Enterprise JavaBeans™ コンポーネントのプログラミング

Sun™ ONE Studio 4 プログラミングシリーズ

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 817-0838-10
2002 年 9 月 Revision A

Copyright © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. は、この製品に組み込まれている技術に関連する知的所有権を持っています。具体的には、これらの知的所有権には `http://www.sun.com/patents` に示されている 1 つまたは複数の米国の特許、および米国および他の各国における 1 つまたは複数のその他の特許または特許申請が含まれますが、これらに限定されません。

本製品はライセンス規定に従って配布され、本製品の使用、コピー、配布、逆コンパイルには制限があります。本製品のいかなる部分も、その形態および方法を問わず、Sun およびそのライセンサーの事前の書面による許可なく複製することを禁じます。

フォント技術を含む第三者のソフトウェアは、著作権法により保護されており、提供者からライセンスを受けているものです。

本製品には、RSA Data Security からライセンスを受けたコードが含まれています。

本製品の一部は、カリフォルニア大学からライセンスされている Berkeley BSD システムに基づいていることがあります。UNIX は、X/Open Company Limited が独占的にライセンスしている米国ならびに他の国における登録商標です。

Sun、Sun Microsystems、Forte、Java、NetBeans、iPlanet および docs.sun.com は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。

すべての SPARC の商標はライセンス規定に従って使用されており、米国および他の各国における SPARC International, Inc. の商標または登録商標です。SPARC の商標を持つ製品は、Sun Microsystems, Inc. によって開発されたアーキテクチャに基づいています。

サンのロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。

すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社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。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャに基づくものです。

Netscape および Netscape Navigator は、米国ならびに他の国における Netscape Communications Corporation の 商標または登録商標です。

Federal Acquisitions: Commercial Software -- Government Useres Subject to Standard License Terms and Conditions

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含み、明示的であるか黙示的であるかを問わず、あらゆる説明および保証は、法的に無効である限り、拒否されるものとします。

本製品が、外国為替および外国貿易管理法 (外為法) に定められる戦略物資等 (貨物または役務) に該当する場合、本製品を輸出または日本国外へ持ち出す際には、サン・マイクロシステムズ株式会社の事前の書面による承諾を得ることのほか、外為法および関連法規に基づく輸出手続き、また場合によっては、米国商務省または米国所轄官庁の許可を得ることが必要です。

原典： *Building Enterprise JavaBeans Components*
Part No: 816-7864-10
Revision A

目次

はじめに　xvii

1. Enterprise JavaBeans の概念　1

J2EE アーキテクチャ　2

EJB コンポーネントの役割　4

アプリケーションビルダーの役割　6

EJB アプリケーションの内側　7

エンタープライズ Bean の要素　8

Bean メソッド　8

インタフェースの種類　11

Bean クラス　14

EJB QL　15

配備記述子　15

EJB アプリケーションの実行時のワークフロー　16

エンタープライズ Bean の開発ライフサイクル　17

IDE のエンタープライズ Bean 機能　18

IDE を使用したエンタープライズ Bean の開発　19

関係 CMP エンティティ Bean セットの生成　20

トランザクション機能　20

持続性機能　21
セキュリティ機能　21
アプリケーションクライアントの作成　21
詳細情報の参照先　22

2. 設計とプログラミング　23

必要な Bean の種類の決定　23
セッション Bean　24
ステートレスセッション Bean の使用　25
ステートフルセッション Bean の使用　26
トランザクションモードの選択　27
セッション Bean のライフサイクル　29
エンティティ Bean　33
EJB コンテナのサービスの利用　34
エンティティ Bean のライフサイクル　35
関係 CMP エンティティ Bean のセットとコンテナ管理による関係　40
メッセージ駆動型 Bean　42
メッセージソースの使用 (送信先)　42
メッセージ駆動型 Bean が有効な利用形態　43
メッセージ駆動型 Bean が有効でない利用形態　44
メッセージ駆動型 Bean のライフサイクル　44
アプリケーションでのエンタープライズ Bean の使用　47
例外を使用した問題の対処　48
配備記述子の操作　49
セキュリティポリシーの適用　49
エンタープライズ Bean のセキュリティの宣言　50
エンタープライズ Bean のセキュリティのプログラミング　50
アプリケーションサーバーとデータベース　51

詳細情報の参照先　52

3. セッション Bean の開発　53

EJB ビルダーの使用　54

セッション Bean の種類の選択　55

ステートフルセッション Bean とステートレスセッション Bean　55

コンテナ管理によるトランザクションと Bean 管理によるトランザクション　57

セッション Bean の定義　58

パッケージの作成　58

EJB ビルダーのウィザードの起動　58

デフォルトのセッション Bean の作成　59

セッション Bean のクラスの参照　62

ノードの展開　63

生成されたクラスの確認　63

デフォルトの生成メソッド　63

ライフサイクルメソッド　64

セッション Bean の完成　66

推奨する エンタープライズ Bean の開発手順　66

生成メソッドの完成　66

ステートレス Bean の生成メソッドの完成　67

ステートフル Bean の生成メソッドの完成　67

ステートフル Bean への生成メソッドの追加　68

ライフサイクルメソッドの完成　68

`ejbPassivate` メソッドの完成　69

`ejbActivate` メソッドの完成　69

ビジネスメソッドの追加　70

トランザクションのコーディング　70

トランザクション範囲　71

トランザクション範囲とロールバックの指定　71
セッション Bean を作成した後の作業　74
詳細情報の参照先　75

4. CMP エンティティ Bean の開発　77

CMP エンティティ Bean 作成のための EJB ビルダーの使用　77
CMP および BMP エンティティ Bean の比較　79
関係 CMP エンティティ Bean のセットの作成　80
CMP エンティティ Bean の定義　81
パッケージの作成　81
データソースの準備　82
EJB ビルダーのウィザードの開始　83
CMP エンティティ Bean のインフラストラクチャの作成　83
データベースの表からの持続フィールドの指定　85
Bean の持続フィールドの新規作成　93
CMP エンティティ Bean のクラス　95
ノードの展開　96
生成されたクラスの確認　99
デフォルトの検索メソッド　99
持続フィールドと補助メソッド　99
主キークラスと必要なメソッド　101
CMP エンティティ Bean のライフサイクルメソッド　102
CMP エンティティ Bean の完成　104
推奨する エンタープライズ Bean の開発手順　105
生成メソッドの定義　105
主キーの追加または置き換え　107
主キーの新規作成　108
外部キーの取り扱い　109

ビジネスメソッドの定義　109
検索メソッドの追加　110
ホームメソッドの定義　112
選択メソッドの定義　113
追加フィールドの定義　114
CMP エンティティ Bean を作成した後の作業　114
詳細情報の参照先　115

5. 関係 CMP エンティティ Bean のセットの開発　117

EJB ビルダーと関係 CMP エンティティ Bean　118
関係 CMP エンティティ Bean の一括作成　118
関係 CMP エンティティ Bean のセットの作成　119
関係 CMP エンティティ Bean のセットの定義　119
パッケージの生成　120
データベースまたはスキーマを使用する準備　120
EJB ビルダーのウィザードの開始　121
Bean のセットのインフラストラクチャの生成　122
データベース接続の使用　123
データスキーマオブジェクトの使用　130
CMP エンティティ Bean のコンポーネント　130
EJB モジュールのノードの展開　132
生成されたクラスの確認　132
関係 CMP エンティティ Bean のセットの完成　132
推奨する エンタープライズ Bean の開発手順　134
セットへの Bean の追加　134
関係 CMP エンティティ Bean を作成した後の作業　137

6. BMP エンティティ Bean の開発　139

作成方法の決定　139
BMP エンティティ Bean の構築　140
パッケージの作成　140
EJB ビルダーのウィザードの起動　140
BMP エンティティ Bean のインフラストラクチャの生成　141
BMP エンティティ Bean のクラス　142
ノードの展開　142
生成されたクラスの確認　143
`findByPrimaryKey` メソッド　143
BMP エンティティ Bean のライフサイクルメソッド　143
BMP エンティティ Bean の完成　145
推奨する エンタープライズ Bean の開発手順　146
持続性ロジックの追加　146
主キークラスの追加　146
メソッドの追加　147
生成メソッドの定義　147
検索メソッドの追加　149
ビジネスメソッドとホームメソッドの定義　149
BMP エンティティ Bean を作成した後の作業　150
詳細情報の参照先　150

7. メッセージ駆動型 Bean の開発　151

EJB ビルダーとメッセージ駆動型 Bean　152
トランザクション管理の決定　153
メッセージ駆動型 Bean の定義　154
パッケージの作成　155
EJB ビルダーのウィザードの開始　155
基本のメッセージ駆動型 Bean の生成　155

メッセージ駆動型 Bean の参照　156
ノードの展開　156
生成されたクラスの確認　157
メッセージ駆動型 Bean の完成　158
推奨する エンタープライズ Bean の開発手順　159
`onMessage` メソッドの完成　159
`setMessageDrivenContext` メソッドの完成　160
メッセージ駆動型 Bean を作成した後の作業　160
メッセージ駆動型送信先の設定　161
メッセージセレクタの指定　162
クライアントメッセージ駆動型 Bean のリソースの指定　162
リソースファクトリの指定　162
リソースの指定　164
RI 上のリスナーメッセージ駆動型 Bean に対するリソースの指定　164
メッセージ駆動型 Bean 開発時の注意　165
詳細情報の参照先　166

8. エンタープライズ Bean の配備　167

配備情報とは　167
生成された配備記述子の参照　168
EJB モジュール配備記述子の編集　169
EJB モジュール配備記述子の直接編集　169
EJB モジュール配備記述子の状態復帰　169
プロパティシートによる配備記述子の編集　170
Bean のプロパティの指定　170
「プロパティ」タブ　170
エンティティ Bean のプロパティ　172
セッション Bean のプロパティ　172

メッセージ駆動型 Bean のプロパティ　172
「参照」タブの使用　173
EJB ローカル参照の指定　174
EJB 参照の指定　176
環境エントリの指定　176
リソース環境参照の設定　177
リソース参照の指定　178
セキュリティロール参照の指定　180
J2EE RI タブの使用　181
セッション Bean とエンティティ Bean に対する J2EE RI プロパティの設定　182
メッセージ駆動型 Bean に対する J2EE RI プロパティの設定　183
EJB モジュールの作成　184
EJB モジュールに格納する内容　185
EJB モジュールへの エンタープライズ Bean の格納　185
CMP エンティティ Bean へのデータベース関連プロパティの設定　187
RI の生成する SQL　188
CMP Bean による新規データベース表の使用　189
CMP Bean による既存データベース表の使用　192
EJB 1.1 CMP エンティティ Bean を含んだ EJB モジュール　192
CMP フィールドの値の順序　194
EMB モジュールへのトランザクション属性の追加　195
EJB モジュールまたはアプリケーション内での EJB 参照の変更　197
モジュールレベルでの参照指定の優先　197
アプリケーションレベルでの参照指定の優先　198
EJB JAR ファイルの作成　199
EJB モジュールへの JAR ファイルの追加　199
9. エンタープライズ Bean のテスト　201

テストの前提条件　201
J2EE RI への配備の準備　202
PointBase データベースでの Bean のテストの準備　202
データベースサーバーと Web ブラウザの開始　204
テストオブジェクトの生成　204
サーバーへのテストアプリケーションの配備　208
テストアプリケーションの配備と実行の一括処理　208
テストアプリケーションの実行　209
テストクライアントを使用した Bean のテスト　209
テストクライアントページとは　209
例題 Bean のホームインタフェースのテスト　211
例題 Bean のビジネスメソッドのテスト　212
テスト用クラスの作成　213
配備後の変更　214
異なる方法でのテスト準備　214
CMP または BMP Bean のテスト　214
EJB 参照を持つ Bean のテスト　215
Bean へのリモートインタフェースの追加　215
A. エンタープライズ Bean の操作　219
推奨する Bean の操作方法　219
論理ノードを使用した作業　219
カスタマイザまたはプロパティシートの利用　221
ソースエディタを利用した Bean の編集　221
IDE のエラー情報　223
エンタープライズ Bean のコンパイルと検証　224
変更の保存　226
エンタープライズ Bean 名の変更　226

ほかの Bean から取り込んだクラスの修正　227
エンタープライズ Bean のコピーとペースト　227
Bean のクラスやインタフェースの交換　228
Bean のメソッドの編集　229
メソッドの表示　230
エンティティ Bean のフィールドの変更　230
フィールド名の変更　230
フィールドの型の変更　230
エンタープライズ Bean の削除　231

B. EJB 1.1 エンタープライズ Bean の移行とアップグレード　233

現在のバージョンでの変更点　233
変更操作　235
CMP 1.x エンティティ Bean の変換　235
EJB 1.1 Bean での使用を避けるべき新規機能　236
CMP 1.x エンティティ Bean へのローカルインタフェースの追加禁止　236
CMP 1.x エンティティ Bean へのローカル EJB 参照の追加禁止　236
PointBase ユーザー名とパスワードの変更　237
`transient` 修飾子の禁止　237
EJB モジュールへの Bean の RI プロパティの移動　238
テスト前の CMP エンティティ Bean プロパティの変更　239

索引　241

図目次

図 1-1　Sun ONE Studio IDE がサポートする J2EE アプリケーションモデル　3

図 1-2　EJB アプリケーションの典型的な基本構成　4

図 1-3　3 種類のエンタープライズ Bean すべてを使用したアプリケーションの構成例　7

図 1-4　アプリケーションの実行時のワークフロー　16

図 1-5　エンタープライズ Bean の開発、アセンブル、および配備　18

図 1-6　生成されたエンタープライズ Bean 要素のエクスプローラウィンドウでの表示　20

図 2-1　Sun ONE Studio IDE でのエンタープライズ Bean の基本的な選択項目　24

図 3-1　ウィザードで選択できるステートレスセッション (またはステートフル BMT) Bean に対する指定　59

図 3-2　リモートインタフェースを持つ典型的なセッション Bean のデフォルトクラス　62

図 3-3　リモートインタフェースを持つ典型的なセッション Bean の詳細表示　63

図 4-1　CMP エンティティ Bean 作成時の EJB ビルダーウィザードでの選択項目　84

図 4-2　一般的な CMP エンティティ Bean のデフォルトクラス　95

図 4-3　ローカルインタフェースを持つ一般的な CMP エンティティ Bean のエクスプローラでの詳細表示　97

図 4-4　複合主キーを持つ一般的な CMP エンティティ Bean のエクスプローラでの詳細表示　98

図 5-1　EJB ビルダーのウィザードでの CMP エンティティ Bean に関する選択項目　122

図 5-2　関係 CMP エンティティ Bean の一般的なセットのデフォルトクラス　131

図 5-3　関係 CMP エンティティ Bean を含んだ EJB モジュールのノードの展開図　132

図 6-1　BMP エンティティ Bean のエクスプローラでの詳細表示　142

図 7-1　メッセージ駆動型 Bean のデフォルトクラスとメソッドの一般的な例　156

図 7-2　一般的なメッセージ駆動型 Bean のエクスプローラでの詳細表示　157

図 8-1　CMP エンティティ Bean の「プロパティ」ダイアログでの「参照」タブ　174
図 8-2　CMP Bean を含む EJB モジュールのプロパティを表示した「J2EE RI 」タブ　187
図 8-3　EJB モジュール内の CMP エンティティ Bean の表関連の設定　190
図 8-4　CMP Bean の `createTable` メソッドに対して RI プラグインが生成した SQL 文の例　191
図 8-5　CMP Bean の検索メソッドに対して生成された SQL 文の例　193
図 8-6　エンタープライズ Bean のローカル参照の上書きを選択した「EJB ローカル参照」プロパティエディタ　198
図 9-1　生成されたエンタープライズ Bean のテストオブジェクトの例　206
図 9-2　`dollarToYen` セッション Bean のテスト用に生成されたクライアント JSP ページ　210
図 9-3　Bean へインタフェースクラスを追加するカスタマイザ　217

表目次

表 3-1　ステートフルセッション Bean とステートレスセッション Bean の選択　56

表 3-2　コンテナ管理によるトランザクションと Bean 管理によるトランザクションの選択　57

表 3-3　セッション Bean クラスでのライフサイクルメソッドの目的　64

表 3-4　セッション Bean クラスでのセッション同期化メソッドの目的　65

表 3-5　トランザクションとメソッドの関係　71

表 4-1　CMPエンティティ Bean とBMP エンティティ Bean の選択　79

表 4-2　CMP エンティティ Bean クラスでのデフォルトのライフサイクルメソッドの目的　103

表 6-1　BMP エンティティ Bean クラスでのライフサイクルメソッドの目的　144

表 7-1　コンテナ管理または Bean 管理によるトランザクションの決定　153

表 7-2　メッセージ駆動型 Bean での `ejbCreate` メソッドと `onMessage` メソッドの目的　157

表 7-3　メッセージ駆動型 Bean の Bean クラス中にあるデフォルトのライフサイクルメソッドの目的　158

表 7-4　`setMessageDrivenContext` メソッドの例　160

はじめに

このマニュアルでは、Sun™ Open Net Environment (ONE) Studio, Enterprise Edition for Java の IDE を使用して Enterprise JavaBeans™ (エンタープライズ Bean) を開発する方法を説明します。

エンタープライズ Bean にはいくつかの種類があります。セッション Bean には、ステートフル (状態を保持する) とステートレス (保持しない) があります。どちらのセッション Bean も、自分自身のトランザクションを自分で管理したり、EJB™ コンテナに管理させたりすることができます。エンティティ Bean は、自分自身の持続性を管理したり、データベースとの関係を EJB コンテナに管理させたりすることができます。Sun ONE Studio IDE を使用すると、これらのエンタープライズ Bean を始め、メッセージ駆動型 Bean と EJB コンテナによって関係が管理されるエンティティ Bean を開発することができます。Sun ONE Studio IDE には、これらのエンタープライズ Bean の開発を柔軟に支援する機能があります。Sun ONE Studio IDE を使用すると、効率的にコーディングを行い、Java 2 Platform, Enterprise Edition Blueprints (J2EE™ Blueprints) に従ったコードを作成できるようになります。

このマニュアルシリーズの別のマニュアル、『J2EE アプリケーションのプログラミング』では、エンタープライズ Bean やその他の J2EE コンポーネントを利用した業務目的のアプリケーションの開発について説明しています。同マニュアルでは、様々なアプリケーションの利用形態を挙げているほか、作成したエンタープライズ Bean やその他のコンポーネントをモジュールにアセンブルする方法、これらのモジュールをアプリケーションに配備する方法、そして配備したアプリケーションを実行する方法を説明しています。このマニュアル、『Enterprise JavaBeans コンポーネントのプログラミング』はエンタープライズ Bean の設計と作成、アセンブル・配備・およびテストに関する基本事項を中心に記述しています。エンタープライズ Bean の作成、アセンブル、アプリケーションサーバーへの配備をすべて担当するプログラマは、両方のマニュアルを参照してください。

このマニュアルで説明しているプログラム例は、実際に作成することができます。作業環境については、以下の Web サイトにあるリリースノートを参照してください。

`http://sun.co.jp/forte/ffj/documentation/index.html`

使用するプラットフォームによっては、このマニュアルに掲載している画面イメージと異なることがあります。その場合でも表示上の違いはわずかであるため、内容を理解するのには問題ありません。ほとんどの手順で IDE のユーザーインタフェースを使用しますが、場合によっては、コマンド行にコマンドを入力する必要があります。その場合は、Microsoft Windows の「コマンドプロンプトウィンドウ」で次の構文を入力します。

```
c:\>cd MyWorkSpace\MyPackage
```

UNIX® や Linux 環境では、次のようなプロンプトとなり、¥ マーク (またはバックスラッシュ) ではなくスラッシュを使用します。

```
% cd MyWorkSpace/MyPackage
```

お読みになる前に

このマニュアルは、Sun ONE Studio IDE を使用してエンタープライズ Bean を開発する場合に役立ちます。前提知識として、次のことを理解しておく必要があります。

- Java プログラミング言語
- EJB コンポーネントモデル
- JDBC™ API および JDBC のドライバ構文
- リレーショナルデータベースの概念 (表、列、キーなど)
- 利用するデータベースの使用方法
- Java メッセージサービス (JMS) API
- XML 構文

エンタープライズ Bean を開発するには、J2EE の概念とエンタープライズ Bean についての一般的な知識が必要です。詳細情報が必要な場合は、次の資料を参照してください。

- *Enterprise JavaBeans Specification*, version 2.0
  `http://java.sun.com/products/ejb/docs.html`
- Java 2 Platform, Enterprise Edition Blueprints
  `http://java.sun.com/j2ee/blueprints`
- *Java 2 Platform, Enterprise Edition Specification*
  `http://java.sun.com/j2ee/download.html#platformspec`
- *The J2EE Tutorial* (J2EE SDK version 1.3)
  `http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html`
- *Java Message Service Tutorial*
  `http://java.sun.com/products/jms/tutorial/index.html`
- *Java Transaction API (JTA) Specification*
  `http://java.sun.com/products/jta`

Bean の開発中、特定のアプリケーションサーバーの知識が必要になる場合があります。詳細については、各サーバーのマニュアル等を参照してください。

注 - Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

内容の紹介

第 1 章では、J2EE と Enterprise JavaBeans の概念を紹介し、エンタープライズ Bean 作成と EJB モジュールのアセンブルに関する Sun ONE Studio IDE の機能について概説します。

第 2 章では、エンタープライズ Bean の開発と EJB モジュールのアセンブルに IDE を使用するプログラマ向けに、設計とプログラミングを説明します。

第 3 章では、トランザクションを自身で管理するステートレスセッション Bean とステートフルセッション Bean を作成する方法と、セッション Bean のトランザクション管理を EJB コンテナに管理させる方法について説明します。

第 4 章では、コンテナ管理による持続性を持つエンティティ Bean (CMP エンティティ Bean) を作成する方法について説明します。

第 5 章は、互いに関係を持つ複数の CMP エンティティ Bean を含んだセットを作成する方法を説明します。

第 6 章では IDE を使用して、Bean 管理による持続性を持つエンティティ Bean (BMP エンティティ Bean) を作成する方法を説明します。

第 7 章では IDE を使用してメッセージ駆動型 Bean を作成する方法を説明します。

第 8 章では、Bean や EJB モジュールのプロパティを指定することによって配備する Bean を準備する方法について説明します。

第 9 章では、エンタープライズ Bean をテストするための IDE のテスト機能について説明します。

付録 A では、IDE でのエンタープライズ Bean の操作方法について説明します。

付録 B では、EJB 1.1 エンタープライズ Bean を現在のバージョンの IDE で維持、実行できるように更新、変換する方法について説明します。

書体と記号について

書体または記号	意味	例
`AaBbCc123`	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	`.cvspass` ファイルを編集します。 `DIR` を使用してすべてのファイルを表示します。 `Search is complete.`
`AaBbCc123`	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表わします。	`> login` `Password:`
AaBbCc123 または ゴシック	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	削除するには **`DEL`** *filename* と入力します。 `rm` ファイル名 と入します。
『 』	参照する書名を示します。	『Solaris ユーザーマニュアル』
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 これらは、「クラス」オプションと呼ばれます。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合、バックスラッシュは、継続を示します。	`machinename% grep '^#define \` `  XV_VERSION_STRING'`
➤	階層メニューのサブメニューを選択することを示します。	作成:「返信」➤「送信者へ」

シェルプロンプトについて

シェル	プロンプト
UNIX の C シェル	`machine_name%`
UNIX の Bourne シェルと Korn シェル	`machine_name$`
スーパーユーザー (シェルの種類を問わない)	#

関連マニュアル

Sun ONE Studio 4 のマニュアルは、Acrobat Reader (PDF) ファイル、リリースノート、オンラインヘルプ、サンプルアプリケーションの `readme` ファイル、Javadoc™ 文書の形式で提供しています。

オンラインで入手可能なマニュアル

以下に紹介するマニュアルは、Sun ONE Studio 4 のドキュメントサイト (http://sun.co.jp/forte/ffj/documentation/index.html) および `docs.sun.com`™ (http://docs.sun.com) から入手できます。

`docs.sun.com` ウェブサイトでは、サンのマニュアルをインターネットを通じて閲覧、印刷、購入することができます。サイト内でマニュアルを見つけられない場合には、製品と一緒にローカルシステムまたはローカルネットワークにインストールされているマニュアルインデックスを参照してください。

- リリースノート (HTML 形式)

  Sun ONE Studio 4 の Edition ごとに用意されています。このリリースでの変更情報と技術上の注意事項を説明しています。

- インストールガイド (PDF 形式)

  対応プラットフォームへの Sun ONE Studio 4 統合開発環境 (IDE) のインストール手順を説明しています。さらに、システム要件、アップグレード方法、Web サーバーやアプリケーションサーバーのインストール、コマンド行での操作、インストールされるサブディレクトリ、Javadoc の設定、データベースの統合、アップデートセンターの使用方法などが含まれます。

  - 『Sun ONE Studio 4, Community Edition インストールガイド』 - Part No. 817-0845-10
  - 『Sun ONE Studio 4, Enterprise Edition for Java インストールガイド』 - Part No. 817-0844-10
  - 『Sun ONE Studio 4, Mobile Edition インストールガイド』 - Part No. 817-0846-10

- Sun ONE Studio 4 プログラミングシリーズ (PDF 形式)

  Sun ONE Studio 4 の各機能を使用して、優れた J2EE アプリケーションを開発するための方法を詳細に説明しています。

  - 『Web コンポーネントのプログラミング』 - Part No. 817-0837-10

    JSP ページ、サーブレット、タグライブラリを使用し、クラスやファイルをサポートする Web アプリケーションを J2EE Web モジュールとして構築する方法を説明しています。

  - 『J2EE アプリケーションのプログラミング』 - Part No. 817-0839-10

    EJB モジュールや Web モジュールを J2EE にアセンブルする方法を説明しています。また、J2EE アプリケーションの配備や実行についても説明しています。

  - 『Enterprise JavaBeans コンポーネントのプログラミング』 - Part No. 817-0838-10

    Sun ONE Studio 4 の EJB ビルダーウィザードや、他の IDE コンポーネントを使用し、EJB コンポーネント (コンテナ管理や Bean 管理の持続性の機能を持つセッション Bean やエンティティ Bean、メッセージ駆動型 Bean) を作成する方法を説明しています。

  - 『Web サービスのプログラミング』- Part No. 817-0816-10

Sun ONE Studio 4 IDE を使用して Web サービスを構築したり、UDDI レジストリを経由して第三者に Web サービスを利用させたり、また、ローカル Web サービスや UDDI レジストリから Web サービスクライアントを生成する方法などを説明しています。

- 『Java DataBase Connectivity の使用』 - Part No. 817-0840-10

  Sun ONE Studio 4 IDE の JDBC 生産性向上ツールを使用し、JDBC アプリケーションを作成する方法について説明しています。

- Sun ONE Studio 4 チュートリアル (PDF 形式)

  Sun ONE Studio 4 の Edition ごとに用意されており、主な機能の活用方法を紹介しています。

  - 『Sun ONE Studio 4, Community Edition チュートリアル』
    - Part No. 817-0842-10

    簡単な J2EE Web アプリケーションの構築方法を順を追って解説します。

  - 『Sun ONE Studio 4, Enterprise Edition for Java チュートリアル』
    - Part No. 817-0841-10

    EJB コンポーネントと Web サービス技術を使用したアプリケーションの構築方法を順を追って解説します。

  - 『Sun ONE Studio 4, Mobile Edition チュートリアル』
    - Part No. 817-0843-10

    携帯やPDA 端末などの無線機器を対象とした簡単なアプリケーションの構築方法を順を追って解説します。このアプリケーションは Java 2 Platform, Micro Edition (J2ME™ プラットフォーム) に準拠し、Mobile Information Device Profile (MIDP) と Connected, Limited Device Configuration (CLDC) を満たすものです。

  チュートリアルアプリケーションは、以下のサイトからもアクセスできます。
  `http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`

オンラインヘルプ

オンラインヘルプは、Sun ONE Studio IDE から参照できます。ヘルプを起動するには、ヘルプキー (Windows および Linux 環境では F1 キー、Solaris オペレーティング環境では Help キー) を押すか、「ヘルプ」 -> 「内容」を選択します。ヘルプの項目と検索機能が表示されます。

プログラム例

Sun ONE Studio 4 の機能を紹介したプログラム例とチュートリアルアプリケーションを、以下の Sun ONE Studio Developer Resource のポータルサイトからダウンロードすることができます。

```
http://forte.sun.com/ffj/documentation/tutorialsandexamples.html
```

このチュートリアルで使用するアプリケーションも上記サイトに収録されています。

Javadoc

Javadoc 形式のマニュアルは、Sun ONE Studio 4 の多くのモジュールに用意されており、IDE の中で参照できます。このマニュアルの使用方法については、リリースノートを参照してください。IDE を起動すると、エクスプローラの Javadoc タブで Javadoc マニュアルを参照できます。

ご意見の送付先

Sun のマニュアルについてのご意見やご要望をお寄せください。今後のマニュアル作成の参考にさせていただきます。次のアドレスまで電子メールをお送りください。

```
docfeedback@sun.com
```

電子メールのタイトルに、このマニュアルの Part No. (817-0838-10) を明記してください。

第1章

Enterprise JavaBeans の概念

Enterprise JavaBeans™ コンポーネント (エンタープライズ Bean) は、Java™ 2 Platform, Enterprise Edition (J2EE™) アーキテクチャの主要な構成要素です。この章では次の内容について説明しています。

- J2EE アーキテクチャの概要
- エンタープライズ Bean をはじめとする、J2EE モデルの EJB™ 層の各要素の役割
- EJB アプリケーションのコンポーネントとワークフロー
- EJB ビルダー : Sun™ Open Net Envirionment (ONE) Studio, Enterprise Edition for Java ソフトウェア (Sun ONE Studio 4) IDE のウィザードと GUI 機能の集合

すでに J2EE とエンタープライズ Bean 開発についての知識があり、IDE による Bean の作成および操作方法だけを知りたい場合は、第 3 章から第 9 章、および付録を参照してください。

エンタープライズ Bean は、関連する Bean とともに EJB モジュールに組み込み、アプリケーションにアセンブルし、サーバーに配備することで使用できるようになります。開発、アセンブル、配備の各作業は、複数の開発者や作業グループが専門知識に応じて分担できます。このマニュアルでは、EJB の開発者が、アプリケーションのアセンブルを行い、サーバーに配備するエンタープライズ Bean (または関連する Bean) を完成させるまでの作業を説明しています。J2EE アプリケーションの設計、アセンブルおよび配備については、マニュアル『J2EE アプリケーションのプログラミング』を参照してください。

注 - Sun ONE Studio 4 IDE は Enterprise JavaBeans 仕様 2.0 に対応しています。

J2EE アーキテクチャ

Java 2 Platform, Enterprise Edition (J2EE) のマニュアルセットでは、サービスに基づくアプリケーションアーキテクチャを説明しています。このアーキテクチャの内部に、トランザクション機能を持ち、スケーラブルかつ安全な、移植性がある Java コンポーネントを配備および再配備することができます。J2EE モデルのデータベース層、サーバー層、クライアントアクセス層を組み合わせると、企業全体をサポートするアプリケーションを開発することができます。

J2EE アプリケーションアーキテクチャは、基本的に次の機能から構成されます。

- クライアント層 - J2EE の仕様に基づき、この層にはブラウザ上で動作する HTML や Java アプレット、HTTP を介して転送される XML (Extensible Markup Language) 文書、クライアントコンテナ上で動作する Java クライアントを含めることができます。

  Sun ONE Studio 4 IDE は、Java クライアント、JSP ページ、サーブレット、およびその他のエンタープライズ Bean をクライアントとして使用するアプリケーションの実行と配備に対応しています。

- 1 つまたは複数の EJB 層またはサーバー層 - これらの層には次の機能を含めることができます。
  - プレゼンテーションロジック - Web サーバー上で動作するサーブレットや JavaServer Pages™ (JSP™ ページ)
  - アプリケーションロジック - アプリケーションサーバー上で動作する Enterprise JavaBeans コンポーネント (エンタープライズ Bean)
- データベース層

通常の J2EE アプリケーションのサーバー層には、図 1-1 に示す任意の、またはすべての要素を含めることができます。

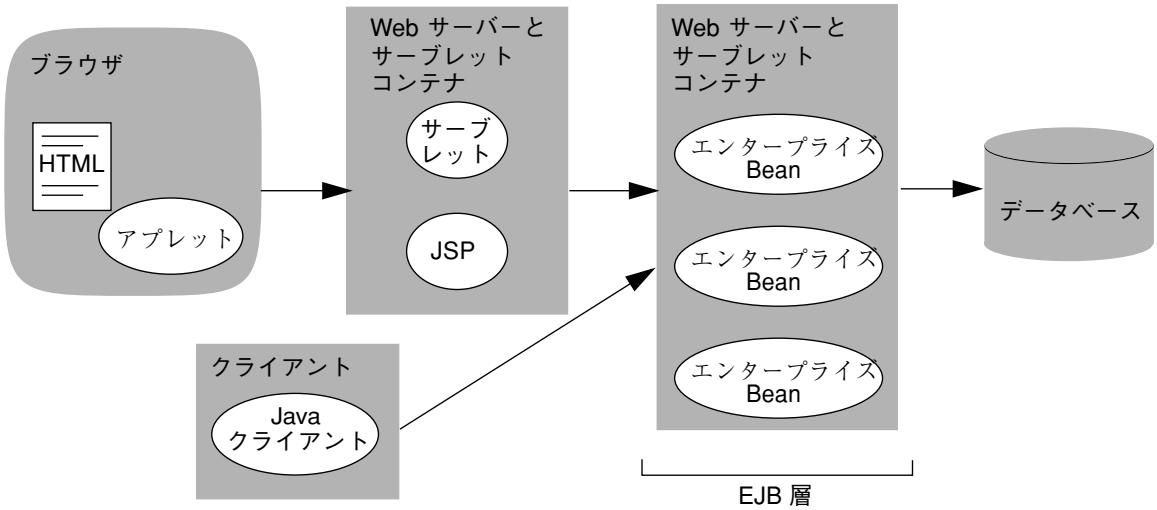


図 1-1　Sun ONE Studio IDE がサポートする J2EE アプリケーションモデル

エンタープライズ Bean は、ビジネスエンティティを構成する Java インタフェースと Java クラスからなる Java コンポーネントです。これらのインタフェースやクラスは、アプリケーションサーバー上にビジネスロジックを実装するメソッドを含んでいます。これらのメソッドに加えて、データベースの列に対応付けられるフィールドを含んでいるエンタープライズ Bean もあります。また、同じアプリケーション中の異なるエンタープライズ Bean 間の対話処理を管理する能力を持ったエンタープライズ Bean もあります。エンタープライズ Bean と、図 1-1 に示す任意の種類のコンポーネントを組み合わせて、アプリケーションを作成できます。

エンタープライズ Bean と JavaBeans™ コンポーネントは、どちらも Java プログラミング言語で作成しますが、これらは同じではありません。JavaBeans コンポーネントは、Java クラスのインスタンスをカスタマイズするために、設計ツールとともに使用します。カスタマイズしたオブジェクトは、イベントを介してリンクすることができます。これに対して、エンタープライズ Bean は、マルチユーザー用の分散型のコンテナ管理によるトランザクションサービスを実装します。

EJB 層により、Java コンポーネントのモジュール性と移植性がさらに強化されます。そのため、EJB 開発者の業務が一層モジュール化され、開発者は分散コンピューティングよりも、アプリケーションのビジネスデータに集中することができます。JavaBeans コンポーネントを使用してアプリケーションを作成する場合は、サーバーフレームワークも作成する必要があります。しかし、エンタープライズ Bean を使った J2EE モデルでアプリケーションを作成する場合は、サーバー側のインフラストラ

クチャはアプリケーションサーバーにすでに組み込まれています。したがって、トランザクション機能、セキュリティ機能、リモートアクセス機能といった一般的なサービスを提供する必要はありません。

EJB コンポーネントの役割

典型的な EJB アプリケーションの基本構成 (一部を除きます) を図 1-2 に示します。この中には、アプリケーションクライアント、アプリケーションサーバー、EJB コンテナ、少なくとも 1 つのエンタープライズ Bean、そして何らかのデータソースが含まれています。この図ではデータベースを使用しています。

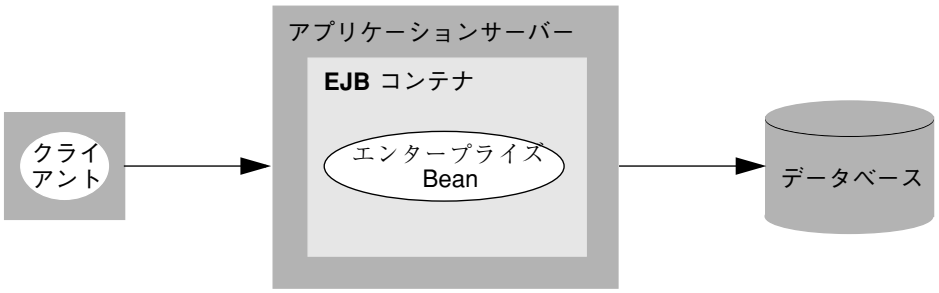


図 1-2　EJB アプリケーションの典型的な基本構成

エンタープライズ Bean、EJB コンテナ、アプリケーションサーバーの間の契約 (対話処理と暗黙の合意) により、エンタープライズ Bean の作成作業が簡単になるとともに、J2EE アプリケーションの柔軟性と機能性が一層高まります。

EJB コンテナは、オブジェクトというよりも、概念です。EJB コンテナは、EJB サーバー上のエンタープライズ Bean を取り囲む環境で、ライフサイクル管理、セキュリティ、分散トランザクション機能といったサービスを提供します。

1 つのコンテナに、1 つ以上のエンタープライズ Bean を配備できます。それぞれのコンテナは、標準の JNDI (Java Naming and Directory Interface™) API を使用して、個々の Bean を検出し、クライアントから使用できるようにします。

コンテナは、Bean とそのクライアントとの仲介役になります。クライアントがエンタープライズ Bean に処理要求を送ると、コンテナがそのメソッドに対する呼び出しを代替受信します。コンテナは、エンタープライズ Bean やクライアントに代わっ

て、(セキュリティやトランザクションといった) サービス、コンポーネント、さらにほかのサーバー上で動作しているほかのコンテナを管理できます。この機能によって、コンテナは柔軟で透過的なサービスを実行できます。

コンテナは、個々のエンタープライズ Bean のために、データベースの持続性とトランザクションを管理します。そのため、状態管理イベントを標準的な手法で処理することができます。また、Bean 自身がデータベースへのアクセスを管理できるため、EJB 開発者が完全な SQL コードを記述したり、JDBC™ API を直接使う必要がありません (ただし、コンテナのデフォルトの動作を無効にしたい場合は EJB 開発者がこれらの処理をする必要があります)。

クライアントが異常終了したり、サーバーが停止したりした場合は、コンテナのサービスによってエンタープライズ Bean の持続データが確実に保存されます。

アプリケーションサーバーは、ネームサービス、ディレクトリサービス、電子メールサービスといった基本的な機能を提供します。

エンタープライズ Bean には、セッション Bean、エンティティ Bean、およびメッセージ駆動型 Bean の 3 種類があります。これらの Bean については、第 2 章以降で詳しく取り上げます。ここでは、これらの Bean の役割の概要を説明するにとどめます。

- セッション Bean は、クライアントと EJB サーバーとの対話を管理し、エンティティ Bean との複雑な対話を管理できます。たとえば、セッション Bean はデータ要求を エンティティ Bean に渡し、取得したデータをパッケージ化し、それをクライアントに戻すといった処理をします。
- エンティティ Bean は、通常、データベース中のエンティティ (データの表) を表します。多くのエンティティ Bean はアプリケーションサーバー上の Java 仮想マシン (JVM™) 内で協調的に動作します。
- メッセージ駆動型 Bean は、クライアントとサーバー間で機能層を構成します。メッセージ駆動型 Bean はクライアントメッセージ通知を受信すると、同じサーバーに配備されている他のエンタープライズ Bean と非同期型の対話を開始します。

作成したエンタープライズ Bean は、EJB モジュール (EJB JAR ファイルの論理構成) にパッケージ化され、アプリケーションにアセンブルされて、サーバーへ配備されます。アプリケーションサーバーは 1 つ以上の J2EE アプリケーションから構成され、J2EE アプリケーションは 1 つ以上の EJB モジュールから構成され、EJB モジュールは 1 つ以上のエンタープライズ Bean から構成されます。

アプリケーションビルダーの役割

J2EE アーキテクチャは、現在のアプリケーション開発プロセスでの責任分担の方法論と支援のあり方を異なる役割へと変える可能性を秘めています。通常の開発組織には、業務の知識が豊富なチームメンバーもいれば、システムレベルの開発に精通しているチームメンバーもいます。誰もが異なる作業を担当している組織に J2EE モデルを適用した場合、業務や組織に関する知識が豊富なメンバーは Bean 提供者や EJB 開発者としての役割を担当し、システム開発に詳しいメンバーは Bean のアセンブルや配備を担当するといった分担が考えられます。

開発環境では、少なくとも次の 3 つの役割があるのが一般的です。

- **EJB の開発** - EJB 開発者 (アプリケーション開発者であり、通常は組織について詳しい人) は、エンタープライズ Bean を作成します。作成の際に、フレームワークを考慮する必要はありません。テストを目的として、または利便性を考慮して EJB 開発者が作成した Bean を EJB モジュールにパッケージ化することもあります。
- **EJB モジュールのアセンブル** - アセンブル担当者は作成したエンタープライズ Bean を EJB モジュールにパッケージ化し、これらのモジュールと他の J2EE 構築ブロックを組み合わせてアプリケーションを作成します。 J2EE を使ったモジュール化ができるため、この時点ではコンテナに依存しない決定ができます。
- **アプリケーションの配備** - 配備担当者はコンテナ固有およびサーバー固有の決定を行いながら、作成した J2EE アプリケーションを特定の環境に配備します。

エンタープライズ Bean のサポート機能を内蔵した IDE は、J2EE のアプリケーション開発手法に役立つように作成されています。IDE を使ったエンタープライズ Bean の開発工程では、EJB 開発者はアプリケーションに必要なビジネスロジックを記述することに集中できます。EJB 開発の役割の中では、アセンブル段階や配備段階に関して最小限のことだけを考慮するだけで充分です。

ただし、必要であれば、上の 3 つの役割を 1 人の人がすべて担当することができます。IDE には、エンタープライズ Bean の開発、アセンブル、および配備のすべての段階を途切れることなく行える機能があります。

EJB アプリケーションの内側

図 1-3 に示すような典型的な EJB アプリケーションでは、EJB サーバー上に常駐し、EJB コンテナによって管理されるアプリケーションの心臓部に、多数のクライアントプログラムが同時にアクセスできます。EJB 層の内部では、2 つの異なるセッション Bean (このうちの 1 つはメッセージ駆動型 Bean によって開始されます) のインスタンスが、4 つのエンティティ Bean との対話処理を管理し、クライアントが日程を参照し、会議室を予約できるようにします。データベースのデータがエンティティ Bean のインスタンスに読み込まれ、クライアントがエンティティ Bean のインスタンスに適用した更新がデータベースに書き込まれます。

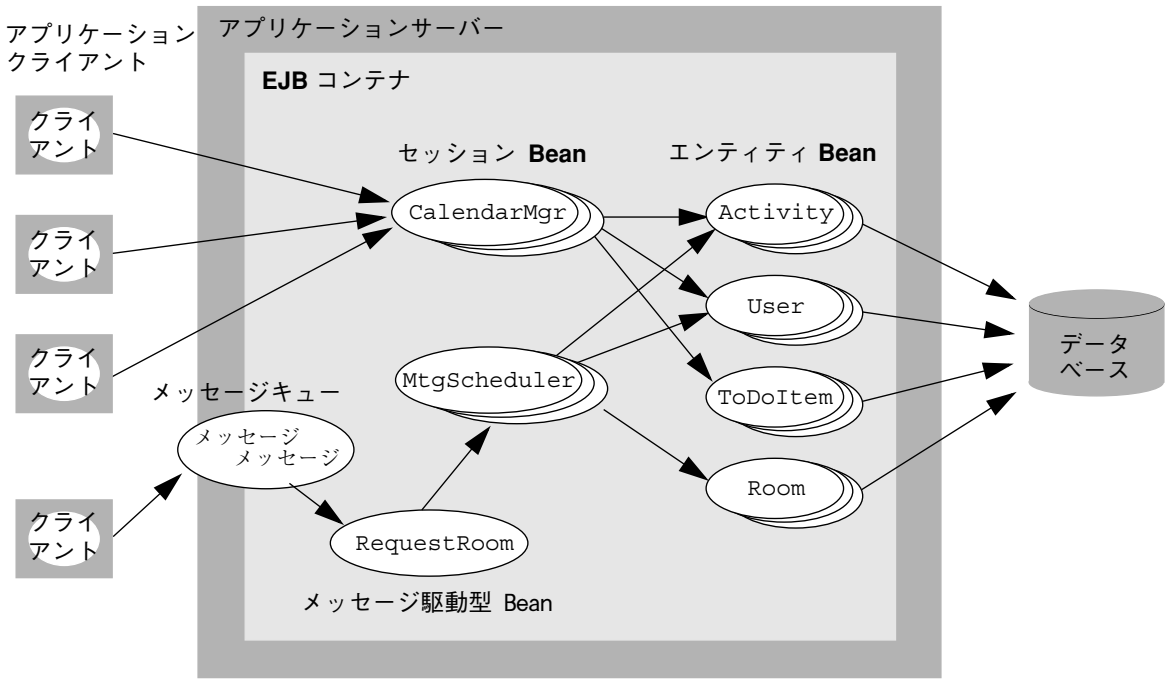


図 1-3　3 種類のエンタープライズ Bean すべてを使用したアプリケーションの構成例

エンタープライズ Bean の要素

それぞれのエンタープライズ Bean には少なくとも 1 つのクラスがあります。

- メッセージ駆動型 Bean にはインタフェースを持たない自身の Bean クラスだけが必要です。
- セッション Bean は、通常、自身の Bean クラス 1 つとリモート型インタフェース 2 つ (ホームインタフェースとリモートインタフェース) から成る 3 つの要素で構成されます。また、セッション Bean はローカル型インタフェースを持つこともできます。
- エンティティ Bean は、通常、自身の Bean クラス 1 つとリモート型インタフェース 2 つ (ローカルホームインタフェースとローカルインタフェース) から成る 3 つの要素で構成されています。エンティティ Bean もローカル型インタフェースを持つことができ、また、主キークラスを持つこともあります。

エンタープライズ Bean インタフェースの詳細については、 11 ページの「インタフェースの種類」を参照してください。

EJB アプリケーションの要素の役割は次のようになります。

- EJB 開発者は、IDE を使用して、Bean クラスとそれぞれのエンタープライズ Bean に対するインタフェースを作成します。必要であれば、エンティティ Bean に対して主キークラスも定義します。EJB 開発者は、IDE が生成したコードを完成させて、配備情報を宣言します。
- Bean が配備されるコンテナは、Bean のインタフェースを実装し、コンポーネントとデータストレージとの対話を管理します。
- Bean を使用するクライアントは、Bean のインタフェースを呼び出すスタブを作成します。このインタフェースを介して Bean クラスと対話し、アプリケーションの処理を実行させることができます。または、クライアントが目的の相手にメッセージを送信し、メッセージ駆動型 Bean がこれらのメッセージを待機しているため、必要に応じてセッション Bean と対話します。

Bean メソッド

J2EE アプリケーションでは、クライアントが呼び出す Bean メソッドによって処理が実行されます。ここからは、エンタープライズ Bean のメソッドの概要を説明します (これらのメソッドの詳細については、第 3 章から第 7 章を参照してください)。メ

ソッドの宣言は、IDE によって自動的に追加されるか、EJB 開発者が明示的に追加します。ダイアログで 1 つの簡単な手続きに従うだけで、メソッドの宣言に必要なあらゆる情報を追加することができます。IDE は、それに対応するメソッドの情報を生成し、適切なクラスに配置します。

- **検索メソッド** - クライアントはホームインタフェースを介して、エンティティ Bean のインスタンスを主キーを基準にして検索します。EJB 開発者は、他の検索メソッドを追加することもできます。

  IDE は、各エンティティ Bean のローカルホームインタフェースに、また、その Bean 中にホームインタフェースがあればそのホームインタフェースにも、`findByPrimaryKey` メソッドの宣言を自動生成します。さらに、持続性を自分自身で管理するエンティティ Bean (これを Bean 管理による持続性 (BMP) エンティティ Bean という) の Bean クラスに、それに対応する `ejbFindByPrimaryKey` メソッドの宣言を配置します。EJB 開発者が他の検索メソッドを追加した場合、IDE はそれに対応するメソッドの宣言をローカルホームインタフェース (およびホームインタフェース) に、また、BMP エンティティ Bean に対しては Bean クラスに自動的に追加します。

  追加した検索メソッドには EJB 照会言語 (EJB QL: EJB Query Language) 文が含まれています。この文は、Bean のアプリケーションサーバープラグインによって、サーバーが必要とする SQL コードに自動的に変換されます。

- **生成メソッド** - コンテナは生成メソッドの引数を使用して、エンタープライズ Bean のインスタンスを初期化します。

  IDE は、各セッション Bean のホームインタフェースに (Bean のローカルホームインタフェースがあれば、それにも) 生成メソッドの宣言を生成します。さらに Bean クラスに、対応する `ejbCreate` メソッドの宣言を配置します。

  また、IDE は各メッセージ駆動型 Bean の Bean クラスに `ejbCreate` メソッドの宣言を生成します。

  エンティティ Bean には生成メソッドは必要ありません。そのため、エンティティ Bean については、この宣言は自動的に生成されません。ただし、EJB 開発者がエンティティ Bean に生成メソッドを追加した場合は、IDE はそれに対応する `create` メソッド、`ejbCreate` メソッド、および `ejbPostCreate` メソッドの宣言を、該当するクラスに配置します。エンティティ Bean には 1 つ以上の生成メソッドを組み込むことができます。

- **ビジネスメソッド** - クライアントはリモートインタフェース (場合によってはローカルインタフェース) を介して、Bean のビジネスメソッドを呼び出します。

  EJB 開発者が、Bean にビジネスメソッドを明示的に追加します。IDE は、デフォルトのビジネスメソッドの宣言を生成しません。ただし、EJB 開発者がビジネスメソッドを指定した場合は、IDE はそれに対応するメソッドの宣言を、Bean クラスとリモートインターフェース、ローカルインタフェース、またはリモートとローカルのインタフェース両方に配置します。

- **ホームメソッド** - エンティティ Bean は、Bean 中の特定のインスタンスへのアクセスが必要とされない軽量操作にはホームメソッドを使用します (逆に、ビジネスメソッドの使用時には、特定のインスタンスへのアクセスが必要となります)。EJB 開発者がホームメソッドを明示的に追加すると、IDE によって、そのホームメソッドに対応するメソッド宣言が Bean クラスと Bean のローカルホームインタフェースまたはホームインタフェースに生成されます。

- **選択メソッド** - 選択メソッドを使用できるのは、持続性をコンテナが管理するエンティティ Bean (これをコンテナ管理による持続性 (CMP) エンティティ Bean という) です。検索メソッドのように、選択メソッドはデータベースに対する照会を実行して、ローカルインターフェース、リモートインタフェース、またはコレクションを戻します。さらに、選択メソッドは、同じ EJB モジュール中にある関係エンティティ Bean に対する照会を実行し、その結果の値を持続フィールドを介して戻すこともできます。リモート型インタフェースから選択メソッドに直接アクセスすることや、クライアントから選択メソッドを起動することはできません。

  選択メソッドは、EJB 開発者が Bean クラスに明示的に 1 つ以上追加します。選択メソッドには EJB 照会言語 (EJB QL) 宣言が含まれます。EJB QL は、Bean のアプリケーションサーバーのプラグインによって、そのサーバーが必要とする SQL コードに自動的に変換されます。

- **`OnMessage` メソッド** - クライアントは、Java メッセージサービス (JMS: Java Message Service) 送信先にメッセージを送信することで、メッセージ駆動型 Bean 上の `OnMessage` メソッドを呼び出します。

  `onMessage` メソッドの宣言は、IDE によって Bean クラス中に自動的に生成されます。EJB 開発者はメソッドの本体を完成させます。

- **ライフサイクルメソッド** - コンテナは、いくつかのメソッドを呼び出して、エンタープライズ Bean のライフサイクルを管理します。Bean の種類によって、これらのメソッドの取り扱い方法に多少の違いがあります。一部のメソッドについては、EJB 開発者がパラメータを指定することができます。

IDE は、Bean の種類に応じて適切なライフサイクルメソッドの宣言を生成し、Bean クラスに配置します。

インタフェースの種類

セッション Bean は、エンタープライズ Bean が常駐しているアプリケーションサーバーの外にあるアプリケーションクライアントによって頻繁に呼び出されます。このため、IDE では、各セッション Bean に対し、リモート型インタフェース (リモートインタフェースとホームインタフェースを指す) をデフォルトで生成します。EJB 開発者は、セッション Bean の作成時にリモートインタフェースとホームインタフェースのどちらかまたは両方を、そのセッション Bean が呼び出される形態に応じて選択できます。

エンティティ Bean は通常、セッション Bean と同じアプリケーションサーバーにある他のエンティティ Bean によって呼び出されます。このようなエンティティ Bean にはローカル型インタフェース (ローカルインタフェースおよびローカルホームインタフェースを指す) があれば充分です。ローカル型インタフェースを使用すると、パラメータ値をシリアライズすることなく参照によってパラメータを渡すことができるため、処理時間が短縮できます。これらのインタフェースも、EJB 開発者がエンティティ Bean の作成時にどちらか、あるいは両方を使用するのかどうかを状況に応じて指定できます。

注 - IDE のテスト機能を使用してテストする Bean にはすべて、リモートインタフェースが必要です。

これら 4 種類のインタフェースについて次に説明します。

リモートインタフェース

クライアントは、エンタープライズ Bean のリモートインタフェースを介して Bean を参照したり、アクセスしたりします。クライアントが呼び出す対象となる、その Bean のビジネスメソッドのシグニチャはリモートインタフェース中にあります。ただし、そのビジネスメソッドの完全なコードは、Bean クラスにあります。コンテナは、リモートインタフェースを実装するクラスを作成します。

リモートインタフェースは `javax.ejb.EJBObject` のサブクラスです。クライアントはこのインタフェースを介し、JNDI ルックアップ呼び出しを使用してホームインタフェースを検出します。次に、ホームインタフェース中のメソッドを呼び出して、戻り値の型にリモートインタフェースを持つ Bean 固有のインスタンスを取得し、そのインスタンス中のビジネスメソッドを呼び出します。

Sun ONE Studio IDE を使用してリモート型インタフェースを持つエンタープライズ Bean を作成すると (サーバーの外から呼び出しはこのリモート型インタフェースを介して実行される)、EJB ビルダーの GUI 機能と検証機能により、リモートインタフェースのメソッドが J2EE マニュアルで定義されている規則に従っているかどうかが確認されます。これらの規則には次のようなものがあります。

- リモートインタフェースのメソッドシグニチャに対応するメソッドが、Bean クラスに存在する
- 引数と戻り値は有効な RMI 型である
- メソッドの `throws` 句に適切な例外クラスが含まれている

IDE のエクスプローラウィンドウでは、インタフェースのノードは のように表示されます。デフォルトのラベルは、該当するエンタープライズ Bean の名前となります。

ホームインタフェース

エンタープライズ Bean のホームインタフェースは `javax.ejb.EJBHome` のサブクラスです。このインタフェースでは、クライアントがエンタープライズ Bean を呼び出すことのできる生成メソッドと検索メソッドを定義します。クライアントは JNDI を使用してホームインタフェースを検出し、コンテナはホームインタフェースを実装するクラスを提供します。

IDE を使用してリモート型インタフェースを持つエンタープライズ Bean を作成すると、EJB ビルダーの GUI 機能と検証機能により、ホームインタフェースのメソッドがエンタープライズ Bean の基本規則に従っているかどうかが確認されます。これらの規則には次のようなものがあります。

- ホームインタフェースのメソッドシグニチャに対応するメソッドが、Bean クラスに存在する (コンテナに持続性を管理させるエンタープライズ Bean の検索メソッドを除く)
- 引数と戻り値は有効な RMI 型である

- メソッドの `throws` 句に適切な例外クラスが含まれている

IDE のエクスプローラウィンドウでは、ホームインタフェースのノードは のように表示されます。デフォルトのラベルは、 *Bean_name*`Home` となります。

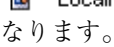
ローカルインタフェース

ローカルインタフェースはリモートインタフェースに似ています。この型のインタフェースには、Bean 上で呼び出すことのできるビジネスメソッドのシグニチャが含まれています。メソッドの完全なコードは Bean クラスに含まれています。ローカルインタフェースを実装するクラスはコンテナによって作成されます。Bean のローカルインタフェース呼び出しは、同じサーバー上にある他の Bean または Web コンポーネントからの呼び出しに限ります。

ローカルインタフェースは、 `javax.ejb.EJBOLocalObject` のサブクラスです。クライアントはこのインタフェースを介し、JNDI ルックアップ呼び出しを使用してローカルホームインタフェースを検出します。次に、ローカルホームインタフェースのメソッドを呼び出して、 (値を戻すときにリモートインタフェースの使用が指定されている) Bean 固有のインスタンスを取得し、そのインスタンス上のビジネスメソッドを呼び出します。

Sun ONE Studio IDE を使用してローカル型インタフェースを持つエンタープライズ Bean を作成すると、EJB ビルダーの GUI 機能と検証機能により、ローカルインタフェースのメソッドが J2EE のマニュアルで定義されている規則に従っているかどうかが確認されます。これらの規則には次のようなものがあります。

- ローカルインタフェースのメソッドシグニチャに対応するメソッドが、Bean クラスに存在する
- メソッドの `throws` 句に適切な例外クラスが含まれている

IDE のエクスプローラウィンドウでは、リモートインタフェースのノードは  のように表示されます。デフォルトのラベルは、`Local`*Bean_name* となります。

ローカルホームインタフェース

エンタープライズ Bean のローカルホームインタフェースは、ホームインタフェースと同じように `javax.ejb.EJBLocalHome` のサブクラスです。また、同じサーバー上の他の エンタープライズ Bean から呼び出すことのできるエンタープライズ Bean 上の生成メソッドと検索メソッドを定義します。ローカルホームインタフェースを実装するクラスはコンテナが提供します。

IDE を使用してローカル型インタフェースを持つエンタープライズ Bean を作成すると、EJB ビルダーの GUI 機能と検証機能により、ローカルホームインタフェースのメソッドがエンタープライズ Bean の基本規則に従っているかどうかが確認されます。これらの規則には次のようなものがあります。

- ローカルホームインタフェースのメソッドシグニチャに対応するメソッドが、Bean クラスに存在する (コンテナに持続性を管理させるエンタープライズ Bean の検索メソッドを除く)
- メソッドの `throws` 句に適切な例外クラスが含まれている

エクスプローラウィンドウに表示されるリモートインタフェースのノードは のようになります。デフォルトのラベルは `Local`*Bean_name*`Home` となります。

Bean クラス

Bean クラスは、ほかの 2 つのクラスで定義されたメソッドの実装を含んでおり、エンタープライズ Bean の心臓部です。エンティティ Bean の Bean クラスは `javax.ejb.EntityBean` インタフェースのサブクラス、セッション Bean の Bean クラスは `javax.ejb.SessionBean` インタフェースのサブクラス、そしてメッセージ駆動型 Bean の Bean クラスは `javax.ejb.MessageDrivenBean` のサブクラスです。Bean クラスでは、エンタープライズ Bean の検索メソッド、生成メソッド、ビジネスメソッド、ホームメソッド、および選択メソッドを実装します。さらに、コンテナから呼び出されるライフサイクルメソッドも実装します。

Sun ONE Studio IDE を使用してエンタープライズ Bean を作成すると、EJB ビルダーの GUI 機能と検証機能により、Bean クラスがエンタープライズ Bean の基本規則に従っているかどうかが確認されます。これらの規則には次のようなものがあります。

- Bean クラスが abstract クラスおよび public クラスとして定義されている
- Bean クラスにパラメータのない public なコンストラクタが含まれている

- Bean クラスに、ホームインタフェースまたはローカルホームインタフェースで定義された生成メソッドに対応する `ejbCreate` メソッドが実装されている
- 自分自身のトランザクションを管理するエンティティ Bean の場合は、Bean クラスに、ホームインタフェースまたはローカルホームインタフェースで定義された検索メソッドに対応する `ejbFind` メソッドが含まれている

IDE のエクスプローラウィンドウでは、Bean クラスのノードは のように表示されます。デフォルトのラベルは、*Bean_name*`Bean` となります。

EJB QL

検索メソッドまたは選択メソッドを CMP エンティティ Bean に追加すると、EJB 照会言語 (EJB QL) に、メソッドの照会を定義する宣言を埋め込んだことになります。EJB QL に記述された照会を使用して、Bean は、抽象スキーマに定義されている関係に従いナビゲートできるようになります。抽象スキーマは、Bean の持続フィールドと関係を定義する配備記述子の一部です。EJB QL による照会は、同じ EJB JAR ファイルにパッケージされているすべての関係エンティティ Bean の抽象スキーマにわたって実行されます。

Bean がアプリケーションサーバーに配備されると、EJB QL による照会はデータが保存されている形式の言語に翻訳されます。このことによって、EJB QL を使用しているエンティティ Bean は異なる方式で保存されているデータ間で移植することができます。

配備記述子

エンタープライズ Bean の配備記述子は、Bean のサーバーへの配備方法を記述したものです。配備記述子は、エンタープライズ Bean を構成しているクラス、Bean から他の Bean への参照、Bean の実行環境の設定、および実行時の Bean の管理方法をリストし、記述した XML ファイルです。また、このファイルには、コンテナが持続性の管理をするエンティティ Bean の持続フィールドもリストされています。

Sun ONE Studio IDE を使用してエンタープライズ Bean を作成すると、EJB ビルダーによって J2EE 標準に従った配備記述子が自動的に作成されます (通常、配備記述子は直接操作するのではなく、エンタープライズ Bean のプロパティシートで操作するため、IDE のエクスプローラウィンドウに記述子ファイルは表示されません。この記述子ファイルは、エクスプローラから開くことができます)。

EJB アプリケーションの実行時のワークフロー

実行時には、アプリケーションクライアントは最初にエンタープライズ Bean のホームインタフェースと通信し、次にリモートインタフェースと通信します。アプリケーションクライアントがエンタープライズ Bean オブジェクトと直接通信することはありません。クライアントのすべての処理は EJB コンテナを介して実行されます。

実行時のアプリケーションの構成要素の対話処理を図 1-4 に示します。図中の番号は後述の手順に対応しています。これはワークフローの概念図です。例では、リモート型インタフェースを持つ 1 つのエンタープライズ Bean だけが使われています。また、エンタープライズ Bean の種類によっては、一部の手順 (インスタンスプールへの格納など) が適用されない場合があります。

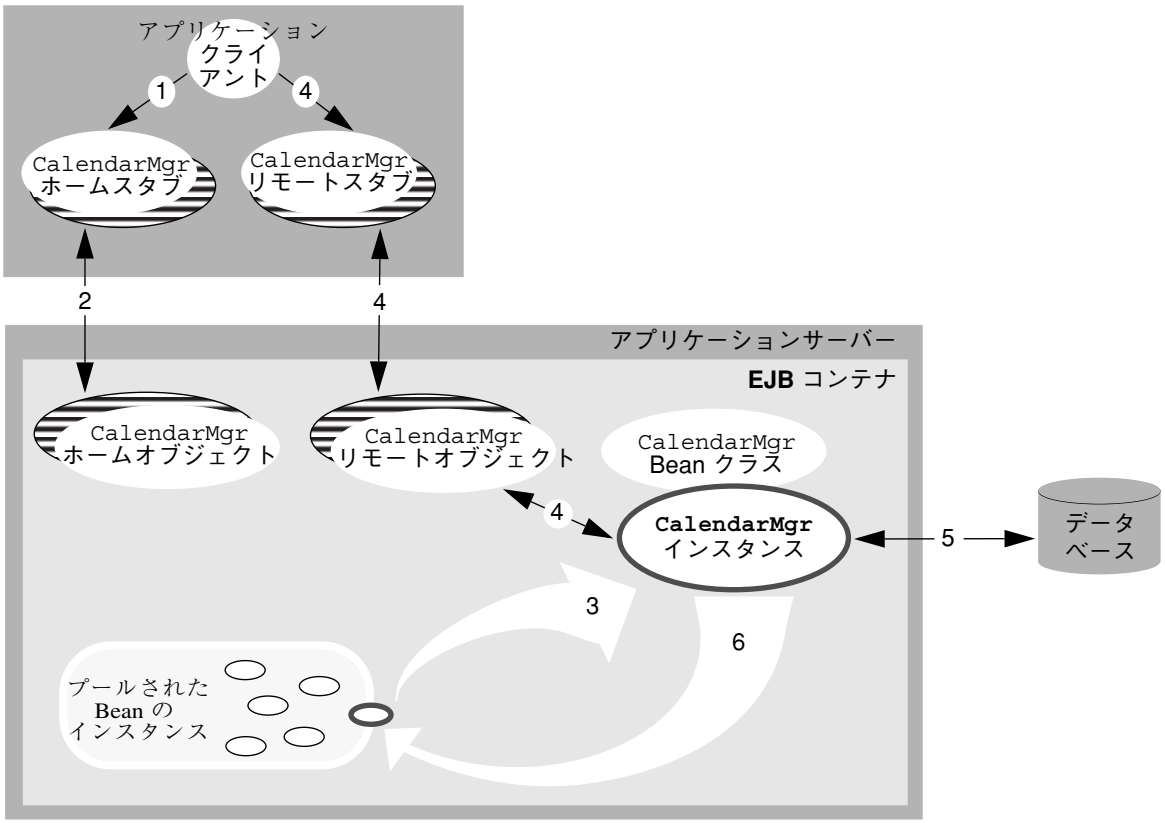


図 1-4　アプリケーションの実行時のワークフロー

1. クライアントはアプリケーションサーバーとコンテナに含まれているエンタープライズ Bean を検出します (すなわち、クライアントは JNDI 検索メソッドを使って、エンタープライズ Bean のホームインタフェースへのリモート参照を取得します)。それに対応するホームスタブがクライアントに作成されます。
2. サーバー側にホームオブジェクトが作成され、Bean のホームインタフェースが実装されます。ホームスタブは、(ファクトリとして機能する) Bean のホームオブジェクトに要求を送信し、このセッションでこのクライアントが使用するエンタープライズ Bean のインスタンスを作成してもらいます。
3. コンテナはプールから Bean インスタンスを取り出します。
4. サーバー側にリモートオブジェクトが作成され、Bean のリモートインタフェースが実装されます。クライアントはリモートスタブとリモートオブジェクトを介して、Bean インスタンスのビジネスメソッドを呼び出します。
5. データベースから Bean インスタンスにデータが読み取られ、クライアントに転送されます。更新はトランザクションによってデータベースに書き込まれます。
6. クライアントは要求した結果を受け取り、コンテナはインスタンスをプールに返します。

このアーキテクチャでは、マルチスレッドプログラミングを使わなくても、複数の同時ユーザーに対応することができます。エンタープライズ Bean のユーザーは自分自身の Bean インスタンスをプールから取得するため、EJB 開発者は単純なシングルスレッドコードを作成するだけで済みます。

エンタープライズ Bean の開発ライフサイクル

図 1-5 に示すように、エンタープライズ Bean は、EJB 開発者が作成してから使用できるようになるまでにいくつかの手順が必要です。

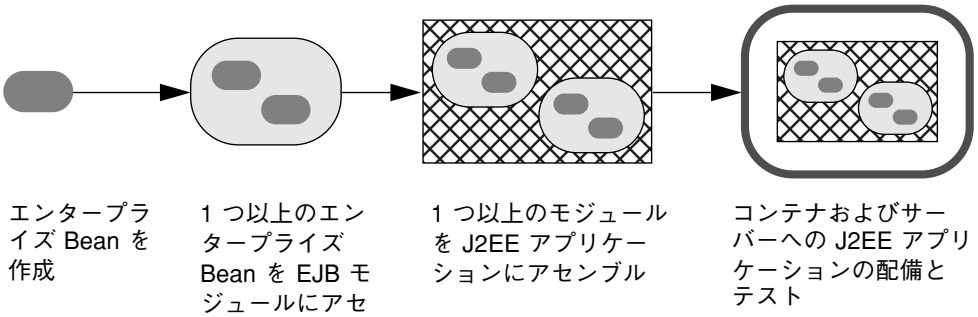


図 1-5　エンタープライズ Bean の開発、アセンブル、および配備

Sun ONE Studio IDE を使用する EJB 開発者は、次の手順に従ってエンタープライズ Bean を作成し、アセンブルと配備を行えるようにします。

1. 第 3 章から第 7 章までで述べる EJB ビルダーのウィザードとその他の GUI 機能を使用し、エンタープライズ Bean のクラスを生成します。
2. IDE のソースエディタと GUI 機能を使用し、エンタープライズ Bean のコードを記述します。セッション Bean のトランザクションとエンティティ Bean の持続性を EJB コンテナに管理させることにより、記述するコードはずっと少なくなります。
3. IDE を使用し、エンタープライズ Bean を関連するほかのエンタープライズ Bean とともに EJB モジュールにパッケージ化します。EJB のプロパティシートを使用し、Bean の外部依存性を配備記述子に追加します。
4. 第 9 章で述べる IDE の EJB テストアプリケーションを使用し、Bean 用の EJB Web テストクライアントを作成し、テストを実行します。この作業の準備として、ある程度のアセンブルと配備の処理が必要です。テスト後、Bean を実際の稼働環境にあるサーバーに配備します。

IDE のエンタープライズ Bean 機能

Sun ONE Studio IDE では、エンタープライズ Bean の開発に必要な多くの作業が自動化されます。IDE を使用した場合に自動的に実行される (ユーザーが行う必要のない) 作業を次に示します。

- 基本クラスのメソッドの宣言 - IDE は Bean に必要なクラスと、それらのクラスでのメソッドの宣言を生成します。

- 持続性を管理するコードの作成 - CMP Bean の作成時には、アプリケーションサーバーによって持続性が管理されます。
- トランザクションを管理するコードの作成 - CMP Bean の作成時、およびセッション Bean とメッセージ駆動型 Bean のトランザクションをコンテナに管理させる場合には、アプリケーションサーバーによってトランザクションが管理されます。
- Bean クラス、インタフェース、およびメソッドの同期処理 - IDE が整合性を維持します。
- 配備記述子の XML コードの作成 - このファイルは IDE が生成します。
- エンタープライズ Bean をテストするテストクライアントの作成 - セッション Bean とエンティティ Bean をテストする際には、IDE の提供する GUI ベースの総合的なテスト機能を使用できます。
- J2EE マニュアルの検索 - IDE が生成したエンタープライズ Bean のソースコードは J2EE 標準に準拠しています。生成されたコードには自動的にコメントと関連マニュアルへの参照が含まれます。また、次の機能も提供されます。
  - J2EE 標準に則ったアプリケーションプログラミングインタフェース (API) に対応したコード補完 - コードの編集中に Ctrl キーを押しながら Space キー (Solaris の場合は \ (バックスラッシュ) キー) を押します。
  - 適切な Javadoc™ マニュアルの簡単な表示 - クラスまたはインタフェースを選択して Shift キーを押しながら F1 キーを押します。

IDE を使用したエンタープライズ Bean の開発

EJB ビルダーのウィザードを使用して、エンタープライズ Bean のインフラストラクチャを作成します。このウィザードは、ユーザーが選択した Bean の種類 (セッション Bean、エンティティ Bean、またはメッセージ駆動型 Bean) に合わせて調節され、Bean の持続性データのソースや Bean のトランザクションや持続性の管理に関するオプションを提供します。このウィザードの指示に従うことで、基本的なコンポーネントすべてを作成できます。

図 1-6 に、IDE が生成する一般的なエンタープライズ Bean の要素と、これらの要素のエクスプローラウィンドウでの表示例を示します。この図ではリモート型インタフェースを持つセッション Bean が例として使われています。

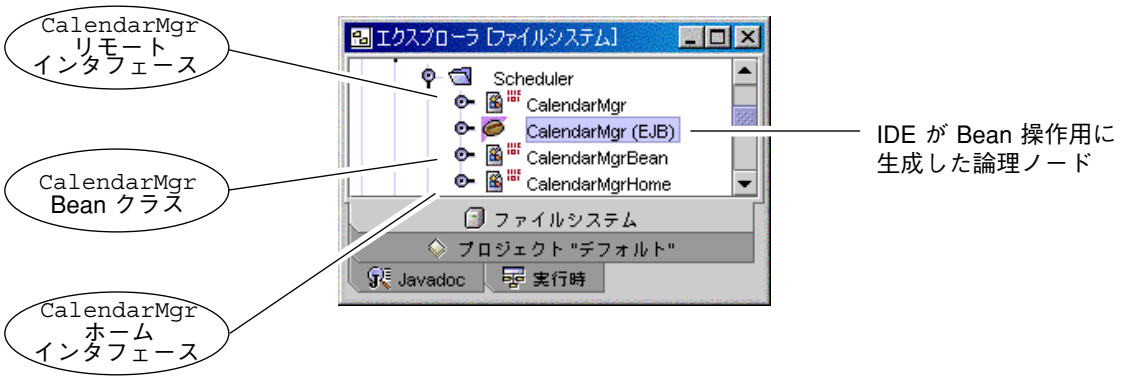


図 1-6　生成されたエンタープライズ Bean 要素のエクスプローラウィンドウでの表示

ウィザードを使ってこれらの基本要素を生成し終わったら、EJB ビルダーのほかの GUI 機能を使用して Bean にメソッドを追加し、ソースエディタを使用して Bean のコーディングを完成させます。

関係 CMP エンティティ Bean セットの生成

EJB ビルダーのウィザードを使用すると、CMP エンティティ Bean のセット全体のインフラストラクチャと、それを格納する EJB モジュールを一度に生成することができます。この機能は、Bean が表しているデータベース表が外部キーに関連している場合に特に便利です。エンティティ Bean のセットでは、これらの外部キーはコンテナ管理による関係として保持されます。

トランザクション機能

エンタープライズ Bean モデルでは、トランザクション処理は暗黙的にも、明示的にも取り扱うことができます。Bean インスタンスのメソッドが呼び出されると、EJB コンテナが Bean 開発者に代わってトランザクションを管理します。Bean 開発者にはトランザクションを記述する専門知識は必要ありません。すなわち、トランザクションのコンテキストを制御するコードを作成し、デバッグする必要はありません。EJB ビルダーのウィザードで簡単な選択を行うだけで、Bean のトランザクション属性を宣言することができます。これらの属性は、Bean の EJB モジュールのプロパティシートを使用して、後から修正することができます。

ただし、セッション Bean でトランザクションの明示的なプログラミングが必要な場合もあります。IDE では、コンテナを明示的に上書きし、JDBC API と JTA (Java Transaction API) を使用して、Bean のトランザクション処理を管理させることができます。

持続性機能

トランザクションと同様に、IDE では EJB コンテナに Bean の持続性を完全に処理させることも、持続性を EJB 開発者自身がコーディングすることもできます。EJB 開発者自身が持続性を処理する場合は、JDBC コードを記述します。コンテナ管理による持続性を使用したい場合は、最初に EJB ビルダーのウィザードで必要な内容を選択し、次にコンテナが保存データを特定の言語形式で検出できるような宣言をプロパティシートに記述します。

セキュリティ機能

エンタープライズ Bean の特定のメソッドを、特定のロールを持ったユーザーだけが呼び出せるようにするには、プログラムによるセキュリティを Bean に追加します。このために Bean のソースコードで完全なセキュリティルーチンを記述する必要はありません。Bean のコードのセキュリティへの参照を、メソッドで宣言したセキュリティロールに一致させるだけです。これを一致させるには、Bean のプロパティシートでフィールドを修正し、Bean の配備記述子にセキュリティ情報を追加するだけです。

クライアントがセキュリティ保護された Bean メソッドを呼び出そうとすると、EJB コンテナがユーザーのロールをアクセス制御リスト (どのロールのユーザーに Bean メソッドの実行権限が与えられているかを記録したリスト) と比較し、実行を許可するか、拒否するかを決定します。

アプリケーションクライアントの作成

作成したアプリケーションの EJB 層を構成するエンタープライズ Bean の開発に加え、アプリケーションクライアントも IDE を使用して作成できます。この場合、アプリケーションクライアントとは、自分自身の main メソッドで開始し、J2EE クライアントコンテナ中で実行され、EJB モジュールをはじめとする他の J2EE アプリケーションコンポーネントと対話的に動作する独立した Java プログラムを指します。クライアントの設計と開発の詳細については、マニュアル『J2EE アプリケーションのプログラミング』を参照してください。

詳細情報の参照先

エンタープライズ Bean と EJB 層の設計についての詳細は、次の Web サイトにある「Enterprise JavaBeans Specification 2.0」を参照してください。

```
http://java.sun.com/products/ejb/docs.html
```

その他の情報源については、xviii ページの「お読みになる前に」を参照してください。

第2章

設計とプログラミング

エンタープライズ Bean の設計とプログラミングに精通していない場合は、まず各種の Bean の違いと使用目的を考慮する必要があります。それぞれの Bean のライフサイクル、メソッドと例外の適用方法、および別のアプリケーション環境で Bean を再利用するための設定方法についての知識も必要です。また、持続性、トランザクション、およびセキュリティの取り扱い方法を理解する必要もあります。この章では、これらの内容について説明し、最後に詳細情報についての文献リストを掲載します。

必要な Bean の種類の決定

「Enterprise JavaBeans 仕様 2.0」では、セッション Bean、エンティティ Bean、およびメッセージ駆動型 Bean という 3 種類のエンタープライズ Bean が定義されています。さらに、セッション Bean とエンティティ Bean にもいくつかの種類があり、それぞれに目的に応じた機能が組み込まれています。Sun ONE Studio IDE の EJB ビルダーを使用すると、これらのエンタープライズ Bean を画面上の指示に従って効率的に作成できます。

この章では、設計作業の参考情報として、エンタープライズ Bean の種類について説明します。

図 2-1 に、IDE のテンプレートを使用してエンタープライズ Bean を作成する前に決定する基本的な項目を示します。

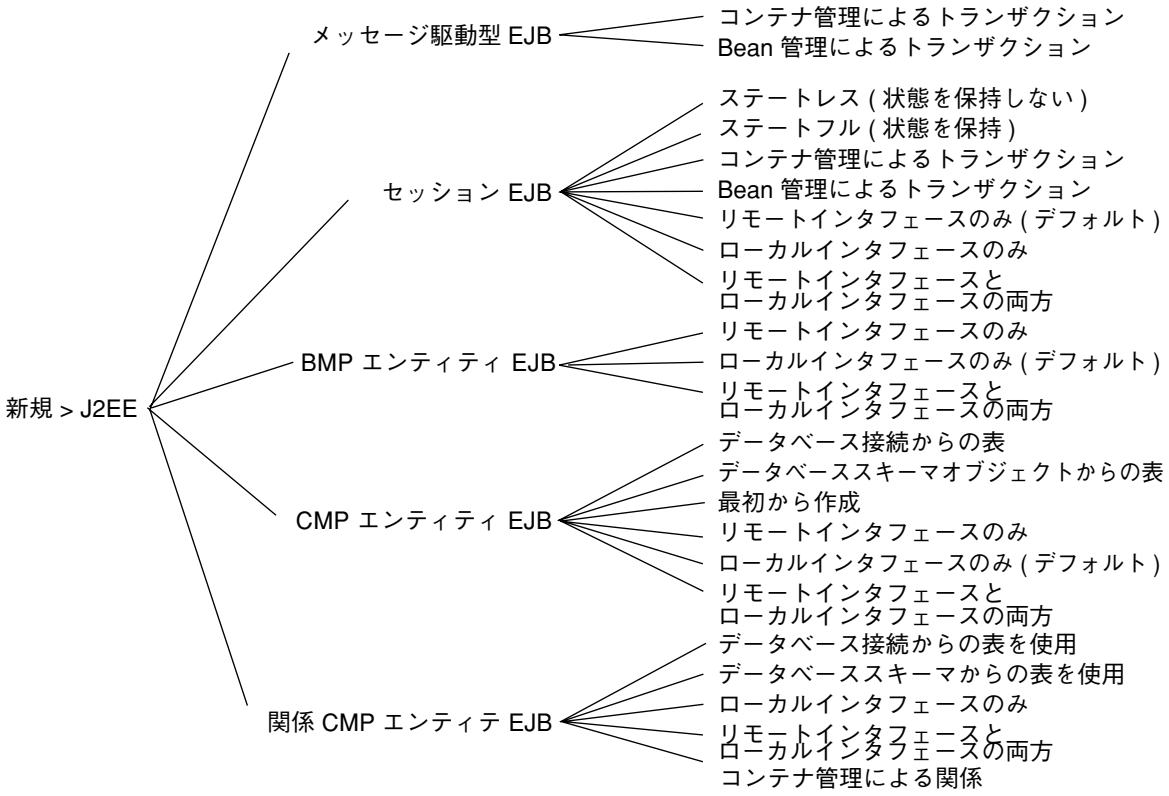


図 2-1　Sun ONE Studio IDE でのエンタープライズ Bean の基本的な選択項目

セッション Bean

セッション Bean は、アプリケーションのトラフィックを管理するプログラムとして機能し、アプリケーションのワークフローを制御し、ビジネスプロセスをカプセル化します。MVC (モデル・表示・制御) アーキテクチャに当てはめて考えると、セッション Bean は制御層に相当します (ただし、セッション Bean は EJB アプリケーションに含まれています)。セッション Bean は、クライアントに代わってデータベースへのアクセスやバランス演算などの処理を行うことができます。セッション Bean では、データベースのデータは直接表現されませんが、セッション Bean からエンティティ Bean を操作し、データベースにアクセスすることができます。

エンタープライズ Bean を使用するアプリケーションのコンテキストでは、セッション Bean は EJB コンテナによって管理され、クライアントと EJB サーバー側のアプリケーションコンポーネントとの通信を管理します。これらのセッション Bean 以外の

アプリケーションコンポーネントには、しばしばエンティティ Bean が含まれ、エンティティ Bean 独自の層に、持続性に対応した Bean からアクセス可能なデータベースが含まれています。

セッション Bean は、1 つ以上のエンティティ Bean を操作し、エンティティ Bean 間の情報のやり取りを管理し、エンティティ Bean によって表現されたデータと、そのデータに適用されるビジネスロジックとの橋渡しを行います。1 つのセッション Bean から、同じアプリケーションに含まれている複数のエンティティ Bean のトランザクション処理を管理できます。

セッション Bean が管理する通信 (セッション) や、セッション Bean の内部のデータは一時的なものです。クライアント・サーバーセッションが終了したり、クライアント、またはサーバーが停止すると、クライアントがそのセッション用に作成したセッション Bean のインスタンスが破棄されます。ただし、クライアントはセッションのハンドルを保存し、停止後に同じセッションを再開することができます。

セッション Bean に主キーはありません。エンティティ Bean と異なり、セッション Bean は 1 度に 1 つのクライアントからのみ使用されるため、クライアントに対してセッション Bean を匿名にすることができます。そのため、セッション Bean には、主キーが提供する一意の識別子は必要ありません。

オンラインショッピングアプリケーションの `ShoppingCart` オブジェクトのように、セッション Bean でエンティティが表現される場合もありますが、ほとんどのセッション Bean は、エンティティの状態をデータベースに保存することを想定していません。たとえば、ユーザーがオンラインショッピングを行なっている間、`ShoppingCart` Bean インスタンスはユーザーがショッピングカートに入れた商品を一時的に保持します。ユーザーによる商品の購入処理がコミットされる前にサーバーが停止した場合は、これらの商品をそのトランザクション内でデータベースに保存するのは適切ではありません。これらのデータを破棄し、次のセッションでユーザーに新しいショッピングカートを使用させるのが一般的な設計手法です。

ステートレスセッション Bean の使用

クライアントとセッション Bean との対話処理は、単一のメソッドにパラメータを渡すような短くて単純なこともあります。また、多数のメソッドと複数のデータベーストランザクションにわたる複雑で長い通信をセッション Bean で管理することもできます。その場合、セッション Bean は複数のメソッド呼び出しにわたって情報を保持する必要があります。

最初の状況、すなわち 1 回の要求と 1 回の応答で構成されるセッションには、ステートレスセッション Bean が最適です。ステートレスセッション Bean は、メソッド呼び出しの間の状態を保持しません。このような軽量の Bean は、アプリケーションのリソースをほとんど消費せず、コンテナによる管理が簡単で、高速な処理が可能です。また、多数のクライアントを持つアプリケーションでは、スケーラビリティも高まります。

その代わり、ステートレスセッション Bean には、データ操作の自由度が制約されるという弱点もあります。ステートレスセッション Bean は、クライアントから渡されたパラメータしか操作できません。それぞれのメソッドの呼び出しは、それ以前のメソッドの呼び出しとは無関係です。

たとえば、ステートレスセッション Bean で郵便番号を検索する場合を考えてみます。それぞれの検索処理は、`getZip` というメソッドを 1 回呼び出すだけで完了できます。これは、検索を処理するのに必要な情報が、このメソッドのパラメータにすべて含まれているからです。すべてのトランザクションが、呼び出されたメソッドの内部で、そしてコンテナの内部で実行されます (トランザクションについては、この章の後続の節と第 3 章を参照してください)。

ステートレスセッション Bean のインスタンス変数に、メソッドを実行している間だけ状態を格納することができます。ステートレスセッション Bean のインスタンスは、プールに格納されている間はすべて同じです。したがって、EJB コンテナは、これらの Bean インスタンスを自由に割り当て、クライアントからメソッドが呼び出されるたびに、使用するインスタンスを入れ換えることができます。この方法で、複数のクライアントの間でステートレスセッション Bean を共有できます。それぞれの Bean は、クライアントからは匿名に見えます。

ステートレスセッション Bean を使用できるのは、別々のクライアントがセッション Bean を順番に使用し、特定のクライアントに合わせてセッション Bean を調節する必要がない場合です。ステートレスセッション Bean は、特定のクライアントの状態情報を保持しません。ただし、クライアントに固有ではない状態、たとえば開かれているデータベース接続を保持することは可能です。

ステートフルセッション Bean の使用

セッション Bean は、クライアントとの間で複雑な通信を行うこともできます。このような Bean は、複数のメソッドを使用してビジネスロジックをカプセル化し、複数のメソッドの呼び出しにわたって状態を保持する必要があります。このような Bean

が、ステートフルセッション Bean です。クライアントが対話型のアプリケーションの場合や、作成時にセッション Bean の状態を初期化する必要がある場合は、ステートフルセッション Bean を使用してください。

セッション Bean の状態は必要に応じてデータベースに書き込むことができます。状態はクライアントに固有で、セッションが終了するまではメモリーに保持されますが、持続的ではありません。ステートフルセッション Bean をメモリーから削除する必要がある場合は、EJB コンテナが状態を管理します。Bean のインスタンスの状態はセッション中は保持されますが、クライアントが終了したり、サーバーが停止したりした場合は保持されません。

コンテナのクラッシュ、インスタンス非活性化中のタイムアウト、またはメソッドがスローしたシステム例外の発生後は、`ejbRemove` メソッドは呼び出されません。こういった状態をリセットするためのプログラムの用意が必要なこともあります。

ステートフルセッション Bean は、複数のクライアントの間で共有されません。この Bean はただ 1 つのクライアントのために機能し、セッション全体にわたって、クライアントとの通信状態を保持します。ステートフルセッション Bean のインスタンスはプールには格納されません。

前述のオンラインショッピングカートは、ステートフルセッション Bean の使用例の 1 つです。ショッピングの論理的なビジネストランザクションに、ユーザーの複数の意志決定が含まれているのと同様に、このアプリケーションのセッション Bean には複数のメソッド呼び出しが含まれています。`ShoppingCart` Bean は、ユーザーが選択した商品を蓄積し、ユーザーが購入商品のリストを確認した時点で、商品ごとに購入を受け付けるか拒否するかを決定し、注文を確定する必要があります。

トランザクションモードの選択

ステートフルセッション Bean とステートレスセッション Bean のどちらをプログラミングする場合も、EJB ビルダーのウィザードで次のいずれかの項目を選択する必要があります。

- **コンテナ管理によるトランザクション。**Bean のトランザクションを EJB コンテナに管理させます。トランザクションを管理するコードを記述する必要はありません。この Bean を CMT (Container-Managed Transaction) セッション Bean といいます。

- **Bean 管理によるトランザクション**。Bean のトランザクションを Bean 自身に管理させます。Bean のメソッドをコーディングし、それぞれのトランザクションを明示的に指定する必要があります。この Bean を BMT (Bean-Managed Transaction) セッション Bean といいます。

CMT セッション Bean では、コーディング作業が簡単になり、すべてのトランザクションが予測可能で整合性のとれた方法で処理されます。また、作成した Bean に設定したトランザクションポリシーは、宣言的に変更することができます。ただし、それぞれのメソッドは 1 つのトランザクションしか処理できません。通常は、コンテナによって、メソッドが開始される直前にトランザクションが開始され、メソッドが終了する直前にトランザクションがコミットされます。1 つのメソッドで、入れ子になったトランザクションや複数のトランザクションを処理することはできません。

トランザクション属性の割り当て

Bean のトランザクションを EJB コンテナに管理させる場合は、コンテナは Bean や Bean 内の特定のメソッドのトランザクション属性を参照します。トランザクション属性とは、トランザクションのスコープ (トランザクションにどのメソッドが含まれ、これらのメソッドの結果をトランザクションごとにどのように取り扱うか) を指定したものです。これらの属性は次のように割り当てます。

- **CMT セッション Bean** - IDE が CMT セッション Bean の必須トランザクション属性を自動的に割り当て、これらのトランザクション属性が Bean 内のすべてのビジネスメソッドに適用されます。ただし、特定のメソッドのトランザクション属性を手作業で割り当てたり、Bean の優先トランザクション属性を設定したりすることもできます (CMT セッション Bean のトランザクション属性は EJB モジュールレベルで設定します)。
- **エンティティ Bean** - CMT セッション Bean の場合と同じです。エンティティ Bean は、すべてコンテナ管理によるトランザクションを使用します。

BMT セッション Bean のトランザクション属性は設定できません。BMT セッション Bean のトランザクションの範囲は、Bean クラスで明示的に指定する必要があります。

JTA や JDBC の使用

Bean 管理によるトランザクションを明示的にコーディングする場合は、Java Transaction API (`javax.transaction.UserTransaction` インタフェース、すなわち JTA) か JDBC API を使用できます。

- JTA - Sun ONE Studio IDE を使用して新しい BMT セッション Bean を作成する場合は、JTA を使用してください。JDBC API よりも高機能で、柔軟性にすぐれています。
- JDBC API - 従来の JDBC テクノロジを使用したコードや、SQL コードをカプセル化したコードをセッション Bean に組み込む場合は、JDBC API を使用してください。

JTA には、JDBC API といったほかのリソース用のトランザクションを含めることができます。JTA を使用してエンタープライズ Bean のトランザクションをコーディングする場合は、データベース接続には JDBC API を、トランザクションには JTA を使用します。

トランザクション処理では、Bean のメソッドから JTA メソッドを呼び出します。呼び出された JTA メソッドは、JTS (Java Transaction Service、J2EE が使用するトランザクションマネージャ) の下位ルーチンを呼び出します。このような間接的な呼び出しにより、JTA ではトランザクションマネージャの実装とは無関係に、トランザクションを指定す定することができます。1 つの JTA トランザクションで、別々のベンダーの複数のデータベースを更新することもできます。

JDBC トランザクションは、使用しているデータベースのトランザクションマネージャによって管理されます。

JTA では、入れ子になったトランザクションを処理できません。トランザクションを終了しないと、別のトランザクションを開始できません。

トランザクションに関する詳細は、マニュアル『J2EE アプリケーションのプログラミング』を参照してください。

セッション Bean のライフサイクル

実行時には、アプリケーションサーバーが EJB クライアントからの要求に応じて Bean インスタンスを作成します。作成された Bean インスタンスは、EJB コンテナによって管理される複数の処理段階で使用されます。インスタンスは不要になった時点で破棄されます。

ここからは、セッション Bean のライフサイクル段階、セッション Bean を次のライフサイクル段階へ移行させるメソッド、プログラマが行う必要のある作業を説明します。

Bean インスタンスの作成と初期化

セッション Bean の実行時のライフサイクルは、EJB クライアントが Bean に何らかの処理を要求したときに始まります。このライフサイクル段階は次のように進行します。

クライアントが Bean のホーム (またはローカルホーム) インタフェースの生成メソッドを呼び出します。それに応じて、コンテナは次の 3 つのメソッドを順番に呼び出します。

1. `newInstance` メソッドを呼び出して、セッション Bean の新しいインスタンスを作成します。
2. `setSessionContext` メソッドを呼び出して、作成したインスタンスをセッションコンテキストオブジェクトに関連付けます。
3. `ejbCreate` メソッドを呼び出して、このインスタンスを初期化します。

注 - IDE は `setSessionContext` メソッドと `ejbCreate` メソッドのシグニチャを生成します。プログラマは、これらのメソッドのコードを完成させる必要があります。

クライアントは、Bean インスタンスのリモートオブジェクトの参照を受け取ります。

ビジネスロジックの実行

Bean インスタンスが作成され、初期化されると、EJB クライアントはこのインスタンスに処理を行わせます。このライフサイクル段階は次のように進行します。

クライアントが Bean のリモートオブジェクトのビジネスメソッドを呼び出します。それに応じて、コンテナは次の処理を行います。

- セキュリティ認可をチェックし、クライアントにビジネスメソッドを実行する権限が与えられているかどうかを確認します。
- メソッドのトランザクション属性で指定されたトランザクション制御を適用します。
- インスタンスのビジネスメソッドを呼び出します。

クライアントはビジネスメソッドの結果を受け取ります。

注 - プログラマは、セキュリティ制御を Bean のコードの中でプログラム的に指定することも、EJB モジュールのプロパティインスペクタを使用して宣言的に指定することもできます。プログラマは、EJB モジュールのプロパティシートを使用して、Bean のメソッドのトランザクション属性を設定します。

Bean インスタンスの削除

クライアントはセッションを完了したときに、Bean インスタンスを破棄することができます。このライフサイクル段階は次のように進行します。

クライアントがホームインタフェース (またはローカルホームインタフェース) かリモートインタフェース (またはローカルインタフェース) の `remove` メソッドを呼び出します。それに応じて、コンテナは `ejbRemove` メソッドを呼び出し、インスタンスで使用されていたリソースをすべて閉じます。その後で、コンテナはメモリーからインスタンスを削除します。

注 - IDE は `ejbRemove` メソッドのシグニチャを生成します。プログラマは、このメソッドのコードを完成させる必要があります。

ステートレスインスタンスのプールへの格納

通常の本稼働環境では、多数のクライアントが同じエンタープライズ Bean に同時に処理を要求します。この状況に対処するため、コンテナはステートレスセッション Bean の複数のインスタンスを同時に作成し、後で使用できるようにプールに格納しておくことができます。コンテナは、自分自身の判断でプールにインスタンスを格納することができます。

ステートレスセッション Bean は、クライアントに関連する状態情報を複数のメソッドの呼び出しにわたって保持しません。そのため、プールに格納されたインスタンスは相互に交換できます。コンテナは、同じクライアントからの複数の要求を処理するために、プールから別々のセッション Bean を呼び出すことができます。

コンテナは、クライアントから大量かつ頻繁に要求があっても、ステートレスセッション Bean のインスタンスが不足しないように、プールの中のインスタンス数を絶えず調節します。たとえば、クライアントからの要求の数が増加すると、ステートレスセッション Bean のインスタンスを新しく作成し、メモリーが足りなくなると、こ

れらのインスタンスを削除します。コンテナは、ステートレスセッション Bean の ejbCreate メソッドと ejbRemove メソッドを自分自身の判断で呼び出して、プールを管理します。

ステートフルインスタンスの非活性化

ステートフルセッション Bean は、セッション全体にわたって、クライアントとの通信状態を保持する必要があります。そのため、EJB コンテナはこれらの Bean のインスタンスをプールには格納しません。これらのインスタンスは、クライアントから明示的に指示されたときだけ、作成または削除されます。

その場合でも、リソースの使用量を制御するには、ステートフルセッション Bean が一定時間保持するアクティブなインスタンス数を、コンテナによって管理させる必要があります。コンテナは、メモリーが足りなくなったときにインスタンスを非活性化し、その通信状態を二次記憶領域に待避させることができます。これにより、ほかのクライアントのセッションを処理できるようになります。この処理では、コンテナはまずインスタンスの ejbPassivate メソッドを呼び出し、プログラマがこのメソッドに記述したコードに従ってリソースを解放し、すべてのフィールドを直列可能な状態にします。コンテナは、その後でインスタンスの一時的ではないフィールドを二次記憶域に書き出します。

クライアントが非活性化インスタンスのビジネスメソッドを呼び出すと、コンテナは該当するインスタンスの状態を二次記憶域から復元し、そのインスタンスの ejbActivate メソッドを呼び出します。プログラマがこのメソッドに記述したコードに従って、ejbPassivate メソッドで解放されたリソースを取得し、直列可能ではなかったフィールドの値を復元します。

注 - IDE は、ステートフルセッション Bean とステートレスセッション Bean のどちらについても、ejbPassivate メソッドと ejbActivate メソッドのシグニチャを生成します。プログラマは、これらのメソッドのコードを完成させる必要があります。

セッションでの状態の同期化

プログラマは、ステートフル CMT セッション Bean にセッション同期化インタフェースを実装できます。コンテナは、ステートフル Bean のライフサイクルの間に、またトランザクションの特定の時点でこのインタフェースを使用し、Bean のインスタンスにトランザクションが開始または終了されようとしていることを通知します。プログ

ラマは、このインタフェースのメソッドをプログラミングし、Bean のインスタンス変数をデータストアの最新のデータに同期させたり、トランザクションを中止させたりすることができます。このインタフェースには、`afterBegin`、`beforeCompletion`、および `afterCompletion` の 3 つのメソッドが含まれています。

注 - IDE はセッション同期化メソッドのシグニチャを生成します。プログラマは、これらのメソッドのコードを完成させる必要があります。

エンティティ Bean

エンティティ Bean はデータストア中の持続データを表現します。この種類の Bean は、データベース表の行といったデータセットのオブジェクトビューを提供します。エンティティ Bean のそれぞれのインスタンスには、データのエンティティが 1 つずつ含まれています (データのエンティティに加えて、そのエンティティに固有のビジネスロジックが含まれる場合もあります)。クライアントとして動作する、またはクライアントの処理を代行するセッション Bean は、エンティティ Bean を使用してデータベースからデータを検索したり、データベースにデータを挿入したりできます。

エンティティ Bean の状態は環境に依存しません。エンティティ Bean は主キーとリモート参照が持っているため、サーバー、EJB コンテナ、またはクライアントに障害が発生しても、消失することはありません。エンティティの状態は、最後にコミットされたトランザクションの直後の状態に自動的に復元されます。

それぞれのクライアントは、エンティティ Bean の別々のインスタンスを取得するため、複数のユーザーが同じデータセットへアクセスすることができます。2 つのクライアントがエンティティ Bean の同じ検索メソッドを実行した場合は、両方のクライアントが同じリモートオブジェクトを参照します。それぞれの検索は互いに独立しているため、競合の問題は発生しません。そのため、エンタープライズ Bean では、マルチスレッドに対応したコードを使用する必要はありません (ただし、状況によっては並行プロセスの実行が必要なこともあります。メッセージ駆動型 Bean を使用すると、J2EE アプリケーションでのマルチスレッドをほぼ実現できます。この方法については 42 ページの「メッセージ駆動型 Bean」を参照してください)。

クライアントは、固有のオブジェクト識別子 (Bean の主キー) に基づいて特定のエンティティ Bean を検出します。

EJB コンテナのサービスの利用

エンティティ Bean のすべてのトランザクションは、EJB コンテナによって自動的に管理されます。プログラマは、エンティティ Bean のコードを作成し、EJB モジュールを作成した後で、EJB モジュールのプロパティシートを使用して Bean のトランザクション属性を宣言します。コンテナは、宣言されたトランザクション属性に従って、Bean のトランザクションの範囲を識別します。IDE は、エンティティ Bean のビジネスメソッド、生成メソッド、削除メソッド、検索メソッド、選択メソッド、およびホームメソッドのすべてに、デフォルトのトランザクション属性を自動的に割り当てます。

EJB プログラマは、エンティティ Bean の持続性をコンテナに管理させることも、コードを記述して、エンティティ Bean 自身にデータストアとの関係を管理させることもできます。

IDE を使用して、コンテナ管理による持続性を使用するエンティティ Bean (CMP エンティティ Bean) を作成する場合は、Bean クラスにデータストアに対する JDBC 呼び出しを記述する必要はありません。Bean のインスタンス変数をデータストアに同期させるコードは、コンテナが提供します。プログラマは、インスタンス変数をデータベース表の列にマップするための情報をコンテナに提供するだけで済みます。また、サーバーに Bean の照会メソッドの実装させる方法は、EJB 照会言語 (EJB QL) で定義できます。

検索メソッド中の EJB QL による照会は、クライアントが既存のエンティティオブジェクトを選択する際に使用できます。また、クライアントに結果を直接公開させないで、オブジェクトやエンティティ Bean の状態に関連する値を選択し、その値をクライアントに戻すようにする場合にも、オブジェクトの EJB QL による照会を利用できます。このように情報を検索したい場合、EJB QL の照会処理では Bean の抽象持続性スキーマを使用することができます。抽象持続性スキーマは、Bean の配備記述子の一部で、Bean の持続フィールドと関係を定義するものです。

J2EE リファレンス実装 (RI) サーバーにアプリケーションを配備する例を考えてみます。サーバーは、 Bean 中のメソッドと指定された EJB QL 照会を見つけます。サーバーは、この EJB QL 照会から、この照会をマッピングするためにサーバー固有の SQL 文を生成します。

場合によっては、サーバープラグインが自身で利用するために生成した SQL 文を部分的に変更する必要があります。これは、そのサーバーが自身で作成した SQL へのアクセスを許可していることを前提としています。たとえば、J2EE リファレンス実装 (RI) サーバーを使用していて、アプリケーションに EJB 1.1 環境で作成された CMP エンティティ Bean が含まれている場合、現在の環境で正しく動作させるために、サー

バーが生成したあるメソッドに対する SQL を変更する必要があります。また、生成した SQL を書き換えると、CMT エンティティ Bean のマッピング規則も変更できます。

エンティティ Bean 間での関係は EJB コンテナによって管理できます。外部キーを使用しているデータベースから、複数の関係 CMP エンティティ Bean を含んだセットを生成すると、Bean 間の関係は IDE によって自動的に保持されます。

コンテナ管理による持続性を使用すると、コーディング作業が簡単になり、特定のデータストアに依存しないエンティティ Bean を作成できるようになります。

それと同時に、マッピングツールによってサポートされない旧式のコードをラップするための EJB アプリケーションの作成が必要なこともあります。また、複数の表の連結や、場合によっては異なるデータベースの連結 (リレーショナルでないデータベースなど) が必要なこともあります。こういった場合、EJB アプリケーションを配備するアプリケーションサーバーの能力に応じて、持続性を Bean に管理させ、EJB プログラマがすべてのデータベース呼び出しをエンティティ Bean に手作業で記述する必要が出てくることもあります。利用するサーバーが、処理に必要な持続性をサポートしている場合は、コンテナに持続性を管理させる方法が最良です。ただし、エンティティの状態管理の柔軟性の点では、Bean が持続性を管理した方が一般的に優れています。

エンティティ Bean のライフサイクル

アプリケーションサーバーは、EJB クライアントが使用するエンティティ Bean インスタンスのプールを作成します。実行時には、EJB コンテナによって管理される複数の処理段階で、これらのインスタンスがクライアントからの要求に応じて使用されます。これらのインスタンスは、不要になった時点で破棄されます。

ここからは、エンティティ Bean のライフサイクル段階、エンティティ Bean を次のライフサイクル段階へと移行させるメソッド、プログラマが行う必要のある作業を説明します。

Bean インスタンスのプールの作成と管理

エンティティ Bean の実行時のライフサイクルは、コンテナが Bean のインスタンスを作成し、プールに格納したときに始まります。

多数の EJB クライアントが多数のエンティティ Bean を同時に使用し、処理を行わせる場合があります。コンテナは、自分自身の判断で Bean の複数の匿名インスタンスを前もって作成し、プールに格納しておくことができます。これらのインスタンスを

検索メソッドによる照会の実行に使用したり、これらのインスタンスに識別情報を割り当てたりすることができます。データストアのデータを格納するためにインスタンスが必要になると、コンテナはプールに格納されているインスタンスを使用可能状態にします (使用可能状態のインスタンスには、そのインスタンスを一意に識別する主キーが割り当てられます)。コンテナは、新しいインスタンスを作成したり、不要になったインスタンスを削除したりして、プールのサイズを調節できます。

コンテナは、次のメソッドを呼び出して、プールに新しいインスタンスを作成します。

1. `newInstance` メソッドを呼び出して、エンティティ Bean の新しいインスタンスを作成します。
2. `setEntityContext` メソッドを呼び出して、作成したインスタンスをエンティティコンテキストオブジェクトに関連付けます。

これで、インスタンスがプール状態になります。

コンテナは、インスタンスをプール状態から使用可能状態へと、さらに使用可能状態からプール状態へと切り替えます。クライアントが識別情報を使用してエンティティを要求し、使用可能状態のインスタンスの中に、その識別情報に対応するものがない場合は、コンテナはインスタンスをプール状態から使用可能状態に切り替えます。その際に、コンテナは該当するインスタンスの `ejbActivate` メソッドを呼び出します。プログラマがこのメソッドにコードを記述して、(プール状態のインスタンスではなく) 識別情報が割り当てられたインスタンスに必要なリソースを取得できます。コンテナは、`ejbActivate` メソッドを呼び出した後で、エンティティのインスタンス変数に値を格納し、そのインスタンスをリモートオブジェクトに関連付けます。

これで、インスタンスが使用可能状態になります。

`ejbActivate` メソッドでは、エンティティのインスタンス変数に値を読み込まないことに注意してください。インスタンス変数への値の読み込みは、BMP エンティティ Bean では `ejbLoad` メソッドで処理され、CMP エンティティ Bean ではコンテナ自身によって処理されます。

使用可能状態のインスタンスが増えすぎた場合は、コンテナはそれらのインスタンスを非活性化し、プール状態に戻すことができます。その際に、コンテナは該当するインスタンスの `ejbPassivate` メソッドを呼び出し、プログラマがこのメソッドにコードを記述して、プール状態のインスタンスに不要なリソースを解放できます。さらに、コンテナはこのインスタンスをリモートオブジェクトから切り離し、エンティティのインスタンス変数の現在値をデータベースに保存します。

この場合も、ejbPassivate メソッドでは、エンティティのインスタンス変数値をデータベースに保存しないことに注意してください。インスタンス変数値のデータベースへの保存は、BMP エンティティ Bean では ejbStore メソッドで処理され、CMP エンティティ Bean ではコンテナ自身によって処理されます。

非活性化インスタンスをプールから取り除く場合は、コンテナは該当するインスタンスの unsetEntityContext メソッドを呼び出し、そのインスタンスをエンティティコンテキストオブジェクトから切り離します。コンテナは、その後でこのインスタンスを破棄します。

注 - IDE は、setEntityContext メソッド、unsetEntityContext メソッド、ejbActivate メソッド、ejbPassivate メソッドのシグニチャを生成します。プログラマは、特定のエンティティのコンテキストやリソースが必要な場合に、これらのメソッドを完成させる必要があります。

Bean インスタンスを使用した新しいエンティティの作成

(データをデータストアに挿入するために) 新しいエンティティを作成する必要が生じると、EJB クライアントは Bean のホームインタフェースの生成メソッドを呼び出します。それに応じて、コンテナは次の処理を行います。

1. セキュリティチェックを実行し、生成メソッドのトランザクション属性で指定されたトランザクション制御を適用します。

2. プール内の Bean インスタンスの ejbCreate メソッドを呼び出します。CMP エンティティ Bean では、このメソッドは持続フィールドの値を初期化し、コンテナがデータストアにデータを挿入できるようにします。BMP エンティティ Bean では、このメソッドはフィールドの値を初期化し、レコードをデータベースに挿入します。

3. Bean のリモートオブジェクトを作成し、新しい Bean インスタンスに関連付けます。

4. Bean インスタンスの ejbPostCreate メソッドを呼び出して、初期化を完了します。この Bean インスタンスには、コンテナによって識別情報がすでに割り当てられています。そのため、ejbPostCreate メソッドでは、関連付けられたリモート (またはローカル) インタフェース、主キーといった識別情報を、ほかのエンタープライズ Bean に渡すことができます。

クライアントは、インスタンスのリモートオブジェクトの参照を受け取ります。このインスタンスは使用可能状態になり、ビジネスメソッドを実行できるようになります。30 ページの「ビジネスロジックの実行」を参照してください。

注 - IDE は `ejbCreate` メソッドと `ejbPostCreate` メソッドのシグニチャを生成します。プログラマは、これらのメソッドを完成させる必要があります。さらに、コンテナによって適用されるセキュリティ制御とトランザクション属性も指定する必要があります。

既存の Bean インスタンスの検出

EJB クライアントは、Bean インスタンスのホームオブジェクトの検索メソッドを呼び出して、既存のエンティティを検出することができます。検索メソッドは、特定の検索条件に適合するエンティティをすべて返します。`findByPrimaryKey` メソッドに加えて、エンティティ Bean にほかの検索メソッドをいくつでも追加することができます。

クライアントがインスタンスのホームオブジェクトの検索メソッドを呼び出すと、次の処理が実行されます。

1. コンテナはセキュリティチェックを行い、検索メソッドのトランザクション属性で指定されたトランザクション制御を適用します。
2. コンテナはプール内の匿名インスタンスの検索メソッドを呼び出します。
3. 検索メソッドは 1 つ以上のインスタンスの主キーを返します。検索メソッドが返すのは主キーだけです。
4. コンテナは、それぞれの主キーに対応するリモートオブジェクトを検出または作成し、クライアントにこれらのオブジェクトの参照を返します。

注 - IDE は `findByPrimaryKey` メソッドのシグニチャを生成します。プログラマは、Bean に必要なその他の検索メソッドを作成する必要があります。

クライアントは、リモートオブジェクトのメソッドを使用して、検出されたインスタンスのビジネスメソッドを呼び出すことができます。30 ページの「ビジネスロジックの実行」を参照してください。

ビジネスロジックの実行

EJB クライアントは、エンティティ Bean のインスタンスに処理を行わせる必要が生じたときに、該当するインスタンスのリモートオブジェクトのビジネスメソッドを呼び出します。それに応じて、コンテナは次の処理を行います。

1. セキュリティチェックを行い、ビジネスメソッドのトランザクション属性で指定されたトランザクション制御を適用します。
2. インスタンスのビジネスメソッドを呼び出します。

ビジネスメソッドが終了すると、クライアントはその結果を受け取ります。35 ページの「Bean インスタンスのプールの作成と管理」で説明したように、コンテナは必要に応じてインスタンスを休止させます。

注 - IDE を使用すると、リモート (またはローカル) インタフェースと Bean クラスの両方にビジネスメソッドのシグニチャを簡単に作成することができます。プログラマは、Bean クラスのビジネスメソッドのコードを完成させる必要があります。

Bean インスタンスを使用した既存のエンティティの削除

(データストアからデータを削除するために) 既存のエンティティを取り除く必要が生じると、EJB クライアントはインスタンスのホームオブジェクトやリモートオブジェクトの削除メソッドを呼び出します。それに応じて、コンテナは次の処理を行います。

1. セキュリティチェックを行い、削除メソッドのトランザクション属性で指定されたトランザクション制御を適用します。
2. インスタンスの `ejbRemove` メソッドを呼び出します。これにより、CMP エンティティ Bean のインスタンスではコンテナが削除するデータが用意され、BMP エンティティ Bean のインスタンスではデータが削除されます。
3. トランザクションを適切にコミットします。

注 - IDE は、`ejbRemove` メソッドのシグニチャを生成します。プログラマは、このメソッドを完成させる必要があります。

インスタンスとデータストアとの同期

コンテナは、トランザクションの特定の時点で、Bean インスタンスのデータをデータストアのデータに同期させる必要があります。そのために、コンテナは次の処理を行います。

- エンティティがアクティブなトランザクションに移行したときに、インスタンスの `ejbLoad` メソッドを呼び出します。
  - CMP エンティティ Bean では、コンテナがエンティティオブジェクトの状態を、データストアから Bean のコンテナ管理フィールドに読み込んだ後で、このメソッドが呼び出されます。プログラマは、このメソッドを使用して、コンテナが読み込んだフィールド値に演算を適用できます。
  - BMP エンティティ Bean では、通常このメソッドはデータストアからデータを読み込み、Bean のインスタンス変数に代入します。
- トランザクションがコミットされるか、インスタンスが非活性化されたときに、インスタンスの `ejbStore` メソッドを呼び出します。
  - CMP エンティティ Bean では、コンテナがコンテナ管理フィールドの内容をデータストアに書き込む前に、このメソッドが呼び出されます。プログラマは、このメソッドを使用して、コンテナ管理フィールドの内容 (すなわちデータストアに書き込む値) を用意することができます。
  - BMP エンティティ Bean では、このメソッドは Bean のインスタンス変数の値をデータストアに書き込みます。

注 - IDE は、`ejbLoad` メソッドと `ejbStore` メソッドのシグニチャを生成します。BMP エンティティ Bean では、プログラマがこれらのメソッドを完成させる必要があります。CMP エンティティ Bean では、データストアとの同期はコンテナによって管理されるため、通常はこれらのメソッドにコードを追加する必要はありません。

関係 CMP エンティティ Bean のセットとコンテナ管理による関係

CMP エンティティ Bean のインフラストラクチャの作成には、1 度に 1 つのインフラストラクチャだけを作成する EJB ビルダーのウィザードを使用できます。ただし、外部キーをもつデータベース表や連結された表に対して、複数の CMP エンティティ Bean のインフラストラクチャを作成したい場合は、こういった CMP エンティティ

Bean 全体のインフラストラクチャを一度にまとめて生成した方が簡単で信頼性も高くなります。この操作に用意された EJB ビルダーのウィザード画面では、データベースまたはスキーマ中の表を表示し、選択した表から対応する CMP エンティティ Bean のセットを生成できます。Bean を作成すると共に、ウィザードは外部キーを表す論理エンティティを作成し、表を連結します。さらに、格納用の EJB モジュールを生成し、最後に Bean の関係を追跡します。

作成したセットに含まれる CMP エンティティ Bean 1 つ 1 つは、個々に独立して作成された CMP エンティティ Bean と違いがありません。Bean の機能、処理能力、プロパティ、およびライフサイクルは同じです。ただし、関係 CMP エンティティ Bean のセットの生成にウィザードを使用した場合、表の連結や外部キー用に必要な Enterprise Bean のコードにあたるようなコードを記述する必要はありません。IDE は、こういったリンクをコンテナ管理による関係 (CMR: Container-Managed Relationship) と呼ばれる論理フィールドとして表します。CMR フィールドは外部キーのようなものです。EJB QL による照会では、CMP フィールドの代わりに CMR フィールドを使用して表の連結に等しい処理を実行できます。

EJB コンテナは、Enterprise JavaBeans 仕様に沿って CMR を管理することで、関連する複数の CMP エンティティ Bean 間の参照の完全性を確保します。IDE Collection API を使用すれば、Bean の CMR を操作することができます。CMR についての情報は、関連する Bean のセットが常駐する EJB モジュールに含まれています。

Bean では、クラスは CMR の指向性と基本性を指定する抽象アクセサメソッドとなっています。たとえば、`Order` と `LineItems` という 2 つの Bean 間の関係は次のようになります。

- `Order` Bean には `getLineItems` メソッドと `setLineItems` メソッドがあります。`Order` Bean は、これらのメソッドを使用して注文用の商品リストにアクセスできます。
- `LineItems` Bean には `getOrder` メソッドと `setOrder` メソッドがあります。`Order` Bean は、これらのメソッドを使用して商品が属する注文にアクセスできます。

CMR には、カスケード削除機能が使用できます。この機能は宣言的に指定され、配備記述子に保存されます。

CMR フィールドは、CMP エンティティ Bean からのローカルインスタンスへのアクセスを提供します。したがって、CMR フィールドを持てるのはローカル型インタフェースを持つ Bean だけです。

メッセージ駆動型 Bean

クライアントからメッセージを受信し、そのメッセージに従いプロセスを非同期的に開始する、アプリケーションコンポーネント間の仲介として動作する特殊なエンタープライズ Bean があります。これをメッセージ駆動型 Bean といい、エンタープライズ Bean の多くの機能と Java メッセージサービス (JMS) メッセージ指向型ミドルウェア (MOM: Message-Oriented Middleware) のリスナーとしての機能を兼ね備えています。メッセージ駆動型 Bean を使用すると、スレッドや並行処理に近い処理が EJB 環境で実行できます。

J2EE アプリケーションのエンタープライズ Bean が RMI 呼び出しに応答するのに対し、メッセージ駆動型 Bean は他のアプリケーションコンポーネント (通常はクライアント) から送信されるメッセージの特定のリソースを待機します。こういったメッセージが着信すると、その時点で動作しているプロセッサまたはサーバーに関係なく、`onMessage` メソッドが起動され、メッセージの受信がメッセージ駆動型 Bean に通知されます。メッセージ駆動型 Bean は、ステートレスセッション Bean を呼び出してプロセスを開始すると共に、そのメッセージに従った処理を実行します。

メッセージソースの使用 (送信先)

送信先はクライアントがメッセージを送信する相手で、メッセージ駆動型 Bean が待機するリソースです。キューやトピックなどが送信先となります。

- **キュー** - メッセージキューは、ポイントツーポイントまたはプル (電子メールの発信・受信形態に類似したモデル) を使用します。クライアントは、メッセージをキューオブジェクトに送信します。メッセージ駆動型 Bean はキューに対して定期的にポーリングし、自分宛のメッセージを取得します。メッセージ 1 つは 1 つの受信者に対して送信されます。
- **トピック** - メッセージトピックはパブリッシュとサブスクライブまたはプッシュ (オンラインニュースへの登録に類似したモデル) を使用します。クライアントはトピックオブジェクトへメッセージを送信します。その特定のトピックに登録しているすべてのメッセージ受信者は同じメッセージを受信します。1 つのメッセージは多くのメッセージ受信者にブロードキャストされます。

  トピックサブスクリプションには永続と非永続があります。

  - **永続** - メッセージは、後でメッセージ受信者がシステムに接続したときに取り出せるよう保存されます。

- **非永続** - メッセージは、そのときに接続していた受信者だけが取得でき、古いメッセージは保存されません。

メッセージ駆動型 Bean が有効な利用形態

メッセージ駆動型 Bean を使用するアプリケーションでは、他のアプリケーションコンポーネントの状態への依存性が最少になります。メッセージ駆動型 Bean は 1 方向の操作でだけ使用されます。

送信先が有効になっている限り、メッセージ駆動型 Bean のサーバーや目的のアプリケーションが現在配備されているかどうかに関係なく、アプリケーションクライアントはメッセージを送信先に送信できます。コンテナが、クライアントによって起動されたプロセスの完了を待つ必要はありません。さらに、メッセージ駆動型 Bean とそれが呼び出した Bean が処理を実行している間、クライアントはサーバーに拘束される必要がなくなります。したがって、1 つ以上のクライアントが 1 つ以上のサーバーにメッセージを送信し、複数のプロセスを起動させることができます。

メッセージが着信する前に、アプリケーションが処理に時間のかかるプロセスを開始させたり、サーバーが停止したり、またはその他の理由によってリソースが利用できなくなった場合、メッセージ駆動型 Bean が形成する中間層を利用して処理を継続させることができます。メッセージ駆動型 Bean は、一方でユーザーへの応答を確保しながら、他方でプロセスを開始できるようなクライアントに理想の手段を提供します。たとえば、オンラインショッピングのアプリケーションではメッセージ駆動型 Bean を使用して、顧客のクレジットカードの番号が有効かどうかを確認している最中にも、顧客が他の商品の検索を続けられるような処理を実行できます。ショッピングアプリケーションのクライアントコンポーネントはメッセージ駆動型 Bean にメッセージを送信するだけで、これまでの処理を継続できます。

メッセージ駆動型 Bean の利用は、アプリケーションの負荷分散とスケジューリングの効率化をもたらします。たとえば、データベースの閑散期に処理を開始したりできます。非同期型プロセスは、異なる時間帯に属していたり、地理的に離れたりしている複数のシステムでの通信や処理で特に有利です。

あるアプリケーションが、処理手続きのよくわからない他のアプリケーションと対話型の処理を実行する必要がある場合、メッセージ駆動型 Bean を使ってこの 2 つのアプリケーションをゆるやかに接続させることができます。多くの旧型システムはメッセージによる処理ができるので、 J2EE アプリケーションとのインタフェースとしてメッセージ駆動型 Bean が利用できます。

メッセージ駆動型 Bean は `onMessage` メソッドが呼び出された場合にだけ、JMS 環境と対話します。IDE の EJB ビルダーのウィザードを使用して生成したメッセージ駆動型 Bean は、JMS と透過的に結合できるため、 JMS コードを別に記述する必要がありません。JMS への接続とそのメッセージ用のチャネル (送信先) は Bean のプロパティとして指定するため、必要であれば 1 つのメッセージ駆動型 Bean を簡単に書き換えて、異なる送信先へ接続できます。

メッセージ駆動型 Bean が有効でない利用形態

状況によっては、メッセージ起動型 Bean の利用が適切でないこともあります。次にそういった状況の例を挙げます。

- 戻り値が必要な場合。メッセージ駆動型 Bean が、`void` 以外の値を戻す場合は、手作業によるコーディングが必要です。また、これは特定のクライアントだけに対して記述される必要があります。結果を戻す必要がある場合には、セッション Bean を利用した方が効率的です。
- 処理が成功したことを確認する必要がある場合。メッセージ駆動型 Bean は、他のエンタープライズ Bean と異なり例外をスローできません。
- Bean の操作が、一定時間内に処理を終了する必要があるトランザクションの一部である場合。
- サーバーがクライアントのセキュリティ ID を知る必要がある場合。メッセージは、メッセージ駆動型 Bean にセキュリティ ID を伝播しません。この種類の Bean では、すべてのインスタンスは同じに扱われます。
- 性能が要求される場合。メッセージによる処理はクライアントとサーバーとの中間層になります。メッセージ駆動型 Bean が比較的軽量だとはいえ、層をもう 1 つ持つことはシステム応答時間を遅らせる原因になります。
- アプリケーションを小型で単純に保ちたい場合。非同期型プロセスを必要としないアプリケーションは、コーディングとデバッグが簡単です。

メッセージ駆動型 Bean のライフサイクル

実行時には、アプリケーションクライアントによってメッセージ駆動型 Bean が待機する送信先にメッセージが送信されます。これらのメッセージが着信すると、アプリケーションサーバーは、その Bean のインスタンスを作成してクライアントの要求に応えます。

この種類の Bean は非常に単純なライフサイクルを持っています。ステートレスセッション Bean の場合、インスタンスは EJB コンテナによって動的に管理されるいくつかの段階を経て処理されます。そして、インスタンスが不要になると破棄されます。

メッセージ駆動型 Bean のライフサイクルでの個々の段階、Bean が次の段階に移るきっかけとなるメソッド、および EJB プログラマの役割についてを次の節で説明します。

Bean インスタンスの生成と初期化

メッセージ駆動型 Bean の実行時ライフサイクルは、クライアントがキューまたはトピックに対して、メッセージ駆動型 Bean に取得される (読み込まれて処理される) メッセージを送信した時点で開始します。それに応えて、EJB コンテナは次の 2 つのメソッドを順に呼び出します。

1. `newInstance` メソッド - メッセージ駆動型 Bean の新規インスタンスを生成します。
2. `setMessageDrivenContext` メソッド - 各インスタンスをメッセージ駆動型コンテキストオブジェクトと関連付けます。
3. `ejbCreate` メソッド - インスタンスを初期化します。

注 - IDE は、`setMessageDrivenContext` メソッドと `ejbCreate` メソッドのシグニチャを生成します。プログラマは、このメソッドの本体を完成させる必要があります。

メッセージの送信後は、結果が戻される場合を除き、クライアントが処理に関与する必要はありません。

ビジネスロジックの実行を目的とした他の Bean の起動

これでメッセージ駆動型 Bean のインスタンスが生成され、初期化されました。インスタンスとコンテナは、次のように協調動作します。

- Bean インスタンスは、メッセージを取得し、クライアントの要求するタスクを確認します。
- 同じ Bean インスタンスが、適切なステートレスセッション Bean のインスタンスの作成をコンテナに依頼します。

- コンテナがセッション Bean インスタンスを作成し、メッセージ駆動型 Bean の onMessage メソッドのトランザクション属性よって指定されたトランザクション制御を適用します。
- メッセージ駆動型 Bean インスタンスによって、ステートレスセッション Bean のインスタンスにあるビジネスメソッドを呼び出します。

最後に、指定がある場合は、クライアントが再び同じセッション Bean またはそのサーバー上の他のエンタープライズ Bean を呼び出して、ビジネスメソッドの結果を受け取ります。

注 - IDE は、onMessage メソッドのシグニチャを生成します。プログラマはこのメソッドの本体を完成される必要があります。また、EJB モジュールのプロパティシートを使用して、Bean のメソッドにトランザクション属性を設定します。

Bean インスタンスの削除

メッセージ駆動型 Bean の仕事は、アプリケーション中の他の Bean に割り当てられたタスクを手渡した時点で終了します。コンテナはメッセージ駆動型 Bean インスタンスの ejbRemove メソッドを呼び出して、そのインスタンスが使用して開いたままになっているリソースを終了します。その後、コンテナはその処理を終了したインスタンスをメモリーから削除します。

注 - IDE は ejbRemove メソッドのシグニチャを生成します。プログラマは、このメソッドの本体を完成させる必要があります。

メッセージ駆動型 Bean インスタンスのプールへの格納

ステートレスセッション Bean の場合、コンテナはメッセージ駆動型 Bean の複数のインスタンスを同時に生成し、プールへ格納できます。コンテナは、自身の命令でインスタンスプールを管理できます。着信メッセージの数が増えた場合には新規インスタンスを作成でき、メモリーが足りなくなった場合はインスタンスを削除できます。

メッセージ駆動型 Bean のインスタンスが状態情報を維持していない場合は、プール中のメッセージ駆動型インスタンスはすべて同一で、どれも同じように使用できます。

J2EE 仕様は、あるメッセージ駆動型 Bean の複数のインスタンスに対するメッセージが特定の順序で配信されることを保証しません。したがって、アプリケーションは、任意の順序で配信された複数メッセージを処理できることが前提となります。

アプリケーションでのエンタープライズ Bean の使用

メッセージ駆動型 Bean、セッション Bean、エンティティ Bean の組み合わせの要否およびその方法は、アプリケーションの目的によって決まります。場合によっては、1 種類の Bean だけを使用した方が良い結果を得られることもあります。ごく単純なアプリケーションの場合は (CRUD 操作を 1 つだけ実行するアプリケーションなど)、EJB モジュールにセッション Bean またはエンティティ Bean を 1 つだけ持たせることができます。それ以外の場合には、複数の種類のエンタープライズ Bean の能力と機能を最大に実行できるような形態が望まれます。

EJB モジュールにエンタープライズ Bean を追加することで、アプリケーションの処理範囲と能力を増やし続けていくことができます。EJB アプリケーションは、高い拡張性を持っています。

次に、アプリケーション内でのエンタープライズ Bean とその他のコンポーネントとの組み合わせの例を挙げます。

- ステートフルセッション Bean 1 つと複数の CMP エンティティ Bean を含む EJB モジュールの場合。セッション Bean は、複数のユーザーセッションを形成します。各セッションでは、そのセッション Bean のインスタンスがエンティティ Bean のインスタンスに命令し、データベースに対する検索とデータの書き込みを実行します。EJB コンテナは、そのためのエンティティ Bean の持続性とトランザクションを管理します。
- 複数の CMP エンティティ Bean を含む EJB モジュールの場合。このモジュールは、同じアプリケーション内にある Web モジュールと対話します。Web モジュールはアプリケーションクライアントとして動作し、それが持つ 1 つ以上のコンポーネントによって EJB モジュール中の個々のエンティティ Bean のメソッドが呼び出されます。エンティティ Bean はデータベースと対話し、エンドユーザーには Web モジュールのコンポーネントを介して結果が戻されます。

- メッセージ駆動型 Bean とセッション Bean を 1 つずつと、1 つ以上のエンティティ Bean を含む EJB モジュールの場合。このモジュールは、Web モジュールと対話します。この Web モジュール中のクライアントコンポーネントはキューに対してメッセージを送信します。メッセージ駆動型 Bean は待機し、このキューへのメッセージを取得すると、そのセッション Bean で非同期型プロセスを開始します。このプロセスを通じて、エンティティ Bean からのデータベースに対する処理が実行されます。

これらの例については、マニュアル『J2EE アプリケーションのプログラミング』で詳しく説明されています。

例外を使用した問題の対処

実行時に問題が発生した場合の対処方法は、Bean クラスで定義します。データベース接続を使用できない、データベースのメモリが不足しているために SQL の挿入エラーが発生する、オブジェクトが見つからないといったシステムレベルの問題は、`javax.ejb.EJBException` インタフェースを使用するシステム例外で表現します。コンテナは、このタイプの例外を検出し、リモート例外の中に組み込み、システム管理者が対処できるようにクライアントに返します。

エンタープライズ Bean のビジネスロジックのエラーといったアプリケーションレベルの問題は、`javax.ejb` パッケージなどに用意されている事前定義例外か、プログラマが作成するカスタム例外で処理できます。コンテナは、このタイプの例外を検出し、クライアントに返して対処させます。

Sun ONE Studio IDE のウィザードを使用して、エンタープライズ Bean やそのメソッドを作成すると、必要な例外がメソッドのシグニチャに自動的に追加されます。たとえば、ホームインタフェースとリモートインタフェースのすべてのメソッドのシグニチャには、`java.rmi.RemoteException` が組み込まれます。さらに、すべての生成メソッドのシグニチャには、`javax.ejb.CreateException` が組み込まれます。

IDE のエクスプローラウィンドウの GUI 機能を使用してメソッドを作成した場合は、メソッドからスローするアプリケーションレベルの例外も指定することができます。これらのアプリケーション例外は、リモート (またはローカル) インタフェースと Bean クラスの両方に自動的に追加されます。

配備記述子の操作

エンタープライズ Bean の基本目的は、同じエンタープライズ Bean を別々のアプリケーションで再利用できるようにし、別々のサーバーに配備できるようにすることです。この目的のために、個々のサーバーが実行時に知る必要のあるすべての情報を、配備記述子という XML メタファイルにまとめます。この記述子ファイルには、Bean の構造、ほかの Bean との関係、データストアの場所、ユーザーがデータストアにアクセスするのに必要な情報、その他外部依存性についての情報がすべて含まれています。

エンタープライズ Bean を作成すると、IDE によって、その Bean の初期配備記述子が生成されます。Bean のプロパティシートを使用すると、Bean の外部依存性をすべて宣言することができます。IDE を使用して、Bean を EJB モジュールにアセンブルするときに、Bean のデフォルトのプロパティ値に優先する値を指定したり、EJB モジュール全体のプロパティを設定したりできます。これらのプロパティは、EJB モジュールのプロパティシートで設定することもできます。配備時には、IDE は EJB モジュールの配備記述子を生成し、その中に プログラマが指定したプロパティをすべて組み込みます。

セキュリティポリシーの適用

EJB コンテナには、アプリケーションを保護する機能、すなわちエンタープライズ Bean のメソッドを呼び出すことのできるユーザーを制限する機能があります。アプリケーションのセキュリティポリシーは、宣言とプログラムのどちらかで指定できます。宣言によるセキュリティは、配備記述子の中で指定するため、配備を行う前であれば、いつでも変更できます。プログラムによるセキュリティは、エンタープライズ Bean のコードの中で、プログラマが定義します。

ほとんどの場合は、宣言によるセキュリティ指定が適しています。宣言によるセキュリティは、指定するのが簡単で、開発、アセンブル、配備のどの段階でも指定することができます。

プログラムによるセキュリティ指定は、これよりも複雑です。ただし、セキュリティをより細かく制御できるので、アプリケーションによっては、このセキュリティが必要になります。たとえば、呼び出し元のユーザーの識別情報に応じて、メソッドの本体で別々のロジックを実行したい場合は、プログラムによるセキュリティを使用する必要があります。

エンタープライズ Bean のセキュリティポリシーを指定するには、アプリケーションに対する一連のセキュリティロールを定義します。セキュリティロールとは、エンタープライズ Bean のメソッドの実行権限を共有するユーザーの集まりです。

宣言によるセキュリティ指定では、それぞれのセキュリティロールを、そのロールを持つユーザーが実行可能な Bean メソッドに割り当てます。実行時には、コンテナはメソッドを呼び出したユーザーのセキュリティロールをチェックし、そのユーザーにメソッドを実行する権限を与えるかどうかを決定します。

プログラムによるセキュリティ指定では、コンテナから提供されるメソッド (`getCallerPrincipal` と `isCallerInRole`) を使用して、メソッドを呼び出したユーザーの識別情報やロールを特定し、必要に応じて条件付きロジックを使用することができます。

エンタープライズ Bean のセキュリティの宣言

エンタープライズ Bean を EJB モジュールにアセンブルした後で、セキュリティロールとメソッドの実行権限を宣言します。EJB モジュールのプロパティシートでは、該当する EJB モジュールに対するセキュリティロールを定義できます。アセンブル後の EJB コンポーネントのプロパティシートでは、セキュリティロールごとに、そのロールを持つユーザーが実行可能なメソッドのリストを定義します。

宣言によるセキュリティを使用した場合は、開発段階やテスト段階の任意の時点で、セキュリティ権限を変更することができます。同じ Bean を含んでいる EJB モジュールごとに、セキュリティロールやメソッドの実行権限を使い分けることもできます。

エンタープライズ Bean のセキュリティのプログラミング

プログラムによるセキュリティ指定では、次の情報を特定することができます。

- メソッドを呼び出したユーザーの個人識別情報
- メソッドを呼び出したユーザーに、特定のセキュリティロールが与えられているかどうか

これらの情報をもとに、ユーザーの識別情報やロールに応じてロジックを条件分岐させることができます。

呼び出し元のユーザーの識別情報を特定するには、`javax.ejb.EJBContext` オブジェクトの `getCallerPrincipal` メソッドを使用します。このメソッドから返される `java.security.Principal` オブジェクトから、呼び出し元のユーザーの名前を特定することができます。この情報を使用してユーザーについてのさらに詳しい情報をデータベースに照会できます。

呼び出し元のユーザーに、特定の論理的ロールが与えられているかどうかを確認するには、`javax.ejbEJBContext` オブジェクトの `isCallerInRole(String` <ロール名>) メソッドを使用します。このメソッドは、呼び出し元のユーザーに、引数で指定された論理的ロールが与えられているかどうかを示す Boolean 値を返します。このメソッドを使用する場合は、コードで使用する <ロール名> を、Bean のプロパティシートでのセキュリティロールの参照として宣言する必要があります。

アセンブル時に、この Bean を EJB モジュールに組み込むと、その Bean でのセキュリティロールの参照が、EJB モジュールで定義されたセキュリティロールに対応付けられます。したがって、プログラマは、アセンブル時に決定される実際のセキュリティロール名を知る必要はありません。

エンタープライズ Bean と J2EE アプリケーションのセキュリティ機能の実装に関する詳細については、マニュアル『J2EE アプリケーションのプログラミング』を参照してください。

アプリケーションサーバーとデータベース

Sun ONE Studio IDE で作成したエンタープライズ Bean は、IDE に組み込まれているアプリケーションサーバーを使用してテストするのが一般的です。このサーバーは、J2EE リファレンス実装 (RI) サーバーであり、非商用のオペレーションサーバーです。このサーバーは、デモンストレーション用、プロトタイプ用、教育用に無料で使用することができます。エンタープライズ Bean を RI でテストして、さまざまなアプリケーション条件での Bean の動作を確認することができます。このマニュアルでの例題はすべて、RI をアプリケーションサーバーとして記述されています。

その他のアプリケーションサーバーおよび IDE に使用できるサーバープラグインについての情報は、「Sun ONE Studio リリースノート」を参照してください。

IDE を使用して生成したエンティティ Bean は、IDE に組み込まれているデータベース PointBase Server 4.2 Restricted Edition を使用してテストできます。このマニュアルでの例題はすべて、PointBase をデータベースとして記述されています。

詳細情報の参照先

このマニュアルですでに紹介した仕様書と Blueprints に加えて、EJB プログラマ向けの多数の資料があります。たとえば、次の各文書には、エンタープライズ Bean の設計やプログラミングを改良するためのテクニックが示されています。

- Sun ONE Studio 4 tutorials and example applications
  `http://forte.sun.com/ffj/documentation/tutorialsandexamples.html`
- Java 2 Platform, Enterprise Edition Blueprints
  `http://java.sun.com/j2ee/blueprints`
- Designing Enterprise Applications with the J2EE Platform, Second Edition
  `http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/index.html`
- Seven Rules for Optimizing Entity Beans (著: Akara Sucharitakul)
  `http://developer.java.sun.com/developer/technicalArticles/eBeans/sevenrules/`
- Working with J2EE Application Clients (著: Monica Pawlan)
  `http://developer.java.sun.com/developer/technicalArticles/J2EE/appclient/`
- Designing Entity Beans for Improved Performance (著: Beth Stearns)
  `http://developer.java.sun.com/developer/technicalArticles/eBeans/ejbperformance/`

第3章

セッション Bean の開発

Sun ONE Studio IDE の EJB ビルダーを使用して、セッション Bean をプログラミングすることができます。セッション Bean は、Enterprise JavaBeans アプリケーションの内部で、クライアントのためにサーバー側のビジネスロジックを実行します。この章では、ステートフル (状態を保持する)、およびステートレス (状態を保持しない) セッション Bean の作成、操作方法を説明します。

どちらの種類のセッション Bean でも、EJB コンテナにトランザクションを管理させたり、Bean 提供者がコードを作成して、セッション Bean 自身にトランザクションを管理させたりすることができます。セッション Bean は、JDBC API を使用して持続データにアクセスします。代わりに、IDE の透過的持続性 (Transparent Persistence) モジュールを使用することもできます。1 つのセッション Bean で、1 つ以上のエンティティ Bean を管理することができます。

IDE のウィザードを使用すると、エンタープライズ Bean の要素である Bean クラスとそのインタフェースを 2 つまたは 4 つ作成できます。デフォルトのインタフェースはリモートインタフェースとホームインタフェースですが、これらをローカルインタフェースおよびローカルホームインタフェースで代替したり、前者の 2 種類のインタフェースに後者 2 種類のインタフェースを追加したりできます。セッション Bean の作成に関する処理の多くは自動化されています。

セッション Bean のプログラミングでは、この章に記載するオプションのほかにも、さまざまなオプションを指定することができます。Sun ONE Studio IDE は、コーディング作業を省力化することを目的にしていますが、これらのオプションを柔軟にサポートし、Bean 提供者が多数の情報を指定できるように配慮されています。セッション Bean の詳細なコーディング手順については、xviii ページの「お読みになる前に」に記載した文献、または エンタープライズ Bean のプログラミングについての資料を参照してください。

EJB ビルダーの使用

EJB ビルダーは、ウィザード、プロパティシート、およびエディタから構成されています。これらの機能を使用して、エンタープライズ Bean を整合性のとれた方法で簡単に作成することができます。EJB ビルダーがインストールされているかどうかを確認するには、メインウィンドウから「ツール」 > 「オプション」 > 「IDE 構成」 > 「システム」 > 「モジュール」 > 「J2EE サポート」を選択してください。モジュールのリストに「EJB 2.0 ビルダー」が表示されていて、プロパティシートの「使用可能」プロパティが True に設定されていれば、EJB ビルダーは使用できる状態になっています。

Sun ONE Studio IDE では、いくつかの方法でセッション Bean を作成できますが、この章で推奨している手順に従うと、もっとも広範にサポート機能を活用し、通常はもっとも早く Bean を完成させることができます。この章では、 IDE の機能を最大限に活用した、Bean の一貫性と J2EE 標準への準拠を確実にする手法が記載されています。

最良の結果を得るために、EJB ビルダーを使用して、次の手順でセッション Bean をプログラミングしてください。

- セッション Bean と、セッション Bean に必要なクラスを作成します。

  EJB ビルダーのウィザードの手順に従うと、セッション Bean の枠組みができあがります。その Bean の 3 つ (または 5 つ) のクラスと論理ノードは、エクスプローラの「ファイルシステム」タブに表示されます。ウィザードは、これらのすべてのクラス用の宣言を生成します。Bean 提供者は、メソッドの実装を提供する必要があります。

  論理ノードは、セッション Bean を操作する出発点として最適です。論理ノードは、エクスプローラにアイコンで表示されます。

- メソッド、パラメータ、および例外を追加します。

  後述の手順に従って、IDE の GUI 機能を使用します。コンテキストメニューから表示できるダイアログを使用するか、必要なクラスのセットを直接編集することで、Bean にメソッドを追加することができます。

- セッション Bean の配備記述子を設定します。

  論理ノードで使用できるセッション Bean のプロパティシートを使用して、プロパティを編集します。このプロパティシートは、論理ノードから表示させることができます。

セッション Bean の論理ノードから、Bean を検査できます。また、Bean に対して IDE のテスト機能を使用することもできます。

セッション Bean の種類の選択

セッション Bean は、クライアントとアプリケーションサービスとの相互作用を処理します。この相互作用が続く期間をセッションといいます。まず、ステートフルセッション Bean とステートレスセッション Bean のどちらを作成すればよいか、Bean 管理によるトランザクションとコンテナ管理によるトランザクションのどちらを使用すればよいかを決定する必要があります。ここからは、これらの選択肢について説明します。

EJB ビルダーは、これらの選択肢をすべてサポートしています。どのようなセッション Bean を作成する場合も、同じウィザードを使用してセッション Bean の基盤を作成することができます。この作業の後で、セッション Bean の種類ごとに指定作業を行います。

詳細については、次の節を参照してください。

- 55 ページの「ステートフルセッション Bean とステートレスセッション Bean」
- 57 ページの「コンテナ管理によるトランザクションと Bean 管理によるトランザクション」

ステートフルセッション Bean とステートレスセッション Bean

セッション Bean の主な目的は、クライアントアプリケーションの代わりに処理を実行することです。つまり、クライアント側とサーバー側のエンティティ Bean との通信を橋渡しすることです。この通信が、複数回の要求と応答の組み合わせから構成される場合は、通信を管理するセッション Bean は、通信が終了するまで特定の情報を

保持する必要があります。その場合には、ステートフルセッション Bean が必要です。それよりも単純な通信を管理する場合は、ステートレスセッション Bean を使用することができます。

この選択肢の詳細については、第 2 章を参照してください。表 3-1 に、設計上の検討項目をいくつか示します。

表 3-1 ステートフルセッション Bean とステートレスセッション Bean の選択

項目	ステートレス	ステートフル
スコープ	ステートレスセッション Bean は、クライアントーエンティティ間の単純な通信を管理し、1 回のセッションで 1 つのメソッドだけを呼び出す	ステートフルセッション Bean は、クライアントーエンティティ間のより複雑な通信を管理し、1 回のセッションで複数のメソッドを呼び出す
初期化	ステートレスセッション Bean には、初期化が必要なデータはない	ステートフルセッション Bean の状態を初期化する必要がある。たとえば、リモートリソースへのアクセスを設定する Bean は、リソースファクトリへの参照を取得する必要がある
情報の保存	ステートレスセッション Bean は、セッション全体にわたって、メソッド呼び出しの間の状態情報を保存しない	ステートフルセッション Bean は、セッション全体にわたって、クライアントーサーバー間の通信状態を保持する。複数のメソッドの呼び出しにわたって状態情報を保存し、セッションが終了したときに、これらの情報を破棄する
クライアントとの関係	ステートレスセッション Bean のインスタンスは、1 度に 1 つのクライアントのために、1 つのオペレーションだけを実行する。1 つのメソッドの呼び出しが完了した時点で、このインスタンスをプールに格納し、同じセッションの間であっても、別のクライアントに再割り当てすることができる	ステートフルセッション Bean のインスタンスは、1 度に 1 つのクライアントのために、一連のオペレーションを実行する。そのクライアントのセッションが完了した時点で、このインスタンスは破棄される (プールには格納されない)

表 3-1　ステートフルセッション Bean とステートレスセッション Bean の選択 (続き)

項目	ステートレス	ステートフル
アプリケーション例	ステートレスセッション Bean はカタログビューアとして使用できる。Bean のメソッド 1 つで、エンドユーザーにオンラインカタログの商品を 1 つだけ検索させるようにできる。	ステートフルセッション Bean はオンラインショッピングカートとして使用できる。この Bean から複数のメソッドを起動して、エンドユーザーが選択した商品を蓄積し、複数の商品の購入を一括して処理できる。

コンテナ管理によるトランザクションと Bean 管理によるトランザクション

Bean のトランザクションをコンテナに管理させるか、それとも Bean 自身に管理させるかを指定する必要があります。詳細については、第 2 章を参照してください。これらの選択肢の違いを表 3-2 に示します。

表 3-2　コンテナ管理によるトランザクションと Bean 管理によるトランザクションの選択

項目	コンテナ管理によるトランザクション	Bean 管理によるトランザクション
トランザクション範囲の設定	EJB コンテナが Java 2 Platform, Enterprise Edition 仕様に従って、トランザクションをいつ開始し、いつコミットするかを決定する	プログラマがトランザクションの範囲を明示的にコーディングする。トランザクションをより綿密に制御できる
トランザクションの管理	コンテナ自身がトランザクションを管理する	JTA を使用してトランザクションを管理する。JTA には、JDBC といったほかのリソース用のトランザクションを組み込むことができる
トランザクションとメソッドの関係	1 つのメソッドでは 1 つのトランザクションしか取り扱えない。トランザクションに関係しないメソッドも使用することができる	1 つのメソッドで複数のトランザクションをコーディングすることができる。ただし、状況がより複雑になる

コンテナ管理によるトランザクション (CMT) を使用する通常の エンタープライズ Bean では、コンテナはメソッドが開始される直前にトランザクションを開始し、メソッドが終了する直前にトランザクションをコミットします。CMT を使用した場合は、クライアントにトランザクションを制御させることができます。たとえば、クライアントは、ステートフル CMT セッション Bean から呼び出される別々のメソッドを使用して、論理的なビジネストランザクションを 1 つにまとめることができます。

Bean 管理によるトランザクション (BMT) を使用するセッション Bean では、Bean 提供者が作成するコードの中で、トランザクションの開始および終了を指定する必要があります。

セッション Bean の定義

EJB ビルダーのウィザードを使用すると、セッション Bean に必要な 3 つのデフォルトクラスである Bean クラスと選択したインタフェース (リモートかローカル、またはその両方) の作成作業の大半が自動化されます。セッション Bean を定義するには、次の手順に従います。

1. セッション Bean を格納するパッケージを選択または作成します。
2. EJB ビルダーのウィザードを使用して、セッション Bean のインフラストラクチャを作成します。
3. Bean のコードに生成メソッドとビジネスメソッドを追加します。
4. 追加したメソッドの本体を完成させます。

ここからは、これらの基本手順を説明します。

パッケージの作成

セッション Bean を格納するパッケージを作成する必要がある場合は、ファイルシステムを選択して右クリックし、「新規」->「パッケージ」を選択します。

EJB ビルダーのウィザードの起動

セッション Bean を作成する方法を次に示します。

1. IDE のメインウィンドウから「表示」->「エクスプローラ」を選択し、IDE のエクスプローラウィンドウを開きます。
2. エクスプローラの「ファイルシステム」タブから、セッション Bean を格納するパッケージを選択します。
3. 右クリックし、「新規」->「J2EE」->「セッション EJB」を選択します。

   EJB ビルダーのウィザードが表示されます。左側のパネルに、現在の手順と、セッション Bean の作成を終えるまでの一連の手順が表示されます。

デフォルトのセッション Bean の作成

EJB ビルダーの「セッションBean 名とプロパティ」区画で、状態、トランザクション型、およびインタフェースの種類を選択する必要があります。操作方法は次のとおりです。

1. セッション Bean の名前を入力して、必要な種類のセッション Bean を選択します。

   Bean の状態、トランザクションモード、および実装するインタフェースの種類を指定するボタンをクリックします。図 3-1 に選択画面を示します。デフォルトは、「ステートレス」、「コンテナ管理」、および「リモートインタフェースのみ」です。

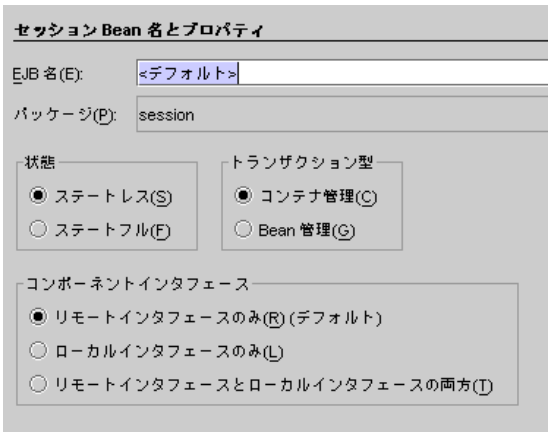


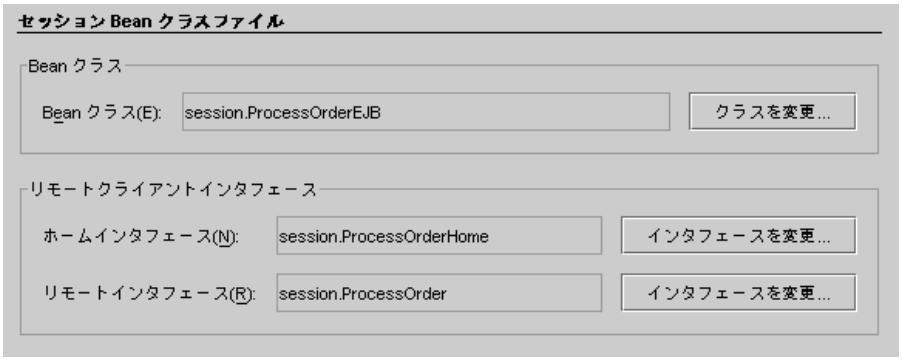
図 3-1　ウィザードで選択できるステートレスセッション (またはステートフル BMT) Bean に対する指定

注 - この最初の区画で選択した項目によって、ウィザードが生成するコードが決定されます。これらの最も基本的な選択項目を後から変更したい場合は、第 8 章で説明する Bean のプロパティシートを使用して変更できます。

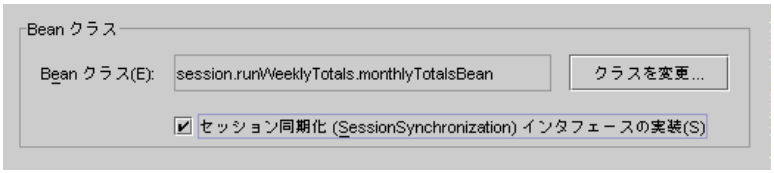
---

2. 「次へ」をクリックします。

ステートレスセッション Bean 用に、次のような「セッション Bean クラスファイル」区画が表示されます。



ステートフルセッション Bean については、さらに「セッション同期化 (Session Synchronization) インターフェースの実装」を選択できます。



この選択項目については表 3-4 と 72 ページの「セッション同期化の使用」で説明します。

3. Bean クラスとインタフェースを確認し、必要に応じて変更します。

この区画に、セッション Bean を構成するクラスがパスと共に表示されています。

- パッケージの場所を変更することができます。

- 各項目の変更ボタンを使用してクラス名を変更し、既存のクラスを指定するか、新しいクラスを作成することができます。たとえば、ホームインタフェースとリモートインタフェースはすでに指定されているものを使用し、Bean クラスだけを新しく生成できます。

  指定したパッケージの外部にあるクラスを指定した場合は、生成されるクラスは図 3-2 とは別の形式で表示されます。

- Bean のインタフェースの スーパークラスは変更しないで下さい (スーパークラスがある場合は、IDE はコード生成をスーパークラスの実装に委任します。ただし、生成されたコードの内容は基本的に確認する必要があります)。

これらのフィールドを変更する前には、次の点を考慮してください。

- **サーバーの要件**。EJB ビルダーのウィザードでは、セッション Bean の構成要素を別の場所に移動することができます。たとえば、関連するオブジェクトの格納先のパッケージ名を変更し、Bean クラスをホームインタフェースやリモートインタフェースとは別のディレクトリに格納することができます。その場合は、使用するアプリケーションサーバーが、このようなファイルの分散配置に対応しているかどうかを事前に確認しておく必要があります。

- **クラスの再利用**。必要であれば、この時点で Bean の各クラス (Bean クラス、ホームインタフェース、リモートインタフェース) を別のセッション Bean のものに置き換えることができます。置き換えようとしているクラスに、必要なメソッドや例外が含まれていない場合は、そのことを通知するメッセージが表示されます。

- **パッケージ名とディレクトリ名**。
  Java の有効なパッケージ名とディレクトリ名を指定する必要があります。

4. 「完了」をクリックします。

セッション Bean の基盤になる構成要素が自動的に生成されます。これらの構成要素については、次の節を参照してください。

セッション Bean のクラスの参照

EJB ビルダーのウィザードは、セッション Bean のデフォルトクラスを自動的に生成し、これらのクラス間の関係を設定します。典型的なセッション Bean (すべてのクラスを同じパッケージに収容する Bean) は、エクスプローラの「ファイルシステム」区画に図 3-2 のように表示されます。

図 3-2　リモートインタフェースを持つ典型的なセッション Bean のデフォルトクラス

クラスアイコン が付いたノードは、セッション Bean のクラスを表しています。Bean アイコン が付いたノードは、セッション Bean の論理ノードです。編集作業は、すべて論理ノードで行います。例題で使用されている Bean の主ノードは次のとおりです。

- リモートインタフェースは、`javax.ejb.EJBObject` インタフェースのサブクラスです。このクラスは、その Bean の EJB モジュールの外から呼び出されるセッション Bean のビジネスメソッドのシグニチャを提供します。
- Bean クラスは、`javax.ejb.SessionBean` インタフェースを実装したクラスです。このクラスには、セッション Bean のメソッドが実装されます。
- ホームインタフェースは、`javax.ejb.EJBHome` インタフェースのサブクラスです。このクラスは、そのセッション Bean の EJB モジュールの外から呼び出される生成メソッドのシグニチャを提供します。
- ローカルインタフェースを選択した場合は、`Local`*Bean_name* というラベルの付いたノードが表示されます。このインタフェースは `javax.ejb.EJBLocalObject` インタフェースのサブクラスで、Bean の EJB モジュール内から呼び出されるセッション Bean のビジネスメソッドのシグニチャを提供します。
- ローカルホームインタフェースを選択した場合は、`Local`*Bean_name*`Home` というラベルの付いたノードが表示されます。このインタフェースは `javax.ejb.EJBLocalHome` インタフェースのサブクラスで、Bean の EJB モジュール内から呼び出されるセッション Bean の生成メソッドのシグニチャを提供します。
- 論理ノードは、エンタープライズ Bean のすべての要素を 1 か所にまとめ、これらの要素を操作しやすくするために作成されます。

ノードの展開

セッション Bean のパッケージノードに含まれている 4 つのノードを展開すると、図 3-3 のようなツリーが表示されます。

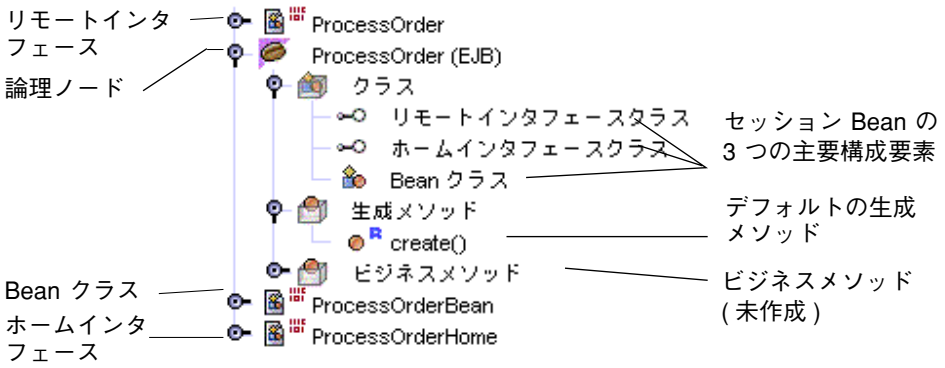


図 3-3　リモートインタフェースを持つ典型的なセッション Bean の詳細表示

生成されたクラスの確認

EJB ビルダーは、1 つの生成メソッドといくつかのライフサイクルメソッドを、セッション Bean に自動的に配置します。ここからは、これらのメソッドを説明します。

デフォルトの生成メソッド

ウィザードは、セッション Bean の各クラスに、`ejbCreate` メソッドのシグニチャを配置します。

```
public void ejbCreate() {
}
```

これに対応する `create` メソッドが、セッション Bean のホームインタフェースに配置されます。

```
public interface AcctBalHome extends EJBHome {
    public AcctBal create()
        throws RemoteException, CreateException;
}
```

詳細については、66 ページの「生成メソッドの完成」を参照してください。

ライフサイクルメソッド

ウィザードは、セッション Bean の Bean クラスに、次のライフサイクルメソッドを追加します。

```
public void setSessionContext(SessionContext context) {
    this.context = context; }
public void ejbActivate() {
}
public void ejbPassivate() {
}
public void ejbRemove() {
}
```

セッション Bean クラスでの、これらのメソッドの使用目的を表 3-3 に示します。

表 3-3　セッション Bean クラスでのライフサイクルメソッドの目的

メソッド	目的
`setSessionContext`	このメソッドは、フィールドに `SessionContext` の参照を格納し、インスタンス変数に値を格納できるようにする。このメソッドを使用して、セッション Bean の全期間にわたって存続するリソース (たとえばデータベース接続ファクトリ) を割り当てることができる。デフォルトでは、`context` というフィールドに `SessionContext` を代入するコードが生成される
`ejbActivate`	このメソッドは、Bean を初期化して使用できるようにし、インスタンスに必要なリソースを取得する
`ejbPassivate`	このメソッドは、Bean のインスタンスが非活性化される前に、その Bean が使用していたリソースを解放する
`ejbRemove`	このメソッドは、`ejbCreate` メソッドやビジネスメソッドで取得されたリソースを解放する

セッション Bean でセッション同期化インタフェースを使用する場合は、ウィザードはさらに 3 つのメソッドを Bean クラスに生成します。

```
public void afterBegin() {
}
public void beforeCompletion() {
}
public void afterCompletion(boolean committed) {
}
```

これらのセッション同期化メソッドを表 3-4 に示します。

表 3-4　セッション Bean クラスでのセッション同期化メソッドの目的

メソッド	目的と使用法
`afterBegin`	このメソッドは、新しいトランザクションが開始されたことをインスタンスに通知する。EJB コンテナは、ビジネスメソッドを呼び出す直前に、このメソッドを呼び出す。このメソッドの中で、データベースからインスタンス変数に値を読み込むことができる
`beforeCompletion`	このメソッドは、ビジネスメソッドが完了したが、トランザクションはまだコミットされていないことをインスタンスに通知する。これが、セッション Bean がトランザクションをロールバックする最後の機会になる。データベースにインスタンス変数の値がまだ格納されていない場合は、このメソッドの本体にデータベースの更新を行うコードを記述することができる
`afterCompletion`	このメソッドは、トランザクションが完了したことをインスタンスに通知する。このメソッドはパラメータを 1 つ受け取る。Boolean 値 `true` はトランザクションがコミットされたことを意味し、`false` はトランザクションがロールバックされたことを意味する。トランザクションが失敗し、ロールバックされた場合は、このメソッドでセッション Bean のインスタンス変数をリフレッシュし、データベースから値を読み込み直すことができる

セッション Bean の完成

セッション Bean を完成させる手順は、選択した Bean の種類によって異なります。ここからは、次の作業のガイドラインを示します。

- 生成メソッドの完成
- ライフサイクルメソッドの完成
- ビジネスメソッドの追加
- トランザクションのコーディング

推奨する エンタープライズ Bean の開発手順

付録 A では、エンタープライズ Bean に変更を加える際の最適な方法や、推奨手順以外の方法で行った場合に発生する可能性のあるエラーや異常状態について説明しています。一般に、個々のクラスノードを操作するよりは論理ノードを操作する、メソッドの変更には Bean のプロパティシートや「カスタマイザダイアログ」を使用する、およびダイアログからでは操作できない Bean のコードを完成させたり、編集したりするには IDE のソースエディタを使用するといった方法をお勧めします。

生成メソッドの完成

ステートレス Bean には、生成メソッドは 1 つしかなく、このメソッドはパラメータを受け取りません。ステートレスセッション Bean では、ユーザーやクライアントに固有のデータは保持できません。

ステートフル Bean には、1 つ以上の生成メソッドを含めることができます。また、これらのメソッドはパラメータを受け取ることができます。

どちらの Bean の場合も、論理ノードで作業を行います。`create()` ノードを右クリックし、「開く」を選択して、ソースエディタで生成メソッドを開きます。このエディタを使用して、生成メソッドを完成させます。

ステートレス Bean の生成メソッドの完成

ステートレスセッション Bean では、生成メソッドはリソースに接続するのによく使われます。たとえば、このメソッドでリソースファクトリの参照を検出し、フィールドとして保存することができます。このようにすると、それ以降のメソッドの呼び出しで JDBC 接続を取得することができます。

ステートフル Bean の生成メソッドの完成

ステートフルセッション Bean では、生成メソッドのパラメータを使用してリソースファクトリの参照を検出したり、ユーザー名およびパスワードといったクライアント固有の情報を送出することができます (コード例 3-1 を参照)。このメソッドでは、後で使用する情報を保存することができます。コード例 3-1 の生成メソッドでは、`IdVerifier` というヘルパークラスを使用しています。

コード例 3-1　ステートフルセッション Bean の生成メソッド

```
public void ejbCreate(String userid, String pwd)
    throws CreateException {

    if (userid == null) {
        throw new CreateException("Please enter a user ID.");
    }
    else {
        this.userid = userid;
    }

    IdVerifier idChecker = new IdVerifier();
    if (idChecker.validate(pwd)) {
        this.pwd = pwd;
    }
```

コード例 3-1　ステートフルセッション Bean の生成メソッド (続き)

```
    else {
        throw new CreateException("Invalid password: " + pwd);
    }

    contents = new Vector();
}
```

ステートフル Bean への生成メソッドの追加

ステートフルセッション Bean に生成メソッドを追加するには、次の手順に従います。

1. Bean の論理ノードを選択し、右クリックし、「生成メソッドを追加」を選択します。

   「新規生成メソッドを追加」ダイアログが表示されます。

2. `create` で始まるメソッド名を入力し、必要に応じてパラメータと例外を追加して、「了解」をクリックします。

   Bean のホームインタフェースに生成メソッドのシグニチャが追加され、それに対応する `ejbCreate` メソッドが Bean クラスに追加されます。

3. ソースエディタでコードを完成させます。

   Bean の論理ノードに含まれている「クラス」ノードを展開し、「Bean クラス」を右クリックし、「開く」を選択します。

ライフサイクルメソッドの完成

EJB ビルダーは、4 つのライフサイクルメソッドを自動的に生成します。ステートレスセッション Bean では、生成されるライフサイクルメソッドを使用するだけで十分です。ステートフルセッション Bean では、2 つのメソッド、`ejbPassivate` と `ejbActivate` にコードを追加する必要があります。

たとえば、ステートフル Bean にシリアライズ不可能なフィールドが含まれていて、それらのフィールドが参照を置き換えることでシリアライズ可能になる場合があります。また、Bean の通信状態の中に、開かれているリソースが含まれていて、Bean のインスタンスが非活性化された場合に、コンテナがそれらのリソースを保持できない

場合もあります。どちらの場合も、ejbPassivate メソッドでシリアライズ不可能なフィールドを解放し、ejbActivate メソッドでこれらのフィールドを復元する必要があります。

ejbPassivate メソッドの完成

このメソッドで、インスタンスのフィールドをコンテナがシリアライズできるようにする必要があります。たとえば、コード例 3-2 に示すように、JDBC 接続をすべて閉じて、これらの接続を格納しているフィールドに null を代入する必要があります。

コード例 3-2 ejbPassivate メソッド

```
public void ejbPassivate() {

        try {
            con.close();
        } catch (Exception ex) {
            throw new EJBException("ejbPassivate Exception: " +
               ex.getMessage());
        } finally {
           con = null;
        }
    }
```

ejbActivate メソッドの完成

コード例 3-3 に示すように、このメソッドでインスタンスのフィールドを再び使用できるようにする必要があります。

コード例 3-3 ejbActivate メソッド

```
public void ejbActivate() {

        try {
            InitialContext ic = new InitialContext();
            DataSource ds = (DataSource) ic.lookup(dbName);
            con =  ds.getConnection();
        } catch (Exception ex) {
            throw new EJBException("ejbActivate Exception: " +
               ex.getMessage());
        }
    }
```

ビジネスメソッドの追加

セッション Bean には、クライアントのビジネスタスクを実行するためのビジネスメソッドを追加します。ビジネスメソッドでは、データベースにアクセスしたり、エンティティ Bean を管理したり、その持続フィールドを使用してデータベースエンティティを操作したりできます。

ステートフルセッション Bean にビジネスメソッドを追加するには、次の手順に従います。

1. Bean の論理ノードを選択し、右クリックし、「ビジネスメソッドを追加」を選択します。

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. メソッドに名前を付け、戻り値の型が適切かどうかを確認し、必要に応じてパラメータと例外を追加し、「了解」をクリックします。

   Bean のリモートインタフェースにビジネスメソッドのシグニチャが追加され、それに対応するメソッドが Bean クラスに追加されます。

3. ソースエディタでコードを完成させます。

   Bean の論理ノードに含まれている「クラス」ノードを展開し、「Bean クラス」を選択して右クリックし、「開く」を選択します。

   セッション Bean からデータベースにアクセスする場合は、Bean で使用する JDBC 呼び出しを削減し、システムリソースとネットワーク帯域幅を節約するために、データベースアクセスをデータアクセスオブジェクト (DAO) にカプセル化し、DAO で実際のデータ取得処理を実行できます。DAO を使用すると、セッション Bean のコードが単純でわかりやすくなり、特定のベンダーのツールやデータベースに依存しない Bean を作成できるようになります。

トランザクションのコーディング

トランザクションをコーディングする方法は、ステートフルセッション Bean とステートレスセッション Bean のどちらを使用するか、それらのセッション Bean で BMT と CMT のどちらを使用するかによって異なります。ここからは、トランザクション範囲の指定、ロールバック処理、セッション同期化インタフェースの使用についてのガイドラインを示します。

トランザクション範囲

トランザクション範囲は、セッション Bean の種類によって異なります。この違いを表 3-5 に示します。ステートフルセッション Bean で CMT を使用すると、柔軟性がより高くなります。

表 3-5　トランザクションとメソッドの関係

(自身のトランザクションを管理する) ステートレス BMT Bean では、トランザクションの範囲は 1 つのメソッドに限られる。	ステートフル BMT Bean では、トランザクションは同じセッション Bean の 1 つ以上のメソッドにわたって実行できる。
(トランザクションがコンテナによって管理される) ステートレス CMT Bean の場合、1 つ以上のメソッドにわたるトランザクションが実行できるが、それぞれのメソッドは異なるセッション Bean のものに限られる。	ステートフル CMT Bean の場合、トランザクションは同じセッション Bean の 1 つ以上のメソッドにわたって実行できる。

トランザクション範囲とロールバックの指定

ここでは、CMT Bean と BMT Bean の両方について、トランザクションの開始と終了をコーディングするためのガイドラインを示します。まず、次の 2 つの一般的な規則に注意してください。

- セッション Bean や JTA コードでは、入れ子になったトランザクションは取り扱えません。
- JDBC トランザクションと JTA トランザクションを併用しない方が、コードの保守が簡単です。一般には、JTA を使用した方が便利です。JTA には、JDBC といったほかのリソース用のトランザクションを含めることができるからです。

CMT Bean

CMT Bean では、EJB コンテナがすべてのトランザクション範囲を設定します。したがって、Bean 提供者は、トランザクションをいつ開始し、いつ終了するかを指定しません。通常、EJB コンテナはメソッドが開始される直前にトランザクションを開始し、メソッドが終了する直前にトランザクションをコミットします。

コンテナが設定したトランザクション範囲と競合する可能性があるメソッドは呼び出さないでください。問題のあるメソッドを次に示します。

- `java.sql.Connection` の `commit` メソッド、`setAutoCommit` メソッド、ロールバックメソッド
- `javax.ejb.EJBContext` の `getUserTransaction` メソッド
- `javax.transaction.UserTransaction` のすべてのメソッド

セッション Bean は、次の 2 通りの方法でコンテナ管理によるトランザクションをロールバックすることができます。

- システム例外がスローされると、コンテナは該当するトランザクションを自動的にロールバックします。
- `javax.ejb.EJBContext` の `setRollBackOnly` メソッドを呼び出して、アプリケーション例外がスローされているかどうかに関係なく、コンテナにトランザクションをロールバックさせます。

BMT Bean

BMT Bean では、トランザクションの開始と終了を明示的にコーディングする必要があります。トランザクション範囲を明示的に指定するには、`javax.transaction.UserTransaction` インタフェースを使用します。JTA インタフェースを使用した場合のコード例を次に示します。

```
UserTransaction ut = ejbContext.getUserTransaction();
    ut.begin();
    // ここでトランザクション処理を実行
    ut.commit();
```

トランザクションで指定した更新が保存された場合は、トランザクションがコミットされます。トランザクションが失敗した場合は、トランザクションがロールバックされ、そのトランザクションに含まれている文の効果がすべて取り消されます。BMT を使用するセッション Bean では、`getRollbackOnly` メソッドや `setRollbackOnly` メソッドでロールバックを行わないでください。この 2 つのメソッドは CMT Bean からしか使用できません。

セッション同期化の使用

ステートフル CMT セッション Bean では、セッション同期化インタフェースを使用することができます。このインタフェースを使用すると、トランザクションでキャッシュされたデータベースデータを、より綿密に制御することができます。

このインタフェースは、EJB コンテナがトランザクションを開始、コミット、またはロールバックする前に呼び出すコールバックメソッドを提供します。このインタフェースを使用すると、トランザクションの特定の時点で、セッション Bean のインスタンス変数が、データベース中の対応する値に自動的に同期化されます。セッション Bean は、トランザクションが完了する前であれば、インスタンス変数の値をロールバックすることができます。

- `afterBegin`。コンテナは、トランザクションの最初のビジネスメソッドが開始される前に、セッション Bean の `afterBegin` メソッドを呼び出します。Bean 提供者は、このメソッドにコードを記述して、該当するトランザクションのスコープでインスタンスに必要なデータベース処理を実行することができます。

- `beforeCompletion`。コンテナは、セッション Bean のクライアントが現在のトランザクションの処理を完了したときに (ただし、インスタンスによって使用されていたリソースマネージャがコミットされる前に)、`beforeCompletion` メソッドを呼び出します。Bean 提供者は、このメソッドにコードを記述して、Bean によってキャッシュされていたデータベース更新情報を書き出すことができます。さらに、セッションコンテキストの `setReadbackOnly` メソッドを呼び出して、トランザクションをロールバックさせることもできます。

- `afterCompletion`。コンテナは、このメソッドを呼び出して、現在のトランザクションが完了したことを通知します。トランザクションがコミットされた場合は状態 True が渡され、トランザクションがロールバックされた場合は状態 False が渡されます。Bean 提供者は、このメソッドにコードを記述して、トランザクションがロールバックされた場合にインスタンスの状態をリセットすることができます。

セッション Bean にセッション同期化インタフェースを追加するには、ウィザードで次のように設定します。

1. セッション EJB ウィザードの最初の区画にある「状態」から「ステートフル」を選択します。
2. ウィザードの次の区画で「セッション同期化 (Session Synchronization) インタフェースの実装」を選択します。

   このように設定すると、コード例 3-4 のようなコードがセッション Bean クラスに挿入されます。この例では、`afterBegin` メソッドに `checkingBalance` 変数と `savingBalance` 変数が読み込まれています。

コード例 3-4　afterBegin メソッドの例

```
public void afterBegin() {
    System.out.println("afterBegin()");
    try {
        checkingBalance = selectChecking();
        savingBalance = selectSaving();
    } catch (SQLException ex) {
        throw new EJBException("afterBegin Exception: " +
            ex.getMessage());
    }
}
```

afterCompletion メソッドの例をコード例 3-5 に示します。このメソッドを使用すると、トランザクションが失敗し、ロールバックされた場合に、セッション Bean の口座残高フィールドに、データベースから値を読み込み直すことができます。

コード例 3-5　afterCompletion メソッドの例

```
public void afterCompletion(boolean committed) {
    System.out.println("afterCompletion: " + committed);
    if (committed == false) {
        try {
            checkingBalance = selectChecking();
            savingBalance = selectSaving();
        } catch (SQLException ex) {
            throw new EJBException("afterCompletion SQLException: " +
                ex.getMessage());
        }
    }
}
```

セッション Bean を作成した後の作業

作成したセッション Bean を、最終環境で使用できるようにする必要があります。配備記述子、プロパティシートの使用法といった、モジュールのアセンブルとアプリケーションの配備についての情報は、第 8 章を参照してください。

また、付録 Aでは、作成した エンタープライズ Bean の望ましい操作手順を説明しています。

詳細情報の参照先

エンタープライズ Bean は、非常に高機能で、高い柔軟性を備えたアプリケーションの構成要素になります。エンタープライズ Bean の基本要素の作成は、特に Sun ONE Studio IDE のようなツールを使用すれば非常に簡単です。しかし、アプリケーションのニーズを満たすように Bean を完成させることは、場合によっては非常に複雑です。詳細については、次の Web サイトにある「Enterprise JavaBeans 仕様 2.0」を参照してください。

`http://java.sun.com/products/ejb/docs.html`

第4章

CMP エンティティ Bean の開発

Sun ONE Studio IDE の EJB ビルダーを使用すると、J2EE アプリケーションでデータを表すのに必要なエンティティ Bean をプログラムすることができます。この章では、コンテナ管理による持続性が設定された個々のエンティティ Bean (CMP エンティティ Bean) を開発する方法について説明します。

IDE では、Enterprise JavaBeans コンポーネント (エンタープライズ Bean) に必要なクラスを作成するためのウィザードが用意されています。作成するコンポーネントには Bean クラス、インタフェース (ローカル、リモート、または両方)、および場合に応じて主キーのクラスがあります。ほとんどの処理は自動化されています。

エンティティ Bean をプログラムする場合、この章で説明する以外の方法を選択することもできます。Sun ONE Studio IDE は、コーディング処理のほとんどを自動化していますが、プログラマが自身で多くを決定できるように柔軟な手段も提供しています。詳細については、xviii ページの「お読みになる前に」で紹介しているマニュアルを参照するか、市販の エンタープライズ Bean のプログラミングに関する文献を参照してください。

CMP エンティティ Bean 作成のための EJB ビルダーの使用

EJB ビルダーは、ウィザード、プロパティシート、およびエディタから構成されています。これらの機能を使用して、エンタープライズ Bean を整合性のとれた方法で簡単に作成することができます。EJB ビルダーがインストールされているかどうかを確認するには、メインウィンドウから「ツール」 > 「オプション」 > 「IDE 構成」 > 「システム」 > 「モジュール」 > 「J2EE サポート」を選択してください。モジュー

ルのリストに「EJB 2.0 ビルダー」が表示されていて、プロパティシートの「使用可能」プロパティが True に設定されていれば、EJB ビルダーは使用できる状態になっています。

Sun ONE Studio IDE でエンティティ Bean を作成するにはいくつかの方法がありますが、この章で推奨している方法を取ることで、最もわかりやすく、かつ、短期間で Bean を完成させることができます。ここで説明する操作方法は、Bean の一貫性と J2EE 標準への準拠を確実にするために IDE の機能を最大限に活用した手法となっています。

最良の結果を得るために、EJB ビルダーを使用して、次の手順でエンティティ Bean をプログラミングしてください。

- **エンティティ Bean と、エンティティ Bean に必要なクラスを作成します。**EJB ビルダーのウィザードの手順に従うと、セッション Bean の枠組みができあがります。その Bean の 3 つ または 4 つの必要なクラスと論理ノードは、エクスプローラの「ファイルシステム」タブに表示されます。ウィザードは、これらのうち 2 つのクラスであるホームインタフェースとリモートインタフェースの宣言を生成します。生成された Bean クラスには必要なメソッドの宣言とプログラマの指定した持続フィールドが含まれます。

  エンティティ Bean は、論理ノードから操作するのが最適です。エクスプローラでは、すべての論理ノードは として表示されます。

- **メソッド、パラメータ、および例外を追加します。**この章で後述する手順に従って、IDE の GUI 機能を使用します。コンテキストメニューから表示できるダイアログを使用するか、必要なクラスのセットを直接編集することで、Bean にメソッドを追加することができます。

- **エンティティ Bean の配備記述子を設定します。**論理ノードで使用できるエンティティ Bean のプロパティシートを使用して、プロパティを編集します。このプロパティシートは、論理ノードから表示させることができます。

エンティティ Bean の論理ノードからは、Bean の検査、クラスへのメソッドの追加、フィールドの追加、Bean の配備関連プロパティの指定、アプリケーションに Bean を配備する EJB モジュールの作成が実行できます。また、Bean に対して IDE のテスト機能を使用することもできます。

CMP および BMP エンティティ Bean の比較

エンティティ Bean を作成する前に、CMP を利用するのか、BMP を利用するのかを検討します。IDE の EJB ビルダーはどちらのエンティティ Bean にも対応していますが、これら 2 種類の Bean は異なる手順で作成します。どちらのエンティティ Bean を使用するのかについては第 2 章で説明しています。この章で検討する設計時の考慮点を表 4-1 に挙げます。

表 4-1　CMPエンティティ Bean とBMP エンティティ Bean の選択

項目	CMP	BMP
データベースとの関連	CMP エンティティ Bean は、自身とデータベースとの関係の管理をコンテナに依存するが、特定のデータストアには依存しない。	BMP エンティティ Bean は特定のデータベースとの関係を自身で管理する。
持続性	持続性を保つためにコンテナがデータベースアクセスとアプリケーション中の各 CMP エンティティ Bean を管理する。Bean コードには、データベースへの呼び出しは含まれない。Bean の持続状態は仮想持続フィールドによって表現される。	BMP エンティティ Bean には、指定されたデータベースへ自身を接続するためのすべてのコードが含まれる。(インスタンス変数として記述されている) 持続データを持つ BMP エンティティ Bean には、データベースへの呼び出しもすべて記述されている必要がある。SQL コードはすべて手動で追加する。EJB コンテナがデータストアへの適切な持続性をマッピングできない場合は、プログラマが BMP エンティティ Bean を作成する。

表 4-1　CMPエンティティ Bean とBMP エンティティ Bean の選択 (続き)

項目	CMP	BMP
処理	CMP エンティティ Bean の基本構造 (デフォルトのクラス) は、BMP エンティティ Bean に比べ、簡単に早く作成できる。コードも少なくて済む。	BMP エンティティ Bean に必要なコードは CMP エンティティ Bean に比べ多い。経験の豊かな JDBC プログラマには魅力とも考えられる。
対象範囲	単一の CMP エンティティ Bean は、通常、1 つの表だけを表すが、Bean は複数の表にマップできる。	BMP エンティティ Bean は、手動でコーディングすることで 1 つ以上の表として表現できる。
能力と柔軟性	CMP エンティティ Bean は、データベースへのアクセスをコンテナに依存するが、Bean そのものは異なるデータベース環境に配備できる。	個々の BMP エンティティ Bean のデータベースへのアクセスは手動でプログラムする必要がある。BMP エンティティ Bean はプログラムされた環境でだけ動作する。

以降では、CMP エンティティ Bean の作成方法と開発中の留意点について説明します。BMP エンティティ Bean の作成手順については第 6 章を参照してください。

関係 CMP エンティティ Bean のセットの作成

J2EE アプリケーションの多くには、関係 CMP エンティティ Bean が含まれています。関係 CMP エンティティ Bean とは、コンテナ管理による関係 (CMR) によって表現される関係を互いに持った 2 つの CMP エンティティ Bean を指します。この関係は、2 つのエンティティまたは 2 つの表が関連する列を含む場合のデータベースやデータベーススキーマに類似しています。たとえば、スキーマが `Customer`、`Order`、`LineItem`、および `Part` という表を含んでいたとします。この場合、`Order` は `Customer` の外部キーを持ち、`LineItem` は `Order` への外部キーと、`Part` への外部キーを持つことになります。

IDE を使用すると、関係 CMP エンティティ Bean のセットを 1 度にまとめて簡単に作成できます。EJB ビルダーのウィザードを使用して、関連したデータベースのエンティティを操作するための CMP エンティティ Bean のセットを生成すると、これらのエンティティの関係は作成された CMP エンティティ Bean の中に再生成されます。こ

れを利用して、Bean 間に別の関係を指定できるようになります。このような関係は、CMP エンティティ Bean 中で CMR フィールドして表現されます。これらのフィールドを編集して、濃度、種類、およびカスケード削除機能を指定できます。

関係 CMP エンティティ Bean のセットを作成したい場合、また、2 つのエンティティ Bean の間の外部キーの関係を維持したい場合は、第 5 章を参照してください。

CMP エンティティ Bean の定義

EJB ビルダーのウィザードを使用すると、CMP エンティティ Bean に必要最低限のクラスである Bean クラスと、選択したインタフェース (ローカルかリモート、または両方) を自動的に作成できます。複合主キーを指定した場合、または作成する表に複合主キーが必要な場合は、ウィザードによって主キークラスが自動的に作成されます。CMP エンティティ Bean を定義するには、次の操作を実行してください。

1. Bean を格納するパッケージを作成または選択します。
2. EJB ビルダーのウィザードを使用して CMP エンティティ Bean のインフラストラクチャを作成します。
3. 必要に応じて、生成メソッド、ビジネスメソッド、検索メソッド、選択メソッド、およびホームメソッドを Bean に追加します。
4. 追加したメソッドの本体を完成させます。
5. 必要であれば、主キークラスを追加します。

以上の基本的な操作を、次に詳しく説明します。

パッケージの作成

エンティティ Bean を格納するためのパッケージを作成するには、ファイルシステムを右クリックし、「新規」->「Java パッケージ」を選択します。

データソースの準備

CMP エンティティ Bean はデータベースの実際の表に基づいていて、Bean の持続フィールドは表の列と対応しています。 EJB ビルダーのウィザードを使用すると、実際のデータベース接続またはデータベーススキーマオブジェクト (データベースのスナップショット) から表を収集できます。また、ウィザードの特定の区画を使用すると、表の列を Bean の持続フィールドとして手動で指定できます。その後の配備時には、それらのフィールドはデータベースの列に対応付けられます。

列とフィールドを対応付ける処理方法は EJB コンテナによって異なります。以降では、IDE に含まれる PointBase データベースサーバーを例に説明しています。PointBase データベースサーバー以外のサーバーを IDE と共に使用している場合の対応付けの処理については、それらのサーバーのマニュアルを参照してください。

使用するデータソースの種類を決定する際は、次の点を考慮してください。

- **稼働中のデータベース接続** - CMP エンティティ Bean を稼働中のデータベースから作成することを検討している場合は、稼動中のデータベースサーバーに接続している必要があります。データベースへの接続は EJB ビルダーのウィザードの開始前でも開始後でも構いません。次に示す手順では、ウィザードの開始前にサーバーを起動する場合を説明しています。ウィザードでサーバーを起動する手順については、86 ページの「データベース接続からの表の選択」で説明しています。

  PointBase データベース以外のデータベースを使用している場合は、Forte for Java のインストールディレクトリ配下の `lib/ext` ディレクトリにそのデータベースのドライバファイルが含まれていることを確認してください。新規スキーマの作成時には、必ずこのことを確認して正しいデータベースドライバを選択してください。エクスプローラでドライバファイルをマウントしたり、`CLASSPATH` 環境変数にドライバファイルを指定したりすることはできません。

  起動 - PointBase データベースを開始するには、メインウィンドウから「ツール」->「PointBase ネットワークサーバー」->「サーバーを起動」を選択します。

  接続 - 稼動中の PointBase データベースに接続するには、エクスプローラの「実行時」タブを表示します。「Databases」ノードを展開します。ラベルが `jdbc:pointbase` で始まり、2 つに裂かれているように見えるアイコンを持つノードを選択します。選択したノードを右クリックして「接続」を選択します。PointBase アイコンが 1 つにまとまり、Tables、Views、Processes がサブノードとして表示されます。

この方法が、データベースへの接続には適しています。特に複数のエンティティ Bean を作成するのに最適です。EJB ビルダーウィザードを開始する前にデータベースに接続しないで、ウィザードの中からデータベースに接続することもできますが、この場合はエンティティ Bean を作成するたびにデータベースに再接続する必要があります。

- **データベーススキーマ** - データベーススキーマの表から Bean を作成する場合は、IDE のエクスプローラウィンドウにある「ファイルシステム」タブからスキーマにアクセスできるようになっている必要があります。

EJB ビルダーのウィザードの開始

単一 CMP エンティティ Bean を作成するには、次の操作を実行します。

1. IDE のメインウィンドウで「表示」->「エクスプローラ」を選択し、エクスプローラウィンドウを表示します。
2. エクスプローラの「ファイルシステム」タブで、CMP エンティティ Bean の格納先となる Java パッケージを選択します。
3. 右クリックして、「新規」->「J2EE」->「CMP エンティティ EJB」を選択します。

   EJB ビルダーのウィザードが表示されます。左側のパネルに、現在の手順と、CMP エンティティ Bean の作成を終えるまでの一連の手順が表示されます。

CMP エンティティ Bean のインフラストラクチャの作成

図 4-1 に示す EJB ビルダーの「CMP エンティティ Bean 名とプロパティ」区画で、CMP エンティティ Bean の名前を指定します。この区画では、Bean がどこから持続フィールドを取得するのかと、その Bean が持つインタフェースの種類も指定します。必要であれば、Bean の格納先であるパッケージの変更もできます。

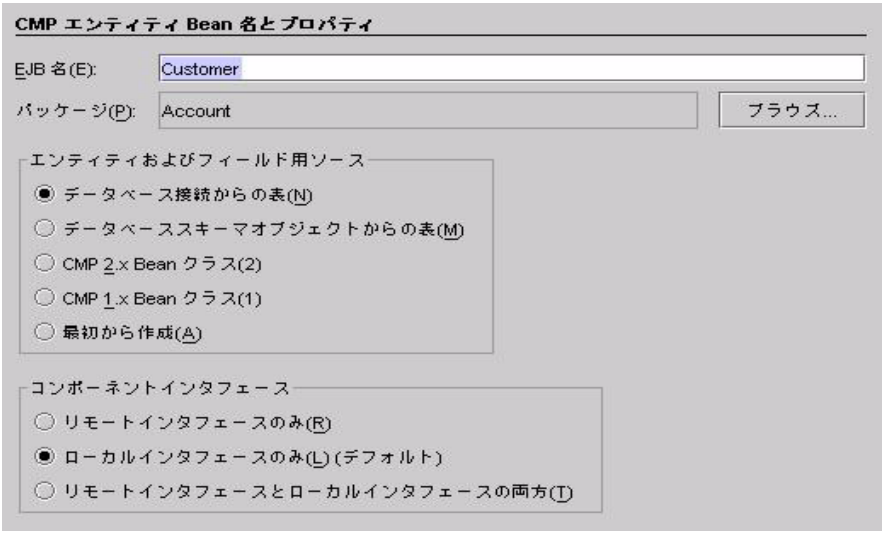


図 4-1　CMP エンティティ Bean 作成時の EJB ビルダーウィザードでの選択項目

各選択項目の説明と、対応する操作の参照先を次の表に示します。

「エンティティおよびフィールド用ソース」と表示されたラジオボタンボックスでは、次の項目を選択できます。

データベース接続からの表	CMP エンティティ Bean が既存のデータベースの表を表現する場合に選択する	86 ページの「データベース接続からの表の選択」を参照
データベーススキーマオブジェクトからの表	使用できるデータベーススキーマがあり、動作中のデータベースに接続したくない場合に選択する	89 ページの「データベーススキーマオブジェクトからの表の選択」を参照

CMP 2.x Bean クラス	CMP エンティティ Bean が、EJB 2.0 仕様に準じる既存の Bean クラスを元にしている場合に選択する	90 ページの「CMP 2.x Bean クラスの使用」を参照
CMP 1.x Bean クラス	CMP エンティティ Bean が、EJB 1.x 仕様に準じる既存の Bean クラスを元にしている場合に選択する	92 ページの「CMP 1.x Bean クラスの使用」を参照
最初から作成	すべての CMP フィールドをプログラマが指定する場合に選択する	93 ページの「Bean の持続フィールドの新規作成」を参照

「コンポーネントインタフェース」と表示されたラジオボタンボックスでは、次の項目を選択できます。

リモートインタフェースのみ	作成する CMP エンティティ Bean のメソッドに対し、外部クライアントからの呼び出しはあっても、内部のローカルクライアントからの呼び出しがない場合に選択する。
ローカルインタフェースのみ (デフォルト)	作成する CMP エンティティ Bean のメソッドに対し、ローカルインタフェースを介した呼び出しはあっても、外部クライアントからの直接呼出しがない場合に指定する。
リモートインターフェース とローカルインタフェースの両方	作成する CMP エンティティ Bean のメソッドに対し、外部と内部のローカルクライアント (他の Bean である可能性もある) の両方から呼び出しがある場合に選択する。

必要な項目を選択して「次へ」をクリックすると、ウィザードによって次の操作が表示されます。この操作について、次の節で説明します。

データベースの表からの持続フィールドの指定

データベースからの持続フィールドの指定を選択する場合は、稼動中のデータベースにすでに接続しているか、既存のデータベーススキーマが使用できる状態になっている必要があります。詳細については、82 ページの「データソースの準備」および 88 ページの「データベーススキーマの収集」を参照してください。EJB ビルダーウィザードは、データベース (「データベース接続からの表」を選択した場合) 、またはス

キーマ (「データベーススキーマオブジェクトからの表」を選択した場合) の表の列を対応付けて、CMP エンティティ Bean の持続フィールドを作成します。どちらを選択しても、作成された CMP エンティティ Bean の内容は同じとなります。

ほとんどのアプリケーションサーバーでは、Bean の配備時に CMP フィールドとデータベースの列とを対応付けることになります。その際、サーバーは、そのサーバープロセス内でのマッピングを表す SQL 文を動的に生成します。J2EE RI サーバーでは、このマッピングを表す SQL 文は静的に作成されます。このため、マッピングに対して変更を加えたい場合は、生成された SQL 文を直接編集する必要があります。この編集操作については第 8 章で説明しています。

データベース接続からの表の選択

データベースへ直接アクセスでき、データベースのユーザー間での競合が問題にならない場合は、データベースへの直接接続を選択できます (データベースの起動・接続については、82 ページの「データソースの準備」を参照してください)。

ウィザードの「データベース接続からの表」区画が表示されていることを確認してください。エンティティ Bean を接続できるデータベースが、ウィザードの区画内にツリーで表示されているはずです。ここで、次の操作を実行します。

1. データベースを選択します。

   データベースの状態に応じて、次の操作のうち、どちらかを実行してください。

   - **データベースはインストールされているが、接続の定義がない。**データベースがインストールされていても、接続の定義がない場合は、「接続を追加」をクリックします。表示された「データベースの新規接続」ダイアログで、次の操作を実行します。

   a. 「名前」コンボボックスからデータベースを選択します。

   b. 「ドライバ」フィールドでパスが正しいかどうかを確認します。

   c. 「データベース URL」フィールドに必要な情報を指定します。

   d. データベースへのアクセスにユーザー名とパスワードが必要であれば、指定します。

   e. 必要に応じて「セッション中はパスワードを保存」ボックスにチェックをつけます。

f. 「了解」をクリックします。ウィザードの「データベース接続からの表」区画にある「次へ」をクリックします。

データベースへ接続します。

- **データベースがインストールされていて、接続の定義までが済んでいる。**データベースがインストールされていて接続も定義済みだが、有効になっていない場合は、データベースのアイコンが 2 つに分かれているように表示されます。次の操作を実行してください。

a. データベースを選択して、「データベースに接続」をクリックします。

2 つに分かれたアイコンが 1 つにまとまります。

b. データベースのノードを展開して、「Tables」ノードを表示させます。

「接続できません」というエラーメッセージが表示された場合は、データベースが起動しているかどうかを確認してください。

- **データベースがインストールされていて、接続も有効になっている。**データベースへの接続が定義済みで有効になっている場合 (同じ接続を使って 2 つ目のエンティティ Bean を作成している場合など)、アイコンは 2 つに分かれていないで、1 つにまとまって表示されます。この場合に必要な操作は、データベースのノードを展開して、「Tables」ノードを表示させることだけです。

2. 選択したデータベースの階層を下がっていって、Bean に対応付けたい表のノードを見つけます。該当する表を選択し、「次へ」をクリックします。

「CMP フィールド」区画が表示されます。この区画では、データベース表の列とそれに対応するフィールドが表示されます。このフィールドは、 EJB ビルダーのウィザードがこれから CMP エンティティ Bean 内に作成するフィールドです。ウィザードはこれらのデータベース中の列と Bean の持続フィールドを対応付けます。

3. Java フィールド名と型を確認し、必要に応じて変更を加えます。

Java フィールドには、IDE によってデフォルトの名前と型が割り当てられます。これらの名前とタイプは必要に応じて変更できます。変更する際は、該当するフィールドを選択し、「編集」ボタンをクリックしてほかに指定できる型の中から選択します。

この操作の詳細についてはマニュアル「JDBC API ご使用にあたって」の第 8 章「SQL と Java タイプのマッピング」を参照してください。このマニュアルは次の URL から入手できます。

```
http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/
GettingStartedTOC.fm.html
```

4. 「次へ」をクリックします (ウィザードが割り当てたデフォルトのクラスの確認や変更が必要ない場合は、この手順を省いて「完了」をクリックします)。

   「CMP エンティティ Bean クラスファイル」区画が表示されます。区画には、エンティティ Bean のインフラストラクチャの要素である Bean クラス、選択したインタフェース (ローカルかリモート、または両方)、および主キークラスの型が表示されます。

   この区画では、Bean のクラスの確認または変更ができます。必要に応じて、他の Bean クラス、インタフェース、または主キークラスを指定できます。各項目の変更ボタンをクリックすると、そのクラスやインタフェースを格納したり、取り出したりするパッケージを指定できるようになります。

   - たとえば、関連する 1 つ以上のオブジェクトに対するパッケージ名を変更することで、Bean クラスはあるディレクトリに保存し、ホームインタフェースとリモートインタフェースは別のディレクトリに保存する、といった処理ができます。

     ただし、これを実行する前には、利用するアプリケーションがファイルの分散保存をサポートしているかどうかを確認する必要があります。

   - 指定した既存のクラスまたはインタフェースが、必要とされるメソッドや例外を持っていない場合は、エラーメッセージが表示されます。

   - パッケージおよびディレクトリ名には、有効な Java 識別子を使用してください。

5. 「完了」をクリックします。

   EJB ビルダーによって、CMP エンティティ Bean のインフラストラクチャが自動的に生成されます。これについての詳細は、95 ページの「CMP エンティティ Bean のクラス」を参照してください。

データベーススキーマの収集

表に対応する CMP エンティティ Bean を、直接データベースに接続しないで、データベーススキーマから作成することもできます。スキーマがまだない場合は、IDE のデータベーススキーマウィザードを使用してスキーマを作成できます。最初にデータベースを起動・接続する必要がある場合は、82 ページの「データソースの準備」を参照してください。その後、次の操作を実行します。

1. Sun ONE Studio IDE を使用して、次のどちらかの方法でデータベーススキーマウィザードを開きます。

- エクスプローラでパッケージノードを選択します。右クリックして「新規」->「データベース」->「データベーススキーマ」を選択します。
- メインウィンドウから「ツール」->「データベーススキーマの収集」を選択します。

2. ウィザードの表示に従い、使用するデータベースを指定し、スキーマに含める表を選択します。

   選択した表とビューが進捗ダイアログの進捗バーに表示されます。

データベーススキーマオブジェクトからの表の選択

データベースへのアクセスが制限されてはいるが、スキーマオブジェクトは使用できるといった場合、スキーマからの表を利用して CMP エンティティ Bean を作成できます (スキーマがない場合は作成する必要があります。前述の節を参照してください) 。

ウィザードの最初の区画で「データベーススキーマオブジェクトからの表」を選択すると、「データベーススキーマオブジェクトからの表」区画が表示されます。この区画には、エクスプローラの「ファイルシステム」タブでマウントされたディレクトリが表示されています。次の操作を実行してください。

1. Bean に対応付けたい表が含まれているデータベーススキーマを見つけます。

   選択したスキーマの階層を下っていって、Bean に対応付けたい表のノードを見つけます。

2. スキーマのノードを展開して、使用したい表を見つけます。表が見つかったら選択します。

   「次へ」と「完了」ボタンが有効になります。

3. 「次へ」をクリックすると、Bean の持続フィールドに対応付けられるデータベース列が表示されます (ウィザードに Bean のインフラストラクチャを自動生成させたい場合は、「完了」をクリックしてこの手順と次の手順を省くこともできます)。

   「CMP フィールド」区画が表示されます。この区画では、データベース表の列とそれに対応するフィールドが表示されます。このフィールドは、 EJB ビルダーのウィザードがこれから CMP エンティティ Bean 内に作成するフィールドです。必要に応じてフィールド名や型の変更もできます。フィールドを選択して「編集」ボタンをクリックすると、ほかに指定できる型が表示されます。

4. 「次へ」をクリックして、ウィザードが割り当てたデフォルトのクラスを確認または変更します。この手順を省きたい場合は、「完了」をクリックするとウィザードによって Bean のインフラストラクチャが自動的に生成されます。

   ウィザードの「CMP エンティティ Bean クラスファイル」区画が表示されます。区画には、エンティティ Bean のインフラストラクチャの要素である Bean クラス、選択したインタフェース (ローカルかリモート、または両方)、および主キークラスの種類が表示されます。

   この区画では、Bean のクラスの確認または変更ができます。必要に応じて、他の Bean クラス、インタフェース、または主キークラスを指定できます。各項目の変更ボタンをクリックすると、そのクラスやインタフェースを格納したり、取り出したりするパッケージを指定できるようになります。

   - たとえば、関連する 1 つ以上のオブジェクトに対するパッケージ名を変更することで、Bean クラスはあるディレクトリに保存し、ホームインタフェースとリモートインタフェースは別のディレクトリに保存する、といった処理ができます。

     ただし、これを実行する前には、利用するアプリケーションがファイルの分散保存をサポートしているかどうかを確認する必要があります。

   - 指定した既存のクラスまたはインタフェースが、必要とされるメソッドや例外を持っていない場合は、エラーメッセージが表示されます。

   - パッケージおよびディレクトリ名には、有効な Java 識別子を使用してください。

5. 「完了」 をクリックします。

   EJB ビルダーによって、CMP エンティティ Bean のインフラストラクチャが自動的に生成されます。これについての詳細は、95 ページの「CMP エンティティ Bean のクラス」を参照してください。

CMP 2.x Bean クラスの使用

EJB 2.0 環境で作成された既存の CMP エンティティ Bean をもとに、新しい CMP エンティティ Bean を作成することができます。ウィザードの「 CMP エンティティ Bean 名とプロパティ」区画で、「CMP 2.x Bean クラス」を選択し、「次へ」をクリックします。以下の手順に従って、表示されるナビゲーションリストから使用する Bean クラスを選択します。

「CMP 2.x Bean クラスを選択」区画が表示されたら、以下のようにします。

1. 使用する Bean クラスを選択します。

   選択できる Bean クラスのみが表示され、Bean の他の要素については表示されません。Bean クラスを選択すると、「次へ」ボタンが選択可能になります。

2. 「次へ」をクリックします。

   選択した Bean クラスのフィールドが表示されます。元のCMP エンティティ Bean が主キーを保持している場合でも、この区画で 1 つ以上のフィールドを主キーとして指定する必要があります。

3. 主キーとなるフィールドを選択し、「編集」をクリックします。

   「編集 持続フィールド」ダイアログが表示されます。

4. 「主キー」チェックボックスをチェックするなどの必要な変更を行い、「了解」をクリックします。

   「CMP フィールド」区画で、主キーとして編集したフィールドが表示されます。

   Bean に複数の主キーを設定する場合には、必要に応じて他のフィールドに対しても、手順 3 と手順 4 を繰り返します。

   この区画では、既存のフィールドを編集のみ可能です。フィールドを削除したり、新しいフィールドを追加することはできません。

5. 「次へ」をクリックします。

   「CMP エンティティ Bean クラスファイル」区画で、作成中の CMP エンティティ Bean の要素が一覧に表示されます。

   Bean が別のインターフェースを使用する必要がある場合、「インターフェースを変更...」ボタンから指定します。

   Bean が異なる主キークラスを使用する場合、新規または既存の主キーのどちらであっても、「クラスを変更... 」ボタンから指定します。

6. 設定を完了したら、「完了」をクリックします。

   EJB ビルダーによって、CMP エンティティ Bean の構造が自動的に生成されます。更に詳しい内容については、95 ページの「CMP エンティティ Bean のクラス」を参照してください。

CMP 1.x Bean クラスの使用

EJB 1.0 環境で作成された既存の CMP エンティティ Bean をもとに、新しい CMP エンティティ Bean を作成することができます。このオプションを選択すると、作成される Bean は CMP バージョン 1.x の エンタープライズ JavaBean バージョン 2.0 となり、ローカルとローカルホームのインターフェースなどの EJB 2.0 機能をサポートできません。

ウィザードの「 CMP エンティティ Bean 名とプロパティ」区画で、「CMP 1.x Bean クラス」を選択します (この選択をすると、リモートインターフェースのみが選択可能になります)。「次へ」をクリックします。以下の手順に従って、表示されるナビゲーションリストから使用する Bean クラスを選択します。

「CMP 1.x Bean クラスを選択」区画が表示されたら、以下のようにします。

1. 使用する Bean クラスを選択します。

   選択できる Bean クラスのみが表示され、Bean の他の要素については表示されません。Bean クラスを選択すると、「次へ」ボタンが選択可能になります。

2. 「次へ」をクリックします。

   選択した Bean クラスのフィールドが表示されます。元のCMP エンティティ Bean が主キーを保持している場合でも、この区画で 1 つ以上のフィールドを主キーとして指定する必要があります。

3. 主キーとなるフィールドを選択し、「編集」をクリックします。

   「編集 持続フィールド」ダイアログが表示されます。

4. 「主キー」チェックボックスをチェックするなどの必要な変更を行い、「了解」をクリックします。

   「CMP フィールド」区画で、主キーとして編集したフィールドが表示されます。

   Bean に複数の主キーを設定する場合には、必要に応じて他のフィールドに対しても、手順 3 と手順 4 を繰り返します。

   この区画では、既存のフィールドを編集のみ可能です。フィールドを削除したり、新しいフィールドを追加することはできません。

5. 「次へ」をクリックします。

   「CMP エンティティ Bean クラスファイル」区画で、作成中の CMP エンティティ Bean の要素が一覧に表示されます。

Bean が別のインターフェースを使用する必要がある場合、「インターフェースを変更...」ボタンから指定します。

Bean が異なる主キークラスを使用する場合、新規または既存の主キーのどちらであっても、「クラスを変更... 」ボタンから指定します。

6. 設定を完了したら、「完了」をクリックします。

EJB ビルダーによって、CMP エンティティ Bean の構造が自動的に生成されます。更に詳しい内容については、95 ページの「CMP エンティティ Bean のクラス」を参照してください。

Bean の持続フィールドの新規作成

EJB ビルダーの「CMP エンティティ Bean 名とプロパティ」区画で「最初から作成」を選択する理由として、データベースが作成されていない、アクセスが許可されていない、またはデータベースがどこにあるかわからないといったことが考えられます。また、エンタープライズ Bean を含んだアプリケーションが配備されたときに、アプリケーションサーバーにデータベースを作成させたい、といったこともあります。

CMP エンティティ Bean のコンテナによって、Bean とデータベースの対応付けが要求されることもありますが、これはアセンブルと配備の段階になるまでは要求されません。「最初から作成」オプションを選択した場合、指定したフィールドは配備記述子には「持続」として記録され、後に、そのフィールドが特定のデータベーススキーマに対応していることがコンテナに通知されます。この対応付けは、J2EE アプリケーションが配備される直前に実行されます。

IDE では、Bean の Java フィールド名を指定することで、エンティティ Bean の接続を設定することができます。残りのデータベース接続に関する情報は、配備の実行中に指定できます。

ウィザードの「CMP エンティティ Bean 名とプロパティ」区画で次の操作を実行します。

1. 「EJB 名」フィールドで Bean の名前を指定します。
2. 表示された以外の場所に Bean を保存したい場合は、「ブラウズ」ボタンを使って既存の Java パッケージを選択します。
3. 「最初から作成」を選択し、「次へ」をクリックします。

「CMP フィールド」区画が表示されます。Bean には、ウィザードによって自動的に `defaultField` という名前を持つデフォルトの CMP フィールドが作成されます。

4. CMP フィールドの名前を指定したい場合は、デフォルトのフィールドを選択して「編集」をクリックします。

   次のガイドラインに従って、フィールドに名前を付けて定義します。

   - 通常は、最少でも 1 つの主キーフィールドを指定します。ただし、CMP エンティティ Bean の主キーは省略できます。
   - 指定したいフィールドの型がコンボボックスに表示されない場合は、別の型を指定できます。型には、`java.lang.Integer` などの有効なパス名を指定してください。

5. 「追加」をクリックして、追加する持続フィールドを定義します。
6. 「次へ」をクリックして、ウィザードが割り当てたデフォルトのクラスを確認または変更します。この手順を省きたい場合は、「完了」をクリックするとウィザードによって Bean のインフラストラクチャが自動的に生成されます。

   ウィザードの「CMP エンティティ Bean クラスファイル」区画が表示されます。区画には、エンティティ Bean のインフラストラクチャの要素である Bean クラス、選択したインタフェース (ローカルかリモート、または両方)、および主キークラスの型が表示されます。

   この区画では、Bean のクラスの確認または変更ができます。必要に応じて、他の Bean クラス、インタフェース、または主キークラスを指定できます。各項目の変更ボタンをクリックすると、そのクラスやインタフェースを格納したり、取り出したりするパッケージを指定できるようになります。

   - たとえば、関連する 1 つ以上のオブジェクトに対するパッケージ名を変更することで、Bean クラスはあるディレクトリに保存し、ホームインタフェースとリモートインタフェースは別のディレクトリに保存する、といった処理ができます。

     ただし、これを実行する前には、利用するアプリケーションがファイルの分散保存をサポートしているかどうかを確認する必要があります。

   - 指定した既存のクラスまたはインタフェースが、必要とされるメソッドや例外を持っていない場合は、エラーメッセージが表示されます。
   - パッケージおよびディレクトリ名には、有効な Java 識別子を使用してください。

7. 「完了」をクリックします。

   EJB ビルダーウィザードによって、CMP エンティティ Bean のインフラストラクチャが自動的に生成されます。次に生成されたクラスを見てみます。

CMP エンティティ Bean のクラス

EJB ビルダーウィザードは、デフォルトの CMP エンティティ Bean クラスを生成し、すべてのクラスの関係を設定します。図 4-2 に、エクスプローラの「ファイルシステム」区画に表示される一般的な CMP エンティティ Bean を示します。この例では、デフォルトのインターフェース設定「ローカルインターフェースのみ」が有効になっています。この Bean は同じ JVM 内で動作しているアプリケーションコンポーネントによってのみ呼び出されます。

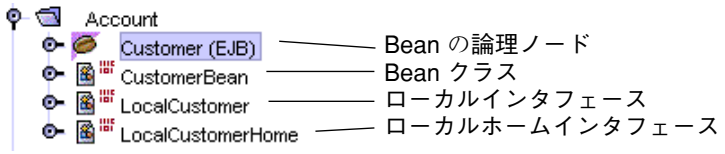
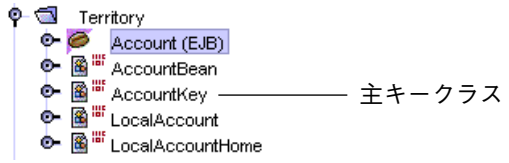


図 4-2　一般的な CMP エンティティ Bean のデフォルトクラス

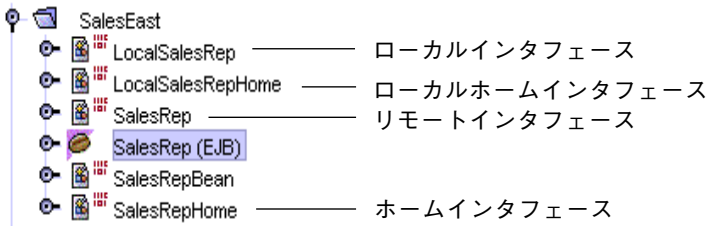
図 4-2 に示されている 4 つのノードのうち、(クラスアイコンで表示されている) 3 つは実際のクラスを表し、 (Bean アイコンで表示されている) 1 つは論理ノードを表しています。編集はすべて論理ノードで行ってください。例題に示されている Bean の主なノードについて説明します。

- エクスプローラによって提供される論理ノードには、エンタープライズ Bean のすべての要素がグループとしてまとめられており、論理ノードへの操作がそれぞれのクラスに伝達されます。
- Bean クラスは、 `javax.ejb.EntityBean` インタフェースと、エンティティ Bean のメソッドを実装します。
- ローカルインタフェースは `javax.ejb.EJBLocalObject` のサブクラスで、同じコンテナに格納されているビーン間の通信を実現します。
- ローカルホームインタフェースは `javax.ejb.EJBLocalHome` のサブクラスで、生成メソッドと検索メソッドのシグニチャを提供します。

主キークラスを作成した場合は (Bean に複合主キーが使われている場合など)、その Bean に対する追加のノードがエクスプローラに表示されます。



「リモートインターフェースとローカルインターフェースの両方」を選択した場合 (Bean がアプリケーションの同じ JVM 中の Bean と外部 JVM の Bean の両方から使用される場合)、作成されたエンティティ Bean は 4 つのインタフェースすべてを持つことになります。



- リモートインタフェースは `javax.ejb.EJBObject` のサブクラスで、CMP エンティティ Bean のビジネスメソッドを宣言します。
- ホームインタフェースは `javax.ejb.EJBHome` のサブクラスで、クライアントが呼び出す対象となる CMP エンティティ Bean の生成メソッドと検索メソッドを宣言します。

ノードの展開

エンティティ Bean のパッケージノードの下位にあるノードを展開すると、図 4-3 に示すようなツリーが表示されます (これは、デフォルトの「ローカルインターフェースのみ」が選択された場合の例です)。

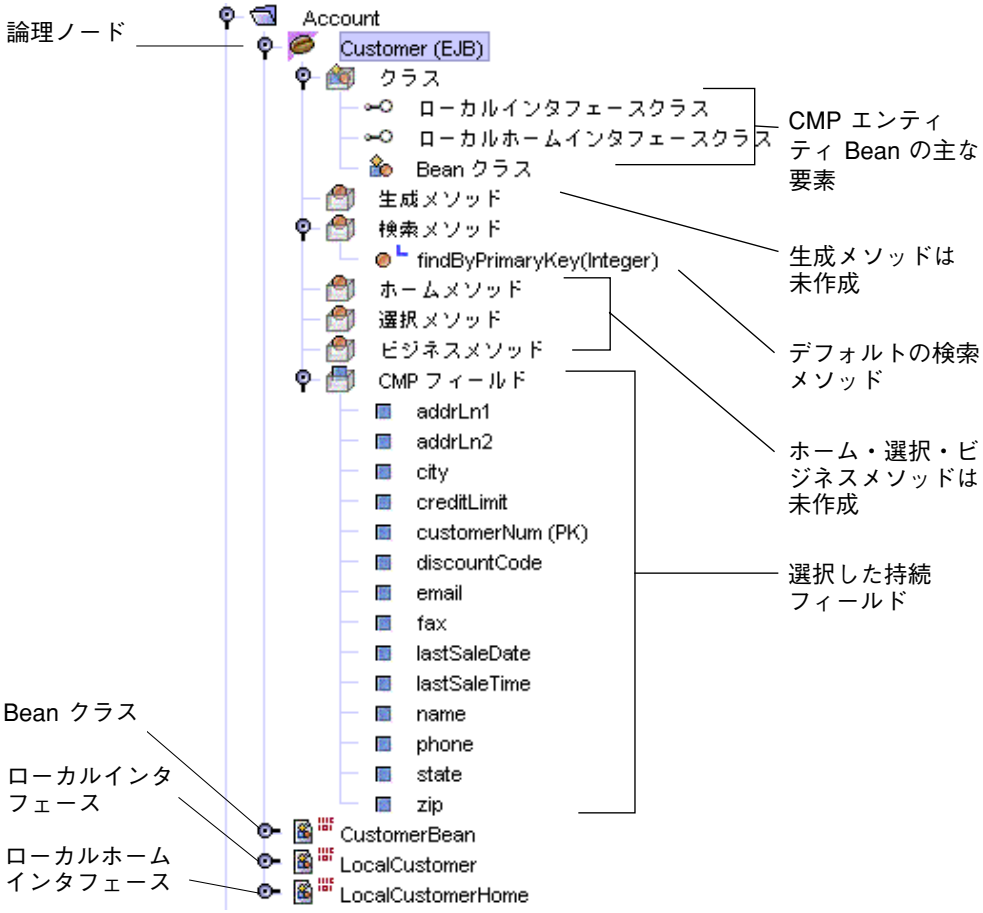


図 4-3　ローカルインタフェースを持つ一般的な CMP エンティティ Bean のエクスプローラでの詳細表示

新しい主キークラスを生成した場合のエクスプローラの表示例は 図 4-4 のとおりです。

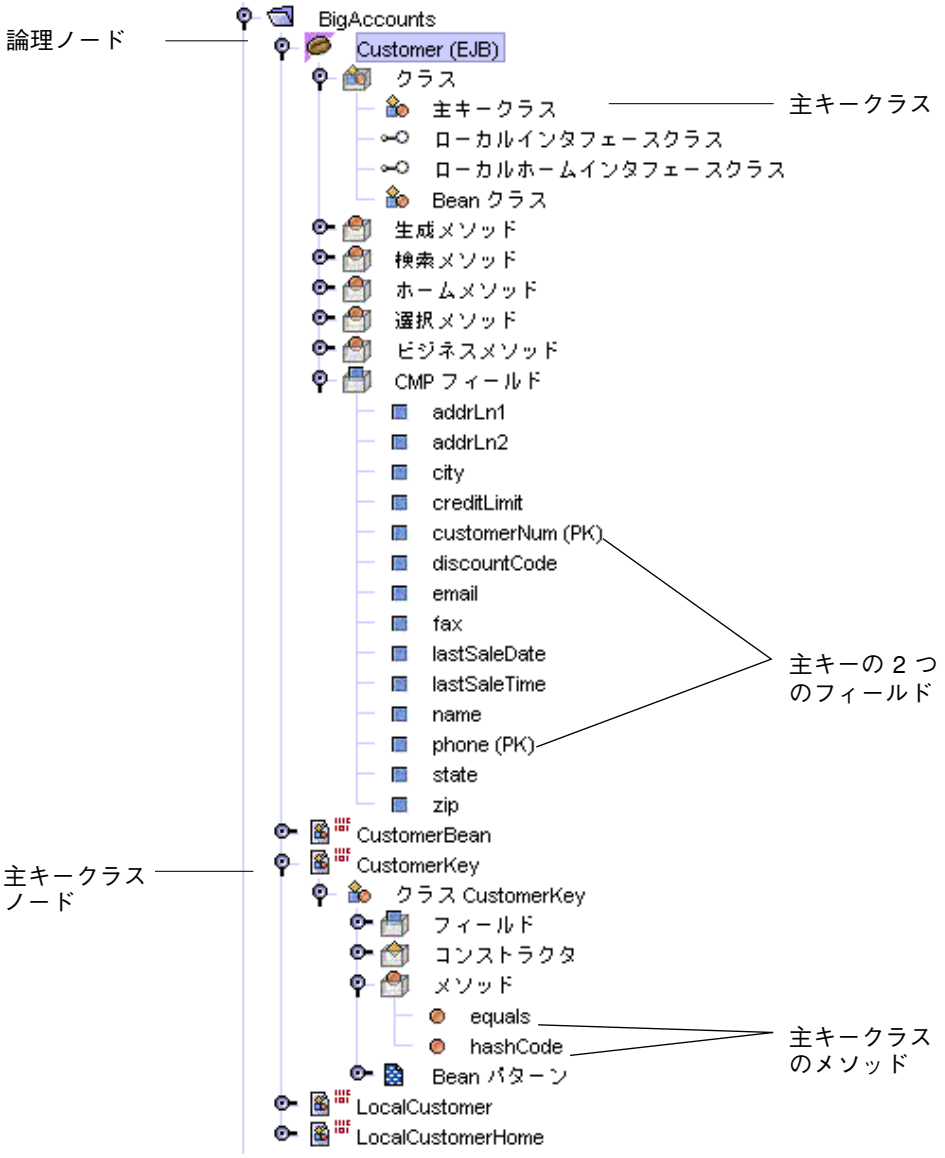


図 4-4　複合主キーを持つ一般的な CMP エンティティ Bean のエクスプローラでの詳細表示

生成されたクラスの確認

データベースの列に対応付けられたフィールドはすべて CMP エンティティ Bean に表示されます。更に、特定のデフォルトメソッドはすべてのエンティティ Bean 内に自動的に作成されます。

デフォルトの検索メソッド

すべてのエンティティ Bean は主キーによって検索できるように *Enterprise JavaBeans* 仕様によって規定されているため、エンティティ Bean のホームインタフェースにはメソッドシグニチャである `findByPrimaryKey` が自動的に追加されます。CMP エンティティ Bean では、Bean のコンテナに findByPrimaryKey メソッドが実装されるため、メソッドシグニチャだけで充分です。

```
public CustomerLocal findByPrimaryKey(java.lang.Integer aKey)
    throws javax.ejb.FinderException;
```

持続フィールドと補助メソッド

IDE は、CMP エンティティ Bean に指定した各持続フィールドに対し、get メソッドと set メソッドを作成し、それらを Bean クラスに設定します。これらの補助メソッドを参照するには論理ノードを右クリックし、「開く」を選択します。ソースエディタが開いて、生成された Bean クラスのコードが表示されます。コードの終わりの方に次に示す例のような行が見つかるはずです。

```
public abstract java.lang.Integer getCustomerNum();
public abstract void setCustomerNum(java.lang.Integer
    customerNum);
public abstract java.lang.String getDiscountCode();
public abstract void setDiscountCode(java.lang.String
    discountCode);
public abstract java.lang.String getName();
public abstract void setName(java.lang.String name);
```

CMP フィールドそのものは、配備記述子内に宣言されます。これを参照するには、論理ノードの Bean クラスを選択し、右クリックしてから「配備記述子を表示」を選択します。次の例では、 `Customer` という名前の 抽象スキーマ (持続性プランともいう)

を持つ CMP エンティティ Bean に対して、customerNum、discountCode、および name という名前を持つ CMP フィールドを宣言したときの配備記述子の XML コードを示しています。

```
<abstract-schema-name>Customer</abstract-schema-name>
<cmp-field>
    <field-name>customerNum</field-name>
</cmp-field>
<cmp-field>
    <field-name>discountCode</field-name>
</cmp-field>
<cmp-field>
    <field-name>name</field-name>
</cmp-field>
...
<primkey-field>customerNum</primkey-field>
```

持続フィールドをまだ指定していない場合は、CMP エンティティ Bean には defaultField と呼ばれるデフォルトの CMP フィールドが 1 つと、そのフィールドに対する補助メソッドだけが含まれています。このフィールドは自動的に主キーとして扱われます。

ウィザードを使用して CMP エンティティ Bean を定義した後は、 いつでも CMP フィールドを変更できます。Bean の論理ノードを選択後、右クリックして「CMP フィールドを追加」を選択するとフィールドを追加できます。また、新しい CMP フィールドを主キーとして指定することもできます。

Bean の作成後でも CMP フィールドの名前は変更できます。名前の選択は、論理ノードの下にある CMP フィールドを選択し、右クリックしてから「名前を変更」を選択するだけで実行できます。EJB ビルダーによって変更範囲の入力が促されます。

残りの CMP エンティティ Bean の持続性 (アセンブルとサーバーへの配備に Bean が必要とする実際の SQL 文) については、後述します。EJB QL 文に選択メソッドか検索メソッドを追加することで、Bean はすべてのアプリケーションとデータベースサーバー に移植することができます。この EJB QL 文は EJB コンテナの配備記述子と指定したアプリケーションサーバーに保持されます。配備の実行中に、この EJB QL コードはサーバ固有のデータベースへのアクセス用のコードに変換されます。ほとんどの持続性はリレーショナルデータベースで実装されるので、通常は SQL がオブジェクトコードとなります。J2EE RI サーバーの場合、変換後の SQL コードは Bean のプロパティシートにあるサーバーのタブに表示されます。

エンタープライズ Bean の準備の詳細については、第 8 章を参照してください。EJB QL コードのコーディングについては IDE のオンラインヘルプを参照してください。

主キークラスと必要なメソッド

これまでの操作で、EJB ビルダーのウィザードによってデータベース表の主キーが CMP エンティティ Bean の主キーフィールドに対応付けられているか、1 つ以上の主キーフィールドが手動で定義されているはずです。Bean が複合主キーを持っている場合、ウィザードが主キークラスを生成します (複合主キーを持っていない場合は、Bean には主キークラスがありません。後でデータベースの主キーを対応付ける主キーフィールドを作成する際には、最初に主キークラスを作成する必要があります。108 ページの「主キーの新規作成」を参照してください)。

主キークラスには、Bean のインスタンスを一意に識別する一連のデータが含まれています。Bean に単一の主キーフィールドがある場合、ウィザードはフィールドのクラスをその Bean の主キークラスとして使用します。 Bean が複合主キー (複数の持続フィールドで構成されている主キー) を持っている場合、ウィザードは同じ名前とタイプを持つ複数フィールドに対して主キークラスを 1 つ生成します。

更に、新しい主キークラスが作成されると、EJB ビルダーによってコンテナに必要な 2 つのメソッドが次のように挿入されます。

```
public boolean equals(java.lang.Object otherOb) {
    ...
}
public int hashCode() {
    ...
}
```

`equals` メソッドは、`id` 値が同一のオブジェクト (すなわちハッシュコードが同じキー) 同士を比較します。このメソッドには、パラメータとしてキー値が渡されます。このメソッドは、渡されたキー値が現在のキー値と一致しているかどうかを示す論理値を返す必要があります。

`hashCode` メソッドは、キーを整数値に変換し、ハッシュ表からキーをすばやく検出できるようにします。このメソッドは、現在のインスタンスのハッシュコードキーを返す必要があります。この値は一意である必要はありませんが、ハッシュ値が重複する確率が低いほど、エンティティ Bean のパフォーマンスが高くなります。

主キークラスは、java.rmi.Remote インタフェースではなく java.io.Serializable インタフェースを実装している必要があります。

IDE のテスト機能を使用して、CMP エンティティ Bean のメソッドをテストする場合に知っていると便利な点を次に挙げます。

- Bean の主キークラスに全フィールドコンストラクタを含めるか、クラスメンバーに設定メソッドを設定します。
- テストするアプリケーションの表示がわかりやすいように適切な toString メソッドを定義します。

テスト機能の使用に関する詳細は第 9 章を参照してください。

CMP エンティティ Bean のライフサイクルメソッド

ウィザードは、すべてのエンティティ Bean の Bean クラスに次のデフォルトのライフサイクルメソッドを追加します。

```
public void setEntityContext(EntityContext context) {
    this.context = context;
}
public void unsetEntityContext() {
    context = null;
}
public void ejbActivate() {
}
public void ejbPassivate() {
}
public void ejbLoad() {
}
public void ejbStore() {
}
public void ejbRemove() {
}
```

表 4-2 に CMP エンティティ Bean のライフサイクルメソッドの目的を示します。

表 4-2　CMP エンティティ Bean クラスでのデフォルトのライフサイクルメソッドの目的

メソッド	目的
`setEntityContext`	このメソッドは、フィールドに `EntityContext` の参照を格納し、非持続フィールドに値を格納できるようにする。このメソッドを使用して、EJB オブジェクトに依存せず、エンティティ Bean の全期間にわたって存続するリソース (たとえばデータベース接続ファクトリ) を割り当てることができる。デフォルトでは、`context` という非持続フィールドに `EntityContext` を代入するコードが生成される
`unsetEntityContext`	このメソッドを使用して、コンテナがエンティティ Bean のインスタンスを破棄する前に、そのインスタンスで使用されていたリソースの割り当てを解除し、メモリーを解放することができる。デフォルトでは、`context` フィールドの値を `null` に設定するコードが生成される
`ejbActivate`	このメソッドは、Bean を初期化して使用できるようにし、インスタンスに必要なリソースを取得する
`ejbPassivate`	このメソッドは、Bean のインスタンスが汎用インスタンスプールに戻される前に、その Bean が使用していたリソースを解放する
`ejbLoad`	CMP エンティティ Bean では、このメソッドのコードを編集する必要はない。コンテナは、使用可能状態の Bean インスタンスの `ejbLoad` メソッドを呼び出し、その Bean インスタンスの状態をデータベースに同期させる
`ejbStore`	CMP エンティティ Bean では、このメソッドのコードを編集する必要はない。コンテナは、使用可能状態の Bean インスタンスの `ejbStore` メソッドを呼び出し、その Bean の状態をデータベース中のエンティティの状態に同期させる
`ejbRemove`	CMP エンティティ Bean では、このメソッドはクリーンアップ処理を行い、コンテナがデータを削除できるようにする

CMP エンティティ Bean の完成

CMP エンティティ Bean を完成させるには、次の操作を実行します。

- Bean のクライアントがデータベースにデータを挿入できるようにするには、生成メソッドを定義します。1 つのエンティティ Bean に 1 つ以上の生成メソッドを含めることができます。
- 必要に応じて主キーを追加または置き換えます。
- 必要なビジネスメソッドを定義します。
- `findByPrimaryKey` 以外の検索メソッドが必要な場合は、それらの検索メソッドを定義します。
- Bean が Bean インスタンスに依存しない操作を実行する必要がある場合は、1 つ以上のホームメソッドを定義します。
- CMP エンティティ Bean に同じ EJB モジュール内の他の Bean を照会させたい場合、またはデータベースを照会させて結果をローカルインターフェースまたはリモートインタフェースを介して戻したい場合は、1 つ以上の選択メソッドを定義します。
- 必要に応じて `setEntityContext`、`unsetEntityContext`、`ejbActivate`、`ejbPassivate`、`ejbRemove` の各メソッドにコードを追加し、これらのメソッドを完成させます。

ウィザードで Bean に必要なフィールドをすべて指定しなかった場合は、後から 1 つ以上の CMP フィールドを追加することができます。

これらの作業は、エクスプローラで基本的に Bean の論理ノードに用意された GUI ツールを使用して行います。これらのメソッドの内容を次の手順で指定します。

1. ダイアログでメソッド名を指定し、メソッドのシグニチャを定義します。論理ノードを選択し、右クリックし、「生成メソッドを追加」、「ビジネスメソッドを追加」、「検索メソッドを追加」、「ホームメソッドを追加」または「選択メソッドを追加」のいずれかを選択します。定義したメソッドが CMP エンティティ Bean の適切なクラスに伝達されます。
2. ソースエディタを使用してメソッドの本体を完成させます。

推奨する エンタープライズ Bean の開発手順

付録 A では、エンタープライズ Bean に変更を加える際の最適な方法や、推奨手順以外の方法を行った場合に発生する可能性のあるエラーや異常状態について説明しています。一般に、個々のクラスノードを操作するよりは論理ノードを操作する、メソッドの変更には Bean のプロパティシートや「カスタマイザダイアログ」を使用する、およびダイアログからでは操作できない Bean のコードを完成させたり、編集したりするには IDE のソースエディタを使用するといった方法をお勧めします。

生成メソッドの定義

エンティティ Bean には、複数の生成メソッドを含めることができます。各 Bean では、生成メソッドをホームインタフェースに、それに対応する `ejbCreate`*Xxx* メソッドと `ejbPostCreate`*Xxx* メソッドを Bean クラスに配置する必要があります。ここで推奨している手順に従うと、これらのメソッドが適切に生成および伝達されます。

通常、CMP エンティティ Bean の `ejbCreate`*Xxx* メソッドは次の処理を実行します。

- クライアントから渡された引数を検証します。
- インスタンスの変数 (CMP エンティティ Bean では CMP フィールド) を初期化します。コンテナは、Bean の CMP フィールドをデータベースに書き込む直前に、`ejbCreate` メソッドを呼び出します。

`ejbPostCreate`*Xxx* メソッドは IDE によって自動的に追加されます。このメソッドは、EJB オブジェクトについての情報 (ホームインタフェース、リモートインタフェースなど) を、その情報を参照する必要のあるほかの エンタープライズ Bean に伝達したい場合に使用します。このメソッドは、コンテナからパラメータとして渡される `EntityContext` を使用して、リモートインタフェースにアクセスすることができます。また、このメソッドは通常、他に依存する Bean の作成に使用されます。たとえば、`Order Bean` の `ejbCreateLineItem` メソッドによって `ejbPostCreate`*Xxx* メソッド内に商品のデータが作成される、という場合などです。

新しい生成メソッドを定義するには、次の手順に従います。

1. CMP エンティティ Bean の論理ノードを選択し、右クリックし、「生成メソッドを追加」を選択します。

   「新規生成メソッドを追加」ダイアログが表示されます。

2. `create` に続けて (create から始まる名前でなくてもよい)、生成メソッドの名前を指定します。

   この時点で生成メソッドのパラメータを追加します。

3. 「追加」をクリックします。

4. 「メソッドのパラメータを入力」ダイアログで、パラメータの型と名前を指定します。

   CMP エンティティ Bean では、create メソッドは主キー型または主キーと同じ型を返す必要があります。次のコード例に示すように、Bean クラスのメソッドシグニチャでは主キー型が指定されますが、メソッドの本体では `null` を返す必要があります。これは、CMP エンティティ Bean の主キーはコンテナが管理するためです。

```
public 主キー型 ejbCreate( パラメータ 1...) throws 例外 1
```

5. 「了解」をクリックします。

   追加したメソッドが、Bean クラスのコードでは `ejbCreate`*Xxx* として、ホームインタフェースでは `create` として表示されます。さらに、Bean クラスにはメソッド `ejbPostCreate`*Xxx* も表示されます。メソッドの追加時にすでにソースエディタを開いている場合は、コードの表示がただちに更新されます。

   Bean クラスに生成される `ejbCreate` メソッドと `ejbPostCreate` メソッドの例を次に示します。

```
public String ejbCreate(java.lang.String custname)
    throws CreateException {
}
public void ejbPostCreate(java.lang.String custname)
    throws CreateException {
}
```

6. ソースエディタを使用して、return 文やその他の必要なコードを新しい生成メソッドに追加します。

   銀行のスタッフが、各支店のサービス品質アンケートに対する顧客からの回答を参照する Web アプリケーションの生成メソッドを、コード例 4-1 に示します。この例では、作成された CMP エンティティ Bean のインスタンスに、custName、branchNo、response の 3 つのフィールドが含まれているものとします。

コード例 4-1　CMP エンティティ Bean クラスの生成メソッドの例

```
public CustomerSurveyKey ejbCreateResponse(java.lang.String custName,
    java.lang.String branchNo,java.lang.String response)
    throws CreateException {
        if ((branchNo == null) || (custName == null)){
            throw new CreateException("Both the branch number and
                the customer name are required.");
        }
        setCustName(custName);
        setBranchNo(branchNo);
        setResponse(response);

        return null;
}
```

主キーの追加または置き換え

エンティティ Bean の主キークラスが削除されている場合や、クラスに主キーを追加する必要がある場合は、プロパティシートを次のように使用します。

1. 論理ノードを選択し、右クリックし、「プロパティ」を選択します。

   エンティティ Bean のプロパティシートが表示されます。

2. 「主キークラス」フィールドを選択し、省略符号ボタン (...) をクリックします。

   「プロパティエディタ」ダイアログが表示されます。

3. 既存のフィールドか既存のユーザー定義クラスを選択し、「了解」をクリックします。

   「主キークラス」フィールドに、新しいフィールドやクラスの戻り値の型が表示されます。

主キーの新規作成

主キークラスのないエンティティ Bean に新しい主キーを追加する場合、最初に新しい主キーフィールドを 1 つ以上追加する必要があります。その後、EJB ビルダーのウィザードを使用して、主キークラスを持つ新しいエンティティ Bean を作成します。これは一時的に利用するだけで、ここでは一時 Bean と呼びます。最後に、既存の Bean に新しい主キークラスを利用するよう指定します。

次の操作を実行します。

1. エクスプローラで、主キーが必要な既存エンティティ Bean を含んでいるパッケージを見つけます。
2. パッケージを右クリックし、「新規作成」->「J2EE」->「CMP エンティティ EJB」を選択します。
3. ウィザードで次の操作を実行します。
   a. 既存エンティティ Bean に主キー用のフィールドを必要な数だけ指定します。
   b. 最後から 1 つ目のパネルで、主キークラスの名前を必要に応じて変更します。これは、そのクラスを使用する既存 Bean との対応をわかりやすくするためです。

      たとえば、Bean の名前が `Account` だった場合は、主キークラスの名前を `AccountKey` とするとわかりやすくなります。
4. 「完了」をクリックします。

   これで一時 Bean が作成されました。次の操作を続けてください。
5. エクスプローラウィンドウで、一時 Bean のクラスと論理ノードを必要に応じて削除します。ただし、主キークラスは削除しないでください。
6. 既存 Bean の論理ノードを右クリックし、「プロパティ」を選択します。
7. プロパティシートの「プロパティ」タブで「主キークラス」フィールドをクリックし、省略符号ボタン (...) をクリックします。

   「主キークラス」ダイアログが開きます。
8. 「既存のユーザー定義クラスを選択」をクリックします。

   ファイル選択画面が開きます。

9. 新しい主キークラスまでナビゲートし、主キーを選択して「了解」をクリックします。

   プロパティシートの「主キークラス」フィールドに、新しい主キークラスの名前が表示されます。

   Bean の論理ノードに警告マークやエラーマークが表示される場合があります。これらのマークはこの時点では無視します。設定を終えたらプロパティシートを閉じます。

10. 既存 Bean の論理ノードを右クリックし、「EJB の検査」または「エラー情報」を選択します。

    解決が必要なエラーが IDE によって表示されます。

11. エラーをすべて修正し、再び Bean 検査を実行するか、Bean をコンパイルします。

    主キークラスが必要とする `equals` と `hashcode` の 2 つのメソッドは、この処理では再生成されません。したがって、通常は `equals` メソッド中のクラス名を変更する必要があります。Bean クラスにある `findByPrimaryKey` メソッドには異なるパラメータの型の指定が、すべての `ejbCreate` メソッドには異なる戻り値の型の指定が必要なこともあります。

12. これまでの操作をすべて保存します。

外部キーの取り扱い

外部キーとして実装されている複数の CMP エンティティ Bean の関係を維持したい場合、または Bean が複数のデータストアにアクセスする必要がある場合は、個々の Bean を独立して作成するのではなく、関連した CMP エンティティ Bean のセットとして作成します。EJB ビルダーのウィザードを使用すると、関連した Bean のセットをまとめて 1 度に作成できます。これには、ソースとなる表の外部キーを、それらの Bean 間のコンテナ管理による関係 (CMR) の表現としてそのまま使用します。この詳細については第 5 章を参照してください。

ビジネスメソッドの定義

CMP エンティティ Bean にはビジネスメソッドを追加します。ビジネスメソッドは、エンティティ Bean にカプセル化されたビジネスロジックを実行します。通常、ビジネスメソッドでは持続フィールドを操作し、データベースには直接アクセスしませ

ん。ビジネスメソッドの役割は、インスタンス変数を更新することです。EJB コンテナは、トランザクションの必要な時点で `ejbLoad` メソッドと `ejbStore` メソッドを呼び出し、その時にインスタンス変数がデータベースに書き込まれます。

注 - ビジネスロジックはなるべくデータベースアクセスコードから分離してください。

ビジネスメソッドを定義するには、次の手順に従います。

1. **論理ノードを選択し、右クリックし、「ビジネスメソッドを追加」を選択します。**

   「新規ビジネスメソッドを追加」ダイアログが表示されます。

2. **メソッド名を入力し、戻り値の型、パラメータと例外を指定します。「了解」をクリックすると、ダイアログが閉じます。**

3. **ソースエディタでメソッドのコーディングを完成させます。**

   または「新規ビジネスメソッドを追加」ダイアログで、メソッド名だけを指定して「了解」をクリックします。ダイアログが閉じたら、ソースエディタでコードを完成させます。

   コード例 4-1 と同じアプリケーションのビジネスメソッドをコード例 4-2 に示します。この例では、銀行の顧客の電話による回答がデータベースから読み出されます。

コード例 4-2　CMP エンティティ Bean のビジネスメソッドの例

```
public java.lang.String retrieveComments() {
    return phoneResponse();
}
```

エンタープライズ Bean に作成したメソッドを参照したい場合は、該当する Bean の論理ノードを展開して、参照するメソッドの種類までサブノードをナビゲートします。メソッドのノードを右クリックして「開く」を選択します。ソースエディタが開いて、そのクラスのメソッドのコードが表示されます。

検索メソッドの追加

EJB ビルダーのウィザードは、デフォルトの `findByPrimaryKey` メソッドを自動的に生成します。CMP エンティティ Bean でそれ以外の照会を実行したい場合は、検索メソッドを追加する必要があります。

また、関連するエンティティ Bean の持続フィールドの値を照会したい場合、またはクライアントによって起動されるメソッドが必要ない場合は、選択メソッドが使用できます。詳細については 113 ページの「選択メソッドの定義」を参照してください。

検索メソッドを追加するには、次の操作を実行します。

1. Bean の論理ノードを選択し、右クリックして「検索メソッドを追加」を選択します。

   「新規検索メソッドを追加」ダイアログが表示されます。

2. `find` で始まるメソッド名を入力します。

3. 次の戻り値の型のうち、どれかを選択します。

   - デフォルト名で示される単一オブジェクト
   - Collection (集合)
   - Enumeration (列挙型)

4. パラメータと例外を指定します。

5. 「SELECT、FROM、WHERE」フィールドに EJB QL 文を指定します。

   EJB QL の構文と例題については、IDE のオンラインヘルプを参照してください。また、過去の CMP エンティティ Bean の検索メソッドでの `WHERE` 文の指定方法については 192 ページの「EJB 1.1 CMP エンティティ Bean を含んだ EJB モジュール」を参照してください。

   この時点で EJB QL 文を入力する準備がまだできていない場合、コンパイラの EJB QL コードの入力要求を解除できます。224 ページの「エンタープライズ Bean のコンパイルと検証」を参照してください。ただし、RI サーバーに Bean を配備する前には必ず正しい EJB QL 文を指定してください。

   メソッドの EJB QL コードは、配備記述子に組み込まれます。EJB QL 文は、カスタマイザダイアログまたはメソッドのプロパティシートを使用して編集または追加できます。ただし、配備記述子を直接編集するのは通常は避けてください。

6. 指定が完了したら「了解」をクリックします。

次のようにして、ソースエディタを直接開いて検索メソッドを表示させます。CMP エンティティ Bean の論理ノードを展開して検索メソッドまでナビゲートします。目的の検索メソッドを選択して、右クリックして「開く」を選択します。ソースエディタに検索メソッドのホームインタフェースクラスが表示されます (検索メソッドは Bean クラスでは宣言されていません)。

例題の `Account` Bean での検索メソッドは次の 2 つの処理を実行します。

- 特定の口座に関するデータが記述されている `AccountEJB` インスタンスを検索し、そのインスタンスにリモートオブジェクトを戻します。これを実行するには、口座を口座番号で SELECT する必要があります。
- すべての赤字口座の `AccountEJB` インスタンスを検索し、これらのリモートオブジェクトを集合として戻します。これを実行するには、残高がマイナスになっている口座を SELECT する必要があります。

検索メソッドはカスタマイザダイアログ (メソッドのノードを右クリックし「カスタマイズ」を選択します) か、メソッドのプロパティシート (メソッドのノードを右クリックし「プロパティ」を選択します) を使用して編集できます。

ホームメソッドの定義

ホームメソッドを使用すると、エンティティ Bean のインスタンスに依存しない処理を実行することができます。ホームメソッドは、静的メソッドに似ていて、指定されたクラスのすべての Bean に対して適用されるビジネスロジックを含んでいます (一方、ビジネスメソッドはエンティティ Bean 中の 1 つのインスタンスに固有の識別子とロジックを持っています) 。ホームメソッドは Bean の持続性状態 (インスタンス変数) またはコンテナ管理による関係には関係しません。

CMP エンティティ Bean の `Invoices` が顧客への請求書を表していて、各請求書にはその顧客が支払った額が記録されていたとします。これら未払いになっている請求書の合計額を参照したい場合、該当する請求書の Bean インスタンスの収集を繰り返し、それらの未払い合計額をビジネスメソッドに計算させる `getAmountDue` というメソッドを作成し、ホームメソッドに追加することができます。

ホームメソッドを定義するには、次の操作を実行します。

1. 論理ノードを選択し、右クリックして「ホームメソッドを追加」を選択します。

「新規ホームメソッドを追加」ダイアログが表示されます。

2. メソッド名を指定します。
3. 戻り値の型、パラメータ、および例外を指定します。
4. 指定が完了したら「了解」をクリックします。

   新規ホームメソッドの名前を指定した時点で「了解」をクリックし、ソースエディタでコーディングを完成させることもできます。

   CMP エンティティ Bean にクライアントビューが 2 つ (リモートとローカル両方のインタフェース) あった場合、EJB ビルダーによって含めるホームメソッドがローカルホームインタフェースに対するものか、(リモート) ホームインタフェースに対するものか、または両方に対するものかが確認されます。

   IDE によってどちらかまたは両方のホームインタフェースにホームメソッドが追加されます。また、対応する `ejbHome` メソッドも Bean クラスに追加されます。

選択メソッドの定義

データベースに照会を実行し、ローカルインタフェースまたはリモートインタフェース (あるいは両方のインタフェースの集合) を値として戻す CMP エンティティ Bean を使用する場合、または関連するエンティティ Bean の持続フィールドの値 (複数ある場合はそれらの値の集合) を返すメソッドを使用したい場合、選択メソッドを使用することができます。選択メソッドは、複数の CMP エンティティ Bean の関係が定義される際に作成される get メソッドに直接関係していてます。また、選択メソッドは、そのエンティティ Bean クラス内の 1 つのメソッド (通常はビジネスメソッド) だけから呼び出すことができます。選択メソッドはリモート型インタフェースからはアクセスできないため、クライアントから呼び出すことはできません。

選択メソッドを追加するには、次の操作を実行します。

1. Bean の論理ノードを選択し、右クリックして「選択メソッドを追加」を選択します。
2. `ejbSelect` で始まるメソッド名を指定します。
3. 次のどちらかの戻り値の型を選択します。
   - デフォルト名で示される単一オブジェクト
   - コンボボックスに表示されている型のどれか
4. パラメータと例外を指定します。

5. 「SELECT、FROM、WHERE」フィールドに EJB QL 文を指定します。

   EJB QL の構文および例題については IDE のオンラインヘルプを参照してください。

6. 指定が完了したら「了解」をクリックします。

   選択メソッドはカスタマイザダイアログ (メソッドのノードを右クリックし「カスタマイズ」を選択します) か、メソッドのプロパティシート (メソッドのノードを右クリックし「プロパティ」を選択します) を使用して編集できます。

   メソッドの EJB QL コードは、配備記述子に組み込まれます。EJB QL 宣言文は、カスタマイザまたはメソッドのプロパティシートを使用して編集または追加できます。ただし、Bean の配備記述子は直接編集できません。

追加フィールドの定義

CMP エンティティ Bean を作成し終わったら、次のようにして CMP フィールドを追加できます。

- 論理ノードを選択し、右クリックして「CMP フィールドを追加」を選択します。

注 - ソースエディタを使用して Bean クラスに直接フィールドを書き込まないでください。IDE は、手動で書き込まれたフィールドと配備記述子との持続性を認識できません。

CMP エンティティ Bean を作成した後の作業

作成した CMP エンティティ Bean は、最終環境で使用できるようにする必要があります。これらの最終作業については、第 8 章を参照してください。

また、付録 A では、推奨する エンタープライズ Bean の操作手順を説明しています。

詳細情報の参照先

エンタープライズ Bean は、アプリケーションに高い機能性と柔軟性を付加します。また、エンタープライズ Bean の基本要素は簡単に作成できます。Sun ONE Studio IDE のようなツールを使用すれば更に簡易化できます。しかし、アプリケーションのニーズをすべて満たす Bean の作成は複雑なこともあります。Bean の作成に関する詳細は次の文献を参照してください。

- *Enterprise JavaBeans* 仕様 , 2.0
  `http://java.sun.com/products/ejb/docs.html`
- *Java 2 Platform, Enterprise Edition Tutorial*
  `http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/CMP3.html`
- CMP Example Overview　(著 : Beth Stearns)
  `http://java.sun.com/j2ee/sdk_1.3/techdocs/release/CMP-RI.html`

第5章

関係 CMP エンティティ Bean のセットの開発

多くの J2EE アプリケーションには、コンテナ管理による持続性 (CMP エンティティ Bean) を使用する関連エンティティ Bean が含まれています。1 つのアプリケーションに含まれる 2 つの CMP エンティティ Bean には、Bean 間の関係を示すフィールドが含まれている場合があります。このフィールドは、データベースまたはデータベーススキーマにある、異なるエンティティまたは表が関連した列を含んでいることを示すものです。たとえば、あるスキーマに `Customer`、`Order`、`LineItem`、および `Part` の表が含まれていたとします。`Order` には `Customer` に対する外部キーが含まれ、`LineItem` には `Order` に対する外部キーが含まれます。また、`LineItem` には `Part` に対する外部キーが含まれます。

この章では、Sun ONE Studio 4 IDE で EJB ビルダーを使って一連の関係 CMP エンティティ Bean を一度に作成する方法を、必要なインタフェースと共に説明します。CMP エンティティ Bean は、データベースまたはスキーマモデル中のすべてのエンティティから、またはそれらのエンティティのサブセットから作成することができます。作成された Bean はウィザードによって自動的に EJB に保存されます (モジュールに関する詳細は第 8 章を参照してください)。

また、ウィザードはデータソース中のエンティティ間の関係も考慮します。作成されるCMP エンティティ Bean 間の関係は CMR フィールドと呼ばれる論理エンティティとして保持されます。IDE は、2 つのデータベース表の関係を誤って認識するようなこともありません。

関係 CMP エンティティ Bean のセットに対するインフラストラクチャのすべての作成処理は、ウィザードによって自動的に行われるため、メッセージに従って操作だけです。

関係 CMP エンティティ Bean を作成する際は、この章で説明する以外にも様々な方法を取ることができます。詳細については、xviii ページの「お読みになる前に」を参照するか、エンタープライズ Bean のプログラミングに関する市販の文献を参照してください。

EJB ビルダーと関係 CMP エンティティ Bean

EJB ビルダーは、ウィザード、プロパティシート、およびエディタの集まりで、これを使用して一貫性のある エンタープライズ Bean を簡単に作成できます。EJB ビルダーを使用すると、わかりやすい機能を利用しながら短期間で Bean を完成できます。この章では、 IDE の機能を最大限に活用した、Bean の一貫性と J2EE 標準への準拠を確実にする手法を説明しています。

関係 CMP エンティティ Bean の一括作成

EJB ビルダーで CMP エンティティ Bean を作成する際には、次の手順に従ってください。

- **CMP エンティティ Bean のセットと必要なクラスを作成します。**EJB ビルダーのウィザードで、関係 CMP エンティティ Bean のセットの枠組みを最初に作成します。各 Bean はクラス、インタフェース、および論理ノードを持ち、作成された Bean はエクスプローラの「ファイルシステム」タブに表示されます。次に、ウィザードはインタフェース宣言文を生成します。各 CMP エンティティ Bean の Bean クラスには、必要なメソッドの宣言と、データのソースがあるデータベース列に一致している持続フィールドが作成されます。

  ウィザードによって、関係 Bean として含める Bean を指定するよう促されます。含めない Bean は明示的に除外する必要があるため、Bean 間の関係を見失うことはありません。

  エンティティ Bean は、論理ノードから操作するのが最適です。エクスプローラでは、すべての論理ノードが アイコンで示されています。

- **メソッド、パラメータ、および例外を追加します。**IDE の GUI 機能の操作方法が、この章で後述されています。この GUI 機能とソースエディタを併用して、CMP エンティティ Bean のメソッド実装部分を完成させます。たとえば、Bean へのメソッドの追加は、コンテキストメニューから使用できるダイアログを使用しても、必要なクラスのセットを直接編集することもできます。
- **Bean の配備記述子に値を設定します。**論理ノードから表示できるエンティティ Bean のプロパティシートを使用して、配備に関するプロパティを編集します。

Bean の論理ノードからの個々の CMP エンティティ Bean の妥当性を検査することができます。IDE のテスト機能をそれぞれの Bean に対して実行することもできます。

関係 CMP エンティティ Bean のセットの作成

関係 CMP エンティティ Bean のセットは、必要であれば手動で構築できます。それぞれの Bean を作成し、次に EJB モジュールに追加し、最後にモジュールのプロパティシートを使って Bean 間の関係 (CMR) を宣言するという方法です。ただし、EJB ビルダーのウィザードを使用すればこの処理は自動的に実行されます。ウィザードを使用すれば、Bean 間の関係についても手動に比べて高い完全性や正確性を期待できます。

EJB コンテナの持続性管理が利用できず、自身で持続性 (BMP エンティティ Bean) を管理するようなエンティティ Bean のセットを作成する必要がある場合、これらの Bean 間の関係はすべて手動で記述する必要があります。 CMP エンティティ Bean と BMP のエンティティ Bean の違いについては、表 4-1 を参照してください。BMP エンティティ Bean の作成の詳細については第 6 章を参照してください。

この章ではこれ以降、ウィザードを使った関係 CMP エンティティ Bean のセットを一度に作成する方法と、開発時の注意点について説明します。

関係 CMP エンティティ Bean のセットの定義

EJB ビルダーのウィザードを使用すると、関係 CMP エンティティ Bean セットのコンポーネント作成処理の大部分を自動化できます。ウィザードが、各 CMP エンティティ Bean に対して実行する処理は次のとおりです。

- Bean クラスと選択したインタフェース (ローカルだけ、またはローカルとリモートの両方) で構成される必要最小限のクラスを生成します。

- CMP エンティティ Bean 中での主キークラスを作成します。これは、選択した表が複合主キーを必要とする場合、または主キーに Java の単純型が指定されている場合に作成されます。
- CMP エンティティ Bean 間での関係を作成します。これらの関係はコレクション (Collection) を戻す補助メソッドとして表されます。
- 関係 CMP エンティティ Bean のセットを格納する EJB モジュールを作成します。

一連の関係 CMP エンティティ Bean を定義する方法を次に説明します。

1. Bean を格納するパッケージと EJB モジュールを選択または作成します。
2. EJB ビルダーのウィザードを使用して、関係 CMP エンティティ Bean のインフラストラクチャを作成します。
3. 各 CMP エンティティ Bean のコードに生成メソッド、ビジネスメソッド、検索メソッド、選択メソッド、およびホームメソッドのいずれか適切なメソッドを追加します。
4. 追加したメソッドの本体を完成させます。
5. EJB モジュールのプロパティシートに配備情報を追加します。

以降に、これらの基本手順を説明します。

パッケージの生成

関係 CMP エンティティ Bean を格納するために、パッケージを生成する必要がある場合は、ファイルシステムを選択して、右クリックして「新規 」->「Java パッケージ」を選択します。

データベースまたはスキーマを使用する準備

関係 CMP エンティティ Bean のセットのソースとしてデータベースを利用するのか、データベーススキーマ (データベースのスナップショット) を利用するのかを決める必要があります。EJB ビルダーのウィザードは、データベースまたはデータベーススキーマ中の表の列を、関係 CMP エンティティ Bean の中に作成する持続フィールドに対応付けます。データベースまたはデータベーススキーマのどちらを使用しても完成したエンティティ Bean の内容は同じになります。

関係 CMP エンティティ Bean のセットを構築すると、EJB ビルダーによってデータベースの複数の表の関係が、セット中の各 CMP エンティティ Bean に保存されます。

使用するデータソースを決定する場合は、次の点に考慮してください。

- **稼動中のデータベースから関係 CMP エンティティ Bean のセットを作成する。** データベースに直接アクセスができ、データベースのユーザー間で競合が発生しない場合は、直接データベース接続を使用して関係 CMP エンティティ Bean のセットを作成することができます。この方法を実行する場合は、EJB ビルダーのウィザードを開始する前に、データベースが稼動していることが前提となります。
- **データベーススキーマから関係 CMP エンティティ Bean のセットを作成する。** データベースへのアクセスが制限されてはいるが、スキーマオブジェクトは使用できるといった場合、スキーマからの表を利用できます。この方法を実行する場合は、スキーマを使用するのに IDE のエクスプローラウィンドウが必要です。既存のスキーマがない場合は、データベースから収集する必要があります。

IDE に含まれる PointBase データベースサーバーに関する詳細、およびデータベーススキーマ収集に関する詳細については、82 ページの「データソースの準備」を参照してください。

関係 CMP エンティティ Bean のセットを生成している間は、データベースにアクセスしている必要があります。したがって、少なくとも EJB ビルダーのウィザードを終了するまではサーバーが稼動している必要があります (その後、プラグインを使って表を作成する場合には、CMP エンティティ Bean を使用するアプリケーションの配備中および実行中にも、データベースサーバーとアプリケーションサーバーが稼動している必要があります)。

元のデータベースとフィールドとの対応付けに関する情報は、EJB ビルダーのウィザードからアプリケーションに提供されます。通常、この情報はプラグインによってデフォルトでマッピングされます。 列とフィールドの対応付けをどう扱うかは EJB コンテナによって異なります。詳細については、コンテナおよびサーバーのプラグインのマニュアルを参照してください。

EJB ビルダーのウィザードの開始

関係 CMP エンティティ Bean のセットを作成するには、次の操作を実行します。

1. IDE のメインウィンドウで「表示」->「エクスプローラ」と選択し、エクスプローラウィンドウを開きます。
2. エクスプローラの「ファイルシステム」タブで CMP エンティティ Bean を格納するパッケージまたはファイルシステムを選択します。

3. 右クリックして「新規」->「J2EE」->「関係 CMP エンティティ EJB」を選択します。

   EJB ビルダーのウィザードが表示され、ウィンドウのタイトルバーに「新規ウィザード — 関係 CMP エンティティ EJB」と表示されます。左側のパネルに、現在の手順と、セッション Bean の作成を終えるまでの一連の手順が表示されます。

Bean のセットのインフラストラクチャの生成

EJB ビルダーの「EJB モジュール名およびデータソースを指定」区画で、関係 CMP エンティティ Bean を格納する EJB モジュール名を指定します。また、図 5-1 に示すように、持続フィールドと関係のソースについても指定します。

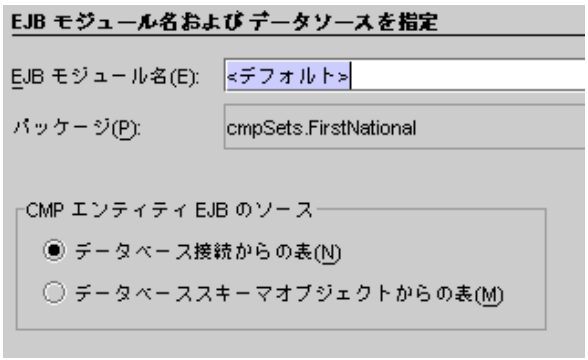


図 5-1　EJB ビルダーのウィザードでの CMP エンティティ Bean に関する選択項目

「EJB モジュール名およびデータソースを指定」区画では、Bean セットに関する次の基本的な項目を選択してください。

1. モジュール名を指定します。

   個々の Bean を EJB ビルダーのウィザードを使用して作成した場合、入力フィールドには Bean の名前を指定するように第 4 章では説明しました。しかし、関係 CMP エンティティ Bean のセット内の各 Bean の名前は、生成後すぐにウィザードによって付けられています。ここでは、関係 CMP エンティティ Bean のセットを格納するモジュールの名前を指定してください。

2. データベースソースを指定します。

   - これから生成する CMP エンティティ Bean が既存のデータベースからの表を表す場合は、「データベース接続からの表」を選択します (データベースはすでに稼動している必要があります)。詳細は、次の節を参照してください。

- 既存のスキーマを使用する場合は、「データベーススキーマオブジェクトからの表」を選択します (スキーマは作成済みで、IDE のエクスプローラで表示されるファイルシステム中に含まれている必要があります)。130 ページの「データスキーマオブジェクトの使用」 を参照してください。

3. 「次へ」をクリックします。

データベース接続の使用

前述のウィザードの区画で「データベース接続からの表」を選択すると「データベース接続を指定」区画が表示されます。

データソースとなるデータベースが稼動中であることを確認してください。IDE が提供されているデータベースを使用しない場合は、そのデータベース用のドライバファイルが Sun ONE Studio のインストールディレクトリ配下の `lib/ext` ディレクトリにあることを確認し、データベースを起動してください。

「データベース接続を指定」区画では、次の操作を実行します。

1. 「既存の接続」または「新規接続」をクリックします。
   - 接続は定義されているが、有効にはなっていないインストール済みまたは提供されているデータベースを使用する場合は、「既存の接続」を選択します。コンボボックスから該当するデータベースを選択します。
   - データベースがインストールされていても、接続がまだ定義されていない場合は「新規接続」を選択します。コンボボックスからデータベースのドライバを選択してください。データベースの接続に必要な情報を入力します。

   関係 CMP エンティティ Bean のセットの構築に使用できる表が、ウィザードの次の区画である「データベース表を選択」区画に表示されます。

2. 左側に表示される「使用可能な表」の一覧から使用したい表を選択し、右側の「選択した表」の一覧に加えます。

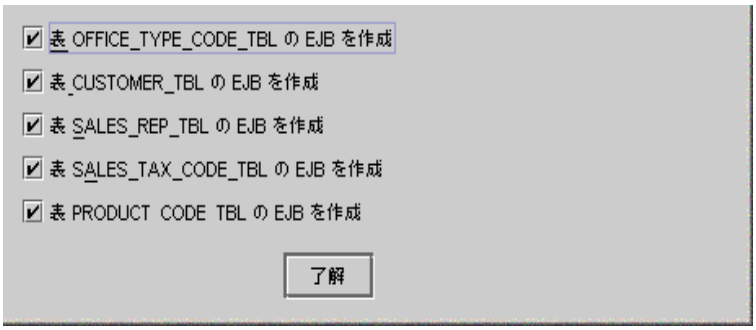
   EJB ビルダーでは、表間のすべての関係を関係 CMP エンティティ Bean のセット内に保持することができます。

   必要のない表が含まれていれば、それを除外することもできます。この場合、除外した表と選択した表との関係は失われます。表が除外された場合、EJB ビルダーは外部キーの列と非外部キーの列を同じように処理します。

3. 「次へ」をクリックします。

選択した表には、選択しなかった表を参照するための外部キーが含まれていることもあります。この場合、警告ダイアログが表示され、選択されなかった関係のある表の一覧が表示されます。選択されなかったこれらの表が、セット内のどの CMP エンティティ Bean によっても参照されることがないかどうかを確認してください。

このダイアログの一部の例を次に示します。これには、直前のダイアログで選択されなかった表の一覧が表示されています。デフォルトではすべての表が、セットに含まれるように選択されています。



(このダイアログに表示されている表は、最初に表示される一覧で選択状態のままになっています。これらの表には外部キーでアクセスできます。前述のダイアログで選択した表のセットと、このダイアログで選択が確認された表のセットが、CMR に含められるすべての外部キーでアクセスできる表となります。)

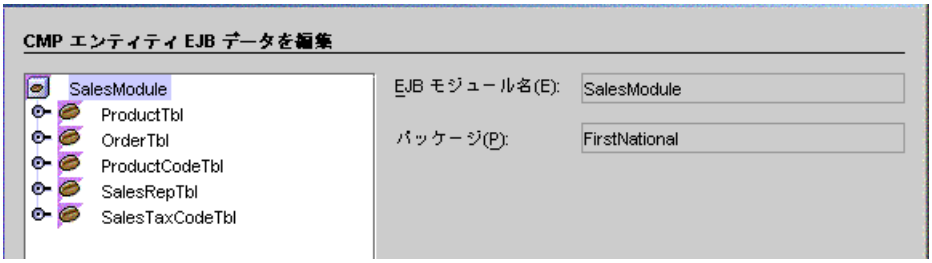
関係 CMP エンティティ Bean のセットには、後からでも表を追加できます。ただし、その場合は、追加の際に他の CMP エンティティ Bean との関係も指定する必要があります。使用するかどうかわからない表があって、それを追加してもアプリケーションの性能に影響が出ないことがわかっている場合は、この時点で追加しておいた方が操作が簡単に済みます。

4. **警告ダイアログで、関係 CMP エンティティ Bean のセットに含めたくない表のチェックボックスからチェックを外します。**

   スクロールバーで一覧全体を参照して、見落としのないようにしてください。

   「了解」をクリックすると、選択状態のまま残した表すべてが (手順 2 で明示的に選択した表に加え) セットに含まれる CMP エンティティ Bean となります。

「CMP エンティティ EJB データを編集」区画が表示されます。この区画には、これから作成される EJB モジュール、そのモジュールに含まれる CMP エンティティ Bean、EJB モジュール名、およびパッケージ名が表示されます。この区画の一部を次に示します。



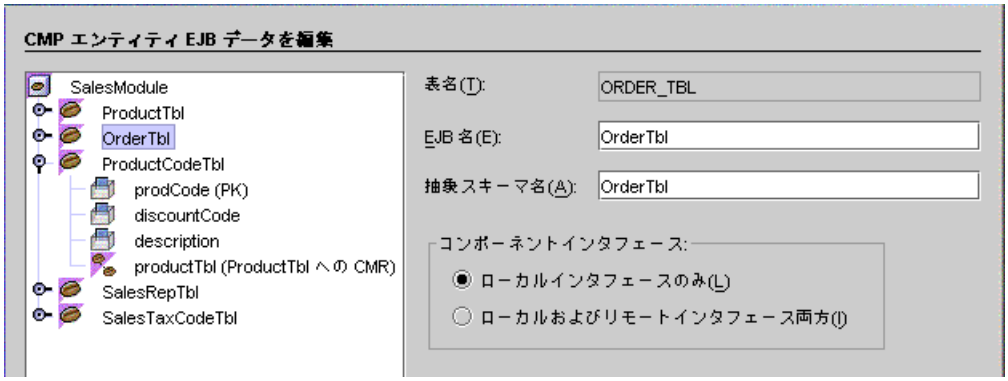
この区画では、CMP エンティティ Bean、フィールド、または 2 つの Bean 間のコンテナ管理による関係 (CMR) を選択・編集できます。Bean とそのフィールドには、IDE によってデフォルトの名前と型が割り当てられていますが、必要であれば変更することもできます。

注 - この区画は参考のために示しているだけです。この区画での操作は、ウィザードを終了した後に、関係 CMP エンティティ Bean のセットの論理ノードと EJB モジュールを使っても実行できます。

---

5. **必要に応じて、1 つまたは複数の CMP エンティティ Bean を編集します。**

   表示されている Bean ノードを選択すると、ウィンドウの表示が変わります。次に例を示します。



CMP フィールドを表すアイコン ( ) と、コンテナ管理による関係を表すアイコン ( ) があります。

このウィンドウでは次の項目を変更できます。

- (オプション)「EJB 名」フィールドで、選択した CMP エンティティ Bean の名前を変更できます。

  加えた変更は、EJB ビルダーのウィザードによって Bean クラス、適切なインタフェース、および Bean 間の関係に反映されます。

- (オプション)「抽象スキーマ名」フィールドで、選択した Bean の抽象スキーマの名前を変更できます。

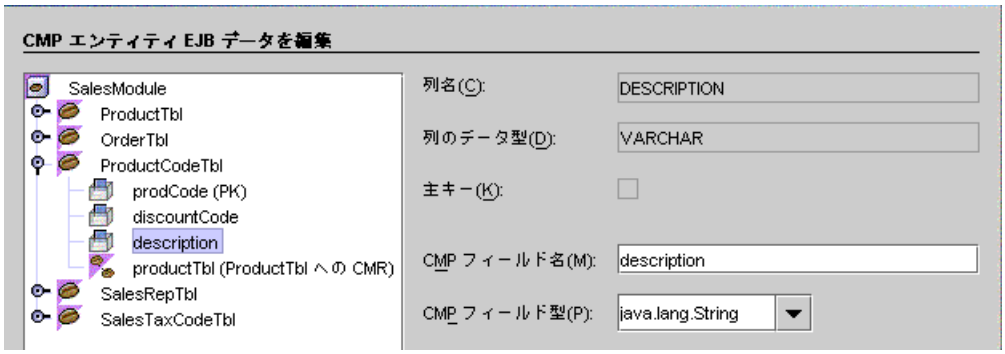
  関係 CMP エンティティ Bean のセットを指定すると、そのセットの配備記述子の一部も自動的に作成されます。作成されるのは、Bean の持続性を処理するためのコンテナに対する宣言型命令です。この命令を、抽象持続性スキーマまたは抽象スキーマといいます。後で検索メソッドまたは選択メソッドを追加すると (110 ページの「検索メソッドの追加」を参照してください)、そのメソッドに含まれる EJB QL による照会処理でこの抽象スキーマ名が使用されます。
  スキーマには異なる名前を指定することもできますが、基本的にはデフォルトの Bean 名を使用することをお勧めします。

- (オプション)「コンポーネントインタフェース」ラジオボタンを使用して、選択した Bean に異なるインタフェースを指定できます。

「ローカルおよびリモートインタフェース両方」を選択した場合を除き、モジュール内の各 CMP エンティティ Bean には自動的にローカルインタフェースとローカルホームインタフェースだけが作成されます。CMP エンティティ Bean が他のコンテナ (厳密にいうと他の JVM) にある Bean に使用されることがわかっている場合は、ローカルインタフェースとリモートインタフェースの両方が必要です。

6. 必要に応じて、1 つまたは複数の CMP フィールドを編集します。

   CMP フィールドを選択すると、ウィンドウの表示が変わります。次に例を示します。



   このウィンドウでは次の項目を変更できます。

   - (オプション)「CMP フィールド名」フィールドで、デフォルトの名前を別の名前に変更できます。

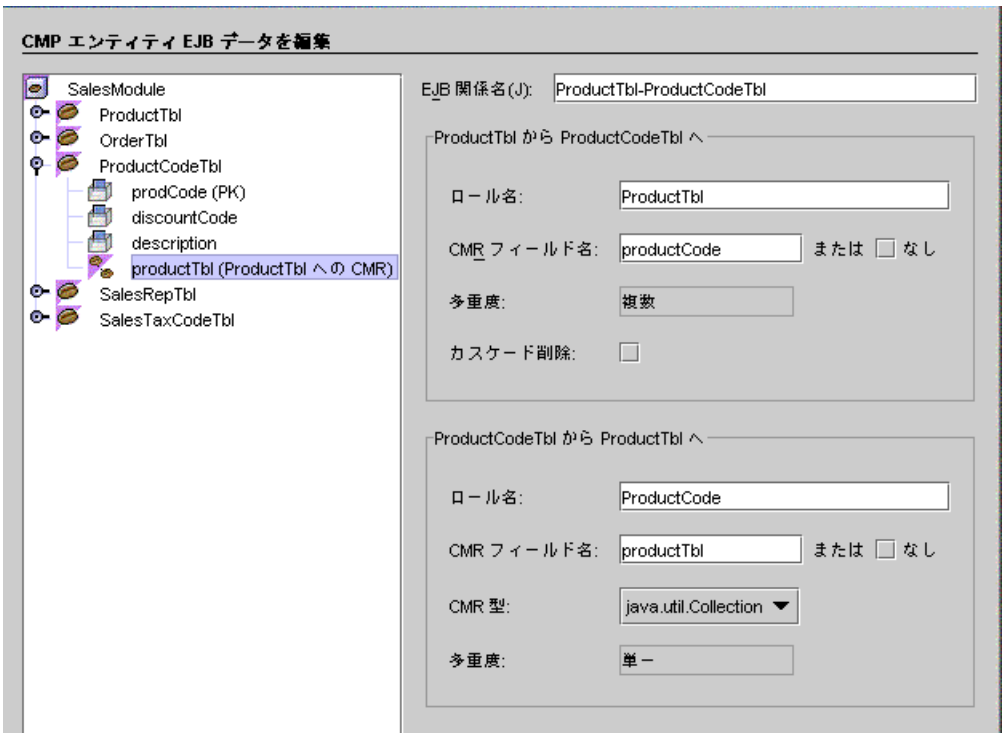
     加えた変更は、EJB ビルダーのウィザードによって Bean クラスと適切なインタフェースに、このフィールドと他のフィールドの関係を保ったまま反映されます。

   - (オプション)「CMP フィールド型」フィールドで、別のフィールドの型を指定できます。

7. 必要に応じて、2 つの CMP エンティティ Bean の関係を編集します。

   EJB ビルダーのウィザードは、2 つの Bean の間の関係を異なるノードとして表示します。このノードは実際のオブジェクトを表すものではありません。これは論理ノードであり、この論理ノードが指す Bean と選択した Bean との間に、コンテナ管理による関係 (CMR) があることを示すものです。複数の CMP エンティティ Bean 間での関係は、外部キーを持つ複数の表の間での関係に似たものです。

次に示す例では、`ProductCodeTbl` Bean は `ProductTbl` Bean に対する CMR を持っています。これは、それぞれ対応する表が共通の外部キーを持っているためです。CMR を選択すると、ウィンドウの表示が変わります。次に例を示します。



このウィンドウでは次の項目を変更できます。

- (オプション)「EJB 関連名」フィールドで、ウィザードが 2 つの Bean の関係に割り当てた名前を変更できます。
- (オプション)「ロール名」フィールドでロール名を変更できます。

  区画の右側には 2 つの CMP エンティティ Bean が表示されます。ロール名は、この区画の上部に表示されている Bean が、下部に表示されている Bean に対して持つ役割を説明するものです。

- (オプション)「CMR フィールド名」フィールドでフィールドの名前を変更できます。

  ウィザードは、各 Bean がそれぞれの関係をナビゲートできるように CMR フィールドに名前を与えます。たとえば、外部キーはこの CMR フィールドに対応付けられていることがあります。関係には、単方向 (2 つの関連する CMP エンティティ Bean の間に CMR フィールドが 1 つだけあること) と、双方向 (2 つの関連する CMP エンティティ Bean の間に CMR フィールドが 2 つあること) の 2 種類があります。ここで示している例での関係は双方向なので、各 Bean は異なる名前の CMR フィールドを 1 つずつ持っています。

  CMR フィールド名は、Bean クラスでの抽象メソッドとなります。この抽象メソッドはエンティティに対しては直接影響を与えません。

  2 つの CMP エンティティ Bean がその間に複数の関係を持っていた場合、CMR フィールド名にはそれぞれの関係を意味する名前を付けておくと便利です。この処理はこのウィンドウで実行できます。

- (オプション) ある Bean の関係に関するレコードが削除されたときに、対応する他の Bean のレコードも削除させたい場合は、「カスケード削除」チェックボックスを選択します。

  これを指定するかどうかは、2 つの Bean の関係がどのような意味を持つかによって決まります。たとえば、関連したいくつかの商品に対して注文が発生したとします。この注文と商品の関係には、常にカスケード削除を指定できます。これは、注文がなければ商品との関係がないからです。ただし、ピア関係では、参照される側の Bean が削除されても、参照する側の Bean が削除されないようにしておく必要があります。

  2 つの CMP エンティティ Bean の関係を操作するために、EJB コンテナが CMR フィールドを使用する場合は、この参照に関する完全性は EJB コンテナによって正しく保持されます。

8. 操作が終了したら「完了」をクリックします。

EJB ビルダーによって、関係 CMP エンティティ Bean のセットのインフラストラクチャ (Bean クラス、指定したインタフェース型、および Bean 間の関係) が自動的に生成されます。次の操作を、130 ページの「CMP エンティティ Bean のコンポーネント」で説明します。

データスキーマオブジェクトの使用

最初のウィザードの区画で「データベーススキーマオブジェクトからの表」を選択すると、「データベーススキーマオブジェクトを選択」区画が表示されます (図 5-1 を参照)。IDE のエクスプローラからスキーマにアクセスできることを確認してください。

「データベーススキーマオブジェクトを選択」区画には、エクスプローラからアクセスできるファイルシステムが表示されます。使用したいスキーマが表示されていることを確認してください。その後、次の操作を実行します。

1. 関係 CMP エンティティ Bean のセットで表現したい表が含まれているデータベーススキーマを選択し、「次へ」をクリックします。

   データベーススキーマ中で使用可能な表が含まれている「使用可能な表」と空白の「選択した表」が 2 つ並んで表示されます。

2. 左側に表示される「使用可能な表」の一覧から使用したい表を選択し、右側の「選択した表」の一覧に加えます。

   この時点から、操作手順と使用する GUI 機能は 123 ページの「データベース接続の使用」で説明した手順と同じになります。

   a. 使用したい表をすべて選択したら「次へ」をクリックします。

   b. 警告ダイアログで、関係 CMP エンティティ Bean のセットに含めたくない表のチェックボックスからチェックを外します。「了解」をクリックします。

   c. 「CMP エンティティ EJB データを編集」区画で、Bean、フィールド、または関係を必要に応じて編集します。

3. 操作が終了したら「完了」をクリックします。

   関係 CMP エンティティ Bean のセットのインフラストラクチャが EJB ビルダーによって自動的に生成されます。

CMP エンティティ Bean のコンポーネント

EJB ビルダーのウィザードは、基本の CMP エンティティ Bean クラスを自動的に生成し、すべての Bean とそれらのクラスとの間の関係を設定します。図 5-2 では、エクスプローラの「ファイルシステム」タブに表示された、関係 CMP エンティティ Bean の一般的なセットとそれらが格納される EJB モジュールの例を示しています。この例

では、CMP エンティティ Bean に対してデフォルトの「ローカルインタフェースのみ」が選択されています。これが選択されているとオブジェクト間の参照は同じコンテナ内に限られますが、「ローカルおよびリモートインタフェース両方」が選択されていると、他の CMP エンティティ Bean との参照もできるようになります。

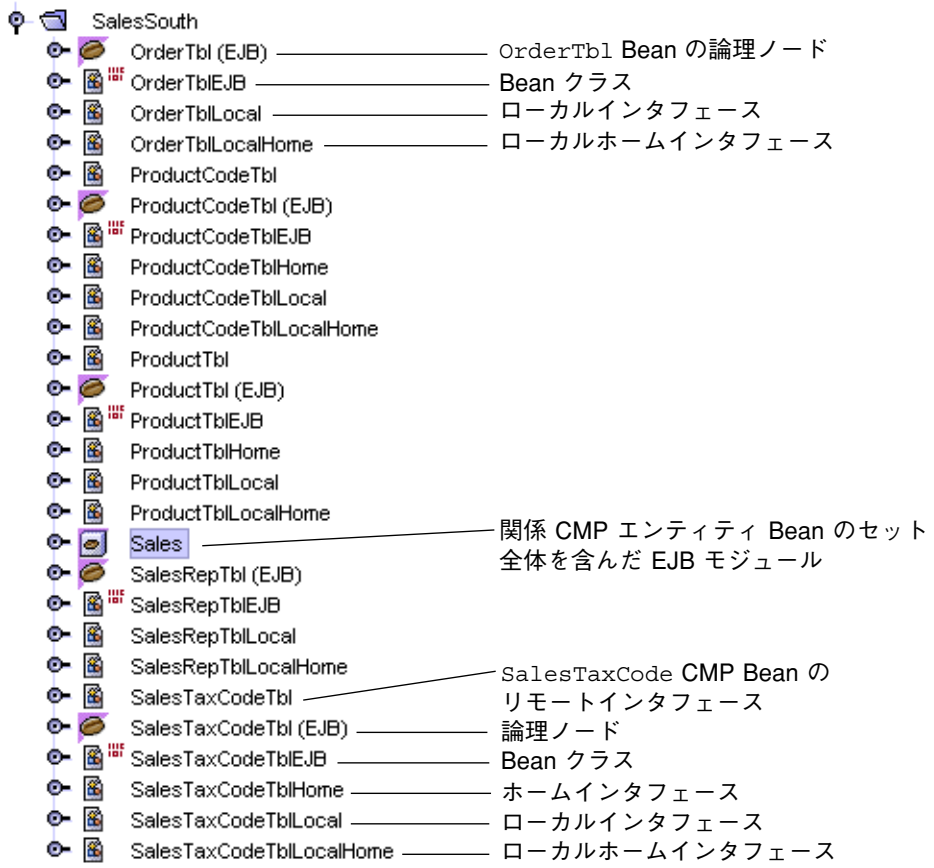


図 5-2　関係 CMP エンティティ Bean の一般的なセットのデフォルトクラス

EJB モジュールノードを除き、ノードは 95 ページの「CMP エンティティ Bean のクラス」で説明したコンポーネントと同じものを表しています。各 Bean の論理ノードも、同じように Bean アイコンで示されています。必要な編集は論理ノードで実行してください。

EJB モジュールのノードの展開

関係 CMP エンティティ Bean のセット内での Bean 間の関係には違いがあります。これらの関係は 、図 5-3 で示すように EJB モジュールのレベルに保存・表示されます。

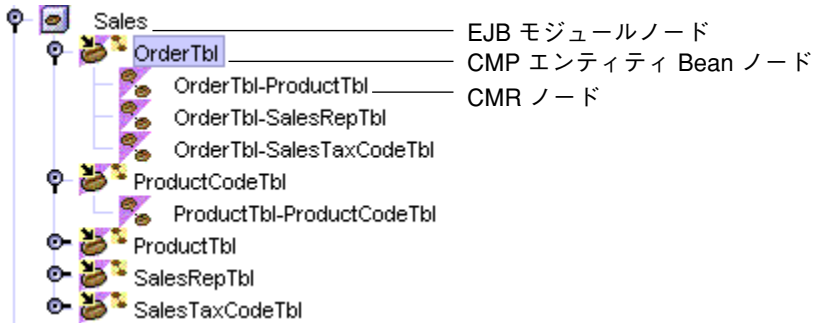


図 5-3　関係 CMP エンティティ Bean を含んだ EJB モジュールのノードの展開図

モジュールのコンポーネント Bean のノードと関係のノードが異なることに着目してください。EJB モジュール内に表示された Bean は単なる論理的なリンクで、実際の Bean のコピーではありません。

生成されたクラスの確認

第 4 章で説明したデフォルトのメソッドは、自動的にすべての CMP エンティティ Bean 内に作成されます。これについての詳細は 99 ページの「生成されたクラスの確認」の説明を参照してください。

関係 CMP エンティティ Bean のセットの完成

関係 CMP エンティティ Bean のセットを完成させるには、次の操作を実行します。

- セットに必要となるその他の CMP エンティティ Bean を、セット内の既存の Bean に対する関係と共に追加します。
- 必要に応じて CMR を編集します。

- データベースへのデータの挿入を実行するクライアントを持つ Bean を作成したい場合は、Bean に生成メソッドを定義します。1 つのエンティティ Bean には複数の生成メソッドを指定できます。生成メソッドの追加操作は、 単独の CMP エンティティ Bean であっても、セット内の関連した Bean であっても同じです。105 ページの「生成メソッドの定義」での操作に従ってください。
- 必要に応じて、主キーを追加または変更します。この方法もすべての CMP エンティティ Bean と同じです。操作手順については、107 ページの「主キーの追加または置き換え」を参照してください。
- 各 Bean が必要とするすべてのビジネスメソッドを、109 ページの「ビジネスメソッドの定義」での説明に従って定義します。
- `findByPrimaryKey` 以外に Bean が必要とする検索メソッドを定義します。操作手順は 110 ページの「検索メソッドの追加」で説明しています。
- Bean が実行する処理を、提供されている Bean インスタンスに依存させない場合は、ホームメソッドを 1 つ以上定義します。112 ページの「ホームメソッドの定義」を参照してください。
- CMP エンティティ Bean に、同じ EJB モジュール内にある他の Bean に照会を実行させたい場合、またはデータベースに照会させて結果をローカルまたはリモートインタフェースを介して取得したい場合は、選択メソッドを 1 つ以上定義します。
- Bean の `setEntityContext`、`unsetEntityContext`、`ejbActivate`、`ejbPassivate`、および `ejbRemove` の各メソッドにコードを追加し、これらのメソッドを完成させます。

Bean が必要とする CMP フィールドのうち、一部だけが自動生成された場合は、残りを追加します。

こういった基本的な追加処理は、IDE の GUI ツールを使用して論理ノードから実行します。これらのメソッドの内容を次の手順で指定します。

- ダイアログでメソッド名を指定し、メソッドのシグニチャを定義します。論理ノードを選択し、右クリックして「生成メソッドを追加」、「ビジネスメソッドを追加」、「検索メソッドを追加」、「ホームメソッドを追加」または「選択メソッドを追加」のいずれかを選択します。定義したメソッドが CMP エンティティ Bean の適切なクラスに保存されます。
- ソースエディタを使用してメソッドの本体を完成させます。

推奨する エンタープライズ Bean の開発手順

付録 A では、エンタープライズ Bean に変更を加える際の最適な方法や、推奨手順以外の方法を行った場合に発生する可能性のあるエラーや異常状態について説明しています。一般に、個々のクラスノードを操作するよりは論理ノードを操作する、メソッドの変更には Bean のプロパティシートや「カスタマイザダイアログ」を使用する、およびダイアログからでは操作できない Bean のコードを完成させたり、編集したりするには IDE のソースエディタを使用するといった方法をお勧めします。

セットへの Bean の追加

関係 CMP エンティティ Bean のセットを生成後に、これまでの操作で選択しなかった別のデータベースの表に対応する新しい Bean をセットに追加したい場合は、次の操作を実行します。

1. セットに追加したい CMP エンティティ Bean を決定します。

   追加したい CMP エンティティ Bean はウィザードによってすでに生成されているはずです。これは、ウィザードが既存のセット用に CMP エンティティ Bean をすでに生成しているためです。したがって、追加する CMP エンティティ Bean は単独の Bean か、他の関係 CMP エンティティ Bean のセットに含まれているものを選択します。

2. 目的の関係 CMP エンティティ Bean が含まれている EJB モジュールノードを選択し、右クリックして「EJB を追加」を選択します。

   「EJB を EJB モジュールに追加」ダイアログが表示されます。IDE のエクスプローラウィンドウにマウントされているすべてのファイルシステムが、ツリーで表示されます。

3. セットに追加したい CMP エンティティ Bean を選択し、「了解」をクリックします。

   関係 CMP エンティティ Bean のセットに Bean が追加されます。

4. EJB モジュールノードを展開して、関係 CMP エンティティ Bean のセットに CMP エンティティ Bean が追加されたかどうか確認します。

   追加した Bean が表示されますが、他の Bean への関係はまだ定義されていません。次に `Customer` という Bean を追加した例を示します。

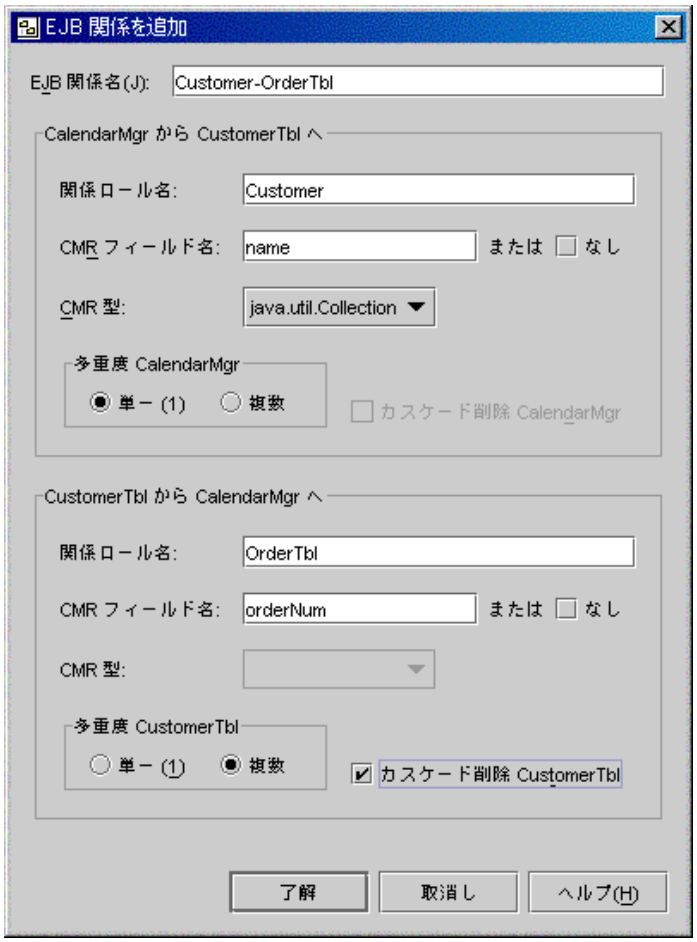
次に、新しい CMP エンティティ Bean またはセット内の他の Bean に関係を追加します。

5. EJB モジュールノードから、関係を持たせたい 2 つの CMP エンティティ Bean を選択します。

   この例では、 `Customer` Bean と `OrderTbl` Bean のノードを選択します。

6. 右クリックして「EJB 関係を追加」を選択します。

   「EJB 関係を追加」ダイアログが次のように表示されます。ダイアログには、2 つの CMP エンティティ Bean 間の関係の名前を示すフィールド、一方のエンティティ Bean に対する説明、および他方のエンティティ Bean に対する説明の 3 つが主に表示されます。



7. 2 つの CMP エンティティ Bean の関係を定義します。

   フィールドには、選択された 2 つの Bean に関する既存の情報に基づいたデフォルトの情報が表示されています。必要に応じて次の項目を変更します。

   - (オプション)「EJB 関係名」フィールドで、2 つの Bean 間の関係の名前を変更します。
   - (オプション) それぞれの Bean の「関係ロール名」フィールドで、各 Bean が関係について果たす役割の名前を変更します。
   - (オプション) それぞれの Bean の「CMR フィールド名」フィールドで、2 つの Bean を関連付けるフィールドの名前を変更します。

- (オプション) 各 Bean の「CMR 型」フィールドで、別の型を選択します。CMR フィールド名を変更したくない場合は、CMR 型はそのままにしておいてください。
- (オプション) それぞれの Bean の「多重度」では、関係のある Bean の濃度を変更します。「CMR フィールド名」フィールドまたは 「CMR 型 」フィールドを変更したくない場合は、この多重度はそのままにしておいてください。「複数」と表示されたラジオボタンを選択すると、「カスケード削除」チェックボックスが有効になります。

8. 操作が終了したら「完了」をクリックします。

   EJB モジュールノードで、関係を追加した Bean のメインアイコンに関係マーク が表示されます。

関係 CMP エンティティ Bean を作成した後の作業

作成した CMP エンティティ Bean は、最終環境で使用できるようにする必要があります。これらの最終作業については、第 8 章を参照してください。

また、付録 A では、推奨する エンタープライズ Bean の操作手順を説明しています。

第6章

BMP エンティティ Bean の開発

これまでの 2 つの章では、持続性管理を EJB コンテナに依存するエンティティ Bean の開発について説明しました。この章では、自身の持続性を管理するコードをすべて含んだエンティティ Bean を作成し、完成させる方法について説明します。これを Bean 管理による持続性 (BMP) Bean といいます。CMP Bean と BMP エンティティ Bean の開発には似た点が多くあります。この章では、主に違いについて説明します。

BMP エンティティ Bean に必要となるさまざまなクラスは、Sun ONE Studio IDE の提供するウィザードを使用して作成できます。作成されるクラスには、Bean クラス、リモートインタフェースかローカルインタフェース、または両方、そして必要に応じて、主キークラスがあります。このような BMP エンティティ Bean のインフラストラクチャの作成処理は、EJB ビルダーによって自動化されています。

エンティティ Bean を作成する場合、この章に記述された以外の方法を取ることもできます。Sun ONE Studio IDE は、コーディング処理のほとんどを自動化していますが、プログラマが自身で多くを決定できるように柔軟な手段も提供しています。詳細については、xviii ページの「お読みになる前に」で紹介しているマニュアルを参照するか、市販の エンタープライズ Bean のプログラミングに関する文献を参照してください。

作成方法の決定

IDE を使用したエンティティ Bean の作成方法にはいくつかありますが、77 ページの「CMP エンティティ Bean 作成のための EJB ビルダーの使用」で推奨されている方法に従うと、わかりやすい機能を利用しながら短期間で Bean を完成できます。この操作方法は、IDE の機能を最大限に活用した、Bean の一貫性と J2EE 標準への準拠を確実にする手法です。

エンティティ Bean による持続性の管理が必要かどうかわからない場合は、表 4-1 を参照してください。

BMP エンティティ Bean の構築

EJB ビルダーのウィザードは、BMP エンティティ Bean のデフォルトクラスの作成の一部を自動化します。デフォルトクラスはウィザードによって自動的に生成されます。ただし、ウィザードは、その BMP エンティティ Bean とデータベースとの対話処理の内容はまったく考慮しません。したがって、デフォルトクラスの初期設定はごく単純なものです。BMP エンティティ Bean を作成するには、次の操作を実行します。

1. BMP エンティティ Bean を格納するパッケージを選択します。
2. EJB ビルダーのウィザードを使用して、BMP エンティティ Bean のインフラストラクチャを生成します。
3. 必要に応じて、主キークラスを Bean に追加します。
4. 必要に応じて、生成メソッド、ビジネスメソッド、検索メソッド、選択メソッド、およびホームメソッドを Bean に追加します。
5. 追加したメソッドの本体を完成させます。
6. 持続性のコードを記述します。データベース中のデータに関係するメソッドをすべて完成させます。

それぞれの手順について、次に説明します。

パッケージの作成

エンティティ Bean を格納するパッケージの作成が必要な場合は、ファイルシステムを選択し、右クリックして「新規」->「Java パッケージ」を選択します。

EJB ビルダーのウィザードの起動

BMP エンティティ Bean を作成するには、次の操作を実行します。

1. IDE のメインウィンドウから「表示」->「エクスプローラ」を選択し、IDE のエクスプローラウィンドウを開きます。

2. エクスプローラの「ファイルシステム」タブから、セッション Bean を格納する Java パッケージを選択します。
3. 右クリックし、「新規」->「J2EE」->「BMP エンティティ EJB」を選択します。

   EJB ビルダーのウィザードが表示されます。

BMP エンティティ Bean のインフラストラクチャの生成

ウィザードの BMP エンティティ EJB 区画で、次の操作を実行します。

1. BMP エンティティ Bean の名前を指定します。
2. BMP エンティティ Bean に、ローカルインフェースだけを指定するのか (デフォルト)、リモートインタフェースだけを指定するのか、または両方を指定するのかを決定します。

   必要に応じて、Bean のパッケージの保存先を変更できます。
3. 「次へ」をクリックします (この手順を省いて、その次の手順に進むこともできます)。

   「BMP エンティティ Bean クラスファイル」区画に、BMP エンティティ Bean 用に生成されるクラスファイルが表示されます。必要に応じて次の操作を実行します。
   - 各項目の変更ボタンを使用し、既存のクラスの名前を変更したり、新規クラスに新しい名前を指定できます。これは、ホームインタフェースとリモートインタフェースがすでに指定されている Bean を実装中に、新しい Bean クラスを追加したくなったときなどに使用できます。
   - 各項目の変更ボタンをクリックすることで、スーパークラスを変更できます。この操作を実行する際は、正しいインタフェースのサブクラスを選択してください。
4. 操作が終了したら「完了」をクリックします。

   ウィザードによって、BMP エンティティ Bean のデフォルトクラスが生成されます。これらのクラスについて次に説明します。

BMP エンティティ Bean のクラス

EJB ビルダーのウィザードは、BMP エンティティ Bean に必要なエンティティ Bean クラスすべてを作成し、クラス間に必要な通信を設定します。ただし、持続性を保つロジックはプログラマが記述する必要があります。

エクスプローラの「ファイルシステム」タブでは、BMP エンティティ Bean は CMP エンティティ Bean とほぼ同じように表示されます。違いは、BMP エンティティ Bean の論理ノードにカーソルを置くと「BMP エンティティ Bean 論理ノード」と表示される点です。

クラスアイコン が付いたノードは実際のクラスを表しています。Bean アイコン が付いたノードは論理ノードです。編集はすべて論理ノードから実行してください。

BMP エンティティ Bean のクラスの実装は、CMP エンティティ Bean のクラスと同じです。ただし、BMP エンティティ Bean は abstract でなく、public として定義されます。

ノードの展開

BMP エンティティ Bean のパッケージノードを展開すると、図 6-1 のように表示されます。この例での Bean にはローカル型インタフェースが指定されています。BMP エンティティ Bean には、選択メソッドはありません。

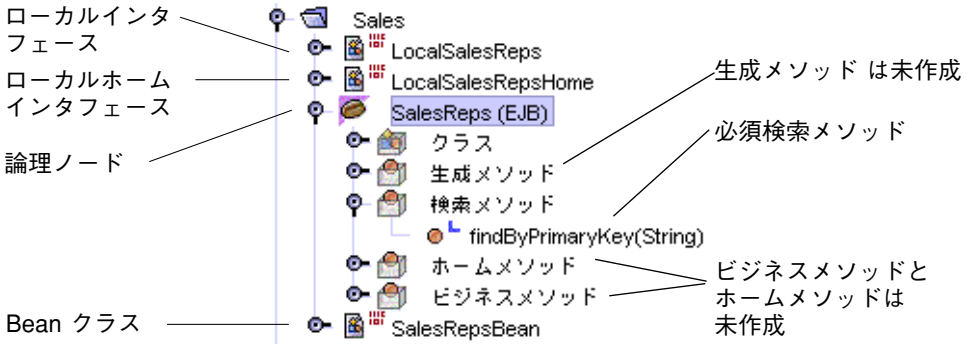


図 6-1　BMP エンティティ Bean のエクスプローラでの詳細表示

主キークラスが生成された場合は、それもエクスプローラのノードとして表示されます。

生成されたクラスの確認

EJB ビルダーのウィザードは、各エンティティ Bean にいくつかのデフォルトメソッドを追加します。

findByPrimaryKey メソッド

BMP エンティティ Bean のホームインタフェースに自動的に追加されるメソッドシグニチャ `findByPrimaryKey` を次の例に示します。

```
public customer findByPrimaryKey(String aKey)
    throws RemoteException, FinderException;
```

これが BMP エンティティ Bean であるため、ウィザードはこのメソッドの対となる `ejbFindByPrimaryKey` を Bean クラスに追加します。

```
public String ejbFindByPrimaryKey(String aKey) {
    return aKey;
}
```

BMP エンティティ Bean のライフサイクルメソッド

BMP エンティティ Bean の Bean クラスにウィザードが追加するデフォルトのライフサイクルメソッドを コード例 6-1 に示します。

コード例 6-1　BMP エンティティ Bean のデフォルトのライフサイクルメソッド

```
public void setEntityContext(javax.ejb.EntityContext aContext) {
    context=aContext;
}
public void ejbActivate() {
}
public void ejbPassivate() {
}
public void ejbRemove() {
}
```

コード例 6-1　BMP エンティティ Bean のデフォルトのライフサイクルメソッド (続き)

```
public void unsetEntityContext() {
    context=null;
}
public void ejbLoad() {
}
public void ejbStore() {
}
```

BMP エンティティ Bean クラスに含まれるこれらのメソッドの目的を表 6-1 に示します (CMP エンティティ Bean クラスでの目的と比較したい場合は、表 4-2を参照してください)。

表 6-1　BMP エンティティ Bean クラスでのライフサイクルメソッドの目的

メソッド	目的
`setEntityContext`	このメソッドは、フィールドに `EntityContext` の参照を格納し、非持続フィールドに値を格納できるようにする。このメソッドを使用して、EJB オブジェクトに依存せず、エンティティ Bean の全期間にわたって存続するリソース (データベース接続ファクトリなど) を割り当てることができる。デフォルトでは、`context` というフィールドに `EntityContext` を代入するコードが生成される
`ejbActivate`	このメソッドは、Bean を初期化して使用できるようにし、インスタンスに必要なリソースを取得する
`ejbPassivate`	このメソッドは、Bean のインスタンスが汎用インスタンスプールに戻される前に、その Bean が使用していたリソースを解放する
`ejbRemove`	BMP では、このメソッドは `SQL DELITE` 文を実行し、データストレージからデータを削除する。また、データアクセスオブジェクト (DAO: Data Access Object) などの他のオブジェクトを呼び出してデータを削除することもできる

表 6-1　BMP エンティティ Bean クラスでのライフサイクルメソッドの目的 (続き)

メソッド	目的
unsetEntityContext	コンテナがエンティティ Bean インスタンスを破棄する前に、そのインスタンスが使用していたメモリやリソースを解放する
ejbLoad	BMP では、このメソッドは SQL SELECT 文を実行し、データソースから Bean インスタンスにデータを読み込む。この処理は、Bean が有効になったとき、またはエンティティが新しいトランザクションのコンテキスト中に参照されたときに実行される。また、DAO などの他のオブジェクトを呼び出して、データを読み込むこともできる
ejbStore	BMP では、このメソッドは SQL UPDATE 文を実行し、Bean の状態 (現在の保存先は持続フィールド) をデータストレージに保存する。この処理は、Bean が非活性化状態になったとき、またはトランザクションがコミットされたときに実行される。また、DAO などの他のオブジェクトを呼び出して、データを保存することもできる。

BMP エンティティ Bean の完成

BMP エンティティ Bean を完成させるには、次の操作を実行します。

- すべての持続性ロジックを追加します。
- BMP エンティティ Bean が複合主キーを持つ場合は、主キークラスを追加します。
- データベースへのデータの挿入を実行するクライアントを持つ Bean を作成したい場合は、Bean に生成メソッドを定義します。1 つのエンティティ Bean には複数の生成メソッドを指定できます。
- findByPrimaryKey 以外に Bean が必要とする検索メソッドを定義します。その後、すべての検索メソッドの本体を完成させます。
- データベースからのレコードの削除処理には、ejbRemove メソッドを記述します。
- BMP エンティティ Bean に必要なすべてのビジネスメソッドとホームメソッドを定義・完成させます。

- エンティティの状態をメモリーに保持し、これらのフィールドの値を格納するには `private` フィールドを追加します。

推奨する エンタープライズ Bean の開発手順

付録 A では、エンタープライズ Bean に変更を加える際の最適な方法や、推奨手順以外の方法を取った場合に発生する可能性のあるエラーや異常状態について説明しています。一般に、個々のクラスノードを操作するよりは論理ノードを操作する、メソッドの変更には Bean のプロパティシートや「カスタマイザダイアログ」を使用する、およびダイアログからでは操作できない Bean のコードを完成させたり、編集したりするには IDE のソースエディタを使用するといった方法をお勧めします。

持続性ロジックの追加

エンティティ Bean にエンティティデータストアとの対話処理を実行させるには、データへのアクセス、持続フィールドの操作、Bean インスタンスの変数とデータストアとの間でのデータ転送に必要なコードを記述する必要があります。こういったコードはソースエディタを使用して記述してください。Bean が使用するデータソースの指定は、リソースの参照 (第 8 章で説明しています) を利用してください。

主キークラスの追加

次の場合は、ソースエディタを使用して主キークラスを追加してください。

- BMP エンティティ Bean の作成時に主キークラスを作成しなかったが、Bean に主キークラスが必要になった。
- 他の既存のクラスでは代用できないような主キーが必要になった。
- 主キーに `java.lang.String` 以外の型、または既存の主キークラス以外が指定されている。
- `equals` メソッドと `hashcode` メソッドの定義にカスタマイズが必要である。
- データベースでキーを使用する前に値が有効かどうかを検査するなど、主キーに機能を付加したい。

また、主キークラスは次の要件を満たしている必要があります。

- クラスにアクセス制御修飾子 `public` がある。

- すべてのフィールドが public として宣言されている。
- クラスに public デフォルトコントラクタがある。
- クラスが hashCode メソッドと equals メソッドを実装している。
- クラスが直列化可能である。つまり、 java.io.Serializable インタフェースを実装している。
- クラスが java.rmi.Remote インタフェースを実装していない。

以上の項目の詳細については、107 ページの「主キーの追加または置き換え」を参照してください。

メソッドの追加

新しいメソッドを定義するには、エクスプローラを開いて、論理ノードを右クリックし、コンテキストメニューの GUI ツールを利用します。メソッドに名前を指定したり、シグニチャを定義するには、ダイアログを使用します。IDE は、メソッドを自動的に正しいクラスに伝播します。最後にソースエディタでメソッドのコードを完成させます。

生成メソッドの定義

BMP エンティティ Bean のホームインタフェースには生成メソッドを指定できます。生成メソッドを指定する場合は、Bean クラスには対となる ejbCreate メソッドと ejbPostCreate メソッドが必要です。推奨される手順にしたがっていれば、これらのメソッドは IDE によって生成され、正しく伝播されます。

BMP エンティティ Bean の ejbCreate メソッドは、一般的に次の処理を実行します。

1. クライアントが提供する引数の妥当性を検査する。
2. インスタンスの変数を初期化する。
3. SQL INSERT 文を実行する (また、DAO などの他のオブジェクトを呼び出して、データストレージにデータを挿入できる)。
4. 主キーを戻す。

プログラマは、BMP エンティティ Bean に SQL INSERT 文を生成して実行するのに必要なコードを記述する必要があります。

IDE によって自動的に追加される `ejbPostCreate` メソッドを使用すると、(ホームインターフェースやリモートインタフェースなど) EJB オブジェクトに関する情報を、その情報の参照を必要とする他の エンタープライズ Bean に転送させることができます。メソッドは、コンテナからパラメータとして受け取る `EntityContext` を介して、リモートインタフェースにアクセスできます。このメソッドは、依存する Bean の作成に使われるのが一般的です。たとえば、`Order` Bean の `ejbCreateLineItem` メソッドは、 `ejbPostCreate` メソッド内に指定された商品を作成する、というように使用されます。

エンティティ Bean には 1 つ以上の生成メソッドを指定できます。新しい生成メソッドを定義するには、次の操作を実行します。

1. 論理ノードを選択し、右クリックしてから「生成メソッドを追加」を選択します。

   「新規生成メソッドを追加」ダイアログが表示されます。

2. `create` に続けて、生成メソッドに名前をここで指定する必要があります。

   生成メソッドのパラメータを追加します。

3. ダイアログで「追加」をクリックします。

4. 「メソッドのパラメータを入力」ダイアログで、パラメータの名前と型を指定します。

   BMP エンティティ Bean クラスのメソッドシグニチャとメソッドの本体は、主キー型を返します。

5. 「了解」をクリックし、「メソッドのパラメータを入力」ダイアログを閉じます。

6. 「新規生成メソッドを追加」ダイアログで、例外を追加します。

7. 「了解」をクリックし、「新規生成メソッドを追加」ダイアログを閉じます。

   追加したメソッドが Bean クラスのコードに `ejbCreate` として、ホームインタフェースに `create` として表示されます。Bean クラスには `ejbPostCreate` メソッドも表示されます。

8. ソースエディタを使用して、 return 文と生成メソッドに必要なその他のコードを追加します。

検索メソッドの追加

この時点で、デフォルトの検索メソッドは EJB によって作成されています。BMP エンティティ Bean では、このメソッドはホームインタフェース (`findByPrimaryKey`) と Bean クラス (`ejbFindByPrimaryKey`) の両方に表示されます。エンティティ Bean にこれ以外の照会をさせたい場合は、追加の検索メソッドを定義する必要があります。

次の操作を実行すると、新しく作成した検索メソッドはホームインタフェースと Bean クラスに自動的に伝播されます。

1. **論理ノードを選択し、右クリックして「検索メソッドを追加」を選択します。**
2. **「`find`」で始まるメソッド名を入力します。パラメータ、例外、および戻り値の型を指定し、「了解」をクリックします。**

   ソースエディタを使用して検索メソッドのコードを完成させます。データソースから主キーを取得する場合は、JDBC を記述するか、その他のデータベースへのアクセス手段を使用する必要があります。

ビジネスメソッドとホームメソッドの定義

ビジネスメソッドを BMP エンティティ Bean に追加するには、次の操作を実行します。

- **論理ノードを選択し、右クリックして「ビジネスメソッドを追加」を選択します。**

  「新規ビジネスメソッドを追加」ダイアログが表示されます。このダイアログで、メソッド名のパラメータと例外を指定できます。または、新しいビジネスメソッドに名前を指定してから「了解」クリックし、残りのコーディングをソースエディタで記述することもできます。

  ビジネスメソッドは一般的に、持続フィールドの値へのアクセスと変更を実行します。データベースに直接アクセスすることはありません。EJB コンテナは、トランザクションセマンティクスの要求に応じて `ejbLoad` メソッドと `ejbStore` メソッドを呼び出します。

  エンティティ Bean に提供されているインスタンスに依存しない操作を実行する場合にも、ホームメソッドを追加できます。ホームメソッドの詳細については 112 ページの「ホームメソッドの定義」を参照してください。

BMP エンティティ Bean を作成した後の作業

作成した BMP エンティティ Bean は、最終環境で使用できるようにする必要があります。これらの最終作業については、第 8 章を参照してください。

また、付録 A では、完成した エンタープライズ Bean の推奨する操作方法を説明しています。

詳細情報の参照先

エンタープライズ Bean は、アプリケーションに高い機能性と柔軟性を付加します。また、エンタープライズ Bean の基本要素は簡単に作成できます。Sun ONE Studio IDE のようなツールを使用すれば更に簡易化できます。しかし、アプリケーションのニーズをすべて満たす Bean の作成は複雑なこともあります。Bean の作成に関する詳細は、次の URL からアクセスできる Enterprise JavaBeans 仕様 2.0 を参照してください。

`http://java.sun.com/products/ejb/docs.html`

第7章

メッセージ駆動型 Bean の開発

Sun ONE Studio IDE の EJB ビルダーを使用して、アプリケーションクライアントによる非同期処理要求に必要とされるメッセージ駆動型 Bean を開発することができます。この章では、メッセージ駆動型 Bean の作成と開発について説明します。これらの Bean によるトランザクションは、通常、EJB コンテナによって管理されます。しかし、必要であれば、プログラマがトランザクション管理用のコードを記述することもできます。

メッセージ駆動型 Bean を使用する目的は次のとおりです。

- **マルチタスクの実行とサポート** - アプリケーションクライアントは、メッセージを送信したら、そのメッセージへの応答を待つことなく次の処理に進むことができます。これは、クライアントが自身の利用するメッセージ駆動型 Bean を非同期的に起動できるためです。
- **信頼性** - アプリケーションが Java メッセージサービス (JMS) を使用していれば、クライアントの要求は、アプリケーション層が全面的に停止しない限り必ず処理されます。

それでも、メッセージ駆動型 Bean の利用が適さない状況もあります。次のような場合は、代替手段を選択してください。

- クライアントが要求が受信されたことを確認する必要がある、または処理結果を受け取る必要がある場合。
- 該当する処理が、実行時間帯が決まっているトランザクションの一部で、閑散期には実行できない場合。
- アプリケーションが小型で比較的単純であり、もう 1 つ層を追加することで構築、デバッグ、および実行に遅れが出る場合。

メッセージ駆動型 Bean の向き・不向きについては、42 ページの「メッセージ駆動型 Bean」を参照してください。

IDE では、メッセージ駆動型 Bean に必要な単一 Bean クラスを作成するためのウィザードが用意されています。メッセージ駆動型 Bean は単に、クライアントからメッセージを受け取り、それを使って他の Bean プロセスを開始するものです。インタフェースクラスは必要ありません。メッセージ駆動型 Bean の作成処理のほとんどはウィザードによって自動化されます。後はソースエディタとプロパティシートを利用して、Bean を完成させてください。

メッセージ駆動型 Bean を開発する場合、この章で説明する以外の方法を選択することもできます。Sun ONE Studio IDE は、コーディング処理のほとんどを自動化していますが、プログラマが自身で多くを決定できるように柔軟な手段も提供しています。詳細については、xviii ページの「お読みになる前に」で紹介しているマニュアルを参照するか、市販の エンタープライズ Bean のプログラミングに関する文献を参照してください。

EJB ビルダーとメッセージ駆動型 Bean

EJB ビルダーは、ウィザード、プロパティシート、およびエディタから構成されています。これらの機能を使用して、エンタープライズ Bean を整合性のとれた方法で簡単に作成することができます。EJB ビルダーがインストールされているかどうかを確認するには、メインウィンドウから「ツール」 > 「オプション」 > 「IDE 構成」 > 「システム」 > 「モジュール」 > 「J2EE サポート」を選択してください。モジュールのリストに「EJB 2.0 ビルダー」が表示されていて、プロパティシートの「使用可能」プロパティが True に設定されていれば、EJB ビルダーは使用できる状態になっています。

IDE でメッセージ駆動型 Bean を作成するにはいくつかの方法がありますが、この章で推奨している方法を取ることで、最もわかりやすく、短期間で Bean を完成させることができます。ここで説明する操作方法は、Bean の一貫性と J2EE 標準への準拠を確実にするために IDE の機能を最大限に活用した手法となっています。

最良の結果を得るために、EJB ビルダーを使用して、次の手順でメッセージ駆動型 Bean をプログラミングしてください。

- **Bean に必要なクラスを 1 つ作成します。** EJB ビルダーのウィザードの手順に従うと、メッセージ駆動型 Bean の枠組みができあがります。これには Bean クラスと Bean の構成要素をまとめる論理グループがあります。クラスと論理グループのノードのどちらも、それぞれのサブノードと共に エクスプローラの「ファイルシステム」タブで表示することができます。Bean クラスには、`ejbCreate` メソッドと `onMessage` メソッドの宣言がウィザードによって生成されます。メソッドの本体を記述して完成させてください。

  メッセージ駆動型 Bean は論理ノードから操作するのが最適です。エクスプローラでは、すべての論理ノードが アイコンで示されています。

- **Bean クラスを必要に応じて完成させます。**この章で後述している IDE の機能を使用してください。

- **Bean の配備記述子に値を設定します。**メッセージ駆動型 Bean のプロパティは、論理ノードからプロパティシートを使用して編集してください。

メッセージ駆動型 Bean の論理ノードからは、Bean のコードを検査できます。

トランザクション管理の決定

メッセージ駆動型 Bean を作成する前に、その Bean のトランザクションを EJB コンテナに管理させるか、そのためのコードを手動で記述するかどうかを決めます。トランザクション管理の形態が異なる Bean を作成するときには、IDE の EJB ビルダーでの操作手順が異なります。表 7-1 に、設計に関する主な考慮点を挙げます。

表 7-1　コンテナ管理または Bean 管理によるトランザクションの決定

項目	コンテナ管理による トランザクション	Bean 管理による トランザクション
トランザクション マネージャ	トランザクションマネージャは コンテナ自身である。	JTA を使用して、プログラマが トランザクションを管理する コードを記述する。これには JDBC といった他のリソースの トランザクションも含まれる ことがある。

表 7-1　コンテナ管理または Bean 管理によるトランザクションの決定 (続き)

項目	コンテナ管理による トランザクション	Bean 管理による トランザクション
トランザクション境界の設定	EJB コンテナは Java 2 Platform, Enterprise Edition 仕様に従ってトランザクションの開始・コミットの時期を決定する。	プログラマは、トランザクションの実行を細かく制御するために、トランザクション境界を明示的に記述する。
トランザクションのタイミング	メッセージ駆動型 Bean はメッセージを受信すると、そのトランザクション内でビジネスロジックを実行する。	メッセージ駆動型 Bean によるメッセージの受信後、トランザクションが開始される。
問題発生時の対応	コンテナがトランザクションをロールバックし、メッセージ駆動型 Bean にそのことを通知する。	メッセージ駆動型 Bean の作成後に指定した通知モードに従って、メッセージ駆動型 Bean は応答する。

どちらを選択するかについての詳細は、マニュアル『J2EE アプリケーションのプログラミング』のトランザクションの章を参照してください。

以降では、それぞれの種類のメッセージ駆動型 Bean の作成方法と、開発時の注意点を説明します。

メッセージ駆動型 Bean の定義

メッセージ駆動型 Bean に必要な Bean クラスの作成処理のほとんどは、EJB ビルダーウィザードによって自動化されています。メッセージ駆動型 Bean を定義するには、次の操作を実行します。

1. Bean を格納するパッケージを選択または作成します。
2. EJB ビルダーのウィザードを使用して、メッセージ駆動型 Bean のインフラストラクチャを生成します。
3. `onMessage` メソッドの本体と、必要に応じて `setMessageDrivenContext` メソッドと `ejbCreate` メソッドの本体も完成させます。

次に、それぞれの手順について説明します。

この章で説明している操作を実行してメッセージ駆動型 Bean を作成したら、次は Bean のプロパティシートに情報を追加します。この情報は、メッセージ駆動型 Bean による他の Bean との対話処理、リソースの検索、および適切なメッセージの待機に必要なものです。メッセージ駆動型 Bean をアプリケーションで実際に使用するための準備については、第 8 章で説明しています。

パッケージの作成

メッセージ駆動型 Bean を格納するパッケージを作成する場合は、ファイルシステムを選択し、右クリックして「新規」->「 Java パッケージ」を選択します。

EJB ビルダーのウィザードの開始

メッセージ駆動型 Bean を作成するには、次の操作を実行します。

1. IDE のメインウィンドウから「表示」->「エクスプローラ」を選択し、IDE のエクスプローラウィンドウを開きます。
2. エクスプローラの「ファイルシステム」タブから、メッセージ駆動型 Bean を格納するパッケージを選択します。
3. 右クリックし、「新規」->「J2EE」->「メッセージ駆動型 EJB」を選択します。
   タイトルバーに「新規ウィザード — メッセージ駆動型 EJB」と表示された EJB ビルダーのウィザードが表示されます。

基本のメッセージ駆動型 Bean の生成

EJB ビルダーの「メッセージ駆動型 Bean 名とプロパティ」区画で、メッセージ駆動型 Bean に、名前とその Bean が実行するトランザクションの管理形態を指定します。デフォルトの管理形態は「コンテナ管理によるトランザクション」です。目的に応じて、Bean クラスにトランザクションを管理するコードを記述するよう決定することもできます。

どちらかの管理形態を指定したら「完了」をクリックします (代わりに「次へ」をクリックして、次の区画でメッセージ駆動型 Bean に既存の Bean クラスを指定することもできます。Bean クラスを指定し終わったら「完了」をクリックします)。

新しく作成されたメッセージ駆動型 Bean が IDE のエクスプローラ中の「ファイルシステム」タブに表示されます。Bean のインフラストラクチャ (基本の Bean クラスと 2 つのコンポーネントメソッド) は EJB ビルダーによって自動的に生成されます。

メッセージ駆動型 Bean の参照

図 7-1 に、エクスプローラの「ファイルシステム」タブでのメッセージ駆動型 Bean の一般的な表示例を示します。

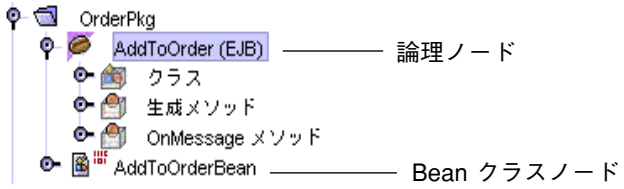


図 7-1　メッセージ駆動型 Bean のデフォルトクラスとメソッドの一般的な例

2 つの主なノードのうち、1 つは (Bean アイコンで示される) 論理ノードで、もう 1 つは (クラスアイコンで示される) 実際のアイコンを表しています。編集はすべて論理ノードから実行してください。これら 2 つのノードについて次に説明します。

- 論理ノードは、エクスプローラにメッセージ駆動型 Bean のすべての要素をまとめて表示し、各要素に対する操作をしやすくするために作成されるものです。
- Bean クラスは `javax.ejb.MessageDrivenBean` と `javax.jms.MessageListener` インタフェースを実装し、メッセージ駆動型 Bean のメソッドを公開します。

クラスノードには、両方のメソッドを含んだ Bean クラスのコードが含まれます。生成メソッドノードは、メッセージ駆動型 Bean を初期化するコードを示し、OnMessage メソッドノードは、 Bean がメッセージを受信したときに起動するメソッドを示しています。

ノードの展開

メッセージ駆動型 Bean のパッケージのノードにある 2 つのノードを展開すると、図 7-2 に示す内容が表示されます。

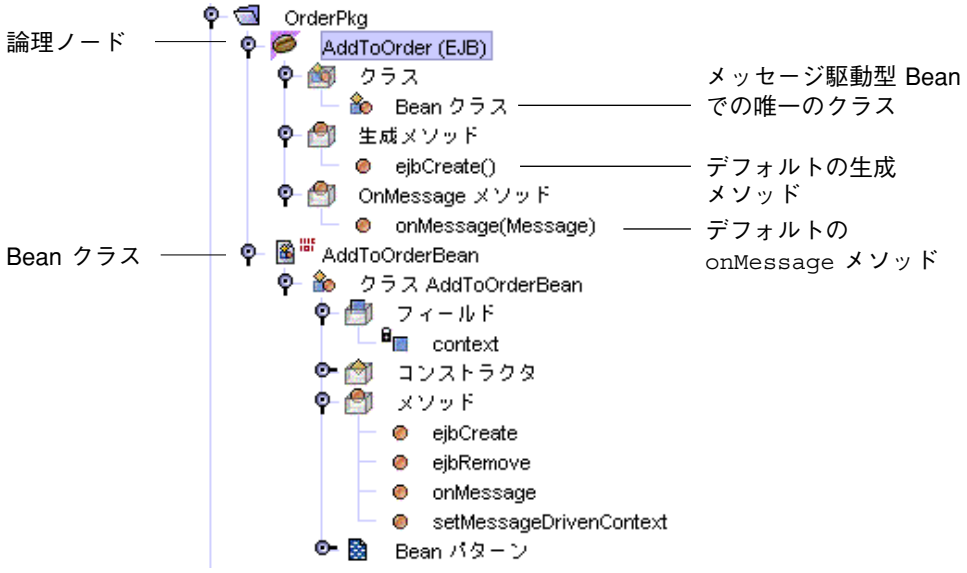


図 7-2　一般的なメッセージ駆動型 Bean のエクスプローラでの詳細表示

生成されたクラスの確認

ウィザードは、各メッセージ駆動型 Bean 内にデフォルトのメソッドを作成します。これらのメソッドは、生成メソッド 1 つ、onMessage メソッド 1 つ、および 2 つのライフサイクルメソッドです。表 7-2 に示すように、生成メソッド ejbCreate は、他の エンタープライズ Bean の生成メソッドとほぼ同様に動作します。一方、onMessage メソッドは、他のメソッドとは異なる種類のメソッドです。

表 7-2　メッセージ駆動型 Bean での ejbCreate メソッドと onMessage メソッドの目的

メソッド	目的
ejbCreate	必要に応じてメッセージ駆動型 Bean を初期化する。
onMessage	メッセージ駆動型 Bean が受信したメッセージを開いて、対応するかどうか決定し、処理を実行する。

また、ウィザードは表 7-3 に示すデフォルトのライフサイクルウィザードも追加します。

表 7-3　メッセージ駆動型 Bean の Bean クラス中にあるデフォルトのライフサイクルメソッドの目的

メソッド	目的
`setMessageDrivenContext`	このメソッドは `ejbCreate` の前に呼び出され、メッセージ駆動型 Bean とコンテキストオブジェクトを対応付ける。
`ejbRemove`	このメソッドは、メッセージ駆動型 Bean インスタンスが削除される直前に呼び出され、そのインスタンスが使用していたリソースを解放する。単純なメッセージ駆動型 Bean の場合だと、このメソッドを使用することもない。

メッセージ駆動型 Bean の完成

メッセージ駆動型 Bean を完成させるには、次の操作を実行します。

- Bean の `onMessage` メソッドの本体を記述して、完成させます。
- Bean の `setMessageDrivenContext` メソッドに必要なコードを記述して、完成させます。

  `ejbCreate` メソッドおよび `ejbRemove` メソッドは、簡単なメッセージ駆動型 Bean では必要ありません。ただし、`ejbCreate` メソッドはリソースを割り当てに、`ejbRemove` メソッドはリソースの解放に使用することができます。
- プロパティシート (Bean が配備されるアプリケーションサーバーのタブ) を使用して、リソースの型、リソースファクトリ、およびメッセージ駆動型 Bean が使用するサーバーを指定します。詳細は、162 ページの「クライアントメッセージ駆動型 Bean のリソースの指定」と第 8 章で記述しています。

以上の操作は、ソースエディタで追加してください。ソースエディタを開くには、エクスプローラで論理ノード配下の Bean コンポーネントを右クリックして「開く」を選択します。

推奨する エンタープライズ Bean の開発手順

付録 A では、エンタープライズ Bean に変更を加える際の最適な方法や、推奨手順以外の方法を取った場合に発生する可能性のあるエラーや異常状態について説明しています。一般に、個々のクラスノードを操作するよりは論理ノードを操作する、メソッドの変更には Bean のプロパティシートや「カスタマイザダイアログ」を使用する、およびダイアログからでは操作できない Bean のコードを完成させたり、編集したりするには IDE のソースエディタを使用するといった方法をお勧めします。

onMessage メソッドの完成

メッセージ駆動型 Bean の単一のインスタンスは、メッセージを 1 度に 1 つだけ処理できます。また、Bean が持てる onMessage メソッドは 1 つだけです。メソッドの記述例を次に示します。

```
public void onMessage(Message inMessage) {
    TextMessage msg = null;

    try {
        if (inMessage instanceof TextMessage) {
            msg = (TextMessage) inMessage;
            System.out.println("MESSAGE Bean: Message " +
            "received: " + msg.getText());
        } else {
            System.out.println("Message of wrong type: " +
            inMessage.getClass().getName());
        }
    } catch (JMSException e) {
        System.err.println("MessageBean.onMessage: " +
        "JMSException: " + e.toString());
        context.setRollbackOnly();
    } catch (Throwable te) {
        System.err.println("MessageBean.onMessage: " +
        "Exception: " + te.toString());
    }
}
```

setMessageDrivenContext メソッドの完成

setMessageDrivenContext メソッドは、メッセージ駆動型コンテキスト参照をフィールドに保存し、非持続フィールドに値を格納できるようにします。必要であれば、このメソッドを使用して、Bean オブジェクトに依存しないリソースを、その Bean が存続する間だけ割り当てることができます。これらのリソースには、キュー接続ファクトリやトピック接続ファクトリなどがあります。

EJB ビルダーのウィザードは、メッセージ駆動型コンテキストを非持続フィールド context に割り当てる コードをデフォルトで作成します。通常は、生成されたメソッドには何も追加する必要はありません。しかし、メソッドを完成させる必要がある場合は、生成されたコンテキストをインスタンス変数にコピーしてください。次に例を示します。

表 7-4　setMessageDrivenContext メソッドの例

```
public void setMessageDrivenContext(javax.ejb.MessageDrivenContext aContext ) {
    this.context=context;
}
```

メッセージ駆動型 Bean を作成した後の作業

作成したメッセージ駆動型 Bean を、最終環境で使用できるようにする必要があります。Bean のプロパティシートで次の項目を指定します。

- Bean のメッセージ駆動型の送信先、つまり Bean がキューからメッセージを受け取るのか、トピックから受け取るのかを指定します。
- Bean がトピックからのメッセージを待機する場合、サブスクリプションが永続か非永続かを指定します。
- Bean が受信するメッセージを限定するためのメッセージセレクタ (フィルタ) を適用するかどうかを指定します。

クライアントからのメッセージを受信するメッセージ駆動型 Bean を J2EE RI に配備する場合は、Bean のプロパティシートにある「J2EE RI」タブで送信先を指定する必要があります。

送信先に向けてメッセージを送信するメッセージ駆動型 Bean をクライアントとして動作させる場合は、Bean のプロパティシートにある「参照」タブで次の項目を指定する必要があります。

- リソース参照 (Bean がメッセージ駆動型の送信先にアクセスするために使用する接続ファクトリ)
- リソース環境参照 (実際の送信先であるキューまたはトピック)

これらのプロパティ設定について次に説明します。

メッセージ駆動型送信先の設定

メッセージ駆動型 Bean に待機させるのがキューなのかトピックなのかを指定するには、次の操作を実行します。

1. IDE のエクスプローラウィンドウで、メッセージ駆動型 Bean の論理ノードを右クリックをし「プロパティ」を選択します。

   Bean のプロパティシートが表示されます。

2. 「プロパティ」タブで「メッセージ駆動型送信先」フィールドをクリックし、省略符号ボタン (...) をクリックします。

   プロパティエディタが表示されます。

3. 「キュー」、「トピック」、または「設定なし」を選択します。

   - クライアントが特定の Bean にだけメッセージを送信し、ポイントツーポイント型通信の利用が必要な場合は、「キュー」を選択します。
   - 複数のクライアントがパブリッシュとサブスクリプト型の通信でこの Bean にメッセージを送信する場合は、「トピック」を選択します。「トピック」を選択した場合、Bean のサブスクリプションが「永続」なのか「非永続」なのかも指定します。
     - Bean がメッセージを取得するまでメッセージを保持させたい場合は、「永続」を選択します。これを選択すると、Bean のアプリケーションサーバーが障害によって停止した場合でも、Bean が復帰したときにメッセージを取得できます。
     - Bean が使用できる間に送信されたメッセージだけを取得させるには、「非永続」を選択します。それ以外の間に送信されたメッセージはすべて削除されます。

- このプロパティを後で設定する場合は、「設定なし」を選択します。「メッセージ駆動型送信先」フィールドが空白のままになります。

4. 「了解」をクリックしてプロパティエディタを終了します。

メッセージセレクタの指定

Bean が受信するメッセージをフィルタで限定したい場合は、次の操作を実行します。

1. 「メッセージセレクタ」フィールドをクリックして、省略符号ボタン (...) をクリックします。
   プロパティエディタが表示されます。
2. フィルタを指定すると、Bean が待機する必要のあるメッセージの数が減ります。
3. 「了解」をクリックしてプロパティエディタを終了します。

クライアントメッセージ駆動型 Bean のリソースの指定

メッセージ駆動型 Bean のプロパティシートにある「参照」タブには、「リソース参照」フィールドと「リソース環境参照」フィールドが含まれています。これらのフィールドは、メッセージを送信するクライアントに対して指定されます。たとえば、あるアプリケーション内で Web モジュールがキューにメッセージを送信するようになっていて、同じアプリケーション内のメッセージ駆動型 Bean がこのキューを待機していたとします。この場合、「リソース参照」と「リソース環境参照」フィールドは、この Web モジュールの提供者によって指定される必要があります。

また、あるモジュール内でメッセージ駆動型 Bean をメッセージをキューかトピックにメッセージを送信するクライアントとして動作させる場合は、これらのフィールドにリソースファクトリとリソースを指定します。

リソースファクトリの指定

送信先オブジェクトを作成するファクトリオブジェクトにメッセージ駆動型 Bean を対応付ける場合は、次の操作を実行します。

1. 「参照」タブで、「リソース参照」フィールドをクリックしてから省略符号ボタン (...) をクリックします。

   プロパティエディタには、接続ファクトリを指定するフィールドが用意されています。この接続ファクトリは、クライアント (またはクライアントとしてのメッセージ駆動型 Bean) が、メッセージのリソースへアクセスするために使用されます。

2. 「追加」ボタンをクリックします。

   次の 2 つのタブを含む「追加 リソース参照」ダイアログが表示されます。

   - 「標準」タブでは次の項目を指定します。
     - キューまたはトピックに対する Bean の接続を作成するオブジェクトの名前を指定します。
     - 「型」コンボボックスで、Bean が使用するリソースファクトリの型を選択します。この型は、プロパティタブの「メッセージ駆動型送信先」フィールドで選択した内容に対応している必要があります。リソースファクトリの型についての説明は、177 ページの「リソース環境参照の設定」を参照してください。
     - 「認証」フィールドで、Bean によるリソースの使用を認可するのが EJB コンテナなのか、アプリケーションクライアントなのかを指定します。
     - 「共有スコープ」フィールドで、同じアプリケーション内の他の Bean にこのリソースへの接続を共有させるかどうかを指定します。同じトランザクションコンテキストで複数の Bean が同じリソースを使用できる場合、コンテナはトランザクションをローカルに実行し、処理時間を短縮します。

   メッセージ駆動型 Bean を RI に配備する場合は、次のフィールドも指定してください。

   - 「J2EE RI 」タブで次の項目を指定します。
     - 「JNDI 名」フィールドで、リソースファクトリがある場所を指定します。
     - ユーザー情報フィールドに、リソースにアクセスするために必要な情報を指定します。
     - メールの構成フィールドに、JavaMail セッションファクトリを使用するのに必要な情報を指定します。この指定は必要がある場合だけです。

3. 操作が終了したら、「了解」をクリックしてダイアログを閉じます。

リソースの指定

メッセージ駆動型 Bean を特定の送信先オブジェクトに関連付けるには、次の操作を実行します。

1. 「参照」タブで、「リソース環境参照」フィールドをクリックしてから省略符号ボタン (...) をクリックします。

   プロパティエディタには、クライアント (またはクライアントとしてのメッセージ駆動型 Bean) がメッセージを送信する対象となる実際のリソースを指定するフィールドが用意されています。

2. 「追加」ボタンをクリックします。

   次の 2 つのタブを含む「追加 リソース環境参照」ダイアログが表示されます。

   - 「標準」タブでは次の項目を指定します。
     - クライアントまたは Bean がメッセージを送信するキューまたはトピックの名前を指定します。
     - コンボボックスでリソースの型を指定します。

   メッセージ駆動型 Bean を RI に配備する場合は、次のフィールドも指定してください。

   - 「J2EE RI 」タブの「JNDI 名」フィールドで、メッセージリソース (キューまたはトピック) がある場所を指定します。

3. 操作が終了したら、「了解」をクリックしてダイアログを閉じます。

RI 上のリスナーメッセージ駆動型 Bean に対するリソースの指定

Bean が RI に配備される場合、この Bean の待機の対象としてキューまたはトピックを指定するだけでは不十分です。これに加えて、リソースの JNDI 名も指定し、RI が検出できるようにする必要があります。次の操作を実行します。

1. 「J2EE RI」タブで、「送信先 JNDI 名」フィールドをクリックして省略符号ボタン (...) をクリックします。

2. プロパティエディタで、Bean がメッセージを待機する対象となる JNDI 名を指定します。

   <型>/<リソース> の形式で指定します。たとえば、メッセージキューは jms/myQueue のように指定します。

   メッセージ駆動型送信先についての詳細は第 2 章を、プロパティの設定については第 8 章を参照してください。

   推奨する Bean の操作方法については付録 A を参照してください。

メッセージ駆動型 Bean 開発時の注意

アプリケーションのメッセージ層での問題を避けるために、次の点について理解しておいてください。

- **メッセージの順序** - メッセージ駆動型 Bean は、メッセージがどんな順序で受信されても処理できるように作成してください。JMS サーバーはメッセージ駆動型 Bean のプールに対し、任意の順序でメッセージを配信します。
- **ejbRemove が起動されない場合の対応** - 単純なメッセージ駆動型 Bean では ejbCreate メソッドや ejbRemove メソッドを使用する必要がありません。しかし、複雑な Bean でこれらのメソッドを使用している場合、(障害によるシステムやコンテナの停止といった) 特定の状況では ejbRemove メソッドが呼び出されないことがあります。この場合、Bean が自身をクリーンアップできるような手段を準備しておいてください。具体的な手段はアプリケーションサーバーの動作によっても変わるので、詳細はサーバーのマニュアルを参照してください。
- **メッセージの無限送信** - トランザクションを自身で管理するようなメッセージ駆動型 Bean のインフラストラクチャを、EJB ビルダーのウィザードを使用して作成している場合、Bean プロパティの「確認モード」を「自動」に設定できます。これを設定すると、 Bean がメッセージを受信するたびに自動的に通知させることができます。このことによって、トランザクションが失敗したときに、メッセージの送信先がメッセージを受信したことが確認できずに、同じメッセージを繰り返し送信してしまうのを避けることができます。

設計上の詳細な留意点については、「Enterprise JavaBeans 仕様 2.0」と エンタープライズ Bean の開発に関する文献を参照してください。

詳細情報の参照先

エンタープライズ Bean は、アプリケーションに高い機能性と柔軟性を付加します。また、エンタープライズ Bean の基本要素は簡単に作成できます。Sun ONE Studio IDE のようなツールを使用すれば更に簡易化できます。しかし、アプリケーションのニーズをすべて満たす Bean の作成は複雑なこともあります。詳細は、次の URL からアクセスできる Enterprise JavaBeans 仕様 2.0 を参照してください。

```
http://java.sun.com/products/ejb/docs.html
```

第8章

エンタープライズ Bean の配備

これまでの章では、個々の エンタープライズ Bean の作成について説明しました。作成した Bean をアプリケーションに組み込み、使用環境に配備するには、次の 3 つの準備処理が必要です。

1. Bean の外部依存性と動作要件に関する情報を指定します。この情報は、次の節で説明する Bean の配備記述子となります。
2. アプリケーション内で協調的に動作する Bean の集まりを EJB モジュールに格納します。また、必要に応じて EJB モジュールを EJB JAR ファイルに追加します。
3. Sun ONE Studio IDE の機能を使用して、Bean とモジュールをテストします。

通常は、エンタープライズ Bean は EJB モジュールに組み込まれ、EJB モジュール (1 つ以上) はアプリケーションに組み込まれます。更に、アプリケーションはアプリケーションサーバーに配備されます。また、個々の EJB モジュールを直接配備することもできます。

アプリケーションの設計とアセンブル、およびサーバーへの配備に関する詳細は、マニュアル『J2EE アプリケーションのプログラミング』で詳しく説明しています。

配備情報とは

エンタープライズ Bean のプロパティシートで指定した配備情報は、Bean の配備記述子の一部となります。これは、Bean の構造、他の Bean との関係、およびその外部依存性に関する情報を収集する、XML 形式のテキストファイルです。配備記述子には、

Bean のアプリケーションが配備されるときにアプリケーションサーバーが必要とする、すべての命令が含まれます。 この記述子に対する変更は、アプリケーション内での Bean の動作に反映されます。

これまでの章で説明したように、EJB ビルダーのウィザードを使用して エンタープライズ Bean を作成すると、Bean の配備記述子が自動的に生成されます。Bean の記述子の基本的な部分は、ウィザードによって自動的に生成されます。

エンタープライズ Bean を EJB モジュールに格納すると (184 ページの「EJB モジュールの作成」で説明しています)、IDE によってそのモジュールの配備記述子も同時に生成されます。この記述子は、次の情報を収集します。

- Bean の配備記述子から取り込まれる各 Bean の宣言的なメタ情報 (コードには含まれていないその Bean に関する外部的な情報)。
- アプリケーションへの Bean の組み込み方や、モジュールの配備に関する高レベルな情報。
- セキュリティとトランザクションに関する情報で、Bean レベルで指定された情報は EJB モジュールが上書きできる。
- エンティティ Bean が表現するデータソースについてコンテナに与える命令。

EJB モジュールの記述子を変更することで、そのコンポーネントとなっている Bean のソースコードを書き替えることなく、アプリケーションの動作を変更できます。

配備記述子の内容は、対応する Bean またはモジュールのプロパティシートからアクセスできます。必要であれば、EJB モジュールの記述子を直接編集することもできます。以降の 3 つの節では、配備記述子の表示と編集について説明します。

生成された配備記述子の参照

配備記述子の XML 形式のファイルを参照するには、次の操作を実行します。

- エクスプローラから Bean の論理ノードを選択し、右クリックして「配備記述子を表示」を選択します (または EJB モジュールの論理ノードを選択し、右クリックして「配備記述子」->「表示」を選択します)。

  ソースエディタに、ファイルの内容が読み取り専用として表示されます。

EJB モジュール配備記述子の編集

通常は、Bean または EJB モジュールの配備記述子は、Bean または EJB モジュールそれぞれのプロパティーシートを使って編集してください (操作は次の節に記述しています)。「プロパティ」ダイアログから配備記述子ファイルを編集する際は、ファイル名と値 (エンティティ Bean の場合は、表と列の名前) を指定するだけで済みます。XML のコードを記述する必要はありません。ダイアログで項目を選択すると、IDE によって エンタープライズ Bean の適切なクラスすべてに、変更内容が自動的に反映されます。

ただし、必要であれば、配備記述子を直接編集することもできます。編集後に、生成された記述子を元に戻すこともできます。

EJB モジュール配備記述子の直接編集

EJB モジュールの配備記述子ファイルに対して直接編集が必要な場合は、次の操作を実行します。

- **EJB モジュールを右クリックし、「配備記述子」->「最終的な編集」を選択します。**

  EJB モジュールの配備記述子を直接編集し終わった後でも、プロパティシートを使用して別の変更を加えることができます。たとえば、プロパティシートの「J2EE 」タブから「データソースの JNDI 名」、「データソースのパスワード」、または「データソースのユーザー名」フィールドを指定したり、変更したりできます (これらのフィールドをプロパティシートと直接編集の両方で変更しないようにしてください)。配備記述子での項目を表現するフィールドは、プロパティシートの編集操作では表示されません。

EJB モジュール配備記述子の状態復帰

「最終的な編集」機能で配備記述子を編集したが、最後に生成された配備記述子の状態に戻して、プロパティシートで変更を加えたい場合、次の操作を実行します。

- **EJB モジュールのノードを右クリックし、「配備記述子」->「生成物に戻す」を選択します。**

注 - 「生成物に戻す」を選択すると、配備記述子に直接加えた編集はすべて失われます。

プロパティシートによる配備記述子の編集

プロパティシートを使用して、エンタープライズ Bean または EJB モジュールの配備記述子を追加、編集、または完成させるには、次の操作を実行します。

- **Bean または EJB モジュールの論理ノードを選択し、右クリックして「プロパティ」を選択します。**

  エンタープライズ Bean の場合は、次の 3 つのタブを含む「プロパティ」ダイアログが表示されます。

  - プロパティ
  - 参照
  - J2EE RI (Java™ 2 Platform, Enterprise Edition のリファレンス実装サーバー)

  これらのタブに加えて、IDE にインストールされているほかのアプリケーションサーバーのプラグインのタブも表示されます。

  EJB モジュールの場合は、「プロパティ」(参照用プロパティを持つ)と「J2EE RI」の 2 つのデフォルトタブが表示されます。

  エンタープライズ Bean の 3 つのデフォルトタブについて次に説明します。

Bean のプロパティの指定

エンタープライズ Bean を EJB モジュールに格納する以前に、個々の Bean のプロパティを指定します。これらのプロパティは以降の節でタブごとに説明しています。170 ページの「「プロパティ」タブ」、173 ページの「「参照」タブの使用」、および 181 ページの「J2EE RI タブの使用」を参照してください。

「プロパティ」タブ

これまでの章で説明した Bean のコードについての知識があれば、「プロパティ」タブのほとんどのフィールドの意味を理解できるはずです。ここでは、特に注意する必要のあるフィールドだけを取り上げます。

- エンタープライズ Bean の「プロパティ」タブには、読み取り専用のフィールドが含まれています。これらのフィールドは、Bean の作成時に EJB ビルダーによって自動的に指定されます。これらのフィールドが指定するプロパティは各 Bean の型の本質的な部分なので変更する必要はありません。これらを変更する必要がある場合は、ウィザードに戻って Bean を再作成してください。

- 「名前」フィールドと Bean のインタフェース用のフィールドは、Bean の作成時に EJB ビルダーウィザードによって指定されます (または Bean の作成時にプログラマがこれらのクラス名を変更することもできます)。

  現在のクラスの代わりに、既存の別のクラスを使用したい場合は、「プロパティ」タブの該当するフィールドに表示されたクラス名を上書きすることができます。

  クラスの内容はそのままに、クラス名だけを変更したい場合は、そのクラスをプロパティエディタで開いて変更します (論理ノードを開かないでください)。またはソースエディタでクラスのコードを直接編集します。

- 「ラージアイコン」フィールドと「スモールアイコン」フィールドでは、エンタープライズ Bean のアイコンを指定します。ここで指定したアイコンは、アプリケーションサーバーなどのツールから使用できます。ラージアイコンは 32 × 32 ピクセルの、スモールアイコンは 16 × 16 ピクセルの JPEG 画像か GIF 画像でなければなりません。どちらのフィールドについても、拡張子が `.jpg` か `.gif` のファイルを指定する必要があります。

- 「セキュリティ ID」フィールドには、エンタープライズ Bean が呼び出しの際に使用する識別情報を指定します。
  - 「run-as セキュリティロール」を選択し、関連する Bean に指定されているロールを指定した場合、Bean はそのセキュリティロールの下で実行され、その Bean から呼び出されるすべてのメソッドはそのセキュリティロールを持つことになります。このようにして、この Bean は、他の Bean のために予約されているデータにごく普通にアクセスできます。また、このフィールドでデフォルトのセキュリティロールを Bean に指定することもできます。
  - 「呼び出し元のセキュリティ ID を仕様」を選択すると、実行中の Bean は呼び出し元のセキュリティ ID を利用します。
  - このフィールドを「設定なし」のままにしておくと、Bean はアプリケーションサーバーのレベルで指定されたセキュリティ ID を使用するか、呼び出し元のセキュリティ ID を使用して実行されます。

このタブで指定するプロパティのいくつかは、Bean の種類に固有なものです。これらのプロパティについて次の節で説明します。

エンティティ Bean のプロパティ

エンティティ Bean の「プロパティ」タブには、次のフィールドが表示されます。

- **主キークラス** - EJB ビルダーのウィザードによる指定を変更したい場合は、「既存のフィールドを選択」、「未知の主キークラス」、「既存のユーザー定義クラスを選択」のいずれかを選択します。
- **再入可能** - 意図しないマルチスレッドの問題を避けたい場合は、このフィールドを False のままにしておきます。こうしておくと、あるトランザクションコンテキスト内で Bean インスタンスがクライアント要求を実行しているとき、同じコンテキスト内の同じエンティティオブジェクトに対して次の要求が来た場合に、コンテナは 2 度目の要求に対して例外を発生させます。ただし、Bean を他の Bean から必ず呼び出させたい場合は、このフィールドを True に設定します。

CMP エンティティ Bean の「プロパティ」タブでは、次のフィールドも表示されます。

- **抽象スキーマ名** - 抽象スキーマは、Bean の持続フィールドと関係を定義します。CMP エンティティ Bean の EJB QL 照会は、Bean の抽象持続スキーマとその依存オブジェクトクラスに基づいています。
- **CMP バージョン** - このフィールドは読み取り専用で、該当する CMP エンティティ Bean が バージョン 1.x Bean クラスを持っているか、バージョン 2.x Bean クラスを持っているかを表しています。

セッション Bean のプロパティ

セッション Bean の「プロパティ」タブでは、次のフィールドが表示されます。

- **Bean 型** - ステートレス (Stateless) かステートフル (Stateful) のどちらかに変更できます。
- **トランザクション型** - トランザクションをコンテナに管理させるか、Bean に管理させるかを指定します。このフィールドを変更すると、EJB ビルダーによって Bean のクラスコードも変更されます。

メッセージ駆動型 Bean のプロパティ

メッセージ駆動型 Bean の「プロパティ」タブには、次のフィールドが表示されます。

- **トランザクション型** - トランザクションをコンテナに管理させるか、Bean に管理させるかを指定します。このフィールドは EJB ウィザードが自動的に指定しますが、必要に応じて変更することもできます。

- **通知モード** - メッセージ駆動型 Bean のトランザクション型に `Bean` が指定されている場合、「確認モード」プロパティがタブに表示されます。Bean がメッセージを取得するたびに通知させたい場合は、このプロパティを「自動」に設定します。重複したメッセージの取得を Bean から通知させたい場合は、「重複可能」を設定します。

- **メッセージセレクタ** - このフィールドで 1 つ以上のフィルタを指定すると、メッセージ駆動型 Bean が待機する必要のあるメッセージの数を減らすことができます。たとえば、次のように指定すると、`AccountStatus` プロパティに `Late` または `Delinquent` という値が設定されているメッセージだけが、Bean に待機されるようになります。

  ```
  AccountStatus = 'Late' OR AccountStatus = 'Delinquent'
  ```

  メッセージセレクタの構文は SQL92 条件式構文のサブセットに基づいていて、詳細は JMS 仕様および JMS 入門に説明されています。

- **メッセージ駆動型送信先** - メッセージ駆動型 Bean がメッセージキューをリッスンするのか、特定のトピックを受信するのか (そしてこの場合は、永続なのか非永続なのか) を指定します。このフィールドのデータは、プロパティシートにあるアプリケーションサーバー用のタブのフィールドの内容に関連しています。このフィールドに指定した内容は、「J2EE RI」タブの「接続ファクトリ名」、「送信先 JNDI 名」および「持続性サブスクリプション名」フィールドに対応している必要があります。詳細については、183 ページの「メッセージ駆動型 Bean に対する J2EE RI プロパティの設定」を参照してください。

「参照」タブの使用

エンタープライズ Bean をアセンブル・配備する前に、このタブのフィールドを設定しておく必要があります。エンタープライズ Bean のほとんどの外部依存性は、これらのフィールドで指定できます。一部の情報は Bean レベルで指定しておいて、後からモジュールまたはアプリケーションレベルの指定で上書きすることもできます。

「参照」タブの表示例を次に示します。

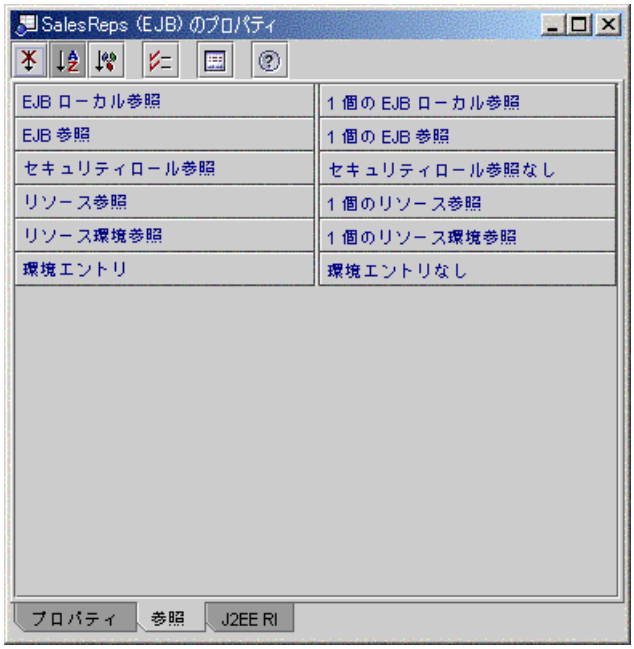


図 8-1　CMP エンティティ Bean の「プロパティ」ダイアログでの「参照」タブ

このタブの各フィールドと指定内容について、以降に説明します。

EJB ローカル参照の指定

「EJB ローカル参照」フィールドと「EJB 参照」フィールドには、Enterpise Bean が呼び出す対象となるメソッドを含んだ Bean の情報を指定します。「EJB ローカル参照」フィールドには、同じ JVM 内に常駐する他の Bean への参照を指定します。この指定があった場合、Bean の EJB ローカル参照は同じ JVM 内で実行されている他の Bean のローカルインタフェースにアクセスします。これらの Bean が異なる EJB モジュールで動作していてもアクセスは実行されます (176 ページの「EJB 参照の指定」を参照して、EJB ローカル参照との違いも確認してください)。

エンタープライズ Bean の開発時には、Bean の名前も指定できます。また、EJB モジュールに Bean をアセンブルする際は名前を変更することができます。

Bean のコードでは、JNDI インタフェースを使用して、ほかの Bean のホームインタフェースを検索します。Bean を EJB モジュールに組み込む前に、それと同じ参照を Bean のプロパティシートで指定し、Bean 同士をリンクさせます。アセンブル担当者

は「EJB 参照」フィールドを参照して、Bean が機能するためには、ほかにどの Bean が必要かを確認します。アセンブル担当者と配備担当者は、これらの参照をそのまま使用することも、実行環境に合わせて、モジュールレベルで上書きすることもできます。

複数の Bean を EJB モジュールに組み込む前に、まず Bean クラスで参照をコーディングし、プロパティシートで EJB 参照を指定して、これらの Bean 同士をリンクさせる必要があります。

プロパティシートで EJB 参照を指定するには、次の操作を実行します。

1. **「EJB ローカル参照」フィールドをクリックし、省略符号ボタン (...) をクリックします。**

   「EJB ローカル参照」プロパティエディタが表示されます。

2. **「追加」をクリックします。**

   「追加 EJB ローカル参照」ダイアログが表示されます。

3. **各フィールドを設定します。**

   次のフィールドは省略できません。

注 - これらのフィールドを指定する簡単な方法は、「参照される EJB 名」の「ブラウズ」ボタンをクリックしてローカルの エンタープライズ Bean を選択することからはじめることです。そうすれば、「型」フィールドと 2 つの「インタフェース」フィールドは IDE によって自動的に指定されます。「インタフェース」フィールドは、必要に応じて変更することもできます。

- **参照名** - 参照先の Bean 名です。Bean クラスのコードに含まれている `context.lookup` メソッドの呼び出し文で指定されている名前を指定します。このフィールドには、あらかじめ「`ejb/`」が入力されています。この名前は、`java:comp/env` コンテキストの `ejb/` サブコンテキストを基準にした相対名です。スラッシュ (/) に続けて、参照先の Bean 名を入力します。

  例として、`Account` という Bean から、`DiscountCodeTbl` という Bean のホームインタフェースの参照を検索する場合を考えてみましょう。この場合の完全参照名は `ejb/DiscountCodeTblHome` になります。代わりに、JNDI 検索コードで使用した参照名に相当する別の参照名を指定することもできます。

- **参照される EJB 名** - 指定されたローカルホームインタフェースおよびローカルインタフェースを実装しているエンタープライズ Bean 名を指定します。状況に応じて次のどれかの操作で指定できます。
  - 「ブラウズ」をクリックして Bean を選択します。選択後、ローカルホームインタフェースとローカルインタフェースのフィールドが、IDE によって自動的に指定されます。
  - 「参照される EJB 名」フィールドに、現在の Bean と同じインタフェースを持つ他の Bean の名前を直接入力し、その Bean に参照を変更します。
  - EJB モジュールまたはアプリケーションに Bean が組み込まれるまで、「参照される EJB 名」フィールドを空白のままにしておきます。
- **型** - 参照先の Bean の型です。セッション Bean とエンティティ Bean のどちらかを指定します。
- **ローカルホームインターフェース** - 参照先の Bean のローカルホームインタフェースです。
- **ローカルインターフェース** - 参照先の Bean のローカルインタフェースです。

次の「説明」フィールドは省略できます。EJB モジュールをアプリケーションに組み込むときにわかりやすいように説明を記述しておくことができます。

- **説明** - 参照される Bean の目的または参照する側の Bean の目的などを記述します。

EJB 参照の指定

「EJB 参照」フィールドでは、呼び出し対象のメソッドを含んだ、外部 JVM で実行されている エンタープライズ Bean へのリンクを指定します。このフィールドは「EJB ローカル参照」フィールドと同様に使用しますが、他の JVM で実行されている Bean のリモートインタフェースを参照させるという違いがあります。

環境エントリの指定

環境エントリは、Bean の実行環境で保存される名前付きのデータ値です。この値は、配備先のポリシーや手順によって変わります。環境エントリを使用すると、Bean のソースコードを変更することなく、配備時に Bean の動作を変更できます。プロパティシートのこのフィールドで設定した値は、配備時に EJB モジュールやアプリケーションの配備記述子で上書きすることができます。

たとえば、Account という Bean で、(boolean 型の) overdraftAllowed という環境エントリを使用した場合を考えてみましょう。この変数は、この Bean を使用する銀行で、預金口座の利用客が残高以上の金額を引き出せるかどうか (超過引き出しを認めるかどうか) を示しています。Account Bean は、overdraftAllowed の値を参照して、利用客が超過引き出しを要求したときに、どのように対処するかを決定します。

環境エントリを追加するには、環境ごとに次の手順に従います。

1. 「環境エントリ」フィールドをクリックし、省略符号ボタン (...) をクリックします。
   「環境エントリ」プロパティエディタが表示されます。
2. 「追加」をクリックします。
   「環境エントリの追加」ダイアログが表示されます。
3. 各フィールドを設定します。
   次の 2 つのフィールドは省略できません。
   - **名前** - 環境変数の名前です。
   - **型** - 環境変数のデータ型です。

   さらに、次の 2 つのフィールドを指定することができます。
   - **説明** - 環境変数の使用目的など、アセンブル担当者や配備担当者が、該当する環境で Bean を使用するときに知っておく必要のある情報を指定します。
   - **値** - 初期値です。

リソース環境参照の設定

このフィールドでは、JMS 送信先 (キューまたはトピック) など、Bean が使用する必要のある管理オブジェクトを指定します。リソース環境参照は、キューまたはトピックの論理名です。この論理名は、Bean クラスの InitialContext.lookup に記述する名前と必ず一致させてください。このフィールドで指定したリソースを Bean が参照する際には、リソースのインスタンスが次の節で説明するファクトリによって作成されます。

ソース環境参照を追加するには、Bean が使用するオブジェクトそれぞれに対して、次の操作を実行します。

1. 「リソース環境参照」フィールドをクリックし、省略符号ボタン (...) をクリックします。

   「リソース環境参照」プロパティエディタが表示されます。

2. 「追加」をクリックします。

   「追加 リソース環境参照」ダイアログが表示されます。

3. 各フィールドを設定します。

   次のフィールドは省略できません。

   - **名前** - Bean クラスの `InitialContext.lookup` メソッドに記述されている名前を指定します。
   - **型** - リソースファクトリの型を指定します。使用する型を指定し、次のどちらかを選択します。
     - `javax.jms.Queue`。Java メッセージサービスキューです。
     - `javax.jms.Topic`。Java メッセージサービストピックです。

リソース参照の指定

このフィールドには、Bean が必要なリソースへ接続するファクトリ名を指定します。このリソースとしては、(リレーショナルデータベースなどの) データソース、(キューまたはトピックといった) 管理オブジェクト、JavaMail セッション、URL、または (Bean を他のアプリケーションシステムや EIS に接続する) J2EE コネクタが該当します。「リソース参照」フィールドに指定する情報は、Bean クラスに記述されている JNDI lookup メソッド呼び出しの内容と必ず一致させてください。

リソース参照を追加するには、Bean が必要とするリソースそれぞれに対して次の操作を実行します。

1. 「リソース参照」をクリックし、省略符号ボタン (...) をクリックします。

   「リソース参照」プロパティエディタが表示されます。

2. 「追加」をクリックします。

   「追加 リソース参照」ダイアログが表示されます。

3. 各フィールドを設定します。

   次のフィールドは省略できません。

- **名前** - Bean クラスの `InitialContext.lookup` メソッドに記述されている名前を指定します。たとえば、`myDatabase` と指定されている JDBC リソースファクトリがあった場合、lookup メソッドは次のように記述されている必要があります。

```
javax.naming.InitialContext myContext =
    new javax.naming.InitialContext();
javax.sql.DataSource mySource = (javax.sql.DataSource)
    myContext.lookup("java:comp/env/jdbc/myDataBase");
```

この場合の、対応するリソース参照は `jdbc/myDataBase` です。

- **型** - リソースファクトリの型を指定します。使用する型を指定して、次のどれかを選択します。
  - `javax.sql.DataSource`。JDBC 接続ファクトリです。
  - `javax.jms.QueueConnectionFactory`。Java メッセージサービス接続ファクトリです。
  - `javax.jms.TopicConnectionFactory`。Java メッセージサービス接続ファクトリです。
  - `javax.mail.Session`。JavaMail セッションファクトリです。
  - `javax.resource.cci.ConnectionFactory`。他のアプリケーションシステムや EIS に接続する Bean を宣言します。これを指定すると、エンタープライズ Bean はコネクタアーキテクチャである CCI (Common Client Interface) API とリソースアダプタを使用して外部データにアクセスできます。アプリケーションに CCI API を実装する詳細については、マニュアル「Java 2 Platform, Enterprise Edition Tutorial」を参照してください。
  - `java.net.URL`。URL 接続ファクトリです。
- **認証** - ユーザーを認証し、このリソースを使用する権限を与える方法です。
  - Container - EJB コンテナがリソースマネージャにサインオンします。サインオンするための情報は、配備担当者が配備時に指定します。
  - Application - エンタープライズ Bean にプログラミングしたコードによって、マネージャへのサインオンを実行します。

次のフィールドの指定は任意です。

- **共有スコープ** - このリソースへの接続を、同じアプリケーション中の他の エンタープライズ Bean と共有させるかどうかを指定します。複数の Bean が同じトランザクションコンテキストで同じリソースを使用する場合、コンテナはトランザクションをローカルに実行し、処理時間を節約します。

セキュリティロール参照の指定

エンタープライズ Bean が独自のセキュリティチェックを行う (すなわち、Bean を使用して作業を行う権限がユーザーに与えられているかどうかを独自の方法でチェックする) 場合は、このフィールドでセキュリティロールの参照を指定する必要があります (メッセージ駆動型 Bean にセキュリティロールは必要ありません。この Bean は、クライアントからのメッセージに含まれたセキュリティ情報を伝播します。セキュリティ検査が必要な場合は、処理を要求するメッセージ駆動型 Bean が含まれているコンテナか エンタープライズ Bean が後に実行します)。

このフィールドを使用するためには、それに対応するコード (プログラムによるセキュリティ) が Bean クラスに含まれている必要があります。たとえば、Bean クラスのコードに `javax.ejb.EJBContext` インタフェースの次のメソッドが含まれているとします。

`isCallerInRole(`*rolename*`)`

この場合、プロパティシートのセキュリティロールの参照に、該当するすべてのロール名を追加します。

セキュリティロールはモジュールのレベルでも定義できます。詳細については、マニュアル『J2EE アプリケーションのプログラミング』を参照してください。

Bean レベルでセキュリティロールを指定する場合は、次の操作を実行します。

1. **「セキュリティロール参照」フィールドをクリックし、省略符号ボタン (...) をクリックします。**

   「セキュリティロール参照」プロパティエディタが表示されます。

2. **「追加」をクリックします。**

   「追加 セキュリティロール参照」ダイアログが表示されます。

3. **各フィールドを設定します。**

   - **名前** - セキュリティロール名です。Bean クラスのコードで指定されているロール名を指定します。このフィールドは省略できません。

- **説明** - ロールの説明です。このフィールドの指定は省略できます。
- **セキュリティロールリンク** - 配備先の環境でのセキュリティロールに対するリンクです (このフィールドは Bean レベルでは省略できます。この情報は開発段階では決定されていない場合があります。このフィールドは、配備担当者が配備時に指定するのが一般的です)。

J2EE RI タブの使用

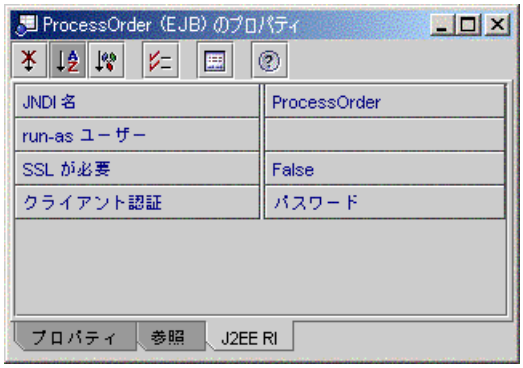
「J2EE RI (J2EE リファレンス実装サーバー)」のタブには、EJB ビルダーを使用してエンタープライズ Bean を作成したときに自動的に割り当てられるプロパティが表示されます。これらのプロパティには、Bean を配備し、テストするのに適したデフォルト値が設定されていますが、エンティティ Bean をアプリケーションの構成要素として配備する前に、これらの値を変更・指定する必要があります。

メッセージ駆動型 Bean、セッション Bean、または持続性を自分自身で管理するエンティティ Bean (BMP Bean) の場合は、J2EE RI 関連のプロパティを参照しても、ほとんど情報は表示されません。これに対して、EJB コンテナに 1 つ以上の CMP Bean を含んだ EJB モジュールの場合は、J2EE RI 関連のプロパティフィールドに多くの情報が必要です。これらの情報が、Bean とデータストレージとの間で情報を受け渡すために、コンテナが使用する命令になります。

最初に、個々の エンタープライズ Bean の J2EE RI タブに指定するプロパティフィールドの値について説明します。その後、エンタープライズ Bean を格納する EJB モジュールの作成手順を 184 ページの「EJB モジュールの作成」で説明します。最後に、CMP Bean のデータベース関連のプロパティの EJB モジュールレベルでの設定について説明します。

セッション Bean とエンティティ Bean に対する J2EE RI プロパティの設定

セッション Bean、CMP Bean、または BMP Bean の「J2EE RI」タブには、「クライアント認証」、「JNDI 名」、「run-as ユーザー」、および「SSL が必要」プロパティが表示されます。例として、ProcessOrder という名前のセッション Bean の「J2EE RI」タブを次に示します。



このタブでは次のフィールドを指定します。

- **クライアント認証** - クライアントをパスワード (ユーザにユーザ ID とパスワードの指定が要求されます) で認証するのか、証明書 (より厳重なセキュリティのために公開鍵証明書が要求されます) で認証するのかを指定します。または、クライアントにパスワードまたは証明書による認証を選ばせる「クライアントの選択をサポート」を指定します。
- **JNDI 名** - RI プラグインによってこのフィールドのデフォルト値が設定されます。この値は、JNDI lookup メソッドが配備記述子の作成前に Bean の保存場所を見つけるための名前です。名前は変更できますが、空白にしておくことはできません。
- **run-as ユーザー** - Bean に、呼び出し元のセキュリティ ID ではなく特定のセキュリティ ID を使用させたい場合に、このフィールドにユーザー ID を指定します。「セキュリティ ID 」ダイアログで「run-as セキュリティロール」を選択した場合は、表示されたロールの中からユーザー ID を選択できます。
- **SSL が必要** - デフォルトは False です。Bean に SSL (Secure Socket Layer) プロトコルが必要な場合は、True に設定します。

CMP エンティティ Bean を RI に配備する場合、最初に EJB モジュールを作成します。次に、データストレージと Bean の関係を EJB モジュールレベルで操作してください。詳細については、187 ページの「CMP エンティティ Bean へのデータベース関連プロパティの設定」を参照してください。

メッセージ駆動型 Bean に対する J2EE RI プロパティの設定

メッセージ駆動型 Bean の J2EE RI タブには、前述のフィールドに加え、Bean が取得するメッセージの種類に関するフィールドも含まれています。

各フィールドには、次のように値を設定します。

- **クライアント認証** - 他の種類の Bean と同様、クライアントをパスワードで認証するのか、証明書で認証するのかを指定します。または、クライアントがパスワードか証明書のどちらかの認証を選べる「クライアントの選択をサポート」を指定します。
- **接続ファクトリ名** - メッセージ駆動型 Bean のキュー接続ファクトリまたはトピック接続ファクトリがある場所を指定します。
- **送信先 JNDI 名** - メッセージ駆動型 Bean がリッスンするキューまたはトピックのある場所を指定します。
- **持続性サブスクリプション名** - メッセージ駆動型 Bean が永続のトピックのサブスクリプションを実行する場合のトピック名を指定します。

- run-as ユーザー - メッセージ駆動型 Bean に特定のセキュリティ ID を使用させたい場合は、その ID を指定します。指定を省略すると Bean は呼び出し元の ID を使用します。
- SSL が必要 - デフォルトは False です。Bean に SSL (Secure Socket Layer) プロトコルが必要な場合は、True に設定します。

CMP Bean のデータベース関連のプロパティを設定するには、最初に CMP Bean を格納する EJB モジュールを作成する必要があります。EJB モジュールについて次に説明します。

EJB モジュールの作成

Bean は単独で動作するようにも、他の Bean と協調して動作するようにも設計できます。どちらの場合でも、正しい組み合わせの エンタープライズ Bean が、それらの Bean の処理要件に欠かせない情報と共にまとめられて、EJB モジュールに格納されていることが大切です。

協調的に動作する複数の エンタープライズ Bean は、異なる EJB モジュールに含まれていても問題はありません。ただし、これらのモジュールは同じ JVM 内に常駐している必要があります。場合にもよりますが、協調的に動作する Bean は同じモジュール内に格納しておくと便利です。

EJB モジュールは IDE での論理エンティティで、物理的な EJB JAR ファイル (`.jar` の拡張子を持つ Java アーカイブファイル) を表現しています。EJB モジュールは、 EJB JAR ファイルに含まれる Bean の一覧を、Bean 間の関係と配備環境に設定するプロパティと共に追跡します。EJB モジュールはアプリケーションサーバーに配備できる エンタープライズ Bean の最少単位です。

IDE の「エクスプローラ」ウィンドウで表示されるように、EJB モジュールノードはモジュールの配備記述子も表しています。配備記述子は、モジュールを構成する エンタープライズ Bean とその Bean のソースコードが保存されている場所を示すものです。EJB モジュールに含まれる Bean は、1 つのディレクトリにまとまっていても、複数のディレクトリにわたっていても、異なるファイルシステムに分かれて保存されていても構いません。Bean そのもの、またはそのコピーが EJB モジュールと同じ場所に保存されている必要はありません。

EJB モジュールに格納する内容

1 つの EJB モジュールに含める エンタープライズ Bean の数に関する一般的なガイドラインをここでは説明します (詳細については、Java 2 プラットフォーム Enterprise Edition のマニュアルを参照してください)。Bean を EJB モジュールに格納する際には、次の点に考慮します。

- **再利用性の最大化** - ある エンタープライズ Bean の再利用性が高い場合は、その エンタープライズ Bean だけを格納した EJB モジュールを作成します。そうしておくと、アプリケーションがアセンブルされたときに、このモジュールを他のモジュールと自由に組み合わせることで、アプリケーションの規模を抑えたままでそのアプリケーションに必要な機能だけを提供できます。

  モジュールにまとめる Bean は、協調的に利用される可能性が最も高いものだけとします。たとえば、EJB 2.0 環境で構築された CMP Bean すべてと、それらと関連のある Bean は同じモジュールに格納する必要があります。

- **アセンブルの簡易性の最大化** - アプリケーションに必要な エンタープライズ Bean がすべて 1 つのモジュールに格納されていれば、アプリケーションのアセンブル担当者の作業は少なくて済みます。少なくとも、そのアプリケーションで使用する Bean がすべて何らかの形でまとまっていると便利です。モジュールの数を少なく保つことは、再利用性が重要でない場合には効果的な方法です。

- **再利用性とアセンブルの簡易性の釣り合い** - 中規模の J2EE アプリケーションでは、関連の強い、または対になっている エンタープライズ Bean であれば 1 つのモジュールにまとめ、単独で再利用できる Bean はそれぞれ独立したモジュールに格納しておけます。モジュールにまとめるとよい Bean としては、機能的に関連がある、互いに依存している、循環参照がある、または共通のセキュリティ情報を使用しているものなどがあります。

EJB モジュールへの エンタープライズ Bean の格納

エンタープライズ Bean を 1 つだけ EJB モジュールへ格納するには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで Bean の論理ノードを右クリックし、「新規 EJB モジュールを作成」を選択します。
2. 「新規 EJB モジュール」ダイアログで、必要に応じてモジュールの名前を変更します 。

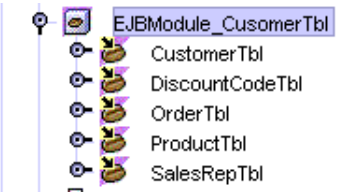
3. ファイルシステムのツリー表示から、モジュールの保存場所を選択します。

4. 「了解」をクリックします。

協調する複数の エンタープライズ Bean を格納する EJB モジュールを作成する場合は、上記の操作に従います。ただし、Bean を選択する際に Cntl キーまたは Shift キーを使用して、モジュールに格納したい複数の Bean すべてを選択します。

複数の Bean を格納する EJB モジュールを作成する別の方法として、Bean の Java パッケージを右クリックし、「新規」->「J2EE」->「EJB モジュール」を選択する方法があります。この操作によって、IDE が新しい空の EJB モジュールをパッケージ内に作成します。作成されたモジュールを右クリックして「EJB を追加」を選択します。「EJB を EJB モジュールに追加」ダイアログが表示されるので、格納したいすべての Bean を Cntl キーまたは Shift キーを使用しながら選択します。

エクスプローラでノードを展開して、モジュールの内容を確認します。



EJB モジュールに協調動作する CMP Bean を複数持たせたい場合は、EJB ビルダーのウィザードを使用して、データベースまたはデータベーススキーマからの関係 CMP Bean のセットを最初に作成するのが良い方法です。こうすると、ウィザードによって EJB モジュールも Bean のセットと共に作成されます。また、Bean 間の関係も IDE によって自動的に保たれます。

既存の EJB モジュールに エンタープライズ Bean を 1 つ追加したい場合は、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで EJB モジュールのノードを右クリックし、「EJB を追加」を選択します。

ファイルシステムがツリー表示された「EJB を EJB モジュールに追加」ダイアログが表示されます。

2. ツリー表示から エンタープライズ Bean を選択します。

3. 「了解」をクリックします。

CMP エンティティ Bean へのデータベース関連プロパティの設定

CMP エンティティ Bean に対する RI 関連のプロパティは、EJB モジュールのレベルで指定します。図 8-2 に `EJBModule_AccountsSouth` という名前の EJB モジュールの「J2EE RI」タブの表示例を示します。この EJB モジュールには、関係 CMP Bean がいくつか含まれています。このタブにあるフィールドは、Bean とそれらが基づいているデータストレージとの関係、およびそれらの関係を管理するためにコンテナが果たす役割についてを指定します。

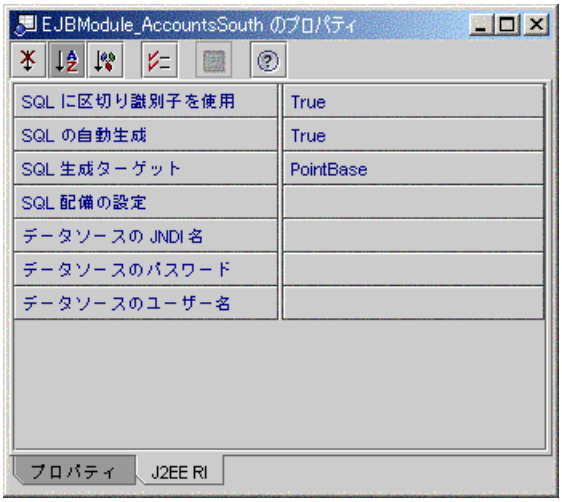


図 8-2　CMP Bean を含む EJB モジュールのプロパティを表示した「J2EE RI 」タブ

注 - CMP Bean に対して IDE のテスト機能を使用する場合、EJB モジュールは後で作成します。テスト機能を実行すると EJB モジュールが自動的に作成されます。これは、アプリケーションに実際に配備できるモジュールではなく、CMP Bean のコードと配備に関する設定を検査するためのものです。テストプロセスから CMP Bean を実行することで、開発時間を短縮できます。テストでは、後述する手順に従って、設定したプロパティの動作状態を見たり、実際に使用する EJB モジュールでの設定を再現したりできます。エンタープライズ Bean のテストについては、第 9 章を参照してください。

CMP Bean それぞれに対して、次のフィールドを設定します。

- **データソースの JNDI 名** - エンティティの状態に合わせて更新するデータストアの JNDI 名を指定します。`jdbc\Pointbase` と指定した場合、`jdbc` は、データストレージの種類を表し、`Pointbase` は特定のデータストアを表しています。
- **データソースのパスワード** - データベースにアクセスするのに必要なパスワードを指定します。
- **データソースのユーザー名** - データベースにアクセスするのに必要なユーザー名を指定します。

注 - IDE で提供されている PointBase データベースサーバーを使用している場合、JNDI 名を入力する際には、前述の例で示しているとおりに Pointbase の最初の 1 文字を大文字で指定してください。デフォルトのユーザー名とパスワードは同じで、`pbpublic` です (パスワードを入力したら Enter キーまたはリターンキーを必ず押してください)。

EJB モジュールに CMP Bean を格納した方法、および CMP Bean が既存のデータベース表を利用する方法によって、残りの RI 関連のプロパティフィールドに対する指定は異なります。

RI の生成する SQL

EJB ビルダーのウィザードを使用して CMP Bean を作成し、その際に「SQL の自動生成」フィールドを `True` のままにしておいた場合、J2EE RI プラグインによって SQL 文が提供されます。SQL 文は、CMP Bean が対応するデータベース中の持続レコードを管理するものです。プラグインは、Bean のメソッドとフィールド、それにプログラマが特別にメソッドに記述した EJB QL コードに基づいて、それぞれのサーバー固有の SQL 文を生成します。これは IDE をインストールしたサーバーすべてで実行されます。どのサーバーのプラグインであっても、Bean のコードと記述された EJB QL に基づいて、そのサーバー固有の SQL 文を生成します。

プラグインは、CMP フィールドの追加時、CMP フィールドの削除時、主キーを構成するフィールドの変更時、および Bean のコンテナ管理による関係の変更時に、SQL 文を再生成します。この SQL 生成機能を使用するかどうかは選択できます。使用したい場合は、「SQL の自動生成」フィールドを True に設定して、SQL 文は変更しないでください。ただし、EJB モジュールに バージョン 1.x Bean クラスを持つ CMP Bean

が含まれている場合は、変更が必要です (この場合の変更が必要な 1 文については 192 ページの「EJB 1.1 CMP エンティティ Bean を含んだ EJB モジュール」を参照してください)。

生成された SQL 文は次の処理を実行します。

- Bean が配備された際に表を作成する。表は <*Bean* 名>Table と名付けられます。表の列には CMP フィールドに対応した名前が付けられます。
- Bean の配備が解除されたときに表も削除する。
- Bean の実行に応じてレコードを削除、挿入、選択、および更新する。

次の節では、CMP Bean のデータベース関連を設定する際のガイドラインについて説明します。

CMP Bean による新規データベース表の使用

テストの最中など、既存のデータベース表ではなく、RI プラグインが新しく作成した表を CMP Bean に使用させる場合があります。このとき、「SQL の自動生成」フィールドと「SQL に区切り識別子を使用」フィールドはデフォルトの設定のままにしておきます。

注 - Bean 名、CMP フィールド名、および CMR フィールド名がデータソースで指定された SQL データベースでの予約語であることがはっきりしている場合は、区切り識別子だけは False にしておきます。

「SQL 生成ターゲット」フィールドのデフォルト設定を使用するか、他のデータベースを指定します。

表の作成と削除に関するデフォルト設定を参照するには、「SQL 配備の設定」フィールドをクリックし、省略符号ボタン (...)をクリックします。図 8-3 に示すように「SQL 配備の設定」プロパティエディタが表示されます。

図 8-3　EJB モジュール内の CMP エンティティ Bean の表関連の設定

表関連の設定は、Bean ノードが選択されたときに表示されます。

CMP Bean 用に RI プラグインによって生成されたデフォルトの SQL 文を参照するには、このプロパティエディタで Bean ノードを展開し、Bean のメソッドのうちどれかを選択します。図 8-4 に示すようにメソッドの SQL 文が表示されます。

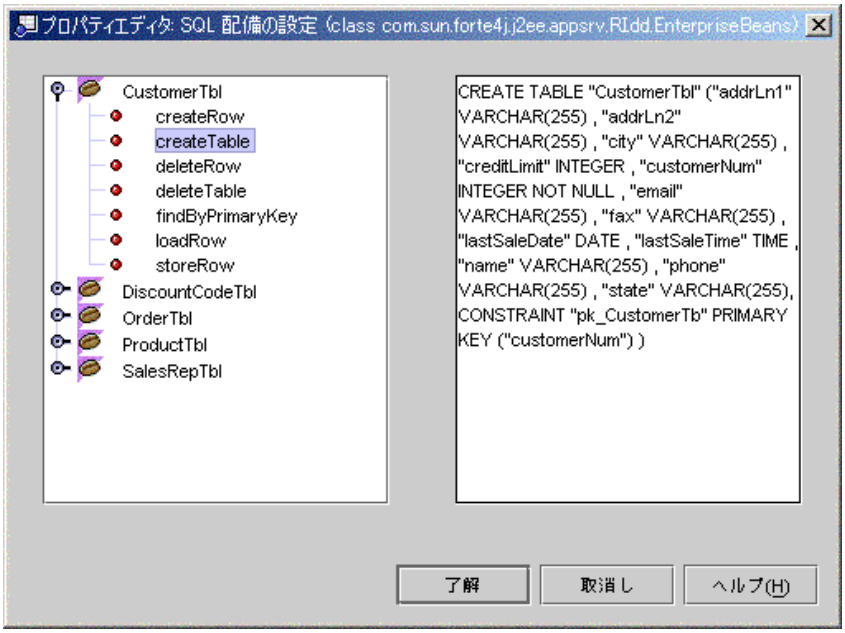


図 8-4　CMP Bean の `createTable` メソッドに対して RI プラグインが生成した SQL 文の例

サーバーが生成した SQL 文は変更しないでください。ただし、EJB モジュールに EJB 1.1 CMP Bean、つまり Sun ONE Studio を使用して バージョン 1.x Bean クラスを持つ CMP Bean がある場合は変更が必要です。この場合は、192 ページの「EJB 1.1 CMP エンティティ Bean を含んだ EJB モジュール」を参照してください。

注意 - 自動生成された SQL 文に対して Bean のプロパティシートから変更を加えた場合、変更を防ぐ機能は IDE にはありません。逆に加えた変更は、 Bean の EJB QL 文すべてに反映され、すべてのアプリケーションサーバーのプラグインによって変更が自動的に伝播されます。Bean のサーバー固有の SQL 文も生成されます。ただし、自動生成された SQL 文では加えた変更が反映されないこともあります。

CMP Bean による既存データベース表の使用

CMP Bean が既存のデータベース表を使用する必要がある場合は (Bean のインフラストラクチャを表から生成し、その表に対して Bean をテストしたい場合など)、次のように設定します。

SQL の自動生成	True (デフォルト)
SQL 生成ターゲット	(表を含んだデータベース名。これには「データソース JNDI 名」での指定が対応付けられるのと同じデータベースを指定する。)
SQL に区切り識別子を使用	False

「SQL の自動生成」フィールドに `True` が指定されていると、CMP Bean の持続フィールドとコンテナ管理による関係を変更した際に、RI プラグインによって SQL 文が自動生成され、最新の状態に保たれます (これには 1 つだけ例外があります。例外については、次の節を参照してください)。

注 - Bean (表) または Bean のフィールド (列) のどちらにも SQL での予約語が使用されていないことを確認してください。

「SQL 配備の設定」フィールドをクリックし、省略符号ボタン (...) をクリックします。表示されるプロパティエディタで、表と照らし合わせてテストしたい各 CMP Bean を選択します。選択したそれぞれの Bean の「配備時の表の作成」および「配備取り消し時の削除」のチェックボックスからチェックを外します。こうしておくと、RI プラグインによって Bean が使用するための表も作成されませんし、アプリケーションが実行されなくなったときに表が削除されることもありません。

前述したとおり、EJB モジュールに EJB 1.1 CMP Bean が含まれている場合を除き、自動生成された SQL 文は変更しないでください。

EJB 1.1 CMP エンティティ Bean を含んだ EJB モジュール

アプリケーションには、Sun ONE Studio IDE の旧バージョンを利用して作成された、以前の EJB 仕様に基づいた CMP エンティティ Bean が含まれていることがあります。また、アプリケーションで使用されている CMP Bean が、バージョン 1.x Bean クラスに基づいている場合があります (CMP Bean の作成についての詳細は、第 4 章

を参照してください)。このような場合、1 つだけ、重要な変更を施すことによって、CMP Bean を EJB 2.0 CMP Bean と同様に処理できます。これには、自動生成された SQL 文の検索メソッドにある WHERE 文を変更します。

この変更を加えるには、EJB モジュールのプロパティシートの「J2EE RI」タブにある「SQL 配備の設定」フィールドをクリックします。その後、省略符号ボタン (...) をクリックすると、「SQL 配備の設定」プロパティエディタが表示されます。

WHERE 句を編集するには、次の操作を実行します。

1. EJB 1.1 CMP Bean のノードを展開します。
2. Bean の検索メソッドのノードを選択します。

   図 8-5 に示すような SQL 文が表示されます。

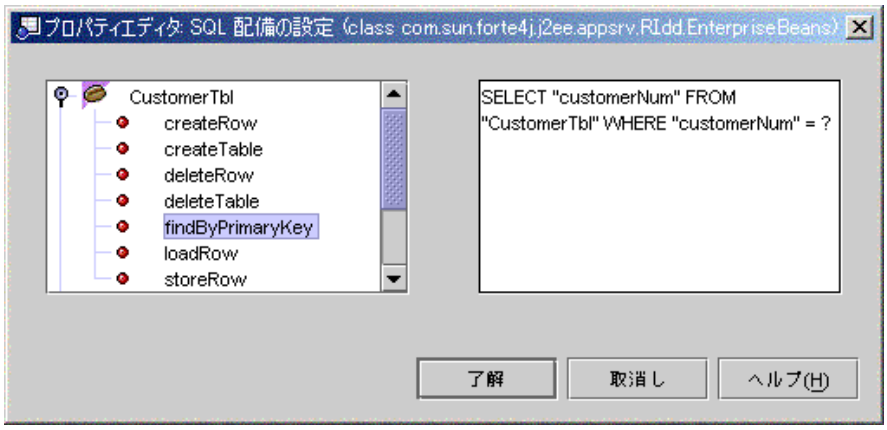


図 8-5　CMP Bean の検索メソッドに対して生成された SQL 文の例

3. WHERE 文を完成します。

   - FindByPrimaryKey メソッドにプラグインが自動生成した SQL を使用していない場合は、カラム名を変更できます。自動生成した SQL を使用している場合は、カラム名は変更しないでください。
   - その他の検索メソッドに対しては制約が必要です。たとえば、findByY2000Accounts という検索メソッドが次のように記述されていたとします。

     ```
     findInCustIDRange(double low, double high)
     ```

     この場合、SQL 文は次のようになっています。

```
SELECT “custID” FROM “CustomerTbl” WHERE “custID” > ?2000 AND
“custID” < ?2001
```

疑問符に続く数字は、メソッドのパラメータリストにある、対応するパラメータを示しています。

WHERE 文での変更は、RI プラグインが SQL 文を再生成したときにも保持されます。

IDE の現在のバージョンでの EJB 1.1 CMP エンティティ Bean の取り扱いについては付録 B を参照してください。

CMP フィールドの値の順序

RI プラグインが自動生成した SQL 文の一部でも変更する場合は、CMP フィールドの値の順序について理解しておいてください。

コンテナは、CMP フィールドの値をデータベースにアルファベット順に送信します。表の列に CMP フィールド名と同じ名前がなかった場合、これらの列に、対応付けられるフィールドと同じアルファベット順で名前を付けてください。EmployeeEJB という名前の CMP Bean での例を次に示します。

- ソースデータベース表の名前は Employee と指定します。
- CMP フィールドが address、city、id、および name という名前だったとします。
- 対応する列は Mail_Address、City、Emp_ID、および Emp_Name とします。
- INSERT 文に生成されるデフォルトの SQL は次のようになります。

```
INSERT INTO "EmployeeEJBTable" ("address", "city", "id",
"name") VALUES (?, ?, ?, ?)
```

この例では、SQL を次のように変更する必要があります。

```
INSERT INTO "Employee" ("Mail_Address", "City", "Emp_ID",
"Emp_Name") VALUES (?, ?, ?, ?)
```

表列名は、対応する CMP フィールドと同じアルファベット順 (表列のアルファベット順ではありません) になっている必要があります。これは、VALUES パラメータが CMP フィールドのアルファベット順で送信されるためです。

EMB モジュールへのトランザクション属性の追加

トランザクション属性は、EJB コンテナに CMT Bean のトランザクションの制御方法を指示します。自身のトランザクションを管理するセッション Bean (BMT Bean) では、Bean のトランザクションを明示的に記述する必要があります。CMT Bean では、トランザクションを明示的に記述する必要はありませんが、代わりに Bean のメソッドにトランザクション属性を指定します。コンテナはこの属性に基づいてトランザクションを制御します。

デフォルトでは、IDE はすべての Bean メソッドに「Required」属性を設定します。トランザクション属性は Bean レベルとメソッドレベルに別々に指定することができます。たとえば、あるメソッドをトランザクションのコンテキストに含めたくない場合は、「Required」から「NotSupported」に変更できます。

トランザクション属性は、配備記述子に保存され、EJB モジュールのプロパティシートから編集することができます。CMT Bean を含んだ EJB モジュールがアプリケーションに組み込まれる前に、そのモジュールに適切なトランザクション属性が設定されていることを確認する必要があります。

個々の Bean またはメソッドに対するトランザクション属性を変更した場合、その属性はその EJB モジュール内の実行に対してだけ変更されます。Bean のソースコードは変更されません。同じ Bean を他の EJB モジュールで再利用したい場合は、別のトランザクション属性のセットを適用することができます。

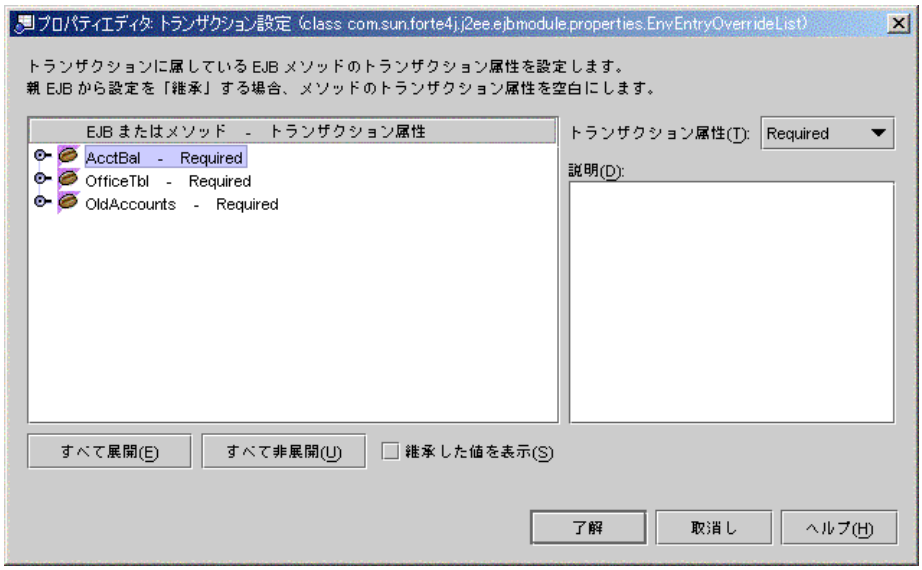
EJB モジュール内で CMT Bean のトランザクション属性を変更するには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで、Bean の EJB モジュールを右クリックして「プロパティ」を選択します。

   「プロパティ」タブの「トランザクション設定」フィールドには、 コンテナトランザクションと表示されています。

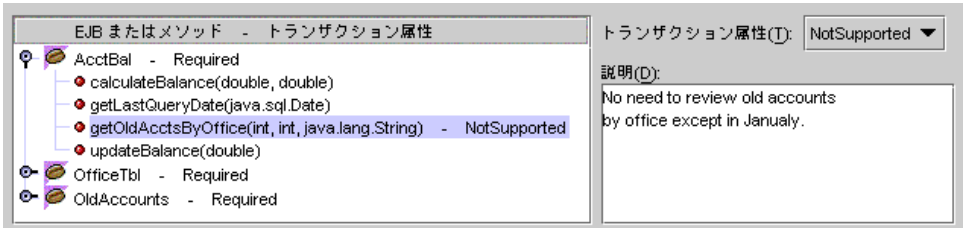
2. 「プロパティ」タブの「コンテナトランザクション」フィールドをクリックし、省略符号ボタン (...) をクリックします。

   「トランザクション設定」プロパティエディタが表示されます。大きいほうの区画に CMT を使用するモジュールの エンタープライズ Bean が、それぞれの Bean 全体に適用されるトランザクション属性と共に表示されます。



- Bean のトランザクション属性を変更するには、Bean を選択してから「トランザクション属性」コンボボックスの別の項目を選択します。新しい属性が大きい区画にある Bean 名の横に表示されます。

- 特定のメソッドのトランザクション属性を変更するには、Bean を展開してメソッドを選択し、「トランザクション属性」コンボボックスの別の項目を選択します。次の例のように、大きい区画にあるメソッド名の横に新しい属性が表示されます。Bean の他のメソッドにはトランザクション属性が表示されていませんが、これはデフォルトの属性が変更された場合にだけ表示されるためです。



「トランザクション設定」プロパティエディタの「説明」には、モジュール中の個々の Bean やメソッドのトランザクション属性に関する説明を、アプリケーションのアセンブル担当者のために記述しておくことができます。この EJB モジュールで、特定のトランザクション属性を変更した理由などを記述しておきます。

3. トランザクション属性に対する操作が終了したら「了解」をクリックします。

EJB モジュールまたはアプリケーション内での EJB 参照の変更

174 ページの「EJB ローカル参照の指定」および 176 ページの「EJB 参照の指定」では、個々の Enterpris Bean レベルでの Bean 間での参照の宣言の指定について説明しました。これらの参照は、次の場合に他の指定が優先されることがあります。

- 該当する複数の Bean が同じ EJB モジュールにある場合、これらの参照にを EJB モジュールのレベルで指定した参照で上書きすることができます。
- 該当する複数の Bean が同じアプリケーション内にはあるが、異なる EJB モジュールにある場合、これらの参照をアプリケーションレベルで指定した参照で上書きすることができます。

この機能は、ある エンタープライズ Bean を 3 つの異なる EJB モジュールに格納し、1 つ以上のアプリケーションで多様に利用したいといった場合などに便利です。

この機能を使用するには、同じインタフェースが指定されている複数の エンタープライズ Bean を持つ EJB モジュール (またはアプリケーション) が必要です。この機能はローカルインタフェース、リモートインタフェース、またはローカルインタフェースとリモートインタフェースの両方で使用できますが、それぞれの Bean のインタフェースは同じである必要があります。ただし、クラスやプロパティ (配備情報) は異なっていても構いません。

モジュールレベルでの参照指定の優先

Bean の参照を EJB モジュールの指定で上書きするには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで Bean の EJB モジュールを右クリックし、「プロパティ」を選択します。

2. 表示された参照フィールド (「EJB ローカル参照」または「EJB 参照」) をクリックし、省略符号ボタン (...) をクリックします。

   適切なプロパティエディタが表示されます。

3. 上書きする参照を持つ エンタープライズ Bean を見つけ、対応する「上書き」チェックボックスをクリックします。

4. 「値を上書き」フィールドをクリックします。

   「値を上書き」フィールドの下に、エンタープライズ Bean 名を選択するコンボボックスが表示されます。図 8-6 にコンボボックスの例を示します。

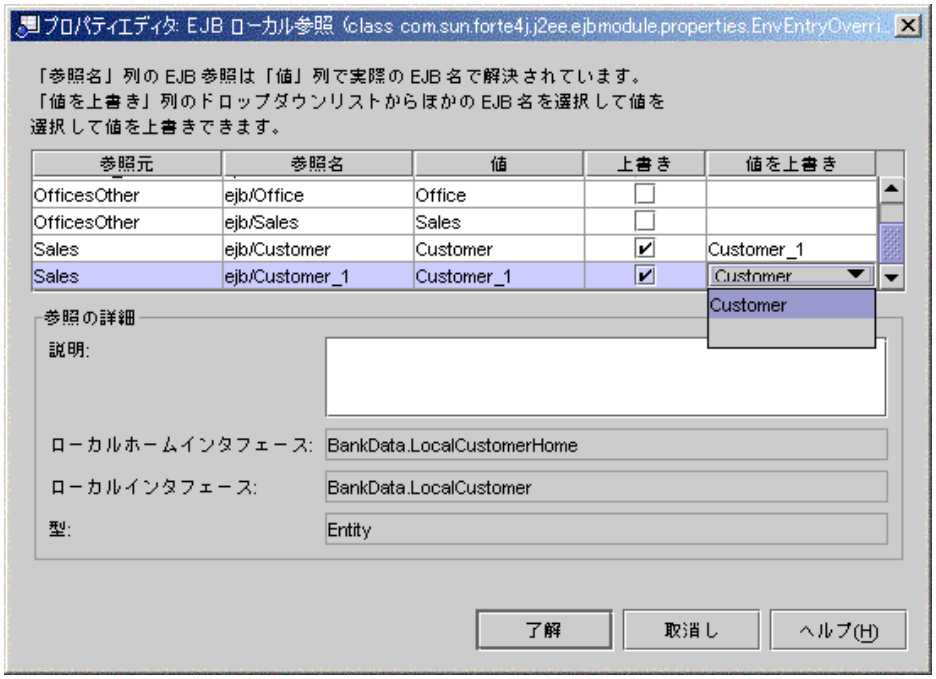


図 8-6　エンタープライズ Bean のローカル参照の上書きを選択した「EJB ローカル参照」プロパティエディタ

5. エンタープライズ Bean に使用させたい参照を選択し、「了解」をクリックします。

アプリケーションレベルでの参照指定の優先

Bean の参照をアプリケーションの指定によって上書きするには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで Bean のアプリケーションを右クリックし、「プロパティ」を選択します。
2. 表示された参照フィールド (「EJB ローカル参照」または「EJB 参照」) をクリックし、省略符号ボタン (...) をクリックします。

   適切なプロパティエディタが表示されます。

   EJB 参照と EJB ローカル参照のプロパティエディタは、モジュールとアプリケーションレベルでは同じです。これ以降は、直前の節での操作手順に従ってください。

EJB JAR ファイルの作成

EJB モジュールの作成後、アプリケーションのアセンブルのためにモジュールを他の操作で使用する場合は、EJB JAR ファイルを作成しておきます。このファイルはモジュールやその内容をアプリケーションで使用するためのエクスポートをするために使用されます。EJB JAR ファイルを作成する場合は、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで EJB モジュールを選択します。
2. 右クリックして「EJB JAR ファイルをエクスポート」を選択します。

   出力ウィンドウにモジュールとその内容のコンパイル実行状態が表示されます。エラーメッセージがある場合は、このウィンドウに表示されます。

   EJB JAR ファイルが作成された後でも、モジュールやその内容をこの EJB JAR ファイルに限らず、他の EJB JAR ファイル用にも調整できます。

   1 つ以上の EJB モジュールを J2EE アプリケーションに格納してから配備することができます。アセンブルと配備の詳細については、マニュアル『J2EE アプリケーションのプログラミング』を参照してください。

EJB モジュールへの JAR ファイルの追加

ある EJB モジュールにある エンタープライズ Bean に、他のコンテナにある Bean が提供する機能を使用させることができます。たとえば、ユーザーを認証するセッション Bean 用に、セキュリティ検査を実行する Bean を含んだ JAR ファイルを追加することができます。既存の EJB JAR ファイルを EJB モジュールに追加するには、次の操作を実行します。

1. エクスプローラの「ファイルシステム」タブで、JAR ファイルを含んだファイルシステムをマウントします。

2. EJB モジュールのノードを右クリックし、「プロパティ」を選択します。
3. モジュールのプロパティシートで「追加のファイル」フィールドをクリックし、省略符号ボタン (...) をクリックします。
4. 「ソース」区画で追加したい JAR ファイルを表示した後、ファイルを選択し、「追加」をクリックします。

   「追加されるファイル」区画に JAR ファイルが表示されます。
5. 「了解」をクリックして JAR ファイルを追加します。

   EJB モジュールを配備すると、この JAR ファイルがモジュールの EJB JAR ファイルに含まれます。

第9章

エンタープライズ Bean のテスト

エンタープライズ Bean を開発していると、全面的なアプリケーションのアセンブルや最終的なアプリケーションサーバーへの配備を実行する前に、Bean のテストができれば便利なことがあります。Sun ONE Studio IDE を使用すると、テストを目的とした J2EE アプリケーションを生成することができます。このアプリケーションには、JavaServer Pages™ (JSP™) によるテスト用のページを持つ Web モジュールや Bean 用の EJB モジュールも含まれます。その後、JSP ページの結果が出力される HTML ページを Web ブラウザで表示するのに IDE のテスト機能を使用することができます。HTML ページでは、エンタープライズ Bean のインスタンスを作成したり、Bean のメソッドを実行したりできます。

IDE が作成するオブジェクトは、テストでの使用を目的として作成されるもので、実際の環境に配備できるようには作成されていません。

IDE のテスト機能は、サポート対象となっているデータベースおよびアプリケーションサーバーで使用できます。以降に記述する手順と例では、テストデータベースとして PointBase を、テストサーバーとして J2EE RI アプリケーションサーバーを、そして Web ブラウザとして Netscape Navigator を使用しています。

テストの前提条件

設定は Bean の種類によって異なります。どの種類の エンタープライズ Bean をテストするかによって、テストの設定も変わります。次に前提となる条件をいくつか挙げます。

- **EJB モジュールの設定の検討** - セッション Bean をテストする場合は、テスト機能によって生成された EJB モジュールを使用できます。ただし、エンティティ Bean の EJB モジュールは 2 つの異なる方法で操作できます。214 ページの「CMP または BMP Bean のテスト」を参照してください。

  現バージョンの IDE では、メッセージ駆動型 Bean をテストすることはできません。

- **すべての参照 Bean は同じモジュールに含める** - テスト機能を使用すると、Bean を 1 つずつ実行していくことができます。ただし、テストする Bean からの参照先は使用できるようにしておく必要があります。

- **リモート参照の用意** - テスト対象となる エンタープライズ Bean には、リモートインタフエースが必要です。ローカルインタフェースが指定されていても構いませんが、テストにはリモートインタフェースが必要です。テスト対象となる Bean から参照される Bean にはローカルとリモートのどちらでも、または両方のインタフェースを指定できます。詳細は、215 ページの「Bean へのリモートインタフェースの追加」を参照してください。

- **Bean のリソースの準備** - RI と必要なデータベースを稼動させておきます。詳細は次の節を参照してください。

エンタープライズ Bean のテストに関する様々な設定については、214 ページの「CMP または BMP Bean のテスト」で説明しています。

J2EE RI への配備の準備

エンタープライズ Bean に対して IDE のテスト機能を実行する場合、Bean のアプリケーションを RI に配備できるようにしておく必要があります。この RI はテスト用のマシンにローカルにインストールする必要があります。最低でも 1 つの RI サーバーインスタンスを実行しておきます。

RI サーバーのインストールとインスタンス化は、IDE のインストール時に自動的に実行されます。これに関する詳細は RI のオンラインヘルプを参照してください。

PointBase データベースでの Bean のテストの準備

データベースへのアクセスが必要な エンタープライズ Bean を、IDE の一部である PointBase データベースを使ってテストすることができます (通常これはエンティティ Bean です)。

- テスト機能で、テスト用の EJB モジュールを作成する場合は、次の手順に従ってプロパティを設定してください。
- エンティティ Bean のテストに既存の EJB モジュールを使用する場合は、テストアプリケーションがデータベースを見つけてログインできるようにモジュールのプロパティを設定してください。

プロパティは次のように設定します。

1. IDE の「エクスプローラ」ウィンドウで EJB モジュールのノードを右クリックし、「プロパティ」を選択します。

   モジュールのプロパティウィンドウが表示されます。

2. 「プロパティ」ウィンドウの J2EE RI タブで、データベース接続を次のように指定します。

フィールド	指定内容
データソースの JNDI 名	`jdbc/Pointbase`
データソースのパスワード	`PBPUBLIC`
データソースのユーザー名	`PBPUBLIC`

注 - このウィンドウで指定する **`Pointbase`** の最初の 1 文字は大文字で指定してください。パスワードの入力後は Enter キーを押してください。

RI のデフォルト設定のうちの 2 つの指定に基づき、テスト対象となる Bean 用のテーブルがサーバーによって作成されます。Bean に、稼動中の特定のデータベースからのデータが必要な場合は、次の変更を加えてください。

3. 「SQL 配備の設定」フィールドをクリックし、省略符号ボタン (...) をクリックします。

   「SQL 配の備設定」プロパティエディタが表示されます。

4. テスト対象となる Bean を選択し、次の 2 つのチェックボックスからチェックを外してください。

   - 配備時の表の作成
   - 配備取り消し時の表の削除

5. 「了解」をクリックして、EJB モジュールのプロパティシートを閉じます。
6. 「ファイル」->「すべてを保存」を選択して、操作内容を保存します。

データベースサーバーと Web ブラウザの開始

PointBase データベースサーバーを開始するには、メインメニューから「ツール」->「PointBase ネットワークサーバー」->「サーバーを起動」を選択します。

通常の操作手順で Web ブラウザを開始します。

これらのウィンドウは最小化しても構いませんが、テストがすべて完了するまでは終了しないでください。

テストオブジェクトの生成

これで エンタープライズ Bean のテストを実行する EJB モジュール、Web モジュール、およびアプリケーションを作成する IDE ウィザードを使用する準備ができました。

テストする エンタープライズ Bean のオブジェクトを作成するには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで Bean の論理ノードを選択し、右クリックしてから「新規 EJB テストアプリケーションを作成」を選択します。
   アプリケーションのテストに必要なすべてのコンポーネントにデフォルト値が指定された状態で、ウィザードが表示されます。

注 - Bean を右クリックして表示されたメニューで、「新規 EJB テストアプリケーションを作成」項目が無効になっていたら、Bean にはリモートインタフェースがありません。215 ページの「Bean へのリモートインタフェースの追加」の操作を実行後、再びメニューを確認してください。

「パッケージ」フィールドには、Bean が含まれる現在のパッケージ名が表示されます。このフィールドに別のパッケージ名を入力すると、作成される EJB モジュールと、テスト用のアプリケーションオブジェクトを移動できます。

次のフィールドでも、必要に応じて別のパッケージとモジュール名を指定できます。ただし、テスト用に生成される EJB モジュールではなく、既存の EJB モジュールを使用したい場合は、ここで既存の EJB モジュール名を指定しないでください。代わりに、アプリケーションのプロパティシートで後に指定します。

「アプリケーションサーバー」コンボボックスの選択内容を確認します。テスト用のサーバーとして RI を使用するか、デフォルトサーバーとして指定されている別のサーバーを使用することができます。

2. **自動配備チェックボックスにチェックを入れるか、空白のままにします。**

   チェックボックスにチェックを入れた場合は、ウィザードで Bean のテストモジュールを作成した直後に、IDE によってそのテストモジュールがサーバーに自動的に配備されます。テスト対象となっている Bean が独立型で他の Bean と協調動作しない場合は、自動配備を選択します。

   このチェックボックスを空白のままにすると、手動による配備が後に必要となります。他の Bean と協調させながら Bean をテストする場合には、自動配備は選択しません。

3. 「了解」をクリックして、EJB モジュール、Web モジュール、およびアプリケーションを作成します。(指定があれば) アプリケーションは自動的に配備されます。

   モジュール生成と配備の進行状況が表示されます。配備が完了すると IDE メインウィンドウのステータスバーにメッセージが表示されます。

   ウィザードにパッケージの内容を自動生成させた場合、`Accounts_North` というパッケージの `Customer` Bean には、図 9-1 に示すようなオブジェクトが生成されます。

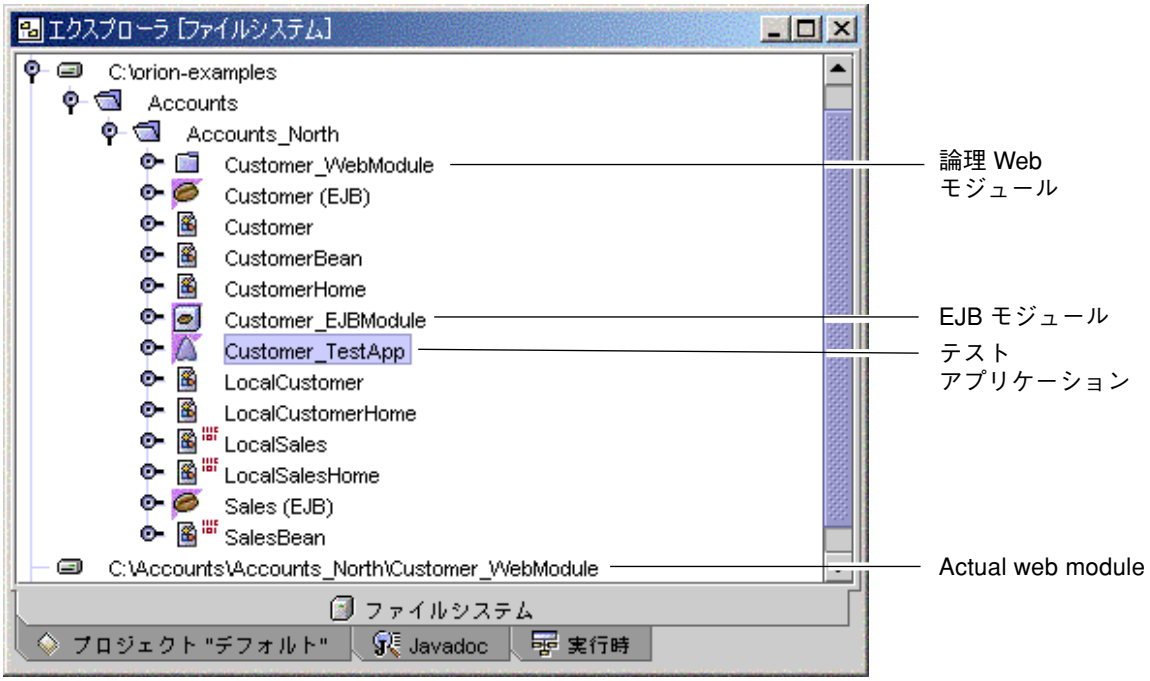


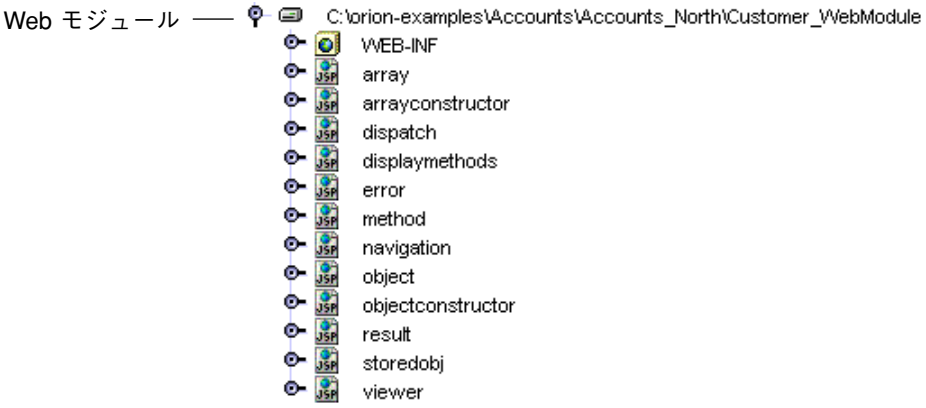
図 9-1　生成されたエンタープライズ Bean のテストオブジェクトの例

Bean のパッケージに追加された生成オブジェクトは次のとおりです。

- 論理 Web モジュール
- テスト対象となる エンタープライズ Bean を含んだ EJB モジュール
- EJB モジュールと Web モジュールを含んだテストアプリケーション

ウィザードで特に指定がない場合は、IDE は実際の Web モジュールを新しいファイルシステムに単独で保存します (JSP ページとヘルパー Java クラスを含みます)。

また、ウィザードは、テスト用の Web モジュールと他のオブジェクトを保存するための新規ファイルシステムも作成します。このファイルシステムのノードを展開すると、Web モジュールが最初のサブノードとして表示されます。次に示すこの Web モジュールには、JSP ページとヘルパー Java クラスが含まれます。



これで基本的なテストアプリケーションが出来上がりました。テストする Bean が他の Bean を参照している場合は、次の手順に従って、参照先の Bean を EJB モジュールに追加します。

4. 「エクスプローラ」ウィンドウで生成された EJB モジュールのノードを選択し、右クリックしてから「EJB を追加」を選択します。

5. ツリー表示で、参照先の Enterpirse Bean を表示し、選択してから「了解」をクリックします。

   参照先 Bean が EJB モジュールに追加され、その Bean への参照がテストアプリケーションに追加されます。

   参照される Bean ごとに 手順 4 と 手順 5 を繰り返します。

   これで、テストアプリケーションを配備する準備ができました。手順 2 で自動配備チェックボックスを選択した場合は自動的にアプリケーションが配備されます。また、アプリケーションの配備と実行をまとめて処理することもできます。次の節では、配備だけの処理と、配備と実行の一括処理の両方について説明します。

サーバーへのテストアプリケーションの配備

テストアプリケーションを RI に配備するには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで J2EE アプリケーションノードを選択し、右クリックしてから「配備」を選択します。

   配備の進行状況がモニターと出力ウィンドウに表示されます。RI (`localhost` として参照されます) にテストアプリケーションが配備されたことなどを示すメッセージが表示されます。

2. テストアプリケーションが配備されたことを確認します。RI インスタンスの出力ウィンドウを開いて、"Application *name*_TestApp deployed." のメッセージがあることを確認してください。

参照 - 配備に失敗した場合は、IDE の J2EE RI 実行に関する設定が正しいかどうか確認してください。特に、`RI_Home` プロパティが *J2EE_HOME* の値に設定されているかどうかを調べてください。

配備が正常に終了したら、209 ページの「テストアプリケーションの実行」の手順に進んでください。

テストアプリケーションの配備と実行の一括処理

テストアプリケーションをサーバーに配備して、そのまま実行も開始するには、次の操作を実行します。

- 「エクスプローラ」ウィンドウで J2EE アプリケーションノードを選択し、右クリックしてから「実行」を選択します。

  配備と実行の進行状況がモニターと出力ウィンドウに表示されます。アプリケーションサーバーに接続したこと、エンタープライズ Bean が RI (`localhost` として参照されます) に配備されたこと、ラッパーと RMI-IIOP コードがコンパイルされたこと、サーバーとクライアント用の JAR ファイルが作成されたこと、Web サーバーが接続されテストアプリケーションの実行を行うこと、そしてすべての生成コードが保存されたことなどを示すメッセージが表示されます。

この一括処理が完了すると、Web ブラウザが表示されます。Web ブラウザには、テストクライアント、つまり、エンタープライズ Bean をテストするための GUI 機能を含んだ JSP ページが表示されます。図 9-2 に Web ブラウザの表示例を示します。

209 ページの「テストクライアントを使用した Bean のテスト」に続きます。

テストアプリケーションの実行

テストアプリケーションの配備と実行を一括処理するための「実行」を選択しなかった場合、エンタープライズ Bean のテスト実行をするには、次の操作を実行します。

- Web ブラウザを開いて、適切な URL を入力します。

  J2EE RI を使用している場合の URL は次の形式になっています。

  `http://localhost:`<ポート>`/`<アプリケーション名>`/`

  <ポート> は RI のインストール時に指定したポート番号です。

  <アプリケーション名> はアプリケーションの名前です。

  テストアプリケーションのクライアント、つまり、JSP ページがブラウザに表示されます。

テストクライアントを使用した Bean のテスト

テストクライアントの JSP ページの手順に従って、エンタープライズ Bean のインスタンスを作成し、そのビジネスメソッドを呼び出します。次の節では、US ドルを日本円に換算する `dollarToYen` という単純なセッション Bean をテストする方法を説明します (このセッション Bean は `Converter` という Java パッケージに含まれています)。

テストクライアントページとは

図 9-2 に、アプリケーションクライアント用に IDE が作成した JSP ページの例を示します。この JSP ページは、例題のセッション Bean のテスト用に生成されたものです。

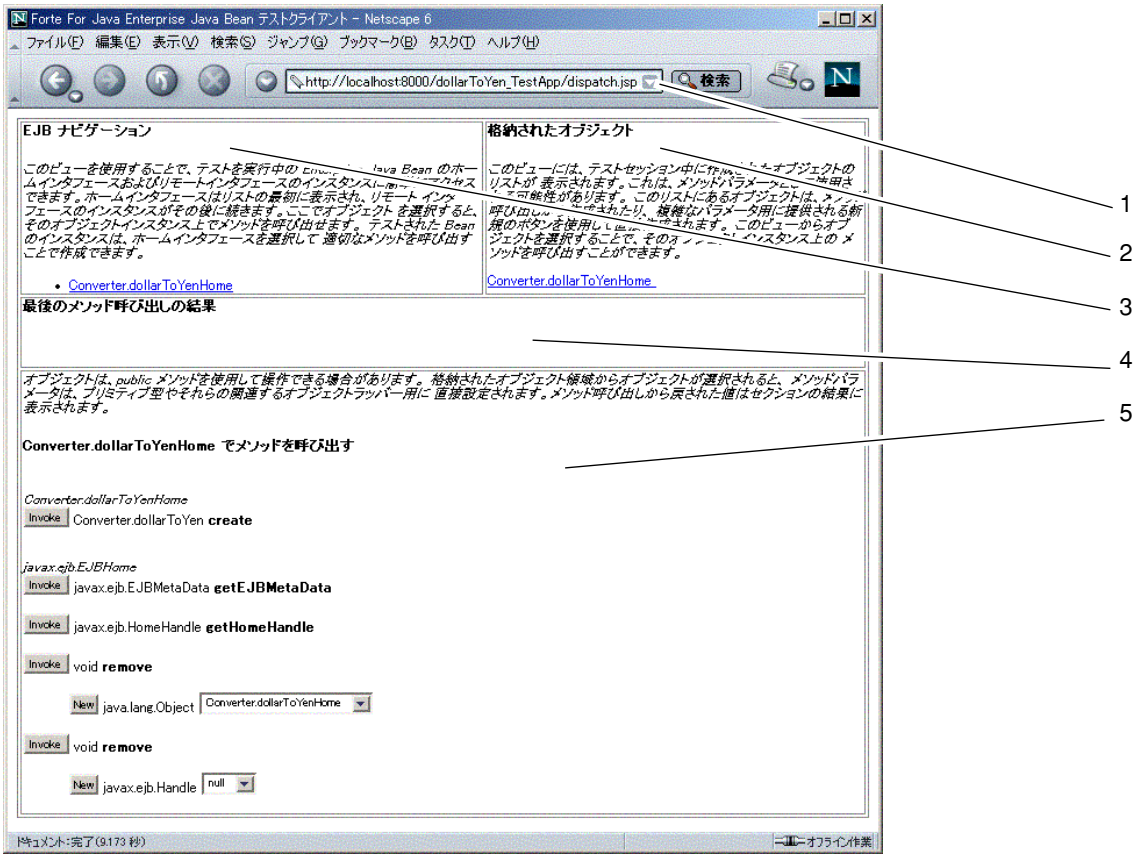


図 9-2　dollarToYen セッション Bean のテスト用に生成されたクライアント JSP ページ

図 9-2 で示しているテスト用ウィンドウの表示項目を次に説明します。

1. ブラウザの URL フィールドには、テストクライアントの JSP の場所が表示されています。クライアントの URL は IDE のテスト機能によって自動的に生成されます。この URL を指定してテストウィンドウに戻ることもできます。

2. 「格納されたオブジェクト」区画には、オブジェクトがまとまって表示されます。これらのオブジェクトは、テスト中に IDE が作成したり、プログラマによってインタフェース中または Bean クラス中のメソッドへの呼び出しなどの結果、作成されたものです。この時点では、ホームインタフェースだけが表示されています。

   「格納されたオブジェクト」中のオブジェクトは、「Remove Selected」または「Remove All」ボタンを使用して削除できます。

3. 「EJB ナビゲーション」区画には、IDE が Bean のテスト用に生成したオブジェクトが表示されます。テスト中の Bean が他の Bean を参照していて、EJB モジュールに複数の Bean が含まれている場合、この区画には作成されたオブジェクトが表示されます。これらのオブジェクトは、モジュール中の Bean が他の Bean を呼び出してオブジェクトを作成した順、つまり論理的な順序で表示されます。

   図 9-2 で示されている「EJB ナビゲーション」区画には、`Converter.dollarToYenHome` が表示されています。これは、セッション Bean のホームインタフェースが IDE によって作成されたことを示しています。後でこのホームインタフェースをクリックして、セッション Bean の新しいインスタンスを生成・初期化します。

   この区画に複数のオブジェクトが表示されている場合は、オブジェクトのどれかをクリックして、テストする Bean コンポーネントを変更してください。オブジェクトを 1 つクリックするたびに、他の区画の表示内容が変わります。

4. 「最後のメソッド呼び出しの結果」区画には、直前に呼び出したメソッドとそのパラメータなどが表示されます。この時点では、セッション Bean のテストをはじめていないのでこの区画には何も表示されていません。

5. 最下部の区画には、テストで使用できるメソッドが表示されます。この区画に表示される内容は、選択している Bean のコンポーネント (「EJB ナビゲーション」区画に表示されます) によって変わります。

次に `dollarToYen` Bean のホームインタフェースとビジネスメソッドをテストしてみます。

例題 Bean のホームインタフェースのテスト

`Converter.dollarToYenHome` がセッション Bean のインスタンスを正しく作成するかどうかを検査するには、次の操作を実行します。

1. 「EJB ナビゲーション」区画でホームインタフェース名をクリックします。

2. 最下部の区画で、ホームインタフェース名の下の「Invoke」ボタンをクリックします。

   この例では、`Converter.dollarToYenHome` の下の「Invoke」ボタンをクリックします。ページ内の区画が次のように変化します。

- 「EJB ナビゲーション」区画には Bean のインスタンスが追加されます。この例では `Converter.dollarToYen` というインスタンスが追加され、続いてプロセス番号が表示されます。
- 「格納されたオブジェクト」区画では、Bean インスタンスがスタックの最上部に追加されます。
- 「結果」区画では、パラメータなしで生成メソッドの起動に伴う Bean インスタンスが表示されます。

次に `dollarToYen` Bean のビジネスメソッドをテストします。

例題 Bean のビジネスメソッドのテスト

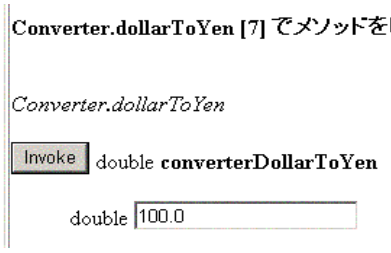
`Converter.dollarToYen` のインスタンスが US ドルを正しく日本円に変換するかどうかを検査するために、ビジネスメソッドを次のように呼び出します。

1. 「EJB ナビゲーション」区画で (ホームインターフェース名の下の) Bean 名をクリックします。

   「結果」区画の内容が消去されて、最下部区画の起動できるメソッドのリストの最上部に Bean のビジネスメソッドが表示されます。

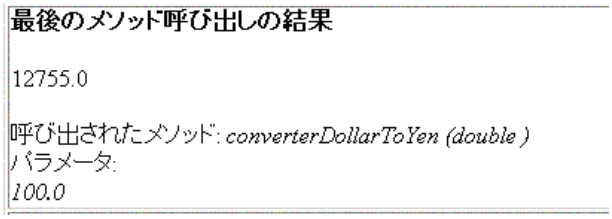
2. 最下部の区画に表示されているビジネスメソッドの下の入力フィールドにパラメータを入力し、「Invoke」ボタンをクリックします。

   次に示しているように、この例では `100` を入力しています (続けて IDE が `.0` を追加しています。)



「Invoke」ボタンをクリックすると、ページの区画の内容が次のように変わります。

- 「結果」区画には、次に示すようにメソッドの起動結果が表示されます (この例では 1 ドル 127.55 円の計算式がビジネスメソッドに組み込まれているので、結果は 12755 円となっています)。続けてテスト用に入力したパラメータ (100 ドル) が表示されます。

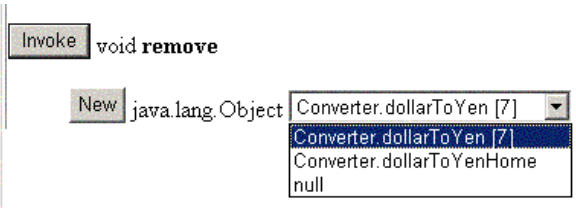


- 「格納されたオブジェクト」区画のオブジェクトスタックには、結果オブジェクトとパラメータオブジェクトが表示されます。

このほかに、最下部の区画に表示されている Bean メソッドはどれを呼び出すこともできます。

テスト用クラスの作成

Bean のその他のメソッドを呼び出すことで作成されたオブジェクトは、その Bean を更にテストするために使用することができます。たとえば、`Converter.dollarToYen` をテストしてそれまでに作成されたオブジェクトは、次の例に示すようにコンボボックスにも表示されます。これらのオブジェクトのどれかを選択して、テスト用に新しいクラスを作成することができます。



配備後の変更

変更を加えた エンタープライズ Bean を、新しくテストアプリケーションを生成することなく再テストすることができます。ただし、コンポーネントが変更された Bean を再テストする前に、テストアプリケーションを再配備する必要があります。再配備を実行するには、テストウィンドウを一度閉じます。その後、「エクスプローラ」ウィンドウでテストアプリケーションモジュールを右クリックし、「実行」を選択します。同じテストセッション中に Bean を再配備しないでください。

注意 - IDE は、エンタープライズ Bean をテストするためだけに使用される EJB モジュール、Web モジュール、および J2EE アプリケーションを生成します。これらの生成されたオブジェクトは変更しないでください。変更してしまうと、その J2EE アプリケーションを再び配備できなくなることがあります。

異なる方法でのテスト準備

201 ページの「テストの前提条件」で記述したように、テストの対象がリモートインタフェースがすでに指定されている単純なセッション Bean とは異なる場合、次に説明するような準備が必要になることもあります。

CMP または BMP Bean のテスト

エンティティ Bean は以降のどれかの方法でテストすることができます。

- テスト中に自動的に作成された EJB モジュール中で Bean をテストします。この Bean をテストする前には、データソース関連のプロパティを設定します。
- テスト機能を使用しないで Bean の EJB モジュールを作成し (185 ページの「EJB モジュールへの エンタープライズ Bean の格納」を参照してください)、プロパティはすべてそこで記述します。その後、Bean をテストする前に、作成しておいた EJB モジュールをテスト機能が自動生成したモジュールで置き換えます。

どちらの方法を取った場合でも、エンティティ Bean をテストする際には、Bean のプロパティが記述された EJB モジュールを使用する必要があります。テスト機能で生成された EJB モジュールはテスト専用なので、エンティティ Bean の実際の利用には実用の EJB モジュールを作成する必要があります。このモジュールは、テストの前でも後でも作成できます。

EJB 参照を持つ Bean のテスト

他の Bean と対話的に実行される エンタープライズ Bean をテストする場合は (エンティティ Bean が実行する処理を管理するセッション Bean のテストなど)、必ず参照先の Bean も EJB モジュールに含めておいてください。

参照のある Bean は、次の方法でテストできます。

- **既存の EJB モジュールの利用** - このモジュールには必要なプロパティすべてを指定しておきます。テストに既存の EJB モジュールを使用したい場合は、「新規 EJB テストアプリケーションを作成」ウィザードの「変更」ボタンを使用して既存の EJB モジュールを指定します。
- **参照元・参照先の Bean を含んだモジュールの作成** - 参照元・参照先の Bean を含んだモジュールを作成するには、最初にテストしたい Bean のテストアプリケーションを作成します。その後、「エクスプローラ」ウィンドウで IDE が作成した EJB モジュールを見つけます。EJB モジュールのノードを右クリックし、「EJB を追加」を選択します。

テストアプリケーションを生成する前に、必要なすべての EJB 参照を Bean のプロパティシートで必ず指定しておいてください。また、それに優先する参照がある場合は、 EJB モジュールのプロパティシートで指定しておきます。これらを怠ると、テストが失敗したり、結果が一定しません。

詳細は、 176 ページの「EJB 参照の指定」を参照してください。

Bean へのリモートインタフェースの追加

テスト中の Bean にローカルインタフェースだけがある場合、ローカルインタフェースをコピーしてテスト用のリモートインタフェースを作成できます。既存のインタフェースを利用する方法の概要を次に示します。

- ローカルインタフェース (`Local<`*Bean* 名`>`) を使用して対応するリモートインタフェースを作成します (`<`*Bean* 名`>`) 。

- ローカルホームインタフェース (Local<*Bean* 名>Home) を使用して対応するホームインタフェースを作成します (<*Bean* 名>Home)。

注意 - この方法を使って、現在のバージョンの Enterprise JavaBeans Specification に準拠していない EJB 1.1 CMP エンティティ Bean に EJB 2.0 機能 (ローカルインタフェースや参照など) を追加しないでください。変更された Bean は無効となり、修正ができません。詳細は、236 ページの「EJB 1.1 Bean での使用を避けるべき新規機能」を参照してください。

リモートインタフェースは 1 度に 1 つずつ追加します (この手順はローカルインタフェースから始めて、対応するリモートインタフェースを追加します。後に同じ手順を、ローカルホームインタフェースとホームインタフェースを追加するために繰り返します)。

次の操作を実行します。

1. 「エクスプローラ」ウィンドウで Local<Bean 名> というインタフェースのノードを右クリックし、メニューから「コピー」を選択します。
2. 該当する Bean の Java パッケージを選択し、右クリックして「ペースト」->「コピー」を選択します。

   インタフェースのコピーが Local<*Bean* 名>_1 としてフォルダに表示されます。
3. コピーを選択し、右クリックして「名前を変更」を選択します。「名前を変更」ダイアログでコピーしたインタフェースに J2EE の規定に従った名前を指定します (<*Bean* 名>)。
4. エクスプローラで Bean の論理ノードを右クリックし、「カスタマイズ」を選択します。

   「カスタマイザ」ダイアログが表示されます。

   「カスタマイザ」では、多くのプロパティを指定できます。プロパティは、Bean の論理ノードを右クリックしてから「プロパティ」を選択することで表示されるプロパティシートからでも指定できます。どちらのダイアログからでも Bean のプロパティを指定できますが、カスタマイザはその Bean のプロパティだけを処理し、アプリケーションサーバーに関連したプロパティは処理しません。また、Bean へのリモートインタフェースの追加は、カスタマイザを介してだけ実行できます。

5. 「カスタマイザ」ダイアログで空の「リモートインタフェース」フィールドを見つけ、「ブラウズ」ボタンをクリックします。

   図 9-3 に示すようなクラスファイル選択ダイアログが表示されます。

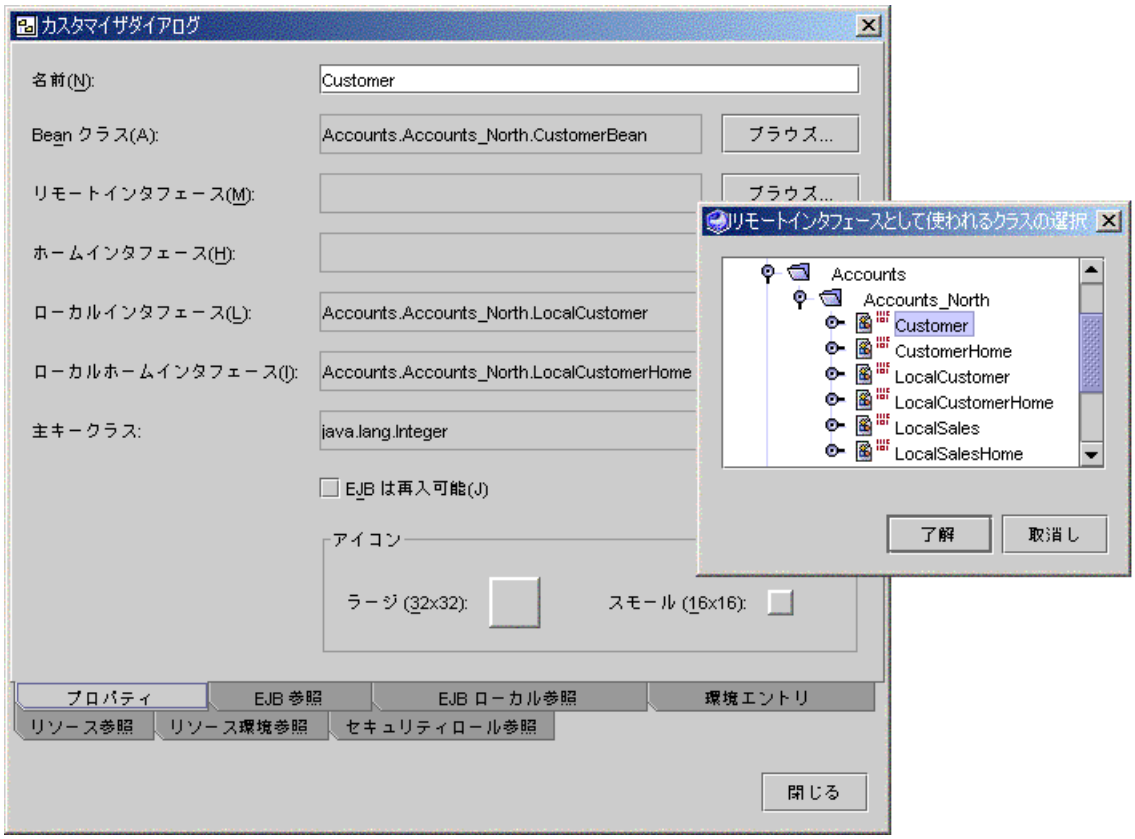


図 9-3　Bean へインタフェースクラスを追加するカスタマイザ

6. コピーとペーストで作成したインタフェースクラスを表示します。インタフェースノードを選択して「了解」をクリックします。

   「リモートインタフェース」フィールドにクラスが表示されます。

   (Bean のプロパティシートにある「リモートインタフェース」フィールドは、自動的に更新されます。)

7. 「カスタマイザ」ダイアログを消去します。

8. エクスプローラで Bean の論理ノードを展開します。クラスノードを展開し、新規リモートインタフェース (「リモートインターフェースクラス」と表示) を選択し、ソースエディタを開きます。

   リモートインタフェースクラスを編集して、javax.ejb.EJBObject のサブクラスとします。

9. リモートクラスの各メソッドに例外 java.rmi.RemoteException を追加します。

10. 各 ejbCreate メソッドの戻り値の型を、新しいリモートインタフェースに変更します。

11. 単一オブジェクトを検索する各検索メソッドの戻り値の型を、新しいリモートインタフェースに変更します。

12. エンタープライズ Bean をコンパイルして、エラーがあれば修正します。

13. 手順 1 から 手順 12 を繰り返して、ローカルホームインタフェースのコピーからホームインタフェースを作成します。

    ただし、手順 8 では、ホームインタフェースクラスを javax.ejb.EJBHome のサブクラスとしてください。

注 - リモートメソッドとして使用できないメソッドはすべて、上記の 2 つの新しいインタフェースから削除してください。

付録 A

エンタープライズ Bean の操作

エンタープライズ Bean の要素間の関係が、複雑でわかりにくくなる場合があります。Sun ONE Studio IDE は、特定の前提条件に基づいて Bean の整合性を保ちますが、Bean を再利用する多彩なオプションにも柔軟に対応しています。この付録では、既存のエンタープライズ Bean の推奨する操作方法を具体的に説明しています。

推奨する Bean の操作方法

エンタープライズ Bean への変更が意図したとおりに加えられるように、編集には Bean の論理ノードとプロパティシートを使用してください。 IDE を使用したこれらの推奨操作手順に従うことによって、J2EE specification 標準への準拠を確実にできます。

推奨する操作手順を次に説明します。

論理ノードを使用した作業

一般的に、エンタープライズ Bean のコードへの変更は、Bean の論理ノードを介して加えてください。論理ノードのアイコンはコーヒー豆で表されます 。このノードにエンタープライズ Bean のすべての要素がまとめられています。

論理ノードから操作を実行すると、IDE によって、Bean 全体に変更内容が適切に伝播されます。

Bean 中のすべてのクラスは、この論理ノードの下位に 1 つのオブジェクトとして表現されています。このオブジェクトを編集することで、どのクラスにどういった変更を加えるかということを考慮しないで変更操作が実行できます。次に例を挙げます。

- Bean の論理ノードの下位にある生成メソッドノードにメソッドを新しく加えると、メソッドの本体である `ejbCreate`*Xxx* とその関連メソッド `ejbPostCreate`*Xxx* (エンティティ Bean に必要です) が Bean クラスに追加されます。Bean が持つインタフェースの型によって、対応するメソッドシグニチャ (`create`*Xxx*) が、ローカルホームインタフェースかホームインタフェース、または両方のインタフェースに追加されます。

- Bean の論理ノードの下位にある検索メソッドノードにメソッドを新しく加えると、「新規検索メソッドを追加」ダイアログによってメソッドの正しい名前の入力が促されます。操作後、メソッドシグニチャが、ローカルホームインタフェースかホームインタフェース、または両方のインタフェースに追加されます。

  - CMP エンティティ Bean の場合、「新規検索メソッドを追加」ダイアログでは EJB QL の Select、From、および Where 文の指定も促されます。Bean をアプリケーションサーバーに配備すると、この EJB QL コードが自動的にそのサーバーで必要とされる SQL 文の一種に変換されます。

    EJB QL コードの準備がまだできていない段階で、新しい検索メソッドまたは選択メソッドを定義したい場合は、EJB コンパイラによる EJB QL の要求を無効にすることができます。224 ページの「エンタープライズ Bean のコンパイルと検証」を参照してください。

  - BMP エンティティ Bean では、対応する `ejbFind` メソッドも Bean クラスに追加されます。

- Bean の論理ノードの下位にあるビジネスメソッドノードにメソッドを新しく加えると、メソッドの本体が Bean クラスに追加されます。操作後、メソッドシグニチャが、ローカルホームインタフェースかホームインタフェース、または両方のインタフェースに追加されます。

- Bean の論理ノードの下位にあるホームメソッドノードにメソッドを新しく加えると、Bean クラスにメソッドの本体が追加されるほか、適切な 1 つまたは複数のインタフェースにシグニチャも追加されます。

- CMP エンティティ Bean の論理ノードの下位にある選択メソッドノードにメソッドを新しく加える場合、「新規選択メソッドを追加」ダイアログによって、EJB QL 文を含むメソッドの完成に必要なすべての情報の入力が促されます。メソッドの本体も Bean クラスに追加されます。

エンタープライズ Bean の論理ノード以外のノードから Java コードを編集した場合でも、IDE は通常、変更を Bean のクラスとインタフェースに伝播し、同期化します。ただ、論理ノードから変更を加えた場合に、結果として生成されるコードの整合性が最も高くなります。

カスタマイザまたはプロパティシートの利用

メソッドの名前や戻り値の型を変更したい場合、またはパラメータや例外を編集・追加したい場合は、「カスタマイザ」ダイアログかプロパティシートを利用することをお勧めします。論理ノードの下位でメソッドを選択し、右クリックしてから「カスタマイズ」または「プロパティ」を選択してください。

- 「カスタマイザ」はメソッドを作成したダイアログと同じ形式になっています。
- プロパティシートには、選択したメソッドが含まれるクラスに対応したインタフェースのタブが表示され、その中にメソッドの一部が表示されます。

上記のどちらかに対して加えた変更は、検証された上で正しい形式で正しいクラスに伝播されます。

メソッドを完成させるためにコードを追加する場合は、IDE のソースエディタを使用してください。

注意 - Bean クラスノードやインタフェースノードなど、論理ノード以外から変更を加えた場合でも、EJB ビルダーによって変更は伝播されます。ただし、状況に応じて、記述したコードがサンの J2EE 仕様 に準拠しているかどうか、手動での確認が必要になることがあります。これらの状況については、次で説明します。

ソースエディタを利用した Bean の編集

IDE のソースエディタだけを使用して エンタープライズ Bean の任意の部分を作成・変更することができます。ただし、IDE のウィザードとその他の GUI 機能を使用すれば、作業の省力化とコード内の整合性保持が実現でき、J2EE に準拠した エンタープライズ Bean を短期間で作成できます。

EJB ビルダーに備わっている機能を利用しないで作業した場合、結果にはばらつきが出ます。EJB ビルダーは、プログラマがあるクラスに加えた変更を他のクラスに適切に同期化させようとしますが、結果が意図したものとは異なったり、必要な変更が部

分的に反映されなかったりする場合もあります。したがって、クラスのコードに直接変更を加えた場合、結果として生成された エンタープライズ Bean には、手動で修正が必要なエラーが含まれる可能性があります。

次に例を挙げます。

- ソースエディタで Bean のインタフェースクラス (ホーム、リモート、ローカルホーム、またはローカル) を開いて、新しいメソッドのコードを追加した場合。

  有効なメソッドには、メソッド名と正しい戻り値の型が必要です。また、メソッドには throw 文によるシステム例外も記述されている必要があります。新しいメソッドが有効であれば、Bean クラスに自動的に追加されます。無効な場合は、メソッドは伝播されないでインタフェース内にそのまま残されます。

  後に、メソッドに問題が見つかって修正が必要なことがあります。EJB ビルダーは、指定したメソッドを Bean クラスに追加できないこともあります。この場合は、手動で追加する必要があります。手動による追加が終了するまでは、エクスプローラのメソッドノードに赤い X 印でエラーバッジが表示されます (223 ページの「IDE のエラー情報」を参照してください)。

  クラス間で変更が伝播される例を次に挙げます。

  - ホームインタフェースに生成メソッドを正しく追加した場合、EJB ビルダーは対応する `ejbCreate`*Xxx* メソッドを Bean クラスに自動的に追加します。
  - Bean クラスに正しく `ejbCreate`*Xxx* メソッドを追加した場合、EJB ビルダーは対応する生成メソッドを適切な 1 つまたは複数のインタフェースに自動的に追加します。

- ソーエエディタで Bean クラスを開いて、検索メソッドを追加した場合。

  コンパイラがコードを検証します。メソッドが有効だった場合は、正しい 1 つまたは複数のインタフェースに自動的に追加されます。

- ソースエディタで Bean クラスを開いて、ビジネスメソッドを追加した場合。

  コンパイラがコードを可能な範囲で検証します。ただし、リモートインタフェースまたはローカルインタフェースには伝播されません。これは、追加したメソッドが Bean クラス内での処理を助けるものであったり、ユーティリティとして使用するものだったりする可能性があるためです。

- Bean のリモートインタフェースまたはローカルインタフェースに、正しい例外を持つビジネスメソッドを追加した場合。

  メソッドは自動的に Bean クラスに伝播されます。

- ソースエディタを使用して、Bean のホームインタフェース (またはローカルホームインタフェース) 中の生成メソッド、または Bean のリモートインタフェース (またはローカルインタフェース) 中のビジネスメソッドを変更した場合。

  EJB ビルダーが自動的に変更を Bean クラスに伝播します。

- Bean クラスの `ejbCreate`*Xxx* メソッドを直接変更した場合。

  コンパイラがコードを可能な範囲で検証しますが、ホームインタフェースまたはローカルホームインタフェースには変更は伝播されません (こういった Java インタフェースクラス間の関連性は IDE すべてで同じように扱われます)。

- Bean のホームインタフェースのメソッドの変更中に、誤って必要な例外を削除してメソッドを無効にしてしまった場合。

  コンパイラがコードを検証し、エラー情報を出力します。Bean クラスに変更は伝播されません。

- Bean クラスに新しい生成メソッドを追加し、`ejbCreate` または `ejbCreate`*Xxx* 以外の名前を指定した場合。ホームインタフェースまたはローカルホームインタフェースに新しい検索メソッドを追加して `findBy`*Xxx* 以外の名前を指定した場合 (または、どちらかのメソッドを変更した場合)。

  コンパイラによって、宣言の構文が正しいかどうか、および戻り値の型とパラメータ型がきちんと解決できる有効な Java クラスであるかどうかが検証されます。

注意 - ソースエディタでクラスを編集する場合、変更は自動保存されません。必ず保存操作を実行してください。

IDE のエラー情報

J2EE 仕様に準拠していないコードを作成した場合、エクスプローラのノードアイコンの部分に警告バッジまたはエラーバッジが表示されます。問題についての解説を参照したい場合は、エラーの表示されているノードを選択し、右クリックしてから「エラー情報」または「EJB の検査」を選択します。

論理ノードの部分に黄色い三角形で示された警告バッジは、この Bean または Bean 中のメソッドの有効性に問題があることを示しています。論理ノードを展開して、問題のある箇所を見つけてください。たとえば、リモートインタフェースで定義

されたメソッドが Bean クラスになかったり、クラスが誤った Java スーパークラスのサブクラスになっていることがあります。Bean をコンパイルできたとしても、こういった問題は発生します。

論理ノードの部分に赤い X 印で示されたエラーバッジは、この Bean または Bean 中のメソッドに深刻な問題があることを示しています。たとえば、クラス全体がない、といった問題です。エラーバッジが表示されている Bean は、実行できなかったり、対話できなかったりします。

エンタープライズ Bean のコンパイルと検証

EJB ビルダーには、作成された エンタープライズ Bean を Enterprise JavaBeans Specification に照らし合わせて検証するカスタムコンパイラが含まれます。コンパイルと検証を別々に実行することもできます。エンタープライズ Bean を選択し、右クリックしてから「EJB の検査」を選択すると、デフォルトの動作として、Bean がコンパイルされ、続けて検証が実行されます。

コンパイルだけではエラーを発見できません。また、コンパイルエラーが見つかった場合は、検証は実行されていません。

IDE の検証とコンパイルには別々の目的があります。エンタープライズ Bean をコンパイルすると、Bean を構成している各種のクラスがコンパイルされます。これらのクラスの Java コードが構文的に正しければ、たとえクラス間に不整合があったり、J2EE の仕様に従っていないコードが含まれていたりしたとしても、コンパイルエラーは発生しません。エンタープライズ Bean 内の要素間の整合性を確実にするには、Bean の検証が欠かせません。

EJB コンパイラは、IDE の「構築」、「コンパイル」および「生成物を削除」の定義に一致しています。

- Bean の有効期限が切れていた場合 (つまり、Bean の配備記述子が変更された、または直接参照される Java クラスが最後に成功したコンパイル以降に変更された場合)、コンパイルを実行すると Java クラスは Enterprise JavaBeans Specification に従って、コンパイル・検証されます。
- 「生成物を削除」を実行すると、直接参照される Java クラスファイルと最後の EJB コンパイルの実行時のタイムスタンプを削除します。

- 「構築」は、最初に「生成物を削除」を実行し、次に参照されるクラスをコンパイルします。

ほとんどの場合、EJB コンパイラとそのデフォルトのオプションを対話的に使用してエンタープライズ Bean を開発することで、大幅な時間を節約できます。しかし、状況によっては (正常に終了したコンパイル後に、スーパークラスの変更が必要だったりするなど)、コンパイルによって Java や EJB の検証のエラーが見つからないこともあります。こういった場合は、(Bean の Java ファイルが保存されている場所にもよりますが、場合に応じて複数のディレクトリにわたって) 「構築」を実行して、コンパイルのエラーを検出・解決する必要があります。

コンパイラは、CMP エンティティ Bean 中にある選択メソッドおよび検索メソッドの EJB QL の構文チェックは実行しません。前述したように、コンパイラによる EJB QL 文の要求は無効にすることができます。エンタープライズ Bean の開発、コンパイル、および検証中に、見つからない EJB QL 文に関するエラーメッセージを表示させたくない場合は、この要求を無効にしておくと便利です。ただし、IDE で提供されている J2EE RI アプリケーションサーバーに Bean を配備する以前に、(コンパイラの EJB QL 要求の機能を有効に戻して、) EJB QL 文を追加する必要があります。このサーバーと他の種類のサーバーには EJB QL が必要です。これらのサーバーでは、サーバープラグインによって、EJB QL が、検索メソッドおよび選択メソッドに記述された照会を実行するのに必要な SQL に変換されます。

エンタープライズ Bean にコンパイルと検証のオプションを設定するには、次の操作を実行します。

1. メインメニューから「ツール」->「オプション」を選択します。
2. 「オプション」ノード、「構築」ノード、および「コンパイラの種類」ノードを展開します。
3. 「EJB 2.0 コンパイラ」を選択します。
4. プロパティシートで、次の項目を選択します。

   両方のフィールドのデフォルトの設定は `True` です。次の場合には設定を変更します。

   - Bean をコンパイルする前に、独立した手順で検証を実行したい場合は、「コンパイルが必要」フィールドを `False` に変更します。

- 検索メソッドと選択メソッドに EJB QL 文を指定しないで Bean の検証を済ませたい場合は、「EJB QL が必要」フィールドを `False` に変更します。

コンパイルから独立した検証を行いたい場合、Bean は次のように検証してください。

- **エクスプローラの「ファイルシステム」区画で、Bean の論理ノードを選択し、右クリックしてから「EJB の検査」を選択します。**

  Bean のサイズにもよりますが、検証には時間がかかることがあります。検証が終わると、出力ウィンドウが開いて Bean に関するメッセージが表示されます。

  EJB モジュールをコンパイルする場合、またはコンパイルを含む任意の操作をモジュールに対して実行する場合 (EJB JAR ファイルをエクスポートしたり、エクスポート用にクラスファイルを計算するなど)、モジュールとそのコンポーネント Bean すべては自動的に検証されます。

変更の保存

ほとんどの場合、作業内容は自動的に保存されます。ただし、されないこともあります。たとえば、Bean のコンパイルでは保存されないことがあります。IDE を終了すると、確認ダイアログが表示され、それまでの作業を保存するかどうかが確認されます。保存を確実にするには、IDE での作業中に「ファイル」->「すべてを保存」を定期的に指定してください。

エンタープライズ Bean 名の変更

Bean 名を変更するために、Bean のすべての関連オブジェクト名と、その内部の参照名を手作業で変更する必要はありません。次の手順で IDE の GUI 機能を使用すると、すべてのインタフェース (その外部オブジェクト名と該当する参照の両方) が自動的に同期化されます。次の操作を実行します。

1. **Bean の論理ノードを選択して右クリックし、「名前を変更」を選択します。**

   「名前の変更」ダイアログが表示されます。「新しい名前」フィールドに文字を入力すると、チェックボックスオプションが有効になります。

2. チェックボックスを使用して、Bean のすべての関連オブジェクト名を一度に変更します。

   ただし、外部から取得したオブジェクトがある場合は、それらのオブジェクト名も変更してよいかどうかを慎重に検討してください。たとえば、ホームインタフェースとリモートインタフェースが複数のエンティティ Bean によって共有されている場合は、それらのインタフェースを使用するすべての Bean で、インタフェース名を同じにした方が便利かもしれません。

注意 - 関連オブジェクト名を個別に変更すると、オブジェクト間の結び付きが失われてしまう可能性があります。

ほかの Bean から取り込んだクラスの修正

エンタープライズ Bean のクラスとして、ほかの Bean のクラスを使用することができます。たとえば、ほかの Bean のリモートインタフェースを使用する エンタープライズ Bean を作成することができます。このようなほかの Bean から取り込んだクラスを修正すると、実際には元のクラスが修正されます。最新の Bean は元の Bean のクラスを指しているため、クラスファイルは 1 つしかありません。IDE がこのような設計になっているのは、エンタープライズ Bean の要素を簡単に再利用できるようにするためです。

エンタープライズ Bean のコピーとペースト

Bean をコピーし、別のパッケージにペーストすると、ペースト先のパッケージには元のパッケージのクラスとインタフェースを指したノードが作成されます。IDE では、すべての要素が同じパッケージに含まれていなくても構いません。これは、Bean の要素を柔軟に再利用できるようにするためです。したがって、Bean の複製を作成し、別のパッケージで使用したい場合は、Bean のすべてのクラスとインタフェースのパスを、新しいパッケージに合わせて変更する必要があります。

あるパッケージから エンタープライズ Bean の複製を作成し、別のパッケージにペーストするには、次の操作を実行します。

1. エクスプローラの「ファイルシステム」区画から、コピー元の Bean の論理ノードを右クリックし、「コピー」を選択します。

2. Bean のコピー先のパッケージを右クリックし、「ペースト」 >「コピー」を選択します。

3. コピー元の Bean の論理ノードの下位にある「クラス」ノードを展開し、クラスまたはインタフェースのノードを右クリックして「コピー」を選択します。次に、コピー先のパッケージを右クリックし、「ペースト」->「コピー」を選択して、クラスまたはインタフェースをコピー先のパッケージにペーストします。

   Bean のクラスとインタフェースごとに、この手順を繰り返します。

4. コピーした Bean の論理ノードを右クリックし、「プロパティ」を選択します。

5. プロパティシートで、各クラスをコピー先のパッケージに合わせて変更します。次のフィールドを見つけてください。

   - Bean クラス
   - ホームインタフェース (ホームインタフェースがある場合)
   - リモートインタフェース (リモートインタフェースがある場合)
   - ローカルホームインタフェース (ローカルホームインタフェースがある場合)
   - ローカルインタフェース (ローカルインタフェースがある場合)
   - 主キークラス (主キークラスがある場合)

   a. 各フィールドをクリックしてから省略記号 (...) ボタンをクリックします。

   b. プロパティエディタでコピー先のパッケージに移動し、手順 3 でペーストしたクラスやインタフェースの複製を選択し、「了解」をクリックします。

      プロパティの値が *PackageName.classname* に変更されます。

   これで、それぞれのクラスやインタフェースのプロパティ値が、コピー先の *PackageName* の後ろに元の *classname* (または元のインタフェース名) を付け加えたものになります。コピー先のパッケージには、Bean の完全な複製が含まれています。

Bean のクラスやインタフェースの交換

作成済みの エンタープライズ Bean を変更し、それまでの要素の代わりに、ほかの Bean の要素 (ホームインタフェース、リモートインタフェースなど) を使用することができます。そのためには、Bean のプロパティシートを次の手順で使用します。

1. エクスプローラウィンドウから、クラスやインタフェースを変更する必要のある Bean を選択して右クリックし、「プロパティ」を選択します。
2. 「プロパティ」タブで、適切なプロパティ (Bean クラス、ホームインタフェース、主キークラス、またはリモートインタフェース) をクリックし、省略記号 (...) ボタンをクリックします。
3. 使用したいクラスやインタフェースを選択し、「了解」をクリックします。

   プロパティフィールドに、新しく選択したクラスやインタフェースの完全修飾パス名が表示されます。これらの要素は複製ではなく、元のクラスやインタフェースを指しているだけです。

Bean のメソッドの編集

エクスプローラウィンドウの GUI 機能を使用して追加したメソッドは、ソースエディタで編集することができます。Bean クラスのメソッドの本体を完成させるだけでよく、編集内容がほかのクラスやインタフェースに影響しない場合は、ソースエディタを使用してください。ただし、編集内容がほかのクラスに影響する場合は、Bean の関連オブジェクトに同期をとって変更を反映させる必要があります。この例については、221 ページの「ソースエディタを利用した Bean の編集」を参照してください。

代わりに、次のように「カスタマイザ」ダイアログを使用することもできます。

1. エクスプローラウィンドウで、Bean の論理ノードを展開し、編集したいメソッドまで移動します。
2. そのメソッドを選択して右クリックし、「カスタマイズ」を選択します。

   「カスタマイザ」ダイアログが開き、新規メソッドダイアログと同じフィールドが表示されます。
3. フィールドを必要に応じて編集し、「閉じる」をクリックします。

   IDE によって変更内容が Bean 全体に反映されます。

メソッドの表示

エンタープライズ Bean に作成したメソッドを参照するには、Bean の論理ノードを展開し、参照したいメソッドのあるサブノードまで移動します。メソッドのノードを右クリックしたら「開く」を選択します。ソースエディタにメソッドのコードが開かれます。

エンティティ Bean のフィールドの変更

エンティティ Bean でコンテナ管理による持続性と Bean 管理による持続性のどちらが使用されているかによって、フィールドの名前や型の変更方法が異なります。

フィールド名の変更

CMP Bean では、エクスプローラウィンドウの GUI 機能を使用します。次の操作を実行します。

1. Bean の論理ノードを展開し、CMP フィールドを選択して右クリックし、「名前を変更」を選択します。
2. チェックボックスを使用して変更範囲を指定します。

BMP Bean では、ソースエディタを使用して持続フィールドや非持続フィールドの名前を変更します。

フィールドの型の変更

CMP Bean では、エクスプローラの GUI 機能を使用します。フィールドの型を変更するには、次の操作を実行します。

1. Bean の論理ノードを展開し、CMP フィールドを選択して右クリックし、「カスタマイズ」を選択します。
2. 「カスタマイザ」ダイアログから別の型を選択します。

BMP Bean では、ソースエディタを使用して持続フィールドや非持続フィールドの型を変更します。

エンタープライズ Bean の削除

どの種類のエンタープライズ Bean についても、Bean を削除する方法は次の 1 つだけです。

1. Bean の論理ノードを選択して右クリックし、「削除」を選択します。

   「EJB 削除の確認」ダイアログが表示されます。

2. チェックボックスを使用して、Bean のすべての関連オブジェクトを同時に削除するかどうかを指定します。

   ほかの Bean で使用されているオブジェクトがある場合は、それらのオブジェクトを削除してもよいかどうかを慎重に検討してください。たとえば、同じ主キークラスを複数のエンティティ Bean で使用している場合は、そのクラスのチェックボックスの選択を解除し、Bean の残りのクラスだけを削除してください。

注意 - メニューバーから「編集」 > 「削除」を選択する方法で、Bean を削除しないでください。選択したクラスが単純に削除され、Bean を構成しているクラス間の整合性が失われてしまう可能性があります。

付録 B

EJB 1.1 エンタープライズ Bean の移行とアップグレード

Sun ONE Studio IDE の旧バージョンを使用して エンタープライズ Bean を作成した場合、つまり、エンタープライズ Bean が Enterprise JavaBeans Specification version 1.1 (EJB 1.1) に基づいて作成された場合、その Bean を現在のバージョンの IDE と Enterprise JavaBeans Specification version 2.0 (EJB 2.0) に移行させることができます。EJB 1.1 エンタープライズ Bean の種類と機能によっては、IDE を使用して Bean を自動的に変換することもできます。ここでは、手作業による変更を加える方法について説明します。

現在のバージョンでの変更点

現在と旧バージョンの IDE では、CMP (コンテナ管理による持続性) エンティティ Bean、そのプロパティ、および エンタープライズ Bean の検査一般に関して違いがあります。次に主な項目を挙げます。

- **大部分の Bean 変換の自動化** - 現在のバージョンの IDE に EJB 1.1 エンタープライズ Bean をインポートすると、ほとんどの場合、IDE がそれらの Bean を EJB 2.0 エンタープライズ Bean に変換します。ただし、 EJB 1.1 環境で作成された CMP エンティティ Bean は変換されません。

  これらの Bean はバージョンタグによって識別できます。これは、現バージョンの IDE に新しく導入された機能です。CMP エンティティ Bean のプロパティシートにある「CMP バージョン」フィールドを参照すると、現バージョンの IDE で作成された Bean には `2.x` と表示されますが、旧バージョンの CMP エンティティ Bean では `1.x` と表示されます (以降、この Bean を CMP 1.x エンティティ Bean と呼びます)。

- **EJB 検査機能の拡充** - 現バージョンの IDE では検査機能が充実されました。新しい検査機能では、以前の検査では見逃されていた Java コードの不良が見つかるようになりました。次の項目で挙げている `transient` 修飾子などがその例です。

  「EJB の検査」(IDE のエクスプローラに表示された エンタープライズ Bean の「コンテキスト」メニューから選択) または「すべてを構築」(パッケージの「コンテキスト」メニューから選択) を使用して、その Bean を含んでいるアプリケーションが現バージョンの IDE に配備できるかどうかを調べることができます。検査によってコード中にエラーが見つかった場合は、次の節を参照してください。

  Bean の検査結果に問題がなかった場合は、モジュールからアプリケーションへとアセンブルし、配備を試みます。配備に失敗した場合は、Bean のプロパティシートからの変更が問題を起こしていないかどうか調べます。

- **セッション Bean での `transient` の廃止** - 旧バージョンの IDE では `transient` 修飾子の指定は許容されていましたが、現バージョンの検査機能でこの修飾子が見つかるとエラーメッセージが出力されます。修飾子の検索と変更については、237 ページの「transient 修飾子の禁止」を参照してください。

- **旧 CMP エンティティ Bean が使用できる範囲** - Bean の `transient` 修飾子を変更していれば、現バージョンの IDE を使った開発段階で CMP 1.x エンティティ Bean を使用し続けるのに問題はありません。ただし、CMP 1.x エンティティ Bean を現バージョンの IDE で配備する際には、次の手作業による変更が必要となります。

  - **Bean の RI 関連プロパティの EJB モジュールでの指定** - 現バージョンでの大きな変更は、J2EE RI アプリケーションサーバーに配備される CMP エンティティ Bean に対するデータソース参照の宣言方法です。これらの CMP エンティティ Bean に対する RI 関連のプロパティは、現バージョンでは EJB モジュールで指定します。詳細については、238 ページの「EJB モジュールへの Bean の RI プロパティの移動」を参照してください。

  - **生成された SQL の RI 配備時の変更** - CMP 1.x エンティティ Bean を RI に配備する場合は、RI プラグインが自動生成した SQL 文の 1 行を変更する必要があります。具体的には、検索メソッドの SELECT の宣言にある `WHERE` 文の編集が必要です。詳細は192 ページの「EJB 1.1 CMP エンティティ Bean を含んだ EJB モジュール」を参照してください。

  これらの変更を加えても、CMP エンティティ Bean の CMP バージョンの値は 1.x のままで、CMP 2.x には変換されていません。しかし、EJB 2.0 に準拠しているアプリケーションサーバーにこれらの Bean を配備する場合、Bean とそのインタフェースは正しく動作し、現在の IDE でも問題なく配備できるはずです。

- **新機能追加前の旧 CMP エンティティ Bean の変換** - CMP 1.x エンティティ Bean には、ローカルインタフェースなどの EJB 2.0 の機能を追加することはできません。旧 Bean に新しい機能を追加したい場合は、該当する Bean を手作業で CMP 2.x に変換する必要があります。この操作手順については、235 ページの「CMP 1.x エンティティ Bean の変換」で説明しています。

変更操作

以降の節では、以前のバージョンの IDE を使用して作成された エンタープライズ Bean を更新する方法について説明します。

CMP 1.x エンティティ Bean の変換

CMP 1.x エンティティ Bean と同じ機能を持つ CMP 2.x Bean を最初から作り直すことは無理だが、ローカルインタフェース、ローカル参照、選択メソッドやホームメソッドといった新しい EJB 2.0 機能を利用したいといった場合は、手作業による CMP 1.x エンティティ Bean のアップグレードが必要です。CMP 1.x エンティティ Bean を CMP 2.x Bean に変換するには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで新規 Java パッケージを作成します。
2. CMP 1.x エンティティ Bean の旧パッケージから Java ファイルをコピーします。それらのファイルを新しいパッケージに (リンクではなく複製として) ペーストします。
3. EJB ビルダーウィザードを使用して、新規 CMP エンティティ Bean を作成します (第 4 章を参照)。

   ウィザードの最後の区画で、コピーしたクラスを Bean のリモートインタフェースとホームインタフェースとして指定します。更に、Bean クラスと主キークラスも指定します。

   この時点では、CMP (持続) フィールドが見つからないという IDE の警告は無視してください。
4. 必要なフィールドを追加して、EJB 検査によるエラーがあれば修正してください。

   他の種類の EJB 1.1 エンタープライズ Bean を現バージョンの IDE にインポートした場合は、EJB 2.0 に準拠するように自動的にアップデートされます。

EJB 1.1 Bean での使用を避けるべき新規機能

以前のバージョンの IDE で作成したエンタープライズ Bean に対して新規機能の追加を試みると、予期しない結果になることがあります。次に CMP 1.x エンティティ Bean の例を 2 つ挙げます。

CMP 1.x エンティティ Bean へのローカルインタフェースの追加禁止

EJB 1.1 Bean では「コンテキスト」メニュー (「エクスプローラ」ウィンドウで Bean の論理ノードを右クリックすると表示されるメニュー) が異なります。また、メニューのオプションの数も限られています。ただ、「コンテキスト」メニューから「カスタマイズ」を選択すると、ローカルホームインタフェースとローカルインタフェースファイルが指定できるかのように見えるウィンドウが表示されます。

これらのフィールドは直接編集できません。また、EJB 1.1 CMP エンティティ Bean のプロパティシートにローカルインタフェースへのパスが表示されることもありません。

注意 - ローカルインタフェースをこの方法で追加しないでください。この方法で追加したインタフェースは削除できません。また、CMP エンティティ Bean も無効となり、修正できなくなります。

CMP 1.x エンティティ Bean へのローカル EJB 参照の追加禁止

ローカル EJB 参照についても同じ注意が必要です。CMP 1.x エンティティ Bean にはローカル EJB 参照を追加しないでください。「カスタマイザ」ウィンドウが表示されて追加できるように見えますが、実行しないで下さい。ローカル EJB 参照を追加してしまった場合は、再び「カスタマイザ」ウィンドウから追加した参照を削除してください。

EJB 2.0 標準は大規模な規定であり、新しいコーディング規則は現バージョンの IDE で強く推奨または施行されています。できれば、旧 エンタープライズ Bean を現在の IDE の EJB ビルダーウィザードを使用して再作成することをお勧めします。変換する場合は、コードを詳しく調べて、現在の EJB 標準に準拠するように必要な箇所を更新してください。

注 - 必要であれば、CMP 1.x エンティティ Bean 以外の EJB 1.1 エンタープライズ Bean にはローカルインタフェースとローカル EJB 参照を追加することができます。これは IDE がこれらの Bean を自動的に現在の EJB 標準に準拠するように変換するためです。これらの Enterpirse Bean の変更には「カスタマイズ」ダイアログとプロパティシートのどちらも使用できます。

PointBase ユーザー名とパスワードの変更

PointBase データベースに対するデフォルトのユーザー名とパスワードは、`PUBLIC` から `pbpublic` に変更されました。すべてのプロパティシートでこの変更を反映させてください。パスワードを入力した後には必ず Enter キーかリターンキーを押してください。

`transient` 修飾子の禁止

旧バージョンの IDE では、セッション Bean での次のような宣言文は許容されていました。

```
public class HelloBean implements javax.ejb.SessionBean {
private transient SessionContext context;
```

旧バージョンの検査機能では `transient` 修飾子は検出されませんでした。現バージョンでは、この修飾子は検出され、使用が許可されません。

これは、EJB 1.1 セッション Bean に対して実行される EJB 検査コマンドが単純で、この修飾子を検出できなかったためです。Bean が完全に検査されるように、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで Bean を選択します。

2. メインメニューから「構築」->「構築」を選択します。

   IDE によってすべての Bean の Java クラスが構築され、検査が実行されます。構築の結果が、エラーメッセージと共にコンパイラによって出力されます。

   セッション Bean に `transient` 修飾子がある場合は、Bean の構築時またはその Bean に対してクライアントが実行されたときにエラーが出力されます。問題を修正するには、EJB Bean クラスから `transient` という記述を削除してから再び Bean を検査するか構築し直してください。検査または構築が正常に終了したら、Bean を配備して再びクライアントを実行してください。

   Enterprise JavaBeans Specification Version 2.0 では次のように規定されています。

   > Bean 提供者は、`ejbPassivate` および `ejbActivate` 通知の間では transient フィールドの内容が失われると考える必要がある。したがって、Bean 提供者は次のオブジェクトに対する参照を `transient` フィールドに保存してはならない。`SessionContext` オブジェクト、環境 JNDI ネーミングコンテキストまたはそのサブコンテキスト、ホームインタフェースとリモートインタフェース、または、`UserTransaction` インタフェース。

EJB モジュールへの Bean の RI プロパティの移動

RI に配備される CMP エンティティ Bean では、RI 関連のプロパティを Bean レベルから EJB モジュールレベルに移動させる必要があります。

エンタープライズ Bean のプロパティシートは、やや変更されています。ある エンタープライズ Bean が他の Bean を呼び出したとき、参照は呼び出し元の Bean に設定されています。このようなアプリケーションを正しく実行できるようにするには、2 つの変更が必要です。この変更が加えられないと、「保存できません」という配備エラーが発生します。このエラーは、CMP エンティティ Bean の誤ったプロパティシートにデータベース関連の情報が記述されていることに起因している可能性があります。この問題を修正するには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで Bean の EJB モジュールノードを右クリックし、「プロパティ」を選択します。
2. 「プロパティ」ウィンドウの J2EE RI タブを表示します。

   旧バージョンの IDE での Bean の J2EE RI プロパティシートにあった 3 つのプロパティが表示されます。これらは、CMP エンティティ Bean が使用するデータベース接続を表しています。

3. データベースに対する各プロパティを指定します。

   PointBase データベースを使用している場合は、次の値を入力します。

   - 「データソースの JNDI 名」フィールドには `jdbc/Pointbase` と指定します。これはアプリケーションサーバー中での JNDI 名です。`Pointbase` は最初の 1 文字だけを大文字で指定します。
   - 「データソースのパスワード」フィールドには `pbpublic` と指定します。これは新しいデフォルトのパスワードです。パスワードの入力後には Enter キーかリターンキーを必ず押してください。
   - 「データソースのユーザー名」フィールドには `pbpublic` と指定します。これは新しいデフォルトのユーザー名です。

テスト前の CMP エンティティ Bean プロパティの変更

旧バージョンの IDE では、データベース接続に関して正しい情報が CMP エンティティ Bean に指定されていれば、その Bean から生成されたテストアプリケーションは追加の変更なく RI に配備できました。

現バージョンでも、セッション Bean または BMP (Bean 管理による持続性) エンティティ Bean の EJB テストアプリケーションには変更は必要ありません。

ただし、CMP エンティティ Bean が RI に配備されることになっていて、PointBase に対するテストが必要とされる場合には、その Bean の変更が必要です。前述のように、データソース JNDI 名、ユーザー名、およびパスワードのプロパティは、エンタープライズ Bean レベルから EJB モジュールレベルに移動されました。また、PointBase データベースのデフォルトのユーザー名とパスワードも変更されていて、EJB 参照の J2EE RI タブにある JNDI 名にはプロパティが 1 つだけになりました。これらに対して、CMP Bean プロパティシートは自動的には更新されません。

以上のプロパティを変更するには、次の操作を実行します。

1. 「エクスプローラ」ウィンドウで、テスト機能によって生成された EJB モジュールのノードを右クリックして「プロパティ」を選択します。
2. プロパティシートで「J2EE RI」タブをクリックします。

3. J2EE RI タブで 238 ページの「EJB モジュールへの Bean の RI プロパティの移動」で説明したように 3 つのデータソースを更新します。

   これらのプロパティを設定する前にアプリケーションを配備しようとすると、エラーメッセージが表示され、出力ウィンドウの「RI アプリケーションの配備」タブにエラーの詳細が表示されます。

索引

A

afterBegin メソッド
  ステートフル CMT セッション Bean, 33, 65, 73
afterCompletion メソッド
  ステートフル CMT セッション Bean, 33, 65, 73

B

Bean
  クラス, 14
    エンティティ Bean, 131
    セッション Bean, 62
    メッセージ駆動型 Bean, 157
  クラスとインタフェース, 8, 62, 95, 143
  クラスやインタフェースの変更, 228
  検証, 224 ~ 226
  コピーとペースト, 227 ~ 228
  種類、エンティティ Bean, 119
  セッション Bean の種類, 55
  プロパティ, 49, 170 ~ 184
    プロパティシート, 170 ~ 184
Bean 管理による持続性 (BMP), 34
  CMP との比較, 79
  生成したコードの完成, 145 ~ 149
Bean 管理によるトランザクション (BMT), 28, 57, 59, 153
Bean 提供者が行う必要のある作業
  エンティティ Bean のコーディング, 35
Bean への変更, 219 ~ 223
Bean ホーム名、抽象スキーマ名を参照
Bean メソッドの変更, 221
Bean メソッドの編集, 229
beforeCompletion メソッド
  ステートフル CMT セッション Bean, 33, 65, 73

C

CMP フィールド
  CMR, 126
  値の初期化, 37
  関係 CMP エンティティ Bean のセット, 126
  個別指定, 93 ~ 94
  単一 CMP エンティティ Bean, 86
  追加, 114
  データベースの列への対応付け, 86
  データベース表の列, 126
CMR
  基本性, 41
  指向性, 41
CMR (コンテナ管理による関連性)
  EJB モジュールによる管理, 132
  概要, 41
  関連 CMP エンティティ Bean, 127
  追加, 134 ~ 137
  編集, 127 ~ 129
commit メソッド, 72

E

EJB JAR ファイル, 5
EJB QL, 220
ejbActivate メソッド, 64, 103, 144
  エンティティ Bean のインスタンス, 36
  ステートフルセッション Bean での完成, 68 ~ 69
  ステートフルセッション Bean のインスタンス, 32
ejbCreate メソッド, 37, 147, 220
  BMP エンティティ Bean, 147 ~ 148
  CMP エンティティ Bean, 105 ~ 107
  ステートレスセッション Bean インスタンスのプールへの格納, 32
  セッション Bean, 30, 66 ~ 68
  メッセージ駆動型 Bean, 157
  メッセージ駆動型 Bean のメソッド, 45
ejbFin メソッド, 220
EJB JAR ファイル, 184 ~ 199
ejbLoad メソッド, 103, 145
  BMP エンティティ Beanのインスタンス, 36
  データストアとの同期, 40
ejbPassivate メソッド, 64, 103, 144
  エンティティ Bean のインスタンス, 36
  ステートフルセッション Bean のインスタンス, 32
ejbPassivate メソッドステートフルセッション Bean での完成, 68 ~ 69
ejbPostCreate メソッド, 37, 148, 220
  BMP エンティティ Bean, 148
  CMP エンティティ Bean, 105 ~ 107
  セッション Bean, 66 ~ 68
EJB QL
  エラー, 226
  外部キー, 41
  検索メソッド, 111
  コンパイル時の要否, 226
  選択メソッド, 114
  選択メソッド中の EJB QL, 10
  表の連結, 41
ejbRemove メソッド, 64, 103, 144, 158
  ステートレスセッション Bean インスタンスのプールへの格納, 32
  データベースエンティティの削除, 39
ejbStore メソッド, 40, 103, 145
  BMP エンティティ Bean のインスタンス, 37
EJB アプリケーションの構成, 47
EJB アプリケーションのワークフロー, 16
EJB グループ、「関係 CMP エンティティ Bean のセット」を参照
EJB コンテナ
  J2EE アプリケーションでの役割, 4
  エンティティ Bean インスタンスのプールへの格納, 35
  エンティティ Bean へのサービスの提供, 34
  ステートレスセッション Bean インスタンスのプールへの格納, 31
  トランザクションの管理, 27
  メッセージ駆動型 インスタンスのプールへの格納, 46
EJB 参照, 176
EJB ビルダーウィザード, 18, 54 ~ 74
  Bean クラスを介した変更の伝播, 219 ~ 223
  BMP エンティティ Bean の定義, 139 ~ 150
  CMP エンティティ Bean の定義, 77 ~ 114
  CMP エンティティ Bean クラスの生成, 122 ~ 130
  関係 CMP エンティティ Bean のセットの定義, 118 ~ 130
  セッション Bean のクラスの作成, 62 ~ 65
  セッション Bean の定義, 58 ~ 61
  メソッドシグニチャの生成, 30, 37, 45
  例外の作成, 48
EJB モジュール, 5
  作成, 184 ~ 199
  テスト用, 202
  トランザクション属性, 195 ~ 197
  プロパティ, 49
EJB モジュールへの Bean のアセンブル, 185 ~ 186
equals メソッド, 101

F

findByPrimaryKey メソッド, 38, 143

G

getCallerPrincipal メソッド, 50
getRollbackOnly メソッド, 72
getter および setter メソッド, 41
getUserTransaction メソッド, 72

H

hashCode メソッド, 101

I

IDE
  Bean の検証, 224
  BMP エンティティ Bean の完成, 145 ~ 149
  CMP エンティティ Bean の完成, 104 ~ 114, 132 ~ 137
  EJB コンパイラ, 224
  エクスプローラウィンドウ, 59, 83, 121, 140, 155
  エラー情報, 223
  推奨する操作方法, 219 ~ 223
  セッション Bean の完成, 66 ~ 74
  ソースエディタ, 221
  配備記述子の完成, 167 ~ 194
  変更の保存, 226
  メッセージ駆動型 Bean の完成, 158 ~ 160
IDE が対応しているクライアント, 2
IDE のエクスプローラウィンドウ, 59, 83, 140, 155
isCallerInRole メソッド, 50

J

J2EE アーキテクチャの内部の規約, 6
J2EE アーキテクチャの機能, 2
J2EE アーキテクチャの処理層, 2
J2EE アプリケーションアーキテクチャ, 2
J2EE アプリケーションモデルに含まれるデータ, 4
J2EE リファレンス実装 サーバー、「RI」を参照
JAR、「EJB JAR ファイル」を参照
java.io.Serializable インタフェース, 102
java.rmi.RemoteException, 48
java.rmi.Remote インタフェース, 102
java.security.Principal オブジェクト, 51
java.sql.Connection, 72
JavaBeans、エンタープライズ Bean との違い, 3
javax.ejb.CreateException, 48
javax.ejb.EJBContext, 72
javax.ejb.EJBException, 48
javax.ejb.EJBHome, 62
javax.ejb.EJBHome インタフェース, 95
javax.ejb.EJBObject, 62
javax.ejb.EJBObject インタフェース, 96
javax.ejb.EntityBean インタフェース, 95
javax.ejb.MessageDrivenBean インタフェース, 156
javax.ejb.MessageListener インタフェース, 156
javax.ejb.SessionBean, 62
javax.transaction.UserTransaction, 72
Java トランザクション API, 29
Java トランザクションサービス (JTS), 29
Java メッセージサービス (JMS), 42
JDBC API, 29, 34 ~ 35
  JTA コードと併用しない, 71
JDBC API を使用した従来のコードの組み込み, 35, 29
JNDI, 4, 174
JTA, 29, 71

N

newInstance メソッド
  エンティティ Bean, 36
  メッセージ駆動型 Bean のメソッド, 45

newInstance メソッド
セッション Bean, 30

O

onMessage メソッド, 10, 159

P

private フィールド, 146

R

RI, 51
CMP エンティティ Bean のプロパティ, 187 ~ 194
CMP フィールドの順序, 194
検索メソッド, 192 ~ 194
セッション Bean と BMP エンティティ Bean 用のプロパティ, 182
プロパティ宣言, 187 ~ 194
RI でのテスト, 201 ~ 214
RI を使用したテスト, 51
RI を使用したプロトタイプ, 51

S

setAutoCommit メソッド, 72
setEntityContext メソッド, 36, 144
setMessageDrivenContext メソッド, 158, 160
メッセージ駆動型 Bean のメソッド, 45
setRollBackOnly メソッド, 72
setSessionContext メソッド, 30, 64
SQL, 5, 34
EJB QL 文からの生成, 220
RI タブでのプロパティの設定, 192 ~ 194
SQL 挿入文, 147

U

unsetEntityContext メソッド, 37, 103, 145
UserTransaction (UT) メソッド, 72

X

XML 配備記述子ファイル, 49

あ

アクセサメソッド, 41
アプリケーション開発の役割, 6
アプリケーション、構成, 47
アプリケーションサーバー
EJB コンテナのサービス, 4, 34
RI での要件, 181 ~ 182, 187 ~ 194
プロパティシートのタブ, 170
要件, 61
アプリケーションのアセンブル、マニュアル「Building J2EE Applications 」を参照
アプリケーションの例
http://forte.sun.com/ffj/documentation/
アプリケーションレベルの問題, 48

い

一覧表
エンティティ Bean の種類, 79
入れ子になったトランザクション, 29
インスタンスプール
エンティティ Bean, 35
ステートレスセッション Bean, 31
メッセージ駆動型 Bean, 46

う

ウィザード、「EJB ビルダーウィザード」を参照

え

エクスプローラウィンドウ
  IDE, 121
エクスプローラでのパッケージ (フォルダ) ノード, 58, 81
エラー情報, 223
エンタープライズ Bean
  JavaBeans との違い, 3
  更新, 219 ~ 231
  削除, 231
  Bean の要素, 8
  EJB コンテナへの関連
  アプリケーションでの使用, 47
  開発ライフサイクル, 17
  クラス, 8
  最適化, 52
  持続性, 21
  セキュリティ, 21, 49, 180 ~ 181
  設計手法, 52
  テスト, 201 ~ 214
  トランザクション, 20
  パフォーマンス, 52
  メソッド, 8
  ワークフロー, 16
エンタープライズ Bean のメソッド, 8
  ホーム, 10
  afterBegin, 33
  afterCompletion, 33
  beforeCompletion, 33
  ejbActivate, 32
  ejbCreate, 30, 37, 45
  ejbLoad, 40
  ejbPassivate, 32, 36
  ejbPostCreate, 37
  ejbRemove, 32
  ejbStore, 37, 40
  findByPrimaryKey, 38
  getCallerPrincipal, 50
  hashCode, 101
  isCallerInRole, 50
  newInstance, 30, 36, 45
  onMessage, 10
  setEntityContext, 36
  setMessageDrivenContext, 45
  setSessionContext, 30
  unsetEntityContext, 37
  検索メソッド, 9, 38
  実行権限, 50
  生成メソッド, 9, 37
  セキュリティ, 21
  選択, 10
  ハッシュコード, 101
  ビジネスメソッド, 10
  ライフサイクルメソッド, 10
エンティティ
  Bean のデータベースへのマッピング, 21
  context メソッド, 36
  セッション Bean による表現, 25
エンティティ Bean, 33
  Bean クラス, 131
  BMP クラスの生成, 140 ~ 141
  BMP のコードの完成, 145 ~ 149
  CMP エンティティ Bean のコードの完成, 104 ~ 114
  CMP クラスの生成, 83 ~ 94, 122 ~ 130
  CMP のコードの完成, 132 ~ 137
  EJB コンテナとの関係, 34
  findByPrimaryKey メソッド, 38
  インスタンスの検出, 38
  主キー, 38
  主キークラス, 95
  種類, 119
  使用可能状態, 36
  匿名インスタンス, 38
  フィールドの変更, 230
  プール状態, 36
  ホームインタフェース, 131, 95
  ライフサイクル, 35
  リモートインタフェース, 95, 131
エンティティ Bean のインスタンスの検出, 38

か

回避、メッセージ駆動型 Bean の問題, 165
外部依存性、「配備記述子」を参照

外部キー, 109
カスタマイザ
Bean へのインタフェースの追加, 215 ~ 218
メソッド、パラメータ、および例外の変更, 221
活性化
エンティティ Bean のインスタンス, 36
ステートフルセッション Bean のインスタンス, 32
環境
実行時用の情報、「配備記述子」を参照
プロパティシートの環境エントリ, 176 ~ 177
関係 CMP エンティティ Bean のセット, 117
関連オブジェクト
エンティティ Bean, 88
セッション Bean, 60

き

基本クラス、「Bean クラスとインタフェース」を参照
キュー, 160

く

クライアントと Bean との関係, 24, 33
クラスのファイル
エンティティ Bean のクラス, 88
繰り返しメッセージ, 165

け

検索メソッド, 9, 38, 99, 110 ~ 112, 149, 220
検証による整合性, 224 ~ 226

こ

コード、完成
メッセージ駆動型 Bean, 158 ~ 160
コードの完成
エンティティ Bean, 35, 104 ~ 114, 132 ~ 137
セッション Bean, 66 ~ 74, 29
コードの生成
エンティティ Bean でのメソッドシグニチャ, 37
エンティティ Bean のクラス, 78
セッション Bean のクラス, 54
例外, 48
固有の識別子、エンティティ Bean, 33
コンテナ、「EJB コンテナ」を参照
コンテナ管理による持続性 (CMP), 34, 79
生成されたコードの完成, 104
コンテナ管理によるトランザクション (CMT), 27, 57, 59, 153
コンパイラオプション
検証の要否, 225
EJB QLの要否, 226
コンパイルと検証の比較, 224

さ

サーバー、「アプリケーションサーバー」を参照
サーバーの停止、エンティティ Bean の状態の保持, 33
サービス
EJB コンテナからの提供, 4, 34
再利用
ウィザードでの設定, 61
実行時情報宣言を介した再利用, 167
宣言による実行時情報, 49
削除
エンティティ Bean のインスタンス, 37
セッション Bean のインスタンス, 31
メッセージ駆動型 Bean のインスタンス, 46
作成
新しいエンティティ, 37
エンタープライズ Bean を格納する EJB モジュール, 184
エンティティ Bean の新しいインスタンス, 37
エンティティ Bean のインスタンス, 35
セッション Bean のインスタンス, 30

テスト用オブジェクト, 204

し

システム例外, 72
システムレベルの問題, 48
事前定義例外, 48
持続性, 21
  BMP エンティティ Bean の完成, 146
  CMP エンティティ Bean に対する選択項目, 83 ~ 84
  EJB コンテナによる管理, 34
  ウィザードでの CMP エンティティ Bean に関する選択項目, 122 ~ 123
  プロパティの設定, 181 ~ 194
持続フィールド, 99
  個別指定, 93 ~ 94
実行時情報, 49
主キー, 38
  エンティティ Bean の主キーの新規作成, 108 ~ 109
  エンティティ Bean への主キーの追加, 107 ~ 109
主キークラス, 146
  エンティティ Bean 中の, 95
  必要なメソッド, 101 ~ 102
種類
  エンティティ Bean, 119
  セッション Bean, 55
順序、メッセージ, 165
順不同なメッセージ, 165
書体と記号について, xxi
使用可能状態、エンティティ Bean のインスタンス, 36
状態、複数のメソッドの呼び出しにわたる保持, 26
仕様への対応
  EJB 2.0, 1
  J2EE 2.0, 1
初期化
  持続フィールド, 37
  ステートフルセッション Bean の状態, 56
  メッセージ駆動型 Bean のインスタンス, 45
除去
  メモリーからメッセージ駆動型 Bean のインスタンスを取り除く, 46

す

スーパークラス, 61, 224
ステートフルセッション Bean, 26, 55
  ウィザードでの選択項目, 59
  非活性化と活性化, 32
ステートレスセッション Bean, 25, 55
  ウィザードでの選択項目, 59
スレッド、メッセージ駆動型 Bean による模擬実行, 42

せ

生成
  メッセージ駆動型 Bean の新規インスタンス, 45
生成コード
  CMP エンティティ Bean セットクラス, 118
  メッセージ駆動型 Bean のメソッドシグニチャ, 45
  配備記述子, 167
生成されたコード
  メッセージ駆動型 Bean クラス, 153
生成するコード
  エンティティ Bean のクラス, 78
生成メソッド, 9, 220
  エンティティ Bean, 105 ~ 107, 147
  セッション Bean, 63, 66 ~ 68
  データストアへのデータの挿入, 37
  メッセージ駆動型 Bean, 157
セキュリティ, 21, 49
  `getCallerPrincipal` メソッド, 50
  `isCallerInRole` メソッド, 50
  配備記述子でのセキュリティロール, 180 ~ 181
セキュリティの確認

エンティティ Bean, 37
セッション Bean, 30
メッセージ駆動型 Bean, 180
メッセージ駆動型 Bean での確認, 45
セキュリティのコーディング, 49
セキュリティの指定, 50
セッション Bean, 24, 66 ~ 68
Bean クラス, 62
エンティティの表現, 25
コードの完成, 66 ~ 74
種類, 55
ステートフルセッション Bean, 26
ステートレスセッション Bean, 25
セッションでの状態の同期化, 32
プールへの格納, 26
ホームインタフェース, 62
ライフサイクル, 29
リモートインタフェース, 62
セッション Bean インスタンスの初期化, 30
セッション同期化インタフェース, 32, 72 ~ 74
クラス, 65
接続ファクトリ
エンタープライズ Bean 一般, 178
メッセージ駆動型 Bean, 161
接続ファクトリへのリソース参照, 161
宣言
実行時情報, 49, 167 ~ 194
セキュリティ, 49, 180 ~ 181
トランザクション属性, 28, 195 ~ 197
選択メソッド, 10, 220

そ

送信先、メッセージ駆動型, 160
ソースエディタ, 221 ~ 223
属性、「トランザクション属性」 を参照

た

対応している仕様, 1

ち

違い
JTA と JDBC API, 28
エンタープライズ Bean と JavaBeans, 3
検索メソッドと選択メソッド, 111
コンテナ管理による持続性と Bean 管理による持続性, 34
コンテナ管理によるトランザクションと Bean 管理によるトランザクション, 27
ステートレスセッション Bean とステートフルセッション Bean, 25
セッション Bean とエンティティ Bean, 24, 33
ビジネスメソッドとホームメソッド, 112
抽象アクセサメソッド, 41
抽象スキーマ名, 99, 126, 136, 172
重複メッセージ, 165

つ

通信セッション, 24 ~ 33

て

データストアへのデータの挿入
生成メソッドの使用, 37
データストレージ接続, 177 ~ 180
データの同期, 40
データベースサーバー
IDE に含まれる, 51
データベーススキーマ
CMPフィールド生成時の使用, 130
CMP フィールドの生成, 89 ~ 90
収集, 88 ~ 89
データベースのマッピング, 21
CMP フィールドとの対応付け, 123 ~ 129
データベースの接続, 86 ~ 88
CMP エンティティ Bean の作成時, 120 ~ 121
CMP エンティティ Bean の生成, 82 ~ 83
プロパティシートでの指定, 177 ~ 180

と

同期化
- エンティティ Bean のインスタンスとデータストア, 40
- セッションでの状態, 32, 65

匿名インスタンス
- メッセージ駆動型 Bean, 45

トピック, 160

トランザクション, 20
- Bean 管理, 28, 57, 153
- JDBC API の使用, 29
- JTA の使用, 29
- 入れ子、JTA では使用不能, 29, 71
- エンティティ Bean でのトランザクション, 34
- メッセージ駆動型 Bean, 153
- コンテナ管理, 27, 57, 153
- セッション Bean でのトランザクション, 26, 57, 70 ~ 74
- 属性, 28
  - EJB モジュール, 195 ~ 197
  - 個々の Bean, 196
  - 個々のメソッド, 196
- 範囲, 71
- ロールバック, 71

トランザクション制御
- エンティティ Bean, 37
- セッション Bean, 30
- メッセージ駆動型 Bean での制御, 45

な

名前の変更
- Bean のフィールド, 230
- エンタープライズ Bean, 226

並び、メッセージ, 165

の

ノード
- エンティティ Bean, 95, 131
- セッション Bean, 62
- メッセージ駆動型 Bean, 156
- 論理, 95, 131
- 論理ノード, 62

は

配備記述子, 15, 49

破棄
- エンティティ Bean インスタンスのプールからの破棄, 37
- セッション Bean インスタンスのメモリーからの破棄, 31

パッケージ (フォルダ) ノード、エクスプローラ, 120, 140, 155

ひ

比較
- セッション Bean の種類, 56 ~ 57

非活性化
- エンティティ Bean のインスタンス, 36
- ステートフルセッション Bean のインスタンス, 32

ビジネスメソッド, 10, 220
- BMP エンティティ Bean, 149
- CMP エンティティ Bean, 109
- エンティティ Bean, 39
- セッション Bean, 30, 70
- ホームメソッドとの比較, 112
- メッセージ駆動型 Bean, 45

ビジネスロジックの実行
- エンティティ Bean, 39
- セッション Bean, 30
- メッセージ駆動型 Bean からの実行, 45

表の対応付け, 85 ~ 90

表のマッピング, 123 ~ 129

EJB ピルダーウィザード
- メッセージ駆動型 Bean の定義, 152 ~ 165

ふ

フィールドの型の変更, 230

プールへの格納
エンティティ Bean のインスタンス, 35
ステートレスセッション Bean のインスタンス, 26, 31
メッセージ駆動型 Bean のインスタンス, 46
複数のメソッドの呼び出しにわたる状態の保持, 26
プログラマが行う必要のある作業
セッション Bean のコーディング, 29
プログラマの役割, 6
プログラムによるセキュリティ, 49, 50
プロパティシート, 49

へ

並行処理、メッセージ駆動型 Bean による模擬実行, 42
変更
Bean、一般規則, 219 ~ 223
変更の伝播, 219 ~ 223
変更の保存, 226
編集
Bean, 219 ~ 223
Bean メソッド, 229
CMP Bean のプロパティシートでの SQL 文, 187 ~ 194
CMR, 127 ~ 129
EJB QL 文, 111

ほ

ホームインタフェース, 12, 14
エンティティ Bean, 95, 131
セッション Bean, 62
「ローカルホームインタフェース」も参照
ホームメソッド, 10, 220
ほかの Bean から取り込んだクラスの修正, 227

ま

マルチスレッド
メッセージ駆動型 Bean による模擬実行, 42
マルチスレッド対応コード、エンタープライズ Bean では不要, 33

む

無限送信メッセージ, 165

め

メソッドの実行権限, 50
メッセージ駆動型
送信先, 160
メッセージ駆動型 Bean
Bean クラス, 157
`onMessage` メソッド, 159
`setMessageDrivenContext` メソッド, 160
開発, 151 ~ 165
概要, 42
加入型モデル
コードの完成, 158 ~ 160
通知モード, 165
トランザクション管理, 153
発行型モデル, 160
問題回避, 165
メッセージ指向型ミドルウェア, 42
メッセージセレクタ, 160
メッセージの順序, 165
メッセージ用のフィルタ、メッセージセレクタを参照

も

モジュール、「EJB モジュール」 を参照
問題
アプリケーションレベル, 48
エラー情報, 223
システムレベル, 48
論理ノード以外での操作, 219 ~ 221

や

役割
J2EE アプリケーションの開発, 6

ゆ

有効化
メッセージ駆動型 Bean のインスタンス, 44
ユーザーのセキュリティロール, 49

ら

ラージアイコン, 171
ライフサイクル
エンティティ Bean, 35
セッション Bean, 29
メソッド, 10
BMP エンティティ Bean, 143 ~ 144
CMP エンティティ Bean, 102 ~ 103
セッション Bean, 68 ~ 69
メッセージ駆動型 Bean, 44

り

リソース
ステートフルセッション Bean, 32
リソース環境参照
キューまたはトピック, 161 ~ 165
リソースファクトリ参照, 177 ~ 180
リソースへの接続
エンタープライズ Bean 一般, 177
メッセージ駆動型 Bean, 161
リモートインタフェース, 11, 13
エンティティ Bean, 95, 131
セッション Bean, 62
「ローカルインタフェース」も参照
リモートオブジェクト, 37
リモートに参照される エンタープライズ Bean, 176
リモート例外, 48

れ

例外, 48
`java.rmi.RemoteException`, 48
`javax.ejb.CreateException`, 48
`javax.ejb.EJBException`, 48
システムレベルとアプリケーションレベル, 72
事前定義例外, 48
リモート例外, 48
例外を使用した問題への対処, 48

ろ

ローカルインタフェース
概要, 13
「リモートインタフェース」も参照
ローカルホームインタフェース
概要, 14
「ホームインタフェース」も参照
ロール
セキュリティ, 49
論理ノード, 62, 95, 131, 219, 219 ~ 221