンの WebSphere Commerce では、ポリシーは、そのポリシーの所有者組織の子孫によって所有されるすべてのリソースに適用されていました。

WebSphere Commerce バージョン 5.5 では、組織が 1 つ以上のポリシー・グループにサブスクライブすると、そのようなポリシー・グループに含まれるポリシーだけが、その組織のリソースに適用されます。ポリシー・グループへサブスクライブしていない組織によってリソースが所有される場合、アクセス制御ポリシー・マネージャーは、少なくとも 1 つのポリシー・グループにサブスクライブする、最も近い上位組織を見つけるまで組織階層を検索します。見つかったら、それらのポリシー・グループに属するポリシーを適用します。

以下の図を考えてみます。

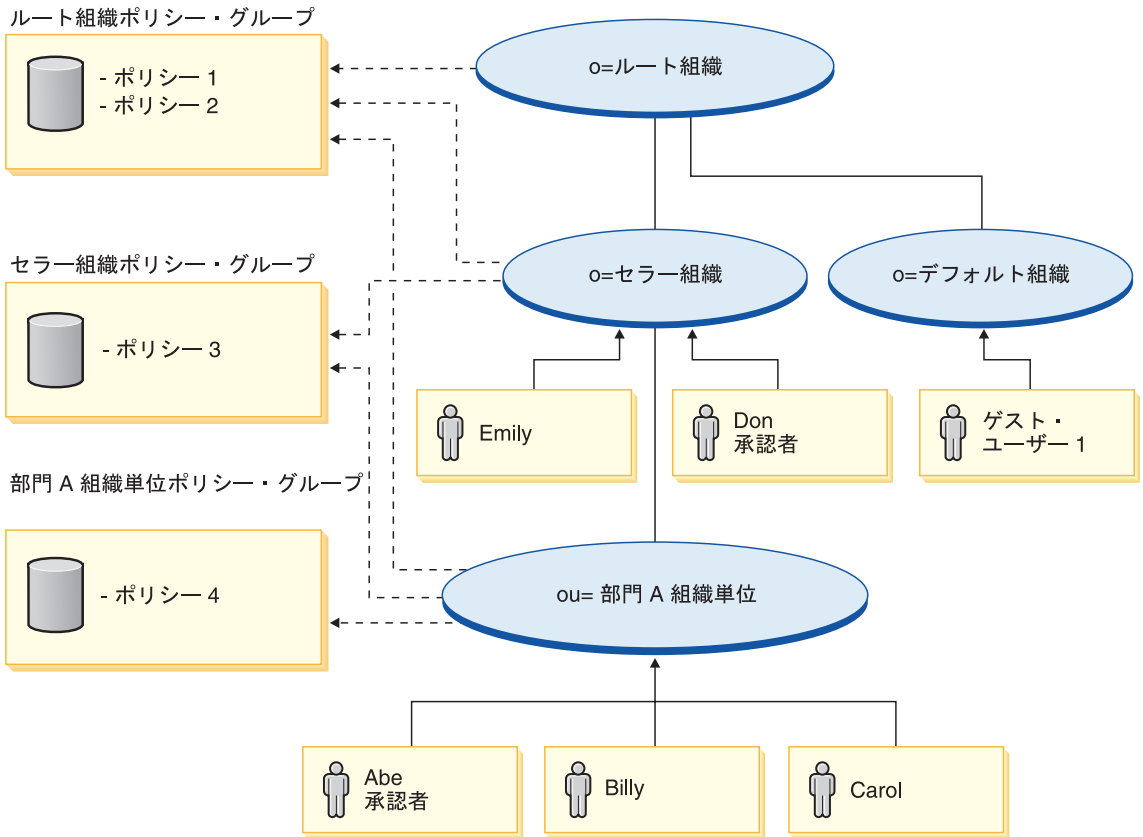


図 21.

セラー組織によって所有されているリソースの場合、セラー組織ポリシー・グループのポリシーとルート組織ポリシー・グループのポリシーが適用されます。セラー組織がそれら 2 つのポリシー・グループに明示的にサブスクライブしているため (点線矢印は、サブスクライブを示します)、これらのポリシーを適用できます。この組織には、全体で 3 つのポリシーが適用されます。この例は、リソースを所有した組織がポリシー・グループに明示的にサブスクライブする例です。さらにこの図は、組織がポリシー・グループに明示的にサブスクライブせず、アクセス制御フレームワークが階層内をさらに検索し続ける例でもあります。デフォルト組織によって所有されるリソースの場合、ルート組織まで階層をさかのぼる必要があります。ルート組織は、1 つのポリシー・グループにサブスクライブするだけなので、デフォルト組織のリソースに適用される数少ないポリシー (ポリシー 1 および 2) になります。

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せん。

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lement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` ファイルに基づき、独自の XML ファイルを作成できます。この XML ベースの情報を作成することについての詳細は、「*WebSphere Commerce セキュリティー・ガイド*」を参照してください。

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 Production* 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 ExecuteStoreAdministrators CmdResourceGroup	-2001	-8	10052	10018	ヌル

MBRGRP

MbrGrp_Id	MbrGrpName
-8	StoreAdministrators

MBRGRPCOND

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

ACACTGRP

AcActGrp_Id	GroupName
10052	ExecuteCommandActionGroup

ACRESGRP

AcResGrp_Id	GrpName
10018	StoreAdministratorsCmdResourceGroup

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。

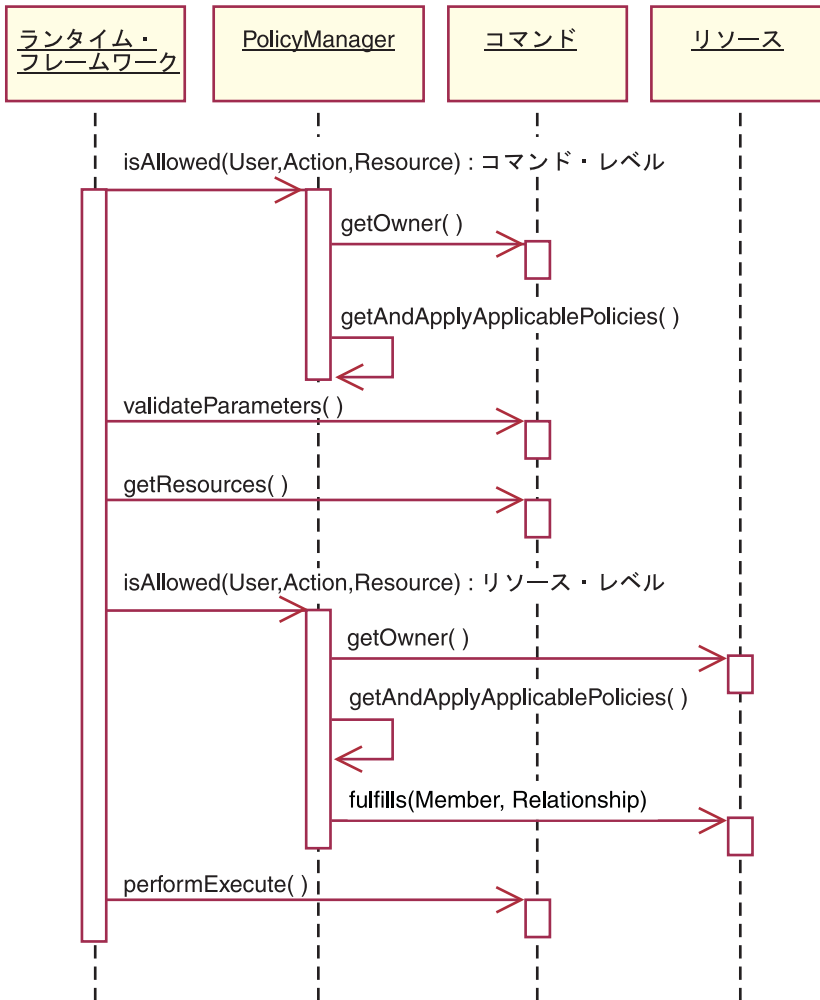


図 23.

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。リソース所有者がサブスクライブするグループのポリシーを検索することで、これを決定します。リソース所有者がポリシー・グループにサブスクライブしていない場合、リソース所有者の最も近い祖先がサブスクライブするグループのポリシーを検索します。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. `getAndApplyApplicablePolicies()`
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、適切なポリシーを検索して処理します。少なくとも 1 つの該当ポリシーがアクセス権を付与している場合、コマンド・レベルのアクセス検査は通り、ポリシー・マネージャーは次のステップに進み、リソース・レベルの許可を検査し始めます。逆に、該当ポリシーでコマンド・レベルのアクセス権を付与しているものがない場合、ポリシー・マネージャーはこの位置で戻り、アクセスを拒否します。
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 `Access Bean` です。
`Access 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、115 ページの『Enterprise 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 Enterprise 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。
8. `getAndApplyApplicablePolicies()`
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または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を満たすかどうか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 Enterprise Bean および Data 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、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 セキュリティ・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 `Enterprise Bean` および `Data Bean` です。しかし、`Enterprise Bean` と `Data 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。一般に、何を保護すべきかという問題は、新規に `Enterprise Bean` および `Data 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 `Enterprise Bean` を戻す場合は、`Enterprise 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

Enterprise 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 Enterprise Bean および Data 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ます、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 Entity Bean は保護可能なリソースですが、OrderItem Entity 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 Entity Bean は保護可能なリソースですが、Address Entity 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、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。

Enterprise 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 Enterprise 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 Enterprise 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. アクセス・グループ・ポリシーを適用する目的で、Java 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、Bean のリモート・インターフェースは、`com.ibm.commerce.grouping.Groupable` インターフェースも拡張しなければなりません。
4. Enterprise Bean クラスは、`com.ibm.commerce.base.objects.ECEntityBean` からの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を継承します。
 - `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。

`fulfills` メソッドは、リソース・グループにこのリソースを含むアクセス制御ポリシーが存在し、なおかつ関係または関係グループを指定している場合に、インプリメントする必要があります。 `getGroupingAttributeValue` メソッドは、特定の属性値に基づく、このリソースの特定インスタンスを含む暗黙的なリソース・グループを指定したアクセス制御ポリシーが存在する場合 (たとえば、 `status = 'P'` (保留) を指定したオーダーにだけ属するアクセス制御ポリシーが存在する場合) に、インプリメントする必要があります。

必要な関係が “owner” だけである場合、`fulfills` メソッドをオーバーライドする必要はないことに注意してください。この場合、ポリシー・マネージャーは、`getOwner()` メソッドの結果を利用します。

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。これらのインプリメンテーションは、`ECEntityBean` クラスからのものです。

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

以下に、`OrderBean Bean` で使用されるインプリメンテーションに基づく、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。

- `getOwner` メソッドの場合、示されているメソッドのロジックは、以下のとおりです。

```
*****
com.ibm.commerce.common.objects.StoreEntityAccessBean storeEntAB =
    new com.ibm.commerce.common.objects.StoreEntityAccessBean();
```

- ```

 storeEntAB.setInitKey_storeEntityId(getStoreEntityId().toString());
 return storeEntAB.getMemberIdInEJBType();

```
- `fulfills` メソッドの場合、示されているメソッドのロジックは、以下のとおりです。

```

 if ("creator".equalsIgnoreCase(relationship))
 {
 return member.equals(bean.getMemberId());
 }
 else if ("BuyingOrganizationalEntity".equalsIgnoreCase(relationship))
 {
 return (member.equals(bean.getOrganizationId()));
 }
 else if ("sameOrganizationalEntityAsCreator".
 equalsIgnoreCase(relationship))
 {
 com.ibm.commerce.user.objects.UserAccessBean creator =
 new com.ibm.commerce.user.objects.UserAccessBean();
 creator.setInitKey_MemberId(bean.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;

```
  - `getGroupingAttributeValue` メソッドの場合、示されているメソッドのロジックは、以下のとおりです。

```

 if (attributeName.equalsIgnoreCase("Status"))
 return getStatus();
 return null;

```

## 5. Enterprise Bean の Access Bean および生成コードを作成 (または再作成) します。

他の WebSphere Commerce public Entity Bean を調べて、`getOwner`、`fulfills`、および `getGroupingAttributeValue` メソッドのインプリメント方法を理解する場合、それらのメソッドは、Bean のアクセス・ヘルパー・クラスにインプリメントされることに気付かれるでしょう。メソッドは、Bean クラスの中に直接ではなく、アクセス・ヘルパー・クラスにインプリメントされるので、メソッド・シグニチャーは若干異なります。オブジェクト自体の別の入力パラメーターをアクセス・ヘルパーに渡すメソッドの場合に、特に当てはまります。

新規 Bean を作成するときに、それらのメソッドを Bean クラスへ直接にインプリメントする必要があります。さらに、WebSphere Commerce public Entity Bean のアクセス・ヘルパー・クラスで、それらのメソッドを変更することがないようにします。

## Data Bean でのアクセス制御のインプリメント

Data 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。Data Bean が直接保護される場合は、その特定の Data Bean に適用されるアクセス制御ポリシーが存在します。Data 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 Data Bean に保護を代行させています。

Data Bean を保護するかどうかを決定する際には、以下の点を考慮します。

1. その情報は、保護を必要とする Data Bean 情報内に存在するものでしょうか? たとえば、それは個人的な性質のものでしょうか? そうでない場合、アクセス制御による直接の保護は必要ありません。個人的なものであれば、保護の設定を続けてください。
2. Data Bean は、ビューによってインスタンス化されます。ビューは、コマンド・コンテキストからの情報 (または、事前決定された他の情報) を使用して、Data Bean をインスタンス化しますか? そうする場合で、アクセス制御がアクセスの許可をすでに決定している場合、この Data Bean を直接に保護することは必要ありません。そうしない場合には、保護の設定を続けてください。
3. ビューは、何らかの入力パラメーターからの情報を使用して、Data Bean をインスタンス化しますか? この場合、アクセス制御がこの情報へのアクセス許可を決定したかどうか分からないため、新規 Data Bean を保護する必要があります。

アクセス制御ポリシーに直接保護される新規の Data Bean を作成する場合は、Data 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。

Data Bean が `Protectable` インターフェースをインプリメントする際は、Data Bean マネージャーが `isAllowed` メソッドを呼び出して、既存のアクセス制御ポリシーに基づいて、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);
```

ここで `protectable_databean` は、保護する Data Bean です。

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 `Data Bean` そのものを戻さなければなりません (つまり、`Data 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 `Data Bean` と間接的に保護されるべき `Data 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。`Data 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。`Data Bean` の存在が別の `Data Bean` の存在に依存する場合は、他の `Data Bean` に保護を代行させるべきです。

直接保護される `Data Bean` の例としては、`Order Data Bean` があります。間接的に保護される `Data Bean` の例としては、`OrderItem Data 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 `Data Bean` を作成する場合は、`Data 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 Data 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。

`Data 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が、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・アクセス制御ポリシーの場合、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.command.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。

アクションは、同じリソース・クラスの異なるインスタンスに対して、コマンドが読み取り操作と書き込み操作の両方を実行する場合にのみ、指定する必要があります。たとえば、`OrderCopy` コマンドでは、ソース・オーダーから読み取って、宛先オーダーに書き込むことができます。この場合、2つのアクションの間に差を設ける必要があります。このことは、ソース・オーダーに“-Read”アクションを指定し、宛先オーダーに“-Write”アクションを指定して実現します。アクセス制御フレームワークがこのようなアクションを検出すると、適用可能なポリシーを検索する前に、自動的にコマンドのインターフェース名を先頭につけます。この場合、最終的にポリシーで使用されるアクションは、“`com.ibm.commerce.order.commands.OrderCopyCmd-Read`” および、“`com.ibm.commerce.order.commands.OrderCopyCmd-Write`” アクションになります。

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、同じインスタンス変数を使用できます。

`getResources` メソッドをオーバーライドするかどうかを決定する際には、以下の点を考慮します。

- コマンド・コンテキストなど、事前定義されたソースの情報に基づいてリソースを派生する場合、 `getResources` メソッドをオーバーライドする必要はありません。たとえば、`WebSphere Commerce UserRegistrationUpdate` コマンドは、コマンド・コンテキストからユーザーの ID を派生します。この場合、ユーザーは、すでに自分の登録情報を操作することを許可されているので、 `getResources` メソッドをオーバーライドする必要はありません。
- コマンドで任意に新規リソースを指定している場合（そしてこのリソースが個人的な性質のものである場合）、 `getResources` メソッドをオーバーライドする必要があります。一例として、`WebSphere Commerce OrderItemUpdate` コマンドは、入力パラメーターとしてオーダー ID をとります。この場合、オーダー・リソースがインスタンス化されると、そのリソースに対してアクションを実行する権限があるかどうかについて分かりません。その場合には、 `getResources` メソッドはオーバーライドされます。

以下は、`getResources` メソッドの例です。

```
private AccessVector resources = null;

public AccessVector getResources() throws ECEException {

 if (resources == null) {
 OrderAccessBean orderAB = new OrderAccessBean();
 orderAB.setInitKey_orderId(getOrderId().toString());
 resources = 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、既存のオーダーを更新するときに、1 つ以上の `Order` オブジェクト (保護可能なもの) 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るで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。このメソッドを使用するかどうかは任意です。

### 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.AbstractECCCommand 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 AbstractECCCommand 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 ECEException
```

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 ECAApplicationException 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。その場合には、getResources メソッドは新規リソースのコンテナを戻します。たとえば、オーダーを作成する場合、ストア・リソース内で行われます。そして、新規ユーザーが組織リソース内に作成されます。

### 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、Data Bean マネージャーによって実行されます。Data 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、Data 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 Data 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 Data 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 Data Bean は、Protectable インプリメントをインターフェースした Data 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 Data Bean を使用せずに、Access Bean を直接呼び出す場合。
2. JSP テンプレートが Data 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 Data Bean を応答プロパティ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、Data 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 (Data 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。

---

## 既存の WebSphere Commerce リソースでのアクセス制御の変更

ここでは、既存の WebSphere Commerce リソースでアクセス制御を変更するときに役立つ情報があります。特に、以下のシナリオを検討します。

- アクセス制御の下ですでに保護されている既存の WebSphere Commerce Entity Bean へ新規の関係を追加する。
- アクセス制御の下でまだ保護されていない既存の WebSphere Commerce Entity Bean へアクセス制御保護を追加する。
- 既存のコントローラー・コマンドを拡張するときのアクセス制御への影響を理解する。

## 既存の WebSphere Commerce Entity Bean への新規の関係の追加

Protectable インターフェースをインプリメントする WebSphere Commerce Entity Bean は、すでにアクセス制御の下で保護されています。アクセス制御の要件は、Bean が WebSphere Commerce の創造的機能で使用されるときの方法で決まります。そのような Bean のアクセス制御に対して、別の関係を追加しなければならないことが生じるかもしれません。たとえば、カスタマイズしたコードの一部で既存の Bean を使用する場合、または既存の WebSphere Commerce public Entity Bean を変更する場合に、Bean に別の関係を追加する場合があります。

以下のリストには、すでにアクセス制御によって保護されている既存の WebSphere Commerce Entity Bean に、新しい関係を追加するためのハイレベルなステップが示されています。

1. Entity Bean の既存の fulfills メソッドを調べます。これは、Bean のアクセス・ヘルパー・クラスにあります。このクラスは変更しないでください。これは、このログ

クに 1 つ以上の新しい関係を追加する必要があるかどうか、このメソッドをオーバーライドする必要があるかどうかを判別する場合にのみ使用してください。たとえば、以下の `fulfills` メソッドは、

`com.ibm.commerce. fulfillment.objsrc.FulfillmentCenterBeanAccessHelper` クラスにあります。

```
public boolean fulfills(Object obj, Long member, String relationship)
 throws Exception {

 FulfillmentCenterBean bean = (FulfillmentCenterBean) obj;

 if ("ShippingArrangementOrganizationalEntity".
 equalsIgnoreCase(relationship))
 {
 FulfillmentJDBCHelperAccessBean ffmJDBCAB =
 new FulfillmentJDBCHelperAccessBean();
 int count = ffmJDBCAB.
 checkFulfillmentCenterByMemberIdAndFulfillmentCenterId(
 member,bean.getFulfillmentCenterId());
 if(count>0)
 return true;
 }
 return false;
}
```

2. 次のステップは、Bean クラスに新しい `fulfills` メソッドを作成することです。たとえば、`com.ibm.commerce. fulfillment.objects.FulfillmentBean.java` クラスに、新しい `fulfills` メソッドを作成することができます。メソッドの宣言は、以下のようになります。

```
public boolean fulfills(Long member, String relationship)
 throws Exception {
 // Place holder for relationship information
}
```

3. 既存の关系到別の関係を追加する場合、メソッドの最初の行は、スーパークラスから `fulfills` メソッドを呼び出すものでなければなりません。そして、そのメソッドが以下のように `false` を戻す場合、新しい関係を調べます。

```
public boolean fulfills(Long member, String relationship)
 throws Exception {
 if (super.fulfills().equals(false))
 {
 // Check if new relationship is met
 return true;
 }

 return false;
}
```

4. 以下のように、関係を元のインプリメンテーションからすべて置き換える場合、`super.fulfills` メソッドを呼び出さないようにします。

```

public boolean fulfills(Long member, String relationship)
 throws Exception {
 // Check if new relationship is met
 // If it is, then return true;

 // If the relationship is not met, return false;
}

```

5. 変更内容を保管します。Bean のデプロイされたコードおよび RMIC コードと、対応する Access Bean を再生成します。

## まだ保護されていない既存の WebSphere Commerce Entity Bean へのアクセス制御の追加

既存の WebSphere Commerce Entity Bean を使用していて、アプリケーションでアクセス制御により Bean を保護する必要がある場合、この保護を追加できます。

以下のリストには、WebSphere Commerce アクセス制御システムの下で、既存の WebSphere Commerce Entity Bean を保護するときのハイレベルなステップが示されています。

1. *BeanName.java* クラスをオープンします。これは、リモート・インターフェースです。これを変更して、`com.ibm.commerce.security.Protectable` インターフェースを拡張できるようにします。
2. アクセス・グループ・ポリシーを適用する目的で、Java 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、Bean のリモート・インターフェースは、`com.ibm.commerce.grouping.Groupable` インターフェースも拡張しなければなりません。
3. 変更内容をリモート・インターフェースに保管します。
4. *BeanNameBean.java* クラスをオープンします。ここで *BeanName* は、アクセス制御保護を追加する Entity Bean の名前です。
5. Enterprise Bean クラスは、`com.ibm.commerce.base.objects.ECEntityBean` からの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を継承します。
  - `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詳細は、115 ページの『Enterprise Bean でのアクセス制御のインプリメント』を参照してください。

6. 変更内容を保管します。Bean のデプロイされたコードおよび RMIC コードと、対応する Access Bean を再生成します。

## コントローラー・コマンド実行時のアクセス制御の影響の知識

WebSphere Commerce プログラミング・モデルに従い、既存のコントローラー・コマンドの独自のインプリメンテーションを作成できます。この場合、新しい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することにより、その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ます。

そのような拡張を実行するときには、以下の 3 つのアクセス制御関連の影響が生じる可能性があります。

1. `getResources` メソッドへの影響
2. コマンド・レベルのアクセス制御ポリシーへの影響
3. リソース・レベルのアクセス制御ポリシーへの影響

以下のセクションでは、これらの点をそれぞれ詳しく説明します。

### `getResources` メソッドへの影響

既存のコントローラー・コマンドを拡張する場合、つまり、新しいカスタマイズしたロジックと共に、既存のコマンドのロジックを実行する場合、新しいコマンドは既存のコマンドの下位に分類されます。新規インプリメンテーションの `performExecute` メソッドは、スーパークラスの `performExecute` メソッドを呼び出します。新規コマンドが保護を必要とする新規リソースにアクセスしない場合、`getResources` メソッドをオーバーライドする必要はありません。しかし、保護可能な新規リソースにアクセスする場合、新しいコマンドは、自分の `getResources` ロジックをインプリメントする前に、自分の `getResources` メソッドをインプリメントし、このメソッドの中でスーパークラスから `getResources` メソッドを呼び出す必要があります。

スーパークラスから `getResources` メソッドを呼び出した結果は、タイプ `AccessVector` の私用インスタンス変数に保管する必要があります。その後、ローカル `getResources` メソッドの結果を、このベクトルの最後に追加しなければなりません。たとえば、新規インプリメンテーションクラスには、以下の疑似コードのようなコードが含まれます。

```
private AccessVector resources = null;

public AccessVector getResources() throws ECEException {
 // First, get the resources from the original implementation
 resources = super.getResources();

 // Now, append the new resources

 //////////////////////////////////////
 // Logic for getting new resources //
 // and appending to the vector. //
 //////////////////////////////////////

 return resources;
}
```

既存のコントローラー・コマンドのロジックを拡張せずに、ロジックを完全に置き換える場合、新しいインプリメンテーションで `super.performExecute` メソッドを呼び出さないようにします。そうすると、自分のインプリメンテーション内から `super.getResources` メソッドを呼び出さなくても良くなります。その代わりに、必要に応じて自分の `getResources` メソッドをインプリメントするだけです。

### コマンド・レベルのアクセス制御ポリシーへの影響

コントローラー・コマンドのコマンド・レベル・ポリシーは、コマンド・リソースで機能する「実行」アクションで構成されています。このコマンド・リソースは、インターフェース名で指定されます。既存のコマンドを拡張する場合、コマンドは元のインターフェースをインプリメントしたままですので、以前のコマンド・レベルのアクセス制御を維持するには、既存のコマンド・レベル・ポリシーで十分です。そのため、変更は必要ありません。


### リソース・レベルのアクセス制御ポリシーへの影響

既存のコマンドの新規インプリメンテーションを作成し、このインプリメンテーションをその既存のコマンドのインターフェースに関連付けた場合、リソース・レベルのアクセス制御ポリシーで変更は必要ありません。

そうではなく、既存のインプリメンテーションを拡張する新規インプリメンテーション・クラスを作成し、このクラスの中で `super.performExecute` メソッドを呼び出し、新規インターフェースもインプリメントする場合、リソース・レベルのアクセス制御ポリシーで変更が必要です。

後者の場合、新規コマンドが、`getResources` メソッドをインプリメントするか、基本コマンドから `getResources` メソッドの重要なインプリメンテーションを継承する場合、リソース・レベルのポリシー変更が必要です。通常は、リソース・レベルのポリシーは、ビジネス・オブジェクト・リソースで機能する、コマンド・アクションで構成されています。アクションは、コマンドのインターフェースをストリング表記したものに過ぎないため、通常は、新規コマンドを基本コマンドのアクション・グループに追加する必要があります。しかし、新規コマンドが、ヌルを戻すことにより、`getResources` メソッドの基本インプリメンテーションをオーバーライドする場合、この新規コマンドにはアクセス制御検査は実行されません。この作業は注意深く実行しないと、新規コマンドを悪意のあるユーザーに公開してしまう結果となる可能性があるため、注意してください。

リソース・レベルのアクセス制御ポリシーを変更する場合、ハイレベルなステップは以下のとおりです。

1. 以下のディレクトリーで、`defaultAccessControlPolicies.xml` ファイルを探し出します。
  -  `WCStudio_installdir¥Commerce¥xml¥policies¥xml`
2. このファイルのコピーを作成します。一例として、ファイルに `myDefaultAccessControlPolicies.xml` と名前を付けます。

3. この新規ファイルでは、新規アクションとして新規インターフェースを定義する必要があります。たとえば、以下を追加します。

```
<Action Name="yourNewInterface"
 CommandName="yourNewInterface">
</Action>
```

ここで *yourNewInterface* は、新規インターフェースの名前です。

4. 次に、ファイル全体を検索し、元のアクションが属していたすべてのアクション・グループを突き止め、新規アクションに追加する必要があります。
5. 新規コマンドを、以前に基本コマンドによって指定されていなかったリソースで機能させる場合、対応するアクセス制御ポリシーのリソース・グループも変更して、新規リソースに合わせるようにします。
6. XML ファイルに対する変更を完了したら、変更していないセクションを除去できます。 <Policies> タグの前のテキストは、残しておいてください。
7. 「*WebSphere Commerce セキュリティー・ガイド*」に載せられている指示に従い、新しいポリシー情報をデータベースにロードします。

---

## 開発用のサンプル・アクセス制御ポリシー

このセクションでは、非常に簡単なアクセス制御ポリシーをいくつか紹介します。これらは、開発環境で使用できるもので、新規リソースをすぐにテストすることができます。 *WebSphere Commerce* 実稼働環境で使用するよう設計されてはいませんので、十分なリソース保護は備わっていません。

これらのポリシーのロード方法の詳細は、「*WebSphere Commerce セキュリティー・ガイド*」を参照してください。

## 新規ビューのサンプル・アクセス制御ポリシー

新規ビューを作成する場合、開発環境で新規ビューをテストできるように、以下のアクセス制御ポリシーを使用できます (ポリシーは、使用する環境に合わせて変更し、`apclload` コマンドを使用してロードしてください)。

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>
 <Action Name="YourNewView"
 CommandName="YourNewView">
 </Action>
 <ActionGroup Name="AllSiteUsersViews"
 OwnerID="RootOrganization">
 <ActionGroupAction Name="YourNewView"/>
 </ActionGroup>
</Policies>
```

ここで *YourNewView* は、新しく作成したビューの名前です。上記のアクセス制御ポリシーは、新規ビューを、既存の *AllSiteUsersViews* アクション・グループに追加します。このポリシーにより、すべてのユーザーが新規ビューにアクセスできるようになります。

## 新規コントローラー・コマンド用のサンプル・コマンド・レベル・アクセス制御ポリシー

コントローラー・コマンドでは、アクセス制御フレームワークの要件を満たすために、アクセス制御ポリシーが必要です。新規コントローラー・コマンドを作成する場合、コマンドのインターフェースの名前がリソースとして指定されます。以下の XML 断片を新規コマンド用に変更し、*acpload* コマンドを使用してロードできます。

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>

 <Action Name="ExecuteCommand"
 CommandName="Execute">
 </Action>

 <ResourceCategory Name="com.yourcompany.yourpackage.commands.
 YourControllerCmdResourceCategory"
 ResourceBeanClass="com.yourcompany.yourpackage.commands.
 YourControllerCmd">
 <ResourceAction Name="ExecuteCommand"/>
 </ResourceCategory>

 <ResourceGroup Name="AllSiteUserCmdResourceGroup"
 OwnerID="RootOrganization">
 <ResourceGroupResource Name="com.yourcompany.yourpackage.commands.
 YourControllerCmdResourceCategory" />
 </ResourceGroup>

</Policies>
```

ここで、

- *com.yourcompany.yourpackage.commands* は、パッケージ構造を表します。
- *YourControllerCmd* は、新規コントローラー・コマンドの名前を表します。

一例として、本書のチュートリアルで作成する新規コントロール・コマンドのアクセス制御ポリシーをロードするには、以下の XML ファイルを使用します。

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>
 <Action Name="ExecuteCommand"
 CommandName="Execute">
 </Action>
```

```

<ResourceCategory Name="com.ibm.commerce.sample.commands.
 MyNewControllerCmdResourceCategory"
 ResourceBeanClass="com.ibm.commerce.sample.commands.
 MyNewControllerCmd">
 <ResourceAction Name="ExecuteCommand" />
</ResourceCategory>

<ResourceGroup Name="AllSiteUserCmdResourceGroup"
 OwnerID="RootOrganization">
 <ResourceGroupResource Name="com.ibm.commerce.sample.commands.
 MyNewControllerCmdResourceCategory" />
</ResourceGroup>

</Policies>

```

## 新規コマンドおよび Enterprise Bean のサンプル・リソース・レベル・アクセス制御ポリシー

以下の XML ファイルは、本書のチュートリアルからとられたものです。これは、新規 Entity Bean 作成時のアクセス制御要件のテンプレートとして活用できます。以下のファイルの場合、新規 Entity Bean は Bonus Bean と呼ばれ、XBONUS データベース・テーブルに対応しています。また、MyNewControllerCmd コントローラー・コマンドによって使用されます。このアクセス制御ポリシーでは、Bonus Bean オブジェクトの作成者だけが、そのオブジェクトに対して MyNewControllerCmd アクションを実行できます。

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE Policies SYSTEM "../dtd/accesscontrolpolicies.dtd">

<Policies>
 <Action Name="MyNewControllerCmd"
 CommandName="com.ibm.commerce.sample.commands.MyNewControllerCmd">
 </Action>

 <ResourceCategory Name="com.ibm.commerce.extension.objects.
 BonusResourceCategory"
 ResourceBeanClass="com.ibm.commerce.extension.objects.Bonus" >

 <ResourceAction Name="MyNewControllerCmd" />
 </ResourceCategory>

 <ActionGroup Name="MyNewControllerCmdActionGroup"
 OwnerID="RootOrganization">
 <ActionGroupAction Name="MyNewControllerCmd" />
 </ActionGroup>

 <ResourceGroup Name="BonusResourceGroup" OwnerID="RootOrganization" >
 <ResourceGroupResource Name="com.ibm.commerce.extension.objects.
 BonusResourceCategory" />
 </ResourceGroup>
 <Policy Name="AllUsersUpdateBonusResourceGroup"
 OwnerID="FashionFlowMemberId"
 UserGroup="AllUsers"

```



```

 UserGroupOwner="RootOrganization"
 ActionGroupName="MyNewControllerCmdActionGroup"
 ResourceGroupName="BonusResourceGroup"
 RelationName="creator"
 PolicyType="groupableStandard">
</Policy>

<PolicyGroup Name="ManagementAndAdministrationPolicyGroup"
 OwnerID="RootOrganization">
<!-- Define policies in this policy group -->
<PolicyGroupPolicy Name="AllUsersUpdateBonusResourceGroup"
 PolicyOwnerID="FashionFlowMemberId" />

</PolicyGroup>
</Policies>

```

ここで *FashionFlowMemberId* は、新規リソースを使用する予定のストアのメンバー ID です。

上記のアクセス制御ポリシーでは、コントローラー・コマンドのインターフェース名は、パッケージ名で完全修飾しなくても、アクションとして指定されます。アプリケーションに、同じ名前でも複数のインターフェースが存在する場合、アクセス制御ポリシーでアクションとして指定する際には、その名前にパッケージ名を付けて完全修飾する必要があります。一例として、インターフェース名がはっきりしなかった場合、上記のアクセス制御ポリシーは、以下のように変更することになります (変更された行だけが示されていて、変更は太字で示されていることに注意してください)。

```

<Action Name="com.ibm.commerce.sample.commands.MyNewControllerCmd"
CommandName="com.ibm.commerce.sample.commands.MyNewControllerCmd">
.
.
.
<ResourceAction Name="com.ibm.commerce.sample.commands.MyNewControllerCmd" />
.
.
.
<ActionGroupAction Name="com.ibm.commerce.sample.commands.MyNewControllerCmd"/>

```



---

## 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返されます。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` が返されます。この例外が返されると、`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返されます。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・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

一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際には、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 Data 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。戻されたシステム例外は、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・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・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・ファイルを決定します。

`ecServerMessagesXX_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・メッセージのキーが含まれています (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・キー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・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。

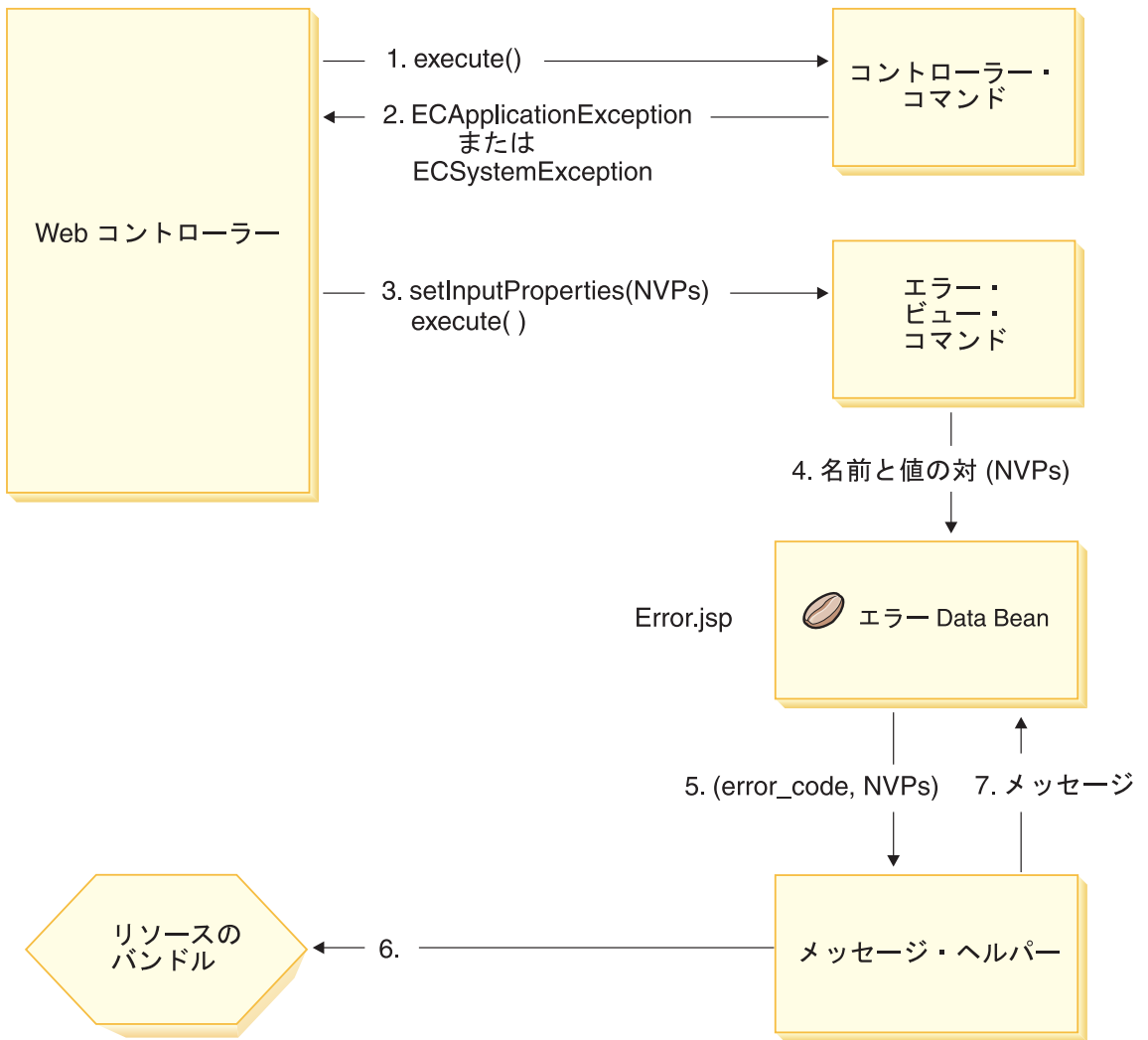


図 24.

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、 ECEException オブジェクトからプロパティのセットを構成し、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・ファイルから取得します。
7. エラー Data 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. ECApplcationException 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. ECApplcationException 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・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
<code>_ERR_FINDER_EXCEPTION</code>	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。
<code>_ERR_REMOTE_EXCEPTION</code>	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。
<code>_ERR_CREATE_EXCEPTION</code>	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。
<code>_ERR_NAMING_EXCEPTION</code>	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。
<code>_ERR_GENERIC</code>	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・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せん。

```
try {
 // your business logic
}
// catch some new type of application exception
catch(//your new exception)
{
 throw new ECApplcationException (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 `ECApplcationException` 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。

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・キーは、ログイン・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、

`WebSphereCommerceServerExtensionsLogic` プロジェクト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 ログイン・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 という名前でのテキスト・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。

▶ Studio `workspace_dir\Stores\Web Content\WEB-INF\classes\storeDir`  
ここで `storeDir` は、ストアの名前です。

メッセージが WebSphere Commerce アクセラレーター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。

▶ Studio `workspace_dir\CommerceAccelerator\Web Content\WEB-INF\classes`

メッセージが管理コンソール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。

▶ Studio `workspace_dir\SiteAdministration\Web Content\WEB-INF\classes`

▶ Business メッセージが組織管理コンソール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。

▶ Studio `workspace_dir\OrganizationAdministration\Web Content\WEB-INF\classes`

メッセージがエンタープライズ・アプリケーションのサブレットによってグローバルに使用される場合、このファイルを次のディレクトリーに入れてください。

▶ Studio `workspace_dir\WebSphereCommerceServer\properties`

上記のディレクトリーは、開発環境のコンテキスト内で指定されます。該当環境でのテストを完了したら、ターゲット WebSphere Commerce Server へのデプロイについて、213 ページの『第 9 章 デプロイメントに関する詳細情報』を参照してください。

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}".
```

## 実行フローのトレース

コマース・アプリケーションでの実行のフローをトレースする必要がある場合、WebSphere Application Server JRes 機能を使用する必要があります。これは、アプリケーションで使用できる、メッセージ・ロギングおよび診断トレース API です。

カスタマイズしたコードでこの機能を使用する方法の詳細は、WebSphere Application Server InfoCenter を参照してください。

開発環境でのコンポーネント・トレースの構成の詳細は、403 ページの『付録 A. WebSphere Commerce Studio での WebSphere Commerce コンポーネント・トレースの構成』を参照してください。

---

## 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、Data 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、Data Bean がアクティブにされたかどうかに応じて、Data Bean から戻された例外を検出するか、または各 Data 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、適切なりカバリー・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 の下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、モジュール・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 Data 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 Data 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、169 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 Data 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 Data 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

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
タスク・コマンド	MyTaskCmdImpl	com.ibm.commerce.command.TaskCommandImpl	MyTaskCmd
Data 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。

### 新規コントローラー・コマンド

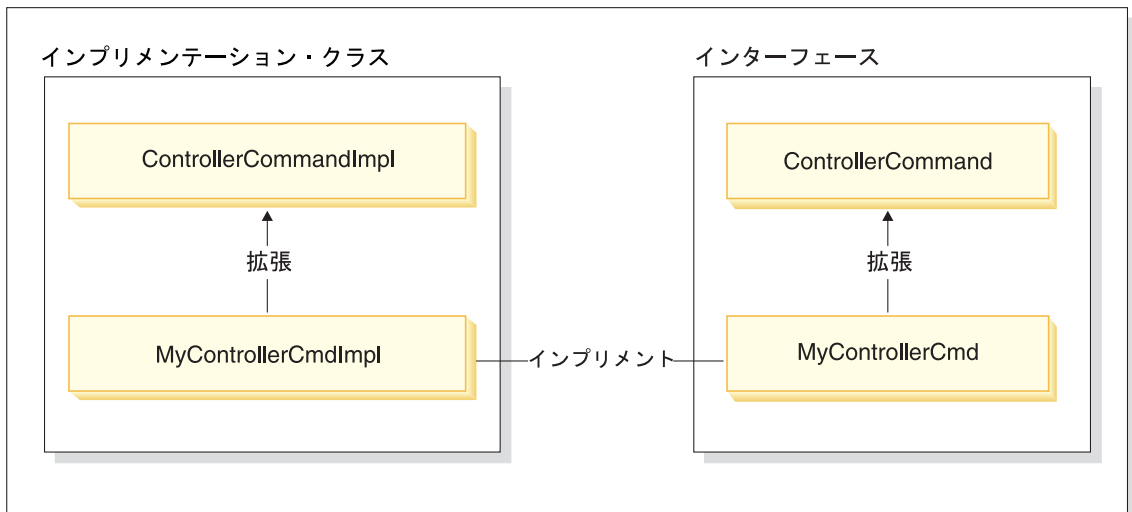


図 25.

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。

## 新規タスク・コマンド

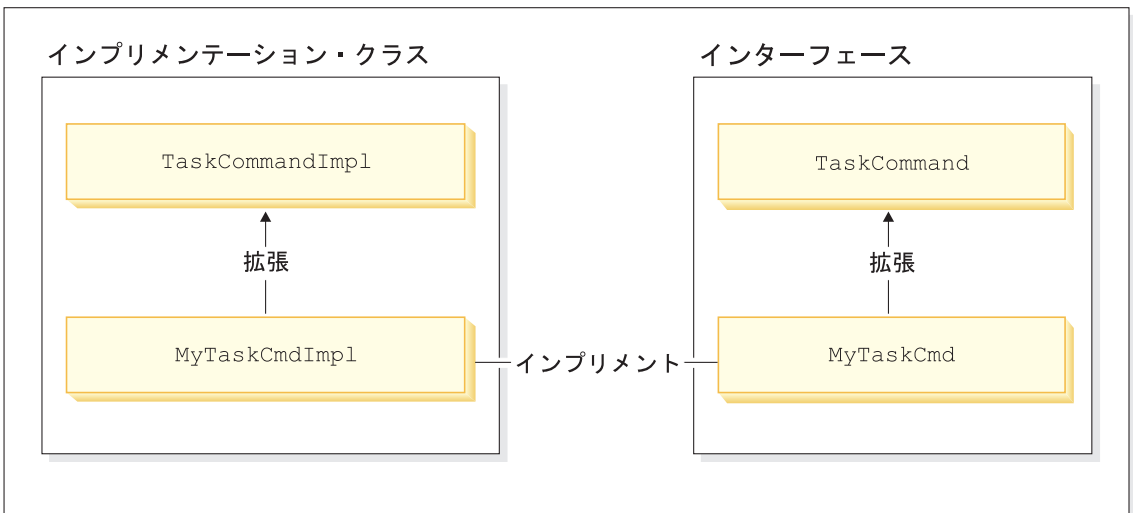


図 26.

以下の図は、新規 Data 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・クラスとの関係、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。

## 新規 Data Bean コマンド

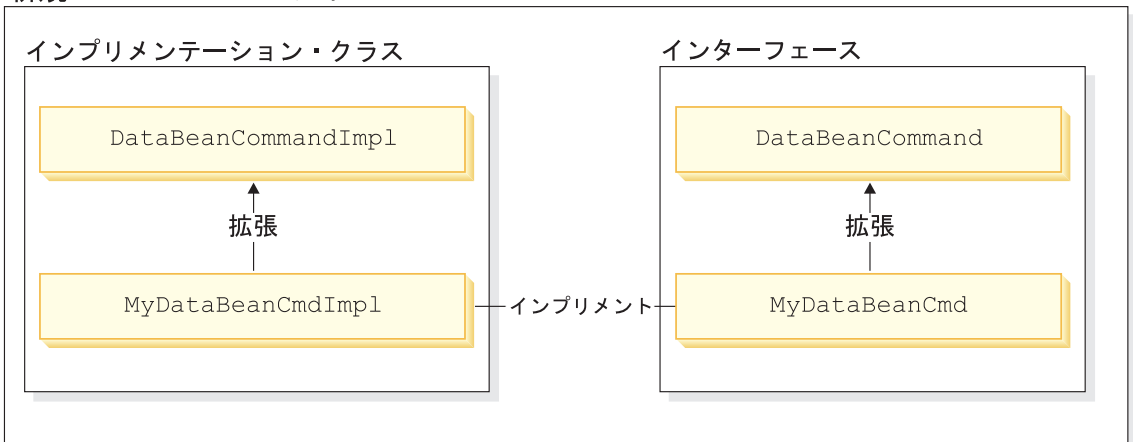


図 27.

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、カスタマイズされたコード用に事前に定義された `WebSphere Commerce` ワークスペースのプロジェクトに保管されます。

`WebSphereCommerceServerExtensionsLogic` プロジェクトと

`WebSphereCommerceServerExtensionsData` プロジェクトという 2 つの事前定義されたプロジェクトが用意されています。最初のプロジェクトはコマンドおよび `Data Bean` ロジック用で、2 番目のプロジェクトは作成する `Enterprise Bean` 用です。

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を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。

新規 `Data 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 (`WebSphereCommerceServerExtensionsLogic`) の中に入れるようにします。上記の例に従えば、`com.bigbusiness.databeans` パッケージを `WebSphereCommerceServerExtensionsLogic` プロジェクトの中を含めることができます。

新規 `Entity Bean` を作成するときには、`WebSphereCommerceServerExtensionsData` プロジェクトの中に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 `WebSphereCommerceServerExtensionsData` 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、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、以下の重要な情報をコマンド・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ル化・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。通貨はグローバル化・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()` メソッドのロジックに似ています。

### **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 Production and Development オンライン・ヘルプの『参照 (Reference)』のトピックを参照してください。

---

## **URL コマンドのコンテキスト情報に対する一時的な変更**

いくつかのコマンド・コンテキスト情報をオーバーライドし、別のストアのコンテキスト内で、または別のユーザーの代わりに、URL コマンドを実行することが可能です。URL コマンドには、コマンド・コンテキストでこの一時的な切り替えができるように、以下の URL 入力パラメーターが備えられています。

- `forStoreId`
- `forUser`
- `forUserId`

`forStoreId` URL 入力パラメーターでは、この URL 要求に使用するストア ID を指定できます。実際に、コマンド・コンテキストの `storeId` 値は指定したストアの `storeId` 値に一時的に変更されますが、この変更は URL コマンドの継続期間にだけ有効です。

forUser および forUserId URL 入力パラメーターではどちらも、現在ログインしているユーザーが異なっていたとしても、指定したユーザーのためにコマンドを実行することを指定できます。これは、サービス担当者が顧客を支援する必要がある場合に特に便利です。たとえば、顧客サービス担当者は、URL 入力パラメーターを使用して顧客のユーザー名またはユーザー ID を指定することにより、その顧客に代わって顧客の住所情報を更新することができます。ユーザー情報でのこのような変更は、指定された URL 要求の継続期間にだけ有効です。

---

## 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返します。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 を返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を構文解析して、それぞれの個別の

プロパティ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の明示的設定は、タイプ・セーフなプロパティ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、110 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、97 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(EConstants.EC_VIEWTASKNAME, "MyView");
////////////////////////////////////
// The following line is optional. The VIEWREG //
// table can specify the redirect URL. //
////////////////////////////////////

rspProp.put(EConstants.EC_REDIRECTURL, MyURL);

////////////////////////////////////
// If you are using a forward view, you can set the //
// response properties as follows: //
// TypedProperty rspProp = new TypedProperty(); //
// rspProp.put(EConstants.EC_VIEWTASKNAME, "MyView"); //
// rspProp.put(EConstants.EC_DOCPTHNAME, "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。

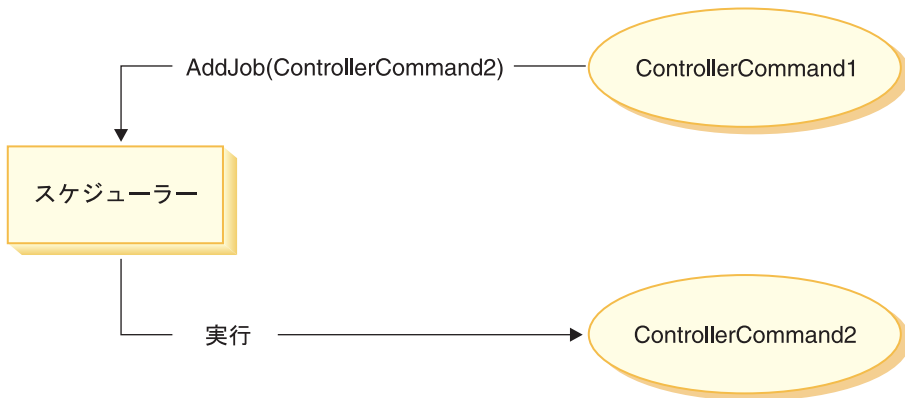


図 28.

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
- VIEWREG テーブルの PROPERTIES 列からのデフォルト・プロパティ
- URL からの入力プロパティ

これらのプロパティがマージされて 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 47 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 `HttpServletRequest` 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 original 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せん。

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(EConstants.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せん。

```
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・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。これで、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.command.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。145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) によって設定されるため、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 ECEException
{
 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` を返し、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 Data 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があります。

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・コマンドのインプリメンテーションを、ご希望の方法でタスクを実行する新しいタスク・コマンドのインプリメンテーション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、164 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 Production and Development オンライン・ヘルプの『参照 (Reference)』のトピック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。

以下のコード断片は、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。また、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ければなりません。

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せん。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 Production and Development` オンライン・ヘルプの『[参照 \(Reference\)](#)』のトピックを参照してください。

---

## Data Bean のカスタマイズ

`Data Bean` は、通常、`Access Bean` から拡張します。( `WebSphere Studio Application Developer` を使用して生成できる) `Access Bean` を使えば、`Entity Bean` 情報に簡単にアクセスできます。いずれかの `Entity 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、`Access Bean` が再生成されるとただちに `Access Bean` に反映されます。 `Data Bean` は `Access 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、Entity Bean の新しい属性を Data 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。

Entity Bean から派生した属性ではない新しい属性を Data 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 Data 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
 }
}
```

新規 Data 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
{
}
}
```

WebSphere Studio Application Developer 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せん。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。

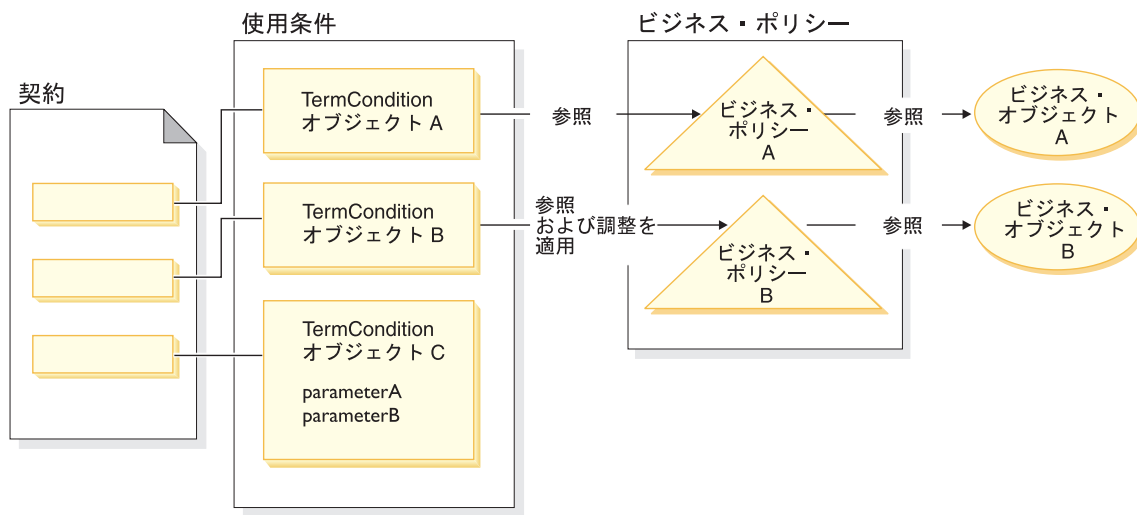


図 29.

上図では、以下の点に気をつけてください。

- 「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、図の 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、図の 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・カタログ用の商取引配備を定義するオブジェクト位置コンテナになります。

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、172 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  
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  
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() コマンドは特定条件の価格を取り込みます。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。

## 新規のビジネス・ポリシー・コマンド

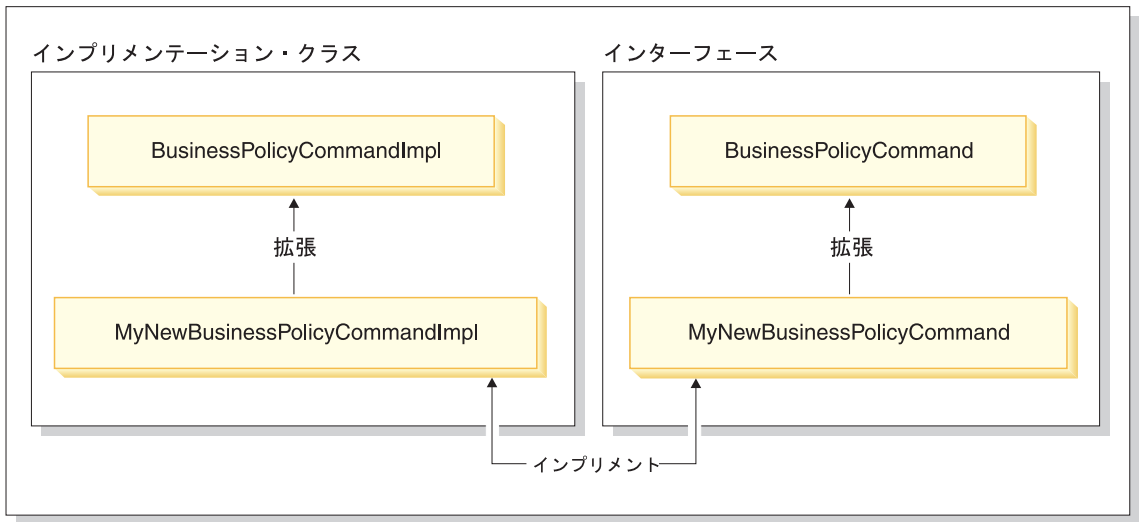


図 30.

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、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、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
STRINGFIELD1	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、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、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、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			
	TRADEPOSCN_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、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ハイ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. 新規のビジネス・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 XSD (XML スキーマ定義) に登録する。
  - c. 使用条件のための新規 CMP Enterprise 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。

WebSphere Commerce バージョン 5.5 には、新しいタイプの契約が導入されました。使用可能な契約のタイプについての詳細は、WebSphere Commerce Production オンライン・ヘルプで、『概念 (Concepts)』トピックの『ビジネス・アカウント (Business accounts)』サブトピックを参照してください。

次のセクションでは、拡張例の契約タイプとして BuyerContract を使用します。他の契約タイプにも、同様の拡張方式が使用されます。

---

## 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、以下のようなハイレベルな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ームを表します。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必要があります。

- 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、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 PLCYTPDSC (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 テーブルの外部キーです。
- 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キーの参照です。
- 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、オプション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、アカウントティング・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、XSD (XML スキーマ定義) 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・ポリシーの関係をとり込みます) 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. 新しい条件を反映するように XSD ファイルを更新します。
3. 使用条件のために新規の Enterprise 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。更新する必要があるデータベース・テーブルは、TCTYPE と TCSUBTYPE です。

以下の SQL ステートメントは、データベースに新しい条件を登録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 XSD への新しい条件の登録

新しい条件を契約中で使用できるようにするには、新規の条件を定義する新しい XSD ファイルを作成する必要があります。さらに、Package.xsd ファイルを更新して、新しい XSD ファイルを組み込む必要もあります。

新しい XSD ファイルを作成するには、以下のようにします。

1. 以下のディレクトリに移動します。

-  `WC_installdir\xml\trading\xsd`
-    `WC_installdir/xml/trading/xsd`
-  `WC_userdir/instances/instanceName/xml/trading/xsd`

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

2. このディレクトリで、新しい XSD ファイルを作成します。例 MyTC に使用される XSD を以下に示します。ファイルは、CustomizedBuyerContract.xsd です。

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ibm.com/WebSphereCommerce"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:wc="http://www.ibm.com/WebSphereCommerce"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified">

 <!-- include basic trading agreement xsd -->

 <include schemaLocation="BuyerContract.xsd" />
 <complexType name="MyTCType">
 <complexContent>
 <extension base="wc:TermConditionType"/>
 </complexContent>
 </complexType>
 <element name="MySubTC" substitutionGroup="wc:AbstractCustomizedTC">
 <complexType>
 <complexContent>
```

```






 <extension base="wc:MyTCType">
 <sequence>
 <element ref="wc:ProductSetPolicyRef"/>
 </sequence>
 <attribute name="attr1" type="normalizedString"
 use="required"/>
 <attribute name="attr2" type="int" use="required"/>
 </extension>
 </complexContent>
</complexType>
</element>
</schema>

```

3. 新しいファイルを保管します。

次に、Package.xsd を更新して BuyerContract.xsd を除去し、以下のようにして CustomizedBuyerContract.xsd を組み込みます。

1. 以下のディレクトリーに移動します。

-  `WC_installdir\xml\trading\xsd`
-    `WC_installdir/xml/trading/xsd`
-  `WC_userdir/instances/instanceName/xml/trading/xsd`

ここで *instanceName* は WebSphere Commerce インスタンスの名前です。

2. テキスト・エディターで Package.xsd ファイルをオープンします。
3. BuyerContract.xsd についてのセクションを探し、以下のように変更します。

```

<!--include schemaLocation="BuyerContract.xsd"/-->
<include schemaLocation="CustomizedBuyerContract.xsd"/>

```

## 使用条件のための新規 CMP Enterprise Bean の作成

使用条件オブジェクトのために新規 CMP Enterprise 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 Enterprise Bean を作成するときには、WebSphere Commerce Entity 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、WebSphereCommerceServerExtensionsData プロジェクトの中に 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 Entity Bean は WebSphere Commerce TermCondition Bean から継承する必要がありますので、新規の使用条件 Bean を Enablement-RelationshipManagementData プロジェクトに入れる必要があります。以下のセクションでは、WebSphere Studio Application Developer のツールを使用して新規 Enterprise Bean を作成する方法を説明します。

**新規 Enterprise Bean の作成:** 新しい使用条件のために新規 CMP Enterprise Bean を作成するには、以下のようにしてください。

1. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、「EJB モジュール (EJB Modules)」を拡張表示します。

2. **Enablement-RelationshipManagementData** モジュールを右クリックして、「新規」> (「その他」>) 「**Enterprise Bean**」を選択します。  
「Enterprise Bean の作成 (Enterprise Bean Creation)」ウィザードがオープンします。
3. 「**EJB プロジェクト (EJB Project)**」ドロップダウン・リストで、**Enablement-RelationshipManagementData** が選択されています。「次へ」をクリックします。
4. 「Enterprise Bean の作成」ウィンドウで、以下のようにします。
  - a. 「**コンテナ管理パーシスタンス (CMP) フィールドを持つ Entity Bean**」を選択します。
  - b. 「**Bean 名 (Bean name)**」フィールドに、Bean の適切な名前を入力します。この例では、MySubTC と入力します。
  - c. 「**ソース・フォルダー (Source folder)**」フィールドを、指定されているデフォルト値 (ejbModule) のままにします。
  - d. 「**デフォルト・パッケージ (Default package)**」フィールドに、`com.ibm.commerce.contract.objects` と入力します。
  - e. 「次へ」をクリックします。
5. 「Enterprise Bean の詳細 (Enterprise Bean Details)」ウィンドウで、以下のようにします。
  - a. 「**Bean スーパータイプ (Bean supertype)**」ドロップダウン・リストで、**TermCondition** を選択します。
  - b. 「追加」をクリックして、新規 CMP 属性を追加します。  
「CMP 属性の作成 (Create CMP Attribute)」ウィンドウがオープンします。このウィンドウで、以下のようにします。
    - 1) 「名前」フィールドで、新規 CMP フィールドに適切な名前を入力します。この例では、attr1 と入力します。
    - 2) 「タイプ」フィールドで、フィールドに適切なデータ・タイプを入力します。この例では、String と入力します。
    - 3) 「**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。
    - 4) 「**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**」チェック・ボックスをチェックします。
    - 5) 「**getter を読み取り専用にする (Make getter read-only)**」チェック・ボックスをクリアします。
    - 6) 「適用」をクリックします。
    - 7) 別の属性を作成します。「名前」フィールドで、attr2 と入力します。
    - 8) 「タイプ」フィールドで、フィールドに適切なデータ・タイプを入力します。この例では、Integer と入力します。

- 9) 「**getter** を読み取り専用にする (**Make getter read-only**)」チェック・ボックスをクリアします。
  - 10) 「**適用**」をクリックします。
  - 11) 「**クローズ**」をクリックしてこのウィンドウをクローズします。
6. 「**終了**」をクリックします。

**新規 Bean から TERMCOND テーブルへのフィールドのマッピング:** 次のステップは、新規 Bean から TERMCOND テーブル内の列に、フィールドをマッピングすることです。このマッピングを作成するには、以下のようにします。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 以下のフォルダーを拡張表示します。  
「**Enablement-RelationshipManagementData**」 > 「**ejbModule**」 > 「**META-INF**」。
3. **Map.mapxmi** ファイルをダブルクリックします。
4. 「Enterprise Bean」ペインで、**TermCondition** Bean を拡張表示してから、**MySubTC** Bean を拡張表示し、属性が見えるようにします。
5. 「テーブル (Tables)」ペインで、**TERMCOND** テーブルを拡張表示して、列が見えるようにします。
6. **attr1** フィールドを、MySubTC から TERMCOND テーブルの **STRINGFIELD3** 列にドラッグします。
7. **attr2** フィールドを、MySubTC から TERMCOND テーブルの **INTEGERFIELD3** 列にドラッグします。
8. 変更内容を保管します。

**新規 ejbCreate メソッドの追加:** このステップでは、以下のようにして新規 ejbCreate メソッドを MySubTC Bean に追加します。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、**MySubTCBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 以下のコードをクラスに追加して、新規 ejbCreate(Long, Element) メソッドを作成します。

```
public com.ibm.commerce.contract.objects.TermConditionKey
 ejbCreate(java.lang.Long argTradingId,
 org.w3c.dom.Element argElement)
 throws javax.ejb.CreateException,
 javax.ejb.FinderException,
 javax.naming.NamingException,
 javax.ejb.RemoveException {
 _initLinks();
 super.ejbCreate (argTradingId, argElement);
 this.attr1= null;
 this.attr2 = null;
 return null;
}
```

コードの変更内容を保管します。

3. ホーム・インターフェースに新規 `ejbCreate(Long, Element)` メソッドを追加しなければなりません。これにより、生成された `Access Bean` でメソッドを使用できるようになります。ホーム・インターフェースにメソッドを追加するには、以下のようになります。
  - a. 「概略 (Outline)」ビューで **`ejbCreate(Long, Element)`** メソッドを右クリックして、「**Enterprise Bean**」>「ホーム・インターフェースへのプロモート (**Promote to Home Interface**)」を選択します。

**新規 `ejbPostCreate` メソッドの追加:** 次に、以下のようにして、

`ejbPostCreate(Long, Element)` メソッドを新規作成し、`ejbCreate(Long, Element)` メソッドと同じパラメーターになるようにします。

1. **MySubTCBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 以下のコードをクラスに追加して、新規 `ejbPostCreate(Long, Element)` 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,
 javax.naming.NamingException,
 javax.ejb.RemoveException
 {
 parseXMLElement(argElement);
 }
```

コードの変更内容を保管します。

**`parseXMLElement` メソッドの追加:** このステップでは、以下のよう  
に `MySubTCBean` に `parseXMLElement` メソッドを作成する必要があります。

1. **MySubTCBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. メソッドを以下のように更新します。

```
public void parseXMLElement(org.w3c.dom.Element argElement) throws
 javax.ejb.CreateException,
 javax.ejb.FinderException,
 javax.naming.NamingException,
 javax.ejb.RemoveException
 {
 super.parseXMLElement(argElement);
 if (argElement == null)
 return;
 String nodeName = argElement.getNodeName();
 if (nodeName.equals("TCCopy"))
 return;
 }
```



```

this.attr1 = argElement .getAttribute("attr1").trim();
this.attr2 = new Integer (argElement.
 getAttribute("attr2").trim());
// get element "ProductSetPolicyRef" from "MySubTC"
Element ePolicyReference = null;
ePolicyReference = ContractUtil.getElementByTag(
 argElement, "ProductSetPolicyRef");

parseElementPolicyReference(ePolicyReference);

}

```

3. 作業内容を保管します。

**createNewVersion メソッドの追加:** このステップでは、以下のように MySubTCBean に新規 createNewVersion メソッドを作成する必要があります。

1. **MySubTCBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. メソッドを以下のように更新します。

```

public Long createNewVersion(Long argNewTradingId) throws
 javax.ejb.CreateException,
 javax.ejb.FinderException,
 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);
 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;
}

```

3. 作業内容を保管します。

**getXMLString メソッドの追加:** このステップでは、以下のように MySubTCBean に新規 getXMLString メソッドを作成する必要があります。

1. **MySubTCBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. メソッドを以下のようにオーバーライドします。

```

public String getXMLString() throws javax.ejb.CreateException,
 javax.ejb.FinderException,
 javax.naming.NamingException
{
 return getXMLString(false);
}

```

```

public String getXMLString(boolean tcdata) throws
 javax.ejb.CreateException,
 javax.ejb.FinderException,
 javax.naming.NamingException
{
 String xmlTC = " <MySubTC %TC_DATA% " +
 " attr1=\"\" + this.attr1 +
 "\" attr2=\"\" + this.attr2.toString() + \">\" +
 \"%TC_DESC%\" +
 \"%PARTICIPANT%\" +
 \"%XML_POLICYREFERENCE%\" +
 " </MySubTC>";
 xmlTC = ContractUtil.replace(xmlTC, "%TC_DATA%",
 getXMLStringForTCData(tcdata));
 String xmlPolicy = getXMLStringForElementPolicyReference(
 "ProductSet");
 xmlTC = ContractUtil.replace(xmlTC, "%XML_POLICYREFERENCE%",
 xmlPolicy);
 xmlTC = ContractUtil.replaceAll(xmlTC, "%POLICY_REF_TYPE%",
 "ProductSetPolicyRef");
 return xmlTC;
}

```

3. 作業内容を保管します。

**markForDelete メソッドの追加:** このステップでは、以下のように MySubTCBean に新規 markForDelete メソッドを作成する必要があります。

1. **MySubTCBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. メソッドを以下のようにオーバーライドします。

```

public void markForDelete() throws
 javax.ejb.CreateException,
 javax.ejb.FinderException,
 javax.naming.NamingException
{
 // code: remove entries from associated tables which
 // cannot be deleted though delete cascade
}

```

3. 作業内容を保管します。

**リモート・インターフェースの更新:** 以下のメソッドを以下の方法でリモート・インターフェースに追加するようにします。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。

2. **Enablement-RelationshipManagementData** プロジェクトを拡張表示します。
3. **com.ibm.commerce.contract.objects** パッケージを拡張表示します。
4. **MySubTCBean** Bean をダブルクリックします。
5. 「概略 (Outline)」ビューで **getXMLString()** メソッドを右クリックし、「Enterprise Bean」>「リモート・インターフェースへのプロモート (Promote to Remote Interface)」を選択します。
6. 「概略 (Outline)」ビューで **getXMLString(boolean tcdata)** メソッドを右クリックし、「Enterprise Bean」>「リモート・インターフェースへのプロモート (Promote to Remote Interface)」を選択します。
7. 「概略 (Outline)」ビューで **parseXMLElement(org.w3c.dom.Element argElement)** メソッドを右クリックし、「Enterprise Bean」>「リモート・インターフェースへのプロモート (Promote to Remote Interface)」を選択します。
8. 「概略 (Outline)」ビューで **createNewVersion(Long argNewTradingId)** メソッドを右クリックし、「Enterprise Bean」>「リモート・インターフェースへのプロモート (Promote to Remote Interface)」を選択します。
9. 「概略 (Outline)」ビューで **markForDelete()** メソッドを右クリックし、「Enterprise Bean」>「リモート・インターフェースへのプロモート (Promote to Remote Interface)」を選択します。
10. 変更内容を保管します。

**MySubTC の Access Bean の作成:** Access Bean を作成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示してから、**Enablement-RelationshipManagementData** を右クリックし、「新規」>「Access Bean」を選択します。  
「Access Bean の追加 (Add an Access Bean)」ウィンドウがオープンします。
2. 「コピー・ヘルパー (Copy Helper)」を選択して、「次へ」をクリックします。
3. **MySubTC** Bean を選択して、「次へ」をクリックします。
4. コンストラクター・メソッドのドロップダウン・リストから、コンストラクター・メソッドとして  
**findByPrimaryKey(com.ibm.commerce.contract.objects.MySubTCKey)** を選択します。
5. 「属性ヘルパー (Attribute Helpers)」セクションで、すべての属性を選択します。
6. 「終了」をクリックします。
7. 作業内容を保管します。

**新規ストリング・コンバーターの追加:** 以下のようにして、新規ストリング・コンバーターを追加する必要があります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「**EJB モジュール (EJB Modules)**」>「**Enablement-RelationshipManagementData**」>「**ejbModule**」>「**META-INF**」を拡張表示します。
2. **ibm-ejb-access-bean.xmi** ファイルをダブルクリックします。
3. MySubTC についてのセクションを見つけます。
4. copyHelperProperties エlementごとに、以下の属性を追加します。  
`converterClassName="com.ibm.commerce.base.objects.WCSStringConverter"`
5. 作業内容を保管します。
6. 次に、以下のようにして MySubTC の Access Bean を再生成する必要があります。
  - a. **Enablement-RelationshipManagementData** を右クリックし、「**Access Bean**」>「**Access Bean の再生成 (Regenerate Access Beans)**」を選択します。
  - b. 「**MySubTC**」を選択して、「**終了**」をクリックします。

**デプロイメント・コードの生成:** 以下のようにして、MySubTC Bean と TermCondition Bean の両方のために、デプロイメント・コードを生成する必要があります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「**EJB モジュール (EJB Modules)**」を拡張表示します。
2. **Enablement-RelationshipManagementData** を右クリックし、「**生成 (Generate)**」>「**デプロイメントおよび RMIC コード (Deploy and RMIC Code)**」を選択します。
3. 「**MySubTC**」を選択して、「**終了**」をクリックします。
4. **Enablement-RelationshipManagementData** を右クリックし、「**生成 (Generate)**」>「**デプロイメントおよび RMIC コード (Deploy and RMIC Code)**」を選択します。
5. 「**全選択**」をクリックして、「**終了**」をクリックします。

**注:** 技術的には、親 Bean (TermCondition Bean) およびすべての兄弟 Bean (名前に “TC” が含まれる、Enablement-RelationshipManagementData 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があります。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、Access Bean をそれぞれ自身のため、またそのすべての子 Bean のために再生成する必要があります。簡明にするため、前述の指示では、このプロジェクトのすべての Bean を選択します。

**validateContract タスク・コマンドのメソッドのオーバーライド:** 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。

この変更を行う方法についての詳細は、164 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。

**新規デプロイメント・コマンドの作成:** 使用条件をデプロイメントしなければならぬ場合は、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。

1. 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');
```

2. `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。

3. 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 Data 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。

-  `WC_installdir\samples\contract\Extensions.js`
-  `WCStudio_installdir\samples\contract\Extensions.js`
-      
`WC_installdir/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。

-  `WAS_installdir\installedApps\cell_name\WC_instanceName.ear\CommerceAccelerator.war\tools\contract`
-  `workspace_dir\CommerceAccelerator\Web Content\tools\contract`

- ▶ AIX ▶ Solaris ▶ Linux WAS\_installdir/installedApps/cell\_name/  
WC\_instanceName.ear/CommerceAccelerator.war/tools/contract
- ▶ 400 WAS\_userdir/installedApps/cell\_name/WC\_instanceName.ear/  
CommerceAccelerator.war/tools/contract

ここで *instanceName* は WebSphere Commerce インスタンスの名前で、*cell\_name* は WebSphere Application Server セル名です。

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 XSD 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(contract) {
 var tcModel = get("ContractMyTCModel");

 if (tcModel != null) {
 var myTC = new Object();
 myTC.attr1 = tcModel.attr1;
 myTC.attr2 = tcModel.attr2;

 myTC.ProductSetPolicyRef = new Object();
 myTC.ProductSetPolicyRef.policyName = tcModel.policyList[
 tcModel.selectedPolicyIndex].policyName;
 myTC.ProductSetPolicyRef.StoreRef = new Object();
 myTC.ProductSetPolicyRef.StoreRef.name = tcModel.policyList[
 tcModel.selectedPolicyIndex].storeIdentity;
 myTC.ProductSetPolicyRef.StoreRef.Owner = new Object();
 myTC.ProductSetPolicyRef.StoreRef.Owner = 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";
}

contract.MySubTC = 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 `WC_installdir¥samples¥contract¥ContractMyTC.jsp`
- Studio `WCstudio_installdir¥samples¥contract¥ContractMyTC.jsp`
- AIX Solaris Linux 400 `WC_installdir/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 `WAS_installdir¥installedApps¥cell_name¥WC_instanceName.ear¥CommerceAccelerator.war¥javascript¥tools¥contract`
- Studio `workspace_dir¥CommerceAccelerator¥Web Content¥tools¥contract`
- AIX Solaris Linux `WAS_installdir/installedApps/cell_name/WC_instanceName.ear/CommerceAccelerator.war/javascript/tools/contract`
- 400 `WAS_userdir/installedApps/cell_name/WC_instanceName.ear/CommerceAccelerator.war/javascript/tools/contract`

ここで `instanceName` は WebSphere Commerce インスタンスの名前で、`cell_name` は WebSphere Application Server セルの名前です。

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") %>

 <INPUT type=text name=Attr1 value="" size=10 maxlength=10>

 <%= contractsRB.get("MyTCAttr2Label") %>

 <INPUT type=text name=Attr2 value="" size=10 maxlength=10>

 <%= contractsRB.get("MyTCPolicyLabel") %>

 <SELECT NAME="PolicyList" SIZE="1">
 </SELECT>

</FORM>

```

**新規の Data Bean の作成:** このステップでは、MySubTC Access Bean から必要なデータをロードする、新規の Data 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・ファイルを更新しなければなりません。

- Windows `WAS_installdir¥installedApps¥cell_name¥  
WC_instanceName.ear¥properties¥  
com¥ibm¥commerce¥tools¥contract¥properties¥  
ContractRB_locale.properties`
- Studio `workspace_dir¥WebSphereCommerceServer¥properties¥com¥ibm¥  
commerce¥tools¥contract¥properties¥ContractRB_locale.properties`
- AIX Solaris Linux `WAS_installdir/installedApps/cell_name/  
WC_instanceName.ear/properties/  
com/ibm/commerce/tools/contract/properties/  
ContractRB_locale.properties`
- 400 `WAS_userdir/installedApps/cell_name/WC_instanceName.ear/  
properties/com/ibm/commerce/tools/contract/properties/  
ContractRB_locale.properties`

ここで *instanceName* は WebSphere Commerce インスタンスの名前で、*cell\_name* は WebSphere Application Server セルの名前です。

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。

-  `WC_installdir\xml\tools\contract\ContractNotebook.xml`
-  `WCStudio_installdir\Commerce\xml\tools\contract\ContractNotebook.xml`
-    `WC_installdir/xml/tools/contract/ContractNotebook.xml`
-  `WC_installdir/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。

```
<MySubTC attr1="abc" attr2="123">
 <ProductSetPolicyRef policyName = "Product Set 1">
 <StoreRef name = "StoreGroup1">
 <Owner>
 <OrganizationRef distinguishName = "o=Root Organization"/>
 </Owner>
 </StoreRef>
 </ProductSetPolicyRef>
</MySubTC>
```

---

## 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 Production and Development 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。

- リベートをアクティブにする。

### 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・シナリオのインプリメンテーションには、次のようなハイ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 Entity Beanを作成します。この Entity 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 XSD ファイルを更新します。
  - c. RebateTC 用の新規 Enterprise 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 Enterprise Bean の作成

既存のデータベース・スキーマにはリバ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、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 Entity 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、新規 CMP Entity Bean (XRebateCodeBean) も作成できます。この Bean の作成の詳細については、64 ページの『CMP Enterprise 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:** リバ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・ポリシーのリポート・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ポ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&policyStoreId=-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 XSD ファイルを更新します。



3. RebateTC 用の新規 Enterprise 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。更新する必要があるデータベース・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

**契約タイプ定義へのリベート条件の登録:** リベートの条件を契約中で使用できるようにするには、新規の条件を定義する新しい XSD ファイルを作成する必要があります。さらに、Package.xsd ファイルを更新して、新しい XSD ファイルを組み込む必要もあります。

新しい XSD ファイルを作成するには、以下のようになります。

1. 以下のディレクトリーに移動します。

-  `WC_installdir\xml\trading\xsd`
-    `WC_installdir/xml/trading/xsd`
-  `WC_userdir/instances/instanceName/xml/trading/xsd`

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

2. このディレクトリーで、新しい XSD ファイルを作成します。例 `RebateTC` に使用される XSD を以下に示します。ファイルは、`RebateCustomizedBuyerContract.xsd` です。

```
<?xml version="1.0"?>
<schema targetNamespace="http://www.ibm.com/WebSphereCommerce"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:wc="http://www.ibm.com/WebSphereCommerce"
 elementFormDefault="qualified"
 attributeFormDefault="unqualified">

 <!-- include basic trading agreement xsd -->

 <include schemaLocation="BuyerContract.xsd" />
 <complexType name="RebateTCType">
 <complexContent>
 <extension base="wc:TermConditionType"/>
 </complexContent>
 </complexType>
 <element name="RebateTC" substitutionGroup="wc:AbstractCustomizedTC">
 <complexType>
 <complexContent>
 <extension base="wc:RebateTCType">
 <sequence>
 <element ref="wc:RebatePolicyRef"/>
 </sequence>
 </extension>
 </complexContent>
 </complexType>
 </element>






 <element name="RebatePolicyRef" type="wc:BusinessPolicyRef" />

</schema>
```

3. 新しいファイルを保管します。

次に、`Package.xsd` を更新して `BuyerContract.xsd` を除去し、以下のようにして `RebateCustomizedBuyerContract.xsd` を組み込みます。

1. 以下のディレクトリーに移動します。

-  `WC_installdir\xml\trading\xsd`
-    `WC_installdir/xml/trading/xsd`
-  `WC_userdir/instances/instanceName/xml/trading/xsd`

- ここで `instanceName` は WebSphere Commerce インスタンスの名前です。
2. テキスト・エディターで `Package.xsd` ファイルをオープンします。
  3. `BuyerContract.xsd` についてのセクションを探し、以下のように変更します。

```
<!--include schemaLocation="BuyerContract.xsd"/-->
<include schemaLocation="RebateCustomizedBuyerContract.xsd"/>
```

**RebateTC のための新規 Enterprise Bean の作成:** 新しい RebateTC のための新規 Enterprise Bean を作成する必要があります。この新規 Bean は WebSphere Commerce TermCondition Bean から継承する必要があります。

使用条件用の新規 Enterprise 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、183 ページの『使用条件のための新規 CMP Enterprise Bean の作成』を参照してください。

表 8.

属性	値
EJB プロジェクト	Enablement-RelationshipManagementData
Bean タイプ	コンテナ管理パーシスタンス・フィールドを持つ Entity Bean
Bean 名	RebateTC
パッケージ	com.ibm.commerce.contract.objects
Bean スーパータイプ	TermCondition
Bean クラス	RebateTCBean

新規 Bean で、以下の値に使用される 3 つの CMP フィールドを作成します。

- リポート・コード ID
- 金額
- 通貨

**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、192 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 Production and Development 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なります。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 開発環境

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。

---

### 代表的な開発環境

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。一般に、本書ではこれらの製品のことを集散的に WebSphere Commerce Studio と呼んでいます。

WebSphere Commerce Studio に対する主要なコンポーネントは、以下の 4 つです。

1. WebSphere Studio Application Developer 内で使用される WebSphere Commerce ワークスペース
2. 開発データベース
3. ファイル・システム資産
4. WebSphere Studio Application Developer に対する WebSphere Commerce プラグイン

この開発環境では、カスタマイズしたコードを作成して、WebSphere テスト環境のコンテキスト内でテストできます。

開発環境でストアを作成するには、WebSphere Studio Application Developer でローカルに定義された WebSphereCommerceServer テスト・サーバー上で実行される管理コンソールを立ち上げ、そのツールを使用して、いずれかのサンプル・ストアをベースにしてストアを発行するだけです。別の方法として、独自のストアを作成することもできます。

提供されている WebSphere Commerce ワークスペースに加えて、WebSphere Commerce Studio には別のツールおよびプラグインが用意されています。以下のプラグインが備えられています。

- WebSphere Commerce (および WebSphere Commerce Payments) インスタンスの管理に役立つ構成マネージャー・プラグイン。
- WebSphere Commerce オンライン・ヘルプ・プラグインでは、WebSphere Studio Application Developer 内から WebSphere Commerce オンライン・ヘルプへアクセスで

きます。さらに、このプラグインを使用する場合、WebSphere Commerce ツール (たとえば、管理コンソール) の実行時に F1 を押すと、WebSphere Commerce コンテキストに依存したオンライン・ヘルプを立ち上げることができます。

- WebSphere Studio Application Developer 内から WebSphere Commerce API 参照情報へアクセスできるようにする WebSphere Commerce API 参照情報プラグイン。

WebSphere Commerce Enterprise Bean 変換ツールも用意されています。このツールを使用することにより、ターゲット実動データベースとは別の開発データベースを使用して、Enterprise Bean を開発することができます。たとえば、このツールを使用して、ローカルな DB2 データベースを開発に使用できますが、使用する実動データベースは iSeries プラットフォーム上や Oracle データベース上に存在していても構いません。

## WebSphere Studio Application Developer

WebSphere Commerce Studio パッケージには、IBM の中核となる開発環境である、WebSphere Studio Application Developer が含まれています。これは、最良実例、テンプレート、コード生成、およびクラス最高の広範な開発環境が備えられていて、Java2 Enterprise Edition (J2EE) および Web サービス開発を最適化および単純化するのに役立ちます。この洗練された統合開発環境 (IDE) では、Java コンポーネント、Enterprise Bean、サーブレット、JSP ファイル、HTML、XML、および Web サービスすべての統合サポートを、1 つの開発環境に収めています。

その他の魅力的な機能の中に、テスト・クライアントをすぐに生成できる、ローカル・テスト・ツールも含まれています。さらに、1 つのローカル環境でコードを端から端までテストできる、完全な WebSphere Application Server テスト環境も含まれています。

---

## iSeries での開発環境

一言で言うと、iSeries 用にカスタマイズしたコードを作成するのに、特別な開発環境は必要ありません。開発ワークステーションは、「*WebSphere Commerce Studio インストール・ガイド*」に示されているのと同じ手順でセットアップされます。使用されるローカル開発データベースは、DB2 でなければなりません。この構成を使用し、ローカル DB2 データベースおよびローカル WebSphereCommerceServer テスト・サーバーを使用することで、カスタマイズしたコードを作成してテストできます。

カスタマイズしたコードがテスト・サーバーのコンテキスト内で満足できる程度に機能することが分かったら、それを iSeries プラットフォームで稼働するターゲット WebSphere Commerce Server にデプロイする必要があります。Windows プラットフォームのデータベースと iSeries プラットフォームのデータベースの違いを明らかにするため、WebSphere Commerce Enterprise Bean 変換ツールが用意されています。このツールの詳細は、211 ページの『WebSphere Commerce Enterprise Bean 変換ツールの概要』に示されています。



---

## 実稼働環境で Oracle データベースを使用するときの開発用ローカル DB2 データベースの使用

開発者は、実稼働環境のデータベースが Oracle データベースであるとしても、開発マシン上のローカル DB2 データベースを使用できます。この場合、WebSphere Commerce Enterprise Bean 変換ツールを使用して、Bean のメタデータを、DB2 形式から Oracle 形式へ変換します。このツールの詳細は、『WebSphere Commerce Enterprise Bean 変換ツールの概要』に示されています。

---

## WebSphere Commerce Enterprise Bean 変換ツールの概要

一般に、Enterprise Bean 変換ツールが使用されるのは、以下の 2 つのケースがあります。

- ターゲット実稼働環境が iSeries プラットフォーム上で稼働している場合。
- ターゲット実動データベースは Oracle データベースだが、開発マシンはローカル DB2 データベースを使用する場合。

この WebSphere Commerce 固有のツールを使用すると、特定のタイプのデータベースを対象に開発を行い、後から別のデータベース・タイプにデプロイすることができます。このツールを使用することで、Enterprise Bean のメタデータはターゲット・データベースに適した形式および情報に変換され、デプロイメント・コードもこの新しいメタデータを使用して生成されます。このツールの使用方法についての 1 つ 1 つ詳細は、216 ページの『変換を行う EJB JAR ファイルの作成』を参照してください。

---

## 開発環境内の支払いオプション

前のバージョンの WebSphere Commerce Studio では、テスト支払いメソッドが提供されていたので、開発者は、リモート支払いプロバイダーに連絡しなくても、テスト環境のストア内で購入を完了できました。WebSphere Commerce Studio バージョン 5.5 からは、テスト環境内で WebSphere Commerce Payments コンポーネントを実行できるようになりました。

つまり、ローカル WebSphere Commerce Payments インスタンスを使用することもできますし、リモート WebSphere Commerce Payments インスタンスを使用するよう WebSphere Commerce 開発インスタンスを構成することもできます。

デフォルトでは、WebSphere Commerce Studio をインストールすると、ローカル WebSphere Commerce Payments インスタンスが作成されます。さらに、このインスタンスがサンプル・ストアを発行する時点で実行されていれば、ストアは自動的にローカル WebSphere Commerce Payments インスタンスを使用するよう構成されます。

支払いオプションの構成についての詳細は、「*WebSphere Commerce Studio* インストール・ガイド」に示されています。



---

## 第 9 章 デプロイメントに関する詳細情報

WebSphere Commerce Studio でカスタマイズ・コードを作成して WebSphere テスト環境内でテストしたなら、今度はそれを WebSphere テスト環境の外で実行しているターゲット WebSphere Commerce Server にデプロイメントする必要があります。このターゲット WebSphere Commerce Server 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。

この章では、カスタマイズしたコードを WebSphere テスト環境の外部で稼働するターゲット WebSphere Commerce Server 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。


この章は、ご使用のカスタマイズ済みアプリケーションに含まれている可能性のある、さまざまなタイプのコードのデプロイメントの方法について説明します。以下のタスクについて説明されます。

- Enterprise Bean のデプロイメント
- コマンドと Data Bean のデプロイメント
- ストア資産のデプロイメント
- ターゲット・データベースの更新

以下のセクションで、これらのタスクについて詳しく説明します。

---

### デプロイメント・ステップのユーザー許可要件

 WebSphere Commerce インストール・プロセスの準備時に作成したルート以外のユーザー ID を使用し、ターゲット WebSphere Commerce Server で、すべてのデプロイメント・ステップ (アクセス制御更新を除く) を実行する必要があります。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があります。

アクセス制御更新の実行に必要なユーザー許可についての詳細は、「*WebSphere Commerce セキュリティー・ガイド*」で、『XML の変更をデータベースにロードする』トピックを参照してください。

---

## 増分デプロイメント

この章で説明されているデプロイメントのタイプは、増分 デプロイメントです。増分デプロイメントでは、ターゲット WebSphere Commerce Server に、WebSphere Commerce エンタープライズ・アプリケーションをインストールしておく必要があります。さらに、デプロイメント・プロセスでは、既存のエンタープライズ・アプリケーションに対して、資産 (コマンド、Data Bean、Enterprise Bean など) をデプロイするだけです。

ターゲット WebSphere Commerce Server で初期エンタープライズ・アプリケーションを作成するには、WebSphere Commerce をインストールしてから、構成マネージャーを使用して WebSphere Commerce インスタンスを作成する必要があります (そして、WebSphere Commerce エンタープライズ・アプリケーションを作成します)。

---

## Enterprise Bean のデプロイメント

このセクションでは、Enterprise Bean のデプロイメントの方法について説明します。この種の Bean には、e-commerce アプリケーション用に作成した新規 Enterprise Bean と、変更を加えた WebSphere Commerce Entity Bean の両方が含まれます。どちらの場合でも、デプロイメントのステップは基本的には同じです。

WebSphere Commerce アプリケーションのデプロイメント・プロセスについて理解するための主要なエレメントとして、カスタマイズ済みの WebSphere Commerce コードに使用するパッケージ化のスキームについて理解することがあります。特に、WebSphere Commerce ワークスペース中で EJB プロジェクトを作成する必要はありません。新規 Enterprise Bean は WebSphereCommerceServerExtensionsData プロジェクトに入れられ、変更を加えた WebSphere Commerce Entity Bean のカスタマイズ済みコードは元の WebSphere Commerce EJB プロジェクトに残ります。

このデプロイメント・プロセスには 2 つの主要なステップがあります。

- EJB JAR ファイルの作成
- ターゲット WebSphere Commerce Server 中の EJB JAR ファイルの更新

### EJB JAR ファイルの作成

EJB JAR ファイルの作成には、開発のシナリオに応じて、以下の 2 つの異なる方法があります。

- 開発環境と同じタイプのデータベースを使用するターゲット WebSphere Commerce Server にデプロイする EJB JAR ファイルを作成する場合、215 ページの『変換を行わない EJB JAR ファイルの作成』に示されている指示に従ってください。
- 開発環境とは異なるタイプのデータベースを使用するターゲット WebSphere Commerce Server にデプロイする EJB JAR ファイルを作成する場合、216 ページの『変換を行う EJB JAR ファイルの作成』に示されている指示に従ってください。

## 変換を行わない EJB JAR ファイルの作成

EJB JAR ファイルを作成するには、以下のようにします。

1. WebSphere Commerce Studio (「スタート」>「プログラム」>「IBM WebSphere Commerce Studio」>「WebSphere Commerce 開発環境」) をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 以下のようにして、デプロイメントしようとしている Bean を含む EJB プロジェクトを拡張表示します。
  - Enterprise Bean を新規作成した場合は、**WebSphereCommerceServerExtensionsData** EJB プロジェクトを拡張表示します。
  - WebSphere Commerce Entity Bean に変更を加えた場合は、変更を加えた Bean を含むプロジェクトを拡張表示します。たとえば、User Bean に変更を加えた場合は、**Member-MemberManagementData** EJB プロジェクトを拡張表示します。
3. 「EJB Deployment Descriptor」をダブルクリックします。
4. 「概要 (Overview)」タブを選択しながら、ペインの下部にスクロールし、「WebSphere バインディング (WebSphere Bindings)」セクションを見つめます。
5. 「データ・ソース JNDI 名 (DataSource JNDI name)」フィールドに、ターゲットの WebSphere Commerce Server のデータ・ソース JNDI 名を入力します。値の例は以下のとおりです。

 jdbc/WebSphere Commerce DB2 DataSource demo

ここでターゲット WebSphere Commerce Server は、DB2 データベースを使用し、WebSphere Commerce インスタンス名は“demo”です。

 jdbc/WebSphere Commerce Oracle DataSource demo

ここでターゲット WebSphere Commerce Server は Oracle データベースを使用し、WebSphere Commerce インスタンス名は“demo”です。



DataSource JNDI 名の値は、“jdbc/”を、ターゲット WebSphere Commerce Server のデータ・ソース名に追加することで作成されます。ターゲット WebSphere Commerce Server で *instanceName.xml* ファイルをオープンし、ファイル内で *DatasourceName=* を探すことで、データ・ソース名を確認できます。

6. デプロイメント記述子の変更内容を保管します (Ctrl + S)。
7. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、EJB プロジェクト (WebSphereCommerceServerExtensionsData か、変更を加えた WebSphere Commerce Entity Bean を含むプロジェクト) を右クリックして、「エクスポート」を選択します。  
「エクスポート」ウィザードがオープンします。
8. 「エクスポート」ウィザードで、以下のようにします。

- a. 「EJB JAR ファイル」を選択し、「次へ」をクリックします。
  - b. 「エクスポートしたいリソース (What resources do you want to export?)」の値として、EJB プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。
  - c. 「リソースのエクスポート先にしたい場所 (Where do you want to export resources to?)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。たとえば、`C:\%ExportTemp%JarFileName.jar` (`JarFileName` は JAR ファイルの名前) と入力します。Enterprise Bean を新規作成した場合は、`yourDir%\WebSphereCommerceServerExtensionsData.jar` と入力します。既存の WebSphere Commerce public Entity Bean に変更を加えた場合は、この EJB グループの事前定義済みの JAR ファイル名を使用しなければなりません。たとえば、変更を加えた Bean が Member-MemberManagementData EJB モジュール中にある場合は、`yourDir%\Member-MemberManagementData.jar` と入力します。
  - d. 「ソース・ファイルのエクスポート (Export source files)」が選択されていないことを確認します。
  - e. 「終了」をクリックします。
9. JAR ファイルを作成し終えたら、EJB デプロイメント記述子をオープンし、ステップ 5 で加えた変更を復元して、ローカル・テスト・サーバーに必要な設定を復元します。変更内容を保管します。

## 変換を行う EJB JAR ファイルの作成

メタデータを変換して EJB JAR ファイルを作成するには、以下のようになります。

1. WebSphere Commerce Studio (「スタート」>「プログラム」>「IBM WebSphere Commerce Studio」>「WebSphere Commerce 開発環境」) をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 以下のようにして、デプロイメントしようとしている Bean を含む EJB プロジェクトを拡張表示します。
  - Enterprise Bean を新規作成した場合は、**WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
  - WebSphere Commerce Entity Bean に変更を加えた場合は、変更を加えた Bean を含むプロジェクトを拡張表示します。たとえば、User Bean に変更を加えた場合は、**Member-MemberManagementData** EJB プロジェクトを拡張表示します。
3. 「EJB Deployment Descriptor」をダブルクリックします。
4. 「概要 (Overview)」タブを選択しながら、ペインの下部にスクロールし、「WebSphere バインディング (WebSphere Bindings)」セクションを見つけます。
5. 「データ・ソース JNDI 名 (DataSource JNDI name)」フィールドに、ターゲットの WebSphere Commerce Server のデータ・ソース JNDI 名を入力します。値の

例は以下のとおりです。

▶ **DB2** jdbc/WebSphere Commerce DB2 DataSource demo

ここでターゲット WebSphere Commerce Server は、DB2 データベースを使用し、WebSphere Commerce インスタンス名は “demo” です。

▶ **Oracle** jdbc/WebSphere Commerce Oracle DataSource demo

ここでターゲット WebSphere Commerce Server は、Oracle データベースを使用し、WebSphere Commerce インスタンス名は “demo” です。



DataSource JNDI 名の値は、“jdbc/” を、ターゲット WebSphere Commerce Server のデータ・ソース名に追加することで作成されます。ターゲット WebSphere Commerce Server で *instanceName.xml* ファイルをオープンし、ファイル内で *DatasourceName=* を探すことで、データ・ソース名を確認できます。

6. デプロイメント記述子の変更内容を保管します (Ctrl + S)。
7. WebSphere Studio Application Developer をクローズします。
8. コマンド・プロンプトで、以下のディレクトリに移動します。

```
WCStudio_installDir%Commerce%bin
```

9. 以下のコマンドを入力します。

```
ejbDeploy.bat projName outputJarName mapFile workspace_dir eclipseDir
ejbDeployXmlFile WCStudio_commercedir
```

ここで、

- *projName* は変換する EJB プロジェクトの名前です。
- *outputJarName* は出力 JAR ファイルの完全修飾名です。WebSphere Commerce エンタープライズ・アプリケーションにすでに存在する JAR ファイルの名前を使用するようにします。以下は例です。

```
C:%ExportTemp%WebSphereCommerceServerExtensionsData.jar
```

- *mapFile* はマッピング・ファイルの名前です。この例としては、以下のファイルがあります。

```
WCStudio_installDir%Commerce%properties%com%ibm%commerce%
metadata%conversion%oracle.mapping
WCStudio_installDir%Commerce%properties%com%ibm%commerce%
metadata%conversion%as400.mapping
```

- *workspace\_dir* は現在の開発ワークスペース用のディレクトリです。
- *eclipseDir* は Eclipse ディレクトリへのパスです。デフォルトでは、以下のようになります。

```
WCStudio_installDir%Studio5%eclipse
```

- *ejbDeployXmlFile* は完全修飾された *ejbDeploy.xml* ファイルです。デフォルトでは、以下のようになります。

```
WCStudio_installdir¥Commerce¥xml¥ejbDeploy.xml
```

注: このコマンドを実行すると、いくつかのファイルを展開できないことを示すエラーが示される場合があります。これは問題ありません。

- `WCStudio_commercedir` は、WebSphere Commerce Studio インストール時に作成される Commerce ディレクトリーへのパスです。デフォルトでは、以下のようになります。

```
WCStudio_installdir¥Commerce
```

さらに特定することもできます。

```
C:¥WebSphere¥CommerceStudio55¥Commerce
```

以下は、このコマンドですべての値を指定する場合の使用例です。

```
ejbDeploy.bat WebSphereCommerceServerExtensionsData
WebSphereCommerceServerExtensionsData.jar
C:¥WebSphere¥CommerceStudio55¥Commerce¥properties¥com¥ibm¥
commerce¥metadata¥conversion¥oracle.mapping
C:¥WebSphere¥workspace_db2 C:¥WebSphere¥Studio55¥eclipse
C:¥WebSphere¥CommerceStudio55¥Commerce¥xml¥ejbDeploy.xml
C:¥WebSphere¥CommerceStudio55¥Commerce
```

改行がなされているのは、表記上の都合に過ぎません。


10. WebSphere Commerce Studio をオープンして EJB デプロイメント記述子をオープンし、ステップ 5 で加えた変更を復元して、ローカル・テスト・サーバーに必要な設定を復元します。変更内容を保管します。
11. 1 つのディレクトリー (たとえば、`C:¥ExportTemp`) にデプロイするすべての資産を収集している場合、新しく作成された EJB JAR ファイルをそのディレクトリーにコピーします。

## ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新

次のステップとして、新規作成した EJB JAR ファイルを、ターゲット WebSphere Commerce Server 上の該当する場所にコピーします。

ターゲット WebSphere Commerce Server で EJB JAR ファイルを更新するには、以下のようになります。

1. WebSphere Application Server 内で実行している WebSphere Commerce インスタンスを停止します。このインスタンスを停止する方法の詳細は、使用しているプラットフォームおよびデータベースの「*WebSphere Commerce* インストール・ガイド」を参照してください。
2. WebSphere Commerce インスタンス中で元の EJB JAR ファイルを見つけます。たとえば、以下のファイルを見つけます。

-  `WAS_installdir¥installedApps¥cellName¥
WC_instance_name.ear¥JarFileName.jar`



- ▶ AIX
▶ Solaris
▶ Linux
 WAS\_installdir/installedApps/cellName/  
 WC\_instance\_name.ear/JarFileName.jar
- ▶ 400
 WAS\_userdir/installedApps/WAS\_node\_name/  
 WC\_instance\_name.ear/JarFileName.jar

ここで

- instance\_name はユーザーの WebSphere Commerce インスタンスの名前です。
  - ▶ 400
 WAS\_node\_name は、 WebSphere Application Server 製品がインストールされている、 iSeries システムを表します。
  - JarFileName はカスタマイズしたコードを含む JAR ファイルの名前です。
- 元の EJB JAR ファイルのコピーを作成します。
  - 新しい EJB JAR ファイルを、開発マシンからステップ 2 の場所にコピーします。
  - EJB デプロイメント記述子に変更を加えた場合は、以下のようにします。
    - この WebSphere Application Server セルのデプロイメント・リポジトリ (META-INF ディレクトリ) を見つけます。これは、一般的に以下の形式をとります。


- ▶ Windows
 WAS\_installdir¥config¥cells¥cellName  
 ¥applications¥WC\_instance\_name.ear¥deployments¥  
 WC\_instance\_name¥EJBModuleName.jar¥META-INF
- ▶ AIX
▶ Solaris
▶ Linux
 WAS\_installdir/config/  
 cells/cellName/applications/WC\_instance\_name.ear/  
 deployments/WC\_instance\_name/EJBModuleName.jar/META-INF
- ▶ 400
 WAS\_userdir/config/cells/WAS\_node\_name/applications/  
 WC\_instance\_name.ear/deployments/WC\_instance\_name/  
 EJBModuleName.jar/META-INF

ここで


- instance\_name はユーザーの WebSphere Commerce インスタンスの名前です。
- ▶ 400
 WAS\_node\_name は、 WebSphere Application Server 製品がインストールされている、 iSeries システムを表します。
- EJBModuleName は変更された EJB モジュールの名前です。

以下に示すのは、プラットフォーム別の META-INF ディレクトリの例です。

- ▶ Windows
 WAS\_installdir¥config¥cells¥myCell¥applications¥  
 WC\_demo.ear¥deployments¥WC\_demo¥Member-MemberManagementData.jar¥  
 META-INF

-    `WAS_installdir/config/cells/myCell/applications/WC_demo.ear/deployments/WC_demo/Member-MemberManagementData.jar/META-INF`
-  `WAS_userdir/config/cells/myNode/applications/WC_demo.ear/deployments/WC_demo/Member-MemberManagementData.jar/META-INF`

ここで

- myCell は WebSphere Application Server セルの名前です。
  - demo は WebSphere Commerce インスタンスの名前です。
  - Member-MemberManagementData は変更された EJB モジュールの名前です。
  -  myNode は WebSphere Application Server ノードの名前です。
- 上記のディレクトリー中のすべてのファイルのバックアップを取ります。
  - ツールを使用して、新しい `JarFileName.jar` ファイルをオープンし、その内容を表示します。
  - META-INF ディレクトリーの内容を、この `JarFileName.jar` ファイルから、ステップ 5a のディレクトリー中に抽出します。
- 使用しているプラットフォームおよびデータベースの「*WebSphere Commerce* インストール・ガイド」で説明されているように、WebSphere Commerce インスタンスを再始動します。

---

## コマンドと Data Bean のデプロイメント

以下のいずれかのタスクを行う際には、カスタマイズ済みのコードを `WebSphereCommerceServerExtensionsLogic` プロジェクト中にパッケージする必要があります。

- 新規コマンドの作成
- 新規 Data Bean の作成
- 既存 WebSphere Commerce コマンドの変更
- 既存 WebSphere Commerce Data Bean の変更

コマンドまたは Data Bean の既存のクラス (または既存のクラスを含むプロジェクト) 中で直接変更を加えることはできません。

カスタマイズ済みコマンドおよび Data Bean のデプロイメントには、以下のステップが関係しています。

- JAR ファイルの作成
- ターゲット WebSphere Commerce Server 中の JAR ファイルの更新

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。

カスタマイズしたコードが、コマンド・レジストリー、新規アクセス制御ポリシー、または他のデータベース更新に対して更新を必要としている場合は、必ず 224 ページの『ターゲット・データベースの更新』を参照してください。

## JAR ファイルの作成

JAR ファイルを作成するには、以下のようになります。

1. WebSphere Commerce Studio (「スタート」>「プログラム」>「IBM WebSphere Commerce Studio」>「WebSphere Commerce 開発環境」) をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. **WebSphereCommerceServerExtensionsLogic** 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。「エクスポート」ウィザードがオープンします。
3. 「エクスポート」ウィザードで、以下のようになります。
  - a. 「JAR ファイル」を選択し、「次へ」をクリックします。
  - b. 「エクスポートするリソースの選択 (Select the resources to export?)」の下の左側のペインに、プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。
  - c. 「エクスポートするリソースの選択 (Select the resources to export?)」の下の右側のペインで、以下のリソースだけが選択されていることを確認します。
    - .classpath
    - .project
    - .serverPreference
  - d. 「生成されたクラス・ファイルおよびリソースのエクスポート (Export generated class files and resources)」が選択されていることを確認します。
  - e. 「Java ソース・ファイルおよびリソースのエクスポート (Export Java source files and resources)」を選択しないでください。
  - f. 「エクスポート先の選択 (Select the export destination)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。たとえば、`C:\%ExportTemp%\WebSphereCommerceServerExtensionsLogic.jar` と入力します。JAR ファイル名は `WebSphereCommerceServerExtensionsLogic.jar` でなければならぬことに注意してください。
  - g. 「終了」をクリックします。





JAR ファイルを正常に作成し終えたら、次のステップとして、ターゲットの WebSphere Commerce Server 上の該当する場所にファイルを転送します。

## ターゲット WebSphere Commerce Server 上の JAR ファイルの更新


次のステップとして、新規作成した JAR ファイルを、ターゲット WebSphere Commerce Server 上の該当する場所にコピーします。

JAR ファイルを更新するには、以下のようにします。

1. WebSphere Application Server 内で実行している WebSphere Commerce インスタンスを停止します。このインスタンスを停止する方法の詳細は、使用しているプラットフォームおよびデータベースの「*WebSphere Commerce インストール・ガイド*」を参照してください。
2. WebSphere Commerce インスタンス中で元の EJB JAR ファイルを見つけます。たとえば、以下のファイルを見つけます。

-  `WAS_installdir\installedApps\cellName\WC_instance_name.ear\WebSphereCommerceServerExtensionsLogic.jar`
-    `WAS_installdir/installedApps/cellName/WC_instance_name.ear/WebSphereCommerceServerExtensionsLogic.jar`
-  `WAS_userdir/installedApps/WAS_node_name/WC_instance_name.ear/WebSphereCommerceServerExtensionsLogic.jar`

ここで

- `instance_name` はユーザーの WebSphere Commerce インスタンスの名前です。
  -  `WAS_node_name` は、WebSphere Application Server 製品がインストールされている、iSeries システムを表します。
3. 元の JAR ファイルのコピーをバックアップ場所に作成します。
  4. JAR ファイルを、開発マシンからステップ 2 の場所にコピーします。
  5. 使用しているプラットフォームおよびデータベースの「*WebSphere Commerce インストール・ガイド*」で説明されているように、WebSphere Commerce インスタンスを再始動します。

---

## ストア資産のデプロイメント

ストア資産には、以下のような資産が含まれます。

- JSP テンプレート
- HTML ファイル
- イメージ・ファイル
- XML ファイル
- プロパティ・ファイルおよびリソース・バンドル

これらの資産を開発環境からターゲットの WebSphere Commerce Server にデプロイメントしなければなりません。この処理に含まれるステップは以下のとおりです。

- WebSphere Studio Application Developer からのストア資産のエクスポート
- ターゲット WebSphere Commerce Server への資産の転送

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。

## ストア資産のエクスポート

開発環境からストア資産をエクスポートするには、以下のようにします。

1. WebSphere Commerce Studio (「スタート」 > 「プログラム」 > 「IBM WebSphere Commerce Studio」 > 「WebSphere Commerce 開発環境」) をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 「ストア」フォルダーを拡張表示します。
3. 「Web コンテンツ (Web Content)」フォルダーを右クリックして、「エクスポート」を選択します。  
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようにします。
  - a. 「ファイル・システム (File system)」を選択して、「次へ」をクリックします。
  - b. デプロイメントしたいリソースをすべて選択します。つまり、すべての JSP テンプレート、HTML ファイル、イメージ、プロパティ・ファイル、およびデプロイを必要とする他のストア資産を選択します。
  - c. 「選択したファイルのディレクトリー構造の作成 (Create directory structure for selected files)」を選択します。
  - d. 「ディレクトリー」フィールドに、これらのリソースを入れる一時ディレクトリーを入力します。たとえば、C:\ExportTemp\StoreAssets と入力します。
  - e. 「終了」をクリックします。

次のステップでは、ターゲットの WebSphere Commerce Server 上の該当する場所にこれらのリソースをコピーします。

## ストア資産の転送

開発マシンからターゲットの WebSphere Commerce Server にストア資産を転送するには、以下のようにします。

1. デプロイする資産のタイプと固有の構成詳細に応じ、WebSphere Application Server 内で実行している WebSphere Commerce インスタンスを停止させなければならない場合があります。自分のデプロイメント・シナリオで再始動が必要かどうか分からない場合、インスタンスを停止するようにしてください。このインスタンスを停止する方法の詳細は、使用しているプラットフォームおよびデータベースの「WebSphere Commerce インストール・ガイド」を参照してください。
2. ターゲットのマシンで、Stores.war ディレクトリーを見つけます。以下は、このディレクトリーの例です。

-  WAS\_installdir\installedApps\cellName\WC\_instance\_name.ear\Stores.war

-    WAS\_installdir/installedApps/cellName/WC\_instance\_name.ear/Stores.war

- ▶ 400 `WAS_userdir/installedApps/WAS_node_name/  
WC_instance_name.ear/Stores.war`

ここで

- `instance_name` はユーザーの WebSphere Commerce インスタンスの名前です。
  - ▶ 400 `WAS_node_name` は、 WebSphere Application Server 製品がインストールされている、 iSeries システムを表します。
3. ストア資産のエクスポートでエクスポートしたファイルを、 `Stores.war` ディレクトリー中にコピーします。
  4. 前に WebSphere Commerce インスタンスを停止した場合、インスタンスを開始します。このインスタンスを開始する方法の詳細は、使用しているプラットフォームおよびデータベースの「*WebSphere Commerce インストール・ガイド*」を参照してください。

---

## 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またはビューの登録の更新、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。

▶ 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、IBM iSeries Access for Windows 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. 「**iSeries ナビゲーター (iSeries Navigator)**」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. 左側のペインで、iSeries システムを拡張表示してから、「**DATABASES**」を拡張表示します。「**リレーショナル・データベース (Relational Database)**」を右クリックし、ドロップダウン・リストから、「**SQL スクリプトの実行 (Run SQL Scripts)**」を選択します。

「SQL スクリプトの実行 (Run SQL Scripts)」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。「**接続 (Connection)**」選択項目の下の「**JDDBC のセットアップ (JDDBC Setup)**」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。

## アクセス制御の更新

アクセス制御情報はデータベース中に含まれていますが、これは特殊なタイプの情報なので、開発環境からターゲットの環境に必ずしもそのまま複製できるとは限りません。特に、開発環境では、実稼働（または次のレベルのテスト）環境では適切でない、非常に開放的なアクセス制御ポリシーを使用するよう決めている場合が考えられます。一例として、開発環境の範囲内で、すべてのユーザーがコマンドを実行できるように新規コマンドのポリシーを設定した場合、この設定は環境によっては適切でない可能性があります。

したがって、開発環境からターゲットの環境にアクセス制御情報をコピーする場合は、その前に新しい環境のアクセス制御要件を考慮し、それによってポリシーを調整する必要があります。

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 セキュリティー・ガイド*」を参照してください。





---

## 第 4 部 チュートリアル

次のチュートリアルは、WebSphere Commerce アプリケーション向けにカスタマイズしたコードを作成することに関連した、さまざまなタスクを紹介するよう設計されています。開発関連のステップは、WebSphere Commerce Business Edition 開発環境と WebSphere Commerce Studio, Professional Developer Edition 開発環境のいずれかで実行されます。これらの開発ステップは、Windows 2000 上で稼働する WebSphere Commerce で実行されます (開発環境に応じて、Business Edition または Professional Edition)。



---

## 第 10 章 チュートリアル: ビジネス・ロジックの新規作成

このチュートリアルでは、新規ビジネス・ロジックの作成に関係したステップを紹介するようになっています。作成される資産のタイプには、新規ビュー、新規コントローラー・コマンド、新規タスク・コマンド、新規 Data Bean、そして新規 Entity Bean があります。このチュートリアルでは、ボーナス・ポイントの収支を変更できるようにする小さなインターフェース開発のシナリオを使用します。これは、デモンストレーションのためのものであり、正式なプログラム・アプリケーションを作成するのに必要なロジックを反映しているわけではありません。しかし、このチュートリアルから、前述のタイプのコード資産それぞれを作成するときに共通する開発ステップを学習できます。

このチュートリアルは以下のサブタスクに分かれています。

1. ワークスペースの準備
2. 新規ビューの作成
3. 新規コントローラー・コマンドの作成
4. コントローラー・コマンドからビューへの情報の引き渡し
5. コントローラー・コマンド中の URL パラメーターの構文解析と妥当性検査
6. 新規タスク・コマンドの作成
7. 新規タスク・コマンドの変更
8. Enterprise Bean の新規作成
  - a. 新規テーブルの作成
  - b. CMP Enterprise Bean の新規作成
  - c. 新規 Bean のテーブルへのマッピングとスキーマの作成
  - d. 関連した Access Bean の作成
  - e. デプロイメント・コードの生成
  - f. 汎用テスト・クライアントを使った Bean のテスト
9. Bonus Bean の MyNewControllerCmd への統合
  - a. 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 XBONUS テーブルに保管するようにします。
  - b. 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。
  - c. BonusDataBean を作成し、ボーナス・ポイントを JSP テンプレートに表示しやすくするようにします。

- d. 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。
  - e. MyNewJSPTemplate.jsp ファイルに変更を加えて、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。
  - f. 統合したコードをテストします。
10. 前述のすべてのコード、アクセス制御ポリシー、JSP、イメージ、およびリソース・バンドルの WebSphere Application Server 中で実行しているターゲット WebSphere Commerce Server へのデプロイメント

以下のセクションで、これらのステップについて1つ1つ詳しく説明します。

---

## サンプル・コードの場所

このチュートリアルを始める前に、WC\_SAMPLE\_55.zip パッケージをダウンロードしてください。このパッケージには、この種のプログラミング・チュートリアルの開始点が含まれています。ご使用の開発マシンにこのファイルを保管してください。一例として、ファイルを WCStudio\_install\_dir ディレクトリーに保管できます。

このパッケージは、以下の Web サイトに、「WebSphere Commerce プログラミング・ガイドとチュートリアル」と共に置かれています。

<http://www.ibm.com/software/commerce/library/>

---

## ワークスペースの準備

このステップでは、サンプル・コードを WebSphere Commerce ワークスペースにインポートします。このサンプル・コードは、チュートリアルの開始点です。

ワークスペースを準備するため、以下のようになります。

1. WebSphere Commerce Studio バージョン 5.5 がインストール済みであることと、開発環境の構成を完了していることを確認します。また、開発環境中に FashionFlow サンプル・ストア (消費者向けモデルの一例) を基にしたストアを公開している必要もあります。ストアの公開方法についての指示は、WebSphere Commerce Production and Development オンライン・ヘルプにあります。
2. 以下のようにして、サンプル・コードをワークスペースにインポートします。
  - a. WebSphere Commerce Studio を始動します (「スタート」>「プログラム」>「IBM WebSphere Commerce Studio」>「WebSphere Commerce 開発環境」)。
  - b. J2EE パースペクティブに切り替えて (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「J2EE」) から、「J2EE ナビゲーター (J2EE Navigator)」ビューを選択します。
  - c. **WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。

- d. 「src」フォルダーを右クリックして、「インポート (Import)」を選択します。  
「インポート (Import)」ウィザードがオープンします。
  - e. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
  - f. 「ブラウズ」(「ZIP ファイル (Zip file)」フィールドの隣)をクリックして、サンプル・コードに移動します。ファイルは、  
`yourDirectory\WC_SAMPLE_55.zip`  
にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
  - g. 「全選択解除 (Deselect All)」をクリックし、ディレクトリーを拡張表示して、以下のファイルをインポートするよう選択します。
    - `com.ibm.commerce.sample.commands.MyNewControllerCmd.java`
    - `com.ibm.commerce.sample.commands.MyNewControllerCmdImpl.java`
    - `com.ibm.commerce.sample.commands.MyNewTaskCmd.java`
    - `com.ibm.commerce.sample.commands.MyNewTaskCmdImpl.java`
    - `com.ibm.commerce.sample.databeans.MyNewDataBean.java`
  - h. 「フォルダー (Folder)」フィールドでは、「WebSphereCommerceServerExtensionsLogic/src」フォルダーがすでに指定されています。この値をそのまま使用します。
  - i. 「終了」をクリックします。
3. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックし、「プロジェクトの再作成 (Rebuild project)」を選択します。
  4. 以下のようにして、チュートリアル用の JSP テンプレートを該当するディレクトリーにインポートします。
    - a. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、**Stores** プロジェクトを拡張表示します。次に、「Web コンテンツ (Web Content)」>  
「**FashionFlow\_name**」(*FashionFlow\_name* は FashionFlow サンプル・ストアを基にしたストアの名前)を拡張表示します。



ストアを公開したばかりの場合、*FashionFlow\_name* ディレクトリーは表示されない場合があります。そのような場合、Stores プロジェクトを右クリックし、「最新表示」を選択します。*FashionFlow\_name* ディレクトリーがワークスペースに表示されます。

- b. **FashionFlow\_name** ディレクトリーを右クリックして、「インポート (Import)」を選択します。  
「インポート (Import)」ウィザードがオープンします。
- c. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。

- d. 「ブラウズ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、`yourDirectory\WC_SAMPLE_55.zip` にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
  - e. 「全選択解除 (Deselect All)」をクリックし、 `MyNewJSPTemplate_All.jsp` ファイルを選択します。
  - f. 「フォルダー (Folder)」フィールドでは、「ストア/Web コンテンツ/`FashionFlow_name` (`Stores/Web Content/FashionFlow_name`)」フォルダーがすでに指定されています。この値をそのまま使用します。
  - g. 「終了」をクリックします。
5. 以下のようにして、チュートリアル用のプロパティ・ファイルを該当するディレクトリーにインポートします。
- a. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、以下のディレクトリーを拡張表示します。  
「Stores」 > 「Web Content」 > 「WEB-INF」 > 「classes」 > 「`FashionFlow_name`」ディレクトリー。
  - b. `FashionFlow_name` ディレクトリーを右クリックして、「インポート (Import)」を選択します。  
「インポート (Import)」ウィザードがオープンします。
  - c. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
  - d. 「ブラウズ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、`yourDirectory\WC_SAMPLE_55.zip` にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
  - e. 「全選択解除 (Deselect All)」をクリックし、 `Tutorial_All_en_US.properties` ファイルを選択します。
  - f. 「フォルダー (Folder)」フィールドでは、「ストア/Web コンテンツ/WEB-INF/クラス/`FashionFlow_name` (`Stores/Web Content/WEB-INF/classes/FashionFlow_name`)」フォルダーがすでに指定されています。この値をそのまま使用します。
  - g. 「終了」をクリックします。
6. 4 つのファイルを、チュートリアル中に作成する新規リソースについてのアクセス制御ポリシーのロードに使用します。以下のファイルが該当します。
- `MyNewViewACPolicy.xml` — この XML ファイルは、新規のビュー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。
  - `MyNewControllerCmdACPolicy.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\_template.xml — この XML ファイルは、新規の Enterprise 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\_template\_en\_US.xml— この XML ファイルは、新規の Enterprise Bean を作成するときにアクセス制御ポリシーの説明を含みます。

以下のようにして、上記のファイルを該当するディレクトリーにコピーします。

- ファイル・システム上で、以下のディレクトリーに移動します。  
`yourDirectory\WC_SAMPLE_55.zip`。
  - この Zip ファイルを拡張表示して、上記の 4 つのファイルを以下のディレクトリーに抽出します。  
`WCStudio_installdir\Commerce\xml\policies\xml`
7. 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。
- WebSphere Studio Application Developer で、サーバー・パースペクティブに切り替えます。
  - Payment Server を開始します。ローカル Payment Server を実行している場合は、「**WebSphereCommercePaymentsServer**」を右クリックして、「スタート」（または「再始動 (Restart)」）を選択します。
  - 「**WebSphereCommerceServer**」を右クリックして、「スタート」（または「再始動 (Restart)」）を選択します。
  - コンソールを注視して、WebSphereCommerceServer サーバーで始動プロセスが終了した時点を見定めます。サーバーを始動すると、以下の情報が表示されます。

```
[4/2/03 12:56:06:286 EST] 66adf8d1 ApplicationMg A WSVR0221I:
Application started: WebSphereCommerceServer
[4/2/03 12:56:06:777 EST] 66adf8d1 HttpTransport A SRVE0171I:
Transport http is listening on port 9,080.
[4/2/03 12:56:10:742 EST] 66adf8d1 HttpTransport A SRVE0171I:
Transport https is listening on port 9,443.
[4/2/03 12:56:10:762 EST] 66adf8d1 HttpTransport A SRVE0171I:
Transport http is listening on port 8,080.
[4/2/03 12:56:10:762 EST] 66adf8d1 HttpTransport A SRVE0171I:
Transport http is listening on port 80.
[4/2/03 12:56:11:123 EST] 66adf8d1 HttpTransport A SRVE0171I:
Transport https is listening on port 443.
[4/2/03 12:56:11:183 EST] 66adf8d1 HttpTransport A SRVE0171I:
Transport https is listening on port 9,043.
[4/2/03 12:56:11:653 EST] 66adf8d1 RMIConnectorC A ADMC0026I:
RMI Connector available at port 2809
[4/2/03 12:56:12:575 EST] 66adf8d1 WsServer
A WSVR0001I: Server server1 open for e-business
```

- 「J2EE ナビゲーター (J2EE Navigator)」ビューで、**Stores** プロジェクトを拡張表示します。次に、「**Web コンテンツ (Web Content)**」 > 「**FashionFlow\_name**」を拡張表示します。

- f. **index.jsp** ファイルを右クリックし、「サーバー上で実行 (Run on Server)」を選択します。  
サンプル・ストアがオープンします。
  - g. 商品を選択し、購入できることを確認します。
- これでチュートリアルを続行する準備ができました。

---

## 新規ビューの作成

このチュートリアルの最初のステップとして、ビューを新規作成します。この新規ビューの名前は **MyNewView** で、対応する JSP テンプレート **MyNewJSPTemplate.jsp** があります。

チュートリアルのこのセクションでは、以下の点を学習します。

- 特定のストアに適用する JSP テンプレートとグラフィック・ファイルを入れる場所
- WebSphere Studio Application Developer を使用して JSP テンプレートを作成する方法
- JSP テンプレートのテキストを含むプロパティ・ファイルを作成する方法
- ビュー・レジストリー (VIEWREG テーブル) を新規「MyNewView」で更新する方法
- 新規ビューのアクセス制御をセットアップする方法
- WebSphere テスト環境 (WTE) を使用して、新規ビューをテストする方法

一般に、新規ビューの作成には、次のステップが含まれています。

1. ビューを命名してビュー・レジストリー中に登録します。
2. JSP テンプレートの変換可能なテキストが保管される新規プロパティ・ファイルを作成します。
3. 新規ビューの新しい JSP テンプレートを作成します。
4. ビューのアクセス制御ポリシーを作成してロードします。

## MyNewView の登録

このシナリオで作成するビューの名前は **MyNewView** です。このビューを **VIEWREG** テーブル中に登録しなければなりません。このテーブルはコマンド・レジストリーの一部です。新規ビューを登録するには、単純な **SQL** ステートメントを使用して **VIEWREG** テーブル中に新規エントリーを作成するだけで済みます。

このステップに進む前に、ご使用のストアの固有 **ID** が分かっているなければなりません。開発データベースに対して以下の **SQL** クエリーを実行すると、この **ID** を判別できます。

```
select STOREENT_ID from STOREENT where IDENTIFIER = 'FashionFlow_name'
```



*FashionFlow\_name* はストアの名前です。次のセクションで必要になるので、ここに値を記録しておいてください。

**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 MyNewView を登録します。

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」>「コマンド行ツール (Command Line Tools)」>「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. ツール設定をクローズします。
5. 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、VIEWREG テーブルに必要なエントリーを作成します。

```
connect to developmentDB user dbuser using dbpassword;
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
 CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('MyNewView', -1, FF_storeent_ID,
'com.ibm.commerce.command.ForwardViewCommand',
'com.ibm.commerce.command.HttpForwardViewCommandImpl',
'docname=MyNewJSPTemplate.jsp', 'This is my new view for tutorial one',
0, null)
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード
- *FF\_storeent\_ID* は、FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID。

「実行」アイコンをクリックします。

SQL コマンドが正常に完了したことを示すメッセージが表示されます。

**Oracle** Oracle データベースを使用している場合は、以下のようにしてデータベースにビューを登録してください。

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。

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('MyNewView',-1, FF_storeent_ID,
 'com.ibm.commerce.command.ForwardViewCommand',
 'com.ibm.commerce.command.HttpForwardViewCommandImpl',
 'docname=MyNewJSPTemplate.jsp','This is my new view for tutorial 1',
 0, null);
```

ここで

- `FF_storeent_ID` は、FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID。

Enter を押して SQL ステートメントを実行します。

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

これで MyNewView が登録されました。

## チュートリアル用のプロパティ・ファイルの作成

このステップでは、JSP テンプレート中で使用される変換可能テキストを保持するプロパティ・ファイルを作成します。変換可能テキストと JSP テンプレート自体を別々にすると、変換のタスクを単純化できますし、このことは世界を結ぶ Web サイトを所有する点で重要な点でもあります。

プロパティ・ファイルを作成するには、WebSphere Studio Application Developer で以下のようにします。

1. Web パースペクティブをオープンします (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「Web」)。
2. Stores Web プロジェクト内で、「Web コンテンツ (Web Content)」>「WEB-INF」>「クラス (classes)」>「FashionFlow\_name」フォルダーを拡張表示します。Tutorial\_All\_en\_US.properties ファイルがあります。
3. 「Tutorial\_All\_en\_US.properties」を右クリックし、「オープンに使用 (Open With)」>「プロパティ・ファイル・エディター (Properties File Editor)」を選択します。
4. 「FashionFlow\_name」フォルダーを右クリックし、「新規」>「その他」>「シンプル」>「ファイル」>「次へ」を選択して、プロパティ・ファイルを新規作成します。  
「新規ファイル (New File)」ウィンドウがオープンします。

- 「ファイル名 (File name)」フィールドで、TutorialNLS\_en\_US.properties と入力してから、「終了」をクリックします。  
新しい空ファイルがオープンします。
- Tutorial\_All\_en\_US.properties ファイルから、セクション 1 を新しい TutorialNLS\_en\_US.properties ファイルにコピーします。これによって、以下の名前と値の対が TutorialNLS\_en\_US.properties ファイルに導入されます。

```
-- SECTION 1 --

ProgrammerGuide=Programmer's Guide
Tutorial=Tutorial: Creating new business logic
ParametersFromCmd= List of parameter-value pairs sent from the
 controller command
CalledByControllerCmd=MyNewView was called by a controller command
CalledByWhichControllerCmd=MyNewView was called by the controller
 command which is -
ControllerParm1=ControllerParm1=
ControllerParm2=ControllerParm2=
Example=This is an example of using the <if> tag from JSP Standard
 Tag Library (JSTL)
UserName=UserName=
Points=Points=
Greeting=Greeting=
UserId=UserId=
FirstInput=Your first input parameter
RegisteredUser=is a registered user
ReferenceNumber=The member reference number of this user is
NotRegisteredUser=is not a registered user
BonusAdmin=Bonus Administration
PointBeforeUpdate=The bonus point before update is
PointAfterUpdate=The bonus point after update is
EnterPoint=Please enter the points, then submit it to the controller
 command

-- END OF SECTION 1 --
```

値の途中での行の中断は、表示の目的です。

- TutorialNLS\_en\_US.properties ファイルを保管します (Ctrl+S)。

新しい TutorialNLS\_en\_US.properties ファイルが、Stores¥Web Content¥WEB-INF¥classes ¥FashionFlow\_name ディレクトリの下に保管され、新規 JSP テンプレートでリソース・バンドルとして使用されます。

**注:** プロパティ・ファイルの内容を変更した場合に、テスト中に変更内容が反映されるようにするには、サーバーを再始動しなければなりません。

## MyNewJSPTemplate の作成

このステップでは、WebSphere Studio で Page Designer を使用して、JSP テンプレートを新規作成します。特に、MyNewView と併用する MyNewJSPTemplate.jsp を作成し

ます。このテンプレートを作成するには、空の JSP テンプレートを作成してから、該当するセクションを別の JSP テンプレート (MyNewJSPTemplate\_All.jsp) からその中に追加します。

JSP テンプレートを新規作成するには、以下のようにします。

1. Web パースペクティブで、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. **Stores** Web プロジェクトを右クリックして、「プロパティ」を選択します。
3. 左側の「**Web**」を選択してから、「使用可能な Web プロジェクト・フィーチャー (Available Web Project Features)」から「**JSP 標準タグ・ライブラリーの組み込み (Include the JSP Standard Tag Library)**」を選択します。「適用」をクリックします。更新が完了したら、「OK」をクリックして、プロパティ・エディターをクローズします。
4. **Web Content**¥*FashionFlow\_name* ディレクトリーを拡張表示します。
5. **MyNewJSPTemplate\_All.jsp** ファイルを右クリックして、「オープンに使用 (Open With)」 > 「Page Designer」を選択します。
6. 「*FashionFlow\_name*」フォルダーを右クリックし、「新規」 > 「JSP ファイル (JSP File)」を選択して、このフォルダー中に JSP テンプレートを新規作成します。  
「新規 JSP ファイル (New JSP File)」ウィンドウがオープンします。
7. 新規ファイルの値を以下のように指定します。
  - a. 「ファイル名 (File name)」フィールドで、MyNewJSPTemplate.jsp と入力します。
  - b. 「マークアップ言語 (Markup Language)」ドロップダウン・リストで、「XHTML」を選択して、「次へ」をクリックします。
  - c. 「タグ・ライブラリーの追加 (Add Tag Library)」をクリックします。  
「タグ・ライブラリーの選択 (Select a Tag Library)」ウィンドウがオープンします。
  - d. 以下のタグ・ライブラリーを選択します。
    - <http://java.sun.com/jstl/core>
    - <http://java.sun.com/jstl/fmt>「OK」に続いて「次へ」をクリックします。
  - e. 「次へ」をクリックします。
  - f. 「(ワークベンチのデフォルトとして使用) ワークベンチ・エンコード ((Use workbench default) Workbench Encoding)」チェック・ボックスのチェックを外します。
  - g. 「エンコード (Encoding)」ドロップダウン・リストで、ISO Latin -1 を選択します。

- h. 「**文書タイプ (Document Type)**」ドロップダウン・リストで、**XHTML 1.0 Transitional** を選択します。
- i. 「**終了**」をクリックします。  
MyNewJSPTemplate.jsp ファイルがオープンします。ファイルのさまざまなビューの「**デザイン (Design)**」、「**ソース (Source)**」、および「**プレビュー**」タブをクリックします。
8. 「**デザイン (Design)**」タブを選択し、「**MyNewJSPTemplate.jsp の内容をここに発行 (Place MyNewJSPTemplate.jsp's content here)**」テキストをクリックします。このテキストを Hello world! に置き換えます。
9. 「**ソース (Source)**」タブに切り替えてから、「**プレビュー**」タブに切り替えます。テキストが変更されていることに注意します。
10. 次に、準備用セクションを MyNewJSPTemplate\_All.jsp ファイルから新しい MyNewJSPTemplate.jsp ファイルにコピーしなければなりません。このセクションは、ファイルに対する更新用のプレースホルダーを設定します。 <!--PREPARATION SECTION マーカーと END OF PREPARATION SECTION --> マーカーの間のテキストを、新しい JSP テンプレートにコピーします。このテキストを JSP テンプレートにコピーする際に、以下のテキストを上書きします。

```
<title> MyNewJSPTemplate.jsp </title>
</head>
<body>
<p> Hello World! </p>
</body>
</html>
```

注: <!--PREPARATION SECTION マーカーと END OF PREPARATION SECTION --> マーカーは、新しい JSP テンプレートにコピーしないでください。これらのマーカーの間にあるテキストのみコピーしてください。

11. セクション 1A と 2 を、MyNewJSPTemplate\_All.jsp ファイルから新しい MyNewJSPTemplate.jsp ファイルにコピーします。 <!-- SECTION 1A --> と <!-- END OF SECTION 1A -->、および <!-- SECTION 2 --> と <!-- END OF SECTION 2 --> のマーカーの間に新規テキストを入れます。これによって、以下のテキストが MyNewJSPTemplate.jsp に導入されます。

```
<!-- SECTION 1A -->

 <%@ include file="include/EnvironmentSetup.jsp"%>

<!-- END OF SECTION 1A -->

<!-- SECTION 2 -->

<fmt:setLocale value="${CommandContext.locale}" />
<fmt:setBundle basename="${sdb.directory}/TutorialNLS" var="tutorial" />

<!-- END OF SECTION 2 -->
```

最初のセクションには、環境変数のセットアップに使用する EnvironmentSetup.jsp ファイルが組み込まれています。2 つ目のセクションを使用して、プロパティ・ファイルからの情報の取り出しに使用するリソース・バンドル・オブジェクトを作成し、ロケールを設定します。

- 次に、グラフィックとテキストを JSP テンプレートに追加します。このステップでも、テキストを MyNewJSPTemplate\_All.jsp ファイルから MyNewJSPTemplate.jsp ファイルにコピーします。今回は、セクション 3 を MyNewJSPTemplate\_All.jsp ファイルから MyNewJSPTemplate.jsp ファイルにコピーします。これによって、以下のテキストが JSP テンプレートに導入されます。

```
<!-- SECTION 3 -->

<table cellpadding="0" cellspacing="0" border="0">
 <tr>
 <td bgcolor="#ff2d2d" >
 border="0"/>
 </td>
 </tr>
</table>

<h1><fmt:message key="ProgrammerGuide" bundle="\${tutorial}" /> </h1>

<h2><fmt:message key="Tutorial" bundle="\${tutorial}" /> </h2>

<!-- END OF SECTION 3 -->
```

セクション 3 は、ストア固有のイメージ・サブフォルダー（「Stores¥Web Content¥FashionFlow\_name¥images (Stores¥Web Content ¥FashionFlow\_name¥images)」）にあるイメージを導入します。また、プロパティ・ファイルからテキストを取り出します。

- MyNewJSPTemplate.jsp ファイルに加えた変更内容を保管します (Ctrl + S)。

## MyNewView のアクセス制御ポリシーの作成とロード

この新規のビューについて、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ビューの実行を許可されていると指定します。このタイプのアクセス制御ポリシーは、開発環境には適していますが、それ以外の環境には適していない場合があることに注意してください。拡張アクセス制御の要件について詳しくは、「WebSphere Commerce セキュリティー・ガイド」を参照してください。

アクセス制御ポリシーは、MyNewViewACPolicy.xml ファイルで定義されています。このファイルは、準備ステップの一部として以下のディレクトリーに含まれています。

```
WCStudio_installdir¥Commerce¥xml¥policies¥xml
```

新しいポリシーをロードする方法は、以下のとおりです。

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。  
`WCStudio_installdir¥Commerce¥bin`
2. 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、ユーザーの開発データベースの名前
- `db_user` は、データベースのユーザーの名前
- `db_password` は、データベースのユーザーのパスワード
- `inputXMLFile` は、アクセス制御ポリシーの仕様を含む XML ファイル。この場合は、`MyNewViewACPolicy.xml` を指定します。

以下はこのコマンドの例で、変数が指定されています。

```
acpload Demo_Dev db2user db2user MyNewViewACPolicy.xml
```

## MyNewView のテスト

新規ビュー作成の最終ステップとして、WebSphere テスト環境で新規ビューをテストします。新規ビュー（および後で新規コマンド）をテストする際には、最初にご使用のストアのホーム・ページを立ち上げなければならないことに注意してください。ストアを立ち上げる必要がある理由は、新規ビューはご使用のストアに対して固有に登録するからです。したがって、新規ビューへのアクセスを試行する際には、その前にまずストアのホーム・ページを立ち上げて、コマンド・コンテキスト中にストア ID 値を設定します。同様に、後で作成するコントローラー・コマンドも、ストアに対して固有に登録し、コマンド・コンテキストからのストア ID が必要です。

ビューをテストするには、以下のようにします。

1. WebSphere Studio Application Developer で、サーバー・パースペクティブをオープンします（「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」）。
2. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」（または「再始動 (Restart)」）を選択します。
3. `Stores¥Web Content¥FashionFlow_name` ディレクトリーの下の `index.jsp` を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。  
Web ブラウザー中にストアのホーム・ページが表示されます。
4. Web ブラウザーで、以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewView
```

数秒後に、以下の画面ショットのように新規 JSP テンプレートが表示されます。

Hello World!



---

# Programmer's Guide

## Tutorial: Creating new business logic

図 31.

---

### 新規コントローラー・コマンドの作成

このステップでは、MyNewControllerCmd と呼ばれる新規のコントローラー・コマンドを作成します。最初の段階では、このコマンドは MyNewView ビューのみ戻します。

チュートリアルはこのセクションでは、以下の点を学習します。

- コントローラー・コマンドに含まれるコードの最小要件
- 新規のコントローラー・コマンド・インターフェースとインプリメンテーション・クラスを作成する方法
- コントローラー・コマンドがビューを戻すようにセットアップする方法
- コマンド・レジストリー中でコントローラー・コマンドを登録する方法
- コントローラー・コマンドのアクセス制御をセットアップする方法



一般に、新規コントローラー・コマンドの作成には、次のステップが関係しています。

1. 新規コマンドをコマンド・レジストリー中に登録します。
2. コマンドのインターフェースを作成します。
3. コマンドのインプリメンテーション・クラスを作成します。
4. コマンドのアクセス制御ポリシーを作成してロードします。
5. コマンドをテストします。

## MyNewControllerCmd の登録

チュートリアルはこの部分では、MyNewControllerCmd と呼ばれる新規のコントローラー・コマンドを作成します。このコマンドはコマンド・レジストリーに登録する必要があります。特に、インターフェースを URLREG テーブル中に登録し、インターフェースとそのインプリメンテーション・クラスとの関連を CMDREG 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、以下のようして MyNewControllerCmd を登録します。

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」>「コマンド行ツール (Command Line Tools)」>「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、URLREG テーブルに必要なエントリーを作成します。

```
connect to developmentDB user dbuser using dbpassword;
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
 AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
 CLASSNAME, TARGET)
values (FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 'This is a new controller command for tutorial one.',
 'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl',
 'local');
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード

- *FF\_storeent\_ID* は、 FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID。

「実行」アイコンをクリックします。

SQL コマンドが正常に完了したことを示すメッセージが表示されます。

**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 MyNewControllerCmd を登録します。

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。

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
 AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
 CLASSNAME, TARGET)
values (FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 'This is a new controller command for tutorial one.',
 'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl','local');
```

ここで

- *FF\_storeent\_ID* は、 FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID。

Enter を押して SQL ステートメントを実行します。

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

CMDREG テーブル中に挿入する 2 つ目のステートメントは、必ずしも必要ではないことに注意してください。このシナリオではインターフェースはデフォルトのインプリメンテーションを使用しますが、実際にインターフェースとインプリメンテーション・クラス間のこの関連をコマンド・レジストリー中に指定する必要があるわけではありません。この点が本書に記載されている目的は、完全な情報にするためです。

## MyNewControllerCmd インターフェースの作成

WebSphere Commerce のプログラミング・モデルに従って、すべての新規コントローラー・コマンドにインターフェースとインプリメンテーション・クラスがなければなりま

せん。このチュートリアルの場合は、インターフェースのベースはサンプル・コード中にあります。現時点では、コード中でコメントとして数種類のセクションに分割されています。チュートリアルを続行していくにつれて、コード中のそれぞれのセクションのコメントを解除していきます。

MyNewControllerCmd インターフェースを作成するには、以下のようになります。

1. WebSphere Studio Application Developer で、Java パースペクティブをオープンします (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「Java」)。
2. **WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。
3. **src** ディレクトリーに移動してから、**com.ibm.commerce.sample.commands** パッケージを拡張表示します。
4. **MyNewControllerCmd.java** インターフェースをダブルクリックして、ファイルをオープンします。
5. ソースで、セクション 1 のコメントを外します (セクションの前の “/\*” と、セクションの後の “\*/” を削除します)。これによって、以下のコードがインターフェースに導入されます。

```
/// Section 1 //

// set default command implement class

static final String defaultCommandClassName =
 "com.ibm.commerce.sample.commands.MyNewControllerCmdImpl";

/// End of section 1////////////////////////////////////
```

このコードのセクションは、デフォルトでインターフェースが MyNewControllerCmdImpl インプリメンテーション・クラスを使用することを指定します。

6. インターフェースに対する変更内容を保管します (Ctrl + S)。

## MyNewControllerCmdImpl インプリメンテーション・クラスの作成

インターフェースを作成し終えたら、次のステップとしてコマンドのインプリメンテーション・クラスを作成します。このチュートリアルの場合は、インプリメンテーション・クラスのベースはサンプル・コード中にあります。現時点では、コード中でコメントとして数種類のセクションに分割されています。チュートリアルを続行していくにつれて、コード中のそれぞれのセクションのコメントを解除していきます。

MyNewControllerCmdImpl インプリメンテーション・クラスを作成するには、以下のようになります。

1. **MyNewControllerCmdImpl.java** クラスをダブルクリックして、オープンします。

2. 「概略 (Outline)」ビューで、**performExecute** メソッドを選択して、そのソース・コードを表示します。
3. **performExecute** メソッド用のソース・コード内で、「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。

```
/// Section 1 //////////////////////////////////////

 /// create a new TypedProperties for output purpose.

 TypedProperty rspProp = new TypedProperty();

/// End of section 1 //////////////////////////////////////
```

これで、コマンドの応答プロパティの保持に使用する **TypedProperty** オブジェクトが新規作成されます。

4. **performExecute** メソッド用のソース・コード内で、「Section 5」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 5 //////////////////////////////////////

 /// see how controller command call a JSP

 rspProp.put(ECConstants.EC_VIEWTASKNAME, "MyNewView");
 setResponseProperties(rspProp);

/// End of section 5////////////////////////////////////
```

このセクションのコードは、2 つのメインタスクを実行します。まず、すべてのコントローラー・コマンドがビューに戻すという **WebSphere Commerce** プログラミング・モデルにとって必要です。このセクション中のコードは、戻されるビューが、以前に作成した **MyNewView** であることを指定します。さらに、コマンドの応答プロパティが、新規 **rspProp** オブジェクトになるように設定します。

5. 変更内容を保管します (Ctrl + S)。
6. コードに加えた変更をコンパイルするには、**WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックして、「プロジェクトの構築 (Build Project)」を選択します。

## コマンドのアクセス制御ポリシーの作成とロード

この新規のコマンドについて、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。このタイプのアクセス制御ポリシーは、開発環境には適していますが、それ以外の環境には適していない場合があることに注意してください。拡張アクセス制御の要件について詳しくは、「*WebSphere Commerce* セキュリティ・ガイド」を参照してください。

アクセス制御ポリシーは、**MyNewControllerCmdACPolicy.xml** ファイルで定義されています。このファイルは、準備ステップの一部として以下のディレクトリーに含まれてい

ます。

```
WCStudio_installDir¥Commerce¥xml¥policies¥xml
```

新しいポリシーをロードする方法は、以下のとおりです。

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。

```
WCStudio_installDir¥Commerce¥bin
```

2. 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、ユーザーの開発データベースの名前
- `db_user` は、データベースのユーザーの名前
- `db_password` は、データベースのユーザーのパスワード
- `inputXMLFile` は、アクセス制御ポリシーの仕様を含む XML ファイル。この場合は、`MyNewControllerCmdACPolicy.xml` を指定します。

以下はこのコマンドの例で、変数が指定されています。

```
acpload Demo_Dev db2user db2user MyNewControllerCmdACPolicy.xml
```

## MyNewControllerCmd のテスト

インターフェース、インプリメンテーション・クラス、コマンド登録、およびアクセス制御情報をすべて作成し終えたので、新規コントローラー・コマンドをテストできます。

Java コードを変更している場合に、変更内容が認識されるようにするには、その前にテスト・サーバーを再始動しなければなりません。

新規コードをテストするには、以下のようになります。

1. サーバー・パースペクティブに切り替えます (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」)。
2. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」 (または「再始動 (Restart)」) を選択します。
3. `Stores¥Web Content¥FashionFlow_name` ディレクトリーの下での `index.jsp` を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。  
Web ブラウザー中にストアのホーム・ページが表示されます。
4. Web ブラウザーで、以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd
```

数秒後に、以下の画面ショットのように新規 JSP テンプレートが表示されます。

Hello World!



---

# Programmer's Guide

## Tutorial: Creating new business logic

図 32.

---

### MyNewControllerCmd から MyNewView への情報の引き渡し

このステップでは、MyNewControllerCmd に変更を加えて、MyNewView に情報を渡せるようにします。情報をビューに渡すときの 2 つの別個の方法が示されています。最初に、応答プロパティー用に TypedProperties オブジェクトを使用する方法と、このオブジェクトから JSP テンプレートに情報を抽出する方法について学びます。次に、情報を JSP テンプレートに渡すのに使用する Data Bean を新規作成する方法を学びます。

### TypedProperties オブジェクトを使用した情報の引き渡し

このセクションでは、MyNewControllerCmdImpl を変更して、情報を JSP テンプレートに渡します。特に、コマンドに変更を加えて、コマンドからの応答プロパティーに使用する既存の rspProp TypedProperties オブジェクトに追加の名前と値の対を加えます。JSP テンプレート中で JSTL 式の言語を使用して、応答プロパティーから情報を抽出し

ます。

チュートリアルはこのセクションでは、以下の点を学習します。

- コントローラー・コマンドに変更を加えて、 `TypedProperty` 中に追加の応答プロパティを組み込む方法
- JSP テンプレートに変更を加えて、 JSTL 式の言語を使用して応答プロパティから情報を取り出す方法

JSP テンプレートで `TypedProperties` オブジェクト中の情報を表示可能にするには、以下のようにします。

1. 最初のステップとして、以下のように `MyNewControllerCmdImpl` クラスに変更を加えます。

- a. Java パースペクティブに切り替えます。
- b. 以下のディレクトリーを拡張表示します。  
「**WebSphereCommerceServerExtensionsLogic**」 > 「**src**」 > 「**com.ibm.commerce.sample.commands**」。
- c. **MyNewControllerCmdImpl.java** をダブルクリックして、「概略 (Outline)」ビューでその **performExecute** メソッドを選択します。
- d. `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。

```
/// Section 2 //////////////////////////////////////

 /// see how the controller command pass in variables to JSP

 /// add additional parameters in controller command to rspProp
 /// for response
 String message1 = "Hello from IBM!";

 rspProp.put("controllerParm1", message1);
 rspProp.put("controllerParm2", "Have a nice day!");

/// End of section 2////////////////////////////////////
```

上記の部分コードでは、応答プロパティに入れられる 2 つの新しいパラメーターを作成します。このオブジェクトは、最終的にビューに渡されます。

- e. 変更内容を保管します。
  - f. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックして、「プロジェクトの構築 (**Build Project**)」を選択し、コマンドをコンパイルします。
2. 次に、以下のようにして `MyNewJSPTemplate.jsp` ファイルを更新しなければなりません。

- a. まだ JSP テンプレート・ファイルをオープンしていない場合は、以下のようにします。

- 1) Web パースペクティブで、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替え **Stores** Web プロジェクトを拡張表示します。
- 2) **Web Content¥FashionFlow\_name** ディレクトリーに移動します。
- 3) **MyNewJSPTemplate\_All.jsp** を右クリックして、「オープンに使用 (Open With)」>「Page Designer」を選択します。
- 4) **MyNewJSPTemplate.jsp** を右クリックして、「オープンに使用 (Open With)」>「Page Designer」を選択します。

- b. セクション 4 を、MyNewJSPTemplate\_All.jsp ファイルから新しい MyNewJSPTemplate.jsp ファイルにコピーします。新規テキストを、`<!-- SECTION 4 -->` マーカーと `<!-- END OF SECTION 4 -->` マーカーの間に入れます。これによって、以下のテキストが MyNewJSPTemplate.jsp に導入されます。

```
<!-- SECTION 4 -->

<h3><fmt:message key="ParametersFromCmd" bundle="\${tutorial}" /> </h3>

<fmt:message key="ControllerParm1" bundle="\${tutorial}" />
<c:out value="\${controllerParm1}"/>

<fmt:message key="ControllerParm2" bundle="\${tutorial}" />
<c:out value="\${controllerParm2}"/>

<!-- END OF SECTION 4 -->
```

このセクションは、JSTL 式の言語を使用して、コントローラー・コマンドから渡された値を取得して表示します。

- c. 変更内容を保管します。
3. 次のステップとして、以下のようにして、コントローラー・コマンドと JSP テンプレートに対する変更内容をテストします。

- a. サーバー・パースペクティブに切り替えます (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」)。
- b. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」(または「再始動 (Restart)」) を選択します。
- c. Stores¥Web Content¥FashionFlow\_name ディレクトリーの下 **index.jsp** を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。Web ブラウザー中にストアのホーム・ページが表示されます。
- d. Web ブラウザーで、以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd
```

数秒後に、以下の画面ショットのように新規 JSP テンプレートが表示されます。



Hello World!



---

## Programmer's Guide

### Tutorial: Creating new business logic

#### List of parameter-value pairs sent from the controller command

ControllerParm1= Hello from IBM!

ControllerParm2= Have a nice day!

図 33.

#### Data Bean を使用した情報の引き渡し

このセクションでは、ビューが直接呼び出されたのか、それともコントローラー・コマンドによって呼び出されたのかを判別するコードを追加します。後者の場合は、ビューを呼び出したコマンドの名前も JSP テンプレートに表示されるはずですが。

この情報をビューに渡すには、MyNewDataBean という Data Bean を新規作成します。MyNewJSPTemplate は、新しい情報を表示できるようにするためにも変更されません。

MyNewDataBean は、コントローラー・コマンドから JSP テンプレートに情報を入手できるようにするためだけに使用されます。このことは、データベースから情報を入手できるようにすることと対比されます。このチュートリアルでは、データベースから JSP テンプレートに情報を入手できるようにする Data Bean を新規作成する方法が後述されています。

このチュートリアルで、コントローラー・コマンドから情報を入手できるようにするためだけに Data Bean を使用する理由について以下に説明します。この Bean を作成する理由は 2 つあります。まず、属性のロジック・グループを使用できるようになるための良いプログラミングの練習になるという理由と、そして 2 つ目に、Web ページの開発者にとって、Data Bean を使用して Web ページに情報を追加する方が、TypedProperties 応答プロパティー・オブジェクトを使用して追加するよりも簡単という理由です。

チュートリアルのこのセクションでは、以下の点を学習します。

- 新規 Data Bean を作成する方法
- コントローラー・コマンドに変更を加えて、Data Bean をインスタンス化する方法
- コントローラー・コマンドを使用して、Data Bean 中で属性を設定する方法
- インスタンス化した Data Bean を JSP テンプレートに渡す方法
- JSP テンプレートに変更を加えて、Data Bean から情報を取り出す方法
- JSP テンプレート中での <if> タグの使用例

## MyNewDataBean の作成

MyNewDataBean は、MyNewJSPTemplate.jsp ページに情報を渡すのに使用します。このチュートリアルの他のセクションと同様に、Data Bean のベースはサンプル・コード中にあります。現時点では、コード中でコメントとして数種類のセクションに分割されています。チュートリアルを続行していくにつれて、コード中のそれぞれのセクションのコメントを解除していきます。

MyNewDataBean を作成するには、以下のようになります。

1. Java パースペクティブをオープンして、「パッケージ探査 (Package Explorer)」ビューを使用します。
2. 以下のディレクトリーを拡張表示します。  
**WebSphereCommerceServerExtensionsLogic > src > com.ibm.commerce.sample.databeans。**
3. **MyNewDataBean.java** をダブルクリックして、そのソース・コードを表示します。
4. メイン・クラス用のソース・コード内で、「Section 1」をコメント解除します。これによって、以下のコードがクラスに導入されます。

```

/// Section 1 //////////////////////////////////////
/// create fields and accessors (setter/getter methods)

private java.lang.String callingCommandName = null;
private boolean calledByControllerCmd = false;

public java.lang.String getCallingCommandName() {
 return callingCommandName;
}

public void setCallingCommandName(java.lang.String newCallingCommandName)
{
 callingCommandName = newCallingCommandName;
}

public boolean getCalledByControllerCmd() {
 return calledByControllerCmd;
}

public void setCalledByControllerCmd(boolean newCalledByControllerCmd)
{
 calledByControllerCmd = newCalledByControllerCmd;
}

/// End of Section 1 //////////////////////////////////////

```

上記のコードは、ビューの URL によってビューが直接呼び出された場合ではなく、コントローラー・コマンドによってビューが戻された場合に、情報の表示に使用する 2 つの変数を導入します。

5. 変更内容を保管します。

## MyNewControllerCmd を使用した MyNewDataBean のインスタンス化とその属性の設定

このステップでは、MyNewControllerCmdImpl に変更を加えて、MyNewDataBean をインスタンス化し、この Bean の属性を設定します。

MyNewControllerCmdImpl を変更するには、以下のようになります。

1. Java パースペクティブ中で、以下のディレクトリーを拡張表示します。  
**「WebSphereCommerceServerExtensionsLogic」 > 「src」 > 「com.ibm.commerce.sample.commands」**。
2. **MyNewControllerCmdImpl.java** をダブルクリックして、そのソース・コードを表示します。
3. メイン・クラス用のコード内で、「Import Section 1」をコメント解除し、このクラス中で新規 Data Bean を使用できるようにします。これによって、以下のコードがクラスに導入されます。

```

/// Import Section 1 //////////////////////////////////////
import com.ibm.commerce.sample.databeans.*;
/// End of Import Section 1 //////////////////////////////////////

```

4. 「概略 (Outline)」ビューで、**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 3A」および「Section 3B」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 3A////////////////////////////////////

/// instantiate the MyNewDataBean databean and set the properties,
/// then add the instance to rspProp for response

MyNewDataBean mndb = new MyNewDataBean();
mndb.setCallingCommandName(this.getClass().getName());
mndb.setCalledByControllerCmd(true);

/// end of section 3A////////////////////////////////////

/// Section 3B////////////////////////////////////
rspProp.put("mndbInstance", mndb);

/// end of section 3B////////////////////////////////////

```

上記の部分コードは、**MyNewDataBean** オブジェクトをインスタンス化し、このオブジェクト中に 2 つのパラメーターを設定し (コントローラー・コマンドによって呼び出されたことと、呼び出し側のコマンドを示す)、**Data Bean** オブジェクトを応答プロパティーに書き込んで JSP テンプレートで使用できるようにします。

6. 変更内容を保管します。
7. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックして、「プロジェクトの構築 (Build Project)」を選択し、コード変更をコンパイルします。

### MyNewJSPTemplate での MyNewDataBean の使用

このセクションでは、**MyNewJSPTemplate** に変更を加えて、ビューがコントローラー・コマンドによって戻されたかどうかを示します。コントローラー・コマンドによって戻された場合は、そのコントローラー・コマンドの名前も表示します。JSP テンプレートは、**MyNewDataBean** の値を基にして、この判別のための条件ロジックに **JSTL** タグを使用します。

JSP 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. まだ JSP テンプレート・ファイルをオープンしていない場合は、以下のようにします。
  - a. Web パースペクティブで、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替え **Stores** Web プロジェクトを拡張表示します。
  - b. **Web Content**¥**FashionFlow\_name** ディレクトリに移動します。

- c. **MyNewJSPTemplate\_All.jsp** ファイルと **MyNewJSPTemplate.jsp** ファイルを両方とも強調表示し、右クリックして、「**オープンに使用 (Open With)**」>「**Page Designer**」を選択します。
2. 「Section 5」を **MyNewJSPTemplate\_All.jsp** ファイルから **MyNewJSPTemplate.jsp** ファイルにコピーします。これによって、以下のテキストが JSP テンプレートに導入されます。

```
<!-- SECTION 5 -->
```

```
<c:if test="${mndbInstance.calledByControllerCmd}">
 <fmt:message key="Example" bundle="${tutorial}" />

 <fmt:message key="CalledByControllerCmd" bundle="${tutorial}" />

 <fmt:message key="CalledByWhichControllerCmd" bundle="${tutorial}" />
 <c:out value="${mndbInstance.callingCommandName}" />

</c:if>
```

```
<!-- END OF SECTION 5 -->
```

このコードのセクションは、JSTL `<if>` タグを使用して、呼び出し側のコントローラー・コマンドに関する情報を表示するかどうかを判別します。また、チュートリアルのリソース・バンドルから変換可能テキストを取り出します。

3. 変更内容を保管します。

### 変更を加えた JSP テンプレートのテスト

変更を加えた JSP テンプレートをテストするには、以下のようになります。

1. サーバー・パースペクティブに切り替えます (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」)。
2. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」(または「再始動 (Restart)」) を選択します。
3. Stores¥Web Content¥FashionFlow\_name ディレクトリーの下での **index.jsp** を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。  
Web ブラウザー中にストアのホーム・ページが表示されます。
4. Web ブラウザーで、以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd
```

数秒後に、以下の画面ショットのように JSP テンプレートが表示されます。

Hello World!



---

## Programmer's Guide

### Tutorial: Creating new business logic

#### List of parameter-value pairs sent from the controller command

ControllerParm1= Hello from IBM!

ControllerParm2= Have a nice day!

This is an example of using the tag from JSP Standard Tag Library (JSTL)

MyNewView was called by a controller command

MyNewView was called by the controller command which is -

**`com.ibm.commerce.sample.commands.MyNewControllerCmdImpl`**

図 34.

- 次に、以下の URL を入力してビューを直接に呼び出し、表示されている情報で異なっている部分を見つけます。

<http://localhost/webapp/wcs/stores/servlet/MyNewView>

## MyNewControllerCmd 中の URL パラメーターの構文解析と妥当性検査

このステップでは、コントローラー・コマンドに変更を加えて、そのコントローラー・コマンドを呼び出す URL を介して渡されるパラメーターを使用できるようにします。またコマンドに妥当性検査ロジックを組み込んで、必要なパラメーターが組み込まれていることと、パラメーターに適切な値が使用されていることを確認します。

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・スタブのみです。それは以下のコードで構成されます。

```
public void validateParameters() throws ECAApplicationException {
}
```

この時点で、カスタマイズ済みのパラメーター検査をコマンドに追加し、URL パラメーターを JSP テンプレートに渡さなければなりません。 `validateParameters` メソッドを変更するとき、URL パラメーターに対応する新しいフィールドを追加します。

チュートリアルはこのセクションでは、以下の点を学習します。

- 新しいフィールドをコマンドに追加して、URL パラメーターに対応させる方法
- `getRequestProperties` メソッドを使用して、URL 入力パラメーターをこれらのフィールドに取り込む方法
- 欠落パラメーター例外をキャッチする方法
- URL パラメーターをビューに渡す適切な方法
- 欠落しているパラメーター値や誤っているパラメーター値に関するさまざまなテスト・ケースの例

## MyNewControllerCmd への新規フィールドの追加

このステップでは、URL パラメーターの新規フィールドを 2 つ作成します。これらは、コマンド・インターフェースとインプリメンテーション・クラスの両方に追加されます。1 つの URL パラメーターは、ユーザー名を示す文字列値です。もう 1 つは、ボーナス・ポイントの入力値の受諾に使用する整数です。

これらの新規フィールドを追加するには、以下のようにします。

1. Java パースペクティブに切り替えます。
2. 以下のディレクトリーを拡張表示します。  
「WebSphereCommerceServerExtensionsLogic」 > 「src」 > 「com.ibm.commerce.sample.commands」。
3. **MyNewControllerCmd.java** インターフェースをダブルクリックして、そのソース・コードを表示します。
4. 「Section 2」をコメント解除して、新規フィールドおよび対応する `getter` メソッドをインターフェースに追加します。これによって、以下のコードがクラスに導入されます。

```

/// Section 2 //////////////////////////////////////
// set interface methods

public java.lang.Integer getPoints() ;
public java.lang.String getUsername() ;
public void setPoints(java.lang.Integer newPoints) ;
public void setUsername(java.lang.String newUserName) ;

```

```

/// End of section 2////////////////////////////////////

```

5. 変更内容を保管します。
6. **MyNewControllerCmdImpl.java** クラスをダブルクリックして、そのソース・コードを表示します。
7. メイン・クラス中で「Section 1」をコメント解除して、新規フィールドおよび対応する getter メソッドと setter メソッドをクラスに追加します。これによって、以下のコードがクラスに導入されます。

```

/// Section 1 //////////////////////////////////////

/// create and implement controller command's fields and accessors
/// (setter/getter methods)

private java.lang.String userName = null;
private java.lang.Integer points;

public java.lang.Integer getPoints() {
 return points;
}

public java.lang.String getUsername() {
 return userName;
}

public void setPoints(java.lang.Integer newPoints) {
 points = newPoints;
}

public void setUsername(java.lang.String newUserName) {
 userName = newUserName;
}

```

```

/// End of Section 1 //////////////////////////////////////

```

8. 変更内容を保管します。



## URL パラメーターのビューへの引き渡し

このステップでは、コードを組み込んで、入力パラメーターを JSP テンプレートに渡します。そのためには、入力パラメーターの値を使用して、Data Bean 中のフィールドを設定します。

URL パラメーターを渡すには、以下のようにします。

1. **MyNewControllerCmdImpl.java** をダブルクリックします。
2. 「概略 (Outline)」ビューで、**performExecute** メソッドを選択します。
3. **performExecute** メソッドのソース・コード内で、「Section 3C」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 3C////////////////////////////////////

// pass the input information to the databean
mndb.setUsername(this.getUserName());
mndb.setPoints(this.getPoints());

/// end of section 3C////////////////////////////////////
```

このコードは、Data Bean オブジェクト中に値を設定し、JSP テンプレート中でそれらの値を使用できるようにします。

4. 変更内容を保管します。

## 欠落パラメーターと妥当性検査値のキャッチ

このステップでは、**validateParameters** メソッドに変更を加えて、エラー検査とパラメーター妥当性検査ロジックを導入します。変更し終わると、コードは以下の点を検査するようになります。

- 最初の入力パラメーターがない場合、「パラメーター未検出」例外がスローされます。これは、最初の入力パラメーターが必須パラメーターであることが理由です。この場合、顧客に対して一般エラー・ページが表示されます。
- 2 番目のパラメーターはオプションです。ただし、2 番目の入力パラメーターがない場合、「パラメーター未検出」例外はスローされません。代わりに、2 番目のパラメーターの値にはデフォルトのゼロが使用され、顧客はエラーの影響を受けません。処理を続行します。

このエラー検査を追加するには、以下のようにします。

1. **MyNewControllerCmdImpl.java** をダブルクリックします。
2. 「概略 (Outline)」ビューで、**validateParameters** を選択します。
3. **validateParameters** 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。

```
/// Section 1 //////////////////////////////////////

/// uncomment to check parameters
```

```

 final String strMethodName = "validateParameters";

TypedProperty prop = getRequestProperties();

/// retrieve required parameters
try {
 setUsername(prop.getString("input1"));

} catch (ParameterNotFoundException e) {
 /// the next exception uses _ERR_CMD_MISSING_PARAM EMessage object
 /// defined in EMessage class
 throw new EApplicationException(EMessage._ERR_CMD_MISSING_PARAM,
 this.getClass().getName(), strMethodName,
 EMessageHelper.generateMsgParms(e.getParamName()));
}

/// retrieve optional Integer
/// set input2 = 0 if no input value
setPoints(prop.getInteger("input2",0));

/// End of section 1////////////////////////////////////

```

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、または値が誤りである場合に、値をゼロに設定します。

4. 変更内容を保管します。

## MyNewDataBean への新規フィールドの追加

このステップでは、新規フィールドとそれらの関連 getter メソッドを MyNewDataBean Data Bean に追加し、JSP テンプレートで URL パラメーターを使用できるようにします。

MyNewDataBean を変更するには、以下のようにします。

1. **MyNewDataBean.java** をダブルクリックして、そのソース・コードを表示します。
2. 「Section 2」のコメントを解除して、以下のコードをクラスに導入します。

```

/// Section 2 //////////////////////////////////////

private java.lang.String userName = null;
private java.lang.Integer points;

public String getUserName() {
 return userName;
}

public void setUsername(java.lang.String newUserName) {
 userName = newUserName;
}

```

```

 }

 public Integer getPoints() {
 return points;
 }

 public void setPoints(java.lang.Integer newPoints) {
 points = newPoints;
 }

 /// End of Section 2 //////////////////////////////////////

```

3. 変更内容を保管します。
4. **WebSphereCommerceServerExtensionsLogic** を右クリックし、「プロジェクトの構築 (Build Project)」を選択して、コードの変更をコンパイルします。

## URL パラメーターを表示するための MyNewJSPTemplate の変更

このステップでは、MyNewJSPTemplate.jsp ファイルに変更を加えて、URL 入力パラメーターを表示する新規セクションを追加します。そのためには、以下のようにします。

1. まだ JSP テンプレート・ファイルをオープンしていない場合は、以下のようにします。
  - a. Web パースペクティブで、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替え **Stores Web** プロジェクトを拡張表示します。
  - b. 「Web コンテンツ¥FashionFlow\_name (Web Content¥FashionFlow\_name)」サブフォルダーに移動します。
  - c. **MyNewJSPTemplate\_All.jsp** ファイルと **MyNewJSPTemplate.jsp** ファイルを両方とも強調表示し、右クリックして、「オープンに使用 (Open With)」> 「**Page Designer**」を選択します。
2. 「Section 6」を MyNewJSPTemplate\_All.jsp ファイルから MyNewJSPTemplate.jsp ファイルにコピーします。これによって、以下のテキストが JSP テンプレートに導入されます。

```

<!-- SECTION 6 -->

<fmt:message key="UserName" bundle="${tutorial}" />
<c:out value="{mndbInstance.userName}" />

<fmt:message key="Points" bundle="${tutorial}" />
<c:out value="{mndbInstance.points}" />

<!-- END OF SECTION 6 -->

```

3. 変更内容を保管します。

## URL パラメーター値のテスト

次のステップとして、以下のようにして、新規エラー検査が適切に作動するかどうかをテストします。

1. サーバー・パースペクティブに切り替えます (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」)。
2. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」 (または「再始動 (Restart)」) を選択します。
3. Stores¥Web Content¥FashionFlow\_name ディレクトリーの下 **index.jsp** を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。  
Web ブラウザー中にストアのホーム・ページが表示されます。
4. ケース 1: 最初のテスト・ケースとして、URL から両方のパラメーターとも除外します。ストア・ホーム・ページが表示されたら、以下の URL を入力します。

```
http://localhost/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・エラーを示します。

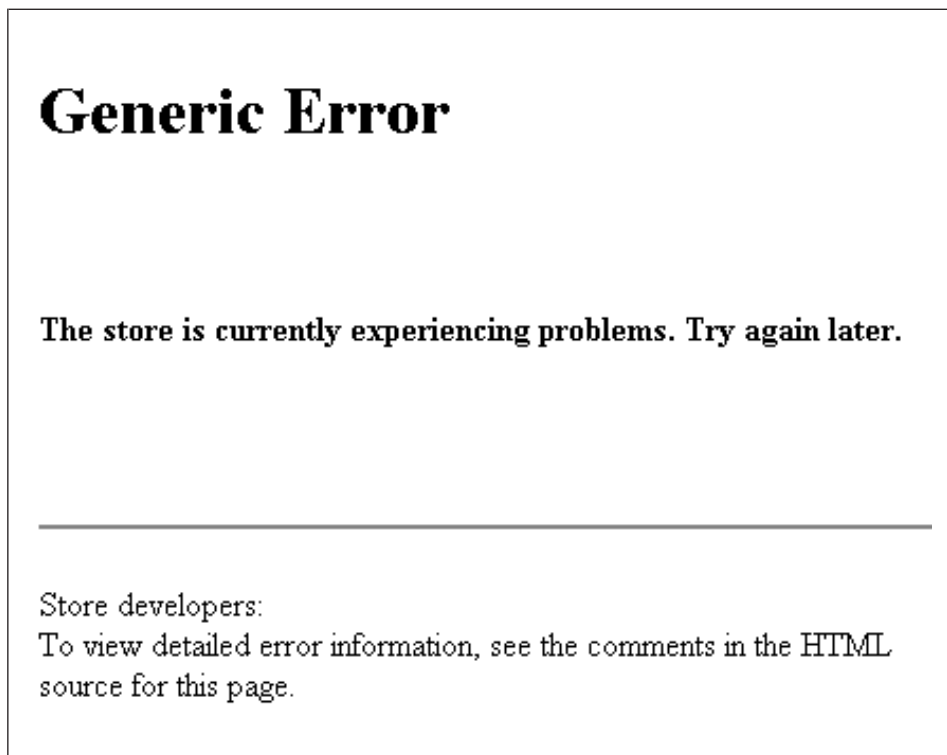


図 35.

WebSphere Studio Application Developer 中のコンソールに以下のような情報が表示されます。

```
timeStamp 6730e546 CommerceSrvr E
com.ibm.commerce.sample.commands.MyNewControllerCmdImpl validateParameters
CMN0206E すべてのフィールドを確認してください。 "input1" は必須フィールドで
す。
```

5. ケース 2: 次のテスト・ケースとして、最初のパラメーターは有効値を使用しますが、2 番目のパラメーターはオフにします。このケースでは、エラーが検出されないことが予想されます。なぜなら、欠落している 2 番目のパラメーターにデフォルトのゼロが使用されるからです。以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?input1=abc
```

このコマンドの結果として、MyNewJSPTemplate ページが表示され、input2 の値としてゼロが表示されます。

## Programmer's Guide

### Tutorial: Creating new business logic

#### List of parameter-value pairs sent from the controller command

```
ControllerParm1= Hello from IBM!
```

```
ControllerParm2= Have a nice day!
```

```
This is an example of using the tag from JSP Standard Tag Library
(JSTL)
```

```
MyNewView was called by a controller command
```

```
MyNewView was called by the controller command which is -
```

```
com.ibm.commerce.sample.commands.MyNewControllerCmdImpl
```

```
UserName= abc
```

```
Points= 0
```

図 36.

6. ケース 3: このテスト・ケースでは、最初の入力パラメーターには有効パラメーターを使用し、2 番目のパラメーターには無効パラメーターを使用 (整数の代わりにストリングを使用) します。ケース 2 と同様に、エラー検査により 2 番目の入力パラメーターがゼロに変更されるので、エラーは表示されません。以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
input1=abc&input2=abc
```

このコマンドの結果として、MyNewJSPTemplate ページが表示され、input2 の値としてゼロが表示されます。

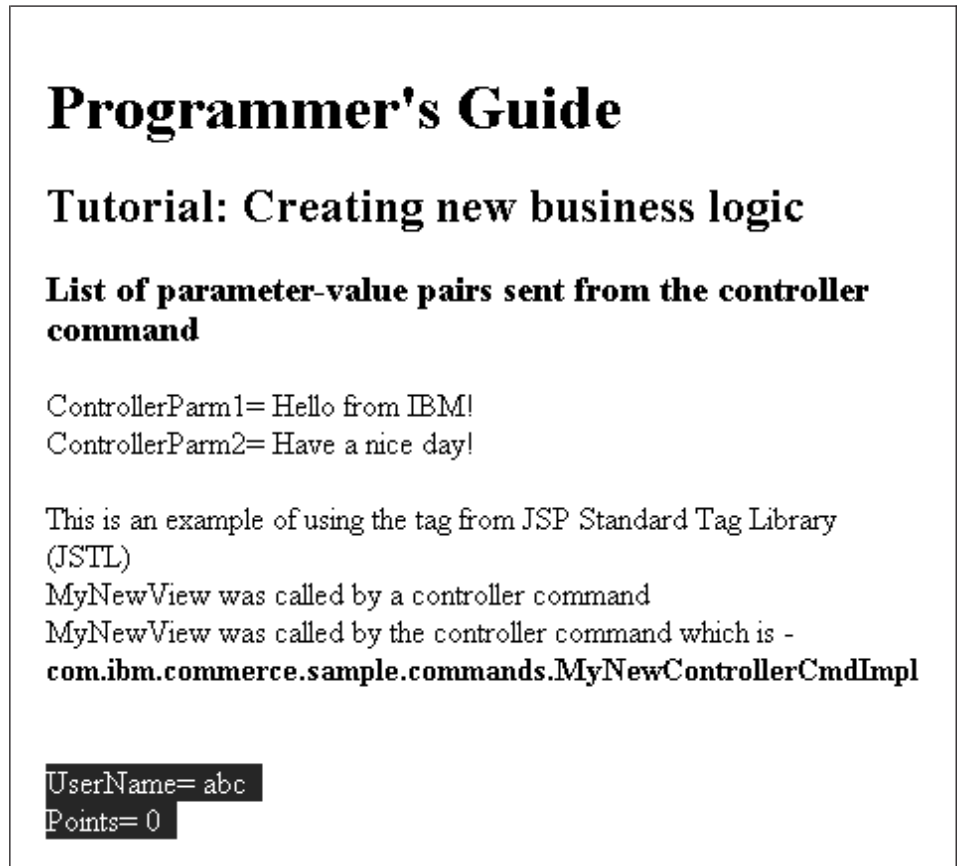


図 37.

7. ケース 4: このケースでは、両方の URL 入力パラメーターとも有効なパラメーターを使用します。以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
input1=abc&input2=1000
```

このコマンドの結果として、MyNewJSPTemplate ページが表示され、ユーザーに 1000 ポイントが表示されます。

# Programmer's Guide

## Tutorial: Creating new business logic

### List of parameter-value pairs sent from the controller command

```
ControllerParm1= Hello from IBM!
ControllerParm2= Have a nice day!
```

This is an example of using the tag from JSP Standard Tag Library (JSTL)

MyNewView was called by a controller command

MyNewView was called by the controller command which is -

**com.ibm.commerce.sample.commands.MyNewControllerCmdImpl**

```
UserName= abc
Points= 1000
```

図 38.

## 新規タスク・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。

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。このセクションは、2 つのステップに分けられます。最初のステップでは、新しい

タスク・コマンドを作成します。2 番目のステップでは、コントローラー・コマンドの `performExecute` メソッドに変更を加えて、新しいタスク・コマンドを呼び出すようにします。

このステップでは、新規のタスク・コマンド・インターフェースおよびその関連インプリメンテーション・クラスを作成します。最初の段階では、新しいタスク・コマンドは、`handle view` パラメーター以外は非常に小さいものです。`defaultCommandClassName`、URL パラメーター、およびボーナス・ポイントの現行値に関するフィールドしかありません。ボーナス・ポイントの現行値を取得するメソッドがあります。

チュートリアルはこのステップでは、以下の点を学習します。

- 新規のタスク・コマンド・インターフェースおよびその関連インプリメンテーション・クラスを作成する方法
- タスク・コマンド中に最小限必要なコード
- タスク・コマンドにフィールドとメソッドを追加する方法

## MyNewTaskCmd の作成

このステップでは、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` という新規タスク・コマンドが使用できるようになります。このコマンドは、`FashionFlow` サンプルに基づくストアで使用されません。

`MyNewTaskCmd` を作成するには、以下のようになります。

1. Java パースペクティブに切り替え、**WebSphereCommerceServerExtensionsLogic** プロジェクトを選択します。
2. **src** ディレクトリを拡張表示してから、**com.ibm.commerce.sample.commands** ディレクトリを拡張表示します。
3. **MyNewTaskCmd.java** インターフェースをダブルクリックして、そのソース・コードを表示します。
4. このインターフェース用のソース・コード内で、「Section 1」をコメント解除し、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。これによって、以下のコードがインターフェースに導入されます。



```

/// Section 1 //////////////////////////////////////
// set default command implement class

static final String defaultCommandClassName=
 "com.ibm.commerce.sample.commands.MyNewTaskCmdImpl";

/// End of section 1////////////////////////////////////

```

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、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・クラス間のマッピングを作成する必要があります。

5. 次に、「Section 2」をコメント解除し、MyNewTaskCmdImpl インプリメンテーション・クラスで使用される getter メソッドと setter メソッドを作成します。これらのメソッドは、以下のタイプの情報に対応するフィールドに関するメソッドです。
  - 顧客のユーザー ID
  - ボーナス・ポイントの値
  - グリーティング・メッセージ

「Section 2」をコメント解除すると、以下のコードがインターフェースに導入されます。

```

/// Section 2 //////////////////////////////////////
// set interface methods

public void setInputUserName(java.lang.String inputUserName);
public void setInputPoints(Integer inputPoints);
public void setGreetings(java.lang.String greeting);

public java.lang.String getInputUserName();
public java.lang.Integer getInputPoints();
public java.lang.String getGreetings();

/// End of section 2////////////////////////////////////

```

6. 変更内容を保管します。
7. **MyNewTaskCmdImpl.java** インプリメンテーション・クラスをダブルクリックして、そのソース・コードを表示します。
8. 「Section 1A」と「Section 1B」をコメント解除し、インプリメンテーション・クラス中にフィールドおよび対応する getter メソッドと setter メソッドを作成します。これによって、以下のコードがクラスに導入されます。

```

//// Section 1A //////////////////////////////////////
private java.lang.String inputUserName;
private java.lang.String greetings;
private java.lang.Integer inputPoints;

////End of Section 1A //////////////////////////////////////

//// Section 1B //////////////////////////////////////

public void setInputUserName(java.lang.String newInputUserName) {
 inputUserName = newInputUserName;
}

public void setInputPoints(Integer newInputPoints) {
 inputPoints = newInputPoints;
}

public void setGreetings(java.lang.String newGreetings) {
 greetings = newGreetings;
}

public java.lang.String getInputUserName() {
 return inputUserName;
}

public Integer getInputPoints() {
 return inputPoints;
}

public java.lang.String getGreetings() {
 return greetings;
}

////End of Section 1B //////////////////////////////////////

```

作業内容を保管します。

9. 「概略 (Outline)」ビューで、MyNewTaskCmdImpl クラスの **performExecute** メソッドを選択します。
10. この **performExecute** 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。

```

/// Section 1 //////////////////////////////////////
/// modify the greetings and see it in the NVP list
setGreetings("Hello ! " + getInputUserName());
/// End of section 1 //////////////////////////////////////

```

このコードはグリーティング値を更新します。それから、`getGreetings()` メソッドを介してこのグリーティング値を他のオブジェクトで使用できるようになります。この値は、名前と値の組 (NVP) のリストに追加されます。

11. 作業内容を保管します。

## 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. Java パースペクティブで、**MyNewControllerCmdImpl.java** クラスをダブルクリックします。
2. 「概略 (Outline)」ビューで、**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、「Area 4」に移動します。
4. 「Section 4A」、「Section 4B」、および「Section 4C」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。

```
/// Section 4A
/// see how the controller command call a task command

MyNewTaskCmd cmd = null;

try {

 cmd = (MyNewTaskCmd) CommandFactory.createCommand(
 "com.ibm.commerce.sample.commands.MyNewTaskCmd", getStoreId());

 // this is required for all commands
 cmd.setCommandContext(getCommandContext());

 // set input parameters to task command
 cmd.setInputUserName(getUserName());
 cmd.setInputPoints(getPoints()); // change to Integer

/// End Section 4A //////////////////////////////////////

/// Section 4B //////////////////////////////////////

 // invoke the command's performExecute method
 cmd.execute();

 // retrieve output parameter from task command, then put it to
 // response properties
 rspProp.put("taskOutputGreetings", cmd.getGreetings());

/// End Section 4B //////////////////////////////////////

/// Start Section 4C //////////////////////////////////////
} catch (EException ex) {
 // throw the exception as is
```

```

 throw (EException) ex;
 }

 /// End Section 4C //////////////////////////////////////

```

「Section 4A」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・オブジェクトを作成します。その後、コマンド・コンテキストを設定し、タスク・コマンドの入力パラメーターを設定します。「Section 4B」は、タスク・コマンドの実行を呼び出す前に、アクセス制御のために `validateParameters` メソッドを呼び出します。その後、タスク・コマンドからグリーティング値を取り出します。「Section 4C」は、単純な例外の試行ブロックです。

5. 変更内容を保管します。
6. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックし、「プロジェクトの構築 (Build Project)」を選択して、コードの変更をコンパイルします。

## グリーティング・メッセージを追加するための MyNewJSPTemplate の変更

このステップでは、MyNewJSPTemplate.jsp ファイルに変更を加えて、グリーティング・メッセージを表示する新規セクションを追加します。そのためには、以下のようになります。

1. まだ JSP テンプレート・ファイルをオープンしていない場合は、以下のようになります。
  - a. Web パースペクティブで、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替え **Stores** Web プロジェクトを拡張表示します。
  - b. 「Web コンテンツ ¥FashionFlow\_name (Web Content¥FashionFlow\_name)」サブフォルダーに移動します。
  - c. **MyNewJSPTemplate\_All.jsp** ファイルと **MyNewJSPTemplate.jsp** ファイルを両方とも強調表示し、右クリックして、「オープンに使用 (Open With) > 「Page Designer」を選択します。
2. 「Section 7」を MyNewJSPTemplate\_All.jsp ファイルから MyNewJSPTemplate.jsp ファイルにコピーします。これによって、以下のテキストが JSP テンプレートに導入されます。

```

<!-- SECTION 7 -->

<fmt:message key="Greeting" bundle="${tutorial}" />
<c:out value="${taskOutputGreetings}" />

<!-- END OF SECTION 7 -->

```

3. 変更内容を保管します。

## MyNewTaskCmd のテスト

次のステップとして、以下のようにして、新規タスク・コマンドが適切に作動するかどうかをテストします。

1. サーバー・パースペクティブに切り替えます (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」)。
2. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」 (または「再始動 (Restart)」) を選択します。
3. Stores¥Web Content¥FashionFlow\_name ディレクトリーの下での **index.jsp** を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。  
Web ブラウザー中にストアのホーム・ページが表示されます。
4. 次に、以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
input1=abc&input2=1000
```

MyNewJSPTemplate が表示されます。タスク・コマンドで作成されたグリーティング・メッセージが含まれます。

# Programmer's Guide

## Tutorial: Creating new business logic

### List of parameter-value pairs sent from the controller command

```
ControllerParm1= Hello from IBM!
ControllerParm2= Have a nice day!
```

This is an example of using the tag from JSP Standard Tag Library (JSTL)

MyNewView was called by a controller command

MyNewView was called by the controller command which is -

**com.ibm.commerce.sample.commands.MyNewControllerCmdImpl**

```
UserName= abc
Points= 1000
Greeting= Hello ! abc
```

図 39.

## MyNewTaskCmd の変更

チュートリアルはこのステップでは、MyNewTaskCmd に変更を加えて、URL に含まれるユーザー名が登録済みユーザーの名前かどうかを判別するようにします。また、MyNewDataBean にも変更を加えて、タスク・コマンド中のフィールド (ユーザー名のフィールドなど) を処理するようにします。さらに、MyNewJSPTemplate に変更を加えて、ユーザーが登録済みかどうかを表示します。

チュートリアルはこのセクションでは、以下の点を学習します。

- URL パラメーターを使用して、独自のカスタマイズ・コード内から既存の WebSphere Commerce 情報にアクセスする方法

## タスク・コマンド用のオブジェクトを作成するための MyNewControllerCmdImpl の変更

オブジェクトの効率的な使用を促進するため、コントローラー・コマンドは、UserRegistryAccessBean オブジェクト・インスタンス変数を作成します。その結果、このオブジェクトはタスク・コマンドでも使用できるようになります。この方法を採用することで、タスク・コマンドがオブジェクトのインスタンスを別個に作成する必要はなくなります。この UserRegistryAccessBean オブジェクトを後で (タスク・コマンド内で) 使用し、ショッパーが登録ユーザーかどうかを判別します。

MyNewControllerCmdImpl を変更するには、以下のようになります。

1. Java パースペクティブで、**MyNewControllerCmdImpl.java** クラスをダブルクリックして、そのソース・コードを表示します。
2. このクラスの本体の中で、「Section 2」のコメントを解除して、以下のコードをクラスに導入します。

```
/// Section 2 //////////////////////////////////////
/// create a user registry accessbean resource instance variable
```

```
private UserRegistryAccessBean rrb = null;
```

```
/// End of Section 2 //////////////////////////////////////
```

3. 「概略 (Outline)」ビューで、**performExecute** メソッドを選択します。
4. performExecute メソッド用のソース・コード内で、「Section 4D」および「Section 4F」をコメント解除し、インスタンス変数をタスク・コマンドに渡してから、戻されるユーザー ID を応答プロパティの中で使用できるようにします。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 4D //////////////////////////////////////
/// pass rrb instance variable to the task command
```

```
cmd.setUserRegistryAccessBean(rrb);
```

```
// End of section 4D //////////////////////////////////////
```

```
// Section 4F //////////////////////////////////////
///using access bean to get information from database
if (cmd.getFoundUserId() != null) {
 rspProp.put("taskOutputUserId", cmd.getFoundUserId());
}
```

```
// End of section 4F //////////////////////////////////////
```

いくつかのメソッドが未定義であることを示すエラーを受け取りますが、このエラーは次のステップでタスク・コマンドに変更を加える際に解決します。

5. 作業内容を保管します。

## ユーザー名妥当性検査のための新規タスク・コマンドの変更

次に、以下のようにして新規タスク・コマンドに変更を加え、URL に入力されたユーザー名が登録したユーザーのユーザー名かどうかを検査しなければなりません。

1. **MyNewTaskCmd.java** インターフェースをダブルクリックして、そのソース・コードを表示します。
2. 「Section 3」のコメントを解除して、以下のコードをインターフェースに導入します。

```
/// Section 3 //

public void setFoundUserId(java.lang.String inputUserId);
public java.lang.String getFoundUserId();

public void setUserRegistryAccessBean(UserRegistryAccessBean rrb);

/// End of section 3////////////////////////////////////
```

3. 作業内容を保管します。
4. **MyNewTaskCmdImpl.java** クラスをダブルクリックして、そのソース・コードを表示します。「Import section 1」をコメント解除して、以下の 2 つのインポート・ステートメントをコードに導入します

```
/// Import section 1 //
import com.ibm.commerce.user.objects.*;
import com.ibm.commerce.sample.databeans.*;
/// End of Import section 1 //////////////////////////////////////
```

5. 「Section 2A」および「Section 2B」をコメント解除して、インターフェース中に追加するメソッドに対応するフィールドおよび getter メソッドと setter メソッドを新規作成します。これによって、以下のコードがクラスに導入されます。

```
//// Section 2A //

private java.lang.String foundUserId = null;

private UserRegistryAccessBean rrb = null;

////End of Section 2A //

//// Section 2B //

public void setUserRegistryAccessBean(UserRegistryAccessBean newRRB) {
 rrb = newRRB;
}

public void setFoundUserId(java.lang.String newFoundUserId) {
 foundUserId = newFoundUserId;
}

public java.lang.String getFoundUserId() {
 return foundUserId;
}
```



```
/// End of section 2B //////////////////////////////////////
```

6. 「概略 (Outline)」ビューで、**validateParameters** メソッドを選択し、そのソース・コードを調べます。「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。

```
// section 1 //////////////////////////////////////
```

```
// use UserRegistryAccessBean to check user Id
```

```
try {
 if (rrb!=null){
 setFoundUserId(rrb.getUserId());
 } else {
 rrb =new UserRegistryAccessBean();
 rrb=rrb.findByUserLogonId(getInputUserName());
 setFoundUserId(rrb.getUserId());
 }
}

} catch (javax.ejb.FinderException e) {
 return;
}

} catch (java.rmi.RemoteException e) {
 throw new ECSYSTEMException(ECMESSAGE._ERR_REMOTE_EXCEPTION,
 this.getClass().getName(), "validateParameters");
} catch (javax.naming.NamingException e) {
 throw new ECSYSTEMException(ECMESSAGE._ERR_NAMING_EXCEPTION,
 this.getClass().getName(), "validateParameters");
} catch (javax.ejb.CreateException e) {
 throw new ECSYSTEMException(ECMESSAGE._ERR_CREATE_EXCEPTION,
 this.getClass().getName(), "validateParameters");
}
}
```

```
// end of section 1 //////////////////////////////////////
```

7. 作業内容を保管します。
8. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックし、「プロジェクトの構築 (Build Project)」を選択して、コードの変更をコンパイルします。

## ユーザー名妥当性検査のための MyNewJSPTemplate の変更

ユーザー名妥当性検査情報を表示するためには、現在の JSP テンプレートを変更する必要があります。このファイルに変更を加えるには、以下のようになります。

1. Web パースペクティブに切り替えます。
2. **MyNewJSPTemplate\_All.jsp** ファイルと **MyNewJSPTemplate.jsp** ファイルを両方ともオープンします。

- 「Section 8」を MyNewJSPTemplate\_All.jsp ファイルから MyNewJSPTemplate.jsp ファイルにコピーします。これによって、以下のテキストが JSP テンプレートに導入されます。

```
<!-- SECTION 8 -->

<c:if test="${!empty taskOutputUserId}">
 <fmt:message key="UserId" bundle="${tutorial}" />
 <c:out value="${taskOutputUserId}"/>

 <fmt:message key="FirstInput" bundle="${tutorial}" />
 <c:out value="${userName}"/>
 <fmt:message key="RegisteredUser" bundle="${tutorial}" />

 <fmt:message key="ReferenceNumber" bundle="${tutorial}" />
 <c:out value="${taskOutputUserId}"/>

</c:if>

<c:if test="${empty taskOutputUserId}">
 <fmt:message key="FirstInput" bundle="${tutorial}" />
 <c:out value="${userName}"/>
 <fmt:message key="NotRegisteredUser" bundle="${tutorial}" />

</c:if>

<!-- END OF SECTION 8 -->
```

- MyNewJSPTemplate.jsp ファイルに加えた変更内容を保管します。

## ユーザー名妥当性検査のテスト

ユーザー名妥当性検査ロジックをテストするには、以下のようにします。

- サーバー・パースペクティブに切り替えます (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」)。
- WebSphereCommerceServer** サーバーを右クリックして、「スタート」 (または「再始動 (Restart)」) を選択します。
- Stores¥Web Content¥FashionFlow\_name ディレクトリーの下に **index.jsp** を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。  
Web ブラウザー中にストアのホーム・ページが表示されます。
- 以下のようにして、新しい登録ユーザーを作成します。
  - 「登録」をクリックします。
  - もう一度「登録」をクリックして、新しい顧客を作成します。
  - 登録フォームで、すべての必須フィールドに適切な値を入力します。たとえば、E メール・フィールドには、tester@mycompany と入力します。E メール・アドレスの値を記録しておいてください: \_\_\_\_\_。
  - 値を入力したら、「送信」をクリックします。
- 次に、input1 の値として有効なユーザー名を入力します。以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
input1=new_e-mail&input2=1000
```

ここで *new\_e-mail* は、ステップ 4 で作成したユーザーの E メール・アドレスです。MyNewJSPTemplate が表示されます。今度は、input1 の値が有効なユーザー名であることが示されているはずですが。

# Programmer's Guide

## Tutorial: Creating new business logic

### List of parameter-value pairs sent from the controller command

```
ControllerParm1= Hello from IBM!
ControllerParm2= Have a nice day!
```

This is an example of using the tag from JSP Standard Tag Library (JSTL)

MyNewView was called by a controller command  
MyNewView was called by the controller command which is -  
**com.ibm.commerce.sample.commands.MyNewControllerCmdImpl**

```
UserName= tester@mycompany
Points= 1000
Greeting= Hello ! tester@mycompany

UserId= 1002
Your first input parameter is a registered user
The member reference number of this user is 1002
```

図 40.

- 次に、ユーザー名の値が無効になる、以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
input1=abc&input2=1000
```

汎用例外が表示されます。ページのソースを見ている場合、`_ERR_FINDER_EXCEPTION` の生じていることが分かります。これは、指定されたユーザー名が登録されたユーザーに対応していないことが理由です。

---

## 新規 Entity Bean の作成

このセクションでは、新規 Entity 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・テーブルを作成する必要があります。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 Entity Bean を作成する必要があります。

### XBONUS テーブルの作成

Entity 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 XBONUS 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」>「コマンド行ツール (Command Line Tools)」>「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 developmentDB user dbuser using dbpassword;
create table XBONUS (MEMBERID BIGINT NOT NULL,
 BONUSPOINT INTEGER NOT NULL,
 constraint p_xbonus primary key (MEMBERID),
 constraint f_xbonus foreign key (MEMBERID)
 references users (users_id) on delete cascade)
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード

「実行」アイコンをクリックします。

SQL ステートメントが正常に完了したことを示すメッセージが表示されます。

▶ **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 XBONUS (MEMBERID NUMBER NOT NULL,
 BONUSPOINT INTEGER NOT NULL,
 constraint p_xbonus primary key (MEMBERID),
 constraint f_xbonus 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。XBONUS テーブルが作成されました。

6. データベースの変更をコミットするには、以下のように入力します。

```
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 Entity Bean の作成

テーブルが作成されたら、新規 Entity Bean の作成を始めることができます。次のステップでは WebSphere Studio Application Developer を使用してこの Bean を作成します。



次に以下のようにして、Bonus Bean を新規作成します。

1. WebSphere Studio Application Developer で、J2EE パースペクティブに切り替えます。
2. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、「EJB モジュール (EJB Modules)」を拡張表示します。
3. **WebSphereCommerceServerExtensionsData** モジュールを右クリックして、「新規」 > (「その他」 >) 「Enterprise Bean」を選択します。  
「Enterprise Bean の作成 (Enterprise Bean Creation)」ウィザードがオープンします。
4. 「EJB プロジェクト (EJB Project)」ドロップダウン・リストで、「WebSphereCommerceServerExtensionsData」を選択して、「次へ」をクリックします。
5. 「Enterprise Bean の作成」ウィンドウで、以下のようにします。
  - a. 「コンテナ管理パーシスタンス (CMP) フィールドを持つ Entity Bean」を選択します。
  - b. 「Bean 名 (Bean name)」フィールドに、Bonus と入力します。

- c. 「ソース・フォルダー (Source folder)」フィールドを、指定されているデフォルト値 (ejbModule) のままにします。
  - d. 「デフォルト・パッケージ (Default package)」フィールドに、`com.ibm.commerce.extension.objects` と入力します。
  - e. 「次へ」をクリックします。
6. 「Enterprise Bean の詳細 (Enterprise Bean Details)」ウィンドウで、以下のようになります。
- a. 「追加」をクリックして、BONUS テーブル中の MEMBERID 列と BONUSPOINT 列の新規 CMP 属性を追加します。  
「CMP 属性の作成 (Create CMP Attribute)」ウィンドウがオープンします。このウィンドウで、以下のようになります。
    - 1) 「名前」フィールドで、`memberId` と入力します。
    - 2) 「タイプ」フィールドで、`java.lang.Long` と入力します。  
  
注: `long` データ・タイプではなく、`java.lang.Long` データ・タイプを使用してください。
    - 3) 「鍵フィールド」チェック・ボックスを選択します。
    - 4) 「適用」をクリックします。
    - 5) 「名前」フィールドで、`bonusPoint` と入力します。
    - 6) 「タイプ」フィールドで、`java.lang.Integer` と入力します。  
  
注: `integer` データ・タイプではなく、`java.lang.Integer` データ・タイプを使用してください。
    - 7) 「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。
    - 8) 「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」チェック・ボックスをクリアします。「getter を読み取り専用にする (Make getter read-only)」チェック・ボックスが使用不能になります。
    - 9) 「適用」をクリックします。
    - 10) 「クローズ」をクリックしてこのウィンドウをクローズします。
  - b. 「鍵クラスに単一の鍵属性タイプを使用 (Use the single key attribute type for the key class)」チェック・ボックスをクリアしてから、「次へ」をクリックします。
7. 「EJB Java クラスの詳細 (EJB Java Class Details)」ウィンドウで、以下のようになります。
- a. Bean のスーパークラスを選択するには、「ブラウズ」をクリックします。  
「タイプの選択 (Type Selection)」ウィンドウがオープンします。

- b. 「使用するクラスの選択: (任意) (Select a class using: (any))」フィールドに、 `ECEntityBean` と入力し、「OK」をクリックします。スーパークラスとして `com.ibm.commerce.base.objects.ECEntityBean` が選択されます。
- c. 「追加」をクリックして、リモート・インターフェースが拡張する必要があるインターフェースを指定します。「タイプの選択 (Type Selection)」ウィンドウがオープンします。
- d. 「使用するクラスの選択: (任意) (Select a class using: (any))」フィールドに、 `Protectable` と入力し、「OK」をクリックします。  
`com.ibm.commerce.security.Protectable` が選択されます。アクセス制御下の新規リソースを保護するには、このインターフェースが必要です。
- e. 「終了」をクリックします。

新規 Bean の分離レベルを設定するには、以下のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示します。
2. **WebSphereCommerceServerExtensionsData** プロジェクトをダブルクリックし、**Deployment Descriptor Editor** を使用してオープンします。(このエディターでこのファイルを開く代替方法として、以下のようにすることもできます。
  - a. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、  
「WebSphereCommerceServerExtensionsData」 > 「ejbModule」 > 「META-INF」を拡張表示します。
  - b. **ejb-jar.xml** を右クリックして、「オープンに使用 (Open With)」 > 「Deployment Descriptor Editor」を選択します。)
3. 「アクセス」タブをクリックします。
4. 「分離レベル (Isolation Level)」テキスト・ボックスの隣の「追加」をクリックします。  
「分離レベルの追加 (Add Isolation Level)」ウィンドウがオープンします。
5.  「反復可能読み取り (Repeatable Read)」を選択してから、「次へ」をクリックします。
6.  「コミット済み読み取り (Read Committed)」を選択してから、「次へ」をクリックします。
6. 「検出された Bean (Beans found)」リストから **Bonus** Bean を選択してから、「次へ」をクリックします。
7. 「検出されたメソッド (Methods found)」リストから、「Bonus」を選択してそのメソッドをすべて選択し、「終了」をクリックします。
8. ここまでの作業を保管し (Ctrl + S)、エディターはオープンしたままにします。

次に、以下のようにして、Bean のセキュリティー ID を設定します。

1. Deployment Descriptor Editor で、「アクセス」タブを選択してあることを確認します。
2. 「セキュリティー ID (Security Identity)」テキスト・ボックスの隣の「追加」をクリックします。  
「セキュリティー ID の追加 (Add Security Identity)」ウィンドウがオープンします。
3. 「EJB サーバーの ID の使用 (Use identity of EJB server)」を選択してから、「次へ」をクリックします。
4. 「検出された Bean (Beans found)」リストから **Bonus** Bean を選択してから、「次へ」をクリックします。
5. 「検出されたメソッド (Methods found)」リストから、「**Bonus**」を選択してそのメソッドをすべて選択し、「終了」をクリックします。
6. ここまでの作業を保管し (Ctrl + S)、エディターはオープンしたままにします。

次に、以下のようにして、Bean 内のメソッドのセキュリティー役割を設定します。

1. Deployment Descriptor Editor で、「アセンブリー記述子 (Assembly Descriptor)」タブを選択します。
2. 「メソッド許可 (Method Permissions)」セクションで、「追加」をクリックします。
3. セキュリティー役割として **WCSecurityRole** を選択して、「次へ」をクリックします。
4. 見つかった Bean のリストから **Bonus** を選択し、「次へ」をクリックします。
5. 「メソッド・エレメント (Method elements)」ページで、「すべてに適用 (Apply to All)」をクリックし、「終了」をクリックします。
6. ここまでの作業を保管し (Ctrl + S)、Deployment Descriptor Editor をクローズします。

次のステップとして、WebSphere Studio Application Developer が生成するエンティティ・コンテキストに関連したフィールドとメソッドの一部を除去します。これらのフィールドを削除する必要のある理由は、EJEntityBean の基本クラスにはこれらのメソッドの独自のインプリメンテーションがあるからです。生成されたエンティティ・コンテキストのフィールドとメソッドを削除するには、以下のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、**WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
2. **Bonus** Bean を拡張表示してから、**BonusBean** クラスをダブルクリックします。
3. 「概略 (Outline)」ビューで、以下のようにします。
  - a. 「**myEntityCtx**」フィールドを右クリックし、「削除」を選択します。
  - b. **getEntityContext()** メソッドを右クリックし、「削除」を選択します。
  - c. **setEntityContext(EntityContext)** メソッドを右クリックし、「削除」を選択します。



- d. **unsetEntityContext()** メソッドを右クリックし、「削除」を選択します。
4. ここまでの作業を保管します (Ctrl + S)。BonusBean クラスはオープンしたままにしておきます。

次に以下のようにして、Enterprise Bean に新しい `getMemberId` メソッドを追加します。

1. **BonusBean** クラスのソース・コードを表示します。
2. このクラスの末尾 (ただしクラス中) に以下のコードを追加します。

```
public java.lang.Long getMemberId() {
 return memberId;
}
```

3. 以下のようにして、リモート・インターフェースに新しいメソッドを追加する必要があります。
  - a. 「概略 (Outline)」ビューで **getMemberId** メソッドを右クリックし、「Enterprise Bean」>「リモート・インターフェースへのプロモート (Promote to Remote Interface)」を選択します。この操作を完了すると、メソッドの隣に小さな R アイコンが表示され、リモート・インターフェースにプロモートされていることが示されます。
4. 作業内容を保管します。
5. BonusBean エディターをクローズします。

次に、以下のようにして、BonusHome インターフェースに新しい FinderHelper メソッドを追加します。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示します。
2. **WebSphereCommerceServerExtensionsData** プロジェクトをダブルクリックし、EJB Deployment Descriptor Editor をオープンします。
3. 「Bean」タブをクリックします。
4. 「Bean」ペインで、**Bonus Bean** を選択してから、このペインの右側でスクロールダウンして、「WebSphere 拡張版 (WebSphere Extensions)」を拡張表示します。
5. 「ファインダー (Finder)」テキスト・ボックスの隣の「追加」をクリックします。「ファインダー記述子の追加 (Add Finder Descriptor)」ウィンドウがオープンします。
6. 「新規」を選択してから、「名前」フィールドで、`findByMemberId` と入力します。
7. 「パラメーター」テキスト・ボックスの隣の「追加」をクリックしてから、以下のようにします。
  - a. 「名前」フィールドで、`memberId` と入力します。
  - b. 「タイプ」フィールドで、`java.lang.Long` と入力します。

- c. 「OK」をクリックします。
8. 「戻りタイプ」ドロップダウン・リストから、**com.ibm.commerce.extension.objects.Bonus** を選択し、「次へ」をクリックします。
9. 「ファインダーのタイプ (Finder type)」ドロップダウン・リストで、「WhereClauseFinderDescriptor」を選択します。
10. 「ファインダーのステートメント (Finder statement)」フィールドで、T1.MEMBERID = ? と入力してから、「終了」をクリックします。
11. ここまでの作業を保管してから、EJB Deployment Descriptor Editor をクローズします。

次に以下のようにして、Bonus Bean に新規 ejbCreate メソッドを追加します。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、**BonusBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 以下のコードをクラスに追加して、`ejbCreate(Long, Integer)` メソッドを新規作成します。

```
public com.ibm.commerce.extension.objects.BonusKey ejbCreate(
 java.lang.Long memberId,java.lang.Integer bonusPoint)
 throws javax.ejb.CreateException {
 _initLinks();
 this.memberId=memberId;
 this.bonusPoint=bonusPoint;
 return null;
}
```

3. コードの変更内容を保管します。
4. ホーム・インターフェースに新規 `ejbCreate(Long, Integer)` メソッドを追加しなければなりません。これにより、生成された Access Bean でメソッドを使用できるようになります。ホーム・インターフェースにメソッドを追加するには、以下のようにします。
  - a. 「概略 (Outline)」ビューで **ejbCreate(Long, Integer)** メソッドを右クリックし、「Enterprise Bean」>「ホーム・インターフェースへのプロモート (Promote to Home Interface)」を選択します。

次に、以下のようにして、`ejbPostCreate(Long, Integer)` メソッドを新規作成し、`ejbCreate(Long, Integer)` メソッドと同じパラメーターになるようにします。

1. **BonusBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 以下のコードをクラスに追加して、`ejbPostCreate(Long, Integer)` メソッドを新規作成します。

```

public void ejbPostCreate(java.lang.Long memberId,
 java.lang.Integer bonusPoint)
 throws javax.ejb.CreateException
 {
 }

```

3. コードの変更内容を保管します。

次のステップは、新規メソッドを Bonus Bean に追加して、WebSphere Commerce アクセス制御システムで保護できるようにすることです。以下のようにして、getOwner メソッドと fulfills メソッドを Bean に追加しなければなりません。

1. **BonusBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 以下のコードをクラスの末尾に追加して、新しい getOwner メソッドを BonusBean クラスに追加します。

```

public java.lang.Long getOwner()
 throws java.lang.Exception {
 return getMemberId();
}

```

3. 作業内容を保管します。
4. 以下のコードをクラスの末尾に追加して、新しい fulfills メソッドを追加します。

```






public boolean fulfills(Long member, String relationship)
 throws java.lang.Exception {
 if (relationship.equalsIgnoreCase("creator"))
 {
 return member.equals(getMemberId());
 }
 return false;
}

```

5. 作業内容を保管し、Bonus Bean エディターをクローズします。

次のステップとして、XBONUS テーブルを BonusBean Entity Bean にマップします。Meet-in-the-middle マッピングが使用されます。マッピングを作成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、**WebSphereCommerceServerExtensionsData** を右クリックし、「生成 (Generate)」>「EJB 対 RDB のマッピング (EJB to RDB Mapping)」を選択します。  
「EJB 対 RDB のマッピング(EJB to RDB Mapping)」ウィンドウがオープンします。
2. 「meet-in-the-middle (Meet In The Middle)」を選択し、「次へ」をクリックします。
3. 「データベース接続」ウィンドウで、以下のようになります。
  - a. 「接続名 (Connection name)」フィールドに、WebSphereCommerceServerExtensionsData と入力します。

- b. 「データベース」フィールドに、開発データベースの名前を入力します。
  - c. 「ユーザー ID」フィールドに、データベース・ユーザー ID を入力します。
  - d. 「パスワード」フィールドに、データベース・ユーザーのパスワードを入力します。
  - e. 「データベース・メーカー・タイプ (Database vendor type)」ドロップダウン・リストから、ご使用の開発データベースのデータベース・メーカー・タイプを選択します。
    -  DB2 Universal Database 8.1
    -  Oracle 9i
  - f.  「ホスト」フィールドに、データベース・サーバーの完全修飾ホスト名を入力します。たとえば、dbserver.yourcompany.com と入力します。
  - g.  「クラスの場所 (Class Location)」フィールドに、 classes12.zip ファイルの場所を入力します。たとえば D:\oracle\ora92\jdbc\lib\classes12.zip と入力します。
  - h. 「次へ」をクリックします。接続を確立し終わると、データベース中のテーブルのリストが表示されます。また、「データ」パースペクティブの「データベース・サーバー」ビューを確認すると、後から接続文書を確認できます。
4. **XBONUS** テーブルを選択して、「次へ」をクリックします。
5. 「名前およびタイプ別に突き合わせ (Match By Name and Type)」を選択してから、「終了」をクリックします。これで、Mapping Editor がオープンします。
6.  **XBONUS** テーブルを右クリックし、「テーブル・エディターのオープン (Open Table Editor)」を選択します。テーブル・エディターで、以下のようになります。
- a. 「列 (Column)」タブを選択します。
  - b. BONUSPOINT 列を選択し、列タイプを NUMBER から INTEGER に変更します。
  - c. 変更内容を保管します。
7. 「Enterprise Bean」ペインで、**Bonus** Bean を拡張表示します。「テーブル (Tables)」ペインで、**XBONUS** テーブルを拡張表示します。
8. 以下のようにして、Bonus Bean 中のフィールドを、XBONUS テーブル中の列にマップします。
- a. Bonus Bean を右クリックして、「名前別に突き合わせ (Match By Name)」を選択します。
9. Ctrl + S を押して Map.mapxmi ファイルを保管します。ファイルをクローズします。

10. **Oracle** テキスト・エディターを使用し、テーブル定義を以下のように編集する必要があります。
  - a. テキスト・エディターで XBONUS.xmi ファイルをオープンします。
  - b. SQLNumeric6 が出現するすべての箇所を、SQLNumeric3 に置き換えます。
  - c. 変更内容を保管します。

次のステップとして、スキーマ名を修正し、新規 Bean を他のデータベースに移植できるようにします。この変更を行うには、以下のようにします。

1. J2EE パースペクティブで、「J2EE 階層 (J2EE Hierarchy)」ビューに切り替えます。
2. 「データベース (Databases)」を拡張表示してから、「**WebSphereCommerceServerExtensionsData**」を拡張表示します。
3. スキーマ・ノード (DB2USER など) を右クリックし、「名前変更」を選択します。
4. 値を NULLID に設定します。

BonusBean エンティティが作成され、スキーマが正しくマップされたら、Entity Bean 用の Access Bean を作成する必要があります。この Access Bean は、Bonus エンティティ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 Entity Bean に基づいて、この Access Bean を生成するために、WebSphere Studio Application Developer 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、Access Bean によって使用されることとなります)。Bonus Entity Bean 用の Access Bean を作成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示してから、**WebSphereCommerceServerExtensionsData** を右クリックし、「新規」>「Access Bean」を選択します。  
「Access Bean の追加 (Add an Access Bean)」ウィンドウがオープンします。
2. 「コピー・ヘルパー (Copy Helper)」を選択して、「次へ」をクリックします。
3. **Bonus** Bean を選択して、「次へ」をクリックします。
4. コンストラクター・メソッドのドロップダウン・リストから、コンストラクター・メソッドとして **findByPrimaryKey(com.ibm.commerce.extension.objects.BonusKey)** を選択します。
5. 「属性ヘルパー (Attribute Helpers)」セクションで、すべての属性を選択します。
6. 「終了」をクリックします。

「J2EE ナビゲーター (J2EE Navigator)」タブに切り替えてから、「**WebSphereCommerceServerExtensionsData**」>「**ejbModule**」>「**com.ibm.commerce.extension.objects**」を拡張表示して、新しく生成したコードを表示できます。

BonusAccessBean という新しいクラスと BonusAccessBeanData という新しいインターフェースが作成され、パッケージ内に表示されます。

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 サーバーの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 Enterprise 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 アクセスが必要とされる)、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。

デプロイメント・コードを生成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示し、**WebSphereCommerceServerExtensionsData** を右クリックし、「生成 (Generate)」 > 「デプロイメントおよび RMI コード (Deploy and RMI Code)」を選択します。
2. **Bonus Bean** を選択して、「終了」をクリックします。

「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えると、新しく生成したコードを表示できます。以下のように表示されます。

表 11.

コードのタイプ	クラス名
コンテナ・インプリメンテーション生成コード	EJSCMPBonusHomeBean.java
	EJSRemoteCMPBonus.java
	EJSRemoteCMPBonusHome.java
	EJSFinderBonusBean.java
JDBC アクセス・コード	EJSJDBCPersisterCMPBonusBean.java
RMI タイおよびスタブ・コード	_EJSRemoteCMPBonus_Tie.java
	_Bonus_Stub.java
	_EJSRemoteCMPBonusHome_Tie.java
	_BonusHome_Stub.java

次のステップとして、以下のようにして、汎用テスト・クライアントを使用して新規 Enterprise Bean をテストします。

1. サーバー・パースペクティブに切り替えます。

2. 「サーバー構成 (Server Configuration)」ビューで、 **WebSphereCommerceServer** サーバーをダブルクリックし、「構成」タブをクリックします。
3. 「汎用テスト・クライアントの使用可能化 (Enable universal test client)」を選択します。変更内容を保管します。
4. 「サーバー (Servers)」ビューで、 **WebSphereCommerceServer** サーバーを右クリックして、「スタート」を選択します。
5. 「J2EE 階層 (J2EE Hierarchy)」で、「EJB モジュール (EJB Modules)」 > 「WebSphereCommerceServerExtensionsData」を拡張表示します。
6. **Bonus Bean** を右クリックし、「サーバー上で実行 (Run on Server)」を選択します。  
IBM 汎用テスト・クライアントがオープンします。
7. 左側のペインで、「**Bonus**」をクリックし、「**Bonus Home**」をクリックします。
8. **Bonus create(Long, Integer)** メソッドをクリックします。
9. 右側のペインの、「**Long**」フィールドに -1000 と入力し、「**Integer**」フィールドに 1000 と入力します。
10. 「呼び出し (Invoke)」をクリックすると、下部のペインに結果が表示されます。
11. 「オブジェクトの処理 (Work with Object)」をクリックして、リモート・インターフェースを「参照 (Reference)」ペインに追加すると、「EJB 参照 (EJB References)」の下に入力した値が表示されます。 **BONUS** テーブル中にレコードが新規作成されています。
12. **getMemberId** メソッドを選択し、「呼び出し (Invoke)」をクリックすると、下部のペインに結果の -1000 が表示されます。
13. テスト・クライアントをクローズして、サーバーを停止します。

## Bonus エンティティと MyNewControllerCmd の統合

ここまでのセクションでは、WebSphere Studio Application Developer 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 Entity Bean** をテストしてきました。この操作により、データベース情報を正常に更新できたことを判別しました。次に、**Bonus Entity 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、**MyNewJSPTemplate.jsp** ファイルが更新されて、ボーナス・ポイントの顧客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 Entity 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. ボーナス・ポイントのフィールドとメソッドを組み込むよう **MyNewTaskCmd** タスク・コマンドを変更し、**validateParameters** メソッドを更新し、ユーザーのボーナス・ポイントのバランスを更新するロジックを追加します。
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 `BonusDataBean` を作成し、ボーナス・ポイントを JSP テンプレートに表示できるようにします。
4. 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。
5. `MyNewJSPTemplate.jsp` テンプレートに変更を加えて、ユーザーのボーナス・ポイントを入力してからそのユーザーの新しいボーナス・ポイントのバランスを表示できるようにします。

### ボーナス・ポイントを組み込むように `MyNewTaskCmd` インターフェースに変更を加える

このステップでは、`MyNewTaskCmd` interface ファイルに変更を加えて、ボーナス・ポイントの必要フィールドとメソッドを指定します。そのためには、以下のようになります。

1. Java パースペクティブに切り替え、  
**WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。
2. **com.ibm.commerce.sample.commands¥src** ディレクトリーを拡張表示します。
3. **MyNewTaskCmd** インターフェースをダブルクリックして、そのソース・コードを表示します。
4. 「Import section 2」をコメント解除して、以下のパッケージを組み込みます。

```
/// Import section 2 //////////////////////////////////////
import com.ibm.commerce.extension.objects.*;
/// End of import section 2 //////////////////////////////////////
```

5. 「Section 4」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。

```
/// Section 4 //////////////////////////////////////

public java.lang.Integer getOldBonusPoints();
public Integer getTotalBonusPoints();

public void setBonusAccessBean(BonusAccessBean bb);
public BonusAccessBean getBonusAccessBean();

/// End of section 4////////////////////////////////////
```

6. 変更内容を保管します。

### ボーナス・ポイントを計算するように `MyNewTaskCmdImpl` に変更を加える

`MyNewTaskCmdImpl` は、`Bonus Entity Bean` と `MyNewControllerCmd` との間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・ポイントを計算するように `MyNewTaskCmdImpl` に変更を加えるには、以下のようになります。

1. **MyNewTaskCmdImpl** クラスを選択して、そのソース・コードを表示します。
2. 「Import section 2」をコメント解除して、以下のパッケージを導入します。



```

/// Import section 2 //////////////////////////////////////
import com.ibm.commerce.extension.objects.*;
/// End of Import section 2 //////////////////////////////////////

```

3. 「Section 3A」と「Section 3B」のコメントを解除して、以下のコードをクラスに導入します。

```

//// Section 3A //////////////////////////////////////

private java.lang.Integer oldBonusPoints;
private java.lang.Integer totalBonusPoints;

private BonusAccessBean bb = null;

////End of Section 3A //////////////////////////////////////

//// Section 3B //////////////////////////////////////

public void setBonusAccessBean(BonusAccessBean newBB) {
 bb = newBB;
}

public BonusAccessBean getBonusAccessBean(){
 return bb;
}

 public java.lang.Integer getOldBonusPoints() {
 return oldBonusPoints;
 }

public Integer getTotalBonusPoints(){
 return totalBonusPoints;
}

/// End of section 3B //////////////////////////////////////

```

4. 「概略 (Outline)」ビューで、**validateParameters** メソッドを選択し、「Section 2」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。

```

// section 2 //////////////////////////////////////

try {
 oldBonusPoints = bb.getBonusPoint();
} catch (javax.ejb.FinderException e) {
 try {
 // If bb is null, create a new instance
 bb = new BonusAccessBean(new Long(foundUserId), new Integer(0));
 oldBonusPoints = new Integer(0);
 } catch (javax.ejb.CreateException ec) {
 throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
 this.getClass().getName(), "validateParameters");
 } catch (javax.naming.NamingException ec) {
 throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
 this.getClass().getName(), "validateParameters");
 }
}

```

```

 } catch (java.rmi.RemoteException ec) {
 throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
 this.getClass().getName(), "validateParameters");
 }
 } catch (javax.naming.NamingException e) {
 throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
 this.getClass().getName(), "validateParameters");
 }
 } catch (java.rmi.RemoteException e) {
 throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
 this.getClass().getName(), "validateParameters");
 }
 } catch (javax.ejb.CreateException e) {
 throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
 this.getClass().getName(), "validateParameters");
 }
}

```

// end of section 2 //////////////////////////////////////

5. 「概略 (Outline)」ビューで、**performExecute** メソッドを選択します。
6. この 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。

```

/// use BonusAccessBean to update new bonus point
/// Section 2 //////////////////////////////////////

int newBP = oldBonusPoints.intValue() + getInputPoints().intValue();
totalBonusPoints = new Integer (newBP);
bb.setBonusPoint(totalBonusPoints) ;

try {
 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");
}

/// End of section 2 //////////////////////////////////////

```

7. 変更内容を保管します。

### 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 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. **MyNewControllerCmdImpl** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. ソース・コードの中で「Import Section 2」をコメント解除して、以下のパッケージをクラスに導入します。

```
/// Import Section 2 //////////////////////////////////////
import com.ibm.commerce.extension.objects.*;
/// End of Import Section 2 //////////////////////////////////////
```

3. 「Section 3」のコメントを解除して、以下のコードをクラスに導入します。

```
/// Section 3 //////////////////////////////////////
/// Create an instance variable of type AccessVector to hold
/// the resources and a BonusAccessBean instance variable for
/// access control purposes.
```

```
private AccessVector resources = null;
private BonusAccessBean bb = null;
```

```
/// End of Section 3 //////////////////////////////////////
```

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。

```
/// AccessControl Section //////////////////////////////////////
```

```
public AccessVector getResources() throws ECException{

 if (resources == null) {

 /// use UserRegistryAccessBean to check user reference number

 String refNum = null;
 String methodName = "getResources";

 rrb = new UserRegistryAccessBean();

 try {
 rrb = rrb.findByUserLogonId(getUserName());
 refNum = rrb.getUserId();
 } catch (javax.ejb.FinderException e) {
 throw new ECSystemException(ECMessage._ERR_FINDER_EXCEPTION,
 this.getClass().getName(),methodName,e);
 } catch (javax.naming.NamingException e) {
 throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
 this.getClass().getName(), methodName,e);
 } catch (java.rmi.RemoteException e) {
 throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
 this.getClass().getName(), methodName,e);
 } catch (javax.ejb.CreateException e) {
 throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
 this.getClass().getName(), methodName,e);
 }
 }
}
```

```

 /// find the bonus bean for this registered user

 bb = new com.ibm.commerce.extension.objects.BonusAccessBean();
 try {
 if (refNum != null) {
 bb.setInitKey_memberId(new Long(refNum));
 bb.refreshCopyHelper();
 resources = new AccessVector(bb);
 }
 } catch (javax.ejb.FinderException e) {

 ///doesn't have a bonus object so return the container that
 ///will hold the bonus object when it's created
 resources = new AccessVector(rrb);
 return resources;

 } 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 resources;
}

/// End of AccessControl Section //////////////////////////////////////

```

5. 変更内容を保管します。

**MyNewControllerCmdImpl クラスの performExecute メソッドの変更**

このステップでは、以下のように MyNewControllerCmdImpl の performExecute メソッド中のコードに変更を加えて、新規 Bonus Bean に関連したコードを組み込みます。

1. **MyNewControllerCmdImpl** クラスをダブルクリックします。
2. 「概略 (Outline)」ビューで、**performExecute** メソッドを選択します。
3. ソース・コードの中で「Section 4E」、「Section 4G」および「Section 4H」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。

```

// Section 4E //////////////////////////////////////
/// pass bb instance variable to the task command
cmd.setBonusAccessBean(bb);
// End of section 4E //////////////////////////////////////

// Section 4G //////////////////////////////////////
if (cmd.getOldBonusPoints() != null) {
 rspProp.put("oldBonusPoints", cmd.getOldBonusPoints());
}

```

```
// End of section 4G //////////////////////////////////////
// Section 4H //////////////////////////////////////
///Instantiate the bonus data bean , then put it to response properties
BonusDataBean bdb =
 new com.ibm.commerce.sample.databeans.BonusDataBean(
 cmd.getBonusAccessBean());
 rspProp.put("bdbInstance", bdb);
// End of section 4H //////////////////////////////////////
```

#### 4. 変更内容を保管します。

**注:** BonusDataBean が定義されていないため、エラーが表示されます。これらは、次のセクションで修正されます。

### BonusDataBean Data Bean の作成

プログラミング・モデルと一致するように、新規 Bonus Entity Bean に対応する Data Bean を新規作成する必要があります。すべての Entity Bean が対応する Data Bean を持つ必要はありませんが、Entity Bean 中の情報を JSP テンプレート中に表示できるようにしたい場合は、そのために Data Bean を新規作成する必要があります。

このシナリオでは、BonusAccessBean を拡張する BonusDataBean Data Bean を新規作成する必要があります。チュートリアル他の部分と同様に、基本コードが備えられているので、コードのさまざまなセクションをコメント解除する必要があります。

BonusDataBean をコードに導入するには、以下のようにします。

1. 最初のステップとして、以下のように新規 Data Bean の基本コードをインポートします。
  - a. Java パースペクティブで、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
  - b. **WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。
  - c. 「src」フォルダーを右クリックして、「インポート (Import)」を選択します。「インポート (Import)」ウィザードがオープンします。
  - d. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
  - e. 「ブラウザ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、  
`yourDirectory\WC_SAMPLE_55.zip`  
 にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
  - f. 「全選択解除 (Deselect All)」をクリックし、ディレクトリーを拡張表示して、インポートする以下のファイルを選択します。
    - `com\ibm\commerce\sample\databeans\BonusDataBean.java`

- g. 「フォルダー (Folder)」フィールドでは、  
「WebSphereCommerceServerExtensionsLogic/src」フォルダーがすでに指定されています。この値をそのまま使用します。
  - h. 「終了」をクリックします。
2. **BonusDataBean** クラスをダブルクリックして、そのソース・コードを表示します。
  3. 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 Bean に導入します。

```

/// Section 1 //////////////////////////////////////

// create fields and accessors (setter/getter methods)

private java.lang.String userId;
private java.lang.Integer totalBonusPoints;

public java.lang.String getUserId() {
 return userId;
}

public void setUserId(java.lang.String newUserId) {
 userId = newUserId;

 //////////////////////////////////////
 /// Section A : instantiate BonusAccessbean

 if (userId != null)
 this.setInitKey_memberId(new Long(newUserId));

 //////////////////////////////////////
}

 public java.lang.Integer getTotalBonusPoints() {
 return totalBonusPoints;
 }
 public void setTotalBonusPoints(java.lang.Integer newTotalBonusPoints) {
 totalBonusPoints= newTotalBonusPoints;
 }

 /// End of section 1 //////////////////////////////////////

```

4. 次に、「Section 2」をコメント解除して、以下のコードのセクションを Bean に導入します。

```

/// Section 2////////////////////////////////////

// create a new constructor for passing access bean into databean
// so that JSP can work with the access bean

public BonusDataBean(BonusAccessBean bb)

```

```

 throws com.ibm.commerce.exception.ECException {
 try {
 super.setEJBRef(bb.getEJBRef());
 } catch (javax.ejb.FinderException e) {
 throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
 "BonusDataBean", "BonusDataBean(bb)");
 } catch (javax.naming.NamingException e) {
 throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
 "BonusDataBean", "BonusDataBean(bb)");
 } catch (java.rmi.RemoteException e) {
 throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
 "BonusDataBean", "BonusDataBean(bb)");
 } catch (javax.ejb.CreateException e) {
 throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
 "BonusDataBean", "BonusDataBean(bb)");
 }
}
}

```

```

//// End of section 2 //////////////////////////////////////

```

5. 次に、「Section 3」をコメント解除して、以下のコードを Bean に導入します。

```

//// Section 3 //////////////////////////////////////

```

```

// set additional data field that is used for instantiating BonusAccessbean

```

```

try
{
 setUserId(getRequestProperties().getString("taskOutputUserId"));

 try {
 super.refreshCopyHelper();
 } catch (javax.ejb.FinderException e) {
 throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
 "BonusDataBean", "populate");
 } catch (javax.naming.NamingException e) {
 throw new ECSystemException(ECMessage._ERR_NAMING_EXCEPTION,
 "BonusDataBean", "populate");
 } catch (java.rmi.RemoteException e) {
 throw new ECSystemException(ECMessage._ERR_REMOTE_EXCEPTION,
 "BonusDataBean", "populate");
 } catch (javax.ejb.CreateException e) {
 throw new ECSystemException(ECMessage._ERR_CREATE_EXCEPTION,
 "BonusDataBean", "populate");
 }
}

}
catch (ParameterNotFoundException e){}

```

```

////// End of Section 3 //////////////////////////////////////
}

```

6. 次に、「Section 4」をコメント解除して、以下のコードを Bean に導入します。

```

/// Section 4 //////////////////////////////////////
// copy input TypedProperteis to local

requestProperties = aParam;

/// End of section 4 //////////////////////////////////////

```

7. 変更内容を保管します。
8. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックして、「プロジェクトの再作成 (**Rebuild Project**)」を選択し、変更を加えたコードをコンパイルします。

### 新規 Entity Bean のためのアクセス制御ポリシーの作成

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 Entity 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 `tester@mycompany` ユーザーとしてログオンしている場合、このユーザーは自分のボーナス・ポイント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. アクセス制御ポリシーに変更を加えて、ご使用の環境を反映させます。
2. `SampleACPolicy.xml` ファイルを、`acpload` コマンドを使用してロードします。
3. `SampleACPolicy_en_US.xml` の説明を、`acpnlsload` コマンドを使用してロードします。



それぞれの環境のアクセス制御ポリシーをカスタマイズするには、以下のようにします。

1. 以下のようにして、FashionFlow ストアのメンバー ID 値を判別します。

- **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. 開発データベースに接続します。
- b. 以下の SQL ステートメントを出します。

```
select member_id from storeent where storeent_id=FF_storeent_ID
```

ここで *FF\_storeent\_ID* は、FashionFlow ストアのストア・エンティティ ID です。たとえば、以下のように入力します。

```
select member_id from storeent where storeent_id=10001
```

メンバー ID の値を記録しておいてください: \_\_\_\_\_

- **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、開発データベースに接続します。
- b. 以下を入力します。

```
column member_id format 99999999999999999999999999999999 (9 を 35 個)
```

これで、値全体を表示できるだけの member\_id 列の表示が設定されます。

- c. 以下の SQL ステートメントを出します。

```
select member_id from storeent where storeent_id=FF_storeent_ID
```

ここで *FF\_storeent\_ID* は、FashionFlow ストアのストア・エンティティ ID です。たとえば、以下のように入力します。

```
select member_id from storeent where storeent_id=10001
```

メンバー ID の値を記録しておいてください: \_\_\_\_\_

- d. 以下を入力して、member\_id 列の表示幅をリセットします。

```
column member_id clear
```

2. テキスト・エディターを使用して、(*WCStudio\_installdir*\Commerce\xml\policies\xml ディレクトリーの) *SampleACPolicy\_template.xml* ファイルをオープンし、*FashionFlowMemberId* を、ステップ 1 で判別した Fashion Flow ストアのメンバー ID 値に置き換えます。変更を加えたファイルを (同じディレクトリー内に) *SampleACPolicy.xml* として保管します。
3. テキスト・エディターを使用して、(*WCStudio\_installdir*\Commerce\xml\policies\xml ディレクトリーの) *SampleACPolicy\_template\_en\_US.xml* ファイルをオープンし、

*FashionFlowMemberId* を、ステップ 1 で判別した Fashion Flow ストアのメンバー ID 値に置き換えます。変更を加えたファイルを (同じディレクトリー内に) *SampleACPolicy\_en\_US.xml* として保管します。

4. コマンド・プロンプトで、以下のディレクトリーに切り替えます。

```
WCStudio_installdir¥commerce¥bin
```

5. *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。ここでは *SampleACPolicy.xml* と入力します。

たとえば、以下のコマンドを出すとします。

```
acpload Demo_dev db2user db2user SampleACPolicy.xml
```

6. 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 Demo_dev db2user db2user SampleACPolicy_en_US.xml
```

7. テスト環境用のサーバーが現在実行中の場合は、WebSphere Commerce 管理コンソール中のレジストリー最新表示オプションを使用して、以下のようにアクセス制御レジストリーを更新できます。

- a. Web ブラウザーをオープンし、以下の URL を入力します。

```
https://localhost/webapp/wcs/admin/servlet/ToolsLogon?XMLFile=adminconsole.AdminConsoleLogon
```

- b. プロンプトが出されたら、サイト管理者 ID を使用してログインします。
- c. 「サイト」上の作業を選択して、「OK」をクリックします。
- d. 「構成」メニューから「レジストリー」を選択します。
- e. 「すべてを更新」をクリックしてから、直後に「最新表示」をクリックし、更新されていることを検証します。
- f. 管理コンソール・ウィンドウをログアウトしてクローズします。

## ボーナス・ポイントを組み込むように MyNewJSPTemplate.jsp テンプレートに変更を加える

表示ページに変更を加えるには、以下のようにします。

1. WebSphere Studio Application Developer で、Web パースペクティブに切り替えます。
2. **MyNewJSPTemplate\_All.jsp** ファイルと **MyNewJSPTemplate.jsp** ファイルを両方ともオープンします。
3. 「Section 9」を MyNewJSPTemplate\_All.jsp ファイルから MyNewJSPTemplate.jsp ファイルにコピーします。これによって、以下のテキストが JSP テンプレートに導入されます。

```

<!-- SECTION 9 -->

<h2><fmt:message key="BonusAdmin" bundle="\${tutorial}" /> </h2>

<c:if test="\${!empty taskOutputUserId}">

 <fmt:message key="PointBeforeUpdate" bundle="\${tutorial}" />
 <c:out value="\${oldBonusPoints}" />

 <fmt:message key="PointAfterUpdate" bundle="\${tutorial}" />
 <c:out value="\${bdbInstance.bonusPoint}" />

</c:if>

<fmt:message key="EnterPoint" bundle="\${tutorial}" /><p />

<form name="Bonus" action="MyNewControllerCmd">
<table>
<tr>
<td>
 Logon ID
</td>
<td>
 <input type="text" name="input1" value="<c:out
value="\${userName}" />" />
</td>
</tr>
<tr>
<td>
 Bonus Point
</td>
<td>
 <input type="text" name="input2" />
</td>
</tr>

```

```
<tr>
 <td colspan="2">
 <input type="submit" />
 </td>
</tr>
</table>
</form>
```

```
<!-- END OF SECTION 9 -->
```

4. MyNewJSPTemplate.jsp ファイルを保管します。

### 統合した Bonus Bean のテスト

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。

新規ロジックをテストするには、以下のようにします。

1. 「サーバー (Server)」ビューに切り替えます。
2. **WebSpherCommerceServer** サーバーを右クリックして、「スタート」 (または「再始動 (Restart)」) を選択します。
3. ストアの **index.jsp** ファイルを右クリックし、「サーバー上で実行 (Run on Server)」を選択します。  
ストアのホーム・ページが表示されます。
4. 以下のようにして、登録済みユーザーとしてログオンします。
  - a. 「登録」リンクをクリックします。  
「登録」ページが表示されます。
  - b. 「E メール・アドレス」フィールドで、276 ページの『ユーザー名妥当性検査のテスト』で作成したユーザーの E メール・アドレスを入力します。
  - c. 「パスワード」フィールドでこの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。
  - d. ログインが完了してから、同じブラウザに以下の URL を入力します。

```
http://localhost/webapp/wcs/stores/servlet/MyNewControllerCmd?
input1=user_e-mail&input2=1000
```

ここで *user\_e-mail* は、276 ページの『ユーザー名妥当性検査のテスト』で作成したユーザーの E メール・アドレスです。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・ポイントの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。

**Bonus Administration**

- The bonus point before update is 3000
- The bonus point after update is 4000

Please enter the points, then submit it to the controller command

Logon ID

Bonus Point

図 41.

- e. 次に、「ログオン ID」フィールドにユーザーの E メール・アドレスを入力し、「ボーナス・ポイント (Bonus Point)」フィールドに 500 と入力します。「送信」をクリックします。以下のような、ボーナス・ポイントのバランスが更新されたことを示すページが表示されます。

**Bonus Administration**

- The bonus point before update is 4000
- The bonus point after update is 4500

Please enter the points, then submit it to the controller command

Logon ID

Bonus Point

図 42.

---

## ボーナス・ポイント・ロジックのデプロイメント

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 (FashionFlow 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。

ターゲットの WebSphere Commerce Server にデプロイメントしなければならない資産にはさまざまなタイプがあります。以下のものが含まれます。

- コントローラー・コマンド、タスク・コマンド、および Data Bean のロジック
- Enterprise Bean のロジック
- JSP テンプレートおよびイメージ・ファイル
- プロパティ・ファイルおよびリソース・バンドル
- スキーマの更新 (新規テーブル) やコマンド・レジストリーの更新を含む、データベースの更新
- アクセス制御の更新

このセクションでは、これらのすべての資産をターゲットの WebSphere Commerce Server に増分 デプロイメントする方法について説明します。この増分デプロイメントは、EAR ファイル全体のデプロイメントとは対照的な方法です。

## コマンドと Data Bean の JAR ファイルの作成

このセクションでは、コントローラー・コマンド、タスク・コマンド、および Data Bean のロジックを含む JAR ファイルを作成する方法について説明します。

この JAR ファイルを作成するには、開発マシンで以下のステップを実行します。

1. `drive:\ExportTemp` というローカル・ファイル・システム上にディレクトリーを作成します。
2. WebSphere Studio Application Developer で、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
3. **WebSphereCommerceServerExtensionsLogic** 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。  
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようになります。
  - a. 「JAR ファイル」を選択し、「次へ」をクリックします。

- b. 「エクスポートするリソースの選択 (Select the resources to export)」の下の左側のペインに、プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。
- c. 右側のペインで、以下のリソースだけが選択されていることを確認します。
  - .classpath
  - .project
  - .serverPreference
- d. 「生成されたクラス・ファイルおよびリソースのエクスポート (Export generated class files and resources)」が選択されていることを確認します。
- e. 「Java ソース・ファイルおよびリソースのエクスポート (Export Java source files and resources)」を選択しないでください。
- f. 「エクスポート先の選択 (Select the export destination)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。ここでは、`drive:¥ExportTemp¥WebSphereCommerceServerExtensionsLogic.jar` と入力します。JAR ファイル名は `WebSphereCommerceServerExtensionsLogic.jar` でなければならないことに注意してください。
- g. 「終了」をクリックします。

## EJB JAR ファイルの作成

EJB JAR ファイルを作成するには、以下のようになります。

1. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. **WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
3. 「EJB Deployment Descriptor」をダブルクリックします。
4. 「概要 (Overview)」タブを選択しながら、ペインの下部にスクロールし、「WebSphere バインディング (WebSphere Bindings)」セクションを見つけます。
5. 「データ・ソース JNDI 名 (DataSource JNDI name)」フィールドに、ターゲットの WebSphere Commerce Server のデータ・ソース JNDI 名を入力します。値の例は以下のとおりです。

 jdbc/WebSphere Commerce DB2 DataSource demo

ここでターゲット WebSphere Commerce Server は、DB2 データベースを使用し、WebSphere Commerce インスタンス名は “demo” です。

 jdbc/WebSphere Commerce Oracle DataSource demo

ここでターゲット WebSphere Commerce Server は、Oracle データベースを使用し、WebSphere Commerce インスタンス名は “demo” です。



DataSource JNDI 名の値は、“jdbc/” を、ターゲット WebSphere Commerce Server のデータ・ソース名に追加することで作成されます。ターゲット WebSphere Commerce Server で *instanceName.xml* ファイルをオープンし、ファイル内で *DatasourceName=* を探すことで、データ・ソース名を確認できます。

6. デプロイメント記述子の変更内容を保管します (Ctrl + S)。
7. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、**WebSphereCommerceServerExtensionsData** 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。  
「エクスポート」ウィザードがオープンします。
8. 「エクスポート」ウィザードで、以下のようになります。
  - a. 「EJB JAR ファイル」を選択し、「次へ」をクリックします。
  - b. 「エクスポートしたいリソース (What resources do you want to export?)」の値として、EJB プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。
  - c. 「リソースのエクスポート先にしたい場所 (Where do you want to export resources to?)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。ここでは、  
`drive:¥ExportTemp¥WebSphereCommerceServerExtensionsData.jar` と入力します。
  - d. 「終了」をクリックします。
9. JAR ファイルを作成し終わったら、ステップ 5 でローカル・デプロイメント記述子に加えた変更を取り消して、ローカル・テスト・サーバーに必要な設定を復元します。

**注:** このチュートリアルでは、開発データベースと、ターゲット WebSphere Commerce Server で使用するデータベースが、同じタイプであると想定しています。異なるデータベース・タイプにデプロイしていた場合、216 ページの『変換を行う EJB JAR ファイルの作成』の指示に従ってください。

## ストア資産のエクスポート

ストア資産をエクスポートするには、以下のようになります。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 「ストア」フォルダーを拡張表示します。
3. 「Web コンテンツ (Web Content)」フォルダーを右クリックして、「エクスポート」を選択します。  
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようになります。



- a. 「ファイル・システム (File system)」を選択して、「次へ」をクリックします。
- b. 「全選択解除 (Deselect All)」をクリックします。
- c. 以下のリソースをエクスポートすることを選択します。
  - Web Content¥FashionFlow\_name¥MyNewJSPTemplate.jsp
  - Web Content¥FashionFlow\_name¥images¥male\_blueshirt.gif
  - Web Content¥WEB-INF¥classes¥FashionFlow\_name¥Tutorial\_NLS\_en\_US.properties
  - Web Content¥WEB-INF¥lib¥jstl.jar
  - Web Content¥WEB-INF¥lib¥standard.jar
- d. 「ファイルのディレクトリー構造の作成 (Create directory structure for files)」を選択します。
- e. 「ディレクトリー」フィールドに、これらのリソースを入れる一時ディレクトリーを入力します。たとえば C:¥ExportTemp と入力します。
- f. 「終了」をクリックします。

## アクセス制御ポリシーのパッケージ化

このセクションでは、以下のようにして、作成した新規リソースのアクセス制御ポリシーを、 `drive:¥ExportTemp` ディレクトリーにコピーします。

1. `WCStudio_installdir¥Commerce¥xml¥policies¥xml` ディレクトリーに移動します。
2. 以下のファイルを、 `drive:¥ExportTemp¥ACPolicies` ディレクトリーにコピーします。
  - `MyNewViewACPolicy.xml`
  - `MyNewControllerCmdACPolicy.xml`
  - `SampleACPolicy_template.xml`
  - `SampleACPolicy_template_en_US.xml`

## ターゲット WebSphere Commerce Server への資産の転送

このステップでは、ターゲットの WebSphere Commerce Server 上に一時ディレクトリーを作成してから、このディレクトリー中にボーナス・ポイント資産をコピーします。以後のステップでは、 WebSphere Commerce アプリケーション中の該当する場所にさまざまなタイプのコードを入れます。

開発マシンからターゲットの WebSphere Commerce Server にファイルをコピーするには、以下のようにします。

1. ターゲットの WebSphere Commerce Server で、 `drive:¥ImportTemp` という一時ディレクトリーを作成します。

2. コンピューター間でファイルをコピーする方法を決めます。ターゲットの WebSphere Commerce Server 上のドライブを開発マシンにマッピングするか、または FTP アプリケーションが構成済みの場合はこのアプリケーションを使用して、コピーを実行できます。
3. 開発マシンから、`drive:¥ExportTemp` の内容をターゲットの WebSphere Commerce Server の `drive:¥ImportTemp` にコピーします。

## ターゲット WebSphere Commerce Server の停止

デプロイメント・ステップを開始する前に、ターゲット WebSphere Commerce Server を停止する必要があります。WebSphere Commerce Server 停止の詳細は、「*WebSphere Commerce Studio* インストール・ガイド」を参照してください。


## ターゲット WebSphere Commerce Server 上のデータベースの更新

ターゲット・データベースを更新する前に、カスタマイズ・ロジックをデプロイするストアの、ストア・エンティティ ID を確認します。この値を判別するには、以下の SQL ステートメントを使用できます。

```
select STOREENT_ID from STOREENT where IDENTITY='FashionFlow_name'
```

ここで *FashionFlow\_name* は、コードをデプロイするストアの名前です。

## テーブルへのビューの登録

 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 MyNewView を登録します。

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」>「コマンド行ツール (Command Line Tools)」>「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. ツール設定をクローズします。
5. 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、VIEWREG テーブルに必要なエントリーを作成します。

```
connect to targetDB user dbuser using dbpassword;
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
 CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('MyNewView',-1, FF_storeent_ID,
 'com.ibm.commerce.command.ForwardViewCommand',
 'com.ibm.commerce.command.HttpForwardViewCommandImpl',
 'docname=MyNewJSPTemplate.jsp','This is my new view for tutorial 1',
 0, null);
```

ここで

- *targetDB* は、ターゲット・データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード
- *FF\_storeent\_ID* は、 FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID

「実行」アイコンをクリックします。コマンド・センターはオープンしたままにしておきます。

**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。

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。

```
insert into VIEWREG (VIEWNAME, DEVICEFMT_ID, STOREENT_ID, INTERFACENAME,
 CLASSNAME, PROPERTIES, DESCRIPTION, HTTPS, LASTUPDATE)
values ('MyNewView',-1, FF_storeent_ID,
 'com.ibm.commerce.command.ForwardViewCommand',
 'com.ibm.commerce.command.HttpForwardViewCommandImpl',
 'docname=MyNewJSPTemplate.jsp','This is my new view for tutorial 1',
 0, null);
```

ここで

- *FF\_storeent\_ID* は、 FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID。

Enter を押して SQL ステートメントを実行します。

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

これで MyNewView が登録されました。

## 新規コントローラー・コマンドの登録

MyNewControllerCmd を登録するには、以下のようにします。

1. **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 MyNewControllerCmd を登録します。


- a. コマンド・センターで、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。

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
 AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
 CLASSNAME, TARGET)
values (FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 'This is a new controller command for tutorial one.',
 'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl',
 'local');
```

ここで

- *targetDB* は、ターゲット・データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード
- *FF\_storeent\_ID* は、 FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID。

「実行」アイコンをクリックします。コマンド・センターはオープンしたままにしておきます。

2.  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 MyNewControllerCmd を登録します。

- 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、以下の SQL ステートメントを入力します。

```
insert into URLREG (URL, STOREENT_ID, INTERFACENAME, HTTPS, DESCRIPTION,
 AUTHENTICATED) values ('MyNewControllerCmd',FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 0, 'This is a new controller command for tutorial one.',null);
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
 CLASSNAME, TARGET)
values (FF_storeent_ID,
 'com.ibm.commerce.sample.commands.MyNewControllerCmd',
 'This is a new controller command for tutorial one.',
 'com.ibm.commerce.sample.commands.MyNewControllerCmdImpl','local');
```

ここで

- `FF_storeent_ID` は、FashionFlow サンプル・ストアを基にしたご使用のストアの固有 ID。

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、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

## XBONUS テーブルの作成

このステップでは、ターゲットの WebSphere Commerce Server で使用されているデータベース中に XBONUS テーブルを作成します。

**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. 「スクリプト」ウィンドウで、以下を入力します。

```
create table XBONUS (MEMBERID BIGINT NOT NULL,
 BONUSPOINT INTEGER NOT NULL, constraint p_xbonus
 primary key (MEMBERID),
 constraint f_xbonus foreign key (MEMBERID)
 references users (users_id) on delete cascade)
```

ここで

- `targetDB` は、ターゲット・データベースの名前
- `dbuser` は、データベースのユーザー
- `dbpassword` は、データベースのユーザーのパスワード

「実行」アイコンをクリックします。

これで、XBONUS テーブルが作成されました。

**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 XBONUS (MEMBERID NUMBER NOT NULL,
 BONUSPOINT INTEGER NOT NULL, constraint p_xbonus
 primary key (MEMBERID),
 constraint f_xbonus 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。XBONUS テーブルが作成されました。

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

## ターゲット WebSphere Commerce Server 上へのアクセス制御ポリシーのロード

このステップでは、新規リソースのアクセス制御ポリシーを、ターゲットの WebSphere Commerce Server 上にロードします。

新しいポリシーをロードする方法は、以下のとおりです。

1. 以下のように、アクセス制御ポリシーを更新して、ターゲットの WebSphere Commerce Server に固有の値を反映させます。
  - a. 以下のようにして、FashionFlow ストアのメンバー ID 値を判別します。

- **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。

- 1) 開発データベースに接続します。
- 2) 以下の SQL ステートメントを出します。

```
select member_id from storeent where storeent_id=FF_storeent_ID
```

ここで *FF\_storeent\_ID* は、FashionFlow ストアのストア・エンティティ ID です。たとえば、以下のように入力します。

```
select member_id from storeent where storeent_id=10001
```

メンバー ID の値を記録しておいてください: \_\_\_\_\_

- **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。

- 1) 「Oracle SQL Plus」コマンド・ウィンドウをオープンし、開発データベースに接続します。
- 2) 以下を入力します。

```
column member_id format 99999999999999999999999999999999 (9 を 35 個)
```

これで、値全体を表示できるだけの member\_id 列の表示が設定されます。

- 3) 以下の SQL ステートメントを出します。

```
select member_id from storeent where storeent_id=FF_storeent_ID
```

ここで *FF\_storeent\_ID* は、FashionFlow ストアのストア・エンティティ ID です。たとえば、以下のように入力します。

```
select member_id from storeent where storeent_id=10001
```

メンバー ID の値を記録しておいてください: \_\_\_\_\_

4) 以下を入力して、member\_id 列の表示幅をリセットします。

```
column member_id clear
```

b. 以下のディレクトリーに移動します。

```
drive:\ImportTemp\ACPolicies
```

c. テキスト・エディターを使用して、SampleACPolicy\_template.xml ファイルをオープンし、FashionFlowMemberId を、ステップ 1a で判別した Fashion Flow ストアのメンバー ID 値に置き換えます。変更を加えたファイルを SampleACPolicy.xml として保管します。

d. テキスト・エディターを使用して、SampleACPolicy\_template\_en\_US.xml ファイルをオープンし、FashionFlowMemberId を、ステップ 1a で判別した Fashion Flow ストアのメンバー ID 値に置き換えます。変更を加えたファイルを SampleACPolicy\_en\_US.xml として保管します。

2. 以下のアクセス制御ポリシーを、WC\_installdir\xml\policies\xml ディレクトリーにコピーします。

- MyNewViewACPolicy.xml
- MyNewControllerCmdACPolicy.xml
- SampleACPolicy.xml
- SampleACPolicy\_en\_US.xml

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。

```
WC_installdir\bin
```

4. 以下のフォームの acpload コマンドを出す必要があります。

```
acpload targetDB dbuser dbpassword inputXMLFile
```

ここで

- targetDB は、ユーザーの開発データベースの名前
- dbuser は、データベースのユーザーの名前
- dbpassword は、データベースのユーザーのパスワード
- inputXMLFile は、アクセス制御ポリシーの仕様を含む XML ファイル。この場合は、MyNewViewACPolicy.xml を指定します。

以下はこのコマンドの例で、変数が指定されています。

```
acpload Demo_Dev db2admin db2admin MyNewViewACPolicy.xml
```

5. 以下のアクセス制御ポリシーごとに、ステップ 4 を繰り返します。

- MyNewControllerCmdACPolicy.xml
- SampleACPolicy.xml

6. ポリシーの説明 (SampleACPolicy\_en\_US.xml ファイルに含まれている) をロードするには、以下のフォームの `acpnload` コマンドを出す必要があります。

```
acpnload 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。

```
acpnload Demo_dev user password SampleACPolicy_en_US.xml
```

7. アクセス制御ポリシーのロード時に生成されたエラー・ログがあるかどうか、`WC_installdir¥xml¥policies¥xml` ディレクトリーを調べます。

## ターゲット WebSphere Commerce Server 上のストア資産の更新

このステップでは、以下のようにして、変更を加えたストア資産でストアを更新します。

1. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war` ディレクトリー (ここで、`cellName` はご使用のマシンのホスト名であることが多く、`instanceName` は WebSphere Commerce インスタンスの名前です) のバックアップを取ります。
2. `drive:¥ImportTemp¥Stores¥Web Content` ディレクトリーに移動します。
3. 「`FashionFlow_name`」および「`WEB-INF`」フォルダーを、以下のディレクトリーにコピーします。  
`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war`

## ターゲット WebSphere Commerce Server 上のコマンドと Data Bean の JAR ファイルの更新

このステップでは、以下のようにして、新しいコマンドと Data Bean の JAR ファイルを使用するようにターゲット WebSphere Commerce Server を更新します。

1. 以下のようにして、既存の JAR ファイルのバックアップ・コピーを作成する必要があります。
  - a. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーに移動します。
  - b. `WebSphereCommerceServerExtensionsLogic.jar` ファイルのコピーを作成して、バックアップ場所に保管します。
2. 新しい `WebSphereCommerceServerExtensionsLogic.jar` ファイルを、`drive:¥ImportTemp` ディレクトリーから、`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーにコピーします。

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

## ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新

このステップでは、以下のようにして、新しい EJB JAR ファイルを使用するようにターゲット WebSphere Commerce Server を更新します。



1. 以下のようにして、既存の JAR ファイルのバックアップ・コピーを作成する必要があります。
  - a. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーに移動します。
  - b. `WebSphereCommerceServerExtensionsData.jar` ファイルのコピーを作成して、バックアップ場所に保管します。
2. 新しい `WebSphereCommerceServerExtensionsData.jar` ファイルを、`drive:¥ImportTemp` ディレクトリーから、`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーにコピーします。
3. 次に、以下のようにして、EJB デプロイメント記述子の情報に変更を加えなければなりません。
  - a. この WebSphere Application Server セルのデプロイメント・リポジトリー (META-INF ディレクトリー) を見つけます。これは一般的に、以下のような形式になります。

```
WAS_installdir¥config¥cells¥cellName
¥applications¥WC_instance_name.ear¥deployments¥
WC_instance_name¥EJBModuleName.jar¥META-INF.
```

以下の形式は、このチュートリアルに固有の例です。

```
D:¥WebSphere¥AppServer¥config¥cells¥myCell¥applications¥
WC_demo.ear¥deployments¥WC_demo¥
WebSphereCommerceServerExtensionsData.jar¥META-INF
```
  - b. このディレクトリーには、以下のファイルが含まれています。
    - `ejb-jar.xml`
    - `ibm-ebj-access-bean.xmi`
    - `ibm-ebj-jar-bnd.xmi`
    - `ibm-ebj-jar-ext.xmi`
    - `MANIFEST.MF`これらすべてのファイルのバックアップを取ります。
  - c. ツールを使用して、新しい `WebSphereCommerceServerExtensionsData.jar` ファイルをオープンし、その内容を表示します。
  - d. `meta-inf` ディレクトリー (前述のリストされたファイル) の内容を、この `WebSphereCommerceServerExtensionsData.jar` ファイルから、ステップ 3a のディレクトリー中に抽出します。ファイルを抽出した後も、ディレクトリー構造が正しいことを確認してください。
4. コマンド行で `WebSphere Application Server startServer` コマンドを使用して、`WebSphere Commerce` インスタンスを再始動します。このインスタンスの開始と停止の詳細は、使用しているプラットフォームおよびデータベースの「*WebSphere Commerce* インストール・ガイド」を参照してください。

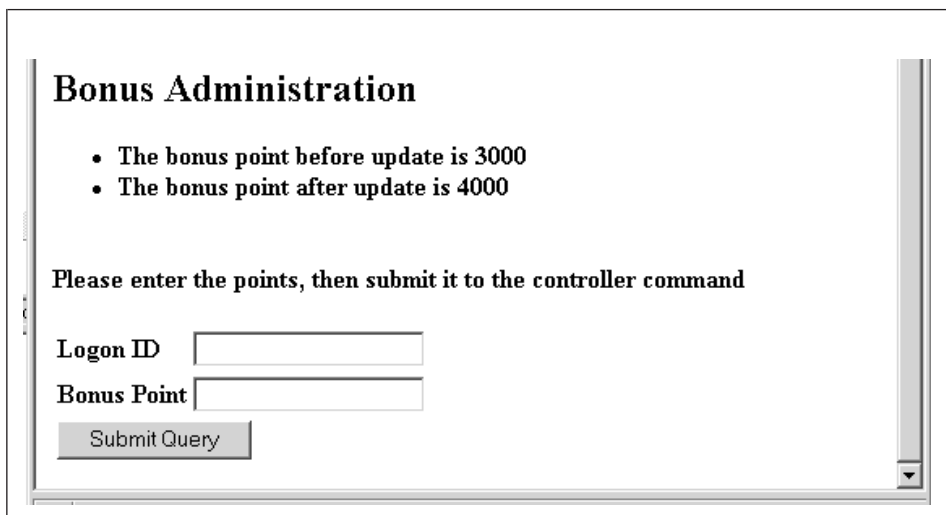
## ターゲット WebSphere Commerce Server 上のボーナス・ポイント・ロジックの検証

このステップでは、以下のようにして、ボーナス・ポイント・ロジックが、ターゲット WebSphere Commerce Server へ正常にデプロイされたことを検証します。

1. Web ブラウザーをオープンし、以下の URL を入力して、FashionFlow を基にしたご使用のストアを立ち上げます。
2. 以下のようにして、新しい登録ユーザーを作成します。
  - a. 「登録」をクリックします。
  - b. もう一度「登録」をクリックして、新しい顧客を作成します。
  - c. 登録フォームで、すべての必須フィールドに適切な値を入力します。たとえば、E メール・フィールドには、`tester@mycompany` と入力します。E メール・アドレスの値を記録しておいてください: \_\_\_\_\_。
  - d. 値を入力したら、「送信」をクリックします。
3. 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://hostname/webapp/wcs/stores/servlet/MyNewControllerCmd?
input1=user_e-mail&input2=1000
```

ここで `hostname` はホスト名で、`user_e-mail` はステップ 2 で作成したユーザーの E メール・アドレスです。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・ポイントの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。



**Bonus Administration**

- The bonus point before update is 3000
- The bonus point after update is 4000

Please enter the points, then submit it to the controller command

Logon ID

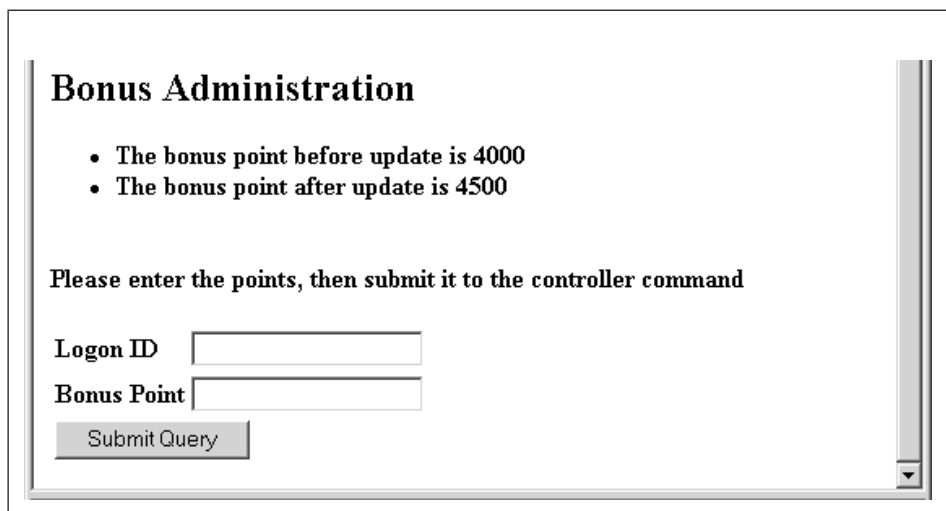
Bonus Point

Submit Query

図 43.

4. 次に、「ログオン ID」フィールドにユーザーの E メール・アドレスを入力し、「ボーナス・ポイント (Bonus Point)」フィールドに 500 と入力します。「送信」

をクリックします。以下のような、ボーナス・ポイントのバランスが更新されたことを示すページが表示されます。



**Bonus Administration**

- The bonus point before update is 4000
- The bonus point after update is 4500

Please enter the points, then submit it to the controller command

Logon ID

Bonus Point

図 44.



---

## 第 11 章 チュートリアル: 既存のコントローラー・コマンドの変更

このチュートリアルの目的は、既存のコントローラー・コマンドの変更を使用するプロセスを示すことです。

このチュートリアルでは、顧客のショッピング・カート中のアイテム数を 5 以下に制限します。このソリューションをインプリメントするには、カート中のアイテム数を検査するロジックを含む独自のインプリメンテーションで `OrderItemAddCmdImpl` をオーバーライドします。顧客が 6 つめのアイテムをショッピング・カートに追加しようとすると、例外がスローされます。この例外は、新しいエラー・メッセージを使用します。

このチュートリアルの目的は、既存のコマンド・ロジック変更を使用される開発プロセスを示すことです。ショッピング・カートのアイテムを制限する、適用枠の狭い例ではまったくありません。このチュートリアルで使用されるロジックは、チュートリアル用に単純化されています。

このチュートリアルでは、以下の点を学習します。

- 既存コントローラー・コマンドの新規インプリメンテーションを作成する方法
- コマンド・レジストリーを更新して、アプリケーション中で新規インプリメンテーションを使用できるようにする方法
- 変更を加えたコントローラー・コマンドを既存の WebSphere Commerce アプリケーションにデプロイメントする方法

---

### 前提条件

この章のチュートリアルは、229 ページの『第 10 章 チュートリアル: ビジネス・ロジックの新規作成』を完了している必要はありません。このチュートリアルを完了している場合でも、そのコードはこの章のチュートリアルと競合しないので、コードをワークスペース中に残しておいても害はありません。

この章のチュートリアルを開始する前に、FashionFlow サンプル・ストアを基にしたストアを発行していなければなりません。このストア中で購入を完了できなければなりません (カタログのブラウズ、ショッピング・カートへのアイテムの追加、オーダーの確認のチェックアウトや表示など)。

デプロイメントのステップを完了するには、ターゲットの WebSphere Commerce Server 上にもストアがなければなりません。

---

## 新規の MyOrderItemAddCmdImpl クラスの作成

チュートリアルはこのステップでは、MyOrderItemAddCmdImpl クラスを新規作成します。このクラスを作成するには、以下のようにします。

1. WebSphere Studio Application Developer で、Java パースペクティブをオープンします (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「Java」)。
2. **WebSphereCommerceServerExtensionsLogic** プロジェクトに移動します。
3. **src** ディレクトリーに移動します。
4. 229 ページの『第 10 章 チュートリアル: ビジネス・ロジックの新規作成』を完了していない場合は、**src** ディレクトリーを右クリックし、「新規」>「パッケージ」を選択します。  
「新規 Java パッケージ (New Java Package)」ウィザードがオープンします。このウィザードで、以下のようにします。
  - a. 「名前」フィールドで、`com.ibm.commerce.sample.commands` と入力します。
  - b. 「終了」をクリックします。
5. **com.ibm.commerce.sample.commands** パッケージを右クリックします。「新規」>「クラス (Class)」を選択します。  
「新規 Java クラス (New Java Class)」ウィザードがオープンします。
6. 「新規 Java クラス (New Java Class)」ウィザードで、以下のようにします。
  - a. 「名前」フィールドで、`MyOrderItemAddCmdImpl` と入力します。
  - b. スーパークラスを指定するために、「スーパークラス (Superclass)」フィールドの隣の「ブラウズ」ボタンをクリックし、`OrderItemAddCmdImpl` と入力します。  
「OK」をクリックします。
  - c. インプリメントするインターフェースを指定するには、「追加」をクリックし、`OrderItemAddCmd` と入力して、「OK」をクリックします。
  - d. 「終了」をクリックします。

MyOrderItemAddCmdImpl クラスのソース・コードが表示されます。

次のステップとして、以下のように、このクラス中のインポート・ステートメントを更新します。

1. 「概略 (Outline)」ビューで、「インポート宣言 (import declarations)」を選択してオープンします。2 つのインポート・ステートメントがすでに作成されていることが分かります。
2. インポート・ステートメントのソース・コードに、以下のインポート・ステートメントを追加します。

```
import com.ibm.commerce.exception.ECApplicationException;
import com.ibm.commerce.exception.ECException;
import com.ibm.commerce.exception.ECSystemException;
```

```
import com.ibm.commerce.order.objects.OrderAccessBean;
import com.ibm.commerce.ras.ECMessage;
import com.ibm.commerce.sample.messages.MyNewMessages;
```

3. 変更内容を保管します。

次のステップとして、ビジネス・ロジックと例外処理を追加し、オーダー・アイテムを追加する前に顧客のショッピング・カート中のアイテムが 5 つより多いかどうかを判別するようにします。コードを以下のように更新します。

1. 「概略 (Outline)」ビューで、MyOrderItemAddCmdImpl クラスを選択して、そのソース・コードを表示します。現在、以下のコードのみあります。

```
public class MyOrderItemAddCmdImpl
 extends OrderItemAddCmdImpl
 implements OrderItemAddCmd {

}
```

2. 新しい performExecute メソッドをこのクラスに追加しなければなりません。このメソッドには、ショッピング・カート中のアイテム数を検査し、5 つ未満の場合はスーパークラス (OrderItemAddCmdImpl) の正規の performExecute メソッドを通常どおり呼び出すロジックが含まれています。アイテムが 5 つ以上ある場合は、例外がスローされ、ユーザーはカートにアイテムを追加できません。このメソッドを追加するには、以下のソース・コードをクラス中にコピーします (クラスの末尾を示す最後の右中括弧「}」の前に組み込んだことを確認してください)。

```
public void performExecute() throws ECException {
 // Get a list of order ids
 String[] orderIds = getOrderIds();

 // Check to make sure that an id exists at all
 // if order id exists then get number of items in the order
 // else if no order id exists then execute normal code
 if (orderIds != null && orderIds.length > 0) {
 // An exception should be thrown when trying to add a sixth item
 // to the cart. Since this code is run before any items are added
 // throw and exception if there are 5 or more items in the cart
 if (itemsInOrder(orderIds[0]) >= 5) {
 throw new ECApplicationException(
 MyNewMessages.ERR_TOO_MANY_ITEMS,
 this.getClass().getName(),
 "performExecute");
 }
 //else perform normal flow
 }
 super.performExecute();
}

//get number of items in the order
protected int itemsInOrder(String orderId) throws ECException {
 try {
 OrderAccessBean order = new OrderAccessBean();
 order.setInitKey_orderId(orderId);
 order.refreshCopyHelper();
 }
}
```

```

 return order.getOrderItems().length;
 } catch (javax.ejb.FinderException e) {
 throw new ECSystemException(
 ECMessage.ERR_FINDER_EXCEPTION,
 this.getClass().getName(),
 "itemsInOrder");
 } catch (javax.naming.NamingException e) {
 throw new ECSystemException(
 ECMessage.ERR_NAMING_EXCEPTION,
 this.getClass().getName(),
 "itemsInOrder");
 } catch (java.rmi.RemoteException e) {
 throw new ECSystemException(
 ECMessage.ERR_REMOTE_EXCEPTION,
 this.getClass().getName(),
 "itemsInOrder");
 } catch (javax.ejb.CreateException e) {
 throw new ECSystemException(
 ECMessage.ERR_CREATE_EXCEPTION,
 this.getClass().getName(),
 "itemsInOrder");
 }
}
}

```



新しいコードをクラスに貼り付けたら、ソース・コードを右クリックして、「**フォーマット (Format)**」を選択して、コードをフォーマットします。

### 3. 作業内容を保管します

**注:** メッセージ情報がないことを示す警告があります。これは次のステップで訂正されます。

## メッセージ情報の作成

新しいコマンド・インプリメンテーションでは、`_ERR_TOO_MANY_ITEMS` という、新しいエラー・メッセージを使用します。このセクションでは、その新しいメッセージのコードと、それに関連付けられるプロパティー・ファイルを作成します。このコードをインポートするには、以下のようにします。

1. **WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。
2. **src** ディレクトリーを右クリックし、「**新規**」>「**パッケージ**」を選択します。  
「新規 Java パッケージ (New Java Package)」ウィザードがオープンします。このウィザードで、以下のようにします。
  - a. 「名前」フィールドで、`com.ibm.commerce.sample.messages` と入力します。
  - b. 「終了」をクリックします。



3. **com.ibm.commerce.sample.messages** パッケージを右クリックします。「新規」>「クラス (Class)」を選択します。  
「新規 Java クラス (New Java Class)」ウィザードがオープンします。
4. 「新規 Java クラス (New Java Class)」ウィザードで、以下のようになります。
  - a. 「名前」フィールドで、MyNewMessages と入力します。
  - b. 「終了」をクリックします。
5. **MyNewMessages** クラスをダブルクリックして、そのソース・コードを表示します。
6. コードの `public class MyNewMessages` 行のすぐ前に、以下の `import` ステートメントを追加します。
 

```
import com.ibm.commerce.ras.ECMessage;
import com.ibm.commerce.ras.ECMessageSeverity;
import com.ibm.commerce.ras.ECMessageType;
```
7. クラス内に、以下のコードを追加します。
 

```
// Resource bundle used to extract the text for an exception
static final String errorBundle = "MyNewErrorMessages";

// An ECMessage is used to describe an ECException and is passed
// into the ECException when thrown
public static final ECMessage _ERR_TOO_MANY_ITEMS =
 new ECMessage(ECMessageSeverity.ERROR, ECMessageType.USER,
 MyNewMessageKeys._ERR_TOO_MANY_ITEMS, errorBundle);
```
8. 変更内容を保管します。
9. **com.ibm.commerce.sample.messages** パッケージを右クリックします。「新規」>「クラス (Class)」を選択します。  
「新規 Java クラス (New Java Class)」ウィザードがオープンします。
10. 「新規 Java クラス (New Java Class)」ウィザードで、以下のようになります。
  - a. 「名前」フィールドで、MyNewMessageKeys と入力します。
  - b. 「終了」をクリックします。
11. クラスのコードを、以下のように定義します。
 

```
public class MyNewMessageKeys {
// This class defines the keys used to create new exceptions that are
// thrown by customized code.
public static final String _ERR_TOO_MANY_ITEMS = "_ERR_TOO_MANY_ITEMS";
}
```
12. 変更内容を保管します。
13. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックし、「プロジェクトの構築 (Build Project)」を選択して、コードをコンパイルします。
14. メッセージ情報を含む新規プロパティ・ファイルを、以下のように作成します。
  - a. Web パースペクティブをオープンします (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「Web」)。

- b. **Stores** Web プロジェクト内で、「**Web コンテンツ (Web Content)**」 > 「**WEB-INF**」 > 「**クラス (classes)**」 フォルダを拡張表示します。
- c. 「**クラス (classes)**」 フォルダを右クリックし、「**新規**」 > 「**その他**」 > 「**シンプル**」 > 「**ファイル**」 > 「**次へ**」を選択して、プロパティ・ファイルを新規作成します。  
「新規ファイル (New File)」ウィンドウがオープンします。
- d. 「**ファイル名**」フィールドに `MyNewErrorMessages.properties` と入力し、「**終了**」をクリックします。  
新しい空ファイルがオープンします。
- e. 新しいファイルに、以下のテキストをコピーします。  

```
_ERR_TOO_MANY_ITEMS=You are trying to place too many different items
in one shopping cart.
```


注: 上記のコードで改行されているのは、表示上の都合に過ぎません。テキストは 1 行で入力してください。
- f. 変更内容を保管します。

---

## コマンド・レジストリーに変更を加える

このステップでは、コマンド・レジストリーに変更を加えて、元の `OrderItemAddCmdImpl` インプリメンテーション・クラスの代わりに新しい `MyOrderItemAddCmdImpl` インプリメンテーション・クラスが使用されるようにします。コマンド・レジストリー中のテーブルのうち変更を加える必要があるのは、`CMDREG` テーブルのみです。ここでは、すべてのストアで新規インプリメンテーション・クラスを使用します。

コマンド・レジストリーを変更するには、以下のようになります。

1.  **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 `MyOrderItemAddCmdImpl` を登録します。
  - 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 developmentDB user dbuser using dbpassword;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
WHERE INTERFACENAME=
'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
and storeent_Id=0;
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード

「実行」アイコンをクリックします。

2. **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 `MyOrderItemAddCmdImpl` を登録します。
  - 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、以下の SQL ステートメントを入力します。

```
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
WHERE INTERFACENAME=
'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
and storeent_Id=0;
```

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、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

---

## MyOrderItemAddCmdImpl コマンドのテスト

次のステップとして、新規ロジックの機能が良好かテストして確認します。このテストを行うのは、ショッピング・カートに 5 つのアイテムを正常に追加でき、カートに 6 つ目のアイテムを追加しようとするエラーがスローされることを期待することです。

新規ビジネス・ロジックをテストするには、以下のようにします。

1. サーバー・パースペクティブに切り替えます（「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」）。

2. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」（または「再始動 (Restart)」）を選択します。
3. Stores¥Web Content¥FashionFlow\_name ディレクトリーの下の **index.jsp** を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。  
Web ブラウザー中にストアのホーム・ページが表示されます。
4. ストアでショッピングを行い、ショッピング・カートにアイテムを 5 つ追加します。5 つ目のアイテムを追加し終わると、ショッピング・カートは以下のようになります。

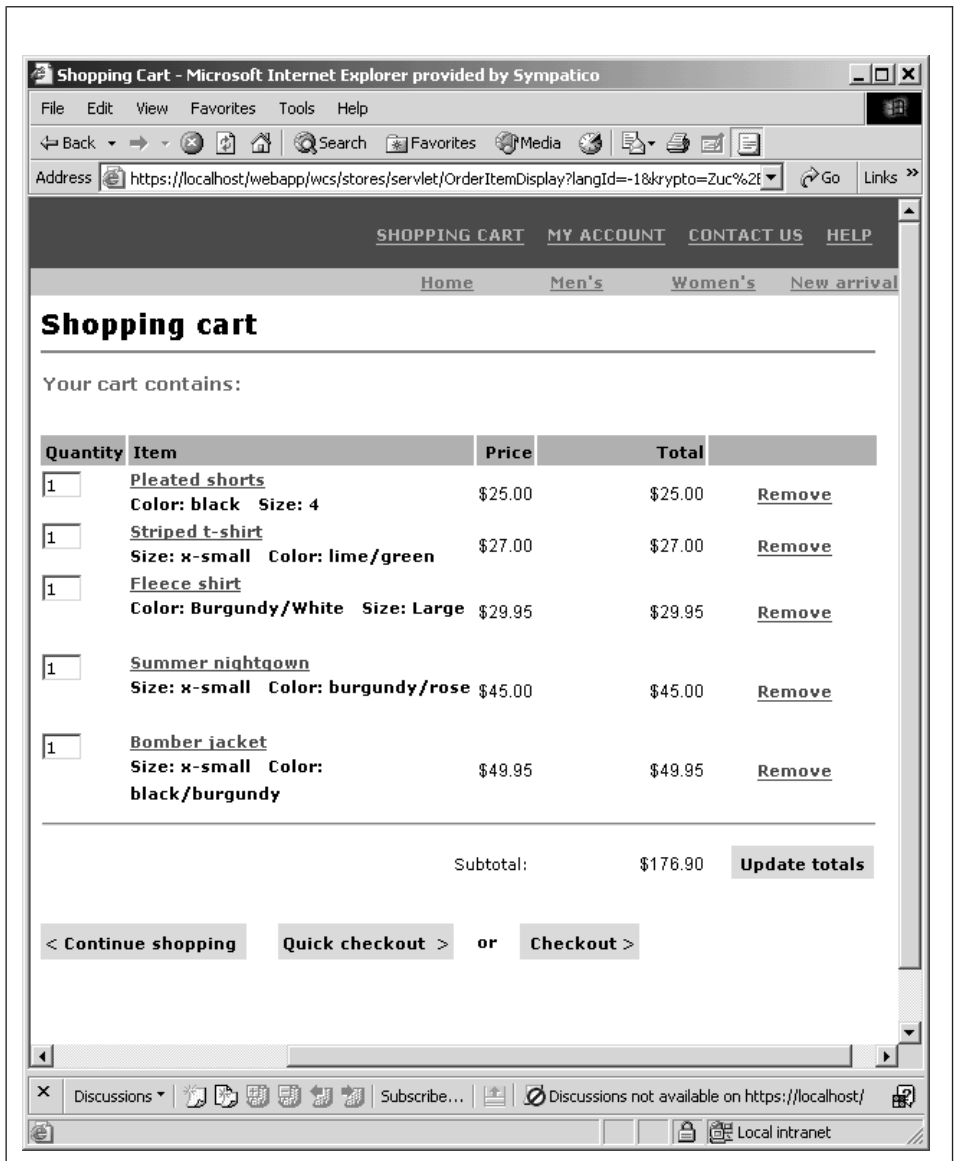


図 45.

- 「ショッピングの継続」をクリックして、別のアイテムを選択します。この選択アイテムについては、「ショッピング・カートに追加」をクリックします。以下のエラー・ページが表示されます。

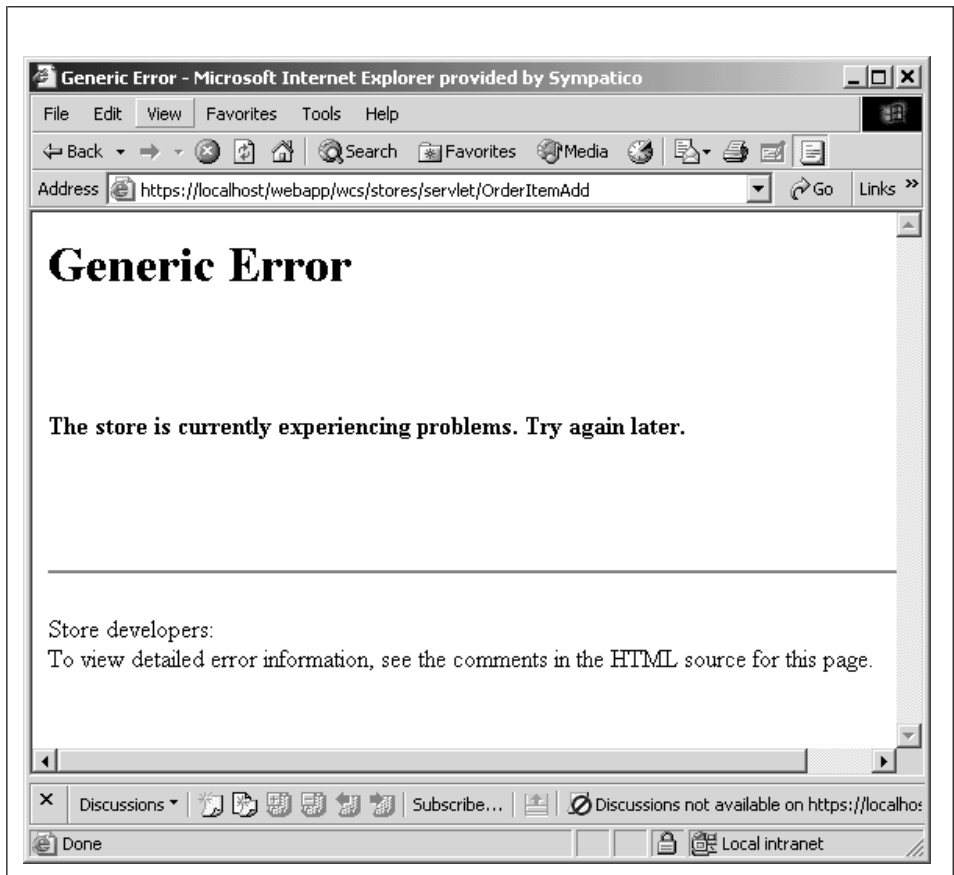


図 46.

エラー・ページのソースを表示します。ソースでは、スクロールダウンして、以下のエラー情報を見つめます。

Message Key: `_ERR_TOO_MANY_ITEMS`

Message: You are trying to place too many different items in one shopping cart

---

## MyOrderItemAddCmdImpl のデプロイメント

このステップでは、変更を加えたビジネス・ロジックをターゲットの WebSphere Commerce Server にデプロイメントします。ここでは、デプロイメントは以下のハイレベルなステップから成ります。

1. コマンド・ロジックおよびエラー・クラスを含む JAR ファイルを作成します。
2. エラー・メッセージ・プロパティ・ファイルをエクスポートします。
3. 資産をターゲット WebSphere Commerce Server へ転送します。

4. 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。
5. ターゲットの WebSphere Commerce Server 上の新規ロジックを妥当性検査します。

## コマンド JAR ファイルの作成



WebSphere Commerce コードのカスタマイズ手順の実行後に、カスタマイズしたコマンドと Data Bean は

WebSphereCommerceServerExtensionsLogic プロジェクトに入れられます。その結果、カスタマイズ済みのコードをデプロイメントする時点で、以前にカスタマイズしたコードを JAR ファイルに組み込むよう通知されます。たとえば、229 ページの『第 10 章 チュートリアル: ビジネス・ロジックの新規作成』を完了している場合、JAR ファイルを作成して MyOrderItemAddCmdImpl クラスをデプロイメントする際に、以前に作成した MyNewControllerCmd クラスとその他のクラスを組み込むよう通知されます。

MyOrderItemAddCmdImpl クラスを含む JAR ファイルを作成するには、開発マシンで以下のステップを実行します。

1. `drive:\ExportTemp2` というローカル・ファイル・システム上にディレクトリーを作成します。
2. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
3. **WebSphereCommerceServerExtensionsLogic** 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。  
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようにします。
  - a. 「JAR ファイル」を選択し、「次へ」をクリックします。
  - b. 「エクスポートするリソースの選択 (Select the resources to export)」の下の左側のペインに、プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。
  - c. 右側のペインで、以下のリソースだけが選択されていることを確認します。
    - .classpath
    - .project
    - .serverPreference
  - d. 「生成されたクラス・ファイルおよびリソースのエクスポート (Export generated class files and resources)」が選択されていることを確認します。
  - e. 「Java ソース・ファイルおよびリソースのエクスポート (Export Java source files and resources)」を選択しないでください。

- f. 「エクスポート先の選択 (**Select the export destination**)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。ここでは、  
`drive:¥ExportTemp2¥WebSphereCommerceServerExtensionsLogic.jar` と入力します。JAR ファイル名は `WebSphereCommerceServerExtensionsLogic.jar` でなければならないことに注意してください。
- g. 「終了」をクリックします。

## メッセージ・プロパティ・ファイルのエクスポート

このセクションでは、新しいメッセージのテキストを含むプロパティ・ファイルを、以下のようにしてエクスポートします。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 「ストア」フォルダーを拡張表示します。
3. 「**Web コンテンツ (Web Content)**」フォルダーを右クリックして、「エクスポート」を選択します。  
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようにします。
  - a. 「**ファイル・システム (File system)**」を選択して、「次へ」をクリックします。
  - b. 「**全選択解除 (Deselect All)**」をクリックします。
  - c. 以下のリソースをエクスポートすることを選択します。
    - `Web Content¥WEB-INF¥classes¥MyNewErrorMessages.properties`
  - d. 「**ファイルのディレクトリー構造の作成 (Create directory structure for files)**」を選択します。
  - e. 「ディレクトリー」フィールドに、これらのリソースを入れる一時ディレクトリーを入力します。たとえば、`C:¥ExportTemp2` と入力します。
  - f. 「終了」をクリックします。

## ターゲット WebSphere Commerce Server への資産の転送

このステップでは、ターゲットの WebSphere Commerce Server 上に一時ディレクトリーを作成してから、このディレクトリー中に `MyOrderItemAddCmdImpl` 資産をコピーします。

開発マシンからターゲットの WebSphere Commerce Server にファイルをコピーするには、以下のようにします。

1. ターゲットの WebSphere Commerce Server で、`drive:¥ImportTemp2` という一時ディレクトリーを作成します。



2. コンピューター間でファイルをコピーする方法を決めます。ターゲットの WebSphere Commerce Server 上のドライブを開発マシンにマッピングするか、または FTP アプリケーションが構成済みの場合はこのアプリケーションを使用して、コピーを実行できます。
3. 開発マシンから、`drive:¥ExportTemp2` の内容をターゲットの WebSphere Commerce Server の `drive:¥ImportTemp2` にコピーします。

## ターゲット WebSphere Commerce Server の停止

デプロイメント・ステップを開始する前に、ターゲット WebSphere Commerce Server を停止する必要があります。WebSphere Commerce Server 停止の詳細は、「WebSphere Commerce Studio インストール・ガイド」を参照してください。

## ターゲット WebSphere Commerce Server 上のデータベースの更新

このステップでは、コマンド・レジストリーに変更を加えて、新しい `MyOrderItemAddCmdImpl` インプリメンテーション・クラスが使用されるようにします。

**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 `MyOrderItemAddCmdImpl` を登録します。

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. ツール設定をクローズします。
5. 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 targetDB user dbuser using dbpassword;
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
WHERE INTERFACENAME=
'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
and storeent_Id=0;
```

ここで

- `targetDB` は、ターゲットの WebSphere Commerce Server で使用されるデータベースの名前
- `dbuser` は、データベースのユーザー
- `dbpassword` は、データベース・パスワード

「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 MyOrderItemAddCmdImpl を登録します。

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。

```
update CMDREG
set CLASSNAME='com.ibm.commerce.sample.commands.MyOrderItemAddCmdImpl'
WHERE INTERFACENAME=
'com.ibm.commerce.orderitems.commands.OrderItemAddCmd'
and storeent_Id=0;
```

Enter を押して SQL ステートメントを実行します。

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

## ターゲット WebSphere Commerce Server 上のコマンド JAR ファイルの更新

このステップでは、以下のようにして、新しい MyOrderItemAddCmdImpl を含む JAR ファイルを使用するように、ターゲットの WebSphere Commerce Server を更新します。

1. コマンド行で WebSphere Application Server stopServer コマンドを使用して、WebSphere Commerce インスタンスを停止します。必要であれば、このインスタンスの開始と停止の詳細は、使用しているプラットフォームおよびデータベースの「WebSphere Commerce インストール・ガイド」を参照してください。
2. 以下のようにして、既存の JAR ファイルのバックアップ・コピーを作成する必要があります。
  - a. WAS\_installdir¥installedApps¥cellName¥WC\_instanceName.ear ディレクトリーに移動します。ここで cellName は通常はマシンのホスト名です。
  - b. WebSphereCommerceServerExtensionsLogic.jar ファイルのコピーを作成して、バックアップ場所に保管します。

3. 新しい `WebSphereCommerceServerExtensionsLogic.jar` ファイルを、  
`drive:¥ImportTemp2` ディレクトリーから、  
`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーにコピー  
します。

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

## ターゲット WebSphere Commerce Server 上のメッセージ・プロパティの更新

このステップでは、新しいメッセージ・プロパティ・ファイルを、以下のようにしてアプリケーションに追加します。

1. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war` ディレクトリー (ここで、`cellName` はご使用のマシンのホスト名であることが多く、`instanceName` は WebSphere Commerce インスタンスの名前です) のバックアップを取ります。
2. `drive:¥ImportTemp¥Stores¥Web Content` ディレクトリーに移動します。
3. 「WEB-INF」フォルダーを以下のディレクトリーにコピーします。  
`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war`
4. コマンド行で WebSphere Application Server `startServer` コマンドを使用して、WebSphere Commerce インスタンスを再始動します。

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

## ターゲット WebSphere Commerce Server 上の MyOrderItemAddCmdImpl ロジックの検証

このステップでは、コードのデプロイメントを終えた時点で、即時検査を実行して、コードが良好に作動するか検証します。

コードを検証するには、以下のようにします。

1. Web ブラウザーをオープンし、FashionFlow ストアを立ち上げます。たとえば、以下の URL を入力してストアを立ち上げます。  
`http://hostname/webapp/wcs/stores/servlet/FashionFlow/index.jsp`  
`hostname` はご使用のインスタンスのホスト名です。
2. ストアでショッピングを行い、ショッピング・カートにアイテムを 5 つ追加します。5 つ目のアイテムを追加し終わると、ショッピング・カートは以下のようになります。

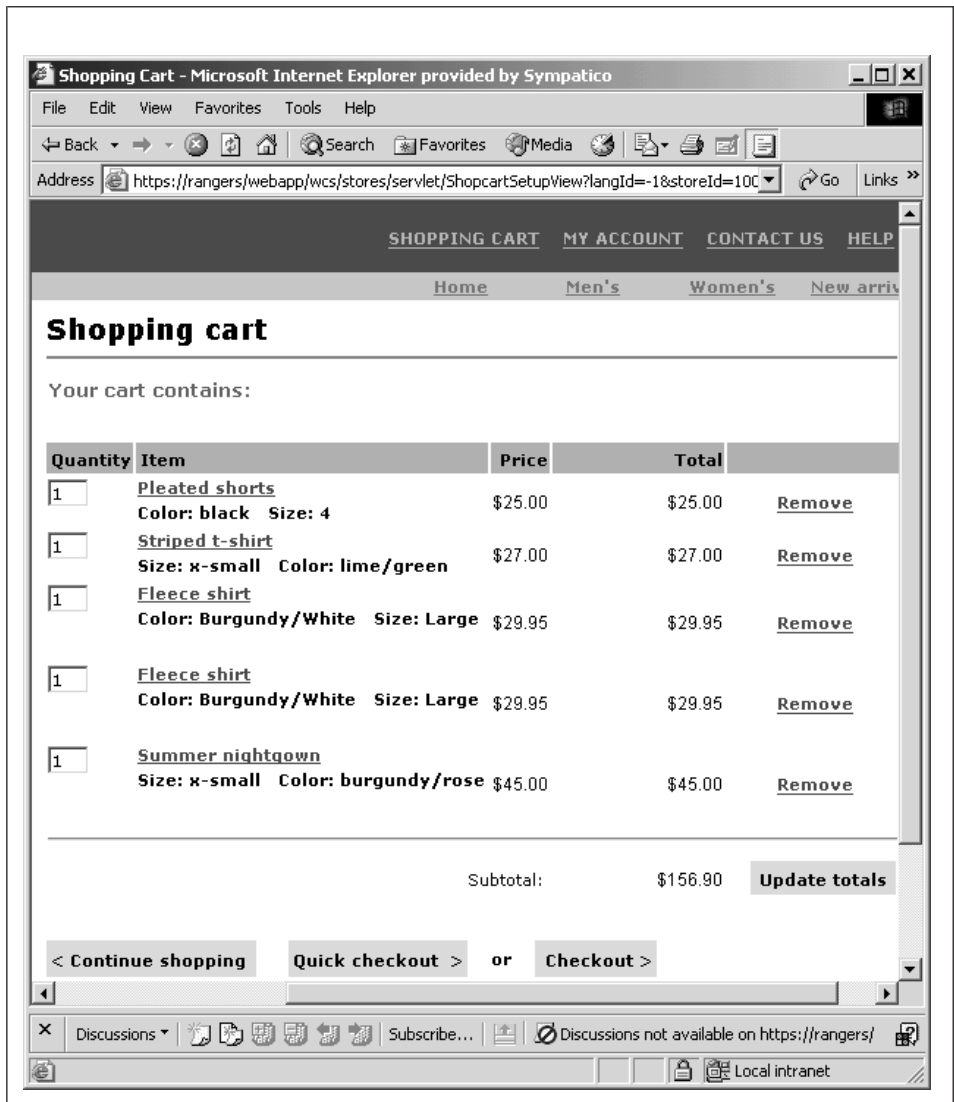


図 47.

3. 「ショッピングの継続」をクリックして、別のアイテムを選択します。この選択アイテムについては、「ショッピング・カートに追加」をクリックします。以下のエラー・ページが表示されます。

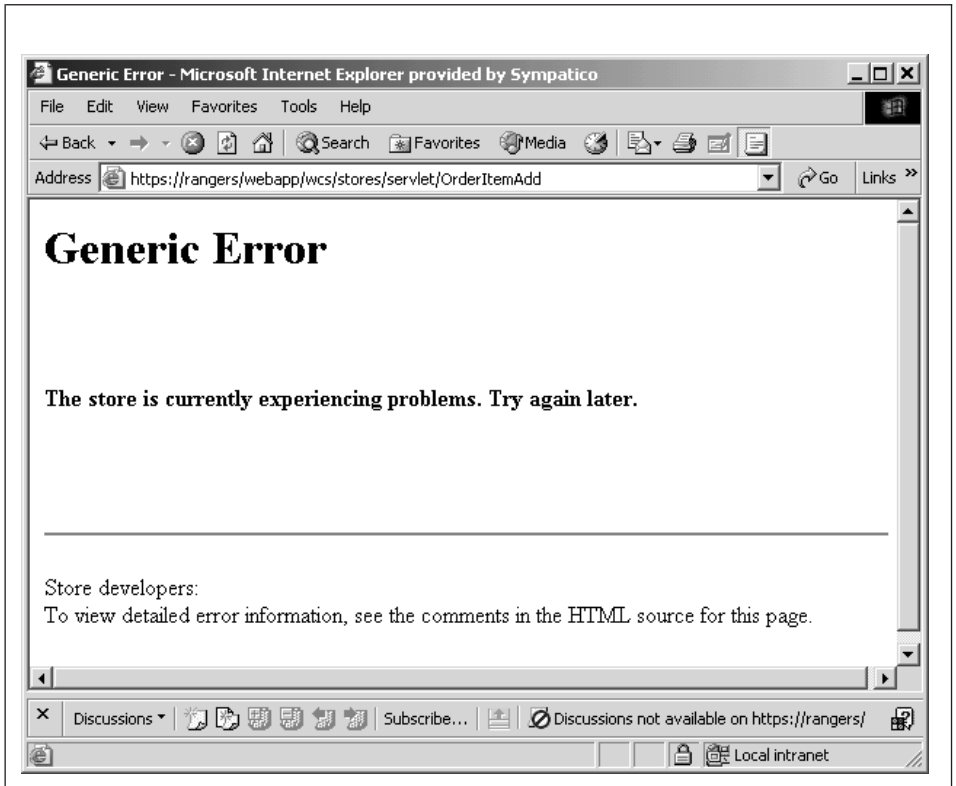


図 48.

エラー・ページのソースを表示します。ソースをスクロールダウンすると、エラーのメッセージ・キーが `_ERR_TOO_MANY_ITEMS` であることが分かります。



---

## 第 12 章 チュートリアル: オブジェクト・モデルの拡張および既存のタスク・コマンドの変更

このチュートリアルでは、オーダーのギフト・カード情報を収集する要件を考慮します。収集しなければならない情報には、宛先の名前、差出人の名前、2 つのメッセージが含まれます。この情報は、顧客がオーダーを送信する際に収集されます。

ギフト・カードに関する情報がデータベース中に保管されていくにつれて、新しいデータベース・テーブルが必要であることが分かってきます。このプロセスはオーダー・プロセスの延長なので、オブジェクト・モデルに変更を加える方法は 2 つあります。通常、既存の WebSphere Commerce public Entity Bean に変更を加えて、変更を加えた Bean 中の新規フィールドを新規テーブルにマップするか、または新規 Entity Bean を作成して新規テーブルに直接マップできます。前者の方法は Order Entity Bean を拡張する必要がありますが、この Bean は「更新用検索」タイプの SQL クエリーを使用するので、この方法でこの Bean を拡張することはできません。したがって、ここでは新規 Entity Bean を作成して新規テーブルにマップしなければなりません。

新規 Entity Bean に加えて、既存の ExtOrderProcessCmdImpl タスク・コマンドも拡張されます。この拡張により、新規テーブルに対応する新規 Data Bean をインスタンス化したり、データベース中のギフト情報を更新したりします。

このチュートリアルには以下のハイレベルなタスクが含まれています。

1. 新規 XORDGIFT テーブルを作成して取り込みます。
2. 新規 OrderGift Entity Bean を作成します。
3. XORDGIFT スキーマを作成します。
4. テーブル定義を作成し、OrderGift Enterprise Bean 中のフィールドを、XORDGIFT テーブル中の列にマップします。
5. OrderGift Bean 用のデプロイメント・コードと Access Bean を生成します。
6. 新規 OrderGiftDataBean を作成します。
7. 新規 MyExtOrderProcessCmdImpl タスク・コマンドのインプリメンテーションを作成します。
8. メッセージ情報を変更するように、OrderSubmitForm.jsp に変更を加え、メッセージ情報を表示するように、OrderDetailDisplayForm.jsp に変更を加えます。
9. 変更を加えたコードをテストします。

---

## 前提条件

この章のチュートリアルは、これまでのチュートリアルを完了している必要はありません。これまでのチュートリアルを完了している場合でも、そのコードはこの章のチュートリアルと競合しないので、コードをワークスペース中に残しておいても害はありません。

この章のチュートリアルを開始する前に、FashionFlow サンプル・ストアを基にしたストアを発行していなければなりません。このストア中で購入を完了できなければなりません (カタログのブラウズ、ショッピング・カートへのアイテムの追加、オーダーの確認のチェックアウトや表示など)。

---

## XORDGIFT テーブルの作成と値の取り込み

このステップでは、XORDGIFT テーブルを作成します。

**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」>「コマンド行ツール (Command Line Tools)」>「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 developmentDB user dbuser using dbpassword;
create table XORDGIFT (ORDERSID bigint not null, RECEIPTNAME varchar(50),
SENDERNAME varchar(50), MSGFIELD1 varchar(50), MSGFIELD2 varchar(50),
constraint p_xordgift primary key (ORDERSID),
constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)
on delete cascade);
insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード

「実行」アイコンをクリックします。

これで、XORDGIFT テーブルが作成されました。

**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 XORDGIFT (ORDERSID NUMBER NOT NULL, RECEIPTNAME VARCHAR(50),
SENDERNAME VARCHAR(50), MSGFIELD1 VARCHAR(50), MSGFIELD2 VARCHAR(50),
constraint p_xordgift primary key (ORDERSID),
constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)
on delete cascade);
insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
```

それから Enter を押して SQL ステートメントを実行します。 XORDGIFT 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、 XORDGIFT 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 XORDGIFT;
```

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

---

## OrderGift Entity Bean の作成

データベース・テーブルが作成されたら、新規 Entity Bean の作成を始めることができます。次のステップでは WebSphere Studio Application Developer を使用してこの Bean を作成します。

次に以下のようにして、OrderGift Bean を新規作成します。

1. WebSphere Commerce Studio を始動します (「スタート」 > 「プログラム」 > 「IBM WebSphere Commerce Studio」 > 「WebSphere Commerce 開発環境」)。
2. J2EE パースペクティブをオープンします。
3. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、「EJB モジュール (EJB Modules)」を拡張表示します。
4. **WebSphereCommerceServerExtensionsData** モジュールを右クリックして、「新規」 > 「Enterprise Bean」を選択します。  
「Enterprise Bean の作成 (Enterprise Bean Creation)」ウィザードがオープンします。
5. 「EJB プロジェクト (EJB Project)」ドロップダウン・リストで、「WebSphereCommerceServerExtensionsData」を選択して、「次へ」をクリックします。
6. 「Enterprise Bean の作成」ウィンドウで、以下のようにします。

- a. 「コンテナ管理パーシスタンス (CMP) フィールドを持つ **Entity Bean**」を選択します。
  - b. 「**Bean 名 (Bean name)**」フィールドに、 OrderGift と入力します。
  - c. 「ソース・フォルダー (**Source folder**)」フィールドを、指定されているデフォルト値 (ejbModule) のままにします。
  - d. 「デフォルト・パッケージ (**Default package**)」フィールドに、com.ibm.commerce.extension.objects と入力します。
  - e. 「次へ」をクリックします。
7. 「CMP 属性 (CMP Attributes)」ウィンドウで、以下のようにします。
- a. 「追加」をクリックして、以下の列に関する新規フィールドをテーブル中に追加します。
    - ORDERSID
    - RECEIPTNAME
    - SENDERNAME
    - MSGFIELD1
    - MSGFIELD2
- 「CMP 属性の作成 (Create CMP Attribute)」ウィンドウがオープンします。このウィンドウで、以下のようにします。
- 1) 以下のようにして、ordersId フィールドを作成します。

表 12.

パラメーター名	パラメーター値
名前	ordersId
タイプ	java.lang.Long 注: long データ・タイプではなく、java.lang.Long データ・タイプを使用してください。
鍵フィールド	選択

「適用」をクリックします。

- 2) 以下のようにして、receiptName フィールドを作成します。

表 13.

パラメーター名	パラメーター値
名前	receiptName
タイプ	java.lang.String
getter および setter メソッドを使ったアクセス	選択

表 13. (続き)

パラメーター名	パラメーター値
<b>getter</b> および <b>setter</b> 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	クリア

「適用」をクリックします。

- 3) 以下のようにして、senderName フィールドを作成します。

表 14.

パラメーター名	パラメーター値
名前	senderName
タイプ	java.lang.String
<b>getter</b> および <b>setter</b> メソッドを使ったアクセス	選択
<b>getter</b> および <b>setter</b> 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	クリア

「適用」をクリックします。

- 4) 以下のようにして、msgField1 フィールドを作成します。

表 15.

パラメーター名	パラメーター値
名前	msgField1
タイプ	java.lang.String
<b>getter</b> および <b>setter</b> メソッドを使ったアクセス	選択
<b>getter</b> および <b>setter</b> 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	クリア

「適用」をクリックします。

- 5) 以下のようにして、msgField2 フィールドを作成します。

表 16.

パラメーター名	パラメーター値
名前	msgField2
タイプ	java.lang.String
<b>getter</b> および <b>setter</b> メソッドを使ったアクセス	選択



表 16. (続き)

パラメーター名	パラメーター値
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	クリア

「適用」をクリックします。

- 6) 「クローズ」をクリックしてこのウィンドウをクローズします。
- b. 「鍵クラスに単一の鍵属性タイプを使用 (Use the single key attribute type for the key class)」チェック・ボックスをクリアしてから、「次へ」をクリックします。
8. 「EJB Java クラスの詳細 (EJB Java Class Details)」ウィンドウで、以下のようになります。
  - a. Bean のスーパークラスを選択するには、「ブラウズ」をクリックします。「タイプの選択 (Type Selection)」ウィンドウがオープンします。
  - b. 「使用するクラスの選択: (任意) (Select a class using: (any))」フィールドに、ECEntityBean と入力し、「OK」をクリックします。スーパークラスとして com.ibm.commerce.base.objects.ECEntityBean が選択されます。
  - c. 「終了」をクリックします。

新規 Bean の分離レベルを設定するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示します。
2. **WebSphereCommerceServerExtensionsData** プロジェクトをダブルクリックし、Deployment Descriptor Editor を使用してオープンします。
3. 「アクセス」タブをクリックします。
4. 「分離レベル (Isolation Level)」テキスト・ボックスの隣の「追加」をクリックします。「分離レベルの追加 (Add Isolation Level)」ウィンドウがオープンします。
5.  「反復可能読み取り (Repeatable Read)」を選択してから、「次へ」をクリックします。  
 「コミット済み読み取り (Read Committed)」を選択してから、「次へ」をクリックします。
6. **OrderGift** Bean を選択してから、「次へ」をクリックします。
7. **OrderGift** を選択してそのメソッドをすべて選択し、「終了」をクリックします。
8. ここまでの作業を保管します (Ctrl + S)。

次に、以下のようにして、Bean のセキュリティー ID を設定します。

1. Deployment Descriptor Editor で、「アクセス」タブを選択します。

- (メソッド・レベルの)「セキュリティー ID (Security Identity)」テキスト・ボックスの隣の「追加」をクリックします。  
「セキュリティー ID の追加 (Add Security Identity)」ウィンドウがオープンします。
- 「EJB サーバーの ID の使用 (Use identity of EJB server)」を選択してから、「次へ」をクリックします。
- OrderGift** Bean を選択してから、「次へ」をクリックします。
- OrderGift** を選択してそのメソッドをすべて選択し、「終了」をクリックします。
- ここまでの作業を保管し (Ctrl + S)、エディターはオープンしたままにします。

次に、以下のようにして、Bean 内のメソッドのセキュリティー役割を設定します。

- Deployment Descriptor Editor で、「アセンブリー記述子 (Assembly Descriptor)」タブを選択します。
- 「メソッド許可 (Method Permissions)」セクションで、「追加」をクリックします。
- セキュリティー役割として **WCSecurityRole** を選択して、「次へ」をクリックします。
- 見つかった Bean のリストから **OrderGift** を選択し、「次へ」をクリックします。
- 「メソッド・エレメント (Method elements)」ページで、「すべてに適用 (Apply to All)」をクリックし、「終了」をクリックします。
- ここまでの作業を保管し (Ctrl + S)、Deployment Descriptor Editor をクローズします。

次のステップとして、WebSphere Studio Application Developer が生成するエンティティ・コンテキストに関連したフィールドとメソッドを除去します。これらのフィールドを削除する必要がある理由は、EJEntityBean の基本クラスにはこれらのメソッドの独自のインプリメンテーションがあるからです。生成されたエンティティ・コンテキストのフィールドとメソッドを削除するには、以下のようにします。

- 「J2EE 階層 (J2EE Hierarchy)」ビューで、**WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
- OrderGift** EJB モジュールを拡張表示してから、「**OrderGiftBean**」をダブルクリックします。
- 「概略 (Outline)」ビューで、以下のようにします。
  - 「**myEntityCtx**」フィールドを右クリックし、「削除」を選択します。
  - getEntityContext()** メソッドを右クリックし、「削除」を選択します。
  - setEntityContext(EntityContext)** メソッドを右クリックし、「削除」を選択します。
  - unsetEntityContext()** メソッドを右クリックし、「削除」を選択します。
- ここまでの作業を保管します (Ctrl + S)。

新規 OrderGift Bean が作成されるときにすべてのパラメーターが明示的に設定されるように、生成された ejbCreate メソッドを以下のようにして変更する必要があります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、**OrderGiftBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 「概略 (Outline)」ビューで、ejbCreate(Long) を選択します。このソース・コードは、以下のとおりです。

```
public com.ibm.commerce.extension.objects.OrderGiftKey
 ejbCreate(java.lang.Long ordersId)
 throws javax.ejb.CreateException {
 _initLinks();
 this.ordersId = ordersId;
 return null;
}
```

3. コードを以下のように変更します。

```
public com.ibm.commerce.extension.objects.OrderGiftKey
 ejbCreate(java.lang.Long ordersId)
 throws javax.ejb.CreateException {
 _initLinks();
 this.ordersId = ordersId;
 this.senderName = null;
 this.receiptName = null;
 this.msgField1 = null;
 this.msgField2 = null;
 return null;
}
```

4. コードの変更内容を保管します。

次に以下のようにして、OrderGift Bean に新規 ejbCreate メソッドを追加します。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、**OrderGiftBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 以下のコードをクラスに追加して、新規 ejbCreate(Long, String, String, String, String) メソッドを作成します。

```
public com.ibm.commerce.extension.objects.OrderGiftKey
 ejbCreate(java.lang.Long ordersId,
 java.lang.String receiptName,
 java.lang.String senderName,
 java.lang.String msgField1,
 java.lang.String msgField2)
 throws javax.ejb.CreateException {
 _initLinks();
 this.ordersId = ordersId;
 this.senderName = senderName;
 this.receiptName = receiptName;
 this.msgField1 = msgField1;
 this.msgField2 = msgField2;
 return null;
}
```

3. コードの変更内容を保管します。
4. ホーム・インターフェースにこの新規 `ejbCreate` メソッドを追加しなければなりません。これにより、生成された `Access Bean` でメソッドを使用できるようになります。ホーム・インターフェースにメソッドを追加するには、以下のようになります。
  - a. 「概略 (Outline)」ビューで **`ejbCreate(Long, String, String, String, String)`** メソッドを右クリックし、「Enterprise Bean」>「ホーム・インターフェースへのプロモート (Promote to Home Interface)」を選択します。

次に、以下のようにして、`ejbPostCreate(Long, String, String, String, String)` メソッドを新規作成し、`ejbCreate(Long, String, String, String, String)` メソッドと同じパラメーターになるようにします。

1. **OrderGiftBean** クラスをダブルクリックしてオープンし、そのソース・コードを表示します。
2. 以下のコードをクラスに追加して、新規 `ejbPostCreate(Long ordersId, String receiptName, String senderName, String msgField1, String msgField2)` メソッドを作成します。

```
public void ejbPostCreate(
 java.lang.Long ordersId,
 java.lang.String receiptName,
 java.lang.String senderName,
 java.lang.String msgField1,
 java.lang.String msgField2)
 throws javax.ejb.CreateException {
}
```

3. コードの変更内容を保管します。

次に、`XORDGIFT` テーブルの定義を作成しなければなりません。229 ページの『第 10 章 チュートリアル: ビジネス・ロジックの新規作成』をまだ完了していない場合は、以下のようにして `XORDGIFT` テーブル定義を作成します。

1. データ・パースペクティブをオープンし、「データ定義 (Data Definition)」ビューに切り替えます。
2. 以下のディレクトリーに移動します。  
「WebSphereCommerceServerExtensionsData」>「ejbModule」>  
「META-INF」
3. **META-INF** を右クリックして、「新規データベース定義 (New database definition)」を選択します。  
「新規データベース定義 (New Database Definition)」ウィザードがオープンします。
4. 「データベース名」フィールドに、開発データベースの名前を入力します。たとえば `Demo_Dev` と入力します。

- 「データベース・メーカー・タイプ (Database vendor type)」ドロップダウン・リストで、ご使用の開発環境に適したデータベース・タイプを選択します。



▶ DB2 **DB2 Universal Database V8.1**

▶ Oracle **Oracle 9i**

- 「終了」をクリックします。新規データベース定義が作成されました。
- 以下のディレクトリーに移動します。  
「WebSphereCommerceServerExtensionsData」 > 「ejbModule」 > 「META-INF」 > 「Schema」 > 「developmentDB」
- developmentDB を右クリックして、「新規」 > 「新規スキーマ定義 (New schema definition)」を選択します。  
「新規スキーマ定義 (New Schema Definition)」ウィザードがオープンします。
- 「スキーマ名 (Schema name)」フィールドに NULLID と入力して、「終了」をクリックします。
- 「WebSphereCommerceServerExtensionsData」 > 「ejbModule」 > 「META-INF」 > 「Schema」 > 「developmentDB」 > 「NULLID」 > 「Tables」 に移動します。
- Tables を右クリックして、「新規テーブル定義 (New table definition)」を選択します。  
「新規テーブル定義 (New Table Definition)」ウィザードがオープンします。
- 「テーブル名 (Table name)」フィールドに XORDGIFT と入力して、「次へ」をクリックします。
- 次に、以下のようにして、キー列をテーブル定義に追加しなければなりません。
  - 「別のものを追加 (Add Another)」をクリックします。
  - 「列名」フィールドに、ORDERSID と入力します。
  - 「列タイプ」ドロップダウン・リストで、以下を選択します。  
▶ DB2 「BIGINT」  
▶ Oracle 「NUMBER」
  - 「キー列 (Key column)」を選択します。
  - ▶ Oracle 「数値の精度 (Numeric precision)」フィールドで、38 と入力します。
  - ▶ Oracle 「数値のスケール (Numeric Scale)」の値は 0 のままにします。
- 次に、以下のようにして、追加列をテーブル定義に追加します。
  - 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。





表 17.

プロパティ	値
列名	RECEIPTNAME
列タイプ	 DB2 VARCHAR  Oracle VARCHAR2
ヌル可能 (Nullable)	選択
文字列の長さ (String length)	50
ビット・データ用	クリア



- b. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 18.

プロパティ	値
列名	SENDERNAME
列タイプ	 DB2 VARCHAR  Oracle VARCHAR2
ヌル可能 (Nullable)	選択
文字列の長さ (String length)	50
ビット・データ用	クリア

- c. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 19.

プロパティ	値
列名	MSGFIELD1
列タイプ	 DB2 VARCHAR  Oracle VARCHAR2
ヌル可能 (Nullable)	選択
文字列の長さ (String length)	50
ビット・データ用	クリア

- d. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 20.

プロパティ	値
列名	MSGFIELD2
列タイプ	<div style="display: flex; align-items: center; margin-bottom: 5px;"> <span style="background-color: #4CAF50; color: white; padding: 2px 5px; font-weight: bold;">DB2</span> VARCHAR         </div> <div style="display: flex; align-items: center;"> <span style="background-color: #F44336; color: white; padding: 2px 5px; font-weight: bold;">Oracle</span> VARCHAR2         </div>
ヌル可能 (Nullable)	選択
ストリングの長さ (String length)	50
ビット・データ用	クリア

- e. 「終了」をクリックします。
- f. Oracle テキスト・エディターを使用し、テーブル定義を以下のように編集する必要があります。
  - 1) 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
  - 2) **WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
  - 3) 以下のように拡張表示します。「**ejbModule**」 > 「**META-INF**」 > 「スキーマ」。
  - 4) **WebSphereCommerceServerExtensionsData\_NULL\_XORDGIFT.xmi** ファイルを右クリックして、「オープンに使用 (Open With)」 > 「テキスト・エディター (Text Editor)」を選択します。
  - 5) SQLNumeric\_6 が出現するすべての箇所を、SQLNumeric\_3 に置き換えます。
  - 6) 変更内容を保管し、テキスト・エディターをクローズします。

229 ページの『第 10 章 チュートリアル: ビジネス・ロジックの新規作成』を完了している場合は、以下のようにして XORDGIFT テーブル定義を作成します。

1. データ・パースペクティブをオープンし、「データ定義 (Data Definition)」ビューに切り替えます。
2. 以下のディレクトリーに移動します。  
**「WebSphereCommerceServerExtensionsData」** > **「ejbModule」** >  
**「META-INF」** > **「Schema」** > **「developmentDB」** > **「NULLID」** >  
**「Tables」**
3. **Tables** ディレクトリーを右クリックして、「新規」 > 「新規テーブル定義 (New Table Definition)」を選択します。  
「新規テーブル定義 (New Table Definition)」ウィザードがオープンします。
4. 「**テーブル名 (Table name)**」フィールドに XORDGIFT と入力して、「次へ」をクリックします。
5. 次に、以下のようにして、キー列をテーブル定義に追加しなければなりません。









- a. 「別のものを追加 (Add Another)」をクリックします。
  - b. 「列名」フィールドに、 ORDERSID と入力します。
  - c. 「列タイプ」ドロップダウン・リストで、以下を選択します。
    -  「BIGINT」
    -  「NUMBER」
  - d. 「キー列 (Key column)」を選択します。
  - e.  「数値の精度 (Numeric precision)」フィールドで、 38 と入力します。
  - f.  「数値のスケール (Numeric Scale)」の値は 0 のままにします。
6. 次に、以下のようにして、追加列をテーブル定義に追加します。
- a. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 21.

プロパティ	値
列名	RECEIPTNAME
列タイプ	 VARCHAR  VARCHAR2
ヌル可能 (Nullable)	選択
文字列の長さ (String length)	50
ビット・データ用	クリア

- b. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 22.

プロパティ	値
列名	SENDERNAME
列タイプ	 VARCHAR  VARCHAR2
ヌル可能 (Nullable)	選択
文字列の長さ (String length)	50
ビット・データ用	クリア

- c. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 23.

プロパティ	値
列名	MSGFIELD1
列タイプ	<div style="display: flex; align-items: center; margin-bottom: 5px;"> <span style="background-color: #4CAF50; color: white; padding: 2px 5px; margin-right: 5px;">DB2</span> VARCHAR         </div> <div style="display: flex; align-items: center;"> <span style="background-color: #F44336; color: white; padding: 2px 5px; margin-right: 5px;">Oracle</span> VARCHAR2         </div>
ヌル可能 (Nullable)	選択
文字列の長さ (String length)	50
ビット・データ用	クリア

- d. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 24.

プロパティ	値
列名	MSGFIELD2
列タイプ	<div style="display: flex; align-items: center; margin-bottom: 5px;"> <span style="background-color: #4CAF50; color: white; padding: 2px 5px; margin-right: 5px;">DB2</span> VARCHAR         </div> <div style="display: flex; align-items: center;"> <span style="background-color: #F44336; color: white; padding: 2px 5px; margin-right: 5px;">Oracle</span> VARCHAR2         </div>
ヌル可能 (Nullable)	選択
文字列の長さ (String length)	50
ビット・データ用	クリア

- e. 「終了」をクリックします。
7. Oracle テキスト・エディターを使用し、テーブル定義を以下のように編集する必要があります。
- a. 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
  - b. **WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
  - c. 以下のように拡張表示します。「**ejbModule**」>「**META-INF**」>「**スキーマ**」。
  - d. **WebSphereCommerceServerExtensionsData\_NULL\_XORDGIFT.xmi** ファイルを右クリックして、「**オープンに使用 (Open With)**」>「**テキスト・エディター (Text Editor)**」を選択します。
  - e. SQLNumeric\_6 が出現するすべての箇所を、SQLNumeric\_3 に置き換えます。
  - f. 変更内容を保管し、テキスト・エディターをクローズします。

次のステップとして、XORDGIFT テーブルを OrderGiftBean Entity Bean にマップします。開発データベースと XORDGIFT テーブルが既存なので、meet-in-the-middle マッピングを使用します。

229 ページの『第 10 章 チュートリアル: ビジネス・ロジックの新規作成』を完了していない場合は、以下のようにしてマッピングを作成します。

1. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、  
**WebSphereCommerceServerExtensionsData** プロジェクトを右クリックし、「生成 (Generate)」>「EJB 対 RDB のマッピング (EJB to RDB Mapping)」を選択します。  
「新規 EJB/RDB マッピングの作成 (Create new EJB/RDB Mapping)」ウィンドウがオープンします。
2. 「meet-in-the-middle (Meet In The Middle)」を選択し、「次へ」をクリックします。
3. 「名前およびタイプ別に突き合わせ (Match By Name, and Type)」を選択し、「終了」をクリックします。  
Map.mapxmi エディターがオープンします。
4. 「Enterprise Bean」ペインで、**OrderGift Bean** を拡張表示します。「テーブル (Tables)」ペインで、**XORDGIFT** テーブルを拡張表示します。
5. 以下のようにして、Bonus Bean 中のフィールドを、XORDGIFT テーブル中の列にマップします。
  - a. **OrderGift Bean** を右クリックして、「名前別に突き合わせ (Match By Name)」を選択します。
6. Ctrl + S を押して Map.mapxmi ファイルを保管します。ファイルをクローズします。

229 ページの『第 10 章 チュートリアル: ビジネス・ロジックの新規作成』を完了している場合は、以下のようにしてマッピングを作成します。

1. 「J2EE 階層 (J2EE Hierarchy)」ビュー内で、  
**WebSphereCommerceServerExtensionsData** プロジェクトを右クリックし、「生成 (Generate)」>「EJB 対 RDB のマッピング (EJB to RDB Mapping)」を選択します。  
Map.mapxmi エディターがオープンします。
2. 「Enterprise Bean」ペインで、**OrderGift Bean** を拡張表示します。「テーブル (Tables)」ペインで、**XORDGIFT** テーブルを拡張表示します。
3. 以下のようにして、Bonus Bean 中のフィールドを、XORDGIFT テーブル中の列にマップします。
  - a. **OrderGift Bean** を右クリックして、「名前別に突き合わせ (Match By Name)」を選択します。
4. Ctrl + S を押して Map.mapxmi ファイルを保管します。ファイルをクローズします。

OrderGiftBean エンティティが作成され、スキーマが正しくマップされたら、Entity Bean 用の Access Bean を作成できます。この Access Bean は、OrderGift エンティティ

イ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 Access Bean を生成するために、WebSphere Studio Application Developer 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、Access Bean によって使用されることとなります。) OrderGift Entity Bean 用の Access Bean を作成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示してから、**WebSphereCommerceServerExtensionsData** を右クリックし、「新規」>「Access Bean」を選択します。  
「Access Bean の追加 (Add an Access Bean)」ウィンドウがオープンします。
2. 「コピー・ヘルパー (Copy Helper)」を選択して、「次へ」をクリックします。
3. **OrderGift** Bean を選択して、「次へ」をクリックします。
4. コンストラクター・メソッドのドロップダウン・リストから、**findByPrimaryKey(com.ibm.commerce.extension.objects.OrderGiftKey)** を選択します。
5. 「属性ヘルパー (Attribute Helpers)」セクションで、すべての属性を選択します。
6. 「終了」をクリックします。

「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えてから、**WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示し、「**ejbModule**」サブフォルダーを拡張表示して、**com.ibm.commerce.extension.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。OrderGiftAccessBean という新しいクラスと、OrderGiftAccessBeanData という新しいインターフェースが作成されて、パッケージ中に表示されます。

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 サーバーの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 Enterprise 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 アクセスが必要とされる)、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。

デプロイメント・コードを生成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「EJB モジュール (EJB Modules)」を拡張表示してから、**WebSphereCommerceServerExtensionsData** を右クリックし、「生成 (Generate)」>「デプロイメントおよび RMIC コード (Deploy and

**RMIC Code)」**を選択します。

「デプロイメントおよび RMIC コード (Deploy and RMIC Code)」ウィンドウがオープンします。

2. 「**OrderGift**」を選択して、「**終了**」をクリックします。

「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えると、新しく生成したコードを表示できます。以下のように表示されます。

表 25.

コードのタイプ	クラス名
コンテナ・インプリメンテーション生成コード	EJSCMPOrderGiftHomeBean.java
	EJSRemoteCMPOrderGift.java
	EJSRemoteCMPOrderGiftHome.java
	EJSFinderOrderGiftBean.java
JDBC アクセス・コード	EJSJDBCPersisterCMPOrderGiftBean.java
RMI タイおよびスタブ・コード	_EJSRemoteCMPOrderGift_Tie.java
	_OrderGift_Stub.java
	_EJSRemoteCMPOrderGiftHome_Tie.java
	_OrderGiftHome_Stub.java

---

## OrderGift Entity Bean のショッピング・フローへの統合

このセクションでは、以下のようにして、OrderGift Entity Bean をサンプル・ストアの正規のショッピング・フローに統合します。

1. OrderGiftDataBean Data Bean を新規作成します。
2. MyExtOrderProcessCmdImpl タスク・コマンドを新規作成します。
3. OrderSubmitForm.jsp 表示ページに変更を加えます。

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。

### OrderGiftDataBean の作成

OrderGiftDataBean を作成して、OrderGift Entity Bean の属性を JSP 表示ページに表示できるようにしなければなりません。チュートリアル他の部分と同様に、基本コードが備えられているので、コードのさまざまなセクションをコメント解除する必要があります。

OrderGiftDataBean を作成するには、以下のようにします。

1. 最初のステップとして、以下のように新規 Data Bean の基本コードをインポートします。

- a. 230 ページの『サンプル・コードの場所』のステップを完了していることを確認します。
- b. J2EE パースペクティブに切り替えてから、「J2EE ナビゲーター (J2EE Navigator)」ビューを選択します。
- c. **WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。
- d. 「**src**」フォルダーを右クリックして、「**インポート (Import)**」を選択します。「インポート (Import)」ウィザードがオープンします。
- e. 「**インポート・ソースの選択 (Select an import source)**」リストで、「**ZIP ファイル (Zip file)**」を選択し、「**次へ**」をクリックします。
- f. 「**ブラウザ**」(「**ZIP ファイル (Zip file)**」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、`yourDirectory\WC_SAMPLE_55.zip` にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
- g. 「**全選択解除 (Deselect All)**」をクリックし、ディレクトリーを拡張表示して、インポートする以下のファイルを選択します。
  - `com\ibm\commerce\sample\databaseans\OrderGiftDataBean.java`
- h. 「**フォルダー (Folder)**」フィールドでは、「`WebSphereCommerceServerExtensionsLogic/src`」フォルダーがすでに指定されています。この値をそのまま使用します。
- i. 「**終了**」をクリックします。

Bean のコードをインポートしたら、コードを調べます。

## MyExtOrderProcessCmdImpl クラスの作成

このセクションでは、`OrderProcess` ビジネス・プロセスの末尾に新規ロジックを追加し、`XORDGIFT` データベース・テーブル中のギフト・オーダーに関連した情報を更新できるようにします。`ExtOrderProcessCmdImpl` コマンドは、このビジネス・プロセスに対する拡張点として備えられています。プログラミング・モデルに従って、このロジックを拡張するには、タスク・コマンドのインプリメンテーション・クラスを新規作成し、このクラス中に新規ロジックを組み込みます。さらに、コマンド・レジストリーを更新して、新しいインプリメンテーション・クラスを `ExtOrderProcessCmd` インターフェースに関連付けなければなりません。

`MyExtOrderProcessCmdImpl` クラスを作成するには、以下のようになります。

1. 最初のステップとして、以下のように新規コマンドの基本コードをインポートします。
  - a. **WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。



- b. 「src」フォルダーを右クリックして、「インポート (Import)」を選択します。  
「インポート (Import)」ウィザードがオープンします。
- c. インポート・ソースとして「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
- d. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
- e. 「ブラウザ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、  
`yourDirectory\WC_SAMPLE_55.zip`  
にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
- f. 「全選択解除 (Deselect All)」をクリックし、ディレクトリーを拡張表示して、インポートする以下のファイルを選択します。
  - `com\ibm\commerce\sample\commands\MyExtOrderProcessCmdImpl.java`
- g. 「フォルダー (Folder)」フィールドでは、「WebSphereCommerceServerExtensionsLogic/src」フォルダーがすでに指定されています。この値をそのまま使用します。
- h. 「終了」をクリックします。

この新規コマンドのコードをインポートしたら、ソースを調べてコマンドの実行内容を確認できます。コマンドは、スーパークラスの `performExecute()` メソッドを呼び出して、そのコマンドからのすべての処理を実行する点に注意してください。さらに、ギフト・オーダー情報を新規 `Data Bean` に設定するロジックも含まれています。

このステップでは、コマンド・レジストリーに変更を加えて、元の `ExtOrderProcessCmdImpl` インプリメンテーション・クラスの代わりに新しい `MyExtOrderProcessCmdImpl` インプリメンテーション・クラスが使用されるようにします。コマンド・レジストリー中のテーブルのうち変更を加える必要があるのは、`CMDREG` テーブルのみです。ここでは、すべてのストアで新規インプリメンテーション・クラスを使用します。

コマンド・レジストリーを変更するには、以下のようにします。

**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 `MyExtOrderProcessCmdImpl` を登録します。


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」 > 「コマンド行ツール (Command Line Tools)」 > 「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. ツール設定をクローズします。
5. 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 developmentDB user dbuser using dbpassword;
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
 CLASSNAME, TARGET)
values (FashionFlow_storeent_ID,
 'com.ibm.commerce.order.commands.ExtOrderProcessCmd',
 'This is a new task command for tutorial two.',
 'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImpl',
 'local');
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
  - *dbuser* は、データベースのユーザー
  - *dbpassword* は、データベースのユーザーのパスワード
  - *FashionFlow\_storeent\_ID* は、サンプル・ストアのストア ID
- 「実行」アイコンをクリックします。

 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 MyOrderItemAddCmdImpl を登録します。

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。

```
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
 CLASSNAME, TARGET)
values (FashionFlow_storeent_ID,
 'com.ibm.commerce.order.commands.ExtOrderProcessCmd',
 'This is a new task command for tutorial two.',
 'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImpl',
 'local');
```

Enter を押して SQL ステートメントを実行します。

6. データベースの変更をコミットするには、以下のように入力します。
- ```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

変更のコンパイル



重要: Stores Web プロジェクトを再作成していないことを確認してください。

このセクションでは、以下のようにして、コードに加えた変更をコンパイルします。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、以下のプロジェクトを選択します。
 - WebSphereCommerceServerExtensionsData
 - WebSphereCommerceServerExtensionsLogic
2. 上記のプロジェクトを強調表示して、「プロジェクトの構築 (Build project)」を右クリックして選択します。

ギフト・メッセージの表示ページの変更

このステップでは、OrderSubmitForm および OrderDetailDisplay テンプレートに変更を加えて、顧客がギフト・オーダーに関するメッセージ情報を入力したり、要約ページに情報を表示したりできるようにします。これらのページを変更するときの戦略は、ページの新規情報を指定する、別の JSP テンプレートを組み込むことです。これらの新規ページ (OrderSubmitFormInclude.jsp および OrderDetailDisplayInclude.jsp) は、JSTL を使用して新しい情報を表示します。

表示ページに変更を加えるには、以下のようにします。

1. Web パースペクティブに切り替えて「J2EE ナビゲーター (J2EE Navigator)」ビューを使用します。
2. 229 ページの『第 10 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、以下のようにして、Stores Web プロジェクトのプロパティを変更する必要があります。
 - a. Stores Web プロジェクトを右クリックして、「プロパティ」を選択します。
 - b. 左側の「Web」を選択してから、「使用可能な Web プロジェクト・フィーチャー (Available Web Project Features)」から「JSP 標準タグ・ライブラリーの組み込み (Include the JSP Standard Tag Library)」を選択します。「適用」をクリックします。更新が完了したら、「OK」をクリックして、プロパティ・エディターをクローズします。
3. 以下のディレクトリーを拡張表示します。
Stores\Web Content\FashionFlow_name。
4. 以下のようにして、OrderSubmitForm.jsp ファイルのバックアップ・コピーを作成します。

- a. 「ShoppingArea」 > 「CheckoutSection」 > 「StandardCheckoutSubsection」ディレクトリーを拡張表示します。
 - b. **OrderSubmitForm.jsp** ファイルを右クリックし、「名前変更」を選択します。
 - c. 「名前変更」ウィンドウで OrderSubmitForm_bak.jsp と入力し、「OK」をクリックします。
 - d. プロンプトが出されて、このファイルへのリンクを更新するのであれば、「いいえ」をクリックします
5. 以下のようにして、OrderDetailDisplay.jsp ファイルのバックアップ・コピーを作成します。
- a. 「UserArea」 > 「ServiceSection」 > 「TrackOrderStatusSubsection」 ディレクトリーを拡張表示します。
 - b. **OrderDetailDisplay.jsp** ファイルを右クリックし、「名前変更」を選択します。
 - c. 「名前変更」ウィンドウで OrderDetailDisplay_bak.jsp と入力し、「OK」をクリックします。
 - d. プロンプトが出されて、このファイルへのリンクを更新するのであれば、「いいえ」をクリックします
6. *FashionFlow_name* ディレクトリーを右クリックして、「インポート (Import)」を選択します。
「インポート (Import)」ウィザードがオープンします。
7. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
8. 「ブラウズ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、*yourDirectory*¥WC_SAMPLE_55.zip にあります。 *yourDirectory* はパッケージをダウンロードしたディレクトリーです。
9. 「全選択解除 (Deselect All)」をクリックし、ディレクトリーを拡張表示して、インポートする以下のファイルを選択します。
- ShoppingArea¥CheckoutSection¥StandardCheckoutSubsection¥OrderSubmitForm.jsp
 - ShoppingArea¥CheckoutSection¥StandardCheckoutSubsection¥OrderSubmitFormInclude.jsp
 - UserArea¥ServiceSection¥TrackOrderStatusSubsection¥OrderDetailDisplay.jsp
 - UserArea¥ServiceSection¥TrackOrderStatusSubsection¥OrderDetailDisplayInclude.jsp

10. 「フォルダー (Folder)」フィールドでは、「ストア/Web コンテンツ/*FashionFlow_name* (Stores/Web Content/*FashionFlow_name*)」フォルダーがすでに指定されています。この値をそのまま使用します。
11. 「終了」をクリックします。

OrderSubmitForm.jsp ファイルを調べると、以下の部分が含まれることがわかります。

```
<!-- tutorial start-->
<jsp:include page="OrderSubmitFormInclude.jsp" flush="true" />
<!-- tutorial done -->
```

このようにして、新しい OrderSubmitFormInclude.jsp ファイルは、既存の JSP テンプレートに組み込まれます。OrderSubmitFormInclude.jsp を、ご使用のテンプレートで JSTL を使用方法の一例として調べてください。同様に、OrderDetailDisplay.jsp ファイルおよび OrderDetailDisplayInclude.jsp ファイルも調べてください。

変更した JSP テンプレートで使用されるストリング値を含むプロパティ・ファイルもインポートする必要があります。このファイルは `ordergift.properties` というものです。このファイルをインポートするには、以下のようになります。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、以下のディレクトリーを拡張表示します。
「Stores」 > 「Web Content」 > 「WEB-INF」 > 「classes」 > 「*FashionFlow_name*」 ディレクトリー。
2. *FashionFlow_name* ディレクトリーを右クリックして、「インポート (Import)」を選択します。
「インポート (Import)」ウィザードがオープンします。
3. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
4. 「ブラウズ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、
`yourDirectory\WC_SAMPLE_55.zip`
にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
5. 「全選択解除 (Deselect All)」をクリックし、ディレクトリーを拡張表示して、インポートする以下のファイルを選択します。
 - `ordergift.properties`
6. 「フォルダー (Folder)」フィールドでは、「ストア/Web コンテンツ/*WEB-INF*/*classes*/*FashionFlow_name* (Stores/Web Content/*WEB-INF*/*classes*/*FashionFlow_name*)」フォルダーがすでに指定されています。この値をそのまま使用します。
7. 「終了」をクリックします。

新規ギフト・メッセージ機能のテスト

このセクションでは、以下のようにして、新しいギフト・メッセージ機能をテストします。

1. サーバー・パースペクティブに切り替えます (「ウィンドウ」>「パースペクティブのオープン (Open Perspective)」>「サーバー (Server)」)。
2. Payment Server を開始します。
3. **WebSphereCommerceServer** サーバーを右クリックして、「スタート」 (または「再始動 (Restart)」) を選択します。
4. Web ブラウザーをオープンし、以下の URL を入力します。
`http://localhost/webapp/wcs/stores/servlet/FashionFlow/index.jsp`
5. 新規ユーザーとしてログオンします。たとえば、「登録」をクリックしてから、新規ユーザーを作成するか、既存のユーザーとしてログオンします。
6. 新規登録ユーザーとして、ストア全体をブラウズし、ショッピング・カートにアイテムを追加してから、購入を完了します。以下の画面ショットに示されているように、ギフト・メッセージをオーダーに追加できるようになります。

[SHOPPING CART](#) [MY ACCOUNT](#) [CONTACT](#)

[Home](#) [Men's](#) [Women's](#)

Checkout - Order summary

*= required fields

Estimated ship date: March 27, 2003

| Quantity | Item | Shipping address | Shipping method |
|----------|---|----------------------|-----------------|
| 1 | Pleated shorts
Color: black
Size: 4 | a
a
a a a
a | Regular mail |

Billing address

a
a
a a a
a

Payment Information

Credit card

*Credit card type:

*Card number:

*Expiration month:

*Expiration year:

At FashionFlow we use standard Secure Socket Layer technology to encrypt your information. As a result, your information is protected, and will not be shared with anyone other than the merchant. For more information, see our privacy policy.

Gift Order

Recipient:

Sender:

Message Field 1:

Message Field 2:

< Previous
Order now

図 49.

このオーダーに関連したオーダー番号をメモします。

7. 「アカウント」をクリックします。
8. 「オーダーの表示」をクリックします。
9. ステップ 6 で作成したオーダーを選択します。以下のような画面が表示されます。

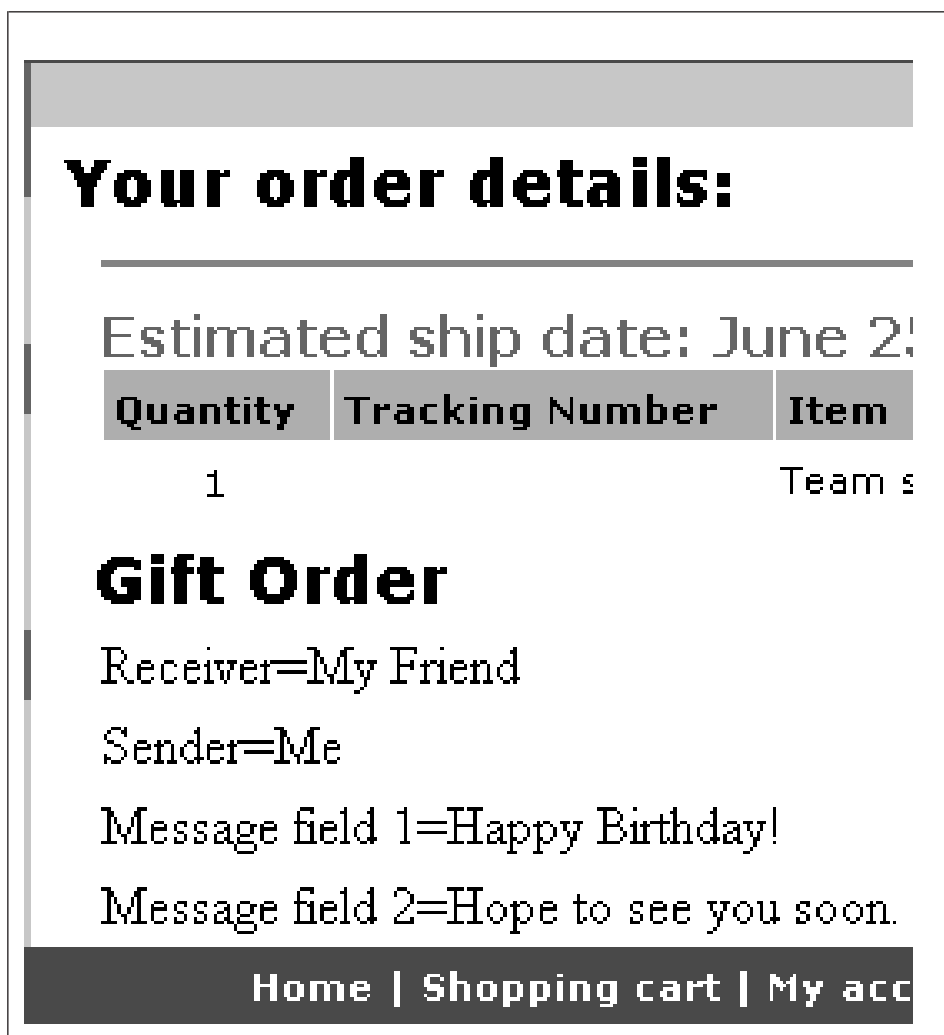


図 50.

新規ギフト・メッセージ機能のデプロイメント

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 (FashionFlow 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。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。

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。

ターゲットの WebSphere Commerce Server にデプロイメントしなければならない資産にはさまざまなタイプがあります。以下のものが含まれます。

- タスク・コマンドおよび Data Bean のロジック
- Enterprise Bean のロジック
- 更新した JSP テンプレート
- スキーマの更新 (新規テーブル) やコマンド・レジストリーの更新を含む、データベースの更新

このセクションでは、これらのすべての資産をターゲットの WebSphere Commerce Server に増分 デプロイメントする方法について説明します。この増分デプロイメントは、EAR ファイル全体のデプロイメントとは対照的な方法です。

コマンドと Data Bean の JAR ファイルの作成

このセクションでは、コントローラー・コマンド、タスク・コマンド、および Data Bean のロジックを含む JAR ファイルを作成する方法について説明します。

この JAR ファイルを作成するには、開発マシンで以下のステップを実行します。

1. `drive:\ExportTemp3` というローカル・ファイル・システム上にディレクトリを作成します。
2. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
3. **WebSphereCommerceServerExtensionsLogic** 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。
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようにします。
 - a. 「JAR ファイル」を選択し、「次へ」をクリックします。
 - b. 「エクスポートするリソースの選択 (Select the resources to export)」の下の左側のペインに、プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。

- c. 右側のペインで、以下のリソースだけが選択されていることを確認します。
 - .classpath
 - .project
 - .serverPreference
- d. 「生成されたクラス・ファイルおよびリソースのエクスポート (**Export generated class files and resources**)」が選択されていることを確認します。
- e. 「Java ソース・ファイルおよびリソースのエクスポート (**Export Java source files and resources**)」を選択しないでください。
- f. 「エクスポート先の選択 (**Select the export destination**)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。ここでは、`drive:¥ExportTemp3¥WebSphereCommerceServerExtensionsLogic.jar` と入力します。JAR ファイル名は `WebSphereCommerceServerExtensionsLogic.jar` でなければならないことに注意してください。
- g. 「終了」をクリックします。

EJB JAR ファイルの作成

EJB JAR ファイルを作成するには、以下のようにします。

1. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. **WebSphereCommerceServerExtensionsData** プロジェクトを拡張表示します。
3. 「**EJB Deployment Descriptor**」をダブルクリックします。
4. 「概要 (Overview)」タブを選択しながら、ペインの下部にスクロールし、「**WebSphere バインディング (WebSphere Bindings)**」セクションを見つけます。
5. 「データ・ソース JNDI 名 (**DataSource JNDI name**)」フィールドに、ターゲットの WebSphere Commerce Server のデータ・ソース JNDI 名を入力します。値の例は以下のとおりです。

 jdbc/WebSphere Commerce DB2 DataSource demo

ここでターゲット WebSphere Commerce Server は、DB2 データベースを使用し、WebSphere Commerce インスタンス名は“demo”です。

 jdbc/WebSphere Commerce Oracle DataSource demo

ここでターゲット WebSphere Commerce Server は、Oracle データベースを使用し、WebSphere Commerce インスタンス名は“demo”です。



DataSource JNDI 名の値は、“jdbc/”を、ターゲット WebSphere Commerce Server のデータ・ソース名に追加することで作成されます。ターゲット WebSphere Commerce Server で `instanceName.xml` ファイルをオープンし、ファイル内で `DatasourceName=` を探すことで、データ・ソース名を確認できます。

6. デプロイメント記述子の変更内容を保管します (Ctrl + S)。
7. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、**WebSphereCommerceServerExtensionsData** 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。
「エクスポート」ウィザードがオープンします。
8. 「エクスポート」ウィザードで、以下のようにします。
 - a. 「EJB JAR ファイル」を選択し、「次へ」をクリックします。
 - b. 「エクスポートしたいリソース (What resources do you want to export?)」の値として、EJB プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。
 - c. 「リソースのエクスポート先にしたい場所 (Where do you want to export resources to?)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。ここでは、
`drive:¥ExportTemp3¥WebSphereCommerceServerExtensionsData.jar` と入力します。
 - d. 「終了」をクリックします。
9. JAR ファイルを作成し終わったら、ステップ 5 でローカル・デプロイメント記述子に加えた変更を取り消して、ローカル・テスト・サーバーに必要な設定を復元します。

ストア資産のエクスポート

OrderSubmitForm および OrderDetailDisplay 表示テンプレートを WebSphere Studio Application Developer からエクスポートするには、以下のようにします。

1. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 「ストア」フォルダーを拡張表示します。
3. 「Web コンテンツ (Web Content)」フォルダーを右クリックして、「エクスポート」を選択します。
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようにします。
 - a. 「ファイル・システム (File system)」を選択して、「次へ」をクリックします。
 - b. 以下のリソースをデプロイメント対象として選択します。
 - Web Content¥FashionFlow_name¥ShoppingArea¥CheckoutSection¥StandardCheckoutSubsection¥OrderSubmitForm.jsp
 - Web Content¥FashionFlow_name¥ShoppingArea¥CheckoutSection¥StandardCheckoutSubsection¥OrderSubmitFormInclude.jsp

- Web Content¥FashionFlow_name¥UserArea¥ServiceSection¥TrackOrderStatusSubsection¥OrderDetailDisplay.jsp
 - Web Content¥FashionFlow_name¥UserArea¥ServiceSection¥TrackOrderStatusSubsection¥OrderDetailDisplayInclude.jsp
 - Web Content¥WEB-INF¥lib¥jstl.jar
 - Web Content¥WEB-INF¥lib¥standard.jar
 - Web Content¥WEB-INF¥classes¥FashionFlow_name¥ordergift.properties
- c. 「ファイルのディレクトリー構造の作成 (**Create directory structure for files**)」を選択します。
- d. 「ディレクトリー」フィールドに、これらのリソースを入れる一時ディレクトリーを入力します。たとえば C:¥ExportTemp3 と入力します。
- e. 「終了」をクリックします。

ターゲット WebSphere Commerce Server への資産の転送

このステップでは、ターゲットの WebSphere Commerce Server 上に一時ディレクトリーを作成してから、このディレクトリー中にギフト・オーダー資産をコピーします。以後のステップでは、WebSphere Commerce アプリケーション中の該当する場所にさまざまなタイプのコードを入れます。

開発マシンからターゲットの WebSphere Commerce Server にファイルをコピーするには、以下のようにします。

1. ターゲットの WebSphere Commerce Server で、`drive:¥ImportTemp3` という一時ディレクトリーを作成します。
2. コンピューター間でファイルをコピーする方法を決めます。ターゲットの WebSphere Commerce Server 上のドライブを開発マシンにマッピングするか、または FTP アプリケーションが構成済みの場合はこのアプリケーションを使用して、コピーを実行できます。
3. 開発マシンから、`drive:¥ExportTemp3` の内容をターゲットの WebSphere Commerce Server の `drive:¥ImportTemp3` にコピーします。


ターゲット WebSphere Commerce Server の停止

デプロイメント・ステップを開始する前に、コマンド行で `stopServer` コマンドを発行し、ターゲット WebSphere Commerce Server を停止する必要があります。このコマンドの詳細は、「*WebSphere Commerce Studio* インストール・ガイド」を参照してください。

ターゲット WebSphere Commerce Server 上のデータベースの更新

タスク・コマンド・インプリメンテーションの登録

コマンド・レジストリーを変更するには、以下のようにします。


1.  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 MyExtOrderProcessCmdImpl を登録します。
 - 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」 > 「コマンド行ツール (Command Line Tools)」 > 「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. ツール設定をクローズします。
 - e. 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 targetDB user dbuser using dbpassword;  
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,  
CLASSNAME, TARGET)  
values (FashionFlow_storeent_ID,  
       'com.ibm.commerce.order.commands.ExtOrderProcessCmd',  
       'This is a new task command for tutorial two.',  
       'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImpl',  
       'local');
```

ここで

- *targetDB* は、ターゲット・データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード
- *FashionFlow_storeent_ID* は、サンプル・ストアのストア ID

「実行」アイコンをクリックします。コマンド・センターはオープンしたままにしておきます。

2.  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 MyOrderItemAddCmdImpl を登録します。
 - 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、以下の SQL ステートメントを入力します。

```
insert into CMDREG (STOREENT_ID, INTERFACENAME, DESCRIPTION,
  CLASSNAME, TARGET)
values (FashionFlow_storeent_ID,
  'com.ibm.commerce.order.commands.ExtOrderProcessCmd',
  'This is a new task command for tutorial two.',
  'com.ibm.commerce.sample.commands.MyExtOrderProcessCmdImp1',
  'local');
```

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、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

XORDGIFT テーブルの作成

このステップでは、XORDGIFT テーブルを作成します。

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. 「スクリプト」ウィンドウで、以下を入力します。

```
create table XORDGIFT (ORDERSID bigint not null, RECEIPTNAME varchar(50),
  SENDERNAME varchar(50), MSGFIELD1 varchar(50), MSGFIELD2 varchar(50),
  constraint p_xordgift primary key (ORDERSID),
  constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)
  on delete cascade);
insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
```

ここで

- *targetDB* は、ターゲット・データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード

「実行」アイコンをクリックします。

これで、XORDGIFT テーブルが作成されました。

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 XORDGIFT (ORDERSID NUMBER NOT NULL, RECEIPTNAME VARCHAR(50),  
SENDERNAME VARCHAR(50), MSGFIELD1 VARCHAR(50), MSGFIELD2 VARCHAR(50),  
constraint p_xordgift primary key (ORDERSID),  
constraint f_xordgift foreign key (ORDERSID) references ORDERS(ORDERS_ID)  
on delete cascade);  
insert into XORDGIFT (ORDERSID) (select ORDERS_ID from ORDERS);
```

それから Enter を押して SQL ステートメントを実行します。XORDGIFT テーブルが作成されました。

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

ターゲット WebSphere Commerce Server 上のストア資産の更新

このステップでは、以下のようにして、変更を加えたストア資産でストアを更新します。

1. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war` ディレクトリ内のバックアップを取ります。
2. `drive:¥ImportTemp3¥Stores¥Web Content` ディレクトリに移動します。
3. 「`FashionFlow_name`」フォルダーを、以下のディレクトリにコピーします。
`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war`
ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

ターゲット WebSphere Commerce Server 上のコマンドと Data Bean の JAR ファイルの更新

このステップでは、以下のようにして、新しいコマンドと Data Bean の JAR ファイルを使用するようにターゲット WebSphere Commerce Server を更新します。

1. 以下のようにして、既存の JAR ファイルのバックアップ・コピーを作成する必要があります。
 - a. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリに移動します。
 - b. `WebSphereCommerceServerExtensionsLogic.jar` ファイルのコピーを作成して、バックアップ場所に保管します。
2. 新しい `WebSphereCommerceServerExtensionsLogic.jar` ファイルを、
`drive:¥ImportTemp3` ディレクトリから、
`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリにコピーします。

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新

このステップでは、以下のようにして、新しい EJB JAR ファイルを使用するようにターゲット WebSphere Commerce Server を更新します。

1. 以下のようにして、既存の JAR ファイルのバックアップ・コピーを作成する必要があります。
 - a. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーに移動します。
 - b. `WebSphereCommerceServerExtensionsData.jar` ファイルのコピーを作成して、バックアップ場所に保管します。

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

2. 新しい `WebSphereCommerceServerExtensionsData.jar` ファイルを、`drive:¥ImportTemp3` ディレクトリーから、`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーにコピーします。
3. 次に、以下のようにして、EJB デプロイメント記述子の情報に変更を加えなければなりません。

- a. この WebSphere Application Server セルのデプロイメント・リポジトリー (META-INF ディレクトリー) を見つけます。これは一般的に、以下のような形式になります。

```
WAS_installdir¥config¥cells¥cellName
¥applications¥WC_instance_name.ear¥deployments¥
WC_instance_name¥EJBModule.jar¥META-INF.
以下の形式は、このチュートリアルに固有の例です。
D:¥WebSphere¥AppServer¥config¥cells¥myCell¥applications¥
WC_demo.ear¥deployments¥WC_demo¥
WebSphereCommerceServerExtensionsData.jar¥META-INF.
```

ここで、

- `mycell` は WebSphere Application Server セルの名前です。
 - `demo` は WebSphere Commerce インスタンスの名前です。
- b. このディレクトリーには、以下のファイルが含まれています。
 - `ejb-jar.xml`
 - `ibm-ejb-access-bean.xmi`
 - `ibm-ejb-jar-bnd.xmi`
 - `ibm-ejb-jar-ext.xmi`
 - `MANIFEST.MF`

これらすべてのファイルのバックアップを取ります。

- c. ツールを使用して、新しい `WebSphereCommerceServerExtensionsData.jar` ファイルをオープンし、その内容を表示します。

- d. meta-inf ディレクトリーの内容を、この
WebSphereCommerceServerExtensionsData.jar ファイルから、ステップ 3a のディ
レクトリー中に抽出します。
4. コマンド行で WebSphere Application Server startServer コマンドを使用して、
WebSphere Commerce インスタンスを再始動します。

ターゲット WebSphere Commerce Server でのギフト・メッセージ機能の検 査

このセクションでは、ターゲット WebSphere Commerce Server 上でギフト・メッセー
ジ・ロジックが正しく機能することを、以下のようにして検査します。

1. ブラウザーをオープンし、FashionFlow サンプル・ストアを基にしたご使用のストア
の URL を入力します。
2. 新規ユーザーとしてログオンします。たとえば、「登録」をクリックして、ユーザー
「shopper」を作成します。
3. 新規登録ユーザーとして、ストア全体をブラウズし、ショッピング・カートにアイテ
ムを追加してから、購入を完了します。以下の画面ショットに示されているように、
ギフト・メッセージをオーダーに追加できるようになります。

[SHOPPING CART](#) [MY ACCOUNT](#) [CONTACT](#)

[Home](#) [Men's](#) [Women's](#)

Checkout - Order summary

* = required fields

Estimated ship date: March 27, 2003

| Quantity | Item | Shipping address | Shipping method |
|----------|---|----------------------|-----------------|
| 1 | Pleated shorts
Color: black
Size: 4 | a
a
a a a
a | Regular mail |

Billing address

a
a
a a a
a

Payment Information

Credit card

* Credit card type:

* Card number:

* Expiration month:

* Expiration year:

Gift Order

Recipient:

Sender:

Message Field 1:

Message Field 2:

< Previous
Order now

At FashionFlow we use standard Secure Socket Layer (SSL) technology to encrypt your information. As a result, your information you enter is protected, and will not be shared with anyone other than the merchant. For more information, see our privacy policy.

図 51.

このオーダーに関連したオーダー番号をメモします。

4. 「アカウント」をクリックします。
5. 「オーダーの表示」をクリックします。
6. ステップ 3 で作成したオーダーを選択します。以下のような画面が表示されます。



図 52.

第 13 章 チュートリアル: 既存の WebSphere Commerce Entity Bean の拡張

このチュートリアルでは、既存の WebSphere Commerce Entity Bean を変更するときに使用するプロセスを学習します。ここでは、ユーザー登録プロセスに、追加のハウジング情報調査書を追加するシナリオを使用します。この場合、User Entity Bean が変更され、ユーザーのハウジング・タイプの値と場所の値を保管する、別のフィールドが組み込まれます。(XHOUSING という) 新しいデータベース・テーブルが作成されると同時に、User Bean の既存のマッピング情報が変更されて、この新しいテーブルが組み込まれます。

新しいテーブルと Entity Bean の変更に加えて、新しい MyPostUserRegistrationAddCmdImpl インプリメンテーション・クラスが作成されます。これには、ハウジング調査書情報を処理し、新しい情報でデータベースを更新するロジックが含まれます。

ハウジング情報を収集し、後で結果を表示できるように、UserRegistrationAddForm.jsp ファイルと UserRegistrationUpdateForm.jsp ファイルが変更されます。


前提条件

この章のチュートリアルは、これまでのチュートリアルを完了している必要はありません。これまでのチュートリアルを完了している場合でも、そのコードはこの章のチュートリアルと競合しないので、コードをワークスペース中に残しておいても害はありません。

この章のチュートリアルを開始する前に、FashionFlow サンプル・ストアを基にしたストアを発行していなければなりません。このストア中で購入を完了できなければなりません (カタログのブラウズ、ショッピング・カートへのアイテムの追加、オーダーの確認のチェックアウトや表示など)。

XHOUSING テーブルの作成と値の取り込み

Entity Bean の作成に備えて、まず新しいデータベース・テーブルを作成して移植する必要があります。作成するテーブルの名前は XHOUSING とします。

 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」>「コマンド行ツール (Command Line Tools)」>「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 developmentDB user dbuser using dbpassword;  
create table XHOUSING (MEMBERID bigint not null,  
    HOUSINGTYPE integer, LOCATION integer,  
    constraint p_xhousing primary key (MEMBERID),  
    constraint f_xhousing foreign key (MEMBERID)  
    references USERS (USERS_ID) on delete cascade);  
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード

「実行」アイコンをクリックします。

SQL ステートメントが正常に完了したことを示すメッセージが表示されます。

▶ **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 XHOUSING (MEMBERID number not null,  
    HOUSINGTYPE integer, LOCATION integer,  
    constraint p_xhousing primary key (MEMBERID),  
    constraint f_xhousing foreign key (MEMBERID)  
    references USERS (USERS_ID) on delete cascade);  
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

それから Enter を押して SQL ステートメントを実行します。 XHOUSING テーブルが作成されました。

6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

新規フィールドの User Entity Bean への追加

このセクションでは、ハウジング情報の取り込みに使用する 2 つの新規フィールドを User Entity Bean へ追加します。新規フィールドの名前は、housingType および location とします。これらのフィールドは最終的に XHOUSING テーブルの HOUSINGTYPE 列と LOCATION 列にマップします。

これらの新規フィールドを追加するには、以下のようにします。

1. まだオープンしていなければ、WebSphere Commerce Studio を始動します (「スタート」>「プログラム」>「IBM WebSphere Commerce Studio」>「WebSphere Commerce 開発環境」)。
2. サーバー・パースペクティブに切り替え、WebSphereCommerceServer テスト・サーバーが停止していることを確認します。
3. J2EE パースペクティブに切り替え、「J2EE 階層 (J2EE Hierarchy)」ビューを選択します。
4. 「EJB モジュール (EJB Modules)」を拡張表示してから、**Member-MemberManagementData** EJB プロジェクトをダブルクリックします。EJB Deployment Descriptor Editor がオープンします。
5. 「Bean」タブをクリックし、表示された Bean のリストから **User Bean** を選択します。
6. 「CMP フィールド (CMP fields)」テキスト・ボックスの隣の「追加」をクリックします。
「CMP 属性の作成 (Create CMP attribute)」ウィンドウがオープンします。
7. 以下のようにプロパティを設定して、CMP フィールドを作成します。
 - a. 「名前」フィールドで、housingType と入力します。
 - b. 「タイプ」フィールドで、java.lang.Integer と入力します。
 - c. 「getter および setter メソッドを使ったアクセス」を使用可能にします。
 - d. 「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」チェック・ボックスをクリアします。(このようにすると、getter を読み取り専用によるオプションが自動的にクリアされます。)
 - e. 「OK」をクリックします。
8. 再度「追加」をクリックし、以下のようにプロパティを設定してもう 1 つ CMP フィールドを作成します。
 - a. 「名前」フィールドで、location と入力します。
 - b. 「タイプ」フィールドで、java.lang.Integer と入力します。
 - c. 「getter および setter メソッドを使ったアクセス」を使用可能にします。
 - d. 「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」チェック・ボックスをクリアします。(このようにすると、getter を読み取り専用によるオプションが自動的にクリアされます。)

- e. 「OK」をクリックします。
- CMP フィールドのリストに新規フィールドが表示されます。
9. 変更内容を保管してから、EJB Deployment Descriptor Editor をクローズします。

スキーマとテーブル・マッピング情報の更新

ここからは、新しい XHOUSING テーブルによって User スキーマを更新し、その新しいテーブル用の外部キー関係を作成し、User Entity Bean のフィールドと XHOUSING テーブルの列との間のテーブル・マップを作成します。この方法でオブジェクト・モデルを拡張すると、新規列が USERS テーブルに直接追加されているかのようなコードに見えます。

XHOUSING テーブルのテーブル定義の作成

XHOUSING テーブル定義を作成し、USERS テーブルと XHOUSING テーブルとの間の外部キー関係を作成するには、以下のようになります。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「データベース (Databases)」を拡張表示します。
2. **Member-MemberManagementData** データベースを拡張表示してから、**NULLID** スキーマを拡張表示します。
3. 「テーブル (Tables)」を右クリックして、「新規」>「新規テーブル定義 (New Table Definition)」を選択します。
「テーブル定義 (Table Definition)」ウィンドウがオープンします。
4. 「テーブル名 (Table name)」フィールドに XHOUSING と入力して、「次へ」をクリックします。
5. 次に、以下のようにして、キー列をテーブル定義に追加しなければなりません。
 - a. 「別のものを追加 (Add Another)」をクリックします。
 - b. 「列名」フィールドに、MEMBERID と入力します。
 - c.  「列タイプ」ドロップダウン・リストで、「BIGINT」を選択します。
 - d.  「列タイプ」ドロップダウン・リストで、「NUMBER」を選択します。
 - e. 「キー列 (Key column)」を選択します。
 - f.  「数値の精度 (Numeric precision)」フィールドで、38 と入力します。
 - g.  「数値のスケール (Numeric Scale)」の値は 0 のままにします。
6. 次に、以下のようにして、追加列をテーブル定義に追加します。
 - a. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。


表 26.

| プロパティ | 値 |
|-----------------|-------------|
| 列名 | HOUSINGTYPE |
| 列タイプ | INTEGER |
| ヌル可能 (Nullable) | 選択 |

- b. 「別のものを追加 (Add Another)」をクリックし、以下のプロパティの列を作成します。

表 27.

| プロパティ | 値 |
|-----------------|----------|
| 列名 | LOCATION |
| 列タイプ | INTEGER |
| ヌル可能 (Nullable) | 選択 |

- c. 「次へ」をクリックします。
7. 「基本キー名 (Primary key name)」フィールドで、 p_xhousing を入力し「次へ」をクリックします。
 8. 「別のものを追加 (Add another)」をクリックして、外部キーをクリックします。以下の値を指定します。
 - a. 「外部キー名 (Foreign key name)」フィールドに、 f_xhousing と入力します。
 - b. 「削除方法 (On Delete)」ドロップダウン・リストで、 **CASCADE** を選択します。
 - c. 「ターゲット・テーブル (Target Table)」ドロップダウン・リストで、 **NULLID.USERS** を選択します。
 - d. 「ソース列 (Source Columns)」ペインで、「MEMBERID」をクリックしてから、「>」をクリックして、外部キーを追加します。
 9. 「終了」をクリックします。
 10.  テキスト・エディターを使用し、テーブル定義を以下のように編集する必要があります。
 - a. 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
 - b. **Member-MemberManagementData** プロジェクトを拡張表示します。
 - c. 以下のように拡張表示します。「ejbModule」>「META-INF」>「スキーマ」。
 - d. **Member-MemberManagementData_NULL_XHOUSING.xml** ファイルを右クリックして、「オープンに使用 (Open With)」>「テキスト・エディター (Text Editor)」を選択します。

- e. SQLNumeric_6 が出現するすべての箇所を、SQLNumeric_3 に置き換えます。
- f. 変更内容を保管し、テキスト・エディターをクローズします。

Member-MemberManagementData/NULLID の下の「テーブル (Tables)」フォルダーを拡張表示すると、新しい Member-MemberManagementData_NULL_XHOUSING テーブル定義が表示されます。

XHOUSING テーブル・マップの作成

このセクションでは、XHOUSING テーブルの 2 つの列 (HOUSINGTYPE および LOCATION) と、User Entity Bean の 2 つのフィールド (housingType および location) との間にマッピングを作成します。

このマッピングを作成するには、以下のようになります。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 以下のフォルダーを拡張表示します。「Member-MemberManagementData」 > 「ejbModule」 > 「META-INF」。
3. **Map.mapxmi** ファイルをダブルクリックします。
4. 「テーブル (Tables)」ペインで、以下のテーブルを強調表示します。

- **MEMBER**
- **USERS**
- **XHOUSING**

(一度に複数のテーブルを選択するには、Ctrl キーを押したままにします。)

5. 「Enterprise Bean」ペイン (エディターの一番上) で、**Member** グループを拡張表示してから、**User Entity Bean** を右クリックし、「マッピングの作成 (Create Mapping)」を選択します。
6. **User Entity Bean** を拡張表示します。
7. 「テーブル (Tables)」ペインで、**XHOUSING** テーブルを拡張表示して、列が見えるようにします。
8. **housingType** Bean の属性を強調表示し、**HOUSINGTYPE** 列の上にドラッグして、マッピングを作成します。
9. **location** Bean の属性を強調表示し、**LOCATION** 列の上にドラッグして、マッピングを作成します。
10. 変更内容を保管し、Map.mapxmi ファイルをクローズします。

マッピング・ファイルの更新

1. 以下のフォルダーを拡張表示します。「Member-MemberManagementData」 > 「ejbModule」 > 「META-INF」。
2. **Map.mapxmi** ファイルを右クリックして、「オープンに使用 (Open With)」 > 「テキスト・エディター (Text Editor)」を選択します。

3. 以下のストリングを見つけます。

User_EJB

この行の 2 行下に、以下の行があります。

```
<discriminatorValues>User</discriminatorValues>
```

前述の行があったら、以下のように変更する必要があります。

```
<discriminatorValues>'U'</discriminatorValues>
```

4. 変更内容を保管します。

Access Bean およびデプロイメント・コードの生成

User Entity Bean を変更したので、その Access Bean とデプロイメント・コードを再生成する必要があります。

Access Bean を再生成するには、以下のようにします。

1. 「J2EE 階層 (J2EE Hierarchy)」ビューで、「**EJB モジュール (EJB Modules)**」を拡張表示します。
2. **Member-MemberManagementData** EJB モジュールを右クリックし、「**Access Bean**」>「**Access Bean の編集 (Edit Access Bean)**」を選択します。
3. **CopyHelper** を選択して、「次へ」をクリックします。
4. 「EJB プロジェクトの選択 (Select EJB Project)」ウィンドウで、「次へ」をクリックします。
5. 「ヘルパー Access Bean のコピー (Copy Helper Access Bean)」ウィンドウで、以下のようにします。
 - a. 「Enterprise Bean」ドロップダウン・リストで、**User** を選択します。
 - b. コンストラクター・メソッドのドロップダウン・リストから、**findByPrimaryKey(com.ibm.commerce.user.objects.MemberKey)** を選択します。
 - c. 「属性ヘルパー (Attribute Helpers)」リストで、他のすべての属性と共に、**housingType** 属性と **location** 属性が選択されていることを確認します。
6. 「終了」をクリックします。
7. **Member-MemberManagementData** EJB モジュールを右クリックし、「**Access Bean**」>「**Access Bean の再生成 (Regenerate Access Beans)**」を選択します。
8. 「全選択」をクリックして、「終了」をクリックします。

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。

1. **Member-MemberManagementData** EJB モジュールを右クリックし、「生成 (Generate)」 > 「デプロイメントおよび RMIC コード (Deploy and RMIC Code)」を選択します。
2. 「全選択」をクリックして、「終了」をクリックします。

MyPostUserRegistrationAddCmdImpl インプリメンテーションの作成

このステップでは、UserRegistrationAddCmd コントローラー・コマンドを拡張して、登録フォームに収集される新規ハウジング情報の解析に使用する新規ロジックを組み込みます。この拡張を行うには、PostUserRegistrationCmd インターフェースをインプリメントする MyPostUserRegistrationAddCmdImpl 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・クラスを新規作成するには、コマンド・レジストリを更新して、この変更を反映させなければなりません。他のチュートリアルと同様に、この新規コマンドのコードも備えられています。

新しい MyPostUserRegistrationAddCmdImpl 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. 230 ページの『サンプル・コードの場所』のステップを完了していることを確認します。
2. WebSphere Studio Application Developer で、Java パースペクティブをオープンします (「ウィンドウ」 > 「パースペクティブのオープン (Open Perspective)」 > 「Java」)。
3. **WebSphereCommerceServerExtensionsLogic** プロジェクトを拡張表示します。
4. 「src」フォルダーを右クリックして、「インポート (Import)」を選択します。「インポート (Import)」ウィザードがオープンします。
5. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
6. 「ブラウズ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、`yourDirectory\WC_SAMPLE_55.zip` にあります。 `yourDirectory` はパッケージをダウンロードしたディレクトリーです。
7. 「全選択解除 (Deselect All)」をクリックし、ディレクトリーを拡張表示して、以下のファイルをインポートするよう選択します。
 - `com\ibm\commerce\sample\commands\MyPostUserRegistrationAddCmdImpl.java`
8. 「フォルダー (Folder)」フィールドでは、「WebSphereCommerceServerExtensionsLogic/src」フォルダーがすでに指定されています。この値をそのまま使用します。
9. 「終了」をクリックします。

10. **com.ibm.commerce.sample.commands** パッケージを拡張表示します。
11. 新しい **MyPostUserRegistrationAddCmdImpl** クラスをダブルクリックします。
12. 「Section 1」をコメント解除します。このコードは、変数を設定し、これらの変数向けに getter および setter を作成します。

```

/// Section 1 //////////////////////////////////
Integer housingType = null;
Integer location = null;

public Integer getHousingType() {
    return housingType;
}

public Integer getLocation() {
    return location;
}

public void setHousingType(Integer newHousingType) {
    housingType = newHousingType;
}

public void setLocation(Integer newLocation) {
    location = newLocation;
}
/// End of Section 1 //////////////////////////////////

```

13. 「Section 2」のコメントを解除します。これによって、以下のコードがクラスに導入されます。

```

/// Section 2 //////////////////////////////////
public void performExecute() throws ECEException {
    super.performExecute();

    // Set the needed fields before processing the survey
    setHousingType(requestProperties.getInteger("housingType", 0));
    setLocation(requestProperties.getInteger("location", 0));

    processSurvey();
}
/// End of Section 2 //////////////////////////////////

```

前述のコードは、 `PostUserRegistrationAddCmdImpl` の通常のロジックを実行できるように、スーパークラスの `performExecute` メソッドを呼び出します。これが完了したら、新しい `processSurvey` メソッドが呼び出されます。

14. 「Section 3」のコメントを解除して、以下のコードをクラスに導入します。

```

/// Section 3 //////////////////////////////////

private void processSurvey() throws ECEException {

    try {
        // load up the user data

```

```

UserAccessBean abUser = new UserAccessBean();
abUser.setInitKey_MemberId(commandContext.getUserId().toString());
abUser.refreshCopyHelper();

// store the new attributes
abUser.setHousingType(getHousingType());
abUser.setLocation(getLocation());

abUser.commitCopyHelper();
} catch (javax.ejb.FinderException e) {
    throw new ECSystemException(
        ECMessage._ERR_FINDER_EXCEPTION,
        this.getClass().getName(),
        "processSurvey");
} catch (javax.naming.NamingException e) {
    throw new ECSystemException(
        ECMessage._ERR_NAMING_EXCEPTION,
        this.getClass().getName(),
        "processSurvey");
} catch (java.rmi.RemoteException e) {
    throw new ECSystemException(
        ECMessage._ERR_REMOTE_EXCEPTION,
        this.getClass().getName(),
        "processSurvey");
} catch (javax.ejb.CreateException e) {
    throw new ECSystemException(
        ECMessage._ERR_CREATE_EXCEPTION,
        this.getClass().getName(),
        "processSurvey");
}
}
}
/// End of Section 3 //////////////////////////////////


```

15. 変更内容を保管します。
16. **WebSphereCommerceServerExtensionsLogic** プロジェクトを右クリックし、「プロジェクトの構築 (Build project)」を選択します。

コマンド・レジストリーに変更を加える

コマンド・レジストリーに変更を加えて、ショッピング・フロー中で新規インプリメンテーション・クラスを使用できるようにしなければなりません。

コマンド・レジストリーを変更するには、以下のようになります。


1.  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 MyPostUserRegistrationAddCmdImpl を登録します。
 - 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を作成します。

```
connect to developmentDB user dbuser using dbpassword;  
insert into CMDREG values (FashionFlow_storeent_Id,  
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd'  
'Command for modified user bean tutorial',  
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',  
null, null, 'Local');
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
 - *dbuser* は、データベースのユーザー
 - *dbpassword* は、データベースのユーザーのパスワード
 - *FashionFlow_storeent_Id* は、ストアの固有のストア・エンティティ ID
- 「実行」アイコンをクリックします。

2.  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 MyPostUserRegistrationAddCmdImpl を登録します。

- 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、以下の SQL ステートメントを入力します。

```
insert into CMDREG values (FashionFlow_storeent_Id,  
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd'  
'Command for modified user bean tutorial',  
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',  
null, null, 'Local');
```

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、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

ハウジング情報を収集して表示するための JSP テンプレートの変更

このステップでは、UserRegistrationAddForm および UserRegistrationUpdateForm テンプレートに変更を加えて、顧客がログイン時にハウジング情報を入力したり、要約ページにハウジング情報を表示したりできるようにします。これらのページを変更するときの戦略は、ページの新規情報を指定する、別の JSP テンプレートを組み込むことです。これらの新規ページ (UserRegistrationAddFormInclude.jsp および UserRegistrationUpdateFormInclude.jsp) は、JSTL を使用して新しい情報を表示します。

これらのページを変更するには、以下のようにします。

1. Web パースペクティブに切り替えます。
2. 229 ページの『第 10 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、以下のようにして、Stores Web プロジェクトのプロパティを変更する必要があります。
 - a. **Stores Web** プロジェクトを右クリックして、「プロパティ」を選択します。
 - b. 左側の「Web」を選択してから、「使用可能な Web プロジェクト・フィーチャー (Available Web Project Features)」から「**JSP 標準タグ・ライブラリーの組み込み (Include the JSP Standard Tag Library)**」を選択します。「適用」をクリックします。更新が完了したら、「OK」をクリックして、プロパティ・エディターをクローズします。
3. 以下のディレクトリーを拡張表示します。
Stores\Web Content\FashionFlow_name。
4. 以下のようにして、UserRegistrationAddForm.jsp ファイルのバックアップ・コピーを作成します。
 - a. 「**UserArea**」 > 「**AccountSection**」 > 「**RegistrationSubsection**」 ディレクトリーを拡張表示します。
 - b. **UserRegistrationAddForm.jsp** ファイルを右クリックし、「名前変更」を選択します。
 - c. 「名前変更」ウィンドウで UserRegistrationAddForm_bak.jsp と入力し、「OK」をクリックします。
 - d. プロンプトが出されて、このファイルへのリンクを更新するのであれば、「いいえ」をクリックします
5. 以下のようにして、UserRegistrationUpdateForm.jsp ファイルのバックアップ・コピーを作成します。
 - a. 「**UserArea**」 > 「**AccountSection**」 > 「**RegistrationSubsection**」 ディレクトリーを拡張表示します。
 - b. **UserRegistrationUpdateForm.jsp** ファイルを右クリックし、「名前変更」を選択します。
 - c. 「名前変更」ウィンドウで UserRegistrationUpdateForm_bak.jsp と入力し、「OK」をクリックします。

- d. プロンプトが出されて、このファイルへのリンクを更新するのであれば、「いいえ」をクリックします
6. *FashionFlow_name* ディレクトリーを右クリックして、「**インポート (Import)**」を選択します。
「インポート (Import)」ウィザードがオープンします。
 7. 「**インポート・ソースの選択 (Select an import source)**」リストで、「**ZIP ファイル (Zip file)**」を選択し、「次へ」をクリックします。
 8. 「**ブラウズ**」（「**ZIP ファイル (Zip file)**」フィールドの隣）をクリックして、サンプル・コードに移動します。ファイルは、
yourDirectory¥WC_SAMPLE_55.zip
にあります。 *yourDirectory* はパッケージをダウンロードしたディレクトリーです。
 9. 「**全選択解除 (Deselect All)**」をクリックし、ディレクトリーを拡張表示して、インポートする以下のファイルを選択します。
 - *UserArea*¥*AccountSection*¥*RegistrationSubsection*¥
UserRegistrationAddForm.jsp
 - *UserArea*¥*AccountSection*¥*RegistrationSubsection*¥
UserRegistrationAddFormInclude.jsp
 - *UserArea*¥*AccountSection*¥*RegistrationSubsection*¥
UserRegistrationUpdateForm.jsp
 - *UserArea*¥*AccountSection*¥*RegistrationSubsection*¥
UserRegistrationUpdateFormInclude.jsp
 10. 「**フォルダー (Folder)**」フィールドでは、「ストア/Web コンテンツ/
FashionFlow_name (*Stores/Web Content/FashionFlow_name*)」フォルダーがすでに指定されています。この値をそのまま使用します。
 11. 「**終了**」をクリックします。

このチュートリアルでは、*UserRegistrationAddForm.jsp* ファイルを調べると、以下のセクションが追加されていることがわかります。

```
<%-- Add for tutorial --%>
<tr>
  <td colspan="3">
    <jsp:include page="UserRegistrationAddFormInclude.jsp" flush="true" />
  </td>
</tr>
<%-- End of tutorial --%>
```

さらに、*UserRegistrationAddFormInclude.jsp* ファイルを調べ、新しい情報が JSTL を使用して収集される仕組みを確認できます。同様に、*UserRegistrationAddForm.jsp* ファイルおよび *UserRegistrationAddFormInclude.jsp* ファイルも調べてください。

変更した JSP テンプレートで使用される文字列値を含むプロパティ・ファイルもインポートする必要があります。このファイルは `Housing.properties` というものです。このファイルをインポートするには、以下のようにします。

1. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、以下のディレクトリーを拡張表示します。
「Stores」 > 「Web Content」 > 「WEB-INF」 > 「classes」 > 「*FashionFlow_name*」 ディレクトリー。
2. *FashionFlow_name* ディレクトリーを右クリックして、「インポート (Import)」を選択します。
「インポート (Import)」ウィザードがオープンします。
3. 「インポート・ソースの選択 (Select an import source)」リストで、「ZIP ファイル (Zip file)」を選択し、「次へ」をクリックします。
4. 「ブラウズ」 (「ZIP ファイル (Zip file)」フィールドの隣) をクリックして、サンプル・コードに移動します。ファイルは、
`yourDirectory\WC_SAMPLE_55.zip`
にあります。 *yourDirectory* はパッケージをダウンロードしたディレクトリーです。
5. 「全選択解除 (Deselect All)」をクリックし、ディレクトリーを拡張表示して、インポートする以下のファイルを選択します。
 - `Housing.properties`
6. 「フォルダー (Folder)」フィールドでは、「ストア/Web コンテンツ/WEB-INF/クラス/*FashionFlow_name* (Stores/Web Content/WEB-INF/classes/*FashionFlow_name*)」フォルダーがすでに指定されています。この値をそのまま使用します。
7. 「終了」をクリックします。

変更を加えたコードのテスト

次に、以下のようにして、変更された登録情報をテストする必要があります。

1. サーバー・パースペクティブに切り替えます。
2. **WebSphereCommerceServer** テスト・サーバーを右クリックして、「スタート」を選択します。
3. `Stores\Web Content\iFashionFlow_name` ディレクトリーの下に **index.jsp** を右クリックして、「サーバー上で実行 (Run on Server)」を選択します。
4. ストアのホーム・ページがオープンしたら、「登録」をクリックします。
5. もう一度「登録」をクリックして、新規ユーザーを作成します。
6. 変更された登録ページが表示され、以下の画面ショットに示されているように、ハウジング調査書が示されます。

Send me e-mails about specials and featured clothing.

Would you like to receive e-mails about:

Our men's fashions

Our women's fashions

Our specials

Housing Survey Information

Housing Type

Location

図 53.

- 必要に応じて新規ユーザーの情報を入力し、「送信」をクリックします。
- 情報を送信したら、「個人情報の変更」をクリックして、ハウジング情報が取り込まれたことを確認します。以下のように、調査書返信の要約が示された画面が表示されます。

Housing Survey Information

Housing Type: Townhouse

Location: Downtown

図 54.

ハウジング調査書ロジックのデプロイ

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 (FashionFlow 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。

ターゲットの WebSphere Commerce Server にデプロイメントしなければならない資産にはさまざまなタイプがあります。以下のものが含まれます。

- コマンド・ロジック
- 変更された Enterprise Bean ロジック
- JSP テンプレート
- プロパティ・ファイル
- スキーマの更新 (新規テーブル) やコマンド・レジストリーの更新を含む、データベースの更新

このセクションでは、これらのすべての資産をターゲットの WebSphere Commerce Server に増分 デプロイメントする方法について説明します。

コマンド JAR ファイルの作成

このセクションでは、新しい MyPostUserRegistrationAddCmdImpl ロジックを含む JAR ファイルを作成する方法について説明します。

この JAR ファイルを作成するには、開発マシンで以下のステップを実行します。

1. `drive:\ExportTemp4` というローカル・ファイル・システム上にディレクトリーを作成します。
2. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
3. **WebSphereCommerceServerExtensionsLogic** 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。
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようになります。
 - a. 「JAR ファイル」を選択し、「次へ」をクリックします。
 - b. 「エクスポートするリソースの選択 (Select the resources to export)」の下の左側のペインに、プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。

- c. 右側のペインで、以下のリソースだけが選択されていることを確認します。
 - .classpath
 - .project
 - .serverPreference
- d. 「生成されたクラス・ファイルおよびリソースのエクスポート (**Export generated class files and resources**)」が選択されていることを確認します。
- e. 「Java ソース・ファイルおよびリソースのエクスポート (**Export Java source files and resources**)」を選択しないでください。
- f. 「エクスポート先の選択 (**Select the export destination**)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。ここでは、`drive:\ExportTemp4\WebSphereCommerceServerExtensionsLogic.jar` と入力します。JAR ファイル名は `WebSphereCommerceServerExtensionsLogic.jar` でなければならないことに注意してください。
- g. 「終了」をクリックします。

EJB JAR ファイルの作成

EJB JAR ファイルを作成するには、以下のようにします。

1. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. **Member-MemberManagementData** プロジェクトを拡張表示します。
3. 「**EJB Deployment Descriptor**」をダブルクリックします。
4. 「概要 (Overview)」タブを選択しながら、ペインの下部にスクロールし、「**WebSphere バインディング (WebSphere Bindings)**」セクションを見つけてみます。
5. 「データ・ソース JNDI 名 (**DataSource JNDI name**)」フィールドに、ターゲットの WebSphere Commerce Server のデータ・ソース JNDI 名を入力します。値の例は以下のとおりです。
 - ▶ **DB2** jdbc/WebSphere Commerce DB2 DataSource demo
ここでターゲット WebSphere Commerce Server は、DB2 データベースを使用し、WebSphere Commerce インスタンス名は“demo”です。
 - ▶ **Oracle** jdbc/WebSphere Commerce Oracle DataSource demo
ここでターゲット WebSphere Commerce Server は、Oracle データベースを使用し、WebSphere Commerce インスタンス名は“demo”です。
6. デプロイメント記述子の変更内容を保管します (Ctrl + S)。
7. 「J2EE ナビゲーター (J2EE Navigator)」ビューで、**Member-MemberManagementData** 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。
「エクスポート」ウィザードがオープンします。

8. 「エクスポート」ウィザードで、以下のようになります。
 - a. 「EJB JAR ファイル」を選択し、「次へ」をクリックします。
 - b. 「エクスポートしたいリソース (What resources do you want to export?)」の値として、EJB プロジェクトの名前がすでに取り込まれています。このフィールドは現状のままにします。
 - c. 「リソースのエクスポート先にしたい場所 (Where do you want to export resources to?)」フィールドに、使用する完全修飾 JAR ファイル名を入力します。ここでは、`drive:¥ExportTemp4¥Member-MemberManagementData.jar` と入力します。
 - d. 「終了」をクリックします。
9. JAR ファイルを作成し終わったら、ステップ 5 でローカル・デプロイメント記述子に加えた変更を取り消して、ローカル・テスト・サーバーに必要な設定を復元します。

ストア資産のエクスポート

変更した JSP テンプレートと新規プロパティ・ファイルのエクスポートするには、以下のようになります。

1. WebSphere Studio Application Developer をオープンし、「J2EE ナビゲーター (J2EE Navigator)」ビューに切り替えます。
2. 「ストア」フォルダーを拡張表示します。
3. 「Web コンテンツ (Web Content)」フォルダーを右クリックして、「エクスポート」を選択します。
「エクスポート」ウィザードがオープンします。
4. 「エクスポート」ウィザードで、以下のようになります。
 - a. 「ファイル・システム (File system)」を選択して、「次へ」をクリックします。
 - b. 以下のリソースをデプロイメント対象として選択します。
 - `Web Content¥FashionFlow_name¥UserArea¥AccountSection¥RegistrationSubsection¥UserRegistrationAddForm.jsp`
 - `Web Content¥FashionFlow_name¥UserArea¥AccountSection¥RegistrationSubsection¥UserRegistrationAddFormInclude.jsp`
 - `Web Content¥FashionFlow_name¥UserArea¥AccountSection¥RegistrationSubsection¥UserRegistrationUpdateForm.jsp`
 - `Web Content¥FashionFlow_name¥UserArea¥AccountSection¥RegistrationSubsection¥UserRegistrationUpdateFormInclude.jsp`
 - `Web Content¥WEB-INF¥lib¥jstl.jar`
 - `Web Content¥WEB-INF¥lib¥standard.jar`
 - `Web Content¥WEB-INF¥classes¥FashionFlow_name¥Housing.properties`

- c. 「選択したファイルのディレクトリー構造の作成 (Create directory structure for selected files)」を選択します。
- d. 「ディレクトリー」フィールドに、これらのリソースを入れる一時ディレクトリーを入力します。たとえば C:¥ExportTemp4 と入力します。
- e. 「終了」をクリックします。

ターゲット WebSphere Commerce Server への資産の転送

このステップでは、ターゲットの WebSphere Commerce Server 上に一時ディレクトリーを作成してから、このディレクトリー中にハウジング調査書資産をコピーします。以後のステップでは、 WebSphere Commerce アプリケーション中の該当する場所にさまざまなタイプのコードを入れます。

開発マシンからターゲットの WebSphere Commerce Server にファイルをコピーするには、以下のようにします。


1. ターゲットの WebSphere Commerce Server で、 `drive:¥ImportTemp4` という一時ディレクトリーを作成します。
2. コンピューター間でファイルをコピーする方法を決めます。ターゲットの WebSphere Commerce Server 上のドライブを開発マシンにマッピングするか、または FTP アプリケーションが構成済みの場合はこのアプリケーションを使用して、コピーを実行できます。
3. 開発マシンから、 `drive:¥ExportTemp4` の内容をターゲットの WebSphere Commerce Server の `drive:¥ImportTemp4` にコピーします。

ターゲット WebSphere Commerce Server の停止

デプロイメント・ステップを開始する前に、コマンド行で `stopServer` コマンドを発行し、ターゲット WebSphere Commerce Server を停止する必要があります。このコマンドの詳細は、「*WebSphere Commerce Studio* インストール・ガイド」を参照してください。

ターゲット WebSphere Commerce Server 上のデータベースの更新

XHOUSING テーブルの作成

 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」>「コマンド行ツール (Command Line Tools)」>「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 developmentDB user dbuser using dbpassword;
create table XHOUSING (MEMBERID bigint not null,
    HOUSINGTYPE integer, LOCATION integer,
    constraint p_xhousing primary key (MEMBERID),
    constraint f_xhousing foreign key (MEMBERID)
    references USERS (USERS_ID) on delete cascade);
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード

「実行」アイコンをクリックします。

SQL ステートメントが正常に完了したことを示すメッセージが表示されます。

 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 XHOUSING (MEMBERID number not null,
    HOUSINGTYPE integer, LOCATION integer,
    constraint p_xhousing primary key (MEMBERID),
    constraint f_xhousing foreign key (MEMBERID)
    references USERS (USERS_ID) on delete cascade);
insert into XHOUSING (MEMBERID) (select USERS_ID from USERS);
```

それから Enter を押して SQL ステートメントを実行します。XHOUSING テーブルが作成されました。


6. データベースの変更をコミットするには、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

タスク・コマンドの登録

コマンド・レジストリーを変更するには、以下のようにします。

1.  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 MyPostUserRegistrationAddCmdImpl を登録します。

- 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 developmentDB user dbuser using dbpassword;  
insert into CMDREG values (FashionFlow_storeent_Id,  
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd',  
'Description',  
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',  
null, null, 'Local');
```

ここで

- *developmentDB* は、ユーザーの開発データベースの名前
- *dbuser* は、データベースのユーザー
- *dbpassword* は、データベースのユーザーのパスワード
- *FashionFlow_storeent_Id* は、ストアの固有のストア・エンティティ ID
「実行」アイコンをクリックします。

2. **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 MyPostUserRegistrationAddCmdImpl を登録します。
 - 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、以下の SQL ステートメントを入力します。

```
insert into CMDREG values (FashionFlow_storeent_Id,  
'com.ibm.commerce.usermanagement.commands.PostUserRegistrationAddCmd',  
'Description',  
'com.ibm.commerce.sample.commands.MyPostUserRegistrationAddCmdImpl',  
null, null, 'Local');
```

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、以下のように入力します。

```
commit;
```

それから Enter を押して SQL ステートメントを実行します。

ターゲット WebSphere Commerce Server 上のストア資産の更新

このステップでは、以下のようにして、変更を加えたストア資産でストアを更新します。

1. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war` ディレクトリー (`cellName` はご使用のマシンのホスト名である場合が多い) のバックアップを取ります。
2. `drive:¥ImportTemp4¥Stores¥Web Content` ディレクトリーに移動します。
3. 「*FashionFlow_name*」および「WEB-INF」フォルダーを、以下のディレクトリーにコピーします。

`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear¥Stores.war`

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

ターゲット WebSphere Commerce Server 上のコマンド JAR ファイルの更新

このステップでは、以下のようにして、新しいコマンド JAR ファイルを使用するようにターゲット WebSphere Commerce Server を更新します。

1. 以下のようにして、既存の JAR ファイルのバックアップ・コピーを作成する必要があります。
 - a. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーに移動します。
 - b. `WebSphereCommerceServerExtensionsLogic.jar` ファイルのコピーを作成して、バックアップ場所に保管します。
2. 新しい `WebSphereCommerceServerExtensionsLogic.jar` ファイルを、`drive:¥ImportTemp4` ディレクトリーから、`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーにコピーします。

ここで `instanceName` は WebSphere Commerce インスタンスの名前です。

ターゲット WebSphere Commerce Server 上の EJB JAR ファイルの更新

このステップでは、以下のようにして、新しい EJB JAR ファイルを使用するようにターゲット WebSphere Commerce Server を更新します。

1. 以下のようにして、既存の JAR ファイルのバックアップ・コピーを作成する必要があります。
 - a. `WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーに移動します。
 - b. `Member-MemberManagementData.jar` ファイルのコピーを作成して、バックアップ場所に保管します。

2. 新しい Member-MemberManagementData.jar ファイルを、`drive:¥ImportTemp4` ディレクトリーから、`WAS_installdir¥installedApps¥cellName¥WC_instanceName.ear` ディレクトリーにコピーします。
3. 次に、以下のようにして、EJB デプロイメント記述子の情報に変更を加えなければなりません。
 - a. この WebSphere Application Server セルのデプロイメント・リポジトリー (META-INF ディレクトリー) を見つけます。これは一般的に、以下のような形式になります。


```
WAS_installdir¥config¥cells¥cellName
¥applications¥WC_instance_name.ear¥deployments¥
WC_instance_name¥EJBModuleName.jar¥META-INF。
```

 以下の形式は、このチュートリアルに固有の例です。

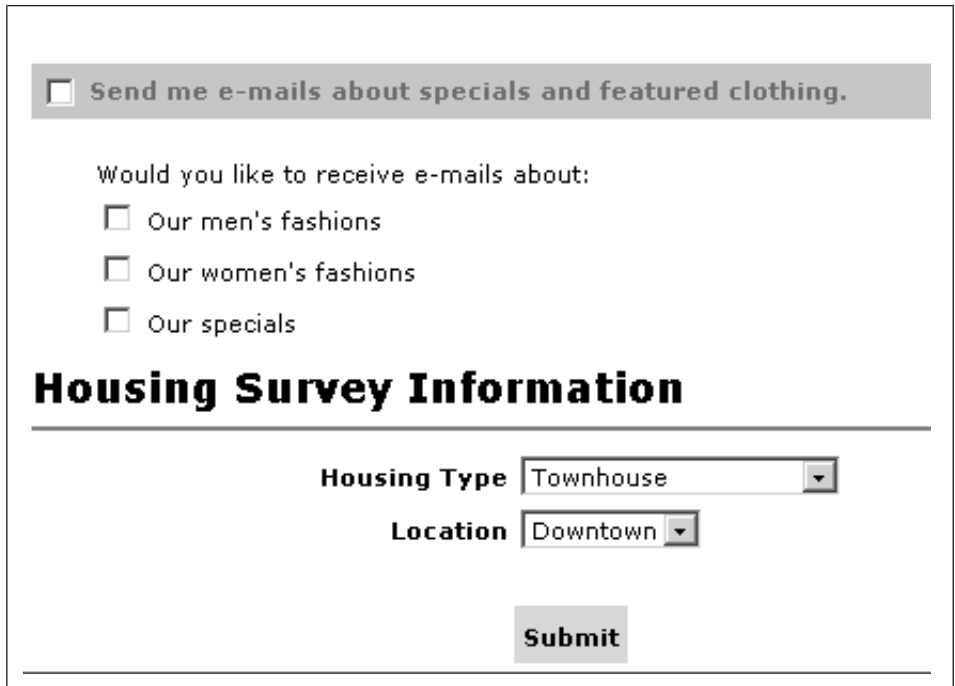

```
D:¥WebSphere¥AppServer¥config¥cells¥myCell¥applications¥
WC_demo.ear¥deployments¥WC_demo¥
Member-MemberManagementData.jar¥META-INF
```

 ここで、
 - myCell は、WebSphere Application Server セルの名前です。
 - demo は、WebSphere Commerce インスタンスの名前です。
 - Member-MemberManagementData は、カスタマイズされた EJB モジュールの名前です。
 - b. このディレクトリーには、以下のファイルが含まれています。
 - ejb-jar.xml
 - ibm-ejb-access-bean.xmi
 - ibm-ejb-jar-bnd.xmi
 - ibm-ejb-jar-ext.xmi
 - MANIFEST.MF
 これらすべてのファイルのバックアップを取ります。
 - c. ツールを使用して、新しい Member-MemberManagementData.jar ファイルをオープンし、その内容を表示します。
 - d. meta-inf ディレクトリーの内容を、この Member-MemberManagementData.jar ファイルから、ステップ 3a のディレクトリー中に抽出します。
4. コマンド行で WebSphere Application Server `startServer` コマンドを使用して、WebSphere Commerce インスタンスを再始動します。

ターゲット WebSphere Commerce Server 上のハウジング調査書ロジックの検証

このステップでは、以下のようにして、ハウジング調査書ロジックが、ターゲット WebSphere Commerce Server へ正常にデプロイされたことを検証します。

1. Web ブラウザーをオープンし、以下の URL を入力して、FashionFlow を基にしたご使用のストアを立ち上げます。
2. ストアのホーム・ページがオープンしたら、「登録」をクリックします。
3. もう一度「登録」をクリックして、新規ユーザーを作成します。
4. 変更された登録ページが表示され、以下の画面ショットに示されているように、ハウジング調査書が表示されます。



Send me e-mails about specials and featured clothing.

Would you like to receive e-mails about:

- Our men's fashions
- Our women's fashions
- Our specials

Housing Survey Information

Housing Type

Location

図 55.

5. 必要に応じて新規ユーザーの情報を入力し、「送信」をクリックします。
6. 情報を送信したら、「個人情報の変更」をクリックして、ハウジング情報が取り込まれたことを確認します。以下のように、調査書返信の要約が示された画面が表示されます。

Housing Survey Information

Housing Type: Townhouse

Location: Downtown

図 56.

第 5 部 付録

付録 A. WebSphere Commerce Studio での WebSphere Commerce コンポーネント・トレースの構成

この付録では、WebSphere Commerce 開発環境で実行するときに、さまざまな WebSphere Commerce コンポーネントのトレースを有効にする方法を説明します。コンポーネント・トレースを使用可能にするには、以下のようにします。

1. 必要であれば、WebSphere Commerce 開発環境をオープンします (「スタート」>「プログラム」>「IBM WebSphere Commerce Studio」>「WebSphere Commerce 開発環境」)。
2. サーバー・パースペクティブに切り替えます。
3. 「サーバー (Servers)」ビューで、**WebSphereCommerceServer** を右クリックし、「停止」を選択します (サーバーが稼働している場合)。
4. 「サーバー構成 (Server Configuration)」ビューで、「サーバー構成 (Server Configurations)」フォルダーを拡張表示します。
5. **WebSphereCommerceServer** をダブルクリックします。
WebSphereCommerceServer エディターがオープンします。
6. 「トレース」タブを選択します。
7. 「トレースを使用可能にする (Enable trace)」を選択します。
8. 「トレース・ストリング (Trace string)」テキスト・ボックスで、トレースを使用可能にする対象のコンポーネントを指定します。 WebSphere JRes 拡張機能のトレース・ロガー ID 値を使用し、後に "=all=enabled" を指定します。これらの値の完全なリストは、「*WebSphere Commerce 管理ガイド*」の『構成 (Configuration)』トピックを参照してください。複数のコンポーネントは、コロン (;) で区切るようにしてください。一例として、SERVER コンポーネントと RAS コンポーネントの両方にトレースを使用可能にするには、トレース・ストリングを以下のように指定します。

```
com.ibm.websphere.commerce.WC_SERVER=all=enabled:  
com.ibm.websphere.commerce.WC_RAS=all=enabled
```
9. 変更内容を保管します (Ctrl + S)。

改行がなされているのは、表記上の都合に過ぎません。

出力ファイル

出力ログ・ファイルは、デフォルトでは activity.log と呼ばれます。このファイルは、以下のディレクトリにあります。

```
workspace_dir¥.metadata¥.plugin¥com.ibm.etools.server.core¥tmp0¥logs
```

activity.log ファイルはバイナリー・ファイルであるため、このファイルを読むには、Log Analyzer を使用します。コンポーネント・トレースを使用可能にしたら、WebSphere JRes は、プレーン・テキスト形式で、ログ・エントリーを、トレース・エントリーと共にトレース出力ファイルに書き込みます。

WebSphere Commerce Studio での Log Analyzer ツールの構成については、「*WebSphere Commerce Studio* インストール・ガイド」を参照してください。

さらに、WebSphere Studio Application Developer の「コンソール」ビューにメッセージが表示されます。

付録 B. 追加情報の入手先

WebSphere Commerce Studio システムおよびそのコンポーネントについての詳細は、さまざまな形式のさまざまなソースで利用できます。以下のセクションでは、利用可能な情報とそのアクセス方法を示します。

WebSphere Commerce Studio 情報

WebSphere Commerce Studio の情報源は、以下のとおりです。

- 『WebSphere Commerce Studio オンライン・ヘルプ』
- 406 ページの『WebSphere Commerce Web サイト』
- 406 ページの『WebSphere Developer Domain』
- 406 ページの『IBM レッドブック』

WebSphere Commerce Studio オンライン・ヘルプ

WebSphere Commerce Studio オンライン情報は、WebSphere Commerce Studio でストアを作成して公開するときの主要な情報源です。

WebSphere Commerce Studio オンライン・ヘルプを表示するには、以下のようになります。

1. 「スタート」→「プログラム」→「IBM WebSphere Commerce Studio」→「WebSphere Commerce 開発環境」を選択して、WebSphere Commerce Studio を開始します。
2. 「ヘルプ」メニューで、「ヘルプ目次」を選択します。

注: 『WebSphere Commerce Studio オンライン・ヘルプ』で複数のプラットフォームの指示を参照する場合、WebSphere Commerce Studio の指示に従うようにしてください。ヘルプ・ページに複数のプラットフォームの情報が含まれる場合、WebSphere Commerce Studio 固有の情報は以下のアイコンで示されます。



WebSphere Commerce Studio 固有の情報が利用できない場合、Windows 固有の指示に従ってください。ヘルプ・ページに複数のプラットフォームの情報が含まれる場合、Windows 向けの指示は、以下のアイコンで示されます。



WebSphere Commerce Web サイト

WebSphere Commerce Studio 製品情報は、WebSphere Commerce Web サイトで利用できます。詳細な製品情報は、以下の URL を参照してください。

<http://www.ibm.com/software/webservers/commerce/library/>

WebSphere Developer Domain

WebSphere Commerce Studio および WebSphere Commerce の追加情報は、WebSphere Developer Domain の WebSphere Commerce Zone でも利用できます。

<http://www.ibm.com/websphere/developer/zones/commerce/>

IBM レッドブック

WebSphere Commerce Studio および WebSphere Commerce 情報は、IBM Redbooks™ Web サイトで利用できます。

<http://www.ibm.com/redbooks>

WebSphere Studio Application Developer 情報

WebSphere Studio Application Developer の情報源は、以下のとおりです。

- 『WebSphere Studio Application Developer オンライン・ヘルプ』
- 『WebSphere Studio Application Developer Web サイト』
- 407 ページの『WebSphere Developer Domain』
- 407 ページの『IBM レッドブック』

WebSphere Studio Application Developer オンライン・ヘルプ

WebSphere Studio Application Developer オンライン・ヘルプは、WebSphere Studio Application Developer 内でタスクを実行する方法に関する主要な情報源です。

WebSphere Studio Application Developer オンライン・ヘルプを表示するには、以下のようになります。

1. 「スタート」→「プログラム」→「IBM WebSphere Studio」→「Application Developer 5.0」を選択し、WebSphere Commerce Studio を開始します。
2. 「ヘルプ」メニューで、「ヘルプ目次」を選択します。

WebSphere Studio Application Developer Web サイト

WebSphere Studio Application Developer 製品情報は、WebSphere Studio Application Developer Web サイトで利用できます。

<http://www.ibm.com/software/ad/studioappdev/library/>

WebSphere Developer Domain

WebSphere Studio Application Developer の追加情報は、WebSphere Developer Domain の WebSphere Studio Zone の WebSphere Studio Application Developer ページで利用できます。

<http://www.ibm.com/websphere/developer/zones/studio/appdev/>

IBM レッドブック

WebSphere Studio Application Developer 情報は、IBM Redbooks Web サイトで利用できます。

<http://www.ibm.com/redbooks>

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 以外の製品とプログラムの操作またはサービス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。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒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を提供し、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。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更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認できません。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問い合わせください。

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ます、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 が小売り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。従って 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。

©Copyright International Business Machines Corporation 2000, 2003. このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 ©Copyright IBM Corp. 2000, 2003. All rights reserved.

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	@server
AIX	IBM
AS/400	iSeries
DB2	WebSphere
DB2 Universal Database	

Windows 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等はそれぞれ各社の商標です。

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]

- アクセス制御 97
 - コマンド・レベル 108
 - 保護可能なインターフェース 113
 - 保護リソース 114
 - ポリシー 101
 - リソース・レベル 108
 - Groupable インターフェース 114
- アダプター 10
- アプリケーション・アーキテクチャー 4
- 永続 51
- エラー処理 133
 - カスタム・コードの場合 136
 - コマンド 133
 - トレース 140
 - フロー 134
 - 例外のタイプ 133
 - JSP 141
- オブジェクトのライフ・サイクル 84
- オブジェクト・モデルの拡張方法 54

[カ行]

- カスタマイズ・コード
 - パッケージ 146
- 関係グループ 107
- コマンド
 - インターフェース 24
 - インプリメンテーション 143
 - 既存のものをカスタマイズ 160
 - コマンド・コンテキスト 147

- コマンド (続き)
 - コントローラー・コマンドの新規記述 149
 - タイプ 12
 - タスク・コマンドの新規記述 159
 - 登録 30
 - ビジネス・ポリシー・コマンドの新規記述 175
 - ファクトリー 26
 - フレームワーク 23
- コマンドのフロー 28
- コマンド・デザイン・パターン 23
- コマンド・レジストリー 30
- コントローラー・コマンド
 - 既存のものをカスタマイズ 160
 - 新規に作成 149
 - 長時間実行 152
- コントローラー・コマンド呼び出し側 Data Bean 46

[サ行]

- サブレット・エンジン 9
- 使用条件 181
- ソフトウェア・コンポーネント 3

[タ行]

- タスク・コマンド
 - 既存のものをカスタマイズ 164
 - 新規に作成 159
- データベースに関する考慮事項
 - データ・タイプ 92
 - 命名 90
- データベース・コミット 156
- データベース・ロック 85
- デザイン・パターン 21
 - コマンド 23
 - 表示 41

- デザイン・パターン (続き)
 - モデル・ビュー・コントローラー 21
- デプロイメント記述子 53
- トランザクションの有効範囲 156
- 取引の合意事項 169
- トレース、実行フローの 140

[ハ行]

- パッケージ、カスタマイズ・コードの 146
- ビュー・コマンド
 - 入力プロパティの形式 153
 - 必須プロパティ 49
- 表示デザイン・パターン 41
- プロトコル・リスナー 9

[マ行]

- メッセージ
 - プロパティ・ファイル 134
 - メッセージの作成 138
- モデル・ビュー・コントローラー・デザイン・パターン 21

[ラ行]

- ランタイム・アーキテクチャー 7

C

- CMDREG 32

D

- Data Bean
 - インターフェース 43
 - コマンド Data Bean 45
 - 入力 Data Bean 45

Data Bean (続き)

インターフェース (続き)

Smart Data Bean 43

活動化 45

既存のものをカスタマイズ 166

説明 14

タイプ 42

BeanInfo 45

W

Web コントローラー 11

E

Entity Bean

概要 51

拡張 54

キャッシュ 86

使用 89

説明 13

デプロイメント記述子 53

トランザクション 84

F

flushRemote メソッド 86

J

JSP テンプレート 14

属性の設定 47

S

Session Bean

新規に作成 82

推奨される使用法 59

U

URLREG 30

V

VIEWREG 36



Printed in Japan

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