

サーバーアプリケーションの移行 および再配備

Sun™ ONE Application Server

Version 7, Enterprise Edition

817-5551-10

2003 年 9 月

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.

Copyright © 2003 Sun Microsystems, Inc. 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、iPlanet、Sympton、Java、Sun™ ONE、Java Coffee Cup のロゴマークおよび 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 | 7 |
| 前提事項 | 7 |
| マニュアルの構成 | 8 |
| マニュアルの使用法 | 9 |
| マニュアルの表記規則 | 11 |
| 一般的な表記規則 | 11 |
| ディレクトリ名の表記規則 | 12 |
| 製品サポート | 13 |
| | |
| 第1章 移行に関する注意事項 | 15 |
| 移行理由 | 15 |
| Sun ONE Application Server 7 の利点 | 16 |
| 開発機能 | 16 |
| 運用機能 | 17 |
| Enterprise Edition の追加機能 | 18 |
| クラスタリングによる高度なスケーラビリティ | 18 |
| ロードバランス機能によるパフォーマンスの向上 | 18 |
| フェイルオーバーによる高可用性の実現 | 18 |
| 移行方法 | 19 |
| 移行を必要とする要素 | 19 |
| 再配備 | 20 |
| | |
| 第2章 Sun ONE Application Server 7, Enterprise Edition の概要 | 21 |
| 製品ラインの概要 | 21 |
| Platform Edition | 22 |
| Standard Edition | 22 |

| | |
|------------------------------------------------|----|
| Enterprise Edition | 23 |
| Sun ONE Application Server 7 のアーキテクチャ | 23 |
| J2EE コンポーネント標準 | 26 |
| 開発環境 | 27 |
| Sun ONE Application Server 6.0/6.5 の開発環境 | 27 |
| Sun ONE Application Server 7 の開発環境 | 28 |
| 管理ツール | 29 |
| Sun ONE Application Server 6.0 の管理ツール | 29 |
| Sun ONE Application Server 6.5 の管理ツール | 30 |
| Sun ONE Application Server 7 の管理ツール | 30 |
| データベース接続 | 32 |
| J2EE アプリケーションコンポーネントと移行 | 32 |

第 3 章 Sun ONE Application Server 6.x から Sun ONE Application Server 7 への移行 35

| | |
|------------------------------------------------------|----|
| Sun ONE Application Server 6.0/6.5 について | 36 |
| 配備記述子の移行 | 37 |
| Sun ONE Application Server 6.x から 7 への移行に関する問題 | 38 |
| J2EE コンポーネントの移行 | 39 |
| JDBC コードの移行 | 39 |
| DriverManager インタフェース経由の接続の確立 | 40 |
| JDBC 2.0 データソースの使用方法 | 41 |
| Java Server Pages および JSP カスタムタグライブラリの移行 | 45 |
| サーブレットの移行 | 46 |
| JNDI コンテキストからのデータソースの取得 | 47 |
| JNDI コンテキスト内での EJB の宣言 | 47 |
| EJB の移行 | 47 |
| Sun ONE Application Server 7 に対応した EJB の変更 | 48 |
| セッション Beans | 48 |
| エンティティ Beans | 48 |
| メッセージ駆動型 Beans | 49 |
| Web アプリケーションの移行 | 50 |
| Web アプリケーションモジュールの移行 | 50 |
| サーブレットおよび JSP の移行時の特定の障害 | 51 |
| エンタープライズ EJB モジュールの移行 | 53 |
| エンタープライズアプリケーションの移行 | 54 |
| アプリケーションルートコンテキストとアクセス URL | 55 |
| 固有の拡張子の移行 | 56 |
| UIF の移行 | 56 |
| 方法 1: レジストリファイルのチェック | 57 |
| 方法 2: インストールディレクトリの UIF バイナリをチェック | 57 |
| 移行プロセス | 58 |
| リッチクライアントの移行 | 59 |

| | |
|---------------------------------------------------------------------------------|------------|
| 6.x のクライアントの認証 | 59 |
| 7 SE/EE のクライアントの認証 | 59 |
| 6.x と 7 EE での ACC の使用 | 60 |
| 第 4 章 インストール、管理、および配備 | 63 |
| インストールの相違 | 63 |
| 最小要件 | 64 |
| インストール手順の相違 | 65 |
| 管理と配備の相違 | 66 |
| root 以外のユーザーによるインストールと管理 | 66 |
| 配備トポロジ | 67 |
| 第 5 章 iBank アプリケーションの移行手順 | 69 |
| iBank アプリケーションの移行の準備 | 70 |
| iBank アプリケーションの手動移行 | 72 |
| Web アプリケーションの変更 | 72 |
| EJB の変更 | 73 |
| 配備用アプリケーションのアセンブル | 91 |
| asadmin ユーティリティを使用した Sun ONE Application Server 7 での iBank アプリケーションの配備 | 92 |
| 第 6 章 6.5 アプリケーションの Sun ONE Studio へのインポート | 93 |
| アプリケーションの移行の準備 | 93 |
| アプリケーションコンポーネントの移行 | 95 |
| Sun ONE Studio for Java での Web アプリケーションモジュールの作成 | 96 |
| CMP エンティティ EJB の 1.1 から 2.0 への変換 | 102 |
| Sun ONE Studio for Java での EJB モジュールの作成 | 113 |
| Sun ONE Studio for Java でのエンタープライズアプリケーションの作成 | 130 |
| Sun ONE Application Server 7 でのアプリケーションの配備 | 133 |
| 付録 A iBank アプリケーションの仕様 | 135 |
| アプリケーション開発用ツール | 136 |
| データベーススキーマ | 136 |
| アプリケーション間の移動とロジック | 140 |
| アプリケーションコンポーネント | 144 |
| 移行時に発生する問題を考慮した最適な設計の選択 | 147 |
| 付録 B 移行に関する資料 | 151 |
| その他のアプリケーションサーバー環境からのアプリケーションの移行 | 151 |

| | |
|---------------------------------------------------------------------------------------|------------|
| BEA WebLogic から Sun ONE Application Server 7 への移行 | 151 |
| IBM WebSphere から Sun ONE Application Server 7 への移行 | 152 |
| 移行ツール | 152 |
| Sun ONE Studio Enterprise Edition for Java、リリース 4.1 | 152 |
| Sun ONE Migration Tool for Application Server | 153 |
| Sun ONE Migration Toolbox for Applogic and NetDynamics | 153 |
| Sun One Connector Builder | 154 |
| Native Connector Toolkit | 154 |
| J2EE Application Verification Kit | 155 |
| 参考資料 | 156 |
| Sun ONE Application Server 6.0 への移行 | 156 |
| Sun ONE Application Server 6.5 への移行 | 156 |
| Sun ONE Application Server 7 への移行 | 156 |
| 移行したアプリケーションの再配備 | 157 |
| 付録 C Enterprise Java Beans 1.1 仕様から Enterprise Java Beans 2.0 仕様への移行 | 159 |
| EJB 1.1 と EJB 2.0 の相違点 | 159 |
| EJB クエリ言語 | 160 |
| ローカルインタフェース | 160 |
| EJB 2.0 コンテナ管理による持続性 (CMP) | 161 |
| 持続性フィールドの定義 | 162 |
| エンティティ Bean の関係の定義 | 162 |
| メッセージ駆動型 Beans | 163 |
| EJB クライアントアプリケーションの移行 | 163 |
| JNDI コンテキスト内での EJB の宣言 | 163 |
| EJB JNDI 参照の使用方法的要約 | 164 |
| JNDI コンテキスト内の EJB 参照の配置 | 164 |
| グローバル JNDI コンテキストとローカル JNDI コンテキスト | 165 |
| CMP エンティティ EJB の移行 | 165 |
| Bean クラスの移行 | 166 |
| ejb-jar.xml の移行 | 169 |
| カスタム検索メソッド | 169 |
| 索引 | 173 |

本書について

この『サーバーアプリケーションの移行および再配備』では、以前のバージョンの Sun ONE Application Server から Sun ONE Application Server 7 製品ラインへの J2EE アプリケーション移行方法を説明します。

このマニュアルは、移行の詳細を知る必要があるシステム管理者、ネットワーク管理者、アプリケーションサーバー管理者、および Web 開発者を対象にしています。

前提事項

このマニュアルでは、次の項目について熟知していることを前提とします。

- HTML
- アプリケーションサーバー
- クライアント/サーバープログラミングモデル
- インターネットおよび WWW (World Wide Web)
- Windows 2000 または Solaris™ オペレーティングシステム
- Java プログラミング
- EJB および JSP (Java Server Pages) の仕様に定義されている Java API
- Java Database Connectivity (JDBC)
- SQL などの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

マニュアルの構成

このマニュアルの構成は次のとおりです。

- [第 1 章「移行に関する注意事項」](#) - Sun ONE Application Server 7 で利用可能な機能および Enterprise Edition で拡張された機能についての説明
- [第 2 章「Sun ONE Application Server 7, Enterprise Edition の概要」](#) - Sun ONE Application Server 7 のアーキテクチャ、J2EE 標準間の変更点、およびこのバージョンと以前のバージョンの Sun ONE Application Server の導入アプリケーションの違いについての説明
- [第 3 章「Sun ONE Application Server 6.x から Sun ONE Application Server 7 への移行」](#) - Sun ONE Application Server 6.x から 7 Standard Edition または Enterprise Edition にアプリケーションを移行する際の注意事項と方法についての説明。また、移行プロセス全体の説明に使用するサンプルアプリケーションを掲載
- [第 4 章「インストール、管理、および配備」](#) - Sun ONE Application Server 7, Enterprise Edition と Sun ONE Application Server 6.x Enterprise Edition のインストールおよび管理の相違についての説明。配備トポロジの簡単な説明も付属
- [第 5 章「iBank アプリケーションの移行手順」](#) - J2EE アプリケーションの主要コンポーネントを Sun ONE Application Server 7 Enterprise Edition に移行するプロセスについての説明。iBank サンプルアプリケーションを使って手順を解説
- [第 6 章「6.5 アプリケーションの Sun ONE Studio へのインポート」](#) - Sun ONE Application Server 6.x アプリケーションを Sun ONE Application Server 7 Enterprise Edition にインポートする手順と配備する手順を説明
- [付録 A 「iBank アプリケーションの仕様」](#) - EJB 1.1 から EJB 2.0 仕様に移行するために必要な変更内容についての説明
- [付録 B 「移行に関する資料」](#) - J2EE アプリケーションを Sun ONE Application Server 7, EE に移行する際に役立つリソースの説明
- [付録 C 「Enterprise Java Beans 1.1 仕様から Enterprise Java Beans 2.0 仕様への移行」](#) - このマニュアルを通して使用する iBank 仕様の解説

マニュアルの使用法

Sun ONE Application Server のマニュアルは、次の URL から PDF 形式または HTML 形式で入手できます。

<http://docs.sun.com/db/prod/slappsrv>

次の表は、Sun ONE Application Server 7, Enterprise Edition 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 7, Enterprise Edition のマニュアル紹介

| 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です。 | 『リリースノート』 |
| サポート対象のハードウェア、オペレーティングシステム、JDK、および JDBC/RDBMS の概要について表形式にまとめたものです。 | 『プラットフォームの概要』 |
| 各製品版の機能説明をはじめとする Sun ONE Application Server 7 の概要情報です。 | 『製品の概要』 |
| Sun ONE Application Server アーキテクチャ方式のサーバーアーキテクチャと利点について、図を交えながら説明します。 | 『サーバーアーキテクチャの概要』 |
| Sun ONE Application Server 7 の新しいエンタープライズ機能、開発者向け機能、オペレーショナル機能について説明します。 | 『新機能』 |
| Sun ONE Application Server 7 製品の入門マニュアルです。新機能とアーキテクチャの概要情報、サンプルアプリケーションチュートリアルも付属しています。 | 『入門ガイド』 |
| Sun ONE Application Server ソフトウェアとそのコンポーネントのインストール方法を示します。サンプルアプリケーション、管理インタフェース、高可用性コンポーネントなどのコンポーネント群があります。基本的な高可用性設定の実装方法についても解説します。 | 『インストールガイド』 |
| Sun ONE Application Server をサイトに最適な方法で配備できるように、システム要件ならびにエンタープライズの評価を行います。一般的な問題やアプリケーションサーバーを配備する上で考慮すべき事項についても解説します。 | 『システム配備ガイド』 |
| アプリケーションの設計者と開発者向けに、HTTP セッション可用性のベストプラクティスを紹介します。 | 『Application Design Guidelines for Storing Session State』 |

Sun ONE Application Server 7, Enterprise Edition のマニュアル紹介 (続き)

| 情報の内容 | 参照するマニュアル |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------|
| サーブレット、Enterprise JavaBeans™ (EJBs™)、JavaServer Pages™ (JSPs™) などの J2EE コンポーネントの Java オープンスタンダードモデルに準拠した Sun ONE Application Server 7 上で実行する Java™ 2 Platform, Enterprise Edition (J2EE™ プラットフォーム) アプリケーションの作成および実装方法について説明します。アプリケーションの設計、各種開発ツール、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方法などの一般的な情報を取り上げます。包括的な Sun ONE Application Server 用語集も付属しています。 | 『開発者ガイド』 |
| Sun ONE Application Server 7 上で Java™ Servlet および JavaServer Pages (JSP) 仕様に準拠した J2EE Web アプリケーションを作成し、実装する方法を示します。Web アプリケーションの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、関連資料も付属しています。結果キャッシュ機能、JSP のプリコンパイル、セッション管理、セキュリティ、配備、SHTML、CGI などのトピックを取り上げます。 | 『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 Bean 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を示します。Enterprise JavaBeans (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、関連資料も付属しています。コンテナ管理持続性、読み取り専用 Bean、エンタープライズ Bean に関連付けられた XML ファイルや DTD ファイルなどについて取り上げます。 | 『Enterprise JavaBeans 開発者ガイド』 |
| Sun ONE Application Server 7 上の J2EE アプリケーションにアクセスするアプリケーションクライアントコンテナ (ACC) クライアントの作成方法について説明します。 | 『Developer's Guide to Clients』 |
| Sun ONE Application Server 環境での Web サービスの作成方法について説明します。 | 『Developer's Guide to Web Services』 |
| Java™ Database Connectivity (JDBC™)、トランザクション、Java Naming and Directory Interface™ (JNDI)、Java™ Message Service (JMS)、および JavaMail™ API について説明します。 | 『Developer's Guide to J2EE Services and APIs』 |
| カスタム NSAP プラグインの作成方法について説明します。 | 『NSAPI Developer's Guide』 |
| 管理インタフェースとコマンド行インタフェースを使った Sun ONE Application Server のサブシステムおよびコンポーネントの設定、管理、配備について説明します。クラスタ管理、高可用性データベース、不可均衡、セッション持続などのトピックを取り上げます。包括的な Sun ONE Application Server 用語集も付属しています。 | 『管理者ガイド』 |

マニュアルの表記規則

この節では、このマニュアルの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

このマニュアルは、次の表記規則に従っています。

- 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)。
- URL は次の書式で記述します。

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

server はアプリケーションを実行するサーバー名、*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、次のように使い分けます。
 - 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 タグに使用します。
 - 斜体文字はコード変数に使用します。
 - 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用します。
 - 太字は、段落の先頭またはリテラルに使われる文字の強調に使用します。
- 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、[12 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* は次の場所になります。

- Solaris™ 8 のパッケージベースでない評価 (Evaluation) バージョンのインストール

ユーザーのホームディレクトリ `/sun/appserver7`

- Solaris にバンドルされていない評価用以外のバージョンのインストール
`/opt/SUNWappserver7`

default_config_dir ディレクトリおよび *install_config_dir* ディレクトリは、*install_dir* と同義です。例外と追加情報については、[12 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は、*instance_dir* と記述します。これは以下のパスの省略形式です。

default_config_dir/domains/*domain*/*instance*

ディレクトリ名の表記規則

Solaris™ 8 および 9 のインストール では、アプリケーションサーバーのファイルはデフォルトで様々なディレクトリ場所に保存されます。ここでは、これらのディレクトリについて説明します。

- **Solaris 8 および 9 のインストール**では、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されます。
 - *install_dir* は /opt/SUNWappserver7 を示します。このディレクトリにはインストールイメージの静的な要素が保存されます。ユーティリティ、実行可能ファイル、およびアプリケーションサーバーを構成するライブラリは、すべてここに保存されます。
 - *default_config_dir* は /var/opt/SUNWappserver7/domains を示します。このディレクトリは、作成されるドメインのデフォルトの保存場所です。
 - *install_config_dir* は /etc/opt/SUNWappserver7/config を示します。このディレクトリには、ライセンスなどのインストール全体に適用される設定情報や、このインストール用に設定した管理ドメインのマスターリストが保存されます。

製品サポート

製品またはマニュアルに関する一般的なフィードバックは、appserver-feedback@sun.com にお送りください。

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/supporttraining/>
- 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。

サポートのご依頼の前に、次の情報を用意してください。サポート担当がお客様の問題を解決するために必要な情報です。

- 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他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

移行に関する注意事項

この章では、Sun™ Open Net Environment (ONE) Application Server で利用可能な機能に加え、Enterprise Edition で拡張された機能の概要についても説明しています。

移行理由

Sun ONE Application Server は、最新の Java テクノロジーを開発者向けに使いやすくパッケージ化した製品です。Application Server は、高度なスケーラビリティを備えたアプリケーションサーバー技術を実現したいという Sun の専門家たちの 6 年以上におよぶ開発努力の成果です。本製品により、JavaServer Pages (JSP™) テクノロジー、Java™ Servlet、Enterprise Java Beans™ (EJB™) テクノロジーをベースにした堅牢なアプリケーションを迅速に開発できます。本製品のテクノロジーは、小規模な部門アプリケーションからエンタープライズ規模のミッションクリティカルなサービスまで、広い範囲のビジネス要件をカバーしています。

J2EE 仕様は、アプリケーションに対する要求を幅広くカバーしていますが、その一方で標準として進化を続けています。これは、アプリケーションのある面をカバーしていない、または実装の詳細をアプリケーションプロバイダに任せているということではありません。

これらの製品の実装に依存する面は、アプリケーションサーバーの構成方法の違い、およびアプリケーションサーバー上への J2EE コンポーネントの配備方法の違いとして明らかになります。使用可能な構成の配列、および特定のアプリケーションサーバー製品で使用する配備ツールも、製品実装の差異の原因となります。

仕様が進化すること自体が、アプリケーションプロバイダにとっては課題になります。各コンポーネント API は別々に進化します。このため、製品ごとに統一性の度合いが異なります。特に、Sun ONE Application Server などの新しく開発された製品は、他の確立されたアプリケーションサーバープラットフォームに配備された J2EE アプリケーションコンポーネント、モジュール、およびファイルの違いに対処しなければなりません。このような違いに対処するには、ファイル命名規則、メッセージ構文など、J2EE 標準以前の実装詳細間でのマッピングが必要になります。

さらに、製品プロバイダは通常、製品に追加機能とサービスをバンドルしています。これらの機能は、カスタムの JSP タグや、専用の Java API ライブラリとして利用できます。ただし、このような専用機能を使用すると、アプリケーションを移植できなくなります。

Sun ONE Application Server 7 の利点

Sun ONE Application Server 7 は、開発および運用環境を拡張する新しい機能を多数提供しています。

これらの新機能は、製品のすべてのエディションで利用できます。

- [開発機能](#)
- [運用機能](#)
- [Enterprise Edition の追加機能](#)

開発機能

Sun ONE Application Server 7 には、次の開発機能があります。

- Java 2 Enterprise Edition 1.3 準拠。次のような機能が含まれます
 - JavaServer Pages (JSP) 1.2 および Servlet 2.3 のサポート
 - Enterprise JavaBeans (EJB) 2.0 テクノロジ
 - メッセージ駆動型 Beans (MDB)
- Java 2 Platform, Standard Edition (J2SE™ プラットフォーム) 1.4
- 統合された Java Web サービス
- 統合開発環境 Sun ONE Studio インテグレーションのアップデートによるシームレスなデバッグおよび配備
- 動的な (ホット) 配備と再読み込み
- JSP ソースレベルデバッグ機能
- コンテナ管理による持続性 (CMP) サポートの大幅な拡張
- 設定しやすい XML ベースのサーバー設定
- ライフサイクルリスナークラス (高度な起動および停止クラス)
- 統合 J2EE アプリケーション検証ユーティリティ
- 多数のサンプルアプリケーション

- Ant ビルド機能の統合
- 依存性の最小化による簡単なインストール (独立した Web サーバーやディレクトリサーバーは不要)

運用機能

Sun ONE Application Server 7 には、次の運用機能があります。

- 統合された高パフォーマンスの HTTP サーバー
- JMS プロバイダとして統合された検証済み Sun ONE Message Queue 3.0, Platform Edition
- Web アプリケーション用の仮想 HTTP サーバーサポート
- インストールイメージごとに複数の管理ドメインを設定する機能 (単一のインストールイメージから複数のアプリケーションサーバーの構成が可能)
- Web ベースの管理
- Web サーバーからアプリケーションサーバーへ要求をリダイレクトするプロキシプラグイン
- 完全機能のリモートコマンド行インタフェースによるリモート監視
- Java 認証承認サービス (JAAS) に基づいたプラグイン可能な認証
- 複数のログレベルを設定できる改良されたログ機能
- Solaris の SVR4 パッケージ形式によるプラットフォーム固有のパッケージ
- Web ベースのリモート管理インタフェース
- バンドルされている Sun ONE Directory Server をユーザー認証とアプリケーション構成の制限のために使用する権限
- Web サーバー上に構成された URI を別のアプリケーションサーバーにリダイレクトすることによる負荷分散
- SNMP 監視機能によるリモート監視

詳細は、『Sun ONE Application Server 7, Enterprise Edition プラットフォームの概要』を参照してください。

Enterprise Edition の追加機能

Sun ONE Application Server 7, Enterprise Edition には、Standard Edition にはない次の機能があります。

- クラスタリングによる高度なスケーラビリティ
- ロードバランス機能によるパフォーマンスの向上
- フェイルオーバーによる高可用性の実現

クラスタリングによる高度なスケーラビリティ

Sun ONE Application Server では、複数のインスタンスを 1 つのマシンに追加できることから、システム全体の性能の低下なしに、容量を増加できます。特に、Application Server のインスタンスは、複数のマシンに分散できることから、効率的な管理のために、「クラスタ」によるグループ化をサポートしています。

ロードバランス機能によるパフォーマンスの向上

複数のインスタンス、またはクラスタ内の個々のインスタンスの動的ロードバランスにより、Sun ONE Application Server と J2EE アプリケーションのパフォーマンスが最適化されます。ロードバランス設定は動的に再読み込みされるため、サービスを中断することなくクラスタにインスタンスを追加できます。

フェイルオーバーによる高可用性の実現

Sun ONE Application Server 7, Enterprise Edition は、ロードバランスと高度なフェイルオーバー機構を利用して、信頼性の高い高可用性ソリューションを提供します。

また、バンドル版高可用性データベースには、HTTP/S セッション情報を安全に格納できます。

これらの拡張機能と高可用性データベースサーバーにより、J2EE アプリケーションを 24 時間 365 日連続して稼働できます。

| | |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 注 | ユーザーが使用できる機能は、製品のインストール時に合意したライセンス契約によって決定します。『補足条項』より使用できる機能を確認してください。Sun ONE Application Server 7 は、Sun のまったく新しいアプリケーションサーバー製品です。本製品は、iPlanet Application Server でサポートされていた AppLogic 形式のアプリケーションはサポートしていません。 |
|----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

移行方法

この節では、移行が必要なアプリケーションコンポーネントについて説明します。移行プロセスの詳細については、[第 5 章「iBank アプリケーションの移行手順」](#)を参照してください。ハードウェアとソフトウェアの要件の計画については、[第 4 章「インストール、管理、および配備」](#)を参照してください。

移行を必要とする要素

移行のために、J2EE アプリケーションは次のファイルカテゴリで構成されています。

- [配備記述子 \(XML ファイル\)](#)
- [独自の API を含む JSP ソースファイル](#)
- [独自の API を含む Java ソースファイル](#)

配備記述子 (XML ファイル)

配備は、EJB (ejb-jar)、フロントエンド Web コンポーネント (war)、およびエンタープライズアプリケーション (ear) のそれぞれに対する配備記述子 (DD) を指定して行います。配備記述子は、J2EE コンポーネントおよびアプリケーションのすべての外部依存関係の解決に使用されます。DD の J2EE 仕様はすべてのアプリケーションサーバー製品で共通です。ただし、製品の実装に依存するアプリケーション関連コンポーネントの配備に関する面で、仕様が規定されていない部分があります。

独自の API を含む JSP ソースファイル

J2EE では、カスタムタグ追加による JSP 拡張方法を指定しています。一部の製品には開発者のタスクを単純化する JSP 拡張が含まれています。ただし、これらの専用カスタムタグを使用すると、JSP ファイルを移植できなくなります。また、JSP では別の Java ソースファイルで定義されているメソッドも同様に呼び出せます。JSP で専用の API を使用している場合は、移行前にその部分を変更する必要があります。

独自の API を含む Java ソースファイル

Java ソースファイルはサーブレット、EJB、または他のヘルパークラスのいずれかです。サーブレットおよび EJB は標準 J2EE サービスを直接呼び出せます。また、ヘルパークラスで定義されたメソッドも呼び出せます。Java ソースファイルは、EJB などのビジネスレイヤのアプリケーションのエンコードに使用します。ベンダーはそれぞれの製品にいくつかのサービスおよび専用 Java API をバンドルしています。このような専用 Java API を使用すると、アプリケーションが移植できなくなります。J2EE は標準として進化し続けているため、製品が変わればサポートしている J2EE コンポーネント API のバージョンが変わります。このような点についても移行で対応します。

上記のファイルカテゴリ内のファイルは、Sun ONE Application Server に移行する必要があります。上記のファイルカテゴリの各ファイルの移行方法の詳細は、「Sun ONE Application Server 6.x から 7 への移行に関する問題」を参照してください。

再配備

再配備とは、以前のバージョンの Sun ONE Application Server や、これまで配備されていた移行済みのアプリケーション、また競合するアプリケーションサーバープラットフォームから、以前に配備されていたアプリケーションを再び配備することです。

アプリケーションの再配備は、『Sun ONE Application Server 管理者ガイド』に示されている標準的な配備概要に従って行います。

Sun ONE Application Server 7, Enterprise Edition の概要

この章では、Sun™ ONE Application Server 7 製品ラインの概要、Sun™ ONE Application Server 7 のアーキテクチャ、およびサーバー環境に不可欠な J2EE コンポーネントについて説明します。さらに、Sun ONE Application Server 7 製品ラインの環境とそれ以前の Sun ONE Application Server 環境の違いについても説明します。

次の項目について説明します。

- 製品ラインの概要
- Sun ONE Application Server 7 のアーキテクチャ
- J2EE コンポーネント標準
- 開発環境
- 管理ツール
- データベース接続
- J2EE アプリケーションコンポーネントと移行

製品ラインの概要

Sun ONE Application Server 7 は、アプリケーションサーバー関連の技術の向上に寄与する、画期的な製品です。最新の Java テクノロジーを開発者向けに使いやすくパッケージ化しています。Application Server は、高度なスケーラビリティを備えたアプリケーションサーバー技術を実現したいという Sun の専門家たちの 6 年以上におよぶ開発努力の成果です。本製品により、JavaServer Pages (JSP™) テクノロジー、Java™ Servlet、Enterprise Java Beans™ (EJB™) テクノロジーをベースにした堅牢なアプリケー

ションを迅速に開発できます。本製品のテクノロジーは、小規模な部門アプリケーションからエンタープライズ規模のミッションクリティカルなサービスまで、広い範囲のビジネス要件をカバーしています。本稼動環境と開発環境の両方に幅広く対応するため、次の3種類のアプリケーションサーバーが用意されています。

- [Platform Edition](#)
- [Standard Edition](#)
- [Enterprise Edition](#)

Platform Edition

Platform Edition は、Sun ONE Application Server 7 製品ラインの中核を成す製品です。本製品では、J2EE 1.3 仕様に準拠した高性能な実行環境を提供します。この環境は、基本的な配備作業やサードパーティ製のアプリケーションの組み込みにも適しています。

Platform Edition の配備先は、単一アプリケーションサーバーインスタンスに限られています。このアプリケーションサーバーインスタンスは、単一の Java プラットフォーム用の仮想マシン、すなわち Java 仮想マシン (JVM™) になります。Platform Edition では、複数層の配備トポロジがサポートされます。ただし、Web サーバー層のプロキシはロードバランスを行いません。さらに、管理ユーティリティを使用できるのは、ローカルクライアントに限定されています。

Sun ONE Application Server 7, Platform Edition は Solaris 9 にバンドルされています。

Standard Edition

Standard Edition では、Platform Edition の機能に加え、拡張されたリモート管理機能の層が提供されます。リモート管理機能では、中央の管理ステーションから複数のアプリケーションサーバーインスタンスを管理することが可能です。また、Web サーバー層のプロキシにより Web アプリケーションのトラフィックを分散する機能もあります。Standard Edition では、管理ドメインごとに複数のアプリケーションサーバーインスタンスを設定できます。また、SNMP (Simple Network Management Protocol) を使って Standard Edition のアプリケーションサーバーを監視できます。バンドルされている Sun ONE Directory Server は、ユーザー認証機能を持ち、限られたアプリケーション構成データを格納しています。

Enterprise Edition

Enterprise Edition では、アプリケーションサーバープラットフォームの基本機能に、高可用性、ロードバランス機能、クラスタ管理機能が追加されています。これにより、高性能な実行環境を必要とするような、J2EE ベースのアプリケーションもスムーズに配備できます。Standard Edition より高度な管理機能により、複数のインスタンスの配備にも対応しています。

クラスタリング機能では、複数のアプリケーションサーバーインスタンスの複製をグループとして構成し、クライアント要求のロードバランスを行うことができます。Enterprise Edition では、Web 層のロードバランスプラグインとサードパーティのハードウェアロードバランサがサポートされます。「Always On (常時配信)」という独自の高可用性データベーステクノロジーを、高可用性持続ストアの基盤として採用しています。

Sun ONE Application Server 7 のアーキテクチャ

アプリケーションサーバーは、クライアントがバックエンドソースに接続するための基盤を提供し、アプリケーションロジックを実行し、実行結果をクライアントに戻します。アプリケーションサーバーは 3 層コンピューティングモデルの中間層を使用します。

Sun ONE Application Server 7 製品ラインは、多種多様なサーバー、クライアント、およびデバイスを対象とした電子商取引アプリケーションサービスを開発、配備、管理するための堅牢な J2EE プラットフォームです。

Sun ONE Application Server 7, Enterprise Edition アーキテクチャの主な機能は次のとおりです。

- **クラスタリング** : クラスタは、Sun ONE Application Server インスタンスのグループです。このグループは通常、単一の持続的なデータストア、高可用性データベース、およびロードバランスプラグインが組み込まれた Web サーバーにセッションデータを格納するように設定されます。クラスタは、外部クライアントには、Sun ONE Application Server のシングルインストールのように見えます。個々の Sun ONE Application Server インスタンスはクラスタ内のノードとして参照されます。

クラスタリングの設定の詳細については、『管理者ガイド』の「クラスタ管理」の章を参照してください。

- **ロードバランス** : ロードバランスプラグインは Web サーバーの拡張機能の 1 つです。Web サーバーはプラグインへの特定の HTTP 要求に応答するように設定されています。プラグインは、HTTP 要求をクラスタ内の個々のノードに配信します。

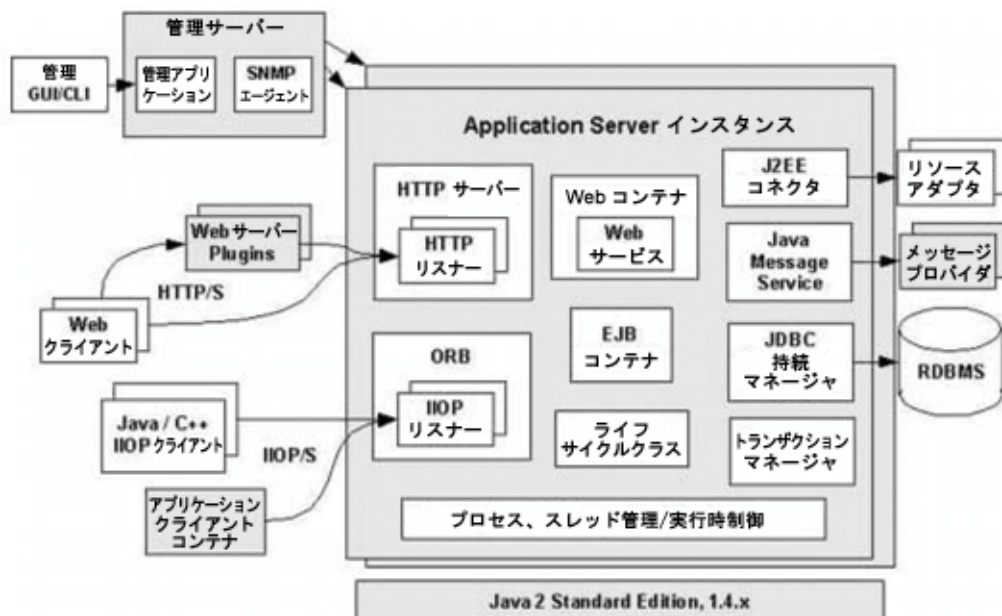
ロードバランスの設定の詳細については、『管理者ガイド』の「ロードバランスの設定」の章を参照してください。

- セッション持続性フェイルオーバー**：セッションデータを持続するように設定されているクラスタ内のノードに障害が発生した場合、別のノードが問題のノードのセッションにアクセスし、引き続きクライアントに応答します。セッションデータは、高い可用性とスケーラビリティを実現したトランザクションデータストアである高可用性データベース (HADB) に格納されます。

高可用性データベースによるセッション持続性の設定の詳細については、『管理者ガイド』の「高可用性データベースの設定」および「セッションの持続性」の章を参照してください。

Sun ONE Application Server 7 はまた、第一世代の Sun ONE アプリケーションサーバー製品から大きく進歩したアーキテクチャを持っています。既存の強力な Sun ONE 製品およびテクノロジーと J2EE 1.4 標準を組み合わせた Sun ONE Application Server 7 のアーキテクチャは、様々なテクノロジーの基盤として最適なものです。

図 2-1 Sun ONE Application Server 7 Enterprise Edition のアーキテクチャ



上の図は、Sun ONE Application Server 7 アーキテクチャを図解したものであり、Sun ONE Application Server のコンポーネントアーキテクチャ、サブシステム、アクセスパス、および外部エンティティとコアサーバーのインタフェースが示されています。

Sun ONE Application Server 7 のアーキテクチャは高度にコンポーネント化されており、非常に管理しやすいものになっています。J2EE 仕様で要求されるすべてのサービスは、明確に定義された標準インタフェースとして提供され、アプリケーションから呼び出して使用できます。

Web ユーザーインタフェースにより、容易にリモートサーバー管理ができます。実際は 1 台の管理サーバーで複数のサーバーを管理することができる設計になっています。

Sun ONE Application Server のオペレーションで JDK 1.4 を使用することにより、このバージョンで拡張された機能を利用できるようになっています。

一般的な J2EE アプリケーションは n 層のシステムで構成され、クライアントは処理された情報を Web サーバーまたはアプリケーションサーバーから取得します。サーバーは RDBMS や ERP などの企業規模のシステムが持つ情報にアクセスし、ビジネスロジックで処理し、処理後の情報を適切な形式でクライアントに配信します。これらのレイヤはそれぞれクライアントレイヤ (Web ブラウザまたはリッチ Java クライアント)、中間レイヤ (Web サーバーおよびアプリケーションサーバー)、およびバックエンドレイヤまたはデータレイヤ (データベースなどの企業規模のシステム) に対応します。

Sun ONE Application Server の J2EE アプリケーションモデルを使用すれば、J2EE コンポーネントが低レベルの詳細に関するすべての処理を受け持つため、開発者はビジネスロジックの作成に集中することができます。したがって、アプリケーションおよびサービスを容易に拡張して短期間で配備し、競合相手の変化に素早く対応することができます。J2EE プラットフォームでオープンスタンダードアーキテクチャを提供している Sun ONE Application Server は、拡張性と高い可用性を持ち、安全で信頼性の高い多層サービスの開発における複雑さと費用の問題を解決します。

J2EE コンポーネント標準

Sun ONE Application Server 7 は J2EE 1.3 に準拠したサーバーです。Java コミュニティによって開発された、サーブレット、Java Server Pages、および Enterprise JavaBeans (EJB) 用のコンポーネント標準をベースにしています。

Sun ONE Application Server 7 以前の Sun ONE Application Server 6.0/6.5 は、J2EE 1.2 に準拠したサーバーです。2 つの J2EE バージョンの間には、J2EE アプリケーションコンポーネント API に関して大きな違いがあります。

J2EE 1.3 準拠の Sun ONE Application Server 7 Enterprise Edition および Standard Edition と、J2EE 1.2 準拠の Sun ONE Application Server 6.0/6.5 で使用されるコンポーネント API 間の相違点を次の表に示します。

表 2-1 J2EE コンポーネント用 API の Application Server 間でのバージョン比較

| コンポーネント API | Sun ONE Application Server 6.0/6.5 | Sun ONE Application Server 7, Enterprise Edition と Standard Edition |
|-------------|------------------------------------|------------------------------------------------------------------------|
| JDK | Sun 1.2.2 | Sun 1.4 |
| サーブレット | 2.2 | 2.3 |
| JSP | 1.1 | 1.2 |
| JDBC | 2.0 | 2.0 |
| EJB | 1.1 | 2.0 |
| JNDI | 1.2 | 1.2 |
| JMS | 1.0 | 1.1 |
| JTA | 1.0 | 1.01 |

さらに、この 2 つの製品では、XML 標準および Web サービスと関連のあるテクノロジーを多数サポートしています。これらは J2EE 仕様の一部ではないものの、企業アプリケーションでますます広く使用されるようになっていきます。これらの標準を次の表に示します。

表 2-2 Application Server がサポートするその他のテクノロジー

| テクノロジー | Sun ONE Application Server 6.0/6.5 | Sun ONE Application Server 7 |
|----------------------------|------------------------------------|------------------------------|
| XML 文書処理 (API および XML パーサ) | JAXP 1.0、Apache Xerces | JAXP 1.1 |

表 2-2 Application Server がサポートするその他のテクノロジー (続き)

| | | |
|--------------------------|-------------------------------|-----------------------------------------------|
| Web サービス用 SOAP/Java サポート | SOAP 1.1 (IBM SOAP4J フレームワーク) | Apache SOAP 2.2、JAX-RPC 1.0、JAXM 1.1、JAXR 1.0 |
|--------------------------|-------------------------------|-----------------------------------------------|

開発環境

この節では Sun ONE Application Server 6.0/6.5 と Sun ONE Application Server 7 の開発環境の違いを説明します。次の項目があります。

- [Sun ONE Application Server 6.0/6.5 の開発環境](#)
- [Sun ONE Application Server 7 の開発環境](#)

Sun ONE Application Server 6.0/6.5 の開発環境

Sun ONE Application Server 6.0/6.5 では Sun ONE Studio for Java の評価バージョンを提供しています。これは特にこのバージョンの Sun ONE Application Server のアプリケーション開発向けに調整されています。

これは NetBeans プラットフォームをベースにした完全な Java 開発環境です。この IDE は、Java アプリケーションおよび EJB コンポーネントの設計および開発のための、非常に広範囲の機能を提供します。また、プラグインによって Sun ONE Application Server と統合し、アプリケーションの様々な J2EE コンポーネントのアセンブリ、配備およびデバックに使用できます。Windows および Solaris のどちらの環境でも利用できます。

市場に出回っているサードパーティベンダー提供のソリューションの中では、最近リリースされた *Borland JBuilder 6 Enterprise* が非常に完成度が高く、利用範囲の広い製品です。Windows、Solaris、Linux、および MacOS X など複数のプラットフォームで動作するという利点もあります。JBuilder は、サーブレット、JSP ページ、EJB コンポーネント、グラフィックアプリケーションなどの Java の開発機能だけではなく、UML 設計、単体テスト、共同開発、および XML 開発の機能も提供しています。さらに、JBuilder を Sun ONE Application Server を含むメインストリームのアプリケーションサーバーと完全に統合し、Web アプリケーションおよび EJB コンポーネントのアセンブリ、配備およびデバックに使用できます。

Sun ONE Application Server 7 の開発環境

Sun ONE Application Server 7 をうまく使いこなせるかどうかは、完全に統合された開発環境を利用できるかどうかで決まります。Sun ONE Studio for Java Enterprise Edition 4 は Sun ONE アプリケーション開発のための重要なツールです。

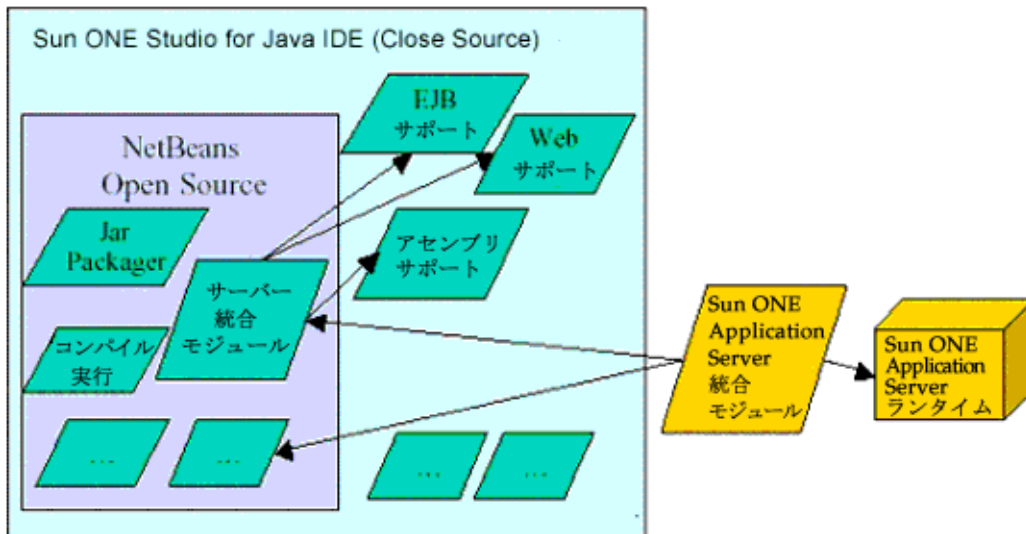
Sun ONE Studio for Java 4 は Sun ONE Application Server とともに提供されます。

Sun ONE Studio for Java Enterprise Edition 4 の中心的な機能を次に示します。

- EJB を短期間で容易に構築する機能
- EJB および配備用パッケージアプリケーションからアプリケーションをアセンブルする機能
- 配備のためのアプリケーションサーバーの統合
- Web サービスの開発およびパブリッシュ機能
- Java enterprise service presentation toolkit 用の Sun ONE studio
- Sun ONE Studio for Java Code Management Software (旧称 Forte TeamWare) による並行開発
- Sun ONE Application Server 7 との統合機能

次の図のように、Sun ONE Application Server 7 統合モジュールは、Sun ONE Studio Close Source で実装される NetBeans Open Source モジュールを基にしています。

図 2-2 Sun ONE Studio 4.0 Enterprise Edition と Sun ONE Application Server 7 の統合



管理ツール

この節では Sun ONE Application Server 6.0/6.5 と Sun ONE Application Server 7 の管理ツールの違いについて説明します。次の項目があります。

- [Sun ONE Application Server 6.0 の管理ツール](#)
- [Sun ONE Application Server 6.5 の管理ツール](#)
- [Sun ONE Application Server 7 の管理ツール](#)

Sun ONE Application Server 6.0 の管理ツール

Sun ONE Application Server 6.0 は、サーバー管理のあらゆる機能をカバーする、完全なグラフィカル管理ツールを提供しています。

- **Sun ONE Console:** 管理ツールのメインコントロールパネル。Sun ONE console で Administration Server Console、Directory Server、および Administration Tool へのすばやいアクセスが可能
- **Administration Server Console:** イベントログオプションの設定と SSL セキュリティ証明書の作成に使用
- **Sun ONE Directory Server Console:** Sun ONE Directory Server の管理に使用。Directory Server は、ユーザーおよび組織単位の管理に関する情報を持つユーザーディレクトリ、およびサーバー設定情報を持つ設定ディレクトリの、2 つの主要な情報ディレクトリツリーを管理
- **Sun ONE Administration Tool:** Sun ONE Application Server 6.0 の 1 つ以上のインスタンスを、配備済みのアプリケーションと共に管理するために使用。JDBC ドライバとデータソースの設定も行う
- **Sun ONE Registry Editor (*kregedit*):** Windows のレジストリエディタ (*regedit*) と似たグラフィカルなツール。特定のレジストリに格納された、Sun ONE Application Server 固有のパラメータ調整に使用

Sun ONE Application Server 6.5 の管理ツール

Sun ONE Application Server 6.5 は、統合されている Administration Tool、Sun ONE registry editor およびコマンド行ツールで管理できます。以下はその説明です。

- **Sun ONE Application Server Administration Tool:** グラフィカルユーザーインタフェースのスタンドアロンの Java アプリケーション。管理アプリケーションコンポーネントとともに使用し、Sun ONE Application Server の 1 つ以上のインスタンス管理が可能
- **コマンド行ツール :** Windows のコマンドプロンプト、および Solaris のシェルプロンプトから実行可能なツール。基本設定からアプリケーションの配備まで、様々なタスクを実行できる。コマンド行ツールに関する詳細を表示するには、コマンドプロンプトで [command] -help と入力する。ほとんどのツールは Sun ONE Application Server Administration Tool および Sun ONE Application Server Deployment Tool と統合されており、簡単に使用することができる
- **Sun ONE Registry Editor (*kregedit*):** Windows のレジストリエディタ (*regedit*) と似たスタンドアロンの GUI ツール。Sun ONE Application Server のレジストリ情報の表示および編集が可能

Sun ONE Application Server 7 の管理ツール

Sun ONE Application Server 7 の管理サーバーはサーバーの特別なインスタンスです。管理インタフェースとしての機能を持ち、すべてのサーバーインスタンスに共通のグローバル設定を管理します。Web ベースのサーバーであり、Sun ONE Application Server 設定用のフォームを持ちます。

アプリケーションサーバーを管理するためのグラフィカルツールです。エラーやアクセスログの表示、サーバーの稼動状況の監視、仮想サーバーの作成と編集、構成内容の変更、サーバーインスタンスの起動および停止などを行います。

Sun ONE Application Server をインストールする際に、管理サーバーのポート番号を選択していない場合は、デフォルトのポート番号 **4848** を使用しています。管理インタフェースにアクセスするには、Web ブラウザで次の URL を指定します。

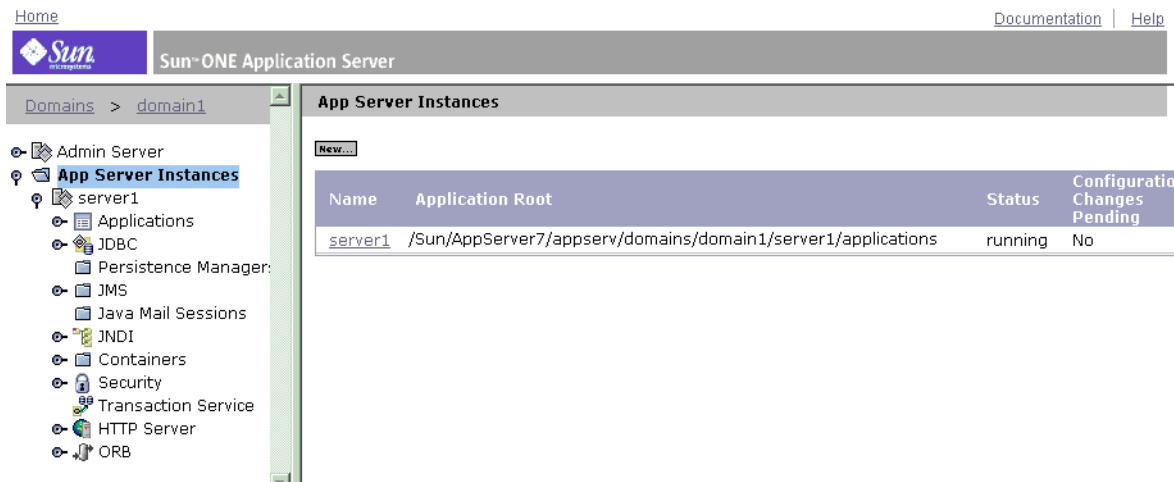
http://hostname:port/admin

設定済みのユーザー名とパスワードの入力が要求されます。入力して「OK (了解)」ボタンをクリックすると、次の図のような管理インタフェースのホームページが表示されます。

左側のペインには、Sun ONE Application Server で設定できるすべての項目がツリー表示されます。管理インタフェースを使用するには、この左側のペインのどれか 1 つの項目をクリックします。右側のペインに、クリックした項目に対応するページが表示されます。

管理インタフェースのどのページのヘルプも、管理インタフェースの最上部のバナーにある「**Help** (ヘルプ)」ボタンをクリックして表示できます。オンラインヘルプには、表示されているページの使用法や、ページ内のフィールドに入力する内容についての説明が記載されています。

図 2-3 管理インタフェースのホームページ



Sun ONE Application Server 7 はコマンド行インタフェースを装備しています。ユーティリティおよびコマンドを使用して、管理インタフェースと同じタスクを実行できます。コマンド行インタフェースは、シェル内のコマンドプロンプトから使用できるほか、スクリプトやプログラムから呼び出すこともできます。これらのコマンドを使用して、繰り返し行う管理タスクを自動化できます。

Sun ONE Application Server 7 のコマンド行ユーティリティ *asadmin* は、Windows のコマンドプロンプト、および Solaris のシェルプロンプトで実行できます。*asadmin* ユーティリティは、管理タスクを実行するためのコマンドセットです。これらのコマンドを使用して、基本設定からアプリケーション配備までのすべてのタスクを、管理インタフェースと同じように実行できます。コマンドの詳細を表示するには、*asadmin* ユーティリティに続いて *help* と入力します。

asadmin は、シングルモードまたはマルチモードで実行できます。シングルモードでは、コマンドプロンプトからコマンドを 1 つずつ実行します。マルチモードでは、環境レベル情報を再登録することなく、複数のコマンドを実行できます。

データベース接続

Sun ONE Application Server の各リリースでサポートされているデータベースの詳細な説明は、『プラットフォームの概要』を参照してください。

J2EE アプリケーションコンポーネントと移行

J2EE は、標準化されたモジュールコンポーネントをベースとして、これらのコンポーネントへの一連のサービスを提供し、アプリケーションの詳細な動作の多くを自動処理することにより、企業アプリケーションの開発を単純化します。複雑なプログラミングは必要ありません。J2EE 1.3 アーキテクチャには、さまざまなコンポーネント API が含まれます。主な J2EE API を次に示します。

- サブレット
- Java Server Pages (JSP)
- EJB。メッセージ駆動型 Beans (MDB) を含む
- Java Database Connectivity (JDBC)
- Java Transaction Service (JTS)
- Java Naming and Directory Interface (JNDI)
- Java Message Service (JMS)

J2EE コンポーネントは個別にパッケージ化され、配備用 J2EE アプリケーションにバンドルされます。各コンポーネント、GIF および HTML ファイルやサーバーサイドユーティリティクラスなどの関連ファイル、および配備記述子は、1 つのモジュールとしてアSEMBルされ、J2EE アプリケーションに追加されます。J2EE アプリケーションは、1 つまたは複数の Enterprise JavaBeans、Web、またはアプリケーションクライアントコンポーネントモジュールで構成されます。最終的な企業用ソリューションは、設計要件に応じて 1 つの J2EE アプリケーションを使用することも、2 つ以上の J2EE アプリケーションで構成することもできます。

J2EE アプリケーションおよびその各モジュールは、独自の配備記述子を備えています。配備記述子は、拡張子 .xml を持つ XML ドキュメントであり、コンポーネントの配備設定を記述します。たとえば、Enterprise JavaBean モジュールの配備記述子は、Enterprise JavaBean 用のトランザクション属性およびセキュリティ認証を宣言します。配備記述子情報が宣言されるため、Bean ソースコードを修正しなくても変更できます。J2EE サーバーは、実行時に配備記述子を読み込み、そのコンポーネント上で適切に動作します。

J2EE アプリケーションとそのすべてのモジュールは、EAR ファイル (Enterprise アーカイブファイル) で配布されます。EAR ファイルは、拡張子 .ear を持つ標準 JAR ファイル (Java アーカイブファイル) です。EAR ファイルには、EJB JAR ファイル、アプリケーションクライアント JAR ファイル、あるいは WAR ファイル (Web アーカイブファイル) があります。これらのファイルの特徴を次に示します。

- 各 EJB JAR ファイルは、配備記述子、Enterprise JavaBeans ファイル、および関連ファイルを含む
- 各アプリケーションクライアントの JAR ファイルは、配備記述子、アプリケーションクライアントのクラスファイル、および関連ファイルを含む
- 各 WAR ファイルは、配備記述子、Web コンポーネントファイル、および関連リソースを含む

モジュールおよび EAR ファイルを使用すると、同一コンポーネントのいくつかを使用して、種類の異なる多数の J2EE アプリケーションをアセンブルできます。追加のコーディングは必要ありません。さまざまな J2EE モジュールを単純に J2EE EAR ファイルにアセンブルするだけです。

移行プロセスでは、主に J2EE アプリケーションコンポーネント、モジュール、およびファイルの移動を行います。

各種の J2EE コンポーネントの移行に関する詳細は、[第 3 章「Sun ONE Application Server 6.x から Sun ONE Application Server 7 への移行」](#)を参照してください。

J2EE の基本情報の詳細については、以下を参照してください。

- J2EE チュートリアル - <http://java.sun.com/j2ee/tutorial/>
- J2EE の概要 - <http://java.sun.com/j2ee/overview.html>
- J2EE トピック - <http://java.sun.com/j2ee>

Sun ONE Application Server 6.x から Sun ONE Application Server 7 への移行

この章では、Sun ONE Application Server 6.0 または 6.5 から Sun ONE Application Server 7 製品ラインに、J2EE アプリケーションを移行する場合の注意事項と方針について説明します。

この節では、コンポーネントレベルでの特定の移行作業についても説明します。

次の項目について説明します。

- [Sun ONE Application Server 6.0/6.5 について](#)
- [Sun ONE Application Server 6.x から 7 への移行に関する問題](#)
- [J2EE コンポーネントの移行](#)
- [Web アプリケーションの移行](#)
- [エンタープライズ EJB モジュールの移行](#)
- [エンタープライズアプリケーションの移行](#)
- [固有の拡張子の移行](#)
- [UIF の移行](#)

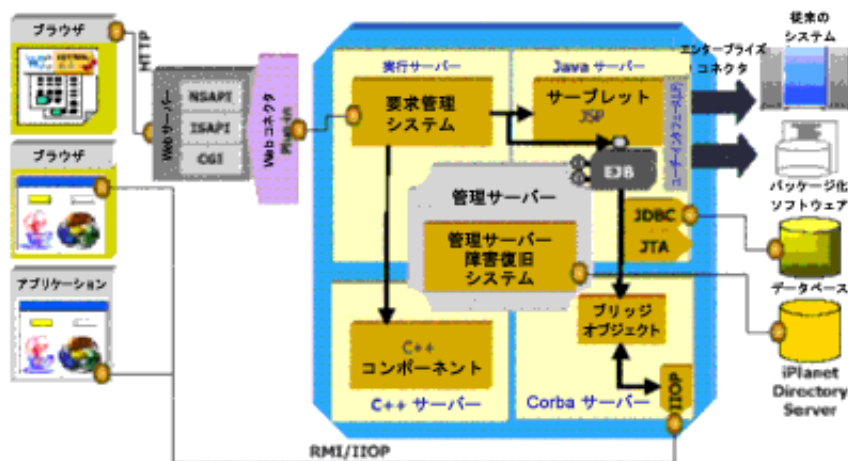
Sun ONE Application Server 6.0/6.5 について

Sun ONE Application Server 6.0 は、J2EE 1.2 仕様を全面的にベースにしたマルチプラットフォームのアプリケーションサーバーです。Windows NT/2000、Solaris、AIX、HP-UX などのプラットフォームがサポートされています。

また、Sun ONE Application Server 6.0 は、付属の専用 Web コネクタプラグインによって、多数の Web サーバーを統合化します。これらのコネクタによって、Sun ONE Web Server、Microsoft IIS、Apache などの Web サーバーと連携して動作することが可能になります。

次の図は、Sun ONE Application Server 6.0/6.5 のアーキテクチャを示しています。

図 3-1 Sun ONE Application Server 6.0/6.5 のアーキテクチャ



上の図に示されるように、一般的にエンジンまたはプロセスと呼ばれる 4 つの内部サーバーが存在します。これらのプロセスが、Sun ONE Application Server でのすべての処理を実行します。Sun ONE Application Server 6.0/6.5 の 4 つの内部サーバーには、次のようなものがあります。

実行サーバー：ほとんどのシステムサービスを提供する（一部のサービスは、「管理サーバー」によって管理される）

管理サーバー：Sun ONE Application Server の管理および障害復旧用のシステムサービスを提供する

Java サーバー：Java アプリケーションのサービスを提供する

C++ サーバー：C++ で作成されたコンポーネントは、「C++ サーバー」で動作する

Web サーバーが Sun ONE Application Server 6.0/6.5 に要求を転送すると、要求はまず「実行サーバー」プロセス (KXS) によって受信されます。KXS プロセスは、その要求を「Java サーバー」プロセス (KJS) または「C++ サーバー」プロセス (KCS) のどちらかに転送します。KJS プロセスが Java プログラミングロジックを実行するのに対し、KCS プロセスは C++ プログラミングロジックを実行します。KJS プロセスと KCS プロセスはそれぞれ、特定数のスレッドを管理し、それらのスレッドでプログラミングロジックを最後まで実行します。結果は Web サーバーに返されてから、クライアントブラウザに送信されます。

配備記述子の移行

次の表は、配備記述子の対応関係をまとめたものです。

| ソース配備記述子 | ターゲット配備記述子 |
|-------------------------|----------------------|
| ejb-jar.xml -1.1 | ejb-jar.xml -2.0 |
| ias-ejb-jar.xml | sun-ejb-jar.xml |
| <bean-name>-ias-cmp.xml | sun-cmp-mappings.xml |
| web.xml | web.xml |
| ias-web.xml | sun-web.xml |
| application.xml | application.xml |

J2EE 標準配備記述子 ejb-jar.xml、web.xml、および application.xml に大きな違いはありません。一方、ejb-jar.xml 配備記述子は、Sun ONE Application Server 7 EE 上にアプリケーションを配備できるようにするため、EJB 2.0 準拠の仕様に変更されています。

sun-ejb-jar.xml および sun-web.xml の作成に必要な情報の大部分は、それぞれ ias-ejb-jar.xml および ias-web.xml に基づいています。ただし、移行対象の sun-ejb-jar.xml に宣言がある場合、一部の必要な情報が CMP エンティティ Bean のホームインタフェース (java ファイル) から抽出されます。これは、CMP エンティティ Bean のホームインタフェースの情報を必要とする sun-ejb-jar.xml 内に、<query-filter> コンストラクトを構築するために必要な条件です。移行時にこのソースファイルが見つからない場合、<query-filter> コンストラクトが作成されますが、多くの情報が失われます (移行後の sun-ejb-jar.xml 内の "REPLACE ME" で示される部分)。

さらに、ias-ejb-jar.xml に <message-driven> 要素が含まれる場合、この要素の内部の情報が ejb-jar.xml と sun-ejb-jar.xml の内部に取り込まれます。また、ias-ejb-jar.xml の <message-driven> 要素の内部には <destination-name> という要素があり、MDB が待機するトピックまたはキューの JNDI 名はここに格納されます。Sun ONE Application Server 6.5 の場合、この JNDI 名の命名規則は "cn=<SOME_NAME>" のようになります。この名前の JMS トピックまたはキューは Sun ONE Application Server 7 EE に配備できません。そこで、これを "<SOME_NAME>" という名前に変更し、この情報を sun-ejb-jar.xml に追加します。すべての有効な入力ファイル (.java ファイル、.jsp ファイル、および .xml ファイル) で、この変更を適用する必要があります。JNDI 名の変更は、アプリケーション全体にグローバルに影響を及ぼします。この JNDI 名を参照するソースファイルがあり、使用できない状態になっている場合は、該当ソースファイルに手動で変更を加え、アプリケーションを配備できるようにする必要があります。

Sun ONE Application Server 6.x から 7 への移行に関する問題

この節では、Sun ONE Application 6.0 または 6.5 から Sun ONE Application Server 7 に、標準的な J2EE アプリケーションの主要コンポーネントを移行する場合に生じる可能性のある問題について説明します。

この節で説明する移行時の問題点は、オンラインバンキングサービスをシミュレートする iBank という名前の J2EE アプリケーションを Sun ONE Application Server 6.0 または 6.5 から Sun ONE Application Server 7 に、実際に移行したときのプロセスに基づいています。このアプリケーションは、通常の J2EE アプリケーションを構成するすべての要素を反映しています。

iBank アプリケーションに適用される J2EE 仕様の重要なポイントとして、次のようなものが挙げられます。

- サブレット、特に JSP ページへの切り替えを行うサブレット (model-view-controller アーキテクチャ)
- JSP ページ、特に静的および動的なページの取り込みを含む JSP ページ
- JSP カスタムタグライブラリ
- HTTP セッションの作成および管理
- JDBC API 経由のデータベースアクセス
- Enterprise JavaBeans: ステートフル、またはステートレスなセッション Beans、CMP、および BMP エンティティ Beans

- J2EE アプリケーションの標準パッケージ化メソッドに対応したアセンブリおよび配備

iBank アプリケーションの詳細については、付録 A「iBank アプリケーションの仕様」を参照してください。

J2EE コンポーネントの移行

この節では次の移行プロセスについて説明します。

- [JDBC コードの移行](#)
- [Java Server Pages および JSP カスタムタグライブラリの移行](#)
- [サーブレットの移行](#)
- [JNDI コンテキストからのデータソースの取得](#)
- [EJB の移行](#)
- [Sun ONE Application Server 7 に対応した EJB の変更](#)

JDBC コードの移行

JDBC API を使用したデータベースアクセスには、2 つの方法があります。

- [DriverManager インタフェース経由の接続の確立](#)
(JDBC 1.0 API)。特定のドライバを読み込み、接続 URL を提供します。この方法は、IBM の WebSphere 4.0 のような他のアプリケーションサーバーで使用されます。
- [JDBC 2.0 データソースの使用方法](#)
データソースインタフェース (JDBC 2.0 API) は、設定可能な接続プールを介して使用できます。J2EE 1.2 に従って、データソースは JNDI ネーミングサービス経由でアクセスされます。

| | |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 注 | Sun ONE Application Server 7 は、Sun ONE Application Server 6.x にバンドルされている Native Type 2 JDBC ドライバをサポートしません。Type 2 のドライバを使用するコードは、サードパーティの JDBC ドライバを使用するように手動で移行する必要があります。 |
|---|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

DriverManager インタフェース経由の接続の確立

この方法でデータベースにアクセスするのは、古い方法であり効率的でないためお勧めしませんが、まだこの方法を使用しているアプリケーションもいくつか存在します。

この場合、アクセスコードは、以下のようになります。

```
public static final String driver =
"oracle.jdbc.driver.OracleDriver";

public static final String url =
"jdbc:oracle:thin:tmb_user/tmb_user@iben:1521:tmbank";

Class.forName(driver).newInstance();

Properties props = new Properties();

props.setProperty("user", "tmb_user");

props.setProperty("password", "tmb_user");

Connection conn = DriverManager.getConnection(url, props);
```

このコードは、Sun ONE Application Server 6.0/6.5 から Sun ONE Application Server 7 に完全に移植できます。ただし、Sun ONE Application Server が適切な JDBC ドライバを読み込むのに必要なクラスを検出できることが前提になります。Sun ONE Application Server 7 に配備されたアプリケーションにアクセスできる、必要なクラスを作成するには、次のいずれかの方法を実行します。

- Sun ONE Application Server 7 のインストールディレクトリ内の *lib* ディレクトリに、ドライバ実装用のアーカイブ (JAR または ZIP) を置きます。

管理サーバーの GUI を介して、ドライバのパスを設定し、*CLASSPATH* を修正します。サーバーインスタンス「server1」をクリックした後、右側のペインの「JVM Settings (JVM 設定)」タブをクリックします。次に、「Path Settings (パス設定)」オプションをクリックし、テキスト入力ボックスの「Classpath Suffix (クラスパスのサフィックス)」にパスを追加します。設定変更を行ったら、「Save (保存)」をクリックして、新しい設定を適用します。サーバーを再起動すると、設定ファイル *server.xml* が変更されます。

JDBC 2.0 データソースの使用法

データベースへのアクセスに JDBC 2.0 データソースを使用すると、透過的な接続プーリングなど、パフォーマンス上の利点があります。コードや実装を単純化することで、生産性を向上させ、コードの移植性を高めることができます。

アプリケーション内のデータソースを使用するには、初期設定のフェーズを実行する必要があります。その後で、アプリケーションサーバーの JNDI ネーミングコンテキスト内でデータソースを登録します。データソースを登録すると、アプリケーションでは、JNDI コンテキストから対応する *DataSource* オブジェクトを取り出すことによって、データベースへの接続を簡単に取得できます。これらの作業については、次の項目で説明します。

- データソースの設定
- JNDI 経由でのデータソースの検索および接続の取得

データソースの設定

Sun ONE Application Server 6.0 では、データソースとそれに対応する JDBC ドライバを、サーバーのグラフィック管理コンソールから設定します。接続プールは、アプリケーションサーバーによって自動的に管理され、その管理ツールはプロパティの設定に使用できます。組み込まれている Type 2 の JDBC ドライバを使用すると、接続プールのプロパティは、ドライバごとに定義され、そのドライバを使用するすべてのデータソースで共有されます。

一方、サードパーティの JDBC ドライバを使用した場合、接続プールのプロパティは、データソースごとに定義されます。サードパーティの JDBC ドライバは、管理ツールから、または別のユーティリティ (Sun Solaris では `db_setup.sh`、Windows NT/2000 では `jdbcsetup`) から設定できます。さらに、コマンド行ユーティリティ `iasdeploy` を使用すると、プロパティを記述している XML ファイルからデータソースを設定できます。これらのユーティリティはすべて、Sun ONE Application Server のインストールルートディレクトリ内のサブディレクトリ `/bin/` に格納されています。

Sun ONE Application Server 7 では、サーバーのグラフィック管理コンソールまたはコマンド行ユーティリティ `asadmin` を使用して、データソースを設定できます。コマンド行ユーティリティ `asadmin` を起動するには、Solaris の `asadmin` ファイルを実行します。このファイルは、Sun ONE Application Server 7 のインストールディレクトリ内の `bin` ディレクトリに格納されています。`asadmin` プロンプトから次のコマンドを入力すると、接続プールと JNDI リソースが作成されます。

`asadmin` ユーティリティを起動して、接続プールを作成するための構文は、次のとおりです。

```
asadmin>create-jdbc-connection-pool -u username -w password -H
hostname -p adminport [-s] [--instance instancename]
--datasourceclassname classname [--steadypoolsize=8]
[--maxpoolsize=32] [--maxwait=60000] [--poolresize=2]
[--idletimeout=300] [--isconnectvalidatereq=false]
[--validationmethod=auto-commit] [--validationtable tablename]
[--failconnection=false] [--description text] [--property
(name=value)[:name=value]*] connectionpoolid
```

次に例を示します。

```
asadmin>create-jdbc-connection-pool -u admin -w password -H c11
-p 4848 -instance server1 --datasourceclassname
oracle.jdbc.pool.OracleConnectionPoolDataSource --property
(user-name=ibank_user):(password=ibank_user) oraclepool
```

この場合、Oracle データベース用の JDBC 接続プール「oraclepool」は、「ibank_user」というユーザー名と「ibank_user」というパスワードを持つデータベーススキーマを使用して作成されます。

JDBC リソースを作成する構文は、次のとおりです。

```
asadmin>create-jdbc-resource -u username -w password -H hostname
-p adminport [-s] [--instance instancename] --connectionpoolid id
[--enabled=true] [--description text] [--property
(name=value)[:name=value]*] jndiname
```

次に例を示します。

```
asadmin>create-jdbc-resource -u admin -w password -H c11 -p 4848
--instance server1 --connectionpoolid oraclepool jdbc/IBANK
```

この場合、上記の「jdbc/IBANK」という JNDI 名で作成された接続プールに対して、JDBC リソースが作成されます。

グラフィカルインタフェースを使用して、Sun ONE Application Server 7 にデータソースを登録する場合の実行手順は以下のとおりです。

1. データソースのクラス名を登録します。
 - a. Sun ONE Application Server 7 のインストールディレクトリ内の *lib* ディレクトリに、データソースのクラス実装用のアーカイブ (JAR または ZIP) を置きます。

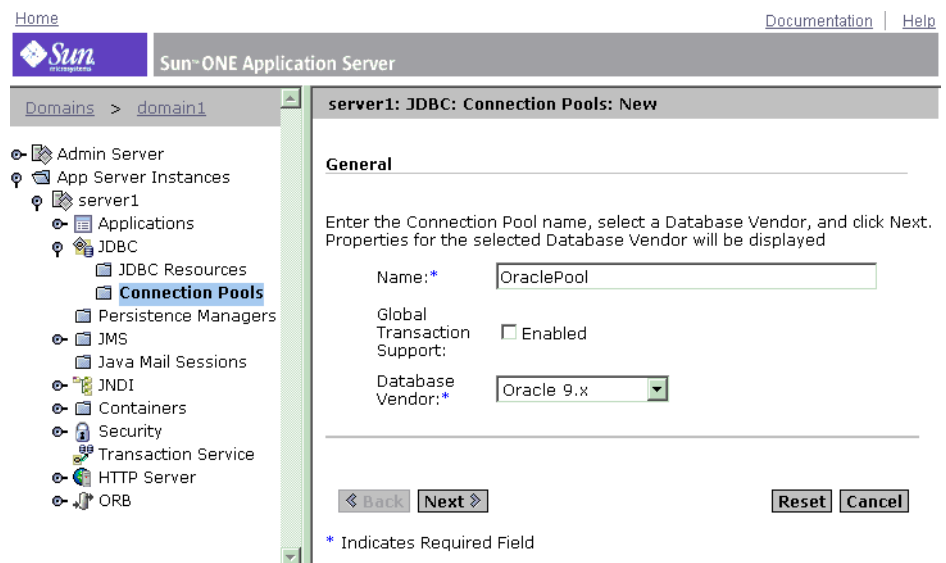
- b. 管理サーバーの GUI を介して、ドライバのパスを設定し、CLASSPATH を修正します。サーバーインスタンス「server1」をクリックした後、「JVM Settings (JVM 設定)」タブをクリックします。次に、「Path Settings (パス設定)」をクリックして、「Classpath Suffix (クラスパスのサフィックス)」にパスを追加します。設定変更を行ったら、変更内容を保存して、新しい設定を適用します。サーバーを再起動すると、設定ファイル server.xml が変更されます。

2. データソースを登録します。

Sun ONE Application Server 7 では、データソースとそれに対応する JDBC ドライバを、サーバーのグラフィック管理インタフェースから設定します。

左側のペインには、Sun ONE Application Server で設定できるすべての項目がツリー表示されます。左側のペインにある「Connection Pool (接続プール)」をクリックすると、関連するエントリを指定できるページが右側のペインに表示されます。

図 3-2 GUI を使用した接続プールの設定



同様に、「Data Source (データソース)」をクリックすると、データソースのセットアップに必要なエントリが右側のペインに表示されます。

Sun ONE Application Server 7 固有の配備記述子 *sun-web.xml* は、それに応じて変更する必要があります。

たとえば、iBank アプリケーションに新しいデータソースを設定する場合、*sun-web.xml* は次のようなエントリを保持します。

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "Http://localhost:8000/sun-web-app_2_3.dtd">

<sun-web-app>

    <resource-ref>

        <res-ref-name>jdbc/iBank</res-ref-name>

        <jndi-name>jdbc/iBank</jndi-name>

        <default-resource-principal>

            <name>ibank_user</name>

            <password>ibank_user</password>

        </default-resource-principal>

    </resource-ref>

</sun-web-app>
```

JNDI 経由でのデータソースの検索および接続の取得

データソースから接続を取得するためのプロセスは、以下のようになります。

- 初期 JNDI コンテキストを取得する
- JNDI 検索を使用して、データソースへの参照を取得する
- この参照を使用して、接続を取得する

1. 初期 JNDI コンテキストを取得する

異なる環境間での移植性を保証するため、`InitialContext` オブジェクト (サブレット、JSP ページ、または EJB 内にある) を取得するのに使用するコードは、次のような単純なものにする必要があります。

```
InitialContext ctx = new InitialContext();
```

2. データソースへの参照を取得する

JNDI コンテキストにバインドされたデータソースへの参照を取得するには、初期のコンテキストオブジェクトから、データソースの JNDI 名を検索します。次に、この方法で検索されたオブジェクトを、`DataSource` タイプオブジェクトとしてキャストします。

```
ds = (DataSource)ctx.lookup(JndiDataSourceName);
```

3. 接続を取得する

この操作は非常に単純なもので、次のようなコード行を指定する必要があります。

```
conn = ds.getConnection();
```

Sun ONE Application Server 6.0/6.5 および 7 はどちらも上記の技術に従って、データソースから接続を取得します。つまり、移行を行うときにコードを変更する必要はありません。

Java Server Pages および JSP カスタムタグライブラリの移行

Sun ONE Application Server 6.0/6.5 が JSP 1.1 仕様に準拠しているのに対し、Sun ONE Application Server 7 は JSP 1.2 仕様に準拠しています。

JSP 1.2 仕様には、多数の新機能に加えて、JSP 1.1 仕様ではあまり適切ではなかった部分の修正および説明が含まれています。

最も重要な変更点を以下に示します。

- JSP 1.2 は、サーブレット 2.3 および Java 2 に基づいている。JSP 1.2 アプリケーションは、JDK 1.1 だけをサポートするプラットフォームでは動作しない。JSP 1.2 は、JSP 1.1 との下位互換性があるため、JSP 1.1 アプリケーションは JSP 1.2 に準拠するコンテナを調整しなくても動作する
- JSP ページ用の XML 構文の定義が確定している。このため、JSP 1.2 に準拠するコンテナは、JSP 1.1 形式と JSP Document と呼ばれる新しい XML 形式の両方のファイルを受け入れる必要がある
- タグライブラリでは、サーブレット 2.3 のイベントリスナーを利用できる
- タグライブラリに、JSP ページを検証する新しいタイプの検証が追加されている
- タグライブラリの配布および配備用の新しいオプションが追加されている

基本的に、これらの変更は機能の拡張なので、JSP ページを JSP API 1.1 から 1.2 に移行する場合には必要ありません。

Sun ONE Application Server 6.0 または 6.5 の JSP カスタムタグライブラリの実装は、J2EE 仕様に準拠します。このため、JSP カスタムタグライブラリを Sun ONE Application Server 7 に移行する場合に、特別な問題が生じたり、修正が必要になることはありません。

サーブレットの移行

Sun ONE Application Server 6.0 および 6.5 で、サーブレット 2.2 API がサポートされているのに対し、Sun ONE Application Server 7 では、サーブレット 2.3 API がサポートされます。

サーブレット API 2.3 では、サーブレットのコアは比較的变化を受けずに残されており、ほとんどの変更は、コアの外側に追加された新しい機能に関係しています。

最も重要な機能を以下に示します。

- サーブレットには JDK 1.2 以降が必要
- フィルタメカニズムを新たに作成
- アプリケーションライフサイクルイベントを追加
- 新しい国際化サポートを追加
- 新しいエラーおよびセキュリティ属性を追加
- HttpUtils クラスを廃止
- 複数の DTD 動作を拡張、明確化

基本的に、これらの変更は機能の拡張なので、サーブレットをサーブレット API 2.2 から 2.3 に移行する場合には必要ありません。

ただし、アプリケーション内のサーブレットが JNDI を使用して J2EE アプリケーションのリソース (データソース、EJB など) にアクセスする場合、ソースファイルまたは配備記述子の変更が必要な場合があります。

これらの変更の詳細については、次の節で説明します。

- [JNDI コンテキストからのデータソースの取得](#)
- [JNDI コンテキスト内での EJB の宣言](#)

最後のケースとして、JSP ページが既存の Java クラスと同じ名前を持つ場合に、Sun ONE Application Server で命名の競合が発生するため、サーブレットのコード修正が必要になることがあります。この場合、競合は問題の生じている JSP ページの名前を修正することで解決します。また、さらに、この JSP ページを呼び出すサーブレットのコードを編集する必要があります。Sun ONE Application Server 6.0/6.5 と比べて、Sun ONE Application Server 7 では新しいクラスローダ階層が採用されているため、この問題は解決しています。この新しい方式では、特定のアプリケーションについて、1 つのクラスローダがすべての EJB モジュールを読み込み、別のクラスローダが Web モジュールを読み込みます。これらの 2 つのローダ同士は通信しないため、命名の競合が発生することはありません。

JNDI コンテキストからのデータソースの取得

JNDI コンテキストにバインドされたデータソースへの参照を取得するには、初期のコンテキストオブジェクトから、データソースの JNDI 名を検索します。次に、この方法で検索されたオブジェクトを、DataSource タイプオブジェクトとしてキャストします。

```
ds = (DataSource) ctx.lookup(JndiDataSourceName);
```

詳細については、前述の「[JDBC コードの移行](#)」を参照してください。

JNDI コンテキスト内での EJB の宣言

159 ページの「[Enterprise Java Beans 1.1 仕様から Enterprise Java Beans 2.0 仕様への移行](#)」の「[JNDI コンテキスト内での EJB の宣言](#)」の節を参照してください。

EJB の移行

「[Sun ONE Application Server 7, Enterprise Edition の概要](#)」で説明したように、Sun ONE Application Server 6.0 および 6.5 で EJB 1.1 仕様がサポートされているのに対し、Sun ONE Application Server 7 では EJB 2.0 仕様もサポートされます。EJB 2.0 仕様は、次のような新機能をアーキテクチャに導入しています。

- メッセージ駆動型 Beans (MDB)
- コンテナ管理による持続性 (CMP) の向上
- CMP によるエンティティ Beans のコンテナ管理関係
- ローカルインタフェース
- EJB クエリ言語 (EJB QL)

Sun ONE Application Server 7 でも、EJB 1.1 仕様は引き続きサポートされますが、その拡張機能を活用するためには、EJB 2.0 アーキテクチャの使用をお勧めします。

EJB 1.1 から EJB 2.0 への移行については、[付録 C 「Enterprise Java Beans 1.1 仕様から Enterprise Java Beans 2.0 仕様への移行」](#)を参照してください。

Sun ONE Application Server 7 に対応した EJB の変更

Sun ONE Application Server 6.0/6.5 から Sun ONE Application Server 7 に EJB を移行する場合、EJB のコード自体を変更する必要はありません。ただし、次のような DTD の変更が必要となります。

セッション Beans

- `ejb-jar.xml` など、J2EE 標準 DD を持つ最新の DTD を示すように、`<!DOCTYPE>` 定義を変更します。
- `ias-ejb-jar.xml` ファイルを、DD に従って手動で作成した `sun-ejb-jar.xml` という名前の修正バージョンのファイルに置き換えます。詳細については、http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-0.dtd を参照してください。
- `sun-ejb-jar.xml` の場合、すべての EJB の JNDI 名は `ejb/` で開始する必要があります。これは Sun ONE Application Server 6.5 の場合に必要です。EJB の JNDI 名には、`ejb/<ejb-name>` しか指定できません。`<ejb-name>` には、`ejb-jar.xml` 内で宣言した EJB の名前が入ります。

Sun ONE Application Server 7 では、EJB の JNDI 名を宣言できる新しいタグが `sun-ejb-jar.xml` 内に導入されました。

| | |
|---|--------------------------------------------------------------------------------------------------------------------------------------|
| 注 | アプリケーション内で JNDI 名が変更されるのを防ぐため、EJB の JNDI 名を <code><jndi-name></code> タグ内で、 <code>ejb/<ejb-name></code> として宣言することをお勧めします。 |
|---|--------------------------------------------------------------------------------------------------------------------------------------|

エンティティ Beans

- `ejb-jar.xml` など、J2EE 標準 DD を持つ最新の DTD を示すように、`<!DOCTYPE>` 定義を変更します。
- `ejb-jar.xml` のすべての CMP に対して、1.1 の値を指定した `<cmp-version>` タグを挿入します。
- すべての `<ejb-name>`-`ias-cmp.xml` ファイルを、手動で作成した `sun-cmp-mappings.xml` ファイルに置き換えます。詳細については、http://www.sun.com/software/sunone/appserver/dtds/sun-cmp_mapping_1_0.dtd を参照してください。
- Sun ONE Application Server 7 のインストールディレクトリ内の `bin` ディレクトリに格納されている `capture-schema` ユーティリティを使用して `dbschema` を生成し、エンティティ Beans 用の `META-INF` フォルダより上位に置きます。

- `ias-ejb-jar.xml` を、Sun ONE Application Server 7 で `sun-ejb-jar.xml` という名前に変更された新しいバージョンのファイルに置き換えます。
- Sun ONE Application Server 6.5 では、ファインダ `sql` が `<ejb-name>-ias-cmp.xml` に直接組み込まれていました。Sun ONE Application Server 7 では、数学式を使用して、さまざまな検索メソッド用の `<query-filter>` を宣言するように変更されています。

メッセージ駆動型 Beans

Sun ONE Application Server 7, Enterprise Edition は、メッセージ駆動をシームレスにサポートします。このサポートは、Sun ONE Application Server 7, Enterprise Edition と Sun ONE Message Queue の緊密な統合によって実現されています。これにより、ネイティブの組み込み型 JMS サービスが提供されます。

MQ をインストールすると、Sun ONE Application Server に、任意の数の Sun ONE Application Server インスタンスをサポートする JMS メッセージングシステムが提供されます。各サーバーインスタンスには、デフォルトで、そのインスタンスで実行中のすべての JMS クライアントをサポートする組み込み JMS サービスが割り当てられています。

『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。

iPlanet Application Server のメッセージ駆動型 Beans サポートは、開発者に限定されており、以前の専用 API を多用していました。メッセージングサービスは、iPlanet Message Queue for Java 2.0 によって提供されていました。Queue Connection Factory オブジェクトを設定するため、iPlanet Application Server の下に LDAP ディレクトリも必要でした。

`ejb-jar.xml` ファイルには、QueueConnectionFactory と、Sun ONE Application Server 内でメッセージ駆動型 Beans を設定するために必要なその他の情報を指定する必要があります。

配備記述子の変更の詳細については、「[配備記述子の移行](#)」を参照してください。Sun ONE Application Server 7 のメッセージ駆動型 Beans の実装については、『Sun ONE Application Server 7, Enterprise Edition, Developer's Guide to Enterprise Java Bean Technology』を参照してください。

Web アプリケーションの移行

Sun ONE Application Server 6.0 と 6.5 は、Servlet API 2.2 と JSP 1.1 をサポートします。一方、Sun ONE Application Server 7 は、Servlet API 2.3 と JSP 1.2 をサポートします。

これらの環境内では、異なるアプリケーションコンポーネント (サブレット、JSP、HTML ページ、およびその他のリソース) を 1 つのアーカイブファイル (J2EE 標準 Web アプリケーションモジュール) にまとめてから、そのアーカイブファイルをアプリケーションサーバー上に配備する必要があります。

J2EE 1.3 仕様によると、Web アプリケーションは、次のような構造を持ったアーカイブファイル (WAR ファイル) です。

- HTML ページ、JSP、画像、およびその他の「静的な」アプリケーションのリソースを含むルートディレクトリ
- 使用している SDK のバージョン情報、およびアーカイブに含まれるファイルのリスト (オプション) が入ったアーカイブマニフェストファイル (MANIFEST.MF) を含む *META-INF/* ディレクトリ
- アプリケーション配備記述子 (*web.xml* ファイル)、およびアプリケーションで 사용되는すべての Java クラスとライブラリを含む *WEB-INF/* ディレクトリ。次のように構成される
 - サブディレクトリ *classes/* には、アプリケーションのコンパイル済みのクラス (サブレット、補助クラスなど) がツリー構造で格納され、パッケージに編成される
 - *lib/* ディレクトリには、アプリケーションによって使用される任意の Java ライブラリ (*jar* ファイル) が含まれる

Web アプリケーションモジュールの移行

Sun ONE Application Server 6.0/6.5 から Sun ONE Application Server 7 にアプリケーションを移行する場合、Java/JSP のコードを変更する必要はありません。ただし、次のような変更が必要となります。

- *web.xml*

Sun ONE Application Server 7 は J2EE 1.3 標準に準拠しているため、WAR 内の *web.xml* ファイルは、改訂された DTD に従う必要があります。この改訂版 DTD は、http://java.sun.com/dtd/web-app_2_3.dtd から入手できます。幸い、この DTD は以前のバージョンの DTD のスーパーセットなので、移行する *web.xml* 内の `<!DOCTYPE` 定義を変更するだけで済みます。修正した `<!DOCTYPE` 宣言は、次のようになります。

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
```

- **ias-web.xml**

Sun ONE Application Server 7 では、このファイルの名前は `sun-web.xml` に変更されています。

この XML ファイルは、Web アプリケーションに必要な Sun ONE Application Server 7 固有のプロパティまたはリソースを宣言するときに使用します。

注：このファイルの内容については、次の節を参照してください。

Sun ONE Application Server 6.5 アプリケーションの `ias-web.xml` が存在し、Sun ONE Application Server 6.5 固有のプロパティが宣言されている場合、Sun ONE Application Server 7 標準に移行するときに、このファイルが必要となります。DTD ファイル名を `sun-web.xml` に変更する必要があります。詳細については、
http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-0.dtd
 を参照してください。

前述した方法で `web.xml` と `ias-web.xml` を移行すると、管理サーバーの Sun ONE Application Server 7 の GUI インタフェース、またはコマンド行ユーティリティ `asadmin` を使用して、Web アプリケーション (.WAR アーカイブ) を配備することができます。配備コマンドでは、アプリケーションのタイプを `web` として指定する必要があります。

コマンド行ユーティリティ `asadmin` を起動するには、Sun ONE Application Server 7 のインストールディレクトリ内の `bin` ディレクトリに格納されている `asadmin.bat` ファイルを実行します。

`asadmin` プロンプトから入力するコマンドは、次のようになります。

```
asadmin> deploy -u username -w password -H hostname -p adminport
--type web [--contextroot contextroot] [--force=true] [--name
component-name] [--upload=true] [--instance instancename]
filepath
```

133 ページの「[Sun ONE Application Server 7 でのアプリケーションの配備](#)」に記載されるように、配備は Sun ONE Studio 開発環境からでも行うことができます。

サーブレットおよび JSP の移行時の特定の障害

Sun ONE Application Server 6.0/6.5 から Sun ONE Application Server 7 に、サーブレット /JSP アプリケーションのコンポーネントを実際に移行する場合、コンポーネントのコードを修正する必要はありません。

Web アプリケーションがデータソースなどのサーバーリソースを使用している場合、Sun ONE Application Server 7 では、このリソースを web.xml 内および sun-web.xml 内で同様に宣言する必要があります。jdbc/iBank という名前のデータソースを宣言する場合、web.xml 内で宣言される <resource-ref> タグは、次のようになります。

```
<resource-ref>
  <res-ref-name>jdbc/iBank</res-ref-name>
  <res-type>javax.sql.XADataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

これに対応する sun-web.xml 内の宣言は、次のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE FIX ME:need confirmation on the DTD to be used for
this file
<sun-web-app>
  <resource-ref>
    <res-ref-name>jdbc/iBank</res-ref-name>
    <jndi-name>jdbc/iBank</jndi-name>
  </resource-ref> </sun-web-app>
```

エンタープライズ EJB モジュールの移行

Sun ONE Application Server 6.0 および 6.5 で、EJB 1.1 API がサポートされているのに対し、Sun ONE Application Server 7 では、EJB 2.0 API がサポートされます。その結果、どちらも以下のものをサポートできます。

- ステートフル、またはステートレスなセッション Beans
- Bean 管理による持続性 (BMP)、またはコンテナ管理による持続性 (CMP) を持つエンティティ Beans

ただし、EJB 2.0 API では、セッション Beans およびエンティティ Beans に加えて、メッセージ駆動型 Beans と呼ばれる新しいタイプの Enterprise Java Beans が導入されています。

J2EE 1.3 仕様によると、EJB のさまざまなコンポーネントは、次のような構造を持った JAR ファイルにまとめる必要があります。

- *ejb-jar.xml* という名前の XML 配備記述子を持つ META-INF/ ディレクトリ
- パッケージ内のホームインタフェースおよびリモートインタフェース、Bean の実装クラスおよび補助クラスに対応する .class ファイル

Sun ONE Application Server では、このアーカイブ構造を維持します。ただし、EJB 1.1 仕様では、各 EJB コンテナベンダーに、適切であると思われる次のような機能を実装する余地を残しています。

- CMP EJB のデータベースの持続性 (特に、データベース表内の Bean の CMP フィールドとカラム間でのマッピング設定)
- CMP Beans 用カスタムファインダメソッドのロジックの実装

予想されていたように、Sun ONE Application Server 6.0 または 6.5 と Sun ONE Application Server 7 では、異なる点がいくつか存在するため、アプリケーションを移行する場合に、特別な留意が必要な部分があります。以下のように、いくつかの XML ファイルを修正する必要があります。

- *ejb-jar.xml* のような J2EE 標準 DD に備えて、最新の DTD URL を示すように、`<!DOCTYPE` 定義を変更します。
- *ias-ejb-jar.xml* ファイルを、DTD に従って手動で作成した *sun-ejb-jar.xml* という名前の修正バージョンのファイルに置き換えます。詳細については、
http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-0.dtd を参照してください。
- すべての `<ejb-name>`-*ias-cmp.xml* ファイルを、手動で作成した 1 つの *sun-cmp-mappings.xml* ファイルに置き換えます。詳細については、
http://www.sun.com/software/sunone/appserver/dtds/sun-cmp_mapping_1_0.dtd を参照してください。

- CMP エンティティ Beans のみ : Sun ONE Application Server 7 のインストールディレクトリ内の bin ディレクトリに格納されている *capture-schema* ユーティリティを使用して *dbschema* を生成し、エンティティ Beans 用の META-INF フォルダより上位に置きます。

エンタープライズアプリケーションの移行

J2EE 仕様によると、エンタープライズアプリケーションは、次のような構造を持った EAR ファイルです。

- *application.xml* という名前の J2EE アプリケーションの XML 配備記述子を持つ META-INF/ ディレクトリ
- エンタープライズアプリケーションの EJB モジュール用の .JAR アーカイブファイルと Web モジュール用の .WAR アーカイブファイル

アプリケーション配備記述子では、エンタープライズアプリケーションと Web アプリケーションのコンテキストルートを構成するモジュールを定義します。

Sun ONE Application Server 6.0/6.5 と 7 では、主に J2EE モデルがサポートされます。このモデルでは、アプリケーションがエンタープライズアーカイブ (EAR) ファイル (拡張子は .ear) の形式でパッケージ化されます。アプリケーションはさらに J2EE モジュールの集まりに分割され、EJB 用の Java アーカイブ (拡張子が .jar の JAR ファイル) とサーブレットおよび JSP 用の Web アーカイブ (拡張子が .war の WAR ファイル) にパッケージ化されます。

このため、エンタープライズアプリケーションを配備する前に、以下の手順を実行する必要があります。

- EJB を 1 つ以上の EJB モジュールにパッケージ化します。
- Web アプリケーションのコンポーネントを Web モジュールにパッケージ化します。
- EJB モジュールと Web モジュールをエンタープライズアプリケーションモジュールにアSEMBルします。
- エンタープライズアプリケーションのルートコンテキスト名を定義します。これによって、アプリケーションにアクセスするための URL を特定します。

注 : Sun ONE Application Server 7 では、Sun ONE Application Server 6.0/6.5 にはなかった新しいクラスローダ階層を使用しています。この新しい方式では、特定のアプリケーションについて、1 つのクラスローダがすべての EJB モジュールを読み込み、別のクラスローダが Web モジュールを読み込みます。これら 2 つには、親子階層の関係があり、JAR モジュールクラスローダが WAR モジュールクラスローダの親モジュールになります。このため、JAR クラスローダによって読み込まれたクラスはす

べて、WAR モジュールでも使用可能またはアクセス可能ですが、反対に WAR クラスローダによって読み込まれたクラスを JAR モジュールからアクセスすることはできません。したがって、JAR ばかりでなく WAR が必要とするクラスが存在する場合、そのクラスは JAR モジュール内だけでパッケージ化します。この指針に従わない場合、クラス競合が発生する可能性があります。

アプリケーションルートコンテキストとアクセス URL

アプリケーションのアクセス URL (アプリケーションの Web モジュールのルートコンテキスト) に関して、Sun ONE Application Server 6.0/6.5 と Sun ONE Application Server 7 では、特別な違いが 1 つあります。

hostname という名前のサーバー上に配備されたアプリケーションのルートコンテキスト名が AppName である場合、このアプリケーションのアクセス URL は、使用されるアプリケーションサーバーによって異なります。

- Sun ONE Application Server 6.0 または 6.5 の場合、常に Web フロントエンドと連携して使用されるので、アプリケーションのアクセス URL は、次のような形式になる (Web サーバーは、標準の HTTP ポート番号 80 で設定されているものとする)

```
http://hostname/NASApp/AppName/
```

- Sun ONE Application Server 7 の場合、URL は次のような形式になる

```
http://hostname:port/AppName/
```

Sun ONE Application Server 7 がデフォルトで使用する TCP ポートは、ポート番号 80 です。

Sun ONE Application Server 6.0/6.5 と Sun ONE Application Server 7 でのアクセス URL についての違いはわずかなものに見えますが、絶対 URL 参照を使用するアプリケーションを移行する場合に、問題が生じる可能性があります。このような問題が発生する場合、Sun ONE Application Server 6.0/6.5 用の Web サーバープラグインで使用される固有のマーカーが付加されないように、コードを編集して、絶対 URL 参照を更新する必要があります。

固有の拡張子の移行

Sun ONE Application Server 6.0/6.5 環境独自の多数のクラスが、アプリケーションで使用されている可能性があります。Sun ONE Application Server 6.x で使用される独自の Sun ONE パッケージの一部を以下に示します。

- com.ipplanet.server.servlet.extension
- com.kivasoft.dlm
- com.ipplanetiplanet.server.jdbc
- com.kivasoft.util
- com.netscape.server.servlet.extension
- com.kivasoft
- com.netscape.server

これらの API は、Sun ONE Application Server 7 ではサポートされていません。上記のパッケージに属するクラスを使用するアプリケーションの場合、アプリケーションが標準 J2EE API を使用するように作成し直す必要があります。また、カスタム JSP タグと UIF フレームワークを使用するアプリケーションについても、標準 J2EE API を使用するように作成し直す必要があります。

iBank アプリケーションを使用した移行手順の例は、[第 5 章「iBank アプリケーションの移行手順」](#)を参照してください。

UIF の移行

Sun ONE Application Server 7.0 EE は、アプリケーションの UIF (Unified Integration Framework) API の使用をサポートしません。その代わりに、アプリケーションを統合する JCA (J2EE Connector Adapter) の使用をサポートします。ただし、Sun ONE Application Server 6.5 で開発されたアプリケーションは UIF を使用します。こうしたアプリケーションを Sun ONE Application Server 7.0 EE に配備するには、UIF を JCA に移行する必要があります。この節では、UIF を使ってアプリケーションを Sun ONE Application Server 7.0 EE へ移行する際の必要条件と手順を示します。

アプリケーションを移行する前に、Sun ONE Application Server 6.5 に UIF がインストールされていることを確認する必要があります。インストールされているかどうかをチェックする方法として、次のいずれかを選択できます。

方法 1: レジストリファイルのチェック

UIF はアプリケーションサーバーの拡張セットとしてインストールされます。これらは、インストール時にアプリケーションサーバーレジストリに登録されます。レジストリ内で次の文字列を検索すると、UIF がインストールされているかどうかわかります。

拡張名セット

- Extension DataObjectExt-cDataObject
- Extension RepositoryExt-cLDAPRepository
- Extension MetadataService-cMetadataService
- Extension RepoValidator-cRepoValidator
- Extension BSPRuntime-cBSPRuntime
- Extension BSPErrorLogExt-cErrorLogMgr
- Extension BSPUserMap-cBSPUserMap

Solaris オペレーティング環境のレジストリファイルは、次の場所にあります。

`AS_HOME/AS/registry/reg.dat`

方法 2: インストールディレクトリの UIF バイナリをチェック

UIF インストーラは、アプリケーションサーバーインストールに特定のバイナリファイルをコピーします。以下のファイルが見つければ、UIF はインストールされています。

Solaris、Windows でのファイルの場所は次のとおりです。 `AS_HOME/AS/APPS/bin`

Solaris 環境の検索対象ファイル

- `libcBSPRlop.so`
- `libcBSPRuntime.so`
- `libcBSPUserMap.so`
- `libcDataObject.so`
- `libcErrorLogMgr.so`
- `libcLDAPRepository.so`
- `libcMetadataService.so`
- `libcRepoValidator.so`
- `libx2cBSPRuntime.so`
- `libx2cDataObject.so`

- libjx2cLDAPRepository.so
- libjx2cMetadataService.so

Windows 環境の検索対象ファイル

- cBSPRlop.dll
- cBSPRuntime.dll
- cBSPUserMap.dll
- cDataObject.dll
- ErrorLogMgr.dll
- cLDAPRepository.dll
- cMetadataService.dll
- cRepoValidator.dll
- jx2cBSPRuntime.dll
- jx2cDataObject.dll
- jx2cLDAPRepository.dll
- jx2cMetadataService.dll

UIF を Sun ONE Application Server 7 EE に移行する前に、アプリケーションが UIF API を使用していることを確認してください。

確認方法

- java ソースに `netscape.bsp` というパッケージ名があるか確認する
- java ソースに `access_cBSPRuntime.getCBSPRuntime` というメソッド名があるか確認する。UIF ランタイムを取得する際、このメソッドを呼び出す必要がある

移行プロセス

UIF の移行には、Sun ONE Connector Builder ツールを使用できます。Sun ONE Studio には、このツールも統合済みです。このため、UIF ベースのアプリケーションを JCA ベースのアプリケーションに移行する際は、Sun ONE Studio を使用することができます。Sun ONE Connector Builder ツールの主要機能は次のとおりです。

- コネクタコードの生成
- テストとデバッグ
- 配備

- コネクタ配備記述子の作成
- 配備用コネクタのアセンブル
- 管理フック
- 監視フック
- 環境のカスタマイズ

注 : Sun ONE Application Server 6.5 の UIF ベースのアプリケーションを移行する際、コードに変更を加える必要はありません。

Sun ONE Connector Builder ツールの詳細は、次の URL に記載されています。

<http://docs.sun.com/db/coll/s1.conblldr>

リッチクライアントの移行

この節では、iPlanet Application Server 6.x で開発された RMI/IIOP クライアントや ACC クライアントを Sun ONE Application Server 7 EE へ移行する手順を示します。

6.x のクライアントの認証

iPlanet Application Server では、アプリケーションがユーザー名やパスワードのような認証データをユーザーから収集できるようにするクライアントサイドのコールバック機構が備わっています。iPlanet CORBA インフラストラクチャによって収集された認証データは、IIOP 経由で Application Server に送信されます。

RMI/IIOP の ORB として ORBIX 2000 を使用している場合、移植性のあるインターセプタは、要求 / 応答シーケンスの流れを定義するフックやインターセプトポイントによって、セキュリティを実装します。

7 SE/EE のクライアントの認証

認証は JAAS (Java Authorization and Authentication System API) に基づいて行われるので、クライアントは CallBackHandler を実装できます。クライアントが CallBackHandler を提供しない場合、ACC は LoginModule によって呼び出されるデフォルトの CallBackHandler を使用して認証データを取得します。

JAAS による認証方法の詳細は、『Sun ONE Application Server 7 Developer's Guide to Clients』を参照してください。

6.x と 7 EE での ACC の使用

6.x では、独立した appclient スクリプトは提供されません。クラスパスに iascleint.jar ファイルではなく iasacc.jar ファイルを配置する必要があります。6.x で ACC を使ってアプリケーションをパッケージ化する唯一の利点は、クライアントアプリケーションに指定された JNDI 名が EJB の絶対 JNDI 名に間接的にマップされている点です。

6.x アプリケーションでは、スタンドアロンクライアントは、JNDI ルックアップで EJB の絶対名を使用します。つまり、ACC 外では、JNDI ルックアップは次のように行われます。

```
initial.lookup("ejb/ejb-name");
initial.lookup("ejb/module-name/ejb-name");
```

6.5 SP3 で開発されたアプリケーションの場合、絶対参照を使ってルックアップを行う際、プレフィックス "java:comp/env/ejb/" を使用します。

```
initial.lookup("java:comp/env/ejb/ejb-name");
```

Sun ONE Application Server 7 では、JNDI ルックアップは EJB の jndi-name 名で行います。EJB の絶対名は使用しないでください。また、バージョン 7 では、プレフィックス java:comp/env/ejb はサポートされていません。クラスパス内の次の jar ファイルを appserv-ext.jar で置き換えてください。

iasclient.jar、iasacc.jar、または javax.jar

バージョン 7 では、アプリケーションのロードバランス機能は、クライアントサイドのコンテキストファクトリとして S1ASCTXFactory の形式だけがサポートされます。したがって、com.sun.appserv.iio.loadbalancingpolicy システムプロパティを次のように設定して、クラススタ内に代替のホストおよびポートを指定します。

```
com.sun.appserv.iio.loadbalancingpolicy=roundrobin,host1:port1,host2:port2,...,
```

このプロパティは、ORB をラウンドロビンさせるためのホストとポートの組み合わせのリストを提供します。また、これらのホスト名は、複数の IP アドレスにマップされます。このプロパティをシステムプロパティとして

org.omg.CORBA.ORBInitialHost および org.omg.CORBA.ORBInitialPort とともに使用する場合、ラウンドロビンアルゴリズムは提供されたすべての値をラウンドロビンします。ただし、コード内の環境オブジェクトにホスト名とポート番号を含めた場合、その値がシステムプロパティ設定の値をオーバーライドします。

6.5 でクライアントが接続するプロバイダ URL は、CORBA Executive Engine (CXSEnジン) の IIOP ホストとポートになります。Application Server 7 の場合、クライアントは、インスタンスの IIOP リスナーホストおよびポート番号を指定する必要があります。Application Server 7 では、別途 CXSEnジン は用意されていません。

Application Server 7 のデフォルトの IIOP ポートは 3700 です。実際の IIOP ポートの値は、`server.xml` 設定ファイルに記載されています。

インストール、管理、および配備

この章では、Sun™ ONE Application Server 7, Enterprise Edition と iPlanet™ Application Server 6.x Enterprise Edition のインストールおよび管理の違いについて説明します。また、配備トポロジについても概要を説明します。

この章には次の節があります。

- [インストールの相違](#)
- [管理と配備の相違](#)
- [配備トポロジ](#)

インストールの相違

iPlanet Application Server 6.x, Enterprise Edition の場合、アプリケーションサーバーを起動し実行するため、いくつかのサポート製品をインストールする必要があります。一方、Sun ONE Application Server 7 インストールでは、多くの追加コンポーネントが組み込み済み、または必要とされないため、インストールが簡単に行えます。

次の表に、2 つの製品の相違をまとめます。左の欄にさまざまな製品を示します。中央の欄と右の欄には、左の欄の製品に対応する iPlanet Application Server と Sun ONE Application Server 7, Enterprise Edition の製品をそれぞれ示します。

表 4-1 インストールの相違

| 製品 | iPlanet Application Server 6.x | Sun ONE Application Server 7 |
|----------|--------------------------------|-------------------------------------------------------|
| Web サーバー | iPlanet Web Server | 組み込み型 ロードバランスプラグイン用に追加の Web サーバーをインストールする必要がある |

表 4-1 インストールの相違 (続き)

| 製品 | iPlanet Application Server 6.x | Sun ONE Application Server 7 |
|-------------------|--------------------------------|------------------------------|
| LDAP サーバー | iPlanet Directory Server | 不要 |
| JMS | iPlanet Message Queue for Java | 組み込み型 (Java Message Queue) |
| 高可用性データベース (HADB) | 不要 | HTTP/S セッション状態の格納用に必要 |

最小要件

次の表には、iPlanet Application Server 6.x と Sun ONE Application Server に関し、その要点を示しています。左の欄にコンポーネントを示します。中央の欄と右の欄には、iPlanet Application Server 6.x と Sun ONE Application Server 7, Enterprise Edition の要件をそれぞれ示します。

表 4-2 2 つのバージョンのアプリケーションサーバーの最小要件

| コンポーネント | iPlanet Application Server 6.x | Sun ONE Application Server 7 |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| オペレーティングシステム | <ul style="list-style-type: none">• Solaris 2.6、8、9 (SPARC)• HP-UX• IBM AIX• Windows NT、Windows 2000 | Solaris 8、9 (SPARC) |
| ハードディスク容量 | 400M バイト | 250M バイト |
| RAM | 512M バイト | 256M バイト |
| J2SE | 1.3.1_06 (iPlanet Application Server 6.5、SP1、Enterprise Edition) | 1.4.0_02 |
| Web サーバー | <ul style="list-style-type: none">• iPlanet/Sun ONE Web Server 6.0、SP1、SP2、または 6.0.1、iPlanet Web Server 4.7• Apache 1.3.19、1.13.26• Microsoft IIS 4.0、5.0 | <ul style="list-style-type: none">• Sun ONE Web Server 6.0 SP2• Apache 1.3.27 |
| ディレクトリサーバー | iPlanet/Sun ONE Directory Server 5.0 | 不要 |
| 高可用性データベース | 使用不可 | 製品に付属 |

インストール手順の相違

- Sun ONE Application Server 7 のインストールでは、iPlanet Application Server 6.x のインストールとは異なり、Web Server や Directory Server の事前のインストールは、かならずしも必要ではなくなりました。ただし、ロードバランスプラグインをインストールする Web サーバーが別途必要になります。
- Sun ONE Application Server 7, Enterprise Edition ソフトウェアをインストールする前に、アプリケーションサーバーと高可用性データベース (HADB) サーバーの製品トポロジを決定する必要があります。

一般に、これらのホスト方法は次の 2 通りです。

- アプリケーションサーバーと HADB サーバーのノードを同一システム上でホストする
- アプリケーションサーバーと HADB サーバーのノードを別々のシステム上でホストする

どちらの場合でも、高可用性を実現するためには、1 つのコンポーネントに対して、少なくとも 2 つのシステムが必要になります。

| | |
|---|---------------------------------------------------------------------------------------|
| 注 | 有効な接続の数は、HADB のノード 1 つあたり、8 から 16 コネクションに限定されます。高負荷に対応するためには、HADB ノードの追加を計画する必要があります。 |
|---|---------------------------------------------------------------------------------------|

「Always On」テクノロジー用に実装可能なさまざまなトポロジの詳細については、『システム配備ガイド』の「Enterprise Edition」を参照してください。

- iPlanet Application Server 6.x は、Web サーバープラグインとアプリケーションサーバー駆動型メカニズムに加えて、さまざまなロードバランススキームをサポートしています。また、Web サーバープラグイン駆動型の方式は、多種多様なスティッキーなラウンドロビンスキームをサポートしています。

こうしたロードバランスの方法の詳細については、『管理者ガイド』を参照してください。

Sun ONE Application Server 7, Enterprise Edition は、高度なヘルスチェックおよびフェイルオーバー機能に加えて、シンプルなラウンドロビンスキームをサポートします。

管理と配備の相違

以下の表に、2つのバージョンのアプリケーションサーバーの管理ユーティリティと配備ユーティリティの相違を示します。

表 4-3 管理と配備の相違

| 製品 | iPlanet Application Server 6.x | Sun ONE Application Server 7 |
|-----------------|--------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| 管理インタフェース | 管理サーバー 場所: <i>install_dir/ias/bin</i> GUI: コンソールベースの管理ツール (iASAT) コマンド行ツール: <i>iascontrol</i> | 管理サーバー 場所: <i>install_dir/domains/domain_dir/admin-server/bin</i> GUI: ブラウザベースの管理ツール コマンド行ツール: <i>asadmin</i> |
| 配備ツール | 独立した配備ツール GUI: コンソールベースの配備ツール コマンド行ツール: <i>iasdeploy</i> | Sun ONE Studio 4 または <i>asadmin</i> . |
| Registry Editor | GUI: iPlanet Registry Editor (<i>kregedit</i>) | 不要 |
| cladmin | 使用不可 | 一般的な作業要件の HTTP/S セッション持続性を設定 |
| clsetup | 使用不可 | 一般的な作業要件の典型的なクラスタ環境で Sun ONE Application Server を設定 |
| ロードバランス | 管理ツール | 設定ファイルベース。設定パラメータは <i>loadbalancer.xml</i> ファイルに含まれる |
| セッションフェイルオーバー | 管理ツール | HADB コマンド行ユーティリティ |

root 以外のユーザーによるインストールと管理

Sun ONE Application Server 7, Enterprise Edition をインストールする場合、root 権限は、かならずしも必要ではありません。詳細については、『インストールガイド』を参照してください。

配備トポロジ

Sun ONE Application Server 7 のさまざまな配備トポロジの詳細については、『システム配備ガイド』を参照してください。

iBank アプリケーションの移行手順

この章では、Sun ONE Application 6.0 または 6.5 から Sun ONE Application Server 7 に、標準的な J2EE アプリケーションの主要コンポーネントを移行するプロセスについて説明します。各種コンポーネントの移行中に生じる問題に焦点をあてて、これらの問題に役立つ解決策を紹介します。

ここでは、iBank と呼ばれる J2EE アプリケーションを使用し、バージョン 6.0 または 6.5 の Sun ONE Application Server から Sun ONE Application Server 7 への、iBank アプリケーションの実際の移行作業を基にしたプロセスが示されています。iBank はオンラインバンキングサービスをシミュレートし、通常の J2EE アプリケーションに付随するあらゆる特徴を持っています。

iBank アプリケーションに適用される J2EE 仕様の重要なポイントを以下に示します。

- サブレット、特に JSP ページへの切り替えを行うサブレット (model-view-controller アーキテクチャ)
- JSP ページ、特に静的および動的なページの取り込みを含む JSP ページ
- JSP カスタムタグライブラリ
- HTTP セッションの作成および管理
- JDBC API 経由のデータベースアクセス
- Enterprise JavaBeans: ステートフル、またはステートレスなセッション Beans、CMP、および BMP エンティティ Beans
- J2EE アプリケーションの標準パッケージ化メソッドに対応したアセンブリおよび配備

iBank アプリケーションの詳細については、[付録 A 「iBank アプリケーションの仕様」](#)を参照してください。

iBank アプリケーションの移行の準備

移行プロセスを開始する前に、配備記述子の相違点を把握しておきましょう。詳細については、[37 ページの「配備記述子の移行」](#)を参照してください。

ターゲットの選択

アプリケーションがフェイルオーバー機能をサポートしている場合、移行のターゲットサーバーとして Sun ONE Application Server 7, Enterprise Edition を選択する必要があります。持続型 *ha* は、Sun ONE Application Server 7, Enterprise Edition のフェイルオーバー機能を必要とする本稼働環境に対応しています。移行対象アプリケーションでフェイルオーバー機能を有効にする方法の詳細については、『管理者ガイド』を参照してください。

アプリケーションが高可用性機能を使用しない場合は、アプリケーションのターゲットサーバーとして Sun ONE Application Server 7, Standard Edition あるいは Sun ONE Application Server 7, Platform Edition を選択できます。

ターゲットサーバーを選択したら、移行環境にサーバーをインストールします。ソフトウェアの詳しいインストール方法については、『インストールガイド』を参照してください。

コンポーネントの移行に Sun ONE Migration Tool を使用する場合は、このツールをインストールする必要があります。Sun ONE Migration Tool は、次の URL からダウンロードできます。

<http://www.sun.com/migration>

ツールの使用方法については、Sun ONE Migration Tool のオンラインヘルプを参照してください。

開発環境

Sun ONE Application Server 6.5 の iBank アプリケーションは、Sun ONE Application Server 6.5 および Sun ONE Studio で開発されています。Sun ONE Application Server 6.5 は、次の機能をサポートします。

- サーバーインスタンス間のアプリケーションのロードバランス
- サーバーインスタンスノードのフェイルオーバー

iBank アプリケーションのコンポーネントの識別

- ローカルディレクトリ内に zip 形式で格納されているアプリケーションを抽出します。

iBank アプリケーションのソース (iBank65.zip) は、移行サイト <http://www.sun.com/migration/sunonetools.html> から入手できます。iBank65.zip ファイルを解凍すると、次のようなディレクトリ構造が作成されます。

```
iBank  
/docroot  
/session  
/entity  
/misc
```

- /docroot のルートには、HTML ファイル、JSP ファイル、および画像ファイルが含まれます。また、サーブレットや EJB のソースファイルも、パッケージ構造 `com.sun.bank.*` に従って、サブフォルダ `WEB-INF\classes` に含まれます。WAR ファイルは、このディレクトリの内容から生成されます。
- /session には、パッケージ構造 `com.sun.bank.ejb.session` に従ってセッション Beans のソースコードが含まれます。このディレクトリはセッション Beans の EJB モジュールを形成します。
- /entity には、パッケージ構造 `com.sun.bank.ejb.entity` に従ってエンティティ Beans が含まれます。このディレクトリはエンティティ Beans の EJB モジュールを形成します。
- /misc には、データベースセットアップ用の SQL スクリプトが含まれます。

スキーマのセットアップ

/misc フォルダに入っている SQL スクリプトを実行して、iBank アプリケーションのスキーマをセットアップします。これらのスクリプトは、Oracle データベース用です。これらのスクリプトによって、ユーザーの作成、表の作成、および表へのデータの挿入が行われます。次の順序で、スクリプトを実行します。

- 01_iBank_CreateUser.sql
- 02_iBank_CreateTables.sql
- 03_iBank_InsertData.sql

iBank アプリケーションの手動移行

Sun ONE Application Server 7 で CMP 1.1 がサポートされているため、手動の移行でソースコードを大幅に修正する必要はありません。ただし、アプリケーションを手動で移行する場合、以下の点について、いくつか変更が必要になります。

Web アプリケーションの変更

Sun ONE Application Server 6.0/6.5 から Sun ONE Application Server 7 に iBank を移行する場合、iBank アプリケーションの Web アプリケーション部分を変更する必要はありません。ソースディレクトリから `ias-web.xml` ファイルを削除します。これは Sun ONE Application Server 7 の配備記述子 `sun-web.xml` ファイル内の対応するデータにアクセスできる情報がこのファイルに入っていないからです。`web.xml` を変更する必要はありません。

ただし、一般的には、サーバー固有のリソースにマップする必要がある `web.xml` 内に情報が含まれていることがあるため、`sun-web.xml` での宣言が必要となります。たとえば、`web.xml` ファイルで `javax.sql.DataSource` タイプのリソース参照が宣言されている場合、`sun-web.xml` 内で、サーバー上の実際のデータソースの JNDI 名に、その参照をマップする必要があります。

`sun-web.xml` は新たに作成する必要があります。以下に、Sun ONE Application Server 固有の配備記述子 `sun-web.xml` の作成プロセスの概要を示します。

1. 最上部に、次のような DOCTYPE 定義を記述した新しい XML ファイルを作成します。

```
'http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-0.dtd'>
```

`sun-web.xml` という名前で、このファイルを保存します。

2. DOCTYPE 定義から見て分かるように、この XML ファイルのルートタグは `sun-web` です。DTD では、この要素は次のように定義します。

```
<!ELEMENT sun-web-app (security-role-mapping*, servlet*,
session-config?, resource-env-ref*, resource-ref*, ejb-ref*,
cache?, class-loader?, jsp-config?, locale-charset-info?,
property*)>
```

上記の宣言から、すべてのタグがオプションであることは明白です。したがって、デフォルトの `sun-web.xml` は次のようになります。

```
<!DOCTYPE sun-web-app SYSTEM
"http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-0.dtd">
</sun-web-app>
```

3. リソース参照を宣言するには、次のように定義します。


```
<!ELEMENT resource-ref (res-ref-name, jndi-name,  
default-resource-principal?)> サブ要素は、次のようになります。
```

```
<!ELEMENT res-ref-name (#PCDATA)>  
<!ELEMENT default-resource-principal (name, password)>  
<!ELEMENT jndi-name (#PCDATA)>
```

iBank アプリケーションの場合、sun-web.xml に次のリソース参照宣言を含める必要があります。

```
<sun-web-app>  
  <resource-ref>  
    <res-ref-name>jdbc/IBank</res-ref-name>  
    <jndi-name>jdbc/IBank</jndi-name>  
    <default-resource-principal>  
      <name>ibank_user</name>  
      <password>ibank_user</password>  
    </default-resource-principal>  
  </resource-ref>  
</sun-web-app>
```

EJB の変更

Sun ONE Application Server 6.5 から Sun ONE Application Server 7 に iBank を移行する場合、EJB のコードを変更する必要はありません。

セッション Beans :

ejb-jar.xml : ejb-jar.xml では、最新の DTD URL を示すように、<!DOCTYPE 定義を変更します。この新しい定義は、次のようになります。

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

Sun ONE Application Server 6.5 の ias-ejb-jar.xml は、Sun ONE Application Server 7 では sun-ejb-jar.xml に置き換えられています。これらの 2 つの XML ファイルの DTD は根本的に異なっているため、ejb-jar.xml と ias-ejb-jar.xml から必要な情報を抽出して、新しい sun-ejb-jar.xml を作成する必要があります。以下に、ejb-jar.xml ファイルの作成プロセスの概要を示します。

1. 最上部に、次のような DOCTYPE 定義を記述した新しい XML ファイルを作成します。

```
'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2  
_0-0.dtd'>
```

このファイルに sun-ejb-jar.xml という名前を付けて、修正した ejb-jar.xml と一緒に保存します。

2. DOCTYPE 定義から見て分かるように、この XML ファイルのルートタグは sun-ejb-jar です。DTD では、この要素は次のように定義します。

```
<!ELEMENT sun-ejb-jar (security-role-mapping*,enterprise-beans)>
```

security-role-mapping タグは、ejb-jar.xml で宣言されているセキュリティロールをマッピングするためのものです。iBank アプリケーションの ejb-jar.xml ファイルにはセキュリティの宣言がないので、このタグの宣言は省略できます。enterprise-beans タグに注目してください。現在、sun-ejb-jar.xml ファイルの内容は、次のようになっています。

```
<sun-ejb-jar>
    <enterprise-beans>
    </enterprise-beans>
</sun-ejb-jar>
```

注：ここでは、ドキュメントのヘッダー部分、すなわち、XML 宣言と DOCTYPE 定義は省略しています。

3. enterprise-beans 要素は、次のように DTD で定義します。

```
<!ELEMENT enterprise-beans (name?, unique-id?, ejb*,
pm-descriptors?, cmp-resource?)>
```

オプションの名前の要素には、<enterprise-beans> の正準名が入ります。この要素には、任意の名前を付けることができます。

<unique-id> 要素は、Sun ONE Application Server で使用されるもので、アプリケーションの配備時に、Application Server によって自動的に挿入されます。

最も重要なタグは、EJB 要素タグです。この要素は、単一の EJB の実行時バインドを指定します。これは、次のように DTD で定義します。

```
<!ELEMENT ejb (ejb-name, jndi-name?, ejb-ref*, resource-ref*,
resource-env-ref*, pass-by-reference?, cmp?, principal?,
mdb-connection-factory?, jms-durable-subscription-name?,
jms-max-messages-load?, ior-security-config?,
is-read-only-bean?, refresh-period-in-seconds?, commit-option?,
gen-classes?, bean-pool?, bean-cache?)>
```

この場合、ejb 要素に ejb-name 要素が含まれます。ejb-name 要素には、EJB の正準名が入ります。この名前は、EJB の ejb-jar.xml の ejb-name 要素内に宣言した名前と一致している必要があります。また、この要素には、EJB の jndi-name も含まれます。Sun ONE Application Server 6.5 から 7 で改善された点の 1 つは、Bean の開発者が EJB の ejb-name と jndi-name を自由に変更できるようになったという柔軟性です。Sun ONE Application Server 6.5 では、EJB の JNDI 名はデフォルトで ejb/<ejb-name> に設定されていました。

移行を円滑に行うため、EJB の `jndi-name` をそのままの状態にしておき、その他のすべてのリソースについても Sun ONE Application Server 6.5 のときと同じものにしておきます。このため、ここではすべての EJB の `ejb-name` を `ejb/<ejb-name>` と宣言します。

上記のロジックを使用すると、`sun-ejb-jar.xml` の内容は、次のようになります。

```
<sun-ejb-jar>
<enterprise-beans>
<ejb>
  <ejb-name>BankTeller</ejb-name>
  <jndi-name>ejb/BankTeller</jndi-name>
</ejb>
<ejb>
  <ejb-name>InterestCalculator</ejb-name>
  <jndi-name>ejb/InterestCalculator</jndi-name>
</ejb>
</enterprise-beans>
</sun-ejb-jar>
```

4. `ejb-jar.xml` 内の各 `<ejb-ref>` 要素について、`sun-ejb-jar.xml` 内に対応する `<ejb-ref>` 要素が存在します。`ejb-jar.xml` 内の `<ejb-ref>` 要素は、その EJB の Bean クラス内から、参照されるすべての EJB を宣言するために使用します。Bean クラスコードは、`<ejb-ref-name>` を使用して、EJB を参照するとき、この `<ejb-ref-name>` をアプリケーションサーバー上の Bean の実際の `<jndi-name>` にマップする必要があります。このため、これは EJB 実装で参照される名前と Bean の実際の JNDI 名の間に、抽象化層を追加するためのメカニズムとして機能します。

上記で説明したロジックを使用して、BankTeller EJB について見てみます。`ejb-jar.xml` では、2 つの `<ejb-ref>` 宣言がこの EJB 内に存在します。最初の宣言は Customer EJB (エンティティ Bean モジュール内のエンティティ Bean) 用です。前述の手順 3 で説明したように、すべての EJB の JNDI 名は、`ejb/<ejb-name>` のままにしておき、この宣言を `sun-ejb-jar.xml` 内に追加します。

```
<sun-ejb-jar>
<enterprise-beans>
<ejb>
<ejb-name>BankTeller</ejb-name>
<jndi-name>ejb/BankTeller</jndi-name>
```

```

<ejb-ref>
<ejb-ref-name>Customer</ejb-ref-name>
<jndi-name>ejb/Customer</jndi-name>
</ejb-ref>
</ejb>
<ejb>
<ejb-name>InterestCalculator</ejb-name>
<jndi-name>ejb/InterestCalculator</jndi-name>
</ejb>
</enterprise-beans>
</sun-ejb-jar>

```

同様に、Account EJB 用の <ejb-ref> タグを追加します。
InterestCalculator Bean の場合、<ejb-ref> タグは、ejb-jar.xml 内に存在しないため、sun-ejb-jar.xml 内でも必要ありません。現在、sun-ejb-jar.xml ファイルの内容は、次のようになっています。

```

<sun-ejb-jar>
<enterprise-beans>
<ejb>
<ejb-name>BankTeller</ejb-name>
<jndi-name>ejb/BankTeller</jndi-name>
<ejb-ref>
<ejb-ref-name>Customer</ejb-ref-name>
<jndi-name>ejb/Customer</jndi-name>
</ejb-ref>
<ejb-ref>
<ejb-ref-name>Account</ejb-ref-name>
<jndi-name>ejb/Account</jndi-name>
</ejb-ref>
</ejb>
<ejb>
<ejb-name>InterestCalculator</ejb-name>
<jndi-name>ejb/InterestCalculator</jndi-name>
</ejb>
</enterprise-beans>

```

</sun-ejb-jar>

5. `ejb` 要素には、`pass-by-reference` 要素 `<!ELEMENT pass-by-reference (#PCData)>` が含まれます。

`pass-by-reference` 要素は、`pass-by-reference` (参照渡し) セマンティクスの使用を制御します。EJB 仕様では、デフォルトの動作モードとなる参照渡しを必要としています。これは、対応できない操作やより高速なパフォーマンスに向けて、`true` を設定することができます。これは、閉ざされた環境にある EJB モジュールすべてに適用できます。設定できる値は、`true` と `false` です。デフォルト値は `false` です。

6. `ejb` 要素には、`<bean-cache>` 要素も含まれます。

```
<!ELEMENT bean-cache (max-cache-size?,
is-cache-overflow-allowed?, cache-idle-timeout-in-seconds?,
removal-timeout-in-seconds?, victim-selection-policy?)>
```

この要素は、ステートフルセッション Bean とエンティティ Bean だけで使用されます。iBank アプリケーションの場合、BankTeller セッション Bean だけが、このエントリを含みます。

このタグでは、`max-cache-size` によって、キャッシュ内の Beans の最大数を定義します。`cache-idle-timeout-in-seconds` では、ステートフルセッション Bean やエンティティ Bean がアイドル状態でキャッシュ内に存在できる最大時間を指定しますこの時間を過ぎると、Bean はバックアップストアで非活性化されます。この時間はサーバーが参考にします。`cache-idle-timeout-in-seconds` のデフォルト値は 10 分です。

Bean が非活性化される (バックアップストアでアイドル状態になっている) 時間は、`removal-timeout-in-seconds` パラメータによって制御されます。Bean は `removal-timeout-in-seconds` の値を超えた時間アクセスされなかった場合、バックアップストアから削除されてしまうので、クライアントからアクセスできなくなります。`removal-timeout-in-seconds` のデフォルト値は 60 分です。

上記のエントリが追加されて、`sun-ejb-jar.xml` ファイルは、次のようになります。

```
<sun-ejb-jar>
<enterprise-beans>
<ejb>
<ejb-name>BankTeller</ejb-name>
<jndi-name>ejb/BankTeller</jndi-name>
<ejb-ref>
<ejb-ref-name>Customer</ejb-ref-name>
<jndi-name>ejb/Customer</jndi-name>
</ejb-ref>
```

```

<ejb-ref>
<ejb-ref-name>Account</ejb-ref-name>
<jndi-name>ejb/Account</jndi-name>
</ejb-ref>
<pass-by-reference>>false</pass-by-reference>
<bean-cache>
<cache-idle-timeout-in-seconds>
0
</cache-idle-timeout-in-seconds>
<removal-timeout-in-seconds>
0
</removal-timeout-in-seconds>
</bean-cache>
</ejb>
<ejb>
<ejb-name>InterestCalculator</ejb-name>
<jndi-name>ejb/InterestCalculator</jndi-name>
<pass-by-reference>>false</pass-by-reference>
</ejb>
</enterprise-beans>
</sun-ejb-jar>

```

7. ステートレスセッション Bean とメッセージ駆動型 Bean のプールを定義するためだけに使用される要素は、<bean-pool> です。

```

<!ELEMENT bean-pool (steady-pool-size?, resize-quantity?,
max-pool-size?, pool-idle-timeout-in-seconds?,
max-wait-time-in-millis?)>

```

Beans の初期数と最小数を指定する *steady-pool-size* は、プールで保持する必要があります。

resize-quantity では、プールが「プールマネージャ」によってサービスが実行されている場合に、作成または削除する Bean の数を指定します。

max-pool-size は、プールの最大サイズを指定します。指定できる値は、0 から MAX_INTEGER の範囲です。

max-pool-size は、プールの最大サイズを指定します。

pool-idle-timeout-in-seconds では、ステートレスセッション Bean やメッセージ駆動型 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します。

最終的に sun-ejb-jar.xml ファイルの内容は、次のようになります。

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>BankTeller</ejb-name>
      <jndi-name>ejb/BankTeller</jndi-name>
      <ejb-ref>
        <ejb-ref-name>Customer</ejb-ref-name>
        <jndi-name>ejb/Customer</jndi-name>
      </ejb-ref>
      <ejb-ref>
        <ejb-ref-name>Account</ejb-ref-name>
        <jndi-name>ejb/Account</jndi-name>
      </ejb-ref>
      <pass-by-reference>false</pass-by-reference>
      <bean-cache>
        <cache-idle-timeout-in-seconds>
          0
        </cache-idle-timeout-in-seconds>
        <removal-timeout-in-seconds>
          0
        </removal-timeout-in-seconds>
      </bean-cache>
    </ejb>
    <ejb>
      <ejb-name>InterestCalculator</ejb-name>
      <jndi-name>ejb/InterestCalculator</jndi-name>
```

```

    <pass-by-reference>false</pass-by-reference>
    <bean-pool>
        <pool-idle-timeout-in-seconds>
            0
        </pool-idle-timeout-in-seconds>
    </bean-pool>
</ejb>
</enterprise-beans>
</sun-ejb-jar>

```

エンティティ Beans :

ejb-jar.xml : ejb-jar.xml に備えて、最新の DTD URL を示すように、<!DOCTYPE 定義を変更します。この新しい定義は、次のようになります。

```

'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2
_0-0.dtd'>

```

ejb-jar.xml のすべての CMP に対して、1.1 の値を指定した <cmp-version> タグを挿入します。

エンティティ Bean のエントリは、次のようになります。

```

<エンティティ>
    <description>Account CMP entity bean</description>
    <ejb-name>Account</ejb-name>
    <home>com.sun.bank.ejb.entity.AccountHome</home>
    <remote>com.sun.bank.ejb.entity.Account</remote>
    <ejb-class>com.sun.bank.ejb.entity.AccountEJB</ejb-class>
    <persistence-type>Container</persistence-type>
    <prim-key-class>
        com.sun.bank.ejb.entity.AccountPK
    </prim-key-class>
    <reentrant>False</reentrant>
    <cmp-version>1.x</cmp-version>
<cmp-field>
    <field-name>branchCode</field-name></cmp-field>

```



```

    <cmp-field>
        <field-name>accTypeId</field-name></cmp-field>
    <cmp-field>
        <field-name>accBalance</field-name></cmp-field>
    <cmp-field>
        <field-name>custNo</field-name></cmp-field>
    <cmp-field>
        <field-name>accNo</field-name></cmp-field>
</entity>

```

同様に、すべての CMP Beans にこのエントリが含まれます。

セッション Beans の場合と同様に、Sun ONE Application Server 6.5 の `ias-ejb-jar.xml` は、Sun ONE Application Server 7 では `sun-ejb-jar.xml` に置き換えられています。これらの 2 つの XML ファイルの DTD は根本的に異なっているため、`ejb-jar.xml` と `ias-ejb-jar.xml` から必要な情報を抽出して、新しい `sun-ejb-jar.xml` を作成する必要があります。以下に、`sun-ejb-jar.xml` ファイルの作成プロセスの概要を示します。

1. 最上部に、次のような DOCTYPE 定義を記述した新しい XML ファイルを作成します。

```
'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-0.dtd'>
```

このファイルに `sun-ejb-jar.xml` という名前を付けて、修正した `ejb-jar.xml` と一緒に保存します。

2. DOCTYPE 定義から見て分かるように、この XML ファイルのルートタグは `sun-ejb-jar` です。DTD では、この要素は次のように定義します。

```
<!ELEMENT sun-ejb-jar (security-role-mapping*, enterprise-beans)
>
```

`security-role-mapping` タグは、`ejb-jar.xml` で宣言されているセキュリティロールをマッピングするためのものです。iBank アプリケーションの場合、`ejb-jar.xml` ファイルで宣言されているセキュリティがないので、オプションタグである `security-role-mapping` の宣言を省略して、`enterprise-beans` タグに重点を置きます。現在、`sun-ejb-jar.xml` ファイルの内容は、次のようになっています。

```

<sun-ejb-jar>
<enterprise-beans>
</enterprise-beans>

```

</sun-ejb-jar>

注：ここでは、ドキュメントのヘッダー部分、すなわち、XML 宣言と DOCTYPE 定義は省略しています。

3. <enterprise-beans> 要素は、次のように DTD で定義します。

```
<!ELEMENT enterprise-beans (name?, unique-id?, ejb*,
pm-descriptors?, cmp-resource?)>
```

オプションの名前の要素には、<enterprise-beans> の正準名が入ります。この要素には、任意の名前を付けることができます。

<unique-id> 要素は、Sun ONE Application Server で使用されるもので、アプリケーションの配備時に、Application Server によって自動的に挿入されます。

ejb 要素は、単一の EJB の実行時バインドを指定します。これは、次のように DTD で定義します。

```
<!ELEMENT ejb (ejb-name, jndi-name?, ejb-ref*, resource-ref*,
resource-env-ref*, pass-by-reference?, cmp?, principal?,
mdb-connection-factory?, jms-durable-subscription-name?,
jms-max-messages-load?, ior-security-config?,
is-read-only-bean?, refresh-period-in-seconds?, commit-option?,
gen-classes?, bean-pool?, bean-cache?)>
```

この場合、*ejb* 要素に *ejb-name* 要素が含まれます。*ejb-name* 要素には、EJB の正準名が入ります。この名前は、その EJB 用の *ejb-jar.xml* の *ejb-name* 要素内で宣言された名前と同じになります。また、この要素には、EJB の *jndi-name* も含まれます。Sun ONE Application Server 6.5 から 7 で改善された点の 1 つは、Bean の開発者が EJB の *ejb-name* と *jndi-name* を自由に変更できるようになったという柔軟性です。Sun ONE Application Server 6.5 では、EJB の JNDI 名はデフォルトで *ejb/<ejb-name>* に設定されていました。

移行を円滑に行うため、EJB の *jndi-name* をそのままの状態にしておき、その他のすべてのリソースについても Sun ONE Application Server 6.5 のときと同じものにしておきます。ここではすべての EJB の *ejb-name* を *ejb/<ejb-name>* と宣言します。

上記のロジックを使用すると、*sun-ejb-jar.xml* に次のエントリができます。

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name> Account</ejb-name>
      <jndi-name>ejb/Account</jndi-name>
    </ejb>
  </ejb> --- </ejb>
```

```

    <ejb> --- </ejb>
    other ejb's
    <ejb> --- </ejb>
    <ejb> --- </ejb>
  </enterprise-beans>
</sun-ejb-jar>

```

4. **ejb** 要素には、**pass-by-reference** 要素 `<!ELEMENT pass-by-reference (#PCData)>` が含まれます。

pass-by-reference 要素は、**pass-by-reference** (参照渡し) セマンティクスの使用を制御します。EJB 仕様では、デフォルトの動作モードとなる参照渡しを必要としています。これは、対応できない操作やより高速なパフォーマンスに向けて、**true** を設定することができます。これは、閉ざされた環境にある EJB モジュールすべてに適用できます。設定できる値は、**true** と **false** です。デフォルト値は **false** です。

5. **CMP** エンティティ Beans の場合、**cmp** 要素を宣言し、EJB1.1 および EJB2.0 の Beans の **CMP EntityBean** オブジェクトに関する実行時情報を記述します。

```

<!ELEMENT cmp (mapping-properties?, is-one-one-cmp?,
one-one-finders?)>

```

この *mapping-properties* タグには、持続性ベンダー固有の O/R マッピングファイルの場所が入ります。*is-one-one-cmp* フィールドは古い記述子を使用した **CMP 1.1** の識別に使用されます。**CMP 1.1** を古い記述子とする場合、このフィールドに **true** を指定します。*one-one-finders* には、**CMP 1.1** のファインダが入ります。

このルート要素 `<finder>` には、メソッド名とクエリパラメータを持つ **CMP 1.1** のファインダが入ります。

```

<!ELEMENT finder (method-name, query-params?, query-filter?,
query-variables?)>

```

method-name 要素には、クエリフィールドのメソッド名が入ります。*query-params* 要素には、**CMP 1.1** ファインダのクエリパラメータが入ります。

query-filter は、**CMP 1.1** ファインダのクエリフィルタが入る任意指定の要素です。

上記の **iBank** のエントリを作成すると、**sun-ejb-jar.xml** は、次のようになります。

```

<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name> Account</ejb-name>
      <jndi-name>ejb/Account</jndi-name>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>

```

```

    <pass-by-reference>false</pass-by-reference>
    <cmp>
    <mapping-properties>
        META-INF/sun-cmp-mappings.xml
    </mapping-properties>
    <is-one-one-cmp>true</is-one-one-cmp>
    <one-one-finders>
    <finder>
    <method-name>
        findOrderedAccountsForCustomer
    </method-name>
    <query-params>int custNo</query-params>
    <query-filter>
        custNo == custNo
    </query-filter>
    </finder>
    </one-one-finders>
    </cmp>
</ejb>
<ejb> --- </ejb>
<ejb> --- </ejb>
other ejb's
<ejb> --- </ejb>
<ejb> --- </ejb>
</enterprise-beans>
</sun-ejb-jar>

```

Account は、主キー以外のファインダを持つエンティティ Bean だけです。このため、上記のファインダのエントリは、Account Bean の場合だけになります。

6. <!ELEMENT commit-option (#PCDATA)> では、コミットのオプションを指定します。
7. <ejb> 要素には、<bean-cache> 要素も含まれます。

```
<!ELEMENT bean-cache (max-cache-size?,
is-cache-overflow-allowed?, cache-idle-timeout-in-seconds?,
removal-timeout-in-seconds?, victim-selection-policy?)>
```

この要素は、ステートフルセッション Bean とエンティティ Bean だけで使用されます。このタグでは、*max-cache-size* によって、キャッシュ内の Beans の最大数を定義します。*cache-idle-timeout-in-seconds* では、ステートフルセッション Bean やエンティティ Bean がアイドル状態でキャッシュ内に存在できる最大時間を指定します。この時間を過ぎると、Bean はバックアップストアで非活性化されます。この時間はサーバーが参考にします。*cache-idle-timeout-in-seconds* のデフォルト値は 10 分です。

Bean が非活性化される (バックアップストアでアイドル状態になっている) 時間は、*removal-timeout-in-seconds* パラメータによって制御されます。Bean は *removal-timeout-in-seconds* の値を超えた時間アクセスされなかった場合、バックアップストアから削除されてしまうので、クライアントからアクセスできなくなります。*removal-timeout-in-seconds* のデフォルト値は 60 分です。

上記のエントリが追加されて、*sun-ejb-jar.xml* ファイルは、次のようになります。

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name> Account</ejb-name>
      <jndi-name>ejb/Account</jndi-name>
      <pass-by-reference>false</pass-by-reference>
      <cmp>
        <mapping-properties>
          META-INF/sun-cmp-mappings.xml
        </mapping-properties>
        <is-one-one-cmp>true</is-one-one-cmp>
        <one-one-finders>
          <finder>
            <method-name>
              findOrderedAccountsForCustomer
            </method-name>
            <query-params>int custNo</query-params>
            <query-filter>
              custNo == custNo
            </query-filter>
          </finder>
        </one-one-finders>
      </cmp>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

```

        </query-filter>
    </finder>
</one-one-finders>

</cmp>

<commit-option>C</commit-option>

<bean-cache>

<max-cache-size>60</max-cache-size>

<cache-idle-timeout-in-seconds>

    0

</cache-idle-timeout-in-seconds>

</bean-cache>

</ejb>
<ejb> --- </ejb>
<ejb> --- </ejb>
    other ejb's
    <ejb> --- </ejb>
    <ejb> --- </ejb>
</enterprise-beans>
</sun-ejb-jar>

```

8. <!ELEMENT enterprise-beans (name?, unique-id?, ejb*, pm-descriptors?, cmp-resource?)>

<pm-descriptors> 要素は次のようになります。

<!ELEMENT pm-descriptors (pm-descriptor+, pm-inuse)>

持続マネージャ記述子には1つまたは複数の pm 記述子を含めることができます。ただし一度に使える記述子は1つだけです。

<pm-descriptor> は、エンティティ Bean に関連付けられた持続マネージャのプロパティを示します。 *pm-identifier* 要素は、PM 実装を提供したベンダーを示します。さらに、 *pm-version* では、使用する PM ベンダー製品のバージョンを指定します。 *pm-config* では、使用するベンダー固有の設定ファイルを指定します。 *pm-class-generator* では、ベンダー固有の具象クラスジェネレータを指定します。このクラスの名前はベンダーによって異なります。 *pm-mapping-factory* では、ベンダー固有のマッピングファクトリを指定します。このクラスの名前はベンダーによって異なります。 *pm-insue* では、この特定の PM を使用する必要があるかどうかを指定します。

<cmp-resource> 要素には、ejb-jar の CMP Bean を格納するのに使用するデータベースが入ります。

```
<!ELEMENT cmp-resource (jndi-name, default-resource-principal?)>
```

jndi-name 要素では、JNDI 名の文字列を指定します。*default-resource-principal* 要素には、リソースへのアクセス時に何も指定しない場合に使用する要素名とパスワードがあります。

```
<!ELEMENT default-resource-principal (name, password)>
```

これで、sun-ejb-jar.xml ファイルの内容は、次のようになります。

```
<sun-ejb-jar>
```

```
  <enterprise-beans>
```

```
    <ejb>
```

```
      <ejb-name> Account</ejb-name>
```

```
      <jndi-name> ejb/Account</jndi-name>
```

```
      <pass-by-reference>false</pass-by-reference>
```

```
      <cmp>
```

```
        <mapping-properties>
```

```
          META-INF/sun-cmp-mappings.xml
```

```
        </mapping-properties>
```

```
        <is-one-one-cmp>true</is-one-one-cmp>
```

```
        <one-one-finders>
```

```
          <finder>
```

```
            <method-name>
```

```
              findOrderedAccountsForCustomer
```

```
            </method-name>
```

```
            <query-params>int custNo</query-params>
```

```
            <query-filter>
```

```
              custNo == custNo
```

```
            </query-filter>
```

```
          </finder>
```

```
        </one-one-finders>
```

```
      </cmp>
```

```

    <commit-option>C</commit-option>
    <bean-cache>
        <max-cache-size>60</max-cache-size>
        <cache-idle-timeout-in-seconds>
            0
        </cache-idle-timeout-in-seconds>
    </bean-cache>
</ejb>

<ejb> --- </ejb>
<ejb> --- </ejb>

        other ejb's

<ejb> --- </ejb>
<ejb> --- </ejb>
<pm-descriptors>
    <pm-descriptor>
        <pm-identifier>IPLANET</pm-identifier>
        <pm-version>1.0</pm-version>
        <pm-class-generator>
            com.iplanet.ias.persistence.
            internal.ejb.ejbcb.JDOCodeGenerator
        </pm-class-generator>
        <pm-mapping-factory>
            com.iplanet.ias.cmp.NullFactory
        </pm-mapping-factory>
    </pm-descriptor>
    <pm-inuse>
        <pm-identifier>IPLANET</pm-identifier>
        <pm-version>1.0</pm-version></pm-inuse>
</pm-descriptors>

```



```

        <cmp-resource>

        <jndi-name>jdo/pmf</jndi-name>

        </cmp-resource>

    </enterprise-beans>

</sun-ejb-jar>

```

Sun ONE Application Server 7 のインストールディレクトリ内の bin ディレクトリに格納されている *capture-schema* ユーティリティを使用して、*dbschema* を生成します。¥bin ディレクトリに格納されている *capture-schema.bat* ファイルを実行して、データベース URL、ユーザー名、パスワードについて有効な値を指定した後、スキーマを生成する必要がある表を指定します。デフォルトでは、アプリケーションで使用されるすべての表について、スキーマを生成する必要があります。iBank の場合、スキーマを生成する必要がある表は 6 つです。このスキーマファイルには、*myschema.dbschema* という名前を付けます。iBank で使用される表を以下に示します。

ACCOUNT

ACCOUNT_TYPE

BRANCH

CUSTOMER

TRANSACTION_HISTORY

TRANSACTION_TYPE

この *myschema.dbschema* ファイルをエンティティ Beans の META-INF フォルダ上に格納します。

<ejb-name>-ias-cmp.xml : Sun ONE Application Server 6.0/6.5 のすべての <ejb-name>-ias-cmp.xml ファイルを 1 つの sun-cmp-mappings.xml ファイルに置き換えます。このファイルは、少なくとも 1 つの Beans のセットを特定の *dbschema* の表およびカラムにマップします。これらの 2 つの XML ファイルの DTD は、根本的に異なっているため、以下に示す手順に従って、新しいファイルを実際に作成する必要があります。

1. 最上部に、次のような DOCTYPE 定義を記述した新しい XML ファイルを作成します。

```
'http://www.sun.com/software/sunone/appserver/dtds/sun-cmp_mapping_1_0.dtd'>
```

sun-cmp-mappings.xml という名前で、このファイルを保存します。

2. DOCTYPE 定義から見て分かるように、この XML ファイルのルートタグは `sun-cmp-mappings` です。DTD では、この要素は次のように定義します。

```
<!ELEMENT sun-cmp-mappings ( sun-cmp-mapping+ ) >
```

`sun-cmp-mapping` 要素は、次のようになります。

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

この場合、`schema` 要素は、スキーマファイルへのパス名になります。

CMP Bean 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。`entity-mapping` 要素は、次のような要素を保持します。

```
<!ELEMENT entity-mapping (ejb-name, table-name,
cmp-field-mapping+, cmr-field-mapping*, secondary-table*,
consistency?)>
```

`ejb-name` 要素は、標準 `ejb-jar` DTD の EJB 名です。`table-name` 要素は、データベース表の名前です。`cmp-field-mapping` は、フィールド、すなわち、`cmr-field mapping` にマップする 1 つ以上のカラムを持ちます。`cmr` 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。`secondary-table` 要素は、使用する二次表です。`iBank` では、二次表は使用しません。

上記の変更後、Account エンティティ Bean のエントリを持つ `sun-cmp-mappings.xml` ファイルは、次のようになっています。

```
<sun-cmp-mapping>

  <schema>mySchema</schema>

  <entity-mapping>

    <ejb-name>Account</ejb-name>

    <table-name>ACCOUNT</table-name>

    <cmp-field-mapping>

      <field-name>custNo</field-name>

      <column-name>CUST_NO</column-name>

    </cmp-field-mapping>

    <cmp-field-mapping>

      <field-name>branchCode</field-name>

      <column-name>BRANCH_CODE</column-name>

    </cmp-field-mapping>

    <cmp-field-mapping>

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

```

        <column-name>ACCTYPE_ID</column-name>
    </cmp-field-mapping>
    <cmp-field-mapping>
        <field-name>accNo</field-name>
        <column-name>ACC_NO</column-name>
    </cmp-field-mapping>
    <cmp-field-mapping>
        <field-name>accBalance</field-name>
        <column-name>ACC_BALANCE</column-name>
    </cmp-field-mapping>
</entity-mapping>
</sun-cmp-mapping>

```

注：ここでは、ドキュメントのヘッダー部分、すなわち、XML 宣言と DOCTYPE 定義は省略しています。

すべての CMP エンティティ Beans について、エントリを作成する必要があります。

配備用アプリケーションのアセンブル

Sun ONE Application Server 7 では、主に J2EE モデルがサポートされます。このモデルでは、アプリケーションがエンタープライズアーカイブ (EAR) ファイル (拡張子は .ear) の形式でパッケージ化されます。アプリケーションはさらに J2EE モジュールの集まりに分割され、EJB 用の Java アーカイブ (拡張子が .jar の JAR ファイル) とサーブレットおよび JSP 用の Web アーカイブ (拡張子が .war の WAR ファイル) にパッケージ化されます。

すべての JSP とサーブレットが WAR ファイルに、またすべての EJB が JAR ファイルに、それぞれパッケージ化され、最終的には WAR ファイルと JAR ファイルが配備記述子と一緒に EAR ファイルにパッケージ化されます。この EAR ファイルは、配備が可能なコンポーネントです。

asadmin ユーティリティを使用した Sun ONE Application Server 7 での iBank アプリケーションの配備

最後の作業は、Sun ONE Application Server 7 のインスタンスでのアプリケーションの配備です。アプリケーションを配備するプロセスは、以下に示すとおりです。

Sun ONE Application Server 7 の *asadmin* には、「Help (ヘルプ)」メニューからアクセスできる配備に関するヘルプ項目が含まれています。

コマンド行ユーティリティ *asadmin* を起動するには、Windows の場合は *asadmin.bat* ファイルを、Solaris オペレーティング環境の場合は *asadmin* ファイルをそれぞれ実行します。これらのファイルは、Sun ONE Application Server 7 のインストールディレクトリ内の *bin* ディレクトリ (*Install_dir/AppServer7/appserv/bin* など) に格納されています。

asadmin プロンプトから、次のような配備用のコマンドを入力します。

```
asadmin> deploy -u username -w password -H hostname -p adminport
[--type application|ejb|web|client|connector] [--contextroot
contextroot] [--force=true] [--name component-name] [--upload=true]
[--instance instancename] filepath
```

サーバーインスタンスを再起動してから、ブラウザに

`http://<machine_name>:<port_number>/IBank` という URL を入力して、アプリケーションをテストします。使用可能なユーザー名とパスワードを指定してテストします。これによって、iBank アプリケーションのメインメニューページが表示されます。

6.5 アプリケーションの Sun ONE Studio へのインポート

この章では、Sun ONE Studio による Sun ONE Application Server 6.5 で開発したアプリケーションの移行について説明します。Sun ONE Studio for Java を使って各アプリケーションコンポーネントを移行する手順を示します。Sun ONE Application Server 6.5 付属の iBank サンプルアプリケーションを使って具体的な手順を解説します。

iBank アプリケーションの詳細については、付録 A 「iBank アプリケーションの仕様」を参照してください。

アプリケーションの移行の準備

Sun ONE Studio を使用した移行作業に先立ち、次の手順を実行し、開発環境を移行するためのセットアップを行います。

1. 開発環境に Sun ONE Application Server 7 EE および Sun ONE Studio for Java 4.0 をインストールします。

Sun ONE Application Server 7 EE および Sun ONE Studio のインストール方法については、『インストールガイド』を参照してください。

2. ローカルディレクトリ内に zip 形式で格納されているアプリケーションを抽出します。

iBank アプリケーションのソース (iBank65.zip) は、移行サイト <http://www.sun.com/migration/sunonetools.html> から入手できます。iBank65.zip ファイルを解凍すると、次のようなディレクトリ構造が作成されます。

```
iBank  
/docroot  
/session  
/entity  
/script
```

- /docroot のルートには、HTML ファイル、JSP ファイル、および画像ファイルが含まれます。また、サーブレットや EJB のソースファイルも、パッケージ構造 `com.sun.bank.*` に従って、サブフォルダ `WEB-INF\classes` に含まれます。WAR ファイルは、このディレクトリの内容から生成されます。
 - /session には、パッケージ構造 `com.sun.bank.ejb.session` に従ってセッション Beans のソースコードが含まれます。このディレクトリはセッション Beans の EJB モジュールを形成します。
 - /entity には、パッケージ構造 `com.sun.bank.ejb.entity` に従ってエンティティ Beans が含まれます。このディレクトリはエンティティ Beans の EJB モジュールを形成します。
 - /scripts には、データベースセットアップ用の SQL スクリプトが含まれます。
3. /scripts フォルダに入っている SQL スクリプトを実行して、iBank アプリケーションのスキーマをセットアップします。これらのスクリプトは、Oracle データベース用です。これらのスクリプトによって、ユーザーの作成、表の作成、および表へのデータの挿入が行われます。次の順序で、スクリプトを実行します。
- 01_iBank_CreateUser.sql
 - 02_iBank_CreateTables.sql
 - 03_iBank_InsertData.sql

上記のスクリプトを実行するには、システムに Oracle データベースと SQL コマンドがインストールされている必要があります。

4. Sun ONE Application Server を起動します。
- 詳細については、『入門ガイド』を参照してください。
5. サンプルアプリケーションの「iBank」のアセンブルと配備を行うために、Sun ONE Studio を準備します。

次の場所に格納されている `runide.sh` ファイルを使って、Sun ONE Studio ツールを起動します。

Sun ONE App Server_Root/AppServ/Sun ONE Studio forJava_Root/bin

- a. IDE のエクスプローラウィンドウで「Runtime (実行時)」タブを選択します。
- b. 「Server Registry (サーバーレジストリ)」をクリックします。
- c. 「Installed Servers (インストールされたサーバー)」をクリックします。
- d. 「Sun ONE Application Server」を選択します。

- e. 「Sun ONE Application Server」ノードを右クリックし、「Add Admin Server (管理サーバーを追加)」を選択して、管理サーバーをセットアップします。
- f. 次の情報を入力します。
 - o `host` - Sun ONE Application Server のインストールホスト。アプリケーションサーバーがローカルマシンにインストールされている場合、「localhost」と入力
 - o `port number` - `port` アプリケーションサーバーにアクセスできるポート番号。デフォルトのポート番号は **4848**
 - o `username` - インストール時に指定したユーザー名
 - o `password` - アプリケーションサーバーのインストール時に指定したパスワード
- g. 管理サーバーのセットアップ後、管理サーバーをクリックして、サーバーインスタンスをインストールします。
- h. このサーバーインスタンスをデフォルトサーバーとして設定するには、サーバーインスタンスを右クリックし、「Set As Default (デフォルトとして設定)」を選択します。

アプリケーションコンポーネントの移行

この節では、iBank アプリケーションの各コンポーネントの移行手順を説明します。

配備記述子の移行方法の詳細については、[37 ページの「配備記述子の移行」](#)を参照してください。

- [96 ページの「Sun ONE Studio for Java での Web アプリケーションモジュールの作成」](#)の指示に従って、Web モジュールを作成します。
- EJB を移行します。Sun ONE Studio は EJB の移行をサポートしません。EJB は手動で移行する必要があります。ただし、Sun ONE Studio でも、EJB のソースコードを開いて変更することは可能です。EJB の移行方法の詳細については、[47 ページの「EJB の移行」](#)を参照してください。
- JDBC コードを移行します。JDBC コードの移行方法の詳細については、[39 ページの「JDBC コードの移行」](#)を参照してください。
- iBank アプリケーションには CMP 1.1 のエンティティ Beans が付属しています。エンティティ Beans の CMP 1.1 を CMP 2.0 に変換する方法については、[165 ページの「CMP エンティティ EJB の移行」](#)を参照してください。

Account エンティティ Bean のコードには Enumeration が使用されています。このコードは、[165 ページの「CMP エンティティ EJB の移行」](#)に記載される指示に従って、手動で変更する必要があります。CMP を 1.1 から 2.0 に変換する変更作業の例については、[165 ページの「CMP エンティティ EJB の移行」](#)を参照してください。

- エンティティ Beans とセッション Beans それぞれに EJB モジュールを作成します。[113 ページの「Sun ONE Studio for Java での EJB モジュールの作成」](#)を参照してください。
- エンタープライズアプリケーションを作成します。[130 ページの「Sun ONE Studio for Java でのエンタープライズアプリケーションの作成」](#)の手順に従ってください。EJB モジュールの作成手順のほか、Web モジュールの作成手順も記載されています。この手順を実行すると、最終的に配備が可能な .ear ファイルが作成されます。
- [133 ページの「Sun ONE Application Server 7 でのアプリケーションの配備」](#)に記載される指示に従って、Sun ONE Application server 7 で .ear ファイルを配備します。

Sun ONE Studio for Java での Web アプリケーションモジュールの作成

Sun ONE Studio for Java で Web モジュールを作成するには、以下の手順を実行します。

1. ソースファイルが格納されている /docroot ディレクトリをマウントします。Sun ONE Studio for Java でディレクトリをマウントするには、左側のペインの「FileSystems Explorer (FileSystems Explorer)」タブを選択します。「File (ファイル)」メニューから「Mount the FileSystem (ファイルシステムをマウント)」を選択します。「Browser (ブラウザ)」ダイアログボックスで、マウントするディレクトリを選択します。
2. ソースファイルディレクトリ構造 (entity および session) に、EJB が格納されているディレクトリをマウントします。
3. ルートディレクトリ構造に Web モジュール用の空ディレクトリを作成します。たとえば、WarContent などです。
4. 新しく作成した WarContent ディレクトリをマウントします。
5. ファイルシステム WarContent を Web モジュールに変換します。マウントしたディレクトリをエクスプローラで選択し、右クリックして、「Tools (ツール)」サブメニューから「Convert FileSystem to Web Module (ファイルシステムを Web モジュールに変換)」オプションを選択します。
6. ソースの JSP、HTML、および画像ファイルを docroot ディレクトリから Web アプリケーションルート、すなわち、「WarContent」ディレクトリにコピーします。

7. サブレットと補助クラスソースを WEB-INF/classes ディレクトリにコピーします。つまり、docroot ディレクトリのサブフォルダ com を WEB-INF/classes ディレクトリにコピーすることになります。
8. docroot ディレクトリの WEB-INF にあるタグライブラリを、WarContent ディレクトリの WEB-INF にコピーします。
9. ソースコードを Sun ONE Application Server 7 に移行する必要がある場合 (移行ツールを使用して変更しない場合) は、以下の手順に従ってソースコードを編集します。
 - 変更する必要がある JSP を確認します。
 - カスタム JSP タグがアプリケーションで使用されているかどうかを確認します。
 - 選択した JSP コードを Sun ONE Studio で開きます。Sun ONE Studio で JSP ファイルを開くには、左側のペインで JSP ファイルを選択し、右クリックして、ポップアップメニューから「Open (開く)」オプションを選択します。
 - [45 ページの「Java Server Pages および JSP カスタムタグライブラリの移行」](#)に記載される手順に従って、ソースを修正します。
 - 同様に、[46 ページの「サブレットの移行」](#)に記載される手順に従って、サブレットを移行します。
10. アプリケーションをアSEMBルし、(WEB-INF/ ディレクトリ内の) 配備記述子 web.xml に必要な情報を入力します。web.xml ファイルをクリックし、要素のプロパティを編集します。このアSEMBリ段階で、EJB や Web アプリケーションで使用するデータソース参照に加えて、各サブレット、JSP ページ、および JSP タグライブラリを設定します。

以下では、Sun ONE Studio for Java で Web アプリケーションをアSEMBルする方法について説明します。

サブレットの設定

配備記述子内のサブレットを設定するには：

1. Warcontent フォルダの Web モジュールを選択します。Web モジュールを右クリックし、ポップアップメニューからプロパティオプションを選択します。「Properties (プロパティ)」ウィンドウが表示されます。
2. サブレットを設定するには、「Deployment (配備)」タブの「Servlets (サブレット)」ボタンをクリックします。「Servlets (サブレット)」ダイアログボックスが表示されます。
3. 「Add (追加)」をクリックしてサブレットを追加します。Web アプリケーションの各サブレットについて、「Browse (ブラウズ)」ボタンをクリックして、サブレット名、サブレットの実装クラスの完全名を指定します。また、「Mapping (マッピング)」をクリックして、サブレットのマッピング要素を、さらに、初期パラメータを指定します。

図 6-1 サブレットの設定

The screenshot shows the 'Add Servlet' dialog box. The 'Servlet Name' field is set to 'LoginServlet'. The 'Display Name' field is also 'LoginServlet'. The 'Description' field is empty. The 'Servlet Class' field is 'rvlets.LoginServlet' with a 'Browse ...' button next to it. The 'Load On Startup' checkbox is unchecked. The 'Order' field is empty. The 'Mappings' field is '/CheckLogin' with a '...' button. The 'Small Icon (16x16)' and 'Large Icon (32x32)' fields are empty with '...' buttons. The 'Run As' section has 'Role Name' and 'Description' fields, both empty. Below these are sections for 'Init Parameters' and 'Security Role References', each with a table header (Init Param Name, Description, Init Param Value and Role Ref Name, Description, Role Ref Link respectively) and 'Add ...', 'Edit ...', and 'Remove' buttons. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

iBank アプリケーションのサブレットとそのマッピングについて、「[サブレットとマッピング](#)」の表に示します。左の欄にサブレット名、中央の欄に表示名、右の欄にマッピングを示します。

表 6-1 サブレットとマッピング

| サブレット名 | 表示名 | マッピング |
|-----------------------------|-----------------------------|-----------------------|
| LoginServlet | LoginServlet | /CheckLogin |
| CheckTransferServlet | CheckTransferServlet | /CheckTransfer |
| CustomerProfileServlet | CustomerProfileServlet | /CustomerProfile |
| DataSourceTestServlet | DataSourceTestServlet | /DataSourceTest |
| HelloWorldServlet | HelloWorldServlet | /HelloWorld |
| LookUpDataSourceTestServlet | LookUpDataSourceTestServlet | /LookUpDataSourceTest |
| ProjectEarningsServlet | ProjectEarningsServlet | /ProjectEarnings |
| ShowAccountSummaryServlet | ShowAccountSummaryServlet | /ShowAccountSummary |

表 6-1 サブレットとマッピング (続き)

| サブレット名 | 表示名 | マッピング |
|------------------------------|------------------------------|------------------------|
| TestContextServlet | TestContextServlet | /TestContext |
| TransferFundsServlet | TransferFundsServlet | /TransferFunds |
| UpdateCustomerDetailsServlet | UpdateCustomerDetailsServlet | /UpdateCustomerDetails |

web.xml に上記のすべてのサブレットのエントリが追加されるように、すべてのサブレットの設定を行います。

上記の手順が完了すると、「Deployment (配備)」タブに 11 個のサブレットのマッピングと 11 個のサブレットが表示されます。

JSP タグライブラリの設定

JSP タグライブラリを設定するには：

1. 「Properties (プロパティ)」ウィンドウの「Deployment (配備)」タブの「Tag Libraries (タグライブラリ)」ボタンをクリックします。「Tag Libraries (タグライブラリ)」ダイアログボックスが表示されます。
2. Web アプリケーションの配備記述子で JSP タグライブラリを定義するには、「Add (追加)」をクリックします。続いて、「Add Taglib (タグライブラリの追加)」ダイアログボックスで、ライブラリの URI (JSP ページがライブラリにアクセスするための識別子) とライブラリの配備記述子 (.tld ファイル) へのパスを指定します。

iBank には、1 つの JSP タグライブラリ TMBHisto.tld があります。配備記述子は WEB-INF フォルダに格納されます。

図 6-2 タグライブラリの設定

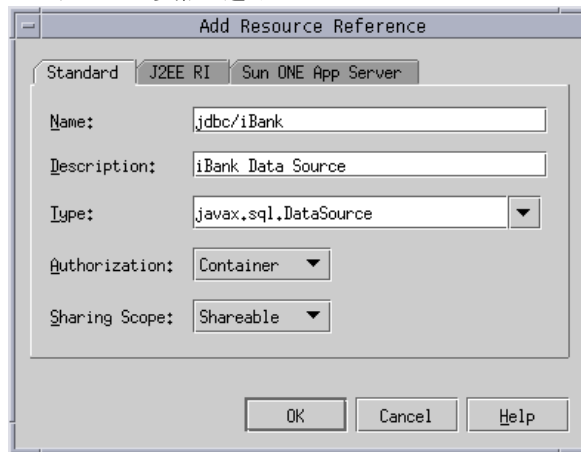


リソース参照の追加

配備記述子にリソース参照を追加するには：

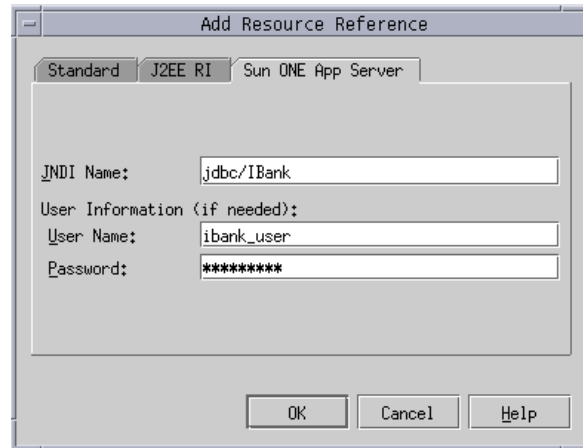
1. 「Properties (プロパティ)」ウィンドウの「References (参照)」タブを選択します。「Resource Reference (リソース参照)」ボタンをクリックします。
2. 新しいリソースを追加するには、「Add (追加)」をクリックします。「Resource Reference (リソース参照)」ダイアログボックスの「Standard (標準)」タブで、リソース参照の詳細情報を入力します。次の図は、iBank にデータソースの新しいリソース jdbc/iBank を追加する画面です。

図 6-3 リソース参照の追加



3. 「Sun ONE App Server」タブをクリックして、JNDI 名を jdbc/iBank に設定し、さらに、使用するデータベーススキーマに応じたユーザー名とパスワードも設定します。

図 6-4 Sun ONE Studio を使用して、Sun ONE Application Server 用の JDBC リソース参照を追加するためのダイアログボックスを示す図

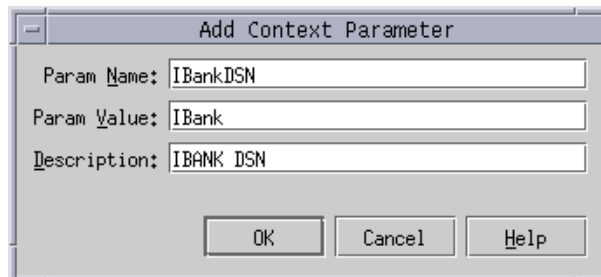


コンテキストパラメータの追加

JNDI 名のコンテキストパラメータを追加し、iBank データソースを検索するには：

1. 「Properties (プロパティ)」ウィンドウの「Deployment (配備)」タブの「Context Parameter (コンテキストパラメータ)」ボタンをクリックします。「Add Context Parameter (コンテキストパラメータの追加)」ダイアログボックスが表示されます。
2. コンテキストパラメータを追加するには、「Add (追加)」をクリックします。続いて、下の図のようにコンテキストパラメータの詳細情報を入力します。

図 6-5 コンテキストパラメータの追加



Welcome ファイルの指定

「Deployment (配備)」 タブの「Welcome Files (Welcome ファイル)」 ボタンをクリックして、Welcome ファイルを指定します。iBank の場合、Welcome ファイルは index.jsp です。

CMP エンティティ EJB の 1.1 から 2.0 への変換

iBank アプリケーションには CMP 1.1 のエンティティ Beans が付属しています。エンティティ Beans の CMP 1.1 を CMP 2.0 に変換する方法については、[165 ページの「CMP エンティティ EJB の移行」](#)を参照してください。

Account エンティティ Bean のコードには Enumeration が使用されています。このコードは手動で変更する必要があります。ここでは、Account エンティティ Bean を CMP 1.1 から CMP 2.0 へ変換する手順を示します。

Account エンティティ Bean の関連ファイルは次のとおりです。

```
Account.java
AccountEJB.java
AccountHome.java
AccountPK.java
```

上記のファイルには、次の手順に従って変更を加える必要があります。

- Account.java

ファイルを編集して、主キーのセッターをコメントアウトします。その他のセッターは変更しません。

ファイルを修正する前のコードは、以下のようになります。

```
public String getBranchCode()
    throws RemoteException;

public void setBranchCode(String branchCode)
    throws RemoteException;

public String getAccNo()
    throws RemoteException;

public void setAccNo(String accNo)
    throws RemoteException;

-----
-----
-----other getters and setters-----
```

branchCode や accNo など、主キーのセッターをコメントアウトした後のコードは、以下のようになります。

```
public String getBranchCode()
    throws RemoteException;

/* public void setBranchCode(String branchCode)
    throws RemoteException; */

public String getAccNo()
    throws RemoteException;

/* public void setAccNo(String accNo)
    throws RemoteException; */
-----
-----
-----other getters and setters-----
```

- AccountEJB.java :

Bean クラスには、次の変更を加える必要があります。

- Bean クラス宣言の先頭に、キーワード **abstract** を付けます。
修正前のコードは以下のようになります。

```
public class AccountEJB implements EntityBean
{
--
--
}
```

修正後のコードは以下のようになります。

```
public abstract class AccountEJB implements EntityBean
{
--
```

```
--  
}
```

- すべての **CMP** フィールドをコメントアウトし、**accessor** メソッドの前にキーワード **abstract** を付けます。

修正前:

```
public String branchCode;  
public String accNo;  
public int custNo;  
public String accTypeId;  
public double accBalance;  
public String accTypeDesc;  
public double accTypeInterestRate;  
private EntityContext context;  
  
public String getBranchCode() {  
    return (branchCode);  
}  
  
public void setBranchCode(String branchCode) {  
    this.branchCode = branchCode;  
}  
  
public String getAccNo() {  
    return (accNo);  
}  
  
public void setAccNo(String accNo) {  
    this.accNo = accNo;  
}  
  
public int getCustNo() {
```



```

        return(custNo);
    }

    public void setCustNo(int custNo) {
        this.custNo = custNo;
    }

    public String getAccTypeId() {
        return(accTypeId);
    }

    public void setAccTypeId(String accTypeId) {
        this.accTypeId = accTypeId;
    }

    public BigDecimal getAccBalance() {
        return new BigDecimal(accBalance);
    }

    public void setAccBalance(BigDecimal accBalance) {
        this.accBalance = accBalance.doubleValue();
    }

```

修正後：

```

private EntityContext context;
public abstract void setBranchCode(String branchCode);
public abstract String getBranchCode();
public abstract void setAccNo(String accNo);
public abstract String getAccNo();
public abstract void setCustNo(int custNo);
public abstract int getCustNo();

```

```

public abstract void setAccTypeId(String accTypeId);
public abstract String getAccTypeId();
public abstract void setAccBalance(BigDecimal accBalance);
public abstract BigDecimal getAccBalance();

```

- コード内の ejbCreate() メソッドをすべて検索します。ejbCreate が複数存在する場合もあります。パターン「<cmp-field>= 値またはローカル変数」を検索し、「abstract ミューテータメソッド名 (値またはローカル変数)」に置き換えます。

修正前のコードは以下のようになります。

```

public void setEntityContext(EntityContext ec) {
    context = ec; }

public void unsetEntityContext() {
    this.context = null;
}

public void ejbActivate() {

    this.branchCode =
((com.sun.bank.ejb.entity.AccountPK)
    context.getPrimaryKey()).branchCode;

    this.accNo = ((com.sun.bank.ejb.entity.AccountPK)
    context.getPrimaryKey()).accNo;
}

public void ejbPassivate() {
}

public void ejbLoad() {
}

public void ejbStore() {
}

```

```

    }

    public AccountPK ejbCreate(String branchCode,
                               String accNo, int custNo, String accTypeId,
                               BigDecimal accBalance) {
        this.branchCode = branchCode;
        this.accNo       = accNo;
        this.custNo       = custNo;
        this.accTypeId    = accTypeId;
        this.accBalance   = accBalance.doubleValue();
        return null;
    }

    public void ejbPostCreate(String branchCode,
                              String accNo, int custNo, String accTypeId,
                              BigDecimal accBalance) {
    }

    public void ejbRemove() {
    }

```

修正後のコードは以下のようになります。

```

    public void setEntityContext(EntityContext ec) {
        context = ec;
    }

    public void unsetEntityContext() {
        this.context = null;
    }

    public void ejbActivate() {

```

```

    }

    public void ejbPassivate() {
    }

    public void ejbLoad() {
    }

    public void ejbStore() {
    }

    public AccountPK ejbCreate(String branchCode,
                               String accNo, int custNo, String accTypeId,
                               BigDecimal accBalance) {
        setBranchCode(branchCode);
        setAccNo(accNo);
        setCustNo(custNo);
        setAccTypeId(accTypeId);
        setAccBalance(accBalance);
        return null;
    }

    public void ejbPostCreate(String branchCode,
                              String accNo, int custNo, String accTypeId,
                              BigDecimal accBalance) {
    }

    public void ejbRemove() {
    }

```

- AccountPK.java

このファイルは、変更する必要がありません。

- AccountHome.java

検索メソッドの戻り型が `java.util.Enumeration` でなければ、`Bean` のホームインタフェース内で変更を加える必要はありません。`Account Bean` の場合、ホームインタフェースに戻り型が `Enumeration` の検索メソッド `findOrderedAccountsForCustomer` があります。この場合、戻り型を `Collection` に変更する必要があります。さらに、この検索メソッドを使用するセッション `Bean` も変更する必要があります。これは、検索メソッドの戻り型 `Collection` を受け付けるためです。

ホームインタフェースで行う修正を以下に示します。

修正前のコードは以下のようになります。

```
public interface AccountHome extends EJBHome
{
    public Account findByPrimaryKey(AccountPK key)
        throws FinderException, RemoteException;

    public Enumeration findOrderedAccountsForCustomer(int
custNo)
        throws FinderException, RemoteException;
}
```

修正後のコードは以下のようになります。

```
public interface AccountHome extends EJBHome
{
    public Account findByPrimaryKey(AccountPK key)
        throws FinderException, RemoteException;

    public Collection findOrderedAccountsForCustomer(int
custNo)
        throws FinderException, RemoteException;
}
```

上記の変更の結果、この検索メソッドにアクセスするセッション `Bean` の `BankTeller` についても、`Collection` で検索メソッドの結果を取得できるように変更する必要があります。

次の抜粋コードは、`BankTellerEJB.java` への変更内容を示しています。

検索メソッド `findOrderedAccountsForCustomer` を使用するメソッド `getAccountSummary` について見てみます。

修正前のコードは以下ようになります。

```
public AccountSummary getAccountSummary()
throws EJBException
{
    int custNo          = 0;
    Enumeration accEnum = null;
    AccountSummary accSum = new AccountSummary();
    -----
    ----
    try
    {
        AccountHome home= (AccountHome) PortableRemoteObject.
            narrow(accHomeHandle.getEJBHome(),
AccountHome.class);

        AccountTypeHome accTypeHome = (AccountTypeHome)
PortableRemoteObject.narrow(accTypeHomeHandle.getEJBHome(),
AccountTypeHome.class);

        accEnum = (Enumeration) home.
            findOrderedAccountsForCustomer(this.custNo);

        AccountTypePK accTypePK = new AccountTypePK();
        Account accRef = null;
        AccountType accTypeRef = null;
        String accTypeDesc = null;
        int i = 0;
        while(accEnum.hasMoreElements())
        {
            i++;
            accRef = (Account) accEnum.nextElement();
            accTypePK.accTypeId = accRef.getAccTypeId();
            accTypeRef = (AccountType) PortableRemoteObject.
                narrow(accTypeHome.findByPrimaryKey(accTypePK),
                    AccountType.class);
```

```

        accTypeDesc = accTypeRef.getAccTypeDesc();
        accSum.addElement(
            accRef.getBranchCode(),
            accRef.getAccNo(),
            accRef.getAccBalance(),
            accTypeDesc
        );
    }
}
-----
-----
}

```

修正後のコードは以下ようになります。

```

public AccountSummary getAccountSummary()
throws EJBException
{
    int custNo          = 0;
    //Enumeration    accEnum = null;
    Collection    accEnum = null;
    AccountSummary accSum = new AccountSummary();

    try
    {
        AccountHome home = (AccountHome) PortableRemoteObject.
            narrow(accHomeHandle.getEJBHome(), AccountHome.class);
        AccountTypeHome accTypeHome = (AccountTypeHome)
            PortableRemoteObject.narrow(accTypeHomeHandle.
                GetEJBHome(), AccountTypeHome.class);
        // accEnum = (Enumeration) home.
        // findOrderedAccountsForCustomer(this.custNo);
        accEnum = (Collection) home.
            findOrderedAccountsForCustomer(this.custNo);
    }
}

```

```

AccountTypePK accTypePK = new AccountTypePK();
Account        accRef    = null;
AccountType    accTypeRef = null;
String         accTypeDesc = null;
int i = 0;
Iterator iterator = accEnum.iterator();
// while(accEnum.hasMoreElements())
while(iterator.hasNext())
{
    i++;
    // accRef = (Account) accEnum.nextElement();
accRef = (Account) PortableRemoteObject.
        narrow(iterator.next(), Account.class);
    accTypePK.accTypeId = accRef.getAccTypeId();
    accTypeRef = (AccountType) PortableRemoteObject.
        narrow(accTypeHome.findByPrimaryKey(accTypePK),
            AccountType.class);
    accTypeDesc = accTypeRef.getAccTypeDesc();
    accSum.addElement(
        accRef.getBranchCode(),
        accRef.getAccNo(),
        accRef.getAccBalance(),
        accTypeDesc
    );
}
}
----
----
}

```


Sun ONE Studio for Java での EJB モジュールの作成

ここでは、既存のソースファイルを使用して、Sun ONE Studio for Java で EJB モジュールを作成する手順を説明します。

セッション Beans 用のモジュールの作成

session フォルダには、次の Bean クラスと、次のセッション Beans のインタフェースが格納されています。

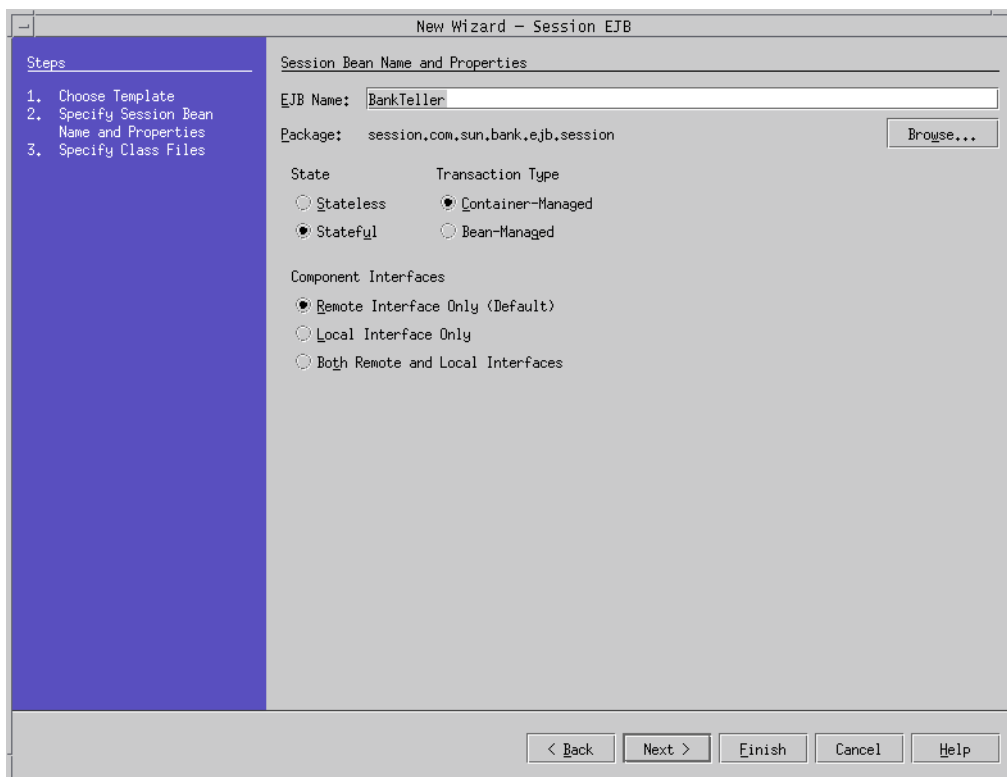
- BankTeller
- InterestCalculator

フォルダには、上記のほかに Exception クラスも格納されています。

セッション Beans のモジュールを作成するには：

1. 左のペインの「FileSystems (ファイルシステム)」タブで、マウントされたディレクトリ session を選択します。com サブフォルダに移動し、session パッケージを選択します。
2. session パッケージを右クリックします。「File (ファイル)」メニューから「New (新規)」を選択します。「New Wizard (新規作成ウィザード)」のオプションから「J2EE」->「EJB Module (EJB モジュール)」を選択します。
3. EJB の主な特性、たとえば EJB 名、Bean の状態 (ステートフルまたはステートレス)、EJB のパッケージなどを指定します。次の図は、セッション Bean BankTeller の作成画面です。このセッション Bean はステートフルセッション Bean ですが、InterestCalculator セッション Bean はステートレスです。「Browse (参照)」ボタンをクリックして、パッケージを指定します。

図 6-6 新しいセッション Bean の作成

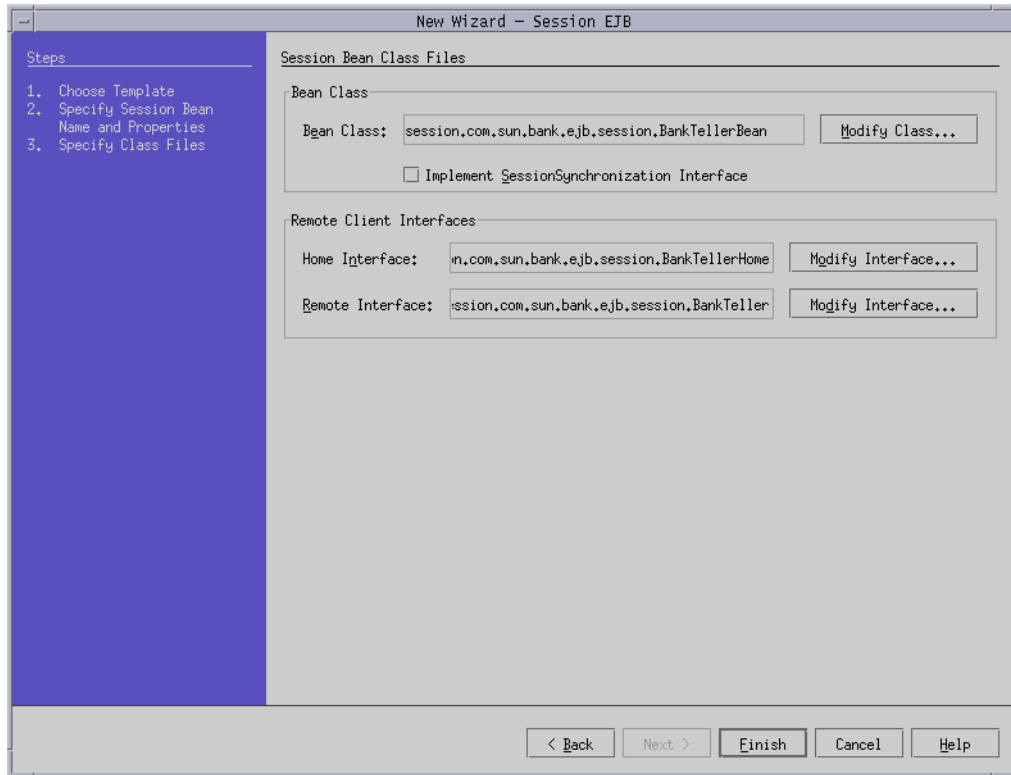


4. 既存のソースファイルに一致する実装クラス、ホームインタフェース、およびリモートインタフェースを指定するには、ダイアログボックスの「Modify (変更)」ボタンをクリックし、「Select an existing source file (既存のソースファイルを選択)」を選択します。

上記の手順を繰り返して、すべてのセッション Beans を作成します。

次の図は、セッション Bean の Bean クラス、ホームインタフェース、およびリモートインタフェースを指定する画面です。「Modify (インタフェースを変更)」ボタンをクリックし、既存のクラスを使用するオプションを選択すると、選択が可能な既存ファイルが表示されます。

図 6-7 Bean クラス、ホームインタフェース、およびリモートインタフェースの指定

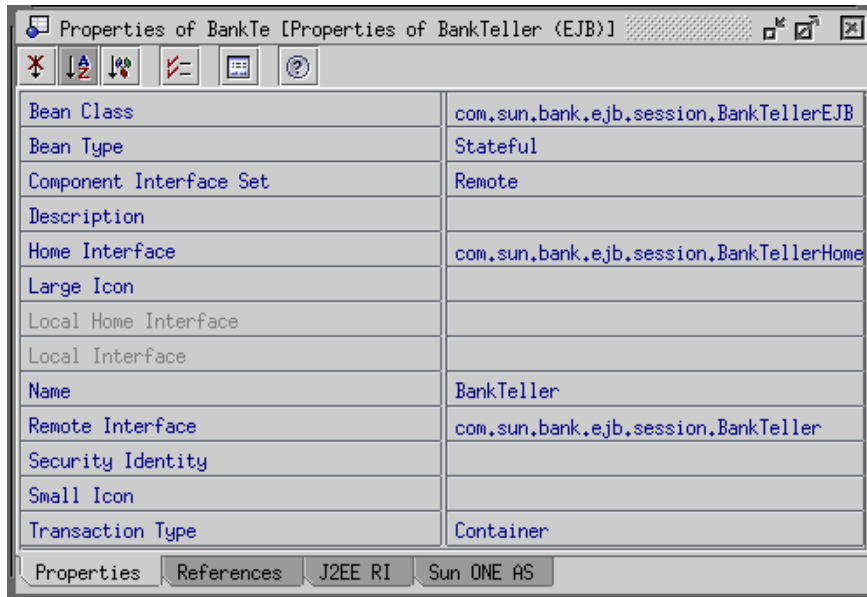


上記の手順に従って、InterestCalculator セッション Bean を作成します。

5. EJB のプロパティを編集します。

EJB のプロパティを編集すると、EJB リソース参照を宣言し、EJB の環境エントリを指定することができます。

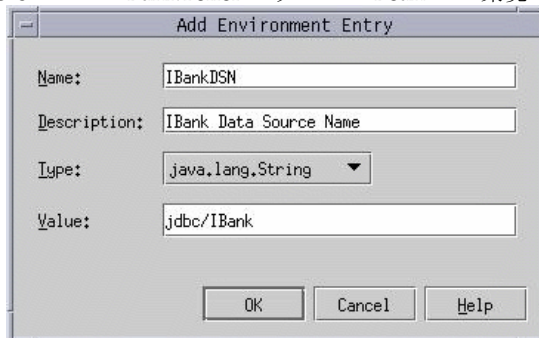
図 6-8 BankTeller セッション Bean の「Properties (プロパティ)」ウィンドウ



次の図は、BankTeller セッション Bean の環境エントリを宣言する画面です。
InterestCalculator Bean では、このエントリは必要ありません。

「References (参照)」タブの「Environment Entries (環境エントリ)」をクリックしてから、「Add (追加)」をクリックして、データソース名の新しいエントリを追加します。

図 6-9 BankTeller セッション Bean への環境エントリの追加



BankTeller セッション Bean の「Properties (プロパティ)」 ウィンドウ内の「References (参照)」 タブで、「Resource Reference (リソース参照)」 をクリックして、新しいリソースを追加します。iBank にデータソースの新しいリソース jdbc/iBank を追加します。

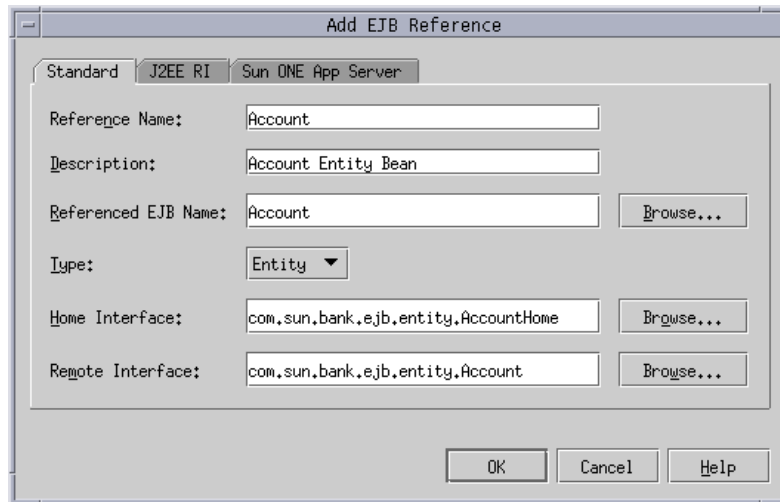
「Sun ONE App Server」 タブをクリックして、「JNDI Name (JNDI 名)」 に jdbc/iBank を設定し、さらに、使用するデータベーススキーマに応じたユーザー名とパスワードも設定します。

InterestCalculator Bean では、このエントリは必要ありません。

「Properties (プロパティ)」 ウィンドウ内の「Reference (参照)」 タブで、「EJB Reference (EJB 参照)」 をクリックして、EJB 参照を追加します。次の図は、BankTeller セッション Bean の EJB 参照を追加する画面です。BankTeller セッション Bean には、Account エンティティ Bean と Customer エンティティ Bean の参照が含まれています。これらのエンティティ Bean の参照を BankTeller セッション Bean に追加する必要があります。

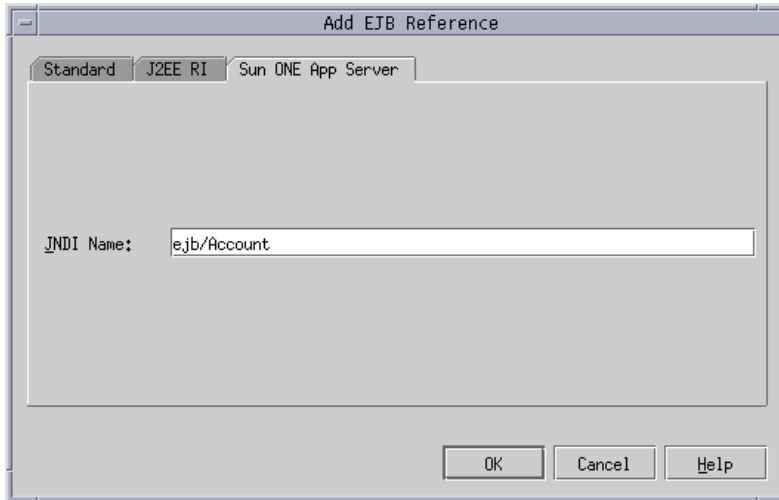
「Modify (変更)」 ボタンをクリックし、ホームインタフェースとリモートインタフェースを指定します。Beans オプションでは既存のソースを選択します。

図 6-10 EJB 参照の追加



「EJB Reference (EJB 参照)」 の「Sun ONE App Server」 タブをクリックして、JNDI 名を指定します。次の図に、Account エンティティ Bean の JNDI エントリ ejb/Account を示します。同様に、Customer Bean の EJB 参照を追加した場合は、「Sun ONE App Server」 タブで JNDI 名 jndi/Customer を指定します。

図 6-11 Sun ONE Studio を使用して、Sun ONE Application Server への EJB 参照を追加するダイアログボックスを示す図

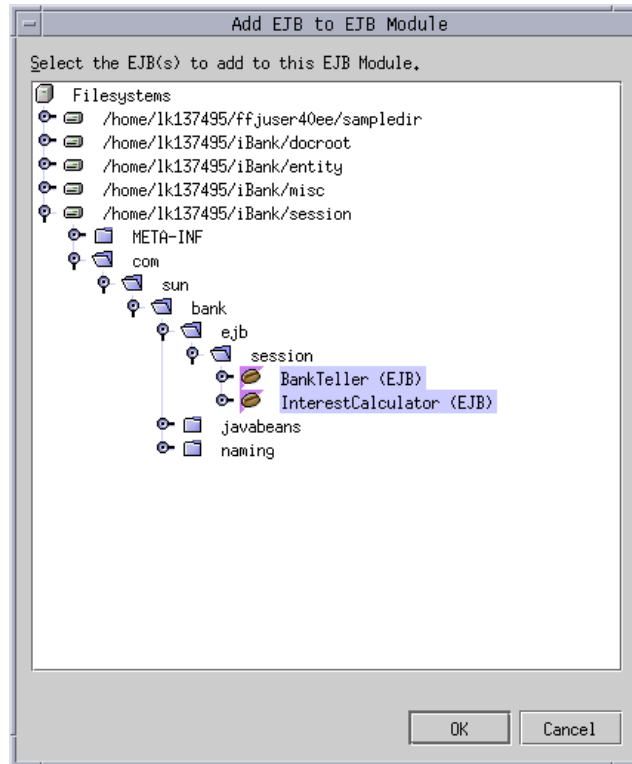


6. ソースファイルをコンパイルします。
7. EJB モジュールを作成し、モジュール内で EJB をアセンブルします。

J2EE 1.3 仕様に基づいて、Sun ONE Application Server 7 EE では、EJB モジュールをグループ化する必要があります。ルートディレクトリ `session` で、`SessionModule` として新しい EJB モジュールを作成します。新しいセッション EJB モジュールを作成するには、`session` フォルダを選択します。「File (ファイル)」メニューから「New (新規)」->「J2EE」->「EJB Module (EJB モジュール)」を選択します。EJB モジュールが作成されたら、そこにセッション EJB を追加します。

次の図は、BankTeller EJB と InterestCalculator EJB を EJB モジュール `SessionModule` に追加する画面です。

図 6-12 Sun ONE Studio を使用して、EJB モジュールに EJB を追加している図



エンティティ Beans 用のモジュールの作成

1. エンティティ Beans のフォルダには、次のエンティティ Bean の Bean クラス、リモートインタフェース、ホームインタフェースが格納されています。
 - Account
 - AccountType
 - Branch
 - Customer
 - Transaction
 - TransactionType

Customer エンティティ Bean は Bean 管理で、その他のエンティティ Bean はコンテナ管理になります。

2. JDBC ドライバを設定します。

エクスプローラの「Runtime (実行時)」ビューで、ツリーから「Database (データベース)」の下に「Drivers (ドライバ)」を選択します。右クリックして、ポップアップメニューから「Add Driver (ドライバの追加)」を選択します。「Add Driver (ドライバの追加)」ダイアログボックスで、ドライバ名、実装クラス、関連 URL のプレフィックスを指定します。対応するドライバ用の JAR または ZIP は、Sun ONE Studio for Java にアクセスできるようにするため、Sun ONE Studio for Java Root/lib/ext ディレクトリにコピーする必要があります。

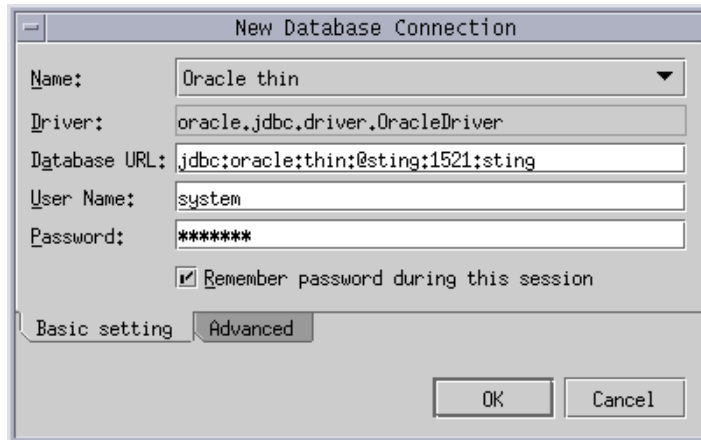
Solaris で、適切な Sun ONE Studio for Java ディレクトリにドライバクラスを格納するには、シェル (sh または ksh) から次のコマンド行を実行します。

```
cp $ORACLE_HOME/jdbc/lib/classes12.zip Sun ONE Studio for Java  
Root/lib/ext
```

3. データベース接続プロパティを定義します。

エクスプローラの「Runtime (実行時)」ビューで「Database (データベース)」を選択します。右クリックして、ポップアップメニューから「Add Connection... (接続を追加)」を選択します。「New Database Connection (新規データベース接続)」ダイアログボックスで、使用するドライバ、完全な接続 URL、ユーザー名とパスワード、および適切なデータベーススキーマを指定します。

図 6-13 Sun ONE Studio でデータベース接続を設定するダイアログボックスを示す図

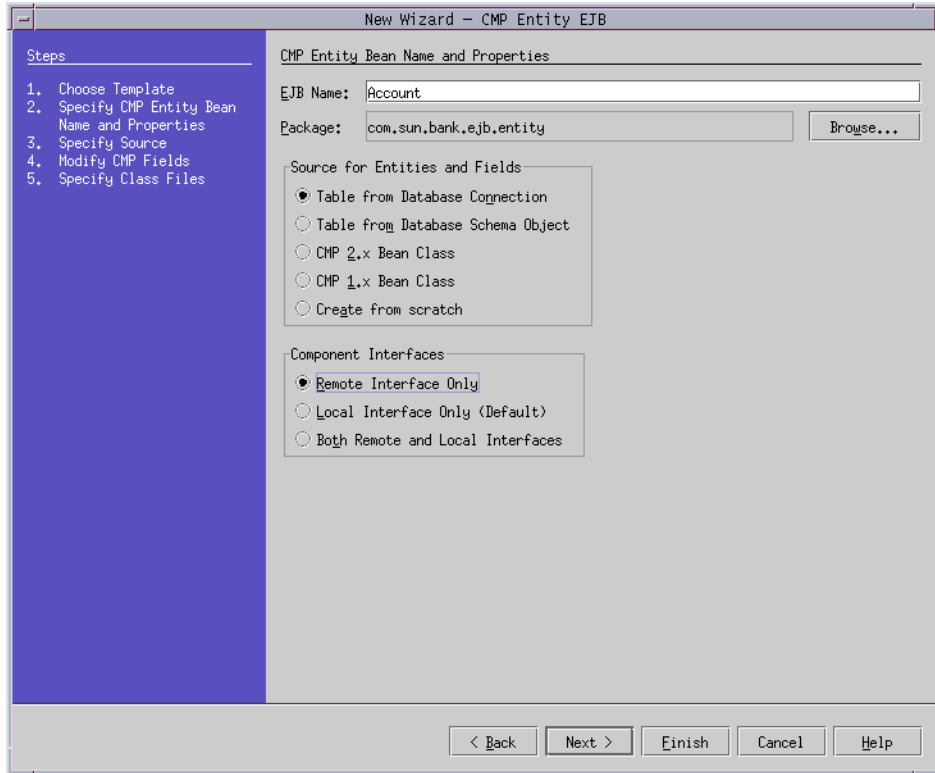


4. 既存のソースファイルから、新しい EJB を作成します。

エクスプローラの「File Systems (ファイルシステム)」タブを選択します。マウントされたディレクトリ entity を選択します。「File (ファイル)」メニューから「New (新規)」->「J2EE」->「CMP Entity EJB (CMP エンティティ EJB)」を選択します。New EJB Wizard (EJB の新規作成ウィザード) が表示されます。

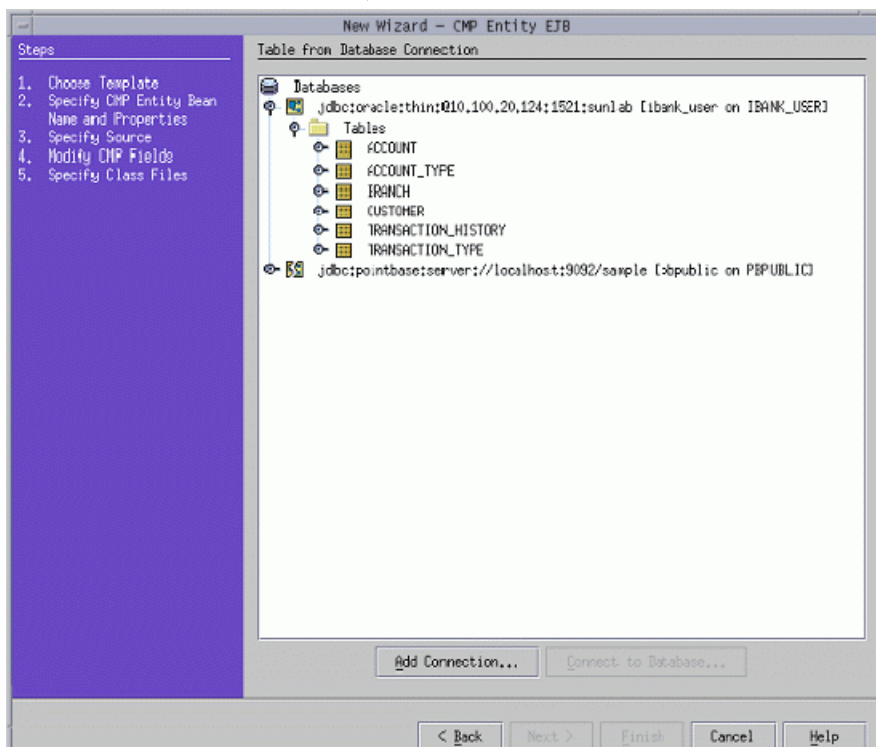
EJB 名を指定し、EJB のパッケージを定義します。「Source for Entities and Fields (エンティティのソースとフィールド)」ペインで「Table from Database Connection (データベース接続表)」を選択して、EJB フィールドの持続性のために使用するデータベース表を指定します。

図 6-14 コンテナ管理による持続性を保持するエンティティ Bean の作成



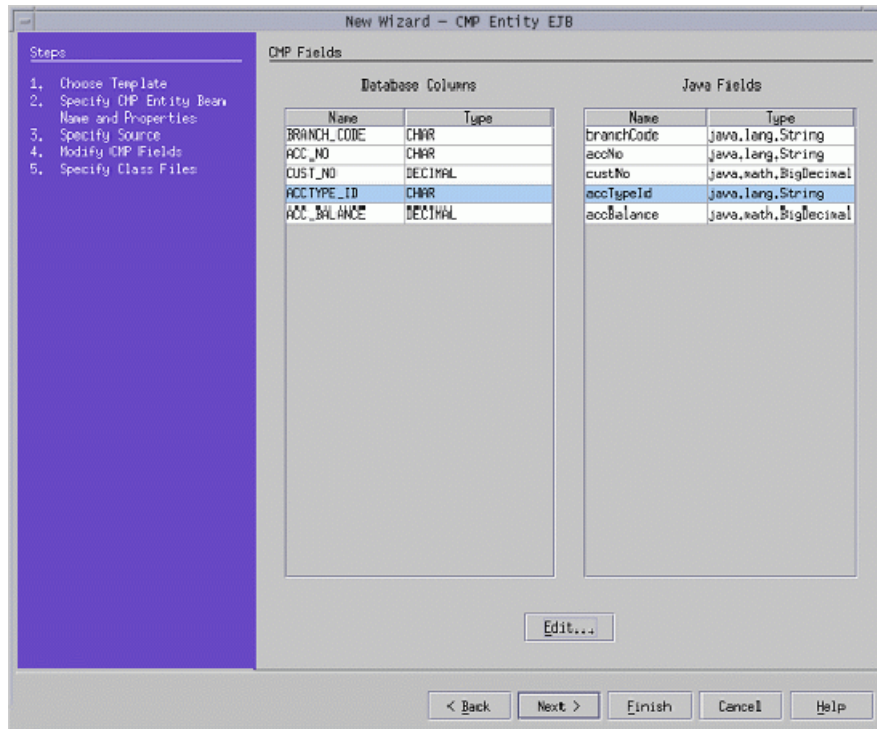
次の図のウィザード画面で、定義されているデータベース接続のリストから適切な接続を選択します。接続を選択すると、この接続からアクセスできる表のリストが表示されるので、適切な表を選択します。

図 6-15 CMP Bean フィールドのマッピング用の表の選択



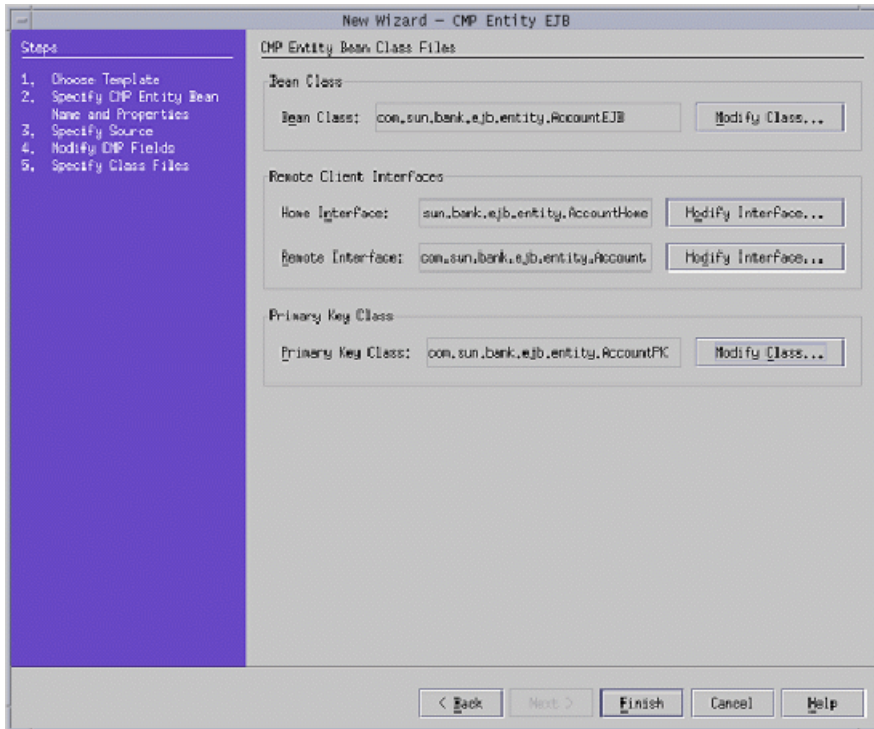
次の画面では、選択した表のカラムと **Bean** の **CMP フィールド** 間でのマッピングを設定します。ここでは、**Bean** のフィールド名と対応する **Java** タイプを正しく指定するように、注意する必要があります。

図 6-16 表のカラムと **Bean** の **CMP フィールド** 間でのマッピング



次の画面では、エンティティ Bean のソースファイルを指定します。

図 6-17 表の列と Bean の CMP フィールド間でのマッピング

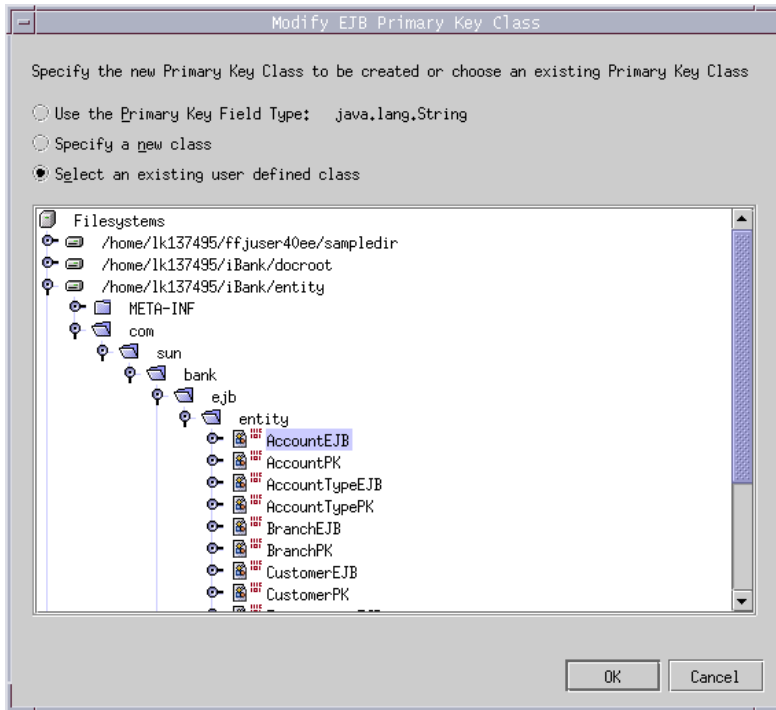


次の手順では、「Modify Class (クラスを変更)」ボタンをクリックして既存のソースファイルから EJB を作成することを Sun ONE Studio for Java に通知します。

既存のソースファイルを選択したときに、エラーが発生した場合、前の手順をきちんと行っていないか、あるいはソースが正しく移行されていない可能性があります。このようなエラーが発生した場合、修正を行って対処する必要があります。

次の画面ショットは、EJB Bean クラス用に、既存のソースファイルを選択しているところです。

図 6-18 既存の Bean クラスを選択して Bean クラスを作成している図



上記の手順を繰り返して、すべてのエンティティ Beans を作成します。

注 既存クラスの選択のオプションや新規クラスを指定するオプションを使用すると、エラーが発生する可能性があります。その場合には、同じクラスを選択するオプションを使用してください。また、Sun ONE Studio の内部に予定外の結果が発生した場合、問題を解決するためには、Sun ONE Studio をいったん終了し、再起動することが必要です。このような状況では、Sun ONE Studio で予期せぬ結果が生じる可能性があるため、Sun ONE Studio をいったん終了し、再起動してください。

5. EJB のプロパティを編集します。

エクスプローラで新しい EJB を選択し、右クリックして、ポップアップメニューから「Properties (プロパティ)」を選択します。「Properties (プロパティ)」ウィンドウが表示されます。

「Properties (プロパティ)」ウィンドウの「References (参照)」タブで、このテキストフィールドの右側の「Resource References (リソース参照)」ボタンをクリックします。

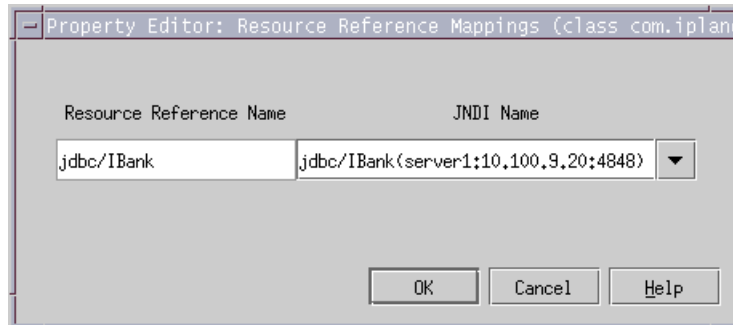
「Add Resource Reference (リソース参照の追加)」ダイアログボックスで、エンティティ Bean Customer に次のプロパティを設定します。

「Standard (標準)」タブで、データソースの完全名 (jdbc/DataSourceName) とリソースタイプ (javax.sql.DataSource) を指定し、このリソースへのアクセスを管理するオプション (「Authorization (認証)」) のドロップダウンリストから、「Container (コンテナ)」を選択します。

宣言を作成したら、「Sun ONE App Server」タブを選択し、前に定義したリソース参照に対応するエントリの JNDI 名のカラムに、データソースの JNDI 名 jdbc/iBank を指定します。ユーザー名とパスワードも指定します。

「Properties (プロパティ)」ウィンドウの「Sun ONE AS」タブを選択し、「Reference Resource Mapping (リソース参照マッピング)」をクリックしてから、使用する必要があるサーバーインスタンスで、データソース jdbc/iBank を選択します。

図 6-19 Sun ONE Application Server 用のリソース参照のマッピング

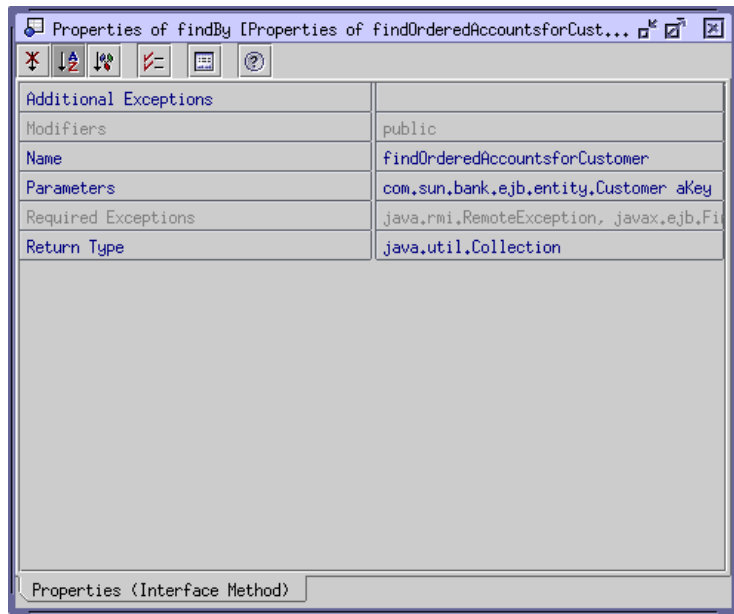


6. findByPrimaryKey メソッド以外の検索の EJB QL を設定します。

EJB QL は、検索に対して指定する必要があります。CMP 2.0 仕様に従って、検索は EJB QL を使用します。

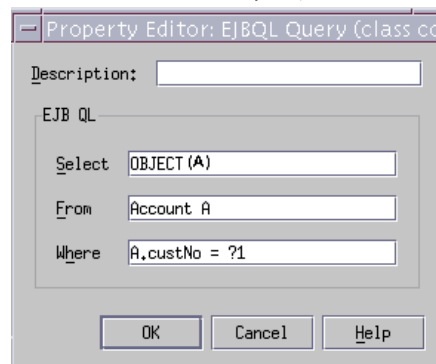
iBank アプリケーションの場合、このタイプの編集が必要なエンティティ Bean は Account Bean です。Sun ONE Studio のエクスプローラウィンドウで、AccountEJB ノードを選択し、そのノードのファインダメソッドを展開します。findByPrimaryKey 以外の検索メソッドをクリックして、その「Properties (プロパティ)」ウィンドウを開きます。

図 6-20 検索メソッドのプロパティ



「EJBQL Query (EJBQL 照会)」フィールドをクリックして、クエリを入力します。次の図は、クエリが入力されているところです。

図 6-21 検索メソッドの EJB QL の編集



7. EJB モジュールを作成し、モジュール内で EJB をアセンブルします。

EntityModule という名前の新しい EJB モジュールを作成します。作成できたら右クリックし、EJB を追加するオプションを選択して、このモジュールにすべてのエンティティ Beans を追加します。J2EE 1.3 仕様に基づいて、EJB モジュールをグループ化する必要があります。

8. 新しいデータベーススキーマを作成します。

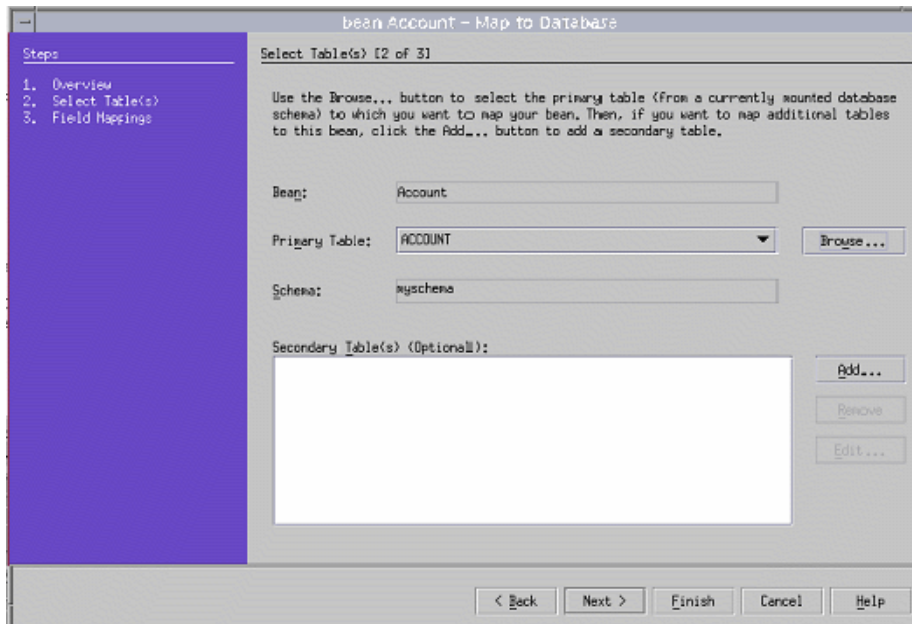
「File (ファイル)」メニューから「New (新規)」を選択し、新しいデータベーススキーマを選択します。スキーマを取り込む必要があるデータベースの接続情報を定義します。

9. Sun ONE Application Server 7 のデータベースエントリをマップします。

EJB モジュールの EJB ノードを選択し、このノードを右クリックして、「Properties (プロパティ)」ウィンドウを選択し、さらに「Sun ONE AS」タブを選択します。この特定のエンティティ Bean について、データベーススキーマと主表名を指定します。EJB モジュール内の他のエンティティ Bean についても、同様のプロセスを繰り返します。

次の図は、エンティティ Bean Account の主表を選択しているところです。

図 6-22 データベースのマッピング

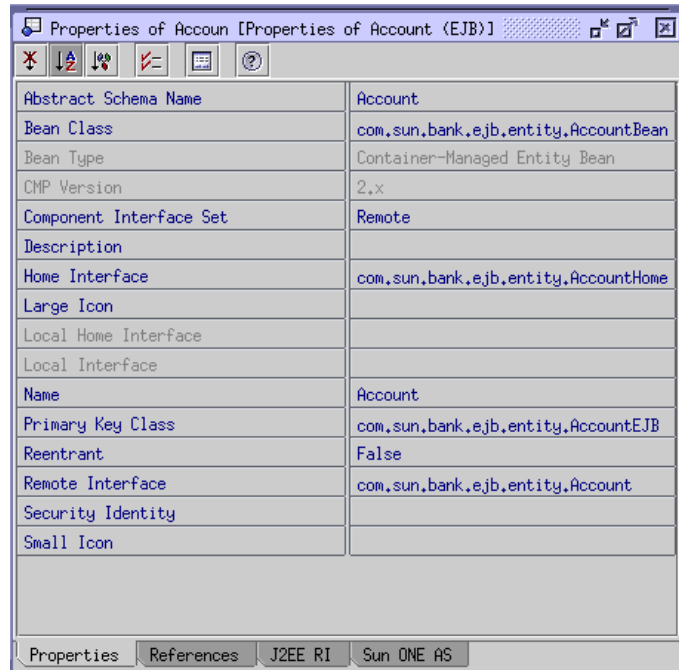


「Next (次へ)」をクリックして、Bean の cmp フィールドと表フィールドのマッピングを指定します。

次に、「Properties (プロパティ)」ウィンドウから「Sun ONE Mapping (Sun ONE マッピング)」タブを選択し、再度マッピングを入力します。

次の図は、Account EJB のマッピングを示しています。

図 6-23 エンティティ Bean 「Account」 のプロパティ



同様に、すべてのエンティティ Beans のマッピングを設定します。

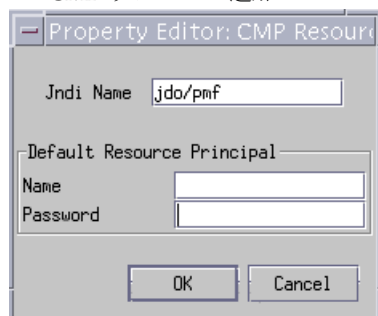
データベース表フィールドに対応する特定のエンティティ Bean のマッピングについては、[162 ページの「エンティティ Bean の関係の定義」](#)を参照してください。

10. CMP リソースを追加します。

EntityModule を選択し、プロパティを表示します。「Sun ONE AS」タブを選択し、今度は「CMP Resource (CMP リソース)」ボタンをクリックして持続性マネージャファクトリを設定します。

次の図に設定のようすを示します。

図 6-24 CMP リソースの追加



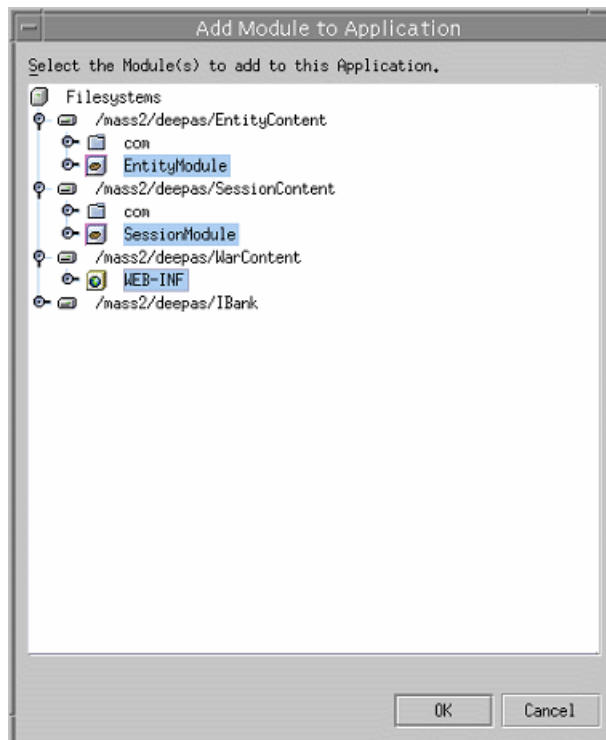
Sun ONE Studio for Java でのエンタープライズアプリケーションの作成

Web アプリケーションと EJB ファイルを作成したら、今度はすべてのモジュールをグループ化するエンタープライズアプリケーションを作成します。エンタープライズアプリケーションの作成プロセスは、以下のとおりです。

1. ソースに利用できる同じパッケージの下にある新しいディレクトリ **IBank** に、新しいエンタープライズアプリケーションモジュールを作成します。
2. Web モジュールと EJB モジュールをエンタープライズアプリケーションモジュールに追加します。

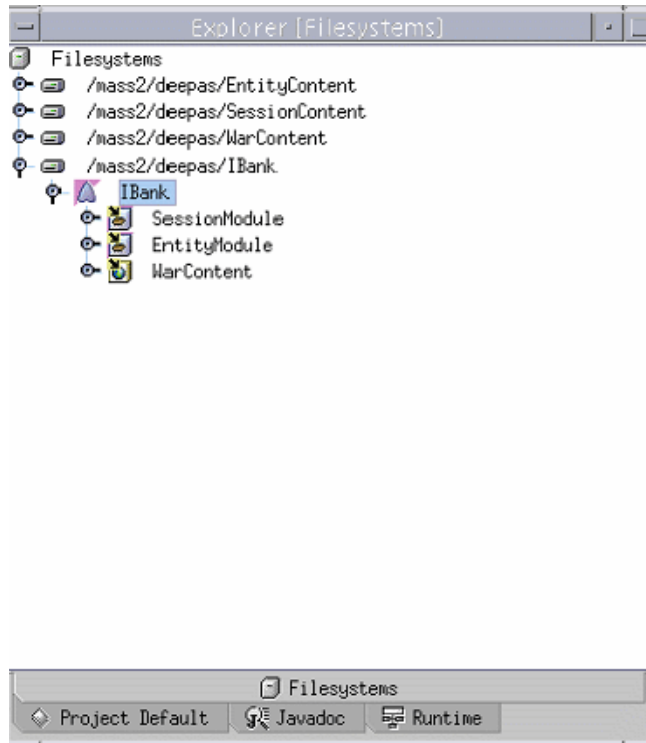
次の図は、WarContent という Web モジュール、および SessionModule と EntityModule という EJB モジュールを含んだ iBank エンタープライズアプリケーションを示しています。

図 6-25 アプリケーションへのモジュールの追加



次の図は、3つのモジュールを含んだ iBank アプリケーションを示しています。

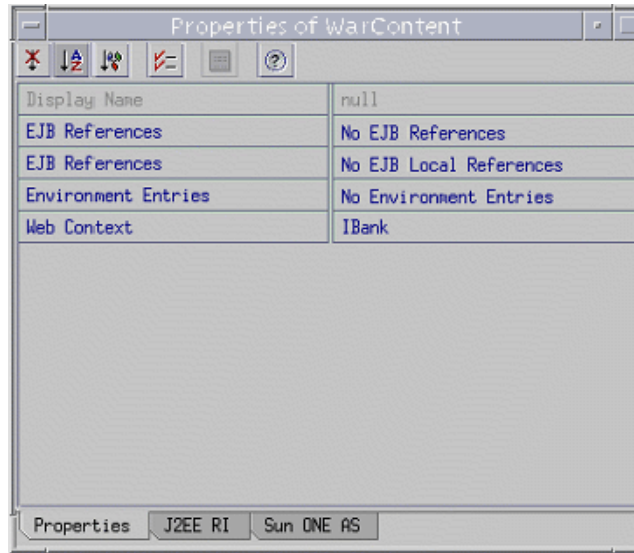
図 6-26 異なるモジュールを含んだ iBank アプリケーションのファイルシステムを示す図



3. エンタープライズアプリケーションのプロパティを編集します。

プロパティエディタを使用すると、エンタープライズアプリケーションモジュールに、異なるプロパティを設定することができます。特に、このエディタでは、エンタープライズアプリケーションの Web モジュールについて、ルートコンテキスト名を定義します。

図 6-27 Web コンテキストの指定



4. EAR ファイルをエクスポートします。

エンタープライズアプリケーションを右クリックして EAR ファイルをエクスポートします。次に、EAR ファイルをエクスポートするオプションを選択します。この EAR ファイルには、JAR ファイル、WAR ファイル、および XML ファイルが含まれています。この EAR ファイルには、Sun ONE Application Server 7 での配備に必要な Sun ONE 固有の XML ファイルが含まれます。この EAR ファイルは、これで配備することができます。

Sun ONE Application Server 7 でのアプリケーションの配備

最後の作業は、Sun ONE Application Server 7 のインスタンスでのアプリケーションの配備です。アプリケーションを配備するプロセスは、以下に示すとおりです。

1. Sun ONE Studio for Java から Sun ONE Server 7 のインスタンスへのアプリケーションの配備

EAR ファイルを右クリックし、「Deploy (配備)」オプションを選択します。これによって、デフォルトのサーバーインスタンスに、アプリケーションが配備されます。サーバーインスタンスを再起動し、アプリケーションをテストします。

2. Sun ONE Application Server 7 の asadmin ユーティリティによるインスタンスへのアプリケーションの配備

Sun ONE Studio for Java を使用しないで、Sun ONE サーバーインスタンスに、エンタープライズアプリケーションを配備する場合、Sun ONE Studio for Java からアプリケーションの EAR アーカイブを作成およびエクスポートした後で、Sun ONE Application Server 7 の asadmin ユーティリティを使用します。

iBank アプリケーションの仕様

この章では、移行例として *iBank* アプリケーションを使用します。このアプリケーションは、次の機能を持つ基本的なオンラインバンキングサービスをシミュレートします。

- オンラインバンキングサービスへのログイン
- 個人データおよび支店データの表示および編集
- 清算勘定を示す預金口座の概要表示
- 個々のトランザクション履歴を表示するために、口座ごとにドリルダウンする機能
- 口座から口座へ資金をオンラインで移動できる振替サービス
- 一定の元本および年利回り率で複数年に及ぶ複利配当額の見積り

アプリケーションは MVC (Model-View-Controller) モデルに基づいて設計されています。

- EJB はアプリケーションのビジネスおよびデータモデルコンポーネント定義に使用
- Java Server Pages はプレゼンテーションロジックを処理し、ビューを表す
- サーブレットはコントローラとして動作し、アプリケーションロジックの処理、ビジネスロジックコンポーネントの呼び出し、EJB 経由のビジネスデータへのアクセス (モデル)、Java Server Pages 表示のための処理データのディスパッチ (ビュー) を行う

アプリケーションコンポーネントのアセンブルと配備には、標準 J2EE メソッドを使用します。これには、配備記述子の定義とアプリケーションコンポーネントのアーカイブファイルへのアセンブルが含まれます。

- HTML ページ、画像、サーブレット、JSP、カスタムタグライブラリ、および必要に応じてサーバーサイド Java クラスを含み、Web アプリケーションで使用する WAR アーカイブファイル

- 配備記述子、Bean クラスおよびインタフェース、スタブおよびスケルトンクラス、および必要に応じてその他のヘルパークラスを含み、1 つ以上の EJB のアセンブルで使用する EJB-JAR アーカイブファイル
- Web アプリケーションモジュールおよびアプリケーションで使用される EJB モジュールを含み、エンタープライズアプリケーションモジュールのパッケージングで使用する EAR アーカイブファイル

標準の J2EE アセンブルメソッドを使用すると、Sun ONE Application Server 6.0/6.5 と Sun ONE Application Server 7 間の相違やそれに伴う問題点を指摘するのに役立ちます。

アプリケーション開発用ツール

Sun ONE Studio Enterprise Edition for Java、リリース 4.0

Sun ONE Application Server 7 は EJB 1.0 および EJB 1.1 標準をサポートしています。したがって iBank アプリケーションの他の EJB (2 セッション EJB および BMP エンティティ Bean) は Sun ONE Studio for Java で開発され、asadmin deploy を使用して Sun ONE Application Server 7 にパッケージされ、配備されます。これにより、Sun ONE Application Server 7 で 1.1 EJB 開発用の他社製の IDE の使用をテストする際にも役立ちます。さらに、Sun ONE Application Server 6.5 で開発された 1.1 EJB を Sun ONE Application Server 7 に移行する際にも役立ちます。

Sun ONE Studio for Java 開発環境は、iBank アプリケーションの EJB コンポーネントを Sun ONE Application Server に移行する場合にも使用されます。EJB 1.0 標準から EJB 1.1 へのコードの移行、CMP エンティティ Beans 用の O/R マッピング、アプリケーションの複数のモジュールの配備プロパティおよびパッケージングの設定などを行います。

Oracle 8i 8.1.6

データベースは Oracle 8i バージョン 8.1.6 で開発されており、アクセスに使用される JDBC ドライバは thin Oracle ドライバ (type 4) です。

データベーススキーマ

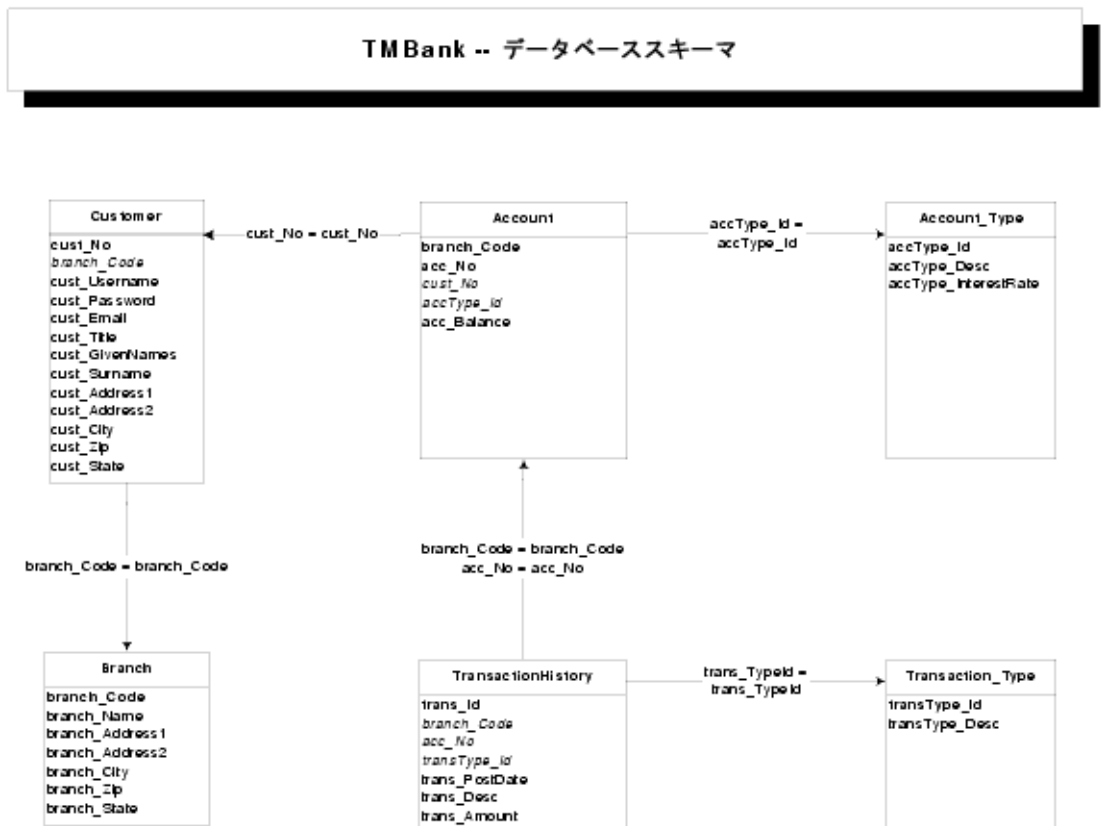
iBank データベーススキーマは次のビジネスルールに基づいて決定されます。

- iBank 企業は主要都市に支店を持つ
- 支店は各地域のすべての顧客を管理する
- 顧客は自分の居住地の支店に 1 つ以上の口座を持つ
- 顧客の口座は支店番号と口座番号で一意に識別され、これとは別に顧客はそれぞれ顧客番号を持つ。引落可能な現在の決済残高が口座に記録される
- 口座の種類は当座預金や普通預金などの口座タイプで区別される

- 各口座タイプには、支店や顧客に関わらず、このタイプのすべての口座に適用される金利や当座借越限度額などの細則が保存される
- 顧客の口座に入金があった場合、または口座からの引き落としがあった場合、その取引は取引履歴と呼ばれるすべての取引のログに記録される
- 取引履歴には支店コード、口座番号、記帳（記録）日付、取引タイプ識別コード、特定の取引に対する補足説明、および取引金額など、各取引についての詳細な情報が保存される
- 現金預金、クレジットカード支払、口座間の送金など、様々な取引が取引タイプで識別される

これらのビジネスルールを図解したものが次のエンティティ関連図です。

図 A-1 データベーススキーマ



次に示すのはデータベースモデルから作成された表定義です。主キー列は太字、外部キー列は斜体で示されます。

| BRANCH | | | |
|-----------------|-------------|----------|-----------------|
| BRANCH_CODE | CHAR(4) | NOT NULL | 支店を識別する 4 桁のコード |
| BRANCH_NAME | VARCHAR(40) | NOT NULL | 支店名 |
| BRANCH_ADDRESS1 | VARCHAR(60) | NOT NULL | 支店の住所、1 行目 |
| BRANCH_ADDRESS2 | VARCHAR(60) | | 支店の住所、2 行目 |
| BRANCH_CITY | VARCHAR(30) | NOT NULL | 支店の所在都市 |
| BRANCH_ZIP | VARCHAR(10) | NOT NULL | 支店の郵便番号 |
| BRANCH_STATE | CHAR(2) | NOT NULL | 支店の住所、都市名略称 |

| CUSTOMER | | | |
|-----------------|-------------|----------|----------------------|
| CUST_NO | INT | NOT NULL | iBank 顧客番号 (グローバル) |
| BRANCH_CODE | CHAR(4) | NOT NULL | この顧客が口座を持つ支店 |
| CUST_USERNAME | VARCHAR(16) | NOT NULL | 顧客のログインユーザー名 |
| CUST_PASSWORD | VARCHAR(10) | NOT NULL | 顧客のログインパスワード |
| CUST_EMAIL | VARCHAR(40) | | 顧客の電子メールアドレス |
| CUST_TITLE | VARCHAR(3) | NOT NULL | 顧客の称号 |
| CUST_GIVENNAMES | VARCHAR(40) | NOT NULL | 顧客の名 |
| CUST_SURNAME | VARCHAR(40) | NOT NULL | 顧客の姓 |
| CUST_ADDRESS1 | VARCHAR(60) | NOT NULL | 顧客の住所、1 行目 |
| CUST_ADDRESS2 | VARCHAR(60) | | 顧客の住所、2 行目 |
| CUST_CITY | VARCHAR(30) | NOT NULL | 顧客の居住都市 |
| CUST_ZIP | VARCHAR(10) | NOT NULL | 顧客の郵便番号 |
| CUST_STATE | CHAR(2) | NOT NULL | 顧客の住所、都市名略称 |

| ACCOUNT_TYPE | | | |
|--------------|---------|----------|--------------|
| ACCTYPE_ID | CHAR(3) | NOT NULL | 3 桁の口座タイプコード |

| | | | |
|----------------------|--------------|-------------|----------|
| ACCTYPE_DESC | VARCHAR(30) | NOT NULL | 口座タイプの説明 |
| ACCTYPE_INTERESTRATE | DECIMAL(4,2) | DEFAULT 0.0 | 年利 |

| | | | |
|-------------|---------------|-------------|------------------------|
| ACCOUNT | | | |
| BRANCH_CODE | CHAR(4) | NOT NULL | 支店コード (主キー構成要素 1) |
| ACC_NO | CHAR(8) | NOT NULL | 口座番号 (主キー構成要素 2) |
| CUST_NO | INT | NOT NULL | 口座名義人 |
| ACCTYPE_ID | CHAR(3) | NOT NULL | 口座タイプ。ACCOUNT_TYPE を参照 |
| ACC_BALANCE | DECIMAL(10,2) | DEFAULT 0.0 | 決済後の利用可能残高 |

| | | | |
|------------------|-------------|----------|-------------|
| TRANSACTION_TYPE | | | |
| TRANSTYPE_ID | CHAR(4) | NOT NULL | 4桁の取引タイプコード |
| TRANSTYPE_DESC | VARCHAR(40) | NOT NULL | 取引コードの説明 |

| | | | |
|---------------------|---------------|----------|--------------------------|
| TRANSACTION_HISTORY | | | |
| TRANS_ID | LONGINT | NOT NULL | 全取引のシリアル番号 |
| BRANCH_CODE | CHAR(4) | NOT NULL | ACCOUNT 参照キー構成要素 1 |
| ACC_NO | CHAR(8) | NOT NULL | ACCOUNT 参照キー構成要素 2 |
| TRANSTYPE_ID | CHAR(4) | NOT NULL | TRANSACTION_TYPE を参照する項目 |
| TRANS_POSTDATE | TIMESTAMP | NOT NULL | 取引の記帳日付および時刻 |
| TRANS_DESC | VARCHAR(40) | | 取引に関する追記事項 |
| TRANS_AMOUNT | DECIMAL(10,2) | NOT NULL | 取引額 |

アプリケーション間の移動とロジック

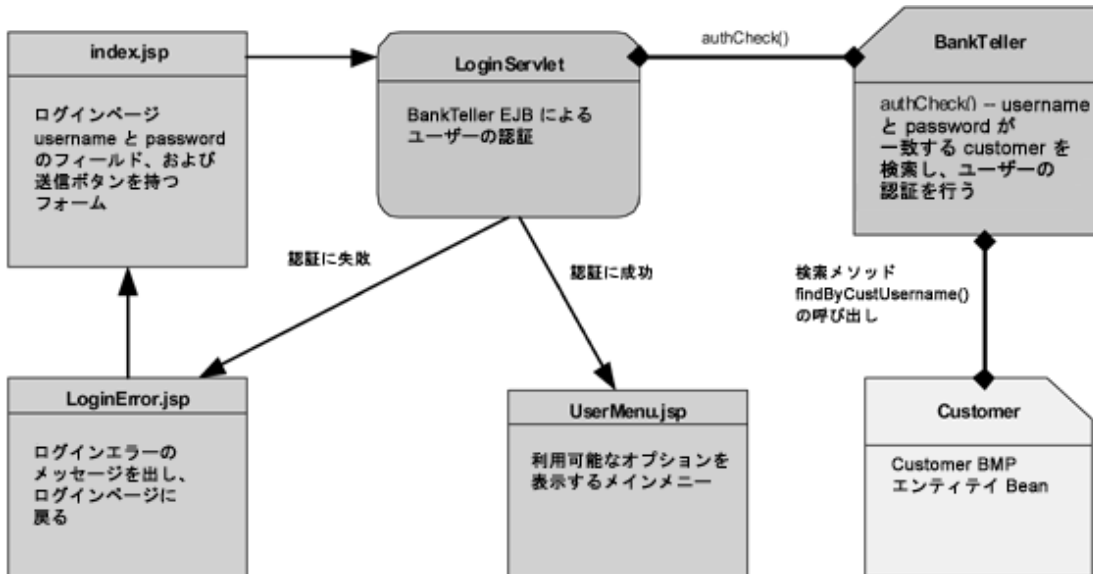
アプリケーション間の移動についての上位レベルビュー

図 A-2 アプリケーション間の移動とロジック



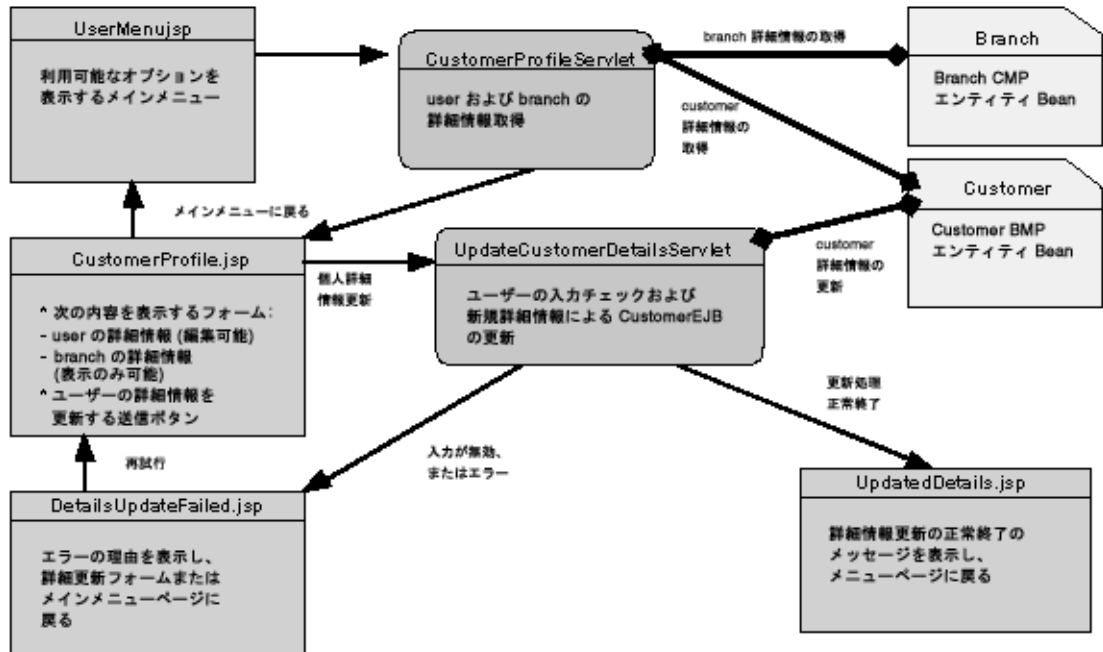
- ログインのプロセス

図 A-3 ログインのプロセス



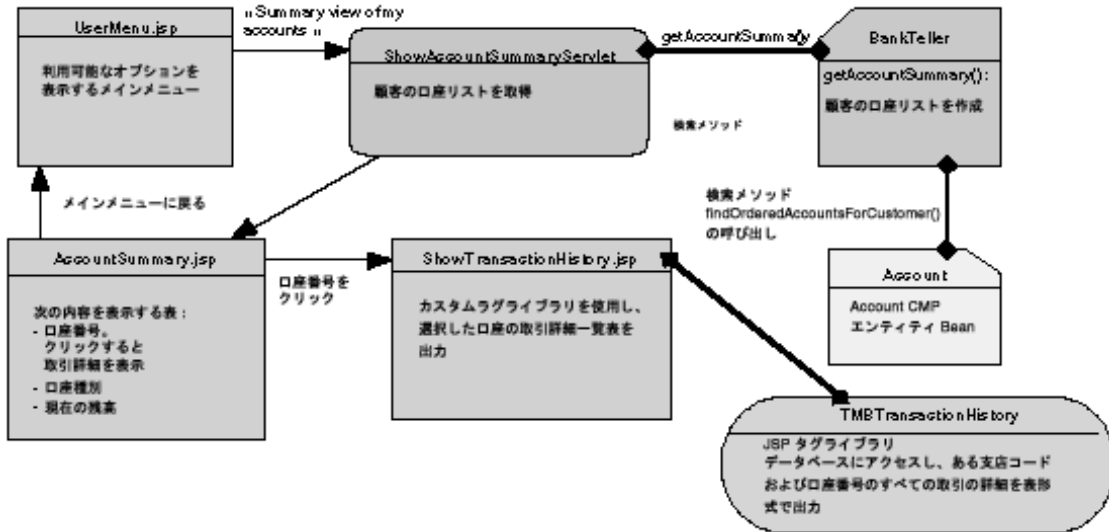
- 詳細情報の表示 / 編集

図 A-4 詳細情報の表示 / 編集



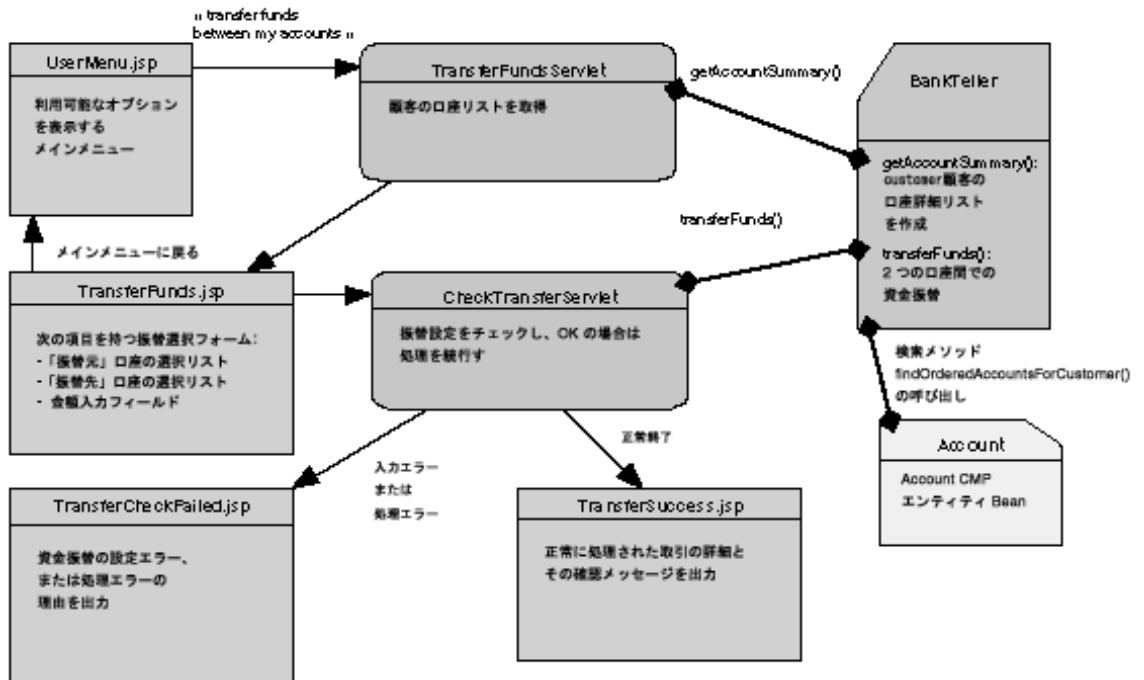
- 口座一覧と取引履歴

図 A-5 口座一覧と取引履歴



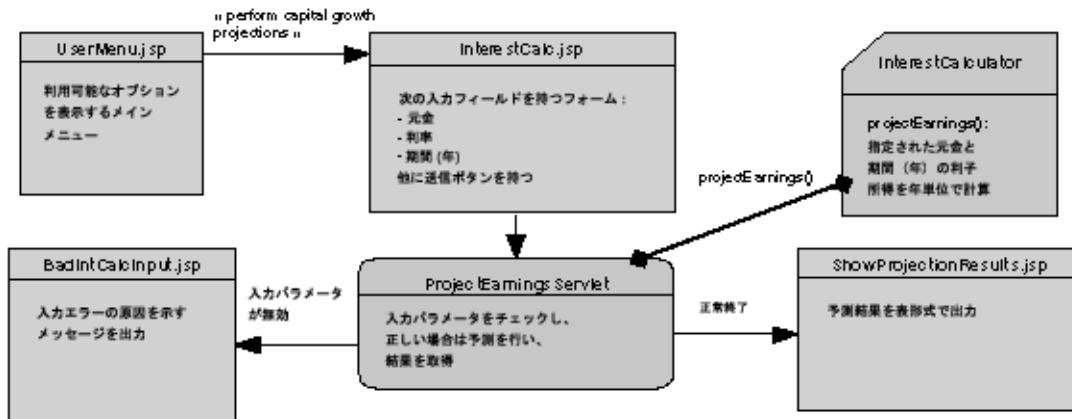
- 資金振替

図 A-6 資金振替



- 利息計算

図 A-7 利息計算



アプリケーションコンポーネント

- データコンポーネント

データベーススキーマの各表はエンティティ Bean としてカプセル化されます。

| エンティティ Bean | データベーステーブル |
|-----------------|-----------------------|
| Account | ACCOUNT 表 |
| AccountType | ACCOUNT_TYPE 表 |
| Branch | BRANCH 表 |
| Customer | CUSTOMER 表 |
| Transaction | TRANSACTION_HISTORY 表 |
| TransactionType | TRANSACTION_TYPE 表 |

ほとんどすべてのエンティティ Beans がコンテナ管理による持続性 (CMP) を使用します。ただし Customer は例外で、Bean 管理による持続性 (BMP) を使用します。

現在アプリケーション側では Account、AccountType、Branch、および Customer beans だけを使用しています。

- ビジネスコンポーネント

アプリケーションのビジネスコンポーネントはセッション Beans でカプセル化されます。

BankTeller Bean はステートフルセッション Bean であり、顧客とシステムとの対話をすべてカプセル化します。BankTeller は特に次の目的で使用されます。

- `authCheck()` メソッドによる顧客の認証
- `getAccountSummary()` メソッドによる顧客の口座リスト取得
- `transferFunds()` メソッドによる 1 人の顧客の口座間の資金振替

InterestCalculator Bean はステートレスセッション Bean であり、資金計算をカプセル化します。`projectEarnings()` メソッドによる複利計算で使用されます。

- アプリケーションロジックコンポーネント (サーブレット)

| コンポーネント名 | 目的 |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LoginServlet | BankTeller セッション Bean (<code>authCheck()</code> メソッド) によるユーザーの認証、HTTP セッションの生成、およびセッションのユーザー情報保存を行う。正常に認証された場合は要求をメインメニューページ (<code>UserMenu.jsp</code>) に転送する |
| CustomerProfileServlet | 顧客および支店に関する詳細情報をそれぞれのエンティティ Beans から取得し、要求を「 <code>view/edit details</code> 」ページ (<code>CustomerProfile.jsp</code>) に転送する |
| UpdateCustomerDetailsServlet | <code>CustomerProfile.jsp</code> での顧客の詳細情報の変更が妥当かどうかをチェックし、妥当な場合は顧客のエンティティ Bean を更新して反映する。処理が正常に実行された場合は <code>UpdatedDetails.jsp</code> にリダイレクトし、入力不正の場合は <code>DetailsUpdateFailed.jsp</code> にリダイレクトする |
| ShowAccountSummaryServlet | 顧客の口座リストを BankTeller セッション Bean から <code>getAccountSummary()</code> メソッドで取得し、要求を <code>AccountSummary.jsp</code> に転送して表示する |
| TransferFundsServlet | 顧客の口座リストを BankTeller セッション Bean から <code>getAccountSummary()</code> メソッドで取得後、要求を <code>TransferFunds.jsp</code> に転送し、ユーザーによる転送設定を許可する |
| CheckTransferServlet | ユーザーが振替元および振替先として指定した口座をチェックし、入力した金額の振替ができるかどうかを確認する。BankTeller セッション Bean の <code>transferFunds()</code> メソッドを呼び出し、振替を行う。入力または処理エラーの場合は <code>CheckTransferFailed.jsp</code> にリダイレクトし、処理が正常に実行された場合は <code>TransferSuccess.jsp</code> にリダイレクトする |

| | |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ProjectEarningsServlet | InterestCalc.jsp でユーザーが定義した利息計算パラメータを取得し、InterestCalculator ステートレスセッション Bean の projectEarnings() メソッドを呼び出して計算を行う。結果を ShowProjectionResults.jsp ページに転送して表示する。入力が無効な場合は BadIntCalcInput.jsp にリダイレクトする |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

- プレゼンテーションロジックコンポーネント (JSP ページ)

| コンポーネント名 | 目的 |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| index.jsp | ログインページとしても使用されるアプリケーションの索引ページ |
| LoginError.jsp | 入力されたユーザー証明書が無効な場合に表示されるログインエラーのページ。ログインできなかった理由を示すメッセージを表示する |
| Header.jsp | アプリケーションのすべての HTML ページに動的に追加されるページヘッダ |
| CheckSession.jsp | このページはアプリケーションのすべてのページに静的に追加され、ユーザーがログインしているかどうか、つまり有効な HTTP セッションを保有しているかどうかを確認する。有効なセッションを保有していない場合は、NotLoggedIn.jsp ページにリダイレクトされる |
| NotLoggedIn.jsp | 最初にログインの手順を実行せずに、アプリケーションのページにアクセスしようとした場合にリダイレクトされるページ |
| UserMenu.jsp | 正常にログインするとリダイレクトされる、メインのアプリケーションメニューページ。実行できるすべての操作に対するリンクを持つ |
| CustomerProfile.jsp | 編集可能な顧客の詳細情報や支店の詳細情報が表示されるページ。このページで住所変更が可能 |
| UpdatedDetails.jsp | 詳細情報の正常更新後にリダイレクトされるページ |
| DetailsUpdateFailed.jsp | 入力エラーのため詳細情報を更新できない場合にリダイレクトされるページ |
| AccountSummaryPage.jsp | ある顧客の持つ複数の口座を一覧表として表示するページ。口座番号、口座のタイプ、現在の残高を表示する。表の口座番号をクリックすると、その口座の詳細な取引履歴が表示される |
| ShowTransactionHistory.jsp | ある口座の詳細な取引履歴を表示するページ。カスタムタグライブラリを使用して表示を行う |
| TransferFunds.jsp | ある口座から別の口座への指定された金額の振替を設定するページ |

| | |
|---------------------------|-------------------------------------------------------------------|
| TransferCheckFailed.jsp | 選択された資金振替の設定が正しくない場合にリダイレクトされるページ |
| TransferSuccess.jsp | ユーザーが設定した送金が正常に実行された場合に表示されるページ。確認メッセージが表示される |
| InterestCalc.jsp | 複利計算パラメータ入力用のページ |
| BadIntCalcInput.jsp | 複利計算パラメータが正しくない場合にリダイレクトされるページ |
| ShowProjectionResults.jsp | 利息計算が正常に実行された場合にリダイレクトされるページ。予測結果を表形式で表示する |
| Logout.jsp | アプリケーションの終了ページ。ユーザーに関連付けられたステートフルセッション Bean を削除し、HTTP セッションを無効化する |
| Error.jsp | 予期しないエラーが発生するとリダイレクトされるページ。発生したエラーの詳細情報を表示する |

移行時に発生する問題を考慮した最適な設計の選択

アプリケーションの設計については、特に「実際の」コンテキストで議論すべきものが多いのは確かですが、一般的な J2EE アプリケーションの移行で発生する可能性がある問題を、サンプルアプリケーションでできるだけ網羅することも考慮する必要があります。

このセクションでは J2EE アプリケーションの移行時に発生する可能性がある問題を検討し、移行時にこれらの問題をチェックするために iBank に用意されているコンポーネントについて説明します。

選択された移行を処理するための次のテクノロジーに焦点を当てて説明します。

サーブレット

iBank アプリケーションにはサーブレットがいくつか用意されており、以下についての潜在的な問題を特定できます。

- サーブレット API の一般的な機能の使用
- HTTP セッションおよび HTTP 要求の属性の保存または取得
- サーブレットコンテキスト初期化パラメータの取得
- ページのリダイレクト

Java Server Pages

JSP の仕様に関しては次の面を重視しています。

- JSP 宣言、スクリプトレット、式、およびコメントの使用

- 静的なインクルード (<%@ include file="..." %>)。特に CheckSession.jsp ファイルの各ページへの追加でテスト
- 動的なインクルード (<jsp:include page=... />)。各ページへの Header.jsp の動的なインクルードに対応
- カスタムタグライブラリの使用。カスタムタグライブラリを ShowTransactionHistory.jsp で使用
- JSP 例外処理のエラーページ。アプリケーションエラーリダイレクト用の Error.jsp ページ

JDBC

iBank アプリケーションは接続プールおよびデータソース経由でデータベースにアクセスします。BMP エンティティ Bean、BankTeller セッション Bean、カスタムタグライブラリでアクセスロジックをプログラミングすることもでき、CMP エンティティ Beans でアクセス方法を宣言することもできます。

Enterprise Java Beans (EJB)

iBank アプリケーションでは様々な Enterprise Java Beans を使用しています。

エンティティ Beans :

Bean 管理による持続性 (Customer Bean) では、次のテストが可能です。

- 初期コンテキストの JNDI 検索
- JDBC 経由のプールデータソースアクセス
- BMP カスタム検索の定義 ("findByCustUsername()")

コンテナ管理による持続性 ("Account" および "Branch" Beans) では、次のテストが可能です。

- 開発ツールを使用する、配備記述子内のオブジェクトとリレーショナル間の (O/R) マッピング
- 複合主キーの使用 ("Account")
- "Account" Bean とその findOrderedAccountsForCustomer() メソッドを使用するカスタム CMP 検索の定義。これによって配備記述子のクエリロジック宣言の差異を確認し、オブジェクト集計を返す場合の複雑な例を見ることができる

セッション Beans :

ステートレスセッション Beans。InterestCalculator では次のテストが可能です。

- ステートレスセッション Bean の使用と配備
- 計算用ビジネスメソッドの呼び出し

ステートフルセッション Beans。BankTeller では次のテストが可能です。

- JNDI と初期コンテキストを使用する様々なインタフェースの検索
- JDBC によるデータベースクエリ
- Bean methods の様々なトランザクション属性の使用
- コンテナ境界設定トランザクションの使用
- 呼び出し間の会話型ステートの維持
- "getAccountSummary()" メソッドなど、エンティティ Beans のフロントエンドとして動作するビジネスメソッド

アプリケーションアセンブリ

iBank は次の標準 J2EE プロシージャによってアセンブルされます。次のコンポーネントがあります。

- Web アプリケーションモジュールには Web アプリケーションアーカイブファイルを使用し、EJB には EJB-JAR アーカイブを使用
- エンタープライズアプリケーションアーカイブファイル (EAR ファイル) は、Web アプリケーションおよび EJB モジュールの最終パッケージ化で使用

移行に関する資料

ここでは、J2EE アプリケーションコンポーネントの、Sun ONE Application Server 7 Enterprise Edition 環境への移行時に役立つ情報について、概要を説明します。

この付録には次の項目があります。

- [その他のアプリケーションサーバー環境からのアプリケーションの移行](#)
- [移行ツール](#)
- [参考資料](#)
- [移行したアプリケーションの再配備](#)

その他のアプリケーションサーバー環境からのアプリケーションの移行

J2EE テクノロジを元に構築されたアプリケーションは、あるアプリケーションサーバーの環境から別の環境へ移行する場合に、それほど大きなソースコードの変更はありません。この節では、他のアプリケーションサーバー環境で作成されたアプリケーションを Sun ONE Application Server 7, Enterprise Edition に移行する場合に役立つリソースを紹介します。

BEA WebLogic から Sun ONE Application Server 7 への移行

J2EE テクノロジベースのアプリケーションを BEA Web Logic Application Server 5.1 または 6.x から Sun ONE Application Server 7 に移行する方法については、次の URL を参照してください。

http://www.sun.com/migration/ResourceGuides_BEAS5.1wpr1a.pdf

http://www.sun.com/migration/ResourceGuides_BE6.1r1awp.pdf

これらのガイドには、移行方法の概要、移行プロセス、移行時の注意事項のほか、サンプルアプリケーションを使った実際の移行例が記載されています。

IBM WebSphere から Sun ONE Application Server 7 への移行

J2EE テクノロジベースのアプリケーションを IBM Web Sphere Application Server から Sun ONE Application Server 7 に移行する方法については、次の URL を参照してください。

http://www.sun.com/migration/ResourceGuides_WAS4.0r1wp.pdf

このガイドは、移行元アプリケーションサーバーと移行先アプリケーションサーバーについて、アーキテクチャと機能の面に注目して説明しています。移行時の問題とサンプルアプリケーションを使った実際の移行例も付属しています。

移行ツール

Sun ONE Application Server は、各種 J2EE アプリケーションコンポーネントの移行を自動化するさまざまな移行ツールをサポートしています。この節では、サポートされているツールに関するリソース情報を示します。

Sun ONE Studio Enterprise Edition for Java、リリース 4.1

Sun ONE Studio for Java Development Environment では、J2EE アプリケーションの各種コンポーネントを Sun ONE Application Server 7 に移行できます。

このツールを使ってアプリケーションを移行する方法については、[第 6 章「6.5 アプリケーションの Sun ONE Studio へのインポート」](#)を参照してください。

ツールとその機能に関する一般情報は、Sun ONE Studio のマニュアルに記載されています。URL は次のとおりです。

<http://docs.sun.com/db/coll/790.3>

Sun ONE Migration Tool for Application Server

Sun ONE Migration Tool 3.x/4.x for Application Servers では、大幅なソースコードの修正なしに J2EE アプリケーションの Sun ONE Application Server 7 への移行を自動化します。

このツールの主要機能は次のとおりです。

- アプリケーションサーバー固有の配備記述子の移行
- 選択されたカスタム JavaServer Pages™ (JSP™) タグと独自の API のランタイムサポート
- 選択された設定パラメータと Sun ONE Application Server における同等の機能の変換
- ターゲットアプリケーションサーバーに移行したアプリケーションを構築および配備する Ant ベースのスクリプトの自動生成
- 移行後の包括的移行レポートの生成

Sun ONE Migration Tool for Application Server は、次の URL からダウンロードできます。

<http://www.sun.com/migration/sunonetools.html#1>

詳しいインストール方法と使用方法は、オンラインヘルプに記載されています。

Sun ONE Migration Toolbox for Applogic and NetDynamics

Sun ONE Migration Toolbox は、NetDynamics または Kiva/NAS プラットフォームで構築されたアプリケーションを Sun ONE Application Server または J2EE 互換コンテナに移行するために使用されます。Sun ONE Migration Toolbox の主要インタフェースは、Toolbox アプリケーションです。このアプリケーションは
%MIGTBX_HOME%/bin/toolbox.bat スクリプトを実行して呼び出します。ただし setenv.bat ファイルが適切にカスタマイズされていなければなりません。詳細については README.txt を参照してください。

Toolbox を使った移行に対応しているソースプラットフォームは、次のとおりです。

- Windows NT 4.0
- Windows 2000

Windows 95/98/Me など、上記以外の Windows プラットフォームでも動作すると思われませんが、これらについてはテストしていないため、Sun ONE Migration Toolbox インストールのマニュアルに記載された以外の追加設定が必要となることがあります。

Toolbox を正常に実行するためには JDK 1.2.2 が必要です。JDK 1.3.1 はテストされています。

Sun One Connector Builder

Connector Builder には、J2EE Connector Architecture 1.0 for Enterprise Information Systems (EIS) と従来のアプリケーションに対応したリソースアダプタを構築するための各種ツール、コンポーネント、ライブラリが付属しています。Connector Builder で構築したリソースアダプタを使用すると、標準的な方法で、J2EE アプリケーションから外部システムに簡単にアクセスできます。生成されたリソースアダプタのアクセスには、J2EE CA に記載されているとおり、Common Client Interface (CCI) が使用されます。これらのリソースアダプタには、Simple Object Access Protocol (SOAP) を使ってアクセスすることもできます。この場合、アプリケーションを統合する Web サービスソリューションが提供されます。

試用版ツールは、次の URL からダウンロードできます。

http://www.sun.com/software/products/connector_builder/home_connector_builder.html

ツールの使用方法は、Connector Builder ツールのマニュアルに記載されています。次の URL を参照してください。

<http://docs.sun.com/db/coll/s1.conbldr>

Native Connector Toolkit

この節では、Connector Builder を使って従来の C/C++ API 向けリソースアダプタを構築する方法について説明します。

Connector Builder を使用する場合は、Java API が必要になります。ところが、従来の EIS やアプリケーションのほとんどは C/C++ API しか提供していません。この節では、Connector Builder を使って、こうした従来の API 向けのリソースアダプタを構築する方法について説明します。

最初に、C/C++ API の Java (JNI) ラッパーを作成する必要があります。Connector Builder の仕様では、IDE の Native Connector Toolkit (NCT) ツールの使用を推奨しています。NCT は、JNI を使って C/C++ アプリケーションの Java ラッパーをすばやく生成する使いやすいツールです。Connector Builder で EIS API を定義する際、NCT によって生成された出力 Java ラッパーファイルが入力ファイルになります。

Native Connector Toolkit の詳細は、Toolkit のヘルプと、Connector Builder ツールのマニュアルに記載されています。Connector Builder ツールのマニュアルの URL は次のとおりです。

<http://docs.sun.com/db/coll/s1.conbldr>

J2EE Application Verification Kit

Java Application Verification Kit (AVK) for the Enterprise では、特定のガイドラインおよび規則に従って、J2EE API を正しく使用してその他の J2EE 対応アプリケーションサーバーに移行できるようなアプリケーションを構築し、テストすることができます。

Java Application Verification Kit (AVK) は、次の URL からダウンロードできます。

<http://java.sun.com/j2ee/verified/>

参考資料

この節では、参考用として、Sun ONE Application Server 6.x の移行マニュアルを紹介します。

Sun ONE Application Server 6.0 への移行

KIVA/NAS/NetDynamics のアプリケーションを Sun ONE Application Server 6.0 に移行する方法については、『Sun ONE Application Server Migration Guide』を参照してください。URL は次のとおりです。

<http://docs.sun.com/db/doc/816-5780-10>

Sun ONE Application Server 6.5 への移行

KIVA/NAS/NetDynamics のアプリケーションを Sun ONE Application Server 6.5 に移行する方法については、『Sun ONE Application Server 6.5 Migration Guide』を参照してください。URL は次のとおりです。

<http://docs.sun.com/db/doc/816-5793-11>

Sun ONE Application Server 7 への移行

KIVA/NAS/NetDynamics のアプリケーションを Sun ONE Application Server 7 に移行する方法については、『サーバーアプリケーションの移行および再配備ガイド』を参照してください。URL は次のとおりです。

<http://docs.sun.com/db/doc/817-2158-10>

移行したアプリケーションの再配備

利用可能な移行ツールで自動的に移行されたアプリケーションでは、ほとんどの場合、『Sun ONE Application Server 管理者ガイド』で説明する標準配備タスクを使用します。

場合によっては、自動移行では、ソースアプリケーションから特定のメソッドやシンタックスを移行できないことがあります。Sun ONE Migration Tool for Application Server でこの問題に直面した場合、移行を完了するのに必要な手順があります。この場合には、移行後の手動による手順をいったん終了した後、『Sun ONE Application Server 管理者ガイド』で説明する標準方式でアプリケーションを配備できます。

Enterprise Java Beans 1.1 仕様から Enterprise Java Beans 2.0 仕様への移行

Sun ONE Application Server 7 でも Enterprise Java Beans™ (EJB) 1.1 仕様は引き続きサポートされますが、その拡張機能を活用するため、EJB 2.0 アーキテクチャの使用をお勧めします。EJB 1.1 を EJB 2.0 へ移行するには、コンポーネントのソースコード内部を含め、多くの修正が必要です。

この章には次の節があります。

- [EJB 1.1 と EJB 2.0 の相違点](#)
- [EJB クライアントアプリケーションの移行](#)
- [CMP エンティティ EJB の移行](#)
- [Bean クラスの移行](#)
- [ejb-jar.xml の移行](#)
- [カスタム検索メソッド](#)

EJB 1.1 と EJB 2.0 の相違点

基本的に、必要とされる修正は、EJB 1.1 と EJB 2.0 の間の相違に関連しています。これらの相違のすべてについて、次の各項目で説明します。

- [EJB クエリ言語](#)
- [ローカルインタフェース](#)
- [EJB 2.0 コンテナ管理による持続性 \(CMP\)](#)
- [持続性フィールドの定義](#)
- [エンティティ Bean の関係の定義](#)

- [メッセージ駆動型 Beans](#)

EJB クエリ言語

EJB 1.1 仕様には、個別のアプリケーションサーバーに対する検索メソッドのクエリを形成し、記述する方法および言語が残されています。多くのアプリケーションサーバーのベンダーは、開発者が SQL を使用してクエリを作成するよう求めますが、一方で、特定のアプリケーションサーバー製品に固有の、独自の専用言語を使用する場合もあります。このようにクエリの実装が混在しているため、アプリケーションサーバー間の不整合が生じています。

EJB 2.0 仕様では、これらの不整合および不備を修正した *EJB Query Language*、または *EJB QL* と呼ばれるクエリ言語が導入されています。EJB QL は、SQL92 に基づいています。EJB QL は、検索と選択メソッドの両形式で、特にコンテナ管理による持続性を使用したエンティティ Beans 用にクエリメソッドを定義します。SQL に対する EJB QL の主な長所は、EJB コンテナを通じた移植性と、エンティティ Beans の関係をナビゲートする機能にあります。

ローカルインタフェース

EJB 1.1 アーキテクチャでは、セッション Beans およびエンティティ Beans にリモートインタフェースという 1 つのインタフェースタイプがあり、そのインタフェースを通じて、クライアントや他のアプリケーションコンポーネントからのアクセスが可能になります。リモートインタフェースは、Bean インスタンスにリモート機能があるものとして設計されています。Bean は RMI からそれを継承し、ネットワークを通じて分散クライアントとやり取りを行うことができます。

EJB 2.0 では、セッション Beans およびエンティティ Beans はクライアントにメソッドを公開するインタフェースを 2 種類利用できます。1 つはリモートインタフェース、もう 1 つはローカルインタフェースです。2.0 リモートインタフェースは、1.1 アーキテクチャで使用されているリモートインタフェースと同じものです。Bean は RMI からリモートインタフェースを継承し、ネットワーク層を通じてそのメソッドを公開します。また、分散クライアントとやり取りするための同一の機能を備えています。

ただし、セッションおよびエンティティ Beans 用のローカルインタフェースは、ローカルクライアント、つまり同一の EJB コンテナに共存しているクライアントである EJB からの軽量アクセスをサポートしています。EJB 2.0 仕様では、さらに、ローカルインタフェースを使用した同一アプリケーション内にある EJB を必要とします。つまり、ローカルインタフェースを使用したアプリケーションの EJB 用の配備記述子が、1 つの ejb-jar ファイルに含まれている必要があります。

ローカルインタフェースは、標準 Java インタフェースです。これは、RMI からは継承されません。Enterprise JavaBeans は、ローカルインタフェースを使用して、同一コンテナ内にある他の Beans にメソッドを公開します。ローカルインタフェースを使用することにより、Bean はクライアントと緊密に連携でき、リモートメソッド呼び出しのオーバーヘッドなしに直接アクセスが可能になります。

さらに、ローカルインタフェースは、参照セマンティクスにより Beans 間で渡される値を許可します。この方法では、オブジェクト自体ではなくオブジェクトへの参照を渡すため、大量のデータを持つオブジェクトを渡す際に発生するオーバーヘッドが軽減し、パフォーマンス上の利点があります。

リモートインタフェースではなくローカルインタフェースを使用すると、セッションまたはエンティティ Bean の設定は簡単になります。クライアントに公開されている Bean のメソッド経由のローカルインタフェースは、EJBObject ではなく EJBLocalObject を拡張します。同様に、Bean のホームインタフェースは、EJBHome ではなく EJBLocalHome を拡張します。実装クラスは、同じ EntityBean または SessionBean インタフェースを拡張します。

| | |
|---|---------------------------------------------------------------------------------------------------|
| 注 | EJB 2.0 でリモートに予定されている Bean は、リモートインタフェースで EJBObject を、ホームインタフェースで EJBHome を、EJB 1.1 の場合と同様に拡張します。 |
|---|---------------------------------------------------------------------------------------------------|

EJB 2.0 コンテナ管理による持続性 (CMP)

EJB 2.0 仕様では、CMP を拡張し、CMP 間で関係のある複数のエンティティ Beans を許可します。これは、コンテナ管理による関係 (CMR) と呼ばれます。コンテナでは、関係と、関係の参照整合性を管理します。

EJB 1.1 仕様で提示されていた CMP モデルは、より制限の多いものでした。1.1 アーキテクチャでの CMP は、データベースやリソースマネージャタイプから独立したデータアクセスに制限されていました。CMP は、リモートインタフェースを経由したエンティティ Bean のインスタンスの状態表示だけを行い、Bean の関係は表示しませんでした。CMP の 1.1 バージョンでは、エンティティ Bean クラスのインスタンス変数を、データベースまたはリソースマネージャで状態を表すデータ項目にマッピングする必要があります。CMP インスタンスフィールドは配備記述子で指定されており、Bean を配備する際に、記述子はツールを使用してインスタンスフィールドのデータ項目へのマッピングを実装するコードを生成します。

Bean の実装クラスのコーディング方法も変更する必要があります。2.0 仕様に従って CMP を使用するエンティティ Bean 用の実装クラスは、抽象クラスとして定義されます。

持続性フィールドの定義

EJB 2.0 仕様では、エンティティ Bean のインスタンス変数を CMP フィールドまたは CMR フィールドとして指定できます。これらのフィールドは、配備記述子で定義します。CMP フィールドは、cmp-field 要素でマークされ、コンテナ管理による関係のフィールドは、cmr-field 要素でマークされます。

実装クラスでは、CMP および CMR フィールドを public 変数として宣言しないことに注意してください。代わりに、エンティティ Bean で get および set メソッドを定義して、これらの CMP および CMR フィールドの値を取得し、設定することができます。このような意味で、2.0 CMP を使用した Beans は、JavaBeans モデルに準拠しています。クライアントは、インスタンス変数に直接アクセスする代わりに、エンティティ Bean の get および set メソッドを使用してこれらのインスタンス変数を取得し、設定します。get および set メソッドは、CMP または CMR フィールドとして指定された変数にのみ関連しています。

エンティティ Bean の関係の定義

前述したとおり、EJB 1.1 アーキテクチャでは、エンティティ Beans 間の CMR をサポートしていません。EJB 2.0 アーキテクチャは、1 対 1、および 1 対多の CMR を両方ともサポートしています。関係は CMR フィールドを使用して表現されており、これらのフィールドは、配備記述子に関係としてマークされています。アプリケーションサーバーで適切な配備ツールを使用して、配備記述子に CMR フィールドを設定します。

CMP フィールドと同様、Bean はインスタンス変数として CMR フィールドを宣言しません。代わりに、Bean は、これらのフィールド用に get および set メソッドを備えています。

メッセージ駆動型 Beans

メッセージ駆動型 Beans は、EJB 2.0 アーキテクチャで導入されたもう一つの新しい機能です。メッセージ駆動型 Beans は、Java Message Service (JMS) 経由で配布された非同期メッセージを処理するトランザクション認識型のコンポーネントです。JMS API は、J2EE 1.3 プラットフォームの重要な部分です。

非同期メッセージングでは、アプリケーションはメッセージの交換によって通信することができるため、送信者と受信者を独立させることができます。送信者は、メッセージを送信した後に、受信者がそのメッセージを受信または処理するのを待つ必要はありません。これは、同期通信のプロセスとは異なります。同期通信では、別のコンポーネント上のメソッドを呼び出し、処理が完了して呼び出しコンポーネントに制御が戻るまで待機またはブロックするコンポーネントが必要です。

EJB クライアントアプリケーションの移行

この節には次の項目があります。

- [JNDI コンテキスト内での EJB の宣言](#)
- [EJB JNDI 参照の使用方法の要約](#)

JNDI コンテキスト内での EJB の宣言

Sun ONE Application Server 7 では、EJB は JNDI のサブコンテキスト *ejb/* に計画的にマッピングされています。JNDI 名 *Account* を EJB に所属させると、Sun ONE Application Server 7 は参照 *ejb/Account* をグローバル JNDI コンテキストに自動生成します。したがって、この EJB のクライアントは、対応するホームインタフェースを取得するために *ejb/Account* を検索する必要があります。

Sun ONE Application Server 6.0/6.5 に配備されているサーブレットメソッドのコードを見てみましょう。

ここに示すサーブレットは、JNDI コンテキストのルートにマッピングされているステートフルセッション Bean の *BankTeller* を呼び出します。ここに挙げるコードでは EJB のホームインタフェースを取得し、*BankTeller* オブジェクトのインスタンス化とこのオブジェクトのリモートインタフェース取得を可能にし、ビジネスメソッドからのこのコンポーネントの呼び出しを可能にします。

```
/**
 * Look up the BankTellerHome interface using JNDI.
 */
```

```

private BankTellerHome lookupBankTellerHome(Context ctx)
    throws NamingException
{
    try
    {
        Object home = (BankTellerHome) ctx.lookup("ejb/BankTeller");
        return (BankTellerHome) PortableRemoteObject.narrow(home,
BankTellerHome.class);
    }
    catch (NamingException ne)
    {
        log("lookupBankTellerHome:unable to lookup BankTellerHome" +
            "with JNDI name 'BankTeller':" + ne.getMessage() );
        throw ne;
    }
}

```

このコードでは `ejb/BankTeller` を検索の引数としてすでに使用していますが、Sun ONE Application Server 7 に配備されるコードを変更する必要はありません。

EJB JNDI 参照の使用方法的要約

この節では、EJB JNDI 参照を使用した場合の注意事項をまとめて説明します。ここで説明する注意事項の詳細については、特定のソースアプリケーションサーバーのプラットフォームに固有のものであることに注意してください。

JNDI コンテキスト内の EJB 参照の配置

既存の WebLogic アプリケーションで JNDI コンテキストのルートに EJB がマップされている場合は、上述の JNDI コンテキスト内の EJB 参照の名前だけを変更する必要があります。変更によってこれらの参照が JNDI コンテキストルートからサブコンテキスト `ejb/` に移動します。

これらの EJB が、既存のアプリケーションの JNDI サブコンテキスト `ejb/` にすでにマップされている場合、変更の必要はありません。

ただし、重要なのは、Sun ONE Studio for Java の IDE 内の配備記述子に EJB の JNDI 名を設定する場合に、EJB の JNDI 名の中に接頭辞 `ejb/` を含めないことです。これらの EJB 参照は、Sun ONE Application Server 7 では自動的に JNDI の `ejb/` サブコンテキスト内に配備されることに留意してください。このため、EJB に、その配備記述子で *BankTeller* という JNDI 名が与えられた場合、この EJB への参照は Sun ONE Application Server によって `ejb/BankTeller` に「変換」されます。この EJB のクライアントコンポーネントは検索時にこの JNDI 名を使用することになります。

グローバル JNDI コンテキストとローカル JNDI コンテキスト

EJB 参照を取得するためにグローバル JNDI コンテキストを使用することは、Sun ONE Application Server 7 では完全に有効かつ最適なアプローチです。しかしながら、J2EE 仕様にできるだけ近い形で、EJB クライアントアプリケーションのローカル JNDI コンテキストを経由して EJB 参照を取得することをお勧めします。ローカル JNDI コンテキストを使用する場合、まずクライアント部分の配備記述子で EJB リソース参照を宣言する必要があります。これは Web アプリケーションでは `web.xml`、EJB コンポーネントでは `ejb-jar.xml` になります。

CMP エンティティ EJB の移行

この節では、アプリケーションコンポーネントを EJB 1.1 アーキテクチャから EJB 2.0 アーキテクチャに移行する手順について説明します。

CMP 1.1 の Bean を CMP 2.0 に移行するためには、まず特定の Bean が移行可能であることを確認する必要があります。確認の手順は次のとおりです。

1. `ejb-jar.xml` ファイルから `<cmp-field>` 名へ移動し、オプションのタグ `<prim-key-field>` が `ejb-jar.xml` 内にあり、指示された値を保持していることを確認します。問題がなければ、次の手順に進みます。

`ejb-jar.xml` 内でフィールド名 `<prim-key-class>` を探し、クラス名を取得して、クラス内で宣言された `public instance variables` を取得します。
2. ここで、これらの変数のシグネチャ (名前と大文字小文字の区別) が上記の `<cmp-field>` 名と一致するかどうかを確認します。見つかったフィールドは分割します。これらの分割されたフィールドで、大文字で始まっているものがあるかどうかを調べます。もしある場合は、移行は実行できません。
3. Bean クラスのソースコードを調べて、すべての `<cmp-field>` 変数の `java` 型を取得します。
4. すべての `<cmp-field>` 名を小文字に変更し、それらの名前からアクセサを構築します。たとえば、元のフィールド名が `Name` で、その `java` 型が `String` である場合、アクセサメソッドシグネチャは次のようになります。

```
Public void setName(String name)
```

```
Public String getName()
```

5. これらのアクセサメソッドシグネチャを、Bean クラスのメソッドシグネチャと比較します。正確に一致しない場合は、移行は実行されません。
6. カスタム検索のメソッドシグネチャと、対応する SQL を取得します。SQL 内に Join、Outer join、OrderBy があるかどうかを確認します。ある場合は、移行は実行できません。これは、EJB QL では joins、Outer join、および OrderBy をサポートしていないためです。
7. CMP 1.1 検索が java.util Enumeration を使用している場合は、代わりに java.util.Collection を使用する必要があります。これを反映するようにコードを修正してください。CMP2.0 検索では、java.util Enumeration を返すことができません。

次の項目、「[Bean クラスの移行](#)」では移行プロセスについて説明します。

Bean クラスの移行

この節では、Bean クラスの Sun ONE Application Server への移行に必要な手順について説明します。

1. Bean クラス宣言の先頭にキーワード *abstract* を付けます。たとえば、Bean クラス宣言は次のようになります。

```
Public class CabinBean implements EntityBean // before
modification
```

```
abstract Public class CabinBean implements EntityBean // after
modification
```

2. アクセサの接頭辞にキーワード *abstract* を付けます。
3. 変更後、Bean クラスのソースファイル、つまり拡張子 .java のファイルにすべてのアクセサをクラスレベルで挿入します。
4. Bean クラスのソースファイル内で、すべての cmp フィールドをコメントにします。
5. 小文字の cmp-field 名で *protected* インスタンス変数の宣言を構築し、クラスレベルでそれを挿入します。
6. すべての ejbCreate() メソッド本体 (複数の ejbCreate が存在する場合もある) を読み込みます。パターン「<cmp-field>= 値またはローカル変数」を検索し、「*abstract* ミューテータメソッド名 (値またはローカル変数)」に置き換えます。たとえば、移行前の ejbCreate の本体が次のとおりであるとします。

```
public MyPK ejbCreate(int id, String name)
{
    this.id = 10*id;
    Name = name;//1
    return null;
}
```

移行後、メソッド本体は変更され、次のようになります。

```
public MyPK ejbCreate(int id, String name)
{
    setId(10*id);
    setName(name);//1
    return null;
}
```

| | |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 注 | //1 の abstract アクセサのメソッドシグネチャは、EJB 2.0 仕様で定められた Camel Case 規約に従っています。また、キーワード「 <i>this</i> 」は、元のソース内に存在する場合としない場合がありますが、変更後のソースファイルからは削除する必要があります。 |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------|

7. 手順5で `ejbPostCreate()` メソッドに宣言したすべての **protected** 変数を初期化する必要があります。**protected** 変数の数は、`ejbCreate()` メソッドの数と同じになります。初期化は、次のように初期化コードを挿入することにより行われます。

```
protected String name;//from step 5
protected int id;//from step 5
public void ejbPostCreate(int id, String name)
{
    name /*protected variable*/ = getName();/*abstract accessor*/
    //inserted in this step
    id /*protected variable*/ = getId();/*abstract accessor*/
    //inserted in this step
}
```

8. `ejbLoad` メソッドの内部で、Beans のデータベースの状態に `protected` 変数を設定する必要があります。次のコードを挿入します。

```
public void ejbLoad()
{
    name = getName();//inserted in this step
    id = getId(); //inserted in this step
    ..... //already present code
}
```

9. 同様に、データベースの状態が更新されるように、`ejbStore()` 内部の Beans の状態を更新する必要があります。ただし、`ejbCreate()` の外部の主キーに対応するセッターは更新できないため、このメソッドの内部にはセッターを含めないでください。次のコードを挿入します。

```
public void ejbStore()
{
    setName(name);//inserted in this step
    // setId(id);//Do not insert this if it is a part of the
    primary key
    .....//already present code
}
```

10. Bean クラスソース、つまり拡張子 `.java` のファイルに対する最後の変更は、コード全体を調べ、すべての `<cmp-field>` 変数名を、手順 5 で宣言した対応する `protected` 変数名と置き換えることです。

Bean を移行しない場合、少なくとも `<cmp-version>1.x</cmp-version>` タグを `ejb-jar.xml` 内部の適切な場所に挿入する必要があります。これで、移行しない Bean も Sun ONE Application Server 上で動作し続けます。

ejb-jar.xml の移行

ejb-jar.xml ファイルを Sun ONE Application Server に移行するには、次の手順を実行します。

1. ejb-jar.xml で、すべての <cmp-field> を小文字に変更します。
2. ejb-jar.xml ファイルで、<reentrant> タグの後に <abstract-schema-name> タグを挿入します。このスキーマ名は、<ejb-name> タグと同様、接頭辞 ias_ が付いた Bean 名になります。
3. <primkey-field> タグの後に、次のタグを挿入します。
<security-identity><use-caller-identity/></security-identity>
4. SQL から EJB QL を構築するために、上記で取得した SQL を使用します。
5. <query> タグ、およびネストされた子タグすべてを、すべての必要な情報とともに ejb-jar.xml に挿入します。挿入位置は <security-identity> タグのすぐ後です。

カスタム検索メソッド

カスタム検索メソッドは、エンティティ Bean のホームインタフェースで定義される findBy... メソッドです。デフォルトの findByPrimaryKey メソッドとは異なります。EJB 1.1 仕様は、これらの検索メソッドのロジックを定義する標準を定めていないため、EJB サーバーのベンダーは、実装を自由に選択できます。その結果、メソッドの定義に使用される手順は、ベンダーが選択する実装によって大幅に異なります。

Sun ONE Application Server 6.0 および 6.5 では、ファインダーロジックの指定に標準 SQL を使用します。

EJB の持続性記述子 Account-ias-cmp.xml に格納されるこの検索メソッドの定義に関する情報は、次のとおりです。

```
<bean-property>
  <property>
    <name>findOrderedAccountsForCustomerSQL</name>
    <type>java.lang.String</type>
    <value>
      SELECT BRANCH_CODE,ACC_NO FROM ACCOUNT where CUST_NO = ?
    </value>
    <delimiter>,</delimiter>
```

```

    </property>
</bean-property>
<bean-property>
  <property>
    <name>findOrderedAccountsForCustomerParms</name>
    <type>java.lang.Vector</type>
    <value>CustNo</value>
    <delimiter>,</delimiter>
  </property>
</bean-property>

```

このように、各 findxxx 検索メソッドには、配備記述子 (クエリ用 SQL コード、および関連パラメータ) 内に対応する 2 つのエントリがあります。

Sun ONE Application Server では、カスタム検索メソッドロジックも宣言型ですが、EJB のクエリ言語、EJB QL に基づいています。

EJB-QL 言語自体は使用できません。ejb-jar.xml ファイル内部の、<ejb-ql> タグで指定する必要があります。このタグは、<query> タグの内部にあり、EJB 内にクエリ (検索または選択メソッド) を定義します。EJB コンテナでは、各クエリを検索または選択メソッドの実装に変換できます。ここでは、<ejb-ql> タグの例を示します。

```

<ejb-jar>
  <enterprise-beans>
    <エンティティ>
      <ejb-name>hotelEJB</ejb-name>
      ...
      <abstract-schema-name>TMBankSchemaName</abstract-schema-name>
      <cmp-field>...
      ...
      <query>
        <query-method>
          <method-name>findByCity</method-name>
          <method-params>
            <method-param>java.lang.String</method-param>
          </method-params>
        </query-method>
      <ejb-ql>
        <![CDATA[SELECT OBJECT(t) FROM TMBankSchemaName AS t WHERE
t.city = ?1]]>
      </ejb-ql>
    
```

```
        </query>
    </entity>
    ...
</enterprise-beans>
...
</ejb-jar>
```


索引

A

asadmin, [31](#), [51](#), [92](#), [133](#)

B

BMP, [53](#)

C

CMP, [47](#), [53](#)

D

DriverManager, [40](#)

E

EAR ファイル, [33](#)

EJB, [47](#)

EJB 1.1 から EJB 2.0 への移行

 CMP エンティティ EJB の移行

 Bean クラスの移行, [166](#)

 ejb-jar.xml の移行, [169](#)

 カスタム検索メソッド, [169](#)

 EJB 2.0 コンテナ管理による持続性 (CMP), [161](#)

 ejb-jar.xml の移行, [169](#)

 EJB クエリ言語 (EJB Query Language), [160](#)

 EJB クライアントアプリケーションの移行, [163](#)

 JNDI コンテキスト内での EJB の宣言, [163](#)

 エンティティ Bean の関係の定義, [162](#)

 メッセージ駆動型 Beans, [163](#)

ejbCreate, [106](#)

EJB JAR, [33](#)

EJB QL, [47](#)

EJB の移行, [47](#)

Enterprise JavaBeans, [26](#)

I

iBank, [38](#), [56](#)

 Sun ONE Studio for Java 4.0 を使用した iBank の移行

 EJB モジュールの作成, [113](#)

 Web アプリケーションモジュールの作成, [96](#)

 アプリケーションの配備, [133](#)

 エンタープライズアプリケーションの作成, [130](#)

iBank アプリケーションの手動移行, [72](#)

 EJB の変更, [73](#)

 Web アプリケーションの変更, [72](#)

 配備用アプリケーションのアセンブル, [91](#)

iBank アプリケーションの仕様

 アプリケーション開発用ツール, [136](#)

 アプリケーション間の移動とロジック, [140](#)

J

アプリケーションコンポーネント, [144](#)
移行時に発生する問題を考慮した最適な設計の選択, [147](#)
データベーススキーマ, [136](#)

J

J2EE, [26](#)
J2EE アプリケーション
コンポーネント, [32](#)
J2EE アプリケーションコンポーネントと移行, [32](#)
J2EE コンポーネント標準, [26](#)
JavaServer Pages, [26](#)
JDBC コード, [39](#)
JDBC 2.0 データソースの使用方法, [41](#)
JNDI 経由でのデータソースの検索, [44](#)
データソースの設定, [41](#)
JNDI コンテキスト, [44](#)
JNDI コンテキストからのデータソースの取得, [47](#)
JSP 1.2 仕様, [45](#)
JSP および JSP カスタムタグライブラリ, [45](#)

M

MDB, [47](#)

R

Registry Editor, [29](#)

S

S1AS 6.x から S1AS 7 への移行, [38](#)
S1AS 7 に対応した EJB の変更, [48](#)
Sun ONE Application Server 6.0/6.5 について, [36](#)
Sun ONE Application Server 7 について, [15, 21](#)

Sun ONE Console, [29](#)
Sun ONE Studio, [28, 93](#)

T

Toolbox アプリケーション, [153](#)

W

WAR, [33](#)
WEB-INF, [97](#)
web.xml, [97](#)
Web アプリケーション, [50](#)
Web アプリケーションモジュールの移行, [50](#)
サーブレットおよび JSP の移行時の特定の障害, [51](#)
Web モジュール, [130](#)
Welcome ファイル, [102](#)

あ

アーキテクチャ, [21, 24](#)
Sun ONE Application Server 6.0/6.5 のアーキテクチャ, [36](#)
Sun ONE Application Server 7 のアーキテクチャ, [23](#)
アプリケーションクライアント JAR, [33](#)

い

移行と再配備, [33](#)
再配備とは, [20](#)
移行に関する注意事項および方針, [35](#)

え

エンタープライズ EJB モジュール, [53](#)
エンタープライズアプリケーション, [54](#), [130](#)
 アプリケーションルートコンテキストとアクセス URL, [55](#)
 固有の拡張子の移行, [56](#)
エンティティ Beans, [48](#)

か

開発環境, [27](#)
 Sun ONE Application Server 6.0/6.5, [27](#)
 Sun ONE Application Server 7, [28](#)
カスタマサポート, [13](#)
管理サーバー, [30](#)
管理ツール, [29](#)
 Sun ONE Application Server 6.0, [29](#)
 Sun ONE Application Server 7, [30](#)

さ

サーブレット, [26](#), [46](#)

せ

セッション Beans, [48](#)

て

データソース, [41](#)
データベース接続, [32](#)

は

配備, [133](#)
配備記述子, [19](#), [32](#)

ほ

ホームインタフェース, [114](#)
本書について, [7](#)
 前提事項, [7](#)
 マニュアルの構成, [8](#)

り

リモートインタフェース, [114](#)

