Oracle® Application Server Containers for J2EE

セキュリティ・ガイド

10*g* リリース 2（10.1.2）

部品番号 : B15768-02

2005 年 10 月

ORACLE®

Oracle Application Server Containers for J2EE セキュリティ・ガイド, 10*g* リリース 2（10.1.2）

部品番号 : B15768-02

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

原本部品番号 : B14013-02

原本著者 : Elizabeth Hanes Perry

原本協力者 : Brian Wright, Ganesh Kirti, Raymond Ng, Rachel Chan, Nithya Muralidharan, Moushmi Banerjee, Bill Bathurst, Tom Snyder, Jeff Trent, Cania Lee Chung

目次

3 OC4J セキュリティの概要

4 全体的なセキュリティの構成

9 外部 LDAP プロバイダの構成

10 カスタム・ログイン・モジュール

11 OC4J と SSL の構成

12 EJB のセキュリティの構成

13 クライアント接続用 Oracle HTTPS

14 パスワード管理

15 CSIv2 の構成

索引

例リスト

図リスト

表リスト

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。Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider について詳しく説明します。また、EJB、J2EE Connector Architecture、SSL および CSIv2 など、個々の J2EE 機能のセキュリティ上の影響についても説明します。

ドキュメントのアクセシビリティについて

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

ドキュメント内のサンプル・コードのアクセシビリティについて

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

外部 Web サイトのドキュメントのアクセシビリティについて

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

関連ドキュメント

詳細は、Oracle Java プラットフォーム・グループから入手できる次の Oracle ドキュメントを参照してください。

- 『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』

  このマニュアルには、OC4J の概要と一般情報が記載されています。サーブレット、JSP ページ、EJB に関する入門章と、一般的な構成およびデプロイ方法の説明が含まれます。

- 『Oracle Application Server Containers for J2EE スタンドアロン・ユーザーズ・ガイド』

  このユーザーズ・ガイドは、特にスタンドアロン・バージョンの OC4J を対象としており、OTN からスタンドアロン・バージョンをダウンロードした場合に入手できます。スタンドアロン OC4J は、開発環境では使用されますが、本番環境では通常使用されません。

- 『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』

  このマニュアルには、サーブレット開発者を対象に、OC4J でのサーブレットおよびサーブレット・コンテナの使用方法に関する情報が記載されています。基本的なサーブレット開発、JDBC と EJB の使用方法、アプリケーションの構築とデプロイ、サーブレットと Web サイトの構成に関する説明が含まれます。このマニュアルでは、開発環境のスタンドアロン OC4J と本番環境の Oracle Application Server での OC4J についての検討事項を参照できます。

- 『Oracle Application Server Containers for J2EE JavaServer Pages 開発者ガイド』

  このマニュアルには、OC4J で独自ページを実行する JSP 開発者向けの情報が記載されています。JSP 標準の概要とプログラミング上の検討事項に加え、付加価値の高い Oracle 機能の説明と、OC4J 環境を導入するための手順が含まれます。

- 『Oracle Application Server Containers for J2EE JSP タグ・ライブラリおよびユーティリティ・リファレンス』

  このマニュアルには、タグ・ライブラリ、JavaBeans、および OC4J で提供されるその他の Java ユーティリティについて、概念的な情報と、構文および使用方法の詳細情報が記載されています。また、他の Oracle 製品グループのタグ・ライブラリの概要も含まれます。

- 『Oracle Application Server Containers for J2EE サービス・ガイド』

  このマニュアルには、JTA、JNDI、JMS、JAAS、Oracle Application Server Java Object Cache など、OC4J に付属の標準ベースの Java サービスに関する情報が記載されています。

Oracle Java プラットフォーム・グループでは、次のドキュメントも入手できます。

- 『Oracle Database Java 開発者ガイド』
- 『Oracle Database JDBC 開発者ガイドおよびリファレンス』
- 『Oracle Database JPublisher ユーザーズ・ガイド』

Oracle Application Server グループでは、次のドキュメントを入手できます。

- 『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 Internet Directory 管理者ガイド』
- 『Oracle Application Server 管理者ガイド』
- 『Oracle Application Server セキュリティ・ガイド』
- 『Oracle Application Server パフォーマンス・ガイド』
- 『Oracle Enterprise Manager 概要』
- 『Oracle HTTP Server 管理者ガイド』
- 『Oracle Application Server グローバリゼーション・サポート・ガイド』
- 『Oracle Application Server Web Cache 管理者ガイド』
- 『Oracle Application Server Web Services 開発者ガイド』
- 『Oracle Application Server アップグレードおよび互換性ガイド』

Oracle JDeveloper グループでは、次のドキュメントを入手できます。

- Oracle JDeveloper オンライン・ヘルプ
- Oracle Technology Network にある Oracle Jdeveloper のドキュメント

  `http://www.oracle.com/technology/products/jdev/index.html`

Oracle サーバー・テクノロジ・グループでは、次のドキュメントを入手できます。

- 『Oracle XML Developer's Kit プログラマーズ・ガイド』
- 『Oracle XML API リファレンス』
- 『Oracle Database アプリケーション開発者ガイド - 基礎編』
- 『PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』
- 『Oracle Database SQL リファレンス』
- 『Oracle Database Net Services 管理者ガイド』
- 『Oracle Advanced Security 管理者ガイド』
- 『Oracle Database リファレンス』

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、次の URL の Java Authentication and Authorization Service（JAAS）の Web サイト。

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

表記規則

このマニュアルでは次の表記規則を使用します。

規則	意味
太字	太字は、アクションに関連するグラフィカル・ユーザー・インタフェース要素、または本文中で定義されている用語および用語集に記載されている用語を示します。
固定幅フォント	固定幅フォントは、段落内のコマンド、URL、例に含まれるコード、画面に表示されるテキストまたはユーザーが入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

オラクル社カスタマ・サポート・センター

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

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

製品マニュアル

製品のマニュアルは、次の URL にあります。

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

研修およびトレーニング

研修に関する情報とスケジュールは、次の URL で入手できます。

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

その他の情報

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

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

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

1

概要

この章には、次の項目が含まれます。

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 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・モデルは Oracle Application Server Java Authentication and Authorization Service（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。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。

関連資料 :

- Java 2 セキュリティのチュートリアルは、次の URL を参照してください。

  `http://java.sun.com/docs/books/tutorial/security1.2/index.html`

- Java 2 セキュリティの詳細は、次の URL を参照してください。

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

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 1-1 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 1-1 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。

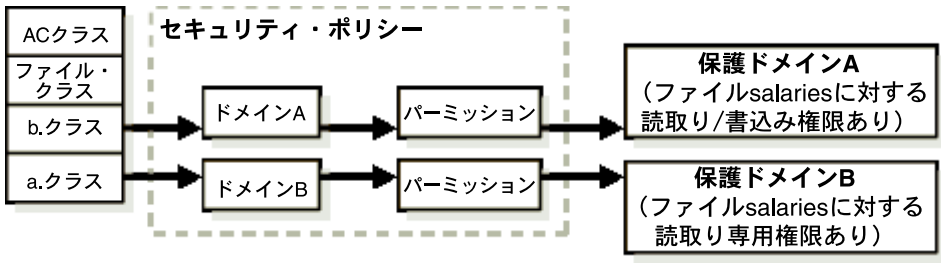
実行時には、スタック・イントロスペクションにより認可チェックが実行されます。この認可チェックでは、ランタイム・スタックが検討され、スタック上のクラスに関連した保護ドメインに基づいてパーミッションがチェックされます。これは、通常、次のどちらかのコールによりトリガーされます。

- `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 1-1 に、実行時の認可チェックの基本モデルを示します。

図 1-1 Java 2 セキュリティ・モデル



関連資料：

- Sun 社の Java ドキュメントは、次の URL を参照してください。

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

OracleAS JAAS Provider のパーミッション・クラス

表 1-2 に、OracleAS 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。

関連資料：

- 次の表に示すクラスの詳細は、OracleAS JAAS Provider の Javadoc の JAAS Provider API Reference を参照してください。

表 1-2 OracleAS 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。

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、指定されたリクエストを実行します。

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、「誰がサービスにアクセスしようとしているか」という質問に対処します。システムやアプリケーションで最も重要なことは、アプリケーションにアクセスしようとしているエンティティまたはコール元の ID がセキュアな方法で識別されることを保証することです。多層アプリケーションでのエンティティまたはコール元には、ユーザー、ビジネス・アプリケーション、ホストあるいは別のエンティティのかわりに機能する（ふりをする）エンティティなどがあります。

  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、3-4 ページの「認証環境」を参照してください。

- 認可は、「どのコンポーネントが提供する何のサービスに誰がアクセスできるか」という質問に対処します。各種のビジネスクリティカルなサービスおよびリソースに対する何百万ものユーザーによるアクセスを管理する必要がある大企業の場合、重要なのは、スケーラブルな認可インフラストラクチャが、ユーザーおよびアプリケーションのプロビジョニングを処理するのために正しく機能していることです。残念なことに、認可の性質が複雑であることが原因の一端となって大勢の混乱を招き、互換性のないテクノロジと規格が普及しているという現状もあります。

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、暗号化、認証およびデータ整合性により機密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。

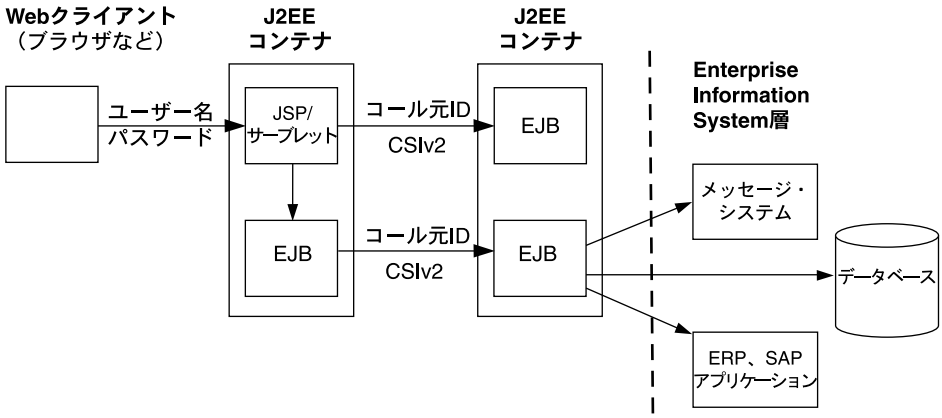
HTTPS

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 1-2 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 1-2 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。OracleAS 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。OracleAS 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、サーブレットおよびアプリケーション・クライアントを作成します。

   OracleAS 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 OracleAS 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。

2

Oracle Application Server での JAAS の概要

この章では、前の章で紹介した OC4J の Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider について詳細に説明します。OracleAS JAAS Provider 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。

OracleAS JAAS Provider

Oracle Application Server では、OracleAS JAAS Provider により JAAS がサポートされます。OracleAS JAAS Provider により、開発者がアプリケーション環境に統合できるユーザー認証、認可および委任の各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 OracleAS 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。OracleAS 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。

  注意： 本番環境では、LDAP ベース・プロバイダを使用することをお薦めします。XML ベース・プロバイダが適しているのは、プロトタイピングの場合のみです。

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。OracleAS 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。OracleAS JAAS Provider により、次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
- アプリケーション
- ポリシーとパーミッション

関連資料：

- 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、2-5 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、`LoginContext.login()` メソッドにより実行されます。認証に成功した場合は、`LoginContext.getSubject()` メソッド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()` メソッド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、`java.security.Principal` インタフェース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。

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 OracleAS 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。OracleAS JAAS Provider から提供される管理 API は、Oracle の拡張機能です。

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

パラメータ	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`

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 `admin` プリンシパルに対して、ログインのパーミッションを付与しています。

```
<jazn-policy>
  <grant>
   <grantee>
     <principals>
      <principal>
        <class>oracle.security.jazn.samples.SampleUser</class>
        <name>admin</name>
      </principal>
    </principals>
   </grantee>
   <permissions>
     <permission>
       <class>oracle.j2ee.server.rmi.RMIPermission</class>
       <name>login</name>
     </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 の構成」を参照してください。

JAAS フレームワークの機能

表 2-2 に、OracleAS 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 OracleAS JAAS Provider の機能

機能	説明	関連項目
認証	■ J2EE アプリケーション環境でログイン認証を使用できるように Oracle Application Server Single Sign-On と統合します。 ■ OracleAS Core または Java Edition など、非 OracleAS Single Sign-On 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 3 章「OC4J セキュリティの概要」
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 4 章「全体的なセキュリティの構成」
認可	■ 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。	2-8 ページの「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 ページの「レルム」
管理	■ コマンドライン・ツール（Admintool）または Oracle Enterprise Manager 10*g* 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。	
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。	3-3 ページの「JAZNUserManager」

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。

関連資料 :

- 詳細は、OracleAS JAAS Provider の Javadoc の JAAS Provider API Reference を参照してください。

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。ただし、このインタフェースは将来のリリースでは廃止される予定です。

関連資料 :

- カスタム UserManager の作成方法は、次の URL を参照してください。

  http://www.oracle.com/technology/sample_code/tech/xml/xmlnews/News_Security.html

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 ■ サード・パーティの LDAP プロバイダを使用するタイプ
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

すべてのアプリケーションに対してデフォルト UserManager を、または特定のアプリケーションに対して 1 つの UserManager を定義する方法は、4-6 ページの「orion-application.xml でのユーザー・マネージャの指定」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、OracleAS JAAS Provider を OC4J のセキュリティ・インフラストラクチャとして活用することです。

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 8 章「XML ベース・プロバイダの構成」を参照してください。

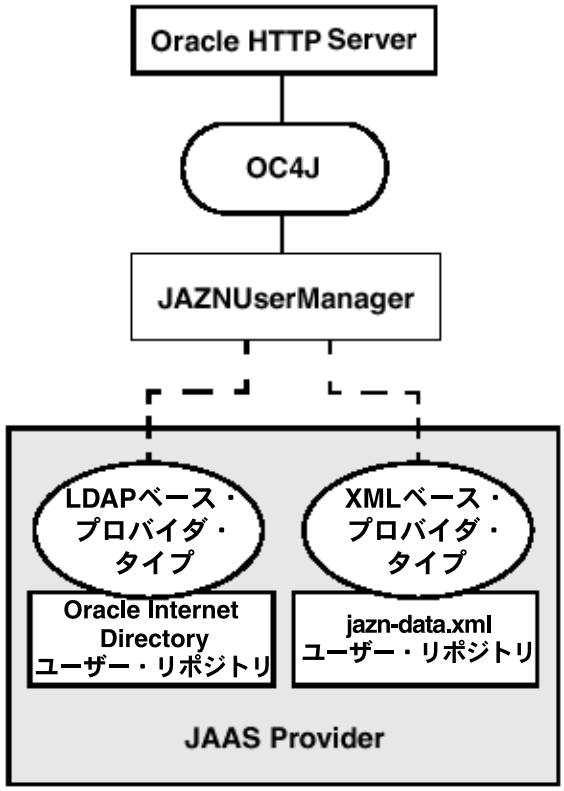
  Enterprise Manager で、JAZN-XML をユーザー・マネージャとして選択します。そのユーザー、ロールおよびグループを、JAZN Admintool を使用して構成します。

- LDAP ベース・プロバイダは、スケーラブルかつセキュアで、OracleAS 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、OracleAS Single Sign-On をサポートする唯一の OracleAS 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 Oracle Delegated Administration Service 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、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・プロバイダはセキュリティ・オプションとして選択できません。LDAP ベース・プロバイダの構成方法は、第 7 章「LDAP ベース・プロバイダの構成」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、下位互換性を維持するためにのみサポートされていますが、将来のリリースではサポートされない予定です。OracleAS JAAS Provider タイプのいずれかを使用することをお薦めします。

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。OracleAS 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	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）OracleAS 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。

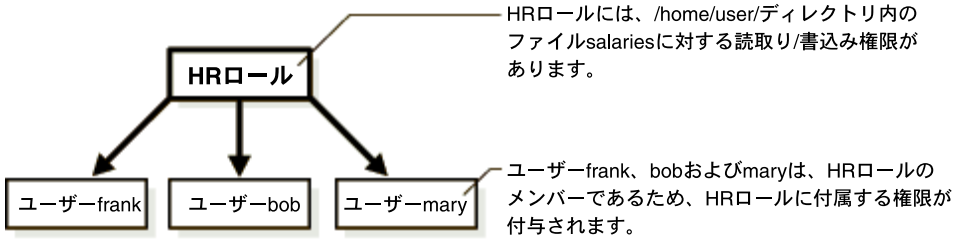
- ロール階層
- ロールのアクティブ化

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()` メソッド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、最小限の権限の原則が守られます。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。

10*g*（9.0.4）以降の変更

- 動的ライブラリのロードを制御する環境変数（Solaris の `LD_LIBRARY_PATH` など）の正しい設定は、現在、*`ORACLE_HOME`*`/lib` ではなく *`ORACLE_HOME`*`/lib32` です。表 2-5 に、詳細な設定内容を示します。

表 2-5 動的ライブラリ・パスの設定

オペレーティング・システム	変数	値
Solaris	`LD_LIBRARY_PATH`	*`ORACLE_HOME`*`/lib32`
	`LD_LIBRARY_PATH_64`	*`ORACLE_HOME`*`/lib`
HP/UX	`SHLIB_PATH`	*`ORACLE_HOME`*`/lib32`
	`LD_LIBRARY_PATH`	*`ORACLE_HOME`*`/lib`
AIX 32 ビット・アプリケーション	`LIBPATH`	*`ORACLE_HOME`*`/lib32`
	`LD_LIBRARY_PATH`	null
AIX 64 ビット・アプリケーション	`LIBPATH`	*`ORACLE_HOME`*`/lib`
	`LD_LIBRARY_PATH`	null
Windows	該当なし	該当なし

- Java Development Kit 1.3 のデフォルト・インストールには、JAAS サポートは組み込まれません。JAAS を JDK 1.3 で使用するには、Sun 社の Web サイト（`http://java.sun.com/products/jaas/index-10.html`）から JAAS 1.0_01 をダウンロードし、インストールおよびデプロイの指示に従う必要があります。

注意： JDK 1.3 を使用してアプリケーションを開発する場合、JDK クラス・ローダーの問題に注意する必要があります。クラス・ローダーでは、カスタム・ログイン・モジュールを標準拡張機能として *ORACLE_HOME*/jre/lib/ext ディレクトリに含む JAR をデプロイする場合のみ、それらを探すことができます。この問題は、次期リリースで解決される予定です。

- このリリースでは、サード・パーティの LDAP プロバイダがサポートされます。詳細は、第 9 章「外部 LDAP プロバイダの構成」を参照してください。
- このリリースでは、デフォルト・ファイル（jazn.security.props）が *ORACLE_HOME*/j2ee/home/startup ディレクトリに提供されます。このファイルは、JAAS ログイン構成およびポリシーに使用される OracleAS JAAS Provider を指定します。これらのプロパティは、OC4J の起動時にデフォルトで設定されるため、ほとんどの場合、これらのプロパティの設定について憂慮する必要はありません。詳細は、4-5 ページの「代替ポリシー・プロバイダの指定（オプション）」を参照してください。
- カスタムの UserManager クラスは、このリリースでもサポートされていますが、将来のリリースでは廃止される予定です。
- principals.xml ファイルは、将来のリリースではサポートされない予定です。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。移行手順は、8-5 ページの「principals.xml ファイルからのプリンシパルの移植」を参照してください。
- SSL クライアント証明書を取得するためのインタフェースが変更されています。現在は、次の方法を使用します。

```
servletRequest.getAttribute("javax.servlet.request.X509Certificate");
```

次の方法は使用しません。

```
servletRequest.getAttribute("javax.security.cert.X509certificate");
```

3

OC4J セキュリティの概要

この章では、OC4J 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

OC4J セキュリティ・アーキテクチャのコンポーネントは次のとおりです。

- OracleAS 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。OracleAS JAAS Provider では、次の 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（Oracle Application Server Infrastructure がインストールされている場合にのみ使用可能）
- JAAS ログイン・モジュール（`RealmLoginModule`、サード・パーティのログイン・モジュールおよびカスタムのログイン・モジュールなど）

関連資料：

- 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 のインストレーション・ガイドを参照してください。

開発およびデプロイ時のセキュリティ上の考慮事項

OracleAS 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、OracleAS 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。

デプロイ

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。
- OracleAS JAAS Provider のリポジトリとして XML フラット・ファイルと Oracle Internet Directory（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。

  関連資料：

  - これらの意思決定とその実装の詳細は、第 6 章「アプリケーションのデプロイ時におけるセキュリティ上の考慮事項」を参照してください。
  - デプロイの詳細は、『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 OracleAS JAAS Provider

OC4J 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 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

OracleAS 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 OC4J および OracleAS Single Sign-On と統合されます。この統合には、次のメリットがあります。

- 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）
- OracleAS Single Sign-On の完全サポート
- カスタムのログイン・モジュールのサポート

JAZNUserManager

OracleAS JAAS Provider は JAZNUserManager を介してサポートされます。JAZNUserManager は、OC4J の UserManager インタフェースの実装であり、次の機能をサポートします。

- 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 OracleAS JAAS Provider と同じ JVM で実行できます。
- OracleAS Single Sign-On と OC4J の統合。
- 非 OracleAS Single Sign-On 環境での RealmLoginModule の統合。
- カスタムの JAAS ログイン・モジュールのサポート。
- ID の伝播。

認証環境

OracleAS JAAS Provider は、J2EE アプリケーションの各種ログイン認証環境と統合されます。

- **OracleAS Single Sign-On**

  ログインの認証に OracleAS Single Sign-On を使用します。

- **SSL**
  - クライアント証明書ベース認証に Secure Sockets Layer を使用します。
  - ログイン認証にログイン・モジュール（`RealmLoginModule` など）を使用します。
- **Basic 認証**
  - OracleAS 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します。
  - ログイン認証にログイン・モジュール（`RealmLoginModule` など）を使用します。
- **フォームベース認証**

  ユーザーが保護リソースへのアクセスを試行すると、OC4J はユーザーがすでに認証されているかどうかをチェックします。認証されていない場合、OC4J はアプリケーション固有のログイン画面を表示し、ユーザー名とパスワードを要求します。

- **Digest 認証**

  ダイジェスト・メカニズム（標準仕様だが J2EE のオプション機能）では、クライアントが自己認証を行うために示す資格証明が、MD5 ダイジェストで構成されます。これは、リクエスト・メッセージで送信されます。

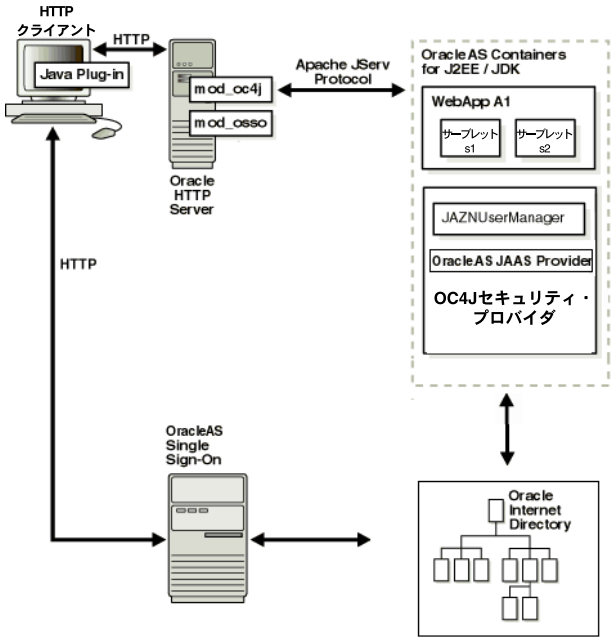
次の各項では、OracleAS JAAS Provider とこれらの各認証タイプがどのように統合されるかについて説明します。

J2EE アプリケーションでの OracleAS Single Sign-On の有効化

OracleAS Single Sign-On 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アプリケーションにアクセスできます。図 3-1 に、OracleAS Single Sign-On 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。

注意： 別の方法として、特に OC4J 環境を対象とするシングル・サインオン実装があります。3-10 ページの「軽量な J2EE のシングル・サインオン」を参照してください。

図 3-1 OracleAS Single Sign-On と J2EE 環境



OracleAS Single Sign-On 対応 J2EE 環境 : 代表的な使用例

この項では、OracleAS Single Sign-On 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、OC4J（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。
   a. WebApp A1 アプリケーションで HTTP クライアントの認証用に Web ベースの OracleAS Single Sign-On が必要かどうかを判別します。
   b. HTTP クライアント・リクエストを Web ベースの OracleAS 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、ユーザー名とパスワード、またはユーザー証明書により OracleAS Single Sign-On から認証を受けます。その後、OracleAS Single Sign-On が次を実行します。
   a. 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。
   b. OracleAS Single Sign-On の Cookie（ユーザーの識別名とレルムを含む）を設定します。
   c. WebApp A1 アプリケーション（OC4J 内）にリダイレクトします。
4. OracleAS JAAS Provider が OracleAS Single Sign-On ユーザーを取得します。

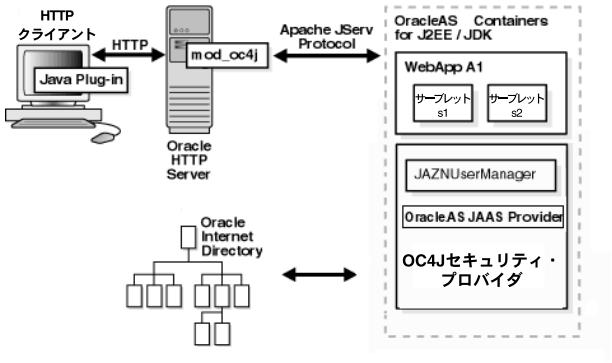
関連資料 :

- OracleAS Single Sign-On の詳細は、『Oracle Application Server Single Sign-On 管理者ガイド』を参照してください。

OracleAS 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 3-2 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。

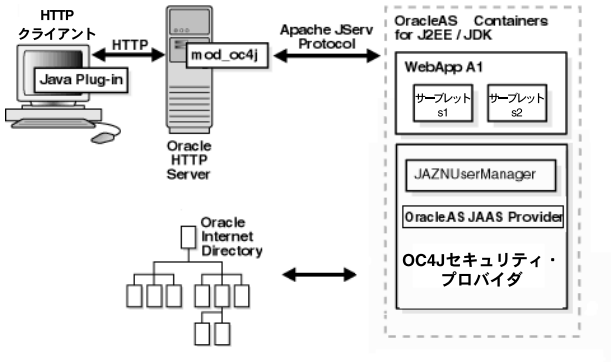
図 3-2 SSL 対応 J2EE 環境での Oracle コンポーネントの統合



OracleAS JAAS Provider と Basic 認証の統合

Basic 認証は OracleAS Single Sign-On をバイパスします。図 3-3 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 3-3 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、OracleAS 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、OC4J（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. ユーザー資格証明が必要になるたびに、OC4J が `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 OracleAS 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。

3. OracleAS JAAS Provider がユーザーを取得します。

関連資料：

- J2EE の詳細は、次の URL にある 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 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。Oracle Application Server では、J2EE 環境の認証は次のいずれかにより実行されます。

- OracleAS Single Sign-On（OracleAS Single Sign-On 環境の場合）または OracleAS JAAS Provider の `RealmLoginModule` または他のログイン・モジュール（非 OracleAS Single Sign-On 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、OC4J 内で JAAS サブジェクトを設定します。

  注意： スタンドアロン OC4J には、別のシングル・サインオン実装があります。3-10 ページの「軽量な J2EE のシングル・サインオン」を参照してください。

- 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`

  注意： 開発者提供の `UserManager` は、廃止されており、将来のリリースではサポートされない予定です。

認証済 ID を使用した実行

`JAZNUserManager` は、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 に関連付けられたパーミッションとロールで実行できるように構成することができます。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、3-3 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

> **注意：** 戻されるプリンシパルは、java.security.Principal インタフェースを拡張する oracle.j2ee.security.User インタフェースのインスタンスです。

- getAttribute("javax.servlet.request.X509certificate"): SSL クライアント証明書

J2EE 環境での認可

認可とは、認証済ユーザーに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 ベースの認可を有効化します。

認可は、次を介して実行されます。

- JAZNUserManager
- JAAS 認可に基づくメソッド
  - 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：

- 4-11 ページの「J2EE 認可の構成」

セキュリティ・ロールのマッピング

J2EE 環境でセキュアな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。OC4J 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
- デプロイ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）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、定義済のアクセス権をそのプリンシパルに割り当てます。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>
```

デプロイ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、developer が jazn-data.xml ファイルにロール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

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 OC4J の 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> エントリが OracleAS 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。

軽量な J2EE のシングル・サインオン

この項では、OC4J 用の代替オプションである軽量なシングル・サインオン（SSO）の使用方法について説明します。Oracle Application Server には、この章で先に説明したとおり OracleAS Single Sign-On という SSO コンポーネントがすでに存在しますが、その機能を使用するには複数のコンポーネントのインストールが必要です。OC4J では、より単純な代替オプションが必要とされていました。

この項には、次の項目が含まれます。

- 軽量な J2EE のシングル・サインオンの概要
- 軽量な J2EE のシングル・サインオンの構成
- 軽量な J2EE のシングル・サインオンの有効化

軽量な J2EE のシングル・サインオンの概要

クライアントが特定の Web アプリケーションで認証されると、SSO を使用して、同じ J2EE アプリケーション（EAR ファイル）内の別の Web アプリケーションに対しても現在のセキュリティ資格証明を適用できます。OracleAS Single Sign-On は、完全な Oracle Application Server 環境（3-4 ページの「J2EE アプリケーションでの OracleAS Single Sign-On の有効化」を参照）でこの機能を実現しますが、現在はより軽量な SSO の代替オプションも OC4J で使用できます。

Web アプリケーションのセキュリティ資格証明は、通常、HTTP クライアント・セッションに格納されます。ただし、サーブレットの仕様では、HTTP セッションの拡張が許可されるのは単一の Web アプリケーションのスコープ内のみであるため、OC4J では、セッション外部のコンポーネントを使用して複数の Web アプリケーション全体で J2EE の SSO を管理する必要があります。軽量な J2EE のシングル・サインオンでこれを実装するために選択されたコンポーネントが、ステートフル・セッション EJB です。

J2EE のシングル・サインオンを有効にすると、OC4J では、ホーム・インタフェース、Bean インタフェースおよび実装クラスを含むステートフル Session Bean が自動的に作成されます。J2EE のシングル・サインオンに加わっている Web アプリケーションで、一度ユーザー認証に成功すると、そのユーザーは OC4J によって Bean に登録されます。

OC4J では、J2EE SSO Cookie も作成されます。Cookie は、J2EE のシングル・サインオンで認証されたユーザーの存在を Web アプリケーションに示します。

次のことにも留意してください。

- 軽量な J2EE の SSO 実装は、EJB レプリケーションが 3-12 ページの「特別な orion-ejb-jar.xml ファイルの内容」のように J2EE の SSO の Bean で有効化されている場合、分散環境で動作します。
- J2EE の SSO 実装は、スタンドアロン OC4J で動作します。次の条件がすべて満たされる場合は、完全な Oracle Application Server（OPMN 管理）環境でも動作します。
  - web.xml ファイルの <web-app> の <distributable/> サブ要素を通じて、関連する各 Web アプリケーションでセッション・レプリケーションが有効化されている場合
  - J2EE の SSO の Bean で EJB レプリケーションが有効化されている場合
  - 単一の OC4J クラスタ・アイランドのみを使用している場合
- J2EE の SSO 実装は、わずかながらパフォーマンスに影響を与えます。
- 検索されたユーザーが、制限されたリソースにアクセスするための十分な権限を保持している場合、OC4J ではそのリソースが表示されます。それ以外の場合は、フォーム認証ページが表示され、十分な権限を持つユーザーをログイン用に指定できます。

- HTTP セッションが無効になった場合、ユーザー名は設定されません。次にリソースのアクセスが試行されると、フォーム認証ページを通じて再度ログインが強制されます。

軽量な J2EE のシングル・サインオンの構成

軽量な J2EE のシングル・サインオンのリリース 10.1.2 での実装では、使用する EAR ファイルに OC4JINTERNAL_httpSession.jar という特別な JAR ファイルを含める必要があります。この JAR ファイルには、次の XML ファイルが含まれている必要があります。また、各 XML ファイルには、続けて記載されている特定の内容が含まれている必要があります。

- /META-INF/ejb-jar.xml
- /META-INF/orion-ejb-jar.xml

特別な ejb-jar.xml ファイルの内容

ejb-jar.xml ファイルには、次の内容が含まれている必要があります。

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
   <display-name>stateful, ejb-jar:</display-name>
   <enterprise-beans>
      <session>
         <display-name>WebAppsClusterBean</display-name>
         <ejb-name>WebAppsClusterBean</ejb-name>
         <home>
            oracle.oc4j.server.ejb.httpSession.HttpSessionSharingSFSBHome
         </home>
         <remote>
            oracle.oc4j.server.ejb.httpSession.HttpSessionSharingSFSB
         </remote>
         <ejb-class>
            oracle.oc4j.server.ejb.httpSession.HttpSessionSharingSFSBBean
         </ejb-class>
         <session-type>Stateful</session-type>
         <transaction-type>Container</transaction-type>
      </session>
   </enterprise-beans>
   <assembly-descriptor>
      <container-transaction>
         <method>
            <ejb-name>WebAppsClusterBean</ejb-name>
            <method-name>*</method-name>
         </method>
         <trans-attribute>Supports</trans-attribute>
      </container-transaction>
      <security-role>
         <role-name>users</role-name>
      </security-role>
   </assembly-descriptor>
</ejb-jar>
```

注意： <display-name> 要素は必須ではありませんが、慣例として記述します。他のすべての要素は、このとおりの設定にする必要があります。

特別な orion-ejb-jar.xml ファイルの内容

非分散環境では、orion-ejb-jar.xml ファイルに次の内容が含まれている必要があります。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE orion-ejb-jar PUBLIC "-//Evermind//DTD Enterprise JavaBeans 1.1
 runtime//EN" "http://xmlns.oracle.com/ias/dtds/orion-ejb-jar.dtd">
<orion-ejb-jar>
   <enterprise-beans>
      <session-deployment name="WebAppsClusterBean"
                          location="webAppsCluster/WebAppsClusterBean"
                          idletime="600" copy-by-value="false"  />
   </enterprise-beans>
   <assembly-descriptor>
      <security-role-mapping name="users" />
      <default-method-access>
         <security-role-mapping name="default-ejb-caller-role"
                                impliesAll="true" />
      </default-method-access>
   </assembly-descriptor>
</orion-ejb-jar>
```

分散（クラスタ化）環境では、次のように属性設定 replication="EndOfCall" を <session-deployment> 要素に追加する必要があります。

```
      <session-deployment name="WebAppsClusterBean"
                          location="webAppsCluster/WebAppsClusterBean"
                          idletime="600" copy-by-value="false"
                          replication="EndOfCall" />
```

これ以外で分散環境と非分散環境の構成に違いはありません。

注意：

- <security-role-mapping> 要素では、*default-ejb-caller-role* のかわりに適切なロールを使用してください。
- idletime 属性では、ステートフル Session Bean のタイムアウト値を秒単位で指定します。この値は必要に応じて調整できますが、HTTP セッション・タイムアウト値以上とする必要があります。
- その他の設定は変更しないでください。

軽量な J2EE のシングル・サインオンの有効化

軽量な J2EE のシングル・サインオン実装を有効化するには、OC4J の起動時にシステム・プロパティの oracle.application.sso.*appname* を true に設定します。ここで、*appname* は、使用する J2EE アプリケーションの名前です。たとえば、アプリケーション名が petstore の場合、次のように指定します。

```
java ... -Doracle.application.sso.petstore=true ... -jar oc4j.jar
```

4

全体的なセキュリティの構成

この章では、セキュリティ・システム全体の構成に関連するタスクについて説明します。この章には次の項目が含まれます。

- XML ベース・プロバイダまたは LDAP ベース・プロバイダの選択
- jazn.xml、jazn-data.xml および <jazn> 要素の検索
- Admintool の概要
- 代替ポリシー・プロバイダの指定（オプション）
- OracleAS JAAS Provider 設定の指定
- デバッグ・ロギングの有効化
- ユーザー・マネージャの指定
- RealmLoginModule のカスタマイズ
- 認証（auth-method）の指定
- J2EE 認可の構成
- 認証プリンシパルからのレルム名の削除
- サード・パーティの LDAP プロバイダの構成
- EJB RMI クライアント・アクセスの許可
- Java 2 ポリシー・ファイルの作成
- <principals> 要素および principals.xml の使用

XML ベース・プロバイダまたは LDAP ベース・プロバイダの選択

OC4J のインストールの一環として、LDAP ベース・プロバイダと XML ベース・プロバイダのどちらを使用するかを決定します。この項では、その選択方法についてガイドラインを示します。

- **XML ベース・プロバイダ :** 開発環境および小規模なユーザー・コミュニティでデプロイされたアプリケーションでは、XML ベース・プロバイダを使用します。
- **LDAP ベース・プロバイダ :** 本番環境では LDAP ベース・プロバイダを使用します。

  XML ベース・プロバイダと比較して、LDAP ベース・プロバイダの方がセキュリティとパフォーマンスに優れています。集中型 Oracle Internet Directory サーバーは、アプリケーションやユーザーの数が増加するにつれて適切に規模が拡大します。よりよいパフォーマンスのためと、集中的なアカウントの作成および管理、単一パスワード、資格証明管理などの機能を利用するために、本番環境では集中型 Oracle Internet Directory サーバーを利用することをお薦めします。

  LDAP ベース・プロバイダを使用すると、キャッシュ・パラメータを構成して、認証と認可の全体的なパフォーマンスを向上させることができます。Oracle Internet Directory サーバーと OC4J 間にセキュアな通信が必要な場合は、必要なセキュリティ・レベルを使用するようにシステムを構成します。

インフラストラクチャをインストールし、OC4J インスタンスを中間層の Oracle Internet Directory または Oracle Application Server Single Sign-On インスタンスに関連付けると、インストーラは LDAP ベース・プロバイダを自動的に選択します。詳細は、Oracle Application Server のインストレーション・ガイドを参照してください。LDAP ベース・プロバイダを使用するように OC4J を構成する必要がある場合は、『Oracle Application Server 管理者ガイド』の指示を参照してください。

jazn.xml、jazn-data.xml および <jazn> 要素の検索

OracleAS JAAS Provider を構成するには、場合によっては、テキスト・エディタを使用して様々な構成ファイルを編集する必要があります。この項では、構成ファイルおよび `<jazn>` 要素の検索方法について説明します。

jazn.xml の検索

OracleAS 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。`jazn.xml` ファイルは、OracleAS JAAS Provider の構成に使用されます。

OC4J `home` インスタンスの `jazn.xml` ファイル（ブートストラップ jazn.xml ファイル）は、デフォルトでは *`ORACLE_HOME`*`/j2ee/home/config` ディレクトリに存在します。OracleAS JAAS Provider は、OC4J の起動前にこのファイルの情報を読み取ります。特定の設定は、ブートストラップ・ファイルにのみ指定できます。OC4J の起動後に設定の変更が読み取られても、OracleAS JAAS Provider には反映されません。

追加の各 OC4J インスタンスの `jazn.xml` ファイルは、*`ORACLE_HOME`*`/j2ee/`*`instance_name`*`/config` ディレクトリに存在します。

関連項目 :

- 16-2 ページの「jazn.xml の検索場所」

jazn-data.xml の検索

jazn-data.xml ファイルは、XML ベースの JAAS Provider 用データストアです。デフォルトでは、各 OC4J インスタンスの jazn-data.xml ファイルは、*ORACLE_HOME*/j2ee/*instance_name*/config ディレクトリに存在すると見なされます。OC4J home インスタンスの jazn-data.xml ファイル（ブートストラップ jazn-data.xml ファイル）は、*ORACLE_HOME*/j2ee/home/config ディレクトリに存在します。

jazn.xml の <jazn> 要素で、次のように jazn-data.xml の代替パスを指定できます。

```
<jazn provider="xml" location="pathname">
   ...
</jazn>
```

<jazn> 要素の検索

<jazn> 要素は、OracleAS JAAS Provider の構成に使用されます。ほとんどの場合、<jazn> 要素は、次の 2 箇所のいずれかにあります。

1. グローバル構成用のグローバル application.xml
2. アプリケーション固有の orion-application.xml

また、<jazn> 要素は、仮想マシンのプロパティの構成に使用される場合、jazn.xml ファイル内に配置されることもあります。詳細は、5-2 ページの「インスタンス・レベルの jazn.xml ファイル」を参照してください。

Admintool の概要

この項では、JAZN Admintool を理解し、使用するために必要な基本情報について説明します。JAZN Admintool は、jazn-data.xml に格納されたユーザー・アカウントの管理に使用できる軽量ツールです。LDAP ベース・プロバイダのアカウントを管理するには、Oracle Delegated Administration Service で提供されるツールを使用します。

注意： Admintool による変更は、次に OC4J が再起動されるまで有効になりません。

Admintool の前提条件

JAZN Admintool を使用すると、デフォルトでは、OC4J ホーム・インスタンスの config ディレクトリ下の jazn-data.xml ファイルが編集されます。jazn-data.xml の検索の詳細は、4-3 ページの「jazn-data.xml の検索」を参照してください。admin 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（ias_admin）のパスワードと同じ値に設定されます。

Admintool を LDAP ベース・プロバイダで使用するには、環境設定を正しく設定する必要があります。詳細は 7-2 ページの「LDAP 使用の準備」を参照してください。

自己認証

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。Admintool のプロンプトで次のようにユーザー名とパスワードを入力し、認証を行います。

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

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。ツールを実行するユーザーは、Admintool の操作を Oracle Internet Directory サーバーに対して実行できます。つまり、OC4J で LDAP ベース・プロバイダが使用される本番マシンへのアクセスを保護することが必要不可欠です。LDAP ベース・プロバイダの使用時には、`-user` および `-password` オプションを指定しても無視されます。
- コマンドラインの `-user` および `-password` を使用してユーザー名とパスワードを指定することも可能ですが、この方法は使用しないことをお薦めします。コマンドラインでパスワードを指定すると、セキュリティ上の脆弱性が生じます。

クラスタ化サポートの追加

-clustersupport *oracle_home*

このオプションを指定すると、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 *ORACLE_HOME* 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
```

jazn-data.xml での Admintool のログイン・モジュールの指定

JAZN Admintool でユーザーの認証に使用する `LoginModule` を指定するには、`<login-modules>` 要素を `jazn-data.xml` の `<application>` 要素に追加する必要があります。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 JAZN Admintool を実行すると、デフォルトのオプションで RealmLoginModule が使用されます。デフォルトのオプションは、表 4-2「RealmLoginModule のオプション」を参照してください。

代替ポリシー・プロバイダの指定（オプション）

Oracle Application Server に同梱の Java Virtual Machine を使用する場合は、OracleAS 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。これは、デフォルトでは、Sun 社の JAAS Provider が JVM で使用されるためです。

注意： OC4J を使用する場合、JAAS 構成プロパティは、OC4J の起動時にデフォルトで設定されるため、ほとんどの場合、これらのプロパティの設定について憂慮する必要はありません。これらのプロパティを設定するのは、OC4J の外で J2SE アプリケーションを実行する場合のみです。

Oracle 固有の JAAS プロパティを別のファイルに指定し、コマンドラインで JVM に渡すことができます。Oracle では、OracleAS JAAS Provider を指定するデフォルト・ファイル（*ORACLE_HOME*/j2ee/home/config/jazn.security.props）を提供しています。

- すべてのセキュリティ・プロパティを Oracle プロパティで置き換える場合：

  ```
  java -Djava.security.properties==propfile
  ```

- Oracle 固有のプロパティを他のプロパティの後に追加する場合：

  ```
  java -Djava.security.properties=propfile
  ```

OracleAS JAAS Provider 設定の指定

home インスタンスの jazn.xml ファイルは、*ORACLE_HOME*/j2ee/home/config ディレクトリに存在します。OracleAS JAAS Provider は、OC4J の起動前にこのファイルの情報を読み取ります。特定の設定は、このファイルにのみ指定できます。OC4J の起動後に設定の変更が読み取られても、OracleAS JAAS Provider には反映されません。これらのプロパティの詳細は、第 5 章「OC4J インスタンスの構成」を参照してください。

デバッグ・ロギングの有効化

OracleAS JAAS Provider デバッグ・ロギングを有効にするには、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 起動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、Oracle Enterprise Manager の「サーバー・プロパティ」ページの「Java オプション」テキスト・ボックスを使用して 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、*ORACLE_HOME*/opmn/logs ディレクトリにある、各 OC4J インスタンスに関連付けられたログ・ファイルに書き込まれます。

ユーザー・マネージャの指定

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インスタンス内の全アプリケーションに継承されます。インスタンス UserManager は、application.xml ファイルに定義されます。
- 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 orion_application.xml 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 インスタンスおよびオラクル提供の UserManager インスタンス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 インスタンスを使用し、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 インスタンスを使用できますが、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。

ユーザー・マネージャの指定

OC4J インスタンス全体またはそのインスタンス内の特定のアプリケーションに対して UserManager を指定するには、Enterprise Manager を使用します。詳細は、Enterprise Manager のヘルプ・トピック「すべてのアプリケーション用のユーザー・マネージャの変更」を参照してください。

orion-application.xml でのユーザー・マネージャの指定

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。

> **注意 :** JAZNUserManager を使用することをお薦めします。ユーザー定義のユーザー・マネージャは、将来のリリースではサポートされない予定です。

UserManager は、接続がアプリケーションに対して行われる場合のユーザー認証に使用されます。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、表 4-1 に示されている 3 つの要素を使用できます。

表 4-1 UserManager の要素

タグ	意味
<user-manager>	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>	JAZNUserManager
<principals>	principals.xml ファイルに定義されているユーザー・マネージャ 4-14 ページの「<principals> 要素および principals.xml の使用」を参照してください。

高度な構成

1 つの <orion-application> 要素に複数の user-manager の構成を指定できます。どの要素に基づいて UserManager が決定されるかは、表に示した要素の順序によって決まります。つまり、<user-manager> は <jazn> に優先し、<jazn> は <principals> に優先します。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要素に複数の要素が存在する場合、UserManager インスタンスは親として連鎖します。つまり、最初の要素に指定された UserManager が 2 番目の要素に指定された 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 ファイル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 ファイルのデフォルト・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、デプロイ時に jazn-data.xml に基づいて JAAS UserManager が作成されます。必要に応じて、前述の箇条書きと同じ方法で 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 ファイルに <jazn> 要素が書き込まれます。
- アプリケーションのデプロイ時に、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。

RealmLoginModule のカスタマイズ

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 OracleAS Single Sign-On 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。

関連資料：

- OracleAS Single Sign-On 構成タスクの詳細は、Oracle Application Server のインストレーション・ガイドを参照してください。

RealmLoginModule を構成するには、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 の使用方法の詳細は、10-4 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。

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

表 4-2 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サブジェクトに追加されます。	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サブジェクトに追加されます。	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）が認証の成功後にサブジェクトに追加されます。	false
supportCSIv2	true に設定すると、RealmLoginModule で CSIv2 がサポートされます。詳細は、第 15 章「CSIv2 の構成」を参照してください。	false
supportNullPassword	（LDAP ベース・プロバイダのみ）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

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>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>addRoles</name>
               <value>true</value>
             </option>
         </options>
       </login-module>
     </login-modules>
   </application>
  </jazn-loginconfig>
</jazn-data>
```

関連資料：

- OracleAS JAAS Provider の Javadoc の JAAS Provider API Reference
- 10-4 ページの「ログイン・モジュールの追加と削除」

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。構成ファイルは手動で編集する必要があります。

web.xml での auth-method の指定

認証方式を Web モジュール・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。

```
<web-app>
   ...
   <login-config>
      <auth-method>BASIC</auth-method>
   </login-config>
   ...
</web-app>
```

web.xml では、表 4-3 に示す値を <auth-method> 要素に指定できます。これらの値は、orion-application.xml の <jazn-web-app> 要素の属性を使用してアプリケーション・レベルで上書きできます。

表 4-3 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（標準認証）が使用されます。 **注意：** Oracle SSO 機能またはダイジェスト機能を使用する場合も、BASIC 設定を使用します。この場合、web.xml の設定は、<jazn-web-app> の設定で上書きされます。次の「orion-application.xml での auth-method の指定」を参照してください（これは、OC4J 10.1.2 実装でオプションのダイジェスト・メカニズムを実装する XML ベース・プロバイダ専用の方法です）。
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。
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。

orion-application.xml での auth-method の指定

<jazn-web-app> 要素で指定した auth-method 属性の設定は、web.xml 内の auth-method の設定に優先します。OC4J 10.1.2 実装では、<jazn-web-app> での SSO 設定または DIGEST 設定がサポートされます。

auth-method での SSO の指定

orion-application.xml ファイルの <jazn-web-app> 要素で auth-method="SSO" を設定すると、アプリケーションで OracleAS Single Sign-On が使用されます。インストールに LDAP が組み込まれている場合は、Oracle Enterprise Manager により auth-method="SSO" が自動的に設定されます。OracleAS Single Sign-On の使用を止める場合は、ファイルを編集してこの認証方式を削除します。

SSO 認証メカニズムを使用するには、次のように web.xml の auth-method 設定に BASIC を使用します。

```
<web-app>
   ...
   <login-config>
      <auth-method>BASIC</auth-method>
   </login-config>
   ...
</web-app>
```

これにより、この設定は次のような orion-application.xml の設定で上書きされます。

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

auth-method での Digest の指定

Digest 認証メカニズムを使用するには、次のように web.xml の auth-method 設定に BASIC を使用します。

```
<web-app>
   ...
   <login-config>
      <auth-method>BASIC</auth-method>
   </login-config>
   ...
</web-app>
```

これにより、この設定は次のような orion-application.xml ファイルの auth-method="DIGEST" 設定で上書きされます。

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

重要： 次のことに留意してください。

- OC4J 10.1.2 の実装では、Digest 認証メカニズムは JAZN XML ベース・プロバイダでのみサポートされます。
- 古いバージョンの Web ブラウザでは、Digest 認証メカニズムがサポートされないことがあります。
- OC4J の Digest 認証機能は、Oracle HTTP Server の mod_digest 機能とは無関係です。

J2EE 認可の構成

J2EE では、アプリケーションを基礎となるセキュリティ・インフラストラクチャから切り離す、宣言による許可モデルを定義します。このモデルを使用すると、アプリケーションの認可ポリシーは、移植可能な方法でアプリケーションのデプロイメント・ディスクリプタに指定できます。このモデルの成功度は非常に高いことが判明しており、大部分のアプリケーションのニーズに十分応えます。

ただし、高度な使用例によっては、J2EE 認可モデルは静的で精密さに欠けるように見える場合があります。このような場合には、J2EE セキュリティ・モデルのかわりに（または追加して）JAAS 認可モデルを使用できます。J2EE 認可モデルと比較して、JAAS 認可モデルはより強力（ファイングレインかつ動的）で柔軟性（カスタムのパーミッション・タイプをサポート）があります。ただし、このような強力さと柔軟性はコストがかかります。また、JAAS 認可モデルは、理解、デプロイおよび管理する上で J2EE 認可モデルよりも複雑です。

OC4J では、両方のモデルが完全サポートされています。

サーブレット、runas-mode および doasprivileged-mode

ほとんどの場合、アプリケーションには URL 認可で十分です。ただし、アプリケーションで詳細な認可が必要な場合は、この拡張された認可を使用して JAAS ポリシーベースの認可をメソッド・レベルで実現できます。

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、orion-web.xml ファイルまたは orion-application.xml ファイルの <jazn> 要素内に含まれる <jazn-web-app> 要素の runas-mode 属性および doasprivileged-mode 属性を設定する必要があります。そのためには、テキスト・エディタで該当するファイルを開きます。

- subject.doAs() メソッドは、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。OracleAS 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() メソッドは、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、OracleAS 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 属性は、サーブレットの起動に subject.doAsPrivileged() メソッドと subject.doAs() メソッド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 属性は、servlet.runAs() メソッドとは無関係です。

表 4-4 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" />
```

論理ロールからセキュリティ・ロールへのマッピング

場合によっては、サーブレットで想定されているものとは異なるロール名を使用するコンテナにサーブレットをデプロイすることがあります。そのためには、web.xml ファイルで論理ロール（サーブレットで使用されるロール名）をセキュリティ・ロール（コンテナで使用されるロール名）にマップします。

たとえば、次のエンティティは、セキュリティ・ロール corpmanagers を論理ロール mgmt にマップしています。

```
<security-role-ref>
  <role-name>mgmt</role-name>
  <role-link>corpmanagers</role-link>
</security-role-ref>
```

この例では、corpmanagers に属するユーザーとして動作しているサーブレットが isUserInRole("mgmt") を起動すると、メソッドは true を戻します。コンテナでは、セキュリティ・ロールと一致する security-role-ref を検出できないときは常に、Web アプリケーションのセキュリティ・ロールのリスト全体に対して <role-name> をチェックします。

認証プリンシパルからのレルム名の削除

通常、様々なメソッド・コールで戻されるプリンシパルを解析しないことが望ましいです。OracleAS JAAS Provider は、戻されるプリンシパルにレルム名が含まれないように構成できます。そのためには、jaas.username.simple プロパティをインスタンス・レベルの jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）の <jazn> 要素に追加します。このプロパティが true に設定されていると、戻されるプリンシパルにはレルム名が含まれません。デフォルトの false に設定されていると、戻されるプリンシパルには完全なレルム名が含まれます。

jaas.username.simple プロパティは、指定した OC4J インスタンスにのみ影響します。このプロパティを orion-application.xml に設定しても、効果はありません。

このプロパティは、次のメソッドの戻り値に影響します。

- javax.servlet.http.HTTPServletRequest: getRemoteUser() および getUserPrincipal() の各メソッド
- javax.ejb.EJBContext: getCallerIdentity() および getCallerPrincipal() の各メソッド

デフォルトでは、これらのメソッドから戻されるプリンシパルの形式は、*realm_name*/*simple_name*（たとえば jazn.com/john）となります。jaas.username.simple を true に設定すると、戻されるプリンシパルの形式は、*simple_name*（たとえば john）となります。

jaas.username.simple を次のように設定します。

1. インスタンス・レベルの jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）を探します。テキスト・エディタでファイルを開き、ファイル内の <jazn> 要素に移動します。
2. <jazn> 要素内で name="jaas.username.simple" に設定されている <property> サブ要素を探します。
3. サブ要素が存在する場合は、value を true または false に変更します。サブ要素が存在しない場合は作成します。どちらの場合も、サブ要素は次のようになります。

```
<jazn provider="XML" location="./jazn-data.xml">
   <property name="jaas.username.simple" value="true" />
</jazn>
```

> **注意：** <jazn> のプロパティには、この章で指定した以外の編集を加えないでください。

サード・パーティの LDAP プロバイダの構成

第 9 章「外部 LDAP プロバイダの構成」を参照してください。

EJB RMI クライアント・アクセスの許可

RMI を使用した 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。（Admintool の使用方法の一般情報は、4-3 ページの「Admintool の概要」を参照してください。）ユーザー、ロールまたはグループに login の RMIPermission を付与する必要があります。

```
java -jar jazn.jar -grantperm myrealm -role administrators \
  oracle.j2ee.server.rmi.RMIPermission login
```

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 ディレクトリで稼働している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";
```

<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、廃止されていますが、下位互換性を維持するためにのみサポートされています。将来のリリースでは、XMLUserManager および principals.xml はサポートされない予定です。

表 4-5 principals.xml の要素

要素	使用できるサブ要素	属性	説明
<principals>	<groups>、<users>	NA	プリンシパルのトップレベル要素。
<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>			OracleAS JAAS Provider では使用されませんが、一部の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: oracle.j2ee.security.AdministrationPermission() と等価です。 ■ rmi:login: oracle.j2ee.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>	username、password、deactivated	username は、ユーザーの識別に使用される文字列です。 password は、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です。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 属性は、true または false のいずれかです。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。
<group-membership>		group	このユーザーが属しているグループの name 属性。

principals.xml 内のグループは、OracleAS JAAS Provider のロールに対応します。principals.xml ファイルでは、OracleAS 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。

次に principals.xml ファイルの例を示します。

```
<?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>
```

5

OC4J インスタンスの構成

この章では、インスタンス・レベルの OC4J 構成について説明します。この章で説明するタスクはすべて、OC4J インスタンス全体およびそのインスタンス下で稼働しているアプリケーションすべてに影響します。この章には、次の項目が含まれます。

- admin アカウント
- インスタンス・レベルの jazn.xml ファイル
- LDAP 接続プロパティの指定
- LDAP JNDI 接続プール・サイズの指定
- LDAP キャッシングの構成
- LDAP SSL プロパティの構成
- LDAP デフォルト・レルムの構成

admin アカウント

新規 OC4J インスタンスを作成すると、そのインスタンスには常に管理者アカウント admin が welcome というパスワード付きで付与されます。このパスワードは、Oracle Enterprise Manager 10*g* Application Server Control コンソールを使用してただちに変更する必要があります。

次に、パスワードを変更する手順を示します。

1. Application Server のホームページで、OC4J インスタンスを選択します。
2. OC4J インスタンスのホームページで、「管理」を選択します。
3. OC4J の「管理」ページで、「セキュリティ」を選択します。
4. 「セキュリティ」ページで、「ユーザー」の下の jazn.com/admin を選択して管理ユーザー・プロパティを編集します。

インスタンス・レベルの jazn.xml ファイル

この章で説明するタスクはすべて、OC4J インスタンス・レベルの jazn.xml ファイルの編集に基づいています。このファイルは、インスタンスの起動時に読み取られます。インスタンス・レベルの jazn.xml ファイルは、*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml です。このファイルに対する変更はすべて、OC4J インスタンス全体に影響します。この項に示すプロパティは、インスタンス・レベルの jazn.xml ファイルでのみ変更できます。

注意： jazn.xml ファイルは、Application Server Control コンソールを使用して変更することはできません。テキスト・エディタを使用して編集する必要があります。

LDAP 接続プロパティの指定

LDAP 接続プロパティを変更するプロパティは 2 つあります。表 5-1 に、これらのプロパティを示します。

表 5-1 LDAP 接続プロパティ

プロパティ名	意味	デフォルト値
ldap.connect.max.retry	OracleAS JAAS Provider が放棄する前に LDAP 接続の作成を試行する回数	5
ldap.connect.sleep	OracleAS 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	5000

LDAP 接続プロパティを構成するには、次の手順に従います。

1. jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）をテキスト・エディタで開き、<jazn> 要素まで移動します。
2. <jazn> 要素内で <property> サブ要素を探します。<property> サブ要素の構文は次のとおりです。

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

   変更するプロパティに対応する <property> サブ要素がない場合は、作成します。
3. OC4J を再起動します。

LDAP JNDI 接続プール・サイズの指定

LDAP 接続プール・プロパティを変更するプロパティは 2 つあります。表 5-2 に、これらのプロパティを示します。

表 5-2 LDAP JNDI 接続プール・プロパティ

プロパティ名	意味	デフォルト値
jndi.ctx_pool.init_size	JNDI/LDAP 接続プールの初期サイズ	5
jndi.ctx_pool.inc_size	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	10

JNDI で使用される接続プールのサイズを指定するには、次のようにします。

1. jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）をテキスト・エディタで開き、<jazn> 要素まで移動します。
2. <jazn> 要素内で <property> サブ要素を探します。<property> サブ要素の構文は次のとおりです。

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

   変更するプロパティに対応する <property> サブ要素がない場合は、作成します。たとえば、初期サイズを 20 に設定する <property> サブ要素は次のようになります。

   ```
   <property name="jndi.ctx_pool.init_size" value="20">
   ```

   注意 : <jazn> のプロパティには、このマニュアルで指定した以外の編集を加えないでください。

3. OC4J を再起動します。

LDAP キャッシングの構成

LDAP ベースの OracleAS JAAS Provider では、パフォーマンスとスケーラビリティを改善するキャッシングがサポートされています。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（Cookie が有効化されている Web ベース・クライアントでのみ使用可能）。

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 が保持されます。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無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ますが、有効期限は表 5-3 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・キャッシュの詳細の変更

HttpSession オブジェクトは、サーバー・サイド・セッションが継続する間、存続します。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。

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。

関連資料：

- 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。

LDAP 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。管理タスクをプログラムで実行する場合は、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。

注意： JAZN Admintool は、動作時にキャッシングを自動的に無効化し、終了時にキャッシングを再度有効化します。

LDAP キャッシュを無効化するには、次の手順に従います。

1. jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）をテキスト・エディタで開き、<jazn> 要素まで移動します。
2. <jazn> 要素を次のように編集します。

```
<jazn provider="LDAP">
   <property name="ldap.user"
             value="orclApplicationCommonName=jaznadmin1,cn=JAZNContext,
                    cn=products,cn=OracleContext"/>
   <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>
```

3. OC4J を再起動します。

LDAP キャッシュ構成

LDAP キャッシュに影響するプロパティは、<jazn> 要素内の <property> サブ要素により制御されます。これらのプロパティを変更するには、jazn.xml ファイルを編集し、<jazn> 要素を変更します。

LDAP キャッシュ・プロパティを構成するには、次の手順に従います。

1. jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）をテキスト・エディタで開き、<jazn> 要素まで移動します。
2. <jazn> 要素内で <property> サブ要素を探します。<property> サブ要素の構文は次のとおりです。

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

   変更するプロパティに対応する <property> サブ要素がない場合は、作成します。
3. OC4J を再起動します。

表 5-3 に、LDAP キャッシュ・プロパティとそのデフォルト値を示します。これらのプロパティは、jazn.xml の <jazn> 要素で、インスタンス・レベルでのみ設定できます。

表 5-3 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。	0.7
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	オブジェクトが無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：

- <jazn> のプロパティには、このマニュアルで指定した以外の編集を加えないでください。
- ldap.cache.policy.enable プロパティは、廃止された ldap.cache.enable プロパティにかわるものです。

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 要素は、次のようになります。

```
< 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>
```

LDAP SSL プロパティの構成

SSL に影響するプロパティは、<jazn> 要素内の <property> サブ要素により制御されます。これらのプロパティを変更するには、<jazn> 要素を含むファイルを編集する必要があります。

LDAP SSL プロパティを構成するには、次の手順に従います。

1. jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）をテキスト・エディタで開き、<jazn> 要素まで移動します。
2. <jazn> 要素内で <property> サブ要素を探します。<property> サブ要素の構文は次のとおりです。

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

   変更するプロパティに対応する <property> サブ要素がない場合は、作成します。
3. OC4J を再起動します。

表 5-4 に、SSL プロパティを示します。

表 5-4 <jazn> 要素の <property> サブ要素の値

プロパティ名	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、14-2 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.protocol	SSL を使用して LDAP と通信する際に使用されるプロトコル。
ldap.user	LDAP ユーザー名または DN。この要素は、自動的に移入されます。内容を変更しないでください。次に例を示します。 orclApplicationCommonName=jaznadmin1,cn=JAZNContext, cn=products,cn=OracleContext

注意： <jazn> のプロパティには、このマニュアルで指定した以外の編集を加えないでください。

SSL 認証の選択

この項では、Oracle Internet Directory で SSL を使用するように OracleAS 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 Oracle Internet Directory 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。

10*g* リリース 2（10.1.2）では、Oracle Internet Directory と通信する際に NULL 認証を使用する必要があります。NULL 認証とは、データは Anonymous Diffie-Hellman 暗号スイートを使用して暗号化されますが、認証に証明書が使用されないことを意味します。

インストール時に SSL を選択すると、SSL は NULL 認証により有効化されます。手動で SSL を有効化する必要があるのは、インストールの一環として SSL を選択しなかった場合のみです。その場合、NULL 認証のために、jazn.xml の <jazn> 要素に <property> 要素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）。

```
<jazn provider="LDAP" location="ldap://example.com:5000" default-realm="us">
   ...
   <property name="ldap.protocol" value="ssl"/>
   ...
</jazn>
```

LDAP デフォルト・レルムの構成

デフォルト・レルムとは、認証または認可の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のことです。この属性は、デフォルトの Oracle Identity Management レルムにより自動的に移入されます。この属性を編集する必要があるのは、デフォルトがアプリケーションに不適切な場合のみです。LDAP デフォルト・レルムを構成するには、次の手順に従います。

1. jazn.xml ファイル（*ORACLE_HOME*/j2ee/*instance_name*/config/jazn.xml）をテキスト・エディタで開き、<jazn> 要素まで移動します。
2. <jazn> 要素の default-realm 属性を編集します。次に構文を示します。

   ```
   <jazn provider="LDAP" default-realm="myrealm">
      ...
   </jazn>
   ```

3. OC4J を再起動します。

> **注意：** <jazn> のプロパティには、このマニュアルで指定した以外の編集を加えないでください。

たとえば、default-realm を Sales に設定する <jazn> 要素は次のようになります。

```
<jazn provider="LDAP" default-realm="Sales" ... more attributes ... >
   ...
</jazn>
```

6

アプリケーションのデプロイ時における セキュリティ上の考慮事項

この章では、アプリケーションのデプロイ時に考慮すべき問題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。

ユーザー・マネージャの選択

デフォルトでは、OC4J インスタンスをインフラストラクチャに関連付けた場合、JAZN LDAP の `UserManager` が新しくデプロイされたアプリケーションに使用されます。関連付けなかった場合は、JAZN XML の `UserManager` がアプリケーションに使用されます。なんらかの理由でアプリケーションのユーザー・マネージャを変更する必要がある場合、Application Server Control コンソールから変更できます。詳細は、Application Server Control コンソールのヘルプ・トピック「すべてのアプリケーション用のユーザー・マネージャの変更」を参照してください。

セキュリティ・ロールのマッピング

アプリケーションのセキュリティ・ロールをマップするには、Application Server Control コンソールの「セキュリティ」ページを使用します。次の手順に従います。

1. Application Server Control コンソールからアプリケーションを選択し、「セキュリティ」リンクをクリックします。
2. 「セキュリティ・ロール」リストからロールを選択します。
3. 「**ロールをプリンシパルにマップ**」ボタンをクリックします。「ロール : *yourrole*」というタイトルの新しいページが表示されます。
4. 必要なグループまたはユーザーの横にあるチェック・ボックスを選択します。(「ロールをグループにマップ」および「ロールをユーザーにマップ」というラベルの 2 つの領域があります。)「**適用**」をクリックします。
5. 確認ページが表示されます。「**OK**」をクリックします。

パーミッションの付与

パーミッションを付与するには、次の 2 つの方法があります。

- RMI パーミッションまたは `administration` パーミッションを付与するには、Oracle Enterprise Manager 10*g* Application Server Control コンソールを使用します。詳細は、次の「RMI パーミッションまたは administration パーミッションの付与」を参照してください。
- RMI パーミッションまたは `administration` パーミッション以外のパーミッションを付与するには、JAZN Admintool を使用します。詳細は、「その他すべてのパーミッションの付与と取消し」を参照してください。

RMI パーミッションまたは administration パーミッションの付与

RMI または `administration` パーミッションは、Oracle Enterprise Manager 10*g* Application Server Control コンソールを使用してグループに付与できます。次に手順を示します。

1. アプリケーションを選択し、「セキュリティ」ページにナビゲートします。
2. グループのリストからグループ名を選択します。「グループの追加」または「グループの編集」ページが表示されます。
3. 追加するパーミッションをすべてチェックし、「**適用**」をクリックします。

その他すべてのパーミッションの付与と取消し

ユーザー・パーミッションの付与と取消しには、JAZN Admintool を使用します。JAZN Admintool の実行の基本情報は、4-3 ページの「Admintool の概要」を参照してください。

```
-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]
```

この構文の *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です。また、*principal_parameters* は、単一の String 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 の場合、realmname action）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。

注意： Admintool から「パーミッション・クラスが見つかりません。」というエラー・メッセージが発行された場合、付与するパーミッションがクラスパス内にないことを意味します。権限クラスを含む JAR を jdk/jre/lib/ext ディレクトリに置き、Admintool で検出できるようにする必要があります。

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 -user martha java.io.FilePermission
     a.txt read,write
```

Admintool のシェルの場合

```
JAZN:> grantperm foo -user martha java.io.FilePermission a.txt read,write
```

ユーザーおよびグループの作成

各プロバイダにおけるユーザーおよびグループの作成の詳細は、第 7 章「LDAP ベース・プロバイダの構成」または第 8 章「XML ベース・プロバイダの構成」を参照してください。

7

LDAP ベース・プロバイダの構成

この章では、OC4J を構成して Oracle Internet Directory（OID）の LDAP ベース・プロバイダを使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。

- LDAP 使用の準備
- LDAP ユーザーおよびグループの作成

LDAP プロパティには、OC4J インスタンス全体に影響するものがあります。これらのプロパティは、4-5 ページの「OracleAS JAAS Provider 設定の指定」を参照してください。

LDAP 使用の準備

通常、OC4J は、インストール時にインフラストラクチャに関連付けますが、Oracle Enterprise Manager 10*g* Application Server Control コンソールを使用してインフラストラクチャに関連付けることもできます。Oracle Enterprise Manager 10*g* のヘルプ・トピック「Application Server Infrastructure ページ」を参照してください。

OC4J インスタンスを Oracle Application Server Infrastructure（Oracle Internet Directory 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ます。

管理ユーザーおよびグループの作成

LDAP ベース・プロバイダを使用するには、特定のユーザー、グループおよびパーミッションを Oracle Delegated Administration Service で設定し、これらのユーザーおよびグループに適切なパーミッションを付与する必要があります。

LDAP ベース・プロバイダを *ORACLE_HOME*/j2ee/*instance_name*/config/application.xml 構成ファイルでグローバルに指定する場合、7-3 ページの「ldapmodify を使用した匿名ユーザーの作成」の説明に従って匿名ユーザーを作成する必要もあります。一般的な環境では、application.xml を変更する必要はありません。この作業を実行する主な理由は、OC4J インスタンスでデフォルト・アプリケーションを構成し、ユーザー・マネージャとして LDAP ベース・プロバイダを使用するためです。デフォルト・アプリケーションは、内部で使用するために OC4J が作成するシステム・アプリケーションです。（OC4J デフォルト・アプリケーションの詳細は、『Oracle Application Server Containers for J2EE サーブレット開発者ガイド』のデプロイおよび構成の概要を参照してください。）

LoadOidData を使用したユーザーの作成

適切なグループおよびユーザーは、oracle.security.jazn.util.LoadOidData ツールを使用して設定できます。このツールは、*ORACLE_HOME* ディレクトリの jazncore ライブラリにあります。次のようにコマンドラインでツールを実行します。

```
java -cp ./jazncore.jar oracle.security.jazn.util.LoadOidData
```

このツールの構文は次のとおりです。

```
LoadOidData [-h ldaphost] [-p ldapport] [-D binddn] [-w passwd]
            [-f filename [-oc4jAdminPwd passwd] [-ignoreError true|false]
```

サポートされるオプションは次のとおりです。

- -h *ldaphost*: LDAP ホスト名。
- -p *ldapport*: LDAP サーバーのポート。
- -D *binddn*: Oracle Internet Directory 管理者の識別名。
- -w *password*: Oracle Internet Directory 管理者のパスワード。
- -f *filename*: ロード対象のエントリが含まれる次のファイル。

  ORACLE_HOME/j2ee/*instance_name*/jazn/install/oidConfigForOc4j.sbs

- -oc4jAdminPwd *password*: OC4J 管理者に割り当てられるパスワード。
- -ignoreError *boolean*: エラーの発生時に、ツール実行を継続するか（true の場合）、即座に停止するか（false）を指定します。

たとえば、Oracle データベース管理者のパスワードが welcome1 で、OC4J 管理ユーザーのパスワードが welcome2 だと仮定します。$J2EE_HOME が *ORACLE_HOME*/j2ee/home である場合、コマンドラインには次のように入力します。

```
java -cp $J2EE_HOME/jazncore.jar oracle.security.jazn.util.LoadOidData
     -h oidhost -p oidport -D cn=orcladmin -w welcome1
     -f $J2EE_HOME/jazn/install/oidConfigForOc4j.sbs -oc4jAdminPwd welcome2
```

このツールの実行後、デフォルトの Oracle Identity Management レルムには、次のものが含まれます。

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

administrators グループには、次のパーミッションがあります。

- oracle.j2ee.server.AdministrationPermission ("administration")
- oracle.j2ee.server.rmi.RMIPermission("login")

最後に、ldap.user プロパティに admin を、ldap.password プロパティに適切なパスワードを設定します。この手順は、5-6 ページの「LDAP SSL プロパティの構成」を参照してください。

ldapmodify を使用した匿名ユーザーの作成

匿名ユーザーを作成するには、LDIF（Lightweight Directory Interchange Format）ファイルを作成し、その LDIF ファイルを ldapmodify ツールへの入力用として指定します。例 7-1 に、適切な LDIF ファイルを示します。ただし、*yourDistinguishedName* は、デフォルトの ID 管理レルムの識別名で置き換える必要があります。

例 7-1 匿名ユーザー作成用の anony.ldif ファイル

```
dn: cn=anonymous, cn=Users, yourDistinguishedName
changetype: add
uid: anonymous
givenName: anonymous
cn: anonymous
sn: anonymous
description: This entry is used as the identification for unauthenticated users.
orclisenabled: disabled
objectClass: top
objectclass: person
objectclass: organizationalPerson
objectClass: inetorgperson
objectClass: orcluser
objectClass: orcluserV2
```

anony.ldif ファイルの作成後、ldapmodify コマンドを使用して匿名ユーザーを追加します。このコマンドの構文は、次のとおりです。

```
ORACLE_HOME/bin/ldapmodify -D cn=orcladmin -w password -h hostname -p port \
                           -f anony.ldif
```

このコマンドを発行する場合、*password*、*hostname* および *port* は、実際のインストール環境のパスワード、ホスト名およびポートで置き換えます。

> **注意：** anonymous アカウントは、OC4J サーバー専用として Oracle Internet Directory サーバーで作成される特別なユーザー・アカウントです。このアカウントはパスワードなしで作成されるため、エンド・ユーザーがアプリケーションにログインするためには使用できません。

LDAP ベース・プロバイダの環境変数

開発を始める前に、動的ライブラリ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（Solaris の LD_LIBRARY_PATH など）が適切に設定されていることを確認してください。詳細は、2-9 ページの表 2-5「動的ライブラリ・パスの設定」を参照してください。

Oracle Enterprise Manager を使用して OC4J を管理する場合は、この変数は自動的に設定されます。

LDAP ユーザーおよびグループの作成

LDAP ベース・プロバイダの使用時にユーザーおよびグループを作成するには、Oracle Delegated Administration Service ツールを使用します。詳細は、『Oracle Identity Management 委任管理ガイド』を参照してください。

8

XML ベース・プロバイダの構成

この章では、Oracle Enterprise Manager 10*g* Application Server Control コンソールおよび JAZN Admintool を使用した、ユーザー、グループおよびロールの基本管理タスクの実行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。

注意： この章では「ロール」という用語を使用しますが、これはこの用語が JAZN Admintool で使用されるためです。「ロール」は、より一般的に使用される「グループ」という用語と同義です。

ユーザーの作成

XML ベース・プロバイダでユーザーを作成するには、次のように Enterprise Manager を使用します。

1. Application Server Control コンソールに移動します。
2. 該当する OC4J インスタンスの「セキュリティ」ページにナビゲートします。
3. 「**ユーザーの追加**」ボタンをクリックし、ページの指示に従います。

> **注意：** `a/b/c` のように、スラッシュ文字（/）を含むユーザー名は作成しないでください。

ロール（グループ）の作成

XML ベース・プロバイダでロール（グループ）を作成するには、次のように Enterprise Manager を使用します。

1. Application Server Control コンソールに移動します。
2. 該当する OC4J インスタンスの「セキュリティ」ページにナビゲートします。
3. 「**グループの追加**」ボタンをクリックし、ページの指示に従います。

ユーザーの削除

XML ベース・プロバイダでユーザーを削除するには、次のように Enterprise Manager を使用します。

1. Application Server Control コンソールに移動します。
2. 該当する OC4J インスタンスの「セキュリティ」ページにナビゲートします。
3. ラジオ・ボタンを使用してユーザーを選択します。
4. 「**削除**」ボタンをクリックし、ページの指示に従います。

> **注意：** インスタンス・レベルの `jazn-data.xml` ファイルには、`admin` および `anonymous` のアカウントが含まれている必要があります。これらのアカウントを削除しないでください。削除すると、OracleAS JAAS Provider は動作を停止します。

ロール（グループ）の削除

XML ベース・プロバイダでロール（グループ）を削除するには、次のように Enterprise Manager を使用します。

1. Application Server Control コンソールに移動します。
2. 該当する OC4J インスタンスの「セキュリティ」ページにナビゲートします。
3. ラジオ・ボタンを使用してグループを選択します。
4. 「**削除**」ボタンをクリックし、ページの指示に従います。

レルムの作成

レルムを追加するには、JAZN Admintool を使用します。Admintool の使用方法の詳細は、4-3 ページの「Admintool の概要」を参照してください。

Admintool の `-addrealm` オプションを使用してレルムを追加します。このオプションは、レルム名、管理者の名前およびパスワードを引数として使用します。次に構文を示します。

```
-addrealm realm admin adminpwd adminrole
```

たとえば、XML ベース・プロバイダを使用している場合、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
```

レルムの削除

レルムを削除するには、JAZN Admintool を使用します。Admintool の使用方法の詳細は、4-3 ページの「Admintool の概要」を参照してください。

Admintool の `-remrrealm` オプションを使用してレルムからロールを削除します。このオプションの引数は `realm` のみで、レルム名を指定します。次に構文を示します。

```
-remrealm realm
```

レルム `foo` を削除するには、次のように入力します。

```
java -jar jazn.jar -remrealm foo
```

パーミッションの付与

6-2 ページの「パーミッションの付与」を参照してください。

パーミッションの取消し

パーミッションの取消しには、JAZN Admintool を使用します。Admintool の使用方法の詳細は、4-3 ページの「Admintool の概要」を参照してください。`-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* 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*"）。次に構文を示します。

```
-revokeperm {realm {-user user|-role role} | principal_class principal_parameters}
            permission_class [permission_parameters]
```

この構文の *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です。また、*principal_parameters* は、単一の `String` パラメータです。

`perm1` パーミッションを取り消すには、次のように入力します。

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

ロール（グループ）の付与

XML ベース・プロバイダでロールを付与するには、次のように Enterprise Manager を使用します。

1. Application Server Control コンソールに移動します。
2. 選択した OC4J インスタンスの「セキュリティ」ページにナビゲートします。
3. ラジオ・ボタンを使用してユーザーを選択します。
4. 付与するロールに対応するチェック・ボックスを選択します。
5. 「**適用**」ボタンをクリックします。

ロール（グループ）の取消し

XML ベース・プロバイダでロールを取り消すには、次のように Enterprise Manager を使用します。

1. Application Server Control コンソールに移動します。
2. 選択した OC4J インスタンスの「セキュリティ」ページにナビゲートします。
3. ラジオ・ボタンを使用してユーザーを選択します。
4. 取り消すロールに対応するチェック・ボックスを選択します。
5. 「**適用**」ボタンをクリックします。

永続性モードの設定

永続性モードは、データへの変更が `jazn-data.xml` に書き込まれるタイミングを制御します。永続性に対して使用可能な値は次の 3 つです。

- `NONE`

  `jazn-data.xml` には変更は書き込まれません。

- `ALL`

  変更後にその内容が書き込まれます。

- `VM_EXIT`（デフォルト）

  Java Virtual Machine が終了する際に変更が書き込まれます。

XML ベース・プロバイダで永続性モードを構成するには、`jazn.xml` ファイルの `<jazn>` 要素を手動で編集する必要があります。

関連項目：

- `jazn.xml` の検索の詳細は、4-2 ページの「jazn.xml、jazn-data.xml および <jazn> 要素の検索」を参照してください。

1. `jazn.xml` をテキスト・エディタで開き、`<jazn>` 要素まで移動します。
2. `<jazn>` 要素の `persistence` 属性を編集します。たとえば、変更後にその内容を書き込む場合は、`<jazn>` 要素を次のように編集します。

```
<jazn persistence="ALL" ... other attributes >
   ...
</jazn>
```

注意： `<jazn>` 要素の他の属性は変更しないでください。

XML デフォルト・レルムの構成

デフォルト・レルムとは、認証または認可の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のことです。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。XML デフォルト・レルムを構成するには、次の手順に従います。

1. <jazn> 要素を含むファイルを探します（4-2 ページの「jazn.xml、jazn-data.xml および <jazn> 要素の検索」を参照してください）。そのファイルをテキスト・エディタで開き、<jazn> 要素まで移動します。
2. <jazn> 要素の default-realm 属性を編集します。次に構文を示します。

   ```
   <jazn provider="XML" default-realm="myrealm" ... >
   ```

3. たとえば、default-realm を Sales に設定する <jazn> 要素は次のようになります。

   ```
   <jazn provider="XML" default-realm="Sales" ... more attributes ... >
     ...
   </jazn>
   ```

> **注意：** <jazn> のプロパティには、この章で指定した以外の編集を加えないでください。

principals.xml ファイルからのプリンシパルの移植

principals.xml ファイルからデータを移植するには、JAZN Admintool を使用します。JAZN Admintool の実行の基本情報は、4-3 ページの「Admintool の概要」を参照してください。

```
-convert filename realm
```

-convert オプションを使用して、principals.xml ファイルを現行の OracleAS 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。

移植により、principals.xml のユーザーが JAAS のユーザーに、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。
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
   ```

   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。
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. `<default-realm>` を `principals.com` に変更します。

   関連項目：

   - 8-4 ページの「永続性モードの設定」

5. OC4J を停止して再起動します。

9

外部 LDAP プロバイダの構成

この章では、Oracle 以外の LDAP サーバーを使用するための OC4J の構成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。

注意：

- Oracle 以外の LDAP サーバーを利用するには、JDK1.4 以上を使用する必要があります。
- OC4J には、Oracle 以外の LDAP サーバーに対して認証と認可を行うための JAAS ログイン・モジュールがあります。Oracle 以外の LDAP サーバー用の JAAS ログイン・モジュールを構成して、Oracle Internet Directory（OID）に対する認証および認可を行わないでください。これを実行すると、ネイティブ LDAP プロバイダの使用時に提供される最適化と統合の機能が失われます。ネイティブ LDAP プロバイダを使用して Oracle Internet Directory を構成する方法の詳細は、第 7 章「LDAP ベース・プロバイダの構成」を参照してください。

前提条件

OC4J を構成する前に、次の前提条件をすべて満たす必要があります。

1. Sun Java System Application Server（旧 iPlanet）または Active Directory をインストールし、構成します。
2. OC4J をインストールし、構成します。
3. OC4J インスタンスに関連付けられた jazn-data.xml ファイルを探します。このファイルは、通常、*ORACLE_HOME*/j2ee/*instance_name*/config ディレクトリにあります。このファイルはテキスト・エディタを使用して編集します。

   注意： OC4J home インスタンスの jazn-data.xml ファイル（*ORACLE_HOME*/j2ee/home/config 内）は、JAAS ログイン・モジュールのデフォルト・リポジトリとして機能します。

4. アプリケーションを制御する orion-application.xml ファイルを探します。このファイルは、通常、次のディレクトリにあります。

   ORACLE_HOME/j2ee/*instance_name*/application-deployments/*application_name*

   このファイルはテキスト・エディタを使用して編集します。

   注意：

   - Sun Java System Application Provider および Microsoft Active Directory のサンプル・ログイン・モジュール・エントリは、*ORACLE_HOME*/j2ee/home/jazn/config ディレクトリにあります。非プロバイダ固有のログイン・モジュール・エントリは、同じディレクトリのファイル ldap_login_module.template に提供されています。
   - Oracle 以外の LDAP サーバーを使用している場合、jazn-data.xml でプリンシパルに権限付与を行うときは、次のことに注意してください。完全な識別名（DN）を使用してプリンシパルの名前を指定する場合、その識別名は、LDAP サーバーの設定とまったく同じ形式で、空白なしで指定する必要があります。次に例を示します。

     ```
     cn=jdoe,dc=us,dc=example,dc=com
     ```

jazn-data.xml での <login-module> 要素の作成

<login-module> の各オプションは、LDAP プロバイダの構成設定と対応しています。表 9-1、表 9-2 および表 9-3 に、サポート対象のオプションを示します。（オプション）と記載されていなければ、オプションはすべて明示的に指定する必要があります。

表 9-1 ログイン・モジュール・プロバイダのオプション

オプション名	意味
oracle.security.jaas.ldap.provider.url	*hostname*:*portname* 形式の LDAP プロバイダの URL。
oracle.security.jaas.ldap.provider.principal	LDAP サーバーへの接続に使用される LDAP ユーザーの識別名（DN）。このユーザーは、ユーザーおよびグループを検索するための権限と、ターゲット・ディレクトリでサポートされている場合にはユーザー・パスワードに対して ldapcompare を起動するための権限を持つ管理者であることが必要です。
oracle.security.jaas.ldap.provider.credential	principal に定義されている LDAP ユーザーの認証に使用される資格証明（一般にパスワード）。
oracle.security.jaas.ldap.provider.type	（オプション）LDAP プロバイダの製品名。サポートされる値は、iplanet、active directory および other です。iplanet または active directory を指定すると、ログイン・モジュールは一部の LDAP プロパティを推測し（Active Directory のグループ・オブジェクト・クラスは group など）、なんらかの最適化を実行できます。
oracle.security.jaas.ldap.provider.connect.pool	（オプション）接続プーリングが使用できるかどうかを決定するブール値。true は接続プーリングを有効化し、false は無効化します。
oracle.security.jaas.ldap.lm.cache_enabled	（オプション）ログイン・モジュール・キャッシングが使用できるかどうかを決定するブール値。true（デフォルト）はキャッシングを有効化し、false は無効化します。

表 9-2 ログイン・モジュール・ユーザーのオプション

オプション名	意味
oracle.security.jaas.ldap.user.name.attribute	ユーザー名を一意に識別する LDAP 属性の名前。Sun Java System Application Server では uid、Active Directory では sAMAccountName です。
oracle.security.jaas.ldap.user.objectclass	ユーザーを表すのに使用される LDAP スキーマ・オブジェクト・クラスの空白で区切られたリスト。Sun Java System Application Server では inetOrgPerson です。
oracle.security.jaas.ldap.user.searchbase	ユーザーを含む LDAP ディレクトリ内のベース識別名（DN）の空白で区切られたリスト。次に例を示します。 cn=users,dc=us,dc=abc,dc=com
oracle.security.jaas.ldap.user.searchscope	ユーザーを検索する LDAP ディレクトリ・ツリーの深さを指定します。サポートされる値は subtree および onelevel です。

表 9-3 ログイン・モジュール・ロールのオプション

オプション名	意味
oracle.security.jaas.ldap.role.name.attribute	ロール名を一意に識別する LDAP 属性の名前。iPlanet では uniqueMember、Active Directory では member です。
oracle.security.jaas.ldap.role.object.class	グループを表すのに使用される LDAP スキーマ・オブジェクト・クラスの空白で区切られたリスト。Sun Java System Application Server では groupOfUniqueNames です。Active Directory では group です。
oracle.security.jaas.ldap.role.searchbase	グループを含む LDAP ディレクトリ内の識別名（DN）の空白で区切られたリスト。次に例を示します。 cn=groups,dc=us,dc=abc,dc=com
oracle.security.jaas.ldap.role.searchscope	ロールを検索する LDAP ディレクトリ・ツリーの深さを指定します。サポートされる値は subtree および onelevel です。
oracle.security.jaas.ldap.role.membership.searchscope	ロール・メンバーシップを検索する LDAP ディレクトリ・ツリーの深さを指定します。サポートされる値は direct および nested です。
oracle.security.jaas.ldap.role.member.attribute	グループのメンバーの識別名（DN）を指定する静的 LDAP グループ・オブジェクトの属性。Sun Java System Application Server では uniqueMember、Active Directory では member です。

サンプル LDIF の説明

例 9-1 に、ユーザー・オブジェクトとロール・オブジェクトに関する宣言のサンプルを示します。次の 2 つの各項では、これらのオブジェクトを LDAP プロバイダにマップする方法について説明します。

例 9-1 ユーザーとロールを定義するサンプル LDIF

```
# An example user object entry
uid= jdoe,dc=us,dc=example,dc=com
uid= jdoe
givenName=John
sn=Doe
cn=John Doe
userPassword={SSHA}zD/44JbZY33osry4mzfLn0du7nBhIIAHKDG5Fg==
uidNumber=1
gidNumber=1
homeDirectory=c:¥
objectClass=top
objectClass=person
objectClass=organizationalPerson
objectClass= inetOrgPerson
objectClass=posixAccount

# An example role object entry
cn=managers,ou=groups,dc=us,dc=example,dc=com
objectClass=top
objectClass= groupOfUniqueNames
cn=managers
uniqueMember=uid=jdoe,dc=us,dc=example,dc=com
```

LDAP プロバイダとしての Sun Java System Application Server の構成

このリリースでは、jazn-data.xml ファイルを編集し、Sun 社の製品に対応する <login-module> 要素を追加して、Sun Java System Application Server を LDAP プロバイダとして構成する必要があります。この項では、必要な変更について説明します。

注意： Sun Java System Application Server のサンプル・ログイン・モジュール・エントリが含まれるテンプレート・ファイルは、*ORACLE_HOME*/j2ee/home/jazn/config ディレクトリのファイル sample_login_module.sun に提供されています。

---

1. テキスト・エディタを使用して jazn-data.xml ファイルを開きます（9-2 ページの「前提条件」を参照してください）。
2. アプリケーションを表す <application> 要素を探します。<application> 要素がない場合は、作成します。
3. <application> 要素内で <login-modules> セクションを探します。<login-modules> 要素がない場合は、作成します。
4. テキスト・エディタを使用して orion-application.xml ファイルを開きます（9-2 ページの「前提条件」を参照してください）。
5. orion-application.xml 内で <jazn> 要素を探します。provider プロパティを "XML" に設定し、custom.ldap.provider を true に設定する <property> 要素を追加します。編集後の <jazn> 要素は次のようになります。

```
<jazn provider="XML">
    <property name="custom.ldap.provider" value="true"/>
</jazn>
```

6. Enterprise Manager を使用して OC4J インスタンスを再起動します。

SunOne の例

Sun Java System Application Server のインストールが、例 9-1 に示されている一連の LDIF エントリによって記述されているとします。

対応する <jazn-loginconfig> エンティティを例 9-2 に示します。

例 9-2 例 9-1 に対応する JAAS ログイン・モジュール構成

```
<jazn-loginconfig>
   <application>
      <name>callerInfo</name>
      <login-modules>
         <login-module>
            <class>oracle.security.jazn.login.module.LDAPLoginModule</class>
            <control-flag>required</control-flag>
            <options>
               ... irrelevant options omitted ...
               <option>
                  <name>oracle.security.jaas.ldap.user.name.attribute</name>
                  <value>uid</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.user.object.class</name>
                  <value>inetOrgPerson</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.user.searchbase</name>
```

```
            <value>dc=us,dc=example,dc=com</value>
          </option>
          <option>
            <name>oracle.security.jaas.ldap.role.name.attribute</name>
            <value>cn</value>
          </option>
          <option>
            <name>oracle.security.jaas.ldap.role.object.class</name>
            <value>groupOfUniqueNames</value>
          </option>
          <option>
            <name>oracle.security.jaas.ldap.role.searchbase</name>
            <value>ou=groups,dc=us,dc=example,dc=com</value>
          </option>
          <option>
            <name>oracle.security.jaas.ldap.member.attribute</name>
            <value> uniqueMember </value>
          </option>
        </options>
      </login-module>
    </login-modules>
  </application>
</jazn-loginconfig>
```

LDAP プロバイダとしての Microsoft Active Directory の構成

このリリースでは、jazn-data.xml ファイルを編集し、Microsoft 社の製品に対応する <login-module> 要素を追加して、Microsoft Active Directory を LDAP プロバイダとして構成する必要があります。この項では、必要な変更について説明します。

注意： Active Directory のサンプル・ログイン・モジュール・エントリが含まれるテンプレート・ファイルは、*ORACLE_HOME*/j2ee/home/jazn/config ディレクトリのファイル sample_login_module.ad に提供されています。

---

1. アプリケーションを表す <application> 要素を探します。<application> 要素がない場合は、作成します。
2. <application> 要素内で <login-modules> セクションを探します。<login-modules> 要素がない場合は、作成します。
3. <option> 要素を編集して、Microsoft Active Directory に適切な値を指定します。編集したファイルを保存します。
4. テキスト・エディタを使用して orion-application.xml ファイルを開きます（9-2 ページの「前提条件」を参照してください）。
5. orion-application.xml 内で <jazn> 要素を探します。provider プロパティを "XML" に設定し、custom.ldap.provider を true に設定する <property> 要素を追加します。編集後の <jazn> 要素は次のようになります。

```
<jazn provider="XML">
    <property name="custom.ldap.provider" value="true"/>
</jazn>
```

6. Enterprise Manager を使用して OC4J インスタンスを再起動します。

10

カスタム・ログイン・モジュール

OC4J には、JAAS 規格に準拠する 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により、アプリケーション・サーバーと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・サーバーまたはアプリケーション・コードに変更を加えずに、JAAS ログイン・モジュールを介して代替認証サービスをプラグインできます。

この章では、OracleAS JAAS Provider で使用する `LoginModule` を作成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。

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した場合、そのアプリケーションでの `UserManager` API の使用はサポートされなくなります。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 ログイン・モジュール構成は、常に OC4J `home` インスタンスの `jazn-data.xml` ファイルに格納されるため、デプロイ時にアプリケーションのセキュリティ・プロバイダとして XML ベース・プロバイダを選択する必要があります。LDAP ベース・プロバイダとカスタム・ログイン・モジュールの併用は、サポートされません。
- ほとんどの場合、カスタム・ログイン・モジュールを使用するようにアプリケーションを構成するには、JAZN プロパティの `role.mapping.dynamic` を `true` に設定する必要があります。デフォルトでは、このプロパティは `false` です。`<jazn>` 要素のこのプロパティの詳細は、10-8 ページの「orion-application.xml ファイル」も参照してください。

カスタム JAAS ログイン・モジュールの統合

カスタム JAAS LoginModule は、Oracle Identity Management を使用できず、ユーザーとロール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。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。カスタム LoginModule の作成時に、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、パッケージ化およびデプロイ :** J2SE1.4 付属のログイン・モジュールを使用するか。またはカスタムやサード・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。

ログイン・モジュールの開発

JAAS 準拠の LoginModule は OC4J フレームワーク内で使用できます。

関連資料 :

- ログイン・モジュール開発の一般情報は、次の URL にある Sun 社の JAAS ドキュメントを参照してください。

  http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASLMDevGuide.html

LoginModule の開発時に、次のような重要な問題を考慮する必要があります。

- サブジェクト・ベースの認可
- J2EE のセキュリティ認可
- コールバックのサポート
- デバッグのヒント

次の各項では、これらの問題について詳しく説明します。

サブジェクト・ベースの認可

カスタム LoginModule をアプリケーションと関連付ける場合、サブジェクトおよびそれに含まれるプリンシパルは、J2EE セキュリティ制約の評価などの認可タスクすべてに対する唯一の基礎として使用される必要があります。認可ですべての関連プリンシパルを必ず考慮に入れるために、LoginModule では、JAAS 認証プロセスのコミット・フェーズで、関連プリンシパル（認証済ユーザーが属するすべてのロールおよびグループなど）をサブジェクトに追加する必要があります。

J2EE のセキュリティ認可

OracleAS JAAS Provider のカスタム LoginModule フレームワークでは、J2EE の宣言によるセキュリティ・モデルをサポートします。つまり、サブジェクト・ベースの認可により、デプロイメント・ディスクリプタ（web.xml および ejb-jar.xml など）に宣言されている J2EE のセキュリティ制約が施行されます。可能な場合は、J2EE セキュリティ・モデルを利用することをお薦めします。

コールバックのサポート

OracleAS JAAS Provider では、標準の javax.security.auth.callback の名前（NameCallback）およびパスワード（PasswordCallback）の各コールバックをサポートします。

デバッグのヒント

セキュアなアプリケーションのデバッグ時に、次の問題に留意してください。

- デバッグ・ロギング
- ログイン・モジュールのデバッグ

デバッグ・ロギング

JAAS Provider のデバッグ・ロギングを有効にするには、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。

これは、OC4J インスタンスの `<java-options>` 設定を変更することで実行できます。Oracle Application Server では、通常、Oracle Enterprise Manager 10*g* Application Server Control コンソールを使用してこれらの設定を管理します。この場合、これらの設定は、`opmn.xml` に格納されます。OC4J インスタンスのホームページで、次の操作を実行します。

1. 「管理」を選択します。
2. 「管理」ページで、「サーバー・プロパティ」を選択します。
3. 「サーバー・プロパティ」ページの「コマンドライン・オプション」に、デバッグ・ロギングのオプションを入力します（次の説明を参照してください）。

Oracle Application Server の外部で OC4J を実行する場合は、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、スタンドアロン OC4J 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
```

デバッグ・ロギングを有効にすると、OracleAS JAAS Provider によってコンソールにデバッグ出力が記録されます。Oracle Application Server では、デバッグ出力は *ORACLE_HOME*`/opmn/logs` ディレクトリに取得されます。

ログイン・モジュールのデバッグ

カスタム `LoginModule` にデバッグ・オプションを組み込むことをお薦めします。たとえば、デフォルト・ログイン・モジュール `RealmLoginModule` は、`debug` が `true` に設定されていると、診断出力を提供します。

カスタム・ログイン・モジュール使用時の EJB へのアクセス

カスタム `LoginModule` を使用して EJB にアクセスするには、次の作業を実行する必要があります。

- OC4J `home` インスタンスの `jazn-data.xml` ファイルで、ユーザー `JDOE_ENDUSER` に `login` パーミッションを付与します。
- `orion-application.xml` で、ユーザー `JDOE_ENDUSER` にネームスペース・アクセス権を付与します。

`JDOE_ENDUSER` にログイン・パーミッションを付与するには、次の例のように JAZN Admintool を使用します。

```
java -jar jazn.jar -grantperm login -user JDOE_ENDUSER
oracle.j2ee.server.rmi.RMIPermission
```

`JDOE_ENDUSER` にネームスペース・アクセス権を付与するには、`orion-application.xml` を編集して次のように `<namespace-access>` 要素を追加します。

```
<namespace-access>
  <read-access>
    <namespace-resource root="">
```

```
      <security-role-mapping>
        <user name="JDOE_ENDUSER" />
      </security-role-mapping>
    </namespace-resource>
  </read-access>
</namespace-access>
```

ログイン・モジュールの追加と削除

ログイン・モジュールの追加と削除には、JAZN Admintool を使用します。JAZN Admintool の実行の基本情報は、4-3 ページの「Admintool の概要」を参照してください。

```
java -jar jazn.jar -addloginmodule application_name login_module_name
     control_flag [optionname=value ...]
java -jar jazn.jar -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 設定には、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 10-1 を参照してください。

表 10-1 ログイン・モジュール制御フラグ

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）。
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、次のように指定します。

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

MyLoginModule を myapp から削除するには、次のように指定します。

```
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
```

ログイン・モジュールのリスト表示

ログイン・モジュールのリスト表示には、JAZN Admintool を使用します。

関連項目：

- JAZN Admintool の実行の基本情報は、4-3 ページの「Admintool の概要」を参照してください。

```
java -jar jazn.jar -listloginmodules [application_name [login_module_class]]
```

`-listloginmodules` オプションを使用して、指定の *`application_name`* 内のログイン・モジュール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ログイン・モジュールが表示されます。*`application_name`* の後に *`login_module_class`* を指定すると、アプリケーション内の指定したクラスに関する情報のみが表示されます。

たとえば、アプリケーション `myapp` のログイン・モジュールをすべて表示するには、次のように指定します。

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

Admintool のシェルの場合

```
JAZN:> listloginmodules myapp
```

パッケージ化とデプロイ

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 JAAS Provider は、デフォルト・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 OracleAS 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化およびデプロイするときには、次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

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 OracleAS 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、*`ORACLE_HOME`*`/j2ee/`*`instance_name`*`/lib/ext` ディレクトリにデプロイすることです。

関連資料：

- 追加情報は、次の URL を参照してください。

  `http://java.sun.com/j2se/1.4.2/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、アプリケーションの一部としてログイン・モジュールをパッケージ化するだけで、OracleAS JAAS Provider は検出できます。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度パッケージ化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っても、`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

OracleAS 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 OracleAS 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`（インスタンス・レベル）
- `orion-application.xml`（アプリケーション固有）

関連資料：

- `<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。

アプリケーションの構成

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 ファイル
- web.xml または ejb-jar.xml ファイル
- orion-application.xml ファイル
- oc4j-ra.xml ファイル（J2EE Connector Architecture）

この項では、これらの構成ファイルの詳細を説明します。

注意： カスタム・ログイン・モジュールを使用する際は、XML ベース・プロバイダを選択する必要があります。10-2 ページの「カスタム JAAS ログイン・モジュールの統合」を参照してください。

---

jazn-data.xml ファイル

すべてのログイン・モジュールの構成情報は、OC4J home インスタンスの jazn-data.xml ファイルに格納されます。このファイルは、通常、*ORACLE_HOME*/j2ee/home/config ディレクトリにあります。

> **注意：** home インスタンスの jazn-data.xml には、admin および anonymous のアカウントが含まれている必要があります。これらのアカウントを削除しないでください。削除すると、OracleAS JAAS Provider の管理機能は動作しません。

home インスタンスの jazn-data.xml ファイルは、アプリケーションを新規 OC4J インスタンスにデプロイするたびに変更する必要があります。このファイルは、JAZN Admintool を使用して編集します。

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>

この要素には、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。

例 10-1 <jazn-loginconfig> 要素の例

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

このサンプルは、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。

> **注意：** RealmLoginModule に関するログインの構成情報は削除しないでください。

<jazn-policy>

この要素には、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。EJB へのファット・クライアント・アクセスを可能にする場合は、明示的にパーミッションを有効化する必要があります。カスタム LoginModule を OC4J にデプロイする場合は、通常、カスタム・プリンシパル・クラスまたはタイプを使用します。これらのタイプに対するパーミッションの付与または取消しには、JAZN Admintool を使用します。

> **注意：** これらのポリシーは、*ORACLE_HOME*/j2ee/home/config ディレクトリにある home インスタンスの jazn-data.xml ファイルに設定する必要があります。

例 10-2 <jazn-policy> 要素の例

```
<jazn-policy>
  <grant>
   <grantee>
     <principals>
      <principal>
        <class>oracle.security.jazn.samples.SampleUser</class>
        <name>admin</name>
      </principal>
    </principals>
   </grantee>
   <permissions>
     <permission>
       <class>oracle.j2ee.server.rmi.RMIPermission</class>
       <name>login</name>
     </permission>
   </permissions>
  </grant>
</jazn-policy>
```

このサンプルは、ターゲット名 `login` を持つパーミッション `oracle.j2ee.server.rmi.RMIPermission` を、クラス `oracle.security.jazn.samples.SampleUser` を持つプリンシパル `admin` 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。

関連項目：

- JAZN Admintool の詳細は、第 8 章「XML ベース・プロバイダの構成」を参照してください。

web.xml または ejb-jar.xml ファイル

J2EE の宣言によるセキュリティをアプリケーションで利用するには、IDE を使用するか、手動で `web.xml` または `ejb-jar.xml` ファイルを編集して、適切なセキュリティ制約を構成する必要があります。

関連資料：

- これらのファイルの詳細は、次の URL の J2EE 標準ドキュメントを参照してください。

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

orion-application.xml ファイル

このファイルは、OC4J にデプロイされたアプリケーションごとに生成されるコンテナ固有のデプロイメント・ディスクリプタです。次の要素は、カスタム・ログイン・モジュールの作成に関連します。

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

注意： この項では、セキュリティに関連する要素についてのみ説明します。

関連資料：

- orion-application.xml ファイルの詳細は、『Oracle Application Server Containers for J2EE ユーザーズ・ガイド』を参照してください。

<jazn>

次の <jazn> 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、OracleAS JAAS Provider に対して、アプリケーション固有の jazn-data.xml ファイルで定義されたユーザーおよびロールではなく、認証済 Subject インスタンス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。

  1 つ以上の 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ることを保証する必要があります。Subject インスタンス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。

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

関連項目：

- <jazn> 要素の検索方法については、4-3 ページの「<jazn> 要素の検索」を参照してください。

<security-role-mapping>

J2EE セキュリティ制約を web.xml または ejb-jar.xml ファイルに設定する場合、セキュリティ・ロール・マッピングを構成する必要があります。

オプションの <security-role-mapping> 要素には、静的セキュリティ・ロール・マッピング情報を記述します。J2EE セキュリティ制約をアプリケーションのデプロイメント・ディスクリプタ（web.xml または ejb-jar.xml）に設定する場合、セキュリティ・ロール・マッピングを構成する必要があります。

関連項目：

- 12-3 ページの「EJB アプリケーションの認証と認可」

<library>

この要素は、アプリケーションに関連するクラスパスに追加します。次に例を示します。

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

oc4j-ra.xml ファイル（J2EE Connector Architecture）

次の例のように、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。この例では、Oracle JDBC を介してデータベースに接続するための <config-property> の設定も示しています。

```
<connector-factory connector-name="myBlackbox" location="eis/myEIS1">
   <config-property name="connectionURL"
                    value="jdbc:oracle:thin:@localhost:5521/myservice" />
   <security-config use="jaas-module">
      <jaas-module>
         <jaas-application-name>JAASModuleDemo</jaas-application-name>
      </jaas-module>
   </security-config>
</connector-factory>
```

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、パッケージ化およびデプロイの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。

関連資料 :

- 次の URL にある javax.security.auth.spi.LoginModule の Javadoc を参照してください。

  http://java.sun.com/j2se/1.4.2/docs/api/

パッケージ化

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化します。Web アプリケーションの場合は、各クラスを WEB-INF/classes の下に含めます。

デプロイ

LoginModule を home インスタンスの jazn-data.xml ファイルにデプロイするには、次のようにします。

1. アプリケーションのログイン・モジュールを jazn-data.xml ファイルの <application> 要素で登録します。

   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 oracle.security.jazn.samples.SampleLoginModule が登録されます。

```
<application>
   <name>sampleLM</name>
   <login-modules>
      <login-module>
         <class>oracle.security.jazn.samples.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 oracle.j2ee.rmi.RMIPermission を付与する必要があります。

```
<grant>
  <grantee>
    <principals>
      <principal>
        <class>oracle.security.jazn.samples.SampleUser</class>
        <name>admin</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.j2ee.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>
```

カスタム・ログイン・モジュールの例

この項では、CallerInfo の例で使用される単純なカスタム LoginModule のソース・コードを示します。修正された例の完全なソース・コードは、次の URL の Oracle Technology Network で参照してください。

http://www.oracle.com/technology/index.html

例 10-3 SampleLoginModule.java

```
package oracle.security.jazn.samples;

import java.util.Set;
import java.util.Iterator;
import java.util.Map;
import java.security.Principal;
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
```

```
public class SampleLoginModule implements LoginModule {

    // initial state
    protected Subject _subject;
    protected CallbackHandler _callbackHandler;
    protected Map _sharedState;
    protected Map _options;

    // configuration options
    protected boolean _debug;

    // the authentication status
    protected boolean _succeeded;
    protected boolean _commitSucceeded;

    // username and password
    protected String _name;
    protected char[] _password;

    protected Principal[] _authPrincipals;


    /**
     * Initialize this <code>LoginModule</code>.
     * <p/>
     * <p/>
     *
     * @param subject         the <code>Subject</code> to be authenticated. <p>
     * @param callbackHandler a <code>CallbackHandler</code> for communicating
     *                        with the end user (prompting for usernames and
     *                        passwords, for example). <p>
     * @param sharedState     shared <code>LoginModule</code> state. <p>
     * @param options         options specified in the login
     *                        <code>Configuration</code> for this particular
     *                        <code>LoginModule</code>.
     */
    public void initialize(Subject subject,
                           CallbackHandler callbackHandler,
                           Map sharedState,
                           Map options) {
        this._subject = subject;
        this._callbackHandler = callbackHandler;
        this._sharedState = sharedState;
        this._options = options;

        // initialize any configured options
        _debug = "true".equalsIgnoreCase((String) _options.get("debug"));

        if (debug()) {
            printConfiguration(this);
        }
    }


    final public boolean debug() {
        return _debug;
    }


    protected Principal[] getAuthPrincipals() {
        return _authPrincipals;
    }
```

```
/**
 * Authenticate the user by prompting for a username and password.
 * <p/>
 * <p/>
 *
 * @return true if the authentication succeeded, or false if this
 *         <code>LoginModule</code> should be ignored.
 * @throws FailedLoginException if the authentication fails. <p>
 * @throws LoginException       if this <code>LoginModule</code>
 *                              is unable to perform the authentication.
 */
public boolean login() throws LoginException {
    if (debug())
        System.out.println("\t\t[SampleLoginModule] login");

    if (_callbackHandler == null)
        throw new LoginException("Error: no CallbackHandler available " +
                "to garner authentication information from the user");

    // Setup default callback handlers.
    Callback[] callbacks = new Callback[] {
        new NameCallback("Username: "),
        new PasswordCallback("Password: ", false)
    };

    try {
        _callbackHandler.handle(callbacks);
    } catch (Exception e) {
        _succeeded = false;
        throw new LoginException(e.getMessage());
    }


    String username = ((NameCallback)callbacks[0]).getName();
    String password =
               new String(((PasswordCallback)callbacks[1]).getPassword());
    if (debug())
    {
        System.out.println("\t\t[SampleLoginModule] username : " + username);
    }

    // Authenticate the user. On successfull authentication add principals
    // to the Subject. The name of the principal is used for authorization by
    // OC4J by mapping it to the value of the name attribute of the group
    // element in the security-role-mapping for the application.
    if(username.equals("developer") && password.equals("welcome"))
    {
        _succeeded = true;
        _name = "developer";
        _password = password.toCharArray();
        _authPrincipals = new SamplePrincipal[2];
        //Adding username as principal to the subject
        _authPrincipals[0] = new SamplePrincipal("developer");
        //Adding role developers to the subject
        _authPrincipals[1] = new SamplePrincipal("developers");
    }

    if(username.equals("manager") && password.equals("welcome"))
    {
        _succeeded = true;
        _name = "manager";
        _password = password.toCharArray();
```

```
            _authPrincipals = new SamplePrincipal[3];
            //Adding username as principal to the subject
            _authPrincipals[0] = new SamplePrincipal("manager");
            //Adding roles developers and managers to the subject
            _authPrincipals[1] = new SamplePrincipal("developers");
            _authPrincipals[2] = new SamplePrincipal("managers");
        }


        ((PasswordCallback)callbacks[1]).clearPassword();
        callbacks[0] = null;
        callbacks[1] = null;

        if (debug())
        {
            System.out.println("\t\t[SampleLoginModule] success : " + _succeeded);
        }

        if (!_succeeded)
            throw new LoginException
                        ("Authentication failed: Password does not match");

        return true;
    }


    /**
     * <p> This method is called if the LoginContext's
     * overall authentication succeeded
     * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
     * succeeded).
     * <p/>
     * <p> If this LoginModule's own authentication attempt
     * succeeded (checked by retrieving the private state saved by the
     * <code>login</code> method), then this method associates a
     * <code>Principal</code>
     * with the <code>Subject</code> located in the
     * <code>LoginModule</code>.  If this LoginModule's own
     * authentication attempted failed, then this method removes
     * any state that was originally saved.
     * <p/>
     * <p/>
     *
     * @return true if this LoginModule's own login and commit
     *         attempts succeeded, or false otherwise.
     * @throws LoginException if the commit fails.
     */
    public boolean commit()
            throws LoginException {
        try {

            if (_succeeded == false) {
                return false;
            }

            if (_subject.isReadOnly()) {
                throw new LoginException("Subject is ReadOnly");
            }

            // add authenticated principals to the Subject
            if (getAuthPrincipals() != null) {
                for (int i = 0; i < getAuthPrincipals().length; i++) {
                    if(!_subject.getPrincipals().contains(getAuthPrincipals()[i]))
```

```
{
                        _subject.getPrincipals().add(getAuthPrincipals()[i]);
                    }
                }
            }

            // in any case, clean out state
            cleanup();
            if (debug()) {
                printSubject(_subject);
            }

            _commitSucceeded = true;
            return true;

        } catch (Throwable t) {
            if (debug()) {
                System.out.println(t.getMessage());
                t.printStackTrace();
            }
            throw new LoginException(t.toString());
        }
    }


    /**
     * <p> This method is called if the LoginContext's
     * overall authentication failed.
     * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
     * did not succeed).
     * <p/>
     * <p> If this LoginModule's own authentication attempt
     * succeeded (checked by retrieving the private state saved by the
     * <code>login</code> and <code>commit</code> methods),
     * then this method cleans up any state that was originally saved.
     * <p/>
     * <p/>
     *
     * @return false if this LoginModule's own login and/or commit attempts
     *         failed, and true otherwise.
     * @throws LoginException if the abort fails.
     */
    public boolean abort() throws LoginException {
        if (debug()) {
            System.out.println
                    ("\t\t[SampleLoginModule] aborted authentication attempt.");
        }

        if (_succeeded == false) {
            cleanup();
            return false;
        } else if (_succeeded == true && _commitSucceeded == false) {
            // login succeeded but overall authentication failed
            _succeeded = false;
            cleanup();
        } else {
            // overall authentication succeeded and commit succeeded,
            // but someone else's commit failed
            logout();
        }
        return true;
    }
```

```
protected void cleanup() {
    _name = null;
    if (_password != null) {
        for (int i = 0; i < _password.length; i++) {
            _password[i] = ' ';
        }
        _password = null;
    }
}


protected void cleanupAll() {
    cleanup();

    if (getAuthPrincipals() != null) {
        for (int i = 0; i < getAuthPrincipals().length; i++) {
            _subject.getPrincipals().remove(getAuthPrincipals()[i]);
        }
    }
}


/**
 * Logout the user.
 * <p/>
 * <p> This method removes the <code>Principal</code>
 * that was added by the <code>commit</code> method.
 * <p/>
 * <p/>
 *
 * @return true in all cases since this <code>LoginModule</code>
 *         should not be ignored.
 * @throws LoginException if the logout fails.
 */
public boolean logout() throws LoginException {
    _succeeded = false;
    _commitSucceeded = false;
    cleanupAll();
    return true;
}

// helper methods //

protected static void printConfiguration(SampleLoginModule slm) {
    if (slm == null) {
        return;
    }
    System.out.println("\t\t[SampleLoginModule] configuration options:");
    if (slm.debug()) {
        System.out.println("\t\t\tdebug = " + slm.debug());
    }
}


protected static void printSet(Set s) {
    try {
        Iterator principalIterator = s.iterator();
        while (principalIterator.hasNext()) {
            Principal p = (Principal) principalIterator.next();
            System.out.println("\t\t\t" + p.toString());
        }
    } catch (Throwable t) {
    }
```

```
    }


    protected static void printSubject(Subject subject) {
        try {
            if (subject == null) {
                return;
            }
            Set s = subject.getPrincipals();
            if ((s != null) && (s.size() != 0)) {
                System.out.println
                     ("\t\t[SampleLoginModule] added the following Principals:");
                printSet(s);
            }

            s = subject.getPublicCredentials();
            if ((s != null) && (s.size() != 0)) {
                System.out.println
              ("\t\t[SampleLoginModule] added the following Public Credentials:");
                printSet(s);
            }
        } catch (Throwable t) {
        }
    }
}
```

この LoginModule で使用される Principal を例 10-4 に示します。

例 10-4 SamplePrincipal の例

```
package oracle.security.jazn.samples;

import java.security.Principal;

public class SamplePrincipal implements Principal {

    private String _name = null;


    public SamplePrincipal(String name) {
        _name = name;
    }


    public boolean equals(Object another) {
        return ((SamplePrincipal)another).getName().equals(_name);
    }


    public String getName() {
        return _name;
    }


    public int hashCode() {
        return _name.hashCode();
    }


    public String toString() {
        return "[SamplePrincipal] : " + _name;
    }

}
```

11

OC4J と SSL の構成

OC4J では、セキュアな AJP を使用して、Oracle Application Server 環境での Oracle HTTP Server と OC4J 間の Secure Sockets Layer（SSL）通信がサポートされます。このプロトコルは、セキュアなバージョンの Apache JServ Protocol で、Oracle HTTP Server では OC4J との通信に使用されます。ただし、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・ユーザーには見えないことに注意してください。

この章では、SSL を利用するための OC4J の構成についてのみ説明します。他の Oracle Application Server コンポーネントの構成の詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

この章には、次の項目が含まれます。

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とは無関係です。この章では、Oracle HTTP Server と OC4J 間のセキュアな通信についてのみ説明します。

この章では、セキュリティおよび SSL の概念についてある程度の予備知識があることを前提にしています。

関連資料：

Oracle Application Server セキュリティおよび Oracle HTTP Server の追加情報は、次のマニュアルを参照してください。

- 『Oracle Application Server セキュリティ・ガイド』
- 『Oracle HTTP Server 管理者ガイド』

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、その鍵のやり取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。（CA 自体の証明書は常に自己署名であることに注意してください。）

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。キーストアのパスワードは不明瞭化されます。Oracle HTTP Server には、これと同じ用途を持つ Wallet と呼ばれるものがあります。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。

関連資料：

- `keytool` の詳細は、次の URL 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 での鍵と証明書の使用

次の手順では、OC4J で SSL 通信に鍵と証明書を使用する方法について説明します。ここで説明するのはサーバー・レベルの手順で、一般に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前、通常は Oracle Application Server インスタンスの初期設定時に実行します。

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ることに注意してください。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 では、これと同じ用途を持つ Wallet と呼ばれるものを使用します。

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。

Oracle Wallet Manager には、Oracle Wallet のための機能があります。この機能は、キーストアのための `keytool` の機能に相当します。

関連資料：

- Oracle Wallet Manager の詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

次に、OC4J と Oracle HTTP Server 間で証明書を使用する手順を示します。

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。

   独自の署名を次の手順で生成します。

   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、事実上、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します。

   a. 手順 1 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、Oracle Application Server Certificate Authority（OCA）が組み込まれています。OCA により、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認識される可能性が低くなります。

関連資料：

- OCA の詳細は、『Oracle Application Server Certificate Authority 管理者ガイド』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 Oracle Application Server のスコープおよび制御の対象外のため、ここでは説明しません。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を実行しますが、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。

関連資料：

- Wallet および Oracle Wallet Manager の詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

前述の手順 1 と 2 に加えて、次の手順も必要に応じて実行します。

1. **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：** エンティティの証明書を取得して、Oracle HTTP Server にインポートします。問題の OC4J の証明書が自己署名されているかどうかによって、詳細は次のように異なります。

   OC4J が自己署名された証明書を持っている（基本的に Oracle HTTP Server は OC4J を信頼していない）場合：

   a. OC4J で `keytool` を使用して OC4J の証明書をエクスポートします。この手順により、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。

   b. 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。

   または、OC4J が（Oracle HTTP Server で信頼されていない）別のエンティティが署名した証明書を持っている場合：

   a. そのエンティティから証明書を適切な方法（エンティティの証明書をエクスポートするなど）で、取得します。正確な手順は、エンティティによって大きく異なります。

   b. Oracle HTTP Server で Oracle Wallet Manager を使用してエンティティの証明書をインポートします。

2. **Oracle HTTP Server の証明書に OC4J で信頼され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**（11-8 ページの「クライアント認証の要求」を参照）：

   a. そのエンティティから証明書を適切な方法（エンティティの証明書をエクスポートするなど）で、取得します。正確な手順は、エンティティによって大きく異なります。

   b. OC4J で `keytool` を使用してエンティティの証明書をインポートします。

注意： Oracle HTTP Server と OC4J 間での SSL を介した通信中には、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. この初期交換後の通信はすべて暗号化されます。

例：SSL 証明書の作成と独自の署名の生成　この例は、証明書の署名を取得するステップに相当し、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 に属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が要求されます。

```
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/~j1m5path/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。

OC4J での SLL の有効化

Oracle HTTP Server と OC4J 間のセキュアな接続のために、次の項で説明するように、接続の両端で構成手順を実行する必要があります。

SSL のための Oracle HTTP Server の構成

Oracle HTTP Server で、mod_oc4j.conf ファイル内の SSL 設定がセキュアな通信に適した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。

関連資料 :

- 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。

例 11-1 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 のレジストリ エディタで、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. *ORACLE_HOME*/j2ee ディレクトリに移動します。

   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、パスワードに 123456、有効期間に 21 日を指定して、ファイル mykeystore に属するキーストアを生成します。

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

このツールでは、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が要求されます。

```
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/~j1m5path/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 ファイルをコピーして *ORACLE_HOME*/j2ee/home/config/secure-web-site.xml ファイルを作成します。
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="<yourkeystore>"
            keystore-password="<yourpassword>" />
```

yourkeystore はキーストアへのフルパス、*yourpassword* 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. （まだ設定されていない場合）`secure-web-site.xml` ファイルを指すように `server.xml` を編集します。

a. `secure-web-site.xml` ファイルが読み取られるように、ファイル `server.xml` 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` ファイル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。

```
<!-- Comment out the following to remove SSL -->
<web-site path="./secure-web-site.xml" />
<!-- Comment out the following to remove non-SSL -->
<default-site path="./default-web-site.xml" />
```

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。Oracle Application Server 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・ポイントはキーストア内の証明書と一致するとして OC4J で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 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。

`<ssl-config>` 要素の `needs-client-auth="true"` 設定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. 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（11-3 ページの「OC4J および Oracle HTTP Server での鍵と証明書の使用」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。

   注意 : スタンドアロン OC4J を実行している場合、証明書はクライアント上に保存されます。

5. 証明書を提供します。
   a. Oracle HTTP Server を実行している場合 : 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。
   b. OC4J をスタンドアロン環境で実行している場合 :
      – 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。
      – 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。

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。

関連資料 :

- Oracle Wallet Manager、PKI およびセキュリティ基礎の情報は、『Oracle Application Server 管理者ガイド』を参照してください。
- JSSE のドキュメントは、次の URL を参照してください。

  ```
  http://java.sun.com/products/jsse/doc/guide/
  API_users_guide.html
  ```

- `java.net` パッケージのドキュメントは、次の URL を参照してください。

  ```
  http://java.sun.com/j2se/1.4.2/docs/api/
  ```

一般的な SSL 問題の解決策

この項では、いくつかの一般的な SSL エラーと、その原因および解決策について説明します。その後に、一般的な SSL のデバッグ方法について簡単に説明します。

一般的な SSL エラーと解決策

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 : 後続の空白があります。これは、keytool ユーティリティでは使用できません。

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 : 旧バージョンの JDK から keytool ユーティリティを使用していると思われます。

処置 : サポートされる最新 JDK の keytool ユーティリティ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 ユーティリティでは、キーストア内でルート CA の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 : 次のコマンドを実行します。

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

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

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

証明書署名要求（CSR）から証明書を取得して、次のコマンドを実行します。

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

No available certificate corresponds to the SSL cipher suites that are enabled

原因 : 証明書に誤りがあります。

処置 : 問題を特定し、訂正します。

一般的な SSL のデバッグ方法

OC4J スタンドアロンで開発している場合、Java Secure Socket Extension（JSSE）実装からの詳細なデバッグ情報を表示できます。オプション・リストを取得するには、OC4J を次のように起動します（% はシステム・プロンプト）。

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

次のように起動すると、全詳細情報が使用できます。

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

このコマンドにより、ブラウザ・リクエスト・ヘッダー、サーバー HTTP ヘッダー、サーバー HTTP ボディ、コンテンツの長さ（暗号化の前後）および SSL バージョンが表示されます。

12

EJB のセキュリティの構成

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 ファイルにクラスパス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・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
```

コード実装内の JNDI プロパティ

Java コード内でも、前項と同様にプリンシパル・ユーザー名と資格証明を設定します。たとえば、コンテナ内で実行される JavaBeans は、リモート EJB の検索用に作成される InitialContext 内で資格証明を渡します。

たとえば、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",
      "oracle.j2ee.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);
```

注意： ApplicationClientInitialContextFactory は、oc4jclient.jar ファイル内にあります。

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。

関連資料 :

- 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。

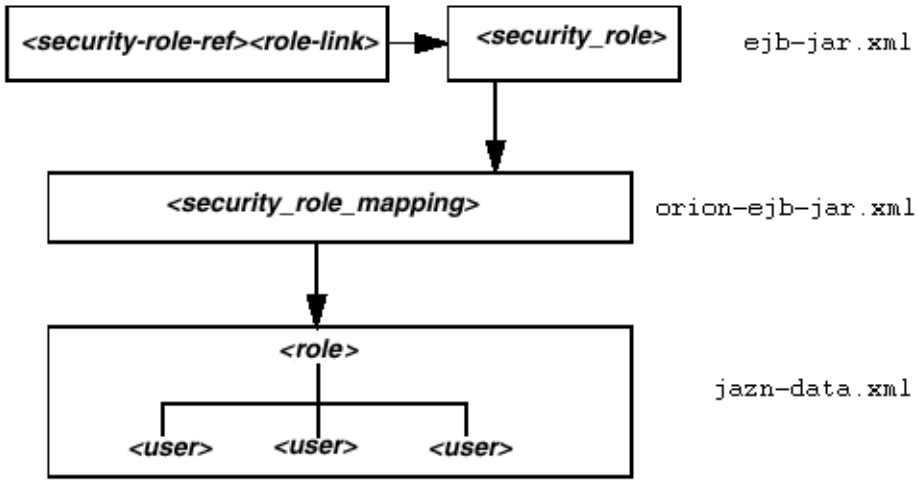
この項では、認証と認可について EJB デプロイメント・ディスクリプタ内の XML 構成に重点を置いて説明します。EJB の認可は、EJB および OC4J 固有のデプロイメント・ディスクリプタ内で指定されます。デプロイメント・ディスクリプタ内では、セキュリティの認可部分を次のように管理できます。

- 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。ユーザー名とパスワードのペアは、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
- run-as 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。

関連資料：

- 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。

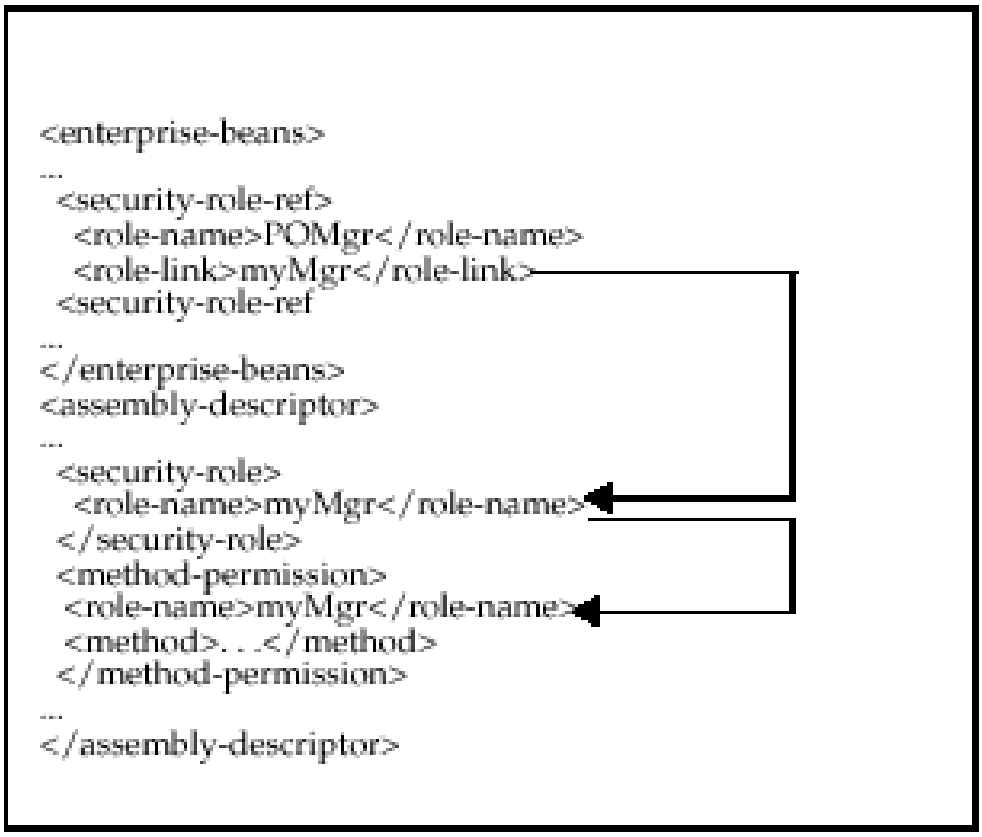
EJB デプロイメント・ディスクリプタでの論理ロールの指定

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。

関連項目：

- 12-8 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」

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 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 <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、通常は必要ありません。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 <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 の定義に関連付けられます。

```
<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 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を結合したものが付与されます。

<method-permission> 要素の <method> サブ要素は、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。

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。

run-as 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 run-as 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。

関連資料 :

- 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。

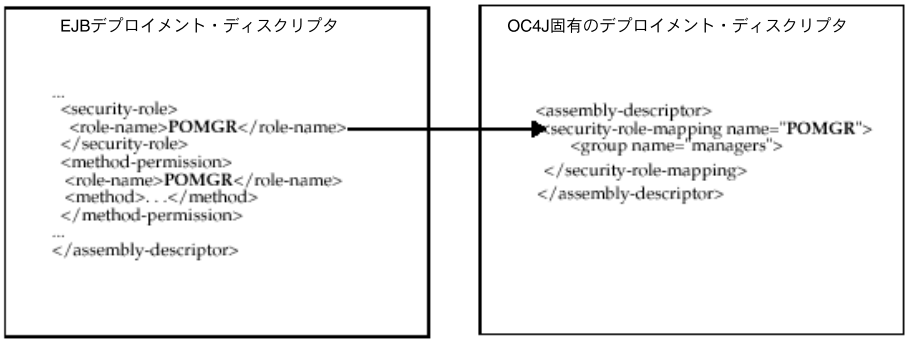
```
<security-role-mapping name="POMGR">
  <user name="guest" />
</security-role-mapping>
```

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" />
</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 に設定される場合、<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。

関連項目：

- 次の「EJB クライアントでの資格証明の指定」

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 の検索用に作成される `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=
   oracle.j2ee.server.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://myhost/ejbsamples
```

アプリケーション・プログラムで、次の例に示すようにリモート 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",
   "oracle.j2ee.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

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 OC4J 実装について説明します。ここでは、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と、Oracle HTTPS および JSSE の使用について説明します。この章には、次の項目が含まれます。

- Oracle HTTPS とクライアント
- Oracle HTTPS の機能概要
- デフォルトのシステム・プロパティの指定
- Oracle HTTPS の例
- JSSE と HTTPClient の使用

この章では、鍵および証明書をすでに取得していることを前提としています。

関連項目：

- Secure Sockets Layer を使用するための OC4J の構成の概要は、第 11 章「OC4J と SSL の構成」を参照してください。

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とは無関係です。（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・ユーザーには見えないことに注意してください。）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。

Oracle HTTPS とクライアント

HTTPS は、クライアントとサーバー間の対話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取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` パッケージに精通している 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対話を簡単に保護できます。

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、Java Secure Socket Extension（JSSE）と OracleSSL の 2 つの異なる SSL 実装がサポートされます。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、公開鍵インフラストラクチャ（PKI）デジタル証明と Wallet のサポートを提供するために、13-7 ページの「Oracle HTTPS の例」に示すメソッドが追加されています。

関連資料：

- `HTTPClient` の Javadoc の『Oracle Application Server HTTPClient API Reference』

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 を使用して個々の証明書を追加できます。資格証明がすべて揃うと、`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、暗号スイートの選択、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

次の各項では、Oracle HTTPS 機能を詳細に説明します。

- SSL 暗号スイート
- OracleSSL でサポートされる SSL 暗号スイート
- JSSE でサポートされる SSL 暗号スイート
- セキュリティ対応アプリケーションのサポート
- java.net.URL フレームワークのサポート

SSL 暗号スイート

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、暗号スイートと呼ばれます。SSL 接続の参加者は、特定の暗号スイート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暗号スイートがサポートされます。ここでは、これらの暗号スイートについて説明します。

暗号スイートの選択

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 暗号スイート

OracleSSL では、表 13-1 に示す暗号スイート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 13-1 OracleSSL でサポートされる暗号スイート

暗号スイート	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 暗号スイート

JSSE では、表 13-2 に示す暗号スイート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 13-2 JSSE でサポートされる暗号スイート

暗号スイート	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_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、接続に使用された暗号スイート、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。

関連資料 :

- 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。

- 暗号スイート選択オプション
- 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 クライアントのかわりとして選択します。

```
java.protocol.handler.pkgs=HTTPClient
```

関連資料 :

- JSSE を使用するためのクライアントの構成方法は、次の「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。
- Wallet および Oracle Wallet Manager の詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

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 プロパティ
- javax.net.ssl.KeyStorePassword プロパティ
- Oracle.ssl.defaultCipherSuites プロパティ（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 プロパティ

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
```

この構文で、*`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` がスローされます。

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 プロパティ

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
```

この構文で、*`welcome1`* は、Wallet ファイルを開くために必要なパスワードです。

重要： 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 プロパティ（OracleSSL のみ）

このプロパティは、カンマ区切りの暗号スイート・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暗号スイートは、新規 HTTPS 接続のデフォルト暗号スイートとして使用されます。

関連項目：

- OracleSSL でサポートされる暗号スイートの詳細なリストは、13-4 ページの表 13-1 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.servlet.request.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、次のように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、次のように 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

Oracle Application Server 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。

関連資料：

- JSSE の詳細は、次の URL にある Sun 社のドキュメントを参照してください。

  http://java.sun.com/products/jsse/

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。例 13-1 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。

例 13-1 HTTPClient での JSSE の使用

```
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。
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。
   a. （**接続ごと**）クライアントで HTTPConnection インスタンスの setSSLSocketFactory(SSLSocketFactory factory) メソッドをコールします。
   b. （**VM 全体**）クライアントで HTTPConnection クラスの静的メソッド 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 を使用するように設定されています。追加構成は不要であり、基本的な使用方法は同じです。

注意： SSL の JSSE 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。

関連資料：

- keytool の使用方法の詳細は、次の URL を参照してください。

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

14

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。次の「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。
- 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。14-3 ページの「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` 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 構成ファイルでは、14-3 ページの「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。

OracleAS 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。

注意 : セキュリティ上の理由で、Oracle Internet Directory に格納されている資格証明は、通常、復号化された（クリアテキスト）形式で取得できません。つまり、LDAP ベースの JAAS Provider はアプリケーションのパスワード・マネージャとして使用できません。これを解決するために、アプリケーションで LDAP ベースの JAAS Provider がユーザー・マネージャとして使用される場合でも、XML ベースの JAAS Provider をアプリケーションのパスワード・マネージャとして指定できます。

そのためには、次のエントリを `application.xml` に追加します。

```
<password-manager>
  <jazn provider="XML"
   location="ORACLE_HOME/j2ee/instance_name/config/jazn-data.xml"/>
</password-manager>
```

追加しない場合は、パスワードは不明瞭化されません。

---

jazn-data.xml の編集

必要な場合は、`jazn-data.xml` 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（属性は 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、「->」に続けてユーザー名、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 を使用して JAZNUserManager 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。
- <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」がパスワードです（間接パスワードではありません）。

application.xml でのユーザー・マネージャの指定

application.xml の <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ユーザー・マネージャを指定します。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ユーザー・マネージャが使用されます。<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。

<password-manager> 要素には、インスタンス・レベルの jazn-data.xml ファイルのパス名を含める必要があります。

たとえば、通常のユーザー・マネージャに LDAP ベースのユーザー・マネージャを使用できますが、間接パスワードの認証には XML ベースのユーザー・マネージャを使用します。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。

注意： 交換可能なユーザー・マネージャをパスワード・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。

関連項目：

- 14-3 ページの「間接パスワードの作成」
- 4-6 ページの「ユーザー・マネージャの指定」

15

CSIv2 の構成

OC4J 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。OC4J 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 OracleAS JAAS Provider を構成する必要があります。詳細は、4-8 ページの表 4-2「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

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`（15-4 ページの「internal-settings.xml 内の CSIv2 のセキュリティ・プロパティ」を参照）
- `orion-ejb-jar.xml`（15-5 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）
- `ejb_sec.properties`（15-7 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」を参照）

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 15-1 にプロパティのリストを示します。

この表では、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各ファイルについて言及しています。各ファイルには、JDK 仕様の形式である 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 15-1 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`。
`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・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、またはホスト名のパターンを指定できます。ホスト名のパターンには、`*.example.com` や `*` などがあり、`*` のみの場合は、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。

表 15-1 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
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）。

次のことに留意してください。

- OC4J が（スタンドアロンではなく）Oracle Application Server 環境で Oracle Process Management Notification（OPMN）サービスにより起動されると、internal-settings.xml に指定したポートは上書きされます。IIOP SSL ポートは、キーストアやトラストストアの場所またはパスワードが存在しないか、間違っている場合、起動に失敗する可能性があります。この場合、*ORACLE_HOME*/opmn/logs/OC4J-home-default_island-1 のログを確認し、失敗の正確な原因を調査してください。
- OPMN が特定の OC4J インスタンスに対する 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 を介して有効化されるように設定されていても、IIOP は有効化されません。
- キーストアとトラストストアの設定は、スタンドアロン OC4J と完全な Oracle Application Server 環境の両方ともに internal-settings.xml でサポートされます。Oracle Application Server では、これらの値を設定する OPMN オプションが存在しないため、手動で構成する必要があります。
- Oracle Application Server と OPMN を組み合せて実行する場合、キーストアおよびトラストストアの絶対パス名が必要です。

次の例に、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 のデフォルト値より小さい値ですが、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、前項の「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。

OC4J 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 は使用できません。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、必ずしも SSL を使用して通信する必要はありません。
- ssl-port を指定しないと、orion-ejb-jar.xml 内で <ior-security-config> 要素を構成していても、サーバーでは IOR に CSIv2 のコンポーネント要素は挿入されません。

サーバー上で IIOP/SSL が有効化されている場合、OC4J 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します。サーバーおよびクライアント認証リスナーでは、その次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
- 両方とも指定しない

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・サイドの認証を使用する 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・サイドの認証はユーザー名とパスワードより優先されます。

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 ID 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 に信頼できるサーバーをリストします。サーバーは、internal-settings.xml 内の <sep-property> 要素の trusted-client プロパティに信頼できるクライアントをリストします。

関連項目：

- 15-7 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」
- 15-2 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」

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信頼できるクライアントの（または中間）ホストのリストも指定できます。

OC4J では、両方の種類の信頼が提供されます。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。次の「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。

<ior-security-config> の 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) >
```

<transport-config>

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 がその機能をサポートせず、使用しないことを意味します。`supported` は、Bean がその機能の使用をクライアントに許可することを意味します。`required` は、Bean がその機能の使用をクライアントに要求す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、送信されたとおり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ることができなかった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。この要素は、supported または none に設定します。required に設定することはできません。
- <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。この要素は、supported、required または none に設定する必要があります。

> **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> のプロパティのいずれかを 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>

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>: OC4J 10.1.2 実装では、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>

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>
```

ejb_sec.properties 内の EJB クライアントのセキュリティ・プロパティ

クライアントには、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・プロパティがあります。表 15-2 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、OC4J 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の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 *ORACLE_HOME*/j2ee/*instance_name*/config 内で検索します。このファイルの場所は、次のように明示的に指定できます。

-Dejb_sec_properties_location=*pathname*

表 15-2 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	クライアントがクライアント・サイドの認証をサポートするかどうかを指定します。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。
oc4j.iiop.ciphersuites	有効化する暗号スイート。有効な暗号スイートは次のとおりです。 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	サーバーとの初期接続時に SSL を使用するかどうかを指定します。
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テキスト形式（非暗号化）で送信するかどうかを指定します。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、このプロパティの効果はありません。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、またはホスト名のパターンを指定できます。ホスト名のパターンには、*.example.com や * などがあり、* のみの場合は、すべてのサーバーが信頼できることを意味します。

注意：

- ejb_sec.properties ファイルは、internal-settings.xml の存在しないクライアント・サイドの操作に使用されます。
- # のマークが付いているプロパティは、ejb_sec.properties で設定するか、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。

16

セキュリティ問題のトラブルシューティング

この章では、OC4J アプリケーションでセキュリティ上の問題を特定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。

- jazn.xml の検索場所
- JAZN Admintool
- カスタム・ログイン・モジュール
- LDAP ベース・プロバイダの問題
- サーブレット、runas-mode および doasprivileged-mode
- レルムの作成
- プリンシパルからのレルム名の削除
- JAAS プロバイダの指定

jazn.xml の検索場所

OracleAS JAAS Provider の起動時に、jazn.xml ファイルが検索されます。jazn.xml ファイルは、様々な場所に置くことができますが、通常、*ORACLE_HOME*/j2ee/home/config ディレクトリにあります。ただし、このファイルの場所をシステム・プロパティに指定すると、システム・プロパティに指定したファイルが優先されます。

OracleAS 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、OracleAS 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 Admintool

Admintool を使用するには、動的ライブラリのロードを制御する環境変数（Solaris の LD_LIBRARY_PATH など）を設定する必要があります。詳細は、2-9 ページの表 2-5「動的ライブラリ・パスの設定」を参照してください。

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。ツールを実行できるユーザーには、すべての権限が付与されます。つまり、本番環境では Admintool を保護することが必要不可欠です。そのために、通常、ファイル・システム・プロパティを使用します。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。

パーミッションを付与しようとして Admintool から「パーミッション・クラスが見つかりません。」というエラー・メッセージが発行された場合、付与するパーミッションがクラスパス内にないことを意味します。権限クラスを含む JAR を jdk/jre/lib/ext ディレクトリに置き、Admintool で検出できるようにする必要があります。

カスタム・ログイン・モジュール

カスタム LoginModule の作成時には、次の問題に注意する必要があります。

- サブジェクト・ベースの認可
- J2EE セキュリティ統合

サブジェクト・ベースの認可

アプリケーションでカスタム・ログイン・モジュールを使用する場合、サブジェクト（およびそれに含まれるプリンシパル）は、J2EE セキュリティ制約の評価などの認可に対する唯一の基礎として使用されます。認可時にすべての関連プリンシパルを必ず考慮に入れるために、ログイン・モジュールでは、JAAS 認証プロセスのコミット・フェーズ時に、関連プリンシパル（認証済ユーザーが属するすべてのロールおよびグループなど）をサブジェクトに追加する必要があります。

J2EE セキュリティ統合

カスタム LoginModule フレームワークでは、J2EE の宣言によるセキュリティ・モデルをサポートします。つまり、サブジェクト・ベースの認可を使用して、アプリケーションのデプロイメント・ディスクリプタ（web.xml および ejb-jar.xml など）に宣言されている J2EE のセキュリティ制約が施行されます。

可能な場合は、独自のセキュリティ実装を記述するのではなく、J2EE セキュリティ・モデルを利用することを J2EE 開発者にお薦めします。これにより、将来のリリースとの上位互換性が保証されます。

LDAP ベース・プロバイダの問題

LDAP ベース・プロバイダのトラブルシューティング時には、次の重要な問題があります。

- JAZN-LDAP 構成のチェック
- キャッシングの有効化と無効化

JAZN-LDAP 構成のチェック

Oracle Application Server インスタンスと Oracle Application Server Infrastructure を関連付けると、インストール時または Enterprise Manager の使用時に、インスタンスは LDAP ベース・プロバイダを使用するように自動的に構成されます。Oracle Internet Directory の場所とポートは、*ORACLE_HOME*/config/ias.properties ファイルによって決定されます。

LDAP ベース・プロバイダが適切に構成されていることを確認するには、次のようにします。

1. Enterprise Manager を使用して、ユーザー・マネージャが LDAP に設定されていることを確認します。
2. JAZN Admintool の -listrealms コマンドを発行して、LDAP ベース・プロバイダが Oracle Internet Directory からデータを取得できることを確認します。

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

3. Admintool から通信エラーというメッセージの応答が戻された場合は、Oracle Internet Directory が停止している可能性があります。
4. Admintool から無効な資格証明というメッセージの応答が戻された場合は、LDAP ユーザーおよび資格証明が正しく構成されていません。

キャッシングの有効化と無効化

LDAP キャッシングは、デフォルトでは有効化されています。キャッシングは JVM 単位でアプリケーション単位ではありません。パーミッションまたはロールの付与など、JAZN Admintool の管理コマンドを使用するには、キャッシングを無効化する必要があります。Admintool の使用後には、キャッシングを再度有効化する必要があります。

関連項目：

- キャッシングの有効化と無効化の詳細は、5-3 ページの「LDAP キャッシングの構成」を参照してください。

サーブレット、runas-mode および doasprivileged-mode

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、orion-web.xml または orion-application.xml ファイルの <jazn-web-app> 要素の runas-mode 属性および doasprivileged-mode 属性を設定する必要があります。

関連項目：

- 4-11 ページの「J2EE 認可の構成」

レルムの作成

重要なのは、適切なツールを使用してレルムを作成することです。一般に、LDAP ベース・プロバイダまたは Oracle Application Server Single Sign-On を使用している場合は、Oracle Delegated Administration Service を使用してレルムを作成します。XML ベース・プロバイダを使用している場合は、JAAS Admintool を使用してレルムを作成します。JAAS Admintool を使用して作成したレルムは、外部レルムまたはアプリケーション・レルムです。レルム・ツリーでのこれらの場所は、ID 管理レルムとは異なります。

プリンシパルからのレルム名の削除

一部のアプリケーションでは、様々なメソッド・コールで戻されるプリンシパルを解析しないようにする場合があります。OracleAS JAAS Provider は、戻されるプリンシパルにレルム名が含まれないように構成できます。そのためには、次の例のように、プロパティ `jaas.username.simple` をインスタンス・レベルの *ORACLE_HOME*`/j2ee/`*instance_name*`/config/jazn.xml` ファイルの `<jazn>` 要素に追加します。

```
<property name="jaas.username.simple" value="true" />
```

このプロパティは、次のメソッドの戻り値に影響します。

- `javax.servlet.http.HTTPServletRequest` の場合：`getRemoteUser()` および `getUserPrincipal()` の各メソッド
- `javax.ejb.EJBContext` の場合：`getCallerIdentity()` および `getCallerPrincipal()` の各メソッド

JAAS プロバイダの指定

次のような例外およびスタック・トレースを検討します。

```
Exception in thread "main" java.lang.SecurityException: Unable to locate a login
configuration
at com.sun.security.auth.login.ConfigFile.<init>(ConfigFile.java:97)
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance
```

この場合、OracleAS JAAS Provider を JAAS ポリシー・プロバイダとして指定することに失敗している可能性があります。4-5 ページの「代替ポリシー・プロバイダの指定（オプション）」を参照してください。

17

セキュリティのヒント

この章では、次の項目別にヒントを示します。

- HTTPS のヒント
- 全体的なセキュリティのヒント
- JAAS のヒント

この章で説明するヒントは、次の URL にある Oracle Technology Network からダウンロード可能な『Security Best Practices』から抜粋したものです。

```
http://www.oracle.com/technology/index.html
```

更新情報の有無を OTN の Web サイトでチェックしてください。

HTTPS のヒント

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。

- 弱い暗号化を使用できないように 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 システム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 1 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のヒント

全体的なセキュリティの構成に関する提案事項は、次のとおりです。

- 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。Oracle Application Server 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待機時間が発生します。

  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 のヒント

JAAS に関する提案事項は、次のとおりです。

- ユーザー管理を principals.xml から OracleAS JAAS Provider に移行します。以前のリリースの Oracle Application Server 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 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、クリアテキストにパスワードは格納されません。また、OracleAS JAAS Provider は、出荷時の Oracle Application Server Infrastructure（OracleAS Single Sign-On と Oracle Internet Directory 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、OracleAS JAAS Provider、OracleAS Single Sign-On および Oracle Internet Directory を拡張します。OC4J 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、OracleAS JAAS Provider、OracleAS Single Sign-On および Oracle Internet Directory 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。OracleAS Single Sign-On と Oracle Internet Directory 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。
- OracleAS JAAS Provider での認証メカニズムとして OracleAS Single Sign-On を使用します。OracleAS JAAS Provider では、様々な認証オプションを使用できます。ただし、次の理由から、できるだけ OracleAS Single Sign-On サーバー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 Oracle Application Server 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設定でき、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。
- OracleAS 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 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。

- OracleAS 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 OracleAS 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、OracleAS JAAS Provider の中央リポジトリとして Oracle Internet Directory を使用します。OracleAS 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 Oracle Internet Directory を使用するように構成する必要があります。Oracle Internet Directory 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。
- OracleAS JAAS Provider の認可機能を利用します。OracleAS JAAS Provider では、JAAS 1.0 仕様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

OracleAS JAAS Provider 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。サンプルは次のとおりです。

サンプル : jazn-data.xml の構成

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、ユーザーおよびロールが含まれています。

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-9_04.dtd">
<jazn-data>

<!-- JAZN Realm Data -->
<jazn-realm>
  <realm>
    <name>jazn.com</name>
    <users>
      <user>
        <name>anonymous</name>
        <description>The default guest/anonymous user</description>
      </user>
      <user>
        <name>SCOTT</name>
        <display-name>SCOTT</display-name>
        <credentials>!TIGER</credentials>
      </user>
      <user>
        <name>admin</name>
        <display-name>OC4J Administrator</display-name>
        <description>OC4J Administrator</description>
        <credentials>!welcome</credentials>
      </user>
      <user>
        <name>user</name>
        <description>The default user</description>
        <credentials>!456</credentials>
      </user>

            <!-- users used for password hiding -->
      <user>
        <name>pwForScott</name>
        <description>Password for database user Scott</description>
        <credentials>!TIGER</credentials>
      </user>
      <user>
        <name>pwForSSL</name>
        <description>Password for ssl key and trust stores</description>
        <credentials>!123456</credentials>
      </user>
      <user>
        <name>pwForSystem</name>
        <description>Password for database system user </description>
        <credentials>!manager</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>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>
      <role>
        <name>jmxusers</name>
        <display-name>JMX users</display-name>
        <description>
            Allows access to application level user defined MBeans
        </description>
        <members>
        </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/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>createrealm</actions>
  </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.policy.AdminPermission</class>
    <name>oracle.security.jazn.policy.RoleAdminPermission$jazn.com/*$</name>
  </permission>
  <permission>
    <class>oracle.j2ee.server.AdministrationPermission</class>
    <name>administration</name>
    <actions>administration</actions>
  </permission>
  <permission>
    <class>oracle.security.jazn.realm.RealmPermission</class>
    <name>jazn.com</name>
    <actions>droprealm</actions>
  </permission>
  <permission>
    <class>oracle.security.jazn.realm.RealmPermission</class>
    <name>jazn.com</name>
    <actions>dropuser</actions>
  </permission>
  <permission>
    <class>oracle.security.jazn.policy.RoleAdminPermission</class>
    <name>jazn.com/*</name>
  </permission>
  <permission>
    <class>oracle.j2ee.server.rmi.RMIPermission</class>
    <name>login</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>modifyrealmmetadata</actions>
  </permission>
  <permission>
    <class>oracle.security.jazn.policy.AdminPermission</class>
    <name>oracle.security.jazn.realm.RealmPermission$jazn.com$droprole</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>oracle.j2ee.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/jmxusers</name>
        </principal>
      </principals>
    </grantee>
    <permissions>
      <permission>
        <class>oracle.j2ee.server.rmi.RMIPermission</class>
        <name>login</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.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>
  <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>addAllRoles</name>
            <value>true</value>
          </option>
          <option>
            <name>debug</name>
            <value>false</value>
          </option>
        </options>
      </login-module>
    </login-modules>
  </application>
  <application>
    <name>oracle.security.jazn.oc4j.DigestAuthenticator</name>
    <login-modules>
      <login-module>
        <class>oracle.security.jazn.login.module.digest.DigestLoginModule</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>
</jazn-loginconfig>

</jazn-data>
```

サンプル : ユーザーのパーミッションの変更

例 A-2 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-1 に、例 A-2 のオブジェクトを示します。

表 A-1 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-2 ユーザーのパーミッションの変更

```
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 {
            JAZNConfig _jc = JAZNConfig.getJAZNConfig();
            RealmManager realmMgr = _jc.getRealmManager();
            Realm realm = realmMgr.getRealm("abcRealm");
            UserManager userMgr = realm.getUserManager();
            RoleManager roleMgr = realm.getRoleManager();
            final JAZNPolicy policy = _jc.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();
        }
    }
}
```

このサンプル・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

JAZN Admintool リファレンス

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。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。JAZN Admintool は、*ORACLE_HOME*/j2ee/home/jazn.jar にあります。

> **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。

この付録は、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の主要なリファレンスです（一部には、このマニュアルですでに説明済の情報が含まれます）。この章には次の項が含まれています。

JAZN Admintool のコマンドライン・オプションおよび構文の概要

この項では、JAZN Admintool の各オプションを機能と構文別に簡単に説明します。各オプションの詳細は、この付録の後の部分で説明します。

Admintool 全体の構文を次に示します。

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

注意： -user、-password および -clustersupport オプションを 1 つ以上使用する場合、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。次の例のように -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
```

Admintool の認証（XML ベース・プロバイダのみ）

-user *username* password *password*

関連項目：

- B-4 ページの「認証と JAZN Admintool（XML ベース・プロバイダのみ）」

クラスタ化操作

-clustersupport *oracle_home*

関連項目：

- B-5 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）」

構成操作

-getconfig

関連項目：

- B-9 ページの「構成操作」

対話型シェル

-shell

関連項目：

- B-15 ページの「JAZN Admintool シェルの使用」

ログイン・モジュール

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

関連項目：

- B-6 ページの「ログイン・モジュールの追加と削除（XML ベース・プロバイダのみ）」
- B-10 ページの「ログイン・モジュールのリスト表示」

移植操作

```
-convert filename realm
```

関連項目：

- B-14 ページの「principals.xml ファイルからのプリンシパルの移植」

パスワード管理

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

関連項目：

- B-9 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）」
- B-14 ページの「パスワードの設定（XML ベース・プロバイダのみ）」

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_class [permission_params]
-listperm permission
-listprncpls [principal_name]
-listprncpl principal_name
-remperm permission
-remprncpl principal_name
-revokeperm {realm {-user user|-role role} | principal_class principal_params}
            permission_class [permission_params]
```

関連項目：

- B-5 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）」
- B-7 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）」
- B-9 ページの「パーミッションの付与と取消し」
- B-11 ページの「パーミッションのリスト表示」
- B-11 ページの「パーミッション情報のリスト表示」
- B-12 ページの「プリンシパル・クラスのリスト表示」
- B-12 ページの「プリンシパル・クラス情報のリスト表示」

レルム・オプション

```
-addrealm realm admin {adminpwd adminrole | adminrole
    userbase rolebase realmtype }
-addrole realm role
-adduser realm username password
-grantrole role realm {user | -role to_role}
-listrealms realm
-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}
```

関連項目：

- B-7 ページの「レルムの追加と削除」
- B-8 ページの「ロールの追加と削除（XML ベース・プロバイダのみ）」
- B-8 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）」
- B-10 ページの「ロールの付与と取消し」
- B-12 ページの「レルムのリスト表示」
- B-13 ページの「ロールのリスト表示」
- B-13 ページの「ユーザーのリスト表示」

ヘルプ

```
-help  [command_name]
```

特定のコマンドのヘルプを表示します。

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 username -password password -listrealms
```

> **注意：** -user および -password オプションを使用する場合、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
```

注意：

- コマンドラインの -user および -password を使用してユーザー名とパスワードを指定することも可能ですが、この方法は使用しないことをお薦めします。コマンドラインでパスワードを指定すると、セキュリティ上の脆弱性が生じます。
- Java の制限事項により、Admintool に入力したパスワードが、コマンド・ウィンドウに表示される場合があります。Admintool の使用後には、コマンド・ウィンドウを必ず閉じてください。
- 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。ツールを実行するユーザーは、Admintool の操作を Oracle Internet Directory サーバーに対して実行できます。つまり、OC4J で LDAP ベース・プロバイダが使用される本番マシンへのアクセスを保護することが必要不可欠です。LDAP ベース・プロバイダの使用時には、-user および -password オプションを指定しても無視されます。

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
```

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*

このオプションを指定すると、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 *ORACLE_HOME* 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。

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（XML ベース・プロバイダのみ）

ログイン・モジュールの追加と削除には、JAZN Admintool を使用します。

```
java -jar jazn.jar -addloginmodule application_name login_module_name
     control_flag [optionname=value ...]
java -jar jazn.jar -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 B-1 を参照してください。

関連項目：

- JAZN Admintool の実行の基本情報は、4-3 ページの「Admintool の概要」を参照してください。

表 B-1 ログイン・モジュール制御フラグ

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）。
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
```

プリンシパルの追加と削除（XML ベース・プロバイダのみ）

-**addprncpl** *principal_name principal_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場合、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場合、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、external または application です。

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（XML ベース・プロバイダのみ）

-**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。」という応答が戻され、ユーザー名またはパスワード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*]

この構文の *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です。また、*principal_parameters* 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 の場合、realmname action）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 -user martha java.io.FilePermission
     a.txt read, write
```

Admintool のシェルの場合

```
JAZN:> grantperm foo -user 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*] [*login_module_class*]

ログイン・モジュールのリスト表示には、JAZN Admintool を使用します。

関連項目：

- JAZN Admintool の実行の基本情報は、4-3 ページの「Admintool の概要」を参照してください。

```
java -jar jazn.jar -listloginmodules [application_name [login_module_class]]
```

-listloginmodules オプションを使用して、指定の *application_name* 内のログイン・モジュール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ログイン・モジュールが表示されます。*application_name* の後に *login_module_class* を指定すると、アプリケーション内の指定したクラスに関する情報のみが表示されます。

たとえば、アプリケーション myapp のログイン・モジュール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
- principal 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 -user martha
```

Admintool のシェルの場合

```
JAZN:> listperms foo -user 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、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** *principal_name*

-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。*principal_name* 引数が存在する場合、指定されたプリンシパル・クラスのみがリスト表示されます。

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** [*realm*]

-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。引数を指定すると、指定されたレルムのみがリスト表示されます。

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

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。

```
JAZN:> listusers foo
```

principals.xml ファイルからのプリンシパルの移植

principals.xml ファイルからデータを移植するには、JAZN Admintool を使用します。JAZN Admintool の実行の基本情報は、4-3 ページの「Admintool の概要」を参照してください。

-convert *filename realm*

-convert オプションを使用して、principals.xml ファイルを現行の OracleAS 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。

移植により、principals.xml のユーザーが JAAS のユーザーに、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してください。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. 次の例のように入力して、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. <default-realm> を principals.com に変更します。8-4 ページの「永続性モードの設定」を参照してください。

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。次に例を示します。

```
java -jar jazn.jar -user 'Oracle DBA' ...
```

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、B-4 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 シェルのコマンド

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。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。この項では、コマンドの構文について説明します。

関連項目：

- Admintool ディレクトリ構造の詳細は、B-17 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。

Admintool のナビゲーション・コマンドは、add、cd、clear、exit、help、ls、man、pwd、rm および set 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 mydir にナビゲートするには、次のように入力します。

```
cd mydir
```

cd / とコマンドを入力すると、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*

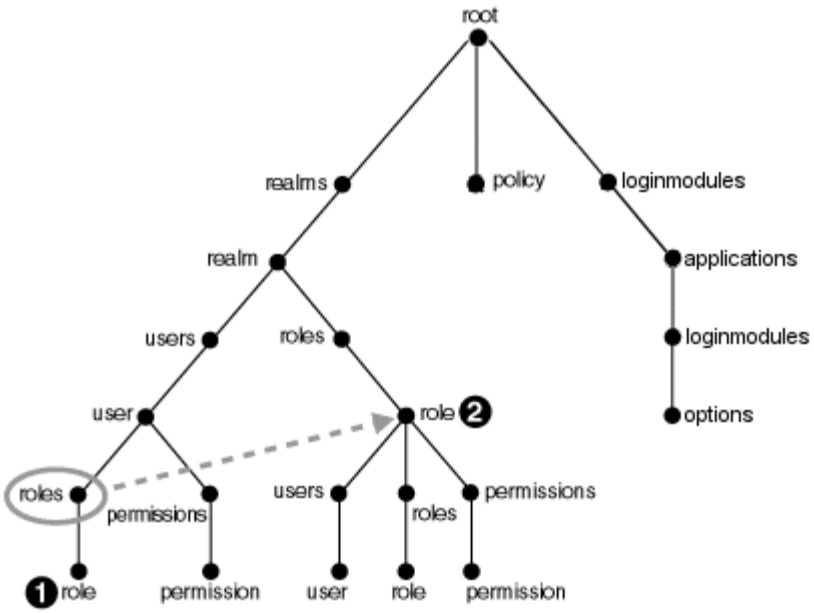
set コマンドを実行すると、指定した名前の値が更新されます。たとえば、このコマンドでは、作業ディレクトリに応じて、ログイン・モジュール・クラス、ログイン・モジュール制御フラグまたはログイン・モジュール・クラス・オプションが更新されます。

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 B-1 にノード構造を示します。

図 B-1 JAZN シェルのディレクトリ構造

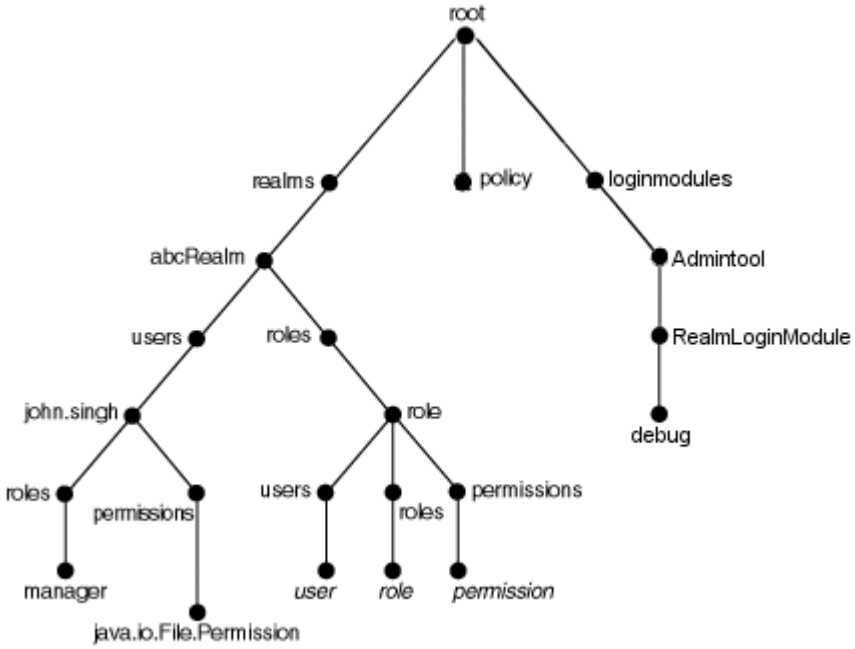


この構造では、user および role ノードがリンクされています。これは、user の下の roles リンクが、realm の下の roles リンクと同じであることを意味します。UNIX 用語では、この図で番号 1 の role は、この図で番号 2 の role へのシンボリック・リンクです。

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 B-2 に、A-2 ページの「サンプル : jazn-data.xml の構成」に示した jazn-data.xml ファイルにより作成される abcRealm のノードを示します。

図 B-2 シェルのディレクトリ構造



索引

記号

A

C

D

E

G

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

へ

ほ

ま

も

ゆ

り

れ

ろ