Oracle® Application Server Containers for J2EE

セキュリティ・ガイド

10*g*（9.0.4）

部品番号 : B12330-01

2004 年 3 月

ORACLE®

Oracle Application Server Containers for J2EE セキュリティ・ガイド , 10*g*（9.0.4）

部品番号 : B12330-01

原本名 : Oracle Application Server Containers for J2EE Security Guide, 10*g* (9.0.4)

原本部品番号 : B10325-02

原本著者 : Elizabeth Hanes Perry

原本協力者 : Rick Sapir, Alfred Franci

目次

第 I 部　JAAS

2　Oracle Application Server での JAAS の概要

3　JAAS Provider の構成とデプロイ

4 JAAS Provider 管理タスク

5 JAZN Admintool の使用

6 セキュリティと J2EE アプリケーション

7 カスタム LoginModule

8 JAAS と Enterprise Manager

12 EJB のセキュリティ

13 J2EE Connector Architecture のセキュリティ

14 CSIv2 の構成

15 セキュリティのヒント

A JAAS Provider の標準とサンプル

B JAAS Provider のスキーマ

索引

表

図

例

はじめに

このマニュアルでは、Oracle Application Server Containers for J2EE（OC4J）のセキュリティ機能を有効活用する方法について説明します。

この章には、次の項目が含まれます。

対象読者

このマニュアルは、OC4J のセキュリティ機能を理解する必要のある経験豊富な Java 開発者、デプロイヤおよびアプリケーション管理者を対象としています。

このマニュアルの構成

このマニュアルは、次の章で構成されています。

- 第 1 章「概要」
- 第 2 章「Oracle Application Server での JAAS の概要」
- 第 3 章「JAAS Provider の構成とデプロイ」
- 第 4 章「JAAS Provider 管理タスク」
- 第 5 章「JAZN Admintool の使用」
- 第 6 章「セキュリティと J2EE アプリケーション」
- 第 7 章「カスタム LoginModule」
- 第 8 章「JAAS と Enterprise Manager」
- 第 9 章「Java 2 セキュリティ」
- 第 10 章「パスワード管理」
- 第 11 章「クライアント接続用 Oracle HTTPS」
- 第 12 章「EJB のセキュリティ」
- 第 13 章「J2EE Connector Architecture のセキュリティ」
- 第 14 章「CSIv2 の構成」
- 第 15 章「セキュリティのヒント」
- 付録 A「JAAS Provider の標準とサンプル」
- 付録 B「JAAS Provider のスキーマ」

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

- 『Oracle Application Server 10*g* セキュリティ・ガイド』
- 『Oracle Application Server 10*g* 管理者ガイド』
- 『Oracle Identity Management 概要および配置プランニング・ガイド』
- 『Oracle Application Server Certificate Authority 管理者ガイド』
- 『Oracle Application Server Single Sign-On 管理者ガイド』
- 『Oracle Application Server Single Sign-On アプリケーション開発者ガイド』
- 『Oracle Internet Directory 管理者ガイド』
- 『Oracle Internet Directory アプリケーション開発者ガイド』
- 『Oracle Application Server Containers for J2EE サービス・ガイド』
- 『Oracle Application Server Containers for J2EE Enterprise JavaBeans 開発者ガイド』
- 『Oracle Application Server Web Services 開発者ガイド』
- OC4J の Javadoc

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

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

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

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

追加情報は、次の Web サイトから入手できます。

- Sun 社の Java および J2EE Web ページ。特に、Java Authentication and Authorization Service（JAAS）Web サイト（`http://java.sun.com/products/jaas/index-14.html`）

表記規則

このマニュアルでは次の表記規則を使用しています。

規則	意味
. . .	垂直の省略記号は、例に直接関連しない情報が省略されていることを示します。
. . .	水平の省略記号は、例に直接関連しない文またはコマンドの一部が省略されていることを示します。
太字	太字は、本文中で定義されている用語および用語集に記載されている用語を示します。
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。
[ ]	大カッコは、カッコ内の句を 1 つ選択するか、まったく選択しないことを表します。

1

概要

この章には、次の項目が含まれます。

- Java 2 セキュリティ・モデル
- プリンシパルとサブジェクト
- 認証と認可
- セキュアな J2EE アプリケーションの開発

インターネットに接続する中間層環境での Oracle Application Server セキュリティの詳細は、『Oracle Application Server 10*g* セキュリティ・ガイド』を参照してください。Web サービスの詳細は、『Oracle Application Server Web Services 開発者ガイド』を参照してください。

Java 2 セキュリティ・モデル

Java 2 セキュリティ・モデルを使用すると、すべての制限レベルでセキュリティを構成できます。これにより、開発者や管理者にとっては、エンタープライズ・アプレット、コンポーネント、サーブレットおよびアプリケーションのセキュリティの様々な側面の制御が向上します。Java 2 セキュリティ・モデルはケーパビリティ・ベースであり、保護ドメインを設定し、それに対するセキュリティ・ポリシーを設定できます。

Java 2 セキュリティのチュートリアルは、`http://java.sun.com/docs/books/tutorial/security1.2/index.html` を参照してください。Java 2 セキュリティの詳細は、`http://java.sun.com/security` を参照してください。

プリンシパルとサブジェクト

プリンシパル

プリンシパルとは、特定の ID（`frank` という名前のユーザーや `hr` という名前のロールなど）です。プリンシパルは、コンピューティング・サービスに対する認証に成功することにより、サブジェクトに関連付けられます。また、プリンシパルは `java.security.Principal` インタフェースを実装するクラスのインスタンスです。プリンシパル・クラスでは、そのクラスの各インスタンスの一意名を含むネームスペースを定義する必要があります。

サブジェクト

サブジェクトは、ユーザー、コンピュータまたはプロセスなど、コンピューティング・サービスを使用する単体の関連情報のグループを表します。この関連情報には、サブジェクトの ID およびセキュリティ関連属性（パスワードや暗号鍵など）が含まれます。

サブジェクトは複数の ID を持つことができ、プリンシパルはサブジェクト内の ID を表します。サブジェクトは、コンピューティング・サービスに対する認証が成功した時点でプリンシパル（ユーザー `frank`）に関連付けられます。つまり、サブジェクトは ID を証明する証拠（パスワードなど）を提供します。

プリンシパルは、サブジェクトに名前をバインドします。たとえば、ユーザー・サブジェクトであるユーザー `frank` は、次の 2 つのプリンシパルを持つことができます。

- 一方は、プリンシパル `frank doe`（運転免許証に記載されている氏名）をサブジェクトにバインドします。
- 他方は、ID プリンシパル `999-99-9999`（学生証番号）をサブジェクトにバインドします。

どちらのプリンシパルも同じサブジェクトを参照します。

また、サブジェクトはセキュリティ関連属性（資格証明）を持つことができます。秘密暗号鍵など、特別な保護を必要とする重要な資格証明は、秘密資格証明セットに格納されます。公開鍵証明書や Kerberos サーバー・チケットなど、共有することを意図した資格証明は、公開資格証明セットに格納されます。様々な資格証明セットにアクセスして変更するには、それぞれ異なるアクセス権が必要です。

サブジェクトは `javax.security.auth.Subject` クラスで表されます。

アプリケーションは、特定のサブジェクトとして動作するために、メソッド `Subject.doAs(Subject, PrivilegedAction)`（またはそのバリエーションの 1 つ）を起動します。このメソッドは、サブジェクトを現行スレッドの `AccessControlContext` に関連付けてから、指定された要求を実行します。

認証と認可

ソフトウェアのセキュリティは、認証および認可という 2 つの基本概念に依存しています。

- 認証は、サブジェクトの ID と資格証明を設定するプロセスです。

  認証情報はユーザー・リポジトリに格納されます。サブジェクトが J2EE アプリケーションへのアクセスを試行すると、ユーザー・マネージャによりユーザー・リポジトリ内でサブジェクトが検索され、その ID が検証されます。ユーザー・リポジトリには、環境に応じてファイルまたはディレクトリ・サーバーを使用できます。Oracle Internet Directory は、ユーザー・リポジトリの一例です。

  各 J2EE アプリケーションでは、そのアプリケーションを使用できるユーザーが決定されますが、ユーザー・リポジトリを使用してユーザーの ID を認証するのはユーザー・マネージャです。

  OC4J では、複数の異なる認証オプションがサポートされます。詳細は、6-5 ページの「認証環境」を参照してください。

- 認可は、認証済サブジェクトに権限を付与するプロセスです。

  開発者は、アプリケーションの J2EE および OC4J 固有のデプロイメント・ディスクリプタに、サブジェクトに対する認可を指定します。これらのデプロイメント・ディスクリプタは、アプリケーション各部へのアクセスに必要なロールを示します。ロールは、各アプリケーションで様々なオブジェクトへのアクセス権を示すために使用される ID です。OC4J 固有のデプロイメント・ディスクリプタは、論理的なロール、ユーザーおよび OC4J で認識されるグループの間のマッピングを提供します。

安全な通信

安全な通信を行うには、アプリケーションは次の条件を満たす必要があります。

- 安全な通信 : ネットワーク経由で送信されるデータを第三者が不正傍受、読取りまたは改変することはできません。OC4J では、Secure Sockets Layer 経由の HTTP プロトコルを使用して安全な通信がサポートされます。
- ネットワーク認証 : クライアントとサーバーは、ネットワーク上で相互を認証できる必要があります。これは、デジタル証明、シングル・サインオンまたはユーザー名 / パスワードの組合せを使用して実現されます。
- ID の伝播 : あるクライアントが自身の ID を使用して他のクライアントのエージェントとして動作できるようにします。

Secure Sockets Layer

Secure Sockets Layer（SSL）は、機密性、暗号化、認証およびデータ整合性を提供する業界標準の Point-to-Point プロトコルです。SSL は多数のプロトコルで使用されますが、OC4J では、HTTP ブラウザ・プロトコルや、OHS と OC4J プロセス間の AJP リンクに使用される場合に最も重要です。

証明書

アプリケーションは、認証および認可情報をネットワークで送信する必要があります。デジタル証明は X.509 バージョン 3 規格で指定されたもので、プリンシパルの認証および認可情報の設定データが含まれています。証明書の内容は次のとおりです。

- 公開鍵インフラストラクチャ（PKI）操作に使用される公開鍵
- 識別情報（名前、会社、国など）
- 証明書の所有者に権限を付与するオプションのデジタル権利

各証明書には、トラスト・ポイントによるデジタル署名が添付されます。証明書に署名するトラスト・ポイントは、VeriSign 社などの認証局、法人または個人のいずれかです。

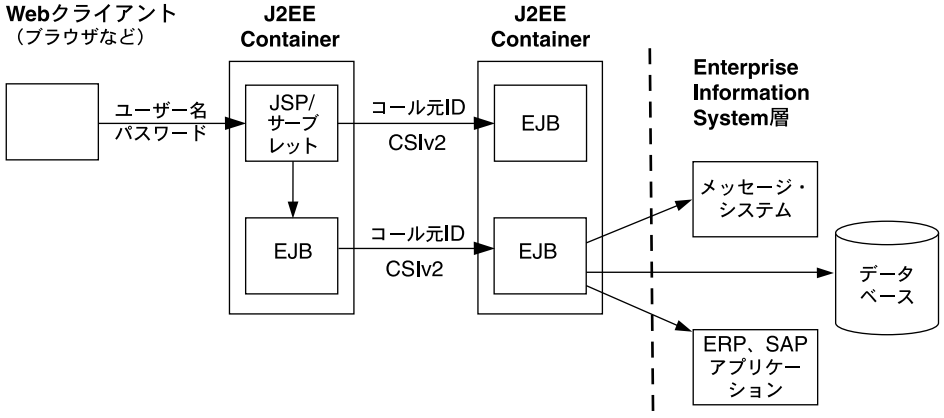
HTTPS

便宜上、このマニュアルでは、SSL 上で実行する HTTP を説明する際の短縮名として HTTPS を使用します。`https:` という URL 接頭辞はありますが、HTTPS というプロトコルは存在しません。

ID の伝播

OC4J は、コンテキスト間でのプリンシパル ID の伝播をサポートしています。図 1-1 に示すように、Web クライアントは自身の ID をサーブレットに対して設定でき、サーブレットはその ID を使用して他の EJB やサーブレットと通信できます。

図 1-1 CSIv2 を使用した ID の伝播



セキュアな J2EE アプリケーションの開発

J2EE ソフトウェアの開発は、開発 - デプロイ - 管理というサイクルで行われます。Oracle JAAS Provider は、このサイクルのデプロイ - 管理部分で重要な役割を果たします。Oracle JAAS Provider は J2EE セキュリティと統合されています。これにより、開発者はセキュリティをプログラムで統合する必要がなくなり、宣言によるセキュリティ・モデルを使用できるため、負荷が軽減されます。

次のリストに、セキュアなアプリケーション開発に固有のタスクに重点を置いて J2EE 開発サイクルの概要を示します。

1. ソフトウェア開発者が Web コンポーネント、Enterprise Bean、アプレット、サーブレットまたはアプリケーション・クライアント、あるいはそのすべてを作成します。

   JAAS Provider はプログラム・インタフェースを提供しますが、開発者はそれを使用せずにコンポーネントを作成できます。

2. アプリケーション・アセンブラは、これらのコンポーネントから構成される Enterprise Archive（EAR）ファイルを作成します。

   このプロセスの一部として、アプリケーション・アセンブラは環境に適切な JAAS Provider オプションを指定します。

3. デプロイヤが EAR を OC4J のインスタンスにインストールします。

   デプロイメント・プロセスの一部として、デプロイヤはユーザーにロールをマップできます。

4. システム管理者がデプロイされたアプリケーションをメンテナンスおよび管理します。

   このタスクには、アプリケーション・ユーザーからの要求に応じた JAAS ロールとユーザーの作成と管理が含まれます。

第 I 部

JAAS

第 I 部では、Oracle Application Server Containers for J2EE（OC4J）の Java Authentication and Authorization（JAAS）について説明します。JAAS は、コア Java セキュリティ・テクノロジの 1 つです。

第 I 部は、次の章で構成されています。

2

Oracle Application Server での JAAS の概要

この章では、Oracle Application Server Containers for J2EE（OC4J）の Oracle Application Server Java Authentication and Authorization Service（JAAS）サポートについて説明します。JAAS を使用すると、アプリケーション開発者は認証サービス、認可サービスおよび委任サービスをアプリケーションと統合できます。

この章には、次の項目が含まれます。

JAAS Provider

OracleAS では、JAAS Provider の実装により JAAS がサポートされます。JAAS Provider により、開発者がアプリケーション環境に統合可能なユーザー認証、認可および委任サービスが実装されます。アプリケーション開発者はこれらのサービスの開発にリソースを費やすかわりに、アプリケーションのプレゼンテーションおよびビジネス・ロジックに重点を置くことができます。

> **注意：** 一部のクラス名とコンポーネント名には JAZN という語が含まれていますが、これは OC4J JAAS Provider を表す短縮名です。

JAAS フレームワークと Java 2 セキュリティ・モデルが JAAS の基盤を形成します。OracleAS JAAS Provider により、JAAS ポリシーのサポートが実装されます。ポリシーには、ファイルの読取りなど、ユーザーによるリソース使用を認可するためのルール（パーミッション）が含まれています。サービスは JAAS を使用してリソース・ユーザーを認証し、アクセス制御を強化できます。JAAS Provider は、Java 2 セキュリティ・モデルを使用する J2SE および J2EE アプリケーションと簡単に統合できます。

プロバイダのタイプ

OC4J JAAS では、2 つの異なるタイプのプロバイダがサポートされます。プロバイダ・タイプごとに、プロバイダ・データをセキュアに集中的に格納、取得および管理するリポジトリが実装されます。このデータは、レルム（ユーザーとロール）および JAAS ポリシー（パーミッション）情報で構成されます。

- XML ベース・プロバイダ

  XML ベース・プロバイダは、XML ファイルへの軽量な情報の格納に使用されます。XML ベース・プロバイダにより、ユーザー、レルムおよびポリシー情報が XML ファイル（通常は `jazn-data.xml`）に格納されます。

  > **注意：** XML ファイルは、LDAP ベースと XML ベースの両方のプロバイダ・タイプでプロパティおよび構成ファイルとして使用されます。ただし、ユーザー、レルムおよびポリシー情報を XML ファイル（通常は `jazn-data.xml`）に格納するのは、XML ベース・プロバイダのみです。

- LDAP ベース・プロバイダ

  LDAP ベース・プロバイダは、情報をディレクトリに集中的に格納するための Lightweight Directory Access Protocol（LDAP）に準拠しています。LDAP ベース・プロバイダでは、ユーザー、レルムおよびポリシー情報は LDAP ベースの Oracle Internet Directory に格納されます。

JAAS とは

JAAS は、アプリケーションでユーザーの認証とアクセス制御の強化に使用される Java パッケージです。JAAS Provider は、JAAS インタフェースの実装です。

JAAS は、既存のコードベース Java 2 セキュリティを補完するように設計されています。JAAS では、標準の Pluggable Authentication Module（PAM）フレームワークの Java バージョンが実装されます。これにより、アプリケーションは認証サービスから独立した状態を維持できます。

JAAS により Java 2 セキュリティ・モデルのアクセス制御アーキテクチャが拡張され、プリンシパルベースの認証がサポートされます。

この項では、次の認証、認可およびユーザー・コミュニティ（レルム）機能に対する JAAS サポートについて説明します。JAAS Provider により、これらの機能の一部が拡張されます。

- ログイン・モジュール認証
- ロール
- レルム
- ポリシーとパーミッション

関連項目：

- 鍵の認証、認可およびユーザー・コミュニティ（レルム）機能を明示的に定義するための OracleAS JAAS Provider による JAAS フレームワークの拡張については、2-7 ページの「JAAS フレームワークの機能」を参照してください。
- JAAS 機能の詳細は、次の Web サイトで JAAS ドキュメントを参照してください。

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

ログイン・モジュール認証

クライアントは、プリンシパル（frank など）をサブジェクトと関連付けるためにアプリケーションにログインします。ログイン・モジュール認証では、ユーザー、ロール、またはコンピューティング・サービスなどのサブジェクトの認証に使用する基本メソッドが LoginContext クラスで提供されます。LoginContext クラスは構成設定を参照し、認証モジュール（ログイン・モジュール）がサブジェクトのアクセス先となる特定のアプリケーションで使用するように構成されているかどうかが判別されます。様々なログイン・モジュールを異なるアプリケーションで構成したり、単一のアプリケーションで複数のログイン・モジュールを使用できます。

LoginContext によりアプリケーション・コードが認証サービスから分離されるため、アプリケーション・コードに影響を与えずにアプリケーションに異なるログイン・モジュールをプラグインできます。

実際の認証は、メソッド LoginContext.login() により実行されます。認証に成功した場合は、LoginContext.getSubject() を起動して認証済サブジェクトを取得できます。実際の認証プロセスでは、複数のログイン・モジュールを使用できます。JAAS フレームワークでは、アプリケーション用に構成されたログイン・モジュールを調整するために 2 段階の認証プロセスが定義されています。

LoginContext からサブジェクトを取得したアプリケーションは、Subject.doAs() または Subject.doAsPrivileged() を起動することでサブジェクトとして操作を実行できます。

ロール

JAAS フレームワークでは、ロールまたはグループは明示的に定義されません。そのかわりに、インタフェース java.security.Principal を使用する具象クラスとして、ロールまたはグループは実装されます。

JAAS フレームワークでは、ロールをロールに付与できるロールベースのアクセス制御（RBAC）によるロール階層のサポート方法は定義されません。

レルム

JAAS フレームワークでは、ユーザー・コミュニティは明示的に定義されません。ただし、J2EE リファレンス実装（RI）により、レルムと呼ばれるユーザー・コミュニティと同様の概念が定義されます。レルムはユーザーとロール（グループ）にアクセス権を提供し、必要に応じて管理機能を提供します。ユーザー・コミュニティ・インスタンスは、実際には認可システムにより内部的にメンテナンスされるレルムです。J2EE の RI Realm API では、サブクラス化を介してユーザー定義レルムがサポートされます。

関連項目：

レルムに対する JAAS Provider 拡張は、4-4 ページの「JAAS Provider のレルムのフレームワーク」を参照してください。

アプリケーション

JAAS フレームワークでは、認可ルールのパーティション化に関するアプリケーションやサブシステムは明示的に定義されません。

ポリシーとパーミッション

ポリシーは、JAAS 認可ルールのリポジトリです。ポリシーには、プリンシパルに対するパーミッションの付与、つまり、権限受領者とその権限受領者に対して付与されるパーミッションが含まれます。

ポリシー情報は JAAS Provider から提供されます。JAAS フレームワークでは、ポリシー管理用の管理 API は定義されません。JAAS Provider から提供される管理 API は Oracle 拡張です。

表 2-1 に、Sun 社によるポリシー・ファイル・パラメータの実装を示します。

表 2-1 ポリシー・ファイルのパラメータ

対象	定義	例
サブジェクト	1 つ以上のプリンシパル	`duke`
コードソース	*`codebase`*, *`signer`*	`http://www.example.com,` *`mysigner`*

Sun 社のポリシーの例

次の例に、Sun 社の JAAS ポリシー・プロバイダ実装により実装される JAAS ポリシー・ファイルの標準的なエントリを示します。

```
grant   CodeBase "http://www.example.com",
        Principal com.sun.security.auth.SolarisPrincipal "duke"
{
        permission java.io.FilePermission "/home/duke", "read, write";
};
```

`www.example.com` からのコードは、ユーザー名 `duke` で `SolarisPrincipal` として実行されます。このコードには、実行コードによる `/home/duke` 内のファイルの読取り / 書込みを許可するパーミッションが付与されています。

XML ベースの例

JAAS の XML ベース・プロバイダは、ポリシー・データをファイル `jazn-data.xml` に格納できます。次の例では、`jazn-data.xml` ファイルで `jazn.com/administrators` に対して、レルム・データ変更、レルム削除およびロール作成のパーミッションを付与しています。

```
<!--JAZN Policy Data -->
<jazn-policy>
  <grant>
     <grantee>
        <principals>
            <principal>
                <realm>jazn.com</realm>
                <type>role</type>
                <class>oracle.security.jazn.spi.xml.XMLRealmRole
                     </class>
                <name>jazn.com/administrators</name>
             </principal>
         </principals>
     </grantee>
     <permissions>
        <permission>
           <class>oracle.security.jazn.policy.AdminPermission</class>
           <name>oracle.security.jazn.realm.
               RealmPermission$jazn.com$modifyrealmmetadata</name>
        </permission>
        <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>
            <name>oracle.security.jazn.realm.
                RealmPermission$jazn.com$droprealm</name>
        </permission>
        <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>
            <name>oracle.security.jazn.realm.RealmPermission$jazn.
                com$createrole</name>
        </permission>
        <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>createrealm</actions>
          </permission>
     </permissions>
  </grant>
</jazn-policy>
```

関連項目：

- `jazn-data.xml` ファイルの詳細は、A-2 ページの「サンプル jazn-data.xml コード」を参照してください。

JAAS フレームワークの機能

表 2-2 に、Oracle Application Server の JAAS Provider により実装される JAAS フレームワークの機能を示します。

表 2-2 JAAS Provider の機能

機能	説明	参照
認証	■ J2EE アプリケーション環境で SSO ログイン認証を使用できるように Oracle Application Server Single Sign-On（SSO）と統合します。 ■ OracleAS Core または Java Edition など、非 SSO 環境向けにデフォルトの `RealmLoginModule` クラスを提供します。 ■ JAAS 準拠のカスタム `LoginModule` をすべてサポートします。	第 6 章「セキュリティと J2EE アプリケーション」
宣言によるモデル	■ `web.xml` などの J2EE デプロイメント・ディスクリプタを JAAS セキュリティと統合します。 ■ プログラムによるモデルもサポートします。	第 3 章「JAAS Provider の構成とデプロイ」
ロールベースのアクセス制御（RBAC）	■ 階層形式のロールのサポートを含め、集中的なロールベースのアクセス制御を提供します。	2-12 ページ「ロールベースのアクセス制御（RBAC）」
レルム	■ ユーザー・コミュニティをベースにユーザーとロール（グループ）を編成します。ユーザーおよびロール管理をサポートするために、Oracle API パッケージ（`oracle.security.jazn.realm`）が提供されます。この API には、`java.security.Principal` からの拡張である `RealmPrincipal` インタフェースが組み込まれており、レルムをユーザーおよびロールに関連付けます。	2-4 ページ「レルム」 4-4 ページ「JAAS Provider のレルムのフレームワーク」
管理	■ コマンドライン・ツール（Admintool）またはプログラム・レベルの API を使用して設定とデータを管理します。 ■ Oracle Internet Directory で集中管理されるプロバイダ・タイプをサポートします。	第 4 章「JAAS Provider 管理タスク」
`JAZNUserManager`	■ XML ベースと LDAP ベースの両方のプロバイダと統合する OC4J の `UserManager` の実装を提供します。	6-3 ページ「OC4J と JAAS Provider」 第 6 章「セキュリティと J2EE アプリケーション」

ユーザー・マネージャ

OC4J セキュリティでは、J2EE アプリケーションにアクセスするユーザーとグループの認証と認可にユーザー・マネージャを使用します。どのユーザー・マネージャを選択するかは、パフォーマンス上とセキュリティ上のニーズによって決まります。

すべての UserManager クラスで com.evermind.security.UserManager インタフェースが実装されます。UserManager クラスでは、createUser()、getUser() および getGroup() などのメソッドを介してユーザー、グループおよびパスワードが管理されます。

OC4J には、JAZNUserManager および XMLUserManager という 2 つの事前定義ユーザー・マネージャが用意されています。JAZNUserManager では、XML ベースと LDAP ベースの両方のプロバイダがサポートされます。JAZNUserManager は JAAS 仕様に準拠しており、Oracle Application Server Single Sign-On および Oracle Internet Directory と統合されているため、JAZNUserManager を使用することをお薦めします。JAZNUserManager はデフォルトのセキュリティ・プロバイダであり、強力で柔軟なセキュリティ制御を提供します。また、UserManager インタフェースを実装する独自のクラスをユーザーが提供することもできます。

注意： カスタム UserManager の作成方法は、http://otn.oracle.com/sample_code/tech/xml/xmlnews/News_Security.html を参照してください。

表 2-3 に、OC4J 提供のユーザー・マネージャを示します。

表 2-3 OC4J のユーザー・マネージャとリポジトリ

ユーザー・マネージャ・クラス	ユーザー・リポジトリ
oracle.security.jazn.oc4j.JAZNUserManager	次の 2 つのタイプがあります。 ■ XML ベース・プロバイダを使用するタイプ : jazn-data.xml ■ LDAP ベース・プロバイダを使用するタイプ : Oracle Internet Directory
com.evermind.server.XMLUserManager	principals.xml ファイル
カスタム・ユーザー・マネージャ	カスタマイズ済のユーザー・リポジトリ

Enterprise Manager を使用して、全アプリケーション用のデフォルト UserManager または特定のアプリケーション用の単一 UserManager を定義する方法は、2-11 ページの「UserManager の指定」を参照してください。

次の各項では、JAZN および XML ユーザー・マネージャについて説明します。

- JAZNUserManager の使用
- XMLUserManager の使用

JAZNUserManager の使用

`JAZNUserManager` クラスは、デフォルトのユーザー・マネージャです。`JAZNUserManager` クラスの主な用途は、JAAS Provider を OC4J のセキュリティ・インフラストラクチャとして活用することです。

OC4J セキュリティでは、XML ベースおよび LDAP ベースの 2 つの JAAS Provider が提供されます。

- XML ベース・プロバイダは、JAAS Provider API の高速で軽量な実装です。このプロバイダ・タイプでは、XML を使用してユーザー名と暗号化されたパスワードが格納されます。ユーザー・リポジトリは、jazn.xml ファイルに指定されたディレクトリにある `jazn-data.xml` ファイルに格納されます。詳細は、第 3 章「JAAS Provider の構成とデプロイ」を参照してください。

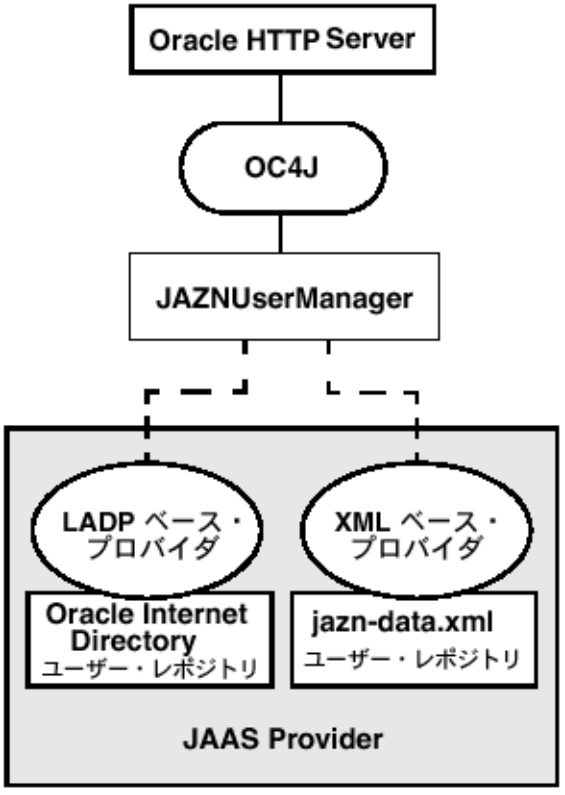
  Enterprise Manager で、JAZN-XML をユーザー・マネージャとして選択します。そのユーザー、ロールおよびグループを、JAZN Admintool を使用して構成します。Admintool の詳細は、第 5 章「JAZN Admintool の使用」を参照してください。

- LDAP ベース・プロバイダは、スケーラブルかつセキュアで、Single Sign-On と統合されたエンタープライズ対応のプロバイダです。また、LDAP ベース・プロバイダは Single Sign-On をサポートする唯一の JAAS Provider です。

  Enterprise Manager で、JAZN-LDAP をユーザー・マネージャとして選択します。そのユーザーとグループの構成には、Oracle Internet Directory の Delegated Administration Service（DAS）を使用します。ユーザー・リポジトリは Oracle Internet Directory インスタンスで、そのインフラストラクチャにアプリケーション・サーバー・インスタンスを関連付ける必要があります。サーバーが Oracle Internet Directory インスタンスに関連付けられていない場合は、LDAP ベース・プロバイダをセキュリティ・オプションとして選択できません。Enterprise Manager の使用方法は、第 8 章「JAAS と Enterprise Manager」を参照してください。

図 2-1 に、OC4J 提供の 2 つの JAAS Provider を示します。

図 2-1 JAZNUserManager クラスの OC4J セキュリティ・アーキテクチャ



XMLUserManager の使用

XMLUserManager クラスは、XML ベース・システム内でユーザー、グループおよびロールを管理する単純なユーザー・マネージャです。このユーザー・マネージャでは、ユーザー・パスワードはクリアテキスト形式で格納されるため、JAZNUserManager ほどセキュアではありません。XMLUserManager 構成情報はすべて、XMLUserManager クラスのユーザー・リポジトリである principals.xml ファイルに格納されます。

注意： XMLUserManager クラスは、下位互換性のみを維持するためにサポートされています。JAAS Provider タイプのいずれかを使用することをお薦めします。

UserManager の指定

ユーザー・マネージャでは、ユーザー名、パスワードおよびユーザー・リポジトリ内の情報を使用してユーザーの ID が検証されます。ユーザー・マネージャには、ユーザー、グループまたはロールの定義が含まれています。デフォルトのユーザー・マネージャは JAZNUserManager です。

ユーザー・マネージャは、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に対して定義できます。

- グローバル・ユーザー・マネージャ : グローバル（デフォルト）ユーザー・マネージャは、個別ユーザー・マネージャが定義されていない全アプリケーションに継承されます。
- 個別ユーザー・マネージャ : このユーザー・マネージャは、単一のアプリケーション専用に定義されます。他のアプリケーションでは使用されません。

注意： 単一の OC4J インスタンス内では、アプリケーション固有の UserManager インスタンスおよびグローバル UserManager インスタンスに対して異なる値を指定できます。これを行うときは、カスタムの UserManager およびオラクル提供の UserManager を併用しないことをお薦めします。アプリケーションおよびグローバル・インスタンスに対して異なるカスタムの UserManager を使用し、アプリケーションおよびグローバル・インスタンスに対して異なるオラクル提供の UserManager を使用できますが、あるインスタンスにカスタムの UserManager を使用し、別のインスタンスにオラクル提供の UserManager を使用することは避けてください。

- アプリケーションをグローバル・アプリケーションから継承せずに別のアプリケーションから継承する場合、アプリケーションの親ユーザー・マネージャは、親アプリケーションで指定された UserManager インスタンスではなくグローバル UserManager インスタンスになることがあります。

アクセス制御のケーパビリティ・モデル

ケーパビリティ・モデルは、認可情報を編成する手段です。JAAS Provider は Java 2 セキュリティ・モデルに準拠し、ケーパビリティ・モデルを使用してパーミッションへのアクセスを制御します。ケーパビリティ・モデルを使用して認可がプリンシパル（次の例ではユーザー frank）に関連付けられます。表 2-4 に、ユーザー frank が使用を認可されているパーミッションを示します。

表 2-4 ユーザーの権限

ユーザー	付与されているファイル・アクセス権
frank	/home/user ディレクトリ内のファイル salaries.txt に対する読取り権限と書込み権限

ユーザー frank がログインし、正常に認証を受けると、表 2-4 に示すパーミッションが（LDAP ベースの Oracle Internet Directory であるか XML ベースのプロバイダであるかに関係なく）JAAS Provider から取得され、ユーザー frank に付与されます。ユーザー frank は、これらのパーミッションにより、操作を実行できるようになります。

ロールベースのアクセス制御（RBAC）

RBAC により、ロールにパーミッションを割り当てることができます。ユーザーにパーミッションを付与するには、そのユーザーを適切なロールのメンバーにします。RBAC のサポートは、重要な JAAS 機能です。この項では、次の RBAC 機能について説明します。

- ロール階層
- ロールのアクティブ化

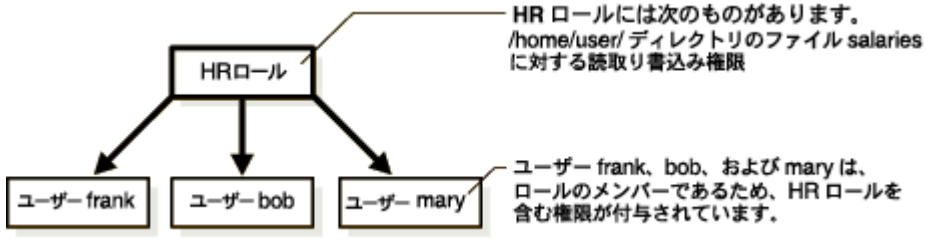
ロール階層

RBAC により、ユーザーにパーミッションを直接割り当てることによって生じる管理上の問題が単純化されます。複数のユーザーにパーミッションを直接割り当てる操作は、管理タスクの大半を占める可能性があります。複数のユーザーが特定のパーミッションへのアクセスを必要としなくなった場合は、そのパーミッションを各ユーザーから個別に削除する必要があります。

パーミッションをユーザーに直接割り当てるかわりに、ロールに割り当て、ユーザーをそのロールのメンバーにすることで各ユーザーにパーミッションを付与します。1 人のユーザーに複数のロールを付与できます。また、あるロールを別のロールに付与してロール階層を形

成することもでき、管理者はこれを社内のセキュリティ・ポリシー・モデルを作成するためのツールとして使用できます。図 2-2 に、ロールベースのアクセス制御の例を示します。

図 2-2 ロールベースのアクセス制御



ユーザーの職責が（昇進などによって）変わる場合は、アクセス制御リスト内に多数存在する、そのユーザーのエントリを更新するかわりに、そのユーザーに異なるロールを割り当てることで認可情報を簡単に更新できます。

たとえば、複数のユーザーが /home/user ディレクトリ内のファイル salaries に対する書込み権限を必要としなくなった場合は、その権限を HR ロールから削除します。HR ロールのメンバー全員のファイル・アクセス権と権限が自動的に更新されます。

ロールのアクティブ化

通常、ユーザーには複数のロールが付与されます。ただし、すべてのロールがデフォルトで有効になるわけではありません。アプリケーションでは、ユーザー・セッションで特定のタスクを完了するために、run-as セキュリティ ID と Subject.doAS() を使用して必要なロールを選択して有効化できます。ロールを選択して有効化することで、最小限の権限の原理が守られます。つまり、アプリケーションでは、タスクに不要なファイル・アクセス権や権限は有効化されません。これによって、アクシデントやエラーによる不具合が制限されます。

3

JAAS Provider の構成とデプロイ

この章では、Java 2 Platform Enterprise Edition（J2EE）環境の Oracle Application Server Containers for J2EE（OC4J）で Java Authentication and Authorization（JAAS）プロバイダを使用するための必須構成タスクについて説明します。また、J2EE および OC4J のデプロイメント・ディスクリプタの概要についても説明します。この章には、次の項目が含まれます。

- LDAP ベース・プロバイダの環境設定
- J2EE のデプロイメント・ディスクリプタ
- OC4J デプロイメント・ディスクリプタ
- JAAS Provider の構成ファイル
- 認証（auth-method）の指定
- <jazn-web-app> でのサーブレット認証（runas-mode および doasprivileged-mode）の構成
- サーブレット内のセキュリティ・ロールのマッピング（run-as）
- RealmLoginModule の構成
- Oracle Internet Directory で SSL を使用するための JAAS Provider の構成
- EJB RMI クライアント・アクセスの構成
- キャッシングの構成（LDAP ベース・プロバイダのみ）
- orion-application.xml での UserManager の指定

注意： この章では、OC4J の構成方法全体については説明しません。詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

LDAP プロバイダを使用するには、Oracle Application Server と Oracle Internet Directory（OID）をインストールする必要があります。詳細は、『Oracle Application Server 10*g* インストレーション・ガイド』を参照してください。

JAAS ベースのアプリケーションを使用する前に、JAAS Provider コンポーネントを構成する必要があります。この章では、OC4J および J2EE 環境で JAAS を構成する方法について説明します。

LDAP ベース・プロバイダの環境設定

JAAS の LDAP ベース・プロバイダは、次のディレクトリの OID クライアント・ライブラリ `ldapjclnt9.jar` に応じて異なります。

[ORACLE_HOME]`/jlib`

OID クライアント・ライブラリは、次のディレクトリのネイティブ・ライブラリ（たとえば、Solaris の `libldapjclnt9.so` など）に応じて異なります。

[ORACLE_HOME]`/lib`

特に LDAP ベース・プロバイダを使用しているときに、JAZN Admintool の起動方法はこれらの依存状態の影響を受けます。Admintool を起動する前に、次のことをする必要があります。

- クラスパスに次が含まれていることを確認します。

  [ORACLE_HOME]`/jlib/ldapjclnt9.jar`

- 動的ライブラリ（たとえば、Solaris の `LD_LIBRARY_PATH` など）のロードを制御する、オペレーティング・システム固有の環境変数に次が含まれていることを確認します。

  [ORACLE_HOME]`/lib`

Enterprise Manager を使用して OC4J を管理すると、これらの 2 つの変数は自動的に設定されます。

J2EE のデプロイメント・ディスクリプタ

J2EE には、セキュリティ上の含意を持つ次の XML デプロイメント・ディスクリプタが用意されています。

表 3-1 J2EE のデプロイメント・ディスクリプタ

ファイル名	セキュリティ・タグ
web.xml	Web アプリケーション、サーブレットおよび JSP の場合、モジュールレベルのセキュリティ・ロール、セキュリティ制約および認可制約
ejb-jar.xml	EJB の場合、モジュールレベルとメソッドレベルのセキュリティ・ロール、セキュリティ制約および認可制約
application.xml（アプリケーションの EAR ファイルに含まれるファイル）	アプリケーションの場合、複数モジュール用のアプリケーションレベル・ディスクリプタ

注意： これらのディスクリプタの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。この章では、これらのディスクリプタのセキュリティ関連の側面についてのみ説明します。

OC4J デプロイメント・ディスクリプタ

OC4J には、セキュリティ上の含意を持つ、次のコンテナ固有の XML デプロイメント・ディスクリプタ・ファイルが用意されています。

表 3-2 OC4J の構成ファイル

ファイル	セキュリティ関連のタグ
グローバル application.xml	マッピング、ユーザー・マネージャ、<jazn>、<security-role-mapping>、<jazn> に埋め込まれた <jazn-web-app>
orion-web.xml	<jazn-web-app>
orion-application.xml	<jazn> タグ、<jazn> に埋め込まれた <jazn-web-app>

JAAS Provider の構成ファイル

JAAS Provider では構成情報が各種ファイルに格納され、サンプル構成ファイルが製品に同梱されています。JAAS Provider 構成ファイルを編集するには、JAZN Admintool を使用する方法と、テキスト・エディタを使用して手動で編集する方法があります。

注意： これらの構成ファイルを手動で編集し、OC4J をスタンドアロンで実行しない場合は、`dcmctl updateconfig` を実行して変更内容をクラスタ全体に伝播させる必要があります。

JAAS Provider の構成は、表 3-2 に示す全ファイルと次の JAAS 固有のファイル内で行います。

- `jazn.xml`

  JAAS Provider 構成ファイル。このファイルでは、プロバイダが LDAP ベース（データ・ストアとして OID を使用）であるか XML ベース（データ・ストアとして `jazn-data.xml` を使用）であるかなど、JAAS Provider のデフォルト構成を指定します。

- `jazn-data.xml`

  XML ベース・プロバイダの場合は、このファイルにユーザー、ロールおよびポリシー情報が格納されます。このファイルの編集には JAZN Admintool を使用します。

  注意： LDAP ベース・プロバイダを使用する場合、ユーザーとグループの管理には DAS を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

- `java2.policy`

  コードベースにパーミッションを付与する標準の Java 2 ポリシー・ファイル。このファイルは `$ORACLE_HOME/j2ee/home/config` にあります。

ポリシー・プロバイダとしての JAAS の指定（オプション）

Oracle Application Server 10*g* に同梱の JVM を使用する場合は、Oracle JAAS Provider が JAAS ポリシー・プロバイダとして自動的に指定されます。他の JVM を使用する場合は、ポリシー・プロバイダとして `oracle.security.jazn.spi.PolicyProvider` を明示的に指定する必要があります。デフォルトにより、JVM では Sun 社の JAAS Provider が使用されます。

ポリシー・プロバイダとして JAAS Provider を指定する手順は、次のとおりです。

1. `$JAVA_HOME/jre/lib/security/java.security` ファイルの末尾に次の情報を追加します。

   ```
   auth.policy.provider=oracle.security.jazn.spi.PolicyProvider
   login.configuration.provider=oracle.security.jazn.spi.LoginConfigProvider
   ```

jazn.xml の検索

ファイル `jazn.xml` は、XML ベースおよび LDAP ベースの JAAS Provider の構成ファイルです。JAAS Provider は、実行前に有効な `jazn.xml` ファイルを検索する必要があります。

JAAS Provider の起動時には、次に指定されているディレクトリ内で `jazn.xml` が順番に検索されます。

1. `oracle.security.jazn.config`（システム・プロパティ）
2. `java.security.auth.policy`（システム・プロパティ）
3. *`$J2EE_HOME`*`/config`（*`$J2EE_HOME`* はシステム・プロパティ `oracle.j2ee.home` で指定）
4. *`$ORACLE_HOME`*`/j2ee/home/config`（*`$ORACLE_HOME`* はシステム・プロパティ `oracle.home` で指定）
5. `./config`

`jazn.xml` ファイルが見つかると、JAAS Provider は検索を停止します。このファイルが見つからない場合は、「JAZN が正しく構成されていません。」というエラー・メッセージが戻されます。

<jazn> タグ

<jazn> タグを使用して JAAS Provider を構成します。<jazn> タグは、次の任意の場所に使用できます。

- アプリケーションの orion-application.xml
- グローバル application.xml
- jazn.xml

サンプル orion-application.xml ファイルと、すべての属性とプロパティ名の指定は、3-13 ページの「orion-application.xml での auth-method の指定」を参照してください。

このタグでサポートされる属性は、XML ベース・プロバイダと LDAP ベース・プロバイダのどちらを使用するかに応じて異なります。次の各項では、この 2 つについて個別に説明します。

- <jazn> タグと XML ベース・プロバイダ
- <jazn> タグおよび LDAP ベース・プロバイダ
- <jazn> の <property> サブ要素

注意： Enterprise Manager を使用して <jazn> タグを編集することはできません。

<jazn> タグと XML ベース・プロバイダ

XML ベース・プロバイダを使用する場合、<jazn> タグでは表 3-3 に示す属性がサポートされます。

表 3-3（XML ベース・プロバイダ）orion-application.xml 内の <jazn> タグ

属性	値（太字はデフォルト）	例
provider	**XML** または LDAP	provider="XML"
location	（必須）プロバイダ・データが格納されているファイルへのパス。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 を含むディレクトリ内で jazn-data.xml を検索します。	location="./jazn-data.xml"
persistence	NONE jazn-data.xml には永続的な変更は書き込まれません。 ALL 変更後はその内容が永続します。 **VM_EXIT** JVM の終了時も変更が永続します。	persistence="ALL"
default-realm	認証または認可要求でレルムが明示的に指定されていない場合に使用されるレルム。リポジトリ内で構成しているレルムが 1 つのみの場合、この属性は不要です。	default-realm="*myrealm*"
config	config 属性が表示される場合、JAAS Provider はパス名に指定されているファイルからすべてのプロバイダ・プロパティを読み取ります。この属性を他の属性と組み合せることはできず、単独で使用する必要があります。	config="./jazn.xml"

関連項目： <jazn-web-app> 要素とその属性 auth-method、runas-mode および doasprivileged-mode の詳細は、3-12 ページの「認証（auth-method）の指定」を参照してください。

<jazn> タグおよび LDAP ベース・プロバイダ

orion-application.xml ファイルに次の例に類似したエントリを追加して、LDAP ベース・プロバイダを使用するようにアプリケーションを構成します。

```
<jazn provider="LDAP"/>
```

これにより、インストーラまたは Enterprise Manager のいずれかを使用して、OC4J インスタンスは OID に適切に関連付けられているものと想定されます。

OC4J インスタンスを Oracle Application Server Infrastructure（Oracle Internet Directory（OID）を含む）に関連付けると、アプリケーションで LDAP ベース・プロバイダをユーザーの集中管理に利用できるようになります。複数の異なる構成ファイルでの LDAP ベース・プロバイダの使用を指定できます。

LDAP ベース・プロバイダを application/xml 構成ファイルでグローバルに指定する場合、OID DAS で次のユーザーおよびグループを設定する必要があります。

- administrators グループ
- administrators グループのメンバーである admin ユーザー

JAZN Admintool を使用して、次のパーミッションを administrators グループに付与する必要があります。

- com.evermind.server.AdministrationPermission ("administration")
- com.evermind.server.rmi.RMIPermission("login")

属性とプロパティは追加設定できます。表 3-4 に、orion-application.xml 内の <jazn> タグの属性を示します。

表 3-4（LDAP ベース・プロバイダ）orion-application.xml 内の <jazn> タグ

属性	値（太字はデフォルト）	例
provider	**XML** または LDAP（この属性も指定できます）。	provider="LDAP"
location	LDAP サーバーの URL。	使用しないでください。注意を参照。
default-realm	認証または認可要求でレルムが明示的に指定されていない場合に使用されるレルム。構成しているレルムが 1 つのみの場合、この属性は不要です。	default-realm="us"

表 3-4（LDAP ベース・プロバイダ）orion-application.xml 内の <jazn> タグ（続き）

属性	値（太字はデフォルト）	例
config	config 属性が表示される場合、JAAS Provider はパス名に指定されているファイルからすべてのプロバイダ・プロパティを読み取ります。この属性を他の属性と組み合せることはできず、単独で使用する必要があります。	config="*configpath*"
persistence	**ALL** 変更後はその内容が永続します。 LDAP ベース・プロバイダではこの値を常に ALL に設定します。	

注意：

- orion-application.xml 内で <jazn> タグに provider 属性を指定しない場合は、jazn.xml 内で指定できます。
- provider および location 属性は、Enterprise Manager を使用して編集できます。他のすべての属性は手動で編集する必要があります。
- JAAS の LDAP ベース・プロバイダは <jazn> 要素の location 属性には依存しません。この属性を指定しない場合（デフォルトでは指定されません）、JAAS ランタイムは構成済のシステム設定からインフラストラクチャ情報を取得します。Enterprise Manager を使用して中間層をインフラストラクチャに関連付けると、システム設定が構成されます。この機能を利用するためには、jazn.xml および orion-application.xml などの、JAAS 構成ファイル内の <jazn> 要素の location 属性を設定しないことをお薦めします。

<jazn> タグ内に、認証情報を指定する <jazn-web-app> タグを使用できます。

関連項目： <jazn-web-app> 要素とその属性 auth-method、runas-mode および doasprivileged-mode の詳細は、3-12 ページの「認証（auth-method）の指定」を参照してください。

アプリケーションに対する匿名の読取り専用ログインを許可する場合は、これらの属性に値を割り当てないでください。

サンプル orion-application.xml ファイルと、すべての属性とプロパティ名の指定については、3-13 ページの「orion-application.xml での auth-method の指定」を参照してください。

<jazn> の <property> サブ要素

<jazn> タグには <property> 要素を指定できます。これらのプロパティのほとんどは、グローバル jazn.xml 内で VM ごとにのみ設定できます。例外は ldap.password および ldap.user のみです。<property> サブ要素の構文は次のとおりです。

```
<property name="propname" value="propvalue">
```

表 3-5 に、サポートされるプロパティを示します。

表 3-5 <jazn> タグの <property> 要素の値

プロパティ名	デフォルト	値
classpath		このプロパティは、サード・パーティ・クラスと JAR ファイルが他の場所で見つからない場合の検索場所を JAAS Provider に対して指定するときに使用します。 classpath="./loginmodules"
external.synchronization	**false**	true に設定すると、XML ベースの JAAS Provider は外部更新用のデータ・リポジトリ（通常は jazn-data.xml）を監視します。新規ユーザーをアプリケーション外で（たとえば、Admintool を使用して）追加すると、これらのユーザーは OC4J によって自動的に取得されます。このプロパティを設定しない場合、新規ユーザーを表示可能にするために OC4J を停止および再起動する必要があります。 本番環境では false に設定してください。
role.mapping.dynamic	**false**	role.mapping.dynamic="true" true に設定すると、JAAS Provider では、静的構成を使用するかわりに現行の Subject に基づいて認可チェックが実行されます。false に設定すると、認可チェックの基礎として静的構成が使用されます。
jndi.ctx_pool.init_size	**5**	JNDI/LDAP 接続プールの初期サイズ。
jndi.ctx_pool.inc_size	**10**	JNDI/LDAP 接続プールの増分サイズ : プール内の接続がすべて使用されるたびにプールに追加される接続数です。
ldap.cache.*	詳細は、3-19 ページの「Oracle Internet Directory で SSL を使用するための JAAS Provider の構成」および表 3-9 を参照してください。	
ldap.connect.max.retry	**5**	JAAS Provider が放棄する前に LDAP 接続の作成を試行する回数。
ldap.connect.sleep	**5000**	JAAS Provider が失敗した LDAP 接続を再試行する前に待機するミリ秒数。

表 3-5 <jazn> タグの <property> 要素の値（続き）

プロパティ名	デフォルト	値
ldap.password		LDAP ユーザー名の不明瞭化されたパスワード。次に例を示します。 {903}oZZYqmGc/iyCaDrD4qs2FHbXf3LAWtMN 不明瞭化の詳細は、10-2 ページの「jazn-data.xml および jazn.xml 内のパスワードの不明瞭化」を参照してください。
ldap.user		LDAP ユーザー名または DN。次に例を示します。 orcladmin または cn=orcladmin
ldap.walletloc		Oracle Wallet のパス名。
ldap.walletpwd		Oracle Wallet 用の不明瞭化されたパスワード。不明瞭化の詳細は、10-2 ページの「jazn-data.xml および jazn.xml 内のパスワードの不明瞭化」を参照してください。

注意：

- クリアテキストによる ldap.password または ldap.walletpwd エントリを指定するには、!welcome のようにパスワード値の前に感嘆符（!）を置きます。 エントリは自動的に不明瞭化されます。
- false に設定すると、XML ベースの JAAS Provider では、XML ベースのデータ・リポジトリに対する外部更新については、ファイル・システムを定期的にチェックしません。動的外部同期化を使用可能にするには、<jazn> タグで external.synchronization プロパティを true に設定してください。

認証（auth-method）の指定

認証方式（auth-method）は、<jazn-web-app> または <login-config> 要素を使用して複数の構成ファイルのいずれかで指定します。

web.xml での auth-method の指定

認証方式をグローバル・レベルで指定するには、web.xml の <login-config> 要素を使用します。次に例を示します。

```
<login-config>
   <auth-method>BASIC</auth-method>
</login-config>
```

web.xml では、表 3-6 に示す値を auth-method に指定できます。

表 3-6 web.xml での auth-method の値

設定	意味
BASIC（デフォルト）	アプリケーションでは Basic 認証である標準 J2EE 認証が使用されます。
FORM	アプリケーションではフォームベース認証が使用されます。
DIGEST	アプリケーションでは HTTP Digest 認証が使用されます。
CLIENT-CERT	アプリケーションでは、クライアントに対して SSL 用の独自証明書を提供するように要求します。

これらの値は、orion-web.xml または orion-application.xml の <jazn-web-app> 要素を使用してアプリケーション・レベルでオーバーライドできます。

orion-web.xml および orion-application.xml での auth-method の指定

トップレベルの <jazn-web-app> 要素で指定した auth-method により、web.xml 内の auth-method がオーバーライドされます。

orion-web.xml および orion-application.xml の auth-method に可能な値は 1 つのみです。それは、アプリケーションで Oracle Single Sign-On を使用することを意味する SSO です。

orion-web.xml のサンプル・エントリを次に示します。

```
<jazn-web-app
    auth-method="SSO"
/>
```

orion-application.xml のサンプル・エントリを次に示します。

```
<jazn provider="LDAP"
    <jazn-web-app auth-method="SSO"/>
</jazn>
```

orion-application.xml での auth-method の指定

<jazn> 要素の <jazn-web-app> 要素で指定した auth-method により、web.xml 内の auth-method がオーバーライドされます。可能な値は、アプリケーションで Oracle SSO を使用することを意味する SSO と、アプリケーションでは web.xml に指定された認証方式を使用することを意味する BASIC です。サンプル・エントリを次に示します。

```
<jazn provider="XML"
   location="jazn-data.xml"
   default-realm="JAZN.com"
   persistence="ALL"

<!-- default values for this application -->
   <jazn-web-app
      auth-method="SSO"
    />
</jazn>
```

<jazn-web-app> でのサーブレット認証（runas-mode および doasprivileged-mode）の構成

subject.doAs() または subject.doAsPrivileged() を使用してサーブレットを起動する場合は、orion-web.xml または orion-application.xml ファイル内で <jazn-web-app> 要素の runas-mode および doasprivileged-mode 属性を設定する必要があります。

- subject.doAs() は、特定のサブジェクトの権限を使用してサーブレットを起動します。サブジェクトは javax.security.auth.Subject クラスのインスタンスで定義され、ユーザーなど、単一エンティティに関する一連の事項が含まれます。これらの事項には、ID とパスワードや暗号鍵などのセキュリティ関連属性があります。JAAS Provider は、メソッド・コールで Subject インスタンスに渡します。

  doAs() メソッドを使用すると、AccessControlContext インスタンスは現行スレッド（サーバー）から取得されます。

- subject.doAsPrivileged() は、サーバーのアクセス制御制限を受けずに、特定のサブジェクトの権限を使用します。

  doAsPrivileged() メソッドを使用すると、JAAS Provider は現行サーバーの AccessControlContext インスタンスの制限を受けずに処理の更新を開始してサーブレットを実行するために、このメソッドを null の java.security.AccessControlContext インスタンスで起動します。

runas-mode および doasprivileged-mode は、サーブレットの起動に subject.doAsPrivileged() と subject.doAs() のどちらが使用されるかを制御します。デフォルトでは、runas-mode は false に設定されます。これは、subject.doAsPrivileged() も subject.doAs() も起動されないことを意味します。

注意： runas-mode は、servlet.runAs メソッドとは無関係です。

表 3-7 runas-mode と doasprivileged-mode の設定

runas-mode の設定値	doasprivileged-mode の設定値	サーブレットの起動
false（デフォルト）	true または false	特別な権限なし
true	true（デフォルト）	subject.doAsPrivileged()
true	false	subject.doAs()

したがって、サーブレットの起動に subject.doAsPrivileged() を使用させるには、次のような <jazn-web-app> 要素が必要です。

```
<jazn-web-app
    auth-method="SSO"
    runas-mode="true"
    doasprivileged-mode="true"
/>
```

サーブレット内のセキュリティ・ロールのマッピング（run-as）

OC4J のグループを使用して、J2EE のセキュリティ・ロールを JAAS のロールにマップできます。これにより、セキュリティ・ロールまたは特定の RealmPrincipal クラスの権限を使用して、アプリケーションを実行できます。次のタスクは、両方の権限に関係しています。

注意： run-as 要素は、runas-mode とは無関係です。

---

run-as 要素を指定すると、<role-name> は Web アプリケーションに対して定義されているセキュリティ・ロールにマップされます。

次の手順は、sr_manager がすでに web.xml 内で次のようにセキュリティ・ロールとして定義されていることを想定しています。

```
<security-role>
   <role-name>sr_manager</role-name>
</security-role>
```

J2EE のセキュリティ・ロールを JAAS のロールにマップする手順は、次のとおりです。

1. <servlet> タグ内の run-as 要素を、web.xml ファイル内で特定の J2EE セキュリティ・ロールまたは特定の RealmPrincipal クラスとして実行するように指定します。

   たとえば、セキュリティ・ロール sr_manager として実行するには、次のように指定します。

   ```
   <servlet>
     <servlet-name>DevGroup</servlet-name>
     <servlet-class>DevGroupServlet</servlet-class>
    <!--  run as security role "sr_manager" -->
       <run-as>
         <role-name>sr_manager</role-name>
       </run-as>
   </servlet>
   ```

2. jazn-data.xml ファイル内で JAAS <role> 要素を定義します。

   たとえば、developer をロール内で定義します。

```
<roles>
    <role>
     <name>developer</name>
     <members>
      <member>
   <type>user<type>
   <name>john<name>
       </member>
      </members>
    </role>
   </roles>
```

jazn-data.xml の XSD スキーマは、付録 B「JAAS Provider のスキーマ」を参照してください。

3. ステップ 1 とステップ 2 で作成した定義を統合します。orion-application.xml ファイル内で OC4J のグループを次のように使用します。

   - web.xml ファイルに定義されている role-name をセキュリティ・ロール（sr_manager）としてマップします。
   - jazn-data.xml に定義されている role を OC4J のグループ名（developer）としてマップします。

   たとえば、sr_manager セキュリティ・ロールは JAAS Provider 内のグループ developer にマップされます。

```
<security-role-mapping name="sr_manager">
  <group name="developer" />
</security-role-mapping>
```

developer グループは J2EE のセキュリティ・ロール sr_manager にマップされるため、ユーザー（この例では john）は sr_manager ロールで定義されたアプリケーション・リソースへのアクセス権を持ちます。

関連項目：

- http://java.sun.com/products/servlet/download.html で入手可能な Java サーブレット仕様を参照してください。
- 第 6 章「セキュリティと J2EE アプリケーション」

RealmLoginModule の構成

RealmLoginModule クラスは、jazn-data.xml ファイルを使用して構成されるデフォルトの LoginModule です。RealmLoginModule クラスでは、ユーザーが J2EE アプリケーションにアクセスする前にユーザー・ログイン資格証明が認証されます。認証は、OC4J コンテナベース認証（HTTP BASIC、FORM など）を使用して実行されます。アプリケーションで SSO 認証を使用する場合、RealmLoginModule クラスを有効化する必要はありません。

> **関連項目：** SSO 構成タスクの詳細は、『Oracle Application Server 10*g* インストレーション・ガイド』を参照してください。

RealmLoginModule を有効化するには、JAZN Admintool を使用する方法と jazn-data.xml を編集する方法があります。Admintool を使用する方法の詳細は、5-8 ページの「ログイン・モジュールの追加と削除」を参照してください。

テキスト・エディタを使用した RealmLoginModule の有効化

必要な場合は、テキスト・エディタを使用してログイン構成ファイル jazn-data.xml を変更します。

jazn-data.xml ファイルでの RealmLoginModule クラス設定のデフォルト構成は、次のとおりです。

```
<!DOCTYPE jazn-data (View Source for full doctype...)>
 <jazn-data>
    .
    .
    .
<!--  Login Module Data -->
 <jazn-loginconfig>
  <application>
     <name>JAZNUserManager</name>
    <login-modules>
      <login-module>
   <class>oracle.security.jazn.realm.RealmLoginModule</class>
   <control-flag>required</control-flag>
   <options>
     <option>
       <name>addRoles</name>
       <value>true</value>
     </option>
   </options>
      </login-module>
     </login-modules>
```

```
   </application>
  </jazn-loginconfig>
 </jazn-data>
```

<login-module> タグでは、次の <option> 値がサポートされます。

表 3-8 RealmLoginModule のオプション

名前	意味	デフォルト
debug	true に設定すると、デバッグ・メッセージが出力されます。	false
addRoles	true に設定すると、RealmLoginModule では、ユーザーに直接付与されているロールすべてが認証の成功後に Subject に追加されます。	true
addAllRoles	true に設定すると、RealmLoginModule では、ユーザーに直接または間接的に付与されているロールすべてが認証の成功後に Subject に追加されます。	true
storePrivateCredentials	true に設定すると、RealmLoginModule では、すべての秘密資格証明（パスワード資格証明など）が認証の成功後に Subject に追加されます。	false
supportCSIv2	true に設定すると、RealmLoginModule で CSIv2 がサポートされます。	false
supportNullPassword	true に設定すると、RealmLoginModule では入力されたパスワードが null または空かどうかがチェックされません。false に設定すると、入力されたパスワードが null または空の場合に認証が失敗します。	false

関連項目：

- JAAS Provider の Javadoc を参照してください。
- 5-8 ページ「ログイン・モジュールの追加と削除」

Oracle Internet Directory で SSL を使用するための JAAS Provider の構成

この項では、OID で SSL を使用するように JAAS Provider を構成する方法について説明します。SSL を使用するように OID を構成する方法の詳細は、『Oracle Internet Directory 管理者ガイド』および『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。

SSL を使用して OID と通信するには、次の 3 つの方法があります。

1. NULL 認証 : データは匿名 Diffie-Hellman 暗号スイートを使用して暗号化されますが、証明書は認証に使用されません。

   注意 : サポートされる Cipher Suite のリストは、表 11-1「OracleSSL でサポートされる Cipher Suite」を参照してください。

2. サーバー・サイド認証のみ（サーバー認証）: サーバーはデジタル証明を使用してクライアントに対して自己認証を行いますが、クライアントは自己認証を行いません。
3. クライアントおよびサーバー認証（双方向認証）: クライアントとサーバーの両方がデジタル証明を使用して自己認証を行います。

NULL 認証（方式 1）の場合は、<jazn> タグに <property> タグを追加してプロトコルを指定します（NULL 認証では証明書が使用されないため、Wallet の場所やパスワードは指定しないことに注意してください）。

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<jazn provider="LDAP" location="ldap://pichan-sun.us.oracle.com:5000"
default-realm="us">

   <property name="ldap.protocol" value="ssl"/>

</jazn>

<property name="ldap.protocol" value="ssl"/>
```

一方向または双方向の認証（方式 2 および 3）の場合は、<property> タグを使用してプロトコル、Wallet の場所および Wallet のパスワードを指定します。

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<jazn provider="LDAP" location="ldap://pichan-sun.us.oracle.com:5000"
default-realm="us">

   <property name="ldap.protocol" value="ssl"/>
   <property name="ldap.walletloc" value="/private/oracle/oid/ssl2/testwallet.txt"/>
   <property name="ldap.walletpwd" value="!welcome1"/>

</jazn>
```

EJB RMI クライアント・アクセスの構成

(LDAP ベース・プロバイダのみ)

RMI を介した EJB へのファット・クライアント・アクセスを有効化するには、JAZN Admintool を使用して適切なパーミッションを付与する必要があります。ユーザーまたはロールに RMIPermission (login) を付与する必要があります。次に例を示します。

```
java -jar jazn.jar -grantperm myrealm -role administrators \
  com.evermind.server.rmi.RMIPermission login
```

キャッシングの構成（LDAP ベース・プロバイダのみ）

LDAP ベースの JAAS Provider では、キャッシングがサポートされ、パフォーマンスとスケーラビリティが改善されます。次の 3 つのキャッシュがあります。

- ポリシー・キャッシュ。権限受領者とパーミッションが格納されます。
- レルム・キャッシュ。レルム、ユーザーとロールおよびロール・グラフが格納されます。
- セッション・キャッシュ。ユーザーおよびロール・グラフが HTTP セッション・オブジェクトに格納されます（このキャッシュを使用できるのは、Cookie が有効化されている Web ベース・クライアントの場合のみです)。

キャッシング・サービスにより、キャッシュ内のオブジェクトの格納と取得に使用されるグローバル `HashMap` がメンテナンスされます。デーモン・スレッドがバックグラウンドで定期的に実行され、`HashMap` 内の期限切れオブジェクトが失効化されてクリーン・アップされます。キャッシュ内のオブジェクトは TTL アルゴリズムに基づいて期限切れとなり、有効期限は表 3-9 に示すキャッシュ・プロパティで設定できます。

注意： これらのキャッシュは、LDAP ベースのプロバイダでのみ提供されます。XML ベース・プロバイダでは、デフォルトで XML 文書全体がキャッシュされます。

セッション・キャッシュの詳細

HttpSession オブジェクトは、サーバー・サイド・セッションの継続時間中は継続します。アプリケーションでは HttpSession.invalidate() を起動してセッションを明示的に終了でき、コンテナでは <session-timeout> 値に基づいてセッションを終了できます。

> **関連項目：** OC4J でのセッション・サポートの詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。

OC4J サーバーの場合、デフォルトのセッション・タイムアウトは 20 分です。このデフォルトを変更するには、web.xml を編集して <session-timeout> を変更します。

```
<session-config>
  <session-timeout>10 </session-timeout>
</session-config>
```

JAAS Provider は、必要な場合は HttpSession.invalidate() を起動してセッションを明示的に失効化します。たとえば、WebSSOUtil.logout() が起動されると、JAAS Provider によりセッションが失効化されます。

> **注意：** HttpSession インスタンスに格納されているオブジェクトの場合は、web.xml 内の <distributable /> フラグを使用してデプロイされるように、java.io.Serializable インタフェースを実装する必要があります。

キャッシングの無効化

キャッシングはデフォルトで有効化されています。管理タスクの実行時には、特に次のキャッシュを無効化する必要があります。

- ポリシーを管理する場合はポリシー・キャッシュを無効化します。ポリシー・キャッシュが有効化されている場合は、Policy.grant() または Policy.revoke() をコールすると UnsupportedOperationException が発生します。
- レルムを管理する場合はレルム・キャッシュを無効化します。レルム管理タスクには、レルムの追加、レルムの削除、ロールの付与およびロールの取消しが含まれます。
- HTTP セッションの Cookie を無効化する場合は、セッション・キャッシュを無効化します。

次は、キャッシュを無効化する方法を示す、jazn.xml からのサンプル引用句です。

```
<jazn provider="LDAP">
   <property name="ldap.user" value="cn=orcladmin"/>
   <property name="ldap.password"
value="{903}3o4PTHbgMzVlzbVfKITIO5Bgio6KK9kD"\/>
   <property name="ldap.cache.session.enable"
```

```
value="false" />
   <property name="ldap.cache.realm.enable"
value="false" />
   <property name="ldap.cache.policy.enable"
value="false" />
</jazn>
```

構成

次の表に、キャッシュのプロパティとデフォルト値を示します。これらのプロパティは、仮想マシン・レベルのみでグローバル jazn.xml ファイル内の <jazn> タグに設定する必要があります。

表 3-9 LDAP キャッシュ・プロパティ

プロパティ	説明	デフォルト
ldap.cache.policy.enable（**注意**を参照）	true に設定するとキャッシュが有効化され、false に設定すると無効化されます。	true
ldap.cache.realm.enable	true に設定するとキャッシュが有効化され、false に設定すると無効化されます。	true
ldap.cache.session.enable	true に設定するとキャッシュが有効化され、false に設定すると無効化されます。	true
ldap.cache.initial.capacity	HashMap の初期容量です。	20
ldap.cache.load.factor	HashMap のロード・ファクタです。	.7
ldap.cache.purge.initial.delay	デーモン・スレッドが期限切れオブジェクトのチェックを開始する前に待機するミリ秒数を表す整数の文字列です。	3600000
ldap.cache.purge.timeout	オブジェクトが失効化されて削除される前にキャッシュに残っているミリ秒数を表す整数の文字列表現です。これは、デーモン・スレッドが期限切れオブジェクトの検索を実行する間のスリープ時間でもあります。	3600000

注意： 廃止予定のプロパティ ldap.cache.enable は、ldap.cache.policy.enable で置き換えられています。

すべてのキャッシュが有効化され、キャッシュ・サイズ 100、タイムアウト 10000 ミリ秒が指定されている jazn.xml ファイルは、次のようになります。

```
< ?xml version="1.0" encoding="UTF-8" standalone='yes'?>
< !DOCTYPE jazn PUBLIC "JAZN Config"
     "http://xmlns.oracle.com/ias/dtds/jazn.dtd">
< jazn provider="LDAP" location="ldap://example.com:389" >
   < property name="ldap.cache.initial capacity" value="100" />
   < property name="ldap.cache.purget.timeout" value="10000" />
</jazn>
```

orion-application.xml での UserManager の指定

トップレベルのデフォルト・アプリケーションを含め、すべてのアプリケーションには UserManager が関連付けられています。UserManager の主な機能は、Web ページと EJB へのアクセスを試行するユーザーを認証することです。UserManager は、接続ユーザーをアプリケーションに対して認証するために使用されます。これらは、構成を定義する <orion-application> 要素内でサブ要素を使用して指定します。UserManager の指定には、次の 3 つのタグを使用できます。.

表 3-10 UserManager タグ

タグ	意味
<user-manager>	なんらかのユーザー定義クラスにより実装されるユーザー・マネージャ。
<jazn>	JAAS ユーザー・マネージャ。
<principal>	principals.xml ファイルに定義されているユーザー・マネージャ。3-24 ページの「<principals> 要素および principals.xml の使用」を参照してください。

1 つの <orion-application> 要素にこの 3 つの要素のうち複数を指定できます。どの要素に指定した UserManager が使用されるかは、表に示した要素の順序に従って決定されます。つまり、<user-manager> は <jazn> より優先され、<jazn> は <principals> より優先されます。たとえば、<jazn> 要素と <principals> 要素の両方が存在する場合、UserManager は <jazn> 要素に基づいて決定されます。優先順位が最も高いタグに複数の要素が存在する場合、UserManager は親として連鎖します。つまり、最初のタグに指定された UserManager が 2 番目のタグに指定された UserManager の親となります。最後に指定された UserManager は、アプリケーションの UserManager となります。最初の UserManager の親は、アプリケーションの親アプリケーション（存在する場合）に関連付けられている UserManager です。デフォルト・アプリケーションには親アプリケーションがなく、その UserManager の親は null です。

ユーザー・マネージャを指定しない場合は、次のルールに従って UserManager が決定されます。

- デフォルト・アプリケーションの場合は、application.xml を含むディレクトリ内の jazn-data.xml に基づいて JAAS UserManager が作成されます。そのディレクトリに jazn-data.xml が存在しない場合は、このファイルが作成されます。作成される jazn-data.xml のデフォルト・レルムは jazn.com です。
- 親アプリケーションの UserManager が JAAS UserManager の場合は、デプロイメント時に jazn-data.xml に基づいて JAAS UserManager が作成されます。必要な場合は、前項と同じ方法で jazn-data.xml ファイルが作成されます。アプリケーションに関連付けられている orion-application.xml に <jazn> 要素が書き込まれます。
- デプロイメント時に、親アプリケーションの UserManager が principals.xml に基づいている場合、アプリケーションの UserManager はプリンシパルの UserManager となります。principals.xml ファイルが存在しない場合は、空のファイルが作成されます。アプリケーションに関連付けられている orion-application.xml に <jazn> 要素が書き込まれます。
- 親アプリケーションの UserManager がユーザー作成の場合は、親の UserManager がアプリケーションの UserManager となります。

<principals> 要素および principals.xml の使用

<principals> 要素は OC4J に対して、プリンシパル・ファイル（通常は principals.xml）に記述されている UserManager を使用するように指示します。<principals> 要素の属性は <path> のみであり、この属性ではプリンシパル・ファイル（通常は principals.xml）のパスを指定します。

次に例を示します。

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

principals.xml ファイルには <principals> 要素も含まれており、この要素には 2 つのサブ要素 <groups> および <users> が含まれています。<groups> 要素には 1 つ以上の <group> 要素が含まれ、<users> 要素には 1 つ以上の <user> 要素が含まれます。

注意： XMLUserManager クラスは、下位互換性のみを維持するためにサポートされています。JAAS Provider タイプのいずれかを使用することをお薦めします。

表 3-11 principals.xml の要素

要素	使用できるサブ要素	属性	説明
<principal>	<groups>、<users>	NA	ファイルに要素を含めます。
<groups>	<group>		このユーザー・マネージャに認識されるグループのリスト。
<group>	<description>、<permission>	name	単一のユーザー・グループを識別します。name 属性でグループ名を指定します。
<description>			JAAS Provider では使用されませんが、様々な状況で表示されます。
<permission>		name	プリンシパルに付与される java.security.Permission。次の 2 つの特殊な値があります。 ■ administrator: com.evermind.security.AdministrationPermission() と等価です。 ■ rmi:login: com.evermind.server.rm.RMIPermission("login") と等価です。
<users>	<user>		UserManager に認識されるユーザーのリスト。
<user>	<description>、<group-membership>		このグループに属するシングル・ユーザー。
		username	ユーザーの識別に使用される文字列。
		password	ユーザーの認証に使用されるクリアテキストのパスワード。このパスワードを不明瞭化するメカニズムはありません。
		deactivated	true または false。true の場合、このユーザーは検出されず、認証を受けることができません。
<description>			様々な状況で表示される任意の内容です。
<group-membership>		group	このユーザーが属している <group> の name 属性。

principals.xml 内のグループは、JAAS Provider のロールに対応します。principals.xml ファイルでは、JAAS Provider でのレルムに相当する概念はサポートされません。グループに付与されたパーミッションは明示的にチェックでき、OC4J では前述の特別なパーミッションがチェックされます。ただし、グループのパーミッションは、SecurityManager により実行される通常の Permission チェックとは統合されていません。

principals.xml ファイルの例を次に示します。

```
<?xml version="1.0" standalone='yes'?>
<!DOCTYPE principals PUBLIC "//Evermind - Orion Principals//"
"http://xmlns.oracle.com/ias/dtds/principals.dtd">

<principals>
  <groups>
<group name="guests">
    <description>users</description>
  </group>
  <group name="administrators">
    <description>administrators</description>
    <permission name="administration" />
  </group>
  </groups>
  <users>
  <user username="SCOTT" password="TIGER">
  <group-membership group="guests" />
  </user>
  <user username="anonymous" password="">
    <description>The default guest/anonymous user</description>
    <group-membership group="guests" />
  </user>
  <user username="admin" password=""  deactivated="true">
    <description>The default administrator</description>
    <group-membership group="users" />
    <group-membership group="administrators" />
  </user>
  </users>
</principals>
```

4

JAAS Provider 管理タスク

この章では、Oracle Application Server Containers for J2EE（OC4J）の JAAS Provider を管理する方法について説明します。

この章には、次の項目が含まれます。

- JAAS Provider の管理の概要
- レルムとポリシーの管理
- JAAS Provider デバッグ・ロギング

JAAS Provider の管理の概要

J2SE および J2EE 環境における JAAS Provider の管理には、レルム、ユーザー、ロール、パーミッションおよびポリシーの作成と管理が含まれます。OC4J には、JAAS 構成の管理用に JAAS Provider Admintool および Oracle Enterprise Manager（OEM）という 2 つのツールが用意されています。また、JAAS をプログラムで管理することもできます。

- Oracle Enterprise Manager
- Oracle Internet Directory Delegated Administrative Service（OID DAS）
- JAZN Admintool、コマンドライン・ツール
- JAAS Provider API

注意： 構成タスクは、アプリケーションで使用する JAAS Provider が XML ベースと LDAP ベースのどちらであるかによって異なります。

表 4-1 に、XML および LDAP プロバイダ環境で使用できるツールを示します。

表 4-1 XML ベースと LDAP ベースのプロバイダ環境の管理ツール

使用するツール	XML ベース・プロバイダ・タイプの場合	LDAP ベース・プロバイダ・タイプの場合
Oracle Enterprise Manager	セキュリティ・ロールのマップ	セキュリティ・ロールのマップ
OID DAS	該当せず	ユーザーおよびグループの管理
JAZN Admintool	ユーザー、ロールおよびポリシーの管理	ポリシーの管理

レルムとポリシーの管理

JAAS Provider では、プロバイダ・タイプと呼ばれる次の 2 種類のリポジトリ・プロバイダがサポートされます。

- XML ファイル（通常は `jazn-data.xml`）で使用される XML ベース・プロバイダ・タイプ
- Oracle Internet Directory（OID）で使用される LDAP ベース・プロバイダ・タイプ

> **注意：** LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

OID と `jazn-data.xml` は、レルム（ユーザーとロール）およびポリシー（パーミッション）情報の格納に使用されるリポジトリです。この項では、この 2 つのプロバイダ・タイプに関連する次の項目について説明します。

- レルムとポリシーの管理ツール
- JAAS Provider のレルムのフレームワーク
- XML ベース環境でのレルム管理
- LDAP ベース環境でのレルム管理
- JAAS Provider のポリシー管理

レルムとポリシーの管理ツール

レルムおよびポリシー情報については、複数の管理ツールが用意されています。表 4-2 に、これらのツールと動作環境を示します。

表 4-2 レルムとポリシーの管理ツール

方式 / 環境	説明	参照
Oracle Enterprise Manager LDAP ベースのみ	ユーザー、ロールおよびグループを管理するための GUI（Graphical User Interface）ツール。	8-1 ページ「JAAS と Enterprise Manager」
JAZN Admintool LDAP ベース環境と XML ベース環境	管理者がユーザー、レルム、ロールおよびポリシーを作成して管理するためのコマンドライン・インタフェース・ツール。JAZN Admintool の機能は次のとおりです。 ■ JAAS Provider API を使用して機能を実行します。 ■ オペレーティング・システムのコマンドラインから実行できます。 JAZN Admintool の機能と制限事項は、JAAS Provider API と同じです。たとえば、プロバイダ・タイプが LDAP ベース Oracle Internet Directory の場合、JAZN Admintool を使用してユーザーを作成することはできません。ただし、プロバイダ・タイプが XML ベースの場合はユーザーを作成できます。	5-1 ページ「JAZN Admintool の使用」

関連項目：

使用するプロバイダ・タイプのインストール方法は、『Oracle Application Server 10*g* インストレーション・ガイド』を参照してください。

JAAS Provider のレルムのフレームワーク

J2EE 環境では、ユーザー・コミュニティの概念が定義されます。ユーザー・コミュニティ・インスタンスは、実際には認可システムにより内部的にメンテナンスされるレルムです。

レルムをサポートするために、API パッケージ `oracle.security.jazn.realm` が用意されています。この API パッケージは、JAAS ポリシー・プロバイダへの拡張機能です。

レルムは、どちらのプロバイダ・タイプ環境でも管理できます。

- XML ベース

  レルムと JAAS ポリシーに軽量の格納形式を提供します。

- LDAP ベース Oracle Internet Directory

  レルムと JAAS ポリシーをディレクトリに集中的に格納します。

XML ベース環境でのレルム管理

レルムは、ユーザーおよびロール管理機能を提供します。XML ベース・プロバイダのほうが、LDAP ベース・プロバイダよりも軽量で制限が少ない高速のレルム実装を提供します。

XML ベース・レルム

JAAS Provider を使用して XML ベース環境用に 1 つ以上のレルムを作成できます。

> **関連項目：** レルムの作成指示は、4-1 ページの「JAZN Admintool の使用」を参照してください。

XML ベースのレルムおよびポリシー情報の格納

XML ベース・プロバイダを使用すると、次の操作ができます。

- レルム、ユーザーおよびロールの作成
- ユーザーおよび他のロールに対するロールの付与
- 特定のユーザーおよびロール（プリンシパル）へのパーミッションの割当て

この情報は XML ファイル（通常は `jazn-data.xml`）に格納されます。次の例に、`jazn-data.xml` ファイルでレルム、ユーザーおよびロールの作成に使用される構造を示します。

```
<!--JAZN Realm Data -->

   <jazn-realm>
        <realm>
             <name>jazn.com</name>
             <users>
                  <user>
                       <name>admin</name>
                       <displayName>Realm Administrator</displayName>
                       <description>Administrator for this realm</description>
                  <credentials>
                    {903}ZcOsWfcw5YRI0Bsq4sNFuLioZgX3a6CF
                  </credentials>
                   </user>
                   <user>
                       <name>anonymous</name>
                       <description>The default guest/anonymous
                                user</description>
                   </user>
             </users>
             <roles>
                  <role>
                       <name>guests</name>
```

```
                    <members>
                        <member>
                            <type>user</type>
                            <name>admin</name>
                        </member>
                        <member>
                            <type>user</type>
                            <name>anonymous</name>
                        </member>
                    </members>
                </role>
                <role>
                    <name>administrators</name>
                    <displayName>Realm Admin Role</displayName>
                    <description>Administrative role for this
                                realm</description>
                    <members>
                        <member>
                            <type>user</type>
                            <name>admin</name>
                        </member>
                    </members>
                </role>
                <role>
                    <name>users</name>
                    <members>
                        <member>
                            <type>user</type>
                            <name>admin</name>
                        </member>
                    </members>
                </role>
            </roles>
        </realm>
    </jazn-realm>
```

関連項目： `jazn-data.xml` ファイルの詳細は、A-2 ページの「サンプル jazn-data.xml コード」を参照してください。

注意： <credentials> 要素を次のように設定すると、jazn-data.xml ファイル内でクリアテキストによる（判読可能な）パスワードを使用できます。

- <credentials clear="true">welcome</credentials>
- <credentials>!welcome</credentials>

この設定により、管理者は jazn-data.xml ファイルをテキスト・エディタで直接編集できます。ファイルが読み取られて永続性が発生すると、jazn-data.xml 内のパスワードが不明瞭化されて判読不可能になります。

LDAP ベース環境でのレルム管理

レルムは、ユーザーおよびロール管理機能を提供します。LDAP ベースのレルムでは、次の方法でデータを管理できます。

- 内部的に JAAS Provider でユーザー情報を作成および管理します。第 4 章「JAAS Provider 管理タスク」を参照してください。
- 外部的に Oracle Internet Directory でユーザーおよびロール情報を作成し管理して、JAAS Provider に統合します。

LDAP ベースのレルム・タイプ

JAAS Provider では、LDAP ベース環境で 3 タイプのレルムがサポートされます。ユーザーおよびロール管理機能はレルムごとに異なります。表 4-3 にこれらのレルムを示します。

表 4-3 レルム・タイプの実装

レルム・タイプ	説明	使用する場合	参照
外部レルム	■ 外部の読取り専用ユーザーおよびロールの管理をサポート ■ 既存のユーザー・コミュニティを JAAS Provider と統合	非ホスティング環境の場合	4-8 ページの図 4-1
ID 管理レルム	■ プロビジョニング・ツールを介して作成 ■ ホスティング環境で使用 ■ 外部の読取り専用ユーザーおよびロールの管理をサポート	複数の顧客や会社が共有サービスにサブスクライブするホスティング環境の場合	4-9 ページの図 4-2
アプリケーション・レルム	■ 外部の読取り専用ユーザーの管理をサポート ■ 内部のロール管理をサポート	JAAS Provider のロール管理機能を使用する場合	4-10 ページの図 4-3

OracleAS Single Sign-On とともに LDAP ベースの JAAS Provider を使用するときは、ID 管理レルムを使用する必要があります。OracleAS Single Sign-On を使用するときは、外部レルムおよびアプリケーション・レルムはサポートされていません。

各レルム・タイプの構成要素は次のとおりです。

- ロール管理用のロール・マネージャ
- ユーザー管理用のユーザー・マネージャ

ユーザー・マネージャとロール・マネージャの機能は、内部的に（JAAS のパーミッションを介して）または外部的に（OID の Delegated Administration Service（DAS）を介して）実行されます。

注意： JAAS Provider には、ユーザー作成用の内部ユーザー・マネージャは用意されていません。ユーザーの作成には、DAS または `ldapadd` などのコマンドライン・ツールを使用できます。

図 4-1 に、JAAS Provider にインスタンスとして登録されている外部レルムを含むサンプル LDAP DIT を示します。レルム・タイプは、Realms コンテナの下に作成されます。

図 4-1 簡略化された外部レルム用ディレクトリ情報ツリー

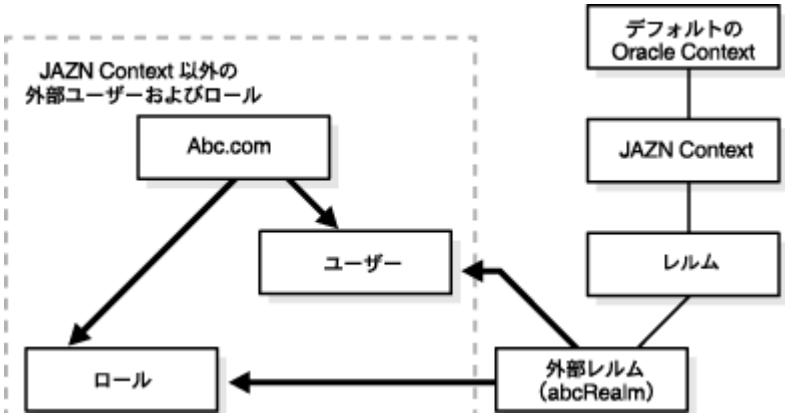


表 4-4 に、外部レルムのユーザーおよびロール管理の役割を示します。

表 4-4 外部レルムの役割

外部レルム名	ロール管理	ユーザー管理
abcRealm	外部の読取り専用ロールの取得	外部の読取り専用ユーザーの取得

図 4-2 に、JAAS Provider にインスタンスとして登録されている ID 管理レルムを含むサンプル LDAP DIT を示します。レルム・タイプは、Realms コンテナの下に作成されます。

図 4-2 簡略化された ID 管理レルム用ディレクトリ情報ツリー

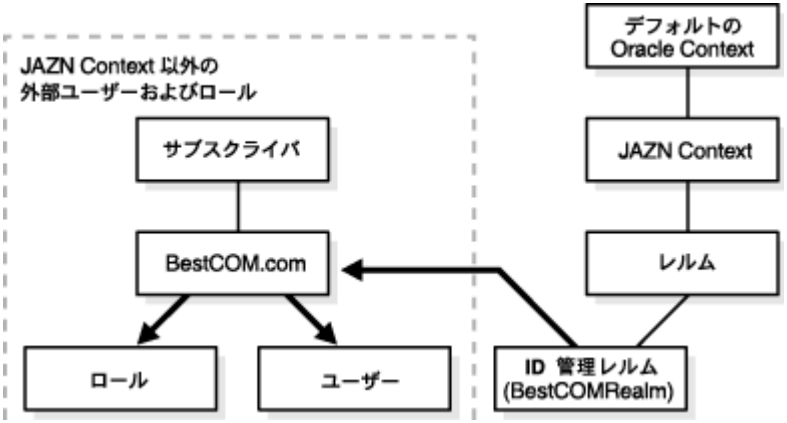


表 4-5 に、ID 管理レルムのユーザーおよびロール管理の役割を示します。

表 4-5 ID 管理レルムの役割

ID 管理レルム名	ロール管理	ユーザー管理
BestCOMRealm	サブスクライバの外部読取り専用ロールの取得	外部の読取り専用ユーザーの取得

図 4-3 に、JAAS Provider にインスタンスとして登録されているアプリケーション・レルムを含むサンプル LDAP ディレクトリ情報ツリー（DIT）を示します。レルム・タイプは、Realms コンテナの下に作成されます。

図 4-3 簡略化されたアプリケーション・レルム用ディレクトリ情報ツリー

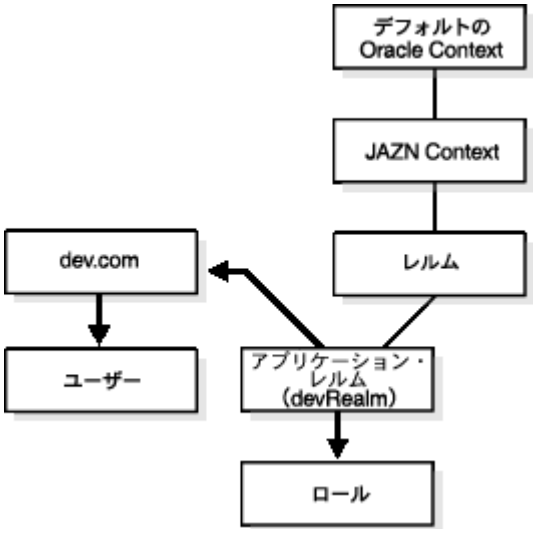


表 4-6 に、アプリケーション・レルムのユーザーおよびロール管理の役割を示します。

表 4-6 アプリケーション・レルムの役割

アプリケーション・レルム名	ロール管理	ユーザー管理
devRealm	変更可能なロールの内部的な作成および管理	外部の読取り専用ユーザーの取得

LDAP ベースのレルム・データの格納

レルム・フレームワークには、レルム・インスタンスを JAAS Provider に登録し、その情報を管理する手段が用意されています。

レルム・コンテナ・オブジェクトは、サイト全体の JAAS コンテキストで作成されます。（例は、4-8 ページの図 4-1 の Realms コンテナを参照してください。）登録済のレルム・インスタンスごとに、そのレルムの属性を格納する Realms コンテナの下に対応するレルム・エントリが作成されます。このディレクトリ階層は JAAS Provider に認識され、JAAS Provider では必要なディレクトリに新規のレルム・インスタンスを作成し、登録済のレルムすべてを実行時に検索できます。

たとえば、レルム oracle の識別名（DN）に "cn=oracle,cn=realms,cn=JAZNContext,cn=site root" を使用できます。

JAAS Provider が正常にインストールされると、デフォルトのレルム・インスタンスがインストールされます。事前定義済のレルム・プロパティは、デフォルト・レルムの起動用に構成されます。すべてのレルム・タイプは、システム定義の Java インタフェース

UserManager および RoleManager の具体的な実装を提供する必要があります。JAAS Provider は、実行時にすべての登録済レルムとその属性（名前、ユーザー・マネージャの実装クラス、ロール・マネージャの実装クラスおよびプロパティ）をプロバイダ・タイプ（Oracle Internet Directory）から検出し、初期化のためにプロパティを使用してレルムの実装クラスをインスタンス化します。

レルム階層 図 4-4 に示すように、JAAS Provider のエントリは製品コンテナ cn=JAZNContext に格納されます。cn=JAZNContext の下には、レルム・エントリが格納される cn=Realms コンテナと、グローバルな JAAS Provider ポリシーが格納される cn=Policy コンテナがあります。cn=Policy コンテナには、2 つのタイプのエントリ cn=Permissions および cn=Grantees が格納されます。

JAAS Provider には、専用の Groups および Users コンテナがあることに注意してください。Groups コンテナにはグループ JAZNAdminGroup が含まれています。Users コンテナには、これらのグループを移入するユーザーが含まれています。

図 4-4 グローバルな JAZNContext のサブツリー

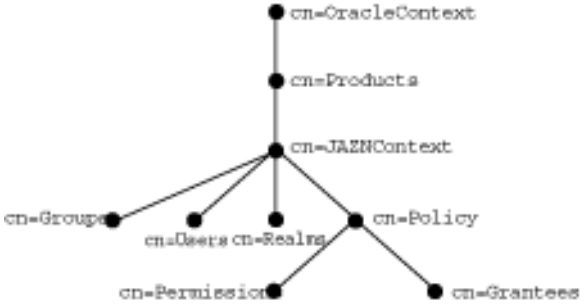
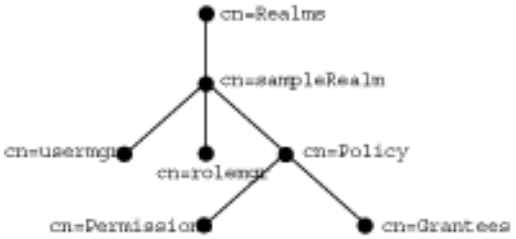


図 4-5 に、サンプル・レルム cn=sampleRealm の下に置かれるディレクトリ・エントリを示します。エントリ cn=usermgr にはユーザー管理の関連情報が格納され、エントリ cn=rolemgr にはロール（グループ）管理の関連情報が格納されます。cn=sampleRealm の下にあるポリシー関連エントリには、レルム固有のポリシーが格納されます。

図 4-5 レルム固有のサブツリー



ID 管理ベースの環境では、サブスクライバはレルムとして登録されます。JAAS Provider は、サブスクライバ DN を使用してサブスクライバ固有の Oracle Context を検索し、`cn=JAZNContext` サブツリーを作成します。この場合、JAAS Provider では、エントリ `cn=usermgr` および `cn=rolemgr` とポリシー関連エントリがサブスクライバの `JAZNContext` の下に格納されます。

図 4-6 では、`cn=oracle` がサブスクライバです。

図 4-6 サブスクライバの JAZNContext のサブツリー

ACL と JAZN ディレクトリ・エントリ JAAS Provider のディレクトリ・エントリは、製品サブツリーのルートにある ACL で保護されています。これらの ACL により、グループ `JAZNAdminGroup` と JAAS Provider のスーパー・ユーザー `JAZNAdminUser` に、JAAS Provider のディレクトリ・オブジェクトに対するすべての権限（読取り、書込み）が付与されます。`JAZNAdminGroup` のメンバーでない非スーパー・ユーザーは、JAAS Provider エントリへのアクセスを拒否されます。

ID 管理用 `JAZNContext` サブツリーはサイト全体の親のミラー・イメージであり、両者でエントリの保護に使用されるセキュリティ対策は同じです。

LDAP ベースのレルムのパーミッション

`RealmPermission` クラスは、レルムのパーミッションを表すように定義されます。`RealmPermission` は `java.security.Permission` からの拡張です。これは、通常の Java のパーミッションと同様に使用されます。`RealmPermission` には次の特性があります。

- ターゲット名とも呼ばれるレルム名
- アクション（レルムの作成、ロールの削除など、レルムに適用可能なパーミッション）のリスト

関連項目：

- JAAS Provider の Javadoc を参照してください。

JAAS Provider のポリシー管理

JAAS Provider の `javax.security.auth.Policy` 実装では、ポリシー（認可ルール）の格納に LDAP ベースの Oracle Internet Directory または XML ベース・プロバイダ・タイプが使用されます。JAAS Provider 管理者は、`JAZNPolicy` クラスの様々な権限付与および取消しメソッドを使用して、プリンシパルの認可ポリシーを作成します。

ポリシー・プロバイダはセキュアな方法で管理する必要があります。JAAS Provider のポリシーを管理するには、次の方法があります。

- Oracle Enterprise Manager（LDAP 環境のみ）
- JAAS Provider Admintool
- Oracle Internet Directory による管理
- AdminPermission クラス

関連項目： Oracle Enterprise Manager の詳細は、4-4 ページの表 4-2、JAZN Admintool の詳細は、5-1 ページの「JAZN Admintool の使用」を参照してください。

Oracle Internet Directory による管理

LDAP ベース・アプリケーション環境の場合は、レルムおよびポリシー・データを次の方法で Oracle Internet Directory エントリとして管理します。

- OID の `DAS` および `Oidadmin` 管理ツール
- Oracle Internet Directory 内のアクセス制御リストの定義

次の 2 つの管理グループがデータを管理できます。

- JAAS Provider のサイト全体の管理グループ。このグループには、サイト全体の JAZNContext と ID 管理固有の JAZNContext にアクセスして変更するためのパーミッションが付与されます。
- 各レルム・インスタンスまたは管理ユーザーごとのレルム固有の管理グループ。

ホスティングされたアプリケーション環境では、ポリシー・データの一部をサブスクライバ境界に沿ってパーティション化し、サブスクライバ・サブツリーに格納される場合があります。このようなポリシー・データは、レルム固有の管理グループでは管理できません。これはロール情報の場合も同じです。

JAAS Provider のポリシー・データ（レルム・データを含む）を使用すると、JAZNAdminGroup に属するユーザーにのみプロバイダ・データに対する読取りアクセス権が付与されます。

LDAP ベース環境では、プロバイダのポリシー・データがキャッシュされます。3-19 ページの「キャッシングの構成（LDAP ベース・プロバイダのみ）」を参照してください。

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

AdminPermission クラス

AdminPermission クラスは、LDAP ベース環境でも XML ベース環境でも使用できます。

AdminPermission クラスは、パーミッションを管理するための権利を表します。このクラスより、権限受領者（ユーザー frank など）は、自分が付与されている権利 / パーミッションをさらに他の権限受領者に付与したり、取り消すことができます。このパーミッション・クラスのインスタンスには、他のパーミッションのインスタンスが含まれます。これはパーミッションに関するパーミッションであるため、単純な名前とアクションのペアを含むパーミッション定義とは少し異なります。この違いは、パーミッション・インスタンスを文字列としてエンコードし、AdminPermission のインスタンス名として使用することで解消されます。表 4-7 に例を示します。

表 4-7 ADMIN パーミッションの例

ユーザーに付与されているパーミッション	ユーザーが実行できる操作
frank に、java.io.FilePermission("/tmp/*","read,write") に対する AdminPermission が付与されています。	frank は、埋め込まれているパーミッション（この例では FilePermission）が示すパーミッションをさらに他の権限受領者に付与したり取り消すことができます。

```
 grant Principal com.oracle.security.jazn.JAZNPrincipal "frank"
{
  permission com.oracle.security.jazn.policy.AdminPermission
     "class=java.io.FilePermission, name=\"/tmp/*\", actions=\"read, write\""
};
```

JAAS Provider では、AdminPermission の再帰的な埋込み（つまり、AdminPermission インスタンスを別の AdminPermission インスタンスに埋め込むこと）はサポートされていません。初期ポリシーでは、このユーザーには java.security.AllPermission に対する AdminPermission が付与されており、JAAS Provider ユーザーは他のすべてのユーザーについてもすべてのパーミッションを付与したり取り消すことができます。

RoleAdminPermission クラスはロールに対して定義されます。つまり、ロール hr が frank に付与されている場合、frank にはロール hr と RoleAdminPermission の両方が付与されます。後者により、frank はロール hr をさらに他のユーザーに付与したり取り消すことができます。

ポリシーのパーティション化

JAAS Provider では、レルム間でのポリシーのパーティション化がサポートされます（つまり、各レルムが独自のレルム固有のポリシーを持ちます）。このレルム固有のポリシーは、レルム固有の管理グループにより管理されます。

各サブスクライバはレルムで表され、サブスクライバ固有の情報サブツリーはサブスクライバ固有の JAZNContext の下に格納されます。ただし、このサブスクライバ固有のサブツリーは、LDAP サーバー（Oracle Internet Directory）の観点から主として JAAS Provider 管理グループにより管理されます。

JAAS Provider デバッグ・ロギング

JAAS Provider デバッグ・ロギングをオンにするには、Java Virtual Machine（JVM）の起動時に、システム・プロパティ jazn.debug.log.enable を true に設定します。OC4J インスタンスの JVM 設定を変更することでもこれを実行できます。Oracle Application Server では、通常、opmn.xml の <java-options> 要素を介して JVM 設定を管理します。スタンドアロン・モードでは、JVM コマンドライン・オプションを使用してこのプロパティを設定します。 たとえば、次のようなコマンドラインを使用して、OC4J スタンドアロンを起動できます。

```
java -Djazn.debug.log.enable=true -jar oc4j.jar
```

または、次のコマンドを使用して、デバッグ・モードで Admintool シェルを起動できます。

```
java -Djazn.debug.log.enable=true -jar jazn.jar -shell
```

Oracle Application Server では、デバッグ出力は OPMN によって取得され、OC4J インスタンスに関連するログ・ファイルに書き込まれます。

5

JAZN Admintool の使用

JAZN Admintool を使用すると、XML ベースと LDAP ベースの JAAS 構成およびデータをコマンド・プロンプトから管理できます。

> **注意：** JAZN Admintool では、XML ベースのロールおよびユーザーのみが管理されます。LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

JAZN Admintool は柔軟な Java コンソール・アプリケーションであり、その機能はコマンドラインから直接コールするか、または対話型シェルを介してコールできます。シェルでは、UNIX 派生コマンドを使用して特定の JAAS 機能が実行されます。JAZN Admintool は、*OC4J_HOME*/j2ee/home/jazn.jar にあります。

この章では、JAZN Admintool を使用して一般的な管理タスクを実行する方法について説明します。この章には次の項が含まれています。

- パスワードのチェック（XML ベース・プロバイダのみ）
- 構成操作
- パーミッションの付与と取消し
- ロールの付与と取消し
- ログイン・モジュールのリスト表示
- パーミッションのリスト表示
- パーミッション情報のリスト表示
- プリンシパル・クラスのリスト表示
- レルムのリスト表示
- ロールのリスト表示
- ユーザーのリスト表示
- principals.xml ファイルからのプリンシパルの移植（XML ベース・プロバイダのみ）
- パスワードの設定（XML ベース・プロバイダのみ）
- JAZN Admintool シェルの使用

はじめに

Admintool を使用して XML プロバイダ・データを管理するときは、デフォルトにより、OC4J ホーム・インスタンスの config ディレクトリ下でファイル jazn-data.xml が編集されます。jazn-data.xml のパス名は、jazn.xml の <jazn provider="xml" location="*pathname*"> 要素に指定されています。管理ユーザーのパスワードは、インストール時に Oracle Application Server 管理者の（ias_admin）パスワードと同じ値に設定されます。

Admintool を LDAP ベース・プロバイダとともに使用する場合、必ず次のことをしてください。

1. 3-2 ページの「LDAP ベース・プロバイダの環境設定」に示すように、適切な環境設定を設定します。
2. 3-21 ページの「キャッシングの無効化」に示すように、キャッシュを無効化します。

認証と JAZN Admintool（XML ベース・プロバイダのみ）

XML ベース・プロバイダを使用する場合は、管理上の変更を加える前に、JAZN Admintool に対して自己認証を行う必要があります。自己認証を行うには、次の 2 つの方法があります。

- -user および -password スイッチを指定する方法

```
java -jar jazn.jar -user myusername -password mypassword -listrealms
```

> **注意：** -user、-password または -clustersupport オプションを使用する場合は、コマンドラインで他のすべてのオプションの前に指定する必要があります。

- Admintool のプロンプトでユーザー名とパスワードを指定する方法

```
java -jar jazn.jar -listrealms
>RealmLoginModule username: martha
>RealmLoginModule password: mypass
```

どちらの場合も、jazn-data.xml 内で Admintool の LoginModule を指定できます。指定しない場合は、RealmLoginModule がデフォルトで使用されます。

> **注意：** Admintool を LDAP ベース・プロバイダとともに使用する場合、認証は不要です。LDAP の使用時には、-user および -password オプションを指定しても無視されます。

jazn-data.xml での Admintool の LoginModule の指定

jazn-data.xml ファイルでは、Admintool でユーザーの認証に使用する LoginModule を指定できます。次に例を示します。

```
<application>
  <name>oracle.security.jazn.tools.Admintool</name>
  <login-modules>
    <login-module>
     <class>oracle.security.jazn.realm.RealmLoginModule</class>
      <control-flag>required</control-flag>
      <options>
          <option>
            <name>debug</name>
            <value>false</value>
          </option>
          <option>
             <name>addAllRoles</name>
             <value>true</value>
          </option>
      </options>
    </login-module>
  </login-modules>
</application>
```

LoginModule を指定せずに実行すると、デフォルトのオプションで RealmLoginModule が使用されます。

JAZN Admintool のコマンドライン・オプション

JAZN Admintool には、次のコマンド・オプションが用意されています。詳細は、以降の各項で説明します。このツールでは、構文やパラメータが正しくないとエラー・メッセージが出力されます。次のように -help オプションを使用すると、すべてのオプションとその構文をリスト表示できます。

```
java -jar jazn.jar -help
```

構文

Admintool 全体の構文を次に示します。

```
java -jar jazn.jar [-user username -password mypassword
  -clustersupport ORACLE_HOME] [otheroptions]
```

> **注意：** -user、-password または -clustersupport オプションを使用する場合は、コマンドラインで他のすべてのオプションの前に指定する必要があります。

この項では、すべての Admintool コマンド・オプションをリスト形式で示します。

Admintool の認証（XML ベース・プロバイダのみ）

-user *username* -password *mypassword*
5-3 ページの「認証と JAZN Admintool（XML ベース・プロバイダのみ）」を参照してください。

クラスタ化操作

-clustersupport *oracle_home*

5-8 ページの「クラスタ化サポートの追加（XML ベース・プロバイダのみ）」を参照してください。

構成操作

-getconfig

5-14 ページの「構成操作」を参照してください。

対話型シェル

```
-shell
```

5-22 ページの「JAZN Admintool シェルの使用」を参照してください。

ログイン・モジュール

```
-addloginmodule application_name login_module_name
  control_flag [options]
-listloginmodules [application_name]
-remloginmodule application_name login_module_name
```

5-8 ページの「ログイン・モジュールの追加と削除」および 5-16 ページの「ログイン・モジュールのリスト表示」を参照してください。

移植操作

```
-convert filename realm
```

5-21 ページの「principals.xml ファイルからのプリンシパルの移植（XML ベース・プロバイダのみ）」を参照してください。

その他

```
-help  [<command name>]
```

特定のコマンドのヘルプを表示します。

パスワード管理（XML ベース・プロバイダのみ）

```
-checkpasswd realm user [-pw password]
-setpasswd realm user old_pwd new_pwd
```

5-14 ページの「パスワードのチェック（XML ベース・プロバイダのみ）」および 5-22 ページの「パスワードの設定（XML ベース・プロバイダのみ）」を参照してください。

ポリシー操作

```
-addperm permission permission_class action target [description]
-addprncpl principlename principle_class parameters [description]
-grantperm {<realm> {-user user|-role <role>} | <principal_class>
        <principal_params>} <permission_class> [<permission_params>] |
-listperms [<realm> {-user <user> |-role <role>} |
        <principal_class> <principal_params> | <permission_name>] |
-listperm permission
-listprncpls
-listprncpl principal_name
-remperm permission
-remprncpl principal_name
-revokeperm {<realm> {-user user|-role <role>} | <principal_class>
        <principal_params>} <permission_class> [<permission_params>] |
```

5-10 ページの「ポリシーのパーミッションの追加と削除（XML ベース・プロバイダのみ）」、5-11 ページの「プリンシパルの追加と削除（XML ベース・プロバイダのみ）」、5-15 ページの「パーミッションの付与と取消し」、5-17 ページの「パーミッションのリスト表示」、5-17 ページの「パーミッション情報のリスト表示」、5-18 ページの「プリンシパル・クラスのリスト表示」および「プリンシパル・クラス情報のリスト表示」を参照してください。

レルム操作

```
-addrealm realm admin {adminpwd adminrole | adminrole
  userbase rolebase realmtype}
-addrole realm role
-adduser realm username password
-grantrole role realm {user|-role to_role}
-listrealms
-listroles [realm [user|-role role]]
-listusers [realm [-role role|-perm permission]]
-remrealm realm
-remrole realm role
-remuser realm user
-revokerole role realm {user|-role from_role}
```

5-11 ページの「レルムの追加と削除」、5-12 ページの「ロールの追加と削除」、5-13 ページの「ユーザーの追加と削除（XML ベース・プロバイダのみ）」、5-16 ページの「ロールの付与と取消し」、5-19 ページの「レルムのリスト表示」、5-19 ページの「ロールのリスト表示」および 5-20 ページの「ユーザーのリスト表示」を参照してください。

クラスタ化サポートの追加（XML ベース・プロバイダのみ）

-clustersupport *oracle_home*

このオプションは Admintool に対して、すべての JAAS 構成変更をクラスタ全体に伝播させるように指示します。*oracle_home* 引数には、Oracle ホーム・ディレクトリの絶対パス名を指定します。-clustersupport オプションは -shell オプションとともに使用できます。

注意： -clustersupport オプションを使用する場合は、コマンドラインで他のすべてのオプションの前に指定する必要があります。

-clustersupport オプションは、XML ベース・プロバイダを使用している場合にのみ使用できます。

次に例を示します。

```
java -jar jazn.jar -clustersupport /oracle_home -shell
```

ログイン・モジュールの追加と削除

-addloginmodule *application_name login_module_name*
  control_flag [*optionname=value ...*]
-remloginmodule *application_name login_module_name*

-addloginmodule オプションを指定すると、指定したアプリケーション用に新規 LoginModule が構成されます。

control_flag には、javax.security.auth.login.Configuration に指定されているとおり、required、requisite、sufficient または optional のいずれかを指定する必要があります。表 5-1 を参照してください。

表 5-1 LoginModule 制御フラグ

フラグ	意味
Required	LoginModule が成功する必要があります。成功するかどうかに関係なく、認証は LoginModule リストの下位に進みます。
Requisite	LoginModule が成功する必要があります。成功すると、認証は引き続き LoginModule リストの下位に進みます。失敗すると、制御は即時にアプリケーションに戻ります（認証は LoginModule リストの下位に進みません）。

表 5-1 LoginModule 制御フラグ（続き）

フラグ	意味
Sufficient	LoginModule の成功は必須ではありません。成功すると、制御は即時にアプリケーションに戻り、認証は LoginModule リストの下位に進みません。失敗すると、認証は引き続き LoginModule リストの下位に進みます。
Optional	LoginModule の成功は必須ではありません。成功するかどうかに関係なく、認証は LoginModule リストの下位に進みます。

LoginModule が独自のオプションを受け入れる場合は、各オプションとその値を *optionname=value* のペアとして指定します。各 LoginModule には、独自の個別オプション・セットがあります。

たとえば、debug を true に設定して MyLoginModule を必須モジュールとしてアプリケーション myapp に追加するには、次のように入力します。

```
java -jar jazn.jar -addloginmodule myapp MyLoginModule required debug=true
```

MyLoginModule を myapp から削除するには、次のように入力します。

```
java -jar jazn.jar -remloginmodule myapp MyLoginModule
```

Admintool のシェルの場合

```
JAZN:> addloginmodule myapp MyLoginModule required debug=true
JAZN: remloginmodule myapp MyLoginModule
```

ポリシーのパーミッションの追加と削除（XML ベース・プロバイダのみ）

-**addperm** *permission permission_class action target [description]*
-**remperm** *permission*

-addperm オプションを使用して、JAAS Provider の PermissionClassManager にパーミッションを登録します。-remperm オプションを指定すると、指定したパーミッション・クラスから登録が削除されます。*permission* または *description* 引数に複数の語を指定するには、その語を引用符で囲みます（"*three word permission*"）。

既存のパーミッションを追加すると、Admintool ではそのパーミッションのアクション・リストとターゲット・リストが更新されます。

たとえば、レルムを削除するためのパーミッションを作成するには、次のように入力します。

```
java -jar jazn.jar -addperm perm1 oracle.security.jazn.realm.RealmPermission
droprealm "permission to drop a realm"
```

droprealm パーミッションを削除するには、次のように入力します。

```
java -jar jazn.jar -remperm perm1
```

Admintool のシェルの場合

```
JAZN:> addperm perm1 oracle.security.jazn.realm.RealmPermission droprealm -null
"permission to drop a realm"
JAZN: remperm perm1
```

プリンシパルの追加と削除（XML ベース・プロバイダのみ）

-**addprncpl** *principlename principle_class parameters [description]*
-**remprncpl** *principal_name*

-addprncpl オプションを使用して、JAAS Provider の PrincipalClassManager にプリンシパルを登録します。-remprncpl オプションを指定すると、指定したプリンシパル・クラスから登録が削除されます。*principal_name* および *description* 引数に複数の語を指定するには、その語を引用符で囲みます（"*three word description*"）。

既存のプリンシパルを追加すると、Admintool ではそのプリンシパルのパラメータ・リストが更新されます。

たとえば、プリンシパル staff を追加するには、次のように入力します。

```
java -jar jazn.jar -addprincpl staff oracle.security.jazn.spi.xml.XMLRealmUser
  "a staff user"
```

Admintool のシェルの場合

```
JAZN:> addprincpl staff oracle.security.jazn.spi.xml.XMLRealmUser -null "a staff
user"
```

レルムの追加と削除

-**addrealm** *realm admin {adminpwd adminrole | adminrole*
  userbase rolebase realmtype}
-remrealm *realm*

-addrealm オプションを使用して、指定したタイプのレルムを指定した名前で作成し、-remrealm オプションを使用してレルムを削除します。

たとえば、XML ベース・プロバイダを使用している管理者 martha がパスワード mypass、ロール hr を使用してレルム employees を追加するには、次のように入力します。

```
java -jar jazn.jar -addrealm employees martha mypass hr
```

LDAP ベース・プロバイダを使用している管理者 martha がロール hr を使用して外部レルムのユーザーベース ub とロールベース rb にレルム employees を追加するには、次のように入力します。

```
java -jar jazn.jar -addrealm employees martha hr ub rb external
```

注意： *realmtype* 引数が必須となるのは、LDAP ベース・プロバイダを使用している場合のみです。*realmtype* に可能な値は次のとおりです。

- external
- application

どちらの環境でも、管理者が employees を削除するには次のように入力します。

```
java -jar jazn.jar -remrealm employees
```

ロールの追加と削除

-addrole *realm role*
-remrole *realm role*

-addrole オプションを使用して指定のレルムにロールを作成し、-remrole オプションを使用してレルムからロールを削除します。

注意： LDAP ベース・プロバイダを使用している場合、-addrole と -remrole がサポートされるのはアプリケーション・レルムの場合のみで、外部管理レルムや ID 管理レルムについてはサポートされません。

たとえば、ロール roleFoo をレルム foo に追加するには、次のように入力します。

```
java -jar jazn.jar -addrole foo fooRole
```

レルムからロールを削除するには、次のように入力します。

```
java -jar jazn.jar -remrole foo fooRole
```

Admintool のシェルの場合 `JAZN:> remrole foo fooRole`

ユーザーの追加と削除（XML ベース・プロバイダのみ）

-adduser *realm username password*
-remuser *realm user*

-adduser オプションを使用して指定のレルムにユーザーを追加し、-remuser オプションを使用してユーザーをレルムから削除します。たとえば、パスワード mypass を指定してユーザー martha をレルム foo に追加するには、次のように入力します。

```
java -jar jazn.jar -adduser foo martha mypass
```

注意：

- パスワードを指定せずにユーザーを挿入するには、次のようにコマンドラインの最後に -null オプションを指定します。
  `jazn -jar jazn.jar -adduser foo martha -null`
- LDAP ベース・プロバイダを使用している場合、これらのコマンドは動作しません。

レルムから martha を削除するには、次のように入力します。

```
java -jar jazn.jar -remuser foo martha
```

Admintool のシェルの場合 `JAZN:> adduser foo martha mypass`

パスワードのチェック（XML ベース・プロバイダのみ）

-**checkpasswd** *realm user* [-pw *password*]

-checkpasswd オプションを使用して、指定のユーザーの認証にパスワードが必要かどうかを指定します。

-checkpasswd を単独で指定すると、Admintool では、ユーザーがパスワードを持っている場合は「このプリンシパルのためのパスワードが存在します。」、パスワードを持っていない場合は「このプリンシパルのためのパスワードは存在しません。」という応答が戻されます。

-checkpasswd を -pw オプションとともに指定すると、Admintool では、ユーザー名とパスワードのペアが正しい場合は「ユーザー / パスワードのペアの検証に成功しました。」、ユーザー名またはパスワード、あるいはその両方が正しくない場合は「ユーザー / パスワードのペアの検証に失敗しました。」という応答が戻されます。

たとえば、レルム foo 内のユーザー martha がパスワード Hello を使用するかどうかをチェックするには、次のように入力します。

```
java -jar jazn.jar -checkpasswd foo martha -pw Hello
```

Admintool のシェルの場合 `JAZN:> checkpasswd foo martha -pw Hello`

構成操作

-**getconfig**

-getconfig オプションを使用して、jazn.xml 内の現行の構成設定を表示します。

たとえば、レルム foo の構成設定をチェックするには、次のように入力します。

```
java -jar jazn.jar -getconfig
```

Admintool のシェルの場合 `JAZN:> getconfig foo`

パーミッションの付与と取消し

-**grantperm** *realm* {-*user user*|-role *role* } | *principal_class principal_parameters}* *permission_class* [*permission_parameters]*
-**revokeperm** *realm* {-user user|-role *role*} | *principal_class principal_parameters}* *permission_class* [*permission_parameters]*
-**listperms** *realm* {-user user|-role *role*} | *principal_class principal_parameters}* *permission_class* [*permission_parameters]*

principal_class は、プリンシパル・インタフェース（com.sun.security.auth.NTDomainPrincipal など）を実装するクラスの完全修飾名、*principal_paramters* は単一の文字列パラメータです。

-grantperm オプションを使用して、指定したパーミッションをユーザー（-user でコールする場合）、ロール（-role でコールする場合）またはプリンシパルに付与します。-revokeperm オプションを使用すると、指定したパーミッションがユーザー、ロールまたはプリンシパルから取り消されます。

permission_descriptor は、パーミッションの明示的なクラス名（oracle.security.jazn.realm.RealmPermission など）、そのアクション、およびアクション・パラメータとターゲット・パラメータ（RealmPermission、realmname action の場合）で構成されます。アクション・パラメータとターゲット・パラメータは複数指定できるため注意してください。

たとえば、ターゲット a.txt とアクション "read, write" を指定して、レルム foo のユーザー martha に FilePermission を付与するには、次のように入力します。

```
java -jar jazn.jar -grantperm foo martha java.io.FilePermission
 a.txt read, write
```

Admintool のシェルの場合 JAZN:> grantperm foo martha java.io.FilePermission a.txt read, write

ロールの付与と取消し

-grantrole *role realm* {user|-role *to_role*}
-revokerole *role realm* {user|-role *from_role*}

-grantrole オプションを使用して、指定のロールをユーザー（ユーザー名でコールする場合）またはロール（-role でコールする場合）に付与します。-revokerole オプションを使用すると、指定のロールがユーザーまたはロールから取り消されます。

> **注意：** LDAP ベース・プロバイダを使用している場合、-grantrole と -revokerole がサポートされるのはアプリケーション・レルムの場合のみで、外部管理レルムや ID 管理レルムについてはサポートされません。

たとえば、ロール editor をレルム foo のユーザー martha に付与するには、次のように入力します。

```
java -jar jazn.jar -grantrole editor foo martha
```

Admintool のシェルの場合 `JAZN:> grantrole editor foo martha`

ログイン・モジュールのリスト表示

-**listloginmodules** [*application_name*]

-listloginmodules オプションを使用して、指定の *application_name* 内の LoginModule をすべて表示します。*application_name* を指定しないと、すべてのアプリケーション内の LoginModule が表示されます。

たとえば、アプリケーション myapp の LoginModule をすべて表示するには、次のように入力します。

```
java -jar jazn.jar -listloginmodules myapp
```

Admintool のシェルの場合 `JAZN:> listloginmodules myapp`

パーミッションのリスト表示

-listperms *realm* {-user *user*|-role *role*} | *principal_class principal_parameters*} *permission_class* [*permission_parameters*]

-listperms オプションを使用して、リスト基準と一致するパーミッションをすべて表示します。このオプションでは、次のパーミッションのリストが表示されます。

- JAAS Provider の PermissionClassManager に登録されているすべてのパーミッション
- -role オプションを使用した場合は、ロールに付与されているパーミッション
- prinicpal に付与されているパーミッション

たとえば、レルム foo のユーザー martha のパーミッションをすべて表示するには、次のように入力します。

```
java -jar jazn.jar -listperms foo martha
```

Admintool のシェルの場合 `JAZN:> listperms foo martha`

パーミッション情報のリスト表示

-**listperm** *permission*

-listperm オプションを使用して、パーミッションの表示名、クラス、説明、アクションおよびターゲットなど、指定したパーミッションの詳細情報を表示します。

たとえば、パーミッション perm1 に関する情報をすべてリスト表示するには、次のように入力します。

```
java -jar jazn.jar -listperm perm1
```

標準的な出力は次のようになります。

```
Name:
perm1

Class:
oracle.security.jazn.realm.RealmPermission

Description:
permission to drop realm

Targets:
```

```
Actions:
droprealm  <no description available>
```

Admintool のシェルの場合 `JAZN:> listperm perm1`

プリンシパル・クラスのリスト表示

`-listprncpls`

`-listprncpls` オプションを使用して、`PrincipalClassManager` に登録されているプリンシパル・クラスをすべてリスト表示します。

次に例を示します。

```
java -jar jazn.jar -listprncpls
```

Admintool のシェルの場合 `JAZN:> listprncpls`

プリンシパル・クラス情報のリスト表示

`-listprncpl` *principal_name*

`-listprncpl` オプションを使用して、表示名、クラス、説明およびアクションなど、指定したプリンシパルの詳細情報を表示します。

たとえば、プリンシパル `martha` に関する情報をすべてリスト表示するには、次のように入力します。

```
java -jar jazn.jar -listprncpl martha
```

この例では、出力は次のようになります。

```
Name:
martha
Class:
oracle.security.jazn.spi.xml.XMLRealmUser
Description:
a staff user
Parameters:
```

Admintool のシェルの場合 `JAZN:> listprncpl martha`

レルムのリスト表示

-listrealms

-listrealms オプションを使用して、現行の JAAS 環境のレルムをすべて表示します。

たとえば、レルムすべてのリストを表示するには、次のように入力します。

```
java -jar jazn.jar -listrealms
```

Admintool のシェルの場合 `JAZN:> listrealms`

ロールのリスト表示

-**listroles** [*realm* [*user*|-role *role*]]

-listroles オプションを使用して、リスト基準と一致するロールのリストを表示します。このオプションでは、次のユーザーがリスト表示されます。

- パラメータを指定せずにコールした場合は、すべてのレルムのすべてのロール。
- レルム名とユーザー名を指定してコールした場合は、そのユーザーに付与されているすべてのロール。
- レルム名とオプション -role を指定してコールした場合は、指定した *role* に付与されているロール。

たとえば、レルム foo のすべてのロールのリストを表示するには、次のように入力します。

```
java -jar jazn.jar -listroles foo
```

Admintool のシェルの場合 `JAZN:> listroles foo`

ユーザーのリスト表示

-**listusers** [*realm* [-role *role*|-perm *permission*]]

-listusers オプションを使用して、リスト基準と一致するユーザーのリストを表示します。このオプションでは、次のユーザーがリスト表示されます。

- パラメータを指定せずにコールした場合は、すべてのレルムのすべてのユーザー。
- レルム名を指定してコールした場合は、そのレルムのすべてのユーザー。
- レルム名とオプション -role または -perm を指定してコールした場合は、特定のロールまたはパーミッションが付与されているユーザー。

たとえば、レルム foo のすべてのユーザーのリストを表示するには、次のように入力します。

```
java -jar jazn.jar -listusers foo
```

たとえば、パーミッション bar を使用しているレルム foo のすべてのユーザーのリストを表示するには、次のように入力します。

```
java -jar jazn.jar -listusers foo -perm bar
```

Admintool では、次のようにユーザーが 1 行に 1 人ずつリスト表示されます。

```
scott
admin
anonymous
```

Admintool のシェルの場合 `JAZN:> listusers foo`

principals.xml ファイルからのプリンシパルの移植（XML ベース・プロバイダのみ）

-**convert** *filename realm*

-convert オプションを使用して、principals.xml ファイルを現行の JAAS Provider の指定したレルムに移植します。*filename* 引数には、入力ファイルのパス名（通常は *ORACLE_HOME*/j2ee/home/config/principals.xml）を指定します。

> **注意：** 以前のリリースでは、このオプションは -migrate と呼ばれていました。-convert の構文と動作は -migrate と同じです。

移植により、principals.xml のユーザーが JAAS の users に、principals.xml のグループが JAAS のロールに変換されます。それまで principals.xml のグループに付与されていたパーミッションは、すべて JAAS のロールにマップされます。移植時にアクティブ化されていなかったユーザーは移植されません。このため、移植を介してユーザーに意図せずにアクセス権が付与されることはありません。

入力ファイルにエラーがあると、エラー（Javax.naming.AuthenticationException:Invalid username/password または javax.naming.NamingException:Lookup Error）が戻されます。

principals.xml を変換する前に、レルムの管理を認可されている管理者ユーザーがいることを確認する必要があります。次に手順を示します。

1. principals.xml 内で、デフォルトではアクティブ化されていない管理ユーザーをアクティブ化します。管理者用のパスワードを必ず作成してください。

   レルムの作成に、principals.xml 内の管理者名とは異なる管理者名を使用したことを確認します。管理者名の違いを確認するのは、convert コマンドでは重複するユーザーは移植されませんが、重複するロールは古い方を上書きすることで移植されるためです。

2. ダミー・ユーザーとダミー・ロールを使用してレルム principals.com を作成します。たとえば、Admintool シェルに次のように入力します。

   ```
   JAZN> addrealm principals.com u1 welcome r1
   ```

3. 次のように入力して、principals.xml を principals.com レルムに移植します。

   ```
   java -jar jazn.jar -convert config/principals.xml principals.com
   ```

4. jazn.xml を編集し、<default-realm> エントリを principals.com に変更します。

5. OC4J を停止して再起動します。

パスワードの設定（XML ベース・プロバイダのみ）

-setpasswd *realm user old_pwd new_pwd*

-setpasswd オプションを使用すると、管理者は古いパスワードを指定したユーザーのパスワードを再設定できます。

たとえば、レルム foo のユーザー martha のパスワードを mypass から a2d3vn に変更するには、次のように入力します。

```
java -jar jazn.jar -setpasswd foo martha mypass a2d3vn
```

Admintool のシェルの場合 `JAZN:> setpasswd foo martha mypass a2d3vn`

JAZN Admintool シェルの使用

-shell

-shell オプションを使用して、JAZN Admintool シェルを起動します。JAZN Admintool シェルを使用すると、UNIX 派生インタフェースを介して JAAS のプリンシパルとポリシーを対話形式で管理できます。

```
java -jar jazn.jar -user martha -password mypass -shell
JAZN:>
```

このシェルは JAZN:> プロンプトで応答します。インタフェース・シェルを終了するには、exit と入力します。

注意： 複数の語で構成される引数は、引用符で囲む必要があります。たとえば、`java -jar jazn.jar -user 'Oracle DBA' ...` などです。

XML ベース・プロバイダを使用している場合は、Admintool にユーザー名とパスワードを入力する必要があります。詳細は、5-3 ページの「認証と JAZN Admintool（XML ベース・プロバイダのみ）」を参照してください。LDAP ベース・プロバイダを使用している場合、-user および -password 引数を指定する必要はありません。

JAZN Admintool シェルのナビゲート

Admintool シェルでは、JAZN 構造内でのナビゲーション用に UNIX に似たコマンドがサポートされます。Admintool ディレクトリ構造の詳細は、5-25 ページの「Admintool シェルのディレクトリ構造」を参照してください。すべての Admintool コマンドで相対パスと絶対パスがサポートされます。

Admintool のナビゲーション・コマンドは次のとおりです。

add: プロバイダ・データの作成

add *directory_name* [*other_parameter*]
mkdir *directory_name* [*other_parameter*]
mk *directory_name* [*other_parameter*]

add、mkdir および mk コマンドはシノニムで、いずれも現行のディレクトリにサブディレクトリまたはノードを作成します。たとえば、現行のディレクトリがルートの場合、mk では realm が作成されます。現行のディレクトリが /realm/users の場合、mk では user が作成されます。add の効果は、現行のディレクトリに応じて異なります。一部のコマンドには、名前の他にもパラメータが必要です。

cd: プロバイダ・データのナビゲート

cd *path*

cd コマンドを使用すると、ディレクトリ・ツリーをナビゲートできます。相対パス名と絶対パス名がサポートされます。ディレクトリを出るには、次のように入力します。

```
cd ..
```

cd / と入力すると、ルート・ノードに戻ります。指定したディレクトリが存在しない場合は、エラー・メッセージが表示されます。

clear: 画面の消去

clear

clear コマンドを実行すると、80 の空白行が表示されて端末画面がクリアされます。

exit: JAZN シェルの終了

exit

exit コマンドを実行すると、JAZN シェルが終了します。

help: JAZN Admintool のシェル・コマンドのリスト表示

help

help コマンドを実行すると、有効なすべてのコマンドのリストが表示されます。

ls: データのリスト表示

ls [*path*]

ls コマンドを実行すると、現行のディレクトリまたはノードの内容がリスト表示されます。たとえば、現行のディレクトリがルートの場合、ls ではすべてのレルムのリストが表示されます。現行のディレクトリが /realm/users の場合、ls ではレルムのすべてのユーザーのリストが表示されます。表示されるリストは、現行のディレクトリに応じて異なります。ls コマンドには、* ワイルドカードを使用できます。

man: JAZN Admintool の Man ページの表示

man *command_option*
man *shell_command*

man コマンドを実行すると、指定したシェル・コマンドまたは JAZN Admintool コマンド・オプションの詳細な使用方法が表示されます。man ページに表示される情報とこのマニュアルの情報が矛盾する場合、このマニュアルにコマンドに関する正しい使用方法が記載されています。

pwd: 作業ディレクトリの表示

pwd

pwd コマンドを実行すると、ディレクトリ・ツリー内のユーザーの現在位置が表示されます。未定義の値は、このリストに空白として表示されます。

rm: プロバイダ・データの削除

rm *directory_name*

rm コマンドを実行すると、現行のディレクトリからディレクトリまたはノードが削除されます。たとえば、現行のディレクトリがルートの場合、rm では指定したレルムが削除されます。現行のディレクトリが /realm/users の場合は、指定したユーザーが削除されます。rm の効果は、現行のディレクトリに応じて異なります。指定したディレクトリが存在しない場合は、エラー・メッセージが表示されます。

rm コマンドには、* ワイルドカードを使用できます。

set: 値の更新

set *name=value*

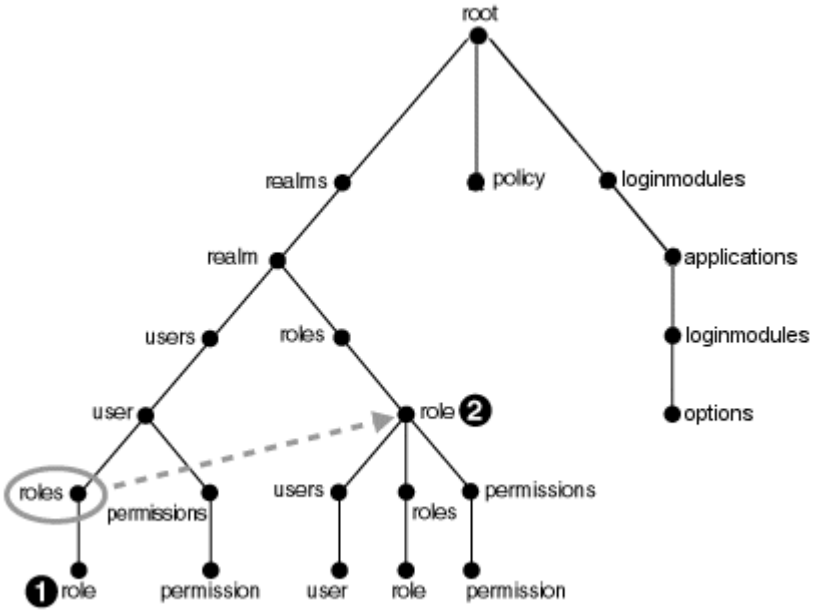
set コマンドを実行すると、指定した名前の値が更新されます。たとえば、このコマンドでは、作業ディレクトリに応じて、ログイン・モジュール・クラス、ログイン・モジュール制御フラグまたはログイン・モジュール・クラス・オプションが更新されます。

Admintool シェルのディレクトリ構造

JAZN Admintool には、JAZN シェル・インタフェースと呼ばれるシェルが組み込まれています。JAZN シェルは、JAAS Provider API への対話型インタフェースです。

シェルのディレクトリ構造はノードで構成され、各ノードには親ノードのプロパティを表すサブノードが含まれています。図 5-1 にノード構造を示します。

図 5-1 JAZN シェルのディレクトリ構造

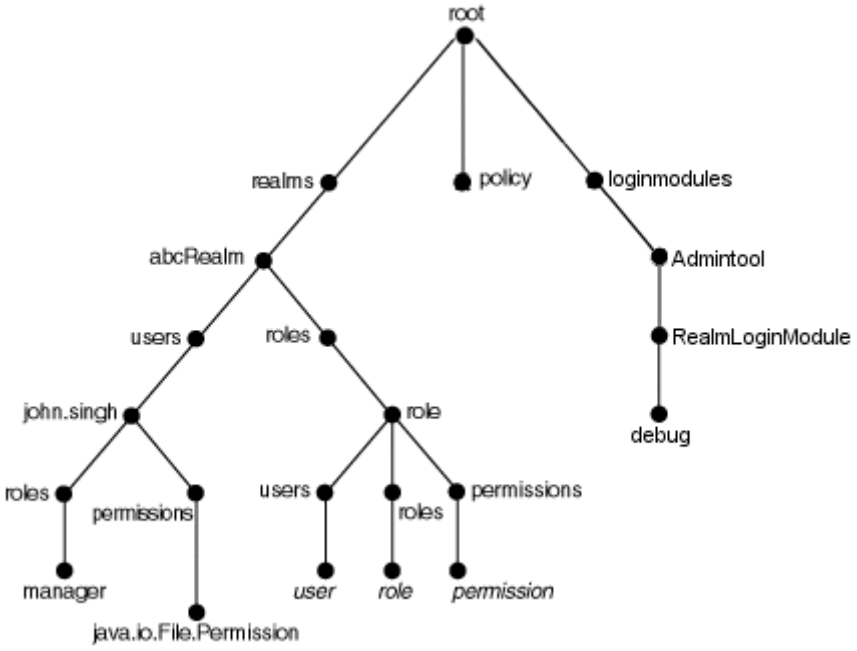


この構造では、user および role ノードがリンクされています。これは、user の下の roles リンクが、realm の下の roles リンクと同じであることを意味します。Unix 用語では、このダイアグラムで番号 1 の role は、このダイアグラムで番号 2 の role へのシンボリック・リンクです。

注意： このリリースでは、ポリシー・ディレクトリは常に空です。

図 5-2 に、A-2 ページの「サンプル jazn-data.xml コード」に示した `jazn-data.xml` ファイルにより作成される `xmlRealm` のノードを示します。

図 5-2 シェルのディレクトリ構造



6

セキュリティと J2EE アプリケーション

この章では、Oracle Application Server Containers for J2EE（Oracle Application Server Containers for J2EE）で J2EE アプリケーションに影響するセキュリティ上の問題について説明します。

この章には、次の項目が含まれます。

概要

JAAS Provider が Java 2 プラットフォーム向けに開発されたアプリケーションと統合されている場合、開発者は次の Oracle コンポーネントを使用できます。

- JAAS Provider。レルム情報（ユーザーとロール）およびポリシー情報（パーミッション）の格納、取得および管理をサポートします。JAAS Provider では、2 つのリポジトリ候補、つまりプロバイダ・タイプがサポートされます。
  - XML ベース・プロバイダ・タイプ。
  - LDAP ベースの Oracle Internet Directory（OracleAS Infrastructure がインストールされている場合にのみ使用可能）。
- JAAS Provider の `RealmLoginModule` などのログイン・モジュール。

関連項目：

- プロバイダ・タイプの詳細は、2-2 ページの「プロバイダのタイプ」を参照してください。
- Oracle Internet Directory のインストール方法は、『Oracle Application Server 10*g* セキュリティ・ガイド』を参照してください。

開発およびデプロイメント時のセキュリティ上の考慮事項

JAAS Provider は、J2EE の宣言によるセキュリティ・モデルで動作するように設計されています。この宣言によるモデルでは、アプリケーションで JAAS セキュリティを使用するためのプログラミングは不要であるか、必要であるとしてもごくわずかです。かわりに、セキュリティ上のほとんどの決定はデプロイメント処理中に行われ、再びコーディングしなくても容易に変更できます。宣言によるモデルでは不十分な場合、JAAS Provider ではすべての J2SE 環境で JAAS が使用されるのと同じ方法でプログラムによるセキュリティもサポートされます。

開発

アプリケーションが宣言によるセキュリティ・モデルに依存する場合（J2EE のセキュリティ・ロールが `web.xml` などのデプロイメント・ディスクリプタに定義されている場合）、開発者はアプリケーション固有のロールを使用するかどうかを決定する必要があります。固有のロールを使用する場合、開発者はデプロイメント・フェーズで J2EE の論理ロールにマップできるように、これらのロールを定義します。

アプリケーションで JAAS をプログラムで使用する場合、開発者は JAAS `LoginContext` を作成し、`login()` メソッドを明示的にコールして JAAS `LoginModule` を起動する必要があります。

デプロイメント

宣言によるセキュリティ・モデルを使用する場合、デプロイヤ担当はセキュリティに関連して次の事項を決定する必要があります。

- アプリケーションで想定される J2EE の論理ロールを決定し、これらのロールをデプロイメント・ディスクリプタに定義します。たとえば、HR アプリケーションを実行する J2EE の論理ロールが `hr_manager` であると想定される場合、デプロイヤ担当はそのロールを定義する必要があります。
- これらのロールに適用される認可制約を決定し、それをデプロイメント・ディスクリプタに定義します。Web モジュールの場合、通常、これらの制約は J2EE 仕様に定義されている URL パターンに適用されます。EJB モジュールの場合、制約は通常、EJB メソッド・レベルに適用されます。
- JAAS Provider のリポジトリとして XML フラット・ファイルと OID（LDAP）のどちらを使用するかを決定します。これにより、アプリケーションで使用するプロバイダ（XML ベースまたは LDAP ベース）とユーザー・マネージャも決まります。
- セキュリティ・ロール（存在する場合はアプリケーション固有のロールを含む）を、OC4J ユーザー・マネージャ（`JAZNUserManager` など）により定義されたユーザーとグループにマップします。たとえば、J2EE の論理ロール `hr_manager` を、OC4J ユーザー・マネージャにより定義された指定のユーザー・セットにマップできます。

これらの意志決定とその実装の詳細は、第 3 章「JAAS Provider の構成とデプロイ」、デプロイメントの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

OC4J と JAAS Provider

Oracle Application Server Containers for J2EE は、HTTP および RMI クライアント接続を受け入れる J2EE コンテナです。これらの接続では、サーブレット、JavaServer Pages（JSP）および Enterprise JavaBeans（EJB）へのアクセスが許可されます。

J2EE コンテナにより、ビジネス・ロジックがリソースおよびライフ・サイクル管理から分離されます。これにより、開発者はエンタプライズ・インフラストラクチャの記述ではなくビジネス・ロジックの記述に集中できます。たとえば、Java サーブレットは、Web サーバーと統合された Web コンテナにコンポーネント、通信およびセッション管理のインフラストラクチャを提供することで、Web 開発作業を簡略化します。

OC4J の統合

JAAS Provider は、アプリケーションのセキュリティを強化するために Oracle Application Server Containers for J2EE と統合されます。この統合には、次のメリットがあります。

- Single Sign-On（SSO）との統合
- Java 2 のパーミッションを介したファイングレイン・アクセス制御
- `run-as` ID のサポート、委任のサポート（サーブレットから Enterprise JavaBeans へ）
- ユーザー・パスワードのセキュアなファイルベースの格納

JAZNUserManager

J2EE 環境における JAAS 統合のもう 1 つの主要コンポーネントは `JAZNUserManager` です。`JAZNUserManager` は、Oracle Application Server Containers for J2EE の `UserManager` インタフェースの実装です。

principals.xml の置換

OC4J の `principals.xml` ファイルは、`JAZNUserManager` による認証ほどセキュアではなく、柔軟性もありません。`JAZNUserManager` の機能は次のとおりです。

- 不明瞭化されたパスワードのセキュアな格納。
- 階層形式のロールなど、完全ロールベースのアクセス制御（RBAC）。
- Java 2 のパーミッション・モデルと JAAS の完全サポート。
- Java 2 のパーミッション・モデルに基づくセキュアな実装。これにより、非トラステッド（または一部トラステッド）・コードを JAAS Provider と同じ JVM で実行できます。

注意： 既存のアプリケーションを、`principals.xml` の使用から `JAZNUserManager` の使用へと移行することをお薦めします。移行指示は、5-21 ページの「principals.xml ファイルからのプリンシパルの移植（XML ベース・プロバイダのみ）」を参照してください。

JAZNUserManager の機能

JAZNUserManager には、6-4 ページの「principals.xml の置換」で説明した機能の他にも次のように多数の機能が用意されています。

- Single Sign-On（SSO）と Oracle Application Server Containers for J2EE の統合
- 非 SSO 環境での RealmLoginModule の統合
- カスタム・ログイン・モジュールのサポート
- ID の伝播
- ユーザーおよびグループ・オブジェクトの検索、読取り、編集、削除および管理
- セキュリティ制約の規定

認証環境

JAAS Provider は、J2EE アプリケーションの 3 つの異なるログイン認証環境と統合されます。

- **SSO**

  ログインの認証に Oracle AS Single Sign-On を使用

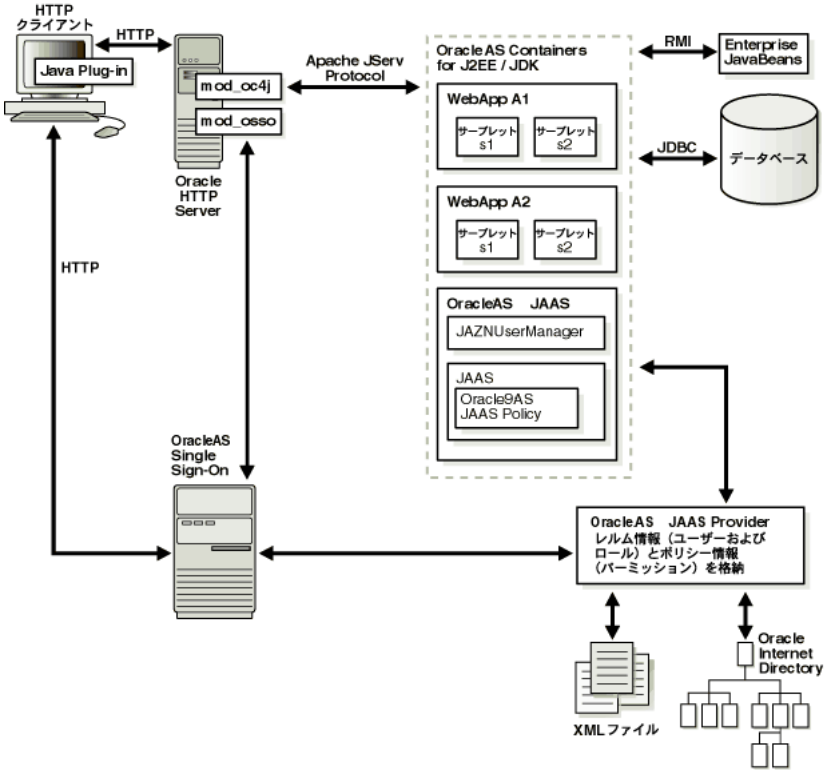
- **SSL**
  - クライアント証明書ベース認証に Secure Sockets Layer を使用
  - ログイン認証にログイン・モジュール（RealmLoginModule など）を使用
- **Basic 認証**
  - Oracle AS Single Sign-On を介さずに、ユーザーに対してユーザー名とパスワードを直接要求
  - ログイン認証にログイン・モジュール（RealmLoginModule など）を使用

次の項では、JAAS Provider と前述の各認証タイプがどのように統合されるかについて説明します。

JAAS Provider と SSO 対応アプリケーションの統合

SSO を使用すると、ユーザーは 1 組のログイン資格証明を使用して複数の勘定科目およびアプリケーションにアクセスできます。図 6-1 に、SSO 対応 J2EE 環境で動作するアプリケーションでの JAAS の統合を示します。

図 6-1 SSO 対応 J2EE 環境での Oracle コンポーネントの統合



SSO 対応 J2EE 環境 : 代表的な使用例

この項では、SSO 対応 J2EE 環境で HTTP クライアント・リクエストが開始された場合の、Oracle コンポーネントの役割について説明します。

1. HTTP クライアントが、Oracle Application Server Containers for J2EE（サーブレット実行用の Web コンテナ）によりホスティングされている Web アプリケーション（WebApp A1）にアクセスします。Oracle HTTP Server （Apache リスナーを使用）がリクエストを処理します。

2. mod_osso/Oracle HTTP Server がリクエストを受信して次を実行します。
   - WebApp A1 アプリケーションで HTTP クライアントの認証用に Web ベース SSO が必要かどうかを判別します。
   - HTTP クライアント・リクエストを Web ベース SSO の Oracle AS Single Sign-On にリダイレクトします（未認証のため）。
3. HTTP クライアントが、HTTP 認証または公開鍵インフラストラクチャ（PKI）認証を介して Oracle AS Single Sign-On から認証を受けます。その後、Oracle AS Single Sign-On が次を実行します。
   - ユーザーについて格納されているログイン資格証明を検証します。
   - SSO の Cookie（ユーザーの識別名とレルムを含む）を設定します。
   - WebApp A1 アプリケーション（Oracle Application Server Containers for J2EE 内）にリダイレクトします。
4. JAAS Provider が SSO ユーザーを取得します。
5. 最後のステップは、`<jazn-web-app>` 要素の `runas-mode` の設定に応じて異なります。

   `runas-mode` が `false` に設定されている場合は、次の処理が発生します。

   a. ターゲット・サーブレットが起動します。

   `runas-mode` が `true` に設定されている場合は、次の処理が発生します。

   b. JAAS Provider は、`Subject.doAs()` を介して `PrivilegedAction` ブロック内でターゲット・サーブレットを起動します。`JAZNUserManager` はセキュリティ制約を規定します。
      - `Subject.doAs()` がコールされると、JAAS は `getPermissions()` メソッドを介して JAAS Provider 内で SSO ユーザーに関連付けられているパーミッションを参照します。
      - JAAS Provider は、プロバイダ・タイプ（LDAP ベースまたは XML ベース）から指定の権限受領者に関連したパーミッションを取得し、必要に応じてポリシー・キャッシュを更新します。次に、付与されているパーミッション・セットを JAAS Provider ランタイムに戻します。
      - JAAS Provider ランタイムは、`getPermissions()` から戻されたパーミッションに基づいて新規 `AccessControlContext` を作成します。

   c. サーブレットのコードが、SSO ユーザーの `AccessControlContext` で実行されます。

   d. サーブレットのコードがオペレーティング・システムのファイル・システム内のファイルへの書込みを試行すると、`SecurityManager.checkPermission()` がコールされます。

e. JVM により次が実行されます。

   – 現行スレッドに関連付けられた認可コンテキストが検査されます。

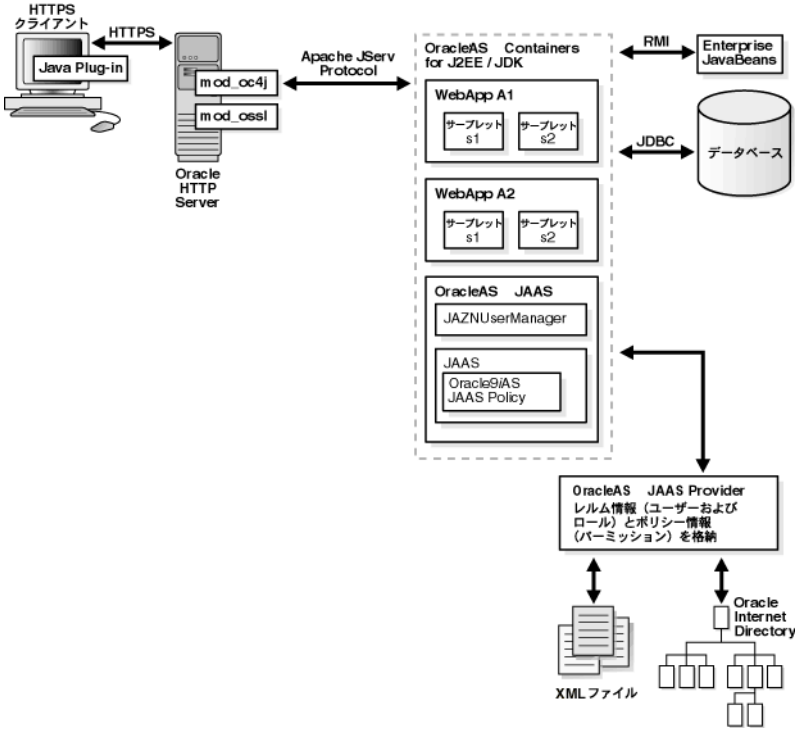
   – 現行サブジェクトにファイルへの書込みに必要なパーミッションが付与されているかどうかが判別されます。

f. `SecurityManager.checkPermission()` が安全に戻し、クライアント HTTP リクエストが進行します。

JAAS Provider と SSL 対応アプリケーションの統合

SSL は、インターネットでのメッセージ送信のセキュリティを管理するための業界標準プロトコルです。図 6-2 に、SSL 対応 J2EE 環境で動作するアプリケーションでの JAAS の統合を示します。

図 6-2 SSL 対応 J2EE 環境での Oracle コンポーネントの統合



SSL 対応 J2EE 環境 : 代表的な使用例

この項では、SSL 対応 J2EE 環境で HTTP クライアント・リクエストが開始された場合の、Oracle コンポーネントの役割について説明します。この環境では、Oracle AS Single Sign-On は使用されません。ログイン・モジュール（RealmLoginModule など）が使用されます。

1. HTTP クライアントが、Oracle Application Server Containers for J2EE（サーブレット実行用の Web コンテナ）によりホスティングされている Web アプリケーション（WebApp A1）にアクセスします。Oracle HTTP Server （Apache リスナーを使用）がリクエストを処理します。
2. mod_ossl/Oracle HTTP Server がリクエストを受信し、WebApp A1 アプリケーションで HTTP クライアントに SSL サーバー認証が必要かどうかを判別します。
3. サーバーまたはクライアント、あるいはその両方の Wallet 証明書が構成されている場合、HTTP クライアントは OHS のサーバー証明書を受け入れてクライアント証明書を提供するように要求されます。
4. JAAS Provider が SSL クライアント証明書を取得します。
5. JAAS Provider が証明書から SSL ユーザーを取得します。
6. 最後のステップは、<jazn-web-app> 要素に指定されている runas-mode に応じて異なります。

   runas-mode が false に設定されている場合は、ターゲット・サーブレットが起動します。

   runas-mode が true に設定されている場合は、次の処理が発生します。

   a. JAAS Provider は、Subject.doAs() を介して PrivilegedAction ブロック内でターゲット・サーブレットを起動します。JAZNUserManager はセキュリティ制約を規定します。

      – Subject.doAs() がコールされると、このメソッドは getPermissions() メソッドを介して SSL ユーザーに関連付けられているパーミッションを参照します。

      – JAAS Provider は、プロバイダ・タイプ（LDAP ベースまたは XML ベース）から指定の権限受領者に関連したパーミッションを取得し、必要に応じてポリシー・キャッシュを更新します。次に、付与されているパーミッション・セットを JAAS Provider ランタイムに戻します。

      – JAAS Provider ランタイムは、getPermissions() から戻されたパーミッションに基づいて新規 AccessControlContext を作成します。

   b. サーブレットのコードが、SSL ユーザーの AccessControlContext で実行されます。

   c. サーブレットのコードがオペレーティング・システムのファイル・システム内のファイルへの書込みを試行すると、SecurityManager.checkPermission() がコールされます。

d. JVM により次が実行されます。

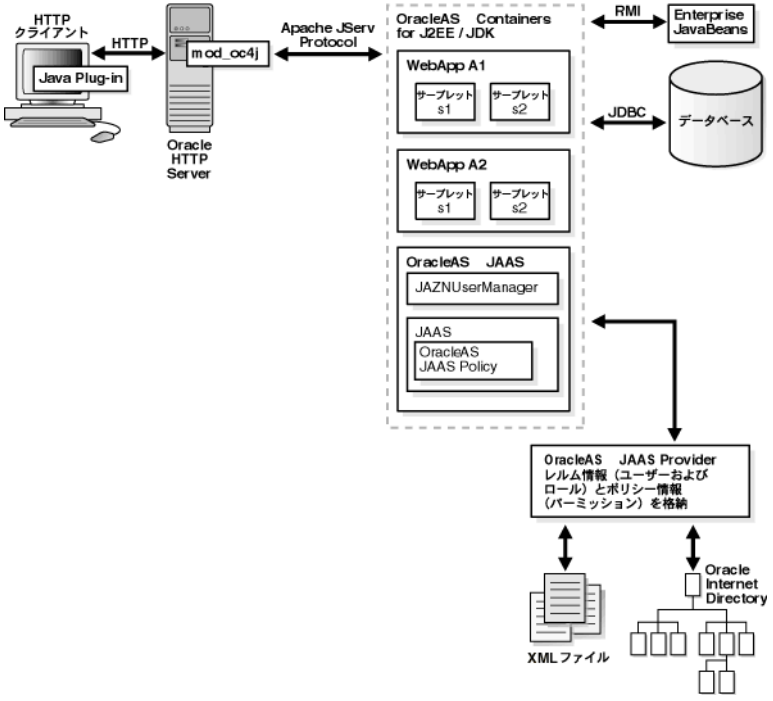
- 現行スレッドに関連付けられた認可コンテキストが検査されます。
- 現行サブジェクトにファイルへの書込みに必要なパーミッションが付与されているかどうかが判別されます。

e. `SecurityManager.checkPermission()` が安全に戻し、クライアント HTTP リクエストが進行します。

JAAS Provider と Basic 認証の統合

Basic 認証は Oracle AS Single Sign-On をバイパスします。図 6-3 に、J2EE 環境で Basic 認証用に構成されているアプリケーションでの特定の JAAS の統合を示します。

図 6-3 J2EE 環境での Oracle コンポーネントの統合



J2EE 環境での Basic 認証 : 代表的な使用例

この項では、Basic 認証用に構成された J2EE 環境で HTTP クライアント・リクエストが開始された場合の、Oracle コンポーネントの役割について説明します。この環境では、Oracle AS Single Sign-On は使用されません。ログイン・モジュール（`RealmLoginModule` など）が使用されます。

> **注意 :** Basic 認証を構成している場合、OC4J はユーザー資格証明が必要になるたびに `RealmLoginModule` を起動します。たとえば、リクエストが保護ページにヒットすると、OC4J は JAAS Provider に対してユーザー認証を要求し、`RealmLoginModule` が起動し、HTTP を介してブラウザ経由でユーザーから送信された資格証明を使用してユーザーが認証されます。

1. HTTP クライアントが、Oracle Application Server Containers for J2EE（サーブレット実行用の Web コンテナ）によりホスティングされている Web アプリケーション（WebApp A1）にアクセスします。
2. JAAS Provider がユーザーを取得します。
3. 最後のステップは、`jazn-web-app` 要素の `runas-mode` の設定に応じて異なります。

   `runas-mode` が `false` に設定されている場合は、次の処理が発生します。

   a. ターゲット・サーブレットが起動します。

   `runas-mode` が `true` に設定されている場合は、次の処理が発生します。

   a. JAAS Provider は、`Subject.doAs()` を介して `PrivilegedAction` ブロック内でターゲット・サーブレットの `service()` メソッドを起動します。`JAZNUserManager` はセキュリティ制約を規定します。

      – `Subject.doAs()` がコールされると、JAAS は `getPermissions()` メソッドを介して JAAS Provider 内で SSO ユーザーに関連付けられているパーミッションを参照します。

      – JAAS Provider は、プロバイダ・タイプ（LDAP ベースまたは XML ベース）から指定の権限受領者に関連したパーミッションを取得し、必要に応じてポリシー・キャッシュを更新します。次に、付与されているパーミッション・セットを JAAS Provider ランタイムに戻します。

      – JAAS Provider ランタイムは、`getPermissions()` から戻されたパーミッションに基づいて新規 `AccessControlContext` を作成します。

   b. サーブレットのコードが、ユーザーの `AccessControlContext` で実行されます。

   c. サーブレットのコードがオペレーティング・システムのファイル・システム内のファイルへの書込みを試行すると、`SecurityManager.checkPermission()` がコールされます。

d. JVM により次が実行されます。

- 現行スレッドに関連付けられた認可コンテキストが検査されます。
- 現行サブジェクトにファイルへの書込みに必要なパーミッションが付与されているかどうかが判別されます。

e. `SecurityManager.checkPermission()` が安全に戻し、クライアント HTTP リクエストが進行します。

関連項目： J2EE の詳細は、次の URL を使用して Sun 社の Java ドキュメントを参照してください。

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

J2EE と JAAS Provider のロールのマッピング

J2EE 環境で JAAS 統合アプリケーションを作成するアプリケーション開発者は、J2EE ロールおよび JAAS ロールという 2 つの異なるロール・タイプを使用できます。Oracle Application Server Containers for J2EE のグループ・マッピングを使用してこれらのロール・タイプをマップすると、ユーザーはそのロールにマップされていれば、定義済のロール・パーミッション・セットを使用してアプリケーションにアクセスできます。

この項では、これらのロール・タイプと両者のマッピング方法について説明します。

- J2EE のセキュリティ・ロール
- JAAS Provider のロールとユーザー
- J2EE のセキュリティ・ロールへの OC4J グループ・マッピング

J2EE のセキュリティ・ロール

J2EE 開発環境には、サーブレットと JavaServer Pages（JSP）について `web.xml` ファイルに定義されている移植可能なセキュリティ・ロール機能が組み込まれています。セキュリティ・ロールでは、アプリケーションについて一連のリソース・アクセス権が定義されます。プリンシパル（この場合は JAAS ユーザーまたはロール）がロールにマップされている場合は、プリンシパルをセキュリティ・ロールに関連付けることで、そのプリンシパルに定義済のアクセス権を割り当てます。たとえば、アプリケーションでセキュリティ・ロール `sr_developer` を定義します。

```
<security-role>
     <role-name>sr_developer</role-name>
</security-role>
```

また、`sr_developer` ロールのアクセス権を定義します。

```
 <security-constraint>
    <web-resource-collection>
      <web-resource-name>access to the entire application</web-resource-name>
```

```
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
       <!-- authorization -->
    <auth-constraint>
      <role-name>sr_developer</role-name>
    </auth-constraint>
  </security-constraint>
```

JAAS Provider のロールとユーザー

JAAS のロールとユーザーは、プロバイダ・タイプ（LDAP ベースまたは XML ベース）に応じて定義されます。

たとえば、XML ベース・プロバイダ・タイプの場合、開発者は jazn-data.xml ファイルにロール要素としてリストされます。

```
<role>
  <name>developer</name>
  <members>
    <member>
      <type>user<type>
      <name>john<name>
    </member>
  </members>
</role>
```

J2EE のセキュリティ・ロールへの OC4J グループ・マッピング

Oracle Application Server Containers for J2EE（OC4J）では、J2EE の web.xml ファイルに定義されている移植可能な J2EE セキュリティ・ロールを orion-application.xml ファイル内のグループにマップできます。

プロバイダ環境で定義されているロールとユーザーは、orion-application.xml ファイル内で Oracle Application Server Containers for J2EE developer グループのロールにマップされます。

たとえば、sr_developer セキュリティ・ロールはグループ developer にマップされます。

```
<security-role-mapping name="sr_developer">
       <group name="developer" />
  </security-role-mapping>
```

<security-role-mapping> 要素内の <group> が JAAS Provider のロールに対応していることに注意してください。したがって、この関連付けにより、developer グループは sr_developer セキュリティ・ロールに許可されているリソースにアクセスできます。

この例では、ユーザー john は developer ロールのメンバーとしてリストされています。developer グループは orion-application.xml ファイル内の J2EE のセキュリティ・ロール sr_developer にマップされるため、john は sr_developer ロールで定義されたアプリケーション・リソースへのアクセス権を持ちます。

J2EE 環境での認証

認証とは、通常はシステム内のリソースへのアクセス権を付与するための前提条件として、コンピュータ・システム内でユーザーの ID を検証する処理です。J2EE 環境では、ユーザー認証は次のいずれかにより実行されます。

- Oracle AS Single Sign-On（SSO 環境の場合）または JAAS Provider の RealmLoginModule または他のログイン・モジュール（非 SSO 環境の場合）

  HTTP リクエストをターゲット・サーブレットにディスパッチする前に、JAZNUserManager は HTTP リクエスト・オブジェクトから認証済ユーザー情報（mod_osso により設定）を取得して、Oracle Application Server Containers for J2EE 内で JAAS サブジェクトを設定します。

- 次のいずれかのユーザー・マネージャ
  - JAZNUserManager
  - XMLUserManager
  - 開発者提供の UserManager

認証済 ID を使用した実行

JAZNUserManager の構成として、フィルタを適用してターゲット・サーブレットが認証済 ID または run-as ID に関連したパーミッションとロールで実行できるように選択できます。そのためには <jazn-web-app> 要素を構成します。

> **関連項目：** <jazn-web-app> 要素など、JAZNUserManager フィルタのオプションと構成の詳細は、6-4 ページの「JAZNUserManager」を参照してください。

認証情報の取得

次の javax.servlet.HttpServletRequest API は、サーブレット内で認証情報を取得します。

- getRemoteUser: 認証済ユーザー名
- getAuthType: 認証方式
- getUserPrincipal: 認証済プリンシパル・オブジェクト
- getAttribute("java.security.cert.X509certificate"): SSL クライアント証明書

認証は、Subject.doAs() のコールで開始されます。

J2EE 環境での認可

認可とは、ユーザーにアクセス権と権限を付与する処理です。この項では、サーブレット内での認証について説明します。

サーブレットが doAs() を許可するように構成されている場合、JAZNUserManager は Subject.doAs() ブロック内で認証済ターゲット・サーブレットを起動して、ターゲット・サーブレット内で JAAS ベースの認可を使用可能にします。

認可は、次を介して実行されます。

- JAZNUserManager
- Java 2 セキュリティ・モデルに基づくメソッド
  - サーブレット内の Servlet.service()
  - クライアント内の Subject.doAs() および Subject.doAsPrivileged()
  - サーバー内の SecurityManager.checkPermission()

関連項目： 3-14 ページ「<jazn-web-app> でのサーブレット認証（runas-mode および doasprivileged-mode）の構成」

7

カスタム LoginModule

この章では、Oracle Application Server Containers for J2EE（OC4J）の JAAS Provider で使用する `LoginModule` を記述してインストールする方法について説明します。この章には、次の項目が含まれます。

- JAAS のカスタム LoginModule と OC4J の統合
- パッケージとデプロイメント
- 構成
- 単純なログイン・モジュールによる J2EE の統合

> **注意：** ユーザー管理は JAAS 仕様の対象外であるため、カスタム `LoginModule` を使用するようにアプリケーションを構成すると、そのアプリケーションでは `UserManager` API を効果的に使用できなくなります。ただし、J2EE の API はアプリケーションで引き続き機能します。

JAAS のカスタム LoginModule と OC4J の統合

OC4J の JAAS サポート機能は JAAS 1.0 仕様に完全に準拠しているため、必要に応じて JAAS 準拠の任意の `LoginModule` 実装をプラグインできます。OC4J には、OracleAS の Single Sign-On（SSO）が使用できない環境向けに、J2EE のセキュリティ制約を XML ベースまたは LDAP ベース・プロバイダ・タイプと併用する `LoginModule` である、`RealmLoginModule` が組み込まれています。Oracle Internet Directory（OID）の使用中は、Oracle Identity Management を使用して他の認証および ID 管理システムと統合することをお薦めします。

詳細は、『Oracle Identity Management 概要および配置プランニング・ガイド』を参照してください。

カスタムの JAAS `LoginModule` は、OracleAS Identity Management を使用できず、ユーザーが外部リポジトリで定義されている場合に適しています。この場合は XML ベース・プロバイダ・タイプを使用して `LoginModule` を構成できますが、次の前提事項を考慮する必要があります。

1. **開発**。J2EE のセキュリティ制約を利用する必要があるかどうか。
2. **開発、パッケージおよびデプロイメント**。J2SE1.4 付属のログイン・モジュールを使用するか。または社内やサード・パーティのログイン・モジュールをデプロイするか。

注意： カスタム・ログイン・モジュールは、XML ベース・プロバイダでのみサポートされます。

パッケージとデプロイメント

J2SE 1.3 および 1.4 に付属のデフォルト・ログイン・モジュール（J2SE1.4 の `com.sun.security.auth.module.Krb5LoginModule` など）を 1 つ以上使用する場合、追加構成は不要です。OracleAS の JAAS Provider は、デフォルトのログイン・モジュールを検出できます。

カスタム・ログイン・モジュールとともにアプリケーションをデプロイする場合は、ログイン・モジュールをデプロイし、そのモジュールを実行時に検出できるように JAAS Provider を適切に構成する必要があります。

カスタム・ログイン・モジュールをパッケージおよびデプロイするときには、次の 4 つのオプションから選択できます。

- 標準拡張機能またはオプション・パッケージのデプロイ
- J2EE アプリケーション内でのデプロイ
- OC4J のクラス・ロード・メカニズムの使用
- JAAS Provider のクラス・ロード・メカニズムの使用

この項では、これらのオプションの詳細を説明します。

標準拡張機能またはオプション・パッケージのデプロイ

ログイン・モジュールを標準拡張機能としてデプロイすると、それを JAAS Provider で検出できます。追加構成は不要です。この方法でデプロイしたログイン・モジュールは、複数のアプリケーションで共有できます。

たとえば、ログイン・モジュールを標準拡張機能としてデプロイする方法の 1 つは、`${java.home}/lib/ext` ディレクトリにデプロイすることです。

関連項目：
`http://java.sun.com/j2se/1.4/docs/guide/extensions`

J2EE アプリケーション内でのデプロイ

ログイン・モジュールを複数のアプリケーション間で共有するのではなく、単一の J2EE アプリケーションでのみ使用する場合は、単にアプリケーションの一部としてパッケージすると、JAAS Provider で検出できます。追加構成は不要です。

以降のアプリケーションに同じ `LoginModule` が必要になった場合は、ログイン・モジュールと関連クラスを新規アプリケーションとともに再パッケージする必要があります。

複数のアプリケーションで同じ `LoginModule` を共有可能にする必要があるが、`LoginModule` を拡張機能としてデプロイできない場合は、OC4J のクラス・ロード・メカニズムまたは JAAS Provider のクラス・ロード・メカニズムの使用を検討できます。

OC4J のクラス・ロード・メカニズムの使用

JAAS Provider は、OC4J のクラス・ロード・アーキテクチャと統合されています。デプロイしたカスタム・ログイン・モジュールがアプリケーションの `classpath` の一部になるようにアプリケーションを構成すると、それを JAAS Provider で検出できます。

その方法の 1 つは、次のどちらかのファイルで `<library>` 要素を使用することです。

- `application.xml`
- `orion-application.xml`

関連項目： `<library>` 要素の詳細は、『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

JAAS Provider のクラス・ロード・メカニズムの使用

なんらかの理由でアプリケーションの `classpath` を構成できない場合は、JAAS Provider のクラス・ロード・メカニズムを使用できます。ログイン・モジュールをデプロイし、`<jazn>` タグの `classpath` プロパティを使用して位置を指定します。`<jazn>` タグのプロパティの詳細は、表 3-3「(XML ベース・プロバイダ) orion-application.xml 内の <jazn> タグ」を参照してください。

構成

カスタム・ログイン・モジュールを利用するようにアプリケーションを構成するには、次のファイルを変更します。

- jazn-data.xml
- orion-application.xml

これらのファイルの詳細を次に説明します。

jazn-data.xml

このファイルは、XML ベース・プロバイダのリポジトリとして機能します。

1 つの OC4J インスタンスに多数の jazn-data.xml ファイルを関連付けることができますが、デフォルトの jazn.xml に指定した jazn-data.xml が OracleAS の JAAS Provider のデフォルト・リポジトリとして機能します。

Oracle では、カスタム・ログイン・モジュールに関連する XML ベース・プロバイダのみがサポートされることに注意してください。

以降の各項では、次の 2 つの XML 要素について説明します。

- <jazn-loginconfig>
- <jazn-policy>

<jazn-loginconfig>

このタグには、アプリケーションをログイン・モジュールに関連付ける情報が含まれます。

次に例を示します。

```
<jazn-loginconfig>
  <application>
    <name>sampleLM</name>
    <login-modules>
       <login-module>
          <class>sample.SampleLoginModule</class>
          <control-flag>required</control-flag>
       </login-module>
    </login-modules>
  </application>
</jazn-loginconfig>
```

このサンプル断片は、アプリケーション sampleLM をログイン・モジュール sample.SampleLoginModule に関連付けています。

<jazn-policy>

このタグには、権限受領者をパーミッションに関連付ける情報が含まれます。

次に例を示します。

```
<jazn-policy>
  <grant>
   <grantee>
     <principals>
      <principal>
        <class>sample.SampleUser</class>
        <name>admin</name>
      </principal>
    </principals>
   </grantee>
   <permissions>
     <permission>
       <class>com.evermind.server.rmi.RMIPermission</class>
       <name>login</name>
     </permission>
   </permissions>
  </grant>
</jazn-policy>
```

このサンプル断片は、ターゲット名 `login` を持つパーミッション `com.evermind.server.rmi.RMIPermission` を、クラス `sample.SampleUser` を持つプリンシパル `ray` に関連付けています。

> **注意：** `jazn-data.xml` の内容は、JAZN Admintool を使用して管理することをお薦めします。

JAZN Admintool の詳細は、第 5 章「JAZN Admintool の使用」を参照してください。

orion-application.xml

このファイルには、OC4J 固有のアプリケーション構成情報が含まれています。ここでは次のタグの詳細について説明します。

- <jazn>
- <security-role-mapping>
- <library>

<jazn>

<jazn> タグの詳細は、3-6 ページの「<jazn> タグ」を参照してください。

次のプロパティは LoginModule 構成固有です。

- role.mapping.dynamic

  このプロパティを true に設定すると、JAAS Provider に対して、アプリケーション固有の jazn-data.xml で定義されたユーザーおよびロールではなく、現行の Subject に基づいて認可チェックを実行するように指示したことになります。

  LoginModule インスタンスでは、認可プロセスでプリンシパルを考慮に入れるために、認証プロセスのコミット・フェーズで適切なプリンシパル（ユーザー、ロールまたはグループ）が Subject インスタンスに関連付けられていることを確認する必要があります。Subject へのプリンシパルのこの関連付けは、通常、標準の JAAS API を使用して実装されます。

- classpath

  このプロパティは、サード・パーティ・クラスと JAR ファイルが他の場所で見つからない場合の検索場所を JAAS Provider に対して指定するときに使用します。次に例を示します。

```
<jazn provider="XML" location="./jazn-data.xml">
   <property name="classpath"
value="../../shared/lib/sample.jar;../../shared/lib/samplemodule.jar" />
   <property name="role.mapping.dynamic" value="true" />
</jazn>
```

  詳細は、表 3-3「（XML ベース・プロバイダ）orion-application.xml 内の <jazn> タグ」を参照してください。

<security-role-mapping>

このオプション・タグには、静的なセキュリティ - ロール・マッピング情報を記述します。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。

<library>

このタグでは、アプリケーションに関連した `classpath` を設定します。可能な場合は、`<jazn>` タグ内で `classpath` プロパティのかわりにこのタグを使用してください。次に例を示します。

```
<library path="../../shared/lib/sample.jar"/>
<library path="../../shared/lib/samplemodule.jar"/>
```

詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

単純なログイン・モジュールによる J2EE の統合

単純な `LoginModule` の開発作業は、標準的な開発、パッケージおよびデプロイメント・サイクルをたどります。以降の各項では、このサイクルの各ステップについて説明します。

開発

JAAS SPI に従って JAAS 準拠の `LoginModule` を開発します（詳細は、`javax.security.auth.spi.LoginModule` の Javadoc を参照してください）。

パッケージ

次のどちらかの方法で `LoginModule` をパッケージします。

- `LoginModule` の各クラスをアプリケーションの EAR ファイルの一部としてパッケージします。Web アプリケーションの場合は、各クラスを `WEB-INF/classes` の下に組み込みます。
- これを別個にパッケージし、クラス・ロード・メカニズムを使用して参照します。

デプロイメント

LoginModule をグローバル jazn-data.xml ファイルにデプロイする手順は、次のとおりです。

1. アプリケーションのログイン・モジュールを <application> タグで登録します。

   次のエントリでは、sampleLM アプリケーションにアクセスするユーザーの認証用にログイン・モジュール sample.SampleLoginModule が登録されます。

```
 <application>
         <name>sampleLM</name>
         <login-modules>
            <login-module>
               <class>sample.SampleLoginModule</class>
               <control-flag>required</control-flag>
               <options>
                  <option>
                     <name>debug</name>
                     <value>true</value>
                  </option>
               </options>
            </login-module>
         </login-modules>
      </application>
```

2. **オプション**。関連するパーミッションをユーザーとロールに付与します。

   たとえば、プリンシパル admin が EJB アクセスを必要とする場合は、admin にパーミッション com.evermind.rmi.RMIPermission を付与する必要があります。

```
<grant>
  <grantee>
    <principals>
      <principal>
        <class>sample.SampleUser</class>
        <name>admin</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>com.evermind.server.rmi.RMIPermission</class>
      <name>login</name>
    </permission>
  </permissions>
</grant>
```

LoginModule をアプリケーション固有の orion-application.xml ファイルにデプロイする手順は、次のとおりです。

1. <jazn> のプロパティ role.mapping.dynamic を true に設定します。

```
<jazn provider="XML" location="./jazn-data.xml" >
  <property name="role.mapping.dynamic" value="true" />
</jazn>
```

2. 適切な <security-role-mapping> エントリを作成します。

```
<security-role-mapping name="sr_developer">
  <user name="developer" />
</security-role-mapping>
<security-role-mapping name="sr_manager">
  <group name="managers" />
</security-role-mapping>
```

8

JAAS と Enterprise Manager

JAAS の LDAP ベース・プロバイダは、情報を Oracle Internet Directory（OID）に格納します。この章では、Oracle Enterprise Manager を使用して Oracle Application Server Containers for J2EE（OC4J）の JAAS Provider 内のデータを管理する方法について説明します。

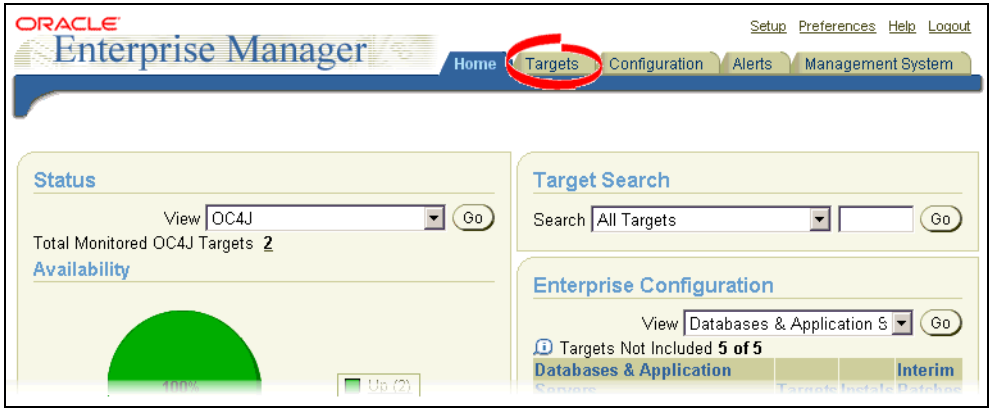
この章には、次の項目が含まれます。

起動

Oracle Enterprise Manager の OC4J ホームページにアクセスする手順は、次のとおりです。

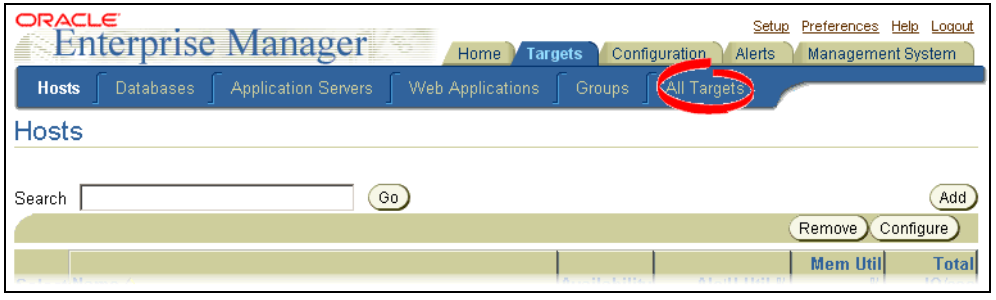
1. Enterprise Manager のホームで「**ターゲット**」をクリックします。

図 8-1 Enterprise Manager の「ホーム」タブ



2. 「ターゲット」タブの「**全ターゲット**」をクリックします。

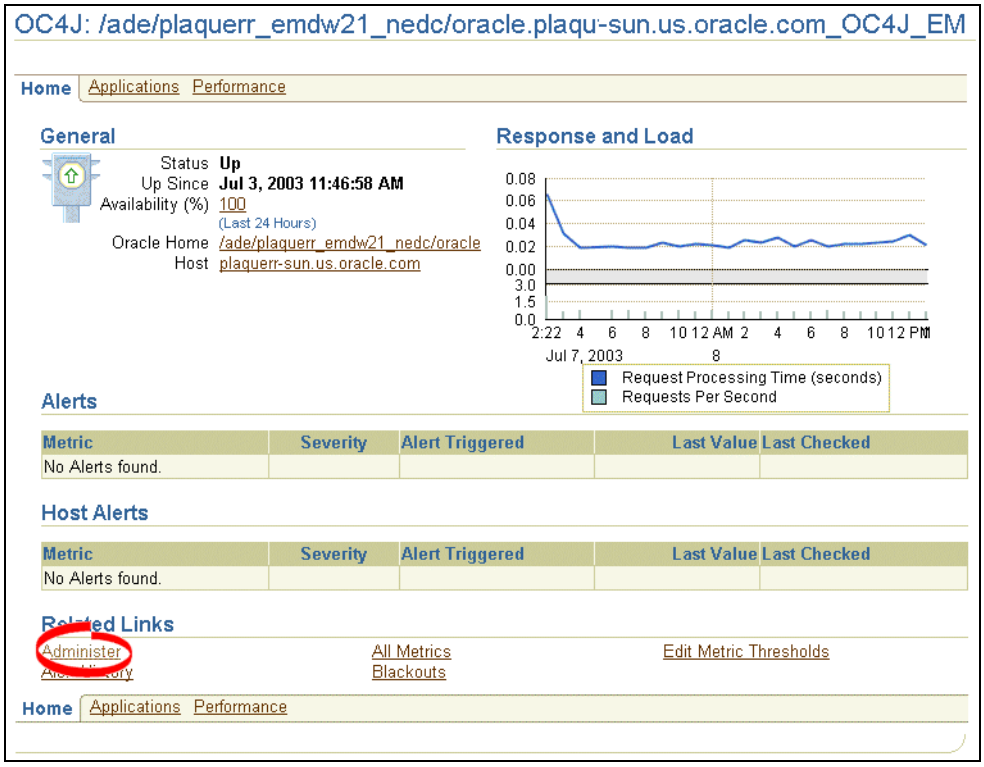
図 8-2 Enterprise Manager の「ターゲット」タブ



3. 「全ターゲット」ページで、特定の OC4J インスタンスを選択します。

4. OC4J インスタンスのホームページで、ページの下部付近にある「関連リンク」領域の「**管理者**」をクリックします。

図 8-3 OC4J インスタンスのホームページ



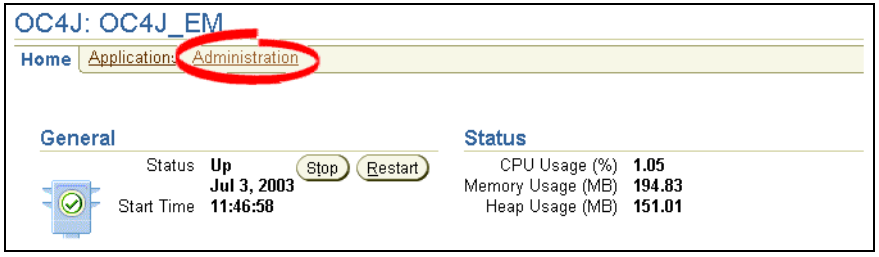
> **注意：** 管理者ページにログインするために、ユーザー名とパスワードの指定が必要な場合もあります。

5. Enterprise Manager の OC4J ホームページから、次のオプションのいずれかを選択します。
   - グローバル・アプリケーションのセキュリティ設定を編集するには、「**管理**」をクリックします。8-4 ページの「グローバル・セキュリティ設定の編集」を続行します。
   - 個別のモジュールのセキュリティ設定を編集するには、「**アプリケーション**」をクリックします。8-5 ページの「個別のセキュリティ設定の編集」を続行します。

グローバル・セキュリティ設定の編集

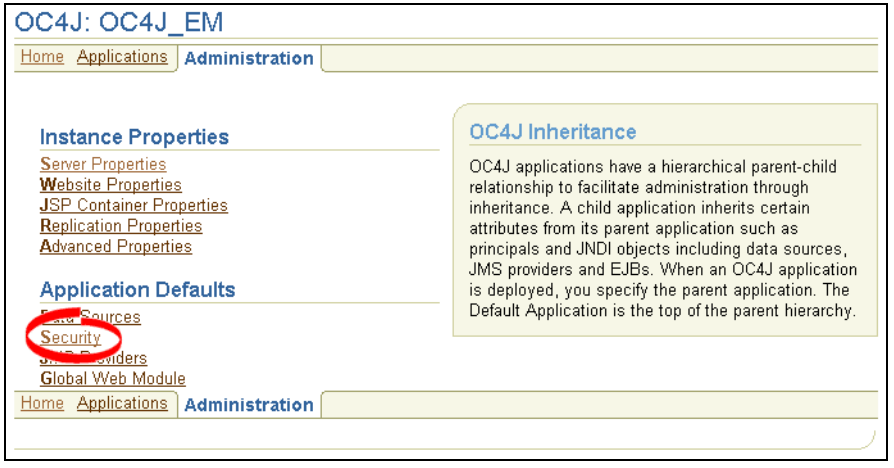
1. Enterprise Manager の OC4J ホームページで、「**管理**」をクリックします。

図 8-4 Oracle Enterprise Manager の OC4J ホームページ



2. 「管理」ページの「**セキュリティ**」をクリックします。

図 8-5 Oracle Enterprise Manager の「管理」ページ

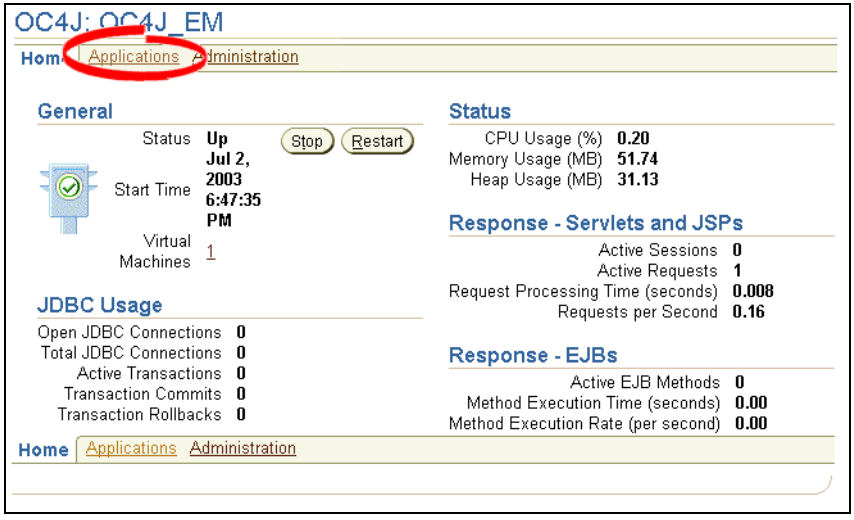


セキュリティ・ページが表示されます（8-9 ページの図 8-10 を参照）。

個別のセキュリティ設定の編集

1. Enterprise Manager の OC4J ホームページで、「**アプリケーション**」をクリックします。

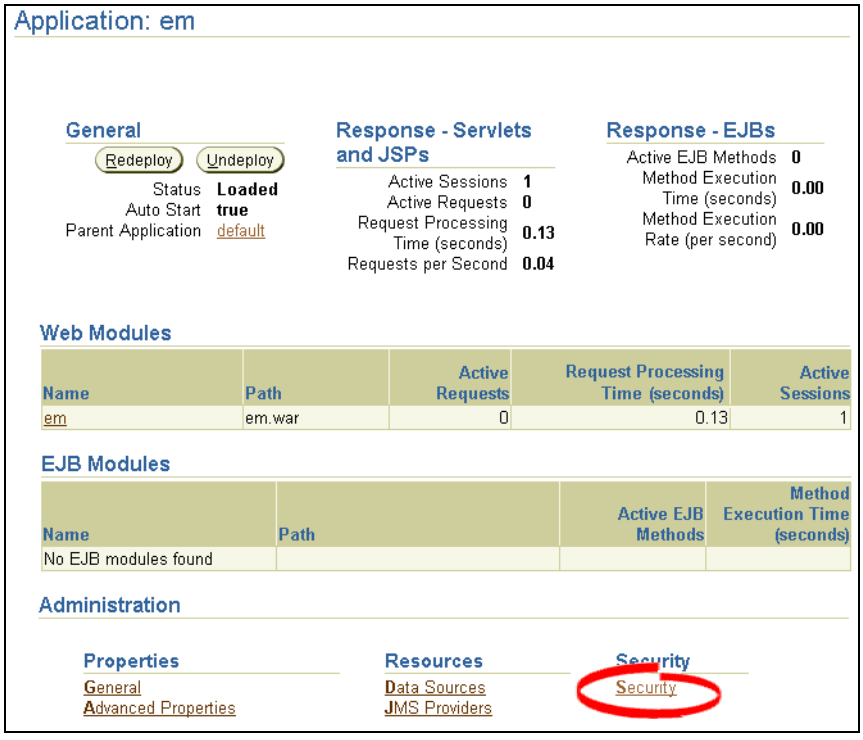
図 8-6 Oracle Enterprise Manager の OC4J ホームページ



2. デプロイされたアプリケーションを選択して、「**デフォルト**」をクリックします。

3. モジュールの「アプリケーション」ページで「**セキュリティ**」をクリックします。

図 8-7 Oracle Enterprise Manager の「アプリケーション」ページ

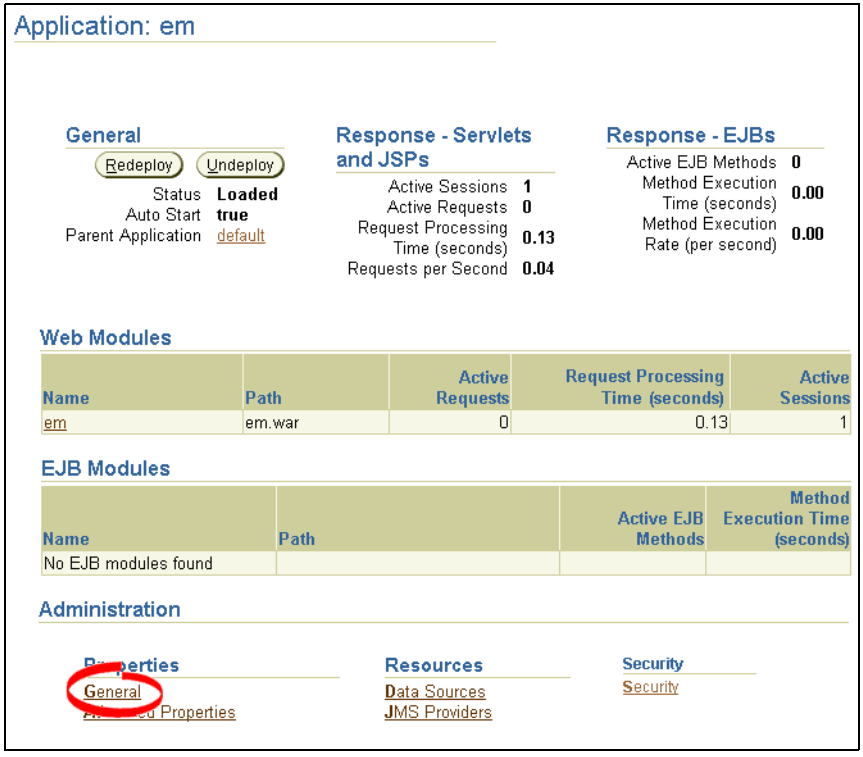


「セキュリティ」ページが表示されます（8-9 ページの図 8-10 を参照）。

UserManager の選択

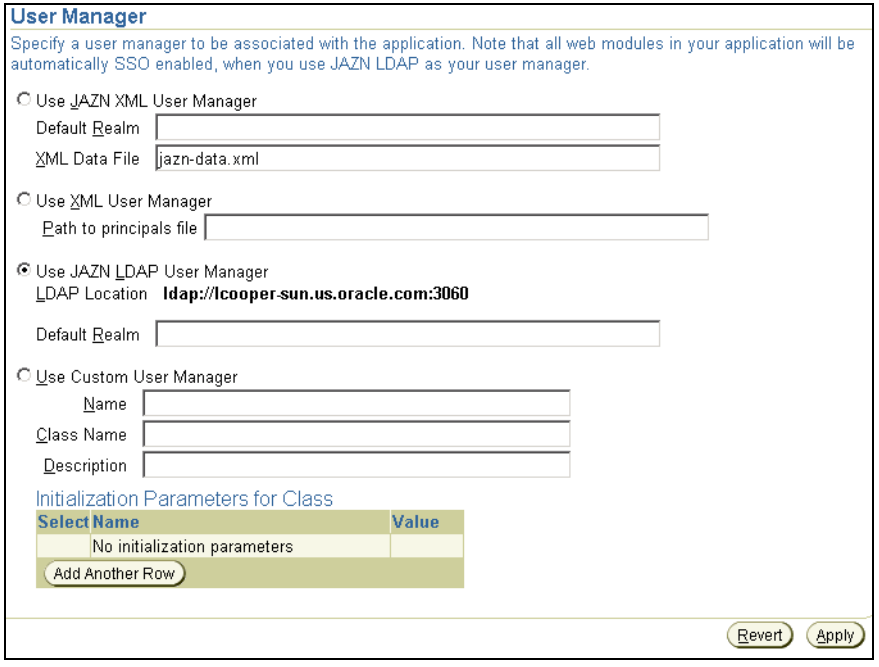
1. モジュールの「アプリケーション」ページ（8-6 ページの図 8-7 を参照）で、「管理」領域の「プロパティ」下の「**一般**」をクリックします。

図 8-8 Oracle Enterprise Manager の「アプリケーション」ページ



2. 「プロパティ」画面で、「**ユーザー・マネージャ**」領域にスクロールします。

図 8-9「プロパティ」ページの「ユーザー・マネージャ」領域

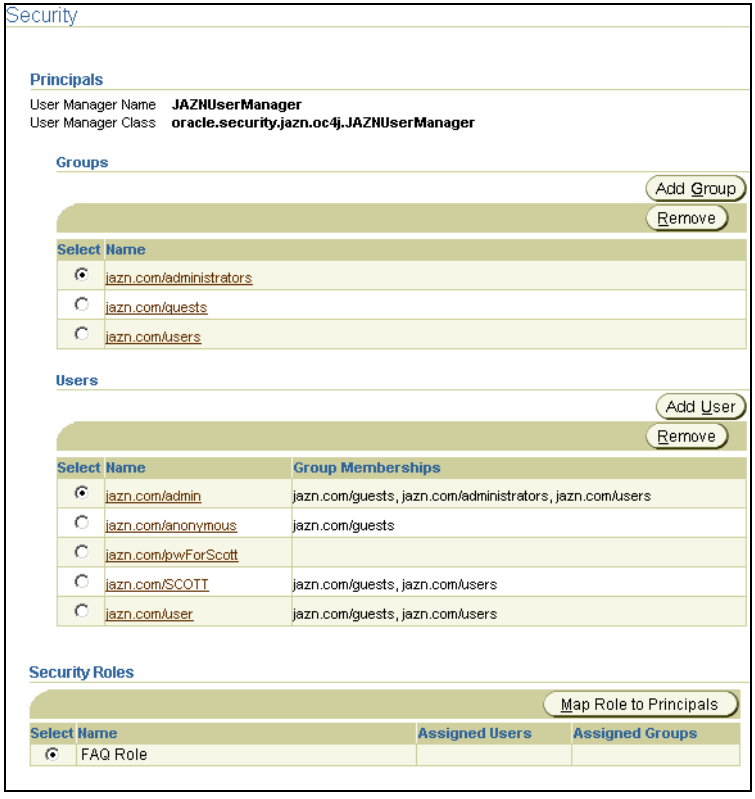


3. 使用するユーザー・マネージャをクリックし、適切なパス名およびレルム情報を入力します。

セキュリティ・ロールのマッピング

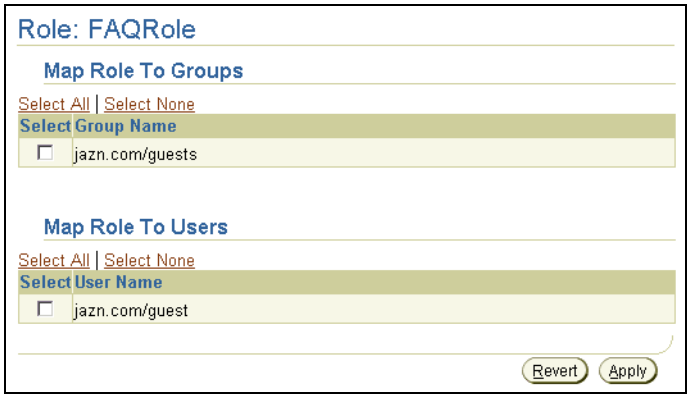
1. 8-2 ページの「起動」の説明に従って「**セキュリティ**」ページにナビゲートし、「セキュリティ・ロール」領域の「**ロールをプリンシパルにマップ**」をクリックします。

図 8-10「セキュリティ」ページ



2. ロールにマップするグループまたはユーザー（あるいはその両方）を選択して、「**適用**」をクリックします。

図 8-11「セキュリティ：ロールのマップ」画面

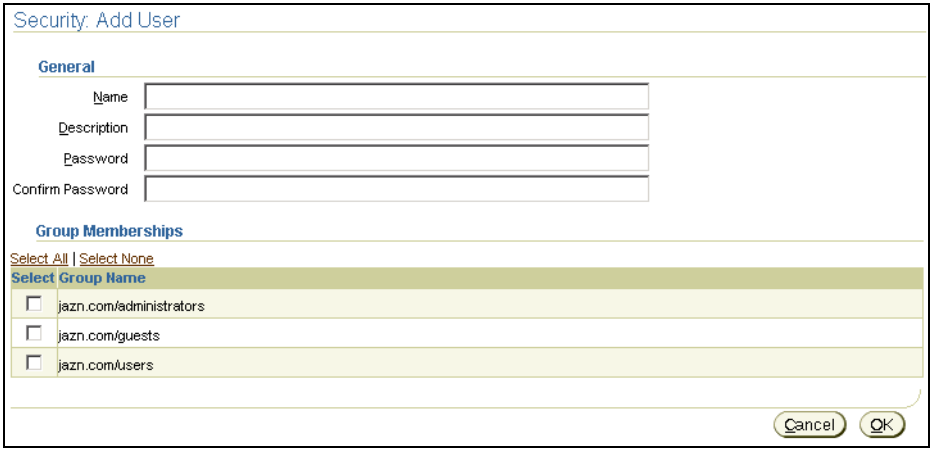


ユーザーの作成

> **注意：** Enterprise Manager で管理されるのは、XML ベースのロールとユーザーのみです。LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

1. 「セキュリティ」ページ（8-9 ページの図 8-10 を参照）で、「ユーザー」領域の「**ユーザーの追加**」をクリックします。「**セキュリティ：ユーザーの追加**」画面が表示されます。

図 8-12「セキュリティ：ユーザーの追加」画面



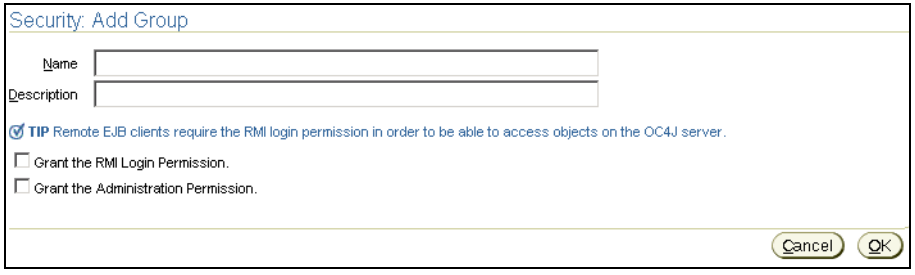
2. 「**名前**」、「**説明**」、「**パスワード**」および「**パスワードの確認**」フィールドに入力し、ユーザーをメンバーにするグループのチェックボックスをオンにします。「**OK**」をクリックします。

グループの作成

> **注意：** Enterprise Manager で管理されるのは、XML ベースのロールとユーザーのみです。LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

1. 「セキュリティ」ページ（8-9 ページの図 8-10 を参照）で、「グループ」領域の「**グループの追加**」をクリックします。「**セキュリティ：グループの追加**」画面が表示されます。

図 8-13「セキュリティ：グループの追加」画面



2. 「**名前**」および「**説明**」フィールドに入力し、グループに付与するパーミッションのチェックボックスをオンにします。「**OK**」をクリックします。

ユーザーまたはグループの削除

> **注意：** Enterprise Manager で管理されるのは、XML ベースのロールとユーザーのみです。LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

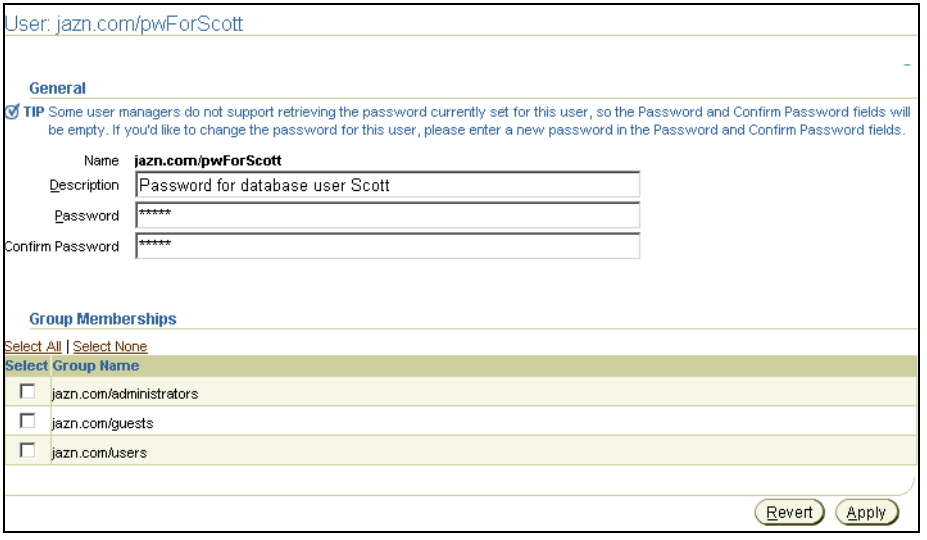
1. 「セキュリティ」ページ（8-9 ページの図 8-10 を参照）で、該当するリストからユーザーまたはグループを選択します。
2. 「**削除**」をクリックします。
3. 指定したユーザーまたはグループを削除するかどうかを尋ねる確認画面が表示されます。「**はい**」をクリックします。

ユーザーの編集

> **注意：** Enterprise Manager で管理されるのは、XML ベースのロールとユーザーのみです。LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

1. 「セキュリティ」ページ（8-9 ページの図 8-10 を参照）でユーザーを選択します。

図 8-14「ユーザー」画面



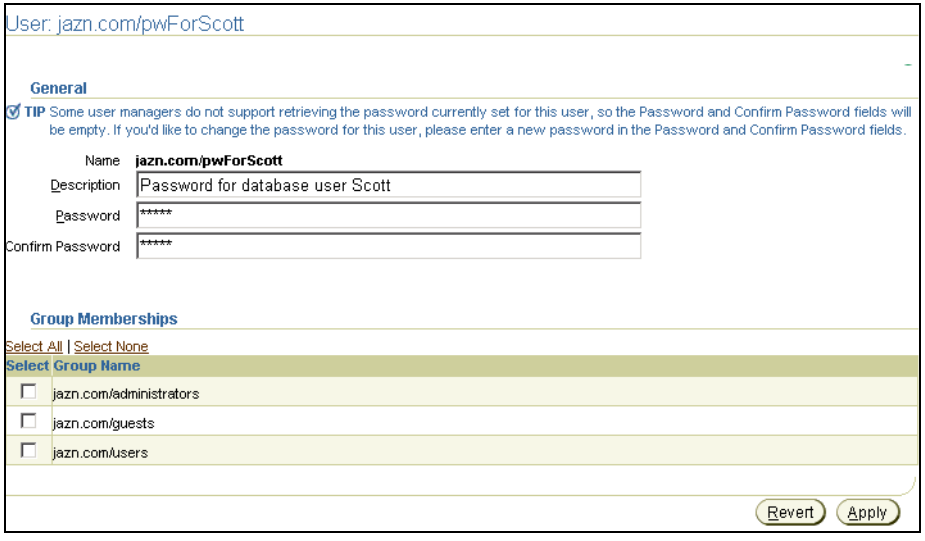
2. 該当するテキスト・ボックスに説明またはパスワードを入力します。（入力ミスを回避するために、パスワードを 2 回入力する必要があります。）
3. 「**適用**」をクリックします。

グループへのユーザーの割当て

注意： Enterprise Manager で管理されるのは、XML ベースのロールとユーザーのみです。LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

1. 「セキュリティ」ページ（8-9 ページの図 8-10 を参照）で、「**ユーザー**」リストからユーザーを選択します。「**ユーザー**」画面が表示されます。

図 8-15「ユーザー」画面



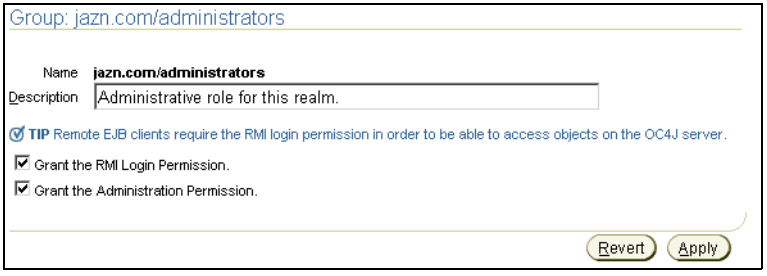
2. 選択したユーザーをグループに追加するには、そのグループのチェックボックスをオンにします。
3. 「**適用**」をクリックします。

グループに対する権限の付与

注意： Enterprise Manager で管理されるのは、XML ベースのロールとユーザーのみです。LDAP ベースのユーザーとロールを管理するには、Delegated Administration Service（DAS）を使用します。詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

1. 「セキュリティ」ページ（8-9 ページの図 8-10 を参照）で、「**グループ**」リストからグループを選択します。「**グループ**」画面が表示されます。

図 8-16「グループ」画面



2. グループ RMI または Administration に権限を付与するには、該当するチェックボックスをオンにします。
3. 「**適用**」をクリックします。

第 II 部

その他のテクノロジ

第 II 部では、アプリケーションのセキュリティに影響する JAAS 以外の Java テクノロジについて説明します。

第 II 部は、次の章で構成されています。

9

Java 2 セキュリティ

この章では、Java 2 セキュリティ機能について説明します。この章には次の項目が含まれます。

概要

Java 2 セキュリティ・モデルは JAAS Provider の基盤です。Java 2 セキュリティ・モデルを使用すると、すべての制限レベルでセキュリティを構成できます。これにより、開発者や管理者にとっては、エンタープライズ・アプレット、コンポーネント、サーブレットおよびアプリケーションのセキュリティの様々な側面の制御が向上します。

> **関連項目：** Java 2 セキュリティのチュートリアルは、`http://java.sun.com/docs/books/tutorial/security1.2/index.html` を参照してください。Java 2 セキュリティの詳細は、`http://java.sun.com/security` を参照してください。

パーミッション

パーミッションは Java 2 セキュリティ・モデルの基礎です。すべての Java クラスには（ローカルで実行するかリモートでダウンロードするかに関係なく）、そのクラスに使用可能なパーミッション・セットを定義するように構成されたセキュリティ・ポリシーが適用されます。各パーミッションは特定のリソースへの特定のアクセス権を表します。表 9-1 に、Java パーミッション・インスタンスを構成する要素を示します。

表 9-1 Java パーミッション・インスタンスの要素

要素	説明	例
クラス名	パーミッション・クラス	`java.io.FilePermission`
ターゲット	このパーミッションが適用されるターゲット名（リソース）	ディレクトリ `/home/*`
アクション	このターゲットに関連したアクション	ディレクトリ `/home/*` に対する読取り、書込みおよび実行権限

保護ドメイン

各 Java クラスは、ロード時に保護ドメインに関連付けられます。保護ドメインは、すべての制限レベル（リソースに対する完全制限からすべてのリソースへの完全アクセス権まで）について構成できます。各保護ドメインには、Java 仮想マシン（JVM）の起動時に、構成済のセキュリティ・ポリシーに基づいてパーミッション・グループが割り当てられます。

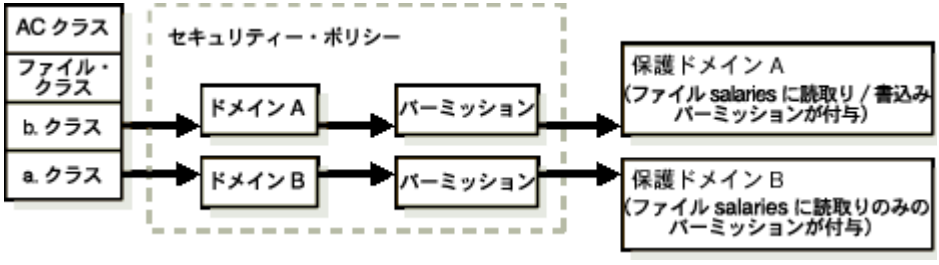
実行時には、スタック・イントロスペクションにより認可チェックが実行されます。この認可チェックでは、ランタイム・スタックが検討され、スタック上のクラスに関連した保護ドメインに基づいてパーミッションがチェックされます。これは、通常、次のどちらかのコールによりトリガーされます。

- `SecurityManager.checkPermission()`
- `AccessController.checkPermission()`

有効なパーミッション・セットは、セキュリティ・チェック時に、保護ドメインに割り当てられているすべてのパーミッション・セットの共通部分として定義されます。

図 9-1 に、実行時の認可チェックの基本モデルを示します。

図 9-1 Java 2 セキュリティ・モデル



関連項目：

- 第 4 章「JAAS Provider 管理タスク」
- 次の URL にある Sun 社の Java ドキュメント
- `http://java.sun.com/security/`

JAAS Provider のパーミッション・クラス

表 9-2 に、JAAS Provider に用意されているパーミッション・クラスを示します。これらのクラスを使用すると、アプリケーションでリソースへのアクセスを制御できます。この表に示すクラスの詳細は、JAAS Provider の Javadoc を参照してください。

表 9-2 JAAS Provider のパーミッション・クラス

パーミッション	パッケージ部分	説明
`AdminPermission`	`oracle.security.jazn.policy`	パーミッションを管理する（つまり、他のユーザーのパーミッション割当てを付与または取り消す）ための権利を表します。
`RoleAdminPermission`	`oracle.security.jazn.policy`	このパーミッションの受領者には、ターゲット・ロールをさらに付与または取り消すための権利が付与されます。
`JAZNPermission`	`oracle.security.jazn`	認可パーミッションの場合。`JAZNPermission` には、名前（ターゲット名）が含まれますが、アクション・リストはありません。指定のパーミッションがある場合とない場合があります。
`RealmPermission`	`oracle.security.jazn.realm`	レルムに対するパーミッションのアクション（`createRealm`、`dropRealm` など）を表します。`RealmPermission` は `java.security.Permission` からの拡張であり、通常の Java のパーミッションと同様に使用されます。

Java 2 ポリシー・ファイルの作成

Java 2 ポリシー・ファイルは、実行するトラステッド・コードまたはアプリケーションにパーミッションを付与します。これにより、コードまたはアプリケーションは特定のアクセス権を必要とする JAAS または JDK API に対する Oracle サポートにアクセスできます。

事前構成済の Java 2 ポリシー（java2.policy）は、*ORACLE_HOME*/j2ee/home/config に用意されています。

この Java 2 ポリシー・ファイルを、トラステッド・コードまたはアプリケーションにパーミッションを付与するように変更する必要があります。

たとえば、java2.policy ファイルの次のセクションでは、トラステッド jazn.jar に java.security.AllPermission を付与しています。

```
/* grant the JAZN library AllPermission */
grant codebase "file:${oracle.home}/j2ee/home/jazn.jar" {
    permission java.security.AllPermission;
};
```

次の例では、$ORACLE_HOME/appdemo ディレクトリで実行中のすべてのアプリケーションに特定のパーミッションを付与しています。

```
/* Assuming you are running your application demo in $ORACLE_HOME/appdemo/, */
/* Grant JAZN permissions to the demo to run JAZN APIs*/
grant codebase "file:/${oracle.ons.oraclehome}/appdemo/-" {
   permission oracle.security.jazn.JAZNPermission "getPolicy";
   permission oracle.security.jazn.JAZNPermission "getRealmManager";
   permission oracle.security.jazn.policy.AdminPermission
"oracle.security.jazn.realm.RealmPermission$*$createRealm,dropRealm,
  createRole, dropRole,modifyRealmMetaData";
```

Java 2 セキュリティ・マネージャ

JAAS Provider でパーミッションがチェックされるのは、SecurityManager がインストールされている場合のみです。SecurityManager を指定するには、次の 2 つの方法があります。

- System.setSecurityManager() をコールする方法
- OC4J の起動時にシステム・プロパティ java.security.manager を設定する方法

どちらかのメカニズムを使用して、デフォルトの SecurityManager またはカスタムの UserManager をインストールできます。

> **注意：** システム・プロパティを設定するには、Enterprise Manager で -D コマンドライン・オプションを使用します。詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』の「高度な構成」の章を参照してください。

デフォルトの SecurityManager により特定のクラスに付与されるパーミッションは、ポリシー・ファイルを読み取ることで決定されます。デフォルトのポリシー・ファイルは、J2EE に付属しています。ポリシー・ファイルは、次のようにシステム・プロパティ java.security.policy を使用して明示的に指定できます。

```
-Djava.security.policy=policyfilepath
```

Oracle Application Server 10*g* のインストール中には、デフォルトで SecurityManager なしで OC4J インスタンスが実行されます。SecurityManager をインストールするように選択した場合は、標準の OC4J 機能を妨げないセキュリティ・マネージャを指定する必要があります。サンプル・ポリシー・ファイルは、*ORACLE_HOME*/j2ee/home/config/java2.policy にあります。このサンプル・ファイルでは、ほとんどの OC4J JAR に「AllPermission」が付与されます。このファイルの標準的なブロックを次に示します。

```
grant codebase "file:${oracle.home}/j2ee/home/ejb.jar" {
   permission java.security.AllPermission;
};
```

ORACLE_HOME の位置を指定するために「${oracle.home}」が使用されていることに注意してください。oracle.home は、次のシステム・プロパティを指定することで設定できます。

```
-Doracle.home=oraclehomepathname
```

パスは java.io.File のルールに従って標準化されます。UNIX 上では、パスにシンボリック・リンクを使用できません。標準パスを指定しないと、デフォルトの SecurityManager ではポリシー・ファイル内の codebase 指定が適用されません。

アプリケーション・コードと OC4J により生成されるクラスには、パーミッションの追加付与が必要になることがあります。サンプル java2.policy ファイルの末尾には、Java 2 セキュリティが有効化されている状態でデモを実行するために必要なブロックが含まれています。必要なパーミッションはアプリケーションの詳細に応じて異なり、必要なコードベースはインストールの詳細に応じて異なります。

PrintingSecurityManager を使用した Java 2 ポリシーのデバッグ

付与する必要のあるパーミッションの決定を簡略化するために、Oracle にはカスタム SecurityManager である PrintingSecurityManager が用意されています。このセキュリティ・マネージャでは、SecurityException は発生しません。かわりに、通常はセキュリティ例外が発生する場合に、PrintingSecurityManager ではデフォルトの SecurityManager で発生する例外を示すメッセージが出力されます。アプリケーションに付与する必要のあるパーミッションを決定するには、PrintingSecurityManager を使用して OC4J を起動し、対象アプリケーションを実行します。

デフォルトの SecurityManager を使用してアプリケーションを実行すると、最初の SecurityException が発生した時点でアプリケーションが終了します。最初の問題を訂正した後に、アプリケーションを何度か実行して例外の原因をすべて追跡する必要があります。PrintingSecurityManager を使用すると、すべての SecurityExceptions を 1 つにまとめた大きいリストを作成できます。

PrintingSecurityManager をインストールするには、次のように、OC4J を起動するコマンドラインで指定する必要があります。

```
-Djava.security.manager=oracle.oc4j.security.PrintingSecurityManager
-Djava.security.policy=yourpolicypath
```

PrintingSecurityManager を含む oc4j.jar には、十分なパーミッション（通常は java.security.AllPermission）を付与する必要があります。十分なパーミッションを付与しないと、実行する特定の操作によりパーミッション・チェックがトリガーされ、そのチェックが失敗するため、PrintingSecurityManager が無限ループに陥ります。これにより、最終的には StackOverflowError となります。

PrintingSecurityManager では、System.out に 2 種類のメッセージが出力されます。最初のメッセージを次に示します。

```
SecurityManager would throw java.security.AccessControlException: access denied
  (java.io.FilePermission path read)
```

path は、クラスの実際のパスです。このメッセージは、デフォルトの SecurityManager で指定の例外が発生したことを意味します。PrintingSecurityManager は、この種のメッセージを生成するたびに、指定の Permission を付与する必要のあるコードソースの判別を試みます。PrintingSecurityManager が必要なパーミッションの判別に成功すると、次のような診断メッセージが出力されます。

```
(file:path/yourclass) needs (java.io.FilePermission path read)
```

PrintingSecurityManager は、デフォルトの SecurityManager のアクションを完全には再生できません。これは、このメッセージが不正確な推論である場合があることを意味します。

10

パスワード管理

この章では、XML ファイル内のパスワード管理について説明します。この章には次の項目が含まれます。

概要

多数の OC4J コンポーネントには、認証用にパスワードが必要です。これらのパスワードをデプロイメント・ファイルと構成ファイルに埋め込む方法は、特にファイルのパーミッションによりどのユーザーでも読取り可能な場合には、セキュリティ上のリスクを伴います。この問題を回避するために、OC4J には次の 2 つの解決策が用意されています。

- パスワードの不明瞭化。クリアテキスト・ファイルに格納されているパスワードを暗号化バージョンで置換します。詳細は、「jazn-data.xml および jazn.xml 内のパスワードの不明瞭化」を参照してください。
- パスワードの間接化。クリアテキストのパスワードを、他の場所で検索するために必要な情報で置換します。詳細は、「間接パスワードの作成」を参照してください。

jazn-data.xml および jazn.xml 内のパスワードの不明瞭化

JAAS 構成ファイル `jazn.xml` および `jazn-data.xml` には、JAAS 認可用のユーザー名とパスワードが含まれています。これらのファイルを保護するために、OC4J ではパスワードの不明瞭化が使用されます。

jazn.xml または `jazn-data.xml` ファイルを更新するたびに、OC4J によりファイルが読み取られ、すべてのパスワードの不明瞭化（暗号化）バージョンで書き換えられます。他のすべての OC4J 構成ファイル内では、次の「間接パスワードの作成」で説明するようにパスワードの間接化を使用してパスワードのクリアテキストの公開を回避できます。

JAAS Provider は、`orion-application.xml` 内のパスワードを不明瞭化しません。これは、パスワードを `orion-application.xml` に格納されている `<jazn>` 要素に埋め込まないことを意味します。

LDAP ベース・プロバイダを使用する場合は、アプリケーションを定義する `<jazn>` 要素を含む別個の `jazn.xml` ファイルを作成する必要があります。このファイルには、ユーザーやグループのデータは含まれません。この `<jazn>` 要素を次に示します。

```
<jazn provider="LDAP" location="yourlocation">
    <property name="ldap.name" value="cn=orcladmin" />
    <property name="ldap.password" value="!welcome1" />
</jazn>
```

次のように、`orion-application.xml` 内で `config` 属性を使用して、`jazn.xml` ファイルを指す `<jazn>` 要素を作成します。

```
<jazn config="./jazn.xml" />
```

JAZN は、この別個の `jazn.xml` ファイルを初めて読み取るときに、このファイルに格納されているパスワードを自動的に不明瞭化します。

手動による jazn-data.xml の編集

必要な場合は、jazn-data.xml ファイルをテキスト・エディタで直接編集できます。次回に OC4J が jazn-data.xml を読み取ると、このファイルは不明瞭化され判読できないすべてのパスワードで書き換えられます。

<credentials> 要素の clear 属性を true に設定すると、jazn-data.xml ファイル内でクリア（判読可能）なパスワードを使用できます。

```
<credentials clear="true">welcome</credentials>
<credentials>!welcome</credentials>
```

間接パスワードの作成

次の OC4J XML 構成ファイルとデプロイメント・ファイルでは、1 つ以上のエンティティでパスワードの間接化がサポートされます。

- data-sources.xml: <data-source> 要素の password 属性
- ra.xml: <res-password> 要素
- rmi.xml: <cluster> 要素の password 属性
- application.xml: <resource-provider> および <commit-coordinator> 要素の password 属性
- jms.xml: <password> 要素
- internal-settings.xml: <sep-property> 要素、属性 name="keystore-password" および name="truststore-password"

これらのパスワードのいずれかを間接化するには、リテラルのパスワード文字列を、「->」に続くユーザー名、またはスラッシュ（/）で区切ったレルムとユーザー名を含む文字列で置換します。

注意： リテラル（非間接）パスワードを文字列「->」で始めるには、パスワードの前に「->!」を付けます。たとえば、直接パスワード「->silly」は「->!->silly」と表記します。

間接パスワードの例

- <data-source password="->Scott">: JaznUserManager で Scott を検索し、そこに格納されているパスワードを使用します。
- <res-password="->customers/Scott">: JaznUserManager を使用して customers レルム内で Scott を検索し、そこに格納されているパスワードを使用します。
- <cluster password="martha">: リテラル文字列「martha」がパスワードであり、間接パスワードではありません。

orion-application.xml での UserManager の指定

<password-manager> 要素には、グローバル・アプリケーションで間接パスワードの検索に使用される UserManager を指定します。（10-3 ページの「間接パスワードの作成」を参照してください。）この要素を省略すると、間接パスワードの認証と認可にはグローバル・アプリケーションの UserManager が使用されます。<password-manager> 要素内の <jazn> 要素には、トップレベルの <jazn> 要素とは異なる値を指定できます。

たとえば、通常の UserManager に LDAP ベースの UserManager を使用できますが、間接パスワードの認証には XML ベースの UserManager を使用します。LDAP で間接パスワードを使用するには、この方法を使用する必要があります。

詳細は、3-23 ページの「orion-application.xml での UserManager の指定」を参照してください。

注意： 交換可能な UserManagers をパスワード・マネージャとして使用できます。ただし、XMLUserManager をパスワード・マネージャとして使用すると、principals.xml のパスワードは不明瞭化されません。

11

クライアント接続用 Oracle HTTPS

この章では、クライアントの HTTP 接続に SSL 機能を提供する HTTPS の、Oracle Application Server Containers for J2EE（Oracle Application Server Containers for J2EE）実装について説明します。この章には、次の項目が含まれます。

概要

この章では、Secure Sockets Layer プロトコルを使用して、ネットワーク・アプリケーション間でセキュアに通信する方法について説明します。最初に SSL の基本概念、次に Oracle HTTPS と JSSE の使用方法について説明します。

注意：

- クライアントと Oracle HTTP Server 間のセキュアな通信は、Oracle HTTP Server と OC4J 間のセキュアな通信から独立しています。（Oracle HTTP Server と OC4J 間で使用されるセキュアな AJP プロトコルは、エンド・ユーザーには表示できないことにも注意してください。）この項では、OC4J とクライアント間のセキュアな通信についてのみ説明します。
- OC4J スタンドアロンでは、クライアントと OC4J 間で直接行われる SSL 通信が HTTPS を使用してサポートされます。詳細は、11-23 ページの「SSL 用の OC4J スタンドアロンの構成」を参照してください。

SSL の鍵と証明書の概要

企業や個人など、2 つのエンティティ間で行われる SSL 通信の場合、サーバーには公開鍵とそれに関連付けられた秘密鍵があります。それぞれの鍵は番号で、エンティティの秘密鍵はそのエンティティが機密として保管し、エンティティの公開鍵はセキュアな通信を必要とする第三者に公開されます。交換されるデータのセキュリティは、秘密鍵の機密を保つことと複雑な暗号化アルゴリズムにより保証されます。この方式は、データの暗号化と復号化に異なる鍵が使用されるため、非対称型暗号化と呼ばれます。

非対称型暗号化は複雑なため、パフォーマンスに負荷がかかります。それに比べて大幅に高速な方式が対称型暗号化で、この方式ではデータの暗号化と復号化に同じ鍵が使用されます。ただし、対称型暗号化には、送信側と受信側の両方が同じ鍵を知っており、その鍵のやりとりを第三者に不正傍受されると通信がセキュアでなくなるという弱点があります。

SSL では、通信に非対称型暗号化と対称型暗号化の両方が使用されます。非対称鍵（PKI 公開鍵）を使用して対称暗号鍵（バルク暗号鍵）がエンコードされ、バルク暗号鍵を使用して以降の通信が暗号化されます。両者がバルク暗号鍵に同意した後は、セキュリティと信頼性を損なわずに高速な通信が可能になります。

SSL セッションの折衝時には、次のステップが発生します。

1. サーバーがクライアントに公開鍵を送信します。
2. クライアントが、指定された暗号スイートを使用してバルク暗号鍵（通常は 128 ビットの RC4 鍵）を作成します。
3. クライアントがサーバーの公開鍵を使用してバルク鍵を暗号化し、暗号化したバルク鍵をサーバーに送信します。

4. サーバーが、その秘密鍵を使用してバルク暗号鍵を復号化します。

この一連の操作は、鍵交換と呼ばれます。鍵交換が発生した後、クライアントとサーバーはバルク暗号鍵を使用して、交換するすべてのデータを暗号化します。

注意： まれに、クライアントが独自の秘密鍵と公開鍵を使用する場合もあります。

SSL では、サーバーの公開鍵は X.509 証明書と呼ばれるデータ構造を使用してクライアントに送信されます。この証明書は認証局（CA）により作成され、公開鍵、証明書の所有者情報および必要な場合は所有者のなんらかのデジタル権利が含まれています。証明書には、それを作成した CA が自身のデジタル証明の公開鍵を使用してデジタル形式で署名します。

SSL では、CA の署名が受信側プロセスによりチェックされ、CA 署名の承認済リストに含まれていることが確認されます。このチェックは、証明連鎖の分析により実行される場合があります。この処理が発生するのは、受信側プロセスの承認済リストに署名した CA の公開鍵が含まれていない場合です。その場合、受信側プロセスは、CA の証明書の署名者が承認済リストに含まれているかどうか、署名者の署名者かどうかなどがチェックされます。この証明書、証明書の署名者、証明書の署名者の署名者という連鎖は、証明連鎖と呼ばれます。連鎖に含まれる最上位の証明書（オリジナルの署名者）は、証明連鎖のルート証明書と呼ばれます。

ルート証明書は、通常は受信側プロセスの承認済リストに含まれています。承認済リストに含まれている証明書は、トラスト・ポイントまたは信頼できる証明書と呼ばれます。ルート証明書には、CA が署名するか、自己署名できます。自己署名は、ルート証明書を検証するデジタル署名が、上位 CA の秘密鍵ではなく、証明書に含まれる公開鍵に対応する秘密鍵を介して暗号化されることを意味します。

証明書は、公開鍵および関連する署名のコンテナとして機能します。単一の証明書ファイルには、連鎖全体の範囲内で 1 つ以上の証明書を含めることができます。通常、秘密鍵は意図しない公開を防ぐために別個に保管されますが、アプリケーション間での移植性を考慮して証明書ファイルの別個のセクションに組み込むことができます。

キーストアは、すべてのトラステッド・ユーザーの証明書など、プログラムで使用される証明書の格納に使用されます。OC4J などのエンティティは、そのキーストアを介して第三者の認証や、第三者に対する自己認証を行うことができます。Oracle HTTP Server には、これと同じ用途を持つ Wallet と呼ばれる機能があります。Sun 社の SSL 実装にはトラストストアという表記が導入されています。これは、クライアントが SSL ハンドシェイク中に暗黙的に受け入れる、信頼できる認証局が含まれたキーストア・ファイルです。

Java では、キーストアは `java.security.KeyStore` インスタンスで、Sun 社の JDK に付属の `keytool` ユーティリティを使用して作成および操作できます。このオブジェクトの基礎となっているのは、物理的にはファイルです。`keytool` の詳細は、次のサイトを参照してください。

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

OC4J および Oracle HTTP Server を使用した鍵と証明書の作成

OC4J で SSL 通信に鍵と証明書を使用する手順は、次のとおりです。ここで説明するのはサーバー・レベルの手順で、通常はセキュアな通信を必要とするアプリケーションのデプロイメント前、通常は OracleAS インスタンスの初期設定時に実行します。

1. `keytool` を使用して、秘密鍵、公開鍵および未署名の証明書を生成します。この情報は、新規または既存のキーストアに置くことができます。
2. 次のどちらかのアプローチを使用して証明書の署名を取得します。

   次の手順で独自の署名を生成できます。

   a. `keytool` を使用して証明書の自己署名を行います。この方法は、クライアントから独自の認証局として信頼される場合に適しています。

   または、認識されている認証局から次の手順で署名を取得できます。

   a. 手順 1 からの証明書を使用し、`keytool` を使用して、認証局に証明書への署名を要求する証明書リクエストを生成します。

   b. 証明書リクエストを認証局に対して発行します。

   c. 認証局から署名を受け取り、`keytool` を使用してキーストアにインポートします。キーストアでは、署名が関連する証明書と照合されます。

注意： Oracle Application Server には、OracleAS Certificate Authority（OCA）が組み込まれています。このため、顧客は自身とそのユーザーのために証明書を作成して発行できますが、これらの証明書は事前に手配しなければ顧客の組織以外では認識されません。OCA の詳細は、『Oracle Application Server 10*g* セキュリティ・ガイド』を参照してください。

署名を要求して受け取るプロセスは、使用する認証局に応じて異なります。このプロセスは OracleAS のスコープおよび制御の対象外のため、OracleAS マニュアルでは説明しません。詳細は、任意の認証局の Web サイトにアクセスしてください。（ブラウザには、信頼できる認証局のリストがあります。）たとえば、VeriSign 社と Thawte 社の Web アドレスは次のとおりです。

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

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

OC4J と Oracle HTTP Server 間の SSL 通信の場合は、Oracle HTTP Server に関する前述の手順も実行しますが、キーストアと `keytool` ユーティリティのかわりに Wallet と Oracle Wallet Manager を使用します。Wallet と Oracle Wallet Manager の詳細は、『Oracle Application Server 10*g* セキュリティ・ガイド』を参照してください。

また、必要に応じて次の手順を実行します。

OC4J の証明書に、Oracle HTTP Server で信頼されていないエンティティが署名している場合の手順は、次のとおりです。

3. OC4J で `keytool` を使用して OC4J の証明書をエクスポートします。これにより、Oracle HTTP Server からアクセス可能なファイルに証明書が置かれます。
4. Oracle HTTP Server で、Oracle Wallet Manager を使用して OC4J の証明書をインポートします。

Oracle HTTP Server の証明書に OC4J が信頼していないエンティティが署名しており、OC4J がクライアント認証を必要とする操作モードになっている場合（11-7 ページの「クライアント認証の要求」を参照）の手順は、次のとおりです。

5. Oracle HTTP Server で、Oracle Wallet Manager を使用して Oracle HTTP Server の証明書をエクスポートします。これにより、OC4J からアクセス可能なファイルに証明書が置かれます。
6. OC4J で `keytool` を使用して Oracle HTTP Server の証明書をインポートします。

Oracle HTTP Server と OC4J 間で SSL を介して通信中には、両者間の通信チャネル上のデータがすべて暗号化されます。次の手順が実行されます。

1. 暗号化されたチャネルの確立中に、OC4J の証明連鎖が Oracle HTTP Server に対して認証されます。
2. OC4J がクライアント認証モードになっている場合は、必要に応じて Oracle HTTP Server が OC4J に対して認証されます。この処理は、暗号化されたチャネルの確立時にも発生します。
3. バルク暗号鍵が PKI 公開鍵を使用してセキュアに交換され、チャネル上の以降の通信を暗号化するために使用されます。

例 : SSL 証明書の作成と独自の署名の生成

この例は前述のステップ 2 に対応しており、このモードでは証明書の自己署名用に keytool を使用して独自の署名を生成します。

最初に、keytool コマンドを使用し、RSA の秘密鍵 / 公開鍵のペアを指定してキーストアを作成します。この例（% はシステム・プロンプト）では、RSA 鍵ペア・アルゴリズムを使用し、パスワードに 123456、有効期間に 21 日を指定して、ファイル mykeystore に常駐するキーストアを生成しています。

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

各項目の意味は次のとおりです。

- keystore オプションには、鍵が格納されるファイルの名前を指定します。
- storepass オプションでは、キーストアを保護するためのパスワードを設定します。
- validity オプションでは、証明書の有効期間を表す日数を設定します。

keytool により、次のように詳細情報を求めるプロンプトが表示されます。

```
What is your first and last name?
  [Unknown]: Test User
What is the name of your organizational unit?
  [Unknown]:  Support
What is the name of your organization?
  [Unknown]:  Oracle
What is the name of your City or Locality?
  [Unknown]:  Redwood Shores
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Test User, OU=Support, O=Oracle, L=Reading, ST=Berkshire, C=GB> correct?
  [no]:  yes

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

注意 : 2 文字の国コードを判断するには、次の URL にある ISO 国コード・リストを使用してください。

```
http://www.bcpl.net/~jspath/isocodes.html
```

---

mykeystore ファイルは、現行ディレクトリに作成されます。鍵のデフォルトの別名は mykey です。

クライアント認証の要求

OC4J では、クライアント認証モードがサポートされます。このモードでは、サーバーはクライアントと通信する前に、クライアントからの認証を明示的に要求します。OracleAS 環境では、Oracle HTTP Server は OC4J のクライアントとして機能します。

クライアント認証の場合、Oracle HTTP Server は独自の証明書を持ち、その証明書とルート証明書で終わる証明連鎖を送信して自己認証を行う必要があります。OC4J は、クライアントに至る信頼の連鎖を確立するときに、指定のリストからルート証明書のみを受け入れるように構成できます。

OC4J が信頼する証明書は、トラスト・ポイントと呼ばれます。Oracle HTTP Server からの証明連鎖では、トラスト・ポイントは OC4J でキーストア内の証明書との一致として検出される最初の証明書です。この信頼関係は、次の 3 つの方法のいずれかで確立されます。

- クライアント証明書がキーストアにあること。
- Oracle HTTP Server からの証明連鎖に含まれる中間 CA の証明書の 1 つがキーストアにあること。
- Oracle HTTP Server からの証明連鎖に含まれるルート CA 証明書がキーストアにあること。

OC4J では、捏造された証明書を防ぐために、トラスト・ポイントを含む証明連鎖全体が有効かどうかが検証されます。

`needs-client-auth` 属性を使用してクライアント認証を要求する場合の手順は、次のとおりです。この属性の構成方法は、11-22 ページの「OC4J の SSL 用構成手順」を参照してください。

1. Oracle HTTP Server からの連鎖のうちトラスト・ポイントにする証明書を決定します。このトラスト・ポイントを使用して証明書の発行を制御できること、または認証局を発行者として信頼できることを確認します。
2. クライアント証明書の認証用に、中間またはルート証明書をトラスト・ポイントとしてサーバーのキーストアにインポートします。

   注意： OC4J が特定のトラスト・ポイントを受け入れないようにする場合は、これらのトラスト・ポイントがキーストアにないことを確認してください。

3. クライアント証明書の作成手順を実行します（11-4 ページの「OC4J および Oracle HTTP Server を使用した鍵と証明書の作成」を参照してください）。クライアント証明書には、サーバーにインストールされている中間またはルートの証明書が含まれます。他の認証局を信頼する場合は、その認証局から証明書を取得します。
4. 証明書を Oracle HTTP Server 上のファイルに保存します。
5. 安全な AJP 接続を開始できるように、Oracle HTTP Server 用の証明書を提供します。

クライアントと OC4J 間のセキュアな通信中には、次の機能が実行されます。

- 両者間のリンク（すべての通信）が暗号化されます。
- OC4J がクライアントに対して認証されます。秘密鍵がセキュアに交換され、リンクの暗号化に使用されます。
- OC4J がクライアント認証モードになっている場合は、必要に応じてクライアントが OC4J に対して認証されます。

関連項目：

- Oracle Wallet Manager、PKI およびセキュリティの基礎の詳細は、『Oracle Application Server 10*g* セキュリティ・ガイド』および『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。
- JSSE と `java.net` パッケージの詳細は、`http://www.java.sun.com` のドキュメントを参照してください。

Oracle HTTPS とクライアント

HTTPS は、クライアントとサーバー間の相互作用の保護に不可欠です。多数のサーバー・アプリケーションの場合、HTTPS は Web サーバーにより処理されます。ただし、他の Web サーバーへの接続を開始するサーブレットなど、クライアントとして機能するアプリケーションには、サーバーとの間で情報をセキュアにやりとりするために独自の HTTPS 実装が必要です。HTTP パッケージ `HTTPClient` または Sun 社の `java.net` パッケージに慣れている Java アプリケーション開発者は、Oracle HTTPS を使用してクライアントとサーバーとの相互作用を容易に保護できます。

Oracle HTTPS では、完全な HTTP クライアント・ライブラリを提供する、`HTTPClient` パッケージの `HTTPConnection` クラスが拡張されます。クライアントの HTTPS 接続をサポートするために、OracleSSL クラス `OracleSSLCredential` を使用する `HTTPConnection` クラスには複数のメソッドが追加されています。

注意： Oracle `HTTPClient` では、2 つの異なる SSL 実装がサポートされます。その一方は Java Secure Socket Extension（JSSE）、他方は OracleSSL です。このマニュアルでは、この 2 つの実装について個別に説明します。

HTTPConnection クラス

HTTPConnection クラスは、SSL を使用するかどうかを問わず、HTTP を使用する新規接続の作成に使用されます。このクラスには、PKI（公開鍵インフラストラクチャ）デジタル証明と Wallet のサポートを提供するために、11-16 ページの「Oracle HTTPS の例」に示すメソッドが追加されています。

関連項目： HTTPClient の Javadoc を参照してください。

OracleSSLCredential クラス（OracleSSL のみ）

セキュリティ資格証明は、サーバーとクライアントの相互認証に使用されます。Oracle HTTPS では、Oracle Java SSL パッケージ OracleSSLCredential を使用して、base64 または DER エンコード証明書からユーザー証明書とトラスト・ポイントがロードされます。（DER は X.690 ASN.1 規格の一部で、Distinguished Encoding Rules の略語です。）

Oracle Java SSL 用の API では、接続が確立される前にセキュリティ資格証明が HTTP 接続に渡される必要があります。OracleSSLCredential クラスは、これらのセキュリティ資格証明の格納に使用されます。通常、Oracle Wallet Manager により生成される Wallet は、OracleSSLCredential オブジェクトの移入に使用されます。または、OracleSSLCredential クラスの API を使用して個別証明書を追加できます。資格証明が完了すると、setCredentials メソッドとの接続に使用されます。

Oracle HTTPS の機能概要

Oracle HTTPS では、クライアントとサーバー間の HTTP 1.0 および HTTP 1.1 接続がサポートされます。SSL 機能を提供するために、このパッケージの HTTPConnection クラスに新規メソッドが追加されています。これらのメソッドは、Cipher Suite の選択、Oracle Wallet Manager でのセキュリティ資格証明管理、セキュリティ対応アプリケーションおよび後述するその他機能をサポートするために、Oracle Java SSL と併用されます。Oracle HTTPS では Oracle Java SSL クラス OracleSSLCredential が使用され、HTTPClient パッケージの HTTPConnection クラスが拡張されます。HTTPClient では、OracleSSL および JSSE という 2 つの SSL 実装がサポートされます。

Oracle HTTPS では、HTTPClient パッケージに組み込まれている機能に加えて次の機能がサポートされます。

- 複数の暗号化アルゴリズム
- Oracle Wallet Manager による証明書と鍵の管理
- java.net.URL フレームワークの限定サポート
- OracleSSL と JSSE の SSL 実装

また、Oracle HTTPS では、HTTPClient パッケージを使用して次の機能がサポートされます。

- プロキシを介した HTTP トンネリング
- HTTP プロキシ認証

以降の各項では、Oracle HTTPS 機能を詳細に説明します。

SSL Cipher Suite

データが SSL 接続を介して流れる前に、接続の両端がデータ送信に使用する共通アルゴリズムを折衝する必要があります。混在型のセキュリティ機能を提供するために結合されているアルゴリズムのセットは、Cipher Suite と呼ばれます。SSL 接続の参加者は、特定の Cipher Suite を選択すると適切な通信レベルを確立できます。

HTTPClient で 2 つの SSL 実装がサポートされ、それぞれの実装で異なる Cipher Suite がサポートされます。ここでは、各 Cipher Suite について説明します。

Cipher Suite の選択

通常は、次の優先順位に従う必要があります。

- RSA では多数のセキュリティ攻撃が無効化されるため、Diffie-Hellman よりも RSA を優先します。
- 3DES と RC4 128 には強力な鍵があるため、他の暗号方式よりも 3DES または RC4 128 を優先します。
- SHA1 の方が強力なダイジェストが生成されるため、MD5 よりも SHA1 ダイジェストを優先します。

OracleSSL でサポートされる SSL Cipher Suite

OracleSSL では、表 11-1 に示す Cipher Suite がサポートされます。NULL 暗号化の場合、SSL は認証とデータ整合性のためにのみ使用されることに注意してください。

表 11-1 OracleSSL でサポートされる Cipher Suite

Cipher Suite	認証	暗号化	ハッシュ関数（ダイジェスト）
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES EDE CBC	SHA1
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4 128	SHA1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4 128	MD5
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES CBC	SHA1
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA	RC4 40	MD5
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	RSA	DES40 CBC	SHA1
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH anon	3DES EDE CBC	SHA1
SSL_DH_anon_WITH_RC4_128_MD5	DH anon	RC4 128	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH anon	DES CBC	SHA1
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	DH anon	RC4 40	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH anon	DES40 CBC	SHA1
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA1
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5

JSSE でサポートされる SSL Cipher Suite

JSSE では、表 11-2 に示す Cipher Suite がサポートされます。NULL 暗号化の場合、SSL は認証とデータ整合性のためにのみ使用されることに注意してください。

表 11-2 JSSE でサポートされる Cipher Suite

Cipher Suite	認証	暗号化	ハッシュ関数（ダイジェスト）
SSL_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES EDE CBC	SHA1
SSL_RSA_WITH_RC4_128_SHA	RSA	RC4 128	SHA1
SSL_RSA_WITH_RC4_128_MD5	RSA	RC4 128	MD5
SSL_RSA_WITH_DES_CBC_SHA	RSA	DES CBC	SHA1
SSL_RSA_EXPORT_WITH_RC4_40_MD5	RSA	RC4 40	MD5

表 11-2 JSSE でサポートされる Cipher Suite（続き）

Cipher Suite	認証	暗号化	ハッシュ関数（ダイジェスト）
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA	DH anon	3DES EDE CBC	SHA1
SSL_DH_anon_WITH_RC4_128_MD5	DH anon	RC4 128	MD5
SSL_DH_anon_WITH_DES_CBC_SHA	DH anon	DES CBC	SHA1
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5	DH anon	RC4 40	MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA	DH anon	DES40 CBC	SHA1
SSL_RSA_WITH_NULL_SHA	RSA	NULL	SHA1
SSL_RSA_WITH_NULL_MD5	RSA	NULL	MD5
SSL_DHE_DSS_WITH_DES_CBC_SHA	DH	DES CBC	SHA1
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA	DH	3DES EDE CBC	SHA1
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA	DH	DES40 CBC	SHA1

確立された SSL 接続に関する情報へのアクセス

Oracle HTTPS の `getSSLSession` メソッドを使用すると、確立された SSL 接続に関する情報にアクセスできます。接続の確立後は、接続に使用された Cipher Suite、ピア証明連鎖および現行の接続に関するその他の情報を取得できます。

セキュリティ対応アプリケーションのサポート

Oracle HTTPS では、Oracle Java SSL を使用してセキュリティ対応アプリケーションのサポート機能が提供されます。セキュリティ対応アプリケーションでトラスト・ポイントが設定されていない場合は、Oracle Java SSL を使用すると、独自の検証を実行して、ピアから完全な証明連鎖が送信された場合にのみハンドシェイクを正常終了させることができます。アプリケーションでトラスト・ポイント・レベルまで認証する場合は、トラスト・ポイントの下の個々の証明書を認証する必要があります。

ハンドシェイクの完了後に、アプリケーションは SSL セッション情報を取得して、接続について追加の検証を実行する必要があります。

トラスト・ポイントのチェックを必要とするセキュリティ非対応アプリケーションでは、HTTPS のインフラストラクチャ内でトラスト・ポイントが設定されていることを確認する必要があります。

関連項目： Oracle Java SSL の詳細は、『Oracle Advanced Security 管理者ガイド』を参照してください。

java.net.URL フレームワークのサポート

HTTPClient パッケージは、HTTPClient.HttpUrlConnection クラスにより java.net.URL フレームワークの基本サポートを提供します。ただし、Oracle HTTPS の機能の多くはシステム・プロパティを介してのみサポートされます。

システム・プロパティを介してのみサポートされる機能は、次のとおりです。

- Cipher Suite 選択オプション
- 機密保護専用オプション
- サーバー認証オプション
- 相互認証オプション
- Oracle Wallet Manager によるセキュリティ資格証明管理

注意： java.net.URL フレームワークを使用する場合は、java.protocol.handler.pkgs システム・プロパティを次のように設定し、HTTPClient パッケージを JDK クライアントの置換えとして選択します。

```
java.protocol.handler.pkgs=HTTPClient
```

関連項目：

- JSSE を使用するようにクライアントを構成する方法は、11-14 ページの「デフォルトのシステム・プロパティの指定」を参照してください。
- java.net.URL フレームワークについては、次の URL にあるドキュメントを参照してください。

  http://java.sun.com

デフォルトのシステム・プロパティの指定

HTTPS の多数のユーザーは、一部のデフォルト・プロパティを非プログラムによる方法で指定することを望みます。そのための最善の方法は、`java.lang.System` クラスを介してアクセス可能な Java システム・プロパティを使用することです。`java.net.URL` フレームワークのユーザーがセキュリティ資格証明情報を設定するには、これらのプロパティを使用する必要があります。Oracle HTTPS では、次のプロパティが認識されます。

- javax.net.ssl.KeyStore
- javax.net.ssl.KeyStorePassword
- Oracle.ssl.defaultCipherSuites（OracleSSL のみ）

以降の各項では、これらのプロパティの設定方法について説明します。

javax.net.ssl.KeyStore

このプロパティは、Oracle Wallet Manager からエクスポートされるテキスト Wallet ファイルを指すように設定できます。このファイルには、特定の接続に使用される資格証明が含まれています。次に例を示します。

```
javax.net.ssl.KeyStore=/etc/ORACLE/WALLETS/Default/default.txt
```

`default.txt` は、資格証明を含むテキスト Wallet ファイル名です。

HTTPS 接続用に他の資格証明が設定されていない場合は、ハンドシェイクが最初に発生した時点で、このプロパティに指定したファイルが開きます。このファイルの読取り中にエラーが発生すると、接続は失敗して `IOException` が発生します。

このプロパティを設定しない場合、アプリケーションでは証明連鎖に信頼できる証明書が含まれているかどうかを検証する必要があります。ただし、Oracle SSL を使用する `HTTPClient` では、ユーザー証明書からルート CA に至るまで、証明連鎖に含まれる証明書すべてがサーバーから送信されたかどうかと、これらの証明書すべてに有効な署名が含まれているかどうかが検証されます。

javax.net.ssl.KeyStorePassword

このプロパティは、Wallet ファイルを開くために必要なパスワードに設定できます。次に例を示します。

```
javax.net.ssl.KeyStorePassword=welcome1
```

welcome1 は、Wallet ファイルを開くために必要なパスワードです。

システム・プロパティにパスワードを格納した場合に考えられるセキュリティ上のリスク

Wallet ファイルのパスワードを Java システム・プロパティとして格納すると、環境によってはセキュリティ上のリスクが生じる可能性があります。このリスクを回避するために、次のいずれかの代替策を使用します。

- アプリケーションに相互認証が不要な場合は、秘密鍵を含まないテキスト Wallet を使用します。これらの Wallet は、パスワードがなくても開くことができます。
- パスワードが必要な場合は、クリアテキスト・ファイルに格納しません。かわりに、`System.setProperty()` を使用して、`HTTPConnection` を起動する前にプロパティを動的にロードします。ハンドシェイクの完了後に、このプロパティを設定解除します。

Oracle.ssl.defaultCipherSuites（OracleSSL のみ）

このプロパティは、カンマ区切りの Cipher Suite リストに設定できます。次に例を示します。

```
Oracle.ssl.defaultCipherSuites=
            SSL_RSA_WITH_DES_CBC_SHA,\
            SSL_RSA_EXPORT_WITH_RC4_40_MD5,\
            SSL_RSA_WITH_RC4_128_MD5
```

このプロパティに設定する Cipher Suite は、新規 HTTPS 接続のデフォルト Cipher Suite として使用されます。

> **関連項目：** OracleSSL でサポートされる全 Cipher Suite のリストは、11-11 ページの表 11-1 を参照してください。

Oracle HTTPS の例

次のコードは、Oracle HTTPS、HTTPClient および OracleSSL を使用して Web サーバーに接続し、GET リクエストを送信して Web ページをフェッチする単純なプログラムです。このプログラムの完全なコードの後に、Oracle HTTPS を使用して安全な接続を設定する方法について説明します。

```
import HTTPClient.HTTPConnection;
import HTTPClient.HTTPResponse;
import oracle.security.ssl.OracleSSLCredential;
import java.io.IOException;

public class HTTPSConnectionExample
{
    public static void main(String[] args)
    {
        if(args.length < 4)
        {
            System.out.println(
            "Usage: java HTTPSConnectionTest [host] [port] " +
            "[wallet] [password]");
            System.exit(-1);
        }

        String hostname = args[0].toLowerCase();
        int port = Integer.decode(args[1]).intValue();
        String walletPath = args[2];
        String password = args[3];

        HTTPConnection httpsConnection = null;
        OracleSSLCredential credential = null;

        try
        {
            httpsConnection = new HTTPConnection("https", hostname, port);
        }
        catch(IOException e)
        {
            System.out.println("HTTPS Protocol not supported");
            System.exit(-1);
        }

        try
        {
            credential = new OracleSSLCredential();
            credential.setWallet(walletPath, password);
        }
```

```
catch(IOException e)
{
    System.out.println("Could not open wallet");
    System.exit(-1);
}
httpsConnection.setSSLCredential(credential);

try
{
    httpsConnection.connect();
}
catch (IOException e)
{
    System.out.println("Could not establish connection");
    e.printStackTrace();
    System.exit(-1);
}

javax.security.cert.X509Certificate[] peerCerts = null;
try
{
    peerCerts =
      (httpsConnection.getSSLSession()).getPeerCertificateChain();
}
catch(javax.net.ssl.SSLPeerUnverifiedException e)
{
    System.err.println("Unable to obtain peer credentials");
    System.exit(-1);
}

String peerCertDN =
  peerCerts[peerCerts.length -1].getSubjectDN().getName();
peerCertDN = peerCertDN.toLowerCase();
if(peerCertDN.lastIndexOf("cn="+hostname) == -1)
{
    System.out.println("Certificate for " + hostname + " is issued to "
      + peerCertDN);
    System.out.println("Aborting connection");
    System.exit(-1);
}

try
{
    HTTPResponse rsp = httpsConnection.Get("/");
    System.out.println("Server Response: ");
    System.out.println(rsp);
}
```

```
        catch(Exception e)
        {
            System.out.println("Exception occured during Get");
            e.printStackTrace();
            System.exit(-1);
        }
    }
}
```

OracleSSL での SSL 資格証明の初期化

このサンプル・プログラムは、Oracle Wallet Manager により作成された Wallet を使用して資格証明情報を設定します。最初に、次のコードを使用して資格証明が作成され、Wallet がロードされます。

```
credential = new OracleSSLCredential();
credential.setWallet(walletPath, password);
```

作成された資格証明は、次のコードを使用して HTTPSConnection に渡されます。

```
httpsConnection.setSSLCredential(credential);
```

これにより、Wallet に置かれた秘密鍵、ユーザー証明書およびトラスト・ポイントを接続に使用できます。

接続情報の検証

SSL では、サーバーから提示された証明連鎖が有効で、クライアントが信頼できる証明書が 1 つ以上含まれているかどうかが検証されますが、悪意のある第三者による偽装は防止されません。この問題に対処する HTTPS 規格は、HTTPS サーバーがそのホスト名に対して発行された証明書を持つように求めています。このため、クライアントは SSL 接続の確立後に、この検証を実行する必要があります。

このサンプル・プログラムでこの検証を実行するために、HTTPSConnectionExample は、次を使用してデータを送信せずにサーバーへの接続を確立しています。

```
httpsConnection.connect();
```

接続の確立後は、接続情報（この場合はサーバーの証明連鎖）が次を使用して取得されます。

```
peerCerts = (httpsConnection.getSSLSession()).getPeerCertificateChain();
```

最後に、次を使用してサーバー証明書の共通名が取得されます。

```
String peerCertDN = peerCerts[peerCerts.length -1].getSubjectDN().getName();
peerCertDN = peerCertDN.toLowerCase();
```

証明書名がサーバーへの接続に使用されたホスト名と異なる場合は、次を使用して接続が異常終了します。

```
if(peerCertDN.lastIndexOf("cn="+hostname) == -1)
{
    System.out.println("Certificate for " + hostname + " is issued to " +
        peerCertDN);
    System.out.println("Aborting connection");
        System.exit(-1);
}
```

HTTPS を使用したデータ送信

クライアントまたはサーバーからデータが送信される前に、接続情報を検証することが重要です。HTTPS でのデータ送信は、HTTP の場合と同様に実行されます。このサンプル・プログラムでは、次を使用してサーバーに対する GET リクエストが発行されます。

```
HTTPResponse rsp = httpsConnection.Get("/");
```

JSSE と HTTPClient の使用

OracleAS では、Java Secure Socket Extension（JSSE）を使用して HTTPS クライアント接続がサポートされます。クライアントでは、基礎となる SSL プロバイダとして JSSE を使用するように HTTPClient を構成できます。

注意：

- JSSE の SSL 実装はスレッド・セーフではありません。スレッド化アプリケーションで SSL を使用する必要がある場合は、OracleSSL を使用してください。
- JSSE の詳細は、http://java.sun.com/products/jsse/ にある Sun 社のドキュメントを参照してください。

HTTPClient はデフォルト・プロバイダとして OracleSSL も使用しますが、開発者は HTTPConnection クラスで SSLSocketFactory を設定してデフォルト・プロバイダを簡単に変更できます。次のコード断片に、SSL 通信に JSSE を使用するようにクライアントで HTTPClient を構成する方法を示します。

```
public void obtainHTTPSConnectionUsingJSSE() throws Exception
{
// set the trust store to the location of the client's trust store file
      // this value specifies the certificate authorities the client accepts
   System.setProperty("javax.net.ssl.trustStore", KEYSTORE_FILE);
```

```
// creates the HTTPS URL
URL testURL = new URL("https://" + HOSTNAME + ":" + HTTPS_PORTNUM);
// call SSLSocketFactory.getDefault() to obtain the default JSSE implementation
// of an SSLSocketFactory
SSLSocketFactory socketFactory = (SSLSocketFactory)SSLSocketFactory.getDefault();
HTTPConnection connection = new HTTPConnection(testURL);

// configure HTTPClient to use JSSE as the underlying
// SSL provider
connection.setSSLSocketFactory(socketFactory);
// call connect to setup SSL handshake
try
{
    connection.connect();
}
catch (IOException e)
{
    e.printStackTrace();    }

HTTPResponse response = connection.Get("/index.html");

    }
```

JSSE を使用するための HTTPClient の構成

HTTPClient で JSSE を使用するための必須手順は、次のとおりです。

1. keytool を使用してトラストストアを作成します。

注意：

- keytool の使用方法の詳細は、http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html を参照してください。
- JSSE の SSL 実装は、Oracle の実装とはわずかに異なっています。OracleSSL とは異なり、トラストストアを設定しないと、JDK のデフォルトのトラストストアが使用されます。このデフォルトは、Verisign 社や Thawte 社など、既知の認証局を受け入れます。多数の自己署名付き証明書は、このデフォルトでは拒否されます。

---

2. トラストストアのプロパティを設定します。JSSE の使用を希望するクライアントでは、javax.net.ssl.trustStore にクライアント・トラストストアの位置を指定する必要があります。OracleSSL とは異なり、クライアントで javax.net.ssl.keyStore プロパティを設定する必要はありません。

3. SSLSocketFactory.getDefault() をコールして JSSE SSLSocketFactory を取得します。
4. HTTPClient 接続を作成します。
5. SSL の JSSE 実装を使用するように HTTPClient 接続を構成します。JSSE を使用するように HTTPClient を構成するには、次の 2 つの方法があります。
    1. （**接続ごと**）クライアントで HTTPConnection.setSSLSocketFactory(SSLSocketFactory factory) をコールします。
    2. （**VM 全体**）クライアントで静的メソッド HttpConnection.setDefaultSSLSocketFactory(SSLSocketFactory factory) をコールします。この静的メソッドは、HTTPConnection インスタンスをインスタンス化する前にコールする必要があります。
6. HTTPS データを送信する前に、HTTPConnection.connect() をコールします。これにより、接続では、データを暗号化して送信する前に、クライアントとサーバー間に発生する必要のある SSL ハンドシェイキングを検証できます。
7. HTTPConnection インスタンスを通常の方法で使用します。この時点で、クライアントは JSSE とともに HTTPClient を使用するようにセットアップされています。追加構成は不要であり、基本的な使用方法は同じです。

SSL を使用するための Oracle HTTP Server と OC4J の構成

Oracle HTTP Server と OC4J 間のセキュアな接続のために、接続の両端で構成手順を実行する必要があります。以降の各項では、それぞれの構成手順について説明します。

- Oracle HTTP Server の SSL 用構成手順
- OC4J の SSL 用構成手順

Oracle HTTP Server の SSL 用構成手順

Oracle HTTP Server で、mod_oc4j.conf 内の SSL 設定がセキュアな通信用に適切な値に設定されていることを確認します。次のように、Wallet ファイルとパスワードを指定して SSL を有効化する必要があります。

```
Oc4jEnableSSL on
Oc4jSSLWalletFile wallet_path
Oc4jSSLWalletPassword pwd
```

wallet_path 値は、ファイル名を除く Wallet へのディレクトリ・パスです。（Wallet ファイル名は判明済です。）*pwd* 値は Wallet パスワードです。

mod_oc4j.conf ファイルの詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。

OC4J の SSL 用構成手順

default-web-site.xml ファイル（または、該当する場合は他の Web サイトの XML ファイル）内で、<web-site> 要素の下に適切な SSL 設定を指定します。

1. 次のように、secure フラグをオンにしてセキュアな接続を指定します。

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

   secure="true" に設定することで、AJP プロトコルで SSL ソケットを使用するように指定します。

2. 次のように、<ssl-config> サブ要素と keystore および keystore-password 属性を使用して、キーストアのパスとパスワードを指定します。

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

   secure フラグを "true" に設定している場合は、<ssl-config> 要素が必須です。

   path_and_file 値では、ファイル名を含むディレクトリの絶対パスまたは相対パスを指定できます。相対パスは、Web サイトの XML ファイルの位置への相対パスです。

3. 必要な場合は、クライアント認証を必須として指定するために needs-client-auth フラグをオンにします。これは <ssl-config> 要素の属性です。

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

   これにより、OC4J がセキュアな接続のために Oracle HTTP Server などのクライアント・エンティティを ID に応じて受け入れるか拒否するモードが設定されます。needs-client-auth フラグでは、OC4J に対して接続時にクライアント証明連鎖を要求するように指示します。OC4J でクライアントのルート証明書が認識されると、そのクライアントが受け入れられます。

   <ssl-config> 要素に指定するキーストアには、セキュアな AJP および SSL を介した OC4J への接続が認可されたクライアントの証明書が含まれている必要があります。

次に、クライアント認証によるセキュアな接続をセットアップする例を示します。

```
<web-site display-name="OC4J Web Site" protocol="ajp13" secure="true" >
   <default-web-app application="default" name="defaultWebApp" root="/j2ee" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome"
               needs-client-auth="true" />
</web-site>
```

セキュリティ固有の部分は、太字部分のみです。セキュアな通信を使用するかどうかに関係なく、Oracle HTTP Server を介した通信の場合、protocol 値は常に "ajp13" です。secure="false" の protocol 値 ajp13 は AJP プロトコルを示し、secure="true" の ajp13 はセキュアな AJP プロトコルを示します。

<web-site> および <ssl-config> 要素とその属性の詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』を参照してください。

SSL 用の OC4J スタンドアロンの構成

クライアントと OC4J 間のセキュアな通信のためには、OC4J スタンドアロン上での構成が必要です。クライアント・サイドで証明書を提供する必要があるのは、クライアント認証を構成する場合のみです。

OC4J の default-web-site.xml ファイル（または、該当する場合は他の Web サイトの XML ファイル）内で、<web-site> 要素の下に適切な SSL 設定を指定します。

1. 次のように、secure フラグをオンにしてセキュアな接続を指定します。

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

   secure="true" に設定することで、HTTP プロトコルで SSL ソケットを使用するように指定します。

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

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

   secure フラグを "true" に設定している場合は、<ssl-config> 要素が必須です。

   path_and_file 値では、ファイル名を含むディレクトリの絶対パスまたは相対パスを指定できます。

注意： パスワードの間接化を介してパスワードを非表示にすることができます。

3. 必要な場合は、次のように <ssl-config> 要素の属性である needs-client-auth フラグをオンにして、クライアント認証が必須であることを指定します。

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

この手順では、OC4J が ID に応じてセキュアな通信のためにクライアント・エンティティを受け入れるか拒否するモードを設定します。needs-client-auth 属性では、OC4J に対して接続時にクライアント証明連鎖を要求するように指示します。クライアントのルート証明書が認識されると、そのクライアントが受け入れられます。

<ssl-config> 要素に指定するキーストアには、HTTPS を介した OC4J への接続が認可されたクライアントの証明書が含まれている必要があります。

4. 必要な場合は、Web サイト内の各アプリケーションを共有として指定します。<web-app> 要素の shared 属性は、複数のバインド（異なる Web サイト、またはポートおよびコンテキスト・ルート）を共有できるかどうかを示します。サポートされる値は、"true" および "false"（デフォルト）です。

共有は、セッション、サーブレット・インスタンスおよびコンテキスト値など、Web アプリケーションを構成するすべてを共有することを意味します。このモードの標準的な使用方法は、通信のすべてではなく一部に SSL が必要な場合に、同じコンテキスト・パスにある HTTP サイトと HTTPS サイト間で Web アプリケーションを共有することです。すべての情報ではなく重要な情報のみを暗号化することでパフォーマンスが改善されます。

HTTPS Web アプリケーションを共有としてマークすると、セッションの追跡には SSL 証明書ではなく Cookie が使用されます。この方法には、SSL 証明書で 50K を使用して追跡中に各証明書が格納されるというメリットがありますが、セッションがタイムアウトになる前にセッションにメモリー不足の問題が発生する場合があります。このため、Web アプリケーションのセキュリティが低下する可能性はありますが、一部のブラウザでは SSL セッションのタイムアウトが適切にサポートされないなどの問題について回避作業が必要になる場合があります。

5. shared が true で、デフォルト・ポートが使用されない場合は、必要に応じて Cookie ドメインを設定します。クライアントが別個のポートで Web サーバーと対話する場合、Cookie ではそれぞれのポートが別個の Web サイトを指すものと見なされます。HTTP のデフォルト・ポート 80 と HTTPS のデフォルト・ポート 443 を使用すると、クライアントではこの 2 つが同じ Web サイトの 2 つの異なるポートとして認識され、Cookie は 1 つのみ作成されます。ただし、デフォルト以外のポートを使用すると、クライアントではこれらのポートが同じ Web サイトの一部として認識されず、Cookie ドメインを指定しなければポートごとに Cookie が個別に作成されます。

   Cookie ドメインでは、DNS ドメイン内で複数のサーバーにまたがるクライアントの通信が追跡されます。HTTP と HTTPS を使用する共有環境にデフォルト以外のポートを使用する場合は、アプリケーションの orion-web.xml ファイル内で <session-tracking> 要素の cookie-domain 属性を設定します。cookie-domain 属性には、指定のドメイン名の構成要素を 2 つ以上持つ DNS ドメインが含まれます。

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

例 11-1 クライアント認証を使用した HTTPS 通信

次の例では、Web サイトをクライアント認証を使用した HTTPS によるセキュアな通信用に構成しています。

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

セキュリティ固有の部分は、太字部分のみです。セキュアな通信を使用するかどうかに関係なく、HTTP 通信の場合、protocol 値は常に "http" です。secure="false" の protocol 値 http は HTTP プロトコルを示し、secure="true" の http は HTTPS プロトコルを示します。

次に、HTTP 接続と HTTPS 接続の両方を受け入れるように新規アプリケーションを構成します。

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

この Web サイトでは、HTTP 通信と HTTPS 通信にデフォルトのポート番号を使用します。デフォルト以外のポート番号を使用する場合は、cookie-domain 属性も追加します。

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

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

例 11-2 SSL 証明書の作成と HTTPS の構成

この例では、keytool を使用してテスト証明書を作成し、HTTPS の動作に必要なすべての XML 構成を示します。本番環境で使用する有効な証明書を作成する方法は、keytool のドキュメントを参照してください。

1. 適切な JDK をインストールします。

   JDK 1.3.x がインストールされていることを確認します。OC4J と SSL には、このバージョンが必須です。JAVA_HOME を JDK 1.3 のディレクトリに設定します。JDK 1.3.x の JAVA_HOME/bin がパスの先頭に指定されていることを確認します。このパスの設定手順は、次のとおりです。

   UNIX

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

   Windows

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

   この JDK バージョンが Windows レジストリ内で現行バージョンとして設定されていることを確認します。Windows Registry Editor で、HKEY_LOCAL_MACHINE/SOFTWARE/JavaSoft/Java Development Kit の下の 'CurrentVersion' を 1.3（以上）に設定します。

2. 証明書を要求します。

   a. ディレクトリ ORACLE_HOME/j2ee に移動します。

   b. keytool コマンドを使用し、RSA の秘密鍵 / 公開鍵のペアを指定してキーストアを作成します。この例では、次の構文で 'RSA' 鍵ペア生成アルゴリズムを使用し、パスワード '123456'、有効期間 21 日を指定して、ファイル 'mykeystore' に常駐するキーストアを生成します。

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

   各項目の意味は次のとおりです。

   - keystore オプションでは、鍵が格納されるファイルの名前を設定します。
   - storepass オプションでは、キーストアを保護するためのパスワードを設定します。
   - validity オプションでは、証明書の有効期間を表す日数を設定します。

keytool により、次のように詳細情報を求めるプロンプトが表示されます。

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

What is your first and last name?
  [Unknown]:  Test User
What is the name of your organizational unit?
  [Unknown]:  Support
What is the name of your organization?
  [Unknown]:  Oracle
What is the name of your City or Locality?
  [Unknown]:  Redwood Shores
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Test User, OU=Support, O=Oracle, L=Reading, ST=Berkshire, C=GB> correct?
  [no]:  yes

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

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

---

mykeystore ファイルは、現行ディレクトリに作成されます。鍵のデフォルトの別名は mykey です。

3. secure-web-site.xml ファイルがない場合は、default-web-site.xml を $ORACLE_HOME/j2ee/home/config/secure-web-site.xml にコピーします。
4. 次の要素を使用して secure-web-site.xml を編集します。
   a. 次のように、<web-site> 要素に secure="true" を追加します。

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

   b. <web-site> 要素内に次の行を追加してキーストアとパスワードを定義します。

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

<Your-Keystore> はキーストアへのフルパス、<Your-Password> はキーストアのパスワードです。この例では、次のように指定します。

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

注意： キーストアのパスは、XML ファイルのディレクトリへの相対パスです。

c. 使用可能なポートを使用できるように、Web サイトのポート番号を変更します。たとえば、SSL のデフォルト・ポートは 443 であるため、Web サイトの port 属性を port="4443" に変更します。デフォルトの 443 を使用するには、スーパー・ユーザーになる必要があります。

d. 変更結果を secure-web-site.xml に保存します。

5. secure-web-site.xml ファイルがない場合は、secure-web-site.xml ファイルを指すように server.xml を編集します。

a. secure-web-site.xml ファイルが読み取られるように、ファイル server.xml 内で次の行をコメント解除するか追加します。

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

注意： Windows の場合も、XML ファイルには円記号ではなくスラッシュを使用します。

b. 変更結果を server.xml に保存します。

6. OC4J を停止してから再起動し、secure-web-site.xml ファイルの追加を初期化します。ブラウザを使用して SSL ポートでサイトにアクセスし、SSL ポートをテストします。アクセスに成功すると、受け入れられる認証局の署名がないため、証明書を受け入れるかどうかを確認するプロンプトが表示されます。

完了すると、OC4J はあるポートで SSL リクエストをリスニングし、別のポートで非 SSL リクエストをリスニングします。server.xml 構成ファイル内で適切な *web-site.xml をコメント解除すると、SSL リクエストまたは非 SSL リクエストを無効化できます。

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

OC4J スタンドアロンでのクライアント認証の要求

OC4J では、クライアント認証モードがサポートされます。このモードでは、サーバーはクライアントと通信する前に、クライアントからの認証を明示的に要求します。この場合、クライアントには独自の証明書が必要です。クライアントは、証明書とルート証明書で終わる証明連鎖を送信することで自己認証を行います。OC4J は、クライアントに至る信頼の連鎖を確立するときに、指定のリストからルート証明書のみを受け入れるように構成できます。

OC4J が信頼する証明書は、トラスト・ポイントと呼ばれます。これは、OC4J がクライアントからの証明連鎖で検出する最初の証明書であり、独自のキーストア内の証明書と一致しています。この信頼関係は、次の 3 つの方法のいずれかで構成されます。

- クライアント証明書がキーストアにあること。
- クライアントの連鎖に含まれる中間の認証局の証明書の 1 つがキーストアにあること。
- クライアントの連鎖に含まれるルートの認証局の証明書がキーストアにあること。

OC4J では、捏造された証明書を防ぐために、トラスト・ポイントを含む証明連鎖全体が有効かどうかが検証されます。

`needs-client-auth` 属性を使用してクライアント認証を要求する場合の手順は、次のとおりです。

1. クライアントの連鎖のうちトラスト・ポイントにする証明書を決定します。このトラスト・ポイントを使用して証明書の発行を制御できること、または認証局を発行者として信頼できることを確認します。
2. クライアント証明書の認証用に、中間またはルート証明書をトラスト・ポイントとしてサーバーのキーストアにインポートします。
3. OC4J が特定のトラスト・ポイントにアクセスしないようにする場合は、これらのトラスト・ポイントがキーストアにないことを確認してください。
4. 前述の手順を実行して、サーバーにインストールされている中間またはルートの証明書を含むクライアント証明書を作成します。他の認証局を信頼する場合は、その認証局から証明書を取得します。
5. 証明書をクライアント上のファイルに保存します。
6. クライアントが HTTPS 接続を開始するときに、証明書を提供します。
   a. クライアントがブラウザの場合は、クライアントのブラウザのセキュリティ領域内で証明書を設定します。
   b. クライアントが Java クライアントの場合は、HTTPS 接続の開始時に、クライアント証明書と証明連鎖をプログラムで提示する必要があります。

HTTPS の一般的な問題と解決策

SSL 証明書の使用時には、次のエラーが発生する場合があります。

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

原因 : keytool では、後続の空白は使用できません。

処置 : 後続の空白をすべて削除します。それでもエラーが発生する場合は、証明書応答ファイルに新規の 1 行を追加します。

Keytool Error: KeyPairGenerator not available

原因 : 以前のバージョンの JDK から keytool を使用していると思われます。

処置 : システム上の最新 JDK から keytool を使用してください。最新 JDK を使用していることを確認するには、この JDK のフルパスを指定します。

Keytool Error: Failed to establish chain from reply

原因 : keytool では、キーストア内でルート CA 証明書が見つからないため、サーバーの鍵から信頼できるルート認証局への証明連鎖を構築できません。

処置 : 次を実行します。

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

中間 CA の keytool を使用する場合は、次を実行します。

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

証明書署名要求から証明書を取得して、次を実行します。

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

No available certificate corresponds to the SSL cipher suites which are enabled

原因 : 証明書に誤りがあります。

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

原因 : 同じポートおよび IP アドレスをリスニングするように SSL Web サイトと非 SSL Web サイトを構成することはできません。

処置 : `secure-web-site.xml` および `default-web-site.xml` ファイル内で異なるポートが割り当てられているかどうかをチェックします。

Keytool does not work on HP-UX

原因 : HP-UX 上では、'keytool' が RSA オプションでは動作しないことが報告されています。

処置 : 別のプラットフォーム上で鍵を生成し、それを HP-UX サーバーに FTP で転送します。

12

EJB のセキュリティ

この章では、EJB に影響するセキュリティの問題について説明します。この章には、次の項目が含まれます。

- EJB の JNDI セキュリティ・プロパティ
- セキュリティの構成

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

EJB の JNDI セキュリティ・プロパティ

セキュリティ固有の 2 つの JNDI プロパティがあります。これらのプロパティは、jndi.properties ファイル内または EJB 実装内で設定できます。

jndi.properties 内の JNDI プロパティ

jndi.properties ファイル内で JNDI プロパティを設定する場合は、各プロパティを次のように設定します。この jndi.properties ファイルに CLASSPATH からアクセスできることを確認してください。

リモート・コンテナ内の EJB にアクセスする場合は、このコンテナに有効な資格証明を渡す必要があります。スタンドアロン・クライアントでは、資格証明をクライアントのコードとともにデプロイされる jndi.properties ファイル内で定義します。

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

実装内の JNDI プロパティ

各プロパティには、構文は異なりますが同じ値を設定します。たとえば、コンテナ内で実行される JavaBeans は、リモート EJB の参照用に作成される InitialContext 内で資格証明を渡します。

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

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

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

セキュリティの構成

EJB のセキュリティには 2 つのレルムが関係します。ブラウザにダウンロードする場合はパーミッションを付与し、認証と認可のためにアプリケーションを構成します。この項には、次の項目が含まれます。

- ブラウザでのパーミッションの付与
- EJB アプリケーションの認証と認可
- EJB クライアントでの資格証明の指定

ブラウザでのパーミッションの付与

EJB アプリケーションをセキュリティ・マネージャがアクティブになっているクライアントとしてダウンロードする場合は、実行前に次のパーミッションを付与する必要があります。

```
permission java.net.SocketPermission "*:*", "connect,resolve";
permission java.lang.RuntimePermission "createClassLoader";
permission java.lang.RuntimePermission "getClassLoader";
permission java.util.PropertyPermission "*", "read";
permission java.util.PropertyPermission "LoadBalanceOnLookup",
  "read,write";
```

EJB アプリケーションの認証と認可

EJB の認証と認可のために、EJB デプロイメント・ディスクリプタを構成し、各メソッドが実行されるプリンシパルを定義します。コンテナでは、メソッドの実行を試みるユーザーはデプロイメント・ディスクリプタに定義されているユーザーと同じであると規定されます。

EJB デプロイメント・ディスクリプタを使用すると、各メソッドの実行が許可されるセキュリティ・ロールを定義できます。これらのメソッドは、OC4J 固有のデプロイメント・ディスクリプタ内でユーザーまたはグループにマップされます。ユーザーとグループは、JAZN または XML ユーザー・マネージャを使用する指定のセキュリティ・ユーザー・マネージャ内で定義します。セキュリティ・ユーザー・マネージャの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』および『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

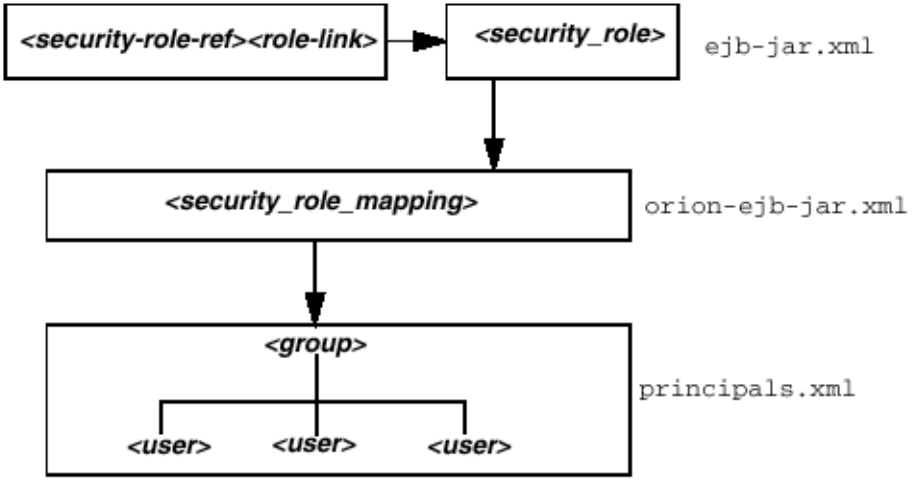
この項では、認証と認可について EJB デプロイメント・ディスクリプタ内の XML 構成に重点を置いて説明します。EJB の認可は、EJB および OC4J 固有のデプロイメント・ディスクリプタ内で指定されます。デプロイメント・ディスクリプタ内では、セキュリティの認可部分を次のように管理できます。

- EJB デプロイメント・ディスクリプタには、論理ロールを使用してアクセス・ルールを記述します。
- OC4J 固有のデプロイメント・ディスクリプタでは、論理ロールを JAZN または XML ユーザー・マネージャで定義されている個別のユーザーとグループにマップします。

ユーザーとグループは、コンテナにより認識される ID です。ロールは、各アプリケーションで様々なオブジェクトへのアクセス権を示すために使用される論理 ID です。ユーザー名 / パスワードは、デジタル証明であり、SSL の場合は秘密鍵のペアです。

図 12-1 に、ロールの定義とマッピングを示します。

図 12-1 ロールのマッピング



以降の各項では、ユーザー、グループおよびロールの定義について説明します。

- ユーザーとグループの指定
- EJB デプロイメント・ディスクリプタでの論理ロールの指定
- EJB メソッドのセキュリティ・チェック対象外の指定
- runAs セキュリティ ID の指定
- 論理ロールからユーザーおよびグループへのマッピング
- 未定義のメソッドに関するデフォルトのロール・マッピングの指定
- クライアントによるユーザーとグループの指定

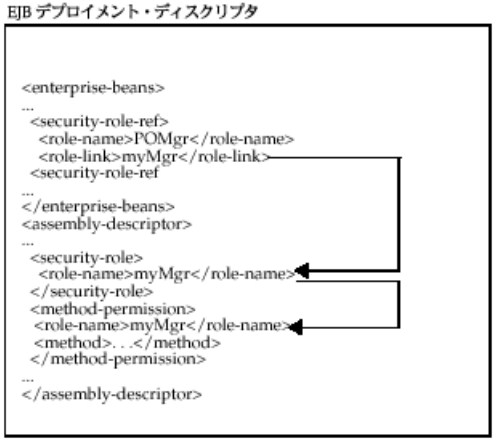
ユーザーとグループの指定

OC4J では、ユーザーとグループの定義がサポートされます。これらは、デプロイされたすべてのアプリケーションで共有される場合と、特定のアプリケーションに固有の場合があります。共有またはアプリケーション固有のユーザーとグループは、JAZN または XML ユーザー・マネージャで定義します。詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』および『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

EJB デプロイメント・ディスクリプタでの論理ロールの指定

図 12-2 に示すように、Bean 実装内でロールの論理名を使用し、この論理名を適切なデータベース・ロールまたはユーザーにマップできます。データベース・ロールに対する論理名のマッピングは、OC4J 固有のデプロイメント・ディスクリプタで指定します。詳細は、12-9 ページの「論理ロールからユーザーおよびグループへのマッピング」を参照してください。

図 12-2 セキュリティのマッピング



isCallerInRole などのメソッドの Bean 実装内でデータベース・ロールに論理名を使用する場合は、その論理名を次の方法で実際のデータベース・ロールにマップできます。

1. <enterprise-beans> セクションの <security-role-ref> 要素内で論理名を宣言します。たとえば、発注の例で使用するロールを定義するには、Bean の実装内でコール元が発注への署名を認可されているかどうかをチェックしておくことができます。そのため、コール元は適切なロールに基づく署名を受ける必要があります。発注書に署名できるのは発注管理者のみであるため、Bean がデータベース・ロールを認識しなくてもよいように、POMgr などの論理名の isCallerInRole をチェックできます。つまり、次のように、<enterprise-beans> セクションの

<security-role-ref><role-name> 要素内で論理セキュリティ・ロール POMgr を定義します。

```
<enterprise-beans>
...
  <security-role-ref>
   <role-name>POMgr</role-name>
   <role-link>myMgr</role-link>
  </security-role-ref>
</enterprise-beans>
```

<security-role-ref> 要素内の <role-link> 要素に実際のデータベース・ロールを指定し、このデータベース・ロールをさらに <assembly-descriptor> セクション内で定義できます。または、同じく <assembly-descriptor> セクション内で定義する別の論理名を使用し、Oracle 固有のデプロイメント・ディスクリプタ内で実際のデータベース・ロールにマップできます。

注意： <security-role-ref> 要素は必須ではありません。この要素を指定するのは、Bean 内でセキュリティ・コンテキスト・メソッドを使用する場合のみです。

---

2. ロールとそれを適用するメソッドを定義します。発注の例では、PurchaseOrder Bean 内で実行されるすべてのメソッドは、メソッド自体を myMgr として認可している必要があります。PurchaseOrder は <entity | session><ejb-name> 要素内で宣言されている名前であることに注意してください。

   次の例では、ロールとして myMgr、EJB として PurchaseOrder を定義し、'*' 記号を使用してすべてのメソッドを定義します。

注意： <security-role> 要素内の myMgr ロールは、<enterprise-beans> セクションの <role-link> 要素と同じです。これにより、POMgr の論理名が myMgr の定義に結合されます。

```
<assembly-descriptor>
 <security-role>
  <description>Role needed purchase order authorization</description>
  <role-name>myMgr</role-name>
 </security-role>
 <method-permission>
  <role-name>myMgr</role-name>
  <method>
   <ejb-name>PurchaseOrder</ejb-name>
   <method-name>*</method-name>
```

```
  </method>
 </method-permission>
...
</assembly-descriptor>
```

手順 1 および 2 を実行した後、Bean の実装内で POMgr を参照でき、コンテナにより POMgr を myMgr に変換できます。

> **注意：** 同じ EJB 内でメソッドの <method-permission> 要素に異なるロールを定義すると、この Bean のメソッドに対して定義されているメソッド・パーミッションすべての共用体が付与されます。

<method-permission><method> 要素は、インタフェースまたは実装内で 1 つ以上のメソッドのセキュリティ・ロールを指定するために使用します。EJB 仕様によれば、この定義には次のいずれかの書式を使用できます。

1. 次のように、Bean 名を指定し、Bean 内のすべてのメソッドを示す '*' 文字を使用して、Bean 内のすべてのメソッドを定義します。

```
<method-permission>
  <role-name>myMgr</role-name>
  <method>
  <ejb-name>EJBNAME</ejb-name>
  <method-name>*</method-name>
  </method>
</method-permission>
```

2. Bean 内で一意に識別される特定のメソッドを定義します。次のように、適切なインタフェース名とメソッド名を使用します。

```
<method-permission>
  <role-name>myMgr</role-name>
  <method>
  <ejb-name>myBean</ejb-name>
  <method-name>myMethodInMyBean</method-name>
  </method>
</method-permission>
```

> **注意：** 同じオーバーロード名を持つ複数のメソッドが存在する場合、このスタイルの要素はそのオーバーロード名を持つすべてのメソッドを参照します。

3. 次のように、多数のオーバーロード・バージョンのうち特定のシグネチャを持つメソッドを定義します。

```
<method-permission>
  <role-name>myMgr</role-name>
  <method>
      <ejb-name>myBean</ejb-name>
   <method-name>myMethod</method-name>
   <method-params>
      <method-param>javax.lang.String</method-param>
      <method-param>javax.lang.String</method-param>
       </method-params>
  </method>
</method-permission>
```

各パラメータは、メソッドの入力パラメータの完全修飾 Java 型です。メソッドに入力引数がない場合、<method-params> 要素に他の要素は含まれません。配列を指定するには、int[ ][ ] のように、配列要素の型に続けて 1 組以上の大カッコを使用します。

EJB メソッドのセキュリティ・チェック対象外の指定

特定のメソッドをセキュリティ・ロールのチェック対象外にする場合は、次のようにこれらのメソッドをチェック対象外として定義します。

```
<method-permission>
  <unchecked/>
  <method>
     <ejb-name>EJBNAME</ejb-name>
     <method-name>*</method-name>
  </method>
</method-permission>
```

<role-name> 要素のかわりに <unchecked/> 要素を定義します。EJBNAME Bean 内のメソッドを実行すると、コンテナではセキュリティ・チェックが実行されません。チェック対象外のメソッドは、常に他のロール定義より優先されます。

runAs セキュリティ ID の指定

特定の ID を使用して実行される EJB のメソッドすべてを指定できます。つまり、コンテナでは、様々なロールで特定のメソッドを実行するためのパーミッションはチェックされず、指定されたセキュリティ ID を使用する EJB メソッドがすべて実行されます。特定のロールまたはコール元の ID をセキュリティ ID として指定できます。

<enterprise-beans> セクションの <security-identity> 要素内でセキュリティ ID を指定します。次の XML に、POMgr はすべてのエンティティ Bean メソッドが実行されるロールであることを示します。

```
<enterprise-beans>
 <entity>
 ...
  <security-identity>
     <run-as>
        <role-name>POMgr</role-name>
     </run-as>
  </security-identity>
...
 </entity>
</enterprise-beans>
```

また、次の XML の例に、Bean のすべてのメソッドをコール元の ID を使用して実行するように指定する方法を示します。

```
<enterprise-beans>
 <entity>
 ...
  <security-identity>
     <use-caller-identity/>
  </security-identity>
...
 </entity>
</enterprise-beans>
```

論理ロールからユーザーおよびグループへのマッピング

EJB デプロイメント・ディスクリプタには、論理ロールまたは実際のユーザーおよびグループを使用できます。ただし、論理ロールを使用する場合は、それを JAZN または XML ユーザー・マネージャで定義されている実際のユーザーおよびグループにマップする必要があります。

アプリケーションのデプロイメント・ディスクリプタに定義されている論理ロールを JAZN または XML ユーザー・マネージャのユーザーまたはグループにマップするには、OC4J 固有のデプロイメント・ディスクリプタの `<security-role-mapping>` 要素を使用します。

- この要素の `name` 属性には、マップする論理ロールを定義します。
- `group` または `user` 要素により、論理ロールがグループ名またはユーザー名にマップされます。このグループまたはユーザーは、JAZN または XML ユーザー・マネージャ構成内で定義する必要があります。JAZN および XML ユーザー・マネージャの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』および『Oracle Application Server Containers for J2EE サービス・ガイド』を参照してください。

例 12-1 論理ロールから実際のロールへのマッピング

この例では、論理ロール POMGR を orion-ejb-jar.xml ファイル内で managers グループにマップします。このグループのメンバーとしてログインできるユーザーは、POMGR ロールを持つものと見なされます。つまり、PurchaseOrderBean のメソッドを実行できます。

```
<security-role-mapping name="POMGR">
 <group name="managers" />
</security-role-mapping>
```

注意： 論理ロールは、1 つまたは複数のグループにマップできます。

このロールを特定のユーザーにマップするには、次のように指定します。

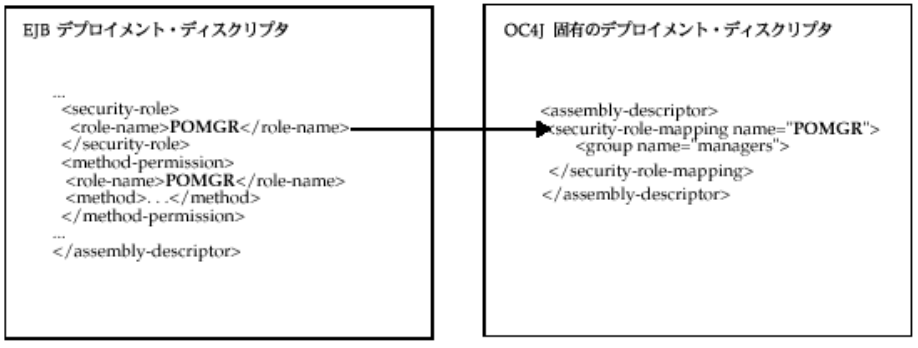
```
<security-role-mapping name="POMGR">
 <user name="guest" />
</security-role-mapping>
```

次のように、1 つのロールを特定のグループに属する特定のユーザーにマップできます。

```
<security-role-mapping name="POMGR">
 <group name="managers" />
 <user name="guest" />
</security-role-mapping>
```

図 12-3 に示すように、EJB デプロイメント・ディスクリプタに定義されている POMGR の論理ロール名は、OC4J 固有のデプロイメント・ディスクリプタの <security-role-mapping> 要素内の managers にマップされます。

図 12-3 セキュリティのマッピング



EJB デプロイメント・ディスクリプタ内の <role-name> が、OC4J 固有のデプロイメント・ディスクリプタの <security-role-mapping> 要素内の name 属性と同じであることに注意してください。この値によりマッピングが識別されます。

未定義のメソッドに関するデフォルトのロール・マッピングの指定

メソッドがロール・マッピングに関連付けられていない場合は、orion-ejb-jar.xml ファイル内の <default-method-access> 要素を介してデフォルトのセキュリティ・ロールにマップされます。次の例に、セキュアでないメソッドの自動マッピングを示します。

```
<default-method-access>
   <security-role-mapping name="&lt;default-ejb-caller-role&gt;"
                          impliesAll="true" />
   </security-role-mapping>
</default-method-access>
```

デフォルト・ロールは <default-ejb-caller-role> で、name 属性に定義されています。この文字列は、任意のデフォルト・ロール名で置き換えることができます。impliesAll 属性は、これらのメソッドに対するセキュリティ・ロール・チェックが発生するかどうかを示します。この属性のデフォルトは true で、これらのメソッドにはセキュリティ・ロール・チェックが発生しないことを示します。この属性を false に設定すると、コンテナではこれらのメソッドでこのデフォルト・ロールがチェックされます。

impliesAll 属性が false の場合は、<user> および <group> 要素を介して、name 属性に定義されているデフォルト・ロールを JAZN または XML のユーザーまたはグループにマップする必要があります。次の例に、メソッド・パーミッションに関連付けられていないすべてのメソッドを「others」グループにマップする方法を示します。

```
<default-method-access>
   <security-role-mapping name="default-role" impliesAll="false" />
      <group name="others" />
   </security-role-mapping>
</default-method-access>
```

クライアントによるユーザーとグループの指定

クライアントがユーザー別およびグループ別に保護されているメソッドにアクセスするには、JAZN または XML ユーザー・マネージャで認識されるパスワードを指定して適切なユーザー名またはグループ名を提供する必要があります。また、ユーザーまたはグループは、アクセスするメソッドのセキュリティ・ロールに指定されているユーザーまたはグループと同じであることが必要です。詳細は、12-12 ページの「EJB クライアントでの資格証明の指定」を参照してください。

EJB クライアントでの資格証明の指定

リモート・コンテナ内の EJB にアクセスする場合は、このコンテナに有効な資格証明を渡す必要があります。

- スタンドアロン・クライアントでは、資格証明を EAR ファイルとともにデプロイされる `jndi.properties` ファイル内で定義します。
- コンテナ内で実行されるサーブレットまたは JavaBeans は、リモート EJB の参照用に作成される `InitialContext` 内で資格証明を渡します。

JNDI プロパティ内の資格証明

`jndi.properties` ファイル内でリモート EJB の検索時に使用するユーザー名（プリンシパル）とパスワード（資格証明）を指定します。

たとえば、リモート EJB に `POMGR/welcome` としてアクセスする場合は、次のプロパティを定義します。`factory.initial` プロパティは、Oracle JNDI 実装を使用することを示します。

```
java.naming.security.principal=POMGR
java.naming.security.credentials=welcome
java.naming.factory.initial=
   com.evermind.server.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://myhost/ejbsamples
```

アプリケーション・プログラムで、次のようにリモート EJB を認証してアクセスします。

```
InitialContext ic = new InitialContext();
CustomerHome =
(CustomerHome)ic.lookup("java:comp/env/purchaseOrderBean");
```

初期コンテキスト内の資格証明

サーブレットまたは JavaBeans からリモート EJB にアクセスするには、次のように `InitialContext` オブジェクト内で資格証明を渡します。

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

13

J2EE Connector Architecture のセキュリティ

この章では、Oracle Application Server Containers for J2EE（OC4J）アプリケーションの J2EE Connector Architecture に影響するセキュリティの問題について説明します。J2EE Connector Architecture 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。この章には、次の項目が含まれます。

リソース・アダプタのデプロイ

この項では、デプロイメント・ディスクリプタ、スタンドアロン・リソース・アダプタのデプロイおよび埋込みリソース・アダプタのデプロイについて説明します。

Oracle Application Server Containers for J2EE では、ra.xml、oc4j-ra.xml および oc4j-connectors.xml という 3 つのデプロイメント・ディスクリプタがサポートされます。通常、ra.xml ディスクリプタはリソース・アダプタに付属しています。EAR ファイル内のリソース・アダプタをデプロイするたびに、Oracle Application Server Containers for J2EE により oc4j-connectors.xml および oc4j-ra.xml が生成されます。後者のファイルを手作業で編集する必要があります。

oc4j-ra.xml ディスクリプタ

oc4j-ra.xml ディスクリプタは、リソースアダプタに関する Oracle Application Server Containers for J2EE 固有のデプロイメント情報（Java Naming and Directory Interface（JNDI）パス名およびコネクタ・プロパティ）を提供します。oc4j-ra.xml には、リソース・アダプタごとに、一連の構成パラメータ値に対応する JNDI 名を指定する 1 つ以上の <connector-factory> 要素が含まれています。Oracle Application Server Containers for J2EE では、各接続が適切な JNDI 名前空間の場所に ConnectionFactory インスタンスとしてバインドされます。

<connector-factory> 要素内で、EIS に対するユーザー名とパスワードの指定方法を記述するオプションの <security-config> 要素を使用できます。

<security-config> 要素

<security-config> 要素では、コンテナ管理のサインオンに使用するユーザー名とパスワードを指定します。

この情報を oc4j-ra.xml ファイルの <security-config> 要素で指定するには、次の 2 つの方法があります。

- マッピング・サブ要素を（<principal-mapping-entries> サブ要素内で）明示的に指定する方法
- oracle.j2ee.connector.PrincipalMapping を実装するか oracle.j2ee.AbstractPrincipalMapping から継承する、ユーザー作成のマッピング・クラスの名前を（<principal-mapping-interface> サブ要素内で）指定する方法

認証問題の詳細は、13-6 ページの「コンテナ管理のサインオンでの認証」を参照してください。この項では、<security-config> 要素の構文についてのみ説明します。

<security-config> 要素には、ユーザー名とパスワードを明示的に指定する <principal-mapping-entries> 要素、マッピング・クラス名を指定する <principal-mapping-interface> 要素または認証に使用する JAAS モジュールを指定する <jaas-module> 要素のいずれかが含まれます。

```
<security-config>
  <principal-mapping-entries>                          // 1
     <default-mapping>                                 // 2
        <res-user>username</res-user>                  // 3
        <res-password>password</res-password>          // 4
     </default-mapping>
     <principal-mapping-entry>                         // 5
        <initiating-user>iuname</initiating-user>      // 6
        <res-user>username</res-user>
        <res-password>password</res-password>
     </principal-mapping-entry>
  </principal-mapping-entries>

  <principal-mapping-interface>                        // 7
    <impl-class>classname</impl-class>                 // 8
    <property name="propname"
              value="propvalue" />                     // 9
  </principal-mapping-interface>

  <jaas-module>                                        // 10
      <jaas-application-name>                          // 11
         appname
      </jaas-application-name>
  </jaas-module>
</security-config>
```

1. <principal-mapping-entries>: リソース・マッピングの宣言仕様を提供します。この要素はオプションの <default-mapping> 要素で始まり、1 つ以上の <principal-mapping-entry> 要素が続きます。
2. <default-mapping>: デフォルト・リソース・プリンシパルのユーザー名とパスワードを指定します。
3. <res-user>: ユーザー名を指定します。
4. <res-password>: パスワードを指定します。

> **注意：** この要素ではパスワードの間接化がサポートされます。詳細は、10-3 ページの「間接パスワードの作成」を参照してください。

5. <principal-mapping-entry>: 単一の開始プリンシパルからリソース・プリンシパルとパスワードへのマッピングを指定します。
6. <initiating-user>: 開始プリンシパルを指定します。
7. <principal-mapping-interface>: ユーザー作成クラスを使用してマッピングを提供するために必要な情報を指定します。

8. <impl-class>: ユーザー提供の PrincipalMapping 実装の名前を指定します。
9. <property name="*propname*" value="*propvalue*">: PrincipalMapping 実装固有の情報を指定します。たとえば、プリンシパル・マッピング・ファイルのパスや LDAP サーバー接続情報などです（この要素はオプションで、繰り返し使用できます)。
10. <jaas-module>: 認証に使用する JAAS モジュールを指定します。この要素に含まれる要素は <jaas-application-name> の 1 つのみです。
11. <jaas-application-name>: 認証に使用する JAAS モジュールの名前を指定します。

oc4j-connectors.xml ディスクリプタ

oc4j-connectors.xml ディスクリプタでは、oc4j-ra.xml によりデプロイされるリソース・アダプタが構成されます。oc4j-connectors.xml ディスクリプタでは、アプリケーションに埋め込まれているリソース・アダプタとこの Oracle Application Server Containers for J2EE インスタンスでデプロイされるスタンドアロン・リソース・アダプタのリストが表示されます。このディスクリプタには、コネクタごとに、その名前とパス名を指定する <connector> 要素が含まれます。各 <connector> 要素には、各リソース・アダプタに付与されるパーミッションを定義する <security-permission> 要素が含まれます。次に構文を示します。

```
<security-permission enabled="booleanvalue">
```

この要素では、各リソース・アダプタに付与するパーミッションを指定します。各 <security-permission> には、Java 2 セキュリティ・ポリシー・ファイルの構文に準拠する <security-permission-spec> が含まれます。

Oracle Application Server Containers for J2EE では、ra.xml 内の <security-permission> 要素ごとに、oc4j-connectors.xml 内に <security-permission> 要素が自動的に生成されます。生成された各要素の enabled 属性は、false に設定されています。enabled 属性を true に設定すると、指定したパーミッションが付与されます。

例 :

```
<oc4j-connectors>
  <connector name="myEIS" path="eis.rar">
  . . .
    <security-permission>
      <security-permission-spec enabled="false">
        grant {permission java.lang.RuntimePermission "LoadLibrary", *'};
      </security-permission-spec>
    </security-permission>
  </connector>
</oc4j-connectors>
```

コンテナ管理またはコンポーネント管理のサインオンの指定

アプリケーションでは、アプリケーション・コンポーネントまたは Oracle Application Server Containers for J2EE アプリケーション・サーバーを使用して、EIS システムに対するリソース・アダプタのサインオンを管理できます。EJB または Web コンポーネントの場合は、<res-auth> デプロイメント・ディスクリプタ要素を使用してマネージャを指定します。<res-auth> を Application に設定すると、アプリケーション・コンポーネントは EIS にプログラムでサインオンします。アプリケーション・コンポーネントは、サインオン用のセキュリティ情報を明示的に提供する必要があります。<res-auth> を Container に設定すると、EIS へのサインオンに必要なリソースのプリンシパルと資格証明は Oracle Application Server Containers for J2EE が提供します。

例 :

```
Context initctx = new InitialContext();
// perform JNDI lookup to obtain a connection factory
javax.resource.cci.ConnectionFactory cxf =

(javax.resource.cci.ConnectionFactory)initctx.lookup("java:com/env/eis/MyEIS");
  // For container-managed sign-on, no security information is passed in the
getConnection call
  javax.resource.cci.Connection cx = cxf.getConnection();
  // If component-managed sign-on is specified, the code should instead provide
explicit security
  // information in the getConnection call
  // We need to get a new ConnectionSpec implementation instance for setting login
  // attributes
  com.myeis.ConnectionSpecImpl connSpec = ...
  connSpec.setUserName("EISuser");
  connSpec.setPassword("EISpassword");
  javax.resource.cci.Connection cx = cxf.getConnection(connSpec);
```

どちらの場合も、javax.resource.spi.ManagedConnectionFactory インタフェースのリソース・アダプタ実装内の createManagedConnection メソッドがコールされ、EIS への物理接続が作成されます。

コンポーネント管理のサインオンを指定すると、Oracle Application Server Containers for J2EE は null の Subject およびアプリケーション・コンポーネント・コードから渡される ConnectionRequestInfo オブジェクトを指定して createManagedConnection メソッドを起動します。コンテナ管理のサインオンを指定すると、Oracle Application Server Containers for J2EE は createManagedConnection メソッドに javax.security.auth.Subject オブジェクトを提供します。Subject オブジェクトの内容は、リソース・アダプタのデプロイメント・ディスクリプタに含まれる <authentication-mechanism-type> および <credential-interface> 要素の値に応じて異なります。

<authentication-mechanism-type> が BasicPassword で、かつ <credential-interface> が javax.resource.spi.security.PasswordCredential の場合、Subject オブジェクトは秘密資格証明セット javax.resource.spi.security.PasswordCredential にオブジェクトを含む必要があります。

一方、<authentication-mechanism-type> が Kerberos バージョン 5（Kerbv5）または他の非パスワードベース認証メカニズムで、かつ <credential-interface> が javax.resource.spi.security.GenericCredential の場合、Subject オブジェクトは javax.resource.spi.security.GenericCredential インタフェースの実装のインスタンスで表される資格証明を含む必要があります。GenericCredential インタフェースは、Kerberos など、非パスワードベース認証メカニズムをサポートするリソース・アダプタに使用されます。

コンテナ管理のサインオンでの認証

コンテナ管理のサインオンを使用する場合、Oracle Application Server Containers for J2EE は EIS にリソースのプリンシパルと資格証明を提供する必要があります。プリンシパルと資格証明は、次のいずれかの方法で取得できます。

- 構成済 ID: リソース・プリンシパルは開始プリンシパルつまりコール元プリンシパルに依存せず、デプロイメント時にデプロイメント・ディスクリプタ内で構成できます。
- プリンシパル・マッピング : リソース・プリンシパルは開始またはコール元プリンシパルの ID およびセキュリティ属性からのマッピングにより決定されます。
- コール元の偽装 : リソース・プリンシパルは、コール元の ID と資格証明を EIS に委任して、開始またはコール元プリンシパルにかわる役割を果たします。
- 資格証明マッピング : リソース・プリンシパルは開始またはコール元プリンシパルと同じですが、その資格証明は Oracle Application Server Containers for J2EE で使用される認証タイプから EIS で使用する認証タイプにマップされます。たとえば、プリンシパルに関連した公開鍵証明書ベースの資格証明は、Kerberos の資格証明にマップされます。

Oracle Application Server Containers for J2EE では、次の 3 つの認証メカニズムにより、これらの方式がすべてサポートされます。

- JAAS Pluggable Authentication
- ユーザー作成の認証クラス
- oc4j-ra.xml の変更

以降の各項では、これらのメカニズムを詳細に説明します。

JAAS Pluggable Authentication

Oracle Application Server Containers for J2EE には、Connector Architecture 1.0 仕様の付録 C に準拠する JAAS Pluggable Authentication フレームワークが用意されています。このフレームワークでは、アプリケーション・サーバーとその基礎となる認証サービスは相互に独立した状態を維持し、アプリケーション・サーバーに変更を加えずに新規認証サービスをプラグインできます。

認証サービスは、次のいずれかのモジュールを使用してリソース・プリンシパルと資格証明を取得します。

- プリンシパル・マッピング JAAS モジュール
- 資格証明マッピング JAAS モジュール
- Kerberos JAAS モジュール（コール元の偽装用）

JAAS ログイン・モジュールは、顧客、EIS ベンダーまたはリソース・アダプタ・ベンダーが提供できます。ログイン・モジュールは、Sun 社の JAAS 仕様に記載されているとおり `javax.security.auth.spi.LoginModule` インタフェースを実装する必要があります。

Oracle Application Server Containers for J2EE は、公開証明書を含む `javax.security.auth.Subject` のインスタンスと Oracle Application Server Containers for J2EE ユーザーを表す `oracle.j2ee.connector.InitiatingPrincipal` のインスタンスを渡すことで、開始ユーザー・オブジェクトをログイン・モジュールに提供します。Oracle Application Server Containers for J2EE では、未認証ユーザー（つまり、匿名ユーザー）がいる場合に、null の `Subject` を渡すことができます。JAAS ログイン・モジュールのログイン・メソッドは、開始ユーザーに基づいて対応するリソース・プリンシパルを検索し、そのリソース・プリンシパル用の新規 `PasswordCredential` または `GenericCredential` インスタンスを作成する必要があります。リソース・プリンシパルおよび資格証明オブジェクトは、`commit` メソッドの開始 `Subject` に追加されます。リソース資格証明は、リソース・アダプタが提供する `javax.resource.spi.ManagedConnectionFactory` 実装の `createManagedConnection` メソッドに渡されます。null の `Subject` が渡された場合、JAAS ログイン・モジュールはリソース・プリンシパルと適切な資格証明を含む新規 `javax.security.auth.Subject` を作成する必要があります。

InitiatingPrincipal クラスと InitiatingGroup クラス

クラス `oracle.j2ee.connector.InitiatingPrincipal` および `oracle.j2ee.connector.InitiatingGroup` は、JAAS ログイン・モジュールに対して Oracle Application Server Containers for J2EE ユーザーを表します。Oracle Application Server Containers for J2EE は `oracle.j2ee.connector.InitiatingPrincipal` のインスタンスを作成し、それをログイン・モジュールの `initialize` メソッドに渡されるサブジェクトに取り込みます。`oracle.j2ee.connector.InitiatingPrincipal` クラスでは、`java.security.Principal` インタフェースが実装され、メソッド `getGroups()` が追加されます。

```
/**
 * Returns a Set of groups (or roles in JAZN terminology) that this
 * principal is a member of.
 *
 * @return A set of InitiatingGroup objects representing the groups
 *      that this pricipal belongs to.
 */
public Set getGroups()
```

getGroups メソッドは、この Oracle Application Server Containers for J2EE ユーザーについて Oracle Application Server Containers for J2EE のグループまたは JAZN のロールを表す oracle.j2ee.connector.InitiatingGroup の java.util.Set を戻します。グループ・メンバーシップは、ユーザー・マネージャに応じて、principals.xml や jazn-data.xml など、Oracle Application Server Containers for J2EE 固有のディスクリプタ・ファイルに定義されます。oracle.j2ee.connector.InitiatingGroup クラスでは、java.security.Principal インタフェースが実装されますが実行はされません。

ログイン・モジュールでは、getGroups() を使用して Oracle Application Server Containers for J2EE のグループと EIS ユーザーとの間のマッピングを提供できます。このマッピングは、java.security.Principal インタフェースのメソッドによりサポートされます。Oracle Application Server Containers for J2EE のグループと EIS ユーザーの間のマッピングを提供しない場合は、ログイン・モジュールで oracle.j2ee.connector.InitiatingPrincipal クラスと oracle.j2ee.connector.InitiatingGroup クラスを参照する必要はありません。

JAAS と <connector-factory> 要素

oc4j-ra.xml の各 <connector-factory> 要素では、異なる JAAS ログイン・モジュールを指定できます。<jaas-module> 要素内でコネクタ・ファクトリ構成名を指定します。次の例に、コンテナ管理のサインオンに JAAS ログイン・モジュールを使用する oc4j-ra.xml の <connector-factory> 要素を示します。

```
  <connector-factory connector-name="myBlackbox" location="eis/myEIS1">
    <description>Connection to my EIS</description>
    <config-property name="connectionURL"
value="jdbc:oracle:thin:@localhost:5521:orcl" />
    <security-config>
      <jaas-module>
        <jaas-application-name>JAASModuleDemo</jaas-application-name>
      </jaas-module>
    </security-config>
  </connector-factory>
```

JAAS では、特定のアプリケーションに使用する LoginModule と各 LoginModule の起動順序を指定する必要があります。JAAS では、<jaas-application-name> 要素に指定した値を使用して LoginModule が検索されます。

ユーザー作成の認証クラス

Oracle Application Server Containers for J2EE には、プリンシパル・マッピング用に oracle.j2ee.connector.PrincipalMapping インタフェースが用意されています。

```
package oracle.j2ee.connector;

public interface PrincipalMapping
{
/**
* Initializes the various settings for the PrincipalMapping implementation class.
* Implementation class may use the properties for setting default user name and
* password, LDAP connect info, or default mapping.
*
* OC4J will pass the properties specified in the <principal-mapping-interface>
* element in oc4j-ra.xml to this method.
*
* @param prop A Properties object containing the set up information required
*             by the implementation class.
*/
  public void init(Properties prop);

/**
* The ManagedConnectionFactory instance that can be used in creating a
   * PasswordCredential.
   *
 * @param mcf The ManagedConnectionFactory instance that is needed when
  *creating a PasswordCredential instance
  */
  public void setManagedConnectionFactory(ManagedConnectionFactory mcf);

  /**
* Passes the authentication mechanism(s) supported by the resource
* adapter to the PrincipalMapping implementation class.
   * The key of the map passed is a String containing the supported mechanism
* type, such as "BasicPassword", or "Kerbv5". The value is a String
  * containig the corresponding credentials interface as declared in ra.xml,
* such as "javax.resource.spi.security.PasswordCredential".
  *
 * The map may contain multiple elements if the resource adatper supports
  * multiple authentication mechanisms.
   *
 * @param authMechanisms The authentication mechanisms and their corresponding
   *  credentials intereface supported by the resource adapter
  */
  public void setAuthenticationMechanisms(Map authMechanisms);

    /**
```

```
* This is the method that performs the principal mapping. An application user
   * subject is passed, and the implemetation of this method should return
   * a subject for use by the resource adapter to log in to the EIS resource
* per the Connector specifications.
   *
 * OC4J will only called this method for container-managed sign on.
*
 * @param initiatingSubject A Subject containing the application server logged
   *     in principals and public credentials.
     *
    * @return A Subject for use by resource adapter to log in to the remote EIS.
    *      It may return null if the proper resource principal cannot be determined.
  */
  public Subject mapping(Subject initiatingSubject);
}
```

mappingメソッドは、リソース・プリンシパルと資格証明を含むSubjectを戻す必要があります。戻されるSubjectは、Connector Architecture 1.0仕様のセクション8.2.6のオプションAまたはオプションBに準拠する必要があります。Oracle Application Server Containers for J2EEは、開始ユーザーとしてinitiatingPrincipalを指定してmappingメソッドを起動します。

また、Oracle Application Server Containers for J2EEには抽象クラスoracle.j2ee.connector.AbstractPrincipalMappingも用意されています。このクラスには、setManagedConnectionFactory()およびsetAuthenticationMechanism()メソッドのデフォルト実装と、リソース・アダプタで認証方式としてBasicPasswordとKerberosバージョン5（Kerbv5）のどちらがサポートされるかを判別するユーティリティ・メソッド、およびアプリケーション・サーバー・ユーザーのSubjectからPrincipalを抽出するメソッドがあります。開発者は、oracle.j2ee.connector.AbstractPrincipalMappingクラスを拡張して単にinitおよびmappingメソッドを実装します。

次に、oracle.j2ee.connector.AbstractPrincipalMappingクラスに用意されているユーティリティ・メソッドを示します。

```
/**
   *  Utility method provided by this abstract class to return
  *  the ManagedConnectionFactory instance for use to create a
    *  PasswordCredentials object
   *
    *  @return The ManagedConnectionFactory instance that is needed when
    *    creating a PasswordCredential instance
    */
  public ManagedConnectionFactory getManagedConnectionFactory()

   /**
   * Utility method provided by this abstract class to return the Map
```

```
 * of all authentication mechanisms supported by this resource adapter.
* The key of the map passed is a String containing the supported mechanism
 * type, such as "BasicPassword", or "Kerbv5". The value is a String
* containig the corresponding credentials interface as declared in ra.xml,
 * such as "javax.resource.spi.security.PasswordCredential".
*
  * @return The authentication mechanisms and their corresponding
    *      credentials intereface supported by the resource adpater
 */
  public Map getAuthenticationMechanisms()

   /**
 * Utility method provided by this abstract class to return whether
 * BasicPassword authention mechanism is supported by this resource
 * adapter.
 *
  * @return true if BasicPassword authentication mechanism is supported
 *      by the resource adapter, false otherwise.
*/
  public boolean isBasicPasswordSupported()

  /**
 * Utility method provided by this abstract class to return whether
 * Kerbv5 authention mechanism is supported by this resource
* adapter.
 *
  * @return true if Kerbv5 authentication mechanism is supported
*      by the resource adapter, false otherwise.
*/
  public boolean isKerbv5Supported()

 /**
 * Utility method provided by this abstract class to extract the
    * Principal object from the given application server user subject
  * passed from OC4J.
*
  * @param subject The application server user subject passed from
   *       OC4J.
*
  * @return The principal extracted from the given subject
   */
  public Principal getPrincipal(Subject subject)
```

実装クラスの作成後に、そのクラスを含むJARファイルを、解凍済RARファイルを含むディレクトリにコピーします。このディレクトリは、通常は*OC4J_HOME*/applications/application_name/*rar-name*です。ファイルをコピーした後、新規クラス用の<principal-mapping-interface>要素を含むようにoc4j-ra.xmlを編集します。詳細は、13-2ページの「<security-config>要素」を参照してください。

AbstractPrincipalMappingの拡張

次の単純な例に、oracle.j2ee.connector.AbstractPrincipalMapping抽象クラスを拡張して、ユーザーを常にデフォルトのユーザーとパスワードにマップするプリンシパル・マッピングを提供する方法を示します。oc4j-ra.xml内で<principal-mapping-interface>要素の下のプロパティを使用して、デフォルトのユーザーとパスワードを指定します。

PrincipalMappingクラスはMyMappingです。このクラスの定義を次に示します。

```
package com.acme.app;

import java.util.*;
import javax.resource.spi.*;
import javax.resource.spi.security.*;
import oracle.j2ee.connector.AbstractPrincipalMapping;
import javax.security.auth.*;
import java.security.*;

public class MyMapping extends AbstractPrincipalMapping
{
  String m_defaultUser;

   String m_defaultPassword;

    public void init(Properties prop)
    {
      if (prop != null)
      {
           // Retrieves the default user and password from the properties
         m_defaultUser = prop.getProperty("user");
           m_defaultPassword = prop.getProperty("password");
      }
    }

    public Subject mapping(Subject initiatingSubject)
    {
      // This implementation only supporst BasicPassword authentication
      // mechanism. Return if the resource adapter does not support it.
      if (!isBasicPasswordSupported())
            return null;
```

```
      // Use the utility method to retrieve the Principal from the
      // OC4J user. This code is included here only as an example.
      // The principal obtained is not being used in this method.
       Principal principal = getPrincipal(initiatingSubject);

    char[] resPasswordArray = null;
      if (m_defaultPassword != null)
         resPasswordArray = m_defaultPassword.toCharArray();

       // Create a PasswordCredential using the default user name and
      // password, and add it to the Subject per option A in section
    // 8.2.6 in the Connector 1.0 spec.
     PasswordCredential cred = new PasswordCredential(m_defaultUser,
resPasswordArray);
        cred.setManagedConnectionFactory(getManagedConnectionFactory());
       initiatingSubject.getPrivateCredentials().add(cred);
      return initiatingSubject;
    }
}
```

oc4j-ra.xml に、プリンシパル・マッピング・メカニズム用の com.acme.app.MyMapping を指定する <principal-mapping-interface> エントリを追加します。

```
  <connector-factory name="..." location="...">
    ...
    <security-config>
   <principal-mapping-interface>
     <impl-class>com.acme.app.MyMapping</impl-class>
     <property name="user" value="scott" />
     <property name="password" value="tiger" />
   </principal-mapping-interface>
    </security-config>
    ...
  </connector-factory>
```

oc4j-ra.xml の変更

必要な場合は、oc4j-ra.xml ファイル内でデフォルトのプリンシパル・マッピングを作成できます。デフォルトのプリンシパル・マッピング・メカニズムを使用するには、<security-config> 要素の <principal-mapping-entries> サブ要素を使用します。構文の詳細は、13-2 ページの「<security-config> 要素」を参照してください。

<default-mapping> 要素を使用してデフォルト・リソース・プリンシパルのユーザー名とパスワードを指定します。このプリンシパルは、現行の開始プリンシパルに対応する開始ユーザーを含む <principal-mapping-entry> 要素がない場合に、EIS へのログオンに使用されます。デフォルト・マッピングを指定しなければ、Oracle Application Server Containers for J2EE では、リソース・アダプタにデフォルトが受け入れられるものと想定され、デプロイメント・ディスクリプタ（ra.xml または oc4j-ra.xml 内）からの構成プロパティ UserName および Password の値が使用されます。構成プロパティもデフォルト・マッピングも指定しなければ、Oracle Application Server Containers for J2EE では EIS にログインできません。

各 <principal-mapping-entry> 要素には、開始プリンシパルからリソース・プリンシパルとパスワードへのマッピングが含まれます。

たとえば、Oracle Application Server Containers for J2EE のプリンシパル scott をユーザー名 scott、パスワード tiger を使用して myEIS1 という特定の EIS にログインさせ、他のすべての Oracle Application Server Containers for J2EE ユーザーはユーザー名 guest、パスワード guestpw を使用して EIS にログインさせる必要がある場合、oc4j-ra.xml の <connector-factory> 要素は次のようになります。

```
<connector-factory name="..." location="...">
    ...
  <security-config>
    <principal-mapping-entries>
      <default-mapping>
        <res-user>guest</res-user>
        <res-password>guestpw</res-password>
      </default-mapping>
      <principal-mapping-entry>
        <initiating-user>scott</initiating-user>
        <res-user>scott</res-user>
        <res-password>tiger</res-password>
      </principal-mapping-entry>
    </principal-mapping-entries>
  </security-config>
    ...
</connector-factory>
```

14

CSIv2 の構成

Oracle Application Server Containers for J2EE では、Common Secure Interoperability Version 2 プロトコル（CSIv2）がサポートされます。CSIv2 では、様々な準拠レベルが指定されています。Oracle Application Server Containers for J2EE は、準拠レベル 0 を必要とする EJB 仕様に準拠しています。

この章には、次の項目が含まれます。

注意： アプリケーションで JAAS を使用する場合は、CSIv2 を使用するように JAAS Provider を構成する必要があります。詳細は、表 3-8「RealmLoginModule のオプション」を参照してください。

CSIv2 セキュリティ・プロパティの概要

Common Secure Interoperability version 2（CSIv2）は、認可と ID の委任をサポートするセキュアで相互運用可能なワイヤ・プロトコルに関する Object Management Group（OMG）規格です。CSIv2 のプロパティは次の 3 箇所で構成します。

- `internal_settings.xml`
- `orion-ejb-jar.xml`
- `ejb_sec.properties`

この 3 つの構成ファイルについては、14-5 ページの「internal-settings.xml 内の CSIv2 のセキュリティ・プロパティ」、14-7 ページの「ejb_sec.properties 内の EJB クライアントのセキュリティ・プロパティ」、14-7 ページの「orion-ejb-jar.xml 内の CSIv2 のセキュリティ・プロパティ」および 14-9 ページの「ejb_sec.properties 内の EJB クライアントのセキュリティ・プロパティ」を参照してください。

internal-settings.xml 内の EJB サーバーのセキュリティ・プロパティ

サーバーのセキュリティ・プロパティは `internal-settings.xml` 内で指定します。

注意： `internal-settings.xml` は Enterprise Manager では編集できません。

このファイルでは、`<sep-property>` エンティティ内の値として特定のプロパティを指定します。表 14-1「EJB サーバーのセキュリティ・プロパティ」にプロパティのリストを示します。

この表は、鍵と証明書の格納に使用するキーストアとトラストストアのファイルを指しています。各ファイルには、JDK 仕様の書式である Java Key Store（JKS）が使用されます。キーストアには、秘密鍵と証明書のマップが格納されます。トラストストアには、認証局（CA。VeriSign 社および Thawte 社など）の信頼できる証明書が格納されます。

表 14-1 EJB サーバーのセキュリティ・プロパティ

プロパティ	意味
`port`	IIOP ポート番号（デフォルトは `5555`）。
`ssl`	IIOP/SSL がサポートされる場合は `true`、それ以外の場合は `false`。

表 14-1 EJB サーバーのセキュリティ・プロパティ（続き）

プロパティ	意味
ssl-port	IIOP/SSL のポート番号（デフォルトは 5556）。このポートはサーバー・サイド認証にのみ使用されます。アプリケーションでクライアントおよびサーバー認証を使用する場合は、ssl-client-server-auth-port を設定する必要があります。
ssl-client-server-auth-port	クライアントおよびサーバー認証に使用されるポート（デフォルトは 5557）。これは、OC4J によりクライアントとサーバー両方の認証に必要な SSL 接続がリスニングされるポートです。このプロパティを設定しないと、OC4J ではクライアント・サイド認証が ssl-port + 1 でリスニングされます。
keystore	キーストア名（ssl が true の場合にのみ使用）。
keystore-password	キーストアのパスワード（ssl が true の場合にのみ使用）。
trusted-clients	ID アサーションを信頼できるホストのカンマ区切りのリスト。このリストの各エントリには、IP アドレス、ホスト名、ホスト名のパターン（*.example.com など）を指定するか、または * を指定できます。単独の * は、すべてのクライアントが信頼できることを意味します。デフォルトでは、信頼できるクライアントはありません。
truststore	トラストストア名。サーバーのトラストストアを指定しないと、OC4J ではトラストストアとしてキーストアが使用されます（ssl が true の場合にのみ使用）。
truststore-password	トラストストアのパスワード（ssl が true の場合にのみ設定可能）。

注意： 表 14-1 のプロパティ keystore-password および truststore-password では、パスワードの間接化がサポートされます。

Oracle Application Server Containers for J2EE が OracleAS（スタンドアロンではなく）環境で Oracle Process Management Notification サービス（OPMN）により起動されると、internal-settings.xml に指定したポートは無視されます。OPMN が特定の Oracle Application Server Containers for J2EE インスタンスの IIOP を無効化するように構成されている場合は、IIOP が（server.xml が指す）internal-settings.xml を介して有効化するように設定されていても、IIOP は有効化されません。

次の例に、標準的な internal-settings.xml ファイルを示します。

```
<server-extension-provider name="IIOP"
        class="com.oracle.iiop.server.IIOPServerExtensionProvider">
    <sep-property name="port" value="5555" />
    <sep-property name="host" value="localhost" />
    <sep-property name="ssl" value="false" />
    <sep-property name="ssl-port" value="5556" />
    <sep-property name="ssl-client-server-auth-port" value="5557" />
    <sep-property name="keystore" value="keystore.jks" />
    <sep-property name="keystore-password" value="123456" />
    <sep-property name="truststore" value="truststore.jks" />
    <sep-property name="truststore-password" value="123456" />
    <sep-property name="trusted-clients" value="*" />
</server-extension-provider>
```

注意： port のデフォルト値は ssl-port のデフォルト値 -1 ですが、この関係は必須ではありません。

次に internal-settings.xml の DTD を示します。

```
<!-- A server extension provider that is to be plugged in to the server.
-->
<!ELEMENT server-extension-provider (sep-property*) (#PCDATA)>
<!ATTLIST server-extension-provider name class CDATA #IMPLIED>
<!ELEMENT sep-property (#PCDATA)>
<!ATTLIST sep-property name value CDATA #IMPLIED>
<!-- This file contains internal server configuration settings. -->
<!ELEMENT internal-settings (server-extension-provider*)>
```

internal-settings.xml 内の CSIv2 のセキュリティ・プロパティ

この項では、internal_settings.xml 内の <sep-property> 要素に設定する値のセマンティックについて説明します。構文の詳細は、14-2 ページの「internal-settings.xml 内の EJB サーバーのセキュリティ・プロパティ」を参照してください。

Oracle Application Server Containers for J2EE で CSIv2 プロトコルを使用するには、ssl を true に設定して IIOP/SSL ポート（ssl-port）を指定する必要があります。

- ssl を true に設定しないと、CSIv2 は有効化されません。ssl を true に設定すると、クライアントとサーバーでは CSIv2 を使用できますが、SSL を使用した通信に必須ではありません。
- ssl-port を指定しないと、orion-ejb-jar.xml 内で <ior-security-config> エンティティを構成していても、サーバーでは IOR に CSIv2 のコンポーネント・タグは挿入されません。

サーバー上で IIOP/SSL が有効化されている場合、Oracle Application Server Containers for J2EE は 2 つの異なるソケットでリスニングします。一方はサーバー認証専用で、他方はサーバーおよびクライアント認証用です。<sep-property> 要素にサーバー認証ポートを指定すると、サーバーおよびクライアント認証リスナーでは、直後のポート番号が使用されます。

サーバー認証のみを使用する SSL クライアントの場合は、次のいずれかを指定できます。

- トラストストアのみ
- キーストアとトラストストアの両方
- なし

キーストアもトラストストアも指定しないと、セキュリティ・プロバイダによりデフォルトのトラストストアが設定されていない場合にハンドシェイクが失敗する可能性があります。

クライアント・サイド認証を使用する SSL クライアントは、キーストアとトラストストアの両方を指定する必要があります。クライアント認証には、キーストアからの証明書が使用されます。

ejb_sec.properties 内の CSIv2 のセキュリティ・プロパティ

クライアントがクライアント・サイド SSL 認証を使用しない場合は、クライアント・ランタイムがセキュリティ・コンテキストを挿入してユーザー名とパスワードを送信できるように、`ejb_sec.properties` ファイル内で `client.sendpassword` を設定する必要があります。また、サーバーを含むように `server.trustedhosts` を設定する必要があります。

注意： サーバー・サイド認証は、ユーザー名とパスワードよりも優先されます。

クライアントがクライアント・サイド SSL 認証を使用する場合、サーバーはクライアントの証明書から `DistinguishedName` を抽出し、それを対応するユーザー・マネージャ内で検索します。パスワード認証は実行されません。

信頼関係

次の 2 種類の信頼関係が存在します。

- サーバーが非 SSL 接続を使用してユーザー名とパスワードを送信することを、クライアントが信頼している場合
- クライアントが発信側クライアントの識別を委任する ID アサーションを送信することを、サーバーが信頼している場合

クライアントは、EJB プロパティ `oc4j.iiop.trustedServers` にトラステッド・サーバーをリストします。詳細は、14-9 ページの表 14-2「EJB クライアントのセキュリティ・プロパティ」を参照してください。サーバーは、`internal-settings.xml` 内の `<sep-property>` 要素の `trusted-client` プロパティにトラステッド・クライアントをリストします。詳細は、14-2 ページの「internal-settings.xml 内の EJB サーバーのセキュリティ・プロパティ」を参照してください。

EJB 規格の準拠レベル 0 では、信頼関係について次の 2 つの処理方法が定義されています。

- 推定による信頼。サーバーは、論理クライアントがサーバーに対して自己認証を行わず、接続が安全でない場合も、論理クライアントが信頼できると推定します。
- 認証済の信頼。ターゲットは、トランスポート・レベルまたは `trusted-client` リスト、あるいはその両方での認証に基づいて中間サーバーを信頼します。

注意： SSL クライアント・サイド認証を要求するようにサーバーを構成し、ID アサーションの挿入が許可されているトラステッド・クライアントの（または中間）ホストのリストも指定できます。

Oracle Application Server Containers for J2EE では両方の種類の信頼が提供されますが、信頼関係は orion-ejb-jar.xml 内の Bean の <ior-security-config> 要素を使用して構成します。詳細は、14-7 ページの「orion-ejb-jar.xml 内の CSIv2 のセキュリティ・プロパティ」を参照してください。

orion-ejb-jar.xml 内の CSIv2 のセキュリティ・プロパティ

この項では、EJB 用の CSIv2 セキュリティ・プロパティについて説明します。各 Bean の CSIv2 セキュリティ・ポリシーを orion-ejb-jar.xml 内で個別に構成します。CSIv2 セキュリティ・プロパティは <ior-security-config> 要素内で指定します。各要素には <transport-config> 要素、<as-context> 要素および <sas-context> 要素が含まれます。

<transport-config> 要素

この要素には、トランスポートのセキュリティ・レベルを指定します。<transport-config> 内の各要素は、supported、required または none に設定する必要があります。None は Bean でその機能がサポートも使用もされないことを意味し、supports は Bean でクライアントによる機能の使用が許可されることを意味し、required は Bean でクライアントによる機能の使用が強制されることを意味します。各要素を次に示します。

- <integrity>: すべての送信が発信時のまま正確に受信されるという保証があるかどうか。
- <confidentiality>: 第三者が送信内容を読み取れないという保証があるかどうか。
- <establish-trust-in-target>: サーバーがクライアントに対して自己認証を行うかどうか。
- <establish-trust-in-client>: クライアントがサーバーに対して自己認証を行うかどうか。

注意： <establish-trust-in-client> を required に設定すると、<as-context> 内の username_password の指定は無視されます。この場合は、<as-context> セクションの <required> ノード値も false に設定しないと、アクセス権の問題が発生します。

<transport-config> プロパティを required に設定すると、Bean では通信に RMI/IIOP/SSL が使用されます。

<as-context> 要素

この要素では、メッセージレベルの認証プロパティを指定します。

- <auth-method>: username_password または none に設定する必要があります。username_password に設定すると、Bean ではコール元の認証にユーザー名とパスワードが使用されます。
- <realm>: このリリースでは default に設定する必要があります。
- <required>: true に設定すると、Bean はコール元に対してユーザー名とパスワードの指定を要求します。

<sas-context> 要素

この要素では、ID 委任プロパティを指定します。この要素には 1 つの要素 <caller-propagation> があり、supported、required または none に設定できます。<caller-propagation> 要素を supported に設定すると、この Bean は中間サーバーから委任された ID を受け入れます。required に設定すると、この Bean は他のすべての Bean に対して委任された ID の送信を要求します。none に設定すると、この Bean は ID の委任をサポートしません。

次に例を示します。

```
<ior-security-config>
  <transport-config>
    <integrity>supported</integrity>
    <confidentiality>supported</confidentiality>
    <establish-trust-in-target>supported</establish-trust-in-target>
    <establish-trust-in-client>supported</establish-trust-in-client>
  </transport-config>
  <as-context>
    <auth-method>username_password</auth-method>
    <realm>default</realm>
    <required>true</required>
  </as-context>
  <sas-context>
    <caller-propagation>supported</caller-propagation>
  </sas-context>
</ior-security-config>
```

DTD

次に <ior-security-config> 要素の DTD を示します。

```
<!ELEMENT ior-security-config (transport-config?, as-context?
sas-context?) >
<!ELEMENT transport-config (integrity, confidentiality,
establish-trust-in-target, establish-trust-in-client) >
<!ELEMENT as-context (auth-method, realm, required) >
<!ELEMENT sas-context (caller-propagation) >
<!ELEMENT integrity (#PCDATA) >
<!ELEMENT confidentiality (#PCDATA)>
<!ELEMENT establish-trust-in-target (#PCDATA) >
<!ELEMENT establish-trust-in-client (#PCDATA) >
<!ELEMENT auth-method (#PCDATA) >
<!ELEMENT realm (#PCDATA) >
<!ELEMENT required (#PCDATA)> <!-- Must be true or false -->
<!ELEMENT caller-propagation (#PCDATA) >
```

ejb_sec.properties 内の EJB クライアントのセキュリティ・プロパティ

クライアントは、サーバー内で実行されるかどうかに関係なく、EJB セキュリティ・プロパティを持ちます。表 14-2 に、ejb_sec.properties ファイルにより制御される EJB クライアントのセキュリティ・プロパティを示します。デフォルトでは、Oracle Application Server Containers for J2EE は、クライアントとして実行される場合はこのファイルを現行ディレクトリ内で検索し、サーバー内で実行される場合は J2EE_HOME/config 内で検索します。このファイルの場所は、-Dejb_sec_properties_location=*pathname* で明示的に指定できます。

表 14-2 EJB クライアントのセキュリティ・プロパティ

プロパティ	意味
# oc4j.iiop.keyStoreLoc	キーストアのパス名。
# oc4j.iiop.keyStorePass	キーストアのパスワード。
# oc4j.iiop.trustStoreLoc	トラストストアのパス名。
# oc4j.iiop.trustStorePass	トラストストアのパスワード。
# oc4j.iiop.enable.clientauth	クライアントでクライアント・サイド認証がサポートされるかどうか。このプロパティを true に設定した場合は、キーストアの場所とパスワードを指定する必要があります。

表 14-2 EJB クライアントのセキュリティ・プロパティ（続き）

プロパティ	意味
oc4j.iiop.ciphersuites	有効化する Cipher Suite。有効な Cipher Suite は次のとおりです。 TLS_RSA_WITH_RC4_128_MD5 SSL_RSA_WITH_RC4_128_MD5 TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA TLS_RSA_EXPORT_WITH_RC4_40_MD5 SSL_RSA_EXPORT_WITH_RC4_40_MD5 TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
nameservice.useSSL	サーバーへの初期接続時に SSL を使用するかどうか。
client.sendpassword	SSL を使用しない場合に、サービス・コンテキスト内でユーザー名とパスワードをクリア（非暗号化）形式で送信するかどうか。このプロパティを true に設定すると、ユーザー名とパスワードは trustedServer リストに含まれるサーバーにのみ送信されます。
oc4j.iiop.trustedServers	クリアテキスト形式で送信されるパスワードの受信について信頼できるサーバーのリスト。client.sendpassword が false に設定されている場合は、このプロパティを指定しても効果はありません。リストはカンマ区切りです。このリストの各エントリには、IP アドレス、ホスト名、ホスト名のパターン（*.example.com など）を指定するか、または * を指定できます。単独の * は、すべてのサーバーが信頼できることを意味します。

注意： # でマークされているプロパティは、ejb_sec.properties 内で、またはシステム・プロパティとして設定できます。ejb_sec.properties での設定は、システム・プロパティとして指定した設定よりも常に優先されます。

15

セキュリティのヒント

この章で説明するヒントは、Oracle Technology Network Japan（http://otn.oracle.co.jp）からダウンロード可能な Best Practices のドキュメントから抜粋したものです。更新情報の有無を OTN-J の Web サイトでチェックしてください。

HTTPS

Oracle HTTP Server（OHS）には、アプリケーションを変更せずにアプリケーションにセキュリティを提供できるように複数の機能が用意されています。類似の機能をコーディングする前に、これらの機能を評価した上で利用してください。次の HTTP セキュリティ機能があります。

- 認証 : OHS では、標準的な方法でユーザーを認証して認証済ユーザー ID をアプリケーションに渡すことができます（REMOTE_USER）。また、シングル・サインオンもサポートされるため、既存のログイン・メカニズムが再利用されます。
- 認可 : OHS には、エンド・ユーザーが認証と認可を受けている場合にのみアプリケーションへのアクセスを許可できるディレクティブがあります。この機能についてもコード変更は不要です。
- 暗号化 : OHS は、アプリケーションのコード変更なしにエンド・ユーザーに透過的 SSL 通信を提供できます。

その他、HTTPS の保護に関して次の提案事項があります。

- 弱い暗号化の使用が失敗するように Oracle Application Server を構成します。Oracle Application Server は、指定した暗号化のみを HTTPS 接続に使用するように構成できます。たとえば、アプリケーションでは、非 128 ビット・クライアント・サイド SSL ライブラリからの接続を拒否できます。この機能は、各接続の暗号化の強度をサーバー・サイドで制御できるため、銀行や他の金融機関に特に役立ちます。
- SSL を介した HTTP を加速するために、HTTPS/HTTP アプライアンスを使用します。必要なすべての場所で HTTPS を使用します。ただし、HTTPS では極端に大きいパフォーマンス上のオーバーヘッドが発生するため、状況によってはトレードオフが必要になります。

  比較的に見て低コストの場合は、HTTPS/HTTP アプライアンスにより 500 MHz UNIX マシン上でのスループットが毎秒 20 ～ 30 トランザクションから毎秒 6000 トランザクションへと変化する場合があり、これによりトレードオフの決定が容易になります。

  さらに、これらのアプライアンスは、UNIX、Windows または Linux マシンに数学または暗号カードを追加するよりも優れたソリューションを提供します。

  逐次的な HTTPS 送信が同じ Web サーバーを介して要求されることを確認します。500 MHz のマシン上では、SSL セッションの開始に 40 ～ 50 ミリ秒の CPU タイムを予想します。この CPU タイムのほとんどは鍵交換ロジックに費やされ、その間にバルク暗号鍵が交換されます。アクセスが同じ Web サーバーにルーティングされる場合は、バルク暗号化をキャッシュすると以降のアクセス時に CPU オーバーヘッドが大幅に減少します。
- セキュアなページとセキュリティ不要なページは別個のサーバーに保管します。アプリケーションのページすべてを 1 つの HTTPS サーバーに置く方が容易ですが、この方法は莫大なパフォーマンス・コストを伴います。SSL を必要とするページ用に HTTPS サーバーを予約し、SSL を必要としないページは HTTP サーバーに置きます。

セキュアなページが同じ画面に表示される多数の GIF、JPEG または他のファイルで構成されている場合、セキュアなコンテンツをセキュアでない静的コンテンツから分離するだけの価値はないと思われます。SSL 鍵交換（CPU サイクルの主要コンシューマ）はコールされるのが常に一回のみなので、バルク暗号化のオーバーヘッドはそれほど大きくありません。

全体的なセキュリティ

- モジュールに権限を割り当てるときには、そのモジュールの機能の実行に必要な最下位レベルを使用します。下位レベルの権限を使用すると、「障害封じ込め」が提供されます。セキュリティが危険にさらされても、それはネットワークの小さい領域に封じ込められ、インターネット全体に侵入することはできません。
- SSL を使用する場合は、SSLSessionCacheTimeout ディレクティブをチューニングします。OracleAS の Apache サーバーは、デフォルトでクライアントの SSL セッション情報をキャッシュします。セッション・キャッシングを使用すると、サーバーへの最初の接続にのみ大きな遅延が発生します。

  SSL 対応サーバーへの単純な接続および切断テストでは、5 接続の場合の経過時間は、SSL セッション・キャッシングなしで 11.4 秒、セッション・キャッシングが有効化されている場合は 1.9 秒でした。

  デフォルトの `SSLSessionCacheTimeout` は 300 秒です。SSL セッションの継続時間は、HTTP 固定接続の使用には無関係であることに注意してください。`httpd.conf` ファイル内の `SSLSessionCacheTimeout` ディレクティブは、アプリケーションのニーズにあわせて変更できます。

JAAS

- ユーザー管理を principals.xml から JAAS Provider に移行します。以前のリリースの OracleAS の場合、J2EE アプリケーション・サーバー・コンポーネントでは、すべてのユーザー情報がファイル `principals.xml` に格納されていました（クリアテキストによるパスワードの格納を含みます）。OracleAS の JAAS Provider には同様の単純なセキュリティ・モデルがデフォルトとして用意されており、クリアテキストによるパスワードは格納されません。また、JAAS Provider は、出荷時の OracleAS Infrastructure（SSO と OID を含む）との密接な統合を提供します。
- カスタムのユーザー・マネージャを記述するのは避け、JAAS Provider、SSO および OID を拡張します。Oracle Application Server Containers for J2EE（OC4J）のコンテナは、引き続き複数のメソッドとセキュリティ・プロバイダ拡張レベルを提供します。`UserManager` クラスを拡張してカスタム・ユーザー・マネージャを作成できますが、JAAS Provider、SSO および OID の豊富な機能を利用すると、インフラストラクチャ・コードではなく実際のビジネス・ロジックに専念できます。SSO と OID の両方に、それぞれ外部認証サーバーおよびディレクトリと統合するための API が用意されています。

- JAAS Provider での認証メカニズムとして SSO を使用します。JAAS Provider を使用すると、様々な認証オプションから選択できます。ただし、次の理由から、できるだけ SSO Server を使用することをお薦めします。
  - Portal、Forms、Reports、Wireless など、ほとんどの OracleAS コンポーネントのデフォルト・メカニズムです。
  - 宣言を使用して容易にセットアップでき、カスタム・プログラミングを必要としません。
  - シームレスな PKI 統合を提供します。
- JAAS Provider の宣言機能を使用してプログラミング作業を軽減します。OracleAS の JAAS Provider の機能のほとんどは、特に認証の領域において宣言で制御されるため、開発者はデプロイメント時までセットアップを延期できます。これにより、JAAS ベースのアプリケーションの統合に必要なプログラミング・タスクが減少するのみでなく、デプロイヤはそのアプリケーションに環境固有のセキュリティ・モデルを使用できます。
- JAAS Provider と Java パーミッション・モデルが提供するファイングレイン・アクセス制御を使用します。従来の「粗密」な J2EE 認可モデルとは異なり、OC4J と統合されている JAAS Provider では、Java のパーミッションを使用して任意の保護リソースのモデルを作成できます。Java パーミッション・モデル（および関連 Permission クラス）は拡張可能であり、ファイングレイン・アクセス制御を柔軟に定義できます。
- 本番環境では、JAAS Provider の中央リポジトリとして OID を使用します。JAAS Provider では、開発およびテスト環境に役立つフラットファイル XML ベースのリポジトリがサポートされますが、本番環境では OID を使用するように構成する必要があります。OID には、管理メタデータのモデル作成用に LDAP 標準機能が用意されており、Oracle データベース・プラットフォーム上に構築されています。また、スケーラビリティ、信頼性、管理性およびパフォーマンスについてデータベース・プロパティをすべて継承します。
- JAAS Provider の認可機能を利用します。OracleAS の JAAS Provider では、JAAS 1.0 に定義されている認可機能に加えて次の機能がサポートされます。
  - 階層形式によるロールベースのアクセス制御（RBAC）
  - セキュリティ・ポリシーをサブスクライバ（つまり各ユーザー・コミュニティ）別にパーティション化する機能

  これらの拡張機能は、大規模なユーザー・コミュニティを対象とするセキュリティ・ポリシーに、よりスケーラブルで管理しやすいフレームワークを提供します。

A

JAAS Provider の標準とサンプル

この付録には、補足的なサンプルと標準が記載されています。

この付録には、次の項目が含まれます。

サンプル jazn-data.xml コード

この項には、XML ファイルが準拠する必要のある特定の標準を示すサンプル jazn-data.xml ファイルが記載されています。この jazn-data.xml ファイルには、レルム jazn.com、ユーザー（不明瞭化されたパスワードを使用する 2 人のユーザー）およびロールが含まれています。

関連項目：

- 4-5 ページ「XML ベース環境でのレルム管理」
- XML ベース・プロバイダ環境で JAAS Provider を管理する方法の詳細は、4-3 ページの「レルムとポリシーの管理」を参照してください。

例 A-1 サンプル jazn-data.xml ファイル

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<!DOCTYPE jazn-data PUBLIC "JAZN-XML Data"
"http://xmlns.oracle.com/ias/dtds/jazn-data.dtd">
<jazn-data>


<!-- JAZN Realm Data -->
<jazn-realm>
  <realm>
    <name>jazn.com</name>
    <users>
      <user>
        <name>SCOTT</name>
        <display-name>SCOTT</display-name>
        <credentials>{903}oZZYqmGc/iyCaDrD4qs2FHbXf3LAWtMN</credentials>
      </user>
      <user>
        <name>admin</name>
        <display-name>OC4J Administrator</display-name>
        <description>OC4J Administrator</description>
        <credentials>{903}FVb95KHGyzR9MkAS2Ru/72P/Ol6eOsQD</credentials>
      </user>
      <user>
        <name>anonymous</name>
        <description>The default guest/anonymous user</description>
      </user>
      <user>
        <name>pwForScott</name>
        <description>Password for database user Scott</description>
        <credentials>{903}pjbjHNP53w3haB3ygstBpsglEhQJ1dnN</credentials>
      </user>
      <user>
```

```
        <name>user</name>
        <description>The default user</description>
        <credentials>{903}Zg4KSjPqwZ6FGsCWbxiFSJpPFJNrq9Ww</credentials>
      </user>
      <user>
        <name>pwForSSL</name>
        <description>Password for ssl key and trust stores</description>
        <credentials>{903}uMg+4/e5znCrcQSH36NjbrkpHdgC6oMh</credentials>
      </user>
      <user>
        <name>pwForSystem</name>
        <description>Password for database system user </description>
        <credentials>{903}IUHuvYYGY5R9trDfQp7qY//livlqHjVV</credentials>
      </user>
    </users>
    <roles>
      <role>
        <name>administrators</name>
        <display-name>Realm Admin Role</display-name>
        <description>Administrative role for this realm.</description>
        <members>
          <member>
            <type>user</type>
            <name>admin</name>
          </member>
        </members>
      </role>
      <role>
        <name>jmxusers</name>
        <display-name>JMX users</display-name>
        <description>Allows access to application level user defined
MBeans</description>
        <members>
        </members>
      </role>
      <role>
        <name>users</name>
        <members>
          <member>
            <type>user</type>
            <name>user</name>
          </member>
          <member>
            <type>user</type>
            <name>SCOTT</name>
          </member>
          <member>
```

```
            <type>role</type>
            <name>administrators</name>
          </member>
        </members>
      </role>
      <role>
        <name>guests</name>
        <members>
          <member>
            <type>user</type>
            <name>anonymous</name>
          </member>
          <member>
            <type>role</type>
            <name>users</name>
          </member>
        </members>
      </role>
    </roles>
  </realm>
</jazn-realm>


<!-- JAZN Policy Data -->
<jazn-policy>
  <grant>
    <grantee>
      <principals>
        <principal>
          <realm-name>jazn.com</realm-name>
          <type>role</type>
          <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
          <name>jazn.com/jmxusers</name>
        </principal>
      </principals>
    </grantee>
    <permissions>
      <permission>
        <class>com.evermind.server.rmi.RMIPermission</class>
        <name>login</name>
      </permission>
    </permissions>
  </grant>
  <grant>
    <grantee>
      <principals>
        <principal>
```

```
        <realm-name>jazn.com</realm-name>
        <type>role</type>
        <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
        <name>jazn.com/users</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>com.evermind.server.rmi.RMIPermission</class>
      <name>login</name>
    </permission>
  </permissions>
</grant>
<grant>
  <grantee>
    <principals>
      <principal>
        <realm-name>jazn.com</realm-name>
        <type>role</type>
        <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
        <name>jazn.com/administrators</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jazn.policy.AdminPermission</class>
      <name>oracle.security.jazn.realm.RealmPermission$jazn.com$createrealm</name>
    </permission>
    <permission>
      <class>oracle.security.jazn.realm.RealmPermission</class>
      <name>jazn.com</name>
      <actions>dropuser</actions>
    </permission>
    <permission>
      <class>com.evermind.server.AdministrationPermission</class>
      <name>administration</name>
      <actions>administration</actions>
    </permission>
    <permission>
      <class>oracle.security.jazn.realm.RealmPermission</class>
      <name>jazn.com</name>
      <actions>modifyrealmmetadata</actions>
    </permission>
    <permission>
      <class>com.evermind.server.rmi.RMIPermission</class>
```

```
        <name>login</name>
      </permission>
      <permission>
        <class>oracle.security.jazn.policy.AdminPermission</class>
        <name>oracle.security.jazn.realm.RealmPermission$jazn.com$createrole</name>
      </permission>
      <permission>
        <class>oracle.security.jazn.policy.RoleAdminPermission</class>
        <name>jazn.com/*</name>
      </permission>
      <permission>
        <class>oracle.security.jazn.realm.RealmPermission</class>
        <name>jazn.com</name>
        <actions>createrealm</actions>
      </permission>
      <permission>
        <class>oracle.security.jazn.policy.AdminPermission</class>
        <name>oracle.security.jazn.realm.RealmPermission$jazn.com$droprole</name>
      </permission>
      <permission>
        <class>oracle.security.jazn.policy.AdminPermission</class>
        <name>oracle.security.jazn.realm.RealmPermission$jazn.com$droprealm</name>
      </permission>
      <permission>
        <class>oracle.security.jazn.policy.AdminPermission</class>

<name>oracle.security.jazn.realm.RealmPermission$jazn.com$modifyrealmmetadata</name>
      </permission>
      <permission>
        <class>oracle.security.jazn.realm.RealmPermission</class>
        <name>jazn.com</name>
        <actions>droprealm</actions>
      </permission>
      <permission>
        <class>oracle.security.jazn.policy.AdminPermission</class>
        <name>oracle.security.jazn.policy.RoleAdminPermission$jazn.com/*$</name>
      </permission>
    </permissions>
  </grant>
</jazn-policy>


<!-- Permission Class Data -->
<jazn-permission-classes>
</jazn-permission-classes>
```

```
<!-- Principal Class Data -->
<jazn-principal-classes>
</jazn-principal-classes>


<!-- Login Module Data -->
<jazn-loginconfig>
  <application>
    <name>oracle.security.jazn.tools.Admintool</name>
    <login-modules>
      <login-module>
        <class>oracle.security.jazn.realm.RealmLoginModule</class>
        <control-flag>required</control-flag>
        <options>
          <option>
            <name>debug</name>
            <value>false</value>
          </option>
          <option>
            <name>addAllRoles</name>
            <value>true</value>
          </option>
        </options>
      </login-module>
    </login-modules>
  </application>
  <application>
    <name>oracle.security.jazn.oc4j.JAZNUserManager</name>
    <login-modules>
      <login-module>
        <class>oracle.security.jazn.realm.RealmLoginModule</class>
        <control-flag>required</control-flag>
        <options>
          <option>
            <name>addAllRoles</name>
            <value>true</value>
          </option>
        </options>
      </login-module>
    </login-modules>
  </application>
</jazn-loginconfig>

</jazn-data>
```

補足的サンプル・コード

次のサンプル・コードは、補足情報として記載されています。この項には、次のサンプル・コードが含まれます。

- 補足的サンプル・コード : アプリケーション・レルムの作成
- 補足的サンプル・コード : ユーザーのパーミッションの変更

補足的サンプル・コード : アプリケーション・レルムの作成

次のサンプル・コードでは、表 A-1 に示すオブジェクトを含むアプリケーション・レルムが作成されます。変更対象のオブジェクトは太字で示されています。

表 A-1 アプリケーション・レルム作成サンプル・コード内のオブジェクト

オブジェクト	名前
サンプル組織	dev.com
adminUser (オプション)	John.Singh
adminRole	administrator
サンプル・レルム名	devRealm

例 A-2 アプリケーション・レルム作成コード

```
import oracle.security.jazn.spi.ldap.*;
import oracle.security.jazn.*;
import oracle.security.jazn.realm.*;

import java.util.*;

/**
 * Creates an application realm.
 */

public class CreateRealm extends Object
{
    public CreateRealm() {};

    public static void main (String[] args) {
      CreateRealm test = new CreateRealm();
      test.createAppRealm();
    }

    void createAppRealm() {
```

```
    Realm realm=null;


 try {
     Hashtable prop = new Hashtable();
     prop.put(Realm.LDAPProperty.USERS_SEARCHBASE,"cn=users,o=dev.com");

     // specifying the following LDAP directory object class
       // is optional.  When specified, it will
     // be used as a filter to search for users
     prop.put(Realm.LDAPProperty.USERS_OBJ_CLASS,"orclUser");

     // adminUser is optional
    String adminUser = "John.Singh";

     String adminRole = "administrator";

     RealmManager realmMgr = JAZNContext.getRealmManager();

     InitRealmInfo realmInfo = new
     InitRealmInfo(InitRealmInfo.RealmType.APPLICATION_REALM, adminUser,
     adminRole, prop);
     realm = realmMgr.createRealm("devRealm", realmInfo);
     }

catch (Exception e) {
     e.printStackTrace();
    }
  }
}
```

補足的サンプル・コード : ユーザーのパーミッションの変更

例 A-3 は、ユーザー Jane.Smith に対する java.io.FilePermission の付与を示しています。変更対象のオブジェクトは太字で示されています。

表 A-2 に、例 A-3 のオブジェクトを示します。

表 A-2 ユーザー・パーミッション変更サンプル・コード内のオブジェクト

オブジェクト	名前	コメント
RealmUser user	Jane.Smith	
codesource cs	file:/home/task.jar	
ファイルのパス	report.data	パスはファイルのパス名です。
サンプル組織	abc.com	abc.com は、このコードには直接表示されません。
サンプル外部レルム	abcRealm	

例 A-3 ユーザー・パーミッション変更コード

サンプル・コード

```
import oracle.security.jazn.*;
import oracle.security.jazn.policy.*;
import oracle.security.jazn.realm.*;
import java.lang.*;
import java.security.*;
import java.util.*;
import java.net.*;
import java.io.*;

public class Init {

    public static void main(String[] args) {

    try {
      RealmManager realmMgr = JAZNContext.getRealmManager();
            Realm realm = realmMgr.getRealm("abcRealm");
            UserManager userMgr = realm.getUserManager();
            RoleManager roleMgr = realm.getRoleManager();
            final JAZNPolicy policy = JAZNContext.getPolicy();

            final RealmUser user = userMgr.getUser("Jane.Smith");
```

```
            AccessController.doPrivileged (new PrivilegedAction() {
                    public Object run() {

                try {

                  CodeSource cs = new CodeSource(new URL("
                            file:/home/task.jar"), null);
                     HashSet prop = new HashSet();
                     prop.add((Principal) user);

                     // assign permission to principals
                     policy.grant(new Grantee(prop, cs), new
                              FilePermission("report.data", "read"));

                     return null;
                         } catch (JAZNException e1) {
                             e1.printStackTrace();
                         } catch (java.net.MalformedURLException e2) {
                             e2.printStackTrace();
                         }
                     return null;
                     }
                }
             );

        } catch (JAZNException e) {
            e.printStackTrace();
        }
    }
}
```

サンプル・コードの説明

例 A-3 のサンプル・コードは、サンプル・アプリケーション AccessTest1 を使用するためのパーミッションをユーザー Jane.Smith に次のように付与します。

サンプル・アプリケーション AccessTest1 を含む file:/home/task.jar に、cs という名前が割り当てられます。

```
CodeSource cs = new CodeSource(new URL("
                            file:/home/task.jar"), null);
```

Jane.Smith は、hashset prop に追加されるユーザーです。

```
HashSet prop = new HashSet();
                    prop.add((Principal) user);
```

Jane.Smith には、Codesource cs でファイル report.data を読み取るためのパーミッションが付与されます。

```
policy.grant(new Grantee(prop, cs), new
                              FilePermission("report.data", "read"));
```

B

JAAS Provider のスキーマ

この付録には、JAAS の XML ベース・プロバイダで使用される `jazn-data.xml` ファイルと、両方の JAAS Provider の構成に使用される `jazn.xml` ファイルの、XSD（XML Schema）仕様が記載されています。

jazn-data.xml のスキーマ

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" >
  <xsd:element name="jazn-data" type="JAZN-DATAjazn-dataType"/>
  <xsd:complexType name="JAZN-DATAjazn-dataType">
    <xsd:sequence >
      <xsd:element name="jazn-realm" type="JAZN-DATAjazn-realmType" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="jazn-policy" type="JAZN-DATAjazn-policyType" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="jazn-permission-classes"
type="JAZN-DATAjazn-permission-classesType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="jazn-principal-classes"
type="JAZN-DATAjazn-principal-classesType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="jazn-loginconfig" type="JAZN-DATAjazn-loginconfigType"
minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-realmType">
    <xsd:sequence>
      <xsd:element name="realm" type="JAZN-DATArealmType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATArealmType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="users" type="JAZN-DATAusersType" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="roles" type="JAZN-DATArolesType" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="jazn-policy" type="JAZN-DATAjazn-policyType" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAusersType">
    <xsd:sequence>
      <xsd:element name="user" type="JAZN-DATAuserType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAuserType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="display-name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
```

```
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="credentials" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATArolesType">
    <xsd:sequence>
      <xsd:element name="role" type="JAZN-DATAroleType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAroleType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="display-name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="members" type="JAZN-DATAmembersType" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAmembersType">
    <xsd:sequence>
      <xsd:element name="member" type="JAZN-DATAmemberType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAmemberType">
    <xsd:sequence >
      <xsd:element name="type" type="xsd:string" />
      <xsd:element name="name" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-policyType">
    <xsd:sequence>
      <xsd:element name="grant" type="JAZN-DATAgrantType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>


<!-- no way to define key attribute for grant type -->
  <xsd:complexType name="JAZN-DATAgrantType">
```

```
    <xsd:sequence >
      <xsd:element name="grantee" type="JAZN-DATAgranteeType" />
      <xsd:element name="permissions" type="JAZN-DATApermissionsType" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="grantee-names" type="xsd:string" use="default" value=""/>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed"
value="grantee-names"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAgranteeType">
    <xsd:sequence >
      <xsd:element name="display-name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="principals" type="JAZN-DATAprincipalsType" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="codesource" type="JAZN-DATAcodesourceType" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAprincipalsType">
    <xsd:sequence>
      <xsd:element name="principal" type="JAZN-DATAprincipalType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAprincipalType">
    <xsd:sequence >
      <xsd:element name="realm-name" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="type" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="name" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAcodesourceType">
    <xsd:sequence>
      <xsd:element name="url" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATApermissionsType">
    <xsd:sequence>
      <xsd:element name="permission" type="JAZN-DATApermissionType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATApermissionType">
    <xsd:sequence >
```

```
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="actions" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-permission-classesType">
    <xsd:sequence>
      <xsd:element name="permission-class" type="JAZN-DATApermission-classType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATApermission-classType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="type" type="xsd:string" />
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="target-descriptors" type="JAZN-DATAtarget-descriptorsType"
/>
      <xsd:element name="action-descriptors" type="JAZN-DATAaction-descriptorsType"
minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAtarget-descriptorsType">
    <xsd:sequence>
      <xsd:element name="target-descriptor" type="JAZN-DATAtarget-descriptorType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAtarget-descriptorType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAaction-descriptorsType">
    <xsd:sequence>
      <xsd:element name="action-descriptor" type="JAZN-DATAaction-descriptorType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAaction-descriptorType">
```

```
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-principal-classesType">
    <xsd:sequence>
      <xsd:element name="principal-class" type="JAZN-DATAprincipal-classType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAprincipal-classType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="type" type="xsd:string" />
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="name-description-map"
type="JAZN-DATAname-description-mapType" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAname-description-mapType">
    <xsd:sequence>
      <xsd:element name="name-description-pair"
type="JAZN-DATAname-description-pairType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAname-description-pairType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-loginconfigType">
    <xsd:sequence>
      <xsd:element name="application" type="JAZN-DATAapplicationType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAapplicationType">
    <xsd:sequence >
```

```
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="login-modules" type="JAZN-DATAlogin-modulesType" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAlogin-modulesType">
    <xsd:sequence>
      <xsd:element name="login-module" type="JAZN-DATAlogin-moduleType"
minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAlogin-moduleType">
    <xsd:sequence >
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="control-flag" type="xsd:string" />
      <xsd:element name="options" type="JAZN-DATAoptionsType" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="class
control-flag"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAoptionsType">
    <xsd:sequence>
      <xsd:element name="option" type="JAZN-DATAoptionType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAoptionType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="value" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name
value"/>
  </xsd:complexType>



  <xsd:simpleType name="JAZN-DATAkeyset">
    <xsd:list itemType="xsd:string"/>
  </xsd:simpleType>




</xsd:schema>
```

jazn.xml のスキーマ

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" >
<xsd:element name="jazn" type="jaznType"/>
<xsd:complexType name="jaznType">
<xsd:sequence >
<xsd:element name="property" type="jaznPropertyType" minOccurs="0"
maxOccurs="unbounded"/>
<xsd:element name="jazn-web-app" type="jazn-web-appType" minOccurs="0"
maxOccurs="1"/>
</xsd:sequence>
<xsd:attribute name="provider" type="xsd:string" use="optional"/>
<xsd:attribute name="location" type="xsd:string" use="optional"/>
<xsd:attribute name="default-realm" type="xsd:string" use="optional"/>
<xsd:attribute name="persistence" type="xsd:string" use="optional"/>
<xsd:attribute name="config" type="xsd:string" use="optional"/>
</xsd:complexType>
<xsd:complexType name="jazn-web-appType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="auth-method" type="xsd:string" use="optional"/>
<xsd:attribute name="runas-mode" type="runas-modeType" use="default" value="false"/>
<xsd:attribute name="doasprivileged-mode" type="doasprivileged-modeType"
use="default" value="true"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
<xsd:simpleType name="runas-modeType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="true"/>
<xsd:enumeration value="false"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="doasprivileged-modeType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="true"/>
<xsd:enumeration value="false"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="jaznPropertyType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="Key" type="jaznKeyset" use="fixed" value="name"/>
<xsd:attribute name="name" type="xsd:string" use="optional"/>
<xsd:attribute name="value" type="xsd:string" use="optional"/>
</xsd:extension>
```

```
</xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="jaznKeyset">
<xsd:list itemType = "xsd:string"/>
</xsd:simpleType>
</xsd:schema>
<!ENTITY % CHARSET "CDATA">
<!ENTITY % WEBPATH "CDATA">
<!ENTITY % NUMBER "CDATA">
<!ENTITY % HOST "CDATA">
<!ENTITY % PATH "CDATA">
<!ENTITY % CLASSNAME "CDATA">
<!-- A group that this security-role-mapping implies. Ie all the members of the
specified group are included in this role. -->
<!ELEMENT group (#PCDATA)>
<!ATTLIST group name CDATA #IMPLIED
>
<!-- An attribute sent to the context. The only mandatory attribute in JNDI is the
'java.naming.factory.initial' which is the classname of the context factory
implementation. -->
<!ELEMENT context-attribute (#PCDATA)>
<!ATTLIST context-attribute name CDATA #IMPLIED
value CDATA #IMPLIED
>
<!-- Defines the relative/absolute path to a file containing mime-mappings to use.
-->
<!ELEMENT mime-mappings (#PCDATA)>
<!ATTLIST mime-mappings path CDATA #IMPLIED
>
<!-- Specifies a codebase where classes used by this application (servlets/beans,
etc) can be found. -->
<!ELEMENT classpath (#PCDATA)>
<!ATTLIST classpath path CDATA #REQUIRED
>
<!-- The specification of an optional javax.naming.Context implementation used for
retrieving the resource. This is useful when hooking up with 3rd party modules, such
as a 3rd party JMS server for instance. Either use the context implementation
supplied by the resource vendor or if none exists write an implementation which in
turn negotiates with the vendor software. -->
<!ELEMENT lookup-context (context-attribute+)>
<!ATTLIST lookup-context location CDATA #IMPLIED
>
<!-- Specifies a servlet to use as request-tracker, request-trackers are invoked for
every request and are useful for logging purposes etc. -->
<!ELEMENT request-tracker (#PCDATA)>
<!ATTLIST request-tracker servlet-name CDATA #IMPLIED
```

```
>
<!-- The resource-ref element is used for the declaration of a reference to an
external resource such as a datasource, JMS queue, mail session or similar.
The resource-ref-mapping ties this to a JNDI-location when deploying. -->
<!ELEMENT resource-ref-mapping (lookup-context?)>
<!ATTLIST resource-ref-mapping location CDATA #IMPLIED
name CDATA #REQUIRED
>
<!-- Tag that is defined if the application is to be clustered. Clustered
applications have their ServletContext and session data
shared between the apps in the cluster, the values have to be either Serializable or
be remote RMI-objects (implement java.rmi.Remote). -->
<!ELEMENT cluster-config (#PCDATA)>
<!ATTLIST cluster-config host %HOST; "230.0.0.1"
id CDATA "based on local IP"
port %NUMBER; "9127"
>
<!-- Specifies an optional access-mask for this application, hostnames and
ip/subnets can be used to filter out allowed clients of this application. -->
<!ELEMENT access-mask (host-access*, ip-access*)>
<!ATTLIST access-mask default (allow|deny) "allow"
>
<!-- Overrides the value of an env-entry in the assembly descriptor. It is used to
keep the .ear (assembly) clean from deployment-specific values. The body is the
value. -->
<!ELEMENT env-entry-mapping (#PCDATA)>
<!ATTLIST env-entry-mapping name CDATA #IMPLIED
>

<!-- Specifies the Expires setting for a given set of resources, useful for caching
policies (for instance for browsers not to reload images as frequently as documents
etc). -->
<!ELEMENT expiration-setting (#PCDATA)>
<!ATTLIST expiration-setting expires CDATA #IMPLIED
url-pattern CDATA #IMPLIED
>
<!-- Overrides the value of a context-param in the assembly descriptor. It is used
to keep the .ear (assembly) clean from deployment-specific values. The body is the
value. -->
<!ELEMENT context-param-mapping (#PCDATA)>
<!ATTLIST context-param-mapping name CDATA #IMPLIED
>
<!-- Session-tracking settings for this application. -->
<!ELEMENT session-tracking (session-tracker*)>
<!ATTLIST session-tracking autoencode-absolute-urls (true|false) "false"
autoencode-urls (true|false) "true"
autojoin-session (true|false) "false"
```

```
cookie-domain CDATA #IMPLIED
cookie-max-age %NUMBER; "in memory only"
cookies (enabled|disabled) "enabled"
>
<!-- A user that this security-role-mapping implies. -->
<!ELEMENT user (#PCDATA)>
<!ATTLIST user name CDATA #IMPLIED
>
<!-- Adds a virtual directory mapping, used to include files that doesnt physically
reside below the document-root among the web-exposed files. -->
<!ELEMENT virtual-directory (#PCDATA)>
<!ATTLIST virtual-directory real-path %PATH; #IMPLIED
virtual-path %PATH; #IMPLIED
>
<!-- Specifies an ip/netmask who is allowed access. -->
<!ELEMENT ip-access (#PCDATA)>
<!ATTLIST ip-access ip CDATA #REQUIRED
mode (allow|deny) #REQUIRED
netmask CDATA #IMPLIED
>
<!-- Specifies a servlet to use as chainer for a specified mime-type. Useful to
filter/transform certain kinds of output. -->
<!ELEMENT servlet-chaining (#PCDATA)>
<!ATTLIST servlet-chaining mime-type CDATA #IMPLIED
servlet-name CDATA #IMPLIED
>
<!-- Specifies a domain or netmask who is allowed access. -->
<!ELEMENT host-access (#PCDATA)>
<!ATTLIST host-access domain CDATA #REQUIRED
mode (allow|deny) #REQUIRED
>
<!-- The ejb-ref element is used for the declaration of a reference to
another enterprise bean's home. The ejb-ref-mapping ties this to a JNDI-location
when deploying. -->
<!ELEMENT ejb-ref-mapping (#PCDATA)>
<!ATTLIST ejb-ref-mapping location CDATA #IMPLIED
name CDATA #REQUIRED
>
<!-- The runtime mapping (to groups and users) of a role. Maps to a security-role of
the same name in the assembly descriptor. -->
<!ELEMENT security-role-mapping (group*, user*)>
<!ATTLIST security-role-mapping impliesAll CDATA #IMPLIED
name CDATA #IMPLIED
>
<!-- Specifies a servlet to use as session-tracker, session-trackers are invoked as
soon as a session is created and are useful for logging purposes etc. -->
<!ELEMENT session-tracker (#PCDATA)>
```

```
<!ATTLIST session-tracker servlet-name CDATA #IMPLIED
>
<!-- JAZN configuration -->
<!ELEMENT jazn-web-app (#PCDATA)>
<!ATTLIST jazn-web-app auth-method CDATA #IMPLIED
runas-mode (true | false) "false"
doasprivileged-mode (true | false) "true"
>
<!-- Web-app class loader configuration -->
<!ELEMENT web-app-class-loader EMPTY>
<!ATTLIST web-app-class-loader
search-local-classes-first (true | false) "false"
include-war-manifest-class-path (true | false) "true"
>
<!--
Copyright (c) 2000 Sun Microsystems, Inc.,
901 San Antonio Road,
Palo Alto, California 94303, U.S.A.
All rights reserved.
Sun Microsystems, Inc. has intellectual property rights relating to
technology embodied in the product that is described in this document.
In particular, and without limitation, these intellectual property
rights may include one or more of the U.S. patents listed at
http://www.sun.com/patents and one or more additional patents or
pending patent applications in the U.S. and in other countries.
This document and the product to which it pertains are distributed
under licenses restricting their use, copying, distribution, and
decompilation. This Product or document may be reproduced but may not
be changed without prior written authorization of Sun and its
licensors, if any.
Third-party software, including font technology, is copyrighted and
licensed from Sun suppliers.
Sun, Sun Microsystems, the Sun logo, Java, JavaServer Pages, Java
Naming and Directory Interface, JDBC, JDK, JavaMail and and
Enterprise JavaBeans are trademarks or registered trademarks of Sun
Microsystems, Inc. in the U.S. and other countries.
Federal Acquisitions: Commercial Software - Government Users Subject to
Standard License Terms and Conditions.
DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED
CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED
WARRANTY OF MERCHANTABILITY, FITNESS FOR FOR A PARTICULAR PURPOSE OR
NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH
DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

_____________________________________________________________________
Copyright (c) 2000 Sun Microsystems, Inc.,
901 San Antonio Road,
```

```
Palo Alto, California 94303, E'tats-Unis.
Tous droits re'serve's.
Sun Microsystems, Inc. a les droits de proprie'te' intellectuels
relatants a` la technologie incorpore'e dans le produit qui est de'crit
dans ce document. En particulier, et sans la limitation, ces droits de
proprie'te' intellectuels peuvent inclure un ou plus des brevets
ame'ricains e'nume're's a` http://www.sun.com/patents et un ou les
brevets plus supple'mentaires ou les applications de brevet en attente
dans les E'tats-Unis et dans les autres pays.
Ce produit ou document est prote'ge' par un copyright et distribue'
avec des licences qui en restreignent l'utilisation, la copie, la
distribution, et la de'compilation. Ce produit sa documention associe
n peut e^tre reproduite, par quelque moyen que ce soit, sans
l'autorisation pre'alable et e'crite de Sun et de ses bailleurs de
licence, le cas e'che'ant.
Le logiciel de'tenu par des tiers, et qui comprend la technologie
relative aux polices de caracte`res, est prote'ge' par un copyright et
licencie' par des fournisseurs de Sun.
Sun, Sun Microsystems, le logo Sun, Java, JavaServer Pages, Java
Naming and Directory Interface, JDBC, JDK, JavaMail et and
Enterprise JavaBeans sont des marques de fabrique ou des marques
de'pose'es de Sun Microsystems, Inc. aux E'tats-Unis et dans d'autres
pays.
LA DOCUMENTATION EST FOURNIE "EN L'E'TAT" ET TOUTES AUTRES CONDITIONS,
DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT
EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS
NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A
L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE
CONTREFAC,ON.
-->
<!--
This is the XML DTD for the Servlet 2.3 deployment descriptor.
All Servlet 2.3 deployment descriptors must include a DOCTYPE
of the following form:
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
-->
<!--
The following conventions apply to all J2EE deployment descriptor
elements unless indicated otherwise.
- In elements that contain PCDATA, leading and trailing whitespace
in the data may be ignored.
- In elements whose value is an "enumerated type", the value is
case sensitive.
- In elements that specify a pathname to a file within the same
JAR file, relative filenames (i.e., those not starting with "/")
```

```
are considered relative to the root of the JAR file's namespace.
Absolute filenames (i.e., those starting with "/") also specify
names in the root of the JAR file's namespace. In general, relative
names are preferred. The exception is .war files where absolute
names are preferred for consistency with the servlet API.
-->

<!--
The web-app element is the root of the deployment descriptor for
a web application.
-->
<!ELEMENT web-app (icon?, display-name?, description?, distributable?,
context-param*, filter*, filter-mapping*, listener*, servlet*,
servlet-mapping*, session-config?, mime-mapping*, welcome-file-list?,
error-page*, taglib*, resource-env-ref*, resource-ref*, security-constraint*,
login-config?, security-role*, env-entry*, ejb-ref*, ejb-local-ref*)>
<!--
The auth-constraint element indicates the user roles that should
be permitted access to this resource collection. The role-name
used here must either correspond to the role-name of one of the
security-role elements defined for this web application, or be
the specially reserved role-name "*" that is a compact syntax for
indicating all roles in the web application. If both "*" and
rolenames appear, the container interprets this as all roles.
If no roles are defined, no user is allowed access to the portion of
the web application described by the containing security-constraint.
The container matches role names case sensitively when determining
access.

Used in: security-constraint
-->
<!ELEMENT auth-constraint (description?, role-name*)>
<!--
The auth-method element is used to configure the authentication
mechanism for the web application. As a prerequisite to gaining access to any web
resources which are protected by an authorization
constraint, a user must have authenticated using the configured
mechanism. Legal values for this element are "BASIC", "DIGEST",
"FORM", or "CLIENT-CERT".
Used in: login-config
-->
<!ELEMENT auth-method (#PCDATA)>
<!--
The context-param element contains the declaration of a web
application's servlet context initialization parameters.
Used in: web-app
-->
```

```
<!ELEMENT context-param (param-name, param-value, description?)>
<!--
The description element is used to provide text describing the parent
element. The description element should include any information that
the web application war file producer wants to provide to the consumer of
the web application war file (i.e., to the Deployer). Typically, the tools
used by the web application war file consumer will display the description
when processing the parent element that contains the description.
Used in: auth-constraint, context-param, ejb-local-ref, ejb-ref,
env-entry, filter, init-param, resource-env-ref, resource-ref, run-as,
security-role, security-role-ref, servlet, user-data-constraint,
web-app, web-resource-collection
-->
<!ELEMENT description (#PCDATA)>
<!--
The display-name element contains a short name that is intended to be
displayed by tools. The display name need not be unique.
Used in: filter, security-constraint, servlet, web-app
Example:
<display-name>Employee Self Service</display-name>
-->
<!ELEMENT display-name (#PCDATA)>
<!--
The distributable element, by its presence in a web application
deployment descriptor, indicates that this web application is
programmed appropriately to be deployed into a distributed servlet
container
Used in: web-app
-->
<!ELEMENT distributable EMPTY>
<!--
The ejb-link element is used in the ejb-ref or ejb-local-ref
elements to specify that an EJB reference is linked to an
enterprise bean.
The name in the ejb-link element is composed of a
path name specifying the ejb-jar containing the referenced enterprise
bean with the ejb-name of the target bean appended and separated from
the path name by "#". The path name is relative to the war file
containing the web application that is referencing the enterprise bean.
This allows multiple enterprise beans with the same ejb-name to be
uniquely identified.
Used in: ejb-local-ref, ejb-ref
Examples:
<ejb-link>EmployeeRecord</ejb-link>
<ejb-link>../products/product.jar#ProductEJB</ejb-link>
-->
<!ELEMENT ejb-link (#PCDATA)>
```

```
<!--
The ejb-local-ref element is used for the declaration of a reference to
an enterprise bean's local home. The declaration consists of:
- an optional description
- the EJB reference name used in the code of the web application
that's referencing the enterprise bean
- the expected type of the referenced enterprise bean
- the expected local home and local interfaces of the referenced
enterprise bean
- optional ejb-link information, used to specify the referenced
enterprise bean
Used in: web-app
-->
<!ELEMENT ejb-local-ref (description?, ejb-ref-name, ejb-ref-type,
local-home, local, ejb-link?)>
<!--
The ejb-ref element is used for the declaration of a reference to
an enterprise bean's home. The declaration consists of:
- an optional description
- the EJB reference name used in the code of
the web application that's referencing the enterprise bean
- the expected type of the referenced enterprise bean
- the expected home and remote interfaces of the referenced
enterprise bean
- optional ejb-link information, used to specify the referenced
enterprise bean
Used in: web-app
-->
<!ELEMENT ejb-ref (description?, ejb-ref-name, ejb-ref-type,
home, remote, ejb-link?)>
<!--
The ejb-ref-name element contains the name of an EJB reference. The
EJB reference is an entry in the web application's environment and is
relative to the java:comp/env context. The name must be unique
within the web application.
It is recommended that name is prefixed with "ejb/".
Used in: ejb-local-ref, ejb-ref
Example:
<ejb-ref-name>ejb/Payroll</ejb-ref-name>
-->
<!ELEMENT ejb-ref-name (#PCDATA)>
<!--
The ejb-ref-type element contains the expected type of the
referenced enterprise bean.
The ejb-ref-type element must be one of the following:
<ejb-ref-type>Entity</ejb-ref-type>
<ejb-ref-type>Session</ejb-ref-type>
```

```
Used in: ejb-local-ref, ejb-ref
-->
<!ELEMENT ejb-ref-type (#PCDATA)>
<!--
The env-entry element contains the declaration of a web application's
environment entry. The declaration consists of an optional
description, the name of the environment entry, and an optional
value. If a value is not specified, one must be supplied
during deployment.
-->
<!ELEMENT env-entry (description?, env-entry-name, env-entry-value?,
env-entry-type)>
<!--
The env-entry-name element contains the name of a web applications's
environment entry. The name is a JNDI name relative to the
java:comp/env context. The name must be unique within a web application.
Example:
<env-entry-name>minAmount</env-entry-name>
Used in: env-entry
-->
<!ELEMENT env-entry-name (#PCDATA)>
<!--
The env-entry-type element contains the fully-qualified Java type of
the environment entry value that is expected by the web application's
code.
The following are the legal values of env-entry-type:
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.String
java.lang.Short
java.lang.Integer
java.lang.Long
java.lang.Float
java.lang.Double
Used in: env-entry
-->
<!ELEMENT env-entry-type (#PCDATA)>
<!--
The env-entry-value element contains the value of a web application's
environment entry. The value must be a String that is valid for the
constructor of the specified type that takes a single String
parameter, or for java.lang.Character, a single character.
Example:
<env-entry-value>100.00</env-entry-value>
Used in: env-entry
-->
```

```
<!ELEMENT env-entry-value (#PCDATA)>
<!--
The error-code contains an HTTP error code, ex: 404
Used in: error-page
-->
<!ELEMENT error-code (#PCDATA)>
<!--
The error-page element contains a mapping between an error code
or exception type to the path of a resource in the web application
Used in: web-app
-->
<!ELEMENT error-page ((error-code | exception-type), location)>
<!--
The exception type contains a fully qualified class name of a
Java exception type.
Used in: error-page
-->
<!ELEMENT exception-type (#PCDATA)>
<!--
The extension element contains a string describing an
extension. example: "txt"
Used in: mime-mapping
-->
<!ELEMENT extension (#PCDATA)>
<!--
Declares a filter in the web application. The filter is mapped to
either a servlet or a URL pattern in the filter-mapping element, using
the filter-name value to reference. Filters can access the
initialization parameters declared in the deployment descriptor at
runtime via the FilterConfig interface.
Used in: web-app
-->
<!ELEMENT filter (icon?, filter-name, display-name?, description?,
filter-class, init-param*)>
<!--
The fully qualified classname of the filter.
Used in: filter
-->
<!ELEMENT filter-class (#PCDATA)>
<!--
Declaration of the filter mappings in this web application. The
container uses the filter-mapping declarations to decide which filters
to apply to a request, and in what order. The container matches the
request URI to a Servlet in the normal way. To determine which filters
to apply it matches filter-mapping declarations either on servlet-name,
or on url-pattern for each filter-mapping element, depending on which
style is used. The order in which filters are invoked is the order in
```

```
which filter-mapping declarations that match a request URI for a
servlet appear in the list of filter-mapping elements.The filter-name
value must be the value of the <filter-name> sub-elements of one of the
<filter> declarations in the deployment descriptor.
Used in: web-app
-->
<!ELEMENT filter-mapping (filter-name, (url-pattern | servlet-name))>
<!--
The logical name of the filter. This name is used to map the filter.
Each filter name is unique within the web application.
Used in: filter, filter-mapping
-->
<!ELEMENT filter-name (#PCDATA)>
<!--
The form-error-page element defines the location in the web app
where the error page that is displayed when login is not successful
can be found. The path begins with a leading / and is interpreted
relative to the root of the WAR.
Used in: form-login-config
-->
<!ELEMENT form-error-page (#PCDATA)>
<!--
The form-login-config element specifies the login and error pages
that should be used in form based login. If form based authentication
is not used, these elements are ignored.
Used in: login-config
-->
<!ELEMENT form-login-config (form-login-page, form-error-page)>
<!--
The form-login-page element defines the location in the web app
where the page that can be used for login can be found. The path
begins with a leading / and is interpreted relative to the root of the WAR.
Used in: form-login-config
-->
<!ELEMENT form-login-page (#PCDATA)>
<!--
The home element contains the fully-qualified name of the enterprise
bean's home interface.
Used in: ejb-ref
Example:
<home>com.aardvark.payroll.PayrollHome</home>
-->
<!ELEMENT home (#PCDATA)>
<!--
The http-method contains an HTTP method (GET | POST |...).
Used in: web-resource-collection
-->
```

```
<!ELEMENT http-method (#PCDATA)>
<!--
The icon element contains small-icon and large-icon elements that
specify the file names for small and a large GIF or JPEG icon images
used to represent the parent element in a GUI tool.
Used in: filter, servlet, web-app
-->
<!ELEMENT icon (small-icon?, large-icon?)>
<!--
The init-param element contains a name/value pair as an
initialization param of the servlet
Used in: filter, servlet
-->
<!ELEMENT init-param (param-name, param-value, description?)>
<!--
The jsp-file element contains the full path to a JSP file within
the web application beginning with a `/'.
Used in: servlet
-->
<!ELEMENT jsp-file (#PCDATA)>
<!--
The large-icon element contains the name of a file
containing a large (32 x 32) icon image. The file
name is a relative path within the web application's
war file.
The image may be either in the JPEG or GIF format.
The icon can be used by tools.
Used in: icon
Example:
<large-icon>employee-service-icon32x32.jpg</large-icon>
-->
<!ELEMENT large-icon (#PCDATA)>
<!--
The listener element indicates the deployment properties for a web
application listener bean.
Used in: web-app
-->
<!ELEMENT listener (listener-class)>
<!--
The listener-class element declares a class in the application must be
registered as a web application listener bean. The value is the fully qualified
classname of the listener class.

Used in: listener
-->
<!ELEMENT listener-class (#PCDATA)>
<!--
```

```
The load-on-startup element indicates that this servlet should be
loaded (instantiated and have its init() called) on the startup
of the web application. The optional contents of
these element must be an integer indicating the order in which
the servlet should be loaded. If the value is a negative integer,
or the element is not present, the container is free to load the
servlet whenever it chooses. If the value is a positive integer
or 0, the container must load and initialize the servlet as the
application is deployed. The container must guarantee that
servlets marked with lower integers are loaded before servlets
marked with higher integers. The container may choose the order
of loading of servlets with the same load-on-start-up value.
Used in: servlet
-->
<!ELEMENT load-on-startup (#PCDATA)>
<!--
The local element contains the fully-qualified name of the
enterprise bean's local interface.
Used in: ejb-local-ref
-->
<!ELEMENT local (#PCDATA)>
<!--
The local-home element contains the fully-qualified name of the
enterprise bean's local home interface.
Used in: ejb-local-ref
-->
<!ELEMENT local-home (#PCDATA)>
<!--
The location element contains the location of the resource in the web
application relative to the root of the web application. The value of
the location must have a leading `/'.
Used in: error-page
-->
<!ELEMENT location (#PCDATA)>
<!--
The login-config element is used to configure the authentication
method that should be used, the realm name that should be used for
this application, and the attributes that are needed by the form login
mechanism.
Used in: web-app
-->
<!ELEMENT login-config (auth-method?, realm-name?, form-login-config?)>
<!--
The mime-mapping element defines a mapping between an extension
and a mime type.
Used in: web-app
-->
```

```
<!ELEMENT mime-mapping (extension, mime-type)>
<!--
The mime-type element contains a defined mime type. example:
"text/plain"
Used in: mime-mapping
-->
<!ELEMENT mime-type (#PCDATA)>
<!--
The param-name element contains the name of a parameter. Each parameter
name must be unique in the web application.

Used in: context-param, init-param
-->
<!ELEMENT param-name (#PCDATA)>
<!--
The param-value element contains the value of a parameter.
Used in: context-param, init-param
-->
<!ELEMENT param-value (#PCDATA)>
<!--
The realm name element specifies the realm name to use in HTTP
Basic authorization.
Used in: login-config
-->
<!ELEMENT realm-name (#PCDATA)>
<!--
The remote element contains the fully-qualified name of the enterprise
bean's remote interface.
Used in: ejb-ref
Example:
<remote>com.wombat.empl.EmployeeService</remote>
-->
<!ELEMENT remote (#PCDATA)>
<!--
The res-auth element specifies whether the web application code signs
on programmatically to the resource manager, or whether the Container
will sign on to the resource manager on behalf of the web application. In the
latter case, the Container uses information that is supplied by the
Deployer.
The value of this element must be one of the two following:
<res-auth>Application</res-auth>
<res-auth>Container</res-auth>
Used in: resource-ref
-->
<!ELEMENT res-auth (#PCDATA)>
<!--
The res-ref-name element specifies the name of a resource manager
```

```
connection factory reference. The name is a JNDI name relative to the
java:comp/env context. The name must be unique within a web application.
Used in: resource-ref
-->
<!ELEMENT res-ref-name (#PCDATA)>
<!--
The res-sharing-scope element specifies whether connections obtained
through the given resource manager connection factory reference can be
shared. The value of this element, if specified, must be one of the
two following:
<res-sharing-scope>Shareable</res-sharing-scope>
<res-sharing-scope>Unshareable</res-sharing-scope>
The default value is Shareable.
Used in: resource-ref
-->
<!ELEMENT res-sharing-scope (#PCDATA)>
<!--
The res-type element specifies the type of the data source. The type
is specified by the fully qualified Java language class or interface
expected to be implemented by the data source.
Used in: resource-ref
-->
<!ELEMENT res-type (#PCDATA)>
<!--
The resource-env-ref element contains a declaration of a web application's
reference to an administered object associated with a resource
in the web application's environment. It consists of an optional
description, the resource environment reference name, and an
indication of the resource environment reference type expected by
the web application code.
Used in: web-app
Example:
<resource-env-ref>
<resource-env-ref-name>jms/StockQueue</resource-env-ref-name>
<resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
-->
<!ELEMENT resource-env-ref (description?, resource-env-ref-name,
resource-env-ref-type)>
<!--
The resource-env-ref-name element specifies the name of a resource
environment reference; its value is the environment entry name used in
the web application code. The name is a JNDI name relative to the
java:comp/env context and must be unique within a web application.
Used in: resource-env-ref
-->
<!ELEMENT resource-env-ref-name (#PCDATA)>
```

```
<!--
The resource-env-ref-type element specifies the type of a resource
environment reference. It is the fully qualified name of a Java
language class or interface.
Used in: resource-env-ref
-->
<!ELEMENT resource-env-ref-type (#PCDATA)>
<!--
The resource-ref element contains a declaration of a web application's
reference to an external resource. It consists of an optional
description, the resource manager connection factory reference name,
the indication of the resource manager connection factory type
expected by the web application code, the type of authentication
(Application or Container), and an optional specification of the
shareability of connections obtained from the resource (Shareable or
Unshareable).
Used in: web-app
Example:
<resource-ref>
<res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
-->
<!ELEMENT resource-ref (description?, res-ref-name, res-type, res-auth,
res-sharing-scope?)>
<!--
The role-link element is a reference to a defined security role. The
role-link element must contain the name of one of the security roles
defined in the security-role elements.
Used in: security-role-ref
-->
<!ELEMENT role-link (#PCDATA)>
<!--
The role-name element contains the name of a security role.
The name must conform to the lexical rules for an NMTOKEN.
Used in: auth-constraint, run-as, security-role, security-role-ref
-->
<!ELEMENT role-name (#PCDATA)>
<!--
The run-as element specifies the run-as identity to be used for the
execution of the web application. It contains an optional description, and
the name of a security role.
Used in: servlet
-->
<!ELEMENT run-as (description?, role-name)>
```

```
<!--
The security-constraint element is used to associate security
constraints with one or more web resource collections
Used in: web-app
-->
<!ELEMENT security-constraint (display-name?, web-resource-collection+,
auth-constraint?, user-data-constraint?)>
<!--
The security-role element contains the definition of a security
role. The definition consists of an optional description of the
security role, and the security role name.
Used in: web-app
Example:
<security-role>
<description>
This role includes all employees who are authorized
to access the employee service application.
</description>
<role-name>employee</role-name>
</security-role>
-->
<!ELEMENT security-role (description?, role-name)>
<!--
The security-role-ref element contains the declaration of a security
role reference in the web application's code. The declaration consists
of an optional description, the security role name used in the code,
and an optional link to a security role. If the security role is not
specified, the Deployer must choose an appropriate security role.
The value of the role-name element must be the String used as the
parameter to the EJBContext.isCallerInRole(String roleName) method
or the HttpServletRequest.isUserInRole(String role) method.
Used in: servlet
-->
<!ELEMENT security-role-ref (description?, role-name, role-link?)>
<!--
The servlet element contains the declarative data of a
servlet. If a jsp-file is specified and the load-on-startup element is
present, then the JSP should be precompiled and loaded.
Used in: web-app
-->
<!ELEMENT servlet (icon?, servlet-name, display-name?, description?,
(servlet-class|jsp-file), init-param*, load-on-startup?, run-as?,
security-role-ref*)>
<!--
The servlet-class element contains the fully qualified class name
of the servlet.
Used in: servlet
```

```
-->
<!ELEMENT servlet-class (#PCDATA)>
<!--
The servlet-mapping element defines a mapping between a servlet
and a url pattern
Used in: web-app
-->
<!ELEMENT servlet-mapping (servlet-name, url-pattern)>
<!--
The servlet-name element contains the canonical name of the
servlet. Each servlet name is unique within the web application.
Used in: filter-mapping, servlet, servlet-mapping
-->
<!ELEMENT servlet-name (#PCDATA)>
<!--
The session-config element defines the session parameters for
this web application.
Used in: web-app
-->
<!ELEMENT session-config (session-timeout?)>
<!--
The session-timeout element defines the default session timeout
interval for all sessions created in this web application. The
specified timeout must be expressed in a whole number of minutes.
If the timeout is 0 or less, the container ensures the default
behaviour of sessions is never to time out.
Used in: session-config
-->
<!ELEMENT session-timeout (#PCDATA)>
<!--
The small-icon element contains the name of a file
containing a small (16 x 16) icon image. The file
name is a relative path within the web application's
war file.
The image may be either in the JPEG or GIF format.
The icon can be used by tools.
Used in: icon
Example:
<small-icon>employee-service-icon16x16.jpg</small-icon>
-->
<!ELEMENT small-icon (#PCDATA)>
<!--
The taglib element is used to describe a JSP tag library.
Used in: web-app
-->
<!ELEMENT taglib (taglib-uri, taglib-location)>
<!--
```

```
the taglib-location element contains the location (as a resource
relative to the root of the web application) where to find the Tag
Libary Description file for the tag library.
Used in: taglib
-->
<!ELEMENT taglib-location (#PCDATA)>
<!--
The taglib-uri element describes a URI, relative to the location
of the web.xml document, identifying a Tag Library used in the Web
Application.
Used in: taglib
-->
<!ELEMENT taglib-uri (#PCDATA)>
<!--
The transport-guarantee element specifies that the communication
between client and server should be NONE, INTEGRAL, or
CONFIDENTIAL. NONE means that the application does not require any
transport guarantees. A value of INTEGRAL means that the application
requires that the data sent between the client and server be sent in
such a way that it can't be changed in transit. CONFIDENTIAL means
that the application requires that the data be transmitted in a
fashion that prevents other entities from observing the contents of
the transmission. In most cases, the presence of the INTEGRAL or
CONFIDENTIAL flag will indicate that the use of SSL is required.
Used in: user-data-constraint
-->
<!ELEMENT transport-guarantee (#PCDATA)>
<!--
The url-pattern element contains the url pattern of the mapping. Must
follow the rules specified in Section 11.2 of the Servlet API
Specification.
Used in: filter-mapping, servlet-mapping, web-resource-collection
-->
<!ELEMENT url-pattern (#PCDATA)>
<!--
The user-data-constraint element is used to indicate how data
communicated between the client and container should be protected.
Used in: security-constraint
-->
<!ELEMENT user-data-constraint (description?, transport-guarantee)>
<!--
The web-resource-collection element is used to identify a subset
of the resources and HTTP methods on those resources within a web
application to which a security constraint applies. If no HTTP methods
are specified, then the security constraint applies to all HTTP
methods.
Used in: security-constraint
```

```
-->
<!ELEMENT web-resource-collection (web-resource-name, description?,
url-pattern*, http-method*)>
<!--
The web-resource-name contains the name of this web resource
collection.
Used in: web-resource-collection
-->
<!ELEMENT web-resource-name (#PCDATA)>
<!--
The welcome-file element contains file name to use as a default
welcome file, such as index.html
Used in: welcome-file-list
-->
<!ELEMENT welcome-file (#PCDATA)>
<!--
The welcome-file-list contains an ordered list of welcome files
elements.
Used in: web-app
-->
<!ELEMENT welcome-file-list (welcome-file+)>
<!--
The ID mechanism is to allow tools that produce additional deployment
information (i.e., information beyond the standard deployment
descriptor information) to store the non-standard information in a
separate file, and easily refer from these tool-specific files to the
information in the standard deployment descriptor.
Tools are not allowed to add the non-standard information into the
standard deployment descriptor.
-->
<!ATTLIST auth-constraint id ID #IMPLIED>
<!ATTLIST auth-method id ID #IMPLIED>
<!ATTLIST context-param id ID #IMPLIED>
<!ATTLIST description id ID #IMPLIED>
<!ATTLIST display-name id ID #IMPLIED>
<!ATTLIST distributable id ID #IMPLIED>
<!ATTLIST ejb-link id ID #IMPLIED>
<!ATTLIST ejb-local-ref id ID #IMPLIED>
<!ATTLIST ejb-ref id ID #IMPLIED>
<!ATTLIST ejb-ref-name id ID #IMPLIED>
<!ATTLIST ejb-ref-type id ID #IMPLIED>
<!ATTLIST env-entry id ID #IMPLIED>
<!ATTLIST env-entry-name id ID #IMPLIED>
<!ATTLIST env-entry-type id ID #IMPLIED>
<!ATTLIST env-entry-value id ID #IMPLIED>
<!ATTLIST error-code id ID #IMPLIED>
<!ATTLIST error-page id ID #IMPLIED>
```

```
<!ATTLIST exception-type id ID #IMPLIED>
<!ATTLIST extension id ID #IMPLIED>
<!ATTLIST filter id ID #IMPLIED>
<!ATTLIST filter-class id ID #IMPLIED>
<!ATTLIST filter-mapping id ID #IMPLIED>
<!ATTLIST filter-name id ID #IMPLIED>
<!ATTLIST form-error-page id ID #IMPLIED>
<!ATTLIST form-login-config id ID #IMPLIED>
<!ATTLIST form-login-page id ID #IMPLIED>
<!ATTLIST home id ID #IMPLIED>
<!ATTLIST http-method id ID #IMPLIED>
<!ATTLIST icon id ID #IMPLIED>
<!ATTLIST init-param id ID #IMPLIED>
<!ATTLIST jsp-file id ID #IMPLIED>
<!ATTLIST large-icon id ID #IMPLIED>
<!ATTLIST listener id ID #IMPLIED>
<!ATTLIST listener-class id ID #IMPLIED>
<!ATTLIST load-on-startup id ID #IMPLIED>
<!ATTLIST local id ID #IMPLIED>
<!ATTLIST local-home id ID #IMPLIED>
<!ATTLIST location id ID #IMPLIED>
<!ATTLIST login-config id ID #IMPLIED>
<!ATTLIST mime-mapping id ID #IMPLIED>
<!ATTLIST mime-type id ID #IMPLIED>
<!ATTLIST param-name id ID #IMPLIED>
<!ATTLIST param-value id ID #IMPLIED>
<!ATTLIST realm-name id ID #IMPLIED>
<!ATTLIST remote id ID #IMPLIED>
<!ATTLIST res-auth id ID #IMPLIED>
<!ATTLIST res-ref-name id ID #IMPLIED>
<!ATTLIST res-sharing-scope id ID #IMPLIED>
<!ATTLIST res-type id ID #IMPLIED>
<!ATTLIST resource-env-ref id ID #IMPLIED>
<!ATTLIST resource-env-ref-name id ID #IMPLIED>
<!ATTLIST resource-env-ref-type id ID #IMPLIED>
<!ATTLIST resource-ref id ID #IMPLIED>
<!ATTLIST role-link id ID #IMPLIED>
<!ATTLIST role-name id ID #IMPLIED>
<!ATTLIST run-as id ID #IMPLIED>
<!ATTLIST security-constraint id ID #IMPLIED>
<!ATTLIST security-role id ID #IMPLIED>
<!ATTLIST security-role-ref id ID #IMPLIED>
<!ATTLIST servlet id ID #IMPLIED>
<!ATTLIST servlet-class id ID #IMPLIED>
<!ATTLIST servlet-mapping id ID #IMPLIED>
<!ATTLIST servlet-name id ID #IMPLIED>
<!ATTLIST session-config id ID #IMPLIED>
```

```
<!ATTLIST session-timeout id ID #IMPLIED>
<!ATTLIST small-icon id ID #IMPLIED>
<!ATTLIST taglib id ID #IMPLIED>
<!ATTLIST taglib-location id ID #IMPLIED>
<!ATTLIST taglib-uri id ID #IMPLIED>
<!ATTLIST transport-guarantee id ID #IMPLIED>
<!ATTLIST url-pattern id ID #IMPLIED>
<!ATTLIST user-data-constraint id ID #IMPLIED>
<!ATTLIST web-app id ID #IMPLIED>
<!ATTLIST web-resource-collection id ID #IMPLIED>
<!ATTLIST web-resource-name id ID #IMPLIED>
<!ATTLIST welcome-file id ID #IMPLIED>
<!ATTLIST welcome-file-list id ID #IMPLIED>
<!-- This file contains the orion-specific configuration for a web-application. The
path to the file is located at
ORION_HOME/application-deployments/deploymentName/warname(.war)/orion-web.xml or
(web-app-root/)WEB-INF/orion-web.xml if no deployment-directory is specified in
server.xml. -->
<!ELEMENT orion-web-app ( classpath*, context-param-mapping*, mime-mappings*,
virtual-directory*, access-mask?, cluster-config?, servlet-chaining*,
request-tracker*, session-tracking?, resource-ref-mapping*, security-role-mapping*,
env-entry-mapping*, ejb-ref-mapping*, expiration-setting*, web-app?, jazn-web-app?,
web-app-class-loader?)>
<!ATTLIST orion-web-app autoreload-jsp-beans (true|false) "true"
autoreload-jsp-pages (true|false) "true"
default-buffer-size CDATA "2048"
default-charset %CHARSET; "iso-8859-1"
default-mime-type CDATA #IMPLIED
deployment-version CDATA #IMPLIED
development (true|false) "false"
directory-browsing (allow|deny) "deny"
file-modification-check-interval %NUMBER; "1000"
internationalize-resources (true|false) "false"
jsp-cache-directory CDATA #IMPLIED
jsp-cache-tlds (true|fase) "true"
jsp-taglib-locations CDATA #IMPLIED
jsp-print-null (true|false) "true"
jsp-timeout %NUMBER; "0 (never)"
simple-jsp-mapping (true|false) "false"
enable-jsp-dispatcher-shortcut (true|false) "true"
persistence-path CDATA #IMPLIED
servlet-webdir %PATH; "/servlet/"
source-directory CDATA #IMPLIED
temporary-directory CDATA #IMPLIED
>
```

```
<!ENTITY % CLASSNAME "CDATA">
<!-- A group that this security-role-mapping implies. Ie all the members of the
specified group are included in this role. -->
<!ELEMENT group (#PCDATA)>
<!ATTLIST group name CDATA #IMPLIED
>
<!-- A relative (to the application root) or absolute path to a directory where
application state should be stored across restarts. -->
<!ELEMENT persistence (#PCDATA)>
<!ATTLIST persistence path CDATA #IMPLIED
>
<!-- An argument used when invoking the client. -->
<!ELEMENT argument (#PCDATA)>
<!ATTLIST argument value CDATA #IMPLIED
>
<!-- A short description of this component. -->
<!ELEMENT description (#PCDATA)>
<!-- A relative/absolute path to log events to. -->
<!ELEMENT file (#PCDATA)>
<!ATTLIST file path CDATA #IMPLIED
>
<!-- An ODL formated log file. The max-file-size is the maximum number of kilobytes
a single log file is allowed to grow to. The max-directory-size is the maximum
number of kilobytes that the directory is allowed to contain. -->
<!ELEMENT odl (#PCDATA)>
<!ATTLIST odl path CDATA #REQUIRED max-file-size CDATA #IMPLIED max-directory-size
CDATA #IMPLIED>
<!-- A ejb-jar module of the application. -->
<!ELEMENT ejb-module (#PCDATA)>
<!ATTLIST ejb-module path CDATA #IMPLIED
remote (true|false) "false"
>
<!-- A relative/absolute path/URL to a directory or a .jar/.zip to add as a
library-path for this server. Directories are scanned for jars/zips to include at
startup. -->
<!ELEMENT library (#PCDATA)>
<!ATTLIST library path CDATA #IMPLIED
>
<!-- The read-access policy. -->
<!ELEMENT read-access (namespace-resource)>
<!-- A e-mail address to log events to. A valid mail-session also needs to be
specified if this option is used. -->
<!ELEMENT mail (#PCDATA)>
<!ATTLIST mail address CDATA #IMPLIED
>
```

```
<!-- A list of arguments to used when invoking the app-client if starting it
in-process (auto-start="true"). -->
<!ELEMENT arguments (argument*)>
<!-- Namespace (naming context) security policy for RMI clients. -->
<!ELEMENT namespace-access (read-access, write-access)>
<!-- Contains a name/value pair initialization param. -->
<!ELEMENT property (#PCDATA)>
<!ATTLIST property name CDATA #IMPLIED
value CDATA #IMPLIED
>
<!-- -->
<!ELEMENT data-sources (#PCDATA)>
<!ATTLIST data-sources path CDATA #IMPLIED
>
<!-- The write-access policy. -->
<!ELEMENT write-access (namespace-resource)>
<!-- An application-client module of the application. An app-client is a GUI or
console-based standalone client that interracts with the server. -->
<!ELEMENT client-module (arguments?)>
<!ATTLIST client-module auto-start (true|false) "false"
deployment-time CDATA #IMPLIED
path CDATA #IMPLIED
user CDATA #IMPLIED
>
<!-- A user that this security-role-mapping implies. -->
<!ELEMENT user (#PCDATA)>
<!ATTLIST user name CDATA #IMPLIED
>
<!-- Specifies an optional user-manager to use, example user-managers are
com.evermind.sql.DataSourceUserManager,
com.evermind.ejb.EJBUserManager, etc... Used to integrate existing systems and
provide custom user-managers for
web-applications. -->
<!ELEMENT user-manager (description?, property*)>
<!ATTLIST user-manager class %CLASSNAME; #IMPLIED
display-name CDATA #IMPLIED
>
<!-- An orion-ejb-jar.xml file contains the deploy-time info for an application.
It is located in
ORION_HOME/application-deployments/deploymentName/orion-application.xml after
deployment and META-INF/orion-application.xml below the application root if bundled
with the application or if no deployment-directory is specified in server.xml. If
using deployment-directory (which is the default) the bundled version will be copied
to the deployment location if and only if no file exists at that location. It is
used to specify initial (first time) deployment properties.
After each deployment the deployment file is reformatted/augmented/altered by the
server to add any new/missing info to it. -->
```

```
<!ELEMENT orion-application
(ejb-module*,web-module*,client-module*,commit-coordinator?,security-role-mapping*,
persistence?, library*, principals?, mail-session*, user-manager?, log?, jazn?,
data-sources?, connectors?, resource-provider*, namespace-access?,
jndi-clustering?)>
<!ATTLIST orion-application autocreate-tables (true|false) "true"
autodelete-tables (true|false) "false"
default-data-source CDATA #IMPLIED
deployment-version CDATA #IMPLIED
treat-zero-as-null (true|false) "false"
>
<!-- A web-application module of the application. Each web-application can be
installed on any site and in any context on those sites (for instance
http://www.myserver.com/myapp/). -->
<!ELEMENT web-module (#PCDATA)>
<!ATTLIST web-module id CDATA #IMPLIED
path CDATA #IMPLIED
>
<!-- -->
<!ELEMENT principals (#PCDATA)>
<!ATTLIST principals path CDATA #IMPLIED
>
<!-- Logging settings. -->
<!ELEMENT log (file*, mail*, odl*)>
<!-- The runtime mapping (to groups and users) of a role. Maps to a security-role of
the same name in the assembly descriptor. -->
<!ELEMENT security-role-mapping (group*, user*)>
<!ATTLIST security-role-mapping impliesAll CDATA #IMPLIED
name CDATA #IMPLIED
>
<!-- The session SMTP-server host (if using SMTP). -->
<!ELEMENT mail-session (description?, property*)>
<!ATTLIST mail-session location CDATA #IMPLIED
smtp-host CDATA #IMPLIED
username CDATA #IMPLIED
password CDATA #IMPLIED
>
<!-- A resource with a specific security setting. -->
<!ELEMENT namespace-resource (security-role-mapping)>
<!ATTLIST namespace-resource root CDATA #REQUIRED
>
<!-- -->
<!ELEMENT connectors (#PCDATA)>
<!ATTLIST connectors path CDATA #IMPLIED
>
<!-- JAZN configuration -->
<!ELEMENT jazn-web-app (#PCDATA)>
```

```
<!ATTLIST jazn-web-app auth-method CDATA ""
runas-mode (true | false) "false"
doasprivileged-mode (true | false) "true"
>
<!-- -->
<!ELEMENT jazn (property*, jazn-web-app?)>
<!ATTLIST jazn provider (XML | LDAP) "XML"
location CDATA #IMPLIED
default-realm CDATA #IMPLIED
persistence (NONE | ALL | VM_EXIT) "VM_EXIT"
config CDATA ""
>
<!ELEMENT jndi-clustering (#PCDATA)>
<!ATTLIST jndi-clustering enabled (true | false) "true" >
<!-- -->
<!ELEMENT commit-coordinator (commit-class, property*)>
<!-- -->
<!ELEMENT commit-class (#PCDATA)>
<!ATTLIST commit-class class %CLASSNAME; #IMPLIED
>
<!-- Specifies a resource-provider to plug-in, ie.
com.evermind.server.deployment.ContextScanningResourceProvider -->
<!ELEMENT resource-provider (description?, property+)>
<!ATTLIST resource-provider class %CLASSNAME; #IMPLIED
name CDATA #IMPLIED
>
<!-- Specifies a user manager for use access to security sensitive strings.
If no element is present, then the UserManager for the application
itself will become used.
-->
<!ELEMENT password-manager (principals?, jazn?, user-manager?)>
```

索引

記号

A

C

D

E

G

H

I

J

K

L

M

N

O

P

R

S

U

W

X

あ

い

え

か

き

く

け

こ

さ

し

せ

そ

た

ち

て

と

な

に

は

ひ

ふ

ほ

ま

ゆ

り

れ

ろ