Oracle® Containers for J2EE

セキュリティ・ガイド

10*g*（10.1.3.4.0）

部品番号 : B50832-01

2008 年 9 月

ORACLE®

Oracle Containers for J2EE セキュリティ・ガイド , 10*g*（10.1.3.4.0）

部品番号 : B50832-01

Oracle Containers for J2EE Security Guide, 10*g* (10.1.3.4.0)

原本部品番号 : E12514-01

原本著者 : Brian Wright

原本協力者 : Elizabeth Hanes Perry, Joseph Ruzzi, Ganesh Kirti, Raymond Ng, Rachel Chan, Nithya Muralidharan, Kumar Valendhar, Moushmi Banerjee, Dheeraj Goswami, Sam Zhou, Srikant Tirumalai, Sirish Vepa, Vineet Garg, Bill Bathurst, Debu Panda, Steve Button, Tom Snyder, Jeff Trent, Bob Nettleton, Vinay Shukla, Alex Kosowski, Rajbir Chahal, Michael Hwa, Jayanthi Kulkarni, Kavita Tippana, Helen Zhao, Sandeep Bangera, Cania Lee Chung, Deepika Damojipurapu, Lakshmi Thiyagarajan, Soumya Aithal, Serouj Ourishian, Phil Varner, Chaya Ramanujam, Xiaopeng Wu, Jyotsna Laxminarayanan, Lelia Yin, Raghav Srinivisan, Sam Chou, Bhupindra Singh, Ashish Kolli, Frank Nimphius, Dan Hynes, Steve Button, Viresh Garg, Alfred Franci, Peter LaQuerre

目次

3 OC4J セキュリティの概要

4 セキュリティの管理の概要

5 OC4J での認可

6 OC4J セキュリティに関する一般的なタスク

7 ファイルベースのセキュリティ・プロバイダ

8 Oracle Identity Management

9 ログイン・モジュール

10 外部 LDAP セキュリティ・プロバイダ

11 Oracle Access Manager

12 ユーザーおよびロール API フレームワーク

13 交換可能なアイデンティティ管理フレームワーク

14 OC4J Java シングル・サインオン

15 OC4J との SSL 通信

16 クライアント接続用 Oracle セキュリティ

17 Web アプリケーションのセキュリティ構成

18 EJB のセキュリティの構成

19 Common Secure Interoperability プロトコル

20 リソース・アダプタのセキュリティ・サポート

21 Windows のネイティブ認証の構成

A OC4J セキュリティのヒントおよびトラブルシューティング

B OracleAS JAAS Provider のサンプル

C OracleAS JAAS Provider Admintool リファレンス

D OracleAS JAAS Provider 構成ファイル

E サード・パーティ・ライセンス

索引

例一覧

図一覧

表一覧

はじめに

このマニュアルでは、Oracle Containers for J2EE（OC4J）のセキュリティ機能について説明します。

この章の内容は次のとおりです。

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、Web アプリケーション、Enterprise JavaBeans（EJB）、J2EE Connector Architecture、Secure Sockets Layer および Common Secure Interoperability Version 2（CSIv2）プロトコルなど、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。

Oracle サポート・サービスへの TTY アクセス

アメリカ国内では、Oracle サポート・サービスへ 24 時間年中無休でテキスト電話（TTY）アクセスが提供されています。TTY サポートについては、(800)446-2398 にお電話ください。

関連ドキュメント

詳細は、次の Oracle ドキュメントを参照してください。

その他の OC4J ドキュメントには、次のものがあります。

- 『Oracle Containers for J2EE 開発者ガイド』

  サーブレット、EJB、JSP コンテナなど特定のコンテナに固有の（クラスのロードなどの）問題以外の、OC4J で稼働するアプリケーションを作成する開発者が一般に関心を持つ項目について説明します。

- 『Oracle Containers for J2EE デプロイメント・ガイド』

  OC4J 環境にアプリケーションをデプロイするための情報および手順を示します。Oracle Enterprise Manager 10*g* に付属のデプロイ・プラン・エディタについても説明します。

- 『Oracle Containers for J2EE 構成および管理ガイド』

  OC4J のアプリケーションの構成および管理方法について説明します。Oracle Enterprise Manager 10*g* Application Server Control コンソールの使用方法、OC4J に付属する標準準拠の Mbean の使用方法、また必要に応じて OC4J 固有の XML 構成ファイルを直接使用する方法についても説明します。

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

  OC4J に付属する標準ベースの Java サービス（JTA、JNDI、JMS、Oracle Application Server Java Object Cache、XML 問合せサービスなど）に関する情報について説明します。

- 『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』

  リソース・アダプタと J2EE Connector Architecture に関する情報について説明します。

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

  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を示します。基本的なサーブレット開発や、JDBC と EJB の使用方法についても説明します。

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

  JavaServer Pages の開発と JSP 実装、および OC4J のコンテナに関する情報を示します。コマンドライン・トランスレータや OC4J 固有の構成パラメータなどの Oracle の機能についても説明します。

- 『Oracle 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』

  OC4J に付属するタグ・ライブラリおよび JavaBeans の概念と、詳しい構文情報および使用方法を示します。

- 『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』

  Enterprise JavaBean（EJB）の開発と EJB 実装、および OC4J のコンテナに関する情報を示します。

- 『Oracle Application Server Web Services 開発者ガイド』

  OC4J および Oracle Application Server における Web サービスの開発および構成について説明します。

- 『Oracle Application Server Web Services アドバンスト開発者ガイド』

  高度な Web サービス・アセンブリについて説明します。たとえば、一般的な相互運用性の問題を診断する方法、Web サービス管理機能（信頼性、監査、ロギングなど）を有効にする方法、Java の値タイプのカスタム・シリアライズを使用する方法について説明します。

- 『Oracle Application Server Web Services セキュリティ・ガイド』

  OC4J および Oracle Application Server における Web サービスのセキュリティと構成について説明します。

関連する Javadoc のセット：

- 『Oracle Containers for J2EE Security Java API Reference』
  OracleAS JAAS Provider の API、アイデンティティ管理フレームワーク、Java SSO について説明します。
- 『Oracle Containers for J2EE User and Role Java API Reference』
  アイデンティティ管理リポジトリのユーザー情報とロール情報にアクセスする API について説明します。
- 『Oracle Application Server HTTPClient Java API Reference』
  Oracle `HTTPClient` パッケージの API について説明します。

Oracle Application Server コア・ドキュメント・グループには、次のドキュメントが含まれます。

- 『Oracle Application Server 管理者ガイド』
- 『Oracle Application Server エンタープライズ・デプロイメント・ガイド』
- 『Oracle HTTP Server 管理者ガイド』
- 『Oracle Process Manager and Notification Server 管理者ガイド』
- 『Oracle Application Server Certificate Authority 管理者ガイド』
- 『Oracle Application Server ベスト・プラクティス』

Oracle Identity Management、Oracle Internet Directory および Oracle Single Sign-On のドキュメントには、次のものがあります。

- 『Oracle Identity Management インフラストラクチャ管理者ガイド』
- 『Oracle Identity Management 統合ガイド』
- 『Oracle Identity Management 委任管理ガイド』
- 『Oracle Identity Management アプリケーション開発者ガイド』
- 『Oracle Internet Directory 管理者ガイド』
- 『Oracle Internet Directory API Reference』
- 『Oracle Application Server Single Sign-On 管理者ガイド』

Oracle Access Manager のドキュメントには、次のものがあります。

- 『Oracle Access Manager 概要』
- 『Oracle Access Manager インストレーション・ガイド』
- 『Oracle Access Manager System Administration Guide』
- 『Oracle Access Manager ID および共通管理ガイド』
- 『Oracle Access Manager 開発者ガイド』
- 『Oracle Access Manager デプロイメント・ガイド』

追加情報は、次の Web サイトから入手できます。

- Oracle ドキュメントの最上位レベルへのリンクは、次の Oracle Technology Network にあります。

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

- OC4J の様々な使用方法の例は、次の Web サイトを参照してください。

  `http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

- 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、操作に関連する Graphical User Interface 要素を示します。
イタリック	イタリックは、ユーザーが特定の値を指定するプレースホルダ変数を示します。
固定幅フォント	段落内の固定幅フォントは、コマンド、URL、Java のクラス名やメソッド名、ファイル名やディレクトリ名、画面に表示されるテキスト、または入力するテキストを示します。

サポートおよびサービス

次の各項に、各サービスに接続するための URL を記載します。

Oracle サポート・サービス

オラクル製品サポートの購入方法、および Oracle サポート・サービス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.com/lang/jp/support/index.html
```

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://www.oracle.com/technology/global/jp/documentation/index.html
```

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://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=3
```

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.com/lang/jp/index.html
http://www.oracle.com/technology/global/jp/index.html
```

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。

新機能

この項では、リリース 10.1.2.0.2 以降の新機能について説明します。

- リリース 10.1.3.4 での変更点
- リリース 10.1.3.1 での変更点
- リリース 10.1.3.0.0 での変更点

リリース 10.1.3.4 での変更点

OC4J 10.1.3.4 実装では、次のセキュリティ機能と拡張が追加されました。

重要な変更点と追加点

OC4J 10.1.3.4 実装での次の重要な追加点に注意してください。

- OC4J および Java SSO で、Windows のネイティブ認証および Kerberose がサポートされるようになりました。Windows デスクトップでのログインに基づいて、OC4J Web アプリケーションのユーザーを自動的に認証できます。詳細は、第 21 章「Windows のネイティブ認証の構成」を参照してください。
- HTTP クライアント API を使用してリクエスト・ヘッダーに Basic、Digest および NTLM の認証情報を含める方法の説明が追加されました。詳細は、16-2 ページの「非 SSL クライアント認証の使用方法」を参照してください。
- OC4J コンテナ内にサブジェクトを設定する方法の説明が追加されました。5-12 ページの「OC4J コンテナへのサブジェクトの設定」を参照してください。
- 10.1.3.1 リリース以降のドキュメント正誤表の項目は、Oracle Containers for J2EE セキュリティ・ガイドのこのリリースですでに修正されています。

更新された非推奨事項

OC4J 10.1.3.4 実装での次の非推奨事項に注意してください。

- Oracle Java SSL は非推奨であり、`HTTPClient` では使用されなくなりました。JDK のデフォルトの JSSE 実装が、`HTTPClient` のデフォルトの SSL 実装となりました。

リリース 10.1.3.1 での変更点

この項では、OC4J 10.1.3.1 実装での変更点と更新された非推奨事項を示します。この後の「リリース 10.1.3.0.0 での変更点」も確認してください。

重要な変更点と追加点

OC4J 10.1.3.1 実装での次の重要な追加点に注意してください。

- サード・パーティ提供の異種アイデンティティ管理システムをサポートするアイデンティティ管理フレームワーク
- Java SSO（OC4J 付属の代替シングル・サインオン・ソリューション。Oracle Single Sign-On や Oracle Access Manager シングル・サインオンと同じように追加のインフラストラクチャを必要とせず、使用するアイデンティティ管理システムから OC4J を切り離す。）
- `DataSourceUserManager` 機能にかわる `DBTableOraDataSourceLoginModule`
- 新しいユーザーおよびロール API フレームワーク（特に、サポートされている LDAP サーバーで使用され、非推奨の `UserManager`、`User`、`Group` の各クラスにかわる機能が含まれる）

また、次の変更点にも注意してください。

- JDK のデフォルトの JSSE 実装が、`HTTPClient` のデフォルトの SSL 実装となりました。（これにより、これまで `HTTPClient` に対するデフォルトの SSL 実装であった OralceSSL は非推奨となります。）

更新された非推奨事項

OC4J 10.1.3.1 実装での次の非推奨事項に注意してください。

- `com.evermind.security` パッケージとそのクラスは非推奨になりました。**リリース 11*g* では、これらがサポートされなくなります。**
  - `UserManager` クラス : カスタム `com.evermind.security.UserManager` 実装のかわりに、JAAS カスタム・ログイン・モジュールを使用してください。
  - `User` クラス : かわりに標準の JAAS API を使用してください。
  - `Group` クラス : かわりに標準の JAAS API を使用してください。
- `XMLUserManager` クラスおよびそのデータ・ストアである `principals.xml` は、非推奨になりました。**リリース 11*g* では、これらがサポートされなくなります。**移行手順は、7-17 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。

リリース 10.1.3.0.0 での変更点

OC4J 10.1.3.0.0 実装では、次のセキュリティ機能と拡張が追加されました。

- COREid Access（現在の Oracle Access Manager）セキュリティ・プロバイダのサポート
- スタンドアロン OC4J での LDAP ベース・プロバイダのサポート
- Digest 認証のサポート、およびクライアント証明の認証および認可のサポート
- Java Authorization Contract for Containers（JSR-115）の実装
- JAAS の EJB との統合
- SSL のための ORMI 拡張（ORMIS）
- サブジェクト伝播（および ORMI または ORMIS）のサポート
- セキュリティ構成のための JMX および MBean のサポート（JSR-77）
- 新しい OC4J ユーザーおよびロールのアカウント（後述を参照）
- 拡張された Java 2 セキュリティのサポート
- Web Services Security（別のマニュアルに記載）

さらに、OC4J 10.1.2 実装以降には、次のような変更があります。

- 認可用として新しく統合された JAAS モードがあります（サーブレット用と EJB 用の両方）。このモードは、以前の機能である、サーブレット用の `runas-mode` および `dosasprivileged-mode` 機能、ならびに EJB 用の `USE_JAAS` 機能（10.1.3 より前のリリースで導入）に取ってかわるものです。以前の機能もサポートされていますが、OC4J 10.1.3.x 実装では非推奨になりました。
- 以前の各リリースで、ユーザーおよびロール構成（ファイルベース・プロバイダの場合）、ポリシー構成（ファイルベース、外部 LDAP またはカスタム・セキュリティ・プロバイダの場合）およびログイン・モジュール構成（すべてのセキュリティ・プロバイダの場合）を格納するために使用されたインスタンスレベルの `jazn-data.xml` 構成ファイルは、`system-jazn-data.xml` に名前が変更されました。ただし、アプリケーションでは、ファイルベース・プロバイダ用にユーザーおよびロール構成を格納するために、必要に応じてアプリケーション固有の `jazn-data.xml` リポジトリ・ファイルを使用できます。
- `XMLUserManager` クラスおよびこのクラスのデータ・ストアである `principals.xml` は非推奨になり、将来のリリースではサポートされなくなります。このため、既存のアプリケーションを移行することをお薦めします。移行手順は、7-17 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。
- 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。カスタム `com.evermind.security.UserManager` 実装のかわりに、JAAS カスタム・ログイン・モジュールを使用してください。

- Oracle Identity Management セキュリティ・プロバイダでは、アプリケーション・レルムおよび外部レルムは非推奨です。
- `external.synchronization` プロパティはサポートされなくなりました。
- `jaas.username.simple` プロパティのデフォルト設定は `true` になりました。OC4J 10.1.2 実装では、デフォルト設定は `false` でした。これにより、`getUserPrincipal()` や `getRemoteUser()`（サーブレットの場合）、`getCallerPrincipal()`（EJB の場合）などのメソッドから戻される認証済プリンシパルの名前において、レルム名がデフォルトで省略されるようになりました。
- OC4J アカウント名にいくつかの変更があります。`admin` アカウントは `oc4jadmin` に、`administrators` ロールは `oc4j-administrators` に、`jmx-users` ロールは `oc4j-app-administrators` になりました。スタンドアロン OC4J のファイルベース・プロバイダの場合、`oc4jadmin` は最初は非アクティブです。4-12 ページの「事前定義アカウント」を参照してください。
- 必要な OC4J アカウントは、OC4J インスタンスを OID インスタンスに関連付けるときに自動的に Oracle Internet Directory に作成されます。8-9 ページの「Oracle Internet Directory に作成される必須アカウント」を参照してください。
- OC4J 10.1.3.x 実装からは、`LD_LIBRARY_PATH` を設定する必要がなくなりました。
- ロギング用の `jazn.debug.log.enable` フラグがサポートされなくなりました。通常の OC4J ロギング機能を使用してください。A-5 ページの「ロギング」を参照してください。

1

基本的なセキュリティの概要

この章では、次の項目に沿ってセキュリティの概要について説明します。

- アプリケーション・レベルのセキュリティ
- トランスポート・レベルのセキュリティ

この 2 つはセキュリティの基本的なカテゴリであり、個別に構成することができますが、多くの場合は相互に関連しています。通常、アプリケーション・レベルのセキュリティによって、データにアクセスできるユーザーおよびそのユーザーが実行を許可されているタスクが決まります。トランスポート・レベルのセキュリティによって、データが転送されたときにデータのセキュリティが決まります。

アプリケーション・レベルの構成にはトランスポート・レベルの指定を含めることができます。たとえば、トランスポート・レベルの機能である Secure Sockets Layer（後述）を必要とするアプリケーション・レベルの制約を指定することができます。トランスポート・レベルのセキュリティには、データへのアクセスを適切なユーザーに限定する認証を伴うこともあります。たとえば、クライアント証明書がトランスポート・レベル機能の一部として要求されることがあります。

アプリケーション・レベルのセキュリティ

アプリケーション・レベルのセキュリティによって、アプリケーションまたはそのデータにアクセスできるユーザー、およびそのユーザーが実行できるタスクが決まります。ここでは、次のような主な機能について説明します。

- 認証の概要
- 認可の概要

認証の概要

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アイデンティティ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。

ユーザー名やパスワードなどの認証情報は、XML ファイル、データベース、ディレクトリ・サービスなどのユーザー・リポジトリに格納されます。サブジェクトが、ログインなどによって J2EE アプリケーションへのアクセスを試行すると、セキュリティ・プロバイダにより、ユーザー・リポジトリ内でサブジェクトが検索され、サブジェクトのアイデンティティが検証されます。セキュリティ・プロバイダは、認証や認可など、個別のセキュリティ・サービスの実装を提供するモジュールです。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にアクセス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アイデンティティを認証するのはセキュリティ・プロバイダです。

認可の概要

認可は、「どのコンポーネントが提供するどのリソースに誰がタスクを実行できるか」という質問に対処します。J2EE アプリケーションでは、リソースは通常、Web アプリケーションの場合は URL パターンに関連して、EJB の場合はメソッド呼出しの権限に関連して、表現されます。認可はロール単位ベースで行われ、アプリケーションに定義されているロールそれぞれに適切なパーミッションが割り当てられます。

ここでは、認可のタイプ、つまりアクセス制御および関連項目について説明します。

- アクセス制御リストおよびアクセス制御の機能モデル
- ロールベースのアクセス制御

機能モデルとロールベース・モデルは相互に補い合い、多くの場合、一緒に使用されます。

アクセス制御リストおよびアクセス制御の機能モデル

機能モデルは、認可情報を編成する手段です。Java 2 セキュリティ・モデルでは、機能モデルを使用してアクセス権を制御します。機能モデルを使用すると、アクセス制御情報がリソースに関連付けられ、認可が、Amy という名前のユーザーなど、エンティティに関連付けられます（エンティティは、2-13 ページの「プリンシパルとサブジェクト」で定義するように、プリンシパルと呼ばれています）。

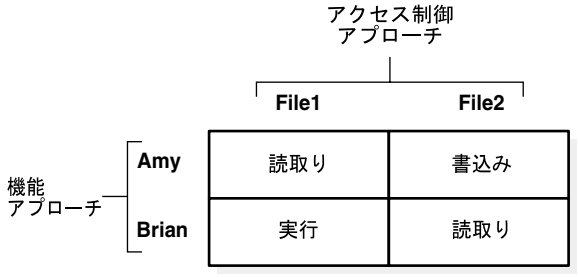
アクセス制御リスト（ACL）は、ディレクトリやファイルなど、保護されたターゲット・リソースに関連付けられます。このリストには、各ユーザーが特定のリソースに対して持っているアクセス権に関する情報が格納されています。たとえば、ファイル・システムの各ファイルが ACL を保持する場合があります。ACL に指定された機能（権限）は特定のユーザーに関連付けられるため、ACL が関連付けられているファイルに対してどのユーザーがどの権限を持っているかが指定されます。

ユーザー Amy がログインし、正常に認証された場合、Amy のパーミッションが取得され、付与され、Amy はパーミッションで許可されるアクションを自由に実行できます。`File1` からの読取りや `File2` への書込みなどです。

機能モデルとアクセス制御は、同じ情報を異なる視点から見たものです。機能はリソースにアクセスを試行するユーザーに関連付けられますが、アクセス制御リストはユーザーがアクセスを試行するリソースに関連付けられます。

図 1-1 は、File1 を読み取り、File2 に書き込むパーミッションを持つユーザー Amy、および File1 を実行し、File2 を読み取るパーミッションを持つユーザー Brian を示しています。アクセス制御リストは File1 と File2 という視点から作成され、各ファイルに対してどのユーザーがアクセス権を持っているか、ユーザーが持っているパーミッションは何であるかを指定します。機能モデルは Amy と Brian という視点から作成され、ユーザーがアクセスできるのはどのファイルか、各ファイルに対するパーミッションの種類は何かをユーザーごとに指定します。

図 1-1 アクセス制御アプローチと機能アプローチの比較



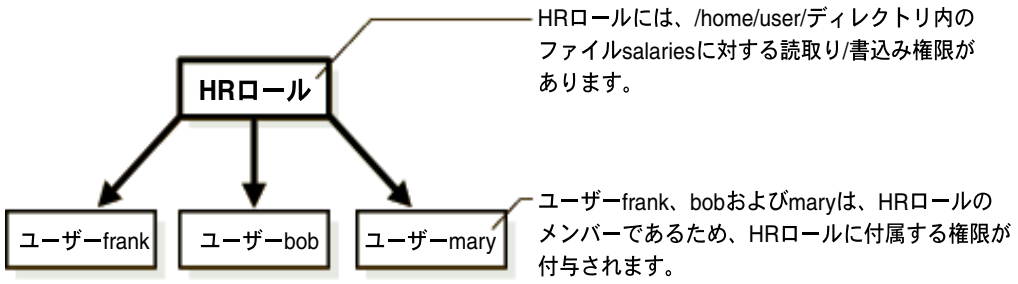
ロールベースのアクセス制御

基本的に、ロールとは認可レベルを定義する職務または役職のことです。1 つのロールに複数のユーザーおよび複数のパーミッションを持たせることができます。ロールは、各アプリケーションで様々なオブジェクトおよび機能へのアクセス権を示すために使用されるアイデンティティです。ユーザーは、これらの適切なリソース・セットへのアクセス権を得るため、ロールを取得します。

ロールベースのアクセス制御は JAAS の機能であり、これにより、ユーザーにパーミッションを直接割り当てることで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。図 1-2 に、ロールベースのアクセス制御の例を示します。

図 1-2 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。

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。

関連項目：

- 3-8 ページの「セキュリティ・ロール・マッピングの概要」

トランスポート・レベルのセキュリティ

これまでに説明した認証と認可の機能からは独立した、データを転送するときに保護する機能です。ここでは、ネットワークやインターネットを経由して転送されるデータが第三者によってインターセプト、読取り、または変更ができないようにする機能の概要を説明します。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。

この項の内容は次のとおりです。

- Secure Sockets Layer および HTTPS
- SSL 認証
- X.509 証明書
- 鍵の暗号化および交換

Secure Sockets Layer および HTTPS

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、Oracle HTTP Server と OC4J プロセス間の Apache JServ Protocol リンクに使用される場合に最も重要です。

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。

サーバーが SSL 通信を構成するかどうかは、クライアントが SSL 通信を構成するかどうかと無関係です。このドキュメントでは、第 15 章「OC4J との SSL 通信」で OC4J エンドでの SSL の有効化、および Oracle Application Server 環境における Oracle HTTP Server と OC4J 間の SSL 通信について説明します。第 16 章「クライアント接続用 Oracle セキュリティ」では、SSL 機能をクライアント HTTPS 接続に提供する HTTPS の OC4J 実装について説明します。

SSL 認証

SSL 通信を使用した場合、次のいずれの認証シナリオも可能です。

- SSL 認証を使用しない（または null 認証）: サーバーは証明書を送信せず、クライアントからの証明書をリクエストしません。SSL という点から見ると、サーバーはリモート・クライアントが誰であるかを知らず、クライアントによる提示が可能な証明書を受け付けません。
- 一方向 SSL 認証 : サーバーまたはクライアントの片方が証明書を必要とします。たとえば、サーバー認証は、サーバーが証明書をクライアントに送信しますが、クライアントからの証明書はリクエストしない一方向認証です。または、サーバーは証明書を必要とすることがありますが、証明書を送信することはなく、クライアントは証明書を必要としません。
- 双方向 SSL 認証 : これは、クライアントとサーバー認証です。この場合、サーバーはクライアントが要求する証明書を送信し、クライアントにも証明書を送信するよう要求します。

サーバーで SSL 認証を構成することは、クライアントで SSL 認証を構成することと無関係です。

X.509 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（CA）、法人または個人のいずれかです。

鍵の暗号化および交換

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。

通常、SSL では非対称型暗号化（公開鍵と秘密鍵）および対称型暗号化を組み合せて通信を保護します。公開鍵の交換は、通信に関わる当事者の相互認証に使用されます。これにより、通信の当事者は、セッションでこれ以降のデータの暗号化と復号化に使用する対称鍵の作成時に安全に協働できます。クライアントとサーバー間で SSL セッションを作成する基本的な例を次に示します。

1. クライアントは暗号スイート、圧縮方法、最上位のプロトコル・バージョンおよびランダムなバイト列をサーバーに送信します。サーバーはクライアントが提供する選択肢から接続パラメータを選択します。
2. 公開鍵（X.509 証明書）が交換されます。
   a. サーバーが自分の公開鍵をクライアントに送信し、クライアントは自分の公開鍵をサーバーに送信します。
   b. 鍵は、双方が他方の証明書を検証する場合の相互認証に使用されます。
3. 対称鍵が交換されます。この手順で、通信は交換された公開鍵を使用することにより保護されます。
   a. サーバーとクライアント間で、マスタ・シークレットが共同で生成されます。
   b. 次に、セッション鍵（バルク暗号鍵）はマスタ・シークレットに基づいて生成されます（128 ビットの RC4 鍵など）。
   c. クライアントとサーバーはそれぞれメッセージを送信します。以降の通信で、メッセージにセッション鍵が使用されます。
4. SSL トラフィックでは、暗号化と復号化に対称鍵が使用されます。

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、受信側プロセスでは、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、署名者の署名者が承認済リストに含まれている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、信頼できる証明書とみなされます。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。

2

Java プラットフォームのセキュリティ

この章では、Java および J2EE のアプリケーションとともに使用できる標準的なセキュリティ・モデルの概要について説明します。この章の内容は次のとおりです。

- J2EE セキュリティ・モデル
- Java 2 セキュリティ・モデル
- Java Authentication and Authorization Service
- 開発時におけるセキュリティの考慮事項

注意： J2EE、Java 2 および JAAS のセキュリティ・モデルは、相互に独立しています。単独でも組み合せても使用できます。方法は、5-22 ページの「認可の方法」を参照してください。

関連項目：

- セキュリティのベスト・プラクティスの詳細は、『Oracle Application Server ベスト・プラクティス』を参照してください（リリース後に入手可能）。

J2EE セキュリティ・モデル

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、コンテナ管理セキュリティ用の宣言による許可モデルを定義します。認可ポリシー（リソースとユーザーまたはロール間の関連付け）は、アプリケーション・コード内ではなく、アプリケーションのデプロイメント・ディスクリプタ内で静的に表現されます。認可はロールベースであるため、アクセスレベルで付与され、通常、Web の URL や EJB メソッドなどのリソースを保護します。

この項では、次の J2EE セキュリティ機能について説明します。

- Web アプリケーションの認証と認可
- Enterprise JavaBean の認証と認可
- アイデンティティの伝播

Web アプリケーションの認証と認可

この項では、Web アプリケーションのセキュリティ（主に宣言によるセキュリティ構成）について説明します。より高度なプログラム機能を実行するため、API についても説明します。API を使用すると、セキュリティ機能を実行時に決定できます。

内容は次のとおりです。

- Web アプリケーションの標準認証方式
- Web アプリケーション URL ベースの認可
- Web アプリケーションにおける run-as モードと伝播されたアイデンティティ
- 関連する Web アプリケーション API

Web アプリケーションの標準認証方式

いくつかの標準認証方式を使用して、J2EE Web アプリケーションにアクセスできます。

- **Basic**

  Basic 認証では、シングル・サインオンの実装を介さずに、ユーザー名とパスワードがユーザーに対して直接要求されます。

- **Digest**

  Digest 認証メカニズムでは、クライアントが自己認証を行うために示すパスワードが、MD5 ダイジェストを使用して暗号化されます。これは、リクエスト・メッセージで送信されます。ユーザーからは、Digest 認証は Basic 認証と同じように動作するように見えます。（OC4J では、Digest 方式は、外部 LDAP プロバイダやカスタム・プロバイダの場合はサポートされません。）

- **フォーム**

  ユーザーがフォームベース認証を使用して保護リソースにアクセスしようとすると、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。

- **CLIENT-CERT**

  この方式は Secure Sockets Layer（SSL）とともに使用され、HTTPS を介してクライアントが認証されます。ユーザーは公開鍵証明を持っている必要があります。

注意：

- OC4J は、3-6 ページの「Oracle Application Server シングル・サインオン代替方法の概要」で説明しているように、いくつかの Oracle 固有シングル・サインオン認証方式もサポートしています。
- ファイルベース・プロバイダまたは Oracle Identity Management でシングル・サインオンを使用する場合は、Basic 認証よりもセキュアなソリューションとして、Digest 認証をお薦めします。

関連項目：

- 17-2 ページの「認証方式（auth-method）の指定」

Web アプリケーション URL ベースの認可

J2EE セキュリティ・モデルの場合、保護する Web リソースは URL パターンによって識別されます。これは、Web アプリケーションの `web.xml` ファイルで指定されます。次に示すのは、アプリケーションの URL パターン `/resource` でリソースを保護する構成の抜粋です。

```
<web-resource-collection>
  <web-resource-name>resource access</web-resource-name>
  <url-pattern>/resource</url-pattern>
</web-resource-collection>
```

これは、`web.xml` のセキュリティ制約の一部です。これにより、リソースへのアクセスを許可されている J2EE 論理ロールも指定されます。J2EE 仕様に記載されている J2EE 論理ロールには、開発者（アプリケーション・コンポーネント・プロバイダ）、アセンブラ、デプロイヤ、システム管理者などがあります。

たとえば、J2EE ロール `sr_developers` が `web.xml` ファイルで宣言されているとします。このロールによるリソースへのアクセスを許可するセキュリティ制約は、次のようになります。

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

これにより、この後の OC4J 固有の構成手順で、`sr_developers` ロールを適切なデプロイ・ロール（セキュリティ・プロバイダに定義されているロール）にマップすることができます。

関連項目：

- 3-8 ページの「セキュリティ・ロール・マッピングの概要」
- 詳細は、17-7 ページの「Web アプリケーションのセキュリティ・ロールおよび制約の構成」を参照してください。

Web アプリケーションにおける run-as モードと伝播されたアイデンティティ

Web アプリケーションから EJB へのコールの場合、デフォルト・モードとして、Web クライアントのセキュリティ・アイデンティティが EJB コンテナに伝播されます。

状況によっては、Web コンテナや EJB コンテナに認識されていない Web クライアントによるコールを、Web コンテナが許可しなければならない場合があります。これには、インターネットからリソースへのアクセスを許可する場合など、コンテナに認証されていない Web クライアントをサポートするシナリオも含まれます。

このような状況の場合、<servlet> 要素の <run-as> サブ要素を使用して、Web アプリケーションの web.xml ファイルに対して、みなし実行識別性を指定することができます。

```
<run-as>
   <role-name>sr_developers</role-name>
</run-as>
```

Web コンテナは、コールのセキュリティ・アイデンティティをサーブレットから EJB レイヤーに伝播します。このとき、<run-as> 要素に指定されたロールが使用されます。このロールは、<security-role> 要素を使用して事前に宣言されている必要があります。伝播されたアイデンティティは、みなし実行識別性によって非表示になります。

関連項目：

- 2-7 ページの「アイデンティティの伝播」
- 17-10 ページの「Web アプリケーションに対する run-as セキュリティ・アイデンティティの指定」

関連する Web アプリケーション API

高度な使用方法として、標準的な J2EE プログラム機能があります。この機能により、Web アプリケーションを使用してユーザーに関する情報を取得できます。次のメソッドは、ユーザーによるリソースへのアクセスが許可されるかどうかを判断する際に使用できます。

Web アプリケーションで使用できるのは、javax.servlet.http.HttpServletRequest インスタンスに含まれる次のメソッドです。

- `Principal getUserPrincipal()`

  リクエストを作成する認証対象ユーザーの名前が含まれるプリンシパル・オブジェクト（ユーザーが認証されていない場合は null）を返します。

  サーブレットと EJB 間のアイデンティティ伝播が使用されている場合、コール元サーブレットの getUserPrincipal() メソッドによって返されるプリンシパル名は、EJB の getCallerPrincipal() メソッドによって返されるものと同じです。

- `String getRemoteUser()`

  リクエストを送信した認証対象ユーザーのログイン名（ユーザーが認証されない場合は null）を戻します。

- `boolean isUserInRole(String rolename)`

  リクエストを送信した認証対象ユーザーが指定されたロールのメンバーであるかどうかを判別します。

注意：

- ここで説明する API は、ユーザー・リポジトリとしてファイルベース・プロバイダまたは Oracle Internet Directory と併用できます。
- 以前のリリースの com.evermind.security.User クラスと Group クラスは非推奨です。（これらのクラスは、リリース 11*g* ではサポートされない予定です。）かわりに、getUserPrincipal() などの標準的な JAAS API と署名を使用し、必要に応じて伝播されたサブジェクトを利用してください。

関連項目：

- プリンシパルの概要は、2-13 ページの「プリンシパルとサブジェクト」を参照してください。
- 『Oracle Containers for J2EE Security Java API Reference』（Javadoc）

Enterprise JavaBean の認証と認可

この項では、EJB のセキュリティ（主に宣言によるセキュリティ構成）について説明します。より高度なプログラム機能を実行するため、API についても説明します。API を使用すると、セキュリティ機能を実行時に決定できます。

内容は次のとおりです。

- EJB 認証
- EJB メソッドベースの認可
- EJB アプリケーションにおける run-as モードと伝播されたアイデンティティ
- 関連する EJB API

EJB 認証

リモート・コンテナ内の EJB にアクセスする場合は、EJB にアクセスしているクライアントの認証が必要です。

- スタンドアロンの Java クライアントは、`jndi.properties` ファイルのユーザー設定とパスワード設定を使用して資格証明を渡すことができます。
- EJB クライアントまたは Web クライアントは、`javax.naming.InitialContext` インスタンスを使用して資格証明を渡すことができます。このインスタンスは、リモート EJB を検索する際に作成されます。

また、ORMIS が使用されている場合（ORMI は Secure Sockets Layer とともに使用）、CLIENT-CERT 認証と EJB を併用できます。

関連項目：

- 18-9 ページの「EJB クライアントでの資格証明の指定」
- 第 15 章「OC4J との SSL 通信」（特に、15-15 ページの「クライアント認証の要求」および 15-18 ページの「OC4J での ORMIS の有効化」）

EJB メソッドベースの認可

J2EE セキュリティ・モデルの場合、保護する EJB リソースは、そのメソッド名または特定の EJB 内にある名前マスクによって識別されます。これは、EJB の `ejb-jar.xml` ファイルで指定されます。次の例は、`PurchaseOrder` Bean のすべてのメソッドを保護する構成を抜粋したものです。

```
<method>
 <ejb-name>PurchaseOrder</ejb-name>
 <method-name>*</method-name>
</method>
```

これは、`ejb-jar.xml` 内にあるメソッド・パーミッションの一部です。これにより、リソースへのアクセスを許可されている J2EE 論理ロールも指定されます。J2EE 仕様に記載されている J2EE 論理ロールには、開発者（アプリケーション・コンポーネント・プロバイダ）、アセンブラ、デプロイヤ、システム管理者などがあります。

次の例では、ロール `myMgr` による `PurchaseOrder` EJB の任意のメソッドへのアクセスが許可されます。

```
<method-permission>
 <role-name>myMgr</role-name>
 <method>
  <ejb-name>PurchaseOrder</ejb-name>
  <method-name>*</method-name>
 </method>
</method-permission>
```

これにより、この後の OC4J 固有構成手順で、`myMgr` ロールを適切なデプロイ・ロール（セキュリティ・プロバイダに定義されているロール）にマップすることができます。

関連項目：

- 3-8 ページの「セキュリティ・ロール・マッピングの概要」
- 詳細は、18-3 ページの「EJB デプロイメント・ディスクリプタでの J2EE ロールおよびメソッド・パーミッションの指定」を参照してください。

EJB アプリケーションにおける run-as モードと伝播されたアイデンティティ

`ejb-jar.xml` デプロイメント・ディスクリプタには、EJB に対するみなし識別性指定を指定できます。これは、ある EJB が別の EJB をコールする際に、アイデンティティ伝播と連携して使用できます。みなし識別性は、伝播されるアイデンティティを非表示にし、コールされた EJB にまとめて適用され、メソッドを実行する場合のアイデンティティとして使用されます。このアイデンティティは、`<security-role>` 要素を使用して事前に宣言されている J2EE 論理ロールになります。

この場合、`<security-identity>` 要素の `<run-as>` サブ要素を次のように使用します。

```
<run-as>
   <role-name>admin</role-name>
</run-as>
```

関連項目：

- 2-7 ページの「アイデンティティの伝播」
- 18-6 ページの「EJB の run-as セキュリティ・アイデンティティまたはコール元セキュリティ・アイデンティティの指定」

関連する EJB API

高度な使用方法として、標準的な J2EE プログラム機能があります。この機能により、EJB を使用してユーザーに関する情報を取得できます。次のメソッドは、コール元によるリソースへのアクセスが許可されるかどうかを判断する際に使用できます。

EJB アプリケーションで使用できるのは、`javax.ejb.EJBContext` インスタンスに含まれる次のメソッドです。

- `Principal getCallerPrincipal()`

  コール元を識別するプリンシパル・オブジェクトを戻します。

  サーブレットと EJB 間のアイデンティティ伝播が使用されている場合、コール元サーブレットの `getUserPrincipal()` メソッドによって返されるプリンシパル名は、EJB の `getCallerPrincipal()` メソッドによって返されるものと同じです。

  EJB 間のアイデンティティ伝播が使用されている場合、クライアントのアイデンティティは、コール連鎖の上位から下位に向かってすべての EJB に伝播され、`getCallerPrincipal()` によって返されるプリンシパル名は、コール連鎖のすべての EJB に対するプリンシパル名と同じものになります。ただしこれは、`subject.propagation` パーミッション（18-14 ページの「サブジェクト伝播のための RMI パーミッションの付与」を参照）がすべての認証ユーザーに対して順に付与されていることを前提としています。

- `boolean isCallerInRole(String rolename)`

  コール元が指定されたロールのメンバーであるかどうかを判別します。

注意：

- ここで説明する API は、ユーザー・リポジトリとしてファイルベース・プロバイダまたは Oracle Internet Directory と併用できます。
- 以前のリリースの `com.evermind.security.User` クラスは非推奨です。（リリース 11*g* ではサポートされない予定です。）かわりに JAAS API を使用し、必要に応じて伝播されたサブジェクトを利用してください。

関連項目：

- プリンシパルの概要は、2-13 ページの「プリンシパルとサブジェクト」を参照してください。

アイデンティティの伝播

J2EE におけるアイデンティティ伝播とは、オリジナルの Web モジュールまたは EJB によって起動される EJB に対して、Web モジュールまたは EJB からセキュリティ・アイデンティティを転送することです。

1. 開始アプリケーション・クライアントまたは Web クライアントは、セキュリティ・アイデンティティを使用して中間の EJB または Web モジュールにアクセスします。
2. 中間の EJB または Web モジュールは、セキュリティ・アイデンティティの転送や伝播を行ってターゲット EJB にアクセスし、ターゲット EJB を起動します。

通常、伝播用のモデルには次の 2 つがあります。

- 中間コンテナがターゲット・コンテナによって信頼されている場合、中間の EJB または Web モジュールのコール元アイデンティティをターゲット EJB に伝播できます。
- 特定のアイデンティティによるターゲット・コンテナへのアクセスが予想される場合、このアイデンティティをターゲット EJB に伝播できます。

通常、中間コンテナはターゲット・コンテナによって信頼されている必要があります。これは、伝播されたアイデンティティを認証する際に、ターゲット・コンテナはデータを使用できないためです。伝播されたアイデンティティは `isCallerInRole()` などの認可チェックに使用されるため、信頼されたアイデンティティである必要があります。

OC4J の場合、アイデンティティ伝播はサブジェクト伝播と呼ばれます。

関連項目：

- 18-12 ページの「ORMI 用のサブジェクト伝播の有効化と構成」
- 詳細は、次のチュートリアルを参照してください。

  ```
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
  Security11.html
  ```

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 のセキュリティの一部としてサポートされています。ただし、OC4J 自体ではなく、基礎となる JDK に実装されています。

このセキュリティ・モデルにより、開発者や管理者にとっては、エンタープライズ・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・モデルは機能ベースであり、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（2-11 ページの「Java 2 認可 : Java 2 セキュリティ・ポリシー」を参照）を設定できます。

ただし、Java 2 セキュリティ・モデル自体にいくつかの制限があります。デプロイメント・ディスクリプタにおいては宣言的であるのとは異なり、Java2 セキュリティ・モデルはアプリケーション・コードに基づいています。また、ポリシー管理 API も含まず、スケーラビリティが高くないファイルベース実装を使用します。

ここでは、Java 2 セキュリティ・モデルの特徴と機能について説明します。

- コードベースのセキュリティ
- セキュリティ・パーミッション
- 保護ドメイン
- Java 2 認可 : Java 2 セキュリティ・ポリシー
- Java 2 認可 : セキュリティ・マネージャおよびアクセス・コントローラ

関連項目：

- 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/security/
  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/javase/technologies/security.jsp
  ```

コードベースのセキュリティ

コードを実行する一連のパーミッションを適用することにより、コードベースのセキュリティは、アプリケーションが実行できる操作を制限します。これにより、信頼されないコードや悪意あるコードのアクションからの保護を行い、次のような制限を実行できます。

- データベースへのアクセスを、信頼されたアプリケーションのみに制限する。
- インターネットからダウンロードされたコードの実行を禁止する。
- サード・パーティのコードによって実行できる操作を制限する。

コードベースのセキュリティは、ユーザーやユーザー・ロールに基づいていないという点で、ロールベースのセキュリティとは異なります。パーミッションは、コードの取得元やデジタル署名の有無（および署名者）など、コードの特性に基づいて付与されます。

コードベースとは、次の例のようにコードの場所を示す URL のことです。

- `file:`（ローカル・ファイル・システム上の任意のファイル）
- `http://*.oracle.com`（oracle.com の任意のホスト上の任意のファイル）
- `file:${j2ee.home}/lib/oc4j-internal.jar`

コードソースは、コードベースの概念を拡張するものです。その場所で生成された署名付きコードを検証する証明書の配列（Java キーストアに格納）をオプションで含むことができます。コードソースは、`java.security.CodeSource` インスタンスによって表され、`java.net.URL` インスタンスと `java.security.cert.Certificate` インスタンスの配列を指定することにより作成されます。

標準的な CodeSource クラスには、次のメソッドが含まれています。

- `Certificate[] getCertificates()`

  コードソースに関連付けられた証明書の配列を返します。

- `URL getLocation()`

  このコードソースに関連付けられた URL の場所を返します。

- `boolean equals(Object)`

  指定されたオブジェクト（通常はコードソース・オブジェクト）とこのコードソース・オブジェクトが一致するかどうかを比較します。2 つのコードソースの場所が同じで、一連の証明書が同じである場合、これらのコードソースは等しいとみなされます。その際、証明書の順序は同じである必要はありません。

- `boolean implies(CodeSource)`

  一連の比較を実行し、このコードソースが指定されたコードソースを暗黙的に定義しているかどうかを確認します。たとえば、次に示す場所のコードソースがあり、null 証明書が指定されている場合、コードソースの場所は `http://java.sun.com/classes/foo.jar` として暗黙的に定義され、null 証明書が指定されます。

  ```
  http:
  http://*.sun.com/classes/*
  http://java.sun.com/classes/-
  http://java.sun.com/classes/foo.jar
  ```

  関連項目：

  - CodeSource クラスやその他の標準的なクラスの詳細は、次の URL を参照してください。

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

セキュリティ・パーミッション

パーミッションは、Java 2 セキュリティ・モデルの基礎となります。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。

`java.security.Permission` クラスは、指定されたリソースへのアクセス権、およびこのリソースに対する指定アクションへのアクセス権（オプション）を表す抽象クラスです。このクラスの主要メソッドは `implies(Permission permission)` です。このメソッドでは、指定されたパーミッションのアクションが、このメソッドのコールに使用されたパーミッション・インスタンスのアクションで暗黙的に定義されているかどうかをチェックします。

一般的なパーミッション・タイプを表すクラスは次のとおりです（いずれも Permission を直接的または間接的に拡張したものです）。

- `java.security.AllPermission`
- `java.lang.RuntimePermission`（リソース・ターゲットのみを含む）
- `java.io.FilePermission`（リソースおよびアクションを含む）

表 2-1 は、Java パーミッション・インスタンスの特性を示したものです。

重要： `AllPermission` は必要な場合のみ使用し、使用する際には十分に注意する必要があります。

表 2-1 Java パーミッション・インスタンスの特性

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

保護ドメインにより、パーミッションがコードソースに関連付けられます（2-8 ページの「コードベースのセキュリティ」を参照）。保護ドメインを決定するのは、現在有効なポリシーです。`Policy` クラスのデフォルト実装では、保護ドメインはファイル内の 1 つの権限エントリです。

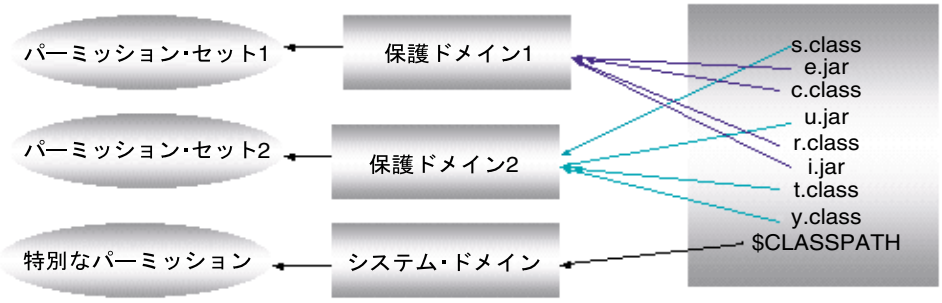
各 Java クラスは、ロード時に保護ドメインに関連付けられます。具体的には、ロードされるクラスはすべて `java.security.ProtectionDomain` インスタンスに関連付けられます。この保護ドメインに付与されるパーミッションは、静的にバインドするか、アクセス制御チェックが行われたときに動的に決定することができます。各保護ドメインには、JVM の起動時に、構成済のセキュリティ・ポリシーに基づいて一連のパーミッションが割り当てられます。

`ProtectionDomain` インスタンスには、1 つ以上のコードソースが含まれます。また、このインスタンスには、コードを実行するユーザー、クラス・ローダー参照および `Permission` オブジェクトのコレクションを表すパーミッション・セット（`java.security.PermissionCollection` インスタンス）を記述する `Principal` 配列を含めることもできます。

図 2-1 は、コード、保護ドメイン、パーミッション・セットの関係を示したものです。それぞれ次のようになります。

- 保護ドメイン 1 は、`e.jar`、`c.class`、`r.class`、`i.jar` のそれぞれのコードソースを、パーミッション・セット 1 に関連付けます。
- 保護ドメイン 2 は、`s.class`、`u.jar`、`t.class`、`y.class` のそれぞれのコードソースを、パーミッション・セット 2 に関連付けます。
- システム・ドメインは、クラスパス内にあるすべての JAR ファイルを、特別なパーミッションに関連付けます。

図 2-1 保護ドメイン



関連項目：

- プリンシパルの概要は、2-13 ページの「プリンシパルとサブジェクト」を参照してください。

Java 2 認可 : Java 2 セキュリティ・ポリシー

Java 2 の場合、ポリシーとは、実行するコードとそのコードに付与されるリソース・アクセス権限の間で行われるマッピングのことです。ポリシーの要素には、コードソース、パーミッション、保護ドメインなどがあります。これらについては、すべて前の項に記載されています。

J2SE 実装の場合、ポリシーは `java.security.Policy` インスタンスとして表されます。通常、このクラスと Java 2 セキュリティ・モデルは、OC4J 自体ではなく、基礎となる JDK に実装されます。

コードベースのパーミッションの場合、Java 2 ポリシーは、`java.policy` や `java2.policy` など（通常の場合の例）の `.policy` ファイル内で宣言されます。ポリシーには、コードベースへのパーミッション権限のコレクションが含まれ、キーストアへの参照を含めることもできます（1-5 ページの「鍵の暗号化および交換」を参照）。Java 2 ポリシー・ファイルは、通常次の場所にあります。

- *JAVA_HOME*`/lib/security/java.policy`
- *USER_HOME*`/java.policy`
- *ORACLE_HOME*`/j2ee/home/config/java2.policy`

  （セキュリティ・マネージャを使用して OC4J を起動すると、関連するパーミッションがこの場所のファイルに格納されます。）

関連項目：

- 5-2 ページの「Java 2 セキュリティ・マネージャおよびポリシー・ファイルの指定」
- 5-4 ページの「Java 2 ポリシー・ファイルの作成または更新」

Java 2 認可 : セキュリティ・マネージャおよびアクセス・コントローラ

セキュリティ・マネージャ（`java.lang.SecurityManager` インスタンス）を使用すると、アプリケーションで Java 2 セキュリティ・ポリシーを実装できるようになります。セキュリティ・マネージャを使用することにより、実行されたすべての操作に対して、操作の内容や実行の可否をアプリケーションが判断できるようになります。`SecurityManager` クラスには多数の `check`*Xxx*`()` メソッドがあり、各メソッドでは、操作 *Xxx* が許可されているかどうかをチェックし、許可されていない場合には例外をスローします。この中には、要求されたアクセス（所定のパーミッションによって指定されたもの）が現在有効なセキュリティ・ポリシーによって許可されていない場合に例外をスローするインスタンス・メソッド `checkPermission(Permission)` も含まれています。

また、アクセス制御の操作および判定には、アクセス・コントローラ（`java.security.AccessController` インスタンス）も関与します。`SecurityManager` メソッド `checkPermission(Permission)` のデフォルト実装では、実際には `AccessController` の静的メソッド `checkPermission(Permission)` がコールされます。ただし、`AccessController.checkPermission(Permission)` メソッドの場合、セキュリティ・マネージャは必要ありません。

基本的には、アクセス・コントローラは次を実行するために使用されます。

- 現在有効なセキュリティ・ポリシーに基づいて、システム・リソースへのアクセスを許可するかどうかを判定します。
- コードを権限ありとマークし、以降のアクセス判定に影響を及ぼすようにします。
- 現在のコール側コンテキストのスナップショットを取得し、この保存したコンテキストとの関連で、異なるコンテキストからのアクセス制御判定を行えるようにします。

システム・リソースへのアクセスを制御するアプリケーションの場合、AccessController メソッドによって使用される特定のセキュリティ・モデルおよびアクセス制御アルゴリズムを使用する予定であれば、アプリケーションによってこれらのメソッドを起動する必要があります。一方、実行時にインストールされる SecurityManager のセキュリティ・モデルに従うアプリケーションの場合、アプリケーションによって、SecurityManager インスタンスの対応するメソッドを起動する必要があります。

比較すると、SecurityManager はアクセス制御の中心点の概念を表すのに対し、AccessController は、doPrivileged() メソッド（有効な権限を使用して、特定の権限を必要とする指定された権限アクションを実行する）などの特別な機能を持つ特定のアクセス制御アルゴリズムを実装するものです。

アクセス制御コンテキスト（java.security.AccessControlContext インスタンス）の概念もあります。アクセス制御コンテキストを使用すると、特定のセキュリティ・コンテキストに基づいてリソースへのアクセスを制限できます。たとえば、特定の保護ドメイン・インスタンスの配列からアクセス制御コンテキストを作成できます。AccessControlContext クラスは、checkPermission(Permission) メソッドも定義します。この場合このメソッドは、現在の実行スレッドのコンテキストではなく、コール元である AccessControlContext インスタンスに基づいてアクセスを決定します。したがって、特定のコンテキストと関連するセキュリティ・チェックを異なるコンテキストから実行する必要がある場合に、AccessControlContext インスタンスを使用すると便利です。

重要： Java 2 ポリシーを使用するには、セキュリティ・マネージャを指定して有効にする必要があります（5-2 ページの「Java 2 セキュリティ・マネージャおよびポリシー・ファイルの指定」を参照）。これにより、JDK および Subject メソッドの doAs() と doAsPrivileged()（この章で後述）を使用して、実行するコードのパーミッションをチェックできます。

関連項目：

- セキュリティ管理の詳細、およびセキュリティ・マネージャとアクセス・コントローラの比較の詳細は、次を参照してください。

  http://java.sun.com/j2se/1.5.0/docs/guide/security/spec/security-spec.doc6.html

Java Authentication and Authorization Service

Java Authentication and Authorization Service（JAAS）は、アプリケーションでユーザーの認証とアクセス制御の強化に使用される Java パッケージです。JAAS は、Java 2 セキュリティを補完するように設計されています。認可は、実行するコード（Java 2 のコードベース・セキュリティ）と、コードの実行者（サブジェクトベースのセキュリティ）に基づいて行われます。

JAAS には、標準の Pluggable Authentication Module（PAM）フレームワークの Java バージョンも実装されます。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、カスタム認証モジュールの使用がサポートされます。

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、サブジェクトベースの認可がサポートされます。また、JAAS では、デプロイメント・ディスクリプタ内での宣言によるセキュリティ設定がサポートされるので、コードベースのセキュリティ設定に制限されることがありません。

OC4J には、スケーラブルな JAAS プロバイダが用意されています。OC4J は、密な認可用の標準メカニズムとして、独自のメカニズムではなく JAAS を使用します。OC4J は、Web ベース・アプリケーションと EJB ベース・アプリケーションの両方に対して JAAS 認可をサポートします。

次に示す項で、JAAS の特性と機能について説明します。

- プリンシパルとサブジェクト
- JAAS 認証 : ログイン・モジュール
- JAAS 認可 : サブジェクト・メソッド doAs() および doAsPrivileged()

関連項目：

- 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/`

プリンシパルとサブジェクト

プリンシパルとは、特定のアイデンティティ（`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、`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として表されます。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一ユーザーの関連情報のグループを表します。この関連情報には、サブジェクトのアイデンティティとロール、その他のセキュリティ関連属性（パスワード、暗号キー、その他の資格証明など）が含まれます。サブジェクトは、`javax.security.auth.Subject` クラスのインスタンスとして表されます。

ユーザーが認証されると、`Subject` インスタンスは認証されたユーザーを表すようになり、次に、適切な `Principal` インスタンスがこの `Subject` インスタンスに追加されます。`Principal` インスタンスは、認証されたユーザーが特定の権限アクションを実行することを認可するために使用されます。

JAAS 認証 : ログイン・モジュール

JAAS Pluggable Authentication フレームワーク内では、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ます。認証サービスは、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プラグインできます。ログイン・モジュールの主な役割は、提供された資格証明（たとえばパスワード）に基づいてユーザーを認証すること、および適切なプリンシパル（たとえばロール）をサブジェクトに追加することです。使用できる JAAS ログイン・モジュールの種類には、プリンシパル・マッピング用モジュール、資格証明マッピング用モジュール、Kerberos モジュールがあります。

関連項目：

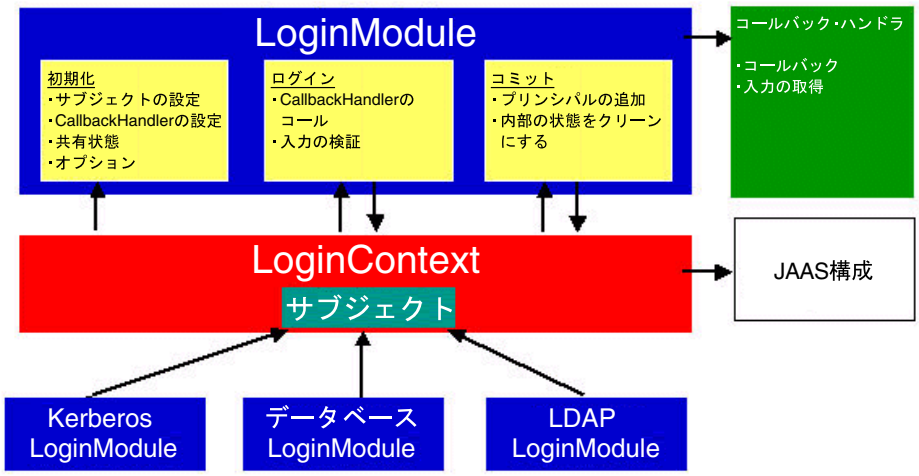
- 第 9 章「ログイン・モジュール」

ログイン・モジュールの概要

ログイン・モジュールは、`javax.security.auth.spi.LoginModule` インタフェースを実装するクラスのインスタンスで、特定タイプの認証を提供するためにアプリケーションにプラグインされます。

このフレームワーク内では、（たとえば、ユーザーがアプリケーションにログインしようとしたときに）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 `javax.security.auth.login.LoginContext` クラスで提供されます。アプリケーションでは、このクラスが、名前と（後述する）コールバック・ハンドラを使用してインスタンス化されます。サブジェクトがアクセスしようとしているアプリケーションによって `LoginContext` インスタンスの `login()` メソッドが起動されると、`LoginContext` インスタンスは、渡された名前を採用するメカニズムを使用して構成設定を参照し、アプリケーションに対して起動する適切なログイン・モジュールを決定します。
図 2-2 に、この概要とログイン・モジュールの機能を示します。

図 2-2 ログイン・モジュール



コールバック・ハンドラは、`javax.security.auth.callback.CallbackHandler` インスタンスであり、ログイン・モジュールがユーザーと対話してログイン情報を取得できるようにします。`CallbackHandler` によって指定される唯一のメソッドは、`handle(Callback[])` メソッドです。このメソッドは、`java.security.auth.callback.Callback` インタフェースを実装するクラスのインスタンスであるコールバックの配列を取ります。コールバックでは、基礎となるセキュリティ・サービスから要求された情報は取得または表示されません。リクエストをアプリケーションに渡し、必要に応じて、要求された情報をセキュリティ・サービスに戻すのみです。`javax.security.auth.callback` パッケージ内のコールバック実装には、ユーザー名を処理する名前コールバック・ハンドラ（`NameCallback`）、パスワードを処理するパスワード・コールバック・ハンドラ（`PasswordCallback`）およびログイン・

フォームにあるユーザー名フィールドやパスワード・フィールド以外の任意のフィールドを処理するテキスト入力コールバック・ハンドラ（TextInputCallback）があります。

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。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。

カスタムまたは外部（サード・パーティ）のログイン・モジュールを、指定のアプリケーションと一緒に使用することができます。Oracle には、ログイン・モジュールとして、RealmLoginModule（ファイルベースおよび LDAP ベースのプロバイダ用）、LDAPLoginModule（外部 LDAP プロバイダ用）、CoreIDLoginModule（Oracle Access Manager 用）および DBTableOraDataSourceLoginModule（データベース・アイデンティティ・ストアを使用するため）が用意されています。

注意： OC4J および OracleAS JAAS Provider とともに宣言による J2EE 認証を使用するアプリケーションの場合、LoginContext インスタンスを作成する必要はありません。これは、OC4J によって暗黙的に作成されます。

ログイン・モジュールのスタック

JAAS PAM アーキテクチャを使用すると、ログイン・モジュールのスタックを定義することにより、エンタープライズ・アプリケーションの認証メカニズムをカスタマイズできます。ログイン・モジュールはそれぞれ独立し、自分自身のユーザー・リポジトリに対して通信を行います。

javax.security.auth.login.Configuration クラスは、アプリケーションの下位にあるログイン・モジュールの構成を表すための抽象クラスです。Configuration インスタンスは、特定のアプリケーションに使用されるログイン・モジュールと、ログイン・モジュールの起動順序を指定します。Configuration クラスは、適切な実装を提供するために拡張されています。

各ログイン・モジュールに対しては、ログイン・モジュール構成の制御フラグ設定により、ログイン・モジュールが、required、requisite、sufficient、optional のいずれになるかが決定されます。これらの設定の意味は、Configuration クラスの標準的な機能に基づいています。9-17 ページの表 9-5「ログイン・モジュール制御フラグ」を参照してください。

ログイン構成には、制御フラグ設定および各ログイン・モジュール固有のオプション設定とともに、完全修飾されたクラス名によって指定されたログイン・モジュールの順序リストも含まれています。認証は、モジュール・リストの順番に従って実行されます。（OC4J の場合、この順序は、system-jazn-data.xml ファイルに構成されているログイン・モジュールの順序によって決定されます。）

認証全体は、個々のログイン・モジュールとその制御フラグ設定によって制御されます。

JAAS 認可 : JAAS セキュリティ・ポリシー

JAAS の場合、ポリシーとは、リソースとユーザー間（またはリソースとロール間）の関連付けのことです。JAAS を Java 2 セキュリティと統合した場合、Policy API はプリンシパルに基づいて問合せを処理し、デフォルトのポリシー実装はプリンシパルに基づいて権限エントリをサポートします。Java 2 セキュリティ・モデルの拡張の場合、アクセス制御は実行するコードのみではなく、コードの実行者にも基づいて行われます。具体的には、ポリシーは認可ルールのリポジトリで、受領者を指定した場合にその受領者に付与されるパーミッションは何か、という質問に答えるための情報が含まれています。

OC4J 10.1.3.x 実装の場合、ポリシーは javax.security.auth.Policy インスタンスによって表されます。このクラスは JDK 1.4 では非推奨ですが、OC4J 10.1.3.x 実装では完全サポートされており、Sun 社の JDK および J2SE では引き続きサポートされています。

OC4J の場合、JAAS ポリシーは system-jazn-data.xml ファイルの <jazn-policy> 要素内（または、Oracle Identity Management セキュリティ・プロバイダを使用している場合は Oracle Internet Directory 内）で宣言されます。（この機能は、Java 2 セキュリティ・モデルの .policy ファイルの機能に相当するものです。）この宣言により、ユーザーとロールにパーミッションが付与されます。OracleAS JAAS Provider Admintool を使用してパーミッションの付与や取消しを行うと、宣言は自動的に更新されます。

JAAS 認可 : サブジェクト・メソッド doAs() および doAsPrivileged()

Subject クラスには、JAAS モデルの認可用として、次のような標準的なメソッドが含まれています。

- Object doAs(Subject, PrivilegedAction)

  指定された権限アクション（有効化された権限を使用して実行される処理）を、指定されたサブジェクトとして実行します。このメソッドは、サブジェクトを（実行するコードソースの）現行スレッドのアクセス制御コンテキスト（AccessControlContext インスタンス）に関連付けます。その際、このアクセス制御コンテキストのパーミッションにサブジェクトのパーミッションを追加し、パーミッションを組み合せて新しいアクセス制御コンテキストを作成します。次に、指定されたアクションと新しいアクセス制御コンテキストを使用して、AccessController.doPrivileged() メソッドがコールされます。

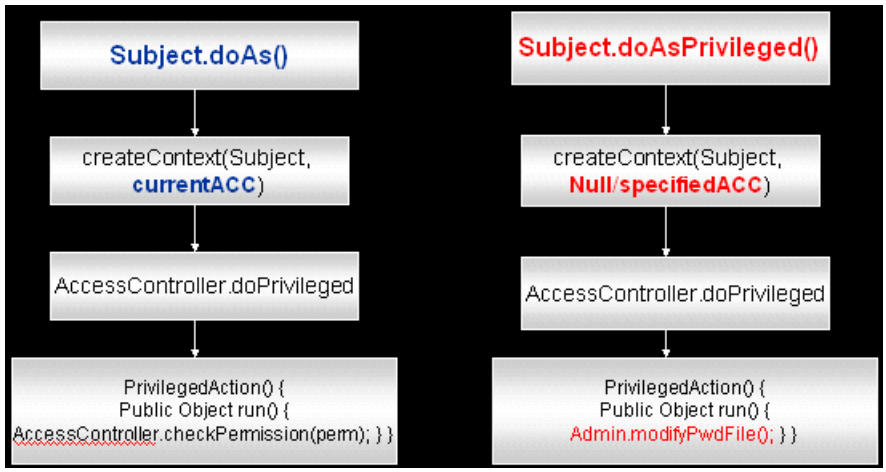
  戻されるオブジェクトは、権限アクションの run() メソッドによって戻されるオブジェクトです。また、チェック済例外をスローする処理用に、java.security.PrivilegedAction のかわりに java.security.PrivilegedExceptionAction を取るバリエーションもあります。

- Object doAsPrivileged(Subject, PrivilegedAction, AccessControlContext)

  このメソッドの機能は doAs() メソッドと同じです。ただし、スレッドのアクセス制御コンテキストを使用するかわりに、指定されたアクセス制御コンテキストにおいて、サブジェクトのパーミッションが指定されたコンテキストのパーミッションに追加されます。コードソースがアクセス権限を必要としないように設定する場合に、null アクセス制御コンテキストを実際に指定するのが、通常の使用方法です。

図 2-3 は、doAs() メソッドと doAsPrivileged() メソッドのサンプル・コード・スタックを示したものです。この場合は、パスワード・ファイルを変更します。doAsPrivileged() メソッドで null アクセス制御コンテキストを渡す場合、パスワード・ファイルにアクセスする十分な権限をサブジェクトに付与する必要があります。

図 2-3 doAs() メソッドおよび AsPrivileged() メソッドのコード・スタック



サブジェクト doAs() メソッドのプリンシパルに必要なパーミッション

サブジェクト doAs() メソッドを権限アクションとともに使用する場合、サブジェクトに関連付けられたプリンシパルに対して、必要なパーミッションを付与してください。この場合、OracleAS JAAS Provider Admintool を使用して付与できます。構成の結果は、system-jazn-data.xml ファイル内の <jazn-policy> 要素の下に表示されます。

次の例では、setContextClassLoader という実行時権限を、myapp という名前の PrincipalImpl プリンシパルに付与しています。

```
% java -jar jazn.jar -grantperm sun.security.acl.PrincipalImpl \
      myapp java.lang.RuntimePermission setContextClassLoader
```

これにより、system-jazn-data.xml ファイル内に次の構成が生成されます。

```
      <grant>
       <grantee>
         <principals>
          <principal>
            <class>sun.security.acl.PrincipalImpl</class>
            <name>myapp</name>
          </principal>
         </principals>
       </grantee>
       <permissions>
         <permission>
           <class>java.lang.RuntimePermission</class>
           <name>setContextClassLoader</name>
         </permission>
       </permissions>
      </grant>
```

関連項目：

- Admintool は、C-15 ページの「パーミッションの付与と取消し」を参照してください。
- `system-jazn-data.xml` ファイルは、4-7 ページの「system-jazn-data.xml ファイル」および 5-16 ページの「system-jazn-data.xml でのポリシー構成」を参照してください。

開発時におけるセキュリティの考慮事項

ここでは、セキュリティ開発サイクル時の考慮事項について説明します。セキュリティ・モデル間の比較の概要と、アプリケーション開発でセキュリティを確保するための手順を説明します。

概要 : J2EE、Java 2、JAAS のセキュリティ・モデル比較

この章で説明する J2EE、Java 2、JAAS の各セキュリティ・モデルの相違点の概要は、次のとおりです。

- J2EE 認可モデルの場合、保護するリソースは、URL パターン（Web アプリケーションの場合）か、メソッド名または名前マスク（EJB の場合）によって識別されます。認可は、デプロイメント構成に定義された宣言ロールベース・セキュリティに基づき、コンテナによって実行されます。コード内に実装する必要はありません。一度リソースへのアクセスが付与されると、そのリソースに対するすべての機能が利用できるようになります。このモデルは相対的に粗密ですが、ほとんどの目的にはこのモデルで十分です。このモデルは静的でもあるため、開発者またはデプロイヤは、ユーザー・コミュニティについて事前に知っておく必要があります。

  J2EE モデルの場合、認証はコンテナによって管理されます。

- Java 2 認可モデルは、コードベースです。定義済 Java 2 ポリシー・ファイルに設定されているパーミッション・セットを適用することにより、アプリケーションで実行できる操作を制限します。パーミッション・セットは、指定されたコードソースのコードに対して許可される操作の種類を決定します。（コードソースは、該当するコードの URL の場所と、資格証明の配列（オプション）から構成されます。）認可チェックはコード内に実装され、セキュリティ・マネージャまたはアクセス・コントローラ・オブジェクトによって実行されます。このモデルには、事前定義済ポリシー・ファイルを除き、静的な構成はありません。この認可モデルは、より動的かつ比較的密なモデルで、柔軟性のあるセキュリティ・ポリシーに基づいて認可が実行されます。

  J2EE モデルと同じように、認証はコンテナによって管理されます。

- JAAS 認可モデルは、Java 2 セキュリティを拡張します。プリンシパルに基づいた問合せを処理できるポリシー・オブジェクトと、プリンシパルに基づいた権限エントリをサポートするデフォルトのポリシー実装を使用します。Java 2 セキュリティと同じように、認可はアプリケーション・コード内で実行されます。JAAS の拡張を使用すると、アクセス制御は実行するコードのみではなく、コードの実行者にも基づいて行われます。ポリシー・オブジェクトは、指定されたコードソースに関連するプリンシパルに付与されたパーミッションを取得できます。Java 2 モデルと同じように、ポリシー・ファイルを除き、静的な構成はありません。J2EE モデルと比較した場合、JAAS モデルはより柔軟なカスタマイズと拡張が可能であり、カスタムのパーミッション・タイプなどの機能を備えています。また、JAAS モデルは Java 2 モデルよりも密であり、プリンシパルに基づいた認可が可能です。たとえば、Web の URL や EJB メソッドの一般的な保護には J2EE セキュリティで十分ですが、誰がファイル・システム内のファイルにアクセスできるのか、または誰がセキュリティ・ポリシーにアクセスし、ユーザーの作成やパスワードの変更を行えるのかを制御するには JAAS セキュリティが必要になってきます。

  また JAAS は、カスタム・ログイン・モジュールを使用して認証をカスタマイズできる唯一のモデルでもあります。さらに、複数のユーザー・リポジトリと照合して認証することもできます。

必要に応じて、任意のモデルを使用するか、アプリケーション内のモデルを組み合せることができます。OC4J の場合、3 種類のモデルすべてが完全サポートされています。ニーズが満たされるのであれば、J2EE 認可モデルを使用することをお薦めします。これは、管理者やデプロイヤにとって最も単純な方法です。必要に応じてアプリケーションを拡張し、密なコードベース・セキュリティやサブジェクトベース・セキュリティに対して、Java 2 セキュリティや JAAS セキュリティを使用することができます。

関連項目：

- アプリケーション・セキュリティの方法の詳細は、5-22 ページの「認可の方法」を参照してください。

セキュアな J2EE アプリケーションを開発する手順

J2EE ソフトウェアの開発は、開発 - デプロイ - 管理というサイクルで行われます。Oracle Application Server のセキュリティ実装は、このサイクルのデプロイ - 管理部分で重要な役割を果します。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、より便利な宣言によるセキュリティ・モデルを使用できます。

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. 開発者は、Web コンポーネント、Enterprise Bean、サーブレット、アプリケーション・クライアントを必要に応じて作成します。

   Oracle Application Server のセキュリティ実装はプログラム・インタフェースを提供しますが、開発者はそれを使用することなくコンポーネントを作成できます。

2. 開発者は J2EE 論理ロールを定義し、セキュリティ制約を使用してこれらのロールを権限に割り当てます。すべてのロールに対して、標準 J2EE デプロイメント・ディスクリプタの構成が使用されます。

3. 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オプションを指定します。

4. アセンブラは、アプリケーション・レベルのセキュリティ制約を定義し、モジュール・レベルの構成間で発生する可能性のある競合を解決します。

5. 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 J2EE ロールをデプロイメント・ユーザーおよびロールにマップすることができます（6-12 ページの「Application Server Control を介したセキュリティ・ロール・マッピングの指定」を参照）。

6. 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デプロイメント環境のロールと、ユーザーの作成および管理が含まれます。

Java 2 または JAAS の機能を使用した密なコードベースのアクセス制御やサブジェクトベースのアクセス制御には、次のような考慮事項があります。

1. 開発者は、アクセスされる可能性があり、必要に応じて保護が必要なすべてのリソースを特定する必要があります。

2. 開発者は、パーミッションを定義してリソースを保護する必要があります。（J2SE には事前に定義されたパーミッションがあり、必要に応じて使用できます。）

3. 開発者は、実行時認可チェック用のコードを実装する必要があります。

4. デプロイヤは開発者と協議し、アプリケーションに対して適切な JAAS モード構成を定義する必要があります。

5. システム管理者は、適切なパーミッションを強制するポリシー構成を維持管理する必要があります。OracleAS JAAS Provider Admintool を使用したパーミッションの付与など、ポリシーのプロビジョニングは、実行時よりも前に完了しておく必要があります。

関連項目：

- セキュリティのベスト・プラクティスの詳細は、『Oracle Application Server ベスト・プラクティス』を参照してください（リリース後に入手可能）。

3

OC4J セキュリティの概要

この章では、Oracle Containers for J2EE（OC4J）のセキュリティ実装の概要について説明します。この実装を使用すると、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。

この実装において重要なコンポーネントは、JAAS 仕様をサポートする Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider です。

この章の内容は次のとおりです。

関連項目：

- セキュリティのベスト・プラクティスの詳細は、『Oracle Application Server ベスト・プラクティス』を参照してください（リリース後に入手可能）。
- Oracle Application Server の一般的なセキュリティ情報およびインフラストラクチャについては、リリース 3（10.1.3.x）のドキュメント・セットに含まれていないが次の URL で入手可能な、リリース 2（10.1.2）の『Oracle Application Server セキュリティ・ガイド』を参照してください。

  ```
  http://www.oracle.com/technology/documentation/
  appserver1012.html
  ```

OracleAS JAAS Provider およびセキュリティ・プロバイダの概要

OC4J には、JAAS 実装、Oracle Application Server Java Authentication and Authorization Service（JAAS）Provider が用意されています。OracleAS JAAS Provider では、Java 2 セキュリティ・モデルを使用する J2SE アプリケーションおよび J2EE アプリケーションに簡単に統合でき、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を実装しています。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。

OC4J アプリケーション用セキュリティ・フレームワークには、ファイルベース、Oracle Identity Management（LDAP ディレクトリベース）、外部 LDAP ディレクトリ、Oracle Access Manager およびカスタム（カスタム・ログイン・モジュールを使用）といういくつかの特定セキュリティ・プロバイダも、OracleAS JAAS Provider 以外にサポートしているという重要な側面があります。

この項の以降の部分で、次の項目について説明します。

OracleAS JAAS Provider の概要

OracleAS JAAS Provider により、JAAS ログイン構成プロバイダ・インタフェースおよび JAAS ポリシー・プロバイダ・インタフェースが実装されます。

- ログイン構成プロバイダ実装の役割は、ログイン・モジュール構成情報を取得すること、および認証用に適切なログイン・モジュールが起動されるようにすることです。JAAS ログイン・モジュールの構成は、XML ファイルに格納されます。
- ポリシー・プロバイダ実装では、認可用ポリシーを格納するリポジトリとして、XML ファイルまたはディレクトリ・サービスの 2 つのいずれかがサポートされます。（これに対し、Sun 社のポリシー・プロバイダ実装では、ポリシー・リポジトリとして、たとえば *JAVA_HOME*`/jre/lib/security/java.policy` ファイルが使用されます。）ポリシーには、ファイルの読み書きなど、ユーザーによるリソースへのアクセスおよび使用を認可するためのルール（パーミッションまたは権限と呼ばれる）が含まれています。

OracleAS JAAS Provider を使用すれば、アプリケーションでリソース・ユーザーに対して密なアクセス制御を実施できます。セキュリティ対応アプリケーションが OC4J で実行されるときの 3 つの重要な手順は次のとおりです。

1. ログイン・モジュールを作成し、起動します。これには、OracleAS JAAS Provider が関与します。OC4J には、サポートされるセキュリティ・プロバイダ用のログイン・モジュールが用意されています。カスタム・ログイン・モジュールを使用することもできます。
2. ログインを試みているユーザーを認証します。これはセキュリティ・プロバイダが行います。
3. ユーザーが実行を試みている操作のパーミッションをチェックすることによってユーザーを認可します。これには、OracleAS JAAS Provider が関与します。

OracleAS JAAS Provider は、デフォルトで OC4J 製品の一部として構成されています。

注意： 以前のリリースでは、OracleAS JAAS Provider のかわりに JAZN という用語が使用されていました。現在、この用語は一般には使用されていませんが、コード（クラス名やパッケージ名として）および Admintool のシェル・プロンプトではまだ使用されています。

JAAS フレームワークの機能の概要

表 3-1 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 3-1 JAAS フレームワークの機能

機能	説明	関連項目
認証	■ J2EE アプリケーション環境のログイン認証用 Oracle シングル・サインオン・ソリューションと統合します。 ■ OracleAS Core または Java Edition など、非 SSO 環境向けにデフォルトの `RealmLoginModule` クラスを提供します。 ■ JAAS 準拠のカスタム・ログイン・モジュールをすべてサポートします。	3-6 ページの「OC4J 環境での認証機能の概要」
宣言によるモデル	■ `web.xml`、`ejb-jar.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。	
認可	■ J2EE 認可モデルをサポートします。 ■ JAAS 認可モデルをサポートします。 ■ Java Authorization Contract for Containers をサポートします。	5-5 ページの「OC4J 環境における認可 API、JAAS モードおよび JACC」
レルム管理	■ ユーザーおよびロール管理をサポートするために、パッケージ `oracle.security.jazn.realm` が提供されます。	
ポリシー管理	■ 認可ポリシーの管理のために、パッケージ `oracle.security.jazn.policy` が提供されます。	
管理	■ Oracle Enterprise Manager 10*g* またはコマンドラインの OracleAS JAAS Provider Admintool を使用して管理および構成をサポートします。	4-2 ページの「Oracle Application Server および OracleAS JAAS Provider 向けのツール」
`JAZNUserManager`	■ ファイルベース・プロバイダ、Oracle Identity Management および Oracle Access Manager と統合できるセキュリティ・プロバイダ実装を提供します。このクラスは、`oracle.security.jazn.oc4j` パッケージに含まれています。	

OracleAS JAAS Provider のセキュリティ・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 にはレルムと呼ばれるユーザー・コミュニティの概念があります。

レルムは、同じ認証ポリシーによって制御されるユーザーとロールのコレクションです。すなわち、レルムは認証済ユーザーに対して一連のパーミッションを定義するセキュリティ・ドメインです。

各レルムには、一連の構成されたユーザーおよびロールが含まれています。(OC4J 構成では、ユーザーとロールはすべてレルム定義内に構成できます。)

関連項目：

- 6-4 ページの「OC4J でのセキュリティ・レルムの使用方法」
- 8-18 ページの「LDAP ベース・プロバイダのレルム管理」

サポートされているセキュリティ・プロバイダ

Oracle Application Server では、次のセキュリティ・プロバイダがサポートされます。各セキュリティ・プロバイダは、事実上セキュリティ・プロバイダの一部である、該当するログイン・モジュール（ファイルベース・プロバイダおよび LDAP ベース・プロバイダの場合は `RealmLoginModule`）に関連付けられます。さらに、各セキュリティ・プロバイダでは、レルム情報（ユーザーとロール）と JAAS ポリシー情報（パーミッション）からなるデータをセキュアかつ集中的に格納、取得および管理するために、リポジトリが使用されます。

- ファイルベース（XML ベース）プロバイダ

  第 7 章「ファイルベースのセキュリティ・プロバイダ」で説明しているファイルベース・プロバイダは、XML リポジトリを使用する、高速かつ軽量な JAAS ログイン・モジュール実装です。ユーザー、ロールおよびポリシーの情報は、通常は OC4J インスタンスレベル・ファイル `system-jazn-data.xml` に格納されます。

  これがデフォルトのセキュリティ・プロバイダです。

- LDAP ベース・プロバイダ : Oracle Identity Management

  第 8 章「Oracle Identity Management」で説明しているように、Oracle Internet Directory をユーザー・リポジトリとして使用する場合（Oracle Single Sign-On の共用の有無は問わない）は、このセキュリティ・プロバイダを使用します。Oracle Identity Management プロバイダは、情報を集中的に格納するための Lightweight Directory Access Protocol（LDAP）に準拠しています。ユーザー、ロール、レルムおよびポリシー情報は、Oracle Internet Directory に格納されます。

  この本番環境向けのセキュリティ・プロバイダは、スケーラブルかつセキュアな、Oracle Single Sign-On と統合されたエンタープライズ対応のセキュリティ・プロバイダです。

  Oracle Identity Management を使用するためには、OC4J を Oracle Internet Directory インスタンスと関連付ける必要があります。

- 外部 LDAP プロバイダ

  Oracle Application Server では、第 10 章「外部 LDAP セキュリティ・プロバイダ」で説明しているように、Sun Java System Directory Server や Microsoft Active Directory などの外部（サード・パーティ）LDAP プロバイダがサポートされています。外部 LDAP プロバイダにより、ログイン・モジュール `LDAPLoginModule` が実装されます。

- カスタム・セキュリティ・プロバイダ

  Oracle Application Server では、ユーザーまたはサード・パーティが、アプリケーションに特別な認証機能を実装できるように、カスタム・ログイン・モジュールを使用してカスタム・セキュリティ・プロバイダを実装できます。これについては、第 9 章「ログイン・モジュール」で説明しています。カスタム・ログイン・モジュールでは、標準の JAAS ログイン・モジュール・インタフェースが実装されます。カスタム・ログイン・モジュールは、Oracle Enterprise Manager 10*g* を介してアプリケーションをデプロイするときに構成できます。この構成は、OC4J の `system-jazn-data.xml` ファイルに格納されます。

  カスタム・ログイン・モジュールのサポートは、ファイルベース・プロバイダの拡張を介して実装されます。

- Oracle Access Manager（旧称 Oracle COREid Access and Identity）

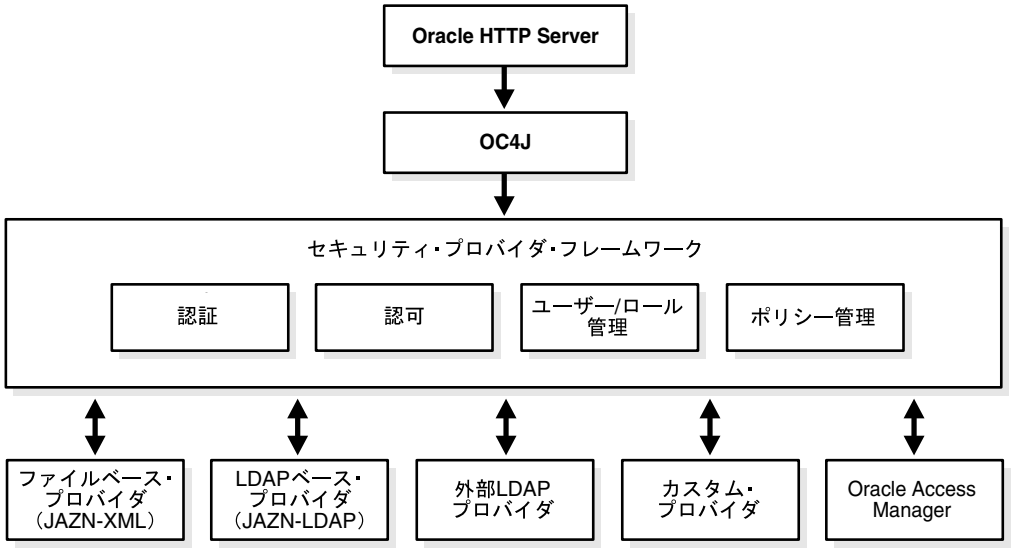
  OC4J 10.1.3.x 実装より、セキュリティ・プロバイダの選択肢に Oracle Access Manager が追加されました（第 11 章「Oracle Access Manager」を参照）。これは、集中的なセキュリティ管理を提供するエンタープライズ・クラスの認証、認可および監査ソリューションです。Oracle Access Manager には、様々なアプリケーション・サーバー、レガシー・アプリケーションおよびデータベースにわたる異機種間アプリケーション環境でアクセス制御、シングル・サインオン（Oracle Single Sign-On とは別）、パーソナライズおよびユーザー・プロファイル管理を行うための機能が含まれています。Oracle Access Manager によりログイン・モジュール `CoreIDLoginModule` が実装されます。

注意： このマニュアルでは、次の用語に注意してください。

- 「ファイルベース・プロバイダ」と「XML ベース・プロバイダ」（「JAZN-XML」とも呼ばれる）は、同義語です。
- Oracle Application Server との関連では、用語「LDAP ベース・プロバイダ」（「JAZN-LDAP」とも呼ばれる）は、Oracle Identity Management およびそのリポジトリである Oracle Internet Directory を意味します。
- OC4J 10.1.3.x 実装では、用語「カスタム・セキュリティ・プロバイダ」は、基本的には「カスタム・ログイン・モジュール」と同義です。

図 3-1 に、サポートされているセキュリティ・プロバイダがどのようにセキュリティ・プロバイダ・フレームワーク全体と対話するかを示します。

図 3-1 OC4J セキュリティ・アーキテクチャ



OC4J 環境での認証機能の概要

この項では、OC4J における認証に関する次の項目について説明します。

- サポートされる Web アプリケーションの認証方式
- OC4J ログイン・モジュールの概要
- Oracle Application Server シングル・サインオン代替方法の概要
- JAZNUserManager の委任（ファイルベース・プロバイダ）

サポートされる Web アプリケーションの認証方式

OC4J では、Basic、Digest、フォームベースおよび Client-Cert という標準的な認証方式がサポートされます。これら認証方式の概要は 2-2 ページの「Web アプリケーションの標準認証方式」に記載されています。また、認証方式は標準の `web.xml` ファイルに構成されています。詳細は、17-2 ページの「認証方式（auth-method）の指定」を参照してください。

OC4J では、Oracle Application Server と OC4J に提供されている、各種シングル・サインオン認証方式もサポートされます。これら認証方式の概要は、以降の「Oracle Application Server シングル・サインオン代替方法の概要」に記載されています。また、この項には詳細の参照先も記載されています。

OC4J ログイン・モジュールの概要

Oracle は、次のログイン・モジュールを提供しています。

- `RealmLoginModule`。ファイルベース・プロバイダまたは Oracle Identity Management 用のデフォルト・ログイン・モジュールです（9-4 ページの「RealmLoginModule」を参照）。
- `LDAPLoginModule`。LDAP 外部プロバイダ用です（10-2 ページの「外部 LDAP プロバイダの構成と管理の概要」を参照）。
- `DBTableOraDataSourceLoginModule`。データベースのユーザー・リポジトリ用です。`DataSourceUserManager` クラスの以前の機能と置き換えられました（9-5 ページの「DBTableOraDataSourceLoginModule」を参照）。
- `CoreIDLoginModule`。Oracle Access Manager とともに使用するためのものです（11-19 ページの「Oracle Access Manager ログイン・モジュールの構成」を参照）。

OC4J では、ログイン・モジュール標準に準拠するカスタム・ログイン・モジュールもサポートされます。このことについては、第 9 章「ログイン・モジュール」全体で説明しています。

Oracle Application Server シングル・サインオン代替方法の概要

シングル・サインオンとは、ユーザーが一度ログインすると、OC4 のインスタンスまたはクラスタ内で複数の Web アプリケーションにアクセスできる機能のことです。次のシングル・サインオン認証方式は、Oracle に固有です。これらの認証方式を使用するかどうかは、標準の `web.xml` ファイルではなく、Oracle の `orion-application.xml` ファイルの構成で指定されます。

- **SSO**

  この認証方式の場合、ユーザーの認証には OracleAS Single Sign-On が使用されます。このためには、Oracle Identity Management、Oracle Internet Directory および OracleAS Single Sign-On を含む Oracle Application Server インフラストラクチャが必要です。OracleAS Single Sign-On の詳細は、第 8 章「Oracle Identity Management」を参照してください。

- **COREIDSSO**

  この認証方式の場合、ユーザーの認証には Oracle Access Manager シングル・サインオン（OracleAS Single Sign-On とは異なる）が使用されます。このためには、Oracle Access Manager を含むインフラストラクチャが必要です。Oracle Access Manager SSO の詳細は、第 11 章「Oracle Access Manager」を参照してください。

- **CUSTOM_AUTH**（Java シングル・サインオン用）

  OC4J 10.1.3.1 実装の新機能である Java SSO は、小規模なデプロイメント環境を希望するカスタマ向けの代替 SSO ソリューションであり、OC4J にパッケージ化されています。Java SSO を適切に構成し、有効化すると、OC4J アイデンティティ管理フレームワーク（概要は、3-9 ページの「アイデンティティ管理フレームワークと API の一般的な使用の概要」を参照）の CUSTOM_AUTH 設定によって Java SSO が有効になります。Java SSO は、アイデンティティ管理フレームワークのデフォルト実装です。Java SSO は OC4J コンテナ自体の一部としてパッケージ化されており、追加インフラストラクチャを必要としません。Java SSO の詳細は、第 14 章「OC4J Java シングル・サインオン」を参照してください。

JAZNUserManager の委任（ファイルベース・プロバイダ）

OracleAS JAAS Provider JAZNUserManager によって、OC4J の認証が調整されます。Web アプリケーションの場合、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 が認証対象ユーザーの情報（SSO 用 Oracle HTTP Server mod_osso モジュールなどによって設定）を HTTP リクエスト・オブジェクトから取得し、JAAS サブジェクトを OC4J 内に設定します。

JAZNUserManager は、後述するように委任モデルをサポートしますが、実質的にはこれはファイルベース・プロバイダにのみ適用されます。委任を使用すると、ユーザーまたはグループがアプリケーションレベル JAZNUserManager インスタンスに見つからない場合、リクエストが親ユーザー・マネージャに委任されます。

次の制限事項および説明に特に注意してください。

- アプリケーションと親アプリケーションが両方ともファイルベース・プロバイダを使用するように構成されている場合、親がファイルベース・プロバイダを使用する構成でなくなるまで、必要なだけ親階層をさかのぼって委任が行われます。委任はそのポイントを越えては伝播されません。
- アプリケーションがファイルベース・プロバイダを使用するように構成され、親が LDAP ベース・プロバイダ、外部 LDAP プロバイダまたはカスタム・ログイン・モジュールを使用するように構成されている場合、委任はサポートされません。
- アプリケーション自体が LDAP ベース・プロバイダ、外部 LDAP プロバイダまたはカスタム・ログイン・モジュールを使用するように構成されている場合、委任はサポートされません。

重要： これは、開発者が JAZNUserManager のかわりに提供した UserManager 実装にも適用されます。ただし、開発者提供の UserManager クラスは、OC4J 10.1.3.x 実装では非推奨になっており、将来のリリースではサポートされない予定です。かわりとしてカスタム・ログイン・モジュールを使用してください。

注意： OC4J では、system アプリケーションが階層のルートにありますが、default アプリケーションがデプロイされたアプリケーションのデフォルトの親です。両方とも system-jazn-data.xml をユーザー・リポジトリとして使用します。

関連項目：

- 2-2 ページの「Web アプリケーションの標準認証方式」

OC4J 環境での認可機能の概要

OC4J 環境の認可には次のような機能があります。詳細は、第 5 章「OC4J での認可」を参照してください。

- Java 2 セキュリティおよびコードベース・ポリシー管理のサポート（標準の `java2.policy` ファイルの使用を含む）。
- JAAS モードの設定。これにより、標準 `Subject` クラスの `doAs()` メソッドおよび `doAsPrivileged()` メソッドに関連するセキュリティ動作が決定されます。
- パーミッションの付与、チェックおよび取消し（Oracle 権限クラス `RMIPermission`、`AdminPermission`、`RoleAdminPermission`、`JAZNPermission` および `RealmPermission`）。
- JAAS ポリシーの管理。
- Java Authorization Contract for Containers の実装。

セキュリティ・ロール・マッピングの概要

OC4J では、標準のディスクリプタに定義された J2EE 論理ロール（単に「J2EE ロール」と呼ばれます）をデプロイ・ロール（このドキュメントの以前のバージョンでは「JAAS ロール」と呼ばれていました）にマップし、特定のデプロイ・ロールのメンバーであるユーザーがリソースにアクセスできるようにします。このリソースには、関連付けられた J2EE ロールからアクセスできます。デプロイ・ロールはセキュリティ・プロバイダ内に定義されます。たとえば、ファイルベース・プロバイダの場合は `system-jazn-data.xml` に、LDAP ベース・プロバイダの場合は Oracle Internet Directory に、外部 LDAP プロバイダの場合はカスタム・ログイン・モジュールまたは Oracle Access Manager に定義されます。

セキュリティ・ロールの構成およびマッピングの基本的な手順は、次のとおりです。

1. 標準の J2EE 機能を使用して、J2EE 論理ロールをデプロイメント・ディスクリプタ（`web.xml` および `ejb-jar.xml`）に指定します。この手順には、OC4J 固有のものはありません。J2EE ロールは、`<security-role>` 要素で宣言されます。
2. 必要に応じて、アプリケーション・コード内に定義されたアプリケーション論理ロールを `<security-role>` 要素に宣言された J2EE ロールにリンクするためのセキュリティ・ロール参照を指定します。これは、標準のデプロイメント・ディスクリプタ内で `<security-role-ref>` 要素を使用して行えます。このメカニズムを利用することで、アプリケーション・コードを変更せずに論理セキュリティ・ロールの定義を調整することができます。後は、希望するように J2EE 論理ロールをアプリケーション・ロールにリンクするのみです。この手順には、OC4J 固有のものはありません。
3. デプロイ・ロールを構成するか、デフォルトのロールを使用します。たとえばファイルベース・プロバイダの場合、デプロイ・ロールが定義されている場所は OC4J の `system-jazn-data.xml` ファイル、またはアプリケーション固有の `jazn-data.xml` ファイルです。LDAP ベース・プロバイダの場合、デプロイ・ロールは Oracle Internet Directory に定義されています。
4. J2EE ロールをデプロイ・ロールにマップします。これを行うには、Application Server Control を使用して、マッピングを `orion-application.xml`、`orion-ejb-jar.xml` または `orion-web.xml` の `<security-role-mapping>` 要素に反映します。

関連項目：

- 6-11 ページの「セキュリティ・ロールのマッピング」
- 17-7 ページの「Web アプリケーションのセキュリティ・ロールおよび制約の構成」
- 18-2 ページの「EJB アプリケーションの認証と認可」

アイデンティティ管理フレームワークと API の一般的な使用の概要

OC4J 10.1.3.1 実装には、新機能であるサード・パーティのアイデンティティ・リポジトリ用の汎用目的サポートが用意されています。このサポートには、次のような機能があります。

- 交換可能なアイデンティティ管理フレームワーク。これにより、異機種サード・パーティ・システムを OC4J に統合し、J2EE アプリケーションをサード・パーティ・システムと相互運用することができます。サード・パーティのアイデンティティ管理システムと OC4J との統合は、標準の JAAS ログイン・モジュールに基づいています。

  このフレームワークの使用方法、プログラム・インタフェース、構成機能などは、第 13 章「交換可能なアイデンティティ管理フレームワーク」を参照してください。

  Java SSO は代替の Java シングル・サインオン・ソリューションであり、他のシングル・サインオン製品で必要な追加インフラストラクチャに依存しません。Java SSO は交換可能なアイデンティティ管理フレームワークを介して実装されます。Java SSO は、第 14 章「OC4J Java シングル・サインオン」を参照してください。

- 異種のアイデンティティ管理リポジトリからユーザーとロールの情報にアクセスするためのアイデンティティ管理 API のフレームワーク。このユーザーとロールの API フレームワークでは、基礎となるアイデンティティ・リポジトリに関係なく、一貫した移植可能な方法でアプリケーションがアイデンティティ情報（ユーザーとロール）にアクセスできます。基礎となるリポジトリは、Oracle Internet Directory、Active Directory（Microsoft 社提供）、Sun Java System Directory Server（Sun 社提供）などの LDAP ディレクトリ・サーバー、データベース、フラット・ファイルまたは他のカスタム・リポジトリのいずれでも可能です。サポートされる操作には、ユーザーとロールの検索、作成、更新、削除などがあります。

  第 13 章で説明する交換可能なアイデンティティ管理フレームワークとの混同を避けるために、このアイデンティティ管理 API フレームワークを「ユーザーおよびロール API」または「ユーザーおよびロール API フレームワーク」と呼びます。これら API の使用方法は、第 12 章「ユーザーおよびロール API フレームワーク」を参照してください。

4

セキュリティの管理の概要

この章では、OC4J および Oracle Application Server のセキュリティの管理および構成のための各機能 / ツールの概要を説明します。この章の内容は次のとおりです。

- OC4J のデプロイおよび構成のための一般的機能
- Oracle Application Server および OracleAS JAAS Provider 向けのツール
- JMX および MBean の管理
- 構成ファイルおよびその重要な要素の概要
- OC4J の System アプリケーション
- OC4J アカウントのサマリー
- 構成リポジトリおよびセキュリティ管理ツールのサマリー

OC4J のデプロイおよび構成のための一般的機能

OC4J では、J2EE 環境においてアプリケーションをデプロイおよび管理するために、次の標準がサポートされています。

- Java Management Extensions（JMX）1.2 仕様 : これを使用すると、J2EE 環境においてサービスおよびアプリケーションなどのリソースを管理するための標準インタフェースを作成できます。JMX の OC4J 実装では、OC4J サーバーおよびサーバー上で動作するアプリケーションを完全に管理できるユーザー・インタフェースが提供されます。
- Java 2 Platform, Enterprise Edition 管理仕様（JSR-77）: これを使用すると、MBean（マネージド Bean）と呼ばれるオブジェクトを作成し、J2EE 環境におけるアプリケーションのランタイム管理に使用できます。OC4J では、MBean には Oracle Enterprise Manager 10*g* のシステム MBean ブラウザから直接アクセスできますが、MBean のプロパティの多くは、Enterprise Manager のその他の機能を使用して、よりユーザー・フレンドリな形で公開できます。
- Java 2 Enterprise Edition デプロイメント API 仕様（JSR-88）: J2EE のアプリケーションおよびモジュールを J2EE 互換環境として構成およびデプロイするための標準 API が定義されています。OC4J 実装には、コンポーネントを OC4J にデプロイするために必要な OC4J 固有の構成データを入れるデプロイ・プランの作成および編集機能が搭載されています。

関連項目：

- OC4J のデプロイ、構成および管理の一般情報は、『Oracle Containers for J2EE デプロイメント・ガイド』および『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

Oracle Application Server および OracleAS JAAS Provider 向けのツール

J2SE および J2EE 環境におけるセキュリティの管理では、レルム、ユーザー、ロール、パーミッションおよびポリシーの作成および管理を行う必要があります。次の Oracle ツールをセキュリティ構成の管理で使用します。

- Oracle Enterprise Manager 10*g* Application Server Control: デプロイ中およびデプロイ後の全体的なセキュリティ管理および構成、ならびにファイルベース・プロバイダの管理に使用します。
- OracleAS JAAS Provider Admintool: ファイルベース・プロバイダの管理、および任意のセキュリティ・プロバイダのポリシーおよびログイン・モジュールの管理に使用します。
- Oracle Identity Management および Oracle Internet Directory ツールの Oracle Delegated Administration Service（DAS）および Oracle Directory Manager（`oidadmin`）: Oracle Internet Directory において、Oracle Identity Management 用にユーザーおよびロールの管理に使用します。

これらのツールに関しては、次の各項で詳細に説明します。

注意： OC4J セキュリティを含む OC4J の管理ツールとして、できるだけ Oracle Enterprise Manager 10*g* Application Server Control を使用するようにしてください。Application Server Control がサポートしていない機能については、必要に応じて OracleAS JAAS Provider Admintool を使用してください。場合によっては、構成ファイル、特にインスタンス・レベルの構成ファイルである `jazn.xml`（詳細は 4-9 ページの「jazn.xml ファイル」を参照）を直接操作する必要があります。

関連項目：

- 4-16 ページの「構成リポジトリおよびセキュリティ管理ツールのサマリー」

Oracle Enterprise Manager 10*g* Application Server Control の概要

アプリケーションのデプロイおよび管理には、通常 Application Server Control を使用します。このためのユーザー・インタフェースが Application Server Control コンソールです。Application Server Control には次の機能が用意されています。

- アプリケーションの OC4J へのデプロイ。デプロイ・プラン・エディタがあります。また、セキュリティに関しては、デプロイ時にセキュリティ・プロバイダおよびセキュリティ・ロール・マッピングを指定する機能があります。
- システム MBean ブラウザを使用した MBean の構成および操作（詳細は 4-5 ページの「JMX および MBean の管理」を参照）。ただし、MBean のプロパティに該当する多くのパラメータは、Application Server Control コンソールのその他のページでも公開されていることに注意してください。OC4J MBean の直接操作はできるだけ避けてください。
- デプロイ後のセキュリティ・プロバイダの変更またはセキュリティ・プロバイダ設定の更新。
- OC4J ランタイムの管理および構成の実行。

OC4J 固有の XML 構成ファイルは、Application Server Control コンソールを使用すると、OC4J によって自動更新されます。

注意：

- スタンドアロン OC4J では、OC4J の `admin_client.jar` のコマンドライン・ツールも使用できます。これは OC4J `system` アプリケーションを介して、J2EE アプリケーションをデプロイおよびバインドします。あるいは、Oracle JDeveloper ツールを使用してアプリケーションを開発している場合は、アプリケーションおよびリソース・アダプタのデプロイにも使用できます。
- Application Server Control または OC4J セキュリティ・プロバイダ MBean を使用して構成を変更した場合は、アプリケーションを再起動する必要があります。アプリケーションを再起動するまで、セキュリティ・プロバイダ MBean の他の操作はすべて無効になり、エラー・メッセージが戻されます。

関連項目：

- Application Server Control の詳細は、『Oracle Application Server 管理者ガイド』を参照してください。
- `admin_client.jar` ユーティリティの詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。
- 4-11 ページの「OC4J の System アプリケーション」

OracleAS JAAS Provider Admintool の概要

開発時に使用する OracleAS JAAS Provider Admintool は、次の管理機能を持つ軽量の Java アプリケーションです。

- ファイルベース・プロバイダ用 : ユーザー、ロール、ポリシーの管理
- Oracle Identity Management 用 : ポリシーの管理と、ユーザーおよびロールに対する読取り専用アクセスの管理
- 外部 LDAP プロバイダ用 : ポリシーおよびログイン・モジュールの管理
- カスタム・セキュリティ・プロバイダ用 : ポリシーおよびログイン・モジュールの管理

管理機能は、コマンドラインまたは対話型のシェルから直接コールできます。Admintool は、*ORACLE_HOME*/j2ee/home/jazn.jar にあります。

一般的なコマンドライン構文は次のとおりです。

```
% java -jar jazn.jar [-user username -password pwd] [option1 option2 ... ]
```

ファイルベース・プロバイダに対して Admintool を使用すると、*ORACLE_HOME*/j2ee/home/config ディレクトリの system-jazn-data.xml ファイルがデフォルトで更新されます。

注意 : 一般に、Admintool による変更は、OC4J を再起動するまで有効になりません。

関連項目 :

- 付録 C「OracleAS JAAS Provider Admintool リファレンス」

Oracle Identity Management および Oracle Internet Directory ツールの概要

この項では、Oracle Identity Management をセキュリティ・プロバイダとして使用する場合の、Oracle Internet Directory の管理ツールについての概要を説明します。

Delegated Administration Service の概要

委任管理は、Oracle Identity Management インフラストラクチャの重要な機能の 1 つです。ユーザー、グループおよびサービスのすべてのデータを中央のディレクトリに保存し、それらのデータの管理を複数の管理者およびエンド・ユーザーに委任できます。委任は、使用する環境の様々なセキュリティ要件を満たす方法で行われます。

たとえば、企業がすべてのユーザー、グループおよびサービスのデータを中央ディレクトリに保存しており、ユーザー・データの管理に 1 人の管理者、電子メール・サービスの管理にもう 1 人の管理者が必要であると想定します。Oracle Identity Management インフラストラクチャが提供する委任管理を使用することで、セキュリティ要件の異なる複数の管理者が、中央で保存されているデータをセキュアでスケーラブルな方法で管理することができます。Oracle Delegated Administration Service で委任できる権限は、(特に) ユーザーおよびグループの作成、編集および削除、ユーザーおよびグループへの権限の割当て、サービスおよびアカウントの管理です。

Oracle Delegated Administration Service (DAS) には、ユーザーの代理として行う Web ベースのディレクトリ操作単位のセットが事前定義されています。ディレクトリ管理者は、これを使用して、各役割をより多くその他の管理者およびエンド・ユーザーに委任することで、日常的なディレクトリ管理タスクから開放されます。ユーザー・エントリの作成、グループ・エントリの作成、エントリの検索およびユーザー・パスワードの変更など、ディレクトリ対応のアプリケーションで必要になるほとんどの機能が用意されています。

DAS を使用すると、ディレクトリのアプリケーション・データを管理する独自のツールを開発できます。または、DAS をベースにしたツールである Oracle Internet Directory セルフ・サービス・コンソールを使用することもできます。このツールは、Oracle Internet Directory で使用できるようになる予定です。

関連項目：

- 『Oracle Identity Management 委任管理ガイド』

Oracle Directory Manager の概要

Oracle Directory Manager は、Java ベースのグラフィカル・ユーザー・インタフェースを持つ管理ツールであり、Oracle Internet Directory の管理に使用できます。実行可能ファイルは *ORACLE_HOME*/bin ディレクトリに置かれ、コマンドラインから次のように実行できます。

```
% oidadmin
```

一般に、Application Server Control で実行できない、ディレクトリ固有の構成またはメンテナンス・タスクは、Oracle Directory Manager（および Oracle Internet Directory が提供する各種コマンドライン・インタフェース）で実行できます。

Oracle Directory Manager は次のタスクに対して使用できます。

- レルムの構成
- パスワード・ポリシーの指定
- Oracle ディレクトリ同期化サービスならびに Oracle Internet Directory のコネクタおよびエージェントの構成

また、属性の一意性、プラグイン、ガベージ・コレクション、変更ログ、レプリケーション、問合せの最適化、デバッグのロギングおよびアクセス制御リストなどの機能も管理できます。

関連項目：

- Oracle Directory Manager の一般情報は、『Oracle Internet Directory 管理者ガイド』を参照してください。

JMX および MBean の管理

OC4J は JMX 仕様をサポートしているため、J2EE 環境においてリソースを動的に管理する、標準インタフェースを作成できます。JMX の OC4J 実装によって、JMX クライアントであるシステム MBean ブラウザが提供され、OC4J 付属の MBean を介して、OC4J インスタンスを管理できます。

MBean は、JMX で管理可能なリソースを表す Java オブジェクトです。OC4J 内にある管理可能なリソースは、適切な MBean のインスタンスを介して管理されます。OC4J 付属の各 MBean は、Application Server Control コンソールのシステム MBean ブラウザを介してアクセス可能な管理インタフェースを公開しています。MBean 属性の設定、MBean のメソッドをコールする操作の実行、エラーまたは特定イベントの通知のサブスクライブ、および実行統計の表示ができます。

OC4J ホームページからこのブラウザにアクセスするには、**「管理」**タブを選択し、タスク・リストから JMX タスクの「システム MBean ブラウザ」を選択します。ブラウザからは次の処理が可能です。

- 左側のフレームで作業対象の MBean を選択します。
- 右側のフレームの**「属性」**タブを使用し、属性を表示または変更します。設定可能な属性には、新しい値を入力できるフィールドが表示されます。入力後、変更を適用します。
- 右側のフレームの**「操作」**タブを使用し、MBean のメソッドを起動します。実行する操作を選択します。操作ウィンドウで、指定したパラメータ設定を使用して操作を起動できます。
- 右側のフレームに**「通知」**タブが表示される場合は、このタブを使用し、通知をサブスクライブします。通知の必要なアイテムを選択し、変更を適用します。
- 右側のフレームに**「統計」**タブが表示される場合は、このタブを使用し、実行統計を表示します。

MBean およびその属性の変更が有効になるタイミングは様々であることに注意してください。ランタイム・モデルでは、変更がただちに有効になります。構成モデルでは、変更が有効になるタイミングには、変更の種類に応じて、リソースの再起動時、アプリケーションの再起動時、および OC4J の再起動時があります。変更が永続するかどうかもそれぞれ異なります。

構成ファイルおよびその重要な要素の概要

この項では、セキュリティ構成の上で重要な、次の XML ファイルおよびその要素の概要を示します。

- orion-application.xml ファイル（<jazn> および <jazn-web-app> 要素）
- system-application.xml ファイル
- system-jazn-data.xml ファイル
- アプリケーション固有の jazn-data.xml ファイル（オプション）
- jazn.xml ファイル

注意： 一般的には、（どちらもこの章で前述した）Application Server Control コンソールまたは OracleAS JAAS Provider Admintool を構成および管理に使用し、構成ファイルの直接操作は行いません。これらのツールを使用することで、適切なエントリが構成ファイルに自動的に設定されます。

関連項目：

- 4-16 ページの「構成リポジトリおよびセキュリティ管理ツールのサマリー」
- `orion-application.xml`（および `system-application.xml`）ファイルの要素の詳細は、『Oracle Containers for J2EE 開発者ガイド』の付録を参照してください。

orion-application.xml ファイル（<jazn> および <jazn-web-app> 要素）

OC4J の `orion-application.xml` ファイルは、（セキュリティ関連のみでなく）一般的なアプリケーション・レベルの構成に使用されます。このファイルの設定は、1 つの J2EE アプリケーション（EAR ファイル）にのみ適用されます。

`orion-application.xml` のセキュリティ設定は、`<jazn>` 要素で行います。具体的には、この要素は、セキュリティ・プロバイダ、ユーザーおよびロール・リポジトリの場所、およびアプリケーションのデフォルト・レルムを指定できます。ファイルベース・プロバイダを使用している次の例を参照してください。

```
<jazn provider="XML" location="./system-jazn-data.xml" default-realm="jazn.com" >
   ...
</jazn>
```

（4-7 ページの「system-jazn-data.xml ファイル」で説明している `system-jazn-data.xml` ファイルは、実際はデフォルトのリポジトリですが、ここでは説明上指定してあります。）

`orion-application.xml` の `<jazn>` のサブ要素は、Web アプリケーション用の OC4J 固有の認証方式を（`auth-method` 属性を使用して）指定する `<jazn-web-app>` 要素です。

注意：

- orion-application.xml に対する変更を有効にするには、Application Server Control またはセキュリティ・プロバイダ MBean によって変更した場合はアプリケーションの再起動、手動によって変更した場合は OC4J の再起動が必要です。
- <jazn> 要素が orion-application.xml に指定されていない場合は、インスタンス・レベルの jazn.xml ファイルのセキュリティ・プロバイダの設定が採用されます（このファイルでのデフォルト設定は、system-jazn-data.xml リポジトリおよび jazn.com のデフォルト・レルムです）。

system-application.xml ファイル

OC4J 構成ファイルは、4-11 ページの「OC4J の System アプリケーション」で説明されている、OC4J の system アプリケーションと関連付けられています。system アプリケーションの場合は、system-application.xml が、デプロイ済アプリケーションに対する orion-application.xml ファイルに相当します。

system-application.xml ファイルは、それに含まれる <jazn> 要素を介して、OC4J インスタンス・レベルのユーザーおよびロール設定（特殊な OC4J 機能で使用されるものを含む）にファイルベースのセキュリティ・プロバイダを指定します。system-application.xml ファイルは、同じくインスタンス・レベルの system-jazn-data.xml ファイル（次の項を参照）を、これらの設定（格納場所は <jazn-realm> 要素内）のリポジトリとして参照します。

デフォルトでは、OC4J は system-application.xml を、*ORACLE_HOME*/j2ee/*instance_name*/config ディレクトリに存在するものとみなします。

system-jazn-data.xml ファイル

system-jazn-data.xml ファイルは、OC4J 10.1.3 実装で追加されたファイルです。このファイル（および system-application.xml）は、4-11 ページの「OC4J の System アプリケーション」で説明されている、OC4J の system アプリケーションに関連付けられています。

system-application.xml ファイルは、system-jazn-data.xml ファイルを、ファイルベース・プロバイダに対する、OC4J インスタンス・レベルのユーザーおよびロール設定のリポジトリとして参照します（設定は <jazn-realm> 要素下に格納されます）。ファイルベース・プロバイダは、認証および認可に system-jazn-data.xml を使用します。（ファイルベース・プロバイダがデフォルトのセキュリティ・プロバイダであることに注意してください。）

アプリケーションに対してファイルベース・プロバイダを使用する場合、オプションでユーザー・リポジトリとして system-jazn-data.xml を使用することができます。また、次項「アプリケーション固有の jazn-data.xml ファイル（オプション）」で説明するように、アプリケーションとともにパッケージ化するアプリケーション固有の jazn-data.xml ファイルを使用することもできます。

system-jazn-data.xml ファイルには、JAAS ログイン・モジュール構成（<jazn-loginconfig> 要素下に）と JAAS ポリシー構成（<jazn-policy> 要素下に）も保存されます。

デフォルトでは、OC4J は system-jazn-data.xml ファイルを、*ORACLE_HOME*/j2ee/*instance_name*/config ディレクトリに存在するものとみなします。

変更内容を system-jazn-data.xml ファイルや、（ファイルベース・プロバイダについて必要な場合に）アプリケーション・レベルの jazn-data.xml ファイルに書き込む頻度を制御する永続性モードがあります。永続性には、インスタンス・レベルの jazn.xml ファイルまたはアプリケーション・レベルの orion-application.xml ファイル内の <jazn> 要素の persistence 属性により、次に示す 3 種類の値があります。

- NONE: 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。

次に例を示します。

```
<jazn provider="XML" persistence="ALL" ... >
   ...
</jazn>
```

関連項目：

- ファイルの階層、要素、属性の詳細は、付録 D「OracleAS JAAS Provider 構成ファイル」を参照してください。

注意：

- system-jazn-data.xml に対する変更が Application Server Control またはセキュリティ・プロバイダ MBean によって行われた場合、変更を反映するために OC4J を再起動する必要はありません。ただし、変更を手動で行った場合には再起動が必要です（通常はお薦めしません）。
- 前のリリースでは、system-jazn-data.xml に対応するファイルは jazn-data.xml でした。ファイルベース・プロバイダに関しては、ユーザーおよびロール情報の保存に、jazn-data.xml ファイルを引き続き使用できます。ただし、このファイルはアプリケーション固有になります。次の「アプリケーション固有の jazn-data.xml ファイル（オプション）」を参照してください。
- system-jazn-data.xml ファイルの設定は、Application Server Control または OracleAS JAAS Provider Admintool で操作できます。
- system-jazn-data.xml ファイルへの変更は、そのファイルを使用するすべてのアプリケーションから可視です。
- system-jazn-data.xml ファイルには、事前定義された OC4J ユーザー / ロールのアカウントが格納されます。7-13 ページの「system-jazn-data.xml の事前定義済 OC4J アカウント」を参照してください。
- 要素設定では空白が重要な意味を持ちます。次の違いに注意してください。

```
<name>scott</name>
<name>scott </name>
<name> scott</name>
<name> scott </name>
```

アプリケーション固有の jazn-data.xml ファイル（オプション）

ファイルベース・プロバイダを使用する場合は、オプションでユーザーおよびロールのリポジトリとして jazn-data.xml ファイルを使用できます。OC4J 10.1.3.x 実装の場合、このファイルはアプリケーション固有になります。デプロイ先に応じて、orion-application.xml ファイルの <jazn> 要素で場所を指定できます。

```
<jazn provider="XML" location="path/jazn-data.xml">
   ...
</jazn>
```

関連項目：

- 7-11 ページの「アプリケーション固有の jazn-data.xml ファイルの供給」
- ファイルの階層、要素、属性の詳細は、付録 D「OracleAS JAAS Provider 構成ファイル」を参照してください。

orion-application.xml を正確に次の例のように構成してあるが、jazn-data.xml ファイルがアプリケーションとともにパッケージ化されていない場合は、jazn-data.xml ファイルがデプロイ時に作成されます。

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

また、system-jazn-data.xml に関する前項で説明している、リポジトリへの変更に関する永続性モードは、jazn-data.xml にも影響します。

注意：

- アプリケーション固有の jazn-data.xml ファイルは、構成ファイルではなくリポジトリとみなしてください。
- 要素設定では空白が重要な意味を持ちます。次の違いに注意してください。

  ```
  <name>scott</name>
  <name>scott </name>
  <name> scott</name>
  <name> scott </name>
  ```

---

jazn.xml ファイル

jazn.xml ファイルは、OracleAS JAAS Provider 用の OC4J インスタンス・レベルの構成ファイルであり、*ORACLE_HOME*/j2ee/*instance_name*/config ディレクトリに置かれます。このファイルには、インスタンス・レベルのセキュリティ・プロバイダ、ならびにポリシーおよびパーミッション設定のリポジトリを指定します。jazn.xml ファイルの主要な要素は <jazn> 要素です。これはアプリケーション・レベル設定用の orion-application.xml ファイルとほぼ同じ機能を持ちます。

デフォルトでは、jazn.xml は、system-jazn-data.xml をリポジトリとして、jazn.com をデフォルトのレルムとして設定し、ファイルベース・プロバイダを指定します。

```
<jazn provider="XML" location="./system-jazn-data.xml" default-realm="jazn.com">
   ...
</jazn>
```

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、通常、*ORACLE_HOME*/j2ee/home/config ディレクトリに置かれます。このファイルは、OC4J の起動時に読み取られ、OracleAS JAAS Provider ランタイムによって使用されます。有効な jazn.xml ファイルが存在しない場合は、OracleAS JAAS Provider は起動できません。

Oracle Identity Management セキュリティ・プロバイダを使用するために、Application Server Control を使用して OC4J を Oracle Internet Directory インスタンスと関連付けた場合は、ブートストラップ jazn.xml ファイルの <jazn> 要素が Oracle Internet Directory インスタンス用に適切に更新されます。次に例を示します。

```
<jazn provider="LDAP" location="ldap://myoid.oracle.com:389" default-realm="us" >
   ...
</jazn>
```

注意： jazn.xml への変更を有効にするには、OC4J の再起動が必要です。

関連項目：

- ファイルの階層、要素、属性の詳細は、付録 D「OracleAS JAAS Provider 構成ファイル」を参照してください。

オプションでシステム・プロパティを使用し、ブートストラップ jazn.xml ファイルの代替場所を指定できます。OracleAS JAAS Provider は、起動されると、jazn.xml を次の順序で検索し、ファイルの存在を確認した時点で検索を終了します。

1. システム・プロパティ oracle.security.jazn.config で指定されている場所
2. システム・プロパティ java.security.auth.policy で指定されている場所
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 で指定され、通常は *J2EE_HOME*/config と同じ場所）
5. ./config

サンプルの jazn.xml ファイル

サンプルの jazn.xml ファイルを次にあげます。最初のファイルは、ファイルベース・プロバイダ用のデフォルト構成を含んだものです。

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<jazn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
                "http://xmlns.oracle.com/oracleas/schema/jazn-10_0.xsd"
 schema-major-version="10"
 schema-minor-version="0"
 provider="XML"
 location="./system-jazn-data.xml"
 default-realm="jazn.com"
/>
```

次に示すのは、LDAP ベース・プロバイダ用のデフォルト構成を含んだファイルです。

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<jazn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
                "http://xmlns.oracle.com/oracleas/schema/jazn-10_0.xsd"
 schema-major-version="10"
 schema-minor-version="0"
 provider="LDAP"
 location="ldap://myoid.us.oracle.com:389"
/>
```

OC4J の System アプリケーション

OC4J の system アプリケーションは、OC4J 10.1.3.x 実装で定義される内部コンポーネントです。OC4J を最初に起動したときに、OC4J のインスタンスに自動デプロイされます。このアプリケーションが追加された主な理由は、OC4J へのアプリケーションのデプロイまたは再デプロイに関する問題を解決するためです。

system アプリケーションは、アプリケーション階層のルートに置かれ、OC4J 起動時に必要なクラス（これには他のすべてのデプロイ済アプリケーションによってデフォルトでインポートされる共有ライブラリも含まれる）および構成を提供します。これは OC4J の内部コンポーネントとしてのみ使用されます。これに対するアプリケーションのデプロイ、または他のアプリケーションの親としての宣言はできません。（すべてのデプロイ対象アプリケーションのデフォルトの親としての役割は、前の OC4J 実装と同様に、OC4J の default アプリケーションが引き続き受け持ちます。）

デフォルトで、system アプリケーションは、system-jazn-data.xml をリポジトリとして使用し、ユーザーおよびロール設定に関してファイルベース・プロバイダを使用するように構成されます。（これは、MBeans および admin_client.jar など、すべてのシステム・リソースの認可が system-jazn-data.xml 内のエントリに基づいていることを意味します。）この構成を変更することはお薦めしません。

system アプリケーションの場合は、system-application.xml が OC4J 固有のアプリケーション・ディスクリプタであり、デプロイ済アプリケーションに対する orion-application.xml ファイルと同じ機能を持ちます。（default アプリケーションの場合、OC4J 固有のアプリケーション・ディスクリプタは application.xml です。デプロイ済アプリケーションに対する J2EE 標準の application.xml ファイルと混同しないでください。これはデフォルトで、system-jazn-data.xml リポジトリも使用する default アプリケーションを構成します。）これらのファイルは、*ORACLE_HOME*/j2ee/*instance_name*/config ディレクトリにあります。

関連項目：

- OC4J アプリケーション階層と、system アプリケーションおよび default アプリケーションの詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

OC4J アカウントのサマリー

この項では、主要な OC4J アカウントのサマリーを提供します。この項の内容は次のとおりです。

事前定義アカウント

OC4J 10.1.3.x 実装には、Oracle Internet Directory（Oracle Identity Management を使用する場合）またはファイルベース・プロバイダ用のブートストラップ・ユーザーとロールが事前定義され、含まれています。

ファイルベース・プロバイダの場合は、これらのアカウントが事前定義されている場所は `system-jazn-data.xml` ファイルです。Oracle Internet Directory（OID）の場合、アカウントは、OC4J と OID の関連付けプロセスの一環としてデフォルトで自動作成されます。

次の事前定義アカウントは、両方のプロバイダに共通です。

- `oc4jadmin` ユーザー（以前の `admin`）。

  これがデフォルトの管理者アカウントです。Oracle Application Server または OC4J のインストール中に、このアカウントのパスワードを指定する必要があります。パスワードの指定後、このアカウントを使用して Application Server Control コンソールにログインできます。このアカウントは、（OC4J の `system` アプリケーションを介した）管理機能の実行時に、Application Server Control からも使用されます。

  クラスタ環境の場合、`oc4jadmin` 名とパスワードは、クラスタ管理用の資格証明として機能します。クラスタ管理用の資格証明は 1 セットのみであるため、各 OC4J インスタンスにはデフォルトで、同じパスワードを持つ `oc4jadmin` アカウントが必要です。そのため、このアカウントのパスワードを変更する場合には注意が必要です。

  日常の管理業務には、`oc4jadmin` を使用するのではなく、追加の管理アカウントを作成することをお薦めします。

  関連項目：

  - 4-13 ページの「新しい管理者アカウントの作成と構成」

- `oc4j-administrators` ロール（以前の `administrators`）。メンバーとして `oc4jadmin` を含み、RMI パーミッション login および管理パーミッション administration が付与されています（それぞれ、`com.evermind.server.rmi.RMIPermission` および `oracle.security.jazn.policy.AdminPermission`）。
- `oc4j-app-administrators` ロール（以前の `jmx-users`）。RMI パーミッション login が付与されており、JMX のアプリケーション・レベルのコネクタにアクセスできます。
- メンバー `oc4jadmin` を含んだ `ascontrol_admin`（Application Server Control を含めたすべての SOA コントロールの管理ロール）。
- `ascontrol_appadmin`（Application Server Control の必須ロール）。
- `ascontrol_monitor`（Application Server Control の必須ロール）。

重要： 外部 LDAP セキュリティ・プロバイダを使用する場合、（プロバイダに適したプロビジョニング・ツールにより）手動で前述のユーザーとロールのアカウントを作成し、`oc4j-administrators` ロールと `ascontrol_admin` ロールを `oc4jadmin` に付与する必要があります。また、OracleAS JAAS Provider Admintool を使用して前述のパーミッションを付与します。

ファイルベース・プロバイダに対してのみ、次の追加アカウントが事前定義されています。

- `anonymous` ユーザー。初期段階では非アクティブです。

  `anonymous` は、`system-jazn-data.xml` ファイルで直接アクティブにします。これを行うには、そのファイル内で、`<user>` 要素の `deactivated` 属性を `true` から `false` に変更します。`oc4jadmin` とは異なり、OracleAS JAAS Provider Admintool では、`anonymous` のアクティブ化はサポートされていません。

- `users` ロール。EJB/RMI アクセス用です。
- `jtaadmin` ユーザー。トランザクションを ORMI 全体に伝播できます。

この項で説明するアカウントは削除しないでください。削除すると、OracleAS JAAS Provider の管理機能が動作しなくなります。

関連項目：

- 次の「oc4jadmin アカウントのアクティブ化（スタンドアロンの OC4J）」を参照してください。
- Application Server Control のオンライン・ヘルプの oc4jadmin ユーザーの概要、管理ユーザーとロール（Application Server Control のロールの追加情報）、および管理ユーザーとロールを定義する際のベスト・プラクティスに関するトピック。

oc4jadmin アカウントのアクティブ化（スタンドアロンの OC4J）

`oc4jadmin` アカウント（以前の `admin` アカウント）は、Oracle Application Server のインストール時にアクティブになりますが、スタンドアロン OC4J のファイルベースのプロバイダに対しては、最初は非アクティブです。次の状況でアクティブになります。

- スタンドアロン OC4J の最初の起動時（パスワードの入力が必要です）
- OracleAS JAAS Provider Admintool に `-activateadmin` オプションを指定して実行した場合

  このコマンドも、その一部としてパスワードの入力が必要です。

  ```
  % java -jar jazn.jar -activateadmin password
  ```

新しい管理者アカウントの作成と構成

前述のように、デフォルトでは、`oc4jadmin` が OC4J の管理者アカウントになります。この項では、SOA コンポーネント（Application Server Control コンソール や Oracle Web Services Manager など）にアクセスする場合などに代替の管理者アカウントを使用する方法と、OC4J の `system` アプリケーションによる内部使用について説明します。`system` アプリケーションは、Application Server Control により、様々な管理機能（アプリケーションのデプロイとアンデプロイなど）に対して使用されます。

新しい管理者アカウントの作成

`oc4jadmin` を使用するかわりに、新しい管理者アカウントを作成して使用することができます。これにより、Application Server Control コンソールなどの SOA コンポーネントに（通常は Java SSO を介して）アクセスできます。ここでは、このシナリオを想定しています。

実行する手順は、SOA コンポーネントのセキュリティ・プロバイダがファイル・ベース・プロバイダか Oracle Identity Management かによって異なります。いずれのプロバイダにも、必要な RMI パーミッションが設定された主要な管理者ロールがあります（デフォルトで定義）。外部 LDAP プロバイダなど他のセキュリティ・プロバイダの場合、ロールとパーミッションを手動で設定する必要があります。

注意： 次の追加の項目を Application Server Control に適用できます。

- クラスタ環境で Application Server Control が適切に動作するには、クラスタ全体に新しいユーザーを作成する必要があります。
- `oc4jadmin` アカウントを削除すると、Application Server Control のログイン後のページが表示されます。Application Server Control は、以前にキャッシュされた `oc4jadmin` ユーザーにそれ以後接続できなくなるため、これは正常な動作です。新しいユーザー名とパスワードを入力すると、このページは表示されなくなります。

ファイルベース・プロバイダまたは Oracle Identity Management の場合 ファイルベース・プロバイダまたは Oracle Identity Management を SOA コンポーネントのセキュリティ・プロバイダとして使用する場合は、次の手順を実行してカスタム管理者アカウントを使用する必要があります。

1. 新しい管理ユーザー・アカウントを作成します。
2. 新しい管理ユーザー・アカウントに、`oc4j-administrators` ロールと `ascontrol_admin` ロールを付与します。
3. オプションで `admin.user` プロパティを設定し、OC4J の `system` アプリケーションで新しいアカウントを使用できるようにします。4-15 ページの「システム・アプリケーションに対する新しい管理者アカウントの構成」を参照してください。
   `system-jazn-data.xml` をリポジトリとして使用しない場合、`system` アプリケーションで使用するための新しい管理者アカウントを `system-jazn-data.xml` で作成し、必要な管理者ロールを付与する必要もあります。

追加の手順は必要ありません。デフォルトで、`oc4j-administrators` ロールと `ascontrol_admin` ロールは、ファイルベース・プロバイダと Oracle Internet Directory 内に存在し、RMI パーミッションの login が付与されています。

注意：

- ファイルベース・プロバイダの場合、ユーザーおよびロールの管理に Application Server Control コンソールを使用します。7-2 ページの「Application Server Control でのファイルベース・プロバイダの構成」を参照してください。
- Oracle Internet Directory の場合、ユーザーおよびロールの管理に Oracle Delegated Administration Service（DAS）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。

その他のセキュリティ・プロバイダの場合 ファイルベース・プロバイダまたは Oracle Identity Management 以外を SOA コンポーネントのセキュリティ・プロバイダとして使用する場合は、次の手順を実行してカスタム管理者アカウントを使用する必要があります。

1. セキュリティ・プロバイダに対する適切なツールを使用して、新しい管理ユーザー・アカウントを作成します。
2. 管理者ロールをまだ作成していない場合は、適切なツールを使用して、管理者ロールの `oc4j-administrators` と `ascontrol_admin` を作成します。外部 LDAP プロバイダの場合、これらのロールは、LDAP プロバイダ用に構成されたグループ検索ベースの下に存在する必要があります。（グループ検索ベースの詳細は、10-5 ページの表 10-4「Application Server Control 外部 LDAP のロールおよびメンバーのオプション」および 10-8 ページの表 10-7「外部 LDAP のロールおよびメンバーのオプション」を参照してください。）
3. ロールをまだ付与していない場合は、適切なツールを使用して、新しい管理者ユーザー・アカウントに `oc4j-administrators` ロールと `ascontrol_admin` ロールを付与します。
4. 追加の管理者ロールの `oc4j-app-administrators`、`ascontrol_appadmin` および `ascontrol_monitor` を作成します。（これらは管理ユーザーに付与する必要はありません。）

5. 管理者が EJB にアクセスする必要があり、RMI パーミッションの login をまだ付与していない場合は、`oc4j-administrators` ロールと `ascontrol_admin` ロールに RMI パーミッションの login を付与します。この場合、次の例に示すように OracleAS JAAS Provider Admintool を使用できます。

```
% java -jar jazn.jar -grantperm myrealm -role oc4j-administrators \
     com.evermind.server.rmi.RMIPermission login
```

ロールは外部 LDAP プロバイダで定義されていますが、これらのパーミッション付与は `system-jazn-data.xml` ファイルに格納されていることに注意してください。

6. オプションで `admin.user` プロパティを設定し、OC4J の `system` アプリケーションで新しいアカウントを使用できるようにします。次項の「システム・アプリケーションに対する新しい管理者アカウントの構成」を参照してください。この手順の場合、`system` アプリケーションで使用するため `system-jazn-data.xml` に新しい管理アカウントを作成し、必要な管理ロールを付与する必要もあります。

関連項目：

- 10-10 ページの「管理ユーザーとロールの作成および RMI パーミッションの付与」

システム・アプリケーションに対する新しい管理者アカウントの構成

OC4J の `system` アプリケーションで使用するため、次のように別の管理者アカウントを指定できます。

1. 目的のアカウントが存在しない場合、`system-jazn-data.xml` ファイルに目的のアカウントを作成し、`oc4j-administrators` ロールと `ascontrol_admin` ロールを付与します。ユーザーおよびロールの管理に、Application Server Control コンソールを使用します。詳細は、7-2 ページの「Application Server Control でのファイルベース・プロバイダの構成」を参照してください。

2. 次のように、インスタンス・レベルの `jazn.xml` ファイルに `admin.user` プロパティを設定します。

```
<jazn ... >
   ...
   <property name="admin.user" value="desired_admin_user_name" />
   ...
</jazn>
```

（ここに示す手順は、通常のように、OC4J の `system` アプリケーションがファイルベースのセキュリティ・プロバイダを使用するように構成されていることを前提としています。）

指定されたアカウントは、Application Server Control によるデプロイとアンデプロイなど、様々な管理機能で使用されます。

匿名ユーザーの構成

ファイルベース・プロバイダまたは Oracle Identity Management を使用している場合は、インスタンス・レベルの `jazn.xml` ファイルで `anonymous.user` プロパティを設定することで、匿名ユーザーを既存のユーザーにマップできます。たとえば、ユーザー `PUBLIC` が Oracle Internet Directory に存在する場合は、次のようにします。

```
<jazn ... >
   ...
   <property name="anonymous.user" value="PUBLIC" />
   ...
</jazn>
```

構成リポジトリおよびセキュリティ管理ツールのサマリー

管理ツールおよび構成リポジトリに関しては前述していますが、表 4-1 に、各セキュリティ・プロバイダの様々なタイプの構成別に、使用する構成リポジトリおよび推奨する管理ツールを一覧で示します。

使用可能な場合は、Application Server Control が推奨ツールになります。

表 4-1 構成リポジトリおよび推奨管理ツール

セキュリティ・プロバイダ	レルム、ユーザー、ロール用のリポジトリおよび管理ツール	ポリシー用のリポジトリおよび管理ツール	JAAS ログイン・モジュール用のリポジトリおよび管理ツール
ファイルベース	system-jazn-data.xml（またはアプリケーション固有の jazn-data.xml） Application Server Control コンソール（推奨）または OracleAS JAAS Provider Admintool を使用。	system-jazn-data.xml OracleAS JAAS Provider Admintool を使用。	なし
Oracle Identity Management	Oracle Internet Directory DAS（または読取りのみの場合は OracleAS JAAS Provider Admintool）を使用。	Oracle Internet Directory OracleAS JAAS Provider Admintool を使用。	なし
外部 LDAP	外部（サード・パーティ）LDAP サーバー プロバイダ提供のツールを使用。	system-jazn-data.xml OracleAS JAAS Provider Admintool を使用。	system-jazn-data.xml Application Server Control コンソール（推奨）または OracleAS JAAS Provider Admintool を使用。
カスタム・セキュリティ・プロバイダ	カスタム・セキュリティ・リポジトリ プロバイダ提供のツールを使用。	system-jazn-data.xml OracleAS JAAS Provider Admintool を使用。	system-jazn-data.xml Application Server Control コンソール（推奨）または OracleAS JAAS Provider Admintool を使用。
Oracle Access Manager	Oracle Access Manager Policy Manager ツールを使用します。	system-jazn-data.xml OracleAS JAAS Provider Admintool を使用。	system-jazn-data.xml Application Server Control コンソール（推奨）または OracleAS JAAS Provider Admintool を使用。

注意： この表内の「ポリシー」は、サブジェクトベースのポリシーを指しています。コードベースのポリシーは、標準の Java 2 ポリシー・ファイル（`java2.policy` や `java.policy` など）に格納されます。現在 OC4J では、Java 2 ポリシー・ファイルの更新や管理を行う管理ツールは提供されていません。

5

OC4J での認可

第 2 章「Java プラットフォームのセキュリティ」では、3 つの主要な Java セキュリティ・モデルである J2EE ロールベース・セキュリティ、Java 2 コードベース・セキュリティおよび JAAS サブジェクトベース・セキュリティの概要を説明しました。

この章では、これらの各モデルで認可機能を使用する方法について説明します。これらの機能は、単独でも組み合せても使用できます。章の最後に、セキュリティ・モデルの選択方法について説明します。

内容は次のとおりです。

- Java 2 セキュリティおよびコードベースのポリシー管理
- OC4J 環境における認可 API、JAAS モードおよび JACC
- OracleAS JAAS Provider ポリシー管理
- 認可のコーディングと構成
- 認可の方法

Java 2 セキュリティおよびコードベースのポリシー管理

この項では、Java 2（コードベース）セキュリティのセキュリティ・マネージャとポリシー・ファイルについて説明します。この項の内容は次のとおりです。

Java 2 セキュリティ・マネージャおよびポリシー・ファイルの指定

OC4J および基礎となる JDK によって Java 2（コードベース）セキュリティ・ポリシーを強制するには、セキュリティ・マネージャ（`java.lang.SecurityManager` インスタンス）を有効にする必要があります。これにより、たとえば、クラス・ローダーへのアクセス、JDK リソースへのアクセス、JDK API の実行、JAAS API の実行（`Subject.doAs()` または `Subject.doAsPrivileged()` メソッドを実行できるかどうかなど）が影響を受ける可能性があります。セキュリティ・マネージャを有効にすると、Java 2 ポリシー・ファイルに指定されたポリシーによって、実行元のコードがアクセス可能なリソースが判別されます。

セキュリティ・マネージャは、次のいずれかの方法で指定できます。

- `java.lang.System` クラスの静的な `setSecurityManager(SecurityManager)` メソッドをコールして、目的のセキュリティ・マネージャを指定します。
- OC4J の起動時に、システム・プロパティ `java.security.manager` を使用します。このときに、デフォルトのセキュリティ・マネージャを使用する設定を解除するか、または任意のセキュリティ・マネージャを指定する設定を行います。

デフォルト・セキュリティ・マネージャとともに OC4J を起動する例を次に示します。

```
% java -Djava.security.manager ... -jar oc4j.jar
```

次の例は、セキュリティ・マネージャを指定したものです。

```
% java -Djava.security.manager=com.abc.MySecurityManager ... -jar oc4j.jar
```

注意：

- 通常、OC4J ではセキュリティ・マネージャを使用する必要はありません。Oracle Application Server のインストールでは、OC4J インスタンスは、デフォルトではセキュリティ・マネージャなしで実行されます。セキュリティ・マネージャをインストールすると、パフォーマンスに重大な影響が出ることがあります。カスタム・セキュリティ・マネージャを使用する場合は、OC4J の機能に影響しないことを確認してください。
- `AccessController`（Sun JDK 付属）のデフォルトの実装を使用する場合は、5-19 ページの「checkPermission() メソッドの使用」で説明する `AccessController.checkPermission()` をコールすることにより、セキュリティ・マネージャが有効かどうかに関係なく、アプリケーションに対して Java 2 セキュリティ・ポリシーが強制的に適用されます。

デフォルト・セキュリティ・マネージャによって特定のクラスに付与されるパーミッションは、ポリシー・ファイルを読み取ることで決定されます。デフォルト・ポリシー・ファイルが、J2SE の一部として提供されます。目的のポリシー・ファイルを指定するには、システム・プロパティ `java.security.policy` を目的のファイルのフルパスに設定します。次の例では、デフォルトのセキュリティ・マネージャと OC4J 付属のポリシー・ファイルを指定して OC4J を起動しています。

```
% java -Doracle.home=$ORACLE_HOME -Djava.security.manager \
       -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy \
       -jar oc4j.jar
```

注意：

- この java2.policy ファイルには、セキュリティ・マネージャとともに OC4J を実行するために必要なパーミッションが含まれていることに注意してください。別のポリシー・ファイルを使用する場合は、そのファイルに同じパーミッションが含まれていることを確認してください。
- .policy ファイルは、Java 2（コードベース）ポリシー専用です。OracleAS JAAS Provider の場合、JAAS（サブジェクトベース）ポリシーは、system-jazn-data.xml ファイルの <jazn-policy> 要素内または Oracle Identity Management セキュリティ・プロバイダを使用している場合は Oracle Internet Directory 内で宣言されます（5-15 ページの「OracleAS JAAS Provider ポリシー構成」を参照）。

関連項目：

- 関連情報は、この後の「必要な Java 2 パーミッションを判別する際の PrintingSecurityManager の使用」および「Java 2 ポリシー・ファイルの作成または更新」を参照してください。
- セキュリティ・マネージャの概要は、2-11 ページの「Java 2 認可 : セキュリティ・マネージャおよびアクセス・コントローラ」を参照してください。

必要な Java 2 パーミッションを判別する際の PrintingSecurityManager の使用

OC4J 上で動作するアプリケーションに必要となるすべてのパーミッションを識別できるように、Oracle は、セキュリティ例外をスローしないカスタム・セキュリティ・マネージャ PrintingSecurityManager を提供しています。このセキュリティ・マネージャは、デフォルトのセキュリティ・マネージャがスローする例外を指定するメッセージを出力します。また、PrintingSecurityManager は、セキュリティ例外を回避するポリシー権限を生成します。

PrintingSecurityManager を、次の例のように実行します。ここでは、OC4J を *ORACLE_HOME*/j2ee/home から実行するものと想定しています。

```
% java -Xbootclasspath/p:lib/oc4j-psm.jar -Doracle.home=$ORACLE_HOME \
       -Djava.security.manager=oracle.oc4j.security.PrintingSecurityManager \
       -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy \
       -jar oc4j.jar
```

（-Xbootclasspath により、PrintingSecurityManager をブート・クラスパスに追加することで、このセキュリティ・マネージャがすべてのパーミッション付きで実行されます。）

PrintingSecurityManager により、次の情報をリストした出力が生成されます。

- どのコードソースでどのパーミッションが必要になるか
- ポリシー・ファイルにコピー・アンド・ペースト可能なポリシー権限

デフォルトでは、出力先は System.out ですが、次のシステム・プロパティで出力ファイルを指定できます。1 番目のプロパティでは不足しているパーミッションに関するメッセージ、2 番目のプロパティではポリシー権限に関するメッセージの出力ファイルを指定します。

```
-Doracle.oc4j.security.manager.printing.file=filenamepath
-Doracle.oc4j.security.manager.printing.generated.grants.file=filenamepath
```

注意： PrintingSecurityManager は OC4J とは無関係のため、OC4J の外部でも使用できます。

Java 2 ポリシー・ファイルの作成または更新

Java 2 ポリシー・ファイルは、信頼コードやアプリケーションにパーミッションを付与することで、それらが環境内で正しく動作するための適切なアクセス権を割り当てます。

事前構成済の Java 2 ポリシー・ファイル `java2.policy` は、*ORACLE_HOME*`/j2ee/home/config` に格納されています。このファイルは、必要に応じて変更できます。または、代替のポリシー・ファイルを作成して指定することもできます。ただし、Oracle 提供の `java2.policy` ファイルには、セキュリティ・マネージャとともに OC4J を実行する際に必要なパーミッションが含まれていることに注意してください。別のポリシー・ファイルを使用する場合は、そのファイルに同じパーミッションが含まれていることを確認してください。

次のポリシー・ファイルの例では、システム・ファイルを開く操作や、ソケットやポートを開く操作に対するすべてのパーミッションを、トラステッド `jazn.jar`（OracleAS JAAS Provider Admintool）に付与しています。

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

同様に次の例では、すべてのパーミッションを `wssecurity.jar`（Web サービスのセキュリティ機能用）に付与しています。

```
/* grant the WSSecurity AllPermission */
grant codebase "file:${oracle.home}/webservices/lib/wssecurity.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 $ORACLE_HOME/appdemo/, */
/* Grant JAAS permissions to the demo to run JAAS APIs*/
grant codebase "file:/${oracle.home}/appdemo/-" {
   permission oracle.security.jazn.JAZNPermission "getPolicy";
   permission oracle.security.jazn.JAZNPermission "getRealmManager";
   permission oracle.security.jazn.policy.AdminPermission;
}
```

ユーザーのアプリケーション・コードや OC4J に生成されたクラスに対し、必要に応じて追加のパーミッションを付与できます。そのためには、`.policy` ファイルに手動で追加のエントリを作成します。（現在、この操作を行うためのツールはありません。）必要なパーミッションは、アプリケーションの詳細に依存し、必要なコードベースは、インストールの詳細に依存します。

注意： `${oracle.home}` を使用して、*ORACLE_HOME* の場所を指定している点に注意してください。この環境変数は、デフォルトで適切に設定されます。

関連項目：

- ポリシー・ファイルの構文の詳細は、次の URL を参照してください。

  `http://java.sun.com/j2se/1.5.0/docs/guide/security/PolicyFiles.html`

OC4J 環境における認可 API、JAAS モードおよび JACC

この項では、OC4J の次の認可機能について説明します。

- JAAS 認可および OracleAS JAAS Provider の JAAS モード
- OC4J コンテナへのサブジェクトの設定
- Java Authorization Contract for Containers の実装

JAAS 認可および OracleAS JAAS Provider の JAAS モード

OracleAS JAAS Provider により、Java パーミッションを使用して任意の保護リソースをモデル化できます。Java パーミッション・モデル（および関連 `java.security.Permission` クラス）は拡張可能であり、密なアクセス制御を柔軟に定義できます。

OracleAS JAAS Provider は J2SE 標準に基づいており、標準の API を使用して密な認可に関連する次の機能をサポートします。

- JAAS モード。このモードは、Web アプリケーションまたは EJB の標準的な `Subject.doAs()` の機能および `Subject.doAsPrivileged()` の機能と関連があります。
- OracleAS JAAS Provider のレルムおよびポリシーの API 機能。
- パーミッションを付与する機能。
- パーミッションをチェックする機能。

注意： ここで説明する API は、`system-jazn-data.xml` または Oracle Internet Directory とともに、ポリシー・リポジトリとして使用できます。

関連項目：

- 関連するタスク用の手順および例の詳細は、5-17 ページの「認可のコーディングと構成」を参照してください。
- 標準的な JAAS プログラミング参照情報は、次の URL を参照してください。

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

- JAAS API の詳細情報は、次の URL の `java.security.*` パッケージおよび `javax.security.auth.*` パッケージを参照してください。

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

- パーミッションの詳細は、次の URL を参照してください。

  ```
  http://java.sun.com/j2se/1.5.0/docs/guide/security/permissions.html
  ```

- OracleAS JAAS Provider の『Oracle Containers for J2EE Security Java API Reference』（Javadoc）

JAAS モードの概要

OC4J 10.1.3.x 実装では、JAAS モードと呼ばれる密な認可機能が提供されます。この機能は、Subject クラスに属する静的メソッド doAs() および doAsPrivileged() の標準機能と関連しています（2-16 ページの「JAAS 認可 : サブジェクト・メソッド doAs() および doAsPrivileged()」を参照）。

これらのメソッドは、アプリケーションでは JAAS モードの設定に従って使用されます。JAAS モードは、アプリケーションの orion-application.xml ファイルに含まれる <jazn> 要素の jaas-mode 属性で設定します。JAAS モードでは、doAs() または doAsPrivileged() のどちらを使用するかは次のように決定されます。

- 設定が jaas-mode="doAs" となっている場合は、アプリケーション・モジュール（Web モジュールおよび EJB）が OC4J によって Subject.doAs() ブロック内で実行されます。

  このモードは、サブジェクトベース・セキュリティとともにコードベース・セキュリティが必要な場合に便利です。

- 設定が jaas-mode="doAsPrivileged" となっている場合は、アプリケーション・モジュールが Subject.doAsPrivileged() ブロック内で null アクセス制御コンテキストを使用して実行されます。

  このモードは、サブジェクトベース・セキュリティのみが必要な場合に便利です。

- jaas-mode="null"（デフォルト）と設定している場合は、どちらのメソッドも使用されずにモジュールが実行されます。

  このモードは、コードベース・セキュリティのみが必要な場合に便利です。

jaas-mode はアプリケーション・レベルの orion-application.xml ファイルで設定されるので、アプリケーション内のすべての Web モジュールまたは EJB に影響します。

重要 :

- jaas-mode は orion-application.xml のみで設定し、jazn.xml では設定しません。
- Application Server Control を介して任意の時点で任意のアプリケーションに対してファイルベース・プロバイダから Oracle Identity Management に切り替えると、そのアプリケーションの orion-application.xml 内にある <jazn> 要素が次のように置き換えられます。以前の <jazn> の設定はすべて失われるため、設定をやりなおす必要があります。

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

注意 : JAAS モードは、以前のリリースで orion-application.xml または orion-web.xml の <jazn-web-app> 要素に含まれていた runas-mode および doasprivileged-mode 設定に取ってかわるものです。

これらの設定は、OC4J 10.1.3.x 実装では非推奨ですが、下位互換性を維持するために引き続きサポートされています。

設定 jaas-mode="null" は runas-mode="false" と等価であり、jaas-mode="doas" は doasprivileged-mode="false" 付きの runas-mode="true" と等価であり、jaas-mode="doAsPrivileged" は doasprivileged-mode="true" 付きの runas-mode="true" と等価です。

関連項目：

- アクセス・コントローラおよびアクセス制御コンテキストの概要は、2-11 ページの「Java 2 認可 : セキュリティ・マネージャおよびアクセス・コントローラ」を参照してください。
- 5-20 ページの「JAAS モードの構成と使用」

OracleAS JAAS Provider のレルム API とポリシー API

この項では、JAAS 認可と関連する OracleAS JAAS Provider のクラスとメソッドについて説明します。

`oracle.security.jazn.JAZNConfig` クラスのインスタンスは、`<jazn>` 要素の構成を表します。このクラスには、次のメソッドが含まれています。

- `JAZNConfig getJAZNConfig()`

  `JAZNConfig` クラスのこの静的メソッドでは、`JAZNConfig` インスタンスが戻されます。

- `RealmManager getRealmManager()`

  `JAZNConfig` クラスのこのインスタンス・メソッドでは、レルムの管理に使用される `RealmManager` インスタンスが戻されます。

`oracle.security.jazn.realm.RealmManager` クラスには、次のインスタンス・メソッドが含まれています。

- `Realm getRealm(String)`

  このメソッドでは、指定したレルム名のレルム・オブジェクトが戻されます。

`oracle.security.jazn.realm.Realm` クラスのインスタンスでは、特定レルムのユーザーおよびロールのストアへのアクセスが提供されます。`realm` パッケージでは、ユーザー管理は `UserManager` インタフェース、ロール管理は `RoleManager` インタフェースによって定義されます。`Realm` クラスには、次のインスタンス・メソッドが含まれています。

- `UserManager getUserManager()`

  このメソッドでは、`UserManager` インスタンスが戻されます。このインスタンスは、このレルムのユーザーの管理に使用できます。

- `RoleManager getRoleManager()`

  このメソッドでは、`RoleManager` インスタンスが戻されます。これは、このレルムのロールの管理に使用できます。

レルム内のユーザーの管理（追加、取得、削除）には、`oracle.security.jazn.realm.UserManager` インスタンスを使用します。このインタフェースには、次のメソッドが含まれています。

- `RealmUser getUser(String)`

  このメソッドでは、レルムに含まれる指定名ユーザーのユーザー・オブジェクトが戻されます。

関連項目：

- 現在利用可能で、かつ今後のリリースでここに説明した API の一部に取ってかわる予定の新しいユーザー API とロール API の詳細は、第 12 章「ユーザーおよびロール API フレームワーク」を参照してください。

パーミッションの付与または取消しを行う OracleAS JAAS Provider API

前述の項で説明した JAZNConfig クラスには、次のメソッドもあります。

- JAZNPolicy getPolicy()

  このメソッドでは、oracle.security.jazn.policy.JAZNPolicy インスタンスが戻されます。このインスタンスは、JAAS（サブジェクトベース）認可ポリシーのリポジトリを表しています。

JAZNPolicy インタフェースには、次のメソッドが含まれています。

- void grant(Grantee, Permission)

  このメソッドでは、oracle.security.jazn.policy.Grantee インスタンスおよび java.security.Permission インスタンスが入力として使用されて、指定された受領者に指定のパーミッションが付与されます。

- void revoke(Grantee, Permission)

  このメソッドでは、指定された受領者の指定パーミッションが取り消されます。

- boolean hasPermission(Grantee, Permission)

  このメソッドでは、指定された受領者が指定パーミッションを持っているかどうかが判別されます。

Grantee クラスには、Principal インスタンスを入力として取るコンストラクタが含まれています。

- new Grantee(Principal)

表 5-1 に、Oracle 提供のパーミッション・クラスを示します。

注意： ロールにパーミッションを付与する機能は、非推奨の com.evermind.security.Group クラスの機能に取ってかわるものです。

関連項目：

- これらのクラスの詳細は、OracleAS JAAS Provider の『Oracle Containers for J2EE Security Java API Reference』（Javadoc）を参照してください。

表 5-1 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。

表 5-1 OracleAS JAAS Provider のパーミッション・クラス（続き）

パーミッション	パッケージ部分	説明
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 および dropRealm など）を表します。java.security.Permission の拡張であり、通常の Java パーミッションと同様に使用されます。RealmPermission インスタンスでは、レルム名（ターゲット名）がアクションのリストと関連付けられます。
RMIPermission	com.evermind.server.rmi	このパーミッションは、ORMI プロトコルを介して EJB にアクセスする際に必要です。通常は、RMIPermission login（OracleAS JAAS Provider Admintool からなど）のように使用します。

これらのパーミッション・クラスのインスタンスは、次のようにして構築します。

- new AdminPermission(String)
- new AdminPermission(Permission)

  AdminPermission コンストラクタは、エンコードされたパーミッション文字列や java.security.Permission インスタンスを取ります。

- new RoleAdminPermission(String)
- new RoleAdminPermission(RealmRole)

  RoleAdminPermission コンストラクタは、次のいずれかを取ります。

  – 管理パーミッションが付与されるロールの名前を *realmname/rolename* という形式で指定する文字列。* のみを指定すると、システム内のすべてのロールが対象になり、*realm*/* を指定すると、このレルム内のすべてのロールが対象となります。

  – 特定のレルムに関連付けられているロールを表す oracle.security.jazn.realm.RealmRole インスタンス。これは、RealmRole インタフェースを実装するクラスのインスタンスです。

- new JAZNPermission(String)

  JAZNPermission コンストラクタは、パーミッションのシンボリック名を取ります（getRealmManager、getPolicy、getProperty.*propertyname* など）。

- new RealmPermission(String realm, String actions)

  RealmPermission コンストラクタは、レルム名の文字列と、レルムに適用されるアクションのカンマ区切りリストで構成される文字列を取ります。

- new RMIPermission(String param, String actions)

  RMIPermission コンストラクタは、RMIPermission パラメータの文字列（login など）と、適用されるアクションのカンマ区切りリストで構成される文字列を取ります。

標準的なパーミッション・クラスのインスタンスも、必要に応じて構成できます。

- `new Permission(String permname)`

  `java.security.Permission` コンストラクタでは、パーミッションの名前を指定するために文字列を取ります。

- `new BasicPermission(String permname)`

  `java.security.BasicPermission` コンストラクタは、パーミッションの名前を指定するために文字列を取ります。

- `new FilePermission(String path, String actions)`

  `java.security.FilePermission` コンストラクタは、該当ファイルのパスを指定するために 1 つの文字列を取り、許容アクションのカンマ区切りリストとしてもう 1 つの文字列を取ります。サポートされているアクションは、read、write、execute および delete です。

- `new AllPermission()`

  `java.security.AllPermission` クラスのインスタンスは、その他すべてのパーミッションを表すパーミッションです。このインスタンスのコンストラクタにはパラメータはありません。

重要： `AllPermission` は必要な場合のみ使用し、使用する際には十分に注意する必要があります。

注意： JAAS には、パーミッションのチェック用標準メカニズム（次の項で説明）はありますが、ユーザーの管理用およびパーミッションの付与用の標準メカニズムは用意されていません。このために、この項で説明しているクラスおよびメソッドは Oracle 固有となっています。

パーミッションをチェックする API

パーミッションのチェックなど、アクセス制御の概要は、2-11 ページの「Java 2 認可 : セキュリティ・マネージャおよびアクセス・コントローラ」を参照してください。パーミッションの取得とチェックに関係する可能性がある API は、次のとおりです。

`java.security.AccessController` クラスには、次のメソッドが含まれています。

- `AccessControlContext getContext()`

  このメソッドは、現行スレッドの継承されたアクセス制御コンテキストを含んでいる現在の呼び出しコンテキストを検査し、そのコンテキストを `java.security.AccessControlContext` インスタンスとして表します。

- `void checkPermission(Permission)`

  `java.security.Permission` インスタンスの場合、この静的メソッドは、そのパーミッションによって指定されるアクセス・リクエストを許可するかどうかをチェックし、リクエストを拒否する必要がある場合に `AccessControlException` をスローします。このメソッドは、デフォルトのアクセス制御コンテキストを使用します。

  このメソッドを使用すると、セキュリティ・マネージャを有効にすることなく、Java 2 ポリシーをチェックできます。

java.security.AccessControlContext クラスには、次のメソッドが含まれています。

- void checkPermission(Permission)

  このメソッドは、AccessController.checkPermission() メソッドと同じ機能を備えていますが、そのアクセス制御コンテキストを使用する際には特定の AccessControlContext インスタンス上でコールされます。(AccessControlContext インスタンスは、ProtectionDomain インスタンスの配列から構築できます。)

  このメソッドを使用すると、セキュリティ・マネージャを有効にすることなく、Java 2 ポリシーをチェックできます。

java.lang.SecurityManager クラスには、次のメソッドが含まれています。

- void checkPermission(Permission)

  このメソッドは、AccessController クラスと AccessControlContext クラスの checkPermission() メソッドと同じ機能を備えていますが、セキュリティ・マネージャを有効にした状態で使用され、セキュリティ・マネージャ・インスタンスに対してコールされます。また、アクセスを拒否する必要がある場合、SecurityException をスローします。

javax.security.auth.Subject クラスには、次のメソッドが含まれています。

- Subject getSubject(AccessControlContext)

  この静的メソッドでは、指定のアクセス制御コンテキストに関連付けられたサブジェクトが戻されます。デフォルトのアクセス制御コンテキストは、次のように指定できます。

  ```
  mysubject = Subject.getSubject(AccessController.getContext());
  ```

抽象クラス javax.security.auth.Policy には、次のメソッドが含まれています。

- Policy getPolicy()

  この静的メソッドでは、Policy インスタンスが戻されます。

- PermissionCollection getPermissions(Subject, CodeSource)

  javax.security.auth.Subject インスタンス、java.security.CodeSource インスタンスまたはその両方（入力は null にできます）の場合、このメソッドは、許容されるパーミッションのセットを示す java.security.PermissionCollection インスタンスを戻します。codesource フィールドは null にできます。

PermissionCollection クラスには、次のメソッドが含まれています。

- boolean implies(Permission)

  この抽象メソッドでは、指定したパーミッションがパーミッション・コレクション内のパーミッション・セットによって暗黙的に定義されるかどうかが示されます。たとえば、パーミッション・コレクションがサブジェクトのパーミッションで構成される場合は、次の例のように、指定のパーミッションがサブジェクトにあるかどうかがこのメソッドによって通知されます。

  ```
  jaaspolicy = javax.security.auth.Policy.getPolicy();
  jaaspolicy.getPermissions(mysubject,null).implies(perm);
  ```

注意： javax.security.auth.Policy クラスは JDK 1.4 では非推奨ですが、OC4J 10.1.3.x 実装では完全サポートされており、Sun 社の JDK および J2SE では引き続きサポートされています。このリリースでは、OC4J 自体は java.security.Policy クラスをサポートしていません。

関連項目：

- コードソースの概要は、2-11 ページの「Java 2 認可 : Java 2 セキュリティ・ポリシー」を参照してください。

OC4J コンテナへのサブジェクトの設定

JAAS ベースの認証は、Java EE と十分に統合されていません。JAAS 認証では、通常、ログイン・モジュールを使用する必要があります。ただし、ログイン・モジュール（カスタム・ログイン・モジュールを含む）では、コンテナにサブジェクトがアサートされません。この動作により、アプリケーションでは認証サブジェクトが使用されません。この問題を解決するために、OC4J では API が提供されます。この API を使用すると、コンテナ管理セキュリティ（宣言によるセキュリティ）のかわりに使用できる OC4J コンテナへのサブジェクトのアサートが可能になります。

この API は、`oracle.oc4j.security` パッケージに含まれており、次のように使用されます。

```
Security.setSubject(Subject subject, Longevity longevity);
```

`setSubject` メソッドにより、このリクエストおよび Web コンテナに関連付けられているその他のすべてのリクエストについて、サブジェクト・アイデンティティがコンテナにアサートされます。現在のアプリケーション・コンテキストに Web コンポーネントが含まれていない場合、現在のアプリケーション・サーバーのスレッド・コンテキストのみによってサブジェクトが活用され、プールへ戻される際に消去されます。この API は、クライアント側のアプリケーション・コードには関連せず、サーバー側の実行のみに関連します。

- `subject`: コンテナへアサートするためのアイデンティティ。
- `longevity`: アイデンティティ・アサーションが保持される時間。`Longevity.REQUEST` または `Longevity.SESSION` を使用できます。

Java Authorization Contract for Containers の実装

OC4J 10.1.3.x 実装では、JSR-115 の規約に従って、Java Authorization Contract for Containers（JACC）がサポートされます。これは、コンテナと認可サービス・プロバイダの間の規約で、認可をコンテナから切り離せるようにします。OC4J の認可機能は、標準の JACC プロバイダに委任されています。

JACC 規約に準拠するため、J2EE セキュリティ制約は Java 2 パーミッションに変換され、J2EE セキュリティ・モデルで J2SE セキュリティ・モデルを完全に利用できるようになります。ただし、JACC は、J2EE セキュリティ API のみでなく既存の J2EE の宣言によるセキュリティ・モデルも引き続き完全に保持しています。基本的には、JACC を有効にすることで、アプリケーションで使用する認可の構成とコードを変更することなく、J2EE 認可の拡張バージョンを使用できます。

通常、JACC は開発時ではなくデプロイ時に考慮されます。

JSR-115 に定義されている規約は、アプリケーション・サーバー・コンテナ、デプロイメント・ツールおよびポリシー・プロバイダとの関連で、次のサブ規約に分割されます。

- プロバイダ構成に関するサブ規約
- ポリシー構成に関するサブ規約
- ポリシーの決定および強制に関するサブ規約

JACC では、J2EE 認可モデルに準拠した新しい `java.security.Permission` クラス実装が用意されています。JACC では、コンテナによるアクセスの決定は、これらの `Permission` クラスのインスタンスでの操作に従って行われます。JACC では、J2EE 認可モデルの要件に対応するためにポリシー・プロバイダが新しいパーミッション・クラスを活用する方法が、次のように定義されています。

- パーミッションのコレクションとしてのロールの定義
- ロールのパーミッションのプリンシパルへの付与
- プリンシパルにロールのパーミッションが付与されているかどうかの判別

関連項目：

- 5-21 ページの「Java Authorization Contract for Containers の有効化」
- JACC に関する一般情報は、次の URL を参照してください。

  ```
  http://java.sun.com/j2ee/javaacc/
  http://www.jcp.org/en/jsr/detail?id=115
  ```

OracleAS JAAS Provider ポリシー管理

サブジェクトベースのポリシー管理には規定の標準がないため、実装は各ベンダーに任されています。この項では、サブジェクトベースのポリシー管理に対する OracleAS JAAS Provider の機能の使用方法として、パーミッションの付与、結果の構成、パーミッションのチェック、OracleAS JAAS Provider ポリシー・プロバイダの明示的な指定（必要な場合）について説明します。

- OracleAS JAAS Provider Admintool を介したパーミッションの付与
- OracleAS JAAS Provider ポリシー管理 API の使用
- OracleAS JAAS Provider ポリシー構成
- Oracle ポリシー・プロバイダの指定

関連項目：

- コードベースのポリシーの詳細は、5-2 ページの「Java 2 セキュリティおよびコードベースのポリシー管理」を参照してください。

OracleAS JAAS Provider Admintool を介したパーミッションの付与

OracleAS JAAS Provider Admintool では、`system-jazn-data.xml` または Oracle Internet Directory をポリシー・リポジトリとして、`grantperm`、`revokeperm`、`listperm` の各コマンドを使用して、パーミッションを付与、取消し、リスト表示を行うことができます。（または、実行時に次項の「OracleAS JAAS Provider ポリシー管理 API の使用」で説明する API を使用する方法もあります。）これらのコマンドの完全な構文については、C-15 ページの「パーミッションの付与と取消し」および C-17 ページの「パーミッションのリスト表示」を参照してください。

パーミッションを付与する対象は、ユーザー（`grantperm -user` オプションを使用）、ロール（`-role` オプションを使用）、プリンシパルです。

たとえば、RMI パーミッション login をレルム `myrealm` 内のロール `developers` に付与するには、次のように入力します。

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

（RMI パーミッション login は、EJB/RMI アクセスを許可するために頻繁に付与する必要があるパーミッションです。）

同じパーミッションをユーザー `JDOE_ENDUSER` に付与するには、次のように入力します。

```
% java -jar jazn.jar -grantperm myrealm -user JDOE_ENDUSER \
     com.evermind.server.rmi.RMIPermission login
```

このパーミッションを LDAP プリンシパル `hobbes` に付与する（外部 LDAP プロバイダを使用する場合）には、次のように入力します。

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.LDAPPrincipal hobbes \
       com.evermind.server.rmi.RMIPermission login
```

ファイル sample.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
     sample.txt read,write
```

grantperm コマンドによって生成される構成については、5-15 ページの「OracleAS JAAS Provider ポリシー構成」を参照してください。

関連項目：

- パーミッションの付与によって生成される構成の詳細は、5-15 ページの「OracleAS JAAS Provider ポリシー構成」を参照してください。

OracleAS JAAS Provider ポリシー管理 API の使用

system-jazn-data.xml または Oracle Internet Directory をポリシー・リポジトリとして、OracleAS JAAS Provider ポリシー管理 API を使用してパーミッションの付与または取消しを行うことができます。（または、非実行時に前項の「OracleAS JAAS Provider Admintool を介したパーミッションの付与」で説明した OracleAS JAAS Provider Admintool のコマンドを使用する方法もあります。）次の例では、5-18 ページの「J2EE 認可 API の使用」で示した doGet() メソッドを拡張し、OracleAS JAAS Provider ポリシー管理 API を使用してユーザーにファイル権限を付与します。

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
   ServletOutputStream out = response.getOutputStream();

   response.setContentType("text/html");
   out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
   out.println("Time stamp: " + new Date().toString());
   out.println("request.isUserInRole('ar_developers') = " +
               request.isUserInRole("ar_developers") + "<br>");
   try {
   //Grant Permissions to a user developer

   //get JAZNConfiguration related info
      JAZNConfig jc = JAZNConfig.getJAZNConfig();

   //create a Grantee for "developer"
      RealmManager realmmgr = jc.getRealmManager();
      Realm realm = realmMgr.getRealm("jazn.com");
      UserManager userMgr = realm.getUserManager();
      final RealmUser user = userMgr.getUser("developer");

   //grant developer file permission
      JAZNPolicy policy = jc.getPolicy();
      if ( policy != null) {
         Grantee gtee = new Grantee( (Principal) user);
         java.io.FilePermission fileperm =
                                 new java.io.FilePermission("foo.txt", "read");
         policy.grant( gtee, fileperm);
      }
   } catch (Exception e) { /* print stack trace */ }

   out.println("</BODY>");
   out.println("</HTML>");
}
```

関連項目：

- 5-7 ページの「OracleAS JAAS Provider のレルム API とポリシー API」
- 5-8 ページの「パーミッションの付与または取消しを行う OracleAS JAAS Provider API」
- 5-10 ページの「パーミッションをチェックする API」
- パーミッションの付与によって生成される構成の詳細は、次項の「OracleAS JAAS Provider ポリシー構成」を参照してください。

OracleAS JAAS Provider ポリシー構成

この項では、これまでの各項で説明したように、OracleAS JAAS Provider Admintool またはポリシー管理 API を使用してパーミッションを付与した結果生成されるサブジェクトベースのポリシー構成について説明します。OC4J では、この構成がセキュリティ・プロバイダに応じて `system-jazn-data.xml` ファイルまたは Oracle Internet Directory に配置されます。

内容は次のとおりです。

- jazn.xml でのポリシー・リポジトリ設定
- system-jazn-data.xml でのポリシー構成
- Oracle Internet Directory でのポリシー構成

注意： ここで説明する構成は、サブジェクトベース・セキュリティ専用です。コードベース・セキュリティは、Java 2 ポリシー・ファイル（`.policy`）で構成されます（5-2 ページの「Java 2 セキュリティ・マネージャおよびポリシー・ファイルの指定」および 5-4 ページの「Java 2 ポリシー・ファイルの作成または更新」を参照）。

---

jazn.xml でのポリシー・リポジトリ設定

OC4J インスタンスのセキュリティ・ポリシーに使用されるポリシー・リポジトリは、`jazn.xml` ファイルで指定されたプロバイダです。これは、次のように `<jazn>` 要素の `provider` 設定で指定されます。

- ポリシー構成に `system-jazn-data.xml` を使用する場合は、`provider="XML"`
- ポリシー構成に Oracle Internet Directory を使用する場合は、`provider="LDAP"`

デフォルトでは、`jazn.xml` の `provider` が `XML` に設定され、`system-jazn-data.xml` がポリシー・リポジトリとして使用されます。Oracle Identity Management を使用し、Oracle Internet Directory インスタンスを OC4J インスタンスに関連付けると、`jazn.xml` のプロバイダ設定が `LDAP` に変更され、Oracle Internet Directory がポリシー・リポジトリとして使用されます。

（同様に、`orion-application.xml` の `provider` 設定の場合、アプリケーションのセキュリティ・プロバイダが指定されます。ファイルベースのプロバイダの場合、これが `XML` に設定され、Oracle Identity Management の場合は `LDAP` に設定されます。また、規則上、外部 LDAP プロバイダ、カスタム・ログイン・モジュールまたは Oracle Access Manager の場合も、`XML` に設定されます。）

注意： `jazn.xml` でのポリシー・リポジトリ構成（`<jazn>` 要素の `provider` 属性や `location` 属性の設定など）は、OC4J インスタンス・レベルの構成です。アプリケーションを OC4J インスタンスにデプロイする場合にアプリケーションで別のプロバイダを構成すると、`orion-application.xml` で構成されたプロバイダが認証に使用されるアイデンティティ・ストアになり、`jazn.xml` で指定されたプロバイダが認可に使用されるポリシー・ストアになるという混在した状態になります。

---

system-jazn-data.xml でのポリシー構成

ファイルベースのプロバイダ、Oracle Access Manager、外部 LDAP プロバイダ、カスタム・ログイン・モジュールの場合、ポリシー構成が system-jazn-data.xml ファイルの <jazn-policy> 要素に配置されます。

例として、5-13 ページの「OracleAS JAAS Provider Admintool を介したパーミッションの付与」で使用した grantperm の例の 1 つをもう一度示します。

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

これにより、次の例のような <jazn-policy> 構成が生成されます。

```
<jazn-data>
   ...
   <jazn-policy>
     <grant>
       <grantee>
         <principals>
           <principal>
             <realm-name>myrealm</realm-name>
             <type>role</type>
             <class>oracle.security.jazn.XMLRealmRole</class>
             <name>developers</name>
           </principal>
         </principals>
       </grantee>
       <permissions>
         <permission>
           <class>com.evermind.server.rmi.RMIPermission</class>
           <name>login</name>
         </permission>
       </permissions>
     </grant>
     ...
   </jazn-policy>
   ...
</jazn-data>
```

Oracle Internet Directory でのポリシー構成

Oracle Identity Management（LDAP ベースのプロバイダ）の場合、ポリシー構成は Oracle Internet Directory に配置されます。system-jazn-data.xml と同様に、Oracle Internet Directory に保存されているポリシー情報には、OracleAS JAAS Provider Admintool またはポリシー管理 API を介してアクセスできます。

Oracle ポリシー・プロバイダの指定

Oracle Application Server に同梱の Java Virtual Machine を使用する場合は、OracleAS JAAS Provider 付属の JAAS ポリシー・プロバイダが、OC4J で使用するポリシー・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（つまり OC4J の外部でアプリケーションを実行する場合）は、ポリシー・プロバイダとして Oracle JAAS ポリシー・プロバイダ `oracle.security.jazn.spi.PolicyProvider` を明示的に指定する必要があります。（デフォルトでは、Oracle 以外の JVM によって Sun 社の JAAS Provider が使用されます。）

Oracle 固有の JAAS プロパティ（ポリシー・プロバイダなど）をセキュリティ・プロパティ・ファイルに指定し、OC4J の起動時に JVM に渡すことができます。Oracle が提供するデフォルトのセキュリティ・プロパティ・ファイル *ORACLE_HOME*`/j2ee/home/config/jazn.security.props` により、OC4J で使用するポリシー・プロバイダとして `oracle.security.jazn.spi.PolicyProvider` 指定されます。構成は次のとおりです。

```
auth.policy.provider=oracle.security.jazn.spi.PolicyProvider
```

前述の Oracle JAAS ポリシー・プロバイダの指定を含めたデフォルトの Oracle 固有セキュリティ・プロパティ設定を既存のセキュリティ・プロパティに追加する場合は、次のように `java.security.properties` システム・プロパティを設定します。

```
-Djava.security.properties=ORACLE_HOME/j2ee/home/config/jazn.security.props
```

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は、次のようにします。等号が 2 つ（==）であることに注意してください。

```
-Djava.security.properties==ORACLE_HOME/j2ee/home/config/jazn.security.props
```

認可のコーディングと構成

この項では、実行時に認可をチェックする次の手順について説明し、その例を示します。

1. J2EE 認可 API の使用
2. サブジェクトの取得
3. checkPermission() メソッドの使用
4. JAAS モードの構成と使用
5. Java Authorization Contract for Containers の有効化（J2EE 認可のオプションの拡張機能）

ここで示すサンプルではサーブレット・メソッドを使用しますが、その基本機能は EJB の基本機能と同じです。

関連項目：

- 詳細な例は、付録 B「OracleAS JAAS Provider のサンプル」を参照してください。

J2EE 認可 API の使用

次のサンプル・サーブレットの doGet() メソッドでは、標準の J2EE 認可メソッドを使用してユーザーとプリンシパルが取得され、ユーザーが指定のロールに定義されているかどうかが判別されます。

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    ServletOutputStream out = response.getOutputStream();

    response.setContentType("text/html");
    out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
    out.println("Time stamp: " + new Date().toString());
    out.println("request.isUserInRole('ar_developers') = " +
                request.isUserInRole("ar_developers") + "<br>");
    out.println("</BODY>");
    out.println("</HTML>");
}
```

関連項目：

- 2-4 ページの「関連する Web アプリケーション API」
- 2-6 ページの「関連する EJB API」

サブジェクトの取得

コード内で認可を実行するには、多くの場合、リソースにアクセスしようとする認証対象ユーザーの Subject インスタンスを取得する必要があります。そのためには、標準の JAAS 機能を介して静的な Subject.getSubject() メソッドを使用し、アクセス制御コンテキストを指定します。

通常は、デフォルトのアクセス制御コンテキストを使用します。

```
mysubject = Subject.getSubject(AccessController.getContext());
```

または、特定のアクセス制御コンテキスト（一連の保護ドメインから構築したものなど）を指定することもできます。

```
mysubject = Subject.getSubject(acc);
```

JAAS モードを使用すると、OC4J によって認証済サブジェクトとそのパーミッションがデフォルトのアクセス制御コンテキストとそのパーミッションに関連付けられることに注意してください。

関連項目：

- 5-10 ページの「パーミッションをチェックする API」
- 5-20 ページの「JAAS モードの構成と使用」

checkPermission() メソッドの使用

通常、認可コードでは checkPermission(Permission) コールを使用します。このメソッドでは、現在有効なセキュリティ・ポリシーに基づいて、指定のパーミッションで指定されるアクセス・リクエストを許可するかどうかがチェックされ、許可しない場合は例外がスローされます。

このメソッドは、AccessController クラスと AccessControlContext クラスで使用できます。デフォルト実装（Sun JDK 付属）を使用する場合、セキュリティ・マネージャが有効かどうかに関係なく、いずれかのクラスの checkPermission() メソッドにより、アプリケーション内で Java 2（コードベース）ポリシーを強制できます。

セキュリティ・マネージャを使用する場合、SecurityManager クラスでもこのメソッドを使用できますが、checkPermission() のデフォルトの SecurityManager 実装では、AccessController.checkPermission() がコールされます。

通常は、静的な AccessController.checkPermission() メソッドを使用します。このコールでは、デフォルトのアクセス制御コンテキスト（スレッド作成時に継承されたコンテキスト）が使用されます。ただし、他のコンテキストについてパーミッション・チェックを行う必要がある場合、特定の AccessControlContext インスタンス上でインスタンス・メソッド checkPermission() をコールできます。次の例では、AccessController メソッドを使用しています。

```
//create permission
FilePermission perm = new FilePermission("/home/developer/foo.txt","read");
//check permission
AccessController.checkPermission(perm);
```

OC4J では、AccessController.checkPermission() がコールされる際に、アクセス制御コンテキストの構成内容と JAAS モードの設定が次のように関係します。

- JAAS モードを使用しない場合（jaas-mode="null"）、Java 2 ポリシー・ファイルで指定されているように、checkPermission() が有効なセキュリティ・ポリシーに基づいてコードベース・セキュリティを強制します。サブジェクトベース・セキュリティの場合、条件はありません。
- doAs JAAS モード（jaas-mode="doas"）の場合、OC4J によってアプリケーション・コードが実行される doAs() ブロックを介して作成された新しいアクセス制御コンテキストに従って、checkPermission() がコードベース・セキュリティとサブジェクトベース・セキュリティの組合せを強制します。OC4J により、デフォルトのアクセス制御コンテキストのパーミッションにサブジェクトのパーミッションが追加されます。
- doAsPrivileged JAAS モード（jaas-mode="doasprivileged"）の場合、checkPermission() は doAs モードの場合と同じように機能しますが、OC4J は、doAsPrivileged() をコールする際に、指定に従って null アクセス制御コンテキストを使用します。これは、サブジェクトベース・セキュリティのみを使用するためです。

関連項目：

- 5-6 ページの「JAAS モードの概要」
- 5-10 ページの「パーミッションをチェックする API」
- 2-16 ページの「JAAS 認可 : サブジェクト・メソッド doAs() および doAsPrivileged()」

JAAS モードの構成と使用

次の例では、5-18 ページの「J2EE 認可 API の使用」に示す doGet() メソッドが拡張され、パーミッションの作成とチェックが行われます。さらに、JAAS モード doAsPrivileged を使用します。これは、アプリケーションの orion-application.xml ファイルで次のような構成によって設定されます。

```
<orion-application ... >
   ...
   <jazn ... jaas-mode="doAsPrivileged" />
   ...
</orion-application>
```

次のコードでは、説明のために 2 つの方法でパーミッションをチェックしています。JAAS モード設定のため、アクション・メソッド（この場合 doGet()）が、OC4J によって Subject.doAsPrivileged() ブロック内で、認証サブジェクトに対して実行されます。

```
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       ServletOutputStream out = response.getOutputStream();

       response.setContentType("text/html");
       out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
       out.println("Time stamp: " + new Date().toString());
       out.println("request.isUserInRole('ar_developers') = " +
                   request.isUserInRole("ar_developers") + "<br>");

      //create Permission
      FilePermission perm = new FilePermission("/home/developer/foo.txt","read");
      {
         // CHECK PERMISSION VIA ACCESS CONTROLLER
         AccessController.checkPermission(perm);

         // CHECK PERMISSION VIA JAAS POLICY
         //get current AccessControlContext
         AccessControlContext acc = AccessController.getContext();

         javax.security.auth.Policy currPolicy =
                                    javax.security.auth.Policy.getPolicy();

         // Query policy now
         out.println("Policy permissions for this subject are " +
                     currPolicy.getPermissions(Subject.getSubject(acc),null));

         //Check Permissions
         out.println("Policy.implies permission: "+ perm +" ? " +
         currPolicy.getPermissions(Subject.getSubject(acc),null).implies(perm));
      }
      out.println("</BODY>");
      out.println("</HTML>");
   }
```

関連項目：

- 5-10 ページの「パーミッションをチェックする API」

Java Authorization Contract for Containers の有効化

この項では、OC4J で Oracle JACC プロバイダを有効にする方法について説明します。JACC を有効にすると、アプリケーションで使用する認可の構成とコードを変更することなく J2EE セキュリティ制約が Java 2 パーミッションに変換され、効率的に J2EE 認可の拡張バージョンが提供されます。

内容は次のとおりです。

- JACC 機能を有効にするシステム・プロパティ
- JACC プロバイダを指定するシステム・プロパティ

注意： JACC は、ファイルベースのプロバイダに対してのみサポートされています。生成されたポリシーは、`system-jazn-data.xml` に格納されます。

関連項目：

- 概要については、5-12 ページの「Java Authorization Contract for Containers の実装」を参照してください。

JACC 機能を有効にするシステム・プロパティ

デフォルトでは、JACC は OC4J で無効になっています。JACC を有効にするには、OC4J 起動時に次のシステム・プロパティが設定されるようにします。

```
-Doracle.oc4j.security.jacc=true
```

JACC プロバイダを指定するシステム・プロパティ

JACC プロバイダを使用するには、アプリケーション・サーバー起動時に、表 5-2 に示すシステム・プロパティが適切に設定されるようにする必要があります。Oracle JACC プロバイダの場合は、JACC を有効にすると、適切な設定であるカッコ内のプロパティが自動的に設定されます。

表 5-2 JACC プロバイダのシステム・プロパティ

プロパティ	説明
javax.security.jacc.policy.provider	ポリシー・プロバイダのクラス名（oracle.security.jacc.provider.J2SEPolicy）
javax.security.jacc.policy.PolicyConfigurationFactory.provider	ポリシー・マッピング構成ファクトリのクラス名（oracle.security.jacc.provider.JACCPolicyConfigurationFactory）
oracle.security.jacc.provider.RoleMappingConfigurationFactory.provider	ロール・マッピング構成ファクトリのクラス名（oracle.security.jacc.provider.JACCRoleMappingConfigurationFactoryImpl）

認可の方法

この項では、これまでに説明した主要な Java セキュリティ・モデル（J2EE、Java 2、JAAS）それぞれについて、使用に適した環境とその仕組みを説明します。操作上の詳細は、このマニュアルの各所（ほとんどはこの章）で説明されています。

関連項目：

- 2-18 ページの「開発時におけるセキュリティの考慮事項」

J2EE セキュリティについて

J2EE（静的なロールベース）セキュリティは、Web アプリケーションや EJB にアクセスできるセキュリティ・ロールを指定する粗密なモデルです。

いつ使用するか

J2EE アプリケーションでは、これが最も単純かつ基本的な形式のセキュリティです。ほぼすべての J2EE アプリケーションで使用され、多くの場合、このセキュリティで十分です。このセキュリティは標準であるため、プラットフォームに依存しません。

ただし、このセキュリティには制限があるため、他のセキュリティ・モデルと併用される場合があります。J2EE セキュリティのみでは、特定のリソースへのアクセスを制限したり、ロールに対して特定のパーミッションを定義できないことに注意してください。また、特定のコード・エンティティでアクセスを制御することもできません。さらに、J2EE セキュリティ・モデルは静的なモデルであるため、実行時にポリシーを変更できません。

設定方法

J2EE セキュリティの設定は、OC4J 固有のディスクリプタ（`orion-application.xml` など）を介したロールマップの指定に加えて、`web.xml` および `ejb-jar.xml` ファイルでのセキュリティ・ロール、ロールリンクおよびセキュリティ制約の標準的な指定を介して行います。

J2EE セキュリティのみを使用する場合、JAAS モードは不要です（`jaas-mode="null"`）。

強制方法

J2EE セキュリティは、J2EE コンテナ（OC4J）によって強制されます。

Java 2 セキュリティについて

Java 2（コードベース）セキュリティでは、実行元コードの場所またはコード署名者に基づいてリソースへのアクセスが制御されます。

いつ使用するか

アプリケーションでコードベースのパーミッションをチェックする必要がある場合、Java 2 セキュリティにアクセスしようとするユーザーやロールに関係なく、Java 2 セキュリティを使用します。Java 2 セキュリティを使用すると、管理者がセキュリティ・マネージャを有効化または無効化することで、セキュリティ・マネージャがセキュリティ・チェックを実行するタイミングを制御できます。

通常、コードベース・セキュリティとセキュリティ・マネージャの使用が必要となるのは、信頼できないコードがアプリケーション・サーバーを参照している場合に限られます。また、セキュリティ・マネージャを使用すると、パフォーマンスに影響が出る可能性があることにも注意してください。

Java 2 セキュリティは、J2EE セキュリティまたは JAAS セキュリティ（あるいはその両方）と併用できます。

設定方法

標準の .policy ファイル（通常は java.policy または java2.policy）を介して Java 2 ポリシーを指定します。ポリシー・ファイル *ORACLE_HOME*/j2ee/home/config/java2.policy は、OC4J によって提供されます。このファイルには、セキュリティ・マネージャによって OC4J を実行する際に必要なパーミッションが格納されています。

Java 2 ポリシー・ファイルを保守する Oracle ツールは提供されていないため、この操作は手動で行うか、または JDK ベンダーやサード・パーティ・ベンダーが提供するツールを使用する必要があります。

Java 2 セキュリティを単独で使用する場合または J2EE セキュリティのみと併用する場合、JAAS モードは不要です（jaas-mode="null"）。

強制方法

Java 2 セキュリティ・ポリシーをアプリケーション・サーバーおよび基礎となる JDK によって強制するには、セキュリティ・マネージャを有効にする必要があります。

アプリケーション内で Java 2 セキュリティ・ポリシーを強制するには、次のようにします。

- セキュリティ・チェックが実行されるタイミングを管理者が制御できる状態で、アプリケーション内でコードベース・セキュリティを実行するには、セキュリティ・マネージャが有効である場合のみコードベース・セキュリティを強制するように選択できます。この場合、SecurityManager インスタンスの checkPermission() メソッドを使用します。

  このシナリオの場合、セキュリティ・マネージャを有効にして JVM を起動する必要があることに注意してください。

- アプリケーション内のコードベース・セキュリティの要件がセキュリティ・マネージャとは無関な場合、セキュリティ・マネージャの存在に関係なくアプリケーション内でコードベース・セキュリティを強制するように選択できます。この場合、静的な AccessController.checkPermission() メソッドを使用してパーミッションをチェックします。

  このシナリオの場合、セキュリティ・マネージャを使用しないと環境全体がセキュアにならないことに注意する必要があります。JDK クラスを含めた環境内のその他のコードは、セキュア・モードでは実行されません。

- チェック対象となる特定の保護ドメインを指定するには、ProtectionDomain インスタンスの配列から AccessControlContext インスタンスを構築し、AccessControlContext インスタンスで checkPermission() メソッドをコールします。（ただし、このシナリオは一般的ではありません。）

JAAS セキュリティについて

JAAS（サブジェクトベース）セキュリティは、認証済サブジェクトの特定のパーミッションに従ってリソースへのアクセスを制御する、比較的密なモデルです。

いつ使用するか

通常は、J2EE モデルの粗密なセキュリティで十分です。JAAS セキュリティは管理とデプロイがより複雑ですが、次のようにリソースをより密に制御する場合に有効です。

- Web モジュールまたは EJB の実行の一部としてリソースがアクセスされるかどうか以外の要因に従って制御する場合
- 管理者によって事前に、または実行時にコードで付与または取り消される個々のパーミッションに従って制御する場合

これとは対照的に、J2EE セキュリティはより静的で、アクセス制御は実行時に変更できません。

通常、JAAS セキュリティは J2EE セキュリティと併用されます。Java 2 セキュリティと併用されることもあります。

設定方法

パーミッションの付与（または取消し）は、OracleAS JAAS Provider Admintool を使用して事前に行うか（5-13 ページの「OracleAS JAAS Provider Admintool を介したパーミッションの付与」を参照）、または実行時に OracleAS JAAS Provider API を介して行うことができます（5-14 ページの「OracleAS JAAS Provider ポリシー管理 API の使用」を参照）。これにより生成される JAAS（サブジェクトベース）ポリシーは、`system-jazn-data.xml` に反映されます。Oracle Identity Management をセキュリティ・プロバイダとして使用する場合は、Oracle Internet Directory に反映されます。

JAAS モードをどのように設定するかも関係します。Java 2（コードベース）認可なしで JAAS 認可を使用するには、`jaas-mode="doasprivileged"` を設定することによって `doAsPrivileged` JAAS モードを使用し、`Policy.implies()` メソッドを使用してパーミッションをチェックします。

JAAS 認可を Java 2 認可と併用するには、次の操作を行います。

- `jaas-mode="doas"` を設定することにより、`doAs` JAAS モードを使用します。
- 標準の `.policy` ファイル（通常は Oracle 提供の `java2.policy` ファイル）を介して Java 2 ポリシーを指定します。

注意： `doAsPrivileged` モードでアクセスされるリソースが JVM ベンダーによって定義されたリソースの場合、セキュリティ・マネージャが有効になっていると、追加のセキュリティ・チェックが実行される可能性があります。

強制方法

JAAS セキュリティは、`AccessController.checkPermission()` メソッドまたは `Policy.implies()` メソッドを介して強制できます。

Java 2 セキュリティを併用している場合にセキュリティ・マネージャを有効にすると、`SecurityManager.checkPermission()` メソッドも使用できるようになります。

（Java 2 ポリシーの強制に関する前述の説明も参照してください。）

6

OC4J セキュリティに関する一般的なタスク

この章では、OC4J で使用可能な各種セキュリティ・プロバイダ全体に該当する一般的なタスクと関連するガイドラインについて説明します。

- パスワード管理のタスク
- OC4J でのセキュリティ・レルムの使用方法
- セキュリティに関するデプロイ・タスク
- セキュリティに関するデプロイ後のタスク
- ライブラリを共有するためのタスク

関連項目：

- セキュリティのベスト・プラクティスの詳細は、『Oracle Application Server ベスト・プラクティス』を参照してください（リリース後に入手可能）。
- OC4J の様々な使用方法の例は、次の Web サイトを参照してください。

  `http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

パスワード管理のタスク

OC4J コンポーネントの多くは、認証にパスワードを必要とします。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。

- パスワードの間接化 : クリアテキストのパスワードを、他の場所（OC4J の `system-jazn-data.xml`）で検索するために必要な情報で置換します。
- 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。

この項の以降の部分で、次の項目について説明します。

- パスワードの間接化の使用方法
- system-application.xml でのパスワード・マネージャの指定
- OC4J 構成ファイルのパスワードの不明瞭化

パスワードの間接化の使用方法

次に示す構成ファイルと各構成ファイル内の 1 つ以上の要素で、パスワードの間接化がサポートされています。

- `data-sources.xml`: `<native-data-source>` および `<managed-data-source>` 要素の `password` 属性
- `ra.xml`: `<res-password>` 要素
- `rmi.xml`: `<ssl-config>` 要素の `keystore-password` 属性
- `jms.xml`: `<password>` 要素
- `*-web-site.xml`: `<ssl-config>` 要素の `keystore-password` 属性

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。

次に、間接パスワードと直接パスワードの例をいくつか示します。

- `<data-source password="->Scott">`

  デフォルト・レルムで `Scott` を検索し、パスワード・マネージャに格納されているパスワードを使用します。

- `<res-password="->customers/Scott">`

  `customers` レルムで `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="mypass">`

  リテラル文字列 `mypass` がパスワードですが、間接パスワードではありません。

注意 : 現在の OC4J 実装で間接パスワードの使用を選択すると、`system-jazn-data.xml` ファイルに間接ユーザーが作成されます。自動または手動で間接ユーザー・アカウントが作成された場合、アプリケーションをアンデプロイしてもそのアカウントは自動的に削除されないため、手動で削除する必要があります。

---

system-application.xml でのパスワード・マネージャの指定

OC4J 固有の system-application.xml ファイル（OC4J system アプリケーションと関連）の <password-manager> 要素では、間接パスワードの検索に使用するセキュリティ・プロバイダを指定します（前項の「パスワードの間接化の使用方法」を参照してください）。この要素を省略すると、間接パスワードの認証と認可にはグローバル・アプリケーションのセキュリティ・プロバイダが使用されます。system-application.xml の <password-manager> 要素内にある <jazn> 要素は、このファイルの最上位にある <jazn> 要素と異なっていてもかまいません。

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、Oracle Internet Directory 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 Oracle Identity Management がセキュリティ・プロバイダとして使用される場合でも、ファイルベースのプロバイダ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。

それには、次のようなエントリを OC4J 固有の system-application.xml ファイルに追加します。

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

注意： デフォルトでは、system-jazn-data.xml をパスワード・マネージャとして使用します。

OC4J 構成ファイルのパスワードの不明瞭化

JAAS 構成ファイル jazn.xml および system-jazn-data.xml（またはオプションでアプリケーション固有の jazn-data.xml ファイル）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。

通常、jazn.xml または system-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。

また（Oracle Identity Management に関連して）、orion-application.xml などの <jazn> 要素内の ldap.password プロパティは不明瞭化されます。次に例を示します。

```
<jazn ... >
   <property name="ldap.password" value="welcome123"/>
   ...
</jazn>
```

他の OC4J 構成では、6-2 ページの「パスワードの間接化の使用方法」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。

注意： system-jazn-data.xml またはアプリケーション固有の jazn-data.xml ファイルでは、次のいずれかの方法でクリア（判読可能）なパスワードを指定できます。ただし、これはお薦めできません。

- <credentials> 要素の clear 属性を true に設定します。

```
<user>
   <name>myname</name>
   <credentials clear="true">welcome</credentials>
   ...
</user>
```

- パスワードの前に ! を付加します。（この場合、! はパスワードの一部とみなされません。）

```
   <credentials>!welcome</credentials>
```

OC4J でのセキュリティ・レルムの使用方法

OC4J では、ファイルベースのプロバイダと LDAP ベースの Oracle Identity Management の両方でセキュリティ・レルムの概念が使用されます（3-3 ページの「OracleAS JAAS Provider のセキュリティ・レルム」を参照してください）。レルムは単一でも複数でも構成できますが、デフォルトのレルムは OC4J 構成を介して指定します。Active Directory や Sun Java System Directory Server などの外部 LDAP プロバイダを使用する場合は、レルムの概念はサポートされていませんので、注意してください。

この項では、OC4J の認証と認可を制御するセキュリティ・レルムを使用する場合の重要な事項について説明します。この項の内容は次のとおりです。

ファイルベースのプロバイダまたは Oracle Identity Management のデフォルト・レルム

デフォルト・レルムは、<jazn> 要素の default-realm 属性で指定します。ファイルベースのプロバイダの場合、これは orion-application.xml ファイルのアプリケーション・レベルか、インスタンス・レベルの jazn.xml ファイルの OC4J レベルにあります。Oracle Identity Management の場合、これは OC4J home インスタンスの jazn.xml ファイルにあります。

ファイルベースのプロバイダの場合、jazn.com は、インスタンス・レベルのデフォルト・レルムとしてデフォルトで構成されています。

```
<jazn provider="XML" location="./system-jazn-data.xml" default-realm="jazn.com" />
```

Oracle Identity Management の場合、デフォルト・レルムは Oracle Internet Directory に基づいており、Oracle Internet Directory のインストール時に決定されます。OC4J を Oracle Internet Directory インスタンスに関連付けると、デフォルト・レルムの反映先は、次の例のように OC4J インスタンス・レベルになります。

```
<jazn provider="LDAP" location="ldap://www.example.com:636" default-realm="us"/>
```

デフォルト・レルムの使用上のガイドラインについては、6-6 ページの「デフォルト・レルムの使用方法」を参照してください。

重要：

- デフォルト・レルムは必ず指定してください（レルムを 1 つしか使用しない場合はデフォルトとして使用）。これは、ファイルベース・プロバイダの場合、アプリケーション・デプロイ時にセキュリティ・プロバイダを構成する際にデフォルト・レルムを指定するということです。
- `system-jazn-data.xml` から、`jazn.com` レルムの構成を削除しないでください。デフォルトで記述されており、OC4J `system` アプリケーションで使用するため残しておく必要があります。

ファイルベースのプロバイダまたは Oracle Identity Management のデフォルト・レルムの評価

前述の項で説明したように、デフォルト・レルムは必ず構成してください。ただし、この項では、ファイルベース・プロバイダまたは Oracle Identity Management の使用時にデフォルト・レルムを指定しなかった場合に、その特定のために行われる後続の処理を、参考目的でのみ説明します。

アプリケーションでファイルベース・プロバイダを使用する場合：

1. OracleAS JAAS Provider では、アプリケーション・レベルのデフォルト・レルム構成が `orion-appliation.xml` ファイルで検索されます。そこでデフォルト・レルムが見つかると、アプリケーションのデフォルト・レルムとして使用されます。
2. アプリケーション・レベルのデフォルト・レルム設定がない場合、OracleAS JAAS Provider では、`jazn.xml` ファイルで OC4J インスタンス・レベルのデフォルト・レルム構成が検索されます。
   - `jazn.xml` で `provider="XML"` が設定されている場合、OracleAS JAAS Provider では `jazn.xml` で指定されているデフォルト・レルム（指定されている場合）がアプリケーションのデフォルト・レルムとして使用されます。何も指定されていない場合は、デフォルト・レルム属性がないことを示すエラーがスローされます。
   - `jazn.xml` で `provider="LDAP"` が設定されている場合、OracleAS JAAS Provider では `jazn.com` がアプリケーションのデフォルト・レルムとして使用されます。

アプリケーションで Oracle Identity Management を使用する場合：

1. （`jazn.xml` の）構成で LDAP ベースのプロバイダがアプリケーションと OC4J インスタンス・レベルで指定されている場合は、OracleAS JAAS Provider では、`jazn.xml` でデフォルト・レルム構成が検索されます。そこでデフォルト・レルムが見つかると、アプリケーションのデフォルト・レルムとして使用されます。
2. 構成に OC4J インスタンス・レベルの LDAP ベース・プロバイダが指定されていない場合、またはインスタンス・レベルのデフォルト・レルム設定がない場合は、Oracle Internet Directory デフォルト・サブスクライバがデフォルト・レルムとして使用されます。（これは Oracle Internet Directory サーバーに構成されています。）

デフォルト・レルムの使用方法

認証の場合は、デフォルト・レルムを使用するとき、ユーザー名の前にレルム名を接頭辞として付ける必要はありません。たとえば、ユーザー scott がデフォルト・レルム jazn.com にある場合、認証用には、ユーザー名を scott とのみ指定します。

これは該当する OC4J コンポーネントや、JNDI、JMS、J2EE Connector Architecture などのサービスにも当てはまります。

同様に、パスワードの間接化を行う場合にも、OC4J デプロイメント・ディスクリプタでは、間接化対象として指定するユーザー名の前にレルム名を付ける必要はありません。つまり、->scott と指定します。

非デフォルト・レルムの使用方法

この説明での acme.com のような非デフォルト・レルム（つまりカスタム・レルム）を使用している場合は、ユーザー名に接頭辞としてレルム名を付ける必要があります。たとえば、acme.com のユーザー scott を認証するには、ただ scott とするのではなく、acme.com/scott と指定する必要があります。

これは該当する OC4J コンポーネントや、JNDI、JMS、J2EE Connector Architecture などのサービスにも当てはまります。

同様に、パスワードの間接化を行う場合にも、OC4J デプロイメント・ディスクリプタでは、ユーザーが非デフォルト・レルムに定義されているときは、間接化のために指定するユーザー名にレルム名を接頭辞として付ける必要があります。すなわち、->acme.com/scott のように指定します。

また、カスタム・レルムを使用する場合に、そのカスタム・レルムでユーザーまたはロールに JAAS ポリシーが付与されているときは、次の作業を実行する必要があります。

1. アプリケーションの orion-application.xml ファイルの <jazn> 要素で、default-realm 設定として *custom_realm_name* を指定します。
2. <jazn> 要素の location 属性の設定は指定しないでください。
3. jazn.xml で、<jazn> 要素の <property> サブ要素を使用して、jaas.username.simple プロパティを false に設定します（6-7 ページの「認証済プリンシパル取得時のレルム名の省略」を参照してください）。

これらの手順によって、カスタム・レルムとそのユーザー、ロールおよびポリシーを永続化できます。

JAAS 認可を使用する場合、特にカスタム・レルムでユーザーまたはロールにパーミッションを付与する場合は、カスタム・レルムとそのユーザーおよびグループを、アプリケーション固有の jazn-data.xml ファイルではなく、system-jazn-data.xml で定義し、永続化する必要があります。

複数レルムの使用方法

複数レルムを構成するときは、使用する非デフォルト・レルムに対してユーザー名にレルム名を接頭辞として付ける必要があります。この説明では、レルム jazn.com、acme.com および example.org が構成済で、jazn.com がデフォルト・レルムであるとします。さらに、ユーザー scott が jazn.com 内に、ユーザー ralph が example.org 内に定義されているものとします。

scott を認証対象として指定するには、このユーザーを scott としてのみ指定します。このユーザーがデフォルト・レルム jazn.com 内に定義されているためです。

ralph を認証対象として指定するには、example.org/ralph と指定する必要があります。

これは該当する OC4J コンポーネントや、JNDI、JMS、J2EE Connector Architecture などのサービスにも当てはまります。レルム名は、非デフォルト・レルムのユーザーに対して指定します。

同様に、パスワードの間接化を行う場合にも、OC4J デプロイメント・ディスクリプタでは、ユーザーが非デフォルト・レルムに定義されているときは、間接化のために指定するユーザー名にレルム名を接頭辞として付ける必要があります。すなわち、->example.org/ralph のように指定します。ただし、デフォルト・レルムにあるユーザーには、レルム名を指定する必要がありません。すなわち、scott のようにします。

重要： 複数レルムが構成されているときは、常に jaas.username.simple を false に設定します。（次の「認証済プリンシパル取得時のレルム名の省略」を参照してください。）

認証済プリンシパル取得時のレルム名の省略

カスタム・レルムを構成する場合を除いて、通常は認証プリンシパルではレルム名を省略するようにしてください。認証プリンシパルは、サーブレット、EJB および Web サービスの主要なメソッドによって戻されます。OC4J では、この動作の制御に jaas.username.simple プロパティを使用します。このプロパティにより、次のメソッドが影響を受けます。

- HTTPServletRequest インスタンスの getUserPrincipal() メソッド（サーブレット）
- HTTPServletRequest インスタンスの getRemoteUser() メソッド（サーブレット）
- EJBContext インスタンスの getCallerPrincipal() メソッド（EJB）
- ServletEndpointContext インスタンスの getUserPrincipal() メソッド（Web サービス）

プロパティ設定に true（デフォルト）を指定すると、これらのメソッドから戻されるプリンシパル名にレルム名が含まれなくなります。たとえば、scott のようになります。

プロパティを false に設定すると、これらのメソッドから戻されるプリンシパル名に、接頭辞としてレルム名が付けられます。たとえば jazn.com/scott のようになります。

false 設定を指定するには、次のように <jazn> 要素の <property> サブ要素（アプリケーション・レベルの場合は orion-application.xml 内、OC4J インスタンス・レベルの場合はインスタンス・レベルの jazn.xml ファイル内）を使用します。

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

重要：

- Application Server Control を介して任意の時点で任意のアプリケーションに対してファイルベース・プロバイダから Oracle Identity Management に切り替えると、そのアプリケーションの orion-application.xml 内にある <jazn> 要素が次のように置き換えられます。<jazn> 要素の以前の設定はすべて失われるため、設定をやりなおす必要があります。

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

- 複数レルムが構成されているときは、常に jaas.username.simple を false に設定します。

セキュリティに関するデプロイ・タスク

この項では、アプリケーションのデプロイ時に考慮する必要があるセキュリティ上の事項について説明します。この項の内容は次のとおりです。

関連項目：

- デプロイの詳細と関連の注意事項は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。

デプロイ上の注意事項の概要

セキュリティ・プロバイダは、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、セキュリティ・プロバイダではすべての 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。

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。

- 使用するセキュリティ・プロバイダを決定します。Oracle Application Server には、LDAP ベースの Oracle Internet Directory をリポジトリとして使用する Oracle Identity Management と、XML ファイルをリポジトリとして使用するファイルベースのプロバイダが付属しています。また、OC4J では、外部（サード・パーティ）LDAP プロバイダ、カスタム・セキュリティ・プロバイダ（カスタム・ログイン・モジュール）もサポートされています。さらに、OC4J 10.1.3.x 実装からは、Oracle Access Manager もサポートされています。
- 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 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。
- 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、OracleAS JAAS Provider により定義されたユーザーとロールにマップします。たとえば、J2EE ロール `hr_manager` を、OracleAS JAAS Provider 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。
- 共有ライブラリ（たとえばログイン・モジュール）としてロードするコードがあるかどうかを検討します。

アプリケーションのデプロイ

この項では、Application Server Control コンソールの機能に沿って、アプリケーションのデプロイ方法について説明します。

Application Server Control を介したアプリケーションのデプロイ

デプロイ・プランの詳細や Application Server Control コンソールの使用方法などを含めた、OC4J へのアプリケーションのデプロイに関する一般情報は、『Oracle Containers for J2EE デプロイメント・ガイド』を参照してください。ここでは、次の基本手順を確認しておきます。

1. OC4J ホームページで**「アプリケーション」**タブを選択します。（Oracle Application Server 環境では、「クラスタ・トポロジ」ページから目的の OC4J インスタンスを選択してそのホームページにアクセスします。）
2. 表示される「アプリケーション」ページで、**「デプロイ」**を選択します。
3. 表示される「デプロイ : アーカイブの選択」ページ（ページ 1/3）で、デプロイするアーカイブ・ファイルを指定し、目的のデプロイ・プランを選択します。
4. 「デプロイ : アプリケーション属性」ページ（ページ 2/3）で、目的のアプリケーション名、親アプリケーション、Web サイトおよびコンテキスト・ルートを指定します。
5. 「デプロイ : デプロイ設定」ページ（ページ 3/3）で、次のタスクのいずれかを選択できます。
   - 環境参照のマップ（該当する場合）
   - セキュリティ・プロバイダの選択
   - セキュリティ・ロールのマップ（該当する場合）
   - EJB の構成（該当する場合）
   - クラスタリングの構成
   - クラスのロードの構成（共有ライブラリをロードする場合など）

   セキュリティ上で特に重要なのは、セキュリティ・プロバイダの選択とセキュリティ・ロールのマップです。また、共有ライブラリに対するクラスのロードの構成が必要なことがあります。たとえば、共有ライブラリとしてロードするログイン・モジュールがある場合などです。
6. 「デプロイ : デプロイ設定」ページで、前の手順に記載されたタスクの終了後に、**「デプロイ」**を選択します。

関連項目：

- 6-10 ページの「Application Server Control を介したセキュリティ・プロバイダの指定」
- 6-12 ページの「Application Server Control を介したセキュリティ・ロール・マッピングの指定」
- 9-15 ページの「OC4J 共有ライブラリとしてのログイン・モジュールの提供」

セキュリティ・プロバイダの指定

この項では、Application Server Control コンソールを使用してセキュリティ・プロバイダを指定する方法について説明します。また、ファイルベース・プロバイダと LDAP ベース・プロバイダの使用方法の差異に関する注意事項についても説明します。

ファイルベースのプロバイダと Oracle Identity Management との比較

一般に、ファイルベースのプロバイダは、開発中およびユーザー数の少ないデプロイ済アプリケーション（スタンドアロン OC4J 環境など）で使用します。Oracle Identity Management は、大規模な本番環境で使用します。

ファイルベースのプロバイダはより軽量な実装であり、一方の Oracle Identity Management はセキュリティとパフォーマンスの面で優れています。集中型 Oracle Internet Directory サーバーは、アプリケーションとユーザーの数の増大に適切に対応でき、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。また、複数の OC4J インスタンスからのユーザー・リポジトリへのアクセスが簡素化されます。ファイルベースのプロバイダでは、それらの各インスタンスが独自にリポジトリを保有している場合に、ユーザー・データの更新をインスタンス間で整合させる必要があります。

さらに、Oracle Internet Directory では、アカウントの集中的な作成と管理、シングル・パスワード、資格証明管理などの機能を提供します。

注意： Oracle Identity Management を使用するには、事前に Application Server Control を介して OC4J インスタンスを Oracle Internet Directory インスタンスに関連付けておく必要があります。

Application Server Control を介したセキュリティ・プロバイダの指定

Application Server Control コンソールでは、次のように、デプロイ時に「デプロイ : デプロイ設定」ページからセキュリティ・プロバイダを指定します（このページへのナビゲート方法については、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください）。

1. 「セキュリティ・プロバイダの選択」タスクを選択します。
2. 表示される「デプロイ設定 : セキュリティ・プロバイダの選択」ページで、ドロップダウン・リストから目的のセキュリティ・プロバイダを選択します。各選択肢を次に示します。
   - ファイルベース
   - Oracle Identity Management
   - サード・パーティの LDAP サーバー（外部 LDAP プロバイダの場合）
   - カスタム（カスタム・ログイン・モジュールの場合）
3. 各セキュリティ・プロバイダ・タイプには、それを構成するための独自のタスク・セットが必要です。タスクの詳細は、次を参照してください。
   - 7-3 ページの「アプリケーション・デプロイ時のファイルベース・プロバイダの構成」
   - 8-14 ページの「デプロイ時の Oracle Identity Management の指定」
   - 9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」
   - 10-3 ページの「デプロイ時の外部 LDAP プロバイダの指定および構成」
4. 「デプロイ : デプロイ設定」ページが再表示されるので、**「デプロイ」**を選択してデプロイを完了するか、または必要に応じて他のタスクを選択します。タスクのリストは、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください。

注意： Oracle Access Manager セキュリティ・プロバイダは、Application Server Control ではサポートされません（ただし、そのログイン・モジュールは他のログイン・モジュールと同様に Application Server Control を介して構成できます）。Oracle Access Manager の構成方法の詳細は、第 11 章「Oracle Access Manager」を参照してください。

セキュリティ・ロールのマッピング

この項では、次の項目について様々な側面から説明します。

- アプリケーションでのセキュリティ・ロールの定義と、これらのロールと標準デプロイメント・ディスクリプタで宣言されている J2EE 論理ロールとのリンク
- OC4J 固有の構成における J2EE ロールからデプロイ・ロールへのマッピング、および Application Server Control におけるこのマッピングの実行方法

次のような構成で説明します。

- アプリケーション・ロールの定義と参照
- Application Server Control を介したセキュリティ・ロール・マッピングの指定
- OC4J 構成ファイルにおける J2EE ロールのデプロイ・ロールへのマッピング
- OC4J PUBLIC ロールを使用して認証済ユーザーによる一般的なアクセスを許可する方法

注意： セキュリティ・ロール・マッピングは、親アプリケーションからは継承されません。

関連項目：

- 3-8 ページの「セキュリティ・ロール・マッピングの概要」
- 17-7 ページの「Web アプリケーションのセキュリティ・ロールおよび制約の構成」
- 18-7 ページの「J2EE ロールからデプロイ・ユーザーおよびデプロイ・ロールへのマッピング」

アプリケーション・ロールの定義と参照

セキュリティ・ロールの定義および参照の手順は次のとおりです。

1. 標準デプロイメント・ディスクリプタ（`web.xml` または `ejb-jar.xml`）で、`<security-role>` 要素を使用して次の例のような J2EE 論理ロールを定義します。

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

2. 次の例のように、標準デプロイメント・ディスクリプタの `<security-role-ref>` 要素を使用して、アプリケーションで作成したロールを標準ディスクリプタで定義した J2EE ロールにリンクします（この例の `ar_developers` はアプリケーション・ロールです）。

```
<security-role-ref>
   <role-name>ar_developers</role-name>
   <role-link>sr_developers</role-link>
</security-role-ref>
```

これらの手順を実行すると、セキュリティ・プロバイダで定義された（たとえばファイルベースのプロバイダの場合は jazn-data.xml ファイルや system-jazn-data.xml ファイル、LDAP ベースのプロバイダの場合は Oracle Internet Directory で定義）デプロイ・ロールへのマッピングは、OC4J 固有のディスクリプタである orion-web.xml、orion-ejb-jar.xml または orion-application.xml で定義されます。これらのファイルは、Application Server Control を介したアプリケーションのデプロイ時に、必要に応じて定義したマッピングによって更新され、その更新内容は <security-role-mapping> 要素に反映されます。これらのマッピングについては、後続の 2 つの項「Application Server Control を介したセキュリティ・ロール・マッピングの指定」および「OC4J 構成ファイルにおける J2EE ロールのデプロイ・ロールへのマッピング」を参照してください。

> **関連項目：** 前述の説明では、Web アプリケーションと EJB の間で異なる一部の詳細を省略しています。追加情報は次の項を参照してください。
>
> - 17-7 ページの「Web アプリケーションのセキュリティ・ロールおよび制約の構成」
> - 18-2 ページの「EJB アプリケーションの認証と認可」

Application Server Control を介したセキュリティ・ロール・マッピングの指定

Application Server Control コンソールでは、次のように、デプロイ処理中に「デプロイ : デプロイ設定」ページから J2EE ロールをデプロイ・ロールにマップします（このページへのナビゲート方法については、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください）。

1. 「セキュリティ・ロールのマップ」タスクを選択します。
2. 「デプロイ設定 : セキュリティ・ロールのマップ」ページで、マップする J2EE ロールごとに「ロールのマップ」タスクを選択します。（**「すべてのマッピングを消去」**を選択することもできます。）
3. ロール用の「デプロイ設定 : セキュリティ・ロールのマップ」ページで、次のいずれかの操作を実行できます。
   - すべてのユーザーとグループ（デプロイ・ロール）を J2EE ロールにマップします。
   - 選択したユーザーを J2EE ロールにマップします。**「既存のユーザーの追加」**を選択してから、「検索と選択 : ユーザー」ページで目的のユーザーを指定し、**「選択」**を選択します。**「既存のユーザーの追加」**に目的のユーザーが表示されない場合は、「デプロイ設定 : セキュリティ・ロールのマップ」ページの「ユーザーの追加」機能を使用します。
   - 選択したグループを J2EE ロールにマップします。**「既存のグループの追加」**を選択してから、「検索と選択 : グループ」ページで目的のグループを指定し、**「選択」**を選択します。**「既存のグループの追加」**に目的のグループが表示されない場合は、「デプロイ設定 : セキュリティ・ロールのマップ」ページの「グループの追加」機能を使用します。
   - ユーザーとグループのマッピングの終了後、**「続行」**を選択します。
4. 再表示される「デプロイ設定 : セキュリティ・ロールのマップ」ページで、**「OK」**を選択します。
5. 「デプロイ : デプロイ設定」ページが再表示されるので、**「デプロイ」**を選択してデプロイを完了するか、または必要に応じて他のタスクを選択します。タスクのリストは、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください。

これらのアクションにより、orion-application.xml、orion-web.xml、orion-ejb-jar.xml などの該当する OC4J 構成ファイルに <security-role-mapping> 要素が作成されます（次の「OC4J 構成ファイルにおける J2EE ロールのデプロイ・ロールへのマッピング」を参照）。

注意： デプロイ後に Application Server Control を介してセキュリティ・マッピングを変更することはできません。これを行うには、手動で構成を更新し（17-10 ページの「OC4J による J2EE ロールのデプロイ・ロールへのマッピング」および 18-7 ページの「J2EE ロールからデプロイ・ユーザーおよびデプロイ・ロールへのマッピング」を参照）、アプリケーションを再起動または再デプロイします。

OC4J 構成ファイルにおける J2EE ロールのデプロイ・ロールへのマッピング

標準デプロイメント・ディスクリプタに定義された移植可能な J2EE 論理ロールは、`orion-application.xml` ファイル（J2EE アプリケーション全体に適用の場合）、`orion-web.xml` ファイル（特定の Web アプリケーションに適用の場合）、`orion-ejb-jar.xml` ファイル（特定の EJB アプリケーションに適用の場合）のいずれかの `<security-role-mapping>` 設定を介して、デプロイ・ロールにマップされます。

この例では、J2EE ロール `sr_developers` をデプロイ・ロール `developers` にマップしています。`<security-role-mapping>` 要素下の `<group>` サブ要素は、`system-jazn-data.xml` や Oracle Internet Directory で定義されているデプロイ・ロールなどに対応していることに注意してください。`<user>` サブ要素を個々のユーザーにマップすることもできます。

```
<security-role-mapping name="sr_developers">
   <group name="developers" />
</security-role-mapping>
```

この関連付けにより、`developer` ロールは、標準デプロイメント・ディスクリプタで構成されたセキュリティ制約に従って、`sr_developers` ロールがアクセスできるリソースにアクセスできるようになります。

たとえば、`developer` デプロイ・ロールのメンバーであるユーザー `john` について考えてみます。このロールは J2EE ロール `sr_developers` にマップされているため、`john` は、`sr_developers` ロールからアクセス可能なアプリケーション・リソースにアクセスできます。

OC4J PUBLIC ロールを使用して認証済ユーザーによる一般的なアクセスを許可する方法

認可は不要で認証についてのみ考慮が必要な状況のために、OC4J では、認証済ユーザーが所定のアプリケーションまたはリソースにアクセスできるモードがサポートされています。このモードは `PUBLIC` ロールに対してサポートされ、必要に応じて URL 単位またはメソッド単位で構成できます。手順は次のとおりです。

1. public アクセス用の J2EE 論理ロールがまだない場合は、`web.xml`（Web アプリケーションの場合）または `ejb-jar.xml`（EJB の場合）にそのようなロールを定義できます。

   たとえば、`web.xml` では次のようにします。

   ```
   <web-app>
      ...
      <security-role>
         <role-name>public_role</role-name>
      </security-role>
      ...
      <auth-constraint>
         <role-name>public_role</role-name>
      </auth-constraint>
      ...
   </web-app>
   ```

また、ejb-jar.xml では次のようにします。

```
<assembly-descriptor>
   ...
   <security-role>
      <role-name>public_role</role-name>
   </security-role>
   ...
   <method-permission>
      <role-name>public_role</role-name>
      <method>method</method>
   </method-permission>
   ...
</assembly-descriptor>
```

2. orion-application.xml（Web アプリケーションの場合）または orion-ejb-jar.xml（EJB の場合）の PUBLIC ロールに、使用している PUBLIC ロールをマップします。

   上の web.xml で定義したロールをマップするには、orion-application.xml に次を挿入します。

```
<orion-application>
   ...
   <security-role-mapping name="public_role">
      <group name="{{PUBLIC}}"/>
   </security-role-mapping>
   ...
</orion-application>
```

   また、EJB の場合は、かわりに orion-ejb-jar.xml の <security-role-mapping> 要素を使用します（この要素は、<assembly-descriptor> 要素のサブ要素です）。

> **注意：** この例では、OC4J の public グループとして、デフォルト設定の「{{PUBLIC}}」を想定しています。これは、OracleAS JAAS Provider の public.group プロパティを使用して上書きできます。

セキュリティに関するデプロイ後のタスク

この項では、アプリケーションをデプロイした後の次の注意事項について説明します。

- アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート

アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート

アプリケーションのデプロイ後に、Application Server Control コンソールにあるアプリケーションの「セキュリティ・プロバイダ」ページに移動して、アプリケーション・レベルのセキュリティ設定を確認または更新できます。OC4J インスタンスの OC4J ホームページから次の手順を実行します。

1. **「管理」**タブを選択します。
2. 「管理」ページで、(「セキュリティ」の下で)「セキュリティ・プロバイダ」タスクを選択します。
3. 「セキュリティ・プロバイダ」ページの「アプリケーション・レベルのセキュリティ」で、アプリケーションに対して「編集」タスクを選択します。

これにより、「セキュリティ・プロバイダ」ページが表示され、アプリケーションのプロバイダに関する情報が表示されて、設定の更新や、別のセキュリティ・プロバイダへの変更ができます。

ライブラリを共有するためのタスク

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。ライブラリを OC4J 共有ライブラリとしてロードすることで、それらのライブラリがアプリケーションで利用可能になります。たとえば、複数のアプリケーション間で次を共有する場合などは、この機能が役立ちます。

- ログイン・モジュール
- 認可およびサブジェクト伝播用のプリンシパル・クラス
- アイデンティティ管理フレームワーク実装クラス

ライブラリの共有と使用は、主に次の 2 つの手順からなります（具体的に Application Server Control コンソールの機能を考えた場合）。

1. OC4J 共有ライブラリとしてのライブラリのロード
2. アプリケーションへのライブラリのインポート

注意： `<library>` 要素と *ORACLE_HOME*`/j2ee/home/applib` の場所は OC4J 共有ライブラリで現在もサポートされていますが、非推奨になっています。

関連項目：

- OC4J クラスのロードおよび共有ライブラリの詳細は、『Oracle Containers for J2EE 開発者ガイド』を参照してください。

OC4J 共有ライブラリとしてのライブラリのロード

Application Server Control からライブラリを OC4J 共有ライブラリとしてロードするには、「共有ライブラリ」タスクを使用します。

1. OC4J インスタンスの**「管理」**タブを表示します。
2. 「共有ライブラリ」タスク（「プロパティ」の下）を選択します。
3. 「共有ライブラリ」ページで、**「作成」**を選択します。
4. 「共有ライブラリの作成 : 属性」ページで、目的のライブラリの名前とバージョンを指定し、次のページに進みます。
5. 「共有ライブラリの作成 : アーカイブの追加」ページで、**「追加」**を選択してライブラリを追加します。
6. 「アーカイブの追加 : アーカイブの追加」ページで、ローカル・ホストからのライブラリのアップロード、Application Server Control が置かれたサーバーからのライブラリのアップロードまたはライブラリがすでに存在する対象サーバーでの場所の指定を実行できます。追加するライブラリごとにこの処理を繰り返してから、次に進みます。
7. 再表示された「共有ライブラリの作成 : アーカイブの追加」ページで、処理を終了するか、次のページ「共有ライブラリの作成 : 共有ライブラリのインポート」に進みます。このページでは、ライブラリに追加のライブラリをインポートし、処理を終了できます。処理を終了すると、「共有ライブラリ」ページが再表示されます。

ライブラリをロードすると、OC4J `server.xml` ファイルに次のような構成が生成されます。

```
<application-server ... >
   ...
   <shared-library name="mylib.lib" version="1.0" library-compatible="true">
      <code-source path="../mypath" />
   </shared-library>
   ...
</application-server>
```

アプリケーションへのライブラリのインポート

アプリケーションのデプロイ中に、Application Server Control を介してアプリケーションにライブラリをインポートできます。

1. 「デプロイ : デプロイ設定」ページが表示されたら（6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照）、「クラスのロードの構成」タスクを選択して共有ライブラリをインポートできます。
2. 「デプロイ設定 : クラスのロードの構成」ページで、インポートするライブラリを選択し、**「OK」**を選択します。
3. デプロイを続行します。

ライブラリをインポートすると、アプリケーションの `orion-application.xml` ファイルに次のような構成が生成されます。

```
<orion-application ... >
   ...
   <imported-shared-libraries>
      <import-shared-library name="mylib.lib" />
      ...
   </imported-shared-libraries>
   ...
</orion-application>
```

7

ファイルベースのセキュリティ・プロバイダ

OC4J では、ファイルベースのセキュリティ・プロバイダが用意されています。XML ベースのファイルがユーザー、ロールおよびポリシーのリポジトリとして使用されます。このファイルベース・プロバイダは、一般に開発時や小規模な本番環境（スタンドアロン OC4J を使用する場合など）で使用されます。デフォルトのセキュリティ・プロバイダでもあります。具体的には、OracleAS JAAS Provider ではファイルベース（XML ベース）プロバイダの次のタスクがサポートされています。

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

これらの情報は、XML リポジトリ（通常 `system-jazn-data.xml`）に格納されます。ただし、かわりにアプリケーション固有の `jazn-data.xml` ファイルを使用することもできます。

この章では、Application Server Control コンソールの機能に沿って、ファイルベース・プロバイダのユーザー、ロール、およびレルムの基本管理タスクについて説明します。

この章の内容は次のとおりです。

注意：

- ファイルベース・プロバイダでは、認可のためのロール比較では大 / 小文字が区別される点に注意してください。
- デフォルトでは、ファイルベース・プロバイダがセキュリティ・プロバイダで、`system-jazn-data.xml` ファイルがリポジトリ、`jazn.com` がレルムになります。`system-jazn-data.xml` ファイルは、*`ORACLE_HOME`*`/j2ee/`*`instance_name`*`/config` ディレクトリに存在します。このリポジトリへの変更は、リポジトリを使用するすべてのアプリケーションから可視です。

関連項目：

- `oc4jadmin` 以外の管理者アカウントを使用する場合は、4-13 ページの「新しい管理者アカウントの作成」を参照してください。

ファイルベース・プロバイダのポリシー / レルム管理用のツール

ファイルベース・プロバイダのユーザーとロールを管理するには、Application Server Control コンソールを使用します（7-4 ページの「Application Server Control を介したアプリケーション・レルムの管理」を参照）。これにより、ユーザー・リポジトリ（`system-jazn-data.xml` ファイル、または供給するアプリケーション固有 `jazn-data.xml` ファイル）が更新されます。

ファイルベース・プロバイダのポリシーを管理するには、OracleAS JAAS Provider Admintool を使用します。C-4 ページの「Admintool のコマンドライン構文およびオプションの概要」にリストされているポリシー・オプションを参照してください。

通常、`system-jazn-data.xml` または `jazn-data.xml` ファイルは直接操作しないようにします。

注意： ポリシー管理のツールでは例外的に、ファイルベース・プロバイダのロールに RMI パーミッションまたは管理パーミッションを付与することは、Application Server Control を介したロールの編集または追加の一環としてできます。

RMI を使用した EJB へのアプリケーション・アクセスを有効にするには、ユーザーまたはロールに RMI パーミッション login を付与する必要があります。これを有効にするために、Application Server Control を使用しないで、OracleAS JAAS Provider Admintool を使用することもできます。次に例を示します。

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

関連項目：

- 付録 C「OracleAS JAAS Provider Admintool リファレンス」

Application Server Control でのファイルベース・プロバイダの構成

この項では、Application Server Control コンソールを使用して、ファイルベース・プロバイダを使用するアプリケーション用に行う次の管理タスクについて説明します。また、最後にインスタンス・レベルの管理に関する項があります。

- アプリケーション・デプロイ時のファイルベース・プロバイダの構成
- デプロイ後のファイルベース・プロバイダへの変更
- Application Server Control を介したアプリケーション・レルムの管理
- Application Server Control を介したアプリケーション・ユーザーの管理
- Application Server Control を介したロールの管理
- Application Server Control を介したインスタンス・レベル・セキュリティの管理

注意：

- この項で説明する手順を実行する前に、Application Server Control に、必要な管理パーミッションを持つユーザー（たとえば `oc4jadmin` など）としてログインしてください。
- セキュリティ・プロバイダ設定は、オプションでリポジトリ・ファイルの指定を含むことができ、`orion-application.xml` ファイルの `<jazn>` 要素の設定に影響します。レルム、ユーザーおよびロール設定により、リポジトリ・ファイルの `<jazn-realm>` 要素の中の設定が変更されます。

関連項目：

- この項で説明する手順の結果による XML 構成の例は、7-9 ページの「OC4J 構成ファイルにおけるファイルベース・プロバイダ設定」を参照してください。

アプリケーション・デプロイ時のファイルベース・プロバイダの構成

ファイルベース・プロバイダを指定できるのは、Application Server Control を介してアプリケーションをデプロイするときです。オプションで、`jazn-data.xml` ファイルの場所とデフォルト・レルムの指定もできます。

「デプロイ : デプロイ設定」ページで、次の手順を実行します（このページへのナビゲート方法は、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください）。

1. 「セキュリティ・プロバイダの選択」タスクを選択します。
2. 表示される「デプロイ設定 : セキュリティ・プロバイダの選択」ページで、「セキュリティ・プロバイダ」ドロップダウン・リストから「ファイルベース」を選択します。
3. そのドロップダウンでファイルベース・プロバイダを選択すると表示される「ファイルベースのセキュリティ・プロバイダの構成」で、次の手順を実行します。
   - OC4J インスタンスのデフォルト・ファイルベース・プロバイダとアプリケーション固有のファイルベース・プロバイダのどちらを使用するかを選択します。
   - リポジトリの場所を指定します。また、オプションでユーザーおよびロール構成のためのアプリケーション固有 `jazn-data.xml` ファイルを指定します。デフォルトでは、`system-jazn-data.xml` ファイルが使用されます。
   - デフォルト・レルムを指定します。そうしない場合は、`jazn.com` がデフォルト・レルムになります（ただし、インスタンス・レベルの `jazn.xml` ファイルに別の設定がある場合は除きます）。
4. **「OK」**を選択し、セキュリティ・プロバイダの選択を終了します。
5. 「デプロイ : デプロイ設定」ページが再表示されるので、**「デプロイ」**を選択してデプロイを完了するか、または必要に応じて他のタスクを選択します。タスクのリストは、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください。

デプロイ後のファイルベース・プロバイダへの変更

アプリケーションで使用するセキュリティ・プロバイダは、前述のようにデプロイ時に選択できます。また、デプロイ後に、異なるセキュリティ・プロバイダに変更することもできます。次の手順でファイルベース・プロバイダに変更できます。

1. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、アプリケーションの「セキュリティ・プロバイダ」ページを表示します。
2. 「セキュリティ・プロバイダ」ページで、**「セキュリティ・プロバイダの変更」**を選択します。
3. 「セキュリティ・プロバイダの変更」ページで、「セキュリティ・プロバイダ・タイプ」ドロップダウンから「ファイルベースのセキュリティ・プロバイダ」を選択します。
4. 「セキュリティ・プロバイダ属性 : ファイルベースのセキュリティ・プロバイダ」（「ファイルベースのセキュリティ・プロバイダ」を選択すると表示されます）で、次の手順を実行します。
   - OC4J インスタンスのデフォルト・ファイルベース・プロバイダとアプリケーション固有のファイルベース・プロバイダのどちらを使用するかを選択します。
   - （オプション）リポジトリ・ファイル（アプリケーション固有の `jazn-data.xml` ファイルなど）の場所を指定します。そうしない場合は、`system-jazn-data.xml` ファイルが使用されます。

- （オプション）デフォルト・レルムを指定します。そうしない場合は、jazn.com がデフォルト・レルムになります（ただし、インスタンス・レベルの jazn.xml ファイルに別の設定がある場合は除きます）。

5. 「OK」を選択し、変更を終了します。

「セキュリティ・プロバイダ」ページが再表示されます。このページで設定を確認できます。変更を有効にするためにアプリケーションを再起動するよう指示されます。

Application Server Control を介したアプリケーション・レルムの管理

この項では、ファイルベース・プロバイダのレルムを構成する方法について説明します。

後述のどの手順の場合も最初の手順は、アプリケーションの Application Server Control コンソールの「セキュリティ・プロバイダ」ページに進むことです（6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」を参照）。

ここでのタスクにより、リポジトリ・ファイルの <jazn-realm> 要素の下においてサブ要素が作成または変更されます。<jazn-realm> の下には、レルムごとに <realm> サブ要素があります。

注意： レルムの編集タスクというものはありません。レルムの編集とは、ユーザーまたはロールあるいはその両方の更新のことですが、これらについては 7-5 ページの「Application Server Control を介したアプリケーション・ユーザーの管理」および 7-7 ページの「Application Server Control を介したロールの管理」に記載されています。

レルムの検索

レルムを検索するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. 「レルム」タブを選択します。
2. 「レルム」ページの「検索」で、検索文字列を指定してから「実行」を選択します。
3. 検索文字列に一致するレルムが、「結果」の下に表示されます。（検索文字列が空の場合は既存のレルムがすべて表示されます。）

レルムの作成

レルムを作成するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. 「レルム」タブを選択します。
2. 既存レルムのリストの上にある「作成」を選択します。
3. 表示される「レルムの追加」ページで次の手順を実行します。
   - 希望のレルム名を指定します。
   - レルムの管理ユーザーに対する希望の名前を指定します。
   - 管理ユーザーに対する希望のパスワードを指定して確認します。
   - レルムについて希望の管理者ロールを指定します。指定した管理ユーザーは、このレルムに属することになります。
4. 「OK」を選択してレルムを作成します。

「セキュリティ・プロバイダ」ページが再表示されます。このページでレルム・リスト内の新しいレルムを確認できます。

レルムの削除

レルムを削除するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. 既存レルムのリストで、削除するレルムについて「削除」タスクを選択します。
2. 表示される「確認」ページで**「はい」**を選択してレルムを削除します。

「セキュリティ・プロバイダ」ページが再表示されます。

Application Server Control を介したアプリケーション・ユーザーの管理

この項では、ファイルベース・プロバイダのユーザーを構成する方法について説明します。

後述のどの手順の場合も最初の手順は、アプリケーションの Application Server Control コンソールの「セキュリティ・プロバイダ」ページに進むことです（6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」を参照）。

ここでのタスクにより、リポジトリ・ファイルの `<users>` 要素の下においてサブ要素が作成または変更されます。各 `<realm>` 要素には、そのレルムに含まれるユーザー用に `<users>` サブ要素があります。

ユーザーの検索

ユーザーを検索するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ユーザー」および当該レルムの行で、レルムに定義するユーザー数を選択します。これにより、レルムの「ユーザー」ページが表示されます。
3. 「ユーザー」ページの「検索」で、検索文字列を指定してから**「実行」**を選択します。
4. 検索文字列に一致するユーザーが、「結果」の下に表示されます。（検索文字列が空の場合はレルム内のユーザーがすべて表示されます。）

ユーザーの作成

ユーザーを作成するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ユーザー」および当該レルムの行で、レルムに定義するユーザー数を選択します。これにより、レルムの「ユーザー」ページが表示されます。
3. 「ユーザー」ページのレルム内の既存ユーザー・リストの上にある**「作成」**を選択します。
4. 表示される「ユーザーの追加」ページで次の手順を実行します。
   - 希望のユーザー名を指定します。
   - ユーザーに対する希望のパスワードを指定して確認します。
   - 「ロールの割当て」で、ユーザーの所属先として利用可能な希望のロール名を、「選択したロール」列に移動します。
   - **「OK」**を選択してユーザーを追加します。

「ユーザー」ページが再表示されます。このページでユーザー・リスト内の新しいユーザーを確認できます。

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。

ユーザーの削除

ユーザーを削除するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ユーザー」および当該レルムの行で、レルムに定義するユーザー数を選択します。これにより、レルムの「ユーザー」ページが表示されます。
3. 「ユーザー」ページで、削除するユーザーについて「削除」タスクを選択します。
4. 表示される「確認」ページで**「はい」**を選択してユーザーを削除します。

「ユーザー」ページが再表示されます。

ユーザーの編集

ユーザーのプロパティを編集するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ユーザー」および当該レルムの行で、レルムに定義するユーザー数を選択します。これにより、レルムの「ユーザー」ページが表示されます。
3. 「ユーザー」ページで、編集するユーザーを選択します。
4. 表示される「ユーザー」ページで次の手順を実行します。
   - ユーザー・パスワードを変更するには、旧パスワードを入力してから希望の新パスワードを入力して確認します。
   - ユーザーにロールを追加するかまたはユーザーからロールを削除する場合は、「ロールの割当て」で、「選択したロール」列においてロール名を希望にあうように挿入または削除します。
   - **「適用」**を選択してユーザーを変更します。

「ユーザー」ページが再表示されます。

注意： 所定のユーザーの「ユーザー」ページには、ユーザーが属するロールの「ロール」ページからもアクセスできます（7-8 ページの「ロールの編集」を参照してください）。「ロール」ページの「ユーザー」で、目的のユーザーを選択します。

Application Server Control を介したロールの管理

この項では、ファイルベース・プロバイダのロールを構成する方法について説明します。

後述のどの手順の場合も最初の手順は、アプリケーションの Application Server Control コンソールの「セキュリティ・プロバイダ」ページに進むことです（6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」を参照）。

ここでのタスクにより、リポジトリ・ファイルの <roles> 要素の下においてサブ要素が作成または変更されます。各 <realm> 要素には、そのレルムに含まれるロール用に <roles> サブ要素があります。

ロールの検索

ロールを検索するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ロール」、および当該レルムの行で、レルムに定義するロール数を選択します。これにより、レルムの「ロール」ページが表示されます。
3. 「ロール」ページの「検索」で、検索文字列を指定してから**「実行」**を選択します。
4. 検索文字列に一致するロールが、「結果」の下に表示されます。（検索文字列が空の場合はレルム内のロールがすべて表示されます。）

ロールの作成

ロールを作成するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ロール」、および当該レルムの行で、レルムに定義するロール数を選択します。これにより、レルムの「ロール」ページが表示されます。
3. 「ロール」ページのレルム内の既存ユーザー・リストの上にある**「作成」**を選択します。
4. 表示される「ロールの追加」ページで次の手順を実行します。
   - 希望のロール名を指定します。
   - ロール（実際には、ロールに属するユーザーやその他のエンティティ）に付与するパーミッション、つまり RMI パーミッション、管理パーミッションまたはこれらの両方とも付与するか、あるいはこれらのどちらも付与しないかを選択します。

     ユーザーが Remote Method Invocation（RMI）を介して（たとえばリモート EJB クライアントから）OC4J 上でオブジェクトにアクセスする場合、RMI パーミッションが必要になります。

     ユーザーは、起動、停止、構成変更などの管理機能を実行するには、管理パーミッションが必要です。
   - 「ロールの割当て」で、新しいロールの継承元として利用可能な希望のロール名を、「選択したロール」列に移動します。
   - **「OK」**を選択してロールを追加します。

「ロール」ページが再表示されます。このページでロール・リスト内の新しいロールを確認できます。

ロールの削除

ロールを削除するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ロール」、および当該レルムの行で、レルムに定義するロール数を選択します。これにより、レルムの「ロール」ページが表示されます。
3. 「ロール」ページで、削除するロールについて「削除」タスクを選択します。
4. 表示される「確認」ページで**「はい」**を選択してロールを削除します。

「ロール」ページが再表示されます。

ロールの編集

ロールのプロパティを編集するには、アプリケーションの「セキュリティ・プロバイダ」ページから、次の手順を実行します。

1. **「レルム」**タブを選択します。
2. 「レルム」ページの、レルム・リスト内の「ロール」、および当該レルムの行で、レルムに定義するロール数を選択します。これにより、レルムの「ロール」ページが表示されます。
3. 「ロール」ページで、編集するロールを選択します。
4. 表示される「ロール」ページで次の手順を実行します。
   - 必要に応じて、RMI パーミッションおよび管理パーミッションを選択または選択を解除して、ロールのパーミッションを更新します。
   - 「ロールの割当て」で、このロール（編集しているロール）の継承元にする希望のロール名を、「選択したロール」列において挿入または削除します。
   - **「適用」**を選択してロールを変更します。

「ロール」ページが再表示されます。

関連項目：

- ロールにユーザーを追加する方法については、7-6 ページの「ユーザーの編集」を参照してください。

Application Server Control を介したインスタンス・レベル・セキュリティの管理

OC4J インスタンス・レベルのファイルベース・セキュリティ・プロバイダ用に、レルム、ユーザー、ロールを構成できます。このようにして行われた変更は、アプリケーション・レベルの `jazn-data.xml` ファイル（指定されている場合）ではなく、常に `system-jazn-data.xml` ファイルに影響します。

（インスタンス・レベルのファイルベース・プロバイダは、OC4J の `system-application.xml` ファイルの `<jazn>` 要素の設定に従って、`system-jazn-data.xml` として指定されます。）

インスタンス・レベルのファイルベース・プロバイダは、アプリケーション用のファイルベース・プロバイダとほぼ同じ方法で管理できます。Application Server Control コンソールの「インスタンス・レベルのセキュリティ」ページには、次の手順でナビゲートできます。

1. OC4J インスタンスの OC4J ホームページで、**「管理」**タブを選択します。
2. 「管理」ページで、「セキュリティ・プロバイダ」タスク（「セキュリティ」の下）を選択します。
3. 「セキュリティ・プロバイダ」ページで、**「インスタンス・レベルのセキュリティ」**を選択します。

4. 表示される「インスタンス・レベルのセキュリティ」ページで、インスタンス・レベルのレルム、ユーザーおよびロールを管理できます。その手順は、この章の 7-4 ページの「Application Server Control を介したアプリケーション・レルムの管理」、7-5 ページの「Application Server Control を介したアプリケーション・ユーザーの管理」および 7-7 ページの「Application Server Control を介したロールの管理」で説明されている手順と基本的に同じです。

注意： OC4J には、`system-application.xml` および `system-jazn-data.xml` のインスタンス・レベル・セキュリティ・プロバイダ設定への依存性があります。たとえば、`admin_client.jar` では `system-jazn-data.xml` のアカウントが使用されています。これらのファイルにあるインスタンス・レベルのセキュリティ・プロバイダおよび関連アカウントに関するデフォルト設定を削除または変更しないでください。

OC4J 構成ファイルにおけるファイルベース・プロバイダ設定

この項では、主要な OC4J 構成ファイルにおける、ファイルベース・プロバイダに関する重要なセキュリティ構成について、参考情報を提供します。通常の場合、構成ファイルを直接操作するかわりに、Application Server Control コンソール（この章で前述）を使用して構成と管理を行う必要があります。このツールを使用することにより、適切なエントリが構成ファイルに自動的に設定されます。

この項の以降の部分で、次の項目について説明します。

- ファイルベース・プロバイダに対する <jazn> 要素の設定
- リポジトリ・ファイルのレルム構成
- リポジトリ・ファイルのポリシー構成
- system-jazn-data.xml の事前定義済 OC4J アカウント

ファイルベース・プロバイダに対する <jazn> 要素の設定

（`jazn.xml` ファイルと `orion-application.xml` ファイルの両方にある）`<jazn>` 要素には、セキュリティ・プロバイダ、リポジトリおよびデフォルト・レルムの構成を指定します。デフォルトでは、`system-jazn-data.xml` ファイルは、ファイルベース・プロバイダのユーザー、ロールおよびポリシー構成のリポジトリですが、OC4J は、このファイルのかわりにアプリケーション固有の `jazn-data.xml` ファイルを使用するよう構成できます。

この項の内容は次のとおりです。

- orion-application.xml の <jazn> 設定のシナリオ
- アプリケーション固有の jazn-data.xml ファイルを自動作成するための構成
- アプリケーション固有の jazn-data.xml ファイルの供給

orion-application.xml の <jazn> 設定のシナリオ

アプリケーションには、3 つの代表的なデプロイ・シナリオがあります。これらのシナリオは、ファイルベース・プロバイダの使用時に、`orion-application.xml` ファイルおよびインスタンス・レベルの `jazn.xml` ファイルの `<jazn>` 要素の設定によって分類されるものです。

- リポジトリとデフォルト・レルムについては、インスタンス・レベルの `jazn.xml` ファイルに委任します。`jazn.xml` の `<jazn>` 要素に `provider="XML"` 設定がある場合に、`orion-application.xml` ファイルに次の `<jazn>` 要素があるときには、このファイルのリポジトリ設定（`location` 属性）およびデフォルト・レルム設定（`default-realm` 属性）が使用されます。

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

または、jazn.xml ファイルに location および default-realm 設定がない場合は、このファイルはデフォルト・リポジトリ system-jazn-data.xml とデフォルト・レルム jazn.com を使用します。

注意： アプリケーションのデプロイ時に、orion-application.xml に <jazn> 要素がない場合は、これがデフォルトの <jazn> 設定になります。

- リポジトリについては、インスタンス・レベルの jazn.xml ファイルに委任します。jazn.xml の <jazn> 要素に provider="XML" 設定があるが、orion-application.xml ファイルに次のような <jazn> 要素があるときには、このファイルのリポジトリ設定（location 属性）は使用されますが、デフォルト・レルムに関しては、orion-application.xml ファイルの設定（default-realm 属性）が使用されます。

  ```
  <jazn provider="XML" default-realm="abc.com" />
  ```

  または、jazn.xml ファイルに location 設定がない場合は、このファイルはデフォルト・リポジトリ system-jazn-data.xml を使用します。

  注意： この例では、abc.com レルムが system-jazn-data.xml リポジトリで定義されているものと想定しています。

- 委任しません。リポジトリとデフォルト・レルムを orion-application.xml で指定します。次の例では、orion-application.xml で、リポジトリ jazn-data.xml とデフォルト・レルム myrealm を指定しています。

  ```
  <jazn provider="XML" location="./jazn-data.xml" default-realm="myrealm" />
  ```

  注意： アプリケーションでファイルベース・プロバイダ（orion-application.xml の provider="XML"）が使用されているが、jazn.xml ファイルに provider="LDAP" 設定がある状況では、次の点に注意してください。

  - orion-application.xml でリポジトリ・ファイルが指定されていない場合は、system-jazn-data.xml がリポジトリになります。
  - orion-application.xml でデフォルト・レルムが指定されていない場合は、jazn.com ファイルがデフォルト・レルムになります。

アプリケーション固有の jazn-data.xml ファイルを自動作成するための構成

orion-application.xml が次のように構成されている状態で、jazn-data.xml ファイルがアプリケーションとともにパッケージ化されていない場合は、このファイルがデプロイ時に作成されます。

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

アプリケーション固有の jazn-data.xml ファイルの供給

アプリケーションに jazn-data.xml ファイルを供給する場合は、アプリケーションの orion-application.xml ファイルにある <jazn> 要素の location 属性にその位置を指定します。次に例を示します。

1. orion-application.xml で次のように指定します。

   ```
   <jazn provider="XML" location="./jazn-data.xml" default-realm="myrealm" />
   ```

   相対的な場所を指定する場合は、<jazn> 要素が格納されている orion-application.xml ファイルの場所を基準とします。通常これは、アプリケーション EAR ファイルの /META-INF ディレクトリになります。

2. EAR ファイルの /META-INF ディレクトリにある jazn-data.xml ファイルをパッケージ化します。

リポジトリ・ファイルのレルム構成

この項では、system-jazn-data.xml ファイルにおける jazn.com レルム用のユーザーとロールの構成を示します。一般的な構造は、system-jazn-data.xml または jazn-data.xml ファイルにあるレルム構成の構造と同じです。この構成は、Application Server Control を介してレルムを管理するときに自動的に作成されます。

```
<jazn-realm>
  <realm>
    <name>jazn.com</name>
    <users>
      <user deactivated="true">
        <name>anonymous</name>
        <description>The default guest/anonymous user</description>
      </user>
      <user deactivated="true">
        <name>oc4jadmin</name>
        <display-name>OC4J Administrator</display-name>
        <description>OC4J Administrator</description>
        <credentials>!welcome</credentials>
      </user>
      <user>
        <name>JtaAdmin</name>
        <display-name>JTA Recovery User</display-name>
        <description>Used to recover propagated OC4J transactions</description>
        <credentials>!defaultJtaPassword</credentials>
      </user>
    </users>
    <roles>
      <role>
        <name>oc4j-administrators</name>
        <display-name>OC4J Admin Role</display-name>
        <description>Administrative role for OC4J</description>
        <members>
          <member>
            <type>user</type>
            <name>oc4jadmin</name>
          </member>
          <member>
            <type>user</type>
            <name>JtaAdmin</name>
          </member>
        </members>
      </role>
      <role>
        <name>oc4j-app-administrators</name>
        <display-name>OC4J Application Administrators</display-name>
```

```
        <description>OC4J application-level administrators</description>
        <members>
        </members>
      </role>
      <role>
        <name>users</name>
        <display-name>users</display-name>
        <description>users role for rmi/ejb access</description>
        <members>
        </members>
      </role>
    </roles>
  </realm>
</jazn-realm>
```

リポジトリ・ファイルのポリシー構成

OracleAS JAAS Provider Admintool で、-grantperm オプションを使用してカスタム・プリンシパルに JAAS パーミッションを付与できます（C-15 ページの「パーミッションの付与と取消し」を参照）。

ポリシー・データは、system-jazn-data.xml に格納されます。次の例はこのファイルの一部分で、RMI パーミッション login を admin プリンシパルに付与した結果を示しています。（この例では、admin が jazn.com レルムに属するユーザーであると想定しています。）

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

system-jazn-data.xml の事前定義済 OC4J アカウント

次のアカウントは、ファイルベース・プロバイダ用に system-jazn-data.xml に事前定義されています。

- oc4jadmin ユーザー（スタンドアロン OC4J では初期段階では解除されています。）
- oc4j-administrators ロール（oc4jadmin をメンバーとして含んでいます。）
- oc4j-app-administrators ロール
- ascontrol_admin ロール（oc4jadmin をメンバーとして含んでいます。）
- ascontrol_appadmin ロール
- ascontrol_monitor ロール
- anonymous ユーザー（初期段階では解除されています。）
- users ロール
- jtaadmin ユーザー

関連項目：

- これらのアカウントの詳細は、4-12 ページの「事前定義アカウント」を参照してください。
- 4-13 ページの「oc4jadmin アカウントのアクティブ化（スタンドアロンの OC4J）」

OracleAS JAAS Provider 移植ツール

OC4J には、ファイルベース・リポジトリから Oracle Internet Directory リポジトリまたは代替ファイルベース・リポジトリに移植するためのツールが用意されています。（principals.xml からの移植に使用するツールと混同しないでください。そのツールは別のもので、この章で後述します。）

Oracle Internet Directory リポジトリに移植する際には、出力は LDIF ファイルになり、この LDIF ファイルを ldapmodify や bulkload などのコマンドにより Oracle Internet Directory にインポートします。

この項の内容は次のとおりです。

- 移植ツールの概要
- 移植ツールのコマンドの構文
- 移植ツールの API

移植ツールの概要

移植ツールでは、ユーザー、ロール、ロール・メンバーシップおよびポリシーの移植がサポートされます（ロール、ユーザー、カスタム・プリンシパルまたはコードベースにパーミッションが付与されます）。

移植には次の 3 つのモードがあります。

- レルム・モード。ユーザーとロールのみが移植されます。ソース・レルム内のすべてのユーザー（非アクティブなユーザーを除く）とロールが移植されます。移植されたロールにはメンバーシップ情報が含まれます。
- ポリシー・モード。権限受領者および権限受領者に付与されたパーミッションが移植されます。権限受領者には、ユーザーやロールなどのレルムの権限受領者、またはカスタム・プリンシパルやコードベースなどの非レルムの権限受領者を指定できます。Oracle Internet Directory に移植する際、レルムの権限受領者とそのパーミッションは宛先レルム固有のポリシーに移植されますが、非レルムの権限受領者とそのパーミッションはグローバル・ポリシーに移植されます。
- 全モード。レルム・モードとポリシー・モードを組み合せたものです。

注意： 移植ツールを使用する場合は、次の点に注意が必要です。

- Oracle Internet Directory に移植するには、ツール実行時にディレクトリ・サーバーが稼働していて使用可能な状態になっている必要があります。
- LDIF ファイルへの出力が生成された時点では、パスワードはクリアテキストです。この情報の保護に適切な注意を払うのは読者の責任です。
- Oracle Internet Directory に移植する際、パスワードを Oracle Internet Directory 要件（少なくとも 1 文字は数値が含まれるなど）に準拠するよう変更することが必要になることがあります。
- ポリシー・モードまたは全モードの場合 : 1）非レルムのカスタム・プリンシパルのパーミッションを移植する場合は、そのカスタム・プリンシパルのクラス・ファイルを含んだ JAR ファイルをクラスパスに追加する必要があります。2）カスタム・パーミッションを移植する場合は、カスタム・パーミッションのクラス・ファイルを含んだ JAR ファイルをクラスパスに追加する必要があります。
- 移植ツールは、Oracle Internet Directory への間接パスワード・アカウントの移植はできません。
- 競合の可能性に注意してください。移植対象のユーザーとロールが、宛先レルム内にすでに存在していることがあります。Oracle Internet Directory に移植する際、たとえば、`ldapmodify` や `bulkload` などのコマンドを標準の JDK ロギングと組み合せて使用し、競合からのリカバリに役立つ情報を取得できます。

移植ツールのコマンドの構文

移植ツールのコマンドラインの構文とオプションは次のとおりです。

```
% java JAZNMigrationTool [-st xml] [-dt ldap|xml]
                         [-D binddn] [-w passwd] [-h ldaphost] [-p ldapport]
                         [-sf sourcefilename] [-df destfilename]
                         [-sr source_realm] [-dr dest_realm]
                         [-m policy|realm|all]
                         [-help]
```

これらのオプションについて、表 7-1 で説明します。

表 7-1 OracleAS JAAS Provider 移植ツールのオプション

オプション	説明	デフォルト（存在する場合）
-help	オプション情報の表示。	
-st	ソース側のプロバイダのタイプ。 現在サポートされている設定は xml のみです。これはファイルベース・プロバイダからの移植用です。	xml
-dt	宛先側のプロバイダのタイプ。xml（ファイルベース・プロバイダへの移植の場合）または ldap（Oracle Internet Directory への移植の場合）。	ldap
-D	Oracle Internet Directory ユーザー名（Oracle Internet Directory への移植の場合のみ）。	
-w	Oracle Internet Directory ユーザー・パスワード（Oracle Internet Directory への移植の場合のみ）。	
-h	Oracle Internet Directory ホスト・システム（Oracle Internet Directory への移植の場合のみ）。	jazn.xml にある location 設定の <jazn> 要素により決定されます。
-p	Oracle Internet Directory ポート（Oracle Internet Directory への移植の場合のみ）。	jazn.xml にある location 設定の <jazn> 要素により決定されます。
-sf	ソース・ファイル。つまり、移植元のファイルベース・リポジトリへのパス。	*ORACLE_HOME*/j2ee/home/config/system-jazn-data.xml
-df	宛先ファイル。つまり、LDIF 出力ファイル（Oracle Internet Directory への移植の場合）または宛先ファイルベース・リポジトリ（ファイルベースへ移植の場合）へのパス。	ファイルベース・リポジトリへの移植の場合は *ORACLE_HOME*/j2ee/home/config/system-jazn-data.xml（そうでない場合はデフォルトなし）。
-sr	ソース・レルム。つまり、移植元のレルム。	ソース・リポジトリのレルムの名前です（レルムが 1 つしかない場合）。
-dr	宛先レルム。つまり、移植先のレルム。	ファイルベース・リポジトリに移植する場合は、宛先リポジトリのレルム名（レルムが 1 つのみのとき）。Oracle Internet Directory に移植する場合は、デフォルトのサブスクライバ・レルム。
-m	希望の移植モード。つまり、レルム・モード（realm）、ポリシー・モード（policy）、または両方（all）。	all

次の例では、all モードで、指定されたホストの Oracle Internet Directory にあるデフォルトのサブスクライバ・レルムに移植しています。

```
% java oracle.security.jazn.tools.JAZNMigrationTool -D cn=orcladmin -w welcome1 \
       -h myhost.example.com -p 389 -sf /tmp/jazn-data.xml -df /tmp/dest.ldif \
       -sr jazndemo.com
```

移植ツールの API

移植ツール（パッケージ oracle.security.jazn.tools 内のクラス JAZNMigrationTool）はアプリケーションから起動することもできます。Oracle では次の API を用意しています。

```
/**
 * Create an instance with the provided parameters. These parameters are
 * equivalent to the options supported by the executable utility version.
 */
public JAZNMigrationTool(Map params)

/**
 * Perform the migration operation
 */
public  void migrateData() throws JAZNException
```

このコンストラクタ内の params パラメータでは、前項の表 7-1 に記載されているものと同じオプションと同じデフォルトがサポートされています。パラメータ・キーは、定数として JAZNMigrationTool クラスに定義されています。表 7-2 に、JAZNMigrationTool で定義されている定数とコマンドライン・オプション間の相関関係を示します。

表 7-2 JAZNMigrationTool 定数

キー定数	対応するオプション
SRC_TYPE	-st
DEST_TYPE	-dt
OID_USER	-D
OID_PASSWORD	-w
OID_HOST	-h
OID_PORT	-p
SRC_FILE	-sf
DEST_FILE	-df
SRC_REALM	-sr
DEST_REALM	-dr
MIGRATE_OPT	-m

principals.xml ファイルからのプリンシパルの移植

非推奨の principals.xml ファイルからデータを移植するには、OracleAS JAAS Provider Admintool の convert オプションを使用します。

-**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 のユーザーがデプロイ・ユーザーに、principals.xml のグループがデプロイ・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、すべてデプロイ・ロールにマップされます。移植時にアクティブになっ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。

注意： principals.xml ファイルは非推奨です。（リリース 11*g* ではサポートされない予定です。）

---

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にされない管理ユーザーをアクティブにします。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 オプション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。7-9 ページの「ファイルベース・プロバイダに対する <jazn> 要素の設定」を参照してください。
5. OC4J を停止して再起動します。

OC4J グループでのファイルベース・プロバイダの使用

OC4J 10.1.3.1 実装には、OC4J インスタンスをグループに含めるための機能が追加されています(それまでは、グループに入れることができたのは名前が同じであるインスタンスのみでした)。

これらの機能と OC4J `J2EEServerGroup` MBean を使用して、グループ内の各 OC4J インスタンスの `system-jazn-data.xml` ファイルに対する変更の整合性を図ることができます。

OC4J の基本的なグループ機能

OC4J クラスタでは、次のように Application Server Control を介して新しいグループを作成できます。

1. 「クラスタ・トポロジ」ページの「グループ」で、**「作成」**を選択します。
2. 「グループの作成」ページで、次の作業を行います。
   a. 希望のグループ名を指定します。
   b. このグループに移動する OC4J インスタンスを選択します。OC4J インスタンスを新しいグループに移動すると、移動前に属していたグループからそのインスタンスが削除されること、およびインスタンスは停止してからでないと移動できないことに注意してください。

   重要: OC4J インスタンスを停止する場合は、インスタンスをホスティングするアプリケーション・サーバーで他の OC4J インスタンスが 1 つ以上実行されている必要があります。OC4J インスタンスを停止するチェック・ボックスが使用不可になっている場合は、そのアプリケーション・サーバーで他の OC4J インスタンスが実行されていません。

   ---

   c. **「作成」**を選択します。

   注意: グループ作成後に OC4J インスタンスをそのグループに移動することもできます。

   1. 「クラスタ・トポロジ」ページの「グループ」で、対象のグループを選択します。
   2. 「グループ : *groupname*」ページの「OC4J インスタンス」タブで、**「追加」**を選択します。
   3. 「OC4J インスタンスをグループに追加」ページで、必要に応じて OC4J インスタンスを選択し、グループに追加します。

次のように、Application Server Control を介してグループを管理できます。

1. 「クラスタ・トポロジ」ページの「グループ」で、対象のグループを選択します。
2. **「管理」**タブを選択します。
3. 「管理」ページに、グループ全体に対する管理機能が表示されます。ただし、セキュリティ・プロバイダを管理する機能は含まれていないので注意してください。

関連項目:

- OC4J グループ機能の追加情報は、Application Server Control オンライン・ヘルプのグループ OC4J インスタンスのページに関するトピックを参照してください。

クラスタ MBean ブラウザの機能および J2EEServerGroup MBean

グループを作成して OC4J インスタンスを移入した後に、クラスタ MBean ブラウザを使用して、グループ内の各 `system-jazn-data.xml` ファイルに対して設定の整合性をとることができます。この場合、グループの `J2EEServerGroup` MBean で操作を起動します。これには、`system` アプリケーションの `J2EEApplication` MBean も関係します。

この作業は次の手順で行います。

1. 「クラスタ・トポロジ」ページの「グループ」で、対象のグループを選択します。
2. 「グループ : *groupname*」ページで、「**管理**」タブを選択します。
3. 「管理」ページの「JMX」で、「クラスタ MBean ブラウザ」タスクに移動します。
4. 「クラスタ MBean ブラウザ」ページで、次の作業を行います。
   a. `J2EEServerGroup` MBean で、OC4J グループを選択します。
   b. **「操作」**タブを選択します。
   c. `invoke` 操作を選択します。
5. 「操作 : invoke」ページ（この手順の終わりにある図 7-1 を参照）で、懐中電灯のアイコンを使用して MBean を名前で検索します。
6. 「検索と選択 : MBean」ページ（この手順の終わりにある図 7-2 を参照）で、次の作業を行います。
   a. SecurityProvider という MBean 名で検索します。
   b. 検索結果から、`J2EEApplication=system` となっている結果を選択します（図の一番下に示されている結果）。
7. 操作 : invoke ページを再表示して、次の作業を行います。
   a. パラメータ `operationName` に対し、ドロップダウン・メニューから目的の操作を選択します。`J2EEApplication` MBean に対して選択できる操作には、`addUser`、`addRole`、`remUser`、`remRole`、`revokeUserRole`、`revokeUserPerm`、`grantUserPerm`、`grantRolePerm` など、多数の操作があります。
   b. 鉛筆アイコンを使用して、操作に対するパラメータ設定を指定します。たとえば `addUser` に対するパラメータには、`username`、`passwd`、`realm` があります。
   c. パラメータの編集ページで、必要な設定を指定します。
8. 「操作 : invoke」ページを再表示して、**「起動操作」**を選択します。

操作の結果が、グループ内のすべてのインスタンスの `system-jazn-data.xml` ファイルに適用されます。

図 7-1 操作 : invoke

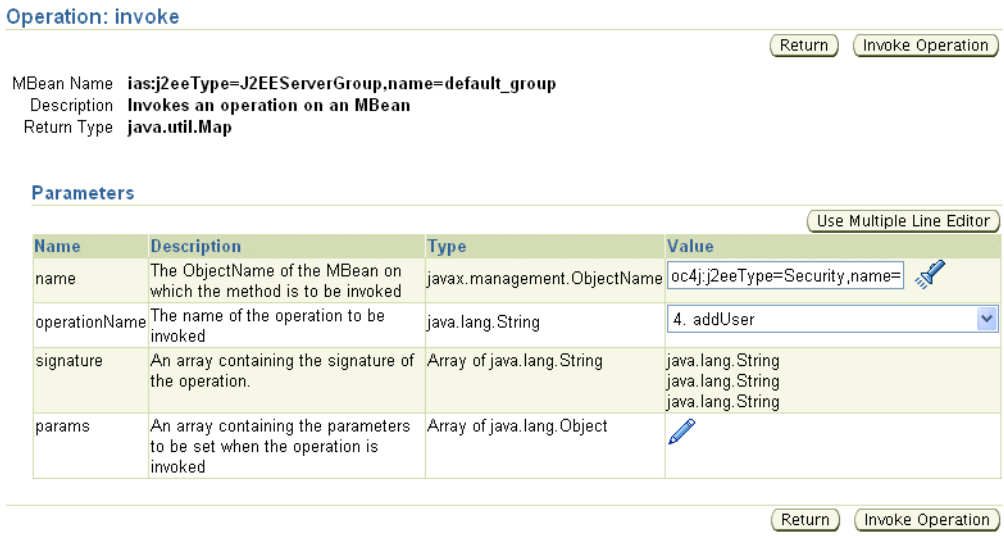
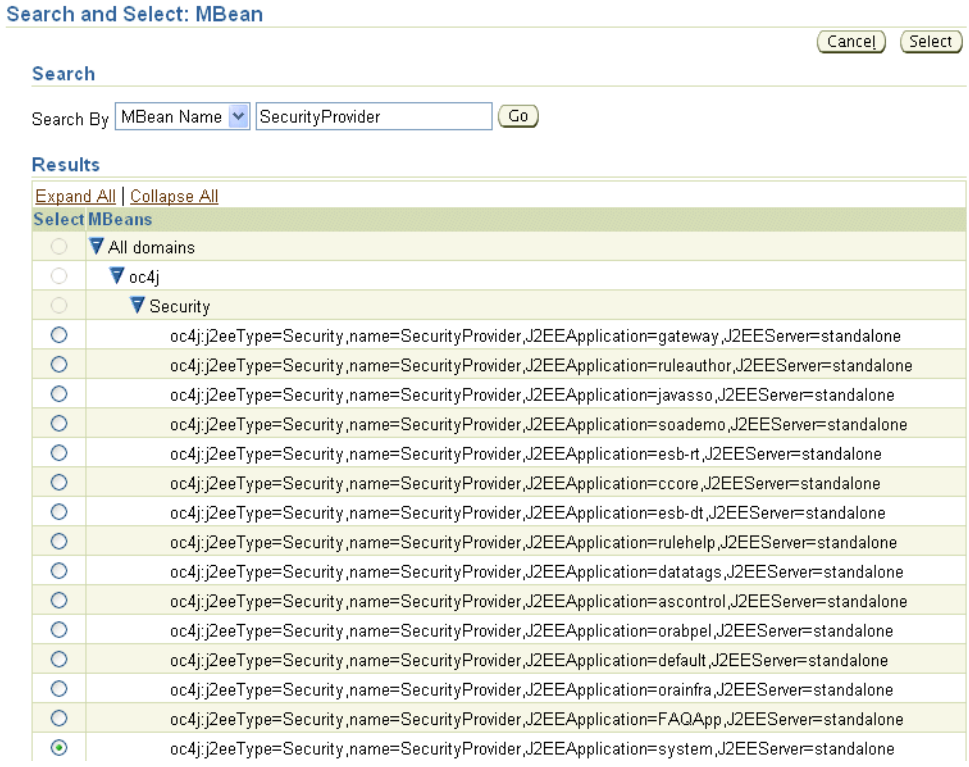


図 7-2「検索と選択 : MBean」ページ



8

Oracle Identity Management

Oracle Application Server では、Oracle Internet Directory および（オプションで）Oracle Single Sign-On を使用する Oracle Identity Management が、LDAP ベースのセキュリティ・プロバイダになります。

この章では、Oracle Identity Management をセキュリティ・プロバイダとして使用しているかまたは使用を計画している読者を対象として、Oracle Identity Management と OC4J の統合について説明します。内容は次のとおりです。

OC4J による Oracle Identity Management のサポートに関する初期の注意事項

OC4J で Oracle Identity Management を使用する場合は、次の点に注意してください。

- OC4J 10.1.3.x 実装から、LDAP ベース・プロバイダは、スタンドアロンの OC4J でも、Oracle Application Server 環境の場合と同様にサポートされるようになりました。
- OracleAS JAAS Provider は、Oracle Internet Directory でのアイデンティティ管理レルム・タイプをサポートします。（このレルム・タイプは、8-18 ページの「Oracle Identity Management の OracleAS JAAS Provider レルムの概要」で説明されています。）外部レルム・タイプおよびアプリケーション・レルム・タイプは非推奨です。具体的には、パッケージ `oracle.security.jazn.realm` の場合、`APPLICATION_REALM` および `EXTERNAL_REALM` は、`RealmType` クラスで非推奨であり、`_extRealm` および `_appRealm` は、`InitRealmInfo` クラスで非推奨です。外部レルムおよびアプリケーション・レルムは、今後のリリースではサポート対象外になります。
- Oracle Internet Directory におけるユーザーおよびロールの管理は、このドキュメントの対象外です。『Oracle Identity Management 委任管理ガイド』を参照してください。
- OC4J には、LDAP サーバーに対して使用するログイン・モジュール `LDAPLoginModule` があります。ただし、Oracle Internet Directory に対しては、デフォルトの `RealmLoginModule` を使用します。（Oracle Internet Directory に対して `LDAPLoginModule` を使用するように構成すると、本来備わっている最適化状態と統合機能が失われることになります。）

注意：

- LDAP ベース・プロバイダでは、認可のためのロール比較では大 / 小文字が区別されない点に注意してください。
- 8-6 ページの「Oracle Internet Directory と OC4J の関連付け」の説明に従って Oracle Internet Directory を OC4J と関連付けている場合は、Oracle Internet Directory においてユーザー・アカウントを追加または変更した後であっても、OC4J を再起動することなくログインできます。

関連項目：

- ファイルベース・リポジトリから Oracle Internet Directory リポジトリへの移植の詳細は、7-13 ページの「OracleAS JAAS Provider 移植ツール」を参照してください。
- Oracle Internet Directory で使用可能なユーザーおよびロール API の詳細は、第 12 章「ユーザーおよびロール API フレームワーク」を参照してください。
- `oc4jadmin` 以外の管理者アカウントを使用する場合は、4-13 ページの「新しい管理者アカウントの作成」を参照してください。

Oracle Identity Management の主要なコンポーネントの概要

Oracle Identity Management は、分散型エンタープライズ・アプリケーションを保護する企業インフラストラクチャとなります。これは、LDAP ベース Oracle Internet Directory、Oracle Single Sign-On、および追加のセキュリティとユーザー管理機能から構成される統合パッケージです。

Oracle Identity Management をセキュリティ・プロバイダとして使用するには、基盤となる Oracle Internet Directory および Oracle Single Sign-On について考慮する必要があります。この項では、これらの機能の概要について説明します。この項の内容は次のとおりです。

関連項目：

- 『Oracle Identity Management インフラストラクチャ管理者ガイド』
- 『Oracle Identity Management アプリケーション開発者ガイド』

Oracle Internet Directory の概要

Oracle Internet Directory には、次に示すような、Windows との統合機能、パスワード・ポリシー・オプション、部分レプリケーションなどの重要なセキュリティ機能が用意されています。

- Windows との統合機能 : Windows の Active Directory Service 用に事前構成された、ディレクトリ同期化ソリューションを提供します。この機能を使用すると、1 つのアイデンティティ / パスワード資格証明を、Oracle および Windows の両方の環境で使用できます。また、Windows 環境に保存されるパスワードの取得および変更をサポートするディレクトリ・プラグインも含まれています。これを使用することで、ユーザーは 2 つの環境間でのパスワード同期に関連したオーバーヘッドおよび潜在的なセキュリティ上の不安から解放されます。
- フレキシブルなパスワード・ポリシー : パスワード・ポリシーを選択できます。さらに、Oracle Internet Directory プラグインをサポートしており、ほぼ無制限にサイト固有の各種パスワード・ポリシーを実装できます。
- 部分レプリケーション : レプリケーション・モデルをサポートしており、大規模ネットワーク構成におけるスケーラビリティおよびパフォーマンスが向上します。
- その他の機能としては、動的グループのサポート、拡張 Oracle Internet Directory セルフ・サービス・コンソール、データベース表とのディレクトリ・データの同期支援、および Oracle E-Business Suite リリース 11*i* とのユーザー・アイデンティティ同期機能があります。

Oracle Internet Directory を OC4J 10.1.3.x 実装で使用する場合は、Basic、Digest、CLIENT-CERT、Username トークン、X.509 トークン、SAML トークンの各認証方式がサポートされます。

関連項目：

- 『Oracle Internet Directory 管理者ガイド』

識別名の概要

識別名または DN という用語を、この章では頻繁に使用します。これは標準 LDAP の概念です。DN は、カンマで区切られた 1 つ以上の相対識別名（RDN）のセットです。次のいずれかを RDN として使用できます。

- DC（ドメイン・コンポーネント）
- CN（一般名）
- OU（組織単位名）
- O（組織名）
- STREET（番地）
- L（市町村）
- ST（都道府県）
- C（国）
- UID（ユーザー ID）

この章では、一般名またはドメイン・コンポーネントで構成される RDN が最も多く使用されています。一般名の例としては、Jeff Smith や Oracle などがあげられます。

Oracle Single Sign-On の概要

Oracle Single Sign-On は、マルチレベル認証をサポートしています。複数の認証メカニズムを設定できます。また、シングル・サインオン対応アプリケーションに対するユーザーの認証方法を指示します。アプリケーション側では、認証方法に応じて、ユーザーに異なる権限を付与できます。

たとえば、パスワード認証の場合は、部分的な権限をユーザーに付与し、より強力な X.509v3 などの認証方法を使用する場合は、より完全な権限を付与できます。

グローバル・ログアウトおよびセッション・タイムアウトもサポートされています。

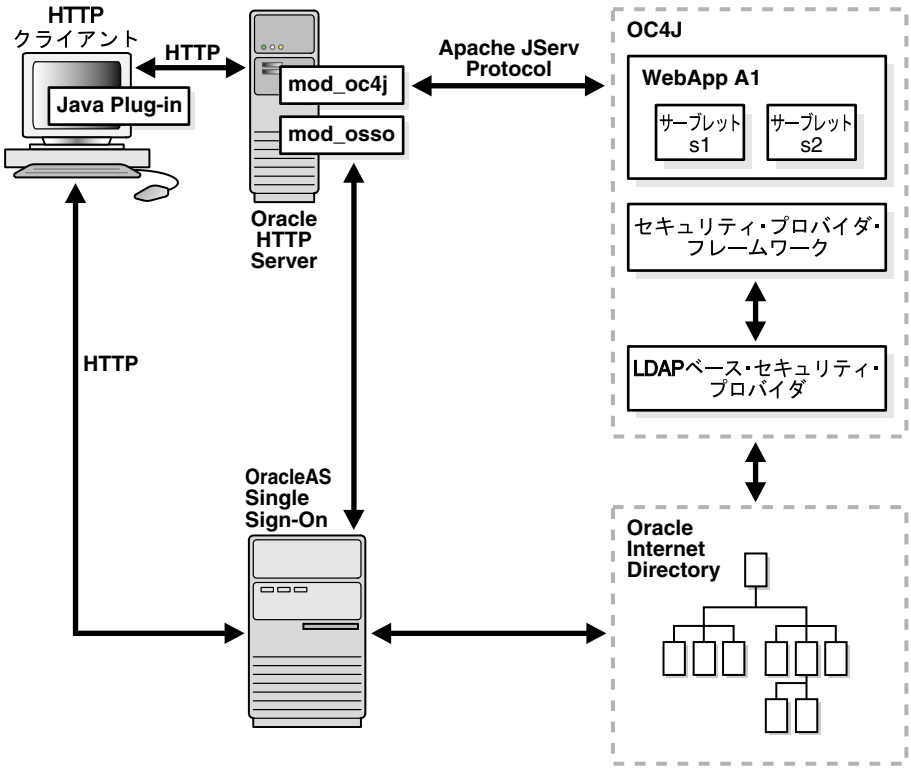
関連項目：

- 『Oracle Application Server Single Sign-On 管理者ガイド』
- 『Oracle Identity Management アプリケーション開発者ガイド』（特に、シングル・サインオン対応のアプリケーション開発に関する章）

SSO 対応の J2EE 環境 : 代表的な使用例

Oracle 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 8-1 に、SSO 対応の J2EE 環境で動作するアプリケーションにおける JAAS 統合を示します。

図 8-1 Oracle Single Sign-On と J2EE 環境



Oracle Single Sign-On 対応の J2EE 環境で HTTP クライアント・リクエストが開始された場合、Oracle コンポーネントにより、次の各ステップが実行されます。

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。Oracle HTTP Server（Apache リスナーを使用）がリクエストを処理します。
2. Oracle HTTP Server/`mod_osso` がリクエストを受信して、次の処理を実行します。
   - WebApp A1 アプリケーションで HTTP クライアントの認証用に Web ベースの Oracle Single Sign-On が必要かどうかを判別します。
   - HTTP クライアント・リクエストを Web ベースの Oracle 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。OracleAS Single Sign-On は、継続して次の処理を行います。
   - ユーザーについて格納されているログイン資格証明を検証します。
   - Oracle 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。
   - WebApp A1 アプリケーション（OC4J 内）にリダイレクトします。
4. セキュリティ・プロバイダが Oracle Single Sign-On ユーザーを取得します。

関連項目 :

- Oracle 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。

前提条件 : インフラストラクチャとしての Oracle Application Server

Oracle Identity Management は、Oracle Application Server インフラストラクチャの構成要素です。Oracle Identity Management をセキュリティ・プロバイダとして使用するには、適切なリリースをインフラストラクチャとしてインストールする必要があります。これは OC4J とは別の *ORACLE_HOME* に格納されます。

OC4J 10.1.3.1 実装で Oracle Identity Management を（Oracle Internet Directory とともに）OracleAS JAAS Provider 下でセキュリティ・プロバイダとして使用する場合、サポートされる Oracle Application Server インフラストラクチャのリリースは、10.1.2.0.1、10.1.2.0.2、10.1.4.x です。

Oracle Application Server インフラストラクチャのインストールの詳細は、各プラットフォームに対応する Oracle Application Server のインストレーション・ガイドを参照してください。

Oracle Identity Management セキュリティ・プロバイダを使用する手順

この項では、Oracle Identity Management をセキュリティ・プロバイダとして設定し、オプションで Oracle Single Sign-On を認証に使用するための手順を示します。

1. Oracle Internet Directory と OC4J の関連付け
2. SSO の構成（オプション）
3. Oracle Identity Management のセキュリティ・プロバイダとしての構成

Oracle Internet Directory と OC4J の関連付け

この項では、Oracle Internet Directory インスタンスと OC4J インスタンスの関連付けの手順を説明します。この手順は、Oracle Identity Management をセキュリティ・プロバイダとして指定する前に、実行する必要があります。また、対応する XML 構成も示します。この項目の内容は次のとおりです。

- Oracle Internet Directory と OC4J の関連付け
- Oracle Internet Directory の関連付けの変更
- Oracle Internet Directory に作成される必須アカウント
- jazn.xml における Oracle Internet Directory の関連付け
- Oracle Internet Directory を関連付ける際の複数の OC4J インスタンスについて

重要 :

- OC4J インスタンスを Oracle Internet Directory インスタンスに関連付けると、OC4J home インスタンスの jazn.xml ファイル内の <jazn> 要素の構成が書き換えられ、以前の設定がすべて失われます。
- Oracle Internet Directory インスタンスを OC4J インスタンスに関連付けると、jazn.xml ファイルの <jazn> 要素における provider 属性や location 属性などのインスタンス・レベルのプロバイダ設定が生成されます。アプリケーションを OC4J インスタンスにデプロイする場合にアプリケーションで別のプロバイダを構成すると、orion-application.xml で構成されたプロバイダが認証に使用されるアイデンティティ・ストアになり、jazn.xml で指定されたプロバイダが認可に使用されるポリシー・ストアになるという混在した状態になります。

Oracle Internet Directory と OC4J の関連付け

Application Server Control コンソールを使用して、Oracle Identity Management のリポジトリである LDAP ベースの Oracle Internet Directory（OID）のインスタンスに、OC4J インスタンスを関連付けます。手順は次のとおりです。

1. インスタンスの OC4J ホームページで、**「管理」**タブを選択します。
2. 表示される「管理」ページで、(「セキュリティ」タスクから)「アイデンティティ管理」タスクを選択します。
3. 表示される「アイデンティティ管理」ページで、**「構成」**を選択します。(この手順は、Oracle Internet Directory インスタンスと OC4J インスタンスの関連付けが以前に実行されておらず、Oracle Internet Directory ホスト名およびポートが「未構成」としてリストに表示される場合を想定しています。この OC4J インスタンスに対して、別の Oracle Internet Directory インスタンスを前に関連付けてある場合は、次の「Oracle Internet Directory の関連付けの変更」を参照してください。)
4. 表示される「アイデンティティ管理の構成 : 接続情報」ページで、次の手順を実行します。
   - Oracle Internet Directory インスタンスの完全修飾ホスト名（たとえば myoid.oracle.com）を指定します。
   - cn=orcladmin など、Oracle Internet Directory ユーザーの識別名を指定します（後述の注意を参照）。ここで指定するユーザーは、Oracle Internet Directory インスタンスの iASAdmins ロールに属している必要があります。
   - Oracle Internet Directory ユーザーのパスワードを指定します。これは、Oracle Internet Directory で作成される oc4jadmin ユーザーのデフォルト・パスワードとしても設定されます（ただし、別の OC4J インスタンスが Oracle Internet Directory インスタンスに関連付けられていたために oc4jadmin アカウントがあらかじめ作成されている場合は除きます）。
   - Oracle Internet Directory インスタンスとの接続に SSL 接続または非 SSL 接続のどちらを使用するかを指定します。また、使用する適切なポートを指定します。ポートは、SSL の場合は通常 636、非 SSL の場合は通常 389 を設定します。(SSL またはポートの設定を後で変更するには、OC4J と OID の関連付けを再実行する必要があります。この手順は、次の「Oracle Internet Directory の関連付けの変更」で説明します。)
   - 作業終了後、次のページに進みます。
5. 「アイデンティティ管理の構成 : Application Server Control」ページで、Oracle Identity Management を Application Server Control のセキュリティ・プロバイダとして指定することもできます。(この指定を行うと、Oracle Internet Directory インスタンスで定義したユーザーおよびロールのみが、Application Server Control にアクセスできるようになります。)

   作業終了後、次のページに進みます。

6. （オプション）「アイデンティティ管理の構成 : デプロイ済アプリケーション」ページで、Oracle Internet Directory（実際は Oracle Identity Management）を SSO の使用有無に関係なく、OC4J インスタンスの各デプロイ済アプリケーションに対するセキュリティ・プロバイダとして指定できます。

   作業が終了したら、**「構成」**を選択します。これにより、OC4J と OID の関連付けのプロセスが完了し、「アイデンティティ管理」ページが再表示されます。

注意 :

- SSL を使用するには、OC4J および Oracle Internet Directory に対して適切に SSL を構成する必要があります。これについては、第 15 章「OC4J との SSL 通信」および『Oracle Internet Directory 管理者ガイド』でそれぞれ説明されています。
- Oracle Internet Directory は OC4J インスタンス・レベルで関連付けられているため、OracleAS JAAS Provider は、Oracle Internet Directory のホスト、ポート、パスワード、SSL 設定を、アプリケーション・レベルの構成からではなく、指定された OC4J インスタンスの `jazn.xml` ファイルからのみ取得します。
- ディレクトリの各ユーザーは、一意の識別名を持つ必要があります。

Oracle Internet Directory の関連付けの変更

この項では、別の Oracle Internet Directory インスタンスを使用するため、あるいはポートや SSL の構成を変更するために、OC4J と OID の関連付けを変更する手順を説明します。Oracle Internet Directory に、新しい OracleAS JAAS Provider 管理者アカウント（内部で使用）が作成されます。

1. 前述の「Oracle Internet Directory と OC4J の関連付け」と同様に、「アイデンティティ管理」ページにナビゲートします。
2. 「アイデンティティ管理」ページで、**「変更」**を選択します。（以前に OC4J と OID の関連付けを実行していない場合は、**「構成」**が表示されます。）
3. 「アイデンティティ管理の変更」ページで、前項の「アイデンティティ管理の構成」ページと同様に、Oracle Internet Directory ホスト名、Oracle Internet Directory ユーザーの識別名およびパスワード、SSL 接続の使用有無、および接続のポート番号を指定します。
4. **「OK」**を選択します。これにより、OC4J と OID の関連付けの変更プロセスが完了し、「アイデンティティ管理」ページが再表示されます。変更を有効にするために OC4J を再起動するよう指示されます。

Oracle Internet Directory に作成される必須アカウント

デフォルトでは、Oracle Internet Directory には、OC4J 実装および Application Server Control 10.1.3.x 実装で必要となる特定のアカウントは含まれません。このため、OC4J と OID の関連付けプロセスの一環として、次のアカウントが、デフォルトのアイデンティティ管理レルムの下に自動的に作成されます。これは、初めて OC4J インスタンスを Oracle Internet Directory インスタンスと関連付けしたときに行われます。この後で、同じ Oracle Internet Directory インスタンスに対して同一または別の OC4J インスタンスを関連付けした場合でも、次のアカウントは変更できません。実際に、OC4J と OID の関連付けプロセスの時点で、これらのアカウントのいずれかがすでに Oracle Internet Directory に存在することが検出された場合は、アカウント作成の手順がスキップされます。

- `oc4jadmin` ユーザー
- メンバー `oc4jadmin` を含んだ `oc4j-administrators` ロール
- `oc4j-app-administrators` ロール
- メンバー `oc4jadmin` を含んだ `ascontrol_admin`（Application Server Control を含めたすべての SOA コントロールの管理ロール）
- `ascontrol_appadmin`（Application Server Control の必須ロール）
- `ascontrol_monitor`（Application Server Control の必須ロール）

注意：

- Oracle Internet Directory には、この他に `JAZNAdminGroup` ロールとその OracleAS JAAS Provider 管理者メンバーが内部用として用意されています。
- ディレクトリ *`ORACLE_HOME`*`/j2ee/home/jazn/install` のファイル `oidConfigForOc4j.sbs` には、OC4J インスタンスが初めてその Oracle Internet Directory インスタンスに関連付けられたときに Oracle Internet Directory に作成された、デフォルトのユーザーおよびロールの OC4J アカウントとパーミッションが格納されます。これらのアカウントは、通常の OC4J 操作に必要であるため、このファイルは変更または削除しないでください。また、これらのデフォルト・アカウントまたはそのパーミッションは、作成後に変更または削除しないでください。

関連項目：

- OC4J アカウントの追加情報は、4-12 ページの「事前定義アカウント」を参照してください。
- 4-13 ページの「oc4jadmin アカウントのアクティブ化（スタンドアロンの OC4J）」

jazn.xml における Oracle Internet Directory の関連付け

OC4J と OID の関連付けは、OC4J home インスタンス・レベルで有効です。OC4J と Oracle Internet Directory の関連付けを行うと、場所、ユーザー、パスワード、LDAP プロトコルの構成が、OC4J home インスタンスの jazn.xml ファイルに反映されます。サンプルのエントリを示します。

```
<jazn provider="LDAP" location="ldap://myoid.oracle.com:389" default-realm="us" >
   <property
      name="ldap.user"
      value="orclApplicationCommonName=jaznadmin1,cn=JAZNContext,cn=products,
             cn=OracleContext"/>
   <property name="ldap.password"
             value="{903}3o4PTHbgMzVlzbVfKITIO5Bgio6KK9kD"/>
   <property name="ldap.protocol" value="no-ssl"/>
</jazn>
```

デフォルト・レルム us は、Oracle Internet Directory のデフォルトのアイデンティティ管理レルムに対応します。サポートされる ldap.protocol 設定は、SSL 接続の使用有無に応じて、ssl または no-ssl のいずれかになります。デフォルトでは、SSL が使用されます。したがって、Application Server Control を使用する際に SSL を指定すると、実際には ldap.protocol 設定は無効になります。

注意： 実行時には、LDAP ベース・プロバイダは、OracleAS JAAS Provider の管理者として Oracle Internet Directory に接続します。このユーザーは、JAZNAdminGroup のメンバーです。

関連項目：

- 8-23 ページの「LDAP ユーザーおよび SSL のプロパティの構成」

Oracle Internet Directory を関連付ける際の複数の OC4J インスタンスについて

複数の OC4J インスタンスが存在する環境（SOA インストール環境における home インスタンスや SOA インスタンスなど）で Oracle Internet Directory を使用する場合は、前述（8-7 ページの「Oracle Internet Directory と OC4J の関連付け」）の OC4J と OID を関連付ける手順の実行後、関連する <jazn> 要素の構成を、home インスタンスの jazn.xml ファイルから他のインスタンスの jazn.xml ファイルに手動でコピーする必要があります。これには、provider 属性および location 属性の設定と、<property> サブ要素内の関連プロパティの設定が含まれます。

前述の例をもう一度示します。

```
<jazn provider="LDAP" location="ldap://myoid.oracle.com:389" default-realm="us" >
   <property
      name="ldap.user"
      value="orclApplicationCommonName=jaznadmin1,cn=JAZNContext,cn=products,
             cn=OracleContext"/>
   <property name="ldap.password"
             value="{903}3o4PTHbgMzVlzbVfKITIO5Bgio6KK9kD"/>
   <property name="ldap.protocol" value="no-ssl"/>
</jazn>
```

この構成をすべてコピーする必要があります。ただし、コピー先インスタンスの jazn.xml ファイル内で、特別な設定を上書きしないように注意してください。

SSO の構成（オプション）

この手順は、Oracle Identity Management で Oracle Single Sign-On 機能を使用する場合にのみ必要です。この項目の内容は次のとおりです。

1. SSO 登録ツールの実行
2. osso.conf ファイルの OC4J インスタンスへの転送
3. osso1013 スクリプトの実行
4. OracleAS JAAS Provider ユーザー・コンテキストとサーブレット・セッションの同期
5. Oracle HTTP Server および OC4J インスタンスの再起動

重要： この SSO と Java SSO を混同しないでください。Java SSO は別の機能です（第 14 章「OC4J Java シングル・サインオン」を参照）。どちらか一方の SSO 製品を使用できますが、両方を使用することはできません。

関連項目：

- 8-16 ページの「Oracle Single Sign-On 認証に対する OC4J 構成」

SSO 登録ツールの実行

Oracle Single Sign-On を構成するための最初の作業は、使用するアプリケーションをパートナ・アプリケーションとして、インフラストラクチャ内のシングル・サインオン・サーバーに登録することです。この手順はインストール後に実行します。この登録を行うには、インフラストラクチャのインストール環境（SSO サーバー・システム）で ssoreg ユーティリティを実行し、（不明瞭化している）osso.conf ファイルを作成します。

ssoreg ユーティリティは、Linux 版インストールでは *ORACLE_HOME*/sso/bin/ssoreg.sh であり、Windows 版インストールでは *ORACLE_HOME*¥sso¥bin¥ssoreg.bat です。

ここでは、この用途で必要となる ssoreg オプションの構文を説明します。オプションは、表 8-1 で説明しています。

```
-oracle_home_path path
-site_name name
-config_mod_osso TRUE
-mod_osso_url url
-remote_midtier
-config_file path
```

表 8-1 ssoreg の主要なオプション

オプション	説明
oracle_home_path	インフラストラクチャのインストール環境の、*ORACLE_HOME* の場所の絶対パス。
site_name	www.example.com などの Web サイト名。
config_mod_osso	TRUE を設定すると（ここでは TRUE に設定）、登録しようとしているアプリケーションが、事実上、mod_osso（Oracle Single Sign-On 用の Apache モジュール）になります。（実際には、アプリケーションは mod_osso を介して登録されます。）これにより、不明瞭化された osso.conf ファイルが生成されます。
mod_osso_url	アプリケーションが動作するホストの名前およびポートから構成される URL。 http://www.example.com:7777

表 8-1 ssoreg の主要なオプション（続き）

オプション	説明
remote_midtier	コマンド行で指定し、登録するアプリケーションをリモートの中間層に配置することを指定します。OC4J インストールは、Oracle Single Sign-On を含むインフラストラクチャとは異なる層にある（*ORACLE_HOME* が異なる）ため、このオプションを指定する必要があります。
config_file	osso.conf ファイルを置く必要がある場所。通常、次のようになります。 *ORACLE_HOME*/Apache/Apache/conf/osso/osso.conf

次に例を示します（$ORACLE_HOME が正しく設定されている場合）。

```
% $ORACLE_HOME/sso/bin/ssoreg.sh -oracle_home_path $ORACLE_HOME \
  -site_name myhost.mydomain.com -config_mod_osso TRUE \
  -mod_osso_url http://myhost.mydomain.com:7777 -remote_midtier \
  -config_file $ORACLE_HOME/Apache/Apache/conf/osso/osso.conf
```

重要： リリース 10.1.2.0.x のインフラストラクチャにおいて、セキュリティ・プロバイダとして Oracle Single Sign-On を Oracle Identity Management とともに OracleAS JAAS Provider の下で使用する場合は、リリース 10.1.2.0.1 以上にアップグレードする必要があります。これより前のバージョンでは、-remote_midtier オプションがサポートされていないため、このオプションが無視され、コマンドを実行するインフラストラクチャ・ホストの Oracle Application Server Distributed Configuration Management（DCM）に対して、予期しない変更が発生する可能性があります。

関連項目：

- ここで説明されていないオプションを含む ssoreg ユーティリティの追加情報は、『Oracle Application Server Single Sign-On 管理者ガイド』を参照してください。

osso.conf ファイルの OC4J インスタンスへの転送

SSO 登録時に作成された osso.conf ファイル（インストール後はインフラストラクチャのインストール環境に存在）を、FTP などの手段で OC4J 中間層の所定の場所に転送します。

osso1013 スクリプトの実行

SSO 登録プロセスを実行するため、OC4J のインストールから、osso.conf ファイルの場所付きでスクリプト osso1013 を実行します。

```
% osso1013 path/osso.conf
```

このスクリプトは、*ORACLE_HOME*/Apache/Apache/bin ディレクトリにあります。

Windows では、Perl を介して実行する必要があります。

```
% perl osso1013 path/osso.conf
```

OracleAS JAAS Provider ユーザー・コンテキストとサーブレット・セッションの同期

Web アプリケーションが、Oracle Identity Management セキュリティ・プロバイダおよび Oracle Single Sign-On（ログイン、タイムアウトおよびログアウト・サービスとして稼働）とともに使用される場合、OC4J 10.1.3.x 実装は、OracleAS JAAS Provider ユーザー・コンテキストとサーブレット・セッションの同期をサポートします。

この同期を行うと、SSO ログアウトまたはタイムアウトが発生した後でも、ユーザーが保護リソースへのアクセスを試行した場合、再び SSO ログイン・プロンプトが表示されます。（パブリック・リソースにのみアクセスを試行している場合は表示されません。）

デフォルトでは、この同期は無効になっています。有効に（または明示的に無効に）するには、単独の Web アプリケーションの構成に使用される `orion-application.xml` ファイルまたは `orion-web.xml` ファイルにある `<jazn-web-app>` 要素で設定可能なプロパティ `sso.session.synchronize` を介して行います。特定の Web アプリケーションで設定が競合する場合は、`orion-web.xml` の設定が優先されます。

次に、`orion-application.xml` で同期を有効にする例を示します。

```
<orion-application ... >
   ...
   <jazn ... >
      ...
      <jazn-web-app auth-method="SSO" >
         <property name="sso.session.synchronize" value="true" />
      </jazn-web-app>
      ...
   </jazn>
   ...
</orion-application>
```

`sso.session.synchronize` プロパティを `true` に設定した場合は、`ldap.cache.session.enable` プロパティも `true` に設定する必要があります。次に例を示します。

```
<jazn-web-app auth-method="SSO">
   <property name="sso.session.synchronize" value="true" />
   <property name="ldap.cache.session.enable" value="true" />
</jazn-web-app>
```

同期機能は、セッションに応じて前に格納された認証済ユーザーを取得し、LDAP セッション・キャッシュが有効であることを前提としています。`ldap.cache.session.enable` プロパティを `false` に設定すると、セッションを使用して認証済ユーザーを格納することは事実上無効になります。Oracle Internet Directory キャッシング・プロパティの設定方法の詳細は、8-25 ページの「LDAP キャッシング・プロパティの構成」を参照してください。

> **注意：** `sso.session.syncronize` プロパティを `true` に設定し、`ldap.cache.session.enable` プロパティを `false` に設定した場合は、アプリケーションによる Oracle Identity Management/Oracle Single Sign-On を使用した認証が最終的にできなくなるため、OC4J インスタンスを再起動する必要があります。

Oracle HTTP Server および OC4J インスタンスの再起動

登録を有効にするには、Oracle HTTP Server および OC4J を再起動する必要があります。

Oracle Identity Management のセキュリティ・プロバイダとしての構成

この項では、Application Server Control コンソールを使用して、Oracle Identity Management をアプリケーションのセキュリティ・プロバイダとして指定する手順を説明します。この項目の内容は次のとおりです。

- デプロイ時の Oracle Identity Management の指定
- デプロイ後の Oracle Identity Management への変更

注意 :

- この項で説明する手順を実行する前に、Application Server Control に、必要な管理パーミッションを持つユーザー（たとえば `oc4jadmin` など）としてログインしてください。
- アプリケーションが RMI を介して EJB にアクセスできるようにするには、ユーザーまたはロールに対して RMI パーミッション login を付与する必要があります。Oracle Identity Management セキュリティ・プロバイダの使用時にこの設定を行うには、OracleAS JAAS Provider Admintool を使用します。次に例を示します。

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

デプロイ時の Oracle Identity Management の指定

前述した OC4J と OID の関連付けが完了している場合は、Application Server Control を介してアプリケーションをデプロイする際に、Oracle Identity Management（LDAP ベース・プロバイダ）をセキュリティ・プロバイダとして指定できます。

「デプロイ : デプロイ設定」ページで、次の手順を実行します（このページへのナビゲート方法は、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください）。

1. 「セキュリティ・プロバイダの選択」タスクを選択します。
2. 表示される「デプロイ設定 : セキュリティ・プロバイダの選択」ページで、「セキュリティ・プロバイダ」ドロップダウン・リストから「Oracle Identity Management」を選択します。
3. そのドロップダウンで「Oracle Identity Management」を選択すると表示される「Oracle Identity Management セキュリティ・プロバイダの構成」で、次の手順を実行します。
   - Oracle Internet Directory のホストおよびポートが、使用する OC4J インスタンスを Oracle Internet Directory インスタンスと関連付けたときに設定した値と一致するかどうかを確認します。
   - オプションで、Oracle Single Sign-On 認証を有効にします。有効にすると、構成 `auth-method="SSO"` が、アプリケーションの `orion-application.xml` に追加されます。8-16 ページの「Oracle Single Sign-On 認証に対する OC4J 構成」を参照してください。

     重要 : この操作を、別の機能である Java SSO（第 14 章「OC4J Java シングル・サインオン」を参照）を有効にする操作と混同しないでください。Java SSO には独自の Application Server Control 構成ページがあります。どちらか一方の SSO 製品を使用できますが、両方を使用することはできません。
4. 「**OK**」を選択し、セキュリティ・プロバイダの選択を終了します。

5. 必要に応じて JAAS モード設定を確認します。

   a. 「デプロイ : デプロイ設定」ページが再表示されるので、「拡張デプロイ・プランの編集」で**「デプロイ・プランの編集」**を選択します。

   b. 「デプロイ : デプロイ設定 : デプロイ・プランの編集」ページの「OC4J ディスクリプタの編集」タブで、jazn ディスクリプタに対して**「jazn の編集」**を選択します。

   c. 「デプロイ : デプロイ設定 : デプロイ・プランの編集」ページで、jaasMode 属性が適切に設定されていることを確認します（たとえば、アプリケーションが doAsPrivileged を必要としている場合はそのモードに設定されているか）。その後に**「続行」**を選択します。

   d. 「デプロイ : デプロイ設定 : デプロイ・プランの編集」ページが再表示されたら、**「OK」**を選択します。

   このモードをいつどのように使用するかの詳細は、5-6 ページの「JAAS モードの概要」および 5-20 ページの「JAAS モードの構成と使用」を参照してください。

6. 「デプロイ : デプロイ設定」ページが再表示されるので、**「デプロイ」**を選択してデプロイを完了するか、または必要に応じて他のタスクを選択します。タスクのリストは、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください。

注意 :

- Oracle Identity Management をアプリケーションのセキュリティ・プロバイダとして指定すると、orion-application.xml の <jazn> 要素内で、provider="LDAP" が設定されます。
- デプロイ時には、Oracle Internet Directory の場所を指定する必要はありません。これは、OC4J と Oracle Internet Directory の関連付けの時点ですでに指定されており、jazn.xml の <jazn> 要素に反映されているためです。
- デフォルト Oracle Identity Management レルムが、デフォルト・レルムになります。これは Oracle Internet Directory のインストール時に設定されています。

デプロイ後の Oracle Identity Management への変更

アプリケーションで使用するセキュリティ・プロバイダは、前述の項で説明したようにデプロイ時に選択できます。また、デプロイ後に、異なるセキュリティ・プロバイダに変更することもできます。前述した OC4J と OID の関連付けが完了している場合は、次の手順で Oracle Identity Management セキュリティ・プロバイダに変更できます。

1. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、アプリケーションの「セキュリティ・プロバイダ」ページを表示します。
2. 「セキュリティ・プロバイダ」ページで、**「セキュリティ・プロバイダの変更」**を選択します。
3. 「セキュリティ・プロバイダの変更」ページで、「セキュリティ・プロバイダ・タイプ」ドロップダウンから、「Oracle Identity Management セキュリティ・プロバイダ」を選択します。

4. そのドロップダウンで「Oracle Identity Management」を選択すると表示される「セキュリティ・プロバイダ属性 : Oracle Identity Management セキュリティ・プロバイダ」で、次の手順を実行します。

   - Oracle Internet Directory のホストおよびポートが、使用する OC4J インスタンスを Oracle Internet Directory インスタンスと関連付けたときに設定した値と一致するかどうかを確認します。
   - オプションで、Oracle Single Sign-On 認証を有効にします。有効にすると、構成 `auth-method="SSO"` が、アプリケーション用に追加されます。8-16 ページの「Oracle Single Sign-On 認証に対する OC4J 構成」を参照してください。

     重要 : この操作を、別の機能である Java SSO（第 14 章「OC4J Java シングル・サインオン」を参照）を有効にする操作と混同しないでください。Java SSO には独自の Application Server Control 構成ページがあります。どちらか一方の SSO 製品を使用できますが、両方を使用することはできません。

5. 「OK」を選択し、変更を終了します。

「セキュリティ・プロバイダ」ページが再表示され、アプリケーションを再起動して変更を有効にするよう指示されます。

Oracle Identity Management での認証方式に関する設定

この項では、Oracle Identity Management が Web アプリケーションのセキュリティ・プロバイダである場合に使用できる特定の認証方式に関する Oracle 固有の設定について説明します。

Oracle Single Sign-On 認証に対する OC4J 構成

Oracle Single Sign-On を認証に使用するには、`auth-method` 属性を `SSO` に設定します。この属性は、OC4J `orion-application.xml` ファイルの `<jazn-web-app>` 要素（`<jazn>` 要素のサブ要素）にあります。

サンプルのエントリを示します。

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

重要 : Application Server Control を介して任意の時点で任意のアプリケーションに対してファイルベース・プロバイダから Oracle Identity Management に切り替えると、そのアプリケーションの `orion-application.xml` 内にある `<jazn>` 要素が次のように置き換えられます。`<jazn>` 要素の以前の設定はすべて失われるため、設定をやりなおす必要があります。

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

注意：

- web.xml ファイルには、<auth-method> 設定は不要です。web.xml 内の設定は、すべて orion-application.xml 内の SSO 設定でオーバーライドされるためです。
- auth-method="SSO" 設定は、アプリケーションを Application Server Control でデプロイする場合に、Oracle Identity Management でシングル・サインオンを使用するよう指定すると、orion-application.xml ファイルに自動的に書き込まれます。
- <jazn-web-app> 要素は、orion-web.xml ファイルでもサポートされます。競合が発生する場合は、問題の Web アプリケーションに関しては、orion-web.xml が orion-application.xml よりも優先されます。

Digest 認証と Oracle Internet Directory の併用

Oracle Identity Management をセキュリティ・プロバイダとして使用する場合は、Digest 認証を使用する前に、次の手順で準備を行う必要があります。

1. Oracle Directory Manager を使用し、レルムの Oracle Internet Directory パスワード・ポリシーを更新します。
   a. Oracle Directory Manager を oidadmin コマンドで起動します。
   b. Oracle Directory Manager のシステム・オブジェクト・ウィンドウの「Oracle Internet Directory サーバー」で、適切なサーバーを選択します（1 つ以上表示されている場合）。
   c. 適切なサーバーの「パスワード・ポリシー管理」下で、セキュリティ・プロバイダに対して構成したレルムの「パスワード・ポリシー」を選択します。たとえば、レルムが us の場合は、「レルム dc=us,dc=oracle,dc=com のパスワード・ポリシー」を選択します。
   d. Oracle Directory Manager の「レルムのパスワード・ポリシー ...」ウィンドウで、「ユーザー・パスワード可逆暗号化」を有効にします。
2. ユーザーを作成し、Oracle Internet Directory 内でロールを割り当てます。この手順は、必ず手順 1 の完了後に実行します。ユーザーおよびロールは、Oracle Delegated Administration Service を介して管理できます。
3. OracleAS JAAS Provider の構成で、LDAP において SSL が無効になっていないかどうかを確認します。OC4J home インスタンスの jazn.xml ファイルの <jazn> 要素下で、ldap.protocol プロパティに no-ssl が設定されていないことを確認します。（デフォルトでは SSL が有効になります。）

関連項目：

- 4-4 ページの「Oracle Identity Management および Oracle Internet Directory ツールの概要」
- 8-23 ページの「LDAP ユーザーおよび SSL のプロパティの構成」

LDAP ベース・プロバイダのレルム管理

この項では、LDAP ベース・プロバイダである Oracle Identity Management のレルム管理について説明します。内容は次のとおりです。

- Oracle Identity Management の OracleAS JAAS Provider レルムの概要
- Oracle Identity Management のレルム管理

Oracle Identity Management の OracleAS JAAS Provider レルムの概要

OracleAS JAAS Provider は、Oracle Internet Directory のアイデンティティ管理レルム・タイプをサポートしています。

レルム・フレームワークは、OracleAS JAAS Provider への Oracle Internet Directory レルム・インスタンスの登録およびその情報の管理を行うための手段を提供します。

この項の内容は次のとおりです。

- OracleAS JAAS Provider のレルム階層
- JAAS Provider レルムと Oracle Internet Directory レルムの関係
- アクセス制御リストおよび OracleAS JAAS Provider のディレクトリ・エントリ

OracleAS JAAS Provider のレルム階層

図 8-2 で示すように、OracleAS JAAS Provider は、ディレクトリ・エントリを自製品のコンテナ `cn=JAZNContext` に保存します。`cn=JAZNContext` 下には、レルム・エントリを保存する `cn=Realms` コンテナと、グローバル OracleAS JAAS Provider ポリシーを保存する `cn=Policy` コンテナがあります。さらに、`cn=Policy` コンテナには、`cn=Permissions` および `cn=Grantees` の 2 種類のエントリがあります。

図 8-2 グローバル JAZNContext サブツリー

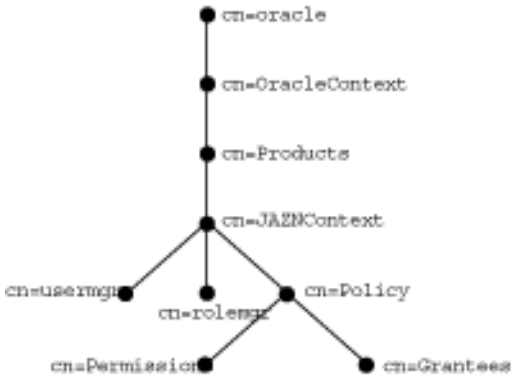
OracleAS JAAS Provider には、独自の `Groups` コンテナと `Users` コンテナがあることに注意してください。`Groups` コンテナには、ロール `JAZNAdminGroup` が格納されます。`Users` コンテナには、このロールに移入される OracleAS JAAS Provider 管理ユーザーが格納されます。ロールとそのメンバー・ユーザーは、どちらも内部でのみ使用されます。管理ユーザー `JAZNAdminUser` は、非推奨であることに注意してください。管理ユーザーは、Oracle Internet Directory に関連付けられた Oracle Application Server の中間層ごとに作成されます。管理者ユーザーには、通常次のような DN が割り当てられます。

```
orclapplicationcommonname=jaznadmin1,cn=jazncontext,cn=products,cn=oraclecontext
```

OracleAS JAAS Provider は、アイデンティティ管理レルム DN を使用して、レルム固有の Oracle コンテキストを検索し、対応する `cn=JAZNContext` サブツリーを作成します。

図 8-3 の場合、cn=oracle がアイデンティティ管理レルムです。OracleAS JAAS Provider では、このアイデンティティ管理レルムに対応する JAZNContext エントリの下に、cn=usermgr エントリ、cn=rolemgr エントリ、ポリシー関連エントリが格納されます。

図 8-3 アイデンティティ管理レルムの JAZNContext サブツリー



JAAS Provider レルムと Oracle Internet Directory レルムの関係

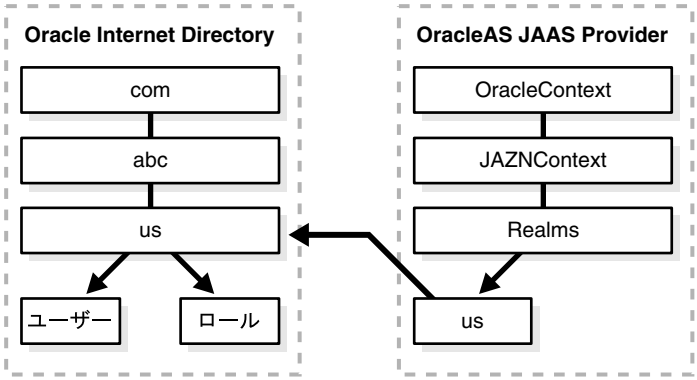
使用するアイデンティティ管理レルムごとに、対応する OracleAS JAAS Provider レルムが作成されます。このメカニズムにより、Oracle Internet Directory 内のアイデンティティ管理レルムが OracleAS JAAS Provider によって認識されます。

OracleAS JAAS Provider では、前述のように、Realms コンテナ・オブジェクトがサイト全体にわたる JAAS コンテキスト下に存在します。各 Oracle Internet Directory レルム・インスタンスに対して、対応するレルム・エントリが Realms コンテナの下に作成され、レルム属性が保存されます。このディレクトリ階層は OracleAS JAAS Provider によって認識され、適切なディレクトリの場所に新しいレルム・エントリを作成し、すべての登録済レルムを実行時に検索する際に使用されます。

Oracle Internet Directory には、システム・ドメインに応じてデフォルトのアイデンティティ管理レルムが用意されています。たとえば米国内では、abc という会社のデフォルト・レルムに対して、dc=us,dc=abc,dc=com などの識別名が割り当てられます。この場合、対応するレルム us が、OracleAS JAAS Provider によってディレクトリ情報ツリーの cn=Realms,cn=JAZNContext,cn=OracleContext の下に作成されます。これを示したのが、次の図 8-4 です。

実行時には、OracleAS JAAS Provider によって各 Oracle Internet Directory レルムとその属性（名前、ユーザー・マネージャ実装クラス、ロール・マネージャ実装クラスおよびこれらのプロパティ）が検索され、これらのレルム・プロパティを持つレルム実装クラスが初期化用にインスタンス化されます。

図 8-4 アイデンティティ管理レルム用の単純化されたディレクトリ情報ツリー



アクセス制御リストおよび OracleAS JAAS Provider のディレクトリ・エントリ

OracleAS JAAS Provider のディレクトリ・エントリは、製品サブツリーのルートにあるアクセス制御リスト（ACL）によって保護されています。これらの ACL は、OracleAS JAAS Provider のディレクトリ・オブジェクトに対する完全な読取り権限と書込み権限を、ロール JAZNAdminGroup と、そのメンバーである OracleAS JAAS Provider の管理スーパーユーザー（どちらも内部使用のみ）に付与します。JAZNAdminGroup に属さない、スーパーユーザー以外のユーザーは、OracleAS JAAS Provider エントリへのアクセスを拒否されます。

アイデンティティ管理の各 JAZNContext サブツリーは、サイト全体にわたる各親のミラー・イメージであるため、エントリ保護にはその親と同じセキュリティ対策が使用されます。

Oracle Identity Management のレルム管理

この項では、LDAP ベース・プロバイダ（Oracle Identity Management）を使用する場合の、Oracle Internet Directory の管理ツールを必要とするレルム管理について説明します。内容は次のとおりです。

- Oracle Internet Directory でのレルムの管理
- デフォルト・レルムの変更
- OC4J での複数レルムと Oracle Single Sign-On の使用方法

Oracle Internet Directory でのレルムの管理

Oracle Internet Directory アイデンティティ管理レルム内のユーザーとロールの管理には、Oracle Delegated Administration Service（DAS）の管理機能を使用します。詳細は『Oracle Identity Management 委任管理ガイド』を参照してください。

また、Oracle Internet Directory レルムの詳細な構成を実行する場合も、DAS を使用します。これには、ユーザー検索ベース、グループ検索ベース、ユーザー作成ベース、グループ作成ベース、ユーザーのニックネーム属性の構成などが含まれます。

OracleAS JAAS Provider 自体は Oracle Internet Directory レルムの管理は行いません。必要に応じて、Oracle Internet Directory に存在する情報の検索のみを行います。

デフォルト・レルムの変更

Oracle Internet Directory にはデフォルト・レルムが用意されていますが、別のレルムをデフォルト・レルムとして使用することもできます。その場合の基本的な手順は次のとおりです。

1. Oracle Internet Directory で、デフォルト・レルムとして使用する新しいアイデンティティ管理レルムを作成します。これは DAS を使用して作成します。詳細は『Oracle Identity Management 委任管理ガイド』を参照してください。これを作成すると、対応する OracleAS JAAS Provider レルムが自動的にプロビジョニングされます。
2. OracleAS JAAS Provider の `default-realm` 属性を、目的のレルムに設定します。詳細は、6-4 ページの「ファイルベースのプロバイダまたは Oracle Identity Management のデフォルト・レルム」を参照してください。

重要： Oracle Internet Directory のレルムを作成する場合は、OracleAS JAAS Provider の Admintool を使用しないでください。このツールで作成したレルムが適しているのは、ファイルベース・プロバイダに対してのみです。Oracle Internet Directory で使用するための情報が不足しています。

OC4J での複数レルムと Oracle Single Sign-On の使用方法

Oracle Internet Directory で追加のアイデンティティ管理レルムを作成する手順は、前項の「デフォルト・レルムの変更」で説明した手順とほぼ同じです。『Oracle Identity Management 委任管理ガイド』で詳述されているように、DAS を使用します。対応する OracleAS JAAS Provider レルムは、自動的にプロビジョニングされます。

重要：

- 複数の Oracle Internet Directory レルムが OracleAS JAAS Provider でサポートされるのは、Oracle Single Sign-On を併用する場合に限られます。
- Oracle Internet Directory のレルムを作成する場合は、OracleAS JAAS Provider の Admintool を使用しないでください。このツールで作成したレルムが適しているのは、ファイルベース・プロバイダに対してのみです。Oracle Internet Directory で使用するための情報が不足しています。
- 場合によっては、レルムを追加する際に、既存のアプリケーションがそのレルムを認識できるようにする必要があります。この手順はアプリケーションによって異なります。各アプリケーションのドキュメントを参照してください。

関連項目： OracleAS JAAS Provider で複数のレルムを使用する場合の重要な追加情報については、次の項目を参照してください。

- 6-6 ページの「非デフォルト・レルムの使用方法」
- 6-6 ページの「複数レルムの使用方法」

複数レルムの環境で Oracle Single Sign-On を使用する場合に、Oracle Single Sign-On サーバーがそれらのレルムを認識できるようにするには、追加の手順が必要です。次に示すのは、複数レルムと Oracle Single Sign-On を併用するための手順をまとめたものです。各手順については、示されている参照先を参照してください。

1. 前述の手順に従ってレルムを作成します。
2. シングル・サインオン・サーバーを、複数レルムに対して構成します。次の手順を実行します。各手順の詳細は、『Oracle Application Server Single Sign-On 管理者ガイド』を参照してください。
   a. シングル・サインオン・サーバーでのホスティングを有効にします。これを行うには、スクリプト `enblhstg.csh` を使用します。
   b. 各レルムのエントリを、Oracle Single Sign-On データベースに作成します。これを行うには、スクリプト `addsub.csh` を使用します。
   c. サンプルのログイン・ページを更新して、複数レルム用のページを作成します。
   d. Oracle Single Sign-On の中間層を停止して再起動します。
3. 複数レルムに対する管理権限を付与します。これについても、『Oracle Application Server Single Sign-On 管理者ガイド』で説明されています。
4. Oracle Single Sign-On を構成します。詳細は、8-11 ページの「SSO の構成（オプション）」を参照してください。
5. OC4J で SSO 認証方式の設定を構成します。詳細は、8-16 ページの「Oracle Single Sign-On 認証に対する OC4J 構成」を参照してください。

複数レルムに対するシングル・サインオンの認証順序は、単一のデフォルト・レルムにおけるシングル・サインオンとほぼ同じです。ユーザーから見て唯一異なるのは、最初のレルム・タイプに属しているユーザーにログイン画面が表示されたときに、ユーザー名とパスワードに加えて、新たな資格証明であるレルムのニックネームを入力する必要がある点のみです。

ユーザーが自分の資格証明を入力すると、そのユーザーのレルム・ニックネームとユーザー名の両方が Oracle Internet Directory 内のエントリにマップされます。具体的には、まずシングル・サインオン・サーバーが、ディレクトリのメタデータを使用してディレクトリ内のレルム・エントリを検索します。このエントリを検出したシングル・サインオン・サーバーは、レルム・メタデータを使用してユーザーを検索します。ユーザーのエントリが検出されると、そのエントリの属性であるパスワードが検証されます。パスワードが検証された時点で、このユーザーは認証されます。

OC4J 構成ファイルにおける LDAP ベース・プロバイダ設定

この項では、LDAP ベース Oracle Internet Directory の様々な構成方法について説明します。この項の内容は次のとおりです。

- LDAP ユーザーおよび SSL のプロパティの構成
- LDAP 接続プロパティの構成
- LDAP キャッシング・プロパティの構成

重要： OC4J インスタンスを Oracle Internet Directory インスタンスに関連付けるまでは、OC4J の home インスタンスの `jazn.xml` ファイルでプロパティ設定を行わないでください。この関連付けを行うと、home インスタンスの `jazn.xml` ファイルにおける `<jazn>` 要素の構成が書き換えられてしまうため、それまでの設定がすべて失われます。

関連項目：

- Oracle Identity Management を使用する場合の、Oracle Delegated Administration Service（DAS）を介したユーザーおよびロールの作成の詳細は、『Oracle Identity Management 委任管理ガイド』を参照してください。

LDAP ユーザーおよび SSL のプロパティの構成

表 8-2 は、LDAP ユーザーと SSL のプロパティをまとめたものです。これらは、OC4J home インスタンスの `jazn.xml` ファイル内の `<jazn>` 要素下の `<property>` サブ要素でサポートされます。これらのパラメータは、この章で説明したように、OC4J と Oracle Internet Directory の関連付けの実行時に、Application Server Control コンソールによってその中の構成が使用され、適切に設定されます。

注意： ここでの説明は、OC4J および Oracle Internet Directory に対して適切な SSL 構成が完了していることを想定しています。これについては、第 15 章「OC4J との SSL 通信」および『Oracle Internet Directory 管理者ガイド』でそれぞれ説明されています。

結果として、構成は次のようになります。

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

変更を有効にするには、OC4J を再起動する必要があります。

表 8-2 LDAP SSL プロパティおよび関連するプロパティ

プロパティ名	プロパティ定義
`ldap.user`	LDAP ユーザー名または識別名。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`
`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` **関連項目：**不明瞭化の詳細は、6-3 ページの「OC4J 構成ファイルのパスワードの不明瞭化」を参照してください。
`ldap.protocol`	SSL を使用するかどうかを決定します。（デフォルトでは SSL を使用します。）サポートされている設定は、`ssl`（通常ポート 636 を使用）または `no-ssl`（通常ポート 389 を使用）です。 **注意：**`ssl` 設定のかわりに、プロトコル `ldaps://` を LDAP URL で使用することもできます。

Oracle Internet Directory は、SSL 通信用に NULL 認証をサポートしています。データが Anonymous Diffie-Hellman 暗号スイートを使用して暗号化されますが、認証に証明書は使用されません。

関連項目：

- 1-4 ページの「SSL 認証」

サンプルの構成を示します。

```
<jazn provider="LDAP" location="ldap://www.example.com:389" default-realm="us">
   <property name="ldap.protocol" value="no-ssl"/>
   ...
</jazn>
```

LDAP 接続プロパティの構成

表 8-3 は、LDAP 接続プロパティをまとめたものです。表 8-4 は、LDAP JNDI 接続プールのプロパティをまとめたものです。これらのプロパティは、インスタンス・レベル `jazn.xml` ファイルの、`<jazn>` 要素の下の `<property>` サブ要素において、次のように設定します。

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

変更を有効にするには、OC4J を再起動する必要があります。

表 8-3 LDAP 接続プロパティ

プロパティ名	プロパティ定義	デフォルト値
`ldap.connect.max.retry`	セキュリティ・プロバイダが放棄する前に LDAP 接続の作成を試行する回数。	5
`ldap.connect.sleep.time`	セキュリティ・プロバイダが失敗した LDAP 接続を再試行する前に待機するミリ秒数。	5000

表 8-4 LDAP JNDI 接続プール・プロパティ

プロパティ名	プロパティ定義	デフォルト値
jndi.ctx_pool.init_size	LDAP JNDI 接続プールの初期サイズ。	5
jndi.ctx_pool.inc_size	LDAP JNDI 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.ctx_pool.timeout	LDAP JNDI 接続プールのタイムアウト値を表すミリ秒数。（たとえば、OracleAS JAAS Provider を含めた中間層と Oracle Internet Directory の間にファイアウォールが存在する場合などに、このプロパティを使用すると便利です。ファイアウォール接続でのタイムアウトを、ディレクトリ接続のタイムアウトに合せることができます。）	0（タイムアウトなし）

注意 : Application Server Control は現在これらの構成をサポートしていないため、ここで説明している構成は手動で実行する必要があります。

LDAP キャッシング・プロパティの構成

Oracle Internet Directory では、パフォーマンスとスケーラビリティの改善を可能に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 が有効になっている 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（java.util.HashMap インスタンス）が保持されます。HashMap 内の期限切れオブジェクトは、必要に応じて定期的に無効にされ、自動的に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、有効期限は次に示すキャッシュ・プロパティで設定できます。

注意 : Oracle Internet Directory のみが、これらのキャッシュをサポートします。ファイルベース・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。

表 8-5 に、LDAP キャッシング・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、インスタンス・レベル jazn.xml ファイルの、<jazn> 要素の下の <property> サブ要素において、次のように設定します。

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

表 8-5 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	true に設定するとポリシー・キャッシュが有効になり、false に設定すると無効になります。	true
ldap.cache.realm.enable	true に設定するとレルム・キャッシュが有効になり、false に設定すると無効になります。	true
ldap.cache.session.enable	true に設定するとセッション・キャッシュが有効になり、false に設定すると無効になります。ユーザー・コンテキストとサーブレット・セッションを同期する場合は、このプロパティを true に設定する必要があります。8-13 ページの「OracleAS JAAS Provider ユーザー・コンテキストとサーブレット・セッションの同期」を参照してください。	true
ldap.cache.initial.capacity	HashMap の初期容量です。このプロパティはパフォーマンスに影響するため、あまり小さな値は設定しないようにします。	20
ldap.cache.load.factor	HashMap のロード・ファクタです。キャッシュ容量が自動的に増加されるようにするキャッシュのロード率を指定します。このプロパティはパフォーマンスに影響するため、あまり大きな値は設定しないようにします。	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 (1 時間)
ldap.cache.purge.timeout	オブジェクトが無効に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 (1 時間)

キャッシングはデフォルトで有効になっています。特定の管理タスクを実行する場合は、特に次のキャッシュを無効にする必要があります。

- レルムを管理する場合はレルム・キャッシュを無効にします。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 を無効にする場合は、セッション・キャッシュを無効にします。

次の例では、3 つのキャッシュをすべて無効にしています。

```
<jazn provider="LDAP" location="ldap://myhost.example.com:636" >
   ...
   <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>
```

または、次のように起動パラメータの設定として無効にすることもできます。

```
-Dldap.cache.session.enable=false
-Dldap.cache.realm.enable=false
-Dldap.cache.policy.enable=false
```

次の例では、キャッシュをすべて有効にしたまま、キャッシュ・サイズを 100 に、タイムアウトを 10,000 ミリ秒に設定します。

```
<jazn provider="LDAP" location="ldap://myhost.example.com:636" >
   <property name="ldap.cache.initial capacity" value="100" />
   <property name="ldap.cache.purget.timeout" value="10000" />
</jazn>
```

注意：

- OracleAS JAAS Provider Admintool は、動作時にキャッシングを自動的に無効にし、終了時にキャッシングを再度有効にします。
- Application Server Control は現在これらの構成をサポートしていないため、ここで説明している構成は手動で実行する必要があります。

LDAP ベース・プロバイダのヒントおよびトラブルシューティング

Oracle Identity Management の LDAP ベース・プロバイダのトラブルシューティング時に注意する必要がある重要事項として、次のものがあります。

- 構成（JAZN-LDAP）のチェック
- ldapsearch を使用した Oracle Internet Directory からのレルム名の取得
- OC4J の再起動により Oracle Internet Directory の変更が有効になることの回避

構成（JAZN-LDAP）のチェック

Oracle Identity Management の使用方法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. Application Server Control を使用して、OC4J が Oracle Internet Directory インスタンスに関連付けられ、セキュリティ・プロバイダが Oracle Identity Management として指定されていることを確認します。
   a. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、「セキュリティ・プロバイダ」ページを表示します。
   b. 「セキュリティ・プロバイダ」ページで、セキュリティ・プロバイダ・タイプとして「Oracle Identity Management セキュリティ・プロバイダ」が表示され、セキュリティ・プロバイダ属性の下に Oracle Internet Directory 用として表示されているホストとポートが正しいことを確認します。
2. Admintool の `-listrealms` コマンドを発行して、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。

注意： OC4J の home インスタンスの `jazn.xml` ファイルでは、`<jazn>` 要素に対して、LDAP ベース・プロバイダを使用することを示す `provider="LDAP"` が設定されています。この要素には、Oracle Internet Directory の場所とポートも反映されています。

ldapsearch を使用した Oracle Internet Directory からのレルム名の取得

OracleAS JAAS Provider の Admintool のかわりに、次のように LDAP の検索コマンドを使用して、Oracle Internet Directory からレルム名を取得できます。

1. ポート、ホスト、ユーザー DN およびパスワードを指定した、次のようなコマンドを開始します。このコマンドでは、`orclSubscriberNicknameAttribute` および `orclSubscriberSearchbase` の値が戻されます。

```
% ldapsearch -p port -h host -D dn_of_user -w password \
        -b "cn=common, cn=products,cn=oraclecontext" -s base "objectclass=*" \
        orclSubscriberNicknameAttribute orclSubscriberSearchbase
```

2. 次に、`orclSubscriberNicknameAttribute` および `orclSubscriberSearchbase` の値を使用して、レルム名を取得します。

```
% ldapsearch -p port -h host -D dn_of_user -w password \
           -b "orclSubscriberSearchbase" \
           -s sub "orclSubscriberNicknameAttribute=*" \
           orclSubscriberNicknameAttribute
```

これにより、Oracle Internet Directory レルムが戻されます。このレルムは Oracle Internet Directory で複数のアイデンティティ管理レルムを使用し、J2EE アプリケーションに対して特定の非デフォルト・レルムを構成する場合に便利です。

関連項目：

- ldapsearch コマンドの詳細は、『Oracle Internet Directory 管理者ガイド』を参照してください。

OC4J の再起動により Oracle Internet Directory の変更が有効になることの回避

Oracle Internet Directory に対して管理操作（権限受領者やグループの追加、パーミッションの付与など）を実行するときは、LDAP キャッシングを無効にする必要があります。キャッシングを有効にしたままにすると、OC4J を停止し、再起度すると、変更が有効になってしまいます。キャッシングを無効にする方法は、8-25 ページの「LDAP キャッシング・プロパティの構成」を参照してください。

Oracle Single Sign-On 管理ページへのアクセス

Oracle Identity Management 10.1.4 実装では、次の URL を入力して Oracle Single Sign-On の管理ページにアクセスできます。

`http://host:port/sso`

これを使用して、Oracle Single Sign-On の設定をチェックできます。

注意： 以前のリリースでは、次の URL を入力して管理ページにアクセスしていました。

`http://host:port/pls/orasso`

関連項目：

- Oracle Single Sign-On 管理ページの追加情報は、『Oracle Application Server Single Sign-On 管理者ガイド』を参照してください。

9

ログイン・モジュール

この章では、OC4J で提供されるログイン・モジュールについてと、カスタム・ログイン・モジュールを実装、インストールおよび構成する方法について説明します。内容は次のとおりです。

- 初期ログイン・モジュールの注意事項
- OC4J で提供されるログイン・モジュール
- カスタム JAAS ログイン・モジュールの概要
- ログイン・モジュールのパッケージ化の選択の概要
- Application Server Control でのカスタム・セキュリティ・プロバイダの構成
- Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与
- 各 OC4J 構成ファイル内のログイン・モジュール構成の概要
- 手順 : カスタム・ログイン・モジュールと OC4J の統合
- カスタム・ログイン・モジュールの例

関連項目：

- 2-14 ページの「JAAS 認証 : ログイン・モジュール」

初期ログイン・モジュールの注意事項

この項の内容は次のとおりです。

Oracle ログイン構成プロバイダの指定

デフォルトで OC4J は、Sun 社のデフォルト JAAS ログイン構成プロバイダの使用を上書きする、OC4J JVM からの OracleAS JAAS Provider ログイン構成プロバイダ（`oracle.security.jazn.spi.LoginConfigProvider`）の使用を指定します。

これは、ファイル *ORACLE_HOME*`/j2ee/home/config/jazn.security.props` の次の構成で実行されます。

```
login.configuration.provider=oracle.security.jazn.spi.LoginConfigProvider
```

Oracle ログイン構成プロバイダでは、ログイン・モジュール構成に `system-jazn-data.xml` ファイルを使用します。

ログイン・モジュールの注意とヒント

OC4J でのログイン・モジュールの使用に関しては、次のことに注意してください。

- 規則上、カスタム・ログイン・モジュール（カスタム・セキュリティ・プロバイダ）を使用するアプリケーションに `provider="XML"` の設定が必要です。`custom.loginmodule.provider` プロパティに関連する情報は、9-23 ページの「<jazn> のログイン・モジュール用設定」を参照してください。
- カスタム・ログイン・モジュールの使用時には、OracleAS JAAS Provider に対して、`system-jazn-data.xml` または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、認証済サブジェクトに基づいて認可チェックを実行するように指示することができます。`role.mapping.dynamic` プロパティに関連する情報は、9-23 ページの「<jazn> のログイン・モジュール用設定」を参照してください。

  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、認証プロセスの `commit` 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を含む）をサブジェクトに追加する必要があります。
- ログイン・モジュール構成（`<jazn-loginconfig>` 要素）は、アプリケーション固有の `jazn-data.xml` ファイル内に置くことはできません。`system-jazn-data.xml` 内に置く必要があります。Application Server Control から、または `orion-application.xml` ファイル内でログイン・モジュールを構成する場合、構成は `system-jazn-data.xml` に自動的に書き出されます。（9-23 ページの「orion-application.xml 内の <jazn-loginconfig> での設定」も参照してください。）
- ファイルベース・プロバイダまたは Oracle Internet Directory 以外のアイデンティティ・リポジトリを使用する場合、管理ユーザー・アカウントおよび管理者ロールを定義し、ロールをユーザーに付与し、必要なパーミッションをロールに付与する必要があります。10-10 ページの「管理ユーザーとロールの作成および RMI パーミッションの付与」を参照してください。
- ユーザー管理は JAAS 仕様の対象外であるため、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。
- カスタム・ログイン・モジュール使用時のサブジェクトベースのポリシー管理の詳細は、5-13 ページの「OracleAS JAAS Provider ポリシー管理」を参照してください。ポリシー構成は `system-jazn-data.xml` 内に存在する必要があります。

- 7-18 ページの「OC4J グループでのファイルベース・プロバイダの使用」では、複数の OC4J インスタンスの間で system-jazn-data.xml の設定を保守管理する方法について説明します。この機能は、setLoginModule、remLoginModule、getLoginModuleControlFlagTypes などの MBean 操作を使用して、ログイン・モジュールにも適用されます。

トラブルシューティングのヒント：

- アプリケーションがカスタム・ログイン・モジュールを使用して構成されるがログイン・モジュールがクラスパス内に見つからない場合、「クラスが見つかりません」という例外がスローされ、「警告 LoginModule クラスが見つかりません。欠落したクラスは Xxxxxx...」という内容のメッセージが出力されます。関連情報は、9-14 ページの「ログイン・モジュールのパッケージ化の選択の概要」を参照してください。
- カスタム・ログイン・モジュールを使用する場合、認可用のロールの比較では大 / 小文字は区別されないので注意してください。ただし、次のプロパティ設定を orion-application.xml の <jazn> 要素に追加した場合は除きます。

```
<property name="role.compare.ignorecase" value="false" />
```

OC4J で提供されるログイン・モジュール

OC4J では、標準の J2EE ログイン・モジュールなど、表 9-1 に示すログイン・モジュールのセットが提供されます。

表 9-1 OC4J で提供されるログイン・モジュール

ログイン・モジュール	説明
oracle.security.jazn.login.module. RealmLoginModule	ファイルベース・プロバイダまたは Oracle Identity Management 用の OC4J ログイン・モジュール（追加情報が続きます。）
oracle.security.jazn.login.module.db. DBTableOraDataSourceLoginModule	データベース内のユーザー・データ用の OC4J ログイン・モジュール（追加情報が続きます。）
oracle.security.jazn.login.module. LDAPLoginModule	外部 LDAP プロバイダ用の OC4J ログイン・モジュール **関連項目：**第 10 章「外部 LDAP セキュリティ・プロバイダ」
oracle.security.jazn.login.module.coreid. CoreIDLoginModule	Oracle Access Manager 用の OC4J ログイン・モジュール **関連項目：**第 11 章「Oracle Access Manager」

注意：

- この表には、OC4J 内部の追加のログイン・モジュールは掲載していません。
- Oracle では現在、OC4J の使用で Sun のログイン・モジュールを検証しません。（パッケージ com.sun.security.auth.module 内のこれらのログイン・モジュールは、現在に JDK に含まれていません。）

この項の残りの部分では、次のログイン・モジュールの追加情報を扱います。

- RealmLoginModule
- DBTableOraDataSourceLoginModule

RealmLoginModule

RealmLoginModule クラスは、ファイルベース・プロバイダまたは Oracle Identity Management で使用するデフォルトのログイン・モジュールで、Application Server Control からセキュリティ・プロバイダを構成するときに構成されます。この構成は、system-jazn-data.xml ファイルの <jazn-loginconfig> の下の <login-module> 要素に反映されます。

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、フォームベースなど）を使用して実行されます。

RealmLoginModule は、表 9-2 に示すオプションをサポートします（<login-module> の下の <option> 要素の <name> および <value> サブ要素に反映されます）。

表 9-2 RealmLoginModule のオプション

オプション	説明	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
supportNullPassword	（Oracle Identity Management 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

注意：

- アプリケーションが Oracle Single Sign-On 認証を使用する場合、RealmLoginModule を有効にする必要はありません。
- RealmLoginModule は、プログラムによるセキュリティではなく、宣言によるセキュリティでのみ使用されます。
- カスタム・ログイン・モジュール、言い換えるとカスタム・セキュリティ・プロバイダとしての RealmLoginModule の使用は、サポートされていません。

関連項目：

- 9-22 ページの「system-jazn-data.xml 内のログイン・モジュール設定」
- Admintool の使用方法の詳細は、9-20 ページの「Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与」を参照してください。

次に、system-jazn-data.xml 内の RealmLoginModule の構成のサンプルを示します。（RealmLoginModule 構成は手動で変更しないでください。この例は単に説明用です。）

```
<jazn-loginconfig>
   <application>
      <name>oracle.security.jazn.tools.Admintool</name>
      <login-modules>
         <login-module>
            <class>oracle.security.jazn.realm.RealmLoginModule</class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>debug</name>
                  <value>false</value>
               </option>
               <option>
                  <name>addAllRoles</name>
                  <value>true</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
   <application>
      <name>oracle.security.jazn.oc4j.JAZNUserManager</name>
      <login-modules>
         <login-module>
            <class>oracle.security.jazn.realm.RealmLoginModule</class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>addAllRoles</name>
                  <value>true</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
</jazn-loginconfig>
```

DBTableOraDataSourceLoginModule

OC4J 10.1.3.1 実装は、データベースにユーザー・アイデンティティ・ストアがある場合に使用できるログイン・モジュールを提供します。

```
oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule
```

これにより、現在では非推奨になった com.evermind.sql.DataSourceUserManager クラスの前の機能が置き換えられます（このクラスは、下位互換性を維持するため引き続きサポートされ、資料を完全なものにするためにこの項の終わりに掲載されています）。また、非推奨の com.evermind.security.User クラスの認証機能の一部も置き換えられます。

データベース・スキーマと、データベースに接続する Oracle データソースを作成すると（『Oracle Containers for J2EE サービス・ガイド』のデータソースの章を参照）、ログイン・モジュールを構成する準備が整います。

DBTableOraDataSourceLoginModule は、データ・ロケーション（表名や列名）やパスワード暗号化のような項目を指定するいくつかのオプションをサポートします。これらのオプションは、Application Server Control または OracleAS JAAS Provider Admintool から設定できます。設定は system-jazn-data.xml ファイルの <jazn-loginconfig> 要素に反映されます。

この項の内容は次のとおりです。

> **注意：** `DBTableOraDataSourceLoginModule` は、OC4J 製品とともに使用でき、自動的にクラスパスに組み込まれます。

DBTableOraDataSourceLoginModule オプション

表 9-3 に `DBTableOraDataSourceLoginModule` がサポートするオプションを要約しています。どのオプションが必須であるかを示し、適用できるデフォルト値や適切な例などを記載しています。出荷時の OC4J にはこのログイン・モジュールの構成はありません。ログイン・モジュールの構成に使用できるツールの詳細は、後続の「Application Server Control 内の DBTableOraDataSourceLoginModule の構成」および「Admintool の DBTableOraDataSourceLoginModule の構成」の項を参照してください。

`DBTableOraDataSourceLoginModule` は、データベース表内の暗号化されたパスワードをサポートします。カスタム・パスワード暗号化アルゴリズムを使用できます。この機能を使用するには、次のインタフェースを実装する必要があります。9-10 ページの「パスワード暗号化の DBLoginModuleEncodingInterface の実装」を参照してください。

```
oracle.security.jazn.login.module.db.DBLoginModuleEncodingInterface
```

Oracle は SHA1 および MD5 アルゴリズムの実装を提供しています。

表 9-3 DBTableOraDataSourceLoginModule オプション

オプション	説明
data_source_name（必須）	データベース用のデータソースの名前。`data-sources.xml` ファイルで構成されます。 デフォルト : なし 例 : jdbc/OracleDS
table（必須）	ユーザー認証情報が含まれるデータベース表の名前（ユーザー名、パスワードなど）。 デフォルト : なし 例 : userinfo
groupMembershipTableName（必須）	ロール情報が含まれるデータベース表の名前。 デフォルト : なし 例 : groupinfo
usernameField（必須）	`table` オプションに指定された表内の、ユーザー名を含む列の名前。 デフォルト : なし 例 : userName

表 9-3 DBTableOraDataSourceLoginModule オプション（続き）

オプション	説明
passwordField（必須）	`table` オプションに指定された表内の、パスワードを含む列の名前。 デフォルト : なし 例 : passWord
pw_encoding_class	パスワード暗号化クラスの名前（パスワードの暗号化を使用する場合）。（この表の直後の説明を参照してください。） デフォルト : oracle.security.jazn.login.module.db.util.DBLoginModuleClearTextEncoder（暗号化なし） 例 : oracle.security.jazn.login.module.db.util.DBLoginModuleSHA1Encoder **関連項目 :** 9-10 ページの「パスワード暗号化の DBLoginModuleEncodingInterface の実装」
pw_key	パスワード暗号化キー（パスワードの暗号化を使用する場合）。このキーは `pw_encoding_class` オプションで指定されたクラスからアクセスされます。 デフォルト : なし 例 : xyz
groupMembershipGroupFieldName（必須）	`groupMembershipTableName` オプションに指定された表内の、ロール名を含む列の名前。
user_pk_column	`table` オプションに指定された表内の、主キーを含む列の名前。 デフォルト : `usernameField` オプションの値。 例 : userName
roles_fk_column	`groupMembershipTableName` オプションに指定された表内の、外部キーを含む列の名前。 デフォルト : `usernameField` オプションの値。 例 : userName
casing	ログイン・ユーザー名とデータベース内の名前を比較するときに大 / 小文字を区別します。大 / 小文字を区別した比較が必要な場合は `sensitive`、ログイン・ユーザー名をすべて大文字に変換する場合は `toupper`、ログイン・ユーザー名をすべて小文字に変換する場合は `tolower` を使用します。（この 3 つの値以外の値が指定された場合は、デフォルト値が使用されます。） デフォルト : sensitive 例 : toupper

注意：

- ユーザー情報とロール情報に同じ表を使用することができます。言い換えると、table オプションと groupMembershipTableName オプションに同じ表を指定できます。ただし、このシナリオでは、それぞれのユーザーが保持できるロールは 1 つのみなので注意してください。通常は、個別の表を使用することをお薦めします。
- DBTableOraDataSourceLoginModule は、null または空のパスワードをサポートしません。
- パスワード暗号化を使用する場合、パスワードは非暗号化文字列ではなく、暗号化された値によって比較されます。ログインするユーザーがパスワードを入力すると、pw_encoding_class オプションに指定したクラスの暗号化メソッドによって暗号化され、データベースに格納されている暗号化されたパスワードと比較されます。データベースに格納されているパスワードの復号化を試みることはありません。

Application Server Control 内の DBTableOraDataSourceLoginModule の構成

9-15 ページの「Application Server Control でのカスタム・セキュリティ・プロバイダの構成」では、デプロイ時のカスタム・ログイン・モジュールの指定と構成、デプロイ後のカスタム・ログイン・モジュールの変更、ログイン・モジュールの追加、ログイン・モジュールの更新について説明します。ここに示す手順で DBTableOraDataSourceLoginModule を構成できます。Application Server Control コンソール・ログイン・モジュール構成ページでは、ログイン・モジュール・クラスを指定し、前項の「DBTableOraDataSourceLoginModule オプション」に記載されたオプションに対応する任意の数のプロパティの名前と値を指定します。

Admintool の DBTableOraDataSourceLoginModule の構成

Application Server Control コンソールを使用する以外の方法として、OracleAS JAAS Provider Admintool から DBTableOraDataSourceLoginModule を構成することもできます。ログイン・モジュールでの Admintool の使用は、9-20 ページの「Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与」を参照してください。

次に例を示します。

```
java -jar jazn.jar -addloginmodule application_name \
     oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule \
     required data_source_name="jdbc/OracleDS" roles_fk_column="username" \
     table="userinfo"  groupMembershipTableName="groupinfo" \
     groupMembershipGroupFieldName="role" usernameField="username" \
     user_pk_column="username" passwordField="password" casing="sensitive"
```

system-jazn-data.xml 内のサンプル DBTableOraDataSourceLoginModule 設定

ログイン・モジュールと同様に、オプション設定およびその他の構成は、system-jazn-data.xml ファイルの <jazn-loginconfig> 要素内に格納されます。次に例を示します。

```
<jazn-loginconfig>
   <application>
      <name>application_name</name>
      <login-modules>
         <login-module>
            <class>
              oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule
            </class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>data_source_name</name>
                  <value>jdbc/OracleDS</value>
```

```
                </option>
                <option>
                   <name>table</name>
                   <value>userinfo</value>
                </option>
                <option>
                   <name>roles_fk_column</name>
                   <value>userName</value>
                </option>
                <option>
                   <name>groupMembershipGroupFieldName</name>
                   <value>role</value>
                </option>
                <option>
                   <name>user_pk_column</name>
                   <value>userName</value>
                </option>
                <option>
                   <name>passwordField</name>
                   <value>passWord</value>
                </option>
                <option>
                   <name>groupMembershipTableName</name>
                   <value>groupinfo</value>
                </option>
                <option>
                   <name>usernameField</name>
                   <value>userName</value>
                </option>
                <option>
                   <name>casing</name>
                   <value>sensitive</value>
                </option>
             </options>
          </login-module>
       </login-modules>
    </application>
    ...
</jazn-loginconfig>
```

関連項目：

- 9-22 ページの「各 OC4J 構成ファイル内のログイン・モジュール構成の概要」

DBTableOraDataSourceLoginModule のプリンシパル

DBTableOraDataSourceLoginModule は次のプリンシパルを使用します。

- oracle.security.jazn.login.module.db.principals.DBUserPrincipal（ユーザー用）
- oracle.security.jazn.login.module.db.principals.DBRolePrincipal（ロール用）

DBTableOraDataSourceLoginModule に渡されるサブジェクトには、これらプリンシパル・タイプのインスタンスが埋め込まれます。

ユーザー表内のユーザー名（table オプションで指定された表の、usernameField オプションで指定された列内の名前）には、対応する DBUserPrincipal インスタンスがサブジェクト内にあります。

ロール表内のロール名（groupMembershipTableName オプションで指定された表の、groupMembershipGroupFieldName オプションで指定された列内の名前）には、対応する DBRolePrincipal インスタンスがサブジェクト内にあります。

ユーザーまたはロールへのパーミッションの付与には、次のように Admintool を使用できます。

```
% java -jar jazn.jar -grantperm \
      oracle.security.jazn.login.module.db.principals.DBUserPrincipal name \
      permissionclass [permission_parameters]

% java -jar jazn.jar -grantperm \
      oracle.security.jazn.login.module.db.principals.DBRolePrincipal name \
      permissionclass [permission_parameters]
```

この構文で、*name* は、DBUserPrincipal インスタンスまたは DBRolePrincipal インスタンスの名前です。*permissionclass* は、付与するパーミッションのパーミッション・クラスの完全修飾名です。*permission_parameters* はパーミッション・クラス（たとえば、RMIPermission の login）の該当するパラメータです。

重要： パーミッション・クラスがクラスパス内にあることを確認してください。

パスワード暗号化の DBLoginModuleEncodingInterface の実装

DBTableOraDataSourceLoginModule でパスワードを暗号化するには、暗号化を実行するための次のインタフェースの実装を使用する必要があります。

```
oracle.security.jazn.login.module.db.DBLoginModuleEncodingInterface
```

実装するクラスには、ログイン・モジュールから呼び出される次のメソッドを実装する必要があります。

- String getKeyDigestString(String text, String key)

このメソッドは、暗号化するテキスト文字列（パスワードなど）、および（適用可能な場合）ログイン・モジュールの pw_key オプションに設定されているようなキーを指定する文字列を取ります。このメソッドは、（指定されている場合はキーを使用して）所定のアルゴリズムでテキスト文字列を暗号化し、必要に応じて、所定のエンコード規格（たとえば、Base64Encoding）で暗号化後のバイナリ・データをフォーマットします。暗号化されたダイジェスト文字列を出力します。

MD5 や SHA1 など、キーを使用しないアルゴリズムでは、キーを渡しても無視されます。pw_key オプションが設定されていない場合、値は null となり、null が渡されます。

Oracle では、oracle.security.jazn.login.module.db.util パッケージに次の実装を提供します。

- DBLoginModuleSHA1Encoder

  このクラスは、指定したパスワード文字列のハッシュ値を SHA1 アルゴリズムで生成し、バイナリ・ハッシュを Base64Encoding でエンコードします。パスワードのバイナリ・ハッシュは、Base64Encoding を使用してデータベースに格納されます。

- DBLoginModuleMD5Encoder

  このクラスは、指定したパスワード文字列のハッシュ値を MD5 アルゴリズムで生成し、バイナリ・ハッシュを Base64Encoding でエンコードします。パスワードのバイナリ・ハッシュは、Base64Encoding を使用してデータベースに格納されます。

- DBLoginModuleClearTextEncoder

  暗号化クラスを指定しない場合、DBLoginModuleClearTextEncoder が使用され、暗号化は行われません。このクラスは、指定したパスワード文字列を単に渡すだけです。

前の機能 : DataSourceUserManager（非推奨）

OC4J 10.1.3.1 実装で DBTableOraDataSourceLoginModule を実装する以前は、com.evermind.sql.DataSourceUserManager クラスによって同等の機能が使用できていました。この機能は下位互換性を維持するために現在でもサポートされていますが、非推奨であり、将来のリリースではサポートされなくなります。かわりに DBTableOraDataSourceLoginModule を使用してください。

しかし、資料を完全なものにするため、DataSourceUserManager 機能についても資料に記載しています。

注意 :

- この場合、データベースをデータソースとして指定し、構成内にはこのデータソース用の JNDI ロケーションを提供する必要があります。構成には、関連するデータベース表とフィールドも指定します。
- OC4J 10.1.3.x 実装では、DataSourceUserManager により、データベースからグループ情報のみが取得されますが、これは以前の実装における動作とは異なっています。したがって、必要に応じてデータベース内のユーザーにグループをマップする必要があります。

DataSourceUserManager（後述）を構成する際、必要に応じて表 9-4 に記載するプロパティの値を指定できます。DataSourceUserManager インスタンスでは、これらのプロパティを使用して、現行の各ユーザーと関連する資格証明がリストされているユーザー定義のデータベース表にアクセスします。

表 9-4 DataSourceUserManager のプロパティ

プロパティ	説明
dataSource	使用するインストール済データソース（データベース）の JNDI ロケーション。
table	ユーザー・データが含まれるデータベース表の名前。
usernameField	データベース表におけるユーザー名用の列名。
passwordField	データベース表におけるパスワード用の列名
certificateIssuerField	証明書発行者の識別子（必要な場合）。
certificateSerialField	証明書発行者のシリアル ID（必要な場合）。
localeField	ロケール（必要な場合）。

表 9-4 DataSourceUserManager のプロパティ（続き）

プロパティ	説明
defaultGroups	ユーザーがメンバーとなるグループのカンマ区切りリスト。
groupMembershipTableName	defaultGroups の使用では不十分な場合に、ユーザーをグループにマップするオプションのデータベース表の名前。
groupMembershipUserNameFieldName	グループ・メンバーシップ・データベース表におけるユーザー名用の列名（必要な場合）。
groupMembershipGroupFieldName	グループ・メンバーシップ・データベース表におけるグループ名用の列名。
staleness	フェッチされたユーザー・データ・セットが有効な期間のミリ秒数。デフォルト設定は -1（永久的）です。
casing	DataSourceUserManager によるデータベース内の既知ユーザーのリストとのユーザー名（グループ名ではない）の照合時に、ユーザー名に対する大 / 小文字の処理方法を制御するフラグ。 デフォルトの sensitive 設定では、大 / 小文字を区別して照合が行われます。toupper および tolower 設定では、名前は照合用にそれぞれすべて大文字またはすべて小文字に変換されます。
debug	デバッグ情報の出力を有効にするフラグ。

DataSourceUserManager を使用するには、orion-application.xml ファイルの <user-manager> 要素で構成します。これは <orion-application> のサブ要素であり、手動で構成する必要があります。Application Server Control 10.1.3.x 実装では、UserManager はサポートされていません。

<user-manager> の class 属性に DataSourceUserManager の完全修飾名を指定します。<property> サブ要素を使用して、設定する各プロパティの名前と値を指定します。

次に例を示します。

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

カスタム JAAS ログイン・モジュールの概要

OC4J でサポートされる JAAS は JAAS 1.0 仕様に完全準拠しているので、ユーザーは、必要に応じて JAAS 準拠 `LoginModule` 実装をプラグインできます。(OC4J には、デフォルトのログイン・モジュール実装として `RealmLoginModule` クラスが組み込まれています。このクラスは、ファイルベース・プロバイダまたは Oracle Identity Management と、J2EE セキュリティ制約を結び付けます。)

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などは、カスタム・ログイン・モジュールの使用をお薦めします。カスタム・ログイン・モジュールの作成時に、次の前提事項を考慮してください。

- 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。
- デバッグ : デバッグ・オプションを、開発時に使用するためログイン・モジュールでサポートする必要があるかどうか。(前述したように、たとえば `RealmLoginModule` では、診断出力を提供する `debug` オプションがサポートされます。また、9-29 ページの「カスタム・ログイン・モジュールの例」にもデバッグ機能が含まれています。)
- 開発およびデプロイ : OC4J または J2SE 1.4 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。ログイン・モジュールとアプリケーションをパッケージ化するか、オプションのパッケージまたはライブラリとして個別に使用できるようにするか。

カスタム・ログイン・モジュールをアプリケーションと関連付ける場合(後述する構成によって)、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、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など)をサブジェクトに追加する必要があります。(サブジェクトベースの認可は、`role.mapping.dynamic` プロパティと関係があります。詳細は9-23 ページの「<jazn> のログイン・モジュール用設定」を参照してください。)

カスタム・ログイン・モジュール・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、アプリケーション・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。

カスタム・ログイン・モジュールは OC4J の `system-jazn-data.xml` ファイルを介して構成しますが、このファイルは、Application Server Control コンソールや OracleAS JAAS Provider Admintool などのツールを使用して自動的に更新できます。ログイン・モジュールを `orion-application.xml` によって構成することもできます。この場合、構成は `system-jazn-data.xml` にコピーされます。

関連項目 :

- 2-14 ページの「JAAS 認証 : ログイン・モジュール」
- カスタム・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`

ログイン・モジュールのパッケージ化の選択の概要

OC4J または J2SE で提供されるデフォルト・ログイン・モジュールを 1 つ以上使用する場合、追加の構成は必要ありません。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。

1 つ以上のカスタム・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。以降の各項では、その実行方法について説明します。

- J2EE アプリケーション内のログイン・モジュールのパッケージ化
- オプション・パッケージとしてのログイン・モジュールの提供
- OC4J 共有ライブラリとしてのログイン・モジュールの提供

この項では、これらのオプションの詳細を説明します。

重要： アプリケーションがカスタム・ログイン・モジュールを使用して構成されるがログイン・モジュールがクラスパス内に見つからない場合、「クラスが見つかりません」という例外がスローされ、「警告 LoginModule クラスが見つかりません。欠落したクラスは Xxxxxx...」という内容のメッセージが出力されます。

J2EE アプリケーション内のログイン・モジュールのパッケージ化

ログイン・モジュールが単一の J2EE アプリケーションでしか使用されない場合は、アプリケーション EAR ファイルにログイン・モジュール JAR ファイルを追加して、ログイン・モジュールをアプリケーションの一部としてそのままパッケージ化できます。

ログイン・モジュールは、次の 2 箇所にあるいずれかの `<jazn-loginconfig>` 設定を介して構成する必要があります。

- `system-jazn-data.xml` ファイル内（9-22 ページの「system-jazn-data.xml 内のログイン・モジュール設定」を参照）
- アプリケーション EAR ファイルに含まれる `orion-application.xml` ファイル内（9-23 ページの「orion-application.xml 内の <jazn-loginconfig> での設定」を参照）

カスタム・ログイン・モジュールは、アプリケーションのデプロイ時、または後で（セキュリティ・プロバイダをカスタムに変更する場合）、Application Server Control コンソールを使用して構成できます。構成を行うと、`system-jazn-data.xml` が自動的に更新されます。

また、OracleAS JAAS Provider Admintool を介してカスタム・ログイン・モジュールの管理を行った場合にも、`system-jazn-data.xml` 設定が自動的に更新されます。

注意： 別のアプリケーションに同じログイン・モジュール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か、オプション・パッケージまたは共有ライブラリとして使用可能にする必要があります。

関連項目：

- 9-15 ページの「Application Server Control でのカスタム・セキュリティ・プロバイダの構成」
- 9-20 ページの「Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与」

オプション・パッケージとしてのログイン・モジュールの提供

ログイン・モジュールをオプション・パッケージ（旧称「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。

オプション・パッケージとしてデプロイするには、次の 2 つの方法があります。

- インストール済オプション・パッケージとしてのログイン・モジュール・クラスの使用。ログイン・モジュール JAR ファイルを `jre/lib/ext` ディレクトリに入れます。アプリケーションは、このディレクトリの JAR ファイル内のクラスを、クラスパスに追加しないでも使用できるようになります。
- ダウンロード・オプション・パッケージとしてのログイン・モジュール・クラスの使用。その他の目的の JAR ファイルのマニフェストにある `Class-Path` ヘッダー・フィールドに、ログイン・モジュール JAR ファイルを指定します。これにより、ログイン・モジュール JAR ファイルに含まれるクラスが、それを参照する他の JAR ファイルのクラスから使用できるようになります。

また、ログイン・モジュールは、`system-jazn-data.xml` 内にも構成する必要があります（9-22 ページの「system-jazn-data.xml 内のログイン・モジュール設定」を参照）。

関連項目：

- 標準「オプション・パッケージ」のデプロイの一般情報は、次の URL を参照してください。

  `http://java.sun.com/j2se/1.4.2/docs/guide/extensions`

OC4J 共有ライブラリとしてのログイン・モジュールの提供

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。このため、ログイン・モジュールは、OC4J 共有ライブラリとしてロードすることで、アプリケーションで使用できるようになります。6-15 ページの「ライブラリを共有するためのタスク」を参照してください。

Application Server Control でのカスタム・セキュリティ・プロバイダの構成

この項では、Application Server Control コンソールを使用したカスタム・セキュリティ・プロバイダ（カスタム・ログイン・モジュール）の次の管理タスクについて説明します。

- デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成
- デプロイ後のカスタム・セキュリティ・プロバイダへの変更
- カスタム・セキュリティ・プロバイダへのログイン・モジュールの追加
- カスタム・セキュリティ・プロバイダのログイン・モジュールの更新
- カスタム・セキュリティ・プロバイダからのログイン・モジュールの削除

注意：

- この項で説明する手順を実行する前に、Application Server Control に、必要な管理パーミッションを持つユーザー（たとえば `oc4jadmin` など）としてログインしてください。
- 指定するログイン・モジュールは、実行時にクラスパス内に存在する必要があります。（共有ライブラリとしてロードする方法もあります。6-15 ページの「ライブラリを共有するためのタスク」を参照してください。）

デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成

カスタム・セキュリティ・プロバイダの使用を計画している場合は、Application Server Control を介してアプリケーションをデプロイする際に、カスタム・ログイン・モジュールを構成できます。

「デプロイ : デプロイ設定」ページで、次の手順を実行します（このページへのナビゲート方法は、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください）。

1. 「セキュリティ・プロバイダの選択」タスクを選択します。
2. 表示される「デプロイ設定 : セキュリティ・プロバイダの選択」ページで、「セキュリティ・プロバイダ」ドロップダウン・リストから「カスタム」を選択します。
3. 「カスタム・セキュリティ・プロバイダの構成」（「カスタム」を選択すると表示される）で、アプリケーションに対して検出されたカスタム・ログイン・モジュールを編集または削除、あるいは新規のカスタム・ログイン・モジュールを追加できます。
   - 新規のカスタム・ログイン・モジュールを追加するには、**「ログイン・モジュールの追加」**を選択します。9-18 ページの「デプロイ時のカスタム・ログイン・モジュールの追加」を参照してください。
   - 既存のカスタム・ログイン・モジュールを編集するには、そのモジュールに対して「編集」タスクを選択します。9-17 ページの「デプロイ時のカスタム・ログイン・モジュール構成の編集」を参照してください。
   - 既存のカスタム・ログイン・モジュールを削除するには、そのモジュールに対して「削除」タスクを選択します。
4. 再表示される「デプロイ設定 : セキュリティ・プロバイダの選択」ページで、**「OK」**を選択して、セキュリティ・プロバイダの選択を終了します。
5. 「デプロイ : デプロイ設定」ページが再表示されるので、**「デプロイ」**を選択してデプロイを完了するか、または必要に応じて他のタスクを選択します。タスクのリストは、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください。

Application Server Control からカスタム・ログイン・モジュールをデプロイまたは構成すると、`orion-application.xml` ファイルに次の設定が自動的に挿入されます。

```
<jazn provider="XML">
   <property name="role.mapping.dynamic" value="true" />
   <property name="custom.loginmodule.provider" value="true" />
</jazn>
```

注意 :

- カスタム・ログイン・モジュールの権限は `system-jazn-data.xml` に格納されていますが、Application Server Control を介しては構成できません。
- 規則上、カスタム・ログイン・モジュールの使用時には、`provider="XML"` の `<jazn>` 設定があります。
- カスタム・ログイン・モジュールの構成設定の反映先は、`system-jazn-data.xml` ファイルの `<jazn-loginconfig>` 要素の下です（9-22 ページの「system-jazn-data.xml 内のログイン・モジュール設定」を参照）。

関連項目 :

- 生成されるログイン・モジュールの XML 構成に関する情報と例については、9-22 ページの「各 OC4J 構成ファイル内のログイン・モジュール構成の概要」および 9-25 ページの「手順 : カスタム・ログイン・モジュールと OC4J の統合」を参照してください。

デプロイ時のカスタム・ログイン・モジュール構成の編集

カスタム・セキュリティ・プロバイダを使用するアプリケーションのデプロイ時にカスタム・ログイン・モジュールを編集するには、「デプロイ設定 : セキュリティ・プロバイダの選択」ページ（このページへのナビゲート方法は、前述の 9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」を参照）の「カスタム・セキュリティ・プロバイダの構成」を選択して、次の手順を実行します。

1. ログイン・モジュール・クラスのリストで、目的のログイン・モジュールに対して「編集」タスクを選択します。
2. 「デプロイ設定 : セキュリティ・プロバイダの選択 : ログイン・モジュールの編集」ページで、次を行います。
   - ドロップダウン・リストから「ログイン・モジュール制御フラグ」の値（Required、Requisite、Optional、Sufficient のいずれか）を指定します。表 9-5 で、この設定について説明します。
   - （オプション）プロパティを作成する場合は、**「行の追加」**を選択します。
   - （オプション）「プロパティ」リスト内の任意のプロパティの名前または値を編集します。
   - （オプション）プロパティを削除する場合は、そのプロパティに対して「削除」タスクを使用します。
   - **「続行」**を選択して「デプロイ設定 : セキュリティ・プロバイダの選択」ページに戻り、9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」に記載のデプロイ手順に戻ります。

表 9-5 ログイン・モジュール制御フラグ

フラグ	意味
`Required`	ログイン・モジュールの成功は表面上必要です。成功するかどうかに関係なく、認証はログイン・モジュール・リストの下位に進みます。
`Requisite`	ログイン・モジュールは表面上成功する必要があります。成功すると、認証は引き続きログイン・モジュール・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（認証はログイン・モジュール・リストの下位に進みません）。
`Sufficient`	ログイン・モジュール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、認証はログイン・モジュール・リストの下位に進みません。失敗すると、認証は引き続きログイン・モジュール・リストの下位に進みます。
`Optional`	ログイン・モジュール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、認証はログイン・モジュール・リストの下位に進みます。

これらの制御フラグは `javax.security.auth.login.Configuration` クラスの標準機能によって使用されます。全体の認証は、Required および Requisite のログイン・モジュールがすべて成功した場合にのみ成功します。ただし、Sufficient のログイン・モジュールが構成されていて成功した場合には、ログイン・モジュール・リスト内の Sufficient のログイン・モジュールの前にある Required および Requisite のログイン・モジュールのみ成功する必要があります。

デプロイ時のカスタム・ログイン・モジュールの追加

カスタム・セキュリティ・プロバイダを使用するアプリケーションのデプロイ時にカスタム・ログイン・モジュールを追加するには、「デプロイ設定 : セキュリティ・プロバイダの選択」ページ（このページへのナビゲート方法は、前述の 9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」を参照）の「カスタム・セキュリティ・プロバイダの構成」を選択して、次の手順を実行します。

1. **「ログイン・モジュールの追加」**を選択します。
2. 「デプロイ設定 : セキュリティ・プロバイダの選択 : ログイン・モジュールの追加」ページで、次を行います。
   - ログイン・モジュールのパッケージとクラス名を指定します。
   - ドロップダウン・リストから「ログイン・モジュール制御フラグ」の値（Required、Requisite、Optional、Sufficient のいずれか）を指定します。この設定の詳細は、前項、表 9-5「ログイン・モジュール制御フラグ」を参照してください。
   - （オプション）プロパティを作成する場合は、**「行の追加」**を選択します。
   - （オプション）「プロパティ」リスト内の任意のプロパティの名前または値を編集します。
   - （オプション）プロパティを削除する場合は、そのプロパティに対して「削除」タスクを使用します。
   - **「続行」**を選択して「デプロイ設定 : セキュリティ・プロバイダの選択」ページに戻り、9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」に記載のデプロイ手順に戻ります。

デプロイ後のカスタム・セキュリティ・プロバイダへの変更

アプリケーションで使用するセキュリティ・プロバイダは、前述のようにデプロイ時に選択できます。また、デプロイ後に、異なるセキュリティ・プロバイダに変更することもできます。カスタム・セキュリティ・プロバイダには次の手順で変更できます。

1. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、アプリケーションの「セキュリティ・プロバイダ」ページを表示します。
2. 「セキュリティ・プロバイダ」ページで、**「セキュリティ・プロバイダの変更」**を選択します。
3. 「セキュリティ・プロバイダの変更」ページで、ドロップダウンから「カスタム・セキュリティ・プロバイダ」を選択します。
4. 「ログイン・モジュール」（ドロップダウンの「カスタム・セキュリティ・プロバイダ」を選択すると表示される）で、使用する最初のログイン・モジュールを次のように指定します。後で「セキュリティ・プロバイダ」に戻って、ログイン・モジュールを追加することもできます（次項、「カスタム・セキュリティ・プロバイダへのログイン・モジュールの追加」を参照）。
   - ログイン・モジュールのパッケージとクラス名を指定します。
   - ドロップダウン・リストから「ログイン・モジュール制御フラグ」の値（Required、Requisite、Optional、Sufficient のいずれか）を指定します。この設定の詳細は、9-17 ページの表 9-5「ログイン・モジュール制御フラグ」を参照してください。
   - （オプション）プロパティを作成する場合は、**「行の追加」**を選択します。
   - （オプション）「プロパティ」リスト内の任意のプロパティの名前または値を編集します。
   - （オプション）プロパティを削除する場合は、そのプロパティに対して「削除」タスクを使用します。
   - **「OK」**を選択し、変更を終了します。

ここで、「セキュリティ・プロバイダ」ページに戻り、変更を反映するためにアプリケーションを再起動するよう求められます。

カスタム・セキュリティ・プロバイダへのログイン・モジュールの追加

デプロイ時またはデプロイ後にカスタム・セキュリティ・プロバイダを設定後、カスタム・ログイン・モジュールを追加するには次のようにします。

1. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、アプリケーションの「セキュリティ・プロバイダ」ページを表示します。
2. 「セキュリティ・プロバイダ」ページの「ログイン・モジュール」で、**「作成」**を選択します。
3. 「ログイン・モジュールの追加」ページで、次を行います。
   - ログイン・モジュールのパッケージとクラス名を指定します。
   - ドロップダウン・リストから「ログイン・モジュール制御フラグ」の値（Required、Requisite、Optional、Sufficient のいずれか）を指定します。この設定の詳細は、9-17 ページの表 9-5「ログイン・モジュール制御フラグ」を参照してください。
   - （オプション）プロパティを作成する場合は、**「行の追加」**を選択します。
   - （オプション）「プロパティ」リスト内の任意のプロパティの名前または値を編集します。
   - （オプション）プロパティを削除する場合は、そのプロパティに対して「削除」タスクを使用します。
   - **「OK」**を選択し、変更を終了します。

「セキュリティ・プロバイダ」ページが再表示されます。このページで設定を確認できます。

カスタム・セキュリティ・プロバイダのログイン・モジュールの更新

デプロイ時またはデプロイ後にカスタム・セキュリティ・プロバイダを設定後、カスタム・ログイン・モジュールを更新するには次のようにします。

1. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、アプリケーションの「セキュリティ・プロバイダ」ページを表示します。
2. 「セキュリティ・プロバイダ」ページでログイン・モジュール・クラスのリストから、更新するログイン・モジュールについて「編集」タスクを選択します。
3. 「ログイン・モジュールの編集」ページで、次を行います。
   - （オプション）ドロップダウン・リストから「ログイン・モジュール制御フラグ」の値（Required、Requisite、Optional、Sufficient のいずれか）を更新します。この設定の詳細は、9-17 ページの表 9-5「ログイン・モジュール制御フラグ」を参照してください。
   - （オプション）プロパティを作成する場合は、**「行の追加」**を選択します。
   - （オプション）「プロパティ」リスト内の任意のプロパティの名前または値を編集します。
   - （オプション）プロパティを削除する場合は、そのプロパティに対して「削除」タスクを使用します。
   - **「適用」**を選択して変更を終了します。

これにより、引き続き「ログイン・モジュールの編集」ページが表示されます。ページ上部のブレッドクラムを使用すると、「セキュリティ・プロバイダ」ページに戻れます。

カスタム・セキュリティ・プロバイダからのログイン・モジュールの削除

デプロイ時またはデプロイ後にカスタム・セキュリティ・プロバイダを設定後、カスタム・ログイン・モジュールを削除するには次のようにします。

1. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、アプリケーションの「セキュリティ・プロバイダ」ページを表示します。
2. 「セキュリティ・プロバイダ」ページでログイン・モジュール・クラスのリストから、削除するログイン・モジュールについて「削除」タスクを選択します。
3. 「確認」ページで**「はい」**を選択します。

「セキュリティ・プロバイダ」ページが再表示されます。このページでログイン・モジュールが削除されたことを確認できます。

Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与

この項では、次の状況で OracleAS JAAS Provider Admintool を使用する方法について説明します。

- ログイン・モジュールの追加と構成を行う Application Server Control コンソールの代替として
- RMI を介して EJB にアクセスする適切なプリンシパルに対して RMI パーミッション login を付与するため

関連項目：

- 付録 C「OracleAS JAAS Provider Admintool リファレンス」

Admintool を介したログイン・モジュールの構成

カスタム・ログイン・モジュールの追加および構成用の推奨ツールは Application Server Control ですが、OracleAS JAAS Provider Admintool を使用することもできます。次の情報を参照してください。

- `-addloginmodule`: 指定したアプリケーション用に新規ログイン・モジュールを構成します。この際、制御フラグの指定が必要です。`javax.security.auth.login.Configuration` および 9-17 ページの表 9-5「ログイン・モジュール制御フラグ」に規定されているように、`Required`、`Requisite`、`Sufficient` または `Optional` のいずれかを指定する必要があります。

  ログイン・モジュールがオプションをサポートする場合は、各オプションとその値を *`optionname=value`* のペアとして指定します。各ログイン・モジュールには、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、必須モジュールとして `MyLoginModule`（`debug` オプションをサポートすると想定）をアプリケーション `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
  ```

- `-remloginmodule`: 指定されたログイン・モジュールを削除します。

  `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
  ```

- `-listloginmodules`: 指定のアプリケーションのログイン・モジュールをすべて表示します。アプリケーション名を指定しない場合は、すべてのアプリケーションのログイン・モジュールを表示します。アプリケーション名の後にログイン・モジュール・クラスを指定すると、そのアプリケーションにある指定クラス関連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
```

これらのコマンドは、Admintool シェルから実行することもできます。

重要： OC4J を再起動して、変更を有効にします。

関連項目：

- C-12 ページの「ログイン・モジュールの追加と削除」
- C-16 ページの「ログイン・モジュールのリスト表示」

Admintool による RMI パーミッションの付与

アプリケーションに EJB が含まれる場合、RMI を介して EJB にアクセスするには、適切なユーザー、ロールまたはプリンシパルに対して RMI パーミッションの login を付与する必要があります。これには、次の例に示すように Admintool を使用できます。

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

% java -jar jazn.jar -grantperm oracle.security.jazn.samples.SamplePrincipal \
       managers com.evermind.server.rmi.RMIPermission login
```

(ユーザーまたはロールにパーミッションを付与するときには常にレルムを指定しますが、プリンシパルにパーミッションを付与するときには指定しません。)

Oracle Application Server 環境に複数の OC4J インスタンスがある場合、該当の OC4J インスタンス用の system-jazn-data.xml ファイルが更新されるように、適用する OC4J インスタンスに対して該当のインスタンス名（および該当のホーム・ディレクトリ）を指定する必要があります。環境に応じて適切に %oracleas.oc4j.instance% を設定します。

```
% java -jar -Doracle.j2ee.home=%ORACLE_HOME%/j2ee/%oracleas.oc4j.instance% \
       jazn.jar -grantperm myrealm -role managers \
       com.evermind.server.rmi.RMIPermission login

% java -jar -Doracle.j2ee.home=%ORACLE_HOME%/j2ee/%oracleas.oc4j.instance% \
       jazn.jar -grantperm oracle.security.jazn.samples.SamplePrincipal managers \
       com.evermind.server.rmi.RMIPermission login
```

重要： OC4J を再起動して、変更を有効にします。

関連項目：

- C-15 ページの「パーミッションの付与と取消し」

各 OC4J 構成ファイル内のログイン・モジュール構成の概要

この項では、カスタム・ログイン・モジュールの構成を含む各ファイルについて説明します。

- system-jazn-data.xml 内のログイン・モジュール設定
- orion-application.xml 内のログイン・モジュール設定
- oc4j-ra.xml でのログイン・モジュール設定（J2EE Connector Architecture）

system-jazn-data.xml 内のログイン・モジュール設定

system-jazn-data.xml ファイルは、ログイン・モジュール構成のリポジトリです。

Application Server Control または OracleAS JAAS Provider Admintool を介してログイン・モジュールの管理を行うと、system-jazn-data.xml の設定が自動的に更新されます。

注意： 複数の OC4J インスタンスがある場合は、ログイン・モジュール構成がインスタンス固有の system-jazn-data.xml ファイルに追加されます。*ORACLE_HOME*/j2ee/home/system-jazn-data.xml には追加されません。

<jazn-loginconfig> 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。

デプロイ時にこの情報が orion-application.xml ファイルにある場合（9-23 ページの「orion-application.xml 内の <jazn-loginconfig> での設定」を参照）、system-jazn-data.xml ファイルはそれに応じて更新されます。

例 9-1 jazn-loginconfig 要素の例

```
<jazn-data>
  ..
  <jazn-loginconfig>
    <application>
      <name>myapp</name>
      <login-modules>
        <login-module>
          <class>mypath.MyLoginModule</class>
          <control-flag>required</control-flag>
          <options>
            <option>
              <name>myoptionname</name>
              <value>myoptionvalue</value>
            </option>
          </options>
        </login-module>
      </login-modules>
    </application>
  </jazn-loginconfig>
  ..
</jazn-data>
```

このフラグメントはログイン・モジュール MyLoginModule とアプリケーション myapp を関連付けます。

注意： system-jazn-data.xml ファイルから RealmLoginModule のログイン構成情報を削除しないでください。

関連項目：

- 4-7 ページの「system-jazn-data.xml ファイル」
- 制御フラグの設定の詳細は、9-17 ページの表 9-5「ログイン・モジュール制御フラグ」を参照してください。
- 9-15 ページの「Application Server Control でのカスタム・セキュリティ・プロバイダの構成」
- 9-20 ページの「Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与」

orion-application.xml 内のログイン・モジュール設定

この項では、OC4J アプリケーション・レベルのディスクリプタ `orion-application.xml` 内のログイン・モジュール向けの設定について説明します。内容は次のとおりです。

- orion-application.xml 内の <jazn-loginconfig> での設定
- <jazn> のログイン・モジュール用設定
- JNDI コンテキストへのアクセス用の <namespace-access> の設定

関連項目：

- `orion-application.xml` の詳細は、『Oracle Containers for J2EE 開発者ガイド』を参照してください。

orion-application.xml 内の <jazn-loginconfig> での設定

`system-jazn-data.xml` 内の `<jazn-loginconfig>` 要素での設定は、前述の 9-22 ページの「system-jazn-data.xml 内のログイン・モジュール設定」を参照してください。この要素はデプロイの前に `orion-application.xml` に追加でき、行った設定は `system-jazn-data.xml` ファイルに自動的に書き込まれます。また、アプリケーションをアンデプロイすると、`<jazn-loginconfig>` 設定は `system-jazn-data.xml` から自動的に削除されます。

<jazn> のログイン・モジュール用設定

次の `<jazn>` のプロパティはログイン・モジュール構成固有です。

- `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、`system-jazn-data.xml` または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、認証済サブジェクトに基づいて認可チェックを実行するように指示したことになります。

  `LoginModule` インスタンスでは、認可プロセスでプリンシパルを考慮に入れるために、認証プロセスのコミット・フェーズで適切なプリンシパル（ユーザーまたはロール）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。サブジェクト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。

- `custom.loginmodule.provider`

  このプロパティを `true` に設定すると、Application Server Control に対して、カスタム・プロバイダをセキュリティ・プロバイダにするよう指示したことになります。この設定がないと、カスタム・セキュリティ・プロバイダでは設定 `provider="XML"` が使用されるため、Application Server Control により、セキュリティ・プロバイダはファイルベース・プロバイダであると誤ってレポートされることになります（ただし、EAR ファイルで指定したカスタム・ログイン・モジュールはまだ機能します）。

これらのプロパティは、orion-application.xml に <jazn-loginconfig> 要素があるとき、または 9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」の説明のように Application Server Control コンソールを使用してカスタム・ログイン・モジュールをデプロイして構成するときは、次の例に示すように orion-application.xml 内で自動的に true に設定されます。

```
<jazn provider="XML" ... >
   <property name="role.mapping.dynamic" value="true" />
   <property name="custom.loginmodule.provider" value="true" />
</jazn>
```

JNDI コンテキストへのアクセス用の <namespace-access> の設定

アプリケーションに EJB が含まれている場合は、アプリケーションのサーバー・サイド JNDI コンテキストでオブジェクトの読取り（検索）および書込み（バインド）を行うためのネームスペース・アクセス権を、必要に応じてリモート・クライアントに付与する必要があります。

orion-application.xml から抜粋した次の例では、読取り操作を行うためのネームスペース・アクセス権を managers および developers ロールに付与する方法を示しています。

```
<orion-application ... >
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="sr_developer">
               <group name="developers"/>
            </security-role-mapping>
            <security-role-mapping name="sr_manager">
               <group name="managers"/>
            </security-role-mapping>
         </namespace-resource>
      </read-access>
   </namespace-access>
   ...
</orion-application>
```

ここでは、示されたロール・マッピングがすでに orion-application.xml 内で設定されていることを前提にしています。

関連項目：

- 18-8 ページの「ネームスペース・アクセスの構成」

マッピング属性の指定

ログイン・モジュール用に、Oracle Internet Directory（OID）プロバイダのマッピング属性は、デフォルトで "DN" に設定されます。

このデフォルト値は、mapping.attribute プロパティの値を構成して変更できます。

1. $Oracle_Home/j2ee/<oc4j_inst>/config/jazn.xml ファイルを探します。
2. mapping.attribute プロパティ構成を <jazn> タグに追加します。次に例を示します。

   ```
   <jazn provider... >
   ...
      <property name = "mapping.attribute" value="cn"/>
   ...
   </jazn>
   ```

3. OC4J インスタンスを再起動します。

jazn.xml ファイルの mapping.attribute 属性に依存するログイン・モジュールは、次のとおりです。

- WS-Security ユーザー名トークン（パスワードなし）（WSSLoginModule）
- SAML（SAMLLoginModule）

  SAMLLoginModule は、最初に SubjectNameIdentifier を参照してマッピングを確認します。空白の場合は、mapping.attribute を使用します。
- X.509 クライアント証明書認証（X509LoginModule）
- サード・パーティのログイン・モジュール（LDAPLoginModule）

これらのログイン・モジュールのマッピング属性は、前述のように jazn.xml ファイルで設定する必要があります。

oc4j-ra.xml でのログイン・モジュール設定（J2EE Connector Architecture）

リソース・アダプタを構成するとき、次の例のように、oc4j-ra.xml ファイルの各 <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。

```
<oc4j-connector-factories ... >
  ...
  <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>
  ...
</oc4j-connector-factories>
```

関連項目：

- リソース・アダプタの構成の詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』を参照してください。

手順 : カスタム・ログイン・モジュールと OC4J の統合

ログイン・モジュールの開発作業は、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。EJB を含むアプリケーションでは、RMI パーミッションを付与し、必要に応じてネームスペース・アクセス権を構成する必要があります。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。

1. ログイン・モジュールの開発
2. ログイン・モジュールの構成とパッケージ化
3. ネームスペース・アクセス権とロール・マッピングの構成（必要な場合）
4. ログイン・モジュールのデプロイ
5. RMI パーミッションの付与（適用可能な場合）
6. JNDI プロパティの設定（適用可能な場合）

ログイン・モジュールの開発

JAAS サービス・プロバイダ・インタフェースに従って JAAS 準拠の LoginModule 実装を開発します。

ログイン・モジュールをアプリケーションと関連付けるとき（前述の <jazn-loginconfig> 構成で実行）、OC4J はサブジェクトに設定されたプリンシパルを使用して、要求された J2EE リソースへのアクセス権をクライアントが持っているかどうかを調べます。したがって、ログイン・モジュールはロール、ユーザーがメンバーとなっているロールをすべて正しく設定する必要があります。

次の例では、ログイン・モジュールは developer ユーザーを認証します。ユーザーが正常に認証された後、ユーザーに関連付けられたロールに対応するプリンシパルが、現在認証されたサブジェクトに追加されます。developer ユーザーが developers ロールのメンバーであることを想定しています。

```
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");
}
```

関連項目：

- 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/

- 9-29 ページの「カスタム・ログイン・モジュールの例」

ログイン・モジュールの構成とパッケージ化

9-14 ページの「ログイン・モジュールのパッケージ化の選択の概要」では、アプリケーションにアクセスできるようにログイン・モジュールをパッケージ化する様々な方法をまとめています。各選択肢を次に示します。

- ログイン・モジュールとアプリケーションのパッケージ化
- オプション・パッケージとしてのログイン・モジュールの提供
- OC4J 共有ライブラリとしてのログイン・モジュールの提供

ログイン・モジュールの構成を指定できる方法にはいくつかの選択肢があります。

- orion-application.xml ファイルの <jazn-loginconfig> 要素を手動で構成します。9-23 ページの「orion-application.xml 内の <jazn-loginconfig> での設定」を参照してください。特に、ログイン・モジュールとご使用のアプリケーションをパッケージ化する場合に役立ちます。それに応じて system-jazn-data.xml 内の <jazn-loginconfig> 要素も更新されます。
- アプリケーションをデプロイするときなどに、Application Server Control コンソールを使用してログイン・モジュール設定を構成します。9-15 ページの「Application Server Control でのカスタム・セキュリティ・プロバイダの構成」を参照してください。system-jazn-data.xml 内の <jazn-loginconfig> 要素が自動的に構成されます。
- OracleAS JAAS Provider Admintool を使用して、ログイン・モジュールを指定します。9-20 ページの「Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与」を参照してください。system-jazn-data.xml 内の <jazn-loginconfig> 要素が自動的に構成されます。

次の項では、キーの構成を中心に説明します。

ログイン・モジュールの使用を有効にする構成

OC4J では、セキュリティ・チェックを実行するカスタム・ログイン・モジュールを使用するために orion-application.xml ファイルに次の構成を必要とします。これは、Application Server Control または orion-application.xml でログイン・モジュール構成を指定すると、自動的に実行されます。

```
<jazn provider="XML" >
   <property name="role.mapping.dynamic" value="true"/>
   <property name="custom.loginmodule.provider" value="true"/>
</jazn>
```

デプロイ後に orion-application.xml ファイルを調べてこれを確認できます。

ログイン・モジュールの構成

次の構成は、9-29 ページの「カスタム・ログイン・モジュールの例」に示すログイン・モジュールのサンプルです。

```
<!-- Configuring a Login Module in an Application EAR file.  -->
<jazn-loginconfig>
  <application>
    <name>cutomjaas</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>
</jazn-loginconfig>
```

デプロイ前にこの構成を手動で orion-application.xml ファイル内に配置することも、Application Server Control からアプリケーションをデプロイする場合に、デプロイ中にプロンプトに応じてパラメータ（クラス名、制御フラグ、オプション名および値）を設定することもできます。いずれの場合も、構成は自動的に system-jazn-data.xml に書き込まれます。

ネームスペース・アクセス権とロール・マッピングの構成（必要な場合）

アプリケーションに EJB が含まれる場合、orion-application.xml 内にネームスペース・アクセス権を構成します。これは、アプリケーションのサーバー・サイド JNDI コンテキストのオブジェクトで、必要に応じて、リモート・クライアントが読取り（検索）操作または書込み（バインド）操作の JNDI バインディングにアクセスできるようにします。次の例に、ネームスペース・アクセス権を managers および developers という名前のロールに付与する方法を示します。

```
<orion-application ... >
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="sr_developer">
               <group name="developers"/>
            </security-role-mapping>
            <security-role-mapping name="sr_manager">
               <group name="managers"/>
            </security-role-mapping>
         </namespace-resource>
```

```
      </read-access>
   </namespace-access>
   ...
</orion-application>
```

これは、次の orion-application.xml の例に示すように、ロール・マッピング（アプリケーション内で定義された論理ロールの、カスタム・ログイン・モジュールでサポートされるロールへのマッピング）が適切に定義されていることを前提としています。

```
<orion-application ... >
   ...
   <!-- Mapping the logical roles to the container roles -->
   <security-role-mapping name="sr_manager">
      <group name="managers" />
   </security-role-mapping>
   <security-role-mapping name="sr_developer">
      <group name="developers" />
   </security-role-mapping>
   ...
</orion-application>
```

ロール・マッピングは Application Server Control から指定できます。6-11 ページの「セキュリティ・ロールのマッピング」を参照してください。これにより、orion-application.xml のエントリが自動的に追加されます。

ログイン・モジュールのデプロイ

9-14 ページの「ログイン・モジュールのパッケージ化の選択の概要」では、ログイン・モジュールをアプリケーションとともに、または個別に（オプション・パッケージまたは共有ライブラリとして）デプロイする方法について説明します。

9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」では、アプリケーションのデプロイ時に Application Server Control によってログイン・モジュールを構成してデプロイする方法について説明します。

RMI パーミッションの付与（適用可能な場合）

アプリケーションに EJB が含まれる場合、RMI を介して EJB にアクセスするには、適切なユーザー、ロールおよびプリンシパルに対して RMI パーミッションの login を付与する必要があります。これには、Admintool を使用できます。9-21 ページの「Admintool による RMI パーミッションの付与」を参照してください。

JNDI プロパティの設定（適用可能な場合）

JNDI を通じてリソースにアクセスするには、jndi.properties ファイルを構成し、環境に応じてプロバイダ URL、プリンシパルおよび資格証明などのプロパティを設定する必要があります。次に例を示します。

```
java.naming.factory.initial=
                    oracle.j2ee.naming.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://localhost:23791/customjaas
#java.naming.provider.url=opmn:ormi://localhost:6003:home/customjaas
java.naming.security.principal=manager
java.naming.security.credentials=welcome
```

（ormi プロトコルはスタンドアロン OC4J 用、opmn:ormi プロトコルは Oracle Application Server 環境用です。適切な方をコメント解除します。）

カスタム・ログイン・モジュールの例

この項では、次の場所での OC4J の使用方法のサンプルとしてログイン・モジュールのコードとプリンシパルのコードを示します。

```
http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html
```

「J2EE Security / JAAS」の下のログイン・モジュールの使用方法を参照してください。

SampleLoginModule コード

この項では、サンプル・ログイン・モジュールのコードを示します。

```
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 LoginModule.
     *
     * subject         the Subject to be authenticated.
     * callbackHandler a CallbackHandler for communicating
     *                 with the end user (prompting for usernames and
     *                 passwords, for example).
     * sharedState     shared LoginModule state.
     * options         options specified in the login
     *                 Configuration for this particular
     *                 LoginModule.
     */
    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.
 *
 * return true if the authentication succeeded, or false if this
 *        LoginModule should be ignored.
 * throws FailedLoginException if the authentication fails.
 * throws LoginException       if this LoginModule
 *                             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");
    }

  if (username.equals("sirish") && password.equals("sirish"))
  {
        _succeeded = true;
        _password = password.toCharArray();
        _name = "sirish";
        _authPrincipals = new SamplePrincipal[1];
        _authPrincipals[0] = new SamplePrincipal("sirish");
   }

    ((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;
}

/*
 * This method is called if the LoginContext's
 * overall authentication succeeded
 * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
 * succeeded).
 *
 * If this LoginModule's own authentication attempt
 * succeeded (checked by retrieving the private state saved by the
 * login method, then this method associates a
 * Principal with the Subject located in the
 * LoginModule.  If this LoginModule's own
 * authentication attempted failed, then this method removes
```

```
     * any state that was originally saved.
     *
     * 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());
        }
    }

    /*
     * This method is called if the LoginContext's
     * overall authentication failed.
     * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
     * did not succeed).
     *
     * If this LoginModule's own authentication attempt
     * succeeded (checked by retrieving the private state saved by the
     * login and commit methods),
     * then this method cleans up any state that was originally saved.
     *
     * 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.
     *
     * This method removes the Principal
     * that was added by the commit method.
     *
     * return true in all cases since this LoginModule
     *        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) {
        }
    }
}
```

SamplePrincipal コード

この項では、サンプル・プリンシパルのコードを示します。

ログイン・モジュールは、現在の Subject インスタンス内の SamplePrincipal インスタンスを設定します。SamplePrincipal インスタンスを作成するために、ログイン・モジュールは直接 SamplePrincipal コンストラクタを呼び出します。そのため、コンストラクタはパブリックとして定義します。

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

import java.security.Principal;

/*
 * This class implements the Principal interface
 * and represents a Sample user.
 *
 * Principals such as this SamplePrincipal
 * may be associated with a particular Subject
 * to augment that Subject with an additional
 * identity.  Authorization decisions can then be based upon
 * the Principals associated with a Subject.
 *
 */
```

```
public class SamplePrincipal implements Principal {

    private String _name = null;


    /*
     * Create a SamplePrincipal with a Sample username.
     *
     */
    public SamplePrincipal(String name) {
       if (name == null)
       throw new NullPointerException("name cannot be null");
        _name = name;
    }

    /*
     * Return a string representation of this SamplePrincipal.
     *
     */
    public String getName() {
        return _name;
    }

    /*
     * Return a hash code for this SamplePrincipal.
     *
     */
    public int hashCode() {
        return _name.hashCode();
    }

    /*
     * Return a string representation of this SamplePrincipal.
     *
     */
    public String toString() {
        return "[SamplePrincipal] : " + _name;
    }

    /*
     * Compares the specified Object with this SamplePrincipal
     * for equality.  Returns true if the given object is also a
     * SamplePrincipal and the two SamplePrincipals
     * have the same username.
     *
     */
    public boolean equals(Object o) {
      if (o == null)
          return false;

        if (this == o)
            return true;

        if (!(o instanceof SamplePrincipal))
            return false;
        SamplePrincipal that = (SamplePrincipal)o;

      if (this.getName().equals(that.getName()))
          return true;
      return false;
    }
}
```

10

外部 LDAP セキュリティ・プロバイダ

この章では、OC4J を構成して Oracle 以外の（サード・パーティまたは外部）LDAP サーバーをユーザー・リポジトリとして使用する方法について説明します。この章の内容は次のとおりです。

- 外部 LDAP プロバイダの構成と管理の概要
- Application Server Control での外部 LDAP プロバイダの構成
- system-jazn-data.xml 内の外部 LDAP プロバイダ設定
- 必要なアカウントの作成と必要なパーミッションの付与
- Sun Java System Directory Server 用の構成のサンプル
- 外部 LDAP プロバイダでの SSL の使用

OC4J 10.1.3.x 実装では、次の外部 LDAP プロバイダがサポートされています。

- Active Directory（Windows Server 2003 対応）
- Sun Java System Directory Server（バージョン 5.2）

注意：

- 外部 LDAP プロバイダのサポートには JDK 1.4 以降が必要です。
- OC4J 10.1.3.x 実装から、外部 LDAP プロバイダは、スタンドアロンの OC4J でも、Oracle Application Server 環境の場合と同様にサポートされるようになりました。
- 外部 LDAP プロバイダを使用する際には、セキュリティ・レルムの概念はサポートされません。
- 外部 LDAP プロバイダ使用時のサブジェクトベースのポリシー管理の詳細は、5-13 ページの「OracleAS JAAS Provider ポリシー管理」を参照してください。ポリシー構成は `system-jazn-data.xml` 内に存在する必要があります。

関連項目：

- 外部 LDAP プロバイダとともに使用できるユーザーとロールの API の詳細は、第 12 章「ユーザーおよびロール API フレームワーク」を参照してください。

外部 LDAP プロバイダの構成と管理の概要

Application Server Control コンソールを使用してアプリケーションをデプロイするときは、外部（サード・パーティ）LDAP プロバイダを指定できます（6-10 ページの「Application Server Control を介したセキュリティ・プロバイダの指定」を参照）。

注意： 前提条件として、Sun Java System Directory Server（旧称 iPlanet）または Active Directory のインストールと構成を完了しているものとします。

外部 LDAP プロバイダを指定すると、自動的に `orion-application.xml` ファイルに次の設定が生成されます。

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

注意：

- 規則上、`<jazn>` 設定の `provider="XML"` は外部 LDAP プロバイダで使用されます。
- 外部 LDAP プロバイダを使用する場合、認可用のロールの比較では大 / 小文字は区別されないので注意してください。ただし、次のプロパティ設定を `orion-application.xml` の `<jazn>` 要素に追加した場合は除きます。

  ```
  <property name="role.compare.ignorecase" value="false" />
  ```

トラブルシューティングのヒント： 外部 LDAP プロバイダの使用でトラブルがあった場合は、次の問題が発生していないか検討してください。

- LDAP ユーザーの識別名（DN）を使用して LDAP サーバーに接続しているかどうか確認してください。このユーザーは、ユーザーとグループの検索権限を持つ管理者である必要があります。
- ログインで正しいユーザー名とパスワードを入力しても、資格証明が無効なために認証障害が再発する場合は、LDAP ホストとポートが正しく構成されているか確認してください。この確認には、`ldapbind` コマンドを使用して構成済 LDAP ホストとポートに対してバインドすることをお薦めします。

OC4J では、ログイン・モジュール `LDAPLoginModule` を備えており、外部 LDAP プロバイダでの認証と認可に使用されます。（または、カスタム・リポジトリで使用するためのカスタム・ログイン・モジュールを提供できます。）外部 LDAP プロバイダには、次の構成可能なオプションがあります。

- 外部 LDAP プロバイダの URL
- 接続用 LDAP プリンシパル DN（ユーザーは、LDAP ディレクトリ内のユーザーのロール情報を問い合せる権限を備えている必要があります。）
- LDAP プリンシパル DN の資格証明
- ユーザーを一意に識別する LDAP 属性
- ユーザーのオブジェクト・クラス、検索ベース、検索範囲
- ロールのオブジェクト・クラス、検索ベース、検索範囲
- 接続プーリングの有効化または無効化
- ログイン・モジュールのキャッシングの有効化または無効化

LDAPLoginModule を構成するオプション設定は、system-jazn-data.xml 内の <jazn-loginconfig> の下の <login-module> 要素内に反映されます。

注意： Sun Java System Directory Server 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、*ORACLE_HOME*/j2ee/home/jazn/config ディレクトリのファイル ldap_login_module.template に提供されています。

Application Server Control での外部 LDAP プロバイダの構成

この項では、Application Server Control コンソールを使用した外部 LDAP プロバイダの管理について説明します。次の項目があります。

- デプロイ時の外部 LDAP プロバイダの指定および構成
- デプロイ後の外部 LDAP プロバイダへの変更

注意： この項で説明する手順を実行する前に、Application Server Control に、必要な管理パーミッションを持つユーザー（たとえば oc4jadmin など）としてログインしてください。

デプロイ時の外部 LDAP プロバイダの指定および構成

外部 LDAP プロバイダの使用と Application Server Control を介したアプリケーションのデプロイを計画している場合は、外部 LDAP プロバイダをセキュリティ・プロバイダとして指定する際にその構成ができます。

「デプロイ : デプロイ設定」ページで、次の手順を実行します（このページへのナビゲート方法は、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください）。

1. 「セキュリティ・プロバイダの選択」タスクを選択します。
2. 表示される「デプロイ設定 : セキュリティ・プロバイダの選択」ページで、「セキュリティ・プロバイダ」ドロップダウン・リストから「サード・パーティの LDAP サーバー」を選択します。
3. 「サード・パーティの LDAP サーバー用の Oracle セキュリティ・プロバイダの構成」（「サード・パーティの LDAP サーバー」を選択すると表示される）で、次に記載されているオプションの設定を指定します。
   - 10-4 ページの表 10-1「Application Server Control 外部 LDAP プロバイダのオプション」
   - 10-4 ページの表 10-2「Application Server Control の外部 LDAP 接続プールのオプション」（接続プーリングを有効にしている場合）
   - 10-5 ページの表 10-3「Application Server Control 外部 LDAP ユーザーのオプション」
   - 10-5 ページの表 10-4「Application Server Control 外部 LDAP のロールおよびメンバーのオプション」

   あるいは、「LDAP ディレクトリ・ベンダー」設定で指定されたベンダーに対して**「値をベンダーのデフォルトに設定」**を選択します。これ以外の場合は、依然として「LDAP ロケーション」、「ユーザー DN」、「ユーザー検索ベース」および「グループ検索ベース」を指定する必要があります。
4. オプションで、デプロイの前に**「LDAP 認可のテスト」**を選択できます。これは、LDAP セッションが正常に作成できるかどうかをテストして、LDAP ホスト、ポート、管理ユーザーおよびパスワードなどの主要な設定を確認するものです。
5. **「OK」**を選択し、セキュリティ・プロバイダの選択を終了します。

6. 「デプロイ：デプロイ設定」ページが再表示されるので、**「デプロイ」**を選択してデプロイを完了するか、または必要に応じて他のタスクを選択します。タスクのリストは、6-9 ページの「Application Server Control を介したアプリケーションのデプロイ」を参照してください。

表 10-1 Application Server Control 外部 LDAP プロバイダのオプション

オプション	必須 または設定 / デフォルト	表 10-5 での対応オプション（詳細用参照先）
LDAP ロケーション	Required	oracle.security.jaas.ldap.provider.url
LDAP ディレクトリ・ベンダー	「Active Directory」、「Sun Directory Server」または「その他」（ドロップダウン・メニューから選択）	oracle.security.jaas.ldap.provider.type
ユーザー DN	Required	oracle.security.jaas.ldap.provider.principal
ユーザー・パスワード	デフォルトなし	oracle.security.jaas.ldap.provider.credential
キャッシング有効化（チェック・ボックス）	デフォルト：`true`	oracle.security.jaas.ldap.lm.cache_enabled
接続プーリング有効化（チェック・ボックス）	デフォルト：`true`	oracle.security.jaas.ldap.provider.connect.pool

表 10-2 Application Server Control の外部 LDAP 接続プールのオプション

オプション	デフォルト	説明
接続プールの初期サイズ	2	初めに各接続 ID 用にプールに作成される接続の数
接続プールの最大サイズ	25	各接続 ID 用にプールに同時に保持できる接続の最大数
接続プールの最適サイズ	10	プールにおける各接続 ID 用接続数の希望値
アイドル接続タイムアウト（ミリ秒）	300000（5 分）	アイドル接続が削除されるまでプールに保持される時間

注意： 前述の接続プール・プロパティは次のものに相当します。

```
com.sun.jndi.ldap.connect.pool.initsize
com.sun.jndi.ldap.connect.pool.maxsize
com.sun.jndi.ldap.connect.pool.prefsize
com.sun.jndi.ldap.connect.pool.timeout
```

詳細は、次のサイトを参照してください。

```
http://java.sun.com/products/jndi/tutorial/ldap/connect/
config.html
```

表 10-3 Application Server Control 外部 LDAP ユーザーのオプション

オプション	必須 または設定 / デフォルト	表 10-6 での対応オプション（詳細用参照先）
ユーザー検索ベース	Required	oracle.security.jaas.ldap.user.searchbase
ユーザー検索範囲	「サブツリー」（デフォルト）または「1 レベル」（ドロップダウン・メニューから選択） **注意**：ドロップダウン・メニューのデフォルトは「サブツリー」ですが、ベンダーのデフォルトは「1 レベル」です。	oracle.security.jaas.ldap.user.searchscope
LDAP ユーザー名の属性	Required	oracle.security.jaas.ldap.user.name.attribute
LDAP ユーザー・オブジェクト・クラス	Required	oracle.security.jaas.ldap.user.object.class

表 10-4 Application Server Control 外部 LDAP のロールおよびメンバーのオプション

オプション	必須 または設定 / デフォルト	表 10-7 での対応オプション（詳細用参照先）
グループ検索ベース	Required	oracle.security.jaas.ldap.role.searchbase
グループ検索範囲	「サブツリー」（デフォルト）または「1 レベル」（ドロップダウン・メニューから選択） **注意：**ドロップダウン・メニューのデフォルトは「サブツリー」ですが、ベンダーのデフォルトは「1 レベル」です。	oracle.security.jaas.ldap.role.searchscope
LDAP グループ名の属性	Required	oracle.security.jaas.ldap.role.name.attribute
LDAP グループ・オブジェクト・クラス	Required	oracle.security.jaas.ldap.role.object.class
LDAP グループ・メンバー属性	Required	oracle.security.jaas.ldap.member.attribute
グループ・メンバーシップ検索範囲	「ダイレクト」（デフォルト）または「ネステッド」（ドロップダウン・メニューから選択）	oracle.security.jaas.ldap.membership.searchscope

デプロイ後の外部 LDAP プロバイダへの変更

アプリケーションで使用するセキュリティ・プロバイダは、前述の項で説明したようにデプロイ時に選択できます。また、デプロイ後に、異なるセキュリティ・プロバイダに変更することもできます。特に、外部 LDAP プロバイダに変更するには、次の手順を実行します。

1. 6-14 ページの「アプリケーションの「セキュリティ・プロバイダ」ページへのナビゲート」の説明に従って、アプリケーションの「セキュリティ・プロバイダ」ページを表示します。
2. 「セキュリティ・プロバイダ」ページで、**「セキュリティ・プロバイダの変更」**を選択します。
3. 「セキュリティ・プロバイダの変更」ページで、「セキュリティ・プロバイダ・タイプ」ドロップダウンから、「サード・パーティの LDAP サーバー用の Oracle セキュリティ・プロバイダ」を選択します。
4. 「セキュリティ・プロバイダ属性 : サード・パーティの LDAP サーバー用の Oracle セキュリティ・プロバイダ」（ドロップダウンで「サード・パーティの LDAP サーバー」を選択すると表示される）で、前項の次の表に記載されているオプションの設定を指定します。
   - 表 10-1「Application Server Control 外部 LDAP プロバイダのオプション」
   - 表 10-2「Application Server Control の外部 LDAP 接続プールのオプション」（接続プーリングを有効にしている場合）
   - 表 10-3「Application Server Control 外部 LDAP ユーザーのオプション」
   - 表 10-4「Application Server Control 外部 LDAP のロールおよびメンバーのオプション」

   あるいは、「LDAP ディレクトリ・ベンダー」設定で指定されたベンダーに対して**「値をベンダーのデフォルトに設定」**を選択します。これ以外の場合は、依然として「LDAP ロケーション」、「ユーザー DN」、「ユーザー検索ベース」および「グループ検索ベース」を指定する必要があります。
5. オプションで、デプロイの前に**「LDAP 認可のテスト」**を選択できます。これは、LDAP セッションが正常に作成できるかどうかをテストして、LDAP ホスト、ポート、管理ユーザーおよびパスワードなどの主要な設定を確認するものです。
6. **「OK」**を選択し、変更を終了します。

ここで、「セキュリティ・プロバイダ」ページに戻り、変更を反映するためにアプリケーションを再起動するよう求められます。

重要 : ロール・マッピングが適切に設定されていて、デプロイ・ロール（外部 LDAP プロバイダで定義されているロール）を J2EE 論理ロールに正しくマップすることも確認します。追加情報は、6-11 ページの「セキュリティ・ロールのマッピング」を参照してください。

---

system-jazn-data.xml 内の外部 LDAP プロバイダ設定

外部 LDAP プロバイダの構成は、system-jazn-data.xml ファイルの <login-module> 要素に反映されます。この要素には、OracleAS JAAS Provider で外部 LDAP プロバイダに使用されるログイン・モジュール LDAPLoginModule が構成されます。すべての <login-module> 要素は、<jazn-loginconfig> の下の <login-modules> 要素のサブ要素です。

<login-module> 要素の各オプション設定は、<option> 要素（<options> のサブ要素）で表され、外部 LDAP プロバイダの構成設定と対応しています。<option> 要素の <name> サブ要素ではオプションの名前を指定し、<value> サブ要素では対応する値を指定します。

これらのオプションの設定は、10-3 ページの「デプロイ時の外部 LDAP プロバイダの指定および構成」に記載されているように、Application Server Control を介して指定できます。また、その項では、この項にリストされているオプションと Application Server Control コンソールに表示されるオプションとの間の対応関係についても示しています。

表 10-5、表 10-6 および表 10-7 に、サポート対象のオプションを示します。表には、Application Server Control を介して外部 LDAP プロバイダを構成し、**「値をベンダーのデフォルトに設定」**を選択したときに使用されるデフォルト値が示されています（存在する場合）。特に記述されている場合を除き、これらのオプションは必須です。直接指定するか、ベンダー・デフォルトを使用する必要があります。

注意： <jazn-loginconfig> 要素は、orion-application.xml ファイルにも使用できます。使用した場合は、そのファイルから system-jazn-data.xml ファイルにコピーされます。

関連項目：

- オプション設定の例は、10-13 ページの「system-jazn-data.xml 内の Sun Java System Directory Server 用設定」を参照してください。

表 10-5 外部 LDAP 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	ldap://*host*:*port* 形式の LDAP プロバイダの URL。次に例を示します。 ldap://myhost.example.com:389
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、ユーザーおよびロールを検索するための権限と、ユーザー・パスワードに対して ldapcompare を起動するための権限（これを起動する機能がターゲット・ディレクトリでサポートされている場合）を持つ管理者であることが必要です。
oracle.security.jaas.ldap.provider.credential	次のものに定義されている 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.principal
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、sun directory server、active directory および other です。sun directory server または active directory を指定すると、ログイン・モジュールでは一部の LDAP プロパティを推測し、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）接続プーリングを有効にするかどうかを指定するブール値。true 設定（デフォルト）は接続プーリングを有効にし、false は無効にします。

表 10-5 外部 LDAP プロバイダのオプション（続き）

オプション名	意味
oracle.security.jaas.ldap.lm.cache_enabled	（オプション）ログイン・モジュールのキャッシングを有効にするかどうかを指定するブール値。true（デフォルト）はキャッシングを有効にし、false は無効にします。
oracle.security.jaas.ldap.context.referral	（オプション）指定できる値は follow、throw または ignore のいずれか。ログイン・モジュールによる参照の処理方法を示します。指定されていない場合は、プロバイダのデフォルト値が使用されます。このプロパティはデフォルトで指定されていません。
oracle.security.jaas.ldap.object.category	（オプション）Active Directory でのみ使用され、グループ検索フィルタで使用する objectCategory を指定します。指定されていない場合、objectCategory はフィルタに含まれません。このプロパティはデフォルトで指定されていません。

表 10-6 外部 LDAP 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。デフォルトは、Sun Java System Directory Server の場合は uid で、Active Directory の場合は sAMAccountName です。
oracle.security.jaas.ldap.user.object.class	ユーザーを表すために使用される LDAP スキーマ・オブジェクト・クラスの、1 つ以上の空白で区切られたリスト。デフォルトは、Sun Java System Directory Server または Active Directory のいずれの場合も、inetOrgPerson です。
oracle.security.jaas.ldap.user.searchbase	ユーザーを含む LDAP ディレクトリ内の識別名（DN）の空白で区切られたリスト。次に DN のサンプルを示します。 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 または onelevel（デフォルト）です。

表 10-7 外部 LDAP のロールおよびメンバーのオプション

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。デフォルトは、Sun Java System Directory Server または Active Directory のいずれの場合も、cn です。
oracle.security.jaas.ldap.role.object.class	ロールを表すために使用される LDAP スキーマ・オブジェクト・クラスの、1 つ以上の空白で区切られたリスト。デフォルトは、Sun Java System Directory Server の場合は groupOfUniqueNames で、Active Directory の場合は group です。
oracle.security.jaas.ldap.role.searchbase	ロールを含む 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

表 10-7 外部 LDAP のロールおよびメンバーのオプション（続き）

オプション名	意味
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` または `onelevel`（デフォルト）です。
oracle.security.jaas.ldap.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（デフォルト）または `nested` です。`direct` 設定は、当該のロールまたはユーザーに直接割り当てられたロールのみがランタイムにより取得されることを意味します。ロール内にネストされたロールは取得されません。
oracle.security.jaas.ldap.member.attribute	ロールのメンバーの識別名（DN）を指定する静的 LDAP ロール・オブジェクトの属性。デフォルトは、Sun Java System Directory Server の場合は `uniqueMember` で、Active Directory の場合は `member` です。

次に例を示します。

```
<jazn-data ... >
   ...
   <jazn-loginconfig>
      <application>
         <name>callerInfo</name>
         <login-modules>
            <login-module>
               <class>oracle.security.jazn.login.module.LDAPLoginModule</class>
               <control-flag>required</control-flag>
               <options>

                  <option>
                     <name>oracle.security.jaas.ldap.provider.url</name>
                     <value>ldap://myhost.example.com:389</value>
                  </option>
                  ...more options...
               </options>
            </login-module>
            ...
         </login-modules>
      </application>
      ...
   </jazn-loginconfig>
   ...
</jazn-data>
```

詳細な例は、10-13 ページの「system-jazn-data.xml 内の Sun Java System Directory Server 用設定」を参照してください。

必要なアカウントの作成と必要なパーミッションの付与

この項では、外部 LDAP プロバイダを使用して必要なアカウントを作成し、必要なパーミッションを付与するときの手順について説明します。この項の内容は次のとおりです。

- 管理ユーザーとロールの作成および RMI パーミッションの付与
- LDAP プリンシパルへの RMI パーミッションの付与
- 外部 LDAP プリンシパルへの追加のパーミッションの付与
- 外部 LDAP プロバイダでの JAAS モードの使用

管理ユーザーとロールの作成および RMI パーミッションの付与

外部 LDAP プロバイダを使用する場合、管理ユーザー・アカウントと管理ロールを定義し、そのロールをユーザーに付与し、必要なパーミッションをそのロールに付与する必要があります。(この手順は、ファイルベースのセキュリティ・プロバイダと Oracle Internet Directory で自動的に処理されます。)

1. セキュリティ・プロバイダの適切なツールを使用して、外部 LDAP ディレクトリに管理ユーザー・アカウントを作成します。(名前 `oc4jadmin` は規則上、管理アカウントに使用されますが、任意の名前を使用することができます。)
2. 適切なツールで、外部 LDAP ディレクトリに管理者ロール `oc4j-administrators` および `ascontrol_admin` を作成します。これらのロールは、LDAP プロバイダ用に構成されたグループ検索ベースの下に存在する必要があります。(グループ検索ベースの詳細は、10-5 ページの表 10-4 および 10-8 ページの表 10-7 を参照してください。)
3. これらのロールを適切なツールで管理ユーザーに付与します。
4. 追加の管理者ロールの `oc4j-app-administrators`、`ascontrol_appadmin` および `ascontrol_monitor` を作成します。(これらは管理ユーザーに付与する必要はありません。)
5. 管理者が EJB にアクセスする必要がある場合、これらのロールに RMI パーミッションの login を付与します。この場合、次の例に示すように OracleAS JAAS Provider Admintool を使用できます。

   ```
   % java -jar jazn.jar -grantperm myrealm -role oc4j-administrators \
        com.evermind.server.rmi.RMIPermission login
   ```

   ロールは外部 LDAP プロバイダで定義されていますが、これらのパーミッション付与は `system-jazn-data.xml` ファイルに格納されていることに注意してください。

   関連項目:

   - 4-12 ページの「事前定義アカウント」

LDAP プリンシパルへの RMI パーミッションの付与

EJB アプリケーション用の外部 LDAP プロバイダを使用するときに、EJB にアクセスする適切な LDAP プリンシパルに RMI パーミッションの login を付与する必要があります。

次の例では、OracleAS JAAS Provider Admintool を使用して、LDAP プリンシパル hobbes に対してパーミッションを付与します。

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.LDAPPrincipal hobbes \
       com.evermind.server.rmi.RMIPermission login
```

この例を実行すると、ポリシー・リポジトリである system-jazn-data.xml ファイルに次のような構成が生成されます。

```
<jazn-policy>
   <grant>
      <grantee>
         <principals>
            <principal>
               <class>oracle.security.jazn.realm.LDAPPrincipal</class>
               <name>hobbes</name>
            </principal>
         </principals>
      </grantee>
      ...
      <permissions>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
         </permission>
         ...
      </permissions>
      ...
   </grant>
   ...
</jazn-policy>
```

外部 LDAP プリンシパルへの追加のパーミッションの付与

外部 LDAP プリンシパルが要求する追加パーミッションがあれば、前項に示した RMI パーミッションと同じ方法で付与できます。

外部 LDAP プロバイダでの JAAS モードの使用

OC4J の JAAS モードは、付与されたパーミッションに関する認可のチェックでアプリケーションが要求する場合に、外部 LDAP プロバイダを使用するアプリケーションでサポートされます。次の例に示すようにして構成します。

```
<orion-application ... >
   ...
   <jazn ... jaas-mode="doAsPrivileged" />
   ...
</orion-application>
```

このモードをいつどのように使用するかの詳細は、5-6 ページの「JAAS モードの概要」および 5-20 ページの「JAAS モードの構成と使用」を参照してください。

Sun Java System Directory Server 用の構成のサンプル

この項では、外部 LDAP プロバイダとして Sun Java System Directory Server を使用するためのサンプル構成を示します。次があります。

- サンプル LDIF の説明
- OC4J 構成ファイルのサンプル・エントリ

この章で前述したように Application Server Control コンソールを使用すると、`orion-application.xml` および `system-jazn-data.xml` の設定は自動的に行われます。

注意： Sun Java System Directory 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。（同様に、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。）

サンプル LDIF の説明

前述の Sun Java System Directory Server の例では、次の LDIF 記述が使用されることが想定されています。

例 10-1 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
```

OC4J 構成ファイルのサンプル・エントリ

この項では、次に示すファイル内にある、外部 LDAP プロバイダ用の OC4J 構成を示します。

- system-jazn-data.xml 内の Sun Java System Directory Server 用設定
- orion-application.xml 内の外部 LDAP サーバー用設定

system-jazn-data.xml 内の Sun Java System Directory Server 用設定

Sun Java System Directory Server のインストールが、10-12 ページの例 10-1 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。`system-jazn-data.xml` ファイル内の対応する `<jazn-loginconfig>` エントリを、次の例に示します。

例 10-2 例 10-1 に対応する JAAS ログイン・モジュール構成

```
<jazn-data ... >
   ...
   <jazn-loginconfig>
      <application>
         <name>callerInfo</name>
         <login-modules>
            <login-module>
               <class>oracle.security.jazn.login.module.LDAPLoginModule</class>
               <control-flag>required</control-flag>
               <options>
                  <option>
                     <name>oracle.security.jaas.ldap.provider.url</name>
                     <value>ldap://myhost.example.com:389</value>
                  </option>
                  <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>
   ...
</jazn-data>
```

orion-application.xml 内の外部 LDAP サーバー用設定

外部 LDAP プロバイダ用として、orion-application.xml の次の設定が使用されます。

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

system-jazn.data.xml 内のログイン・モジュール情報を同期化するには、OC4J を再起動する必要があります。

外部 LDAP プロバイダでの SSL の使用

この項では、OC4J で外部 LDAP プロバイダを使用するときに、Secure Sockets Layer 通信を使用する方法について説明します。この項の内容は次のとおりです。

- 外部 LDAP プロバイダの初期の SSL の注意事項
- 外部 LDAP プロバイダで SSL を使用する OC4J の構成
- 外部 LDAP プロバイダの SSL の構成

関連項目： OC4J サーバー・サイド SSL サポートの詳細は次を参照してください。

- 第 15 章「OC4J との SSL 通信」

外部 LDAP プロバイダの初期の SSL の注意事項

1-4 ページの「SSL 認証」では、Secure Sockets Layer 通信の認証、一方向認証、双方向認証の様々な認証モードについて説明しています。大多数の LDAP プロバイダは「認証なし」モードをサポートしていないので注意してください（Oracle Internet Directory ではサポートしています）。

外部 LDAP プロバイダで SSL を使用する OC4J の構成

外部 LDAP プロバイダに対して SSL 通信を有効にするには、LDAP プロバイダ・オプション oracle.security.jaas.ldap.provider.url に、LDAPS プロトコル、ホスト名、SSL 通信用の適切なポートを示す URL 値を設定します。構文は、前述の LDAP URL と似ていますが、LDAPS のデフォルト・ポートは 389 ではなく 636 です。

外部 LDAP プロバイダ構成の Application Server Control コンソールを使用する場合、このオプションは「LDAP ロケーション」の設定に対応します（10-3 ページの「デプロイ時の外部 LDAP プロバイダの指定および構成」を参照）。これは 10-7 ページの「system-jazn-data.xml 内の外部 LDAP プロバイダ設定」でも説明しています。

たとえば、「LDAP ロケーション」を ldaps://myhost.example.com:636 に設定すると、system-jazn-data.xml 内に設定されている oracle.security.jaas.ldap.provider.url は次のようになります。

```
<login-module>
   <class>oracle.security.jazn.login.module.LDAPLoginModule</class>
   <control-flag>...</control-flag>
   <options>
      <option>
         <name>oracle.security.jaas.ldap.provider.url</name>
         <value>ldaps://myhost.example.com:636</value>
      </option>
      ...more options...
   </options>
</login-module>
```

重要： この構成は、「認証なし」モードの SSL には十分ですが、外部 LDAP プロバイダは（LDAP サーバーとの通信用に）少なくとも一方向認証を必要とする可能性があります。追加の手順の詳細は、次項の「外部 LDAP プロバイダの SSL の構成」で説明します。

外部 LDAP プロバイダの SSL の構成

OC4J と外部 LDAP プロバイダ間で SSL 通信を行うには次が必要です。

- SSL を使用するように外部 LDAP プロバイダを構成します。証明書を含む Wallet またはキーストアと、その証明書を発行した（外部 LDAP プロバイダ証明書を発行した）CA の証明書が必要です。
- SSL を使用するように OC4J を構成する必要があります。第 15 章「OC4J との SSL 通信」を参照してください。Wallet またはキーストアには、コンテナを識別する証明書と、その証明書を発行した（OC4J 証明書を発行した）CA の証明書を格納する必要があります。
- プロセスで使用するすべてのトラスト・ポイント（CA）は、外部 LDAP プロバイダおよび OC4J によって、Wallet またはキーストアにインポートする必要があります。OC4J 証明書と外部 LDAP プロバイダ証明書の署名に異なる CA が使用される場合は、OC4J と外部 LDAP プロバイダの Wallet またはキーストアのそれぞれに、これらの各 CA の証明書のコピーが含まれている必要があります。

次に示す汎用的な手順で、標準的な JSSE 機能と一方向認証を使用する SSL 用に Active Directory や Sun Java System Directory Server などの外部 LDAP プロバイダを構成できます（現在のリリースでは外部 LDAP プロバイダに双方向認証はサポートされません）。

1. ディレクトリ・サーバーからローカル・マシンにルート CA 証明書をインポートします。たとえば、固有のキーストアを作成し、このキーストアにルート CA 証明書をインポートします。これは JSSE `keytool` で実行できます。
2. トラストストアに対して必要に応じて Java システム・プロパティを設定します。
   - トラストストアとして機能する Wallet またはキーストアへのパスを示す `-Djavax.net.ssl.trustStore` プロパティを設定します。
   - トラストストアをセキュアにする必要がある場合は、`-Djavax.net.ssl.trustStorePassword` プロパティも設定します。（通常、トラストストアには公開鍵のみ含まれるため、一方向認証ではこの設定は必須ではありません。）

   スタンドアロン OC4J では、JVM コマンドラインでこの設定を行います。Oracle Application Server 環境では、`opmn.xml` ファイル内の OC4J インスタンスの Java プロパティ設定からこれを実行します。

OC4J `home` インスタンスのサンプル `opmn.xml` エントリを示します。

```
<ias-component id="OC4J">
   <process-type id="home" module-id="OC4J" status="enabled">
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-Xrs -server
                  -Djava.security.policy=
                      $ORACLE_HOME/j2ee/home/config/java2.policy
                  -Djava.awt.headless=true
                  -Dhttp.webdir.enable=false"
                  -Djavax.net.ssl.trustStore=pathtotruststore/>
         </category>
         ...
      </module-data>
      ...
   </process-type>
</ias-component>
```

関連項目：

- keytool の詳細は、15-3 ページの「OC4J および Oracle HTTP Server での鍵と証明書の使用」および 15-5 ページの「スタンドアロン OC4J での SSL の使用」を参照してください。
- keytool ユーティリティの詳細は、次の URL を参照してください。

  ```
  http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/
  keytool.html
  ```

- 『Oracle Containers for J2EE 構成および管理ガイド』の OC4J ランタイムの構成の章には、システム・プロパティの設定に関する一般的な情報が記載されています。

11

Oracle Access Manager

この章では、Oracle Access Manager セキュリティ・プロバイダ（旧称 Oracle COREid Access and Identity）を認証、認可およびシングル・サインオンに使用する方法について説明します。この章は、Oracle Access Manager システムのデプロイを行ったユーザーまたは計画しているユーザー向けです。Oracle Access Manager 10.1.4 実装と OC4J 10.1.3.1 実装の統合、および OC4J にデプロイされたアプリケーションを Oracle Access Manager の機能を介して保護する方法について説明します。これには、Web アプリケーションの詳細な構成手順、および Web アプリケーション、EJB、Web サービス認証方式（Username トークン、X.509 トークンおよび SAML トークンを含む）の例が含まれます。

この章の内容は次のとおりです。

注意：

- この章では、OC4J および OracleAS JAAS Provider から Oracle Access Manager を Oracle Application Server で使用できるようにするために必要な手順に重点を置いています。Oracle Access Manager についてある程度習熟していることを想定しており、その機能またはツールの詳細な説明はありません。Oracle Access Manager の設定の必要事項としては、設定の実行方法よりも、必須の設定およびそれらの OracleAS JAAS Provider 構成へのマッピング方法を中心に示してあります。Oracle Access Manager の機能および管理の詳細は、Oracle Access Manager のドキュメントを参照してください。
- OC4J の 10.1.3.1 実装では、Oracle Access Manager の 10.1.4 実装のみがサポートされています。

関連項目：

- 『Oracle Access Manager インストレーション・ガイド』
- Access Server の設定（この章で後述）およびアクセス制御とアプリケーションとデータへのユーザー・アクセスの定義に関連する情報は、『Oracle Access Manager System Administration Guide』を参照してください。
- Identity Server の設定および Access Server と Identity Server の両方に共通する機能（どちらもこの章で後述）に関連する情報は、『Oracle Access Manager ID および共通管理ガイド』を参照してください。
- 『Oracle Access Manager 開発者ガイド』

これらのドキュメントを含め、Oracle Access Manager ドキュメントは、次の URL で入手できます。

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

「Application Servers」の下にリストされている Oracle Identity Management 10.1.4 ドキュメント・リリースで入手できます。

Oracle Access Manager について

この項では、Oracle Access Manager の概要、前提条件およびアーキテクチャについて説明します。

Oracle Access Manager の概要

Oracle Access Manager は、集中的なセキュリティ管理を提供するエンタープライズ・クラスの認証、認可および監査ソリューションです。様々なアプリケーション・サーバー、レガシー・アプリケーションおよびデータベースにわたる異機種間アプリケーション環境でアクセス制御、シングル・サインオン（Oracle Single Sign-On とは別）、パーソナライズおよびユーザー・プロファイル管理を行うための機能を備えています。Oracle Access Manager は、アクセス・ポリシーの作成、管理および強制を行う上で重要な機能を提供します。異なるデータ・セットへのアクセスを必要とする複数のユーザー・グループがあり、すべてのユーザー・グループが共通のデータ・セットへのアクセスを必要とする場合は、Oracle Access Manager を使用し、すべてのユーザーが適切なデータにのみアクセスできるように、各グループに正しいアクセス・レベルを許可できます。

Oracle Access Manager を、他の認証、シングル・サインオンおよび認可サービスと比較すると、機能面で次の違いがあります。

- 複数の OC4J インスタンスの認証および認可を、1 つの Oracle Access Manager インスタンスで一元化でき、集中的なシングル・サインオンおよび監査機能、ならびにより堅牢な認証オプションを使用できます。
- Oracle Access Manager は、ワークフロー、密な属性管理および管理の委任による高度なアイデンティティ管理を提供します。
- Oracle Access Manager では、指定したアイデンティティ・プロファイルを持つメンバーが含まれる動的グループをベースにしたアクセス制御がサポートされます。
- Oracle Access Manager は、アクセスとアイデンティティのリアルタイムでの統合を可能にします。Oracle Access Manager による実行時の変更は、Access Server キャッシュに自動的に移入され、セキュリティ・ホールを解消します。

OC4J 10.1.3.x 実装では、OracleAS JAAS Provider が、カスタム・ログイン・モジュールと特別な認証方式設定を使用して、Oracle Access Manager における統合をサポートします。

Oracle Access Manager には、次のコンポーネントがあります。

- WebGate はポリシー強制用コンポーネントで、Web サーバーのプラグイン・アクセス・クライアント（Oracle HTTP Server で使用する関連 Apache mod を含む）であり、HTTP リクエストを捕捉し、認証および認可のために Access Server に転送します。これに対して、AccessGate コンポーネントは、Oracle Access Manager SDK を使用して構築されるカスタム・アクセス・クライアントであり、ユーザーまたはアプリケーションからの Web リソースおよび非 Web リソースのリクエストを処理します。ユーザー・リクエストを捕捉し、認証および認可のために Access Server に転送します。用語「WebGate」と「AccessGate」は、ほとんどの文脈で互いに読み換えることができます。
- WebPass は、Web サーバーと Oracle Access Manager Identity Server 間の情報の受渡しを行う Web サーバー・プラグインです。
- Oracle Access Manager Identity Server は、すべてのユーザー・アイデンティティ、グループ、組織および資格証明管理リクエストを処理します。
- Access Server は、ポリシー決定用コンポーネントであり、リクエストを受信し、アクセス・クライアントに応答し、ログイン・セッションを管理します。Access Server は、WebGate からリクエストを受信し、Oracle Internet Directory で認証、認可および監査ルールを問い合せます。また、Access Server は、WebGate によるセッションの終了、ユーザー・セッションのタイムアウト設定、タイムアウト発生時の再認証、セッション・アクティビティの追跡を支援するという形で、ログイン・セッションを管理します。
- Policy Manager コンポーネントは、ポリシー・データを Oracle Internet Directory（または他の適切な LDAP サーバー）に書き込み、ポリシー変更で Access Server を更新します。このコンポーネントには、管理者がポリシーおよびシステム構成の管理に使用可能なアクセス・システム・コンソールが含まれます。

注意： 10.1.3.1 実装では、Application Server Control は、Oracle Access Manager の構成をサポートしません。

Oracle Access Manager の前提条件

この項では、Oracle Access Manager を使用する前にインストールしておく必要があるものについて説明します。Oracle Access Manager の各コンポーネントのバージョンは 10.1.4 です。

関連項目：

- インストール手順は、『Oracle Access Manager インストレーション・ガイド』を参照してください。

上位レベルの前提条件としては、Oracle Access Manager および Oracle Application Server のインストール、OC4J 上での Access Manager SDK および使用アプリケーションの構成があります。

Oracle Access Manager 側の詳細な要件は、次のとおりです。

1. 適切な LDAP リポジトリ。（Oracle Application Server 10.1.4 または 10.1.2 インフラストラクチャに含まれる）Oracle Internet Directory など。
2. Web サーバー。（Oracle Application Server 10.1.3.x 中間層インフラストラクチャに含まれる）Oracle HTTP Server など。
3. Oracle Access Manager Identity Server および Access Server。Oracle Access Manager をインストールすると、使用する LDAP リポジトリの指定を要求されます。実行時に Oracle Access Manager Identity Server および Access Server と通信するためには、LDAP リポジトリがアクセス可能である必要があります。

4. Oracle Access Manager WebGate、WebPass および Policy Manager を Web サーバーにインストールすること。WebGate は SSO インターセプタであり、実行時に Access Server と通信します。WebPass は、Oracle Access Manager Identity Server と通信します。Policy Manager は LDAP リポジトリと通信します。WebGate および WebPass のインストール時に、使用する Access Server および Identity Server の指定を要求されます。

OC4J 側の詳細な要件は、次のとおりです。

1. OC4J および Oracle HTTP Server を持ち、`mod_oc4j` Apache mod を含む、Oracle Application Server 10.1.3.x 中間層のインストール。この中間層は、Oracle Access Manager 側にインストールする Web サーバーが Oracle HTTP Server であるかどうかにかかわらず、その Web サーバーから独立しています。

   Oracle Access Manager を使用する場合は、Oracle HTTP Server が必要です。スタンドアロンの OC4J は使用できません。

2. この Oracle HTTP Server にインストールされた Oracle Access Manager WebGate。

3. （必要に応じて）追加の OC4J インスタンス。通常、Oracle Access Manager SSO を使用する場合は、トポロジで複数の OC4J インスタンスが使用されるため、Oracle HTTP Server インスタンスを、複数の OC4J インスタンスをルーティングおよび保持するように構成する必要があります。

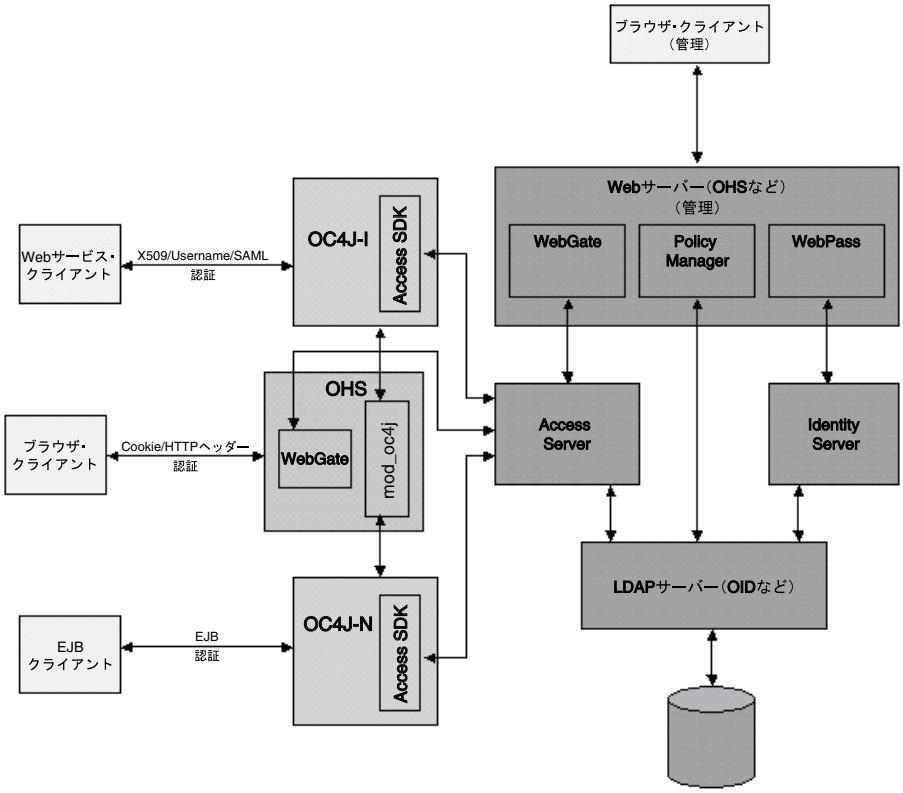
4. Access Manager SDK。OC4J と同じシステム上で、OC4J インスタンスごとに 1 つ必要です。Access Manager SDK は単独でインストールされ、OC4J が実行時に Access Server と通信するために必要となります。各 OC4J インスタンスは、Access Server インスタンスと通信するように構成されている Access Manager SDK インスタンスと通信します。

次項の「Oracle Access Manager のアーキテクチャ」では、Oracle Access Manager コンポーネントと Oracle Application Server 中間層インフラストラクチャの主要コンポーネントがどのように統合されるかを示します。

Oracle Access Manager のアーキテクチャ

図 11-1 は、Oracle Access Manager のアーキテクチャを示します。

図 11-1 Oracle Access Manager のアーキテクチャ



Oracle Access Manager の構成段階の概要

OC4J アプリケーションを Oracle Access Manager とともに使用するための構成手順には、次の 3 つの段階があります。

1. Oracle Access Manager Policy Manager インストール用の一時構成。これには、認証方式の設定および Oracle Access Manager リソース・タイプの構成が含まれます。11-8 ページの「Oracle Access Manager の構成」を参照してください。
2. 各 OC4J インスタンスの構成。これには、Access Manager SDK のインストール用の各 OC4J インスタンスの構成が含まれます。11-14 ページの「Access Manager SDK を使用する OC4J の構成」を参照してください。
3. アプリケーションの構成。これには、`web.xml` の設定、デプロイ時の設定、`orion-application.xml` の設定（デプロイ前またはデプロイ後）および JAAS ログイン・モジュールの設定が含まれます。11-17 ページの「アプリケーションの構成」を参照してください。

注意： 使用する LDAP サーバーに、必要なアカウントがあることも確認してください（11-17 ページの「LDAP サーバーでの必須アカウントの作成」を参照）。

Policy Manager の実行

この章で後述する構成手順のいくつかでは、Policy Manager の実行が必要になります。次のような URL を使用して実行し、ログインします。

```
http://host:port/access/oblix
```

これにより、この章で頻繁に使用するアクセス・システム・コンソールが表示されます。

Oracle Access Manager のコンセプト

この項では、この章で後述する内容に関連する Oracle Access Manager のいくつかのコンセプトの背景について説明します。

- Oracle Access Manager リソース・タイプの概要
- Oracle Access Manager 認証の概要
- HTTP ヘッダー変数を使用した認証
- Oracle Access Manager シングル・サインオン Cookie の概要

Oracle Access Manager リソース・タイプの概要

Oracle Access Manager では、リソース・タイプを使用して、保護対象のリソースの種類を、関連する操作とともに記述します。リソースに関連付けられる操作は、リソース・タイプの範囲内で決定できます。ポリシー・ドメインにリソースを追加するには、リソース・タイプおよび保護するリソースに関連付けられる操作を先に定義する必要があります。

たとえば、Oracle Access Manager はデフォルトで、HTTP および EJB という名前のリソース・タイプをサポートしています。HTTP リソース・タイプに対しては、`CONNECT`、`DELETE`、`GET`、`POST`、`PUT` および `TRACE` などの操作が使用可能です。EJB リソース・タイプに対しては、Bean を実行する操作 `EXECUTE` が使用可能です。カスタム・リソース・タイプに対しては、カスタム操作名を指定できます。

OC4J は、Access Server へのセッションを作成するために Access Manager SDK を使用します。SDK を使用するには、リソース・タイプおよびリソース操作が指定されている必要があります。このため、Oracle Access Manager ログイン・モジュールを構成するときに、次の項目から成る Oracle Access Manager リソース・タイプを構成する必要があります。

- 目的のリソース・タイプ名（任意）
- 目的の操作名（任意）

  単一のリソース操作のみでなく、保護リソースに対して実行する必要がある操作があればすべて指定できます。
- 保護リソースの URL

関連項目：

- Oracle Access Manager リソース・タイプの詳細は、『Oracle Access Manager System Administration Guide』を参照してください。

Oracle Access Manager 認証の概要

任意のユーザーを検証するには、Oracle Access Manager を認証方式用に構成する必要があります。認証方式はプラグインで構成されます。

OC4J による Oracle Access Manager のサポートでは、ユーザー資格証明をプロファイルにマッピングする `credential_mapping` プラグインが使用され、必要に応じて、ユーザー・パスワードを検証する `validate_password` プラグインが使用されます。これらのプラグインは、この章の後出の説明に従って構成する必要があります。

さらに、OC4J では、エンドユーザー認証（アイデンティティ・アサーション）と Oracle Access Manager を統合する次の 2 つの方法がサポートされています。

- Oracle Access Manager SSO Cookie `ObSSOCookie` の使用。詳細は、次項の「Oracle Access Manager シングル・サインオン Cookie の概要」を参照してください。
- HTTP ヘッダーで渡される、ユーザー名およびパスワードの使用。詳細は、11-8 ページの「HTTP ヘッダー変数を使用した認証」を参照してください。

関連項目：

- Oracle Access Manager プラグインの詳細は、『Oracle Access Manager System Administration Guide』を参照してください。

Oracle Access Manager シングル・サインオン Cookie の概要

Oracle Access Manager は、`ObSSOCookie` という名前の暗号化セッション Cookie を使用した、シングル・ドメインおよびマルチ・ドメインのシングル・サインオンを実装します。（これは、使用可能な 2 つのエンド・ユーザー認証方法の 1 つです。もう 1 つの、HTTP ヘッダー変数を使用する方法については次の項を参照してください。）WebGate は、認証が成功すると、ユーザーのブラウザにこの Cookie を送信します。送信された Cookie は、これ以降、このレベル以下の認証を必要とするその他の保護リソースに対する認証メカニズムとして機能します。

ユーザーがリソースへのアクセスをリクエストすると、リクエストは WebGate から Access Server に送信されます。ユーザーが検証されると、`ObSSOCookie` が設定され、OC4J に渡されます。このシングル・サインオン機能では、Oracle Access Manager は、認証資格証明の入力を要求するかわりに、Cookie を後続のリクエストに対して使用します。

OC4J は、`ObSSOCookie` を使用して Access Server に接続し、ユーザー・ロールを取得します。

注意： `ObSSOCookie` はデフォルトではセッション Cookie ですが、永続 Cookie にすることもできます。

関連項目：

- `ObSSOCookie` の詳細は、『Oracle Access Manager System Administration Guide』を参照してください。

HTTP ヘッダー変数を使用した認証

Oracle Access Manager は、HTTP ヘッダー変数を使用した認証をサポートしています。ユーザー名とパスワードが HTTP ヘッダーで渡され、エンドユーザーをアサートします。（これは、使用可能な 2 つのエンドユーザー認証方法の 1 つです。もう 1 つの Oracle Access Manager `ObSSOCookie` を使用する方法については、前項を参照してください。）

この方法を使用するには、このユーザー名およびパスワードを OC4J が Access Server へのアクセスで使用するように、Oracle Access Manager ログイン・モジュールを構成する必要があります（11-19 ページの「Oracle Access Manager ログイン・モジュールの構成」を参照してください）。

HTTP ヘッダー変数および Cookie を使用して、情報を下流のアプリケーションに渡す場合は、HTTP ヘッダーのサイズ制限が 4K であることに注意してください。HTTP ヘッダーのこのサイズ制限は、すべての Cookie、サーバー変数および環境変数、つまり、HTTP ヘッダーのコンテンツの合計に対して適用されます。コンテンツが 4K の制限を超過しなければ、HTTP ヘッダーが含むことのできる個々の要素の数には制限はありません。このため、HTTP ヘッダーの使用可能な領域を評価する場合には、Oracle Access Manager とその他のアプリケーションで使用するデータのバイト・サイズを考慮に入れます。たとえば、Oracle Access Manager とその他のアプリケーションが、合計 1K を HTTP ヘッダーで使用する場合は、ユーザーのデータ用に 3K を使用できます。

Oracle Access Manager の構成

この項では、Oracle Access Manager のインストール用の一時的な構成について説明します。

1. Oracle Access Manager フォームベース認証の構成
2. Oracle Access Manager Basic 認証の構成
3. リソース・タイプの構成
4. アクション URL の保護

Oracle Access Manager フォームベース認証の構成

シングル・サインオン機能では、Basic 認証方式に制限があるため、フォームベース認証方式をリソース保護のために使用する必要があります。（ただし、構成によってはパスワードなしの認証を使用する必要があります。11-11 ページの「Oracle Access Manager Basic 認証の構成」を参照してください。）

次に示す手順では、WebGate がリソースを保護できるように、Oracle Access Manager のフォームベース認証で使用されるログイン・ページを作成および保護します。後から、このフォームベース認証によって保護されるようにアプリケーションを構成します。

1. ログイン・フォームの作成
2. Policy Manager におけるフォームベース認証の定義
3. credential_mapping プラグインのフォームベース認証用構成
4. validate_password プラグインのフォームベース認証用構成

関連項目：

- フォームベース認証の構成方法については、『Oracle Access Manager System Administration Guide』（特に関連する付録）を参照してください。

ログイン・フォームの作成

フォームベース認証用のログイン・ページを作成します。このページで設定するパラメータのいくつかに対応して、Policy Manager および関連するプラグインで設定を行います。これについては、必要に応じて説明します。

ログイン・ページを中間層システムの *OHS_HOME*/*document_root* ディレクトリ（通常は *ORACLE_HOME*/Apache/Apache/htdocs）の下に置きます。

次に、サンプルのログイン・ページ、login.html を示します。このページは、*ORACLE_HOME*/Apache/Apache/htdocs/login ディレクトリにあるものと想定します。

```
<html>
<head>
<title> COREid SSO Login Page</title>
<body bgcolor="white">
<h1 align="center">COREid SSO Provider Example : Sign in</h1>
<form method="POST" action="/coreid/access/test.html" >
  <table border="0" cellspacing="5">
    <tr>
      <th align="right">Username:</th>
      <td align="left"><input type="text" name="usernamevar"></td>
    </tr>
    <tr>
      <th align="right">Password:</th>
      <td align="left"><input type="password" name="passwordvar"></td>
    </tr>
    <tr>
      <td align="right"><input type="submit" value="Log In"></td>
      <td align="left"><input type="reset"></td>
    </tr>
  </table>
</form>
</body>
</html>
```

POST メソッドに対するアクション URL は任意に指定できますが、次の手順で Policy Manager の認証管理を構成する際に、同じアクション URL を指定する必要があります。

ユーザー名の変数（ここでは usernamevar）は、Oracle Access Manager credential_mapping プラグインでの指定内容と一致する必要があります。パスワードの変数（ここでは passwordvar）は、Oracle Access Manager validate_password プラグインでの指定内容と一致する必要があります。

Policy Manager におけるフォームベース認証の定義

この手順では、Policy Manager を使用してフォームベース認証を定義します。Policy Manager で次のようにナビゲートします。

「アクセス・システム・コンソール」→「アクセス・システム構成」→「認証管理」

すべての認証方式をリストし、新しい認証方式がなければそれを追加します。**「一般」**タブで新しい認証方式を定義するには、次のように入力します。

```
Name:                   COREidSSOform
Description:            COREid SSO Form Based
Level:                  1
Challenge Method:       Form
Challenge Parameter:    form: /login/login.html
                        creds: usernamevar passwordvar
                        action: /coreid/access/test.html
                        passthrough: No

SSL Required:           No
Challenge Redirect
Enabled:                Yes
```

任意の名前と説明を選択できます。ここで選択している COREidSSOform および COREid SSO Form Based は単なる例です。チャレンジ・パラメータには、login/login.html がフォームとして指定されています。これは前の手順でログイン・ページを作成した Oracle HTTP Server ドキュメント・ルートに対する相対パスです。「チャレンジ・リダイレクト」は空白のままにしておきます。

ここの「creds」には、前の手順のログイン・ページでユーザーおよびパスワード用に指定した変数を指定する必要があります。これらの変数は、フォームベース認証用に credential_mapping プラグインおよび validate_password プラグインでそれぞれ使用されるものです。

アクション URL（ここでは /coreid/access/test.html）は任意ですが、ログイン・ページの POST メソッドに対するアクション URL と同じものである必要があります。この URL は、この後の「Oracle Access Manager Basic 認証の構成」で説明している、Basic（パスワードなし）認証方式で保護します。

credential_mapping プラグインのフォームベース認証用構成

次に、Policy Manager でフォームベース認証用に Oracle Access Manager credential_mapping プラグインを構成する必要があります。これを使用して、ログイン・フォームを保護します。

次のメニュー・パスで、目的のページにナビゲートします。

「アクセス・システム・コンソール」→「アクセス・システム構成」→「認証管理」

すべての認証方式をリストし、その中からフォームベース方式を選択し、**「プラグイン」**タブを表示します。

credential_mapping プラグインを、次のような入力内容で変更します。

```
obMappingBase="cn=users,dc=us,dc=oracle,dc=com",obMappingFilter="(&(&
(objectclass=inetorgperson)(uid=%usernamevar%))(|(!
(obuseraccountcontrol=*)) (obuseraccountcontrol=ACTIVATED)))"
```

uid に入力する値（ここでは usernamevar）は、ログイン・フォームのユーザー名に指定した変数、および前の各手順で示した、Policy Manager でのフォームベース認証の定義時に指定した変数と同じものである必要があります。

これは、OC4J における Oracle Access Manager ログイン・モジュール構成の coreid.name.attribute オプションの値とも一致します。

関連項目：

- credential_mapping プラグインの詳細は、『Oracle Access Manager System Administration Guide』を参照してください。

validate_password プラグインのフォームベース認証用構成

次に、Policy Manager でフォームベース認証用に Oracle Access Manager validate_password プラグインを構成する必要があります。これを使用して、ログイン・フォームを保護します。

前の手順の credential_mapping プラグインの場合と同じページにナビゲートします。validate_password プラグインを、次のような入力内容で変更します。

```
obCredentialPassword="passwordvar"
```

obCredentialPassword に入力する値（ここでは passwordvar）は、ログイン・ページのパスワード変数、および前の各手順で示した、Policy Manager でのフォームベース認証の定義時に指定した変数と同じものである必要があります。

これは、Oracle Access Manager ログイン・モジュール構成の coreid.password.attribute オプションの値とも一致します。

Oracle Access Manager Basic 認証の構成

パスワードで保護されない、Oracle Access Manager Basic 認証方式を構成する必要があります。（この方式では、`credential_mapping` プラグインのみを使用し、`validate_password` プラグインは使用しません。）この方式は次の 2 つのリソースを保護するために使用します。

- 構成するリソース・タイプに関連付けられる URL。11-13 ページの「構成済リソース・タイプの URL の構成および保護」を参照してください。Oracle Access Manager ログイン・モジュールは、この URL を使用し、Access Manager SDK を介して Access Server と通信します。
- フォーム・ページのアクション URL。11-9 ページの「ログイン・フォームの作成」および 11-9 ページの「Policy Manager におけるフォームベース認証の定義」を参照してください。この URL は、フォーム・リクエストが WebGate で捕捉され、発行される資格証明に対してルールを強制できるようにするために発行されます。

（ただし、ユーザーのアプリケーション自体は、11-8 ページの「Oracle Access Manager フォームベース認証の構成」に従い、フォームベース認証で保護する必要があります。）

次の手順で、パスワードを使用せずにリソースを保護する Basic 認証を定義します。

1. Policy Manager における Basic 認証の定義
2. credential_mapping プラグインの Basic 認証用構成

Policy Manager における Basic 認証の定義

この手順では、Policy Manager を使用して Basic 認証を構成します。Policy Manager で次のようにナビゲートします。

「アクセス・システム・コンソール」→「アクセス・システム構成」→「認証管理」

すべての認証方式をリストし、新しい認証方式がなければそれを追加します。**「一般」**タブで新しい認証方式を定義するには、次のように入力します。

```
Name:                    COREidSSONoPwd
Description:             Authentication without Password
Level:                   1
Challenge Method:        Basic
Challenge Parameter:     realm:NetPoint Basic Over LDAP
SSL Required:            No
Challenge Redirect
Enabled:                 Yes
```

任意の名前と説明を選択できます。ここで選択している COREidSSONoPwd および Authentication without Password は単なる例です。ここで示しているチャレンジ・パラメータのエントリは、ドロップダウン・リストにある選択肢の 1 つです。「チャレンジ・リダイレクト」は空白のままにしておきます。

credential_mapping プラグインの Basic 認証用構成

次に、Policy Manager で Basic 認証用に Oracle Access Manager `credential_mapping` プラグインを構成します。この構成はリソース保護のためですが、パスワードを使用していません。このため、WebGate が結果を捕捉できます。

次のメニュー・パスで、目的のページにナビゲートします。

「アクセス・システム・コンソール」→「アクセス・システム構成」→「認証管理」

すべての認証方式をリストし、その中から Basic 方式を選択し、**「プラグイン」**タブを表示します。

`credential_mapping` プラグインを、次のような入力内容で変更します。

```
obMappingBase="cn=users,dc=us,dc=oracle,dc=com",obMappingFilter="(&(&
(objectclass=inetorgperson)(uid=%usernamevar%))(|(!
(obuseraccountcontrol=*)) (obuseraccountcontrol=ACTIVATED)))"
```

この入力内容は、フォームベース認証用の `credential_mapping` プラグイン用のものと同じです。`uid` に入力する値（ここでは `usernamevar`）は、ログイン・フォームに指定したユーザー名変数 usernamevar と同じものである必要があります。

これは、Oracle Access Manager ログイン・モジュール構成の `coreid.name.attribute` オプションの値とも一致します。

関連項目：

- `credential_mapping` プラグインの詳細は、『Oracle Access Manager System Administration Guide』を参照してください。

リソース・タイプの構成

Oracle Access Manager では、リソース・タイプを使用して、保護対象のリソースの種類を、関連する操作とともに記述します。リソースに関連付けられる操作は、リソース・タイプの範囲内で決定できます。リソースに対して Oracle Access Manager リソース・タイプを構成し、リソース・タイプ、アクション URL およびアプリケーションを保護する必要があります。

リソースに対するリソース・タイプの構成は次の 3 つの部分に分かれます。いずれも Policy Manager を介して実行します。

1. リソース・タイプの名前および操作の構成
2. 構成済リソース・タイプの URL の構成および保護
3. 戻すアクション属性の構成

このリソース・タイプ情報は、後述するように、Oracle Access Manager ログイン・モジュールで必要になります。OC4J は、リソース・タイプを Access Manager SDK の API で使用し、Oracle Access Manager `ObSSOCookie` またはユーザー名に基づいてユーザー情報を取得します。

これらの構成手順が完了すると、リソース URL が、リソース・タイプに関連付けられ、構成したパスワードなしの Basic 認証方式で保護されます。

関連項目：

- 11-6 ページの「Oracle Access Manager リソース・タイプの概要」

リソース・タイプの名前および操作の構成

Policy Manager でリソース・タイプの名前および操作を構成するには、次のようにナビゲートします。

「アクセス・システム・コンソール」→「アクセス・システム構成」→「共通情報の構成」→「リソース・タイプ定義」

すべてのリソース・タイプが表示されたページで、新しいリソース・タイプの追加を選択します。

次のように入力し、新しいリソース・タイプを定義します。

```
Resource Name:        myresourcetype
Display Name:         My Resource Type Display Name
Resource Matching:    Case Insensitive
Resource Operation:   myresourceoperation
```

リソース・タイプとリソース操作には任意の名前を選択できますが、Oracle Access Manager ログイン・モジュールの構成で、`coreid.resource.type` および `coreid.resource.operation` オプションの値として使用した名前と同じものを使用する必要があります。

構成済リソース・タイプの URL の構成および保護

Policy Manager で構成済リソース・タイプの URL を構成および保護するには、次のようにナビゲートします。

「Policy Manager」→「ポリシー・ドメインの作成」

ポリシー・ドメインに対するリンクを選択します。**「リソース」**タブで、次のようなエントリを使用します。

```
Resource Type:      myresourcetype
Host Identifiers:   myhost
URL Prefix:         /myresourceurl
Description:        My Description
```

この構成の保護には、パスワードなしの方式を使用します。11-11 ページの「Policy Manager における Basic 認証の定義」で構成した Basic 方式を使用します。

ドロップダウン・リストからリソース・タイプ（この例では `myresourcetype`）を選択し、次に適切なホスト名を選択します。

URL 接頭辞は / で開始する必要があります。これはリソース・タイプの指定済 URL になります。この URL 接頭辞は、Oracle Access Manager ログイン・モジュール構成の `coreid.resource.name` オプションの値と一致している必要があります。

説明は任意です。My Description は単なる例です。

注意： ここで指定する URL と、先の各手順で認証の設定時に指定したアクション URL を混同しないようにしてください。この 2 つは別のものです。

戻すアクション属性の構成

認証後、OC4J は、認可をチェックするためユーザーのロールにアクセスする必要があります。これを可能にするには、Oracle Access Manager が正常に認証されたユーザーの適切なロールを OC4J に戻すための、Oracle Access Manager の戻りアクションを設定する必要があります。

Oracle Access Manager の戻りアクションを設定するには、次のようにナビゲートします。

「Policy Manager」→「ポリシー・ドメイン」→ myresourcetype を選択→「認可ルール」タブ→ロール名を選択→「アクション」タブ

「認可成功」セクション下で、次のエントリを追加（`myresourcetype` を使用した前述の例をそのまま使用）します。

```
Return Type:      myresourcetype
Return Name:      myresourcetype
Return Attribute: ObMyGroups
```

`ObMyGroups` は、Oracle Access Manager で定義済のアクション属性であり、認証済ユーザーのすべてのロールを戻すために使用します。

アクション URL の保護

Policy Manager を使用して、11-8 ページの「Oracle Access Manager フォームベース認証の構成」で指定したアクション URL を保護します。11-13 ページの「構成済リソース・タイプの URL の構成および保護」で説明している、リソース・タイプ URL の保護と同じ手順を使用します。

- この構成には、パスワードなしの認証方式を使用します。11-11 ページの「Oracle Access Manager Basic 認証の構成」で構成した、Basic 認証方式を使用します。
- リソース・タイプとして HTTP を使用します。
- アクション URL を指定します（この例では `/coreid/access/test.html`）。

関連項目： リソースの保護の詳細は、次のドキュメントを参照してください。

- 『Oracle Access Manager System Administration Guide』

Access Manager SDK を使用する OC4J の構成

この項では、中間層での各 OC4J インスタンスの構成手順について説明します。

前提条件として、WebGate を中間層の Oracle HTTP Server インスタンスにインストールする必要があります。この Oracle HTTP Server インスタンスにより、複数の OC4J インスタンスをサポートできます（通常サポートします）。

この項では、次の手順を説明します。

1. 必要に応じた OC4J インスタンスの作成
2. 各 OC4J インスタンスに対する Access Manager SDK の構成
3. 各 OC4J インスタンスに対する Access Manager SDK ライブラリ・パスの構成

注意： 中間層および OC4J インストールは Oracle Access Manager と同じシステムに配置されていてもかまいませんが、通常は別のシステムに配置されます。

関連項目：

- AccessGate/WebGate のインストールの詳細は、『Oracle Access Manager インストレーション・ガイド』を参照してください。

必要に応じた OC4J インスタンスの作成

通常、Oracle Access Manager SSO を使用する場合は、トポロジで複数の OC4J インスタンスが使用されるため、Oracle HTTP Server インスタンスを、複数の OC4J インスタンスをルーティングおよび保持するように構成する必要があります。

1. 必要に応じて、新しい OC4J インスタンスを `createinstance` ユーティリティによって作成します（『Oracle Containers for J2EE 構成および管理ガイド』を参照してください）。
2. 各 OC4J インスタンスを Oracle HTTP Server インスタンスに関連付ける必要があります。OC4J インスタンスにデプロイされる各アプリケーションは、リクエストが OC4J インスタンスに正しくルーティングされるように、`mod_oc4j` 構成ファイル、*`ORACLE_HOME`*`/Apache/Apache/conf/mod_oc4j.conf` 内で構成される必要があります。この構成は、OC4J インスタンスの作成時に自動的に行われます。

各 OC4J インスタンスに対する Access Manager SDK の構成

Oracle Access Manager SDK は、OC4J と同じシステム上に、OC4J インスタンスごとに 1 つずつインストールする必要があります。Access Manager SDK は、OC4J が実行時に Access Server と通信するために必要です。OC4J に対しては、SDK 初期化のため、起動時に Access Manager SDK の場所が指定されている必要があります（この章で後述するように、`java.library.path` プロパティを使用します）。この初期化は、最低 1 つのアプリケーションが Oracle Access Manager をセキュリティ・プロバイダとして使用している場合にのみ行われる点に注意してください。また、次の点にも注意してください。

1. OC4J インスタンスごとに独立した Access Manager SDK インストールを、OC4J と同じシステム上に作成します。同一システム上に、複数の Access Manager SDK インストールを持つことができます。
2. 各 Access Manager SDK を、適切な Access Server で動作するように構成します。*`Access_SDK_Home`*`/access/oblix/tools/configureAccessGate` ディレクトリから、コマンド `configureAccessGate` を実行します。このユーティリティでは、Access Server ID、AccessGate ID およびその他の関連パラメータが必須です。
3. Access Manager SDK 内の Oracle Access Manager ファイル `jobaccess.jar` を、OC4J のパスにコピーします。このファイルは、*`Access_SDK_Home`*`/AccessServerSDK/oblix/lib` ディレクトリにあります。ディレクトリ *`ORACLE_HOME`*`/j2ee/home/lib/ext` を（存在しない場合は）作成し、`jobaccess.jar` をこのディレクトリにコピーします。

関連項目：

- Access Manager SDK のインストールの詳細は、『Oracle Access Manager 開発者ガイド』を参照してください。
- `configureAccessGate` ユーティリティの詳細は、『Oracle Access Manager System Administration Guide』を参照してください。

各 OC4J インスタンスに対する Access Manager SDK ライブラリ・パスの構成

OC4J インスタンスが実行時に java.library.path プロパティを使用して Access Manager SDK にアクセスできるように、*ORACLE_HOME*/opmn/conf/opmn.xml ファイル内でこのプロパティを OC4J インスタンスごとに構成する必要があります。このプロパティを、SDK の場所を参照するように設定します。

次に、Windows システムでの例を示します。

```
-Djava.library.path=C:¥CoreID¥AccessSDK¥AccessServerSDK¥oblix¥lib
```

次項の「Oracle Access Manager に対する opmn.xml の構成」では、この設定をさらに詳しく示します。

関連項目：

- OPMN および opmn.xml ファイルの詳細は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

Oracle Access Manager に対する opmn.xml の構成

OC4J が OPMN によって管理されている状況で Oracle Access Manager を使用する場合は、次のように Oracle HTTP Server および OC4J に関して opmn.xml に設定を追加します。

1. OC4J については、プロセス・タイプ home、OC4J_SOA および Oracle Access Manager を使用するアプリケーションがデプロイされるその他の OC4J インスタンスで、次の作業を実行します。
   a. LD_ASSUME_KERNEL 環境変数を、値 2.4.19 に設定します。
   b. LD_LIBRARY_PATH 環境変数を、AccessServerSDK ライブラリ・パスを指すように設定します。
   c. java.library.path に、AccessServerSDK ライブラリ・パスを開始パラメータとして追加します。

   OC4J インスタンスを再起動します。
2. Oracle HTTP Server については、プロセス・タイプ HTTP_Server で、LD_ASSUME_KERNEL を 2.4.19 に設定し、Oracle HTTP Server インスタンスを再起動します。

次に示すのは、OC4J home インスタンス用に設定した opmn.xml の例です。これらの設定を、OC4J_SOA インスタンスおよび該当するその他の OC4J インスタンスに対して繰り返します。

```
<ias-component id="OC4J">
   <process-type id="home" module-id="OC4J" status="enabled">
      <environment>
         <variable id="LD_ASSUME_KERNEL" value="2.4.19"/>
         <variable id="LD_LIBRARY_PATH"
                value="/your_asdk_home/AccessServerSDK/oblix/lib" append="true"/>
      </environment>
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-server ...
                  -Djava.library.path=/your_asdk_home/AccessServerSDK/oblix/lib
                  ... />
         </category>
         ...
      </module-data>
      ...
   </process-type>
   ...
</ias-component>
```

次に示すのは Oracle HTTP Server の例です。

```
<ias-component id="HTTP_Server">
   <process-type id="HTTP_Server" module-id="OHS">
      <environment>
         <variable id="LD_ASSUME_KERNEL" value="2.4.19" />
      </environment>
      ...
   </process-type>
   ...
</ias-component>
```

LDAP サーバーでの必須アカウントの作成

使用する LDAP ディレクトリ・サーバー（Oracle Internet Directory など）では、OC4J および Application Server Control 10.1.3.x 実装で次のアカウントが必要になります。

- `oc4jadmin` ユーザー
- メンバー `oc4jadmin` を含んだ `oc4j-administrators` ロール
- `oc4j-app-administrators` ロール
- メンバー `oc4jadmin` を含んだ `ascontrol_admin`（Application Server Control を含めたすべての SOA コントロールの管理ロール）
- `ascontrol_appadmin`（Application Server Control の必須ロール）
- `ascontrol_monitor`（Application Server Control の必須ロール）

Oracle Internet Directory を使用する場合は、8-6 ページの「Oracle Internet Directory と OC4J の関連付け」で説明されているように、OC4J インスタンスを Oracle Internet Directory インスタンスに関連付けるときにこれらのアカウントが自動的に作成されます。（8-9 ページの「Oracle Internet Directory に作成される必須アカウント」はこの項に含まれています。）

外部の LDAP プロバイダを使用する場合は、10-10 ページの「管理ユーザーとロールの作成および RMI パーミッションの付与」で説明されているように、手動でアカウントを作成する必要があります。

関連項目：

- 4-12 ページの「事前定義アカウント」

アプリケーションの構成

この項では、Web アプリケーションを対象とした次の手順を説明します。

1. web.xml でのアプリケーション URL の保護
2. アプリケーション・デプロイメントの設定
3. orion-application.xml における Oracle Access Manager SSO の構成
4. Oracle Access Manager でのアプリケーション URL の保護
5. Oracle Access Manager ログイン・モジュールの構成
6. アプリケーションのテスト

web.xml でのアプリケーション URL の保護

アプリケーション保護の手順 1 は、標準 J2EE 機能を使用して、web.xml ファイル内の設定によって目的の URL または URL 接頭辞を保護することです。

これらは、11-19 ページの「Oracle Access Manager でのアプリケーション URL の保護」で説明した、Oracle Access Manager 構成によって保護する URL と同じものです。

アプリケーション・デプロイメントの設定

Oracle Application Server 10.1.3.x 実装では、Application Server Control がまだ Oracle Access Manager をセキュリティ・プロバイダとしてサポートしていません。Application Server Control コンソールを使用してアプリケーションをデプロイする場合は、ファイルベース・プロバイダを選択してください。この設定は、この章で説明している各構成手順でオーバーライドされます。

orion-application.xml における Oracle Access Manager SSO の構成

Oracle Access Manager Single Sign-On を Web アプリケーションの認証方式として使用するには、OC4J の orion-application.xml ファイルの <jazn-web-app> 要素内で、auth-method 属性を COREIDSSO に設定します。これは、デプロイ前の手順（EAR ファイルへのパッケージ化）またはデプロイ後の手順として実行できます。

注意：

- web.xml ファイルには、<auth-method> 設定は不要です。web.xml 内の設定は、すべて orion-application.xml 内の COREIDSSO 設定でオーバーライドされるためです。
- <jazn-web-app> 要素は、orion-web.xml ファイルでもサポートされます。競合が発生する場合は、問題の Web アプリケーションに関しては、orion-web.xml が orion-application.xml よりも優先されます。

次に、orion-application.xml におけるエントリのサンプルを示します。ここでは、<jazn-web-app> は <jazn> 要素のサブ要素です。

```
<orion-application ... >
   ...
   <jazn provider="XML" >
      <jazn-web-app auth-method="COREIDSSO"/>
      ...
   </jazn>
   ...
</orion-application>
```

Oracle Access Manager でのアプリケーション URL の保護

Policy Manager を使用し、フォームベース認証を介して、ユーザーのアプリケーションの URL または URL 接頭辞を保護します。これらは、11-18 ページの「web.xml でのアプリケーション URL の保護」で使用したものと同じ URL になります。次のようにナビゲートします。

「Policy Manager」→「ポリシー・ドメインの作成」

次に、該当するパブリック・ポリシー・ドメインを選択します。`web.xml` で保護した各 URL または URL 接頭辞を、次の手順で保護します。

1. リソース・タイプとして HTTP を使用します。
2. URL を指定します（たとえば `/foo` など）。
3. この構成を、11-8 ページの「Oracle Access Manager フォームベース認証の構成」で定義したフォームベース認証方式で保護します。

関連項目： リソースの保護の詳細は、次のドキュメントを参照してください。

- 『Oracle Access Manager System Administration Guide』

Oracle Access Manager ログイン・モジュールの構成

Web アプリケーションに対して、OC4J 実装が Oracle Access Manager をサポートするには、Oracle 提供のログイン・モジュール `CoreIDLoginModule` が必要です。次のテンプレートは、`system-jazn-data.xml` ファイルの一般的な構成を示します。`<class>` および `<control-flag>` 要素の設定に注意してください。次に示す例に続いて、表 11-1 で使用可能なオプションについて説明します。個々のシナリオおよびそれぞれの構成のその他の例は、この章で後述します。

注意： 規則上、Oracle Access Manager ログイン・モジュールでも、その他のカスタム・ログイン・モジュールの場合と同様に、`<jazn>` の設定 `provider="XML"` が使用されます。

関連項目：

- `<control-flag>` 設定の詳細は、9-17 ページの表 9-5「ログイン・モジュール制御フラグ」を参照してください。

```
<application>
   <name>yourappname</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            ...
         </options>
      </login-module>
   </login-modules>
</application>
```

表 11-1 Oracle Access Manager ログイン・モジュールのオプション

オプション名	必須 / オプション	オプションの値
addAllRoles	Required	このフラグを `true` に設定することで、認証済ユーザーが、自身のすべてのロールに関するパーミッションを持つようになります。`false` に設定すると、トップ・レベルのロールのパーミッションのみを持つようになり、ネストされたロールのパーミッションは持てません。
coreid.resoure.type	Required	Policy Manager で定義したリソース・タイプ名。 **関連項目：**11-6 ページの「Oracle Access Manager リソース・タイプの概要」および 11-13 ページの「リソース・タイプの名前および操作の構成」
coreid.resource.operation	Required	`coreid.resource.type` で指定したリソース・タイプ（Policy Manager で定義したリソース・タイプと同じ）に関連付けられているリソース操作名。 **関連項目：**11-13 ページの「リソース・タイプの名前および操作の構成」
coreid.resource.name	Required	`coreid.resource.type` で指定したリソース・タイプに関連付けられている URL 接頭辞。Policy Manager で定義した、パスワードなしの Basic 認証方式で保護されます。 **関連項目：**11-13 ページの「構成済リソース・タイプの URL の構成および保護」
coreid.name.attribute	Required	認証対象ユーザー名の変数。`credential_mapping` プラグインで定義した変数を指定します。 **関連項目：**11-7 ページの「Oracle Access Manager 認証の概要」および 11-10 ページの「credential_mapping プラグインのフォームベース認証用構成」
coreid.password.attribute	必須（X.509 トークンまたは SAML 認証を使用する場合を除く）	認証用パスワードの変数。`validate_password` プラグインで定義した変数を指定します。 **関連項目：**11-10 ページの「validate_password プラグインのフォームベース認証用構成」
coreid.name.header	Optional	HTTP ヘッダー変数を認証で使用する場合は、このパラメータが、OC4J が Oracle Access Manager Access Server に対する認証で使用するユーザー名になります。 **関連項目：**11-8 ページの「HTTP ヘッダー変数を使用した認証」および 11-26 ページの「Oracle Access Manager を介して HTTP ヘッダー変数を使用する Web アプリケーション」
coreid.password.header	Optional	HTTP ヘッダー変数を認証で使用する場合は、このパラメータが、`coreid.name.header` で指定されたユーザー名とともに OC4J が Access Server に対する認証で使用するパスワードになります。

注意： coreid.resource.type、coreid.resource.operation および coreid.resource.name の値は、11-12 ページの「リソース・タイプの構成」で説明したように、一時的な Oracle Access Manager 構成で設定し、Oracle Access Manager の同一インストールを使用するすべてのアプリケーションで同じ値になります。これらのプロパティ値は、Oracle Access Manager ログイン・モジュールに対する各アプリケーションの構成で適切に設定する必要があります。この作業は、Application Server Control または OracleAS JAAS Provider Admintool を介して行います。

次のサンプルは、11-8 ページの「Oracle Access Manager フォームベース認証の構成」、11-11 ページの「Oracle Access Manager Basic 認証の構成」および 11-12 ページの「リソース・タイプの構成」を通じて出現している例の全体像です。

```
<application>
   <name>foo</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>usernamevar</value>
            </option>
            <option>
               <name>coreid.password.attribute</name>
               <value>passwordvar</value>
            </option>
         </options>
      </login-module>
   </login-modules>
</application>
```

（このサンプルでは、coreid.name.header および coreid.password.header を除いて、Oracle Access Manager ログイン・モジュールがサポートするすべてのオプションが使用されています。使用しないオプションの使用例は、この章で後述します。）

アプリケーションのテスト

Web アプリケーションのデプロイ、OC4J の再起動および Oracle HTTP Server の再起動を行った後で、アプリケーションを実行します。次の例では、Oracle HTTP Server がポート 6666 をリスニングすることを想定しています。

```
http://www.example.com:6666/foo
```

WebGate は、このリクエストを捕捉し、この URL に対する認証方式をチェックします。この章で前述した構成を使用すると、ユーザーに対し、11-9 ページの「ログイン・フォームの作成」で作成した `login.html` ログイン・フォームが表示され、入力を要求されます。入力後、次のように処理が行われます。

1. WebGate は、ユーザー名およびパスワードをログイン・フォームから捕捉し、Access Server と通信します。
2. Access Server は、Oracle Internet Directory（または使用する他の LDAP リポジトリ）と通信します。
3. ユーザー認証後、Oracle Access Manager SSO トークンが WebGate に戻されます。
4. WebGate は、`ObSSOCookie` を設定し、この Cookie および他の HTTP ヘッダーを `mod_oc4j` に渡します。mod_oc4j は、リクエストを該当する OC4J インスタンスにルーティングします。
5. OC4J は、Cookie を取得して検証するか、この Cookie に関連付けられたユーザーに対するロールを、OC4J 上に構成された Access Manager SDK を使用して Access Server から取得します。

Oracle Access Manager プリンシパルへのパーミッションの付与

アプリケーションで権限を必要とするすべての Oracle Access Manager プリンシパルに対し、必要なパーミッションを付与する必要があります。EJP アプリケーションの場合は、EJB アクセスに必要な `RMIPermission` login を付与します。

この項の内容は次のとおりです。

Oracle Access Manager プリンシパルへの RMI パーミッションの付与

EJB アプリケーションに Oracle Access Manager を使用する場合は、EJB アクセスに必要な RMI パーミッション login を、Oracle Access Manager プリンシパルに付与する必要があります。

次の例では、OracleAS JAAS Provider Admintool を使用して、プリンシパル `orcladmin` に対してこの作業を行っています。

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.CoreIDPrincipal \
      orcladmin com.evermind.server.rmi.RMIPermission login
```

この例を実行すると、`system-jazn-data.xml` ファイルに次のような構成が生成されます。

```
<jazn-policy>
   <grant>
      <grantee>
         <principals>
            <principal>
               <class>oracle.security.jazn.realm.CoreIDPrincipal</class>
               <name>orcladmin</name>
            </principal>
         </principals>
      </grantee>
```

```
      ...
      <permissions>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
         </permission>
         ...
      </permissions>
      ...
   </grant>
   ...
</jazn-policy>
```

重要： 11-24 ページの「Oracle Access Manager プリンシパルの構成済レルム名の確認」も参照してください。

追加の Oracle Access Manager プリンシパルへの必須パーミッションの付与

Oracle Access Manager を使用する場合は、認証は Oracle Access Manager 側で行われますが、JAAS 認可は OC4J 側で行われます。（その他のレベルの認可は、Oracle Access Manager 側で行われます。）アプリケーションでの JAAS 認可が正常に行われるためには、認証後にアプリケーション・サブジェクトに移入される Oracle Access Manager プリンシパルに適切なパーミッションを付与する必要があります。これらの権限は、`system-jazn-data.xml` ファイルに保存する必要があります。

ここでの説明は、プリンシパル `BPMSystemAdmin` が `ServerPermission` server を必要としていることを想定しています。次の例では、これを行うために、OracleAS JAAS Provider の Admintool を使用しています。

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.CoreIDPrincipal \
       BPMSystemAdmin com.collaxa.security.ServerPermission server
```

この例を実行すると、`system-jazn-data.xml` ファイルに次のような構成が生成されます。

```
<jazn-policy>
   <grant>
      <grantee>
         <principals>
            <principal>
               <class>oracle.security.jazn.realm.CoreIDPrincipal</class>
               <name>BPMSystemAdmin</name>
            </principal>
         </principals>
      </grantee>
      ...
      <permissions>
         <permission>
            <class>com.collaxa.security.ServerPermission</class>
            <name>server</name>
            <actions>all</actions>
         </permission>
         ...
      </permissions>
      ...
   </grant>
   ...
</jazn-policy>
```

重要：

- パーミッションを付与する前に、パーミッション・クラスがクラスパスに含まれていることを確認してください。
- 11-24 ページの「Oracle Access Manager プリンシパルの構成済レルム名の確認」も参照してください。

次に、BPMSystemAdmin ロールのサンプルの構成を示します。

```
<role>
    <name>BPMSystemAdmin</name>
    <guid>3E9D3A5037A311DBBFA2B1BC62ED9FBC</guid>
    <members>
        <member>
            <type>user</type>
            <name>bpeladmin</name>
        </member>
        <member>
            <type>user</type>
            <name>oc4jadmin</name>
        </member>
    </members>
</role>
```

注意：

- Oracle Access Manager を使用するアプリケーション向けに OC4J の JAAS モードがサポートされており、付与したパーミッションに関して認可を確認する際にアプリケーションでこのモードを使用できるようになっています。このモードをいつどのように使用するかの詳細は、5-6 ページの「JAAS モードの概要」および 5-20 ページの「JAAS モードの構成と使用」を参照してください。
- 認可が正しく機能するように、ロール・マッピングが適切に設定されてデプロイ・ロールが J2EE 論理ロールに正しくマップされるようになっていることも確認します。追加情報は、6-11 ページの「セキュリティ・ロールのマッピング」を参照してください。

Oracle Access Manager プリンシパルの構成済レルム名の確認

Oracle Access Manager プリンシパルのパーミッション構成では、構成された各プリンシパル名が、レルム名を含めたプリンシパル名と正確に一致している必要があります。レルム名は、プリンシパルがサブジェクトに移入されるときに Oracle Access Manager から取り込まれます。

たとえば、BPMSystemAdmin が Oracle Access Manager の abc レルムに属している場合は、system-jazn-data.xml 内でこのプリンシパル名が次のように指定されている必要があります。

```
<grantee>
   <principals>
      <principal>
         <class>oracle.security.jazn.realm.CoreIDPrincipal</class>
         <name>abc/BPMSystemAdmin</name>
      </principal>
   </principals>
</grantee>
```

重要：

- Oracle Access Manager を使用する場合は、Admintool の `addrealm` オプションを使用しないでください。これを使用すると、構成内のレルム情報が不正確になります。（このコマンドは、ファイルベースのプロバイダのみに使用します。）
- 問題が発生した場合は、`system-jazn-data.xml` で、パーミッション付与の対象となったプリンシパル名に適切なレルム情報のみが含まれていることを確認してください。

Oracle Application Server SOA アプリケーションに関する注意事項

この項では、Oracle Access Manager を使用して Application Server Control や OWSM などの Oracle Application Server SOA アプリケーションを保護する際に特に注意する点について説明します。内容は次のとおりです。

Oracle Application Server SOA アプリケーションのログアウトの構成

Oracle Access Manager で保護されている Oracle Application Server SOA アプリケーションのログアウトを正しく機能させるには、次の手順を実行します（Oracle HTTP Server が Policy Manager の Web サーバーであると想定しています）。

1. すべてのアプリケーションに対して共有ログアウト・ページを作成します。ログアウト・ページが `logout.html` であると想定してこの作業を行う場合は、`logout.html` を Oracle HTTP Server の `Apache/Apache/htdocs` ディレクトリにコピーします。
2. ログアウト・ページ `/logout.html` を使用するように SSO ログアウトを構成します。これにより、この URL がログアウト URL として Policy Manager に登録されます。これを行うには、Policy Manager で次のようにナビゲートします。

   「アクセス・システム・コンソール」→「アクセス・システム構成」→「AccessGate 構成」→「WebGate 構成」。

   `LogOutURLs` を `/logout.html` に設定します。
3. ログアウト・ページは WebGate では保護されず、none 認証方式を使用して保護されることを確認してください。
4. Policy Manager が使用する Oracle HTTP Server インスタンスを再起動します。
5. OC4J `jazn.xml` ファイルの `<jazn>` 要素で、プロパティ `custom.sso.url.logout` を次のようにログアウト・ページの URL を指すように設定します。

   ```
   <jazn ... >
      <property name="custom.sso.url.logout" value="/logout.html" />
      ...
   </jazn>
   ```

6. OC4J インスタンスを再起動します。

ログイン・ページとログアウト・ページが同じ Cookie ドメインに含まれていること、つまりログインおよびログアウト中に設定された Cookie が、共有ドメインにマップされることも確認してください。

Oracle Application Server SOA アプリケーションへのログインに関するトラブルシューティング

Oracle Application Server SOA アプリケーションにログインしようとして（たとえば OWSM の場合は `http://www.example.com:7778/ccore/index.html` を使用して）資格証明を入力した後にフォーム・ログイン・ページでログインがハングした場合は、その原因の 1 つとして SOA アプリケーション（OWSM など）を実行するサーバーと Oracle Access Manager を実行するサーバーの間で時刻同期が一致していないことが考えられます。この場合、WebGate はユーザーに対してセッションを作成できません。この問題が発生した場合は、両方のシステムが同期していることを管理者が確認する必要があります。

J2EE アプリケーションでの Oracle Access Manager の例

この項では、次のように Oracle Access Manager を使用した Web アプリケーションおよび EJB について説明します。

- Oracle Access Manager を介して HTTP ヘッダー変数を使用する Web アプリケーション
- Oracle Access Manager ObSSOCookie を使用する Web アプリケーション
- Oracle Access Manager を使用する EJB アプリケーション

関連項目：

- 11-29 ページの「Web サービスに対する Oracle Access Manager のサポートと使用例」

Oracle Access Manager を介して HTTP ヘッダー変数を使用する Web アプリケーション

Web アプリケーションは、オプションで、HTTP ヘッダー変数を使用して認証を行うように構成できます。ユーザー名用のヘッダー変数は、Oracle Access Manager ログイン・モジュール構成の `coreid.name.header` オプションと同じものです。パスワード用のヘッダー変数は、`coreid.password.header` オプションと同じものです。

これらのヘッダー変数を使用するには、次の手順を実行する必要があります。

1. Policy Manager での名前とパスワードの構成
2. Oracle Access Manager ログイン・モジュールに対する HTTP ヘッダー変数の構成
3. HTTP ヘッダーを使用する Web アプリケーションの保護

関連項目：

- 11-8 ページの「HTTP ヘッダー変数を使用した認証」

Policy Manager での名前とパスワードの構成

Policy Manager を使用して、`credential_mapping` および `validate_password` プラグインを有効化します。

関連項目：

- 11-10 ページの「credential_mapping プラグインのフォームベース認証用構成」および 11-10 ページの「validate_password プラグインのフォームベース認証用構成」。
- HTTP ヘッダー変数の使用の詳細は、『Oracle Access Manager System Administration Guide』を参照してください。

Oracle Access Manager ログイン・モジュールに対する HTTP ヘッダー変数の構成

coreid.name.header および（必要に応じて）coreid.password.header のオプション設定を、system-jazn-data.xml の Oracle Access Manager ログイン・モジュール構成に追加します。次の例では、パスワード認証が使用されています。必要な HTTP ヘッダー変数は、myhttpuservar および myhttppwdvar であると想定しています。

```
<options>
   ...
   <option>
      <name>coreid.name.header</name>
      <value>myhttpuservar</value>
   </option>
   <option>
      <name>coreid.password.header</name>
      <value>myhttppwdvar</value>
   </option>
   ...
</options>
```

> **注意：** HTTP ヘッダー変数を使用する場合は、coreid.name.header および coreid.password.header に対する設定に加えて、coreid.name.attribute および coreid.password.attribute のオプション設定も必要になってきます。

HTTP ヘッダーを使用する Web アプリケーションの保護

標準の Web アプリケーション構成において適切なセキュリティ制約を定義し、orion-application.xml 内で auth-method="COREIDSSO" を設定します（11-18 ページの「orion-application.xml における Oracle Access Manager SSO の構成」を参照してください）。

Oracle Access Manager ObSSOCookie を使用する Web アプリケーション

HTTP ヘッダー変数を指定せずに Web アプリケーションをセキュアにするには、Oracle Access Manager ObSSOCookie を使用して、認証情報を取得します。デフォルトでは、この Cookie には、HTTP ヘッダーの Cookie が格納されます。

この Cookie を使用するには、次の手順を実行する必要があります。

1. Oracle Access Manager ログイン・モジュールに対するユーザー名およびパスワードの構成
2. ObSSOCookie を使用する Web アプリケーションの保護

Oracle Access Manager ログイン・モジュールに対するユーザー名およびパスワードの構成

coreid.name.attribute および（必要に応じて）coreid.password.attribute のオプション設定を、system-jazn-data.xml の Oracle Access Manager ログイン・モジュール構成に追加します。次の例では、パスワード認証が使用されています。credential_mapping および validate_password プラグインに対して定義したユーザー名およびパスワード変数が、usernamevar および passwordvar であるものとします。

```
<options>
   ...
   <option>
      <name>coreid.name.attribute</name>
      <value>usernamevar</value>
   </option>
   <option>
      <name>coreid.password.attribute</name>
      <value>passwordvar</value>
   </option>
```

```
   ...
</options>
```

ObSSOCookie を使用する Web アプリケーションの保護

標準の Web アプリケーション構成において適切なセキュリティ制約を定義し、`orion-application.xml` 内で `auth-method="COREIDSSO"` を設定します（11-18 ページの「orion-application.xml における Oracle Access Manager SSO の構成」を参照してください）。

Oracle Access Manager を使用する EJB アプリケーション

EJB 認証の場合、OC4J はユーザー名およびパスワードを EJB コンテキストから取得し、Oracle Access Manager ログイン・モジュールに渡します。同じユーザー名およびパスワードが、Oracle Access Manager に対する認証でも使用されます。

EJB のシナリオでは、この章で前述したように、`credential_mapping` プラグインおよび `validate_password` プラグインの両方が必要です。プラグインで定義したユーザー名およびパスワード変数は、Oracle Access Manager ログイン・モジュールのオプション設定でも使用する必要があります。11-8 ページの「Oracle Access Manager フォームベース認証の構成」を参照してください。

クライアントは、EJB にアクセスする前に、認証を受けるためにユーザー名およびパスワードを送信する必要があります。

Oracle Access Manager ログイン・モジュールを構成します。Oracle Access Manager の認証変数が次のようになっているものとします。

- `myejbappname` は、EJB アプリケーションの名前です。
- `myejbusernamevar` は、`credential_mapping` プラグインで定義した、EJB ユーザー名の変数名です。
- `myejbpwdvar` は、`validate_password` プラグインで定義した、EJB ユーザーのパスワードの変数名です。

```
<application>
   <name>myejbappname</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>myejbusernamevar</value>
            </option>
            <option>
```

```
            <name>coreid.password.attribute</name>
            <value>myejbpwdvar</value>
          </option>
        </options>
      </login-module>
    </login-modules>
</application>
```

注意： 現行リリースでは、Oracle Access Manager `ObSSOCookie` をユーザー名およびパスワードのかわりに送信して認証を行うシナリオは、直接サポートされていません。

関連項目：

- `coreid.resource.type`、`coreid.resource.operation` および `coreid.resource.name` に関する情報は、11-12 ページの「リソース・タイプの構成」を参照してください。

Web サービスに対する Oracle Access Manager のサポートと使用例

Web サービスは、Oracle Access Manager を Web サービス・クライアントの認証に使用できます。OC4J では、Oracle Access Manager に対して、次のように Username トークン認証、X.509 トークン認証および SAML トークン認証がサポートされています。

- Username トークン : OC4J は、ユーザー名およびパスワードを抽出し、Oracle Access Manager に対する認証に使用します。
- X.509 トークン : OC4J は、X.509 エントリの CN 値を使用して、Oracle Access Manager に対する認証を行います。
- SAML トークン : OC4J は、サブジェクト名を使用して、Oracle Access Manager に対する認証を行います。

次の用途について後続の項で説明します。

- Oracle Access Manager に対して Username トークン認証を使用する Web サービス
- Oracle Access Manager に対して X.509 トークン認証を使用する Web サービス
- Oracle Access Manager に対して SAML トークン認証を使用する Web サービス

注意： 現行リリースでは、Oracle Access Manager `ObSSOCookie` をユーザー名およびパスワードのかわりに送信して認証を行うシナリオは、直接サポートされていません。

関連項目：

- 11-26 ページの「J2EE アプリケーションでの Oracle Access Manager の例」。
- Username トークン、X.509 トークンおよび SAML トークン認証の一般情報は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。

Oracle Access Manager に対して Username トークン認証を使用する Web サービス

Username トークン・クライアントは、ユーザー名およびパスワードを認証に使用します。ユーザー名およびパスワードに対する変数は、Oracle Access Manager の `credential_mapping` および `validate_password` プラグインで、対応する Oracle Access Manager ログイン・モジュール構成の `coreid.name.attribute` および `coreid.password.attribute` オプションの設定を使用して構成する必要があります。11-8 ページの「Oracle Access Manager フォームベース認証の構成」を参照してください。

次の設定を想定して、後述のようにログイン・モジュールを構成します。

- `UsernameAppName` は、Username トークン認証を使用する Web サービス・アプリケーションの名前です。
- `UsernameNamevar` は、`credential_mapping` プラグインで定義した、ユーザー名の変数名です。
- `UsernamePwdvar` は、`validate_password` プラグインで定義した、ユーザーのパスワードの変数名です。

```
<application>
   <name>UsernameAppName</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>UsernameNamevar</value>
            </option>
            <option>
               <name>coreid.password.attribute</name>
               <value>UsernamePwdvar</value>
            </option>
         </options>
      </login-module>
   </login-modules>
</application>
```

関連項目：

- `coreid.resource.type`、`coreid.resource.operation` および `coreid.resource.name` に関する情報は、11-12 ページの「リソース・タイプの構成」を参照してください。

Oracle Access Manager に対して X.509 トークン認証を使用する Web サービス

X.509 クライアントは、X.509 エントリの CN 値を使用して認証を行います。CN ユーザー名に対する変数は、Oracle Access Manager の `credential_mapping` プラグインで、対応する Oracle Access Manager ログイン・モジュール構成の `coreid.name.attribute` オプションを使用して構成する必要があります。11-8 ページの「Oracle Access Manager フォームベース認証の構成」を参照してください。

X.509 トークン認証を使用する場合は、Oracle Access Manager `validate_password` プラグインの構成、またはログイン・モジュールの `coreid.password.attribute` オプションの設定は行いません。

次の設定を想定して、後述のようにログイン・モジュールを構成します。

- `X509AppName` は、X509 トークン認証を使用する Web サービス・アプリケーションの名前です。
- `cn_name_var` は、`credential_mapping` プラグインで定義した、CN ユーザー名の変数名です。

```
<application>
   <name>X509AppName</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>cn_name_var</value>
            </option>
         </options>
      </login-module>
   </login-modules>
</application>
```

関連項目：

- `coreid.resource.type`、`coreid.resource.operation` および `coreid.resource.name` に関する情報は、11-12 ページの「リソース・タイプの構成」を参照してください。

Oracle Access Manager に対して SAML トークン認証を使用する Web サービス

SAML クライアントに対しては、OC4J がサブジェクト名を決定します。このため、SAML サブジェクト認証用の変数名を、Oracle Access Manager credential_mapping プラグインで構成しておく必要があります。この credential_mapping 設定は、Oracle Access Manager ログイン・モジュール構成の coreid.name.attribute オプションの設定にも反映される必要があります。11-8 ページの「Oracle Access Manager フォームベース認証の構成」を参照してください。OC4J は、サブジェクト名および credential_mapping 変数名を Oracle Access Manager に渡し、認証を行います。

SAML トークン認証を使用する場合は、Oracle Access Manager validate_password プラグインの構成、またはログイン・モジュールの coreid.password.attribute オプションの設定は行いません。

次の設定を想定して、後述のようにログイン・モジュールを構成します。

- SAMLAppName は、SAML トークン認証を使用する Web サービス・アプリケーションの名前です。
- subject_name_var は、credential_mapping プラグインで定義した、サブジェクト名の変数名です。

SAML のシナリオでは、SAML ログイン・モジュール SAMLLoginModule も使用されます。これは CoreIDLoginModule ログイン・モジュールと一緒に、次の例のように構成する必要があります。この例では、www.example.com を発行者名として使用しています。

> **重要：** system-jazn-data.xml において、SAMLLoginModule 構成が CoreIDLoginModule 構成よりも先行する必要があります。

```
<application>
   <name>SAMLAppName</name>
   <login-modules>

      <login-module>
         <class>
            oracle.security.jazn.login.module.saml.SAMLLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>issuer.name.1</name>
               <value>www.example.com</value>
            </option>
         </options>
      </login-module>

      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
```

```
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>subject_name_var</value>
            </option>
         </options>
      </login-module>

   </login-modules>
</application>
```

関連項目：

- `coreid.resource.type`、`coreid.resource.operation` および `coreid.resource.name` に関する情報は、11-12 ページの「リソース・タイプの構成」を参照してください。
- `SAMLLoginModule` の詳細は、『Oracle Application Server Web Services セキュリティ・ガイド』を参照してください。

Oracle Access Manager の設定のトラブルシューティング

表 11-2 には、Oracle Access Manager の設定および構成時のトラブルシューティングのヒントをいくつか示しています。

表 11-2 Oracle Access Manager のトラブルシューティング

問題	原因 / 解決策
Oracle Access Manager SSO を使用するようにアプリケーションが構成されています。アプリケーションにアクセスしようとすると、Access Server がクラッシュし、再起動します。	Oracle Internet Directory で不適切な検索ベースを構成しているか、またはグループ名が正しく作成されていません。
Oracle Access Manager SSO アプリケーションにアクセスしようとすると、Class Not Found 例外がスローされます。	Oracle Access Manager のファイル `jobaccess.jar` を、Access Manager SDK から OC4J のパスにコピーしたかどうかを確認します。11-15 ページの「各 OC4J インスタンスに対する Access Manager SDK の構成」を参照してください。
Oracle Access Manager SSO アプリケーションにアクセスしようとすると、内部サーバー・エラーが発生します。	OC4J サーバーにインストールされている Access Manager SDK が、適切な Access Server を使用するよう構成されているかどうかを確認します。11-15 ページの「各 OC4J インスタンスに対する Access Manager SDK の構成」を参照してください。OC4J が実行されているかどうかも確認します。
Oracle Access Manager SSO アプリケーションにアクセスしようとしても、ログイン・ページが表示されません。	Policy Manager を使用して、認証方式を正しい設定で有効にしているかどうかを確認します。11-8 ページの「Oracle Access Manager フォームベース認証の構成」を参照してください。

表 11-2 Oracle Access Manager のトラブルシューティング（続き）

問題	原因 / 解決策
Oracle Access Manager SSO アプリケーションにアクセスしようとしても、ログイン・ページが続行されません。	フォームベース認証方式が有効になっているか、ログイン・ページのフォーム変数名（ユーザーおよびパスワード）が Oracle Access Manager のフォームベース認証方式で構成したものと同じであるか、さらに資格証明マッピング・スキームおよびパスワード検証スキームがフォームベース認証方式で構成されているかどうかを確認します。11-8 ページの「Oracle Access Manager フォームベース認証の構成」を参照してください。
Oracle Access Manager を使用するようにアプリケーションを構成したが、常に unauthorized または unauthenticated エラーが表示されます。	`system-jazn-data.xml` 内でこのアプリケーションに対して Oracle Access Manager ログイン・モジュールが正しく構成されているかどうかを確認します。11-19 ページの「Oracle Access Manager ログイン・モジュールの構成」を参照してください。
Oracle Access Manager を使用するようにアプリケーションを構成したが、常に内部サーバー・エラーが表示されます。	Oracle Access Manager Identity Server を使用するように構成した LDAP サーバー（たとえば Oracle Internet Directory）が実行されていてアクセス可能であるかどうかを確認します。
Oracle Access Manager SSO を使用するようにアプリケーションを構成したが、ユーザー名およびパスワードを入力してアクセスしようとすると、アプリケーションがハングします。	フォーム・ページで使用されているアクション URL が、Basic 方式などの、パスワードを使用しない認証方式で保護されているかどうかを確認します。（アクション URL をパスワード保護認証方式で保護すると、実行ループが発生します。）11-9 ページの「ログイン・フォームの作成」を参照してください。

関連項目：

- 11-26 ページの「Oracle Application Server SOA アプリケーションへのログインに関するトラブルシューティング」

12

ユーザーおよびロール API フレームワーク

OC4J 10.1.3.x 実装は、様々なアイデンティティ管理リポジトリにあるユーザーとロールの情報にアクセスするための新しいプラッガブルなアイデンティティ管理 API フレームワークを提供します。これは、第 13 章で説明するプラッガブルなアイデンティティ管理フレームワークとは無関係です。そこで、このフレームワークとの混同を避けるため、この章ではこの API を「ユーザーおよびロール API」と呼びます。

この API には、`com.evermind.security` パッケージの非推奨クラスである `UserManager`、`User` および `Group` にかわる機能が含まれています。

この章の内容は次のとおりです。

ユーザーおよびロール（アイデンティティ管理）API フレームワークの概要

ユーザーおよびロール API フレームワークを使用すると、基礎となるアイデンティティ・リポジトリの種類に関係なく、一貫した移植可能な方法でアプリケーションがアイデンティティ情報（ユーザーおよびロール）にアクセスできます。基礎となるリポジトリは、Oracle Internet Directory、Active Directory（Microsoft 社提供）、Sun Java System Directory Server（Sun 社提供）などの LDAP ディレクトリ・サーバー、データベース、フラット・ファイルまたは他のカスタム・リポジトリのいずれでも可能です。

この API フレームワークは、移植可能な方法でプログラムからリポジトリにアクセスするための便利な手段を提供します。その結果としてアプリケーション開発者は、個々のアイデンティティ・ソースの複雑さを考慮するという困難な作業から解放されます。このフレームワークにより、様々なリポジトリに対してアプリケーションがシームレスに動作するようになります。アプリケーションは、コードを一切変更することなく、各種アイデンティティ・リポジトリ間の切替えができます。

サポートされている操作には、ユーザーとロールの作成、更新、削除、ユーザーとロールの属性の検索、関連情報の検索があります。たとえば、特定のロールに属するすべてのユーザーの電子メール・アドレスを検索できます。これらの API は、認証または認可機能用ではありません。

基本使用モデル（OC4J 統合なし）またはコードを移植可能にする OC4J 統合の使用モデルが使用できます。

UserManager、User、Group にかわるユーザーおよびロール API の機能

ユーザーおよびロール API には、非推奨の `com.evermind.security.UserManager`、`User`、`Group` の各クラスの機能にかわる機能が含まれています。

- ユーザーとロールを作成および取得する `UserManager` 機能は、アイデンティティ・リポジトリ内でユーザーとロールを作成、変更、取得するユーザーおよびロール API の機能によって置き換えられます。
- ユーザー・ロールを取得または変更する `User` 機能は、ユーザー・ロールを取得または変更（ロールへのユーザーの割当てなど）するユーザーおよびロール API の機能によって置き換えられます。（`User` クラスの認証機能は、`DBTableOraDataSourceLoginModule`（9-5 ページの「DBTableOraDataSourceLoginModule」を参照）によって置き換えられます。）
- ロールを取得する `Group` 機能は、ロールを取得するユーザーおよびロール API の機能によって置き換えられます。（ロールにパーミッションを付与する `Group` クラスの機能は、OracleAS JAAS Provider ポリシー管理 API の機能（5-8 ページの「パーミッションの付与または取消しを行う OracleAS JAAS Provider API」を参照）によって置き換えられます。）

ユーザーおよびロール API フレームワークとプロバイダ

ユーザーおよびロール API は、JNDI に似たフレームワークとプロバイダ・モデルに基づいています。

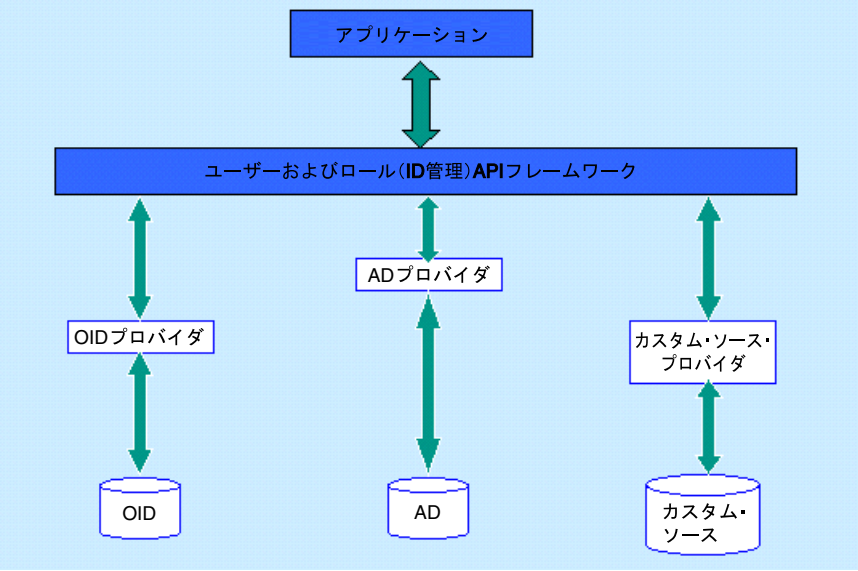
このフレームワークは、アイデンティティ情報にアクセスするための汎用的なメカニズムを規定し、Java インタフェースのみで構成されます。実装の詳細は提供しません。

各プロバイダ（OC4J 付属）は、特定のリポジトリの情報にアクセスする（Active Directory からアイデンティティ情報を読み取る場合など）ための特定の実装クラスにより、フレームワーク・インタフェースを実装します。アイデンティティ・リポジトリのタイプごとに、対応するプロバイダが存在します。

アプリケーション開発者は、汎用フレームワークに基づいてコードを作成します。その後の構成手順で、アプリケーションは目的のアイデンティティ・リポジトリに対する適切なプロバイダにプラグインされます。別のリポジトリを使用するよう、後からアプリケーションを更新することができます。その場合、対応するプロバイダを使用するようにアプリケーションを再構成するだけです。アプリケーション・コードを変更する必要はありません。

OC4J には、Oracle Internet Directory、Active Directory、Sun Java System Directory Server（以前の iPlanet）、Novell eDirectory、OpenLDAP（オープン・ソースの LDAP ディレクトリ）用の各プロバイダが付属しています。次の図 12-1 は、数個の特定のプロバイダに対するフレームワークを表しています。

図 12-1 ユーザーおよびロール API フレームワーク・モデル



ユーザーおよびロールのインタフェースとクラスの概要

この項では、ユーザーおよびロール API パッケージ oracle.security.idm のインタフェースとクラスの概要を説明します。

関連項目：

- 『Oracle Containers for J2EE User and Role Java API Reference』（Javadoc）

ユーザーおよびロール・インタフェースの説明

この項では、oracle.security.idm パッケージのインタフェースの概要を説明します。

アイデンティティ・リポジトリに対するインタフェースは次のとおりです。

- IdentityStore: IdentityStore インスタンスは、アイデンティティ・リポジトリへのハンドルを表します。次の機能に対してメソッドが指定されます。
  - ユーザー・マネージャまたはロール・マネージャの取得
  - ユーザー、ロールまたはそのプロファイルの検索
  - アイデンティティ・リポジトリの検索フィルタまたは検索可能な属性のリストの取得
- IdentityStoreFactory: IdentityStoreFactory インスタンスは、基礎となるアイデンティティ・リポジトリを表します。また、IdentityStore インスタンスを取得するメソッドを含んでおり、このインスタンスの生成に必要なプロバイダ固有のプロパティから構成されるハッシュテーブルを入力として取ります。

アイデンティティ・リポジトリ内のユーザー・エントリに対するインタフェースは次のとおりです。

- User: User インスタンスは、アイデンティティ・ストア内のユーザーを表します。これは Identity のサブインタフェースであり、ユーザー・プロファイルを取得するメソッドを指定します。このインタフェースは、UserProfile のスーパーインタフェースでもあります。
- UserProfile: UserProfile インスタンスは、ユーザーに関する詳細情報を表し、アクセス頻度の高いプロパティ（名前、肩書き、従業員番号、上司、住所、電子メール・アドレス、電話番号、FAX 番号、携帯電話番号など）の定数を格納します。これは、User のサブインタフェースです。このインタフェースは、これらの一般的なプロパティを取得および設定するメソッドと、さらに一般的なメソッドである getProperty()、getProperties()、setProperty()、setProperties() を指定します。setProperty() メソッドと setProperties() メソッドは、oracle.security.idm.ModProperty クラス（後述）のインスタンスを取ります。
- UserManager: UserManager インスタンスは、ユーザーに関係する操作の実行など、リポジトリ内のユーザー・コミュニティの管理に使用されます。これには、ユーザーの作成、認証、削除などがあります。

アイデンティティ・リポジトリ内のロールに対するインタフェースは次のとおりです。

- Role: Role インスタンスは、アイデンティティ・ストア内のロールを表します。これは Identity のサブインタフェースであり、ロール・プロファイルを取得するメソッドを指定します。このインタフェースは、RoleProfile のスーパーインタフェースでもあります。
- RoleProfile: RoleProfile インスタンスは、ロールに関する詳細情報を表します。これは Role のサブインタフェースであり、ロールの所有者を追加、削除、取得するメソッド、ロールが直接的または間接的に付与されたすべての権限受領者を取得するメソッド、ロールがアプリケーション・ロールかエンタープライズ・ロールかを判別するメソッドを指定します。
- RoleManager: RoleManager インスタンスは、ロールに関係する操作の実行など、リポジトリ内のロール・コミュニティの管理に使用されます。これには、ロールの作成、ロールの削除、指定されたプリンシパルに対するロールの付与と取消しなどがあります。

ユーザーおよびロール・クラスの説明

この項では、oracle.security.idm パッケージ内の主要なクラスの概要を説明します。

- IdentityStoreFactoryBuilder: このクラスのインスタンスは、アイデンティティ・ストア・ファクトリを構築する際に使用します。このクラスには、IdentityStoreFactory インスタンスを取得するための、オーバーロードされた getIdentityStoreFactory() メソッドが含まれます。

ユーザーおよびロール API の使用モデル

この項では、基本的な API フレームワークと OC4J 統合機能を使用するための手順を説明し、そのサンプルを示します。この項の内容は次のとおりです。

手順説明 : 基本の使用モデル

この項では、基本的な API フレームワークを使用する手順を説明します。手順 1 と 2 は主に構成に関連し、使用されるプロバイダとその構成を判別します。この 2 つの手順で示すコードは、アイデンティティ・リポジトリとプロバイダによって異なります。手順 3 でリポジトリに対して実行される操作は、汎用的な操作です。別のリポジトリに変更しても、このコードには影響しません。

関連項目 :

- 完全なサンプル・コードは、12-9 ページの「例 : 基本的なユーザーおよびロール API フレームワーク」を参照してください。
- コードを移植可能にする機能は、12-7 ページの「手順説明 : OC4J 統合の使用モデル」を参照してください。

1. IdentityStoreFactory インスタンスを取得します。このファクトリ・インスタンスはアイデンティティ・リポジトリを表し、IdentityStoreFactoryBuilder インスタンスで getIdentityStoreFactory() コールを使用して作成されます。このコールは、特定のアイデンティティ・リポジトリに接続する際に使用されるプロバイダの名前を受け取ります。また、このプロバイダが必要とする構成情報も指定します。

   たとえば、アプリケーションで Oracle Internet Directory に接続する必要があるとします。Oracle Internet Directory のプロバイダ名は次のとおりです。

   ```
   oracle.security.idm.providers.oid.OIDIdentityStoreFactory
   ```

   Oracle Internet Directory などの LDAP プロバイダには、LDAP の URL、セキュリティ・プリンシパル、資格証明などの構成情報が必要です。次に例を示します。

   ```
   IdentityStoreFactoryBuilder builder =
                              new IdentityStoreFactoryBuilder();
   IdentityStoreFactory oidFactory = null;

   Hashtable factEnv = new Hashtable();

   // creating the factory instance
   // set the configuration information
   factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_PRINCIPAL, "cn=orcladmin");
   factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_CREDENTIALS, "welcome1");
   factEnv.put(OIDIdentityStoreFactory.ST_LDAP_URL,
            "ldap://ilinabc10.us.oracle.com:3060/");
   factEnv.put(OIDIdentityStoreFactory.ST_LOGGING,"false");
   ```

```
factEnv.put(OIDIdentityStoreFactory.ST_LOG_LEVEL,
        java.util.logging.Level.ALL);
oidFactory =  builder.getIdentityStoreFactory(
        "oracle.security.idm.providers.oid.OIDIdentityStoreFactory", factEnv);
```

2. IdentityStore インスタンスを取得し、リポジトリで操作を実行します。これを取得するには、IdentityStoreFactory インスタンスで getIdentityStoreInstance() コールを使用します。このコールは、アイデンティティ・ストア・インスタンスの作成に必要な構成情報を受け取ることができます。たとえば Oracle Internet Directory の場合、次の例で示すように、操作が実行されるサブスクライバまたはレルムの名前を指定する必要があります。

```
Hashtable storeEnv = new Hashtable();

// creating the store instance
storeEnv.put(OIDIdentityStoreFactory.ST_SUBSCRIBER_NAME,
                              "dc=us,dc=oracle,dc=com");
oidStore = oidFactory.getIdentityStoreInstance(storeEnv);
```

3. IdentityStore インスタンスを使用して、エントリの検索、更新、作成、削除などの操作をアイデンティティ・リポジトリで実行します。たとえば次のコードでは、名前が john で始まるすべてのユーザーが検索されます。

```
// search filter for users whose name begins with "john"
SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
        UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);

// Add the wildcard character
sf.setValue("john"+sf.getWildCardChar());

// generate the search parameter instance and set the search filter
SearchParameters params = new SearchParameters();
params.setFilter(sf);

// Searching for users
// search on the IdentityStore instance

SearchResponse resp = oidStore.searchUsers(params);
System.out.println("Searched users are:");

// Iterate on the search results
while (resp.hasNext()
{
  User usr = (User) resp.next();
  System.out.println("Name: "+usr.getName());
}
```

手順説明 : OC4J 統合の使用モデル

前項の「手順説明 : 基本の使用モデル」で説明したように、構成に関連するコードはアイデンティティ・リポジトリとそれに関連するプロバイダによって制限されるため、アプリケーションで使用するアイデンティティ・リポジトリを変更する場合は、コードも必ず変更する必要があります。

コードを移植可能にするには、API フレームワークの OC4J 統合機能を使用します。この機能は、アプリケーションの OC4J ログイン・モジュールにあるセキュリティ・プロバイダ情報を使用するため、アプリケーションでリポジトリやプロバイダの構成を指定する必要がなくなります。その結果、アプリケーション・コードは汎用的になり、ログイン・モジュール構成でセキュリティ・プロバイダ情報を変更するだけで、アプリケーションで使用するアイデンティティ・ソースを変更できるようになります。

重要 : OC4J 統合機能はセキュリティに関連する操作であるため、アプリケーションに必要なパーミッションが設定されている必要があります。`java2.policy` の要件は、次項の「OC4J 統合機能に対するパーミッション要件」を参照してください。

関連項目 :

- 完全なサンプル・コードは、12-10 ページの「例 : OC4J 統合を使用するユーザーおよびロール API フレームワーク」を参照してください。

OC4J 統合機能は、次の手順で使用します。

1. ユーザーおよびロール・フレームワークで、ユーザー / ロール・プロパティ・ファイル（一般的な名前は `userrole.properties`）へのパスを、次のように Java システム・プロパティとして指定する必要があります。

```
System.setProperty("oracle.userrole.properties",
                   "/home/jdoe/userrole.properties");
```

   プロパティ・ファイルには、フレームワークで OC4J ログイン・モジュール情報へアクセスする際に必要な設定が格納されています。プロパティ・ファイルの形式は、12-8 ページの「ユーザーおよびロール・プロパティ・ファイル」で示しています。

2. `IdentityStoreFactory` インスタンスの作成は権限を必要とする操作であり、次のように `AccessController.doPrivileged()` ブロック内で実行する必要があります。

```
IdentityStoreFactory factory = null;

try
{
  factory = (IdentityStoreFactory) AccessController.doPrivileged(
                new PrivilegedExceptionAction()
                {
                  public Object run() throws IMException
                  {
                    IdentityStoreFactoryBuilder builder =
                                new IdentityStoreFactoryBuilder();
                    return builder.getIdentityStoreFactory();
                  }
                });

}catch (PrivilegedActionException e)
{
  e.getException().printStackTrace(out);
}
catch (Exception e)
{
  e.printStackTrace(out);
}
```

3. 次のように、IdentityStore インスタンスは簡単に取得できます。

```
IdentityStore store = factory.getIdentityStoreInstance();
```

4. 前項の「手順説明 : 基本の使用モデル」で示したように、アイデンティティ・ストアに対して操作を実行します。

OC4J 統合機能に対するパーミッション要件

OC4J 統合機能はセキュリティに関連する操作であるため、アプリケーション・コードに一定のパーミッションが必要となります。具体的には、IdentityStoreFactoryBuilder クラスの getIdentityStoreFactory() メソッドで、一定のパーミッションを必要とする API コールが行われます。

OC4J の java2.policy ファイルで、次のパーミッションをアプリケーション・コードベースに付与します。

```
grant codebase "file:${oracle.home}/application_code_base"
{
    permission oracle.security.jazn.JAZNPermission "*";
};
```

Java 2 ポリシーを使用するには、セキュリティ・マネージャを有効にする必要があることに注意してください（5-2 ページの「Java 2 セキュリティ・マネージャおよびポリシー・ファイルの指定」を参照）。

ユーザーおよびロール・プロパティ・ファイル

OC4J 統合機能を使用するには、Java システム・プロパティとして表されているユーザー / ロール・プロパティ・ファイルへのパスを、API フレームワークが認識する必要があります。プロパティ・ファイルには、必要な OC4J ログイン・モジュール情報へアクセスする際にフレームワークが必要とするプロパティが格納されています。このファイルには任意の名前を指定できますが、ファイル形式は次のようにする必要があります。

```
# This line should not be changed.
configurationsourceclass=oracle.security.idm.util.OC4JConfigurationSource

# This property specifies the JMX Mbean URL for the OC4J container in which the
# application is deployed.
# For OPMN-managed OC4J, uncomment the URL that follows; comment out all others.
# format: service:jmx:rmi:///opmn://opmnhost[:opmnport]/oc4jInstance

jmxserviceurl=service:jmx:rmi:///opmn://localhost:6008/home

# For standalone OC4J, uncomment the URL that follows; comment out all others.
# format: service:jmx:rmi:///opmn://oc4jhost:rmiport/oc4jContextRoot
#jmxserviceurl=service:jmx:rmi://localhost:23791/oc4j/
```

関連項目：

- OPMN 管理インスタンスまたはスタンドアロンの OC4J インスタンスに対する JMX サービス URI の設定の詳細は、『Oracle Containers for J2EE 開発者ガイド』を参照してください。

例 : 基本的なユーザーおよびロール API フレームワーク

この項では、12-5 ページの「手順説明 : 基本の使用モデル」で説明した手順に対応する完全な例を示します。

```
import oracle.security.idm.*;
import oracle.security.idm.providers.oid.*;
import java.util.*;
import java.io.*;

public class BasicSampleOID
{
  public static void main(String args[])
  {
    IdentityStoreFactoryBuilder builder = new IdentityStoreFactoryBuilder();
    IdentityStoreFactory oidFactory = null;
    IdentityStore oidStore = null;

    try
    {

      Hashtable factEnv = new Hashtable();
      Hashtable storeEnv = new Hashtable();

      // creating the factory instance
      factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_PRINCIPAL,
                  "cn=user,....");
      factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_CREDENTIALS,
                   "password");
      factEnv.put(OIDIdentityStoreFactory.ST_LDAP_URL,
                  "ldap://johnmc.us.oracle.com:3060/");
      factEnv.put(OIDIdentityStoreFactory.ST_LOGGING,"false");
      factEnv.put(OIDIdentityStoreFactory.ST_LOG_LEVEL,
                  java.util.logging.Level.ALL);

      oidFactory =  builder.getIdentityStoreFactory(
                  "oracle.security.idm.providers.oid.OIDIdentityStoreFactory",
                  factEnv);

      // creating the store instance
      storeEnv.put(OIDIdentityStoreFactory.ST_SUBSCRIBER_NAME,
                   "dc=us,dc=oracle,dc=com");
      oidStore = oidFactory.getIdentityStoreInstance(storeEnv);

      // search filter (cn=a*)
      SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
                    UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
      sf.setValue("john"+sf.getWildCardChar());

      SearchParameters params = new SearchParameters();
      params.setFilter(sf);

      // Searching for users
      SearchResponse resp = oidStore.searchUsers(params);
      System.out.println("Searched users are:");
      while (resp.hasNext())
      {
        User usr = (User) resp.next();
        System.out.println("Name: "+usr.getName());
      }

    }catch (IMException e)
    {
```

```
            e.printStackTrace();
          }
        }
      }
```

例 : OC4J 統合を使用するユーザーおよびロール API フレームワーク

この項では、12-7 ページの「手順説明 : OC4J 統合の使用モデル」で説明した手順に対応する完全な例を示します。

```
import java.io.*;
import java.util.*;
import java.security.AccessController;
import java.security.PrivilegedExceptionAction;
import java.security.PrivilegedActionException;

// Packages for Servlets
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.security.idm.*;

public class UserSearch extends HttpServlet
{
    private String USERROLEPROPFILE = "UserRolePropFile";
    IdentityStore store = null;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = "";
        String searchType = "usersearch";
        try
        {
            name = request.getParameter("name");
            searchType = request.getParameter("searchtype");
        } catch (Exception e)
        {
            e.printStackTrace();
        }

        String filter = (name != null)? name: "";

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>UserSearch</title></head>");
        out.println("<body>");
        out.println(
            "<label for=\"fld\"><b>User name begining with</b></label>");
        out.println(
          "<form action=\"usersearch\">"+
          "<P><select name=\"searchtype\" size=\"1\">"+
          "<option value=\"usersearch\">Search Users</option>"+
          "<option value=\"rolesearch\">Search Roles</option>"+
          "<option value=\"membershipsearch\">Search Membership details"+
          "<option value=\"membersearch\">Searchs Members of role"+
          "</option></select></P>"+
          "<input name=\"name\" id=\"fld\" value=\""
```

```
      + filter +
      "\"type=\"text\"/><input type=\"SUBMIT\" value=\"Search\"/></form>");
    out.println("<br><br><br>");

    // Create the IdentityStore instance required for searching
    configureAPI(out);

    // Carries out the actual search using IdentityStore instance obtained
    // above
    doSearch(out, searchType, name);

    out.println("</body></html>");
    out.close();
}

public void configureAPI(PrintWriter out)
{
  IdentityStoreFactoryBuilder builder = new IdentityStoreFactoryBuilder();

  // Set the following system property to specify the location of
  // "userrole.properties" file. This file is used by user-role apis for
  // reading the configuration from OC4J

  // Get the file location from the servlet init parameters
  System.setProperty("oracle.userrole.properties",
            getServletConfig().getInitParameter(USERROLEPROPFILE));

  IdentityStoreFactory factory = null;

  try
  {
    factory = (IdentityStoreFactory) AccessController.doPrivileged(
             new PrivilegedExceptionAction()
             {
               public Object run() throws IMException
               {
                 IdentityStoreFactoryBuilder builder =
                             new IdentityStoreFactoryBuilder();
                 return builder.getIdentityStoreFactory();
               }
             });
    store = factory.getIdentityStoreInstance();
  }catch (PrivilegedActionException e)
  {
    e.getException().printStackTrace(out);
  }
  catch (Exception e)
  {
    e.printStackTrace(out);
  }
}

public void doSearch(PrintWriter out, String searchType, String name)
{
    System.out.println("Inside doSearch");

    if (name == null) return;
    if (searchType.equals("usersearch"))
      searchUsers(out, name);
    else if (searchType.equals("rolesearch"))
      searchRoles(out, name);
    else if (searchType.equals("membershipsearch"))
      searchMembership(out, name);
```

```
    else if (searchType.equals("membersearch"))
      searchMembers(out, name);
}

public void searchMembers(PrintWriter out, String name)
{
    System.out.println("Inside searchMembers");
    if (name == null) return;
    out.println("Results: <br>");

    try {
        Role rle = store.searchRole(IdentityStore.SEARCH_BY_NAME, name);
        out.println("Members of role \""+rle.getName()+
                    "\" are:<br>");
        SearchResponse resp =
         rle.getRoleProfile().getGrantees(null, false);
        while (resp.hasNext()) {
            Identity idy = resp.next();
            out.println("Unique name: " + idy.getUniqueName() + "<br>");
        }

    } catch (IMException e) {
        e.printStackTrace(out);
    }
}

public void searchMembership(PrintWriter out, String name)
{
    System.out.println("Inside searchMembership");
    if (name == null) return;
    out.println("Results: <br>");

    try {
        User usr = store.searchUser(name);
        out.println("Membership details for user \""+usr.getName()+
                    "\" are:<br>");
        SearchResponse resp =
         store.getRoleManager().getGrantedRoles(usr.getPrincipal(), false);
        while (resp.hasNext()) {
            Identity idy = resp.next();
            out.println("Unique name: " + idy.getUniqueName() + "<br>");
        }

    } catch (IMException e) {
        e.printStackTrace(out);
    }
}

public void searchRoles(PrintWriter out, String name)
{
    System.out.println("Inside searchRoles");
    if (name == null) return;
    out.println("Results: <br>");

    try {
        SimpleSearchFilter sf = store.getSimpleSearchFilter(
                RoleProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
        sf.setValue(name + sf.getWildCardChar());

        SearchParameters params = new SearchParameters();
        params.setFilter(sf);

        // Searching for users
```

```
            SearchResponse resp = store.searchRoles(0, params);
            out.println("Searched roles are:<br>");
            while (resp.hasNext()) {
                Identity idy = resp.next();
                out.println("Unique name: " + idy.getUniqueName() + "<br>");
            }

        } catch (IMException e) {
            e.printStackTrace(out);
        }
    }

    public void searchUsers(PrintWriter out, String name)
    {
        System.out.println("Inside searchUsers");
        if (name == null) return;
        out.println("Results: <br>");

        try {
            SimpleSearchFilter sf = store.getSimpleSearchFilter(
                    UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
            sf.setValue(name + sf.getWildCardChar());

            SearchParameters params = new SearchParameters();
            params.setFilter(sf);

            // Searching for users
            SearchResponse resp = store.searchUsers(params);
            out.println("Searched users are:<br>");
            while (resp.hasNext()) {
                Identity idy = resp.next();
                out.println("Unique name: " + idy.getUniqueName() + "<br>");
            }

        } catch (IMException e) {
            e.printStackTrace(out);
        }
    }
}
```

13

交換可能なアイデンティティ管理フレームワーク

OC4J には、このマニュアルですでに説明したセキュリティ・プロバイダの他に、Web ベース・アプリケーションによる異機種サード・パーティのアイデンティティ管理システムの使用を汎用的にサポートするフレームワークもあります。

この章ではこのアイデンティティ管理フレームワークについて説明します。この章の内容は次のとおりです。

OracleAS JAAS Provider アイデンティティ管理フレームワークの概要

ここでは、Web アプリケーションで使用する OC4J アイデンティティ管理フレームワークの概要を説明します。この項の内容は次のとおりです。

注意：

- 規約上、`<jazn>` 設定の `provider="XML"` は、交換可能なアイデンティティ管理フレームワークで使用されます。
- ファイルベース・プロバイダまたは Oracle Internet Directory 以外のアイデンティティ・リポジトリを使用する場合、管理ユーザー・アカウントおよび管理者ロールを定義し、ロールをユーザーに付与し、必要なパーミッションをロールに付与する必要があります。10-10 ページの「管理ユーザーとロールの作成および RMI パーミッションの付与」を参照してください。

交換可能なアイデンティティ管理フレームワークの必要事項

前述のように、Oracle Application Server には、Oracle Identity Management、Oracle Access Manager など、セキュリティ・インフラストラクチャが用意されています。両方ともアイデンティティ・リポジトリが含まれています。すでに説明したように、特定の外部 LDAP プロバイダ（Active Directory と Sun Java System Directory Server）もサポートされます。

ただし、以前のリリースでは、他のサード・パーティのアイデンティティ管理システムとセキュリティ・システムをサポートする汎用的なフレームワークはありませんでした。OC4J 10.1.3.1 実装には、そのようなフレームワークが追加されました。異機種サード・パーティ・システムを OC4J に統合できるようになったため、J2EE アプリケーションがこれらのサード・パーティ・システムと相互運用できるようになりました。

アイデンティティ管理フレームワークの仕組み

ここでは、アイデンティティ管理フレームワークのコンポーネントを紹介し、コンポーネントが連携する仕組みの概要を説明します。（サード・パーティのアイデンティティ管理システムと OC4J との統合は、標準の JAAS ログイン・モジュールに基づきます。）

アイデンティティ管理フレームワークの一般的なモデルでは、コンポーネントが次の図 13-1 に示すように連携します。

1. トークン・コレクタが適切なアクションを実行して、HTTP リクエストからユーザー資格証明を収集します。トークン・コレクタのインタフェースでは、資格証明を収集するメカニズムをプラグインできます。たとえば、フォームベース認証の場合、トークン・コレクタはユーザー名とパスワード入力用のログイン・ページにリダイレクトします。トークン・コレクタでは、Basic、フォームベースまたはカスタムの各認証方式を使用して、ユーザー名とパスワードまたは HTTP Cookie など、各種の資格証明を収集できます。HTTP リクエスト・オブジェクトの一部として受信されるかぎり、X.509 トークンまたは SAML トークンも使用できます。

2. トークン・コレクタは、指定されたトークン・タイプに応じて、ユーザー資格証明に対応する適切なアイデンティティ・トークンを作成し、そのトークンを OC4J に渡します。通常、アイデンティティ管理フレームワークでは、トークンのタイプはユーザー・アイデンティティを含む HTTP Cookie または HTTP ヘッダーです。Cookie またはヘッダーを使用しない場合は、HTTP リクエスト・オブジェクト自体を使用できます。この場合、アイデンティティを取得するリクエスト・オブジェクトの解析方法を実装に設定します。（たとえば、クエリー・パラメータを解析および解釈してアイデンティティを取得するなどです。）

3. トークン・アサータは、OC4J からアイデンティティ・トークンを受信し、アイデンティティを検証します。このとき、アイデンティティをサード・パーティのアイデンティティ管理システムと照合して認証するか、他の方法でトークンを検証します。トークン・アサータのインタフェースを使用すると、アイデンティティ・トークンのアサーション・メカニズムをプラグインできます。（たとえば、OC4J Java SSO でアイデンティティ管理フレームワークを使用する場合、アイデンティティ・トークンは Cookie にエンコードされ、Cookie 内の情報の検証に対称鍵が使用されます。）

   アイデンティティ管理フレームワークにおいては、アサーションとは、アイデンティティ・トークンの解釈機能、トークン内容の検証機能、およびトークンに対応するアイデンティティの設定機能のことです。パスワードや他の資格証明は必ずしも必要ありません。通常、トークンは信頼されたソースから取得されるか、鍵とともに送信されているためです。

   トークン・アサータがシングル・サインオン・システムによって認証されたアイデンティティを受け付けることができるため、シングル・サインオン・システムはカスタム・セキュリティ・プロバイダから資格証明を収集し、アイデンティティを OC4J に渡すことができます。次に、トークン・アサータはアイデンティティを検証し、OC4J コンテナ内にアイデンティティを設定します。

4. アイデンティティが検証されると、トークン・アサータは構成したアイデンティティ・コールバック・ハンドラを介してユーザー情報を OC4J に返します。この時点で、コールバック・ハンドラに渡されたアイデンティティは信頼されたアイデンティティです。コールバック・ハンドラは、アイデンティティ文字列、またはアイデンティティ文字列とリクエスト・オブジェクトなどで渡すだけで構成できます。

5. OC4J は、ユーザー情報とともにアイデンティティ・コールバック・ハンドラをアプリケーション用に構成されたログイン・モジュールに渡します。ログイン・モジュールは、適切なコールバック・タイプを構成してユーザー情報を処理し、コールバック・ハンドラ `handle(Callback[])` メソッド（標準的なログイン・モジュール機能）を介してアイデンティティ・コールバック・ハンドラに渡します。ログイン・モジュールは、必要に応じてサード・パーティのアイデンティティ管理システムから情報を取得して、ユーザーが属するロールなど、ユーザーのプリンシパルを収集します。次に、ログイン・モジュールは移入されたサブジェクトを OC4J に送信します。

   アイデンティティはすでにトークン・アサータによって検証されているため、ログイン・モジュールで認証を再度実行する必要はありません。また、アイデンティティ管理フレームワークのカスタム・ログイン・モジュールも、アイデンティティ・コールバック・ハンドラを処理できる必要があります。

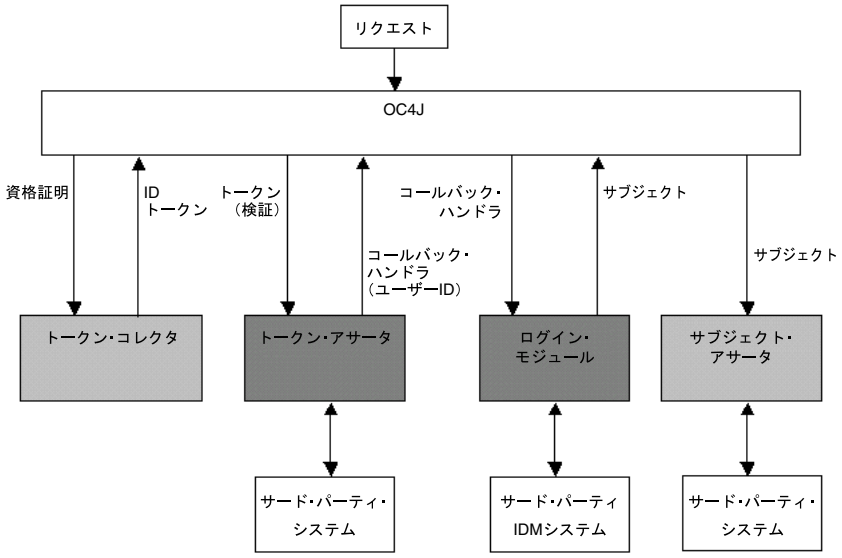
   代替方法 : アプリケーションのログイン・モジュールを実装して構成するかわりに、サブジェクトを作成し、`getSubject()` メソッドを実装するトークン・アサータを指定して、ログイン・モジュールを使用せずに、（ユーザー・ロールを移入された）サブジェクトを直接 OC4J に返すことができます。この方法を選択する場合、アイデンティティ管理フレームワークのプロパティ（`idm.subject.loginmodule.disabled`）を適切に設定する必要があります。

6. オプションでは、サブジェクト・アサータがログイン・モジュールによって伝播されたサブジェクトを検証できます。たとえば、サブジェクト・プリンシパルに署名と検証を行い、プリンシパルの認証性を保護することができます。

これらのコンポーネントには Java インタフェースが提供されており、必要に応じて実装してサード・パーティのアイデンティティ管理システムと OC4J を統合することができます。

管理者は、使用する実装クラスを参照するように OC4J を構成し、ログイン・モジュールを構成する必要があります。

図 13-1 アイデンティティ管理フレームワークのデータ・フロー



注意： トークン・アサータとログイン・モジュールは、異なるバックエンド・アイデンティティ・システムにアクセスできます。たとえば、トークン・アサータは、アイデンティティを検証するためにアイデンティティ管理システムにアクセスします。一方、ログイン・モジュールは、ユーザー・グループやロールなど、ユーザーに関するアプリケーション固有の追加情報を格納した、データベースなど個別のアイデンティティ・ストアにアクセスできます。この追加情報は、ユーザー用に移入されるサブジェクト内に含めることができます。（アイデンティティ・ストアとは、ディレクトリやデータベースなど、ユーザー情報を格納するリポジトリのことです。）

関連項目：

- 2-14 ページの「JAAS 認証 : ログイン・モジュール」
- 13-6 ページの「アイデンティティ管理フレームワークのプログラム・インタフェース」
- 13-13 ページの「アイデンティティ管理フレームワークの構成」

プログラムによるアイデンティティ管理フレームワーク実装の概要

サード・パーティのアイデンティティ管理システムを Oracle フレームワークで使用するには、次のプログラムによる手順を使用します。

1. トークン・コレクタのインタフェースを実装するトークン・コレクタ・クラスを指定します。1 つのオプションとして、Oracle 実装を拡張することがあります。
2. HTTP Cookie アイデンティティ・トークン、HTTP ヘッダー・アイデンティティ・トークンまたは HTTP リクエスト・アイデンティティ・トークンに対して、Oracle が提供する適切なアイデンティティ・トークン・クラスを選択します。
3. トークン・アサータのインタフェースを実装するトークン・アサータ・クラスを指定します。
4. アイデンティティ・コールバック・ハンドラのインタフェースを実装するアイデンティティ・コールバック・ハンドラ・クラスを指定するか、Oracle 実装を使用します。
5. ログイン・モジュールを実装します（トークン・アサータによってサブジェクトに移入し、`getSubject()` メソッドを実装する代替方法を使用しない場合）。ログイン・モジュールは、アイデンティティ・コールバック・ハンドラ・インスタンスを適切に処理できる必要があります。
6. オプションでは、サブジェクト・アサータのインタフェースを実装するサブジェクト・アサータ・クラスを指定します。

関連項目：

- 13-6 ページの「アイデンティティ管理フレームワークのプログラム・インタフェース」

アイデンティティ管理フレームワーク構成の概要

Oracle アイデンティティ管理フレームワークには次の構成が必要です。詳細は、この章で後述します。

1. 管理者は `jazn.xml` を構成して、トークン・コレクタ実装クラス、トークン・アサータ実装クラス、トークン・タイプ（HTTP ヘッダーや Cookie など）などを示す、アイデンティティ管理プロパティを設定します。各プロパティは、`<jazn>` 要素の `<property>` サブ要素に設定されます。
2. 管理者は適切なログイン・モジュールを構成します（オプションで、デフォルト・ログイン・モジュール `RealmLoginModule` を使用します）。構成は、`system-jazn-data.xml` の `<jazn-loginconfig>` 要素の下に格納されます。
3. フレームワークを使用するアプリケーションに対して、アプリケーション・アセンブラがアプリケーションとともにパッケージ化されている `orion-application.xml` ファイル内に認証方式 `CUSTOM_AUTH` を指定します。

関連項目：

- 13-13 ページの「アイデンティティ管理フレームワークの構成」

OC4J Java シングル・サインオンによるアイデンティティ管理フレームワークの使用

OC4J 10.1.3.1 実装には、アイデンティティ管理フレームワークを使用する、代替となる Java シングル・サインオン実装（Java SSO）がパッケージ化されています。Java SSO（詳細は、第 14 章「OC4J Java シングル・サインオン」を参照）は、使用するアイデンティティ管理システムと OC4J の結合を解除する SSO 実装です。Oracle Identity Management（Oracle Single Sign-On に必要）や Oracle Access Manager（Oracle Access Manager SSO に必要）などのような、特定のインフラストラクチャ要件はありません。

Java SSO には、Cookie 資格証明を収集し、アイデンティティ・トークンを OC4J に返すトークン・コレクタ実装、およびトークン内のアイデンティティを検証し、アイデンティティをコールバック・ハンドラで OC4J に戻すトークン・アサータ実装が含まれています。

アイデンティティ管理フレームワークのプログラム・インタフェース

ここでは、サード・パーティのアイデンティティ管理システムと OC4J を統合するためのインタフェース、API および Oracle 実装について説明します。

通常、ここで説明するメソッドは、OC4J のアイデンティティ管理フレームワークによってコールされます。

- アイデンティティ・トークン・インタフェースおよび Oracle 実装
- トークン・コレクタ・インタフェースおよび Oracle 実装
- トークン・アサータ・インタフェース
- アイデンティティ・コールバック・ハンドラ・インタフェース
- サブジェクト・アサータ・インタフェース
- アイデンティティ管理フレームワーク実装クラスのパッケージ化

注意：

- アイデンティティ管理フレームワークは、使用または指定することを選択した実装クラスの構成によって駆動されます。
- 自分で指定するクラスの実装は、単一の JAR ファイルにパッケージ化して OC4J 共有ライブラリとしてデプロイすることをお薦めします。(6-15 ページの「ライブラリを共有するためのタスク」を参照してください。)

関連項目：

- アイデンティティ管理フレームワーク API を含む Javadoc は、『Oracle Containers for J2EE Security Java API Reference』を参照してください。

アイデンティティ・トークン・インタフェースおよび Oracle 実装

Oracle は、次のアイデンティティ・トークン・インタフェースを定義しています。

`oracle.security.jazn.token.IdentityToken`

アイデンティティ・トークン・オブジェクトにユーザー資格証明が格納されています。トークン・オブジェクトは、トークン・コレクタによって返され、トークン・アサータに渡されます。

`IdentityToken` インタフェースには、次のメソッドが指定されています。

- `void setTokenType(String tokenType)`

  このメソッドが指定するトークン・タイプは次のとおりです。`HTTP_COOKIE`、`HTTP_HEADER` または `HTTP_REQUEST`。

- `String getTokenType()`

  このメソッドは、トークン・タイプを返します。

OC4J には、次の `IdentityToken` 実装が用意されています。

- `oracle.security.jazn.token.HttpCookieIdentityToken`

  このクラスのインスタンスをトークン・コレクタによって構成するには、Cookie の名前を Cookie 値へのキーとして格納する `Map` インスタンスを指定します。

  `HttpCookieIdentityToken(java.util.Map Cookies)`

  アイデンティティ情報を取得する次のメソッドが含まれています。

  `Map getCookies()`

- `oracle.security.jazn.token.HttpHeaderIdentityToken`

  このクラスのインスタンスをトークン・コレクタによって構成するには、ヘッダーの名前をヘッダー値へのキーとして格納する `Map` インスタンスを指定します。

  ```
  HttpHeaderIdentityToken(java.util.Map headerValues)
  ```

  アイデンティティ情報を取得する次のメソッドが含まれています。

  ```
  Map getHeaderValues()
  ```

- `oracle.security.jazn.token.HttpRequestIdentityToken`

  このクラスには、次のコンストラクタが含まれています。

  ```
  HttpRequestIdentityToken(HttpServletRequest request)
  ```

  アイデンティティ情報を取得する次のメソッドが含まれています。

  ```
  HttpServletRequest getRequest()
  ```

トークン・コレクタ・インタフェースおよび Oracle 実装

Oracle は、次のトークン・コレクタ・インタフェースを提供しています。

`oracle.security.jazn.collector.TokenCollector`

このインタフェースの実装を使用して、HTTP ベース認証資格証明を受信し、アイデンティティ・トークンを作成します。実装クラスには、次のような機能があります。

1. ユーザー・リクエストから資格証明を収集します。
2. ユーザーのアイデンティティ・トークンを作成します。
3. トークンを OC4J に渡します。

トークン・コレクタは次の入力を取得します。

- 目的のトークン・タイプ（トークン・コレクタは作成するタイプを認識しています）
- HTTP リクエスト・オブジェクト
- Cookie またはヘッダーの名前のリスト（トークン・タイプが Cookie またはヘッダーの場合に必要に応じて）
- 任意のユーザー定義プロパティ（必要に応じて）

ユーザーが指定したトークンが有効でないか、トークンがないか、その他の想定されるシナリオでユーザーを適切にアサートできない場合、OC4J はトークン・コレクタ `fail()` メソッドをコールし、失敗の理由を渡します。こうすると、トークン・コレクタは、必要に応じたアクションを実行できます。`fail()` メソッドは、適切な失敗のアクションを実行するように実装されている必要があります（ユーザーをログイン・ページにリダイレクトして戻すなど）。

`TokenCollector` インタフェースには、次のメソッドが指定されています。

- ```
  IdentityToken getToken(String tokenType,
                         HttpServletRequest request,
                         List names,
                                      Properties props)
  ```

  このメソッドはアイデンティティ・トークンを Oracle `IdentityToken` インスタンスとして返します（前項「アイデンティティ・トークン・インタフェースおよび Oracle 実装」を参照）。このメソッドは、入力としてトークン・タイプ（`HTTP_COOKIE`、`HTTP_HEADER` または `HTTP_REQUEST`）、現在のリクエスト・オブジェクト、Cookie またはヘッダーの名前のリストを HTTP リクエストから受け取ります。これらは、トークンおよび構成できるカスタム・プロパティを構成します。
  ```

HTTP_REQUEST を使用している場合、名前のリストは適用できないため、null になります。HTTP_COOKIE を使用している場合、リストは idm.token.collector.cookie.# プロパティを使用して構成されている Cookie 名に対応し、HTTP_HEADER を使用している場合、リストは idm.token.collector.header.# プロパティを使用して構成されているヘッダー名に対応します。# は、数値（1、2、...、n）に置き換えられます（13-14 ページの「アイデンティティ管理フレームワーク・プロパティの構成」を参照）。

プロパティのリストは、トークン・コレクタ実装に固有であり、必要に応じてトークン・コレクタによって使用されます。規約上、このようなプロパティ名の先頭に custom. を付ける必要があります（ピリオドを含む）。たとえば、Java SSO プロパティには custom.sso.url.login と custom.sso.key.alias が含まれています。

- ```
  void fail(HttpServletRequest request,
            HttpServletResponse response,
            int reason) throws CollectorException
  ```

  このメソッドは、次のような各種失敗モードでコールされます。

  - getToken() メソッドを実行中に、必要なトークンが HTTP リクエスト・オブジェクト内に見つかりません。
  - アイデンティティが正常にアサートされていません（指定されているトークンが無効な場合など）。
  - アイデンティティが正常に設定されていますが、ユーザーが適切なパーミッションを持たずにリソースにアクセスを試行します（認可の失敗）。

  整数 reason は OC4J によって指定されるコードです。これは、失敗が発生した理由を示し、次の値のいずれかになります（クラス oracle.security.jazn.collector.IdmErrorConstants に定義されている）。

  - REASON_CHALLENGE_USER: ユーザーが認証されていませんが、認証を要求できます（ブラウザでは 401 エラー）。通常、これは Basic 認証用です。ブラウザに認証ウィンドウが再度表示されます。この定数の値は 1 です。
  - REASON_INVALID_USER: ユーザーが認証されていません。資格証明が無効である（401 エラー）などのためです。この定数の値は 1 です。
  - REASON_UNAUTHORIZED: ユーザーが認証されていますが、保護されたリソースへのアクセスを認可されていません（403 エラー）。この定数の値は 2 です。
  - REASON_PRECLUDED_ACCESS: リソースが、ロールを含まないセキュリティ制約によって保護されています（403 エラー）。この定数の値は 3 です。

Oracle は、次の TokenCollector 実装を提供しています。

oracle.security.jazn.collector.oc4j.TokenCollectorImpl

この実装は、アイデンティティ・トークンに対する HTTP Cookie または HTTP ヘッダーの使用をサポートする抽象クラスです。2 つのうちいずれかを使用する場合、TokenCollectorImpl を拡張し、fail() メソッド内に適切なエラー処理を追加できます（ユーザーをログイン・ページにリダイレクトするなど）。

TokenCollectorImpl の getToken() メソッドは、構成（HttpCookieIdentityToken または HttpHeaderIdentityToken）に基づいて、適切なトークン・タイプを返します。

Cookie またはヘッダーのかわりに HTTP リクエスト・オブジェクト自体を使用するには、カスタム・トークン・コレクタを実装する必要があります。

トークン・コレクタは、適切なアイデンティティ・トークン・クラスのコンストラクタを使用してトークンを作成します。HttpCookieIdentityToken、HttpHeaderIdentityToken および HttpRequestIdentityToken コンストラクタは、13-6 ページの「アイデンティティ・トークン・インタフェースおよび Oracle 実装」で説明します。

重要： fail() メソッドの独自の実装を指定する必要があります。このメソッドは、TokenCollectorImpl 内の抽象メソッドです。

  ```

トークン・アサータ・インタフェース

Oracle は、次のトークン・アサータ・インタフェースを提供しています。

`oracle.security.jazn.asserter.TokenAsserter`

このインタフェースを実装して、OC4J によって渡されるアイデンティティ・トークンを受け付け、アイデンティティ・コールバック・ハンドラを OC4J に返します。

OC4J がトークン・コレクタからアイデンティティ・トークンを正常に受信すると、OC4J はトークンをトークン・アサータに渡し、アサータはトークンのアイデンティティを検証する必要があります。

アイデンティティが正常に検証されると、ユーザーは認証済とみなされ、アイデンティティがアイデンティティ・コールバック・ハンドラとして OC4J に返されます。

オプションでは、トークン・アサータはアイデンティティのすべてのロールまたはグループを取得し、そのサブジェクトを移入することもできます。またはログイン・モジュールに残すこともできます。

`TokenAsserter` インタフェースには、次のメソッドが指定されています。

- ```
  IdentityCallbackHandler assertIdentity(String tokenType,
                                         IdentityToken token,
                                         Properties props)
  throws AsserterException
  ```

  検証が正常に完了すると、メソッドは `IdentityCallbackHandler` のインスタンス内でアサートされたアイデンティティを返します（次項「アイデンティティ・コールバック・ハンドラ・インタフェース」を参照）。トークン・タイプは、`HTTP_COOKIE`、`HTTP_HEADER` または `HTTP_REQUEST` です。トークンは、前述のように `IdentityToken` のインスタンスです。プロパティは、13-7 ページの「トークン・コレクタ・インタフェースおよび Oracle 実装」で `getToken()` メソッドについて説明したプロパティと同じです。

トークン・コレクタの `TokenCollectorImpl` を拡張する場合、トークン・アサータが適用可能なアイデンティティ・トークン・メソッドを使用してトークンからアイデンティティ情報を取得する必要があります。`HttpCookieIdentityToken`、`HttpHeaderIdentityToken` および `HttpRequestIdentityToken` 内のこれらのメソッドについては、すでに説明しました。

注意： ログイン・モジュールを使用しないためには、アイデンティティ管理システムにアクセスし、サブジェクトに移入し、`getSubject()` メソッドを実装するトークン・アサータを実装します。この代替方法を選択する場合、アイデンティティ管理プロパティ `idm.subject.loginmodule.disabled` を適切に設定する必要があります。

関連項目：

- `IdentityToken` インタフェース、`HttpCookieIdentityToken` 実装、`HttpHeaderIdentityToken` 実装、`HttpRequestIdentityToken` 実装、および関連するメソッドとコンストラクタの詳細は、13-6 ページの「アイデンティティ・トークン・インタフェースおよび Oracle 実装」を参照してください。
  ```

アイデンティティ・コールバック・ハンドラ・インタフェース

アイデンティティ・コールバック・ハンドラは、トークン・アサータとともに使用します。Oracle は、次のアイデンティティ・コールバック・ハンドラ・インタフェースを提供しています。

```
oracle.security.jazn.callback.IdentityCallbackHandler
```

アイデンティティ・コールバック・ハンドラは、トークン・アサータによって構成され、アサートされたアイデンティティを OC4J に渡すために使用されます。これは、通常はログイン・モジュールにも渡されるユーザー・アイデンティティであるため、アイデンティティに対応するサブジェクトに必要に応じてプリンシパルを移入することができます。

`IdentityCallbackHandler` インタフェースには、次のメソッドが指定されています。

- `void setIdentity(String identity) throws AsserterException`

  このメソッドは、文字列を受け取ってユーザー・アイデンティティを指定します。

- `String getIdentity()`

  このメソッドは、ユーザー・アイデンティティとともに文字列を返します。

- `Subject getSubject()`

  トークン・アサータによってサブジェクトが移入された場合、アイデンティティ管理フレームワークはこのメソッドを使用してサブジェクトを取得します。それ以外の場合（ログイン・モジュールがサブジェクトを移入する場合）、このメソッドは `null` を返します。

次のような標準のコールバック・ハンドラ・メソッドもあります。

- `void handle(Callback[] callbacks)`

  このメソッドは、提供されたコールバック内の情報を処理するために（取得、表示など）、1 つ以上の `javax.security.auth.callback.Callback` インスタンスの配列を受け取ります。

Oracle は、次のアイデンティティ・コールバック・ハンドラ実装を提供しています。

```
oracle.security.jazn.callback.IdentityCallbackHandlerImpl
```

Oracle コールバック実装

アイデンティティ管理フレームワークで使用するために、Oracle は `oracle.security.jazn.callback` パッケージ内に次のコールバック実装を提供しています。

- `IdentityCallback`
- `HttpRequestCallback`

アイデンティティ・コールバック

アイデンティティ管理フレームワークで使用するために、Oracle は次のアイデンティティ・コールバック・クラスを提供しています。

```
oracle.security.jazn.callback.IdentityCallback
```

`IdentityCallback` インスタンスを使用すると、アイデンティティの認証状態（認証済であるかどうか）および認証に使用される認証方式に加えて、アイデンティティ自体を取得することができます。

コンストラクタにはパラメータはありません。

```
IdentityCallback()
```

次に、アイデンティティ・コールバック・ハンドラ handle() メソッドを使用して、アイデンティティ・コールバックをコールバック・ハンドラ内に設定できます。

```
// In token asserter:
IdentityCallbackHandler ich = new IdentityCallbackHandlerImpl("identity");
...

// In login module:
IdentityCallback icb = new IdentityCallback();
Callback[] callback = {icb};
ich.handle(callback);
```

IdentityCallback クラスには、次のメソッドがあります。

- String getIdentity()

  アイデンティティ・コールバックからアイデンティティを文字列として取得します。

- boolean isIdentityAsserted()

  アイデンティティがアサートされている場合は true を、アサートされていない場合は false を返します。

- String getAuthenticationType()

  アイデンティティの認証に使用される認証方式を示します（フォームベースや Basic など）。

HTTP リクエスト・コールバック

HTTP Cookie または HTTP ヘッダーをトークン・タイプとして使用しないアイデンティティ管理フレームワーク実装用に、Oracle は次の HTTP リクエスト・コールバック・クラスを提供しています。

oracle.security.jazn.callback.HttpRequestCallback

HTTP リクエスト・コールバック・インスタンスを使用すると、ログイン・モジュールがリクエスト・オブジェクトから資格証明を取得し、ユーザーを認証し、ユーザーのサブジェクトに移入できます。

コンストラクタにはパラメータはありません。

HttpRequestCallback()

次に、アイデンティティ・コールバック・ハンドラ handle() メソッドを使用して、HTTP リクエスト・コールバックをコールバック・ハンドラ内に設定できます。

```
// In token asserter:
IdentityCallbackHandler ich = new IdentityCallbackHandlerImpl("identity");
...

// In login module:
HttpRequestCallback httpreqcb = new HttpRequestCallback();
Callback[] callback = {httpreqcb};
ich.handle(callback);
```

HttpRequestCallback クラスには、次のメソッドがあります。

- void setHttpRequest(HttpServletRequest request)

  このメソッドを使用して HTTP リクエスト・オブジェクトを設定します。ユーザー資格証明が含まれていることが想定されています。

- HttpServletRequest getHttpRequest()

  このメソッドは、HTTP リクエスト・オブジェクトを返します。

ログイン・モジュールの要件

一般的に、アイデンティティ管理フレームワークの実装にはカスタム・ログイン・モジュールが含まれています。アイデンティティがトークン・アサータによって正常にアサートされると、トークン・アサータはアイデンティティをアイデンティティ・コールバック・ハンドラとして OC4J に渡します。OC4J はこのアイデンティティをログイン・モジュールに渡します。

JAAS の典型的な使用方法では、ログイン・モジュールが、ユーザーの認証およびユーザーのサブジェクト移入という 2 つの重要な機能を実行します。ただし、アイデンティティ管理フレームワーク内では、ログイン・モジュールが認証を処理する必要はありません。ログイン・モジュールの重要な機能は、ユーザーのロールに関する情報を取得してサブジェクトに移入するために、必要に応じてアイデンティティ管理システムにアクセスすることです。

ログイン・モジュールとアイデンティティ管理フレームワークを併用する場合、ログイン・モジュールがユーザー名を渡すために使用される `IdentityCallbackHandler` インスタンスを処理できる必要があります。ログイン・モジュールが実行しなければならないことは、次のとおりです。

1. OC4J から `IdentityCallbackHandler` インスタンスを受信します。
2. ユーザー情報を処理するために適切なコールバックを作成します。Cookie またはヘッダーのトークンを使用する場合、このコールバックには、`javax.security.auth.callback.NameCallback` や `PasswordCallback` など、標準的なコールバックに加えて、Oracle コールバック `oracle.security.jazn.callback.IdentityCallback` も含めることができます。HTTP リクエストのトークンを使用する場合、Oracle コールバック `oracle.security.jazn.callback.HttpRequestCallback` を使用します。
3. `handle(Callback[])` メソッドを使用して、コールバックを `IdentityCallbackHandler` インスタンスに渡します。
4. 適切なコールバックを使用してアイデンティティを取得します。
5. アイデンティティ・システムに移動してアイデンティティのロールまたはグループを検索します。
6. サブジェクトに移入して OC4J に送信します。

注意：

- カスタム・ログイン・モジュールは、アイデンティティ管理フレームワークとともに使用する必要はありません。ファイルベース・プロバイダまたは Oracle Identity Management の場合、`RealmLoginModule` で十分です。サポートされる外部 LDAP プロバイダの場合（Active Directory または Sun Java System Directory Server）、`LDAPLoginModule` で十分です。
- ログイン・モジュールを使用しないためには、アイデンティティ管理システムにアクセスし、サブジェクト自体に移入し、`getSubject()` メソッドを実装するトークン・アサータをかわりに実装します。この代替方法を選択する場合、アイデンティティ管理フレームワークのプロパティ `idm.subject.loginmodule.disabled` を適切に設定する必要があります。

関連項目：

- 2-14 ページの「JAAS 認証 : ログイン・モジュール」

サブジェクト・アサータ・インタフェース

オプションでサブジェクト・アサータを実装すると、トークン・アサータまたはログイン・モジュールによって移入および返されたサブジェクトを検証することができます。OC4J は、サブジェクト・アサータが構成されている場合にそれを起動します。典型的な使用例では、サブジェクトが署名され、署名がプリンシパルとしてサブジェクトに追加され、サブジェクト・アサータが署名を確認し、検証します。

Oracle は、次のサブジェクト・アサータ・インタフェースを提供しています。

`oracle.security.jazn.asserter.SubjectAsserter`

`SubjectAsserter` インタフェースには、次のメソッドが指定されています。

- `boolean assertSubject(Subject subject) throws AsserterException`

  このメソッドを使用してサブジェクトを検証します。成功の場合は true が返されます。

アイデンティティ管理フレームワーク実装クラスのパッケージ化

アイデンティティ管理フレームワーク用に作成する実装クラスはアプリケーションの一部ではありません。また、アプリケーションとともにパッケージ化とデプロイもされていません。実装クラスを 1 つの JAR ファイルにパッケージ化し、その JAR ファイルを OC4J クラスパスに追加します。これは、共有ライブラリとして行うことができます。共有ライブラリは、それを使用とするアプリケーションによってインポートできます（6-15 ページの「ライブラリを共有するためのタスク」を参照）。

ログイン・モジュールをアイデンティティ管理フレームワーク実装の一部として実装すると、その実装を同じライブラリ内にも、個別のライブラリとしても含めることができます。

注意： 共有ライブラリの機能は OC4J アプリケーションで使用できる JAR ファイルを作成する場合に適した方法ですが、別な方法として JAR ファイルを次のディレクトリに配置することもできます。

ORACLE_HOME`/j2ee/home/lib/ext`

（この操作を行うと、ファイルがグローバルに入手可能になります。）

アイデンティティ管理フレームワークの構成

ここではアイデンティティ管理フレームワークの構成について説明します。この項の内容は次のとおりです。

- アイデンティティ管理フレームワーク・プロパティの構成
- アイデンティティ管理フレームワーク・ログイン・モジュールの構成
- アイデンティティ管理フレームワークを使用するアプリケーションの構成
- 複数 OC4J インスタンスの場合の考慮事項

管理者はフレームワークのプロパティとログイン・モジュールを構成します。アプリケーション・アセンブラは、認証方式を設定して、フレームワークを使用するアプリケーションを構成します。

注意： デフォルトでは、単一インスタンス OC4J インストールでは、アイデンティティ管理フレームワークの実装である Java SSO は、事前に構成されています（14-14 ページの「単一インスタンス OC4J インストール用デフォルト Java SSO プロパティ設定」を参照）。

アイデンティティ管理フレームワーク・プロパティの構成

管理者は、jazn.xml を構成して、使用するクラス実装に適したアイデンティティ管理フレームワークのプロパティを設定します。表 13-1 は、アイデンティティ管理フレームワークのプロパティを示しています。

表 13-1 アイデンティティ管理フレームワークのプロパティ

プロパティ	説明
idm.authentication.name	これは、アイデンティティ管理フレームワークの使用方法を指定するか限定する任意の名前にすることができます。単なる参照用であり、フレームワークによって使用されることはありません。（たとえば、OC4J Java SSO の場合、規約上、これは JavaSSO に設定されます。）
idm.token.type	使用するアイデンティティ・トークンのタイプ。HTTP_COOKIE、HTTP_HEADER または HTTP_REQUEST です。
idm.token.collector.class	トークン・コレクタ・インタフェースを実装するクラスの完全修飾名。
idm.token.asserter.class	トークン・アサータ・インタフェースを実装するクラスの完全修飾名。
idm.subject.asserter.class	サブジェクト・アサータ・インタフェースを実装するクラスの完全修飾名（必要に応じて）。
idm.subject.loginmodule.disabled	ログイン・モジュールを起動するかどうかを指定するブール（デフォルトでは false であるため、ログイン・モジュールは有効になっています）。ログイン・モジュールが起動されないと、トークン・アサータがサブジェクトに移入する必要があります。
idm.token.collector.cookie.#	アイデンティティ情報を含む Cookie の名前（複数可）。# を数値に置き換えて、idm.token.collector.cookie.1 など、1 つ以上を指定できます。（通常は、1 つのみです。）
idm.token.collector.header.#	アイデンティティ情報を含む HTTP ヘッダーの名前（複数可）。Cookie と同じように、# を数値に置き換えて、1 つ以上を指定できます。（通常、ユーザー名とパスワードの場合は 2 つです。）

<jazn> 要素の <property> サブ要素内にこれらのプロパティを設定します。次に例を示します。

```
<jazn provider="XML" location="./system-jazn-data.xml" default-realm="jazn.com">
   ...
   <!-- properties to configure the 3rd party IDM framework -->
   <property name="idm.authentication.name" value="JavaSSO" />
   <property name="idm.token.asserter.class"
             value="oracle.security.jazn.sso.SSOCookieTokenAsserter" />
   <property name="idm.token.collector.class"
             value="oracle.security.jazn.sso.SSOCookieTokenCollector" />
   <property name="idm.token.type" value="HTTP_COOKIE" />
   <property name="idm.token.collector.cookie.1" value="ORA_OC4J_SSO"/>
   ...
</jazn>
```

重要：

- デフォルトでは、アイデンティティ管理フレームワークは Java SSO を使用するように構成されています（第 14 章「OC4J Java シングル・サインオン」を参照）。
- OC4J インスタンスを Oracle Internet Directory インスタンスに関連付けると、OC4J home インスタンスの jazn.xml ファイル内にある <jazn> 要素構成は再度書き込まれます。以前の設定は失われます。
- 既存の 10.1.3.0.0 インストールの上に OC4J 10.1.3.1 パッチをインストールする場合、（10.1.3.1 フレッシュ・インストールとは異なり）、デフォルト構成は適していません。jazn.xml のプロパティが参照されないためです。この場合、前述の構成を jazn.xml に手動で追加します。

アイデンティティ管理フレームワーク・ログイン・モジュールの構成

一般的にカスタム・ログイン・モジュールを使用することを想定しますが、その場合はモジュールを構成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時またはその後に Application Server Control を使用して、カスタム・ログイン・モジュールを構成できます（9-15 ページの「Application Server Control でのカスタム・セキュリティ・プロバイダの構成」を参照）。（OracleAS JAAS Provider Admintool には、9-20 ページの「Admintool を使用したログイン・モジュールの構成と RMI パーミッションの付与」に記載されているように、ログイン・モジュール構成用のオプションもあります。）

構成は、system-jazn-data.xml ファイルに格納されます。次に、アプリケーション myapp とともに使用されるカスタム・ログイン・モジュール CustomLM の例を示します。

```
<jazn-loginconfig>
   <application>
      <name>myapp</name>
      <login-modules>
         <login-module>
            <class>mypackage.CustomLM</class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>addAllRoles</name>
                  <value>true</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
</jazn-loginconfig>
```

関連項目：

- 第 9 章「ログイン・モジュール」

アイデンティティ管理フレームワークを使用するアプリケーションの構成

アプリケーションでアイデンティティ管理フレームワークを使用する場合、アプリケーション・アセンブラが orion-application.xml ファイルの <jazn-web-app> 要素内に認証方式 CUSTOM_AUTH を指定する必要があります。次に例を示します。

```
<jazn provider="XML" ... >
   ...
   <jazn-web-app auth-method="CUSTOM_AUTH" />
   ...
</jazn>
```

これにより、jazn.xml のフレームワーク・プロパティの構成、および system-jazn-data.xml（該当する場合）のログイン・モジュールの構成に基づいて、アイデンティティ管理フレームワークの使用がトリガーされます。

重要： Application Server Control を介して任意の時点で任意のアプリケーションに対してファイルベース・プロバイダから Oracle Identity Management に切り替えると、そのアプリケーションの orion-application.xml 内にある <jazn> 要素が次のように置き換えられます。アイデンティティ管理フレームワークの CUSTOM_AUTH 設定は失われ、再設定が必要になります。

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

注意：

- <jazn-web-app> 要素も、特定の Web アプリケーションに対して、<orion-web-app> のサブ要素として orion-web.xml ファイルでサポートされます。この設定は、Web アプリケーションの orion-application.xml 設定よりも優先されます。
- orion-application.xml または orion-web.xml 内の認証方式設定は、web.xml 内の認証方式設定よりも優先されます。

複数 OC4J インスタンスの場合の考慮事項

複数の OC4J インスタンスを使用するインストール・タイプの場合、インスタンス間でアイデンティティ管理フレームワークの構成が複製されている必要があります。これには、jazn.xml 内のアイデンティティ管理フレームワーク・プロパティ設定、および system-jazn-data.xml 内のログイン・モジュール構成（該当する場合）が含まれています。必要に応じて次の手順を実行します。

1. 各 OC4J インスタンスに同じセキュリティ・プロバイダ構成を繰り返します。
2. OC4J グループ機能を使用して system-jazn-data.xml 更新を調整します。これにより、各 system-jazn-data.xml ファイルが OC4J インスタンスのグループ内で更新されます。これには、ファイルベース・セキュリティ・プロバイダを使用する場合のユーザー設定も含まれます。7-19 ページの「クラスタ MBean ブラウザの機能および J2EEServerGroup MBean」では、OC4J インスタンス間で各 system-jazn-data.xml ファイルの設定を調整する方法について説明します。インスタンス間でログイン・モジュール構成をメンテナンスする操作もあります（setLoginModule など）。
3. 必要に応じて、各 OC4J インスタンスに移動し、適宜 Application Server Control を使用するか手動で構成を繰り返します。一般に、アイデンティティ管理フレームワーク設定など、jazn.xml ファイルのプロパティ設定に対しては、各 OC4J インスタンスの jazn.xml を手動で構成することが唯一のオプションです。

関連項目：

- OC4J グループ機能の追加情報は、Application Server Control オンライン・ヘルプのグループ OC4J インスタンスのページに関するトピックを参照してください。

アイデンティティ管理フレームワーク使用方法の概要

ここでは、これまでに説明したアイデンティティ管理フレームワークの使用に関連する重要な作業をまとめます。

1. インテグレータ（いくつかのサード・パーティのアイデンティティ管理システムを OC4J に統合する開発者）は、必要に応じて、アイデンティティ管理フレームワーク・コンポーネント、トークン・コレクタ、アイデンティティ・トークン、トークン・アサータ、アイデンティティ・コールバック・ハンドラおよびサブジェクト・アサータのクラスを実装します。
2. インテグレータは、必要に応じてカスタム・ログイン・モジュールを開発します。
3. インテグレータは、実装クラスを JAR ファイルにパッケージ化し、ログイン・モジュールを同じ JAR ファイルまたは個別の JAR ファイルにパッケージ化します。
4. インテグレータまたは管理者は、各 JAR ファイルを共有ライブラリとしてターゲット・システムにデプロイします。
5. アプリケーションは、アイデンティティ管理フレームワークを使用するように有効化されます。これは、デプロイメント前に、アセンブラが、アプリケーションとともにパッケージ化されている `orion-application.xml` ファイルの `<jazn-web-app>` 要素内にある設定 `auth-method="CUSTOM_AUTH"` で行うことができます。
6. アプリケーションのデプロイヤは、Application Server Control を使用してアプリケーションをデプロイします。これには、カスタム・ログイン・モジュールを構成すること、および実装クラスから成る共有ライブラリをインポートすることが含まれています（カスタム・ログイン・モジュールは、後で管理者が構成することもできます）。
7. ターゲット・システム上で、管理者が `jazn.xml` ファイルのアイデンティティ管理フレームワーク・プロパティを構成します。

サンプル使用例 : ヘッダーベース・アイデンティティ・トークンの使用

このサンプルの前提事項は、認証でユーザーがカスタム・アイデンティティ・ストアと照合されることです。次に、ログインしたユーザーのアイデンティティを含むカスタム HTTP ヘッダー（`Acme-Custom-Auth`）がリクエストに追加されます。サンプルのコンポーネントは次のように機能します。

- トークン・コレクタ実装がヘッダー値を抽出します。
- トークン・アサータ実装がヘッダーからアイデンティティを検証し、Oracle が提供する `IdentityCallbackHandlerImpl` クラスを使用し、アイデンティティ・コールバック・ハンドラ機能を介してこのアイデンティティをアサートします。

対応する構成は、`jazn.xml` に示されているようになります。

サンプルのトークン・コレクタ : CollectorImpl.java

ここには、サンプルのトークン・コレクタ実装用コードを示します。

```
package com.acme.idm;

import java.io.IOException;

import java.util.List;
import java.util.Map;
import java.util.Properties;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import oracle.security.jazn.collector.CollectorException;
import oracle.security.jazn.collector.oc4j.TokenCollectorImpl;
import oracle.security.jazn.token.HttpHeaderIdentityToken;
import oracle.security.jazn.token.IdentityToken;
import oracle.security.jazn.token.TokenNotFoundException;
import oracle.security.jazn.collector.IdmErrorConstants;


public class CollectorImpl extends TokenCollectorImpl {
    public CollectorImpl() {
    }

    public IdentityToken getToken(String tokenType,
                                  HttpServletRequest request,
                                  List tokenNames,
                                  Properties properties)
                                            throws CollectorException,
                                                              TokenNotFoundException
{
        if (null == tokenType || 0 == tokenType.length() ||
            !IdentityToken.HTTP_HEADER.equalsIgnoreCase(tokenType)) {
            throw new CollectorException("invalid token type" + tokenType);
        }

        HttpHeaderIdentityToken identityToken =
            (HttpHeaderIdentityToken)super.getToken(tokenType,
                                                    request,
                                                    tokenNames,
                                                    properties);
        Map m = identityToken.getHeaderValues();

        if (m == null || m.size() == 0) {
            throw new TokenNotFoundException("no HTTP Header token was found");
        }

        return identityToken;
    }

    public void fail(HttpServletRequest httpServletRequest,
                     HttpServletResponse httpServletResponse, int reason) {
        try {
            switch (reason) {
                case IdmErrorConstants.REASON_INVALID_USER:
              httpServletResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED);
                case IdmErrorConstants.REASON_UNAUTHORIZED:
              httpServletResponse.sendError(HttpServletResponse.SC_FORBIDDEN);
            }
        } catch (Exception e) {
            System.err.println("failed to send response " + e);
```

```
            e.printStackTrace(System.err);
        }
    }
}
```

サンプルのトークン・アサータ : TokenAsserterImpl.java

ここには、サンプルのトークン・アサータ実装用コードを示します。

前提事項は、すべての認証済ユーザーに対して、Oracle HTTP Server または OC4J のフロントにある他の Web サーバーが "Acme-Custom-Auth" を "ANONYMOUS" に設定することです。Acme 実装には、ユーザーが認証されることが必要です。(本番環境に近い実装では、ヘッダーでクリアなユーザー名を渡さない可能性が高いと考えられます。ヘッダーが生成された場合に検証する方法がないためです。)

```
package com.acme.idm;

import java.util.Map;
import java.util.Properties;

import oracle.security.jazn.asserter.AsserterException;
import oracle.security.jazn.asserter.TokenAsserter;
import oracle.security.jazn.callback.IdentityCallbackHandler;
import oracle.security.jazn.callback.IdentityCallbackHandlerImpl;
import oracle.security.jazn.token.IdentityToken;
import oracle.security.jazn.token.HttpHeaderIdentityToken;

public class TokenAsserterImpl implements TokenAsserter {
    private static final String HEADER_NAME = "Acme-Custom-Auth";
    private static final String AUTH_TYPE = "CUSTOM_HTTP_HEADER";
    public TokenAsserterImpl() {
    }

    public IdentityCallbackHandler assertIdentity(String tokenType,
                                                  IdentityToken identityToken,
                                                  Properties properties)
                                                  throws AsserterException {
        if (tokenType != null &&
            tokenType.length() > 0 &&
            IdentityToken.HTTP_HEADER.equalsIgnoreCase(tokenType)) {
            HttpHeaderIdentityToken token =
                                          (HttpHeaderIdentityToken) identityToken;
            Map m = token.getHeaderValues();
            if (m != null && m.size() > 0) {
                String user = (String) m.get(HEADER_NAME);
                if ("ANONYMOUS".equalsIgnoreCase(user)) {
                    throw new AsserterException
                                    ("anon user - expected authenticated user");
                }
                IdentityCallbackHandler idcb =
                                            new IdentityCallbackHandlerImpl(user);
                idcb.setIdentityAsserted(true);
                idcb.setAuthenticationType(AUTH_TYPE);

                return idcb;
            }
            else {
                throw new AsserterException
                                ("not a valid token - no identity to assert");
            }
        }
    }
}
```

サンプル構成 : jazn.xml

ここには、このサンプル用 jazn.xml ファイルの構成を示します。

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<jazn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
                        "http://xmlns.oracle.com/oracleas/schema/jazn-10_0.xsd"
    schema-major-version="10" schema-minor-version="0"
    provider="XML" location="./system-jazn-data.xml"
    default-realm="jazn.com" >
        <property name="idm.token.asserter.class"
                  value="com.acme.idm.TokenAsserterImpl" />
        <property name="idm.token.collector.class"
                  value="com.acme.idm.CollectorImpl" />
        <property name="idm.token.type" value="HTTP_HEADER" />
        <property name="idm.token.collector.header.1" value="Acme-Custom-Auth" />
        <property name="idm.authentication.name" value="Acme-IDM" />
</jazn>
```

14

OC4J Java シングル・サインオン

OC4J 10.1.3.1 実装には、他の Oracle Application Server シングル・サインオン製品のような、必要な追加インフラストラクチャに依存しない代替 Java シングル・サインオン・ソリューションがパッケージ化されています。この Java SSO は、第 13 章で説明する OracleAS JAAS Provider アイデンティティ管理フレームワークに基づいており、次のいずれのデプロイメント・シナリオでも Web アプリケーション間で使用できます。

- Web アプリケーションが、同じアプリケーション EAR ファイルにデプロイされている場合。
- Web アプリケーションが、同じ OC4J インスタンスの異なるアプリケーション EAR ファイルにデプロイされている場合。
- Web アプリケーションが、異なる OC4J インスタンスの異なるアプリケーション EAR ファイルにデプロイされている場合。この Web アプリケーションは、共通するセキュリティ・ドメインと Cookie ドメインを共有します。
- 複数の Web アプリケーションを含む単一のアプリケーション EAR ファイルが、OC4J クラスタ内の複数の OC4J インスタンスにデプロイされている場合。

この章では、OC4J Java SSO について説明します。この章の内容は次のとおりです。

注意：

- ファイルベース・プロバイダまたは Oracle Internet Directory 以外のアイデンティティ・ストアを使用する場合、管理ユーザー・アカウントと管理ロールを定義してロールをユーザーに付与し、必要なパーミッションをロールに付与する必要があります（10-10 ページの「管理ユーザーとロールの作成および RMI パーミッションの付与」を参照）。
- OC4J Java SSO は、Oracle Application Server 10*g* リリース 2（10.1.2）の軽量 J2EE シングル・サインオン機能にかわる機能です。軽量 J2EE シングル・サインオン機能は、サポートされなくなりました。

OC4J Java SSO の概要

ここでは、次に示すような Java SSO の概要について説明します。

- OC4J コンテナ・レベル Java シングル・サインオン・ソリューションの必要事項
- Java SSO の仕組み
- Java SSO のデプロイメント・シナリオ
- Java SSO 構成の概要
- Java SSO ログイン・ページとエラー・ページの概要

OC4J コンテナ・レベル Java シングル・サインオン・ソリューションの必要事項

Oracle Application Server で Oracle Single Sign-On を使用するには、Oracle Identity Management による個別の Oracle Application Server のインストールが必要です（Oracle Internet Directory、Oracle Single Sign-On および Oracle Database を含む）。または、Oracle Access Manager のシングル・サインオン・ソリューションを使用するには、Oracle Access Manager を完全にインストールする必要があります。これらのシングル・サインオン・ソリューションは、必要なインフラストラクチャと密接に連携しています。どちらも堅牢ですが、小規模なデプロイメント・シナリオでは実行できず、それ自体ではスタンドアロン OC4J と併用できません。

または、OC4J 10.1.3.1 実装によって Java SSO を提供する方法もあります。この場合、Java SSO は OC4J 自体とともにパッケージ化され、使用する Identity Server の種類に関係なく、OC4J との連携が解除されます。これにより、同じ OC4J インスタンスやクラスタにデプロイされたアプリケーションの認証を一元化し、ユーザー・アイデンティティを共有することができます。

OC4J Java SSO は、OracleAS JAAS Provider によってサポートされる任意のセキュリティ・プロバイダと併用できます（概要は、3-2 ページの「OracleAS JAAS Provider およびセキュリティ・プロバイダの概要」を参照）。さらに、特定の要件に準拠するログイン・モジュール（カスタム・モジュールを含む）を Java SSO と併用できます。特に、OC4J に付属するログイン・モジュール `RealmLoginModule`、`LDAPLoginModule`、`DBTableOraDataSourceLoginModule` は、アイデンティティ管理フレームワークのアイデンティティ・アサーション機能をサポートし、Java SSO およびパートナ・アプリケーションと併用できます。（13-10 ページの「Oracle コールバック実装」で説明する `IdentityCallback` のインスタンスは、アサートするアイデンティティの確認に使用されます。）

注意： 10.1.3.1 Java SSO 実装には、次のような制限があります。

- アクセス制御機能と認可機能がありません。認証とアイデンティティ・アサーション専用です。
- 共通する Cookie ドメインを共有している Web アプリケーションに限定されます。
- `UserManager` インタフェース（一般的な使用には非推奨）はサポートされません。

Java SSO の仕組み

Java SSO には、専用の Java SSO アプリケーション（OC4J で事前にデプロイ）と OracleAS JAAS Provider のアイデンティティ管理フレームワークの実装が含まれています。

この項の内容は次のとおりです。

シングル・サインオンの対話とロジック・フロー

OC4J Java SSO と特定のアプリケーションが連携する仕組みを説明するため、次の 3 つの基本的なシナリオを検討します。

- アイデンティティ・トークンを持たないユーザーが、アプリケーションにアクセスしようとする場合。この場合は、ログインが必要です。IT トークンは、ユーザーおよびユーザー資格証明と対応します（第 13 章「交換可能なアイデンティティ管理フレームワーク」を参照）。Java SSO の場合、トークンは HTTP Cookie になります。
- アイデンティティ・トークンを持つユーザーが、アプリケーションにアクセスしようとする場合。この場合、トークンが検証され、アイデンティティがアサートされる必要があります。
- 認証済ユーザーがアプリケーションにアクセスしようとする場合。この場合、サブジェクトとアイデンティティ・トークンの両方を使用でき、追加の認証とアサートはいずれも必要ありません。

通常の Java SSO との対話は、次のとおりです。

1. クライアントが、`App1` などのアプリケーションから、保護されたコンテンツにアクセスしようとします（保護された URL をリクエスト）。
2. `App1` は SSO 認証を必要とするため、クライアントは Java SSO によって資格証明を要求されます。
3. クライアントは資格証明を提示します。
4. Java SSO は、認証をセキュリティ・プロバイダに委任します。
5. 認証が正常に完了すると、セキュリティ・サブシステムがアイデンティティを設定します。
6. SSO プロバイダ（アイデンティティ管理フレームワークの Java SSO 実装）は、認証済ユーザーに対応するアイデンティティ・トークン（Java SSO Cookie）をエンコードし、Cookie をクライアントに返します。
7. クライアントは、Cookie を提示して `App1` にアクセスします。
8. `App1` は、Cookie のクライアント・アイデンティティを確認し、保護されたコンテンツへのクライアントのアクセスを許可します。
9. クライアントは、もう一度この Cookie を使用して、別のアプリケーション（`App2` など）の保護されたコンテンツにアクセスしようとします。
10. `App2` は、Cookie のクライアント・アイデンティティを確認し、保護されたコンテンツへのクライアントのアクセスを許可します。

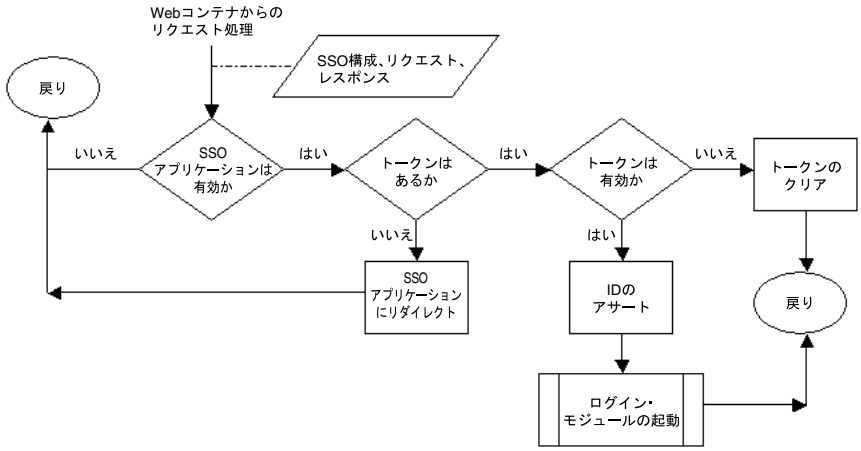
Java SSO の使用中は、次のいずれかが起きるまで、ユーザー・セッションはアクティブの状態になります。

- アプリケーションが Java SSO ログアウト API をコールする（14-18 ページの「Java SSO ログアウト API」を参照）。
- Java SSO の Cookie が無効になる。

Java SSO の使用を共有する一連のアプリケーションは、パートナ・アプリケーションと呼ばれています。パートナ・アプリケーションは、カスタマ・アプリケーションか Oracle Application Server コンソール・アプリケーション（Application Server Control、Oracle BPEL Process Manager または Oracle Web Services Manager など）のいずれかになります。パートナ・アプリケーションはセキュリティ・ドメイン内のアプリケーションの 1 つであり、Java SSO を使用して認証を委任します。セキュリティ・ドメインでは、複数のアプリケーションが認証プロセスと検証プロセスに使用される共通のアイデンティティ・ストア、アルゴリズムおよびキーを共有します。

次の図 14-1 は、Java SSO ロジックのフロー・チャートを示しています。

図 14-1 Java SSO 内部ロジック・フロー



Java SSO 実行時の操作

Java SSO のアプリケーションである javasso は、組込みシステム・アプリケーションとしてすべての OC4J インスタンスにデプロイされています。このアプリケーションは、ログイン・サーブレット SSOLogin とログアウト・サーブレット SSOLogout から構成されます。

OC4J を起動する場合、通常は javasso が無効になります（Oracle Application Server の基本インストールを除く）。管理者は、Application Server Control を使用して javasso を起動できます。

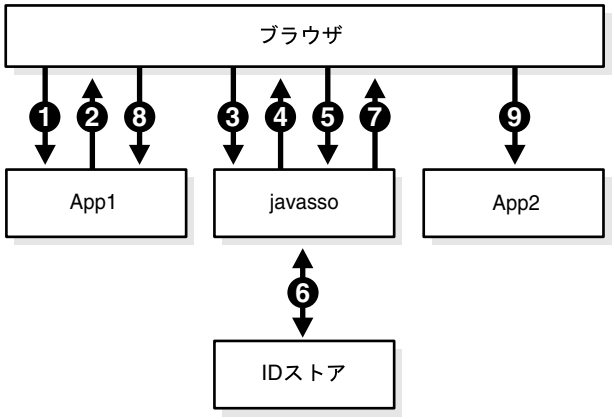
javasso が有効になった後で Java SSO パートナ・アプリケーションにアクセスしようとすると、プロセスは図 14-2 のようになります。

> **注意：** 次の App1 と App2 は、シングル・サインオン用に Java SSO を使用するよう構成されたパートナ・アプリケーションです。

1. ユーザーは、ブラウザを使用してアプリケーション App1 の保護された URL にアクセスしようとします。
2. ブラウザによって javasso にリダイレクトされます（具体的には SSOLogin サーブレット）。
3. App1 の URL が javasso に渡されます。
4. javasso 機能により、ユーザーに対して Java SSO ログイン・ページが表示されます。
5. ユーザーは、ユーザー名とパスワードを入力します。

6. javasso 機能により、ユーザーはアイデンティティ・ストア内の情報に基づいて認証されます。ユーザーが正常に認証されると、SSOLogin サーブレットは認証済アイデンティティを Java SSO の Cookie（アイデンティティ・トークン）にマップし、対称鍵暗号を使用して署名を行い、Cookie の内容を暗号化します。
7. ブラウザによって App1 にリダイレクトされ、保護された Cookie がクライアントに返されます。
8. ユーザーは、App1 のコンテンツにアクセスします。
9. ユーザーがアプリケーション App2 の保護された URL にアクセスしようとすると、認証を受けることなく App2 のコンテンツにアクセスできます。

図 14-2 Java SSO 実行時の操作



アイデンティティ管理フレームワークの Java SSO 実装

Java SSO は、第 13 章「交換可能なアイデンティティ管理フレームワーク」に記載されたアイデンティティ管理フレームワークの実装です。Java SSO 実装では、資格証明に Cookie を使用します。

Java SSO は、OC4J とともにインストールされます。Java SSO には、次のものが含まれています。

- ユーザー資格証明用 Cookie の ORA_OC4J_SSO。
- トークン・コレクタ実装クラスの SSOCookieTokenCollector。HTTP リクエストを介して渡された Cookie から資格証明を収集し、Cookie のアイデンティティ・トークンを作成して OC4J に返します。トークンの内容には、内容の整合性を保証するためのユーザー・アイデンティティ、発行認証局、トークンの有効期間、デジタル署名が含まれます。
- トークン・アサータ実装クラスの SSOCookieTokenAsserter。ユーザー・アイデンティティをアサートします。つまり、OC4J から Cookie トークンを受信して検証し、Cookie 内のアイデンティティを確認し、アサートされたアイデンティティをアイデンティティ・コールバック・ハンドラで OC4J に戻します。

  Cookie トークンをデコードするため、トークン・アサータはすべてのパートナ・アプリケーションと Java SSO アプリケーション間で共有されている共有キーを取得し、Cookie の内容を復号化します。トークン・アサータは、署名を検証してデータを確認します。（Cookie の有効期限が切れている場合はエラーがスローされますが、通常はユーザーがログイン・ページにリダイレクトされます。）

> **注意：** Java SSO 実装では、アイデンティティ・トークンとして使用される Cookie はセッション Cookie であり、永続的なものではありません。

Java SSO のデプロイメント・シナリオ

次の Java SSO デプロメント・シナリオは重要です。

- OC4J の単一インスタンス

  `javasso` アプリケーション、カスタマ・アプリケーション、Application Server Control が、すべて同じ OC4J インスタンスで実行されます。

- ファイルベース・プロバイダを使用し、2 つの OC4J インスタンスを備えた SOA インストール

  Oracle BPEL Process Manager（Business Process Execution Language エンジン）と OWSM（Oracle Web Services Manager）が、`OC4J_SOA` インスタンスで実行されます。`javasso` アプリケーション、カスタマ・アプリケーション、Application Server Control が、`OC4J_Home` インスタンスで実行されます。

  `system-jazn-data.xml` ファイル（ファイルベース・プロバイダ）は、2 つのインスタンス間で同期されている必要があります（14-16 ページの「ファイルベース・プロバイダと 2 つの OC4J インスタンスを使用する場合の考慮事項」を参照）。

- Oracle Internet Directory を使用し、2 つの OC4J インスタンスを備えた SOA インストール

  Oracle BPEL Process Manager と OWSM が、`OC4J_SOA` インスタンスで実行されます。`javasso` アプリケーション、カスタマ・アプリケーション、Application Server Control が、`OC4J_Home` インスタンスで実行されます。

  単一の Oracle Internet Directory インスタンスが、セキュリティ・プロバイダとして使用されます。

- Oracle Internet Directory を使用し、複数の OC4J インスタンスを備えた高可用性インストール

  `javasso` アプリケーションとすべてのカスタマ・アプリケーションは、すべてのインスタンスで使用できます。

  単一の Oracle Internet Directory インスタンスが、セキュリティ・プロバイダとして使用されます。

注意：

- パートナ・アプリケーション（Oracle Application Server のコンソール・アプリケーションまたはカスタマ・アプリケーション）は、必要に応じて OC4J インスタンス間で分割できますが、前述のようなシナリオが一般的です。
- Oracle Identity Management が完備された環境で Oracle Internet Directory を使用するユーザーは、通常の場合 Java SSO ではなく Oracle Single Sign-On を使用します。ただし、Oracle Internet Directory を使用した Java SSO が便利な場合もあります。

Java SSO 構成の概要

ユーザーの J2EE アプリケーションを標準以外に構成する場合、次の構成手順が必要です。構成手順の詳細は、14-8 ページの「Java SSO の設定と構成」に記載されています。

1. Application Server Control を使用して `javasso` アプリケーションが起動されている必要があります。
2. ターゲット OC4J インスタンス内で、`jazn.xml` ファイルに Java SSO とアイデンティティ管理フレームワーク実装のプロパティが正しく設定されている必要があります。Application Server Control を使用してプロパティを設定できます。これにより、対称鍵も生成できます。設定は、`<jazn>` 要素の `<property>` サブ要素に示されています。次の設定があります（デフォルト設定で十分です）。
   - Java SSO のログイン URL およびログアウト URL
   - Cookie トークンの暗号化に使用される対称鍵
   - Java SSO Cookie が限定される DNS ドメイン（複数の OC4J インスタンスとホストにわたる拡張インストールの場合に必要）

   （14-19 ページのトラブルシューティングの項の「資格証明が正しいにもかかわらず、Java SSO ログイン・ページに戻る」に記載されているように、必ず Cookie ドメインを適切に設定してください。）
3. Java SSO を使用するすべてのパートナ・アプリケーションは、`javasso` アプリケーションと同じセキュリティ・プロバイダとアイデンティティ・ストアを使用するよう構成されている必要があります。（特に、ファイルベース・プロバイダを使用する場合、同じリポジトリ・ファイルを使用します。複数の OC4J インスタンスを使用するシナリオは、14-16 ページの「ファイルベース・プロバイダと 2 つの OC4J インスタンスを使用する場合の考慮事項」を参照してください。）デフォルトでは、`javasso` はファイルベース・プロバイダを使用するよう構成されていますが、Application Server Control コンソールを使用すると、この構成を変更できます。

   カスタマ・アプリケーションであるパートナ・アプリケーションに対しては、デプロイメント中にセキュリティ・プロバイダを構成できます。Application Server Control 自体に対しては、Application Server Control コンソールでセキュリティ・プロバイダを変更する特別な手順があります。含めたい他の Oracle Application Server コンソール・アプリケーションに対して変更が必要な場合、同じように処理すると `javasso` 用にセキュリティ・プロバイダを変更できます。
4. 各パートナ・アプリケーションは、Java SSO が使用できるように有効になっている必要があります。これは、Application Server Control を使用して構成することもできます。Java SSO を使用できるように有効にするには、`<jazn-web-app>` 要素の `auth-method="CUSTOM_AUTH"` 設定と、`orion-application.xml` の `<jazn>` のサブ要素設定で指定します。

注意： Java SSO は、OracleAS JAAS Provider によってサポートされるすべてのプロバイダをサポートします。

Java SSO ログイン・ページとエラー・ページの概要

ここでは、Java SSO ログイン・ページとエラー・ページのローカライゼーションとカスタマイズについて説明します。

Java ログイン・ページとエラー・ページのローカライゼーション・サポート

Java SSO ログイン・ページとエラー・ページには、OC4J サポートのローカライゼーションが付属しています。これらのページで表示されるコンテンツは、ブラウザの設定に従ってローカライズされます。

ログイン・ページまたはエラー・ページのカスタマイズ

カスタム・ログイン・ページまたはカスタム・エラー・ページと Java SSO との併用は直接はサポートされていませんが、デプロイされた `login.jsp` ファイル、`loginerror.jsp` ファイル、`error.jsp` ファイルをカスタマイズし、次のいずれかの処理を実行することができます。

- 次のデプロイメント・ディレクトリで既存のファイルを置き換えます。

  ORACLE_HOME`/j2ee/home/applications/javasso/javasso-web/WEB-INF`

- カスタマイズされたファイルを使用して `javasso.ear` ファイルをパッケージしなおし、再度デプロイします。

Java SSO の設定と構成

ここでは、OC4J の Java SSO 構成について説明します。この項の内容は次のとおりです。

- Application Server Control を使用した Java SSO の構成
- Java SSO 構成プロパティ
- Java SSO に対してパートナ・アプリケーションを有効にする構成
- 特殊なシナリオ用の構成

Application Server Control を使用して Java SSO とパートナ・アプリケーションを構成することをお薦めします（最初の項を参照）。2 番目の項と 3 番目の項には、結果として設定される構成パラメータとプロパティが記載されています（必要に応じて手動で設定することもできます）。最後の項には、特別な考慮事項とシナリオが記載されています。

Application Server Control を使用した Java SSO の構成

Application Server Control コンソールでは、「Java SSO 構成」ページを使用します。このページにナビゲートするには、次の手順を実行します。

- Oracle Application Server のクラスタ環境で、「クラスタ・トポロジ」ページから**「Java SSO 構成」**を選択します。
- OC4J スタンドアロン環境で、次の手順を実行します。

  1. OC4J ホームページで**「管理」**タブを選択します。
  2. 「管理」ページの「プロパティ」から、「SSO 構成」タスクに移動します。

この項の以降の部分で、次の項目について説明します。

- javasso アプリケーションの起動
- Java SSO プロパティの設定と対称鍵の生成
- javasso アプリケーション用セキュリティ・プロバイダの構成
- パートナ・アプリケーション用セキュリティ・プロバイダの構成
- Java SSO を使用するパートナ・アプリケーションの有効化

javasso アプリケーションの起動

Oracle Application Server クラスタ環境の場合、「クラスタ・トポロジ」ページから、必要に応じて任意の OC4J インスタンスの javasso アプリケーションを起動できます。

1. 「メンバー」（アプリケーション・サーバーと OC4J インスタンスがリストされている）から**「すべてを開く」**リンクを選択し、インスタンスごとにすべてのアプリケーションを表示します。
2. 該当するインスタンスの javasso アプリケーションを選択します。
3. **「起動」**を選択します。

OC4J スタンドアロン環境で javasso アプリケーションを起動するには、次の手順に従います。

1. OC4J ホームページで**「アプリケーション」**を選択します。
2. 「アプリケーション」ページで、javasso アプリケーションを選択します。
3. **「起動」**を選択します。

Java SSO プロパティの設定と対称鍵の生成

「Java SSO 構成」ページで**「インスタンスおよびプロパティ」**タブを選択し、Java SSO プロパティを設定します。これにより、新しい対称鍵も生成できます。

「インスタンスおよびプロパティ」ページから、次の項目を指定できます。

- キー・タイプ。AES_128_CBC、AES_192_CBC、AES_256_CBC、DES_EDE_CBC のいずれかを選択します。この情報は、custom.sso.key.alias Java SSO プロパティにコード化されます（14-12 ページの「Java SSO 構成プロパティ」を参照）。
- Java SSO の SSOLogin サーブレットの URL（custom.sso.url.login プロパティに対応）。複数のホストが存在する場合、完全修飾ドメイン名を使用します。
- Java SSO の SSOLogout サーブレットの URL（custom.sso.url.logout プロパティに対応）。複数のホストが存在する場合、完全修飾ドメイン名を使用します。
- 秒単位のセッション・タイムアウト（custom.sso.session.timeout プロパティに対応）。これは非アクティブ・タイムアウトではなく、ハード・タイムアウトです。どのような場合でも、セッションはこの時間が経過するとタイムアウトします。
- 許可されるログインの試行回数（custom.sso.login.attempts プロパティに対応）。

Java SSO が適切に構成されていない場合、前述のパラメータを必要に応じて設定した後、**「Java SSO の構成」**を使用して Java SSO を構成します。これにより、新しい対称鍵が自動的に生成されます。

Java SSO が適切に構成された後で再構成する場合、再構成の適用時に新しい対称鍵を生成するかどうかを指定するチェック・ボックスを使用します。次のいずれかの状況の場合は、新しいキーを生成する必要があります。

- 異なる OC4J インスタンスが異なるキーを使用している可能性があるエラー状態が発生した場合。**すべてのインスタンスで同じキーを使用する必要があります。**
- セキュリティ上の理由から慎重を期すために新しいキーを使用する場合。（定期的にキーを新しくすることをお薦めします。）

前述のパラメータを必要に応じて設定した後、**「適用」**（**「Java SSO の構成」**のかわりに表示される）を選択して再構成を適用し、キーを生成します。

設定は、jazn.xml ファイル内の <jazn> 要素下の <property> サブ要素に反映されます。クラスタ内では、設定とキーはすべての OC4J インスタンスに適用されます。

重要：

- 10.1.3.1 のパッチ・セットを使用している場合、10.1.3.1 のフレッシュ・インストールとは異なり、14-14 ページの「単一インスタンス OC4J インストール用デフォルト Java SSO プロパティ設定」に記載されているデフォルトの jazn.xml 設定は存在しません。これらの設定は、Java SSO を構成する前に手動で追加する必要があります。
- デフォルトでは、Java SSO を構成するまで、jazn.xml ファイルにキーのダミー値が保存されます（custom.sso.key.alias 設定内）。
- 同じ Java SSO を使用するように構成されているアプリケーションは、すべて 1 つのキーを共有します。（これは、複数の OC4J インスタンスを使用するシナリオでのみ問題が発生する可能性があります。）
- 変更を有効にするには、OC4J インスタンスを再起動する必要があります。再起動を求める通知が表示されます。
- Oracle Identity Management を使用する場合、OC4J のインスタンスを Oracle Internet Directory のインスタンスに関連付けた後、Java SSO の設定を作成します。この関連付けを行うと、OC4J の home インスタンスの jazn.xml ファイル内にある <jazn> 要素の構成が書き換えられてしまうため、それまでの設定はすべて失われます。

javasso アプリケーション用セキュリティ・プロバイダの構成

使用される javasso アプリケーションと、Java SSO を使用する各パートナ・アプリケーションは、すべて同じセキュリティ・プロバイダを使用するように構成されている必要があります。デフォルトでは、javasso はファイルベース・プロバイダを使用するように構成されています。

特定の OC4J インスタンスで javasso のセキュリティ・プロバイダを変更するには、Application Server Control コンソールでそのインスタンスの OC4J ホームページに移動します。スタンドアロン環境では、OC4J ホームページは 1 つのみです。Oracle Application Server クラスタ環境では、「クラスタ・トポロジ」ページで該当する OC4J インスタンスを選択します。

1. OC4J ホームページで**「管理」**タブを選択します。
2. 「管理」ページの「セキュリティ」から、「セキュリティ・プロバイダ」タスクに移動します。
3. 「セキュリティ・プロバイダ」ページで、javasso の「編集」タスクに移動します。
4. javasso の「セキュリティ・プロバイダ」ページで、**「セキュリティ・プロバイダの変更」**を選択します。

注意：

- セキュリティ・プロバイダの変更を有効にするには、アプリケーションを再起動する必要があります。
- 複数の OC4J インスタンスが存在し、一部の javasso アプリケーションが同じセキュリティ・プロバイダを使用していない場合、Application Server Control コンソールの Java SSO の「インスタンスおよびプロパティ」ページには、各 OC4J インスタンスの javasso アプリケーションに関連する**「セキュリティ・プロバイダの変更」**オプションが表示されます。このオプションを使用すると、javasso アプリケーションを構成するプロセスが単純になります。

各セキュリティ・プロバイダ・タイプには、それを構成するための独自のタスク・セットが必要です。タスクの詳細は、次を参照してください。

- 7-3 ページの「デプロイ後のファイルベース・プロバイダへの変更」
- 8-15 ページの「デプロイ後の Oracle Identity Management への変更」
- 9-18 ページの「デプロイ後のカスタム・セキュリティ・プロバイダへの変更」
- 10-6 ページの「デプロイ後の外部 LDAP プロバイダへの変更」

パートナ・アプリケーション用セキュリティ・プロバイダの構成

前述のように、すべてのパートナ・アプリケーションと `javasso` アプリケーションは、同じセキュリティ・プロバイダを使用するように構成されている必要があります。パートナ・アプリケーションであるカスタマ・アプリケーションに対しては、デプロイメント中に適切なセキュリティ・プロバイダを構成します。各セキュリティ・プロバイダ・タイプには、それを構成するための独自のタスク・セットが必要です。タスクの詳細は、次を参照してください。

- 7-3 ページの「アプリケーション・デプロイ時のファイルベース・プロバイダの構成」
- 8-14 ページの「デプロイ時の Oracle Identity Management の指定」
- 9-16 ページの「デプロイ時のカスタム・セキュリティ・プロバイダの指定および構成」
- 10-3 ページの「デプロイ時の外部 LDAP プロバイダの指定および構成」

Java SSO の使用で Oracle Application Server のコンソール・アプリケーションを共有する場合（Application Server Control など）、コンソール・アプリケーションが同じセキュリティ・プロバイダを使用していることも確認する必要があります。

Application Server Control の場合、次の手順に従ってセキュリティ・プロバイダを変更できます。

1. Application Server Control を実行しているインスタンスの OC4J ホームページで、**「管理」**を選択します。
2. 「管理」ページの「セキュリティ」から、「セキュリティ・プロバイダ」タスクに移動します。
3. 「セキュリティ・プロバイダ」ページで、**「Application Server Control のセキュリティ」**を選択します。
4. 最初の「設定」ページで、**「セキュリティ・プロバイダ」**を選択します。
5. 「セキュリティ・プロバイダ」ページで、**「セキュリティ・プロバイダの変更」**を選択します。
6. 「セキュリティ・プロバイダの変更」ページで、`system-jazn-data.xml` ファイルベース・プロバイダ、アプリケーション固有 `jazn-data.xml` ファイルベース・プロバイダ、Oracle Identity Management（8-7 ページの「Oracle Internet Directory と OC4J の関連付け」に記載されているように、Oracle Internet Directory のインスタンスが事前に OC4J のインスタンスに関連付けられている場合）のどれを使用するかを指定します。
7. **「OK」**を選択します。変更を有効にするには、Application Server Control を再起動する必要があります。

他のコンソール・アプリケーションの場合、セキュリティ・プロバイダを変更する手順は、前の項の「javasso アプリケーション用セキュリティ・プロバイダの構成」で説明した `javasso` の場合に似ています。この場合、セキュリティ・プロバイダの変更を有効にするには、アプリケーションを再起動する必要があります。

Java SSO を使用するパートナ・アプリケーションの有効化

「Java SSO 構成」ページで**「関連アプリケーション」**タブを選択し、Java SSO を使用できるようにアプリケーションを有効にします。

「関連アプリケーション」ページで、次の手順を実行します。

1. 該当する各アプリケーションに対して、**「Java SSO 有効」**を選択します。
2. **「適用」**を選択し、Java SSO を使用できるようにアプリケーションを有効にします。

これにより各アプリケーションに対して、`orion-application.xml` ファイルの `<jazn-web-app>` 要素に `auth-method="CUSTOM_AUTH"` が設定されます。

`javasso` アプリケーションを起動し、Java SSO を使用するようアプリケーションを有効にしてからアプリケーションを起動すると、Java SSO を介してログイン・プロンプトが表示されます。

重要：

- Oracle Identity Management をアプリケーションのセキュリティ・プロバイダとして使用する場合、Java SSO に対してアプリケーションを有効にする前に、セキュリティ・プロバイダを構成する必要があります。Application Server Control を介してファイルベース・プロバイダから Oracle Identity Management に切り替えると、タイミングやパートナ・アプリケーションの種類に関係なく、そのアプリケーションの `orion-application.xml` 内にある `<jazn>` 要素が次のように置き換えられます。この場合、Java SSO に対してアプリケーションを有効にする以前の設定（`CUSTOM_AUTH` 設定）が失われるため、再設定が必要になります。

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

- 変更を有効にするには、OC4J インスタンスを再起動する必要があります。再起動を求める通知が表示されます。
- 複数のホストが存在する状態でブラウザからパートナ・アプリケーションを起動する場合は、完全修飾ドメイン名を使用します。

Java SSO 構成プロパティ

通常の場合、これまで説明したように、Application Server Control を使用して Java SSO を構成することをお薦めします。

ここでは、すべての Java SSO 構成プロパティと、OC4J インストール後のデフォルト設定について説明します。

Java SSO 構成プロパティの説明

表 14-1 は、Java SSO プロパティを示しています。プロパティの設定は、`jazn.xml` ファイル内の `<jazn>` 要素の `<property>` サブ要素に反映されます。

Application Server Control でサポートされないプロパティ（表中で「なし」となっているもの）は、手動で設定する必要があります。

関連項目：

- 14-9 ページの「Java SSO プロパティの設定と対称鍵の生成」

表 14-1 Java SSO プロパティ

プロパティ	説明	Application Server Control での名前
custom.sso.app.url.default	指定されていない場合、`SSOLogin` サーブレットからのデフォルトのリダイレクション URL。	なし
custom.sso.cookie.domain	Java SSO の Cookie が、この DNS ドメインに限定されます。デフォルトでは、ドメイン設定がない場合、Cookie は `SSOLogin` URL にアクセスするホストに限定されます。ドメイン設定は、複数の OC4J インスタンスとホストにわたる拡張インストールの場合に必要です。 (14-19 ページのトラブルシューティングの項の「資格証明が正しいにもかかわらず、Java SSO ログイン・ページに戻る」に記載されているように、必ず Cookie ドメインを適切に設定してください。)	なし
custom.sso.cookie.path	Cookie に対するパスの制限。 デフォルト : /	なし
custom.sso.cookie.secure	`true` の場合、HTTPS サイトのみがサポートされます。 デフォルト : `false`	なし
custom.sso.key.alias	Java SSO の Cookie アイデンティティ・トークンの内容に署名する際に使用される共有対称鍵を格納します。	なし
custom.sso.login.attempts	許可されるログイン試行回数。 デフォルト : 3	ログイン試行回数
custom.sso.session.timeout	Java SSO の Cookie の有効時間(秒)。これは非アクティブ・タイムアウトではなく、ハード・タイムアウトです。どのような場合でも、セッションはこの時間が経過するとタイムアウトします。 デフォルト : 7200	セッション・タイムアウト
custom.sso.token.asserter.authtypes	Java SSO の認証方式。エンド・ユーザーが Java SSO にリダイレクトされた場合に、この認証方式を使用してエンド・ユーザーを認証します。標準的な J2EE Web アプリケーション認証方式を指定できます。 デフォルト : `FORM`	なし
custom.sso.url.login	`SSOLogin` サーブレットの URL。複数のホストが存在する場合、完全修飾ドメイン名を使用します。	Java SSO ログイン URL
custom.sso.url.logout	`SSOLogout` サーブレットの URL。複数のホストが存在する場合、完全修飾ドメイン名を使用します。	Java SSO ログアウト URL

重要： OC4J インスタンスを Oracle Internet Directory インスタンスに関連付けると、OC4J home インスタンスの jazn.xml ファイル内にある <jazn> 要素構成は再度書き込まれます。以前の設定は失われます。

単一インスタンス OC4J インストール用デフォルト Java SSO プロパティ設定

デフォルトでは、単一 OC4J インスタンスのインストールの場合、アイデンティティ管理フレームワーク実装として Java SSO を使用するように OC4J が設定されます。これには、前の項の「Java SSO 構成プロパティの説明」で説明した Java SSO プロパティの設定と、13-14 ページの「アイデンティティ管理フレームワーク・プロパティの構成」で説明したアイデンティティ管理フレームワーク・プロパティの Java SSO 設定が含まれます。

jazn.xml に事前に構成されている設定は、次のとおりです。

```
<jazn provider="XML" location="./system-jazn-data.xml" default-realm="jazn.com">
   <!-- properties to configure the 3rd party IDM framework -->
   <property name="idm.authentication.name" value="JavaSSO" />
   <property name="idm.token.asserter.class"
             value="oracle.security.jazn.sso.SSOCookieTokenAsserter" />
   <property name="idm.token.collector.class"
             value="oracle.security.jazn.sso.SSOCookieTokenCollector" />
   <property name="idm.token.type" value="HTTP_COOKIE" />
   <property name="idm.token.collector.cookie.1" value="ORA_OC4J_SSO" />

   <!-- properties for the out of the box Java SSO -->
   <property name="custom.sso.url.login" value="/jsso/SSOLogin" />
   <property name="custom.sso.url.logout" value="/jsso/SSOLogout" />
   <property name="custom.sso.key.alias" value="ssoSymmetricKey" />
</jazn>
```

重要：

- デフォルトでは、一部のシナリオでこれらの設定が存在しません（14-17 ページの「複数 OC4J インスタンスの場合の一般的な考慮事項」および 14-17 ページの「10.1.3.0.0 に 10.1.3.1 パッチを適用する場合の考慮事項」を参照）。この場合、Java SSO を構成する前に、デフォルト構成を jazn.xml に手動で追加します。
- デフォルトでは、Application Server Control を介して Java SSO を構成するまで、前述の custom.sso.key.alias 設定のキーにダミー値が保存されます（14-9 ページの「Java SSO プロパティの設定と対称鍵の生成」を参照）。Java SSO を構成すると、設定は次のように変更されます。

  value="{AES-128}IdG4OPSqGPJ8hZFPn1W4Uw=="

Java SSO に対してパートナ・アプリケーションを有効にする構成

Java SSO を使用する各パートナ・アプリケーションは、Java SSO を有効にするように構成されている必要があります。この場合、Application Server Control を使用することをお薦めします（14-12 ページの「Java SSO を使用するパートナ・アプリケーションの有効化」を参照）。

Java SSO を使用するパートナ・アプリケーションの有効化は、`orion-application.xml` ファイルの `<jazn-web-app>` 要素内にある認証方式の設定 `"CUSTOM_AUTH"` で指定されます。次に例を示します。

```
<jazn provider="XML" ... >
   ...
   <jazn-web-app auth-method="CUSTOM_AUTH" />
   ...
</jazn>
```

これにより、Java SSO の使用とアイデンティティ管理フレームワークの実装が、`jazn.xml` の構成に基づいてトリガーされます。

重要： Oracle Identity Management をアプリケーションのセキュリティ・プロバイダとして使用する場合、Java SSO に対してアプリケーションを有効にする前に、セキュリティ・プロバイダを構成する必要があります。Application Server Control を介してファイルベース・プロバイダから Oracle Identity Management に切り替えると、タイミングやパートナ・アプリケーションの種類に関係なく、そのアプリケーションの `orion-application.xml` 内にある `<jazn>` 要素が次のように置き換えられます。この場合、Java SSO に対してアプリケーションを有効にする以前の設定（`CUSTOM_AUTH` 設定）が失われるため、再設定が必要になります。

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

注意：

- OC4J デフォルト構成によって Java SSO がアイデンティティ管理フレームワーク実装として指定されているため、アイデンティティ管理フレームワーク（`auth-method="CUSTOM_AUTH"`）を使用するようにパートナ・アプリケーションを有効にすると、このパートナ・アプリケーションは Java SSO を使用するように有効化されます。
- `<jazn-web-app>` 要素も、特定の Web アプリケーションに対して、`<orion-web-app>` のサブ要素として `orion-web.xml` ファイルでサポートされます。この設定は、Web アプリケーションの `orion-application.xml` 設定よりも優先されます。
- `orion-application.xml` または `orion-web.xml` 内の認証方式設定は、`web.xml` 内の認証方式設定よりも優先されます。

関連項目：

- 13-16 ページの「アイデンティティ管理フレームワークを使用するアプリケーションの構成」

特殊なシナリオ用の構成

これまでは Java SSO の一般的な構成について説明しましたが、ここでは次のような特殊な考慮事項について説明します。

- ファイルベース・プロバイダと 2 つの OC4J インスタンスを使用する場合の考慮事項
- 複数 OC4J インスタンスの場合の一般的な考慮事項
- 10.1.3.0.0 に 10.1.3.1 パッチを適用する場合の考慮事項

ファイルベース・プロバイダと 2 つの OC4J インスタンスを使用する場合の考慮事項

ここでは、SOA インストールなど、2 つの OC4J インスタンスにわたるシングル・サインオン・アクセス用に Java SSO とファイルベース・プロバイダを採用する場合の使用例について説明します。ここで重要になるのは、両方の OC4J インスタンスに対して、2 つの `system-jazn-data.xml` ファイル内のユーザー・アカウントとロールを同期化することです。

次に使用例を示します。

- OC4J インスタンスとして、`OC4J_HOME` と `OC4J_SOA` を作成しました（インストーラを使用して作成、または `createinstance` を使用して手動で作成）。
- `OC4J_HOME` で Java SSO をアクティブにし、`OC4J_SOA` に SOA アプリケーションをデプロイします。
- `OC4J_HOME` と `OC4J_SOA` の両方で、ファイルベース・プロバイダを使用します。

デフォルトでは、各 OC4J インスタンスは、それ自体のファイルベース・プロバイダである `system-jazn-data.xml` ファイルを使用して構成されます。ただし、両方のインスタンスで Java SSO を適切に機能させるには、ユーザー・アカウントとユーザー・ロールが 2 つの `system-jazn-data.xml` ファイル間で同期化されている必要があります。このためには、OC4J グループ機能を使用して次の手順を実行します。

1. 例として、Java SSO の OC4J グループ `JSSO_GROUP` を作成します。7-18 ページの「OC4J の基本的なグループ機能」を参照してください。
2. Java SSO アプリケーションがファイルベース・プロバイダ `system-jazn-data.xml` を使用するように構成されていることを確認した後（デフォルト設定）、`OC4J_HOME` で Java SSO アプリケーションを起動します。（この使用例では、1 つの OC4J インスタンスのみで Java SSO アプリケーションを起動します。）
3. Java SSO を使用する J2EE アプリケーションを、`OC4J_SOA` インスタンスにデプロイします。ファイルベース・プロバイダを使用するように各アプリケーションを構成し、各アプリケーションに対して Java SSO を有効にします。
4. `OC4J_HOME` と `OC4J_SOA` を、`JSSO_GROUP` のメンバーとして追加します。「OC4J の基本的なグループ機能」を参照してください。

これらのアプリケーションのユーザーとロールを引き続き管理する場合、`JSSO_GROUP` に対応する OC4J `J2EEServerGroup` MBean を使用し、適切なセキュリティ・プロバイダ MBean 操作を実行します。この MBean により、`JSSO_GROUP` メンバー間でユーザー・アカウントとロールが同期化されます。7-19 ページの「クラスタ MBean ブラウザの機能および J2EEServerGroup MBean」を参照してください。

注意： 2 つのインスタンスの `system-jazn-data.xml` ファイル間でユーザー構成を手動で調整しても、同じ結果が得られます。ただし、エラーが発生する可能性が高くなります。

関連項目：

- OC4J グループ機能の追加情報は、Application Server Control オンライン・ヘルプのグループ OC4J インスタンスのページに関するトピックを参照してください。

複数 OC4J インスタンスの場合の一般的な考慮事項

複数の OC4J インスタンスを使用するインストール・タイプの場合、14-14 ページの「単一インスタンス OC4J インストール用デフォルト Java SSO プロパティ設定」で説明するデフォルト構成は適していません。プロパティは `jazn.xml` 内で参照できますが、設定はされていません。

Java SSO のすべての構成およびアイデンティティ管理フレームワークの関連する構成は、インスタンス間で複製されている必要があります。主な問題は、`system-jazn-data.xml` 設定に加えて、`jazn.xml` 内のプロパティやアイデンティティ管理フレームワークのプロパティが Application Server Control によって設定されないということです。共有 Java SSO キーと Application Server Control によって設定された Java SSO のプロパティは、Application Server Control 機能によってすべての OC4J インスタンスに対してすでに伝播されています。

必要に応じて次の手順を実行します。

1. Java SSO の設定と構成を指定します（14-8 ページの「Application Server Control を使用した Java SSO の構成」を参照）。Application Server Control によって設定された Java SSO プロパティがインスタンスに適用されることに注意してください。
2. Application Server Control によってサポートされない Java SSO プロパティや、関連するアイデンティティ管理フレームワークの追加プロパティなど、`jazn.xml` 内の関連する他のプロパティ設定を OC4J インスタンス間で複製します。通常、このようなプロパティに対しては、各 OC4J インスタンスの `jazn.xml` ファイル内にある構成を手動で複製するのが唯一の方法です。
3. OC4J グループ機能を使用して、グループの `system-jazn-data.xml` を構成します。（これは、ファイルベース・セキュリティ・プロバイダ、ログイン・モジュール設定、ポリシー設定などの場合です。）OC4J グループは、7-18 ページの「OC4J の基本的なグループ機能」を参照してください。また、7-19 ページの「クラスタ MBean ブラウザの機能および J2EEServerGroup MBean」では、OC4J インスタンス間で `system-jazn-data.xml` のユーザー設定を調整する方法について説明しています。インスタンス間でログイン・モジュール構成をメンテナンスする操作もあります（`setLoginModule` など）。

関連項目：

- OC4J グループ機能の追加情報は、Application Server Control オンライン・ヘルプのグループ OC4J インスタンスのページに関するトピックを参照してください。

10.1.3.0.0 に 10.1.3.1 パッチを適用する場合の考慮事項

既存の 10.1.3.0.0 インストールに対して OC4J 10.1.3.1 パッチをインストールする場合、10.1.3.1 フレッシュ・インストールとは異なり、14-14 ページの「単一インスタンス OC4J インストール用デフォルト Java SSO プロパティ設定」で説明するデフォルト構成は適していません。これは、`jazn.xml` のプロパティが参照されないためです。

この場合、デフォルト構成を手動で `jazn.xml` に追加し、14-8 ページの「Application Server Control を使用した Java SSO の構成」で説明する適切な Java SSO 構成を実行する必要があります。

Java SSO API

Java SSO には、次のユーティリティ・クラスが用意されています。

`oracle.security.jazn.sso.util.JSSOUtil`

OC4J 10.1.3.1 実装では、このクラスは次の API を提供します。

- Java SSO ログアウト API

関連項目：

- Java SSO API を含む Javadoc は、『Oracle Containers for J2EE Security Java API Reference』を参照してください。

Java SSO ログアウト API

Java SSO に付属する `JSSOUtil` ユーティリティ・クラスは、静的 `logout()` メソッドを提供します。アプリケーション固有のログアウトの準備と処理が完了したら、このメソッドはアプリケーション・セッション・ログアウトの最後の手順として使用されます。

- `logout(HttpServletResponse response, String targetURL)`

  処理中のリクエストに対応する HTTP レスポンス・オブジェクトを指定し、ログアウト後にアプリケーションがリダイレクトされる URL を指定します。アプリケーションは、最初に Java SSO の `SSOLogout` サーブレットにダイレクトされ、ここで Java SSO Cookie の設定が解除されてから、指定したターゲット URL にダイレクトされます。

次に例を示します。

```
JSSOUtil.logout(response, "http://portal.acme.com");
```

Java SSO の使用方法の概要

ここでは、前述した Java SSO を使用する場合の重要なアクションを要約します。

開発においては、Java SSO を使用することがほとんどの場合で透過的であり、特別なコーディングを必要としません。統合の 1 つのポイントはログアウト API です（前項の「Java SSO ログアウト API」を参照）これは、オプションとして使用できます。

構成する場合、次の手順を実行します。詳細は、14-8 ページの「Java SSO の設定と構成」を参照してください。Application Server Control を使用して、すべての手順を実行することができます。

1. `javasso` アプリケーションを起動します。
2. Java SSO プロパティを構成します。（設定は、`jazn.xml` 内の `<jazn>` 要素下の `<property>` 要素に書き込まれます。）
3. 適切なセキュリティ・プロバイダを使用するように `javasso` アプリケーションを構成します。
4. `javasso` と同じセキュリティ・プロバイダを使用するようにパートナ・アプリケーションを構成します。カスタマ・アプリケーションの場合、デプロイメント中にセキュリティ・プロバイダを設定します。Java SSO で共有する Oracle Application Server のコンソール・アプリケーションの場合（Application Server Control など）、セキュリティ・プロバイダを適切に変更できます。
5. Java SSO を使用するように各パートナ・アプリケーションを有効にします。（これにより、アプリケーションの `orion-application.xml` ファイル内に `auth-method="CUSTOM_AUTH"` が設定されます。）これは、既存アプリケーションを Java SSO 対応に変換するために必要な唯一の構成手順です。`web.xml` ファイルの `<login-config>` 設定は、変更する必要がありません（上書きされます）。

重要： 複数のホストが存在する状態でブラウザからパートナ・アプリケーションを起動する場合は、完全修飾ドメイン名を使用します。

Java SSO のトラブルシューティング

ここでは、発生する可能性がある次の問題について説明します。

ページが見つからない

Java SSO に対して有効になっているパートナ・アプリケーションにアクセスしようとするとページが見つからないというエラーが発生する場合、`javasso` アプリケーションが実行されていない可能性があります。これをテストするには、`SSOLogin` サーブレットへのアクセスを行います。次の URL に直接移動できます。

`http://`*`host`*`:`*`port`*`/jsso/SSOLogin`

または、Java SSO プロパティを構成する Application Server Control のページから**「ログイン URL のテスト」**を選択することもできます。

最上位ページの *`host`*`:`*`port`* にリダイレクトされれば、テストは成功です。

`javasso` アプリケーションを起動する手順は、14-9 ページの「javasso アプリケーションの起動」に記載されています。

資格証明が正しいにもかかわらず、Java SSO ログイン・ページに戻る

Java SSO ログイン・ページで正しい資格証明を指定したにもかかわらずログイン・ページに戻された場合、次のことが考えられます。

- アクセスを許可されていない `localhost` または IP アドレスを使用して、パートナ・アプリケーションへアクセスしようとしたため。
- `javasso` アプリケーションに指定したドメインと一致しないパートナ・アプリケーションにアクセスしようとしたため。たとえば、Application Server Control によって設定された Java SSO ログイン URL（および `custom.sso.url.login` プロパティに対応するログイン URL）が `http://sso.mydomain.com/jsso/SSOLogin` として設定されている場合、`http://sso/myapp/index.jsp` と指定するとアプリケーションにアクセスできません。
- Cookie ドメイン設定（`custom.sso.cookie.domain` プロパティ）が、パートナ・アプリケーションにアクセスしようとしているドメインと一致しないため。たとえば、Cookie ドメインが `.sso.mydomain.com` に設定されている場合、`http://app.mydomain.com/myapp/index.jsp` と指定するとアプリケーションにアクセスできません。ただし、Cookie ドメインが `.mydomain.com` に設定されている場合はアクセスできます。

15

OC4J との SSL 通信

OC4J では、次のように Secure Sockets Layer（SSL）通信がサポートされます。

- Oracle Application Server 環境（OPMN による管理）では、Oracle HTTP Server と OC4J 間の SSL 通信が AJPS を使用してサポートされます。このプロトコルは、セキュアなバージョンの Apache JServ Protocol であり、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 AJPS プロトコルを、エンドユーザーは意識しないことに注意してください。）
- スタンドアロンの OC4J 環境、つまり Oracle HTTP Server を使用せずに OPMN で管理される環境では、クライアントと OC4J 間の直接的な SSL 通信が、HTTPS を使用してサポートされます。
- OC4J はまた、ORMI over SSL、または ORMIS もサポートします。この機能を使用して、OC4J は、複数の OC4J サーバー・インスタンス上のオブジェクト間の、SSL を介した RMI 通信をサポートしています。

この章では、スタンドアロンの OC4J のシナリオとともに、Oracle Application Server 環境での 2 つのシナリオとして、OC4J を Web リスナーとする OPMN 管理 OC4J と、Oracle HTTP Server を Web リスナーとする OPMN 管理 OC4J について説明します。この章の内容は次のとおりです。

注意：

- クライアントと Oracle HTTP Server 間の安全な通信は、Oracle HTTP Server と OC4J 間の安全な通信とは無関係です。
- 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。

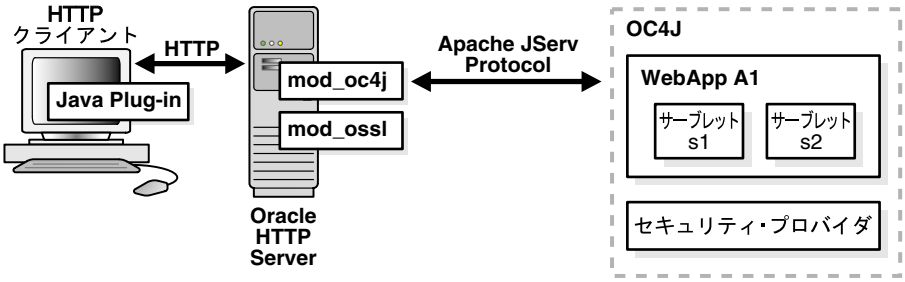
関連項目：

- 1-4 ページの「トランスポート・レベルのセキュリティ」（概要）
- Oracle HTTP Server での SSL の使用に関連する情報は、『Oracle HTTP Server 管理者ガイド』を参照してください。
- SSL を利用するための追加 Oracle Application Server コンポーネントの構成方法は、『Oracle Application Server 管理者ガイド』を参照してください。

セキュリティ・プロバイダと SSL 対応アプリケーションの統合

この項では、SSL 対応 J2EE 環境で HTTP クライアント・リクエストが開始された場合の、Oracle コンポーネントの役割について説明します。図 15-1 は、このような環境で動作するアプリケーションを示しています。

図 15-1 SSL 対応 J2EE 環境での Oracle コンポーネントの統合



このプロセスの手順は次のとおりです。

1. HTTP クライアントが、OC4J 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 がリクエストを処理します。
2. `mod_ossl`/Oracle HTTP Server は、リクエストを受信し、WebApp A1 アプリケーションが HTTP クライアントに対する SSL サーバー認証を要求しているかどうかを判別します。
3. サーバーまたはクライアントの Wallet 証明書を構成している場合は、HTTP クライアントに対して Oracle HTTP Server のサーバー証明書受入れおよびクライアント証明書の提供を要求します。
4. OC4J セキュリティ・プロバイダが、SSL クライアント証明書を取得します。
5. セキュリティ・プロバイダが、証明書から SSL ユーザーを取得します。
6. 最後の手順は、`<jazn>` 要素の `jaas-mode` 設定に応じて異なります。JAAS モードの使用方法を、5-6 ページの「JAAS モードの概要」および 5-20 ページの「JAAS モードの構成と使用」で確認してください。

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、一般に安全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、アプリケーション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 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。

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。

注意：

- 信頼関係を確立する際にキーストアまたは Wallet を使用するには、そのキーストアまたは Wallet に秘密鍵、公開鍵および信頼できる証明書（CA 証明書）が含まれている必要があります。
- OC4J 付属のデフォルト Wallet には、信頼できる証明書が含まれていません。デフォルトの Wallet のかわりに、独自の Wallet を作成してプロビジョニングする必要があります。

関連項目：

- `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
  ```

- 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ための OC4J での手順を示します。

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（VeriSign 社や Thawte 社など。この後にリンクを記載しています）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。

関連項目：

- 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 間で SSL 通信を行うには、Oracle HTTP Server で必要に応じて次の手順を実行する必要があります。

1. 前述の OC4J での手順と同様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。
2. 必要に応じて実行 : **OC4J の証明書に Oracle HTTP Server で信頼されていないエンティティの署名がある場合は、**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。
3. 必要に応じて実行 : **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、かつ 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は、次の操作を行います。**

   （これは 15-15 ページの「クライアント認証の要求」でも説明しています。）

   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。

スタンドアロン OC4J での SSL の使用

スタンドアロン OC4J では、HTTPS を使用して、クライアントと OC4J の間で直接、SSL 通信を行うことができます。この項では、これを行う方法について説明します。

次の手順に従います。

1. keytool ユーティリティを使用し、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、RSA 鍵ペア生成アルゴリズムを使用し、パスワードに 123456 を指定してファイル mykeystore.jks に属するキーストアを生成します。

   ```
   % keytool -genkey -keyalg RSA -keystore mykeystore.jks -storepass 123456
   ```

   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。コマンドラインでこのオプションを省略し、かわりにパスワードの入力が要求されるようにすることもできます。

   keytool により、次のように追加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=Redwood Shores, ST=CA, C=US> correct?
     [no]:  yes

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

   キー・パスワードに対しては、常に [Enter] キーを押します。OC4J 10.1.3.x 実装では、キーストアのパスワードは、キー・エントリのパスワードと同じである必要があります。

   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。

注意： keytool ユーティリティでは、JKS 形式のキーストアとともに、PKCS12 形式の Wallet がサポートされます。

関連項目：

- keytool ユーティリティの詳細は、次の URL を参照してください。

  http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html

- ISO の 2 文字の国コード・リストは、次の URL で参照してください。

  http://www.bcpl.net/~jspath/isocodes.html

2. secure-web-site.xml ファイルがない場合は、*ORACLE_HOME*/j2ee/home/config/secure-web-site.xml（J2EE の規約上、この名前を使用）を作成します。最初に default-web-site.xml から必要な内容をコピーすることもできます。通常、この内容には <web-site> 要素下の次のサブ要素が含まれます。

   - <web-app>（保護の対象となる Web アプリケーションごと）
   - <access-log>（ロギング用。適切なログ・ファイルが指定されていることを確認すること）
   - <default-web-app>

   SSL 構成には <ssl-config> 要素も必要です。これについては、この手順で後述します。

3. 次の要素を使用して、secure-web-site.xml を更新します。

   a. secure="true" を追加し、port を適切なポートに設定して、<web-site> 要素を更新します。（たとえば、port="4443" のように設定します。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。）スタンドアロン OC4J の場合は、デフォルトの設定である HTTP プロトコルを使用します。（protocol="http" 設定と secure="true" を組み合せることで、HTTPS が使用されます。）

   ```
   <web-site port="4443" secure="true" protocol="http"
             display-name="Default OracleAS Containers for J2EE Web Site" >
      ...
   </web-site>
   ```

   （必要に応じて、display-name 設定も変更する必要があります。）

   b. <web-site> 要素内に次を追加してキーストアとパスワードを定義します。

   ```
   <ssl-config keystore="your_keystore" keystore-password="your_password" />
   ```

   your_keystore は、キーストアのパスです。絶対パスまたは（Web サイトの XML ファイルが置かれている）*ORACLE_HOME*/j2ee/home/config に対する相対パスを指定します。*your_password* はキーストアのパスワードです。

   注意： パスワードは、パスワードの間接化によって非表示にできます。6-2 ページの「パスワードの間接化の使用方法」を参照してください。

   ---

   c. 次の「secure-web-site.xml におけるオプションの手順」も参照してください。

   d. 変更結果を secure-web-site.xml に保存します。

   次に例を示します。

   ```
   <?xml version="1.0"?>
   <web-site display-name="OC4J 10g Secure Web Site" protocol="http"
             port="4443" secure="true">
      <ssl-config keystore="./roadrunner.jks" keystore-password="welcome1" />
      <default-web-app application="default" name="defaultWebApp" root="/" />
      <web-app application="SSLDemos-Project1-WS" name="WebServices"
               load-on-startup="true" root="/SSLDemos-Project1-context-root" />
      <access-log path="../log/default-web-access2.log" split="day" />
   </web-site>
   ```

4. server.xml が secure-web-site.xml ファイルを参照しているかどうかを確認します。

   a. 必要に応じて、server.xml において次の行をコメント解除することで追加します。

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

   b. 変更結果を server.xml に保存します。

5. 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 の参照行をコメント解除すると、SSL リクエストまたは非 SSL リクエストを無効にできます。

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

これらの Web サイトは、異なるポートを使用する必要があります。

secure-web-site.xml におけるオプションの手順

前述の secure-web-site.xml 構成手順に加えて、次のオプションの手順も必要に応じて実行します。

1. <ssl-config> 要素の属性である needs-client-auth フラグを、次のように true に設定することで、クライアント認証を必須に指定します。

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

   この手順では、OC4J が安全な通信を行うため、クライアント・エンティティをその ID に応じて受け入れるか拒否するかを選択するモードを設定します。needs-client-auth 属性を設定すると、OC4J は接続時にクライアント証明連鎖をリクエストします。クライアントのルート証明書が認識されると、クライアントは受け入れられます。

   <ssl-config> 要素で指定するキーストアには、HTTPS を介して OC4J への接続の認可を受ける任意のクライアントの証明書が格納されている必要があります。

   重要： スタンドアロン OC4J（Oracle HTTP Server なし）で CLIENT-CERT 認証モードを使用するには、needs-client-auth="true" の設定が必要です。関連情報は、15-15 ページの「クライアント認証の要求」を参照してください。

2. Web サイトの各アプリケーションを、共有アプリケーションとして指定します。<web-app> 要素の shared 属性で、複数のバインディング（様々な Web サイトまたはポートおよびコンテキスト・ルート）を共有できるかどうかを指定します。サポートされる値は true および false（デフォルト）です。

   共有とは、セッション、サーブレット・インスタンスおよびコンテキスト値など、Web アプリケーションの構成要素すべてを共有することを意味します。通常このモードは、SSL が通信のすべてではなく一部で要求されている場合に、同じコンテキスト・パスの HTTP サイトと HTTPS サイトの間で Web アプリケーションを共有するために使用します。すべての情報ではなく重要な情報のみを暗号化することで、パフォーマンスが向上します。

HTTPS Web アプリケーションが shared に設定されている場合は、セッションの追跡には、SSL 証明書ではなく Cookie が使用されます。SSL 証明書は、追跡する各証明書の保存に 50K を使用して、セッションのタイムアウト前にメモリー不足の問題が発生する場合があるため、Cookie の方が利点があります。Cookie は Web アプリケーションのセキュリティを低下させますが、一部のブラウザで正常にサポートされていない、SSL セッションのタイムアウトなどの問題を回避するためには必要になります。

関連項目：

- Web サイト間の Web アプリケーションの共有については、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

3. デフォルト・ポートを使用せずに `shared="true"` を設定する場合は、Cookie ドメインを設定します。クライアントが異なるポートを介して Web サーバーと対話する場合は、Cookie は各ポートが独立した Web サイトを示すものとみなします。HTTP に対して 80、HTTPS に対して 443 の各デフォルト・ポートを使用する場合は、クライアントはこれらを同じ Web サイトの異なる 2 つのポートとみなし、1 つの Cookie のみを作成します。ただし、デフォルト以外のポートを使用する場合は、クライアントはこれらのポートを同一の Web サイトの独立した構成要素とみなし、Cookie ドメインを指定していない場合はポートごとに Cookie を作成します。

   Cookie ドメインは、DNS ドメイン内の複数のサーバーに対するクライアントの通信を追跡します。HTTP および HTTPS を使用する共有環境に対してデフォルト以外のポートを使用する場合は、アプリケーションの `orion-web.xml` ファイルの `<session-tracking>` 要素内で、`cookie-domain` 属性を設定します。`cookie-domain` 属性には、提供されている DNS ドメイン名のうち、最低 2 つのコンポーネントを指定します。

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

4. 使用する暗号スイートを指定します。暗号スイートは、セキュリティ・アルゴリズムと鍵のサイズを定義する複数の暗号化仕様を組み合せたものです。（これはサーバー側での暗号スイートの設定であり、第 16 章「クライアント接続用 Oracle セキュリティ」で説明する `HTTPClient` の設定とは異なります。）次の例のように、`secure-web-site.xml` 内の `<ssl-config>` 要素の `cipher-suites` 属性を指定します。

   ```
   <ssl-config keystore="your_keystore" keystore-password="your_password"
               cipher-suites="SSL_RSA_WITH_RC4_128_SHA,
                              SSL_RSA_WITH_RC4_128_MD5,..." />
   ```

   これは、暗号スイートをカンマで区切ったリストです。この属性を省略すると、次の URL にある参考ドキュメントで「enabled by default」として指定されている一連の暗号スイートが使用されます。

   ```
   http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/
   JSSERefGuide.html
   ```

例 15-1 クライアント認証での HTTPS 通信

次の例では、クライアント認証で HTTPS による安全な通信を実行できる Web サイトを構成しています。

```
<web-site display-name="OC4J Web Site" protocol="http" port="4443" secure="true" >
   <default-web-app application="default" name="defaultWebApp" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome"
               needs-client-auth="true" />
</web-site>
```

太字の部分のみが、セキュリティ関連の記述です。スタンドアロン OC4J での HTTP 通信の場合、protocol の値は安全な通信を使用するかどうかにかかわらず、常に http になります。protocol の値が http で secure="false" の場合は、HTTP プロトコルを示し、http で secure="true" の場合は、HTTPS プロトコルを示します。

needs-client-auth フラグを指定すると、OC4J は接続時にクライアント証明連鎖をリクエストします。OC4J がクライアントのルート証明書を認識すると、クライアントは受け入れられます。

<ssl-config> 要素で指定するキーストアは、HTTP および SSL を介して OC4J への接続の認可を受ける任意のクライアントの証明書が格納されている必要があります。

Oracle HTTP Server を使用しない OPMN 管理 OC4J での SSL の使用

OC4J が OPMN によって管理され、OC4J 自体が Web リスナーとして使用される場合（つまり Oracle HTTP Server を使用しない場合）は、HTTPS を使用してクライアントと OC4J 間の SSL 通信がサポートされます（前述のスタンドアロン OC4J のシナリオと同様）。HTTPS をサポートするには、OPMN も構成する必要があります。

この項では、この OPMN 管理のシナリオで SSL を使用する方法について説明します。これは次の手順で行います。

1. OC4J の SSL 用構成（Oracle HTTP Server を使用しないシナリオ）
2. HTTPS をサポートするための OPMN の構成（Oracle HTTP Server を使用しないシナリオ）

OC4J の SSL 用構成（Oracle HTTP Server を使用しないシナリオ）

Oracle HTTP Server を使用しない OPMN 管理環境において、SSL を使用できるように OC4J を構成する方法は、15-5 ページの「スタンドアロン OC4J での SSL の使用」で説明しているスタンドアロン OC4J の構成方法とほぼ同じです。詳細はこの項を参照してください。

1. キーストアを作成します。
2. secure-web-site.xml（規約上、この名前を使用）を作成します。（必要に応じて、default-web-site.xml から内容をコピーします。）
3. 次の要素を使用して、secure-web-site.xml を更新します。

   a. <web-site> 要素に secure="true" を追加して、この要素を更新します。通常、OPMN 管理環境では、port="0" 設定で指定されるように、OPMN に従ってポートが選択されます。protocol="http" を使用します。（protocol="http" 設定と secure="true" を組み合せることで、HTTPS が使用されます。）

   ```
   <web-site port="0" secure="true" protocol="http"
             display-name="Default OracleAS Containers for J2EE Web Site" >
      ...
   </web-site>
   ```

   （必要に応じて、display-name 設定も変更する必要があります。）

   b. <web-site> の下に <ssl-config> 要素を追加し、keystore と keystore-password 属性を使用して、キーストアの場所とパスワードを定義します。

   c. 変更結果を secure-web-site.xml に保存します。

   次に例を示します。

```
<web-site display-name="OC4J Web Site" protocol="http" port="0"
          secure="true" >
   <default-web-app application="default" name="defaultWebApp" root="/j2ee" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome" />
</web-site>
```

4. server.xml が、secure-web-site.xml を参照しているかどうかを確認します。
5. OC4J を停止してから再起動し、secure-web-site.xml を初期化します。

注意：

- この Web サイトの XML ファイルの設定をオーバーライドしないように OPMN を構成するには、ここでポート 0 のかわりに、実際に使用するポートを指定します。
- Oracle HTTP Server を使用しない OC4J 環境で CLIENT-CERT 認証モードを使用するには、needs-client-auth="true" 設定が必要です。関連情報は、15-15 ページの「クライアント認証の要求」を参照してください。

HTTPS をサポートするための OPMN の構成（Oracle HTTP Server を使用しないシナリオ）

Oracle HTTP Server を使用しない OPMN 管理環境の OC4J で SSL を使用するには、HTTPS をサポートするように OPMN を構成する必要があります。ファイル *ORACLE_HOME*/opmn/conf/opmn.xml を、次のように更新します。

1. コンポーネント ID OC4J 下で、セキュリティ・パラメータを構成します（Wallet 情報）。

```
<ias-component id="OC4J">
   ...
   <category id="security-parameters">
      <data id="wallet-file" value="file:walletfile"/>
      <data id="wallet-password" value="pwd"/>
   </category>
   ...
</ias-component>
```

2. 同じくコンポーネント ID OC4J 下で、Web サイトに対して HTTPS プロトコルを指定します。

```
<ias-component id="OC4J">
   ...
   <port id="secure-web-site" range="12501-12600" protocol="https"/>
   ...
</ias-component>
```

関連項目：

- OPMN および opmn.xml の詳細は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

Oracle HTTP Server を使用する OPMN 管理 OC4J での SSL の使用

OC4J が OPMN によって管理され、かつ Oracle HTTP Server が Web リスナーとして使用される Oracle Application Server 環境では、OC4J が AJPS（セキュアなバージョンの Apache JServ Protocol）を使用して、Oracle HTTP Server と OC4J 間の SSL 通信をサポートします。この項では、このシナリオで SSL を使用する方法について説明します。これは次の手順で行います。

1. OC4J の SSL 用構成（Oracle HTTP Server を使用するシナリオ）
2. AJP over SSL の構成

この項の最後に、サンプルの構成ファイルを示します。

注意： Oracle Application Server 10.1.3.x の実装では、Oracle HTTP Server とクライアント間の通信に対して SSL がデフォルトで有効になります。特別な手順は不要です。これは、OC4J と Oracle HTTP Server 間での SSL の使用に関するこの項での説明とは無関係です。

関連項目：

- Oracle HTTP Server での SSL の使用についての関連情報（クライアント認証を有効にするための構成のカスタマイズ方法も含む）は、『Oracle HTTP Server 管理者ガイド』を参照してください。
- SSL を利用するための追加 Oracle Application Server コンポーネントの構成方法は、『Oracle Application Server 管理者ガイド』を参照してください。

OC4J の SSL 用構成（Oracle HTTP Server を使用するシナリオ）

Oracle Application Server 環境において、SSL を使用できるように OC4J を構成する方法は、15-5 ページの「スタンドアロン OC4J での SSL の使用」で説明している、スタンドアロン OC4J の構成方法とほぼ同じです。詳細はこの項を参照してください。

1. キーストアを作成します。
2. `secure-web-site.xml`（規約上、この名前を使用）を作成します。必要に応じて、`default-web-site.xml` から内容をコピーします。
3. 次の要素を使用して、`secure-web-site.xml` を更新します。
   a. `<web-site>` 要素に `secure="true"` を追加して、この要素を更新します。通常、Oracle Application Server 環境では、自動的に追加される `port="0"` 設定で指定されるように、OPMN に従ってポートが選択されます。また、Oracle Application Server 環境のデフォルトの設定である `protocol="ajp13"` を使用します。（`protocol="ajp13"` 設定と `secure="true"` を組み合せることで、AJPS が使用されます。）

   ```
   <web-site port="0" secure="true" protocol="ajp13"
             display-name="Default OracleAS Containers for J2EE Web Site" >
      ...
   </web-site>
   ```

   （必要に応じて、`display-name` 設定も変更する必要があります。）

   b. `<web-site>` の下に `<ssl-config>` 要素を追加し、`keystore` と `keystore-password` 属性を使用して、キーストアの場所とパスワードを定義します。
   c. 変更結果を `secure-web-site.xml` に保存します。

次に例を示します。

```
<web-site display-name="OC4J Web Site" protocol="ajp13" port="0"
          secure="true" >
   <default-web-app application="default" name="defaultWebApp" root="/j2ee" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome" />
</web-site>
```

Oracle HTTP Server を介する通信の場合、protocol の値は安全な通信を使用するかどうかにかかわらず、常に ajp13 になります。protocol の値が ajp13 で secure="false" の場合は、AJP プロトコルを示し、ajp13 で secure="true" の場合は、AJPS プロトコルを示します。

4. server.xml が、secure-web-site.xml を参照しているかどうかを確認します。
5. OC4J を停止してから再起動し、secure-web-site.xml を初期化します。

重要： OC4J 10.1.3 実装では、単一の AJP/AJPS Web サイトのみがサポートされます。たとえば、AJP Web サイトを構成する default-web-site.xml ファイルと、AJPS Web サイトを構成する secure-web-site.xml ファイルを同時に使用することはできません。

注意：

- この Web サイトの XML ファイルの設定をオーバーライドしないように OPMN を構成するには、ここでポート 0 のかわりに、実際に使用するポートを指定します。
- Oracle HTTP Server を Web リスナーとする Oracle Application Server 環境では、OC4J の <ssl-config> 要素の needs-client-auth 属性は、ブラウザからの SSL 認証に関係しません。この認証は、クライアントと Oracle HTTP Server 間の取決めによって行われます。ただし、OC4J で Oracle HTTP Server からの SSL 認証を必要とする場合は、この属性が関係します。関連情報は、15-15 ページの「クライアント認証の要求」を参照してください。

AJP over SSL の構成

この項では、AJP over SSL を使用する場合の次の状況について説明します。

- OC4J と Oracle HTTP Server 間の AJPS の構成
- AJPS をサポートするための OPMN の構成（Oracle HTTP Server を使用するシナリオ）

OC4J と Oracle HTTP Server 間の AJPS の構成

OC4J と Oracle HTTP Server 間の AJPS を構成するには、次の手順を実行します。

1. Oracle Wallet Manager を使用して、Oracle HTTP Server で使用する自動ログイン Wallet（SSO Wallet とも呼ばれる）を作成します。
2. keytool ユーティリティを使用し、証明書をキーストアからエクスポートします。（前述の、OC4J の SSL 用構成手順により、OC4J にキーストアを作成済であるものとします。）

```
% keytool -export -file cert_file_name -keystore keystore_file_name \
          -storepass=password
```

ここで、*cert_file_name* は、生成される証明書の目的ファイル名、*keystore_file_name* は、作成済のキーストアの名前です。コマンドラインで storepass を省略し、かわりにパスワードの入力が要求されるようにすることもできます。コマンドが成功すると、証明書ファイル名を確認するメッセージが表示されます。

3. Oracle Wallet Manager を使用して、生成された証明書を Wallet にインポートします。「操作」の下の「信頼できる証明書のインポート」を使用します。

4. Oracle HTTP Server で、mod_oc4j.conf ファイル内の SSL 設定が安全な通信に適した値に設定されていることを確認します。SSL を有効にし、手順 1 で作成した Wallet へのパスを指定する必要があります（ここに Wallet のパスワードを指定することは不要です）。

```
Oc4jEnableSSL on
Oc4jSSLWalletFile wallet_path
```

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。）

関連項目：

- mod_oc4j.conf の詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。
- 手順 1 および 3 に関連する Wallet と証明書の管理の詳細は、『Oracle Application Server 管理者ガイド』を参照してください。

AJPS をサポートするための OPMN の構成（Oracle HTTP Server を使用するシナリオ）

Oracle Application Server 環境では、OPMN の構成手順も実行する必要があります。ファイル *ORACLE_HOME*/opmn/conf/opmn.xml を、次のように更新します。

1. コンポーネント ID OC4J 下で、セキュリティ・パラメータを構成します（Wallet 情報）。

```
<ias-component id="OC4J">
   ...
   <category id="security-parameters">
      <data id="wallet-file" value="file:walletfile"/>
      <data id="wallet-password" value="pwd"/>
   </category>
   ...
</ias-component>
```

2. 同じくコンポーネント ID OC4J 下で、Web サイトに対して AJPS プロトコルを指定します。

```
<ias-component id="OC4J">
   ...
   <port id="secure-web-site" range="12501-12600" protocol="ajps"/>
   ...
</ias-component>
```

3. コンポーネント ID HTTP_Server 下で、SSL がデフォルトの ssl-enabled 設定により有効になっているかどうかを確認します。（ssl-disabled の設定では、SSL は無効になっています。）

```
<ias-component id="HTTP_Server">
   ...
   <data id="start-mode" value="ssl-enabled"/>
   ...
</ias-component>
```

関連項目：

- OPMN および opmn.xml の詳細は、『Oracle Process Manager and Notification Server 管理者ガイド』を参照してください。

SSL 用構成ファイルのサンプル

この項では、先行する各項で説明した構成に関連したサンプルを示します。

<web-site> 要素のサンプル

secure-web-site.xml ファイル内の <web-site> 要素のサンプルを示します。

```
<web-site port="0" protocol="ajp13" secure="true">
   <default-web-app application="default" name="defaultWebApp" root="/j2ee" />
   <web-app application="default" name="dms" root="/dmsoc4j" />
   ...
   <ssl-config
        keystore="C:\demotest\j2eetest\tsrc\shiphome\sslfiles\KEYSTORE\keystore"
        keystore-password="welcome1"/>
</web-site>
```

サンプルの mod_oc4j.conf ファイル

サンプルの mod_oc4j.conf ファイルを示します。

```
<IfModule mod_oc4j.c>

    Oc4jEnableSSL on
    Oc4jSSLWalletFile C:\demotest\j2eetest\tsrc\shiphome\sslfiles\ssl.wlt\default
    Oc4jSSLWalletPassword welcome1

    <Location /oc4j-service>
        SetHandler oc4j-service-handler
        Order deny,allow
        Deny from all
        Allow from localhost ani-pc.us.oracle.com ani-pc
    </Location>

</IfModule>
```

サンプルの opmn.xml ファイル

opmn.xml に含まれる、コンポーネント ID OC4J および HTTP_Server の構成のサンプルを示します。

```
<ias-component id="OC4J">
   <process-type id="home" module-id="OC4J" status="enabled">
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-Xrs -server
               -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
               -Djava.awt.headless=true"/>
         </category>
         <category id="security-parameters">
            <data id="wallet-file" value=
              "file:C:/demotest/j2eetest/tsrc/shiphome/sslfiles/ssl.wlt/default"/>
            <data id="wallet-password" value="welcome"/>
         </category>
         <category id="stop-parameters">
            <data id="java-options" value=
              "-Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
               -Djava.awt.headless=true"/>
         </category>
      </module-data>
      <start timeout="600" retry="2"/>
      <stop timeout="120"/>
      <restart timeout="720" retry="2"/>
      <port id="secure-web-site" range="12501-12600" protocol="ajps"/>
      <port id="rmi" range="3201-3300"/>
      <port id="jms" range="3701-3800"/>
```

```
      <process-set id="default_island" numprocs="1"/>
   </process-type>
</ias-component>

<ias-component id="HTTP_Server">
   <process-type id="HTTP_Server" module-id="OHS">
      <module-data>
         <category id="start-parameters">
            <data id="start-mode" value="ssl-enabled"/>
         </category>
      </module-data>
      <process-set id="HTTP_Server" numprocs="1"/>
   </process-type>
</ias-component>
```

クライアント認証の要求

この項では、サーバーに対するクライアントの SSL 認証について、特に OC4J の needs-client-auth 属性を使用する OC4J クライアント認証モードに重点を置いて説明します。

ここで検討するシナリオは次のとおりです。

- エンドユーザーを OC4J（スタンドアロンまたは OPMN 管理）に対する直接のクライアントとするシナリオ
- ここでの説明のため、Oracle HTTP Server を Oracle Application Server 環境内の OC4J に対するクライアントとするシナリオ
- エンドユーザーを Oracle Application Server 環境内の Oracle HTTP Server に対するクライアントとするシナリオ（needs-client-auth 属性を含めた OC4J 構成は無関係）

OC4J クライアント認証モードの概要

OC4J は、クライアント認証モードをサポートします。このモードでは、OC4J サーバーが、クライアントから（または Oracle Application Server 環境では Oracle HTTP Server から）の SSL 認証を明示的に要求します。クライアントは、接続時に OC4J からのリクエストに従って、デジタル証明書で自己認証する必要があります。

クライアントと OC4J 間の認証を使用した安全な通信中に、次の機能が実行されます。

- 両者間の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 に対してクライアントが認証されます。

次の例のように、secure-web-site.xml 内の <ssl-config> 要素の needs-client-auth 属性を通じてクライアント認証をリクエストし、次に示す手順を実行します。

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

1. OC4J が信頼する証明書は、トラスト・ポイントと呼ばれます。クライアント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。15-5 ページの「スタンドアロン OC4J での SSL 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. 証明書をクライアント上のファイルに保存します。
5. クライアントが安全な接続を開始できるようにするための証明書を提供します。

OC4J は、安全な通信のために、クライアント・エンティティをクライアント ID に従って受け入れるか拒否します。クライアントのルート証明書が認識されると、クライアントは受け入れられます。`secure-web-site.xml` 内の `<ssl-config>` 要素で指定するキーストアには、OC4J への接続の認可を受ける任意のクライアントの証明書が格納されている必要があります。

クライアント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
- クライアント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
- クライアント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。

OC4J に対するクライアント認証

スタンドアロン OC4J または Oracle HTTP Server を使用しない OPMN 管理 OC4J で OC4J HTTP リスナーが使用される場合は、`secure-web-site.xml` 内の `<ssl-config>` 要素で `needs-client-auth="true"` を設定することにより、クライアント（エンドユーザー）からの SSL 認証をリクエストできます。実際に、OC4J リスナーで CLIENT-CERT 認証を使用するには、この設定が必要です。

証明書を使用するには、証明書をクライアント・ブラウザのセキュリティ領域に設定するか（クライアントがブラウザの場合）、HTTPS 接続の開始時にクライアント証明書および証明連鎖をプログラムで提示します（Java クライアントの場合）。

追加情報は、前項の「OC4J クライアント認証モードの概要」を参照してください。

関連項目：

- 第 16 章「クライアント接続用 Oracle セキュリティ」

Oracle Application Server での OC4J に対する Oracle HTTP Server 認証

Oracle Application Server 環境では、Oracle HTTP Server が OC4J に対するクライアントとして動作します。このモードでの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。

このシナリオでは、15-15 ページの「OC4J クライアント認証モードの概要」での説明のために、Oracle HTTP Server をクライアントとみなします。

Oracle HTTP Server と OC4J 間の安全な通信には、HTTPS のかわりに AJPS（セキュアな Apache JServ Protocol）が使用されます。

Oracle HTTP Server に対するクライアント認証

Oracle Application Server 環境でクライアント（エンドユーザー）から Oracle HTTP Server に対する SSL 認証をリクエストする場合は、OC4J が関係せず、その構成（needs-client-auth 属性を含む）も無関係です。これには、クライアントと Oracle HTTP Server 間の取決めが必要となります。Oracle HTTP Server での SSL の使用方法（クライアント認証を有効にするための構成のカスタマイズ方法も含む）の詳細は、『Oracle HTTP Server 管理者ガイド』を参照してください。

クライアント・ブラウザのセキュリティ領域に証明書を設定するか（クライアントがブラウザの場合）、HTTPS 接続の開始時にクライアント証明書および証明連鎖をプログラムで提示します（Java クライアントの場合）。

SSL のトラブルシューティングとデバッグ

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 : システムにある最新 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 mykeystore -import -alias cacert -file cacert.cer
  (keytool -keystore mykeystore -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。

```
% keytool -keystore mykeystore -genkey -keyalg RSA -alias serverkey
% keytool -keystore mykeystore -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 mykeystore -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 : javax.net.debug プロパティ

javax.net.debug プロパティを使用して、Java Secure Socket Extension（JSSE）実装から SSL 接続に関する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。

- -Djavax.net.debug=help（オプション・リストを取得する場合）
- -Djavax.net.debug=all（すべての詳細を含んだデバッグ・メッセージを表示する場合）

これには、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 バージョンの表示が含まれます。

OC4J での ORMIS の有効化

ORMI over SSL（ORMIS）は、OC4J ではデフォルトで無効になっています。これは、ORMIS を使用する前にクライアントおよびサーバーのキーストアまたは Oracle Wallet を作成することが推奨されているためです。

この項では、スタンドアロン環境または Oracle Application Server におけるクラスタ環境の OC4J で、ORMIS を有効にする構成について説明します。この構成手順を実行すると、これまで ormi: プロトコルを使用していたすべてのケースで、ormis: プロトコルを使用できるようになります。

この項の内容は次のとおりです。

- スタンドアロン OC4J での ORMIS の構成
- Oracle Application Server 環境における OC4J での ORMIS の構成
- ORMIS アクセス制限の構成
- ORMIS を使用するためのクライアントの構成

関連項目：

- OC4J における ORMI 使用の一般情報は、『Oracle Containers for J2EE サービス・ガイド』を参照してください。

スタンドアロン OC4J での ORMIS の構成

ORMIS 構成および関連する RMI 構成では、各 OC4J インスタンス上の server.xml ファイルおよび rmi.xml ファイルの更新が必要になります。この項の内容は次のとおりです。

- RMI 構成ファイルの場所に関する server.xml の構成
- rmi.xml の ORMIS 用構成
- ORMIS 有効化時の ORMI の無効化（オプション）

RMI 構成ファイルの場所に関する server.xml の構成

OC4J インスタンスで ORMIS を有効にするには、最初に、OC4J サーバー構成ファイル server.xml に、OC4J RMI 構成ファイル rmi.xml へのパスを指定する <rmi-config> 要素が含まれるかどうかを確認します。

rmi.xml へのパスは次のように指定します。

```
<rmi-config path="rmi_path" />
```

server.xml ファイルおよび rmi.xml ファイルのどちらも、通常 *ORACLE_HOME*/j2ee/home/config ディレクトリにあるため、*rmi_path* の値は、通常 ./rmi.xml になります。

関連項目：

- server.xml の詳細は、『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

rmi.xml の ORMIS 用構成

ORMIS を使用するには、次の手順に従い、各 OC4J インスタンス上の rmi.xml に SSL 構成を定義します。

1. <rmi-server> 要素の ssl-port 属性を使用して、SSL リスナー・ポートを指定します。次に例を示します。

```
<rmi-server ... port="23791" ssl-port="23943">
   ...
</rmi-server>
```

（ここでは、ORMI リスナー・ポートの 23791 への設定も行われています。）

> **注意：** デフォルトの RMI ポートは 23791 です。デフォルトの ORMIS ポートは 23943 です。

2. <rmi-server> 要素の下に <ssl-config> サブ要素を追加します。これは、キーストア構成用であり、任意の値を指定でき、OC4J 再起動時には（セキュアでない ORMI リスナーに加えて）ORMIS リスナーが起動されます。ORMIS を有効にする方法には、次に説明するように 2 つあります。1 つはキーストアおよびパスワードを使用する方法であり、もう 1 つは匿名暗号スイートを使用する方法です。

関連項目：

- rmi.xml の追加情報は、『Oracle Containers for J2EE サービス・ガイド』を参照してください。

キーストアおよびパスワードの使用 次の例では、SSL ポートを 23943 に設定し、Oracle Wallet ベースの証明書を使用するように OC4J を構成しています（同時に RMI ログ・ファイルを指定しています）。

```
<rmi-server xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
               "http://xmlns.oracle.com/oracleas/schema/rmi-server-10_0.xsd"
            port="23791" ssl-port="23943">
   <ssl-config keystore="/wallets/wallet-server-a/ewallet.p12"
               keystore-password="serverkey-a" />
   ...
   <log>
      <file path="../log/rmi.log" />
   </log>
</rmi-server>
```

keystore 属性の値には、キーストアの場所（絶対パス、または Web サイトの XML ファイルが置かれている *ORACLE_HOME*/j2ee/home/config に対する相対パス）およびファイル名を指定します。

Oracle Wallet のかわりに Java キーストアを使用するには、<ssl-config> 要素を次の例のように構成します。

```
<ssl-config keystore="/keystores/keystore_a.jks" keystore-password="serverkey-a"/>
```

キーストアおよびパスワードを使用する場合、サーバーのキーストアには、ORMIS を介して OC4J への接続認可を受ける任意のクライアントの署名済証明書、またはクライアントのルート CA 発行の証明書が格納されている必要があります。

匿名暗号スイートの使用 ORMIS を有効にするには、匿名暗号スイートも使用できます。この場合、<ssl-config> 要素から、keystore および keystore-password 属性を削除します。

```
<ssl-config />
```

このモードでは、任意の ORMIS クライアントが、証明書のチェックなしでサーバーに接続できます。

> **重要：** このモードは、クライアントがトランスポート・レベルでの認証を受けるかどうかを問わず SSL 通信を許可するため、慎重に使用してください。

ORMIS 有効化時の ORMI の無効化（オプション）

スタンドアロン OC4J では、ORMIS を有効にした場合は ORMI を無効にできます。これを行うには、ORMI ポートを -1 に設定します。

```
<rmi-server ... port="-1" ssl-port="23943">
   <ssl-config keystore="keystore" keystore-password="password" />
   ...
</rmi-server>
```

この構成を使用すると、OC4J の再起動時に、セキュアでない ORMI リスナーが無効になります。

> **注意：** これは、OPMN によって管理される OC4J インスタンスではサポートされません。

Oracle Application Server 環境における OC4J での ORMIS の構成

OPMN によって管理されるクラスタ Oracle Application Server 環境で ORMIS を有効にするには、次の手順を実行します。

1. 一般的な手順として、15-18 ページの「RMI 構成ファイルの場所に関する server.xml の構成」および 15-19 ページの「rmi.xml の ORMIS 用構成」で説明されている、スタンドアロン OC4J に対する手順を実行します。そこに含まれる手順のうち、rmi.xml の <rmi-server> 要素に ssl-port を設定する手順は実行しません。これは OPMN 管理環境では不要です。OPMN 管理 RMIS ポートが、rmi.xml の ssl-port 属性を物理的にオーバーライドしてしまうためです。
2. opmn.xml ファイルにおいて、クラスタに属する Oracle Application Server インスタンスごとに、<port> 要素を次のように rmis のポート範囲付きで追加します。

```
<ias-component id="OC4J">
   <process-type id="home" module-id="OC4J" status="enabled">
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-server
                -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
                -Djava.awt.headless=true
                -Dhttp.webdir.enable=false"/>
         </category>
         <category id="stop-parameters">
            <data id="java-options" value=
               "-Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
                -Djava.awt.headless=true -Dhttp.webdir.enable=false"/>
         </category>
      </module-data>
      <start timeout="600" retry="2"/>
      <stop timeout="120"/>
      <restart timeout="720" retry="2"/>
      <port id="default-web-site" range="12501-12600" protocol="ajp"/>
```

```
      <port id="rmi" range="12401-12500"/>
      <port id="rmis" range="12701-12800"/>
      <port id="jms" range="12601-12700"/>
      <process-set id="default_group" numprocs="1"/>
   </process-type>
   ...
</ias-component>
```

関連項目：

- OPMN および opmn.xml ファイルの一般情報は、『Oracle Application Server 管理者ガイド』を参照してください。

ORMIS アクセス制限の構成

ORMIS は、ORMI と同様に、アクセス制御リスト（ACL）マスクを設定することで、受信 IP アクセスを制限する機能をサポートします。この設定は rmi.xml 内の <access-mask> 要素、およびそのサブ要素である <host-access> と <ip-access> を介して行います。

アクセス制御には、除外型と包含型があります。

- 除外モードでは、明示的に指定されているものを除くすべての IP アドレスまたはホストからのアクセスが拒否されます。<access-mask> で mode="deny" を使用してから、<host-access> サブ要素か <ip-access> サブ要素（またはこの両方）で mode="allow" を使用することで、アクセスを許可する特定のホストまたは IP アドレスを指定します。
- 包含モードでは、明示的に除外されているものを除くすべての IP アドレスまたはホストからのアクセスが可能になります。<access-mask> で mode="allow" を使用してから、<host-access> サブ要素か <ip-access> サブ要素（またはこの両方）で mode="deny" を使用することで、アクセスを拒否する特定のホストまたは IP アドレスを指定します。

次の例では、除外モードを構成して、localhost および 192.168.1.0 へのアクセスのみを許可しています。（255.255.255.0 は適用されるサブネット・マスクです。）

```
<rmi-server xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
                "http://xmlns.oracle.com/oracleas/schema/rmi-server-10_0.xsd"
            port="23791" ssl-port="23943">

   <ssl-config keystore="../wallets/wallet-server-a/ewallet.p12"
               keystore-password="serverkey-a" />

   <access-mask default="deny">
      <host-access domain="localhost" mode="allow"/>
      <ip-access ip="192.168.1.0" netmask="255.255.255.0" mode="allow"/>
   </access-mask>

   ...

</rmi-server>
```

関連項目：

- orion-web.xml でも同じ機能でサポートされている <access-mask> 要素の追加情報は、『Oracle Containers for J2EE サーブレット開発者ガイド』を参照してください。

ORMIS を使用するためのクライアントの構成

この項では、ORMIS を使用するためのクライアント・サイドの構成について説明します。この項の内容は次のとおりです。

- 適切な Java ネーミング・プロバイダ URL の指定
- キーストアおよびパスワードの指定

適切な Java ネーミング・プロバイダ URL の指定

スタンドアロン OC4J 環境のアプリケーションの場合は、システムおよびアプリケーションの URI を定義する `java.naming.provider.url` 環境プロパティの設定に、`ormis` プロトコルを指定します。

```
java.naming.provider.url=ormis://hostname/appname
```

Oracle Application Server（OPMN 管理）環境のアプリケーションの場合は、`opmn:ormis` プロトコルを指定します。

```
java.naming.provider.url=opmn:ormis://hostname/appname
```

注意： URL には、ポート番号を含める必要はありません。使用されるポートは、プロトコルによって決定されます。

キーストアおよびパスワードの指定

EJB over ORMIS をコールするには、クライアント・サイドで次の指定（必要なもののみ）を行う必要があります。

- クライアント・キーストアへのパス（絶対パスを推奨）

  これは、サーバー証明書がインポートされているクライアント・サイド・キーストアの場所です。

- キーストアのパスワード

これらの設定は 3 箇所で指定できます。優先度の高い順に記述します。

- JSSE プロパティとして指定

  ```
  -Djavax.net.ssl.keyStore=keystore_path
  -Djavax.net.ssl.keyStorePassword=keystore_password
  ```

- `jndi.properties` のプロパティとして指定（JSSE プロパティ設定が指定されている場合は無視されます。）

  ```
  oc4j.[rmi.]keyStoreLoc=keystore_path
  oc4j.[rmi.]keyStorePass=keystore_password
  ```

- `ejb_sec.properties` のプロパティとして指定（JSSE プロパティ設定または `jndi.properties` プロパティ設定が指定されている場合は無視されます。）

  ```
  oc4j.[rmi.]keyStoreLoc=keystore_path
  oc4j.[rmi.]keyStorePass=keystore_password
  ```

注意：

- `oc4j.keyStoreLoc` または `oc4j.rmi.keyStoreLoc` のいずれかを指定できます。`keyStorePass` も同様です。
- `ejb_sec.properties` を使用するには、それを、クライアントの Java VM が起動されたディレクトリに置きます。

HTTPS を介した ORMI トンネリングの有効化

『Oracle Containers for J2EE サービス・ガイド』の RMI に関する章では、HTTP を介した ORMI トンネリングの構成方法が説明されています。

SSL 機能を使用するために、HTTPS を介した ORMI トンネリングを構成することも可能です。基本的な手順は次のとおりです。

1. 15-5 ページの「スタンドアロン OC4J での SSL の使用」または 15-11 ページの「Oracle HTTP Server を使用する OPMN 管理 OC4J での SSL の使用」のいずれか該当する方の説明に従って、SSL の構成を完了させます。

   スタンドアロン OC4J の場合は、ここでキーストアを作成し、`secure-web-site.xml` ファイルを構成します。

   Oracle Application Server 環境の場合は、ここでキーストアを作成して `secure-web-site.xml` ファイルを構成し（記載されているように、スタンドアロン OC4J と比べて一部異なっています）、必要に応じて AJP over SSL を構成し、さらに HTTP を有効にして SSL を使用するように OPMN を構成します。Oracle HTTP Server では、SSL がデフォルトで有効になることに注意してください。

2. クライアントを適切に構成します（15-22 ページの「ORMIS を使用するためのクライアントの構成」で説明されている手順と並行して実行）。

   a. スタンドアロン OC4J または Oracle Application Server 環境のいずれかの場合は、システムおよびアプリケーションの URI を定義する `java.naming.provider.url` 環境プロパティの設定で `ormi:https` プロトコルを指定します。

      `java.naming.provider.url=ormi:https://`*`hostname`*`:`*`https_port`*`/`*`appname`*

      スタンドアロン OC4J の場合は、`secure-web-site.xml` で指定されているように *`https_port`* を指定します。15-5 ページの「スタンドアロン OC4J での SSL の使用」を参照してください。Oracle Application Server 環境では、*`https_port`* に Oracle HTTP Server SSL ポートを指定します。

   b. 15-22 ページの「キーストアおよびパスワードの指定」の説明に従って、キーストアとパスワードを構成します。

次のクライアント・コード Snippet では、`ormi:https` プロトコルが指定された URL が使用されています。

```
private static Context getInitialContext() throws NamingException {
    Hashtable env = new Hashtable();
    env.put( Context.INITIAL_CONTEXT_FACTORY,
      "oracle.j2ee.naming.ApplicationClientInitialContextFactory" );
    env.put( Context.SECURITY_PRINCIPAL, "oc4jadmin" );
    env.put( Context.SECURITY_CREDENTIALS, "welcome1" );
    env.put(Context.PROVIDER_URL, "ormi:https://localhost:443/apache-ejb");
    env.put("oc4j.keyStoreLoc",
      "C:/product/iasSOA0622/Apache/Apache/conf/ssl.wlt/default/ewallet.p12");
    env.put("oc4j.keyStorePass", "welcome");

    return new InitialContext( env );
}
```

16

クライアント接続用 Oracle セキュリティ

この章では、クライアントの Oracle セキュリティの機能について説明します。この機能は、主に、クライアント HTTP 接続に Secure Sockets Layer（SSL）機能と `HTTPClient` パッケージの機能を提供し、標準的な Java Secure Socket Extension（JSSE）の使用をサポートします。また、HTTP クライアントの非 SSL 認証機能を説明する項が含まれています。この章では、OC4J が Web リスナー（スタンドアロンの OC4J など）であるか、OC4J が Oracle HTTP Server の背後にある状況で、SSL を使用する Java アプリケーションについて説明します。
この章の内容は次のとおりです。

注意：

- JSSE の他に Oracle Java SSL もサポートされていますが、OC4J 10.1.3.1 実装では非推奨であり、将来のリリースではサポートされなくなります。そのため、JSSE を使用することをお薦めします。Oracle Java SSL を非推奨とする前段階として、OC4J 10.1.3.1 実装の `HTTPClient` の場合、JSSE がデフォルトの SSL 実装になっています。Oracle Java SSL についての最後の項を除き、この章では JSSE の使用方法を中心に説明します。
- 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。Secure Sockets Layer を使用するための OC4J の構成に関する一般情報は、第 15 章「OC4J との SSL 通信」を参照してください。15-15 ページの「クライアント認証の要求」も参照してください。

関連項目：

- JSSE に関する一般情報は、次の URL を参照してください。

  `http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html`

非 SSL クライアント認証の使用方法

この章では、主にクライアント・セキュリティ用 HTTPS について説明しています。ただし、一部の HTTP クライアント API は、非 SSL 認証用にクライアントで使用できます。ライブラリを使用して、リクエスト・ヘッダーに Basic、Digest および NTLM の認証情報を含める HTTP リクエストを作成できます。この項では、`HTTPClient` パッケージを使用してこれらのタイプのリクエストを作成する方法を説明します。HTTP クライアント API の詳細は、『Oracle Application Server HTTPClient Java API Reference』を参照してください。

Basic ベース・クライアント認証

HTTP クライアントを使用して、Basic 認証を要求するサーバー・リソースにアクセスするための HTTP リクエストを作成できます。次の例は、リクエストの作成、およびリクエストで `addBasicAuthorization` メソッドを使用してレルム、ユーザー名およびパスワードを含める方法を示しています。

```
URL url = new URL("http://localhost:1234");

HTTPConnection client = new HTTPConnection(url);

try {
   client.addBasicAuthorization("realm_name", "user", "password");
   HTTPResponse response = client.Get(url.getFile());
   //assertEquals(200, response.getStatusCode());
}
finally {
   client.stop();
}
```

レルムは、特定のサーバー下にある様々な URL をグループ化するサーバー指定文字列であり、サーバーによって認証チャレンジが発行された場合に正しい情報を選択するために使用されます。レルムを使用しないスキームでは、レルムを空の文字列（`""`）にする必要があります。

Digest ベース・クライアント認証

HTTP クライアントを使用して、Digest 認証を要求するリソースにアクセスするための HTTP リクエストを作成できます。Digest 認証メカニズムでは、クライアントが自己認証を行うために示すパスワードが、MD5 ダイジェストを使用して暗号化されます。これは、リクエスト・メッセージで送信されます。ユーザーからは、Digest 認証は Basic 認証と同じように動作するように見えます。次の例は、リクエストの作成、およびリクエストで `addDigestAuthorization` メソッドを使用してレルム、ユーザー名およびパスワードを含める方法を示しています。

```
HTTPConnection client = new HTTPConnection(url);

try {
   client.addDigestAuthorization("ProxyAuthDigestSchemeTestServlet",
      validUserName,validPassword);
   HTTPResponse response = client.Get(url.getFile());
   // assertEquals(200, response.getStatusCode());
}
finally {
   client.stop();
}
```

レルムは、特定のサーバー下にある様々な URL をグループ化するサーバー指定文字列であり、サーバーによって認証チャレンジが発行された場合に正しい情報を選択するために使用されます。レルムを使用しないスキームでは、レルムを空の文字列（`""`）にする必要があります。

NTLM ベース・クライアント認証

NTLM は、Microsoft のブラウザ、プロキシおよびサーバーで使用される、独自のチャレンジ / レスポンス認証プロトコルです。NTLM を使用するクライアントは、パスワードを送信せずに、サーバーに対してアイデンティティを証明できます。NTLM は接続指向プロトコルです。接続が認証されると、接続がオープンな間はその他の資格証明は不要です。

NTLM では、NT ドメイン名によってユーザー名が修飾されます。アカウント識別子は ¥ です。NT ドメインは、ユーザー名の前に NT ドメイン名とバックスラッシュを付けることで、HTTP クライアントで指定できます。たとえば、NT ドメインが OPERATIONS でユーザー名が jsmith の場合、完全修飾ユーザー名は OPERATIONS¥jsmith です。

NT ドメインが指定されていない場合は、デフォルト（設定されている場合）をそれとみなします。デフォルトの NT ドメインは、システム・プロパティ HTTPClient.ntlm.defaultDomainName を使用して HTTP クライアントで設定されます。NT ドメインなしでユーザー名が指定されており、デフォルトの NT ドメインが HTTP クライアントで定義されていない場合、NTLM 保護されているサーバーでは、独自のデフォルト NT ドメインがそれとみなされます。

NTLM 保護されているリソース・サーバーに接続するには、NTLM 資格証明を HTTP クライアントの AuthorizationInfo 資格証明ストアに追加します。Basic 認証や Digest 認証の場合のように、NTLM サーバーによってチャレンジが行われると、HTTP クライアントは自動的に資格証明ストアに問い合せます。資格証明は、次の方法のいずれかで資格証明ストアに追加できます。HTTP Connection インスタンスを使用する方法は次のとおりです。

```
HTTPConnection conn = new HTTPConnection( myHost, myPort );
conn.addNtlmAuthentication( myUsername, myPassword );
```

AuthorizationInfo クラスを使用して直接追加する方法は次のとおりです。

```
AuthorizationInfo.addNtlmAuthentication( myHost, myPort, myUsername, myPassword )
```

次の例は、リクエストの作成、およびリクエストで addNtlmAuthentication メソッドを使用してユーザー名およびパスワードを含める方法を示しています。

```
HTTPConnection conn = new HTTPConnection( myHost, myPort );
conn.addNtlmAuthentication( myUsername, myPassword );
HTTPResponse response = conn.Get( "/index.htm" );
int status = response.getStatusCode();
assertEquals( 200, status );
```

注意： Basic 認証などの認証スキームで指定されるため、レルムは NTLM に適用されません。NTLM チャレンジにはレルム・ディレクティブは含まれません。そのため、すべての NTLM 資格証明は、HTTP クライアント内の同じ空の（""）レルムの一部とみなされます。

NTLM 保護されているプロキシ・サーバーへの接続方法

プロキシ・サーバーでも、クライアント認証に NTLM が使用されることがあります。ただし、リクエスト指向の認証とは異なり、NTLM クライアントは、リソース・サーバーではなくプロキシとの接続のみを認証します。

NTLM 保護されているプロキシ・サーバーに接続するには、NTLM 資格証明を HTTP クライアントの AuthorizationInfo 資格証明ストアに追加します。Basic 認証や Digest 認証の場合のように、NTLM サーバーによってチャレンジが行われると、HTTP クライアントは自動的に資格証明ストアに問い合せます。資格証明は、AuthorizationInfo を使用して資格証明ストアに直接追加することのみ可能です。次に例を示します。

```
AuthorizationInfo.addNtlmAuthentication( myProxyHost, myProxyPort, myUsername,
   myPassword)
```

次に、保護されているプロキシ・サーバーに NTML を使用して接続するクライアントの例を示します。

```
HTTPConnection conn = new HTTPConnection( myHost, myPort );
conn.setCurrentProxy( myProxyHost, myProxyPort );
AuthorizationInfo.addNtlmAuthentication( myProxyHost, myProxyPort, myUsername,
   myPassword, conn.getContext() )
HTTPResponse response = conn.Get( "/index.htm" );
int status = response.getStatusCode();
assertEquals( 200, status );
```

HTTPS およびクライアント

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 Web アプリケーションなど、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。`HTTPClient` パッケージまたは 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、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 クライアント HTTPS 機能は、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` クラスをベースにしています。`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。

重要： `HTTPClient` の Oracle 実装は、ベースになったオリジナルのオープン・ソース・バージョンから変更されています。Oracle バージョンは別個の製品と考えてください。類似点は多数残っていますが、この 2 つには必ずしも相互に互換性があるわけではありません。

関連項目：

- JSSE と `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/products/jsse/index.jsp`

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

- 『Oracle Application Server HTTPClient Java API Reference』（`HTTPClient` パッケージ用の Javadoc）

クライアント・サイド HTTPS 機能の概要

Oracle クライアント HTTPS は、`HTTPClient` パッケージの `HTTPConnection` クラスを拡張し、暗号スイートの選択、Oracle Wallet Manager による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のサポート、後述するその他の機能などの SSL 機能を提供します。Oracle クライアント 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。

`HTTPClient` では、JSSE および Oracle Java SSL という 2 つの SSL 実装がサポートされます。ただし、後者は OC4J 10.1.3.1 実装で非推奨であり、将来のリリースでサポートされなくなります。そのため、JSSE を使用することをお薦めします。

Oracle クライアント 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

さらに、`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 HTTPS トンネリング
- HTTP プロキシ認証

次項では、次の機能について説明します。

- サポートされているキーストア形式
- 確立された SSL 接続に関する情報へのアクセス
- java.net.URL フレームワークのサポート
- SSL 暗号スイート

サポートされているキーストア形式

JSSE を使用する場合、Oracle JSSE 実装（`OraclePKIProvider`）と PKCS12 または SSO（自動ログイン）Oracle Wallet の組合せか、デフォルトの Sun 社の JSSE 実装と JKS 形式キーストアの組合せを使用できます。（Oracle Java SSL は、テキスト形式の Oracle Wallet のみサポートします。）

PKCS12 または SSO Wallet のいずれかを使用する場合、資格証明情報は暗号化されます。主な違いは、SSO Wallet を使用する場合、アクセス時に Wallet を開くために Wallet パスワードを指定する必要がないことです。

JKS と PKCS12 は標準形式です。SSO Wallet は Oracle 独自の形式です。

関連項目：

- PKCS12 と SSO/ 自動ログイン Wallet の作成と使用の詳細は、『Oracle Application Server 管理者ガイド』（Wallet と資格証明の管理の章）を参照してください。

確立された SSL 接続に関する情報へのアクセス

Oracle `HTTPClient` パッケージの `HTTPConnection` クラスにある `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。

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 クライアント 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。

- 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
```

関連項目：

- java.net.URL クラスの Javadoc については、次の URL を参照してください。

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

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。

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。

OC4J 10.1.3.1 リリースで JSSE がサポートする暗号スイートを、デフォルトの優先順序で次に示します。このリストで、"*" が付いたスイートは、デフォルトで有効になる暗号スイートを表し、"**" が付いたスイートは、Sun 社の JCE Unlimited Strength Jurisdiction Policy Files のインストールが必要な暗号スイートを表しています。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。

注意： `HTTPClient` は、サポートされる暗号スイートのサブセットを選択的に有効にする手段を提供しません。

```
SSL_RSA_WITH_RC4_128_MD5 *
SSL_RSA_WITH_RC4_128_SHA *
TLS_RSA_WITH_AES_128_CBC_SHA *
TLS_DHE_RSA_WITH_AES_128_CBC_SHA *
TLS_DHE_DSS_WITH_AES_128_CBC_SHA *
SSL_RSA_WITH_3DES_EDE_CBC_SHA *
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA *
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA *
SSL_RSA_WITH_DES_CBC_SHA *
SSL_DHE_RSA_WITH_DES_CBC_SHA *
SSL_DHE_DSS_WITH_DES_CBC_SHA *
SSL_RSA_EXPORT_WITH_RC4_40_MD5 *
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA *
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA *
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA *
TLS_RSA_WITH_AES_256_CBC_SHA **
TLS_DHE_RSA_WITH_AES_256_CBC_SHA **
TLS_DHE_DSS_WITH_AES_256_CBC_SHA **
SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA **
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
```

関連項目：

- JSSE に関する一般情報は、次の URL を参照してください。

  `http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html`

サポートされているデフォルトのシステム・プロパティ

この項では、キーストアとトラストストアに対してサポートされる標準の Java システム・プロパティについて説明します。（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 クライアント HTTPS では、次のプロパティが認識されます。

- プロパティ javax.net.ssl.KeyStore
- プロパティ javax.net.ssl.KeyStorePassword
- プロパティ javax.net.ssl.keyStoreType
- プロパティ javax.net.ssl.trustStore
- プロパティ javax.net.ssl.trustStorePassword
- プロパティ javax.net.ssl.trustStoreType

注意： スタンドアロン OC4J の JVM コマンドラインか、Oracle Application Server 環境の opmn.xml ファイルの OC4J インスタンスの Java プロパティ設定で、Java システム・プロパティを設定できます。システム・プロパティの設定は、『Oracle Containers for J2EE 構成および管理ガイド』の OC4J ランタイム構成の章を参照してください。

プロパティ javax.net.ssl.KeyStore

このプロパティは、キーストアとして使用するキーストア・ファイルまたは Wallet ファイルの場所と名前を指定します。

プロパティ javax.net.ssl.KeyStorePassword

このプロパティは、キーストア（キーストア・ファイルまたは Wallet ファイル）を開くために必要なパスワードを指定できます。次に例を示します。

```
javax.net.ssl.keyStorePassword=welcome1
```

重要： キーストアのパスワードを 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。

- アプリケーションで相互認証が必要ない場合は、パスワードを必要としない SSO Wallet を使用します。
- 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() メソッドを使用して、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。

プロパティ javax.net.ssl.keyStoreType

このプロパティは、キーストアで使用されるファイルのタイプを指定します。Oracle JSSE 実装（OraclePKIProvider）の場合、PKCS12 または SSO を指定できます。デフォルトの Sun 社の JSSE 実装の場合、JKS を指定できます。

関連項目：

- PKCS12 のタイプの詳細は、『Oracle Application Server 管理者ガイド』に記載されている SSL の概要を参照してください。

プロパティ javax.net.ssl.trustStore

このプロパティは、javax.net.ssl.keyStore と同様に使用されます。このプロパティは、トラストストア（クライアントが暗黙的に受け入れる信頼できる認証局を格納するファイル）として使用するキーストア・ファイルまたは Wallet ファイルの場所と名前を指定します。

プロパティ javax.net.ssl.trustStorePassword

このプロパティは、javax.net.ssl.keyStorePassword と同様に使用されます。このプロパティは、トラストストア（キーストア・ファイルまたは Wallet ファイル）を開くために必要なパスワードを指定します。

プロパティ javax.net.ssl.trustStoreType

javax.net.ssl.keyStoreType と同様に、このプロパティは、トラストストアで使用するファイル・タイプを指定します。Oracle JSSE 実装の場合は PKCS12 または SSO で、Sun 社の JSSE 実装のデフォルトの場合は JKS です。

JSSE と HTTPClient の使用

この項では、JSSE を使用する HTTPS クライアント接続の Oracle Application Server サポートについて説明します。この項の内容は次のとおりです。

- JSSE の使用の前提条件
- JSSE を使用する HTTPClient の構成

関連項目：

- Wallet の作成と管理を行う Oracle Wallet Manager および orapki 機能の詳細は、『Oracle Application Server 管理者ガイド』（Wallet と証明書の管理を記載した章）を参照してください。
- JSSE の詳細は、次の URL を参照してください。

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

JSSE の使用の前提条件

Oracle クライアント HTTPS で JSSE を使用する場合、次の要件があります。

- Sun 社の JDK バージョン 1.2 以上を使用する必要があります。(JSSE は、JDK 1.4 以上の一部として組み込まれています。)
- ディレクトリ *ORACLE_HOME*/network/jlib にあるファイル oraclepki.jar が、クラスパス内に存在する必要があります。(Oracle Java SSL で使用するファイル jssl-1_1.jar または jssl-1_2.jar は、必要なくなりました。これらのファイルは JSSE と互換性がないため、インストール環境に含まれていません。)

JSSE を使用する HTTPClient の構成

Oracle Application Server は、JSSE を使用して HTTPS クライアント接続をサポートします。クライアントでは、次の手順に従い、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。

1. Sun 社の keytool を使用してトラストストアを作成します。

   関連項目：

   - 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

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 プロパティを介してクライアント・トラストストアの位置を指定する必要があります。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. 静的メソッド SSLSocketFactory.getDefault() をコールして、JSSE SSL ソケット・ファクトリ（javax.net.ssl.SSLSocketFactory インスタンス）を取得します。
4. HTTPClient 接続（HTTPConnection インスタンス）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、次のいずれかの方法があります。

   - （**接続ごと**）クライアントで HTTPConnection インスタンスの次のメソッドをコールします。その際、手順 3 の getDefault() メソッドにより取得された JSSE SSL ソケット・ファクトリを指定します。

     ```
     void setSSLSocketFactory(SSLSocketFactory factory)
     ```

     この場合、SSL ソケット・ファクトリはこの接続インスタンス用にのみ設定されます。その技術的方法については、後出の例 16-1 に示されています。

   - （**VM 全体**）クライアントで HTTPConnection クラスの次の静的メソッドをコールします。

     ```
     void HttpConnection.setDefaultSSLSocketFactory(SSLSocketFactory factory)
     ```

     この場合、メソッドを別の設定で再コールしないかぎり、SSL ソケット・ファクトリが Java VM 内のすべての接続インスタンスに対して設定されます。このメソッドは、対象となる 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 クラスの connect() メソッドを呼び出す必要があります。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。Oracle 実装の場合、HTTPConnection クラスの Get() メソッドなど、HTTP メソッドの呼び出し時にこのメソッドが暗黙的に呼び出されます。また、データ送信の前にコール元アプリケーションが SSL セッション情報を必要とする場合、connect() メソッドを明示的に呼び出すと便利です。16-14 ページの「追加の接続情報の検証」を参照してください。
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。

例 16-1 HTTPClient での JSSE の使用

```
public void obtainHTTPSConnectionUsingJSSE() throws Exception
{
   // set the truststore to the location of the client's truststore 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");

}
```

注意：

- SSL ソケット・ファクトリを指定しなかった状態で、Oracle Java SSL が指定されていないか、アプリケーションのクラスパスに Oracle Java SSL クラスが見つからない場合、デフォルトで JSSE が使用されます。SSL ソケット・ファクトリが指定されていない状態で Oracle Java SSL が指定されている場合は、Oracle Java SSL クラスがクラスパスで検出され、デフォルトで Oracle Java SSL が使用されます。16-17 ページの「HTTPClient に対する SSL 実装としての Oracle Java SSL の指定」を参照してください。
- JSSE SSL 実装はスレッド・セーフではありません。

SSL ホスト名検証の HTTPClient サポート

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。

ホスト名検証機能（`javax.net.ssl.HostnameVerifier` implementation）は、`HTTPClient` によって使用されます。これにより、保護されているサーバーへアクセスする際に使用される URI のホスト名と、SSL 証明書のホスト名が一致するかどうかが検証されます。これは、介入者攻撃の検出に役立ちます。`HTTPClient` は、SSL セッションを確立した直後に `HostnameVerifier` インスタンスを呼び出し、ホスト名の不一致が検出された場合は `javax.net.ssl.SSLPeerUnverifiedException` をスローします。

重要： 独自の `HostnameVerifier` を実装することも、Oracle 提供のインスタンス（後述）を使用することもできます。実装する場合、引数なしのコンストラクタが必要です。

関連項目：

- `javax.net.ssl.HostnameVerifier` の Javadoc は、次のサイトから入手できます。

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

この項では、この機能を有効にして使用する方法を説明します。この項の内容は次のとおりです。

ホスト名検証を実行するには、次に説明するシステム・プロパティ設定またはプログラム設定が必要です。

システム・プロパティ設定によるホスト名検証の有効化

コードを変更することなくホスト名検証を有効にするには、システム・プロパティ `HTTPClient.defaultHostnameVerifier` に、クラスパス内にあるホスト名検証機能実装の完全修飾されたクラス名を設定します。

設定する場合、適切なクラスの名前（引数なしのコンストラクタ付きの `javax.net.ssl.HostnameVerifier` 実装）を指定する必要があります。

プログラムによるホスト名検証の有効化

検証に使用する javax.net.ssl.HostnameVerifier インスタンスを指定することにより、次に示す HTTPConnection クラスのメソッドを使用して、ホスト名検証をプログラムによって有効にすることができます。

- static HostnameVerifier setDefaultHostnameVerifier (HostnameVerifier defaultHostnameVerifier)

  この静的メソッドを使用して、指定された HostnameVerifier インスタンスを JVM のデフォルトとして割り当てます。このメソッドは、以前に設定したデフォルトのホスト名検証機能を返します。ホスト名検証が以前にデフォルトで無効にされている場合は、null を返します。

- HostnameVerifier setHostnameVerifier (HostnameVerifier hostnameVerifier)

  接続で使用するため、このインスタンス・メソッドを使用してデフォルトのホスト名検証機能をオーバーライドし、指定された HostnameVerifier インスタンスを割り当てます。null を指定して、接続のホスト名検証を無効にすることもできます。

  このメソッドは、接続用の前のホスト名検証機能を返します。接続のホスト名検証があらかじめ無効にされている場合は、null を返します。

Oracle 標準のホスト名検証機能の使用

HTTPClient パッケージ内に、ホスト名検証機能の実装の StandardHostnameVerifier が用意されています。

StandardHostnameVerifier は、サイトのアイデンティティをチェックする標準的なホスト名一致規則を実装し、次の機能を備えています。

- 指定されたホスト名が SSL 証明連鎖内の最初の証明書の識別名（DN）の一般名（CN）と同じかどうかを比較することにより、SSL セッションのホスト名が検証されます。この比較の場合、大 / 小文字は区別されません。
- ワイルドカード・マッチングが有効になっている場合、指定されたホスト名が認識され、*.oracle.com などの SSL 証明書のワイルドカード CN と比較されます（存在する場合）。

StandardHostnameVerifier には、次のメソッドがあります。

- boolean setRecognizeWildcardCNs(boolean recognizeWildcardCNs)

  ワイルドカード CN を認識するかどうかを指定します。このメソッドは、前に設定された値を返します。

- boolean isRecognizeWildcardCNs()

  このメソッドは、ワイルドカード CN が認識されるかどうかを通知します。

- boolean verify(java.lang.String hostname, javax.net.ssl.SSLSession sslSession)

  このメソッドを呼び出し、ホスト名がサーバーの認証スキームと許容できる範囲で一致するかどうかを検証します。（これは、javax.net.ssl.HostnameVerifier インタフェースに指定された標準機能です。）

前述のように、プログラムまたはシステム・プロパティ設定により、StandardHostnameVerifier をホスト名検証機能として指定できます。デフォルトのホスト名検証機能として設定するには、次のシステム・プロパティ設定などを使用します。

```
HTTPClient.defaultHostnameVerifier=HTTPClient.StandardHostnameVerifier;
```

追加の接続情報の検証

HTTPConnection クラスの connect() が呼び出された後で、HTTPConnection クラスの getSSLSession() メソッドから返される javax.net.ssl.SSLSession オブジェクト内で検出されたデータを使用して、（ホスト名検証以外の）追加の検証を実行できます。

この検証を実行するためには、次のように、データを転送することなくサーバーへの接続を確立します。

```
httpsConnection.connect();
```

接続の確立後は、次のように接続情報（この場合はサーバーの証明連鎖）が取得されます。

```
peerCerts = (httpsConnection.getSSLSession()).getPeerCertificateChain();
```

最後に、次のようにサーバー証明書の一般名が取得されます。

```
String peerCertDN = peerCerts[0].getSubjectDN().getName();
peerCertDN = peerCertDN.toLowerCase();
```

（ユーザーの証明書は配列の先頭に、ルート CA の証明書は末尾にあります。）

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、次のように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);
}
```

注意： 前の項で説明したように、ホスト名検証機能を使用することをお薦めします。

Oracle Java SSL から JSSE への移行

前述のように、OC4J 10.1.3.1 以降の実装の場合、Oracle Java SSL ではなく JSSE を使用することをお薦めします。Oracle Java SSL は 10.1.3.1 実装で非推奨であり、将来のリリースでサポートされなくなります。この項では、SSL 機能に対して Oracle Java SSL のかわりに JSSE を使用するようクライアント・コードを変更する方法を説明します。

JSSE への移行のコード・サンプル

この項では、JSSE を使用する新しい SSL ソケット・ファクトリを作成する手順を示すため、以前に Oracle Java SSL で使用していたものと同じコードと比較しながら、コード・サンプルを説明します。`java.security.Keystore` クラスと様々な `javax.net.ssl` クラスは、ピア証明の検証用にキーストアまたは Wallet を開き、Oracle 固有のセキュリティ・ポリシーを適用する場合に使用されます。

関連項目：

- `Keystore` の Javadoc および `javax.net.ssl` クラスは、次のサイトから入手できます。

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

次に、説明と比較を加えながら手順を説明します。

1. SSL プロバイダを登録します。プロバイダが JDK の `jre/lib/security/java.security` プロパティ・ファイルに静的に設定されている場合、この手順は必要ありません。

   Oracle Java SSL 用の従来のコード: なし

   JSSE 用の新しいコード:

   ```
   Security.insertProviderAt(new oracle.security.ssl.OraclePKIProvider(), 1);
   ```

   `java.security.Security` クラスの静的な `insertProviderAt()` メソッドは、指定された優先順位に新しいプロバイダを追加します。この順位は、要求されたアルゴリズムでプロバイダを検索する優先順位を表します。1 が最も優先順位が高くなります。Oracle セキュリティ・ポリシーを強制する場合、`OraclePKIProvider` を使用します。プロバイダを設定すると、SSL 機能に対して標準の JSSE クラスを使用できます。

2. キーストア、Wallet または信頼できる認証局をロードします。

   Oracle Java SSL 用の従来のコード:

   ```
   OracleSSLCredential cred = new OracleSSLCredential();
   cred.loadWallet("walletpath", "password");
   ```

   JSSE 用の新しいコード:

   ```
   KeyStore myWallet = KeyStore.getInstance("keystoretype","OraclePKI");
   FileInputStream istr = new FileInputStream("pathtowallet");
   myWallet.load(istr, password);
   ```

   `Keystore` クラスの静的な `getInstance()` メソッドは、指定されたプロバイダから、指定されたキーストア・タイプに対するキーストア・オブジェクトを作成します。プロバイダとして `OraclePKI` を使用します。

   このサンプルの場合、*keystoretype* は、`PKCS12` または `SSO` です。*pathtowallet* は、キーストアまたは Wallet のパスおよびファイル名です。*password* は、パスワードの `char[]` 配列です。SSO Wallet を使用する場合は、`null` になります。

   Oracle Java SSL で使用した `OracleSSLCredential` クラスは、JSSE では使用しません。

3. SSL ソケット・ファクトリを作成します。

   Oracle Java SSL 用の従来のコード:

   ```
   OracleSSLSocketFactory socketFactory = new OracleSSLSocketFactoryImpl();
   SocketFactory.setSSLCredentials(cred);
   ```

JSSE 用の新しいコード :

```
TrustManagerFactory tmf = TrustManagerFactory.getInstance("OracleX509");
tmf.init(trustCerts);
TrustManager[] tmA = tmf.getTrustManagers();

KeyManagerFactory kmf = KeyManagerFactory.getInstance("OracleX509");
kmf.init(trustCerts, password);
KeyManager[] kmA = kmf.getKeyManagers();

SSLContext ctx = SSLContext.getInstance("SSL");
ctx.init(kmA, tmA, null);
SSLSocketFactory factory = ctx.getSocketFactory();
```

SSL 接続がピア証明連鎖を検証できるように、信頼マネージャを作成して設定します。SSL 接続がユーザー証明書と秘密鍵にアクセスできるように、キー・マネージャを作成して設定します。キー・マネージャと信頼マネージャを使用して、SSL コンテキストを構成します。これは、新しい SSL ソケット・ファクトリを作成する際に使用されます。`SSLSocketFactory` インスタンスを作成すると、SSL ソケットを作成する際に使用できます。

JSSE への移行に関連する追加の変更

次に示す追加の変更を Oracle Java SSL と JSSE の間で行うと、アプリケーションに影響する場合があります。

- SSL ソケット・ファクトリの `createSocket()` メソッドの署名が異なる場合。（これは、Oracle Java SSL の場合は `OracleSSLSocketFactory` インスタンス、JSSE の場合は `SSLSocketFactory` インスタンスです。）ただし、`createSocket()` を直接呼び出す場合、`HTTPClient` のユーザーは必要ありません。

  Oracle Java SSL 用の従来の署名 :

  ```
  createSocket(Socket sock)
  ```

  JSSE 用の新しい署名 :

  ```
  createSocket(Socket sock, String host, int port, boolean autoClose)
  ```

  このメソッドは、既存のソケットの上に新しいソケットを作成します。既存のソケット、サーバーのホスト、サーバーのポートを指定し、作成されたソケットを閉じる際に下位のソケットを閉じるかどうかを指定します。

- SSL セッション・オブジェクトによって返されるピア証明連鎖が異なる場合。

  Oracle Java SSL の場合、`OracleSSLSession` の `getPeerCertificateChain()` メソッドは、ルート CA 証明書が最初でピア証明が最後になった証明連鎖を返します。

  JSSE の場合、`SSLSession` の `getPeerCertificates()` メソッド（推奨）または `getPeerCertificateChain()` メソッド（下位互換性のために維持）は、ピア証明が最初でルート CA 証明書が最後になった証明連鎖を返します。

Oracle Java SSL の機能（非推奨）

Oracle Java SSL は、OC4J 10.1.3.1 実装では非推奨ですが、サポートはされています。（将来のリリースではサポートされなくなる予定です。）

この項では Oracle Java SSL 固有の機能について記載していますが、16-14 ページの「Oracle Java SSL から JSSE への移行」で説明しているように、JSSE への移行をお薦めします。

この項の内容は次のとおりです。

注意： Oracle Java SSL の場合、使用する JDK によって、`jssl-1_1.jar` または `jssl-1_2.jar` のいずれかのファイルがクラスパス内に必要です。

関連項目：

- Oracle Java SSL の詳細は、『Oracle Advanced Security 管理者ガイド』を参照してください。

HTTPClient に対する SSL 実装としての Oracle Java SSL の指定

OC4J 10.1.3.1 実装の場合、JDK のデフォルトの JSSE 実装が、`HTTPClient` のデフォルトの SSL 実装になりました。（これにより、`HTTPClient` の以前のデフォルト SSL 実装であった Oracle Java SSL は、今後非推奨となります。）

ただし、次の手順を実行すると、Oracle Java SSL を `HTTPClient` のデフォルト SSL 実装として明示的に指定することができます。

1. 次のシステム・プロパティ設定を指定します。

   ```
   HTTPClient.preferOracleSSL=true
   ```

2. Oracle Java SSL クラスがクラスパス（`oracle.security.ssl.OracleSSLSocketFactory` など）内にあることを確認します。

重要： SSL ソケット・ファクトリ（`javax.net.ssl.SSLSocketFactory` 実装）が、`HTTPConnection` の `setSSLSocketFactory()` メソッドによって明示的に指定されている場合、`HTTPClient.preferOracleSSL` プロパティの設定にかかわらず、指定されたファクトリに関連付けられた SSL 実装が使用されます。

Oracle Java SSL の OracleSSLCredential クラス

Oracle Java SSL を使用するクライアント HTTPS 接続をサポートするため、Oracle Java SSL クラス `OracleSSLCredential` を使用する `HTTPConnection` クラスに複数のメソッドが追加されていました。

Oracle Java SSL の場合、サーバーとクライアントを相互に認証する際にセキュリティ資格証明を使用します。`OracleSSLCredential` は、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・ポイントをロードする際に使用されます。（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、`HTTPConnection` クラスの `setSSLCredential()` メソッドとの接続に使用されます。

Oracle Java SSL におけるセキュリティ対応アプリケーションのサポート

Oracle クライアント HTTPS では、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、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。

Oracle Java SSL での HTTPClient の使用

この項では、`HTTPClient` および Oracle Java SSL 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アプリケーションを示します。

サンプル・コード（Oracle Java SSL）

この項では、`HTTPClient` および Oracle Java SSL を使用するサンプル・コードを示します。

```
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);
        }
    }
}
```

Oracle Java SSL における SSL 資格証明の初期化

この例では、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。

1. 最初に、資格証明を作成し、Wallet をロードします。

   ```
   mycredential = new OracleSSLCredential();
   mycredential.setWallet(wallet_path, password);
   ```

2. 作成された資格証明は、`HTTPConnection` インスタンス（ここでの名前は `httpsConnection`）に、インスタンスのメソッド `setSSLCredential()` を介して渡されます。このメソッドは、次のように、最初の手順で作成された `OracleSSLCredential` インスタンスを入力として取ります。

   ```
   httpsConnection.setSSLCredential(mycredential);
   ```

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。

Oracle Java SSL でのシステム・プロパティ機能

16-8 ページの「サポートされているデフォルトのシステム・プロパティ」では、Java システム・プロパティの `keyStore`、`keyStorePassword`、`keyStoreType`、`trustStore`、`trustStorePassword`、`trustStoreType` について説明しています。この項では、Oracle Java SSL 固有の `javax.net.ssl.keyStore` に関連する次の機能を説明します。

- 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、使用される資格証明が含まれています。
- 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。

Oracle Java SSL の暗号スイートの指定

この項では、Oracle Java SSL の暗号スイートを指定する方法について説明します。

プロパティ Oracle.ssl.defaultCipherSuites

Oracle Java SSL の場合、`Oracle.ssl.defaultCipherSuites` プロパティは、暗号スイートのカンマ区切りの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
```

このプロパティは、SSL 接続を確立する前に Oracle Java SSL を使用して設定できます。このプロパティに設定する暗号スイートは、新規 HTTPS 接続で有効な暗号スイートとして使用されます。

関連項目：

- 後述の表 16-1「Oracle Java SSL でサポートされる暗号スイート」

メソッド setSSLEnabledCipherSuites()

Oracle Java SSL の場合、パッケージ `HTTPClient` の `HTTPConnection` クラスの次のメソッドを使用することにより、接続ごとに暗号スイートを設定することもできます。

- `boolean setSSLEnabledCipherSuites(String[] cipherSuites)`

これは、各配列要素で暗号スイートを指定する Java 文字列配列を取ります。現在の SSL 実装がこのメソッドをサポートするかどうかを示すブール値を返します。

注意： JSSE で HTTP 接続ごとに暗号スイートを指定する方法はありません。

関連項目：

- 追加情報は、『Oracle Application Server HTTPClient Java API Reference』（Javadoc）を参照してください。

Oracle Java SSL でサポートされる SSL 暗号スイート

Oracle Java SSL は、表 16-1 に示す暗号スイートをサポートします。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 16-1 Oracle Java SSL 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

17

Web アプリケーションのセキュリティ構成

この章では、Web アプリケーションに影響するセキュリティの問題について説明します。この章の内容は次のとおりです。

- 認証方式（auth-method）の指定
- Web アプリケーションのセキュリティ・ロールおよび制約の構成

関連項目：

- Web アプリケーションの一般情報は、『Oracle Containers for J2EE サーブレット開発者ガイド』を参照してください。
- Oracle Single Sign-On で Oracle Identity Management をセキュリティ・プロバイダとして使用する場合の関連情報は、8-13 ページの「OracleAS JAAS Provider ユーザー・コンテキストとサーブレット・セッションの同期」を参照してください。
- Web アプリケーションで使用できる JAAS モードの詳細は、5-6 ページの「JAAS モードの概要」および 5-20 ページの「JAAS モードの構成と使用」を参照してください。
- OC4J の様々な使用方法の例は、次の Web サイトを参照してください。

  `http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

認証方式（auth-method）の指定

この項では、Web アプリケーションに対して認証方式を指定する構成設定について説明します。内容は次のとおりです。

web.xml での auth-method の指定

標準の認証方式を Web アプリケーション・レベルで指定するには、web.xml の <login-config> 要素と、その <auth-method> サブ要素を使用します。次に例を示します。

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

表 17-1 は、web.xml 内の標準的な <auth-method> の設定を示しています。

表 17-1 web.xml での auth-method の値

設定	意味
BASIC	アプリケーションでは Basic 認証が使用されます。
DIGEST	アプリケーションでは Digest 認証が使用されます（外部 LDAP プロバイダやカスタム・プロバイダの場合はサポートされません）。
FORM	アプリケーションではカスタム・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 HTTPS 証明書を提供するように要求します。

注意：

- ファイルベース・プロバイダまたは Oracle Identity Management の場合は、Basic 認証よりもセキュアなソリューションとして Digest 認証をお薦めします。
- DIGEST を、セキュリティ・プロバイダが Oracle Identity Management の場合に使用するには、8-17 ページの「Digest 認証と Oracle Internet Directory の併用」に説明されている準備手順を実行する必要があります。
- FORM を使用する場合は、オプションとして、適切なクライアント・サイド・リダイレクトのための OC4J フラグを設定できます。詳細は、17-5 ページの「フォームベース認証の使用」を参照してください。（この項では、フォームベース認証に対する標準構成についても説明します。）
- CLIENT-CERT を使用するには、OracleAS JAAS Provider のプロパティ mapping.attribute も、17-6 ページの「CLIENT-CERT 認証の使用」の説明に従って構成する必要があります。

関連項目：

- 前述の認証方式の概要は、2-2 ページの「Web アプリケーションの標準認証方式」を参照してください。
- Oracle 固有の認証方式の詳細は、次項の「orion-application.xml での auth-method の指定」を参照してください。

orion-application.xml での auth-method の指定

Oracle 固有の認証方式を J2EE アプリケーション・レベルで指定するには、orion-application.xml で <jazn-web-app> 要素とその auth-method 属性を使用します。OC4J 10.1.3.1 実装でサポートされている設定は、SSO（Oracle Single Sign-On の場合）、COREIDSSO（Oracle Access Manager シングル・サインオンの場合）、CUSTOM_AUTH（Java SSO を含めたアイデンティティ管理フレームワークを使用する場合）です。これらの機能の詳細は、3-6 ページの「Oracle Application Server シングル・サインオン代替方法の概要」を参照してください。次に示すのは、Oracle Single Sign-On の場合の例です。

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

注意：

- Application Server Control を介して構成する SSO の選択肢（Oracle Single Sign-On および Java SSO）に対して、`orion-application.xml` 内で自動的に認証方式が設定されます。
- 標準の認証方式は、OC4J 固有のファイルではなく、前項の「web.xml での auth-method の指定」で説明したように、`web.xml` ファイル内で構成する必要があります。（これは以前のリリースにおける独自機能といくつかの点で異なっていますが、下位互換性を維持するため、この機能は引き続きサポートされています。）
- `<jazn-web-app>` 要素は、`orion-web.xml` ファイルでもサポートされます。競合が発生する場合は、問題の Web アプリケーションに関しては、`orion-web.xml` が `orion-application.xml` よりも優先されます。
- `orion-application.xml`（または `orion-web.xml`）内の `auth-method` の設定は、`web.xml` の設定よりも優先されます。

関連項目：

- Oracle 固有の認証方式の概要は、3-6 ページの「Oracle Application Server シングル・サインオン代替方法の概要」を参照してください。

Digest 認証モードでの Basic 認証のフォールバックの使用

OC4J 10.1.3.1 実装のデフォルトの動作として、クライアントから Basic 認証ヘッダーが送信された場合でも、Digest 認証モジュールでは Basic 認証が処理されません。（これは、10.1.3.0.0 実装のデフォルトの動作とは異なります。）クライアントから Basic 認証ヘッダーが送信された場合に Basic 認証のフォールバックを有効にするには、次の例のように、`orion-application.xml` の `<jazn>` 要素の `<property>` サブ要素で、`digest.auth.basic.fallback` プロパティを `true` に設定します。（このプロパティのロジックは、10.1.3.0.0 実装でのロジックの逆であることに注意してください。）

```
<jazn provider="XML" location="jazn-data.xml">
   <property name="digest.auth.basic.fallback" value="true" />
   ...
</jazn>
```

Basic 認証のフォールバックを使用しない場合は、このプロパティを設定しないか、または明示的に `false` に設定します。

重要： Application Server Control を介して任意の時点で任意のアプリケーションに対してファイルベース・プロバイダから Oracle Identity Management に切り替えると、そのアプリケーションの `orion-application.xml` 内にある `<jazn>` 要素が次のように置き換えられます。`<jazn>` 要素の以前の設定はすべて失われるため、設定をやりなおす必要があります。

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

フォームベース認証の使用

この項では、標準および OC4J 固有のフォームベース認証の特徴について説明します。

フォームベース認証の標準構成の設定

FORM の設定には、ログイン・ページおよびエラー・ページの URL を指定する追加構成が、web.xml の <login-config> 要素に必要になります。この機能には、OC4J 固有のものはありません。次に例を示します。

```
<login-config>
   <auth-method>FORM</auth-method>
   ...
   <form-login-config>
      <form-login-page>mylogin.jsp</form-login-page>
      <form-error-page>myerror.jsp</form-error-page>
   </form-login-config>
</login-config>
```

クライアント・サイド・リダイレクト用 OC4J フラグの設定

OC4J は、クライアント・サイド・リダイレクト用の oc4j.formauth.redirect プロパティをサポートしています。このプロパティが true に設定されている場合は、OC4J 起動時にユーザーがフォームベース認証用に資格証明を入力すると、OC4J によって適切なクライアント・サイド・リダイレクトが実行され、ブラウザにリスト表示されるリクエスト URI が変更されます。このプロパティは、次のように設定されます。

```
-Doc4j.formauth.redirect=true
```

デフォルト設定は、false です。

true に設定すると、ユーザーがユーザーとパスワードを入力して、資格証明が十分とされてフォームベース認証に成功した時点で、保護リソースの内容が表示され、ブラウザに表示されるリクエスト URI が、フォームベース認証を起動した URI と同じになります。（フォームベース認証が失敗した場合は、クライアントは前項の「フォームベース認証の標準構成の設定」で説明された構成で指定されたエラー・ページにリダイレクトされます。）

true に設定しない場合は、任意のフォームベース認証後に、OC4J 固有の j_security_check リクエスト URI がブラウザに表示されます。

例として、次のリソースがフォームベース認証で保護されていると想定します。

```
http://myhost:8888/testapp/SecureServlet
```

oc4j.formauth.redirect=true の設定でフォームベース認証が成功した場合は、保護リソースの内容が表示されるときに、前述の SecureServlet URI がブラウザに表示されます。ただし、true フラグが設定されていない場合は、ブラウザに表示されるリクエスト URI は次のようになります。

```
http://myhost:8888/testapp/j_security_check
```

CLIENT-CERT 認証の使用

この項では、HTTPS を介してクライアント認証を行うための OC4J の構成方法、およびこの認証方式に対する OC4J のロジック・フローについて説明します。

注意： Oracle Single Sign-On でクライアント証明書を使用する場合は、ここで説明する手順ではなく、『Oracle Application Server Single Sign-On 管理者ガイド』で説明されている手順に従ってください。

OC4J の CLIENT-CERT 認証用構成

HTTPS を介したクライアント認証を使用するには、次の手順を実行します。

1. 第 15 章「OC4J との SSL 通信」での説明に従って SSL を構成します。
2. `web.xml` の `<auth-method>` を、`CLIENT-CERT` に設定します。17-2 ページの「web.xml での auth-method の指定」を参照してください。
3. 必要に応じて、アプリケーションの `orion-application.xml` ファイルの `<jazn>` 要素に、OC4J `mapping.attribute` プロパティを設定します。
4. `jazn.com`（`jazn.xml` で指定されるデフォルト・レルム）以外のデフォルト・レルムを使用するには、`<jazn>` 要素の `default-realm` 属性を介して指定します。

ファイルベース・プロバイダの場合、`mapping.attribute` のデフォルト値は `CN` です。Oracle Identity Management（LDAP ベース・プロバイダ）または外部 LDAP プロバイダの場合、デフォルト値は `DN` です。次に例を示します。ファイルベース・プロバイダに対して `CN` を明示的に設定し、デフォルト・レルムも指定しています。

```
<orion-application ... >
   ...
   <jazn provider="XML" ... default-realm="myrealm" ... >
      <property name="mapping.attribute" value="CN"/>
      ...
   </jazn>
   ...
</orion-application>
```

重要：

- スタンドアロン OC4J（Oracle HTTP Server なし）で CLIENT-CERT 認証モードを使用するには、`secure-web-site.xml` の `<ssl-config>` 要素で `needs-client-auth="true"` を設定する必要があります。この属性については、15-7 ページの「secure-web-site.xml におけるオプションの手順」を参照してください。
- Application Server Control を介して任意の時点で任意のアプリケーションに対してファイルベース・プロバイダから Oracle Identity Management に切り替えると、そのアプリケーションの `orion-application.xml` 内にある `<jazn>` 要素が次のように置き換えられます。`<jazn>` 要素の以前の設定はすべて失われるため、設定をやりなおす必要があります。

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

OC4J における CLIENT-CERT 実行フロー

次に、OC4J における CLIENT-CERT 認証手順を示します。

1. ユーザーが保護リソースへのアクセスを試行します。
2. OracleAS JAAS Provider は、証明書ユーザーの識別名を証明書から取得します。
3. mapping.attribute の値に従って、OracleAS JAAS Provider は証明書ユーザーの識別名から適切な値を取得します。たとえば、属性の設定が CN の場合、OracleAS JAAS Provider は識別名から一般名を取得します。
4. OracleAS JAAS Provider は、該当するユーザー・リポジトリ（ファイルベース・プロバイダの場合は jazn-data.xml、LDAP ベース・プロバイダの場合は Oracle Internet Directory など）から、証明書ユーザーを検索します。検索フィルタは、mapping.attribute の設定によって決定されます。たとえば、属性の設定が CN の場合、一般名がユーザー・リポジトリ内の検索フィルタとなります。

   ただし、正確な動作は、ファイルベース・プロバイダと LDAP ベース・プロバイダまたは外部 LDAP プロバイダで異なる点に注意してください。たとえば、一般名が johndoe の場合、それぞれの動作は次のようになります。

   - ファイルベース・プロバイダの場合は、OracleAS JAAS Provider は、リポジトリから johndoe を検索します。
   - LDAP ベース・プロバイダまたは外部 LDAP プロバイダの場合は、OracleAS JAAS Provider は、リポジトリから cn=johndoe を検索します。
5. OracleAS JAAS Provider は、資格証明プリンシパルおよび付与されるロールをロードし、この情報を Subject インスタンスに移入します。
6. 認証がサブジェクトに対して実行されます。

Web アプリケーションのセキュリティ・ロールおよび制約の構成

この項では、ロール・タイプとロール・タイプ間のマッピング方法について説明します。

- J2EE ロールおよびセキュリティ制約の構成
- J2EE ロールへのアプリケーション・ロールのリンク
- デプロイ・ロールおよびユーザーの定義
- Web アプリケーションに対する run-as セキュリティ・アイデンティティの指定
- OC4J による J2EE ロールのデプロイ・ロールへのマッピング

J2EE ロールおよびセキュリティ制約の構成

J2EE には、移植可能なセキュリティ・ロールの機能が組み込まれています。この機能は、サーブレットと JavaServer Pages に対して、標準のデプロイメント・ディスクリプタ web.xml 内に定義されています。これらの J2EE ロールは、アプリケーションに対する一連のリソース・アクセス権を定義する際に使用されます。たとえば、sr_developers という J2EE ロールを web.xml で構成するとします。そのためには、<security-role> 要素（トップ・レベルの <web-app> 要素のサブ要素）を使用します。

```
<web-app>
   ...
   <security-role>
      <role-name>sr_developers</role-name>
   </security-role>
   ...
</web-app>
```

また、sr_developers ロールのアクセス権も web.xml で定義します。ロールは、追加の標準ディスクリプタ要素を介して（たとえば、<web-app> のサブ要素でもある <security-constraint> 要素下で）、機能および制約に関連付けられます。

```
<web-app>
   ...
   <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_developers</role-name>
      </auth-constraint>
   </security-constraint>
   ...
</web-app>
```

J2EE ロールへのアプリケーション・ロールのリンク

プリンシパルまたはアプリケーション・コードで定義した論理ロールを J2EE ロールに関連付けると、J2EE ロールの定義済アクセス権が、そのプリンシパルまたはアプリケーション論理ロールに割り当てられます。この作業は web.xml ファイルで行うため、ロール定義を変更する際にアプリケーション・コードを更新する必要はありません。アプリケーション・ロールを J2EE ロールにリンクするには、<security-role-ref> 要素（<servlet> 要素のサブ要素）を使用します。

```
<web-app>
   ...
   <servlet>
      ...
      <security-role-ref>
         <role-name>ar_developers</role-name>
         <role-link>sr_developers</role-link>
      </security-role-ref>
      ...
   </servlet>
   ...
</web-app>
```

<role-name> 要素は、アプリケーション・コードにおけるロールを意味しています。<role-link> 要素は、このアプリケーション・ロール（ar_developers）を、<security-role> 要素で定義した J2EE ロール（sr_developers）にリンクすることを指定します。

この例では、sr_developers に属するユーザーとして動作しているサーブレットが HttpServletRequest メソッド isUserInRole("ar_developers") を起動すると、メソッドは true を戻します。

注意： コンテナは、特定のセキュリティ・ロールと一致する <security-role-ref> 要素を検出できない場合、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常に <role-name> の値をチェックします。

デプロイ・ロールおよびユーザーの定義

デプロイ・ロールおよびユーザーは、使用するセキュリティ・プロバイダで定義されます。たとえばファイルベース・プロバイダの場合、デプロイ・ユーザーおよびロールは、system-jazn-data.xml ファイル（またはオプションとして、ユーザー提供の jazn-data.xml ファイル）で定義されます。

次の例では、developers デプロイ・ロールを構成しています。

```
<jazn-data>
   ...
   <jazn-realm>
      ...
      <realm>
         ...
         <roles>
            ...
            <role>
               <name>developers</name>
               <members>
                  <member>
                     <type>user</type>
                     <name>john</name>
                  </member>
               </members>
            </role>
            ...
         </roles>
         ...
      </realm>
      ...
   </jazn-realm>
   ...
</jazn-data>
```

Web アプリケーションに対する run-as セキュリティ・アイデンティティの指定

Web コンテナで、そのコンテナが認識していないユーザーへのアクセスを許可しなければならない場合があります。このような場合は、標準の Web アプリケーション・ディスクリプタで <run-as> 設定を宣言し、アクセスが許可されるロールを指定できます。

```
<servlet>
   ...
   <run-as>
      <role-name>sr_developers</role-name>
   </run-as>
   ...
</servlet>
```

ロール名は、Web アプリケーションに対してすでに定義されているロールにしてください。Web コンテナは、すべてのコールについて指定されたアイデンティティをサーブレットから EJB レイヤーに伝播する必要があります。

関連項目：

- 2-3 ページの「Web アプリケーションにおける run-as モードと伝播されたアイデンティティ」

OC4J による J2EE ロールのデプロイ・ロールへのマッピング

OC4J を使用すると、web.xml ファイルに定義されている J2EE ロールを、セキュリティ・プロバイダのデプロイ・ロールにマップできます。これは、デプロイ時に Application Server Control を介して行います。6-12 ページの「Application Server Control を介したセキュリティ・ロール・マッピングの指定」を参照してください。マッピングは、<security-role-mapping> の設定に反映されます。これは、J2EE アプリケーションの場合は orion-application.xml ファイルに、単一の Web アプリケーションの場合は orion-web.xml ファイルにあります。orion-application.xml では、<security-role-mapping> がトップ・レベルの <orion-application> 要素のサブ要素となります。同様に orion-web.xml では、これがトップ・レベルの <orion-web-app> 要素のサブ要素となります。

<security-role-mapping> 要素内の <group> サブ要素は、セキュリティ・プロバイダのロールに対応します。

次に示す orion-application.xml の構成では、J2EE ロール sr_developers（web.xml で定義）がデプロイ・ロール developers（たとえばファイルベース・プロバイダの system-jazn-data.xml で定義）にマップされています。

```
<orion-application>
   ...
   <security-role-mapping name="sr_developers">
      <group name="developers" />
   </security-role-mapping>
   ...
</orion-appliction>
```

この関連付けにより、web.xml 内で sr_developers ロールに対してアクセス可能と構成されているリソースに、developers ロールがアクセスできるようになります。

たとえば、developers ロールのメンバーであるユーザー john について考えてみます。このロールは J2EE ロール sr_developers にマップされているため、john は、sr_developers ロールからアクセス可能なアプリケーション・リソースにアクセスできます。

18

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。この章の内容は次のとおりです。

- 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
- EJB RMI クライアント・アクセスの許可
- ブラウザでのパーミッションの付与
- 匿名 EJB 検索の構成
- ORMI 用のサブジェクト伝播の有効化と構成

OC4J 10.1.3.x 実装を開始するにあたり、EJB コンテナによって OracleAS JAAS Provider がサポートされます。

関連項目：

- EJB に関する一般情報および EJB 3.0 セキュリティ・アノテーションの詳細は、『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。
- ORMI の詳細は、『Oracle Containers for J2EE サービス・ガイド』を参照してください。
- ORMIS の詳細は、（このマニュアルの）15-18 ページの「OC4J での ORMIS の有効化」を参照してください。
- EJB アプリケーションで使用できる JAAS モードの詳細は、5-5 ページの「JAAS 認可および OracleAS JAAS Provider の JAAS モード」および 5-20 ページの「JAAS モードの構成と使用」を参照してください。
- EJB とともに使用される CSIv2 の詳細は、第 19 章「Common Secure Interoperability プロトコル」を参照してください。
- OC4J の様々な使用方法の例は、次の Web サイトを参照してください。

  ```
  http://www.oracle.com/technology/tech/java/oc4j/1013/
  how_to/index.html
  ```

EJB アプリケーションの認証と認可

標準の EJB デプロイメント・ディスクリプタでセキュリティ制約および J2EE ロールを定義することで、EJB メソッドを保護できます。これらの J2EE ロールは、アプリケーションで定義したロールにリンクし、さらに必要に応じてセキュリティ・プロバイダ内のデプロイ・ロールにマップできます。(EJB の場合は、たとえば EJB がアクセスするバックエンド・データベース内のロールにデプロイ・ロールが対応している場合があります。) デプロイ・ロールへのマッピングは、デプロイ時に Application Server Control を介して実行でき (6-12 ページの「Application Server Control を介したセキュリティ・ロール・マッピングの指定」を参照)、それによって OC4J 固有のデプロイメント・ディスクリプタに適切な構成が生成されます。

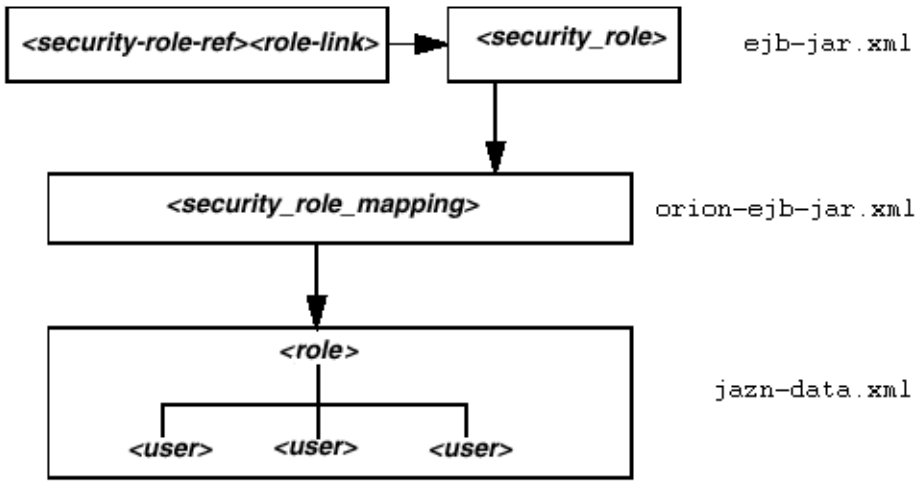
この項では、認証と認可について EJB デプロイメント・ディスクリプタ内の XML 構成に重点を置いて説明します。EJB の認可は、次のように処理されます。

- 標準の EJB デプロイメント・ディスクリプタで、J2EE 論理ロールを使用してアクセス・ルールが記述されます。
- OC4J 固有のデプロイメント・ディスクリプタで、J2EE ロールがデプロイ・ユーザーおよびデプロイ・ロール (`system-jazn-data.xml`、`jazn-data.xml`、Oracle Internet Directory などで構成) にマップされます。

> **注意:** RMI 検索認証が、JAAS カスタム・ログイン・モジュールに統合されます。ログイン・モジュールの詳細は、第 9 章「ログイン・モジュール」を参照してください。

図 18-1 は、EJB ロール定義およびロール・マッピング (この場合はファイルベース・プロバイダが対象) の概要を示しています。

図 18-1 エンドツーエンド・セキュリティ・ロール構成



EJB の認可の手順を次の各項で説明します。

- EJB デプロイメント・ディスクリプタでの J2EE ロールおよびメソッド・パーミッションの指定
- EJB メソッドのセキュリティ・チェック対象外の指定
- EJB の run-as セキュリティ・アイデンティティまたはコール元セキュリティ・アイデンティティの指定
- J2EE ロールからデプロイ・ユーザーおよびデプロイ・ロールへのマッピング
- ネームスペース・アクセスの構成

- 不明なメソッドに対するデフォルトのロール・マッピングの指定

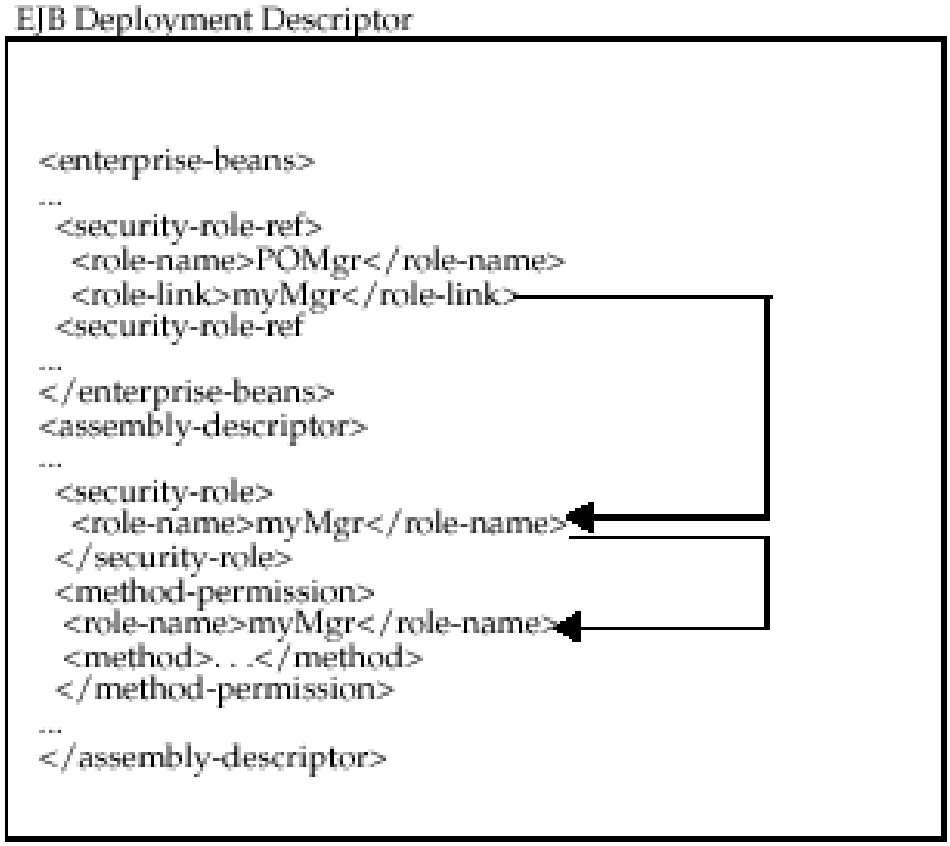
トラブルシューティングのヒント：

- アプリケーションで RMI を使用して EJB にアクセスするには、適切なユーザーまたはロールに RMI パーミッション login を付与する必要があります。18-10 ページの「EJB RMI クライアント・アクセスの許可」を参照してください。
- アプリケーションに EJB が含まれている場合は、アプリケーションのサーバー・サイド JNDI コンテキストで読取り（検索）および書込み（バインド）操作を行うためのアクセス権を、必要に応じてリモート・クライアントに付与する必要があります。18-8 ページの「ネームスペース・アクセスの構成」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 J2EE ロールおよびメソッド・パーミッションの指定

次の図 18-2 に示すように、Bean 実装内で定義されているロールの名前（`POMgr` など）を指定し、この名前を標準の EJB デプロイメント・ディスクリプタで定義されている適切な J2EE ユーザーまたはロール（`myMgr` など）にリンクすることができます。（次の手順であるデプロイ・ロールへのマッピングは、OC4J 固有のデプロイメント・ディスクリプタに反映されます（18-7 ページの「J2EE ロールからデプロイ・ユーザーおよびデプロイ・ロールへのマッピング」を参照）。）

図 18-2 セキュリティ・ロール参照



この設定を、次の手順でさらに詳しく説明します。

1. アプリケーションの論理ロール（前述の POMgr など）を、標準の EJB デプロイメント・ディスクリプタの <enterprise-beans> セクションにある <security-role-ref> 要素の <role-name> サブ要素で宣言します。（この例では、このロールに発注権限があるものと想定しています。このロールには、発注と連動するように、isCallerInRole() コールによる確認に従ってコール元をマップする必要があります。）

   <security-role-ref> の <role-link> サブ要素を使用して、アプリケーション・ロールを目的の J2EE 論理ロールにリンクします（これは、次の手順で標準の EJB デプロイメント・ディスクリプタにも定義します）。この機能によって、様々な J2EE 環境で Bean コードを変更することなくアプリケーションを使用できるようになります。アプリケーション・ロール POMgr は、J2EE ロール myMgr にリンクされます。

```
<enterprise-beans>
  ...
  <security-role-ref>
   <role-name>POMgr</role-name>
   <role-link>myMgr</role-link>
  </security-role-ref>
  ...
</enterprise-beans>
```

   （<role-link> 設定内の J2EE ロールは、デプロイ・ロールと同じにするか、または後述の手順でデプロイ・ロールにマップできます。）

   > **注意：** <security-role-ref> 要素は、Bean 内でセキュリティ・コンテキスト・メソッドを使用する場合のみ必要です。

2. 標準の EJB デプロイメント・ディスクリプタで、J2EE ロールおよびそのロールがパーミッションを持つ EJB メソッドを定義します。この発注の例では、Bean PurchaseOrder 内で実行されるあらゆるメソッドが、自己を myMgr として認可している必要があります。これは、<security-role> の <role-name> サブ要素で宣言される J2EE ロールです。この J2EE ロールは、前の手順でアプリケーション・ロール POMgr にリンクされたものです。PurchaseOrder は、<session> または <entity> 要素のサブ要素である、<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 myMgr を定義し、このロールに EJB PurchaseOrder Bean のすべてのメソッド（* 記号で指定）にアクセスするためのパーミッションを付与しています。

```
<assembly-descriptor>
 ...
 <security-role>
  <description>Role for 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 を実行すると、Bean 実装内で POMgr を参照できるようになり、OC4J により POMgr が myMgr にマップされます。

注意： 同じ EJB 内でメソッドの <method-permission> 要素に異なるロールを指定すると、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> 要素をさらに詳しく見てみると、<method> サブ要素を使用して、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・ロールが指定されています。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。

- 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>
```

- 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。

- 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>java.lang.String</method-param>
      <method-param>java.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> 要素にサブ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、次のように標準の EJB デプロイメント・ディスクリプタを使用して、そ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、空の <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。

EJB の run-as セキュリティ・アイデンティティまたはコール元セキュリティ・アイデンティティの指定

標準の EJB デプロイメント・ディスクリプタで、EJB のすべてのメソッドが特定のアイデンティティを使用して実行されるように指定できます。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・アイデンティティを使用する 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アイデンティティをセキュリティ・アイデンティティ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・アイデンティティを指定します。次の例は、POMgr ロールの下ですべてのエンティティ Bean メソッドが実行されることを示しています。

```
<enterprise-beans>
 <entity>
  ...
  <security-identity>
     <run-as>
        <role-name>POMgr</role-name>
     </run-as>
  </security-identity>
  ...
 </entity>
 ...
</enterprise-beans>
```

また、次の例に、Bean のすべてのメソッドをコール元のアイデンティティ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>
```

J2EE ロールからデプロイ・ユーザーおよびデプロイ・ロールへのマッピング

前述したように、標準の EJB デプロイメント・ディスクリプタで J2EE ロールおよび関連するセキュリティ制約を定義して、EJB メソッドを保護できます。定義した J2EE ロールは、セキュリティ・プロバイダで定義されているデプロイ・ユーザーおよびロールにマップできます。このセキュリティ・ロールのマッピングは、6-12 ページの「Application Server Control を介したセキュリティ・ロール・マッピングの指定」で説明しているように、デプロイ中に Application Server Control を使用して行えます。

マッピングの反映先は、この後の説明で示すように、Oracle 固有のディスクリプタ内の `<security-role-mapping>` 設定です。

関連項目：

- `orion-application.xml` ファイルの詳細は、『Oracle Containers for J2EE 開発者ガイド』を参照してください。
- `orion-ejb-jar.xml` ファイルの詳細は、『Oracle Containers for J2EE Enterprise JavaBeans 開発者ガイド』を参照してください。

ロールのマッピングには Application Server Control を使用することが推奨されますが、次の説明で示すのは、J2EE ロール `myMgr` をデプロイ・ロール `managers` にマップしたときに `orion-ejb-jar.xml` に生成される構成の参考情報です。`managers` ロールのメンバーとしてログインできるユーザーは、`myMgr` ロール（前に `POMgr` アプリケーション論理ロールにリンクされていたロール）のパーミッションを持つものとみなされ、`PurchaseOrder` Bean のメソッドを実行することができます。

標準の EJB デプロイメント・ディスクリプタ：

```
<assembly-descriptor>
   ...
   <security-role>
      <role-name>myMgr</role-name>
   </security-role>
   <method-permission>
      <role-name>myMgr</role-name>
      <method>...</method>
   </method-permission>
   ...
</assembly-descriptor
```

OC4J 固有のデプロイメント・ディスクリプタ：

```
<assembly-descriptor>
 ...
 <security-role-mapping name="myMgr">
  <group name="managers" />
 </security-role-mapping>
 ...
</assembly-descriptor>
```

特定のユーザーにマップする場合：

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

特定のロール内の特定のユーザーにマップする場合：

```
<security-role-mapping name="myMgr">
 <group name="managers" />
 <user name="guest" />
</security-role-mapping>
```

ネームスペース・アクセスの構成

アプリケーションに EJB が含まれている場合は、アプリケーションのサーバー・サイド JNDI コンテキストでオブジェクトの読取り（検索）および書込み（バインド）を行うためのネームスペース・アクセス権を、必要に応じてリモート・クライアントに付与する必要があります。読取りおよび書込みは、javax.naming.Context オブジェクトの lookup() メソッドと bind() メソッドにそれぞれ対応します。

リモート・クライアントのユーザー資格証明（リモート・クライアント・コンテキストに渡される JNDI プロパティ）を、アプリケーションの JNDI コンテキストへのアクセス権を付与されているロールの 1 つにマップする必要があります。

orion-application.xml から抜粋した次の例では、読取り操作を行うためのネームスペース・アクセス権を managers および developers ロールに付与する方法を示しています。

```
<orion-application ... >
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="sr_developer">
               <group name="developers"/>
            </security-role-mapping>
            <security-role-mapping name="myMgr">
               <group name="managers"/>
            </security-role-mapping>
         </namespace-resource>
      </read-access>
   </namespace-access>
   ...
</orion-application>
```

ここでは、示されたロール・マッピングがすでに orion-application.xml 内で設定されていることを前提にしています。

不明なメソッドに対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。

```
<assembly-descriptor>
   ...
   <default-method-access>
      <security-role-mapping name="&lt;default-ejb-caller-role&gt;"
                             impliesAll="true" />
   </default-method-access>
   ...
</assembly-descriptor>
```

デフォルト・ロールは、name 属性で指定されている <default-ejb-caller-role> です。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。true の設定は、セキュリティ・ロール・チェックが行われないことを示します。false の設定は、コンテナがこれらのメソッドでこのデフォルト・ロールをチェックすることを示します。

orion-ejb-jar.xml ファイルでは、impliesAll 属性がデフォルトで次のように設定されます。

- orion-ejb-jar.xml で <security-role-mapping> が指定され、かつ impliesAll が設定されていない場合は、この属性がデフォルトで false に設定され、コンテナがこれらのメソッドでこのデフォルト・ロールをチェックします。
- orion-ejb-jar.xml で <security-role-mapping> が指定されていない場合は、OC4J EJB レイヤーがデフォルトで impliesAll に true を設定し、これらのメソッドに対してセキュリティ・ロール・チェックが行われません。

impliesAll 属性が false の場合は、<user> または <group> サブ要素を介して、name 属性に定義されているデフォルト・ロールをデプロイ・ユーザーまたはロール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 ロール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>
```

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・クライアントは、その資格証明を client.jar（標準の J2EE クライアント・モジュール）にデプロイされている jndi.properties ファイルで定義します。
- 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。

> **注意：** RMI 検索認証が、JAAS カスタム・ログイン・モジュールに統合されます。ログイン・モジュールの詳細は、第 9 章「ログイン・モジュール」を参照してください。

JNDI プロパティ内の資格証明

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・クライアントは、その資格証明を、クライアントのコードでデプロイされている jndi.properties ファイルで次のプロパティを使用して定義します。

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

たとえば、リモート EJB に POMGR/welcome としてアクセスする場合は、次のようにプロパティを設定します。java.naming.factory.initial 設定は、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.naming.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");
```

重要： jndi.properties ファイルは、クラスパスからアクセスできる必要があります。

初期コンテキスト内の資格証明

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 javax.naming.InitialContext インスタンス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、次のようにします。

```
Hashtable env = new Hashtable();
env.put("java.naming.provider.url", "ormi://myhost/ejbsamples");
env.put("java.naming.factory.initial",
        "oracle.j2ee.naming.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。

EJB RMI クライアント・アクセスの許可

アプリケーションで RMI を使用して EJB にアクセスするには、適切なユーザーまたはロールに RMI パーミッション login を付与する必要があります。これを行うには、OracleAS JAAS Provider の Admintool を使用します。

次の例では、このパーミッションをロール（users）に対して設定します。

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

さらに次の例では、ユーザー（JDOE_ENDUSER）に対してパーミッションを設定します。

```
% java -jar jazn.jar -grantperm myrealm -user JDOE_ENDUSER \
       com.evermind.server.rmi.RMIPermission login
```

ファイルベース・プロバイダの場合は、Application Server Control でロールを選択し、RMI 権限の付与チェック・ボックスを選択しても、ロールにこのパーミッションを付与できます。（7-7 ページの「ロールの作成」または 7-8 ページの「ロールの編集」も参照してください。）

OC4J を再起動して、変更を有効にします。

関連項目：

- 付録 C「OracleAS JAAS Provider Admintool リファレンス」
- 5-16 ページの「system-jazn-data.xml でのポリシー構成」

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";
```

匿名 EJB 検索の構成

匿名 EJB 検索は、ほとんどの場合、開発中または非常に特別な状況においてのみ検討対象となるモードです。このモードでは、`InitialContext` の作成時にプリンシパルおよび資格証明を指定しないので、リモートで EJB にアクセスする際にプリンシパルや資格証明を指定する必要がありません。`jndi.properties` ファイルは次のようになります。

```
java.naming.factory.initial=
   oracle.j2ee.naming.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://localhost:23791/ejb30slsb
java.naming.security.principal=
java.naming.security.credentials=
```

重要： このモードを使用すると EJB が完全にセキュアでなくなるため、一般には使用しないことをお薦めします。

このモードは、次のようにして有効にできます。

1. 4-12 ページの「事前定義アカウント」の説明に従い、`system-jazn-data.xml` に `anonymous` ユーザーが構成され、このユーザーがアクティブになっていることを確認します。
2. また、18-10 ページの「EJB RMI クライアント・アクセスの許可」の説明に従い、`system-jazn-data.xml` 内の適切なレルム下で、RMI パーミッションが付与されているロールに `anonymous` ユーザーを割り当てます。たとえば、`users` ロールに RMI パーミッションが付与されていると仮定します。

```
<jazn-data>
   ...
   <jazn-realm>
      <realm>
         <name>myrealm</name>
         ...
         <roles>

            <role>
               <name>users</name>
               <members>
                  <member>
                     <type>user</type>
                     <name>anonymous</name>
                  </member>
               </members>
            </role>
            ...
         </roles>
         ...
      </realm>
      ...
   </jazn-realm>
   ...
</jazn-data>
```

3. ロール（この例では users）に適切なネームスペース・アクセス権を付与して、このロールがアプリケーションのサーバー・サイド JNDI コンテキストで読取り（検索）および書込み（バインド）操作を実行できるようにします。アプリケーションの orion-application.xml ファイルで、次のような構成を使用します。

```
<orion-application>
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="jndi-user-role">
               <group name="administrators" />
               <group name="users" />
            </security-role-mapping>
         </namespace-resource>
      </read-access>
      <write-access>
         <namespace-resource root="">
            <security-role-mapping name="jndi-user-role">
               <group name="administrators" />
               <group name="users" />
            </security-role-mapping>
         </namespace-resource>
      </write-access>
   </namespace-access>
   ...
</orion-application>
```

この構成では、プリンシパルや資格証明を指定しなくてもリモート EJB にアクセスできます。

ORMI 用のサブジェクト伝播の有効化と構成

この項では、OC4J でのサブジェクト伝播、および ORMI を使用するサブジェクト伝播を有効にする方法について説明します。（サブジェクト伝播では、CSIv2 仕様に従って、常に IIOP が使用されます。）内容は次のとおりです。

- OC4J でのサブジェクト伝播の概要
- ORMI 用のサブジェクト伝播の有効化
- サブジェクト伝播用プリンシパル・クラスの共有
- サブジェクト伝播制限の削除と構成

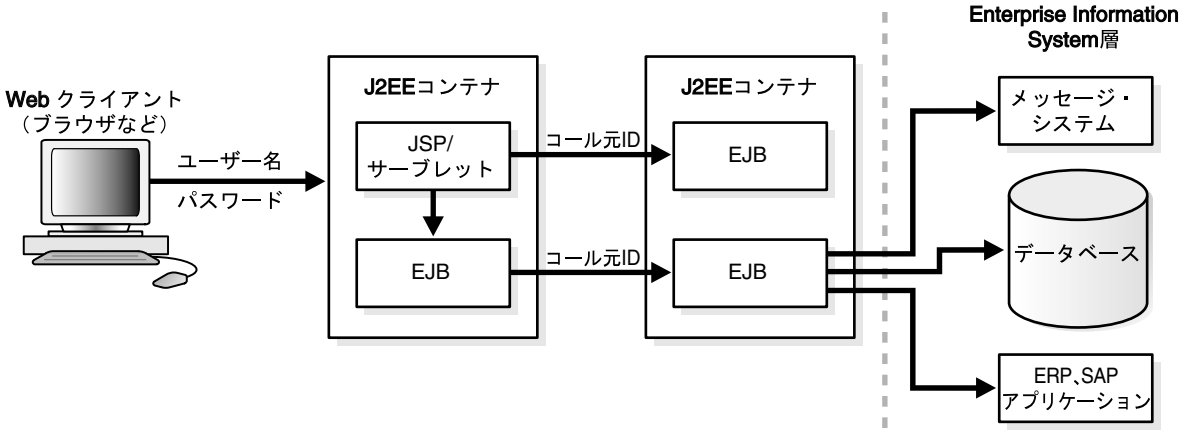
重要：

- サブジェクト伝播は強力な機能なので、サーバーが信頼できないクライアント・アクセスからセキュアになっている環境でのみ使用してください。このため、クライアント・リクエストの整合性が保証されるように、適切な防御手段を講じた上で、この機能を本番環境で使用することをお薦めします。たとえば、アプリケーション・ファイアウォールやネットワーク・ファイアウォール、RMI アクセス制限（15-21 ページの「ORMIS アクセス制限の構成」で説明しているように、rmi.xml の <access-mask> 要素を使用）、または RMI サブジェクト伝播制限（18-15 ページの「サブジェクト伝播制限の削除と構成」で説明している rmi.xml の <subject-propagation-mask> 要素を使用）を使用することを検討してください。
- サブジェクト伝播は、OC4J 10.1.3.x インスタンス間でのみサポートされています。

OC4J でのサブジェクト伝播の概要

OC4J では、図 18-3 で概要を説明しているように、サブジェクト伝播がサポートされています。この機能を使用することで、Web クライアントはそのアイデンティティの認証をサーブレットから受け、サーブレットはそのアイデンティティを使用して他の EJB やサーブレットと通信できます。この場合、アイデンティティは適切なサブジェクト（`javax.security.auth.Subject` インスタンス）です。同様に、リモート EJB ファット・クライアントは、この機能を使用して EJB コンテナをコールできます。

図 18-3 サブジェクト伝播



クライアントの現在のサブジェクトが取得された後、`Subject.getSubject()` コールを介して、サブジェクト伝播は次のように行われます。

1. サブジェクトが RMI サーバーにシリアライズされます。
2. RMI サーバーでは、サブジェクトがデシリアライズされ、サーバー・サイド JAAS コンテキストの設定に使用されます。

サブジェクトは、たとえば一連の EJB 起動によって伝播できます。EJB は、クライアントのアイデンティティを使用するように構成されている場合は、クライアントのアイデンティティを取り込みます。EJB は、特定のロールで run-as モードを使用するようには構成できません。

ORMI 用のサブジェクト伝播の有効化

サブジェクト伝播を使用するには、次の手順を実行する必要があります。

1. サブジェクト伝播システム・プロパティの設定
2. JAAS モードの有効化
3. サブジェクト伝播のための RMI パーミッションの付与

サブジェクト伝播システム・プロパティの設定

OC4J では、ORMI を使用するサブジェクト伝播は、クライアントおよびサーバーの両方で有効にした場合にのみ使用できます。（CSIv2 仕様に従い、IIOP に対しては常にサブジェクト伝播が有効となります。）そのためには、クライアント上およびサーバー上で次のシステム・プロパティ設定を使用します。

```
-Dsubject.propagation=true
```

現行リリースでは、この設定により、グローバル OC4J レベルでサブジェクト伝播が設定されます。

JAAS モードの有効化

サブジェクト伝播が正しく行われるようにするには、サブジェクトの伝播元となる Web アプリケーションおよび伝播先となる EJB で JAAS モードを有効にしておく必要があります。そこで、5-6 ページの「JAAS モードの概要」で説明されているように、`orion-application.xml` ファイルの `<jazn>` 要素で、`jaas-mode="doAs"` または `jaas-mode="doAsPrivileged"` の設定が必要になります。

サブジェクト伝播のための RMI パーミッションの付与

伝播されたサブジェクトがサーバーによって受け入れられるのは、EJB コール元に `RMIPermission subject.propagation` が付与されている場合に限られます。次の例では、OracleAS JAAS Provider の Admintool を使用して、このパーミッションをユーザー `oc4jadmin` に付与して、このユーザーのサブジェクトがサーバーに伝播されるようにします。

```
% java -jar jazn.jar -grantperm myrealm -user oc4jadmin \
      com.evermind.server.rmi.RMIPermission subject.propagation
```

`subject.propagation` パーミッションは、ロールに付与することもできます。次の例では、`users` ロール内のあらゆるユーザーのサブジェクトがサーバーに伝播されるようにします。

```
% java -jar jazn.jar -grantperm myrealm -role users \
      com.evermind.server.rmi.RMIPermission subject.propagation
```

サーバーがサブジェクト内で受け入れるプリンシパル名を指定することで、サブジェクト伝播を制限できます。次の例では、再び `users` ロール内のユーザーに `subject.propagation` パーミッションを付与しますが、今度は伝播されたサブジェクトから `developer` および `manager` プリンシパルを除くすべてがサーバーによって排除されます。（この選別は、プリンシパル・タイプではなくプリンシパル名によって行われることに注意してください。）

```
% java -jar jazn.jar -grantperm myrealm -role users \
      com.evermind.server.rmi.RMIPermission subject.propagation developer,manager
```

デフォルトでは、制限なしにサブジェクト内のすべてのプリンシパルが受け入れられます。ただし、`subject.propagation` パラメータを `"*"`（引用符も含めて指定）に設定することで、この動作を明示的に指定することもできます。

```
% java -jar jazn.jar -grantperm myrealm -user oc4jadmin \
      com.evermind.server.rmi.RMIPermission subject.propagation "*"
```

これは、前述の最初の例に相当します。

サブジェクト伝播用プリンシパル・クラスの共有

`java.security.Subject` が JDK に付属のクラスであるのに対し、`java.security.Principal` は必要に応じて実装可能なインタフェースです。サブジェクト伝播が ORMI で正しく行われるようにするには、リモート・クライアント、アプリケーションおよび OC4J のすべてに、すべての `Principal` クラス定義へのアクセス権がある必要があります。

このようにするには、6-15 ページの「ライブラリを共有するためのタスク」で説明されているように、OC4J 共有ライブラリとしてロードされているライブラリにこれらのクラス定義を追加します。

サブジェクト伝播制限の削除と構成

デフォルトでは、すべての ORMI クライアントが、サブジェクト伝播へのアクセスを拒否されています。アクセスが可能になるように構成するには、rmi.xml 内の <subject-propagation-mask> 要素およびそのサブ要素 <host-access> と <ip-access> の設定を使用します。

サブジェクト伝播へのアクセスは、除外的または包含的にできます。

- 除外モードでは、明示的に指定されているものを除くすべての IP アドレスまたはホストからのアクセスが拒否されます。<subject-propagation-mask> で mode="deny" を使用してから、<host-access> サブ要素、<ip-access> サブ要素またはこの両方で mode="allow" を使用することで、アクセスを許可する特定のホストまたは IP アドレスを指定します。
- 包含モードでは、明示的に除外されているものを除くすべての IP アドレスまたはホストからのアクセスが可能になります。<subject-propagation-mask> で mode="allow" を使用してから、<host-access> サブ要素、<ip-access> サブ要素またはこの両方で mode="deny" を使用することで、アクセスを拒否するホストまたは IP アドレスを指定します。

次の例は、除外モードを構成して、localhost および 192.168.1.0 に対してのみサブジェクト伝播を許可しています。(255.255.255.0 は適用されるサブネット・マスクです。)

```
<rmi-server ... >
   ...
   <subject-propagation-mask default="deny">
      <host-access domain="localhost" mode="allow"/>
      <ip-access ip="192.168.1.0" netmask="255.255.255.0" mode="allow"/>
   </subject-propagation-mask>
   ...
</rmi-server>
```

デフォルト設定は、次のとおりです。

```
<subject-propagation-mask default="deny"/>
```

19

Common Secure Interoperability プロトコル

OC4J では、Common Secure Interoperability Version 2 プロトコル（CSIv2）がサポートされます。CSIv2 は、認可とアイデンティティ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、EJB とともに使用することができます。

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。

CSIv2 構成には、次の 3 つのファイルが関係します。

- `internal-settings.xml`（サーバー・サイド）
- `ejb_sec.properties`（クライアント・サイド）
- `orion-ejb-jar.xml`

この章の構成は次のとおりです。

internal-settings.xml 内の CSIv2 セキュリティ・プロパティ（EJB サーバー）

internal-settings.xml ファイルで、<sep-property> 要素内の属性値を使用して CSIv2 のサーバー・セキュリティ・プロパティを指定します。

表 19-1 はプロパティのリストです。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 19-1 EJB サーバーのセキュリティ・プロパティ

プロパティ	説明
port	IIOP ポート番号（デフォルトは 5555）。
ssl	true の設定は、IIOP/SSL をサポートすることを指定します。
ssl-port	IIOP/SSL ポート番号（デフォルトは 5556）。このポートは、サーバー・サイド認証専用です。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 も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	キーストアの名前とパス（ssl が true の場合にのみ使用）。絶対パスが推奨されます。
keystore-password	キーストアのパスワード（ssl が true の場合にのみ使用）。
trusted-clients	アイデンティティ・アサーションを信頼できるホストのカンマ区切りのリスト。このリストの各エントリには、IP アドレス、ホスト名、ホスト名のパターン（たとえば *.example.com）、または *（* 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示す）を指定できます。デフォルトでは、信頼できるクライアントはありません。
truststore	トラストストアの名前とパス。絶対パスが推奨されます。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 の場合にのみ使用）。

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。次の点に注意してください。

- 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> エンティティを構成していても、CSIv2 のコンポーネント・タグは作成されません。

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> 要素内にサーバー認証ポート番号を指定します。OC4J では、サーバーおよびクライアント認証ポート番号用にこれに 1 が加算されます。

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。

次の例に、一般的な `internal-settings.xml` ファイルを示します。

```
<server-extension-provider name="IIOP"
        class="com.oracle.iiop.server.IIOPServerExtensionProvider">
    <sep-property name="port" value="5555" />
    <sep-property name="host" value="localhost" />
    <sep-property name="ssl" value="true" />
    <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：

- `internal-settings.xml` は Application Server Control では更新できません。
- ここで `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。
- OC4J が Oracle Application Server 環境で Oracle Process Manager and Notification Server（OPMN）により起動されると、`internal-settings.xml` ファイルに指定したポートは上書きされます。IIOP SSL ポートは、キーストアの場所、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、該当する OPMN のログ・ファイルを確認し、失敗の正確な原因を調査してください。
- OPMN が特定の OC4J インスタンスに対して IIOP を無効にするように構成されている場合は、`internal-settings.xml` で IIOP を有効化する設定が無効になります。（OPMN を介した IIOP の有効化と無効化の詳細は、『Oracle Containers for J2EE サービス・ガイド』の RMI に関する章を参照してください。）
- 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 環境の両方について `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。

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*)>
```

ejb_sec.properties 内の CSIv2 セキュリティ・プロパティ（EJB クライアント）

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。次の表 19-2 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/home/config 内で検索します。システム・プロパティ設定 -Dejb_sec_properties_location=*pathname* を使用して、このファイルの場所を明示的に指定できます。

表 19-2 EJB クライアントのセキュリティ・プロパティ

プロパティ	説明
oc4j.iiop.keyStoreLoc	キーストアのパスと名前。絶対パスが推奨されます。
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と名前。絶対パスが推奨されます。
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。
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、ホスト名のパターン（たとえば *.example.com）、または *（* のみの場合は、すべてのサーバーが信頼できることを示す）を指定できます。

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・ランタイムがサブジェク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。

クライアントがクライアント・サイド SSL 認証を使用する場合、サーバーはクライアントの証明書から DN を抽出し、それを対応するセキュリティ・プロバイダ内で検索します。パスワード認証は実行されません。

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・アイデンティティを委任するアイデンティティ・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（19-2 ページの「internal-settings.xml 内の CSIv2 セキュリティ・プロパティ（EJB サーバー）」を参照）。

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。

OC4J では、両方の種類の信頼がサポートされます。信頼関係は、`orion-ejb-jar.xml` 内の `<ior-security-config>` 要素を使用して構成します。これについては次項の「orion-ejb-jar.xml 内の CSIv2 のセキュリティ・プロパティ」で説明します。

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。
- SSL クライアント認証を要求するようにサーバーを構成し、アイデンティティ・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。

---

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` ファイル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>` サブ要素、`<as-context>` サブ要素、および `<sas-context>` サブ要素が含まれます。

`<ior-security-config>` 要素の DTD は次のとおりです。

```
<!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) >
```

この項の以降の部分で、次の要素について説明します。

- <transport-config> 要素
- <as-context> 要素
- <sas-context> 要素

<transport-config> 要素

この要素には、トランスポートのセキュリティ・レベルを指定します。<transport-config> の各サブ要素は、supported、required または none に設定する必要があります。設定 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。各サブ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。

注意 :

- <establish-trust-in-client> を required に設定すると、<as-context> 内の username_password を <auth-method> とする設定は無視されます。この場合は、<as-context> セクションの <required> 要素も 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> 要素

この要素では、メッセージ・レベルの認証プロパティを指定します。各サブ要素を次に示します。

- <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>: 現行の実装では、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> 要素

この要素では、アイデンティティ委任プロパティを指定します。この要素には 1 つのサブ要素 <caller-propagation> があり、次のように supported、required または none に設定できます。

- 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 はアイデンティティの委任をサポートしません。

例 : <ior-security-config>

次の例では、<ior-security-config> 要素とそのサブ要素を使用しています。

```
<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>
```

20

リソース・アダプタのセキュリティ・サポート

この章では、Enterprise Information System（EIS）接続にリソース・アダプタを使用するときの、セキュリティ上の注意事項とセキュリティおよび認証の構成方法について説明します。内容は次のとおりです。

関連項目：

- リソース・アダプタと J2EE Connector Architecture の詳細は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』を参照してください。

EIS 接続のセキュリティおよび認証設定の概要

J2EE アプリケーションと EIS 間での通信の安全を保証するため、J2EE Connector Architecture では、アプリケーション・コンポーネントが、EIS への確立済接続に JAAS サブジェクトを関連付けできるようになっています。これを実現できるようにするため、J2EE Connector Architecture のセキュリティ規約は、標準の JAAS でも使用できるようになっています。次の各項で、概要を示します。

J2EE Connector Architecture のセキュリティ規約の概要

J2EE Connector Architecture のセキュリティ規約は、アプリケーション・サーバーとリソース・アダプタ間のものであり、接続管理規約を安全な接続に関する機能で拡張したものです。このセキュリティ規約は、標準の JAAS インタフェースをサポート対象としているため、特定のセキュリティ・フレームワークまたはメカニズムに依存しないようになっています。具体的には、このセキュリティ規約には次の機能が含まれています。

- サブジェクトを、J2EE コンポーネントからリソース・アダプタに直接伝播する機能（コンポーネント管理サインオン用）
- サブジェクトを、アプリケーション・サーバーからリソース・アダプタに伝播する機能（コンテナ管理サインオン用）

このセキュリティ規約のサポート対象となる具体的な認証メカニズムは、次の 2 つです。

- 一般的に使用されるパスワード付きの Basic メカニズム : ユーザー名とパスワードのペアを使用するものです。このペアはパスワード資格証明オブジェクトに格納されます。このオブジェクトは、アプリケーション・サーバーにより、認証のためリソース・アダプタに渡されます。
- Kerberos バージョン 5 メカニズム（略称、Kerbv5）: マサチューセッツ工科大学によって配布されている認証プロトコルです。このメカニズムでは、Kerberos チケットなどの資格証明情報をカプセル化する汎用資格証明オブジェクトを使用します。このオブジェクトは、アプリケーション・サーバーにより、検証のためリソース・アダプタに渡されます。

セキュリティ規約にある機能で使用される主要インタフェースは、次のとおりです。

- `javax.security.auth.Subject`
- `java.security.principal`
- `javax.security.auth.spi.LoginModule`
- `javax.resource.spi.security.PasswordCredential`

  この J2EE Connector Architecture クラスは、パスワード付きの Basic 認証用のユーザー名とパスワードのペアを表します。
- `org.ietf.jgss.GSSCredential`（J2SE バージョン 1.4）

  このインタフェースは、Kerberos バージョン 5 認証に使用できる汎用資格証明オブジェクトを表します。（これは、非推奨の J2EE Connector Architecture `javax.resource.spi.security.GenericCredential` インタフェースにかわるものです。）

注意 : 再認証がサポートされるようにするには、リソース・アダプタの `ra.xml` ファイルにある `<reauthentication-support>` 要素に `true` の値を設定します。この場合、管理対象の接続を初めに作成したときに使用したサブジェクトとは異なるサブジェクトを持つ接続リクエストに対しても、初めの管理対象の接続を再利用できるようになります。

コンポーネント管理サインオンとコンテナ管理サインオンの比較概要

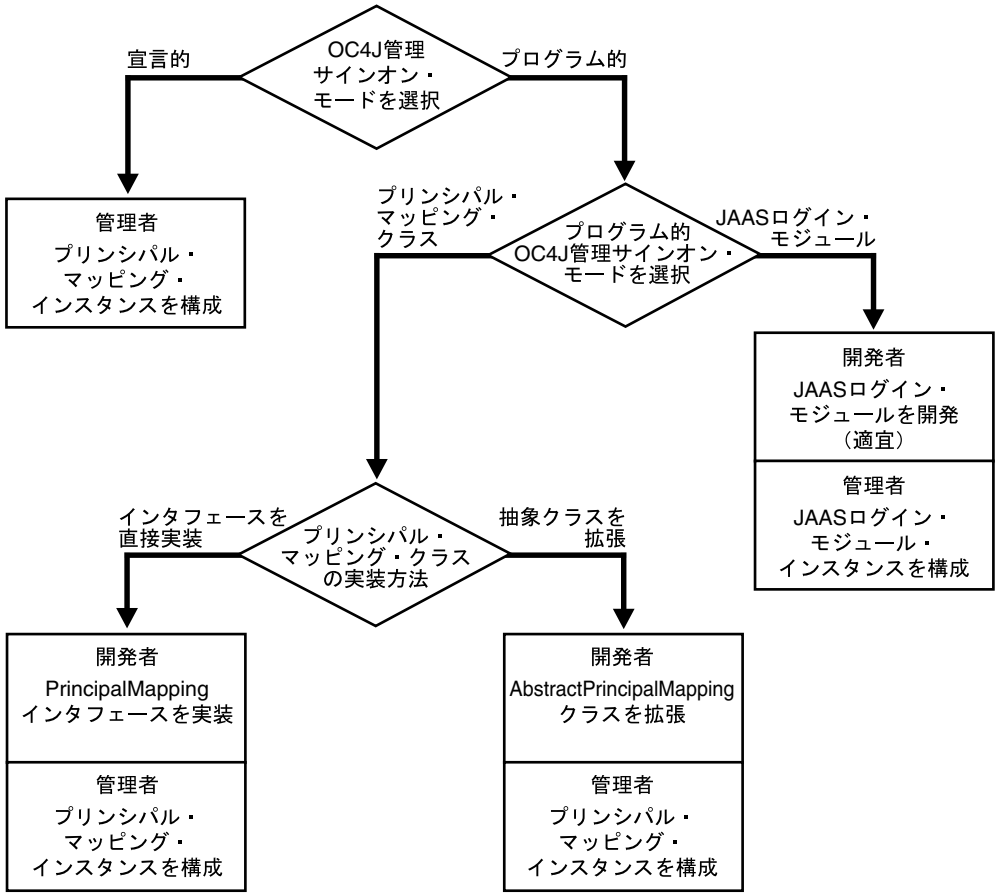
J2EE アプリケーションから EIS へのサインオンは、アプリケーション・コンポーネントまたは J2EE コンテナ（OC4J）によって管理できます。コンポーネント管理サインオンはプログラムで設定する必要があり、OC4J 固有の構成を伴いません。コンテナ管理サインオンは、プログラミングの必要のない、OC4J 固有の構成を使用した宣言的な方法か、または OC4J 固有の構成とプログラミングを組み合せたプログラム的な方法により、設定できます。プログラムによるコンテナ管理サインオンでは、プリンシパル・マッピング・クラスまたは JAAS ログイン・モジュールを使用できます。

次のリストは、コンポーネント管理サインオンとコンテナ管理サインオンに必要な設定のオプションとタイプをまとめたものです。各レベルの箇条書き記号は選択肢を表します。

- コンポーネント管理サインオン : `web.xml` または `ejb-jar.xml` 内の `Application` の `<res-auth>` 設定が必要です。サインオンの設定はプログラムで行います。OC4J 固有の構成はありません。
- コンテナ管理サインオン : `web.xml` または `ejb-jar.xml` 内の `Container` の `<res-auth>` 設定が必要です。サインオンの設定は宣言またはプログラムによって行います。次に示すように、各コンテナ管理サインオン・モードで OC4J 固有の構成を使用します。
  - なし : コンポーネント管理サインオンまたはセキュリティなしを意味します。Application Server Control を介してコンテナ管理サインオンのセキュリティを無効にすることで指定します（20-9 ページの「宣言的コンテナ管理サインオンの使用」を参照）。`oc4j-ra.xml` ファイルの `<security-config>` 要素に `use="none"` として反映されます。
  - 宣言的 : プリンシパル・マッピング・エントリを介した OC4J 構成。Application Server Control を介してコンテナ管理サインオンのセキュリティを有効にすることで指定します（「宣言的コンテナ管理サインオンの使用」を参照）。`<security-config>` 要素に、`use="principal-mapping-entries"` として、該当するサブ要素とともに反映されます。
  - プログラム的（プリンシパル・マッピング・クラスまたは JAAS ログイン・モジュールを使用）：
    - プリンシパル・マッピング・クラス :（パッケージ `oracle.j2ee.connector` において)`PrincipalMapping` インタフェースを直接実装するか `AbstractPrincipalMapping` クラスを拡張します。`oc4j-ra.xml` ファイルで直接（Application Server Control のサポートなし）、`<security-config>` 要素に、`use="principal-mapping-interface"` を、該当するサブ要素とともに構成します。
    - JAAS ログイン・モジュール : JAAS ログイン・モジュールを使用します。`oc4j-ra.xml` ファイルで直接（Application Server Control のサポートなし）、`<security-config>` 要素に、`use="jaas-module"` を、該当するサブ要素とともに構成します。

OC4J のコンテナ管理サインオン用の選択肢は、次のように図 20-1 にも図解されています。

図 20-1 OC4J のコンテナ管理サインオン用の選択フロー・チャート



セキュリティ関連リソース・アダプタ構成要素の概要

この項では、次の主要リソース・アダプタのセキュリティ構成要素について説明します。

- oc4j-ra.xml ファイルの <security-config> 要素
- oc4j-connectors.xml ファイルの <security-permission> 要素

関連項目：

- ここに記載されているファイルと要素の追加情報は、『Oracle Containers for J2EE リソース・アダプタ管理者ガイド』を参照してください。

oc4j-ra.xml ファイルの <security-config> 要素

`oc4j-ra.xml` ディスクリプタにより、OC4J 固有のデプロイ情報（JNDI パス名とコネクタ・プロパティ）がリソース・アダプタに提供されます。`oc4j-ra.xml` には、各リソース・アダプタ用に、構成パラメータ値のセットに対応する JNDI 名を指定する 1 つ以上の `<connector-factory>` 要素が含まれています。OC4J では、各接続を `ConnectionFactory` インスタンスとして、適正な JNDI ネームスペース・ロケーションにバインドします。

`<connector-factory>` 要素には、EIS にユーザー名とパスワードを供給する方法を記述した `<security-config>` 要素（オプション）を含めることができます。

`<security-config>` 要素には、コンテナ管理サインオンのユーザー名とパスワードを指定します。

oc4j-ra.xml ファイルの <security-config> 要素にこの情報を指定する方法は、2 つあります。

- マッピング対象のサブ要素を <principal-mapping-entries> サブ要素に明示的に指定します。
- ユーザー作成マッピング・クラスの名前を指定します。このクラスは、oracle.j2ee.connector.PrincipalMapping を実装するか、(<principal-mapping-interface> サブ要素内の) oracle.j2ee.AbstractPrincipalMapping を継承するものです。

認証問題の詳細は、20-8 ページの「コンテナ管理サインオンでの認証」を参照してください。ここでは、<security-config> 要素の構文についてのみ説明します。

<security-config> 要素には、次のいずれかを含めます。

- <principal-mapping-entries> 要素。ユーザー名とパスワードを明示的に指定します。
- <principal-mapping-interface> 要素。マッピング・クラスの名前を指定します。
- <jaas-module> 要素。認証に使用する JAAS モジュールを指定します。

oc4j-connectors.xml ファイルの <security-permission> 要素

oc4j-connectors.xml ディスクリプタには、アプリケーションに埋め込まれているリソース・アダプタのみでなく、この OC4J インスタンスにデプロイされているスタンドアロン・リソース・アダプタもリストします。このディスクリプタには、コネクタごとに、コネクタの名前とパス名を指定する <connector> 要素を含めます。各 <connector> 要素には、各リソース・アダプタに付与されたパーミッションを定義する <security-permission> 要素を含めます。次に構文を示します。

```
<security-permission enabled="booleanvalue">
```

この要素には、各リソース・アダプタに付与するパーミッションを指定します。各 <security-permission> 要素には、Java 2 セキュリティ・ポリシー・ファイル構文に準拠した <security-permission-spec> 設定を含めます。

OC4J では、ra.xml ファイルの各 <security-permission> 要素に対して、oc4j-connectors.xml ファイルに <security-permission> 要素を自動的に生成します。生成された各要素では、enabled 属性が false に設定されています。enabled 属性を true に設定すると、指定したパーミッションが付与されます。

```
<oc4j-connectors>
  <connector name="myEIS" path="eis.rar">
  . . .
    <security-permission>
      <security-permission-spec enabled="false">
        grant {permission java.lang.RuntimePermission "LoadLibrary", *'};
      </security-permission-spec>
    </security-permission>
  </connector>
</oc4j-connectors>
```

コンポーネント管理サインオンの概要

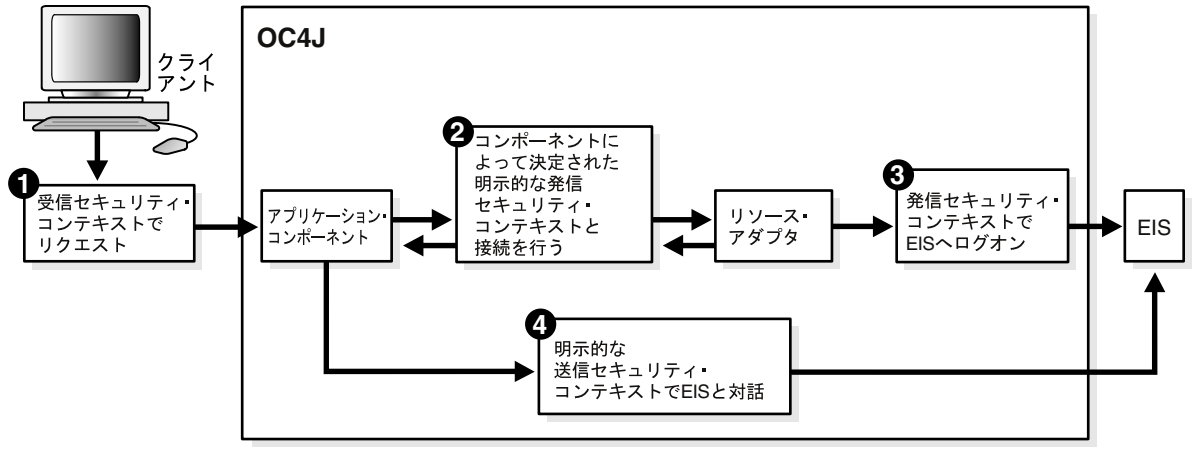
EIS への自身のサインオンを管理するアプリケーションをデプロイする場合は、該当するディスクリプタ・ファイル（Web コンポーネントの場合は web.xml、EJB コンポーネントの場合は ejb-jar.xml）の <res-auth> を Application に設定します。これにより、このアプリケーション・コンポーネントは、サインオンに対して明示的なセキュリティ情報を提供する必要が出てきます。次に例を示します。

```
<resource-ref>
   <res-ref-name>...</res-ref-name>
   <res-type>...</res-type>
   <res-auth>Application</res-auth>
   <res-sharing-scope>...</res-sharing-scope>
</resource-ref>
```

コンポーネント管理サインオンには、OC4J 固有の構成は不要です。

図 20-2 に、コンポーネント管理サインオンの手順を示します。その後に詳細を示すテキストが続きます。

図 20-2 コンポーネント管理サインオン



1. クライアントがリクエストを行い、このリクエストが、開始プリンシパルの受信サブジェクト（セキュリティ・コンテキスト）に関連付けられます。
2. アプリケーション・コンポーネントでは、リクエスト処理の一環として、リソース・プリンシパルの送信サブジェクトに受信サブジェクトをマップするか送信サブジェクトをハードコーディングし、その送信サブジェクトを使用して EIS への接続をリクエストします。
3. 接続取得の一環として、リソース・アダプタは、アプリケーション・コンポーネントから提供された送信サブジェクトを使用して、EIS にサインオンします。
4. 接続が取得されると、アプリケーション・コンポーネントは、確立された送信サブジェクトのもとで EIS と対話できます。

次の例は、コンポーネント管理サインオンを実行するアプリケーションの抜粋です。

```
Context initctx = new InitialContext();
// Perform JNDI lookup to obtain a connection factory.
javax.resource.cci.ConnectionFactory cxf =
                  (javax.resource.cci.ConnectionFactory)initctx.lookup
                                                      ("java:com/env/eis/MyEIS");
// Assume a custom class ConnectionSpecImpl, used to store sign-on credentials.
com.myeis.ConnectionSpecImpl connSpec = ...
connSpec.setUserName("EISuser");
connSpec.setPassword("EISpassword");
// Pass sign-on credentials through getConnection() method call.
javax.resource.cci.Connection cx = cxf.getConnection(connSpec);
```

コンテナ管理サインオンの概要

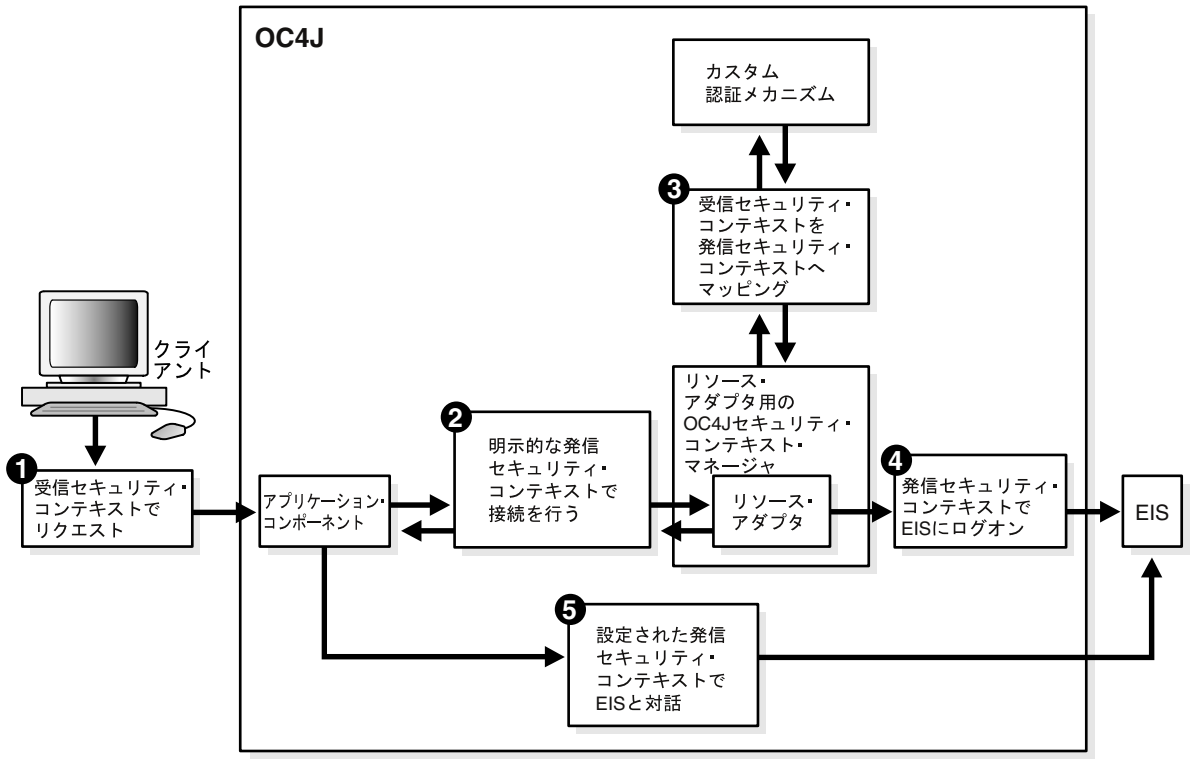
OC4J を使用して EIS へのサインオンを管理するアプリケーションをデプロイする場合は、該当するディスクリプタ・ファイル（Web コンポーネントの場合は `web.xml`、EJB コンポーネントの場合は `ejb-jar.xml`）の `<res-auth>` を `Container` に設定します。これにより、OC4J は、サインオンに対してセキュリティ情報を提供する必要が出てきます。次に例を示します。

```
<resource-ref>
   <res-ref-name>...</res-ref-name>
   <res-type>...</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>...</res-sharing-scope>
</resource-ref>
```

宣言的コンテナ管理サインオンの場合、OC4J では、Application Server Control を介して指定された構成情報を使用します（20-9 ページの「宣言的コンテナ管理サインオンの使用」を参照）。プログラムによるコンテナ管理サインオン（プリンシパル・マッピング・クラスまたは JAAS ログイン・モジュールのいずれを使用するかを問わない）の場合、OC4J では、`oc4j-ra.xml` ファイルに直接指定された構成情報を使用します。アプリケーションで接続を取得しようとすると、OC4J では、該当するメカニズムを使用して送信サブジェクトを確認し、認証を実行します。

図 20-3 に、コンテナ管理サインオンの手順を示します。この手順の詳細はこの図の後に記載されています。

図 20-3 コンテナ管理サインオン



1. クライアントがリクエストを行い、このリクエストが、開始プリンシパルの受信サブジェクト（セキュリティ・コンテキスト）に関連付けられます。
2. アプリケーション・コンポーネントでは、リクエスト処理の一環として EIS への接続をリクエストします。
3. 接続取得の一環として、コンテナ（図に示す OC4J セキュリティ・コンテキスト・マネージャ）は、リソース・プリンシパルの送信サブジェクトに受信サブジェクトをマップします。これは、プリンシパル・マッピング・エントリ要素、プリンシパル・マッピング・クラスまたは JAAS ログイン・モジュールに基づいて行われます。
4. リソース・アダプタでは、OC4J から提供された送信サブジェクトを使用して EIS にログインします。
5. 接続が取得されると、アプリケーション・コンポーネントは、確立された送信サブジェクトのもとで EIS と対話できます。

次の例は、コンテナ管理サインオンを使用するアプリケーションの抜粋です。

```
Context initctx = new InitialContext();

// perform JNDI lookup to obtain a connection factory
javax.resource.cci.ConnectionFactory cxf =
 (javax.resource.cci.ConnectionFactory)initctx.lookup("java:com/env/eis/MyEIS");
// For container-managed sign-on, no security information is passed in the
// getConnection call
javax.resource.cci.Connection cx = cxf.getConnection();
```

コンテナ管理サインオンでの認証

コンテナ管理サインオンを使用する場合、OC4J ではリソース・プリンシパルとその資格証明を EIS に提供する必要があります。プリンシパルと資格証明は、次のいずれかの方法で取得できます。

- 構成済アイデンティティ : リソース・プリンシパルは開始プリンシパルまたはコール元プリンシパルとは関係なく、デプロイ時にデプロイメント・ディスクリプタで構成できます。
- プリンシパル・マッピング : リソース・プリンシパルは、開始プリンシパルまたはコール元プリンシパルのアイデンティティとセキュリティ属性のマッピングによって決定されます。
- コール元擬装 : リソース・プリンシパルは、コール元アイデンティティと資格証明を EIS に委任することで、開始プリンシパルまたはコール元プリンシパルの代理として機能します。
- 資格証明マッピング : リソース・プリンシパルは開始プリンシパルまたはコール元プリンシパルと同じですが、OC4J が使用する認証タイプの資格証明から、EIS が使用する認証タイプの資格証明にマッピングが行われます。一例として、プリンシパルに関連付けられた公開鍵証明書ベースの資格証明から Kerberos 資格証明へのマッピングがあげられます。

OC4J では、これらの方法をすべてサポートするため、JAAS Pluggable Authentication、ユーザー作成認証クラスまたは `oc4j-ra.xml` ファイルの該当する設定が使用されます。

宣言的コンテナ管理サインオンの使用

この項では、プリンシパル・マッピング・エントリの OC4J 固有構成を介して認証を設定する方法について説明します。これを（「プログラムによるコンテナ管理サインオン」に対して）「宣言的コンテナ管理サインオン」と呼びます。Application Server Control を介してこの構成ができます。

プリンシパル・マッピング・エントリのセットとデフォルトのリソース・ユーザーを指定します。各プリンシパル・マッピング・エントリには、開始プリンシパルと対応するリソース・プリンシパルを指定します。プログラム実行時における実際の開始プリンシパル（OC4J ユーザー）が、指定された開始プリンシパルのいずれかと一致する場合は、対応するリソース・プリンシパルが EIS へのサインオンに使用されます。実際の開始プリンシパルが指定されたいずれの開始プリンシパルとも一致しない場合は、EIS へのサインオンにはデフォルトのリソース・ユーザーが使用されます（提供または定義されたものがある場合）。デフォルトのリソース・ユーザーが指定されていない場合は、EIS には `null` サブジェクトが渡されます。この場合、EIS には、EIS 独自のデフォルトでサインオンするオプションがあります。

Application Server Control コンソールで次の手順に従います。

注意： スタンドアロン・リソース・アダプタ用の「リソース・アダプタ」ページにナビゲートする手順は次のとおりです。

1. OC4J ホームページで**「アプリケーション」**タブを選択します。
2. 「スタンドアロン・リソース・アダプタ」を表示します。
3. 目的のリソース・アダプタを選択します。

アプリケーションとともにデプロイされるリソース・アダプタ用の「リソース・アダプタ」ページにナビゲートする手順は次のとおりです。

1. OC4J ホームページで**「アプリケーション」**タブを選択します。
2. 「アプリケーション」を表示します。
3. 目的のアプリケーションを選択します。
4. 表示されるアプリケーションのホームページの「モジュール」で、目的のリソース・アダプタを選択します。

---

1. 目的の「リソース・アダプタ」ページの**「コネクション・ファクトリ」**タブから、編集するコネクション・ファクトリを選択します。コネクション・ファクトリが JNDI ロケーションごとに一覧表示されます。
2. 「コネクション・ファクトリの編集」ページで、**「セキュリティ」**タブを選択します。
3. コンテナ管理サインオンのセキュリティを有効にします。
4. 宣言的プリンシパル・マッピングを指定します。これはデフォルトのリソース・ユーザーを指定するためです。
   a. デフォルトのリソース・ユーザー名を指定します。
   b. デフォルト・リソース・ユーザーのパスワードを指定します。これは、間接パスワードで行うか、またはクリアテキストで希望のパスワードを入力して行います。間接パスワードの場合は、キー（たとえば、単なるユーザー名でもかまいません）を指定します。OC4J では、このキーを使用してセキュリティ・プロバイダで（たとえば、`system-jazn-data.xml` ファイルを介して）検索を実行します。

関連項目：

- 6-2 ページの「パスワードの間接化の使用方法」

5. 開始ユーザー・マッピングを指定します。リソース・プリンシパルにマップする開始プリンシパルごとにマッピングを指定します。既存の行を編集することも、既存のマッピングを変更することも、別の行を追加して新規マッピングを指定することもできます。マッピングごとに、次の手順を実行します。

   a. 開始ユーザー、つまり開始プリンシパルのユーザー名を指定します。

   b. リソース・ユーザー、つまり対応するリソース・プリンシパルのユーザー名を指定します。

   c. リソース・パスワード、つまりマップされたリソース・プリンシパルのパスワードを指定します。デフォルトのプリンシパル・マッピングの場合と同様、これは、間接パスワードで行うかまたは直接パスワードを入力して行います。

6. **「適用」**を選択して変更を適用します。

表 20-1 に、oc4j-ra.xml ファイルでのこれらの設定と XML エンティティの対応関係を示します。表の後に例を示します。

表 20-1 宣言的コンテナ管理サインオンのプロパティ

Application Server Control プロパティ	対応する XML エンティティ	説明
コンテナ管理のサインオンのセキュリティ有効化	<security-config> 要素の use 属性	有効化は use="principal-mapping-entries" に対応します（宣言的コンテナ管理サインオンを想定）。無効化は use="none" に対応します。
デフォルト・リソース・ユーザー	<default-mapping> の <res-user> サブ要素	デフォルト・リソース・プリンシパルのユーザー名。
「間接パスワード」または「パスワード」(「宣言的プリンシパル・マッピング」の場合)	<default-mapping> の <res-password> サブ要素	デフォルト・リソース・プリンシパルのパスワード。間接的または直接的に指定します。
開始ユーザー	<principal-mapping-entry> の <initiating-user> サブ要素	リソース・プリンシパルにマップする開始プリンシパルのユーザー名。これは単純なユーザー名でも、レルム名とスラッシュの付いたユーザー名でもかまいません。
リソース・ユーザー	<principal-mapping-entry> の <res-user> サブ要素	開始プリンシパルにマップするリソース・プリンシパルのユーザー名。開始ユーザーとリソース・ユーザーのペアごとに、独立した <principal-mapping-entry> 要素を使用します。
リソース・パスワード	<principal-mapping-entry> の <res-password> サブ要素	リソース・プリンシパルのパスワード。間接的または直接的に指定します。

```
<oc4j-connector-factories ... >
   <connector-factory ... >
      ...
      <security-config use="principal-mapping-entries">
         <principal-mapping-entries>
            <default-mapping>
               <res-user>scott</res-user>
               <res-password>->tiger</res-password>
            </default-mapping>
            <principal-mapping-entry>
               <initiating-user>servletuser1</initiating-user>
               <res-user>jmsuser1</res-user>
               <res-password>->jmsuser1</res-password>
            </principal-mapping-entry>
            <principal-mapping-entry>
```

```
                <initiating-user>servletuser2</initiating-user>
                <res-user>jmsuser2</res-user>
                <res-password>->jmsuser2</res-password>
              </principal-mapping-entry>
            </principal-mapping-entries>
          </security-config>
        </connector-factory>
        ...
</oc4j-connector-factories>
```

注意： このリリースでは、開始ユーザーの名前は、`<initiating-user>` 要素に、単純な名前（`scott`）またはスラッシュで区切られたレルム名とユーザー名のペア（`myRealm/scott`）として指定できます。ユーザー名は、有効な OracleAS JAAS Provider ユーザーにしてください。

いずれの場合も、OracleAS JAAS Provider デフォルト・レルムを指定する必要があります。6-4 ページの「ファイルベースのプロバイダまたは Oracle Identity Management のデフォルト・レルム」を参照してください。単純なユーザー名を指定する場合、その名前はデフォルト・レルムのメンバーである必要があります。

プログラム的コンテナ管理サインオンの使用

OC4J では、プリンシパル・マッピング・クラスを使用する OC4J 固有のメカニズムを介して、または JAAS ログイン・モジュールを使用するプラッガブル JAAS メカニズムを介して、プログラムによる認証を管理できます。以降の各項では、このメカニズムに加えて追加機能についても説明します。

- プリンシパル・マッピング・クラスの使用
- EIS 接続での JAAS ログイン・モジュールの使用

プリンシパル・マッピング・クラスの使用

プログラムによるコンテナ管理サインオンに対する OC4J でのオプションの 1 つは、プリンシパル・マッピングを実装する Oracle 機能を使用することです。アプリケーションには、`oracle.j2ee.connector.PrincipalMapping` インタフェースを実装したクラスであるプリンシパル・マッピング・クラスを組み込む必要があります。開発者はこれを行うために、直接このインタフェースを実装しても、または便宜上 Oracle によって提供されている `oracle.j2ee.connector.AbstractPrincipalMapping` クラスを拡張してもかまいません。`oc4j-ra.xml` ファイルを介してプリンシパル・マッピング・クラスを構成します。ここでは、プリンシパル・マッピング・クラスの様々な使用方法について説明します。

- プリンシパル・マッピング・インタフェース API の概要
- AbstractPrincipalMapping クラスの拡張
- プリンシパル・マッピング・クラスの構成

プリンシパル・マッピング・インタフェース API の概要

表 20-2 に、PrincipalMapping インタフェースのメソッドの OC4J における使用方法について説明します。

表 20-2 プリンシパル・マッピング・インタフェースのメソッドの説明

メソッド・シグネチャ	OC4J における使用方法
void init (java.util.Properties prop)	OC4J では init() をコールして PrincipalMapping インスタンスの設定を初期化し、oc4j-ra.xml ファイルの <principal-mapping-interface> 要素に指定されたプロパティ値に渡します。(20-15 ページの「プリンシパル・マッピング・クラスの構成」を参照してください。) 実装クラスでは、これらのプロパティを使用して、デフォルトのユーザー名およびパスワード、LDAP 接続の情報またはデフォルト・マッピングを設定できます。
void setManagedConnectionFactory (ManagedConnectionFactory mcf)	OC4J では setManagedConnectionFactory() をコールして、PrincipalMapping インスタンスに ManagedConnectionFactory インスタンス (EIS への接続の場合) を供給し、それが PasswordCredential インスタンスの作成時に使用されます。
void setAuthenticationMechanisms (java.util.Map authMechanisms)	OC4J では setAuthenticationMechanisms() をコールして、リソース・アダプタによってサポートされている認証メカニズムを PrincipalMapping インスタンスに渡します。渡されるマップ内のキーは、サポートされているメカニズム・タイプ ("BasicPassword"、"Kerbv5" など) を含んだ文字列です。このキーに対応する値は、ra.xml ファイルの <credential-interface> 要素に宣言されるような、対応する資格証明インタフェース (たとえば、PasswordCredential インタフェース) の完全修飾名を含んだ文字列です。リソース・アダプタで複数の認証メカニズムがサポートされる場合は、マップに複数のエントリを含めることができます。
Subject mapping (Subject initiatingSubject)	OC4J では mapping() をコールして、PrincipalMapping インスタンスにプリンシパル・マッピングを実行するよう指示します。OC4J ユーザー (開始プリンシパル) 用の Subject インスタンスが渡され、このメソッドにより、リソース・プリンシパル用の Subject インスタンスが、リソース・アダプタによる EIS へのサインオン用に戻されます。(適切なリソース・プリンシパルが見つからなかった場合は、この実装により null が戻されます。)

AbstractPrincipalMapping クラスの拡張

OC4J では、PrincipalMapping インタフェースを実装する抽象クラス AbstractPrincipalMapping が、便宜上用意されています。このクラスには、setManagedConnectionFactory() および setAuthenticationMechanism() メソッドのデフォルト実装の他、次を行えるユーティリティ・メソッドが含まれています。

- EIS への接続に使用される管理コネクション・ファクトリの取得
- リソース・アダプタによってサポートされる認証メカニズムの取得
- リソース・アダプタがパスワード付きの Basic 認証メカニズムをサポートするかどうかの判別
- リソース・アダプタが Kerberos バージョン 5 認証メカニズムをサポートするかどうかの判別
- Subject インスタンスからの Principal インスタンスの抽出

AbstractPrincipalMapping クラスを拡張する場合、開発者は init() および mapping() メソッドを実装するのみで済みます。

AbstractPrincipalMapping クラスによって公開されるメソッドを次の表 20-3 にまとめます。

表 20-3 AbstractPrincipalMapping クラスのメソッドの説明

メソッド・シグネチャ	説明
abstract void init (java.util.Properties prop)	サブクラスでは、init() メソッドを実装する必要があります。詳細は表 20-2「プリンシパル・マッピング・インタフェースのメソッドの説明」を参照してください。
void setManagedConnectionFactory (ManagedConnectionFactory mcf)	サブクラスでは、setManagedConnectionFactory() メソッドを実装する必要がありません。詳細は表 20-2 を参照してください。
void setAuthenticationMechanisms (java.util.Map authMechanisms)	サブクラスでは、setAuthenticationMechanisms() メソッドを実装する必要がありません。詳細は表 20-2 を参照してください。なお、サブクラスでは、isBasicPasswordSupported() および isKerbv5Supported() メソッド（この表で後述）を使用して、どの認証メカニズムがリソース・アダプタによってサポートされているかを判別できます。また、このサブクラスでは getAuthenticationMechanisms() を使用して認証メカニズムを取得することもできます。
abstract Subject mapping (Subject initiatingSubject)	サブクラスでは、mapping() メソッドを実装する必要があります。詳細は表 20-2 を参照してください。
ManagedConnectionFactory getManagedConnectionFactory ()	getManagedConnectionFactory() ユーティリティ・メソッドでは ManagedConnectionFactory インスタンス（EIS への接続用）を戻しますが、PasswordCredential インスタンスの作成にこれが必要な場合があります。
java.util.Map getAuthenticationMechanisms ()	getAuthenticationMechanisms() ユーティリティ・メソッドでは、リソース・アダプタによってサポートされているすべての認証メカニズムのマップを戻します。マップの詳細は、表 20-2 の setManagedConnectionFactory() を参照してください。
boolean isBasicPasswordSupported ()	isBasicPasswordSupported() ユーティリティ・メソッドでは、リソース・アダプタによってパスワード付きの Basic 認証メカニズムがサポートされているかどうかを判別します。
boolean isKerbv5Supported ()	isKerbv5Supported() ユーティリティ・メソッドでは、リソース・アダプタによって Kerbv5 認証メカニズムがサポートされているかどうかを判別します。
Principal getPrincipal (Subject)	getPrincipal() ユーティリティ・メソッドでは、OC4J から渡された OC4J のユーザー Subject インスタンスから Principal インスタンスを抽出します。 **注意 :** サブジェクト内に複数のプリンシパルがある場合、このメソッドでは最初のプリンシパルを取得します。（getPrincipals() メソッドもあります。最初のプリンシパルが、このメソッドによって戻されるプリンシパル・コレクションの最初の要素です。）

例 20-1 では、AbstractPrincipalMapping クラスを拡張して、OC4J ユーザーから EIS のデフォルトのユーザーおよびパスワードにプリンシパル・マッピングを行います。これは、デフォルトのユーザーおよびパスワードが oc4j-ra.xml ファイルの <principal-mapping-interface> 要素に指定されているものと想定しています (20-15 ページの「プリンシパル・マッピング・クラスの構成」を参照)。

例 20-1 AbstractPrincipalMapping の拡張

```
package com.example.app;

import java.util.*;
import javax.resource.spi.*;
import javax.resource.spi.security.*;
import oracle.j2ee.connector.AbstractPrincipalMapping;
import javax.security.auth.*;
import java.security.*;

public class MyMapping extends AbstractPrincipalMapping
{
  String m_defaultUser;
  String m_defaultPassword;

  public void init(Properties prop)
  {
    if (prop != null)
    {
      // Retrieves the default user and password from the properties
      m_defaultUser = prop.getProperty("user");
      m_defaultPassword = prop.getProperty("password");
    }
  }
  public Subject mapping(Subject initiatingSubject)
  {
    // This implementation is for BasicPassword authentication
    // mechanism. Return if the resource adapter does not support it.
    if (!isBasicPasswordSupported())
      return null;
    // Use the utility method to retrieve the Principal from the incoming Subject
    // (security context), corresponding to the OC4J user.
    // This code is included here only as an example.
    // The principal obtained is not actually used in this example.
    Principal principal = getPrincipal(initiatingSubject);
    char[] resPasswordArray = null;
    if (m_defaultPassword != null)
      resPasswordArray = m_defaultPassword.toCharArray();
    // Create a PasswordCredential using the default user name and
    // password, and add it to the Subject, as in "Option A" in the
    // J2EE Connector Architecture specification.
    PasswordCredential cred =
      new PasswordCredential(m_defaultUser, resPasswordArray);
    cred.setManagedConnectionFactory(getManagedConnectionFactory());
    initiatingSubject.getPrivateCredentials().add(cred);
    return initiatingSubject;
  }
}
```

プリンシパル・マッピング・クラスの構成

プリンシパル・マッピング・クラスを使用するには、oc4j-ra.xml を更新してクラス用に <principal-mapping-interface> 要素を追加する必要があります。これは <security-config> 要素のサブ要素で、次のものを含める必要があります。

- <impl-class> サブ要素。これには、プリンシパル・マッピング・クラスの完全修飾名を指定します。
- プリンシパル・マッピング・クラスの実装に必要なプロパティ設定。前項に示したクラスに対して、EIS サインオンのデフォルト・ユーザー名を指定するための name="user" および value 設定が指定された <property> サブ要素と、デフォルト・ユーザーのパスワードを指定するための name="password" および value 設定が指定された <property> サブ要素があります（次の例を参照）。

```
<oc4j-connector-factories>
   <connector-factory name="..." location="...">
      ...
      <security-config use="principal-mapping-interface">
         <principal-mapping-interface>
            <impl-class>com.example.app.MyMapping</impl-class>
            <property name="user" value="scott" />
            <property name="password" value="tiger" />
         </principal-mapping-interface>
      </security-config>
     ...
   </connector-factory>
</oc4j-connector-factories>
```

注意： パスワードの間接化を使用してパスワードを非表示にできます。6-2 ページの「パスワードの間接化の使用方法」を参照してください。

EIS 接続での JAAS ログイン・モジュールの使用

JAAS を介して EIS へのサインオンをプログラムにより管理することもできます。

関連項目：

- 第 9 章「ログイン・モジュール」

OC4J には、Connector Architecture 仕様に準拠する 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・サーバーとその基礎となる認証サービスは相互に独立した状態を維持し、アプリケーション・サーバーに変更を加えずに、新規認証サービスをプラグインできます。

認証サービスでは、次のいずれかのタイプの JAAS ログイン・モジュールを使用してリソース・プリンシパルと資格証明を取得できます。

- プリンシパル・マッピング・ログイン・モジュール
- 資格証明マッピング・ログイン・モジュール
- Kerberos ログイン・モジュール（コール元擬装用）

ログイン・モジュールは、カスタマ、EIS ベンダーまたはリソース・アダプタ・ベンダーが提供できます。ログイン・モジュールは、javax.security.auth.spi.LoginModule インタフェースを実装します。

OC4J では、公開証明書を格納する javax.security.auth.Subject クラスのインスタンス、および OC4J ユーザーを表す oracle.j2ee.connector.InitiatingPrincipal インスタンスを渡すことで、ログイン・モジュールに開始ユーザー・サブジェクトを提供します。認証されたユーザーがない場合、つまり、匿名ユーザーがある場合、OC4J では null サブジェクトを渡すことができます。ログイン・モジュールのログイン・メソッドは、開始ユーザーに基づいて、対応するリソース・プリンシパルを検索して、そのリソース・プリンシパル用に新規の PasswordCredential または GenericCredential インスタンスを作成する必要があ

ります。これにより、リソース・プリンシパルおよび資格証明オブジェクトが、commit() メソッドの開始 Subject インスタンスに追加されます。リソース資格証明は、リソース・アダプタによって提供される javax.resource.spi.ManagedConnectionFactory 実装の createManagedConnection() メソッドに渡されます。null の Subject インスタンスが渡された場合、ログイン・モジュールは、リソース・プリンシパルとそれに対応する資格証明を含んだ新規の Subject インスタンスを作成する必要があります。

InitiatingPrincipal および InitiatingGroup クラス

oracle.j2ee.connector.InitiatingPrincipal クラスは、ログイン・モジュールの OC4J ユーザーを表します。OC4J では、InitiatingPrincipal のインスタンスを作成してサブジェクトに組み込みます。このサブジェクトはログイン・モジュールの initialize() メソッドに渡されます。InitiatingPrincipal クラスは、java.security.Principal インタフェースの実装に getGroups() メソッドが追加されたものです。

また、oracle.j2ee.connector.InitiatingGroup クラスは Principal インタフェースの実装ですが、OC4J ロールを表します。OC4J では、InitiatingPrincipal インスタンスを作成してサブジェクトに組み入れます。このサブジェクトは、ログイン・モジュールの initialize() メソッド、またはプリンシパル・マッピング・クラスの mapping() メソッドに渡されます。また、InitiatingPrincipal クラスには、getGroups() メソッドも含まれています。

getGroups() メソッドでは、InitiatingGroup オブジェクトのセット（java.util.Set インスタンス）を戻します。これは OC4J のロールまたはこの OC4J ユーザーの OracleAS JAAS Provider ロールを表します。ロール・メンバーシップは、OC4J 固有のディスクリプタ・ファイル（通常、system-jazn-data.xml）に定義されます。

ログイン・モジュールでは、getGroups() を使用して OC4J ロールと EIS ユーザー間でマッピングを行えます。Principal インタフェースのメソッドでは、OC4J ロールと EIS ユーザー間のマッピングがサポートされます。ログイン・モジュールは、OC4J ロールと EIS ユーザー間でマッピングを行わない場合は、InitiatingPrincipal および InitiatingGroup クラスを参照する必要がありません。

JAAS と <connector-factory> 要素

oc4j-ra.xml の各 <connector-factory> 要素には、異なる JAAS ログイン・モジュールを指定できます。<jaas-module> 要素に、コネクタ・ファクトリ構成の名前を指定します。次に、oc4j-ra.xml ファイルの <connector-factory> 要素の例を示します。この例では、コンテナ管理サインオンにログイン・モジュールが使用されています。

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

JAAS では、特定のアプリケーションに使用するログイン・モジュールと、ログイン・モジュール間の起動順序を指定する必要があります。JAAS では、<jaas-application-name> 要素に指定された値を使用してログイン・モジュールを検索します。

21

Windows のネイティブ認証の構成

この章では、Web アプリケーション・クライアントが透過的に認証されるように、OC4J で Windows のネイティブ認証（WNA）を設定する方法を説明します。この章の手順を行うには、Kerberos および Active Directory に精通している必要があります。

内容は次のとおりです。

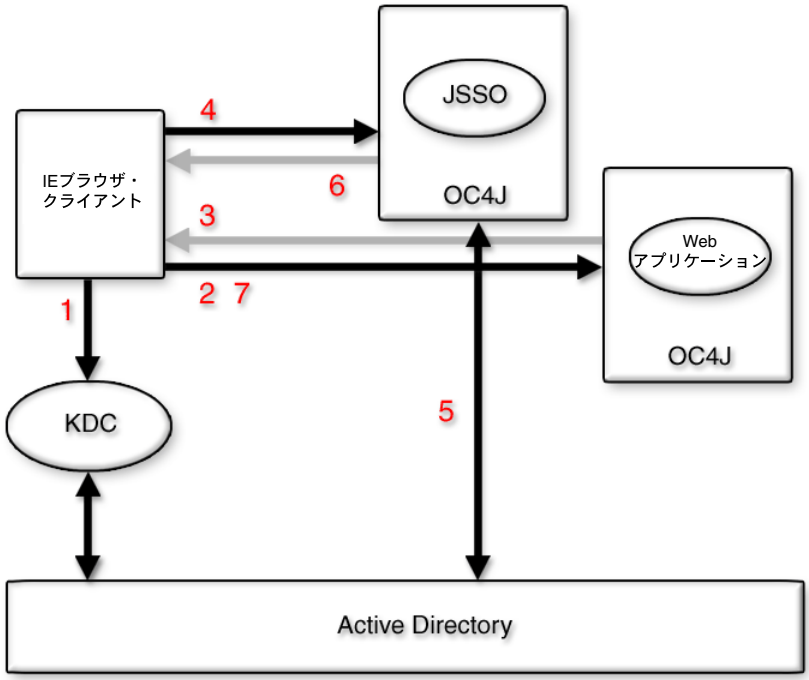
WNA の概要

WNA を使用すると、Microsoft Internet Explorer（または Microsoft の SPNEGO プロトコルをサポートするブラウザ）を使用するデスクトップ・クライアントによって、独自の Windows デスクトップの資格証明を使用して Web アプリケーションが自動的に認証されるようになります。ユーザーのログイン資格証明を調べるかわりに、WNA を使用するように OC4J Web アプリケーションを構成できます。

WNA では、Windows デスクトップへのログイン時に生成された Kerberos セッション・チケットが使用されます。チケットは、ユーザーに対しては表示されず、ログイン資格証明およびその他が含まれています。チケットは、ブラウザを介して Web アプリケーションに渡されます。SPNEGO プロトコルを使用して、ブラウザから Kerberos 資格証明が取得されます。Web アプリケーションは、Windows ドメイン・サーバー上の Key Distribution Center（KDC）サーバーに対してこれらの資格証明をチェックし、検証します。正常に認証された場合は、Web アプリケーションへアクセス権が自動的に付与されます。

次の図 21-1 は、WNA 使用時の関連する主要コンポーネントの概念図です。認証フローの説明は図の下にあります。

図 21-1 WNA の概念図およびフロー



認証フローは次のとおりです。

- （1）ユーザーが Windows デスクトップでドメインにサインインします（Windows サインオン）。ドメインは Kerberos 認証用に構成されています。
- （2、3、4）次に、ブラウザを使用して Web アプリケーションにアクセスします。Web アプリケーションは、Java シングル・サインオン（JSSO）パートナ・アプリケーションとして構成されています。認証されなかった場合、ユーザーは JSSO にリダイレクトされます。JSSO アプリケーションは、WNA 認証を使用して構成されており、Generic Security Services Java API（GSS-API）トークン交換プロトコルを使用してユーザーの Kerberos チケットをリクエストします。OC4J は、チケットを検証してユーザーのセキュリティ・アカウント・マネージャ（SAM）アカウント情報を抽出します。

- （5、6、7）OC4J は、Active Directory に問い合せてユーザーを探し、SAM アカウント名を Active Directory 内のユーザーにマップします。次に、Active Directory を使用してユーザーのエンタープライズ・グループを算出します。ユーザーのアイデンティティおよびエンタープライズ・グループが認証済サブジェクトとして設定され、認証ステップが完了します。その後、ユーザー用に Cookie が設定され、リクエストが元の Web アプリケーションにリダイレクトされます。

WNA を構成するための前提条件

WNA を構成する前に、次のタスクを完了しておく必要があります。

- OC4J サーバーをインストールします。
- 使用する Active Directory サーバーがアクセス可能であり、サーバーの KDC が作動することを検証します。
- Active Directory 内に必要なユーザーおよびグループを作成します。たとえば、`oc4jadmin`、および `oc4j-adminstrators` や `oc4j-app-adminstrators` などの管理グループです。すべての必須アカウントのリストは、11-17 ページの「LDAP サーバーでの必須アカウントの作成」を参照してください。

ステップ 1: Linux ホスト・システムを Kerberos クライアントとして構成する

JSSO がデプロイされている Linux ホスト・システムは、Kerberos クライアントとして構成する必要があります。

Linux ホスト・システムを Kerberos クライアントとして構成するには、次のようにします。

1. Linux ホスト・サーバー上で `/etc/krb5.conf` ファイルを開きます。
2. `[libdefaults]` セクションの `default_realm` の値を編集し、使用する環境のデフォルト Kerberos レルム名に変更します。この名前にはドメイン名を使用できますが、大文字で入力する必要があります。次に例を示します。

```
[libdefaults]
 default_realm = MYDOMAIN.COM
```

3. `[realms]` セクションで、レルム名を手順 2 で定義した名前に置き換え、値を Active Directory サーバーの完全修飾ドメイン名（Kerberos のポート番号を含む）に設定します。次に例を示します。

```
[realms]
 MYDOMAIN.COM = {
  kdc = name.mydomain.com:8787
 }
```

4. `[domain_realm]` セクションで、使用する JSSO サーバーの DNS ドメイン名をデフォルト・レルムと同一に設定します。次に例を示します。

```
[domain_realm]
.mydomain.com = MYDOMAIN.COM
```

5. `krb5.conf` ファイルを保存して閉じます。
6. JSSO サーバーと Active Directory サーバーのクロックが同期していることを確認します。時刻、日付およびタイムゾーンの設定も確認してください。

ステップ 2: Active Directory でユーザー・アカウントを作成する

JSSO がデプロイされているホスト名を使用して、Active Directory サーバーで Web アプリケーションのホスト・ユーザー・アカウントを作成する必要があります。

Web アプリケーション・サービスのホスト・ユーザー・アカウントを作成するには、次のようにします。

1. 管理者として Active Directory サーバーにログインします。
2. **「スタート」→「すべてのプログラム」→「管理ツール」→「Active Directory ユーザーとコンピュータ」**の順にクリックします。
3. 「Users」フォルダを選択します。
4. **「ユーザーの作成」**をクリックします。
5. 次の情報を入力します。
   - 「名」: このアカウントのニックネームを入力します。判読可能な任意の名前を指定できます。
   - 「ユーザー ログオン名」: JSSO ホストの完全 DNS 名。たとえば `mydomain.com` です。
6. **「次へ」**をクリックします。
7. このユーザーのパスワードを入力します。パスワード期限の設定は選択しないでください。
8. **「次へ」**をクリックします。ユーザーのリストに新しいユーザーが表示されます。

ステップ 3: Linux ホストに対して Keytab ファイルを生成してテストする

keytab ファイルは、アカウント名をサービス・プリンシパル名にマップするために Web アプリケーションで使用されます。keytab ファイルは必須であり、`ktpass` コマンドを使用して生成する必要があります。

Linux ホストの keytab ファイルを生成するには、次のようにします。

1. Active Directory サーバーでコマンド・プロンプトを開きます。
2. 次のコマンドを発行します。

```
C:¥> ktpass -princ HTTP/domain@DEFAULT_REALM -pass password
-mapuser host_user_account -out file_name.keytab
```

詳細は次のとおりです。

- `-princ`（プリンシパル）値は、`HTTP/` の後に Web アプリケーション・サービス・ホストの完全修飾ドメイン名、さらにデフォルト・レルム名を続けたものです。大 / 小文字が区別されます。Active Directory のデフォルト・レルムは大文字、Web アプリケーションの完全修飾ドメイン名は小文字で表記する必要があります。次に例を示します。

  ```
  -princ HTTP/name.mydomain.com@MYDOMAIN.COM
  ```

- `-pass` 値は、Active Directory サーバーで作成された Web アプリケーションのホスト・ユーザー・アカウントに割り当てられているパスワードと同じにする必要があります。
- `-mapuser` 値は、Active Directory サーバーで作成された JSSO の完全修飾ドメイン名（ホスト名）のユーザーです。
- `-out` 値は出力ファイルの名前です。次に例を示します。

  ```
  MyFile.keytab
  ```

3. JSSO がデプロイ済で Linux ホスト上で構成済の OC4J インスタンスに、生成された keytab ファイルをコピーします。次に例を示します。

   ORACLE_HOME/j2ee/home/config

4. 次の kinit コマンドを発行して Linux サーバーと Active Directory サーバー間の Kerberos 接続をテストし、両方が適切に構成されていることを確認します。Red Hat Linux 用の kinit 実行可能ファイルは、/usr/kerberos/bin ディレクトリにあります。

   ```
   # /usr/kerberos/bin/kinit -k -t ORACLE_HOME/j2ee/home/config/keytab_file HTTP/
   domain_name
   ```

   コマンドには、手順 2 で生成された keytab ファイルのフルパスおよび Web アプリケーション・サービス・ホストの完全修飾ドメイン名を含める必要があります。次に例を示します。

   ```
   # /usr/kerberos/bin/kinit -k -t oracle/j2ee/home/config/MyFile.keytab HTTP/
   name.mydomain.com
   ```

   コマンドが正常に実行された場合、コマンドからの出力は行われず、別のコマンド・プロンプトに戻ります。出力でレポートされたエラーは、すべて解決する必要があります。

ステップ 4: OC4J インスタンスを構成する

この項のタスクでは、OC4J インスタンスを構成し、WNA を使用できるようにデプロイ済 Web アプリケーションを構成します。これらのタスクを行う前に、OC4J インスタンスを停止してください。また、すべてのタスクが完了した後に、必ず OC4J を起動してください。

WNA を使用するために OC4J インスタンスを構成するには、次のタスクを行う必要があります。

- OC4J のセキュリティ・システム・プロパティの設定
- システムの JAZN 構成ファイルの編集
- JAZN 構成ファイルの編集
- アプリケーションのデプロイメント・ディスクリプタの編集

OC4J のセキュリティ・システム・プロパティの設定

OC4J インスタンスに対して WNA を有効にするには、次の 2 つのシステム・プロパティを設定する必要があります。これらのプロパティは、通常 oc4j.cmd または oc4j.sh（OC4J の起動に使用されるスクリプト）内で設定されます。oc4j.jar を使用して OC4J を直接起動する際に、コマンドラインで設定することもできます。

```
-Djavax.security.auth.useSubjectCredsOnly=false
-Doracle.security.jazn.config =ORACLE_HOME/j2ee/home/config/jazn.xml
```

デバッグ用のシステム・プロパティ

Kerberos 認証用のデバッグ情報を有効にするために、3 つの追加プロパティを任意で設定できます。各プロパティを次に示します。

```
-Djava.security.krb5.realm=kerberos-realm-name
-Djava.security.krb5.kdc=kdc_host_name
-Dsun.security.krb5.debug=true
```

システムの JAZN 構成ファイルの編集

Kerberos WNA と Active Directory の両方のログイン・モジュールを system-jazn-data.xml ファイルに追加する必要があります。ログイン・モジュールは、OC4J に登録済のログイン・モジュールが含まれている <jazn-loginconfig> 要素内に追加されます。

Kerberos WNA および Active Directory のログイン・モジュールを追加するには、次のようにします。

1. *ORACLE_HOME*/j2ee/home/config/system-jazn-data.xml を開きます。
2. com.sun.security.jgss.accept アプリケーションの Kerberos ログイン・モジュール構成を定義する、次の XML ブロックを追加します。keytab および principal の値は、4 ページの「ステップ 3: Linux ホストに対して Keytab ファイルを生成してテストする」で指定されたものと同じにする必要があります。keyTab 値には keytab ファイルの完全パスを含める必要があり、principal 値は、JSSO サーバーを実行するマシンの完全修飾ドメイン名（接頭辞は HTTP/）にする必要があります。

```
...
<jazn-loginconfig>
   <application>
      <name>com.sun.security.jgss.accept</name>
      <login-modules>
         <login-module>
            <class>com.sun.security.auth.module.Krb5LoginModule</class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>debug</name>
                  <value>true</value>
               </option>
               <option>
                  <name>addAllRoles</name>
                  <value>true</value>
               </option>
               <option>
                 <name>useKeyTab</name>
                 <value>true</value>
               </option>
               <option>
                  <name>keyTab</name>
                  <value>C:/j2ee/home/config/MyFile.keytab</value>
               </option>
               <option>
                  <name>principal</name>
                  <value>HTTP/name.mydomain.com</value>
               </option>
               <option>
                  <name>doNotPrompt</name>
                  <value>true</value>
               </option>
               <option>
                  <name>storeKey</name>
                  <value>true</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
   ...
```

3. JSSO アプリケーションの Kerberos ログイン・モジュール構成を定義する、次の XML ブロックを追加します。次の値は、それぞれの Active Directory 環境にあわせて変更する必要があります。

   - `oracle.security.jaas.ldap.provider.user`: Active Directory サーバー管理者
   - `oracle.security.jaas.ldap.provider.credential`: Active Directory サーバー管理者の資格証明
   - `oracle.security.jaas.ldap.user.searchbase`: Active Directory のユーザー検索ベース
   - `oracle.security.jaas.ldap.role.searchbase`: Active Directory のロール検索ベース
   - `oracle.security.jaas.ldap.provider.url`: Active Directory サーバーの URL

```
...
<jazn-loginconfig>
   <application>
      <name>javasso</name>
      <login-modules>
         <login-module>
            <class>oracle.security.jazn.login.module.LDAPLoginModule</class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>oracle.security.jaas.ldap.connect.pool.prefsize</name>
                  <value>10</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.connect.pool</name>
                  <value>true</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.user</name>
                  <value>cn=administrator,cn=users,dc=dlin,dc=net</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.credential</name>
                  <value>!welcome1</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.type</name>
                  <value>Active Directory</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.connect.pool.maxsize</name>
                  <value>25</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.role.searchscope</name>
                  <value>subtree</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.user.searchscope</name>
                  <value>subtree</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.membership.searchscope</name>
                  <value>direct</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.lm.cache_enabled</name>
                  <value>true</value>
```

```
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.member.attribute</name>
                   <value>member</value>
                </option>
                <option>
                <name>oracle.security.jaas.ldap.connect.pool.initsize</name>
                   <value>2</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.connect.pool.timeout</name>
                   <value>300000</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.user.object.class</name>
                   <value>inetOrgPerson</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.provider.url</name>
                   <value>ldap://name.mydomain.com:389</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.role.searchbase</name>
                   <value>cn=builtin,dc=dlin,dc=net</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.user.searchbase</name>
                   <value>cn=users,dc=dlin,dc=net</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.role.object.class</name>
                   <value>group</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.role.name.attribute</name>
                   <value>cn</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.user.name.attribute</name>
                   <value>sAMAccountName</value>
                </option>
             </options>
          </login-module>
       </login-modules>
    </application>
    ...
```

4. `system-jazn-data.xml` ファイルを保存して閉じます。

JAZN 構成ファイルの編集

JAZN 構成ファイルを編集して、WNA 認証タイプを登録する必要があります。このファイルには OC4J の JSSO 構成が含まれています。

WNA 認証タイプを登録するには、次のようにします。

1. *ORACLE_HOME*/j2ee/home/config/jazn.xml ファイルを開きます。
2. <jazn> 要素内に次のプロパティを追加します。

```
<jazn>
   ...
   <property name="custom.sso.token.asserter.authtypes"
             value="WINDOWS-KERBEROS-AUTH"/>
   ...
<jazn>
```

3. ファイルを保存して閉じます。

アプリケーションのデプロイメント・ディスクリプタの編集

WNA 認証を使用するためには、JSSO アプリケーション用 Oracle アプリケーション・デプロイメント・ディスクリプタおよび JSSO を使用するすべての Web アプリケーションを編集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・ディスクリプタを編集するには、次のようにします。

1. *ORACLE_HOME*/j2ee/home/application-deployments/javasso/orion-application.xml ファイルを開きます。
2. 現在の <jazn> 構成をコメント解除します。
3. 次の <jazn> 構成を追加して、独自の JSSO ホストの完全修飾ドメイン名（接頭辞は HTTP@）を使用するように kerberos-servicename 値を変更します。

```
<jazn provider="XML">
   <jazn-web-app auth-method="WINDOWS_KERBEROS_AUTH"/>
   <property name="kerberos-servicename" value="HTTP@name.mydomain.com"/>
</jazn>
```

4. JSSO アプリケーションの orion-application.xml ファイルを保存して閉じます。
5. JSSO アプリケーションを使用するそれぞれの Web アプリケーションについて、orion-application.xml ファイルを開いて次のように XML プロバイダの認証方式属性を変更し、CUSTOM_AUTH が使用されるようにします。

```
<jazn provider="XML">
   ...
   <jazn-web-app auth-method="CUSTOM_AUTH"/>
   ...
</jazn>
```

各アプリケーションの Oracle アプリケーション・デプロイメント・ディスクリプタは、*ORACLE_HOME*/j2ee/home/application-deployments/*application_name*/orion-application.xml にあります。

ステップ 5: ブラウザを構成して WNA をテストする

WNA を使用するように構成されている Web アプリケーションにアクセスする前に、クライアントの Internet Explorer のブラウザに、JSSO アプリケーションがデプロイされているホスト名を信頼できるホストとして追加する必要があります。JSSO ドメイン内のコンピュータで Internet Explorer が実行されている場合は、Kerberos ネゴシエーションが行われます。JSSO ドメインの一部でないコンピュータで Internet Explorer が実行されている場合は、フォールバック・ロジックにより Basic 認証が使用され、ユーザー名およびパスワードの入力を求められます。ユーザー名およびパスワードは、`system-jazn-data.xml` ファイルで構成されている Active Directory に対して認証されます。

ブラウザの設定を構成して WNA 機能をテストするには、次の手順を実行します。

1. Windows ドメインにログインします。
2. Internet Explorer のブラウザを開きます。
3. ブラウザの「ツール」メニューから「インターネット オプション」を選択します。「インターネット オプション」ダイアログ・ボックスが表示されます。
4. 「インターネット オプション」ダイアログ・ボックスの「セキュリティ」タブを選択します。
5. 「イントラネット」アイコンをクリックして強調表示し、**「サイト」**をクリックします。「イントラネット」ダイアログ・ボックスが表示されます。
6. 「イントラネット」ダイアログ・ボックスの**「詳細設定」**をクリックします。
7. テキスト・フィールドに、JSSO サーバーをホストするコンピュータの URL を入力します。たとえば `http://dude.mydomain.com` です。JSSO のログイン・ページやポート番号は含めないでください。
8. **「追加」**をクリックします。URL がローカル・イントラネット・ゾーンの Web サイトのリストに追加されます。
9. **「OK」**をクリックし、もう一度**「OK」**クリックして「イントラネット」ダイアログ・ボックスを閉じます。
10. 「インターネット オプション」ダイアログ・ボックスの「セキュリティ」タブで**「レベルのカスタマイズ」**をクリックします。「セキュリティの設定」ダイアログ・ボックスが表示されます。「イントラネット ゾーンでのみ自動的にログオンする」オプションが選択されていることを確認します。
11. **「OK」**をクリックして「セキュリティの設定」ダイアログ・ボックスを閉じます。
12. **「OK」**をクリックして「インターネット オプション」ダイアログ・ボックスを閉じます。
13. ブラウザの「アドレス」フィールドに次の URL を入力し、JSSO サーバーのログイン・ページに移動します。`http://`*`host:port`*`/jsso/SSOLogin`。*`host:port`* を、JSSO アプリケーションへのアクセスに使用するホストおよびポートに置き換えます。

    ブラウザを介して Kerberos 資格証明が透過的に JSSO サーバーに渡されるため、JSSO サーバーへのログインが自動的に行われます。最上位ページにリダイレクトされれば、テストは成功です。

A

OC4J セキュリティのヒントおよびトラブルシューティング

この付録では、OC4J セキュリティのベスト・プラクティス、およびトラブルシューティングの対象となる要注意の問題と関連するヒントについて説明します。

注意： このマニュアルでは、他にも各章で、その章のトピックに関連するトラブルシューティング情報が記載されています。

OC4J セキュリティのベスト・プラクティス

この項では、次の項目別ベスト・プラクティスについて説明します。

- JAAS のベスト・プラクティス
- HTTPS のベスト・プラクティス

JAAS のベスト・プラクティス

推奨される JAAS のプラクティスは、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 では同様のセキュリティ・モデルがデフォルトとして使用され、パスワードはクリアテキストでは格納されません。また、OracleAS JAAS Provider は、出荷時の Oracle Application Server インフラストラクチャ（Oracle Single Sign-On と Oracle Internet Directory を含む）との密接な統合を提供します。7-17 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。
- カスタムの UserManager クラスを記述するのは避けます。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` インタフェースは引き続き実装できますが（リリース 10.1.3.x では非推奨）、OracleAS JAAS Provider、Oracle Single Sign-On および Oracle Internet Directory の豊富な機能を利用すれば、インフラストラクチャ・コードではなくビジネス・ロジックに専念できます。Oracle 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。カスタム機能が必要な場合は、カスタムの `UserManager` 実装のかわりにカスタムのログイン・モジュールを使用できます。
- 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 は、ファイルベースのリポジトリをサポートしていますが、ほとんどの本番環境では、リポジトリとして Oracle Internet Directory を使用する Oracle Identity Management 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。（または、Oracle がサポートする外部 LDAP プロバイダのいずれかを使用します。）
- OracleAS JAAS Provider での認証メカニズムとして Oracle Single Sign-On を使用します。様々な認証オプションを使用できますが、次の理由から、可能なかぎり Oracle 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。

  （または、インストールに Oracle Application Server インフラストラクチャが含まれていなければ、Java SSO を使用します。）
- 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（構成を介して）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 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
  - 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。
- 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。

HTTPS のベスト・プラクティス

Oracle HTTP Server 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 : Oracle HTTP Server 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 : Oracle HTTP Server 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 : Oracle HTTP Server 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 を構成してください。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-to-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非常に大きくなるため、状況によってはトレードオフが必要になります。

  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。SSL セッション開始時の 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。
- セキュアなページは、セキュリティが不要なページと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、この方法はかなりのパフォーマンス・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。

- SSL を使用する場合は、Oracle HTTP Server SSLSessionCacheTimeout ディレクティブをチューニングします。Oracle HTTP Server 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。

  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。

  関連項目：

  - 『Oracle HTTP Server 管理者ガイド』

OC4J セキュリティの一般的なヒントおよびトラブルシューティング

次の問題とその処理方法に留意してください。

ファイル jazn.xml が見つからない

有効な `jazn.xml` ファイルが存在しない場合は、OracleAS JAAS Provider は起動できません。`jazn.xml` ファイルが見つからない場合は、次のエラー・メッセージが生成されます。

```
"JAZN has not been properly configured"
```

関連項目：

- 4-9 ページの「jazn.xml ファイル」

認証の問題

アイデンティティ・リポジトリでユーザーとパスワードが正しく構成されているにもかかわらず、保護されたアプリケーションにログインしようとして認証が失敗した場合は、アイデンティティ・リポジトリが起動されて使用可能な状態になっていること、およびそのリポジトリが `orion-application.xml` または `jazn.xml`（該当する場合）の `<jazn>` 要素の `location` 属性で指定された場所にあることを、なんらかの方法で確認してください。

OracleAS JAAS Provider を JAAS プロバイダとして指定する際の失敗

次のような例外スタック・トレースを受信することがあります。

```
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
```

この場合は、OracleAS JAAS Provider が JAAS ポリシー・プロバイダとして指定されていない可能性があります。

関連項目：

- 5-17 ページの「Oracle ポリシー・プロバイダの指定」

レルムの問題

この項では、レルムの使用に関してトラブルシューティングの対象となる問題について説明します。

- ユーザー名からのレルム名の省略
- 認証の失敗を解決するためのデフォルト・レルムの指定

ユーザー名からのレルム名の省略

OC4J プロパティ `jaas.username.simple` では、キー・メソッド（たとえば、サーブレットの場合は `getUserPrincipal()` または `getRemoteUser()`、EJB の場合は `getCallerPrincipal()`）から戻されたプリンシパルのユーザー名においてレルム名が接頭辞として付けられているかどうかが判別されます。デフォルトの `true` 設定では、レルム名は接頭辞として付けられません。

カスタム・レルムを構成して使用する場合は、このプロパティを明示的に `false` に設定して、OracleAS JAAS Provider の認証および認可が適切に機能するようにする必要があります。詳細は、6-7 ページの「認証済プリンシパル取得時のレルム名の省略」を参照してください。

認証の失敗を解決するためのデフォルト・レルムの指定

認証に失敗したとき、構成が正しいと思われる場合は、デフォルト・レルムを指定する必要があるかどうかを確認します。インスタンスレベルの `jazn.xml` ファイルで指定した以外のデフォルト・レルムを使用する場合は、それを、`orion-application.xml` ファイル内の `<jazn>` 要素に構成する必要があります。

これは、ファイルベース・プロバイダまたは LDAP ベース・プロバイダのいずれかに適用できます。

ロギング

この項では、デバッグに役に立つロギング機能について説明します。

- OracleAS JAAS Provider での Oracle Diagnostic Logging の使用
- OracleAS JAAS Provider Admintool での標準の JDK ロギングの使用

関連項目：

- 標準の Java ロギング機能の概要は、次の URL を参照してください。

  `http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html`

- 次の URL にある `java.util.logging` パッケージの Javadoc を参照してください。

  `http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html`

- OC4J でのロギング機能およびロギング構成の詳細は、『Oracle Containers for J2EE 開発者ガイド』および『Oracle Containers for J2EE 構成および管理ガイド』を参照してください。

OracleAS JAAS Provider での Oracle Diagnostic Logging の使用

OC4J および OracleAS JAAS Provider では、Oracle Diagnostic Logging フレームワーク（ODL）がサポートされています。このフレームワークでは、標準的な Java ロギング・フレームワークを補完するプラグイン・コンポーネントが提供され、ログ・データが自動的に Oracle ログ分析ツールと統合されます。

OC4J における通常の手順と同様に、*ORACLE_HOME*/j2ee/home/config/j2ee-logging.xml 内のロギング・レベルを、デフォルトの NOTIFICATION:1 から適切なエラー・レベルまたはデバッグ・レベルに変更します。OracleAS JAAS Provider でよく使用される 2 つのレベルは、FINE と FINER で、これはそれぞれ TRACE:1 と TRACE:16 に相当します。

OracleAS JAAS Provider のロギング・エントリは、*ORACLE_HOME*/j2ee/*instance_name*/logs/oc4j/log.xml にあります。このファイルの中で関連のあるエントリは、次のサンプル・メッセージに示すように、COMPONENT_ID が j2ee であるものと MODULE_ID が security であるものです。

```
<MESSAGE>
 <HEADER>
   <TSTZ_ORIGINATING>2005-12-14T11:41:08.974-08:00</TSTZ_ORIGINATING>
   <COMPONENT_ID>j2ee</COMPONENT_ID>
   <MSG_TYPE TYPE="TRACE"></MSG_TYPE>
   <MSG_LEVEL>16</MSG_LEVEL>
   <HOST_ID>www.example.com</HOST_ID>
   <HOST_NWADDR>555.55.5.555</HOST_NWADDR>
   <MODULE_ID>security</MODULE_ID>
   <THREAD_ID>10</THREAD_ID>
   <USER_ID>nmuralid</USER_ID>
 </HEADER>
 <CORRELATION_DATA>
   <EXEC_CONTEXT_ID>
      <UNIQUE_ID>555.55.5.555:30508:1134589268971:0</UNIQUE_ID><SEQ>0</SEQ>
   </EXEC_CONTEXT_ID>
 </CORRELATION_DATA>
 <PAYLOAD>
   <MSG_TEXT>location=system-jazn-data.xml</MSG_TEXT>
 </PAYLOAD>
</MESSAGE>
```

あるいは、最初から OracleAS JAAS Provider メッセージのみがロギングされるようにするには、次の例に示すように、j2ee-logging.xml に構成を追加して、ログ出力名を oracle.j2ee.security に設定することもできます。

```
<logger name="oracle.j2ee.security" level="NOTIFICATION:32"
        useParentHandlers="false">
   <handler name="oc4j-handler"/>
   <handler name="console-handler"/>
</logger>
```

OracleAS JAAS Provider Admintool での標準の JDK ロギングの使用

OracleAS JAAS Provider Admintool では、標準の JDK ロギングが使用されます。Admintool に対してロギングを実行するには、ロギング・レベルを INFO から FINE、FINER または FINEST に変更します。（Admintool からのほとんどのログ・メッセージは、FINE レベルか FINER レベルでロギングされます。）この変更は、*JAVA_HOME*/jre/lib/logging.properties ファイルを編集する、またはこのファイルの更新したコピーを Admintool のコマンドラインに供給することで行えます。次のコマンドは Admintool を実行し、プロパティ・ファイルを提供して妥当なロギング・レベルを設定しています。メッセージは、構成したログ・ハンドラによってロギングされます。

```
% java -jar jazn.jar -Djava.util.logging.config.file=modified_logging_properties
```

注意： 前のリリースで使用されていた jazn.debug.log.enable フラグは、すでにサポートされていません。

B

OracleAS JAAS Provider のサンプル

この付録では、1 つのサーブレットについていくつかのバージョンのサンプルを示します。最初に標準の J2EE セキュリティ API を使用したもの、次にユーザーにパーミッションを付与してポリシーを管理するコードを追加したもの、最後にユーザーのパーミッションをチェックするコード（JAAS モードおよび JAAS 認可）を追加したものを示します。

- サンプル・サーブレットのセキュリティ構成
- サンプル・サーブレット : J2EE セキュリティ API の起動
- サンプル・サーブレット : パーミッションの付与
- サンプル・サーブレット : パーミッションのチェック

関連項目：

- OC4J の様々な使用方法の例は、次の Web サイトを参照してください。

  ```
  http://www.oracle.com/technology/tech/java/oc4j/1013/
  how_to/index.html
  ```

サンプル・サーブレットのセキュリティ構成

この付録に示す各バージョンのサンプル・サーブレットは、ファイルベース・プロバイダを使用し、次の構成に基づいています。

- system-jazn-data.xml: ロール developers に属しているユーザー developer
- web.xml: サーブレットのロール sr_developers およびセキュリティ制約
- orion-application.xml: developers と sr_developers 間のロール・マッピング

これらの構成を、この後の各項で示します。

system-jazn-data.xml の構成

system-jazn-data.xml ファイルでは、jazn.com レルムに、developer ユーザーおよびこのユーザーが属する developers ロールが定義されます。

ファイルベース・プロバイダに対してユーザーおよびロールを定義する場合、Application Server Control を使用する方法をお薦めします（7-2 ページの「Application Server Control でのファイルベース・プロバイダの構成」を参照）。OracleAS JAAS Provider Admintool も使用できます。

```
<jazn-data>
   ...
   <jazn-realm>
      <realm>
         <name>jazn.com</name>
         <users>
            ...
            <user>
               <name>developer</name>
               <display-name>developer</display-name>
               <credentials>{903}CafGQDjOlPMyMiwJEwUfyjhGLAbQkzhR</credentials>
            </user>
            ...
         </users>

         <roles>
            ...
            <role>
               <name>developers</name>
               <display-name>Developer Role</display-name>
               <members>
                  <member>
                     <type>user</type>
                     <name>developer</name>
                  </member>
               </members>
            </role>
            ...
         </roles>
      </realm>
   </jazn-realm>
   ...
</jazn-data>
```

web.xml の構成

web.xml ファイルでは、セキュリティ制約を設定し、ロール sr_developers を定義します。認証方式に関する設定もあります。（web.xml の認証方式は orion-application.xml の <jazn-web-app> 要素の設定で上書きできます。）

```
<web-app>
   ...
   <security-role>
        <role-name>sr_developers</role-name>
   </security-role>
   ...
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>CallerInfoA</web-resource-name>
         <url-pattern>/callerInfoA</url-pattern>
      </web-resource-collection>
      <!-- authorization -->
      <auth-constraint>
         <role-name>sr_developers</role-name>
      </auth-constraint>
   </security-constraint>
   ...
   <!-- authentication -->
   <login-config>
      <auth-method>BASIC</auth-method>
   </login-config>
   ...
</web-app>
```

orion-application.xml の構成

orion-application.xml ファイルでは、ファイルベース・プロバイダを指定し、セキュリティ・ロール sr_developers を、アイデンティティ・ストア（この例では system-jazn-data.xml）に定義されているロール developers にマップします。

Application Server Control を使用して、セキュリティ・プロバイダおよびセキュリティ・ロール・マッピングを指定します（6-10 ページの「セキュリティ・プロバイダの指定」および 6-11 ページの「セキュリティ・ロールのマッピング」を参照）。

```
<orion-application>
   ...
   <security-role-mapping name="sr_developers">
      <group name="developers" />
   </security-role-mapping>
   ...
   <!-- use JAZN-XML by default -->
   <jazn provider="XML" />
   ...
</orion-application>
```

サンプル・サーブレット : J2EE セキュリティ API の起動

この最初のバージョンのサーブレットでは、ユーザーの取得、ユーザーがロールに属しているかどうかの判別およびユーザー・プリンシパルの取得に、標準の J2EE セキュリティ API が使用されます。

```
import java.io.IOException;
import java.util.Date;
import java.util.Properties;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class CallerInfo extends HttpServlet {

    public CallerInfo() {
        super();
    }

    public void init(ServletConfig config)
            throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        ServletOutputStream out = response.getOutputStream();

        response.setContentType("text/html");
        out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
        out.println("Time stamp: " + new Date().toString());
        out.println
           ("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
        out.println("request.isUserInRole('ar_developers') = " +
                    request.isUserInRole("sr_developers") + "<br>");
        out.println
            ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
```

サンプル・サーブレット : パーミッションの付与

このバージョンのサーブレットでは、ユーザーにパーミッションを付与するコードが追加されています。パーミッションの付与には、OracleAS JAAS Provider Admintool を使用することもできます（C-15 ページの「パーミッションの付与と取消し」を参照）。

```
import java.io.*;
import java.util.Date;
import java.util.Properties;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.security.jazn.*;
import oracle.security.jazn.realm.*;
import oracle.security.jazn.oc4j.*;
import oracle.security.jazn.spi.Grantee;
import oracle.security.jazn.policy.*;
import javax.security.auth.*;
import java.security.*;

public class CallerInfo extends HttpServlet {

    public CallerInfo() {
        super();
    }

    public void init(ServletConfig config)
            throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        ServletOutputStream out = response.getOutputStream();
        response.setContentType("text/html");
        out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
        out.println("Time stamp: " + new Date().toString());
        out.println
            ("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
        out.println("request.isUserInRole('ar_developers') = " +
                    request.isUserInRole("ar_developers") + "<br>");
        out.println
            ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");

   //Grant Permissions to a user developer

   //get JAZNConfiguration related info
   JAZNConfig jc = JAZNConfig.getJAZNConfig();

   //create a Grantee for "developer"
   RealmManager realmmgr = jc.getRealmManager();
   Realm realm = realmMgr.getRealm("jazn.com");
   UserManager userMgr = realm.getUserManager();
   final RealmUser user = userMgr.getUser("developer");

   //grant scott file permission
   JAZNPolicy policy = jc.getPolicy();
   if ( policy != null) {
      Grantee gtee = new Grantee( (Principal) user);
      java.io.FilePermission fileperm = new java.io.FilePermission
                                        ("foo.txt","read");
      policy.grant( gtee, fileperm);
   }
```

```
out.println("</BODY>");
   out.println("</HTML>");
}
```

サンプル・サーブレット : パーミッションのチェック

このバージョンのサーブレットでは、パーミッションをチェックするための JAAS モードおよび JAAS 認可の構成およびコードが追加されています。

JAAS モードでは、J2EE アプリケーションを `Subject.doAs()` ブロックで実行するのか `Subject.doAsPrivileged()` ブロックで実行するのかを制御します。このモードを設定すると、認証サブジェクトが、該当するアクセス制御コンテキストと関連付けられます。設定後、標準の JAAS および J2SE API を使用して認可チェックをアプリケーションに組み込むことができます。

関連項目 :

- 5-6 ページの「JAAS モードの概要」

orion-application.xml の JAAS モード構成

この例では、前に示した `orion-application.xml` 構成が拡張され、JAAS モードが `doasprivileged` に設定されています。この設定を使用して、OC4J は `Subject.doAsPrivileged()` ブロック内でサーブレットを実行します。

```
<orion-application>
   ...
   <security-role-mapping name="sr_developers">
      <group name="developers" />
   </security-role-mapping>
   ...
   <!-- use JAZN-XML by default -->
   <jazn provider="XML" jaas-mode="doasprivileged" />
   ...
</orion-application>
```

認可のサーブレット・コード

次に示すサーブレット・コードでは、JAAS ポリシーを使用して、ユーザーに `foo.txt` の読取り権限があるかどうかをチェックしています。前述の構成のために、`doasprivileged` モードが使用されます。

この例では、比較のため、`AccessController` を使用してパーミッションをチェックする等価のコードも示しています。`doAsPrivileged()` ブロック内のコードは、JAAS ポリシー・コードの `doasprivileged` 構成と等価です。

```
import java.io.*;
import java.util.Date;
import java.util.Properties;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;

import oracle.security.jazn.*;
import oracle.security.jazn.realm.*;
import oracle.security.jazn.oc4j.*;
import oracle.security.jazn.spi.Grantee;
import oracle.security.jazn.policy.*;

import javax.security.auth.*;
import java.security.*;
```

```
public class CallerInfo extends HttpServlet {

    public CallerInfo() {
        super();
    }

    public void init(ServletConfig config)
            throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       final ServletOutputStream out = response.getOutputStream();

       response.setContentType("text/html");
       out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
       out.println("Time stamp: " + new Date().toString());
       out.println
           ("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
       out.println("request.isUserInRole('ar_developers') = " +
                   request.isUserInRole("ar_developers") + "<br>");
       out.println
           ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");

       //create Permission
       FilePermission perm = new FilePermission("/home/developer/foo.txt","read");

       // CHECK PERMISSION VIA JAAS POLICY
       //get current AccessControlContext
       AccessControlContext acc = AccessController.getContext();
       javax.security.auth.Policy currPolicy =
                     javax.security.auth.Policy.getPolicy();
       // Query policy now
       out.println("Policy permissions for this subject are " +
                  currPolicy.getPermissions(Subject.getSubject(acc),null));
       //Check Permissions
       out.println("Policy.impiles permission: "+ perm +" ? " +
           currPolicy.getPermissions(Subject.getSubject(acc),null).implies(perm));

       // CHECK USER'S PERMISSION VIA ACCESS CONTROLLER
       Subject.doAsPrivileged(s, new PrivilegedAction() {
            public Object run() {
                try {
                    AccessController.checkPermission(perm);
                    out.println("<br>");
                    out.println
                      ("AccessController checkPermission passed for permission: "
                       + perm);
                    out.println("<br>");
                } catch (IOException e) {
                    e.printStackTrace();
                }
                return null;
            }
        }, null);

      out.println("</BODY>");
      out.println("</HTML>");
   }
}
```

C

OracleAS JAAS Provider Admintool リファレンス

この章には、OracleAS JAAS Provider の Admintool のリファレンス情報が記載されています。この章の内容は次のとおりです。

- Admintool について
- Admintool のコマンドライン構文およびオプションの概要
- Admintool シェル
- Admintool の管理機能

注意：

- Admintool は、`system-jazn-data.xml` または Oracle Internet Directory とともに、ユーザー・リポジトリおよびポリシー・リポジトリとして使用できます。ただし、Admintool の場合、Oracle Internet Directory 内のユーザー情報とロール情報の読取りしかできません。
- ファイルベース・プロバイダを使用しているときに Admintool によって加えられた変更は、OC4J を再起動するまで有効にはなりません。

関連項目：

- 4-4 ページの「OracleAS JAAS Provider Admintool の概要」
- A-7 ページの「OracleAS JAAS Provider Admintool での標準の JDK ロギングの使用」

Admintool について

この項では、OracleAS JAAS Provider の Admintool を使用する際に役立つ情報を提供します。この項の内容は次のとおりです。

- Admintool の実行
- Admintool のユーザー・リポジトリの場所
- Admintool に対する認証
- Admintool でのカスタム・プリンシパルとパーミッションの使用

Admintool の実行

`java -jar` オプションを使用して OracleAS JAAS Provider の `jazn.jar` ファイルを実行することにより、Admintool を実行します。現行のディレクトリを `jazn.jar` が配置されているディレクトリにするか、または次の例に示すように、Java コマンドラインで `jazn.jar` へのパスを指定します。

```
% java -jar /myroot/mydir/jazn.jar ...
```

注意： `jazn.jar` は、通常、次の場所に配置されます。

ORACLE_HOME`/j2ee/home/jazn.jar`

デフォルトでは、Admintool は、現行ディレクトリ下にある `config` ディレクトリの `jazn.xml` 構成ファイルを検索します。これを変更するには、`oracle.security.jazn.config` システム・プロパティによって場所を直接指定するか、`oracle.home` プロパティまたは `oracle.j2ee.home` プロパティで Oracle ホームまたは J2EE ホームの場所を指定します。検索場所の優先度については、4-9 ページの「jazn.xml ファイル」を参照してください。

次の例では、`jazn.xml` ファイルの場所を指定しています。

```
% java -jar -Doracle.security.jazn.config=/tmp/jazn.xml jazn.jar ...
```

次の例では、J2EE ホームの場所を指定しています。この場合の *ORACLE_HOME* は、Oracle ホーム・ディレクトリへのパス、*instancename* は OC4J インスタンスの名前（home など）を表しています。この指定に基づき、Admintool は *ORACLE_HOME*`/j2ee/`*instancename*`/config` ディレクトリ内で `jazn.xml` を検索します。

```
% java -jar -Doracle.j2ee.home=ORACLE_HOME/j2ee/instancename jazn.jar ...
```

Admintool のユーザー・リポジトリの場所

ファイルベースのセキュリティ・プロバイダを使用する場合は、jazn.xml ファイルにある <jazn> 要素の location 属性によって、Admintool が使用するユーザー・リポジトリ（system-jazn-data.xml または jazn-data.xml）の場所が指定されます。デフォルトでは、これは system-jazn-data.xml ファイルですが、location の設定を更新して、アプリケーション固有の jazn-data.xml ファイルを使用することができます。

```
<jazn provider="XML" location="path/jazn-data.xml">
   ...
</jazn>
```

Oracle Identity Management を使用する場合、次の例のように、<jazn> の location 属性によって Oracle Internet Directory の場所が指定されます。ただし、Oracle Identity Management を使用する場合は location の値を手動で更新しないでください。この値は、Application Server Control を介して Oracle Internet Directory インスタンスを OC4J に関連付けるとき（8-7 ページの「Oracle Internet Directory と OC4J の関連付け」を参照）に自動的に設定されます）。次に例を示します。

```
<jazn provider="LDAP" location="ldap://myoid.oracle.com:389" default-realm="us" >
   ...
</jazn>
```

Admintool に対する認証

Admintool を実行する際には、ユーザーは、オプションとして -user および -password コマンドライン・オプションを使用して、認証を受ける必要があります。ユーザー認証は、次のいずれかの方法で行えます。

- 推奨される方法は、コマンドラインに -user および -password の設定を指定しない方法です。この方法を使用すると、次の例のように Admintool からユーザー名とパスワードの入力が求められます。

  ```
  % java -jar jazn.jar ...
  AbstractLoginModule username: username
  AbstractLoginModule password: password
  ...
  ```

  このモードでは、ユーザー名およびパスワードが求められ、これらを入力した後でのみ、指定したオプションが実行されます。次に例を示します。

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

  以降、この付録でこのような例を示しても、コマンドが実行される前（この例では、レルムがリスト表示される前）にユーザー名とパスワードが求められることは記載していません。

- コマンドラインで -user および -password オプションを使用することもできます。

  ```
  % java -jar jazn.jar -user username -password password...
  ```

  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るため、一般的にはこの方法は使用しないことをお薦めします。

  このモードでは、次のようなコマンドでただちにレルムがリスト表示されます。

  ```
  % java -jar jazn.jar -user myname -password mypassword -listrealms
  ```

  重要： コマンドラインで -user および -password オプションを指定する場合は、他のすべてのコマンドライン・オプションの前に置く必要があります。

前述のいずれのモードでも、コマンドラインで指定したオプションが実行されると、システム・プロンプトが再表示されます。さらに他の Admintool コマンドを実行するには、再度ツールを実行してもう 1 回認証される必要があります。

再度認証を受けることなく複数のコマンドを実行するには、Admintool シェル・モードを使用できます。このモードでは、C-7 ページの「Admintool シェル」で説明しているように、シェルを終了するまでは Admintool のプロンプトから繰り返しコマンドを実行できます。

Admintool でのカスタム・プリンシパルとパーミッションの使用

Admintool とカスタム・プリンシパルおよびパーミッションを連携させるには、カスタム・クラスを含んでいる JAR ファイルの場所を指定するように `classpath` プロパティを構成します。そのためには、次の例のように、`jazn.xml` ファイルの `<jazn>` 要素にある `<property>` サブ要素を使用します。

```
<jazn ... >
   ...
   <property name="classpath" value="/tmp/customPrincipal.jar" />
   ...
</jazn>
```

注意： 前のバージョンでは、回避策として JAR ファイルを `jre/lib/ext` ディレクトリに配置していました。この方法は現在でも可能ですが、お薦めしません。

Admintool のコマンドライン構文およびオプションの概要

Admintool には、管理機能用のコマンド・オプションが多数用意されています。一般的な構文は、次のとおりです。

```
% java -jar jazn.jar [-user username -password password] [option1 option2 ... ]
```

重要：

- `-user` および `-password` オプションを使用する場合（前項で説明したように、これは非推奨）は、コマンドライン上の他のすべてのオプションより前に指定する必要があります。
- Admintool の変更を有効にするには、OC4J を再起動します。

この項では、Admintool のコマンドのすべてのオプションを、詳細の参照先とともに示します。すべてのオプションとその構文は、`-help` オプションでもリスト表示できます。

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

コマンドライン・オプションの概要は、次のとおりです。

- 管理オプション

  `-activateadmin`

  関連項目：

  - C-15 ページの「管理操作」

- 認証オプション

  `-user username -password password`

  関連項目：

  - C-3 ページの「Admintool に対する認証」

- ログイン・モジュール・オプション

```
-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：

- C-12 ページの「ログイン・モジュールの追加と削除」
- C-16 ページの「ログイン・モジュールのリスト表示」

- 移行オプション

```
-convert filename realm
```

関連項目：

- C-18 ページの「principals.xml ファイルから JAAS への変換」

- パスワード管理オプション（ファイルベース・プロバイダのみ）

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

関連項目：

- C-14 ページの「パスワードのチェック（ファイルベース・プロバイダのみ）」
- C-14 ページの「パスワードの設定（ファイルベース・プロバイダのみ）」

- ポリシー・オプション

```
-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]
-revokeperm {realm {-user user|-role role} | principal_class principal_params}
            permission_class [permission_params]
```

関連項目：

- C-15 ページの「パーミッションの付与と取消し」
- C-17 ページの「パーミッションのリスト表示」

- ファイルベース・プロバイダのレルム操作オプション

```
-addrealm realm admin {adminpwd adminrole | adminrole
  userbase rolebase realmtype }
-addrole realm role
-adduser realm username password
-remrealm realm
-remrole realm role
-remuser realm user
```

重要： Oracle Internet Directory のレルムを作成する場合は、OracleAS JAAS Provider の Admintool を使用しないでください。このツールで作成したレルムが適しているのは、ファイルベース・プロバイダに対してのみです。Oracle Internet Directory で使用するための情報が不足しています。

関連項目：

- C-12 ページの「レルムの追加と削除（ファイルベース・プロバイダのみ）」
- C-13 ページの「ロールの追加と削除（ファイルベース・プロバイダのみ）」
- C-13 ページの「ユーザーの追加と削除（ファイルベース・プロバイダのみ）」

- 汎用のレルム・オプション

```
-grantrole role realm {user | -role to_role}
-revokerole role realm {user|-role from_role}
-listrealms realm
-listroles [realm [user | -role role]]
-listusers [realm [-role role | -perm permission]]
```

関連項目：

- C-16 ページの「ロールの付与と取消し」
- C-17 ページの「レルムのリスト表示」
- C-17 ページの「ロールのリスト表示」
- C-18 ページの「ユーザーのリスト表示」

- シェル・オプション

```
-shell
```

関連項目：

- 次の「Admintool シェル」を参照してください。

Admintool シェル

Admintool シェルを使用すると、UNIX に似た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。-shell オプションで、シェルを開始します。次に例を示します（ユーザーとパスワードを要求されたら、ユーザー oc4jadmin およびパスワードを入力します）。

```
% java -jar jazn.jar -shell
AbstractLoginModule username: oc4jadmin
AbstractLoginModule password: password
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 シェル・コマンドを使用します。シェル・コマンドのリストを表示するには、help コマンドを使用します。特定のシェル・コマンドの詳細が表示できるように、シェルでは man コマンドがサポートされています。

```
JAZN:> man admintoolcommand
```

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" ...
```

この項の以降の部分で、次の項目について説明します。

- Admintool コマンドライン・オプションのシェル・サポート
- Admintool シェルのディレクトリ構造
- Admintool の特別シェル・コマンドの概要

Admintool コマンドライン・オプションのシェル・サポート

Admintool シェルでは、Admintool コマンドラインと同じオプションがサポートされていますが、オプション名の前にハイフン（-）を含める必要はありません。（ハイフンを含めると、オプションは無視されます。）Admintool シェルを起動すると、次のシェル・コマンドラインは、（システム・プロンプトからの）次の Admintool コマンドラインと同じになります。

```
JAZN:> option1 option2 ... optionN

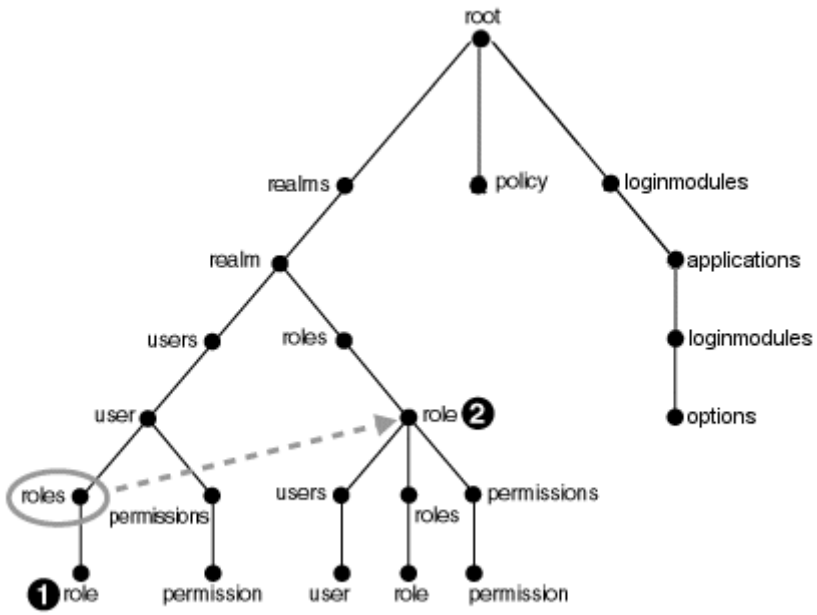
% java -jar jazn.jar -option1 -option2 ... -optionN
```

Admintool シェルのディレクトリ構造

Admintool シェルは、OracleAS 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 C-1 にノード構造を示します。

図 C-1 Admintool シェルのディレクトリ構造

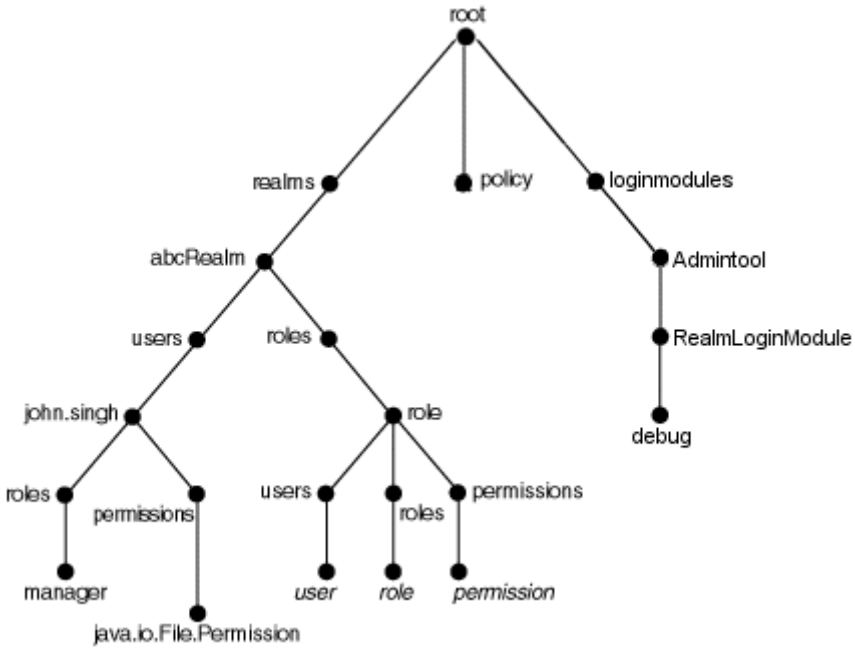


この構造では、user および role ノードがリンクされています。これは、user の下の roles リンクが、realm の下の roles リンクと同じであることを意味します。UNIX 用語では、この図で番号 1 の role は、この図で番号 2 の role へのシンボリック・リンクです。

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。

次の図 C-2 に、レルム abcRealm のノードを示します。

図 C-2 シェルのディレクトリ構造のサンプル



Admintool の特別シェル・コマンドの概要

この項では、次の Admintool のシェル・コマンドの概要を説明します。

- add、mkdir および mk: プロバイダ・データの作成
- cd: プロバイダ・データのナビゲート
- clear: 画面の消去
- exit: Admintool シェルの終了
- help: Admintool のシェル・コマンドのリスト表示
- ls: データのリスト表示
- man: Admintool の Man ページの表示
- pwd: 作業ディレクトリの表示
- rm: プロバイダ・データの削除
- set: 値の更新

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。

add、mkdir および mk: プロバイダ・データの作成

add *name* [*other_parameter*]
mkdir *name* [*other_parameter*]
mk *name* [*other_parameter*]

add、mkdir、mk の各コマンドは等価で、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。コマンドの結果は、現行ディレクトリの場所によって異なります。たとえば、現行のディレクトリがルートの場合、コマンドによってレルムが作成されます。現行のディレクトリが /realm/users の場合、コマンドによってユーザーが作成されます。状況によっては、これらのコマンドが -addrealm や -adduser などの Admintool コマンドと等価となる場合があります。

前述のシェル・コマンドには、追加のパラメータが必要となる場合があります。これは基本的に、同等の Admintool コマンドに対して必要となるパラメータと同じです。たとえば、次のシェル・コマンドを実行するとします。

```
JAZN:> add myrealm myuser mypassword myrole
```

/realms ディレクトリの下で実行する場合、このシェル・コマンドは次のコマンドと等価になります。

```
% java -jar jazn.jar -addrealm myrealm myuser mypassword myrole
```

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。

/ とパスを入力すると、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: Admintool シェルの終了

exit

exit コマンドを実行すると、Admintool シェルが終了します。

help: 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、有効なすべてのシェル・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: 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 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。

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 *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。コマンドの結果は、現行ディレクトリに応じて異なります。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。指定した名前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 の管理機能

この項では、Admintool の管理機能について説明します。内容は次のとおりです。

ログイン・モジュールの追加と削除

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

-addloginmodule オプションを指定すると、指定したアプリケーション用に新規ログイン・モジュールが構成されます。

control_flag には、標準の javax.security.auth.login.Configuration クラス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。これらのフラグ値の意味の概要は、9-17 ページの表 9-5「ログイン・モジュール制御フラグ」で説明しています。

ログイン・モジュール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。各ログイン・モジュールには、独自の個別オプション・セットがあります。

たとえば、MyLoginModule を必須モジュールとしてアプリケーション myapp に追加する場合に、このログイン・モジュールで debug オプションがサポートされているときは、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
```

レルムの追加と削除（ファイルベース・プロバイダのみ）

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

-addrealm オプションを指定すると、指定した名前と管理者でレルムが作成されます。

ファイルベース・プロバイダの場合、レルムの名前、レルムの管理者、管理者のパスワード、管理者のロールを指定します。

-addrealm *realm admin adminpwd adminrole*

-remrealm オプションを指定すると、レルムが削除されます。

重要： Oracle Internet Directory のレルムを作成する場合は、OracleAS JAAS Provider の Admintool を使用しないでください。このツールで作成したレルムが適しているのは、ファイルベース・プロバイダに対してのみです。Oracle Internet Directory で使用するための情報が不足しています。かわりに、Oracle Delegated Administration Service（DAS）を使用してください。

関連項目：

- 4-4 ページの「Delegated Administration Service の概要」
- 『Oracle Identity Management 委任管理ガイド』

たとえば、管理者 martha を指定してレルム employees を作成するときに、管理者のパスワードが mypass で、管理者がロール hr のメンバーである場合は、次のように入力します。

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

employees を削除する場合は、次のように入力します。

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

Admintool のシェルの場合

```
JAZN:> addrealm employees martha mypass hr
JAZN:> remrealm employees
```

ロールの追加と削除（ファイルベース・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。

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:> addrole foo fooRole
JAZN:> remrole foo fooRole
```

ユーザーの追加と削除（ファイルベース・プロバイダのみ）

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

-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。

次に示す例のように、コマンドラインではなく Admintool シェルを使用してユーザーを追加することをお薦めします。

```
% java -jar jazn.jar -shell
AbstractLoginModule username : oc4jadmin
AbstractLoginModule password : adminpassword
JAZN:> adduser jazn.com my_user my_password
```

Admintool のコマンドラインでユーザーを入力するのはあまりセキュアではありません。たとえば、UNIX システムでは、システム上の他のユーザーは、すべてのプロセスをリスト表示する ps -ef コマンドを使用することでパスワードを見ることができます。これに対し、Admintool シェルに入力したコマンドは Admintool のみが読み取れます。

ただし、コマンドラインでユーザーを追加する方法もサポートされています。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
```

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
```

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
JAZN:> remuser foo martha
```

パスワードの設定（ファイルベース・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
```

パスワードのチェック（ファイルベース・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 をパスワード用の -pw パラメータ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。」というレスポンスが戻され、ユーザー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。」というレスポンスが戻されます。

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
```

管理操作

-activateadmin

-activateadmin オプションを使用して、デフォルト・レルム内の oc4jadmin アカウント（旧 admin）をアクティブにし、パスワードを設定するには、次のようにします。（スタンドアロン OC4J では、ファイルベース・プロバイダに対してこのアカウントは最初は非アクティブになっています。）

```
% java -jar jazn.jar -activateadmin password
```

Admintool のシェルの場合

```
JAZN:> activateadmin password
```

注意： -activateadmin コマンドは、1 度だけのコマンドです。すでに管理アカウントがアクティブになっている場合、その旨を知らせるエラーがスローされます。

関連項目：

- 4-13 ページの「oc4jadmin アカウントのアクティブ化（スタンドアロンの OC4J）」

パーミッションの付与と取消し

-grantperm {*realm* {-user *user* |-role *role*} | *principal_class principal_params*}
permission_class [*permission_params*]
-revokeperm {*realm* {-user *user*|-role *role*} | *principal_class principal_params*}
permission_class [*permission_params*]

この構文の *principal_class* は、java.security.Principal インタフェースを実装するクラスの完全修飾名で、*principal_params* は、プリンシパル・クラスによって解釈される文字列です。

-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。

ユーザーまたはロールに対してパーミッションの付与または取消しを行う場合は常にレルムを指定しますが、プリンシパルに対してパーミッションの付与または取消しを行う場合は指定しません。

パーミッションの指定は、パーミッションの明示的なクラス名と、そのアクション・パラメータおよびターゲット・パラメータから構成されます。次の例のように、複数のアクション・パラメータとターゲット・パラメータを指定できることに注意してください。

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。C-4 ページの「Admintool でのカスタム・プリンシパルとパーミッションの使用」を参照してください。

たとえば、プリンシパル・パラメータ hobbes（LDAPPrincipal が理解できる値）およびパーミッション・パラメータ getProtectionDomain（RuntimePermission が理解できる値）を使用してプリンシパル LDAPPrincipal に RuntimePermission を付与するには、次のように入力します。

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.LDAPPrincipal hobbes
     java.lang.RuntimePermission getProtectionDomain
```

もう 1 つ例を示すと、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
```

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
```

Admintool のシェルの場合

```
JAZN:> grantperm foo -user martha java.io.FilePermission a.txt read,write
JAZN:> revokeperm 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 *role*}
-**revokerole** *role realm* {*user* |-role *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。

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
```

あるいは、ロール financial をロール finreporter に付与するには、次のように入力します。

```
% java -jar jazn.jar -grantrole financial foo -role finreporter
```

Admintool のシェルの場合

```
JAZN:> grantrole editor foo martha
JAZN:> revokerole editor foo martha
```

ログイン・モジュールのリスト表示

-**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_params*
permission_class [*permission_params*]

-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。

- -user オプションを使用した場合は、ユーザーに付与されているパーミッション
- -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
- プリンシパルに付与されているパーミッション

ユーザーまたはロールのパーミッションをリスト表示する場合は常にレルムを指定しますが、プリンシパルのパーミッションをリスト表示する場合は指定しません。

重要： PermissionClassManager および関連するクラスと操作（-listperms を含む）は、OC4J 10.1.3.x 実装では非推奨になっており、将来のリリースではサポートされない予定です。

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
```

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。引数としてレルムを指定すると、その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、指定したロールに付与されているすべてのロール

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
```

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

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

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

```
scott
admin
anonymous
```

Admintool のシェルの場合

```
JAZN:> listusers foo
```

principals.xml ファイルから JAAS への変換

-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。次に例を示します。

```
% java -jar jazn.jar \
       -convert $ORACLE_HOME/j2ee/home/config/principals.xml jazn.com
```

Admintool のシェルの場合

```
JAZN:> convert ORACLE_HOME/j2ee/home/config/principals.xml jazn.com
```

関連項目：

- 重要な追加情報が 7-17 ページの「principals.xml ファイルからのプリンシパルの移植」にありますので、参照してください。

D

OracleAS JAAS Provider 構成ファイル

この章では、OracleAS JAAS Provider の `jazn.xml` および `system-jazn-data.xml` 構成ファイルの参照情報について説明します。この章の内容は次のとおりです。

> **注意：** ファイルベース・プロバイダのユーザーおよびロールのリポジトリに関連する `system-jazn-data.xml` の要素もアプリケーション固有の `jazn-data.xml` ファイルに格納されることがあります。

jazn.xml の階層

jazn.xml ファイルは、次に示すような単純な階層構造です。

```
<jazn>
   <property>
```

注意： jazn.xml ファイル内の <jazn> の <jazn-web-app> サブ要素を使用しないでください。<jazn-web-app> 要素は orion-application.xml ファイル内で使用するためのものです。

関連項目：

- このファイルの概要は、4-9 ページの「jazn.xml ファイル」を参照してください。

jazn.xml の要素と属性

この項は、jazn.xml ファイルの要素をアルファベット順に並べた辞書です。

注意： 属性を扱う場合は、attribute="value" のように属性値を引用符で囲んでいます。

関連項目：

- このファイルの概要は、4-9 ページの「jazn.xml ファイル」を参照してください。

<jazn>

親要素： なし（ルート）

子要素： <property>

必須 必須、1 つのみ

これは OracleAS JAAS Provider を構成する jazn.xml ファイルの最上位レベルの要素です。

> **注意：** この要素は、アプリケーション・レベルの設定で（オプションで任意のサブ要素とともに）orion-application.xml ファイルに出現することもあります。

表 D-1 <jazn> 属性

名前	説明
config	値 : なし デフォルト : なし この属性は、OC4J 10.1.3.1 の実装では未使用です。
default-realm	値 : 文字列 デフォルト : なし これは、レルムが明示的に指定されていない場合に、認証または認可のリクエストで使用されるレルムを指定します。リポジトリ内に複数のレルムが定義されている場合に、デフォルトのレルムを指定する必要があります。 **注意：** この属性が設定されていないとデフォルトがありませんが、OC4J に付属する jazn.xml でデフォルト・レルムを jazn.com に設定するので注意してください。（orion-application.xml 内に default-realm 設定がないと、アプリケーションのデフォルト・レルムは jazn.xml 内に指定されたレルムになります。）
jaas-mode	値 : null|doas|doasprivileged デフォルト : null これは、JAAS モードを指定するために使用します。JAAS モードは、Subject クラスの静的メソッド doAs() および doAsPrivileged() の標準機能に関連付けられた OC4J 提供の密な認可機能です。設定が jaas-mode="doAs" となっている場合は、アプリケーション・モジュール（Web モジュールおよび EJB）が OC4J によって Subject.doAs() ブロック内で実行されます。設定が jaas-mode="doAsPrivileged" となっている場合は、アプリケーション・モジュールが Subject.doAsPrivileged() ブロック内で null アクセス制御コンテキストを使用して実行されます。jaas-mode="null"（デフォルト）と設定している場合は、どちらのメソッドも使用されずにモジュールが実行されます。 **関連項目：** 5-6 ページの「JAAS モードの概要」

表 D-1 <jazn> 属性（続き）

名前	説明
location	値 : 文字列 デフォルト : なし ファイルベース・プロバイダの場合、jazn.xml 内のこの属性でインスタンスレベルのユーザー・リポジトリの場所を指定します。OC4J に付属の jazn.xml ファイルでは、これは system-jazn-data.xml に指定されています。この設定は、絶対パスまたは jazn.xml ファイルの場所に対する相対パスです。（orion-application.xml 内のこの属性は、アプリケーション固有のユーザー・リポジトリを指定できます。） Oracle Identity Management（LDAP ベースのプロバイダ）では、これは Oracle Internet Directory インスタンスの URL を示し、Application Server Control によって Oracle Internet Directory インスタンスを OC4J インスタンスと関連付けるときに自動的に設定されます。
persistence	値 : NONE|ALL|VM_EXIT デフォルト : VM_EXIT これは、変更内容を system-jazn-data.xml ファイル、および（ファイルベース・プロバイダについて）必要な場合にアプリケーション・レベルの jazn-data.xml ファイルに書き込む頻度を制御する永続性モードを示します。"NONE" に設定すると、変更内容は書き込まれません。"ALL" に設定すると、変更のたびに変更内容が書き込まれます。"VM_EXIT"（デフォルト）に設定すると、JVM の終了時に変更内容が書き込まれます。
provider	値 : XML|LDAP デフォルト : XML インスタンス・レベルのセキュリティ・プロバイダ設定を指定します。jazn.xml の OC4J インスタンス・レベルで、provider 属性には、ポリシー・リポジトリ（system-jazn-data.xml の "XML" または Oracle Internet Directory の "LDAP"）を指定します。5-15 ページの「jazn.xml でのポリシー・リポジトリ設定」を参照してください。 **注意 :** アプリケーション・レベルのセキュリティ・プロバイダは、orion-application.xml 内の <jazn> 要素の provider 属性によって指定されます。（規則上は、orion-application.xml 内の provider="XML" は、セキュリティ・プロバイダが外部 LDAP プロバイダ、カスタム・ログイン・モジュール、または Oracle Access Manager であるときにも使用されます。）
schema-major-version	値 : 文字列 デフォルト : デフォルトなし jazn.xml XSD のメジャー・バージョン番号。この属性の値は、OC4J 10.1.3.x 実装で使用する場合は 10 です。 **注意 :** この属性は、jazn.xml の XSD で直接定義されていません。最上位レベルの OC4J XSD の attributeGroup の指定に依存します。
schema-minor-version	値 : 文字列 デフォルト : デフォルトなし jazn.xml XSD のマイナー・リリース番号。この属性の値は、OC4J 10.1.3.x 実装で使用する場合は 0 です。 **注意 :** この属性は、jazn.xml の XSD で直接定義されていません。最上位レベルの OC4J XSD の attributeGroup の指定に依存します。

<property>

親要素： <jazn>

子要素： なし

必須 オプション、0 以上

プロパティの設定を名前と値のペアで指定します。各セキュリティ・プロバイダと使用モードでは、固有のプロパティのセットをサポートします。たとえば、前のいくつかの章に示したように、LDAP ベースのプロバイダ（Oracle Identity Management 使用時）に固有のプロパティがあり、アイデンティティ管理フレームワークと Java SSO に固有のプロパティがあります。次に例を示します。

LDAP の場合：

```
<property name="ldap.protocol" value="no-ssl"/>
```

アイデンティティ管理フレームワークの場合：

```
<property name="idm.token.asserter.class"
          value="oracle.security.jazn.sso.SSOCookieTokenAsserter" />
```

Java SSO（アイデンティティ管理フレームワークの実装の 1 つ）の場合：

```
<property name="idm.authentication.name" value="JavaSSO" />
<property name="custom.sso.url.login"
          value="http://host:port/jsso/SSOLogin" />
```

表 D-2 <property> 属性

名前	説明
name	値：文字列 デフォルト：なし プロパティの名前。
value	値：文字列 デフォルト：なし プロパティの値。

system-jazn-data.xml の階層

この項では、system-jazn-data.xml ファイルの要素の階層について説明します。<jazn-data> の直下のサブ要素は、<jazn-policy>、<jazn-realm>、<jazn-loginconfig>、<jacc-repository>、<jazn-permission-classes> および <jazn-principal-classes> ですが、最後の 3 つはこのリリースではユーザー向けの使用を意図したものではありません。

注意： <jazn-realm> の下の要素は、アプリケーション固有の jazn-data.xml ファイルでも使用できます。

---

system-jazn-data.xml の階層

```
<jazn-data>

   <jazn-policy>
      <grant>
         <grantee>
            <display-name>
            <principals>
               <principal>
                  <realm-name>
                  <type>
                  <class>
                  <name>
            <codesource>
               <url>
         <permissions>
            <permission>
               <class>
               <name>
               <actions>

   <jazn-realm>
      <realm>
         <name>
         <users>
            <user>
               <name>
               <display-name>
               <description>
               <guid>
               <credentials>
         <roles>
            <role>
               <name>
               <display-name>
               <description>
               <guid>
               <members>
                  <member>
                     <type>
                     <name>
         <jazn-policy>              DO NOT USE AS SUBELEMENT OF <realm>
```

system-jazn-data.xml の階層

```
<jazn-loginconfig>
   <application>
      <name>
      <login-modules>
         <login-module>
            <class>
            <control-flag>
            <options>
               <option>
                  <name>
                  <value>

<jacc-repository>          NOT INTENDED FOR CUSTOMER USE; SUBHIERARCHY NOT SHOWN

<jazn-permission-classes> NOT INTENDED FOR CUSTOMER USE; SUBHIERARCHY NOT SHOWN

<jazn-principal-classes>  NOT INTENDED FOR CUSTOMER USE; SUBHIERARCHY NOT SHOWN
```

関連項目：

- このファイルの概要は、4-7 ページの「system-jazn-data.xml ファイル」を参照してください。
- 4-9 ページの「アプリケーション固有の jazn-data.xml ファイル（オプション）」

system-jazn-data.xml の要素と属性

この項は、system-jazn-data.xml ファイルの要素をアルファベット順に並べた辞書です。

注意：

- <jazn-realm> の下の要素は、アプリケーション固有の jazn-data.xml ファイルでも使用できます。
- このドキュメントで何度か説明しているように、system-jazn-data.xml 内の設定の大部分は、Application Server Control を通じて設定できます。
- 属性を扱う場合は、attribute="value" のように属性値を引用符で囲んでいます。

関連項目：

- このファイルの概要は、4-7 ページの「system-jazn-data.xml ファイル」を参照してください。
- 4-9 ページの「アプリケーション固有の jazn-data.xml ファイル（オプション）」

<actions>

親要素： <permission>

子要素： なし

必須 オプション、0 または 1

適用可能な場合、この要素は関連するパーミッションのクラスと名前に対して許可されているアクションを指定できます。次に例を示します。

```
<permission>
   <class>oracle.security.jazn.realm.RealmPermission</class>
   <name>jazn.com</name>
   <actions>droprealm</actions>
</permission>
```

<application>

親要素： <jazn-loginconfig>

子要素： <name>、<login-modules>

必須 オプション、0 以上

ログイン・モジュール構成では、この要素は（サブ要素によって）アプリケーションの名前を指定し、そのアプリケーションで使用するログイン・モジュールを構成します。

関連項目：

- 例については、D-14 ページの <jazn-loginconfig> を参照してください。

<class>

親要素： <principal>、<permission> または <login-module>

子要素： なし

必須 親要素内で必須、1 つのみ

この要素はいくつかの使用方法があります。

- （プリンシパルにパーミッションを付与する）<principal> 要素内で、プリンシパル・クラスの完全修飾名を指定します。クラスは、パーミッションのセットを付与されるプリンシパルを表すためにインスタンス化されます。
  例（"role" 型のプリンシパル用）：

  ```
  <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
  ```

- （プリンシパルにパーミッションを付与する）<permission> 要素内で、パーミッション・クラスの完全修飾名を指定します。
  例（EJB のアクセスに使用する RMI パーミッション用）：

  ```
  <class>com.evermind.server.rmi.RMIPermission</class>
  ```

- <login-module> 要素内で、ログイン・モジュール・クラスの完全修飾名を指定します。次に例を示します。

  ```
  <class>
     oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule
  </class>
  ```

<codesource>

親要素： <grantee>

子要素： <url>

必須 オプション、0 または 1

ポリシー構成では、<grantee> 要素内の <principals> 要素または <codesource> 要素のいずれかを使用して、付与するパーミッションを指定します。<codesource> 要素では、コードソースの URL を指定し、該当のコードソースへのパーミッションを付与します。

<control-flag>

親要素： <login-module>

子要素： なし

必須 親要素内で必須、1 つのみ

この要素では、ログイン・モジュールの次のいずれかのコントロール設定を指定します。

```
<control-flag>required</control-flag>

<control-flag>requisite</control-flag>

<control-flag>sufficient</control-flag>

<control-flag>optional</control-flag>
```

これらは、javax.security.auth.login.Configuration クラスの標準機能に応じて使用されます。全体の認証は、Required および Requisite のログイン・モジュールがすべて成功した場合にのみ成功します。ただし、Sufficient のログイン・モジュールが構成されていて成功した場合には、ログイン・モジュール・リスト内の Sufficient のログイン・モジュールの前にある Required および Requisite のログイン・モジュールのみ成功する必要があります。

関連項目：

- 制御フラグの設定の詳細は、9-17 ページの表 9-5「ログイン・モジュール制御フラグ」を参照してください。
- 例については、D-14 ページの <jazn-loginconfig> を参照してください。

<credentials>

親要素： <user>

子要素： なし

必須 オプション、0 または 1

この要素には、ユーザーの認証パスワードが含まれます。

デフォルトで、OC4J は system-jazn-data.xml（またはオプションで、アプリケーション固有の jazn-data.xml ファイル）に指定されたパスワードに対してパスワードの不明瞭化を行います。

クリアテキスト（判読可能）のパスワードを使用するかわりに、clear 属性を "true" に設定するか、パスワードの前に "!" を置きます。（この場合 "!" はパスワードの一部とみなされません。）ただし、クリアテキストのパスワードの使用はお薦めしません。

次の指定は同じ意味になります。

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

<credentials>!welcome</credentials>
```

表 D-3 <credentials> 属性

名前	説明
clear	値：true|false デフォルト：false これを "true" に設定すると、不明瞭化されたパスワードではなくクリアテキストのパスワードを使用します。

注意： clear 属性は、system-jazn-data.xml スキーマ定義で指定されませんが、OracleAS JAAS Provider のランタイム実装でサポートされます。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。
- 6-3 ページの「OC4J 構成ファイルのパスワードの不明瞭化」

<description>

親要素： <user> または <role>

子要素： なし

必須 オプション、0 または 1

これには、項目を説明するテキスト文字列（親要素に応じて、ユーザーまたはロール）が含まれます。
例（ユーザー oc4jadmin の場合）：

```
<description>The OC4J user with administrative privileges</description>
```

<display-name>

親要素： <grantee>、<user> または <role>

子要素： なし

必須 オプション、0 または 1

これには、項目に対して使用する表示名（親要素によって、権限受領者、ユーザーまたはロール）を指定するテキスト文字列が含まれます。
例（ユーザー oc4jadmin の場合）：

```
<display-name>OC4J Administrator</display-name>
```

<grant>

親要素： <jazn-policy>

子要素： <grantee>、<permissions>

必須 オプション、0 以上

ポリシー構成では、この要素には権限受領者にパーミッション・セットを割り当てる権限エントリが含まれます（コードソースまたはプリンシパルのセット）。

注意：（system-jazn-data.xml スキーマ定義で指定された）grantee-names 属性を使用しないでください。<grantee> サブ要素では、権限受領者を指定します。

関連項目：

- 例については、D-15 ページの <jazn-policy> を参照してください。

<grantee>

親要素： <grant>

子要素： <display-name>、<principals>、<codesource>

必須 親要素内で必須、1 つのみ

ポリシー権限を <grant> 要素によって指定する場合、（<permissions> 要素と組み合せて使用される）<grantee> 要素は、パーミッションが付与される対象（プリンシパルのセットまたはコードソース）を指定します。

関連項目：

- 例については、D-15 ページの <jazn-policy> を参照してください。

<guid>

親要素： <user> または <role>

子要素： なし

必須 オプション、0 または 1

この要素では、項目（親要素によって、ユーザーまたはロール）のグローバル一意識別子（GUID）を指定します。GUID は、ユーザーまたはロールを異なるセキュリティ・プロバイダに移行する場合などに、OracleAS JAAS Provider によって生成され、内部的に使用されます。ユーザーが自分で設定する項目ではありません。

<jacc-repository>

親要素： <jazn-data>

子要素： <jacc-policy>

必須 なし

この要素とそのサブ階層（ここに示しています）は、OC4J 10.1.3.1 の実装ではユーザー向けの使用を意図したものではありません。

```
<jacc-repository>
   <jacc-policy>
      <contextID>
      <excluded-policy>          SAME SUBHIERARCHY AS <jazn-policy>
      <unchecked-policy>         SAME SUBHIERARCHY AS <jazn-policy>
      <role-policy>              SAME SUBHIERARCHY AS <jazn-policy>
```

<jazn-data>

親要素： なし（ルート）

子要素： <jazn-policy>、<jazn-realm>、<jazn-loginconfig>（ユーザー向けの使用を意図した場合にのみ考慮）

必須 必須、1 つのみ

これは OracleAS JAAS Provider を構成する `system-jazn-data.xml` ファイルの最上位レベルの要素です。

表 D-4 <jazn-data> 属性

名前	説明
schema-major-version	値 : 文字列 デフォルト : デフォルトなし `system-jazn-data.xml` XSD のメジャー・バージョン番号。この属性の値は、OC4J 10.1.3.x 実装で使用する場合は 10 です。 **注意 :** この属性は、`system-jazn-data.xml` の XSD で直接定義されていません。最上位レベルの OC4J XSD の `attributeGroup` の指定に依存します。
schema-minor-version	値 : 文字列 デフォルト : デフォルトなし `system-jazn-data.xml` XSD のマイナー・リリース番号。この属性の値は、OC4J 10.1.3.x 実装で使用する場合は 0 です。 **注意 :** この属性は、`system-jazn-data.xml` の XSD で直接定義されていません。最上位レベルの OC4J XSD の `attributeGroup` の指定に依存します。

<jazn-loginconfig>

親要素： <jazn-data>

子要素： <application>

必須 オプション、0 または 1

これは、指定したアプリケーション（構成の一部として指定）に関連付けられたログイン・モジュールを構成する最上位レベルの要素です。次に、Oracle 提供の DBTableOraDataSourceLoginModule の例を示します（ここに示すサブ要素は、この付録内で何度か取り上げています）。

```
<jazn-loginconfig>
   <application>
      <name>application_name</name>
      <login-modules>
         <login-module>
            <class>
              oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule
            </class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>data_source_name</name>
                  <value>jdbc/OracleDS</value>
               </option>
               <option>
                  <name>table</name>
                  <value>userinfo</value>
               </option>
               <option>
                  <name>roles_fk_column</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>groupMembershipGroupFieldName</name>
                  <value>role</value>
               </option>
               <option>
                  <name>user_pk_column</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>passwordField</name>
                  <value>passWord</value>
               </option>
               <option>
                  <name>groupMembershipTableName</name>
                  <value>groupinfo</value>
               </option>
               <option>
                  <name>usernameField</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>casing</name>
                  <value>sensitive</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
```

```
   ...
</jazn-loginconfig>
```

<jazn-permission-classes>

親要素： <jazn-data>

子要素： <permission-class>

必須 なし

この要素とそのサブ階層（ここに示しています）は、OC4J 10.1.3.1 の実装ではユーザー向けの使用を意図したものではありません。

```
<jazn-permission-classes>
   <permission-class>
      <name>
      <description>
      <type>
      <class>
      <target-descriptors>
         <target-descriptor>
            <name>
            <description>
      <action-descriptors>
         <action-descriptor>
            <name>
            <description>
```

<jazn-policy>

親要素： <jazn-data>

子要素： <grant>

必須 オプション、0 または 1

これは、権限受領者（プリンシパルまたはコードソース）をパーミッション・セットと関連付けるポリシー権限を指定する、ポリシー構成の最上位レベルの要素です。ここに例を示します（ここに示すサブ要素は、この付録内で何度か取り上げています）。

```
<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/oc4j-administrators</name>
            </principal>
         </principals>
      </grantee>
      <permissions>
         <permission>
            <class>com.evermind.server.AdministrationPermission</class>
            <name>administration</name>
            <actions>administration</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
```

```
            <name>jazn.com</name>
            <actions>modifyrealmmetadata</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>createrealm</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>dropuser</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>droprealm</actions>
         </permission>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
         </permission>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>subject.propagation</name>
         </permission>
         <permission>
            <class>oracle.security.jazn.policy.RoleAdminPermission</class>
            <name>jazn.com/*</name>
         </permission>
      </permissions>
   </grant>
   ...
</jazn-policy>
```

注意： <jazn-policy> を <realm> のサブ要素として使用しないでください。

<jazn-principal-classes>

親要素： <jazn-data>

子要素： <principal-class>

必須 なし

この要素とそのサブ階層（ここに示しています）は、OC4J 10.1.3.1 の実装ではユーザー向けの使用を意図したものではありません。

```
<jazn-principal-classes>
   <principal-class>
      <name>
      <description>
      <type>
      <class>
      <name-description-map>
         <name-description-pair>
            <name>
            <description>
```

<jazn-realm>

親要素： <jazn-data>

子要素： <realm>

必須 オプション、0 または 1

これは、セキュリティ・レルムとそこに含まれるユーザーおよびロールを指定する、ユーザーおよびロールの情報の最上位レベルの要素です。ここに例を示します（ここに示すサブ要素は、この付録内で何度か取り上げています）。

```
<jazn-realm>
   <realm>
      <name>jazn.com</name>
      <users>
         <user deactivated="true">
            <name>anonymous</name>
            <guid>D3D41721D3E311DABFFC25CB9F57C041</guid>
            <description>The default guest/anonymous user</description>
         </user>
         <user>
            <name>oc4jadmin</name>
            <display-name>OC4J Administrator</display-name>
            <guid>D3DB1C00D3E311DABFFC25CB9F57C041</guid>
            <description>OC4J Administrator</description>
            <credentials>{903}r7VKkMgJqP8fkDZCG7YMo7UZnT/B+HcK</credentials>
         </user>
         ...
      </users>
      <roles>
         <role>
            <name>ascontrol_admin</name>
            <display-name>ASControl Admin Role</display-name>
            <description>Administrative role for ASControl</description>
            <guid>D3DB1C05D3E311DABFFC25CB9F57C041</guid>
            <members>
               <member>
                  <type>user</type>
                  <name>oc4jadmin</name>
               </member>
            </members>
         </role>
         <role>
            <name>oc4j-administrators</name>
            <display-name>OC4J Admin Role</display-name>
            <description>Administrative role for OC4J</description>
            <guid>D3DB1C02D3E311DABFFC25CB9F57C041</guid>
            <members>
               <member>
                  <type>user</type>
                  <name>oc4jadmin</name>
               </member>
               ...
            </members>
         </role>
         ...
      </roles>
   </realm>
</jazn-realm>
```

<login-module>

親要素： <login-modules>

子要素： <class>、<control-flag>、<options>

必須 親要素内で必須、1 つ以上

この要素は、クラス名、制御フラグ、ログイン・モジュールのオプション設定を指定するサブ要素とともに、所定のアプリケーションのログイン・モジュールを指定し、構成します。

関連項目：

- 例については、D-14 ページの <jazn-loginconfig> を参照してください。

<login-modules>

親要素： <application>

子要素： <login-module>

必須 親要素内で必須、1 つのみ

この要素は、1 つ以上の `<login-module>` サブ要素によって、所定のアプリケーションのログイン・モジュールのセットを構成します。

関連項目：

- 例については、D-14 ページの <jazn-loginconfig> を参照してください。

<member>

親要素： <members>

子要素： <type>、<name>

必須 オプション、0 以上

この要素は、適用可能なロールのメンバーの名前と、メンバーがユーザーか別のロールか（`<type>` サブ要素によって）を指定します。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<members>

親要素： <role>

子要素： <member>

必須 親要素内で必須、1 つのみ

この要素はロールのメンバーを指定します。メンバーはユーザーまたは他のロールのいずれかです。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<name>

親要素： <principal>、<realm>、<role>、<user>、<member>、<application> または <option>

子要素： なし

必須 親要素内で必須、1 つのみ

この要素はいくつかの使用方法があります。

- <realm> 要素内に、レルムの名前を指定します。次に例を示します。

  ```
  <name>jazn.com</name>
  ```

- <user> 要素内に、アプリケーション・レルム内のユーザーの一意の名前を指定します。次に例を示します。

  ```
  <name>oc4jadmin</name>
  ```

- <role> 要素内に、アプリケーション・レルム内のロールの一意の名前を指定します。次に例を示します。

  ```
  <name>oc4j-administrators</name>
  ```

- <role> の <member> サブ要素内に、ロールのメンバーの名前を指定します。
  例（ユーザー oc4jadmin がロールのメンバーの場合）：

  ```
  <name>oc4jadmin</name>
  ```

- <principal> 要素内に（プリンシパルにパーミッションを付与するために）、所定のレルム内のプリンシパルの一意の名前を指定します。次に例を示します。

  ```
  <name>jazn.com/oc4j-administrators</name>
  ```

- <application> 要素内に、ログイン・モジュールを構成するアプリケーションの完全修飾名を指定します。次に例を示します。

  ```
  <name>oracle.security.jazn.tools.Admintool</name>
  ```

- <option> 要素内に、ログイン・モジュール構成用のオプションの名前を指定します。（オプション値を示す <value> 要素を伴います。）
  例（DBTableOraDataSourceLoginModule のオプション）：

  ```
  <option>
     <name>data_source_name</name>
     <value>jdbc/OracleDS</value>
  </option>
  ```

（この要素は、この後 <permission> のサブ要素としても別個に掲載されています。）

<name>

親要素： <permission>

子要素： なし

必須 オプション、0 または 1

適用可能な場合、この要素ではパーミッション・クラスに対して意味を持つ、パーミッションの名前を指定できます。次に例を示します。

```
<permission>
   <class>com.evermind.server.rmi.RMIPermission</class>
   <name>login</name>
</permission>
```

（この要素は、前述のように、<principal>、<realm>、<role>、<user>、<member>、<application> または <option> のサブ要素としても別個に掲載されています。）

<option>

親要素： <options>

子要素： <name>、<value>

必須 親要素内で必須、1 つ以上

各 <option> 要素は、<name> サブ要素と <value> サブ要素を通じて、ログイン・モジュール用のオプション設定の名前と値を指定します。

関連項目：

- 例については、D-14 ページの <jazn-loginconfig> を参照してください。

<options>

親要素： <login-module>

子要素： <option>

必須 オプション、0 または 1

この要素は、<option> サブ要素を通じて、ログイン・モジュール用のオプション設定を指定します。

関連項目：

- 例については、D-14 ページの <jazn-loginconfig> を参照してください。

<permission>

親要素： <permissions>

子要素： <class>、<name>、<actions>

必須 親要素内で必須、1 つ以上

<permissions> 要素がポリシー権限構成で使用されるとき、各 <permission> サブ要素は該当のプリンシパルに付与される 1 つのパーミッションを指定します。

関連項目：

- 例については、D-15 ページの <jazn-policy> を参照してください。

<permissions>

親要素： <grant>

子要素： <permission>

必須 親要素内で必須、1 つのみ

ポリシー権限を <grant> 要素によって指定する場合、（<grantee> 要素と組み合せて使用される）<permissions> 要素は、<permission> サブ要素のセットによって、付与されるパーミッションを指定します。

注意： system-jazn-data.xml スキーマ定義では、この要素を必須要素として指定しませんが、OracleAS JAAS Provider ランタイム実装では任意の <grant> 要素内でこの要素の使用を必要とします。

関連項目：

- 例については、D-15 ページの <jazn-policy> を参照してください。

<principal>

親要素： <principals>

子要素： <realm-name>、<class>、<type>、<name>

必須 オプション、0 以上

<principals> 要素がポリシー権限構成で使用されるとき、各 <principal> サブ要素は該当のパーミッションが付与される 1 つのプリンシパルを指定します。

関連項目：

- 例については、D-15 ページの <jazn-policy> を参照してください。

<principals>

親要素： <grantee>

子要素： <principal>

必須 オプション、0 または 1

ポリシー構成では、<grantee> 要素内の <principals> 要素または <codesource> 要素のいずれかを使用して、付与するパーミッションを指定します。<principals> 要素ではパーミッションが付与されるプリンシパルのセットを指定します。

これらのパーミッションを付与されるサブジェクトでは、指定されたすべてのプリンシパルをサブジェクトが含んでいる必要があります。

関連項目：

- 例については、D-15 ページの <jazn-policy> を参照してください。

<realm>

親要素： <jazn-realm>

子要素： <name>、<users>、<roles>

必須 オプション、0 以上

この要素は、レルムと、レルムに属するユーザーとロールを指定します。

注意： <jazn-policy> を <realm> のサブ要素として使用しないでください。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<realm-name>

親要素： <principal>

子要素： なし

必須 オプション、0 または 1

プリンシパルにパーミッションを付与する場合、この要素にはプリンシパルが属するレルムの名前を指定します。（この値は、レルムが構成される <realm> 要素の <name> サブ要素の値に対応します。）次に例を示します。

```
<realm-name>jazn.com</realm-name>
```

レルム名が指定されていない場合、デフォルト・レルムとみなされます。

<role>

親要素： <roles>

子要素： <name>、<display-name>、<description>、<guid>、<members>

必須 オプション、0 以上

この要素はロールと、そのロールのメンバーを指定します。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<roles>

親要素： <realm>

子要素： <role>

必須 オプション、0 または 1

この要素はレルムに属するロールのセットを指定します。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<type>

親要素： <member>

子要素： なし

必須 必須、1 つのみ

`<member>` のサブ要素として、ロールのメンバーの指定時に、この要素はメンバーのタイプ（つまり、メンバーがユーザーか別のロールか）を指定します。次に例を示します。

```
<type>user</type>
```

（この要素は、この後、`<principal>` のサブ要素としても別個に掲載されています。）

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<type>

親要素： <principal>

子要素： なし

必須 オプション、0 または 1

<principal> のサブ要素として、プリンシパルへのパーミッションの付与の際に、この要素はオプションでプリンシパルのタイプ（プリンシパルがユーザーかロールか）を指定できます。次に例を示します。

```
<type>role</type>
```

（この要素は、前述のように、<member> のサブ要素としても別個に掲載されています。）

関連項目：

- 例については、D-15 ページの <jazn-policy> を参照してください。

<url>

親要素： <codesource>

子要素： なし

必須 親要素内で必須、1 つのみ

<codesource> 要素がポリシー権限構成で使用されるとき、<url> サブ要素は該当のパーミッションが付与されるコードソースの URL を指定します。次に例を示します。

```
"file:${oracle.home}/j2ee/home/jazn.jar"
```

（これは、5-4 ページの「Java 2 ポリシー・ファイルの作成または更新」に示す java2.policy ファイルと同じ形式です。）

<user>

親要素： <users>

子要素： <name>、<display-name>、<description>、<guid>、<credentials>

必須 オプション、0 以上

この要素はレルム内のユーザーを指定します。

表 D-5 <user> 属性

名前	説明
deactivated	値 : true | false デフォルト : false 構成ファイル内のユーザーを保持するが現在無効にする場合、この属性を "true" に設定できます。これは、たとえば、jazn.com レルム内の anonymous ユーザーの初期構成です。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<users>

親要素： <realm>

子要素： <user>

必須 オプション、0 または 1

この要素はレルムに属するユーザーのセットを指定します。

関連項目：

- 例については、D-17 ページの <jazn-realm> を参照してください。

<value>

親要素： <option>

子要素： なし

必須 親要素内で必須、1 つのみ

この要素はログイン・モジュールの構成用のオプションの値を指定します。（オプション名を示す <name> 要素を伴います。）
例（DBTableOraDataSourceLoginModule のオプション）：

```
<option>
   <name>data_source_name</name>
   <value>jdbc/OracleDS</value>
</option>
```

E

サード・パーティ・ライセンス

この付録には、Oracle Application Server に付属するサード・パーティ製品のサード・パーティ・ライセンスが記載されています。

Apache

このプログラムには、Apache Software Foundation（Apache）から提供されるサード・パーティ・コードが組み込まれています。Apache のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。

Apache のライセンス契約は、次の付属の Apache コンポーネントに適用されます。

- Apache HTTP Server
- Apache JServ
- mod_jserv
- 正規表現パッケージのバージョン 1.3
- Apache Expression Language（commons-el.jar 内にパッケージ）
- mod_mm 1.1.3
- Apache XML Signature および Apache XML Encryption のバージョン 1.4（Java 版）およびバージョン 1.0（C++ 版）
- log4j 1.1.1
- BCEL バージョン 5
- XML-RPC バージョン 1.1
- Batik バージョン 1.5.1
- ANT 1.6.2 および 1.6.5
- Crimson バージョン 1.1.3
- ant.jar
- wsif.jar
- bcel.jar
- soap.jar
- Jakarta CLI 1.0
- jakarta-regexp-1.3.jar
- JSP 標準タグ・ライブラリ 1.0.6 および 1.1
- Struts 1.1
- Velocity 1.3
- svnClientAdapter
- commons-logging.jar
- wsif.jar
- commons-el.jar
- standard.jar
- jstl.jar

The Apache Software License

License for Apache Web Server 1.3.29

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000-2002 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing
Applications,
 * University of Illinois, Urbana-Champaign.
```

License for Apache Web Server 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted"

means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and

do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Apache SOAP

このプログラムには、Apache Software Foundation（Apache）から提供されるサード・パーティ・コードが組み込まれています。Apache のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Apache ソフトウェアを含む）を使用する権利は、この製品に付随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、Apache ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または Apache から提供されません。

Apache SOAP License

Apache SOAP license 2.3.1

```
Copyright (c) 1999 The Apache Software Foundation.  All rights reserved.
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
      or by an individual or Legal Entity authorized to submit on behalf of
```

the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents

of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

mod_mm および mod_ssl

このプログラムには、Ralf S. Engelschall（Engelschall）から提供されるサード・パーティ・コードが組み込まれています。Engelschall のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（Engelschall ソフトウェアを含む）を使用する権利は、この製品に附随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。反対の内容が Oracle プログラム・ライセンス内にあった場合でも、mod_mm ソフトウェアは現状のままで Oracle から提供されるものであり、いかなる種類の保証またはサポートも Oracle または Engelschall から提供されません。

mod_mm

```
Copyright (c) 1999 - 2000 Ralf S. Engelschall.  All rights reserved.
This product includes software developed by Ralf S. Engelschall <rse@engelschall.com>
for use in the mod_ssl project (http://www.modssl.org/).
```

mod_ssl

```
Copyright (c) 1998-2001 Ralf S. Engelschall.  All rights reserved.
This product includes software developed by Ralf S. Engelschall <rse@engelschall.com>
for use in the mod_ssl project (http://www.modssl.org/).
```

OpenSSL

このプログラムには、OpenSSL Project から提供されるサード・パーティ・コードが組み込まれています。OpenSSL Project のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（OpenSSL ソフトウェアを含む）を使用する権利は、この製品に附随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。

OpenSSL License

```
/* ====================================================================
 * Copyright (c) 1998-2005 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
```

```
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

 Original SSLeay License
 -----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
```

```
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

Perl

このプログラムには、Comprehensive Perl Archive Network（CPAN）から提供されるサード・パーティ・コードが組み込まれています。CPAN のライセンス条件に基づき、Oracle は次のライセンス文書を表示することが求められています。ただし、Oracle プログラム（CPAN ソフトウェアを含む）を使用する権利は、この製品に附随する Oracle プログラム・ライセンスによって決定され、次のライセンス文書に含まれる条件でこの権利が変更されることはありません。

Perl Kit Readme

Copyright 1989-2001, Larry Wall

All rights reserved.

This program is free software; you can redistribute it and/or modify it under the terms of either:

1. the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version, or
2. the "Artistic License" which comes with this Kit.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See either the GNU General Public License or the Artistic License for more details.

You should have received a copy of the Artistic License with this Kit, in the file named "Artistic". If not, I'll be glad to provide one.

You should also have received a copy of the GNU General Public License along with this program in the file named "Copying". If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA or visit their Web page on the internet at http://www.gnu.org/copyleft/gpl.html.

For those of you that choose to use the GNU General Public License, my interpretation of the GNU General Public License is that no Perl script falls under the terms of the GPL unless you explicitly put said script under the terms of the GPL yourself. Furthermore, any object code linked with perl does not automatically fall under the terms of the GPL, provided such object code only adds definitions of subroutines and variables, and does not otherwise impair the resulting interpreter from executing any standard Perl script. I consider linking in C subroutines in this manner to be the moral equivalent of defining subroutines in the Perl language itself. You may sell such an object file as proprietary provided that you provide or offer to provide the Perl

source, as specified by the GNU General Public License. (This is merely an alternate way of specifying input to the program.) You may also sell a binary produced by the dumping of a running Perl script that belongs to you, provided that you provide or offer to provide the Perl source as specified by the GPL. (The fact that a Perl interpreter and your code are in the same binary file is, in this case, a form of mere aggregation.) This is my interpretation of the GPL. If you still have concerns or difficulties understanding my intent, feel free to contact me. Of course, the Artistic License spells all this out for your protection, so you may prefer to use that.

mod_perl 1.29 License

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 1996-2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 */
```

mod_perl 1.99_16 License

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of

the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents

of the NOTICE file are for informational purposes only and
do not modify the License. You may add Your own attribution
notices within Derivative Works that You distribute, alongside
or as an addendum to the NOTICE text from the Work, provided
that such additional attribution notices cannot be construed
as modifying the License.

You may add Your own copyright statement to Your modifications and
may provide additional or different license terms and conditions
for use, reproduction, or distribution of Your modifications, or
for any such Derivative Works as a whole, provided Your use,
reproduction, and distribution of the Work otherwise complies with
the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise,
any Contribution intentionally submitted for inclusion in the Work
by You to the Licensor shall be under the terms and conditions of
this License, without any additional terms or conditions.
Notwithstanding the above, nothing herein shall supersede or modify
the terms of any separate license agreement you may have executed
with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade
names, trademarks, service marks, or product names of the Licensor,
except as required for reasonable and customary use in describing the
origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or
agreed to in writing, Licensor provides the Work (and each
Contributor provides its Contributions) on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied, including, without limitation, any warranties or conditions
of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
PARTICULAR PURPOSE. You are solely responsible for determining the
appropriateness of using or redistributing the Work and assume any
risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
whether in tort (including negligence), contract, or otherwise,
unless required by applicable law (such as deliberate and grossly
negligent acts) or agreed to in writing, shall any Contributor be
liable to You for damages, including any direct, indirect, special,
incidental, or consequential damages of any character arising as a
result of this License or out of the use or inability to use the
Work (including but not limited to damages for loss of goodwill,
work stoppage, computer failure or malfunction, or any and all
other commercial damages or losses), even if such Contributor
has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
the Work or Derivative Works thereof, You may choose to offer,
and charge a fee for, acceptance of support, warranty, indemnity,
or other liability obligations and/or rights consistent with this
License. However, in accepting such obligations, You may act only
on Your own behalf and on Your sole responsibility, not on behalf
of any other Contributor, and only if You agree to indemnify,
defend, and hold each Contributor harmless for any liability
incurred by, or claims asserted against, such Contributor by reason
of your accepting any such warranty or additional liability.

Perl Artistic License

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
    a. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
    b. use the modified Package only within your corporation or organization.
    c. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
    d. make other distribution arrangements with the Copyright Holder.
4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:
    a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
    b. accompany the distribution with the machine-readable source of the Package with your modifications.

   c. give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

   d. make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whoever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package through the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

索引

A

B

C

D

E

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

あ

い

え

お

か

き

く

こ

さ

し

せ

そ

て

と

に

ね

は

ひ

ふ

ほ

み

ゆ

ら

り

れ

ろ