Oracle® Application Server Containers for J2EE

ユーザーズ・ガイド

10*g* リリース 2（10.1.2）

部品番号 : B15630-02

2005 年 10 月

ORACLE®

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

部品番号 : B15630-02

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

原本部品番号 : B14011-02

原本著者 : Sheryl Maring, Dan Hynes

原本協力者 : Brian Wright, Timothy Smith

目次

3 高度な構成および開発

4 OC4J のクラスタリング

A OC4J のトラブルシューティング

B 追加情報

はじめに

ここでは、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』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、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を前提としています。

- 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

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、次の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 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 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 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 管理者ガイド』
- 『Oracle Application Server パフォーマンス・ガイド』
- 『Oracle Application Server 高可用性ガイド』
- 『Oracle Database 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

OC4J の概要

この章では、OC4J の概要と、Oracle Application Server に OC4J をインストールする方法を説明します。

この章には、次の項目が含まれています。

OC4J の概要

Oracle Application Server 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 Virtual Machine（JVM）で実行されます。Oracle Application Server Containers for J2EE（OC4J）は、ご使用のオペレーティング・システムにある標準 JDK で実行できます。詳細は、`http://www.oracle.com/technology/index.html` の動作保証マトリックスを参照してください。

OC4J は、J2EE 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。OC4J は、Oracle Application Server Infrastructure に統合されていますが、製品およびドキュメントの一部では、Orion Server を参照するよう指示している箇所もあります。

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 J2EE のサポート

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
JAAS	1.0
J2EE Connector Architecture（JCA）	1.0
JAXP	1.1

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 で提供される例

Oracle Application Server 10*g*（9.0.4）には、コンポーネントごとに 1 つのデモが含まれています。追加のデモは、次の URL の OTN からダウンロードできます。

`http://www.oracle.com/technology/tech/java/oc4j/demos/`

このページから、必要な追加のデモが含まれているコンポーネントの領域までドリルダウンしてください。

JDK 1.4 の考慮事項 : パッケージ内に存在しないクラスは起動不可

Oracle Application Server 10*g* とともに提供される Sun Microsystems JDK 1.4 環境への移行において、考慮事項の 1 つに、サーブレットおよび JSP の開発者に対する重要な注意事項があります。

Sun 社は、「この JDK バージョンでは、名前のないネームスペースから型をインポートする Import 文は、コンパイラで拒否されるようになった」と説明しています。（前の JDK バージョンでのセキュリティ上の問題とあいまいさを解決するための対応です。）これは、パッケージ内に含まれていないクラス（クラスのメソッド）が起動できなくなったことを意味します。このようなパッケージを起動しようとした場合、致命的なコンパイル時のエラーが発生します。

これは、JSP ページから JavaBeans を起動する JSP 開発者に特に関係があります。このような Bean はパッケージに含まれないことが多いためです（ただし、JSP 2.0 仕様では、新しいコンパイラの要件を満たすために、Bean をパッケージ内に含めることが要求されています）。パッケージ外の JavaBeans が起動された場合、OC4J 9.0.3 / JDK 1.3.1 環境で作成され実行された JSP アプリケーションは、Oracle Application Server 10*g* / JDK 1.4 環境では機能しなくなります。

アプリケーションを更新してすべての JavaBeans および起動されるその他のクラスをパッケージ内に含めるまでは、この問題は JDK 1.3.1 環境に戻すことにより回避できます。

注意 :

- `javac -source` コンパイラ・オプションは、JDK 1.3.1 コードを JDK 1.4 コンパイラで透過的に処理することを意図していますが、このオプションでは、「パッケージ内に存在しないクラス」の問題は考慮されません。
- JDK 1.3.1 および JDK 1.4 コンパイラのみが OC4J でサポートされ、OC4J に準拠しています。`<java-compiler>` 要素を `server.xml` ファイルに追加して代替コンパイラを指定することにより「パッケージ内に存在しないクラス」の問題を回避できる場合がありますが、オラクル社では、他のコンパイラを OC4J で使用することについて、動作確認やサポートは行っていません。（また、`server.xml` ファイルを Oracle Application Server 環境で直接更新しないでください。Oracle Enterprise Manager 10*g* を使用してください。）

「パッケージ内に存在しないクラス」の問題、および JDK 1.4 の互換性に関するその他の問題の詳細は、次の Web サイトを参照してください。

`http://java.sun.com/j2se/1.4/compatibility.html`

特に、「Incompatibilities Between Java 2 Platform, Standard Edition, v1.4.0 and v1.3」というサイトは参照してください。

ソース・コードの JDK 1.4 から JDK 1.3 への移行

Java では、JDK 1.4 でコンパイルされた Java コードの JDK 1.3 での実行はサポートされていません（bug 2811379）。これを試みた場合、実行時に次のいずれかの Java エラー・メッセージが表示されます。

- `Unsupported major.minor version 48.0.`
- `The major.minor version '48.0' is too recent for this tool to understand.`

ただし、JDK 1.4 を使用してクラス・ファイルを生成し、JDK 1.3 を使用して実行する必要がある場合は、次のコマンドにより JDK 1.4 コンパイラで JDK 1.3 と互換性のあるクラス・ファイルを生成することができます。

```
% javac -target 1.3 hello.java
```

ソース・コードの JDK 1.3 から JDK 1.4 への移行

JDK 1.3 から JDK 1.4 へ移行する場合、すべてのクラスをパッケージに入れます。JDK 1.4 仕様では、パッケージ内クラスからパッケージ外のクラスを起動することは禁じられています。このため、JDK 1.3 でコンパイルされる一部の Java ソース・コードは、JDK 1.4 でコンパイルされません。このエラーは、次のコンパイラ・メッセージで知らされます。

```
'.' expected import myClass
```

このメッセージは、パッケージおよびクラス名を見つける Java コンパイラでは、クラス名のみを含む行を解析できないことを意味します。

詳細は、JDK に付属している Sun 社の互換性に関するドキュメントを参照してください。

OC4J ドキュメントのナビゲート

J2EE の情報元の多くは公開されています。たとえば、サーブレットを実装および使用する方法については、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』に記載されています。表 1-2 は、J2EE 関連項目と、その情報を提供している OC4J ドキュメントを示しています。

表 1-2 J2EE 関連項目の記載場所

J2EE 関連項目	記載されている OC4J ドキュメント
JSP	『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』
JSP タグ・ライブラリ	『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』
サーブレット	『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』
EJB	『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』
JTA	『Oracle Application Server Containers for J2EE サービス・ガイド』
データ・ソース	『Oracle Application Server Containers for J2EE サービス・ガイド』
JNDI	『Oracle Application Server Containers for J2EE サービス・ガイド』
JMS	『Oracle Application Server Containers for J2EE サービス・ガイド』
RMI および RMI/IIOP	『Oracle Application Server Containers for J2EE サービス・ガイド』
セキュリティ	『Oracle Application Server Containers for J2EE セキュリティ・ガイド』
CSiV2	『Oracle Application Server Containers for J2EE セキュリティ・ガイド』
JCA	『Oracle Application Server Containers for J2EE サービス・ガイド』
Java Object Cache	『Oracle Application Server Containers for J2EE サービス・ガイド』
Web サービス	『Oracle Application Server Web Services 開発者ガイド』
HTTPS	『Oracle Application Server Containers for J2EE サービス・ガイド』

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。OC4J は、Oracle Application Server とともにインストールされます。OC4J インストールの詳細は、Oracle Application Server のインストレーション・ガイドを参照してください。

エンタープライズまたはスタンドアロン環境での OC4J の使用

OC4J は、J2EE エンタープライズ・システムを管理する目的で、Oracle Application Server 内でインストールされます。Oracle Application Server では、クラスタリングされた複数の OC4J プロセスを管理できます。OC4J を含む Oracle Application Server は、複数のアプリケーション・サーバーのインスタンスとホストにわたり OC4J プロセスを管理および構成できる Oracle Enterprise Manager 10*g* によって管理され、構成されます。このため、`admin.jar` ツールを使用、あるいは単独の OC4J プロセスの構成ファイルを手動編集することによって、OC4J プロセスをローカルで管理することはできません。ローカルで管理すると、Oracle Enterprise Manager 10*g* で実現されるエンタープライズ管理を損なうことになります。

ただし、これまでどおりに OC4J を実行することも可能です。開発環境での使用や、簡易なビジネス要件を満たすために単独の OC4J インスタンスを使用する場合は、OC4J をスタンドアロン・モードでドキュメントとともにダウンロードできます。

次の項では、管理オプションについて説明します。

- 複数 OC4J インスタンスのエンタープライズ環境での管理
- 単一 OC4J インスタンスの管理
- OC4J での JDK の使用

また、次の項では、OC4J ドキュメントを理解する方法について説明します。

- OC4J ドキュメントに関する前提

複数 OC4J インスタンスのエンタープライズ環境での管理

OC4J を含む Oracle Application Server は、エンタープライズ・システム内で Oracle Enterprise Manager 10*g* を使用することによって管理します。これには、クラスタリング、高可用性、ロード・バランシングおよびフェイルオーバーが含まれます。

各 OC4J インスタンスとそれぞれのプロパティを、Oracle Enterprise Manager 10*g* を使用して Application Server インスタンスのコンテキスト内で構成します。構成後は、あらゆる OC4J インスタンスの起動、管理および制御を Oracle Enterprise Manager 10*g* によって行います。複数の OC4J プロセスを 1 つのクラスタにグループ化することも可能です。アプリケーションの起動、終了、再起動、構成およびデプロイには、Oracle Enterprise Manager 10*g* 管理ツールまたはコマンドライン・ツールのいずれかを使用する必要があります。

注意： OC4J スタンドアロン用のツール `admin.jar` は、Application Server インスタンスで作成された OC4J インスタンスの管理には使用できません。

XML ファイルはローカルで変更可能です。その場合は、手動で XML ファイルを編集したことを Distributed Configuration Management（DCM）コンポーネント・ツール `dcmctl` によって Oracle Enterprise Manager 10*g* に通知する必要があります。XML ファイルを手動で編集した後に、次のコマンドを実行します。

```
dcmctl updateconfig -ct oc4j
```

DCM によって、Oracle Application Server インスタンスとその Oracle HTTP Server コンポーネントと OC4J コンポーネントの構成が制御および管理されます。DCM の詳細は、『Distributed Configuration Management 管理者ガイド』を参照してください。

このマニュアルでは、エンタープライズ環境での OC4J の起動、終了、管理および構成方法について説明します。

単一 OC4J インスタンスの管理

Oracle Application Server 環境の外部で単一の OC4J を使用することも可能です。OTN から `oc4j_extended.zip` 内の OC4J をダウンロードすると、`oc4j.jar` と `admin.jar` コマンドライン・ツールによってすべての OC4J インスタンスを起動、管理および制御できるようになります。OC4J インスタンスは、`admin.jar` コマンドを使用するか、XML ファイルを手動で変更するかのいずれかで構成します。

スタンドアロン OC4J プロセスは、Oracle Enterprise Manager 10*g* で管理されないため、Oracle Application Server エンタープライズ環境では使用できません。通常、スタンドアロンは、開発用または単一 OC4J インスタンスによる簡易な Web ソリューション用に使用します。

スタンドアロン・プロセスの起動、終了、構成および管理方法を知りたい場合は、『Oracle Application Server Containers for J2EE スタンドアロン・ユーザーズ・ガイド』をダウンロードしてください。

OC4J での JDK の使用

システムにインストールする Java Developer's Kit（JDK）は 1 つです。JDK が OC4J でサポートされているバージョンであることを確認してください。

- JDK 1.3.1
- JDK 1.4.1
- JDK 1.4.2

OC4J Standalone には JDK は含まれていません。OC4J Standalone を使用している場合は、独自に JDK をインストールする必要があります。

製品に JDK をバンドルしているサプライヤもあります。必要に応じて、古いバージョンの JDK を削除し、サポートされているバージョンに置き換え、適切な環境変数を更新します。PATH、CLASSPATH および LD_LIBRARY_PATH（Windows では LIB）の各変数が、すべて公式にサポートされている JDK のバージョンに設定されていることを確認してください。

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" />
```

OC4J ドキュメントに関する前提

このマニュアル以外の OC4J のドキュメントは、スタンドアロンの OC4J を前提として書かれています。それらのドキュメントでは、XML ファイルを手動で変更してインスタンスを管理する方法を解説しています。このマニュアルを読むと、Oracle Enterprise Manager 10*g* 構成ページの概要と機能を理解できます。また、Oracle Enterprise Manager 10*g* の各ページとそれに対応する XML との関連についても理解できます。Oracle Enterprise Manager 10*g* をよく理解した上で、OC4J の他のマニュアルを読んでください。XML 表現を見て、それに対応する Oracle Enterprise Manager 10*g* フィールド名がわかるようになる必要があります。

また、Distributed Configuration Management（DCM）ユーティリティの `dcmctl` は、Oracle Enterprise Manager 10*g* の管理作業の一部をコマンドラインで行う代替方法を提供します。`dcmctl` ツールは、Oracle Enterprise Manager 10*g* と同じ分散アーキテクチャおよび同期化機能を使用するため、スクリプト作成と自動化に理想的なフォーマットで、同じ機能を実現します。

DCM によって、次の機能を管理できます。

- 管理
- Application Server インスタンスの管理
- コンポーネントの管理

- クラスタの管理
- アプリケーションのデプロイ

OC4J に関連するその他の DCM コマンドは、『Distributed Configuration Management 管理者ガイド』を参照してください。

OC4J の通信

HTTP アプリケーションの場合、OC4J は Oracle HTTP Server（OHS）のバックグラウンドで実行されるよう事前に構成されています。Oracle HTTP Server はフロントエンド・リスナーとして使用し、OC4J はバックエンド J2EE アプリケーション・サーバーとして使用します。

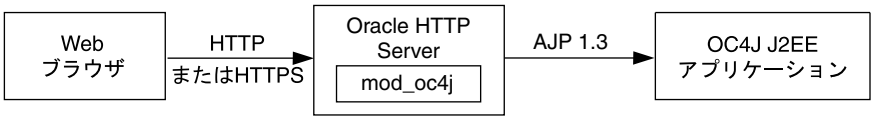
ただし、EJB や JMS など RMI ベースのアプリケーションの場合は、リクエストを直接 OC4J に送信する必要があります。手順は、3-20 ページの「OC4J リスナーの理解および構成」を参照してください。

HTTP 通信

アプリケーション・サーバー環境内のすべての受信 HTTP 通信について、OHS をフロントエンド・リスナー、OC4J をバックエンド J2EE アプリケーション・サーバーとして使用します。これを図 1-1 に示します。

1. ブラウザはすべての HTTP リクエストで OHS リスナーにアクセスします。Oracle HTTP Server は、Apache Server です。デフォルトのポート番号は 7777 です。
2. OHS は、mod_oc4j モジュールを通じて、リクエストを OC4J サーバーに渡します。OHS と OC4J 間の接続では、OC4J 起動時にネゴシエートされたポート番号上で Apache JServ プロトコル（AJP）が使用されます。AJP は、バイナリ形式を使用し、効率的にメッセージ・ヘッダーを処理するため、HTTP よりも高速です。

図 1-1 HTTP アプリケーション・リスナー



mod_oc4j モジュールは、j2ee/ Web コンテキスト下のあらゆる受信 HTTP リクエストを OC4J に送るように、事前に構成されています。これによって、デフォルト・ルーティングを使用する場合、接頭辞として j2ee/ を含むサーブレット・コンテキストに Web アプリケーションをデプロイすることができます。ただし、デプロイ・ウィザードで指定した URL マッピングは、自動的に mod_oc4j モジュールに追加されます。デプロイ時に mod_oc4j に追加されるものについては、3-22 ページの「別の Web コンテキストによる Oracle HTTP Server の構成」を参照してください。mod_oc4j モジュールの詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。

注意： Oracle9*i*AS リリース 1.0.2.2 では、デフォルトの OC4J Web サイトが Oracle HTTP Server をフロントエンドとして使用せず、HTTP プロトコルを使用してポート 8888 でリスニングしていました。

要件

最適なパフォーマンスのためには、Oracle Application Server とともにインストールされる JDK、つまり JDK 1.4.2 を使用して OC4J を実行してください。

OC4J を実行するために、CLASSPATH に何も追加する必要はありません。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 EAR 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。

2

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 は、Oracle Application Server のインストールによってインストールされます。

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。

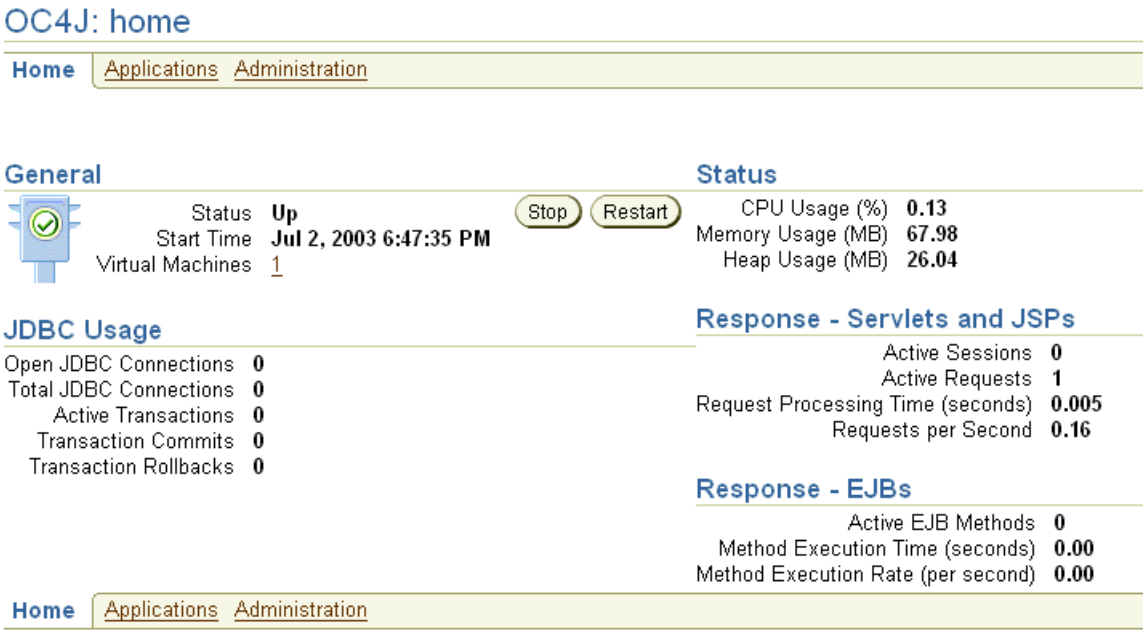
COC4J ホームページの概要

多くの場合、OC4J インスタンスの設定および管理は、OC4J ホームページから実行します。OC4J インスタンスを Oracle Application Server インスタンス・ホームページから作成すると、OC4J インスタンスの設定および管理用の OC4J ホームページが作成されます。OC4J インスタンスには、それぞれに専用の OC4J ホームページがあります。

Oracle9*i*AS コンソールから、「システム・コンポーネント」表内のインスタンスの名前（home など）を選択して、実行中の OC4J インスタンスにドリルダウンできます。Oracle9*i*AS コンソールには、そのインスタンスの OC4J ホームページが表示されます。

図 2-1 は、home インスタンスの OC4J ホームページの一部です。

図 2-1 Oracle Application Server コンソール OC4J ホームページ



OC4J ホームページには、OC4J インスタンスとそのアプリケーションに関するメトリックが表示されます。さらに、このインスタンスに設定されているすべての OC4J プロセスを起動、停止および再起動できます。

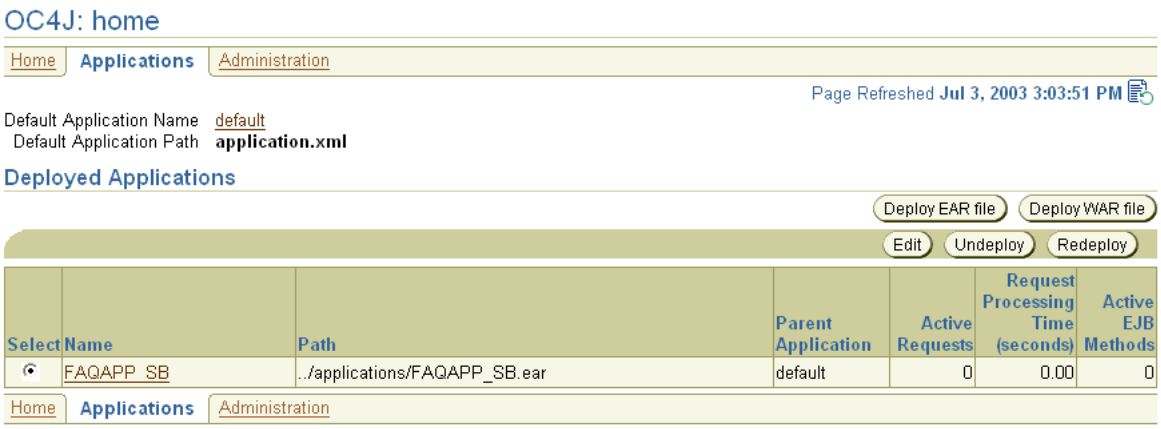
OC4J ホームページから次のページにナビゲートできます。

- **「アプリケーション」**をクリックして、Oracle9*i*AS コンソールの「アプリケーション」ページにアクセスします。詳細は、2-2 ページの「「アプリケーション」ページ」を参照してください。
- **「管理」**をクリックして、Oracle9*i*AS コンソールの「管理」ページにアクセスします。詳細は、2-3 ページの「「管理」ページ」を参照してください。

「アプリケーション」ページ

図 2-2 は「デプロイ済アプリケーション」セクションです。このセクションで**「EAR ファイルのデプロイ」**または**「WAR ファイルのデプロイ」**ボタンを使用すると、アプリケーションをデプロイできます。デプロイ後には、各アプリケーションの構成を変更できます。詳細は、2-11 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。

図 2-2 デプロイ済アプリケーション



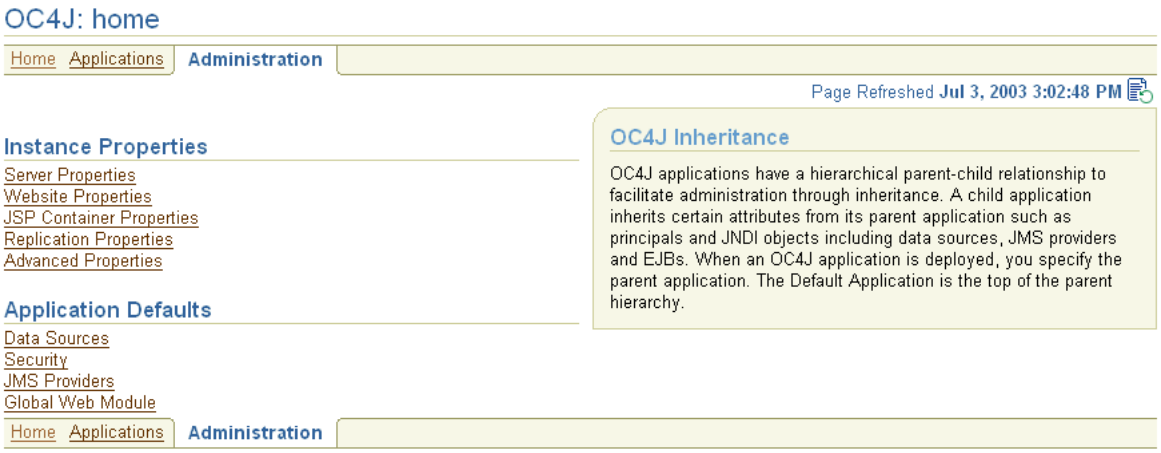
例として、2-6 ページの「FAQ アプリケーション・デモの構成」に、FAQ アプリケーション・デモを構成して OC4J にデプロイする方法を示します。

「管理」ページ

図 2-3 は「管理」ページです。次の設定を変更できます。

- 「インスタンス・プロパティ」では、特定の OC4J インスタンスのグローバルな設定値を変更できます。これには、RMI、JMS、Web サイトなどの OC4J サービスの設定が含まれます。
- 「アプリケーション・デフォルト」では、この OC4J インスタンス内のすべてのデプロイされたアプリケーションによって使用可能なデフォルトのデータ・ソースとセキュリティを設定できます。

図 2-3「管理」セクション



これらのオプションの詳細は、3-2 ページの「Enterprise Manager を使用した OC4J の構成」で説明します。

OC4J の起動および停止

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。これにより、構成を追加することなく、すぐに OC4J を起動できます。

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" />
```

Application Server Control から、次のいずれかの方法で OC4J を起動、停止および再起動できます。

- Oracle Application Server インスタンスのホームページにドリルダウンし、「一般」セクションの**「すべてを起動」**ボタンをクリックして、構成済のすべての OC4J インスタンスを含む Oracle Application Server インスタンス全体を起動します。また、**「すべてを停止」**および**「すべてを再起動」**もそれぞれの目的で使用可能です。
- Oracle Application Server インスタンス・ホームページにドリルダウンし、特定 OC4J インスタンスの横にあるラジオ・ボタンを選択すると、その OC4J インスタンスを起動できます。起動するには、**「起動」**ボタンをクリックします。特定の OC4J インスタンスを停止、再起動または削除するには、**「停止」**、**「再起動」**または**「削除」**をクリックします。
- Oracle Application Server インスタンス・ホームページから、OC4J ホームページにドリルダウンします。このページの「一般」セクションの**「起動」**ボタンをクリックします。また、**「停止」**および**「再起動」**もそれぞれの目的で使用可能です。図 2-1 は、OC4J ホームページの「一般」セクションを示します。

OC4J では、デプロイされたアプリケーションに追加された変更を自動的に検出し、そのアプリケーションを自動でリロードします。そのため、アプリケーションの再デプロイ時には、サーバーを再起動する必要はありません。ただし、「管理」ページのオプションのフィールドを変更した場合は OC4J を再起動する必要があります。

制御コマンドを使用すると、起動、停止および再起動することも可能です。手順は、『Distributed Configuration Management 管理者ガイド』を参照してください。

デフォルトの構成のテスト

OC4J をデフォルトで起動するには、次のようにします。

1. 「Oracle Application Server インスタンス名」ページで、Oracle Application Server インスタンス全体か、少なくとも Oracle HTTP Server コンポーネントおよび OC4J コンポーネントを起動します。Oracle Application Server インスタンスを起動する場合は、**「すべてを起動」**ボタンをクリックし、コンポーネントを起動する場合は、コンポーネントを選択して**「起動」**ボタンをクリックします。
2. 次の URL を Web ブラウザで指定し、OC4J をテストします。

   ```
   http://<ohs_host>:7777/j2ee/j2ee-index.html
   ```

   <ohs_host> に OHS のインストール先のホスト名を代入します。
3. 次の URL を Web ブラウザに指定して、インストール時に OC4J にデプロイされたサーブレットをテストします。

   ```
   http://<ohs_host>:7777/j2ee/servlet/HelloWorldServlet
   ```

   このコマンドは、「Hello World」ページを返します。HelloWorldServlet は、OC4J のインストール時に自動的にデプロイされます。

注意： このガイドに示す例と URL では、OHS Web リスナーのデフォルトのポート 7777 が使用されます。OHS のデフォルトのポート番号を変更している場合は、次のように、ホスト名の後に新しいポート番号を指定します。

```
http://<ohs_host>:<ohs_port>/j2ee/
```

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。これにより、ソースのアーカイブ時には、すでに必要なアーカイブ形式が準備できています。図 2-4 に、この構造を示します。

図 2-4 開発アプリケーションのディレクトリ構造

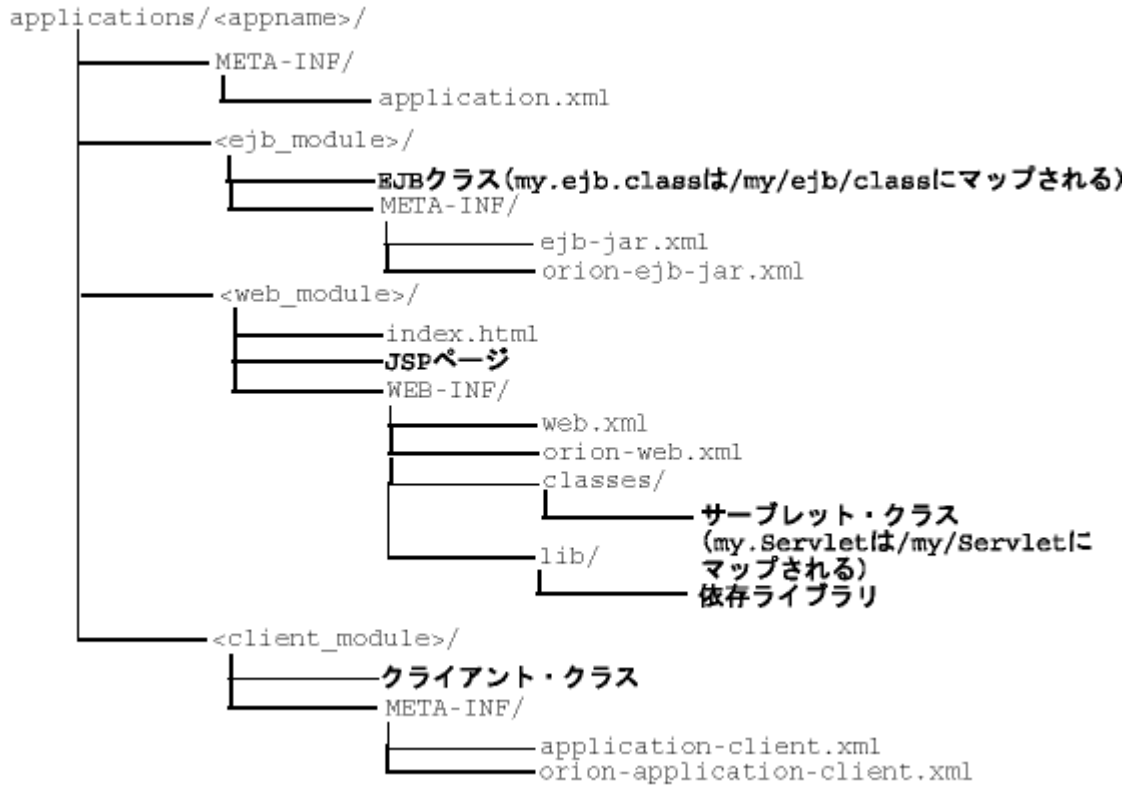


図 2-4 に関しては、次の点を考慮してください。

- 次のディレクトリ名および XML ファイル名は変更できません。`META-INF`、`WEB-INF`、`application.xml`、`ejb-jar.xml`、`web.xml` および `application-client.xml`。
- 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を付けることができます。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` 内に存在するとみなされます。

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。Oracle データベースが稼働していて、OC4J がインストールされている必要があります。このデモは、デモ用にのみ使用し、本番環境では使用しないでください。

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 Web サービス・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

FAQ デモを実行するには、デモを使用する表を含むように、バックエンド・データベースを変更する必要があります。

Oracle データベース

1. データベースでパスワード faq を使用して、faq ユーザーを追加します。
2. SQL 表作成スクリプト `CreateTables.sql` スクリプトを実行して、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、`<FAQApp_home>/faq/sql/CreateTables.sql` にあります。または、OTN（`http://www.oracle.com/technology/tech/java/oc4j/demos/`）から `FAQApp.zip` ファイルで FAQ アプリケーションの残りの部分とともに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。このデータ・ソースは、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。

---

1. Application Server Control の OC4J ホームページにナビゲートします。
2. ページ上部にある「管理」タブを選択します。
3. 「アプリケーション・デフォルト」セクションで**「データ・ソース」**を選択します。デフォルト・アプリケーションは、自動的に各アプリケーションの親になり、使用データ・ソースなど、デプロイされたすべてのアプリケーションに対するグローバル構成を保持します。FAQ アプリケーションで使用するデフォルトのデータ・ソースを変更することになります。
4. `OracleDS` データ・ソースの場合、**「編集」**ボタンをクリックします。これにより、このデータ・ソースの構成情報が表示されます。バックエンド・データベースを指すように、JDBC URL、ドライバ、ユーザー名およびパスワードを変更します。

   終了したら、**「適用」**ボタンをクリックします。

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:ORCL 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 にあるデータベース・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、認証およびユーザー・アクセス制御機能に Oracle の JAAS プロバイダ、JAZN を使用します。

1. OC4J ホームページ上部にある「管理」タブを選択します。
2. 「アプリケーション・デフォルト」セクションで**「セキュリティ」**を選択します。デフォルトでは、デプロイされているすべてのアプリケーションにグローバル構成が設定されます。FAQ アプリケーションで使用するユーザーは、`jazn.com` レルムに追加することになります。
3. 「セキュリティ」ページで、「ユーザー」セクションまでスクロール・ダウンします。
4. **「ユーザーの追加」**をクリックします。構成画面が表示され、新規ユーザーに関する情報を追加できます。次の情報を指定します。
   - ユーザーの名前およびパスワード
   - jazn.com/users レルムの横にあるボックスの選択

   終了したら、**「適用」**ボタンをクリックします。

または、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>
```

前者は、`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）、新規ユーザーに users ロールを付与します。セキュリティ・プロバイダとして JAZN を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/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. Application Server Control の OC4J ホームページにナビゲートします。
3. 画面上部にある「アプリケーション」タブを選択します。
4. **「EAR ファイルのデプロイ」**ボタンをクリックします。これにより、アプリケーション・デプロイ・ウィザードが起動します。
5. 「アプリケーションの選択」ページに EAR ファイルとアプリケーションの名前を入力します。**「参照」**ボタンをクリックして、解凍した `FAQApp.ear` ファイルを検索します。アプリケーション名フィールドに `FAQApp` と入力します。**「続行」**ボタンをクリックします。

6. FAQ アプリケーションのすべての Web モジュールに関するサーブレット・コンテキストの URL マッピングを指定します。FAQApp デモには、Web モジュールが 1 つあり、/FAQApp サーブレット・コンテキストにマッピングする必要があります。URL マッピング・フィールドに /FAQApp と入力し、**「次へ」**ボタンをクリックします。
7. この時点では、ウィザードで FAQApp デモに構成を追加する必要はありません。**「終了」**をクリックして、「サマリー」ページにジャンプできます。
8. FAQApp アプリケーション・デプロイの概要を読みます。**「デプロイ」**ボタンをクリックして、アプリケーション・デプロイを実行します。
9. OC4J ホームページで、「アプリケーション」セクションの「名前」列で FAQApp を選択します。これにより、FAQApp デモ・アプリケーションの構成とデプロイされたすべてのモジュールが表示されます。OC4J サーバーが起動している場合は、アプリケーションが自動的に起動します。
10. デフォルト・ポートが 7777 である OHS にアクセスすることによって、FAQApp アプリケーションをブラウザ内で実行します。

    ```
    http://<ohs_host>:7777/FAQApp
    ```

    「FAQApp」画面が表示されます。

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、移植性がありますが、非アプリケーション（サーバー）構成はそうではありません。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、アプリケーションがデータベースを使用する場合、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、次の項目を設定します。

- 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。
- DataSource オブジェクト : アプリケーション内で使用されるデータベースごとに、「データソース」構成ページで DataSource オブジェクトを構成する必要があります。
- アプリケーションをデプロイする場合、他のサービスと同様に、アプリケーションの名前、Web コンテキスト、およびアプリケーション・レベルのデータ・ソースまたはセキュリティを設定します。

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、次の基本手順を実行します。

基本手順	FAQ アプリケーションの手順の説明
1. アプリケーションを作成または取得します。	OTN から FAQApp.zip をダウンロードします。
2. サーバー環境の必要な変更を行います。	JAVA_HOME 変数を設定します。
3. グローバル構成を変更します。	デフォルトのデータ・ソース OracleDS が変更され、デフォルトの JAZN セキュリティにユーザーが追加されます。
4. アプリケーション・デプロイメント・ディスクリプタを指定します。	FAQApp.EAR ファイルで、web.xml や ejb-jar.xml などのデプロイメント・ディスクリプタが指定されます。
5. 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・ファイルを更新します。	application.xml ファイルは、FAQApp.EAR ファイルに含まれています。

（続き）

基本手順	FAQ アプリケーションの手順の説明
6. アプリケーションが含まれている EAR ファイルがまだない場合は、ビルドします。	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。
7. 適切なサーバー XML ファイルにアプリケーションを登録します。	アプリケーションは、デプロイ・ウィザードを使用してアプリケーションをデプロイしたときに登録されます。

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。

> **注意：** デプロイ・ウィザードの各ステップの画面を 2-11 ページの「アプリケーションのデプロイ」に示します。

1. 前述のように、FAQ デモ・アプリケーションを Oracle OTN サイトからダウンロードします。
2. 必要なサーバー環境の変更を行います。`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. グローバル構成を変更します。
   a. 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。
   b. `jazn.com` レルムに OracleAS JAAS Provider ユーザーを追加します。
4. すべてのアプリケーション XML ファイル（`web.xml` など）が正しく構成されて、ZIP ファイルで提供されます。
5. 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` が、ZIP ファイルで提供されます。OC4J では、`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。
6. アプリケーションが含まれる 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/`

再ビルドしない場合は、ZIP ファイルから `FAQApp.ear` を `j2ee/home/applications` にコピーできます。このステップにより、FAQ アプリケーションが OC4J サーバーに置かれます。

1. J2EE アプリケーションとその Web アプリケーションを、デプロイ・ウィザードに登録します。デプロイするときに、アプリケーションの名前や Web コンテキストなど、アプリケーションの登録に必要な情報をウィザードから要求されます。
2. OC4J を起動します。OC4J の起動オプションの詳細は、2-3 ページの「OC4J の起動および停止」を参照してください。
3. Web ブラウザを開き、次の URL を指定します。

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

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を説明します。デプロイ・ウィザードでアプリケーションをデプロイすると、アプリケーションが OC4J インスタンスにデプロイされ、OC4J からアプリケーションにアクセスできるようにあらゆる Web アプリケーションが URL コンテキストにバインドされます。

この項には、次の項目が含まれています。

- 基本デプロイ
- アプリケーションのアンデプロイ / 再デプロイの影響
- アプリケーションのホット・デプロイの影響

アプリケーションをデプロイするには、OC4J ホームページにドリルダウンして、「デプロイ済アプリケーション」セクションまでスクロールします。図 2-2 は、このセクションを示しています。

注意： 単純なアプリケーションは、`dcmctl` コマンドでデプロイすることもできます。手順は、『Distributed Configuration Management 管理者ガイド』を参照してください。

基本デプロイ

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 ファイルおよび 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、次のようになります。

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 .
```

`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。

- EAR ファイル内にパッケージ化された J2EE アプリケーションをデプロイするには、「アプリケーション」ページの**「EAR ファイルのデプロイ」**ボタンをクリックします。
- WAR ファイル内にパッケージ化された J2EE Web アプリケーションをデプロイするには、「アプリケーション」ページの**「WAR ファイルのデプロイ」**ボタンをクリックします。

この 2 つのボタンをクリックすると、8 つのステップから成るアプリケーション・デプロイ・ウィザードが起動され、アプリケーションのデプロイ方法を案内します。WAR ファイルの場合、`application.xml` ファイルが Web アプリケーション用に作成されます。一方、EAR ファイルの場合は、EAR ファイル内に `application.xml` ファイルを作成する必要があります。このため、WAR ファイルをデプロイする方が Web アプリケーションのデプロイ方法としては簡単です。

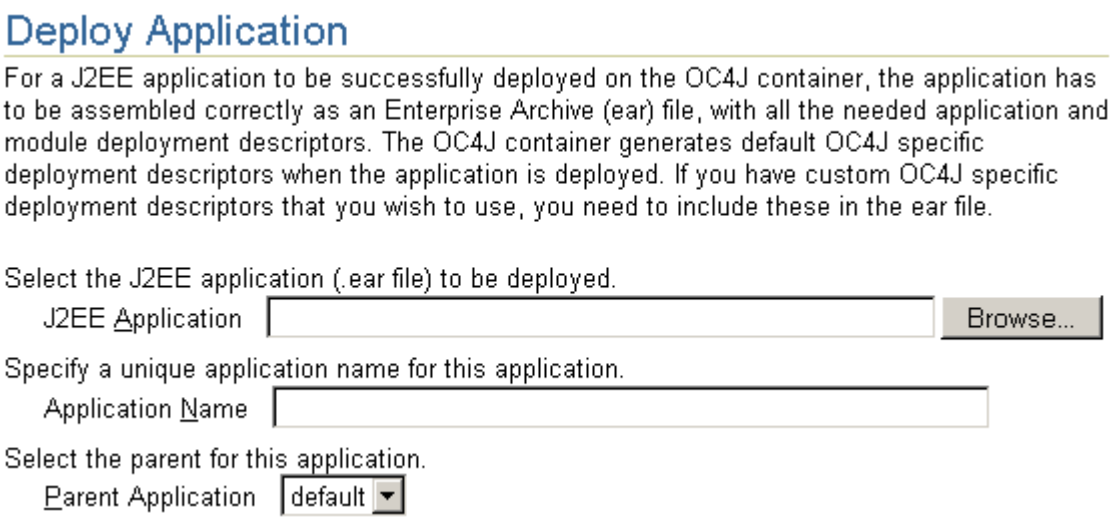
注意： J2EE サービス用に、データ・ソースやセキュリティ構成などの構成を入力する必要があります。

アプリケーションの選択

図 2-5 は最初のページを示しています。このページでは次の作業を実行できます。

- デプロイする EAR ファイルを見つけるためにシステムを参照する。
- このアプリケーションを識別するための名前を指定する。アプリケーション名は、ユーザーが作成し、「アプリケーション」ページでアプリケーションの識別子になります。
- 親アプリケーションを指定する。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。プルダウン・メニューに表示するには、親アプリケーションがデプロイ済になっている必要があります。

図 2-5 EAR ファイルの指定



このステップの情報によって、アプリケーションのデプロイ時に次の処理が実行されます。

1. EAR ファイルが `/applications` ディレクトリにコピーされます。
2. `server.xml` ファイルにアプリケーション用の新規エントリが次のように作成されます。

```
<application name=<app_name> parent="applicationWithCommonClasses" path=<path_
EARfile> auto-start="true" />
```

各項目の説明：

- `name` 変数は、入力したアプリケーションの名前です。
- `parent` は、オプションの親アプリケーションの名前です。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。この設定は、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` は、EAR ファイルをデプロイするディレクトリおよびファイル名を示します。
- `auto-start` 変数は、OC4J の再起動のたびに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。

「**続行**」ボタンをクリックして、デプロイ・ウィザードの次のステップに進みます。ウィザードは EAR ファイルをアップロードし、アプリケーションを検査します。構成ファイルおよびデプロイメント・ディスクリプタに基づいて、ウィザードは、アプリケーションが必要とする構成画面のみを動的に表示します。これらの画面は、次の項で説明するステップのサブセットです。

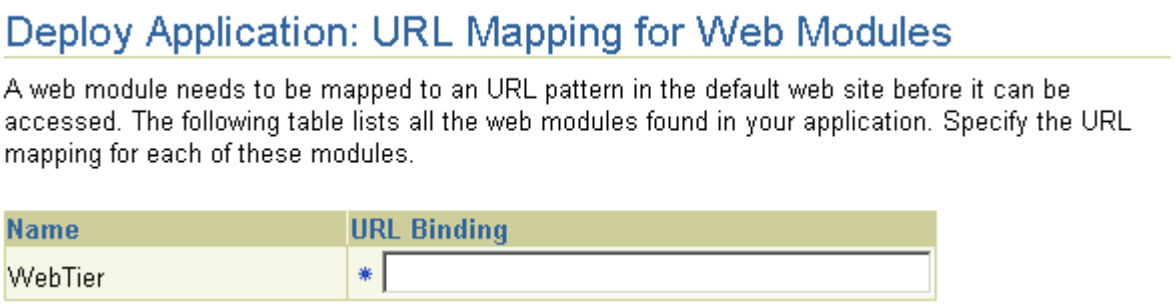
- すべての Web モジュールへの URL マッピング指定
- IIOP スタブの生成
- リソース参照マッピングの指定
- ユーザー・マネージャの指定
- セキュリティ・ロール・マッピングの指定
- Web サービスの公開
- デプロイの確認

すべての Web モジュールへの URL マッピング指定

アプリケーションの Web モジュールをそのサーブレット・コンテキスト用の特定 URL にマッピングします。すべての OC4J サーブレット・コンテキストには、接頭辞としてスラッシュ（/）を付ける必要があります。Web アプリケーションにアクセスするときは、ホスト、ポートおよび Web コンテキストを指定します。

すべての Web モジュールについて、モジュール用の URL マッピングには、この画面でバインドする URL が含まれます。そのため、URL `http://<host>:<port>/url_name` の場合、ウィザードの URL マッピング画面に `/url_name` を指定します。

図 2-6 URL マッピング

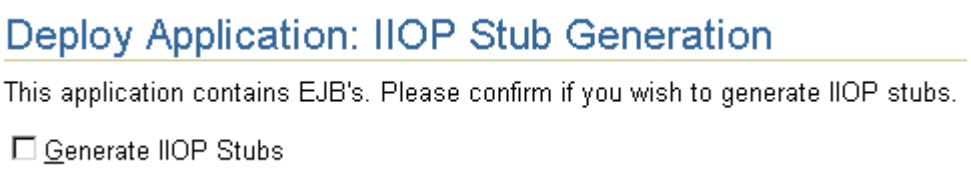


「**次へ**」ボタンをクリックして、デプロイ・ウィザードの次のステップに進みます。

IIOP スタブの生成

「**IIOP スタブの生成**」を選択して、アプリケーション内の EJB で IIOP スタブを生成できます。IIOP スタブの詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』の「相互運用性と RMI トンネリング」を参照してください。

図 2-7 IIOP スタブの生成



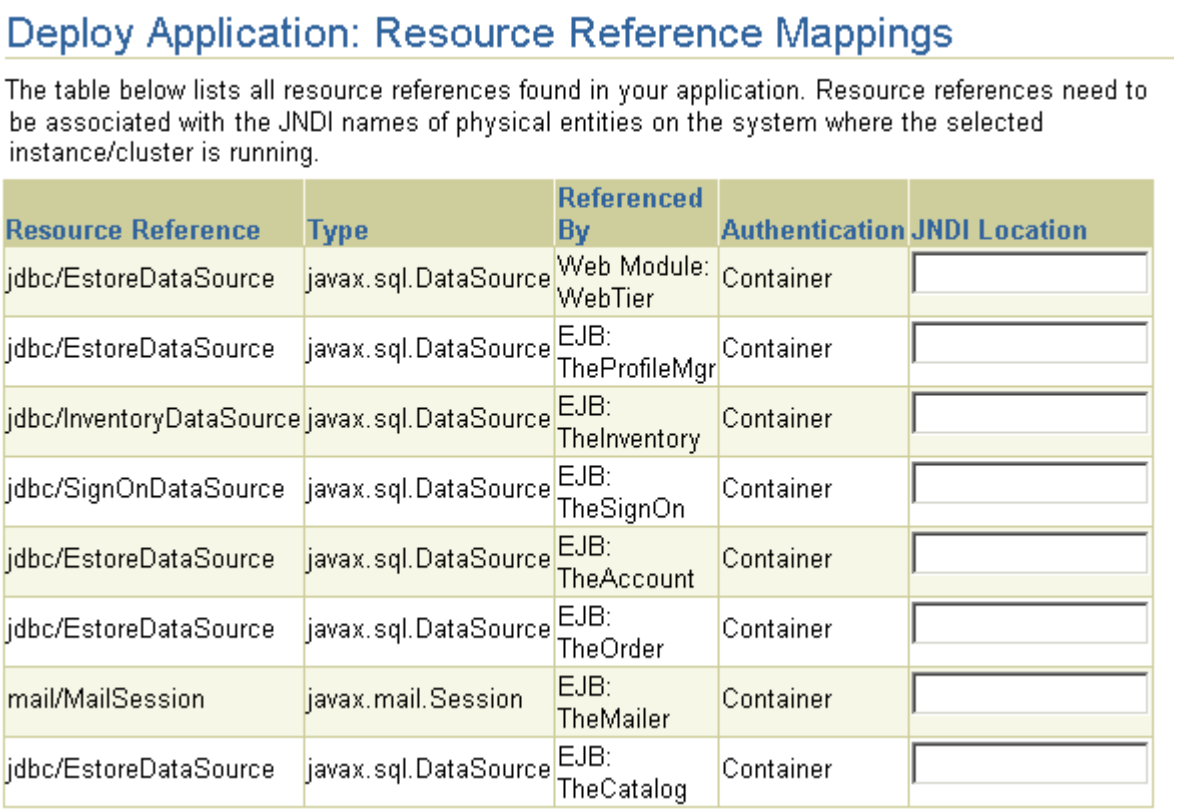
「**次へ**」ボタンをクリックして、デプロイ・ウィザードの次のステップに進みます。

リソース参照マッピングの指定

データ・ソースやメール・キューなどのアプリケーション内の参照リソースを、OC4J コンテナに現在存在する物理エンティティにマッピングします。特定のリソースが必要な場合は、アプリケーションをデプロイする前に、このステップでリソースとアプリケーションを合致させるために、リソースを OC4J コンテナに追加しておく必要があります。

ほとんどのアプリケーションで、指定する必要があるリソース参照は、データ・ソース JNDI 名です。この画面では、データ・ソース情報を構成せず、すでに構成されているデータ・ソースか今後構成するデータ・ソースの指定のみを行います。アプリケーションで使用するデータ・ソースの JNDI 位置名を指定します。

図 2-8 リソース参照マッピング



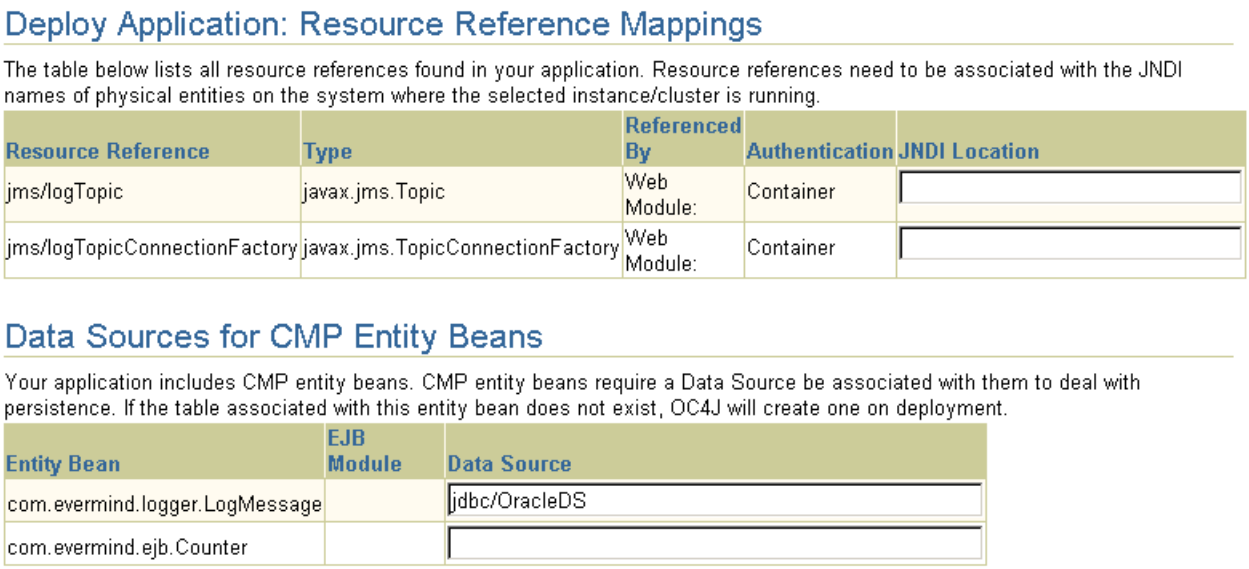
Deploy Application: Resource Reference Mappings

The table below lists all resource references found in your application. Resource references need to be associated with the JNDI names of physical entities on the system where the selected instance/cluster is running.

Resource Reference	Type	Referenced By	Authentication	JNDI Location
jdbc/EstoreDataSource	javax.sql.DataSource	Web Module: WebTier	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheProfileMgr	Container	
jdbc/InventoryDataSource	javax.sql.DataSource	EJB: TheInventory	Container	
jdbc/SignOnDataSource	javax.sql.DataSource	EJB: TheSignOn	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheAccount	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheOrder	Container	
mail/MailSession	javax.mail.Session	EJB: TheMailer	Container	
jdbc/EstoreDataSource	javax.sql.DataSource	EJB: TheCatalog	Container	

EAR ファイル内に MDB がある場合、サブスクリプションまたはトピックの情報の追加が必要になることがあります。CMP Entity Bean の `DataSource` オブジェクトを定義する場合は、これらの `DataSource` オブジェクトの JNDI の場所を追加するオプションがあります。

図 2-9 CMP Entity Bean のリソース参照マッピング



「**次へ**」ボタンをクリックして、デプロイ・ウィザードの次のステップに進みます。

ユーザー・マネージャの指定

セキュリティ用に使用するユーザー・マネージャを指定できます。完全なセキュリティのために、JAZN XML ユーザー・マネージャを選択することをお薦めします。

図 2-10 ユーザー・マネージャの選択

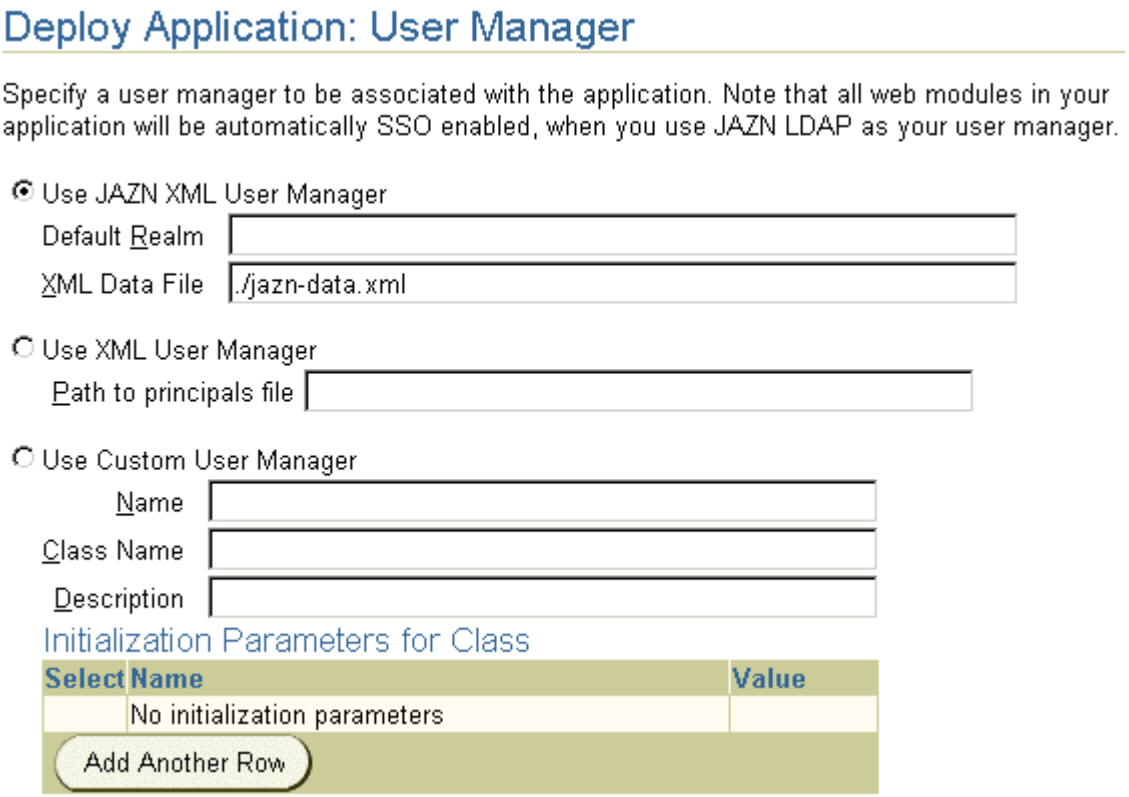


図 2-10 に示すように、事前にユーザー・マネージャを設定および構成しておく必要があります。多くの場合、エントリには、セキュリティのロール、ユーザーおよびグループをセキュリティ・マッピング用に指定する XML ファイルが必要です。

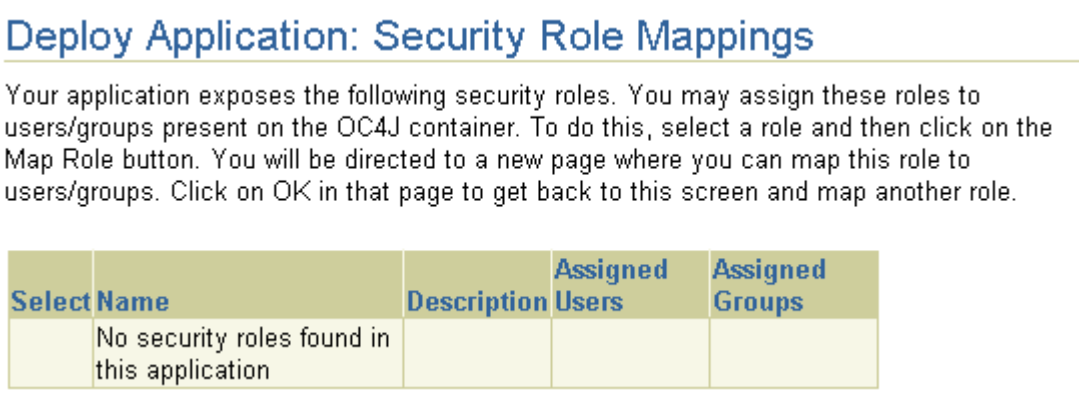
- JAZN XML ユーザー・マネージャ : これは推奨ユーザー・マネージャです。これには、デフォルト・レルムと `jazn-data.xml` ファイルが必要です。
- XML ユーザー・マネージャ : これは、最もセキュアなオプションではありません。これには、`principals.xml` ファイルが必要です。
- ユーザー定義ユーザー・マネージャ : このユーザー・マネージャは、プログラミングされている必要があります。このフィールドには、クラス名を入力します。

セキュリティおよびユーザー・マネージャの詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

セキュリティ・ロール・マッピングの指定

アプリケーションで定義されたセキュリティ・ロールを、既存のユーザーおよびグループにマッピングします。アプリケーション内でセキュリティ・ロールが定義されている場合、このロールをセキュリティ・グループまたはセキュリティ・ロールにマッピングできます。この画面では、セキュリティのグループおよびユーザーは定義しません。ユーザーおよびグループは、ユーザー・マネージャから取得されます。

図 2-11 セキュリティ・ロール・マッピング



「次へ」ボタンをクリックして、デプロイ・ウィザードの次のステップに進みます。

セキュリティ・ロールの詳細は、『Oracle Application Server セキュリティ・ガイド』および『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

Web サービスの公開

アプリケーションで定義された Web サービスを公開します。この機能には、UDDI レジストリが必要です。Web サービスは、コア・インストールではインストールされません。

Web サービスを定義している場合は、次の画面に表示されます。

図 2-12 Web サービス

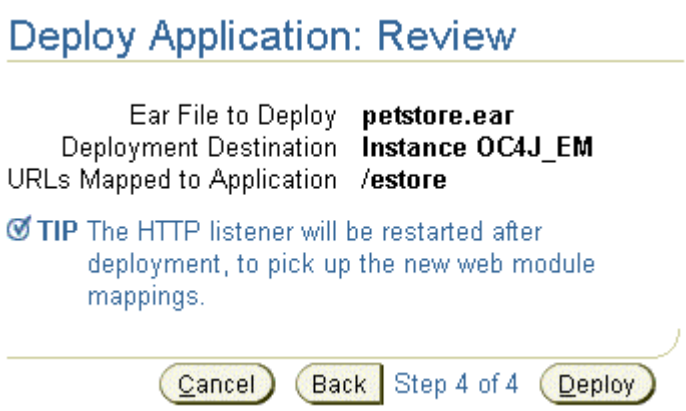
これらの Web サービスを公開するには、「公開」ボタンをクリックします。これにより、Web サービスの公開手順が案内されます。完了したら、この画面に戻ります。

「次へ」ボタンをクリックして、デプロイ・ウィザードの次のステップに進みます。

デプロイの確認

この時点で、アプリケーション・デプロイのモジュールと構成の確認が次のように表示されます。

図 2-13 確認



このアプリケーションをデプロイするには、**「デプロイ」**ボタンをクリックします。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表示されます。

デプロイ後のアプリケーションの変更

フィールドを変更し、構成を追加するには、OC4J ホームページに戻り、「アプリケーション」セクションのアプリケーション名を選択します。これにより、デプロイ済のアプリケーションの詳細が画面に表示されます。

この画面内から、Web モジュールおよび EJB モジュールを表示できます。さらに、アプリケーション固有のプロパティ、リソースおよびセキュリティのオプションを「管理」セクションで追加したり変更したりすることができます。「管理」セクションでは、デプロイ・ウィザードで指定した、アプリケーション固有のデータ・ソースやセキュリティのグループまたはユーザーを追加できます。

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、クライアントからアクセスできなくなります。
- Web アプリケーションのすべてのバインドが、Web モジュールのバインド先のすべての Web サイトから削除されます。
- アプリケーション・ファイルが `applications/` および `application-deployments/` ディレクトリの両方から削除されます。
- Oracle Application Server 環境では、Oracle HTTP Server が再起動され、アプリケーション用に定義されているマウント・ポイントが削除されます。これにより、既存の HTTP セッションが失われます。

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。クラスタ環境でのこの問題の対処方法は、『Oracle Application Server 高可用性ガイド』を参照してください。

デプロイ時の動作

OC4J とアプリケーションのデプロイに習熟したら、OC4J の動作を理解する必要があります。次の項を読むと、これらのタスクを理解できます。

- デプロイ時の OC4J のタスク
- デプロイ済 J2EE アプリケーションの構成の検証

デプロイ時の OC4J のタスク

アプリケーションをデプロイすると、次のようになります。

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・モジュールのモジュール・タイプごとに、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なかった場合、OC4J ではデフォルトを書き込んだディスクプリプタが提供されます。OC4J 固有のデプロイメント・ディスクリプタを指定した場合は、OC4J により要素が追加されることがあります。
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。マスターのディレクトリは、OC4J ホームページから「サーバー・プロパティ」で変更できます。「一般」セクションで、「アプリケーション・ディレクトリ」フィールドを新しいマスター・ディレクトリの場所に変更します。ディレクトリの場所は、`j2ee/home/config` に対する相対ディレクトリです。

   注意： この EAR ファイルを `applications/` ディレクトリから削除せずにデプロイするたびに、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けられます。EAR ファイルが上書きされることはありません。かわりに、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を通知し、EAR を再デプロイします。
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表記して、`server.xml` ファイルを更新します。

デプロイ済 J2EE アプリケーションの構成の検証

デプロイ後は、次のように server.xml ファイルと default-web-site.xml ファイルのアプリケーション構成を確認できます。

- server.xml には、既存のアプリケーションごとに、<application name=... path=... auto-start="true" /> エントリを含む行があります。auto-start 属性は、OC4J 起動時にこのアプリケーションを自動起動するかどうかを指定します。path 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です。詳細は、2-11 ページの「基本デプロイ」、または 3-22 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。
- default-web-site.xml には、OC4J の起動時に Web サイトにバインドされる Web アプリケーションごとに、<web-app...> エントリが存在します。name 属性は WAR ファイル名（.WAR 拡張子を除いた部分）であるため、J2EE アプリケーションに含まれる各 WAR ファイルが 1 行ごとに記述されています。

  WAR ファイル内の Web アプリケーションのバインドごとに、次の行が追加されています。

  ```
  <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、アプリケーションのルート・コンテキストを Web サイトから定義します。たとえば、Web を http://<ohs_host>:7777/j2ee と定義した場合、アプリケーションを起動するには、ブラウザで http://<ohs_host>:7777/j2ee/myapp を指定します。

  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アクセスすると、参照と同時にクライアントが失敗します。

デプロイ・エラーからのリカバリ

デプロイ処理がなんらかの理由で中断した場合は、デフォルトが /var/tmp の一時ディレクトリをクリーンアップする必要があります。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確認してください。

一時ディレクトリを変更するには、OC4J プロセス用のコマンドライン・オプションを java.io.tmpdir=<new_tmp_dir> に設定します。このコマンドライン・オプションは、「サーバー・プロパティ」ページで設定できます。OC4J ホームページにドリルダウンします。「管理」セクションまでスクロールします。**「サーバー・プロパティ」**を選択します。このページで、「コマンドライン・オプション」セクションまでスクロールして、java.io.tmpdir 変数定義を OC4J オプション行に追加します。すべての OC4J 新規プロセスは、このプロパティで起動します。

Web アプリケーションのアンデプロイ

J2EE Web アプリケーションを OC4J Web サーバーから削除するには、OC4J ホームページの「アプリケーション」セクションでアプリケーションを選択し（図 2-2 を参照）、**「アンデプロイ」**ボタンをクリックします。このコマンドによってデプロイされた 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。

注意： アプリケーションのアンデプロイは、コマンドで実行することもできます。手順は、『Distributed Configuration Management 管理者ガイド』を参照してください。

3

高度な構成および開発

第 2 章「構成およびデプロイ」では、J2EE アプリケーションの基本的な構成、開発およびデプロイを説明します。この章では、グローバルな J2EE サービス構成と高度な J2EE アプリケーション構成の両方について説明します。

この章には、次の項目が含まれています。

- Enterprise Manager を使用した OC4J の構成
- OC4J および J2EE の XML ファイルの概要
- ライブラリの共有
- OC4J リスナーの理解および構成
- 別の Web コンテキストによる Oracle HTTP Server の構成
- ディレクトリ内での構築およびデプロイ
- 起動クラスおよび停止クラスの開発
- パフォーマンス・オプションの設定
- OC4J ロギングの有効化
- OC4J のデバッグ

Enterprise Manager を使用した OC4J の構成

J2EE サービス、J2EE アプリケーションおよび Oracle Application Server クラスタは、Enterprise Manager で構成できます。その一部は OC4J インスタンス・レベルで構成され、インスタンス内でデプロイされたすべてのアプリケーションに対してグローバルな構成となります。その他はアプリケーション・レベルで構成されるため、このタイプの構成はローカルで、そのアプリケーションのみに適用されます。

次の項では、Enterprise Manager での OC4J に対する高度な構成の概要を説明します。

- OC4J インスタンス・レベル構成
- アプリケーション・レベル構成

OC4J インスタンス・レベル構成

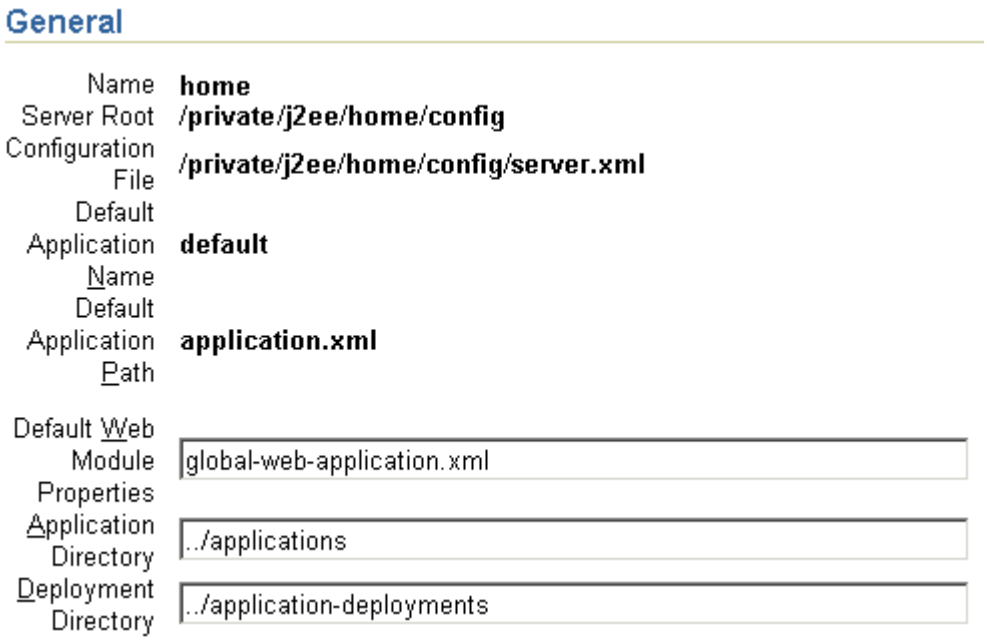
OC4J ホームページから、「管理」ページにドリルダウンすることで、すべてのアプリケーションに適用されるグローバル・サービスを構成できます。「管理」ページでは、次の構成が可能です。

- サーバー・プロパティの構成
- Web サイトの構成
- JSP コンテナ・パラメータの構成
- レプリケーション・パラメータの構成
- XML ファイルによる高度な構成
- データ・ソースの構成
- セキュリティの構成
- JMS の構成
- グローバル Web アプリケーションのパラメータの構成
- RMI の構成

サーバー・プロパティの構成

OC4J プロパティを構成するには、「インスタンス・プロパティ」セクションまでスクロールして、「サーバー・プロパティ」を選択します。このページの「一般」セクションに、次のフィールドが表示されます。

図 3-1「インスタンス・プロパティ」ページの「一般」セクション



デフォルトのサーバー・プロパティに関する次の情報が、ページの上半分に表示されます。

- デフォルト・アプリケーション : デフォルト・アプリケーションは、大部分のデプロイ済アプリケーションがその親として使用したものです。したがって、これらのデプロイ済アプリケーションでは、デフォルト・アプリケーション内のクラスを参照できます。親アプリケーションの詳細は、2-12 ページの「アプリケーションの選択」を参照してください。
- デフォルト・アプリケーションのパス : 各 EAR ファイルに含まれる `application.xml` とは別に、`application.xml` というファイルが存在します。この `application.xml` ファイルは、グローバル `application.xml` ファイルと呼ばれます。このファイルは、OC4J インスタンス内にデプロイされているすべてのアプリケーションで使用するプロパティを定義します。

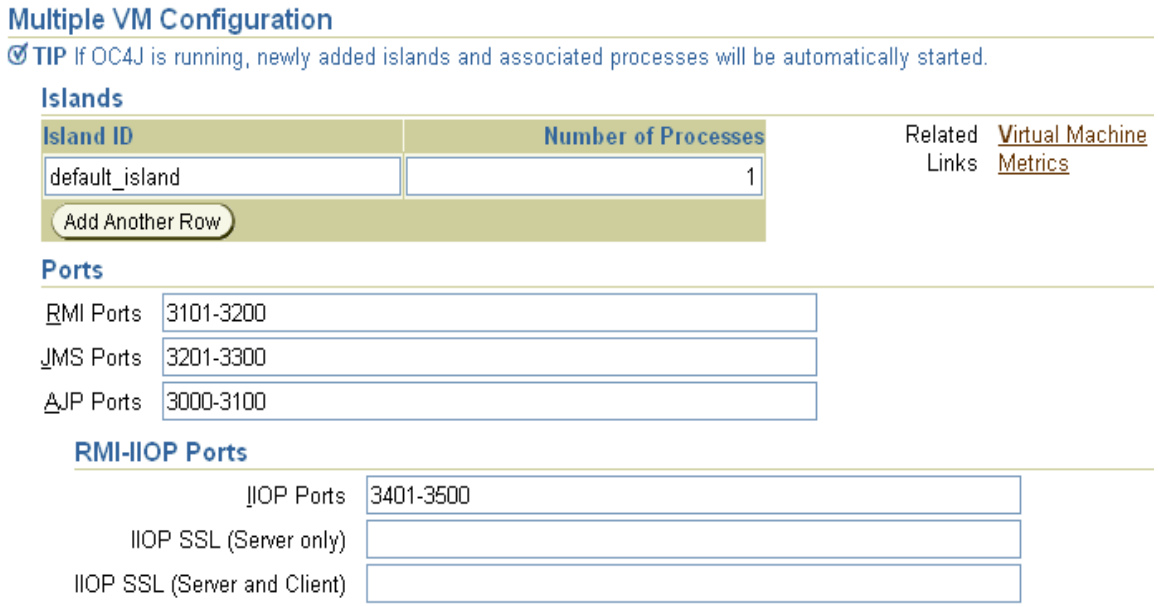
このセクションで、OC4J サーバーのデフォルトを変更できます。これらのデフォルトは次のとおりです。

- デフォルト Web モジュール・プロパティ : これらは、`global-web-application.xml` という XML ファイルで指定されています。別の XML ファイルを参照する場合は、ここでファイルの名前を変更します。ただし、そのファイルは、Oracle 指定の DTD に準拠する必要があります。ディレクトリは、`j2ee/home/config` に対する相対ディレクトリです。

  このファイルに含まれる要素を変更するには、「Web サイト・プロパティ」または「拡張プロパティ」セクションのいずれかのエントリを更新します。詳細は、3-5 ページの「Web サイトの構成」と 3-15 ページの「デプロイ済アプリケーション EAR ファイルに含まれる XML ファイルの変更」を参照してください。
- アプリケーション・ディレクトリ、デプロイ・ディレクトリ : デプロイされているアプリケーションのマスター EAR ファイルを配置するフォルト・ディレクトリは、`/applications` ディレクトリです。デフォルト・ディレクトリは、変更されたモジュール・デプロイメント・ディスクリプタが追加のデフォルトとともに OC4J により配置される場所です。現在、この場所は、`/application-deployments` ディレクトリ下です。デフォルト・ディレクトリの場所は、このフィールドで変更できます。ディレクトリは、`j2ee/home/config` に対する相対ディレクトリです。このディレクトリの使用方法は、2-19 ページの「デプロイ時の動作」を参照してください。

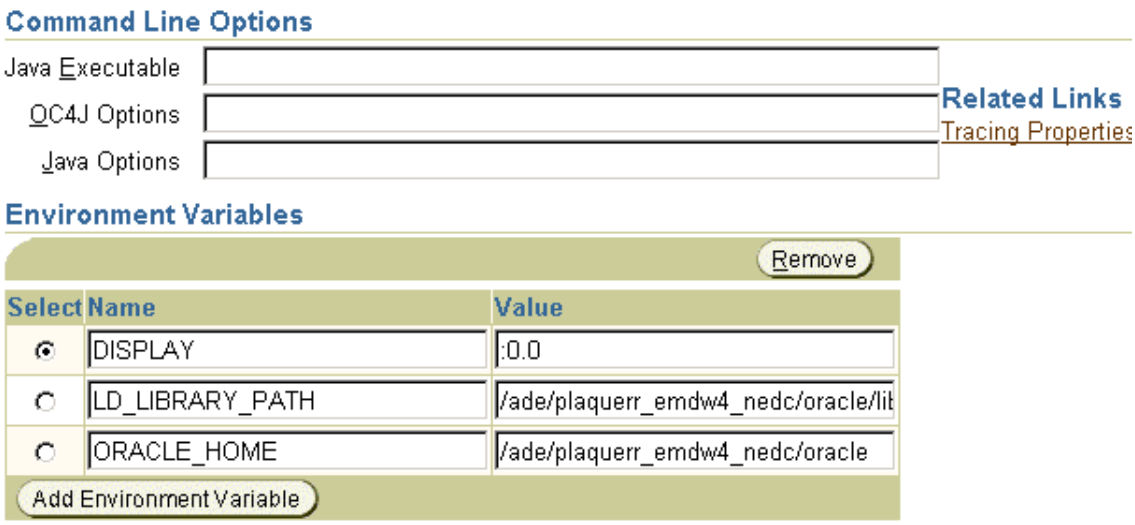
次の「複数仮想マシン構成」セクションは、クラスタ構成の一部として使用されます。次に、各フィールドの意味を詳しく説明します。ただし、このセクションを使用するクラスタの構成方法のコンテキストについては、第 4 章「OC4J のクラスタリング」で詳細を説明します。

図 3-2 クラスタリングとポート



- アイランド : クラスタ内のアイランド数を指定します。各アイランドは、「行の追加」ボタンをクリックすると作成されます。「アイランド ID」フィールドに、各アイランドの名前を指定します。「プロセス数」フィールドに、各アイランド内で起動される OC4J プロセスの数を指定します。クラスタリング用のアイランド構成の詳細は、4-8 ページの「OC4J クラスタの設定」を参照してください。
- ポート : このセクションでは、RMI、JMS、AJP および IIOP のポート範囲を構成します。これらのサービスの詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

図 3-3 コマンドライン・オプションと環境変数



- コマンドライン・オプション : このセクションでは、次の項目を構成します。
  - javac など、使用する必要のある Java 実行コマンド
  - 新規 OC4J プロセスの起動時に指定する OC4J オプション
  - java 実行時に指定する Java オプション

  - OC4J システム・プロパティ

    コマンドライン・オプションおよびシステム・プロパティのリストは、B-29 ページの「OC4J のコマンドライン・オプションおよびシステム・プロパティ」を参照してください。

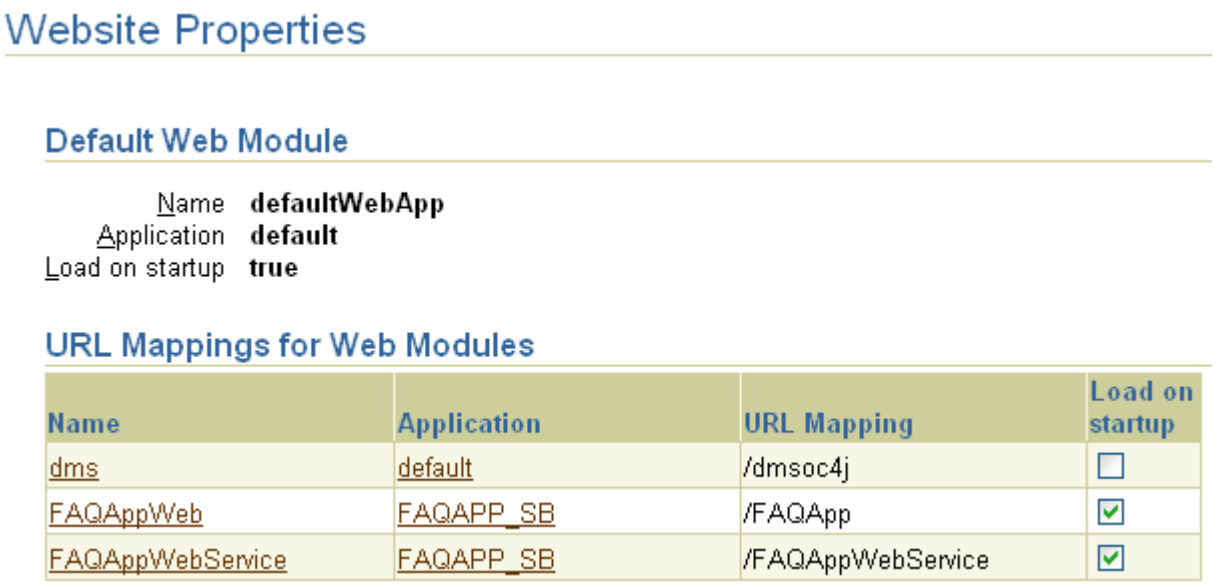
- 環境変数 : このセクションでは、OC4J の環境変数を構成します。新しい環境変数を追加するには、**「環境変数の追加」**をクリックします。新しい行が追加され、左側の列では変数名を、右側の列では値を定義できます。

Web サイトの構成

Web サイトを構成するには、「管理」ページの「インスタンス・プロパティ」列で「Web サイト・プロパティ」を選択します。

Web サイト・ページには、2 つのセクションがあります。第 1 セクションには、デフォルト Web アプリケーションが表示されます。第 2 セクションの「Web モジュールの URL マッピング」には、各 Web アプリケーションが起動時にロードされるかどうかを指定します。これらのパラメータについては、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』で詳しく説明されています。パラメータの格納先は `default-web-site.xml` ファイルです。

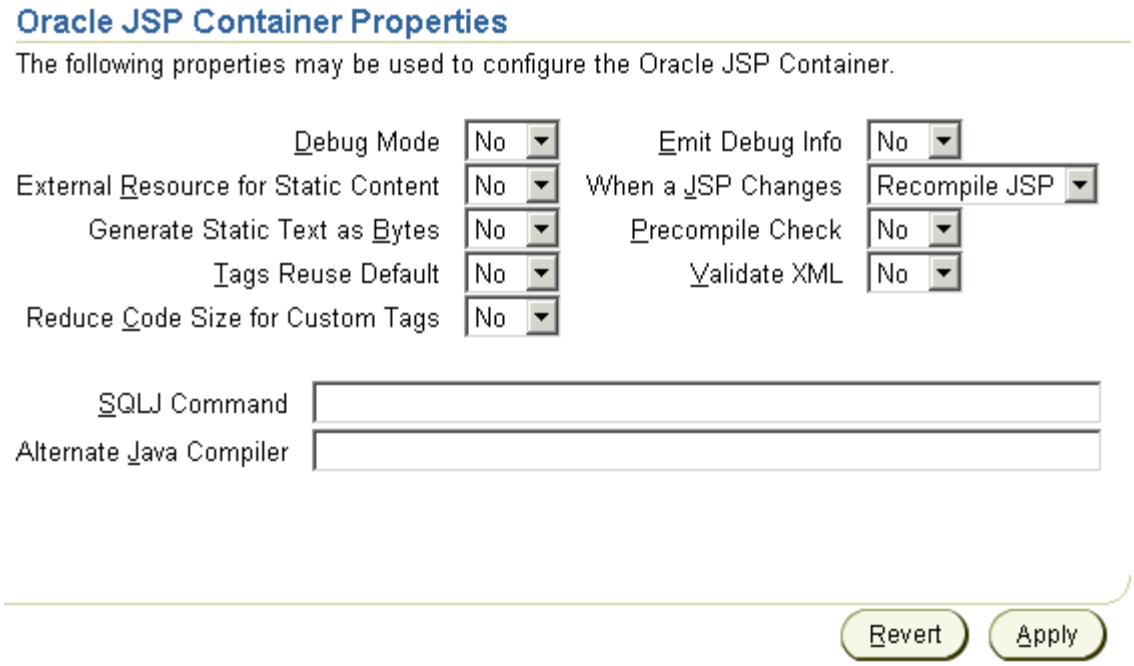
図 3-4 Web サイト・プロパティ



JSP コンテナ・パラメータの構成

グローバル JSP コンテナ・パラメータを構成できます。この OC4J インスタンスにデプロイされているすべての JSP に適用されます。JSP コンテナ・パラメータを構成するには、「管理」ページの「インスタンス・プロパティ」列で「JSP コンテナのプロパティ」を選択します。これにより、次のページが表示されます。

図 3-5 Oracle JSP コンテナのプロパティ



ここに示したプロパティのほとんどは、『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』の第 3 章で説明されています。これらのプロパティは、`<servlet>` 要素内の `global-web-application.xml` ファイルで指定できます。

レプリケーション・パラメータの構成

サーブレットまたは EJB をクラスタリングする場合、レプリケーション・パラメータを構成する必要があります。詳細は、4-8 ページの「OC4J インスタンスの設定」を参照してください。

XML ファイルによる高度な構成

OC4J リリース 1.0.2.2 では、OC4J サーバーとデプロイされているすべてのアプリケーション・パラメータを XML ファイルによって構成しました。この方法は、OC4J サーバーが単一ノードに存在し、高可用性とクラスタ管理が必要でなかったため、うまく機能していました。しかし、OC4J と Oracle Application Server が統合され、クラスタリングと高可用性のオプションによりエンタープライズ管理機能が高まり、あらゆる構成を Enterprise Manager によって行う必要があります。

「管理」ページから「拡張プロパティ」を選択すると、OC4J サーバーの XML ファイルを変更できます。これらのファイルには、サーバーとそのサービスを構成する XML ファイルが含まれます。このグループのファイルは、`server.xml`、`global-web-application.xml`、`rmi.xml`、`jms.xml` および `default-web-site.xml` です。これらの XML ファイルは、OC4J ホームページの「拡張プロパティ」ページで変更します。

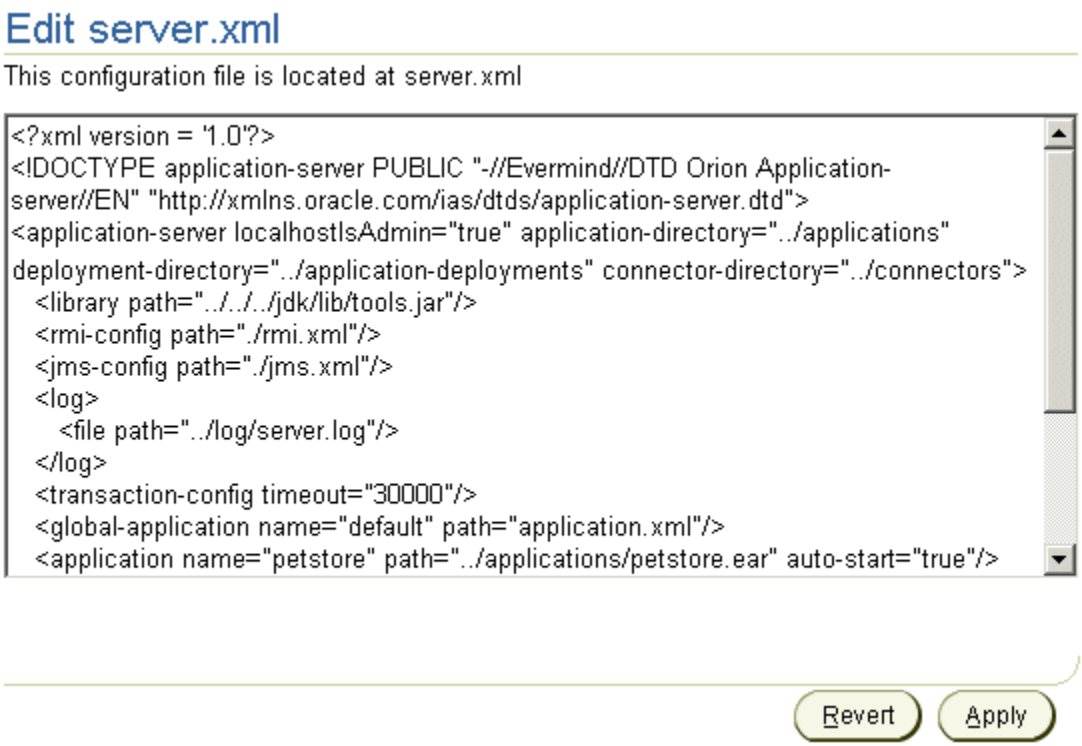
他の XML 構成ファイルは、Oracle9*i*AS コンソールの他の領域で変更できます。

- グローバル・アプリケーション XML ファイル : OC4J インスタンスにデプロイされているすべてのアプリケーションに適用される XML ファイルが含まれます。グローバル・アプリケーション用の `application.xml`、`data-sources.xml`、セキュリティ XML ファイルおよび `oc4j-connectors.xml` が該当します。これらの XML ファイルを変更するには、OC4J ホームページから「アプリケーション」を選択します。「アプリケーション」ページでデフォルトを選択します。デフォルト・アプリケーション・ページで、「管理」セクションまでスクロールして、「拡張プロパティ」を選択します。

- ローカル・アプリケーション XML ファイル : アプリケーション全体を構成する XML ファイルを変更できます。ローカル・データ・ソース、ローカル・セキュリティおよび OC4J 固有のアプリケーション構成が含まれます。これらの XML ファイルには、`data-sources.xml`、`orion-application.xml` およびセキュリティ XML ファイルがあります。これらのファイルを変更するには、「アプリケーション」ページの「デプロイ済アプリケーション」セクションから特定アプリケーションまでドリルダウンします。指定したアプリケーション画面で、「管理」セクションまでスクロールして、「拡張プロパティ」を選択します。
- アプリケーション・モジュール XML ファイル : EAR または WAR ファイルをデプロイするときに、`web.xml`、`orion-web.xml`、`ejb-jar.xml` および `orion-ejb-jar.xml` などのモジュール・デプロイメント・ディスクリプタを指定しています。パラメータの変更は、OC4J 固有の XML ファイル（`orion-xxx.xml`）でのみ可能です。`web.xml` または `ejb-jar.xml` などの J2EE XML ファイルは、変更できません。これらの XML ファイルの変更については、3-15 ページの「デプロイ済アプリケーション EAR ファイルに含まれる XML ファイルの変更」を参照してください。

例として、`server.xml` ページを示します。XML 要素は、手入力で編集できます。

図 3-6 server.XML パラメータの編集



OC4J XML ファイルの概要は、これらのファイルとそれぞれの関係について説明する 3-15 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。各ファイル内の要素については、OC4J ドキュメントのその他のマニュアルで説明されています。

データ・ソースの構成

グローバルまたはローカルのデータ・ソースを構成できます。グローバル・データ・ソースは、この OC4J インスタンスにデプロイされたすべてのアプリケーションで使用可能です。ローカル・データ・ソースは、デプロイされたアプリケーション内に構成され、そのアプリケーションでのみ使用可能です。

データ・ソースと `data-sources.xml` ファイル内の要素の構成方法の詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

グローバル・データ・ソースを構成するには、OC4J ホームページから次のいずれかを選択します。

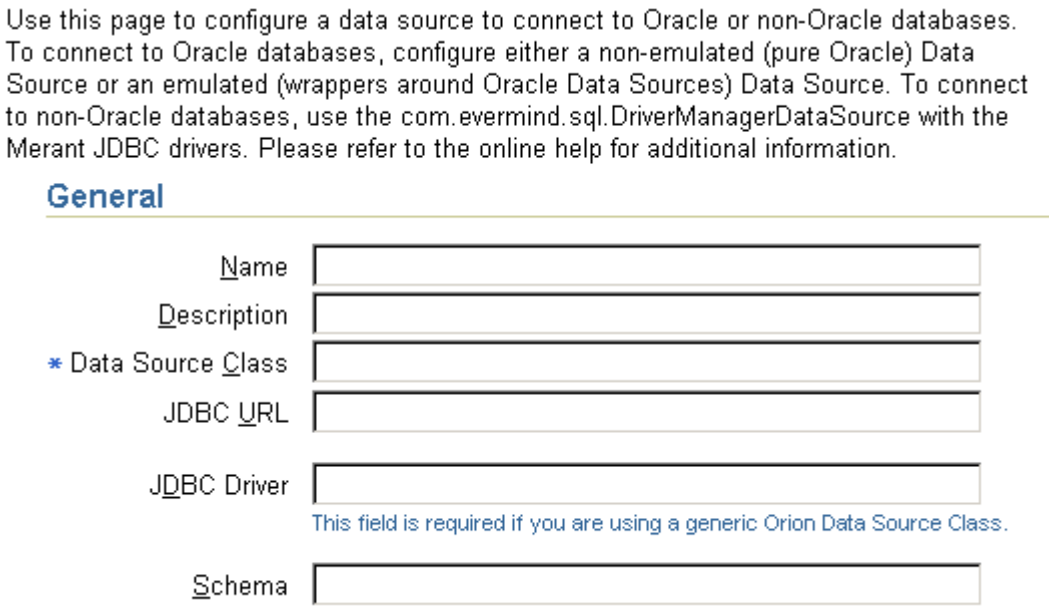
- 「管理」ページの「アプリケーション・デフォルト」列のデータ・ソース : このページでは、一度に 1 フィールドずつデータ・ソース定義を追加できます。このページの詳細は、3-8 ページの「データ・ソース・フィールド・ページ」を参照してください。
- デフォルト・アプリケーション・ページから : GUI を介して、または `data-sources.xml` ファイルに直接アクセスすることで、データ・ソースを変更または追加できます。
  1. OC4J ホームページから「アプリケーション」を選択します。
  2. 「アプリケーション」ページで、「デフォルト・アプリケーション名」の横にあるデフォルトを選択します。
  3. デフォルト・アプリケーション・ページで、「管理」セクションまでスクロールして、次のいずれかを実行します。
     - 「リソース」列で「データ・ソース」を選択します。これにより、GUI フォームを使用してデータ・ソースを追加または変更できます。
     - 「プロパティ」列で「拡張プロパティ」を選択します。このページで `data-sources.xml` を選択します。これにより、XML 定義を使用してデータ・ソースを追加できます。XML 定義がすでにある場合、これは便利な方法です。構成済のデータ・ソースでコピーのみを行います。

ローカル・データ・ソースを構成するには、アプリケーション・ページから同様に選択します。このデータ・ソースが使用される特定アプリケーションにドリルダウンする必要があります。アプリケーション・ページで、「リソース」列の「データ・ソース」を選択します。3-8 ページの「データ・ソース・フィールド・ページ」で説明しているデータ・ソース・フィールド・ページと同じページが表示されます。

データ・ソース・フィールド・ページ 新しいデータ・ソースを構成するには、**「データ・ソースの追加」**をクリックします。これにより、データ・ソースに関するすべての構成の詳細を入力するページが表示されます。このページは、4 つのセクションに分かれています。

図 3-7 は「一般」セクションです。

図 3-7 データ・ソース定義の「一般」セクション

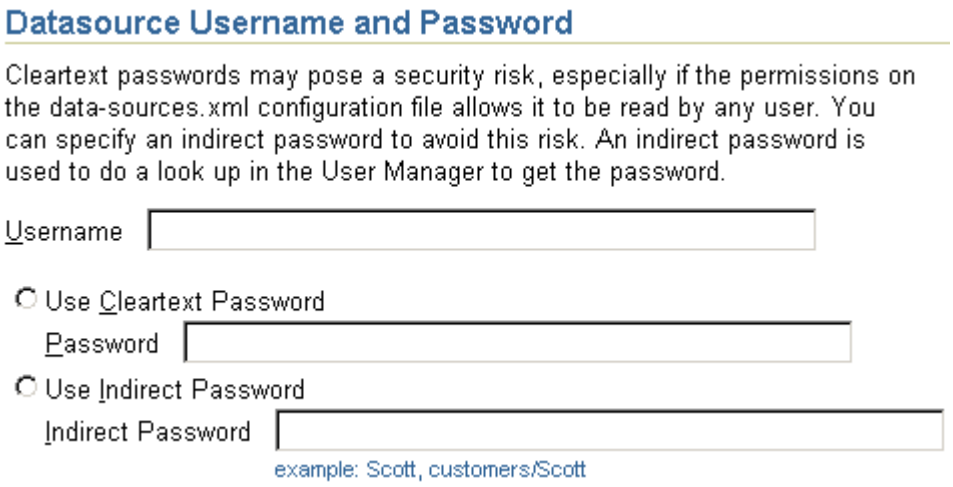


「一般」セクションでは、データ・ソースに関する次のパラメータを定義できます。

- 名前 : ユーザーが定義する、データ・ソースを識別するための名前。
- 説明 : ユーザーが定義する、データ・ソースの説明。
- データ・ソース・クラス : データ・ソースがインスタンス化される `com.evermind.sql.DriverManagerDataSource` などのクラス。
- JDBC URL: このデータ・ソースに相当するデータベース・サービスへの URL。たとえば、Oracle Thin Driver を使用している場合、URL は、`jdbc:oracle:thin:@my-lap:1521:<service_name>` のようになります。
- JDBC ドライバ : 使用する JDBC ドライバ。JDBC ドライバの例として、`oracle.jdbc.driver.OracleDriver` があげられます。
- スキーマ : これはオプションのパラメータです。特定データベース用の Java からデータベースへのマッピングを含むファイル名を入力します。

図 3-8 はユーザー名とパスワードです。

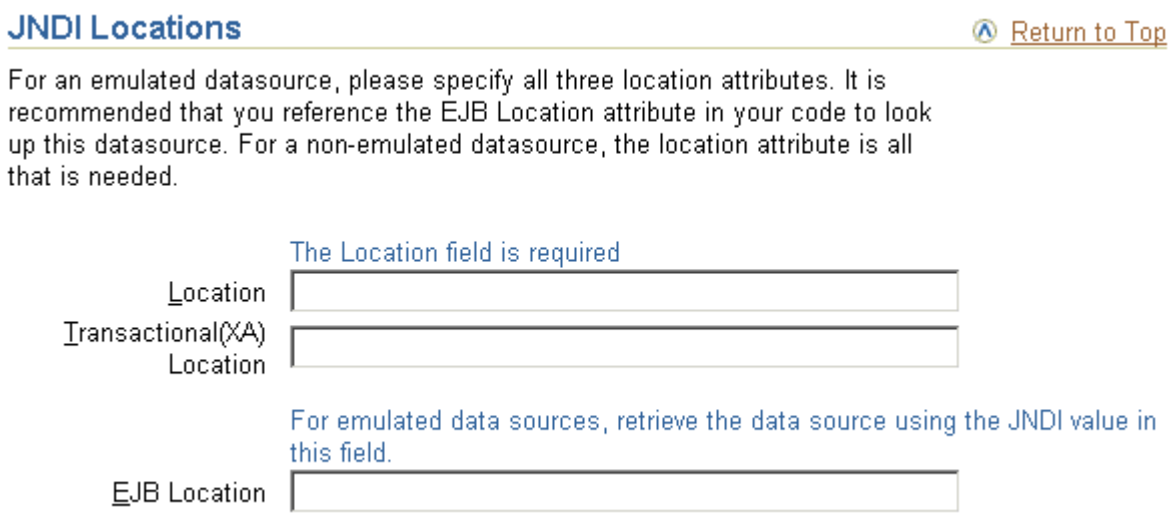
図 3-8 ユーザー名とパスワード



ユーザー名 / パスワード : このデータ・ソースに相当するデータベースへの認証に使用するユーザー名とパスワード。パスワードは平文で入力することも、間接的パスワードにユーザー名を指定することもできます。詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

図 3-9 は「JNDI ロケーション」セクションです。

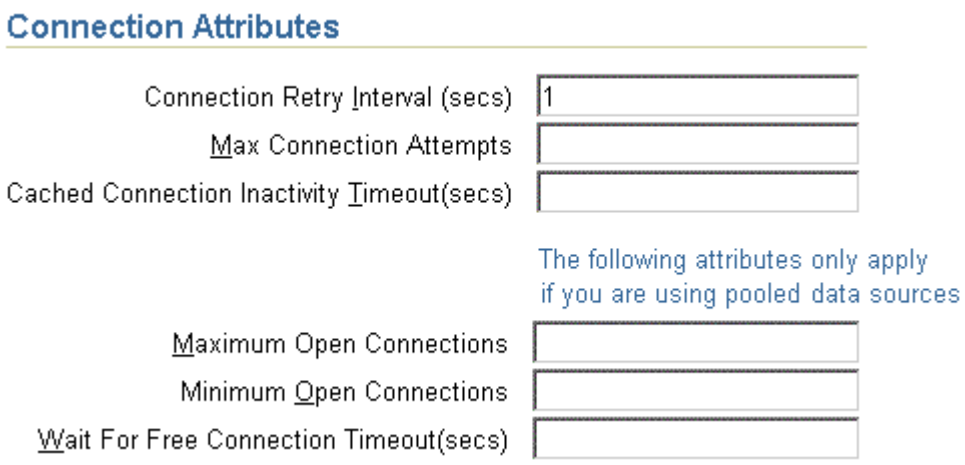
図 3-9 JNDI ロケーション



「JNDI ロケーション」セクションで、データ・ソースがバインドされる JNDI ロケーションの文字列を定義できます。この JNDI ロケーションは、このデータ・ソースを検索するために JNDI 参照内で使用されます。

図 3-10 は「接続の属性」セクションです。

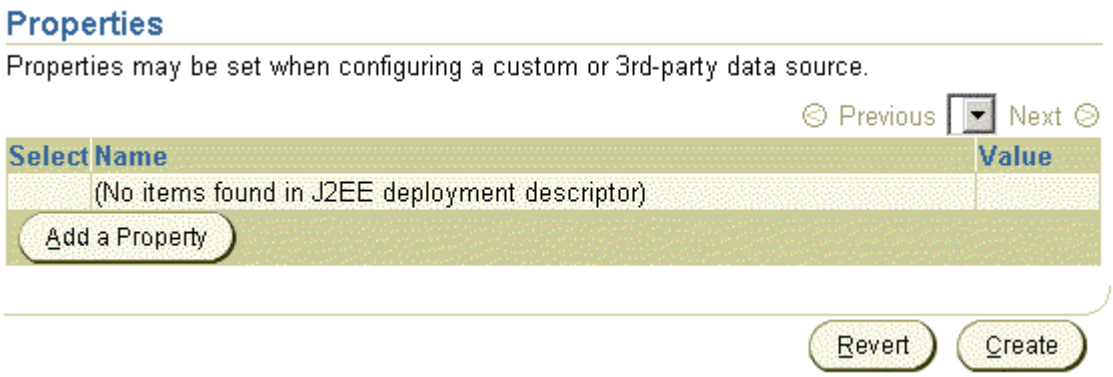
図 3-10 接続の属性



このセクションでは、再試行間隔、プーリング・パラメータ、タイムアウト・パラメータ、最大試行回数パラメータなどの接続チューニング・パラメータを変更します。

図 3-11 はデータ・ソースの「プロパティ」セクションです。

図 3-11 プロパティ



データ・ソースがサード・パーティのデータ・ソースである場合、特定のプロパティの設定が必要になることがあります。そのプロパティは、サード・パーティのドキュメントに定義されています。さらに、2 フェーズ・コミット・コーディネータの JTA トランザクションについて、プロパティを設定する必要があります。

セキュリティの構成

ユーザー・マネージャでは、ユーザー名とパスワードを使用して、ユーザー・リポジトリの情報に基づきユーザーの ID を検証します。使用する認証のタイプは、ユーザー・マネージャにより定義されます。この定義には、ユーザー、グループまたはロールの定義も含まれています。デフォルトのユーザー・マネージャは `JAZNUserManager` です。すべてのアプリケーションまたは特定のアプリケーション用のユーザー・マネージャを指定することができます。

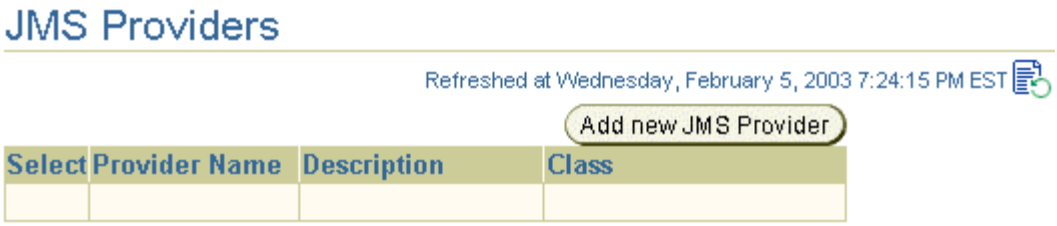
ユーザー・マネージャを含めて、OC4J セキュリティの詳細は、『Oracle Application Server Containers for J2EE セキュリティ・ガイド』を参照してください。

JMS の構成

JMS は、JMS のセクション内で構成することも、`jms.xml` ファイル内で直接構成することもできます。

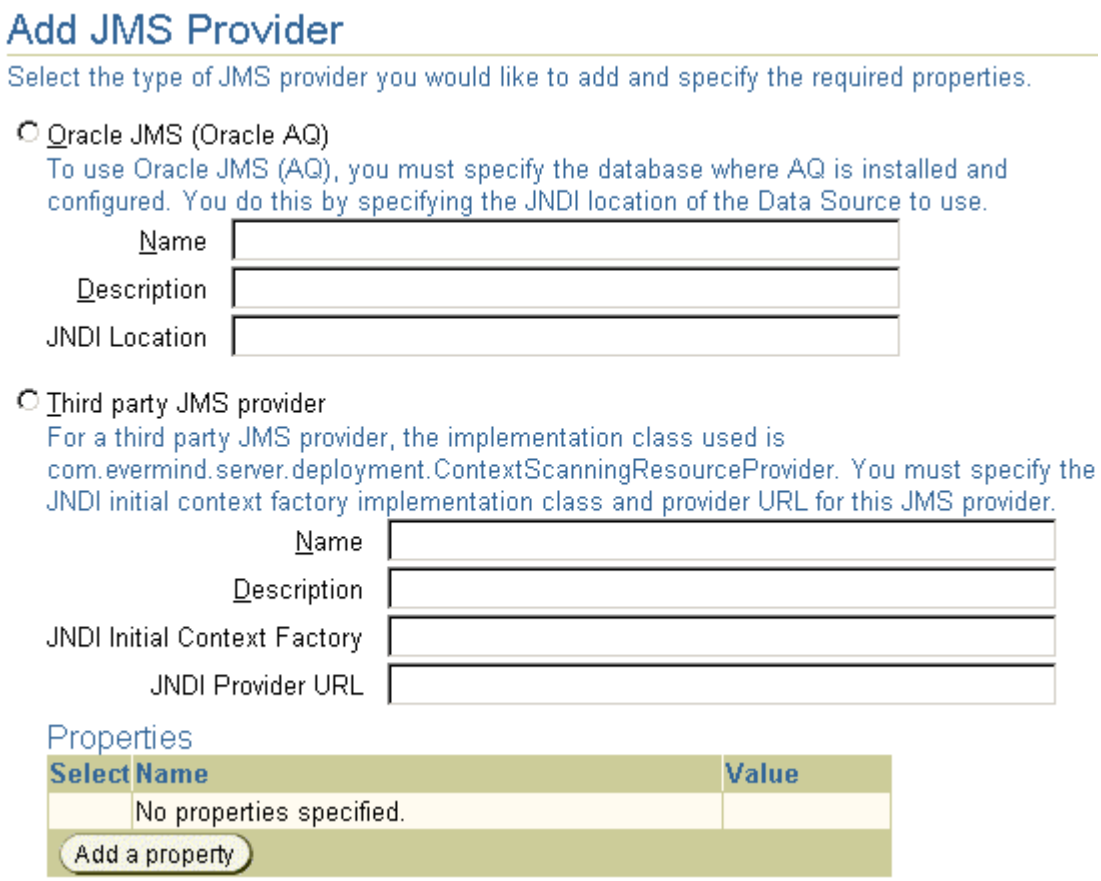
JMS セクションの編集 Oracle JMS またはサード・パーティの JMS プロバイダを追加するには、「管理」ページの「アプリケーション・デフォルト」列で「JMS プロバイダ」を選択します。これにより、次のページが表示されます。

図 3-12 JMS プロバイダの定義



各 JMS プロバイダを構成するために**「新規 JMS プロバイダの追加」**ボタンをクリックすると、次のページが表示されます。

図 3-13 JMS プロバイダの追加



このページでは、Oracle JMS またはサード・パーティの JMS プロバイダのいずれかを構成できます。OC4J がインストールされていると、トピックおよびキュー以外は、常に OracleAS JMS が指定され、事前に構成されています。

JMS プロバイダのタイプを選択した場合、次の情報を指定する必要があります。

- Oracle JMS (Oracle AQ): Oracle JMS がインストールおよび構成されているデータベースのデータ・ソース名および JNDI ロケーションを指定します。
- サード・パーティの JMS プロバイダ : サード・パーティ・プロバイダの名前、JNDI 初期コンテキスト・ファクトリ・クラスおよび JNDI URL を指定します。`java.naming.factory.initial` および `java.naming.provider.url` など、この JMS プロバイダの JNDI プロパティを追加するには、**「プロパティの追加」**をクリックします。各 JNDI プロパティの名前と値の追加用に、1 行ずつ追加されます。

ここで構成されるのはプロバイダのみで、`Destination` オブジェクト（トピック、キューおよびサブスクリプション）は構成されません。JMS プロバイダの詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

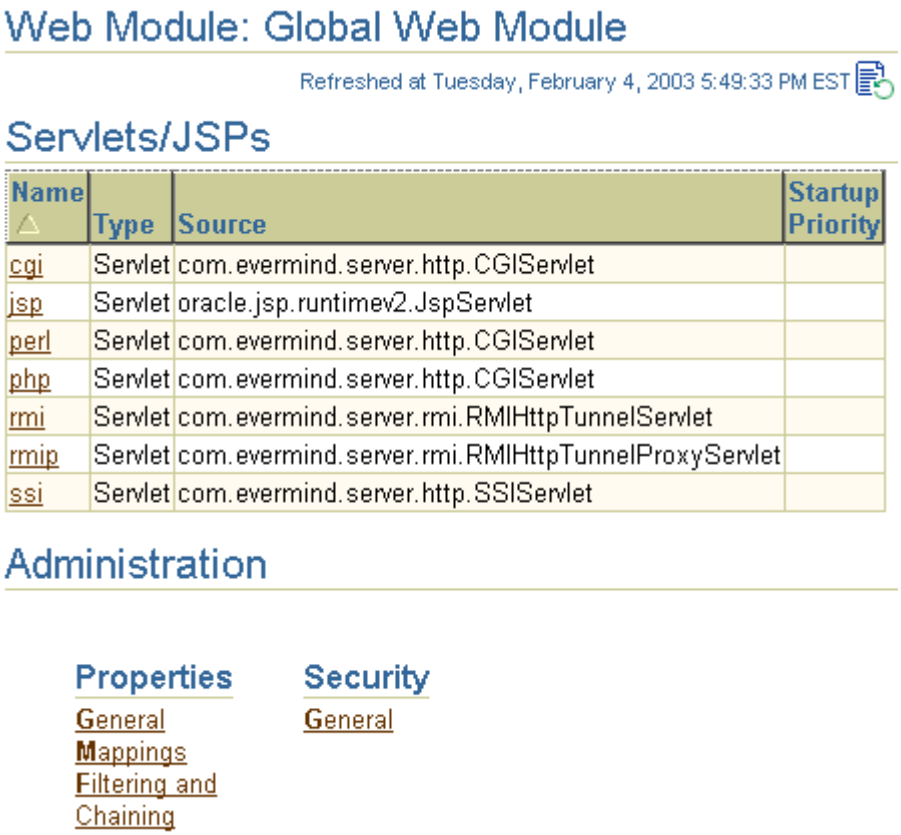
特定のアプリケーション専用の JMS プロバイダを構成するには、「アプリケーション」ページからアプリケーションを選択して、「リソース」列までスクロールし、「JMS プロバイダ」を選択します。表示される画面は、デフォルトの JMS プロバイダの場合と同じです。

JMS XML ファイルの編集 OracleAS JMS 用のキューおよびトピックを追加するには、次のように `jms.xml` ファイルを直接編集できます。「管理」ページの「インスタンス・プロパティ」列で「拡張サーバー・プロパティ」セクションを選択します。このセクションで `jms.xml` を選択し、XML ファイルを修正します。『Oracle Application Server Containers for J2EE サービス・ガイド』で、`jms.xml` ファイル内の要素の説明を参照してください。

グローバル Web アプリケーションのパラメータの構成

デプロイされたすべての Web アプリケーションに適用する Web パラメータを構成するには、「管理」ページの「アプリケーション・デフォルト」列で「グローバル Web モジュール」を選択します。これにより、次のページが表示されます。

図 3-14 Web モジュールの定義



Web モジュール用に定義できるパラメータのタイプは、マッピング、フィルタリング、チェーン、環境およびセキュリティに関するものです。これらのパラメータを変更するには、「プロパティ」列と「セキュリティ」列に表示された各リンクをドリルダウンします。これらのパラメータの詳細は、『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。これらのパラメータは、`global-web-application.xml` および `orion-web.xml` ファイルに格納されます。このマニュアルでは、これらのファイルの要素について説明します。

RMI の構成

RMI は、XML 定義内でのみ定義可能です。`rmi.xml` ファイルを編集するには、「管理」ページの「インスタンス・プロパティ」列で「拡張プロパティ」を選択します。このセクションで `rmi.xml` を選択し、XML ファイルを修正します。『Oracle Application Server Containers for J2EE サービス・ガイド』で、`rmi.xml` ファイル内の要素の説明を参照してください。

アプリケーション・レベル構成

EAR または WAR ファイル・アーカイブ形式で存在する J2EE アプリケーションをデプロイ、再デプロイまたはアンデプロイすることができます。アプリケーションをデプロイするには、「アプリケーション」ページの「デプロイ済アプリケーション」セクションでデプロイ用の**「EAR ファイルのデプロイ」**ボタンまたは**「WAR ファイルのデプロイ」**ボタンをクリックします。

これにより、2-11 ページの「アプリケーションのデプロイ」で説明したデプロイ・ウィザードが起動します。EAR ファイルをデプロイする場合は、アプリケーション・モジュールを説明する application.xml がこのファイルに含まれている必要があり、WAR ファイルをデプロイする場合は、application.xml ファイルが自動で作成されます。

アンデプロイするには、アプリケーションの**「選択」**ラジオ・ボタンを選択してから、**「アンデプロイ」**ボタンをクリックします。

再デプロイするには、アプリケーションの**「選択」**ラジオ・ボタンを選択してから、**「再デプロイ」**ボタンをクリックします。

> **注意：** 単純なアプリケーションのデプロイ、アンデプロイまたは再デプロイは、DCM コマンドで実行することもできます。手順は、『Distributed Configuration Management 管理者ガイド』を参照してください。

アプリケーションをデプロイすると、このアプリケーションのほとんどのパラメータを変更できます。アプリケーション固有のパラメータを構成するには、次のようにします。

1. OC4J ホームページで、「アプリケーション」ページを選択します。
2. 構成を変更するアプリケーションを次のいずれかの方法で選択します。
   a. アプリケーションの**「選択」**ラジオ・ボタンを選択してから、**「編集」**ボタンをクリックします。
   b. アプリケーション・ボックスの「名前」列でアプリケーション名を選択します。

このページは、全般的なアプリケーション構成のみでなく、WAR ファイルなど、デプロイ済アプリケーションの特定の部分に固有の構成を変更するための開始ページです。

次の項では、これらの構成オプションの概要を説明します。

- アプリケーション全般のパラメータの構成
- ローカル J2EE サービスの構成
- デプロイ済アプリケーション EAR ファイルに含まれる XML ファイルの変更

アプリケーション全般のパラメータの構成

アプリケーションまでドリルダウンして、「プロパティ」列までスクロールし、「一般」リンクを選択すると、次のような数多くのアプリケーションの詳細を構成できます。

- 永続性パス
- データ・ソースのパス
- ライブラリ・パス
- EJB プロパティ
  - CMP Bean 用のデータベース表を自動的に作成
  - CMP Bean 用の古いデータベース表を自動的に削除
- デフォルト・データ・ソース（JNDI 名）
- ユーザー・マネージャ

ローカル J2EE サービスの構成

3-7 ページの「データ・ソースの構成」と3-11 ページの「セキュリティの構成」で説明されているように、データ・ソースとセキュリティは、デプロイされているすべてのアプリケーション（グローバル）または特定アプリケーション（ローカル）のいずれかに対して構成できます。アプリケーションに対して J2EE サービスを構成する手順は、これらの項を参照してください。

デプロイ済アプリケーション EAR ファイルに含まれる XML ファイルの変更

デプロイ後に変更できるのは、アプリケーションの OC4J 固有の XML ファイルのみです。これは、`orion-ejb-jar.xml`、`orion-web.xml` および `orion-application-client.xml` です。`web.xml`、`ejb-jar.xml` および `application-client.xml` などの J2EE XML ファイルは、変更できません。

OC4J 固有の XML ファイルを変更するには、次のようにします。

1. アプリケーション画面で、構成を変更する JAR または WAR ファイルを選択します。アプリケーション画面が表示されます。
2. 次のいずれかの方法で、アプリケーションのパラメータを変更します。
   - パラメータ変更用の「管理」セクションのリンクに従います。
   - デプロイされた Bean、サーブレットまたは JSP の詳細を示すセクションで、Bean またはサーブレットを選択します。これにより、構成の次のレベルにドリルダウンします。
   - 「管理」セクションには、`orion-ejb-jar.xml`、`orion-web.xml` および `orion-application-client.xml` などの OC4J 固有デプロイメント・ディスクリプタについて XML を直接変更できる「プロパティ」または「拡張プロパティ」セクションがあります。

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 内の各 XML ファイルは、特定の役割のために存在します。そのため、その役割が必要な場合、どの XML ファイルで変更およびメンテナンスを行うかを理解する必要があります。

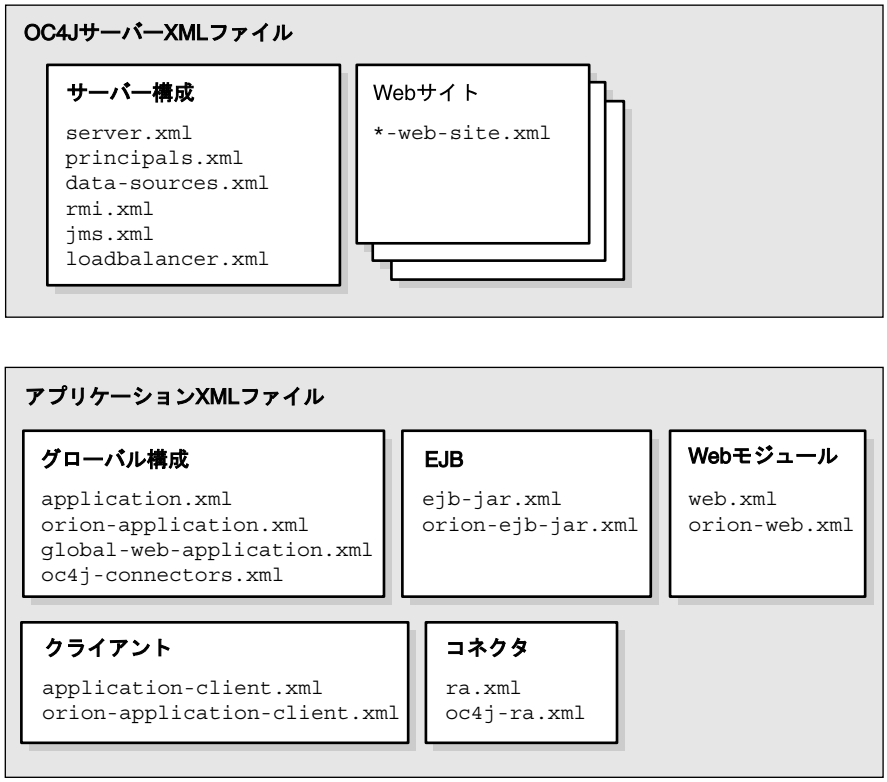
図 3-15 は、OC4J のすべての XML ファイルとそれぞれの役割を示しています。

- 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。

  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 アプリケーションとは直接関係なく、サーバーそのものに関係があります。
- Oracle HTTP Server: これらのファイルは、Oracle HTTP Server 内の構成ファイルです。ただし、OC4J サーバーへのリクエストの送信方法を変更するために、これらのファイルの修正が必要な場合があるため、この図に含まれています。
- 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）には、独自の構成（デプロイ）ファイルが必要です。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コンポーネント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、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 3-15 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、そのアプリケーションにしか使用できず、この OC4J サーバー・インスタンスにデプロイされているすべてのアプリケーションに対して適用されるオプションを構成するグローバルな `application.xml` とは異なります。

表 3-1 は、前の図で示した各 XML ファイルの役割および機能を説明しています。

表 3-1 OC4J の機能および構成要素

XML 構成ファイル	機能 / コンポーネント
`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。その他の XML ファイルのリストによって、サービスを個別のファイルに構成することができますが、`server.xml` ファイルにはそれらのファイルを OC4J 構成に使用することが示されます。
`jazn.xml` `jazn-data.xml`	サーバーへのアクセスに必要な JAZN セキュリティのための OC4J セキュリティ構成。

表 3-1 OC4J の機能および構成要素（続き）

XML 構成ファイル	機能 / コンポーネント
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。
jms.xml	OC4J 内で JMS および MDB によって使用される Destination のトピックおよびキューの OC4J JMS 構成。
default-web-site.xml	OC4J の Web サイトの定義。
mod_oc4j.conf	mod_oc4j モジュールは、OC4J リクエストを転送する Oracle HTTP Server モジュールです。このファイルでは、OC4J に送るコンテキストを示すマウント・ポイントを構成します。
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 インスタンス内のすべてのアプリケーションに共通の設定が含まれます。このファイルは、セキュリティ XML 定義ファイル jazn-data.xml とデータソース XML 定義ファイル data-sources.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、Web アプリケーションごとに存在します。 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 3-1 OC4J の機能および構成要素（続き）

XML 構成ファイル	機能 / コンポーネント
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 3-17 を参照）。

相互関連ファイルは次のとおりです。

- server.xml: 次のファイルへの参照が含まれています。
  - default-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.xml 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、グローバル application.xml で定義されている jazn-data.xml および data-sources.xml は除きます。図 3-15 にこれを示します。
  - 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。
- default-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: jazn-data.xml および data-sources.xml ファイルへの参照が含まれます。

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 3-16 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 3-16 server.xml 内で参照される XML ファイル

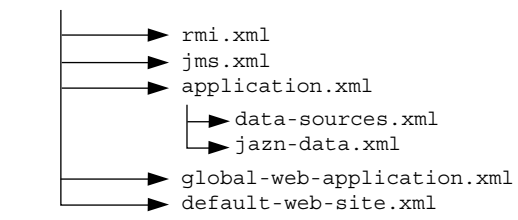


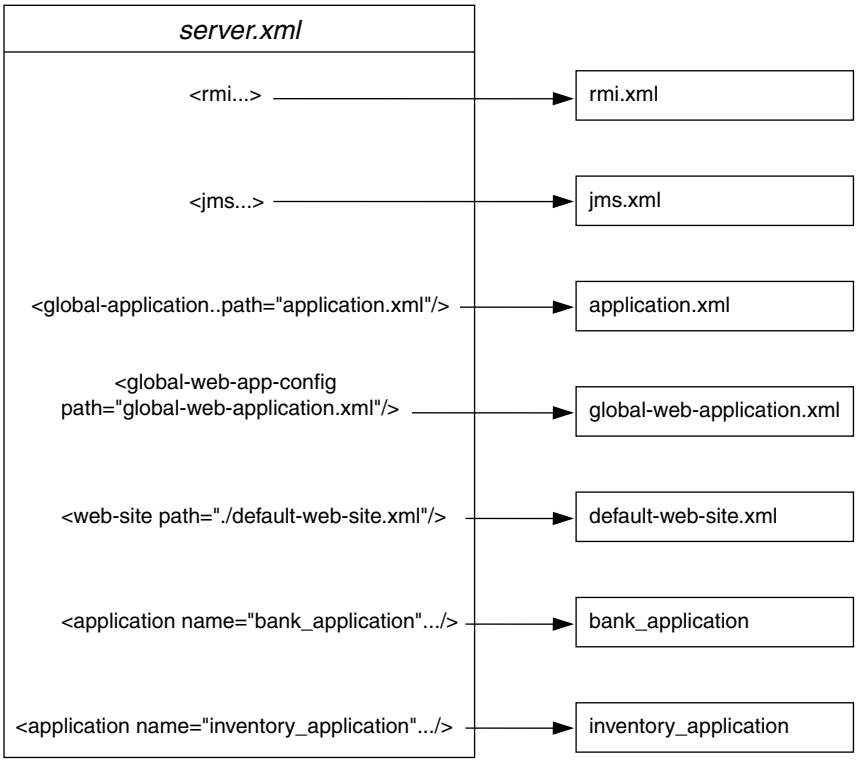
図 3-17 は、server.xml で他の XML 構成ファイルを指定する方法を示します。各 XML ファイルの場所は、絶対パスまたは server.xml ファイルが存在する場所に対する相対パスで指定します。また、XML ファイルの名前は、そのファイル内容が適切な DTD に準拠していればどのような名前でもかまいません。

- <rmi-config> タグは、rmi.xml ファイルの名前と場所を示します。
- <jms-config> タグは、jms.xml ファイルの名前と場所を示します。

- <global-application> タグは、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> タグは、global-web-application.xml ファイルの名前と場所を示します。
- <web-site> タグは、*-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についても記述されています。デプロイされた各アプリケーションは、<application> タグで示します。

図 3-17 server.xml ファイルおよび関連 XML ファイル



server.xml の他のタグについては、3-15 ページの「server.xml ファイルの要素」を参照してください。

> **注意：** 旧リリースの OC4J の OC4J XML ファイルがわかっている場合は、OC4J ホームページにドリルダウンし、「管理」までスクロールして、「拡張プロパティ」をクリックすることによって、大部分の OC4J サーバー XML 構成ファイルを簡単に変更できます。ここでは、Enterprise Manager エディタを使用して XML ファイルを変更できます。

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、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> 要素を他のディレクトリが含まれるように変更するかわりに、ライブラリを applib ディレクトリ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。

できれば、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 ファイルにより、そのアプリケーション専用にしておく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、そのアプリケーション内でのみ使用されるライブラリの場所を指定できます。

OC4J リスナーの理解および構成

受信クライアント・リクエストでは、AJP、HTTP または RMI の 3 つのプロトコルのいずれかが使用されます。AJP と HTTP は、HTTP リクエストに使用されます。AJP は、OHS と OC4J コンポーネントの間で使用されます。HTTP は、OHS を通り過ぎて OC4J に送られる HTTP リクエストに使用されます。RMI は受信 EJB リクエストに使用されます。

HTTP リクエスト

HTTP リクエストは、OHS を経由するか、OC4J に直接送信されるかにかかわらず、1 つの OC4J Web サイトに 1 つのリスナーが構成されている必要があります。OC4J インスタンスごとに複数の Web サイトを持ち、プロトコル・タイプ別に使用できます。1 つの Web サイトで両タイプのプロトコルをリスニングすることはできません。このため、OC4J では、次の 2 つの Web サイト構成ファイルを提供しています。

- default-web-site.xml: これは、AJP プロトコル・リスナーであり、Oracle Application Server を使用する大部分の HTTP リクエストのデフォルトです。インストール後に、Oracle HTTP Server のフロントエンドが受信 HTTP リクエストを AJP ポート上で転送します。OC4J Web サーバーの構成ファイル（default-web-site.xml）で、AJP リスナー・ポートが指定されます。default-web-site.xml ファイルでは、デフォルトの AJP ポートが 0（ゼロ）に定義されます。これにより、OC4J および Oracle HTTP Server では、起動時にポートのネゴシエーションが可能になります。

  AJP ポート値の範囲は、OPMN 構成で構成します。OPMN の詳細は、『Oracle Application Server 管理者ガイド』の「高可用性」の章を参照してください。

  デフォルト AJP リスナー用の default-web-site.xml 内のエントリは、次のとおりです。

  ```
  <web-site host="oc4j_host" port="0" protocol="ajp13"
      display-name="Default OC4J WebSite">
  ```

  AJP のデフォルト Web サイト・プロトコルは、OC4J ホームページの「Web サイト・プロパティ」または「拡張プロパティ」で構成できます。

- http-web-site.xml: これは、HTTP プロトコル・リスナーです。OC4J スタンドアロン環境のように、OHS を介さずに直接 OC4J に送信する場合は、このファイルに定義されたポート番号を使用します。

  AJP ポートは、OC4J の起動時に毎回無作為に選択されることに注意してください。このポート番号がこの XML ファイルにハードコードされたポート番号と同じになると、競合が発生します。受信リクエストにこの方法を使用する場合は、選択したポート番号が AJP ポート番号の範囲外であることを確認します。AJP ポート番号の範囲は、OPMN 構成に定義されています。

  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は、次のとおりです。

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

注意： UNIX 環境では、プロセスに管理権限がある場合を除き、ポート番号を 1025 以上にする必要があります。

RMI リクエスト

RMI プロトコル・リスナーは、RMI 構成（rmi.xml）の OPMN により設定されます。これは、Web サイト構成とは異なります。EJB クライアントおよび 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。OPMN は、RMI リスナーが使用できるポートの範囲を指定します。参照内で opmn:ormi:// という文字列を使用すると、クライアントは割り当てられた RMI ポートを自動的に取得します。詳細は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

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。

別の Web コンテキストによる Oracle HTTP Server の構成

Oracle HTTP Server の mod_oc4j モジュールは、j2ee/ コンテキストにバインドされているアプリケーションをすべて OC4J サーバーに送るよう、インストール時に構成されています。pubs/ など、別のコンテキストを使用する場合、mod_oc4j.conf 構成ファイルで、このコンテキスト用に別のマウントを追加できます。

このファイルを変更するには、「Oracle HTTP Server」ページにドリルダウンして、mod_oc4j.conf を選択します。編集用のファイルが右側のフレームに表示されます。

1. j2ee/ ディレクトリの Oc4jMount ディレクティブを探します。これを別の行にコピーします。マウント・ディレクティブは、次のようになります。

   ```
   Oc4jMount /j2ee/* OC4Jworker
   ```

   注意： OC4Jworker は、mod_oc4j.conf ファイルのさらに下の行で OC4J インスタンスとして定義されています。

   ---

2. j2ee/ コンテキストを、使用するコンテキストに変更します。この例の場合、pubs/ マウント構成の行が追加されます。OC4J のワーカー名が OC4Jworker の場合、この 2 行は次のようになります。

   ```
   Oc4jMount /j2ee/* OC4Jworker
   Oc4jMount /pubs/* OC4Jworker
   ```

3. 新しいマウント・ポイントを使用するには、Oracle HTTP Server を再起動します。

これにより、pubs/ コンテキストの受信リクエストはすべて OC4J サーバーに送られます。デプロイ・ウィザードを使用してアプリケーションをデプロイすると、ここで説明したように URL マッピングに対してマウント・ポイントが自動的に追加されます。

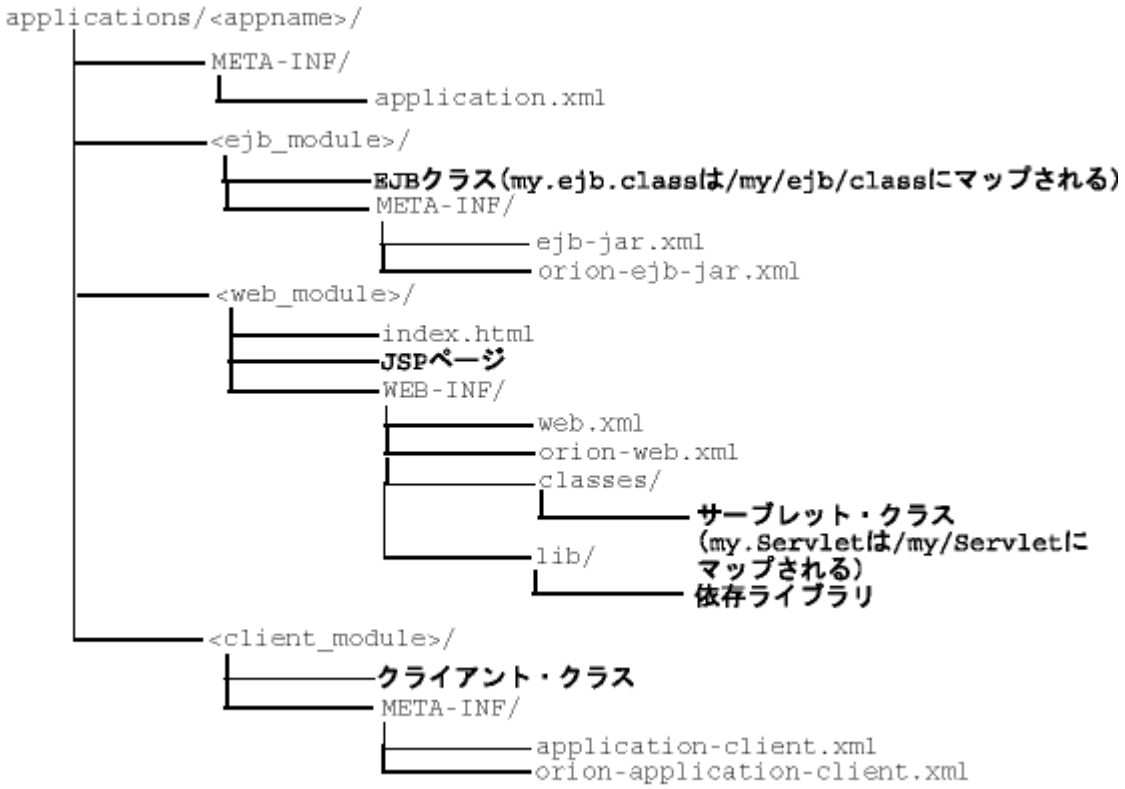
mod_oc4j モジュール構成の詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。

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。OC4J は、図 3-18 の <appname> に書かれたトップ・ディレクトリのタイムスタンプが変わると、自動的にアプリケーションをデプロイします。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 3-18 のようになります。

図 3-18 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 3-18 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 3-18 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. 各モジュールのクラスを、それらのパッケージ構造にマッピングする適切なディレクトリ構造内に置きます。

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 3-18 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 3-18 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 3-18 の例では、<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、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 3-18 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 ファイルで各 Web アプリケーションに対し、<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、この手順は重要です。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： JAR 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、JAR 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。

  ---

3. mod_oc4j.conf 構成ファイルに Web アプリケーションのマウント・ポイントを追加します。この例の root="/myapp" に基づき、次の行が含まれるように mod_oc4j.conf を更新する必要があります。

   ```
   Oc4jMount /myapp home
   Oc4jMount /myapp/* home
   ```

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

起動クラスは、OC4J の初期化後に 1 回のみ実行されます。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返しますが、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。このファイルにアクセスするには、OC4J ホームページの**「拡張プロパティ」**を選択します。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]" /> エレメントで、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 3-1 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。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。

```
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、3-25 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、server.xml ファイルの <init-library> 要素を次のようにもう 1 つ追加します。

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

起動クラスおよび停止クラスの考慮事項

起動クラスと停止クラスの開発時には、次の問題に注意してください。

- 停止前に、アンデプロイされるアーカイブ・ファイル（WAR および EAR）から停止クラスを分離します。
- OC4J では、起動クラスのインスタンスと、停止クラスのインスタンスを作成します。これらは別々のインスタンスです。同じクラスに起動および停止のインタフェースを実装すると、OC4J ではそのクラスの 2 つのインスタンスを作成します。この場合、起動クラスと停止クラスの間で、非静止フィールドを共有できません。

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多くは『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。

- パフォーマンスのコマンドライン・オプション
- スレッド・プールの設定

- 文のキャッシング
- タスク・マネージャの粒度

パフォーマンスのコマンドライン・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-29 ページの「OC4J のコマンドライン・オプションおよびシステム・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]。Oracle9*i*AS 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・メトリックを有効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・プールは、OC4J プロセスで使用するためのスレッドのキューを作成およびメンテナンスします。要求に応じて新規スレッドを作成するのではなく、既存のスレッドを再利用することで、パフォーマンスが高まり、JVM および基礎となるオペレーティング・システムに対する負荷を軽減します。

デフォルトでは、OC4J の起動時に 1 つのスレッド・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、プールに追加されます。各スレッドは解放されるとプールに返され、新しいリクエストで必要となるまで待機します。

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

デフォルト構成で、OC4J によって使用されるほとんどの場合に十分対応できます。ただし、デフォルトで作成された単一のスレッド・プールは、server.xml ファイル内の <global-thread-pool> 要素の min、max、queue および keepAlive 属性により変更できます。

または、<global-thread-pool> を使用してスレッド・プールを 2 つ作成し、これらのプールで異なるタイプのスレッドを振り分けることができます。

- 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・スレッドが、MDB リスナー・スレッドと同様に含まれています。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およびバックグラウンド・スレッドなどのスレッドが含まれます。これらのスレッドは、通常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場合、ワーカー・スレッド・プールには min、max、queue および keepAlive 属性を、接続スレッド・プールには 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。これらの属性はすべてプール作成の場合に構成する必要があり、構成しないと、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> の属性の説明は、3-29 ページの表 3-2 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 3-2 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 に含まれていないので注意してください。

表 3-2 <global-thread-pool> の属性

属性	説明
min	プールで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、デフォルト値は 20 に設定されます。指定できる最小値は 10 です。
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。 スレッドが破棄され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、これは -1 でない場合に設定できる最小値でもあります。

表 3-2 <global-thread-pool> の属性（続き）

属性	説明
cx-min	接続スレッド・プールで作成するスレッドの最小数です。 指定できる最小値は 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で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。 スレッドが破棄され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、これは -1 でない場合に設定できる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は次のとおりです。

- 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 数の倍数としてください。ただし、この数値は小さく設定する必要があり、スレッドの数が増えるほど、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 属性および cx-max 属性は、任意の時点での物理接続数に関連しています。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、『Oracle Database 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で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 3-2 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負荷が大き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。この属性は、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

Oracle9*i*AS 環境内の各 OC4J プロセスには、表 3-3 のようなログ・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 3-3　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。	デプロイ済アプリケーションごとに 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
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	default-web-site.xml

ログ・ファイルには 2 つのタイプがあります。

- Oracle Diagnostic Logging（ODL）のログ・ファイル: これらのファイルに記録されるメッセージは、Enterprise Manager GUI によって読み取られる XML 形式を使用します。Oracle9*i*AS 内で OC4J を使用する場合は、デフォルトではありませんが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: これらのファイルに記録されたメッセージは XML 形式ではなく、単に任意のエディタでの読取り用です。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

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、Enterprise Manager GUI または XML リーダーで読み取ることができます。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 から始まる数字です。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 3-3 に示したそれぞれの XML ファイルで、次のように既存の ODL 構成行を非コメント化するか、XML ファイルに ODL 構成行を追加することにより、OCL ロギングを有効にします。

- `default-web-site.xml` ファイルを除き、表 3-3 に示したすべての XML ファイルの `<log>` 要素内で、`<odl>` 要素を追加するか非コメント化します。
- `default-web-site.xml` ファイルに `<odl-access-log>` 要素を追加します。

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。絶対パスか、または構成 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、`ORACLE_HOME/j2ee/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 *ORACLE_HOME*`/j2ee/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>
```

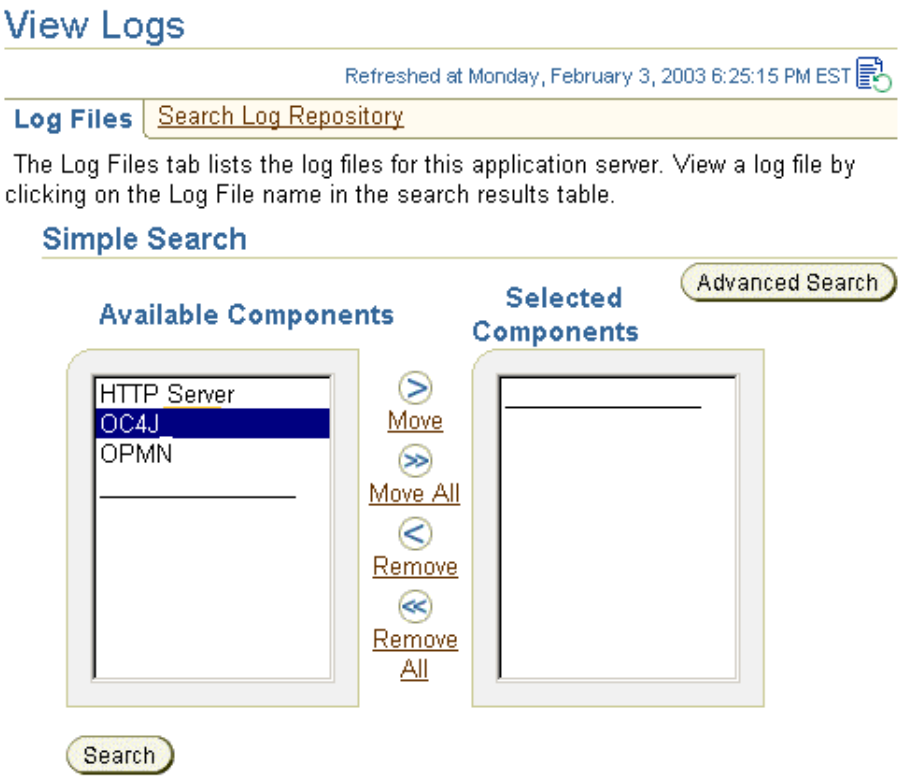
```
<TSTZ_ORIGINATING>2002-11-12T15:02:07.051-08:00</TSTZ_ORIGINATING>
<COMPONENT_ID>oc4j</COMPONENT_ID>
<MSG_TYPE TYPE="ERROR"></MSG_TYPE>
<MSG_LEVEL>1</MSG_LEVEL>
<HOST_ID>myhost</HOST_ID>
<HOST_NWADDR>001.11.22.33</HOST_NWADDR>
<PROCESS_ID>null-Thread[Orion Launcher,5,main]</PROCESS_ID>
<USER_ID>dpda</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・ロギングの両方を有効にできます。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 つは無効にしてください。ODL ロギングを有効にする場合は、`default-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>` 副要素をコメント化して、テキスト・ロギング機能を無効にします。`default-web-site.xml` ファイルについては、`<access-log>` 要素をコメント化してテキスト・ロギングを無効にします。

ODL ログ・ファイルを表示するには次のようにします。

1. Oracle Application Server インスタンスのホームページで**「ログ」**を選択します。
2. 表示するログ・ファイルの OC4J インスタンスを選択します。
   a. 「使用可能なコンポーネント」列で OC4J インスタンスを選択します。
   b. **「移動」**を選択し、選択したものを「選択したコンポーネント」列に移動します。
   c. 「検索」を選択し、これらの OC4J インスタンスのログ・ファイルを表示します。

図 3-19 ログの表示



関連資料： 『Oracle Application Server 管理者ガイド』の「ログ・ファイルの管理」の章

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 ファイルにあわ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 3-3 で示した ODL ロギングと同じ XML ファイルに構成されます。テキスト・メッセージングは、`default-web-site.xml` ファイルを除き、XML ファイルの `<log>` 要素の `<file>` 副要素により有効になります。`default-web-site.xml` ファイルについては、`<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・メッセージングを無効にするには、`<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有効になります。表 3-4 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。

次の表は、OC4J プロセスのテキスト・メッセージング・ログ・ファイルのデフォルトの場所のディレクトリを示しています。

表 3-4 OC4J ログ・ファイルの場所

ログ・ファイル	デフォルトの場所
application.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/application-deployments/<application-name>/<OC4J IslandName>
global-application.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
jms.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
rmi.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
server.log	*$ORACLE_HOME*/j2ee/<OC4J InstanceName>/log/<OC4J IslandName>_<Process#>
web-access.log	この場所は、次のように <access-log> 要素を使用して *-web-site.xml から構成されます。<access-log path="../log/http-web-access.log" />
OPMN ログ・ファイル <OC4J_Instance_Name>.<Island_Name>.<Process#>	*$ORACLE_HOME*/opmn/logs/

前述のログ・ファイルの場所は、`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・ファイルの場所は、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>
```

標準出力および標準エラーのリダイレクト

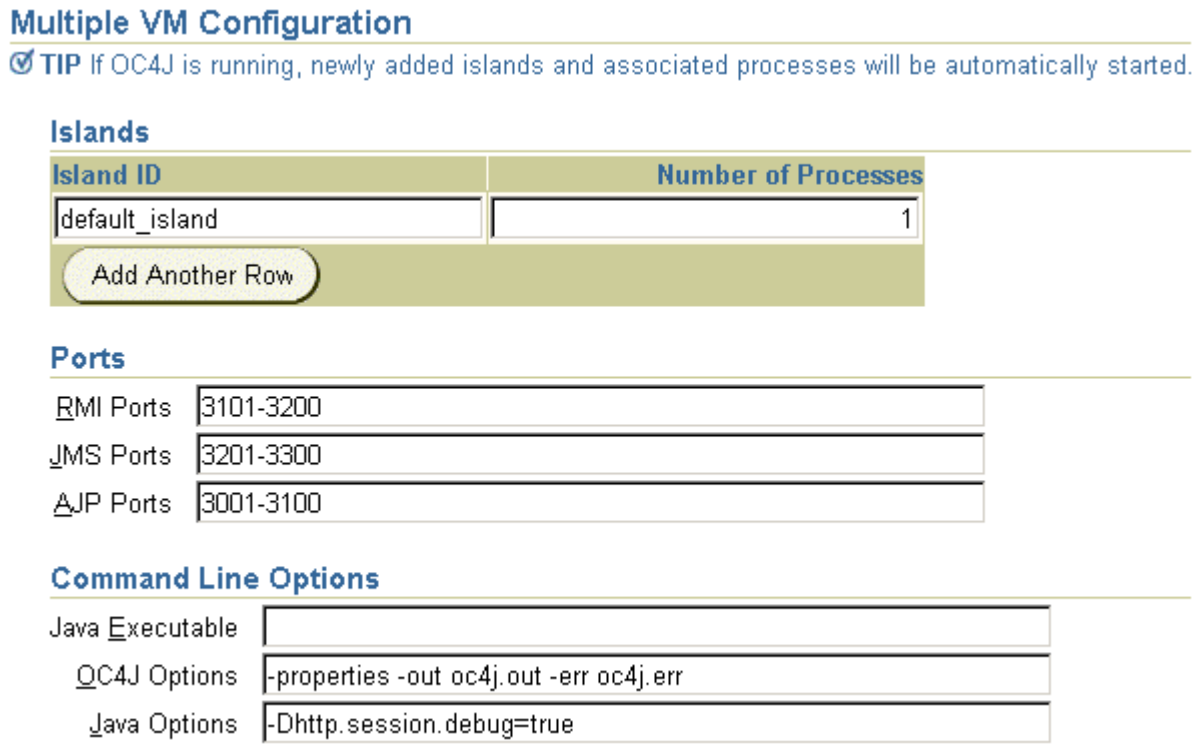
Oracle Application Server 環境では、OC4J の標準出力および標準エラーは OC4J インスタンスの OPMN ログに送られます。たとえば、`OC4J_Demos` という OC4J インスタンスに、`default_island` というアイランドがあり、このアイランドにプロセスが 1 つしかない場合、STDOUT および STDERR ストリームを含むファイルは、*`$ORACLE_HOME`*`/opmn/logs/OC4J_Demos.default_island.1` となります。

図 3-20 は、OC4J コマンドライン・オプションで OC4J インスタンスに `-out` および `-err` パラメータを指定する方法を示しています。特定のディレクトリを指定しないでこれらのパラメータを指定すると、ログ・ファイルは `$J2EE_HOME/<OC4JInstanceName>_<IslandName>_<Process#>` ファイル内に作成されます。たとえば、`OC4J_Demos` というインスタンスに `default_island` というアイランドがある場合、ログ・ファイルは `$J2EE_HOME/OC4J_Demos_default_island_1` 内に作成されます。

> **注意：** 前述の例では、デフォルト・アイランドに OC4J プロセスが 1 つしかないものと仮定しています。アイランドに複数のプロセスがある場合、OC4J プロセスごとに別個のログ・ファイルが作成されます。

Application Server Control で、「管理」ページの「サーバー・プロパティ」を選択すると、図 3-20 の画面が表示されます。

図 3-20 OC4J インスタンスのサーバー・プロパティを変更する Enterprise Manager コンソール



OC4J のデバッグ

OC4J のプロパティは、コマンドラインで設定可能な構成スイッチです。図 3-20 に示すように、「Java オプション」行でプロパティの先頭に `-D` が付いています。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。

OC4J は OPMN により起動および管理されます。図 3-20 に示すように、Enterprise Manager を使用して、OC4J インスタンスの Java システム・プロパティを指定する必要があります。指定されたプロパティは OPMN 構成ファイルに保存されます。ユーザーが OC4J インスタンスを停止して再起動すると、OPMN は指定されたこれらのプロパティを使用して OC4J を起動します。OC4J の一般プロパティの詳細は、B-29 ページの「OC4J のコマンドライン・オプションおよびシステム・プロパティ」を参照してください。

デフォルトでは、デバッグ情報は *$ORACLE_HOME*`/opmn/logs/` ディレクトリ内の OC4J インスタンスの OPMN ログ・ファイルに書き込まれます。たとえば、`OC4J_DEMOS` という OC4J インスタンスにアイランドが 1 つしかなく、このアイランドにこの OC4J インスタンスのプロセスが 1 つしかない場合、デバッグ情報は *$ORACLE_HOME*`/opmn/logs/OC4J_Demos.default_island.1` に記録されます。ただし、OC4J で `-out` および `-err` コマンドライン・オプションが指定されている場合、デバッグ情報は該当するファイルにリダイレクトされます。

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-29 ページの「OC4J のコマンドライン・オプションおよびシステム・プロパティ」を参照してください。

表 3-5 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.cluster.debug	HTTP クラスタリング・イベン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。
http.method.trace.allow	デフォルトは false です。true の場合は、trace HTTP メソッドを有効にします。

表 3-6 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	プールに対して解放されたデータ・ソースと接続を使用したデータ・ソースおよび接続の作成などに関する詳細情報を提供します。
jdbc.debug	JDBC コールの際に非常に詳細な情報を提供します。

表 3-7 EJB デバッグ・オプション

EJB デバッグ	オプションの説明
ejb.cluster.debug	EJB クラスタリング・デバッグ・メッセージを有効にします。

表 3-8 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 3-9 Web サービス・デバッグ・オプション

Web サービス・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有効にします。

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・セクションで Enterprise Manager の -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 3-3 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 3-4 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・ソースの使用に関して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。

次のタスクを実行します。

1. 管理者として Enterprise Manager コンソールにログオンします。
2. OC4J インスタンスまでドリルダウンします。
3. OC4J インスタンスの**「サーバー・プロパティ」**を選択します。
4. Java オプションを `-Dhttp.session.debug=true` および `-Ddatasource.verbose=true` のように入力します。
5. OC4J インスタンスを再起動します。

OC4J インスタンスの再起動後にサーブレットを再実行すると、OC4J インスタンスの標準出力に次のタイプのデバッグ情報が表示されます。

```
DataSource logwriter activated... jdbc:oracle:thin:@localhost:1521/DEBU:
Started
jdbc:oracle:thin:@localhost:1521/DEBU: 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 Orion Pooled
jdbc:oracle:thin:@localhost:1521/DEBU
null: Connection XA XA Orion Pooled jdbc:oracle:thin:@localhost:1521/DEBU
allocated (Pool size: 0)
jdbc:oracle:thin:@localhost:1521/DEBU: Opened connection
Created new physical connection: Pooled
oracle.jdbc.driver.OracleConnection@5f18
Pooled jdbc:oracle:thin:@localhost:1521/DEBU: Connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 allocated (Pool size: 0)
Pooled jdbc:oracle:thin:@localhost:1521/DEBU: Releasing connection Pooled
oracle.jdbc.driver.OracleConnection@5f1832 to pool (Pool size: 1)
null: Releasing connection XA XA Orion Pooled
jdbc:oracle:thin:@localhost:1521/DEBU to pool (Pool size: 1)
Orion Pooled jdbc:oracle:thin:@localhost:1521/DEBU: Cache timeout, closing
connection (Pool size: 0)
com.evermind.sql.OrionCMTDataSource/default/jdbc/OracleDS: Cache timeout,
closing connection (Pool size: 0)
```

Oracle JDeveloper を使用したリモート・デバッグ

JPDA（Java Platform Debugging Architecture）をサポートする任意の Java デバッグ機能を使用して、OC4J にデプロイしたアプリケーションをリモートでデバッグできます。OC4J は Oracle JDeveloper IDE 内に直接埋め込まれているため、ローカルにデプロイした場合も、リモートの J2EE アプリケーションの場合も簡単にデバッグできます。詳細は、Oracle JDeveloper のマニュアルを参照するか、OTN で公開されているリモート・デバッグに関する How To ドキュメントを参照してください。

4

OC4J のクラスタリング

この章では、OC4J のクラスタリングと、クラスタ内の OC4J プロセスおよびアプリケーションの管理方法について説明します。

この章の項目は次のとおりです。

Oracle Application Server 内のクラスタリングの全体像は、『Oracle Application Server 高可用性ガイド』を参照してください。

クラスタ内の OC4J インスタンス

OC4J インスタンスは、J2EE アプリケーションがデプロイおよび設定されたエンティティです。OC4J インスタンスによって、Application Server 内の OC4J プロセスの数と、その OC4J プロセス用の設定が定義されます。OC4J プロセスは、OC4J インスタンスの J2EE アプリケーションを実行するプロセスです。

OC4J インスタンスには、次のような特徴があります。

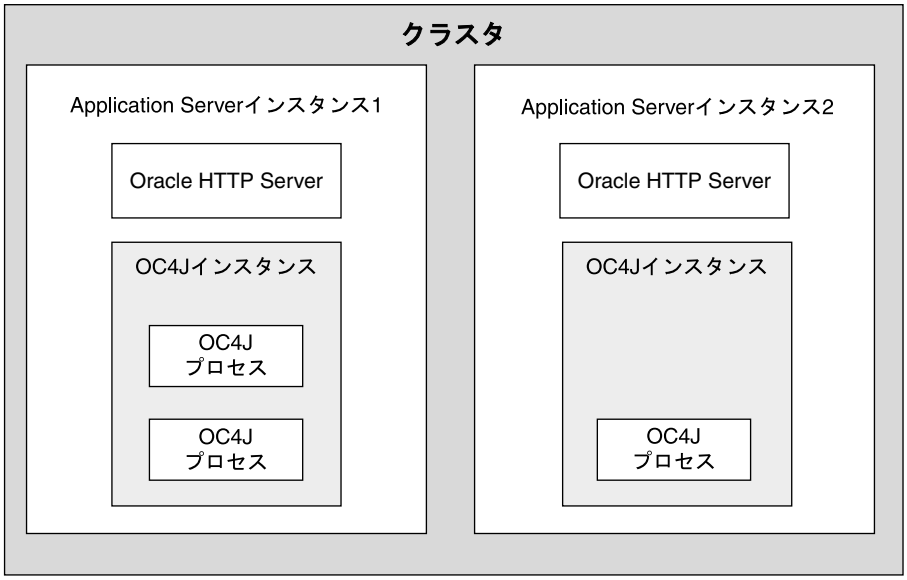
- OC4J インスタンスの設定は、1 つ以上の OC4J 実行可能プロセスに対して有効です。このため、これらのプロセスを OC4J インスタンスの設定で管理することで、複数の OC4J プロセス用の設定を複製できます。OC4J インスタンス内のクラスタ・レベルの設定を変更すると、その変更内容がすべての OC4J プロセスに対して有効になります。
- 各 OC4J インスタンスは、1 つ以上の OC4J プロセスで設定できます。
- ある OC4J インスタンスにアプリケーションをデプロイすると、その OC4J インスタンスによって、その中に定義されているすべての OC4J プロセスに対してそのアプリケーションがデプロイされます。OC4J インスタンスは、アプリケーションの状態をレプリケートする役割も果たします。
- OC4J プロセスの数は、OC4J インスタンスごとに異なります。この数は、クラスタ内の Application Server インスタンスごとに手動で設定する必要があります。OC4J プロセスの設定には柔軟性があり、ホスト特有のハードウェア能力に応じたチューニングが可能となっています。デフォルトでは、OC4J インスタンスがそれぞれ、単一の OC4J プロセスでインスタンス化されます。

Application Server インスタンス内には、複数の OC4J インスタンスを設定できます。各インスタンスにはそれぞれの OC4J プロセスの数を設定できます。これは、クラスタ内の個別の OC4J プロセスに対する構成管理とアプリケーション・デプロイに有利です。

図 4-1 は、デフォルトの OC4J インスタンスである home インスタンスを示しています。クラスタのコンテキストでは、OC4J インスタンスの設定はクラスタ全体の設定の一部となります。つまり、最初のアプリケーション・インスタンスで設定した home インスタンスは、他のすべての Application Server インスタンスにレプリケートされます。

各 home インスタンスのプロセス数は、インスタンス固有のパラメータです。このため、各 Application Server インスタンス上に存在する OC4J プロセスと同数の home インスタンスを、Application Server インスタンスごとに個別に設定する必要があります。図 4-1 は、Application Server インスタンス 1 の home インスタンスには OC4J プロセスが 2 つ含まれているのに対し、Application Server インスタンス 2 の home インスタンスには 1 しか含まれていないことを示しています。デフォルトでは、各 OC4J インスタンス内に 1 つの OC4J プロセスが含まれます。

図 4-1 クラスタ内の OC4J プロセス



クラスタ内の OC4J プロセス

OC4J プロセスは、J2EE アプリケーションを実行する JVM プロセスです。各 OC4J プロセスは OC4J インスタンスに関連付けられ、そのプロセスの設定は OC4J インスタンスから継承されます。ある OC4J インスタンスにデプロイされたすべてのアプリケーションは、その OC4J インスタンスに関連付けられているすべての OC4J プロセスで開始されます。

OC4J インスタンス内に 1 つ以上の OC4J プロセスを定義しておくと、J2EE リクエストの負荷を分散できるようになる上、フェイルオーバー機能も得られます。

OC4J プロセスの数に関する設定は、インスタンス固有のものです。したがって、Application Server インスタンスに各 OC4J インスタンスを設定するときには、その OC4J インスタンス用に起動させる OC4J プロセスの数を設定する必要があります。OC4J プロセスのデフォルトの数は 1 です。

Application Server インスタンスをインストールするホストの能力はそれぞれ異なります。ハードウェア性能を最大限にするには、各 OC4J インスタンスに、これらの性能を適切に使用する OC4J プロセスの数を設定します。たとえば、ホスト A に OC4J プロセスを 1 つ設定し、ホスト B に OC4J プロセスを 5 つ設定することも可能です。

複数の OC4J プロセスを定義しておくと、次のことが可能になります。

- 複数の OC4J プロセスを使用することで、複数のユーザーにサービスを提供できるようになります。
- 複数の OC4J プロセスにわたってアプリケーションの状態をレプリケートしておくことで、フェイルオーバー機能を利用できるようになります。
- OHS は、OC4J インスタンス内のすべての OC4J プロセスに対してロード・バランスを提供します。OPMN コンポーネントは、新規 OC4J プロセスが開始されると、各 OHS に通知します。したがって、クラスタ内の各 OHS は、クラスタ内の各 OC4J プロセスを認識しています。

注意： `mod_oc4j` は OHS と OC4J の間にあり、使用可能な OC4J インスタンスに対して、リクエストのメトリックベース・ロード・バランシングを行うことができます。この機能を実装する手順は、B-6 ページの「server.xml ファイルの要素」にある `<metric-collector>` 要素についての説明を参照してください。

アプリケーションの状態のレプリケート

クラスタ内に含まれる OC4J プロセスは、アプリケーションの状態をすべての OC4J プロセスにレプリケートできます。レプリケーションを設定すると、OC4J によってアプリケーションの状態の伝播が自動的に行われます。

ある OC4J プロセスが失敗した場合は、その OC4J プロセスのアプリケーションの状態をレプリケートされている別の OC4J プロセスが、そのアプリケーションのリクエストを引き継ぎます。ステートフル・リクエスト中に OC4J プロセスに障害が発生すると、OHS は、次の順にリクエストを転送します。

1. 同じ Application Server インスタンス内で別の OC4J プロセスがアクティブな場合、OHS はリクエストをこのプロセスに転送します。
2. それ以外の場合、OHS は、クラスタ内の別の Application Server インスタンスの OC4J プロセスに、その状態のリクエストを転送します。

防止する必要のある障害には、ソフトウェア障害とハードウェア障害の 2 種類があります。

障害のタイプ	防止する方法
OC4J プロセスの失敗によるソフトウェア障害の発生	同じ OC4J インスタンス内に、複数の OC4J プロセスを設定します。1 つの OC4J プロセスに障害が発生すると、OHS は、同じ OC4J インスタンス内の別の OC4J プロセスにリクエストを転送します。
ホストのダウンによるハードウェア障害の発生	クラスタ内の OC4J プロセスをそれぞれ異なるホストに設定します。最初のホストが停止すると、そのリクエストを別のホスト上の OC4J プロセスが引き継ぎます。それには、クラスタに含まれる別のホストに Application Server インスタンスをインストールしておき、OC4J インスタンスに OC4J プロセスが 1 つ以上存在することが必要です。

アイランド

アイランドは、状態をレプリケートする OC4J プロセスを指定する際に使用する、OC4J プロセスの論理グループです。

各 OC4J インスタンス内には、複数の OC4J プロセスを設定できます。すべての OC4J プロセスが状態レプリケーションを試みた場合を考慮すると、CPU の負荷が大幅に増加する可能性があります。パフォーマンスの低下を防ぐために、OC4J インスタンスでは OC4J プロセスをいくつかのグループに分けることができます。このグループをアイランドと呼びます。

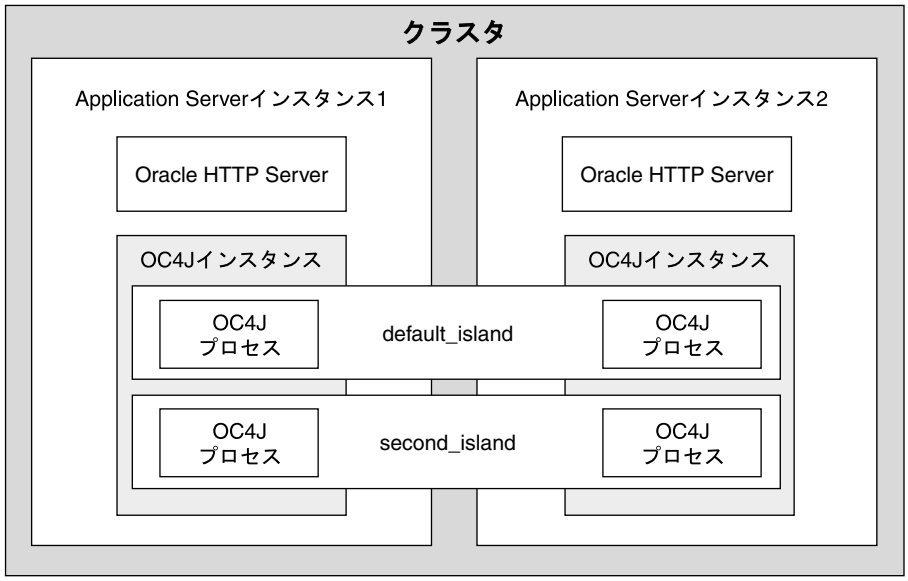
CPU の負荷がプロセス間で振り分けられるようにするには、OC4J インスタンスの OC4J プロセスをいくつかのアイランドに分けます。アプリケーション・リクエストの状態のレプリケート先は、同じアイランド内にグループ化されている OC4J プロセスのみです。すべてのアプリケーションは、今までどおり、OC4J インスタンス内のすべての OC4J プロセスにデプロイされます。唯一異なる点は、このアプリケーションの状態がその一部の OC4J プロセスに限定されることです。

アイランドの設定は、インスタンス固有のものです。アイランドには、アイランドを設定する各 OC4J インスタンスで同じ名前を付けてください。各 Application Server インスタンスに OC4J プロセスの数を設定するときに、それらの OC4J プロセスをさらに別々のアイランドに分けることもできます。OC4J プロセスは、アイランドの名前別に Application Server インスタンス全体にわたってグループ化されます。したがって、アプリケーションの状態は、Application Server インスタンス全体にわたって、同名のアイランドにあるすべての OC4J プロセスにレプリケートされます。

状態レプリケーションの対象となる OC4J プロセスのグループは、Web アプリケーションと EJB アプリケーションとでは異なります。Web アプリケーションの場合は、アイランドのサブグループ内で状態がレプリケートされます。EJB アプリケーションの場合は、OC4J インスタンス内のすべての OC4J プロセス間で状態がレプリケートされ、アイランドのサブグループは使用されません。

図 4-2 は、クラスタ内のアイランドにある OC4J プロセスを示しています。home インスタンスには、default_island および second_island の 2 つのアイランドが設定されています。各 Application Server インスタンスのアイランドにはそれぞれ 1 つの OC4J プロセスが設定されています。OC4J アイランド（網掛けの部分内に示された部分）は、2 つの Application Server インスタンスにまたがっています。

図 4-2 アイランドの例



クラスタに含まれる J2EE アプリケーション

Application Server インスタンスがクラスタ内に含まれているかどうかに関係なく、どのような場合にも OC4J インスタンスに J2EE アプリケーションがデプロイされます。ただし、クラスタ内の OC4J インスタンスにアプリケーションをデプロイする場合は、次の詳細な設定を行う必要があります。

- マルチキャスト・ホストおよびマルチキャスト・ポート : アプリケーションの状態はマルチキャスト・アドレスを介して OC4J プロセス間でレプリケートされます。EJB アプリケーションの場合は、ユーザー名およびパスワードも指定する必要があります。マルチキャスト・アドレスのデフォルトをそのまま使用しても、Enterprise Manager を介して設定してもかまいません。
- 状態レプリケーションのリクエスト : Enterprise Manager を介して、すべてのアプリケーションの状態レプリケーションをリクエストします。
- XML デプロイメント・ディスクリプタの要素 : Web アプリケーションと EJB アプリケーションのどちらの場合にも、それぞれの XML デプロイメント・ディスクリプタに設定を追加する必要があります。
- アイランドの定義 : Web アプリケーションの場合は、状態レプリケーションにアイランドのサブグループを使用します。EJB アプリケーションの場合は、アイランドのサブグループを無視し、状態レプリケーションにすべての OC4J プロセスを使用します。

インスタンス固有のパラメータ

次のパラメータは、クラスタ全体にわたってレプリケートされないため、Application Server ごとに OC4J インスタンス・レベルで設定する必要があります。

- アイランドおよび OC4J プロセスの数 : 複数の Application Server インスタンスにわたってアイランド名の一貫性を保つ必要がありますが、アイランドの定義と OC4J プロセスの数は別々に設定します。各 Application Server インスタンスをインストールするホストの能力はそれぞれ異なります。各ホスト上では、そのホストの能力に応じて OC4J プロセスの数を調整できます。複数のアプリケーションの境界をまたがったアイランドの中で状態がレプリケートされることに注意してください。そのため、アイランド名は各 OC4J インスタンスで同じにする必要があります。
- ポート番号 : RMI、JMS および AJP のポート番号は、ホストごとに異なる値を使用できます。
- コマンドライン・オプション : ホストごと異なるコマンドライン・オプションを使用できます。

OC4J のクラスタリングの例

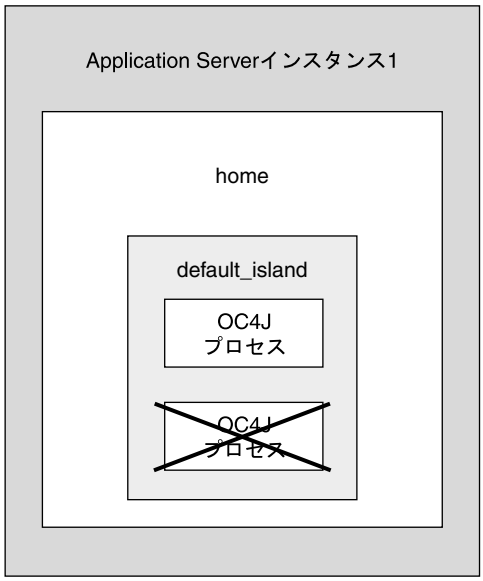
クラスタに追加する Application Server インスタンスの数に関係なく、クラスタ・レベルの設定はクラスタ内でレプリケートされます。ソフトウェア障害およびハードウェア障害の防止は、インスタンス固有のパラメータであるアイランドおよび OC4J プロセスの数の設定によって制御します。

ソフトウェア障害

OC4J インスタンス内に複数の OC4J プロセスを設定した場合は、これらのプロセスの 1 つに障害が発生すると、別のプロセスがそのプロセスのワークロードを引き継ぐことができます。図 4-3 は、クラスタに含まれている Application Server インスタンス 1 を示しています。この Application Server インスタンスには、`home` インスタンスの `default_island` に 2 つの OC4J プロセスが定義されています。最初の OC4J プロセスが失敗した場合は、そのワークロードをもう 1 つのプロセスが引き継いで代行できます。

これらの OC4J プロセスは両方とも同じホスト上にあります。そのため、ホストがダウンした場合は両方の OC4J プロセスが失敗してしまい、クライアントは処理を続行できません。

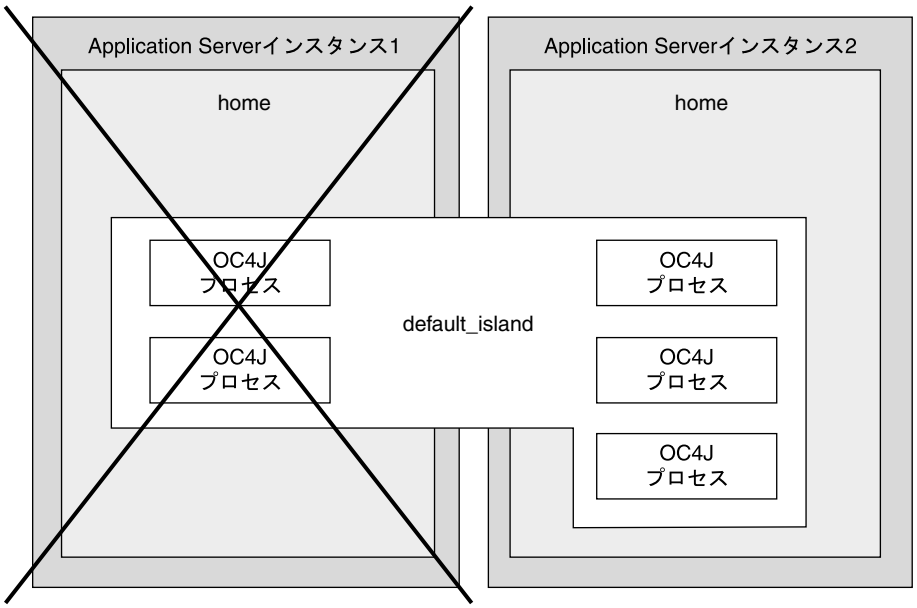
図 4-3 ソフトウェア障害の例



ハードウェア障害

ハードウェア障害に対応するには、同じ OC4J インスタンスの OC4J プロセスを複数のホストにわたって設定する必要があります。図 4-4 は、Application Server インスタンスの 1 と 2 にある home インスタンスを示しています。default_island 内には、2 つの OC4J プロセスが Application Server インスタンス 1 に、3 つの OC4J プロセスが Application Server インスタンス 2 に設定されています。クライアントが Application Server インスタンス 1 の OC4J プロセスの 1 つと対話している際にその対話が異常終了すると、そのクライアントは、Application Server インスタンス 2 の default_island にある OC4J プロセスの 1 つに自動的にリダイレクトされます。このようにして、クライアントはハードウェア障害から保護されます。

図 4-4 ハードウェア障害の例



状態レプリケーション

クライアントがステートフル・アプリケーションである場合、その状態は同じアイランド内でのみレプリケートされます。前述の例では、アイランドが 1 つなので、アプリケーションの状態は保持されます。

パフォーマンスを向上させるために、状態レプリケーションを複数のアイランド間で割り振ることができます。ただし、これらのアイランド内でも、ハードウェア障害とソフトウェア障害を防止する必要があります。

ソフトウェア障害およびハードウェア障害の防止に最適で、しかも最小の OC4J プロセス数で状態を維持できる方法として、同じアイランド内の複数のホストに 1 つ以上の OC4J プロセスを設定する方法があります。たとえば、home インスタンス内に Application Server インスタンス 1 および 2 がある場合、各 Application Server インスタンスの default_island で 1 つの OC4J プロセスを設定します。こうすることで、ソフトウェア障害およびハードウェア障害から保護されるため、いずれの障害が発生してもクライアントは状態を維持できます。

- OC4J プロセスの 1 つが失敗した場合、クライアントのリクエストはそのアイランド内の別の OC4J プロセスにリダイレクトされます。その状態は保持され、クライアント側には異常の発生は認識されません。
- Application Server 1 が異常終了した場合、クライアントは Application Server 2 の default_island 内にある OC4J プロセスにリダイレクトされます。その状態は保持され、クライアント側には異常の発生は認識されません。

需要が増えるにつれて、さらに多くの OC4J プロセスを設定することになります。パフォーマンスが低下するのを防ぐには、その OC4J プロセスをいくつかの異なるアイランドに振り分けます。たとえば、2 つのホスト上で 15 の OC4J プロセスがハードウェアを効率的に利用していて、クライアントの要求にも適切に応えている場合、これらのプロセスを 2 つ以上のアイランドに分けることもできます。15 の OC4J プロセスを 3 つのアイランドに分ける場合の例を次に示します。

アイランド名	Application Server 1	Application Server 2
default_island	2	3
second_island	2	3
third_island	3	2

- Application Server 1 がインストールされているホストで処理可能な OC4J プロセスは 7 つ、Application Server 2 がインストールされているホストで処理可能な OC4J プロセスは 8 つです。
- ソフトウェア障害およびハードウェア障害から保護するために、複数のホストにまたがる各アイランドに OC4J プロセスを 1 つ以上設定します。
- 3 つのアイランドの間で状態レプリケーションを割り振ることで、パフォーマンスが最大になります。

OC4J クラスタの設定

次の各項で、クラスタ内の OC4J 要素（EJB およびサーブレット）を管理する方法を説明します。Oracle Application Server クラスタを作成および変更する方法の手順は、『Oracle Application Server 高可用性ガイド』を参照してください。

- OC4J インスタンスの設定
- OC4J インスタンス固有のパラメータの設定

注意： Enterprise Manager を使用するかわりに、DCM コマンドライン・ツールを使用してクラスタを作成し、Application Server インスタンスをそのクラスタに追加して、クラスタを管理できます。DCM コマンドライン・ツールについては、『Distributed Configuration Management 管理者ガイド』を参照してください。

OC4J インスタンスの設定

次の項では、OC4J インスタンスをクラスタリングするための設定方法について説明します。

- アイランドおよびプロセスの設定
- Web アプリケーションの状態レプリケーションの設定
- EJB アプリケーションの状態レプリケーションの設定

アイランドおよびプロセスの設定

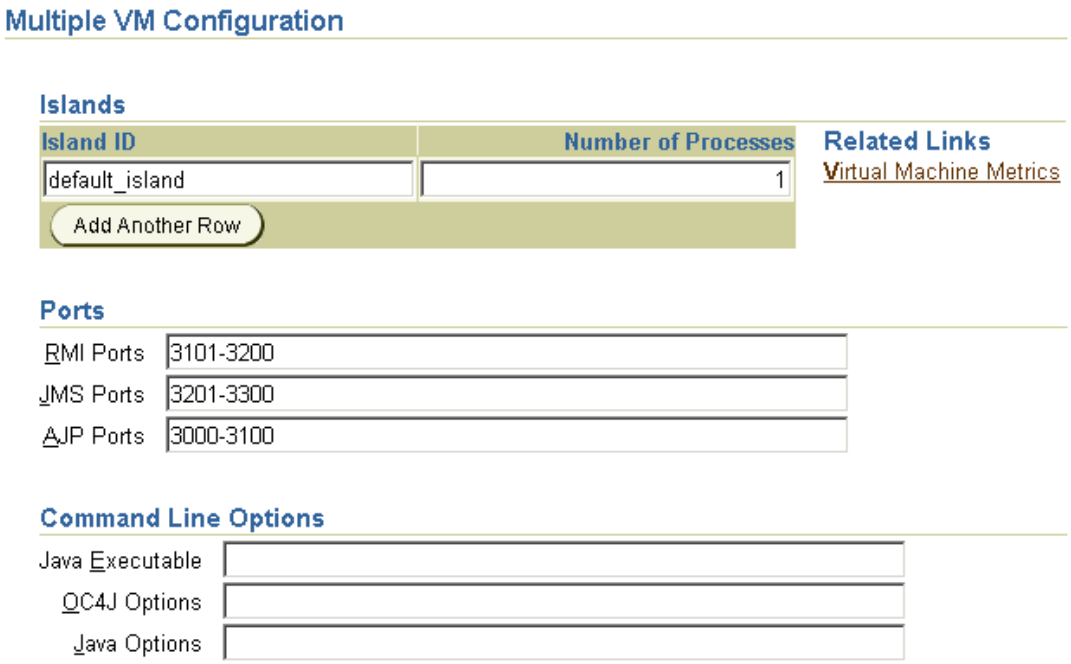
アイランドおよび各アイランドに含まれているプロセスの数を変更するには、次の手順に従います。

1. OC4J ホームページから「管理」ページを選択します。
2. 「インスタンス・プロパティ」列で「サーバー・プロパティ」を選択します。

3. 画面をスクロールして「複数 VM 構成」セクションに移動します。このセクションでアイランドを定義し、各アイランド内の Application Server インスタンス上で起動される OC4J プロセスの数を定義します。
4. クラスタ内にこの OC4J インスタンス用のアイランドを作成する場合は、**「行の追加」**をクリックします。「アイランド ID」フィールドに、各アイランドの名前を指定します。「プロセス数」フィールドに、各アイランド内で起動される OC4J プロセスの数を指定します。

図 4-5 は、「複数仮想マシン構成」セクションです。

図 4-5 アイランドおよびプロセスの設定



Web アプリケーションの状態レプリケーションの設定

ステートフル・アプリケーションの状態レプリケーションの設定は、Web アプリケーションの場合と EJB アプリケーションの場合では異なります。Web アプリケーションの状態レプリケーションを設定するには、次の手順に従います。

1. OC4J ホームページから「管理」ページを選択します。
2. 「インスタンス・プロパティ」列で「レプリケーション・プロパティ」を選択します。
3. 「Web アプリケーション」セクションまでスクロールします。図 4-6 は、このセクションを示しています。
4. 「セッション状態レプリケーション」チェック・ボックスを選択します。
5. オプションとして、マルチキャスト・ホストの IP アドレスおよびポート番号を指定できます。マルチキャスト・アドレスのホストとポートを指定しない場合、デフォルトでホストの IP アドレスは 230.230.0.1 に、ポート番号は 9127 に設定されます。ホストの IP アドレスは、224.0.0.2 ～ 239.255.255.255 で指定する必要があります。HTTP 用と EJB 用のマルチキャスト・アドレスには、それぞれ異なるマルチキャスト・アドレスを使用してください。

注意： マルチキャスト・アドレスを選択するときは、そのアドレスが `http://www.iana.org/assignments/multicast-addresses` に示されているアドレスと衝突しないように注意してください。また、アドレスの下位 23 ビットがローカル・ネットワーク制御ブロック（`224.0.0.0` ～ `224.0.0.255`）と同じである場合は、衝突が発生することがあります。これを回避するには、下位 23 ビットがこの範囲のアドレスと同一でないアドレスを指定します。

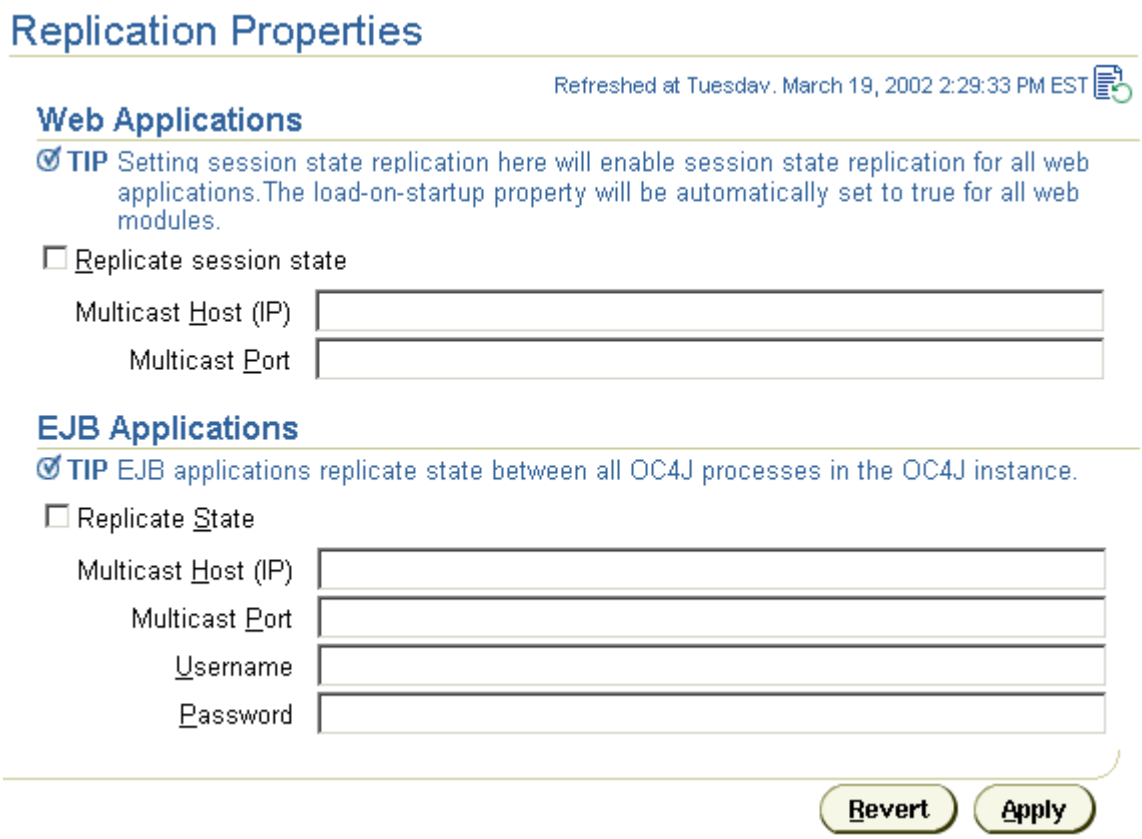
---

6. すべての Web アプリケーションで、すべての `web.xml` ファイルに `<distributable/>` タグを追加します。シリアル化可能な Web アプリケーションでは、このタグを `web.xml` ファイルに追加する必要があります。

   このタグを `web.xml` に追加する例を次に示します。

```
<web-app>
  <distributable/>
  <servlet>
  ...
  </servlet>
</web-app>
```

図 4-6 Web の状態レプリケーションの設定



EJB アプリケーションの状態レプリケーションの設定

EJB クラスタを作成するには、クラスタ内に含める OC4J インスタンスを指定し、それぞれを同じマルチキャスト・アドレス、ユーザー名およびパスワードで設定し、クラスタリングする EJB をクラスタ内の各ノードにデプロイします。

HTTP クラスタリングとは異なり、クラスタに含まれる EJB をアイランド内でサブグループ化することはできません。クラスタ内の EJB は、すべて 1 つのグループに入ります。また、クラスタリングされるのは Session Bean のみです。

すべての Bean の状態が、マルチキャスト・トピックを使用して、各メソッド・コールの終わりでクラスタ内の全ノードに対してレプリケートされます。EJB クラスタに含まれる各ノードは、同じマルチキャスト・アドレスを使用するように設定されます。

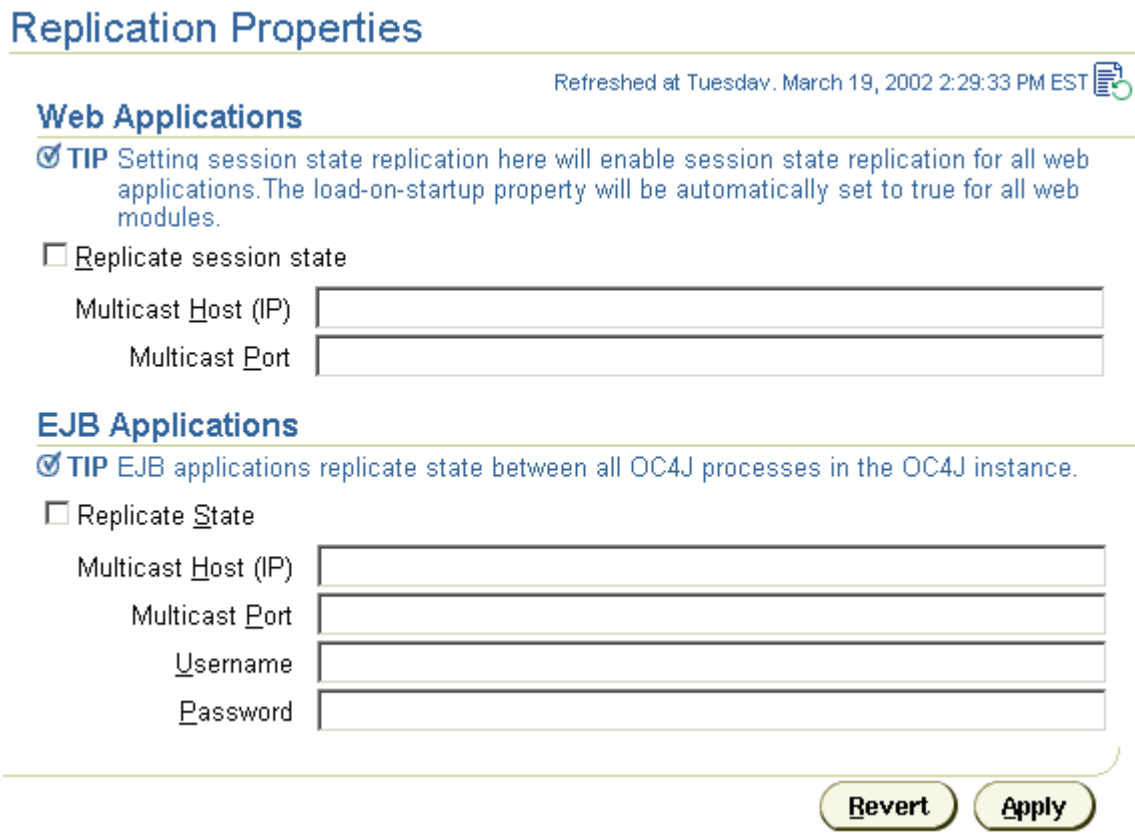
EJB オブジェクトの状態がクラスタ内でレプリケートされる方法を理解するための概念は、『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。EJB レプリケーションを設定するには、次のようにします。

1. OC4J ホームページから「管理」ページを選択します。
2. 「インスタンス・プロパティ」列で「レプリケーション・プロパティ」を選択します。
3. 「EJB アプリケーション」セクションまでスクロールします。図 4-7 は、このセクションを示しています。
4. 「セッション状態レプリケーション」チェック・ボックスを選択します。
5. クラスタ内の他のホストへの認証に使用される、ユーザー名およびパスワードを指定します。そのユーザー名およびパスワードが、クラスタ内の他のホスト間で異なっている場合は、通信に失敗します。マルチキャスト・アドレスの中には、ユーザー名とパスワードの組合せを複数指定できます。ユーザー名とパスワードの組合せが同じクラスタは、一意のクラスタとみなされます。
6. オプションとして、マルチキャスト・ホストの IP アドレスおよびポート番号を指定できます。マルチキャスト・アドレスのホストとポートを指定しない場合、デフォルトでホストの IP アドレスは 230.230.0.1 に、ポート番号は 23791 に設定されます。ホストの IP アドレスは、224.0.0.2 ～ 239.255.255.255 で指定する必要があります。HTTP 用と EJB 用のマルチキャスト・アドレスには、それぞれ異なるマルチキャスト・アドレスを使用してください。

   注意： マルチキャスト・アドレスを選択するときは、そのアドレスが `http://www.iana.org/assignments/multicast-addresses` に示されているアドレスと衝突しないように注意してください。また、アドレスの下位 23 ビットがローカル・ネットワーク制御ブロック（`224.0.0.0` ～ `224.0.0.255`）と同じである場合は、衝突が発生することがあります。これを回避するには、下位 23 ビットがこの範囲のアドレスと同一でないアドレスを指定します。

7. OC4J インスタンスが存在するホスト名を「RMI サーバー・ホスト」フィールドに指定します。
8. JAR ファイルの中の `orion-ejb-jar.xml` ファイル内に EJB のレプリケーションのタイプを設定します。詳細は、4-12 ページの「アプリケーション JAR におけるステートフル Session Bean のレプリケーションの設定」を参照してください。レプリケーションのタイプは、デプロイ前に `orion-ejb-jar.xml` ファイルに構成するか、デプロイ後に Enterprise Manager の画面から追加できます。デプロイ後にこの設定を追加する場合は、アプリケーションのページから JAR ファイルにドリルダウンします。

図 4-7 EJB の状態レプリケーションの設定



アプリケーション JAR におけるステートフル Session Bean のレプリケーションの設定 ステートフル Session Bean の場合は、**`orion-ejb-jar.xml`** ファイルを変更して、状態レプリケーション設定を追加します。ステートフル Session Bean のレプリケーション・タイプは、Bean デプロイメント・ディスクリプタ内で設定するため、各 Bean では異なるタイプのレプリケーションを使用できます。

ステートフル Session Bean の場合は、ノード間で状態をレプリケートする必要があります。実際、ステートフル Session Bean ではその状態をすべてノード間で送信する必要があり、パフォーマンスに大きな影響を及ぼす可能性があります。このため、次のレプリケーション・モードを使用して、レプリケーションがパフォーマンスに与える負荷を制御する方法を決定できます。

JVM 終了時のレプリケーション ステートフル Session Bean の状態は、JVM の終了時にクラスタ内の同一マルチキャスト・アドレスを持つ他の 1 つのノードにのみレプリケートされます。このモードでは、JDK 1.3 のシャットダウン・フックを使用するため、JVM 1.3 以降を使用する必要があります。このモードは、状態のレプリケートが 1 回のみのため、最もパフォーマンスの高いモードです。ただし、次の理由により、信頼性はあまり高くありません。

- 突然に電源が切れた場合は、状態がレプリケートされません。
- Bean の状態はいつでも 1 ノード上にしかないため、障害の程度は、そのノードの障害の程度に匹敵します。

`orion-ejb-jar.xml` ファイルで、`<session-deployment>` タグの `replication` 属性を "`VMTermination`" に設定します。次のようになります。

```
<session-deployment replication="VMTermination" .../>
```

コール終了時のレプリケーション ステートフル Session Bean の状態は、各 EJB メソッド・コールの終了時にクラスタ内の同一マルチキャスト・アドレスを持つ全ノードにレプリケートされます。ノードの電源が切れた場合でも、状態はすでにレプリケートされています。JVM 終了時レプリケーション・モードでは、電源が切れた場合、状態レプリケーションは保証されません。

`orion-ejb-jar.xml` ファイルで、`<session-deployment>` タグの `replication` 属性を `"endOfCall"` に設定します。次のようになります。

```
<session-deployment replication="EndOfCall" .../>
```

OC4J インスタンス固有のパラメータの設定

クラスタの管理機能により、設定がクラスタ内のすべての Application Server インスタンスにわたってレプリケートされ、クラスタ・レベルの設定として定義されます。ただし、一部のパラメータについては、OC4J インスタンスごとに個別に設定する必要があります。これらのパラメータを、OC4J インスタンス固有のパラメータと呼びます。

次に示すパラメータは、OC4J インスタンス固有のパラメータであり、クラスタ全体にわたってレプリケートされません。これらのパラメータは、Application Server インスタンスごとに変更する必要があります。

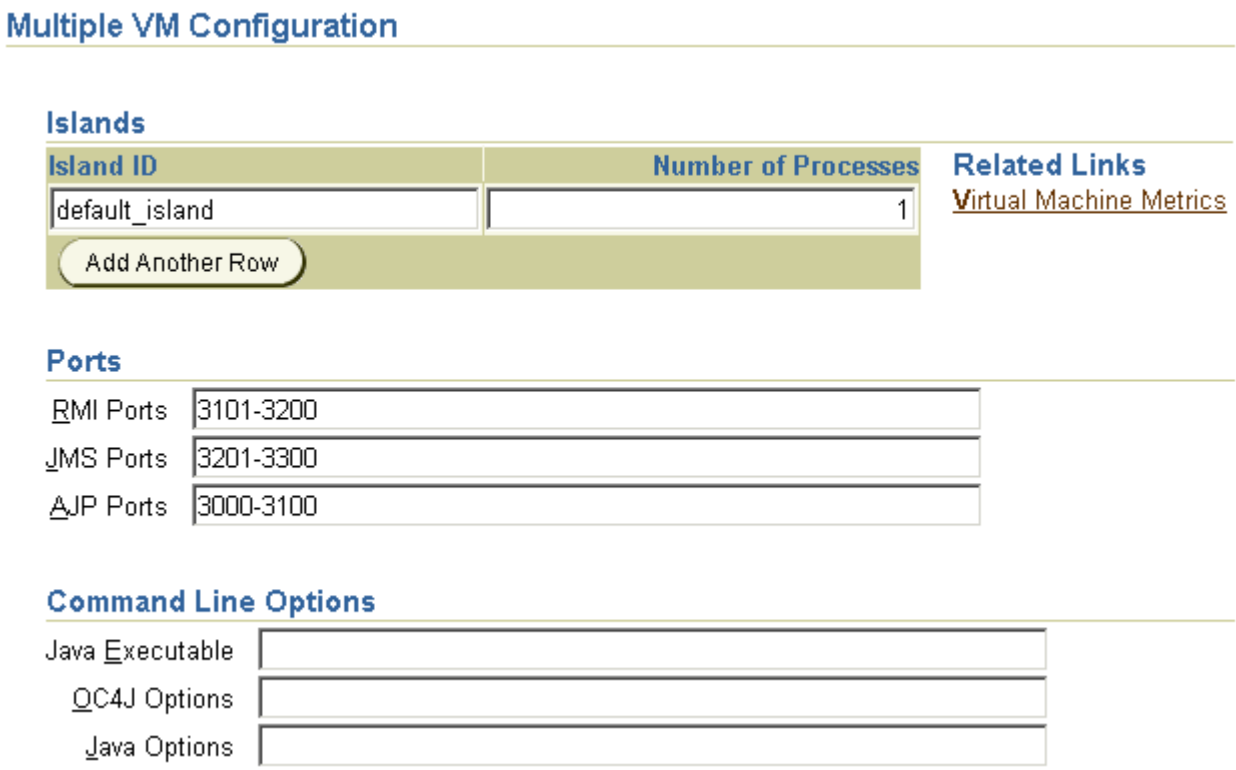
各 OC4J インスタンス内のインスタンス固有のパラメータは次のとおりです。

- アイランド
- OC4J プロセスの数
- ポート番号
- コマンドライン・オプション

他のパラメータはすべてクラスタ全体のパラメータで、クラスタ全体にレプリケートされます。

図 4-8 は、これらのパラメータを変更するセクションです。このセクションは、「管理」ページの「サーバー・プロパティ」にあります。

図 4-8 レプリケートされない設定



A

OC4J のトラブルシューティング

この付録では、OC4J 使用時に発生する可能性のある一般的な問題と、それらの解決策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、一般的な問題とその解決策について説明します。この項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` 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" />
```

OracleAS JMS がアクティブなとき、OC4J を異常終了後に再起動できない

問題

永続性が OracleAS JMS で有効な場合、JMS サーバーでは永続キューまたは永続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。JVM が `kill -9` などにより異常終了しても、ロック・ファイルは削除されません。これが原因で OC4J を再起動できなくなります。

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致しない

問題

フェイルオーバーは、OC4J のインスタンス A からインスタンス B に行われても、再び B から A には行われないというのが一般的です。

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移入されていることを確認します。

OPMN 管理 OC4J では、グローバル・レベルでのクラスタリングのみをサポートし、アプリケーションまたはモジュール・レベルではサポートしていません。

OC4J 専用に JDK の非認証バージョンを使用

問題

この場合、すべての Oracle Application Server コンポーネントでの使用を認証されているバージョンではなく、OC4J でその JDK の後続バージョンを使用することになります。しかし、その JDK の後続バージョンをすべてのコンポーネントでグローバルに使用すると、認証違反の危険性が高まります。

解決策

非認証の JDK の後続バージョンを OC4J 専用に使用するには、opmn.xml 構成ファイルの <java-bin> 要素でそ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 実行時に java.lang.OutOfMemory エラーが発生

問題

このエラーは、Java インスタンスのヒープ・サイズが、OC4J 内で実行中のアプリケーションで必要なメモリーより少なくなっていることを意味します。

解決策

次のように、opmn.xml ファイルの <java-option> 要素で -Xmx を必要な量のメモリーに設定して、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 起動時にシステム・プロパティを次のように設定できます。

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

UNIX または Linux で実行している場合は、ulimit の設定により、JVM プロセスでこの多量のメモリーを割り当てられることを確認します。

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し、接続がステートフル・ファイアウォールの非アクティブのタイムアウトを超えた期間アイドル状態になっている場合、AJP を介して送られたパッケージは拒否されます。

しかし、AJP ソケットは閉じられていないので、ソケットが開いているかぎり、ワーカー・スレッドはそのソケットに接続されていて、スレッド・プールに戻されることはありません。OC4J ではさらに多くのスレッドを作成し続け、最終的にはシステム・リソースが不足します。

解決策

OHS TCP 接続は、ファイアウォールのタイムアウト問題を避けるために、アライブの状態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` 構成ファイルまたは `mod_oc4j.conf` 構成ファイルで、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の時間を秒単位で設定します。

```
Oc4jUserKeepalive  on
```

```
Oc4jConnTimeout  12000（または類似の値）
```

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 OC4J でデフォルトの RMI ポートを介して EJB リソースにアクセスできない

問題

OC4J では、Oracle Application Server のコンポーネントとして実行されている場合、デフォルトの RMI ポートを介して 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` ファイルの `<port>` 要素で指定されているか、アプリケーション・クライアントから動的 `opmn:ormi` 参照を使用して指定されます。

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

停止時間の短いインクリメンタル・コレクタを使用することを検討してください。これは、メジャー・コレクションの一部をマイナー・コレクションごとに行い、メジャー・ガベージ・コレクションによる長い停止を回避します。このコレクタ（トレイン・コレクタとも呼ばれる）では、マイナー・コレクションごとに、古い世代（通常 1 回のメジャー・コレクションでコレクトされるオブジェクトを保持するメモリー・プール）の一部をコレクトします。その結果、多数のマイナー・コレクションに短い停止時間が分散することになります。

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遅くなるので注意してください。

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、次のように設定します。

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

ガベージ・コレクションのチューニングの詳細は、`http://java.sun.com/docs/hotspot/gc1.4.2/` で入手可能な『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るパフォーマンスの低下

問題

OC4J プロセス・メモリーがプログラムの実行中に一貫して増加する場合、グローバルな `application.xml` ファイルの無効なシンボリック・リンクに対する参照があることが考えられます。

この問題は一般に、Java オブジェクト・メモリーではなく、C ヒープが増加するという特徴があり、従来の Java オブジェクト・メモリー・リークとともに見られます。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。これらのリンクが無効の場合、C ヒープは増え続け、OC4J がメモリー不足になります。

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。

さらに、OC4J でロードのために構成される JAR ファイル数を最小限に抑えます。使用されていないすべての JAR ファイルを、構成と、OC4J で検索するように設定されているディレクトリから削除します。OC4J ではすべての JAR ファイルでクラスとリソースを検索するため、ファイル・キャッシュで余分なメモリーとプロセッサ時間が使用される原因になります。

`application.xml` ファイルの `<library>` 要素で、必要な JAR ファイルと ZIP ファイルを、それらが置かれているディレクトリではなく、個々のファイルとして指定すれば、ロード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。

さらにヘルプが必要な場合

その他の解決策は、オラクル社の次のサポート用 Web サイトで検索できます。

- 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）

それでも問題の解決策が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
- 構成およびデプロイの例
- OC4J のコマンドライン・オプションおよびシステム・プロパティ

これらの項では、XML ファイルを変更する方法を主に説明しています。XML ファイルの変更は、すべて Enterprise Manager で行ってください。1 つのノードで XML ファイルを変更しないでください。

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 の構成に必要な次の XML ファイルについて説明します。

- server.xml
- default-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。このファイルで、次のことを指定します。

- 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
- クラスタ ID
- 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指定したエントリを追加します。複数の Web サイトを使用できます。`default-web-site.xml` ファイルはデフォルトの Web サイトを定義するため、これら XML ファイルのうち 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="./default-web-site.xml" />
<web-site path="./another-web-site.xml" />
```

> **注意：** 指定されているパスは、`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。

default-web-site.xml

このファイルには、Web 用の構成が含まれています。default-web-site.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、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
- 1 つ以上のホストからのサイト・アクセスの制限

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、次のことを指定します。

- 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

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 ファイルで、次のことを指定します。

- 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
- スレッド・ポリシー
- ガベージ・コレクションの粒度

- 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 を別のデータベース・システムで動作させるために使用します。OC4J には、タイプ・マッピングや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。ご使用の DBMS 用にこれらを編集することも、新規スキーマを作成することもできます。

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、次のことを指定します。

- 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
- クラスタリングの設定
- 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 ファイルは手動で編集できます。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。

- ライブラリ・パスへのファイルの追加
- 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。

- タイムアウトの設定
- トランザクション再試行の設定
- セッション永続性の設定
- トランザクション分離の設定
- 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
- 最大および最小インスタンス・プールの設定
- 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、次のものが含まれています。

- 自動再ロード（変更チェック時間間隔を含む）
- 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
- クラスタリング
- 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 OC4J 固有のデプロイメント・ファイルは、クライアント・アプリケーション用です。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
- クラスタ ID
- 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、次のことを指定します。

- `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で指定できます。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 サーバーの起動時に、<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します。false に設定すると、アプリケーションの auto-start 属性が true に設定されていなければ、アプリケーションは起動しません。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": この属性はスタンドアロンの OC4J にのみ使用されます。
- 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 コンテナがリカバリにどのように対処するかを指定します。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（開発環境において、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負荷が大き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。何も指定されていない場合（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。一部のコンパイラでは、追加の CLASSPATH が必要となります（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、3-28 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つ間に、スレッドをアライブ（アイドル）にしておく時間（ミリ秒単位）。このタイムアウト値は、アイドル・スレッドをアライブにしておく時間を指定します。タイムアウトに達するとスレッドは破棄されます。最小時間は 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つ間、接続スレッドをアライブ（アイドル）にしておく時間（ミリ秒単位）。このタイムアウト値は、アイドル・スレッドをアライブにしておく時間を指定します。タイムアウトに達するとスレッドは破棄されます。最小時間は 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プロセス外の 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。有効なコンパイラ名は、次のとおりです。
  - プロセス内コンパイラ : modern、classic、javac または ojc
  - プロセス外コンパイラ（フォーク）: modern、javac、ojc または jikes

    これら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、コンパイラはプロセス内で実行されます。false の場合、コンパイラはプロセス外で実行されます。ほとんどのコンパイラは、プロセス内とプロセス外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、プロセス外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、プロセス内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。JDK の bin ディレクトリでこのコンパイラが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 `logN.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` が開かれ、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、新しいファイルが開か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。絶対パスか、または構成 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> 要素は、OC4J から値が 0 ～ 100 のメトリックを mod_oc4j に送信し、mod_oc4j で使用可能な一連の OC4J インスタンスに対して受信リクエストのロード・バランシングを行うためのルーティングを決定できるように指定します。送られるメトリックの値は相対値のみで、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である（ビジーでない）ことを意味します。メトリック・ロード・バランシングの構成を行うと、mod_oc4j ではまず大きな値を持つ OC4J インスタンスにルーティングします。

OC4J から mod_oc4j に送られるメトリックは、メトリックベース・ロード・バランシングが mod_oc4j に指定されていて、OC4J が Oracle Application Server 環境で稼働している場合にのみ使用されます。

メトリックベース・ロード・バランシングを mod_oc4j で指定し、server.xml の <metric-collector> 要素を指定しない場合、mod_oc4j では OC4J がメトリックを送信することを求めるのに対し、OC4J ではメトリックを送信しません。この場合、mod_oc4j からは次の警告メッセージが報告されます。

```
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 では各 OC4J プロセスの値に 50 を使用し、続行します。

同様に、server.xml で <metric-collector> 要素を指定しながら、mod_oc4j ではメトリックベース・ロード・バランシングを指定しなかった場合は、OC4J からはメトリックが送信されますが、mod_oc4j はメトリックを受信するように設定されていません。この場合、mod_oc4j ではメトリックを無視し、ロード・バランシング用に構成されたメソッドを使用します。ロード・バランシングのメソッドは、Oc4jSelectMethod で指定します。Oc4jSelectMethod が指定されていない場合、mod_oc4j ではデフォルトのラウンドロビンを使用します。

<metric-collector> 要素には、1 つの属性、classname が必要です。この属性は、サーバー・レベル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 属性には oracle.oc4j.server.DMSMetricCollector を使用します。DMSMetricCollector インスタンスには、複数のパラメータが必要です。

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>
```

mod_oc4j でのメトリック・ベース・ロード・バランシング使用の詳細は、『Oracle HTTP Server 管理者ガイド』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」の章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（ミリ秒単位）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・ファイルの概要を説明します。

<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 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

このセクションでは、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。

<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プロセス内で開始する場合（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（プロセス内で）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: プロセス内（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、（リモートまたはクラスタ内にある）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: 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。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。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 を検索します。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 のようになります。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: 認証メソッドを 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 で説明しています。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 が開かれ、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、新しいファイルが開か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。絶対パスか、または構成 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、JAZN LDAP UserManager を全体的な UserManager として使用し、JAZN 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 タグの値 *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 *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・タグ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

このセクションでは、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・ファイルの概要を説明します。

<orion-application-client> の要素の説明

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。

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 デモの XML 構成ファイルの変更方法は、2-11 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。

この例では、myapp アプリケーションに、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 構成ファイルの場所、Java クライアント・ファイル、および JSP ファイルを表すツリー構造を、次の「構成ファイルの例」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。

構成ファイルの例

各種構成ファイルの例は次のとおりです。

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ものを指定します。

- アプリケーションのルート・コンテキスト（http://<host>:<port>/j2ee/myapp）用に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

デプロイ・ウィザード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` 変数を `FALSE` に変更します。

> **注意：** `auto-start` を `FALSE` に設定すると、Enterprise Manager を使用して手動でアプリケーションを起動することができます。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` 変数は、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` 変数は、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。

default-web-site.xml の追加

デプロイ・ウィザードは、Web アプリケーションのルート・コンテキストを定義し、Web コンテキストをバインドし、次の行を `default-web-site.xml` ファイルに追加します。

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

- `name` 変数は、`.WAR` 拡張子を持たない WAR ファイルの名前です。

- root 変数は、Web サイト外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://<host>:7777/j2ee" と定義する場合、アプリケーションを起動するには、ブラウザで "http://<host>:7777/j2ee/myapp" を指定します。

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=
  opmn:ormi://<opmnhost>:<oc4j_instance>:7777/j2ee/myapp
  java.naming.security.principal=admin
  java.naming.security.credentials=welcome
  ```

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/jndi.jar
../../../j2ee/home/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
```

OC4J のコマンドライン・オプションおよびシステム・プロパティ

OC4J の起動の前に、OC4J コマンドラインにシステム・プロパティとコマンドライン・オプションを設定できます。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、`-Dhttp.session.debug` などです。コマンドライン・オプションは、すべて先頭にハイフン（`-`）が付きます。たとえば、`-help` などです。

- 表 B-2 は、OC4J のコマンドライン・オプションの詳細です。
- 表 B-3 は、一般的なシステム・プロパティを示します。
- 表 B-4 は、デバッグ用プロパティを示します。

3-2 ページの「サーバー・プロパティの構成」の説明と図 B-1「OC4J インスタンスのサーバー・プロパティを変更する Enterprise Manager コンソール」に示されているように、`-D` システム・プロパティは「Java オプション」行に入力し、OC4J コマンドライン・オプションは「OC4J オプション」行に入力します。

図 B-1 OC4J インスタンスのサーバー・プロパティを変更する Enterprise Manager コンソール

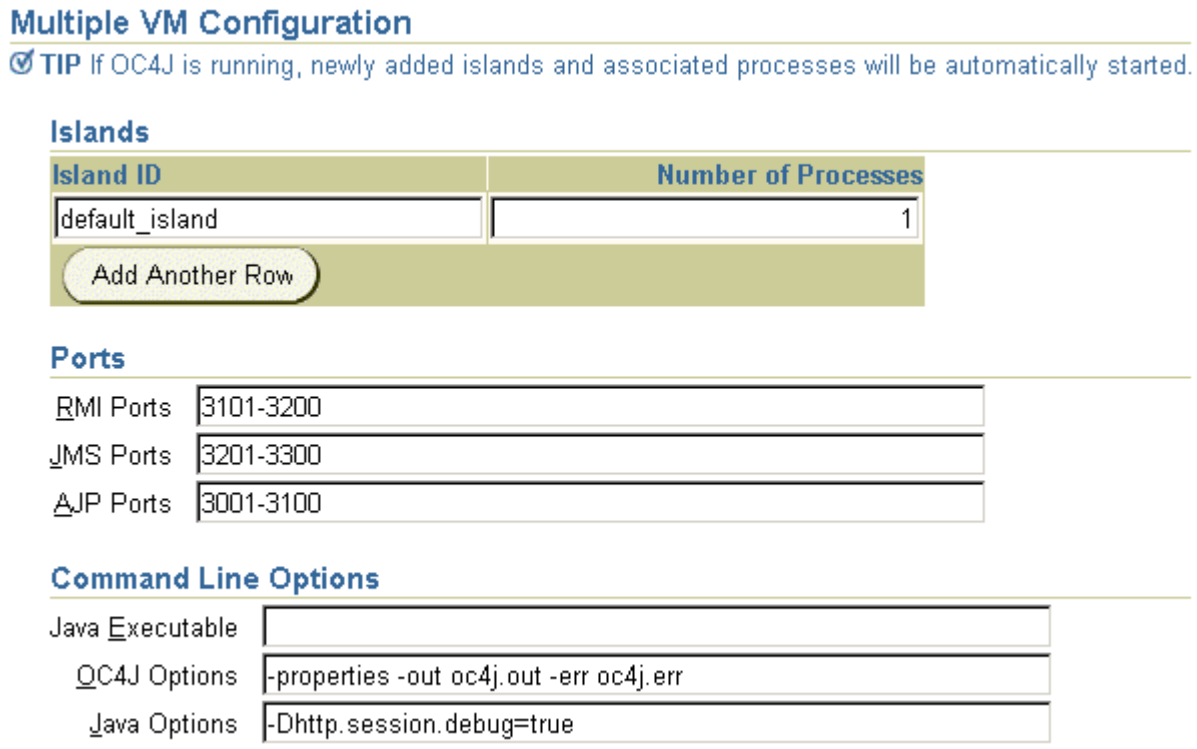


表 B-2 OC4J のコマンドライン・オプション

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、オペレーティング・システムの改行に一致するようにテキスト・ファイルを再作成するなどの操作を行います。
`-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。
`-rewriteXML`	不良 XML ファイルを（プロンプトした後に）できるだけ正確に再作成します。警告 : XML ファイルが破損し、破損の程度が大きいと、再作成のときにデータが失われることがあります。このコマンドは注意して使用してください。
`-out [file]`	標準出力のルーティング先ファイルを指定します。ファイルには、`System.out` に出力されるメッセージが、サーブレット・ロギング・インタフェースにより出力に送られるメッセージとともに含まれています。指定しない場合、すべての出力が標準出力に書き込まれます。 `stdout` ファイルを管理するために設定できるその他のシステム・プロパティについては、B-34 ページの表 B-5「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]`	標準エラー出力のルーティング先ファイルを指定します。指定しない場合、すべてのエラーが標準エラーに書き込まれます。 `stderr` ファイルを管理するために設定できるその他のシステム・プロパティについては、B-34 ページの表 B-5「stdout/stderr アーカイブ管理プロパティ」を参照してください。
`-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。

表 B-2 OC4J のコマンドライン・オプション（続き）

コマンドライン・オプション	説明
-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。このオプションの例は、例 3-4 を参照してください。
-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。

表 B-3 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	OC4J インスタンス用に使用する JDK を示す、JAVA_HOME 環境変数を設定します。
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確認してください。
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。
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 true 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。

表 B-3 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
DefineColumnType= true/false	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。
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つ秒数を示します。これは、パフォーマンスの面では負荷が大き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。
oracle.dms.sensors=[none, normal, heavy, all].	Oracle9*i*AS 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・メトリックを有効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。

表 B-4 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
ajp.debug	デフォルトは false です。true の場合は、AJP リクエスト（ヘッダー、MIME タイプ、URI など）およびレスポンス（ステータス、エラー・メッセージなど）を表示します。
ajp.io.debug	デフォルトは false です。true の場合は、AJP ポスト・データ（ある場合）とクライアントに送信されるレスポンス・データを表示します。
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。
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 メソッドを有効にします。
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。

表 B-4 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
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、Web サービスのデバッグを有効にします。
javax.net.debug=[ssl|all]	ssl の場合は、SSL のデバッグを有効にします。all の場合は、冗長メッセージを使用した SSL のデバッグを有効にします。

デバッグ・プロパティの詳細は、3-36 ページの「OC4J のデバッグ」を参照してください。

表 B-5 stdout/stderr アーカイブ管理プロパティ

デバッグ・プロパティ	説明
stdstream.filesize=<max_file_size>	アーカイブ内のファイルに対して許容される最大サイズ（MB 単位）。この最大サイズに達すると、ファイルは循環します。
stdstream.filenumber=<max_files>	アーカイブとして保持される最大ファイル数。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>	ログ・ファイルが毎日循環する時刻。

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、Oracle 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 Oracle 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 Oracle 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

H

I

J

K

M

N

O

P

R

S

T

W

あ

か

さ

た

な

は

や

ら