Enterprise JavaBeans 開発者ガイド

Sun™ ONE Application Server

Version 7

817-0605-10
2002 年 9 月

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

このソフトウェアは SUN MICROSYSTEMS, INC. の機密情報と企業秘密を含んでいます。SUN MICROSYSTEMS, INC. の書面による許諾を受けることなく、このソフトウェアを使用、開示、複製することは禁じられています。U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard standard license agreement and applicable provisions of the FAR and its supplements. Use is subject to license terms.

この配布には、第三者が開発したソフトウェアが含まれている可能性があります。

Sun 、Sun Microsystems 、Sun のロゴマーク、Java および Sun ONE のロゴマークは、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。

UNIX は、X/Open Company, Ltd が独占的にライセンスしている米国およびその他の国における登録商標です。

この製品は、米国の輸出規制に関する法規の適用および管理下にあり、また、米国以外の国の輸出および輸入規制に関する法規の制限を受ける場合があります。核、ミサイル、生物化学兵器もしくは原子力船に関連した使用またはかかる使用者への提供は、直接的にも間接的にも、禁止されています。このソフトウェアを、米国の輸出禁止国へ輸出または再輸出すること、および米国輸出制限対象リスト (輸出が禁止されている個人リスト、特別に指定された国籍者リストを含む) に指定された、法人、または団体に輸出または再輸出することは一切禁止されています。

目次

本書について 11
対象読者 11
マニュアルの使用法 12
マニュアルの構成 14
関連情報 15
マニュアルの表記規則 16
　一般的な表記規則 16
　ディレクトリ名の表記規則 17
製品サポート 18

第 1 章 Sun ONE Application Server Enterprise JavaBeans の紹介 19
EJB 2.0 の変更点の概要 20
EJB のアーキテクチャ 21
付加価値機能 23
　読み取り専用 Beans 23
　pass-by-reference 23
　プール機能とキャッシュ機能 24
　監視 24
　Sun ONE Studio 4 との統合 24
　動的な配備と再読み込み 25
Enterprise JavaBeans について 25
　Enterprise JavaBean とは 26
　Beans の種類 27
　EJB フロー 28
　EJB コンテナ 29
　インタフェース 30
　　ホームインタフェース 30

リモートインタフェース 30
ローカルインタフェース 30
プールとキャッシュ 31
プール関連パラメータ 32
キャッシュ関連パラメータ 32
Enterprise JavaBeans がリソースにアクセスするしくみ 33
JNDI 接続 33
データベースへの接続 33
URL への接続 34
トランザクション管理 34
アプリケーションセキュリティのしくみ 34
効率的なアプリケーションの開発について 35
Enterprise JavaBeans 作成の一般的なプロセス 35
Bean 使用の手引き 36
クライアントビューの手引き 37
リモートインタフェースとローカルインタフェースの手引き 38
Sun ONE Application Server 機能へのアクセス 39
EJB のアセンブリと配備について 39

第 2 章 セッション Beans の使用 41
セッション Beans について 42
セッション Bean の特性 42
コンテナ 43
ステートレスコンテナ 43
ステートフルコンテナ 44
セッション Beans の開発 45
開発の要件 45
セッション Beans の使用法の決定 46
ステートフルセッション Beans に関する検討事項 46
ステートレスセッション Beans に関する検討事項 46
インタフェースの提供 47
リモートインタフェースの作成 47
ローカルインタフェースの作成 48
ローカルホームインタフェースの作成 49
リモートホームインタフェースの作成 50
Bean クラス定義の作成 51
セッション同期 52
抽象的なメソッド 53
制限事項と最適化 53
セッション Bean のパフォーマンスの最適化 53
トランザクションの制限 54

第 3 章 エンティティ Beans の使用 55
エンティティ Beans について 56
エンティティ Beans の特性 56
コンテナ 57
持続性 57
Bean 管理による持続性 58
コンテナ管理による持続性 59
読み取り専用 Beans 59
エンティティ Beans の開発 60
エンティティ Bean の使用法の決定 60
Bean 開発者の役割 61
主キークラスの定義 61
リモートインタフェースの定義 61
リモートホームインタフェースの作成 61
findByPrimaryKey メソッド 63
リモートホームインタフェースの例 64
ローカルインタフェースの定義 64
ローカルホームインタフェースの作成 65
ローカルインタフェースの作成 65
リモートインタフェースの作成 67
Bean クラス定義の作成 (Bean 管理による持続性) 69
ejbCreate の使用 70
ejbActivate と ejbPassivate の使用 71
ejbLoad と ejbStore の使用 71
setEntityContext と unsetEntityContext の使用 73
ejbRemove の使用 74
検索メソッドの使用 74
読み取り専用 Beans の使用 75
読み取り専用 Bean の特性とライフサイクル 75
読み取り専用 Bean の開発に関する注意 76
読み取り専用 Bean の更新 76
トランザクションメソッドの起動 76
定期的な更新 77
プログラムによる更新 77
読み取り専用 Bean の配備 78
同時アクセスの同期化処理 78

第 4 章 エンティティ Beans のコンテナ管理による持続性の使用 79
Sun ONE Application Server でのサポート 80
コンテナ管理による持続性について 81
CMP コンポーネント 81
関係 82

一対一の関係 83
一対多の関係 83
多対多の関係 84
抽象スキーマ 84
配備記述子 85
持続性マネージャ 85
コンテナ管理による持続性の使用 86
プロセスの概要 87
フェーズ 1. マッピング配備記述子ファイルの作成 87
フェーズ 2. 具象 Beans および委託の生成とコンパイル 88
フェーズ 3. Sun ONE Application Server 実行時の稼動 88
マッピング機能 88
マッピングの各機能 89
マッピングツール 89
マッピングの技法 89
マッピングでサポートされているデータタイプ 90
BLOB のサポート 92
キャプチャスキーマユーティリティの使用 93
フィールドおよび関係のマッピング 94
マップする Beans の指定 95
マッピングコンポーネントの指定 96
フィールドマッピングの指定 99
関係の指定 101
リソースマネージャの設定 103
EJB QL の使用 105
1.1 検索のクエリの設定 105
クエリのフィルタ式 106
クエリのパラメータ 107
クエリ変数 107
サードパーティ製のプラグイン可能な持続性管理 API 109
制限事項と最適化 110
EAR ファイル内の一意のデータベーススキーマ名 110
コンテナ管理による持続性のプロトコルに関する制限事項 110
リモートインタフェースに関する制限事項 111
sun-cmp-mappings.xml ファイルの要素 111
例 121
スキーマ定義の例 121
CMP マッピング XML ファイルの例 122
EJB QL クエリの例 124

第 5 章 メッセージ駆動型 Beans の使用 129
メッセージ駆動型 Beans について 130
メッセージ駆動型 Beans の相違点 130

メッセージ駆動型 Beans の特性 131
トランザクション管理 131
メッセージの同時処理 132
メッセージ駆動型 Beans の開発 132
Bean クラス定義の作成 132
ejbCreate の使用 133
setMessageDrivenContext の使用 133
onMessage の使用 133
ejbRemove の使用 134
設定 135
接続ファクトリと送信先 135
メッセージ駆動型 Bean プール 136
サーバーインスタンス全体に適用される属性 136
JMS プロバイダへの自動再接続 137
制限事項と最適化 138
JMS に関する制限事項 138
プールの調整と監視 138
onMessage のランタイム例外 139
メッセージ駆動型 Beans XML ファイルの例 140
ejb-jar.xml ファイルの例 140
sun-ejb-jar.xml ファイルの例 141

第 6 章 Enterprise JavaBeans のトランザクション処理 143
JTA トランザクションと JTS トランザクションのサポート 144
トランザクション処理について 144
単層型トランザクション 145
グローバルトランザクションとローカルトランザクション 145
境界設定モデル 146
コンテナ管理トランザクション 146
Bean 管理トランザクション 147
コミットオプション 147
管理と監視 148
コンテナ管理トランザクションの使用法 149
トランザクション属性の指定 150
属性の必要条件の区別 151
属性値 151
コンテナ管理トランザクションのロールバック 154
セッション Beans のインスタンス変数の同期化 155
コンテナ管理トランザクションで使用できないメソッド 155
Bean 管理トランザクションの使用法 156
トランザクションタイプの選択 156
JDBC トランザクション 156
JTA トランザクション 157

コミットなしの復帰 157
Bean 管理トランザクションで使用できないメソッド 158
トランザクションタイムアウトの設定 158
遮断レベルの処理 159

第 7 章 安全な Enterprise JavaBeans の開発 161
安全な Enterprise JavaBeans について 162
承認と認証 162
セキュリティロール 162
配備 163
セキュリティロールの定義 163
メソッドパーミッションの宣言 164
セキュリティロール参照の宣言 165
セキュリティ ID の指定 166
実行 ID 166
プログラムによるセキュリティの使用法 167
保護されていない EJB 層のリソースの処理 168

第 8 章 Enterprise JavaBean のアセンブルと配備 169
EJB の構造 170
配備記述子の作成 170
Enterprise JavaBeans の配備 171
管理インタフェースの使用 172
コマンド行インタフェースの使用 172
Sun ONE Studio 4 IDE の使用 172
Enterprise JavaBeans の再読み込み 173
sun-ejb-jar_2_0-0.dtd ファイルの構造 174
サブ要素 174
データ 175
属性 175
sun-ejb-jar.xml ファイルの要素 176
一般的な要素 176
ロールマッピング要素 183
参照要素 185
メッセージング要素 190
セキュリティ要素 191
持続性要素 196
プールとキャッシュの要素 202
クラス要素 210
EJB XML ファイルの例 213
ejb-jar.xml ファイルの例 213
sun-ejb-jar.xml ファイルの例 214

付録 A Sun ONE Studio 4 インタフェースによる CMP のマッピング 217
CMP Beans のマッピング 217
スキーマの取り込み 217
既存の Enterprise JavaBeans とスキーマとのマッピング 219
関係フィールドのマッピング 223
EJB 持続性プロパティ 226

付録 B 要素一覧 229
sun-ejb-jar_2_0-0.dtd ファイルの要素 229
sun-cmp-mapping_1_0.dtd ファイルの要素 232

索引 233

本書について

この『Enterprise Java Beans 開発者ガイド』では、Sun™ ONE (オープンネットワーク環境) Application Server 7 環境で、Enterprise JavaBeans™ (EJB™) の仕様に準拠する Java 2 Platform Enterprise Edition (J2EE) アプリケーションを作成および実装する方法について説明します。このマニュアルでは、EJB プログラミングの概念およびタスクについて簡潔に説明し、さらに、コード例、実装に関するヒント、および参照資料も提供します。

この章では、次の項目について説明します。

- 対象読者
- マニュアルの使用法
- マニュアルの構成
- 関連情報
- マニュアルの表記規則
- 製品サポート

対象読者

このマニュアルは、企業内で Enterprise JavaBeans の開発、アセンブリ、および配備を担当する方々を対象としています。

このマニュアルでは、次の項目に精通していることを前提としています。

- Java プログラミング
- EJB、Java Server Page (JSP)、および Java Database Connectivity (JDBC) 仕様で定義されている Java API
- SQL 構造化データベースクエリ言語
- リレーショナルデータベースの概念

- デバッグ、ソースコード制御を含むソフトウェア開発プロセス

マニュアルの使用法

このマニュアルは、PDF 形式または HTML 形式でも入手できます。次のサイトを参照してください。

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

次の表は、Sun ONE Application Server のマニュアルに記述されているタスクと概念を示しています。左側の列にタスクと概念、右側の列に参照するマニュアルを示します。

Sun ONE Application Server マニュアルの概要

情報の内容	参照するマニュアル
ソフトウェアおよびマニュアルの最新情報	リリースノート
サポート対象のプラットフォームと環境	プラットフォーム
アプリケーションサーバーの紹介。新機能、評価 (Evaluation) バージョンのインストール、アーキテクチャの概要など	入門ガイド
Sun ONE Application Server とそのコンポーネント (サンプルアプリケーション、管理インタフェース、Sun ONE Message Queue など) のインストール	インストールガイド
Sun ONE Application Server 7 の Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。アプリケーション設計、開発ツール、セキュリティ、アセンブリ、配備、デバッグ、ライフサイクルモジュールの作成に関する情報など	開発者ガイド
Sun ONE Application Server 7 の Web アプリケーション向け Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。Web アプリケーションプログラミングの概念とタスクの説明、サンプルコード、実装のヒント、関連資料の紹介など	Web アプリケーション開発者ガイド
Sun ONE Application Server 7 の エンタープライズ Beans 向け Java オープンスタンダードモデルに準拠した J2EE アプリケーションの作成方法と実装方法。EJB プログラミングの概念とタスクの説明、サンプルコード、実装のヒント、関連資料の紹介など	Enterprise JavaBeans 開発者ガイド

Sun ONE Application Server マニュアルの概要 (続き)

情報の内容	参照するマニュアル
Web サービス、RMI-IIOP、Sun ONE Application Server 7 上の J2EE アプリケーションにアクセスするその他のクライアントの作成方法	Developer's Guide to Clients
JDBC、JNDI、JTS、JMS、JavaMial、リソース、コネクタなどの J2EE 機能	Developer's Guide to J2EE Features and Services
カスタム NSAPI プラグインの作成方法	NSAPI Developer's Guide
次の管理タスクの実行 • 管理インタフェースとコマンド行インタフェースの使用 • サーバーの作業環境の構成 • 管理ドメインの使用 • サーバーインスタンスの使用 • サーバーの稼動状況の監視およびログ記録 • Web サーバープラグインの構成 • Java Messaging Service の構成 • J2EE 機能の使用 • CORBA ベースのクライアント機能の構成 • データベース接続性の構成 • トランザクション管理の構成 • Web コンテナの構成 • アプリケーションの配備 • 仮想サーバーの管理	管理者ガイド
サーバー構成ファイルの編集	管理者用構成ファイルリファレンス
Sun ONE Application Server 7 本稼動環境のセキュリティの設定および管理。一般的なセキュリティ、証明書、SSL/TLS 暗号化に関する情報、Web コアベースのセキュリティの説明など	セキュリティ管理者ガイド
Sun ONE Application Server 7 で利用する J2EE CA コネクタのサービスプロバイダ実装の設定と管理。管理ツール、DTD に関する情報、サンプル XML ファイルなど	J2EE CA Service Provider Implementation Administrator's Guide

Sun ONE Application Server マニュアルの概要 (続き)

情報の内容	参照するマニュアル
Netscape Application Server バージョン 2.1 から新しい Sun ONE Application Server 7 プログラミングモデルへのアプリケーションの移行。Sun ONE Application Server に付属するオンラインバンクアプリケーションの移行サンプルなど	サーバーアプリケーションの移行および再配備
Sun ONE Message Queue の使用法	Sun ONE Message Queue については次の URL を参照 http://docs.sun.com/

マニュアルの構成

このマニュアルは、次のような内容で構成されています。

- 「Sun ONE Application Server Enterprise JavaBeans の紹介」
- 「セッション Beans の使用」
- 「エンティティ Beans の使用」
- 「エンティティ Beans のコンテナ管理による持続性の使用」
- 「メッセージ駆動型 Beans の使用」
- 「Enterprise JavaBeans のトランザクション処理」
- 「安全な Enterprise JavaBeans の開発」
- 「Enterprise JavaBean のアセンブルと配備」
- 「Sun ONE Studio 4 インタフェースによる CMP のマッピング」
- 「要素一覧」

関連情報

12 ページの「マニュアルの使用法」に示された Sun ONE Application Server のマニュアルの情報に加えて、次のリソースを利用できます。

- J2EE 仕様

  `http://java.sun.com/products/`

- Enterprise JavaBeans Specification, Version 2.0

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

- 一般的な EJB 製品情報

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

- Java Software チュートリアル

  `http://java.sun.com/j2ee/docs.html`

- 『Enterprise JavaBeans』、Richard Monson-Haefel 著、O'Reilly 発行
  ISBN 0-596-00226-2

  `http://www.oreilly.com/catalog/entjbeans3/`

- Enterprise JavaBean テクノロジに関する書籍リスト

  http://developer.java.sun.com/developer/Books/ejbtechnology.html

- Enterprise JavaBeans Design Patterns, ISBN 0-471-20831-0

- Core J2EE Patterns, ISBN 0-13-064884-1

マニュアルの表記規則

ここでは、このマニュアル全体に適用される表記規則について説明します。

- 一般的な表記規則
- ディレクトリ名の表記規則

一般的な表記規則

このマニュアルに適用される一般的な表記規則は次のとおりです。

- **ファイルとディレクトリのパス**は、UNIX® の形式で表記します (ディレクトリ名を「/」記号で区切って表記)。Windows バージョンでは、ディレクトリパスについては UNIX と同じですが、ディレクトリの区切り記号には「/」記号ではなく「¥」記号を使用します。
- **URL** は次のように表記されます。

  http://*server*.*domain*/*path*/*file*.html

  これらの URL で、*server* はアプリケーションを実行するサーバー名で、*domain* はユーザーのインターネットドメイン名、*path* はサーバー上のディレクトリの構造、*file* は個別のファイル名を示します。URL の斜体文字の部分は可変部分です。
- **フォント**は次の規則に従います。
  - モノスペースフォントは、サンプルコード、コードの一覧表示、API および言語要素 (関数名、クラス名など)、ファイル名、パス名、ディレクトリ名、および HTML タグに使います。
  - 斜体文字はコード変数に使います。
  - また斜体文字は、変数および可変部分、およびリテラルに使われる文字にも使います。
  - **太字**は、段落の先頭文字またはリテラルに使われる文字の強調に使います。
- このマニュアルでは、ほとんどのプラットフォームの**インストールルートディレクトリ**を *install_dir* と記述します。例外については、17 ページの「ディレクトリ名の表記規則」を参照してください。

  デフォルトでは、**ほとんど**のプラットフォームの *install_dir* は次の位置になります。
  - パッケージに含まれない Solaris 8 の評価 (Evaluation) バージョン

    user's home directory`/sun/appserver7`

- Solaris に含まれない、評価用以外のバージョン

  /opt/SUNWappserver7

- Windows のすべてのインストール

  C:¥Sun¥AppServer7

上記プラットフォームでは、*default_config_dir* と *install_config_dir* は *install_dir* と同じ意味です。これ以外の説明と例外については、17 ページの「ディレクトリ名の表記規則」を参照してください。

- このマニュアルでは、**インスタンスルートディレクトリ**を *instance_dir* と記述します。これは、実際には次のディレクトリを示しています。

  default_config_dir/domains/*domain*/*instance*

- このマニュアルを通じて、特に明記のない限り、**UNIX 固有の表記**は、Linux オペレーティングシステムにも適用されます。

ディレクトリ名の表記規則

デフォルトでは、Solaris 8 および 9 パッケージベースのインストールおよび Solaris 9 バンドル版のインストールを使用すると、アプリケーションサーバーファイルはいくつかのルートディレクトリに分散してインストールされます。この節では、これらのディレクトリについて説明します。

- **Solaris 9 バンドル版のインストール**のデフォルトのインストールディレクトリは、次のように表記します。
  - *install_dir*: /usr/appserver/ を表します。このディレクトリには、インストールイメージの静的な部分が格納されます。アプリケーションサーバーを構成するすべてのユーティリティ、実行可能ファイル、およびライブラリがここに格納されます。
  - *default_config_dir*: 作成されたドメインのデフォルトの格納先となる /var/appserver/domains ディレクトリを表します。
  - *install_config_dir*: /etc/appserver/config ディレクトリを表します。このディレクトリには、インストール全体の設定情報が格納されます。たとえば、このインストールのライセンス、管理ドメインのマスターリストなどが格納されます。
- **Solaris 8 および 9 パッケージベースの、バンドルされていない評価版以外のインストール**ではデフォルトのディレクトリは、次のように表記します。

- *install_dir*: /opt/SUNWappserver7 ディレクトリを表します。このディレクトリには、インストールイメージの静的な部分が格納されます。アプリケーションサーバーを構成するすべてのユーティリティ、実行可能ファイル、およびライブラリがここに格納されます。
- *default_config_dir*: 作成されたドメインの格納先となる /var/opt/SUNWappserver7/domains ディレクトリを表します。
- *install_config_dir*: /etc/opt/SUNWappserver7/config ディレクトリを表します。このディレクトリには、インストール全体の設定情報が格納されます。たとえば、このインストールのライセンス、管理ドメインのマスターリストなどが格納されます。

注 Forte for Java 4.0 は、名称が変更されました。このマニュアルでは Sun ONE Studio 4 と表記されます。

製品サポート

ご使用のシステムに問題が発生した場合は、次のいずれかの方法でカスタマサポートにお問い合わせください。

- 次のオンラインサポート Web サイトをご利用ください。

  http://www.sun.com/supportraining/

- 保守契約を結んでいるお客様の場合は、専用ダイヤルをご利用ください。

連絡の前に、次の情報を準備してください。テクニカルサポートスタッフが問題解決のお手伝いする上で、この情報が役立ちます。

- 問題が発生した箇所や動作への影響など、問題の具体的な説明
- マシン機種、OS バージョン、および、問題の原因と思われるパッチやそのほかのソフトウェアなどの製品バージョン
- 問題を再現するための具体的な手順の説明
- エラーログやコアダンプ

第 1 章

Sun ONE Application Server Enterprise JavaBeans の紹介

この章では、Sun™ ONE Application Server 7 のアプリケーションプログラミングモデルにおける Java Enterprise Edition (J2EE) Enterprise JavaBeans™ (EJB™) テクノロジの働きの概要について説明します。

注 EJB テクノロジに精通していない場合は、Java Software チュートリアルを参照してください。

`http://java.sun.com/j2ee/docs.html`

また、J2EE 仕様も参照してください。

`http://java.sun.com/products/`

Sun ONE Application Server の概要は、『Sun ONE Application Server 製品の概要』にあります。

この章には次の項目があります。

- EJB 2.0 の変更点の概要
- EJB のアーキテクチャ
- 付加価値機能
- Enterprise JavaBeans について
- 効率的なアプリケーションの開発について
- EJB のアセンブリと配備について

Sun ONE Application Server に付属する関連ファイルは、次の場所にあります。

- Sun ONE Application Server DTD ファイル

  install_dir/`apperv/lib/dtds`

- Sun ONE Application Server サンプルアプリケーション

  install_dir/apperv/samples

EJB 2.0 の変更点の概要

Sun ONE Application Server は、『Enterprise JavaBeans Specification, v2.0』で定義された Sun Microsystems Enterprise JavaBeans (EJBs) アーキテクチャをサポートしており、『Enterprise JavaBeans Specification v1.1』に準拠しています。

注 既存の 1.1 Enterprise JavaBean を Sun ONE Application Server に配備できますが、2.0 Enterprise JavaBean として新しい Bean を開発することをお勧めします。

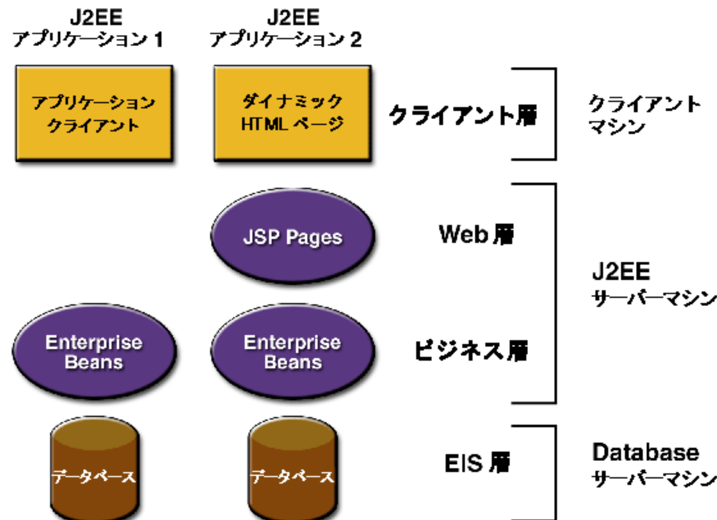
この節では、Sun ONE Application Server 環境の Enterprise JavaBean に影響を及ぼす『Enterprise JavaBeans Specification, v2.0』の変更点の概要を示します。

- コンテナ管理による持続性 - コンテナ管理による持続性の新しい処理方法を提供する。79 ページの「エンティティ Beans のコンテナ管理による持続性の使用」を参照
- コンテナ管理による関係 - エンティティ Beans 間の関係を定義できる。169 ページの「Enterprise JavaBean のアセンブルと配備」を参照
- メッセージ駆動型 Beans - この新しいタイプの EJB は Java Message Service コンシューマである。129 ページの「メッセージ駆動型 Beans の使用」を参照
- ローカルインタフェース - セッション Beans、エンティティ Beans がローカルインタフェースを実装できる。このため、コンテナ管理される EJB 関係は、ローカルインタフェースに基づく。65 ページの「ローカルインタフェースの作成」を参照
- ホームインタフェースの追加メソッド - 特定のエンティティ Beans インスタンスに依存しないビジネスロジックを実装できる。61 ページの「リモートホームインタフェースの作成」を参照
- 新しいクエリ言語 (EJB QL) - 新しい EJB クエリ言語 (EJB QL) により、コンテナ管理関係で定義されたエンティティ Beans のネットワークでのナビゲーションを提供する。105 ページの「EJB QL の使用」を参照

EJB のアーキテクチャ

Sun ONE Application Server を使用すると、トランザクション、セキュリティ、データベース接続などのミドルウェアサービスのサポートを自動化することによって、ミドルウェア開発の複雑さが軽減されます。

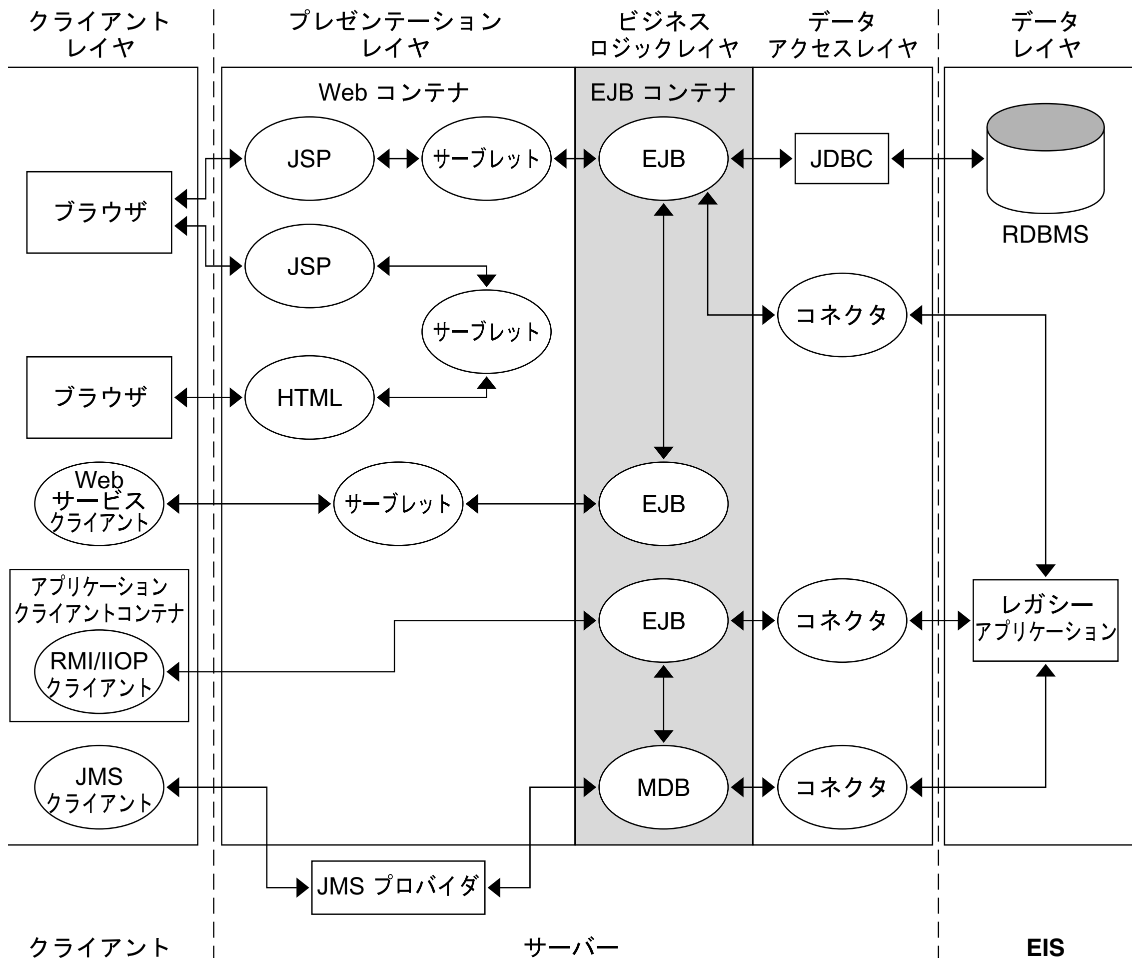
次の図は、Enterprise JavaBean が J2EE 環境のどこに適応するかを示しています。この図では、クライアントマシンで Web ブラウザまたはアプリケーションクライアントを稼動し、J2EE サーバーマシンでは Sun ONE Application Server を稼動 (ホスト) し、データベースサーバーマシンでは Oracle や LDAP などのデータベースをホストします。Enterprise JavaBeans はビジネス層にあり、JSP (およびサーブレット) がクライアント層にインタフェースを提供し、Sun ONE Application Server がクライアントマシンとデータベースマシンの関係を管理しています。



Sun ONE Application Server は、次のような EJB 実行システムの基盤を提供します。

- EJB サービスの標準セット
- 分散トランザクション管理サービス
- データストアアクセスまたはバックエンドシステム接続の手段
- EJB クラスの管理サービスおよび制御サービスを実装するための EJB コンテナ

次の図は、J2EE 環境をさらに詳細に示しています。ビジネスロジック層が EJB フローを示します。


クライアント
レイヤ
プレゼンテーション
レイヤ
ビジネス
ロジックレイヤ
データ
アクセスレイヤ
データ
レイヤ
Web コンテナ
EJB コンテナ
ブラウザ
JSP
サーブレット
EJB
JDBC
RDBMS
JSP
コネクタ
サーブレット
ブラウザ
HTML
Web
サービス
クライアント
サーブレット
EJB
アプリケーション
クライアントコンテナ
RMI/IIOP
クライアント
EJB
コネクタ
レガシー
アプリケーション
JMS
クライアント
MDB
コネクタ
JMS プロバイダ
クライアント
サーバー
EIS

付加価値機能

Sun ONE Application Server は、EJB 開発に関する付加価値機能を提供します。これらの機能については、次の各項目で説明します (詳細な資料への参照も含まれる)。

- 読み取り専用 Beans
- pass-by-reference
- プール機能とキャッシュ機能
- 監視
- Sun ONE Studio 4 との統合
- 動的な配備と再読み込み

読み取り専用 Beans

Sun ONE Application Server には、読み取り専用 Bean を作成する機能があります。このエンティティ Bean は、EJB クライアントによって変更されません。読み取り専用 Beans を使うことで、データベースが完全に更新されることを防げます。

読み取り専用 Bean は、頻繁にアクセスされるが、他の Beans によって外部から更新される頻度の低いデータベースエントリのキャッシュに利用できます。読み取り専用 Bean によってキャッシュされたデータが別の Bean によって更新される場合は、キャッシュデータを更新することを読み取り専用 Bean に知らせることができます。

Sun ONE Application Server には、読み取り専用 Bean の状態を更新する機能がいくつも用意されています。Bean の `refresh-period-in-seconds` 要素とトランザクション属性を設定することで、読み取り専用 Bean を簡単に次のように設定できます。(a) 常に更新される、(b) 定期的に更新される、(c) 更新されない、または (d) プログラムによって更新される。

読み取り専用 Bean は、基になるデータがまったく変更されないか、まれにしか変更されない場合にもっとも適しています。詳細および使用方法のガイドラインについては、59 ページの「読み取り専用 Beans」を参照してください。

pass-by-reference

`sun-ejb-jar.xml` ファイルの `pass-by-reference` 要素を使うことで、Enterprise JavaBeans に適用する受け渡し方法と引数のタイプを指定できます。これにより、パフォーマンスを向上させることができます。186 ページの「pass-by-reference」を参照してください。

プール機能とキャッシュ機能

Sun ONE Application Server には、高度な設定に対応したプールメカニズムが用意されています。これにより、配備担当者はニーズに適した Bean プールを設定できます。

さらに、Sun ONE Application Server には調整可能な多数のパラメータが用意されているので、キャッシュする Beans の数や、Beans をキャッシュしておく時間を制御できます。同じデータベース行を参照する複数の Bean インスタンスをキャッシュできます。

この機能については、31 ページの「プールとキャッシュ」を参照してください。

監視

Sun ONE Application Server では、実行時環境をさまざまな観点から監視できます。EJB コンテナ内のさまざまな要素を利用できるので、アプリケーションのデバッグやパフォーマンスのチューニングに役立ちます。

監視については、『Sun ONE Application Server 管理者ガイド』(「Monitoring and Managing Sun One Application Server section」) および『Performance, Tuning, and Sizing Guide』を参照してください。

Sun ONE Studio 4 との統合

Sun ONE Studio 4 (従来の Forte for Java (FFJ), Enterprise Edition) は、Sun ONE Application Server でのコードの作成、アセンブリ、配備およびデバッグを、単一の使いやすいインタフェースから行うことができる統合開発環境 (IDE) です。Sun ONE Studio 4 IDE と Sun ONE Application Server は、プラグインにより統合されています。

Sun ONE Studio 4 の使用については、Sun ONE Studio 4, Enterprise Edition のチュートリアルおよび 217 ページの「Sun ONE Studio 4 インタフェースによる CMP のマッピング」を参照してください。

動的な配備と再読み込み

アプリケーションまたはスタンドアロンモジュールを配備、再配備、および配備取り消しできます。サーバー稼動中にこれらを行う場合、動的とみなされます。Sun ONE Application Server では、次の動的プロセスを使用できます。

- 動的再読み込み - アプリケーションを構成するクラスがディスク上で変更された場合に、変更されたクラスを再読み込みできる
- 動的再配備 (開発者コミュニティ向け) - サーバーを再起動せずに既存のアプリケーションを再配備できる。また、アプリケーションまたはモジュールを配備取り消しせずに、無効または有効にすることができる

動的な配備の詳細は、『Sun ONE Application Server 開発者ガイド』および『管理者ガイド』を参照してください。

Enterprise JavaBeans について

EJB およびその動作について十分な知識がある場合は、35 ページの「効率的なアプリケーションの開発について」に進んでください。

この節では次の項目について説明します。

- Enterprise JavaBean とは
- Beans の種類
- EJB フロー
- EJB コンテナ
- インタフェース
- プールとキャッシュ
- Enterprise JavaBeans がリソースにアクセスするしくみ
- トランザクション管理
- アプリケーションセキュリティのしくみ

Enterprise JavaBean とは

Enterprise JavaBean (EJB) は、データメンバー、プロパティ、およびメソッドを持つ自己完結型の再利用可能なコンポーネントです。各 Enterprise JavaBean は、データ構造体やオペレーションメソッドなどの 1 つまたは複数のアプリケーションタスク、またはアプリケーションオブジェクトをカプセル化します。

- Enterprise JavaBean メソッドはパラメータを受け取って戻り値を返す
- Enterprise JavaBean の作成と管理は、コンテナによって実行時に行われる
- クライアントアクセスの仲介は、Bean が配備されたコンテナおよびサーバーによって処理される
- Enterprise JavaBeans は、『Enterprise JavaBeans Specification, v2.0』で定義された標準のコンテナサービスのみを使用できる。このため、EJB 対応のすべてのコンテナで Bean の移植および配備が可能である
- Enterprise JavaBeans は、再コンパイルなしで複合アプリケーションにアセンブルできるコンポーネントである
- クライアントの Bean 定義ビューは、Bean 開発者によって完全に制御されるビューは、Bean が動作するコンテナや Bean が配備されたサーバーの影響を受けない

Enterprise JavaBeans を利用すると大規模な分散アプリケーションの開発が簡単になります。その理由は次のとおりです。

- コンテナ提供サービス - EJB コンテナはシステムレベルのサービスを Enterprise JavaBeans に提供するので、Bean 開発者はビジネス問題の解決に専念できる。Bean 開発者ではなく EJB コンテナが、トランザクション管理やセキュリティ認証などのシステムレベルのサービスを受け持つ
- リモートクライアント - クライアントではなく EJB にアプリケーションのビジネスロジックが含まれるので、クライアント開発者はクライアントのプレゼンテーションに専念できる。クライアント開発者は、ビジネスルールの実装やデータベースアクセスのためのルーチンをコーディングする必要がない。その結果、クライアントは軽量になる。これは、小型デバイスで動作するクライアントでは特に重要な利点である
- Bean の再利用性 - Enterprise JavaBeans は移植可能なコンポーネントであるため、アプリケーションのアセンブリ担当者は、新しいアプリケーションを既存の Beans から構築できる。これらのアプリケーションは、J2EE に準拠するすべてのサーバーで動作する

Beans の種類

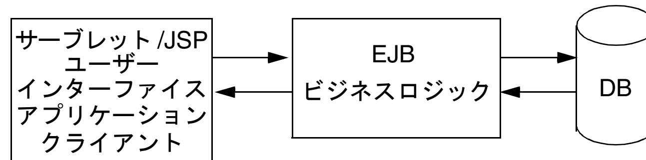
Enterprise JavaBean には次の 3 つの種類があります。

- セッション Bean (ステートフル、またはステートレス)
  - ステートフルセッション Bean は、編集するドキュメントのコピーや、個々のクライアントに固有のビジネスオブジェクトなど、起動全体で状態が維持されるオブジェクトおよびプロセスを表す
  - ステートレスセッション Bean は、起動全体で状態が維持されない特定のクライアントが必要とするビジネスロジックの一部を一時的にカプセル化する
  - セッション Beans の開発については、41 ページの「セッション Beans の使用」を参照
- エンティティ Bean - エンティティ Bean は通常、持続データを表す。このデータは、データベースで直接管理されるか、またはオブジェクトとして Enterprise Information System (EIS) アプリケーションからアクセスされる
  - Bean 管理による持続性 - Bean の持続性を Bean 自身が管理する。エンティティ Beans のコードには、データベースにアクセスする呼び出しを記述する。エンティティ Beans 開発の概要、Bean 管理による持続性の開発については、55 ページの「エンティティ Beans の使用」を参照
  - コンテナ管理による持続性 - Enterprise JavaBean コンテナが、持続マネージャを介して、エンティティ Bean が要求するすべてのデータベースアクセスを処理する。コンテナ管理による持続性については、79 ページの「エンティティ Beans のコンテナ管理による持続性の使用」を参照
- メッセージ駆動型 Bean - メッセージ駆動型 Bean はステートレスサービスを表す。これは JMS によって起動される、基本的には完全に匿名で、クライアントに見える識別情報を持たない非同期メッセージコンシューマである

  メッセージ駆動型 Beans の開発については、129 ページの「メッセージ駆動型 Beans の使用」を参照してください。

EJB フロー

ユーザーがブラウザから Sun ONE Application Server Servlet を呼び出すと、そのサーブレットが 1 つまたは複数の Enterprise JavaBeans を呼び出します。たとえば、サーブレットはユーザーのブラウザに JavaServer Page (JSP) を読み込んでユーザー名とパスワードを要求し、さらに入力された内容をセッション Beans に渡して確認します。



有効なユーザー名とパスワードの組み合わせを受け取ると、サーブレットは 1 つまたは複数のエンティティ Beans およびセッション Beans をインスタンス化してアプリケーションのビジネスロジックを実行し、その後で終了します。Bean 自体もほかのエンティティ Beans またはセッション Beans をインスタンス化して、ビジネスロジックおよびデータ処理をさらに行います。

サンプルシナリオ

サーブレットは、顧客サービス担当者が注文データベースにアクセスするためのセッション Bean を起動します。このアクセスでは、次の機能が必要です。

- データベースをブラウズする
- 購入項目をキューに入れる
- 顧客からの注文を受け付ける
- データベース内の部品数を減らす
- 顧客に請求書を発行する
- 在庫が不足した場合に部品を追加注文する

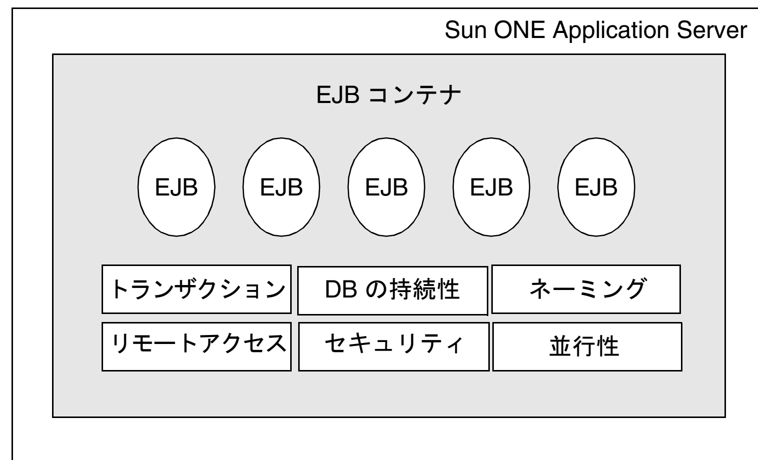
顧客注文プロセスの一環として、サーブレットは「ショッピングカート」を管理するセッション Bean を作成し、顧客が選択したアイテムを一時的に記録します。注文が完了すると、ショッピングカートのデータは注文データベースに転送され、ショッピングカートのセッション Bean が解放されます。

EJB コンテナ

Enterprise JavaBeans は、常にコンテナのコンテキスト内で動作します。コンテナは、Enterprise JavaBeans とホストサーバーとの間でリンクのように機能します。EJB コンテナによって、ユーザー独自のコンポーネントやほかの供給元に提供されたコンポーネントを使った分散アプリケーションを構築できます。

Sun ONE Application Server では、コンテナを通して高レベルのトランザクション管理、セキュリティ管理、状態管理 (持続性)、マルチスレッド、およびリソースプールラッパーを提供するので、ユーザーが低レベル API の詳細を理解する必要はありません。コンテナによって並行性が確保されるため、エンティティ (つまりスレッド) が同時に 1 つの Enterprise JavaBean にアクセスする心配がありません。このコンテナは、『Enterprise JavaBeans Specification, v2.0』に規定されているすべての標準コンテナサービスに加えて、Sun ONE Application Server に固有のサービスも提供します。

Sun ONE Application Server サービスには、リモートアクセス、ネーミング、セキュリティ、並行性、トランザクションコントロール、データベースアクセスなどがあります。次の図は、Sun ONE Application Server で提供される EJB コンテナを示します。



インタフェース

クライアントがセッション Beans またはエンティティ Beans のインスタンスに直接アクセスすることはありません。クライアントは Beans のリモートインタフェースを使って Bean インスタンスにアクセスします。Beans のリモートインタフェースを実装する EJB オブジェクトクラスは、コンテナによって提供されます。

ホームインタフェース

ホームインタフェースは、クライアントが EJB の作成、破棄、および検索を行うためのメカニズムを提供します。EJB は、EJB 仕様に定義されている `javax.ejb.EJBHome` インタフェースを拡張するコンテナに、ホームインタフェースを提供します。ホームインタフェースは、そのもっとも基本的な機能として、Bean を作成する各方法に関連する 0 または 1 つ以上の `create` メソッドを定義します。

エンティティ Beans では、1 つの Bean または Bean の集まりの検索に使用可能な方法ごとに検索メソッドを定義する必要があります。

リモートインタフェース

リモートインタフェース (およびリモートホームインタフェース) は、リモートクライアントがセッション Beans またはエンティティ Beans にアクセスするメカニズムを提供します。リモートクライアントは、同じコンテナまたは別のコンテナで配備される別の EJB でも、アプリケーション、アプレット、サーブレットなどの Java プログラムでもかまいません。EJB のリモートクライアントビューは、場所に依存せず、非 Java クライアント環境にマップできます。

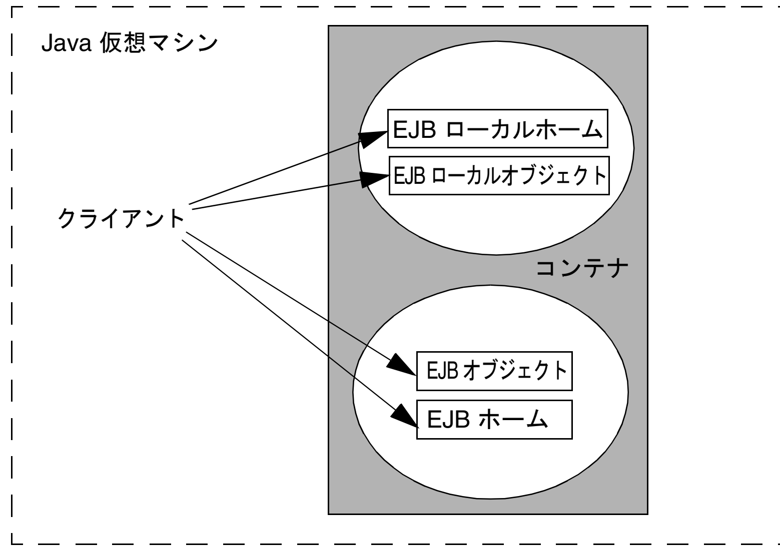
リモートホームインタフェースは、EJB 開発者によって定義され、EJB コンテナによって実装されます。

ローカルインタフェース

ローカルインタフェース (およびローカルホームインタフェース) は、セッション Bean またはエンティティ Bean を持つ同じ Java 仮想マシン (JVM) に配備されたクライアントが、その Bean にアクセスするメカニズムを提供します。これにより、ローカルクライアントビューを提供します。ローカルクライアントは、関連付けられた Bean に密結合される場合があり、セッション Beans およびエンティティ Beans は多数のローカルクライアントを持つことができます。

コンテナは、ローカルホームインタフェースおよびローカルインタフェースを実装するクラスを提供します。これらのインタフェースを実装するオブジェクトは、ローカル Java オブジェクトです。EJB のローカルクライアントビューは、場所に依存します。

次の図は、ローカルインタフェースを介してローカルクライアントが接続するコンテナの 2 つの Enterprise JavaBeans を示します。



開発時にローカルインタフェースを Bean で定義すると、同じコンテナ内からの呼び出しの場合、その Bean への効率的な呼び出しが可能になります。

プールとキャッシュ

Sun ONE Application Server の EJB コンテナは、オブジェクトの作成と削除によるオーバヘッドを削減するために、匿名のインスタンス (メッセージ駆動型 Beans、ステートレスセッション Beans、エンティティ Beans) をプールします。EJB コンテナは、配備される各 Bean の空きプールを維持します。空きプール内の Bean インスタンスには識別情報が含まれず (つまり、主キーが関連付けられていません)、ホームインタフェースのメソッド呼び出しの処理に利用されます。ステートレスセッション Beans では、すべてのメソッドの処理に空き Beans が使われます。

`ejbCreate` メソッドとビジネスメソッドが実行されると、空きプール内の Bean インスタンスの状態は、「Pooled」から「Cashed」に変わります。各プールのサイズと動作は、`server.xml` ファイルと `sun-ejb-jar.xml` ファイルに記録されるプール関連プロパティを使って制御できます。

EJB コンテナは、パフォーマンスを向上させるために「ステートフル」インスタンス (ステートフルセッション Beans とエンティティ Beans) をメモリにキャッシュします。EJB コンテナは、配備される各 Bean のキャッシュを維持します。

スケーラビリティを実現するために、コンテナはキャッシュがオーバーフローした場合などに選択的に一部の Bean インスタンスを削除します。削除された Bean インスタンスは、Bean の空きプールに戻ります。各キャッシュのサイズと動作は、`server.xml` ファイルと `sun-ejb-jar.xml` ファイルに記録されるキャッシュ関連プロパティを使って制御できます。

`sun-ejb-jar.xml` ファイルに記録されるプール関連およびキャッシュ関連のパラメータについては、202 ページの「プールとキャッシュの要素」を参照してください。

プール関連パラメータ

Sun ONE Application Server のプール機能で最も重要なパラメータの一つが `steady-pool-size` です。`steady-pool-size` に 0 より大きな値を指定すると、コンテナは指定数の Beans でプールを満たすだけでなく、空きプール内の Beans の数を常に同数に維持しようとします。これにより、ユーザーからの要求を処理する準備が整った Beans を確保できます。

別のパラメータである `pool-idle-timeout-in-seconds` を使用すると、Bean インスタンスがプール内でアイドル状態でいられる時間を指定できます。`pool-idle-timeout-in-seconds` に 0 より大きな値を設定すると、指定時間が経過してもアイドル状態を続ける Bean インスタンスが削除されます。

キャッシュ関連パラメータ

Sun ONE Application Server では、コミット C オプションを使ってエンティティ Beans のキャッシュを完全に回避できます。コミット C オプションは、アクセスは多いが再利用の少ない Beans に特に適しています。詳細は、147 ページの「コミットオプション」を参照してください。

Sun ONE Application Server のキャッシュは、有限、無制限のいずれかにすることができます。有限キャッシュでは、保持できる Beans の数に制限があり、それを超えると Beans は非活性化されます。ステートフルセッション Beans では、キャッシュがオーバーフローしたときにどの Beans を削除するかを 3 つの方法 (LRU、NRU、FIFO) で指定できます。一定期間アクセスされないアイドル状態の Beans を非活性化するようにキャッシュを設定することもできます。

Enterprise JavaBeans がリソースにアクセスするしくみ

Enterprise JavaBeans は、データベース、JavaMail セッション、JMS オブジェクト、URL など、さまざまな種類のリソースにアクセスできます。J2EE プラットフォームは、それらのリソースすべてに同様の方法でアクセスできるメカニズムを提供します。

この節では次の項目について説明します。

- JNDI 接続
- データベースへの接続
- URL への接続

JNDI 接続

J2EE コンポーネントは、JNDI (Java Naming and Directory Interface) API の `lookup` メソッドを呼び出して、アクセスが必要なオブジェクトを特定します。この呼び出しによって返される値は、呼び出し元がアクセスすべきオブジェクトを表します。Enterprise JavaBeans では、`lookup` を呼び出すと、その Bean のホームインタフェースを参照するオブジェクトが返されます。この参照は、EJB ホームインタフェースのすべての機能の呼び出しにも使用されます。

```
Context initial = new InitialContext();
      Object objref
initial.lookup("java:comp/env/ejb/CompString");
```

配備した Enterprise JavaBean へのアクセスが必要なサーバーの J2EE コンポーネント (JSP、サーブレット、または Enterprise JavaBean) は、配備記述子内の EJB 参照要素を使ってこのアクセスを指定します。EJB 参照は、配備時にアクセス先の Enterprise JavaBean に対応する JNDI 名にマップされます。このマッピングには、Enterprise JavaBeans にアクセスするコンポーネントと、アクセスされる Beans の JNDI 名とを切り離す働きがあります。このため、EJB のホームと結ばれる JNDI 名は、配備時に変更される可能性がありますが、呼び出し元のコードを変更する必要はありません。

データベースへの接続

Enterprise JavaBean の持続タイプによって、データベースにアクセスするための接続ルーチンをコーディングするかどうかが決まります。

- データベースにアクセスするがコンテナ管理による持続性を使用しない Beans - 開発者が持続コードを記述する必要がある。Beans 管理による持続性を使用するエンティティ Beans、およびセッション Beans がこれに含まれる
- コンテナ管理による持続性を使用するエンティティ Beans - コネクションルーチンは、配備時に生成される。エンティティ Beans だけに適用される

URL への接続

Uniform Resource Locator (URL) は、Web ページなど、Web 上のリソースの場所を示します。URL は JNDI 名にマップされ、開発者はこの URL を検索できます。

トランザクション管理

アプリケーションの処理をトランザクションと呼ばれる単位に分割すると、データベースのフェール復旧や整合性の維持など、複雑な問題を処理する必要がなくなります。

開発者は、EJB コード内のプログラムによるトランザクションの境界設定 (Bean 管理) か、または宣言による境界設定 (コンテナ管理) のどちらかを使用できます。Enterprise JavaBean で Bean 管理とコンテナ管理のどちらのトランザクション境界設定を使用するかに関係なく、EJB コンテナおよび Sun ONE Application Server にトランザクション管理の負荷がかかります。このコンテナとサーバーは、必要な低レベルのトランザクションプロトコル (トランザクションマネージャと、データベースシステムまたは Sun ONE Message Queue プロバイダの間の 2 階層コミットプロトコルなど) を実装します。

トランザクション処理の詳細については、143 ページの「Enterprise JavaBeans のトランザクション処理」を参照してください。

アプリケーションセキュリティのしくみ

J2EE アプリケーションプログラミングモデルを利用すると、開発者は、各メカニズムのアプリケーションセキュリティに関する実装を詳細に扱う必要がありません。ほとんどの部分では、コンテナはセキュリティインフラストラクチャを実装できます。J2EE では、アプリケーションの移植性を高めて、コードを追加せずにさまざまなセキュリティ環境にアプリケーションを配備できるようにしています。

アプリケーションで使用される、宣言によるセキュリティメカニズムは、配備記述子に示されます。配備するときは、Sun ONE Application Server の専用ツールを使用して、配備記述子内のアプリケーション要件を、コンテナが実装するセキュリティメカニズムに対応付けます。

詳細については、161 ページの「安全な Enterprise JavaBeans の開発」を参照してください。セキュリティのレルムについては、『Sun ONE Application Server 開発者ガイド』を参照してください。

効率的なアプリケーションの開発について

Sun ONE Application Server アプリケーションのビジネスロジックおよびデータ処理を、もっとも効率的な組み合わせ (サーブレット、JSP、セッション Beans、エンティティ Beans、およびメッセージ駆動型 Beans) に分割することは、開発者の重要な仕事です。Enterprise JavaBeans を使ったオブジェクト指向の設計には特別なルールはありませんが、エンティティ Bean のインスタンスは生存期間が長く、持続的で、複数のクライアント間で共有されるものにし、セッション Bean のインスタンスは生存期間の短く、1 つのクライアントだけに使われるものにします。メッセージ駆動型 Beans は、唯一の非同期の受け手として独自のカテゴリに入ります。

一般に、複数のアプリケーションおよびクライアント間で Enterprise JavaBeans を共有して、アプリケーションを複数のサーバー間に容易に配備することと実行速度とのバランスがとれた、Sun ONE Application Server アプリケーションを作成することを目標とします。

Sun ONE Application Server 環境での Enterprise JavaBeans の開発に役立つように、高レベルな情報およびガイドラインを次の各項で説明します。

- Enterprise JavaBeans 作成の一般的なプロセス
- Bean 使用の手引き
- クライアントビューの手引き
- リモートインタフェースとローカルインタフェースの手引き
- Sun ONE Application Server 機能へのアクセス

Enterprise JavaBeans 作成の一般的なプロセス

ここでは、Enterprise JavaBean 作成の一般的なプロセスを説明します。さまざまな種類の Enterprise JavaBeans を作成する具体的な手順は、この項に記載されている参照箇所にあります。

Enterprise JavaBean の作成手順は次のとおりです。

1. すべての Enterprise JavaBean ファイル用のディレクトリを作成します。
2. 作成する Enterprise JavaBean のタイプを決定します。
    - セッション Bean (45 ページの「セッション Beans の開発」を参照)
        - ステートフル
        - ステートレス
    - エンティティ Bean (60 ページの「エンティティ Beans の開発」を参照)

- Bean 管理による持続性を使用する Bean
- コンテナ管理による持続性を使用する Bean (86 ページの「コンテナ管理による持続性の使用」を参照)

○ メッセージ駆動型 Bean (132 ページの「メッセージ駆動型 Beans の開発」を参照)

3. EJB 仕様に準拠して、次の Enterprise JavaBean のコードを記述します。
   ○ ローカル、リモートのいずれか、または両方のホームインタフェース
   ○ ローカルインタフェース、リモートインタフェースのいずれか、または両方
   ○ 実装クラス (メッセージ駆動型 Bean 用のものだけ)
4. インタフェースおよびクラスをコンパイルします。
5. `META-INF` ディレクトリ、および Enterprise JavaBean の構築に必要なその他のディレクトリを作成します。
6. 配備記述子ファイル (`ejb-jar.xml` および `sun-ejb-jar.xml`) を作成します。169 ページの「Enterprise JavaBean のアセンブルと配備」を参照してください。

   Bean がコンテナ管理による持続性を使用するエンティティ Bean である場合、`sun-cmp-mappings.xml` ファイルと `.dbschema` ファイルも作成する必要があります。86 ページの「コンテナ管理による持続性の使用」を参照してください。
7. 必要に応じて、クラスと XML ファイルを JAR ファイルにパッケージ化します。ディレクトリ配備を利用する場合は、この操作は省略可能です。
8. Bean を単独で配備するか、または J2EE アプリケーション内に含めます。『Sun ONE Application Server 開発者ガイド』を参照してください。

『Sun ONE Application Server 開発者ガイド』で説明されているベリファイアツールを使って、これらのファイルの構造を確認することをお勧めします。

Bean 使用の手引き

アプリケーションのどの部分をエンティティ Beans にするか、およびどの部分をセッション Beans (ステートフルまたはステートレス) またはメッセージ駆動型 Beans にするかを決定することは、アプリケーションの効率に重大な影響を及ぼします。一般には、次のガイドラインに従います。

- ユーザーの対話状態 (特定ユーザーに固有の状態) に対応する共有されないデータの格納にはステートフル Bean を使用する
- データへのアクセスまたはトランザクション操作の実行には、ステートレスセッション Beans を使用する

- タスク限定の小さな一般的なセッション Beans を作成する。これらの Enterprise JavaBeans は、多数のアプリケーションで使われる動作をカプセル化すると理想的である
- Enterprise JavaBeans をプレゼンテーションロジック (サーブレットおよび JSP) と同じサーバー上に配備するようにアプリケーションのアセンブル担当者に依頼する。これにより、アプリケーション実行時のリモートプロシージャコール (RPC) の数が削減される
- アプリケーションでは、`ejbRemove` メソッドを使って不要になった Bean を明示的に削除して、非活性化プロセスを排除することでコンテナのオーバーヘッドを減らす必要がある
- 別々のアプリケーションの EJB では一意の名前を付ける必要はないが、1 つのアプリケーションサーバーインスタンスのコンテキスト内では一意の名前を付ける必要がある。つまり、1 つのアプリケーション内の複数の Enterprise JavaBeans に同じ名前を付けることはできない

EJB 開発のガイドラインの詳細は、41 ページの「セッション Beans の使用」、55 ページの「エンティティ Beans の使用」、および 129 ページの「メッセージ駆動型 Beans の使用」を参照してください。

クライアントビューの手引き

ローカルインタフェースとリモートインタフェースのどちらを選択するかは、設計上の決定項目の 1 つであり、Enterprise JavaBean の開発時に開発者が決定します。ローカルとリモートのどちらのプログラミングモデルを使用するかを決定するときには、次の事項を考慮に入れる必要があります。

- リモートプログラミングモデルでは、場所に依存しない、柔軟な配備を行うことができる。クライアントと Enterprise JavaBeans は疎結合される
- リモート呼び出しでは pass-by-value を使用して、呼び出し元と呼び出される側を分離する層を提供する。これにより、不注意によるデータの変更を防ぐことができる
- ローカルオブジェクトでは、pass-by-reference は省略可能であり、J2EE 仕様には定められていない
- リモート呼び出しは、時間がかかる可能性がありる
- リモート呼び出しでは、パラメータとして渡されるオブジェクトは直列化可能である必要がある
- リモートタイプのナロー変換では、Java 言語キャストではなく `javax.rmi.PortableRemote.Object.narrow` を使用する必要がある

- リモート呼び出しには、ローカル呼び出しにはないエラーがある。クライアントでは、それらのリモート例外のハンドラを明示的にプログラミングする必要がある
- リモートプログラミングは、そのオーバーヘッドのために、一般に、比較的粗いコンポーネントアクセスで使用される
- ローカル呼び出しでは、オプションで pass-by-reference を使用できる。クライアントと Bean は、pass-by-reference セマンティックに依存するようにプログラミングされることがある。このため、ローカル呼び出しではローカルクライアントと Enterprise JavaBean を同じ場所に置く必要がある
- ローカルプログラミングはコンポーネントへの軽量なアクセスを提供するので、より細かいコンポーネントアクセスに適している
- ローカルインタフェースを通して渡されるオブジェクトは共有される可能性があることに留意する

詳細は、『Enterprise JavaBeans Specification, v2.0』を参照してください。

リモートインタフェースとローカルインタフェースの手引き

すべてのオブジェクト指向の開発と同様に、まず、ビジネスロジックおよびデータ処理に必要な細分レベルを決める必要があります。細分レベルにより、アプリケーションをどの程度の数に分割するかが決まります。

- 低レベルの細分 (Beans の数と Bean メソッドの呼び出し数が少ない) - よりモノリシックなアプリケーションを開発する。共有および再利用の推進はほとんど不可能だが、より高速に実行できるアプリケーションを作成できる
- 高レベルの細分 (Beans の数と Bean メソッドの呼び出し数が多い) - アプリケーションは、細かく定義された多数の Enterprise JavaBeans に分割される。サイト内のさまざまなアプリケーションで EJB の共有および再利用を推進できるアプリケーションとなる
- アプリケーションを中程度から多数の Beans に分割すると、アプリケーションパフォーマンスが著しく低下し、オーバーヘッドが増加する。Enterprise JavaBeans は単なる Java オブジェクトではなく、リモート呼び出しインタフェースのセマンティック、セキュリティセマンティック、トランザクションセマンティック、およびプロパティを伴う、より高いレベルのエンティティである。この複雑さにより、オーバーヘッドが生じる。

Sun ONE Application Server 機能へのアクセス

開発可能なエンティティ Beans には、『Enterprise JavaBeans Specification, v2.0』に厳密に準拠しているエンティティ Beans、その仕様と Sun ONE Application Server の付加価値機能の両方を活用するエンティティ Beans があります。

Sun ONE Application Server には、Sun ONE Application Server コンテナだけを通じて提供される機能がいくつかあります。Sun ONE Application Server API によって、Sun ONE Application Server 環境の各機能をプログラムで利用できます。

注　　これらの API は Sun ONE Application Server 環境だけで Beans を使う場合にだけ使用します。

EJB のアセンブリと配備について

Sun ONE Application Server でのモジュールとアプリケーションのアセンブリプロセスは、従来のあらゆる J2EE 定義仕様に準拠します。しかし、J2EE 仕様を超えて Sun ONE Application Server の機能を拡張する Sun ONE Application Server 固有の配備記述子を含めることも可能です。

J2EE モジュールは、1 つまたは複数の J2EE コンポーネントの集合で、各コンポーネントは 2 つの配備記述子を持っています。1 つは J2EE 標準の配備記述子で、もう 1 つは Sun ONE Application Server 固有の配備記述子です。Enterprise JavaBeans では、次の配備記述子ファイルがこれに該当します。

- `ejb-jar.xml` - J2EE 標準ファイル
- `sun-ejb-jar.xml` - Sun ONE Application Server 固有のファイル
- `sun-cmp-mappings.xml` - Sun ONE Application Server に固有のコンテナ管理による持続マッピング用のファイル。

EJB DTD および XML ファイルについては、169 ページの「Enterprise JavaBean のアセンブルと配備」を参照してください。

EJB に関連する要素のリストは、229 ページの「要素一覧」にアルファベット順に記載されています。

アセンブリと配備に関する一般的な情報は、『Sun ONE Application Server 開発者ガイド』を参照してください。

配備の手順は、『Sun ONE Application Server 管理者ガイド』および Administration interface のオンラインヘルプを参照してください。

第 2 章

セッション Beans の使用

この章では、Sun ONE Application Server 7 の環境でセッション Beans を作成するためのガイドラインを示します。

注 セッション Beans または EJB テクノロジに精通していない場合は、Java Software チュートリアルを参照してください。

`http://java.sun.com/j2ee/docs.html`

セッション Beans に関する詳細情報は、『Enterprise JavaBeans Specification, v2.0』の第 6 章～ 8 章にあります。

Sun ONE Application Server の概要は、19 ページの「Sun ONE Application Server Enterprise JavaBeans の紹介」および『Sun ONE Application Server Product Introduction』にあります。

この節には次の項目があります。

- セッション Beans について
- セッション Beans の開発
- 制限事項と最適化

セッション Beans に関する詳細情報は、『Enterprise JavaBeans Specification, v2.0』の第 6 章～ 8 章にあります。

セッション Beans について

この節では、ビジネスプロセスに対して効果的なモデルを開発するために、セッション Beans について注意が必要な事項の概要を示します。

この節には次の項目があります。

- セッション Bean の特性
- コンテナ

セッション Bean の特性

セッション Bean の特性を定義することは、アプリケーション内の持続的でない独立したステータスと関係があります。セッション Bean は、Sun ONE Application Server で動作するクライアントアプリケーションの一時的な論理拡張と考えることもできます。通常、セッション Bean はデータベースの共有データを表しませんが、データのスナップショットを取得します。ただし、セッション Bean はデータを更新できます。

セッション Beans には次の特性があります。

- 1 つのクライアントを対象に実行する
- トランザクションを認識できる
- 基礎となるデータベース内の共有データを直接表さないが、このデータのアクセスおよび更新は可能である
- 短命である
- データベース内で持続されない
- コンテナがクラッシュすると削除され、クライアントは新しいセッションを確立する必要がある

標準的な分散アプリケーションの多くは、繰り返しタスク、時限タスク、およびユーザ依存のタスクを実行するコードの論理的な単位から構成されます。これらのタスクには、単純なものと複雑なものがあり、さまざまなアプリケーションで必要です。たとえば、バンキングアプリケーションでは、トランザクションを実行する前に必ず顧客のアカウント ID と残高を照合する必要があります。このような個々のタスクは、本来は一時的なものであるためセッション Beans に適していると考えられます。

サンプルシナリオ

Web ベースのオンラインショッピングアプリケーションでよく利用されるショッピングカートは、典型的なセッション Bean です。これは、ユーザがアイテムを選択した場合にのみ、オンラインショッピングアプリケーションによって作成されます。アイテムの選択が完了すると、カート内のアイテムの価格が計算され、注文が出され、ショッピングカートオブジェクトが解放されます。ユーザはオンラインカタログで商品のブラウズを続けることができ、ユーザが別の商品を注文したい場合には、新しいショッピングカートが作成されます。

セッション Bean には、ほかのアプリケーションオブジェクトとの依存関係やコネクションを持たないものもあります。たとえば、ショッピングカートの Bean には、アイテム情報を格納するデータリストメンバー、現在カート内にあるアイテムの総額を格納するデータメンバー、およびアイテムの追加、削除、レポート、総計を行うメソッドがあります。一方、ショッピングカートには、データベースへのライブコネクションは一切ありません。

コンテナ

エンティティ Bean と同様に、セッション Bean は JDBC 呼び出しを介してデータベースにアクセスすることができます。また、セッション Bean はトランザクション設定も提供できます。これらのトランザクション設定や JDBC 呼び出しは、セッション Bean のコンテナによって参照されます。これにより、セッション Bean はコンテナによって管理されるトランザクションに関与できます。

コンテナ管理によるステートレスセッション Beans は、コンテナ管理によるステートフルセッション Beans とは異なるチャーターを持ちます。

ステートレスコンテナ

ステートレスコンテナは、ステートレスセッション Beans を管理します。ステートレスセッション Beans はクライアントに固有の状態を持ちません。したがって、特定の種類のすべてのセッション Beans は同じであると見なすことができます。

ステートレスセッション Bean コンテナは、Bean プールを使用して要求を処理します。Bean プールを定義するプロパティは、Sun ONE Application Server 固有の XML ファイルに格納されます。

- `steady-pool-size`
- `resize-quantity`
- `max-pool-size`
- `pool-idle-timeout-in-seconds`

これらのプロパティは、176 ページの「sun-ejb-jar.xml ファイルの要素」に示された配備記述子用に定義されます。

ステートフルコンテナ

ステートフルコンテナは、ステートフルセッション Beans を管理します。ステートフルセッション Beans はクライアントに固有の状態を持ちます。クライアントとステートフルセッション Beans の関係は、1 対 1 です。ステートフルセッション Bean を作成すると、それぞれに一意のセッション ID が割り当てられ、セッション Bean へのアクセス時に参照されます。ID は一意であるため、ステートフルセッション Bean のインスタンスに 1 つのクライアントだけがアクセスします。

ステートフルセッション Beans は、キャッシュを使って管理されます。ステートフルセッション Beans のキャッシュのサイズと動作は、次のパラメータを指定して制御できます。

- `max-cache-size`
- `resize-quantity`
- `cache-idle-timeout-in-seconds`
- `removal-timeout-in-seconds`
- `victim-selection-policy`

`max-cache-size` はキャッシュに保持できるセッション Beans の最大数を制御します。Beans の数が `max-cache-size` を超えてキャッシュがオーバーフローすると、コンテナは一部の Beans を非活性化するか、直列化した Bean の状態をファイルに書き出します。このファイルの書き出し先ディレクトリは、設定 API を使用して `server.xml` ファイルから取得されます。

これらのプロパティは、配備記述子用に定義されています。詳細については、176 ページの「sun-ejb-jar.xml ファイルの要素」を参照してください。

非活性化された Beans はファイルシステム上に保存されます。非活性化された Beans の保存先ディレクトリは、`server.xml` ファイル内の `server` 要素の `session-store` 属性を使って指定できます。非活性化されたステートフルセッション Beans は、デフォルトでは *instance_dir*`/session-store` の下に作成されるアプリケーション固有のサブディレクトリに保存されます。

セッション Beans の開発

クライアントによるセッション Bean の利用が終了すると、その Bean は解放されます。アプリケーションを設計するとき、一時的な 1 つのクライアントオブジェクトを潜在的なセッション Beans として指定する必要があります。

次の各項では、効果的なセッション Beans を開発する方法を説明します。

- 開発の要件
- セッション Beans の使用法の決定
- インタフェースの提供
- Bean クラス定義の作成

開発の要件

セッション Beans を開発するには、次の準備をする必要があります。

- セッション Bean がリモートクライアントビューを提供する場合は、セッション Bean のリモートインタフェースとリモートホームインタフェース
- セッション Bean がローカルクライアントビューを提供する場合は、セッション Bean のローカルインタフェースとローカルホームインタフェース
- Bean クラスの実装
- アセンブルデータと配備のデータ

セッション Bean の実装クラスの要件は次のとおりです。

- `javax.ejb.SessionBean` インタフェースを実装する
- クラスを public として定義する。abstract または final として定義することはできない
- 引数をとらない `ejbCreate` メソッドを 1 つ実装する
- ビジネスメソッドを実装する
- パラメータのないパブリックコンストラクタを含める
- `finalize` メソッドを定義してはならない

セッション Beans の使用法の決定

ここでは、ステートフルセッション Beans を実装するか、ステートレスセッション Beans を実装するかを決定する際のガイドラインを示します。

- ステートフルセッション Beans に関する検討事項
- ステートレスセッション Beans に関する検討事項

ステートフルセッション Beans に関する検討事項

次の条件のいずれかに該当する場合は、ステートフルセッション Beans が適しています。

- Bean の状態が、その Bean と特定のクライアントとの間の対話を表す
- Bean が、メソッドの起動に関するクライアントユーザの会話型状態について情報を保持する必要がある、または会話型状態そのものを表す
- クライアントとアプリケーションのその他のコンポーネントとの間に介在して、単純化したビューをクライアントに表示する Bean
- Bean が、いくつかの Enterprise JavaBeans のワークフローをバックグラウンドで管理する

ステートフルセッション Beans は 1 つのクライアント専用であるため、アプリケーションにアクセスするユーザ数が増えるにつれて、サーバーリソースの要求が増えます。この Beans は、クライアントが明示的に削除するか、タイムアウト時にコンテナによって削除されるまでは、コンテナ内に保持されます。

キャッシュファイルがいっぱいになった場合、およびキャッシュ内の Beans がタイムアウトになった場合は、コンテナはステートフルセッション Beans を非活性化して二次ストレージに送る必要があります。その後、クライアントがその Bean にアクセスすると、コンテナはその Bean を活性化し直します。この活性化と非活性化のプロセスは、サーバーのパフォーマンスオーバーヘッドに影響を与えます。

ステートレスセッション Beans に関する検討事項

次の条件のいずれかに該当する場合は、ステートレスセッション Bean が適しています。

- Bean の状態が特定クライアントのデータを持たない。つまり、メソッドの起動に関するユーザの会話型状態を Bean が保持する必要がない
- Bean が、1 つのメソッド起動で、すべてのクライアントに関する汎用タスクを実行する
- Bean が、クライアントがよく使用する読み取り専用データセットをデータベースからフェッチする。たとえば、今月販売している製品を示す表の行を取得する Bean など

データへのアクセスまたはトランザクション操作の実行には、ステートレスセッション Bean を使用します。ステートレスセッション Beans は、少数で多数のクライアントに対応できるので、高いスケーラビリティを実現します。これらの Beans は、ステートレス Bean プールに含まれるコンテナによって管理されます。これは、ステートレス Beans はクライアントに関係付けられないためです。ステートレスセッション Bean が提供するサービスに対する要求を受け取ったコンテナは、プール内のどの Bean インスタンスにもその要求を自由にディスパッチできます。

- リモートホームインタフェースの `create` メソッドは、セッション Bean のリモートインタフェースに返す必要がある
- ローカルインタフェースの `create` メソッドは、セッション Bean のローカルインタフェースに返す必要がある
- ホームインタフェースにほかの `create` メソッドを入れることはできない
- ステートレスセッション Bean には、`javax.ejb.SessionSynchronization` インタフェースを実装することはできない

インタフェースの提供

開発者は、Bean のインタフェースを提供する必要があります。Bean にリモートビューを実装するときは、リモートコンポーネントインタフェースとリモートホームインタフェースを提供します。ローカルビューを実装するときは、ローカルコンポーネントインタフェースとローカルホームインタフェースを提供します。

インタフェースを安全に使用するためには、考えられる配備シナリオを慎重に検討してから、どのインタフェースをローカルにしてどれをリモートにするかを決定し、それらの選択を念頭においてアプリケーションコードを開発する必要があります。

次の各項で、インタフェースの作成について説明します。

- リモートインタフェースの作成
- ローカルインタフェースの作成
- ローカルホームインタフェースの作成
- リモートホームインタフェースの作成

リモートインタフェースの作成

セッション Bean のリモートインタフェースは、Bean のメソッドへのユーザーのアクセスを定義します。すべてのリモートインタフェースは、`javax.ejb.EJBObject` を拡張します。次に例を示します。

```
import javax.ejb.*;
import java.rmi.*;
public interface MySession extends EJBObject {
// ここでビジネスメソッドを定義します。
public String getACcountname() throws RemoteException;
}
```

リモートインタフェースでは、クライアントが呼び出すセッション Bean のビジネスメソッドを定義します。リモートインタフェースで定義した各メソッドには、Bean クラス内の対応するメソッドを指定する必要があります。Bean クラス内の対応するメソッドには、同じ署名、同じパラメータタイプ、同じ戻り値タイプを持たせる必要があります。メソッド名の最初には「ejb」と記述します。たとえば、`MySession` の実装クラスには次のメソッドが含まれます。

```
String ejbgetAccountname() throws RemoteException
{
method implementation
}
```

ローカルインタフェースの作成

開発中に Bean のローカルインタフェースを定義すると、呼び出し側の Bean が同じコンテナに含まれる場合、つまり同じアドレス空間または同じ Java 仮想マシン (JVM) で実行される場合に、Bean を効率的に呼び出せます。これにより、同じ場所に配置するように設計したアプリケーションのパフォーマンスが向上します。

ただし、ローカルインタフェースの呼び出しのセマンティックは、リモートインタフェースとは異なります。たとえば、リモートインタフェースは pass-by-value セマンティックを使ってパラメータを渡しますが、ローカルインタフェースでは pass-by-reference を使います。開発者は、ローカルインタフェースを通して渡されるオブジェクトは共有される可能性があることに留意する必要があります。特に、ある Enterprise JavaBean の状態が別の Enterprise JavaBean の状態に割り当てられないように注意します。また、トランザクションまたはセキュリティの内容に変化がある場合は特に、ローカルインタフェースを通して渡すオブジェクトを決定する際にも注意が必要です。

ローカルインタフェースで `javax.ejb.EJBLocalObject` インタフェースを拡張して、特別なインタフェースを提供することができます。ローカルインタフェースで定義されたメソッドの `throws` 句に、`java.rmi.RemoteException` を含めることはできません。次に例を示します。

```
import javax.ejb.*;
    public interface MyLocalSession extends EJBLocalObject {
// ここでビジネスメソッドを定義します。
}
```

ローカルインタフェースに定義された各メソッドについて、セッション Bean 内に一致するメソッドがある必要があります。一致するメソッドは、同じ名前、同じ数とタイプの引数、および同じ戻り値タイプを持っている必要があります。セッション Bean クラスの一致するメソッドの throws 句で定義されたすべての例外は、ローカルインタフェースのメソッドの throws 句で定義されている必要があります。これらのメソッドは java.rmi.RemoteException をスローできません。

ローカルホームインタフェースの作成

ホームインタフェースは、クライアントがアプリケーションを使ってセッション Beans の作成および削除を行うためのメソッドを定義します。Enterprise JavaBean のローカルホームインタフェースでは、ローカルクライアントが EJB オブジェクトの作成、検索、および削除を行うことができるようにするメソッドと、Bean インスタンス固有ではないホームビジネスメソッド (セッション Beans が検索メソッドとホームビジネスメソッドを持たない) を定義します。ローカルホームインタフェースは、開発者によって定義され、コンテナによって実装されます。クライアントは、JNDI を使用してセッション Bean のホームインタフェースを検索します。

ローカルホームインタフェースでは、ローカルクライアントは次の操作を実行できます。

- 新しいセッションオブジェクトの作成
- セッションオブジェクトの削除

ローカルホームインタフェースは通常、javax.ejb.EJBLocalHome を拡張します。次に例を示します。

```
import javax.ejb.*;
import java.rmi.*;

public interface MySessionLocalBeanHome extends EJBLocalHome {
    MySessionLocalBean create() throws CreateException;
}
```

create メソッド

この例のように、セッション Bean のホームインタフェースでは、1 つまたは複数の create メソッドを定義します。各メソッドには create という名前を付け、セッション Bean クラスで定義された ejbCreate メソッドの数および引数のタイプと一致させる必要があります。ただし、各 create メソッドの戻り値タイプは、対応する ejbCreate メソッドの戻り値タイプとは一致しません。代わりに、セッション Bean のローカルインタフェースタイプを返します。

ejbCreate メソッドの throws 句で定義するすべての例外は、リモートインタフェース内の一致する create メソッドの throws 句で定義されている必要があります。さらに、ホームインタフェースの throws 句には常に、javax.ejb.CreateException を含める必要があります。

remove メソッド

リモートクライアントは、javax.ejb.EJBObject インタフェースの remove メソッド、または javax.ejb.EJBHome インタフェースの remove(Handle handle) メソッドを使ってセッションオブジェクトを削除できます。

セッションオブジェクトはクライアントへのアクセスに必要な主キーを持たないため、セッションで javax.ejb.EBJHome.remove(Object primaryKey) メソッドを呼び出すと、javax.ejbRemoveException が呼び出されます。

リモートホームインタフェースの作成

コンテナは、コンテナに配備されるリモートホームインタフェースを定義する各セッション Bean のリモートホームインタフェースを実装しています。これを実装するオブジェクトは、セッション EJBHome オブジェクトと呼ばれます。リモートホームインタフェースでは、クライアントは次の操作を実行できます。

- 新しいセッションオブジェクトの作成
- セッションオブジェクトの削除
- セッション Beans の javax.ejb.EJBMetaData インタフェースの取得
- リモートホームインタフェースのハンドルの取得

リモートホームインタフェースは javax.ejb.EJBHome インタフェースを拡張する必要があり、特別なインタフェースを持つことができます。このインタフェースに定義されたメソッドは、RMI/IIOP のルールに従う必要があります。

リモートホームインタフェースでは、create<METHOD>(...) メソッドを 1 つ以上定義する必要があります。

リモートホームインタフェースは通常、javax.ejb.EJBHome を拡張します。次に例を示します。

```
import javax.ejb.*;
import java.rmi.*;

public interface MySessionHome extends EJBHome {
    MySession create() throws CreateException, RemoteException;
}
```

この例のように、セッション Bean のホームインタフェースでは、1 つまたは複数の create メソッドを定義します。ただし、各 create メソッドの戻り値タイプは、対応する ejbCreate メソッドの戻り値タイプとは一致しません。代わりに、セッション Beans のリモートインタフェースタイプを返します。

ejbCreate メソッドの throws 句で定義するすべての例外は、リモートインタフェース内の一致する create メソッドの throws 句で定義されている必要があります。さらに、ホームインタフェースの throws 句には常に、javax.ejb.CreateException および java.rmi.RemoteException を含める必要があります。

注 ステートレスセッション Beans では、ホームインタフェースは create メソッドを 1 つだけ持ち、その Bean は ejbCreate メソッドを 1 つだけ持つ必要があります。どちらのメソッドも引数をとりません。

Bean クラス定義の作成

セッション Bean のクラスは、final でも abstract でもなく、public として定義する必要があります。Bean クラスは、javax.ejb.SessionBean インタフェースを実装する必要があります。

```
import java.rmi.*;
import java.util.*;
import javax.ejb.*;
public class MySessionBean implements SessionBean {
   // セッション Beans の実装。これらのメソッドは常に取り込む
必要があります。
   public void ejbActivate() {
   }
   public void ejbPassivate() {
   }
   public void ejbRemove() {
   }
   public void setSessionContext(SessionContext ctx) {
   }

   // 他のコードは省略します。
   }
```

セッション Bean は、1 つ以上の ejbCreate(...) メソッドを実装する必要があります。クライアントが Bean を呼び出すときは、必ずこのメソッドを 1 つ使います。次に例を示します。

```
public void ejbCreate() {
    string[] userinfo = {"User Name", "Encrypted Password"} ;
}
```

各 ejbCreate(...) メソッドは public として宣言され、void を返し、ejbCreate という名前を付ける必要があります。引数は、正しいタイプの Java RMI である必要があります。throws 句では、アプリケーション固有の例外および java.ejb.CreateException を定義できます。

セッション Beans は、1 つ以上のビジネスメソッドも実装します。これらのメソッドは通常、各 Bean に対して固有であり、その特定の機能を表します。たとえば、セッション Bean がユーザログインを管理する場合に、このセッション Bean に validateLogin という名前の固有の関数を含めることができます。

任意のビジネスメソッド名を付けることができますが、この EJB インタフェースに定義するメソッドの名前と重複しないように注意してください。ビジネスメソッドは、public として宣言する必要があります。メソッドの引数および戻り値は、正しいタイプの Java RMI である必要があります。throws 句では、アプリケーション固有の例外を定義できます。

セッション同期

ステートフルセッション Bean のクラス定義で許可される 1 つのインタフェース実装があります。特に、javax.ejb.SessionSynchronization によって、セッション Bean インスタンスはトランザクション境界を認識し、状態とトランザクションの同期をとることができます。

javax.ejb.SessionSynchronization インタフェースでは、コンテナがステートフルセッション Bean のインスタンスにトランザクション境界を知らせることができます。セッション Bean クラスは、必ずしもこのインタフェースを実装する必要はありません。状態とトランザクションの同期をとる必要がある場合のみ、セッション Bean クラスはこのインタフェースを実装する必要があります。たとえば、このインタフェースを実装した、ステートフルセッション Bean は、トランザクションのコミット前やコミット後ではなく、新しいトランザクションの開始後にコールバックを取得します。

このインタフェースの詳細については、『Enterprise JavaBeans Specification, v2.0』を参照してください。

注　　コンテナは、そのコンテナが管理するトランザクションを使用する、ステートフル Beans のセッション同期インタフェースメソッドだけを起動します。

抽象的なメソッド

リモートインタフェースで定義するビジネスメソッドのほかに、`EJBObject` インタフェースはいくつかの抽象的なメソッドを定義します。これらのメソッドを使うと、次の処理が可能になります。

- Bean のホームインタフェースの取得
- Bean のハンドル (固有の識別子) の取得
- ほかの Bean と比較して、一意かどうかの確認
- 不要になった Bean の解放または削除

これらの組み込みメソッドとその使用法の詳細については、『Enterprise JavaBeans Specification, v2.0』を参照してください。

コンテナが提供する配備ツールは、セッション Bean が配備されたときに追加のクラスを生成します。

制限事項と最適化

セッション Beans の開発に関する制限事項と、最適化に関するガイドラインを示します。

- セッション Bean のパフォーマンスの最適化
- トランザクションの制限

セッション Bean のパフォーマンスの最適化

ステートフルセッション Beans では、クライアントと Bean を同じ場所に配置して、同じプロセスアドレス空間で実行させることで、パフォーマンスを向上できます。

トランザクションの制限

トランザクションについては、コンテナによる次の制限事項があり、セッション Beans を開発するときはこれらを遵守する必要があります。

- セッション Bean が関与できるトランザクションは一度に 1 つだけである
- セッション Bean がトランザクションに関与している場合、配備記述子のトランザクション属性によって、コンテナが別のまたは指定されていないトランザクションコンテキストでメソッドを実行するようなセッション Bean のメソッドをクライアントは呼び出すことができない (呼び出した場合は例外がスローされる)
- セッション Bean のインスタンスがトランザクションに関与している場合、クライアントは、セッションオブジェクトのホームまたはコンポーネントインタフェースオブジェクトの `remove` メソッドを呼び出すことができない (呼び出した場合は例外がスローされる)

第 3 章

エンティティ Beans の使用

この章では、エンティティ Beans について、および Sun ONE Application Server 7 環境でエンティティ Beans を作成する際の要件について説明します。

注 エンティティ Beans または EJB テクノロジに精通していない場合は、Java Software チュートリアルを参照してください。

http://java.sun.com/j2ee/docs.html

エンティティ Beans に関する詳細情報は、『Enterprise JavaBeans Specification, v2.0』の第 9 章、第 10 章、第 12 章、第 13 章、および第 14 章にあります。

Sun ONE Application Server の概要は、19 ページの「Sun ONE Application Server Enterprise JavaBeans の紹介」および『Sun ONE Application Server Product Introduction』にあります。

この節には次の項目があります。

- エンティティ Beans について
- エンティティ Beans の開発
- 読み取り専用 Beans の使用
- 同時アクセスの同期化処理

注 エンティティ Beans にはすでに精通していて、コンテナ管理による持続性のみに関心がある場合は、79 ページの「エンティティ Beans のコンテナ管理による持続性の使用」に進んでください。

エンティティ Beans について

エンティティ Bean は、基礎となるデータベースに格納されているエンティティのオブジェクトビュー、または既存のエンタープライズアプリケーション (メインフレームプログラムや ERP アプリケーションなど) によって実装されるエンティティを実装します。たとえば、ビジネスオブジェクトには、顧客、注文、製品などがあります。エンティティ Bean のインスタンスと基本となるデータベースの間でエンティティの状態を転送するためのデータアクセスプロトコルを、オブジェクト持続性と呼びます。

この節では次の項目について説明します。

- エンティティ Beans の特性
- コンテナ
- 持続性
- 読み取り専用 Beans

エンティティ Beans の特性

エンティティ Beans は、いくつかの点でセッション Beans と異なります。エンティティ Beans には持続性があり、複数のクライアントからの同時アクセスに対応しています。また、主キーを持ち、その他のエンティティ Beans との関係を持つこともできます。

エンティティ Beans には次の特性があります。

- データベース内のデータのオブジェクトを表示する
- 複数のユーザーによる共有アクセスが可能である
- Bean 管理持続性またはコンテナ管理持続性を使用して、すべてのクライアントにとって不要になるまで持続する
- サーバークラッシュを透過的に回避する
- データベースの共有データを表示する

データベース、ドキュメントなどのビジネスオブジェクトを対象とする、カプセル化された持続トランザクション対話は、エンティティ Bean に適しています。

コンテナ

エンティティ Beans は、コンテナが管理するエンティティオブジェクトのセキュリティ、同時性、トランザクション、および他のコンテナ固有のサービスを管理するようにエンティティ Bean コンテナに要求します。複数のクライアントによるエンティティオブジェクトへの同時アクセスが可能であり、コンテナは、トランザクションを介した同時アクセスを透過的に処理します。

各エンティティ Bean は、一意のオブジェクト ID を持ちます。たとえば、顧客のエンティティ Bean を顧客番号によって識別することができます。この一意の ID、すなわち主キーにより、クライアントは特定のエンティティ Bean を見つけることができます。

セッション Bean と同様に、エンティティ Bean は配備記述子を使ってトランザクション属性を設定できるメソッドに含まれる JDBC 呼び出しを介してデータベースにアクセスすることができます。次で説明するように、このコンテナは、Bean 管理による持続性とコンテナ管理による持続性の両方をサポートします。

持続性

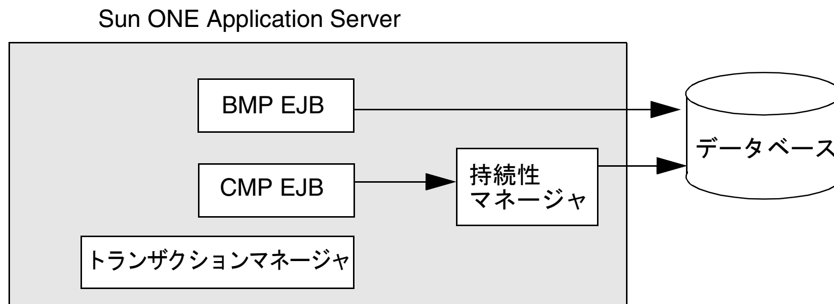
エンティティ Bean の状態は永続的なストレージに保存されるので、持続的です。持続性とは、そのエンティティ Bean の状態が、アプリケーションまたはサーバープロセスの存続期間を超えて存在することを意味します。

エンティティ Beans の持続性は、Bean によって明示的に実現され、Bean 開発者によってプログラムされます。これを BMP (Bean 管理による持続性) と呼びます。

Sun ONE Application Server と Enterprise JavaBeans の持続性管理 API を活用し、持続性の管理をコンテナに任せることもできます。これを CMP （コンテナ管理による持続性）と呼びます。CMP メカニズムによって信頼性の高い持続性を確保するには、Sun ONE Application Server に統合されている持続性マネージャが必要です。コンテナ管理による持続性の詳細については、79 ページの「エンティティ Beans のコンテナ管理による持続性の使用」を参照してください。

次の図は、Sun ONE Application Server 環境で持続性がどのように機能するかを示しています。

エンティティ Bean のフロー



自分のアプリケーションに最も適した持続性メソッドを選択するガイドラインについては、60 ページの「エンティティ Bean の使用法の決定」を参照してください。

この節では次のトピックについて説明します。

- Bean 管理による持続性
- コンテナ管理による持続性

Bean 管理による持続性

Bean 管理による持続性では、Bean の持続性を Bean 自身が管理します。エンティティ Bean のコードには、データベースにアクセスする呼び出しを記述します。

開発者は、JDBC と SQL を介して、データベースアクセス呼び出しを Bean クラスのメソッドに直接提供することによって、Bean 管理エンティティ Bean をコード化します。データベースアクセス呼び出しは、`ejbCreate`、`ejbRemove`、`ejbFind`*XXX*、`ejbLoad`、および `ejbStore` の各メソッドに存在している必要があります。この方法には、これらの Beans をアプリケーションサーバーに簡単に配備できるという利点があります。欠点は、データベースへのアクセスが高コストになることです。データベースへのアクセスの最適化は、場合によってはアプリケーションプログラマが行うよりも、アプリケーションサーバーが行うほうが適しています。また、Bean 管理による持続性では、JDBC コードを記述する必要があります。

JDBC を使ったデータ処理の詳細は、Sun ONE Application Server の『Developer's Guide to J2EE Features and Services』を参照してください。

コンテナ管理による持続性

コンテナ管理による持続性では、Enterprise JavaBean コンテナが、持続性マネージャを介して、エンティティ Bean が要求するすべてのデータベースアクセスを処理します。この Bean のコードには、データベースアクセス (JDBC) 呼び出しは含まれません。したがって、Bean のコードが特定の永続的なストレージメカニズム (データベース) に結び付けられることはありません。この柔軟性により、同じエンティティ Bean を別のデータベースに再配備する場合でも、Bean のコードを修正する必要がありません。つまり、エンティティ Beans は移植性に優れています。

Bean の開発者は、抽象的な Bean クラスを用意する必要があります。一般に、コンテナ管理による持続性では、実行時に、Bean の状態の読み込みと保存の方法 (`ejbLoad` メソッドと `ejbStore` メソッド) を知る具体的な実装クラスが生成されます。

データアクセス呼び出しを生成する場合、コンテナは、エンティティ Bean の抽象スキーマに指定された情報を必要とします。抽象スキーマの詳細については、84 ページの「抽象スキーマ」を参照してください。

読み取り専用 Beans

読み取り専用 Bean は、EJB クライアントが変更できないエンティティ Bean です。ただし、読み取り専用 Bean が表すデータは、ほかの Enterprise JavaBeans や、データベースの直接更新などの方法によって、外部から更新される場合があります。

注 Sun ONE Application Server の今回のリリースでは、Bean 管理による持続性を使用するエンティティ Beans のみを読み取り専用として指定できます。

読み取り専用 Bean は、基になるデータがまったく変更されないか、まれにしか変更されない場合に最も適しています。読み取り専用 Bean の作成手順は、75 ページの「読み取り専用 Beans の使用」を参照してください。

エンティティ Beans の開発

エンティティ Bean を作成するときは、いくつかのクラスファイルを準備する必要があります。次の各項目では、必要な作業について説明します。

- エンティティ Bean の使用法の決定
- Bean 開発者の役割
- 主キークラスの定義
- リモートインタフェースの定義
- ローカルインタフェースの定義
- Bean クラス定義の作成 (Bean 管理による持続性)

エンティティ Bean の使用法の決定

Bean がプロシージャではなくビジネスエンティティを表す場合、または Bean の状態を持続的にする (サーバーがシャットダウンされても Bean の状態がデータベースに存在する) 必要がある場合、あるいはその両方に該当する場合は、エンティティ Bean を使用する必要があります。

セッション Beans とは異なり、エンティティ Bean のインスタンスには、複数のクライアントが同時にアクセスできます。インスタンスの状態の同期は、コンテナがトランザクションを使って行います。この役割がコンテナに委ねられているので、開発者は、複数のトランザクションからの同時アクセスメソッドについて考える必要がありません。

選択する持続性メソッドによっても、次のような影響があります。

- Bean 管理による持続性 - エンティティ Bean を実装してその Bean 自体の持続性を管理する場合は、EJB クラスメソッドに持続性コード (JDBC 呼び出しなど) を直接実装する。その場合、ダウンサイドでは移植性が失われるため、Bean が特定のデータベースに関連付けられるというリスクがある
- コンテナ管理による持続性 - エンティティ Bean の持続性をコンテナで管理する場合は、コンテナが透過的に持続性状態を管理するその場合、Bean メソッドのデータアクセスコードを実装する必要はない。この方法は実装が簡単なだけでなく、Bean を別のデータベースに移植できる利点がある。実装のガイドラインについては、79 ページの「エンティティ Beans のコンテナ管理による持続性の使用」を参照

Bean 開発者の役割

ここでは、持続性を Bean で管理するエンティティ Bean を Sun ONE Application Server に配備する上で必要な処理について説明します。

エンティティ Bean の開発者は、次のクラスファイルを用意する必要があります。

- 主キークラス
- エンティティ Bean がリモートクライアントビューを提供する場合は、エンティティ Bean のリモートインタフェースとリモートホームインタフェース
- エンティティ Bean がローカルクライアントビューを提供する場合は、エンティティ Bean のローカルインタフェースとローカルホームインタフェース
- エンティティ Bean クラス

主キークラスの定義

EJB アーキテクチャでは、RMI-IIOP の タイプが正しいクラスであれば、任意のクラスを主キークラスにできます。このクラスは、`hashCode` メソッドと equals (Object other) メソッドの実装を提供する必要があります。主キークラスを特定のエンティティ Bean クラス専用にして、エンティティ Bean クラスごとに異なるクラスを主キーとして定義できますが、複数の エンティティ Beans が同じ主キークラスを使うこともできます。

開発者は、主キークラスを配備記述子に指定する必要があります。

リモートインタフェースの定義

この節では、次の項目について説明します。

- リモートホームインタフェースの作成
- リモートインタフェースの作成

リモートホームインタフェースの作成

Bean 開発者は、必要に応じて Bean のリモートホームインタフェースを作成する必要があります。ホームインタフェースは、クライアントがアプリケーションにアクセスしてエンティティオブジェクトを作成、検索、および削除するためのメソッドを定義します。作成するリモートホームインタフェースは、次の要件を満たす必要があります。

- インタフェースは `javax.ejb.EJBHome` インタフェースを拡張する必要がある

- このインタフェースに定義されるメソッドは、RMI/IIOP のルールに従う必要がある。つまり、引数と戻り値が正しいタイプの RMI-IIOP で、throws 句が `java.rmi.RemoteException` を含んでいる必要がある
- リモートホームインタフェースに定義されるメソッドが、次のいずれかに該当する必要がある
  - create メソッド
  - リモートホームインタフェースには、常に 1 つのオブジェクトを検索する `findByPrimaryKey` メソッドを含める必要がある。このメソッドは、メソッド引数として主キークラスを宣言する必要がある
  - 検索メソッド
  - home メソッド。create、find、remove メソッドの名前と重複しない限り、home メソッドには任意の名前をつけられる。エンティティ Bean クラスに指定される、一致する `ejbHome` メソッドは、Bean のリモートホームインタフェースに指定される home メソッドと同じ数とタイプの引数を持ち、同じタイプの戻り値を返す必要がある

リモート Create メソッド

- create メソッドには、createXXX という名前をつける必要がある。XXX は、Enterprise JavaBean クラスに定義されるいずれかの `ejbCreateXXX` メソッドと一致する一意のメソッド名の一部である。たとえば、`createEmployee(...)`、`createLargeOrder(....)` のような名前をつける
- Bean 内の一致する `ejbCreateXXX` は、同じ数とタイプの引数を持つ必要がある。ただし、戻り値のタイプは異なる
- createXXX メソッドの戻り値タイプは、エンティティ Bean のリモートインタフェースのタイプと同じである必要がある
- Enterprise JavaBean クラスの一致する `ejbCreateXXX` および `ejbPostCreateXXX` メソッドの throws 句で定義されるすべての例外は、リモートホームインタフェースの一致する create メソッドの throws 句にも含まれる必要がある。つまり、create メソッドに定義される一連の例外は、`ejbCreateXXX` および `ejbPostCreateXXX` メソッド用に定義されるすべての例外のスーパーセット (上位集合) である必要がある
- create メソッドの throws 句には、`javax.ejb.CreateException` を含める必要がある

リモート find メソッド

- ホームインタフェースは、1 つまたは複数の find メソッドを定義できる。各メソッドには findXXX という名前をつける必要がある。XXX は、一意のメソッド名の一部である。たとえば、`findApplesAndOranges` のような名前をつける

- 各検索メソッドは、エンティティ Bean のクラス定義で定義された検索メソッドの 1 つと一致する必要がある
- その数および引数のタイプも Bean クラスの検索メソッド定義と一致する必要がある
- 1 つのオブジェクトを検索する検索の場合、find `<METHOD>` メソッドの戻り値のタイプは、エンティティ Bean のリモートインタフェースのタイプと一致する必要がある。複数のオブジェクトを検索する検索では、そのコレクションと一致する必要がある
- エンティティ Bean クラスの `ejbFind` メソッドの throws 句で定義されるすべての例外は、リモートホームインタフェースの、一致する find メソッドの throws 句にも含まれている必要がある
- 検索メソッドの throws 句には、`javax.ejb.FinderException` を含める必要がある

findByPrimaryKey メソッド

- すべてのリモートホームインタフェースには、常に 1 つのオブジェクトを検索する `findByPrimaryKey` メソッドを含める必要がある
- このメソッドは、メソッド引数として主キークラスを宣言する必要がある
- エンティティ Bean クラスの `ejbFindByPrimaryKey` メソッドの throws 句で定義されるすべての例外は、リモートホームインタフェースの、一致する find メソッドの throws 句にも含まれている必要がある
- `findByPrimaryKey` メソッドの throws 句には、`javax.ejb.FinderException` を含める必要がある

リモート remove メソッド

すべてのホームインタフェースには、不要になった Enterprise JavaBean を破棄する 2 つの `remove` メソッドが `javax.ejb.EJBHome` の拡張によって自動的に定義されます。

```
public void remove(java.lang.Object primaryKey)
throws java.rmi.RemoteException,   RemoveException

public void remove(Handle handle)
throws java.rmi.RemoteException,   RemoveException
```

注 これらの remove メソッドはオーバーライドしないでください。

リモートホームインタフェースの例

```
import javax.ejb.*;
import java.rmi.*;


public interface MyEntityBeanLocalHome
   extends EJBHome
{
   /**
       * Create an Employee
       * @param empName Employee name
       * @exception CreateException If the employee cannot be
         created
         * @return The remote interface of the bean
       */
   public MyEntity create(String empName)
       throws CreateException;
   /**
       * Find an Employee
       * @param empName Employee name
       * @exception FinderException if the empName is not found
       * @return The remote interface of the bean
       */
   public MyEntity findByPrimaryKey(String empName)
       throws FinderException;
}
```

ローカルインタフェースの定義

ローカルアクセスが可能な Enterprise JavaBean を作成するには、ローカルインタフェースおよびローカルホームインタフェースをコーディングする必要があります。ローカルインタフェースは Bean のビジネスメソッドを定義し、ローカルホームインタフェースは Bean のライフサイクル (作成 / 削除) および検索メソッドを定義します。

この節には次のトピックがあります。

- ローカルホームインタフェースの作成
- ローカルインタフェースの作成

ローカルホームインタフェースの作成

ホームインタフェースは、クライアントがアプリケーションを使ってエンティティ Beans の作成および削除を行うためのメソッドを定義します。Bean のローカルホームインタフェースでは、ローカルクライアントが EJB オブジェクトの作成、検索、および削除を行うことができるようにするメソッドと、Bean インスタンス固有ではないホームビジネスメソッド (セッション Beans が検索メソッドとホームビジネスメソッドを持たない) を定義します。ローカルホームインタフェースは、開発者によって定義され、コンテナによって実装されます。クライアントは、JNDI を使用して Bean のホームインタフェースを検索します。

ローカルホームインタフェースでは、ローカルクライアントは次の操作を実行できます。

- ホーム内でのエンティティオブジェクトの新規作成
- ホーム内での既存のエンティティオブジェクトの検索
- ホームからのエンティティオブジェクトの削除
- ホームビジネスメソッドの実行

ローカルホームインタフェースは通常、`javax.ejb.EJBLocalHome` を拡張します。次に例を示します。

```
import javax.ejb.*;
public interface MyEntityLocalBeanHome extends EJBLocalHome {
    MyEntityLocalBean create() throws CreateException;
}
```

ローカルインタフェースの作成

エンティティ Bean がコンテナ管理関係のターゲットである場合、そのエンティティ Bean はローカルインタフェースを持つ必要があります。Bean がターゲットであるかどうかは、関係の方向によって決まります。それらのエンティティ Beans はローカルアクセスを必要とするので、コンテナ管理関係に関与するエンティティ Beans は同じ EJB JAR ファイル内に置く必要があります。同じ場所に置くことによる主な利点は、ローカル呼び出しはリモート呼び出しよりも高速であるため、パフォーマンスを向上できることです。

ローカルインタフェースでは、参照セマンティクスによる値の受け渡しが許可されるので、ローカルインタフェースを通して渡されるオブジェクトは共有される可能性があることに注意する必要があります。特に、ある Enterprise JavaBean の状態が別の Enterprise JavaBean の状態として割り当てられないように注意します。また、トランザクションまたはセキュリティの内容に変化がある場合は特に、ローカルインタフェースを通して渡すオブジェクトを決定する際にも注意が必要です。

- インタフェースは `javax.ejb.EJBLocalHome` インタフェースを拡張する必要がある

- ローカルインタフェースに含まれるメソッドの throws 句に、`java.rmi.RemoteException` を含めることはできない
- ローカルホームインタフェースに定義されるメソッドが、次のいずれかに該当する必要がある
  - create メソッド
  - 検索メソッド
  - home メソッド。

(ローカル) create メソッド

- create メソッドには、createXXX という名前をつける必要がある。XXX は、Enterprise JavaBean クラスに定義されるいずれかの `ejbCreateXXX` メソッドと一致する一意のメソッド名の一部である。たとえば、`createEmployee(...)`、`createLargeOrder(....)` のような名前をつける
- Bean 内の一致する `ejbCreateXXX` は、同じ数とタイプの引数を持つ必要がある。ただし、戻り値のタイプは異なる
- createXXX メソッドの戻り値タイプは、エンティティ Bean のローカルインタフェースのタイプと同じである必要がある
- Enterprise JavaBean クラスの一致する `ejbCreateXXX` および `ejbPostCreateXXX` メソッドの throws 句で定義されるすべての例外は、リモートホームインタフェースの一致する create メソッドの throws 句にも含まれる必要がある。つまり、create メソッドに定義される一連の例外は、`ejbCreateXXX` および `ejbPostCreateXXX` メソッド用で定義されるすべての例外のスーパーセット (上位集合) である必要がある
- create メソッドの throws 句には、`javax.ejb.CreateException` を含める必要がある

(ローカル) find メソッド

- ホームインタフェースは、1 つまたは複数の find メソッドを定義できる。各メソッドには findXXX という名前をつける必要がある。XXX は、一意のメソッド名の一部である。たとえば、`findApplesAndOranges` のような名前をつける
- 各検索メソッドは、エンティティ Bean のクラス定義で定義された検索メソッドの 1 つと一致する必要がある
- その数および引数のタイプも Bean クラスの検索メソッド定義と一致する必要がある
- 1 つのオブジェクトを検索する検索の場合、find `<METHOD>` メソッドの戻り値のタイプは、エンティティ Bean のローカルインタフェースのタイプと一致する必要がある。複数のオブジェクトを検索する検索では、そのコレクションと一致する必要がある

- エンティティ Bean クラスの `ejbFind` メソッドの throws 句で定義されるすべての例外は、リモートホームインタフェースの、一致する find メソッドの throws 句にも含まれている必要がある
- 検索メソッドの throws 句には、javax.ejb.FinderException を含める必要がある

findByPrimaryKey メソッド

- すべてのローカルホームインタフェースには、常に 1 つのオブジェクトを検索する `findByPrimaryKey` メソッドを含める必要がある
- このメソッドは、メソッド引数として主キークラスを宣言する必要がある
- エンティティ Bean クラスの `ejbFindByPrimaryKey` メソッドの throws 句で定義されるすべての例外は、リモートホームインタフェースの、一致する find メソッドの throws 句にも含まれている必要がある
- `findByPrimaryKey` メソッドの throws 句には、`javax.ejb.FinderException` を含める必要がある

(ローカル) home メソッド

- create、find、remove メソッドの名前と重複しない限り、home メソッドには任意の名前をつけられる
- エンティティ Bean クラスに指定される、一致する `ejbHome` メソッドは、Bean のローカルホームインタフェースに指定される home メソッドと同じ数とタイプの引数を持ち、同じタイプの戻り値を返す必要がある

リモートインタフェースの作成

リモートインタフェースで定義するビジネスメソッドのほかに、EJBObject インタフェースはいくつかの抽象的なメソッドを定義します。これらのメソッドを使うと、次の処理が可能になります。

- Bean のホームインタフェースの取得
- Bean のハンドルの取得 - Bean のインスタンスを一意に識別する Bean の主キーを取得するためのハンドル
- ほかの Bean と比較して、一意かどうかの確認
- 不要になった Bean の削除

これらの組み込みメソッドとその使用法の詳細については、『Enterprise JavaBeans Specification, v2.0』を参照してください。

注 『Enterprise JavaBeans Specification, v2.0』では、Bean クラスへのリモートインタフェースのメソッド実装が許可されていますが、一方で、同書で述べられているクライアント - コンテナ - EJB プロトコルに違反して、クライアントに this を介して直接参照を不用意に渡さないようにするために、この方法を行わないように勧めています。

- エンティティ Bean のリモートインタフェースは、Bean のメソッドへのユーザーのアクセスを定義する
- インタフェースは `javax.ejb.EJBObject` インタフェースを拡張する必要がある
- リモートインタフェースに定義されるメソッドは、RMI/IIOP のルールに従う必要がある
- つまり、引数と戻り値が正しいタイプの RMI-IIOP で、throws 句が java.rmi.RemoteException を含んでいる必要がある
- リモートインタフェースに定義された各メソッドについて、エンティティ Bean のクラス内に一致するメソッドがある必要がある。一致するメソッドは、同じ名前、同じ数とタイプの引数、および同じ戻り値タイプを持っている必要がある
- Enterprise JavaBean クラスの一致するメソッドの throws 句で定義されたすべての例外は、リモートインタフェースのメソッドの throws 句で定義されている必要がある
- リモートインタフェースのメソッドに、ローカルインタフェースのタイプ、ローカルホームインタフェースのタイプ、および持続性がコンテナによって管理されるエンティティ Beans が引数または結果として使用する管理コレクションクラスを表示させることはできない

リモートインタフェースの例

次に、リモートインタフェースの例を示します。

```
import javax.ejb.*;
import java.rmi.*;

public interface MyEntity
   extends EJBObject
   {
   public String getAddress() throws RemoteException;
   public void setAddress(String addr) throws RemoteException;
}
```

Bean クラス定義の作成 (Bean 管理による持続性)

Bean 管理による持続性を使用するエンティティ Bean の Bean クラスは、`abstract` ではなく `public` として定義する必要があります。Bean クラスは、`javax.ejb.EntityBean` インタフェースを実装する必要があります。次に例を示します。

```
import java.rmi.*;
import java.util.*;
import javax.ejb.*;
public class MyEntityBean implements EntityBean {
   // エンティティ Bean の実装。これらのメソッドは常に取り込む
   必要があります。
   public void ejbActivate() {
   }
   public void ejbLoad() {
   }
   public void ejbPassivate() {
   }
   public void ejbRemove() {
   }
   public void ejbStore() t{
   }
   public void setEntityContext(EntityContext ctx) {
   }
   public void unsetEntityContext() {
   }
   // 他のコードは省略します。
   }
```

これらのメソッドに加えて、エンティティ Bean のクラスでは、1 つ以上の `ejbCreate` メソッドと、`ejbFindByPrimaryKey` 検索メソッドも定義する必要があります。オプションで、`ejbCreate` メソッドごとに `ejbPostCreate` メソッドを定義できます。また、`ejbFind`*XXX* 形式の開発者定義検索メソッドを追加することもあります。*XXX* は、ほかのメソッド名と重複しない連続した固有のメソッド名を示します (例 : `ejbFindApplesAndOranges`)。

通常、エンティティ Beans は、1 つ以上のビジネスメソッドも実装します。これらのメソッドは通常、各 Bean に対して固有であり、その特定の機能を表します。任意のビジネスメソッド名を付けることができますが、この EJB アーキテクチャで使われるメソッド名と重複しないように注意してください。ビジネスメソッドは、`public` として宣言する必要があります。メソッド引数および戻り値は Java RMI の正しいタイプである必要があります。`throws` 句は、アプリケーション固有の例外を定義し、`java.rmi.RemoteException` を含めることもできます。

エンティティ Bean を実装するには、次の 2 つのタイプのビジネスメソッド使用します。

- 内部メソッド - Bean 内のほかのビジネスメソッドによって使用される。Bean の外部からアクセスされることはない
- 外部メソッド - エンティティ Bean のリモートインタフェースによって参照される

次の各項では、エンティティ Bean のクラス定義内のさまざまなメソッドについて説明します。

- ejbCreate の使用
- ejbActivate と ejbPassivate の使用
- ejbLoad と ejbStore の使用
- setEntityContext と unsetEntityContext の使用
- ejbRemove の使用
- 検索メソッドの使用

ejbCreate の使用

エンティティ Bean は、1 つまたは複数の ejbCreate メソッドを実装する必要があります。クライアントが Bean を呼び出すたびに、必ずこのメソッドを 1 つ使います。次に例を示します。

```
public String ejbCreate(String orderId, String customerId,
   String status, double totalPrice)
   throws CreateException {
      try {
      InitialContext ic = new InitialContext();
      DataSource ds = (DataSource) ic.lookup(dbName);
      con =  ds.getConnection();
      String insertStatement =
         "insert into orders values ( ? , ? , ? , ? )";
      PreparedStatement prepStmt =
         con.prepareStatement(insertStatement);
      prepStmt.setString(1, orderId);
      prepStmt.setString(2, customerId);
      prepStmt.setDouble(3, totalPrice);
      prepStmt.setString(4, status);
      prepStmt.executeUpdate();
      prepStmt.close();
   } catch (Exception ex) {
      throw new CreateException("ejbCreate: "
         +ex.getMessage());
   }
}
```

```
public String ejbPostCreate(String orderId, String customerId,String
status, double totalPrice)
   throws CreateException
{
......
......
}
```

各 ejbCreate メソッドは public として宣言し、主キーのタイプを返し、ejbCreate という名前を付ける必要があります。戻り値タイプは、キーとして使う番号に変換する正しい Java RMI タイプであれば何でもかまいません。すべての引数は、正しいタイプの Java RMI である必要があります。throws 句は、アプリケーション固有の例外を定義し、java.ejb.CreateException を含めることもできます。

このメソッドでは、関係を確立します。各 ejbCreate メソッドに対応させて、作成後すぐにエンティティサービスを処理する ejbPostCreate メソッドをエンティティ Bean のクラスで定義することができます。各 ejbPostCreate メソッドは public として宣言し、void を返し、ejbPostCreate という名前を付ける必要があります。メソッド引数がある場合は、対応する ejbCreate メソッドの数および引数タイプと一致させる必要があります。throws 句は、アプリケーション固有の例外を定義し、java.ejb.CreateException を含めることもできます。

ejbActivate と ejbPassivate の使用

サーバーアプリケーションでエンティティ Bean のインスタンスが必要になると、Bean のコンテナは、ejbActivate を起動して Bean インスタンスを使用可能な状態にします。同様に、インスタンスが不要になると、Bean のコンテナは ejbPassivate を起動して、その Bean とアプリケーションの関連付けを解除します。

Bean が最初にアプリケーションで動作可能になったとき、または Bean が不要になったときに、特定のアプリケーションタスクを実行する必要がある場合は、ejbActivate および ejbPassivate メソッド内でその操作をプログラミングします。たとえば、ejbPassivate の実行時にデータベースとバックエンドリソースへの参照を解放し、ejbActivate の実行時にそれを復元することができます。

ejbLoad と ejbStore の使用

エンティティ Bean は同期をとるために、コンテナを使ってデータベースに Bean の状態情報を格納できます。持続性が Bean によって管理されている場合は、ejbLoad および ejbStore をコーディングする必要があります。コンテナは、トランザクションの開始時に ejbLoad を呼び出し、トランザクションが無事に終了した時点で ejbStore を呼び出すことで、Bean の状態とデータベースを同期させます。

ejbStore を実装するとデータベースに状態情報が保存され、ejbLoad を実装するとデータベースから状態情報が取り出されます。

次の例は、アクティブデータの格納と取得を行う ejbLoad メソッドと ejbStore メソッドの定義を示します。

```
public void ejbLoad()
       throws java.rmi.RemoteException
{
   String itemId;
   javax.sql.Connection dc = null;
   java.sql.Statement stmt = null;
   java.sql.ResultSet rs = null;

   itemId = (String) m_ctx.getPrimaryKey();

   System.out.println("myBean:Loading state for item " + itemId);

   String query =
       "SELECT s.totalSold, s.quantity " +
       " FROM Item s " +
       " WHERE s.item_id = " + itemId;

   dc = new DatabaseConnection();
   dc.createConnection(DatabaseConnection.GLOBALTX);
   stmt = dc.createStatement();
   rs = stmt.executeQuery(query);

   if (rs != null) {
       rs.next();
       m_totalSold = rs.getInt(1);
       m_quantity = rs.getInt(2);
   }
}

public void ejbStore()
       throws java.rmi.RemoteException
{
   String itemId;
   itemId = (String) m_ctx.getPrimaryKey();
   DatabaseConnection dc = null;
   java.sql.Statement stmt1 = null;
   java.sql.Statement stmt2 = null;

   System.out.println("myBean:Saving state for item " + itemId);

   String upd1 =
       "UPDATE Item " +
```

```
        " SET quantity = " + m_quantity +
        " WHERE .item_id = " + itemId;

    String upd2 =
        "UPDATE Item " +
        " SET totalSold = " + m_totalSold +
        " WHERE .item_id = " + itemId;

    dc = new DatabaseConnection();
    dc.createConnection(DatabaseConnection.GLOBALTX);
    stmt1 = dc.createStatement();
    stmt1.executeUpdate(upd1);
    stmt1.close();
    stmt2 = dc.createStatement();

    stmt2.executeUpdate(upd2);
    stmt2.close();
}
```

ほかの Bean と同時にトランザクションにアクセスする Bean の遮断レベルの詳細については、78 ページの「同時アクセスの同期化処理」を参照してください。

setEntityContext と unsetEntityContext の使用

コンテナは、エンティティ Bean のインスタンスを作成してコンテナへの Bean のインタフェースを提供してから、setEntityContext を呼び出します。コンテナから渡されるエンティティのコンテキストを格納するには、このメソッドを実装します。後からこの参照を使って、インスタンスの主キーなどを取得することができます。

```
public void setEntityContext(javax.ejb.EntityContext ctx)
{
m_ctx = ctx;
}
```

同様に、コンテナは unsetEntityContext を呼び出して、インスタンスからコンテナ参照を削除します。これは、Bean のインスタンスが削除の対象となる前にコンテナが呼び出す最後の Bean クラスメソッドです。この呼び出しのあと、Java ガベージコレクションメカニズムは最終的にそのインスタンスに対して finalize を呼び出し、そのインスタンスをクリーンアップして廃棄します。

```
public void unsetEntityContext()
{
m_ctx = null;
}
```

ejbRemove の使用

クライアントは、エンティティ Bean のホームインタフェースまたはコンポーネントインスタンスで remove メソッドを呼び出して、データベースから関連するレコードを削除できます。クライアントがエンティティ Bean のホームインタフェースまたはコンポーネントインタフェースで remove メソッドを呼び出した場合、またはカスケード削除処理が行われた場合は、それに対応してコンテナがエンティティ Bean のインスタンスで `ejbRemove` メソッドを呼び出します。

検索メソッドの使用

エンティティ Beans は持続的ででであり、複数のクライアント間で共有され、一度に複数回インスタンス化される可能性があります。したがって、エンティティ Bean は少なくとも 1 つの `ejbFindByPrimaryKey` メソッドを実装する必要があります。このメソッドによって、クライアントとコンテナは特定の Bean インスタンスを検出することができます。すべてのエンティティ Beans は、識別シグネチャとして固有の主キーを提供する必要があります。`ejbFindByPrimaryKey` メソッドを Bean のクラスに実装して、Bean がその主キーをコンテナに返すことができるようにします。

次の例は、`FindByPrimaryKey` の定義を示します。

```
public String ejbFindByPrimaryKey(String key)
       throws java.rmi.RemoteException,
          javax.ejb.FinderException
```

特定のエンティティ Bean のインスタンスを、その Enterprise JavaBean の動作内容、そのインスタンスが操作する値などの条件に基づいて検索する場合があります。特定の実装に関するこれらの検索メソッドの名前は `ejbFindXXX` の形式です。`XXX` は、ほかのメソッド名と重複しない連続した固有のメソッド名を表します (例 : `ejbFindApplesAndOranges`)。

検索メソッドは `public` として宣言され、その引数および戻り値は正しいタイプの Java RMI である必要があります。各検索メソッドの戻り値のタイプは、エンティティ Bean の主キーのタイプか、同じ主キータイプのオブジェクトのコレクションである必要があります。戻り値のタイプがコレクションのとき、次のいずれかである必要があります。

- JDK 1.1 の `java.util.Enumeration` インタフェース
- Java 2 の `java.util.Collection` インタフェース

検索メソッドの `throws` 句は、アプリケーション固有の例外であり、`java.rmi.RemoteException` または `java.ejb.FinderException`、あるいはその両方を含めることができます。

読み取り専用 Beans の使用

読み取り専用 Bean は、EJB クライアントが変更できないエンティティ Bean です。ただし、読み取り専用 Bean が表すデータは、ほかの Enterprise JavaBean や、データベースの直接更新などの方法によって、外部から更新される場合があります。

注 Sun ONE Application Server の今回のリリースでは、Bean 管理による持続性を使用するエンティティ Beans のみを読み取り専用として指定できます。

読み取り専用 Beans は Sun ONE Application Server に固有のもので、Enterprise JavaBeans Specification, v2.0 には規定されていません。

この節では、次の項目について説明します。

- 読み取り専用 Bean の特性とライフサイクル
- 読み取り専用 Bean の開発に関する注意
- 読み取り専用 Bean の更新
- 読み取り専用 Bean の配備

読み取り専用 Bean の特性とライフサイクル

読み取り専用 Bean は、基になるデータがまったく変更されないか、まれにしか変更されない場合に最も適しています。たとえば、外部で更新される特定企業の株式相場を表す場合に読み取り専用 Bean を使用できます。その場合、通常のエンティティ Bean を使用すると `ejbStore` 呼び出しの負荷が発生する可能性がありますが、読み取り専用 Bean を使用すると、これを避けることができます。

読み取り専用 Beans には次の特性があります。

- エンティティ Beans のみを読み取り専用 Beans にすることができる
- Bean 管理による持続性のみを使用できる
- コンテナ管理トランザクションのみを使用できる。読み取り専用 Beans は、独自のトランザクションを開始できない
- 読み取り専用 Beans は Bean の状態を更新しない
- `ejbStore` はコンテナによって呼び出されない
- `ejbLoad` は、トランザクションメソッドが呼び出されたとき、Bean が最初にキャッシュ内に作成されたとき、または Bean の `refresh-period-in-seconds` によって制御される一定間隔でのみ呼び出される

- ホームインタフェースには、find メソッドをいくつでも設定できる。find メソッドの戻り値タイプは、同じ Bean タイプの主キー (または主キーのコレクション) にする必要がある
- Bean が表すデータが変更される可能性がある場合は、refresh-period-in-seconds を設定して、一定間隔でその Bean を更新する必要がある。この一定間隔で、ejbLoad が呼び出される

読み取り専用 Bean は、適切な find メソッドを使って作成されます。

読み取り専用 Bean はキャッシュされ、エンティティ Beans と同じキャッシュプロパティを持ちます。キャッシュのスペースを空けるための削除候補として読み取り専用 Bean が選択されると、ejbPassivate が呼び出され、その Bean は空きプールに戻されます。空きプールにある Bean は ID を持たず、検索リクエストの処理にのみ使用されます。

読み取り専用 Bean は、通常の読み取り / 書き込みエンティティ Beans と同様にネーミングサービスに結合されるので、クライアントは、読み取り / 書き込みエンティティ Beans を検索する場合と同じ方法で読み取り専用 Bean を検索できます。

読み取り専用 Bean の開発に関する注意

- ホームインタフェースでは、create または remove メソッドを含めない
- メソッドのトランザクション属性には、有効な EJB 2.0 トランザクション属性を使用する

  TX_SUPPORTED を含める理由は､同じトランザクションでコミットされないデータを読み取るためです。また、TX 属性を使用して ejbLoad を強制することもできます。

読み取り専用 Bean の更新

読み取り専用 Bean を更新するには、次に示すようにいくつかの方法があります。

- トランザクションメソッドの起動
- 定期的な更新
- プログラムによる更新

トランザクションメソッドの起動

トランザクションメソッドを起動することによって、ejbLoad を起動します。

定期的な更新

Sun ONE Application Server 固有の XML ファイルで refresh-period-in-seconds 要素を指定すると、読み取り専用 Bean を定期的に更新できます。

- refresh-period-in-seconds にゼロを指定すると、その Bean は (トランザクションメソッドにアクセスするまで) 更新されない
- 1 以上の値を指定すると、その Bean は、指定した間隔で更新される

注　Sun ONE Application Server の外部でデータ変更される可能性がある場合は、これが Bean の状態を更新する唯一の方法です。

プログラムによる更新

一般に、読み取り専用 Bean によってキャッシュされたデータを更新する Bean では、その状態を更新するように読み取り専用 Bean に知らせる必要があります。ReadOnlyBeanNotifier を使用すると、読み取り専用 Beans を強制的に更新することができます。これを行うには、ReadOnlyBeanNotifier Bean で次のメソッドを起動します。

```
public interface ReadOnlyBeanNotifier
   extends java.rmi.Remote
{
   refresh(Object PrimaryKey)
      throws RemoteException;
}
```

ReadOnlyBeanNotifier のインタフェースは、コンテナによって実装されます。ユーザーは、次のコードを使って ReadOnlyBeanNotifier を検索できます。

```
com.sun.ejb.ReadOnlyBeanNotifier notifier =
com.sun.ejb.containers.ReadOnlyBeanHelper.getReadOnlyBeanNotifier
   (<ejb-name-of -the-target>);
   notifier.refresh(<PrimaryKey>);
```

読み取り専用 Beans によってキャッシュされたデータを更新する Beans では、refresh メソッドを呼び出す必要があります。その読み取り専用 Bean に対する次回の (トランザクションでない) 呼び出しでは、ejbLoad が起動します。

読み取り専用 Bean の配備

読み取り専用 Beans は、ほかのエンティティ Beans と同じ方法で配備されます。ただし、Sun ONE Application Server 固有の XML ファイル内の Bean のエントリでは、`is-read-only-bean` 要素を true に設定する必要があります。つまり、次のように設定します。

```
<is-read-only-bean>true</is-read-only-bean>
```

また、`refresh-period-in-seconds` 要素に値を設定して、Bean が更新される間隔を指定することもできます。この要素が存在しない場合は、デフォルトの 600 (秒) とみなされます。

同じトランザクションコンテキストを持つ要求はすべて、同じ 読み取り専用 Bean インスタンスに転送されます。配備するときに、`allow-concurrent-access` 要素を true (同時アクセスを可能にする)、または false (同じ読み取り専用 Bean への同時アクセスを直列化する) に設定することによって、複数の要求を直列化する必要があるかどうかを指定できます。デフォルトは false です。

これらの要素に関する詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

同時アクセスの同期化処理

エンティティ Bean の開発者は、通常は、複数のトランザクションからのエンティティ Bean への同時アクセスについて考慮する必要はありません。このような場合、Bean のコンテナは自動的に同期をとります。Sun ONE Application Server では、コンテナは、Bean を使う同時発生トランザクションごとに 1 つのエンティティ Bean インスタンスを活性化します。

トランザクション同期は、データベースアクセス呼び出し時に基礎となっているデータベースによって自動的に実行されます。通常は、基礎となっているデータベースやリソースと連携して同期をとります。その方法の 1 つは、たとえば適切な遮断レベルを選択したり、`select for update` 句を使用したり、`ejbLoad` メソッド内で対応するデータベースをロックすることです。その特性は、使用しているデータベースによって異なります。

詳細は、『Enterprise JavaBeans Specification, v2.0』の同時アクセスに関連する箇所を参照してください。

第 4 章

エンティティ Beans のコンテナ管理による持続性の使用

この章では、Sun ONE Application Server 7 環境でのコンテナ管理による持続性の動作に関する情報、および実装の手順を示します。

注 コンテナ管理による持続性を実装するには、エンティティ Beans について理解している必要があります。エンティティ Beans については、55 ページの「エンティティ Beans の使用」で説明しています。

この節には次の項目があります。

- Sun ONE Application Server でのサポート
- コンテナ管理による持続性について
- コンテナ管理による持続性の使用
- サードパーティ製のプラグイン可能な持続性管理 API
- 制限事項と最適化
- sun-cmp-mappings.xml ファイルの要素
- 例

コンテナ管理による持続性に関する詳細情報は、『Enterprise JavaBeans Specification, v2.0』の第 10 章、第 11 章、および第 14 章にあります。

Sun ONE Application Server でのサポート

Sun ONE Application Server でのコンテナ管理による持続性のサポート内容は、次のとおりです。

- J2EE v 1.3 仕様のコンテナ管理持続モデルのフルサポート
  - トランザクションのコミットオプション B、C のサポートは、『Enterprise JavaBeans Specification, v2.0』に定義されている。詳細は、147 ページの「コミットオプション」を参照
  - 主キークラスは、`java.lang.Object` のサブクラスである必要がある。これにより移植性が保証される。これは、基本的なタイプ (`int` など) を主キークラスとして一覧表示できるベンダーもあるので記載している
- 次の機能を提供する Sun ONE Application Server のコンテナ管理による持続性の実装
  - コンテナ管理による持続性を使用する Beans を含む EJB JAR ファイルの XML 配備記述子を作成するオブジェクトとリレーショナル間の (O/R) マッピングツール (Sun ONE Application Server Assembly Tool の一部)
  - 複合 (マルチカラム) 主キーのサポート
  - 高度なカスタム検索メソッドのサポート
  - 標準ベースクエリ言語 (EJB QL)
  - コンテナ管理による持続性のランタイムによる次の JDBC ドライバ / データベースのサポート
    - Oracle 8i、Oracle 9
    - Sybase 12
    - Microsoft SQLServer 2000
    - Pointbase 4.2 (Solaris と同時にプリインストールした Sun ONE Application Server には含まれない)
- サードパーティによるオブジェクトとリレーショナル間の (O/R) マッピングツールのサポート。サードパーティ API については、109 ページの「サードパーティ製のプラグイン可能な持続性管理 API」を参照

コンテナ管理による持続性について

コンテナ管理による持続性 (CMP) を使用するエンティティ Bean では、その状態 (または持続性) の管理を Sun ONE Application Server コンテナに委託します。コンテナ管理による持続性を実装する開発者は、Bean 管理による持続性の実装に必要な JDBC コードを記述する代わりに、ツールを使用して Bean 配備記述子を作成します。作成した配備記述子は、リレーショナルデータベースのカラムに Bean フィールドをマップするときにコンテナが使う情報を提供します。

EJB コンテナでコンテナ管理による持続性をサポートするには、次の 2 つを必要とします。

- マッピング - リレーショナルデータベースの表などのリソースにエンティティ Beans をマップする方法に関する情報
- 実行時環境 - マッピング情報を使って各 Bean で持続性操作を実行する、コンテナ管理による持続性の実行時環境

この節では、コンテナ管理による持続性に関する次の項目について説明します。

- CMP コンポーネント
- 関係
- 抽象スキーマ
- 配備記述子
- 持続性マネージャ

CMP コンポーネント

Bean 管理による持続性とは異なり、コンテナ管理による持続性では、エンティティ Bean クラスのメソッドにデータベースアクセス呼び出しを記述する必要はありません。持続性は実行時にコンテナによって処理されるので、コンテナがデータアクセスを処理する持続性フィールドおよび関係を配備記述子に指定する必要があります。持続的なデータにアクセスするには、抽象持続性スキーマとして定義されたアクセサメソッドを使用します。

コンテナ管理による持続性を使用するエンティティ Bean は、次のような相互運用コンポーネントで構成されます。

- 開発者が作成した抽象 Bean クラス
- 開発者が作成したリモートインタフェース
- 開発者が作成したローカルインタフェース
- 開発者が作成した配備記述子

- 開発者が作成した主キークラス (省略可能)
- コンテナ管理による持続性の実装によって生成された具象 Bean クラス

  このクラスは抽象 Bean クラスを継承し、配備記述子からの情報を使用する。Bean クラス内のアクセサ (読み取り) メソッドとミュテータ (書き込み) メソッドは、このクラスで具象状態クラスに実装される
- コンテナ管理による持続性の実装によって生成された具象リモート Bean 実装クラス
- コンテナ管理による持続性の実装によって生成された EJBObject (スケルトン)
- コンテナ管理による持続性の実装によって生成されたリモートスタブ

コンテナ管理による持続性では、次のクラスが使用されます。

- 生成クラス - `ejbc` コンパイルユーティリティから呼び出され、具象クラスを生成する
- 生成済みクラス - コンテナ管理による持続性を使用して、サーバー実行時の持続性動作をもたらす
- 管理クラス - サーバー実行時の統計情報を収集および報告する

関係

注 この項は、コンテナ管理による持続性 2.0 Beans を使用する場合だけ適用されます。

関係により、オブジェクトから関連オブジェクトに移動することができます。関係の方向には、双方向と一方向があります。

- 双方向 - 各エンティティ Bean が、他方の Bean を参照する関係フィールドを持つ。関係フィールドにより、エンティティ Bean のコードで関連オブジェクトにアクセスできる。エンティティ Bean が関係ールドを持つ場合、そのエンティティ Bean は関連オブジェクトについて「知っている」と言える
- 一方向 - 一方のエンティティ Bean のみが、他方の Bean を参照する関係フィールドを持つ

注 関係が一方向であっても、その関係に変更を加えれば、その関係によって関連付けられているほかの Enterprise JavaBean も影響を受けます。

2 つのクラス内のフィールド間のコンテナ管理関係 (CMR) により、関係の片方での操作を他方にも反映することができます。実行時に、あるインスタンスのフィールドが別のインスタンスを参照するように変更されると、参照先のインスタンスの関係フィールドも、関係の変更を反映するように修正されます。

注 管理関係内の 1 つのオブジェクトを削除する場合、警告は表示されません。コンテナ管理による持続性は、確認を求めずに、外部キー側の関係を無効にしてオブジェクトを削除します。

Java コードでは、基本的な関係によって、オブジェクト参照 (EJB ローカルインタフェースに入力されたコレクションまたはフィールド) で関係を表します。関係に含まれる各クラスのインスタンス数によって、一対一、一対多、または多対多の関係があります。データベースでは、これを外部キーカラムによって表し、多対多の関係では、表を結合することができます。

次の各節では、各種の関係について説明します。

- 一対一の関係
- 一対多の関係
- 多対多の関係

一対一の関係

一対一の関係では、両方のクラス内に、タイプとして他方の Bean タイプのローカルインタフェースを持つ単一値フィールドがあります。関係のどちらか一方のフィールドが変更されると、関係の変更として処理されます。一方の側のフィールドが null から null 以外に変更されると、もう一方の側のフィールドは、このインスタンスを参照するように変更されます。もう一方の側のフィールドがすでに null 以外になっている場合は、その変更を行う前に、その関係が null にされます。

一対多の関係

一対多の関係では、多側に単一値フィールドがあり、一側に複数値フィールド (コレクション) があります。

コレクションフィールドにインスタンスが追加されると、新しいインスタンスのフィールドは、そのコレクションフィールドを保有するインスタンスを参照するように更新されます。コレクションからインスタンスが削除されると、そのインスタンスのフィールドは null になります。

多側のフィールドの追加または削除は、関係の変更として処理されます。多側のフィールドが null から null 以外に変更されると、一側のコレクション値フィールドにそのインスタンスが追加されます。多側のフィールドが null 以外から null に変更されると、一側のコレクション値フィールドからそのインスタンスが削除されます。

多対多の関係

多対多の関係では、関係の両側に複数値 (コレクション) フィールドがあります。関係のどちらかの側のコレクションの内容が変更されると、関係の変更として処理されます。一方の側のコレクションにインスタンスが追加されると、このインスタンスがもう一方の側のコレクションに追加されます。一方の側のコレクションからインスタンスが削除されると、このインスタンスがもう一方の側のコレクションから削除されます。

抽象スキーマ

エンティティ Beans の配備記述子の一部である抽象スキーマは、Bean の持続フィールドおよび関係を定義します。抽象という言葉によって、このスキーマは、基になるデータストアの物理スキーマと区別されます。

抽象スキーマの名前は、配備記述子に指定します。この名前は、EJB クエリ言語 (EJB QL) で記述されたクエリによって参照されます。コンテナ管理による持続性を使用するエンティティ Beans では、すべての検索メソッドに対して EJB-QL クエリを定義する必要があります (`findByPrimaryKey` を除く)。EJB-QL クエリは、検索メソッドが起動されたときに EJB コンテナが実行するクエリを決定します。

例

```
<ejb-relation>
   <ejb-relation-name>OrderLineItem</ejb-relation-name>
   <ejb-relationship-role>
       <ejb-relationship-role-name>
          OrderHasLineItems
       </ejb-relationship-role-name>
       <multiplicity>One</multiplicity>
       <relationship-role-source>
          <ejb-name>Order</ejb-name>
       </relationship-role-source>
       <cmr-field>
          <cmr-field-name>lineItems</cmr-field-name>
          <cmr-field-type>java.util.Collection</cmr-field-type>
       </cmr-field>
   </ejb-relationship-role>
<ejb-relationship-role>
   <ejb-relationship-role-name>
```

```
        LineItemInOrder
    </ejb-relationship-role-name>
    <multiplicity>Many</multiplicity>
    <relationship-role-source>
            <ejb-name>LineItemEJB</ejb-name>
        </relationship-role-source>
    </ejb-relationship-role>
</ejb-relation>
```

配備記述子

作成したコンテナ管理フィールドがデータベースフィールドにマップされる場合は、マッピングに関する情報を配備担当者に連絡する必要があります。コンテナ管理による持続性 Beans を含む各モジュールは、配備のために次のファイルが必要です。

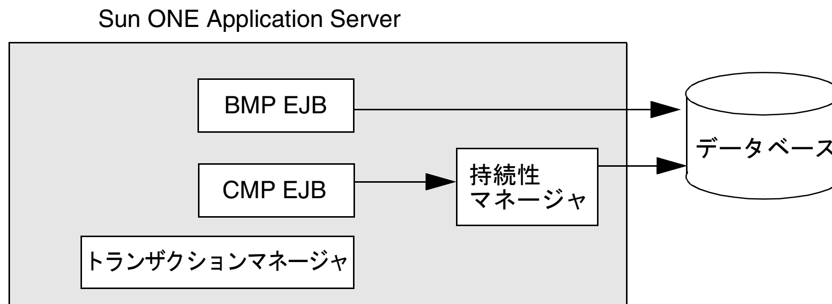
- `ejb-jar.xml` - Bean のトランザクション属性やコンテナ管理される Bean のフィールドなどの情報を格納する
- `sun-ejb-jar.xml` - EJB アセンブリ用の標準ファイル。詳細は、176 ページの「sun-ejb-jar.xml ファイルの要素」および 213 ページの「EJB XML ファイルの例」を参照
- `sun-cmp-mappings.xml` - コンテナ管理による持続性のマッピング用のファイル。詳細は、111 ページの「sun-cmp-mappings.xml ファイルの要素」および 121 ページの「スキーマ定義の例」を参照

持続性マネージャ

Sun ONE Application Server では、コンテナ管理による持続性モデルは、Pluggable Persistence Manager API に基づきます。この API は、エンティティ Beans と持続性ストア間のマッピングを定義およびサポートするときに、持続性マネージャの役割をします。

持続性マネージャは、コンテナにインストールされたエンティティ Beans の持続性を処理するコンポーネントです。持続性マネージャベンダーが提供するクラスは、エンティティ Beans 間の関係および持続的な状態へのアクセスを管理します。持続性マネージャベンダーは、コンテナ管理関係の維持に使用される Java クラスの実装も行います。持続性マネージャは、コンテナから提供されるデータソースレジストリを使ってデータソースにアクセスします。

次の図は、Sun ONE Application Server 環境で持続性がどのように機能するかを示しています。



カスタム持続性マネージャを作成してレガシーシステムをサポートしたり、キャッシュストラテジを実装してコンテナ管理による持続性ソリューションのパフォーマンスを高めることもできます。

コンテナ管理による持続性の使用

コンテナ管理による持続性を使用するエンティティ Beans の実装の主な内容は、マッピングとアセンブリ、配備です。

注 コンテナ管理フィールドに割り当てる Java のタイプは、Java 基本タイプ、Java 直列化可能タイプ、および EJB のリモートインタフェースまたはリモートホームインタフェースの参照に制限する必要があります。

この節には次の項目があります。

- プロセスの概要
- マッピング機能
- マッピングでサポートされているデータタイプ
- BLOB のサポート
- キャプチャスキーマユーティリティの使用
- フィールドおよび関係のマッピング
- リソースマネージャの設定
- EJB QL の使用
- 1.1 検索のクエリの設定

プロセスの概要

コンテナ管理による持続性のプロセスは、マッピング、配備、および実行の 3 つのオペレーションで構成されます。これらのオペレーションについては、次の各フェーズで説明します。

- フェーズ 1. マッピング配備記述子ファイルの作成
- フェーズ 2. 具象 Beans および委託の生成とコンパイル
- フェーズ 3. Sun ONE Application Server 実行時の稼動

フェーズ 1. マッピング配備記述子ファイルの作成

注 Sun ONE Studio IDE は、配備用にこの記述子を自動的に生成します。

このフェーズは、Sun ONE Studio 4 IDE でのコンテナ管理による持続性の開発と並行して実行するか、または開発後の配備準備の段階で実行します。

このフェーズでは、CMP フィールドおよび CMR フィールド (関係) をデータベースにマップします。コンテナ管理による持続性 Bean ごとに主表を選択し、オプションで複数の二次表を選択します。CMP フィールドは、主表または二次表のカラムにマップします。CMR フィールドは、カラムリスト (通常は主キーと外部キーのペアに関連付けられたカラムのリスト) のペアにマップします。

- マッピングは、`sun-cmp-mapping_1_0.dtd` に準拠したファイルに保存される。結果の XML ファイルは、EJB JAR ファイル内のユーザー定義 Bean クラスとともにパッケージ化される。この XML ファイルは `META-INF/sun-cmp-mappings.xml` という名前である必要がある
- エラーは配備プロセスで報告される。エラーは、Sun ONE Studio 4 環境内、またはコマンド行で引き起こされることがある
- マッピング情報は、データベーススキーマファイルと連動して開発される。このファイルは、Sun ONE Studio 4 IDE (217 ページの「スキーマの取り込み」)、またはキャプチャスキーマユーティリティ (93 ページの「キャプチャスキーマユーティリティの使用」) を使って取り込む必要がある
- データベースの表の構造を変更した場合は、データベース管理者が表を更新した後で、まず、変更した表のスキーマを取り込む。次に CMP フィールドと関係をマップし直す

注 この再マッピングを自動的に実行する方法はないので、手動で実行する必要があります。

フェーズ 2. 具象 Beans および委託の生成とコンパイル

このフェーズは、EJB アプリケーションを Sun ONE Application Server に配備するときに実行します。このフェーズでは、フェーズ 1 で作成したマッピング情報と配備情報を結合します。

次のファイルが生成されます。

- 作成した抽象 Bean を拡張する具象 Bean ファイル

  具象 Bean は、EJB ライフサイクルメソッドの `ejbSetEntityContext`、`ejbUnsetEntityContext`、`ejbCreate`、`ejbRemove`、`ejbLoad`、`ejbStore` を実装する。また、各 CMP フィールドと CMR フィールドの `getXXX` と `setXXX`、`ejbFindByPrimaryKey`、その他の検索メソッド、およびユーザーが定義したセレクタメソッドも実装する

- プロパティファイルとして保存された、検索メソッドとセレクタメソッドのコンパイル済み EJB-QL

  このファイルには、コンテナ管理による持続性のクエリパラメータリスト、クエリフィルタ、クエリ順序式、クエリ候補クラス名、およびクエリ結果タイプが格納される

- EJB-QL 構文や使用法などに関するエラーを報告する生成ログファイル
- 状態クラスとヘルパークラス

フェーズ 3. Sun ONE Application Server 実行時の稼動

配備時に提供した情報は、Enterprise JavaBean として実装されたエンティティで要求を処理するときに使用されます。

マッピング機能

マッピングは、データのオブジェクト指向モデルを、関係モデル (通常はリレーショナルデータベースのスキーマ) に関連付ける機能です。コンテナ管理による持続性を実装すると、データおよび関連動作を保持した相互関係のクラスのセットを、相互関係のスキーマのメタデータに関連付けることができます。その後、データベースのこのオブジェクト表現を使用して、Java アプリケーションの基礎を構成できます。また、このマッピングをアプリケーション固有のニーズに応じてカスタマイズし、基礎となるこれらのクラスを最適化することもできます。

結果は単一のデータモデルであり、これを通して持続的なデータベース情報および通常の一時的なプログラムデータの両方にアクセスできます。そのため、Java プログラミング言語オブジェクトだけを理解すればよく、基礎となるデータベーススキーマについて理解する必要がありません。

コンテナ管理による持続性の DTD および XML ファイルの要素については、111 ページの「sun-cmp-mappings.xml ファイルの要素」を参照してください。

マッピングの各機能

Sun ONE Application Server で提供されるマッピング機能を次に示します。

- コンテナ管理による持続性 Bean を単一の表にマップする
- コンテナ管理による持続性 Bean を複数の表にマップする
- コンテナ管理による持続性のフィールドをカラムにマップする
- コンテナ管理による持続性のフィールドを別々のカラムタイプにマップする
- 複合主キーで表をマップする
- コンテナ管理による持続性の関係を外部キーカラムにマップする
- オーバーラップする主キーおよび外部キーで表をマップする

マッピングツール

マッピングツールは、エンティティ Bean のコンテナ管理フィールドを、リレーショナルデータベースの表のカラムなどのデータソースにマップするための情報を生成します。このマッピング情報は、XML ファイルに保存されます。

コンテナ管理による持続性の実装の *meet-in-the-middle* マッピングでは、マッピングツールを使用して、既存のスキーマと既存の Java クラス間のカスタムマッピングを作成します。

マッピングの技法

コンテナ管理による持続性のクラスは、従業員や部署などのデータエンティティを表します。特定のデータエンティティをモデル化するには、データストア内のカラムに対応するクラスに持続的なフィールドを追加します。

もっとも単純なモデル化の方法は、持続性機能のあるクラスで、データストア内の 1 つの表を表すことです。このとき、表のカラムごとに持続的なフィールドを設定します。たとえば、`Employee` クラスに、データストアの `EMPLOYEE` 表内にあるすべてのカラム (`lastname`、`firstname`、`department`、`salary` など) と対応する持続的なフィールドを設定します。

注　　持続的なフィールドとして使用するデータストアのカラムのサブセットのみを設定できますが、フィールドが持続的な場合は、そのフィールドをマップする必要があります。

Sun ONE Studio 4 を使って Enterprise JavaBean 用にコンテナ管理による持続性をマップする方法については、Sun ONE Studio 4 のオンラインヘルプで「Sun ONE Application Server Integration Module」を参照してください。

マッピングでサポートされているデータタイプ

コンテナ管理による持続性は、Java データフィールドを SQL タイプにマップするときに使用される JDBC 1.0 SQL データタイプセットをサポートしています。サポートしている JDBC 1.0 SQL データタイプは次の表のとおりです。

サポートしている JDBC 1.0 SQL データタイプ		
BIGINT	DOUBLE	SMALLINT
BIT	FLOAT	TIME
BLOB	INTEGER	TIMESTAMP
CHAR	LONGVARCHAR	TINYINT
DATE	NUMERIC	VARCHAR
DECIMAL	REAL	

推奨されるマッピングは次の表のとおりです。

データタイプの推奨マッピング

Java のタイプ	JDBC のタイプ	NULL/NON NULL
boolean	BIT	NON NULL
java.lang.Boolean	BIT	NULL
byte	TINYINT	NON NULL
java.lang.Byte	TINYINT	NULL
double	FLOAT	NON NULL
java.lang.Double	FLOAT	NULL
double	DOUBLE	NON NULL
java.lang.Double	DOUBLE	NULL
float	REAL	NON NULL
java.lang.Float	REAL	NULL

データタイプの推奨マッピング (続き)

Java のタイプ	JDBC のタイプ	NULL/NON NULL
int	INTEGER	NON NULL
java.lang.Integer	INTEGER	NULL
long	BIGINT	NON NULL
java.lang.Long	BIGINT	NULL
long	DECIMAL (scale==0)	NON NULL
java.lang.Long	DECIMAL (scale==0)	NULL
long	NUMERIC (scale==0)	NON NULL
java.lang.Long	NUMERIC (scale==0)	NULL
short	SMALLINT	NON NULL
java.lang.Short	SMALLINT	NULL
java.math.BigDecimal	DECIMAL (scale!=0)	NON NULL
java.math.BigDecimal	DECIMAL (scale!=0)	NULL
java.math.BigDecimal	NUMERIC	NULL
java.math.BigDecimal	NUMERIC	NON NULL
java.lang.String	CHAR	NON NULL
java.lang.String	CHAR	NULL
java.lang.String	VARCHAR	NON NULL
serializable	BLOB	NULL

BLOB のサポート

Binary Large Object (BLOB) は、複雑なオブジェクトフィールドの格納と取得に使用されるデータタイプです。BLOB は、画像など、大きいバイト配列に変換し、その後 CMP フィールドに直列化されるバイナリまたは直列化可能なオブジェクトです。

注 Oracle thin ドライバ (Type 4 JDBC) を使用する Oracle では 2000 バイトを超えるデータをカラムに挿入することはできません。この問題を回避するには、OCI ドライバ (Type 2 JDBC) を使います。

Sun ONE Application Server 環境で BLOB のサポートを有効にするには、次のようにします。

1. Bean クラスで変数のタイプを直列化可能として宣言します。
2. XML ファイルを編集して、`sun-cmp-mappings.xml` ファイル内で CMP マッピング配備記述子を宣言します。
3. データベースに BLOB を作成します。

注 BLOB オブジェクトはサイズが大きいため、パフォーマンスに悪影響を及ぼすことがあります。

例

```
<cmp-field-mapping>
   <field-name>syllabus</field-name>
   <column-name>COURSE.SYLLABUS</column-name>
</cmp-field-mapping>
```

例

```
/*******************************************************
Serializable class Syllabus : BLOB Testing
*******************************************************/

package collegeinfo
public class Syllabus implements java.io.Serializable
{
   public String author;
   public String syllabi;
}

Schema for Course:
```

```
table course
------------
courseId Number
deptId Number
courseName Varchar
syllabus BLOB
```

キャプチャスキーマユーティリティの使用

マッピング情報を開発するときは、まず、データベーススキーマを取り込みます。マッピングと実行のためにデータベースメタデータ (スキーマ) を格納するときは、capture-schema コマンドを使います。また、Sun ONE Studio (従来の Forte for Java) IDE を使ってデータベーススキーマを取り込むこともできます。詳細については、217 ページの「スキーマの取り込み」を参照してください。

構文

capture-schema -dburl *url* -username *name* -password *password* -driver *ajdbcdriver* [-schemaname *name*] [-table *TableName*]* [-out *filename*]

指定箇所 :

-dburl *url*: ドライバがデータベースにアクセスするための JDBC URL を指定する

-username *name*: データベースへのアクセスを認証するユーザー名を指定する

-password *password*: 指定したデータベースにアクセスするためのパスワードを指定する

-driver *ajdbcdriver*: JDBC ドライバのクラス名を指定する。このクラスは、作業環境の CLASSPATH に含まれている必要がある

-schemaname *name*: 取り込むユーザースキーマの名前を指定する。指定しない場合は、指定したユーザーがアクセスできるすべてのスキーマからすべての表のメタデータが取り込まれる

注　このパラメータを指定することを強くお勧めします。指定したユーザーが複数のスキーマにアクセスできる場合、同じ名前の複数の表が取り込まれ、問題が生じることがあります。

-table *TableName*: 表名を指定する。複数の表名を指定できる。指定しない場合は、データベーススキーマ内のすべての表が取り込まれる

-out: 出力先を指定する。デフォルトの出力先は stdout である。CMP マッピング用の出力として利用するには、出力ファイル名のサフィックスを .dbschema にする必要がある

コンテナ管理による持続性のマッピングでは、-out パラメータは sun-cmp-mapping_1_0.dtd ファイルの sun-cmp-mapping 要素に含まれる schema サブ要素と相互に関連する

```
<!ELEMENT sun-cmp-mapping (schema, entity-mapping+) >
```

sun-cmp-mappings.xml ファイルでは、この要素は .dbschema というサフィックスなしで表される。たとえば、次のように表示される

```
<schema>RosterSchema</schema>
```

注　表フラグを指定しない場合は、データベース内のすべての表がスキーマに取り込まれます。

例

```
capture-schema -dburl jdbc:pointbase:server://localhost:9092/sample
-username public -password public -driver
com.pointbase.jdbc.jdbcUniversalDriver -out RosterSchema.dbschema
```

フィールドおよび関係のマッピング

ここでは、sun-cmp-mappings.xml 配備記述子を編集して、エンティティ Beans のフィールドおよび関係をマップする方法を説明します。この作業は、手動で行う (XML の編集に精通している場合のみ) か、または Sun ONE Application Server のアセンブルと配備用のツールを使用します。

コンテナ管理による持続性は、名前、主表、1 つ以上のフィールド、0 個以上の関係、0 個以上の二次表、および整合性チェック用フラグを持ちます。sun-cmp-mappings.xml ファイルの要素を使用して、CMP フィールドおよび CMR フィールドをデータベースにマップする必要があります。CMP フィールドはデータベースの主表または二次表のカラムにマップし、CMR フィールドはカラムリストのペアにマップします。

コンテナ管理による持続性の配備記述子のマッピング要素は、111 ページの「sun-cmp-mappings.xml ファイルの要素」にアルファベット順に記載されています。サンプルの XML ファイルについては、121 ページの「スキーマ定義の例」を参照してください。

ここでは、次のマッピング作業を実施するための手順を示します。

- マップする Beans の指定
- マッピングコンポーネントの指定
- フィールドマッピングの指定
- 関係の指定

マップする Beans の指定

最初に、次の要素を使用して、データベーススキーマと、マッピングの対象となるコンテナ管理による持続性 Bean を指定する必要があります。

- sun-cmp-mappings
- sun-cmp-mapping
- schema
- entity-mapping

sun-cmp-mappings

EJB JAR コレクションにマップされるすべての Beans の、サブ要素のコレクションを指定します。

サブ要素は `sun-cmp-mapping` です。

例

121 ページの「スキーマ定義の例」を参照してください。

sun-cmp-mapping

特定のスキーマにマップする Bean を指定します。

サブ要素は `schema`、`entity-mapping` です。

schema

スキーマファイルへのパスを指定します。必要な指定は 1 つだけです。詳細については、124 ページの「EJB QL クエリの例」および 217 ページの「スキーマの取り込み」を参照してください。

例

```
<schema>RosterSchema</schema>
```

entity-mapping

データベースカラムへの Beans のマッピングを指定します。

サブ要素は ejb-name、table-name、cmp-field-mapping、cmr-field-mapping、secondary-table、consistency です。

例

96 ページの「entity-mapping」を参照してください。

マッピングコンポーネントの指定

次の要素を使用して、マッピングの一部となるコンポーネントを指定し、整合性検査の動作を指定します。

- entity-mapping
- ejb-name
- table-name
- secondary-table
- consistency

entity-mapping

データベースカラムへの Beans のマッピングを指定します。

サブ要素は ejb-name、table-name、cmp-field-mapping、cmr-field-mapping、secondary-table、consistency です。

例

```
<entity-mapping>
   <ejb-name>Player</ejb-name>
   <table-name>PLAYER</table-name>
   <cmp-field-mapping>
       <field-name>salary</field-name>
       <column-name>PLAYER.SALARY</column-name>
   </cmp-field-mapping>
   <cmp-field-mapping>
       <field-name>playerId</field-name>
       <column-name>PLAYER.PLAYER_ID</column-name>
   </cmp-field-mapping>
   <cmp-field-mapping>
       <field-name>position</field-name>
       <column-name>PLAYER.POSITION</column-name>
   </cmp-field-mapping>
       <field-name>name</field-name>
```

```
        <column-name>PLAYER.NAME</column-name>
    </cmp-field-mapping>
    <cmr-field-mapping>
        <cmr-field-name>teamId</cmr-field-name>
        <column-pair>
            <column-name>PLAYER.PLAYER_ID</column-name>
            <column-name>TEAMPLAYER.PLAYER_ID</column-name>
        </column-pair>
        <column-pair>
            <column-name>TEAMPLAYER.TEAM_ID</column-name>
            <column-name>TEAM.TEAM_ID</column-name>
        </column-pair>
    </cmr-field-mapping>
</entity-mapping>
```

ejb-name

`ejb-jar.xml` ファイルで、コンテナ管理による持続性 Beans が関係するエンティティ Bean の名前を指定します。必要な指定は 1 つだけです。

例

```
<ejb-name>Player</ejb-name>
```

table-name

データベースの表の名前を指定します。この表はデータベーススキーマファイル内に存在している必要があります。必要な指定は 1 つだけです。

例

```
<table-name>PLAYER</table-name>
```

secondary-table

Bean の二次表を指定します。この指定は省略可能です。

サブ要素は `table-name`、`column-pair` です。

例

この二次表の例は、`StudentEjb` クラスに電子メールのフィールドを追加しています。

```
public abstract class StudentEJB implements EntityBean {

    /****************************************************
    Write ur set,get methods for Entity bean variables and
    business methods here
    ****************************************************/
```

```
//Access methods for CMP fields
public abstract Integer getStudentId();
public abstract void setStudentId(Integer studentId);
public abstract String getStudentName();
public abstract void setStudentName(String studentName);

public abstract void setEmail(String Email); <----- 二次表
                                                    からのカラム
```

外部キーを使って Student と Email 表を関連づける必要があります。Email 表のスキーマは、次の例のようになります。

```
Table Email:
------------
Student_id Number
email varchar

Table Student:
--------------
StudentId Number
StudentName varchar
deptId Number
AddressId Number
AccountId Varchar
```

二次表を追加すると、両方の表が同じ Enterprise JavaBean に適用されます。

consistency

Bean 内のデータに関するトランザクションの整合性を保証する際のコンテナの動作を指定します。この指定は省略可能です。整合性検査フラグ要素が存在しない場合は、none とみなされます。

次の表は、整合性検査で使用する要素を示します。

整合性フラグ

フラグ要素	説明
check-all-at-commit	Sun ONE Application Server 7 では、このフラグは実装されない
check-modified-at-commit	コミット時に、変更されたインスタンスがあるかどうかをチェックする
lock-when-loaded	データの読み込み時にロックを実装する
lock-when-modified	Sun ONE Application Server 7 では、このフラグは実装されない
none	整合性検査は行われない

フィールドマッピングの指定

フィールドマッピングには、次の要素を使用します。

- cmp-field-mapping
- field-name
- column-name
- read-only
- fetched-with
- level
- named-group
- none

cmp-field-mapping

cmp-field-mapping 要素は、1 つのフィールドを、マッピング先の 1 つまたは複数のカラムに関連付けます。Bean の主表のカラム、または定義された二次表のカラムのどちらでも構いません。フィールドを複数のカラムにマップした場合、データベースから値を取得するときは、最初のカラムが SOURCE として使用されます。カラムは表示される順に更新されます。EJB JAR ファイルに定義された cmp-field-mapping 要素ごとに 1 つの cmp-field 要素があります。

フィールドを読み取り専用として指定できます。

サブ要素は field-name、column-name、read-only、および fetched-with です。

例

```
<cmp-field-mapping>
   <field-name>name</field-name>
   <column-name>LEAGUE.NAME</column-name>
</cmp-field-mapping>
```

field-name

フィールドの Java 識別子を指定します。この識別子は、マップする field-name の cmp-field サブ要素の値と一致している必要があります。1 つ指定する必要があります。

例

```
<field-name>name</field-name>
```

column-name

主表のカラム名、または二次表か関連する表のカラムの表修飾名 (TABLE.COLUMN) を指定します。1 つ以上指定する必要があります。

注 複数のカラムをマップするときは、任意の JAVA タイプを使用できます。

例

```
<column-name>PLAYER.NAME</column-name>
```

例

複数の場所に同じ情報が表示される標準化されていない表で使う場合は、更新のたびに情報の同期をとる必要があります。

```
public abstract class StudentEJB implements EntityBean {
.
.
.

public abstract String getInstallments();
```

student 表からの 3 つのカラムを、Student Enterprise JavaBean の 1 つの実装カラムにマップできます。

```
Table student:
.
.
.
installment1 Number
installment2 Number
installment3 Number
```

データベース内のすべてのカラムに同じ値が書き込まれます。

read-only

`read-only` フラグは、フィールドが読み取り専用であることを示します。

例

```
<read-only>name</read-only>
```

fetched-with

フィールドおよび関係のフェッチグループ設定を指定します。フィールドは、階層フェッチグループまたは独立フェッチグループに属することができます。この指定は省略可能です。

fetched-with 要素のデフォルト値は、コンテキストによって異なります。

- `cmp-field-mapping` のサブ要素 `fetched-with` が含まれない場合は、デフォルト値は次のようになります。

```
<fetched-with><level>0</level></fetched-with>
```

- `cmr-field-mapping` のサブ要素 `fetched-with` が含まれない場合は、デフォルト値は次のようになります。

```
<fetched-with><none/></fetched-with>
```

サブ要素は `level`、`named-group`、または `none` です。

level

階層フェッチグループの名前を指定します。値は整数にする必要があります。等しい (または小さい) 値の階層フェッチグループに属するフィールドおよび関係が同時にフェッチされます。`level` の値は 0 よりも大きくする必要があります。1 つだけ指定できます。

named-group

独立フェッチグループの名前を指定します。指定したグループに属するすべてのフィールドおよび関係が同時にフェッチされます。1 つだけ指定できます。

none

持続性レベルのフラグで、このフィールドまたは関係が、それ自体によってフェッチされているかを示します。

関係の指定

コンテナ管理される関係のマッピングの指定には、次の要素を使用します。

- cmr-field-mapping
- cmr-field-name
- column-pair
- fetched-with

cmr-field-mapping

コンテナ管理関係フィールドには、関係を定義する名前および 1 組以上のカラムペアがあります。cmr-field ごとに 1 つの cmr-field-mapping 要素があります。関係をフェッチグループに入れることもできます。

サブ要素は cmr-field-name、column-pair、fetched-with です。

例

```
<cmr-field-mapping>
   <cmr-field-name>teamId</cmr-field-name>
   <column-pair>
       <column-name>PLAYER.PLAYER_ID</column-name>
       <column-name>TEAMPLAYER.PLAYER_ID</column-name>
   </column-pair>
   <column-pair>
       <column-name>TEAM.TEAM_ID</column-name>
       <column-name>TEAMPLAYER.TEAM_ID</column-name>
   </column-pair>
   <fetched-with>
       <none/>
   </fetched-with>
</cmr-field-mapping>
```

cmr-field-name

フィールドの Java 識別子を指定します。この識別子は、マップする cmr-field の cmr-field-name サブ要素の値と一致している必要があります。1 つ指定する必要があります。

例

```
<cmr-field-name>team</cmr-field-name>
```

column-pair

2 つのデータベースの表内の関連カラムのペアの名前を指定します。1 つ以上指定する必要があります。

カラム名は column-name 要素に指定されます。

例

```
<column-pair>
   <column-name>PLAYER.PLAYER_ID</column-name>
   <column-name>TEAMPLAYER.PLAYER_ID</column-name>
</column-pair>
```

column-name

主表のカラム名、または二次表か関連する表のカラムの表修飾名 (TABLE.COLUMN) を指定します。column-pair のサブ要素として 2 つ指定する必要があります。

例

```
<column-name>PLAYER.NAME</column-name>
```

fetched-with

フィールドおよび関係のフェッチグループ設定を指定します。フィールドは、階層フェッチグループまたは独立フェッチグループに属することができます。この指定は省略可能です。

fetched-with 要素のデフォルト値は、コンテキストによって異なります。

- cmp-field-mapping のサブ要素 fetched-with が含まれない場合は、デフォルト値は次のようになります。

```
<fetched-with><level>0</level></fetched-with>
```

- cmr-field-mapping のサブ要素 fetched-with が含まれない場合は、デフォルト値は次のようになります。

```
<fetched-with><none/></fetched-with>
```

サブ要素は level、named-group、または none です。

リソースマネージャの設定

コンテナ管理による持続性実装で使用されるリソースマネージャは PersistenceManagerFactory です。これは、Sun ONE Application Server の DTD ファイル sun-server_7_0-0.dtd を使用して設定します。

新しい持続性マネージャの作成については、『Sun ONE Application Server 管理者ガイド』を参照してください。

コンテナ管理による持続性 Beans を含む EJB モジュールを配備するときは、sun-ejb-jar.xml 配備記述子に次の情報を追加する必要があります。

1. 配備に使用する持続性マネージャを指定します。

```
<pm-descriptors>
   <pm-descriptor>
       <pm-identifier>SunONE</pm-identifier>
       <pm-version>1.0</pm-version>
<pm-class-generator>com.iplanet.ias.persistence.internal.ejb.ejbc.J
DOCodeGenerator
```

```
</pm-class-generator>
       <pm-mapping-factory>com.iplanet.ias.cmp.
          NullFactory</pm-mapping-factory>
   </pm-descriptor
   <pm-inuse>
       <pm-identifier>SunONE</pm-identifier>
       <pm-version>1.0</pm-version>
   </pm-inuse>
</pm-descriptors>
```

2. 持続性マネージャのリソースの JNDI 名 (server.xml ファイルの persistence-manager-factory-resource に表示される) と、cmp-resource の JNDI 名を指定します。この名前は、持続性リソースの管理時に使われます。

たとえば、server.xml ファイルに次のような記述があるとします。

```
<persistence-manager-factory-resource
   factory-class="com.sun.jdo.spi.persistence.support.
       sqlstore.impl.PersistenceMan
   gerFactoryImpl"
   enabled="true"
   jndi-name="jdo/pmf"
   jdbc-resource-jndi-name="jdo/pmfPM"
</persistence-manager-factory-resource>
```

CMP リソースを次のように設定します。

```
<cmp-resource>
   <jndi-name>jdo/pmf</jndi-name
</cmp-resource>
```

注 Sun ONE Studio IDE は、配備用にこの記述子の一部に pm-descriptors を自動的に生成します。コンテナ管理による持続性のリソースを設定する方法については、Sun ONE Studio 4 のオンラインヘルプで「Sun ONE Application Server Integration Module」を参照してください。

EJB QL の使用

Enterprise JavaBeans Specification, v2.0 には、新しいクエリ言語 (EJB QL) が規定されています。この言語は、CMP Beans の移植可能な検索クエリ、および select メソッドの定義に使用されます。このクエリは、SQL に似た構文を使い、抽象スキーマのタイプと CMP Beans の関係に基づいて、エンティティオブジェクトまたはフィールドの値を選択します。

検索メソッドは、Bean のホームインタフェース、ローカルホームインタフェース、またはその両方に定義され、同じ Bean のインスタンスを返します。select メソッドは、抽象 Bean クラスだけに定義され、ローカルタイプまたはリモートタイプのエンティティオブジェクト、および同じスキーマの Beans のフィールド値の選択に使用されます。

詳細については、『Enterprise JavaBeans Specification, v2.0』の第 11 章「EJB QL: EJB Query Language for Container-Managed Persistence Query Methods」を参照してください。

124 ページの「EJB QL クエリの例」では、EJB QL で記述されたサンプルクエリを紹介します。

1.1 検索のクエリの設定

Enterprise JavaBeans Specification, v1.1 には、検索メソッドの記述形式が規定されていません。Sun ONE Application Server では、検索メソッドとセレクタメソッドの実装に JDOQL (Java Data Objects Query Language) クエリを使います。EJB 2.0 では、コンテナが EJB QL クエリを自動的に JDOQL にマップします。EJB 1.1 では、このマッピングの一部を開発者が行います。基本となる JDOQL クエリの次の要素を指定できます。

- フィルタ式 - クエリによって返される各オブジェクトに適用される条件を指定する Java ライクな表現。EJB QL の WHERE 句に対応する
- クエリパラメータ宣言 - クエリの 1 つまたは複数の入力パラメータの名前とタイプを指定する。Java 言語の正式なパラメータの構文で記述する
- クエリ変数宣言 - 1 つまたは複数のクエリ変数の名前とタイプを指定する。Java 言語のローカル変数の構文で記述する。クエリ変数は、結合を実装するためのフィルタで使用されることがある

Sun ONE Application Server 固有の配備記述子 (`sun-ejb-jar.xml`) には、EJB 1.1 検索の設定を格納するために、次の要素が用意されています。

```
query-filter
query-params
query-variables
```

Sun ONE Application Server は、EJB 1.1 エンティティ Bean の持続性に対応したクラスを候補クラスとして使用し、JDOQL クエリを構築します。フィルタ、パラメータ宣言、および変数宣言は、開発者が指定したとおりに JDOQL クエリに追加されます。クエリが実行され、検索メソッドのパラメータが `execute` 呼び出しに渡されます。JDOQL クエリの結果セットからのオブジェクトは主キーインスタンスに変換され、EJB 1.1 `ejbFind` メソッドによって返されます。

JDOQL の詳細は、JDO 仕様書 (JSR 12 を参照) に規定されています。次に、EJB 1.1 検索の定義に必要な要素についてまとめます。

クエリのフィルタ式

フィルタ式は、候補クラスのインスタンスごとに評価されるブール式を含んだ文字列です。フィルタを指定しない場合は、デフォルト値は true となります。式を作成するときは Java 言語のように記述しますが、次の点が異なります。

- ラッパークラスのプリミティブとインスタンスの等価性と順序の比較が有効である
- Date フィールドと Date パラメータの等価性と順序の比較が有効である
- String フィールドと String パラメータの等価性と順序の比較が有効である
- 空白 (印字されない文字スペース、タブ、改行文字) は区切り文字として認識され、それ以外の用途として認識されない
- 次の割り当て演算子はサポートされない
    - =、+=、など
    - 前置インクリメントと後置インクリメント
    - 前置デクリメントと後置デクリメント
- オブジェクトの構築を含め、メソッドはサポートされない。ただし、次のメソッドはサポートされる

    ```
    Collection.contains(Object o)
    Collection.isEmpty()
    String.startsWith(String s)
    String.endsWith(String e)
    ```

    さらに、Sun ONE Application Server は次の標準外 JDOQL メソッドをサポートする

    ```
    String.like(String pattern)
    String.like(String pattern, char escape)
    String.substring(int start, int length)
    String.indexOf(String str), String.indexOf(String str, int
    start)
    String.length()
    Math.abs(numeric n), and Math.sqrt(double d)
    ```

- null 値のフィールドは、サブ式が false を返したのと同様に扱われ、`NullPointerException` がスローされる

注 浮動小数点値の比較は、もともと不正確です。このため、浮動小数点値の等価性の比較 (== および !=) には注意が必要です。式に含まれる識別子は、候補クラスの名前スペースに含まれているものとみなされ、宣言したパラメータと変数が追加されたものとして解釈されます。Java 言語では、これは予約語であり、現在評価しているインスタンスを意味します。

次の表現はサポートされています。

- Java 言語で定義され、すべてのタイプに適用される演算子
  - 関係演算子 (==、!=、>、<、>=、<=)
  - ブール演算子 (&、&&、|、||、~、!)
  - 算術演算子 (+、-、*、/)

  文字列の連結は、「文字列 + 文字列」だけがサポートされる
- 演算子の適用優先度を明示的に決定するカッコ
- キャスト演算子
- 比較演算と算術演算での数値オペランドの昇格。昇格の規則は Java の規則と同様である (Java 言語仕様の数値昇格を参照)。BigDecimal、BigInteger、および数値ラッパークラスによって拡張される

クエリのパラメータ

パラメータ宣言は、1 つのパラメータ宣言または複数の宣言をカンマで区切った文字列です。メソッドシグネチャは、Java の構文に準拠します。

クエリ変数

タイプの宣言は、Java のローカル変数宣言の構文に準拠します。

例 1

次のクエリは、Michael という名前のすべての選手を返します。次の文字列リテラルを使って、name フィールドを比較するフィルタが定義されます。

```
"name == ¥"Michael¥""
```

`sun-ejb-jar.xml` ファイルの `finder` 要素は、次のように記述されます。

```
<finder>
   <method-name>findPlayerByName</method-name>
   <query-filter>name == "Michael"</query-filter>
</finder>
```

例 2

このクエリは、指定した価格帯に含まれるすべての製品を返します。価格の上限と下限を指定する 2 つのクエリパラメータ double low および double high が定義されます。フィルタは、クエリパラメータと次の price フィールドを比較します。

```
"low < price && price < high"
```

`sun-ejb-jar.xml` ファイルの `finder` 要素は、次のように記述されます。

```
<finder>
   <method-name>findInRange</method-name>
   <query-params>double low, double high</query-params>
   <query-filter>low &lt; price &amp;&amp; price &lt
   high</query-filter
</finder>
```

例 3

このクエリは、名前を指定した選手より年俸が高額なすべての選手を返します。`java.lang.String name` という名前を検索するクエリパラメータが定義されます。さらに、比較対象となる選手の変数が定義されます。この変数のタイプは、次の Bean に対応する持続性対応クラスのタイプとなります。

```
mypackage.PlayerEJB_170160966_JDOState p
```

フィルタは、このキーワードによって特定される選手の年俸と、名前で指定した選手の年俸を比較します。

```
(this.salary > p.salary) && (p.name == name)
```

`sun-ejb-jar.xml` ファイルの `finder` 要素は、次のように記述されます。

```
<finder>
   <method-name>findByHigherSalary</method-name>
   <query-params>java.lang.String name</query-params>
   <query-filter>
       (this.salary &gt; p.salary) &amp;&amp;
       (p.name ==name)
   </query-filter>
   <query-variables></query-variables
</finder>
```

サードパーティ製のプラグイン可能な持続性管理 API

EJB コンテナ内のコンテナ管理による持続性では、Sun ONE Application Server Pluggable Persistence Manager API を使用して、持続性ベンダーのランタイムを Sun ONE Application Server に統合できます。この API には、配備時、コード生成時、および実行時の統合要件が記述されています。EJB のコンパイル時に、具体的な Bean の実装を実現するためのコールアウトをサポートしています。

Sun ONE Application Server では、コンテナ管理による持続性の実装により、その起動フレームワークを使用して、クラスの読み込みおよび持続性マネージャの登録を行うことができます。Pluggable Persistence Manager API は、トランザクションおよび動的配備に関する統合要件もサポートしています。

一般に、サードパーティのコンテナ管理による持続性ソリューションで『Enterprise JavaBeans Specification, v2.0』に対応するものはすべて、Sun ONE Application Server で動作させることができます。

サードパーティツールを使用するには、次のようにします。

1. サードパーティ O/R マッピングツールを使用して、Enterprise JavaBeans を作成します。
2. Assembly Tool またはコマンド行インタフェースを使って、その Beans を配備します。

サードパーティの持続性ツールでは、実行時に JDBC (Java Database Connectivity) リソースまたは JCA (Java Connector API) リソースを使って、リレーショナルデータソースにアクセスする必要があります。このため、プラグイン可能な持続性マネージャでは、コネクションのプール、トランザクション処理、およびコンテナのセキュリティ管理機能を自動的に使用できます。サードパーティベンダーは、それぞれの具象クラスジェネレータおよびマッピングファクトリをプラグインして、ベンダー固有の有効なマッピングオブジェクトモデルを生成できます。

設定要件では、Bean に対して定義する必要がある次のプロパティを指定します。

- 持続性メカニズム
- 持続性のベンダーとバージョン
- 持続性メカニズムで必要な追加情報

制限事項と最適化

この節では、エンティティ Beans のコンテナ管理による持続性を実装するときに、留意する制限事項およびパフォーマンスの最適化について説明します。

- EAR ファイル内の一意のデータベーススキーマ名
- コンテナ管理による持続性のプロトコルに関する制限事項
- リモートインタフェースに関する制限事項

EAR ファイル内の一意のデータベーススキーマ名

1 つの EAR ファイル内に複数の JAR ファイルがある場合、たとえば `jar1` と `jar2` がある場合は、`jar1` と `jar2` に対応するデータベーススキーマファイルは、一意の完全修飾名を持つ必要があります。

つまり、データベーススキーマファイル名は、EAR ファイル内で一意にする必要があります。

コンテナ管理による持続性のプロトコルに関する制限事項

- データエイリアスの問題 - 複数のエンティティ Beans のコンテナ管理フィールドを、基礎となっているデータベース内の同じデータ項目にマップすると、同じトランザクション内で複数の Beans が呼び出された場合に、エンティティ Beans のデータビューが正しく表示されないことがある
- 過剰な状態読み込み - コンテナは、抽象 Bean の `ejbLoad` メソッドを起動する前に、エンティティオブジェクト全体の状態をコンテナ管理フィールドに読み込む。このため、大きい状態を持つエンティティオブジェクトの場合、ほとんどのビジネスメソッドが状態の一部にしかアクセスする必要がないときは、最大限の性能が得られない可能性がある。これが問題となるときは、あまり使用されないフィールドの `<fetched-with>` 要素を使用する

リモートインタフェースに関する制限事項

コンテナ管理による持続性を使用するエンティティ Bean のリモートインタフェースには、次の制限事項があります。

- コンテナ管理関係で使用されるコンテナ管理関係フィールドまたは持続性コレクションクラスでは、Bean のリモートインタフェースを介して get メソッドおよび set メソッドを表示することはできない

  ただし、エンティティ Bean のコンテナ管理持続性フィールドに対応する get メソッドおよび set メソッドは、Bean のリモートインタフェースから表示できる
- ローカルインタフェースタイプまたはローカルホームインタフェースタイプを、Bean のリモートインタフェースまたはリモートホームインタフェースから表示することはできない
- 関係で使用されるコンテナ管理コレクションクラスを、Bean のリモートインタフェースから表示することはできない

従属値クラスは、リモートインタフェースまたはリモートホームインタフェースで表示可能であり、クライアントの EJB JAR ファイルに格納できます。

sun-cmp-mappings.xml ファイルの要素

169 ページの「Enterprise JavaBean のアセンブルと配備」で、配備用 Enterprise JavaBeans のアセンブルに関する情報およびガイドラインを提供しています。配備に関する詳細情報および手順は、『Sun ONE Application Server 開発者ガイド』を参照してください。

sun-ejb-jar.xml ファイルに含まれる持続性関連要素については、196 ページの「持続性要素」を参照してください。

サンプルの XML ファイルについては、121 ページの「スキーマ定義の例」を参照してください。

この節では、sun-cmp-mappings.xml ファイル内の次の要素について説明します。

- check-all-at-commit
- check-modified-at-commit
- cmr-field-mapping
- cmr-field-name
- column-name
- column-pair

- consistency
- ejb-name
- entity-mapping
- fetched-with
- field-name
- level
- lock-when-loaded
- lock-when-modified
- named-group
- none
- read-only
- schema
- sun-cmp-mapping
- sun-cmp-mappings
- table-name

check-all-at-commit

Sun ONE Application Server 7 では、このフラグは実装されません。

サブ要素

なし

check-modified-at-commit

コミット時に、変更された Bean インスタンスがあるかどうかをチェックするように指示する整合性レベルフラグです。

サブ要素

なし

cmp-field-mapping

cmp-field-mapping 要素は、1 つのフィールドを、マッピング先の 1 つまたは複数のカラムに関連付けます。Bean の主表のカラム、または定義された二次表のカラムのどちらでも構いません。フィールドを複数のカラムにマップした場合、データベースから値を取得するときは、最初のカラムが SOURCE として使用されます。カラムは表示される順に更新されます。EJB JAR ファイルに定義された cmp-field-mapping 要素ごとに 1 つの cmp-field 要素があります。

フィールドを読み取り専用として指定できます。

fetched-with 要素が指定されていない場合は、フィールドはフェッチグループに属することができます。次のようにみなされます。

```
<fetched-with><level>0</level></fetched-with>
```

サブ要素

次の表は、cmp-field-mapping 要素のサブ要素を示します。

cmp-field-mapping のサブ要素

サブ要素	必要指定数	説明
field-name	1 個のみ	フィールドの Java 識別子を指定する。この識別子は、マップする cmp-field の field-name サブ要素の値と一致している必要がある。1 つ指定する必要がある
column-name	1 個以上	主表のカラム名、または二次表か関連表のカラムの表修飾名 (TABLE.COLUMN) を指定する 1 つ指定する必要がある
read-only	0 または 1 個	フィールドが読み取り専用であることを示すフラグ。この指定は省略可能
fetched-with	0 または 1 個	フィールドおよび関係のフェッチグループ設定を指定する。この指定省略可能

cmr-field-mapping

コンテナ管理関係フィールドは、関係を定義する名前および 1 組以上のカラムペアを持ちます。cmr-field ごとに 1 つの cmr-field-mapping 要素があります。関係をフェッチグループに入れることもできます。

fetched-with 要素が存在しない場合は、次の値が適用されます。
<fetched-with><none/></fetched-with>.

サブ要素

次の表は、cmr-field-mapping 要素のサブ要素を示します。

cmr-field-mapping のサブ要素

サブ要素	必要指定数	説明
cmr-field-name	1 個のみ	フィールドの Java 識別子を指定する。マップする cmr-field の cmr-field-name サブ要素の値と一致している必要がある
column-pair	1 個以上	データベースの表内のカラムのペアの名前を指定する
fetched-with	0 または 1 個	フィールドおよび関係のフェッチグループ設定を指定する。この指定は省略可能

cmr-field-name

フィールドの Java 識別子を指定します。マップする cmr-field の cmr-field-name サブ要素の値と一致している必要があります。

サブ要素

なし

column-name

主表のカラム名、または二次表か関連表のカラムの表修飾名 (TABLE.COLUMN) を指定します。1 つ指定する必要があります。

サブ要素

なし

column-pair

2 つのデータベースの表内の関連カラムのペアの名前を指定します。1 つ指定する必要があります。

サブ要素

次の表は、column-pair 要素のサブ要素を示します。

column-pair のサブ要素

サブ要素	必要指定数	説明
column-name	2 個	主表のカラム名、または二次表か関連表のカラムの表修飾名 (TABLE.COLUMN) を指定する

consistency

Bean 内のデータに関するトランザクションの整合性を保証する際のコンテナの動作を指定します。この指定は省略可能です。整合性検査フラグ要素が存在しない場合は、none とみなされます。

サブ要素

次の表は、整合性検査で使用する要素を示します。

整合性フラグ

フラグ要素	説明
check-all-at-commit	コミット時に、変更されたインスタンスがあるかどうかをチェックする
check-modified-at-commit	Sun ONE Application Server 7 では、このフラグは実装されない
lock-when-loaded	データの読み込み時に排他的なロックを取得する
lock-when-modified	Sun ONE Application Server 7 では、このフラグは実装されない
none	整合性検査は行われない

ejb-name

ejb-jar.xml ファイルで、コンテナ管理による持続性 Beans が関係するエンティティ Bean の名前を指定します。1 つ指定する必要があります。

サブ要素

なし

entity-mapping

データベースカラムへの Bean のマッピングを指定します。

サブ要素

次の表は、entity-mapping 要素のサブ要素を示します。

entity-mapping のサブ要素

サブ要素	必要指定数	説明
ejb-name	1 個のみ	ejb-jar.xml ファイルで、コンテナ管理による持続性 Beans が関係するエンティティ Bean の名前を指定する。1 つ指定する必要がある
table-name	1 個のみ	データベースの表の名前を指定する。この表はデータベーススキーマファイル内に存在している必要がある
cmp-field-mapping	1 個以上	1 つのフィールドを、マッピング先の 1 つまたは複数のカラムに関連付ける。Bean の主表のカラム、または定義された二次表のカラムのどちらでも構わない。フィールドを複数のカラムにマップした場合、データベースから値を取得するときは、最初のカラムが SOURCE として使用される。カラムは表示される順に更新される。EJB JAR ファイルに定義された cmp-field-mapping 要素ごとに 1 つの cmp-field 要素がある。 フィールドを読み取り専用として指定できる
cmr-field-mapping	0 または 1 個以上	コンテナ管理関係フィールドは、関係を定義する名前および 1 組以上のカラムペアを持つ。cmr-field ごとに 1 つの cmr-field-mapping 要素がある。関係をフェッチグループに入れることもできる
secondary-table	0 または 1 個以上	Bean の主表と二次表の関係を示す。この関係を示すには、カラムのペアを使用する
consistency	0 または 1 個	Bean 内のデータに関するトランザクションの整合性を保証する際のコンテナの動作を指定する。整合性検査フラグ要素が存在しない場合は、none とみなされる

fetched-with

フィールドおよび関係のフェッチグループ設定を指定します。この指定は省略可能です。

フィールドは、階層フェッチグループまたは独立フェッチグループに属することができます。fetched-with 要素が存在しない場合は、次の値が適用されます。<fetched-with><none/></fetched-with>.

サブ要素

次の表は、fetched-with 要素のサブ要素を示します。

fetched-with のサブ要素

サブ要素	必要指定数	説明
level	この表の要素のいずれか 1 つだけ	階層フェッチグループの名前を指定する。値は整数にする必要がある。等しい (または小さい) 値の階層フェッチグループに属するフィールドおよび関係が同時にフェッチされる。level の値は 0 よりも大きくする必要がある
named-group		独立フェッチグループの名前を指定する。指定したグループに属するすべてのフィールドおよび関係が同時にフェッチされる
none		整合性レベルのフラグで、このフィールドまたは関係が、それ自体によってフェッチされているかを示す

field-name

フィールドの Java 識別子を指定します。この識別子は、マップする cmp-field の field-name サブ要素の値と一致している必要があります。1 つ指定する必要があります。

サブ要素

なし

level

階層フェッチグループを指定します。この要素の値は整数にする必要があります。等しい (または小さい) 値の階層フェッチグループに属するフィールドおよび関係が同時にフェッチされます。level の値は 0 よりも大きくする必要があります。1 つだけ指定できます。

サブ要素

なし

lock-when-loaded

データの読み込み時にロックを実装するように指示する整合性レベルフラグです。

サブ要素

なし

lock-when-modified

Sun ONE Application Server 7 では、このフラグは実装されません。

サブ要素

なし

named-group

独立フェッチグループの名前を指定します。指定したグループに属するすべてのフィールドおよび関係が同時にフェッチされます。1 つだけ指定できます。

サブ要素

なし

none

このフィールドまたは関係がほかのフィールドまたは関係と一緒にフェッチされないように指示する、または `fetched-with` セマンティックを指定する整合性レベルフラグです。

サブ要素

なし

read-only

フィールドが読み取り専用であることを示すフラグです。

サブ要素

なし

schema

スキーマファイルへのパスを指定します。1 つだけ指定する必要があります。詳細は、217 ページの「スキーマの取り込み」を参照してください。

サブ要素

なし

secondary-table

Bean の二次表を指定します。

サブ要素

次の表は、secondary-table 要素のサブ要素を示します。

secondary table のサブ要素

サブ要素	必要指定数	説明
table-name	1 個のみ	データベースの表の名前を指定する。この表はデータベーススキーマファイル内に存在している必要がある
column-pair	1 個以上	2 つのデータベースの表内の関連カラムのペアの名前を指定する

sun-cmp-mapping

特定のスキーマにマップする Bean を指定します。

注 — 同じ EJB JAR ファイル内に配備する Beans 間であっても、Bean を別のスキーマにマップする Bean に関連付けることはできません。

サブ要素

次の表は、sun-cmp-mapping 要素のサブ要素を示します。

sun-cmp-mapping のサブ要素

サブ要素	必要指定数	説明
schema	1 個のみ	スキーマファイルへのパスを指定する
entity-mapping	1 個以上	データベースカラムへの Beans のマッピングを指定する

sun-cmp-mappings

EJB JAR コレクションにマップされるすべての Beans の、サブ要素のコレクションを指定します。

サブ要素

次の表は、sun-cmp-mappings 要素のサブ要素を示します。

sun-cmp-mappings のサブ要素

サブ要素	必要指定数	説明
sun-cmp-mapping	1 個以上	特定のスキーマにマップする Bean を指定する

table-name

データベースの表の名前を指定する。この表はデータベーススキーマファイル内に存在している必要がある。1 つ指定する必要がある

サブ要素

なし

例

この節では、次の例を紹介します。

- スキーマ定義の例
- CMP マッピング XML ファイルの例
- EJB QL クエリの例

スキーマ定義の例

```
CREATE TABLE Player
(
   player_Id VARCHAR(255) PRIMARY KEY,
   name VARCHAR(255) ,
   position VARCHAR(255) ,
   salary DOUBLE PRECISION NOT NULL ,
   picture BLOB,
);

CREATE TABLE League
(
   league_Id VARCHAR(255) PRIMARY KEY,
   name VARCHAR(255) ,
   sport VARCHAR(255) ,
);

CREATE TABLE Team
(
   team_Id VARCHAR(255) PRIMARY KEY,
   city VARCHAR(255) ,
   name VARCHAR(255) ,
   league_Id VARCHAR(255) ,
   FOREIGN KEY (league_Id)   REFERENCES League (league_Id) ,
);

CREATE TABLE TeamPlayer
(
   player_Id VARCHAR(255) ,
   team_Id VARCHAR(255),
   CONSTRAINT pk_TeamPlayer PRIMARY KEY (player_Id , team_Id) ,
   FOREIGN KEY (team_Id)   REFERENCES Team (team_Id),
   FOREIGN KEY (player_Id)   REFERENCES Player (player_Id) ,
);
```

CMP マッピング XML ファイルの例

これらの要素に関する詳細は、111 ページの「sun-cmp-mappings.xml ファイルの要素」を参照してください。

次のマッピングファイル例は、配備可能な EJB JAR ファイル内で META-INF/sun-cmp-mappings.xml という名前を持ちます。

```
<?xml version="1.0" encoding="UTF-8"?>
<sun-cmp-mappings>
   <sun-cmp-mapping>
       <schema>RosterSchema</schema>
       <entity-mapping>
          <ejb-name>League</ejb-name>
          <table-name>LEAGUE</table-name>
          <cmp-field-mapping>
             <field-name>name</field-name>
             <column-name>LEAGUE.NAME</column-name>
          </cmp-field-mapping>
          <cmp-field-mapping>
             <field-name>leagueId</field-name>
             <column-name>LEAGUE.LEAGUE_ID</column-name>
          </cmp-field-mapping>
          <cmp-field-mapping>
             <field-name>sport</field-name>
             <column-name>LEAGUE.SPORT</column-name>
          </cmp-field-mapping>
          <cmr-field-mapping>
             <cmr-field-name>team</cmr-field-name>
             <column-pair>
                 <column-name>LEAGUE.LEAGUE_ID</column-name>
                 <column-name>TEAM.LEAGUE_ID</column-name>
             </column-pair>
          </cmr-field-mapping>
       </entity-mapping>
       <entity-mapping>
          <ejb-name>Team</ejb-name>
          <table-name>TEAM</table-name>
          <cmp-field-mapping>
             <field-name>name</field-name>
             <column-name>TEAM.NAME</column-name>
          </cmp-field-mapping>
          <cmp-field-mapping>
             <field-name>city</field-name>
             <column-name>TEAM.CITY</column-name>
          </cmp-field-mapping>
          <cmp-field-mapping>
```

```
            <field-name>teamId</field-name>
            <column-name>TEAM.TEAM_ID</column-name>
        </cmp-field-mapping>
        <cmr-field-mapping>
            <cmr-field-name>playerId</cmr-field-name>
                <column-pair>
                <column-name>TEAM.TEAM_ID</column-name>
                <column-name>TEAMPLAYER.TEAM_ID</column-name>
            </column-pair>
            <column-pair>
                <column-name>TEAMPLAYER.PLAYER_ID</column-name>
                <column-name>PLAYER.PLAYER_ID</column-name>
            </column-pair>
            <fetched-with>
                <none/>
            </fetched-with>
        </cmr-field-mapping>
        <cmr-field-mapping>
            <cmr-field-name>leagueId</cmr-field-name>
            <column-pair>
                <column-name>TEAM.LEAGUE_ID</column-name>
                <column-name>LEAGUE.LEAGUE_ID</column-name>
            </column-pair>
            <fetched-with>
                <none/>
            </fetched-with>
        </cmr-field-mapping>
    </entity-mapping>

    <entity-mapping>
        <ejb-name>Player</ejb-name>
        <table-name>PLAYER</table-name>
        <cmp-field-mapping>
            <field-name>salary</field-name>
            <column-name>PLAYER.SALARY</column-name>
        </cmp-field-mapping>
        <cmp-field-mapping>
            <field-name>playerId</field-name>
            <column-name>PLAYER.PLAYER_ID</column-name>
        </cmp-field-mapping>
        <cmp-field-mapping>
            <field-name>position</field-name>
            <column-name>PLAYER.POSITION</column-name>
        </cmp-field-mapping>
        <cmp-field-mapping>
            <field-name>name</field-name>
            <column-name>PLAYER.NAME</column-name>
```

```
                </cmp-field-mapping>
                <cmr-field-mapping>
                   <cmr-field-name>teamId</cmr-field-name>
                   <column-pair>
                       <column-name>PLAYER.PLAYER_ID</column-name>
                       <column-name>TEAMPLAYER.PLAYER_ID</column-name>
                   </column-pair>
                   <column-pair>
                       <column-name>TEAMPLAYER.TEAM_ID</column-name>
                       <column-name>TEAM.TEAM_ID</column-name>
                   </column-pair>
                </cmr-field-mapping>
             </entity-mapping>
         </sun-cmp-mapping>
      </sun-cmp-mappings>
```

EJB QL クエリの例

```
<query>
   <description></description>
   <query-method>
      <method-name>findAll</method-name>
      <method-params />
   </query-method>
   <ejb-ql>select object(l) from League l</ejb-ql>
</query>

<query>
   <description></description>
   <query-method>
      <method-name>findByName</method-name>
      <method-params>
         <method-param>java.lang.String</method-param>
      </method-params>
   </query-method>
   <ejb-ql>select object(l) from League l where l.name = ?1</ejb-ql>
</query>

<query>
   <description></description>
   <query-method>
      <method-name>findByPosition</method-name>
      <method-params>
         <method-param>java.lang.String</method-param>
```

```
      </method-params>
   </query-method>
   <ejb-ql>select distinct object(p) from Player p where p.position = ?1</ejb-ql>
</query>

<query>
   <description>Selector returning SET</description>
   <query-method>
      <method-name>ejbSelectTeamsCity</method-name>
      <method-params>
         <method-param>team.LocalLeague</method-param>
      </method-params>
   </query-method>
   <ejb-ql>select distinct t.city from Team t where t.league = ?1</ejb-ql>
</query>

<query>
   <description>Selector returning single object LocalInterface</description>
   <query-method>
      <method-name>ejbSelectTeamByCity</method-name>
      <method-params>
         <method-param>java.lang.String</method-param>
      </method-params>
   </query-method>
   <result-type-mapping>Local</result-type-mapping>
   <ejb-ql>select distinct Object(t) from League l, in(l.teams) as t where t.city
= ?1</ejb-ql>
</query>

<query>
   <description>Selector returning single object String</description>
   <query-method>
      <method-name>ejbSelectTeamsNameByCity</method-name>
      <method-params>
         <method-param>java.lang.String</method-param>
      </method-params>
   </query-method>
   <ejb-ql>select distinct t.name from League l, in(l.teams) as t where t.city =
?1</ejb-ql>
</query>

<query>
   <description>Selector returning Set using multiple collection
declarations</description>
   <query-method>
      <method-name>ejbSelectPlayersByLeague</method-name>
      <method-params>
         <method-param>team.LocalLeague</method-param>
      </method-params>
```

```
   </query-method>
   <result-type-mapping>Local</result-type-mapping>
   <ejb-ql>select Object(p) from League l, in(l.teams) as t, in(t.players) p
where l = ?1</ejb-ql>
</query>

<query>
   <description>Selector single object int</description>
   <query-method>
      <method-name>ejbSelectSalaryOfPlayerInTeam</method-name>
      <method-params>
         <method-param>team.LocalTeam</method-param>
         <method-param>java.lang.String</method-param>
      </method-params>
   </query-method>
   <ejb-ql>select p.salary from Team t, in(t.players) as p where t = ?1 and p.name
= ?2</ejb-ql>
</query>

<query>
   <description>Finder using the IN Expression</description>
   <query-method>
      <method-name>findByPositionsGoalkeeperOrDefender</method-name>
      <method-params/>
   </query-method>
   <ejb-ql>select object(p) from Player p where p.position IN ('goalkeeper',
'defender')</ejb-ql>
</query>

<query>
   <description>Finder using the LIKE Expression</description>
   <query-method>
      <method-name>findByNameEndingWithON</method-name>
      <method-params/>
   </query-method>
   <ejb-ql>select object(p) from Player p where p.name LIKE '%on'</ejb-ql>
</query>

<query>
   <description>Finder using the IS NULL Expression</description>
   <query-method>
      <method-name>findByNullName</method-name>
      <method-params/>
   </query-method>
   <ejb-ql>select object(p) from Player p where p.name IS NULL</ejb-ql>
</query>
```

```
<query>
   <description>Finder using the MEMBER OF Expression</description>
   <query-method>
      <method-name>findByTeam</method-name>
      <method-params>
         <method-param>team.LocalTeam</method-param>
      </method-params>
   </query-method>
   <ejb-ql>select object(p) from Player p where ?1 MEMBER p.teams</ejb-ql>
</query>

<query>
   <description>Finder using the ABS function</description>
   <query-method>
      <method-name>findBySalarayWithArithmeticFunctionABS</method-name>
      <method-params>
         <method-param>double</method-param>
      </method-params>
   </query-method>
   <ejb-ql>select object(p) from Player p where p.salary = ABS(?1)</ejb-ql>
</query>

<query>
   <description>Finder using the SQRT function</description>
   <query-method>
      <method-name>findBySalarayWithArithmeticFunctionSQRT</method-name>
      <method-params>
         <method-param>double</method-param>
      </method-params>
   </query-method>
   <ejb-ql>select object(p) from Player p where p.salary = SQRT(?1)</ejb-ql>
</query>
```

第 5 章

メッセージ駆動型 Beans の使用

この章では、メッセージ駆動型 Beans について、および Sun ONE Application Server 7 環境で作成する際の要件について説明します。

注 メッセージ駆動型 Beans または EJB テクノロジに精通していない場合は、Java Software チュートリアルを参照してください。

```
http://java.sun.com/j2ee/docs.html
```

メッセージ駆動型 Beans に関する詳細情報は、『Enterprise JavaBeans Specification, v2.0』の第 15 章と第 16 章にあります。

Sun ONE Application Server の概要は、19 ページの「Sun ONE Application Server Enterprise JavaBeans の紹介」および『Sun ONE Application Server 製品の概要』にあります。

この章には次の項目があります。

- メッセージ駆動型 Beans について
- メッセージ駆動型 Beans の開発
- 制限事項と最適化
- メッセージ駆動型 Beans XML ファイルの例

メッセージ駆動型 Beans について

メッセージ駆動型 Bean は、J2EE アプリケーションでメッセージを非同期的に処理できる Enterprise JavaBeans (EJB) です。この Bean はメッセージリスナの働きをします。イベントリスナと似ていますが、イベントの代わりにメッセージを受け取ります。アプリケーションクライアント、別の Enterprise JavaBean、Web コンポーネントなどの J2EE コンポーネントから送信されるメッセージや、J2EE テクノロジを利用しないアプリケーションやシステムから送信されるメッセージを処理できます。

この節では次の項目について説明します。

- メッセージ駆動型 Beans の相違点
- メッセージ駆動型 Beans の特性
- トランザクション管理
- メッセージの同時処理

メッセージ駆動型 Beans の相違点

セッション Beans およびエンティティ Beans では、JMS メッセージを同期的に送受信できますが、非同期の送受信はできません。サーバーリソースの停滞を避けるために、サーバー側のコンポーネントで非同期受信を使用するとよいでしょう。非同期にメッセージを受信するには、メッセージ駆動型 Bean を使用します。

メッセージ駆動型 Beans とセッションおよびエンティティ Beans のもっとも大きな目に見える違いは、クライアントがインタフェースを通じてメッセージ駆動型 Beans にアクセスしないことです。セッション Bean やエンティティ Bean とは異なり、メッセージ駆動型 Bean には 1 つの Bean クラスしかありません。

メッセージ駆動型 Bean は、次のような点で、ステートレスセッション Bean に似ています。

- メッセージ駆動型 Bean のインスタンスは、特定のクライアントのデータや会話型状態を保持しない
- メッセージ駆動型 Bean のすべてのインスタンスは同等であり、コンテナはそれらのメッセージ駆動型 Bean インスタンスをプールできる。このため、メッセージのストリームを同時に処理できる
- 1 つのメッセージ駆動型 Bean が複数のクライアントからのメッセージを処理できる。

メッセージ駆動型 Bean インスタンスのインスタンス変数は、JMS 接続、オープンデータベース接続、EJB オブジェクトへのオブジェクト参照などの、クライアントメッセージの処理に関連する状態のいくつかを保持できます。

メッセージ駆動型 Beans の特性

メッセージ駆動型 Bean インスタンスは、メッセージ駆動型 Bean クラスのインスタンスです。ホームインタフェースもリモートインタフェースもなく、メッセージ駆動型 Bean は匿名です。つまり、クライアントに見える識別情報を持ちません。

クライアントは、メッセージ駆動型 Bean クラスが `MessageListener` であるメッセージの送信先にメッセージを送信することによって、JMS からメッセージ駆動型 Bean にアクセスします。メッセージ駆動型 Bean のキューとトピックは、配備時に Sun ONE Application Server リソースを使って割り当てられます。

メッセージ駆動型 Beans には次の特性があります。

- 1 つのクライアントメッセージを受信すると実行される
- 非同期的に呼び出される
- 比較的短命である
- データベース内の共有データを直接表さないが、このデータのアクセスおよび更新は可能である
- トランザクションを認識できる
- 状態を持たない

トランザクション管理

『Enterprise JavaBeans Specification, v2.0』に定義されたコンテナ管理トランザクションと Bean 管理トランザクションの両方がサポートされています。

コンテナ管理トランザクションでは、トランザクションコンテキスト内でメッセージ駆動型 Bean にメッセージを配信することもできます。その場合、`onMessage` メソッド内のすべてのオペレーションが 1 つのトランザクションに含まれます。メッセージ処理がロールバックされると、メッセージは再度配信されます。

トランザクションについては、143 ページの「Enterprise JavaBeans のトランザクション処理」を参照してください。

メッセージの同時処理

コンテナでは、メッセージ駆動型 Bean クラスの複数のクラスを同時に実行できるので、メッセージのストリームの同時処理が可能です。コンテナは、メッセージ処理の同時性を損なわないように時間順にメッセージを配信しようとしますが、メッセージ駆動型 Bean クラスのインスタンスにメッセージが配信される正確な順序は保証されません。

そのため、メッセージ駆動型 Beans は、順序が乱れたメッセージを処理できるようにしておきます。たとえば、予約を入れるメッセージの前に、予約を取り消すメッセージが配信されることがあります。

メッセージ駆動型 Beans の開発

メッセージ駆動型 Bean モデルを利用すると、非同期に呼び出されて受信メッセージを処理する Enterprise JavaBean を、ほかの JMS リスナーで同じ機能を開発するのと同じくらい簡単に開発できます。さらに、コンテナが提供するメッセージ駆動型 Bean インスタンスのプール機能によって、メッセージのストリームを同時処理することもできます。

次の各項で、メッセージ駆動型 Beans を作成する際のガイドラインを示します。

- Bean クラス定義の作成
- 設定

Bean クラス定義の作成

セッション Beans およびエンティティ Beans とは異なり、メッセージ駆動型 Beans には、クライアントアクセスを定義するリモートインタフェースやローカルインタフェースがありません。クライアントコンポーネントは、メッセージ駆動型 Beans を探して、そのメソッドを直接起動することはできません。

メッセージ駆動型 Beans はビジネスメソッドを持ちませんが、`onMessage` メソッドによって内部的に呼び出されるヘルパーメソッドを含めることができます。

メッセージ駆動型 Beans のクラスは、次の必要条件を満たす必要があります。

- 直接または間接的に、`javax.ejb.MessageDrivenBean` インタフェースを実装する必要がある
- 直接または間接的に、`javax.ejb.MessageListener` インタフェースを実装する必要がある

- public として定義する必要があり、abstract または final として定義してはならない
- 引数をとらないパブリックコンストラクタ (メッセージ駆動型 Bean クラスのインスタンスを作成するためにコンテナが使用する) を持つ必要がある
- `finalize` メソッドを定義してはならない
- `onMessage` メソッドを実装する必要がある
- 引数なしの `ejbCreate` メソッドを 1 つ実装する必要がある
- 引数なしの `ejbRemove` メソッドを 1 つ実装する必要がある

次の各項では、メッセージ駆動型 Bean のクラス定義内のさまざまなメソッドについて説明します。

- ejbCreate の使用
- setMessageDrivenContext の使用
- onMessage の使用
- ejbRemove の使用

ejbCreate の使用

メッセージ駆動型 Bean クラスでは、`ejbCreate` メソッドを 1 つ定義する必要があり、そのシグネチャは次のルールに従っている必要があります。

- メソッド名は `ejbCreate` にする必要がある
- `public` として定義する必要があり、`final` または `static` として定義してはならない
- 戻り値タイプは `void` にする必要がある
- 引数をとってはならない
- `throws` 句でアプリケーション例外を定義してはならない

setMessageDrivenContext の使用

コンテナは、メッセージ駆動型 Beans インスタンスに `MesssageDrivenContext` を提供します。これにより、メッセージ駆動型 Bean インスタンスは、コンテナが維持するそのインスタンスのコンテキストにアクセスできます。

onMessage の使用

`onMessage` メソッドは、1 つの引数 (受信メッセージ) をとります。

onMessage メソッドは、Bean が処理するメッセージが到着したときに、Bean のコンテナによって呼び出されます。このメソッドには、メッセージを処理するビジネスロジックが含まれています。メッセージを解析して必要なビジネスロジックを実行するのが、メッセージ駆動型 Bean の役割です。

メッセージ駆動型 Bean クラスでは、onMessage メソッドを 1 つ定義する必要があり、そのシグネチャは次のルールに従っている必要があります。

- public として定義する必要があり、final または static として定義してはならない
- 戻り値タイプは void にする必要がある
- タイプ javax.jms.Message の引数を 1 つだけとる必要がある
- throws 句でアプリケーション例外を定義してはならない。onMessage から例外をスローするためのセマンティックについては、139 ページの「onMessage のランタイム例外」を参照

onMessage メソッドは、配備記述子内に指定されたトランザクション属性によって決定されるトランザクションの範囲内で呼び出されます。

注　コンテナ管理トランザクションの境界設定を使用するように指定されている Bean では、トランザクション属性の Required または NotSupport のどちらかを配備記述子に指定する必要があります。

---

ejbRemove の使用

メッセージ駆動型 Bean クラスでは、Bean が不要になったときに解放するための ejbRemove メソッドを 1 つ定義します。そのシグネチャは次の規則に従っている必要があります。

- メソッド名は ejbRemove にする必要がある
- public として定義する必要があり、final または static として定義してはならない
- 戻り値タイプは void にする必要がある
- 引数をとってはならない
- throws 句でアプリケーション例外を定義してはならない

注　コンテナが必ずメッセージ駆動型 Bean インスタンスで ejbRemove を呼び出すと想定することはできません。

EJB コンテナがクラッシュしている場合、またはインスタンスの onMessage メソッドからコンテナに例外がスローされた場合は、ejbRemove メソッドは呼び出されません。メッセージ駆動型 Bean のインスタンスが ejbCreate メソッドまたは onMessage メソッド、あるいはその両方でリソースを割り当て、ejbRemove メソッドでリソースを解放する場合、それらのリソースは自動的に解放されません。そのため、アプリケーションで、未解放のリソースを定期的にクリーンアップするメカニズムを提供する必要があります。

設定

ここでは、設定に関する次の項目について説明します。

- 接続ファクトリと送信先
- メッセージ駆動型 Bean プール
- サーバーインスタンス全体に適用される属性
- JMS プロバイダへの自動再接続

接続ファクトリと送信先

メッセージ駆動型 Bean は JMS クライアントです。したがって、メッセージ駆動型 Bean のコンテナは、Sun ONE Application Server に統合された JMS サービスを使用します。JMS クライアントは、JMS 接続ファクトリと送信先が管理するオブジェクトを使用します。JMS 接続ファクトリが管理するオブジェクトは、JMS プロバイダへの接続を作成するためのリソースマネージャの接続ファクトリオブジェクトです。

メッセージ駆動型 Bean の sun-ejb-jar.xml 内の mdb-connection-factory 要素を使用して、コンテナが JMS プロバイダへのコンテナ接続を作成するときに使用する接続ファクトリを指定できます。この要素は、サードパーティの JMS プロバイダでも利用できます。

mdb-connection-factory が指定されていない場合、サーバーの起動時に作成されたデフォルトの接続ファクトリが使用されます。これにより、内蔵の Sun ONE Message Queue のデフォルトユーザー名とパスワード (リソースプリンシパル) を使って、server.xml ファイルの jms-service 要素に指定されているポート (指定が有効化されている場合) の Sun ONE Message Queue ブローカに接続できます。詳細は、『Sun ONE Message Queue 開発者ガイド』を参照してください。

sun-ejb-jar.xml ファイル内の ejb 要素の jndi-name 要素は、メッセージ駆動型 Bean と関連付けられた JMS のキューまたはトピックの送信先として、管理オブジェクトの JNDI 名を指定します。

メッセージ駆動型 Bean プール

コンテナは、一連のメッセージを同時処理できるように、メッセージ駆動型 Bean のプールを管理します。Sun ONE Application Server 固有 Bean の配備記述子には、プールを定義する要素 (`bean-pool` 要素) が含まれます。

- `steady-pool-size`
- `resize-quantity`
- `max-pool-size`
- `pool-idle-timeout-in-seconds`

これらの要素に関する詳細は、202 ページの「プールとキャッシュの要素」を参照してください。

サーバーインスタンス全体に適用される属性

管理者は、`server.xml` ファイル内の `mdb-container` 要素で、サーバーインスタンス全体のメッセージ駆動型 Bean に関する次の属性を制御できます。

- `steady-pool-size`
- `pool-resize-quantity`
- `max-pool-size`
- `idle-timeout-in-seconds`
- `log-level`
- `monitoring-enabled`

これらの属性に関する詳細は、202 ページの「プールとキャッシュの要素」および『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

メッセージ駆動型 Beans の監視については、Sun ONE Application Server 管理インタフェース のオンラインヘルプ、および『管理者ガイド』を参照してください。

注 必要な時以外に監視を行うと、パフォーマンスに影響することがあります。使用しない場合は asadmin コマンドまたは管理インタフェースを使って監視をオフにできます。

JMS プロバイダへの自動再接続

Sun ONE Application Server を起動すると、配備したメッセージ駆動型 Bean ごとにコンテナが JMS プロバイダへの接続を維持します。接続が断たれると、コンテナは JMS プロバイダからのメッセージを受信できなくなるため、メッセージ駆動型 Bean インスタンスにメッセージを配信できなくなります。自動再接続機能を有効にしておくと、接続が断たれた場合にコンテナが JMS プロバイダへの再接続を自動的に試みます。

`server.xml` ファイルの `mdb-container` 要素には、自動再接続のプロパティが含まれます。デフォルトでは、`reconnect-enabled` は true に設定され、`reconnect-delay-in-seconds` は 60 秒に設定されます。つまり、再接続は 60 秒おきに試みられます。`reconnect-max-retries` は 60 回に設定されます。

コンテナは、再接続を試みるたびにメッセージを記録します。

注 メッセージ処理の段階に応じて、接続が断たれた場合に `onMessage` メソッドが正常に完了されなかったり、JMS 例外によってトランザクションがロールバックされたりすることがあります。コンテナが JMS プロバイダへの再接続を確立すると、JMS メッセージ再配信セマンティックが適用されます。

`server.xml` ファイルの `mdb-container` 要素に指定する自動再接続のプロパティについては、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

制限事項と最適化

この節では、メッセージ駆動型 Beans を開発する際に注意する制限事項およびパフォーマンスの最適化について説明します。

- JMS に関する制限事項
- プールの調整と監視
- onMessage のランタイム例外

JMS に関する制限事項

Sun ONE Application Server では、Sun ONE Message Queue 3.0.1 Platform Edition が提供する組み込みの JMS サービスによる JMS メッセージングをサポートしています。スタンドアロン製品としての Sun ONE Message Queue 3.0.1 では、JMS 1.1 仕様をサポートしています。ただし、Sun ONE Application Server 7 では、より限定的な JMS 1.02b 仕様のみを範囲とする J2EE 1.3 仕様をサポートしています。そのため、JMS 1.1 に組み入れられた追加機能は、Sun ONE Application Server 7 で実行するアプリケーションには使用できません。

したがって、JMS メッセージングアプリケーションの開発では、Sun ONE Application Server 環境で実行する JMS クライアントコンポーネントを JMS 1.02b に限定する必要があります。詳細は、『Sun ONE Message Queue 開発者ガイド』または『リリースノート』を参照してください。

プールの調整と監視

メッセージ駆動型 Bean のプールはスレッドのプールでもあり、プール内の各メッセージ駆動型 Bean インスタンスはサーバーセッションに関連付けられ、各サーバーセッションはスレッドに関連付けられます。このため、容量の大きなプールではスレッド数も多く、パフォーマンスとサーバーリソースに影響します。

メッセージ駆動型 Bean プールのプロパティを設定するときは、メッセージの受信頻度と受信パターン、`onMessage` メソッドの処理時間、サーバーリソース全体 (スレッド、メモリなど)、およびメッセージ駆動型 Bean がアクセスするその他リソースに起因する同時処理の要件と制限を考慮する必要があります。

パフォーマンスとリソース使用率の調整では、コンテナが使用する接続ファクトリ (配備記述子の `mdb-connection-factory` 要素) の潜在的な JMS プロバイダプロパティについても注意する必要があります。たとえば、接続ファクトリの Sun ONE Message Queue のフロー制御に関するプロパティは、メッセージの受信頻度が `max-pool-size` が対応する値を大きく上回る状況を考慮して調整する必要があります。

メッセージ駆動型 Bean プールの統計情報を入手する方法については、『Sun ONE Application Server 管理者ガイド』を参照してください。

onMessage のランタイム例外

正常に機能しているその他の JMS MessageListeners と同様、一般に、メッセージ駆動型 Bean にランタイム例外をスローさせることはできません。メッセージ駆動型 Bean の `onMessage` メソッドがシステムレベルの例外やエラーに遭遇し、メソッドが正常に完了しない場合、『Enterprise JavaBeans Specification, v2.0』には次のようなガイドラインが示されています。

- Bean メソッドがランタイム例外またはエラーに遭遇した場合は、Bean メソッドからコンテナに単にエラーを伝達する
- Bean メソッドの動作によって、Bean メソッドが復旧できないチェック済みの例外が発生した場合は、Bean メソッドは元の例外をラップする `javax.ejb.EJBException` をスローする
- 上記以外の予期せぬエラー状況については、`javax.ejb.EJBException` を使って報告する (`javax.ejb.EJBException` は `java.lang.RuntimeException` のサブクラス)

コンテナ管理トランザクションの境界設定では、メッセージ駆動型 Bean の `onMessage` メソッドからランタイム例外を受け取ると、コンテナは、コンテナが開始したトランザクションをロールバックし、JMS メッセージが再配信されます。これは、メッセージ配信自体がコンテナが開始したトランザクションの一部であるためです。デフォルトでは、メッセージ駆動型 Bean インスタンスの `onMessage` メソッドから最初のランタイム例外を受け取ると、Sun ONE Application Server コンテナは JMS プロバイダへのコンテナの接続を閉じます。これにより、メッセージ駆動型 Bean の `onMessage` メソッドに異常が続いても、メッセージ再配信のループを回避でき、サーバーリソースを保護できます。このコンテナのデフォルト設定は、`server.xml` ファイル内の `mdb-container` 要素の `cmt-max-runtime-exceptions` プロパティを使って変更できます。

cmt-max-runtime-exceptions プロパティは、コンテナの JMS プロバイダへの接続を終了し始めるまでに、コンテナがメッセージ駆動型 Bean の onMessage メソッドから何回のランタイム例外を受け取るか、その最大数を指定します。デフォルト値は 1 です。-1 を指定すると、このコンテナ保護を無効化します。

メッセージ駆動型 Bean の onMessage メソッドは、javax.jms.Message getJMSRedelivered メソッドを使うことで、受信したメッセージが再配信されたメッセージであるかどうかを調べることができます。

注 cmt-max-runtime-exceptions プロパティは、将来廃止される可能性があります。

メッセージ駆動型 Beans XML ファイルの例

この節では、次のファイルの例を示します。

- ejb-jar.xml ファイルの例
- sun-ejb-jar.xml ファイルの例

メッセージ駆動型 Bean に関連する要素については、176 ページの「sun-ejb-jar.xml ファイルの要素」および『Sun ONE Application Server 開発者ガイド』を参照してください。

ejb-jar.xml ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
   <enterprise-beans>
      <message-driven>
         <ejb-name>MessageBean</ejb-name>
         <ejb-class>samples.mdb.ejb.MessageBean</ejb-class>
         <transaction-type>Container</transaction-type>
         <message-driven-destination>
            <destination-type>javax.jms.Queue</destination-type>
         </message-driven-destination>
         <resource-ref>
            <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
            <res-type>javax.jms.QueueConnectionFactory</res-type>
            <res-auth>Container</res-auth>
```

```
         </resource-ref>
      </message-driven>
   </enterprise-beans>
      <assembly-descriptor>
         <container-transaction>
            <method>
               <ejb-name>MessageBean</ejb-name>
               <method-intf>Bean</method-intf>
               <method-name>onMessage</method-name>
               <method-params>
                  <method-param>javax.jms.Message</method-param>
               </method-params>
            </method>
         <trans-attribute>NotSupported</trans-attribute>
      </container-transaction>
   </assembly-descriptor
</ejb-jar>
```

sun-ejb-jar.xml ファイルの例

これらの要素の詳細については、176 ページの「sun-ejb-jar.xml ファイルの要素」を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sun-ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Sun ONE Application
Server 7.0 EJB 2.0//EN'
'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-0.dtd'>

<sun-ejb-jar>
   <enterprise-beans>
      <ejb>
         <ejb-name>MessageBean</ejb-name>
         <jndi-name>jms/sample/Queue</jndi-name>
         <resource-ref>
            <res-ref-name>jms/QueueConnectionFactory</res-ref-name>
            <jndi-name>jms/sample/QueueConnectionFactory</jndi-name>
            <default-resource-principal>
               <name>guest</name>
               <password>guest</password>
            </default-resource-principal>
         </resource-ref>
         <mdb-connection-factory>
            <jndi-name>jms/sample/QueueConnectionFactory</jndi-name>
            <default-resource-principal>
               <name>guest</name>
               <password>guest</password>
```

```
            </default-resource-principal>
         </mdb-connection-factory>
      </ejb>
   </enterprise-beans>
</sun-ejb-jar>
```

第 6 章

Enterprise JavaBeans のトランザクション処理

この章では、Sun ONE Application Server 7 の Enterprise JavaBeans (EJBs) プログラミングモデルに組み込まれたトランザクションサポートについて説明します。

注　EJB テクノロジのトランザクション処理に精通していない場合は、Java Software チュートリアルを参照してください。

`http://java.sun.com/j2ee/docs.html`

EJB トランザクションサポートに関する詳細情報は、『Enterprise JavaBeans Specification, v2.0』の第 17 章にあります。

Sun ONE Application Server の概要は、19 ページの「Sun ONE Application Server Enterprise JavaBeans の紹介」および『Sun ONE Application Server 製品の概要』にあります。

この章には次の項目があります。

- JTA トランザクションと JTS トランザクションのサポート
- コンテナ管理トランザクションの使用法
- Bean 管理トランザクションの使用法
- トランザクションタイムアウトの設定
- 遮断レベルの処理

JTA トランザクションと JTS トランザクションのサポート

J2EE では、次の 2 つの仕様により分散トランザクションをサポートしています。

- Java Transaction API (JTA)
- Java Transaction Service (JTS)

JTA は、アプリケーションおよびアプリケーションサーバがトランザクションにアクセスできるようにする、実装に依存しない高レベルなプロトコル API です。

JTS は、JTA をサポートするトランザクションマネージャの実装を指定して、API の下のレベルで OMG Object Transaction Service (OTS) 1.1 仕様の Java マッピングを実装します。JTS は、Internet Inter-ORB Protocol (IIOP) を使用してトランザクションを伝達します。

トランザクションマネージャの現在の実装は、JTS と JTA をサポートしています。EJB コンテナは、Java Transaction API インタフェースを使用して JTS と対話します。

J2EE トランザクションマネージャは、Bean 管理 JBDC (Java Database Connectivity) トランザクションを除くすべての EJB トランザクションを制御し、Enterprise JavaBean がトランザクション内で複数のデータベースを更新できるようにします。

トランザクション処理について

開発者は、複数のサイトに分散された複数のデータベースのデータを更新するアプリケーションを作成できます。サイトでは、異なるベンダーの EJB サーバを使用してもかまいません。

この節では、次の各項目に関する概要を示します。

- 単層型トランザクション
- グローバルトランザクションとローカルトランザクション
- 境界設定モデル
- コミットオプション
- 管理と監視

単層型トランザクション

『Enterprise JavaBeans Specification, v2.0』では、ネストとは対照的な単層型トランザクションのサポートを要求しています。単層型トランザクションでは、各トランザクションはシステムのほかのトランザクションから独立しており、依存していません。現在のトランザクションが終了するまでは、同じスレッド内で別のトランザクションを開始することはできません。

単層型トランザクションは、もっとも普及したモデルであり、ほとんどの商用データベースでサポートされています。ネストされたトランザクションでは、より精細なトランザクションの制御が可能ですが、これをサポートする商用データベースシステムはごく少数しかありません。

グローバルトランザクションとローカルトランザクション

Sun ONE Application Server でのトランザクションのサポートを理解するためには、グローバルトランザクションとローカルトランザクションの違いを理解することが重要です。

- グローバルトランザクション - リソースマネージャによって管理および調整され、複数のデータベースやプロセスにまたがることができるトランザクション。リソースマネージャは通常、XA プロトコルを使って Enterprise Information System (EIS) またはデータベースと対話する
- ローカルトランザクション - 単一の EIS またはデータベースに固有であり、1 つのプロセス内に制限されるトランザクション。ローカルトランザクションが複数のデータソースを扱うことはない

ローカルトランザクションとグローバルトランザクションはどちらも、`javax.transaction.UserTransaction` インタフェースによって境界が設定されるので、クライアントはこのインタフェースを使う必要があります。ローカルトランザクションはトランザクションマネージャをバイパスし、より高速です。

最初は、すべてのトランザクションがローカルトランザクションです。XA 以外のデータソースコネクションがトランザクションの範囲に指定された最初のリソースコネクションである場合、2 番目に指定された XA データソースコネクションが加わったときに、グローバルトランザクションになります。2 番目にも XA 以外のデータソースコネクションが加わろうとすると、例外がスローされます。

Sun ONE Application Server は、グローバルかローカルのどちらかのトランザクションモードで動作し、両方のモードを混在させることはできません。

注　アプリケーションでグローバルトランザクションを使うには、対応する Sun ONE Application Server のリソースマネージャを設定して使用可能にする必要があります。詳細は、Sun ONE Application Server の Administration interface のオンラインヘルプ、および『管理者ガイド』を参照してください。

境界設定モデル

開発者は、EJB コード内のプログラムによるトランザクションの境界設定 (Bean 管理) か、または宣言による境界設定 (コンテナ管理) のどちらかを使用できます。Enterprise JavaBean で Bean 管理とコンテナ管理のどちらのトランザクション境界設定を使用するかに関係なく、EJB コンテナおよび Sun ONE Application Server にトランザクション管理の負荷がかかります。このコンテナとサーバは、必要な低レベルのトランザクションプロトコル (トランザクションマネージャと、dustbowls システムまたは Sun ONE Message Queue プロバイダの間の 2 階層コミットプロトコルなど)、トランザクションコンテキストの伝達、および分散型の 2 階層コミットを実装します。

次の項で、これらの境界設定モデルについて説明します。

- コンテナ管理トランザクション
- Bean 管理トランザクション

コンテナ管理トランザクション

Enterprise JavaBean の主な利点の 1 つは、コンテナ管理トランザクション (宣言型トランザクションとも呼ばれる) に提供するサポートです。コンテナ管理トランザクションを使用する Enterprise JavaBean では、EJB コンテナがトランザクションの境界を設定します。

注　すべての種類の Enterprise JavaBean (セッション、エンティティ、メッセージ駆動型) でコンテナ管理トランザクションを使用できますが、エンティティ Bean で使用できるのはコンテナ管理トランザクションだけです。

コンテナ管理トランザクションを使用すると、EJB コードでトランザクションの境界を明示的に設定しないので、開発作業が簡素化されます。つまり、コードにはトランザクションを開始および終了するステートメントを記述しません。コンテナは、次の処理を実行します。

- トランザクションコンテキストの境界を設定し、透過的に伝達する

- トランザクションマネージャと連係して、トランザクション内のすべての関係要素に一貫した結論を参照させる

Bean 管理トランザクション

EJB の仕様では、`javax.transaction.UserTransaction` を使用した、Bean 管理によるトランザクションの境界設定 (プログラマによる境界設定トランザクションとも呼ばれる) をサポートしています。Bean 管理トランザクションでは、JNDI (Java Naming and Directory Interface) 検索を実行して `UserTransaction` オブジェクトを取得する必要があります。

注　　Bean 管理トランザクションはセッション Bean またはメッセージ駆動型 Bean で使用できますが、エンティティ Bean ではコンテナ管理トランザクションを使用する必要があります。

Bean 管理トランザクションには次の 2 つのタイプがあります。

- JDBC タイプ - 接続インタフェースの commit および rollback メソッドで、JDBC トランザクションの境界を定める
- JTA タイプ - UserTransaction インタフェースの begin、commit、および rollback メソッドを呼び出して、JTA トランザクションの境界を定める

コミットオプション

EJB プロトコルは、トランザクションがコミットされたときに、インスタンスの状態を廃棄できる柔軟性をコンテナに提供するように設計されています。これにより、コンテナは、エンティティオブジェクトの状態のキャッシュや、エンティティオブジェクト ID と EJB インスタンスの関連付けを最適に管理できます。

コミット時のオプションには次の 3 つがあります。

- オプション A - コンテナは、トランザクション間で動作可能なインスタンスをキャッシュする。コンテナは、そのインスタンスが持続ストレージ内のオブジェクトの状態に排他的にアクセスできるようにする

  この場合、コンテナは、次のトランザクションの開始時に、持続ストレージからインスタンスの状態の同期をとる必要はありません。

注　　コミットオプション A は、Sun ONE Application Server 7 ではサポートされていません。

- オプション B - コンテナは、トランザクション間で動作可能なインスタンスをキャッシュするが、そのインスタンスが持続ストレージ内のオブジェクトの状態に排他的にアクセスできるようにはしない。これがデフォルト

  この場合、コンテナは次のトランザクションの開始時に、持続ストレージから ejbLoad を呼び出してインスタンスの状態と同期をとる必要がある

- オプション C - コンテナは、トランザクション間で動作可能なインスタンスをキャッシュしないが、代わりに、トランザクションの完了後に、そのインスタンスを使用可能なインスタンスのプールに戻す

  コミットオプション C における各ビジネスメソッド起動のライフサイクルは、次のようになる

```
ejbActivate->
   ejbLoad ->
      ビジネスメソッド ->
         ejbStore ->
            ejbPassivate
```

  同じエンティティ EJBObject に同時にアクセスしているトランザクションクライアントが 2 つ以上ある場合、最初のクライアントが動作可能なインスタンスを取得し、次の同時アクセスのクライアントがプールから新規インスタンスを取得する。

Sun ONE Application Server の配備記述子には、使用するコミットオプションを指定する commit-option 要素があります。指定したコミットオプションに基づいて、適切なハンドラがインスタンス化されます。

注　　コミットオプション A を使用する場合は、開発者が確実にこのアプリケーションだけがデータベースを更新するようにする必要があります。つまり、これはコンテナの仕事ではありません。

管理と監視

管理者は、server.xml ファイル内の transaction-service 要素で、サーバインスタンス全体のトランザクションサービスに関する次の属性を制御できます。

- automatic-recovery
- timeout-in-seconds
- tx-log-directory
- heuristic-decision
- keypoint-interval
- log-level
- monitoring-enabled

これらの属性に関する詳細は、『Sun ONE Application Server 管理者用設定ファイルリファレンス』を参照してください。

さらに、管理者は、トランザクションマネージャからの統計情報を使ってトランザクションを監視できます。トランザクションマネージャは、サーバが起動してから完了したトランザクション数、ロールバックされたトランザクション数、復旧されたトランザクション数、および現在処理中のトランザクション数など、稼動状況に関する情報を提供します。

トランザクションの管理および監視については、Sun ONE Application Server の Administration interface のオンラインヘルプ、および『Sun ONE Application Server 管理者ガイド』を参照してください。

コンテナ管理トランザクションの使用法

一般に、コンテナは、EJB メソッド起動の直前にトランザクションを開始し、メソッド終了の直前にそのトランザクションをコミットします。各メソッドを、1 つのトランザクションに関連付けることができます。

注 ネストされたトランザクションまたは複数のトランザクションを 1 つのメソッド内に含めることはできません。

コンテナ管理トランザクションでは、すべてのメソッドをトランザクションに関連付ける必要はありません。Enterprise JavaBean の配備時に、トランザクション属性を設定することによって、Bean のどのメソッドをトランザクションと関連付けるかを指定することができます。

コンテナ管理トランザクションを使用する Bean では、コーディングが少なくてすみますが、次の制限があります。

> メソッドの実行中は、1 つのトランザクションに関連付けるか、トランザクションには一切関連付けないかのどちらかしかありません。

この制限により Bean のコーディングが困難になる場合は、Bean 管理トランザクションを選択することをお勧めします。

コミットが発生すると、トランザクションは、Bean の有効な作業が終了したことをコンテナに伝え、基礎となるデータソースと状態の同期をとるようにコンテナに指示します。コンテナはトランザクションの終了を許可し、Bean を解放します。コミットされたトランザクションに関連付けられたリザルトセットは無効になります。同じ Bean に対する連続したリクエストによって、コンテナは、基礎となるデータソースとの同期負荷 (load-to-synchronize) 状態を発行します。

注　　コンテナが開始したトランザクションは暗黙的にコミットされます。

トランザクションに関連した Bean であれば、トランザクションをロールバックできます。

次の各項では、コンテナ管理トランザクションを使用する Enterprise JavaBeans の開発について説明します。

- トランザクション属性の指定
- コンテナ管理トランザクションのロールバック
- セッション Beans のインスタンス変数の同期化
- コンテナ管理トランザクションで使用できないメソッド

トランザクション属性の指定

トランザクション属性は、トランザクションの範囲を制御するパラメータです。

トランザクション属性は配備記述子内に保存されるので、EJB 作成時、アセンブリ (パッケージ化) 時、配備時など、J2EE アプリケーション開発中のいくつかの段階で変更が可能です。ただし、EJB 開発者は、EJB の作成時にトランザクション属性を指定する必要があります。開発者 (またはアセンブリ担当者) がコンポーネントを大規模なアプリケーションにアセンブリするときだけに、この属性を変更する必要があります。

注　　J2EE アプリケーションの配備担当者がトランザクション属性を指定するという想定はしないでください。

Enterprise JavaBean 全体、または個々のメソッドのトランザクション属性を指定できます。メソッドと Bean に別々の属性を指定した場合は、メソッドの属性が優先されます。

ヒント　　EJB の配備記述子内にトランザクションを設定する方法がわからない場合は、コンテナ管理トランザクションを指定します。次に、Enterprise JavaBean 全体に対して `Required` トランザクション属性を設定します。ほとんどの場合、この方法で正しく動作します。

EJB 配備記述子ファイルの詳細は、170 ページの「配備記述子の作成」を参照してください。

ここには次の項目があります。

- 属性の必要条件の区別
- 属性値

属性の必要条件の区別

個々のメソッドの属性を指定する場合は、Bean のタイプによって必要条件が異なります。

- セッション Beans - ビジネスメソッドの属性を定義する必要がありますが、create メソッドの属性は指定できない
- エンティティ Beans - ビジネスメソッド、create メソッド、remove メソッド、および検索メソッドのトランザクション属性が必要となる
- メッセージ駆動型 Beans - `onMessage` メソッドのトランザクション属性 (`Required` か `NotSupported` のどちらか) が必要となる

属性値

トランザクション属性には、次の値のいずれかを設定できます。

- Required
- RequiresNew
- Mandatory
- NotSupported
- Supports
- Never

Required

クライアントがトランザクション内で動作中に Enterprise JavaBean のメソッドを呼び出した場合、そのメソッドはクライアントのトランザクション内で動作します。クライアントがトランザクションに関連付けられていない場合、コンテナは、新しいトランザクションを開始してからメソッドを実行します。

ヒント　`Required` 属性は、ほとんどのトランザクションで使用できます。したがって、デフォルト値として、少なくとも開発の初期段階ではこの値を使用するとよいでしょう。トランザクション属性は宣言型であるため、あとで容易に変更できます。

RequiresNew

クライアントがトランザクション内で動作中に EJB メソッドを呼び出した場合、コンテナは、次の処理を実行します。

1. クライアントのトランザクションを中断します。
2. 新しいトランザクションを開始します。
3. 呼び出しをメソッドに委託します。
4. メソッドの完了後、クライアントのトランザクションを再開します。

クライアントがトランザクションに関連付けられていない場合、コンテナは、新しいトランザクションを開始してからメソッドを実行します。

メソッドを常に新しいトランザクション内で動作させる必要がある場合は、`RequiresNew` 属性を使用します。

Mandatory

クライアントがトランザクション内で動作中に EJB メソッドを呼び出した場合、そのメソッドはクライアントのトランザクション内で動作します。クライアントがトランザクションに関連付けられていない場合、コンテナは `TransactionRequiredException` をスローします。

EJB のメソッドでクライアントのトランザクションを使用する必要がある場合は、`Mandatory` 属性を使用します。

NotSupported

クライアントがトランザクション内で動作中に EJB メソッドを呼び出した場合、コンテナは、クライアントのトランザクションを中断してからメソッドを呼び出します。メソッドの完了後、コンテナはクライアントのトランザクションを再開します。

クライアントがトランザクションに関連付けられていない場合、コンテナは新しいトランザクションを開始せずに、メソッドを実行します。

Supports

クライアントがトランザクション内で動作中に EJB メソッドを呼び出した場合、そのメソッドはクライアントのトランザクション内で動作します。クライアントがトランザクションに関連付けられていない場合、コンテナは新しいトランザクションを開始せずに、メソッドを実行します。

注　メソッドのトランザクション動作は大きく異なるので、`Supports` 属性を使用するには注意が必要です。

Never

クライアントがトランザクション内で動作中に Enterprise JavaBean のメソッドを呼び出した場合、コンテナは `RemoteException` をスローします。クライアントがトランザクションに関連付けられていない場合、コンテナは新しいトランザクションを開始せずに、メソッドを実行します。

トランザクションを必要としないメソッドでは `NotSupported` 属性を使用してください。トランザクションはオーバーヘッドを伴うので、この属性によりパフォーマンスを高めることができます。

次の表は、各トランザクション属性の効果をまとめたものです。左の列はトランザクション属性、中央の列はクライアントトランザクションのタイプ、右の列はビジネスメソッドのトランザクションタイプを示します。トランザクションには、T1、T2、または None (なし) のタイプがあります。T1 および T2 のトランザクションはコンテナによって制御されます。

- T1 トランザクション - EJB 内でメソッドを呼び出すクライアントに関連付けらる。ほとんどの場合、クライアントは別の EJB である
- T2 トランザクション - メソッドの実行直前に、コンテナによって開始される
- None - 3 列目の None は、そのビジネスメソッドが、コンテナによって制御されるトランザクション内で実行されないことを意味する。ただし、ビジネスメソッドなどでのデータベース呼び出しは、そのデータベースのトランザクションマネージャによって制御される場合がある

トランザクション属性と範囲

トランザクション属性	クライアントのトランザクション	ビジネスメソッドのトランザクション
`Required`	なし	T2
	T1	T1
`RequiresNew`	なし	T2
	T1	T2
`Mandatory`	なし	Error
	T1	T1
`NotSupported`	なし	なし
	T1	なし
`Supports`	なし	なし
	T1	T1

トランザクション属性と範囲 (続き)

トランザクション属性	クライアントのトランザクション	ビジネスメソッドのトランザクション
Never	なし	なし
	T1	Error

コンテナ管理トランザクションのロールバック

コンテナ管理トランザクションのロールバックには、次の 2 つの方法があります。

- システム例外がスローされた場合に、コンテナが自動的にトランザクションをロールバックする
- `EJBContext` インタフェースの `setRollbackOnly` メソッドを呼び出すことによって、Bean メソッドがコンテナにトランザクションのロールバックを指示する。Bean がアプリケーション例外をスローした場合は、自動的なロールバックは行われないが、`setRollbackOnly` の呼び出しによってロールバックを起動できる

コンテナがトランザクションをロールバックするときは、コンテナはトランザクション内の SQL 呼び出しによって加えられたデータ変更を実行しません。ただし、エンティティ Beans の場合のみ、コンテナはインスタンス変数に加えられた変更を実行しません。その場合は、エンティティ Beans の `ejbLoad` メソッドを自動的に呼び出して、データベースからインスタンス変数を読み込みます。

セッション Bean では、ロールバックが発生したときに、トランザクション内で変更されたすべてのインスタンス変数を明示的にリセットする必要があります。セッション Bean のインスタンスをリセットするもっとも簡単な方法は、`SessionSynchronization` インタフェースを実装することです。

セッション Beans のインスタンス変数の同期化

セッション Beans にオプションで設定できる SessionSynchronization インタフェースでは、インスタンス変数とデータベース内の対応する値の同期をとることができます。コンテナは、トランザクションの主要ステージごとに SessionSynchronization メソッド (afterBegin、beforeCompletion、および afterCompletion) を呼び出します。

- afterBegin メソッド - 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知らせる。コンテナは、ビジネスメソッドを起動する直前に afterBegin を起動する。afterBegin メソッドは、データベースからインスタンスを読み込むのに適している
- beforeCompletion メソッド - コンテナは、ビジネスメソッドの完了後、トランザクションをコミットする直前に beforeCompletion メソッドを起動する。beforeCompletion メソッドは、セッション Bean が (setRollbackOnly を呼び出して) トランザクションをロールバックする最後の機会である

  インスタンス変数の値によってデータベースがまだ更新されていない場合、セッション Bean は beforeCompletion メソッドでその処理を実行できる
- afterCompletion メソッド - トランザクションが完了したことを示す。このメソッドは 1 つのブール型のパラメータを持ち、その値は、トランザクションがコミットされた場合は true、ロールバックされた場合は false となる

  ロールバックが発生した場合、セッション Bean は、afterCompletion メソッドでデータベースに基づいてインスタンス変数を更新できる

コンテナ管理トランザクションで使用できないメソッド

コンテナ管理トランザクションでは、コンテナが設定するトランザクション境界を侵害する可能性のあるメソッドを起動することはできません。禁止されているメソッドを次に示します。

- java.sql.Connection の commit、setAutoCommit、および rollback 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

ただし、Bean 管理トランザクションで境界を設定する場合は、これらのメソッドを使用できます。

Bean 管理トランザクションの使用法

Bean 管理トランザクションでは、セッション Bean またはメッセージ駆動型 Bean のコードでトランザクションの境界を明示的に指定します。トランザクション管理を Bean レベルに移すことによって、Bean のアクティビティがデータベースアクセスと直接結び付いていなくても、データベース呼び出しと同じトランザクション制御環境ですべての Bean アクティビティを配置できます。これにより、Bean によって制御されるアプリケーション部分はすべて、同じトランザクションの一部として動作します。

障害発生時は、その Bean が管理していたすべてのものがコミットされるか、ロールバックされます。

次の各項では、Bean 管理トランザクションを使用する Enterprise JavaBeans の開発について説明します。

- トランザクションタイプの選択
- コミットなしの復帰
- Bean 管理トランザクションで使用できないメソッド

トランザクションタイプの選択

セッション Beans またはメッセージ駆動型 Beans の Bean 管理トランザクションをコーディングするときは、JDBC または JTA のどちらのトランザクションを使用するかを決める必要があります。

注 Bean 管理トランザクションを使用するセッション Bean では、JDBC と JTA のトランザクションを組み合わせて使用できます。ただし、コードのデバッグおよび保守が難しくなるのでお勧めできません。

次に、両方のトランザクションタイプについて説明します。

- JDBC トランザクション
- JTA トランザクション

JDBC トランザクション

JDBC トランザクションは、データベースのトランザクションマネージャによって制御されます。セッション Beans 内に従来のコードを挿入するときは、JDBC トランザクションの使用をお勧めします。

JDBC トランザクションをコーディングする場合は、`java.sql.Connection` インタフェースの `commit` および `rollback` メソッドを起動します。トランザクションの開始は暗黙的に判別されます。トランザクションは、最新の `commit`、`rollback`、または `connect` ステートメントに続く最初の SQL ステートメントで始まります。通常はこのルールが当てはまりますが、データベースベンダーによっては異なる場合もあります。

JDBC の詳細は、『Sun ONE Application Server Developer's Guide to J2EE Features and Services』を参照してください。

JTA トランザクション

JTA では、トランザクションマネージャの実装に依存しない方法でトランザクションの境界を設定できます。J2EE SDK は、JTS によるトランザクションマネージャを実装しています。しかし、コードでは JTS メソッドを直接呼び出しません。その代わりに、低レベルの JTS ルーチンを呼び出す JTA メソッドを起動します。

JTA トランザクションは、J2EE トランザクションマネージャによって制御されます。異なるベンダーによる複数のデータベースにわたって更新できるため、JTA トランザクションを使用したいと考える場合があります。ただし、特定のデータベースのトランザクションマネージャは異種データベースで動作しない場合があります。

J2EE トランザクションマネージャには、ネストされたトランザクションをサポートしていないという制約があります。つまり、前のトランザクションが終了するまでは、インスタンスのトランザクションを開始できません。

JTA の詳細は、『Sun ONE Application Server Developer's Guide to J2EE Features and Services』を参照してください。

コミットなしの復帰

ビジネスメソッドでトランザクションを開始した、Bean 管理トランザクションのステートレスセッション Bean は、復帰前に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。ただし、ステートフルセッション Beans には、この制約はありません。ステートフルセッション Beans で JTA トランザクションを使う場合、Bean インスタンスとトランザクションの関連付けが複数のクライアント呼び出しで保持されます。

Bean 管理トランザクションで使用できないメソッド

Bean 管理トランザクションでは、EJBContext インタフェースの getRollbackOnly および setRollbackOnly メソッドを起動しないでください。これらのメソッドは、コンテナ管理トランザクションだけで使用されます。

注 Bean 管理トランザクションでは、UserTransaction インタフェースの getStatus および rollback メソッドを起動します。

トランザクションタイムアウトの設定

コンテナ管理トランザクションでは、server.xml ファイル内の timeout-in-seconds プロパティの値を設定することによって、トランザクションタイムアウト間隔を制御します。たとえば、タイムアウト値を 5 秒に設定するには、次のように指定します。

```
timeout-in-seconds=5
```

この設定では、トランザクションが 5 秒以内に完了しなかった場合、EJB コンテナはそのトランザクションをロールバックします。

注 コンテナ管理トランザクションを使用する Enterprise JavaBeans だけが timeout-in-seconds プロパティの適用対象となります。Bean 管理の JTA トランザクションを使用する Enterprise JavaBeans では、UserTransaction インタフェースの setTransactionTimeout メソッドを起動します。

遮断レベルの処理

トランザクションは、トランザクションの内側にあるステートメントを完全に完了 (またはロールバック) するだけでなく、ステートメントによって変更されたデータを遮断します。遮断レベルは、更新されるデータがほかのトランザクションに見える度合いを示します。

トランザクションで、コミットされていないデータをほかのプログラムが読み取ることができるようにすると、ほかのプログラムはそのトランザクションが終了するまで待つ必要がなくなるので、性能を高めることができます。しかし、トランザクションがその後ロールバックされると、他のプログラムは間違ったデータを読み取ってしまうという問題が起こる可能性もあります。

Bean 管理による持続性を使用するエンティティ Beans、およびすべてのセッション Beans では、基礎となるデータベースによって提供される API を使って、プログラムで遮断レベルを設定できます。たとえば、データベースでは、`setTransactionIsolation` メソッドを起動することによって、コミットされていない読み取りを許可することができます。

コンテナ管理による持続性を使用するエンティティ Beans では、`sun-cmp-mapping.xml` ファイルの `consistency` 要素を使って遮断レベルを設定できます。

警告 トランザクションの途中で遮断レベルを変更しないでください。通常、そのような変更は、データベースソフトウェアで暗黙的なコミットが発生する原因となります。データベースベンダーが提供する遮断レベルはさまざまなので、データベースのマニュアルで詳細を確認する必要があります。J2EE プラットフォームでの遮断レベルは標準化されていません。

第 7 章

安全な Enterprise JavaBeans の開発

この章では、EJB アーキテクチャにおけるセキュリティ管理の動作および Sun ONE Application Server 7 環境で安全な Enterprise JavaBeans を開発するための手引きについて説明します。

注 EJB テクノロジに精通していない場合は、Java Software チュートリアルを参照してください。

http://java.sun.com/j2ee/docs.html

EJB のセキュリティに関する詳細情報は、『Enterprise JavaBeans Specification, v2.0』の第 21 章「Security Management」にあります。

アプリケーションのセキュリティに関する一般情報は、『Sun ONE Application Server 開発者ガイド』を参照してください。

この章には次の項目があります。

- 安全な Enterprise JavaBeans について
- セキュリティロールの定義
- メソッドパーミッションの宣言
- セキュリティロール参照の宣言
- セキュリティ ID の指定
- プログラムによるセキュリティの使用法
- 保護されていない EJB 層のリソースの処理

アプリケーションのセキュリティに関する一般情報は、『Sun ONE Application Server 開発者ガイド』を参照してください。

安全な Enterprise JavaBeans について

EJB 開発者の主な役割は、アプリケーションのセキュリティ要件を宣言して、アプリケーションの配備時にそれらの要件が満たされるようにすることです。ほとんどの場合、EJB のビジネスメソッドにはセキュリティ関連のロジックを含めることはできません。

この節では次の項目について説明します。

- 承認と認証
- セキュリティロール
- 配備

承認と認証

承認により、保護されたリソースへのアクセスを制御します。承認は、識別と認証に基づきます。識別は、システムがエンティティを認識できるようにするプロセスです。認証は、コンピュータシステム内のユーザ、デバイス、またはその他のエンティティの識別情報を検証するプロセスであり、通常は、システム内のリソースへのアクセスを許可するための前提条件に従います。

適切な承認レベルを持つユーザのアクセスだけを許可するように Enterprise JavaBeans を設定できます。そのためには、Sun ONE Application Server の Administration interface を使用して、アプリケーションの EAR および EJB JAR ファイル用の配備記述子を生成します。

セキュリティロール

セキュリティロールは、アプリケーション固有の論理的なユーザのグループ分けであり、顧客プロフィールや役職など、一般的な特性によって分類されます。アプリケーションの配備時に、プリンシパル (認証の結果としてユーザに割り当てられる ID) やグループなどのセキュリティ ID にロールがマップされます。これに基づいて、特定のセキュリティロールを持つユーザに Enterprise JavaBean へのアクセス権が割り当てられます。このリンクが、セキュリティロールが参照されるときの実際の名前になります。

グループもユーザのカテゴリを表しますが、グループの範囲はロールの範囲とは異なります。

- ロールは J2EE アプリケーション固有の抽象概念である
- グループは、現在のレルムに基づいた環境固有のユーザの集合である。グループのメンバーシップは、基になるレルムの実装によって決まる

注 メソッドの制限事項とロールマッピングを定義するときは、グループのレルムと J2EE アプリケーションロールが混同されがちです。このような混乱は、予期せぬアクセスや、動作不可能なアプリケーション設定につながりかねません。詳細は、『Sun ONE Application Server 開発者ガイド』を参照してください。

配備

セキュリティロール参照は、`isCallerInRole` (文字列名) を使って ENTERPRISE JAVABEAN から呼び出されるロール名と、アプリケーションで定義されているセキュリティロール名のマッピングを定義します。このセキュリティロール参照により、Enterprise JavaBean は既存のセキュリティロールを参照できます。

アプリケーションを配備するときに、配備担当者は、本稼動環境に存在するセキュリティ ID にロールをマップします。Enterprise JavaBean を開発するときは、開発者はユーザのロールについて知っている必要がありますが、誰がそのユーザになるかについてはほとんど知らない場合があります。それは、J2EE のセキュリティアーキテクチャで処理されます。開発したコンポーネントが配備されたあとで、システム管理者が、デフォルトレルム (通常はファイルレルム) の J2EE ユーザ (またはグループ) にそれらのロールをマップします。

セキュリティロールの定義

J2EE アプリケーションのロールを作成するには、アプリケーションに含まれる EJB JAR ファイルまたは WAR ファイルに対するロールを宣言します。`security-role` 要素で定義されたセキュリティロールは、EJB JAR ファイルレベルの適用対象となり、EJB JAR ファイル内のすべての Enterprise JavaBeans に適用されます。

例

次の例は、配備記述子内のセキュリティロール定義を示しており、`employee` と `admin` の 2 つの role-name 要素を指定しています。

```
...
<assembly-descriptor>
   <security-role>
      <description>
         このロールには、サービスアプリケーションに
         アクセスできる社員が含まれます。
         このロールは、その社員の情報への
         アクセスだけが許可されています。
```

```
        </desciption>
        <role-name>employee</role-name>
        </security-role>
        <security-role>
            <description>
                このロールは、自分の目的に使うアプリケーションの管理業務を実行する
                権限を持つ担当者に割り当てる必要があります。機密情報である人事
                および給与情報に直接アクセスすることはありません。
            </desciption>
        <role-name>admin</role-name>
        </security-role>
...
</assembly-descriptor>
```

メソッドパーミッションの宣言

メソッドパーミッションは、どのロールがどのメソッドの起動を許可されているかを示します。アプリケーションのアセンブリ担当者は、次のように、配備記述子内で method-permission 要素を使ってメソッドパーミッションの関係を宣言します。

- 各 `method-permission` 要素には、1 つまたは複数のセキュリティロールのリスト、および 1 つまたは複数のメソッドのリストが含まれる

  一覧表示されたセキュリティロールは一覧表示されたすべてのメソッドを起動できる。リスト内の各セキュリティロールは `role-name` 要素によって識別され、各メソッド (または下記の一連のメソッド) は method 要素によって識別される。description 要素を使って オプションの説明を `method-permission` 要素に関連付けることができる

- メソッドパーミッションの関係は、個々の method permission 要素に定義したすべてのメソッドパーミッションの結合として定義される

- セキュリティロールまたはメソッドは、複数の `method-permission` 要素内に存在することがある

例

次の配備記述子の例は、配備記述子内でセキュリティロールをメソッドパーミッションに割り当てる方法を示します。これらは配備時にセキュリティ要素に変換されます。

```
...
<method-permission>
    <role-name>employee</role-name>
        <method>
        <ejb-name>EmployeeService</ejb-name>
```

```
      <method-name>*</method-name>
   </method>
</method-permission>

<method-permission>
   <role-name>employee</role-name>
   <method>
      <ejb-name>AardvarkPayroll</ejb-name>
      <method-name>findByPrimaryKey</method-name>
   </method>
   <method>
      <ejb-name>AardvarkPayroll</ejb-name>
      <method-name>getEmployeeInfo</method-name>
   </method>
   <method>
      <ejb-name>AardvarkPayroll</ejb-name>
      <method-name>updateEmployeeInfo</method-name>
   </method
</method-permission>
...
```

セキュリティロール参照の宣言

EJB 開発者は、Enterprise JavaBeans 内でプログラムによって使用されるロールについて、Enterprise JavaBean で使用するすべてのセキュリティロールの名前を配備記述子の `security-role-ref` 要素に宣言する必要があります。

- アプリケーションのアセンブリ担当者は、`security-role-ref` 要素で宣言されたすべてのセキュリティロール参照を、`security-role` 要素で定義されたセキュリティロールにリンクする必要がある
- アプリケーションのアセンブリ担当者は、`role-link` 要素を使って各セキュリティロール参照をセキュリティロールにリンクする

注　　`role-link` 要素の値は、`security-role` 要素で定義されたセキュリティロール名の一つである必要があります。

例

次の配備記述子の例は、`payroll` という名前のセキュリティロール参照を `payroll-department` という名前のセキュリティロールにリンクする方法を示します。

```
<enterprise-beans>
   ...
   <entity>
      <ejb-name>AardvarkPayroll</ejb-name>
      <ejb-class>com.aardvark.payroll.PayrollBean</ejb-class>
   ...
      <security-role-ref>
      <description> このロールは、給与支払い部門の社員に割り当てる必要があり
      ます。このロールが割り当てられたメンバーは全員の給与記録にアクセスできま
      す。このロールは payroll-department ロールにリンクされています。
      </description>
      <role-name>payroll</role-name>
      <role-link>payroll-department</role-link>
      </security-role-ref>
   ....
   </entity>
   ...
</enterprise-beans>
```

このロールは、給与支払い部門の社員に割り当てる必要があります。このロールが割り当てられたメンバーは全員の給与記録にアクセスできます。このロールは payroll-department ロールにリンクされています。

セキュリティロールの詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。EJB アクセス制御の設定については、『Enterprise JavaBeans Specification, v2.0』を参照してください。

セキュリティ ID の指定

オプションで、EJB のアセンブリ担当者は、EJB メソッドの実行に関して呼び出し元の ID を使用するかどうか、あるいは特定の実行 ID を使用するかどうかを指定できます。配備記述子内の security-identity 要素は、この目的に使用されます。security-identity 要素の値は、use-caller-identity または run-as です。

指定しない場合は、デフォルトの呼び出し元の ID が適用されます。

実行 ID

実行 ID は、Enterprise JavaBean が呼び出しを行うときに使用する ID を指定します。呼び出し元の ID には影響しません。呼び出し元の ID は、Enterprise JavaBean のメソッドにアクセスするためのパーミッションを検査する ID です。

EJB のアセンブリ担当者は、run-as 要素を使って、配備記述子内に Enterprise JavaBean の実行 ID を定義できます。実行 ID は、Enterprise JavaBean 全体、つまり、EJB のホームインタフェースおよびコンポーネントインタフェースのすべてのメソッド、またはメッセージ駆動型 Beans の onMessage メソッド、および呼び出される可能性がある Enterprise JavaBean のすべての内部メソッドに適用されます。

アセンブリ担当者は、通常は本稼動環境のセキュリティ環境については把握していないので、実行 ID は、配備記述子内に定義されたセキュリティロールの 1 つに対応する論理 role-name によって指定します。その後、配備担当者は実行 ID のプリンシパルとして使用するセキュリティプリンシパル (本稼動環境で定義される) を割り当てる必要があります。セキュリティプリンシパルは、role-name 要素で指定されたセキュリティロールに割り当てられたプリンシパルである必要があります。

プログラムによるセキュリティの使用法

一般に、セキュリティ管理は、EJB のビジネスメソッドに対して透過的な方法でコンテナによって実行される必要があります。

注 Enterprise JavaBean では、サーブレットと同じようにプログラムによるログインを使います。詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

EJB 層でのプログラムによるセキュリティは、getCallerPrincipal メソッドと isCallerInRole メソッドで構成されます。getCallerPrincipal メソッドを使って EJB の呼び出し元を指定し、isCallerInRole メソッドを使って呼び出し元のロールを指定することができます。

EJBContext インタフェースの getCallerPrincipal メソッドは、Enterprise JavaBean の呼び出し元を特定する java.security.Principal オブジェクトを返します。この場合、プリンシパルはユーザと同じになります。次の例では、Enterprise JavaBean の getUser メソッドが、その呼び出し元の J2EE ユーザの名前を返します。

```
public String getUser()
{
   return context.getCallerPrincipal().getName();
}
```

次のように、isCallerInRole メソッドを呼び出すことによって、EJB の呼び出し元が特定のロールに属するかどうかを判別できます。

```
boolean result = context.isCallerInRole("Customer");
```

プログラムによるセキュリティの実装方法の詳細は、『Enterprise JavaBeans Specification, v2.0』の第 21 章「Security Management」を参照してください。

保護されていない EJB 層のリソースの処理

すべてのユーザには匿名のロールが割り当てられています。デフォルトでは、匿名ロールの値は ANYONE であり、`server.xml` ファイルで設定できます。メソッドパーミッションで、必要なロールは ANYONE (または匿名ロールに設定された値) であると指定されていると、すべてのユーザがこのメソッドにアクセスできます。

注　　あるメソッドを対象とするメソッドパーミッションが存在しない場合は、すべてのユーザがそのメソッドにアクセスできます。

メソッドパーミッションが存在する場合は、常にそのパーミッションが適用されます。たとえば、メソッドパーミッションで、`updateEmployeeInfo` メソッドには `employee` ロールのみがアクセスできるように設定されている場合、`employee` ロールを持たなければこのメソッドにはアクセスできません。`employee` ロールがどのユーザまたはグループにもマップされていない場合は、誰も `updateEmployeeInfo` メソッドを呼び出すことはできません。

第 8 章

Enterprise JavaBean のアセンブルと配備

この章では、Sun ONE Application Server 7 環境で Enterprise JavaBeans をアセンブルおよび配備する方法について説明し、EJB XML ファイルの作成に使用する要素およびサブ要素に関する情報を提供します。

注	アセンブリと配備に関する一般情報については、『Sun ONE Application Server 開発者ガイド』を参照してください。EJB をアセンブルする前に、このマニュアルに示された配備に関する情報を十分に理解している必要があります。

この章には次の項目があります。

- EJB の構造
- 配備記述子の作成
- Enterprise JavaBeans の配備
- sun-ejb-jar_2_0-0.dtd ファイルの構造
- sun-ejb-jar.xml ファイルの要素
- EJB XML ファイルの例

EJB に関連する要素のリストは、229 ページの「要素一覧」にアルファベット順に記載されています。

EJB の構造

EJB Java ARchive (JAR) ファイルは、Enterprise JavaBeans のアセンブリに使う標準形式です。このファイルには、Bean クラス (ホーム、リモート、ローカル、および実装)、すべてのユーティリティクラス、および配備記述子 (`ejb-jar.xml` および `sun-ejb-jar.xml`) が格納されます。

開発者が作成する EJB JAR ファイルには、1 つ以上の Enterprise JavaBeans が含まれ、通常は、アセンブリ手順は含まれません。一方、アセンブリ担当者が作成する EJB JAR ファイルには、1 つ以上の Enterprise JavaBeans と、それらの Enterprise JavaBeans を 1 つのアプリケーション配備ユニットに結合する方法を示すアプリケーションアセンブリ手順が含まれます。

EJB JAR ファイルは、EAR (Enterprise ARchive) ファイルの一部に含まれないスタンドアロンとしても、EAR ファイルの一部としても使用できます。

アプリケーションのファイル例は *install_root*`/samples/j2ee/` にあります。

配備記述子の作成

J2EE モジュールは、1 つまたは複数の J2EE コンポーネントの集合で、各コンポーネントは同一コンテナタイプの 2 つの配備記述子を持っています。1 つは J2EE 標準の配備記述子で、もう 1 つは Sun ONE Application Server 固有の配備記述子です。Enterprise JavaBeans では、次の 3 つの配備記述子ファイルがこれに該当します。

- `ejb-jar.xml`

  J2EE 標準ファイル。『Enterprise JavaBeans Specification, v2.0』に定義されている

- `sun-ejb-jar.xml`

  Sun ONE Application Server 固有のファイル。この章で詳しく説明する

- `sun-cmp-mappings.xml`

  Sun ONE Application Server 固有のファイル。配備する Bean がコンテナ管理による持続性を使用する場合に使用する

注　コンテナ管理による持続性に関連付けられた XML ファイルについては、111 ページの「sun-cmp-mappings.xml ファイルの要素」を参照してください。

配備記述子ファイルを作成するもっとも簡単な方法は、管理インタフェースまたは Sun ONE Studio 4 IDE を使用して EJB モジュールを配備することです。詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。EJB XML ファイルの例は、213 ページの「EJB XML ファイルの例」を参照してください。

これらのファイルを作成した後に、管理インタフェースまたは、エディタとコマンド行ユーティリティ (Ant など) の組み合わせを使用して、編集、再構築、および再配備を行い配備記述子の情報を更新できます。

注　　配備記述子を手動で作成することもできます。

J2EE の標準配備記述子は、1.3 の J2EE 仕様に示されています。EJB 配備記述子の詳細は、『Enterprise JavaBeans Specification, v2.0』の第 22 章を参照してください。サンプルアプリケーションは、アセンブルと配備に役立つ ANT ターゲットをいくつか作成します。ANT については、『Sun ONE Application Server 開発者ガイド』を参照してください。

Enterprise JavaBeans の配備

Enterprise JavaBean の配備、配備取り消し、または再配備を行うときに、サーバーを再起動する必要はありません。

注　　スタブとスケルトンは配備時に生成されます。リッチクライアントで使用するクライアント JAR ファイルは、スタブとスケルトンから作成できます。

この節には次の項目があります。

- 管理インタフェースの使用
- コマンド行インタフェースの使用
- Sun ONE Studio 4 IDE の使用
- Enterprise JavaBeans の再読み込み

管理インタフェースの使用

管理インタフェースを使って EJB アプリケーションを配備するには、次のようにします。

1. サーバーインスタンスの下にある「アプリケーション」コンポーネントを開きます。
2. 「EJB モジュール」ページに移動します。
3. 「配備」をクリックします。
4. JAR モジュールへのフルパスを入力するか、または「ブラウズ」をクリックして目的のモジュールを選択して、「了解」をクリックします。

コマンド行インタフェースの使用

コマンド行を使って Enterprise JavaBean を配備するには、次のようにします。

1. 配備記述子ファイル (`ejb-jar.xml` および `sun-ejb-jar.xml`) を手動で編集します。
2. Ant ビルドコマンド (`build jar` など) を実行して、JAR モジュールを再アセンブリします。
3. `asadmin deploy` コマンドを使って JAR モジュールを配備します。構文は次のとおりです。

   ```
   asadmin deploy -type ejb [-name component-name] [-force=true]
   [-upload=true] -instance instancename filepath
   ```

   たとえば、次のコマンドでは、EJB アプリケーションがスタンドアロンのモジュールとして配備されます。

   ```
   asadmin deploy -type ejb -instance inst1 myEJB.jar
   ```

Sun ONE Studio 4 IDE の使用

Sun ONE Application Server にバンドルされている Sun ONE Studio 4 IDE を使って、Enterprise JavaBeans のアセンブリと配備を行うことができます。Sun ONE Studio 4 の使用については、Sun ONE Studio 4, Enterprise Edition のチュートリアルを参照してください。

注 Sun ONE Studio 4 では、Web アプリケーションの配備を「実行」と呼びます。

Enterprise JavaBeans の再読み込み

Enterprise JavaBean のコードを変更した場合でも、動的再読み込みが有効になっていれば、その Enterprise JavaBean を再配備したり、サーバーを再起動したりする必要はありません。変更したファイルをアプリケーションの配備先ディレクトリ (*instance-dir*/applications など) にドロップするだけで変更が適用されます。

管理インタフェースで動的再読み込みを有効にするには、次のようにします。

1. 管理インタフェースで、サーバーインスタンスを選択します。
2. 「アプリケーション」を選択します。

   「アプリケーション - プロパティ」ページが表示されます。
3. 「再読み込みを有効」ボックスをオンにして動的再読み込みを有効にします。
4. 「再読込のポーリング間隔」フィールドに秒数を入力して、アプリケーションとモジュールにコードの変更がないか確認して動的に再読み込みする間隔を設定します。
5. 「保存」をクリックします。

詳細については、『Sun ONE Application Server 管理者ガイド』を参照してください。

さらに、新しいサーブレットファイルの読み込み、変更に関連する EJB の再読み込み、または配備記述子の変更の再読み込みを行うには、次の操作を行う必要があります。

1. 配備されたアプリケーションのルートに .reload という名前の空のファイルを作成します。

   instance_dir/applications/j2ee-apps/*app_name*/.reload

   または個別に配備されたモジュールに作成します。

   instance_dir/applications/j2ee-modules/*module_name*/.reload
2. 上記の変更を Bean または配備記述子に加えるたびに、.reload ファイルのタイムスタンプ (UNIX では touch .reload) を明示的に更新します。

   再読み込み監視スレッドが定期的に .reload ファイルのタイムスタンプを確認し、変更を検出します。検出間隔は、デフォルトの設定では 2 秒です。この値は、server.xml ファイルの dynamic-reload-poll-interval-in-seconds で変更できます。

EJB アプリケーションは、次のいずれかの方法で配備できます。

- コマンド行インタフェースの使用
- 管理インタフェースの使用
- Sun ONE Studio 4 IDE の使用

配備の詳細については、『Sun ONE Application Server 開発者ガイド』を参照してください。

sun-ejb-jar_2_0-0.dtd ファイルの構造

sun-ejb-jar_2_0-0.dtd ファイルは、sun-ejb-jar.xml ファイルの構造を定義します。含むことができる要素、およびそれらの要素に持たせることができるサブ要素と属性が定義されます。sub-ejb-jar_2_0-0.dtd ファイルは、*install-dir*/lib/dtds ディレクトリに保存されています。

注 sun-ejb-jar_2_0-0.dtd ファイルを編集しないでください。このファイルの内容は、Sun ONE Application Server の新しいバージョンだけで変更されます。

DTD ファイルおよび XML の全般的な情報については、次のサイトにある XML 仕様書を参照してください。

http://www.w3.org/TR/REC-xml

DTD ファイルに定義される各要素 (対応する XML ファイルに含まれていることもあります) には次の情報が含まれます。

- サブ要素
- データ
- 属性

EJB に関連する要素のリストは、229 ページの「要素一覧」にアルファベット順に記載されています。

サブ要素

要素にはサブ要素を含めることができます。たとえば、次のコードは cmp-resource 要素を定義しています。

```
<!ELEMENT cmp-resource (jndi-name, default-resource-principal?)>
```

この ELEMENT タグは、cmp-resource というリソース要素に jndi-name および default-resource-principal というサブ要素を含めることができるように指定しています。疑問符 (?) は、サブ要素 default-resource-principal を 0 または 1 回指定できることを示します。

それぞれのサブ要素には、指定回数を決めるオプション文字 (サフィックス) を追加できます。

次の表は、サブ要素に追加したサフィックスによって決定されるサブ要素の必要規則 (指定可能回数) について説明しています。左側の列にはサブ要素の終了文字、右側の列には対応する必要指定数を示しています。

サブ要素のサフィックスの必要規則

サフィックス	指定回数
*element**	このサブ要素を含まないか、1 個以上含めることができる
?*element*	このサブ要素を含まないか、1 個含めることができる
element+	このサブ要素を 1 個以上含まなければならない
element (サフィックスなし)	このサブ要素を 1 個だけ含まなければならない

要素にほかの要素を含めることができない場合は、カッコで囲まれた要素名のリストの代わりに、EMPTY または (#PCDATA) が表示されます。

データ

要素の中には、サブ要素の代わりに文字データを含むものもあります。これらの要素は、次の形式で定義されます。

```
<!ELEMENT element-name (#PCDATA)>
```

次に例を示します。

```
<!ELEMENT description (#PCDATA)>
```

Sun ONE Application Server の XML ファイルでは、データ要素内の空白はデータの一部として扱われます。そのため、データ要素で区切られたデータの前後には余分な空白がないようにする必要があります。次に例を示します。

```
<description>class name of session manager</description>
```

```
<password>secret</password>
```

属性

要素には属性 (名前と値のペア) を含めることができます。属性は、ATTLIST タグを使って属性リストに定義します。

sun-ejb-jar.xml ファイル内の要素には、属性を含むものはありません。

sun-ejb-jar.xml ファイルの要素

EJB に関連する要素のリストは、229 ページの「要素一覧」にアルファベット順に記載されています。

注	コンテナ管理による持続性のマッピングに関連付けられた DTD および XML ファイルについては、111 ページの「sun-cmp-mappings.xml ファイルの要素」を参照してください。

この節では、sun-ejb-jar_2_0-0.dtd ファイル内の XML 要素について説明します。わかりやすくするために、各要素を次のように分類しています。

- 一般的な要素
- ロールマッピング要素
- 参照要素
- セキュリティ要素
- 持続性要素
- プールとキャッシュの要素
- クラス要素

注	sun-ejb-jar.xml ファイルに Enterprise JavaBean の設定を指定しない場合は、対応する設定が server.xml ファイルの ejb-container 要素に含まれていれば、この設定がデフォルト値となります。server.xml ファイルでデフォルト値を変更すると、その値が定義されていないすべての Enterprise JavaBean に変更を適用できます。

一般的な要素

全般に関する要素は次のとおりです。

- ejb
- ejb-name
- enterprise-beans

- is-read-only-bean
- refresh-period-in-seconds
- sun-ejb-jar
- unique-id

ejb

アプリケーション内の 1 つの Enterprise JavaBean に関する実行時プロパティを定義します。以下に示すサブ要素は、次のように特定の Enterprise JavaBean に適用されます。

- すべてのタイプの Beans - `ejb-name`、`ejb-ref`、`resource-ref`、`resource-env-ref`、`cmp`、`ior-security-config`、`gen-classes`、`jndi-name`
- ステートレスセッション Beans とメッセージ駆動型 Beans - `bean-pool`
- ステートフルセッション Beans - `bean-cache`
- エンティティ Beans (BMP) - `is-read-only-bean`、`refresh-period-in-seconds`、`commit-option`、`bean-cache`
- メッセージ駆動型 Bean - `mdb-connection-factory`、`jms-durable-subscription-name`、`jms-max-messages-load`、`bean-pool`

サブ要素

次の表は、`ejb` 要素のサブ要素を示します。

`ejb` 要素のサブ要素

サブ要素	必要指定数	説明
`ejb-name`	1 個のみ	参照する Bean の表示名と一致させる
`jndi-name`	0 または 1 個以上	JNDI 絶対名 `jndi-name` を指定する。メッセージ駆動型 Bean の場合、これはメッセージ駆動型 Bean クラスに関連付けられた Java Message Service のキューまたはトピックの送信先リソースオブジェクトの JNDI 名となる。キューまたはトピックのどちらのタイプかは、メッセージ駆動配備記述子 message-driven-destination のデスティネーションタイプによって決まる。message-driven-destination 配備記述子が指定されていない場合は、デフォルトのキュータイプとなる

ejb 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
ejb-ref	0 または 1 個以上	対応する J2EE XML ファイルの ejb-ref 要素に JNDI 絶対名をマップする
resource-ref	0 または 1 個以上	対応する J2EE XML ファイルの resource-ref に JNDI 絶対名を割り当てる
resource-env-ref	0 または 1 個以上	対応する J2EE XML ファイルの resource-env-ref に JNDI 絶対名を割り当てる
pass-by-reference	0 または 1 個	サーブレットまたは EJB から同じプロセス内にある別の Bean を呼び出すときに、Sun ONE Application Server は、すべての呼び出しパラメータの整列を自動実行しない
cmp	0 または 1 個	EJB1.1 および EJB2.0 の Beans の コンテナ管理による持続性 (CMP) EntityBean オブジェクトに関する実行時情報を指定する。これは、Bean のマッピング情報が記述されたファイルへのポインタである
principal	0 または 1 個	run-as ロールが指定されている Enterprise JavaBean 内の主体(ユーザー)名を指定する
mdb-connection-factory	0 または 1 個	メッセージ駆動型 Bean クラスに関連付けられたコネクションファクトリを指定する
jms-durable-subscription-name	0 または 1 個	メッセージ駆動型 Bean に関連付けられた永続的なサブスクリプションを示すデータを指定する
jms-max-messages-load	0 または 1 個	メッセージ駆動型 Beans による処理で Java Message Service セッションに一度に読み込む最大メッセージ数を指定する。デフォルト値は 1
ior-security-config	0 または 1 個	IOR のセキュリティ情報を指定する
is-read-only-bean	0 または 1 個	読み取り専用 Bean を示すフラグ

ejb 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
refresh-period-in-seconds	0 または 1 個	読み取り専用 Bean をデータソースによって更新する間隔を指定する。これをゼロ以下に設定すると、Bean は更新されない。1 以上を設定すると、指定した間隔で Bean のインスタンスが更新される。この間隔はコンテナにとって参考程度に過ぎない。デフォルトは 600
commit-option	0 または 1 個	A、B、または C の有効値を持つデータを指定する。デフォルト値は B
gen-classes	0 または 1 個	Bean で生成されるすべてのクラス名を指定する
bean-pool	0 または 1 個の bean-pool	Bean のプールプロパティを指定する。ステートレスセッション Beans、エンティティ Beans、およびメッセージ駆動型 Beans のプールで使用される
bean-cache	0 または 1 個の bean-pool	Bean のキャッシュプロパティを指定する。ステートフルセッション Beans とエンティティ Beans だけで使用される

例

```
<ejb>
      <ejb-name>CustomerEJB</ejb-name>
      <jndi-name>customer</jndi-name>
      <resource-ref>
             <res-ref-name>jdbc/SimpleBank</res-ref-name>
             <jndi-name>jdbc/PointBase</jndi-name>
      </resource-ref>
      <is-read-only-bean>false</is-read-only-bean>
      <commit-option>B</commit-option>
      <bean-pool>
             <steady-pool-size>10</steady-pool-size>
             <resize-quantity>10</resize-quantity>
             <max-pool-size>100</max-pool-size>
             <pool-idle-timeout-in-seconds>
             600
             </pool-idle-timeout-in-seconds>
      </bean-pool>
      <bean-cache>
```

```
            <max-cache-size>100</max-cache-size>
            <resize-quantity>10</resize-quantity>
        <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
            <victim-selection-policy>LRU</victim-selection-policy>
        </bean-cache>
</ejb>
```

ejb-name

参照する Enterprise JavaBean の名前と表示名を一致させます。この名前は、Enterprise JavaBean の名前を EJB JAR ファイルの配備記述子に記録するときに同ファイルの作成者によって割り当てられます。同じ EJB JAR ファイルでは、Enterprise JavaBean の名前を一意にする必要があります。

配備記述子に記載される `ejb-name` と配備担当者が EJB のホームに割り当てる JNDI 名の間には、設計上の関連性はありません。

サブ要素

なし

例

```
<ejb-name>EmployeeService</ejb-name>
```

enterprise-beans

アプリケーションの EJB JAR ファイルに関するすべての実行時プロパティを指定します。

サブ要素

次の表は、`enterprise-beans` 要素のサブ要素を示します。

`enterprise-beans` 要素のサブ要素

サブ要素	必要指定数	説明
`name`	0 または 1 個	名前の文字列を指定する
`unique-id`	0 または 1 個	一意のシステム識別子を指定する。このデータは、配備時および再配備時に自動的に生成および更新される
`ejb`	0 または 1 個以上	アプリケーション内の 1 つの Enterprise JavaBean に関する実行時プロパティを定義する

enterprise-beans 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
pm-descriptors	0 または 1 個	持続性マネージャ記述子を記述する。特定の時間にその 1 つが使用されている必要がある。これは、基本的に、Sun ONE Application Server のプラグイン可能な持続性マネージャ API に適用される
cmp-resource	0 または 1 個	EJB JAR ファイル内のコンテナ管理による持続性 (CMP) Bean を保存するために使用するデータベースを指定する

例

```
<enterprise-beans>
   <ejb>
      ejb-name>CustomerEJB</ejb-name>
      <jndi-name>customer</jndi-name>
      <resource-ref>
         <res-ref-name>jdbc/SimpleBank</res-ref-name>
'         <jndi-name>jdbc/PointBase</jndi-name>
      </resource-ref>
      <is-read-only-bean>false</is-read-only-bean>
      <commit-option>B</commit-option>
      <bean-pool>
         <steady-pool-size>10</steady-pool-size>
         <resize-quantity>10</resize-quantity>
         <max-pool-size>100</max-pool-size>
         <pool-idle-timeout-in-seconds>
         600
         </pool-idle-timeout-in-seconds>
      </bean-pool>
      <bean-cache>
         <max-cache-size>100</max-cache-size>
         <resize-quantity>10</resize-quantity>
         <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
         <victim-selection-policy>LRU</victim-selection-policy>
      </bean-cache>
   </ejb>
</enterprise-beans
```

is-read-only-bean

読み取り専用 Bean を示すフラグです。

サブ要素

なし

例

```
<is-read-only-bean>false</is-read-only-bean>
```

refresh-period-in-seconds

読み取り専用 Bean をデータソースによって更新する間隔を指定します。これをゼロ以下に設定すると、Bean は更新されない。1 以上を設定すると、指定した間隔で Bean のインスタンスが更新されます。この間隔はコンテナにとって参考程度に過ぎません。デフォルト値は 600 です。

サブ要素

なし

sun-ejb-jar

アプリケーションの EJB JAR ファイルに関する Sun ONE Application Server 固有の設定を定義します。これはルート要素であり、`sun-ejb-jar.xml` ファイル内に 1 つの `sun-ejb-jar` しか指定できません。

このファイルの例については、214 ページの「sun-ejb-jar.xml ファイルの例」を参照してください。

サブ要素

次の表は、`sun-ejb-jar` 要素のサブ要素を示します。

`sun-ejb-jar` 要素のサブ要素

サブ要素	必要指定数	説明
`security-role-mapping`	0 または 1 個以上	対応する J2EE XML ファイル内のロールを、ユーザーまたはグループにマップする
`enterprise-beans`	1 個のみ	アプリケーションの EJB JAR ファイルに関するすべての実行時プロパティを記述する

unique-id

一意のシステム識別子を指定します。このデータは、配備時および再配備時に自動的に生成および更新されます。開発者は、配備後にこの値を変更できません。

サブ要素

なし

ロールマッピング要素

ロールマッピング要素は、EJB JAR の `role-name` エントリに指定されたロールを、環境固有のユーザーまたはグループにマップします。ユーザーにマップする場合、そのユーザーは、現在のレルムに存在し、現在の認証方法によってサーバーにログインできる具体的なユーザーでなければなりません。グループにマップする場合は、レルムがグループをサポートしている必要があり、そのグループは現在のレルムに存在する具体的なグループでなければなりません。グループのマッピングを有効にするためには、そのレルム内でそのグループに属するユーザーが 1 人以上存在している必要があります。

次に、ロールマッピング要素を示します。

- group-name
- principal
- principal-name
- role-name
- security-role-mapping
- server-name

group-name

グループ名を指定する

サブ要素

なし

principal

プラットフォームでのユーザー名を指定するノードを定義します。

サブ要素

次の表は、`principal` 要素のサブ要素を示します。

principal 要素のサブ要素

サブ要素	必要指定数	説明
name	1 個のみ	ユーザーの名前を指定する

principal-name

run-as ロールが指定されている Enterprise JavaBean 内の主体 (ユーザー) 名を指定します。

サブ要素

なし

role-name

ejb-jar.xml ファイルの security-role 要素内の role-name を指定します。

サブ要素

なし

例

```
<role-name>employee</role-name>
```

security-role-mapping

ユーザーおよびグループにロールをマップします。

サブ要素

次の表は、security-role-mapping 要素のサブ要素を示します。

security-role-mapping 要素のサブ要素

サブ要素	必要指定数	説明
role-name	1 個のみ	マップする ejb-jar.xml ファイルの role-name を指定する
principal-name	1 個以上の principal-name または group-name	実行ロールが指定されている EJB 内の主体 (ユーザー) 名を指定する

security-role-mapping 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
group-name	1 個以上の principal-name または group-name	グループ名を指定する

server-name

アプリケーションを配備するサーバーの名前を指定します。

サブ要素

なし

参照要素

参照に関する要素は次のとおりです。

- ejb-ref
- ejb-ref-name
- jndi-name
- pass-by-reference
- res-ref-name
- resource-env-ref
- resource-env-ref-name
- resource-ref

ejb-ref

対応する J2EE XML ファイルの ejb-ref 要素に JNDI 絶対名 jndi-name をマップします。ejb-ref 要素は、EJB のホームへの参照の宣言に使用されます。

ステートフルセッション Beans またはエンティティ Beans に適用されます。

サブ要素

次の表は、ejb-ref 要素のサブ要素を示します。

ejb-ref 要素のサブ要素

サブ要素	必要指定数	説明
ejb-ref-name	1 個のみ	対応する J2EE EJB JAR ファイルの ejb-ref エントリ内の ejb-ref-name を指定する
jndi-name	1 個のみ	JNDI 絶対名 jndi-name を指定する

ejb-ref-name

対応する J2EE XML ファイルの ejb-ref エントリ内の ejb-ref-name を指定します。この名前は、同じ Enterprise JavaBean 内では一意にする必要があります。名前の最初に ejb/ というプレフィックスをつけることをお勧めします。

サブ要素

なし

例

```
<ejb-ref-name>ejb/Payroll</ejb-ref-name>
```

jndi-name

JNDI 絶対名 jndi-name を指定します。

すべての Enterprise JavaBean に適用されます。

サブ要素

なし

例

```
<jndi-name>jdbc/PointBase</jndi-name>
```

pass-by-reference

同じプロセス内の別の Bean に含まれるリモートインタフェースメソッドを呼び出すサーブレットまたは Enterprise JavaBean に適用させる受け渡し方法を指定します。デフォルト値は false です。

注 pass-by-reference フラグは、リモートインタフェースへの呼び出しだけに適用されます。『Enterprise JavaBeans Specification, v2.0』に定義されているように、ローカルインタフェースへの呼び出しでは、参照セマンティックによる受け渡しが使用されます。

- false (この要素が存在しない場合のデフォルト値) の場合、このアプリケーションは、『Enterprise JavaBeans Specification, v2.0』で要求されている pass-by-value セマンティックを使用する
- true の場合、このアプリケーションは pass-by-reference セマンティックを使用する

サーブレットまたは Enterprise JavaBean が、同じプロセス内の別の Bean に含まれるリモートインタフェースメソッドを呼び出すと、デフォルトの設定では、Sun ONE Application Server はすべての呼び出しパラメータをコピーして pass-by-value セマンティックを保存します。これにより呼び出しのオーバヘッドが増加し、パフォーマンスが低下します。

しかし、呼び出しメソッドがパラメータとして受け渡すオブジェクトを変更しない場合は、コピーを作成せずにオブジェクト自体を渡しても安全です。この場合は pass-by-reference の値を true に設定します。

複数の EJB モジュールを含む J2EE アプリケーション全体に pass-by-reference セマンティックを適用させるには、`sun-application.xml` ファイルに同じ要素を設定します。Bean レベルとアプリケーションレベルの両方で pass-by-reference を使用する場合は、アプリケーションレベルより Bean レベルが優先されます。

`server.xml` ファイルの詳細については、『Sun ONE Application Server 開発者ガイド』および『管理者用設定ファイルリファレンス』を参照してください。

サブ要素

なし

res-ref-name

対応する J2EE `ejb-jar.xml` ファイルの `resource-ref` エントリ内の `res-ref-name` を指定します。`res-ref-name` 要素は、リソースマネージャの接続ファクトリへの参照を指定します。名前は、java:comp/env コンテキストに関連する JNDI 名です。この名前は、同じ Enterprise JavaBean 内では一意にする必要があります。

サブ要素

なし

例

```
<res-ref-name>jdbc/SimpleBank</res-ref-name>
```

resource-env-ref

対応する J2EE ejb-jar.xml ファイルの resource-env-ref エントリに記録されている resource-env-ref-name を、server.xml ファイルの resources 要素に記録されている絶対名 jndi-name にマップします。resource-env-ref 要素には、Bean の環境で利用されるリソースに関連する管理対象オブジェクトに対する Enterprise JavaBean の参照宣言が含まれます。

エンティティ Beans、メッセージ駆動型 Beans、セッション Beans で使用されます。

サブ要素

次の表は、resource-env-ref 要素のサブ要素を示します。

resource-env-ref 要素のサブ要素

サブ要素	必要指定数	説明
resource-env-ref-name	1 個のみ	対応する J2EE ejb-jar.xml ファイルの resource-env-ref エントリ内の resource-env-ref-name を指定する
jndi-name	1 個のみ	JNDI 絶対名 jndi-name を指定する

例

```
<resource-env-ref>
   <resource-env-ref-name>
   jms/StockQueueName
   </resource-env-ref-name>
   <jndi-name>jms/StockQueue</jndi-name>
</resource-env-ref>
```

resource-env-ref-name

対応する J2EE ejb-jar.xml ファイルの resource-env-ref エントリ内の resource-ref-name を指定します。resource-env-ref-name 要素は、リソース環境への参照を指定します。この値は、EJB コードで使用される環境エントリ名です。名前は java:comp/env コンテクストに関連する JNDI 名で、同じ Enterprise JavaBean では一意にする必要があります。

サブ要素

なし

例

```
<resource-env-ref-name>jms/StockQueue</resource-env-ref-name>
```

resource-ref

対応する J2EE ejb-jar.xml ファイルの resource-ref エントリに記録されている res-ref-name を、server.XML ファイルの resources 要素に記録されている絶対名 jndi-name にマップします。resource-ref 要素には、外部リソースに対する EJB の参照宣言が含まれます。

注 JMS 接続ファクトリから得られる接続は、Sun ONE Application Server の現在のリリースでは共有できません。ejb-jar.xml ファイルの resource-ref エントリ内の res-sharing-scope 要素は、JMS 接続ファクトリでは無視されます。

エンティティ Beans、メッセージ駆動型 Beans、セッション Beans で使用されます。

注 resource-ref が Sun ONE Message Queue の JMS 接続ファクトリを指定する場合は、Sun ONE Message Queue のユーザーレポジトリに default-resource-principal (名前 / パスワード) が存在する必要があります。Sun ONE Message Queue のユーザーレポジトリを管理する方法については、『Sun ONE Message Queue 管理者ガイド』のセキュリティ管理に関する章を参照してください。

サブ要素

次の表は、resource-ref 要素のサブ要素を示します。

resource-ref 要素のサブ要素

サブ要素	必要指定数	説明
res-ref-name	1 個のみ	対応する J2EE ejb-jar.xml ファイルの resource-ref エントリ内の res-ref-name を指定する
jndi-name	1 個のみ	JNDI 絶対名 jndi-name を指定する
default-resource-principal	0 または 1 個	リソースマネージャへのデフォルトのサインオン情報 (名前 / パスワード) を指定する

例

```
<resource-ref>
   <res-ref-name>jdbc/EmployeeDBName</res-ref-name>
   <jndi-name>jdbc/EmployeeDB</jndi-name>
</resource-ref>
```

メッセージング要素

ここでは、次のメッセージング関連要素について説明します。

- jms-durable-subscription-name
- jms-max-messages-load
- mdb-connection-factory

jms-durable-subscription-name

メッセージ駆動型 Bean クラスに関連付けられた永続サブスクリプションを指定します。送信先タイプが Java Message Service Topic で、メッセージ駆動型 Bean 配備記述子のサブスクリプションの永続性が永続 (Durable) の場合にだけ適用されます。

サブ要素

なし

jms-max-messages-load

メッセージ駆動型 Bean による処理で Java Message Service セッションに一度に読み込む最大メッセージ数を指定します。デフォルト値は 1 です。

サブ要素

なし

mdb-connection-factory

メッセージ駆動型 Bean クラスに関連付けられたコネクションファクトリを指定します。キューまたはトピックのタイプは、メッセージ駆動型 Bean クラスに関連付けられた Java Message Service の送信先タイプと一致している必要があります。

サブ要素

次の表は、`mdb-connection-factory` 要素のサブ要素を示します。

`mdb-connection-factory` 要素のサブ要素

サブ要素	必要指定数	説明
`jndi-name`	1 個のみ	JNDI 絶対名 `jndi-name` を指定する
`default-resource-principal`	0 または 1 個	リソースマネージャへのデフォルトのサインオン情報 (名前 / パスワード) を指定する

セキュリティ要素

ここでは、認証、承認、および一般的なセキュリティに関する要素について説明します。次の要素があります。

- as-context
- auth-method
- caller-propagation
- confidentiality
- default-resource-principal
- establish-trust-in-client
- establish-trust-in-target
- integrity
- ior-security-config
- name
- password
- realm
- required
- sas-context
- transport-config

as-context

クライアントの認証に使用する認証メカニズムを指定します。指定する場合は、USERNAME_PASSWORD となります。

サブ要素

次の表は、as-context 要素のサブ要素を示します。

as-context 要素のサブ要素

サブ要素	必要指定数	説明
auth-method	1 個のみ	認証方法を指定する。サポートされている値は USERNAME_PASSWORD のみ
realm	1 個のみ	ユーザーを認証するレルムを指定する

as-context 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
required	1 個のみ	指定された認証方法をクライアント認証で使用する必要があるかどうかを指定する。その必要がある場合、as-context の target_requires フィールドに EstablishTrustInClient ビットが設定される。値は true または false のいずれか

auth-method

認証方法を指定します。サポートされている値は USERNAME_PASSWORD のみです。

サブ要素

なし

caller-propagation

伝達された呼び出し元 ID をターゲットが受け入れるかどうかを指定します。値は NONE、SUPPORTED、または REQUIRED です。

サブ要素

なし

confidentiality

プライバシー保護されたメッセージをターゲットがサポートするかどうかを指定します。値は NONE、SUPPORTED、または REQUIRED です。

サブ要素

なし

default-resource-principal

リソースマネージャへのデフォルトのサインオン情報 (名前 / パスワード) を指定します。

サブ要素

次の表は、default-resource-principal 要素のサブ要素を示します。

default-resource-principal 要素のサブ要素

サブ要素	必要指定数	説明
name	1 個のみ	リソースマネージャへのサインオンに使用するデフォルトのリソース主体名を指定する
password	1 個のみ	デフォルトのリソース主体のパスワードを指定する

establish-trust-in-client

ターゲットがクライアントを認証できるかどうかを指定します。値は NONE、SUPPORTED、または REQUIRED です。

サブ要素

なし

establish-trust-in-target

ターゲットがクライアントに対して認証できるかどうかを指定する。値は NONE、SUPPORTED、または REQUIRED です。

サブ要素

なし

integrity

整合性が保護されたメッセージをターゲットがサポートするかどうかを指定します。値は NONE、SUPPORTED、または REQUIRED です。

サブ要素

なし

ior-security-config

入出力先の変更 (IOR) のセキュリティ情報を指定します。

サブ要素

次の表は、ior-security-config 要素のサブ要素を示します。

ior-security-config 要素のサブ要素

サブ要素	必要指定数	説明
transport-config	0 または 1 個	トランスポートのセキュリティ情報を指定する
as-context	0 または 1 個	クライアントの認証に使用する認証メカニズムを記述する。指定する場合は、USERNAME_PASSWORD になる
sas-context	0 または 1 個	sas-context フィールドを記述する

name

ID を指定します。

サブ要素

なし

password

セキュリティで認証を実行するために必要なパスワードを指定します。

サブ要素

なし

realm

ユーザーを認証するレルムを指定します。

サブ要素

なし

required

指定された認証方法をクライアント認証で使用する必要があるかどうかを指定します。必要がある場合、as-context の target_requires フィールドに EstablishTrustInClient ビットが設定されます。値は true または false のいずれかです。

サブ要素

なし

sas-context

sas-context フィールドを記述します。

サブ要素

次の表は、sas-context 要素のサブ要素を示します。

sas-context 要素のサブ要素

サブ要素	必要指定数	説明
caller-propagation	1 個のみ	伝達された呼び出し元 ID をターゲットが受け入れるかどうかを指定する。値は NONE、SUPPORTED、または REQUIRED

transport-config

トランスポートのセキュリティ情報を指定します。

サブ要素

次の表は、transport-config 要素のサブ要素を示します。

transport-config 要素のサブ要素

サブ要素	必要指定数	説明
integrity	1 個のみ	整合性が保護されたメッセージをターゲットがサポートするかどうかを指定する。値は NONE、SUPPORTED、または REQUIRED
confidentiality	1 個のみ	プライバシー保護されたメッセージをターゲットがサポートするかどうかを指定する。値は NONE、SUPPORTED、または REQUIRED
establish-trust-in-target	1 個のみ	ターゲットがクライアントに対して認証できるかどうかを指定する。値は NONE、SUPPORTED、または REQUIRED
establish-trust-in-client	1 個のみ	ターゲットがクライアントを認証できるかどうかを指定する。値は NONE、SUPPORTED、または REQUIRED

持続性要素

ここでは、コンテナ管理による持続性 (CMP)、持続性マネージャ、および持続性ベンダーに関する要素について説明します。これらの要素に関する詳細は、86 ページの「コンテナ管理による持続性の使用」を参照してください。

次の要素があります。

- cmp
- cmp-resource
- concrete-impl
- finder
- is-one-one-cmp
- mapping-properties
- method-name
- one-one-finders
- pc-class
- pm-class-generator
- pm-config
- pm-descriptor
- pm-descriptors
- pm-identifier
- pm-inuse
- pm-mapping-factory
- pm-version
- query-filter
- query-params
- query-variables

cmp

EJB1.1 および EJB2.0 Bean の コンテナ管理による持続性 (CMP) エンティティ Bean オブジェクトに関する実行時情報を記述します。これは、Bean のマッピング情報が記述されたファイルへのポインタです。

サブ要素

次の表は、cmp 要素のサブ要素を示します。

cmp 要素のサブ要素

サブ要素	必要指定数	説明
mapping-properties	1 個のみ	持続性ベンダー固有のオブジェクトとリレーショナル間 (O/R) のデータベースマッピングファイルの場所を示すデータを指定する
concrete-impl	1 個のみ	持続性ベンダー固有の具象クラス名の場所を示すデータを指定する
pc-class	0 または 1 個	持続性ベンダー固有のクラスを示すデータを指定する
is-one-one-cmp	0 または 1 個	コンテナ管理持続性 (CMP) 1.1 のブール特性を指定する。古い記述子による CMP 1.1 の識別に使用される
one-one-finders	0 または 1 個	コンテナ管理持続性 (CMP) 1.1 の検索を記述する

cmp-resource

EJB JAR ファイル内のコンテナ管理による持続性 (CMP) Bean を保存するために使用するデータベースを指定します。

サブ要素

次の表は、cmp-resource 要素のサブ要素を示します。

cmp-resource 要素のサブ要素

サブ要素	必要指定数	説明
jndi-name	1 個のみ	JNDI 絶対名 jndi-name を指定する
default-resource-principal	0 または 1 個	リソース参照のデフォルトの実行時バインドを指定する

concrete-impl

持続性ベンダー固有の具象クラス名の場所を指定します。

サブ要素

なし

finder

メソッド名とクエリでコンテナ管理による持続性 1.1 の検索を記述します。

サブ要素

次の表は、`finder` 要素のサブ要素を示します。

`finder` 要素のサブ要素

サブ要素	必要指定数	説明
`method-name`	1 個のみ	クエリフィールドのメソッド名を指定する
`query-params`	1 個のみ	コンテナ管理持続性 (CMP) 1.1 の検索のクエリパラメータを指定するオプションデータ
`query-filter`	1 個のみ	コンテナ管理持続性 (CMP) 1.1 の検索のクエリフィルタを指定する
`query-variables`	1 個のみ	コンテナ管理持続性 1.1 の検索のクエリ式内の変数を指定するオプションデータ

is-one-one-cmp

コンテナ管理による持続性 1.1 のブール特性を指定します。古い記述子による CMP 1.1 の識別に使用されます。

サブ要素

なし

mapping-properties

持続性ベンダー固有のオブジェクトとリレーショナル間 (O/R) のデータベースマッピングファイルの場所を指定します。ほとんどの持続性ベンダーでは、プロジェクトという概念を使用します。プロジェクトは、関連するすべての Beans およびその依存クラスを表し、1 つの単位として配備することができます。プロジェクトに関連付けられたベンダー固有 XML ファイルが存在する場合もあります。

サブ要素

なし

method-name

クエリフィールドのメソッド名を指定します。method-name 要素には、EJB メソッドの名前またはアスタリスク (*) が含まれます。アスタリスクは、EJB のコンポーネントとホームインタフェースのすべてのメソッドを意味します。

例

```
<method-name>create</method-name>
```

```
<method-name>*</method-name>
```

サブ要素

なし

one-one-finders

コンテナ管理による持続性 (CMP) 1.1 の検索を記述します。

サブ要素

次の表は、one-one-finders 要素のサブ要素を示します。

one-one-finders 要素のサブ要素

サブ要素	必要指定数	説明
finder	1 つ以上必要	メソッド名とクエリでコンテナ管理持続性 (CMP) 1.1 の検索を記述する

pc-class

持続性ベンダー固有のクラスを指定します。

サブ要素

なし

pm-class-generator

使用するベンダー固有の具象クラスジェネレータを指定します。これは、そのベンダーに固有のクラス名です。

サブ要素

なし

pm-config

使用するベンダー固有のコンフィグレーションファイルを指定します。

サブ要素

なし

pm-descriptor

エンティティ Beans に関連付けられた持続性マネージャのプロパティを記述します。

サブ要素

次の表は、pm-descriptor 要素のサブ要素を示します。

pm-descriptor 要素のサブ要素

サブ要素	必要指定数	説明
pm-identifier	1 個のみ	持続性マネージャ実装を提供したベンダーを指定する。たとえば、Sun ONE Application Server のコンテナ管理持続性またはサードパーティベンダー
pm-version	1 個のみ	使用する持続性マネージャベンダー製品のバージョンを指定する
pm-config	0 または 1 個	使用するベンダー固有のコンフィグレーションファイルを指定する
pm-config	0 または 1 個	使用するベンダー固有の具象クラスジェネレータを指定する。そのベンダーに固有のクラス名である
pm-mapping-factory	0 または 1 個	使用するベンダー固有のマッピングファクトリを指定する。そのベンダーに固有のクラス名である

pm-descriptors

持続性マネージャ記述子を記述します。特定の時間にその 1 つが使用されている必要があります。これは、基本的に、Sun ONE Application Server のプラグイン可能な持続性マネージャ API に適用されます。

サブ要素

次の表は、pm-descriptors 要素のサブ要素を示します。左側の列にはサブ要素名、中央の列には必要指定数、右側の列には要素の説明を示しています。

pm-descriptors 要素のサブ要素

サブ要素	必要指定数	説明
pm-descriptor	1 個以上	エンティティ Beans に関連付けられた持続性マネージャのプロパティを記述する
pm-inuse	1 個のみ	この特定の持続性マネージャを使用する必要があるかどうかを指定する

pm-identifier

持続性マネージャ実装を提供したベンダーを指定します。たとえば、Sun ONE Application Server のコンテナ管理持続性またはサードパーティベンダーなどです。

サブ要素

なし

pm-inuse

この特定の持続性マネージャを使用する必要があるかどうかを指定します。

サブ要素

次の表は、pm-inuse 要素のサブ要素を示します。

pm-insue 要素のサブ要素

サブ要素	必要指定数	説明
pm-identifier	1 個のみ	持続性マネージャ実装を提供したベンダーを示すデータを指定する。たとえば、Sun ONE Application Server のコンテナ管理持続性またはサードパーティベンダー
pm-version	1 個のみ	使用する持続性マネージャベンダー製品のバージョンを示すデータを指定する

pm-mapping-factory

使用するベンダー固有のマッピングファクトリを指定します。これは、そのベンダーに固有のクラス名です。

サブ要素

なし

pm-version

使用する持続性マネージャベンダー製品のバージョンを指定します。

サブ要素

なし

query-filter

コンテナ管理による持続性 1.1 の検索のクエリフィルタを指定します (省略可能)。

サブ要素

なし

query-params

コンテナ管理による持続性 1.1 の検索のクエリパラメータを指定します。

サブ要素

なし

query-variables

コンテナ管理による持続性 1.1 の検索のクエリ式内の変数を指定します (省略可能)。

サブ要素

なし

プールとキャッシュの要素

この節では、キャッシュ、タイムアウト、および EJB プールに関する要素について説明します。これらの要素は、メモリ管理とパフォーマンスの調整に使用されます。詳細については、『Sun ONE Application Server Performance, Tuning, and Sizing Guide』を参照してください。

次の要素について説明します。

- bean-cache
- bean-pool
- cache-idle-timeout-in-seconds
- commit-option
- is-cache-overflow-allowed
- max-cache-size
- max-pool-size
- max-wait-time-in-millis
- pool-idle-timeout-in-seconds
- removal-timeout-in-seconds
- resize-quantity
- steady-pool-size
- victim-selection-policy

bean-cache

エンティティ Bean のキャッシュプロパティを指定します。エンティティ Beans およびステートフルセッション Beans で使用します。

サブ要素

次の表は、`bean-cache` 要素のサブ要素を示します。

`bean-cache` 要素のサブ要素

サブ要素	必要指定数	説明
`max-cache-size`	0 または 1 個	キャッシュに存在できる Bean の最大数を指定する
`is-cache-overflow-allowed`	0 または 1 個	廃止
`cache-idle-timeout-in-seconds`	0 または 1 個	ステートフルセッション Beans またはエンティティ Beans がアイドル状態でキャッシュ内に存在できる非活性化までの最大時間を指定する。デフォルト値は 10 分 (600 秒)

bean-cache 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
removal-timeout-in-seconds	0 または 1 個	Bean が削除されるまでの存続時間を指定する。removal-timeout-in-seconds を idle-timeout より短く設定すると、Bean は非活性化される前に削除される
resize-quantity	0 または 1 個	プールが空の場合に作成する Bean の数を指定する (max-pool-size による制限を受ける)。値は 0 から MAX_INTEGER の範囲
victim-selection-policy	0 または 1 個	削除対象の選択に使用するアルゴリズムを指定する。ステートフルセッション Beans だけに適用される

例

```
<bean-cache>
   <max-cache-size>100</max-cache-size>
   <cache-resize-quantity>10</cache-resize-quantity>
   <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
   <victim-selection-policy>LRU</victim-selection-policy>
      <cache-idle-timeout-in-seconds>
      600
      </cache-idle-timeout-in-seconds>
   <removal-timeout-in-seconds>5400</removal-timeout-in-seconds>
</bean-cache>
```

bean-pool

ステートフルセッション Beans、エンティティ Beans、メッセージ駆動型 Bean のプールのプロパティを指定します。

サブ要素

次の表は、bean-pool 要素のサブ要素を示します。

bean-pool 要素のサブ要素

サブ要素	必要指定数	説明
steady-pool-size	0 または 1 個	プール内で維持する Bean の初期の最少数を指定する。デフォルト値は 32

bean-pool 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
resize-quantity	0 または 1 個	プールが空の場合に作成する Bean の数を指定する (max-pool-size による制限を受ける)。値は 0 から MAX_INTEGER の範囲
max-pool-size	0 または 1 個	プール内の Bean の最大数を指定する。値は 0 から MAX_INTEGER の範囲デフォルト値は server.xml または 60
max-wait-time-in-millis	0 または 1 個	廃止
pool-idle-timeout-in-seconds	0 または 1 個	Bean がアイドル状態でプール内に存在できる最大時間を指定する。この時間を過ぎると Bean は削除される。この時間はサーバーが参考にする。デフォルト値は 600 秒 (10 分)

例

```
<bean-pool>
   <steady-pool-size>10</steady-pool-size>
   <resize-quantity>10</resize-quantity>
   <max-pool-size>100</max-pool-size>
   <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
</bean-pool>
```

cache-idle-timeout-in-seconds

Bean がキャッシュにアイドル状態のままでいられる最長時間を指定します(省略可能)。この時間を過ぎると、コンテナはこの Bean を非活性化できます。値に 0 を指定すると、その Beans は非活性化の対象となりません。デフォルト値は 600 です。

ステートフルセッション Beans およびエンティティ Beans に適用されます。

サブ要素

なし

commit-option

トランザクション完了時に使用するコミットオプションを指定します (省略可能)。Sun ONE Application Server での有効値は B または C で、デフォルト値は B です。

注	コミットオプション A は、Sun ONE Application Server 7 ではサポートされていません。

この要素はエンティティ Beans に適用されます。

サブ要素

なし

例

```
<commit-option>B</commit-option>
```

is-cache-overflow-allowed

この要素は廃止されたので使用できません。

max-cache-size

キャッシュに存在できる Bean の最大数を指定します (省略可能)。ゼロを指定すると、キャッシュは無制限になります。実際には、強い制限値は存在しません。max-cache-size の制限値は、キャッシュ実装にとって参考程度です。デフォルト値は 512 です。

ステートフルセッション Beans およびエンティティ Beans に適用されます。

サブ要素

なし

例

```
<max-cache-size>100</max-cache-size>
```

max-pool-size

プール内に存在できる Bean インスタンスの最大数を指定します (省略可能)。0 (メッセージ駆動型 Bean では 1) から MAX_INTEGER までの範囲で指定します。0 を指定すると、プールは無制限になります。デフォルト値は 64 です。

すべての Beans に適用されます。

サブ要素

なし

例

```
<max-pool-size>100</max-pool-size>
```

max-wait-time-in-millis

この要素は廃止されたので使用できません。

pool-idle-timeout-in-seconds

Beans インスタンスが、アイドル状態でプール内に存在できる最大時間を秒単位で指定します (省略可能)。このタイムアウト時間が過ぎると、プール内の Bean インスタンスは非活性化または削除の対象となります。この時間はサーバーが参考にします。0 を指定すると、アイドル Beans はプール内に残り続けます。デフォルト値は 600 です。

ステートレスセッション Beans、エンティティ Beans、メッセージ駆動型 Beans に適用されます。

注 ステートレスセッション Bean とメッセージ駆動型 Bean では、タイムアウト時間が過ぎると、その Bean を削除 (ガベージコレクション) できます。

サブ要素

なし

例

```
<pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
```

removal-timeout-in-seconds

Bean インスタンスが、削除前にアイドル状態でコンテナ内に存在できる最大時間 (タイムアウト) を指定します (省略可能)。0 を指定すると、コンテナは非活性化された Beans を自動削除しません。デフォルト値は 5400 です。

`removal-timeout-in-seconds` を `cache-idle-timeout-in-seconds` より短く、または同じに設定すると、Beans は非活性化される前に削除されます。

ステートフルセッション Beans だけに適用されます。

関連情報については、`cache-idle-timeout-in-seconds` を参照してください。

サブ要素

なし

例

`<removal-timeout-in-seconds>3600</removal-timeout-in-seconds>`

resize-quantity

次の Bean インスタンスの数を指定します (省略可能)。

- プールに存在する Beans が steady-pool-size より少ない状態で要求を受け取った場合に作成する Beans (作成時にプールだけに適用される)。プールに存在する Beans の数が、steady-pool-size から resize-quantity を差し引いた値より多い場合は、resize-quantity と同数の Beans が作成される
- pool-idle-timeout-in-seconds で指定した時間が経過したときに (タイムアウト)、クリーナスレッドが削除する使用されていない Beans
    - キャッシュでは、max-cache-size に達すると、victim-selection-policy を使って非活性化の対象とする Beans が resize-quantity と同数だけ選択される。また、cache-idle-timeout-in-seconds または cache-remove-timeout-in-seconds が経過すると、非活性化された Beans がキャッシュから削除される
    - プールでは、Beans の数が max-pool-size に達すると、削除の対象とする Beans が resize-quantity と同数だけ選択される。また、pool-idle-timeout-in-seconds が経過すると、Beans の数が steady-pool-size に達するまで Beans は削除される

値は 0 から MAX_INTEGER の範囲です。プールのサイズが steady-pool-size を下回ることはありません。デフォルトは 16 です。

ステートレスセッション Beans、エンティティ Beans、メッセージ駆動型 Beans に適用されます。

EJB プールでは、server.xml ファイルの pool-resize-quantity に記録される ejb-container 要素の値がデフォルト値となります。デフォルト値は 16 です。

EJB キャッシュでは、server.xml ファイルの cache-resize-quantity に記録される ejb-container 要素の値がデフォルト値となります。デフォルト値は 32 です。

メッセージ駆動型 Beans では、server.xml ファイルの pool-resize-quantity に記録される mdb-container 要素の値がデフォルト値となります。デフォルト値は 2 です。

サブ要素

なし

例

```
<resize-quantity>10</resize-quantity>
```

steady-pool-size

プール内で維持する Bean インスタンスの初期の最少数を指定します (省略可能)。デフォルト値は 32 です。

注 — `steady-pool-size` に 0 より大きな値を設定すると、サーバーの起動時に Beans が作成されます。サーバー起動時に `setInitialContext` メソッドで有効でない情報、たとえばユーザーのセキュリティロールなどに依存する Bean は `setInitialContext` メソッド内で `EJBException` をスローする必要があります。コンテナがこの例外を処理し、Beans をインスタンス化しません。Bean がこの例外を受け入れてしまう場合は、`sun-ejb-jar.xml` ファイルで `steady-pool-size` を 0 に設定する必要があります。

ステートレスセッション Beans およびメッセージ駆動型 Beans に適用されます。

サブ要素

なし

例

```
<steady-pool-size>10</steady-pool-size>
```

victim-selection-policy

非活性化するステートフルセッション Beans を選択する方法を指定します (省略可能)。可能な値には、先入れ先出し (FIFO)、最近もっとも使われていないもの (LRU)、最近使われていないもの (NRU) があります。デフォルト値は NRU (実際には pseudo-LRU) です。

注 — ユーザー独自の削除対象選択アルゴリズムを組み込むことはできません。

削除対象は、通常、非活性化されてバックアップストア (一般的にはファイルシステムまたはデータベース) に置かれます。このストアの内容は起動時に削除されます。また、`removal-timeout-in-seconds` が経過すると、定期的なバックグラウンドプロセスによってアイドルエントリが削除されます。バックアップストアはバックグラウンドスレッド (またはスイーパースレッド) によって管理され、不要なエントリは削除されます。

ステートフルセッション Beans だけに適用されます。

サブ要素

なし

例

```
<victim-selection-policy>LRU</victim-selection-policy>
```

クラス要素

ここでは、クラスに関連する要素について説明します。次の要素があります。

- gen-classes
- local-home-impl
- local-impl
- remote-home-impl
- remote-impl

gen-classes

Bean で生成されるすべてのクラス名を指定します。

注	これは、配備時または再配備時にサーバーによって自動的に生成されます。開発者が指定したり、配備後に変更したりする必要はありません。

サブ要素

次の表は、gen-classes 要素のサブ要素を示します。

gen-classes 要素のサブ要素

サブ要素	必要指定数	説明
remote-impl	0 または 1 個	生成された EJBObject impl クラスの完全修飾クラス名を指定する
local-impl	0 または 1 個	生成された EJBLocalObject impl クラスの完全修飾クラス名を指定する
remote-home-impl	0 または 1 個	生成された EJBHome impl impl クラスの完全修飾クラス名を指定する

gen-classes 要素のサブ要素 (続き)

サブ要素	必要指定数	説明
local-home-impl	0 または 1 個	生成された EJBLocalHome impl クラスの完全修飾クラス名を指定する

local-home-impl

生成された EJBLocalHome impl クラスの完全修飾クラス名を指定します。

注 — これは、配備時または再配備時にサーバーによって自動的に生成されます。開発者が指定したり、配備後に変更したりする必要はありません。

サブ要素

なし

local-impl

生成された EJBLocalObject impl クラスの完全修飾クラス名を指定します。

注 — これは、配備時または再配備時にサーバーによって自動的に生成されます。開発者が指定したり、配備後に変更したりする必要はありません。

サブ要素

なし

remote-home-impl

生成された EJBHome impl クラスの完全修飾クラス名を指定します。

注 — これは、配備時または再配備時にサーバーによって自動的に生成されます。開発者が指定したり、配備後に変更したりする必要はありません。

サブ要素

なし

remote-impl

生成された EJBObject impl クラスの完全修飾クラス名を指定します。

注 　これは、配備時または再配備時にサーバーによって自動的に生成されます。開発者が指定したり、配備後に変更したりする必要はありません。

サブ要素

なし

EJB XML ファイルの例

この節では、次のファイルの例を示します。

- ejb-jar.xml ファイルの例
- sun-ejb-jar.xml ファイルの例

Enterprise JavaBean に関連する要素については、176 ページの「sun-ejb-jar.xml ファイルの要素」および『Sun ONE Application Server 開発者ガイド』を参照してください。

ejb-jar.xml ファイルの例

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN'

'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
   <description>no description</description>
   <display-name>CustomerJAR</display-name>
      <enterprise-beans>
         <entity>
            <description>no description</description>
            <display-name>CustomerEJB</display-name>
            <ejb-name>CustomerEJB</ejb-name>
            <home>samples.SimpleBankBMP.ejb.CustomerHome</home>
            <remote>samples.SimpleBankBMP.ejb.Customer</remote>
            <ejb-class>samples.SimpleBankBMP.ejb.CustomerEJB</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>java.lang.String</prim-key-class>
            <reentrant>False</reentrant>
            <security-identity>
               <description></description>
               <use-caller-identity></use-caller-identity>
            </security-identity>
            <resource-ref>
               <res-ref-name>jdbc/SimpleBank</res-ref-name>
               <res-type>javax.sql.DataSource</res-type>
               <res-auth>Container</res-auth>
               <res-sharing-scope>Shareable</res-sharing-scope>
            </resource-ref>
         </entity>
      </enterprise-beans
```

```
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>CustomerEJB</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>NotSupported</trans-attribute>
      </container-transaction>
   </assembly-descriptor>
</ejb-jar>
```

sun-ejb-jar.xml ファイルの例

これらの要素に関する詳細は、176 ページの「sun-ejb-jar.xml ファイルの要素」を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sun-ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Sun ONE Application
Server 7.0 EJB 2.0//EN'

'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-0.dtd'>

<sun-ejb-jar>
   <display-name>First Module</display-name>
   <enterprise-beans>
      <ejb>
         <ejb-name>CustomerEJB</ejb-name>
         <jndi-name>customer</jndi-name>
         <resource-ref>
            <res-ref-name>jdbc/SimpleBank</res-ref-name>
'             <jndi-name>jdbc/PointBase</jndi-name>
         </resource-ref>
         <is-read-only-bean>false</is-read-only-bean>
         <commit-option>B</commit-option>
         <bean-pool>
            <steady-pool-size>10</steady-pool-size>
            <resize-quantity>10</resize-quantity>
            <max-pool-size>100</max-pool-size>
            <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
         </bean-pool>
         <bean-cache>
            <max-cache-size>100</max-cache-size>
            <resize-quantity>10</resize-quantity>
               <removal-timeout-in-seconds>3600</removal-timeout-in-seconds>
            <victim-selection-policy>LRU</victim-selection-policy>
```

```
        </bean-cache>
      </ejb>
    </enterprise-beans>
</sun-ejb-jar>
```

付録 A

Sun ONE Studio 4 インタフェースによる CMP のマッピング

この章では、Sun ONE Studio 4 インタフェースを使って、Java プログラミング言語のクラスセットとリレーショナルデータベースをマップする方法を示します。

この章には次の項目があります。

- CMP Beans のマッピング
- EJB 持続性プロパティ

ここに示す手順を行うには、79 ページの「エンティティ Beans のコンテナ管理による持続性の使用」、および『Enterprise JavaBeans Specification, v2.0』の第 10 章の内容を十分に理解している必要があります。

CMP Beans のマッピング

コンテナ管理による持続性 Beans をマップするには、まず最初にスキーマを取り込み、次にそのスキーマに Beans をマップする必要があります。

この節には次の項があります。

- スキーマの取り込み
- 既存の Enterprise JavaBeans とスキーマとのマッピング

スキーマの取り込み

Enterprise JavaBean をデータベーススキーマにマップする前に、スキーマを取り込んでファイルシステムに作業用コピーを作成する必要があります。これにより、データベース自体に影響を及ぼさずに、開発者は作業を行うことができます。

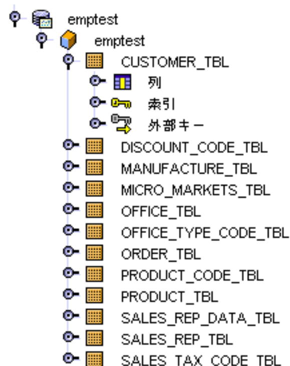
注 取り込んだスキーマをパッケージ内に保存することをお勧めします。スキーマを格納するパッケージがない場合は、ファイルシステム上で右クリックして「新規」-> 「Java パッケージ」を選択します。

1. マッピングツールを表示します。次の 3 つの方法があります。
   - ファイルシステム上で右クリックして、「新規」> 「データベース」> 「データベーススキーマ」を選択する
   - 「ファイル」メニューから「新規」を選択し、「テンプレートを選択」で「データベース」をダブルクリックして、「データベーススキーマ」を選択する
   - 「ツール」メニューから「データスキーマの収集」を選択する
2. 「作成場所」ペインで、スキーマの作業用コピーのファイル名を入力し、次に、取り込んだスキーマのパッケージを選択します。
3. 「データベース接続」ペインで、確立済みの接続がある場合は、「既存の接続」メニューからその接続を選択します。既存の接続がない場合は、「新規接続」で次の情報を入力します。
   - 接続するデータベースの名前。ドロップダウンメニューに目的のデータベースが表示されない場合は、マッピングツールを一旦終了して IDE にドライバをインストールしてから作業を続ける必要がある
   - システムの JDBC ドライバ
   - データベースの JDBC URL。これは、ドライバ識別子、サーバー、ポート、およびデータベース名で構成される。たとえば、`jdbc:pointbase://localhost:9092/sample` のようになる

     JDBC URL の形式は、使用するデータベース管理システム (DBMS) の種類 (Oracle、Microsoft SQL Server、PointBase など)、およびその DBMS のバージョンによって異なる。使用する DBMS の正しい URL 形式については、システム管理者に確認する。
   - データベースのユーザー名
   - そのユーザーのパスワード
4. 「表とビュー」ペインで、取り込む表とビューを選択して、「完了」をクリックします。

注 1 つの表を選択し、外部キーによって取り込まれた表に参照される別の表を除外する場合は、1 つだけ指定した場合でも、両方の表が取り込まれます。

データベースとそのスキーマが、次の図に示すように表示されます。



既存の Enterprise JavaBeans とスキーマとのマッピング

ここでは、コンテナ管理による持続性を使ってマッピングをカスタマイズする、または既存のオブジェクトモデルのマッピングを作成する方法について説明します。

Enterprise JavaBean をデータベーススキーマにマップするには、そのデータベーススキーマを取り込んでエクスプローラのファイルシステムにマウントする必要があります。この手順については、217 ページの「スキーマの取り込み」を参照してください。

「プロパティ」ウィンドウで個々のプロパティを編集することによって、マッピングを部分的に設定または編集することができます。「プロパティ」ウィンドウから、マッピングおよび持続性に関するすべての情報にアクセスできます。マッピングフィールドのプロパティエディタを利用すると、クラスやフィールドのグループを一度に表示および編集できます。さらに、作成したマッピングモデルの有用な概要を表示することもできます。

1. 「ファイルシステム」の下にある「EJB モジュール」を開きます。

   そのモジュールに含まれる Enterprise JavaBeans が表示されます。

2. 開いている EJB モジュールから Enterprise JavaBean を選択します。

   その Enterprise JavaBean のプロパティ表が表示されます。

3. 準備作業が完了している場合は、「次へ」をクリックして、マッピングツールの「表を選択」ペインを表示します。

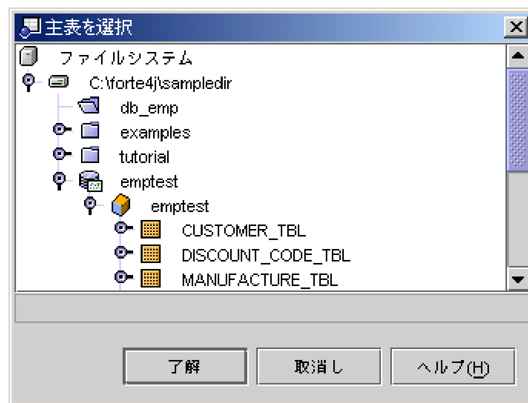
   準備作業がまだ完了していない場合は「取消し」をクリックし、その作業を完了してから、マッピングツールを再起動します。

4. 「主表」コンボボックスから主表を選択し、「ブラウズ」をクリックして「主表を選択」ダイアログを開きます。

5. 「主表を選択」ダイアログを開いたら、スキーマを探して展開し、そのスキーマの表を探します。

6. 表を選択して「了解」をクリックします。

   主表として選択する表は、マップするクラスにもっとも緊密に一致する表である必要があります。



7. 主表を設定したあとは、「追加」をクリックすることによって、1 つまたは複数の二次表をマップできます。

   この場合は、「マップされた二次表の設定」ダイアログボックスが開きます。

   二次表を追加すると、Enterprise JavaBean 内のフィールドを主表以外のカラムにマップできます。たとえば、Employee クラスに department name を含めるために、DEPARTMENT 表を二次表として追加することができます。関係では、関係フィールドによって 1 つのクラスを別のクラスに関係付けます。二次表のマッピ

ングでは、同じクラス内のフィールドを、2 つの異なる表にマップします。二次表を追加すると、フィールドを主表以外のカラムに直接マップできます。このペインでは、二次表を選択するほかに、二次表が主表にどのようにリンクされているかを表示することができます。

二次表は、1 つ以上のカラムによって主表に関係付けられている必要があり、関係付けられたカラムの行は両方の表で同じ値を保有します。通常、これは両表間の外部キーとして定義されます。ドロップダウンメニューから二次表を選択するときに、2 つの表間の外部キーがあるかどうかがマッピングツールによって確認されます。外部キーが存在する場合は、それがデフォルトの参照キーとして表示されます。

a. コンボボックスから二次表を選択します。

   二次表を選択すると、コンテナ管理による持続性の実装によって、主表と二次表の間に外部キーが存在するかどうかが確認されます。存在する場合は、その外部キーがデフォルトの参照キーとして表示されます。外部キーが存在しない場合は、「列を選択」が表示され、参照キーを設定する必要があります。

b. 参照キーを設定するには、「列を選択」をクリックして、ドロップダウンメニューからカラムを選択します。

   主カラムを選択すると、二次カラムの選択肢が、互換性のあるタイプのカラムに限定されます。互換性のあるカラムが存在しない場合は、フィールドに「互換カラムがありません」と表示されます。二次カラムと互換性のない主カラムを選択すると、二次カラムの値が「列を選択」に戻ります。

注 論理的に関連するとみなされるカラムのペアがなく、論理的な参照キーを設定できない場合は、二次表の選択を再考することをお勧めします。

「ペアを追加」キーを選択すると、2 組以上のカラムペアを使って複雑なキーを設定することができます。

8. 「了解」をクリックして、選択した内容を保存します。

9. マッピングツールで「次へ」をクリックして、「Sun ONE AS マッピング」パネルを表示します。

   「Sun ONE AS マッピング」パネルには、Enterprise JavaBean のすべての持続性フィールドと、そのマッピング状態が表示されます。フィールドをカラムにマップするには、そのフィールドのドロップダウンメニューでカラムを選択します。また、マップされていないすべてのフィールドをマップするには、「自動マップ」を選択します。「自動マップ」では、関係フィールドおよびすでにマップされているフィールドは無視して、もっとも論理的な選択が行われます。既存のマッピングはそのまま変更されません。

   設定可能でない別の表のカラムにフィールドをマップしたい場合は、「戻る」をクリックしてマッピングツールの前のページに戻り、目的のカラムが含まれている二次表を追加します。

   1 つまたは複数のフィールドを選択して、マッピングを解除することができます。一度に複数のフィールドのマッピングを解除するには、Shift キーまたは Ctrl キーを押しながら、解除するフィールドを選択します。1 つのフィールドのマッピングを解除する場合は、そのフィールドのドロップダウンメニューで「アンマップ」を選択します。

   a. 1 つのフィールドを複数のカラムにマップするには、「Sun ONE AS マッピング」ペインで目的のフィールドの省略符号ボタン (「...」) をクリックして、フィールドを複数カラムにマップするダイアログボックスを表示します。

      このダイアログボックスで、マッピングカラムのリストにカラムを追加します。追加するカラムは、このクラスにマップした表のカラムです。カラムの順序を変更するには、「上へ移動」および「下へ移動」を使用します。

      マップしたいカラムが表示されない場合は、マッピングに二次表を追加するか、選択した主表を変更する必要があります。カラムが 1 つも表示されない場合は、主表をまだマップしていないか、カラムのない表をマップしています。

      1 つのフィールドを複数のカラムにマップすると、すべてのカラムが、表示された最初のカラムの値で更新されます。したがって、それらのカラムの 1 つの値がコンテナ管理による持続性アプリケーション以外で変更されると、その値が読み取られるのは、その変更が最初のカラムに対して行われた場合だけです。データベースに値を書き込むと、ほかのカラムに加えられた矛盾する変更があればすべて上書きされます。

      また、それらのカラムのいずれかに複数のフィールドをマップする場合は、マッピングを部分的に重複させることはできないことを確認する必要があります。次の 3 つのケースについて検討してみましょう。

      - フィールド A はカラム A と B にマップされ、フィールド B はカラム B にマップされている場合。このマッピングは一部分のみが重複しているので、この例ではコンパイル時に妥当性検証エラーが発生する

- フィールド A がカラム A にマップされ、フィールド B がカラム B にマップされている場合。一意なので、このマッピングは許容される
- フィールド A がカラム A と B にマップされ、フィールド B がカラム A と B にマップされている場合。マッピングが完全に重複しているので、このマッピングは許容される

b. 「了解」をクリックして、マッピングを保存します。

関係フィールドのマッピング

データベースの表間に外部キーがある場合、通常は、Java クラス参照でそれらの関係を保持する必要があります。CMR フィールドのマッピングでは、クラス参照フィールドに対応する関係を指定できます。

1. 関係フィールドをマップするには、「Sun ONE AS マッピング」パネルで、関係フィールドのドロップダウンメニューの隣にある省略符号ボタン (「...」) をクリックして、「関連マッピング」エディタを表示します。

マッピングツールの外部で「関連マッピング」エディタを使用するには、Explorer で関係フィールドをクリックして、そのマッピングプロパティを編集します。

a. このペインで、「関連クラス」が設定されていることを確認する。関連するクラスが設定されていない場合は、これを設定する。選択したいクラスが持続性に対応していない場合は、エディタをキャンセルして終了し、そのクラスを持続性対応に変換してからエディタに戻る必要がある

b. 「関連フィールド」が適切であること (存在する場合)、および関連するクラスの主表が設定されていることを確認する

注 論理的に関連するフィールドがある場合は、主表を選択する必要があります。これにより管理関係が作成されます。

---

c. 表を直接リンクするか、結合表を使うかを選択する

2. 関係が一対一または一対多の場合は、表を直接リンクする方法を選択します。「次へ」をクリックして、「関連マッピング」エディタの「キーにマップ」ペインを開きます。

   このペインには、次の内容が表示されます。

   ○ 既存のマッピングが 1 つあり、最初の設定ページで何も変更がなかった場合は、既存のマッピング

- 既存のマッピングが存在しないか、またはすでに無効になっている場合は、デフォルトのマッピング

  既存の外部キーに基づいて、2 つの関連クラスでもっとも論理的なキーカラムペアが判別される。外部キーが存在しない場合は、ローカルカラムと外部カラムを選択してキーカラムペアを作成する必要がある。各ペアのカラムは同じ値を保有することが求められる

  複雑なキーを作成するには、「ペアを追加」ボタンを使って「キーカラムペア」を追加する。

  「完了」ボタンが使用不可の場合は、「キーと列のペア」を選択する必要がある。

3. 関係が多対多の場合は、結合表によって表をリンクします。「次へ」をクリックして、「キーにマップ : ローカル列から結合表へ」ペインを開きます。

   このペインには、次の内容が表示されます。

   - 関係内の最初のクラスとフィールド
   - フィールド間の関係の作成に使用する結合表
   - フィールド結合表と、関連クラスをマップする表の間のキーカラムペア

     このペインでは、結合表を選択し、関係フィールドをキーにマップする。これは、「このクラス」をマップする表と結合表の間の唯一の関係である。結合表がない場合は、前のパネルに戻って「マップされている表を直接リンク」を選択する

     クラスをマップする 2 つの表間の結合表を選択する。結合表と、「このクラス」をマップする表の間のもっとも論理的なキーカラムペアが判別される

     表間に外部キーがある場合は、その外部キーがデフォルトのキーカラムペアとして使用される。外部キーがない場合は、結合表から「このクラス」をマップする表に移動できるカラムペアを選択して、キーを作成する必要がある。各ペアのカラムは同じ値を保有することが求められる

     複合キーを作成するには、「ペアを追加」ボタンを使って「キーと列のペア」を追加する

     「次へ」ボタンが使用不可の場合は、結合表を選択するか、両方の側にカラムを持つキーカラムペアが 1 つ以上存在していることを確認する必要がある

4. 「次へ」をクリックして、「キーにマップ : 結合表から外部列へ」ペインを開きます。

   このペインでは、前のペインで選択した結合表に 2 番目の表を関係付けます。

   結合表と、「関連クラス」をマップする表の間のもっとも論理的なキーカラムペアが判別されます。

   表間の外部キーがある場合は、その外部キーがデフォルトのキーカラムペアとして使用されます。外部キーがない場合は、結合表から「関連クラス」をマップする表に移動できるカラムペアを選択して、キーを作成する必要があります。各ペアのカラムは同じ値を保有することが求められます。

   複合キーを作成するには、「ペアを追加」ボタンを使ってキーカラムペアを追加します。

   「完了」ボタンが使用不可の場合は、有効なキーカラムペアを選択する必要があります。

5. 「完了」をクリックして、マッピングツールの「Sun ONE AS マッピング」ペインに戻ります。

6. 「完了」をクリックして「Sun ONE AS マッピング」ペインを閉じ、データベーススキーマに Java クラスをマップします。

EJB 持続性プロパティ

コンテナ管理による持続性を使用する Enterprise JavaBean には、マッピングツールの外部で指定できる固有のプロパティがいくつかあります。

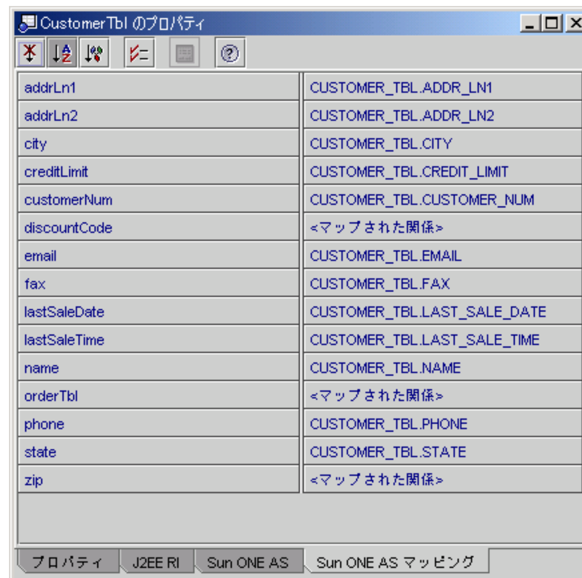
それらの固有プロパティを次の表に示します。

CMP Enterprise JavaBean のプロパティ

プロパティ	説明
マップされた主表	持続性対応クラスに対して選択する主表は、そのクラスともっとも緊密に一致するスキーマの表にする必要がある。持続性対応クラスをマップするには、主表を指定する必要がある。この手順については 219 ページの「既存の Enterprise JavaBeans とスキーマとのマッピング」を参照
マップされたスキーマ	持続性対応クラスをマップする表が属するスキーマ。主表、二次表、および関連クラスはこのスキーマに属している必要がある。217 ページの「スキーマの取り込み」で説明したように、スキーマを取り込むまでは、この設定を行うことができない
マップされた二次表	二次表を追加すると、主表以外のカラムをクラスフィールドにマップできる。たとえば、`Employee` クラスに department name を含めるために、`DEPARTMENT` 表を二次表として追加することができる。複数の二次表を追加できるが、二次表は必須ではない。このプロパティは、マップされた主表が設定されている場合のみ使用できる。二次表の追加については 97 ページおよび 221 ページを参照
整合性のレベル	Bean 内のデータに関するトランザクションの整合性を保証する際のコンテナの動作を指定する。整合性検査フラグ要素が存在しない場合は、`none` とみなされる。整合性レベルの詳細については、115 ページの「consistency」を参照
フェッチグループ	`fetched-with` プロパティは、フィールドおよび関係のフェッチグループ設定を指定する。フィールドは、階層フェッチグループまたは独立フェッチグループに属することができる。`fetched-with` 要素が存在しない場合は、次の値が適用される。`<fetched-with><none/></fetched-with>`。詳細は、117 ページの「fetched-with」を参照

クラスのマッピングを解除するには、「マップされた主表プロパティ」のドロップダウンメニューから <unmapped> を選択します。現在マップされているクラスのマッピングを解除するときに、フィールドマッピングまたは二次表が存在する場合は、警告が表示されます。そのクラスのマッピングを解除する場合は「了解」をクリックします。マッピング解除を中止する場合は、「取消し」をクリックしてマッピングステータスの変更を取り消すと、そのクラスがマップされたままになります。

持続性対応クラスのフィールドマッピングプロパティを表示するには、「プロパティ」ウィンドウの一番下にある「Sun ONE AS マッピング」タブをクリックします。



付録 B

要素一覧

この章では、Sun ONE Application Server 7 環境で Enterprise JavaBeans (EJBs) に関連付けられた DTD ファイルの要素をアルファベット順に示します。

この章には次の項目があります。

- sun-ejb-jar_2_0-0.dtd ファイルの要素
- sun-cmp-mapping_1_0.dtd ファイルの要素

sun-ejb-jar_2_0-0.dtd ファイルの要素

これらの要素に関する説明は、176 ページの「sun-ejb-jar.xml ファイルの要素」にあります。

`as-context`

`auth-method`

`bean-cache`

`bean-pool`

`cache-idle-timeout-in-seconds`

`caller-propagation`

`cmp`

`cmp-resource`

`commit-option`

`concrete-impl`

`confidentiality`

`default-resource-principal`

ejb

ejb-name

ejb-ref

ejb-ref-name

enterprise-beans

establish-trust-in-client

establish-trust-in-target

finder

gen-classes

group-name

integrity

ior-security-config

is-cache-overflow-allowed

is-one-one-cmp

is-read-only-bean

jms-durable-subscription-name

jms-max-messages-load

jndi-name

local-home-impl

local-impl

mapping-properties

max-cache-size

max-pool-size

max-wait-time-in-millis

mdb-connection-factory

method-name

name

one-one-finders

pass-by-reference

password

pc-class

pm-class-generator

pm-config

pm-descriptor

pm-descriptors

pm-identifier

pm-inuse

pm-mapping-factory

pm-version

pool-idle-timeout-in-seconds

principal

principal-name

query-filter

query-params

query-variables

realm

refresh-period-in-seconds

remote-home-impl

remote-impl

removal-timeout-in-seconds

required

res-ref-name

resize-quantity

resource-env-ref

resource-env-ref-name

resource-ref

role-name

sas-context

security-role-mappingserver-name

steady-pool-size

sun-ejb-jar

transport-config

unique-id

victim-selection-policy

sun-cmp-mapping_1_0.dtd ファイルの要素

これらの要素に関する説明は、94 ページの「フィールドおよび関係のマッピング」および 111 ページの「sun-cmp-mappings.xml ファイルの要素」にあります。

check-all-at-commit

check-modified-at-commit

cmp-field-mapping

cmr-field-mapping

cmr-field-name

column-name

column-pair

consistency

ejb-name

entity-mapping

fetched-with

field-name

level

lock-when-loaded

lock-when-modified

named-group

none

read-only

schema

secondary-table

sun-cmp-mapping

sun-cmp-mappings

table-name

索引

A

afterBegin, 155
afterCompletion, 155
allow-concurrent-access 要素 , 78
ANYONE ロール , 168
as-context 要素 , 191
auth-method 要素 , 192

B

bean-cache 要素 , 203
bean-pool 要素 , 204
Bean 管理セキュリティ , 167
Bean 管理トランザクション , 147, 156
  禁止されているメソッド , 158
  コミットなしの復帰 , 157
Bean 管理による持続性 , 58
  Bean クラス定義 , 69
  概要 , 33
Bean クラス定義
  BMP エンティティ Beans の Bean クラス定義の作成 , 69
  セッション Beans のクラス定義の作成 , 51
beforeCompletion, 155
BLOB のサポート , 92

C

caller-propagation 要素 , 192
check-all-at-commit 要素 , 112
check-modified-at-commit 要素 , 112
cmp-field-mapping 要素 , 99, 113
cmp-impl 要素 , 198
cmp-resource, 104
cmp-resource 要素 , 197
CMP でのサードパーティのサポート , 109
cmp 要素 , 197
cmr-field-mapping 要素 , 113
cmr-field-name 要素 , 102, 114
CMR フィールド , 223
column-name 要素 , 100, 103, 114
column-pair 要素 , 102, 114
commit-option 要素 , 205
confidentiality 要素 , 192
consistency 要素 , 98, 115, 226

D

default-resource-principal 要素 , 192
DTD のロールマッピング要素 , 183
DTD ファイル
  一般的な要素 , 176
  キャッシュ要素 , 202

クラス要素，210
参照要素，185
持続性要素，196
セキュリティ要素，191
プール要素，202
メッセージング要素，190
要素，176
ロールマッピング要素，183
DTD ファイルの一般的な要素，176
DTD ファイルのキャッシュ要素，202
DTD ファイルのクラス要素，210
DTD ファイルの参照要素，185
DTD ファイルの持続性要素，196
DTD ファイルのプール要素，202

E

EJB
一般的な使用のガイドライン，36
インタフェース，30
概要，25–34
コンテナ，29
設計要因，35
トランザクション属性，151
ユーザ承認，162
要素，180
EJB QL, 20, 84, 105
EJB 2.0 の変更点の概要，20
ejbActivate, 71
EJBContext, 158, 167
ejbCreate, 49, 52, 69, 71
ejbFindByPrimaryKey, 69, 74
ejbFindXXX, 74
ejb-jar.xml ファイル，140, 170
ejbLoad, 72
ejb-name 要素，115, 180
EJBObject, 53
ejbPassivate, 76
ejbPostCreate, 69
EJB-QL, 80, 84
ejb-ref-name 要素，186
ejb-ref 要素，185
ejbRemove, 74, 134
ejbStore, 72
EJB にアクセスするクライアント，30
EJB のアセンブリ，169–213
EJB の 概要，19–34
EJB のトランザクション属性，151
ejb 要素，177
enterprise-beans 要素，180
entity-mapping, 96, 115
establish-trust-in-client 要素，193
establish-trust-in-target 要素，193

F

fetched-with 要素，101, 103, 117, 226
field-name 要素，117
finder 要素，198
Forte for Java, 18, 24

G

getCallerPrincipal, 167
getRollbackOnly, 158
getStatus, 158
getUser, 167
getUserTransaction, 155
group-name 要素，183

I

IDE, 24
integrity 要素，193
ior-security-config 要素，193
isCallerInRole, 167

is-one-one-cmp 要素 , 198
is-read-only-bean 要素 , 78, 182

J

J2EE トランザクションマネージャ , 144
JAR ファイル、概要 , 170
Java Message Service。JMS を参照
java.ejb.CreateException, 52, 71
java.ejb.FinderException, 74
java.rmi.RemoteException, 51, 69, 74
java.sql.Connection, 157
Java Transaction API, 144
  トランザクション , 147
  トランザクション (Bean 管理), 157
Java Transaction Service, 144
javax.ejb.CreateException, 51
javax.ejb.EJBContext, 155
javax.ejb.EJBHome, 50
javax.ejb.EJBLocalHome, 49, 65
javax.ejb.EJBLocalObject, 48
javax.ejb.EJBMetaData, 50
javax.ejb.SessionSynchronization, 47, 52
javax.rmi.PortableRemote.Object.narrow, 37
javax.transaction.UserTransaction, 155
Java 言語キャスト , 37
JDBC
  トランザクション (Bean 管理), 156
  トランザクションタイプ , 147
JDBC または JTA, 156
JDOQL, 105
JMS, 135, 137
jms-durable-subscription-name 要素 , 190
jms-max-messages-load, 190
JNDI, 33, 34
  コンテナ管理持続性の JNDI 名 , 104
  トランザクション , 147
  メッセージ駆動型 Beans の JNDI 名 , 135
jndi-name, 104
jndi-name 要素 , 186

L

level 要素 , 117
lock-when-loaded 要素 , 118
lock-when-modified 要素 , 118

M

Mandatory 属性 , 152
mapping-properties 要素 , 198
max-cache-size 要素 , 206
max-pool-size 要素 , 206
mdb-connection-factory, 135, 139
mdb-connection-factory 要素 , 190
mdb-container, 137
mdb-container 要素 , 136
MDB のプールの監視 , 138
MDB ファイルの例 , 140
meet-in-the-middle マッピング , 89
method-name 要素 , 199
method-permission 要素 , 164
Microsoft SQLServer 2000, 80

N

named-group 要素 , 118
name 要素 , 194
none 要素 , 101, 118
NotSupported 属性 , 152

O

O/R マッピングツール , 109
one-one-finders 要素 , 199
onMessage, 133, 139, 167
Oracle, 80, 92

P

param-name 要素，183, 184, 186, 187, 188
pass-by-reference 要素，23, 186
pass-by-value セマンティック，187
password 要素，194
pc-class 要素，199
persistence-manager-factory-resource，104
pm-class-generator 要素，199
pm-config 要素，200
pm-descriptors，104
pm-descriptors 要素，201
pm-descriptor 要素，200
pm-identifier 要素，201
pm-inuse 要素，201
pm-mapping-factory 要素，202
pm-version 要素，202
Pointbase，80
pool-idle-timeout-in-seconds，207, 208
principal-name 要素，184
principal 要素，183

Q

query-filter 要素，202
query-params 要素，202
query-variables 要素，202

R

ReadOnlyBeanNotifier，77
read-only 要素，100, 118
realm 要素，194
refresh-period-in-seconds，76, 182
removal-timeout-in-seconds 要素，207
Required 属性，151
required 要素，194
RequiresNew 属性，152
resize-quantity 要素，208
resource-env-ref-name 要素，188
resource-env-ref 要素，188
resource-ref 要素，189
res-ref-name 要素，187
RMI/IIOP，50
role-link 要素，165
role-name 要素，184

S

sas-context 要素，195
schema，95
schema 要素，119
secondary-table，100
security-identity 要素，166
security-role-mapping 要素，184
security-role-ref 要素，165
security-role 要素，163, 165
server.xml ファイル，104, 136, 137, 148, 168
server-name 要素，185
SessionSynchronization インタフェース，155
setAutoCommit，155
setEntityContext，73
setMessageDrivenContext，133
setRollbackOnly，155, 158
setTransactionIsolation，159
setTransactionTimeout，158
steady-pool-size，208
steady-pool-size 要素，209
Sun ONE Studio 4，24, 87, 171, 172, 217
sun-cmp-mapping，95
sun-cmp-mapping.xml，94, 111, 122
sun-cmp-mappings，95, 120
sun-cmp-mappings.xml，36, 85, 170
sun-cmp-mapping 要素，119
sun-ejb-jar.xml，103

sun-ejb-jar.xml ファイル , 170
　例 , 141
sun-ejb-jar_2_0.dtd ファイル , 174
sun-ejb-jar_2_0-0.dtd ファイル , 176
sun-ejb-jar 要素 , 182
Sun ONE Application Server
　付加価値機能 , 20
Sun ONE Studio, 18
Sun ONE Studio 4 のチュートリアル , 24
Supports 属性 , 152
Sybase, 80

T

table-name 要素 , 119, 120
transaction.timeout プロパティ , 158
TransactionRequiredException, 152
transaction-service 要素 , 148
transport-config 要素 , 195

U

unique-id 要素 , 183
unsetEntityContext, 73
URL への接続 , 34
use-caller-identity, 166

V

validateLogin, 52
victim-selection-policy 要素 , 209

X

XA プロトコル , 145
XML ファイル , 170
　概要 , 39
　要素 , 176
　例 , 140, 213
XML ファイルの要素 , 111, 180, 229
XML ファイルの例 , 140, 213

あ

アーキテクチャ , 21, 28
　エンティティ Beans, 57
アクセス
　概要 , 30
　リソースへの , 33

い

一対一の関係 , 83
一対多の関係 , 83
インタフェース , 31, 37, 47, 48, 51, 65
　エンティティ Beans, 61
　概要 , 30

え

エンティティ Beans, 55
　開発 , 60–78
　概要 , 27, 56–59
　コンテナ管理による持続性のマッピング , 217
　持続性 , 57
　抽象スキーマ , 84
　トランザクション属性 , 151
　読み取り専用 Beans, 75

か
カスタマサポート , 18
関係 , 83
　一対一 , 83
　一対多 , 83
　多対多 , 84
　マッピング、フィールド , 223
監視 , 24

き
キャッシュの管理 , 24
キャプチャスキーマユーティリティ , 87

く
クライアントビューのガイドライン , 37
グローバルトランザクション , 145

け
検索メソッド , 74, 105

こ
コミットオプション , 147, 148, 155
コミットなしの復帰 (Bean 管理), 157
コレクションフィールド , 84
コンテナ
　エンティティ Beans, 57
　概要 , 29
　セッション Beans, 43
コンテナ管理トランザクション , 52, 146, 149–155
　禁止されているメソッド , 155
　属性 , 150
　メッセージ駆動型 Beans の , 137
　ロールバック , 154
コンテナ管理による持続性 , 59, 79–122
　1.1 検索の設定 , 105
　アセンブリと配備 , 85
　オペレーション , 87
　概要 , 33, 81–85
　関係 , 83
　サードパーティのサポート , 109
　サポート , 80
　実装 , 86–122
　遮断レベルの設定 , 159
　スキーマのマッピング , 219
　配備 , 103
　プロパティ , 226
　マッピング , 89, 217–227
　マッピングのデータタイプ , 90
　要素 , 111
　リソースマネージャ , 103

さ
細分 , 38
サブ要素 , 174

し
識別 , 162
持続性の概要 , 57
持続性のプロパティ , 226
持続性マネージャ , 103
実行 ID, 166
自動再接続機能 , 137
主キークラス , 80
主表 , 220
主表のマッピング , 226
承認 , 162

す

スキーマの取り込み , 121, 217
スキーマのマッピング , 219, 226
スキーマ、取り込み , 87, 217
スタブとスケルトン , 171
ステートフルセッション Beans, 46
　概要 , 27
　トランザクション , 157
ステートレスセッション Beans, 46
　概要 , 27

せ

制限事項
　コンテナ管理による持続性 , 111
　セッション Bean のトランザクション , 54
　メッセージ駆動型 Beans, 138
製品の付加価値機能 , 23
セキュリティ , 161–168
　DTD 要素 , 191
　ID の指定 , 166
　アセンブリと配置 , 163
　概要 , 162–163
　プログラムによるセキュリティ , 167
　保護されていない EJB 層のリソース , 168
　メソッドパーミッションの宣言 , 164
　ロール , 162
　ロール参照 , 163
　ロール参照の宣言 , 165
設計要因 , 35
セッション Beans, 41–53
　Bean クラス定義の作成 , 51
　開発 , 45, 53
　概要 , 42
　コンテナ , 43
　遮断レベルの設定 , 159
　制限事項 , 54
　トランザクション属性 , 151
セッション Beans の remove メソッド , 50, 63
セッション Beans のビジネスメソッド , 52
接続ファクトリ , 135, 139
設定
　1.1 検索 (CMP), 105
　リソースマネージャ (CMP), 103

そ

属性
　配備記述子内 , 175
　トランザクション , 150

た

タイムアウト、トランザクションでの設定 , 158
多対多の関係 , 84
単層型トランザクション , 145

ち

抽象スキーマ , 84, 217

て

データベーススキーマ、取り込み , 87, 88, 217
データベースへの接続、概要 , 33

と

同期化 , 78
同時アクセス , 78, 132
動的配備 , 171
匿名ロール , 168
トランザクション , 143–159

Bean 管理，147
概要，34, 144–149
管理と監視，148
境界設定モデル，146
グローバル，145
コミットオプション，147
コンテナ管理，146
遮断レベル，159
仕様書，144
タイムアウトの設定，158
単層型，145
ネストされた，145
メッセージ駆動型 Beans，131
ローカル，145
ロールバック，159
トランザクションの監視，149
トランザクションの管理，148
トランザクションの境界設定モデル，146
トランザクションの遮断レベル，159

に

二次表，87, 94, 97, 99, 113, 221
認証，162

ね

ネストされたトランザクション，145

は

配備
概要，39
コンテナ管理による持続性，103
読み取り専用 Beans，78
配備記述子
概要，39, 170
配備、動的，171
パッケージング、「EJB のアセンブリ」を参照

ふ

フィールドのマッピング，223
プール，24, 29, 43, 47, 76, 109, 132, 136
付加価値機能，20
複数カラム，100
複数カラム、マッピング，222
プログラムによるセキュリティ，167
プロパティ (持続性)，226

へ

ベリファイアツール，36
ベンダー，109, 198

ほ

ホームインタフェース，30
保護されていないリソース，168

ま

マッピング，109
CMP の，217–227
DTD ファイルの要素，111
関係フィールド，223
機能，89
主表，226
スキーマ，219, 226
ツール，89
データタイプ，90
二次表，226

複数カラム (CMP), 222
マッピングツール (CMP), 218, 222
マッピングのデータタイプ , 90
マッピングプロパティ , 223

め

メソッドパーミッション、宣言 , 164
メッセージ駆動型 Bean
プール , 136
メッセージ駆動型 Beans, 129–140
Bean クラス定義 , 132
DTD 要素 , 190
JMS に関する制限事項 , 138
onMessage のランタイム例外 , 139
run-as の使用 , 167
XML ファイルの例 , 140
開発 , 132
概要 , 27, 130–136
監視 , 136
接続ファクトリ , 135
トランザクション , 131
トランザクション属性 , 151
プールの監視 , 138
メッセージ駆動型 Beans の管理 , 136
メッセージング要素 , 190

よ

読み取り専用 Bean, 59, 23
読み取り専用 Beans, 75–78
更新 , 76
配備 , 78

り

リソースへの接続 , 33
リソース、保護されていない , 168
リッチクライアント , 171
リモートインタフェース
概要 , 30
セッション Beans, 47
リモートプログラミングモデル , 37
リモートホームインタフェース
概要 , 30
セッション Beans, 50

ろ

ローカルインタフェース
エンティティ Beans, 65
概要 , 30
セッション Beans, 48
ローカルトランザクション , 145
ローカルプログラミングモデル , 38
ローカルホームインタフェース , 49, 65
概要 , 30
ロールバック , 154
トランザクション , 159
メソッド , 155, 158
ロール、セキュリティ , 162
参照の宣言 , 165
定義 , 163