Enterprise JavaBeans コンポーネントのプログラミング

Forte™ for Java™ プログラミングシリーズ

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 816-2845-01
2001 年 10 月 Revision A

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto,California 94303, U.S.A. All rights reserved.

本製品に採用されているテクノロジに関する知的財産権は Sun Microsystems, Inc.が保有しています。 特に、これらの知的財産権には、ウェブサイト http://www.sun.com/patents にリスト表示されている米国特許、 または米国および他の国へ出願中の特許が含まれている可能性 があります。

本製品は、本製品やドキュメントの使用、コピー、配布、および逆コンパイルを規制するライセンス規定に従って配布されます。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 テクノロジを含むサードパーティ製のソフトウェアの著作権およびライセンスは、Sun のサプライヤが保有しています。PointBase ソフトウェアは社内開発での使用のみを目的としており、商用で使用する場合には別途 PointBase からライセンスを取得する必要があります。

Sun、Sun Microsystems、Sun のロゴ、Forte、Java、Jini、Jiro、Solaris、iPlanet、および NetBeans は、米国および他の各国における Sun Microsystems, Inc. の商標または登録商標です。

SPARC は SPARC International, Inc. の米国および他の各国における商標または登録商標であり、 同社とのライセンス契約のもとで使用されています。SPARC の商標を使用した製品は Sun Microsystems, Inc. が開発したアーキテクチャに基づいています。

連邦政府による取得： 市販ソフトウェア -- 米国政府機関による使用は、標準のライセンス条項に従うものとします。

原典： *Building Enterprise JavaBeans™ Components*
Part No: 816-1401-10
Revision A

© 2001 by Sun Microsystems, Inc.

目次

はじめに　xv

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　5

EJB アプリケーションの内側　6

Enterprise Bean の要素　7

Bean メソッド　8

ホームインタフェース　10

リモートインタフェース　10

Bean クラス　11

配備記述子　12

EJB アプリケーションの実行時のワークフロー　12

Enterprise Bean の開発ライフサイクル　14

IDE の Enterprise Bean 機能　15

トランザクション機能　17

持続性機能　17

セキュリティ機能　17

詳細情報の参照先　18

2. 設計とプログラミング　19

どの種類の Bean が必要か　19

セッション Bean　20

ステートレスセッション Bean の使用　21

ステートフルセッション Bean の使用　22

トランザクションモードの選択　23

セッション Bean のライフサイクル　25

エンティティ Bean　28

EJB コンテナのサービスの使用　29

エンティティ Bean のライフサイクル　30

アプリケーションでのセッション Bean とエンティティ Bean の使用　36

1 つのセッション Bean と複数のエンティティ Bean の併用　36

1 つのセッション Bean の使用　36

1 つのエンティティ Bean の使用　37

例外を使用した問題の対処　38

透過的持続性 (Transparent Persistence) の使用　39

配備記述子の操作　39

セキュリティポリシーの適用　39

Enterprise Bean のセキュリティの宣言　40

Enterprise Bean のセキュリティのプログラミング　41

アプリケーションサーバー　41

詳細情報の参照先　42

3. セッション Bean のプログラミング　43

EJB ビルダーの使用　44

セッション Bean の種類の選択　45

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　45
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　47
セッション Bean での透過的持続性　48
セッション Bean の定義　48
パッケージの作成　49
EJB ビルダーのウィザードの起動　49
デフォルトのセッション Bean の作成　49
セッション Bean のクラスの参照　52
論理ノードとプロパティシートを使用した作業　53
ソースエディタを使用したセッション Bean の修正　54
IDE のエラー情報　55
ノードの展開　56
生成されたクラスの確認　56
デフォルトの生成メソッド　56
ライフサイクルメソッド　57
セッション Bean の完成　59
生成メソッドの完成　59
ステートレス Bean の生成メソッドの完成　59
ステートフル Bean の生成メソッドの完成　60
ステートフル Bean への生成メソッドの追加　60
ライフサイクルメソッドの完成　61
ビジネスメソッドの追加　62
トランザクションのコーディング　63
トランザクション範囲　63
トランザクション範囲とロールバックの指定　63
セッション同期化の使用　65
セッション Bean を作成した後の作業　67
詳細情報の参照先　67

4. エンティティ Bean のプログラミング　69

エンティティ Bean 作成のためのEJB ビルダーの
使用　69

CMP Bean と BMP Bean の比較　71

CMP エンティティ Bean の定義　72

パッケージの作成　72

EJB ビルダーのウィザードの起動　72

デフォルトの CMP Bean の生成　73

データベース表からの持続フィールドの指定　74

持続フィールドの個別指定　79

既存の Bean クラスと主キークラスの使用　80

データベースマッピングの延期　83

CMP Bean のクラスの参照　83

論理ノードとプロパティシートを使用した作業　84

ソースエディタを使用したエンティティ Bean の修正　85

IDE のエラー情報　87

ノードの展開　87

生成されたクラスの確認　89

デフォルトの検索メソッド　89

持続フィールド　90

主キークラスと必須メソッド　90

CMP Bean のライフサイクルメソッド　91

CMP Bean の完成　93

生成メソッドの定義　94

主キーの追加または置き換え　96

主キーの新規作成　96

外部キー　98

ビジネスメソッドの定義　98

検索メソッドの追加　99
追加フィールドの定義　99
CMP Bean を作成した後の作業　100
BMP エンティティ Bean の作成　100
パッケージの作成　101
EJB ビルダーのウィザードの起動　101
デフォルトの BMP Bean の作成　101
BMP Bean のクラスの参照　102
BMP Bean の操作　103
ノードの展開　104
生成されたクラスの確認　104
findbyPrimaryKey メソッド　104
BMP Bean のライフサイクルメソッド　105
BMP Bean の完成　106
持続性ロジックの追加　107
主キークラスの追加　107
メソッドの追加　108
生成メソッドの定義　108
検索メソッドの追加　110
ビジネスメソッドの定義　111
サーバーによる違い　111
BMP Bean を作成した後の作業　111
詳細情報の参照先　112
5. プログラミング後の作業　113
配備記述子の指定　114
生成された配備記述子の参照　114
プロパティシートを使用した配備記述子の編集　115

「プロパティ」タブ　115
「参照」タブ　116
「J2EE RI」タブ　122
EJB モジュールの作成　130
EJB モジュールに組み込む Enterprise Bean の決定　131
EJB モジュールへの Enterprise Bean の組み込み　131
EJB モジュールへのトランザクション属性の追加　132
EJB JAR の作成　135
Enterprise Bean のテスト　135
テスト環境の設定　135
テストオブジェクトの生成　137
複数の Enterprise Bean のテスト　138
サーバーへの Bean の配備　139
Bean のテスト　139
Bean の修正と再テスト　140
A. Enterprise Bean での透過的持続性の使用　141
透過的持続性の使用目的　141
Enterprise Bean での透過的持続性の処理内容　142
主な違い　142
リソースの取得方法　143
EJB ビルダーの起動　144
Bean のコードの完成　145
シリアライズ　147
トランザクション管理　148
Bean 管理によるトランザクションのコーディング　149
持続性マネージャの操作　150
照会の記述　151

挿入、削除、および更新の記述　151
詳細情報の参照先　152

B. Enterprise Bean の操作　153

Bean の論理ノードを使用した作業　153
変更内容の保存　153
Enterprise Bean の検証　154
Enterprise Bean 名の変更　154
ほかの Bean から取り込んだクラスの修正　155
Enterprise Bean のコピーとペースト　155
Bean のクラスやインタフェースの変更　156
Bean のメソッドの編集　157
エンティティ Bean のフィールドの変更　157
フィールド名の変更　157
フィールドの型の変更　158
Enterprise Bean の削除　158

索引　159

図目次

図 1-1　Forte for Java, Enterprise Edition 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構成　4

図 1-3　Enterprise Bean によるアプリケーションの構成例　7

図 1-4　`CalendarMgr` Enterprise Bean を使用した単純なアプリケーションの要素　8

図 1-5　アプリケーションの実行時のワークフロー　13

図 1-6　Enterprise Bean の開発、アセンブル、および配備　14

図 1-7　生成された Enterprise Bean 要素のエクスプローラウィンドウでの表示　16

図 2-1　Forte for Java IDE での Enterprise Bean の基本的な選択項目　20

図 2-2　1 つのセッション Bean と複数のエンティティ Bean を併用する典型的なアプリケーション　36

図 2-3　1 つのセッション Bean だけを使用するアプリケーションの例　37

図 2-4　1 つのエンティティ Bean だけを使用する CRUD アプリケーションの例　37

図 3-1　ステートレス (またはステートフル) BMT セッション Bean の設定例　50

図 3-2　ステートフル CMT セッション Bean の設定例　50

図 3-3　典型的なセッション Bean のデフォルトクラス　52

図 3-4　典型的なセッション Bean の詳細表示　56

図 4-1　EJB ビルダーのウィザードでの CMP の選択項目　73

図 4-2　典型的なエンティティ Bean のデフォルトクラス　83

図 4-3　典型的な CMP Bean の詳細表示　88

図 4-4　複合主キーを持つ典型的な CMP Bean の詳細表示　89

図 4-5　BMP エンティティ Bean のデフォルトクラス　102

図 4-6　BMP Bean の詳細表示　104

図 5-1　エンティティ Bean 用の「プロパティ」ダイアログボックスの「参照」タブ　117

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　46
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　47
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　57
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　58
表 3-5　トランザクションとメソッドの関係　63
表 4-1　CMP Bean と BMP Bean の選択　71
表 4-2　CMP 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　92
表 4-3　BMP 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　105

はじめに

このマニュアルでは、Forte™ for Java™, Enterprise Edition, IDE を使用して、Enterprise JavaBeans™ コンポーネント (Enterprise Bean) を開発する方法を説明します。Enterprise Bean には、いくつかの種類があります。セッション Bean には、状態を保持するものと保持しないものがあります。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。Forte for Java IDE には、これらの Enterprise Bean の開発を柔軟に支援する機能があります。Forte for Java 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 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 モジュールおよびアプリケーションのアセンブルと実行』では、作成した Enterprise 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し、これらのモジュールをアプリケーションに配備し、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。J2EE™ Blueprints で提唱されている業務分担に基づき、プログラマの役割に合わせて 2 冊のマニュアルを提供しています。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、Enterprise Bean の開発を担当するプログラマ向けに、『J2EE モジュールおよびアプリケーションのアセンブルと実行』は、EJB™ アプリケーションのアセンブルと配備を担当するプログラマ向けに書かれています。Enterprise 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。

Forte for Java IDE では、次のプラットフォームとオペレーティングシステムで Enterprise Bean を作成できます。

- Solaris™ 8 SPARC™ Platform Edition
- Microsoft Windows 2000、SP2

- Microsoft Windows NT 4.0、SP6
- Red Hat Linux 6.2

このマニュアルに掲載している画面イメージは、すべて Windows NT 版の Forte for Java ソフトウェアに基づいています。その他のプラットフォームについても、画面イメージは多少異なりますが、機能はほとんど同じです。

ほとんどの手順では、Forte for Java 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、次のように Microsoft Windows のコマンドウィンドウのプロンプトと構文を使用しています。

```
c:¥>cd MyWorkDir¥MyPackage
```

UNIX® 環境や 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 MyWorkDir/MyPackage
```

お読みになる前に

このマニュアルは、Forte for Java IDE を使用して Enterprise 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
- Enterprise Bean の基本概念
- JDBC™ API
- リレーショナルデータベースの概念 (表、キーなど)
- データベースの使用方法
- XML 構文

Enterprise Bean を開発するには、J2EE の概念と Enterprise Bean についての一般的な知識が必要です。また、特定の開発段階では、アプリケーションサーバーの知識も必要になります。詳細情報が必要な場合は、次の資料を参照してください。

- Java 2 Platform, Enterprise Edition Blueprints—
  `www.java.sun.com/j2ee/blueprints`

- *Java 2 Platform, Enterprise Edition Specification*—www.java.sun.com/products
- *Java 2 Enterprise Edition Developer's Guide*—
www.java.sun.com/j2ee/j2sdkee/devguide1_2_1.pdf
- *Enterprise JavaBeans™Specification*—
http://java.sun.com/products/ejb/docs.html
- *Java™ Transaction API (JTA) Specification*—
http://java.sun.com/products/jta/
- *Developer's Guide (Java™): iPlanet™ Application Server*—
http://docs.iplanet.com/docs/manuals/ias.html

内容の紹介

第 1 章では、J2EE と Enterprise JavaBeans の概念を紹介し、Forte for Java IDE の Enterprise Bean 作成支援機能の概要を説明します。

第 2 章では、Forte for Java IDE を使用して Enterprise Bean を開発する際の、設計とプログラミングの問題を説明します。

第 3 章では、IDE を使用して、状態を保持し、または状態を保持せず、自分自身のトランザクションを自分で管理する、または EJB コンテナにトランザクションを管理させるセッション Bean を作成する方法を説明します。

第 4 章では、IDE を使用して、コンテナ管理による持続性や Bean 管理による持続性を使用するエンティティ Bean を作成する方法を説明します。

第 5 章では、配備記述子、プロパティシート、アセンブルと配備の問題、そして Enterprise Bean のテストについて取り上げています。

付録 A では、JDBC™ API を使用した持続プログラミングの代わりに、Enterprise Bean に Forte for Java IDE の透過的持続性 (Transparent Persistence) モジュールを使用させる方法を説明します。

付録 B では、IDE での既存の Enterprise Bean の操作方法を説明します。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P-1　表記上の規則

字体または記号	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、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
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示します。	system% **su** Password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	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に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

■ C シェルプロンプト

```
system% command y|n [filename]
```

- Bourne シェルおよび Korn シェルのプロンプト

```
system$ command y|n [filename]
```

- スーパーユーザーのプロンプト

```
system# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: `Shift` キーを押します)。ただし、キーボードによっては `Enter` キーが `Return` キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば `Ctrl-D` は `Control` キーを押したまま `D` キーを押すことを意味します。

関連マニュアル

Forte for Java のマニュアルは、Acrobat Reader (PDF) ファイル、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

次のマニュアルは、Forte for Java のポータルサイト、`docs.sun.com` の Web サイト、およびインターネットオンラインブックストアの Fatbrain.com から入手することができます。

Forte for Java ポータルサイトでのマニュアルの入手先は、`http://www.sun.co.jp/forte/ffj/documentation/index.html` です。`docs.sun.com` の URL は、`http://docs.sun.com` です。Fatbrain.com の URL は、`http://www.fatbrain.com/documentation/sun` です。

- リリースノート (HTML 形式)

  Forte for Java 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)

  Forte for Java の Edition ごとに用意されています。対応プラットフォームへの Forte for Java 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。

- Forte for Java プログラミングシリーズ (PDF 形式)

  Forte for Java 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. 816-2849-01

    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。

  - 『持続プログラミング』 Part No. 816-2850-01

    Forte for Java が提供するさまざまな持続性プログラミングモデルのサポート機能について説明しています。特に、JDBC と透過的な持続性についてを詳細に説明しています。

  - 『Enterprise JavaBeans コンポーネントのプログラミング』 Part No. 816-2845-01 (このマニュアル)

    Forte for Java の EJB ビルダーウィザードや、その他のグラフィカルユーザーインタフェースを使用し、Enterprise JavaBeans コンポーネント (コンテナ管理や Bean 管理による持続性の機能を持つセッション Bean やエンティティ 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. 816-2844-01

    Web サービスモジュールが提供するツールを使用して Web サービスを構築する方法を説明しています。Web サービスは、XML (Extensible Markup Language) 文書の形式で提供されるアプリケーションビジネスサービスであり、HTTP を介して配信されます。

- 『XML データサービス用 JSP のプログラミング』 Part No. 816-2843-01

  Forte for Java Enterprise Service Presentation Toolkit (Forte ESP ツールキット) を使用し、HTML に動的 XML データを組み込む方法を説明しています。

- 『J2EE モジュールおよびアプリケーションのアセンブルと実行』 Part No. 816-2846-01

  EJB モジュールと Web モジュールを組み合わせて J2EE アプリケーションを作成する方法と、J2EE アプリケーションを配備して実行する方法を説明しています。

- Forte for Java チュートリアル (PDF 形式)

  チュートリアルアプリケーションは、ユーザー設定ディレクトリの下の `sampledir/tutorial` ディレクトリにあります。

  - 『Forte for Java, Community Edition チュートリアル』 Part No. 816-2847-01

    Forte for Java, Community Edition のツールを使用し、簡単な J2EE Web アプリケーションを作成する方法を順を追って説明しています。

  - 『Forte for Java, Enterprise Edition チュートリアル』 Part No. 816-2848-01

    Enterprise JavaBeans コンポーネント、アプリケーションテスト機能、Forte for Java Web サービス技術を使用し、アプリケーションを作成する方法を順を追って説明しています。

オンラインヘルプ

オンラインヘルプは、Forte for Java 開発環境内から参照できます。ヘルプキー (Solaris オペレーティング環境では Help キー、Windows および Linux 環境では F1 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

Forte for Java の機能を紹介したプログラム例が、関連する `Readme` ファイルとともに、ユーザー設定ディレクトリの `sampledir/examples` ディレクトリに置かれています。また、Forte for Java のポータルサイトから、Enterprise Edition に固有のサンプルファイルをダウンロードし、それらを `sampledir/examples` ディレクトリに置くこともできます。チュートリアルアプリケーション (『Forte for Java, Community

Edition チュートリアル』と『Forte for Java, Enterprise Edition チュートリアル』で説明されているアプリケーションを含む) はすべて、`sampledir/tutorial` ディレクトリに置かれています。

Javadoc

Javadoc 形式のマニュアルは、Forte for Java 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。

Sun のマニュアルのオンラインでの提供

Sun の各種システムのマニュアルを、次の Web サイトで提供しています。

`http://www.sun.com/products-n-solutions/hardware/docs`

Solaris のマニュアルセットとその他の多くのマニュアルを、次の Web サイトで提供しています。

`http://docs.sun.com`

Sun のマニュアルの注文方法

Sun の製品マニュアルは、Fatbrain.com インターネットブックストアを通じて米国 Sun Microsystems, Inc. に直接注文できます。Fatbrain.com の Sun Documentation Center へは次の URL でアクセスできます。

`http://www.fatbrain.com/documentation/sun`

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、対象マニュアルの Part No. (このマニュアルの場合は 816-2845-01) 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 (Enterprise 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を取り扱います。

- J2EE アーキテクチャの概要
- Enterprise 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 (Forte™ for Java™, Enterprise Edition の統合開発環境 (IDE) のウィザードと GUI 機能の集合)

すでに J2EE と Enterprise Bean プログラミング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、第 4 章、および第 5 章を参照してください。

Enterprise Bean を使用できるようにするには、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必要があります。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、Enterprise Bean の開発者や提供者が、アプリケーションにアセンブルし、サーバーに配備する Enterprise Bean (または関連する Bean) を完成させるまでの作業を取り扱います。このマニュアルシリーズの別のマニュアル、『J2EE モジュールおよびアプリケーションのアセンブルと実行』では、アプリケーションのアセンブルや、サーバーへの配備を担当する開発者が実行する作業を説明します。

注 - Forte for Java, Enterprise Edition IDE は、Enterprise JavaBeans™ Specification 1.1 および Java 2 Platform, Enterprise Edition Specification 1.2 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、スケーラブルで、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、Java Virtual Machine (JVM™) 上で動作する Java クライアントを含めることができます。

  Forte for Java, Enterprise Edition IDE は、JSP ページ、サーブレット、およびその他の Enterprise 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。これ以外のクライアントの実行には今のところ対応していません。

- **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 (Enterprise 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。

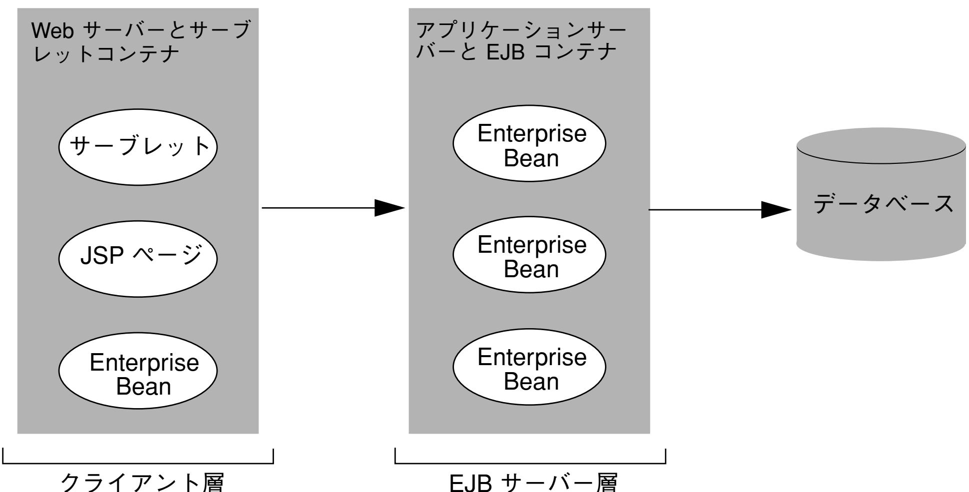


図 1-1　Forte for Java, Enterprise Edition がサポートする J2EE アプリケーションモデル

Enterprise 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、データベース列にマップすることが可能なフィールドを含んでいる Enterprise Bean もあります。また、同じアプリケーションの別々の Enterprise Bean 間の相互作用を管理する能力を持った Enterprise Bean もあります。Enterprise 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。

Enterprise 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、Enterprise 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、Bean 提供者の業務が一層モジュール化され、Bean 提供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が、Enterprise Bean を使用し、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を図 1-2 に示します。この中には、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 Enterprise 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。

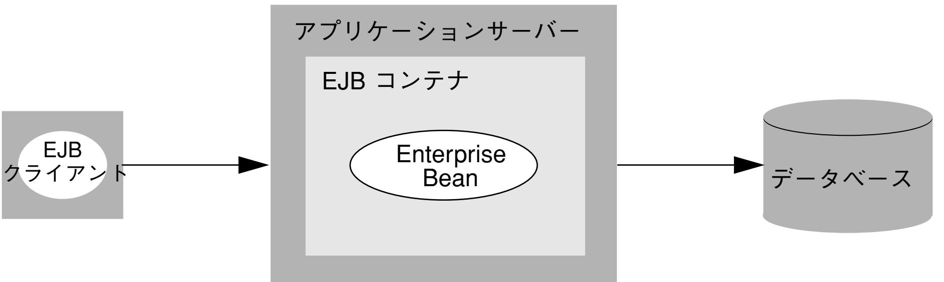


図 1-2　EJB アプリケーションの典型的な構成

Enterprise Bean、EJB コンテナ、アプリケーションサーバーの間の契約 (相互作用と暗黙の合意) により、Enterprise 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、むしろ概念です。EJB コンテナは、EJB サーバー上の Enterprise 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 Enterprise 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 とサーバーとの仲介役になります。クライアントが Enterprise Bean に処理を行わせようとすると、コンテナがメソッドの呼び出しを横取りします。コンテナは、Enterprise Bean やクライアントに代わって、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。この機能により、アプリケーション内部のトラフィックとオーバーヘッドを大幅に削減できます。また、Enterprise Bean の移植性も一層高まります。

コンテナは、個々の Enterprise 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アクセス操作を実行できるため、コンテナのデフォルトの動作を無効にしたい場合を除いて、完全な SQL コードを記述したり、JDBC™ API を直接使用したり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 Enterprise 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低レベル機能を提供します。

Enterprise Bean には、セッション Bean とエンティティ Bean の 2 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 サーバーとの情報のやり取りを管理します。また、エンティティ Bean との複雑な相互作用を指示できます。
- 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) を表現します。

作成した Enterprise Bean は、EJB モジュール (EJB JAR ファイル) にパッケージ化し、アプリケーションへのアセンブルや、コンテナおよびサーバーへの配備に使用します。アプリケーションサーバーは 1 つ以上の EJB コンテナから、EJB コンテナは 1 つ以上の J2EE アプリケーションから、J2EE アプリケーションは 1 つ以上の EJB モジュールから、EJB モジュールは 1 つ以上の Enterprise 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 アーキテクチャには、アプリケーション開発プロセスの方法論と役割が含まれています。通常の開発組織には、ビジネスの知識が豊富なチームメンバーもいれば、システムレベルのプログラミングに精通しているチームメンバーもいます。J2EE モデルでは、前者のメンバーが Bean 提供者の役割を果たし、後者のメンバーがアセンブルや配備を行います。

開発環境では、次の役割が一般的です。

- Bean 提供者 (アプリケーション開発者、通常はドメインエキスパートでもあります) は、再使用可能な Enterprise 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。
- アセンブル担当者は、コンテナに依存しない方針決定を行い、Bean 提供者が作成した Enterprise Bean を EJB モジュールにアセンブルし、これらのモジュールとその他の J2EE コンポーネントを組み合わせてアプリケーションを構築します。
- 配備担当者は、コンテナ固有の方針決定を行い、完成した J2EE アプリケーションを特定の環境に配備します。

Enterprise 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。Bean 提供者がこの IDE を使用して Enterprise Bean を作成するときは、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。アセンブル段階や配備段階のことは、最小限のことしか考慮する必要はありません。

ただし、必要であれば、上の 3 つの役割を 1 人の人間がすべて担当することができます。IDE には、Enterprise Bean のすべての開発段階をシームレスにサポートする機能があります。

EJB アプリケーションの内側

図 1-3 に示す典型的な EJB アプリケーションでは、EJB サーバー上に存在し、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 つのセッション Bean のインスタンスが、4 つのエンティティ Bean との相互作用を管理し、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。

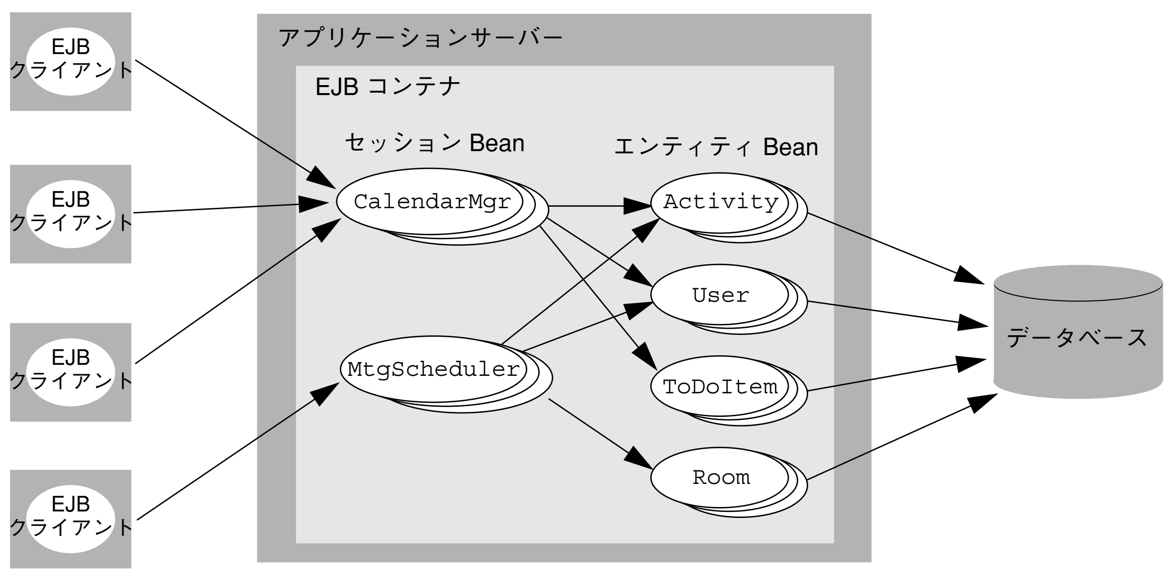


図 1-3　Enterprise Bean によるアプリケーションの構成例

Enterprise Bean の要素

それぞれの Enterprise Bean は、ホームインタフェース、リモートインタフェース、Bean クラスの 3 つのクラスから構成されます (エンティティ Bean には、これらのクラスに加えて主キークラスが含まれています。詳細については第 4 章を参照してください)。ここからは、これらのクラスの目的と基本規則を説明します。これらのクラスと単純なアプリケーションのその他の要素との関係を図 1-4 に示します。

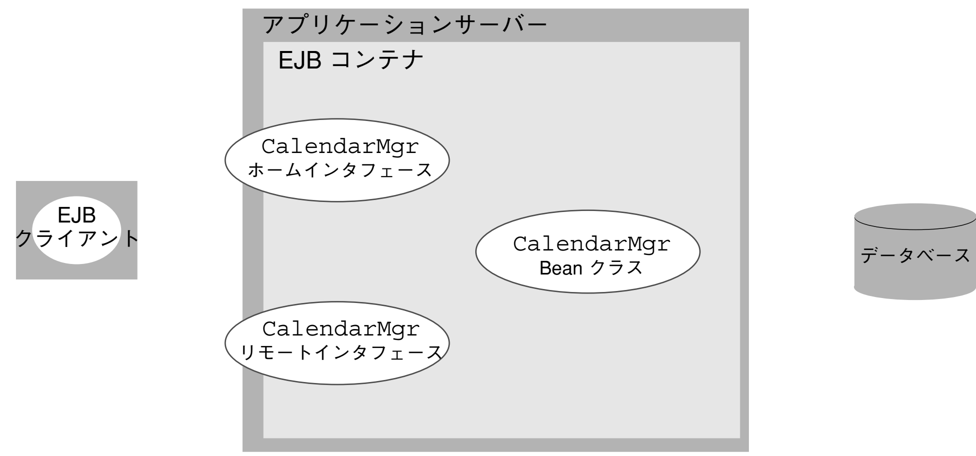


図 1-4　`CalendarMgr` Enterprise Bean を使用した単純なアプリケーションの要素

このアプリケーションの要素の役割は次のようになります。

- Bean 提供者は、IDE を使用して、リモートインタフェース、ホームインタフェース、および Bean クラスを作成します (エンティティ Bean の場合は主キークラスも定義します)。Bean 提供者は、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通信を管理します。
- 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 クラスと通信し、アプリケーションの処理を実行させることができます。

(アプリケーションの各要素が実行時にどのように相互作用するかについては、図 1-5 を参照してください。)

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、Enterprise 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 章と第 4 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、Bean 提供者が明示的に追加します。ダイア

ログボックスで 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。Bean 提供者は、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 のホームインタフェースに `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 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。Bean 提供者が他の検索メソッドを追加した場合は、IDE はそれに対応するメソッドの宣言を、ホームインタフェースと Bean クラスに自動的に追加します。

- **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、Enterprise 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生成メソッドの宣言を生成し、さらに Bean クラスに、それに対応する `ejbCreate` メソッドの宣言を配置します。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、Bean 提供者がエンティティ 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を介して、Bean のビジネスメソッドを呼び出します。

  Bean 提供者は、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、Bean 提供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、リモートインタフェースと Bean クラスに配置します。

- **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、Enterprise 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、Bean 提供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。

ホームインタフェース

Enterprise 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、クライアントから Enterprise 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特定し、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 Enterprise 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 Enterprise 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でのメソッドシグニチャに対応するメソッドが、Bean クラスに含まれていなければなりません (コンテナに持続性を管理させる Enterprise Bean の検索メソッドを除きます)。
- 引数と戻り値は有効な RMI 型でなければなりません。
- メソッドの `throws` 句に適切な例外クラスが含まれていなければなりません。

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 OrderHome のように表示されます。デフォルトのラベルは、Enterprise Bean の名前の後ろに "Home" を付けたものになります。

リモートインタフェース

クライアントは、Bean のリモートインタフェースを介して Enterprise Bean にアクセスします。このインタフェースでは、クライアントから Bean を呼び出すことのできるビジネスメソッドを定義します。ただし、ビジネスメソッドの完全なコードは Bean クラスに配置します。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提供します。

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なので、ホームインタフェースを取得したり、Enterprise Bean インスタンスのハンドルを取得したり、エンティティ Bean インスタンスの主キーを取得したり、コンテナに Bean インスタンスを削除させたりすることができます。

Forte for Java IDE を使用して Enterprise Bean を作成すると、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 の文書で定められた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でのメソッドシグニチャに対応するメソッドが、Bean クラスに含まれていなければならない。
- 引数と戻り値は有効な RMI 型でなければならない。
- メソッドの `throws` 句に適切な例外クラスが含まれていなければならない。

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 Order のように表示されます。デフォルトのラベルは、Enterprise Bean の名前と同じになります。

Bean クラス

Bean クラスは、ほかの 2 つのクラスで定義された実装を含んだ Enterprise Bean の心臓部です。このクラスは、`javax.ejb.EntityBean` インタフェース (エンティティ Bean の場合) または `javax.ejb.SessionBean` インタフェース (セッション Bean の場合) のサブクラスです。Bean クラスには、Enterprise Bean の検索メソッド、生成メソッド、およびビジネス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。

Forte for Java IDE を使用して Enterprise 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 Enterprise 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 クラスは abstract クラスや final クラスではなく、public クラスとして定義されていなければならない。
- Bean クラスにはパラメータのない public なコンストラクタが含まれていなければならない。
- Bean クラスには、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なければならない。
- Bean クラスには、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なければならない (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)。

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 OrderEJB のように表示されます。デフォルトのラベルは、Enterprise Bean の名前の後ろに "EJB" を付けたものになります。

配備記述子

Enterprise Bean がサポートするインフラストラクチャは、その配備記述子で決まります。配備記述子とは、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、Enterprise Bean を構成しているクラス、Bean からほかの Bean への参照、実行先の環境の設定、および実行時の Bean の管理方法を記述した XML ファイルです。

Forte for Java IDE を使用して Enterprise 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 (配備記述子は直接操作するのではなく、Enterprise 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。ただし、エクスプローラから判読可能な形式の記述子ファイルを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、EJB クライアントは最初に Enterprise 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。EJB クライアントが Enterprise 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相互作用を図 1-5 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。Enterprise Bean の種類によっては、一部の手順 (インスタンスのプールへの格納など) が適用されない場合があります。

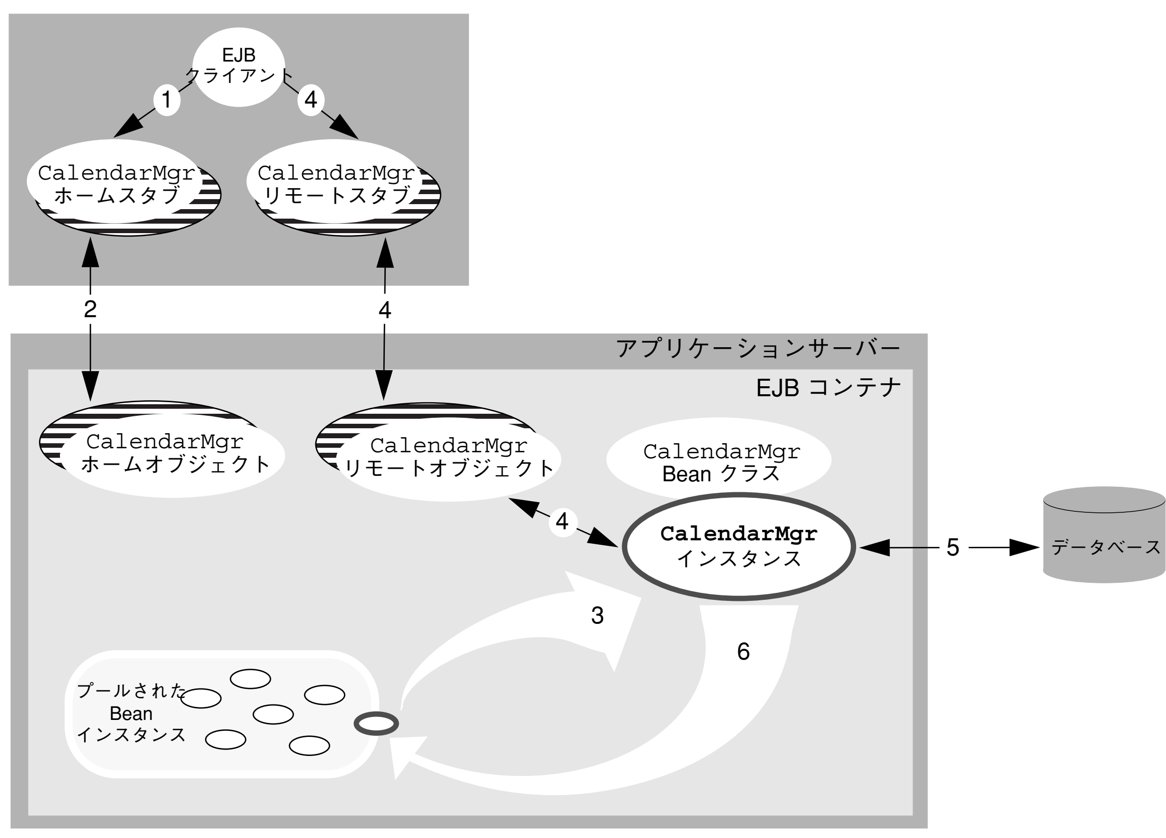


図 1-5　アプリケーションの実行時のワークフロー

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 Enterprise Bean を検出します (すなわち、クライアントは JNDI 検索メソッドを使用し、Enterprise 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出し、このセッションでこのクライアントが使用する Enterprise 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行わなくても、複数の同時ユーザーに対応することができます。Enterprise 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、Bean 提供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。

Enterprise Bean の開発ライフサイクル

図 1-6 に示すように、Enterprise Bean は、Bean 提供者が作成してから使用可能になるまでにいくつかの手順が必要です。

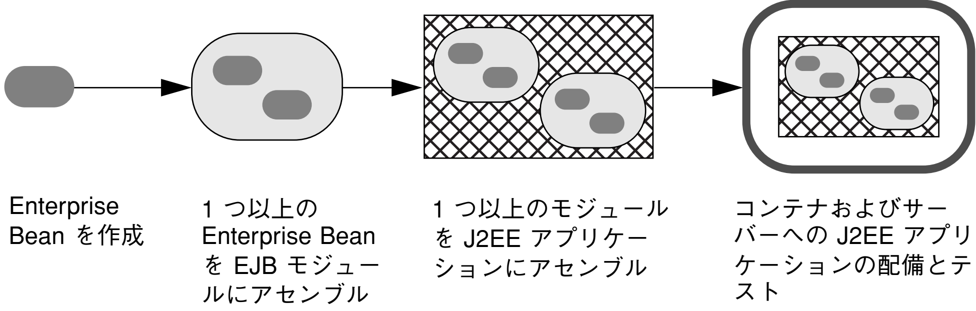


図 1-6　Enterprise Bean の開発、アセンブル、および配備

Forte for Java IDE を使用する Bean 提供者は、次の手順に従って Enterprise 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 章と第 4 章で述べる EJB ビルダーのウィザードとその他の GUI 機能を使用し、Enterprise 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、Enterprise 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、Enterprise Bean を関連するほかの Enterprise 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 5 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、テストを行います。この作業を行うには、ある程度のアセンブル作業と配備作業を実施する必要があります。Bean は後から本稼働環境の別のサーバーに配備することができます。

IDE の Enterprise Bean 機能

Forte for Java IDE では、Enterprise Bean のプログラミング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。
- トランザクションと持続性を管理するコードの作成。これらの管理作業はアプリケーションサーバーが受け持ちます。
- Bean のクラス、インタフェース、およびメソッドの同期処理。IDE が整合性を保ちます。
- 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。
- Enterprise Bean をテストするテストクライアントの作成。IDE が GUI ベースの総合的な Bean テスト機能を提供します。
- J2EE 文書の検索。IDE が生成した Enterprise Bean のソースコードは J2EE 標準に準拠していて、次のようなコメントを含んでいます。

```
/**
 * Reference EJB specification 1.1 section 9.2.5
 */
public String ejbFindByPrimaryKey(String aKey) {
    return aKey;
}

/* Methods required for EntityBean interface. EJB 1.1 section 9.4 */

/**
 * @see javax.ejb.EntityBean#ejbActivate()
 */
public void setEntityContext(EntityContext context){
    this.context = context;
```

EJB ビルダーのウィザードを使用して、基本的な Enterprise Bean 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 またはエンティティ Bean) に合わせて自動的に調節され、トランザクション管理や持続性管理の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を作成できます。

IDE が生成する Enterprise 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を図 1-7 に示します。この図では例としてエンティティ Bean を使用しているため、主キークラスが含まれています。

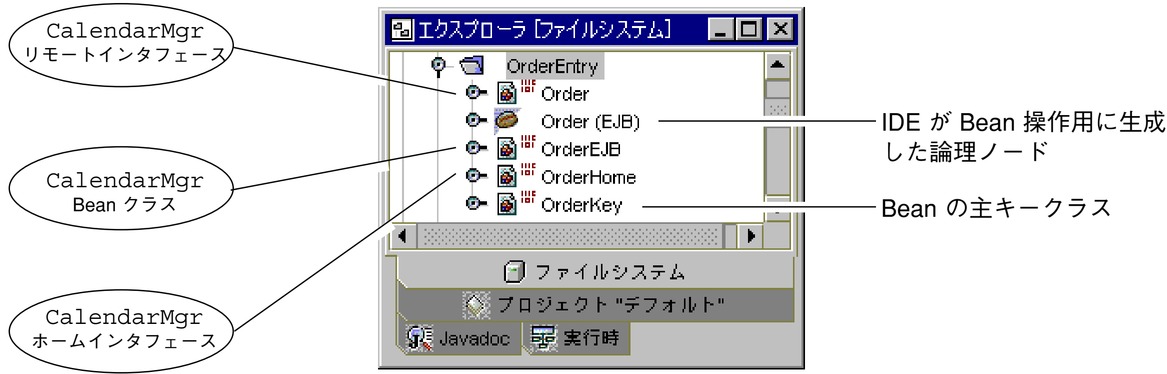


図 1-7 生成された Enterprise Bean 要素のエクスプローラウィンドウでの表示

次に、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。

ここからは、IDE に用意された、Enterprise Bean のトランザクション、持続性、およびセキュリティのプログラミング機能を説明します。

トランザクション機能

Enterprise 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 提供者に代わってトランザクションを管理します。Bean 提供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 のプロパティシートで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を明示的にプログラムしなければならない場合もあります。IDE では、コンテナを明示的に無効にし、JDBC API、JTA (Java Transaction API)、または IDE の透過的持続性 (Transparent Persistence) モジュールを使用して、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 Bean 提供者自身がコーディングすることもできます。Bean 提供者自身が持続性を処理する場合は、JDBC コードを記述するか、透過的持続性を使用します。コンテナ管理による持続性を使用したい場合は、まず EJB ビルダーのウィザードでいくつかの選択を行い、次にプロパティシートでいくつかの宣言を行なって、コンテナがデータストアを検出できるようにします。

セキュリティ機能

Enterprise 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。ただし、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 のコード中のセキュリティの参照が、メソッドで宣言したセキュリティロールに対応付けられます。この対応付けを行うには、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。

詳細情報の参照先

Enterprise 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』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、このマニュアルの「はじめに」を参照してください。

第2章

設計とプログラミング

Enterprise 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。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。この章では、これらの内容を取り上げ、最後に詳細情報についての文献リストを掲載します。

どの種類の Bean が必要か

『Enterprise JavaBeans™ Specification, Version 1.1』では、セッション Bean とエンティティ Bean という 2 種類の Enterprise 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。Forte for Java IDE の EJB ビルダーを使用すると、これらの Enterprise 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、Enterprise 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 Enterprise Bean を作成する場合の基本的な選択項目を示します。

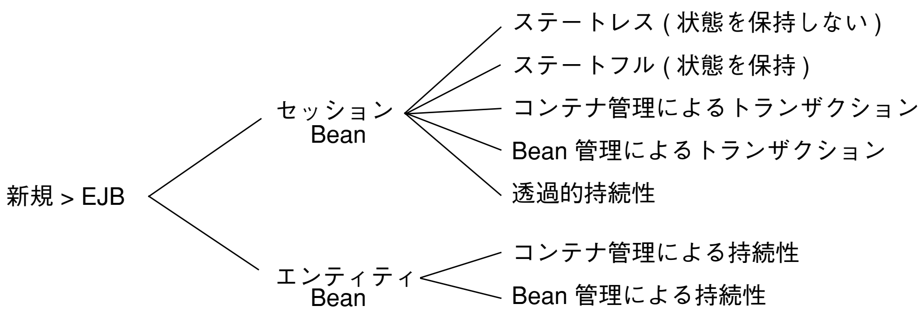


図 2-1　Forte for Java IDE での Enterprise Bean の基本的な選択項目

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。

Enterprise 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しか使用されないため、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。これらの商品をユーザーが実際に購入する手続きをする前にサーバーが停止した場合は、トランザクションでこれらの商品をデータベースに保存すべきではありません。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

セッション Bean は、クライアントとの間で短く、単純な通信を行い、単一のメソッドにパラメータを渡すだけで処理を行わせることができます。代わりに、多数のメソッドとデータベーストランザクションに関係する、複雑で長い通信を管理することもできます。その場合は、セッション Bean は複数のメソッドの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、メソッド呼び出し間の状態を保持しません。このような軽量級の 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 で顧客の注文を処理する場合を考えてみます。それぞれの注文は、`processOrder` 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、注文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。セッションが終了したり、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、Bean のインスタンスおよびその状態は保持されません。

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。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スコープは、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。Forte for Java 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 Enterprise 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) の低レベル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。

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、Bean 提供者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。Bean 提供者は、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 - Bean 提供者は、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。Bean 提供者は、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かリモートインタフェースの `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クローズします。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。Bean 提供者は、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 Enterprise 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休止

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。

ただし、リソースの使用量を制御するため、コンテナはステートフルセッション Bean について一定時間あたりのアクティブなインスタンス数を管理する必要があります。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休止させ、その通信状態を 2 次記憶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、Bean 提供者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シリアライズ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 2 次記憶域に書き出します。

クライアントが休止中の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 2 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。Bean 提供者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、シリアライズ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。Bean 提供者は、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

Bean 提供者は、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。Bean 提供者は、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。Bean 提供者は、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、データのエンティティが 1 つずつ収容されます (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

ロジックが収容される場合もあります)。クライアントや、クライアントのために処理を行うセッション 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 には主キーとリモート参照があります。そのため、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たとしても、エンティティ Bean は失われません。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に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、Enterprise Bean では、マルチスレッド対応コードを使用する必要はありません。

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使用

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。Bean 提供者は、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スコープを識別します。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、および検索メソッドに、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。

Bean 提供者は、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。Bean 提供者は、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。たとえば、J2EE の Reference Implementation (RI) サーバーを使用している場合は、IDE がこ

のマッピングを実行する SQL 文を生成します。ただし、メソッドによっては、もしくはマッピング規則を変更したい場合は、生成された SQL 文を部分的に変更する必要があります。

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。

ただし、マッピングツールがサポートしていない従来のコードをエンティティ Bean に組み込む場合は、Bean 管理による持続性を選択し、エンティティ Bean クラスにデータベース呼び出しをすべてコーディングする必要があります。Bean 管理による持続性では、エンティティの状態をより柔軟に管理できます。たとえば、Bean 管理による持続性を使用するエンティティ Bean (BMP エンティティ Bean) では、複雑な結合や複数の異なるデータベースへのアクセスを、より適切に取り扱うことができます。

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、Bean 提供者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。Bean 提供者がこのメソッドに記述したコードに従って、(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します。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格納し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格納は、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休止させ、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、Bean 提供者がこのメソッドに記述したコードに従って、プール状態のインスタンスに不要なリソースを解放します。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。

休止された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。Bean 提供者は、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インタフェース、主キーといった識別情報を、ほかの Enterprise 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。26 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。Bean 提供者は、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。Bean 提供者は、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。26 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。30 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。Bean 提供者は、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 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 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。Bean 提供者は、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 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。Bean 提供者は、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 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、インスタンスが休止されたときに、インスタンスの `ejbStore` メソッドを呼び出します。
  - CMP 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。Bean 提供者は、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 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 Bean では、Bean 提供者がこれらのメソッドを完成させる必要があります。CMP 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。

アプリケーションでのセッション Bean とエンティティ Bean の使用

セッション Bean やエンティティ Bean をどのように組み合わせればよいかは、アプリケーションのニーズによって決まります。セッション Bean だけ、またはエンティティ Bean だけを使用するのが最適な場合もあります。ここからは、設計例をいくつか紹介します。

1 つのセッション Bean と複数のエンティティ Bean の併用

セッション Bean の典型的な使用法を図 2-2 に示します。1 つのセッション Bean が種類の異なる複数のエンティティ Bean を操作できます。

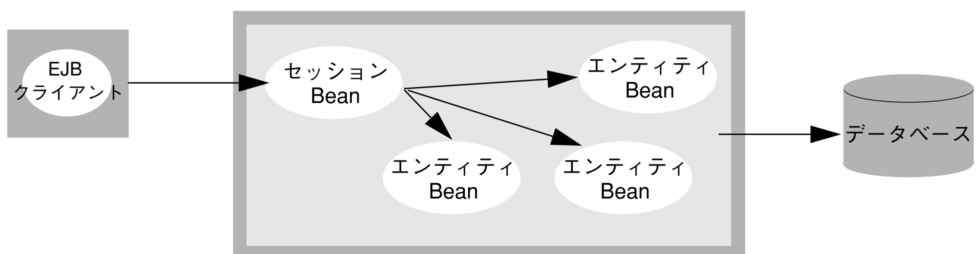


図 2-2　1 つのセッション Bean と複数のエンティティ Bean を併用する典型的なアプリケーション

架空のアプリケーション、`HumanResources` では、この構成のクライアント、(1 つのセッション Bean と複数のエンティティ Bean を持つ) コンテナ/サーバー、データベースを使用します。この例では、新規採用後に従業員データベースを更新するために、アプリケーションクライアントが `HRManager` というセッション Bean を呼び出します。`HRManager` は、新しいエンティティ Bean (`Employee`) を作成し、さらにほかの 2 つのエンティティ Bean (`Manager` と `Department`) を操作します。これらの更新は、すべて単一のデータベーストランザクションの一部になります。`HRManager` はコンテナと連携して、このトランザクションを処理します。

1 つのセッション Bean の使用

セッション Bean の別の使用法を図 2-3 に示します。

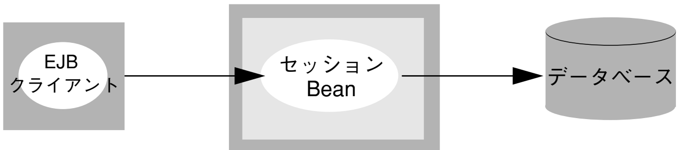


図 2-3　1 つのセッション Bean だけを使用するアプリケーションの例

読み取り専用のデータアクセスを使用して単純な処理を行うアプリケーションでは、この構成のクライアント、(1 つのセッション Bean だけを使用する) コンテナ/サーバー、データベースが一般的です。たとえば、`CatalogController` というセッション Bean は、JDBC API を使用して、データベースからカタログ情報を取り出し、クライアントに返すことができます。

この架空のアプリケーション、`BrowseCatalog` では、データベースへのアクセスは読み取り専用です。そのため、エンティティ Bean は必要ありません。実際、エンティティ Bean を使用すると、不要なオーバーヘッドが発生してしまいます。コンテナがより多くのオブジェクトを作成し、セッション Bean とエンティティ Bean の間で、より多くの RMI/IIOP 通信が発生するからです (ほかのアプリケーションで権限を持つ数人のユーザーが、カタログの読み取りと更新の両方を行います。この場合 1 つ以上のエンティティ Bean を使用する方が適切です)。

単純な操作では、セッション Bean がデータベース中の表を直接更新することができます。その場合は、Enterprise Bean に持続フィールドを作成する代わりに、メソッドのパラメータとして情報を渡すことができます。

1 つのエンティティ Bean の使用

アプリケーションは、必ずしもセッション Bean を使用する必要はありません。数人のユーザー (ここではシステム管理者とします) だけが使用する、企業のデータベース表を管理するアプリケーションを図 2-4 に示します。

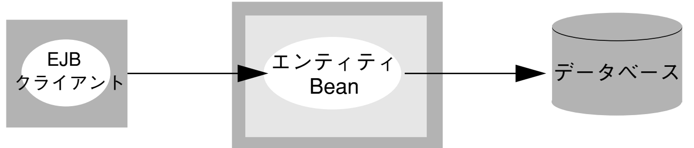


図 2-4　1 つのエンティティ Bean だけを使用する CRUD アプリケーションの例

この例では、アプリケーションは限られたユーザーからしか使用できず、同時に使用されることはめったにありません。セッション Bean は必要ありません。

より複雑なビジネストランザクションを実行する必要がある場合は、アプリケーションの別バージョンにセッション Bean を追加することができます。必要に応じてコンテナ管理によるトランザクションを使用し、同じセッション Bean を別々の種類のクライアントに再利用させることができます。EJB アプリケーションは非常に柔軟性があります。

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、データベースがフルのために 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。

Enterprise 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、Bean 提供者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。

Forte for Java IDE のウィザードを使用して、Enterprise 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。

Forte for Java IDE のウィザード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。

透過的持続性 (Transparent Persistence) の使用

Enterprise Bean の持続性を取り扱うために、JDBC API や、Forte for Java IDE の透過的持続性 (Transparent Persistence) モジュールを使用できます。透過的持続性では、持続データが Bean 内のフィールドとしてではなく、持続可能 Java クラスとして表現されます。これらの持続可能 Java クラスは、持続性を認識可能な Enterprise Bean から操作することができます。外部キーの関係やデータストアへの複数アクセスをサポートする Bean を作成する必要がある場合は、透過的持続性を使用してください。

配備記述子の操作

Enterprise Bean の基本目的は、同じ Enterprise 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。

Enterprise 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をオーバーライドしたり、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 Bean 提供者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 Enterprise 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、Enterprise Bean のコードの中で、Bean 提供者が定義します。

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。

Enterprise 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、Enterprise 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。

Enterprise Bean のセキュリティの宣言

Enterprise Bean を EJB モジュールにアセンブルした後で、Forte for Java を使用して、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。

Enterprise 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、Bean 提供者は、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。

アプリケーションサーバー

Forte for Java IDE で作成した Enterprise Bean は、J2EE 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。J2EE アプリケーションサーバーは、Forte for Java IDE とともに Reference Implementation (RI) に含まれています。RI は、非商用のオペレーションサーバーで、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。Enterprise 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。

本稼働目的では、EJB アプリケーションを iPlanet™ Application Server (iAS) といった商用のアプリケーションサーバーに配備することができます。Forte for Java IDE には、iAS 用のプラグインが含まれています。iAS を使用する場合は、iAS とそのマニュアルを IDE にインストールする必要があります。詳細については、次の Web サイトにある『Developer's Guide (Java™): iPlanet™ Application Server』を参照してください。

http://docs.iplanet.com/docs/manuals/ias.html

この 2 つのサーバーには、共通点もあれば、相違点もあります。詳細については、各サーバーのマニュアルを参照してください。

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ブループリントに加えて、Enterprise Bean 提供者向けの多数の資料があります。たとえば、次の各文書には、Enterprise 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。

- 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/

- The Java 2 Enterprise Edition Developer's Guide

  http://java.sun.com/j2ee/j2sdkee/devguide1_2_1.pdf

第3章

セッション Bean のプログラミング

Forte for Java 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 と JTA (Java Transaction 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、Enterprise Bean の 3 つの要素、すなわちリモートインタフェース、ホームインタフェース、Bean クラスを簡単に作成することができます。必要な作業の多くが自動化されます。

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。Forte for Java 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、「はじめに」に記載した文献、または Enterprise 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、Enterprise 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、「ファイル」 > 「新規」を選択し、テンプレート選択ダイアログのテンプレートリストに「EJB」が含まれているかどうかをチェックします。

Forte for Java 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。この章に記載している手法では、J2EE 標準に従った Bean を作成するために、IDE の機能を最大限に活用しています。

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。必要な 3 つのクラスと論理ノードが作成され、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、この 3 つ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。「新規メソッド」メニュー項目を使用するか、ソースファイルにメソッドの宣言を直接入力して、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、論理ノードから呼び出すことができます。

セッション Bean の論理ノードから、Bean を検証したり、Bean 用のテストアプリケーションを作成することができます。

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、透過的持続性を使用した方が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。

- 45 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」
- 47 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」
- 48 ページの「セッション Bean での透過的持続性」

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 の主目的は、クライアントアプリケーションのために処理を行い、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	この 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	この 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	この Bean には、初期化が必要なデータはない	Bean が作成されたときに、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、リリースファクトリの参照を取得する必要がある
情報の保存	この 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	この 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	この 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	この 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)
アプリケーション例	1 つのメソッドで、エンドユーザーにオンラインカタログの 1 つの商品だけを検索させるカタログビューア	複数のメソッドを使用して、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するオンラインショッピングカート

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 Specification 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	Bean 提供者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 Enterprise 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 での透過的持続性

どの種類のセッション Bean についても、(その中に持続フィールドが含まれ、データベース中のデータを取り扱う限り) 生成されたコードを JTA と JDBC を使用して完成させることができます。代わりに、IDE の透過的持続性 (Transparent Persistence) モジュールを使用して持続可能クラスを作成し、EJB ビルダーのウィザードでこれらのクラスを使用するセッション Bean を作成し、生成されたコードを付録 A のガイドラインに従って完成させることもできます。

透過的持続性は、リレーショナルデータベースに格納される持続データのオブジェクトビューを提供します。持続可能 Java クラスは、Bean クラスで使用する SQL 文に取って代わるもので、データストアに依存しないコードを記述できるという特長があります。透過的持続性を使用した場合は、セッション Bean は持続性マネージャを取得し、持続性マネージャのインタフェースを使用して、持続 Bean インスタンスの作成、読み取り、更新、および削除を行います。

この機能を選択すると、EJB ビルダーのウィザードは Bean クラスにコードを生成し、Bean 提供者が透過的持続性モジュールで定義した持続可能クラスを、Bean から使用できるようにします。持続可能クラスを使用する Enterprise Bean を、持続性認識 Bean といいます。

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 (リモートインタフェース、ホームインタフェース、Bean クラス) 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 の収容先の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、基本的なセッション Bean 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 を収容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。代わりに、エクスプローラの既存のディレクトリ (フォルダ) ノードに Bean を収容することもできます。

EJB ビルダーのウィザードの起動

1. 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」区画から、セッション Bean の収容先のパッケージやファイルシステムを選択します。
3. 右クリックし、「新規」>「EJB」>「セッション Bean」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 EJB 型」区画で、状態、トランザクション型、透過的持続性、セッション同期についての選択を行う必要があります。

1. セッション Bean の種類を選択します。

   ボタンをクリックして、Bean の状態、トランザクションモード、透過的持続性を使用するかどうか、そしてセッション同期化インタフェースを実装するかどうかを指定します。状態を保持し、または状態を保持せず、トランザクションを自分自身で管理するセッション Bean を作成する場合は、図 3-1 のように選択します。「ステートレス (状態を保持しない)」と「コンテナ管理によるトランザクション」がデフォルトの選択項目です。

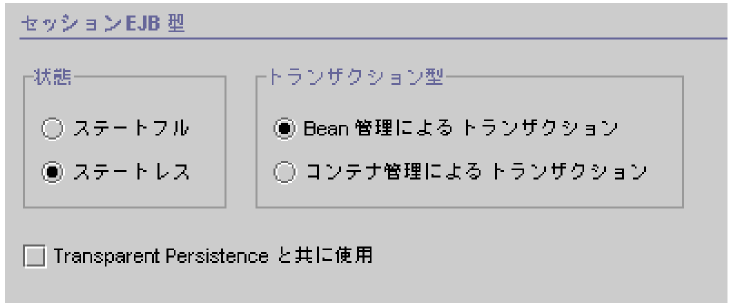


図 3-1　ステートレス (またはステートフル) BMT セッション Bean の設定例

図 3-2 のように、「ステートフル (状態を保持する)」と「コンテナ管理によるトランザクション」を同時に選択した場合は、Bean にセッション同期化インタフェースを実装するかどうかも選択することができます。

図 3-2　ステートフル CMT セッション Bean の設定例

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 (ステートフルかどうか、トランザクションをコンテナに管理させるかどうか、透過的持続性を使用するかどうか、セッション同期化インタフェースを実装するかどうか) を後から変更する場合は、Bean を削除し、作成し直す必要があります。

---

2. 「次へ」をクリックします。

   「EJB コンポーネント」区画が表示されます。

3. セッション Bean の名前を入力します。

4. この区画のほかのフィールドの内容をチェックし、必要に応じて変更します。

「関連するオブジェクト」区画に、セッション Bean を構成する 3 つの必須クラスが表示されます。

- パッケージの場所を変更することができます。
- 「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 3-3 とは別の形式で表示されます。
- 通常は、Bean のスーパークラスは変更しないでください。(クラスの隣にある「変更」ボタンをクリックして) スーパークラスを変更する場合は、表示される警告メッセージを必ず参照してください。Bean のスーパークラスとして選択できるのは、適切なインタフェース (`javax.ejb.SessionBean`、`javax.ejb.EJBObject`、または `javax.ejb.EJBHome`) のサブクラスになっているものだけです。

注 - IDE が生成するコードによって、スーパークラスのメソッドがオーバーライドされる場合があります。スーパークラスを変更した場合は、生成されたコードをチェックし、必要に応じて修正してください。

これらのフィールドを変更する場合は、次のことに注意してください。

- **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収容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収容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。

5. 「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。

注意 - セッション Bean は、明示的に保存しない限り保存されません。

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 3-3 のように表示されます。

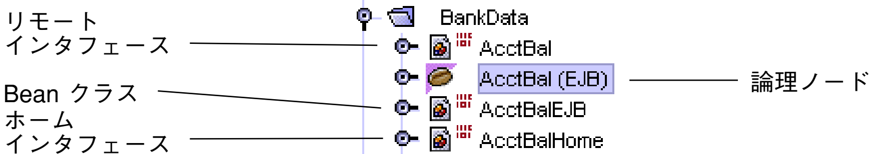


図 3-3　典型的なセッション Bean のデフォルトクラス

表示される 4 つのノードのうち、アイコンが付いた 3 つのノードは、セッション Bean のクラスを表しています。アイコンが付いた残りの 1 つのノードは、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。

■ 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、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、エンティティ 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、生成メソッドのシグニチャを提供します。

■ 論理ノードは、Enterprise 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。

論理ノードとプロパティシートを使用した作業

EJB ビルダーのウィザードで、セッション Bean のデフォルトクラスを生成した後で、論理ノードに移動し、Bean の定義作業を続行します。Bean のクラスは、すべてこの単一のノードにまとめられるため、Bean を単一のオブジェクトであるかのように取り扱うことができます。論理ノードでは、それぞれの変更をどのクラスに適用するかを考慮することなく、Enterprise Bean を編集することができます。このノードを使用して Bean を編集すると、編集内容が Bean 全体に適切に伝達されます。この例を次に示します。

■ 論理ノードの中の「生成メソッド」ノードに新しいメソッドを追加すると、そのメソッド (`ejbCreate`) の本体が Bean クラスに追加され、それに対応するメソッドシグニチャ (`create`) がホームインタフェースに追加されます。

■ 論理ノードの中の「ビジネスメソッド」ノードに新しいメソッドを追加すると、そのメソッドの本体が Bean クラスに追加され、そのメソッドのシグニチャがリモートインタフェースに追加されます。

メソッドを修正する必要がある場合は、そのメソッドのプロパティシートを使用するのが最適です。プロパティシートを呼び出すには、論理ノードから該当するメソッドを選択して右クリックし、「プロパティ」を選択します。プロパティシートには、メソッドの構成要素が、そのメソッドの継承元のクラスのタブに表示されます。プロパティシートで行なった変更は、検証され、適切なクラスに適切な形式で伝達されます。

この手順に従うと、J2EE 標準に準拠した Bean を作成することができます。

注意 - 論理ノード以外の場所で、すなわちリモートインタフェースノード、Bean クラスノード、ホームインタフェースノードの内部で変更を行なった場合も、EJB ビルダーは変更内容を伝達しようとします。ただし、場合によっては、Sun の J2EE の仕様に合わせてコードを手作業で編集する必要があります。次の節の例を参照してください。

ソースエディタを使用したセッション Bean の修正

IDE のソースエディタでコードを記述すると、セッション Bean のあらゆる要素を作成または修正することができます。ただし、EJB ビルダーの支援機能を活用できないと、結果に矛盾が生じる可能性があります。EJB ビルダーは、Bean 提供者があるクラスで行なった変更を、ほかのクラスに適切に反映させようとします。ただし、Bean 提供者の意図を正しく解釈できるとは限らないため、必要な変更が適用されない場合があります。そのため、クラスのコードを直接変更すると、セッション Bean が不正になり、手作業による修正が必要になることがあります。

いくつかの例を次に示します。

- ソースエディタで Bean のホームインタフェースやリモートインタフェースを開き、新しいメソッドのコードを追加した場合。

  メソッドが有効であるためには、そのメソッドが名前を持ち (生成メソッドの場合)、適切な型の戻り値を返し、適切な例外をスローしなければなりません。新しいメソッドが有効な場合は、そのメソッドが Bean クラスに自動的に追加されます。新しいメソッドが不正な場合は、そのメソッドはコードを作成した状態のままになり、インタフェースにしか含まれません。

  メソッドが不正なことを後から検出し、修正することができます。ただし、EJB ビルダーは、修正後のメソッドを Bean クラスに追加できるとは限りません。その場合は、このメソッドを手作業で追加する必要があります。手作業で追加するまでは、エクスプローラでこのメソッドのノードに赤い×印のエラーバッジマークが付きます (55 ページの「IDE のエラー情報」を参照してください)。

  変更はほかのクラスに次のように伝達されます。

  - ホームインタフェースに生成メソッドを追加すると、EJB ビルダーはそれに対応する `ejbCreate` メソッドを自動的に Bean クラスに追加します。
  - Bean クラスに `ejbCreate` メソッドを追加すると、EJB ビルダーはそれに対応する生成メソッドを自動的にホームインタフェースに追加します。

- ソースエディタで Bean クラスを開き、ビジネスメソッドを追加した場合。

  EJB ビルダーは、コードを可能な限り検証します。ただし、追加したメソッドを Bean クラスのヘルパーメソッドやユーティリティメソッドとして使用することはまずないので、このメソッドはリモートインタフェースには伝達されません。

- 適切な例外を使用するビジネスメソッドを Bean のリモートインタフェースに追加した場合。

このメソッドが Bean クラスに自動的に伝達されます。

- ソースエディタを使用して、Bean のホームインタフェースの生成メソッドや、リモートインタフェースのビジネスメソッドを修正した場合。

  変更内容が Bean クラスに伝達されます。

- Bean クラスの `ejbCreate` メソッドを修正した場合。

  EJB ビルダーはコードを可能な限り検証しますが、変更内容はホームインタフェースには伝達されません (Java のインタフェースとクラスとの関係は、IDE 全体にわたって同様に取り扱われます)。

- Bean のホームインタフェースのメソッドを修正し、必要な例外を取り除くなどして、そのメソッドが無効になった場合。

  EJB ビルダーは、コードを検証し、エラー情報を提供します。変更内容は Bean クラスには伝達されません。

- 新しいメソッドに `ejbCreate` 以外の名前を付けた場合、または、これらのメソッドを修正した場合。

  EJB ビルダーは宣言の構文が正しいかどうか、戻り値とパラメータの型が、特定可能な有効な Java クラスかどうかを検証します。

IDE のエラー情報

J2EE の仕様に従っていないコードを作成すると、エクスプローラ上のノードのアイコンに警告バッジマークやエラーバッジマークが付きます。該当するノードを選択して右クリックし、「エラー情報」または「EJB の検査」を選択すると、問題についての情報を参照することができます。

論理ノードに、この黄色い三角形の形をした警告バッジマークが表示された場合は、Bean やそのクラスで検証エラーが発生しています。論理ノードを展開し、どこで問題が発生しているかを確認してください。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 クラスに含まれていなかったり、クラスの Java スーパークラスが適切ではない可能性があります。Bean をコンパイルできたとしても、問題が発生します。

論理ノードに、この赤い×印のエラーバッジマークが表示された場合は、Bean やそのクラスで重大な問題が発生しています。たとえば、クラスそのものが存在していない可能性があります。問題を解消しない限り、Bean を正常に実行できません。

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 3-4 のようなツリーが表示されます。

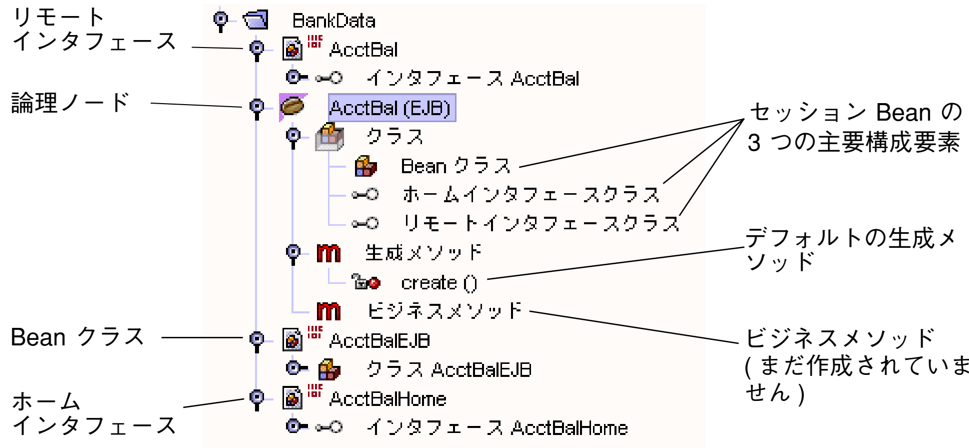


図 3-4　典型的なセッション Bean の詳細表示

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、59 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 のインスタンスが休止される前に、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次のメソッドを 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こと (ただし、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。

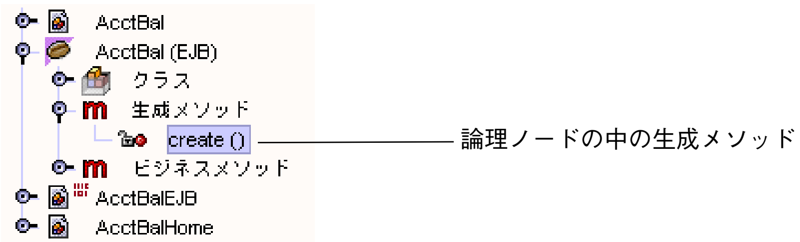
- 生成メソッドの完成
- ライフサイクルメソッドの完成
- ビジネスメソッドの追加
- トランザクションのコーディング

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()` を選択し、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;
    }
    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、「新規生成メソッド」を選択します。

   「新規生成メソッド」ダイアログボックスが表示されます。

2. 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」を選択して右クリックし、「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 のインスタンスが休止された場合に、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();
            con = null;
        } catch (Exception ex) {
            throw new EJBException("ejbPassivate Exception: " +
               ex.getMessage());
        }
    }
```

- `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、クライアントのためにビジネス処理を実行する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、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、「新規ビジネスメソッド」を選択します。

   「新規ビジネスメソッド」ダイアログボックスが表示されます。

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

	ステートレス	ステートフル
BMT	複数のメソッドにまたがったトランザクションは作成できない	同じセッション Bean の 1 つ以上のメソッドにまたがったトランザクションを作成することができる
CMT	複数のメソッドにまたがったトランザクションを作成することができる。ただし、それぞれのメソッドは別々のセッション Bean に含まれていなければならない	同じセッション Bean の 1 つ以上のメソッドにまたがったトランザクションを作成することができる

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、セッション Bean ウィザードの最初の区画で、次のように設定します。

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 5 章を参照してください。

また、付録 B では、作成した Enterprise 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

Enterprise 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。Enterprise Bean の基本要素の作成は、特に Forte for Java 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 Specification』を参照してください。

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

第4章

エンティティ Bean のプログラミング

Forte for Java IDE の EJB ビルダーを使用して、エンティティ Bean をプログラミングすることができます。エンティティ Bean は、Enterprise JavaBeans アプリケーションでデータを表現するのに必要です。この章では、コンテナ管理による持続性 (CMP) や Bean 管理による持続性 (BMP) を使用するエンティティ Bean の作成方法および操作方法を説明します。この両方の種類のエンティティ Bean のプログラミングを取り上げています。

IDE のウィザードを使用すると、Enterprise JavaBeans コンポーネント (Enterprise Bean) に必要なクラス、すなわちリモートインタフェース、ホームインタフェース、Bean クラス、および必要な場合は主キークラスを簡単に作成できます。必要な作業の多くが自動で行われます。

エンティティ 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できます。Forte for Java 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。詳細については、「はじめに」に記載した文献、または Enterprise 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。

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、Enterprise Bean を整合性のとれた方法で簡単に作成できます。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

ウィンドウで「ファイル」>「新規」を選択します。表示されるテンプレート選択ダイアログのテンプレートリストに「EJB」が含まれていれば、EJB ビルダーを使用できます。

Forte for Java IDE では、いくつかの方法でエンティティ 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、サポート機能をもっとも広範に活用し、通常はもっとも早く Bean を完成させることができます。この章に記載している手法では、整合性があり、J2EE 標準に従った Bean を作成するために、IDE の機能を最大限に活用しています。

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を使用すると、エンティティ Bean の枠組みができあがります。必要な 3 つまたは 4 つのクラスと論理ノードが作成され、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、これらのクラスのうち、ホームインタフェースとリモートインタフェース用の宣言を生成します。また、生成される Bean クラスには、必要なメソッドと、Bean 提供者が指定した持続フィールドが組み込まれます。Bean 提供者は、これらの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、エンティティ 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。「メソッドの追加」メニュー項目を使用するか、ソースファイルにメソッドの宣言を直接入力して、メソッドを追加することができます。

- **エンティティ Bean の配備記述子に値を設定します。** 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、論理ノードから呼び出すことができます。

エンティティ Bean の論理ノードから、Bean を検証したり、Bean 用のテストアプリケーションを作成したりできます。

CMP Bean と BMP Bean の比較

エンティティ Bean を作成するには、まず CMP と BMP のどちらを使用すればよいかを検討する必要があります。IDE の EJB ビルダーは、どちらの種類のエンティティ Bean にも対応していますが、これらの種類ごとに作成手順が異なります。CMP と BMP の詳細については、第 2 章を参照してください。表 4-1 に、設計上の検討項目をいくつか示します。

表 4-1　CMP Bean と BMP Bean の選択

項目	CMP	BMP
データベースとの関係	CMP Bean とデータベースとの関係をコンテナが管理する	BMP Bean とデータベースとの関係を BMP Bean 自身が管理する
持続性	コンテナが、アプリケーション中のすべての CMP Bean のデータベースアクセスを管理する。それぞれの CMP Bean は、特定のデータストアに依存しない	BMP Bean の中に、特定のデータベースに接続するコードが含まれている。さらに、持続フィールドを含んでいる BMP Bean には、必要なデータベースの呼び出しも含まれている。SQL をすべて手作業で追加する必要がある。持続性に対応していない EJB コンテナを使用する場合は、BMP Bean を使用する必要がある
プロセス	CMP Bean の基本構造 (デフォルトクラス) を簡単に、すばやく作成できる。コーディングの手間があまりかからない	BMP Bean では、より多くのコーディングが必要。JDBC プログラミングについての知識が必要
設計スコープ	それぞれの CMP Bean は、1 つの表しか表現できない	1 つの BMP Bean で 1 つ以上の表を表現することができる
機能と柔軟性	CMP Bean はコンテナにデータベースアクセスを実行させる。同じ Bean をさまざまなデータベース環境に配備することができる	BMP Bean の中でデータベースアクセスを手作業でプログラミングする。それぞれの BMP Bean は、プログラミング時に想定した環境でしか機能しない

ここからは、各種類のエンティティ Bean の作成方法と、開発時に考慮する必要のある問題を説明します。最初に「CMP エンティティ Bean の定義」を、100 ページからは「BMP エンティティ Bean の作成」を取り上げます。

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 Bean に最低限必要な 3 つのクラス (リモートインタフェース、ホームインタフェース、および Bean クラス) の作成作業の大半が自動化されます。複合主キーを指定した場合は、主キークラスも自動的に作成されます。CMP Bean を定義するには、次の手順に従います。

1. Bean の収容先の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、デフォルトの CMP Bean を作成します。
3. Bean のコードに生成メソッド、ビジネスメソッド、および検索メソッドを必要に応じて追加します。
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

エンティティ Bean を収容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。代わりに、エクスプローラの既存のディレクトリ (フォルダ) ノードに Bean を収容することもできます。

EJB ビルダーのウィザードの起動

1. 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」区画から、CMP Bean の収容先の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、「新規」>「EJB」>「エンティティ Bean」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「テンプレートから新規作成」と表示されます。左側のパネルに、現在の手順と、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。

デフォルトの CMP Bean の生成

EJB ビルダーの「エンティティ EJB 型」区画で、持続性とマッピングについての選択を行う必要があります。CMP Bean の場合の選択項目を図 4-1 に示します。

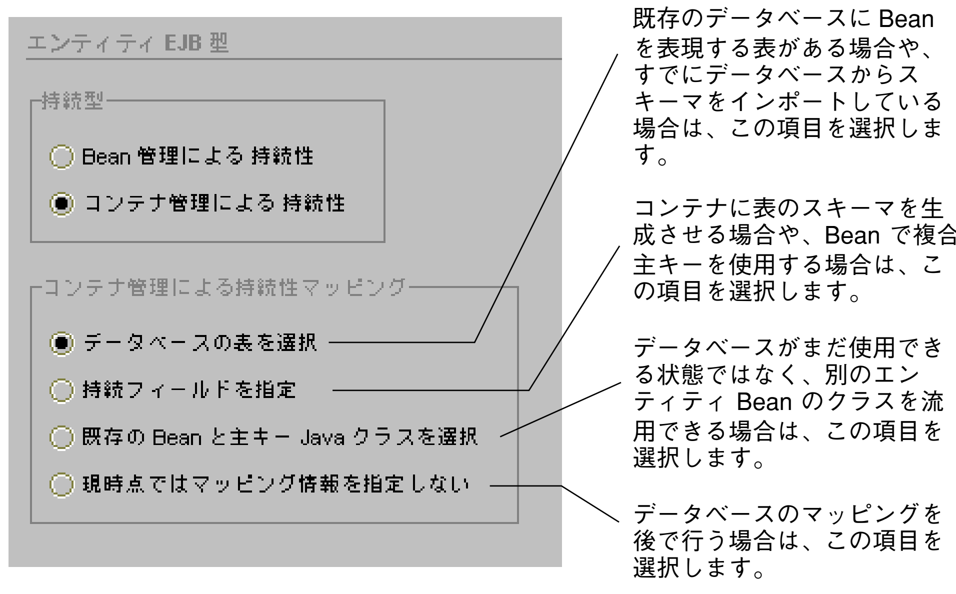


図 4-1　EJB ビルダーのウィザードでの CMP の選択項目

これらの選択項目の詳細については、次の節を参照してください。

- 「データベースの表を選択」。74 ページの「データベース表からの持続フィールドの指定」を参照してください。
- 「持続フィールドを指定」。79 ページの「持続フィールドの個別指定」を参照してください。
- 「既存の Bean と主キー Java クラスを選択」。80 ページの「既存の Bean クラスと主キークラスの使用」を参照してください。

- 「現時点ではマッピング情報を指定しない」。83 ページの「データベースマッピングの延期」を参照してください。

選択を行うと、選択した項目に応じた作業がウィザードに表示されます。

データベース表からの持続フィールドの指定

データベースから持続フィールドを指定する場合は、既存のデータベースや既存の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) が使用できる状態になっていなければなりません。詳細については、78 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 ビルダーのウィザードは、データベースの表に含まれている列 (「データベース接続からデータベースの表を選択」を選択した場合)、またはスキーマに含まれている列 (「データベーススキーマオブジェクトからデータベースの表を選択」を選択した場合) をマッピングし、エンティティ Bean の持続フィールドを作成します。どちらの項目を選択しても、最終的なエンティティ Bean は同じになります。

EJB コンテナによって、列とフィールドとのマッピングの取り扱い方法が異なります。詳細については、使用するコンテナのマニュアルを参照してください。

データベース接続からデータベース表を選択する

Bean 提供者にデータベースに直接アクセスする権限が与えられていて、ほかのデータベースユーザーとのアクセスの競合が問題にならない場合は、データベースに直接接続します。

ウィザードの「エンティティ EJB 型」区画から作業を開始します。

1. データベースのドライバファイルが、Forte for Java の `lib/ext` ディレクトリに含まれていることを確認します。

   IDE を起動する前に、このディレクトリにドライバファイルを配置しておく必要があります。このディレクトリにドライバファイルが含まれていないと、新しいスキーマを作成するときに適切なデータベースを選択できません。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組み込むことはできません。

2. 接続可能なデータベースを特定します。

「データベースの表を選択」をクリックし、「次へ」をクリックします。表示される新しい区画で、「データベース接続からデータベースの表を選択」をクリックします。

ウィザードのツリー表示に、エンティティ Bean から接続可能なデータベースが表示されます。

3. 接続先のデータベースを選択します。

- インストール済みのデータベースとの接続がまだ定義されていない場合は、「接続を追加」をクリックします。表示される「データベースの新規接続」ダイアログボックスで、接続に必要な情報を指定し、必要に応じてユーザー名とパスワードを入力します。「了解」をクリックすると、接続を使用できるようになります。
- インストール済みのデータベースとの接続がすでに定義されていて、その接続がまだアクティブになっていない場合は、該当するデータベースを選択し、「データベースに接続」をクリックします。データベースノードのアイコンが壊れたアイコンから完全なアイコンに切り替わり、表ノードを参照できるようになります。
- データベースとの接続がすでに定義されていて、その接続がすでにアクティブになっている場合 (同じ接続を使用して別のエンティティ Bean を作成する場合など) は、壊れたアイコンではなく、完全なアイコンが表示されています。その場合は、単に該当するデータベースを選択します。

4. 選択したデータベースの階層をたどり、Bean にマップする表のノードを表示します。

使用する表を選択し、「次へ」をクリックすると、データベース表の列と、エンティティ Bean に作成されるフィールドの対応表が表示されます。データベース表のそれぞれの列が、CMP エンティティ Bean の持続フィールドにマップされます。

5. Java フィールドの名前と型を確認し、必要に応じて変更します。

Java フィールドには、デフォルトの名前と型が割り当てられます。これらの名前と型を必要に応じて変更できます。

6. 「次へ」をクリックします。

ウィザードに「EJB コンポーネント」区画が表示され、エンティティ Bean の構成要素が一覧表示されます。

エンティティ Bean には、そのエンティティ Bean で表現されるデータベース表の名前が自動的に付けられます。必要であれば、この名前を変更することができます。変更した名前は、下にある「関連するオブジェクト」のフィールドに自動的に反映されます。

「関連するオブジェクト」に表示されたクラスの情報を参照し、必要に応じて変更します。ただし、次のことに注意してください。

- **サーバーの要件**。EJB ビルダーのウィザードでは、エンティティ 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収容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収容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をサポートす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、およびリモートインタフェース) を別のエンティティ 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。

7. 「完了」をクリックします。

CMP エンティティ 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、83 ページの「CMP Bean のクラスの参照」を参照してください。

データベーススキーマオブジェクトからデータベース表を選択する

データベースへのアクセスが制限されていて、スキーマオブジェクトが使用可能な状態になっている場合は、スキーマから表を取得します (まずスキーマを作成する必要がある場合は、78 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)。

ウィザードの「エンティティ EJB 型」区画から作業を開始します。

1. Bean にマッピングするデータベーススキーマを特定します。

   「データベースの表を選択」をクリックし、「次へ」をクリックします。表示される新しい区画で、「データベーススキーマオブジェクトからデータベースの表を選択」をクリックします。

   ウィザードの区画に、エクスプローラのファイルシステム区画でマウントされているディレクトリが表示されます。

2. エンティティ Bean の表を含んでいるデータベーススキーマを検出します。

   選択したスキーマの階層をたどり、Bean にマップする表のノードを表示します。使用する表を選択します。

3. 「次へ」をクリックします。

   データベース表の列と、エンティティ Bean に作成されるフィールドの対応表が表示されます。

4. データベース列と Java フィールドの対応表をチェックします。

   フィールドの名前と型を必要に応じて変更できます。

   SQL の型と Java の型のマッピングについては、『Getting Started with the JDBC API』の第 8 章「Mapping SQL and Java Types」を参照してください。この文書は、`http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html` に含まれています。

5. 「次へ」をクリックします。

   「EJB コンポーネント」区画が表示され、エンティティ Bean の基盤を構成するクラスが一覧表示されます。

   エンティティ Bean には、そのエンティティ Bean で表現されるデータベース表の名前が自動的に付けられます。必要であれば、この名前を変更できます。変更した名前は、下にある「関連するオブジェクト」のフィールドに自動的に反映されます。

   「関連するオブジェクト」に表示されたクラスの情報を参照し、必要に応じて変更します。ただし、次のことに注意してください。

   - **サーバーの要件**。EJB ビルダーのウィザードでは、エンティティ 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収容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収容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をサポートす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、およびリモートインタフェース) を別のエンティティ 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。

6. 「完了」をクリックします。

   CMP エンティティ 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、83 ページの「CMP Bean のクラスの参照」を参照してください。

データベーススキーマの収集

データベースに直接接続する代わりに、データベーススキーマを使用することができます。まだスキーマを作成していない場合は、IDE の「データベーススキーマ」ウィザードを使用して、次の手順でスキーマを作成できます。

1. IDE を起動する前に、データベースのドライバファイルが、Forte for Java の `lib/ext` ディレクトリに含まれていることを確認します。

   このディレクトリにドライバファイルが含まれていないと、新しいスキーマを作成するときに適切なデータベースを選択できません。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組み込むことはできません。

2. Forte for Java IDE で、次のいずれかの方法で「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、右クリックし、「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。

3. ウィザードの指示に従って必要な情報を指定します。

   スキーマに収集する表やビューを指定しなかった場合は、参照されている表がすべてスキーマに組み込まれます。

   指定を終えると、収集された表とビューの数が進捗バーに表示されます。

持続フィールドの個別指定

データベースがまだ作成されていない場合や、Bean 提供者にデータベースへのアクセス権がまだ与えられていない場合や、データベースの場所が不明の場合は、EJB ビルダーの「エンティティ EJB 型」区画で「持続フィールドの指定」を選択します。さらに、Enterprise Bean を組み込んだアプリケーションを配備したときに、アプリケーションサーバーにデータベースを作成させる場合も、「持続フィールドの指定」を選択することができます。

CMP Bean のコンテナがデータベースアクセスを行うためには、Bean がデータベースにマップされている必要があります。ただし、アセンブルや配備を行うまでは、マップが完了していなくてもかまいません。「持続フィールドの指定」を選択すると、Bean 提供者が指定したフィールドが、配備記述子に持続フィールドとして記録されます。この配備記述子を後ほど使用して、どのフィールドをデータベーススキーマにマップするかをコンテナに通知することができます。実際のマッピングは、J2EE アプリケーションを配備する直前に行います。

IDE では、エンティティ Bean の Java フィールド名を指定して、Bean の接続を設定することができます。後で配備の準備をするときに、残りのデータベース接続情報を指定することができます。

ウィザードの「エンティティ EJB 型」区画で「持続フィールドの指定」を選択し、「次へ」をクリックすると、「エンティティ EJB 持続フィールド」区画が表示されます。

1. 「追加」をクリックし、持続フィールドを 1 つずつ指定します。
   - 少なくとも 1 つの主キーフィールドを指定する必要があります。
   - コンボボックスに含まれていない型を指定する必要がある場合は、`java.lang.Integer` のように、その型を完全修飾パス名で入力することができます。

2. 「次へ」をクリックし、Bean 名を指定します。

「EJB コンポーネント」区画が表示されます。デフォルトの Bean 名として「Entity」が表示され、さらにパッケージ名と Bean の関連オブジェクトの名前が表示されます。指定した主キークラスは、「主キークラス」フィールドに表示されます。

Bean 名を変更すると、その名前が「関連するオブジェクト」のフィールドに自動的に反映されます。CMP Bean では、この区画の Bean 名以外のフィールドは変更しないでください。関連するオブジェクトには、デフォルトで名前が付けられます。これらのフィールドを変更する必要がある場合は、76 ページの注意事項に従ってください。

3. 「完了」をクリックします。

CMP エンティティ 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、83 ページの「CMP Bean のクラスの参照」を参照してください。

既存の Bean クラスと主キークラスの使用

既存の Bean のクラスを、新しく作成するエンティティ Bean で再利用したい場合は、IDE を使用して該当するクラスを簡単に検出し、エンティティ Bean に取り込むことができます。

ウィザードの「エンティティ EJB 型」区画から作業を開始します。「既存の Bean と主キー Java クラスを選択」を選択し、「次へ」をクリックすると、「エンティティ EJB Bean と主キークラス」区画が表示されます。

1. 適切なチェックボックスを選択します。または、「クラスの選択」をクリックしてファイル選択用ダイアログを表示します。

「既存の Bean クラスの使用」チェックボックスまたは「既存の主キークラスの使用」チェックボックスを選択することができます。新しいエンティティ Bean の Bean クラスと主キークラスの両方を、既存の Bean から取り込みたい場合は、両方のチェックボックスを選択します (「エンティティ EJB Bean と主キークラス」区画では、両方のマップを同時に定義できます)。

2. ファイル選択用ダイアログで、実際に使用するクラスを選択し、「了解」をクリックします。

   「選択されたクラスの使用を確認」ダイアログボックスが表示され、検出された検証エラーが一覧表示されます。

   理論上は、ファイルシステムに含まれているものであれば、どの Bean クラスまたは主キークラスでも選択できます。ただし、新しく作成する Bean のインタフェースやその他のクラスとの併用に適したクラスを選択した方が便利です。IDE は、検証の際に、Bean 提供者が選択した Bean クラスを新しく作成するエンティティ Bean と比較し、検出した不整合についての詳細なメッセージを表示します (後でエンティティ Bean を完成させるときに、これらの不整合に対処する必要があります)。

   検証に多少の時間がかかることがあります。検出されるエラーの例を次に示します。

   - 取り込み元のクラスに、必要なインタフェース (`javax.ejb.EntityBean`) が実装されていない。
   - 取り込み元のクラスで、エンティティ Bean に必要なメソッドが隠蔽されている。
   - 取り込み元のクラスに、エンティティ Bean では不正なメソッドが継承されている。
   - 取り込み元のクラスに、次の条件に適合するフィールドが含まれていない。
     - `public`、`static`、または `final` として宣言されている。
     - `primitive` または `serializable` として宣言されている。または `javax.ejb.EJBHome` か `javax.ejb.EJBRemote` のインスタンスになっている。

3. 「選択されたクラスの使用を確認」ダイアログボックスを参照します。特に、表示された検証エラーに注意します。

   別のクラスを選択したり、後ほどソースエディタでクラスを編集したりできます。

4. 選択したクラスを確定します。

   - 選択したクラスを使用する場合は、「このクラスを使用」をクリックします。
   - ファイル選択ダイアログを再表示し、クラスを選択しなおす場合は、「別のクラスを選択」をクリックします。

5. 「次へ」をクリックします。

   「エンティティ EJB 持続フィールド」区画が表示されます。

6. エンティティ Bean の持続フィールドを確認し、「次へ」をクリックします。

「エンティティ EJB 持続フィールド」区画に次の情報が表示されます。

- 既存の Bean クラスと既存の主キークラスの両方を選択した場合は、次の 2 つの表が表示されます。
  - 「主キー Public フィールド」。この最初の表には、取り込み元の Bean クラスから新しいエンティティ Bean に取り込まれた主キー public フィールドが表示されます。通常は、これらのフィールドの名前と型を変更する必要はありません。
  - 「追加 Public フィールド」。この 2 番目の表には、取り込み元の Bean クラスから新しいエンティティ Bean に取り込まれたその他のフィールドが表示されます。チェックボックスを使用して、これらのフィールドを持続フィールドにするかどうかを指定します。
- 既存の Bean クラスだけを選択した場合は、「Public フィールド」という表が表示されます。

  この表には、取り込み元の Bean クラスから取り込まれたフィールドがすべて表示されます。チェックボックスを使用して、これらのフィールドを持続フィールドにするかどうか、主キーにするかどうかを指定します。
- 既存の主キークラスだけを選択した場合は、「主キー Public フィールド」という表が表示されます。必要であれば、この表に表示されたフィールドの名前と型を変更することができます。

「次へ」をクリックします。「EJB コンポーネント」区画が表示されます。

7. エンティティ Bean の名前を指定し、クラスのリストを確認し、「完了」をクリックします。

関連するオブジェクトには自動的に名前が付けられます。これらのフィールドを変更する必要がある場合は、76 ページの注意事項に従ってください。

エンティティ Bean の名前を指定し、パッケージ名と関連するオブジェクトの名前を確認し、「完了」をクリックします。

これで、CMP エンティティ 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、83 ページの「CMP Bean のクラスの参照」を参照してください。

データベースマッピングの延期

仮のエンティティ Bean を作成しておき、後でマッピングを行う場合は、ウィザードの「エンティティ EJB 型」区画で「現時点ではマッピング情報を指定しない」を選択します。「次へ」をクリックすると、「EJB コンポーネント」区画が表示されます。

- エンティティ Bean の名前を指定し、パッケージ名と関連するオブジェクトの名前を確認し、「完了」をクリックします。

CMP エンティティ 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、83 ページの「CMP Bean のクラスの参照」を参照してください。

次のように、1 つのフィールドだけを含んだエンティティ Bean が作成されます。

このエンティティ Bean には、`java.lang.String` 型のデフォルトの主キークラスが 1 つだけ含まれています。

注意 - エンティティ Bean は、明示的に保存しない限り保存されません。

CMP Bean のクラスの参照

EJB ビルダーのウィザードは、エンティティ 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 CMP Bean は、エクスプローラの「ファイルシステム」区画に図 4-2 のように表示されます。

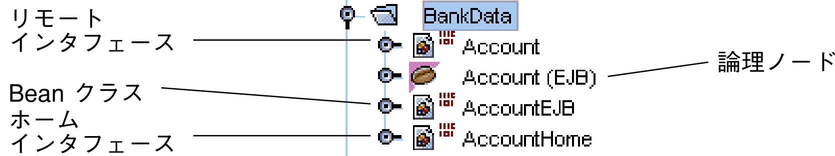
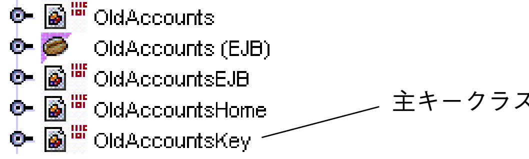


図 4-2　典型的なエンティティ Bean のデフォルトクラス

表示される 4 つのノードのうち、クラスアイコンが付いた 3 つのノードは実際のクラスを表しています。Bean アイコンが付いた残りの 1 つのノードは論理ノードです。編集作業は、すべて論理ノードで行います。

- リモートインタフェースは、javax.ejb.EJBObject のサブクラスです。このクラスは、エンティティ 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.EntityBean インタフェースを実装したクラスです。このクラスには、エンティティ 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 のサブクラスです。このクラスは、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。
- 論理ノードは、Enterprise 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。

主キークラスを作成した場合は (Bean に複合主キーが含まれている場合など)、次のように 5 つ目のノードが表示されます。



論理ノードとプロパティシートを使用した作業

EJB ビルダーのウィザードで、エンティティ Bean のデフォルトクラスを生成した後で、論理ノードに移動し、Bean の定義作業を続行します。Bean のクラスは、すべてこの単一のノードにまとめられるため、Bean を単一のオブジェクトであるかのように取り扱うことができます。論理ノードでは、それぞれの変更をどのクラスに適用するかを考慮することなく、Enterprise Bean を編集することができます。このノードを使用して Bean を編集すると、編集内容が Bean 全体に適切に伝達されます。この例を次に示します。

- 論理ノードの中の「生成メソッド」ノードに新しいメソッドを追加すると、そのメソッド (ejbCreate) の本体と関連するメソッド (エンティティ Bean に必要な ejbPostCreate) が Bean クラスに追加され、対応するメソッドシグニチャ (create) がホームインタフェースに追加されます。
- 論理ノードの中の「検索メソッド」ノードに新しいメソッドを追加すると、そのメソッドの名前を指定するように促され、そのメソッドのシグニチャがホームインタフェースに追加されます (BMP Bean では、シグニチャに加えて、対応する ejbFind メソッドが Bean クラスに追加されます)。

- 論理ノードの中の「ビジネスメソッド」ノードに新しいメソッドを追加すると、そのメソッドの本体が Bean クラスに追加され、そのメソッドのシグニチャがリモートインタフェースに追加されます。

メソッドのシグニチャを修正する必要がある場合は、そのメソッドのカスタマイザダイアログボックスやプロパティシートを使用するのが最適です。カスタマイザダイアログボックスやプロパティシートで行なった変更は、検証され、適切なクラスに適切な形式で伝達されます。

- カスタマイザダイアログボックス。論理ノードから該当する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。このダイアログボックスでは、メソッドのパラメータや例外を追加、編集、移動、または削除できます。

- プロパティシート。論理ノードから該当するメソッドを選択して右クリックし、「プロパティ」を選択します。プロパティシートには、メソッドの構成要素が、そのメソッドの継承元のクラスごとにタブに分かれて表示されます。プロパティシートの「パラメータ」フィールドを使用すると、メソッドのパラメータを追加、編集、移動、または削除できます。また、「追加例外」フィールドを使用すると、例外についても同様の作業を行うことができます。

この手順に従うと、J2EE 標準に準拠した Bean を作成することができます。

注意 - 論理ノード以外の場所で、すなわちリモートインタフェースノード、Bean クラスノード、ホームインタフェースノードの内部で変更を行なった場合も、EJB ビルダーは変更内容を伝達しようとします。ただし、場合によっては、Sun の J2EE の仕様に合わせてコードを手作業で編集する必要があります。次の節の例を参照してください。

ソースエディタを使用したエンティティ Bean の修正

IDE のソースエディタでコードを記述すると、エンティティ Bean のあらゆる要素を作成または修正できます。ただし、EJB ビルダーの支援機能を活用できないと、結果に矛盾が生じる可能性があります。EJB ビルダーは、Bean 提供者があるクラスで行なった変更を、同期をとるためにほかのクラスに適切に反映させようとします。ただし、Bean 提供者の意図を正しく解釈できるとは限らないため、必要な変更が適用されない場合があります。そのため、クラスのコードを直接変更すると、エンティティ Bean が不正になり、手作業による修正が必要になることがあります。

いくつかの例を次に示します。

- ソースエディタで Bean のホームインタフェースやリモートインタフェースを開き、新しいメソッドのコードを追加した場合。

  メソッドが有効であるためには、そのメソッドが名前を持ち (生成メソッドや検索メソッドの場合)、適切な型の戻り値を返し、適切な例外をスローしなければなりません。新しいメソッドが有効な場合は、そのメソッドが Bean クラスに自動的に追加されます。新しいメソッドが不正な場合は、そのメソッドはコードを作成した状態のままになり、インタフェースにしか含まれなくなります。

  メソッドが不正なことを後から検出し、修正することができます。ただし、EJB ビルダーは、修正後のメソッドを Bean クラスに追加できるとは限りません。その場合は、このメソッドを手作業で追加する必要があります。手作業で追加するまでは、このメソッドのノードに赤い×印のエラーバッジマークが付きます (87 ページの「IDE のエラー情報」を参照してください)。

  変更は他のクラスに次のように伝達されます。

  - ホームインタフェースに生成メソッドを追加すると、EJB ビルダーはそれに対応する `ejbCreate` メソッドを Bean クラスに追加します。
  - Bean クラスに `ejbCreate` メソッドを追加すると、EJB ビルダーはそれに対応する生成メソッドをホームインタフェースに追加します。

- ソースエディタで Bean クラスを開き、検索メソッドを追加した場合。

  EJB ビルダーはコードを検証します。追加したメソッドが有効であれば、そのメソッドがホーム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 クラスを開き、ビジネスメソッドを追加した場合。

  EJB ビルダーは、コードを可能な限り検証します。ただし、追加したメソッドを Bean クラスのヘルパーメソッドやユーティリティメソッドとして使用することはまずないので、このメソッドはリモートインタフェースには伝達されません。

- 適切な例外を使用するビジネスメソッドを Bean のリモートインタフェースに追加した場合。

  このメソッドが Bean クラスに自動的に伝達されます。

- ソースエディタを使用して、Bean のホームインタフェースの生成メソッドや、リモートインタフェースのビジネスメソッドを修正した場合。

  変更内容が Bean クラスに伝達されます。

- Bean クラスの `ejbCreate` メソッドを修正した場合。

  EJB ビルダーはコードを可能な限り検証しますが、変更内容はホームインタフェースには伝達されません (Java のインタフェースとクラスとの関係は、IDE 全体にわたって同様に取り扱われます)。

- Bean のホームインタフェースのメソッドを修正し、必要な例外を取り除くなどして、そのメソッドが無効になった場合。

  EJB ビルダーは、コードを検証し、エラー情報を提供します。変更内容は Bean クラスには伝達されません。

- 新しいメソッドに `ejbCreate` や `ejbFind***` (`***` は任意の文字列) 以外の名前を付けた場合、または、これらのメソッドを修正した場合。

  EJB ビルダーは宣言の構文が正しいかどうか、戻り値とパラメータの型が、特定可能な有効な Java クラスかどうかを検証します。

IDE のエラー情報

J2EE の仕様に従っていないコードを作成すると、エクスプローラ上のノードのアイコンに警告バッジマークやエラーバッジマークが付きます。該当するノードを選択して右クリックし、「エラー情報」または「EJB の検査」を選択すると、問題についての情報を参照することができます。

論理ノードに、この黄色い三角形の形をした警告バッジマークが表示された場合は、Bean やそのクラスで検証エラーが発生しています。論理ノードを展開し、どこで問題が発生しているかを確認してください。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 クラスに含まれていなかったり、クラスのスーパークラスが適切ではない可能性があります。Bean をコンパイルすることができたとしても、問題が発生します。

論理ノードに、この赤い×印のエラーバッジマークが表示された場合は、Bean やそのクラスで重大な問題が発生しています。たとえば、クラスそのものが存在しない可能性があります。問題を解消しないかぎり、Bean を正常に実行できません。

ノードの展開

エンティティ 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 4-3 のようなツリーが表示されます。

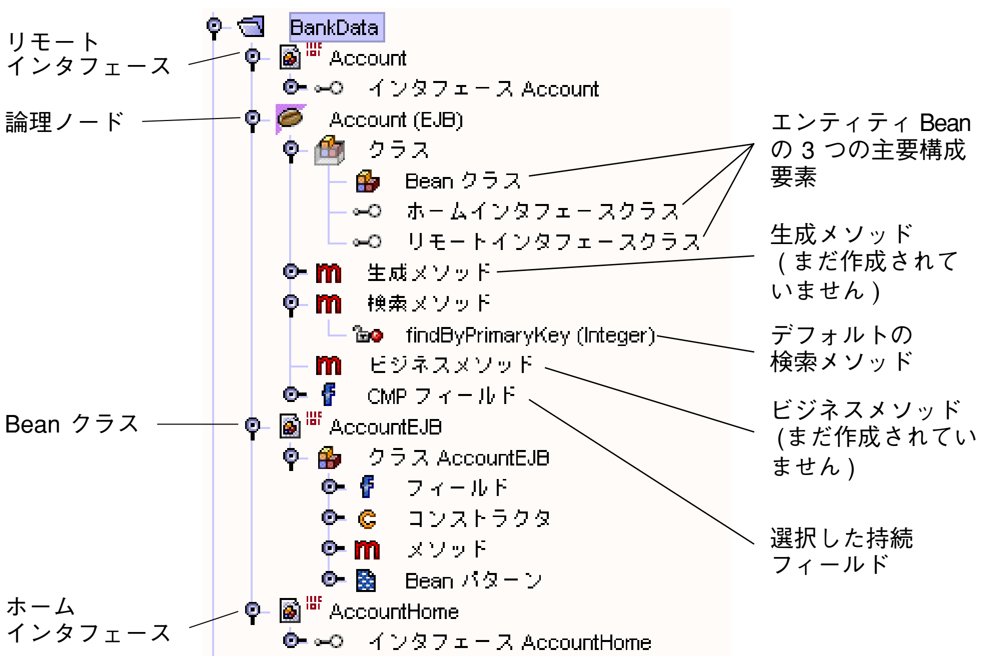


図 4-3　典型的な CMP Bean の詳細表示

主キークラスを新しく作成すると、エクスプローラの表示が図 4-4 のようになります。

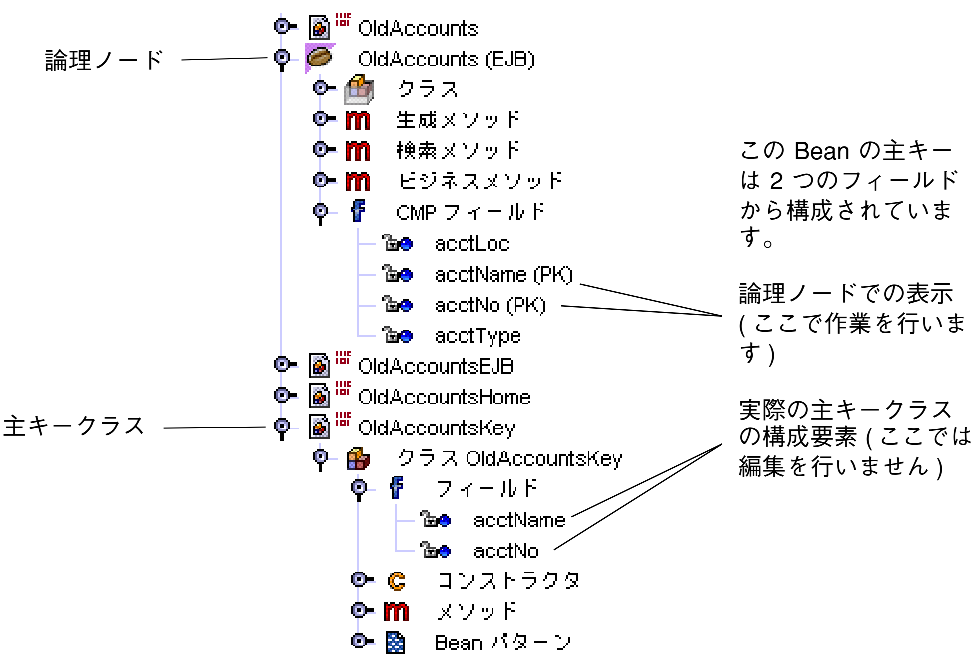


図 4-4　複合主キーを持つ典型的な CMP Bean の詳細表示

生成されたクラスの確認

CMP エンティティ Bean には、マップされたフィールドがすべて含まれています。また、CMP Bean と BMP Bean のどちらについても、特定のデフォルトメソッドが自動的に配置されます。

デフォルトの検索メソッド

『Enterprise JavaBeans™ Specification』で定められた仕様により、エンティティ Bean は主キーを基準にして特定できなければなりません。そのため、findByPrimaryKey メソッドのシグニチャが、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ます。CMP Bean では、このメソッドシグニチャを使用するだけで十分です。この findByPrimaryKey メソッドは、Bean のコンテナが実装します。

```
public DiscountCodeTbl findByPrimaryKey(String aKey)
    throws RemoteException, FinderException;
```

持続フィールド

IDE を使用して作成したエンティティ Bean には、CMP フィールドが組み込まれます。これらのフィールドを参照するには、論理ノードの Bean クラスを選択し、右クリックし、「開く」を選択します (または、論理ノードの Bean クラスをダブルクリックします)。ソースエディタが開き、CMP フィールドを含む Bean クラスのソースコードが表示されます。AccountEJB という CMP Bean の CMP フィールドは次のようになります。

```
/* Container managed fields */

public String id;
public BigDecimal balance;
```

CMP Bean の持続フィールドを 1 つも指定しなかった場合は、defaultField という CMP フィールドが 1 つだけ組み込まれ、このフィールドが自動的に主キーになります。

ウィザードを使用して CMP Bean を定義した後で、CMP フィールドをいつでも追加することができます。

Bean を作成した後で CMP フィールドの名前を変更したい場合は、論理ノードの中の 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、EJB ビルダーの指示に従って変更を行います。

CMP Bean の残りの持続性 (Bean をアセンブルし、配備するときに必要な実際の SQL 文) は、使用する EJB コンテナとアプリケーションサーバー用の配備記述子の中で指定します。詳細については、第 5 章の 122 ページの「CMP エンティティ Bean の J2EE RI プロパティの設定」を参照してください。

主キークラスと必須メソッド

EJB ビルダーのウィザードでは、CMP Bean の主キー (PK: Primary Key) フィールドを選択します。PK フィールドを選択しなかった場合は、デフォルトの PK フィールドが 1 つだけ割り当てられます。実際の主キーを作成した場合は、PK クラスが生成されます (それ以外の場合は、Bean には PK クラスは組み込まれません。後でデータベース表の主キーにマップする PK フィールドを作成する必要が生じたときに、PK クラスを作成する必要があります。96 ページの「主キーの新規作成」を参照してください)。

PK クラスには、Bean のインスタンスを一意に識別するのに必要な一連のデータが記録されます。Bean に PK フィールドが 1 つしか含まれていなければ、ウィザードはそのフィールドのクラスを Bean の PK クラスとして使用します。Bean に複合主キー (複数の持続フィールドから構成される主キー) が含まれている場合は、ウィザードはそれらのフィールドと名前および型が同一のフィールドを含んだ PK クラスを生成します。

さらに、新しい PK クラスが作成された場合は、コンテナに必要な次の 2 つのメソッドが挿入されます。

```
public boolean equals(java.lang.Object obj) {
}
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一のキー) どうしを比較します。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。

PK クラスには、java.io.Serializable インタフェースを実装する必要があります。java.rmi.Remote インタフェースを実装することはできません。

CMP Bean のライフサイクルメソッド

ウィザードは、CMP Bean や BMP 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() {
}
```

CMP Bean での、これらのメソッドの使用目的を表 4-2 に示します (BMP Bean との比較のため、105 ページの「BMP Bean のライフサイクルメソッド」と合わせて参照してください)。

表 4-2 CMP 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

表 4-2　CMP 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
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 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、insert 文を使用して、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
ejbStore	CMP 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、update 文を使用して、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 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 Bean の完成

CMP Bean を完成させるには、次の作業が必要です。

- 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。
- 必要に応じて別の主キーを追加するか、削除した主キーを置き換えます。Bean にまだ主キーがない場合は、主キーを新しく作成します。
- 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。
- 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、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。

- ダイアログボックス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、「新規生成メソッド」、「新規ビジネスメソッド」、「新規検索メソッド」のいずれかを選択します。定義したメソッドが適切なクラスに伝達されます。
- ソースエディタを使用してメソッドのコードを完成させます。

ダイアログボックスを使用せず、ソースエディタだけを使用して作業を行うこともできます。IDE のウィザードと GUI ツールを使用すると、作業が簡単になり、不整合を防止できるため、標準の Enterprise Bean をすばやく作成できます。ただし、ほとんどの場合は、ソースエディタで作成したコードが適切に伝達されます。詳細については、85 ページの「ソースエディタを使用したエンティティ Bean の修正」を参照してください。

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。いずれの場合も、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` メソッドと `ejbPostCreate` 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 Bean の `ejbCreate` メソッドは次の処理を行います。

- 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 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` メソッドは IDE によって自動的に追加されます。このメソッドが使用されることはほとんどありません。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 Enterprise 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。

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. 論理ノードを選択し、右クリックし、「新規生成メソッド」を選択します。

   「新規生成メソッド」ダイアログボックスが表示されます。このダイアログボックスで生成メソッドのパラメータを追加します。

2. 「追加」をクリックします。

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

   CMP 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 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>
   ```

4. 「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メソッドの本体 ejbCreate として、ホームインタフェースではメソッドの宣言 create として表示されます。さらに、Bean クラスにはメソッドの本体 ejbPostCreate 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 {
}
```

5. 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利用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 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 Bean クラスの生成メソッドの例

```
public CustomerSurveyKey ejbCreate(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.");
        }
        this.custname = custname;
        this.branchno = branchno;
        this.response = response;

        return null;

}
```

主キーの追加または置き換え

エンティティ Bean の PK クラスが削除されている場合や、PK 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、「...」をクリックします。

   「プロパティエディタ」ダイアログボックスが表示されます。

3. 既存のフィールドか定義済みのクラスを選択し、「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

PK 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には、まず EJB ビルダーのウィザードを使用し、仮のエンティティ Bean を新しく作成する必要があります。その後で、ウィザードを使用してエンティティ Bean に必要な PK フィールドを 1 つ以上指定します。

仮の Bean を作成した後で、次の作業を行います。

1. エクスプローラウィンドウで、仮の Bean の論理ノードを展開し、PK クラスのクラスノードを選択します。
2. 主キーを追加するエンティティ Bean の名前に合わせて、PK クラスの名前を変更します。

   たとえば、Bean 名が `Account` だとすると、クラス名を `AccountKey` に変更します。
3. ソースエディタで PK クラスを開き、コードに含まれているそれまでのクラス名を、すべて新しいクラス名に変更します。
4. PK クラスを保存します。

   コード表示領域を右クリックし、「保存」を選択します。
5. エクスプローラウィンドウで、主キーを追加するエンティティ Bean の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。
6. プロパティシートで、「主キークラス」プロパティを新しい PK クラス名に設定します。

   Bean の論理ノードに警告バッジマークやエラーバッジ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。
7. エクスプローラウィンドウで「CMP フィールド」ノードを展開し、デフォルトフィールドを削除します。
8. 「CMP フィールド」ノードを右クリックし、「新規 CMP フィールド」を選択します。

   適切な名前と、PK クラスと同じ型を指定して、新しいフィールドを作成します。「主キー」チェックボックスにチェックマークを付けます。
9. PK クラスを開いて編集します。

   それまでのフィールドの宣言を削除し、それまでのフィールド名を新しいフィールド名に置き換えます。
10. 必要に応じて主キーフィールドを追加します。

    96 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。

11. Bean の論理ノードに含まれている主キークラスノードを参照し、エラーを修正します。

    このノードにエラーバッジマークが付いている場合は、このノードを右クリックし、「エラー情報」を選択します。

外部キー

対応しているバージョンの『Enterprise JavaBeans™ Specification』には、CMP Bean 間の関係は定義されていないため、EJB ビルダーは現時点では外部キーを明示的にサポートしていません。外部キーとして実装された関係を維持したい場合や、データストアへの複数アクセスが必要な場合は、エンティティ Bean にそのためのコードを追加する必要があります。代わりに、透過的持続性 (Transparent Persistence) を使用することもできます。詳細については、付録 A を参照してください。

ビジネスメソッドの定義

CMP 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、「新規ビジネスメソッド」を選択します。

   「新規ビジネスメソッド」ダイアログボックスが表示されます。

2. メソッド名を入力し、パラメータと例外を指定し、「了解」をクリックします。

   または、メソッド名だけを指定して「了解」をクリックし、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、銀行の利用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 Bean のビジネスメソッドの例

```
public java.lang.String getResponse() {
    return response;
}
```

検索メソッドの追加

EJB ビルダーは、デフォルトの findByPrimaryKey メソッドを自動的に生成します。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。

1. 論理ノードを選択し、右クリックし、「新規検索メソッド」を選択します。
2. 「find」で始まるメソッド名を入力し、パラメータと例外を指定し、「了解」をクリックします。

   ソースエディタで検索メソッドを直接開くには、論理ノードの中の「検索メソッド」ノードに移動します。このノードから該当する検索メソッドを選択し、右クリックし、「開く」を選択します。

   ソースエディタに、ホームインタフェースクラスの検索メソッドが表示されます。

Account という Bean では、検索メソッドで次のような処理を行うことができます。

- 特定の口座のデータを保持している AccountEJB インスタンスを検出し、そのインスタンスのリモートオブジェクトを返します。その場合は、口座番号を基準にして口座を選択します。
- 残高がマイナスの AccountEJB インスタンスをすべて検出し、それらのインスタンスのリモートオブジェクトのコレクションを返します。その場合は、残高がマイナスの口座を選択します。

追加フィールドの定義

CMP Bean を作成した後で、CMP フィールドを追加することができます。

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

注 - ソースエディタで Bean クラスのフィールドを直接コーディングしないでください。配備記述子で、そのフィールドを持続フィールドとして特定できなくなります。

CMP Bean を作成した後の作業

作成した CMP Bean を、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 5 章を参照してください。

また、付録 B では、作成した Enterprise Bean の望ましい操作手順を説明しています。

ここからは、BMP エンティティ Bean の作成作業を説明します。

BMP エンティティ Bean の作成

EJB ビルダーのウィザードを使用すると、BMP Bean のデフォルトクラスの作成作業が自動化されます。ただし、デフォルトクラス (リモートインタフェース、ホームインタフェース、Bean クラス、および主キークラス) は自動的に作成されますが、BMP Bean とデータベースとの相互作用については、ウィザードは関知しません。そのため、デフォルトクラスの初期設定手順は非常に簡単です。BMP Bean を作成するには、次の手順に従います。

1. BMP Bean の収容先の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、デフォルトの BMP Bean を作成します。
3. 必要に応じて Bean に主キークラスを追加します。
4. Bean のコードに生成メソッド、ビジネスメソッド、および検索メソッドを必要に応じて追加します。
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。データベースのデータに影響するメソッドをすべて完成させます。

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

エンティティ Bean を収容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。代わりに、エクスプローラの既存のディレクトリ (フォルダ) ノードに Bean を収容することもできます。

EJB ビルダーのウィザードの起動

1. 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. エクスプローラで、BMP Bean の収容先のパッケージやファイルシステムを選択します。
3. 右クリックし、「新規」>「EJB」>「エンティティ Bean」を選択します。

   EJB ビルダーのウィザードが表示されます。

デフォルトの BMP Bean の作成

1. EJB ビルダーのウィザードで「Bean 管理による持続性」を選択し、「次へ」をクリックします。

   「EJB コンポーネント」区画が表示されます。
2. BMP Bean の名前を入力し、この区画のほかのフィールドの内容を確認します。

   必要に応じて次の作業を行うことができます。

   - パッケージの場所を変更することができます。
   - 「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作成することができます。

■ 表示されたクラスの「変更」ボタンをクリックして、スーパークラスを変更することができます。ただし、スーパークラスとして選択できるのは、適切なインタフェース (`EntityBean`、`EJBObject`、`EJBHome`、または `Object`) のサブクラスになっているものだけです。

注 - IDE が生成するコードによって、スーパークラスのメソッドがオーバーライドされる場合があります。スーパークラスを変更した場合は、生成されたコードをチェックし、必要に応じて修正してください。

これらのフィールドを変更する必要がある場合は、76 ページの注意事項に従ってください。

3. 「完了」をクリックします。

Bean のデフォルトクラスが生成されます。ここからは、これらのクラスを説明します。

BMP Bean のクラスの参照

EJB ビルダーのウィザードは、BMP Bean についてエンティティ Bean の必須クラスを自動的に生成し、これらのクラスどうしの関係を設定します。ただし、持続性ロジックは、すべて Bean 提供者がコーディングする必要があります。

典型的なデフォルトの BMP Bean は、エクスプローラの「ファイルシステム」区画に図 4-5 のように表示されます。

図 4-5　BMP エンティティ Bean のデフォルトクラス

CMP Bean の場合と同様に、 アイコンが付いた 3 つのノードは実際のクラスを表しています。 アイコンが付いた残りの 1 つのノードは論理ノードです。編集作業は、すべて論理ノードで行います。

Bean の各クラスについては、次のことに注意してください。

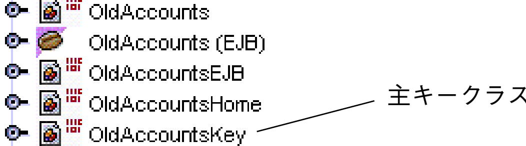
■ リモートインタフェースは、javax.ejb.EJBObject のサブクラスです。このクラスは、エンティティ Bean のビジネスメソッドのシグニチャを提供します。

■ Bean クラスは、javax.ejb.EntityBean インタフェースを実装したクラスです。このクラスには、エンティティ Bean の ejbCreate メソッドとビジネスメソッドが実装されます。BMP Bean の場合は、Bean の ejbFindByPrimaryKey メソッドや、ホームインタフェースで指定されたその他の検索メソッドの定義も、このクラスに含まれます。

■ ホームインタフェースは、javax.ejb.EJBHome のサブクラスです。このクラスは、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。

■ 論理ノードは、Enterprise 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。

主キークラスを作成した場合は、次のように 5 つ目のノードが表示されます。



BMP Bean の操作

EJB ビルダーのウィザードで、BMP Bean の必須クラスを生成した後で、論理ノードに移動し、Bean の定義作業を続行します。このノードで編集を行うと、編集内容が BMP Bean 全体に適切に伝達されます。このようにすると、J2EE 標準に準拠した Bean を作成することができます。

論理ノード以外の場所で、すなわちリモートインタフェースノード、Bean クラスノード、またはホームインタフェースノードの内部で変更を行なった場合も、EJB ビルダーは変更内容を伝達しようとします。ただし、場合によっては、サンの J2EE の仕様に合わせてコードを手作業で編集する必要があります。

詳細情報と例については、84 ページの「論理ノードとプロパティシートを使用した作業」、85 ページの「ソースエディタを使用したエンティティ Bean の修正」、および 87 ページの「IDE のエラー情報」を参照してください。

ノードの展開

BMP 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 4-6 のようなツリーが表示されます。

図 4-6 BMP Bean の詳細表示

主キークラスを作成すると、そのクラスが BMP Bean の 5 番目の主要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、エンティティ Bean にデフォルトのメソッドをいくつか追加します。

findbyPrimaryKey メソッド

メソッドシグニチャ findByPrimaryKey が BMP Bean のホームインタフェースに自動的に追加されます。

```
public ExistingCustomer findByPrimaryKey(String aKey)
    throws RemoteException, FinderException;
```

これは BMP Bean なので、このメソッドに対応する ejbFindByPrimaryKey メソッドが Bean クラスに追加されます。

```
public String ejbFindByPrimaryKey(String aKey) {
    return aKey;
}
```

BMP Bean のライフサイクルメソッド

ウィザードは、BMP 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() {
}
```

BMP Bean での、これらのメソッドの使用目的を表 4-3 に示します (CMP Bean との比較のため、91 ページの「CMP Bean のライフサイクルメソッド」と合わせて参照してください)。

表 4-3　BMP 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

表 4-3　BMP 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
`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
`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`	BMP では、このメソッドは `SQL` の `Select` 文を実行し、データソースから Bean のインスタンスにデータを読み込む。この処理は、Bean がアクティブになったときや、新しいトランザクションのコンテキストでエンティティが参照されたときに実行される。代わりに、データアクセスオブジェクト (DAO) といった別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` の `Update` 文を実行し、Bean の状態 (持続フィールドの現在値) をデータ記憶装置に保存する。この処理は、Bean が休止されたときや、トランザクションがコミットされたときに実行される。代わりに、DAO といった別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
`ejbRemove`	BMP では、このメソッドは `SQL` の `Delete` 文を実行し、データをデータ記憶装置から削除する。代わりに、DAO といった別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

BMP Bean の完成

BMP Bean を完成させるには、次の作業が必要です。

- 持続性ロジックをすべて追加します。

- BMP Bean で複合主キーを使用する場合は、主キークラスを追加します。
- 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。
- `findByPrimaryKey` 以外の検索メソッドが必要な場合は、それらを定義し、コードを記述します。
- `ejbRemove` メソッドに、レコードをデータベースから削除するコードを記述します。
- 必要なすべてのビジネスメソッドを定義し、そのコードを記述します。
- エンティティの状態をメモリーに保持するために `private` フィールドを追加し、これらのフィールドに値を格納できるようにします。

持続性ロジックの追加

BMP Bean からエンティティのデータストアを操作するには、データにアクセスし、持続フィールドを操作し、Bean インスタンスの変数とデータストアとの間でデータを転送するコードを記述する必要があります。ソースエディタを使用して、これらのコードを記述します。

主キークラスの追加

次の条件に該当する場合は、ソースエディタを使用して主キークラスを追加します。

- 複合主キー (複数のフィールドにマップされる主キー) を作成する。
- 主キーの型が `java.lang.String` ではない。
- 主キーをほかの機能 (キーをデータベースで使用する前に、そのキーの値が有効かどうかを判定する機能など) とともにラッピングする。

主キークラスは次の条件を満たしていなければなりません。

- 主キークラスはアクセス権を制御する修飾子 `public` を指定する必要があります。
- すべてのフィールドを `public` フィールドとして宣言する必要があります。
- `public` として宣言されたデフォルトコンストラクタが含まれていなければなりません。

- hashCode メソッドと equals メソッドを実装する必要があります。
- 主キークラスはシリアライズ可能でなければなりません。すなわち、java.io.Serializable インタフェースを実装する必要があります。
- java.rmi.Remote インタフェースは実装できません。

詳細については、96 ページの「主キーの追加または置き換え」、96 ページの「主キーの新規作成」、98 ページの「外部キー」を参照してください。

メソッドの追加

Bean の論理ノードに用意された GUI ツールを使用して、新しいメソッドの追加作業を開始します。ダイアログボックスでメソッド名を指定し、メソッドのシグニチャを定義すると、IDE によって新しいメソッドが適切なクラスに自動的に伝達されます。その後でソースエディタを使用し、メソッドのコードを完成させます。

エクスプローラの GUI ツールを使用すると、メソッドのスタブや本体を、どのクラスまたはインタフェースに配置すればよいかを意識することなく、メソッドを追加することができます。ソースエディタを使用して、適切に伝達されたメソッドシグニチャを修正すると、IDE によって宣言の同期が自動的にとられます。

ダイアログボックスを使用せず、ソースエディタだけを使用して作業を行うこともできます。IDE のウィザードと GUI ツールを使用すると、作業が簡単になり、不整合を防止できるため、標準の Enterprise Bean をすばやく作成することができます。ただし、ほとんどの場合は、ソースエディタで作成したコードが適切に伝達されます。詳細については、85 ページの「ソースエディタを使用したエンティティ Bean の修正」を参照してください。

生成メソッドの定義

BMP 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 メソッドと ejbPostCreate 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、BMP Bean の ejbCreate メソッドは次の処理を行います。

1. クライアントから渡された引数を検証します。
2. インスタンスの変数を初期化します。

3. SQL の Insert 文を実行します (代わりに、DAO といった別のクラスを呼び出して、データをデータストアに挿入することもできます)。

4. 主キーを返します。

BMP Bean では、SQL の Insert 文を生成、実行するコードを記述する必要があります。

ejbPostCreate メソッドは IDE によって自動的に追加されます。このメソッドが使用されることはほとんどありません。このメソッドは、EJB オブジェクトについての情報を、その情報を参照する必要のあるほかの Enterprise 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。

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。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. 論理ノードを選択し、右クリックし、「新規生成メソッド」を選択します。

   「新規生成メソッド」ダイアログボックスが表示されます。このダイアログボックスで生成メソッドのパラメータを追加します。

2. 「追加」をクリックします。

3. 「メソッドのパラメータを入力」ダイアログボックスで、パラメータの型と名前、さらに例外を指定します。

   (CMP Bean と異なり、) BMP Bean クラスのメソッドシグニチャと、メソッドの本体のどちらも主キー型を返します。

4. 「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メソッドの本体 ejbCreate として、ホームインタフェースではメソッドの宣言 create として表示されます。さらに、Bean クラスにはメソッドの本体 ejbPostCreate も表示されます。すでにソースエディタを開いている場合は、コードの表示が直ちに更新されます。

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

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

5. ソースエディタを使用して、必要なコードを新しい生成メソッドに追加します。

検索メソッドの追加

EJB ビルダーは、デフォルトの検索メソッドを自動的に生成します。BMP 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`) の両方に配置されます。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。次の手順に従うと、新しく追加した検索メソッドがホームインタフェースと Bean クラスに自動的に伝達されます。

1. 論理ノードを選択し、右クリックし、「新規検索メソッド」を選択します。
2. 「`find`」で始まるメソッド名を入力し、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。

この作業を行なった後で、ソースエディタを使用し、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り出すには、JDBC コードを記述するか、その他のデータアクセス手段を使用する必要があります。

`select` メソッドでは、データベース表の主キーだけを取り出してください。これにより、`ejbFind` メソッドは、主キークラスのインスタンスか、主キーのコレクションを返します。EJB コンテナは、`ejbLoad` メソッドを呼び出して、インスタンスの値を読み込みます。

ソースエディタで検索メソッドを直接開くには、論理ノードの中の「検索メソッド」ノードに移動します。このノードから該当する検索メソッドを選択し、右クリックし、「開く」を選択します。

ソースエディタに、Bean クラスの検索メソッドが表示されます。`Account` という Bean では、検索メソッドで次のような処理を行うことができます。

- 特定の口座のデータを保持している `AccountEJB` インスタンスを検出し、そのインスタンスのリモートオブジェクトを返します。その場合は、口座番号を基準にして口座を選択します。

- 残高がマイナスの AccountEJB インスタンスをすべて検出し、それらのインスタンスのリモートオブジェクトのコレクションを返します。その場合は、残高がマイナスの口座を選択します。

ビジネスメソッドの定義

- 論理ノードの中の「ビジネスメソッド」を選択し、右クリックし、「新規ビジネスメソッド」を選択します。

「新規ビジネスメソッド」ダイアログボックスが表示されます。このダイアログボックスでメソッドのパラメータと例外を指定します。代わりに、メソッド名だけを指定して「了解」をクリックし、ソースエディタでコードを完成させることもできます。

通常、ビジネスメソッドでは持続フィールドの値を参照および変更し、データベースには直接アクセスしません。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ます。

サーバーによる違い

BMP Bean を使用するアプリケーションの実行時に、ejbStore メソッドや ejbLoad メソッドがアプリケーションサーバーから自動的に呼び出されるとは限りません。J2EE Reference Implementation アプリケーションサーバー (RI) では、これらのメソッドは自動的には呼び出されませんが、iPlanet アプリケーションサーバーでは、これらのメソッドが自動的に呼び出されます。使用しているアプリケーションサーバーについて調べ、必要なメソッドを BMP Bean に追加する必要があります。たとえば、RI で BMP Bean を使用する場合は、ejbLoad メソッドと ejbStore メソッドを手作業で呼び出すために、loadRow メソッドや storeRow メソッドを追加する必要があります。

詳細については、各サーバーのマニュアルを参照してください。

BMP Bean を作成した後の作業

作成した BMP Bean を、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 5 章を参照してください。

また、付録 B では、作成した Enterprise 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

Enterprise Bean は、アプリケーションの非常に高機能で、高い柔軟性を備えた構成要素になります。Enterprise Bean の基本要素を作成することは、Forte for Java 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 Specification』を参照してください。

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

第5章

プログラミング後の作業

Bean 提供者の主な役割は、Enterprise Bean を作成することです。この作業については、これまでの章で説明しましたが、アプリケーションのアセンブルと本稼働環境への配備を担当するプログラマに、作成した Enterprise Bean を渡す前に、次の 3 つの作業が残されています。

1. Bean の外部依存性と操作条件の情報を指定します (これらの情報にもっとも詳しいのは、通常は Bean 提供者です)。
2. アプリケーションの内部で連携して動作する必要のある Bean 群を 1 つにまとめた EJB モジュールを作成します。
3. Forte for Java IDE の自動テスト機能を使用して、Bean と EJB モジュールをテストします。

エンティティ Bean の配備記述子では、コンテナが使用するデータベース表の操作命令を指定することが重要です。これらの命令の指定方法は、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 Bean) を最初から作成したかどうか、または既存の表から作成したかどうかによって異なります。この章では、これらの命令の望ましい指定方法を説明しています。

アプリケーションのアセンブルとサーバーへの配備については、このマニュアルシリーズの別のマニュアル、『J2EE モジュールおよびアプリケーションのアセンブルと実行』で詳しく取り上げています。

配備記述子の指定

配備記述子は Enterprise Bean ごとに指定します。配備記述子とは、Bean をほかの Bean とともにアプリケーションにアセンブルし、特定の環境に配備するときに使用される命令群と、Bean のプロパティを記録した XML 形式のテキストファイルです。

それぞれの Enterprise Bean の配備記述子には、その Bean の構造、ほかの Bean との関係、およびその他の外部依存性についての情報が含まれています。Bean の記述子を変更すると、その Bean の動作を変更することができます。

Enterprise Bean の配備記述子を指定するには、まず EJB ビルダーウィザードを使用して、第 3 章で説明した Bean (セッション Bean) や、第 4 章で説明した Bean (エンティティ Bean) を作成します。これにより、基本的な記述子が自動的に作成されます。

Enterprise Bean を EJB モジュールに組み込むと、EJB モジュールの配備記述子が自動的に作成されます。この記述子ファイルには、次の情報が含まれています。

- それぞれの Bean の宣言的なメタ情報 (Bean のコードに含まれていない情報)。Bean の配備記述子から取り込まれる。
- Bean をアプリケーションに適合させる方法や、配備についての高レベルな情報。
- コンテナに対する、エンティティ Bean がアクセスするデータベース表の操作命令。

EJB モジュールの記述子を変更すると、コンポーネント Bean のソースコードを変更することなく、アプリケーションの動作を変更することができます。

配備記述子の内容は、該当する Bean や EJB モジュールのプロパティシートから操作することができます。

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、次の手順に従います。

- 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、「配備記述子を表示」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。

このファイルを直接編集しないでください。このファイルの内容を編集するには、EJB モジュールのプロパティシートを使用します。「プロパティ」ダイアログボックスを使用して、このファイルを正しい方法で編集すると、ファイル名と値 (エンティティ Bean の場合は、さらに表と列の名前) 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。ダイアログボックスで選択を行うと、Enterprise Bean の適切なクラスに、変更内容が自動的に反映されます。

プロパティシートを使用した配備記述子の編集

Enterprise Bean の記述子を追加、編集、または完成させるには、まず次の手順で Bean のプロパティシートを開きます。

- **Bean の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。**

  「プロパティ」ダイアログボックスが開き、少なくとも次の 3 つのタブが表示されます。

  - プロパティ
  - 参照
  - J2EE RI (Java™ 2 Platform, Enterprise Edition に付属している Reference Implementation 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 (iPlanet など) のタブも表示されます。

ここからは、上の 3 つのデフォルトタブについて説明します。

「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。

- エンティティ Bean の「プロパティ」タブには、読み取り専用のフィールドが 1 つ含まれています。このフィールド、「Bean 型」の内容は、セッション Bean では「ステートフル」か「ステートレス」になり、エンティティ Bean では「コンテナ管理による持続性」か「Bean 管理による持続性」になります。それ以外のフィールドはすべて編集することができます。また、セッション Bean の「プロパティ」タブでは、すべてのフィールドを編集することができます。

- 「ホームインターフェース」、「名前」、および「リモートインターフェース」の各フィールドの内容は、EJB ビルダーのウィザードで Bean を作成したときに設定されます。EJB ビルダーのウィザードは、Bean の各クラスに自動的に名前を付けます。Bean を作成するときに、Bean 提供者がこれらのフィールドに表示されたクラス名を変更することもできます。
  - 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、ウィザードでそのための作業を行います (Bean を作成し直し、現在の Bean のクラスのうち、再利用したいものを指定します)。
  - 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。
- エンティティ Bean の「プロパティ」タブには、「主キークラス」フィールドが表示されます。セッション Bean の「プロパティ」タブには、「トランザクション型」フィールドが表示されます。
- 「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、Enterprise 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。

「参照」タブ

Enterprise Bean のアセンブルと配備を行えるようにするには、このタブのフィールドを設定する必要があります。Enterprise 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。作成された配備記述子の情報の一部は、アセンブル担当者が後から変更できます。

「参照」タブの内容を次の図に示します。

図 5-1　エンティティ Bean 用の「プロパティ」ダイアログボックスの「参照」タブ

ここからは、このタブのフィールドと、その設定方法を説明します。

EJB 参照

このフィールドには、Bean から呼び出すメソッドが実装されているほかの Enterprise Bean の情報が含まれています。メソッドの実装先の Enterprise Bean 名を開発時に指定しておき、Bean を EJB モジュールにアセンブルするときに、その設定を必要に応じて変更できます。

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 どうしをリンクさせます。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 どうし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、次の手順に従います。

1. 「EJB 参照」フィールドをクリックし、「...」ボタンをクリックします。

   「EJB 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 参照」ダイアログボックスが表示されます。

3. 各フィールドを設定します。

次のフィールドは必須です。

- 「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、`OldAccounts` 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。

- 「型」。参照先の Bean の種類です。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。
- 「ホームインターフェース」。参照先の Bean のホームインタフェースです。
- 「リモートインターフェース」。参照先の Bean のリモートインタフェースです。

さらに、EJB モジュールをアプリケーションにアセンブルする担当者が理解しやすいように、必要に応じて次の 2 つのフィールドを指定することができます。

- 「説明」。参照先の Bean の使用目的や、なぜ現在の Bean から参照する必要があるのかを指定します。
- 「参照される EJB 名」。上のフィールドで指定したホームインタフェースやリモートインタフェースが実装されている Enterprise Bean 名です。「ブラウズ」をクリックし、Bean を選択すると、「ホームインターフェース」フィールドと「リモートインターフェース」フィールドに自動的に情報が入力されます。このフィールドのデータを上書きして、同じホームインタフェースまたはリモートインタフェースが実装されている別の Bean を指定したり、参照先を別の Bean に変更したりすることができます。

環境エントリ

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 の動作を変更することができます。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、「...」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。

   「追加　環境エントリ」ダイアログボックスが表示されます。

3. 各フィールドを設定します。

   次の 2 つのフィールドは必須です。

   - 「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る必要のある情報を指定します。
   - 「値」。初期値です。

リソースファクトリ参照

このフィールドには、Bean をデータ記憶装置などのリソースに接続するための情報が含まれています。このフィールドの情報は、Bean クラスのコードに含まれている JNDI 検索メソッドの呼び出し文の内容と一致していなければなりません。

Bean 提供者は、Bean にリソース接続ファクトリが必要かどうかを決定します。アセンブル担当者は、Bean 提供者が定義したリソースファクトリの参照に基づいて、特定のデータストアにアクセスする接続ファクトリを指定します。

リソースファクトリの参照を追加するには、Bean のアクセス先のリソースごとに次の手順に従います。

1. 「リソースファクトリ参照」フィールドをクリックし、「...」ボタンをクリックします。

   「リソースファクトリ参照」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」ダイアログボックスが表示されます。

3. 各フィールドを設定します。

   次のフィールドは必須です。

   - 「名前」。Bean クラスのコードに含まれている InitialContext.lookup メソッドで指定されている名前です。たとえば、myDatabase という JDBC リソースファクトリを使用する場合は、このメソッドのコードは次のようになります。

```
javax.naming.InitialContext myContext =
    new javax.naming.InitialContext();
javax.sql.DataSource mySource = (javax.sql.DataSource)
    myContext.lookup("java:comp/env/jdbc/myDataBase");
```

     その場合は、このフィールドで指定するリソース参照名は jdbc/myDataBase になります。

   - 「型」。リソースファクトリの型です。独自の型を指定するか、次のいずれかの型を選択します。
     - javax.sql.DataSource － JDBC 接続ファクトリ
     - javax.jms.QueueConnectionFactory － Java Message Service 接続ファクトリ
     - javax.jms.TopicConnectionFactory － Java Message Service 接続ファクトリ
     - javax.mail.Session － JavaMail セッションファクトリ
     - java.net.URL － URL 接続ファクトリ
   - 「承認」 － 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。
     - コンテナ － 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。
     - アプリケーション － Enterprise 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。

セキュリティロール参照

Enterprise 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。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なければなりません。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`(ロール名)

その場合は、該当するすべてのロール名を、セキュリティロールの参照として、このプロパティシートに追加する必要があります。

代わりに、モジュールレベルでセキュリティチェックを実装することもできます。詳細については、『J2EE モジュールおよびアプリケーションのアセンブルと実行』を参照してください。

Bean レベルでセキュリティロールを追加するには、次の手順に従います。

1. 「セキュリティロール参照」フィールドをクリックし、「...」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」ダイアログボックスが表示されます。

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。このフィールドは必須です。
   - 「説明」。ロールの説明です。

   「セキュリティロールリンク」フィールドでは、ここで指定したセキュリティロールを、配備先の環境でのセキュリティロールに対応付けることができます。ただし、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。

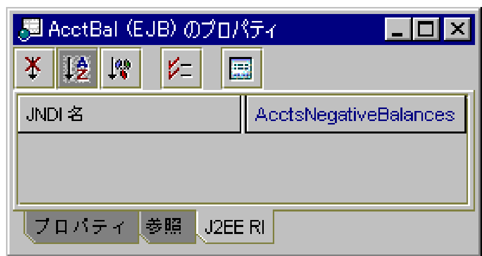
「J2EE RI」タブ

「J2EE RI (J2EE Reference Implementation サーバー)」のタブには、EJB ビルダーを使用して Enterprise 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、これらの値のいくつかを変更する必要があります。

セッション Bean や、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 コンテナに持続性を管理させるエンティティ Bean (CMP Bean) の場合は、J2EE RI 関連のプロパティに多くの情報が記録されています。これらの情報が、Bean とデータ記憶装置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。

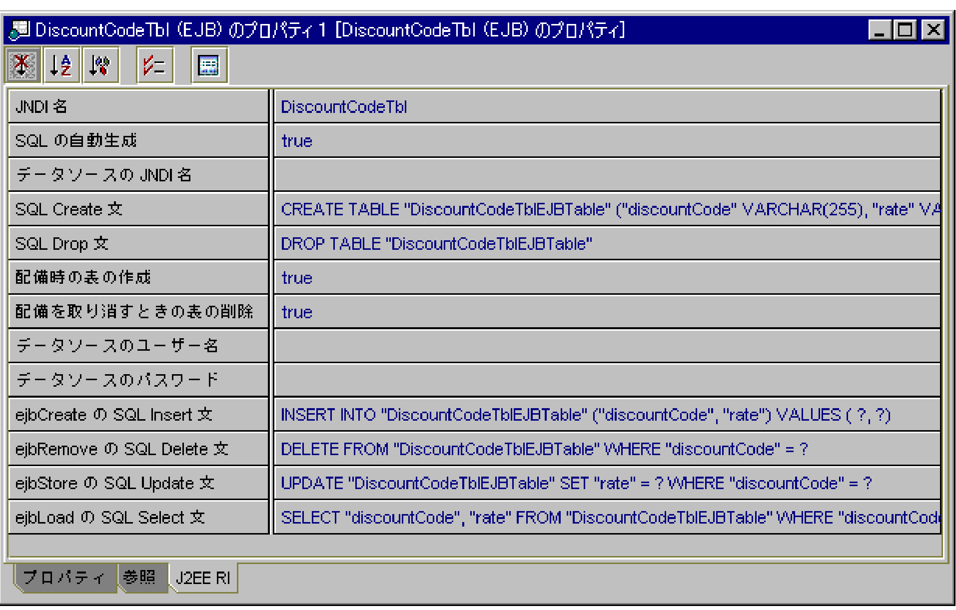
セッション Bean と BMP エンティティ Bean の J2EE RI プロパティの設定

次の図に示すように、セッション Bean や BMP Bean では、このタブには「JNDI 名」フィールドしか表示されません。作成前の Enterprise Bean は、この名前で特定されます。この名前は必要に応じて変更できますが、空白にすることはできません。



CMP エンティティ Bean の J2EE RI プロパティの設定

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 Bean) のプロパティフィールドは次のようになります。これらのフィールドは、Bean とアクセス先のデータ記憶装置との関係や、これらの関係を管理するコンテナの役割を表しています。



ほとんどの場合、CMP Bean は最初から作成するか、既存のデータベース表から生成します。「J2EE RI」タブの設定方法は、このどちらの方法で Bean を作成したかによって異なります。

EJB ビルダーのウィザードを使用して CMP Bean を作成すると、J2EE RI プラグインによって、アクセス先のデータベースの Bean の持続レコードを管理するのに必要な SQL 文が自動的に生成されます。さらに、CMP フィールドを追加または削除したり、主キーを構成しているフィールドを変更したりすると、そのたびにこれらの SQL 文が自動的に再生成されます。RI をサーバー兼コンテナとして使用すると、これらの SQL コードが使用されます。ただし、CMP Bean の作成方法によっては、これらの SQL 文をすべて使用するかどうかを選択できます。また、場合によっては、これらの SQL 文を編集できます。

J2EE RI プラグインが生成する SQL 文は、次の処理を行うように設定されます。

- Bean が配備されたときに表を作成し、この表に <Bean クラス名>`Table` という名前を付けます。また、この表のそれぞれの列に、対応する CMP フィールドと同じ名前を付けます。
- Bean の配備が解除されたときに、この表を削除します。
- Bean の実行中にレコードを削除、挿入、選択、および更新します。

ここからは、CMP Bean の配備プロパティを設定するためのガイドラインを示します。

最初から作成した CMP Bean のプロパティ設定

CMP Bean を既存のデータベース表からではなく、最初から作成する場合、通常は少なくともテスト用として 1 度は、Bean の CMP フィールドと名前と型が一致する列から構成されるデータベース表を、J2EE RI プラグインに作成させる必要があります。このタブのフィールドを次のように設定してください。

- 「SQL の自動生成」。デフォルト値 `true` を使用します。

  この値を使用すると、SQL 文に含まれている表や列の名前を編集したときに、編集内容に合わせてプラグインが SQL 文を自動的に再生成します。SQL 文のほとんどの部分は編集してはなりません。プラグインが SQL 文を再生成したときに、編集した内容が上書きされてしまうからです。ただし、列のデータ型や検索メソッドの `WHERE` 句の編集内容は、そのまま保持されます。

- 「配備時の表の作成」。デフォルト値 `true` を使用します。

  この値を使用すると、RI プラグインに Bean をテストするためのデータベース表を作成させることができます。

  Bean をはじめて配備するときや、CMP フィールドを追加または削除したときは、この値を `true` のままにしておきます (通常は、「配備を取り消すときの表の削除」フィールドの値も `true` にしておきます)。

  ただし、Bean をテストしている間に、それ以降のテストで使用する行をデータベース表に追加する場合があります。その場合は、両方のフィールドの値を `false` に設定します。

- 「データソースの JNDI 名」。エンティティの状態を記録するデータストアの JNDI 名です。`jdbc/Oracle` のように指定します。

- 「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です。

- 「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です。

- 「配備を取り消すときの表の削除」。デフォルト値 `true` を使用します。

通常、この値は「配備時の表の作成」プロパティ値と同じにします。RI プラグインが作成した表が不要になったときに、その表がプラグインによって削除されます。

- 「JNDI 名」。このフィールドには、JNDI 検索呼び出しが Bean を特定するときに使用する名前がデフォルト値として表示されます。この名前を変更したい場合は、このフィールドを編集することができます。ただし、空白にすることはできません。
- 「SQL Create 文」。必要に応じて、列の SQL データ型を変更することができます。それ以外の変更は、RI プラグインが SQL 文を再生成したときに上書きされてしまいます。

  CMP Bean の作成方法にかかわらず、プラグインはこの SQL Create 文と次の SQL 文を自動的に生成します。
- 「ejbRemove の SQL Delete 文」。表や列の名前を変更する必要がある場合を除いて、そのままにしておきます。
- 「SQL Drop 文」。表や列の名前を変更する必要がある場合を除いて、そのままにしておきます。
- 「ejbCreate の SQL Insert 文」。表や列の名前を変更する必要がある場合を除いて、そのままにしておきます。
- 「ejbLoad の SQL Select 文」。表や列の名前を変更する必要がある場合を除いて、そのままにしておきます。
- 「ejbStore の SQL Update 文」。表や列の名前を変更する必要がある場合を除いて、そのままにしておきます。

Bean を最初から作成した場合も、既存のデータベース表から生成した場合も、128 ページの「CMP Bean の検索メソッドのプロパティ設定」に従って、さらにプロパティ値を編集する必要があります。

データベース表から生成した CMP Bean のプロパティ設定

Bean をデータベース表から直接生成した場合、通常はその表を使用して Bean を直接テストします。そのため、RI プラグインに表を作成させる必要はありません。「J2EE RI」タブのプロパティフィールドを使用して、生成された SQL 文のうち、いくつかの文を RI に無視させたり、残りの文をカスタマイズしたりできます。このタブのフィールドを次のように設定してください。

- 「SQL の自動生成」。値 `false` を使用します。

  この値を使用すると、編集を行なったときに、RI プラグインが SQL 文を再生成しません。生成された SQL 文をどのように編集しても、編集内容がプラグインによって上書きされなくなります。

- 「配備時の表の作成」。値 `false` を使用します。

  この値を使用すると、Bean が配備されたときに、RI はデータベース表を作成しません。Bean が使用するデータベース表はすでに存在しているため、データベース表を新しく作成する必要はありません (通常は、「配備を取り消すときの表の削除」フィールドの値も `false` にします)。

- 「データソースの JNDI 名」。エンティティの状態を記録するデータストアの JNDI 名です。`jdbc/Oracle` のように指定します。

- 「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です。
- 「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です。

- 「配備を取り消すときの表の削除」。値 `False` を使用します。

  通常、この値は「配備時の表の作成」プロパティ値と同じにします。「配備時の表の作成」フィールドで `false` を指定しているため、RI プラグインは Bean 用の表を作成しません。

- 「JNDI 名」。このフィールドには、JNDI 検索呼び出しが Bean を特定するときに使用する名前がデフォルト値として表示されます。この名前を変更したい場合は、このフィールドを編集することができます。ただし、空白にすることはできません。

- 「SQL Create 文」。入力する必要はありません。「配備時の表の作成」フィールドで `false` を指定しているため、RI はこのフィールドで設定されているデフォルトの `CREATE TABLE` 文を無視します。

  CMP Bean の作成方法にかかわらず、プラグインはこの SQL Create 文と次の SQL 文を自動的に生成します。

- 「ejbRemove の SQL Delete 文」。次のように編集します。

- Bean が使用する既存のデータベース表で使用できるように、デフォルトの DELETE FROM 文を変更します。CMP Bean の主キー名と表の主キー名が一致していない場合は、表の主キー名を使用します。
- 引用符を取り除きます。詳細については、後述の図を参照してください。

- 「SQL Drop 文」。入力する必要はありません。「配備を取り消すときの表の削除」フィールドで false を指定しているため、RI はこのフィールドで設定されているデフォルトの DROP TABLE 文を無視します。

- 「ejbCreate の SQL Insert 文」。次のように編集します。
  - Bean が使用する既存のデータベース表で使用できるように、デフォルトの INSERT INTO 文を変更します。

    このデフォルトの SQL 文は、データベース表の列名が Bean の CMP フィールド名と一致していることを前提にしています。CMP フィールド名を変更した場合や、Java の有効な名前にするために、IDE がこの名前を自動的に変更した場合は、それに合わせて SQL 文に含まれている列名を変更する必要があります。

    RI は、複数の CMP フィールドの値を特定の順番で送出します。詳細については、130 ページの「CMP フィールド値の順序」を参照してください。
  - 引用符を取り除きます。

- 「ejbLoad の SQL Select 文」。次のように編集します。
  - Bean が使用する既存のデータベース表で使用できるように、デフォルトの SELECT 文を変更します。Bean の CMP フィールド名と一致していない列名をすべて変更します。また、主キー以外の列だけを選択するようにします。
  - 引用符を取り除きます。

- 「ejbStore の SQL Update 文」。次のように編集します。
  - Bean が使用する既存のデータベース表で使用できるように、デフォルトの UPDATE 文を変更します。Bean の CMP フィールド名と一致していない列名をすべて変更します。また、主キー以外の列だけを選択するようにします。
  - 引用符を取り除きます。

例として、RI プラグインが Dept という CMP Bean 用に次の SQL 文を生成した場合を考えてみましょう。

CREATE TABLE "DeptEJBTable" ("deptno" NUMERIC(15), "dname" VARCHAR(255), "loc" VARCHAR(255))
DELETE FROM "DeptEJBTable" WHERE "deptno" = ?
DROP TABLE "DeptEJBTable"
INSERT INTO "DeptEJBTable" ("deptno", "dname", "loc") VALUES (?, ?, ?)
SELECT "dname", "loc" FROM "DeptEJBTable" WHERE "deptno" = ?
UPDATE "DeptEJBTable" SET "dname" = ?, "loc" = ? WHERE "deptno" = ?

その場合は、それぞれの SQL 文を次のように編集します。

CREATE TABLE DEPT (DEPTNO NUMERIC(15), DNAME VARCHAR(255), LOC VARCHAR(255))
DELETE FROM DEPT WHERE DEPTNO = ?
DROP TABLE DEPT
INSERT INTO DEPT (DEPTNO, DNAME, LOC) VALUES (?, ?, ?)
SELECT DNAME, LOC FROM DEPT WHERE DEPTNO = ?
UPDATE DEPT SET DNAME = ?, LOC = ? WHERE DEPTNO = ?

大文字を使用していることと、引用符を取り除いていることに注意してください。

Bean を最初から作成した場合も、既存のデータベース表から生成した場合も、次の節に従って、さらにプロパティ値を編集する必要があります。

CMP Bean の検索メソッドのプロパティ設定

CMP Bean の J2EE RI プロパティの設定を終えるためには、RI が Bean の検索メソッド用に生成した SQL Select 文も編集する必要があります。このフィールドを編集するには、次の手順で検索メソッドのプロパティシートに移動する必要があります。

1. エクスプローラで Bean の論理ノードを展開します。
2. Bean の「検索メソッド」ノードを展開します。
3. 検索メソッドを選択し、右クリックし、「プロパティ」を選択します。

   検索メソッドのプロパティシートが次のように表示されます。

4. RI プラグインが生成した SQL を変更した場合は、表の名前と主キーになっている列の名前を、Bean の INSERT 文、SELECT 文、UPDATE 文で指定したものに合わせて変更します。

   主キーフィールドだけを選択するようにします。検索メソッドは、エンティティの主キーだけを返します。エンティティのデータの取り出しは ejbLoad メソッドが行います。

   CMP Bean の「SQL の自動生成」フィールドの値が TRUE になっている場合は、プラグインが SQL を再生成したときに、表名や列名の変更内容が上書きされてしまいます。

5. CMP Bean を最初から作成した場合も、既存のデータベース表から生成した場合も、WHERE 句を完成させます。

   - FindByPrimaryKey メソッドの場合、プラグインが生成した SQL を変更したときは、それに合わせて列名を変更することができます。それ以外の場合は、このメソッドはそのままにしておきます。

   - それ以外の検索メソッドの場合、制約条件を指定する必要があります。たとえば、次の検索メソッドが指定されているとします。

     ```
     findInSalaryRange(double low, double high)
     ```

     その場合は、SQL 文を次のように指定します。

     ```
     SELECT “EMP_ID” FROM “EMPLOYEE” WHERE “SAL” > ?1 AND “SAL” < ?
     2
     ```

     疑問符 (?) の後ろの数値は、検索メソッドのパラメータリストでのパラメータの順番を示しています。

   WHERE 句の編集内容は、プラグインが SQL 文を再生成しても、そのまま保持されます。

CMP フィールド値の順序

コンテナは、CMP フィールドの値を、フィールド名のアルファベット順にデータベースに送出します。表の列名が、対応する CMP フィールド名と一致していない場合は、それぞれの列を、対応する CMP フィールドのアルファベット順と同じ順序で指定する必要があります。次の EmployeeEJB という CMP Bean を例に、このことを説明します。

- ソースデータベース表の名前は Employee です。
- CMP フィールドの名前は address、city、id、name です。
- 表での対応する列の名前はそれぞれ Mail_Address、City、Emp_ID、Emp_Name です。
- デフォルトで生成される INSERT 文は次のようになります。

```
INSERT INTO "EmployeeEJBTable" ("address", "city", "id",
"name") VALUES (?, ?, ?, ?)
```

その場合は、INSERT 文の SQL コード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 (?, ?, ?, ?)
```

このように、表の列名を、(列名のアルファベット順ではなく) 対応する CMP フィールドのアルファベット順に指定する必要があります。これは、VALUES リストのパラメータがこの順に送出されるからです。

EJB モジュールの作成

単独で動作する Enterprise Bean を作成する場合もあれば、ほかの Bean と連携して動作する Enterprise Bean を作成する場合もあります。どちらの場合も、作成した Enterprise Bean を EJB モジュールにパッケージ化して、複数の Bean を 1 つにまとめ、それらの Bean が動作するのに必要な情報を、Bean とともに組み込む必要があります。

EJB モジュールは IDE での論理表現です。それに対応する物理表現は、EJB JAR (.jar 拡張子が付いた Java アーカイブファイル) です。EJB モジュールには、EJB JAR に組み込む必要のある Bean のリストと、配備先の環境で設定する必要のある Bean のプロパティが記録されます。EJB モジュールは、アプリケーションに配備可能な Enterprise Bean の最小配備単位です。

EJB モジュールに組み込む Enterprise Bean の決定

ここでは、1 つの EJB モジュールに、いくつの Enterprise Bean をパッケージ化すればよいかを決定するための一般的なガイドラインを示します (詳細については、Java 2 Platform, Enterprise Edition のマニュアルを参照してください)。次のことを考慮して Bean をパッケージ化してください。

- 再利用性。非常によく再利用される Enterprise Bean がある場合は、その Bean を単独で EJB モジュールにパッケージ化します。アセンブル担当者は、このモジュールとほかのモジュールを自由に組み合わせて、アプリケーションに必要な機能だけを提供し、アプリケーションのサイズを抑えることができます。

  1 つのモジュールに、同時に使用されることの多い複数の Bean をまとめて組み込むこともできます。

- アセンブルのしやすさ。1 つのモジュールに、アプリケーションに必要な (または、少なくともアプリケーションの特定の機能に必要な) Enterprise Bean をすべてパッケージ化すると、アセンブル担当者の作業が少なくなります。再利用性を考慮する必要がない場合は、この手法が効果的です。

- 再利用性とアセンブルのしやすさのバランス。適切なサイズの J2EE アプリケーションを作成するには、お互いに関連している Enterprise Bean や、密接に結び付けられた Enterprise Bean は 1 つのモジュールにまとめ、単独で再利用可能な Enterprise Bean は、その Bean 専用のモジュールに単独で組み込みます。たとえば、機能的に関係している Bean、相互に依存している Bean、相互に参照されている Bean、セキュリティプロファイルが共通の Bean は、1 つのモジュールにまとめることができます。

EJB モジュールへの Enterprise Bean の組み込み

1 つの Enterprise Bean だけを含んだ EJB モジュールを作成するには、次の手順に従います。

1. エクスプローラウィンドウから Bean の論理ノードを選択します。

2. 右クリックし、「新規 EJB モジュール」を選択します。
3. 「新規 EJB モジュール」ダイアログボックスで、必要に応じてモジュール名を変更します。
4. ファイルシステムのツリー表示から、モジュールの保存先を選択します。
5. 「了解」をクリックします。

   連携して動作する複数の Enterprise Bean を含んだ EJB モジュールを作成する場合も、基本的な手順は同じです。その場合は、Control キーを使用して、モジュールに組み込む複数の Bean を同時に選択します。

   モジュールの内容を参照するには、エクスプローラで該当するモジュールのノードを展開します。

作成済みのモジュールに Enterprise Bean を追加するには、次の手順に従います。

1. エクスプローラウィンドウから、該当するモジュールを選択します。
2. 右クリックし、「EJB の追加」を選択します。
3. ファイルシステムのツリー表示から、モジュールに追加する Enterprise Bean を選択します。
4. 「了解」をクリックします。

EJB 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では、EJB コンテナが CMT Bean のトランザクションを制御する方法を指定します。トランザクションを自分自身で管理するセッション Bean (BMT Bean) の場合は、Bean のトランザクションを明示的にコーディングする必要があります。これに対して、CMT Bean では、トランザクションのコードを明示的に組み込む必要はありません。代わりに、Bean 提供者が Bean のメソッドに割り当てたトランザクション属性に基づいて、コンテナにトランザクションを制御させることができます。

デフォルトでは、Bean のすべてのメソッドで「Required」属性が使用されますが、Bean ごとやメソッドごとに、別のトランザクション属性を割り当てることができます。たとえば、特定のメソッドをトランザクションのコンテキストから除外したい場合は、そのメソッドの属性を「Required」から「NotSupported」に変更することができます。

トランザクション属性は配備記述子に記録され、EJB モジュールのプロパティシートで編集できます。CMT Bean を含んだ EJB モジュールをアセンブル担当者に渡す前に、そのモジュールで適切なトランザクション属性が指定されていることを確認してください。

EJB モジュールのプロパティシートで、Bean (または Bean に含まれているメソッド) のトランザクション属性を変更すると、その EJB モジュールでの実行用としてだけ属性が変更されます。Bean のソースコードは変更されません。その Bean を別の EJB モジュールで再利用する場合は、別のトランザクション属性を適用できます。

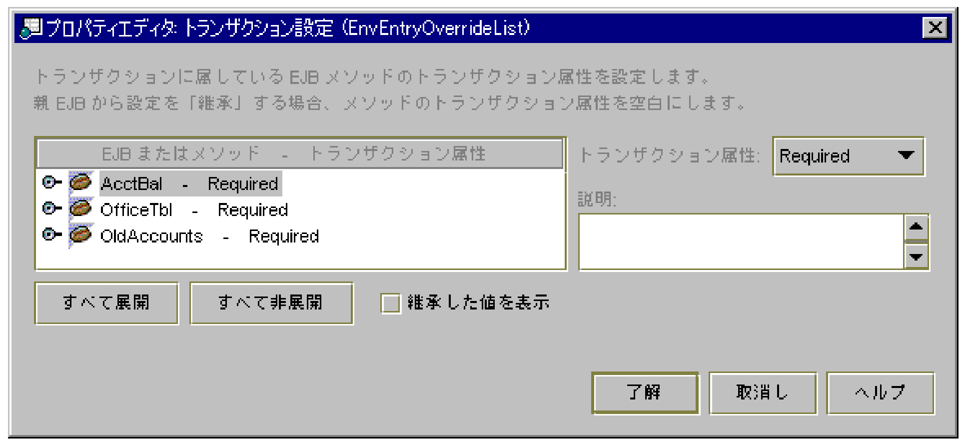
EJB モジュールの内部で CMT Bean のトランザクション属性を変更するには、次の手順に従います。

1. エクスプローラウィンドウから、該当する Bean の EJB モジュールを選択します。
2. 右クリックし、「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「コンテナトランザクション」と表示されます。

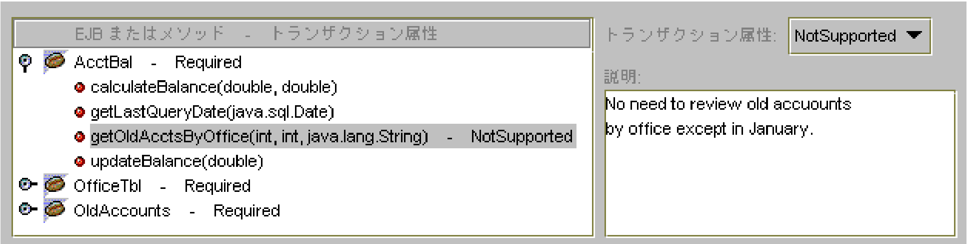
3. 「プロパティ」タブの「コンテナトランザクション」フィールドをクリックし、「...」ボタンをクリックします。

   「トランザクション設定」ダイアログボックスが表示されます。大きい方の区画に、そのモジュールに含まれている CMT Bean と、その Bean 全体に適用されるトランザクション属性が表示されます。



- Bean 全体のトランザクション属性を変更するには、該当する Bean を選択し、「トランザクション属性」コンボボックスから別の属性を選択します。大きい方の区画の Bean 名の隣に、選択した属性が表示されます。

- 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、該当する Bean を展開し、その中のメソッドを選択し、「トランザクション属性」コンボボックスから別の属性を選択します。次の図のように、大きい方の区画のメソッド名の隣に、選択した属性が表示されます。Bean のそれ以外のメソッドのトランザクション属性は、デフォルトの属性を変更しない限り表示されません。



「トランザクション設定」ダイアログボックスでは、アセンブル担当者のために、モジュール内の Bean やメソッドのトランザクション設定についての説明を入力できます。たとえば、この EJB モジュールの特定のトランザクション属性を変更した理由を記述しておくことができます。

4. トランザクション属性を変更した後で、「了解」をクリックします。

EJB JAR の作成

EJB モジュールを作成すると、次の手順で EJB JAR を作成できるようになります。

1. エクスプローラウィンドウから、Bean の EJB モジュールを選択します。
2. 右クリックし、「EJB Jar ファイルをエクスポート」を選択します。

EJB モジュールを J2EE アプリケーションの中に配置し、配備することもできます。アセンブルと配備の詳細については、『J2EE モジュールおよびアプリケーションのアセンブルと実行』を参照してください。

Enterprise Bean のテスト

実際にアプリケーション全体をアセンブルし、本稼働用サーバーに配備する前に、開発中の Enterprise Bean をテストできると便利です。Forte for Java IDE には、そのための J2EE アプリケーションを作成する機能があります。このアプリケーションは、テスト用の JavaServer Pages™ (JSP™) ページを含んだ Web モジュールと、Bean の EJB モジュールから構成されます。このアプリケーションを使用して、JSP ページによって生成された HTML ページを Web ブラウザで参照し、Enterprise Bean のインスタンスを作成し、HTML ページから Enterprise Bean のビジネスメソッドを実行できます。

IDE が作成するテストオブジェクトは、テストで使用することを目的にしています。本稼働環境への配備を想定して作成されたものではありません。

テスト環境の設定

IDE に用意された Enterprise Bean のテスト機能を使用するには、まず EJB モジュールを J2EE RI、または、インストール済みのその他のアプリケーションサーバーに配備する必要があります。テスト用として RI を使用する場合は、次の手順に従います。

1. J2EE RI をローカルマシンにインストールします。
2. エクスプローラの「実行時」タブに含まれている「サーバーレジストリ」ノードを使用して、アプリケーションサーバーインスタンスを作成します。

3. 「アプリケーションサーバー」ノードに表示される「J2EE Reference Implementation」ノードで RIHome プロパティを設定し、J2EE RI のインストール先のパスを指定します。

   詳しい手順については、RI のオンラインヘルプを参照してください。

   IDE に含まれている PointBase データベースを使用して、Enterprise Bean をテストすることができます。テストアプリケーションが PointBase データベースを検出し、ログインできるようにするには、次の手順に従います。

1. エクスプローラウィンドウから Bean の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。

2. 「プロパティ」ウィンドウの「J2EE RI」タブを選択します。

3. 次のように入力し、データベース接続情報を指定します。

フィールド	入力する情報
データソースの JNDI 名	`jdbc/Pointbase`
データソースのパスワード	`PUBLIC`
データソースのユーザー名	`PUBLIC`

注 - `Pointbase` は先頭の文字だけを大文字で入力します。また、パスワードを入力した後で Enter キーを押します。

---

4. 次のプロパティを False に設定します。

   配備時の表の作成

   配備を取り消すときの表の削除

5. 「ファイル」 > 「すべてを保存」を選択し、設定した内容を保存します。

注 - iPlanet Application Server (iAS) を使用して Bean をテストする場合は、iAS のマニュアル、『Developer's Guide (Java™): iPlanet™ Application Server』(`http://docs.iplanet.com/docs/manuals/ias.html`) を参照してください。

テストオブジェクトの生成

Enterprise Bean のテストで使用する Web モジュールと EJB モジュールを作成するには、次の手順に従います。

1. エクスプローラウィンドウから Bean の論理ノードを選択し、右クリックし、「新規 EJB テストアプリケーション」を選択します。

   ウィザードが起動し、アプリケーションをテストするのに必要なすべてのコンポーネントのデフォルト値が表示されます。

   「パッケージ」フィールドには、Bean の現在のパッケージ名が表示されています。このフィールドに別のパッケージ名を入力し、ウィザードが作成する EJB モジュールを別の場所に配置することができます。

   必要であれば、後続のフィールドを使用して、パッケージ名やモジュール名を指定することもできます。

   テスト用として RI を使用する場合は、「アプリケーションサーバー」コンボボックスに RI が表示されていることを確認します。

2. 「作成時にアプリケーションをアプリケーションサーバーに配備」チェックボックスに印を付けるか、このチェックボックスを空白のままにしておきます。

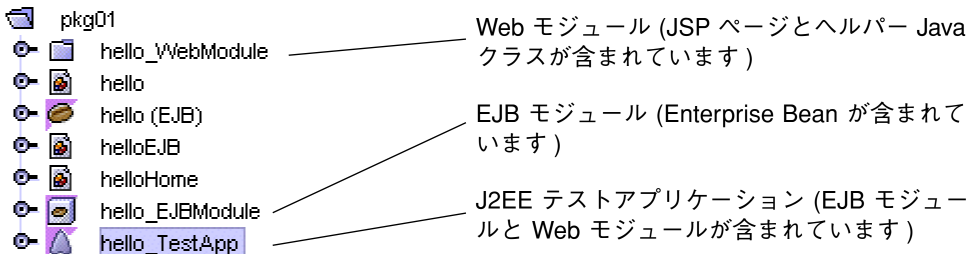
   このチェックボックスに印を付けると、ウィザードを使用して Bean のテストモジュールを作成した時点で、そのモジュールがただちにサーバーに配備されます。

   このチェックボックスを空白のままにした場合は、作成したテストモジュールを後ほど手作業で配備します。Bean をほかの Bean と組み合わせてテストする必要がある場合は、この設定を使用します (138 ページの「複数の Enterprise Bean のテスト」を参照してください)。

3. 「了解」をクリックすると、テスト用の EJB モジュール、Web モジュール、およびアプリケーションが生成されます。設定によっては、作成されたアプリケーションが自動的に配備されます。

   モジュールの作成と配備の進捗状況が表示されます。配備が完了すると、そのことを通知するメッセージが IDE のログウィンドウに表示されます。

   `pkg01` というパッケージに含まれている `hello` という Bean のテストオブジェクトは、エクスプローラウィンドウに次のように表示されます。ここでは、ウィザードが作成したテストオブジェクトを Bean と同じパッケージに配置した場合を想定しています。



複数の Enterprise Bean のテスト

テストする Enterprise Bean を、ほかの Enterprise Bean とともに動作させる必要がある場合 (たとえば、エンティティ Bean による処理を管理するセッション Bean をテストする場合) は、参照先の Bean を、作成した EJB モジュールに組み込む必要があります。

テストアプリケーションを生成する前に、次の作業を行います。

1. Bean のプロパティシートに、適切な EJB 参照を追加します。

   詳細については、117 ページの「EJB 参照」を参照してください。

   テストアプリケーションの作成中に、次の作業を行います。

2. ほかの Bean を参照している Bean の EJB モジュールを作成します。

   たとえば、2 つのエンティティ Bean の相互作用を管理するセッション Bean をテストする場合は、そのセッション Bean の EJB モジュールを作成します。

   生成した EJB モジュールは、このセッション Bean の EJB 参照を認識することができます。作成した EJB モジュールのプロパティシートを参照すると、この情報を確認できます。

   テストアプリケーションを生成した後で、次の作業を行います。

3. エクスプローラウィンドウから、生成した EJB モジュールのノードを選択し、右クリックし、「EJB の追加」を選択します。

4. ツリー表示から、参照先の Bean を選択し、「了解」をクリックします。

   参照先の Bean が EJB モジュールに追加されます。

   参照先の Bean ごとに、手順 3 ～ 手順 4 を繰り返します。

5. 次の節の手順に従って、テストアプリケーションを配備します。

サーバーへの Bean の配備

137 ページの「テストオブジェクトの生成」の手順 2 で、「作成時にアプリケーションをアプリケーションサーバーに配備」チェックボックスに印を付けなかった場合は、Enterprise Bean のテストアプリケーションをサーバーに配備してからテストを行う必要があります。テストアプリケーションを配備するには、次の手順に従います。

1. エクスプローラウィンドウから J2EE テストアプリケーションのノードを選択し、右クリックし、「配備」を選択します。

   配備の進捗状況が表示されます。

2. テストアプリケーションが配備されたかどうかを確認します。J2EE コマンドウィンドウを開きます。このウィンドウに「Application <Bean 名>_TestApp deployed.」という文が含まれていれば、テストアプリケーションが配備されています。

参照 - 配備が失敗した場合は、J2EE RI を正しく実行できるように IDE が設定されているかどうかを確認します。特に、RIHome プロパティが <J2EE_HOME> 値に設定されているかどうかを確認してください。

Bean のテスト

Enterprise Bean をテストするには、次の手順に従います。

1. Web ブラウザを開き、適切な URL を入力します。

   J2EE RI を使用している場合は、次の形式で URL を入力します。

   `http://localhost:<ポート番号>/<アプリケーション名>`

   <ポート番号> は、RI をインストールしたときに指定したポート番号です。

   <アプリケーション名> は、テストアプリケーションの名前です。

2. ブラウザにテストアプリケーションの JSP ページが表示されます。

   このページに表示された手順に従って、Enterprise Bean のインスタンスを作成し、そのビジネスメソッドを実行します。「Results of the Last Method Invocation」区画でテスト結果を確認します。

Bean の修正と再テスト

別のテストアプリケーションを生成しなくても、Enterprise Bean を修正し、再テストすることができます。ただし、修正後の Bean をテストする前に、テストアプリケーションを配備し直す必要があります。

注意 - IDE が生成する EJB モジュール、Web モジュール、および J2EE アプリケーションは、Enterprise Bean のテストで使用することを目的にしています。これらのオブジェクトは修正しないでください。これらのオブジェクトを修正すると、J2EE アプリケーションを配備できなくなる可能性があります。

付録 A

Enterprise Bean での透過的持続性の使用

データベースを操作する Enterprise Bean を作成するときは、JDBC API や Forte for Java IDE の透過的持続性 (Transparent Persistence) モジュールを使用して、Bean の持続性をプログラミングすることができます。JDBC API の使用法については、第 2 章を参照してください。この付録では、透過的持続性の使用目的、透過的持続性の処理内容、透過的持続性を使用して Enterprise Bean を作成するための IDE の支援機能、およびこれらの Bean 用に IDE が生成したコードを完成させる方法を説明します。

透過的持続性の使用目的

透過的持続性では、データが Java オブジェクト (正確には持続可能 Java クラス) として表現されます。これらのオブジェクトは、データベース固有のコードを記述することなく操作できます。これは、透過的持続性がデータソースの種類を識別し、適切な SQL を生成するからです。

透過的持続性では、(フィールドを含んでいる) クラスと (列を含んでいる) データベース表とのマッピングを、自動的に、および手作業で行うことができます。透過的持続性は (外部キーによって結び付けられた) 表どうしの関係を自動的に識別し、その関係をクラス間の関係にマッピングします。透過的持続性は、1 対 1、1 対多、多対多の関係に対応しています。持続 Bean インスタンスの作成、読み取り、更新、および削除は、持続性マネージャのインタフェースを介して実行されます。

透過的持続性は、EJB コンテナと自動的に連携し、Enterprise Bean のトランザクションを管理できます。

透過的持続性を使用すると、データベースに頻繁にアクセスする Enterprise Bean のパフォーマンスを改善できます。別々の `get` メソッドと `set` メソッドを使用する代わりに、シリアライズされた持続可能クラスを値オブジェクトとして使用し、複数のフィールドを表示および更新できます。

Enterprise Bean での透過的持続性の処理内容

EJB 環境で透過的持続性を使用するためには、基本的に次の手順に従う必要があります。

1. IDE の透過的持続性モジュールを使用して、持続可能クラスの開発とマッピングを行います。
2. IDE の EJB ビルダーのウィザードを使用して、`PersistenceManagerFactory` インスタンスと `PersistenceManager` インスタンスの参照を含み、持続可能クラスを使用する Enterprise Bean を作成します。
3. Bean のコードを完成させます。

この手順の詳細については、後続の節、および透過的持続性モジュールのオンラインヘルプとマニュアルを参照してください。

主な違い

すでに 2 層環境で透過的持続性を使用した経験がある場合は、多層 J2EE アプリケーションでは、持続可能クラスが 2 層環境とは多少異なる方法で取り扱われることに気づくでしょう。持続性マネージャファクトリの取得方法とトランザクションの管理方法に主な違いがあります。この違いを次に示します。

- Bean インスタンスがアクティブになっているときに、JNDI 検索呼び出しに持続性マネージャファクトリを検出させます。
- コンテナ管理による持続性を使用する持続性認識 Enterprise Bean のトランザクションを、EJB コンテナと透過的持続性が連携して管理します。

透過的持続性を使用する場合は、Enterprise Bean 自身の処理も多少変更する必要があります。主な違いは、ビジネスメソッドの実装方法、同期化の方法、トランザクションの管理方法です。この違いを次に示します。

- 持続可能クラスのインスタンスの参照を使用して、Bean のビジネスメソッドを実装します。このインスタンスが、Bean の状態を必要に応じて取得および更新します。

- トランザクションの同期化は持続性マネージャが処理します。持続性マネージャは、Bean のライフサイクルの適切な時点で、トランザクション完了コールバックを行います。

- 通常の Enterprise Bean と同様に、Bean のトランザクションを EJB コンテナに管理させることができます。その場合は、透過的持続性と EJB コンテナが連携して処理を行います。トランザクションを Bean 自身に管理させる場合は、ユーザートランザクション (すなわち、`javax.transaction.UserTransaction` インタフェースを実装したトランザクション) か、持続性マネージャから提供される透過的持続性トランザクションを Bean の内部で実行することができます。

リソースの取得方法

Enterprise Bean に必要なリソースは配備記述子で指定します。Enterprise Bean は、実行時に JNDI 検索メソッドを使用して、これらのリソースを動的に取得します。透過的持続性を使用する Bean に必要なリソースは、持続性マネージャファクトリです。たとえば、持続性マネージャファクトリを使用して、`hrdb` という人材データベースにアクセスする場合は、配備記述子を次のように指定します (ここでは、配備記述子の一部だけを掲載しています)。

```
</description>
<res-ref-name>jdo/hrdb</res-ref-name>
<res-type>com.sun.forte4j.persistence.PersistenceManagerFactory</res-type>
<res-auth>Container</res-auth>
</res-ref>
</session></enterprise-beans>
</ejb-jar>
```

該当する持続性マネージャファクトリが見つかると、持続性に必要なあらゆるオブジェクトを Bean から使用できるようになります。

持続性マネージャファクトリは、Bean のライフサイクルの適切な時点で、呼び出し元のトランザクションの関係を検出し、それと同じ関係を備えた持続性マネージャを取得します。Bean は、この持続性マネージャを介して CRUD (作成、読み取り、更新、および削除) 操作を行います。

持続性マネージャファクトリは、同じデータストアを共有するすべての Bean に共通のインスタンスになります。そのため、持続性マネージャファクトリは、トランザクションと持続性マネージャインスタンスとの関係を管理できます。あるトランザクションで、現在アクティブになっている持続性マネージャがない場合は、新しい持続性マネージャが作成され、そのトランザクションに関連付けられます。

接続プールを使用したい場合は、持続性マネージャファクトリの外部でデータソースとプロパティを設定する必要があります。この設定には、第 5 章で説明した Enterprise Bean のプロパティシートを使用します。

EJB ビルダーの起動

Forte for Java IDE には、セッション Bean に透過的持続性を明示的に実装する機能があります。エンティティ Bean に透過的持続性を使用するコードを追加することもできます。

まず、透過的持続性モジュールのマニュアルに記載されている手順に従って、持続可能クラスを設定します。その後で、次の手順を実行し、透過的持続性を使用する基本的なセッション Bean を作成します。

1. 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」区画から、セッション Bean の収容先のパッケージやファイルシステムを選択します。
3. 右クリックし、「新規」 > 「EJB」 > 「セッション Bean」を選択します。

   EJB ビルダーのウィザードが表示されます。
4. セッション Bean の種類を選択します。コンテナ管理によるトランザクション (CMT Bean) と Bean 管理によるトランザクション (BMT Bean) のうち、どちらか一方を選択します。
5. 「透過的持続性と共に使用」チェックボックスに印を付けます。「次へ」をクリックします。

   「EJB コンポーネント」区画が表示されます。
6. セッション Bean の名前を入力し、この区画のそれ以外のフィールドを必要に応じて変更し、「完了」をクリックします。

エクスプローラに新しい Bean が次のように表示されます。

AccountBalance
AccountBalance (EJB)
AccountBalanceEJB
AccountBalanceHome

アイコン にバッジマークが付いていることに注意してください。透過的持続性を使用して生成されたセッション Bean には、このバッジマークが表示されます。

Bean のコードの完成

Enterprise Bean の Bean クラスに、EJB ビルダーによって PersistenceManagerFactory と PersistenceManager の参照が配置されていることを確認してください。

どの種類の Bean についても、次の情報を格納したインスタンス変数が必要です。

- 持続性マネージャファクトリを検出するために、JNDI 検索メソッドで使用するオブジェクト名。このオブジェクト名は次の形式になります。
  java:comp/env/jdo/<持続性マネージャファクトリ名>

  たとえば、hrdb という人材データベースを使用する場合は、Bean のコードに次の変数を含めることができます。

```
private PersistenceManagerFactory jdoPersistenceManagerFactory = null;
private PersistenceManager jdoPersistenceManager = null;
private String jdoPMFName = “java:comp/env/jdo/hrdb”;
```

(第 5 章で説明した Bean のプロパティシートの「参照」タブを使用し、この変数に合わせてリソースファクトリの参照を定義する必要があります。)

その場合は、setSessionContext メソッドで使用するコードは次のようになります。

```
InitialContext ctx = new InitialContext();
jdoPersistenceManagerFactory =
    (PersistenceManagerFactory) ctx.lookup(jdoPMFName);
```

- すべてのライフサイクルメソッドと特定のビジネスメソッドに、持続性マネージャを取得するコードが含まれていなければなりません。また、持続性マネージャを取得したメソッドは、最後に PersistenceManager を閉じなければなりません (エンティティ Bean の ejbLoad メソッドを除きます)。

エンティティ Bean のライフサイクルメソッドは、次の規則に従って作成する必要があります。

- インスタンス変数に、対応する持続可能オブジェクトの参照を格納する必要があります。たとえば、BMP Bean 名が Employee で、持続クラス名が Emp だったとすると、この Bean の Bean クラスで、対応する Emp オブジェクトを格納するインスタンス変数を次のように宣言します。

```
Emp persistentEmp = null;
```

- ejbCreate メソッドでは、PersistenceManager の参照を取得し、主キー値用に PersistenceCapable クラスのインスタンスを作成し、この持続可能インスタンスのフィールドに ejbCreate メソッドに渡されたパラメータを代入します。さらに、このインスタンスをパラメータとして使用して PersistenceManager.makePersistent メソッドを呼び出します。
- ejbRemove メソッドでは、PersistenceManager の参照を取得し、持続可能インスタンスをパラメータとして使用して、PersistenceManager.deletePersistent メソッドを呼び出します。
- ejbLoad メソッドでは、それまでの持続性マネージャを閉じて、持続性マネージャファクトリから新しい持続性マネージャを取得します。さらに、この持続性マネージャインスタンスの PersistenceManager.getObjectById メソッドを呼び出して、特定の主キーを持つ持続可能インスタンスを検出します。
- ejbStore メソッドでは、エラーを防止するため、持続性マネージャを使用する処理は行いません。コンテナは、Bean のライフサイクルのどの時点でも (トランザクションの完了時を含みます)、このメソッドを呼び出す可能性があります。そのため、このメソッドでは持続性マネージャの内容を変更する処理は行わないでください。
- ビジネスメソッドでは、持続可能インスタンスのフィールドを操作することができます。持続可能フィールドの内容の変更は、透過的持続性によって自動的に追跡され、適切な時点で持続性認識インスタンスの状態がデータストアに反映されます。

- ビジネスメソッドでは、メソッドの先頭で持続性マネージャインスタンスを取得し、メソッドの末尾で、取得した持続性マネージャインスタンスを閉じる必要があります。この処理には、getPersistenceManager メソッドと PersistenceManager.close メソッドを使用します。
- ejbActivate メソッドでは、持続性マネージャを使用する処理を実行する必要はありません。エンティティ Bean のビジネスメソッドを呼び出したときに、持続性マネージャが取得されるからです。
- ejbPassivate メソッドでは、まだ PersistenceManager が開かれている場合に、その PersistenceManager を閉じます。さらに、持続可能インスタンスの参照を null に設定します。

シリアライズ

持続可能クラスを EJB メソッドのパラメータや戻り値として渡す場合は、そのクラスをシリアライズ可能にする必要があります。持続可能クラスを作成するときに、そのクラスをシリアライズ可能クラスとして定義する (すなわち、そのクラスに java.io.Serializable インタフェースを実装する) ことができます。

持続性マネージャを管理している仮想マシンの外部にオブジェクトを渡すと、持続性マネージャはそのオブジェクトの状態を追跡できなくなります。したがって、Enterprise Bean のクライアントが、Bean のリモートインタフェースから受け取ったオブジェクトを更新できるようにするには、更新後のオブジェクトを受け取り、その変更内容を持続インスタンスに反映させるメソッドを用意する必要があります。代わりに、更新を行うかどうかをクライアントに判断させ、Bean の別のメソッドを使用して情報を更新させることもできます。

別のケースとして、ビジネスメソッドの内部で、持続可能インスタンスに、トランザクションに含まれている (ただし、Enterprise Bean のコンポーネントに関連付けられていない) 別の持続可能インスタンスの参照が格納される場合があります。このような参照をクライアントに返す場合は、そのインスタンスのクラスがシリアライズ可能でなければなりません。

たとえば、OrderBean というエンティティ Bean で、Order という持続可能クラスをヘルパーインスタンスとして使用し、Order では LineItem という持続可能クラスを使用しているとします。持続可能インスタンス LineItem の配列を返すためには、LineItem クラスをシリアライズ可能にし、OrderBean に次のシグニチャのリモートメソッドを記述する必要があります。

```
LineItem[] getLineItems()
```

トランザクション管理

Enterprise Bean では、Bean の種類によってトランザクションの管理方法に次のような違いがあります。

- コンテナ管理によるトランザクションを使用するセッション Bean や、エンティティ Bean をプログラミングする場合は、トランザクション範囲のコーディングは行わず、Bean のトランザクション属性を適切なトランザクション型に設定します。
- Bean 管理によるトランザクションを使用するセッション Bean をプログラミングする場合は、トランザクションごとにコミットとロールバックをコーディングし、さらに Bean のトランザクション属性を適切に設定します。

どの種類の Bean についても、透過的持続性はコンテナと連携してトランザクションの動作を制御し、トランザクションの完了規則に違反することがないようにします。

持続可能オブジェクトの持続状態は、そのオブジェクトの持続性マネージャを保持している EJB コンテナの内部に含まれている限り保たれます。Bean 提供者は、持続可能オブジェクトのフィールドを自由に更新できます。コンテナのトランザクション制御機能が、定められたトランザクション範囲に従って、これらのフィールドの更新内容を適切にコミットします。

ただし、持続可能オブジェクトを EJB コンテナの外部に渡すと、そのオブジェクトが持続性マネージャから引き離され、持続状態が失われてしまいます。持続可能オブジェクトを EJB メソッドの戻り値として返すと、EJB クライアントのプログラマには、そのオブジェクトの複製が渡されます。すなわち、クライアントは、そのクライアントでどうやってトランザクションを管理するかにかかわらず、このオブジェクトの持続性を活用できません。このオブジェクトは一時オブジェクトになり、値オブジェクトまたはヘルパーオブジェクトとしてしか使用できません。

クライアントのプログラマが、これらのキャッシュされたオブジェクトを更新できるようにするには、クライアントで行われた更新内容を受け取り、内部で保存された持続インスタンスに反映させる更新メソッドを Bean に組み込む必要があります。

Bean 管理によるトランザクションのコーディング

トランザクションを開始してから、終了するまでの間に行なったすべての操作 (更新、挿入、および削除) が、データベースにコミットされます。トランザクションの基本形式は次のようになります。

```
Transaction tx = pm.currentTransaction();
tx.begin();
// ここで挿入、更新、および削除を実行
tx.commit();
```

Bean のトランザクションを完了させるには、EJB コンテナから提供される `javax.transaction.UserTransaction` インタフェースか、持続性マネージャから提供される `com.sun.forte4j.persistence.Transaction` インタフェースを使用します。次のガイドラインに従ってください。

- 複数のトランザクションで同じ持続性マネージャを使用したい場合は、`com.sun.forte4j.persistence.Transaction` を使用します。
- トランザクションごとに別の持続性マネージャを使用する場合は、`javax.transaction.UserTransaction` と `com.sun.forte4j.persistence.Transaction` のどちらを使用してもかまいません。
- `javax.transaction.UserTransaction` を使用してトランザクションを完了させる場合は、トランザクションを開始した後で、持続性マネージャファクトリから持続性マネージャを取得し、取得した持続性マネージャを閉じた後で、トランザクションをコミットまたはロールバックする必要があります。

`com.sun.forte4j.persistence.Transaction` を使用する場合は、次のようにトランザクションを完了させます。

```
// 同じ持続性マネージャを使用して、複数のトランザクションを実行するビジネスメソッド
persistenceManager = persistenceManagerFactory.getPersistenceManager();
persistenceManager.currentTransaction().begin();
// 最初のトランザクションの持続操作を実行
persistenceManager.currentTransaction().commit();
persistenceManager.currentTransaction().begin();
// 2 番目のトランザクションの持続操作を実行
persistenceManager.currentTransaction().commit();
persistenceManager.close();
```

javax.transaction.UserTransaction を使用してトランザクションを完了する場合のコードは次のようになります。

```
sessionContext.getUserTransaction().begin();
persistenceManager = persistenceManagerFactory.getPersistenceManager();
// トランザクションの持続操作を実行
persistenceManager.close();
sessionContext.getUserTransaction().commit();
```

持続性マネージャの操作

持続性マネージャはトランザクションオブジェクトです。すなわち、持続性マネージャには特定のトランザクションの情報が記録されます。持続性マネージャファクトリは持続性マネージャのプールを管理し、プールの中のそれぞれの持続性マネージャが別々のトランザクションに割り当てられます。Bean がトランザクションに適した持続性マネージャを取得することが重要です。そのためには、実行スレッドがトランザクションに関連付けられているときに、持続性マネージャを取得する必要があります。

- CMT セッション Bean では、それぞれのビジネスメソッドが別々のトランザクションに関連付けられる可能性があります。したがって、ビジネスメソッドごとに、持続性マネージャファクトリから持続性マネージャを取得し、ビジネスメソッドを終了するときに、取得した持続性マネージャを閉じます。
- BMT セッション Bean では、持続性マネージャ自身に通信状態を操作させることができます。そのため、いつ持続性マネージャを取得するかは、Bean 提供者が決定します。

Bean の内部で、次の変数をコーディングします。

```
PersistenceManager persistenceManager;
```

それぞれのビジネスメソッドでは、次のようなコードの中で持続インスタンスを操作します。

```
persistenceManager = persistenceManagerFactory.getPersistenceManager();
// 持続操作を実行
persistenceManager.close();
persistenceManager = null;
```

照会の記述

持続インスタンスはビジネスメソッドの処理を行います。特定の持続インスタンスを検出し、そのインスタンスを対象にロジックを実行するには、com.sun.forte4j.persistence.Query インタフェースを使用します。次の例に示すように、このインスタンスの使用法は 2 層アプリケーションの場合と同じです。

```
Float salaryTarget;
com.sun.forte4j.persistence.Query query = persistenceManager.newQuery();
query.setClass (Employee.class);
query.setCandidates (persistenceManager.getExtent(Employee.class));
query.setFilter ("salary == sal");
query.setParameters ("Float sal");
Collection result = (Collection) query.execute (salaryTarget);
for (iterator it = result.iterator(); it.hasNext();;) {
    ((Employee) it.next()).someMethod();
}
```

挿入、削除、および更新の記述

データベースの内容を変更する前に、トランザクションを開始しておく必要があります。トランザクションの開始と終了のガイドラインについては、149 ページの「Bean 管理によるトランザクションのコーディング」を参照してください。

データベースに新しいインスタンスを挿入するには、持続可能クラスの新しいインスタンスを作成し、持続性マネージャを使用して、このインスタンスを持続インスタンスにします。この例を次に示します。

```
Employee emp = new Employee (name, id, salary, addr1, addr2, city, state, zip);
persistenceManager.makePersistent(emp);
```

データベースからインスタンスを削除するには、ナビゲーションか照会を使用してインスタンスの参照を取得し、持続性マネージャを使用して、このインスタンスを削除します。この例を次に示します。

```
Department dept = emp.getDepartment();
persistenceManager.deletePersistent(dept);
```

データベース中のインスタンスを更新するには、ナビゲーションか照会を使用してインスタンスの参照を取得し、このインスタンスのメソッドを呼び出します。この例を次に示します。

```
Department dept = emp.getDepartment();
dept.setLocation(newLocation);
```

詳細情報の参照先

持続可能クラスの開発、PersistenceManagerFactory と PersistenceManager の基本的な使用規則、持続フィールドの操作といった持続性プログラミングの詳細については、『持続プログラミング』や、透過的持続性モジュールのオンラインマニュアルを参照してください。

さらに、「Transparent Persistence: An Introduction to Java Data Objects」(http://www.sun.com/forte/ffj/resources/articles/transparent.html) も参照してください。

付録 B

Enterprise Bean の操作

Enterprise 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。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。IDE の機能を十分に活用できるように、この付録、さらに第 3 章と第 4 章で推奨している手順に従ってください。

Bean の論理ノードを使用した作業

変更を正しく適用するために、Enterprise Bean の論理ノードから作業を開始します。論理ノードは、 アイコンで表示されます。このノードに Bean のすべての要素がまとめられています。

論理ノードで変更を行うと、Forte for Java IDE によって、Bean 全体に変更内容が適切に伝達されます。

ただし、論理ノード以外の場所で変更を行なった場合も、通常は変更内容が伝達され、Bean のクラスおよびインタフェースの整合性が保たれます。詳細については、53 ページ (セッション Bean の場合) および 84 ページ (エンティティ Bean の場合) の「論理ノードとプロパティシートを使用した作業」を参照してください。

変更内容の保存

Bean の内容は、明示的に保存するまでは保存されません。Bean をコンパイルしても、その内容が保存されるとは限りません。Bean の内容を確実に保存するには、「ファイル」 > 「すべてを保存」を選択します。

Enterprise Bean の検証

IDE の検証とコンパイルには別々の目的があります。Enterprise 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、たとえクラス間の不整合があったり、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。Enterprise Bean の要素間に不整合がないことを確認するには、次の手順で Bean を検証します。

- エクスプローラの「ファイルシステム」区画から、Bean の論理ノードを選択して右クリックし、「EJB の検査」を選択します。

  検証に多少の時間がかかることがあります。検証が完了すると、出力ウィンドウが開き、Bean についてのメッセージが表示されます。

Enterprise 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。

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。

   「名前を変更」ダイアログボックスが表示されます。「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

Enterprise 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 Enterprise 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、Enterprise Bean の要素を簡単に再利用できるようにするためです。

Enterprise 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。Bean を柔軟に再利用できるようにするため、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かまいません。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。

Enterprise Bean の複製を作成し、別のパッケージにペーストするには、次の手順に従います。

1. エクスプローラウィンドウから、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 の論理ノードを展開し、その中の「クラス」ノードに含まれているノードを右クリックし、「コピー」を選択します。次に、クラスや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. コピー先の 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、次のプロパティをコピー先のパッケージに合わせて変更します。

   - 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
   - 主キークラス (主キークラスがある場合)

   該当するプロパティをクリックし、「...」ボタンをクリックします。表示されるダイアログボックスで、コピー先のパッケージに移動し、手順 3 で作成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 <パッケージ名>.<クラス名> 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、コピー先のパッケージ名の後ろに元のクラス名 (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変更

作成済みの Enterprise 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、「...」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、第 4 章の 85 ページの「ソースエディタを使用したエンティティ Bean の修正」を参照してください。

代わりに、次のように「カスタマイザ」ダイアログボックス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」ダイアログボックスが開き、「新規メソッド」ダイアログボックス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。

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。

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。

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」ダイアログボックス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。

Enterprise Bean の削除

どの種類の Enterprise Bean についても、Bean を削除する方法は 1 つしかありません。

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」ダイアログボックスが表示されます。

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。

索引

A

afterBegin メソッド
  ステートフル CMT セッション Bean 28, 58, 65
afterCompletion メソッド
  ステートフル CMT セッション Bean 28, 58, 65

B

Bean
  エンティティ Bean の種類 71
  クラス 11
    エンティティ Bean 83
    エンティティ Bean での既存の Bean クラスの使用 80 - 82
    セッション Bean 52
  クラスとインタフェース 7, 52 - 53, 83 - 84
  セッション Bean の種類 45
Bean 管理による持続性 (BMP) 29
  生成されたコードの完成 106 - 111
Bean 管理によるトランザクション (BMT) 23, 47, 49
Bean 提供者が行う必要のある作業
  セッション Bean のコーディング 25
  エンティティ Bean のコーディング 30
Bean 提供者の役割 6
Bean のクラスやインタフェースの変更 156
Bean の検証 54 - 55, 85 - 87
Bean のコピーとペースト 155 - 156
Bean のプロパティ 39, 115
Bean のメソッドの修正 53 - 55, 84 - 87
Bean のメソッドの編集 157
beforeCompletion メソッド
  ステートフル CMT セッション Bean 28, 58, 65

C

CMP Bean のプロパティシートでの SQL 文の編集 122 - 130
CMP フィールド
  値の初期化 32
  個別指定 79 - 80
  追加 99
commit メソッド 64
CRUD 操作 143

E

EJB JAR ファイル 5, 131 - 135
EJB Specification 1.1 への対応 2
ejbActivate メソッド 57, 92, 106
  エンティティ Bean のインスタンス 31
  ステートフルセッション Bean 61 - 62
  ステートフルセッション Bean のインスタンス 28
ejbCreate メソッド 108

BMP エンティティ Bean 108 - 110
CMP エンティティ Bean 94 - 96
エンティティ Bean のインスタンス 32
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 27
セッション Bean 25, 53
ejbLoad メソッド 93, 106, 111
BMP エンティティ Bean のインスタンス 31
データストアとの同期 35
ejbPassivate メソッド 57, 93, 106
エンティティ Bean のインスタンス 31
ステートフルセッション Bean 61
ステートフルセッション Bean のインスタンス 28
ejbPostCreate メソッド 32, 109
BMP エンティティ Bean 109
CMP エンティティ Bean 94 - 95
ejbRemove メソッド 57, 93, 106
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 27
データベースエンティティの削除 34
ejbStore メソッド 35, 93, 106, 111
BMP エンティティ Bean のインスタンス 31
EJB アプリケーション設計例 36
EJB アプリケーションのワークフロー 12
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 30
エンティティ Bean へのサービスの提供 29
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 27
トランザクションの管理 23
EJB 参照 117 - 118
EJB ビルダーのウィザード 16, 44 - 67
Bean のクラスへの変更の伝達 54 - 55, 85 - 87
Bean の検証 54 - 55, 85 - 87
BMP エンティティ Bean のクラスの作成 101 - 102
CMP エンティティ Bean のクラスの作成 73 - 83
エンティティ Bean の定義 69 - 112
セッション Bean のクラスの作成 52 - 58
セッション Bean の定義 48 - 52
透過的持続性の実装 144 - 145
メソッドシグニチャの生成 26, 32
例外の作成 38
EJB モジュール 5
作成 130 - 135
トランザクション属性 132 - 134
プロパティ 39
EJB モジュールへの Bean の組み込み 131 - 132
Enterprise Bean
EJB クライアントとしての使用 2
EJB コンテナとの関係 4
JavaBeans との違い 3
アプリケーションでの使用 36
開発ライフサイクル 14
クラス 7
更新 153 - 158
持続性 17
セキュリティ 17, 39, 121
設計手法 42
テスト 135 - 140
トランザクション 17
メソッド 8
要素 7
ワークフロー 12
Enterprise Bean の開発ライフサイクル 14
Enterprise Bean の管理 153 - 158
Enterprise Bean の検証 154
Enterprise Bean の更新 153 - 158
Enterprise Bean の最適化 42
Enterprise Bean の削除 158
Enterprise Bean の設計手法 42
Enterprise Bean の操作 153 - 158
Enterprise Bean のパフォーマンス 42, 142
Enterprise Bean のメソッド 8
afterBegin 28
afterCompletion 28
beforeCompletion 28
ejbActivate 28
ejbCreate 25, 32

ejbLoad 31, 35
ejbPassivate 28, 31
ejbPostCreate 32
ejbRemove 27, 34
ejbStore 31, 35
findByPrimaryKey 33
getCallerPrincipal 40
hashCode 91
isCallerInRole 40
newInstance 25, 31
setEntityContext 31
setSessionContext 25
unsetEntityContext 32
検索メソッド 9, 33
実行権限 40
生成メソッド 9, 32
セキュリティ 17
ビジネスメソッド 9
ライフサイクルメソッド 9
equals メソッド 91

F

findByPrimaryKey メソッド 33, 104
Forte for Java IDE 44 - 67, 73 - 112

G

getCallerPrincipal メソッド 40
getRollbackOnly メソッド 65
getUserTransaction メソッド 64

H

hashCode メソッド 91

I

iAS 115
マニュアル 42
IDE
BMP エンティティ Bean の完成 106 - 111
CMP エンティティ Bean の完成 93 - 100
エクスプローラウィンドウ 49, 72
セッション Bean の完成 59 - 67
透過的持続性の実装 144 - 145
配備記述子の完成 114 - 130
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 49, 72
iPlanet アプリケーションサーバー、「iAS」を参照
isCallerInRole メソッド 40

J

J2EE
Specification 1.2 への対応 2
アプリケーションアーキテクチャ 2
J2EE Reference Implementation サーバー、「RI」を参照
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
JAR、「EJB JAR ファイル」 を参照
Java Transaction API 24
Java Transaction Service (JTS) 25
java.io.Serializable 91
java.rmi.Remote 91
java.rmi.RemoteException 38
java.security.Principal 41
java.sql.Connection 64
JavaBeans、Enterprise Bean との違い 3
javax.ejb.CreateException 38
javax.ejb.EJBContext 64
javax.ejb.EJBException 38
javax.ejb.EJBHome 51, 52, 84
javax.ejb.EJBObject 51, 52, 84
javax.ejb.EntityBean 84
javax.ejb.SessionBean 51, 52
javax.transaction.UserTransaction 64
JDBC API 5, 24, 29 - 30
JTA コードと併用しない 64

代替手段としての透過的持続性 39
JDBC API を使用した従来のコードの組み込み 24, 30
JNDI 4, 117, 142
JSP ページ、クライアントとしての使用 2
JTA 24, 64

L

loadRow メソッド 111

N

newInstance メソッド
エンティティ Bean 31
セッション Bean 25

P

private フィールド 107

R

RI 41, 111, 115
CMP エンティティ Bean 用のプロパティ 122 - 130
CMP フィールド値の順序 130
検索メソッド 128 - 129
最初からの作成 124 - 125
表からの作成 125 - 128
セッション Bean と BMP エンティティ Bean 用のプロパティ 122
プロパティの宣言 122 - 130
RI を使用したテスト 41, 135 - 140
RI を使用したプロトタイプ 41

S

setAutoCommit メソッド 64
setEntityContext メソッド 31, 92, 105
setRollBackOnly メソッド 64
setSessionContext メソッド 25, 57
SQL 5, 30
透過的持続性による生成 141
SQL Insert 文 109
storeRow メソッド 111

U

unsetEntityContext メソッド 32, 92, 106
UserTransaction メソッド 64
UT メソッド 64

X

XML 配備記述子ファイル 39

あ

アクティブ化
エンティティ Bean のインスタンス 31
ステートフルセッション Bean のインスタンス 27
アセンブル担当者の役割 6
値オブジェクト 142
後からフィールドをデータベースにマッピングする 83
アプリケーション開発の役割 6
アプリケーションサーバー
EJB コンテナのサービス 4, 29
RI と iAS 41
要件 51, 77, 111
アプリケーション設計例 36
アプリケーションのアセンブル、『J2EE モジュールおよびアプリケーションのアセンブルと実行』を参照
アプリケーションのオーバーヘッド 37
アプリケーション例
セッション Bean 46
アプリケーションレベルの問題 38

い

1 対 多の関係、透過的持続性による処理 141
入れ子になったトランザクション 25
インスタンスプール
エンティティ Bean 30
ステートレスセッション Bean 27

う

ウィザード、「EJB ビルダーのウィザード」を参照

え

エクスプローラのパッケージ (フォルダ) ノード 49, 72, 101
エラー情報 54 - 55, 85 - 87
エンティティ
Bean のデータベースへのマッピング 17
context メソッド 31
セッション Bean による表現 21
エンティティ Bean 28
Bean クラス 83
BMP エンティティ Bean のクラスの作成 101 - 102
BMP エンティティ Bean のコードの完成 106 - 111
CMP エンティティ Bean のクラスの作成 73 - 83
CMP エンティティ Bean のコードの完成 93 - 100
EJB コンテナとの関係 29
findByPrimaryKey メソッド 33
インスタンスの検出 33
主キー 33
主キークラス 84
種類 71
使用可能状態 31
プール状態 31
ホームインタフェース 83
ライフサイクル 30
リモートインタフェース 83
エンティティ Bean のインスタンスの検出 33
エンティティ Bean の匿名インスタンス 33
エンティティ Bean のフィールドの変更 157

か

外部依存性、「配備記述子」を参照
外部キー 39, 98
透過的持続性による処理 141
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 118 - 119
関連オブジェクト
エンティティ Bean 76
セッション Bean 51

き

休止
エンティティ Bean のインスタンス 31
ステートフルセッション Bean のインスタンス 27

く

クライアントと Bean との関係 20, 28

け

検索メソッド 9, 33, 89, 99, 110

こ

コードの完成
エンティティ Bean 30, 93 - 100
セッション Bean 59 - 67, 25
コードの生成
エンティティ Bean でのメソッドシグニチャ 32

セッション Bean のクラス 44
配備記述子 114
例外 38
固有の識別子、エンティティ Bean 28
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) 29
生成されたコードの完成 93 - 100
コンテナ管理によるトランザクション (CMT) 23, 47, 49

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 29
サービス
EJB コンテナからの提供 4, 29
サーブレット、クライアントとしての使用 2
再利用
ウィザードでの設定 51
宣言による実行時情報 39, 114
削除
エンティティ Bean のインスタンス 32
セッション Bean のインスタンス 26
データベースエンティティ 34
作成
新しいエンティティ 32
エンティティ Bean の新しいインスタンス 32
エンティティ Bean のインスタンス 30
セッション Bean のインスタンス 25
参照先の Enterprise Bean 117 - 118

し

システム例外 64
システムレベルの問題 38
事前定義例外 38
持続可能 Java クラス 141
持続可能クラス 39
持続性 17
BMP エンティティ Bean 107
EJB コンテナによる管理 29
ウィザードでの CMP エンティティ Bean 用の設定内容 73 - 83
プロパティ設定 122 - 130
持続性マネージャ 141, 150
持続性マネージャファクトリ 143
持続フィールド 90
個別指定 79 - 80
実行時情報 39, 114
主キー 33
エンティティ Bean の主キークラス 84
エンティティ Bean の主キーの新規作成 96 - 98
エンティティ Bean への主キーの追加 96
主キークラス 107
既存の主キークラス、エンティティ Bean での使用 80 - 82
必須メソッド 90 - 91
種類
エンティティ Bean 71
セッション Bean 45
使用可能状態、エンティティ Bean のインスタンス 31
状態、複数のメソッドの呼び出しにわたる保持 22
仕様への対応
EJB 1.1 2
J2EE 1.2 2
初期化
持続フィールド 32
ステートフルセッション Bean の状態 46

す

スーパークラス 51, 87
ステートフルセッション Bean 22, 45
ウィザードでの設定 49
休止とアクティブ化 27
ステートレスセッション Bean 21, 45

ウィザードでの設定 49

せ

生成メソッド 9, 53
エンティティ Bean 94 - 96, 108
セッション Bean 56, 59 - 61
データストアへのデータの挿入 32
セキュリティ 17, 39
getCallerPrincipal メソッド 40
isCallerInRole メソッド 40
配備記述子でのセキュリティロール 121
セキュリティチェック
エンティティ Bean 32
セッション Bean 26
セキュリティのコーディング 40
セキュリティの指定 40
設計手法 42
セッション Bean 20, 59 - 61
Bean クラス 52
アプリケーション例 46
エンティティの表現 21
コードの完成 59 - 67
種類 45
ステートフルセッション Bean 22
ステートレスセッション Bean 21
セッションでの状態の同期化 28
プールへの格納 22
ホームインタフェース 52
ライフサイクル 25
リモートインタフェース 52
セッション Bean インスタンスの初期化 25
セッション同期化インタフェース 28, 65 - 67
ウィザードでの設定 50
クラス 58
宣言
実行時情報 39, 114 - 130
セキュリティ 39, 121
トランザクション属性 132 - 134, 24

そ

ソースエディタ 54 - 55, 85 - 87
属性、「トランザクション属性」を参照

た

対応している仕様 2
多対多の関係、透過的持続性による処理 141

ち

違い
Enterprise Bean と JavaBeans 3
JTA と JDBC API 24
コンテナ管理による持続性と Bean 管理による持続性 29
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 23
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 21
セッション Bean とエンティティ Bean 20, 28

つ

通信セッション 20 - 28

て

データアクセスオブジェクト (DAO) 106
データ記憶装置との接続 119 - 120
データストアへのデータの挿入
生成メソッドの使用 32
データストアへの複数アクセス 39, 141
データの同期 35
データベースからのインスタンスの削除、透過的持続性の使用 151
データベーススキーマ 76 - 78
データベース接続 74 - 76
接続プール 144
データベースへのインスタンスの挿入

透過的持続性の使用 151
データベースマッピング 17
CMP フィールドとのマッピング 74 - 83
延期 83
データベースマッピングの延期 83

と
透過的持続性
Bean のコードの完成 145 - 152
Enterprise Bean での使用 141 - 152
JDBC API の代替手段としての使用 39
ウィザードでの設定 49
更新 151
削除 151
照会 151
シリアライズ 147
セッション Bean での透過的持続性 48
挿入 151
トランザクション管理 148 - 150
同期
エンティティ Bean のインスタンスとデータストア 35
同期化
セッションでの状態 28, 58
トランザクション 17
Bean 管理 23, 47
JDBC API の使用 24
JTA の使用 24
入れ子、JTA では使用不能 25, 63
エンティティ Bean でのトランザクション 29
コンテナ管理 23, 47
セッション Bean でのトランザクション 22, 47, 63 - 67
属性 24
EJB モジュール 132 - 134
個々の Bean 134
個々のメソッド 134
透過的持続性による管理 141
範囲 63
ロールバック 63
トランザクション制御
エンティティ Bean 32
セッション Bean 26

な
名前の変更
Bean のフィールド 157
Enterprise Bean 154

の
ノード
エンティティ Bean 83
セッション Bean 52
論理ノード 52, 83

は
配備記述子 12, 39, 114
配備担当者の役割 6
破棄
エンティティ Bean インスタンスのプールからの破棄 32
セッション Bean インスタンスのメモリーからの破棄 26

ひ
比較
エンティティ Bean の種類 71
セッション Bean の種類 46 - 47
ビジネスメソッド 9
BMP エンティティ Bean 111
CMP エンティティ Bean 98
エンティティ Bean 34
セッション Bean 26, 53, 62 - 63
ビジネスロジックの実行
エンティティ Bean 34
セッション Bean 26
表へのマッピング 74 - 78

ふ
フィールドの型の変更 158
プールへの格納
　エンティティ Bean のインスタンス 30
　ステートレスセッション Bean のインスタンス 22, 27
複数のメソッドの呼び出しにわたる状態の保持 22
プログラマの役割 6
プログラムによるセキュリティ 40, 41
プログラム例、場所 xxi
プロパティシート 39, 115

へ
変更内容の保存 153
変更の伝達 54 - 55, 85 - 87

ほ
ホームインタフェース 10
　エンティティ Bean 83
　セッション Bean 52
ほかの Bean から取り込んだクラスの修正 155

ま
マルチスレッド対応コード、Enterprise Bean では不要 28

め
メソッドの実行権限 40

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 38
　システムレベル 38
　論理ノード以外の場所での作業 54 - 55, 84 - 87

や
役割
　J2EE アプリケーションの開発 6

ゆ
ユーザーのセキュリティロール 39

ら
ラージアイコン 116
ライフサイクル
　エンティティ Bean 30
　セッション Bean 25
　メソッド 9
　　BMP エンティティ Bean 105 - 106
　　CMP エンティティ Bean 91 - 93
　　セッション Bean 61 - 62

り
リソース
　ステートフルセッション Bean 27
リソースファクトリの参照 119 - 120
リモートインタフェース 10
　エンティティ Bean 83
　セッション Bean 52
リモートオブジェクト 32
リモート例外 38

れ
例外 38
　java.rmi.RemoteException 38
　javax.ejb.CreateException 38
　javax.ejb.EJBException 38

　システム例外 64
　事前定義例外 38
　リモート例外 38
例外を使用した問題への対処 38

ろ

ロール
　セキュリティ 39
論理ノード 52, 53 - 55, 83, 84 - 87, 153