Oracle® Application Server 10*g*

WebLogic からの移行

10*g*（9.0.4）

部品番号 : B14110-01

2004 年 6 月

ORACLE®

Oracle Application Server 10*g* WebLogic からの移行 , 10*g*（9.0.4）

部品番号 : B14110-01

原本名 : Oracle Application Server 10*g* Migrating From WebLogic, 10*g* (9.0.4)

原本部品番号 : B10425-02

原本著者 : Kai Li

目次

3 Java サーブレットの移行

4 JSP ページの移行

5 Enterprise JavaBeans コンポーネントの移行

6 JDBC の移行

A その他の機能比較

索引

はじめに

「はじめに」の項目は次のとおりです。

対象読者

『Oracle Application Server 10*g* WebLogic からの移行』は、管理者や開発者など、高可用性が要求される状況で、Oracle Application Server のデプロイや管理を担当する人たちを対象としています。

このマニュアルの構成

このマニュアルは、次の章で構成されています。

第 1 章「概要」

この章では、WebLogic Server 7.0 から Oracle Application Server に J2EE Web アプリケーションを移行する際の問題点や、必要な作業の概要について説明します。

第 2 章「Oracle Application Server と WebLogic Server の比較」

この章では、Sun 社の J2EE プラットフォームおよびコンポーネントの仕様について、オラクル社の実装と BEA Systems の実装を比較します。

第 3 章「Java サーブレットの移行」

この章では、WebLogic Server 7.0 から OracleAS に Java サーブレットを移行する際に必要な情報を説明します。ここでは、単純なサーブレット、WAR ファイルおよび展開 Web アプリケーションの移行を扱います。

第 4 章「JSP ページの移行」

この章では、WebLogic Server 7.0 から OracleAS に JavaServer Pages を移行する際に必要な情報を説明します。ここでは、単純な JSP ページ、JSP のカスタム・タグ・ライブラリおよび WebLogic のカスタム・タグの移行を扱います。

第 5 章「Enterprise JavaBeans コンポーネントの移行」

この章では、WebLogic Server 7.0 から OracleAS に Enterprise JavaBeans を移行する際に必要な情報を説明します。ここでは、ステートフル Session Bean およびステートレス Session Bean と、コンテナ管理の永続性または Bean 管理の永続性を持つ Entity Bean の移行を扱います。

第 6 章「JDBC の移行」

この章では、WebLogic Server 7.0 から OracleAS にデータベース・アクセス・コードを移行する際に必要な情報を説明します。ここでは、JDBC ドライバ、データソースおよび接続プーリングの移行を扱います。

付録 A「その他の機能比較」
この付録では、Oracle Application Server と WebLogic Server にある、他の機能をまとめています。

関連ドキュメント

リリース・ノート、インストール関連ドキュメント、ホワイト・ペーパーまたはその他の関連ドキュメントは、OTN-J（Oracle Technology Network Japan）から、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。登録は、次の Web サイトから無償で行えます。

`http://otn.oracle.co.jp/membership/`

すでに OTN-J のユーザー名およびパスワードを取得している場合は、次の URL で OTN-J の Web サイトのドキュメントのセクションに直接接続できます。

`http://otn.oracle.co.jp/document/`

詳細は、次の URL を参照してください。

- `http://ibm.com/`（WebLogic Server の詳細）
- `http://java.sun.com/`（J2EE の詳細）

表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。この項の内容は次のとおりです。

- 本文の表記規則
- コード例の表記規則
- Microsoft Windows オペレーティング・システム環境での表記規則

本文の表記規則

本文では、特定の項目が一目でわかるように、次の表記規則を使用します。次の表に、その規則と使用例を示します。

規則	意味	例
太字	太字は、本文中で定義されている用語および用語集に記載されている用語を示します。	この句を指定すると、**索引構成表**が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システム指定の要素を示します。このような要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドがあります。また、システム指定の列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみ、この句を指定できます。 BACKUP コマンドを使用して、データベースのバックアップを作成できます。 USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合せます。 DBMS_STATS.GENERATE_STATS プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびユーザーが指定する要素のサンプルを示します。このような要素には、コンピュータ名およびデータベース名、ネット・サービス名および接続識別子があります。また、ユーザーが指定するデータベース・オブジェクトとデータベース構造、列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニットおよびパラメータ値も含まれます。 **注意 :** プログラム要素には、大文字と小文字を組み合せて使用するものもあります。これらの要素は、記載されているとおりに入力してください。	sqlplus と入力して、SQL*Plus をオープンします。 パスワードは、orapwd ファイルで指定します。 /disk1/oracle/dbs ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを作成します。 hr.departments 表には、department_id、department_name および location_id 列があります。 QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。 oe ユーザーとして接続します。 JRepUtil クラスが次のメソッドを実装します。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。ここで、*old_release* とはアップグレード前にインストールしたリリースを示します。

コード例の表記規則

コード例は、SQL、PL/SQL、SQL*Plus または他のコマンドライン文の例です。次のように固定幅フォントで表示され、通常のテキストと区別されます。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例で使用される表記規則とその使用例を示します。

規則	意味	例
[ ]	大カッコは、カッコ内の項目を任意に選択することを表します。大カッコは、入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコは、カッコ内の項目のうち、1 つが必須であることを表します。中カッコは、入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の複数の選択項目の区切りに使用します。項目のうちの 1 つを入力します。縦線は、入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	水平の省略記号は、次のいずれかを示します。 ■ 例に直接関連しないコードの一部が省略されている。 ■ コードの一部を繰り返すことができる。	CREATE TABLE ... AS *subquery*; SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fsl/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fsl/dbs/tbs_09.dbf 9 rows selected.
その他の記号	大カッコ、中カッコ、縦線および省略記号以外の記号は、記載されているとおりに入力する必要があります。	acctbal NUMBER(11,2); acct CONSTANT NUMBER(4) := 3;
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。	CONNECT SYSTEM/*system_password* DB_NAME = *database_name*

規則	意味	例

大文字	大文字は、システム指定の要素を示します。これらの要素は、ユーザー定義の要素と区別するために大文字で示されます。大カッコ内にないかぎり、表示されているとおりの順序および綴りで入力します。ただし、大 / 小文字が区別されないため、小文字でも入力できます。	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

小文字	小文字は、ユーザー指定のプログラム要素を示します。たとえば、表名、列名またはファイル名などです。
	
	注意 : プログラム要素には、大文字と小文字を組み合せて使用するものもあります。これらの要素は、記載されているとおりに入力してください。
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Microsoft Windows オペレーティング・システム環境での表記規則

次の表に、Microsoft Windows オペレーティング・システム環境での表記規則とその使用例を示します。

規則	意味	例
ファイル名およびディレクトリ名	ファイル名およびディレクトリ名は大 / 小文字が区別されません。特殊文字の左山カッコ（<）、右山カッコ（>）、コロン（:）、二重引用符（"）、スラッシュ（/）、縦線（|）およびハイフン（-）は使用できません。円記号（¥）は、引用符で囲まれている場合でも、要素のセパレータとして処理されます。Windows では、ファイル名が ¥¥ で始まる場合、汎用命名規則が使用されていると解釈されます。	`c:¥winnt"¥"system32` は `C:¥WINNT¥SYSTEM32` と同じです。
Windows コマンド・プロンプト	Windows コマンド・プロンプトには、カレント・ディレクトリが表示されます。このマニュアルでは、コマンド・プロンプトと呼びます。コマンド・プロンプトのエスケープ文字はカレット（^）です。	`C:¥oracle¥oradata>`

規則	意味	例

特殊文字	Windows コマンド・プロンプトで二重引用符（"）のエスケープ文字として円記号（¥）が必要な場合があります。丸カッコおよび一重引用符（'）にはエスケープ文字は必要ありません。エスケープ文字および特殊文字の詳細は、Windows オペレーティング・システムのドキュメントを参照してください。	`C:¥>exp scott/tiger TABLES=emp QUERY=¥"WHERE job='SALESMAN' and sal<1600¥"`
		`C:¥>imp SYSTEM/`*`password`*
		`FROMUSER=scott TABLES=(emp, dept)`

HOME_NAME	Oracle ホームの名前を表します。ホーム名には、英数字で 16 文字まで使用できます。ホーム名に使用可能な特殊文字は、アンダースコアのみです。	`C:¥> net start Oracle`*`HOME_NAME`*`TNSListener`
ORACLE_HOME および *ORACLE_BASE*	Oracle8*i* より前のリリースでは、Oracle コンポーネントをインストールすると、すべてのサブディレクトリが最上位の *ORACLE_HOME* の直下に置かれました。*ORACLE_HOME* ディレクトリの名前は、デフォルトでは次のいずれかです。 ■ `C:¥orant`（Windows NT の場合） ■ `C:¥orawin98`（Windows 98 の場合） このリリースは、Optimal Flexible Architecture（OFA）のガイドラインに準拠しています。*ORACLE_HOME* ディレクトリ下に配置されないサブディレクトリもあります。最上位のディレクトリは *ORACLE_BASE* と呼ばれ、デフォルトでは `C:¥oracle` です。他の Oracle ソフトウェアがインストールされていないコンピュータに Oracle9*i* リリース 1（9.0.1）をインストールした場合、Oracle ホーム・ディレクトリは、デフォルトで `C:¥oracle¥ora90` に設定されます。Oracle ホーム・ディレクトリは、*ORACLE_BASE* の直下に配置されます。 このマニュアルに示すディレクトリ・パスの例は、すべて OFA の表記規則に準拠しています。	`%ORACLE_HOME%¥rdbms¥admin` ディレクトリへ移動します。

1

概要

この章では、WebLogic Server 7.0 から Oracle Application Server 10*g*（9.0.4）に J2EE Web アプリケーションを移行する際の問題点や、必要な作業の概要について説明します。

この章は、次の項で構成されています。

J2EE の概要

アプリケーション・サーバーの市場は急速に進歩しています。特に、この数年間で最も大きな進展は、Sun 社の Java 2 Platform, Enterprise Edition（J2EE）仕様が登場したことであり、これは複数ベンダー間での標準化を実現するものです。

J2EE プラットフォームおよびコンポーネントの仕様により、複数階層で Web ベースのエンタープライズ・アプリケーションを開発およびデプロイする標準プラットフォームなどが定義されます。

J2EE は、複数階層のコンピューティング・モデルに移行する際に企業が直面する問題の解決策となります。これにより、信頼性、スケーラビリティ、セキュリティ、アプリケーションのデプロイ、トランザクション処理、Web インタフェース設計、迅速なソフトウェア開発などの問題が解決できます。これは Java 2 Platform, Standard Edition（J2SE）に基づいて構築されたもので、複数階層のコンピューティングに、Sun 社の「Write Once, Run Anywhere」というパラダイムを実現します。

J2EE は、表 1-1 で説明するコンポーネントで構成されています。

表 1-1 J2EE の標準的なアーキテクチャ・コンポーネント

コンポーネント	説明
J2EE アプリケーション・モデル	複数階層の Thin クライアント・サービスを開発するためのアプリケーション・モデル
J2EE プラットフォーム	J2EE アプリケーションをホスティングするためのプラットフォーム
J2EE 互換性テスト・スイート	J2EE プラットフォームおよびコンポーネントの仕様で定められた要件を、J2EE プラットフォーム製品が満たしているかどうかを検証するための互換性テスト・スイート
J2EE リファレンス実装	J2EE プラットフォームのリファレンス実装

J2EE アプリケーション・モデルとは

J2EE アプリケーションとは、複数階層のアプリケーション・モデルです。アプリケーション・コンポーネントは、複数階層でコンテナによって管理されます。コンテナとは標準的なランタイム環境のことであり、ライフ・サイクル管理、デプロイ、セキュリティ・サービスなどのサービスを、アプリケーション・コンポーネントに提供するものです。このコンテナ・ベースのモデルにより、システム・インフラストラクチャからビジネス・ロジックが分離されます。

J2EE プラットフォームとは

J2EE プラットフォームはランタイム環境と標準サービス・セットで構成されており、このサービス・セットは、複数階層で Web ベースのエンタープライズ・アプリケーションの開発に必要な機能を実現するものです。

J2EE プラットフォームは、表 1-2 で説明するコンポーネントで構成されています。

表 1-2 J2EE プラットフォームのコンポーネント

コンポーネント	説明
J2EE ランタイム環境	
アプリケーション・コンポーネント	
アプリケーション・クライアント	デスクトップ・コンピュータで実行する Java プログラムで、通常は GUI に使用される
アプレット	通常は Web ブラウザで実行する Java プログラムのコンポーネント
サーブレットと JSP ページ	サーブレット : Web サーバーで実行する Java プログラムで、動的コンテンツの生成に使用される JSP ページ : Web ブラウザなどのクライアントに、動的コンテンツを返すのに使用するテクノロジ
Enterprise JavaBeans（EJB）	コンポーネント・ベースで分散型のコンピューティング用のアプリケーション・アーキテクチャ
コンテナ	ライフ・サイクル管理、デプロイ、セキュリティ・サービスなどのサービスを、アプリケーション・コンポーネントに提供するエンティティ
リソース・マネージャ・ドライバ	外部データソースに対するネットワーク接続を可能にするシステム・レベルのコンポーネント
データベース	ビジネス・データのストレージに使用される関連ファイルのセットで、JDBC API を介してアクセスできる
J2EE の標準サービス	
HTTP	Web サーバーとブラウザ間でメッセージを送受信する際に、インターネットが使用する標準プロトコル
HTTPS	Web サーバーとブラウザ間でメッセージをセキュアに送受信する際に、インターネットが使用するプロトコル
Java Transaction API（JTA）	アプリケーションとアプリケーション・サーバーが、トランザクションにアクセスできるようにする API

表 1-2 J2EE プラットフォームのコンポーネント（続き）

コンポーネント	説明
RMI-IIOP	RMI: Java オブジェクト間でのリモート通信を可能にするプロトコル IIOP: ブラウザとサーバーが、テキスト以外のものを交換することを可能にするプロトコル RMI-IIOP は、RMI のうち、CORBA IIOP プロトコルを使用するバージョン
JavaIDL	インタフェースを指定するための標準言語で、主に CORBA オブジェクト・インタフェース定義に使用される
JDBC	データベースと J2EE プラットフォームを接続する API
Java Message Service（JMS）	エンタープライズ・メッセージング・システムの使用を可能にする API
Java Naming and Directory Interface（JNDI）	ディレクトリ・サービスとネーミング・サービスを提供する API
JavaMail	電子メールの送受信を可能にする API
JavaBeans Activation Framework（JAF）	JavaMail API で必要とされる API

アプリケーション・サーバーとは

アプリケーション・サーバーとは、Web ベースのクライアント・プログラム、バックエンドのデータベース、レガシー・アプリケーションの間で実行するソフトウェアです。これにより、ビジネス・ロジックからシステムの複雑さを切り離しやすくなるため、開発者はビジネス上の問題解決に専念できます。アプリケーション・サーバーでは、系統立った効率的な方法でクライアント・プログラムが機能とリソースを共有できるため、プログラムのサイズと複雑さを軽減できます。

アプリケーション・サーバーには、利便性、柔軟性、スケーラビリティ、保守性、相互運用性の面で利点があります。

Oracle Application Server の概要

Oracle Application Server は、包括的で統合性の高いアプリケーション・サーバーであり、あらゆる E-Business を正常に実行するために必要なインフラストラクチャと機能が、すべて用意されています。開発チームはどこも似たような課題を抱えています。すなわち、経営上の調整や戦略的な意思決定にビジネス・インテリジェンスを提供する一方で、Web サイトや、あらゆるネットワークおよびあらゆるデバイスで高速実行できるアプリケーションを、迅速に配布することが必要です。Oracle Application Server をチームが使用すると、E-Business をめぐるこうした課題をすべて解決できます。

アプリケーション・サーバー市場では、Oracle Application Server に対する関心が高まり、複数階層で Web ベースのエンタープライズ・アプリケーションのデプロイに、このサーバーを採用する組織が増えています。

Web サイトとアプリケーションの開発、デプロイおよびセキュリティ保護を行う統合インフラストラクチャを実現できるのは、Oracle Application Server のみです。これによって、エンタープライズ Java アプリケーション開発用の完全な J2EE プラットフォームが実現します。開発者が Oracle Application Server を使用すると、Java、Perl、PL/SQL、XML、Forms など、あらゆる言語で Web アプリケーションを開発できます。また、Java、XML、SQL を統合した単一プラットフォームを介することにより、開発とデプロイにかかるコストが削減されます。

Oracle Application Server における J2EE サーバー実装は、Oracle Application Server Containers for J2EE（OC4J）と呼ばれます。OC4J は標準的な JDK で実行され、非常に軽量で、パフォーマンスとスケーラビリティが高く、デプロイと管理が簡単にできます。Oracle Application Server 10*g*（9.0.4）を使用すると、OC4J は J2EE 1.3 API をサポートします。

この移行ガイドは、WebLogic Server 7.0 から Oracle Application Server 10*g*（9.0.4）に J2EE アプリケーションを移行する際に、直面する可能性のある移行上の課題を理解しやすくすることを目的にしています。

注意： このドキュメントでは、バージョン番号なしで WebLogic Server という表現を使用した場合、それは WebLogic Server 7.0 を指しています。これ以外のバージョンの WebLogic Server を指す場合は、バージョン番号を具体的に明記します。

J2EE アプリケーションの移行上の課題

J2EE 規格への準拠度合いがまちまちであるため、アプリケーション・サーバー間でのアプリケーション移行がやっかいな作業になる可能性があります。アプリケーション・サーバー間で J2EE アプリケーションを移行する際の課題には、次のようなものがあります。

- 理論上は、J2EE 準拠のアプリケーション・サーバーであれば、どこにでも J2EE アプリケーションをデプロイできることになっていますが、実際にはデプロイできない場合もあります。
- 所定の J2EE アプリケーションの実装の詳細について、熟知していないことがあります。
- 「J2EE 準拠」という言葉の意味があいまいです（通常は、アプリケーション・サーバーに J2EE 準拠機能があるという意味であり、コード・レベルで J2EE 仕様と互換性があるという意味ではありません）。
- J2EE 規格に対してベンダーが用意した拡張機能のうち、使用中のものの数がデプロイ方法によって異なり、アプリケーション・サーバー間での Java コードの移植性が低下します。
- アプリケーション・サーバー間で、クラスタリング、ロード・バランシングおよびフェイルオーバーの実装に違いがあります。この違いはあちこちに記述されているため、移行プロセスにとって、いっそう大きな課題となっています。

このような課題により、移行作業はやっかいで不確実なものとなり、信頼性の高いプランニングやスケジューリングが難しくなります。この章では、WebLogic Server から Oracle Application Server に、アプリケーションを移行する際の課題を扱い、移行へのアプローチに、J2EE バージョン 1.3 仕様に基づいた解決策を用意します。

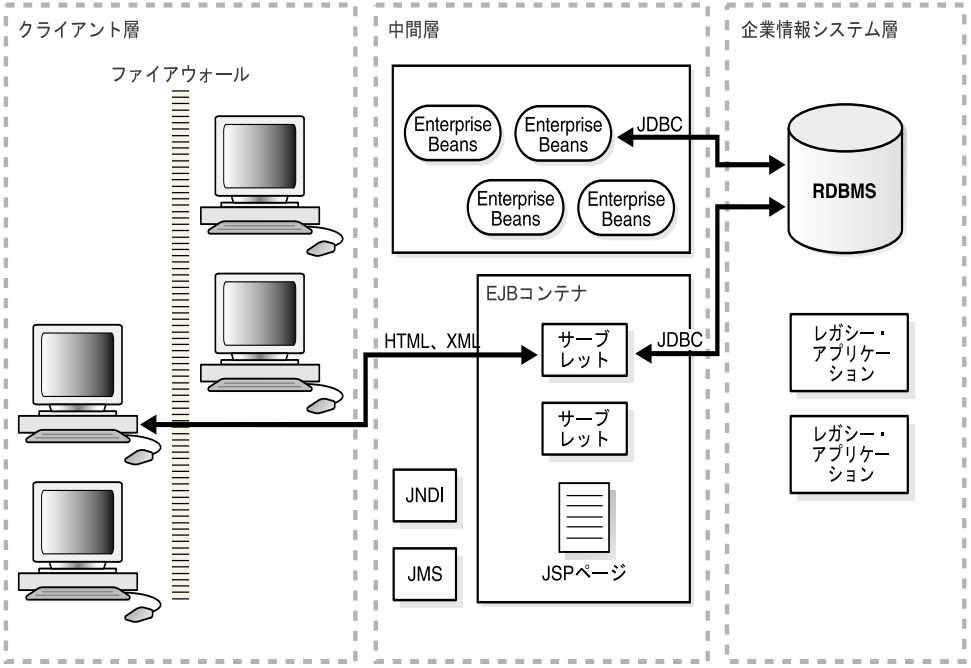
J2EE アプリケーション・アーキテクチャ

J2EE プラットフォームによって、複数階層で分散型のアプリケーション・モデルが実現します。J2EE コンポーネント・ベースの開発モデルで中心となるのが、コンテナという概念です。コンテナとは、標準化されたランタイム環境で、コンポーネントに特定のサービスを提供するものです。したがって、どの組織でも、特定の用途に開発された Enterprise JavaBeans（EJB）には、どのベンダーのどの J2EE プラットフォームでも使用できるトランザクションや EJB ライフ・サイクルなど、汎用的なサービスが考えられます。

コンテナによって、企業情報システムへの標準アクセスも実現します。たとえば、JDBC API を介した RDBMS アクセスなどが実現します。またコンテナにより、組立て時やデプロイ時のアプリケーションの動作を選択するメカニズムも用意されます。

図 1-1 に示すように、J2EE アプリケーション・アーキテクチャは、複数階層のアプリケーション・モデルです。中間層では、コンポーネントはコンテナによって管理されます。たとえば、サーブレットの動作は J2EE Web コンテナによって起動し、EJB のライフ・サイクルとトランザクションは EJB コンテナによって管理されます。このコンテナ・ベースのモデルにより、システム・インフラストラクチャからビジネス・ロジックが分離されます。

図 1-1 J2EE アーキテクチャ



移行上の問題点

移行作業量を計るうえで、移行対象のアプリケーション・コンポーネントを調べる際に、次の問題点を念頭に置くと役に立ちます。

- 移植性

  コードには、J2EE 仕様に対するベンダー固有の拡張機能への参照が埋め込まれているため、移植できない場合があります。コード変更の評価とプランニングは、移行作業の中でも重要な役割を果たす場合があります。

- 独自の拡張機能

  ベンダー固有の拡張機能が使用中の場合は、そうしたコンポーネントの移行は困難または実現不可能となります。J2EE 仕様の再設計作業全体は、このドキュメントでは扱っていません。ベンダー固有の拡張機能が使用中の場合は、そうした拡張機能を移行対象として洗い出すのでなく、再設計と再実装が必要になると考えられます。

- J2EE 仕様からの逸脱

  コンポーネントが J2EE 仕様に準拠しない部分が大きい場合は、Oracle Application Server への移行作業を判断するうえで、このガイドはあまり有用ではありません。コンポーネントの J2EE 仕様バージョンが（このガイドのベースとなっているバージョン）1.3 でない場合、仕様の実装の違いを解決することが必要になります。

移行アプローチ

この移行ガイド作成にあたっては、WebLogic Server から Oracle Application Server に Web アプリケーション・コンポーネントを移行するうえで、弊社が経験した内容を文書化することに努めました。WebLogic Server に同梱されたサンプルを選択し、WebLogic Server でテストしてから Oracle Application Server に移行しました。このサンプルを移行する際に直面した問題点を、このドキュメントのベースにしました。

移行作業

WebLogic Server から Oracle Application Server への移動は、相対的に簡単なプロセスです。独自の API を使用していない標準的な J2EE アプリケーションをデプロイする際には、コードを変更する必要がありません。必要なアクションは、構成とデプロイのみです。独自のユーティリティや API を使用していないアプリケーションは、簡単に移植できます。

移行ツール

Oracle JDeveloper Application Migration Assistant（AMA）は、Oracle プラットフォームにアプリケーションを移行するプロセスを簡略化するために、オラクル社が新たに開発したツールです。このツールは、コードのナビゲーションと進捗状況のレポートを実行することによって、WebLogic Server から Oracle Application Server 10*g* への移行作業を支援します。

この AMA ツールは、Oracle JDeveloper のプラグインとしてインストールされます。そして正規表現を使用して、アプリケーション・ファイル内のコードのうち、Oracle プラットフォームで機能するために変更が必要になる可能性のあるものを特定します。この正規表現は、検索ルール・ファイルと呼ばれる XML ファイルに入っています。AMA では、WebLogic Server アプリケーションを分析し、分析レポートを生成することができます。このレポートでは、プロジェクトの統計を集約し、レビュー項目間のナビゲーションを可能にし、移行に際して行った変更を追跡した包括的なステータスを提供します。AMA はカスタマイズ可能です。カスタマイズするには、拡張可能な API を指定して、特定のアプリケーション用に、追加の検索ルール・ファイルの作成や調整ができるようにします。

オラクル社では、AMA Search Rules Exchange（http://otn.oracle.com/tech/migration/ama/exchange/exchange.html）を介して、検索ルール・ファイルをいくつか提供しています。その 1 つとして、AMA Search Rules for BEA WebLogic Migrations というファイルがあります。このファイルに定義されているルールは、WebLogic 固有のコードのうち、Oracle Application Server への移行に際して、変更が必要になる可能性のあるものを特定するのに使用できます。たとえば、このツールでは、Jolt と BEA JCOM のどちらを使用しているかを判別できます。またアプリケーションの中で、Defweblogic 起動 / 停止、T3 サービス、WebLogic XA およびネイティブ WebLogic JDBC ドライバへの参照を検索することもできます。

このツールおよびそれに関する詳細をダウンロードするには、http://otn.oracle.com/tech/migration/ama にアクセスしてください。

このガイドの使用方法

このガイドでは、WebLogic Server から Oracle Application Server にコンポーネントを移行する作業の詳細を説明します。考えられるあらゆる構成に対する解決策を徹底的に記載したものとは言いませんが、前述の移行上の問題点のいくつかに対する解決策を示しています。ここで取り上げる問題点は、移行作業において、他の問題点とともに表面化してくるものです。このガイドに記載した内容は、WebLogic Server アプリケーションの評価、そして Oracle Application Server への移行のプランニングおよび実行の助けとなります。このガイドの内容は、次の高水準な作業を支援します。

- 前述の問題点に応じたコンポーネントの調査
- 移行対象の洗い出し
- 移行環境と移行ツールの準備
- 対象コンポーネントの移行とテスト

2

Oracle Application Server と WebLogic Server の比較

WebLogic Server と Oracle Application Server は、どちらも J2EE 1.3 の機能をサポートする J2EE サーバーですが、これらのアプリケーション・サーバーには、製品パッケージからランタイム・アーキテクチャに至るまで、本質的な違いがあります。この章ではこれらの違いについて説明します。構成は次のとおりです。

アプリケーション・サーバー製品

WebLogic Server は WebLogic Platform のコンポーネントであり、次に示す各種の WebLogic 製品で構成されています。Oracle Application Server も多くのコンポーネント製品で構成されています。これらの製品については後述します。

WebLogic

WebLogic Server は、次の製品構成で利用できます。

- WebLogic Server
- WebLogic Enterprise
- WebLogic Express

WebLogic Server

WebLogic Server は、J2EE アプリケーションの中核となるサービスとインフラストラクチャを提供します。また J2EE 1.3 の機能をサポートしています。これらの J2EE 1.3 の機能には、JSP 1.2、サーブレット 2.3、EJB 2.0、JCA 1.0 などがあります。

WebLogic Server では、SOAP、UDDI および WSDL を介した Web サービスとして、Java アプリケーションと Java コンポーネントをデプロイできます。BEA Tuxedo を介して、CORBA にも対応しています。

各 WebLogic Server は、HTTP 1.1 対応の HTTP リスナーをそれぞれ使用する Web サーバーとして構成できます。あるいは、Web サーバーとして Apache、Microsoft IIS および Netscape を使用することもできます。こうした Web サーバー構成では、サーブレットまたは JSP によって生成された動的コンテンツに加えて、静的 HTML コンテンツの要求も WebLogic Server で処理できます。

WebLogic Server ノードは、管理サーバーとしてデプロイできます。このノードは、WebLogic ドメイン内の他の WebLogic Server（管理対象サーバー）に管理サービスを提供します。WebLogic ドメインは、管理サーバーによって管理される一連の WebLogic Server と WebLogic Server のクラスタであり、この管理サーバーも含まれます。管理サーバーでは、Web ベースの GUI を使用してドメイン全体を管理します。各ドメインでは、WebLogic Server をクラスタリングすることもスタンドアロンで使用することもできます。クラスタリングの詳細は、「Oracle Application Server による高可用性とロード・バランシングのサポート」を参照してください。

注意： WebLogic Server がサポートしている J2EE 1.3 API のリストについては、この章の「J2EE サポートの比較」を参照してください。

WebLogic Enterprise

WebLogic Enterprise は、WebLogic Server と BEA Tuxedo で構成されています。BEA Tuxedo は分散トランザクション管理プラットフォームで、複数のデータベース間での分散トランザクションを可能にするものです。BEA Tuxedo は、WebLogic Server のコネクタ・アーキテクチャを介して WebLogic Server と統合されています。

WebLogic Enterprise は、Java、C++、C、COBOL など、複数のアプリケーション環境をサポートしています。WebLogic Enterprise は CORBA アプリケーションもサポートしており、異種アプリケーション環境へのシングル・サインオンが可能です。さらに、WebLogic Enterprise は BEA Tuxedo を介して業界標準の SNMP MIBS をサポートしており、サード・パーティのツールを使用して WebLogic Server を監視できます。

WebLogic Express

WebLogic Express は、WebLogic Server の軽量バージョンです。EJB と JMS をサポートしていないため、J2EE 準拠ではありません。また、JSP、サーブレット、JDBC および RMI をサポートしており、Web サーバーも搭載しています。そのため、WebLogic Express を使用すると、JDBC を使用した単純なデータベース・アクセスを備えた初歩的な Web アプリケーションを構築できます（2 フェーズ・トランザクションは未対応）。

Oracle Application Server

Oracle Application Server は、プラットフォームに依存しない J2EE アプリケーション・サーバーであり、インターネットおよびイントラネット用の、複数階層による Web 対応エンタープライズ・アプリケーションをホスティングできます。また、ブラウザやスタンドアロンのクライアントからアクセスできます。Oracle Application Server には、Oracle Application Server Containers for J2EE（OC4J）が組み込まれています。OC4J は、Java で記述された軽量でスケーラブルな J2EE コンテナであり、J2EE 1.3 による認定を取得しています。そのため、OC4J は次の J2EE 1.3 API をサポートしています。

- サーブレット 2.3
- JSP 1.2
- EJB 2.0
- JNDI 1.2
- JavaMail 1.1.2
- JAF 1.0
- JAXP 1.1
- JCA 1.0
- JAAS 1.0
- JMS 1.0

- JTA 1.0
- JDBC 2.0 Extension

Oracle Application Server は、大規模で可用性の高い、分散型 Java エンタープライズ・アプリケーションの実行を目的に設計されています。これらのアプリケーションには、インターネットの商用サイト、エンタープライズ・ポータル、大容量のトランザクション・アプリケーションなどがあります。Oracle Application Server は、現実のアプリケーション実装に不可欠な分野において、J2EE 規格の準拠にとどまらない大きな価値を追加し、次の要素を網羅した統合型ソリューションのスイート全体を実現します。

- Web サービス
- ビジネス・インテリジェンス
- 管理およびセキュリティ
- E-Business 統合
- ワイヤレス・クライアントのサポート
- エンタープライズ・ポータル
- パフォーマンス・キャッシュ

信頼性とスケーラビリティに優れたインフラストラクチャでこれらのソリューションを実装するには、クラスタリング・メカニズムと各種の高可用性ソリューションを使用して、Oracle Application Server を冗長なアーキテクチャにデプロイします。

Oracle Application Server のコンポーネントおよび特徴については、この章の「アーキテクチャの比較」と「Oracle Application Server による高可用性とロード・バランシングのサポート」で詳しく説明します。

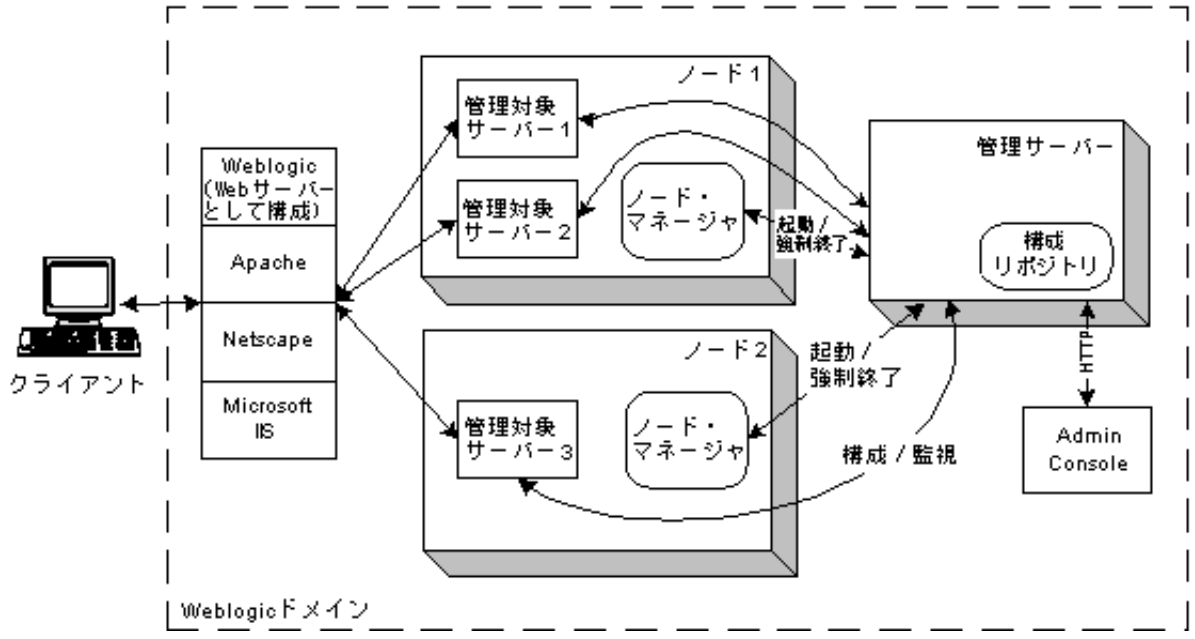
アーキテクチャの比較

この項では、WebLogic Server と Oracle Application Server のアーキテクチャ全体について、説明と比較を行います。

WebLogic Server

WebLogic Server には、固有のコンポーネントと概念がいくつかあります。各 WebLogic Server は、管理対象サーバーまたは管理サーバーとして構成およびデプロイできます。管理対象サーバーは、クライアントからリクエストを受信すると、それぞれにデプロイされているアプリケーション・ロジックをホスティングおよび実行します。管理サーバーは、管理対象サーバーを構成および監視します。図 2-1 は、WebLogic Server のコンポーネントとそこでの情報のやり取りを示しています。

図 2-1 WebLogic Server のコンポーネント



どのノードにも、複数の管理対象サーバーを配置できます。各管理対象サーバーは、J2EE コンテナ（Web および EJB）を実行する Java プロセス（JVM）です。Java プロセスでもある管理サーバーは、管理対象サーバーの起動時に、管理対象サーバーに構成情報を伝播する必要があります。構成情報は、管理サーバー・ノードのファイル・システムに格納されます。

管理サーバーは、個々の管理対象サーバーと WebLogic ドメイン全体に関する情報の監視とログ作成にも使用されます。WebLogic ドメインは、スタンドアロンの管理対象サーバー、管理対象サーバーのクラスタおよび 1 つの管理サーバーで構成できます。管理サーバーがオフラインになっても、管理対象サーバーはクライアント・リクエストを引き続き処理できます。ただし、新しい管理対象サーバーの起動時には、管理対象サーバーでは構成情報を使用できず、サーバー・クラスタでは監視サービスを利用できません。管理サーバーには、自動フェイルオーバーやレプリケーションの機能がありません。WebLogic ドメインの構成データは、手動でバックアップする必要があります。管理サーバーの機能を利用するには、コンソールの GUI（HTTP を介したリモート・アクセス）またはコマンドライン・ユーティリティを使用します。

管理サーバーで管理対象サーバーをリモート起動するには、管理対象サーバーが配置されている各ノードでノード・マネージャを実行する必要があります。このノード・マネージャは、UNIX デーモンまたは Windows サービスとして、バックグラウンドで実行される Java プログラムです。ノード・マネージャがあれば、管理対象サーバーが停止した場合や管理サーバーからのコマンドに応答しない場合、管理サーバーが管理対象サーバーを強制終了することもできます。

WebLogic Server は、Web サーバーとして実行するように設定することもできます。このモードでは、WebLogic Server は HTTP 1.1 をサポートし、XML 構成ファイルの設定に基づいて、管理対象サーバーに対するクライアント・リクエストを解決します。WebLogic Server ではなく、サード・パーティのプロキシ・プラグインを使用して HTTP リクエストを処理することもできます。サポート対象のプラグインは、Apache、Netscape および Microsoft IIS です。

Oracle Application Server のコンポーネントと概念

この項では、Oracle Application Server に固有のいくつかのコンポーネントと概念について説明します。ここではその概要を示します。

関連項目：

『Oracle Application Server 10*g* 概要』

『Oracle Application Server 10*g* 管理者ガイド』

『Oracle Application Server 10*g* 高可用性ガイド』

『Oracle9*i*AS Containers for J2EE ユーザーズ・ガイド』

Oracle Application Server インスタンス

OracleAS インスタンスは、Oracle Application Server のインストールのランタイムの出現です。Oracle Application Server インストールには、対応する Oracle ホームがあり、Oracle Application Server のファイルはここにインストールされます。各 Oracle Application Server インストールは、1 つの OracleAS インスタンスのみを実行時に提供できます。このインスタンスは、中間層インスタンスのこともあれば Infrastructure インスタンスのこともあります。1 つの物理ノードに Oracle ホームが複数ある場合もあります。したがって、Oracle Application Server インストールと OracleAS インスタンスが複数ある場合もあります。

各 OracleAS インスタンスは、複数の相互運用コンポーネントで構成されています。Oracle Application Server は、これらのコンポーネントにより、信頼性とスケーラビリティに優れた方法で、ユーザーのリクエストを処理することができます。これらのコンポーネントは次のとおりです。

- Oracle HTTP Server
- OC4J インスタンス
- Oracle Process Management Notification Server（OPMN）

- Distributed Configuration Management（DCM）
- Oracle Application Server Web Cache
- Oracle Enterprise Manager 10g Application Server Control コンソール
- Oracle Application Server Infrastructure

Oracle HTTP Server

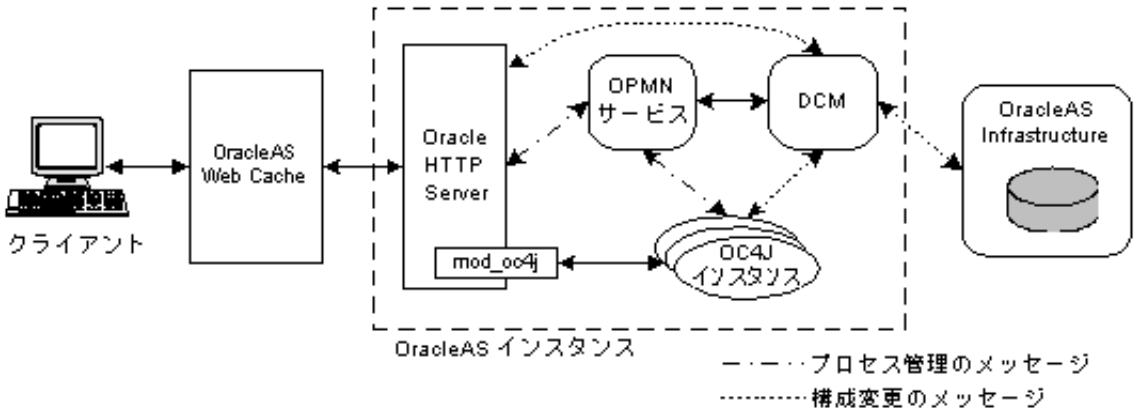
Oracle Application Server には、Oracle HTTP Server（Apache オープン・ソース・プロジェクトに準拠）と、個別の実行スレッドで実行される OC4J 付属のリスナーという、2 つのリスナーがあります。OracleAS インスタンスには、Oracle HTTP Server が 1 つずつ備えられています。

OC4J リスナーは、Oracle HTTP Server の mod_oc4j モジュールからのリクエストをリスニングし、適切な OC4J プロセスにリクエストを転送します。機能面で見ると、Oracle HTTP Server は OC4J のプロキシ・サーバーとして機能し、サーブレット・リクエストと JSP リクエストはすべて、OC4J プロセスにリダイレクトされます。

mod_oc4j は、Apache JServ Protocol バージョン 1.3（AJP 1.3）を介して OC4J リスナーと通信します。mod_oc4j は、Apache モジュールとして Oracle HTTP Server と連携して動作します。OC4J リスナーは、AJP 1.3 リクエストのほかに、HTTP リクエストと RMI リクエストも受け入れることができます。

次の図は、J2EE and Web Cache インストール・タイプの Oracle Application Server のシングル・インスタンスにおける、Oracle HTTP Server とその他の Oracle Application Server ランタイム・コンポーネントを示しています。

図 2-2 OracleAS インスタンスのコンポーネント



OC4J インスタンス

OC4J インスタンスは、Oracle Application Server で OC4J 実装を論理的にインスタンス化したものです。この実装は Java 2 Enterprise Edition（J2EE）の完全なセットであり、すべて Java で記述されています。OC4J インスタンスは、Oracle Application Server（JDK 1.3 対応）と一緒にインストールされる、標準の Java Development Kit（JDK）1.4 の Java Virtual Machine で実行されます。OC4J インスタンスでは、他の Java アプリケーション・サーバーと比べて、ディスク・フットプリントとメモリー・フットプリントが小さくなります。各 OC4J インスタンスは、複数の JVM プロセスで構成できます。JVM プロセスでは、それぞれのプロセスが複数の J2EE コンテナを実行できます。Oracle Enterprise Manager 10*g* Application Server Control コンソールの GUI を使用して、OC4J インスタンスごとに JVM プロセスの数を指定できます。

Oracle Application Server では、スケーラビリティと高可用性を実現するために、複数の OC4J インスタンスを Oracle Application Server Cluster のメンバーとしてクラスタリングできます。OC4J インスタンスをクラスタリングすると、OC4J インスタンスは一貫性のある構成を持ち、これらのインスタンス全体に同じアプリケーションがデプロイされます。クラスタリングの詳細は、後述の「Oracle Application Server による高可用性とロード・バランシングのサポート」を参照してください。

Oracle Process Management Notification Server（OPMN）

各 OracleAS インスタンスは、そのインスタンス内で監視機能とプロセス管理機能を実行する OPMN コンポーネントを備えています。このサービスでは、OracleAS インスタンスのコンポーネント間でメッセージがやり取りされ、コンポーネントの起動、停止検出およびリカバリができます。この通信は、同じ OracleAS Cluster に属する他の OracleAS インスタンスの、他の OPMN サービスにも及んでいます。そのため、クラスタ内の他のインスタンスは、同一クラスタ内にある他の OracleAS インスタンスで、有効な OC4J プロセスと Oracle HTTP Server プロセスを認識できます。

OPMN サービスはまた、Application Server Control コンソールとの通信およびインタフェースを有し、Oracle Application Server の監視、構成および管理を行う統合インタフェースを実現します。Oracle Application Server のコンポーネント、Oracle HTTP Server、OC4J インスタンスおよび Distributed Configuration Management（次項で説明）では、発行と引用（Publish and Subscribe）のメッセージング・メカニズムを使用して OPMN サーバーと通信します。フェイルオーバーと可用性のために、OPMN サーバーを実装するプロセスでは、OPMN プロセスの障害発生時に OPMN プロセスを再起動するシャドウ・プロセスが使用されます。

Distributed Configuration Management（DCM）

各 OracleAS インスタンスの様々なコンポーネントにおける構成の変更を管理および追跡するために、各 OracleAS インスタンスには DCM プロセスが配置され、これらの作業を実行します。OracleAS インスタンスのコンポーネントのいずれかの構成が変更されると、その変更内容が DCM に伝えられます。DCM はそれを受けて、変更内容を認識し、Infrastructure データベースの Oracle Application Server Metadata Repository に変更内容を記録します。このリポジトリには、接続されている OracleAS インスタンスすべての構成情報が格納されます。このように、同一の Oracle Application Server Metadata Repository に接続されている OracleAS インスタンスはすべて、同一の OracleAS Farm に属します。OracleAS インスタンスのいずれかで障害が発生すると、これらのインスタンスを再起動するために Metadata Repository から構成情報が取得されます。

各 DCM はまた、それぞれのインスタンスの OPMN と通信し、リポジトリ・データの変更に関する通知イベントを送信します。これにより、OPMN は Oracle Application Server のコンポーネントに対して、該当する調整を行うことができます。

Oracle Application Server Web Cache

Oracle Application Server に用意されているキャッシュ・ソリューションは、静的に生成された Web コンテンツと動的に生成された Web コンテンツの両方をキャッシュするという、独自の機能を備えています。Oracle Application Server Web Cache（OracleAS Web Cache）では、Oracle HTTP Server へのラウンドトリップ数を減らすことにより、Oracle Application Server がホスティングする高負荷の Web サイトのパフォーマンスとスケーラビリティが大幅に向上しています。さらに、OracleAS Web Cache では、一貫性のある予測可能な応答を実現する機能をいくつか利用できます。これらの機能には、ページ断片のキャッシュ、コンテンツの動的組立て、Web サーバーのロード・バランシング、OracleAS Web Cache クラスタリング、フェイルオーバーなどがあります。OracleAS Web Cache は、クラスタ内の OracleAS インスタンスのロード・バランサとして使用できます。OracleAS Web Cache 自体は、それぞれのクラスタ内にデプロイできます。詳細は、『Oracle Application Server Web Cache 管理者ガイド』を参照してください。

Oracle Enterprise Manager 10*g* Application Server Control コンソール

Oracle Enterprise Manager 10*g* Application Server Control コンソール（Application Server Control コンソール）では、Web ベースのインタフェースを利用して、Oracle Application Server のコンポーネントおよびアプリケーションを管理できます。Application Server Control コンソールを使用すると、次の作業を行えます。

- OracleAS のコンポーネント、OracleAS Middle-Tier および Infrastructure インスタンス、OracleAS Middle-Tier クラスタ、デプロイされた J2EE アプリケーションとそのコンポーネントの監視
- Oracle Application Server のコンポーネント、インスタンス、クラスタおよびデプロイされたアプリケーションの構成
- OracleAS のコンポーネント、インスタンス、クラスタおよびデプロイされたアプリケーションの操作

- OracleAS コンポーネントとデプロイされたアプリケーションのセキュリティ管理

Oracle Enterprise Manager とその 2 つのフレームワークの詳細は、『Oracle Enterprise Manager 概要』を参照してください。

> **関連項目：** Application Server Control コンソールに関する説明とその使用方法については、『Oracle Application Server 10*g* 管理者ガイド』を参照してください。

Oracle Application Server Infrastructure

Oracle Application Server では、完全に統合されたインフラストラクチャとフレームワークを使用して、エンタープライズ・アプリケーションを開発およびデプロイできます。Oracle Application Server Infrastructure インストール・タイプは、集中化された製品メタデータ、セキュリティおよび管理サービス、構成情報とデータ・リポジトリを、Oracle Application Server Middle-Tier に提供します。Middle-Tier で必要とされる Infrastructure のサービスを統合することにより、エンタープライズ・アプリケーションの開発に要する時間と手間を節約できます。その結果、これらのアプリケーションの開発とデプロイに要する総コストが削減され、デプロイされたアプリケーションの信頼性が向上します。

Oracle Application Server Infrastructure は、次の総合的なサービスを提供します。

- **製品メタデータ・サービス**

  Oracle Application Server Infrastructure には、Oracle Application Server Middle-Tier インスタンスに必要なアプリケーション・サーバーのメタデータがすべて格納されます。このデータは Oracle9*i* データベースに格納されるため、Oracle データベースの堅牢性を活かした、信頼性とスケーラビリティに優れ、管理が容易なメタデータ・リポジトリが実現します。

- **セキュリティ・サービス**

  セキュリティ・サービスによって、Oracle Application Server にデプロイされているすべてのアプリケーションで、一貫性のあるセキュリティ・モデルと識別情報管理が実現します。セキュリティ・サービスを利用すると、シングル・サインオンを使用した集中化された認証、Oracle Delegated Administration Services による Web ベースの管理、およびユーザー認証接続情報の集中的な格納を行うことができます。このサービスの基盤となるリポジトリとして、Oracle Internet Directory が使用されます。

- **管理サービス**

  このサービスは Distributed Configuration Management で使用され、Oracle Application Server Middle-Tier インスタンスと Oracle Application Server Infrastructure インスタンスを管理します。このサービスは、Middle-Tier のクラスタリング・サービスの管理にも使用されます。Application Server Control コンソールは、デプロイされた J2EE アプリケーションの管理を集中化して総管理コストを削減します。

OracleAS Infrastructure のコンポーネントで、これらのサービスを実装するものは次のとおりです。

- Oracle Application Server Metadata Repository
- Oracle Identity Management

Oracle Application Server Metadata Repository Oracle Application Server Metadata Repository は、Oracle9*i* Enterprise Edition データベース・サーバーです。Metadata Repository にはコンポーネント固有の情報が格納され、アプリケーション・デプロイの一部である Oracle Application Server Middle-Tier または Infrastructure のコンポーネントからアクセスできます。エンド・ユーザーやクライアント・アプリケーションは、このデータに直接アクセスしません。たとえば、中間層の Portal アプリケーションは、Portal ページ組立ての集合の一部として Portal メタデータにアクセスします。Order Management Demo for BC4J で使用されるデータなど、Oracle Application Server の多くのコンポーネントのデモ・データもメタデータに含まれます。

Oracle Application Server のメタデータ、カスタマ・データまたはアプリケーション・データは、Oracle Application Server Metadata Repository 内で共存できます。これらのデータにアクセスできるアプリケーションは、データによって異なります。

Oracle Application Server Metadata Repository には、3 つの主要なタイプのメタデータが格納され、それは「Oracle Application Server Infrastructure」の項に記載されている 3 つの主要な Infrastructure サービスに対応します。これらのメタデータ・タイプは次のとおりです。

- 製品メタデータ
- 識別情報管理メタデータ
- 管理メタデータ

表 2-1 は、Oracle Application Server のコンポーネントのうち、アプリケーションのデプロイ時に、これらのタイプのメタデータを格納および使用するものを示しています。

表 2-1 メタデータと Infrastructure コンポーネント

メタデータのタイプ	関係する Infrastructure コンポーネント
製品メタデータ（デモ・データを含む）	Oracle Application Server Metadata Repository
識別情報管理メタデータ	OracleAS Single Sign-On、Oracle Internet Directory、Oracle Application Server Certificate Authority
管理メタデータ	Distributed Configuration Management、Oracle Enterprise Manager

Oracle Application Server Metadata Repository（OracleAS Metadata Repository）は、J2EE and Web Cache インストール・タイプを使用するアプリケーション・デプロイ以外のアプリケーション・デプロイすべてに必要です。Oracle Application Server では、次の 3 つの中間層インストール・オプションを利用できます。

- **J2EE and Web Cache:** Oracle HTTP Server、Oracle Application Server Containers for J2EE（OC4J）、Oracle Application Server Web Cache（OracleAS Web Cache）、Web Services、Oracle Business Components for Java（BC4J）および Application Server Control コンソールをインストールします。
- **Portal and Wireless:** J2EE および OracleAS Web Cache のすべてのコンポーネント、UDDI、Oracle Application Server Portal（OracleAS Portal）、Oracle Application Server Syndication Services（OracleAS Syndication Services）、Oracle Ultra Search および Oracle Application Server Wireless（OracleAS Wireless）をインストールします。
- **Business Intelligence and Forms:** J2EE、OracleAS Web Cache、OracleAS Portal および Oracle Application Server Wireless のすべてのコンポーネント、Oracle Application Server Forms Services、Oracle Application Server Reports Services、Oracle Application Server Discoverer および Oracle Application Server Personalization をインストールします。

Oracle Application Server ProcessConnect、Oracle Application Server InterConnect、Oracle Workflow などの統合コンポーネントは、これらの中間層インストール・オプションの上にインストールされます。

Distributed Configuration Management（DCM）コンポーネントによって、Portal and Wireless と Business Intelligence and Forms の両方のインストール・オプションの中間層管理が可能になり、OracleAS Metadata Repository 内にメタデータを格納できます。J2EE and Web Cache インストール・タイプの場合、DCM ではファイル・ベースのリポジトリがデフォルトで使用されます。J2EE and Web Cache インストール・タイプを Infrastructure に関連付けると、ファイル・ベースのリポジトリが OracleAS Metadata Repository に移動します。

> **関連項目：** OracleAS インストールの詳細は、Oracle Application Server のインストレーション・ガイドを参照してください。

Oracle Identity Management Oracle Identity Management のコンポーネントにより、OracleAS のアプリケーションとエンティティのセキュリティ・ライフサイクルに対して、インフラストラクチャが提供されます。Oracle Identity Management は、次のコンポーネントで構成されています。

- Oracle Internet Directory

  Oracle Internet Directory は、Lightweight Directory Access Protocol（LDAP）バージョン 3 を使用したディレクトリ・サービスの、オラクル社による実装です。これは、Oracle9*i* データベースのアプリケーションとして実行され、このデータベースの持つ高いパフォーマンス、スケーラビリティおよび高可用性を活用します。

  Oracle Internet Directory では、OC4J、Oracle Application Server Portal、Oracle Application Server Wireless など、Oracle Application Server の残りのコンポーネントに対するユーザーの作成および管理に、集中型のリポジトリを使用します。ユーザーの認可および認証を集中管理することにより、Oracle Internet Directory でユーザーを中央で定義して、Oracle Application Server のすべてのコンポーネントで共有できます。

  Oracle Internet Directory には、Java ベースの管理ツール（Oracle Directory Manager）、信頼できるプロキシ・ベースの管理を実現する Web ベースの管理ツール（Oracle Delegated Administration Services）、および複数のコマンドライン・ツールが付属しています。Oracle Delegated Administration Services を使用すると、Oracle Internet Directory 管理者でない委任管理者によるエンド・ユーザーのプロビジョニングを、Oracle Application Server 環境で実行できます。さらに、エンド・ユーザーはそれぞれの属性を変更できます。

  Oracle Internet Directory では、ユーザーおよびグループ・イベントに関するデータを Oracle Application Server のコンポーネントで同期化し、ローカル・アプリケーション・インスタンスに格納されているユーザー情報を、各コンポーネントで更新することもできます。

  関連項目：『Oracle Internet Directory 管理者ガイド』

- OracleAS Single Sign-On

  OracleAS Single Sign-On は複数の部分に分かれた環境で、1 回のユーザー認証で複数のパートナ・アプリケーションにアクセス可能な、中間層機能とデータベース機能で構成されています。パートナ・アプリケーションを開発するには、SSOSDK を使用するか、Apache の `mod_osso` モジュールを使用します。このモジュールを使用すると、Apache（ひいては URL）をパートナ・アプリケーションにすることができます。

  OracleAS Single Sign-On は Oracle Internet Directory と完全に統合され、ユーザー情報を格納します。Single Sign-On は Oracle Internet Directory を通じて、LDAP ベースのユーザーおよびパスワード管理をサポートしています。

  OracleAS Single Sign-On は公開鍵インフラストラクチャ（PKI）によるクライアント認証をサポートしているため、様々な Web アプリケーションに対して PKI 認証が可能です。さらに、X.509 によるデジタル・クライアント証明書や Kerberos によるセキュリティ・チケットを使用したユーザー認証もサポートしています。

API を使用すると、Netegrity Site Minder などサード・パーティの認証方式と OracleAS Single Sign-On を統合できます。

関連項目：『Oracle Application Server Single Sign-On 管理者ガイド』

- OracleAS Certificate Authority

  OracleAS Certificate Authority（OCA）は、オラクル社の公開鍵インフラストラクチャ（PKI）製品のコンポーネントです。OCA を使用すると、X.509v3 によるデジタル証明書を作成および管理して、オラクル社またはサード・パーティのソフトウェアで使用するようにできます。OCA は業界標準規格に完全に準拠し、OracleAS Single Sign-On および Oracle Internet Directory と完全に統合されています。OracleAS Certificate Authority では、Web ベースの証明書管理と XML ベースの構成が可能です。OCA には、Oracle9*i* プラットフォームの Identity Management インフラストラクチャ、高可用性およびスケーラビリティが活用されています。

  関連項目：『Oracle Application Server Certificate Authority 管理者ガイド』

高可用性とロード・バランシング

この項では高可用性とロード・バランシングについて説明し、アプリケーション・サーバーの操作におけるその重要性について示します。ここでは、WebLogic Server と Oracle Application Server それぞれの方法論を比較します。

WebLogic Server による高可用性とロード・バランシングのサポート

1 つ以上の WebLogic Server を、クラスタとしてグループ化できます。クラスタ全体のデプロイによって、通常、クラスタ内のすべてのサーバーにアプリケーションをデプロイすることによって、クラスタ全体でクライアント・リクエストをロード・バランシングし、アプリケーションにフェイルオーバー機能を持たせることができます。WebLogic クラスタでクラスタリングの恩恵を受けるエンティティは、HTTP セッション状態と EJB および RMI オブジェクトです。WebLogic では、複数のロード・バランシング・アルゴリズムが使用されます。これらのアルゴリズムには、ラウンドロビン、重みベースおよびパラメータ・ベースがあります。

HTTP セッション状態のロード・バランシングとフェイルオーバー（サーブレットのクラスタリング）

WebLogic クラスタにリクエストを送るクライアントは、クラスタ内のサーバー全体でリクエストをロード・バランシングすることができます。ロード・バランシングを実現するには、WebLogic プロキシ・プラグイン搭載でインストールした Web サーバー、またはハードウェアのロード・バランサを使用する必要があります。WebLogic プロキシ・プラグインでは、リクエストの負荷分散にラウンドロビンのロード・バランシング方式が使用されます。ハードウェアのロード・バランサを使用すると、ハードウェアに組み込まれた方式でクラスタをロード・バランシングできます。

WebLogic Server では、クライアントの HTTP セッション状態をレプリケートすることにより、サーブレットと JSP のフェイルオーバーを実現しています。WebLogic Server は、サーブレットまたは JSP の一番最初のリクエストを受信すると、サーブレットのセッション状態を別のサーバーにレプリケートします。レプリケートされたセッション状態は、レプリケート元と常に同じ状態を保ちます。WebLogic プロキシ・プラグインは、Cookie または URL 書換えによって、これらの 2 つのサーバーの名前をクライアントに返します。元のセッション状態をホスティングするサーバーに障害が発生すると、WebLogic プロキシ・プラグインは Cookie または URL の情報を使用して、セッション状態がレプリケートされているサーバーにクライアントをリダイレクトします。クラスタには常に、アクティブな各セッション状態のオリジナルとレプリカが保持されます。この場合には、セッション状態はメモリーにレプリケートされます。WebLogic Server では、ファイル・システムまたはデータベースへの、JDBC を介したレプリケーションもサポートしていますが、これらのレプリケーション方式では自動フェイルオーバーは実行されません。

EJB および RMI オブジェクトのロード・バランシングとフェイルオーバー

WebLogic Server では、クラスタ対応の JNDI サービスとクライアント・スタブを使用することにより、EJB および RMI オブジェクトでロード・バランシングとフェイルオーバーを利用できます。

クラスタ内の各 WebLogic Server には、ローカルの JNDI ツリーが保持されます。このツリーには、ローカル・サーバーおよびクラスタ（クラスタリング可能なオブジェクトの場合）にデプロイされたオブジェクトに関する情報が表示されます。クラスタリング可能なオブジェクトを複数のサーバーにデプロイすると、サーバーにそのオブジェクトが存在することが、それぞれの JNDI ツリーに示されます。クラスタリング可能なオブジェクトをサーバーにデプロイすると、そのサーバーはクラスタ内の他のサーバーに、この新しいデプロイをマルチキャストで通知します。これに応じて、他のサーバーの JNDI ツリーが更新されます。さらに、オブジェクトがデプロイされているサーバーは、他のサーバーにオブジェクトのスタブを送信します。

クラスタリング可能なオブジェクトをクライアントが JNDI サービスで参照すると、リクエストを処理するサーバーが、そのオブジェクトのスタブをクライアントに返します。このスタブには、オブジェクトが実際にデプロイされているサーバーはどれかという情報が格納されています。スタブには、オブジェクトへのメソッド・コールを分散させるロード・バランシング・ロジックも組み込まれています。利用可能なロード・バランシングのアルゴリズムには、ラウンドロビン、重みベース、ランダムおよびパラメータ・ベースがあります。クライアントには、クラスタは意識されません。JNDI ルックアップとロード・バランシングの実行時、クラスタリングされたオブジェクトに対して、サーバー側でこれらの処理が行われていることが、クライアントには認識されません。

クラスタリングされたオブジェクトが、ステートフル Session EJB のようにステートフルな場合、このオブジェクトの状態は 2 番目のサーバーにレプリケートされます。このレプリケーションは、HTTP セッション状態の場合と類似した方法で行われます。クライアントの一番最初のリクエスト処理に選ばれたサーバーが、オブジェクトの状態を別のサーバーにレプリケートします。このレプリケーションに伴って、クライアント・スタブが更新されます。最初のサーバーで障害が発生すると、スタブはメソッドの起動時に例外を受信します。続いてスタブは、オブジェクト状態がレプリケートされているサーバーに、この起動をリダイレクトします。このサーバーは、状態がレプリケートされているオブジェクトをインスタンス化し、メソッドの起動を実行します。さらに、元のサーバーが停止しているため、このサーバーは状態のレプリケート先として別のサーバーを選択します。ステートフル・オブジェクトのフェイルオーバーは、このように実行されます。

ステートレス・オブジェクトのフェイルオーバーは、状態のレプリケーションが不要なため、さらに簡単です。クライアント・スタブは、サーバーに障害が発生したことを知らせる例外を受信すると、コールされたオブジェクトの別インスタンスをホスティングしているサーバーを新たに選択し、メソッドの起動をそこにリダイレクトするだけです。

Oracle Application Server による高可用性とロード・バランシングのサポート

Oracle Application Server では、高可用性とロード・バランシングの方式がいくつか設計に盛り込まれています。これらの方式により、Infrastructure および中間層のレベルで、フェイルオーバーとスケーラビリティが実現します。フェイルオーバーでは、OracleAS の類似コンポーネントのクラスタを作成できます。これらのクラスタでは、類似コンポーネントに冗長性が確保されます。

この項では、Oracle Application Server の該当コンポーネントにおけるクラスタリングとロード・バランシングの概念および機能について説明します。

関連項目：『Oracle Application Server 10*g* 高可用性ガイド』

Oracle Application Server インスタンス

Oracle Application Server アーキテクチャは、中間層における高可用性をサポートしています。これにより、デプロイされたアプリケーションの計画外停止の大部分を回避できます。この項では、Oracle Application Server インスタンスのアーキテクチャの概要を説明し、中間層の高可用性機能をいくつか示します。

それぞれの Oracle Application Server インスタンスでは、インスタンス内およびそのインスタンスが属するクラスタで、次の機能により高可用性が実現します。

- プロセスの監視 : Oracle Process Management Notification Server（OPMN）システムによって、監視対象のプロセスに障害が検出された場合に、プロセスの停止検出と再起動を行います。

- 構成のクローニング : 構成情報に Oracle Application Server Metadata Repository を使用する Distributed Configuration Management 機能によって、Oracle Application Server インスタンスおよびクラスタ・メンバーの Oracle Application Server インスタンスに、分散および管理された構成が用意されます。
- データのレプリケーション : Web アプリケーション・レベルでステートフル・セッションをレプリケートする OC4J アイランドを備えた OC4J インスタンス、および EJB セッションを使用することにより、Oracle Application Server インスタンス内のプロセス全体、および Oracle Application Server Cluster の使用時には、複数の Oracle Application Server インスタンス全体（ホストが異なる場合も含む）で、データがレプリケートされます。これにより、Oracle Application Server インスタンス内のプロセスが使用不能または停止になっても、ステートフル・セッションをベースにしたアプリケーションはそのまま利用できます。
- スマートなルーティング : Oracle Application Server Web Cache と Oracle HTTP Server（mod_oc4j）は、着信リクエストに対して構成可能でインテリジェントなルーティングを提供します。リクエストのルーティング先は、Oracle Process Management Notification Server（OPMN）システムとの通信を通じて、mod_oc4J が動作中と判断したプロセスとコンポーネントに限定されます。

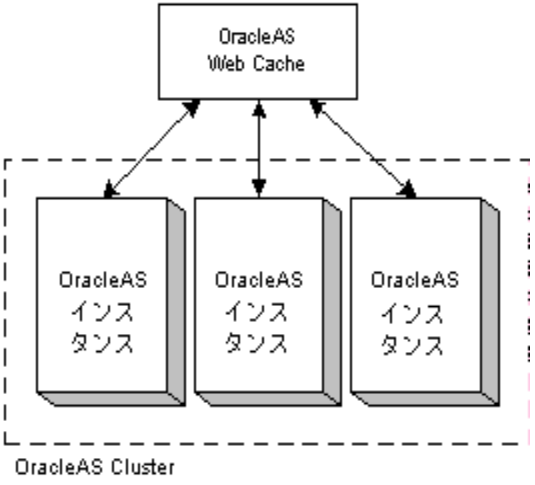
Oracle Application Server Cluster（中間層）

Oracle Application Server Cluster（OracleAS Cluster）は、1 つ以上の OracleAS インスタンスで構成されます（図 2-3 を参照）。クラスタ内の OracleAS インスタンスの構成はすべて同じです。クラスタのメンバーになる最初の OracleAS インスタンスは、2 番目以降のインスタンスがクラスタに加わるたびに、これらのインスタンスにその構成をレプリケートします。構成に加えて、デプロイされた OC4J アプリケーションも新しいインスタンスにレプリケートされます。レプリケートされた構成およびアプリケーションの情報は、クラスタが使用している OracleAS Metadata Repository から取得されます。

各クラスタ内には、OracleAS インスタンスのロード・バランシングまたはフェイルオーバーを実行するメカニズムはありません。つまり、クラスタには、インスタンス内の Oracle HTTP Server コンポーネントへのリクエストに対して、ロード・バランシングやフェイルオーバーを実行する内部メカニズムがありません。OracleAS Web Cache などの単独のロード・バランサまたはハードウェアのロード・バランシング製品を使用すると、クラスタ内の OracleAS インスタンスのロード・バランシング、およびクラスタ内の Oracle HTTP Server インスタンスのフェイルオーバーを行うことができます。

複数の OracleAS Cluster とスタンドアロンの OracleAS インスタンスをさらにグループ化すると、OracleAS Farm を作成できます。このファーム内のクラスタとインスタンスは、同じ OracleAS Metadata Repository を共有します。OracleAS Farm の詳細は、『Oracle Application Server 10*g* 管理者ガイド』を参照してください。

図 2-3 ロード・バランシングに OracleAS Web Cache を使用した OracleAS Cluster

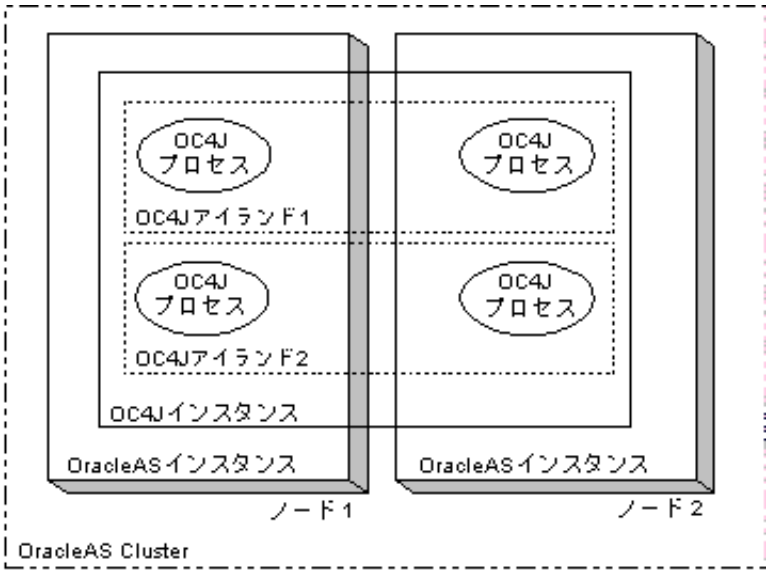


OC4J アイランド

Oracle Application Server のクラスタリング・テクノロジの重要な機能に、マルチキャスト・トラフィックの削減があります。クラスタ内のあらゆるサーバーでセッション状態を共有している場合は、クラスタ内のすべてのノードにセッション状態をレプリケートするために、多くの CPU サイクルがオーバーヘッドとして消費されます。Oracle Application Server では、この問題を解決するために、OC4J アイランドという概念を導入しました。OC4J アイランドでは、OracleAS Cluster 内の OC4J プロセス（JVM）をアイランドにサブグループ化できます。アプリケーションのセッション状態のレプリケーション先は、OracleAS Cluster 内のすべての OC4J プロセスではなく、同じアイランドに属する OC4J プロセスに限定されます。したがって、状態のレプリケート先プロセスの数が減ります。通常、OC4J アイランドは複数の物理ノードにわたって構成されます。したがって、1 つのノードが停止した場合でも、アプリケーション状態のフェイルオーバーが可能です。

2 つのノードで 4 つの OC4J プロセス（ノードあたり 2 プロセス）を実行している OracleAS Cluster を想定します（図 2-4 を参照）。アプリケーションの状態が変わると（変更は同じクライアントからリクエストを受信するたびにも発生）、各プロセスのアプリケーション状態を更新するために、4 つのプロセス全部にマルチキャスト・メッセージが送信されます。これらの 4 つのプロセスが、2 つのノードにわたって、2 プロセスで構成される 2 つのアイランドに分割されている場合、アプリケーション状態のレプリケーションは、同じアイランド内のプロセス間で実行するだけで済みます。マルチキャスト・メッセージは、そのアイランド内の 2 つのプロセス間でのみ必要となり、4 つのプロセス間では必要とされないため、レプリケーションのオーバーヘッドが半分に減少します。その結果、ネットワーク・トラフィックと CPU サイクルが減少します。

図 2-4 OC4J アイランド



OC4J アイランドの構成時には、各アイランドに属する OC4J プロセスの数をノードごとに指定できます。これにより、各ノードのハードウェアとオペレーティング・システムの性能に応じて、プロセス数を増減できます。OracleAS Cluster と OC4J アイランドの構成方法の詳細は、『Oracle Application Server 10*g* 高可用性ガイド』を参照してください。

EJB クラスタリングを使用したステートフル Session EJB の高可用性

OC4J では、ステートフル Session EJB を構成して、アプリケーション・サーバー・インスタンス内または OracleAS Cluster 全体で実行している複数の OC4J プロセスに、状態をレプリケートできます。この EJB レプリケーション構成では、複数の OC4J プロセスを使用して同じステートフル Session EJB のインスタンスを実行することにより、ステートフル Session EJB に高可用性が実現します。

注意： 高可用性のための EJB レプリケーション（EJB クラスタ）の使用は、OracleAS Cluster に依存しません。OracleAS Cluster のメンバーかどうかに関係なく、各ノードにインストールされている複数のアプリケーション・サーバー・インスタンスを対象に含めることができます。

EJB クラスタは、ステートフル Session EJB の高可用性を実現します。EJB クラスタでは、同じマルチキャスト・アドレスで通信する複数の OC4J プロセスで、これらの EJB のフェイルオーバーを実現できます。したがって、ステートフル Session EJB でレプリケーションを使用すると、プロセスおよびノードの障害が防止され、Oracle Application Server で実行しているステートフル Session EJB の高可用性が実現します。

関連項目：

- 『Oracle Application Server 10*g* 高可用性ガイド』
- 『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』
- 『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』

JNDI ネームスペースのレプリケーション EJB のクラスタリングが有効なときは、JNDI ネームスペースのレプリケーションも、OracleAS Cluster 内の OC4J インスタンスの間で有効になります。ある OC4J インスタンスの JNDI ネームスペースに新しくバインドすると、その OracleAS Cluster 内のその他の OC4J インスタンスに伝播されます。再バインドとアンバインドはレプリケートされません。

このレプリケーションは、OC4J アイランドの有効範囲外で行われます。言い換えれば、1 つの OC4J インスタンスに入っている複数のアイランドでは、レプリケートされた同じ JNDI ネームスペースを参照できます。

関連項目：『Oracle Application Server Containers for J2EE サービス・ガイド』

Java Object Cache

Oracle Application Server Java Object Cache に用意されている分散キャッシュは、OC4J にデプロイされたアプリケーションの高可用性ソリューションになります。Java Object Cache は Java オブジェクトのインプロセス・キャッシュで、あらゆる Java プラットフォームのあらゆる Java アプリケーションで利用できます。これにより、複数のリクエストやユーザー間でのオブジェクトの共有や、オブジェクトのライフ・サイクルの複数プロセスにわたる調整をアプリケーションで行うことができます。

Java Object Cache では、異なる OC4J アイランド、アプリケーション・サーバー・インスタンスまたは Oracle Application Server Cluster に属する OC4J プロセス間でも、データのレプリケーションが可能です。

Java Object Cache を使用すると、共有 Java オブジェクトがローカルにキャッシュされるため、オブジェクトをどのアプリケーションで生成するかに関係なく、パフォーマンスが向上します。また可用性も向上します。オブジェクトのソースが使用不能になっても、ローカルにキャッシュされたバージョンを引き続き使用できるからです。

Oracle Application Server Web Cache クラスタ

複数の OracleAS Web Cache インスタンスをクラスタリングして、1 つの論理キャッシュを作成できます。この論理キャッシュは、複数のノードに物理的に分散できます。1 つのノードで障害が発生した場合、そのノードが処理していたリクエストは、同一クラスタ内の残りのノードで実行できます。障害の検出は、そのクラスタで、障害が発生したメンバーのキャッシュ可能コンテンツの所有権を引き継ぐ残りのノードによって行われます。ハードウェアのロード・バランシング機器など、OracleAS Web Cache クラスタの前に設置されたロード・バランシング・メカニズムにより、稼動中の OracleAS Web Cache ノードにリクエストがリダイレクトされます。

OracleAS Web Cache クラスタによって、OracleAS インスタンスの可用性も向上します。OracleAS インスタンスの前で静的コンテンツおよび動的コンテンツをキャッシュすることにより、OracleAS Web Cache でリクエストを処理できます。これにより、OracleAS インスタンス（特に Oracle HTTP Server）がリクエストを実行する必要性が減少します。OracleAS インスタンスにかかる負荷とストレスが減少するため、インスタンス内のコンポーネントの可用性が向上します。

Oracle Application Server Web Cache は、Oracle HTTP Server に対して、ステートレスまたはステートフルなロード・バランシングの役割を果たすこともできます。ロード・バランシングは、各 Oracle HTTP Server の使用可能容量の割合、つまり、各 Oracle HTTP Server の加重使用可能容量に基づいて実行されます。加重使用可能容量が複数の Oracle HTTP Server で等しい場合、OracleAS Web Cache ではラウンドロビンによる負荷分散が行われます。加重使用可能容量の計算式については、『Oracle Application Server Web Cache 管理者ガイド』を参照してください。

Oracle HTTP Server で障害が発生すると、OracleAS Web Cache は残りの Oracle HTTP Server に負荷を再分散し、障害の発生したサーバーがオンラインに戻るまで、このサーバーを断続的にポーリングします。その後、OracleAS Web Cache は、有効範囲内の回復した Oracle HTTP Server で負荷分散を再計算します。

関連項目：『Oracle Application Server Web Cache 管理者ガイド』

OracleAS Infrastructure の高可用性ソリューション

OracleAS Infrastructure には、高可用性を実現するために各種のソリューションがあります。これらのソリューションでは、サイト内フェイルオーバーを使用できます。その内容は次のとおりです。

Oracle Application Server Cold Failover Clusters このコールド・フェイルオーバー・クラスタ・ソリューションでは、同一の構成を備えた 2 ノードのハードウェア・クラスタを使用します。2 つのノードは、一方がアクティブでもう一方はパッシブです。両ノード間にはハードウェア・インターコネクトが存在し、クラスタリング機能を備えたオペレーティング・システムで実行されます。これらの 2 つのノードは、共通の共有記憶域にアクセスします。さらに、2 つのノードで 1 つの論理 IP アドレスが共有されます。各ノードには、一意の物理 IP アドレスも存在します。ただし、中間層で参照可能なのは 1 つの論理 IP アドレスのみであり、これはコールド・フェイルオーバー・クラスタの Infrastructure へのアクセスに使用されます。

OracleAS Infrastructure のインストール中に、その Oracle ホームが、データベース・ファイルと一緒に共有記憶域にインストールされます。操作中は、常に 1 つのノードしか共有記憶域にマウントされません。アクティブ・ノードで障害が発生すると、パッシブ・ノードのクラスタリング・ソフトウェアが障害を検出して、論理 IP アドレスを引き継ぎます。パッシブ・ノードがアクティブ・ノードになり、共有記憶域がマウントされ、中間層からのリクエストが処理されます。

このコールド・フェイルオーバー・クラスタ・ノードは、中間層付きでインストールすることもできます。この場合のノードは、中間層ではアクティブ / アクティブ、Infrastructure ではアクティブ / パッシブになります。

関連項目：『Oracle Application Server 10*g* 高可用性ガイド』

Oracle Application Server Active Clusters コールド・フェイルオーバー・クラスタがアクティブ / パッシブの可用性構成を Infrastructure に提供するのに対し、Oracle Application Server Active Clusters（OracleAS Active Clusters）ソリューションは、アクティブ / アクティブの可用性構成を提供します。OracleAS Active Clusters ソリューションは、Oracle9*i* Real Application Clusters のテクノロジに基づいています。このソリューションでは、クラスタ内のノードを 3 つ以上アクティブにできます。各ノードで使用される基本ハードウェアにも、ハードウェア・クラスタのテクノロジが活用されています。ただし、IP アドレスの引継ぎメカニズムは使用されません。そのかわり、OracleAS Active Clusters ノードの前にハードウェアのロード・バランサ機器が構成され、これらのノードへのリクエストがロード・バランシングされます。このロード・バランサには論理 IP の名前とアドレスがあり、中間層が Infrastructure にアクセスするときに使用されます。Oracle Net 接続では、クラスタ内のノードのアドレス・リストを使用することにより、このハードウェアのロード・バランサを迂回します。ノードに障害が発生した場合、ハードウェアのロード・バランサ機器と Oracle Net はともに、アクティブ・ノードへのリクエストのフェイルオーバーを管理します。

関連項目：『Oracle Application Server 10*g* 高可用性ガイド』

J2EE サポートの比較

この項では、WebLogic Server 7.0 と Oracle Application Server 10*g*（9.0.4）の、J2EE 仕様に対するサポート・レベルの相違点の概要について説明します。

Oracle Application Server OC4J は J2EE 1.3 の認定を受けており、Sun 社の Compatibility Test Suite（CTS）に合格しています。CTS にはテスト項目が 5,000 以上あり、アプリケーションの移植性と J2EE 実装の総合的な質の評価を目的としています。一方、WebLogic Server も J2EE 1.3 の認定を受けています。

表 2-2 は、J2EE テクノロジと、それに対して Oracle Application Server と WebLogic Server が提供するサポート・レベルのリストです。

表 2-2 J2EE サポート

J2EE テクノロジ	WebLogic Server 7.0 がサポートするバージョン	Oracle Application Server 10*g*（9.0.4）がサポートするバージョン
JDK	1.3	1.4 と 1.3
サーブレット	2.3	2.3
JSP	1.2	1.2
EJB	2.0	2.0
JDBC	2.0	2.0 Extension
JNDI	1.2	1.2
JTA	1.0.1	1.0.1
JMS	1.0.2	1.0.2
JavaMail	1.1	1.1.2
JAF	なし	1.0.1
JAXP	1.1	1.1
JCA	1.0	1.0
JAAS	1.0	1.0

注意： Oracle Application Server OC4J のインストールでは、JDK 1.4.1 が組み込まれます。ただし、Oracle Application Server のこのバージョン（10*g*（9.0.4））では、JDK 1.3.x で OC4J を使用することもできます。

Oracle Application Server では、これらの規格へのサポートに加えて、現実の J2EE アプリケーションの構築用に、考え抜いた統合型アーキテクチャを用意しています。これらのアーキテクチャには、EJB 用の JAR ファイル、サーブレットおよび JSP 用の Web Archive（WAR）、アプリケーション用の Enterprise Archives（EAR）など、デプロイ用の標準アーカイブの実装があります。これにより、業界標準規格に準拠した他のアプリケーション・サーバーとの円滑な相互運用が実現します。

Java の開発ツールとデプロイメント・ツール

この項では、WebLogic Platform と Oracle Application Server に用意されている Java ツールを比較します。

WebLogic の開発ツールとデプロイメント・ツール

ここでは、WebLogic の開発環境と管理コンソールについて説明します。

WebLogic Server Workshop

WebLogic Workshop はビジュアルな開発環境で、Java や XML を使用した Web サービスの構築とデプロイに使用できます。WebLogic Workshop では、フレームワークと一連のコントロールを使用して、EJB、データベース、JMS トピック、JMS キューなどの Web サービスや Web アプリケーションと対話します。これらのコントロールには、Java Web サービス（JWS）のファイル定義以外に、WebLogic Platform 独自のものもあります。JWS ファイルには、WebLogic Server での Web サービス実装に使用するロジックが組み込まれています。ただし、JWS は、J2EE サービスや Web サービスの標準規格ではありません。また、他のアプリケーション・サービスには移植できません。

WebLogic Server Administration Console

WebLogic Server の管理コンソールでは、GUI を使用して WebLogic Server ドメインを管理します。WebLogic Server ドメインは、1 つ以上の WebLogic Server インスタンス（各インスタンスは 1 つ以上のアプリケーションを実行）またはインスタンスのクラスタで構成されています。管理コンソールは、ドメインで実行されている指定された管理サーバーに接続されており、ドメイン内のマシンの構成またはランタイム状態の変更に使用できます。管理コンソールを使用すると、ドメインにおけるクラスタの定義、サーバーの追加、アプリケーションのデプロイ、アプリケーションの構成、および Web サーバー、Web サービス、Web リソースの管理ができます。

Oracle Application Server の開発ツールとデプロイメント・ツール

この項では、J2EE アプリケーションの作成に使用する開発ツールとデプロイメント・ツールについて説明します。これらのツールは、Oracle Developer Suite に収録されています。

開発ツール

アプリケーション開発者は、Oracle JDeveloper のツールを使用して、J2EE 準拠のアプリケーションを構築して OC4J にデプロイできます。JDeveloper は、複数階層の Java アプリケーションの作成を目的とした、フル機能搭載の統合開発環境である Oracle Internet Developer Suite のコンポーネントです。JDeveloper を使用すると、Java クライアント・アプリケーション、動的 HTML アプリケーション、Web サーバーおよびアプリケーション・サーバーのコンポーネント、およびデータベースのストアド・プロシージャの開発、デバッグおよびデプロイを、業界標準モデルに基づいて実行できます。JDeveloper では、複数階層の Java アプリケーションの作成に次の機能を利用できます。

- Oracle Business Components for Java（BC4J）
- Web アプリケーションの開発
- Java クライアント・アプリケーションの開発
- Java in the Database
- JavaBeans によるコンポーネント・ベースの開発
- 簡略化されたデータベース・アクセス
- ビジュアル統合開発環境
- J2EE 1.3 の全面サポート
- `.ear` ファイル、`.war` ファイル、`ejb-jar.xml` ファイルおよびデプロイメント・ディスクリプタの自動生成

Oracle JDeveloper でアプリケーションを構築し、Application Server Control コンソールまたは OC4J 管理コンソールを使用して、これらのアプリケーションを手動でデプロイできます。さらに、JDeveloper 以外の開発ツールでアプリケーションを構築することもできます。IBM VisualAge や Borland JBuilder で構築したアプリケーションを、OC4J にデプロイすることもできます。

注意： Oracle Application Server には、JDeveloper に加えて、オブジェクト・リレーショナル・マッピング・ツールの Oracle Application Server TopLink も付属しています。詳細は、『Oracle Application Server TopLink アプリケーション開発者ガイド』を参照してください。

注意： Oracle JDeveloper の移行用プラグイン・ツールである Oracle JDeveloper Application Migration Assistant を使用すると、移行が必要なアプリケーション・コードをすみやかに特定できます。このツールの詳細とダウンロード先については、1-8 ページの「移行ツール」を参照してください。

アセンブリ・ツール

Oracle Application Server では、J2EE アプリケーションの構成およびパッケージ化に、複数のアセンブリ・ツールを利用できます。これらのツールでできたものは J2EE 規格に準拠しており、OC4J 固有のものではありません。その内容は次のとおりです。

- WAR ファイルのアセンブリ・ツール : JSP、サーブレット、タグ・ライブラリおよび静的コンテンツを組み立てて、WAR ファイルにします。
- EJB アセンブラ : EJB ホーム、リモート・インタフェース、デプロイメント・ディスクリプタおよび EJB をパッケージ化して、標準の JAR ファイルにします。
- EAR ファイルのアセンブリ・ツール : WAR ファイルと EJB JAR を組み立てて、標準の EAR ファイルにします。
- タグ・ライブラリのアセンブリ・ツール : JSP タグ・ライブラリを組み立てて、標準の JAR ファイルにします。

管理ツール

Oracle Application Server には、そのコンポーネントの構成、監視および管理に、次の 2 種類の管理機能を利用できます。

- グラフィカルな管理ツールである Oracle Enterprise Manager 10*g* Application Server Control コンソール。OracleAS Cluster、OracleAS Farm および OC4J コンテナ全体にわたって一元管理を行います。
- コマンド・プロンプトによって、管理タスクをローカルまたはリモートで実行するコマンドライン・ツール（Application Server Control コンソールのほうが、このコマンドライン・ツールよりも統合性の高い管理サービスを実現できるため、管理環境としてはそちらを使用することをお薦めします）。

3

Java サーブレットの移行

この章では、WebLogic Server から Oracle Application Server に Java サーブレットを移行する際に必要な情報を説明します。ここでは、単純なサーブレット、WAR ファイルおよび展開 Web アプリケーションの移行を扱います。

この章は、次の項で構成されています。

概要

WebLogic Server から Oracle Application Server への Java サーブレットの移行は容易であり、移行するサーブレットのコード変更は、ほとんどあるいはまったく必要ありません。

どちらのアプリケーション・サーバーも、Sun 社の J2EE サーブレット仕様のバージョン 2.3 に完全に準拠しています。標準仕様に準拠して作成されているサーブレットはすべて正常に機能し、最小限の作業で移行を実現できます。

新環境へのサーブレットの移行に伴う主な作業は、構成とデプロイです。htmlKona など独自の拡張機能を使用する場合は、他の作業も必要となり、移行作業が複雑になります。

サーブレットの移行に伴う作業は、サーブレットがパッケージ化およびデプロイされている方法によっても異なります。サーブレットは、単純なサーブレット、標準ディレクトリ構造で他のリソースと一緒にパッケージ化された Web アプリケーション、または Web アーカイブ（WAR）ファイルとしてデプロイできます。

サーブレットの実装における WebLogic Server と Oracle Application Server の相違点

Oracle Application Server と WebLogic Server は、どちらもサーブレット 2.3 仕様をサポートしています。そのため、WebLogic Server から Oracle Application Server には、サーブレットを容易に移行できます。

OC4J の重要なサーブレット・コンテナ機能

OC4J のきわめて特徴的な機能の 1 つに、Single Sign-On（SSO）および Oracle Internet Directory（OID）とのシームレスな統合があります。この統合は、Java Authentication and Authorization Service（JAAS）規格の、オラクル社による実装を通じて実現します。JAAS プロバイダは OC4J に統合されています。

単純なサーブレットの移行

単純なサーブレットは、OC4J で容易に構成およびデプロイできます。サーブレットのデプロイに使用する手動処理は、WebLogic Server と OC4J で共通しています。

注意： サーブレットのデプロイ方法として推奨かつ優先されるのは、WAR ファイルまたは EAR ファイルでサーブレットをパッケージ化し、Oracle Enterprise Manager 10*g* Application Server Control コンソールまたは `dcmctl` コマンドライン・ユーティリティを使用するという方法です。`java` コマンドを使用して、コマンドラインで XML ファイルの編集と OC4J を起動する作業について、この章で示した手動処理は、開発目的以外には使用しないでください。これはあくまで解説目的で記述したものに過ぎません。

サーブレットは、Web アプリケーションの一部として登録および構成する必要があります。サーブレットを登録および構成するには、Web アプリケーションのデプロイメント・ディスクリプタに複数のエントリを追加する必要があります。

単純なサーブレットをデプロイする全体の手順は次のとおりです（詳細な手順は表 3-1 を参照）。

1. サーブレットのリクエストの解決に使用するサーブレット・クラス名と URL パターンで、Web アプリケーションのデプロイメント・ディスクリプタ（`web.xml`）を更新します。
2. サーブレット・クラス・ファイルを `WEB-INF/classes/` ディレクトリにコピーします。サーブレット・クラス・ファイルにパッケージ文が組み込まれている場合は、パッケージ文のレベルごとにサブディレクトリを追加作成します。その場合、サーブレット・クラス・ファイルは、そのパッケージに対して作成された一番下のサブディレクトリに配置する必要があります。
3. ブラウザでサーブレットの URL を入力し、サーブレットを起動します。

サーブレットの移行に伴う作業を明らかにするために、WebLogic Server 付属のサンプル・サーブレットを選択して移行しました。選択したサンプルは、独自の拡張機能を使用していないものです。

表 3-1 は、WebLogic Server から Oracle Application Server OC4J に、単純なサーブレットである HelloWorld を移行する際の手動処理を示しています。

表 3-1 単純なサーブレットの移行

ステップ	説明	プロセス
1	Web アプリケーションのデプロイメント・ディスクリプタを変更する	Oracle Application Server インストールの中で、次のディレクトリにある web.xml ファイルに、次に示すディスクリプタ情報を追加します。 UNIX の場合、web.xml は次のディレクトリにあります。 *<ORACLE_HOME>*/j2ee/home/default-web-app/WEB-INF/ Windows の場合、web.xml は次のディレクトリにあります。 *<ORACLE_HOME>*¥j2ee¥home¥default-web-app¥WEB-INF¥ 次のディスクリプタ情報を入力します。 <servlet> <servlet-name> HelloWorldServlet </servlet-name> <servlet-class> examples.servlets.HelloWorldServlet </servlet-class> </servlet> <servlet-mapping> <servlet-name> HelloWorldServlet </servlet-name> <url-pattern> /HelloWorldMigrate/* </url-pattern> </servlet-mapping>

表 3-1 単純なサーブレットの移行（続き）

ステップ	説明	プロセス
2	サーブレット・クラス・ファイルを適切なディレクトリにコピーする	WebLogic 付属のサンプル・サーブレットの実行が完了したら、次のように、WebLogic Server インストールのディレクトリから Oracle Application Server の適切なディレクトリに、`HelloWorldServlet.class` をコピーします。 UNIX の場合のコピー元 *<BEA_HOME>*`/weblogic700/samples/server/config/examples/applications/examplesWebApp/WEB-INF/classes/examples/servlets/` コピー先 *<ORACLE_HOME>*`/j2ee/home/default-web-app/WEB-INF/classes/examples/servlets/` Windows の場合のコピー元 *<BEA_HOME>*`¥weblogic700¥samples¥server¥config¥examples¥applications¥examplesWebApp¥WEB-INF¥classes¥examples¥servlets¥` コピー先 *<ORACLE_HOME>*`¥j2ee¥home¥default-web-app¥WEB-INF¥classes¥examples¥servlets¥` 注意 : WebLogic Server インストールに付属しているこのサーブレットは、`examples.servlets` と呼ばれるパッケージに属しています。このクラス・ファイルを Oracle Application Server にコピーする際は、対応するパッケージ・サブディレクトリ（例 : `examples/servlets/`）を作成する必要があります。
3	`home` OC4J インスタンスを再起動し、実行中でない場合は起動する	Oracle Enterprise Manager 10*g* Application Server Control コンソールの管理 Web ページを使用するか、`dcmctl` コマンドを次のように使用します。 `dcmctl start|restart -i` *<appsvr_instance_name>* `-ct oc4j -co home` *<appsvr_instance_name>* には、使用している Oracle Application Server インスタンスの名前を指定します。
4	Web ブラウザでサーブレットを実行する	Web ブラウザで URL を指定してサーブレットにアクセスします。 `http://localhost:7777/j2ee/HelloWorldMigrate` （別のマシンのブラウザを使用している場合は、`localhost` のかわりに、OC4J インスタンスのホスト名を指定します。）

関連項目： サーブレットの構成とデプロイの詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

WAR ファイルの移行

Web アプリケーションは、WAR ファイルとして構成およびデプロイできます。この操作は、Application Server Control コンソールの管理 GUI を使用するか、適切なディレクトリに WAR ファイルを手動でコピーすることにより、OC4J で容易に行うことができます。この操作方法は、WebLogic Server にも当てはまります。Application Server Control コンソールを使用して WebLogic Server のサンプルの WAR ファイルをデプロイする処理方法について示します。

注意： 適切なディレクトリに WAR ファイルを手動でコピーしてデプロイする方法は、OC4J がスタンドアロン・モードの開発環境でのみ使用します（Oracle Application Server インスタンスのコンポーネントでない場合）。

通常、本番用 Web アプリケーションは、Application Server Control コンソールまたは `dcmctl` ユーティリティによって、WAR ファイルまたは EAR ファイルを使用してデプロイされます。Web アプリケーションの開発時には、展開ディレクトリ形式を使用したほうが、編集コードのデプロイおよびテストを迅速化できる場合があります。

表 3-2 は、WebLogic Server から OC4J に WAR ファイルを移行する際の一般的な処理方法を示しています。

表 3-2 WAR ファイルの移行

ステップ	説明	プロセス

1	サンプル・アプリケーションの WAR ファイルを作成する。	WebLogic Server に付属している WebLogic Server サンプルのうち、実行していないものがある場合は、次のディレクトリに Cookie サンプル Web アプリケーションを構築します（ここでは UNIX の例を示しますが、Windows にも対応するディレクトリがあります）。
		`<BEA_HOME>/weblogic700/samples/server/src/examples/webapp/cookie`
		`ant` と入力し、このディレクトリにサンプル・アプリケーションを構築します。
		構築すると、このアプリケーションの WAR ファイルが次のディレクトリに作成されます。
		`<BEA_HOME>/samples/server/config/examples/applications/`

表 3-2 WAR ファイルの移行（続き）

ステップ	説明	プロセス
2	サンプル・アプリケーションをデプロイする。	1. cookie.war ファイルが配置されているマシンでブラウザを起動し、Application Server Control コンソールの URL に移動します。次に例を示します。 http://<*hostname*>:1810 1. 要求された場合は、管理者のユーザー名とパスワードを入力します。アプリケーションのデプロイ先とする Oracle Application Server インスタンスの名前をクリックします。 2. home OC4J コンポーネントをクリックすると、その設定ページが表示されます。 3. 「アプリケーション」をクリックします。home OC4J インスタンスの「アプリケーション」ページで、「WAR ファイルのデプロイ」をクリックします。「Web アプリケーションのデプロイ」ページが表示されます。 4. 「参照」ボタンをクリックし、cookie.war ファイルの場所を入力します。 5. 「アプリケーション名」テキスト・ボックスに「cookie」を、「URL にマップ」テキスト・ボックスに「/cookie」をそれぞれ入力します。「デプロイ」をクリックします。 6. デプロイされたアプリケーションのリストに Cookie アプリケーションが表示されます。
3	デプロイされたアプリケーションをテストする。	ブラウザで次の URL を入力します。 http://<*hostname*>:7777/cookie <*hostname*> には、Cookie サンプル・アプリケーションをデプロイした Oracle Application Server ホストを指定します。

関連項目： WAR ファイルおよび EAR ファイルのデプロイの詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』および『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

展開 Web アプリケーションの移行

Web アプリケーションの構成およびデプロイは、標準ディレクトリ構造に格納されたファイルの集合として実行することも、展開ディレクトリ形式で実行することもできます。この作業を OC4J で実行するには、OC4J インストールの適切なディレクトリに、標準ディレクトリ構造の内容を手動でコピーします。この方法は、WebLogic Server にも適用できます。この項では、展開 Web アプリケーションをデプロイする手動処理について説明します。

> **関連項目：** Oracle Enterprise Manager の管理 GUI の使用方法の詳細は、『Oracle Application Server 10*g* 管理者ガイド』を参照してください。

展開ディレクトリ形式による Web アプリケーションのデプロイは、主に Web アプリケーションの開発中に使用されます。この方法を使用すると、変更内容のデプロイとテストを迅速かつ容易に行うことができます。本番用 Web アプリケーションをデプロイするときは、Web アプリケーションを WAR ファイルでパッケージ化し、Application Server Control コンソールを使用してこの WAR ファイルをデプロイします。

WebLogic Server の展開 Web アプリケーションを手動でデプロイするには、展開 Web アプリケーション・ファイルが格納されている最上位のディレクトリを、WebLogic Server インストールの中で、次のディレクトリにコピーします。

（UNIX の場合）*<BEA_HOME>*/config/*<domain_name>*/applications
（Windows の場合）*<BEA_HOME>*¥config¥*<domain_name>*¥applications

適切なディレクトリに最上位のディレクトリをコピーし終わったら、最上位のディレクトリ内に空のファイルを作成し、REDEPLOY と命名します。WebLogic Server がこのファイルを検出し、Web アプリケーションをデプロイします（WebLogic Server は、このファイルのタイムスタンプを数分おきに読み取り、アプリケーションの再デプロイが必要かどうかを確認します。したがって、アプリケーション・ファイルが更新されるたびに、REDEPLOY ファイルのタイムスタンプが更新され、アプリケーション・ファイルが再デプロイされます。UNIX の場合は、touch コマンドを使用してこの処理を行います）。

OC4J で展開 Web アプリケーションを手動でデプロイする方法は、多少異なります。展開 Web アプリケーションが格納されている最上位のディレクトリを、OC4J インストールの中で、次のディレクトリにコピーします。

（UNIX の場合）*<ORACLE_HOME>*/j2ee/home/applications
（Windows の場合）*<ORACLE_HOME>*¥j2ee¥home¥applications

次のアプリケーション・デプロイメント・ディスクリプタを変更して、Web アプリケーションを組み込みます。

（UNIX の場合）*<ORACLE_HOME>*/config/application.xml
（Windows の場合）*<ORACLE_HOME>*¥config¥application.xml

次の Web サイトの XML ファイル（デフォルト以外の Web サイトを使用する場合は対応する XML ファイル）のエントリを追加し、Web サイトに Web アプリケーションをバインドします。

（UNIX の場合）*<ORACLE_HOME>*/config/default-web-site.xml
（Windows の場合）*<ORACLE_HOME>*¥config¥default-web-site.xml

最後に、次の各ファイルに新しい <application> タグ・エントリを追加して、このアプリケーションを新規登録します。

（UNIX の場合）*<ORACLE_HOME>*/config/server.xml
（Windows の場合）*<ORACLE_HOME>*¥config¥server.xml

server.xml を変更して保存すると、OC4J はこのファイルのタイムスタンプ変更を検出し、アプリケーションを自動的にデプロイします。OC4J を再起動する必要はありません。

構成とデプロイメント・ディスクリプタの移行

WebLogic Server と Oracle Application Server は J2EE 1.3 を全面的にサポートしているため、両方のアプリケーション・サーバー対応の XML 構成ファイルの標準セットを利用できます。その内容は次のとおりです。

- **web.xml**（Web アプリケーションの WAR ファイルの WEB-INF ディレクトリ内）
- **application.xml**（Web アプリケーションの WAR ファイルの META-INF ディレクトリ内）
- **ejb-jar.xml**（EJB モジュールの展開ディレクトリ階層の META-INF ディレクトリ内）

各アプリケーション・サーバーは、標準ファイルのほかに、それぞれの環境でのみ使用される固有のファイルを備えています。その内容は次のとおりです。

Oracle Application Server

- **server.xml**
  格納場所

  （UNIX の場合）*<ORACLE_HOME>*/j2ee/home/config/
  （Windows の場合）*<ORACLE_HOME>*¥j2ee¥home¥config¥

  全体的な OC4J ランタイム構成ファイルです。このファイルでは、デプロイされたアプリケーションのディレクトリ、サーバー・ログ・ファイルのパスと名前、その他の XML ファイルのパスと名前、アプリケーションとその EAR ファイルの名前、ランタイム・ライブラリのパスなどの属性が定義されます。

- **application.xml**
  格納場所

  （UNIX の場合）*<ORACLE_HOME>*/j2ee/home/config/
  （Windows の場合）*<ORACLE_HOME>*¥j2ee¥home¥config¥

  特定の OC4J インストールにデプロイされた全アプリケーションに対するグローバル構成ファイルの共通設定です。このファイルは、J2EE WAR ファイルの application.xml とは異なります。

- **<*website_name*>-web-site.xml**
  格納場所

  （UNIX の場合）<*ORACLE_HOME*>/j2ee/home/config/
  （Windows の場合）<*ORACLE_HOME*>¥j2ee¥home¥config¥

  このファイルでは Web サイトが定義され、ホスト名、HTTP リスナー・ポート番号、Web サイトで提供される Web アプリケーションとその URL コンテキスト、HTTP アクセス・ログのファイルとパスなどの属性が指定されます。*-web-site.xml ファイルごとの名前とパスを、OC4J の server.xml ファイルで指定し、定義された Web サイトを実行時に構成する必要があります。

- **data-sources.xml**
  格納場所

  （UNIX の場合）<*ORACLE_HOME*>/j2ee/home/config/
  （Windows の場合）<*ORACLE_HOME*>¥j2ee¥home¥config¥

  このファイルには、OC4J ランタイムで使用されるデータソースの構成情報が格納されています。このファイルには、使用される JDBC ドライバ、各データソースの JNDI バインド、各データソースのユーザー名とパスワード、使用するデータベース・スキーマ、各データベースへの最大接続数、タイムアウト値などの情報が格納されています。

- **principals.xml**
  格納場所

  （UNIX の場合）<*ORACLE_HOME*>/j2ee/home/config/
  （Windows の場合）<*ORACLE_HOME*>¥j2ee¥home¥config¥

  このファイルには、デフォルトの XMLUserManager クラスのユーザー・リポジトリが格納されています。グループとそのグループに属するユーザーおよびグループの権限が、このファイルで定義されます。ロールへのグループのマッピングが、グローバルな application.xml ファイルで定義されます。

- **orion-application.xml**
  格納場所

  （UNIX の場合）<*ORACLE_HOME*>/j2ee/home/application-deployments/<app_name>

  または

  （Windows の場合）<*ORACLE_HOME*>¥j2ee¥home¥application-deployments¥<*app_name*>

  このファイルには、OC4J インストールにデプロイされたアプリケーション（<app_name>）における、OC4J 固有の情報が格納されています。このファイルには、アプリケーションの Web モジュールと EJB モジュールの名前およびセキュリティ情報も格納されています。このファイルは、デプロイ時に OC4J によって生成されます。

- **global-web-application.xml**
  格納場所

  （UNIX の場合）<*ORACLE_HOME*>/j2ee/home/config/
  （Windows の場合）<*ORACLE_HOME*>¥j2ee¥home¥config¥

  このファイルには、OC4J ランタイムで内部的に使用されるサーブレット構成情報が格納されています。この情報には、JSP トランスレータ・サーブレットなどがあります。

- **orion-web.xml**
  格納場所

  （UNIX の場合）
  <*ORACLE_HOME*>/j2ee/home/application-deployments/
  <app_name>/<*web_app_name*>/

  または

  （Windows の場合）
  <*ORACLE_HOME*>¥j2ee¥home¥application-deployments¥
  <*app_name*>¥<*web_app_name*>¥

  このファイルでは、<web_app_name> に関する、OC4J 内部の JSP 情報およびサーブレット情報が指定されます。このファイルは、デプロイ時に OC4J によって生成されます。

- **orion-ejb-jar.xml**
  格納場所

  （UNIX の場合）
  <*ORACLE_HOME*>/j2ee/home/application-deployments/
  <app_name>/<ejb_jarfile_name>/

  または

  （Windows の場合）
  <*ORACLE_HOME*>¥j2ee¥home¥application-deployments¥
  <*app_name*>¥<ejb_jarfile_name>¥

  このファイルには、アプリケーション <app_name> に属する <ejb_jarfile_name> で指定されている、JAR ファイルの EJB に関する OC4J 内部のデプロイ情報が格納されています。このファイルは、デプロイ時に OC4J によって生成されます。

- **oc4j-connectors.xml**
  格納場所

  （UNIX の場合）<*ORACLE_HOME*>/j2ee/home/config/
  （Windows の場合）<*ORACLE_HOME*>¥j2ee¥home¥config¥

  このファイルには、OC4J インストールのコネクタ情報が格納されています。

WebLogic Server

- **config.xml**
  格納場所

  （UNIX の場合）*<BEA_HOME>*/config/<domain_name>/
  （Windows の場合）*<BEA_HOME>*¥config¥*<domain_name>*¥

  このファイルには、WebLogic Server ドメイン全体の構成情報が格納されています。このファイルで指定される情報には、ドメイン管理サーバーのホスト名や管理ポート番号、データソースへの JNDI マッピング、JDBC 接続プール情報、ドメイン内のすべてのノードにデプロイされたアプリケーション、SSL 証明書情報などがあります。

- **weblogic.xml**
  格納場所

  （UNIX の場合）
  <BEA_HOME>/config/<domain_name>/applications/
  <web_app_name>/WEB_INF/

  または

  （Windows の場合）
  <BEA_HOME>¥config¥*<domain_name>*¥applications¥
  <web_app_name>¥WEB_INF¥

  このファイルでは、JSP のプロパティ、JNDI マッピング、リソース参照、セキュリティ・ロール・マッピング、および Web アプリケーションの HTTP セッションと Cookie パラメータが定義されます。これは WebLogic Server 固有のファイルですが、作成は手動で行います。

- **weblogic-ejb-jar.xml**
  EJB モジュールの META-INF サブディレクトリ内にあります。このファイルによって、WebLogic Server のリソースが EJB にマップされます。これらのリソースには、セキュリティ・ロール名、データソース、JMS コネクション、その他の EJB などがあります。このファイルには、対応する ejb-jar.xml ファイルで定義されている EJB の、キャッシュとクラスタリングのパフォーマンス属性も格納されています。

注意： 前述の各ファイルは、それぞれのアプリケーション・サーバーで使用される XML 構成ファイルをすべて網羅したものではありません。これらのファイルは、サーブレット・アプリケーションの構成とデプロイに関連するものです。これ以外に、HTTP リスナー、RMI、セキュリティなどのコンポーネントを構成する XML ファイルも存在します。

クラスタ対応アプリケーションの移行

Oracle Application Server には、WebLogic Server よりも包括的なクラスタリング機能が用意されています。

WebLogic Server には、HTTP セッション状態のクラスタリングとオブジェクトのクラスタリングという、2 つの主要なクラスタ・サービスがあります。この項では主に、HTTP セッション状態のクラスタリングまたは Web アプリケーションのクラスタリングについて説明します。

WebLogic Server は、クラスタリングされたサーブレットと JSP ページにアクセスするクライアントの HTTP セッション状態をレプリケートすることにより、サーブレットと JSP ページのクラスタリングをサポートします。HTTP セッション状態のクラスタリングによる恩恵を受けるには、メモリー内レプリケーション、ファイル・システムの永続性、JDBC の永続性のいずれかを構成し、HTTP セッション状態を持続させる必要があります。

クラスタリング・サポートは、Oracle Application Server と WebLogic Server で類似しています。Oracle Application Server には、さらに次の特徴があります。

- **サーブレットのクラスタリング**: OC4J には、Web アプリケーションを変更せずにサーブレットをクラスタリングする機能があります。変更が必要なのは、Web アプリケーションに対して透過的なデプロイ構成です。この変更により、複数の OC4J プロセスへのセッションのフェイルオーバーが可能になります。
- **クラスタリングのアーキテクチャと簡易性**: Oracle Application Server の重要な差別化要因として、各種インスタンスのクラスタリングが簡単であることと、クラスタリングに使用されるアーキテクチャが堅牢であることがあげられます。
- **クラスタリングの簡易性**: Oracle Enterprise Manager 10*g* Application Server Control コンソールには、様々な OracleAS インスタンスを、1 つのクラスタに属するように構成する GUI が用意されています。この構成では、インスタンスはロード・バランシングを使用した複数のサーバーであり、それは 1 台のマシンと複数台のマシンのどちらに配置されていてもかまいません。あるいは、1 つの XML ファイルを編集することもできます。一方、1 台または複数台のマシンに複数のインスタンスを配置した状態で、ロード・バランシングを使用した WebLogic Server クラスタを構成するほうが、もっと複雑です。
- **優れたクラスタリング・アーキテクチャ**: OC4J では、動的な IP アドレスによって、インスタンスがクラスタのメンバーとして登録されます。Cisco Local Director や BigIP などの標準的ロード・バランサは、ロード・バランシングの各種メカニズムを利用して、複数の Oracle Application Server インスタンスにリクエストをルーティングできます。さらに、`mod_oc4j` では、ロード・バランシングの各種アルゴリズムのいずれかを使用して、Oracle HTTP Server から OC4J プロセスに対して、リクエストを識別してルーティングすることができます。一方、WebLogic Server では、静的な IP アドレスによってクラスタリングが構成されます。静的な IP アドレスの場合、ロード・バランサを使用してインスタンス間でリクエストを配布することはできません。そのため、WebLogic Server ではクラスタリングとロード・バランシングのいずれかを利用できますが、両方は利用できません。

関連項目：『Oracle Application Server 10*g* 高可用性ガイド』

各 OracleAS Farm は複数の OC4J アイランドで構成されており、各アイランドは複数のアプリケーションで構成される場合があります。フェイルオーバーのためのセッション状態の共有は、特定のアイランド内で行われます。

OracleAS Cluster と OC4J アイランドの設定方法の詳細は、『Oracle Application Server 10*g* 高可用性ガイド』を参照してください。

4

JSP ページの移行

この章では、WebLogic Server から Oracle Application Server に JavaServer Pages を移行する際に必要な情報を説明します。ここでは、単純な JSP ページ、JSP タグのカスタム・ライブラリおよび WebLogic のカスタム・タグの移行を扱います。

この章は、次の項で構成されています。

概要

WebLogic Server から Oracle Application Server への JSP ページの移行は容易であり、コード変更はほとんどあるいはまったく必要ありません。

どちらのアプリケーション・サーバーも、Sun 社の JavaServer Pages 仕様のバージョン 1.1 および 1.2 に完全に準拠しています。標準仕様に準拠して作成されている JSP ページはすべて正常に機能し、最小限の作業で移行を実現できます。

新環境への JSP ページの移行に伴う主な作業は、構成とデプロイです。独自の拡張機能やタグ・ライブラリを使用する場合は、他の作業も必要となり、移行作業が複雑になります。

JSP ページの移行に伴う作業は、JSP ページがパッケージ化およびデプロイされている方法によっても異なります。JSP ページは、単純な JSP ページ、標準ディレクトリ構造で他のリソースと一緒にパッケージ化された Web アプリケーション（WAR ファイル）、またはエンタープライズ・アプリケーションのアーカイブ（EAR）ファイルとしてデプロイできます。展開ディレクトリ形式および WAR ファイルによる Web アプリケーションの移行は、第 3 章「Java サーブレットの移行」で扱っています。

JSP の実装における WebLogic Server と Oracle Application Server の相違点

WebLogic Server と Oracle Application Server Containers for J2EE（OC4J）では、どちらも同じバージョンの JavaServer Pages 仕様が実装されているため、JSP のコア仕様領域については両者に相違点はありません。コア仕様以外の領域には相違点があります。その内容を表 4-1 に列挙します。

表 4-1 JSP 機能の比較

機能	Oracle Application Server	WebLogic Server
サポートしている JSP バージョン	1.2	1.2
JSP タグの基本ライブラリ	○	○
JSP タグの拡張ライブラリ	○	×
JSP のソース・レベルでのデバッグ	○	×
ASP から JSP へのソース・レベルでの変換	○	×

各ベンダーは、自社のカスタム JSP タグを提供しています。WebLogic Server には、JSP ページで使用可能な専用 JSP タグが 4 つあります。OC4J では、Oracle JSP Markup Language（JML）カスタム・タグ・ライブラリ、XML および XSL との統合用タグ、複数の JSP ユーティリティ・タグなど、各種の JSP タグを利用することもできます。これらのタグの総合的な説明については、『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』を参照してください。

OC4J JSP の機能

Oracle Application Server は、市場でトップクラスの高速 JSP エンジンを搭載しています。さらに、グローバリゼーションや SQLJ のサポートなど、付加価値機能と拡張機能も各種用意されています。OC4J を初めて搭載したリリースの Oracle Application Server、すなわち Oracle9*i*AS 1.0.2.2 では、2 つの JSP コンテナがありました。オラクル社が開発したコンテナ（旧名 OracleJSP）と、Ironflare AB 社から使用許諾を受けたコンテナ（旧名 Orion JSP コンテナ）です。

Oracle Application Server では、OC4J JSP コンテナと呼ぶ 1 つの JSP コンテナにこれらのコンテナが統合されました。この新しいコンテナは、OC4J サーブレット・コンテナのサーブレットとして効率的な実行が可能であり、また他の OC4J コンテナとの緊密な統合が可能であるなど、それぞれの旧バージョンの長所を備えています。この統合コンテナは、主として OracleJSP トランスレータ、簡略化された新しいディスパッチャで実行される Orion コンテナ・ランタイム、および OC4J 1.0.2.2 コア・ランタイム・クラスで構成されています。その結果、標準の JSP 仕様を超える機能を備えた、市場でトップクラスの高速な JSP エンジンとなりました。

OC4J JSP では、カスタム・タグ・ライブラリ、カスタム JavaBeans およびカスタム・クラスによる次の拡張機能を利用できます。通常、これらの拡張機能は他の JSP 環境に移植できます。

- スコープの指定が可能な JavaBeans として実装された拡張タイプ。
- イベント処理用の `JspScopeListener`。
- カスタム・タグによる XML および XSL との統合。
- Data-access JavaBeans。
- JSP 開発に必要な Java の技能レベルを軽減する Oracle JSP Markup Language（JML）カスタム・タグ・ライブラリ。

  OC4J JSP には、接続プーリング・タグ、XML タグ、EJB タグ、ファイル・アクセス・タグ、電子メール・タグ、キャッシュ・タグ、OracleAS Personalization タグ、OracleAS Ultrasearch タグ、および SQL 機能のカスタム・タグ・ライブラリが組み込まれています。これに対して WebLogic に組み込まれているのは、`cache`、`process`、`repeat`、フォームの検証の 4 つのみです。

- コンテンツ配信ネットワークのエッジ・サーバーで使用して、Web コンテンツにインテリジェントなキャッシュ・ソリューションを提供する Edge Side Includes for Java（JESI）タグ、Web Object Cache タグおよび API（以降の各項目を参照）。

  関連項目： JSP タグのカスタム・ライブラリの詳細は、『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』を参照してください。

OC4J JSP コンテナでは、自動ページ再コンパイルとクラス再ロードのモード切替え機能、JSP インスタンスのプーリング、タグ・ハンドラ・インスタンスのプーリングなど、いくつかの重要な機能も利用できます。

Edge Side Includes for Java（JESI）タグ OC4J のファイングレイン・コントロールを使用すると、開発者は JSP ページの断片を各タグにキャッシュできます。これらの断片は OracleAS Web Cache にキャッシュして自動的に無効化し、JSP 変更時にリフレッシュできます。この処理は、Edge Side Includes（ESI）というテクノロジに支えられています。これは W3C 規格の XML スキーマ / マークアップ言語で、Web Cache での動的コンテンツのキャッシュやエッジ・ネットワークでの動的コンテンツの作成を可能にするものです。この動的コンテンツをキャッシュすることにより、JSP やサーブレットを実行する必要性が減ります。その結果、パフォーマンスの向上、アプリケーション・サーバーのオフロード、待機時間の短縮が実現します。JESI（JSP、ESI）タグのレイヤーは Edge Side Includes（ESI）フレームワークの一番上に位置し、JSP アプリケーションで ESI キャッシュ機能を利用できます。JESI タグを使用すると、JSP ページの動的コンテンツをキャッシュ可能なコンポーネントまたは断片に分解できます。

Web Object Cache タグ Web Object Cache は、Oracle Application Server の機能で、JSP またはサーブレットによって生成された部分結果や中間結果の取得、格納、再利用、後処理およびメンテナンスを、Java で作成した Web アプリケーションで実行可能にするものです。プログラム・インタフェースでは、タグ・ライブラリ（JSP ページ用）と Java API（サーブレット用）を利用できます。キャッシュ内のオブジェクトを構成するものとしては、HTML 断片、XML 断片、XML DOM オブジェクト、シリアライズ可能な Java オブジェクトなどが考えられます。これらのオブジェクトをメモリーにキャッシュすると、キャッシュ内のオブジェクトに対して、次のような各種操作を実行できます。

- ユーザー・プロファイルまたは格納されている XML のデバイス特性に基づく各種 XSLT の適用
- クライアントへの電子メール送信に使用する SMTP など、HTTP 外部のキャッシュ内のオブジェクトの再利用

Oracle JDeveloper と OC4J JSP コンテナ

Oracle JDeveloper は OC4J JSP コンテナと統合され、JSP ページの編集、ソース・レベルでのデバッグおよび実行から成る JSP アプリケーションの開発サイクルを完全にサポートしています。また、データ対応および Web 対応の JavaBeans（JDeveloper Web Beans）も広範に利用できます。また JSP エレメント・ウィザードが用意されており、事前定義された Web Beans を容易にページに追加できます。JDeveloper 独自の機能もあり、開発者にご好評をいただいています。JDeveloper では、JSP ページ・ソース内にブレーク・ポイントを設定して、JSP ページから JavaBeans へのコールを追跡できます。この方法は、JSP ページ内に出力文を追加し、ブラウザ表示のために状態を応答ストリームに出力したり、状態をサーバー・ログに出力する手動デバッグ手法よりもはるかに便利です。

単純な JSP ページの移行

JSP ページでは、HTTP サーブレットで必要とされる固有のマッピングが不要です。単純な JSP ページをデプロイするには、JSP ページと JSP ページに必要なファイルを、適切なディレクトリにコピーします。それ以外の登録作業は不要です。

注意： JSP を含むアプリケーションのデプロイには、Application Server Control コンソールを使用してください。ただし、内容をわかりやすくするため、次の例では Application Server Control コンソールを使用せず、手動で JSP ファイルをコピーします。

J2EE Web アプリケーションと様々な構成ファイルがデフォルトで用意されているので、OC4J でのデプロイ処理は簡略化されました。

JSP ページの移行に伴う作業を明らかにするために、WebLogic Server 付属のサンプル JSP ページを選択して移行しました。選択したサンプルは、独自の拡張機能を使用していないものです。

表 4-2 は、WebLogic Server から OC4J に単純な JSP ページを移行する際の一般的な処理方法を示しています。

表 4-2 単純な JSP ページの移行

ステップ	説明	プロセス
1	OC4J のインスタンスを起動する（実行中でない場合）	`http://<hostname>:1810` に移動し、起動する OC4J インスタンスを選択します（*<hostname>* は各自の Oracle Application Server ホストの名前です）。または、`dcmctl` コマンドを次のように使用します。 `dcmctl start -i <appsvr_instance_name> -ct oc4j -co home` *<appsvr_instance_name>* には、使用している Oracle Application Server インスタンスの名前を指定します。

表 4-2 単純な JSP ページの移行（続き）

ステップ	説明	プロセス
2	JSP ページを適切なディレクトリにコピーする	次のように、WebLogic Server インストールのディレクトリから Oracle Application Server の適切なディレクトリに、`HelloWorld.jsp` をコピーします。 UNIX の場合のコピー元 *<BEA_HOME>*`/weblogic700/samples/server/src/examples/jsp/` コピー先 *<ORACLE_HOME>*`/j2ee/home/default-web-app/` Windows の場合のコピー元 *<BEA_HOME>*`¥weblogic700¥samples¥server¥src¥examples¥jsp¥` コピー先 *<ORACLE_HOME>*`¥j2ee¥home¥default-web-app¥`
3	JSP ページに必要なファイルをコピーする	次のように、WebLogic Server インストールのディレクトリから Oracle Application Server の適切なディレクトリに、`BEA_Button_Final_web.gif` をコピーします。 UNIX の場合のコピー元 *<BEA_HOME>*`/weblogic700/samples/server/src/examples/images/` コピー先 *<ORACLE_HOME>*`/j2ee/home/default-web-app/images/` Windows の場合のコピー元 *<BEA_HOME>*`¥weblogic700¥samples¥server¥src¥examples¥images¥` コピー先 *<ORACLE_HOME>*`¥j2ee¥home¥default-web-app¥images¥` 注意：`images` ディレクトリの作成が必要になることもあります。

表 4-2 単純な JSP ページの移行（続き）

ステップ	説明	プロセス
4	Web ブラウザで JSP ページを要求する	Web ブラウザで、次の URL を指定して JSP ページを要求します。 `http://<hostname>:7777/j2ee/HelloWorld.jsp` *<hostname>* には、JSP ファイルをコピーした Oracle Application Server ホストを指定します。

関連項目： JSP ページの構成とデプロイの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』および『Oracle9*i*AS Containers for J2EE ユーザーズ・ガイド』を参照してください。

JSP のカスタム・タグ・ライブラリからの移行

WebLogic Server と OC4J では、JSP のカスタム・タグを作成して使用することができます。JSP のカスタム・タグ・ライブラリのデプロイに使用する処理は、WebLogic Server と OC4J でほぼ共通しています。

タグ・ライブラリは、Web アプリケーションの一部としてパッケージ化してデプロイすることが可能であり、Web アプリケーションのデプロイメント・ディスクリプタの特定のセクションで宣言されます。

JSP のカスタム・タグ・ライブラリの移行に伴う作業を明らかにするために、WebLogic Server 付属のサンプル JSP ページを選択して移行しました。選択したサンプルは、独自の拡張機能を使用していないものです。

表 4-3 は、WebLogic Server から OC4J に、JSP のカスタム・タグ・ライブラリを使用する JSP ページを移行する際の一般的な処理方法を示しています。

表 4-3 JSP のカスタム・タグ・ライブラリからの移行

ステップ	説明	プロセス
1	タグ・ライブラリ・ファイルを適切なディレクトリにコピーする	counter.tld をコピーします。WebLogic Server 内のコピー元ディレクトリは次のとおりです。 UNIX の場合 : *<BEA_HOME>*/weblogic700/samples/server/src/examples/jsp/tagext/counter/ Windows の場合 : *<BEA_HOME>*¥weblogic700¥samples¥server¥src¥examples¥jsp¥tagext¥counter¥ OC4J 内のコピー先ディレクトリは次のとおりです。 UNIX の場合 : *<ORACLE_HOME>*/j2ee/home/default-web-app/WEB-INF/ Windows の場合 : *<ORACLE_HOME>*¥j2ee¥home¥default-web-app¥WEB-INF¥
2	タグ・ライブラリを使用する JSP ファイルを適切なディレクトリにコピーする	pagehits.jsp をコピーします。WebLogic Server 内のコピー元ディレクトリは次のとおりです。 UNIX の場合 : *<BEA_HOME>*/weblogic700/samples/server/src/examples/jsp/tagext/counter/ Windows の場合 : *<BEA_HOME>*¥weblogic700¥samples¥server¥src¥examples¥jsp¥tagext¥counter¥ OC4J 内のコピー先ディレクトリは次のとおりです。 UNIX の場合 : *<ORACLE_HOME>*/j2ee/home/default-web-app/ Windows の場合 : *<ORACLE_HOME>*¥j2ee¥home¥default-web-app¥

表 4-3 JSP のカスタム・タグ・ライブラリからの移行（続き）

ステップ	説明	プロセス
3	タグ・ライブラリに必要なクラス・ファイルと JSP ファイルで使用されるクラス・ファイルを、適切なディレクトリにコピーする	Count.class、Display.class および Increment.class をコピーします。WebLogic Server 内のコピー元ディレクトリは次のとおりです。 UNIX の場合 : *<BEA_HOME>*/weblogic700/samples/server/config/examples/applications/examplesWebApp/WEB-INF/classes/examples/jsp/tagext/counter/ Windows の場合 : *<BEA_HOME>*¥weblogic700¥samples¥server¥config¥examples¥applications¥examplesWebApp¥WEB-INF¥classes¥examples¥jsp¥tagext¥counter¥ OC4J 内のコピー先ディレクトリは次のとおりです。 UNIX の場合 : *<ORACLE_HOME>*/j2ee/home/default-web-app/WEB-INF/classes/examples/jsp/tagext/counter/ Windows の場合 : *<ORACLE_HOME>*¥j2ee¥home¥default-web-app¥WEB-INF¥classes¥examples¥jsp¥tagext¥counter¥ WebLogic Server インストールに付属しているこれらの .class ファイルは、examples.jsp.tagext.counter と呼ばれるパッケージに属しています。examples/jsp/tagext/counter/ ディレクトリ（または相当する Windows のディレクトリ）の作成が必要になることもあります。

表 4-3 JSP のカスタム・タグ・ライブラリからの移行（続き）

ステップ	説明	プロセス
4	JSP ファイルで使用されるイメージ・ファイルをコピーする	イメージ・ファイルが格納されているディレクトリをコピーします。WebLogic Server 内のコピー元ディレクトリは次のとおりです。 UNIX の場合 : *<BEA_HOME>*/weblogic700/samples/server/src/examples/jsp/tagext/counter/images/numbers/ Windows の場合 : *<BEA_HOME>*¥weblogic700¥samples¥server¥src¥examples¥jsp¥tagext¥counter¥images¥numbers¥ OC4J 内のコピー先ディレクトリは次のとおりです。 UNIX の場合 : *<ORACLE_HOME>*/j2ee/home/default-web-app/images/numbers/ Windows の場合 : *<ORACLE_HOME>*¥j2ee¥home¥default-web-app¥images¥numbers¥ images/numbers ディレクトリ（または相当する Windows のディレクトリ）の作成が必要になることもあります。

表 4-3 JSP のカスタム・タグ・ライブラリからの移行（続き）

ステップ	説明	プロセス
5	適切な Web アプリケーションのデプロイメント・ディスクリプタを変更し、その内容を保存する	OC4J インストールの中で、次のディレクトリにある web.xml ファイルに、次に示すディレクティブ・エントリを追加します。 UNIX の場合 : <*ORACLE_HOME*>/j2ee/home/default-web-app/WEB-INF/ Windows の場合 : <*ORACLE_HOME*>¥j2ee¥home¥default-web-app¥WEB-INF¥ ディレクティブ・エントリ（<taglib> は <web-app> の子要素）: <taglib> <taglib-uri> counter </taglib-uri> <taglib-location> /WEB-INF/counter.tld </taglib-location> </taglib>
6	OC4J インスタンスを再起動し、実行中でない場合は起動する	http://<*hostname*>:1810 に移動し、home OC4J インスタンスを再起動または起動します。または、dcmctl コマンドを次のように使用します。 dcmctl restart|start -i <*appsvr_instance_name*> -ct oc4j -co home <*appsvr_instance_name*> には Oracle Application Server インスタンスの名前を、<*hostname*> には Oracle Application Server ホストの名前を指定します。
7	Web ブラウザで JSP ファイルを要求する	Web ブラウザで次の URL にアクセスします。 http://<*hostname*>:7777/j2ee/pagehits.jsp <*hostname*> には、ファイルをコピーした Oracle Application Server ホストを指定します。

関連項目：

- JSP ページの構成とデプロイの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。
- JSP タグのカスタム・ライブラリの詳細は、『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』を参照してください。

WebLogic のカスタム・タグからの移行

Web アプリケーション全体で WebLogic のカスタム・タグを広く使用している場合に最適な移行オプションは、WebLogic のタグ・ライブラリを OC4J にデプロイして使用することです。このオプションは、前項の「JSP のカスタム・タグ・ライブラリからの移行」に記載されています。その後、必要に応じて OC4J JSP タグに移行できます。

Web アプリケーション全体で WebLogic のカスタム・タグを多用していない場合に最適な移行オプションは、JSP ページを変更して OC4J JSP タグ・ライブラリを使用することです。このオプションについては、この項で後述します。

WebLogic Server には、JSP ページで使用可能な専用 JSP タグが 3 つあります。そのタグとは、`cache`、`process` および `repeat` です。

WebLogic Server `cache` タグ

OC4J では、WebLogic Server `cache` タグのスーパーセットが、Web Object Cache タグの形式で用意されています。このタグでは、WebLogic `cache` タグにはない機能を利用できます。さらに、OC4J の Web Object Cache タグは、XML タグ・ライブラリなど、他のタグ・ライブラリと緊密に統合されています。たとえば、`cacheXMLObj` タグは、OC4J の XML タグと緊密に統合されています。

ダイレクト機能マッピングが行われない機能の 1 つに非同期があります。ただし、Edge Side Includes（ESI）と Edge Side Includes for Java（JESI）では、これに類似した機能を利用できます。

関連項目： Web Object Cache タグと JESI タグの詳細は、『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』を参照してください。

WebLogic Server `process` タグ

OC4J には、`process` タグにそのまま相当するタグはありません。一番近いオプションは、Oracle JSP Markup Language（JML）の `jml:useForm` タグおよび `jml:if` タグを使用することです。

> **関連項目：** これらの JML タグの詳細は、『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』の第 3 章の中で、JSP Markup Language（JML）タグの説明に関する項のうち、Bean バインド・タグの説明という項目と、ロジックおよびフロー制御タグの説明という項目を参照してください。

または、Java コードを記述してタグを実装することもできます。

WebLogic Server `repeat` タグ

OC4J でこのタグに相当するものは `jml:foreach` タグです。このタグを使用すると、同種の値セットを反復することができます。タグの本体は、セット内の要素ごとに 1 回実行されます。現時点では、このタグは次のタイプのデータ構造の反復をサポートしています。

- Java 配列
- `java.util.Enumeration`
- `java.util.Vector`

ただし、これらのタグは、Iterators、Collections、ハッシュテーブルのキーなどのデータ構造は対象としていません。

> **関連項目：** この JML タグの詳細は、『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』の第 3 章の中で、JSP Markup Language（JML）タグの説明に関する項のうち、ロジックおよびフロー制御タグの説明という項目を参照してください。

`ResultSets` と `ResultSetMetaData` に対しては、SQL Tags for Data Access と呼ばれるタグを OC4J で利用できます。このタグには、WebLogic Server の `repeat` タグとよく似た機能を利用できます。`dbNextRow` タグは、おそらく最も注目度の高いタグです。このタグを使用すると、`dbQuery` タグで取得され、指定された `queryId` に関連付けられている結果セットを、行ごとに処理できます。処理コードは、タグの本体のうち、`dbNextRow` の開始タグと終了タグの間に配置します。本体内のコードは、結果セットの行ごとに実行されます。

関連項目：

- これらの JML タグの詳細は、『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』の第 4 章の中で、SQL Tags for Data Access に関する項のうち、Data-Access タグのカスタム・ライブラリという項目を参照してください。
- JSP タグの標準ライブラリ・フレームワークと tag-extra-info クラスの詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

JSP ページのプリコンパイル

JSP ページは、JSP コンパイラによって自動的にコンパイルされます。ただし、JSP ページのテスト時およびデバッグ時には、JSP コンパイラを直接操作することもできます。

JSP コンパイラによって、`.jsp` ファイルが `.java` ファイルに解析されます。その際、標準的な Java コンパイラによって、`.java` ファイルが `.class` ファイルにコンパイルされます。

WebLogic Server の JSP コンパイラの使用方法

WebLogic Server の JSP コンパイラを起動するには、WebLogic Server のコマンドライン環境に次のコマンドを入力します。

```
java weblogic.jspc -options fileName
```

`fileName` パラメータは、コンパイルする JSP ページの名前を指します。JSP ページ名の前後にオプションを指定できます。次の例は、`-d` オプションを使用して、コンパイル先のディレクトリ `weblogic/classes` に `myFile.jsp` をコンパイルする方法を示しています。

```
java weblogic.jspc -d /weblogic/classes myFile.jsp
```

OC4J JSP プリトランスレータの使用方法

標準的な `jsp_precompile` 方式に加えて、OC4J では、コマンドライン・ユーティリティ `ojspc` を使用して、JSP ページの事前変換を行うことができます。

JSP ページ `HelloWorld.jsp` が OC4J Web アプリケーションの次のデフォルト・ディレクトリに配置されている例を想定します（*<ORACLE_HOME>*`/j2ee/home/default-web-app/` またはこれに相当する Windows のディレクトリから、`HelloWorld.jsp` ファイルをこのサブディレクトリにコピーします）。

UNIX の場合 :

<ORACLE_HOME>`/j2ee/home/default-web-app/examples/jsp/`

Windows の場合 :

```
<ORACLE_HOME>¥j2ee¥home¥default-web-app¥examples¥jsp¥
```

この JSP ページを事前変換するには、カレント・ディレクトリをアプリケーションのルート・ディレクトリに設定し、ojspc で -d オプションを指定して、_pages ディレクトリを出力ベース・ディレクトリに設定します。これで、適切なパッケージ名とファイル階層が設定されます。次に例を示します。

注意 : ojspc に対して適切な java 実行可能ファイルが使用されるように、パス環境変数で <ORACLE_HOME>/jdk/bin を設定してください。

UNIX の場合（% は UNIX のプロンプトを意味します）:

```
% cd j2ee/home/default-web-app
% ojspc -d ../application-deployments/default/defaultWebApp/persistence/_pages
     examples/jsp/HelloWorld.jsp
```

Windows の場合（コマンド・プロンプト・ウィンドウを使用。Oracle は Oracle Application Server インストールの Oracle Home です）:

```
C:¥>cd Oracle¥j2ee¥home¥default-web-app
C:¥>ojspc -d ../application-deployments/default/defaultWebApp/persistence/_pages
     examples/jsp/HelloWorld.jsp
```

このディレクトリ構造では、examples/jsp/HelloWorld.jsp というアプリケーション相対パスが指定されます。変換された JSP は、次のディレクトリに格納されます。

UNIX の場合 :

```
<ORACLE_HOME>/j2ee/home/application-deployments/default/
defaultWebApp/persistence/_pages/_examples/_jsp/
```

Windows の場合 :

```
<ORACLE_HOME>¥j2ee¥home¥application-deployments¥default¥
defaultWebApp¥persistence¥_pages¥_examples¥_jsp¥
```

実行時に JSP コンテナは、_pages サブディレクトリの中で、コンパイルされた JSP ファイルを探します。この例のように実行すると、ojspc によって、_examples/_jsp/ サブディレクトリが自動的に作成されます。

http://*<hostname>*:7777/j2ee/examples/jsp/HelloWorld.jsp という URL を指定して、JSP ページを起動します。事前変換しない場合よりも応答時間が短くなることがわかります。

関連項目：『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』の「JSP の変換とデプロイ」を参照してください。

実行を伴わない標準的な JSP 事前変換（JSP 1.1 仕様に準拠）

実行を伴わない JSP 事前変換を指定するには、ブラウザで JSP ページを起動するときに、標準的な `jsp_precompile` リクエスト・パラメータを有効にします。たとえば、次のように指定します。`http://<hostname>:<port>/foo.jsp?jsp_precompile=true`

<ORACLE_HOME>`/j2ee/home/default-web-app/HelloWorld.jsp` ファイル（または相当する Windows のファイル）を例として使用し、次のディレクトリにある `_HelloWorld*` ファイルをすべて消去します。

UNIX の場合 :

```
<ORACLE_HOME>/j2ee/home/application-deployments/default/
defaultWebApp/persistence/_pages/
```

Windows の場合 :

```
<ORACLE_HOME>¥j2ee¥home¥application-deployments¥default¥
defaultWebApp¥persistence¥_pages¥
```

次に、`http://<hostname>:7777/j2ee/HelloWorld.jsp?jsp_precompile=true` という URL を指定して起動します。事前変換は行われますが、ブラウザにページは表示されません。変換されたファイルを `_pages` サブディレクトリでチェックします。

バイナリ・ファイルのみを実行する JSP コンテナの構成

所有権またはセキュリティのために JSP ページ・ソースを公開しないようにするには、JSP ページを事前変換し、変換済およびコンパイル済のバイナリ・ファイルのみをデプロイします。事前変換された JSP ページは、オンデマンド変換シナリオによる以前の実行または `ojspc` によって、どの標準的な J2EE 環境にもデプロイできます。

詳細は、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』を参照してください。

5

Enterprise JavaBeans コンポーネントの移行

この章では、WebLogic Server から Oracle Application Server に Enterprise JavaBeans コンポーネントを移行する際に必要な情報を説明します。ここでは、単純な EJB JAR の移行、および EAR ファイル形式または展開ディレクトリ形式の J2EE Web アプリケーションの移行を扱います。

この章は、次の項で構成されています。

概要

WebLogic Server から Oracle Application Server への Enterprise JavaBeans（EJB）の移行は容易であり、移行する EJB のコード変更は、ほとんどあるいはまったく必要ありません。どちらのアプリケーション・サーバーも、EJB 2.0 仕様をサポートしています。

EJB 2.0 仕様に準拠して記述および設計されている EJB はすべて正常に機能し、最小限の作業で移行を実現できます。新環境へのアプリケーションの移行に伴う主な作業は、構成とデプロイです。移行作業が複雑になるのは、独自の拡張機能が使用されている場合のみです。

この章では、EAR ファイル形式または展開ディレクトリ形式でデプロイされている EJB の移行を扱います。

WebLogic Server と Oracle Application Server の EJB 機能の比較

WebLogic Server と Oracle Application Server Containers for J2EE（OC4J）では、どちらも同じバージョンの Enterprise JavaBeans 仕様が実装されているため、コア領域については両者に相違点はありません。次の表は、どちらのアプリケーション・サーバーでも利用できる EJB 機能をまとめたものです。

表 5-1 EJB 機能の比較

機能	Oracle Application Server 10*g*	WebLogic Server 7.0
Session Bean	○	○
コンテナ管理の永続性を持つ Entity Bean（CMP）	○	○
Bean 管理の永続性を持つ Entity Bean（BMP）	○	○
Message-Driven Bean	○	○
JTA トランザクション	○	○
JCA エンタープライズ接続	○	○
IMS メッセージング	○	○
EJB スタブの動的生成	○	○
全面的に EAR ファイルをベースにしたデプロイ	○	○
EJB アプリケーションのオート・デプロイ	○	○
ステートレス EJB とステートフル EJB のクラスタリング	○	○

表 5-1 EJB 機能の比較（続き）

機能	Oracle Application Server 10*g*	WebLogic Server 7.0
Enterprise JavaBeans 向けのローカル・インタフェース	○	○
EJB 問合せ言語（EJB QL） - コードの自動生成 - Oracle および Oracle 以外のデータベースのサポート - リレーションシップを持つ CMP	○	○
RMI-over-IIOP のサポート	○	○
リレーションシップを持つ CMP	○	○
並行処理制御 - 読取り専用ロック - 即時ロック - コミット時ロック	○	○

次の項では、前述の機能の一部について詳細に説明します。

効率の優れたコンテナ管理の永続性

OC4J のコンテナ管理の永続性（CMP）の実装を使用すると、WebLogic Server の実装に比べ、次の 2 つの点でパフォーマンスが大幅に向上します。

- **変更された EJB の自動検出**: CMP を使用すると、Oracle Application Server の J2EE コンテナは、EJB が変更されているかどうかを自動的に検出し、EJB の状態をデータベースに書き込むことができます。これにより、必要な場合のみ `ejbStore` を実行します。WebLogic Server にはこのような自動検出機能がないため、`ejbStore` 操作を実行するかどうかを判定するのに WebLogic Server のコンテナが使用する `is-modified` メソッドを、ユーザーが記述する必要があります。
- **CMP の単純なデータベース・マッピングと複雑なデータベース・マッピング**: CMP を使用するとき、Oracle Application Server の J2EE コンテナは、単純な（1 対 1、1 対多の）データベース・フィールド・マッピングと、複雑な（多対多の）データベース・フィールド・マッピングの両方を効率よくサポートします。一方、WebLogic Server は単純な（1 対多の）CMP データベース・フィールド・マッピングを初歩的にサポートしているに過ぎません。たとえば、WebLogic Server では `where` 句の文字列を限定することが難しいため、不必要な全表スキャンを実行することになります。

クラスタリングのサポート

アプリケーション・サーバーのクラスタリングとは基本的に、スケーラブルで可用性の高いサービスを透過的に提供するために、連携してアクションを実行するアプリケーション・サーバー・グループを使用することです。

他社製品と比較すると、Oracle Application Server の J2EE コンテナには次の機能があります。

- **サーブレットのクラスタリング** : Oracle Application Server には、ユーザーのアプリケーションを変更せずにサーブレットをクラスタリングする機能があります。この場合に変更するのはデプロイ構成ですが、これは J2EE アプリケーションに対して透過的です。
- **クラスタリングのアーキテクチャと簡易性** : Oracle Application Server の J2EE コンテナの重要な差別化要因として、各種インスタンスのクラスタリングが簡単であることと、クラスタリングに使用されるアーキテクチャが堅牢であることがあげられます。具体的には、Oracle Application Server では、ロード・バランシングを使用した複数のサーバーが、1 台のマシンと複数台のマシンのどちらに配置されていても、1 つのクラスタ / アイランドに属する各種 OracleAS インスタンスを構成するために変更が必要な XML ファイルは 1 つです。この変更は、Application Server Control コンソールで実行できます。

  一方、1 台または複数台のマシンに複数のインスタンスを配置した状態で、ロード・バランシングを使用した WebLogic Server クラスタを構成するほうが、ずっと複雑です。たとえば、EJB をクラスタ内で使用するように定義する場合は、`ejbc` を使用して EJB スタブが作成される際にこれを指定する必要があります。これにより、デプロイに使用する特殊なクラスタ対応のクラスが作成されます。全体として、Oracle Application Server の J2EE コンテナを、Oracle Application Server の他のコンポーネントと併用することにより、より堅牢で使いやすいクラスタリング・アーキテクチャが実現されます。
- **ステートレス Session Bean のクラスタリング** : Oracle Application Server は、ステートレス Session Bean のクラスタリングをサポートしています。
- **ステートフル Session Bean および Entity Bean のクラスタリング** : Oracle Application Server は、ステートフル Session Bean および Entity Bean のクラスタリングをサポートしています。設計では、次の 2 つに重点が置かれます。
  - クラスタリングによるパフォーマンス : WebLogic Server などの従来のクラスタリング機能では、クラスタリングでインスタンスをステートフルに実行すると、サーバーのパフォーマンスが低下します。このため、アプリケーション開発者は通常、中間層を完全にステートレスにして、その状態をデータベースなどの永続的な記憶域に書き込みます。クラスタリングされた EJB の配布でパフォーマンスが低下しないように、オラクル社は EJB クラスタリングの実装の最適化に取り組んでいます。
  - プログラムの簡易性 : 状態をフェイルオーバーする固有のセッション境界を持つサーブレットとは異なり、EJB には明確な境界がありません。このため、Oracle Application Server には単純なプログラム機能が用意されており、開発者はアプリケーションを変更せずに EJB のクラスタリングを使用できます。

スケーラビリティとパフォーマンスの拡張機能

- **Entity Bean のスケーラビリティ**: Oracle Application Server では、主キー値ごとに Bean ラッパー・インスタンスの構成可能プールを使用し、複数のクライアントが同一の Entity Bean インスタンスのメソッドを同時に検索およびコールでき、それによって Entity Bean のスケーラビリティが向上しました。
- **並行処理制御の向上**: Oracle Application Server では、次の並行処理制御オプションが新たに導入され、大規模な J2EE アプリケーションのスケーラビリティとパフォーマンスが向上しました。
  - 読取り専用ロック: データベースを更新しない読取り専用 Bean の場合、Bean 開発者は OC4J コンテナに対して、`ejbStore()` のコールおよび生成を行わないように指示できます。Bean の状態を、EJB 以外に SQL を使用するアプリケーションなどの外部システムによって更新できるかどうかに応じて、適切な分離モードが選択されます。
  - 即時ロック: Oracle Application Server は、決定的なタイムアウトおよびデッドロックの検出のために、専用の Bean インスタンスを各クライアントに提供しながら、Bean 状態に順次アクセスできます。
  - コミット時ロック: Oracle Application Server では別のロック構造もサポートします。このロック構造は行ロックを使用しないため、データ整合性は Bean の分離モード（「反復不可能読取り」または「シリアライズ可能」）と、クライアントが行を更新する順序に依存します。

WebLogic Server にも同様の機能があります。

セキュリティと LDAP の統合

OC4J のきわめて特徴的な機能の 1 つに、Single Sign-On（SSO）および Oracle Internet Directory（OID）とのシームレスな統合があります。この統合は、Java Authentication and Authorization Service（JAAS）規格の、オラクル社による実装を通じて実現します。『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』および『Oracle Application Server 10*g* セキュリティ・ガイド』を参照してください。

WebLogic Server の注意事項

WebLogic Server での EJB の実装について、その他の注意事項を次に示します。

- WebLogic Server での BMP セキュリティの実装は、J2EE 仕様に完全には準拠していません。仕様に従うと、BMP セキュリティ・ロール権限に違反があったときに、例外をスローする必要があります。OC4J は仕様に従って例外をスローしますが、WebLogic Server はスローしません。
- WebLogic Server と異なり、OC4J の場合、開発者は、WebLogic Server `weblogic-ejb-jar.xml` など、EJB デプロイ用に独自の XML ファイルを作成する必要がありません。OC4J の場合は、内部処理用に OC4J コンテナにより `orion-ejb-jar.xml` が作成されます。開発者は必要に応じてこのファイルを変更できますが、ファイルの作成は不要です。したがって、OC4J のほうが EJB のデプロイ・プロセスは単純です。

EJB の移行に関する考慮事項

EJB を使用する際の目標の 1 つは、様々な環境におけるコンポーネントの移植性をソース・コード・レベルのみならず、バイナリ・レベルにおいても提供することです。これにより、コンパイルおよびパッケージ化されたコンポーネントの移植性が確保されます。EJB では、確かに移植性は実現しますが、実装に依存する移植不可能な機能がいくつかあり、プラットフォーム間でコンポーネントを移植するときには、この問題に対処する必要があります。一般に、EJB コンポーネントでは、コンテナとの低レベルのインタフェースが必要とされます。これは、スタブ・クラス形式およびスケルトン・クラス形式により、クラスは実装に依存させる必要があります。EJB コンポーネントの移植可能な要素と移植不可能な要素は、実際には明確に区別できます。

移植可能な EJB 要素には、次のものがあります。

- 実際のコンポーネントの実装クラスとインタフェース（Bean クラス、リモート・インタフェースおよびホーム・インタフェース）。
- JNDI 名やトランザクション属性など、汎用コンポーネントのプロパティを記述するアセンブリおよびデプロイメント・ディスクリプタ。
- セキュリティ属性。

実装に依存する要素には、次のものがあります。

- ホスト・コンテナとの対話に使用する、低レベルの実装のヘルパー・クラス（スタブおよびスケルトン）。
- CMP Entity Bean のオブジェクト・リレーショナル・マッピング定義。たとえば、実装に依存する形式で宣言された、プラットフォーム独自のカスタム・ファインダ・メソッド用の検索ロジックがあります。
- 各コンポーネントには、デプロイ時に系統だった構成を必要とするプロパティ・セットが割り当てられています。たとえば、EJB コンポーネントで実際のユーザーやグループに対して宣言されたセキュリティ・ロールのマッピングは、デプロイ時に系統的に実行されるタスクです。その理由は、マッピングが事前に認識されていない可能性があるためです。また、ターゲットのデプロイメント・サーバー上のユーザー・ディレクトリの構造と要素に依存する場合もあります。

移行手順

EJB の移行に伴う作業について検討するには、EJB コンテナに EJB をデプロイするために必要な次の手順を調べることが最良の方法です。

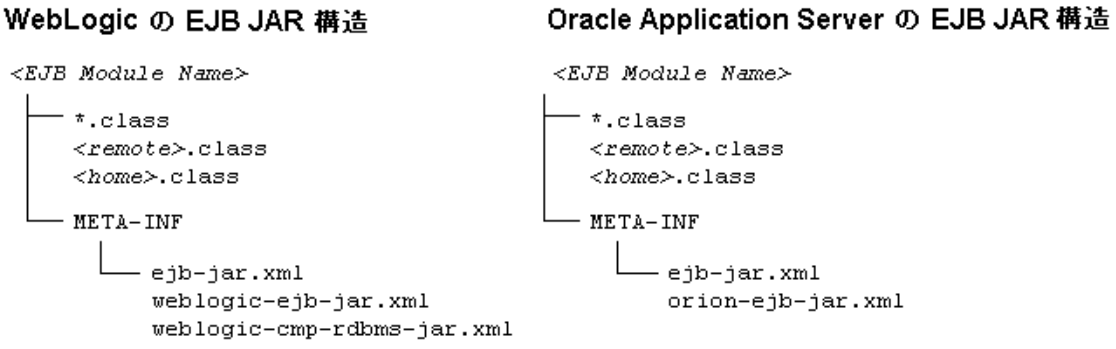
- EJB デプロイメント・ディスクリプタ（特にベンダー固有のデプロイメント・ディスクリプタ）の設定
- EJB コンテナ・クラスの生成
- EJB クラスのサーバーへのロード
- EAR ファイル形式または展開ディレクトリ形式による EJB のデプロイ
- 起動時にデプロイする EJB の構成

移行タスクも同じ方針で対処できます。

デプロイ・プロパティの設定

デプロイ・プロセスは、JAR ファイルまたは J2EE の標準デプロイ・ディレクトリから始めます。これらのファイルまたはディレクトリには、EJB プロバイダで作成されたコンパイル済の EJB インタフェースと実装クラスが含まれます。この中には、バンドルされている EJB を記述する `ejb-jar.xml` ファイルも含まれます。`ejb-jar.xml` ファイルなど、必要な XML のデプロイ・ファイル（通常はベンダー固有のデプロイメント・ディスクリプタ）は、次のように、JAR ファイルまたはデプロイ・ディレクトリの最上位の `META-INF` ディレクトリに格納されている必要があります。

図 5-1 EJB の JAR ファイルの内容と構造



ベンダー固有のデプロイメント・ディスクリプタ

各アプリケーション・サーバーは、J2EE の標準デプロイメント・ディスクリプタのほかに、固有のデプロイメント・ディスクリプタを備えています。

WebLogic Server 最初に WebLogic Server 固有で必須のデプロイメント・ディスクリプタ `weblogic-ejb-jar.xml` を作成および構成し、次にこのファイルをデプロイ・ファイルまたはディレクトリに追加します。`weblogic-ejb-jar.xml` ファイルは、キャッシュ、クラスタリングおよびパフォーマンスの動作の指定に使用します。

コンテナ管理の永続性を使用している Entity EJB をデプロイする場合は、オブジェクト・リレーショナル・マッピングの詳細を指定するために、別のデプロイ・ファイルも含めます。ここで言う詳細とは、`weblogic-cmp-rdbms-jar.xml` というファイルに格納されている RDBMS ベースの永続性サービスのことです。RDBMS の永続性を使用する Bean ごとに、別々のファイルが必要です。

OC4J OC4J の場合には、必要なファイルは 1 つのみです。Application Server Control コンソールで、OC4J 固有で必須のデプロイメント・ディスクリプタ `orion-ejb-jar.xml` を作成し、デプロイ・ファイルまたはディレクトリに追加します。`orion-ejb-jar.xml` ファイルは、キャッシュ、クラスタリングおよびパフォーマンスの動作の定義に使用します。オブジェクト・リレーショナル・マッピングまたは RDBMS ベースの永続性サービスの詳細も、`orion-ejb-jar.xml` ファイルで指定します。これは、2 つのファイルを必要とする WebLogic Server とは異なります。

EJB コンテナ・クラスの生成とデプロイ

EJB クラスをコンパイルして、必要な XML デプロイメント・ディスクリプタ（J2EE のデプロイメント・ディスクリプタおよびベンダー固有のデプロイメント・ディスクリプタ）を追加したら、その次には、EJB へのアクセスに使用するコンテナ・クラスを生成します。このコンテナ・クラスには、EJB を内部で表すために、アプリケーション・サーバーが使用するクラスおよび、クライアントが使用する（ホームおよびリモートの）外部インタフェースの実装が含まれます。

WebLogic Server

WebLogic Server の場合は、`ejbc` コンパイラを使用し、WebLogic Server 固有の XML デプロイ・ファイルに指定されているデプロイ・プロパティに応じて、コンテナ・クラスを生成します。たとえば、EJB をクラスタ内で使用するように定義する場合は、`ejbc` は、デプロイに使用する特殊なクラスタ対応のクラスを作成します。必要なオプションおよび引数を指定して、コマンドラインで直接 `ejbc` を使用することもできます。

コンテナ・クラスが生成されたら、そのクラスを JAR ファイルまたは EAR ファイルにパッケージし、コンソール GUI を使用してデプロイする必要があります。

OC4J

OC4J の場合は、明示的にコンパイルする必要はありません。EJB の JAR ファイルは EAR ファイルに（WAR ファイルがあればそれと一緒に）パッケージ化されます。これで、Application Server Control コンソールの GUI を使用して、デプロイする EAR ファイルを指定できるようになります。OC4J に対するコンテナ・クラスが生成され、EAR ファイルの J2EE Web アプリケーションがあれば、OC4J コンテナにバインドされます。EAR ファイルのデプロイには、Application Server Control コンソールのかわりに `dcmctl` コマンドを使用することもできます。詳細は、『Distributed Configuration Management リファレンス・ガイド』を参照してください。

EJB クラスのサーバーへのロード

この項では、各アプリケーション・サーバーが EJB クラスのロードを管理する方法について説明します。

WebLogic Server

EJB のデプロイ手順の最後として、生成されたコンテナ・クラスを WebLogic Server にロードします。ただし、WebLogic Server を起動し、WebLogic Server に EJB クラスを自動的にロードさせることもできます。この場合、サーバーが起動すると、自動的にデプロイされるデプロイ・ディレクトリに EJB が配置されます。

OC4J

同様に、`server.xml` の `<application>` タグに `auto-start="true"` パラメータを指定して、アプリケーションに定義されているクラスが、OC4J の起動時にロードされるように指定することができます。

EAR ファイル形式または JAR ファイル形式による EJB の移行

WebLogic Server にデプロイされた EJB が含まれている EAR ファイルと JAR ファイルを、Oracle Application Server に移行することができます。ただし、EAR ファイルをいったん復元して再度アーカイブし、ファイルの内容が完全で、XML ディスクリプタのエントリが正しいことを確認する必要があります。ガイドラインとして次の点に注意してください。

- EJB クライアントの XML ディスクリプタが、EJB スタブの JNDI 名を指定していることを確認します。クライアントが Web アプリケーションの場合、JNDI 名は `web.xml` で指定します。クライアントがスタンドアロンの場合、JNDI 名は `application-client.xml` で指定します。
- EJB クライアントがスタンドアロンの場合、クライアント・クラスおよび XML ディスクリプタ・ファイル `application-client.jar` を JAR ファイルにアーカイブし、そのファイルを対象の EJB と同じ EAR ファイルにアーカイブします。

- WebLogic から移行する EJB が JAR ファイル形式の場合は、EAR の `application.xml` で、これらの EJB を再度 EAR ファイルにパッケージ化する必要があります。
- Application Server Control コンソールまたは `dcmctl` を使用して、EAR ファイルを Oracle Application Server にデプロイします。
- `ejbc` や `rmic` などの機能を使用して、EJB スタブをクライアント・アプリケーションにプリコンパイルする必要はありません。OC4J の EJB コンテナでは、必要に応じて EJB スタブが生成されます。

展開 EJB アプリケーションの移行

EJB アプリケーションのデプロイは、J2EE 仕様で定義された標準ディレクトリ構造を使用するファイルの集合として実行することもできます。このようなデプロイでは、アプリケーションは展開ディレクトリ形式でデプロイされます。展開ディレクトリ形式による EJB アプリケーションのデプロイは、主にアプリケーションの開発時に、スタンドアロンの OC4J インスタンスに対してのみ行われます。これは、開発者がソース・ファイルを変更したり、アプリケーションを短期間でテストするには、展開ディレクトリのほうが適しているためです。ただし、Oracle Application Server の本番環境では、アプリケーションを EAR ファイルにパッケージ化し、Application Server Control コンソールまたは `dcmctl` を使用してデプロイします。

展開ディレクトリ構造を WebLogic Server にデプロイする場合は、展開ディレクトリ形式の EJB アプリケーションが格納されている最上位のディレクトリを、配布する WebLogic Server の `mydomain/config/applications/` ディレクトリにコピーします（`mydomain` は WebLogic Server のドメイン名）。コピーが済んだら、WebLogic Server によって、EJB アプリケーションが自動的にデプロイされます。

OC4J の場合は、展開ディレクトリ形式の EJB アプリケーションが格納されている最上位のディレクトリを、OC4J インストールの中で、次のディレクトリにコピーします。

UNIX の場合 :
<*ORACLE_HOME*>`/j2ee/home/applications/`

Windows の場合 :
<*ORACLE_HOME*>`¥j2ee¥home¥applications¥`

次に、<*ORACLE_HOME*>`/j2ee/home/config/` ディレクトリ（UNIX の場合）、または <*ORACLE_HOME*>`¥j2ee¥home¥config¥` ディレクトリ（Windows の場合）にある、J2EE アプリケーションのデフォルトのデプロイメント・ディスクリプタ `server.xml` を変更して、EJB モジュールが含まれるようにします。

WebLogic Server では、ファイルの変更に管理コンソールを使用した場合も使用しない場合も、サーバーを再起動しないと、更新された設定は有効になりません。OC4J の場合は、`server.xml` のタイムスタンプが変更されていると、OC4J がその変更内容を XML ファイルに反映させます。

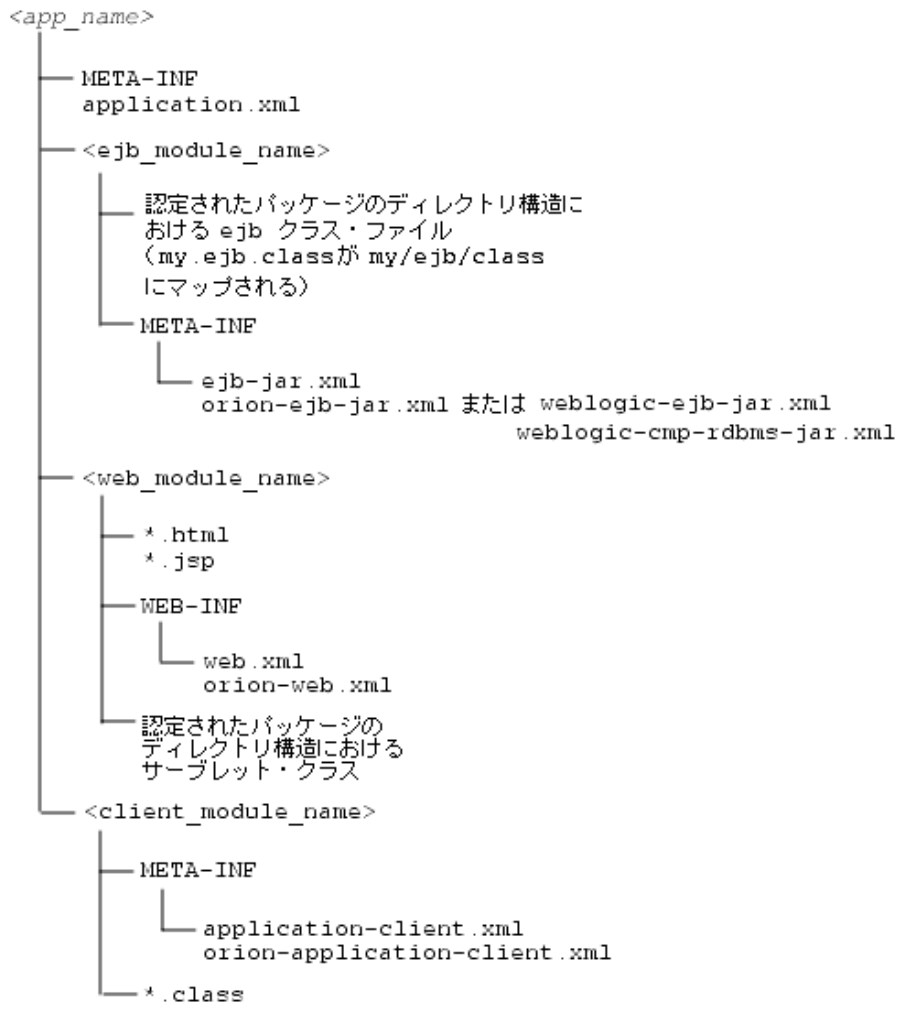
デプロイメント・ディスクリプタを使用した EJB の構成

EJB の構成とデプロイには、2 つのデプロイメント・ディスクリプタを使用します。1 つは `ejb-jar.xml` で、これは EJB 仕様で定義されており、EJB アプリケーションを記述するための標準化された形式です。もう 1 つはベンダー固有のデプロイメント・ディスクリプタで、`ejb-jar.xml` ファイルに定義されているリソースを、アプリケーション・サーバー内のリソースにマップします。このデプロイメント・ディスクリプタを使用して、EJB の動作、キャッシュ、ベンダー固有の機能など、その他の EJB コンテナの特徴を定義することもできます。

WebLogic Server 固有のデプロイメント・ディスクリプタは `weblogic-ejb-jar.xml` と `weblogic-cmp-rdbms-jar.xml` で、OC4J 固有のデプロイメント・ディスクリプタは `orion-ejb-jar.xml` です。

J2EE アプリケーションの一般的なディレクトリ構造は、次のようになります。

図 5-2 J2EE アプリケーションのディレクトリ構造



WebLogic Server 固有のデプロイメント・ディスクリプタ weblogic-ejb-jar.xml は、EJB デプロイメント・ディスクリプタの DTD を定義します。この DTD は WebLogic Server で一意です。weblogic-ejb-jar.xml の DTD には、ステートフル Session EJB のレプリケーションの有効化、Entity EJB のロック動作の構成、および Message-Driven Bean の JMS キューとトピック名の割当てのための要素が含まれます。

EJB の weblogic-ejb-jar.xml に構成されている要素は、次のとおりです。

- weblogic-enterprise-bean
  - ejb-name
  - entity-descriptor
  - stateless-session-descriptor
  - stateful-session-descriptor
  - message-driven-descriptor
  - transaction-descriptor
  - reference-descriptor
  - enable-call-by-reference
  - jndi-name
- Security-role-assignment
- transaction-isolation

WebLogic Server 固有のデプロイメント・ディスクリプタ weblogic-cmp-rdbms-jar.xml は、WebLogic Server の RDBMS ベースの永続性サービスを使用する Entity EJB に対して、デプロイのプロパティを定義します。

weblogic-cmp-rdbms-jar.xml はそれぞれ、次の永続性オプションを定義します。

- CMP 用の EJB 接続プールまたはデータソース
- EJB のフィールド / データベース / 要素間のマッピング
- リレーションシップに対する外部キーのマッピング
- 問合せ用の WebLogic Server 固有のデプロイメント・ディスクリプタ

OC4J 固有のデプロイメント・ディスクリプタ orion-ejb-jar.xml には、Session Bean、Entity Bean、Message-Driven Bean およびセキュリティの詳しいデプロイ情報が含まれています。

Entity EJB は、トランザクション型または非トランザクション型の永続ストレージに状態を保存する（Bean 管理の永続性）ことも、コンテナに対して一時型でないインスタンス変数を自動的に保存するように要求する（コンテナ管理の永続性）こともできます。WebLogic Server および OC4J では、これらの 2 つの方法、およびこれらを併せた方法を使用することができます。

コンテナ管理の永続性を使用している EJB では、`weblogic-ejb-jar.xml` または `orion-ejb-jar.xml` というデプロイメント・ディスクリプタ・ファイルによって、EJB が使用する永続性サービスのタイプが指定されます。WebLogic Server の場合、自動的な永続性サービスでは、デプロイメント・ディスクリプタを指定し、Entity EJB のファインダ・メソッドを定義するために、これ以外のデプロイ・ファイルも使用する必要があります。WebLogic Server の RDBMS ベースの永続性サービスは、Bean の `weblogic-cmp-rdbms-jar.xml` ファイルを使用して、特定の Bean からデプロイメント・ディスクリプタおよびファインダの定義を取得します。この構成ファイルは、`weblogic-ejb-jar.xml` ファイル内で参照する必要があります。OC4J の場合は、同じデプロイメント・ディスクリプタ `orion-ejb-jar.xml` から、永続性サービスのタイプおよび RDBMS ベースの永続性サービスの詳細が構成され取得されます。

Development Mode など、一部の属性は OC4J で一意です。

> **関連項目：** 属性の詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

RDBMS の永続性に対するファインダの作成

RDBMS の永続性を使用する EJB に対して、WebLogic Server では動的なファインダを作成できます。EJB プロバイダは、`EJBHome` インタフェース内にファインダのメソッド・シグネチャを記述し、そのファインダの問合せ式を `ejb-jar.xml` デプロイ・ファイルで定義します。`ejbc` コンパイラは、`ejb-jar.xml` 内の問合せを使用して、デプロイ時にファインダ・メソッドの実装を作成します。

RDBMS の永続性に対するファインダの主なコンポーネントは、次のとおりです。

- `EJBHome` のファインダ・メソッド・シグネチャ
- `ejb-jar.xml` 内で定義されている query スタンザ
- `weblogic-cmp-rdbms-jar.xml` 内の WebLogic Server の query スタンザ（省略可能）

OC4J では、ファインダ・メソッドを自動生成することによって、プロセス全体が簡略化されます。

具体的には、OC4J では `findByPrimaryKey` メソッドの指定が簡単にできます。単純な主キーまたは複雑な主キーを定義するフィールドはすべて、`ejb-jar.xml` デプロイメント・ディスクリプタ内で指定されます。CMP の Entity Bean で、その他のファインダ・メソッドを定義する手順は次のとおりです。

1. ファインダ・メソッドをホーム・インタフェースに追加します。
2. ファインダ・メソッドの定義を、OC4J 固有のデプロイメント・ディスクリプタ、すなわち `orion-ejb-jar.xml` ファイルに追加します。

WebLogic 問合せ言語（WLQL）と EJB 問合せ言語（EJB QL）

WebLogic Server の 5.1 と 6.0 では、`weblogic-cmp-rdbms-jar.xml` ファイルの各 finder-query スタンザに、EJB を返すための問合せを定義する WLQL 文字列を含める必要がありました。これらのリリースの WebLogic Server は EJB 1.1 コンテナを実装し、標準化された EJB QL はサポートしていませんでした。

その後、EJB 2.0 仕様をベースとする標準の EJB 問合せ言語が使用されるようになると、WLQL は使用されなくなりました。WebLogic Server 7.0 の EJB コンテナは EJB 2.0 に準拠し、EJB QL をサポートします。この EJB コンテナには、EJB QL への WLQL 拡張機能も用意されています。これは、WebLogic Server 独自の拡張機能です。

Oracle Application Server では、次の機能を持つ EJB QL が完全にサポートされています。

- コードの自動生成 : EJB QL の問合せは、Entity Bean のデプロイメント・ディスクリプタに定義されます。EJB が Oracle Application Server にデプロイされると、コンテナはその問合せを、ターゲットのデータ・ストアの SQL 言語に自動変換します。この変換のため、コンテナ管理の永続性を持つ Entity Bean は移植可能となり、コードは特定のタイプのデータ・ストアに拘束されません。
- SQL コードの生成の最適化 : さらに、SQL コードの生成では、データベースへのアクセスの効率化を目的として、バルク SQL や SQL 文の一括ディスパッチの使用など、Oracle Application Server にはいくつかの最適化機能があります。
- Oracle および Oracle 以外のデータベースのサポート : Oracle Application Server では、Oracle、MS SQL-Server、IBM DB/2、Informix、Sybase など、あらゆるデータベースに対して EJB QL を実行できます。
- リレーションシップを持つ CMP: Oracle Application Server では、単一の Entity Bean、およびリレーションシップを持つ複数の Entity Bean に対して EJB QL がサポートされます。またあらゆるタイプの多重性および方向性がサポートされます。

Oracle Application Server での EJB QL の詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

Message-Driven Bean

WebLogic Server では、新しい `ejb-jar.xml` 要素のほかに、Message-Driven Bean を WebLogic Server 内の実際の宛先に関連付けるために、1 つの新しい message-driven-descriptor スタンザのみを `weblogic-ejb-jar.xml` ファイルに含める必要があります。この XML 要素は `destination-jndi-name` です。

OC4J で Message-Driven Bean を作成する手順は次のとおりです。

1. EJB 仕様の定義に従い、Message-Driven Bean を実装します。
2. Message-Driven Bean のデプロイメント・ディスクリプタを作成します。
3. OC4J の JMS XML ファイル `jms.xml` で、JMS の `Destination` タイプ（キューまたはトピック）を構成します。
4. OC4J 固有のデプロイメント・ディスクリプタ `orion-ejb-jar.xml` で、JMS の `Destination` タイプを Message-Driven Bean にマップします。
5. Message-Driven Bean のアプリケーションにデータベースが関わる場合は、`data-sources.xml` でデータベースを表すデータソースを構成します。
6. Bean およびデプロイメント・ディスクリプタを含む EJB の JAR ファイルを作成します。このファイルを作成したら、`application.xml` ファイルを構成し、EAR ファイルを作成して EJB を OC4J にデプロイします。

セキュリティの構成

セキュリティは、アプリケーション・サーバーで対処することも、プログラムによって EJB クラスに組み込むこともできます。WebLogic Server と OC4J では、認証、認可、デジタル証明など、セキュリティについて類似したサポートを提供しています。

> **関連項目：** 『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』の「セキュリティの構成」。

クラスタ対応の EJB アプリケーションの OC4J への移行

Oracle Application Server には、パフォーマンスと使い勝手の両面で、WebLogic Server を凌駕するクラスタリング機能があります。また、WebLogic Server から OC4J には、クラスタ対応アプリケーションを容易に移行できます。

WebLogic Server における EJB のクラスタリング

この項では、WebLogic Server による EJB のクラスタリング方法について説明します。

ステートフル Session EJB に対するメモリー内レプリケーション

WebLogic Server の EJB コンテナでは、クラスタリングされた WebLogic Server インスタンス間で、EJB の状態をレプリケートすることができます。

ステートフル Session EJB のレプリケーションは、EJB のクライアントに認識されずにサポートされます。ステートフル Session EJB がデプロイされると、WebLogic Server はステートフル Session EJB に対してクラスタ対応の `EJBHome` スタブおよびレプリカ対応の `EJBObject` スタブを作成します。`EJBObject` スタブには、EJB インスタンスが実行されるプライマリ WebLogic Server インスタンスと、Bean の状態のレプリケートに使用するセカンダリ WebLogic Server の名前のリストがあります。

EJB のクライアントが EJB の状態を変更するトランザクションをコミットするたびに、WebLogic Server はセカンダリ・サーバーのインスタンスに、Bean の状態をレプリケートします。クラスタリングされている環境で最適なパフォーマンスを得るために、Bean の状態のレプリケーションはメモリー内で直接行われます。

プライマリ・サーバーのインスタンスに障害が発生した場合は、クライアントの次のメソッド・コールが、セカンダリ・サーバー上の EJB インスタンスに自動的に転送されます。セカンダリ・サーバーは、このインスタンスに対するプライマリ WebLogic Server となり、これ以降に発生するフェイルオーバーには、新しいセカンダリ・サーバーを使用します。EJB のセカンダリ・サーバーに障害が発生した場合、WebLogic Server は、クラスタから新しいセカンダリ・サーバーのインスタンスを確保します。

ステートフル Session EJB の状態をレプリケートすることによって、プライマリ WebLogic Server のインスタンスに障害が発生した場合でも、多くの場合クライアントは最後にコミットされた EJB の状態を保証できます。ただし、ごくまれなフェイルオーバーのケースとして、最後にコミットされた状態が入手できないことがあります。この事象が発生する場合として考えられるものは次のとおりです。

- クライアントが、ステートフル EJB を使用するトランザクションをコミットしたが、EJB の状態がレプリケートされる前にプライマリ WebLogic Server に障害が発生した場合。この場合には、クライアントの次のメソッド・コールは、（入手可能であれば）1 つ前にコミットされた状態に対して機能します。
- クライアントがステートフル Session EJB のインスタンスを作成し、最初のトランザクションをコミットしたが、EJB の最初の状態がレプリケートされる前にプライマリ WebLogic Server に障害が発生した場合。この場合には、最初の状態がレプリケートされていなかったため、クライアントの次のメソッド・コールで、Bean インスタンスを見つけることができません。クライアントは、クラスタリングされた `EJBHome` スタブを使用して EJB インスタンスを再作成し、トランザクションを再開する必要があります。

- プライマリ・サーバーとセカンダリ・サーバーの両方に障害が発生した場合。この場合には、クライアントは EJB インスタンスを再作成し、トランザクションを再開する必要があります。

要件と構成

WebLogic Server のクラスタでステートフル Session EJB の状態をレプリケートするには、そのクラスタが EJB クラスに対して同種となるようにします。すなわち、クラスタ内のあらゆる WebLogic Server インスタンスについて、同じデプロイメント・ディスクリプタを使用して同じ EJB クラスをデプロイします。異種のクラスタに対しては、メモリー内のレプリケーションはサポートされません。

デフォルトでは、WebLogic Server はクラスタ内のステートフル Session EJB インスタンスの状態をレプリケートしません。レプリケーションを実行できるようにするには、`weblogic-ejb-jar.xml` デプロイ・ファイルのレプリケーション・タイプのデプロイ・パラメータを、`InMemory` に設定します。次に例を示します。

```
<stateful-session-clustering>
...
...
...
<replication-type>InMemory</replication-type>
</stateful-session-clustering>
```

Oracle Application Server における EJB のクラスタリング

Oracle Application Server における EJB のクラスタリングには、EJB のロード・バランシングとフェイルオーバーの機能があります。Oracle Application Server では、HTTP セッションのロード・バランシングおよびフェイルオーバーとは異なるメカニズムを使用して、これらの機能を実現しています。EJB の場合、EJB クライアントのスタブによってロード・バランシングがリダイレクトされ、クラスタ・アイランドを使用せずにフェイルオーバーの状態がレプリケートされます（今後のリリースの Oracle Application Server では、EJB のクラスタ・アイランドを実装する予定です）。

EJB クラスタを作成するには、どの OC4J ノードがクラスタに含まれるかを指定し、そのそれぞれのノードに対して同一のマルチキャスト・アドレス、ユーザー名およびパスワードを設定する必要があります。クラスタリングされる EJB は、これらの各ノードにデプロイできます。クラスタ内のすべてのノードを、同じマルチキャスト・ユーザー名とパスワードで構成すると、ユーザー名とパスワードの 1 つの組合せを使用して、すべてのノードを認証することができます。同じマルチキャスト・アドレスで別のユーザー名とパスワードの組合せを使用した場合は、別のクラスタが実際に定義されます。Application Server Control コンソールでは、ユーザー・インタフェースを使用してマルチキャストのユーザー名とパスワードを指定できます。

ロード・バランシング

EJB のロード・バランシングは、EJB のクライアント側で行われます。クライアントのスタブは、静的検出と動的検出のいずれかの方法で、クラスタ内のノードのアドレスを取得します。あるクラスタ内のすべてのノードがわかったら、クライアントのスタブはその中の 1 つをランダムに選択します。ロード・バランシングはランダムな方法で実行されます。

静的検出と動的検出は、次のように実行されます。

静的検出 参照時に、クラスタ内のすべてのノードの JNDI アドレスが、参照用 URL のプロパティに表示されます。この方法では、各ノードのノード名と `ormi` ポートを、あらかじめわかっている必要があります。次に例を示します。

```
java.naming.provider.url = ormi://serverA:23791/ejb, ormi://serverB:23792/ejb,
                           ormi://serverC:23791/ejb;
```

動的検出 動的検出では、最初の参照時に、アクセスされた最初のノードが、同じマルチキャスト・アドレスおよびユーザー名とパスワードの組合せを持つその他のノードと通信します。これらのノードの `ormi` アドレスが取得され、クライアントのスタブに返されます。クライアントのスタブでは、これらのアドレスの 1 つをランダムに選択します。動的検出を実行できるようにするには、`ormi` の URL の前に `lookup:` を挿入します。

```
ic.lookup("lookup:ormi://serverA:23791/ejb");
```

フェイルオーバー

クラスタリングされる EJB のタイプに応じて、EJB クラスタのフェイルオーバーは、リクエストのリダイレクションおよび状態のレプリケーションによって実行されます。

ステートレス Session EJB ステートレス Session EJB のロード・バランシングおよびフェイルオーバーは、ノードが静的または動的に検出された後で、ランダムに選択されたノードに対して、EJB クライアントのスタブがリクエストをリダイレクトすることによって実行されます。EJB がステートレスであるため、Bean の状態をレプリケートする必要はありません。

ステートフル Session EJB ステートフル Session EJB のロード・バランシングは、ステートレス Session EJB の場合と同じです。フェイルオーバーについては、状態のレプリケーションが必要です。デフォルトでは、各 EJB インスタンスに対するすべてのメソッド・コールが終了した際に、クラスタ内のすべてのノードに対して状態がレプリケートされます。この方法は信頼性の高いものですが、すべてのノードで CPU のオーバーヘッドが大量に発生し、パフォーマンスが低下することは間違いありません。このため、パフォーマンスをあまり低下させずにレプリケートできるように、JVM の終了およびステートフル・セッション・コンテキストという 2 つのレプリケーション・モードが用意されています。

JVM の終了モードは、EJB を実行する JVM が正常に終了した際に、すべてのステートフル Session EJB の状態を、他のいずれかのノードにレプリケートします。このレプリケーションのロジックでは、JDK の終了フックを使用します（したがって、JDK1.3 以上が必要です）。このモードは、レプリケーションが 1 回しか実行されないため、他のモードに比べてパフォーマンスの低下が最も少なくて済みます。ただし、正常にシャットダウンできるかどうかは JVM の機能に依存するため、信頼性は低くなります。

ステートフル・セッション・コンテキスト・モードは、プログラムによって状態をレプリケートします。レプリケートする情報を指定するために、OC4J 独自のクラス `com.evermind.server.ejb.statefulSessionContext` が用意されています。この情報を `setAttribute` メソッドのパラメータとして設定すると、EJB クラスタ内のすべてのノードにこの情報をレプリケートできます。このため、EJB プロバイダでは、レプリケートするタイミングと対象をさらに詳しく設定できます。

Entity EJB Entity EJB のレプリケーションでは、EJB の状態をデータベースに格納できます。Entity EJB の状態が変わると、データベースでも更新されます。状態が変わった Entity EJB は、その Entity EJB が最新でないことをクラスタ内の他のノードに通知します。最新の EJB をホスティングしているノードに障害が発生すると、クライアントのスタブが別のノードにリダイレクトし、そのノードで最新でなくなった Entity EJB は、データベースの情報を使用して状態をもう一度同期化します。

> **関連項目：** EJB クラスタリングの構成方法の詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

JNDI ネームスペースのレプリケーション

EJB のクラスタリングが有効なときは、JNDI ネームスペースのレプリケーションも、EJB クラスタ内でロールを有する OC4J インスタンスの間で有効になります。ある OC4J インスタンスの JNDI ネームスペースに新しくバインドすると、その EJB クラスタ内のその他の OC4J インスタンスに伝播されます。再バインドとアンバインドはレプリケートされません。

JNDI レプリケーションは、OC4J アイランドの有効範囲外で行われます。言い換えれば、1 つの OC4J インスタンスに入っている複数のアイランドでは、レプリケートされた同じ JNDI ネームスペースを参照できます。詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

6

JDBC の移行

この章では、WebLogic Server から Oracle Application Server にデータベース・アクセス・コードを移行する際に必要な情報を説明します。ここでは、JDBC ドライバ、データソースおよび接続プーリングの移行を扱います。

この章は、次の項で構成されています。

概要

WebLogic Server にデプロイされた JDBC を使用するアプリケーション、特に WebLogic JDBC ドライバを OC4J および Oracle JDBC ドライバに移行する作業は容易であり、移行するアプリケーションのコード変更は、ほとんどあるいはまったく必要ありません。どちらのアプリケーション・サーバーも、JDBC API については同じ API レベルをサポートします。つまりバージョン 2.0 の仕様が完全にサポートされます。JDBC の標準仕様に準拠して作成されているアプリケーションはすべて正常に機能し、最小限の作業で移行を実現できます。新環境へのアプリケーションの移行に伴う主な作業は、構成とデプロイです。移行作業が複雑になるのは、独自の拡張機能が使用されている場合のみです。

データベース・アクセスの実装における WebLogic と Oracle Application Server の相違点

WebLogic Server と OC4J は、どちらも J2EE 1.3 に完全に準拠したコンテナを使用しています。このコンテナによって、各種のデータベースへのアクセスに、すべてのタイプの JDBC ドライバを使用することができます。さらに、BEA 社とオラクル社の JDBC ドライバは、同じバージョンの JDBC 規格（バージョン 2.0 仕様）をサポートします。このため、2 つのサーバーの相違点はごくわずかであり、その相違点は主に、独自の拡張機能という領域で発生します。相違点を分析する前に、JDBC ドライバの概要を見ていきます。

JDBC ドライバの概要

JDBC では、指定した JDBC ドライバへの標準 API のコールが定義されます。JDBC ドライバは、データ・インタフェース用のコマンドが実際に実行されるソフトウェアです。このドライバは、低レベルの JDBC API として位置付けられています。ドライバとのやり取りには、データベースのクライアント・コール、またはデータベース・サーバーによってサービスされるデータベースのネットワーク・プロトコル・コマンドを使用します。

JDBC API コールを変換する JDBC ドライバは、そのインタフェース型に応じて、次の 4 種類に分類されます。

- **タイプ 1、JDBC-ODBC Bridge**: JDBC API コールを ODBC API コールに変換します。
- **タイプ 2、ネイティブ API ドライバ**: JDBC API コールをデータベースのネイティブ API コールに変換します。このドライバではネイティブ API が使用されるため、ベンダーに依存します。ドライバは、2 つのコンポーネントで構成されます。変換が行われる Java 言語のコンポーネントとネイティブ API のライブラリ・セットを含むコンポーネントです。
- **タイプ 3、ネットワーク・プロトコル**: JDBC API コールを DBMS に依存しないネットワーク・プロトコル・コールに変換します。データベース・サーバーは、このネットワーク・プロトコル・コールを特定の DBMS 操作に変換します。

- **タイプ 4、ネイティブ・プロトコル**: JDBC API コールを DBMS 固有のネットワーク・プロトコル・コールに変換します。データベース・サーバーは、このコールを DBMS 操作に変換します。

BEA 社では、JDBC API 仕様を使用したデータベース・アクセスのために、様々なオプションを用意しています。オプションには、Oracle 用、Microsoft SQL Server 用および Informix データベース管理システム（DBMS）用の WebLogic jDriver などがあります。WebLogic では、タイプ 2 の WebLogic jDriver for Oracle のほかに、Oracle XA 向けのタイプ 2 ドライバおよび 3 つのタイプ 3 ドライバ（RMI ドライバ、Pool ドライバおよび JTS ドライバ）を用意しています。

同様に、Oracle Application Server にも、データベース・アクセスのための様々なオプションが用意されています。これには、Oracle データベースに最適な JDBC ドライバをはじめとして、DB2 など各社のデータベースにアクセスするためのパートナ企業 Merant 社製 JDBC ドライバがあります。

- **WebLogic jDriver for Oracle**: このドライバでは、Oracle データベースに接続できます。また、Oracle Call Interface API（OCI）に基づいているため、Oracle クライアントのインストールが必要です。WebLogic jDriver for Oracle/XA ドライバは、WebLogic jDriver for Oracle を分散トランザクション用に拡張したものです。

  Oracle Thick ドライバ、つまり JDBC OCI ドライバは、WebLogic jDriver for Oracle および WebLogic jDriver for Oracle/XA に匹敵します。これは、JDBC OCI ドライバに XA 機能が備わっているためです。

- **WebLogic Pool ドライバ**: このドライバでは、HTTP サーブレットや EJB などのサーバー側アプリケーションから接続プールを利用できます。

- **Oracle JDBC OCI ドライバ**: このドライバでは、J2EE アプリケーションが接続プールを使用できます。このドライバは、JDBC 2.0 の接続プール機能を完全にサポートします。

- **WebLogic RMI ドライバ**: これは、WebLogic Server 内で稼動するタイプ 3 の複数階層 Java DataBase Connectivity（JDBC）ドライバで、2 層の JDBC ドライバと併用することによって、データベースにアクセスできます。さらに WebLogic RMI ドライバは、WebLogic Server のクラスタ内に構成すると、クラスタリングされた JDBC に使用できます。これにより JDBC クライアントは、WebLogic Clusters のロード・バランシングおよびフェイルオーバーを利用できます。

- **WebLogic JTS ドライバ**: これは、タイプ 3 の複数階層 JDBC ドライバで、1 つのデータベース・インスタンスを使用する複数のサーバー間で、分散トランザクションに使用されます。JTS ドライバは 2 フェーズ・コミットを行わないため、1 つのデータベース・インスタンスのみを使用する場合は、WebLogic jDriver for Oracle/XA ドライバよりも効率的です。

- **Oracle Thin ドライバ**: これはタイプ 4 の 2 層 Oracle Thin ドライバで、WebLogic Server から Oracle DBMS に接続できます。

すでに WebLogic Server から Oracle OCI ドライバまたは Oracle Thin JDBC ドライバを使用している場合は、コードを変更する必要がないので、OC4J でのデータ ソースの構成に関する項に進んでください。

データ ソースの移行

JDBC 2.0 仕様では、`java.sql.Datasource` クラスが導入され、JDBC プログラムが 100% 移植可能になりました。このバージョンでは、ベンダー固有の接続 URL およびマシンとポートの依存関係が削除されました。また、このバージョンでは `java.sql.DriverManager`、`Driver`、`DriverPropertyInfo` の各クラスの使用を控えるように推奨されています。以前の JDBC `DriverManager` 機能は、今回のデータ ソース機能に完全に置き換わりました。ドライバ・マネージャ・クラスをクライアント・アプリケーション・ランタイムに明示的にロードするかわりに、集中化された JNDI サービス・ルックアップによって `java.sql.Datasource` オブジェクトが取得されます。`Datasource` オブジェクトを使用してデータベースに接続することもできます。JDBC 2.0 API 仕様に従って、JDBC サブコンテキストまたはそのいずれかの子コンテキストの下にデータ ソースが登録されます。JDBC コンテキスト自体は、ルート・コンテキストの下に登録されます。`DataSource` オブジェクトは、データ ソースへのコネクション・ファクトリです。

WebLogic と OC4J は、どちらも JDBC 2.0 データ ソース API をサポートしています。J2EE サーバーは、JDBC ドライバ構成に基づいてドライバを暗黙的にロードするため、ドライバのロードにクライアント固有のコードは必要ありません。`DataSource` オブジェクトの参照には、Java Naming and Directory Interface（JNDI）ツリーを使用できます。

データ ソースのインポート文

`DataSource` オブジェクトを JNDI と併用することによって、データベースに接続するための接続プールにアクセスできます。データ ソースごとに、別々の `DataSource` オブジェクトが必要です。これは、接続プーリングと分散トランザクションのいずれかをサポートする `DataSource` クラスとして実装できます。

`DataSource` オブジェクトを使用するには、クライアント・コードに次のクラスをインポートします。

```
import java.sql.*;
import java.util.*;
import javax.naming.*;
```

WebLogic Server の場合は `weblogic.jdbc.*` パッケージ、OC4J の場合は `oracle.jdbc.*` パッケージを使用します。

アプリケーション・サーバーでのデータソースの構成

Oracle Application Server 用にデータソースを構成するには、Application Server Control コンソールの Web ページを使用して、データソース名、データベース名および JDBC URL 文字列を指定します。複数のデータソースが単一の接続プールを使用するように定義することもできます。そうすると、同じデータベースを共有するトランザクション対応および非対応両方の DataSource オブジェクトを定義できます。

データソースの構成と定義は、Application Server Control コンソールで行うのが最適です。ただしこのドキュメントでは、基礎をなすインフラストラクチャを調べ、構成ファイルの直接操作に焦点を当てます。OC4J では、フラット・ファイルを使用して、デプロイしたそのすべてのアプリケーションに対してデータソースを構成します。データソースの指定は、*<ORACLE_HOME>*/j2ee/home/config/data-sources.xml ファイルで行います。次に示すのは、Oracle データベースのデータソース構成の例です。data-sources.xml で指定するデータソース（xa-location、ejb-location および pooled-location）は、それぞれ一意にする必要があります。

```
<data-source
class="com.evermind.sql.DriverManagerDataSource"
name="Oracle"
url="jdbc:oracle:thin@<database host name><database listener port number>:<database
SID>"
pooled-location="jdbc/OraclePoolDS"
xa-location="jdbc/xa/OracleXADS"
ejb-location="jdbc/OracleDS"
connection-driver="oracle.jdbc.driver.OracleDriver"
username="scott"
password="tiger"
url="jdbc:oracle:thin@<database host name><database listener port number>:<database
SID>"
schema="database-schemas/oracle.xml"
inactivity-timeout="30"
max-connections="20"
/>
```

表 6–1 では、data-sources.xml のすべての構成パラメータを説明しています（前述の例に示されていないパラメータもあります）。

表 6–1 data-sources.xml ファイル内の構成パラメータ

パラメータ	説明
class	データソースのクラス名。
connection-driver	JDBC のクラス名。
connection-retry-interval	失敗した接続を再試行するまでの待機時間（秒）。 デフォルト値は 1 秒。

表 6–1 data-sources.xml ファイル内の構成パラメータ（続き）

パラメータ	説明
ejb-location	EJB 対応で、このデータソースのプーリングされたバージョンをバインドするための JNDI パス。このバージョンは、コンテナ管理のトランザクションに関係します。このタイプのデータソースは、EJB および同様のオブジェクト内から使用します。 このパラメータが適用されるのは、ConnectionDataSource のみです。
inactivity-timeout	使用されていない接続を閉じるまでにキャッシュしておく時間（秒）。
location	このデータソースをバインドするための JNDI パス。
max-connect-attempts	失敗した接続を再試行する回数。 デフォルトは 3 回。
max-connections	データソースをプーリングするために開いておく接続の最大数。
min-connections	データソースをプーリングするために開いておく接続の最小数。 デフォルトはゼロ。
name	データソースの表示名。
password	データソースにアクセスするためのユーザー・パスワード（省略可）。
pooled-location	このデータソースのプーリングされたバージョンをバインドするための JNDI パス。 このパラメータが適用されるのは、ConnectionDataSource のみです。 データベースに接続するための database-schema ファイルへの相対パスまたは絶対パス。
source-location	この特殊データソースの基礎となったデータソース。
url	このデータソースの JDBC URL（java.sql.Connections を扱う一部のデータソースで使用します）。
username	データソースにアクセスするためのユーザー名（省略可）。
wait-timeout	すべての接続が使用されている場合における、接続が解放されるまでの待機時間（秒）。デフォルトは 60。
xa-location	このデータソースのトランザクション・バージョンをバインドするための JNDI パス。 このパラメータが適用されるのは、ConnectionDataSource のみです。

表 6–1 data-sources.xml ファイル内の構成パラメータ（続き）

パラメータ	説明
xa-source-location	特殊なデータソースの基礎となった XADataSource（OrionCMTDataSource で使用します）。

データソース・オブジェクトを使用したクライアント接続の取得

JDBC クライアントから接続を取得するには、JNDI を使用して DataSource オブジェクトを検索し、そのロケーティングを行います。これは、次のコードのうち、WebLogic Server で接続を取得する部分に示されています。

```
try
{
    java.util.Properties parms = new java.util.Properties();
    parms.setProperty(Context.INITIAL_CONTEXT_FACTORY,
    "weblogic.jndi.WLInitialContextFactory");

    javax.naming.Context ctx = new javax.naming.InitialContext(parms);
    javax.sql.DataSource ds = (javax.sql.DataSource)ctx.lookup("jdbc/SampleDB");
    java.sql.Connection conn = ds.getConnection();

    // process the results
    ...
}
```

前述のコードを WebLogic Server から OC4J に移行するには、JNDI ツリーの初期コンテキスト・ファクトリ（Context.INITIAL_CONTEXT_FACTORY）を実装するクラスを、WebLogic 固有のクラスである weblogic.jndi.WLInitialContextFactory から、OC4J 固有のクラスである com.evermind.server.ApplicationClientInitialContextFactory に変更する必要があります。

この変更により、コードは OC4J にデプロイ可能となり、Oracle JDBC ドライバを使用できるようになります。

接続プールの移行

サーブレットやアプリケーション・サーバーなど、Web ベースのリソースの大半は、データベース内の情報にアクセスします。リソースは、データベースにアクセスするたびにデータベースへの接続を確立する必要があり、接続の作成、維持、さらに接続が不要になったときの解放を行う際に、システム・リソースを消費します。このリソースのオーバーヘッドは、Web ベースのアプリケーションで特に大きくなります。これは、Web ユーザーによる接続と切断が頻繁かつ大量に発生するためです。また、データのやり取りそのものよりも、接続と切断の際に消費されるリソースの量が多い場合も珍しくありません。

接続プーリングを使用すると、接続のオーバーヘッドを多数のユーザー・リクエストに分散して、接続リソースの使用を制御できます。接続プールは、データベースのリソースに複数のクライアントがアクセスする必要がある場合に、共有できる接続オブジェクトのセットをキャッシュしたものです。各クライアントがリソースを使用して専用の接続を作成し、データベース操作が完了した時点でその接続を閉じるのではなく、プール内の接続を作成するためのリソースは、指定された数の接続に対して一度しか消費されず、それらの接続が開いた状態で、クライアントからの多数のリクエストによって再利用されます。接続プーリングでは、次のようにして全体的なパフォーマンスを高めます。

- 中間層およびサーバーでの負荷を軽減します。
- セッションの作成操作および終了操作によるリソース使用率を最小化します。
- ソケットとファイル記述子の制限およびユーザー・ライセンス数の制限に起因するボトルネックを解消します。

JDBC 2.0 仕様では、次の目的で JDBC データソース接続のプールを定義することができます。

- リソースに対する接続の可用性を最大化します。
- プール内のアイドル接続数を最小化します。
- 孤立した接続をプールに戻して、他のサーブレットやアプリケーション・サーバーが再利用できるようにします。

これらの目的を実現するには、次の作業を行います。

1. 接続プールの最大サイズ・プロパティを、同時にアクティブとなるユーザーのリクエストの予想最大数と同じ値に設定します。
2. 接続プールの最小サイズ・プロパティを、同時にアクティブとなるユーザーのリクエストの予想最小数と同じ値に設定します。

接続プーリングのプロパティによって、ユーザー・リクエストの数が減少するにつれて、接続が徐々にプールから削除されます。同様に、ユーザー・リクエストの数が増え始めるにつれて、新しい接続が作成されます。接続の再利用を最大化し、接続作成のオーバーヘッドを最小化するように、接続のバランスが維持されます。また、接続プーリングを使用して、データベースへの同時接続の数を制御することもできます。

接続プールの概要

接続プールは、DBMS に対してすぐに使用できる接続のプールです。これらのデータベース接続は、接続プール起動時にすでに確立されているため、データベース接続の確立に要するオーバーヘッドがなくなります。接続プールは、プール・ドライバを使用する HTTP サーブレットや EJB などのサーバー側アプリケーションから、またはスタンドアロンの Java クライアント・アプリケーションから利用できます。

接続プーリングの最大のメリットとして、貴重なプログラム実行時間が節約され、オーバーヘッドがまったく、あるいはほとんどかからないことがあげられます。DBMS 接続の作成には非常に時間がかかります。接続プールを使用すると、ユーザーが必要とする前に接続が確立されて利用可能になります。これとは別に、アプリケーション・コードが必要時に独自の JDBC 接続を作成する方法もあります。DBMS が実行時に着信接続要求を処理する必要がある場合は、専用接続を使用したほうが処理は速くなります。

接続プールによるパフォーマンスの向上

DBMS への JDBC 接続の確立には、かなりの時間がかかる可能性があります。アプリケーションで何度もデータベース接続を開閉する必要がある場合は、この点がパフォーマンス上重大な問題となります。WebLogic Server および Oracle Application Server の接続プールは、この問題を解決します。

WebLogic Server または Oracle Application Server を起動すると、接続プールの接続が開き、すべてのクライアントが利用できるようになります。クライアントが接続プールの接続を閉じると、その接続がプールに戻されて、他のクライアントが利用できるようになります。つまり、接続自体は閉じられません。プール接続の開閉にはほとんどコストがかかりません。

プールにはどれだけの数の接続を作成する必要があるでしょうか。接続プールは、構成されたパラメータに応じて、最小接続数と最大接続数の間で増減します。パフォーマンスが最高となるのは常に、接続プール内の接続数が同時ユーザー数と同じであるときです。

クラスタリングされた JDBC の概要

クラスタリングされた JDBC は、多層構成にのみ関連し、サービスを提供しているクラスタ・メンバーに障害が発生した場合に、外部 JDBC クライアントが、接続パラメータを変更せずに JDBC 接続を再度確立して再開できます。WebLogic の場合、クラスタリングされた JDBC が DBMS に接続するには、データソース・オブジェクトと WebLogic RMI ドライバが必要です。データソース・オブジェクトは、WebLogic Administration Console を使用して、WebLogic Server ごとに定義します。

オラクル社は、OCI の TAF 機能を活用することで、クラスタリングされた JDBC が提供する機能と同等以上に高度な機能を実現しています。

JDBC のパフォーマンス・チューニング

JDBC アプリケーションのパフォーマンス・チューニングは、OC4J と WebLogic Server で類似しています。接続プーリングは、データベース接続の新規作成というコストのかかる操作が不要なため、パフォーマンスの向上に役立ちます。効率的なコードの作成に関するガイドラインは、Oracle Application Server にも WebLogic Server にも当てはまります。

A

その他の機能比較

この付録では、WebLogic Server 7.0 と Oracle Application Server 10*g* の比較情報を補足します。この情報の内容は次のとおりです。

- Java Messaging Service（JMS）
- Java Object Cache
- Dynamic Monitoring Service（DMS）
- Active Components for J2EE（AC4J）
- Oracle Application Server TopLink（OracleAS TopLink）

Java Messaging Service（JMS）

Oracle Application Server 10*g* と WebLogic Server 7.0 では、どちらも JMS 1.0.2 がサポートされています。表 A-1 に、両方のアプリケーション・サーバーがサポートする主要な JMS 機能の一部を示します。

表 A-1 JMS 機能の比較のまとめ

機能	Oracle Application Server 10*g*	WebLogic Server 7.0
プラッガブル JMS プロバイダ	○	○
メッセージ保存および問合せ機能	○	○
JMS メッセージの永続性	○	○
永続化された JMS メッセージのフェイルオーバー	○	×
メッセージ・ペイロード： 構造化データ型、非構造化データ型、リレーショナル・データ、テキスト、XML、オブジェクト、マルチメディア・データ	○	○
メッセージ・トランスポート： SOAP、Net8	○	○
保護アクセス	○	○
メンテナンスの容易な表への、ビジネス・ロジック、ルールおよびルーティングの抽象化	○	×
保証付きの配信	○	×
高可用性構成でのクラスタリング機能	○	×
Java クライアントおよび Java 以外のクライアントとのインタフェース	○	×

Oracle Application Server では、次のように JMS をサポートしています。

- 高速、軽量、準拠 : Oracle Application Server には、2 種類の JMS 実装が用意されています。

  その 1 つである OracleJMS は、Oracle データベースの統合 Advanced Queuing（AQ）を使用して、保護されトランザクション形式のリカバリ可能な保証付きのメッセージ配信を実現します。また、Oracle Application Server には、高速で軽量のメモリー内 JMS があり、中間層でのアプリケーション間のメッセージ交換に使用できます。一方、WebLogic の JMS 実装は単純です。

- プラッガブルの JMS プロバイダ : Oracle Application Server の J2EE アプリケーションは、JMS API を使用してキューとトピックにアクセスできます。これらのアプリケーションは、Oracle Application Server 固有の JNDI ネームスペースを使用して、JMS の `ConnectionFactories` および `Destinations` を検索します。

  Oracle Application Server は、メッセージ・プロバイダをプラグインするための `ResourceProvider` インタフェースを定義し、Oracle Advanced Queuing 用の実装クラスおよび MQSeries、SonicMQ、SwiftMQ などのサード・パーティ製メッセージング・システム用の実装クラスを用意しています。`ResourceProvider` インタフェースによって、メッセージ・プロバイダ間の切り替えが、JMS クライアントに対して透過的に行われます。JMS クライアントは、同じアプリケーションで複数のメッセージ・システムからのメッセージを併用し、ソース・コードは変更せずに JNDI マッピングを変更するだけで、それらのメッセージを切り替えることができます。

  WebLogic では、他の JMS プロバイダのプラグインが十分にサポートされていません。IBM MQ Series のサポートに対する WebLogic のアプローチは複雑です。開発者は、MQ Series をプラグインするために、別個のクラス・ライブラリである BEA WebLogic MQ Series JMS クラスを使用する必要があります。一方 Oracle Application Server では、プラグイン操作が非常に簡単で、`DataSource` と同じくらい単純です。

OracleJMS（OJMS）

OJMS は、Oracle データベースの統合 Advanced Queuing（AQ）に対する Java フロントエンドで、保護されトランザクション形式のリカバリ可能な保証付きのメッセージ配信を実現します。

Advanced Queuing には、重要な機能がいくつかあります。OJMS は、Oracle データベースの堅牢性、問合せ機能と DML 操作、スケーラビリティと高可用性、およびメッセージ・ペイロードでのあらゆるデータ型のサポート（リレーショナル・データ、テキスト、XML、マルチメディアなど）を利用しています。

次に、一般的な機能について説明します。

- メッセージ保存および問合せ機能 : OJMS は、メッセージ・システムと Oracle データベースを統合することで、データベースの堅牢性を利用し、メッセージの保存と監査 / 追跡を保証する一方、メッセージ・システムとデータベースの間で 2 台の PC を操作する必要がなくなります。さらに、キューが Oracle データベースに格納されているため、標準 SQL を使用してキューを問い合せることができます。

- メッセージ・ペイロード : OJMS は、様々な構造化データ型および非構造化データ型を、リレーショナル・データ、テキスト、XML、オブジェクト、マルチメディア・データなどのメッセージ・ペイロードとしてサポートできます。
- メッセージ・トランスポート : OJMS は、SOAP や Net8 など様々なトランスポートを介して、信頼性が高く 1 度のみ実行される、順序に従ったメッセージ配信をサポートします。また、MQ-Series など、他のメッセージ・プロバイダをトランスポートに使用することもできます。
- 保護アクセス : 最後に OJMS は、個々のキューとメッセージに対し、データベースの ACL メカニズムを使用して厳重なアクセス制御を行います。

WebLogic Server には、Oracle Messaging（JMS）が提供する次の機能がありません。

- メンテナンスの容易な表への、ビジネス・ロジック、ルールおよびルーティングの抽象化 : OJMS には、幅広いルール機能があります。ルールは、メッセージを効率的にルーティングするために使用されます。ルールは、サブスクリプションを定義する際に SQL 式として指定できます。ルール・エンジンによって、効率的なルール評価が実行されます。これらの SQL 式には、他の PL/SQL、C または Java の関数を含めることができます。Oracle9*i* Database（リリース 2）の場合は、ルールセットでルールを構成し、ルール機能をメッセージ・キューイングと関係なく使用することもできます。
- 保証付きの配信 : OJMS は、保証された 1 度のみの配信を実現します。これは OJMS 固有の機能です。キュー内のメッセージは、SQL ビューを使用して監視できます。単一の SQL 文を作成して、メッセージの正確な位置を調べることができます。
- 高可用性構成でのクラスタリング機能 : Oracle Advanced Queuing を使用することで、Real Application Clusters（RAC）による永続性、高可用性およびスケーラビリティなど、Oracle スタック全体の利点が得られます。

  一方、BEA WebLogic の JMS クラスタリングは、それほど堅牢ではありません。JMS を選択するうえでの重要な検討事項の 1 つが信頼性です。WebLogic Server には、サーバー障害に関係する永続メッセージのフェイルオーバーがありません。さらに、フェイルオーバーの対象が JMS 宛先のみです。ただし、これらの JMS 宛先はレプリケートされません。たとえば、クラスタのすべてのノードにわたって、同じ JNDI 名で複数の永続キューをデプロイできます。クライアントは 1 つのノードにのみ到達し、そのノードに障害が発生した時点で、クラスタによって、次に利用可能なノード / キューに透過的にフェイルオーバーされます。

  ただし、最初のキューに格納されていた内容はそのキューに残ります。これは、誰かが手動でそのノードを再起動するまでキューが永続化されるためです。あるいは、メッセージを取得するメカニズムを各自で見つける必要があります。一方、Oracle Application Server および Oracle AQ では、Oracle9*i* RAC の高可用性機能を利用して、この問題を回避できます。

- Javaクライアントおよび Java 以外のクライアントとのインタフェース : WebLogic Server の JMS 実装では、Java 以外のクライアントにメッセージを送信できません。Oracle Application Server の JMS には、Oracle AQ を通じて 4 つの API（PL/SQL、Java（JMS）、C、XML）が用意されています。これにより、どの言語でエンキューしたメッセージでも、別の言語でデキューでき、その結果、レガシー・システムを含めて、様々な異種システムとの柔軟な統合が可能となります。

Java Object Cache

Java Object Cache は、共有 Java オブジェクトへのアクセスのパフォーマンスを高め、SQL-to-Java のオーバーヘッド（データベース・アクセスのオーバーヘッド）を削減する目的で設計されています。Java Object Cache は、単一マシン上のプロセス境界、あるいはプロセッサとマシンの境界を越えて分散され、インプロセスでアクセスできます。

パフォーマンスを高めるために、Java オブジェクトの作成を分散してボトルネックを解消しています。キャッシュは構成可能なので、必要に応じて、オブジェクトのグループ化、プーリング、ピンによる固定化およびページングを実行できます。最後に、Java Object Cache はクラスタリング可能で、クラスタ間で更新を同期化できます。

HTTP レベルのクラスタ（サーブレットの場合）および EJB クラスタの目標は、ロード・バランシングされたフォルト・トレラントなシステムを提供することです。パフォーマンスの向上を主眼とする Java Object Cache では、事前に計算したオブジェクトを Oracle Application Server インスタンス間で分散して、コストのかかる計算が 1 度しか実行されないようにします。

Java Object Cache は Java オブジェクトのインプロセス・キャッシュで、あらゆる Java プラットフォームのあらゆる Java アプリケーションで利用できます。これにより、複数のリクエストやユーザー間でのオブジェクトの共有や、オブジェクトのライフ・サイクルの複数プロセスにわたる調整をアプリケーションで行うことができます。Java Object Cache は、アイランド、インスタンスまたはクラスタの関係がないプロセス間でも、データのレプリケーションが可能です。このレプリケーションでは、コストのかかる（共有）Java オブジェクトが、その生成元であるアプリケーションに関係なくキャッシュされるためにパフォーマンスが向上し、Java オブジェクトの再作成に必要なソース（データベースや外部アプリケーションなど）がダウンした場合の可用性も向上します。さらに、あるプロセスで更新または無効化されたオブジェクトは、関連する他のすべてのプロセスでも更新または無効化されます。この分散管理によって、プロセスのシステムを、集中制御で生じるオーバーヘッドなしに、常に同期化しておくことができます。さらに、Java Object Cache はオブジェクトのバージョニングもサポートするため、各種アプリケーションでオブジェクトの複数のバージョンを使用できます。これは、アプリケーションをアップグレードする際に特に便利です。

Java Object Cache は、ローカル・モードと分散モードという 2 つの操作モードをサポートします。ローカル・モードを使用すると、オブジェクトは単一の Java VM プロセスへと分離され、共有されません。分散モードを使用すると、Java Object Cache は、無効化、破棄、置換などのオブジェクト変更を、キャッシュのメッセージング・システムを通じて、単一システムまたはネットワークで稼動する他の通信中のキャッシュに伝播します（Java Object Cache のメッセージング・システムは、TCP/IP をベースに構築されています）。

このように、Java Object Cache の最大の主眼はパフォーマンスの向上ですが、スケーラビリティと可用性が向上するという副次的な効果も得られます。これは、バックエンド・サーバーがダウンしても、キャッシュの内容を計算結果として利用できるためです。

WebLogic Server には、Java Object Cache に匹敵する機能がありません。

Dynamic Monitoring Service（DMS）

デプロイされたアプリケーション・サーバーで重要なパフォーマンス・メトリックを収集することは、管理者がボトルネックをトラブルシューティングし、リソースの可用性に関する問題を特定し、アプリケーション・サーバーをチューニングするうえで不可欠です。Oracle Application Server には、Dynamic Monitoring Service と呼ばれる監視フレームワークが用意されています。

DMS はこれまでにも、Oracle HTTP Server、OC4J、OracleAS Portal、Oracle SOAP、JServ など、Oracle Application Server のいくつかのコンポーネントに装備されています。これらのコンポーネントやその他のコンポーネント内の主要なメトリックを公開することで、DMS は Oracle Application Server によって管理されるアプリケーションのパフォーマンスを包括的に表示します。

たとえば、Oracle HTTP Server 内で現在アクティブなリクエストの数、OC4J Servlet Engine で着信リクエストの解析に要した時間（ミリ秒）、JVM 内の空きメモリ領域の合計などのメトリックや、オペレーティング・システムの多数のメトリックが、DMS によって公開され、Application Server Control コンソールに表示されます。OEM などのクライアントは、HTTP 接続経由で DMS データを取得するため、これらのメトリックをブラウザで表示することもできます。

DMS は、Oracle Application Server に組み込まれる一方で、開発者が独自のアプリケーションを構築する際に、監視フレームワークとしても利用できます。DMS を使用すると、アプリケーション開発者は、カスタマイズしたアプリケーション固有のパフォーマンス・メトリックを測定してエクスポートできるため、管理者、開発者およびサポート・アナリストが、アプリケーションをサポートしやすくなります。

Java Management Extensions（JMX）と呼ばれる Java API があります。この API には、DMS に類似する部分と DMS を補足する部分があります。WebLogic Server は、JMX をある程度サポートしてきました。DMS と JMX はどちらも、仕様への組み入れを Java Community が検討中です。

Active Components for J2EE（AC4J）

既存の J2EE フレームワークは、作成およびデプロイするアプリケーションが、トランザクション・ベースで短時間で終了するものであれば、生産性とスケーラビリティに優れた環境です。ただし、自律型アプリケーションの間で長期間持続される対話を必要とする新たな E-Business には、まだ完全に対応していません。さらに、組織内で Web サービスの数が急増するにつれ、リソースの制限やシステム・クラッシュに妨害されることなく、疎結合環境で長期間にわたって、Web サービスどうしで通信する必要が生じます。自律型アプリケーションのコンポーネントと Web サービス間で長時間持続される対話は、関係するすべてのものがピアとして機能し、非同期かつ信頼できる方式で通信する必要があります。これらの問題に対処するため、Oracle Application Server には Active Components for J2EE（AC4J）が導入されました。これは、一貫性、スケーラビリティおよびリカバリ可能性に優れた疎結合アプリケーションを開発するためのプログラミング・フレームワークです。

AC4J では、Active Enterprise JavaBeans と呼ばれる疎結合 Bean の概念が採用されました。AC4J は、次の重要な機能を備えた標準 EJB です。

- 自律型ピア・モデル : AC4J を使用すると、各アプリケーションは、別のアプリケーションと対話する際に、自律型ピアとして機能します。応答するアプリケーションは、起動側アプリケーションに応答する前に、リクエストを無視するか、リクエスト元に代わって 1 つ以上の機能（場合によってはリクエスト元が依頼した機能とは別の機能）を実行するかを選択できます。どちらのアプリケーションも、ピアとして互いにリクエストすることはできますが、相手方からの送信を要求することはできません。どちらのアプリケーションも、そのピア・アプリケーションが所有するリソースに対する制御を引き継ぐことはできません。

  このプログラミング・モデルをサポートするために、Oracle Application Server では、Active EJB 間の対話は、信頼性に優れている、非同期、切断されている、一方通行、リクエスト / レスポンスといったタイプになります。

- 宣言的な指定 : AC4J は、このパラダイム全体を簡単にプログラミングできるようにするため、ユーザーが EJB デプロイメント・ディスクリプタに属性を追加するだけで、多数の情報を宣言的に指定できるようにしています。デプロイされた Oracle Application Server は、これらのコンポーネントに多数のサービスを自動的に提供するランタイム環境となります。

  AC4J は、キュー / トピックおよび関連する JMS 構成メンバーをアプリケーションから隠し、通信メッセージ形式を自動的に定義し、メッセージをパック / アンパックします。また、適切なサービス・プロバイダへのサービス要求の自動転送、セキュリティ・コンテキストの自動伝播、認可と ID の偽装、例外の自動転送と自動処理（EJB フレームワークに統合）、および Active EJB の計算進行状況の自動的な追跡と文書化を実現します。

- EJB アプリケーションのトランザクション型データ駆動実行 : AC4J は、指定されたルールに基づいて利用可能なデータで複合マッチングを行います。このルールには、これらのデータがどのような条件でどの EJB メソッドを実行するかが記述されます。AC4J では、EJB の透過的なスケジューリングとアクティブ化、およびそれに伴う EJB のメソッドの実行と連動させることで、EJB のトランザクション型データ駆動実行が可能になります。

- 長時間実行トランザクション : 実行中の各種サービスと関連タスク間の通信にはかなりの時間がかかる可能性があるため、AC4J は、必要に応じて状態を自動的にパッシブ化およびアクティブ化するというポータブルな方法で、J2EE の計算状態の自動永続性を実現します。
- 高度な機能 : AC4J を使用することで、Oracle Application Server はロード・バランシングとスケーラビリティをアプリケーションに対して透過的に管理し、対話状態およびグローバルに表示される識別情報（ActiveHandle）を提供します。また、Active EJB メソッドのパラレルな増分起動とその結果の同期化（fork 操作と join 操作）をサポートし、監視中にシステムやアプリケーションで障害が発生した場合の Active EJB ビジネス・モデルの再起動機能、およびアプリケーション整合性の強化をサポートします。

WebLogic Server には、AC4J に匹敵する機能がありません。

Oracle Application Server TopLink（OracleAS TopLink）

エンタープライズ Java 環境での最大の課題の 1 つは、ビジネス・オブジェクトとビジネス・コンポーネントをリレーショナル・データベース（RDB）に格納することです。OracleAS TopLink は、アプリケーション開発の生産性を高めるために、Java オブジェクトをリレーショナル・データベースにマッピングする、使いやすいマッピング・ワークベンチを用意し、アプリケーション開発の最も困難な局面の 1 つであるデータベースでの情報の保持を簡素化します。開発者は OracleAS TopLink を使用することで、望ましいアプリケーション設計やデータベースの整合性にほとんど影響を与えることなく、オブジェクトと Enterprise JavaBeans を、リレーショナル・データベースに柔軟にマッピングできます。その結果、開発者は、インフラストラクチャの構築でなく、ビジネス・ニーズの解決に集中できます。OracleAS TopLink は JDBC に基づいており、Oracle データベース、DB2、SQL Server、Sybase、Informix、Microsoft Access など、JDBC 準拠のあらゆるデータベースに移植できます。

OracleAS TopLink ソリューションには、重要な利点として次の 3 つがあります。

- 完成度を増した設計、柔軟性およびパフォーマンス : OracleAS TopLink は、パフォーマンスの最適化とスケーラビリティを実現した機能を豊富に備えています。パフォーマンスには、JDBC およびデータベースが提供する最適化機能を常に活用しながら、データベースとネットワークのトラフィックを最小化するキャッシュ技術で対応します。
- 簡素化されたアプリケーション開発 : OracleAS TopLink は、アプリケーション開発の生産性を高めるために、Java オブジェクトをリレーショナル・データベースにマッピングする、使いやすいマッピング・ワークベンチを用意し、アプリケーション開発の最も困難な局面の 1 つであるデータベースでの情報の保持を簡素化します。
- リソースの最適化 : アプリケーション開発チームは OracleAS TopLink を使用することで、インフラストラクチャの構築でなく、アプリケーションの構築に集中できます。

本質的に OracleAS TopLink は、Java からリレーショナル・データベースのオブジェクトへのリレーショナル・マッピングを実行するソリューションとして、市場で最も優れています。OracleAS TopLink によって、オラクル社は Java の世界とリレーショナル・データベースの世界を、できるかぎり最良の方法で融合しました。また、Java オブジェクトと Entity Bean をリレーショナル・データベースに生産的にマッピングするという、J2EE の開発者が直面する最大の課題の 1 つを解決しました。

Oracle Application Server 10*g* では、OracleAS TopLink が Oracle Application Server の統合コンポーネントとなっています。具体的には、OracleAS TopLink のフレームワークが Oracle Containers for J2EE と統合され、OracleAS TopLink のマッピング・ワークベンチが JDeveloper と統合されています。

索引

K

L

M

N

O

P

R

S

U

W

X

い

ら

ろ