Oracle® Application Server Containers for J2EE

スタンドアロン・ユーザーズ・ガイド

10*g* リリース 2（10.1.2）

部品番号 : B15631-02

2005 年 10 月

ORACLE®

Oracle Application Server Containers for J2EE スタンドアロン・ユーザーズ・ガイド , 10*g* リリース 2 （10.1.2）

部品番号 : B15631-02

原本名 : Oracle Application Server Containers for J2EE Standalone User's Guide, 10*g* Release 2 (10.1.2)

原本部品番号 : B14361-02

原本著者 : Sheryl Maring, Dan Hynes

原本協力者 : Brian Wright

Copyright © 2002, 2005, Oracle. All rights reserved.

制限付権利の説明

このプログラム（ソフトウェアおよびドキュメントを含む）には、オラクル社およびその関連会社に所有権のある情報が含まれています。このプログラムの使用または開示は、オラクル社およびその関連会社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権と工業所有権に関する法律により保護されています。

独立して作成された他のソフトウェアとの互換性を得るために必要な場合、もしくは法律によって規定される場合を除き、このプログラムのリバース・エンジニアリング、逆アセンブル、逆コンパイル等は禁止されています。

このドキュメントの情報は、予告なしに変更される場合があります。オラクル社およびその関連会社は、このドキュメントに誤りが無いことの保証は致し兼ねます。これらのプログラムのライセンス契約で許諾されている場合を除き、プログラムを形式、手段（電子的または機械的）、目的に関係なく、複製または転用することはできません。

このプログラムが米国政府機関、もしくは米国政府機関に代わってこのプログラムをライセンスまたは使用する者に提供される場合は、次の注意が適用されます。

U.S. GOVERNMENT RIGHTS

Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このプログラムは、核、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションへの用途を目的としておりません。このプログラムをかかる目的で使用する際、上述のアプリケーションを安全に使用するために、適切な安全装置、バックアップ、冗長性（redundancy）、その他の対策を講じることは使用者の責任となります。万一かかるプログラムの使用に起因して損害が発生いたしましても、オラクル社およびその関連会社は一切責任を負いかねます。

Oracle、JD Edwards、PeopleSoft、Retek は米国 Oracle Corporation およびその子会社、関連会社の登録商標です。その他の名称は、他社の商標の可能性があります。

このプログラムは、第三者の Web サイトへリンクし、第三者のコンテンツ、製品、サービスへアクセスすることがあります。オラクル社およびその関連会社は第三者の Web サイトで提供されるコンテンツについては、一切の責任を負いかねます。当該コンテンツの利用は、お客様の責任になります。第三者の製品またはサービスを購入する場合は、第三者と直接の取引となります。オラクル社およびその関連会社は、第三者の製品およびサービスの品質、契約の履行（製品またはサービスの提供、保証義務を含む）に関しては責任を負いかねます。また、第三者との取引により損失や損害が発生いたしましても、オラクル社およびその関連会社は一切の責任を負いかねます。

目次

2 高度な構成および開発

3 セキュリティの構成

A OC4J のトラブルシューティング

B 追加情報

C　サード・パーティ・ライセンス

索引

はじめに

ここでは、このマニュアルの概要、対象読者、構成および表記規則について説明します。また、Oracle の関連ドキュメントの一覧を示します。

対象読者

このマニュアルは、スタンドアロン・モードの Oracle Application Server Containers for J2EE（OC4J）の使用を検討している方すべてを対象としています。ただし、次の基本知識があることが前提となります。

- Java および J2EE
- XML
- JDBC

ドキュメントのアクセシビリティについて

オラクル社は、障害のあるお客様にもオラクル社の製品、サービスおよびサポート・ドキュメントを簡単にご利用いただけることを目標としています。オラクル社のドキュメントには、ユーザーが障害支援技術を使用して情報を利用できる機能が組み込まれています。HTML 形式のドキュメントで用意されており、障害のあるお客様が簡単にアクセスできるようにマークアップされています。標準規格は改善されつつあります。オラクル社はドキュメントをすべてのお客様がご利用できるように、市場をリードする他の技術ベンダーと積極的に連携して技術的な問題に対応しています。オラクル社のアクセシビリティについての詳細情報は、Oracle Accessibility Program の Web サイト http://www.oracle.com/accessibility/ を参照してください。

ドキュメント内のサンプル・コードのアクセシビリティについて

JAWS（Windows のスクリーン・リーダー）は、ドキュメント内のサンプル・コードを正確に読めない場合があります。コード表記規則では閉じ括弧だけを行に記述する必要があります。しかし JAWS は括弧だけの行を読まない場合があります。

外部 Web サイトのドキュメントのアクセシビリティについて

このドキュメントにはオラクル社およびその関連会社が所有または管理しない Web サイトへのリンクが含まれている場合があります。オラクル社およびその関連会社は、それらの Web サイトのアクセシビリティに関しての評価や言及は行っておりません。

関連ドキュメント

OC4J の詳細は、他の OC4J マニュアルに含まれる、次に示すドキュメントを参照してください。

- 『Oracle Application Server Containers for J2EE サービス・ガイド』
- 『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』
- 『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』
- 『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』
- 『Oracle Application Server Containers for J2EE セキュリティ・ガイド』
- 『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』

次のドキュメントも OC4J を理解するのに役立ちます。

- 『Oracle Application Server パフォーマンス・ガイド』
- 『Oracle Application Server 高可用性ガイド』
- 『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』
- 『Oracle HTTP Server 管理者ガイド』
- 『Oracle Application Server DMS API Reference』

表記規則

本文では、次の表記規則を使用します。

規則	意味
太字	太字は、操作に関連するグラフィカル・ユーザー・インタフェース要素、また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。
イタリック	イタリックは、特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	固定幅フォントは、パラグラフ内のコマンド、URL、例に記載されているコード、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

オラクル社カスタマ・サポート・センター

オラクル製品サポートの購入方法、およびオラクル社カスタマ・サポート・センターへの連絡方法の詳細は、次の URL を参照してください。

```
http://www.oracle.co.jp/support/
```

製品マニュアル

製品のマニュアルは、次の URL にあります。

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

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

```
http://www.oracle.co.jp/education/
```

その他の情報

オラクル製品やサービスに関するその他の情報については、次の URL から参照してください。

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

注意： ドキュメント内に記載されている URL や参照ドキュメントには、Oracle Corporation が提供する英語の情報も含まれています。日本語版の情報については、前述の URL を参照してください。

1

構成およびデプロイ

この章では、できるだけ簡単かつ迅速に OC4J を構成および実行する方法を説明します。OC4J では、サーブレット、JSP ページおよび EJB を実行できます。アプリケーションの OC4J へのデプロイ方法の例として、この章では、FAQ アプリケーション・デモの構成方法を説明します。

この章には、次の項目が含まれています。

OC4J スタンドアロンの概要

Oracle Application Server Containers for J2EE（OC4J）スタンドアロンは、すべて Java で作成された完全な Java 2 Enterprise Edition（J2EE）1.3 環境を提供します。これは標準 Java Deployment Kit（JDK）の Java 仮想マシン（JVM）上で実行されます。

OC4J は、J2EE 1.3 に準拠しており、J2EE で指定されるすべてのコンテナ、API およびサービスを提供します。OC4J は、先進的な J2EE コンテナの 1 つ、Orion Server を開発している Ironflare 社からライセンス供与を受けているテクノロジをベースにしています。そのため、製品およびドキュメントの一部には Orion Server の名称が残っています。

表 1-1 に示すように、OC4J は標準 J2EE API をサポートし、これに準拠しています。

表 1-1 OC4J の J2EE サポート

J2EE 1.3 標準 API	サポートしているバージョン
JavaServer Pages（JSP）	1.2
Servlet	2.3
Enterprise JavaBeans（EJB）	2.0
Java Transaction API（JTA）	1.0
Java Message Service（JMS）	1.0
Java Naming and Directory Interface（JNDI）	1.2
Java Mail	1.1.2
Java Database Connectivity（JDBC）	2.0 Extension
Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider	1.0
J2EE Connector Architecture	1.0
JAXP	1.1

OC4J スタンドアロンは、開発および小～中規模の本番デプロイでの使用を想定しています。特に、Oracle HTTP Server を使用せずに HTTP および HTTPS をネイティブ・サポートします。ロード・バランシング、クラスタリング、および Oracle Enterprise Manager 10*g* を介した管理はサポートしていません。これらの機能を使用するには、Oracle Application Server インストール・タイプのいずれか（J2EE + WebCache など）をインストールする必要があります。スタンドアロン・バージョンは、シングル・インスタンス、シングル JVM およびシングル・マシン構成でサポートされます。

OC4J のマニュアルでは、Java プログラミング、J2EE テクノロジ、および Web アプリケーションと EJB アプリケーションのテクノロジの基礎的な知識が前提となります。これには、`/WEB-INF` および `/META-INF` ディレクトリなどのデプロイ規則が含まれます。

OC4J インストール

OC4J は、J2EE に準拠する軽量なコンテナです。OC4J は、デフォルトの状態で強力かつ実用的に構成されているため、インストール後すぐに実行できます。OTN から `oc4j_extended.zip` ファイルをダウンロードした後、このファイルを解凍して OC4J をインストールします。次の項で、この作業を行う方法について説明します。

- 要件
- 基本インストール
- デフォルトの構成のテスト

要件

OC4J は、CLASSPATH に何も追加しなくても実行できます。インストール・ディレクトリ、lib/ サブディレクトリ、およびデプロイ済のアプリケーションの EAR、WAR または ejb-jar ファイルから、Java JAR ファイルおよびクラス・ファイルが直接ロードされるためです。

基本インストール

OC4J は、oc4j_extended.zip という名前の ZIP ファイルとして OTN で配布されています。このファイルを解凍した後、README.TXT に記載された指示に従ってください。パスに指定した任意のディレクトリに、この ZIP ファイルをインストールします。

$PATH に Java2 バージョン（バージョン 1.3.1 または 1.4.1 が望ましい）の Java 実行可能ファイルが必要です。OC4J をインストールするには、次のコマンドを実行します。

```
% cd /your_directory
% unzip oc4j_extended.zip
% cd j2ee/home
% java -jar oc4j.jar -install

Enter an administrator password
```

インストールが完了すると、デフォルト構成での OC4J の実行に必要なファイルがすべて j2ee/home ディレクトリに格納されます。インストール中に、管理コンソール・コマンドライン・ツールで使用される管理ユーザー名およびパスワードの入力を求められます。

注意： oc4j.jar を j2ee/home ディレクトリから実行するかわりに、$J2EE_HOME 変数（UNIX の場合）または %J2EE_HOME% 変数（Windows の場合）を j2ee/home に設定し、コマンドラインで oc4j.jar を任意のディレクトリから実行することも可能です。

たとえば、UNIX 環境では次のコマンドを使用します。

```
% java -jar $J2EE_HOME/oc4j.jar
```

デフォルトの構成のテスト

OC4J は、デフォルトの構成でインストールされます。これには、デフォルトの Web サイトおよびデフォルトのアプリケーションが含まれます。そのため、OC4J をただちに起動してテストできます。

OC4J を起動するには、次の手順を実行します。

1. ディレクトリを OC4J のインストール・ディレクトリ（j2ee/home）に変更し、次のいずれかのコマンドを実行します。
   - java -jar oc4j.jar

     この場合、OC4J は j2ee/home/config にあるデフォルトの構成ファイルを使用して起動します。
   - java -jar oc4j.jar -config /mypath/server.xml

     この場合、OC4J は /mypath にある server.xml ファイルを使用して起動します。

   バージョン番号を含む初期化文字列がサーバーから出力されます。
2. Web ブラウザから http://*hostname:*8888/ にアクセスし、OC4J をテストします。デフォルトのポート番号を変更した場合は、http://*hostname:portnumber*/ を使用して Web サーバーにアクセスします。

たとえば、Web ブラウザを http://*hostname*:8888/servlet/HelloWorldServlet に接続して Web サーバーをテストします。この場合、Hello World ページが返されます。

OC4J の起動および停止の詳細は、1-4 ページの「OC4J の起動および停止」を参照してください。構成の詳細は、1-13 ページの「アプリケーションのデプロイ」を参照してください。

OC4J の起動および停止

OC4J の起動

OC4J は、デフォルトの構成でインストールされます。これには、デフォルトの Web サイトおよびデフォルトのアプリケーションが含まれます。したがって、OC4J をただちに起動できます。

重要： ロギング実装上の依存性の問題のために、JDK 1.3 を使用すると OC4J の起動が失敗します。この問題を解決するには、`ORACLE_HOME/j2ee/home/config/server.xml` 構成ファイルの次のエントリを削除するか、コメント化します。

```
<j2ee-logging-config path="./j2ee-logging.xml" />
```

スタンドアロン環境で OC4J を起動するには、`j2ee/home/` ディレクトリから次のコマンドを実行します。

```
java -jar oc4j.jar options
```

このコマンドでは、OC4J は `j2ee/home/config` ディレクトリにあるデフォルトの構成ファイルを使用して起動します。

このコマンドのオプションは、OC4J を起動する際には不要です。ただし、より強力な制御が必要な場合は、B-25 ページの「OC4J サーバー JAR のオプション」に記載されたオプションを使用するか、`j2ee/home` ディレクトリから次のコマンドを実行してください。

```
java -jar oc4j.jar -help
```

OC4J が起動すると、そのことを示すメッセージが画面に表示されます。

注意： `oc4j.jar` を `j2ee/home` ディレクトリから実行するかわりに、`$J2EE_HOME` 変数（UNIX の場合）または `%J2EE_HOME%` 変数（Windows の場合）を `j2ee/home` に設定し、コマンドラインで `oc4j.jar` を任意のディレクトリから実行することも可能です。

たとえば、UNIX 環境では次のコマンドを使用します。

```
% java -jar $J2EE_HOME/oc4j.jar
```

OC4J の管理

OC4J サーバーの起動後、`<install_directory>/j2ee/home` にある `admin.jar` コマンドライン・ツールを使用してサーバーを管理できます。`admin.jar` コマンドを使用する場合、次の構文を参照してください。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password options
```

変数は次のとおりです。

- *oc4j_host:oc4j_ormi_port*: アプリケーションのデプロイ元となる OC4J サーバーのホスト名およびポート。admin.jar ツールは、OC4J Remote Method Invocation（ORMI）プロトコルを使用して OC4J サーバーと通信します。したがって、この変数で指定するホスト名およびポートは、リクエストの送信先となる OC4J サーバーの rmi.xml ファイルで定義します。

  ORMI プロトコルのデフォルト・ポート番号は 23791 です。このデフォルトを使用しない場合は、rmi.xml ファイルの <rmi-server> 要素で、ホスト名とポート番号の両方を次のように構成します。

  ```
  <rmi-server port="oc4j_ormi_port" host="oc4j_host">
  ```

- *admin_id admin_password*: 管理 ID およびパスワード。この OC4J サーバーの ID およびパスワードは、principals.xml ファイルで指定します。

このツールのオプションについては、B-25 ページの「OC4J 管理 JAR のオプション」で説明しています。

OC4J の再起動

OC4J のタスク・マネージャがデプロイ済アプリケーションの変更を自動的に検出するかどうかを指定できます。変更が検出されると、変更されたアプリケーションが自動的に再ロードされます。その場合、アプリケーションの再デプロイ時にサーバーを再起動する必要はありません。

server.xml ファイルにある <application-server> 要素の check-for-updates 属性はデフォルトで true に設定されており、オート・デプロイが有効になっています。true の場合、タスク・マネージャは XML 構成ファイルが変更されていないかどうかをチェックします。したがって、この属性を false に設定すれば、XML の新たな変更に対する構成の自動更新を無効にできます。また、この属性を false に設定すると、admin.jar -updateConfig を実行するまでアプリケーションのオート・デプロイが停止します。つまり、XML ファイルの内容を反映させて XML 構成を更新したり、オート・デプロイを行う必要がある場合は、admin.jar -updateConfig オプションを使用することになります。

オート・デプロイを有効にした場合、アプリケーションを変更するたびに OC4J プロセスを再起動する必要はありません。ただし、オート・デプロイを有効にするとパフォーマンスも影響を受けます。そのため、オート・デプロイを有効にするのは開発環境のみとし、本番環境では有効にしないことをお薦めします。

オート・デプロイを有効にしている場合でも、グローバル・サーバーの XML 構成ファイルにおける変更は検出されません。そのため、data-sources.xml、rmi.xml、principals.xml などのコンテナレベルの構成ファイルを変更した場合は、変更が認識されるように OC4J プロセスを再起動する必要があります。

デフォルト・パラメータを使用して OC4J を再起動する場合、インストール・ルート・ディレクトリに移動し、次のコマンドを実行します。

```
% java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin
    admin_password -restart
```

このコマンドは、OC4J RMI リスナー・ポートに接続して再起動をリクエストします。JVM がシグナルに応答しない場合や RMI メッセージを受け入れない場合には、このコマンドが機能しないことがあります。その場合、UNIX 環境では kill *process* というオペレーティング・システム・コマンドを使用して JVM を停止します。Windows 環境では、Windows タスク・マネージャを使用して JVM を終了します。

OC4J の停止

OC4J を停止するには、次のコマンドを実行します。

```
% java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin
    admin-password -shutdown force
```

このコマンドでは、コンテナが正常に停止します。コンテナが停止しない場合は、次のように force 引数を指定して、強制的に高速停止を実行します。

```
% java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin
    admin-password -shutdown force
```

この方法でもうまく行かない場合は、ご使用のシステムに応じて、オペレーティング・システム・コマンドまたはツールで OC4J プロセスを中断します。

HTTP および RMI での通信

HTTP アプリケーションの場合、クライアントはリクエストを直接 OC4J に送信できます。デフォルトのポート番号は 8888 です。このポート番号は、`http-web-site.xml` ファイルなどの該当する `*-web-site.xml` ファイルで変更できます。

EJB や JMS など RMI ベースのアプリケーションの場合、クライアントはリクエストを直接 OC4J に送信する必要があります。デフォルトの RMI ポートは 23791 です。このポート番号は `rmi.xml` ファイルで変更します。手順は、2-7 ページの「リスナーの構成」を参照してください。

JSP およびサーブレットのクイック・スタート

Web アプリケーションを OC4J にデプロイするには、次のいずれかを実行します。

- サーブレット・クラスおよび JSP ページを `j2ee/home/default-web-app` ディレクトリに配置します。
- `admin.jar` ツールを使用して J2EE アプリケーションをデプロイします。J2EE アプリケーションは EAR 形式でアーカイブしておく必要があります。

アプリケーションをデプロイする場合や、J2EE アプリケーションを旧バージョンの OC4J から移行する場合には、サーブレットおよび JSP ページを `default-web-app` ディレクトリに配置するのが最も簡単な方法です。

サーブレットまたは JSP を素早くデプロイするには、次の手順を実行します。

1. Java パッケージに対応するディレクトリの `j2ee/home/default-web-app/WEB-INF/classes` サブディレクトリにサーブレット・クラスを配置します。サーブレットには、`http://oc4j_host:8888/servlet/class-name` という形式の URL からアクセスできます。

   たとえば、サーブレット・クラス `my.HelloServlet` を次のように配置します。

   ```
   j2ee/home/default-web-app/Web-INF/classes/my/HelloServlet.class
   ```

   このサーブレット・クラスは次の URL からアクセス可能です。

   ```
   http://oc4j_host:8888/servlet/my.HelloServlet
   ```

2. `j2ee/home/default-web-app` ディレクトリの任意の場所に JSP ページを配置します。JSP ページには `http://oc4j_host:8888/path-to-JSP` という形式の URL からアクセスできます。

   たとえば、`j2ee/home/default-web-app/examples/Hello.jsp` の JSP ページには `http://oc4j_host:8888/examples/Hello.jsp` と指定すればアクセスできます。

開発ディレクトリの作成

アプリケーション開発時には、一貫性があり意味のある命名規則を使用することをお薦めします。たとえば、アプリケーションの名前を使用したディレクトリ内で、アプリケーションをモジュールとして開発します。このディレクトリ内のすべてのサブディレクトリは、JAR、WAR および EAR アーカイブ作成用の構造と一貫性を持たせます。こうすれば、ソースのアーカイブ時には必要なアーカイブ形式がすでに準備できています。図 1-1 に、この構造を示します。

図 1-1 開発アプリケーションのディレクトリ構造

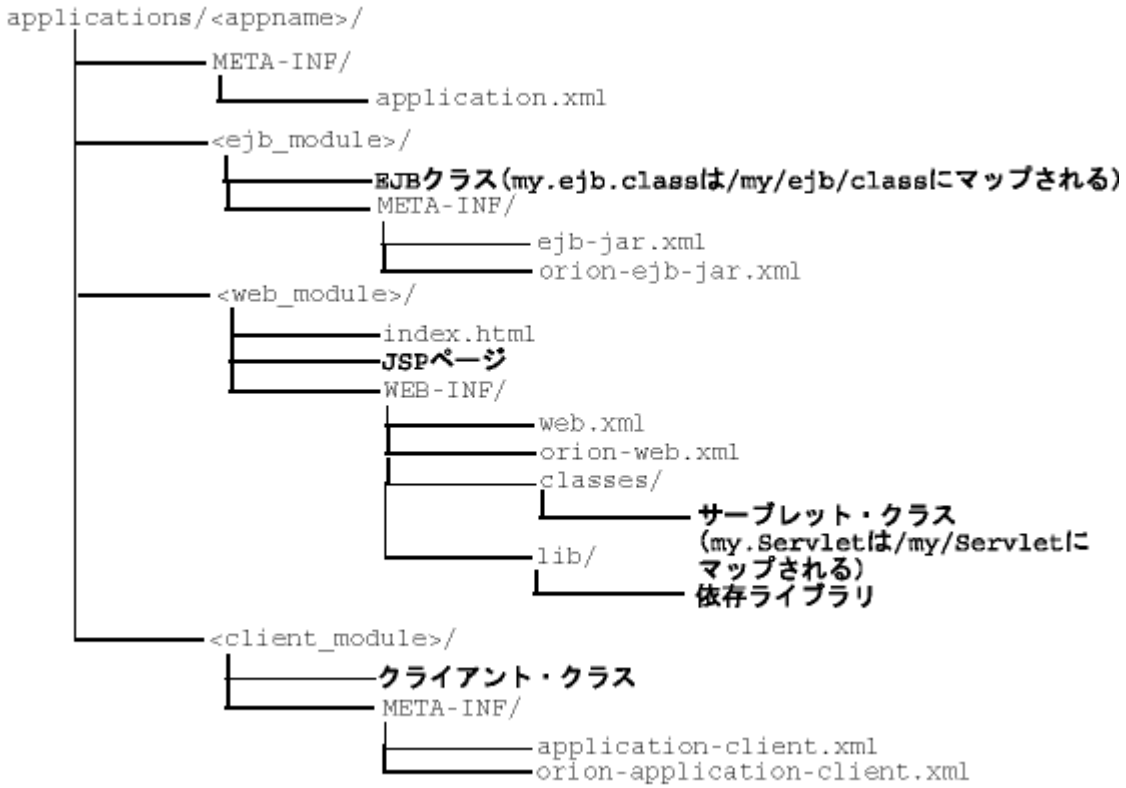


図 1-1 に関しては、次の点を考慮してください。

- ディレクトリ名および XML ファイル名の `META-INF`、`WEB-INF`、`application.xml`、`ejb-jar.xml`、`web.xml`、`application-client.xml` は変更できません。
- ディレクトリを分けることにより、エンタープライズ Java アプリケーションのモジュールがそれぞれ明確に区別されます。標準 J2EE アプリケーション・ディスクリプタ・ファイルとして機能する `application.xml` ファイルにより、これらのモジュールが定義されます。
- 個別のモジュール（`<ejb_module>`、`<web_module>` および `<client_module>`）が入っているディレクトリには任意の名前が使用可能です。ただし、これらの名前は、標準の J2EE アプリケーション・ディスクリプタ・ファイル（ローカルの `application.xml` ファイル）の値と一致する必要があります。
- モジュールの最上位は、クラスの検索パスの開始を示します。したがって、パッケージに所属するクラスは、この下のネストされたディレクトリ構造内に存在しているとみなされます。たとえば、EJB パッケージ・クラス `myapp.ejb.Demo` の参照は、*`appname/ejb_module/`*`myapp/ejb/Demo.class` 内に存在しているとみなされます。

FAQ アプリケーション・デモの構成

この項では、FAQ J2EE デモ・アプリケーションを構成する方法について説明します。FAQ J2EE デモ・アプリケーションは、よくある質問（FAQ）を管理し、Oracle データベースからこれらの FAQ を格納 / 取得するためのサポートを提供します。FAQ は、カテゴリに大別されます。各カテゴリは、トピックに細分化されます。各 FAQ は、複数のカテゴリに関連付けられます。カテゴリには、各カテゴリに関連付けられた 1 つ以上のトピックがあります。

内部または外部に公開するために、特定のカテゴリ用に FAQ のリスト（HTML 形式）を生成できます。

- 内部 : 内部ユーザーのみに公開される FAQ。すべての外部 FAQ および内部 FAQ が含まれます。
- 外部 : 外部フォーラムで公開される FAQ。

デモ内では、カテゴリ、トピックおよび FAQ は、入力 / 更新画面または OracleAS Web Services インタフェースを介して、データベース内で入力または更新されます。各カテゴリ、トピックおよび FAQ は、主キーで一意に識別されます。主キーは、システムによって自動的に生成されます。

これは J2EE 1.3 準拠のアプリケーションで、次のテクノロジを使用して開発されました。

- HTML（リッチ・テキスト・エディタを作成するための MS-HTML を含む）
- JavaScript
- カスケード・スタイル・シート
- Java Server Pages 1.2
- サーブレット 2.3
- JSP 標準タグ・ライブラリ（JSTL）1.0
- Oracle JSP 1.2 ユーティリティ・タグ・ライブラリ
- Enterprise JavaBeans 2.0（ローカル・インタフェース、Abstract クラス、CMR および EJB-QL を使用）
  - Entity Bean（CMP）
  - Session（Facade）Bean（ステートレス）
- Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider
- Oracle Application Server Web Services

次の各項で、FAQ デモ・アプリケーションを構成およびデプロイする方法を説明します。また、最後の項では、OC4J を構成およびデプロイするアプリケーションにこれらの手順を関連付ける方法について説明します。

- FAQ デモの環境設定
- FAQ デモの OC4J システム構成
- FAQ デモのデプロイ
- デプロイメントの詳細の説明

FAQ デモの環境設定

OC4J を構成して FAQ デモをデプロイする前に、バックエンド・データベースを変更し、FAQ デモの実行に必要な表を含める必要があります。

Oracle データベース

SQL 表作成スクリプト CreateTables.sql を実行して、FAQ デモのデータベース表を作成します。このスクリプトは、`http://<hostname>:8888/FAQApp/CreateTables.sql` にあります。または、OTN（`http://www.oracle.com/technology/tech/java/oc4j/demos/904/index.html`）から、FAQ アプリケーションの残りの部分とともに FAQApp.zip ファイルとしてダウンロードできます。

Oracle データベース環境では、表をインストールするデータベースおよびスキーマに接続して、`@CreateTables` を実行することで、SQL*Plus を介して SQL スクリプトを実行できます。SQL*Plus の使用方法、インストール・スクリプトの実行、データベース・ユーザー / スキーマの作成などの詳細は、Oracle データベースのマニュアルを参照してください。

FAQ デモの OC4J システム構成

FAQ デモを正しく実行するには、次のシステムの変更を実装する必要があります。

- バックエンド・データベースを指すように、デフォルトのデータソース OracleDS を変更します。
- jazn.com レルムに FAQ ユーザーを追加して、users ロールに割り当てます。

これらの各手順の詳細は、次の各項で説明します。

- データソースの構成
- セキュリティの設定

データソースの構成

FAQ アプリケーションを実行するには、対応する FAQ アプリケーション・データベース・スキーマがインストールされた、Oracle データベースを使用する必要があります。FAQ アプリケーションは、Application Server に付属する OracleDS という名前のデフォルトのグローバル・データソースを使用します。Application Server は、FAQ 表を作成したデータベースに接続できるように構成する必要があります。

注意： グローバルな OracleDS データソースを適切に更新しないと、I/O 例外がスローされます。

バックエンド・データベースがシン JDBC ドライバを使用し、myhost:1521/MYSERVICE にあり、faq/faq というユーザー名 / パスワードを使用する場合、j2ee/home/config/data-sources.xml ファイルは、次のように、jdbc:oracle:thin:@myhost:1521/MYSERVICE という URL のデータベース・サービスを指すように変更されます。

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="OracleDS"
  location="jdbc/OracleCoreDS"
  xa-location="jdbc/xa/OracleXADS"
  ejb-location="jdbc/OracleDS"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="faq"
  password="faq"
  url="jdbc:oracle:thin:@myhost:1521/MYSERVICE"
  inactivity-timeout="30"
/>
```

セキュリティの設定

FAQ デモは、Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider を使用して、認証およびユーザー・アクセス制御機能を提供します。次のように、jazn.jar コマンドライン・ツールを使用して、デフォルトの jazn.com レルムにアプリケーション・ユーザーが追加されます。

```
> java -jar jazn.jar -adduser jazn.com <username> <passwd>
> java -jar jazn.jar -grantrole users jazn.com <username>
```

この例では、jazn.com レルムにユーザーを追加してから（ユーザー名とパスワードを指定）、新しいユーザーに users ロールを付与しています。Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider をセキュリティ・プロバイダとして使用する方法の詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

FAQ デモのデプロイ

OTN（http://www.oracle.com/technology/tech/java/oc4j/demos/904/index.html）から FAQ デモ・アプリケーション（FAQApp.zip ファイル）をダウンロードします。

1. このファイルを作業ディレクトリに解凍します。この作業ディレクトリは、<FAQApp_Home> と呼ばれます。
2. EAR ファイルを j2ee/home/applications ディレクトリにコピーするか、admin.jar ツールを使用して、FAQ アプリケーションをデプロイします。それぞれの方法について、次の各項で説明します。
3. java -jar oc4j.jar を実行して、OC4J サーバーを起動します。
4. FAQ アプリケーションをブラウザで実行します。その際のデフォルト・ポートは 8888 です。

   http://*your_machine_name:8888*/FAQApp

開発環境でのオート・デプロイを使用したデプロイ

1-5 ページの「OC4J の再起動」で説明したように、OC4J はアプリケーションのオート・デプロイおよび再デプロイをサポートしています。そのため、アプリケーションの EAR ファイルに加えた変更は OC4J の停止および再起動を行わなくてもサーバーで認識されます。この機能は、server.xml ファイルの check-for-updates 属性を使用して有効にします。

オート・デプロイが有効になっている場合、XML 構成ファイルを変更した後、その XML ファイルを使用するアプリケーションを EAR ファイルに再度アーカイブし、EAR ファイルをアプリケーションのディレクトリにコピーするだけで済みます。OC4J サーバーが変更日を認識し、必要に応じてアプリケーションを再デプロイします。

警告： オート・デプロイを使用するのは開発環境のみとしてください。タスク・マネージャでの更新のチェックには長時間かかることがあります。本番環境では、check-for-updates 属性を false に設定し、オート・デプロイをオフにします。

FAQ アプリケーションを初めて（ローカルに）デプロイする場合、次の手順を実行します。

1. <FAQApp_Home>/faq/dist/FAQApp.ear ファイルを j2ee/home/applications ディレクトリにコピーします。
2. j2ee/home/config/server.xml および http-web-site.xml ファイルを次のように変更し、FAQ アプリケーションを j2ee/home/applications ディレクトリに登録します。
   a. j2ee/home/config/server.xml ファイルに、次のような FAQApp エントリを追加します。

      ```
      <application name="FAQApp" path="../applications/FAQApp.ear" />
      ```

      この手順では、FAQ アプリケーションが OC4J にデプロイされます。パスは、j2ee/home/config に対する相対パスです。FAQApp.ear ファイルが j2ee/home/applications にあるため、パスは ../applications/FAQApp.ear になります。

      server.xml 構成ファイルの詳細は、B-6 ページの「server.xml ファイルの要素」を参照してください。

   b. j2ee/home/config/http-web-site.xml ファイルに、次のような FAQApp エントリを追加し、FAQ Web アプリケーションをバインドします。

      ```
      <web-app application="FAQApp" name="FAQAppWeb" root="/FAQApp" />
      ```

      この手順では、OC4J サーバーから FAQ にアクセスできるようになります。

http-web-site.xml 構成ファイルの詳細は、『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。

詳細は、2-7 ページの「開発環境でのアプリケーションの手動追加」、2-10 ページの「OC4J でのアプリケーションのオート・デプロイ」、および 1-16 ページの「デプロイ時の動作」を参照してください。

あらゆる環境での admin.jar ツールを使用したデプロイ

本番環境では、check-for-updates 属性を false に設定し（1-5 ページの「OC4J の再起動」を参照）、すべてのアプリケーションのデプロイを、admin.jar ツールを使用して行う必要があります。admin.jar ツールは、アプリケーションをデプロイし、該当するすべての XML ファイルを変更します。そのため、リモート・デプロイが可能になります。

admin.jar コマンドライン・ツールを使用して、次のように登録およびデプロイを行います。

```
java -jar admin.jar
      ormi://oc4j_host:oc4j_ormi_port
      admin welcome -deploy
      -file d:¥j2ee¥home¥applications¥FAQApp.ear
      -deploymentName FAQApp
      -targetPath applications/
```

この手順では、server.xml ファイルに FAQ アプリケーションのエントリが作成されます。admin.jar コマンドライン・ツールの詳細は、B-25 ページの「OC4J 管理 JAR のオプション」を参照してください。

admin.jar ツールを使用すれば、次のように任意の Web アプリケーションをバインドできます。

```
java -jar admin.jar
   ormi://oc4j_host:oc4j_ormi_port
   admin welcome -bindWebApp
   FAQApp FAQAppWeb http_web_site /FAQApp
```

このコードでは、http-web-site.xml 構成ファイルに <web-app> エントリが作成されます。admin.jar コマンドライン・ツールの詳細は、B-25 ページの「OC4J 管理 JAR のオプション」を参照してください。

Web アプリケーションの構成および管理と http-web-site.xml ファイルの詳細は、『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。

デプロイメントの詳細の説明

J2EE アプリケーションの開発は標準化されており、移植性がありますが、XML 構成ファイルはそうではありません。複数の XML ファイルを構成しないとアプリケーションを OC4J にデプロイできない場合もあります。必要なサーバー構成は、アプリケーションが使用するサービスによって異なります。たとえば、アプリケーションでデータベースが使用されている場合、data-sources.xml ファイルの DataSource オブジェクトを構成する必要があります。

FAQ デモなどの基本アプリケーションの場合、次の OC4J XML ファイルを構成します。

- META-INF/application.xml: application.xml ファイル内に、アプリケーションの標準 J2EE アプリケーション・ディスクリプタが含まれます。このファイルは、正しく構成されていて、デプロイされる J2EE EAR ファイル内に存在する必要があります。
- server.xml および http-web-site.xml: アプリケーションは server.xml ファイルに登録し、Web アプリケーションと Web アプリケーションで使用されるコンテキストは http-web-site.xml ファイル（または選択した他の *-web-site.xml ファイル）に登録します。
- data-sources.xml: アプリケーションで使用されるデータベースごとに、data-sources.xml ファイルの DataSource オブジェクトを構成する必要があります。

単純な J2EE アプリケーションを作成およびデプロイするには、次の基本ステップを実行します。

基本ステップ	FAQ アプリケーションのステップの説明
アプリケーションを作成または取得する。	OTN から FAQApp.zip をダウンロードします。
サーバー環境に必要な変更を加える。	JAVA_HOME 変数を設定します。
アプリケーションの XML 構成ファイルを変更する。	アプリケーションの XML ファイルはすべて、FAQ ZIP ファイルに収められています。
アプリケーションの標準の J2EE アプリケーション・ディスクリプタ・ファイルを更新する。	application.xml ファイルは、FAQApp.EAR ファイルに含まれています。
アプリケーションが含まれている EAR ファイルを（存在していない場合）ビルドする。	FAQ デモを変更する場合は、src ディレクトリ内で変更してから、ANT を使用して EAR ファイルをビルドします。
適切なサーバー XML ファイルにアプリケーションを登録する。	server.xml および http-web-site.xml ファイルを変更します。
使用されるデータベースを構成する。	data-sources.xml ファイルを変更します。

次の手順で、FAQ デモ・アプリケーションを OC4J にデプロイするために加える変更について説明します。

1. 前に述べたように、FAQ デモ・アプリケーションは Oracle OTN-J サイトからダウンロードできます。
2. サーバー環境に必要な変更を加えます。JAVA_HOME 変数を、Java 2 SDK のベース・ディレクトリに設定する必要があります。
3. web.xml などのアプリケーションの XML ファイルはすべて、正しく構成された状態で FAQ ZIP ファイルに収められています。
4. 標準の J2EE アプリケーション・ディスクリプタ・ファイルを更新します。FAQ デモ・アプリケーションの application.xml は、ZIP ファイルに収められています。OC4J は、application.xml ファイルを標準の J2EE アプリケーション・ディスクリプタ・ファイルとして使用します。
5. アプリケーションを含む EAR ファイルをビルドします。FAQ デモ・アプリケーションを変更し、ANT コマンドを使用して再ビルドできます。FAQ デモを再ビルドしてデプロイするには、次のコマンドを実行します。

   ```
   ant deploy
   ```

   ANT build.xml は、FAQ ZIP ダウンロードに含まれます。ANT ファイルの詳細は、次の Jakarta のサイトを参照してください。

   ```
   http://jakarta.apache.org/ant/
   ```

   再ビルドを行わない場合は、FAQApp.ear を ZIP ファイルから j2ee/home/applications にコピーします。この手順では、FAQ アプリケーションが OC4J サーバーに配置されます。
6. Oracle データベースの OC4J DataSource を構成します。バックエンド・データベースを指すように、正しい URL、ユーザー名およびパスワードを使用して、デフォルトのデータソース OracleDS を変更します。
7. J2EE アプリケーションを server.xml ファイルに、Web アプリケーションを http-web-site.xml に登録します。
8. j2ee/home/ ディレクトリから次のコマンドを実行して、OC4J を起動します。

```
java -jar oc4j.jar
```

OC4J の起動オプションの詳細は、1-4 ページの「OC4J の起動」を参照してください。

Web ブラウザを開き、次の URL を指定します。

```
http://oc4j_host:8888/FAQApp
```

OC4J XML 構成ファイルの詳細は、2-2 ページの「OC4J および J2EE の XML ファイルの概要」を参照してください。

アプリケーションのデプロイ

この項では、J2EE アプリケーションを OC4J サーバーにデプロイする方法と、そのアプリケーションに OC4J からアクセスできるようにアプリケーションをサーバーにバインドする方法について説明します。

EAR ファイルへのアプリケーションのアーカイブ

J2EE アプリケーションには、次のモジュールを含めることが可能です。

- Web アプリケーション

  Web アプリケーション・モジュール（WAR ファイル）には、サーブレットおよび JSP ページが含まれます。

- EJB アプリケーション

  EJB アプリケーション・モジュール（EJB JAR ファイル）には、Enterprise JavaBeans（EJB）が含まれます。

- JAR ファイルに含まれるクライアント・アプリケーション

エンタープライズ Java アプリケーションに属する JAR および WAR ファイルを、OC4J にデプロイできるよう、EAR ファイルにアーカイブします。J2EE 仕様で、EAR ファイルのレイアウトが定義されています。

EAR ファイルの内部レイアウトは、次のようにします。

アーカイブ・ディレクトリの形式

次の JAR コマンドを使用して、これらのファイルを *appname* ディレクトリにアーカイブします。

```
% jar cvfM appname.ear .
```

OC4J では、`application.xml` ファイルが標準の J2EE アプリケーション・ディスクリプタ・ファイルとして機能します。

本番環境での admin.jar を使用したデプロイ

OC4J には、J2EE アプリケーションをデプロイするためのコマンドライン・デプロイ・ツール、`admin.jar` コマンドが用意されています。このコマンドのオプションは、B-25 ページの「スタンドアロン OC4J のコマンドライン・オプションおよびプロパティ」に記載されています。必ず check-for-updates 属性を false に設定し、オート・デプロイを無効にしてください（1-5 ページの「OC4J の再起動」を参照）。

EAR ファイルを持つ J2EE アプリケーションをリモート・ノードにデプロイするには、次のように `admin.jar` を実行します。

```
java -jar admin.jar ormi://host:port
username password
-deploy
-file filename -deploymentName app_name
-targetPath path/destination
```

各項目の説明:

- *host:port* は、OC4J サーバーのホストおよびポートです。
- *username password* は、OC4J サーバーの管理ユーザー名およびパスワードです。
- `-file` *path/filename* は、EAR ファイルのローカル・ディレクトリおよびファイル名を示します。
- `-deploymentName` *app_name* 変数は、アプリケーションのユーザー定義名です。
- `-targetPath` *path/destination* は、EAR ファイルのデプロイ先となるサーバー・ノード上のパスを示します。デプロイ対象の EAR ファイルをコピーするディレクトリのターゲット・パスを指定します。デフォルトのパスは、`applications/` ディレクトリです。

注意: EAR ファイル内に Web アプリケーションがある場合は、`admin.jar -bindWebApp` オプションを使用して、Web アプリケーションをバインドします。

このデプロイ手順では、アプリケーションの `server.xml` ファイルに、次のような新規エントリが作成されます。

```
<application name=app_name path=path_EARfile auto-start="true" />
```

各項目の説明:

- `name` 属性は、アプリケーションの名前です。
- `path` は、EAR ファイルのディレクトリおよびファイル名を示します。
- `auto-start` 属性は、OC4J の再起動のたびに、このアプリケーションを自動的に再起動するかどうかを示します。

`server.xml` の要素の説明は、B-6 ページの「server.xml ファイルの要素」を参照してください。

Web アプリケーションのバインド

OC4J Web サーバーから J2EE Web アプリケーションにアクセスできるようにするには、`-bindWebApp` オプションを使用して、次のように Web アプリケーションを OC4J サーバーにバインドします。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port username password
-bindWebApp app_name web_app_name web_site_name context_root
```

`-bindWebApp` に指定する値は次のとおりです。

- *app_name* はアプリケーション名で、`-deploy` オプションの `-deploymentName` に使用されるのと同じ名前です。また、`server.xml` ファイルの `<application name=`*app_name*` />` 属性に保存される名前とも同じです。
- *web_app_name* は、EAR ファイルに格納されている WAR ファイルの名前から `.war` 拡張子を除いたものです。
- *web_site_name* は、Web アプリケーションのバインド先となる Web サイトを示す `*-web-site.xml` ファイルの名前です。これは、Web アプリケーション定義を受け取るファイルです。
- *context_root* は、Web モジュールのルート・コンテキストです。Web コンテキストには、Web アプリケーションのアクセス方法が定義されています。

この手順では、*web_site_name* 変数で指定された OC4J *-web-site.xml 構成ファイルに、エントリが作成されます。admin.jar の全オプションの一覧は、B-25 ページの「OC4J 管理 JAR のオプション」を参照してください。

開発環境でのアプリケーションの手動デプロイ

開発環境でアプリケーションをデプロイするには、XML ファイルを手動で変更します。必ず check-for-updates 属性を true に設定し、オート・デプロイを有効にしてください。

server.xml で、各 J2EE アプリケーションに対し、<application name=... path=... auto-start="true" /> エントリを新しく追加するか、または既存のエントリを変更します。パスには、完全なディレクトリ・パスおよび EAR ファイル名を指定してください。employee アプリケーションの例では、server.xml ファイルに次のエントリを追加します。

```
<application name="employee"
path="/private/applications/Employee.ear"
auto-start="true" />
```

Web アプリケーションを組み込んだ場合、次の作業を行って、Web アプリケーションを Web サーバーにバインドする必要があります。*-web-site.xml で、各 Web アプリケーションに対し、<web-app ...> エントリを追加します。application 属性の値は、server.xml ファイルで指定した値と同一になるようにします。name 属性は、Web アプリケーションの WAR ファイル名から WAR 拡張子を除いた部分です。

employee Web アプリケーションをバインドするには、次のエントリを追加します。

```
<web-app application="employee" name="Employee-web" root="/employee" />
```

デプロイの検証

OC4J は、server.xml へのアプリケーションの追加を検出します。アプリケーションがデプロイされたことを示すメッセージが OC4J サーバーに表示されます。メッセージが表示されたら、アプリケーションに対してリクエストを実行できます。

アプリケーションのアンデプロイ / 再デプロイの影響

OC4J インスタンスから J2EE アプリケーションをアンデプロイすると、次の影響があります。

- アプリケーションが OC4J ランタイムから削除され、アクセスできなくなります。
- Web アプリケーションのバインドは、すべてのバインド先の Web サイトから削除されます。
- すべてのアプリケーション・ファイルが、applications/ および application-deployments/ ディレクトリの両方から削除されます。

再デプロイ時に OC4J は、新しい EAR を再デプロイする前に、既存のアプリケーション（EAR/WAR）を削除します。たとえば、前のアプリケーションには含まれていたが新しいアプリケーションには含まれていない HTML ファイルにアクセスしようとすると、「見つかりません」というエラーが表示されます。

また、再デプロイされた WAR ファイルは、すでに開かれている WAR ファイルをオーバーレイするので、削除が必要な一部の古いファイルが新しいデプロイに残る可能性があります。たとえば、前のデプロイの静的 HTML ファイルは、新しい WAR には含まれなくても、開かれている WAR ディレクトリ構造には残っている可能性があるので、手動で削除する必要があります。

アプリケーションのホット・デプロイの影響

ホット・デプロイとは、本番アプリケーション・サーバーを再起動またはバウンスせずに、本番アプリケーション・サーバーにある EAR、WAR、JAR などのアーカイブ・ファイル、およびそれらと関連する XML ディスクリプタ・ファイルをデプロイするプロセスです。

実行中の OC4J インスタンスに EAR を再デプロイまたはホット再デプロイすると、前のアプリケーションから JVM にロードされるクラスは状況によって異なります。ファイル・システムのクラスまたは JAR ファイルが変更されたことをクラスローダーが認識して、クラスまたはライブラリを再ロードする場合もあります。また、JVM のチューニングによってガベージ・コレクタが既存のクラス定義をフラッシュできるかどうかによって、新しいクラス定義をロードするかどうかが決定される場合もあります。

セッション・データを含むシリアライズ・オブジェクトに関して、問題が発生する場合もあります。セッション・オブジェクトと関連するクラスが変更された場合、クラスの変更によってその変数が別のメモリー・フットプリントを占有している可能性があるため、汎用セッション・オブジェクトをクラスにキャストバックできなくなることがあります。これによってセッション・データが失われる可能性があります。

アクティブな OC4J インスタンスに新しい Web モジュールをデプロイしたときも、既存のセッションに悪影響を与える場合があります。具体的には、そのサーバー・インスタンス内で実行されているすべての Web アプリケーションの HTTP セッションが、デフォルトで失われます。

クラスタリングされていない OC4J インスタンスでは、各 Web アプリケーションの `orion-web.xml` ファイルで永続性ディレクトリを定義することにより、この問題を回避できます。既存の HTTP セッションは、アプリケーション・デプロイメント全体で、この一時ロケーションに格納されます。

各 `orion-web.xml` ファイル内のルート `<orion-web-app>` 要素の `persistence-path` 属性の値として、このディレクトリへの相対パスを指定します。たとえば、次のように入力します。

```
<orion-web-app ...
  persistence-path="persistDir"
  ...>
</orion-web-app>
```

この機能は、クラスタ環境内の OC4J インスタンスには使用できません。

デプロイ時の動作

アプリケーションをデプロイする際に `admin.jar` コマンドを使用したか、XML ファイルを編集したかには関係なく、次の処理が行われます。

OC4J が EAR ファイルをオープンし、ディスクリプタを読み取ります。

1. OC4J は、EAR ファイルに存在する `application.xml` を開いて解析します。`application.xml` ファイルには、EAR ファイルに含まれるすべてのモジュールが記載されています。OC4J は、これらのモジュールを確認して、EAR 環境を初期化します。
2. OC4J は、Web モジュール、EJB モジュール、コネクタ・モジュール、クライアント・モジュールの各モジュール・タイプ用モジュール・デプロイメント・ディスクリプタを読み取ります。J2EE ディスクリプタがメモリーに読み取られます。OC4J 固有のディスクリプタが含まれる場合は、それらのディスクリプタもメモリーに読み取られます。JAR および WAR ファイル環境が初期化されます。
3. OC4J は、デフォルトがある未構成の項目を確認して、そのデフォルトを適切な OC4J 固有のデプロイメント・ディスクリプタに書き込みます。このため、OC4J 固有のデプロイメント・ディスクリプタを指定していない場合は、デフォルトを書き込んだディスクプリプタを OC4J が提供することがわかります。OC4J 固有のデプロイメント・ディスクリプタを指定している場合は、OC4J が要素を追加する場合があります。
4. OC4J は、J2EE デプロイメント・ディスクリプタと OC4J 固有デプロイメント・ディスクリプタの両方に含まれる構成の詳細に対応します。OC4J は、インタフェースで Bean をラップするなど、OC4J での処理が必要な J2EE コンポーネント構成がないか確認します。
5. デフォルトを追加して必要な処理を実行した後に、OC4J は、新規モジュール・デプロイメント・ディスクリプタを `application-deployments/` ディレクトリに書き込みます。OC4J がアプリケーションを起動および再起動するときには、これらのディスクリプタを使用します。しかし、これらのディスクリプタを直接変更しないでください。必ず、マスターの場所にあるデプロイメント・ディスクリプタを変更してください。

6. OC4J は、EAR ファイルをマスターのディレクトリにコピーします。デフォルトは、applications/ ディレクトリです。ただし、admin.jar -targetPath オプションでマスター・ディレクトリの場所を指定できます。

注意： EAR ファイルを applications/ ディレクトリから削除せずに、admin.jar を使用してこの EAR ファイルをデプロイすると、新規のデプロイによって既存の EAR ファイルの名前の前にアンダースコアが付きます。EAR ファイルを上書きすることはありません。手作業で EAR ファイルを上書きすることができます。OC4J は、タイムスタンプの変更と再デプロイされたことを通知します。

---

7. 最後に、OC4J はこのアプリケーションがデプロイされたことを server.xml ファイルに表記します。

Web アプリケーションのアンデプロイ

admin.jar コマンドライン・ツールで -undeploy オプションを使用すると、OC4J Web サーバーから J2EE Web アプリケーションを削除できます。構文は次のとおりです。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin adminpassword
-undeploy applicationName -keepFiles
```

このコマンドでは、applicationName という名前のデプロイ済 J2EE アプリケーションが削除され、次の処理が行われます。

- アプリケーションが OC4J ランタイムから削除されます。
- Web モジュールのバインドは、すべてのバインド先の Web サイトから削除されます。
- アプリケーション・ファイルは、applications/ および application-deployments/ ディレクトリの両方から削除されます。これらのファイルを削除しない場合は、-keepFiles スイッチを使用します。

2

高度な構成および開発

この章では、スタンドアロン・モードの OC4J を開発目的で管理する方法について説明します。J2EE アプリケーションの構成、開発およびデプロイを最も簡単に行う方法については、第 1 章「構成およびデプロイ」で説明しています。しかし、その他の JMS などのサービスを使用する場合は、XML 構成ファイルの操作方法を知っておく必要があります。

この章には、次の項目が含まれています。

OC4J および J2EE の XML ファイルの概要

この項には、次の項目が含まれています。

XML 構成ファイルの概要

OC4J は XML ファイルのみを使用して構成するため、一連の XML ファイルの役割と手法を理解しておく必要があります。各 XML ファイルは、特定の役割を果たすために存在します。したがって、必要な役割がわかれば、どの XML ファイルを変更およびメンテナンスすればよいかが理解できます。

図 2-1 に、OC4J のすべての XML ファイル、およびそれぞれの役割を示します。

- OC4J サーバー : このボックス内のすべての XML ファイルは、OC4J サーバーのこのインスタンスの設定に使用されます。これらのファイルは、リスニング・ポート、管理パスワード、セキュリティおよびその他の基本的な J2EE サービスなどを構成します。

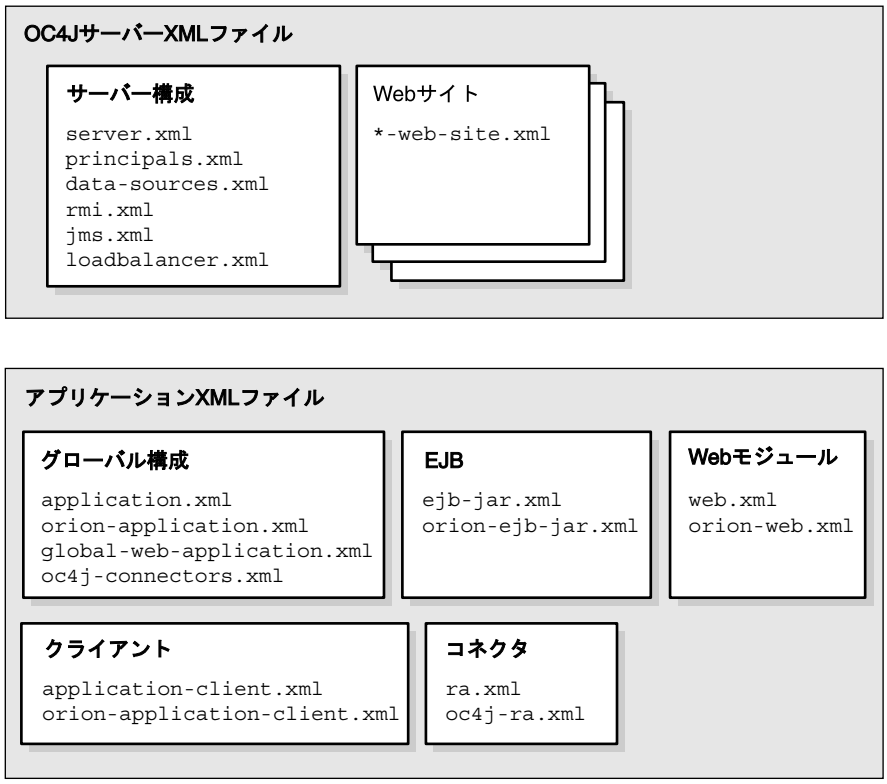
  OC4J サーバーの構成ファイルは `j2ee/home/config/` ディレクトリにあります。これらのファイルは、OC4J サーバーを構成し、その他の主要な構成ファイルを指します。OC4J 構成ファイルの設定は、デプロイされた J2EE アプリケーションに直接関係するのではなく、サーバーそのものに関係します。

- Web サイト : これらの XML ファイルは、OC4J Web サイトのリスニング・ポート、プロトコルおよび Web コンテキストを構成します。

- アプリケーション XML ファイル : 各 J2EE アプリケーション・タイプ（EJB、サーブレット、JSP およびコネクタ）は、独自の構成（デプロイメント）ファイルを必要とします。各アプリケーション・タイプには、J2EE デプロイメント・ディスクリプタと OC4J 固有デプロイメント・ディスクリプタが 1 つずつあり、それぞれに "`orion-`" という接頭辞が付けられています。また、次のファイルは、アプリケーションのあらゆる構成要素のためのグローバル構成ファイルです。

  - この OC4J インスタンス内のすべてのアプリケーションに対する共通設定を含む、グローバル・アプリケーション構成ファイルの `application.xml`。
  - この OC4J インスタンス内のすべてのアプリケーションに対する OC4J 固有のグローバル・アプリケーション情報を含む、`orion-application.xml` ファイル。
  - この OC4J インスタンス内のすべての Web モジュールに対する、共通設定などの OC4J 固有のグローバル Web アプリケーション構成情報を含む `global-web-application.xml` ファイル。
  - グローバル・コネクタ構成情報を含む `oc4j-connectors.xml` ファイル。

図 2-1 OC4J および J2EE のアプリケーション・ファイル



注意： デプロイされた各アプリケーションは、標準の J2EE アプリケーション・ディスクリプタ・ファイルとして `application.xml` を使用します。その XML ファイルは、アプリケーションにローカルでのみ適用され、`j2ee/home/config` ディレクトリの `application.xml` とは別のものです。`j2ee/home/config/application.xml` ファイルは、この OC4J サーバー・インスタンスでデプロイするすべてのアプリケーションに適用されるオプションを構成します。

表 2-1 に、前出の図で示した各 XML ファイルの役割および機能を示します。

表 2-1 OC4J の機能および構成要素

XML 構成ファイル	機能 / 構成要素
`server.xml`	OC4J の全般的なサーバー構成。サーバーを構成し、このファイルに追加する XML ファイル（JMS サポート用の `jms.xml` など）を指定します。一覧表示されたその他の XML ファイルによって、サービスを別々のファイルに構成することが可能になり、`server.xml` ファイルではそれらのファイルを OC4J 構成に使用することを示します。
`principals.xml`	サーバー接続に必要なセキュリティ・タイプの OC4J セキュリティ構成。
`data-sources.xml`	OC4J 内のアプリケーションによって使用されているすべてのデータベースの OC4J データソース構成
`rmi.xml`	OC4J RMI ポート構成と HTTP 上での RMI トンネリング

表 2-1 OC4J の機能および構成要素（続き）

XML 構成ファイル	機能 / 構成要素
jms.xml	OC4J 内で JMS および MDB によって使用されている Destination のトピックおよびキューの OC4J JMS 構成。
*-web-site.xml	OC4J の Web サイト定義 Web サイトはそれぞれ専用の XML ファイル内で定義されます。各 XML ファイルに、ルート要素名 <web-site> に準じた名前を付けることをお薦めします。たとえば、*-web-site.xml を my-web-site.xml とします。通常、グローバル Web サイトの定義は http-web-site.xml にあります。各 Web サイトの XML ファイルを server.xml ファイル内の専用の web-site path 文で指定する必要があります。
application.xml orion-application.xml	J2EE アプリケーションの標準の J2EE アプリケーション・ディスクリプタ・ファイルおよび構成ファイル。 ■ グローバル application.xml ファイルは、j2ee/home/config ディレクトリに存在し、この OC4J インスタンス内のアプリケーションすべてに対する共通設定を含みます。このファイルには、セキュリティ XML 定義ファイル principals.xml の場所を定義します。これは、ローカル application.xml ファイルとは異なる XML ファイルです。 ■ ローカル application.xml ファイルには、J2EE アプリケーション・モジュールを含む J2EE EAR ファイルを定義します。このファイルは、J2EE アプリケーション EAR ファイル内に存在します。 ■ orion-application.xml ファイルは、すべてのアプリケーションに対する OC4J 固有の定義です。
global-web-application.xml web.xml orion-web.xml	J2EE の Web アプリケーションの構成ファイル。 ■ global-web-application.xml は、OC4J 固有のファイルで、すべての Web サイトにバインドされているサーブレットの構成に使用されます。 ■ web.xml および orion-web.xml が各 Web アプリケーションに対して存在します。 web.xml ファイルは、Web アプリケーションのデプロイ・パラメータの定義に使用され、WAR ファイル内に含まれています。さらに、このファイル内で、サーブレットおよび JSP の URL パターンを指定できます。たとえば、サーブレットは <servlet> 要素で定義され、URL パターンは <servlet-mapping> 要素で定義されます。
ejb-jar.xml orion-ejb-jar.xml	J2EE の EJB アプリケーションの構成ファイル。 ejb-jar.xml ファイルは、EJB のデプロイメント・ディスクリプタの定義に使用され、EJB JAR ファイル内に含まれています。
application-client.xml orion-application-client.xml	J2EE のクライアント・アプリケーションの構成ファイル。

表 2-1 OC4J の機能および構成要素（続き）

XML 構成ファイル	機能 / 構成要素
oc4j-connectors.xml ra.xml oc4j-ra.xml	コネクタの構成ファイル。 ■ oc4j-connectors.xml ファイルには、グローバルな OC4J 固有のコネクタ構成が含まれます。 ■ ra.xml ファイルには、J2EE 構成が含まれます。 ■ oc4j-ra.xml ファイルには、OC4J 固有の構成が含まれます。

XML ファイルの相互関連性

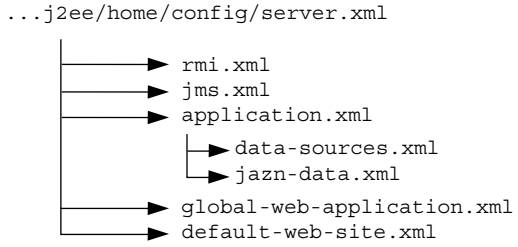
これらの XML ファイルの一部は、相互に関連性があります。つまり、一部の XML ファイルは、他の XML ファイル（OC4J 構成および J2EE アプリケーションの両方）を参照します（図 2-3 を参照）。

次に、相互関連ファイルを示します。

- server.xml: 次のファイルへの参照が含まれています。
  - デフォルトの http-web-site.xml ファイルを含む、この OC4J サーバーの各 Web サイトのすべての *-web-site ファイル。
  - 他の OC4J サーバー構成ファイルそれぞれの場所。ただし、図 2-1 に示すように、グローバル application.xml で定義されている principals.xml は除きます。
  - OC4J でデプロイされている各 J2EE アプリケーション用の application.xml ファイルの場所
- http-web-site.xml: server.xml ファイルで定義されるように、名前でアプリケーションを参照します。また、このファイルはアプリケーション固有の EAR ファイルを参照します。
- application.xml: principals.xml ファイルの参照を含みます。

server.xml ファイルは、OC4J サーバーで使用されている大部分のファイルへの参照を含んでいる中核ファイルです。図 2-2 に、server.xml ファイルで参照される可能性のある XML ファイルを示します。

図 2-2 server.xml 内で参照される XML ファイル



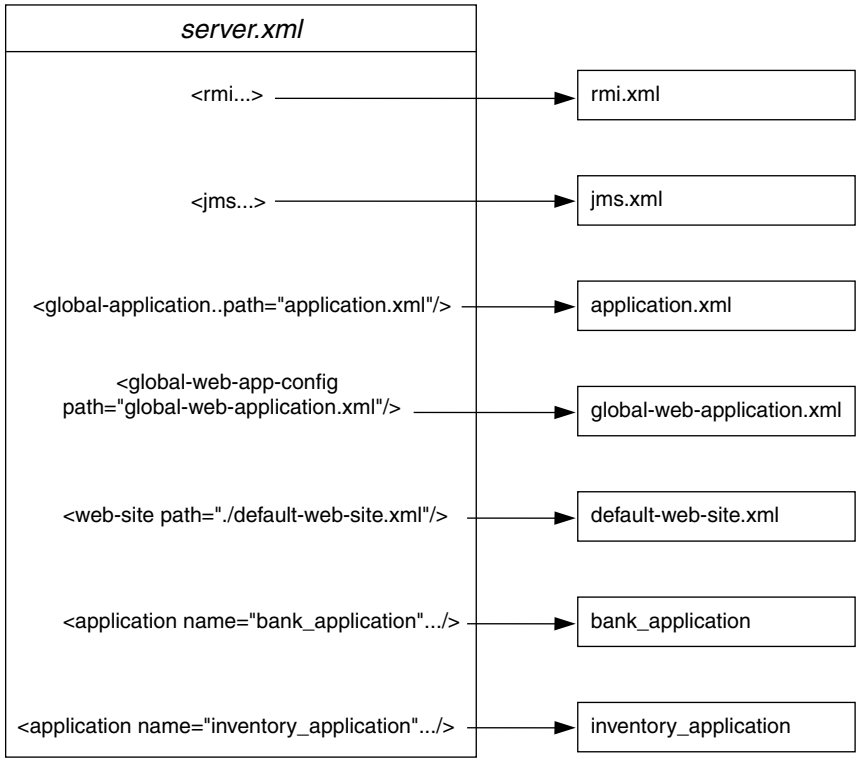
server.xml で他の XML 構成ファイルを指定する方法を図 2-3 に示します。各 XML ファイルの場所は、絶対パスまたは server.xml ファイルが存在する場所に対する相対パスで指定します。また、XML ファイルの名前は、そのファイル内容が適切な DTD に準拠していればどのような名前でもかまいません。

- <rmi-config> 要素は、rmi.xml ファイルの名前と場所を示します。
- <jms-config> 要素は、jms.xml ファイルの名前と場所を示します。
- <global-application> 要素は、グローバル application.xml ファイルの名前と場所を示します。

- <global-web-app-config> 要素は、global-web-application.xml ファイルの名前と場所を示します。
- <web-site> 要素は、*-web-site.xml ファイルの名前と場所を示します。複数の Web サイトを使用できるため、複数の <web-site> エントリを指定できます。

OC4J サーバー構成ファイルの指定に加えて、server.xml ファイルでは、この OC4J サーバーにデプロイされたアプリケーションも指定します。アプリケーションをデプロイするには、admin.jar コマンドで -deploy オプションを使用するか、server.xml ファイルを直接変更します。デプロイされた各アプリケーションは、<application> 要素で示します。server.xml ファイルの直接編集の詳細は、2-7 ページの「開発環境でのアプリケーションの手動追加」を参照してください。

図 2-3 server.xml ファイルおよび関連 XML ファイル



server.xml の他の要素については、B-6 ページの「server.xml ファイルの要素」を参照してください。

ライブラリの共有

複数のアプリケーションでライブラリを共有する場合は、次のように <library> 要素をグローバル application.xml ファイルに追加し、ライブラリを配置するディレクトリを指定します。

Windows の場合 :

```
<library path="d:¥oc4j¥j2ee¥home¥applib¥"/>
```

UNIX の場合 :

```
<library path="/private/oc4j/j2ee/home/applib/"/>
```

次のように、要素を含めるディレクトリごとに、別々の行で別々の <library> 要素を使用します。

```
<library path="/private/oc4j/j2ee/home/applib/"/>
<library path="/private/oc4j/j2ee/home/mylibrary/"/>
```

デフォルトでは、<library> 要素は、j2ee/home/applib ディレクトリ内のグローバル application.xml ファイルに存在します。<library> 要素を変更して他のディレクトリを含めるようにするかわりに、ライブラリを applib ディレクトリ内に移動することもできます。ただし、このディレクトリにライブラリを追加すると、OC4J のサイズが増加し、不明なクラスの検索時にはすべてのライブラリが検索されるために、パフォーマンスに影響が出ます。この手順を使用する場合は注意してください。

注意： デフォルトの j2ee/home/applib ディレクトリは OC4J のインストール時には作成されません。このディレクトリに共有ライブラリを追加する場合は、ライブラリを追加する前に、ディレクトリを作成する必要があります。

可能なかぎり、共有ライブラリは、アプリケーションとともにデプロイされる orion-application.xml ファイルを使用してアプリケーションでローカルに保持することをお薦めします。アプリケーションの orion-application.xml ファイルに <library> 要素を追加することで、ライブラリの場所を指定できます。このライブラリは、このアプリケーション内でのみ使用されます。

開発環境でのアプリケーションの手動追加

開発環境では、開発のたびに admin.jar コマンドを使用するよりも XML ファイルを変更する方が簡単です。次の各項は、XML 構成ファイルの変更方法を理解するのに役立ちます。

- リスナーの構成
- J2EE アプリケーションの構成

リスナーの構成

各 OC4J サーバーは、受信リクエストがないかどうか HTTP または RMI プロトコルをリスニングするよう構成されます。各 OC4J Web サーバーは、専用の *-web-site.xml ファイル内で構成されます。

- HTTP プロトコル・リスナー : HTTP クライアントは、OC4J の HTTP リスナーに直接アクセスできます。そのためには、HTTP リスナー・ポートを示す http-web-site.xml ファイルを構成する必要があります。デフォルトの HTTP ポートは 8888 です。http-web-site.xml 内の、ポート番号 8888 の HTTP リスナーのエントリを次に示します。

  ```
  <web-site host="oc4j_host" port="8888" protocol="http"
   display-name="Default OC4J WebSite">
  ```

- RMI プロトコル・リスナー : EJB クライアント、および admin.jar などの OC4J ツールは、構成済の RMI ポートを通じて OC4J サーバーにアクセスします。したがって、rmi.xml ファイルを構成する必要があります。デフォルトの RMI ポートは 23791 です。rmi.xml ファイルでのデフォルト RMI ポートの構成を次に示します。

  ```
  <rmi-server port="23791" >
  ```

J2EE アプリケーションの構成

J2EE アプリケーションを構成およびデプロイするには、アプリケーションの情報で server.xml および http-web-site.xml ファイルを変更します。

- `server.xml` では、OC4J の起動時に自動的に起動する各アプリケーションに対し、`<application name=... path=... auto-start="true" />` エントリを新しく追加するか、または既存のエントリを変更します。path には、デプロイする EAR ファイルの場所か、またはアプリケーションが構築された展開ディレクトリのいずれかを指定します。詳細は、1-13 ページの「本番環境での admin.jar を使用したデプロイ」または 2-8 ページの「ディレクトリ内での構築およびデプロイ」を参照してください。
- `http-web-site.xml` では、OC4J の起動時に Web サイトにバインドする各 Web アプリケーションに対し、`<web-app...>` エントリを追加します。`name` 属性には WAR ファイル名（`.war` 拡張子を除いた部分）を指定するため、J2EE アプリケーションの WAR ファイル 1 つにつき 1 行必要です。

  WAR ファイルを使用して Web アプリケーションをバインドする場合、次のエントリを追加します。

  ```
  <web-app application="myapp" name="myapp-web" root="/myapp" />
  ```

  - `application` 属性は、`server.xml` でアプリケーション名として指定されている名前です。
  - `name` 属性は、WAR ファイル名から `.WAR` 拡張子を除いた部分です。
  - `root` 属性は、Web サイト外でのアプリケーションのルート・コンテキストを定義します。たとえば、Web サイトを `http://oc4j_host:8888` と定義した場合、アプリケーションを起動するには、ブラウザに `http://oc4j_host:8888/myapp` と指定します。

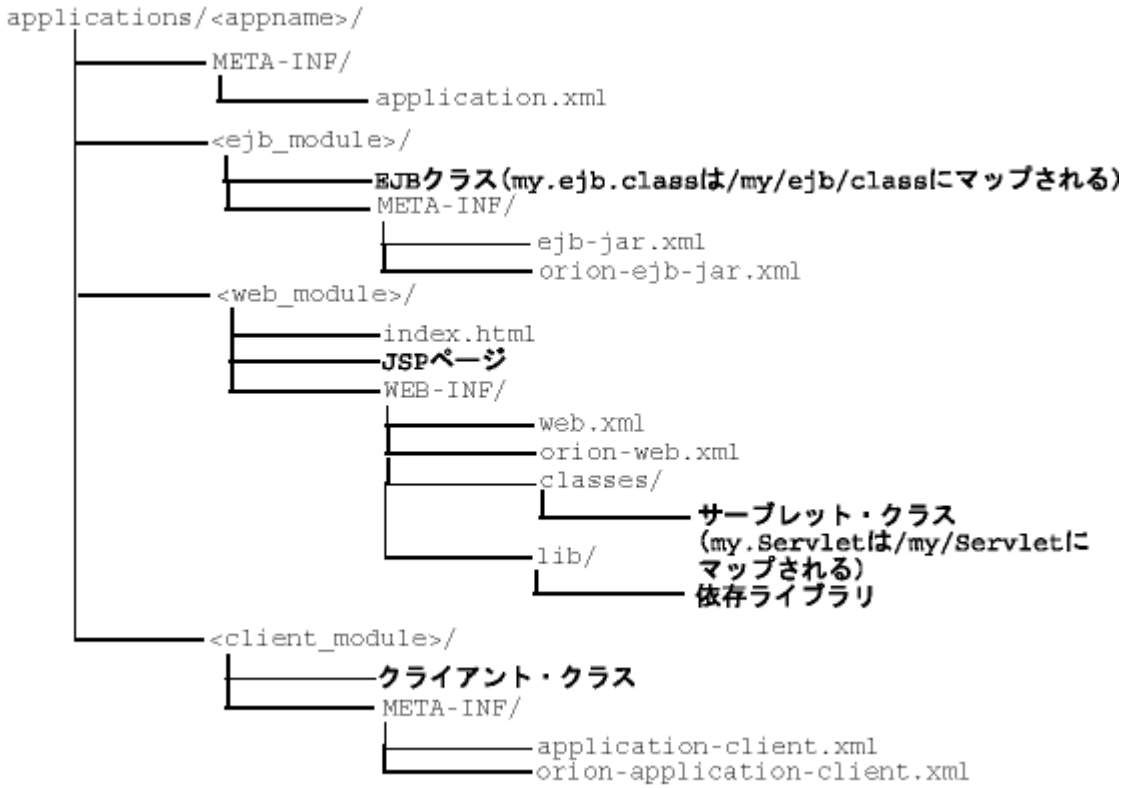
  注意： 自動起動が完了するのを待ってからクライアントにアクセスしてください。これらの処理が完了する前にクライアントがルックアップを行うと、失敗します。

ディレクトリ内での構築およびデプロイ

アプリケーションの開発時には、クラスをすばやく修正、コンパイルおよび実行する必要があります。OC4J では、開かれたディレクトリ形式でアプリケーションを開発しているときにアプリケーションを自動的にデプロイできます。「開発アプリケーションのディレクトリ構造」の *appname* で示されるトップ・ディレクトリのタイムスタンプが変わると、アプリケーションが自動的にデプロイされます。このディレクトリを server.xml がマスターの場所として認識します。

アプリケーションは、JAR、WAR および EAR ファイルに必要な階層形式と同じ階層形式のマスター・ディレクトリに配置する必要があります。たとえば、*appname* が J2EE アプリケーションの存在するディレクトリである場合、必要なディレクトリ構造は図 2-4 のようになります。

図 2-4 開発アプリケーションのディレクトリ構造



開発アプリケーションのディレクトリ構造

EJB または複合 J2EE アプリケーションを開かれたディレクトリ形式でデプロイするには、次の手順を実行します。

1. 任意のディレクトリにファイルを配置します。図 2-4 は、`j2ee/home/applications/`*`appname`*`/` に配置されたアプリケーションを示しています。*appname* 下のディレクトリ構造は、EAR ファイル内で使用されるディレクトリ構造と次のように類似しています。

   a. EJB JAR ファイル名、Web アプリケーション WAR ファイル名、クライアント JAR ファイル名およびリソース・アダプタ・アーカイブ（RAR）ファイル名をそれぞれのモジュールの表示用に選択したディレクトリ名に置き換えます。図 2-4 では、これらのディレクトリ名を *ejb_module/*、*web_module/*、*client_module/* および *connector_module/* で示します。

   b. 各モジュール用クラスを、そのパッケージ構造にマッピングされる、ディレクトリ構造内の場所に入れます。

2. `server.xml`、`application.xml` および `*-web-site.xml` ファイルを、次のように変更します。`server.xml` および `*-web-site.xml` ファイルは `j2ee/home/config` ディレクトリに置かれ、`application.xml` は `j2ee/home/applications/<appname>/META-INF` ディレクトリに置かれます。これらのファイルを次のように変更します。

   - `server.xml` で、各 J2EE アプリケーションに対し、`<application name=... path=... auto-start="true" />` 要素を新しく追加するか、または既存の要素を変更します。パスは、マスターのアプリケーション・ディレクトリを指します。図 2-4 では、`j2ee/home/applications/`*`appname`*`/` に該当します。

     パスは、次のいずれかの方法で指定します。

     * ルートから親ディレクトリへのフルパスを指定します。

図 2-4 の例では、*appname* が "myapp" の場合、絶対パスは次のようになります。

```
<application_name="myapp"
    path="/private/j2ee/home/applications/myapp"
    auto-start="true" />
```

* 相対パスを指定します。このパスは、親ディレクトリの場所に対する server.xml ファイルの相対的な場所を示します。

  図 2-4 の例で、*appname* が myapp の場合、相対パスは次のようになります。

```
<application_name="myapp" path="../applications/myapp" auto-start="true" />
```

- application.xml で、<module> 要素に JAR または WAR ファイルではなくて各モジュールのディレクトリ名を指定します。application.xml ファイルの <web-uri>、<ejb> および <client> 要素は、これらのモジュールが存在するディレクトリを指定するように変更する必要があります。これらの要素に含まれるパスは、マスターのディレクトリの相対パスであり、これらの各アプリケーション・タイプの WEB-INF または META-INF ディレクトリの親ディレクトリである必要があります。

  たとえば、図 2-4 の *web_module*/ ディレクトリが myapp-web/ の場合、次の例では、これを <web-uri> 要素内で Web モジュール・ディレクトリとして指定しています。

```
<module>
  <web>
    <web-uri>myapp-web</web-uri>
  </web>
</module>
```

- *-web-site.xml ファイルで、Web アプリケーションごとに、<web-app...> 要素を追加します。これによって Web アプリケーションが Web サイトにバインドされるため、これは重要です。アプリケーションの属性値は、server.xml ファイルの値と同じである必要があります。name 属性は、Web アプリケーションのディレクトリにする必要があります。name 要素内のディレクトリ・パスは、application.xml ファイル内の <web-uri> 要素のパスの場合と同じ規則に従う必要があります。

  Web アプリケーション "myapp" をバインドするには、次のパスを追加します。

```
<web-app application="myapp" name="myapp-web" root="/myapp" />
```

注意： EAR ファイルを使用してデプロイすると、パフォーマンスが向上します。実行中は、EAR ファイル全体がメモリーにロードされ、索引付けされます。この方法は、必要時に開発ディレクトリからクラスを読み取るより高速です。

OC4J でのアプリケーションのオート・デプロイ

OC4J では、EAR ファイルのタイムスタンプが変わると、アプリケーションが自動的にデプロイされます。アプリケーションをデプロイまたは再デプロイするために OC4J を再起動する必要はありません。オート・デプロイは必ず有効になるわけではなく、次の場合にのみ行われます。

- EAR ファイルの変更がチェックされた場合

  EAR ファイルを変更すると、変更が自動的に検出されます。OC4J は、タイムスタンプの変更とアプリケーションの再デプロイを検出します。

- 2-8 ページの「ディレクトリ内での構築およびデプロイ」で説明した展開ディレクトリ形式（*appname* ディレクトリ）を取る特定の XML ファイルのタイムスタンプが変わった場合。展開ディレクトリ・アプリケーションをオート・デプロイするためには、次の手順を実行する必要があります。

1. <module> 内のクラスを変更し、その J2EE デプロイメント・ディスクリプタに手を加えて XML ファイルのタイムスタンプを変更します。たとえば、サーブレット・クラスを変更する場合は web.xml ファイルに手を加えます。そうすると、この <module> の変更を OC4J が認識します。
2. このアプリケーションの application.xml に手を加えます。application.xml のタイムスタンプを変更することで、オート・デプロイが開始します。OC4J は、起動後、どのモジュールのデプロイメント・ディスクリプタでタイムスタンプが変わっているかを認識し、再デプロイするモジュールをチェックします。

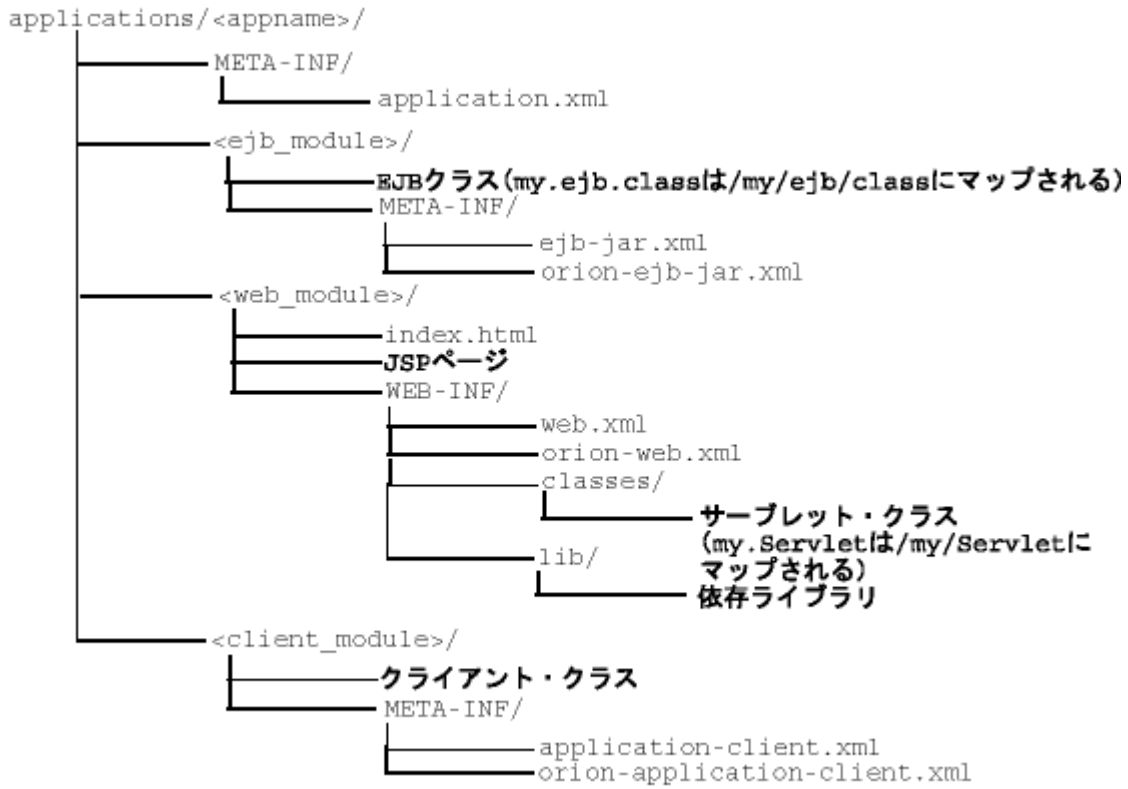
OC4J が更新をチェックしない場合は、admin.jar コマンドライン・ツールを使用するか、OC4J サーバーを手動で再起動して、再デプロイを行います。-deploy オプションの詳細は、B-25 ページの「OC4J 管理 JAR のオプション」を参照してください。

デプロイ後の XML ファイルの変更

アプリケーションをデプロイするたびに、デフォルトの要素を含む OC4J 固有の XML ファイルが自動的に生成されます。このファイルを変更する、または既存の XML ファイルに追加する場合、XML ファイルをアプリケーションの当初の開発ディレクトリにコピーし、そこで変更を行う必要があります。デプロイされた場所で XML ファイルを変更すると、アプリケーションを再デプロイしたときに、変更内容が上書きされてしまいます。変更内容が維持されるのは、開発ディレクトリで変更を加えた場合のみです。

OC4J 固有の XML ファイルはすべて、図 2-5 に示す推奨開発構造に追加できます。

図 2-5 開発アプリケーションのディレクトリ構造



アプリケーションの親の指定

子アプリケーションは、自分の親アプリケーションのネームスペースを参照します。このように、アプリケーションを親として設定することにより、複数の子の間でサービスを共有できます。デフォルトの親は、グローバル・アプリケーションです。

アプリケーションを別のアプリケーションの親として設定するには、次のいずれかを実行します。

- 元のアプリケーションをデプロイする際に、admin.jar コマンドの -parent オプションを使用します。このオプションでは、デプロイするアプリケーションの親になるアプリケーションを指定できます。
- server.xml ファイルのアプリケーション定義行に親を指定します。アプリケーションはそれぞれ、server.xml ファイルの <application> 要素で定義されています。この要素の parent 属性に親アプリケーションを指定します。

```
<application ... parent="applicationWithCommonClasses" .../>
```

起動クラスおよび停止クラスの開発

OC4J の初期化後または OC4J の終了前にコールされるクラスを開発できます。起動クラスは、OC4J の初期化後にサービスを起動して機能を実行し、停止クラスは、OC4J の終了前にこれらのサービスを終了して機能を実行します。これらのクラスをコンパイルする場合、oc4j.jar が Java の CLASSPATH に含まれている必要があります。

OC4J は、server.xml ファイル内のこれらのクラスの構成に基づいて、OC4J 起動クラスおよび停止クラスをデプロイおよび実行します。

- OC4J 起動クラス
- OC4J 停止クラス

OC4J 起動クラス

起動クラスは、OC4J の初期化後に 1 度のみ実行されます。server.xml ファイルが読み込まれるたびに再実行されるわけではありません。起動クラスは、preDeploy および postDeploy という 2 つのメソッドを含む com.evermind.server.OC4JStartup インタフェースを実装します。ここには、サービスの開始およびその他の初期化ルーチンを実行するコードを実装できます。

- preDeploy メソッドは、OC4J アプリケーションの初期化前に実行されます。
- postDeploy メソッドは、すべての OC4J アプリケーションが初期化された後に実行されます。

各メソッドは 2 つの引数を必要とします。Hashtable は構成から移入され、JNDI Context では、Context 内に含まれるプロセス値をバインドできます。両方のメソッドが文字列を戻しますが、現在のところは無視されます。

起動クラスを作成した場合、server.xml ファイルの <startup-classes> 要素内で構成する必要があります。各 OC4JStartup クラスは、<startup-classes> 要素内の 1 つの <startup-class> 要素に定義されます。各 <startup-class> では次のものを定義します。

- com.evermind.server.OC4JStartup インタフェースを実装するクラスの名前。
- 障害が致命的かどうか。致命的な障害の場合、例外がスローされた時点で、OC4J は例外を記録して終了します。致命的な障害でない場合、OC4J は例外を記録して処理を続行します。デフォルトは致命的な障害ではありません。
- 実行の順序。各起動クラスは、クラスの実行順序を指定する整数番号を受け取ります。
- OC4J が受け取る、String 型のキーと値のペアを含む初期化パラメータ。入力された Hashtable 引数の中で指定されます。JNDI を使用して各値をその名前とバインドするため、キーと値のペアの名前は一意である必要があります。

server.xml ファイルの <init-library path="../[xxx]" /> 要素で、起動クラスを配置するディレクトリを構成するか、またはクラスがアーカイブされるディレクトリと JAR ファイル名を構成します。path 属性には絶対パス、または j2ee/home/config の相対パスを指定できます。

例 2-1 起動クラスの例

TestStartup クラスの構成は、server.xml ファイルの <startup-class> 要素に含まれています。構成では次のように定義します。

- failure-is-fatal 属性を true に設定し、例外により OC4J が終了するようにします。
- execution-order を 0（ゼロ）に設定し、これが実行される最初の起動クラスであることを示します。
- String タイプの 2 つの初期化キーと値のペアを定義します。これは、次の Hashtable に移入されます。

```
"oracle.test.startup" "true"
"startup.oracle.year" "2002"
```

> **注意：** JNDI により名前がその値にバインドされるため、キーと値のペアの名前は、すべての起動クラスおよび停止クラスで一意にする必要があります。

このため、server.xml ファイルに次のように構成して TestStartup クラスを定義します。

```
<startup-classes>
        <startup-class classname="TestStartup" failure-is-fatal="true">
            <execution-order>0</execution-order>
                <init-param>
                    <param-name>oracle.test.startup</param-name>
                    <param-value>true</param-value>
                </init-param>
                <init-param>
                    <param-name>startup.oracle.year</param-name>
                    <param-value>2002</param-value>
                </init-param>
        </startup-class>
 </startup-classes>
```

コンテナは、入力した Hashtable パラメータ内で、起動クラスに 2 つの初期化キーと値のペアを提供します。

次の例は、com.evermind.server.OC4JStartup インタフェースを実装する TestStartup を示しています。preDeploy メソッドは Hashtable からキーと値のペアを取得して出力します。postDeploy メソッドは NULL のメソッドです。TestStartup をコンパイルする場合、oc4j.jar が Java の CLASSPATH に含まれている必要があります。

```
import com.evermind.server.OC4JStartup;

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

public class TestStartup implements OC4JStartup {
    public String preDeploy(Hashtable args, Context context) throws Exception {
        // bind each argument using its name
        Enumeration keys = args.keys();
        while(keys.hasMoreElements()) {
            String key = (String)keys.nextElement();
            String value = (String)args.get(key);
            System.out.println("prop: " + key + " value: " + args.get(key));
            context.bind(key, value);
        }
```

```
        return "ok";
    }

    public String postDeploy(Hashtable args, Context context) throws Exception {
        return null;
    }
}
```

TestStartup クラスが "../app1/startup.jar" にアーカイブされているものと仮定し、server.xml ファイルの <init-library> 要素を次のように変更します。

```
<init-library path="../app1/startup.jar" />
```

OC4J を起動すると、すべてのアプリケーションの初期化前に preDeploy メソッドが実行されます。OC4J は Hashtable からの値を使用して JNDI コンテキストを移入します。TestStartup が例外をスローすると、failure-is-fatal 属性が TRUE に設定されているため、OC4J は終了します。

OC4J 停止クラス

停止クラスは OC4J の終了前に実行されます。停止クラスでは、preUndeploy および postUndeploy という 2 つのメソッドを含む com.evermind.server.OC4JShutdown インタフェースを実装します。ここには、サービスの停止およびその他の終了ルーチンを実行するコードを実装できます。

- preUndeploy メソッドは、OC4J アプリケーションが終了する前に実行されます。
- postUndeploy メソッドは、すべての OC4J アプリケーションが終了した後に実行されます。

各メソッドは 2 つの引数を必要とします。Hashtable は構成から移入され、JNDI Context では、Context 内に含まれるプロセス値をバインドできます。

実装と構成は、2-12 ページの「OC4J 起動クラス」で説明した停止クラスと同じですが、<shutdown-classes> および <shutdown-class> 要素内に構成が定義されることと、failure-is-fatal 属性がないことが異なります。このため、TestShutdown クラスの構成は次のようになります。

```
<shutdown-classes>
        <shutdown-class classname="TestShutdown">
            <execution-order>0</execution-order>
                <init-param>
                    <param-name>oracle.test.shutdown</param-name>
                    <param-value>true</param-value>
                </init-param>
                <init-param>
                    <param-name>shutdown.oracle.year</param-name>
                    <param-value>2002</param-value>
                </init-param>
        </shutdown-class>
 </shutdown-classes>
```

TestShutdown クラスが ../app1/shutdown.jar にアーカイブされているものと仮定し、別の <init-library> 要素を server.xml ファイルに次のように追加します。

```
<init-library path="../app1/shutdown.jar" />
```

パフォーマンス・オプションの設定

パフォーマンスの設定のほとんどは『Oracle Application Server パフォーマンス・ガイド』に記載されています。

OC4J コマンドライン・オプションを使用するか、該当する XML ファイル要素を編集して、ユーザー自身でこれらのパフォーマンス設定を管理できます。

パフォーマンス関連のコマンドライン・オプション

`dedicated.rmicontext` オプションを除き、各 `-D` コマンドライン・オプションでは、推奨設定値がデフォルトになります。しかし、OC4J オプションとして各 `-D` コマンドライン・オプションを指定することにより、これらのオプションを変更できます。この例は、B-25 ページの「スタンドアロン OC4J のコマンドライン・オプションおよびプロパティ」を参照してください。

- `dedicated.rmicontext=true/false`。デフォルト値は false です。これにより、すでに使用されなくなった `dedicated.connection` 設定が置き換えられます。同一プロセス内の複数のクライアントが `InitialContext` を取り出すと、OC4J はキャッシュされているコンテキストを返します。したがって、各クライアントは、そのプロセスに割り当てられている同じ `InitialContext` を受け取ります。結果的にサーバーのロード・バランシングにつながるサーバー参照は、クライアントが独自の `InitialContext` を取り出すときにのみ発生します。`dedicated.rmicontext=true` を設定すると、各クライアントは共有コンテキストではなくそのクライアント独自の `InitialContext` を受け取ります。各クライアントに独自の `InitialContext` がある場合、クライアントのロード・バランシングが可能です。

  このパラメータはクライアント用です。JNDI プロパティで設定することもできます。

- `oracle.dms.sensors=[none, normal, heavy, all]`。Oracle Application Server 組込みのパフォーマンス・メトリックの値を、none（オフ）、normal（中程度のメトリック）、heavy（大きなメトリック）または all（可能なすべてのメトリック）に設定できます。デフォルトは normal です。このパラメータは OC4J サーバーで設定する必要があります。これらのパフォーマンス・メトリックをオンにするための以前のメソッドである `oracle.dms.gate=true/false` は、`oracle.dms.sensors` 変数で置き換えられました。ただし、`oracle.dms.gate` を使用している場合、この変数を false に設定すると、`oracle.dms.sensors=none` の設定と同じ意味になります。

- `DefineColumnType=true/false`。デフォルトは false です。9.2 より前の Oracle JDBC ドライバを使用している場合は、true に設定してください。これらのドライバの場合、この変数を true に設定することにより、Oracle JDBC ドライバに対して Select を実行する場合のラウンドトリップを回避できます。このパラメータは OC4J サーバーで設定する必要があります。

  このオプションの値を変更して OC4J を再起動すると、この変更の後にデプロイされるアプリケーションに対してのみ有効になります。変更前にデプロイされたアプリケーションには影響はありません。

  true に設定すると、`DefineColumnType` 拡張により、通常は表の記述に必要なデータベース・ラウンドトリップが節約されます。Oracle JDBC ドライバで問合せを実行した場合、結果セットの列で使用する型を判別するために、最初にデータベースへのラウンドトリップを使用します。次に、JDBC は問合せからのデータを受け取ると、データを必要に応じて変換し、結果セットに移入します。`DefineColumnType` 拡張を true に設定して問合せの列の型を指定すると、Oracle データベースへの最初のラウンドトリップが回避されます。そのように最適化されているサーバーは、必要な型変換を実行します。

スレッド・プールの設定

スレッド・プールでは、OC4J プロセスが使用するスレッドのキューが作成および維持されます。新しいスレッドをオンデマンドで作成するかわりに、既存のスレッドを再利用すると、パフォーマンスが向上し、JVM および基礎となるオペレーティング・システムにかかる負荷が減少します。

デフォルトでは、OC4J の起動時に単一のスレッド・プールが作成されます。必要に応じて新しいスレッドが作成され、このプールに追加されます。解放されたスレッドはそれぞれプールに戻され、新しいリクエストによって必要とされるまで待機します。新しいスレッドが生成される前に、プール内のアイドル・スレッドが使用されます。

プール内のスレッドは、非アクティブ状態が 10 分間続くと自動的に破棄されます。この構成で作成可能なスレッド数に制限はありません。

OC4J の大半の使用方法では、デフォルト構成で十分です。ただし、`server.xml` ファイルにある `<global-thread-pool>` 要素の `min`、`max`、`queue` および `keepAlive` 属性を使用すれば、デフォルトで作成される単一のスレッド・プールを変更できます。

また、`<global-thread-pool>` を使用して 2 つのスレッド・プールを作成することも可能です。この場合、各プールに異なるタイプのスレッドが割り振られます。

- ワーカー・スレッド・プールには、RMI、HTTP および AJP リクエストの処理に使用されるワーカー・スレッドと MDB リスナー・スレッドが含まれます。これらのスレッドはプロセス集中型で、データベース・リソースを使用します。
- 接続スレッド・プールには、リスナー・スレッド、JDBC 接続スレッド、RMI サーバーおよび HTTP サーバー接続スレッド、バックグラウンド・スレッドなどのスレッドが含まれます。通常、これらのスレッドはプロセス集中型ではありません。

2 つのプールを作成するには、ワーカー・スレッド・プールの `min`、`max`、`queue` および `keepAlive` 属性と、接続スレッド・プールの `cx-min`、`cx-max`、`cx-queue` および `cx-keepAlive` 属性を構成する必要があります。プールを作成する場合、これらすべての属性の構成が必要です。構成されていない属性があると、次のエラー・メッセージが表示されます。

```
Error initializing server: Invalid Thread Pool parameter: null
```

`<global-thread-pool>` の属性の詳細は、2-16 ページの表 2-2 を参照してください。

次の例では、OC4J プロセスの 2 つのスレッド・プールを初期化します。各プールは最小 10、最大 100 のスレッドを含みます。各キューに保持可能な未処理リクエストは 200 です。また、アイドル・スレッドは 700 秒間キープ・アライブとします。スレッド・プール情報を起動時に出力します。

```
<application-server ...>
 ...
 <global-thread-pool min="10" max="100" queue="200" keepAlive="700000"
  cx-min="10" cx-max="100" cx-queue="200" cx-keepAlive="700000" debug="true"/>
 ...
</application-server>
```

表 2-2 では、`<global-thread-pool>` 要素の属性について説明しています。この要素は、デフォルトでは `server.xml` に含まれていません。

表 2-2 <global-thread-pool> の属性

属性	説明
`min`	プールに作成するスレッドの最小数です。コンテナの起動時に、最小数のスレッドがデフォルトで事前に割り当てられ、スレッド・プールに設定されています。 `<global-thread-element>` 要素を `server.xml` に追加すると、デフォルト値が `20` に設定されます。指定可能な最小値は `10` です。

表 2-2 <global-thread-pool> の属性（続き）

属性	説明
max	プールに作成できるスレッドの最大数です。最大サイズ未満でかつアイドル・スレッドがない場合には、新しいスレッドが生成されます。新しいスレッドが生成される前にアイドル・スレッドが使用されます。 デフォルトは 40 です。
queue	キューに保持できるリクエストの最大数です。デフォルトは 80 です。
keepAlive	新しいリクエストを待つ間、スレッドをキープ・アライブ（アイドル）の状態にしておく時間（ミリ秒単位）です。タイムアウトに達するとスレッドは破棄されます。 スレッドを破棄しないようにするには、-1 に設定します。デフォルトは 600000 ミリ秒（10 分）です。これは、-1 を設定しない場合の最小値でもあります。
cx-min	接続スレッド・プールに作成するスレッドの最小数です。 指定可能な最小値は 10 です。
cx-max	接続プールに作成できるスレッドの最大数です。デフォルトは 40 です。
cx-queue	接続プールのキューに保持できるスレッドの最大数です。デフォルトは 80 です。
cx-keepAlive	新しいリクエストを待つ間、スレッドをキープ・アライブ（アイドル）の状態にしておく時間（ミリ秒単位）です。タイムアウトに達するとスレッドは破棄されます。 スレッドを破棄しないようにするには、-1 に設定します。デフォルトは 600000 ミリ秒（10 分）です。これは、-1 を設定しない場合の最小値でもあります。
debug	true の場合、起動時にアプリケーション・サーバーのスレッド・プール情報をコンソールに出力します。デフォルトは false です。

スレッド・プール構成に関するその他の注意 :

- queue 属性は、スレッドの最大数の少なくとも 2 倍のサイズに設定してください。
- ワーカー・スレッドの最小数および最大数は、自分のマシンにインストールされている CPU 数の倍数とします。ただし、この数値は小さい値に抑える必要があります。スレッドの数が増えると、オペレーティング・システムおよびガベージ・コレクタの負荷が増大します。
- cx-min および cx-max 属性は、任意の時点での物理的な接続の数を基準としています。cx-queue は接続通信量の急増に対応します。

文のキャッシング

データベース文のキャッシングにより、カーソル作成の反復、および文の解析と作成の反復によるオーバーヘッドを避けることができます。DataSource 構成で JDBC 文のキャッシングを有効にすると、反復的に使用される実行可能文がキャッシングされます。JDBC 文のキャッシュは、特定の物理接続と関連します。文のキャッシングの詳細は、『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』を参照してください。

接続オブジェクトの setStmtCacheSize() メソッドを使用するか、DataSource 構成内の stmt-cache-size XML 属性を使用して、文のキャッシングをプログラム的に動的に有効または無効にできます。キャッシュのサイズの整数値が必要です。指定したキャッシュ・サイズが、キャッシュ内の文の最大数になります。アプリケーションからデータベースに対して発行される個別の文の数を判断してください。そしてキャッシュのサイズをこの数に設定します。

この属性を指定しないか、ゼロに設定すると、キャッシュは無効になります。

例 2-2 文のキャッシング

次の XML は、文のキャッシュ・サイズを 200 に設定します。

```
<data-source>
 ...
 stmt-cache-size="200"
</data-source>
```

タスク・マネージャの粒度

タスク・マネージャは、クリーン・アップを実行するバックグラウンド・プロセスです。ただし、タスク・マネージャを使用するとコストが高くなる可能性があります。`server.xml` 内の `taskmanager-granularity` 属性を使用して、タスク・マネージャの作業のタイミングを管理できます。この要素は、クリーン・アップのためにタスク・マネージャを起動する頻度を示します。値はミリ秒単位です。デフォルトは 1000 ミリ秒です。

```
<application-server ...  taskmanager-granularity="60000" ...>
```

OC4J ロギングの有効化

OC4J はメッセージを標準エラー、標準出力の両方、および OC4J のサービスおよびデプロイ済アプリケーションの複数のログ・ファイルに記録します。

- OC4J システムおよびアプリケーションのログ・メッセージの表示 : この項では、OC4J サブシステムおよびデプロイ済アプリケーションの個別のログ・ファイルについて説明します。これらのファイルの大きさと配置場所を管理できます。
- 標準出力および標準エラーのリダイレクト : この項では、標準出力および標準エラー・メッセージをログ・ファイルに転送する方法を説明します。

注意 : OC4J は Jakarta `log4j` もサポートします。『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』の付録「オープン・ソース・フレームワークおよびユーティリティ」を参照してください。

OC4J システムおよびアプリケーションのログ・メッセージの表示

各 OC4J プロセスには、表 2-3 に示すようなログ・ファイルのセットがあります。1 つの OC4J インスタンスに対して複数のプロセスが実行されている場合は、複数セットのログ・ファイルがあります。

表 2-3 OC4J で生成されるログ・ファイルのリスト

デフォルトのログ・ファイル名	説明	スコープ	構成ファイル
`application.log`	デプロイ済アプリケーションのすべてのイベント、エラーおよび例外。	各デプロイ済アプリケーションに 1 つのログ・ファイル	`orion-application.xml`
`global-application.log`	アプリケーションに関連するすべての共通イベント、エラーおよび例外。	デフォルト・アプリケーションを含めた全アプリケーション	`application.xml`
`jms.log`	すべての JMS イベントおよびエラー。	JMS サブシステム	`jms.xml`

表 2-3 OC4J で生成されるログ・ファイルのリスト（続き）

デフォルトのログ・ファイル名	説明	スコープ	構成ファイル
rmi.log	すべての RMI イベントおよびエラー。	RMI サブシステム	rmi.xml
server.log	特定のサブシステムまたはアプリケーションに関連付けられていないすべてのイベント。これは、サーバーの起動、内部サーバー・エラーのシャットダウンの履歴を記録します。	サーバー全体	server.xml
web-access.log	Web サイトへの全アクセスを記録します。	各 Web サイト	http-web-site.xml

ログ・ファイルには 2 つのタイプがあります。

- テキスト・ログ・ファイル : このファイルに記録されたメッセージは、XML 形式ではなく、テキストベースです。このメッセージは任意のエディタで読むことができます。これがデフォルトです。通常、OC4J をスタンドアロンで使用した場合には、ログ・メッセージをテキスト形式で表示できるという利点があります。
- Oracle Diagnostic Logging（ODL）のログ・ファイル : このファイルに記録されたメッセージは、Oracle Enterprise Manager 10*g* GUI などの GUI ツールで読むことのできる XML 形式を使用しています。Oracle Application Server 内で OC4J を使用する場合は、この形式を使用して記録することをお薦めします。

テキスト・ログ・ファイル

OC4J では完全なテキスト・ロギングも使用できます。OC4J スタンドアロンの場合は、主にテキスト・ロギングを使用してください。XML 形式ではないため、エディタで簡単に読み取ることができます。

テキスト・ロギング機能は、XML ファイルに合せてメッセージを振り分けます。ただし、同一サイズの複数のログ・ファイルに書き込むのではなく、そのコンポーネントのすべてのメッセージを単一のファイルに書き込みます。テキスト・ロギングには制限値やログのロールオーバーはありません。ユーザーが OC4J を停止し、ファイルを削除し、OC4J を再起動してログ・ファイルを新しく開始しないかぎり、ログ・ファイルのサイズは増大し続けます。ログ・ファイルを監視しないと、ディスク領域のオーバーランが発生する可能性があります。これはスタンドアロンの開発環境でのみ使用可能です。

テキスト・メッセージングはデフォルトであり、表 2-3 に示した XML ファイルで構成されます。テキスト・メッセージングは、http-web-site.xml ファイルを除き、XML ファイルの <log> 要素の <file> サブ要素で有効にします。http-web-site.xml ファイルでは、<access-log> 要素を使用してテキスト・メッセージングを有効にします。テキスト・メッセージングをオフにするには、<file> または <access-log> 要素を除去するか、コメント化します。この行を削除しないで ODL ロギングを有効にすると、両方のロギング機能がオンになります。表 2-4 に示すように、テキスト・メッセージングの場所とファイル名にはデフォルトはありませんが、<log> または <access-log> 要素の path 属性で場所とファイル名を指定できます。

表 2-4 に、スタンドアロン OC4J のログ・ファイルのデフォルト位置を示します。これらのファイルの位置と名前を変更するには、表 2-3 で説明した構成ファイルを変更します。

表 2-4 OC4J スタンドアロンのログ・ファイルの位置

ログ・ファイル	デフォルト位置
application.log	*install_dir*/j2ee/home/application-deployments/<application-name>
global-application.log	*install_dir*/j2ee/home/log
jms.log	*install_dir*/j2ee/home/log

表 2-4 OC4J スタンドアロンのログ・ファイルの位置（続き）

ログ・ファイル	デフォルト位置
rmi.log	*install_dir*/j2ee/home/log
server.log	*install_dir*/j2ee/home/log
web-access.log	この位置は、<access-log path="../log/http-web-access.log" /> のように、<access-log> 要素を使用して *-web-site.xml で構成できます。

前述のすべてのログ・ファイルの位置は、web-access.log ファイルを除いて、すべて各構成ファイルの <log> 要素を使用して指定できます。絶対パス、または j2ee/home/config ディレクトリの相対パスを指定できます。たとえば、server.xml 構成ファイルでサーバー・ログ・ファイルを次のように指定します。

```
<log>
<file path="../log/my-server.log" />
</log>
```

ログ・ファイルの位置は、次のように絶対パスで指定することもできます。

```
<log>
<file path="d:¥log-files¥my-server.log" />
</log>
```

Oracle Diagnostic Logging（ODL）のログ・ファイル

各 ODL ログ・エントリは、それぞれのログ・ファイルに XML 形式で書き込まれます。XML メッセージは、XML リーダーを使用して読むことができます。ODL ロギングの利点は、ログ・ファイルとディレクトリに最大値の制限があることです。制限に達すると、ログ・ファイルは上書きされます。

ODL ロギングを有効にすると、新規のメッセージは log.xml という名前の現行ログ・ファイルに記録されます。ログ・ファイルが満杯になる、つまりログ・ファイルの最大サイズに達すると、log*N*.xml という名前のアーカイブ・ログ・ファイルにコピーされます。この N は 1 から開始される数字です。最後のログ・ファイルが満杯になると、次のようになります。

1. ディレクトリ内に領域を確保するため、最も古いログ・ファイルが消去されます。
2. log.xml ファイルは最新の log*N*.xml ファイルに書き込まれます。この N は、最新のログ・ファイルに 1 を加えた数字になります。

このように、ログ・ファイルは常にロールオーバーするため、ディスク領域を侵害することはありません。

表 2-3 に示した各 XML ファイルで、次のように ODL 構成行を非コメント化して ODL ロギングを有効にします。

- http-web-site.xml ファイルを除き、表 2-3 に示したすべての XML ファイルの <log> 要素内の <odl> 要素を非コメント化します。
- <odl-access-log> 要素を http-web-site.xml ファイルに追加します。

次の属性を構成できます。

- path: この領域のログ・フォルダのパスとフォルダ名。絶対パスを使用するか、構成 XML ファイルがある場所（通常は、j2ee/home/config ディレクトリ）に対する相対パスを使用できます。これは、XML 構成ファイルが関係する機能に対して、そのログ・ファイルが置かれる場所を示します。たとえば、server.xml ファイルのこの要素を変更して、サーバー・ログ・ファイルが書き込まれる場所を示します。
- max-file-size: 各ログ・ファイルの最大サイズ（KB 単位）。
- max-directory-size: ディレクトリの最大サイズ（KB 単位）。

ディレクトリの最大サイズに達するまで、ディレクトリ内に新規ファイルが作成されます。各ログ・ファイルは、属性で指定された最大値以下になります。

server.xml ファイルで、<install-dir>/j2ee/home/log/server ディレクトリ内のログ・ファイルを 1000KB、ディレクトリの最大値を 10,000KB に指定するには、次のように構成します。

```
<log>
<odl path="../log/server/" max-file-size="1000" max-directory-size="10000" />
</log>
```

OC4J の実行中、サーバー宛てのすべてのログ・メッセージは <install-dir>/j2ee/home/log/server ディレクトリに記録されます。

記録される XML メッセージの形式は次のようになります。

```
<MESSAGE>
<HEADER>
<TSTZ_ORIGINATING>2002-11-12T15:02:07.051-08:00</TSTZ_ORIGINATING>
<COMPONENT_ID>oc4j</COMPONENT_ID>
<MSG_TYPE TYPE="ERROR"></MSG_TYPE>
<MSG_LEVEL>1</MSG_LEVEL>
<HOST_ID>myhost</HOST_ID>
<HOST_NWADDR>001.11.22.33</HOST_NWADDR>
<PROCESS_ID>null-Thread[Orion Launcher,5,main]</PROCESS_ID>
<USER_ID>dpda</USER_ID>
</HEADER>
<PAYLOAD>
<MSG_TEXT>java.lang.NullPointerException at
com.evermind.server.ApplicationServer.setConfig(ApplicationServer.java:1070)
at com.evermind.server.ApplicationServerLauncher.run
(ApplicationServerLauncher.java:93) at java.lang.Thread.run(Unknown Source)
</MSG_TEXT>
</PAYLOAD>
</MESSAGE/>
```

ODL ロギングとテキスト・ロギングの両方をオンにできます。ディスク領域を節約するために、これらのオプションのうち 1 つはオフにしてください。ODL ロギングを有効にする場合は、http-web-site.xml ファイルを除き、すべての XML ファイルの <log> 要素の <file> サブ要素をコメント化して、テキスト・ロギング機能をオフにします。http-web-site.xml ファイルでは、<access-log> 要素をコメント化してテキスト・ロギングをオフにします。

標準出力および標準エラーのリダイレクト

多くの開発者は、System.out.println() および System.err.println() メソッドをアプリケーションで使用して、デバッグ情報を生成します。通常、これらのメソッド・コールの出力は OC4J プロセスが開始されたコンソールに表示されます。ただし、OC4J の起動時にコマンドライン・オプションを指定すれば、STDOUT および STDERR の出力を直接ファイルに送ることができます。-out および -err パラメータは、エラー・メッセージの送信先を OC4J に指示します。次の起動コマンドは、-out および -err パラメータの例を含んでいます。

```
$ java -jar oc4j.jar -out d:¥log-files¥oc4j.out -err d:¥log-files¥oc4j.err
```

この場合、STDOUT および STDERR に書き込まれるすべての情報が、ファイル d:¥log-files¥oc4j.out および d:¥log-files¥oc4j.err にそれぞれ出力されます。

Web アプリケーションに対するアクセス・ロギングの無効化

OC4J 10.1.2 の実装では、（Web サイトへのリクエストをログに記録するために使用する）OC4J のアクセス・ロギングをモジュールベースで無効にする新機能が、Web サイトの XML ファイルにあります。

一般的には、（default-web-site.xml や http-web-site.xml など）Web サイトの XML ファイルの <web-site> 要素の <access-log> サブ要素を使用して、Web サイトに対するテキストベースのアクセス・ロギングが有効となります。または、<web-site> 要素の <odl-access-log> サブ要素を使用して、Web サイトに対する Oracle Diagnostic Logging（ODL ベースのアクセス・ロギング）が有効となります。

リリース 2（10.1.2）では、特定の Web アプリケーション（モジュール）に対し、その Web アプリケーションの <web-app> 要素の access-log 属性を使用して、テキストベースのロギングまたは ODL ベースのロギング（該当する場合）を無効にできます。<web-app> 要素は、<web-site> の別のサブ要素です。Web アプリケーションに対し access-log="false" と設定すると、すべての <access-log> 要素または <odl-access-log> 要素がオーバーライドされ、その Web アプリケーションが動作している間はロギングが無効となります。

次の例では、default アプリケーションの dms0 モジュールのロギングは無効になりますが、admin_web モジュールのテキストベースのロギングは有効のままです。

```
<web-site ... >
...
   <web-app application="default" name="dms0" root="/dms0" access-log="false" />
   <web-app application="default" name="admin_web" root="/adminoc4j" />
   <access-log path="../log/http-web-access.log" />
...
</web-site>
```

注意： デフォルト設定は access-log="true" です。この設定では、機能は前のリリースと変わりませんが、ロギングの有効 / 無効は、<access-log> 要素または <odl-access-log> 要素の有無のみによって決定されます。

Web サイトの XML ファイルに <access-log> 要素も <odl-access-log> 要素もない場合は、ロギングはそもそも無効で、アプリケーションに access-log="false" と設定しても、なんの影響も発生しません。

アクセス・ロギングの関連情報は、『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。

OC4J のデバッグ

OC4J は、OC4J の各種のサブシステムで実行される操作に関する追加情報を生成するために、いくつかのデバッグ・プロパティを提供しています。OC4J の起動時に特定のサブシステムに対してこれらのデバッグ・プロパティを設定できます。

注意： デバッグ・オプションを過剰に設定すると、アプリケーションの実行速度が低下し、ログ・ファイルの内容を収容するために多量のディスク領域が使用されます。

次の表は、OC4J で使用可能な便利なデバッグ・オプションを示しています。これらのデバッグ・オプションには、true または false という 2 つの状態があります。デフォルトでは false に設定されます。デバッグ・プロパティの全リストは、B-32 ページの「OC4J システム・プロパティ」を参照してください。

表 2-5 HTTP デバッグ・オプション

HTTP デバッグ	オプションの説明
http.session.debug	HTTP セッション・イベントに関する情報を提供します。
http.request.debug	各 HTTP リクエストに関する情報を提供します。
http.error.debug	すべての HTTP エラーを出力します。

表 2-5 HTTP デバッグ・オプション（続き）

HTTP デバッグ	オプションの説明
http.method.trace.allow	デフォルトは false です。true の場合は、trace HTTP メソッドをオンにします。

表 2-6 JDBC デバッグ・オプション

JDBC デバッグ	オプションの説明
datasource.verbose	プールにリリースされた DataSource および接続を使用して、データソースおよび接続の作成時に詳細情報などを提供します。
jdbc.debug	JDBC コールの際に非常に詳細な情報を提供します。

表 2-7 RMI デバッグ・オプション

RMI デバッグ	オプションの説明
rmi.debug	RMI デバッグ情報を出力します。
rmi.verbose	RMI コールに関する非常に詳細な情報を提供します。

表 2-8 OracleAS Web Services デバッグ・オプション

OracleAS Web Services デバッグ	オプションの説明
ws.debug	Web サービスのデバッグをオンにします。

たとえば、HTTP セッション・イベントに関するデバッグ情報を生成する場合は、OC4J を次のように起動します。

```
java -Dhttp.session.debug=true -jar oc4j.jar
```

特定のデバッグ・オプションを使用して OC4J を起動した場合、デバッグ情報が生成されて標準出力に送られます。上の例では、HTTP セッション情報が OC4J コンソールに次のように表示されます。

```
Oracle Application Server Containers for J2EE initialized
Created session with id '36c04d8a1cd64ef2b6a9ba6e2ac6637e' at Mon Apr 15 12:24:20
PDT 2002, secure-only: false
Created session with id '36c04d8a1cd64ef2b6a9ba6e2ac6637e' at Mon APR 15 12:36:06
PDT 2002, secure-only: false
Invalidating session with id '36c04d8a1cd64ef2b6a9ba6e2ac6637e' at Mon APR 15 12:44:32
PDT 2002 (created at Mon APR 15 12:24:23 PDT 2002) due to timeout
```

このデバッグ情報を保存するには、-out または -err コマンドライン・オプションを使用して、次のように標準出力をファイルにリダイレクトします。

```
java -Dhttp.session.debug=true -jar oc4j.jar -out oc4j.out -err oc4j.err
```

特定のサブシステム・スイッチに加えて、指定の詳細レベルで OC4J を起動できます。詳細レベルは 1 ～ 10 の整数で、詳細レベルが高くなるほど、コンソールに表示される情報量は多くなります。詳細レベルは、OC4J コマンドライン・オプションで -verbosity OC4J オプションを使用して指定します。次の例は、詳細情報を含む場合と含まない場合の出力を示しています。

例 2-3 詳細を含まないエラー・メッセージの表示

```
D:¥oc4j903¥j2ee¥home>java -jar oc4j.jar
Oracle Application Server Containers for J2EE initialized
```

例 2-4 詳細レベル 10 のエラー・メッセージの表示

```
D:¥oc4j903¥j2ee¥home>java -jar oc4j.jar -verbosity 10
Application default (default) initialized...
Binding EJB work.ejb.WorkHours to work.ejb.WorkHours...
Application work (work) initialized...
Application serv23 (Servlet 2.3 New Features Demo) initialized...
Web-App default:defaultWebApp (0.0.0.0/0.0.0.0:8888) started...
Oracle Application Server Containers for J2EE initialized
```

サーブレットのデバッグ例

サーブレットに問題のある Web アプリケーションを OC4J にデプロイしました。事前構成したデータソースを使用してデータベース接続を行うと、クライアント・セッションが切断されます。サーブレットがデータソースにアクセスするときに OC4J で何が行われているかを知る必要があります。HTTP セッションおよびデータソースの使用に関するデバッグ情報を生成するために、`http.session.debug` および `datasource.verbose` という 2 つのデバッグ・オプションを true に設定する必要があります。

```
java -Dhttp.session.debug=true -Ddatasource.verbose=true -jar oc4j.jar
```

その後、サーブレットを再実行すると、OC4J プロセスの標準出力に次のようなデバッグ情報が表示されます。

```
DataSource logwriter activated... jdbc:oracle:thin:@localhost:1521/MYSERVICE:
Started
jdbc:oracle:thin:@localhost:1521/MYSERVICE: Started
Oracle Application Server Containers for J2EE initialized
Created session with id '4fa5eb1b9a564869a426e8544963754f' at Tue APR 23
16:22:56 PDT 2002, secure-only: false
Created new physical connection: XA XA OC4J Pooled
jdbc:oracle:thin:@localhost:1521/MYSERVICE
null: Connection XA XA OC4J Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE
allocated (Pool size: 0)
jdbc:oracle:thin:@localhost:1521/MYSERVICE: Opened connection
Created new physical connection: Pooled
oracle.jdbc.driver.OracleConnection@5f18
Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE: Connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 allocated (Pool size: 0)
Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE: Releasing connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 to pool (Pool size: 1)
null: Releasing connection XA XA OC4J Pooled
jdbc:oracle:thin:@localhost:1521/MYSERVICE to pool (Pool size: 1)
OC4J Pooled jdbc:oracle:thin:@localhost:1521/MYSERVICE: Cache timeout, closing
connection (Pool size: 0)
com.evermind.sql.OrionCMTDataSource/default/jdbc/OracleDS: Cache timeout,
closing connection (Pool size: 0)
```

3

セキュリティの構成

OC4J セキュリティでは、ユーザー・マネージャを使用して、J2EE アプリケーションにアクセスしようとするユーザーおよびグループの認証と認可を行います。ユーザー・マネージャはそれぞれパフォーマンスが異なり、必要なセキュリティに応じて使用されます。暗号化による機密保護は SSL を利用して提供されます。

この章では、次の項目について説明します。

- セキュリティ機能の概要
- 認証
- 認可
- ユーザー・マネージャのプラグ・イン
- SSL による機密保護

Oracle Application Server セキュリティの一般的な説明については、『Oracle Application Server セキュリティ・ガイド』および『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

セキュリティ機能の概要

OC4J セキュリティは 2 段階のプロセスに基づいています。J2EE アプリケーションにアクセスしようとするユーザーまたはグループは、まず認証を受けた後、認可されます。認証と認可は、JAZNUserManager および XMLUserManager クラスなどの各種ユーザー・マネージャによって行われます。JAZNUserManager クラスはデフォルトであり、最高のセキュリティを提供します。XMLUserManager は最も単純なセキュリティ方式です。JAZNUserManager は、Lightweight Directory Access Protocol（LDAP）ベースまたは XML ベースのプロバイダ・タイプを使用することで、OracleAS JAAS Provider を OC4J のセキュリティ・インフラストラクチャとして活用します。XMLUserManager はファイルを使用して構成するため、パスワードが目に見える状態になっています。

OracleAS JAAS Provider、プロバイダ・タイプおよびユーザー・マネージャの詳細は、3-8 ページの「ユーザー・マネージャのプラグ・イン」を参照してください。また、OracleAS JAAS Provider およびプロバイダ・タイプの詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』も参照してください。

注意： デフォルトのユーザー・マネージャは、XMLUserManager から JAZNUserManager に変わりました。

認証、認可および OC4J での機密保護の概要を次に示します。

- 認証 : ユーザーの識別情報および資格証明を検証します。

  ユーザーおよびグループをユーザー・リポジトリに定義します。ユーザー・リポジトリは、ユーザー・マネージャが J2EE アプリケーションにアクセスしようとするユーザーまたはグループの識別情報を検証するために使用します。ユーザー・リポジトリは、ご使用の環境によって、ファイルまたはディレクトリ・サーバーのいずれかになります。Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider の LDAP ユーザー・マネージャと XMLUserManager はユーザー・リポジトリの例です。

  J2EE アプリケーションではアプリケーションにアクセスできるクライアントが判別されますが、ユーザー・リポジトリの情報を基にクライアントの識別情報を検証するのは、ユーザー・マネージャ（ユーザー名とパスワードを使用）です。

- 認可 : ユーザーおよびグループによるアプリケーションへのアクセスを許可または拒否します。

  ユーザーおよびグループの認可（識別情報）は、J2EE のデプロイメント・ディスクリプタと OC4J 固有のデプロイメント・ディスクリプタの両方に指定します。J2EE のデプロイメント・ディスクリプタと OC4J 固有のデプロイメント・ディスクリプタは、アプリケーションの様々な部分へのアクセスに必要なロールを示します。ロールとは、各アプリケーションが様々なオブジェクトへのアクセス権を示すのに使用する論理識別情報です。OC4J 固有のデプロイメント・ディスクリプタは、論理ロールと OC4J で認識されるユーザーおよびグループとの間のマッピングを提供します。

- SSL による機密保護 : 暗号化通信を保証します。

  暗号化通信には、HTTP で Secure Sockets Layer（SSL）を使用します。

認証

認証とは、ユーザーの識別情報および資格証明の有効性を検証することです。J2EE アプリケーションでは、アプリケーションを使用できるユーザーが判別されます。ただし、ユーザー・リポジトリの情報を基にユーザーの識別情報を検証するのは、ユーザー・マネージャ（ユーザー名とパスワードを使用）です。認証は認可とは異なります。認可は、ユーザーの識別情報を基に J2EE アプリケーションへのユーザー・アクセスを許可するプロセスです。

OC4J セキュリティでは、HTTP および Enterprise JavaBeans（EJB）という 2 種類のクライアントを認証します。この項では、それぞれのクライアント、およびユーザーとグループの設定について説明します。

ユーザーおよびグループの指定

OC4J は、ユーザーおよびグループの定義をサポートしています。この定義は、デプロイされたすべてのアプリケーションで共有されるか、特定のアプリケーション固有になります。

- 共有されるユーザーおよびグループは、ユーザー・リポジトリにリストされます。ユーザー・リポジトリの場所は、グローバル config/application.xml ファイルで指定します。
- アプリケーション固有のユーザーおよびグループは、アプリケーション固有のユーザー・リポジトリにリストされます。ユーザー・リポジトリの場所は、そのアプリケーションの orion-application.xml ファイルで指定します。

ユーザーおよびグループの定義方法は、使用するユーザー・マネージャによって異なります。たとえば、Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider（OracleAS JAAS Provider）はグループではなくロールを使用するため、JAZNUserManager の XML ベースのユーザー・リポジトリ jazn-data.xml は、XMLUserManager のユーザー・リポジトリ principals.xml とは異なる構造を持ちます。また、JAZNUserManager のユーザー・リポジトリでは、principals.xml とは異なり、パスワードが暗号化されます。

次の各項では、JAZNUserManager および XMLUserManager クラスによってユーザーおよびグループを指定する方法の例を示します。これらのクラスの詳細は、3-8 ページの「ユーザー・マネージャのプラグ・イン」を参照してください。

例 : jazn-data.xml でのユーザーおよびグループの指定

JAZNUserManager のユーザー・リポジトリ構成ファイル jazn-data.xml から取った次の XML は、OracleAS JAAS Provider のロール（グループ）およびユーザーの定義方法を示しています。この XML では、allusers という名前のグループと guest という名前のユーザーが定義されます。

```
<role>
  <name>allusers</name>
  <members>
   <member>
    <type>user</type>
    <name>guest</name>
   </member>
  </members>
</role>
```

XMLUserManager のユーザー・リポジトリ構成ファイル principals.xml とは異なり、JAZNUserManager ではパスワードを暗号化できます。

```
<user>
  <name>guest</name>
  <description>The default user</description>
  <credentials>wEE6aA==</credentials>
</user>
```

> **注意 :** jazn-data.xml ファイルの設定の詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

例 : principals.xml でのユーザーおよびグループの指定

principals.xml ファイル（XMLUserManager クラスのユーザー・リポジトリ構成ファイル）から取った次の XML は、allusers という名前のグループとパスワード welcome を持つ guest という名前のユーザーを定義する方法を示しています。guest ユーザーは、allusers グループのメンバーになります。

JAZNUserManager クラスのかわりに XMLUserManager クラスを使用する場合、すべてのアプリケーションが対象になるときはグローバル application.xml ファイルを、特定のアプリケーションのみが対象になるときは orion-application.xml ファイルを変更する必要があります。次の行を追加します。

```
<principals path="./principals.xml" />
```

パスは、principals.xml ファイルの場所を指しています。また、このファイル内の <jazn provider> 要素を削除するか、コメント化する必要があります。

> **注意：** パスワードの間接化によって、パスワードを隠すことができます。パスワードの間接化の詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

```
<principals>
 <groups>
  <group name="allusers">
   <description>Group for all normal users</description>
   <permission name="rmi:login" />
   <permission name="com.evermind.server.rmi.RMIPermission" />
  </group>
....other groups...
 </groups>
 <users>
  <user username="guest" password="welcome">
   <description>Guest user</description>
   <group-membership group="allusers" />
  </user>
 </users>
</principals>
```

HTTP クライアントの認証

OC4J では、保護された URL にアクセスするクライアントが自己認証を行う必要があります。ユーザー名およびパスワードによる認証、または SSL の場合には SSL 証明書による認証が可能です。ただし、認証が必要になるほとんどの場合は、ユーザーがユーザー名およびパスワードの入力を求められます。SSL 証明書を使用してクライアントを認証する場合は、クライアント証明書とサーバー・キーストアの設定方法について、3-13 ページの「SSL による機密保護」を参照してください。

EJB クライアントの認証

OC4J で EJB にアクセスする場合、このサーバーに有効な資格証明を渡す必要があります。

- スタンドアロン・クライアントの場合は、jndi.properties ファイルに資格証明を定義します。このファイルは、EAR ファイルとともにデプロイされるか、InitialContext オブジェクト内に存在します。
- OC4J で実行されるサーブレットまたは JavaBeans は、InitialContext オブジェクト内の資格証明を渡します。このオブジェクトは、リモート EJB の参照用に作成されます。

JNDI プロパティの設定

クライアントがターゲットと同じアプリケーション内に存在する場合、またはターゲットがその親の中に存在する場合には、JNDI プロパティ・ファイルは不要です。それ以外の場合は、JNDI コールの前に、システム・プロパティの jndi.properties ファイル内、または実装内にある JNDI プロパティを初期化する必要があります。次の各項で、これら 3 つの選択肢について説明します。

- JNDI プロパティ不要
- JNDI プロパティ・ファイル

JNDI プロパティ不要 ターゲット Bean と同じアプリケーション内に存在するサーブレットは、ノードの JNDI プロパティに自動的にアクセスします。したがって、JNDI プロパティが不要になるため、EJB へのアクセスは単純です。

```
//Get the Initial Context for the JNDI lookup for a local EJB
InitialContext ic = new InitialContext();
//Retrieve the Home interface using JNDI lookup
Object empObject = ic.lookup("java:comp/env/employeeBean");
```

このことは、ターゲット Bean の存在するアプリケーションがその親としてデプロイされている場合にも当てはまります。親を指定するには、元のアプリケーションをデプロイする際に、`admin.jar` コマンドの `-parent` オプションを使用します。

JNDI プロパティ・ファイル `jndi.properties` ファイルに JNDI プロパティを設定する場合、各プロパティを次のように設定します。このファイルは必ず `CLASSPATH` からアクセスできるようにしておいてください。

ファクトリ

```
java.naming.factory.initial=
com.evermind.server.ApplicationClientInitialContextFactory
```

場所

ORMI のデフォルト・ポート番号は 23791 です。これは、`j2ee/home/config/rmi.xml` で変更できます。したがって、次のいずれかの方法で `jndi.properties` に URL を設定します。

```
java.naming.provider.url=ormi://hostname/application-name
```

または

```
java.naming.provider.url=ormi://hostname:23791/application-name
```

セキュリティ

OC4J で EJB にアクセスする場合、このサーバーに有効な資格証明を渡す必要があります。スタンドアロン・クライアントの場合は、クライアント・コードとともにデプロイされる `jndi.properties` ファイルに資格証明を定義します。

```
java.naming.security.principal=username
java.naming.security.credentials=password
```

実装内の JNDI プロパティ プロパティを同じ値に設定しますが、使用する構文は異なります。たとえば、コンテナで実行される JavaBeans は、`InitialContext` 内の資格証明を渡します。このオブジェクトは、リモート EJB の参照用に作成されます。

`Hashtable` 環境で JNDI プロパティを渡すには、各プロパティを次のように設定します。

```
Hashtable env = new Hashtable();
env.put("java.naming.provider.url", "ormi://myhost/ejbsamples");
env.put("java.naming.factory.initial",
      "com.evermind.server.ApplicationClientInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "guest");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
Context ic = new InitialContext (env);
Object homeObject = ic.lookup("java:comp/env/employeeBean");

// Narrow the reference to a TemplateHome.
EmployeeHome empHome =
 (EmployeeHome) PortableRemoteObject.narrow(homeObject,
                                            EmployeeHome.class);
```

初期コンテキスト・ファクトリ・クラスの使用

ほとんどのクライアントに対しては、初期コンテキスト・ファクトリ・クラスを `ApplicationClientInitialContextFactory` に設定します。XML 構成ファイルの `<ejb-ref>` で定義された論理名を使用しない場合は、ターゲット Bean の実際の JNDI 名を指定する必要があります。この場合、別の初期コンテキスト・ファクトリ・クラスである `com.evermind.server.RMIInitialContextFactory` クラスを使用できます。

例 3-1 リモート OC4J インスタンスの EJB にアクセスするサーブレット

次のサーブレットは、ターゲット Bean の JNDI 名である `/cmpapp/employeeBean` を使用します。そのため、このサーブレットには `ApplicationClientInitialContext` オブジェクトのかわりに `RMIInitialContext` オブジェクトの JNDI プロパティを指定する必要があります。環境は次のように初期化されます。

- `INITIAL_CONTEXT_FACTORY` は `RMIInitialContextFactory` に初期化されます。
- `InitialContext` は、新たに作成されるのではなく、取得されます。
- 実際の JNDI 名がルックアップで使用されます。

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "ormi://myhost/cmpapp");
env.put(Context.SECURITY_PRINCIPAL, "admin");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.evermind.server.rmi.RMIInitialContextFactory");

Context ic =
new com.evermind.server.rmi.RMIInitialContextFactory().
getInitialContext(env);

Object homeObject = ic.lookup("/cmpapp/employeeBean");

// Narrow the reference to a TemplateHome.
EmployeeHome empHome =
(EmployeeHome) PortableRemoteObject.narrow(homeObject,
EmployeeHome.class);
```

認可

認可とは、ユーザーの識別情報を基に J2EE アプリケーションへのユーザー・アクセスを許可または拒否するプロセスです。認可は認証とは異なります。認証は、ユーザーが有効であることを検証するプロセスです。

ユーザーおよびグループの認可は、J2EE のデプロイメント・ディスクリプタと OC4J 固有のデプロイメント・ディスクリプタの両方に指定します。J2EE のデプロイメント・ディスクリプタでは、論理ロールを使用する際のアクセス・ルールを指定します。OC4J 固有のデプロイメント・ディスクリプタでは、論理ロールを、ユーザー・リポジトリで定義された実際のユーザーおよびグループにマップします。

次の項では、ユーザー、グループおよびロールの定義方法について説明します。

- J2EE アプリケーションの論理ロールの指定
- ユーザーおよびグループへの論理ロールのマッピング

J2EE アプリケーションの論理ロールの指定

アプリケーションが使用する論理ロールを XML デプロイメント・ディスクリプタに指定します。アプリケーションのコンポーネント・タイプに応じて、次のいずれかを論理ロールで更新します。

- Web コンポーネントの場合は `web.xml`
- EJB コンポーネントの場合は `ejb-jar.xml`

- アプリケーションの場合は application.xml

それぞれのデプロイメント・ディスクリプタで、<security-role> という XML 要素を使用してロールを定義します。

例 3-2 EJB JAR のセキュリティ・ロールの定義

次の手順は、ejb-jar.xml デプロイメント・ディスクリプタに VISITOR という名前の論理ロールを作成するために必要な XML を示しています。

1. 論理セキュリティ・ロール VISITOR を <security-role> 要素に定義します。

```
<security-role>
 <description>A role for every user</description>
 <role-name>VISITOR</role-name>
</security-role>
```

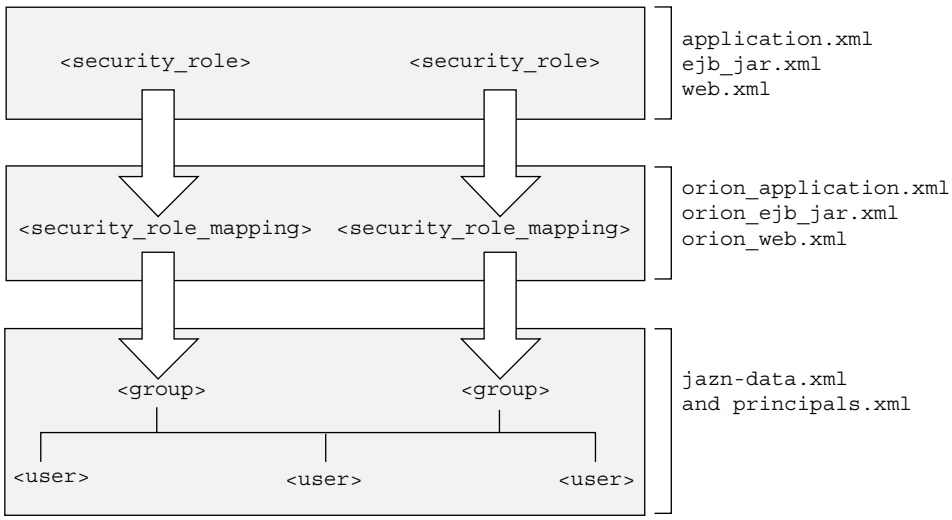
2. このロールがアクセスできる Bean およびメソッドを <method-permission> 要素に定義します。

```
<method-permission>
 <description>VISITOR role needed for CustomerBean methods</description>
 <role-name>VISITOR</role-name>
 <method>
  <ejb-name>customerbean</ejb-name>
  <method-name>*</method-name>
 </method>
</method-permission>
```

ユーザーおよびグループへの論理ロールのマッピング

アプリケーション・デプロイメント・ディスクリプタで定義された論理ロールを、ユーザー・リポジトリで定義された実際のユーザーおよびグループにマップします。マッピングは、<security-role-mapping> 要素を使用して、OC4J 固有のデプロイメント・ディスクリプタに指定します。このマッピングを図 3-1 に示します。

図 3-1 jazn-data.xml で定義されたユーザーおよびグループへの論理ロールのマッピング



注意： principals.xml または jazn-data.xml ファイルで定義されたセキュリティ・ロール・マッピング・レイヤーは、次の条件が true の場合にバイパスされます。

- セキュリティ・ロールおよびグループ（jazn-data.xml の場合はロール）の名前が同じであること。
- セキュリティ・ロール・マッピングが指定されていないこと。

例 3-3 実際のロールへの論理ロールのマッピング

この例では、論理ロール VISITOR を orion-ejb-jar.xml ファイルの allusers グループにマップします。このグループの一部としてログイン可能なユーザーはすべて VISITOR ロールを持っているとみなされるため、customerbean のメソッドを実行できます。このロールは、ユーザー・マネージャ構成ファイル（jazn-data.xml ファイル）で定義された allusers グループにマップされます。

```
<security-role-mapping name="VISITOR">
 <group name="allusers" />
</security-role-mapping>
```

注意： 論理ロールは、単一のグループにも複数のグループにもマップできます。

ユーザー・マネージャのプラグ・イン

OC4J セキュリティを提供するユーザー・マネージャ・クラスはすべて、com.evermind.security.UserManager インタフェースの実装です。これにはカスタム・ユーザー・マネージャも含まれます。ユーザー・マネージャ・クラスは、createUser()、getUser()、getGroup() などのメソッドを使用してユーザー、グループおよびパスワードを管理します。表 3-1 に、OC4J セキュリティで使用できるユーザー・マネージャを示します。

表 3-1 OC4J で使用できるユーザー・マネージャとそのユーザー・リポジトリ

ユーザー・マネージャ	ユーザー・リポジトリ
oracle.security.jazn.oc4j.**JAZNUserManager**	■ XML ベースのプロバイダ・タイプを使用 : jazn-data.xml ■ LDAP ベースのプロバイダ・タイプを使用 : OID
com.evermind.server.**XMLUserManager**	principals.xml
カスタム・ユーザー・マネージャ	ユーザー指定のユーザー・リポジトリ

デフォルトでは、OC4J は JAZNUserManager のユーザー・リポジトリ jazn-data.xml からユーザー名、グループおよびパスワードを読み取ります。OC4J で任意のユーザー・マネージャを使用するには、ユーザー・マネージャ・クラスの名前を次のいずれかの XML ファイルに指定する必要があります。

- orion-application.xml: 単一アプリケーション用のファイル
- config/application.xml: サーバー内のすべてのアプリケーションを対象としたグローバル構成ファイル

次の項では、各ユーザー・マネージャ・タイプの構成方法について説明します。

- JAZNUserManager クラスの使用
- XMLUserManager クラスの使用
- 独自のユーザー・マネージャの作成

JAZNUserManager クラスの使用

JAZNUserManager クラスの主な目的は、OracleAS JAAS Provider を OC4J のセキュリティ・インフラストラクチャとして活用することです。OracleAS JAAS Provider の詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

OracleAS JAAS Provider を OC4J と統合することで、次の利点が得られます。

- シングル・サインオン（SSO）/mod_osso の統合
- SSL/mod_ossl の統合
- OID の統合（LDAP ベースのプロバイダ・タイプを使用）
- Java2 権限を使用したきめ細かなアクセス制御
- run-as 識別情報のサポート、委任のサポート（サーブレットから EJB へ）
- セキュアなファイルベースのパスワード格納（XML ベースのプロバイダ・タイプを使用）

OC4J セキュリティで OracleAS JAAS Provider データのセキュアな一元的格納、取得および管理を可能にするには、JAZNUserManager クラスを使用します。このデータは、レルム（ユーザーおよびロール）と OracleAS JAAS Provider ポリシー（許可）情報で構成されます。図 3-2 に、JAZNUserManager クラスによる OC4J セキュリティ・アーキテクチャを示します。

JAZNUserManager クラスでは、2 種類の OracleAS JAAS Provider を使用して OC4J セキュリティを実現できます。ご使用の環境に適したプロバイダ・タイプを使用してください。

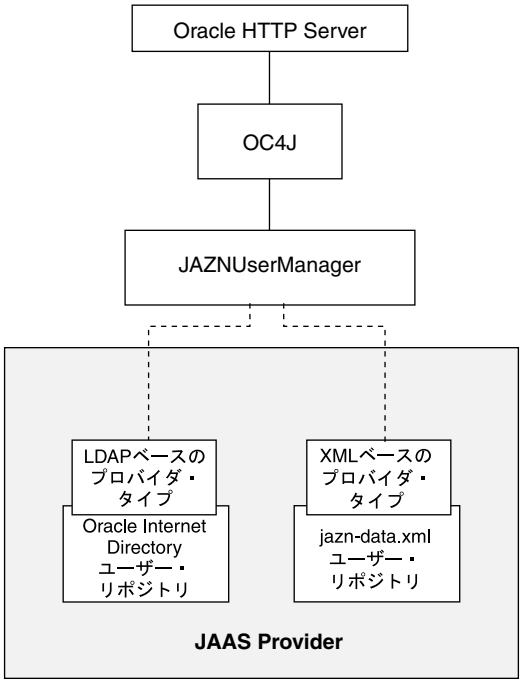
- LDAP ベース

  ディレクトリへの一元的な情報格納が可能です。ユーザー・リポジトリは OID です。

- XML ベース

  XML ファイルへの軽量な情報格納が可能です。ユーザー・リポジトリは jazn-data.xml ファイルです。

図 3-2 JAZNUserManager クラスによる OC4J セキュリティ・アーキテクチャ



OC4J では、OC4J 固有の構成ファイル（config/application.xml または orion-application.xml）に <jazn> または <user-manager> 要素を追加することで、JAZNUserManager クラスを使用するようアプリケーションを構成できます。

LDAP ベースのプロバイダ・タイプでの JAZNUserManager クラスの使用

LDAP ベースのプロバイダ・タイプは、ユーザーおよびグループの管理機能を OID から Delegated Administrative Service（DAS）へ委任します。

OC4J 固有の構成ファイルから取った次の例では、JAZNUserManager クラスを LDAP ベースのプロバイダ・タイプのユーザー・マネージャとして使用しています。

```
<jazn provider="LDAP" default-realm="sample_subrealm"
location="ldap://myoid:389" />
```

または

```
<user-manager class="oracle.security.jazn.oc4j.JAZNUserManager">
<property name="provider.type" value="LDAP" />
<property name="realm.default" value="sample_subrealm" />
<property name="ldap.service"  value="ldap://myoid:389" />
</user-manager>
```

注意： <user-manager> 要素と <jazn> 要素の両方を指定すると、<jazn> 要素が無視されます。

XML ベースのプロバイダ・タイプでの JAZNUserManager クラスの使用

XML ベースのプロバイダ・タイプは OracleAS JAAS Provider API の高速かつ軽量の実装です。このプロバイダ・タイプは、XML を使用してユーザー名と暗号化されたパスワードを格納します。

OC4J 固有の構成ファイルから取った次の例では、JAZNUserManager クラスを XML ベースのプロバイダ・タイプのユーザー・マネージャとして使用しています。ユーザー・リポジトリは .../j2ee/home/jazn/config/**jazn-data.xml** にあります。データファイルにはレルムが 1 つしかないため、realm.default の指定は不要です。

```
<jazn provider="XML"
location=".../j2ee/home/config/jazn-data.xml" />
```

または

```
<user-manager class="oracle.security.jazn.oc4j.JAZNUserManager">
<property name="provider.type" value="XML" />
<property name="xml.store.fs.jazn"
value=".../j2ee/home/config/jazn-data.xml" />
</user-manager>
```

注意： <user-manager> 要素と <jazn> 要素の両方を指定すると、<jazn> 要素が無視されます。

XMLUserManager クラスの使用

XMLUserManager は、ファイルベースのセキュリティ・モデルです。ユーザー、ロール、グループ、パスワードのすべてが principals.xml に格納されます。パスワードが平文で指定される場合があるため、セキュアではありません。

ただし、JAZNUserManager クラスのかわりに XMLUserManager クラスを使用する場合、すべてのアプリケーションが対象になるときはグローバル application.xml ファイルを、特定のアプリケーションのみが対象になるときは orion-application.xml ファイルを変更する必要があります。次の行を追加します。

```
<principals path="./principals.xml" />
```

パスは、principals.xml ファイルの場所を指しています。また、このファイル内の <jazn> 要素を削除するか、コメント化する必要があります。<jazn> 要素を削除もコメント化もしない場合は、最初に指定した方の要素がアプリケーションのユーザー・マネージャになります。たとえば、次のように指定するとします。

```
<principals path="./principals.xml" />
<jazn provider="XML"
location=".../j2ee/home/config/jazn-data.xml" />
```

この場合、<principals> 要素が先に出現しているため、XMLUserManager がセキュリティ・マネージャになります。

独自のユーザー・マネージャの作成

OC4J 付属のユーザー・マネージャの中に自分のユーザー認証ニーズに適したものがない場合は、独自のユーザー・マネージャを作成して、それを使用するように OC4J を構成できます。

独自のユーザー・マネージャを作成するには、次の手順を実行します。

1. カスタム・ユーザー・マネージャを記述します。

   カスタム・ユーザー・マネージャ・クラスでは、com.evermind.security.UserManager インタフェースを実装する必要があります。表 3-2 に、このインタフェースのメソッドを示します。

表 3-2 UserManager インタフェースのメソッド

メソッド	説明
void addDefaultGroup (java.lang.String name)	デフォルト・グループ・セットにグループを追加します。ユーザー・マネージャのすべてのユーザーがこのグループのメンバーになります。 ■ java.lang.String name: デフォルト・グループに追加するグループの名前
Group createGroup (java.lang.String name)	新規グループを作成します。そのグループがすでに存在する場合は、java.lang.InstantiationException がスローされます。 ■ java.lang.String name: 新規グループの名前
User createUser (java.lang.String username, java.lang.String password)	新規ユーザーを作成します。 ■ java.lang.String username: 新規ユーザーの名前 ■ java.lang.String password: 新規ユーザーのパスワード
User getAdminUser()	デフォルトの admin ユーザーを返します。デフォルトの admin ユーザーが存在しない場合は、null を返します。
User getAnonymousUser()	デフォルトの anonymous ユーザーを返します。デフォルトの anonymous ユーザーが存在しない場合は、null を返します。
java.util.Set getDefaultGroups()	ユーザー・マネージャのデフォルト・グループ・セットを返します。
Group getGroup(java.lang.String name)	指定した名前のグループを返します。そのグループが存在しない場合は、null を返します。 ■ java.lang.String name: 指定したグループの名前
int getGroupCount()	ユーザー・マネージャに含まれるユーザーの数を返します。サポートされない場合は、UnsupportedOperationException をスローします。

表 3-2 UserManager インタフェースのメソッド（続き）

メソッド	説明
`java.util.List getGroups (int start,int max)`	ユーザー・マネージャに含まれるグループのリスト（指定した索引間）を返します。サポートされない場合は、`UnsupportedOperationException` をスローします。
`UserManager getParent()`	ユーザー・マネージャの親マネージャを返します。
`User getUser (java.lang.String username)`	指定したユーザー名のユーザーを返します。該当するユーザーが存在しない場合は、`null` を返します。
`User getUser (java.lang.String issuerDN, java.math.BigInteger serial)`	この証明書に関連付けられたユーザーを返します。証明書がサポートされない場合、またはこの証明書に関連付けられたユーザーが存在しない場合は、`null` を返します。
`User getUser (java.security.cert.X509Certificate certificate)`	この証明書に関連付けられたユーザーを返します。証明書がサポートされない場合、またはこの証明書に関連付けられたユーザーが存在しない場合は、`null` を返します。
`int getUserCount()`	このマネージャに含まれるユーザーの数を返します。サポートされない場合は、`UnsupportedOperationException` をスローします。
`java.util.List getUsers (int start,int max)`	このマネージャに含まれるユーザーのリスト（指定した索引間）を返します。サポートされない場合は、`UnsupportedOperationException` をスローします。
`void init (java.util.Properties properties)`	指定した設定でユーザー・マネージャをインスタンス化します。エラーが発生した場合は、`java.lang.InstantiationException` をスローします。
`boolean remove(Group group)`	指定したグループをユーザー・マネージャから削除し、処理が成功した場合には `true` を返します。
`boolean remove(User user)`	指定したユーザーをユーザー・マネージャから削除し、処理が成功した場合には `true` を返します。
`void setParent (UserManager parent)`	親ユーザー・マネージャが存在する場合は、それを設定します。このメソッドは、ネストしたユーザー・マネージャに対してのみコールされます。 ユーザー・マネージャは、親ユーザー・マネージャに処理を委任できます。

2. ユーザー・マネージャをアプリケーションにプラグ・インします。

   単一アプリケーションに対しては、`orion-application.xml` ファイルの `<user-manager>` 要素にカスタム・ユーザー・マネージャを指定します。サーバー内のすべてのアプリケーションに対しては、`config/application.xml` ファイルの `<user-manager>` 要素にカスタム・ユーザー・マネージャを指定します。

3. ユーザーおよびグループを定義します。

   3-3 ページの「ユーザーおよびグループの指定」を参照してください。

4. Web アプリケーションのセキュリティ制約を作成します。

   3-6 ページの「認可」を参照してください。

例 3-4 DataSourceUserManager クラスの使用

次の `DataSourceUserManager` クラスの例は、`UserManager` インタフェースを実装するカスタム・ユーザー・マネージャを示しています。`DataSourceUserManager` クラスは、そのメソッドの中で、`DataSource` インタフェースによって指定されたデータベース内のユーザーを管理します。

カスタム・ユーザー・マネージャを構成するには、グローバル application.xml ファイルまたは orion-application.xml ファイルで、<user-manager> 要素の class 属性にクラス名を指定します。その後、1 つ以上の <property> 要素の name/value 属性を使用して、入力パラメータと値を指定します。

DataSourceUserManager の例では、<property> 要素の name/value ペアに表名と列を定義する必要があります。この例では、次の入力パラメータを設定しています。

- 表が存在するデータベースを示すデータソース
- ユーザー名とパスワードを格納する表
- ユーザーとグループの対応を格納する表

アプリケーションのユーザー・マネージャを登録する場合、通常は orion-application.xml に次のように指定します。

```
<user-manager class="com.evermind.sql.DataSourceUserManager">
      <property name="dataSource" value="jdbc/OracleCoreDS" />
      <property name="table" value="j2ee_users" />
      <property name="usernameField" value="username" />
      <property name="passwordField" value="password" />
      <property name="groupMembershipTableName" value="second_table" />
      <property name="groupMembershipGroupFieldName" value="group" />
      <property name="groupMembershipUserNameFieldName" value="userId" />
</user-manager>
```

<user-manager> プロパティ要素では、UserManager クラスへの入力パラメータを定義します。参照される表がデータベースにすでに存在することが前提となります。

ユーザー・マネージャは、親子関係を利用した階層形式の実装です。DataSourceUserManager クラスの親はデフォルトのファイルベースの XMLUserManager クラスであり、principals.xml ユーザー・リポジトリを使用します。ただし、setParent() メソッドで親を変更できます。例にあげた DataSourceUserManager クラスは、parent.getGroups() を起動して、使用可能なすべてのグループを親の XMLUserManager から取得します。

SSL による機密保護

OC4J は、クライアントとスタンドアロン OC4J 間の HTTPS を使用した Secure Socket Layer（SSL）通信をサポートしています。

次の項では、SSL について詳しく説明します。

- OC4J スタンドアロンでの SSL 使用の概要
- SSL 用の OC4J の構成
- HTTPS のよくある問題と解決策

OC4J スタンドアロンでの SSL 使用の概要

次の項では、セキュリティ機能を紹介し、OC4J スタンドアロンでのセキュリティ機能の使用方法について説明します。

- SSL 鍵および証明書の概要
- OC4J スタンドアロンでの証明書の使用

SSL 鍵および証明書の概要

2 つのエンティティ間の SSL 通信では、各エンティティ（1 つ以上のサーバー）に公開鍵と秘密鍵が 1 つずつ関連付けられます。通信時には、各エンティティが自分の秘密鍵と相手の公開鍵を使用することで、相互に通信可能になります。一方のエンティティが自分の秘密鍵を使用してデータを暗号化した場合、相手は元のエンティティの公開鍵を使用しないとデータを復号化できません。一方のエンティティが相手の公開鍵を使用してデータを暗号化した場合、相手は自分の秘密鍵を使用しないとデータを復号化できません。

それぞれの鍵は数値です。鍵には、1 つのエンティティだけの秘密にしておく秘密鍵と、セキュア通信を行う必要のあるすべての相手に対して公開される公開鍵があります。

証明書とは、エンティティの公開鍵の検証を行う公認の発行者から入手したデジタル署名付きの声明文です。このような発行者を認証局（CA）と呼びます。発行された証明書には、通常、ルート証明書が関連付けられています。このような証明書の関連付け（連鎖）により、信頼の連鎖が確立されます。発行者は独自のルート証明書を持つ場合があり、発行するすべての証明書を独自のルート証明書に連鎖させます。

機能的には、証明書は鍵のコンテナの役割を果たし、秘密鍵（該当する場合）、公開鍵および関連する署名がその中に格納されます。1 つの証明書ファイルに証明連鎖全体を含めることができます。

キーストアは、証明書（信頼できるすべての当事者の証明書を含む）の格納に使用されます。OC4J などのエンティティは、キーストアを使用して、他の当事者に対する自己認証を行います。

キーストアは、Sun Microsystems JDK 付属の `keytool` ユーティリティを使用して作成および操作できる `java.security.KeyStore` インスタンスです。`keytool` の詳細は、次のサイトを参照してください。

`http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html`

クライアントと OC4J 間のセキュア通信では、次の機能が実行されます。

- 二者間のリンク（すべての通信）が暗号化されます。
- セキュリティのチャレンジおよびレスポンスにより、OC4J がクライアントに対して認証されます。秘密鍵が安全に交換され、リンクの暗号化に使用されます。
- （オプション）OC4J がクライアント認証モードの場合、クライアントが OC4J に対して認証されます。

OC4J スタンドアロンでの証明書の使用

OC4J での SSL 通信に鍵と証明書を使用する手順は次のとおりです。これはサーバーレベルの手順です。通常は、セキュア通信が必要とされるアプリケーションのデプロイに先立って、OC4J の最初の設定時に実行します。

1. `keytool` を使用して、秘密鍵、公開鍵および未署名の証明書を生成します。この情報は、新規キーストアと既存キーストアのどちらにも格納できます。
2. 次のいずれかの方法で証明書の署名を取得します。
   - `keytool` を使用して独自の署名を生成し、証明書を自己署名します。この方法は、唯一のクライアントが事実上の独自の認証局を信頼する場合に適しています。
   - 次の手順で、公認の認証局から署名を取得します。

   a. 手順 1 の証明書を利用し、`keytool` を使用して証明書リクエストを生成します。これは、認証局に証明書への署名を求めるリクエストです。

   b. 証明書リクエストを認証局に送信します。

   c. 認証局から署名を受信し、`keytool` を使用してキーストアに署名をインポートします。キーストアでは、署名が関連の証明書と照合されます。

署名のリクエストおよび受信プロセスは、利用する認証局によって異なります。詳細は、各認証局の Web サイトを参照してください。どのブラウザにも、信頼できる認証局のリストがあります。VeriSign 社と Thawte 社（VeriSign 社が買収）の Web アドレスを例として次に示します。

```
http://www.verisign.com/
http://www.thawte.com/
```

また、オラクル社も認証局を提供しています。ただし、各証明書は Oracle アプリケーションでのみ認識されます。Oracle 認証局（OCA）を利用すると、顧客は自身とユーザーに対する証明書を作成および発行できますが、この証明書は、事前に手続きしておかないと顧客の組織外部では認識されません。OCA の詳細は、『Oracle Application Server セキュリティ・ガイド』を参照してください。

SSL 用の OC4J の構成

クライアントと OC4J 間でセキュア通信を行うには、OC4J スタンドアロンでの構成が必要です。クライアント認証を構成する場合にかぎり、クライアント側で証明書を提供する必要があります。

OC4J の `http-web-site.xml` ファイル（または、該当する場合はその他の Web サイト XML ファイル）で、`<web-site>` 要素に適切な SSL 設定を指定する必要があります。

1. 次のように `secure` フラグをオンにして、セキュア通信を指定します。

   ```
   <web-site ... protocol="http" secure="true" ... >
      ...
   </web-site>
   ```

   `secure="true"` という設定は、HTTP プロトコルが SSL ソケットを使用することを示します。

2. `<ssl-config>` サブ要素とその `keystore` および `keystore-password` 属性を使用して、次のようにキーストアのディレクトリ・パスおよびパスワードを指定します。

   ```
   <web-site ... secure="true" ... >
      ...
      <ssl-config keystore="path_and_file" keystore-password="pwd" />
   </web-site>
   ```

   `secure` フラグが `"true"` に設定されている場合は、必ず `<ssl-config>` 要素が必要です。

   `path_and_file` 値は、絶対ディレクトリ・パスと相対ディレクトリ・パスのどちらでもかまいません。この値にはファイル名も含めます。

   > **注意：** パスワードの間接化によって、パスワードを隠すことができます。パスワードの間接化の詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

3. オプションとして、次のように `<ssl-config>` 要素の属性 `needs-client-auth` フラグをオンにし、クライアント認証が必要であることを指定します。

   ```
   <web-site ... secure="true" ... >
      ...
      <ssl-config keystore="path_and_file" keystore-password="pwd"
                  needs-client-auth="true" />
   </web-site>
   ```

   この手順では、クライアントの識別情報に応じて、OC4J がセキュア通信を行うクライアント・エンティティを許可または拒否するモードが設定されます。`needs-client-auth` 属性は、接続時にクライアント証明連鎖をリクエストするよう OC4J に指示します。クライアントのルート証明書が認識されれば、クライアントは許可されます。

<ssl-config> 要素に指定したキーストアには、OC4J への HTTPS 接続が認可されたクライアントの証明書を格納する必要があります。

4. オプションとして、Web サイトの各アプリケーションを共有に指定します。<web-app> 要素の shared 属性は、複数のバインド（異なる Web サイトまたはポートとコンテキスト・ルート）が共有可能かどうかを示します。サポートされる値は、"true" および "false"（デフォルト）です。

   共有とは、セッション、サーブレット・インスタンス、コンテキスト値など、Web アプリケーションのあらゆる構成要素を共有することを意味します。このモードの一般的な使用方法は、すべてではなく一部の通信で SSL が必要とされる場合に、同じコンテキスト・パスにある HTTP サイトと HTTPS サイトの間で Web アプリケーションを共有するというものです。すべての情報ではなく、機密性の高い情報のみを暗号化することで、パフォーマンスが向上します。

   HTTPS Web アプリケーションが共有に設定されている場合、セッションの追跡には、SSL 証明書ではなく Cookie が使用されます。SSL 証明書を追跡する際には 1 つの証明書を格納するのに 50K 消費され、その結果セッションがタイムアウトする前にメモリー不足の問題が発生する場合があることを考えると、これは効果的です。Web アプリケーションの安全性が低下する恐れはあるものの、一部のブラウザで SSL セッション・タイムアウトが正しくサポートされないなどの問題を回避する上で必要な措置であると言えます。

5. オプションとして、shared が true で、デフォルト・ポートを使用しない場合は、Cookie ドメインを設定します。クライアントが別々のポートで Web サーバーとやり取りする場合、Cookie は別々のポートがそれぞれ別々の Web サイトを指していると解釈します。HTTP 用にデフォルト・ポート 80 を、HTTPS 用にデフォルト・ポート 443 を使用すると、クライアントはこれらを同じ Web サイトの 2 つの異なるポートと認識し、Cookie を 1 つだけ作成します。しかし、デフォルト以外のポートを使用すると、Cookie ドメインを指定しないかぎり、クライアントはこれらのポートが同じ Web サイトに属すると認識せず、ポートごとに別々の Cookie を作成します。

   Cookie ドメインは、DNS ドメイン内の複数のサーバーにまたがってクライアントの通信を追跡します。HTTP と HTTPS を利用した共有環境でデフォルト以外のポートを使用する場合、アプリケーションの orion-web.xml ファイルにある <session-tracking> 要素に cookie-domain 属性を設定します。cookie-domain 属性には、ドメイン名の 2 つ以上のコンポーネントで構成される DNS ドメインを指定します。

   ```
   <session-tracking cookie-domain=".oracle.com" />
   ```

例 3-5 クライアント認証を行う HTTPS 通信

次の例では、クライアント認証を行う HTTPS セキュア通信用に Web サイトを構成します。

```
<web-site display-name="OC4J Web Site" protocol="http" secure="true" >
   <default-web-app application="default" name="defaultWebApp" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome"
               needs-client-auth="true" />
</web-site>
```

セキュリティに特有なのは太字の部分のみです。セキュア通信を使用するかどうかにかかわらず、プロトコル値は常に HTTP 通信を示す "http" です。プロトコル値 http と secure="false" の組合せは HTTP プロトコルを意味します。http と secure="true" の組合せは HTTPS プロトコルを意味します。

次に、HTTP と HTTPS の両方の接続を受け入れるよう新規アプリケーションを構成します。

```
<web-app application="news" name="news-web" root="/news" shared="true" />
```

この Web サイトは、HTTP および HTTPS 通信にデフォルトのポート番号を使用します。デフォルトのポート番号を使用しない場合は、cookie-domain 属性も追加します。

```
<session-tracking cookie-domain=".oracle.com" />
```

<web-site>、<web-app> および <session-tracking> 要素とその属性の詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』の XML に関する付録を参照してください。

例 3-6 SSL 証明書の作成と HTTPS の構成

次の例では、keytool を使用してテスト用の証明書を作成し、HTTPS が機能するために必要なすべての XML 構成を示します。本番環境用の有効な証明書を作成する場合は、keytool のマニュアルを参照してください。

1. 正しい JDK をインストールします。

   JDK 1.3.x がインストールされていることを確認します。これは、OC4J で SSL を使用するために必要です。JAVA_HOME を JDK 1.3 のディレクトリに設定します。JDK 1.3.x の JAVA_HOME/bin がパスの先頭に指定されていることを確認します。そのためには、次のコマンドを実行します。

   UNIX

   ```
   $ PATH=/usr/opt/java130/bin:$PATH
   $ export $PATH
   $ java -version
   java version "1.3.0"
   ```

   Windows

   ```
   set PATH=d:¥jdk131¥bin;%PATH%
   ```

   この JDK バージョンが Windows レジストリで現在のバージョンに設定されていることを確認します。Windows レジストリ エディタで、HKEY_LOCAL_MACHINE/SOFTWARE/JavaSoft/Java Development Kit の下にある CurrentVersion を 1.3（またはそれ以上）に設定します。

2. 証明書をリクエストします。

   a. ディレクトリを ORACLE_HOME/j2ee に変更します。

   b. keytool コマンドを使用し、RSA の秘密鍵と公開鍵のペアでキーストアを作成します。たとえば次の構文では、RSA 鍵ペア生成アルゴリズムを使用して、mykeystore という名前のファイルの中に、パスワードが 123456 で 21 日間有効なキーストアが生成されます。

   ```
   keytool -genkey -keyalg "RSA" -keystore mykeystore -storepass 123456 -validity 21
   ```

   各項目の説明 :

   - keystore オプションでは、鍵を格納するファイルの名前を設定します。
   - storepass オプションでは、キーストアを保護するためのパスワードを設定します。
   - validity オプションでは、証明書が効力を持つ日数を設定します。

   次に示すように、keytool ではその他の情報の入力も求められます。

   ```
   keytool -genkey -keyalg "RSA" -keystore mykeystore -storepass 123456 -validity 21

   What is your first and last name?
     [Unknown]:  Test User
   What is the name of your organizational unit?
     [Unknown]:  Support
   What is the name of your organization?
     [Unknown]:  Oracle
   What is the name of your City or Locality?
     [Unknown]:  Redwood Shores
   What is the name of your State or Province?
     [Unknown]:  CA
   What is the two-letter country code for this unit?
     [Unknown]:  US
   ```

```
Is <CN=Test User, OU=Support, O=Oracle, L=Reading, ST=Berkshire, C=GB> correct?
  [no]:  yes

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

注意： 2 文字の国コードを調べるには、http://www.bcpl.net/~jspath/isocodes.html にある ISO 国コード・リストを使用してください。

mykeystore ファイルは、カレント・ディレクトリに作成されます。鍵のデフォルト別名は mykey です。

3. secure-web-site.xml ファイルがない場合、http-web-site.xml を $J2EE_HOME/config/secure-web-site.xml にコピーします。
4. secure-web-site.xml を編集し、次の要素を追加します。
   a. 次のように、<web-site> 要素に secure="true" を追加します。

      ```
      <web-site port="8888" display-name="Default Oracle Application Server
      Containers for J2EE Web Site" secure="true">
      ```

   b. <web-site> 要素に次の新規行を追加し、キーストアおよびパスワードを定義します。

      ```
      <ssl-config keystore="<Your-Keystore>" keystore-password="<Your-Password>" />
      ```

      <Your-Keystore> はキーストアのフルパス、<Your-Password> はキーストアのパスワードです。たとえば、次のように指定します。

      ```
      <!-- Enable SSL -->
      <ssl-config keystore="../../keystore" keystore-password="123456"/>
      ```

      注意： キーストアのパスは、XML ファイルの場所に対する相対パスです。

   c. web-site のポート番号を、使用可能なポートを使用するように変更します。たとえば、SSL ポートのデフォルトが 443 であることから、Web サイトのポート属性を port="4443" に変更します。デフォルトの 443 は、スーパー・ユーザーでないと使用できません。
   d. secure-web-site.xml に加えた変更を保存します。
5. secure-web-site.xml ファイルがなかった場合、secure-web-site.xml ファイルを指すように server.xml を編集します。
   a. secure-web-site.xml ファイルが読み取られるように、ファイル server.xml の次の行を非コメント化するか追加します。

      ```
      <web-site path="./secure-web-site.xml" />
      ```

      注意： Windows の場合も、XML ファイルでは円記号ではなくフォワード・スラッシュを使用します。

   b. server.xml に加えた変更を保存します。
6. OC4J を停止して再起動し、secure-web-site.xml ファイルへの追加事項を初期化します。ブラウザで SSL ポートからサイトにアクセスし、SSL ポートをテストします。証明書が認証局によって署名されていないため、アクセスに成功した場合は証明書を受け入れるよう求められます。

構成が完了すると、OC4J は一方のポートで SSL リクエストを、もう一方のポートで非 SSL リクエストをリスニングします。server.xml 構成ファイルで該当する *web-site.xml ファイルをコメント化すれば、SSL リクエストと非 SSL リクエストのどちらも無効にできます。

```
<web-site path="./secure-web-site.xml" /> - comment out this to remove SSL
<default-site path="./http-web-site.xml" /> - comment out this to
                                                    remove non-SSL
```

OC4J スタンドアロンでのクライアント認証のリクエスト

OC4J は、サーバーがクライアントと通信する前にクライアントからの認証を明示的にリクエストする、クライアント認証モードをサポートしています。この場合、クライアントに独自の証明書が必要です。クライアントは、証明書とルート証明書で終わる証明連鎖を送信することで、自己認証を行います。そのクライアントに戻る信頼の連鎖を確立する上で、指定したリストからのルート証明書のみを受け入れるように OC4J を構成できます。

OC4J が信頼する証明書は、トラスト・ポイントと呼ばれます。これは、キーストア内に一致するクライアントがある場合に、そのクライアントからの連鎖の中で OC4J が検出する最初の証明書です。信頼を構成するには 3 つの方法があります。

- クライアント証明書をキーストアに入れます。
- クライアントの連鎖に含まれる中間認証局証明書のいずれかをキーストアに入れます。
- クライアントの連鎖に含まれるルート認証局証明書をキーストアに入れます。

OC4J は、偽造証明書の防止のために、トラスト・ポイントまで（トラスト・ポイントを含む）の証明連鎖全体が有効であるかどうかを検証します。

needs-client-auth 属性を使用してクライアント認証をリクエストする場合、次の手順を実行します。

1. クライアントの連鎖に含まれるどの証明書をトラスト・ポイントとするかを決定します。必ず、このトラスト・ポイントを使用して証明書の発行を管理するか、または認証局を発行者として信頼します。
2. サーバー・キーストア内の中間証明書またはルート証明書を、クライアント証明書の認証に使用するトラスト・ポイントとしてインポートします。
3. OC4J が特定のトラスト・ポイントにアクセスするのを防止するには、そのトラスト・ポイントがキーストア内にないことを確認します。
4. 前述の手順を実行して、クライアント証明書を作成します。この証明書には、サーバーにインストールされた中間証明書またはルート証明書が含まれます。別の認証局を信頼するには、その認証局から証明書を取得します。
5. 証明書をクライアント上のファイルに保存します。
6. クライアントでの HTTPS 接続の開始時に、証明書を指定します。
   a. クライアントがブラウザの場合、クライアント・ブラウザのセキュリティ領域内に証明書を設定します。
   b. クライアントが Java クライアントの場合、HTTPS 接続の開始時にクライアント証明書および証明連鎖をプログラム的に指定する必要があります。

HTTPS のよくある問題と解決策

SSL 証明書の使用時に、次のエラーが発生することがあります。

Keytool Error: java.security.cert.CertificateException: Unsupported encoding

原因 : keytool では後続の空白を指定できません。

処置 : 後続の空白をすべて削除します。引き続きエラーが発生する場合は、証明書の応答ファイルに新規行を追加します。

Keytool Error: KeyPairGenerator not available

原因 : 古い JDK の keytool を使用している可能性があります。

処置 : システム上の最新の JDK の keytool を使用します。最新の JDK を確実に使用するには、最新の JDK のフルパスを指定します。

Keytool Error: Failed to establish chain from reply

原因 : キーストアにルート CA 証明書が見つかりません。そのため、keytool はサーバー鍵から信頼できるルート認証局への証明連鎖を構築できません。

処置 : 次のコマンドを実行します。

```
keytool -keystore keystore -import -alias cacert -file cacert.cer (keytool
-keystore keystore -import -alias intercert -file inter.cer)
```

中間 CA の keytool を使用する場合は、次のコマンドを実行します。

```
keystore keystore -genkey -keyalg RSA -alias serverkey keytool -keystore keystore
-certreq -file my.host.com.csr
```

証明書署名リクエストから証明書を取得し、次のコマンドを実行します。

```
keytool -keystore keystore -import -file my.host.com.cer -alias serverkey
```

IllegalArgumentException: Mixing secure and non-secure sites on the same ip + port

原因 : SSL および非 SSL の Web サイトが同じポートおよび IP アドレスをリスニングするような構成は行えません。

処置 : `secure-web-site.xml` および `http-web-site.xml` ファイルに異なるポートが割り当てられていることを確認します。

Keytool does not work on HP-UX

原因 : HP-UX では、RSA オプションを指定すると keytool が機能しないことが報告されています。

処置 : 別のプラットフォームでキーを生成し、生成したキーを HP-UX サーバーに FTP 送信します。

一般的な SSL デバッグ

JSSE 実装から、より多くのデバッグ情報を取得できます。オプションのリストを表示するには、次のように OC4J を起動します。

```
java -Djavax.net.debug=help -jar oc4j.jar
```

詳細表示をオンにする場合は、次のコマンドを使用します。

```
java -Djavax.net.debug=all -jar oc4j.jar
```

どちらの場合も、次の項目が表示されます。

- ブラウザ・リクエスト・ヘッダー
- サーバー HTTP ヘッダー
- サーバー HTTP ボディ（サービスの提供を受ける HTML）
- コンテンツ長（暗号化前および後）
- SSL バージョン

UNIX の場合、NOTE 150215.1「Scripts to Administer OC4J on Unix Platforms」の起動スクリプトを修正して使用してください。

A

OC4J のトラブルシューティング

この付録では、OC4J の使用時に発生する可能性のある一般的な問題とその解決方法について説明します。この付録の項目は次のとおりです。

問題と解決策

この項では、一般的な問題と解決策について説明します。この章の項目は次のとおりです。

JDK 1.3 の使用時に OC4J を起動できない

問題

JDK 1.3 の使用時に OC4J が失敗します。

解決策

このような起動失敗は、ロギング実装上の依存性の問題が原因です。この問題を解決するには、`ORACLE_HOME/j2ee/home/config/server.xml` の次のエントリを削除するか、コメント化します。

```
<j2ee-logging-config path="./j2ee-logging.xml" />
```

OracleAS JMS がアクティブな場合、異常終了後に OC4J を再起動できない

問題

OracleAS JMS で永続性が有効になっている場合、JMS サーバーでは永続的なキュー / トピックが作成されます。また、これらのキュー / トピックに関連したロック・ファイル（`.lock`）が /persistence ディレクトリに作成されます。`kill -9` を使用するなどの方法で JVM を異常終了した場合、ロック・ファイルは削除されません。そのため、OC4J を再起動できないという状況が生まれます。

解決策

手動で `/persistence` ディレクトリから `.lock` ファイルをすべて削除します。

ステートフル・レプリケーションが OC4J インスタンス全体で一貫して行われない

問題

よくあるのは、OC4J インスタンス A からインスタンス B へのフェイルオーバーは行われるが、B から A へのフェイルバックが行われない、というケースです。

解決策

OC4J では、ステートフル・レプリケーションをすべてのアプリケーションに対してグローバルに設定する必要はなく、orion-web.xml ディスクリプタ・ファイルの <cluster-config> 要素を使用して、Web モジュールごとにレプリケーションを構成できます。この要素が各 Web モジュールのディスクリプタに正しく挿入されていることを確認します。

OPMN 管理の OC4J がクラスタリングをサポートしているのはグローバル・レベルのみです。アプリケーションまたはモジュールのレベルではサポートしていません。

未認定バージョンの JDK を OC4J でのみ使用

問題

これは、すべての Oracle Application Server コンポーネント用に認定されているバージョンより新しいバージョンの JDK を OC4J で使用したいというケースです。しかし、新しいバージョンの JDK をすべてのコンポーネントでグローバルに使用すると、認定が無効になる可能性が高まります。

解決策

新しいバージョンの JDK を OC4J でのみ使用するには、opmn.xml 構成ファイルの <java-bin> 要素に JDK の場所を指定します。たとえば、次のように入力します。

```
<module-data>
  <category id="start-parameters">
    <data id="java-bin" value="/myjavalocation/jdk/bin/java"/>
  </category>
</module-data>
```

OC4J の実行時に java.lang.OutOfMemory エラーがスローされる

問題

このエラーは、Java インスタンスのヒープ・サイズが OC4J でのアプリケーションの実行に必要なメモリーより小さいことを示します。

解決策

opmn.xml の <java-option> 要素で -Xmx を必要なメモリー・サイズに設定し、ヒープ・サイズを増やします。

```
<module-data>
  <category id="start-parameters">
    <data id="java-options" value="-Xmx256M" />
  </category>
</module-data>
```

または、OC4J の起動時にシステム・プロパティを設定します。

```
java -Xmx256M -jar oc4j.jar
```

UNIX または Linux 上で実行している場合、JVM プロセスがそれだけのメモリーを割り当てられるかどうか、ulimit の設定を確認します。

ステートフル・ファイアウォールによる接続タイムアウトがシステム・パフォーマンスに影響

問題

パフォーマンスを高めるために、各 Oracle HTTP Server プロセスの mod_oc4j コンポーネントは、リクエストの送信先となる各 OC4J インスタンスで、AJP ポートへの TCP 接続をオープンしておきます。

OHS と OC4J の間にファイアウォールが存在する場合、接続のアイドル時間がステートフル・ファイアウォールの非アクティブ・タイムアウトを超えると、AJP 経由で送信されたパッケージは拒否されます。

ただし、AJP ソケットはクローズされません。ソケットがオープンしているかぎり、ワーカー・スレッドはソケットと結び付いており、スレッド・プールには戻されません。OC4J はスレッドを作成し続け、最終的にシステム・リソースが不足します。

解決策

OHS の TCP 接続は、ファイアウォール・タイムアウトの問題が発生しないように、常に有効にしておく必要があります。そのためには、OC4J の構成パラメータと Apache の実行時プロパティを組み合せて使用します。

`httpd.conf` または `mod_oc4j.conf` 構成ファイルに次のパラメータを設定します。`Oc4jConnTimeout` の値は、セッションが非アクティブとみなされるまでの非アクティブ状態の時間（秒数）です。

```
Oc4jUserKeepalive  on
```

```
Oc4jConnTimeout  12000（または同様の値）
```

また、OC4J の起動時に次の AJP プロパティを設定し、ファイアウォール・タイムアウトが原因で OHS と OC4J 間の接続が切断された場合には AJP ソケットがクローズされるようにします。

```
ajp.keepalive=true
```

たとえば、次のように入力します。

```
java -Dajp.keepalive=true -jar oc4j.jar
```

OPMN 管理の OC4J がデフォルト RMI ポートを通じて EJB リソースにアクセスできない

問題

OC4J が Oracle Application Server のコンポーネントとして実行されている場合、デフォルト RMI ポートを通じて EJB リソースにアクセスできません。

解決策

最も一般的な原因は、スタンドアロン OC4J に精通したユーザーが、`rmi.xml` に指定した値は OPMN 管理の Oracle Application Server 環境では使用されないことに気付かずに、このファイルから RMI ポートを読み取っていることです。

OPMN 管理の OC4J インスタンスは、動的 RMI ポート割当てを使用します。ポート値の範囲は、`opmn.xml` の `<port>` 要素に指定するか、アプリケーション・クライアントから `opmn:ormi` を動的に参照して決定されます。

詳細は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

アプリケーションのパフォーマンスが JVM ガベージ・コレクションの一時停止の影響を受ける

問題

OC4J 上で実行されているアプリケーションが無応答になり、単純なリクエストが著しく遅延します。JVM がメモリー不足のしきい値を超え、メモリーを解放しようとフル・ガベージ・コレクションを実行していることが原因です。

解決策

インクリメンタル・ロー・ポーズ・コレクタの使用を検討してください。マイナー・コレクションのたびにメジャー・コレクション処理を部分的に実行することで、メジャー・ガベージ・コレクションの一時停止が長期化するのを防止できます。このコレクタ（トレイン・コレクタとも呼びます）は、マイナー・コレクションのたびに、古い世代（通常はメジャー・コレクションで収集されるオブジェクトを格納するメモリー・プール）の一部を収集します。その結果、一時停止が短期化し、多くのマイナー・コレクションに分散されます。

全体的なスループットを考えると、インクリメンタル・コレクタはデフォルトの古い世代のコレクタよりもさらに低速です。

インクリメンタル・コレクタを使用するには、アプリケーションの起動時に Java コマンドラインで `-Xincgc` オプションを渡す必要があります。`-XX:NewSize` および `-XX:MaxNewSize` オプションを使用して、若い世代（オブジェクト・プール）の初期サイズと最大サイズを同じ値に設定します。`-Xms` および `-Xmx` オプションを使用して、Java の初期ヒープ・サイズと最大ヒープ・サイズを同じ値に設定します。

たとえば、1GB の物理メモリーを持つサーバーでこのコレクタを使用するには、次のように指定します。

```
java -server -Xincgc -XX:NewSize=64m -XX:MaxNewSize=64m -Xms512m -Xmx512m
```

ガベージ・コレクション・チューニングの詳細は、`http://java.sun.com/docs/hotspot/gc1.4.2/` の「Tuning Garbage Collection with the 1.4.2 JavaTM Virtual Machine」を参照してください。

無効または不要なライブラリ要素によりパフォーマンスが低下する

問題

プログラムの実行中に OC4J プロセス・メモリーが一貫して増大し続ける場合、グローバル `application.xml` ファイルで無効なシンボリック・リンクを参照している可能性があります。

一般に、この問題の特徴的な点は、通常の Java オブジェクト・メモリー・リークの場合のように Java オブジェクト・メモリーが増大するのではなく、C ヒープが増大することです。OC4J は、`application.xml` ファイルで定義されているすべてのリソースをロードします。このリンクが無効な場合、C ヒープが増大し続け、OC4J はメモリー不足の状態になります。

解決策

すべてのシンボリック・リンクが `application.xml` で有効であることを確認し、OC4J を再起動します。

また、OC4J がロードする JAR ファイルの数を最小限に抑えます。構成と OC4J が検索するディレクトリから、未使用の JAR ファイルをすべて削除します。OC4J は、すべての JAR ファイルでクラスおよびリソースを検索するため、ファイル・キャッシュが余分なメモリーおよびプロセッサ時間を消費する結果になります。

`application.xml` ファイルの `<library>` 要素が、必要な JAR および ZIP ファイルの位置するディレクトリではなく、それらのファイルを個別に指していれば、ロード処理をより正確に制御できます。

JSP エラー : タグが登録されていません

問題

このエラーは、JSP が OC4J サーバー内にないタグをコールしようとすると発生します。通常、この問題は、予約済のタグ・ライブラリの場所が 1 つ以上、OC4J 内で不十分に定義されている場合に発生します。

解決策

予約済のタグ・ライブラリの場所は、次の 2 段階のプロセスで定義します。

- global-web-application.xml ファイルの <orion-web-app> 要素の jsp-taglib-locations 属性に、ディレクトリを定義します。
- application.xml の <library> 要素の Path 属性に、ディレクトリを追加します。

このエラーは通常、第 2 段階の手順が完了していないことを示します。

別の方法として、デフォルトの予約済みのタグ・ライブラリの場所（ORACLE_HOME/j2ee/home/jsp/lib/taglib/）に、タグ・ライブラリを含む JAR ファイルをコピーすることもできます。

JSP エラー : クラス・ファイルの長さがゼロです

問題

このエラーは、Java クラスへのコンパイルに失敗した JSP を OC4J が処理しようとすると発生します。これは、Java コンパイラをロードできないか、メモリー不足になったため、.class ファイルが 0 バイトになったことが原因です。

解決策

0 バイトの .class ファイルを削除します。JSP を次回リクエストしたとき、このクラスはコンパイルされます。

JSP エラー : <choose> を直接の親としない <when>- スタイルのタグの使用が不正です

問題

このエラーは、JSP 標準タグ・ライブラリ（JSTL）タグを含む、リクエストされた JSP の処理に OC4J が失敗したときに発生します。この場合のタグは、<choose> です。これは、OC4J インスタンス内に複数の JSTL バージョンが存在していることが原因と考えられます。

解決策

OC4J にデフォルトでインストールされたバージョンの JSTL を削除します。このライブラリは、ORACLE_HOME/j2ee/home/jsp/lib/taglib ディレクトリに standard.jar ファイルとしてパッケージされています。

その他のサポート情報

サポートに重点を置いた次のオラクル社 Web サイトで、その他の解決策を検索できます。

- Oracle Technology Network（http://www.oracle.com/technology/documentation/index.html）から入手可能な Oracle Application Server のリリース・ノート
- Oracle MetaLink（http://metalink.oracle.com）

直面している問題の解決策が依然として見つからない場合は、オラクル社カスタマ・サポート・センターに問い合せてください。

B

追加情報

この付録には、次の項目に関する情報が含まれています。

- XML ファイルの内容の説明
- server.xml ファイルの要素
- application.xml ファイルの要素
- orion-application.xml ファイルの要素
- application-client.xml ファイルの要素
- orion-application-client.xml ファイルの要素
- スタンドアロン OC4J のコマンドライン・オプションおよびプロパティ
- OC4J システム・プロパティ
- 構成およびデプロイの例

XML ファイルの内容の説明

OC4J は、構成 XML ファイルおよびデプロイメント XML ファイルを使用します。次の項で、これらのファイルとその機能についてそれぞれ説明します。

OC4J 構成 XML ファイル

この項では、OC4J の構成に必要となる、次のような XML ファイルに関して説明します。

- server.xml
- http-web-site.xml
- jazn-data.xml
- principals.xml
- data-sources.xml
- jms.xml
- rmi.xml

server.xml

このファイルには、アプリケーション・サーバー用の構成が含まれています。`server.xml` ファイルは、ルート構成ファイルで、他の構成ファイルへの参照が含まれています。このファイルには、次のものを指定します。

- アプリケーション・デプロイメント・ディスクリプタに入っているライブラリ・パス
- サービスが提供される、グローバル・アプリケーション、グローバル Web アプリケーションおよびデフォルトの Web サイト
- サーバーが許容する最大 HTTP 接続数
- ロギング設定
- Java コンパイラ設定
- トランザクション・タイムアウト
- SMTP ホスト
- `data-sources.xml` 構成の場所
- JMS および RMI の構成の場所
- デフォルトおよび追加 Web サイトの場所

  これらの場所を指定するには、Web サイト構成ファイルの場所をリストするエントリを追加します。複数の Web サイトを使用できます。`http-web-site.xml` ファイルはデフォルトの Web サイトを定義するものであるため、これらの XML ファイルの中に 1 つしか存在しません。他のすべての Web サイトは、`web-site.xml` 構成ファイル内で定義されます。次のように、`server.xml` ファイル内に各 Web サイトを登録します。

  ```
  <web-site path="./http-web-site.xml" />
  <web-site path="./another-web-site.xml" />
  ```

  注意： 示されているパスは、`config/` ディレクトリに対する相対パスです。

- コンテナがデプロイおよび実行するすべてのアプリケーションを指すポインタ

  コンテナ上で実行されるアプリケーションを `server.xml` ファイルに指定します。アプリケーション・ディレクトリは必要な数を制限なく指定できます。これらのディレクトリは、OC4J インストール・ディレクトリの下に置く必要はありません。

http-web-site.xml

このファイルには、Web 用の構成が含まれています。http-web-site.xml ファイルには、次のものを指定します。

- ホスト名または IP アドレス、このサイトの仮想ホスト設定、リスナー・ポートおよび SSL を使用したセキュリティ
- このサイトのデフォルトの Web アプリケーション
- このサイトの追加の Web アプリケーション
- アクセス・ログ・フォーマット
- ユーザー Web アプリケーションの設定（/~user/ sites 用）
- SSL 構成

jazn-data.xml

このファイルには、OC4J サーバー用のセキュリティ情報が含まれています。これは、デフォルトの JAZNUserManager を使用するユーザーおよびグループ構成を定義します。

jazn-data.xml ファイルで、次を指定します。

- ユーザー名とパスワード
- ユーザーの名前と説明、グループおよびロール

principals.xml

このファイルには、OC4J サーバー用のセキュリティ情報が含まれています。これは、XMLUserManager（現在ではデフォルトのセキュリティ・マネージャではありません）を使用するユーザーおよびグループ構成を定義します。principals.xml ファイルには、次のものを指定します。

- client-admin コンソールのユーザー名およびパスワード
- ユーザーまたはグループの名前と説明、およびユーザーの実名とパスワード
- ユーザー用の X.509 証明書（オプション）

data-sources.xml

このファイルには、使用するデータソースの構成が含まれています。また、JDBC 接続の取得方法に関する情報も含まれています。data-sources.xml ファイルで、次のことを指定します。

- JDBC ドライバ
- JDBC URL
- データソースのバインド先の JNDI パス
- データソースのユーザー名およびパスワード
- 使用するデータベース・スキーマ
- 非アクティブのタイムアウト
- 許容される最大データベース接続数

注意： データベース・スキーマは、自動生成された SQL が別のデータベース・システムで動作するようにするために使用します。OC4J には、タイプ・マッピングと予約語などのプロパティを指定する、XML ファイル・フォーマットが含まれます。OC4J には、MS SQL Server/MS Access、Oracle および Sybase 用のデータベース・スキーマが付属しています。これらを編集したり、ご使用の DBMS 用の新規スキーマを作成することができます。

---

jms.xml

このファイルには、OC4J での Java Message Service（JMS）の実装に関する構成が含まれています。jms.xml ファイルで、次の事項を指定します。

- ホスト名または IP アドレス、および JMS サーバーがバインドするポート番号
- JNDI ツリー内にバインドされるキューおよびトピックの設定
- ログの設定

rmi.xml

このファイルには、Remote Method Invocation（RMI）システムの構成が含まれます。これには、EJB にリモート・アクセスを提供する RMI リスナーの設定が含まれます。rmi.xml ファイルで、次のことを指定します。

- ホスト名または IP アドレス、および RMI サーバーがバインドするポート番号
- 通信相手のリモート・サーバー
- ログの設定

J2EE デプロイ XML ファイル

OC4J 固有のデプロイ XML ファイルには、異なるコンポーネントのデプロイ情報が含まれます。OC4J 固有のファイルを作成しない場合は、アプリケーションのデプロイ時にファイルが自動的に生成されます。OC4J 固有のデプロイ XML ファイルを手動で編集することができます。OC4J は、これらのファイルを使用して、環境エントリ、リソース参照およびセキュリティ・ロールを実際のデプロイ固有の値にマップします。

この項では、J2EE アプリケーションのデプロイに必要な次の XML ファイルについて説明します。

- J2EE の application.xml ファイル
- OC4J 固有の orion-application.xml ファイル
- J2EE の ejb-jar.xml ファイル
- OC4J 固有の orion-ejb-jar.xml ファイル
- J2EE の web.xml ファイル
- OC4J 固有の orion-web.xml ファイル
- J2EE の application-client.xml ファイル
- OC4J 固有の orion-application-client.xml ファイル

J2EE の application.xml ファイル

このファイルは、J2EE アプリケーションに含まれている Web または EJB アプリケーションを識別します。要素の一覧は、B-14 ページの「application.xml ファイルの要素」を参照してください。

OC4J 固有の orion-application.xml ファイル

このファイルは、グローバル・アプリケーションを構成します。orion-application.xml ファイルで、次のことを指定します。

- CMP Bean のテーブルの自動作成および自動削除を行うかどうか
- CMP Bean とともに使用するデフォルトのデータソース
- セキュリティ・ロール・マッピング
- セキュリティ用のデフォルトのユーザー・マネージャ
- JNDI のネームスペースおよびアクセス・ルール（認可）

要素の一覧は、B-16 ページの「orion-application.xml ファイルの要素」を参照してください。

J2EE の ejb-jar.xml ファイル

このファイルは、この JAR ファイルの EJB のデプロイ・パラメータを定義します。要素の詳細は、Sun 社の EJB 仕様を参照してください。

OC4J 固有の orion-ejb-jar.xml ファイル

このファイルは、EJB 用の OC4J 固有デプロイメント・ディスクリプタです。`orion-ejb-jar.xml` ファイルで、次のことを指定します。

- タイムアウト設定
- トランザクションの再試行設定
- セッションの永続性設定
- トランザクションの独立性設定
- CMP マッピング
- OR マッピング
- finder メソッドの仕様
- JNDI マッピング
- インスタンス・プールの最小および最大設定
- リソース参照マッピング

要素の詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』の付録を参照してください。

J2EE の web.xml ファイル

このファイルには、このアプリケーションのサーブレットおよび JSP に関するデプロイ情報が含まれています。要素の詳細は、Sun 社の仕様を参照してください。

OC4J 固有の orion-web.xml ファイル

このファイルは、Web 設定マッピング用の OC4J 固有デプロイメント・ディスクリプタです。この XML ファイルには、次のようなものが含まれます。

- 自動再ロード（変更チェックと時間間隔を含む）
- バッファリング
- キャラクタ・セット
- 開発モード
- ディレクトリのブラウズ
- ドキュメント・ルート
- ロケール
- Web タイムアウト
- 仮想ディレクトリ
- セッション・トラッキング
- JNDI マッピング
- Web アプリケーションのクラスロードの優先順位

要素の詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』の付録を参照してください。

J2EE の application-client.xml ファイル

このファイルには、サーバー・アプリケーションにアクセスするための JNDI 情報、および他のクライアント情報が含まれています。要素の一覧は、B-21 ページの「application-client.xml ファイルの要素」を参照してください。

OC4J 固有の orion-application-client.xml ファイル

このファイルは、クライアント・アプリケーション用の OC4J 固有のデプロイメント・ディスクリプタです。これには、クライアントの JNDI マッピングとエントリが含まれています。

要素の一覧は、B-23 ページの「orion-application-client.xml ファイルの要素」を参照してください。

server.xml ファイルの要素

`server.xml` ファイルでは、次の作業を実行します。

- OC4J の構成
- 他の構成ファイルの参照
- ご使用の J2EE アプリケーションの指定

OC4J の構成

OC4J サーバーを構成するには、`server.xml` ファイル内で次のような要素を定義します。

- ライブラリ・パス
- グローバル・アプリケーション、グローバル Web アプリケーションおよびデフォルトの Web サイト
- サーバーが許容する最大 HTTP 接続数
- ロギング設定
- Java コンパイラ設定
- トランザクション・タイムアウト
- SMTP ホスト

他の構成ファイルの参照

`server.xml` ファイルの他の構成ファイルを参照するには、次のものを指定します。

- `data-sources.xml` の場所
- `jazn-data.xml` の場所
- `jms.xml` と `rmi.xml` の場所

`server.xml` ファイルには複数の XML ファイルとディレクトリが定義されています。これらのファイルまたはディレクトリへのパスは、相対パスまたは絶対パスにすることができます。相対パスの場合は、`server.xml` ファイルの場所と相対的なパスにする必要があります。

<application-server> 要素の説明

`server.xml` ファイルのトップ・レベルの要素は、`<application-server>` 要素です。

`<application-server>`

この要素には、アプリケーション・サーバー用の構成が含まれています。

属性:

- application-auto-deploy-directory="../applications/auto": ここで指定するディレクトリから EAR ファイルが自動的に検出され、稼働中の OC4J サーバーによってデプロイされます。さらに、デフォルト Web サイトの Web アプリケーション・バインドを実行します。
- auto-start-applications="true|false": true に設定すると、OC4J サーバーが起動されたときに、<applications> 要素に定義されているすべてのアプリケーションが自動的に起動されます。false に設定すると、アプリケーションの auto-start 属性が true に設定されなければ、アプリケーションは起動されません。auto-start-applications のデフォルトは、true です。
- application-directory="../applications": アプリケーション（EAR ファイル）を格納するディレクトリを指定します。何も指定されていない場合（デフォルト）、OC4J は、情報を j2ee/home/applications に格納します。
- deployment-directory="../application-deployments": EAR ファイルに含まれるアプリケーションがデプロイされるマスターの場所を指定します。この場所のデフォルトは、j2ee/home/application-deployments/ です。
- connector-directory: oc4j-connectors.xml ファイルの場所とファイル名です。
- check-for-updates="true|false": スタンドアロン OC4J でのデフォルトは、true です。true の場合、タスク・マネージャは XML 構成ファイルが変更されていないかどうかをチェックします。したがって、この属性を false に設定すれば、XML の新たな変更に対する構成の自動更新を無効にできます。また、この属性を false に設定すると、admin.jar -updateConfig を実行するまでアプリケーションのオート・デプロイが停止します。つまり、XML ファイルの内容を反映させて XML 構成を更新したり、オート・デプロイを行う必要がある場合は、admin.jar -updateConfig オプションを使用することになります。
- recovery-procedure="automatic|prompt|ignore": グローバル・トランザクション（JTA）の最中にエラーが発生した場合に、EJB コンテナがグローバル・トランザクション（JTA）をどのようにリカバリするかを指定します。エラー発生時に CMP Bean がグローバル・トランザクションの最中である場合、EJB コンテナはトランザクションの状態をファイルに保存します。次に OC4J が起動されるときに、これらの属性が JTA トランザクションのリカバリ方法を指定します。
  - automatic: 自動的にリカバリを試行（デフォルト）
  - prompt: ユーザーにプロンプトを表示（システム・インおよびアウト）

    CMP Bean が実行中でなかった場合にもリカバリのプロンプトが表示されることがあります。このような場合は、リカバリが必要なものがあったかどうかを調べる許可を OC4J サーバーが求めています。
  - ignore: リカバリを無視（開発環境に有用、または CMP Entity Bean を実行しない場合に有用）
- taskmanager-granularity=*milliseconds*: タスク・マネージャは、クリーン・アップを実行するバックグラウンド・プロセスです。ただし、タスク・マネージャを使用するとコストが高くなる可能性があります。タスク・マネージャが実行されるタイミングを、この属性で管理できます。この属性は、タスク・マネージャがクリーン・アップのために起動される頻度を設定します。値はミリ秒単位です。デフォルトは 1000 ミリ秒です。

<application-server> 内に含まれる要素

<application-server> 要素の中には、次の要素を構成できます（DTD での順序ではなく、アルファベット順に説明します）。

<application>

アプリケーションは、それ自身のユーザー、Web アプリケーションおよび EJB JAR ファイルを持つエンティティです。

属性 :

- auto-start="true|false": OC4J サーバーの起動時にアプリケーションを自動的に起動するかどうかを指定します。デフォルトは true です。複数のアプリケーションがインストールされており、それらを必要に応じて起動する場合には、auto-start を false に設定すると便利です。これにより、通常のサーバー起動時間とリソース使用率を改善できます。
- deployment-directory=".../application-deployments/myapp": アプリケーション・デプロイメント情報を格納するディレクトリを指定します。何も指定しない場合(デフォルト)、OC4J はグローバル deployment-directory を検索し、そこに何も存在しなければ、EAR ファイル内の情報を格納します。このパスは、相対パスでも絶対パスでもかまいません。相対パスの場合は、server.xml ファイルの場所と相対的なパスにする必要があります。
- name="anApplication": アプリケーションを参照するために使用する名前を指定します。
- parent="anotherApplication": オプションの親アプリケーションの名前。デフォルトはグローバル・アプリケーションです。子は、自分の親アプリケーションのネームスペースを参照します。これは、EJB などのサービスを複数のアプリケーション間で共有するために使用します。
- path=".../applications/myApplication.ear" />: アプリケーション・コードを含む EAR ファイルへのパス。この例では、EAR ファイルの名前を myApplication.ear としています。

<compiler>

この要素は、リリース 9.0.4 以降では使用されなくなりました。かわりに使用される要素については、<java-compiler> 要素を参照してください。以前のリリースでは、EJB または JSP コンパイル用の、代替コンパイラ（Jikes など）を指定します。

属性 :

- classpath="/my/rt.jar": コンパイル時の代替または追加の CLASSPATH を指定します。追加の CLASSPATH が必要となるコンパイラも存在します（Java 2 VM の rt.jar ファイルをインクルードする必要がある Jikes など)。
- executable="jikes" />: 使用する実行可能なコンパイラの名前（Jikes、JVC など)。

<cluster>

このサーバーのクラスタ設定。

属性 :

- id="123" />: サーバーの一意のクラスタ ID。

<execution-order>

起動クラスを実行する順番を定義します。値は整数です。OC4J は 0 以上でロードされます。数字が重複している場合、OC4J がこれらのクラスの順位を選択します。

<global-application>

このサーバーのデフォルト・アプリケーション。これは、オブジェクト可視性に関し、他のアプリケーションの親として機能します。

属性 :

- name="default": アプリケーションを指定します。
- path=".../application.xml" />: デフォルト・アプリケーションの設定が含まれるグローバルな application.xml ファイルへのパスを指定します。application.xml ファイルは、アプリケーションごとに標準の J2EE アプリケーション・ディスクリプタ・ファイルとして存在するもので、このファイルとは異なります。この application.xml は、名前は同じですが、すべての J2EE アプリケーションにグローバル設定を提供するために存在しています。

<global-thread-pool>

この要素を使用して、OC4J プロセスに対するスレッド・プール数について、無制限、1 つ、2 つのいずれかを指定できます。この要素を指定しないと、スレッドが無限に作成される可能性があります。詳細は、2-16 ページの「スレッド・プールの設定」を参照してください。

属性 :

- min: OC4J が同時に実行可能な最小スレッド数。コンテナの起動時に、最小数のスレッドがデフォルトで事前に割り当てられ、スレッド・プールに設定されています。値は整数です。デフォルトは 20 です。設定可能な最小値は 10 です。
- max: OC4J が同時に実行可能な最大スレッド数。最大サイズ未満でかつアイドル・スレッドがない場合には、新しいスレッドが生成されます。新しいスレッドが生成される前にアイドル・スレッドが使用されます。値は整数です。デフォルトは 40 です。
- queue: キューの中に保持できるリクエストの最大数。値は整数です。デフォルトは 80 です。
- keepAlive: 新規リクエストを待機中に、スレッドを生存させておく（アイドルにしておく）時間（単位はミリ秒）。このタイムアウト値は、アイドル・スレッドを維持しておく時間の長さを指定します。タイムアウトに達するとスレッドは破棄されます。最小間隔は 1 分です。時間はミリ秒単位で設定します。スレッドを破棄しないようにするには、このタイムアウトを負数に設定します。

  値は LONG 型で、デフォルトは 600000 ミリ秒です。
- cx-min: OC4J が同時に実行可能な接続スレッドの最小数。値は整数です。デフォルトは 20 です。設定可能な最小値は 10 です。
- cx-max: OC4J が同時に実行可能な接続スレッドの最大数。値は整数です。デフォルトは 40 です。
- cx-queue: キューの中に保持できる接続リクエストの最大数。値は整数です。デフォルトは 80 です。
- cx-keepAlive: 新規リクエストを待機中、接続スレッドを生存させておく（アイドルにしておく）時間（単位はミリ秒）。このタイムアウト値は、アイドル・スレッドを維持しておく時間の長さを指定します。タイムアウトに達するとスレッドは破棄されます。最小間隔は 1 分です。時間はミリ秒単位で設定します。スレッドを破棄しないようにするには、このタイムアウトを負数に設定します。

  値は LONG 型で、デフォルトは 600000 ミリ秒です。
- debug: true の場合、起動時にアプリケーション・サーバーのスレッド・プール情報を出力します。デフォルトは false です。

<global-web-app-config>

属性 :

- path: web-application.xml ファイルが存在する場所へのパス。

  ```
  path=".../web-application.xml" />
  ```

<init-library>

属性 :

- path: 起動クラスと停止クラスが存在する場所へのパス。このパスは、クラスが含まれているディレクトリ、またはクラスがアーカイブされている JAR のディレクトリと JAR ファイル名を示します。ディレクトリまたは JAR ファイルが 2 つ以上ある場合は、それぞれのディレクトリと JAR ファイル名に <init-library> 要素を指定します。

  ```
  <init-library path="../xxx">
  ```

<init-param>

属性 :

- 起動クラスに渡すパラメータのキーと値のペアを定義します。

<javacache-config>

属性 :

- path: javacache.xml ファイルへのパスを指定します。

```
<javacache-config path="../../../javacache/admin/javacache.xml" />
```

<java-compiler>

JSP および EJB のコンパイル用の代替コンパイラ（インプロセス・コンパイラまたはアウトプロセス・コンパイラ）を指定できます。デフォルトのコンパイラは、JDK の bin ディレクトリに含まれているアウトプロセスの javac コンパイラです。

属性 :

- name: 使用するコンパイラの名前を指定します。有効なコンパイル名は、次のとおりです。
  - インプロセス・コンパイラ : modern、classic、javac または ojc
  - アウトプロセス・コンパイラ（フォーク）: modern、javac、ojc または jikes

    この名前の定義は次のとおりです。
    - javac: すべての JDK 用の標準コンパイラ名。
    - classic: JDK 1.1 および 1.2 の標準コンパイラ。
    - modern: JDK 1.3 および 1.4 の標準コンパイラ。
    - jikes: Jikes コンパイラ。
    - ojc: Oracle Java コンパイラ。
- in-process: true の場合、コンパイラはインプロセスで実行されます。false の場合、コンパイラはアウトプロセスで実行されます。ほとんどのコンパイラは、インプロセスとアウトプロセスの両方で実行できます。例外は次のとおりです。
  - classic コンパイラは、アウトプロセスでは実行できません。したがって、in-process 属性は常に true です。
  - jikes コンパイラは、インプロセスでは実行できません。したがって、in-process 属性は常に false です。
- encoding: ソース・ファイルの文字エンコード・タイプ（UTF-8、EUCJIS または SJIS など）を指定します。エンコードをサポートするのは、javac コンパイラのみです。デフォルトは、インストールされている JVM の言語バージョンにより決まります。
- bindir: コンパイラ・ディレクトリへの絶対パスを指定します。javac、modern または classic にこの属性を指定する必要はありません。これらのコンパイラでは JDK の bin ディレクトリが自動的に検索されるためです。

  構文は、オペレーティング・システム・プラットフォーム固有です。
  - Sun Microsystems Solaris の例 : /usr/local/bin/jikes にある jikes を使用する場合は、次のように指定します。

    ```
    name="jikes"
    bindir="/usr/local/bin"
    ```
  - Windows の例 : c:¥jdk1.3.1¥bin¥jikes.exe にある jikes を指定するには、次のように指定します。

    ```
    name="jikes"
    bindir="c:¥¥jdk1.3.1¥¥bin"
    ```

- extdirs: コンパイルの対象として使用される拡張ディレクトリを指定します。デフォルトは JDK 拡張ディレクトリです。コロンで区切って、複数のディレクトリを指定できます。指定されたディレクトリ内の各 JAR アーカイブでクラス・ファイルが検索されます。-Djava.ext.dirs システム・プロパティを変更し、検索対象のディレクトリを指定することができます。jikes コンパイラでは、拡張ディレクトリが属性または -Djava.ext.dirs システム・プロパティに指定されている必要があります。

この要素で代替コンパイラを定義する方法について、4 つの例を次に示します。

```
<java-compiler name="jikes" bindir="C:¥java¥jikes¥bin"
in-process="false" />
<java-compiler name="ojc" bindir="C:¥java¥jdev¥jdev¥bin"
in-process="false"/>
<java-compiler name="classic" in-process="true" />
<java-compiler name="modern" in-process="true" />
```

<jms-config>

属性:

- path: jms.xml ファイルへのパスを指定します。

  path=".../jms.xml"

<log>

<file>

属性:

- path="../log/server.log": ログ・イベントが格納されるファイルの相対パスまたは絶対パスを指定します。

<mail>

ログ・イベントの転送先の電子メール・アドレス。このオプションを使用するには、有効なメール・セッションも指定する必要があります。

属性:

- address="my@mail.address": メール・アドレスを指定します。

<odl>

各 ODL ログ・エントリは、それぞれのログ・ファイルに XML 形式で書き込まれます。ログ・ファイルにはサイズの上限があります。制限に達すると、ログ・ファイルは上書きされます。

ODL ロギングを有効にすると、各メッセージは log*N*.xml という名前のそれぞれのログ・ファイルに入れられます。N は、1 から始まる数字です。最初のログ・メッセージにより、ログ・ファイル log1.xml が開始します。このログ・ファイルが最大サイズに達すると、次のログ・ファイル log2.xml がオープンされ、ロギングを続行します。最後のログ・ファイルがいっぱいになると、最初のログ・ファイル log1.xml が削除され、新しいファイルがオープンされて新しいメッセージが入れられます。このように、ログ・ファイルは常にロールオーバーするため、ディスク領域を侵害することはありません。

属性:

- path: この領域のログ・フォルダのパスとフォルダ名。絶対パスを使用するか、構成 XML ファイルがある場所（通常は、j2ee/home/config ディレクトリ）に対する相対パスを使用できます。これは、XML 構成ファイルが関係する機能に対して、そのログ・ファイルが置かれる場所を示します。たとえば、server.xml ファイルのこの要素を変更して、サーバー・ログ・ファイルが書き込まれる場所を示します。
- max-file-size: 各ログ・ファイルの最大サイズ（KB 単位）。
- max-directory-size: ディレクトリの最大サイズ（KB 単位）。デフォルトのディレクトリ・サイズは 10MB です。

ディレクトリの最大サイズに達するまで、ディレクトリ内に新規ファイルが作成されます。各ログ・ファイルは、属性で指定された最大値以下になります。

<max-http-connections>

特定の Web サイトが 1 つの時点において受け付けることができる同時接続の最大数を定義するために使用します。タグ内部にテキストが存在する場合、制限に達するとリダイレクト URL として使用されます。

属性 :

- max-connections-queue-timeout="10": 最大接続数に達した際に、接続が切断され、サーバーがビジーまたは接続がリダイレクトされる旨のメッセージがクライアントに返されるまでの秒数。デフォルトは 10 秒です。
- socket-backlog: ソケット・レベルで接続を拒否するまでキューに入れられる接続数。デフォルトは 30 です。
- value: 最大接続数。

<metric-collector>

<metric-collector> 要素を指定すると、OC4J は mod_oc4j が使用可能な OC4J インスタンスに受信リクエストを分散してロード・バランシングを行えるように、0 ～ 100（100 を含む）のメトリックを mod_oc4j に送信します。送信されるメトリックの値はすべて相対値です。0 は OC4J インスタンスが非常にビジーであることを意味し、100 は OC4J インスタンスが使用可能（ビジーではない）ことを意味します。メトリックベースのロード・バランシング用に構成した場合、mod_oc4j はより大きな値を持つ OC4J インスタンスに最初にルーティングします。

OC4J から mod_oc4j へ送信されるメトリックは、mod_oc4j に対してメトリックベースのロード・バランシングが指定されており、かつ OC4J が Oracle Application Server 環境で実行されている場合にのみ使用されます。

mod_oc4j にメトリックベースのロード・バランシングを指定し、server.xml に <metric-collector> 要素を指定しない場合、mod_oc4j は OC4J からメトリックが送信されると予想しますが、OC4J はメトリックを送信しません。この場合、mod_oc4j は次の警告メッセージを発します。

```
No run time metrics for oc4j(opmnid=%s) in notification Oc4jSelectMethod is configured
to use run time metrics, please make sure OC4J side is configured accordingly. Default
to 50.
```

この場合、mod_oc4j は各 OC4J プロセスに対して値 50 を使用し、続行します。

同様に、server.xml に <metric-collector> 要素を指定し、mod_oc4j にメトリックベースのロード・バランシングを指定しない場合、OC4J はメトリックを送信しますが、mod_oc4j はメトリックを受信するように構成されていません。この場合、mod_oc4j はメトリックを無視し、構成されている方式が何であろうと、それをロード・バランシングに使用します。ロード・バランシング方式は Oc4jSelectMethod で指定します。Oc4jSelectMethod が指定されていない場合、mod_oc4j はデフォルトのラウンドロビン方式を使用します。

<metric-collector> 要素は、classname 属性を取ります。

classname 属性には、サーバー全体のメトリックを収集および計算するためのインタフェースを定義します。dms-noun ベースのメトリック・コレクタを使用する場合、classname 属性で oracle.oc4j.server.DMSMetricCollector を使用します。DMSMetricCollector インスタンスは、いくつかのパラメータを取ります。パラメータ値の詳細は、『Oracle Application Server 10*g* パフォーマンス・ガイド』を参照してください。

たとえば、次のように入力します。

```
<metric-collector classname="oracle.oc4j.server.DMSMetricCollector">
   <init-param>
     <param-name>
       dms-noun
     </param-name>
     <param-value>
       /oc4j/default/WEBs/processRequest.time
     </param-value>
   </init-param>
```

```
    <init-param>
      <param-name>
        history-proportion
      </param-name>
      <param-value>
        0.2
      </param-value>
    </init-param>
    <init-param>
      <param-name>
        debug
      </param-name>
      <param-value>
        false
      </param-value>
    </init-param>
</metric-collector>
```

<metric-collector> 要素の使用、および mod_oc4j でのメトリックベースのロード・バランシングの使用の詳細は、『Oracle Application Server 10*g* パフォーマンス・ガイド』を参照してください。

<rmi-config>

属性 :

- path: rmi.xml ファイルへのパスを指定します。

  ```
  path="../rmi.xml"
  ```

<sep-config>

このファイルの <sep-config> 要素は、サーバー拡張プロバイダ・プロパティのパス名を指定します。通常は、internal-settings.xml です。

属性 :

- path: サーバー拡張プロバイダ・プロパティのパス。

<sfsb-config>

この要素の enable-passivation 属性を false に設定しないかぎり、ステートフル Session Bean は自動的に非アクティブになります。ステートフル Session Bean の非アクティブ化の詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』の「高度な EJB のトピック」を参照してください。

属性

- enable-passivation: デフォルトは true で、ステートフル Session Bean の非アクティブ化が発生することを意味します。ステートフル Session Bean が非アクティブ化されるような状態にない場合は、この属性を false に設定します。

<shutdown-classes>

停止クラスはユーザーにより定義でき、アンデプロイされた後で、コア・サービスが停止する前に実行されます。

<shutdown-class>

停止クラスは、それぞれ <shutdown-class> 要素内に定義されます。

属性 :

- classname: ユーザー定義の停止クラスのクラス名。

<startup-classes>

起動クラスはユーザーにより定義でき、コア・サービス（JMS、RMI）が起動された後でアプリケーションがデプロイされる前に実行されます。停止クラスは、アンデプロイの後でコア・サービスが停止する前に実行されます。

<startup-class>

起動クラスは、それぞれ <startup-class> 要素内に定義されます。

属性 :

- classname: ユーザー定義の起動クラスのクラス名。
- failure-is-fatal: true に設定され、例外がスローされた場合は、OC4J が例外をログして終了します。false の場合、OC4J は例外をログして実行を続けます。デフォルトは false です。

<transaction-config>

サーバーのトランザクション構成。

属性 :

- timeout="30000": トランザクションがタイムアウトのためにロール・バックされるまで、トランザクションで使用可能な最大時間（ミリ秒）を指定します。デフォルト値は 30000 です。このタイムアウトは、OC4J で開始されるすべてのトランザクションのデフォルト・タイムアウトになります。この値は、動的 API の UserTransaction.setTransactionTimeout(milliseconds) を使用して変更できます。

<web-site>

属性 :

- path: Web サイトを定義する *web-site.xml ファイルへのパス。Web サイトごとに別個の *web-site.xml ファイルを指定する必要があります。この例では、my-web-site.xml ファイルに Web サイトが定義されています。

  path=".../my-web-site.xml"

application.xml ファイルの要素

この項では、J2EE アプリケーション・デプロイメント・ディスクリプタ・ファイルの概要を示します。

<application> 要素の説明

application.xml ファイルのトップレベルの要素は、<application> 要素です。

<application> 内に含まれる要素

<application> 要素の中には、次の要素を構成できます（DTD での順序ではなく、アルファベット順に説明します）。

<alt-dd>path/to/dd</alt-dd>

alt-dd 要素は、特定の J2EE モジュール用のデプロイメント・ディスクリプタ・ファイルのポスト・アセンブリ版を指すオプションの URI を指定します。この URI には、デプロイメント・ディスクリプタ・ファイルを、アプリケーションのルート・ディレクトリからのフルパス名で指定する必要があります。alt-dd を指定しない場合、デプロイ担当者は、コンポーネントごとに必須で指定されているデフォルトの場所とファイル名からデプロイメント・ディスクリプタを読み取る必要があります。

<connector>context</connector>

connector 要素は、リソース・アダプタのアーカイブ・ファイルの URI を、アプリケーション・パッケージのトップレベルに相対的に指定します。

<context-root>thedir/</context-root>

context-root 要素は、Web アプリケーションのコンテキスト・ルートを指定します。

<description>A description.</description>

description 要素は、判読可能なアプリケーションの説明を指定します。description 要素には、アプリケーションのアセンブル担当者がデプロイ担当者に提供する必要のある情報をすべて含めます。

<display-name>The name.</display-name>

display-name 要素は、アプリケーション名を指定します。アプリケーション名は、アプリケーションのアセンブル担当者により付けられ、デプロイ時にデプロイ担当者がアプリケーションを識別するために使用します。

<ejb>pathToEJB.jar</ejb>

ejb 要素は、EJB JAR の URI をアプリケーション・パッケージのトップレベルに相対的に指定します。

<icon>

icon 要素には、GUI ツール内でアプリケーションを表すために使用される大小のイメージがアプリケーションのどの位置にあるかを示す small-icon および large-icon 要素が含まれます。

<java>pathToClient.jar</java>

java 要素は、Java アプリケーションのクライアント・モジュールの URI を、アプリケーション・パッケージのトップレベルに相対的に指定します。

<large-icon>path/to/icon.gif</large-icon>

large-icon 要素には、大きな（32x32 ピクセル）アイコン・イメージを含むファイルのアプリケーション内での位置が含まれます。このイメージは GIF または JPEG 形式で、ファイル名は拡張子「.gif」または「.jpg」で終わる必要があります。

<module>

module 要素は単一の J2EE モジュールを表し、EJB、Java または Web 要素を含みます。この要素はモジュール・タイプを表すとともにモジュール・ファイルへのパスを含みます。またオプションで、デプロイメント・ディスクリプタのポスト・アセンブリ版を指すオプションの URI を指定する alt-dd 要素も含みます。アプリケーション・デプロイメント・ディスクリプタには、アプリケーション・パッケージ内の各 J2EE モジュールごとに module 要素を 1 つ含める必要があります。

<role-name>nameOfRole</role-name>

ロールの名前。

<security-role>

security-role 要素には、アプリケーション全体に適用されるセキュリティ・ロールの定義が含まれます。この定義は、セキュリティ・ロールの説明とセキュリティ・ロールの名前で構成されます。このレベルの説明は、コンポーネント・レベルのセキュリティ・ロール定義内の説明よりも優先され、デプロイ担当者に表示される説明ツールである必要があります。

<small-icon>path/to/icon.gif</small-icon>

small-icon 要素には、小さい（16x16 ピクセル）アイコン・イメージを含むファイルのアプリケーション内での位置が含まれます。このイメージは GIF または JPEG 形式で、ファイル名は拡張子「.gif」または「.jpg」で終わる必要があります。

<web>

web 要素には、Web アプリケーション・モジュールの web-uri および context-root が含まれます。

<web-uri>pathTo.war</web-uri>

web-uri 要素は、Web アプリケーション・ファイルの URI をアプリケーション・パッケージのトップレベルに相対的に指定します。

orion-application.xml ファイルの要素

この項では、OC4J 固有のアプリケーション・デプロイメント・ディスクリプタ・ファイルについて説明します。

<orion-application> 要素の説明

orion-application.xml ファイルのトップレベルの要素は、<orion-application> 要素です。

属性 :

- autocreate-tables: このアプリケーションで、CMP Bean 用のデータベース表を自動的に作成するかどうか。デフォルトは true です。
- autodelete-tables: このアプリケーションで、再デプロイ時に CMP Bean 用の古いデータベース表を自動的に削除するかどうか。デフォルトは false です。
- default-data-source: サーバーのデフォルト以外のデータソースが使用される場合のデフォルトのデータソース。この属性を指定する場合は、このアプリケーションの有効な CMT データソースを指す必要があります。
- deployment-version: この JAR がデプロイされた OC4J のバージョン。現在のバージョンに一致しない場合は、再デプロイされます。これはサーバーの内部的な値なので、編集しないでください。
- treat-zero-as-null: ゼロが主キーを表す場合に、ゼロを NULL として扱うかどうか。デフォルトは false です。

<orion-application> 内に含まれる要素

<orion-application> 要素の中には、次の要素を構成できます（DTD での順序ではなく、アルファベット順に説明します）。

<argument value="theValue" />

クライアントを起動するときに使用される引数。

属性 :

- value: 引数の値。

<arguments>

アプリケーションをインプロセスで開始する場合（auto-start="true"）に、アプリケーション・クライアントの起動時に使用する引数のリスト。

<client-module auto-start="true|false"
deployment-time="073fc2ab513bc3ce" path="myappclient.jar"
user="theUser">

アプリケーションのアプリケーション・クライアント・モジュール。アプリケーション・クライアントは、サーバーと通信する GUI またはコンソール・ベースのスタンドアロン・クライアントです。

属性 :

- auto-start: サーバーの起動時にクライアントを自動的に（インプロセスで）起動するかどうか。デフォルトは false です。
- deployment-time: 最終デプロイ時刻属性。OC4J の内部属性なので、編集しないでください。
- path: アプリケーション・クライアントへのパス（エンタープライズ・アーカイブへの相対パスまたは絶対パス）。
- user: インプロセス（autostart="true"）で実行されるかのようにクライアントを実行するユーザー。auto-start がアクティブになっている場合は、この属性を指定する必要があります。

<commit-coordinator>

2 フェーズ・コミット・エンジンを構成します。

<commit-class class="com.evermind.server.OracleTwoPhaseCommitDriver" />

属性 :

- class: 2 フェーズ・コミット・エンジン用の OracleTwoPhaseCommitDriver クラスを構成します。

<connectors path="./oc4j-connectors.xml" />

属性 :

- path: oc4j-connectors.xml ファイルの名前とパス。<connectors> 要素を指定しない場合、デフォルトのパスは、<oc4j>/j2ee/home/connectors/*rarname*./oc4j-connectors.xml です。

<data-sources path="./data-sources.xml" />

属性 :

- path: パス。

<description>A Short description</description>

このコンポーネントの短い説明。

<ejb-module path="myEjbs.jar" remote="true|false" />

アプリケーションの EJB JAR モジュール。

属性 :

- path: ejb-jar へのパス（エンタープライズ・アーカイブへの相対パスまたは絶対パス）。
- remote: このノードで EJB インスタンスをアクティブにするか、別のサーバーからリモートにインスタンスを参照するかを示す true/false 値。デフォルトは false です。

<file path="../log/server.log" />

イベントのログ先の相対 / 絶対パス。

属性 :

- path: ログ・ファイルへのパス。

<group name="theGroup" />

この security-role-mapping が対象とするグループ。つまり、指定されたグループの全メンバーがこのロールに含まれます。

属性 :

- name: グループの名前。

<jazn provider="XML" location="./jazn-data.xml" />

XML ベースのプロバイダ・タイプを使用するように OracleAS JAAS Provider を構成します。

属性 :

- provider: XML
- location: ファイルへのパス。たとえば ./jazn-data.xml のように指定します。これは、絶対パスでも jazn.xml ファイルへの相対パスでもかまいません。OracleAS JAAS Provider は、最初に jazn.xml ファイルを含むディレクトリで jazn-data.xml を探します。jazn.xml ファイルが構成されている場合はオプションで、それ以外の場合は必須です。
- persistence: 可能な値は、NONE（変更は持続しない）、ALL（変更は常に、次の変更まで持続する）、VM_EXIT（これがデフォルトで、VM 終了後も変更は持続する）です。

- default-realm: レルム名。たとえば、sample_subrealm のように指定します。構成されているレルムが 1 つのみの場合、この属性はオプションです。

<jazn-web-app auth-method="SSO" runas-mode="false" doasprivileged-mode="true" />

JAZNUserManager のフィルタ要素です。

属性 :

- auth-method を SSO（シングル・サインオン）に設定します。このパラメータを設定しないと、デフォルトは NULL になります。
- runas-mode および doasprivileged-mode の設定は、表 B-1 に説明してあります。詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

表 B-1 runas-mode および doasprivileged-mode の設定

runas-mode の設定	doasprivileged-mode の設定	結果
true	true（デフォルト）	chain.doFilter（myrequest,response）をコールする privilegedExceptionAction ブロック内の Subject.doAsPrivileged
true	false	chain.doFilter（myrequest,response）をコールする privilegedExceptionAction ブロック内の Subject.doAs
false（デフォルト）	true	chain.doFilter（myrequest,response）
false	false	chain.doFilter（myrequest,response）

<library path="../lib/" />

このサーバーのライブラリ・パスとして追加するディレクトリまたは JAR/ZIP への相対 / 絶対的なパスまたは URL。起動時にディレクトリがスキャンされ、含める JAR/ZIP ファイルが検索されます。

属性 :

- path: パス。

<log>

ロギング設定。

<odl>

各 ODL ログ・エントリは、それぞれのログ・ファイルに XML 形式で書き込まれます。ログ・ファイルにはサイズの上限があります。制限に達すると、ログ・ファイルは上書きされます。

ODL ロギングを有効にすると、各メッセージは log*N*.xml という名前のそれぞれのログ・ファイルに入れられます。N は、1 から始まる数字です。最初のログ・メッセージにより、ログ・ファイル log1.xml が開始します。このログ・ファイルが最大サイズに達すると、次のログ・ファイル log2.xml がオープンされ、ロギングを続行します。最後のログ・ファイルがいっぱいになると、最初のログ・ファイル log1.xml が削除され、新しいファイルがオープンされて新しいメッセージが入れられます。このように、ログ・ファイルは常にロールオーバーするため、ディスク領域を侵害することはありません。

属性 :

- path: この領域のログ・フォルダのパスとフォルダ名。絶対パスを使用するか、構成 XML ファイルがある場所（通常は、j2ee/home/config ディレクトリ）に対する相対パスを使用できます。これは、XML 構成ファイルが関係する機能に対して、そのログ・ファイルが置かれる場所を示します。たとえば、server.xml ファイルのこの要素を変更して、サーバー・ログ・ファイルが書き込まれる場所を示します。
- max-file-size: 各ログ・ファイルの最大サイズ（KB 単位）。

- max-directory-size: ディレクトリの最大サイズ（KB 単位）。デフォルトのディレクトリ・サイズは 10MB です。

ディレクトリの最大サイズに達するまで、ディレクトリ内に新規ファイルが作成されます。各ログ・ファイルは、属性で指定された最大値以下になります。

<mail address="my@mail.address" />

イベントをログする宛先の電子メール・アドレス。このオプションを使用する場合は、有効な mail-session も指定する必要があります。

属性 :

- address: 電子メール・アドレス。

<mail-session location="mail/TheSession" smtp-host="smtp.server.com">

セッションの SMTP サーバー・ホスト（SMTP を使用している場合）。

属性 :

- location: セッションを格納するネームスペース内の位置。
- smtp-host: セッションの SMTP サーバー・ホスト（SMTP を使用している場合）。

<namespace-access>

RMI クライアント用のネームスペース（ネーミング・コンテキスト）のセキュリティ・ポリシー。

<namespace-resource root="the/path">

特定のセキュリティ設定を持つリソース。

属性 :

- root: このルールが適用されるネームスペース部分のルート。

<password-manager>

隠されたパスワードの参照に使用される UserManager を指定します。省略した場合は、現在の UserManager が認証と認可に使用されます。たとえば、OracleAS JAAS Provider LDAP UserManager を全体的な UserManager として使用し、OracleAS JAAS Provider XML UserManager を隠されたパスワードのチェックに使用することができます。

UserManager を識別するには、次のように、この要素内に <jazn> 要素の定義を指定します。

```
<password-manager>
<jazn ...>
</password-manager>
```

<persistence path="./persistence" />

複数回の再起動にわたってアプリケーションの状態が格納されるファイルの（アプリケーション・ルートに対する）相対パスまたは絶対パスを指定します。

属性 :

- path: 永続ディレクトリへのパス（エンタープライズ・アーカイブへの相対パスまたは絶対パス）。

<principals path="principals.xml" />

属性 :

- path: プリンシパル・ファイルへのパス（エンタープライズ・アーカイブへの相対パスまたは絶対パス）。

<property name="theName" value="theValue" />

名前 / 値ペアの初期化パラメータを含みます。

属性 :

- name: パラメータの名前。
- value: パラメータの値。

<read-access>

読取りアクセス・ポリシー。

<resource-provider>

JMS リソース・プロバイダを定義します。カスタムの <resource-provider> を追加するには、orion-application.xml ファイルに次の項目を追加します。

```
<resource-provider class="providerClassName" name="JNDI name">
  <description> description </description>
  <property name="name" value="value" />
</resource-provider>
```

このコードでユーザーが置き換えられる部分（イタリック体の部分）に、次のように変更を加えます。

- class 属性の値 *providerClassName* を、リソース・プロバイダ・クラスの名前に置き換えます。
- name 属性の値 *JNDI name* を、リソース・プロバイダを識別する名前に置き換えます。この名前は、アプリケーションの JNDI でリソース・プロバイダを "java:comp/resource/name/" として見つけるときに使用されます。
- description 要素の値 *description* を、特定のリソース・プロバイダの説明に置き換えます。
- 対応する属性の値 *name* および *value* を、特定のリソース・プロバイダに対してパラメータとして指定する必要のあるプロパティ要素内の同一の名前に置き換えます。

<security-role-mapping impliesAll="true|false" name="theRole">

ロールの（グループおよびユーザーに対する）ランタイム・マッピング。アセンブリ・ディスクリプタ内の同一名のセキュリティ・ロールにマップします。

属性 :

- impliesAll: このマッピングが全ユーザーを対象としているかどうか。デフォルトは false です。
- name: ロールの名前

<user name="theUser" />

この security-role-mapping が対象とするユーザー。

属性 :

- name: ユーザーの名前。

<user-manager class="com.name.of.TheUserManager" display-name="Friendly UserManager name">

使用するオプションのユーザー・マネージャを指定します。たとえば、ユーザー・マネージャには、com.evermind.sql.DataSourceUserManager や com.evermind.ejb.EJBUserManager などがあります。これらのユーザー・マネージャは、既存のシステムを統合し、Web アプリケーション用のカスタム・ユーザー・マネージャを提供するために使用されます。

属性 :

- class: ユーザー・マネージャの完全修飾クラス名。
- display-name: この UserManager インスタンスの説明的な名前。

<web-module id="myWebApp" path="myWebApp.war" />

アプリケーションの Web アプリケーション・モジュール。各 Web アプリケーションは、任意のサイトおよびサイト上の任意のコンテキスト（たとえば、http://www.myserver.com/myapp/）にインストールできます。

属性 :

- id: Web サイトなどで使用されるときに、この Web アプリケーションを参照するために使用される名前。
- path: Web アプリケーションへのパス（エンタープライズ・アーカイブへの相対パスまたは絶対パス）。

<write-access>

書込みアクセス・ポリシー。

application-client.xml ファイルの要素

この項では、J2EE アプリケーション・クライアント・デプロイメント・ディスクリプタ・ファイルについて説明します。

<application-client> 要素の説明

application-client.xml ファイルのトップレベルの要素は、<application-client> 要素です。

<application-client>

application-client 要素は、アプリケーション・クライアントのデプロイメント・ディスクリプタのルート要素です。アプリケーション・クライアントのデプロイメント・ディスクリプタは、アプリケーション・クライアントにより参照される EJB コンポーネントと外部リソースを記述します。

<application-client> 内に含まれる要素

<application-client> 要素の中には、次の要素を構成できます（DTD での順序ではなく、アルファベット順に説明します）。

<callback-handler>

callback-handler 要素は、アプリケーションにより提供されるクラスの名前を指定します。このクラスは、引数を取らないコントラクタを持ち、javax.security.auth.callback.CallbackHandler インタフェースを実装する必要があります。このクラスはアプリケーション・クライアント・コンテナによりインスタンス化され、そのコンテナがユーザーからの認証情報を収集するために使用されます。

<description>A description.</description>

短い説明。

<display-name>The name.</display-name>

display-name 要素には、ツールによる表示を目的とした短い名前が含まれます。

<ejb-link>EmployeeRecord</ejb-link>

ejb-link 要素は、包含する J2EE アプリケーション・パッケージ内の Enterprise Bean に EJB 参照がリンクされていることを指定するために、ejb-ref 要素内で使用されます。ejb-link 要素の値は、同一 J2EE アプリケーション・パッケージ内の Enterprise Bean の ejb-name にする必要があります。

<ejb-ref>

ejb-ref 要素は、Enterprise Bean のホームへの参照の宣言に使用されます。この宣言は、オプションの説明、参照元アプリケーション・クライアントのコード内で使用されている EJB 参照名、参照される Enterprise Bean の想定タイプ、参照される Enterprise Bean の想定ホームおよびリモート・インタフェース、さらにオプションの ejb-link 情報で構成されます。オプションの ejb-link 要素は、参照される Enterprise Bean を指定するために使用されます。

<ejb-ref-name>ejb/Payroll</ejb-ref-name>

ejb-ref-name 要素には、EJB 参照の名前が含まれます。EJB 参照は、Enterprise Bean の環境内のエントリです。この名前の先頭に「ejb/」を付けることをお薦めします。

<ejb-ref-type>Entity/Session</ejb-ref-type>

ejb-ref-type 要素には、参照される Enterprise Bean の想定タイプが含まれます。ejb-ref-type 要素は、Entity か Session のいずれかに指定する必要があります。

<env-entry>

env-entry 要素には、Enterprise JavaBean の環境エントリの宣言が含まれます。この宣言は、オプションの説明、環境エントリの名前およびオプションの値で構成されます。

<env-entry-name>minAmount</env-entry-name>

env-entry-name 要素には、Enterprise JavaBean の環境エントリの名前が含まれます。

<env-entry-type>java.lang.String</env-entry-type>

env-entry-type 要素には、Enterprise Bean のコードが想定する環境エントリ値の完全修飾 Java タイプが含まれます。env-entry-type の有効値は、java.lang.Boolean、java.lang.String、java.lang.Integer、java.lang.Double、java.lang.Byte、java.lang.Short、java.lang.Long および java.lang.Float です。

<env-entry-value>100.00</env-entry-value>

env-entry-value 要素には、Enterprise JavaBean の環境エントリの値が含まれます。

<home>com.aardvark.payroll.PayrollHome</home>

home 要素には、Enterprise JavaBean のホーム・インタフェースの完全修飾名が含まれます。

<icon>

icon 要素には、GUI ツール内でアプリケーションを表すために使用される、GIF または JPEG 形式の小さいアイコン・イメージと大きいアイコン・イメージの URI を指定する、small-icon および large-icon 要素が含まれます。

<large-icon>lib/images/employee-service-icon32x32.jpg</large-icon>

large-icon 要素には、大きな（32x32 ピクセル）アイコン・イメージを含むファイルの名前が含まれます。ファイル名は、アプリケーション・クライアント JAR ファイル内の相対パスです。このイメージは JPEG または GIF 形式で、ファイル名はそれぞれ「.jpg」または「.gif」で終わる必要があります。このアイコンはツールで使用できます。

<remote>com.wombat.empl.EmployeeService</remote>

remote 要素には、Enterprise JavaBean のリモート・インタフェースの完全修飾名が含まれます。

<res-auth>Application/Container</res-auth>

res-auth 要素は、Enterprise JavaBean コードがリソース・マネージャにプログラムでサインオンするか、Bean のかわりにコンテナがリソース・マネージャにサインオンするかを指定します。後者の場合、コンテナはデプロイ担当者が指定する情報を使用します。この要素の値は、Application か Container のいずれかに指定する必要があります。

<resource-env-ref>

resource-env-ref 要素には、アプリケーションの環境内のリソースに関連付けられている管理オブジェクトへのアプリケーション参照の宣言が含まれます。この宣言は、オプションの説明、リソース環境参照名、アプリケーション・コードが想定するリソース環境参照の指定で構成されます。

<resource-env-ref-name>

resource-env-ref-name 要素は、アプリケーション・コード内で使用されるリソース環境エントリ名を指定します。

<resource-env-ref-type>

resource-env-ref-type 要素は、リソース環境参照のタイプを指定します。

<resource-ref>

resource-ref 要素には、外部リソースへの Enterprise JavaBean の参照の宣言が含まれます。この宣言は、オプションの説明、リソース・ファクトリ参照名、Enterprise Bean コードが想定するリソース・ファクトリ・タイプ、認証のタイプ（Bean または Container）で構成されます。

<res-ref-name>name</res-ref-name>

res-ref-name 要素は、リソース・ファクトリ参照の名前を指定します。

<res-sharing-scope>Shareable</res-sharing-scope>

res-sharing-scope 要素は、特定のリソース・マネージャ接続ファクトリ参照を介して取得された接続が共有可能かどうかを指定します。この要素を指定する場合、値は、Shareable か Unshareable のいずれかに指定する必要があります。デフォルト値は Shareable です。

<res-type>javax.sql.DataSource</res-type>

res-type 要素は、データソースのタイプを指定します。このタイプは、データソースによる実装が想定される Java インタフェース（またはクラス）により指定されます。

<small-icon>lib/images/employee-service-icon16x16.jpg
</small-icon>

small-icon 要素には、小さい（16x16 ピクセル）アイコン・イメージを含むファイルの名前が含まれます。ファイル名は、アプリケーション・クライアント JAR ファイル内の相対パスです。このイメージは JPEG または GIF 形式で、ファイル名はそれぞれ「.jpg」または「.gif」で終わる必要があります。このアイコンはツールで使用できます。

orion-application-client.xml ファイルの要素

この項では、OC4J 固有のアプリケーション・クライアント・デプロイメント・ディスクリプタ・ファイルの概要を示します。

<orion-application-client> 要素の説明

orion-application-client.xml ファイルのトップレベルの要素は、<orion-application-client> 要素です。

<orion-application-client>

orion-application-client.xml ファイルには、J2EE アプリケーション・クライアントのデプロイ時の情報が含まれます。この情報は、application-client.xml にあるアプリケーション・クライアント・アセンブリ情報を補完するものです。

<orion-application-client> 内に含まれる要素

<orion-application-client> 要素の中には、次の要素を構成できます（DTD での順序ではなく、アルファベット順に説明します）。

<context-attribute name="name" value="value" />

コンテキストに送信される属性。JNDI で必須の属性は、コンテキスト・ファクトリ実装のクラス名である 'java.naming.factory.initial' のみです。

属性 :

- name: 属性の名前。
- value: 属性の値。

<ejb-ref-mapping location="ejb/Payroll" name="ejb/Payroll" />

ejb-ref 要素は、別の Enterprise Bean のホームへの参照の宣言に使用されます。ejb-ref-mapping 要素が、デプロイ時にこれを JNDI ロケーションに結び付けます。

属性 :

- location: EJB ホームを参照する JNDI ロケーション。
- name: ejb-ref の名前。application-client.xml 内の ejb-ref の名前と同じです。

<env-entry-mapping name="theName">deploymentValue</env-entry-mapping>

アセンブリ・ディスクリプタ内の env-entry の値をオーバーライドします。EAR（アセンブリ）がデプロイ固有の値の影響を受けないようにするために使用されます。ボディがその値です。

属性 :

- name: コンテキスト・パラメータの名前。

<lookup-context location="foreign/resource/location">

リソースを取り出すために使用されるオプションの javax.naming.Context 実装の指定です。これは、サード・パーティ製モジュール（たとえば、サード・パーティ製 JMS サーバーなど）と組み合せるときに役立ちます。リソース・ベンダーにより提供されるコンテキスト実装を使用するか、それがない場合は、ベンダー・ソフトウェアとネゴシエーションする実装を作成します。

属性 :

- location: リソースを取り出すときに外部コンテキスト内で検索する名前。

<resource-env-ref-mapping location="jdbc/TheDS" >

resource-env-ref 要素は、外部リソース（データソース、JMS キュー、メール・セッションなど）への参照の宣言に使用されます。resource-env-ref-mapping 要素が、デプロイ時にこの要素を JNDI ロケーションに結び付けます。

属性 :

- location: リソースのバインド先の JNDI ロケーション。

<resource-ref-mapping location="jdbc/TheDS" name="jdbc/TheDSVar">

resource-ref 要素は、外部リソース（データソース、JMS キュー、メール・セッションなど）への参照の宣言に使用されます。resource-ref-mapping 要素が、デプロイ時にこれを JNDI ロケーションに結び付けます。

属性 :

- location: リソース・ホームを参照する JNDI ロケーション。
- name: resource-ref の名前。application-client.xml 内の resource-ref の名前と同じです。

スタンドアロン OC4J のコマンドライン・オプションおよびプロパティ

OC4J の起動には oc4j.jar を使用します。OC4J の管理には admin.jar ツールを使用します。次の項では、各 JAR のオプションについて説明します。

- OC4J サーバー JAR のオプション
- OC4J 管理 JAR のオプション

OC4J サーバー JAR のオプション

oc4j.jar のコマンドライン・オプションにより、OC4J の起動、停止およびインストールが行えます。

表 B-2 に、oc4j.jar のすべてのコマンドライン・オプションを示します。

表 B-2 OC4J のコマンドライン・オプション

コマンドライン・オプション	説明
-install	サーバーをインストールし、admin アカウントをアクティブにします。オペレーティング・システムの改行に合せてテキスト・ファイルを再作成します。このオプションは最初に一度だけ使用してください。
-quiet	標準出力を抑制します。
-config	server.xml ファイルの位置を指定します。
-out [file]	標準出力のルーティング先ファイルを指定します。このファイルには、System.out に出力されるメッセージ、およびサーブレットのロギング・インタフェースを通じて出力に送られるメッセージが格納されます。指定しない場合は、すべての出力が標準出力に書き込まれます。 stdout ファイルの管理のために設定するその他のシステム・プロパティについては、B-35 ページの表 B-9「stdout/stderr アーカイブ管理プロパティ」を参照してください。
-err [file]	標準エラーのルーティング先ファイルを指定します。このファイルには、System.err に出力されるメッセージが格納されます。指定しない場合は、すべてのエラーが標準エラーに書き込まれます。 stderr ファイルの管理のために設定するその他のシステム・プロパティについては、B-35 ページの表 B-9「stdout/stderr アーカイブ管理プロパティ」を参照してください。
-verbosity	メッセージ出力の冗長性レベルを設定する 1 ～ 10 の整数を定義します。たとえば、-verbosity 10 と指定します。
-monitorResourceThreads	スレッド・リソースのバックアップ・デバッグを使用可能にします。これを使用可能にするのは、コードの重要な部分でスレッドがスタックすることに関連する問題が発生した場合のみです。
-userThreads	ユーザーが作成したスレッドからのコンテキスト検索サポートを有効にします。
-version	バージョンを出力して終了します。
-? -help	ヘルプ・メッセージを出力します。

OC4J 管理 JAR のオプション

admin.jar コマンドライン・ツールでは、コマンドラインを使用して、client-admin コンソールからスタンドアロン OC4J を管理できます。

構文は次のとおりです。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password options
```

admin.jar コマンドライン・ツールのオプションは、次の 4 つの分野を網羅しています。

- OC4J の一般的な管理（表 B-3）。
- アプリケーションのデプロイ（表 B-4）。
- Web サイトの管理（表 B-5）。
- データソースの管理（表 B-6）。

OC4J の一般的な管理

表 B-3 に、OC4J の一般的な管理に使用する admin.jar オプションを示します。たとえば、次のコマンドでは OC4J サーバーが停止します。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -shutdown
```

表 B-3 OC4J 管理用のオプション

オプション	説明
-shutdown [ordinary | force] [reason]	OC4J サーバーを停止します。デフォルトは ordinary です。ordinary を指定すると、各スレッドが正常に終了します。force を指定すると、すべてのスレッドがただちに終了します。reason は、終了時にログに記録される文字列です。
-restart [reason]	OC4J サーバーを再起動します。コンテナが oc4j.jar を使用して起動されている必要があります。reason は、再起動時にログに記録される文字列です。

アプリケーションのデプロイ

表 B-4 に、OC4J アプリケーションの管理に使用する admin.jar オプションを示します。たとえば、次のコマンド構造はアプリケーションのデプロイに使用します。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -deploy -file path/filename
    -deploymentName app_name -targetPath deploy_dir
```

次のコマンド構造は、Web アプリケーションのバインドに使用します。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -bindWebApp app_name web_app_name
                                web_site_name context_root
```

表 B-4 アプリケーション・デプロイ用のオプション

オプション	説明
`-deploy`	アプリケーションをデプロイ（再デプロイ）します。次のサブスイッチにアプリケーション情報を指定します。 `-file` *path/filename*: 必須。デプロイする EAR ファイルのパスおよびファイル名です。 `-deploymentName` *app_name*: 必須。ユーザー定義のアプリケーション・デプロイ名です。この名前は、OC4J でのアプリケーションの識別に使用します。アプリケーションを再デプロイする場合にも指定します。 `-targetPath` *deploy_dir*: オプション。アーカイブのデプロイ先となるサーバー・ノード上のパス。デフォルトは、`applications/` ディレクトリです。デプロイ対象の EAR ファイルをコピーするディレクトリのターゲット・パスを指定することをお薦めします。 `-targetPath` を指定しない場合、EAR ファイルは `applications/` ディレクトリにコピーされます。OC4J は、EAR ファイルの名前を常に一意に保ちます。そのため、EAR ファイルを再デプロイすると、別のアプリケーションの EAR ファイルが上書きされないように、名前の先頭にアンダースコア文字（_）を付加してファイル名を変更します。その後デプロイを行うたびに、アンダースコア文字が 1 つずつ EAR ファイルに付加されます。ただし、同じアプリケーションの場合は、デプロイのたびに `applications/` ディレクトリに別々の EAR ファイルが格納されます。ターゲット・パスを指定すれば、このような問題は発生しません。 `-parent` *parent_appname*: オプション。このアプリケーションの親アプリケーションです。デプロイ時に、子アプリケーション内のメソッドが親アプリケーション内のメソッドを起動します。これは、ある JAR のメソッドが、別の JAR にデプロイされている EJB を参照できるようにするための手段です。JAR ファイルのユーザーがサービス・アプリケーションを親として宣言している場合に、1 つの JAR ファイル内のすべてのサービス EJB をデプロイする際に便利です。デフォルトはグローバル・アプリケーションです。 `-deploymentDirectory` *path*: オプション。指定しない場合、アプリケーションは `application-deployments/` ディレクトリにデプロイされます。アプリケーションのデプロイ先を変更するには、このオプションでパスを指定します。文字列 "[NONE]" を指定すると、アプリケーションがデプロイされるたびにデプロイ構成が必ず EAR ファイルから読み取られます。
`-bindWebApp` *app_name web_app_name web_site_name context_root*	指定したサイトまたはルートに Web アプリケーションをバインドします。 ■ *app_name* はアプリケーション名で、`-deploy` オプションの `-deploymentName` に使用されるのと同じ名前です。また、`server.xml` ファイルの `<application name=`*app_name*`>` 属性に保存される名前とも同じです。 ■ *web_app_name* は、EAR ファイルに格納されている WAR ファイルの名前から `.WAR` 拡張子を除いたものです。 ■ *web_site_name* は、Web アプリケーションのバインド先となる Web サイトを示す *name*`-web-site.xml` ファイルの名前です。これは、Web アプリケーション定義を受け取るファイルです。 ■ *context_root* は、Web モジュールのルート・コンテキストです。 このオプションでは、*web_site_name* 変数で指定された OC4J *name*`-web-site.xml` 構成ファイルに、エントリが作成されます。

表 B-4 アプリケーション・デプロイ用のオプション（続き）

オプション	説明
-updateConfig	check-for-updates を false に設定してある場合、XML ファイルの変更は自動更新されません。このフラグを指定して、XML ファイルに新たに加えた変更をすべてアップロードする必要があります。
-undeploy *app_name*	デプロイ済の J2EE アプリケーションを OC4J Web サーバーから削除します。*app_name* は、-deploymentName サブスイッチで指定した名前です。その結果、次の処理が行われます。 ■ OC4J ランタイムと server.xml ファイルからアプリケーションが削除されます。 ■ すべてのアプリケーションの Web モジュールのバインドが、すべてのバインド先 Web サイトから削除されます。 ■ applications と application-deployments の両方のディレクトリからアプリケーション・ファイルが削除されます。 -keepFiles: アプリケーション・ファイルが削除されないようにするためのオプションのサブスイッチ。ただし、ランタイムからアプリケーションが削除され、Web モジュールのバインドが解除されます。
-deploymentDirectory "[NONE]"	このフラグを "[NONE]" に指定した場合、application-deployments ディレクトリにある前回のデプロイのデプロイメント・ディスクリプタではなく、現在のデプロイの orion-ejb-jar.xml デプロイメント・ディスクリプタが使用されます。
-iiopClientJar	EJB を変換して RMI/IIOP を使用し、EJB コンテナをまたがった EJB の相互起動を可能にすることができます。詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』の「RMI/IIOP」を参照してください。

Web サイトの追加

-site オプションを使用すると、Web サイト構成を XML ファイルに追加できます。表 B-5 に、admin.jar コマンドライン・ツールの -site オプションのサブスイッチをすべて示します。

たとえば、次のコマンド構造では新規 Web サイトがインストールされます。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -site -add -host hostname -port portnumber
      -display-name name -virtual-hosts virtual_host
```

表 B-5 Web サイト管理用のオプション

-site オプション	説明
`-site -add`	新規 Web サイトをインストールします。次のサブスイッチに情報を指定します。
	`-host` *hostname*: Web サイトが存在するホスト。
	`-port` *portnum*: Web サイトのポート。
	`-display-name` *name*: Web サイトの名前。
	`-virtual-hosts` *virtual_hosts*: Web サイトの仮想ホスト。
	`-secure true|false`: Web サイトがセキュアな場合の値は `true`、セキュアでない場合の値は `false` です。
	`-factory` *factory_name*: Java Secure Socket Extension（JSSE）を使用しない場合の `SSLServerSocketFactory` クラスの名前。JSSE では、他のセキュリティ・プロバイダが実装できるプロバイダ・インタフェースが定義されています。Sun 社は、`com.sun.net.ssl.internal.ssl.Provider` で独自の実装を提供しています。
	`-keystore` *keystore*: キーストアの相対パスまたは絶対パス。
	`-storepass` *password*: キーストアのパスワード。
	`-provider` *provider*: JSSE を使用する場合に使用されるプロバイダ。デフォルトは `com.sun.net.ssl.internal.ssl.Provider` です。
	`-needs-client-auth true|false`: `true` に設定した場合、J2EE Web サイトにアクセスするクライアントはデジタル証明によって自身を識別する必要があります。`false` に設定した場合、クライアントがデジタル証明によって自身を識別する必要はありません。デフォルトは `false` です。
`-site -remove`	既存の Web サイトを削除します。この Web サイトのホストおよびポートを次のサブスイッチに指定します。
	`-host` *hostname*: 削除する Web サイトのホスト。
	`-port` *portnum*: 削除する Web サイトのポート。
`-site -test`	既存の Web サイトをテストします。テストする Web サイトのホストおよびポートを次のサブスイッチに指定します。
	`-host` *hostname*: テストする Web サイトのホスト。
	`-port` *portnum*: テストする Web サイトのポート。
`-site -list`	既存の Web サイトをすべてリスト表示します。

表 B-5 Web サイト管理用のオプション（続き）

-site オプション	説明
`-site -update`	既存の Web サイトを更新します。次のサブスイッチに情報を指定します。
	`-oldHost` *hostname*: Web サイトの古いホスト。Web サイトのホストおよびポートは、old および new のサブスイッチで変更できます。
	`-oldPort` *portnum*: Web サイトの古いポート。
	`-newHost` *hostname*: Web サイトの新しいホスト。
	`-newPort` *portnum*: Web サイトの新しいポート。
	`-display-name` *name*: Web サイトの新しい表示名。
	`-virtual-hosts` *vhosts*: Web サイトの新しい仮想ホスト。
	`-secure true|false`: `true` に設定した場合、Web サイトはセキュアです。`false` に設定した場合、Web サイトはセキュアではありません。デフォルトは `false` です。
	`-factory` *classname*: JSSE を使用しない場合の `SSLServerSocketFactory` クラスの新しい名前。
	`-keystore` *path*: キーストアの新しい相対パスまたは絶対パス。
	`-storepass` *password*: キーストアの新しいパスワード。
	`-provider` *provider*: JSSE を使用しない場合に使用される新しいプロバイダ。
	`-needs-client-auth true|false`: `true` に設定した場合、J2EE Web サイトにアクセスするクライアントはデジタル証明によって自身を識別する必要があります。`false` に設定した場合、クライアントがデジタル証明によって自身を識別する必要はありません。デフォルトは `false` です。

データソースおよびアプリケーションのオプション

表 B-6 に、`admin.jar` コマンドライン・ツールの `-application` オプションのサブスイッチを示します。`-application` は、サブスイッチ・コマンドの前にアプリケーションの名前を取ります。この *name* は次のいずれかです。

- 元々 `default` としてインストールされているグローバル・アプリケーションの名前。`server.xml` ファイルで `<global-application>` 要素の `name` 属性に指定されています。
- `server.xml` ファイルの `<application>` 要素に定義されているアプリケーションの名前。

この名前は文字列ですが、引用符で囲まないでください。たとえば、次のコマンドでは定義されているデータソース・オブジェクトがすべてリスト表示されます。

```
java -jar admin.jar ormi://oc4j_host:oc4j_ormi_port admin_id
    admin_password -application default -listDataSource
```

表 B-6 アプリケーションおよびデータソース管理用のオプション

-application オプション	説明
`-application` *name* `-restart`	アプリケーションを再起動します。オート・デプロイが有効になっている場合に、ファイルが変更されていれば、オート・デプロイがトリガーされます。
`-application` *name* `-addUser` *username password*	ユーザーをセキュリティ・ファイル（`principals.xml`）に追加します。
`-application` *name* `-dataSourceInfo`	インストールされている `DataSource` オブジェクトについての動的使用情報を取得します。

表 B-6 アプリケーションおよびデータソース管理用のオプション（続き）

-application オプション	説明
`-application` *name* `-listDataSource`	インストールされている各 DataSource オブジェクトについての静的構成情報を取得します。
`-application` *name* `-testDataSource`	既存の DataSource をテストします。次のサブスイッチに情報を指定します。
	`-location` *location*: DataSource のネームスペース位置。たとえば、jdbc/DefaultDS などです。必須。
	`-username` *username*: パスワードとともにログインに使用するユーザー名。オプション。
	`-password` *password*: ログインに使用するパスワード。オプション。
`-application` *name* `-installDataSource`	新規 DataSource をインストールします。次のサブスイッチに情報を指定します。
	`-jar` *JARfile*: サーバーのライブラリに追加されるドライバを含む JAR ファイル。
	`-url` *URL*: JDBC データベースの URL。
	`-location` *JNDIlocation*: RAW ソースのネームスペース位置。たとえば、jdbc/DefaultPooledDS などです。必須。
	`-pooledLocation` *JNDIlocation*: プールされたソースのネームスペース位置。たとえば、jdbc/DefaultPooledDS などです。
	`-xaLocation` *JNDIlocation*: XA ソースのネームスペース位置。たとえば、jdbc/xa/DefaultXADS などです。-ejbLocation を指定した場合は必須です。
	`-ejbLocation` *JNDIlocation*: コンテナ管理のトランザクション・データソースのネームスペース位置。これは、グローバル JTA トランザクションを実行できる唯一のデータソースです。たとえば、jdbc/DefaultDS などです。
	`-username` *username*: ログインに使用するユーザー名。
	`-password` *password*: ログインに使用するパスワード。
	`-connectionDriver` *driverClass*: JDBC データベースのドライバ・クラス。
	`-classname` *DSclass*: com.evermind.sql.DriverManagerDataSource などのデータソースのクラス名。必須。
	`-sourceLocation` *jndiDS*: この専用データソースの基礎となるデータソース。
	`-xaSourceLocation` *jndiXADS*: この専用データソースの基礎となる XA データソース。
`-application` *name* `-removeDataSource`	既存の DataSource を削除します。次のサブスイッチに情報を指定します。
	`-location` *JNDIlocation*: DataSource のネームスペース位置。たとえば、jdbc/DefaultDS などです。必須。

表 B-6 アプリケーションおよびデータソース管理用のオプション（続き）

-application オプション	説明
`-application` *name* `-updateDataSource`	既存の `DataSource` を更新します。次のサブスイッチに情報を指定します。 `-oldLocation` *JNDIlocation*: `DataSource` の古いネームスペース位置。たとえば、`jdbc/DefaultDS` などです。必須。 `-newLocation` *JNDIlocation*: `DataSource` の新しいネームスペース位置。たとえば、`jdbc/DefaultDS` などです。 `-jar` *JAR*: サーバーのライブラリに追加されるドライバを含む JAR ファイル。 `-url` *URL*: JDBC データベースの URL。 `-pooledLocation` *JNDIlocation*: プールされたソースのネームスペース位置。たとえば、`jdbc/DefaultPooledDS` などです。 `-xaLocation` *JNDIlocation*: XA `DataSource` のネームスペース位置。たとえば、`jdbc/xa/DefaultXADS` などです。`-ejbLocation` を指定した場合は必須です。 `-ejbLocation` *JNDIlocation*: コンテナ管理のトランザクション・データソースのネームスペース位置。これは、グローバル JTA トランザクションを実行できる唯一のデータソースです。たとえば、`jdbc/DefaultDS` などです。 `-username` *username*: ログインに使用するユーザー名。 `-password` *password*: ログインに使用するパスワード。 `-connectionDriver` *driverClass*: JDBC データベースのドライバ・クラス。たとえば、`com.mydb.Driver` などです。 `-className` *dsClass*: データソースのクラス名。たとえば、`com.evermind.sql.DriverManagerDataSource` などです。 `-sourceLocation` *jndiDS*: この専用データソースの基礎となるデータソース。 `-xaSourceLocation` *jndiXADS*: この専用データソースの基礎となる XA データソース。

OC4J システム・プロパティ

起動前に OC4J コマンドラインでシステム・プロパティを設定できます。OC4J が実行中の場合は、これらが有効になるようにインスタンスを再起動する必要があります。システム・プロパティは、すべて先頭に `-D` が付きます。たとえば、`-DGenerateIIOP` などです。

- 表 B-7 は、一般的なシステム・プロパティを示します。
- 表 B-8 は、デバッグ用プロパティを示します。

表 B-7 OC4J の一般的な -D システム・プロパティ

-D オプション	説明
java.home	環境変数 JAVA_HOME を設定します。
java.ext.dirs	コンパイル時にクラスが検索される外部ディレクトリを設定します。
java.io.tmpdir= *new_tmp_dir*	デフォルトは /tmp/var です。デプロイ・ウィザードの一時ディレクトリを変更するためのオプションです。 デプロイ・ウィザードは、デプロイ処理時に情報を格納するために、一時ディレクトリのスワップ領域を 20MB 使用しています。完了すると、追加ファイルの一時ディレクトリをデプロイ・ウィザードがクリーン・アップします。ただし、ウィザードが中断すると、一時ディレクトリをクリーン・アップする時間または機会がありません。したがって、追加されたデプロイ・ファイルをユーザー自身がこのディレクトリからクリーン・アップする必要があります。クリーン・アップしないと、このディレクトリが満杯になり、今後デプロイを実行できなくなる場合があります。Out of Memory エラーが出力される場合は、一時ディレクトリの使用可能領域をチェックしてください。
GenerateIIOP= true/false	デフォルトは false です。true の場合、IIOP スタブの生成を有効化します。
KeepIIOPCode= true/false	デフォルトは false です。true の場合、生成された IIOP スタブ / タイ・コードを保持します。
oracle.arraylist.deepCopy= true/false	true の場合は、配列リストのクローンを作成中にディープ・コピーが実行されます。false の場合は、配列リストにシャロー・コピーが実行されます。デフォルトは true です。
dedicated.rmicontext= true/false	デフォルトは false です。これにより、すでに使用されなくなった dedicated.connection 設定が置き換えられます。同一プロセス内の複数のクライアントが InitialContext を取り出すと、OC4J はキャッシュされているコンテキストを返します。したがって、各クライアントは、そのプロセスに割り当てられている同じ InitialContext を受け取ります。結果的にサーバーのロード・バランシングにつながるサーバー参照は、クライアントが独自の InitialContext を取り出すときにのみ発生します。 dedicated.rmicontext=true を設定すると、各クライアントは共有コンテキストではなくそのクライアント独自の InitialContext を受け取ります。各クライアントに独自の InitialContext がある場合、クライアントのロード・バランシングが可能です。 このパラメータはクライアント用です。JNDI プロパティで設定することもできます。
oracle.mdb.fastUndeploy=<int>	oracle.mdb.fastUndeploy システム・プロパティを使用すると、Windows 環境で MDB を実行している場合、または Windows 環境でバックエンド・データベースが稼働している場合に、OC4J をクリーンに停止できます。通常、MDB を使用する場合、MDB は受信メッセージを待機する受信状態でブロックされます。ただし、Windows 環境で MDB が待機状態のときに OC4J を停止すると、OC4J インスタンスは停止されず、MDB がブロックされているため、アプリケーションはアンデプロイされません。oracle.mdb.fastUndeploy システム・プロパティを設定することで、この環境での MDB の動作を変更できます。このプロパティを整数に設定すると、MDB が受信メッセージの処理中ではなく待機状態の場合、OC4J コンテナはデータベースに移動し (データベース・ラウンドトリップが必要)、セッションが停止しているかどうかを確認するためにポーリングを行います。この整数は、システムがデータベースのポーリングを待機する秒数を示します。これは、パフォーマンスの面ではコストが高くなる可能性があります。このプロパティを 60（秒）に設定すると、OC4J は 60 秒ごとにデータベースをチェックします。このプロパティを設定せずに、[Ctrl] キーを押しながら [C] キーを押して OC4J を停止しようとすると、OC4J プロセスは少なくとも 2.5 時間ハングします。

表 B-7 OC4J の一般的な -D システム・プロパティ（続き）

-D オプション	説明
`oracle.dms.sensors=[none, normal, heavy, all]`	Oracle Application Server 組込みのパフォーマンス・メトリックの値を、none（オフ）、normal（中程度のメトリック）、heavy（大きなメトリック）または all（可能なすべてのメトリック）に設定できます。デフォルトは normal です。このパラメータは OC4J サーバーで設定する必要があります。これらのパフォーマンス・メトリックをオンにするための以前のメソッドである `oracle.dms.gate=true/false` は、`oracle.dms.sensors` 変数で置き換えられました。ただし、`oracle.dms.gate` を使用している場合、この変数を false に設定すると、`oracle.dms.sensors=none` の設定と同じ意味になります。
`associateUsingThirdTable= true/false`	Entity Bean におけるコンテナ管理の関連性の場合、関連性の管理に第 3 のデータベース表が使用されるかどうかを指定できます。第 3 の関連表を必要としない場合は、false に設定します。デフォルトは false です。詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』の「エンティティ関連（E-R）のマッピング」の章を参照してください。
`DefineColumnType= true/false`	デフォルトは false です。9.2 より前の Oracle JDBC ドライバを使用している場合は、true に設定してください。これらのドライバの場合、この変数を true に設定することにより、Oracle JDBC ドライバに対して Select を実行する場合のラウンドトリップを回避できます。このパラメータは OC4J サーバーで設定する必要があります。 このオプションの値を変更して OC4J を再起動すると、この変更の後にデプロイされるアプリケーションに対してのみ有効になります。変更前にデプロイされたアプリケーションには影響はありません。 true に設定すると、`DefineColumnType` 拡張により、通常は表の記述に必要なデータベース・ラウンドトリップが節約されます。Oracle JDBC ドライバで問合せを実行した場合、結果セットの列で使用する型を判別するために、最初にデータベースへのラウンドトリップを使用します。次に、JDBC は問合せからのデータを受け取ると、データを必要に応じて変換し、結果セットに移入します。`DefineColumnType` 拡張を true に設定して問合せの列の型を指定すると、Oracle データベースへの最初のラウンドトリップが回避されます。そのように最適化されているサーバーは、必要な型変換を実行します。

表 B-8 デバッグ用の -D システム・プロパティ

-D デバッグ・システム・プロパティ	説明
`KeepWrapperCode`	デフォルトは false です。true の場合、生成されたラッパー・コードを保持してデバッグします。
`DBEntityHomeDebug`	デフォルトは false です。true の場合、Entity Bean ホーム・インタフェースのデバッグ・メッセージを表示します。
`DBEntityObjectDebug`	デフォルトは false です。true の場合、Entity Bean オブジェクトのデバッグ・メッセージを表示します。
`DBEntityWrapperDebug`	デフォルトは false です。true の場合、Entity Bean プールのデバッグ・メッセージを表示します。
`iiop.runtime.debug`	デフォルトは false です。true の場合、IIOP デバッグ・メッセージを出力します。
`NativeJDBCDebug`	デフォルトは false です。ネイティブ JDBC のデバッグ・メッセージです。
`http.cluster.debug`	デフォルトは false です。HTTP クラスタリングのデバッグ・メッセージです。
`http.request.debug`	デフォルトは false です。true の場合は、各 HTTP リクエストに関する情報を標準出力に出力します。

表 B-8 デバッグ用の -D システム・プロパティ（続き）

-D デバッグ・システム・プロパティ	説明
`http.redirect.debug`	デフォルトは false です。true の場合は、各 HTTP リダイレクトに関する情報を標準出力に出力します。
`http.method.trace.allow`	デフォルトは false です。true の場合は、`trace` HTTP メソッドをオンにします。
`http.session.debug`	デフォルトは false です。true の場合は、HTTP セッションのイベントに関する情報を提供します。
`http.error.debug`	デフォルトは false です。true の場合は、すべての HTTP エラーを出力します。
`http.virtualdirectory.debug`	デフォルトは false です。true の場合は、設定された仮想ディレクトリ・マッピングを起動時に出力します。
`debug.http.contentLength`	デフォルトは false です。true の場合、明示的なコンテンツ長コールのみでなく追加の sendError 情報も出力します。
`ejb.cluster.debug`	デフォルトは false です。EJB クラスタリングのデバッグ・メッセージです。
`cluster.debug`	デフォルトは false です。クラスタリングのデバッグ・メッセージです。
`jms.debug`	デフォルトは false です。JMS のデバッグ・メッセージです。
`multicast.debug`	デフォルトは false です。マルチキャストのデバッグ・メッセージです。
`rmi.debug`	デフォルトは false です。RMI のデバッグ・メッセージです。
`transaction.debug`	デフォルトは false です。true の場合は、JTA イベントのデバッグ・メッセージを出力します。
`rmi.verbose`	デフォルトは false です。RMI の冗長な情報です。
`datasource.verbose`	デフォルトは false です。true の場合は、データソースの作成、データソースを使用する接続、プールに解放された接続などに関する冗長な情報を提供します。
`jdbc.debug`	デフォルトは false です。true の場合は、JDBC コールが行われたときに非常に冗長な情報を提供します。
`ws.debug`	デフォルトは false です。true の場合は、OracleAS Web Services のデバッグをオンにします。
`javax.net.debug=[ssl|all]`	ssl の場合は、SSL のデバッグをオンにします。all の場合は、冗長メッセージを使用した SSL のデバッグをオンにします。

デバッグ・プロパティの詳細は、2-22 ページの「OC4J のデバッグ」を参照してください。

表 B-9 stdout/stderr アーカイブ管理プロパティ

プロパティ	説明
`stdstream.filesize=<max_file_size>`	アーカイブ内のファイルが増大可能な最大サイズ（MB 単位）。この最大値に達すると、ファイルが入れ替えられます。
`stdstream.filenumber=<max_files>`	アーカイブとして保持するファイルの最大数。この制限を超えると、最も古いファイルが自動的に削除されます。
`stdstream.rotatetime=<HH:mm>`	毎日ログ・ファイルの入替えが行われる時刻。

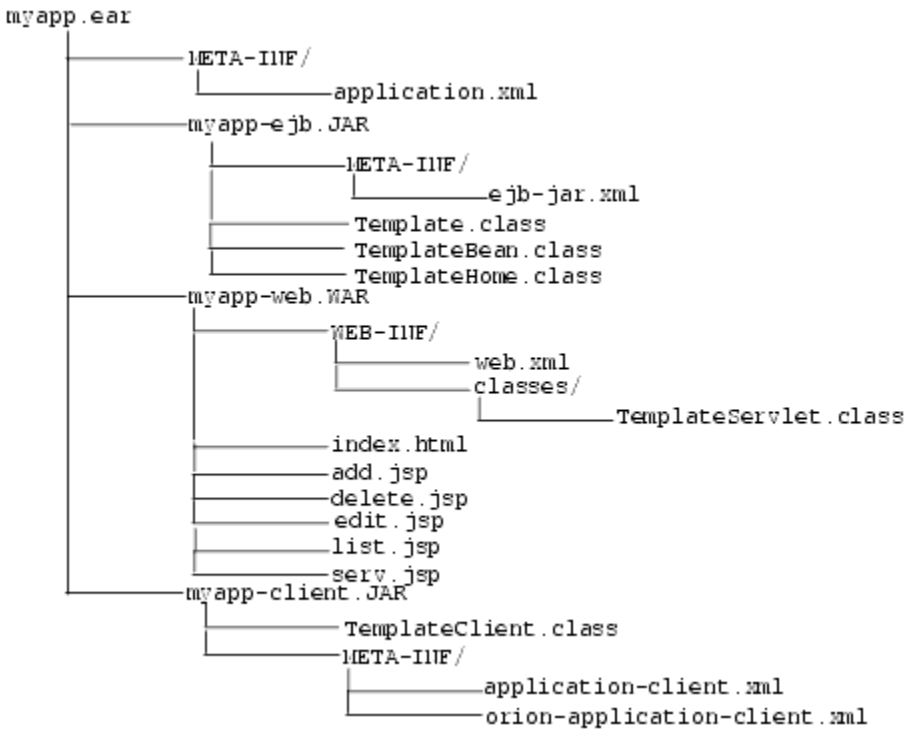
構成およびデプロイの例

次の例で、OC4J における J2EE アプリケーションの構成およびデプロイ方法を示します。FAQ アプリケーション・デモの XML 構成ファイルの変更方法は、1-7 ページの「FAQ アプリケーション・デモの構成」を参照してください。

J2EE アプリケーションの XML 構成例

この例では、myapp アプリケーションには、Java クライアント、JAR ファイルにアセンブルされた EJB、WAR ファイルにアセンブルされたサーブレットと JSP、EJB の JAR ファイルと Web アプリケーションの WAR ファイルの両方が入っている EAR ファイルが含まれています。すべての XML 構成ファイル、Java クライアント・ファイル、および JSP ファイルの場所を表すツリー構造を、次の「アプリケーション EAR の構造」に示します。すべての構成ファイルを、アプリケーション・ディレクトリ内の論理ディレクトリに分けることができる点に注意してください。

アプリケーション EAR の構造



application.xml の例

myapp/META-INF/application.xml ファイルには、<module> 要素を使用する EAR ファイルに含まれる、EJB JAR および Web アプリケーション WAR ファイルがリストされています。

```
<?xml version="1.0"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.3//EN"
"http://java.sun.com/j2ee/dtds/application_1_3.dtd">
<application>
  <display-name>myapp j2ee application</display-name>
  <description>
     A sample J2EE application that uses a Container Managed
     Entity Bean and JSPs for a client.
```

```
  </description>
  <module>
    <ejb>myapp-ejb.jar</ejb>
  </module>
  <module>
    <web>
      <web-uri>myapp-web.war</web-uri>
      <context-root>/myapp</context-root>
    </web>
  </module>
</application>
```

web.xml の例

myapp/web/WEB-INF/web.xml ファイルには、EJB のクラス定義、サーブレット、および Web サイト内で実行される JSP が含まれます。myapp という Web モジュールは、ディスクリプタで次のようなものを指定します。

- admin.jar bind コマンドで指定したアプリケーションのルート・コンテキスト（http://*oc4j_host:port*/myapp）に対して表示するデフォルトのページ
- EJB ホームおよびリモート・インタフェースのスタブを検索する場所
- EJB の JNDI 名
- インクルードされたサーブレット、および各サーブレット・クラスの検索場所
- アプリケーションのルート・コンテキストから <servlet-mapping> 要素（/template）を切り離して使用し、サーブレットをサブコンテキストにマップする方法

Web サーバーは次を検索します。

- WEB-INF/classes/<package>.<class> の下にある全サーブレット・クラス。
- 対応するアプリケーション EAR ファイルにパッケージされている web-site.xml ファイルの <web-app name="<warfile.war>"> が指定する WAR ファイルのルートから全 HTML および JSP を検索します。
- OC4J は、最初に使用したときに、各 JSP を .java から .class にコンパイルして、次回から使用できるようにキャッシュします。

```
<web-app>
   <display-name>myapp web application</display-name>
   <description>
      Web module that contains an HTML welcome page, and 4 JSP's.
   </description>
   <welcome-file-list>
      <welcome-file>index.html</welcome-file>
   </welcome-file-list>
   <ejb-ref>
      <ejb-ref-name>TemplateBean</ejb-ref-name>
      <ejb-ref-type>Entity</ejb-ref-type>
      <home>TemplateHome</home>
      <remote>Template</remote>
   </ejb-ref>
   <servlet>
      <servlet-name>template</servlet-name>
      <servlet-class>TemplateServlet</servlet-class>
      <init-param>
         <param-name>length</param-name>
         <param-value>1</param-value>
      </init-param>
   </servlet>
</web-app>
```

ejb-jar.xml の例

ejb-jar.xml ファイルには、コンテナ管理による永続的な EJB の定義が含まれています。myapp の EJB デプロイメント・ディスクリプタには、次のものが含まれます。

- Entity Bean は、コンテナ管理の永続性を使用します。
- 主キーは、テーブルに格納されます。このディスクリプタは、主キーのタイプおよびフィールドを定義します。
- テーブル名は TemplateBean で、列の名前は、ejb-jar.xml ディスクリプタのフィールド、および j2ee/home/config/database-schemas/oracle.xml のタイプ・マッピングに基づいて付けられます。
- Bean は、orion-application.xml の ejb-location または default-data-source により data-source.xml に指定されているとおり、JDBC を使用してデータベースにアクセスします。

```
<ejb-jar>
   <display-name>myapp</display-name>
   <description>
      An EJB app containing only one Container Managed Persistence
      Entity Bean
   </description>
   <enterprise-beans>
      <entity>
         <description>
            template bean populates a generic template table.
         </description>
         <display-name>TemplateBean</display-name>
         <ejb-name>TemplateBean</ejb-name>
         <home>TemplateHome</home>
         <remote>Template</remote>
         <ejb-class>TemplateBean</ejb-class>
         <persistence-type>Container</persistence-type>
         <prim-key-class>java.lang.Integer</prim-key-class>
         <reentrant>False</reentrant>
         <cmp-field><field-name>empNo</field-name></cmp-field>
         <cmp-field><field-name>empName</field-name></cmp-field>
         <cmp-field><field-name>salary</field-name></cmp-field>
         <primkey-field>empNo</primkey-field>
      </entity>
   </enterprise-beans>
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>TemplateBean</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>NotSupported</trans-attribute>
      </container-transaction>
      <security-role>
         <description>Users</description>
         <role-name>users</role-name>
      </security-role>
   </assembly-descriptor>
</ejb-jar>
```

server.xml の追加

admin.jar -deploy オプションを使用してアプリケーションをデプロイする際に、アプリケーションの EAR ファイルの場所が server.xml ファイルに追加されます。これにより、OC4J が起動するたびにアプリケーションが起動されます。OC4J でアプリケーションが起動されないようにするには、auto-start 属性を FALSE に変更します。

注意： auto-start を FALSE に設定すると、admin.jar ツールを使用して手動でアプリケーションを起動できます。設定しない場合は、クライアントがアプリケーションをリクエストしたときに自動的に起動されます。

```
<application name="myapp" path="../myapp/myapp.ear"
auto-start="true" />
```

各項目の説明：

- name 属性は、アプリケーションの名前です。
- path は、EAR ファイルのディレクトリおよびファイル名を示します。
- auto-start 属性は、OC4J の起動のたびにこのアプリケーションを自動的に起動するかどうかを示します。

http-web-site.xml の追加

WAR ファイル名を指定し、その WAR ファイルにデプロイされた Web アプリケーションのルート・コンテキストを定義する必要があります。admin.jar -bindWebApp オプションを使用して Web コンテキストをバインドするか、http-web-site.xml ファイルを編集して次の要素を追加します。

```
<web-app application="myapp" name="myapp-web" root="/myapp" />
```

- name 属性は、WAR ファイル名から .WAR 拡張子を除いた部分です。
- root 属性は、Web サイト外でのアプリケーションのルート・コンテキストを定義します。たとえば、Web サイトを http://*oc4j_host:*8888 と定義した場合、アプリケーションを起動するには、ブラウザに http://*oc4j_host:*8888/myapp と指定します。

クライアントの例

myapp アプリケーションにアクセスするアプリケーション・クライアントはディスクリプタを持っており、これが EJB スタブ（ホームおよびリモート・インタフェース）および JNDI 名のルックアップを行う場所を示します。

クライアント XML 構成は、application-client.xml および orion-application-client.xml という 2 つのファイルに含まれています。

application-client.xml ファイルには、次のような、EJB 参照が含まれています。

```
<application-client>
<display-name>TemplateBean</display-name>
<ejb-ref>
<ejb-ref-name>TemplateBean</ejb-ref-name>
<ejb-ref-type>Entity</ejb-ref-type>
<home>mTemplateHome</home>
<remote>Template</remote>
</ejb-ref>
</application-client>
```

orion-application-client.xml ファイルは、EJB 参照の論理名を、EJB の JNDI 名にマップします。たとえば、このファイルは、application-client.xml に定義されている "TemplateBean" という <ejb-ref-name> 要素を、次のように、"myapp/myapp-ejb/TemplateBean" という JNDI 名にマップします。

```
<orion-application-client>
<ejb-ref-mapping name="TemplateBean" location="myapp/myapp-ejb/TemplateBean" />
</orion-application-client>
```

クライアントの JNDI プロパティ 初期 JNDI コンテキスト・ファクトリを検索するように、通常のクライアント用の JNDI プロパティを次のいずれかの方法で設定します。

- ハッシュテーブル内の JNDI プロパティを設定してから、そのプロパティを、`javax.naming.InitialContext` に渡します。
- `jndi.properties` ファイルの中の JNDI プロパティを設定します。

  `jndi.properties` ファイルに JNDI プロパティを指定する場合、プロパティを `myapp-client.jar` にパッケージして、必ず `CLASSPATH` 内に置くようにしてください。

```
jndi.properties:
---------------
java.naming.factory.initial=com.evermind.server.ApplicationClientInitialContextFact
ory
java.naming.provider.url=ormi://oc4j_host:23791/myapp
java.naming.security.principal=admin
java.naming.security.credentials=welcome
```

デプロイの例

J2EE アプリケーションをデプロイした後、J2EE アプリケーションの各種モジュール（EJB、Web およびクライアント）を 1 つの EAR ファイルにまとめます。この項では、EJB、Web、クライアントの各セクションを持つ J2EE アプリケーションの例を示します。

`admin.jar` コマンドライン・ツールを使用して、このアプリケーションをクライアントからデプロイするには、`myapp` ディレクトリから次のコマンドを実行します。`-file` オプションで EAR ファイルを定義し、`-targetPath` オプションで EAR ファイルのコピー先となるターゲット・パスを定義していることに注意してください。EAR の位置するパスとターゲット・パスが同じであるため、コピーは行われません。

```
% java -jar $J2EE_HOME/admin.jar ormi://oc4j_host admin welcome
-deploy -file ./myapp.ear -deploymentName myapp
Auto-deploying myapp (New server version detected)...
Auto-deploying myapp-ejb.jar (ejb-jar.xml had been touched since the previous
deployment)... done.
Auto-deploying myapp web application (New server version detected)...
```

注意： EJB の JAR ファイルはただちに解凍されます。WAR ファイルは、Web サーバー上の `/myapp` にナビゲートしたときに解凍されます。

EJB モジュール

EJB モジュールをデプロイしたときに、次のメッセージが表示されました。

```
Auto-deploying myapp (New server version detected)...
Auto-creating table: create table TemplateBean (col_1 NUMBER not null primary key, col_
2 VARCHAR2(255) null, col_3 FLOAT null)
Auto-deploying myapp-ejb.jar (Class 'myapp.myapp-ejb.Template' had been updated)...
done.
```

`TemplateBean` 表は自動的に作成されましたが、まずデータソースをインストールする必要があります。`admin.jar` コマンドライン・ツールを使用して、次のようにデータソースをインストールできます。

```
% java -jar admin.jar ormi://oc4j_host admin welcome
-installDataSource -jar $ORACLE_HOME/jdbc/classes12.jar
-url jdbc:oracle:thin:@oc4j_host:1521/MYSERVICE
-connectionDriver oracle.jdbc.driver.OracleDriver
-location jdbc/DefaultOracleDS -username scott -password tiger
```

Web モジュール : EJB をコールするサーブレットおよび JSP

Web サイト上の J2EE アプリケーション（EAR ファイル）の Web コンポーネント（WAR ファイル）をバインドするには、次のコマンドを実行します。

```
% java -jar admin.jar ormi://oc4j_host admin welcome
-bindWebApp myapp myapp-web http-web-site /myapp
```

このコマンドでは、次の要素が http-web-site.xml に追加されます。

```
<web-app application="myapp" name="myapp-web" root="/myapp" />
```

クライアント・モジュール : EJB を起動するスタンドアロンの Java クライアント

クライアント・モジュールは、META-INF/application-client.xml ディスクリプタを持つ JAR ファイルにパッケージします。

クライアントのマニフェスト・ファイル 次に示すとおり、クライアントを、実行用のメイン・クラスおよび必要な CLASSPATH を持ったマニフェストを持つ実行可能な JAR ファイルにパッケージします。このファイルの相対パスが正しいことを確認してください。必要な OC4J クラス・ライブラリの相対的な場所を指定していることを確認してください。

```
manifest.mf
-----------
Manifest-Version: 1.0
Main-Class: myapp.myapp-client.TemplateClient
Name: "TemplateClient"
Created-By: 1.2 (Sun Microsystems Inc.)
Implementation-Vendor: "Oracle"
Class-Path: ../../../j2ee/home/oc4J.jar ../../../j2ee/home/lib/jndi.jar
../../../j2ee/home/lib/ejb.jar ../myapp-ejb.jar
```

クライアントの実行 クライアントを実行するには次のようにします。

```
% java -jar myapp-client.jar
TemplateClient.main(): start
Enter integer value for col_1: 1
Enter string value for col_2: BuyME
Enter float value for col_3: 99.9
Record added through bean
```

C

サード・パーティ・ライセンス

この付録には、Oracle Application Server に付属するすべてのサード・パーティ製品のサード・パーティ・ライセンスが記載されています。

サード・パーティ・ライセンス

この付録には次の項目が含まれています。

Apache HTTP Server

Apache のライセンス条件に基づき、オラクル社は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Apache ソフトウェアは現状のままでオラクル社から提供されるものであり、いかなる種類の保証またはサポートもオラクル社または Apache から提供されません。

The Apache Software License

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ''AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
```

```
* SUCH DAMAGE.
* ====================================================================
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation.  For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*
* Portions of this software are based upon public domain software
* originally written at the National Center for Supercomputing Applications,
* University of Illinois, Urbana-Champaign.
*/
```

索引

記号

A

C

D

E

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W

X

あ

か

さ

た

な

は

や

ら