# BEA WebLogic Server™

## WebLogic Security プログラマーズ ガイド

バージョン 7.0
マニュアルの改訂 :2003 年 6 月 13 日

## 著作権



## 限定的権利条項



## 商標または登録商標



WebLogic Security プログラマーズ ガイド

| パート番号 | マニュアルの日付 | ソフトウェアのバージョン |
|---|---|---|
| なし | 2003 年 6 月 13 日 | BEA WebLogic Server バージョン 7.0 |

# 目次


## このマニュアルの内容

対象読者 ..... x
e-docs Web サイト ..... xii
このマニュアルの印刷方法 ..... xii
関連情報 ..... xii
サポート情報 ..... xiii
表記規則 ..... xiv

## 1. WebLogic Security のプログラミングの概要

このガイドの対象読者 ..... 1-1
セキュリティとは ..... 1-3
WebLogic Server でサポートされているセキュリティのタイプ ..... 1-5
認証 ..... 1-5
認可 ..... 1-6
J2EE セキュリティ ..... 1-6
セキュリティ API ..... 1-6
JAAS クライアント アプリケーション API ..... 1-7
Java JAAS クライアント アプリケーション API ..... 1-7
WebLogic JAAS クライアント アプリケーション API ..... 1-8
SSL クライアント アプリケーション API ..... 1-8
Java SSL クライアント アプリケーション API ..... 1-8
WebLogic SSL クライアント アプリケーション API ..... 1-9
その他の API ..... 1-9
Administration Console とセキュリティ ..... 1-10
セキュリティ タスクおよびコード例 ..... 1-11

## 2. Web アプリケーションのセキュリティ対策

J2EE セキュリティ モデル ..... 2-2
宣言による認可 ..... 2-2
プログラムによる認可 ..... 2-3
宣言による認可とプログラムによる認可 ..... 2-4

Web ブラウザでの認証 ..... 2-5
ユーザ名およびパスワードの認証 ..... 2-6
デジタル証明書の認証 ..... 2-7
複数の Web アプリケーション、クッキー、および認証 ..... 2-10
セキュアな Web アプリケーションの開発 ..... 2-11
BASIC 認証 Web アプリケーションの開発 ..... 2-11
FORM 認証 Web アプリケーションの開発 ..... 2-17
ID アサーションを使用した Web アプリケーションの認証 ..... 2-24
双方向 SSL を使用した Web アプリケーションの認証 ..... 2-24
Swing ベース認証 Web アプリケーションの開発 ..... 2-26
Web アプリケーションのデプロイメント ..... 2-26
Web アプリケーションでの宣言によるセキュリティの使用 ..... 2-28
Web アプリケーションのセキュリティ関連のデプロイメント記述子 ..... 2-28
web.xml デプロイメント記述子 ..... 2-29
auth-constraint ..... 2-29
security-constraint ..... 2-32
security-role ..... 2-33
security-role-ref ..... 2-34
user-data-constraint ..... 2-34
web-resource-collection ..... 2-35
weblogic.xml デプロイメント記述子 ..... 2-36
global-role ..... 2-37
security-permission ..... 2-39
security-permission-spec ..... 2-39
security-role-assignment ..... 2-40
Web アプリケーションでのプログラムによるセキュリティの使用 ..... 2-41
プログラムによる認証 API の使用 ..... 2-44

## 3. Java クライアントでの JAAS 認証の使用

JAAS と WebLogic Server ..... 3-1
JAAS 認証の開発環境 ..... 3-3
JAAS 認証 API ..... 3-4
JAAS クライアント アプリケーション コンポーネント ..... 3-9
WebLogic LoginModule 実装 ..... 3-11
JVM 全体のデフォルト ユーザと runAs() メソッド ..... 3-11

JAAS 認証を使用するクライアント アプリケーションの作成 ..... 3-12
JNDI 認証の使い方 ..... 3-22
Java クライアントの JAAS 認証コード例 ..... 3-24

## 4. Java クライアントでの SSL 認証の使用

JSSE および WebLogic Server ..... 4-1
JNDI 認証の使い方 ..... 4-2
SSL 証明書認証の開発環境 ..... 4-4
SSL 認証 API ..... 4-4
SSL クライアント アプリケーション コンポーネント ..... 4-9
SSL を利用するアプリケーションの記述 ..... 4-10
WebLogic Server 間の安全な通信 ..... 4-11
SSL クライアントの作成 ..... 4-11
SSLClient サンプル ..... 4-12
SSLSocketClient サンプル ..... 4-16
SSLClientServlet サンプル ..... 4-19
双方向 SSL 認証の使用 ..... 4-20
JNDI を使用した双方向 SSL 認証 ..... 4-21
ユーザ名マッパーの作成 ..... 4-24
WebLogic Server インスタンス間での双方向 SSL 認証の使用 ..... 4-25
サーブレットでの双方向 SSL 認証の使用 ..... 4-27
カスタム ホスト名検証の使い方 ..... 4-28
トラスト マネージャの使い方 ..... 4-30
ハンドシェーク完了リスナの使い方 ..... 4-32
SSLContext の使い方 ..... 4-33
SSL サーバ ソケット ファクトリの使い方 ..... 4-34
URL を使用した発信 SSL 接続 ..... 4-35
SSL クライアントのコード例 ..... 4-37

## 5. エンタープライズ JavaBean (EJB) のセキュリティ対策

J2EE セキュリティ モデル ..... 5-1
宣言による認可 ..... 5-2
プログラムによる認可 ..... 5-3
宣言による認可とプログラムによる認可 ..... 5-4
EJB での宣言によるセキュリティの使用 ..... 5-5

EJB のセキュリティ関連のデプロイメント記述子 ..... 5-7
ejb-jar.xml デプロイメント記述子 ..... 5-7
method ..... 5-8
method-permission ..... 5-9
role-name ..... 5-10
run-as ..... 5-10
security-identity ..... 5-11
security-role ..... 5-11
security-role-ref ..... 5-12
unchecked ..... 5-13
use-caller-identity ..... 5-13
weblogic-ejb-jar.xml デプロイメント記述子 ..... 5-14
client-authentication ..... 5-15
client-cert-authentication ..... 5-16
confidentiality ..... 5-16
global-role ..... 5-17
identity-assertion ..... 5-19
iiop-security-descriptor ..... 5-20
integrity ..... 5-21
principal-name ..... 5-21
role-name ..... 5-22
run-as-identity-principal ..... 5-22
security-permission ..... 5-24
security-permission-spec ..... 5-24
security-role-assignment ..... 5-25
transport-requirements ..... 5-26
EJB でのプログラムによるセキュリティの使用 ..... 5-26

## 6. ネットワーク接続フィルタの使い方

ネットワーク接続フィルタを使用する利点 ..... 6-1
ネットワーク接続フィルタ API ..... 6-2
接続フィルタのインタフェース ..... 6-2
ConnectionFilter インタフェース ..... 6-2
ConnectionFilterRulesListener インタフェース ..... 6-3
接続フィルタのクラス ..... 6-4

ConnectionFilterImpl クラス ..... 6-4
ConnectionEvent クラス ..... 6-4
接続フィルタ ルールの作成ガイドライン ..... 6-5
接続フィルタ ルールの構文 ..... 6-5
接続フィルタ ルールの種類 ..... 6-6
接続フィルタ ルールの評価方法 ..... 6-7
WebLogic 接続フィルタのコンフィグレーション ..... 6-8
カスタム接続フィルタの開発 ..... 6-8
接続フィルタのサンプル ..... 6-9
SimpleConnectionFilter サンプル ..... 6-10
SimpleConnectionFilter2 サンプル ..... 6-10
ネットワーク接続をフィルタ処理する場合の accept メソッドの例 ... 6-11

**7. Java セキュリティを使用しての WebLogic リソースの保護**

J2EE セキュリティを使用しての WebLogic リソースの保護 ..... 7-1
Java セキュリティ マネージャを使用しての WebLogic リソースの保護 .... 7-2
Java セキュリティ マネージャの設定 ..... 7-3
weblogic.policy ファイルの修正 ..... 7-4
アプリケーション型のセキュリティ ポリシーの設定 ..... 7-5
アプリケーション固有のセキュリティ ポリシーの設定 ..... 7-6
レコーディング セキュリティ マネージャ ユーティリティの使い方 ... 7-7

**A. 非推奨のセキュリティ API**

**索引**

# このマニュアルの内容

このマニュアルの内容は以下のとおりです。

- 第 1 章「WebLogic Security のプログラミングの概要」では、このマニュアルの対象読者、セキュリティの必要性、および WebLogic Security のアプリケーション プログラミング インタフェース (API) について解説します。
- 第 2 章「Web アプリケーションのセキュリティ対策」では、Web アプリケーションにセキュリティを実装する方法について説明します。
- 第 3 章「Java クライアントでの JAAS 認証の使用」では、Java クライアントに JAAS 認証を実装する方法について説明します。
- 第 4 章「Java クライアントでの SSL 認証の使用」では、Java クライアントに SSL およびデジタル証明書認証を実装する方法について説明します。
- 第 5 章「エンタープライズ JavaBean (EJB) のセキュリティ対策」では、エンタープライズ JavaBean にセキュリティを実装する方法について説明します。
- 第 6 章「ネットワーク接続フィルタの使い方」では、ネットワーク接続フィルタを実装する方法について説明します。
- 第 7 章「Java セキュリティを使用しての WebLogic リソースの保護」では、Java セキュリティを使用しての WebLogic リソースの保護について説明します。
- 付録 A「非推奨のセキュリティ API」では、`weblogic.security` パッケージのうち、API が非推奨とされているものについて示します。

**注意：** このマニュアルでは、WebLogic セキュリティ プロバイダとカスタム セキュリティ プロバイダのコンフィグレーション方法については説明していません。WebLogic セキュリティ プロバイダとカスタム セキュリティ プロバイダのコンフィグレーションの詳細については、『WebLogic Security の管理』を参照してください。

**注意：** このマニュアルは、WebLogic Server で使用するカスタム セキュリティ プロバイダを記述しようと考えている開発者向けではありません。カスタム セキュリティ プロバイダの記述方法については説明していません。カスタム セキュリティ プロバイダの開発方法については、『WebLogic Security サービスの開発』を参照してください。

# 対象読者

このマニュアルは、以下の読者を対象としています。

- アプリケーション開発者

  クライアント アプリケーションの開発と、Web アプリケーションおよびエンタープライズ JavaBean (EJB) へのセキュリティ機能の付加を主な業務とする Java プログラマのことです。他のエンジニアリング チームや品質保証 (QA) チーム、データベース チームと連携して、セキュリティ機能を実装します。アプリケーション開発者は、サーブレット、JSP、JSEE などの J2EE コンポーネントを含む Java、および Java セキュリティについて実用的かつ深い知識を備えています。

  アプリケーション開発者は、WebLogic.Security と Java 2 セキュリティ アプリケーション プログラミング インタフェース (API) を使用してアプリケーションのセキュリティを確保します。したがって、このマニュアルは Web アプリケーション、Java アプリケーション、およびエンタープライズ JavaBeans (EJB) のセキュリティを確保するためにそれらの API を使用する手順を紹介します。

- セキュリティ開発者

  WebLogic Server に統合されるセキュリティ製品のシステム アーキテクチャとインフラストラクチャの定義と、WebLogic Server で使用するカスタム セキュリティ プロバイダの開発を主な業務とする開発者のことです。アプリケーション設計者と連携して、セキュリティ アーキテクチャを確実に設計に従って、セキュリティ ホールが発生しないように実装します。また、セキュリティが確実に正しくコンフィグレーションされるよう、サーバ管理者とも連携します。セキュリティ開発者は、認証、認可、監査 (AAA)、Java Management eXtension (JMX) などの Java に対する深い知識、および WebLogic Server とセキュリティ プロバイダの機能に対する実践的な知識をはじめとしたセキュリティ概念をしっかりと理解しています。

セキュリティ開発者は、セキュリティ サービス プロバイダ インタフェース (SSPI) を使用して WebLogic Server で使用するカスタム セキュリティ プロバイダを開発します。しかし、このマニュアルではこのタスクについては扱いません。SSPI を使用してカスタム セキュリティ プロバイダを開発する方法については、『WebLogic Security サービスの開発』を参照してください。

- サーバ管理者

  アプリケーション設計者と密接に連携しながら、サーバおよびサーバ上で動作するアプリケーションのセキュリティ方式の設計、潜在的なセキュリティ リスクの特定、およびセキュリティ上の問題を防止するコンフィグレーションの提案を行う管理者のことです。関連する責務として、重要なプロダクション システムの保守、セキュリティ レルムのコンフィグレーションと管理、サーバ リソースとアプリケーション リソースへの認証および認可方式の実装、セキュリティ機能のアップグレード、およびセキュリティ プロバイダのデータベースの保守などが含まれる場合もあります。サーバ管理者は、Web アプリケーションと EJB のセキュリティ、公開鍵セキュリティ、および SSL を含む、Java セキュリティ アーキテクチャについて深い知識を備えています。

- アプリケーション管理者

  サーバ管理者と共同でセキュリティ コンフィグレーション、認証および認可方式を実装および管理したり、定義されたセキュリティ レルムでデプロイされているアプリケーション リソースへのアクセスを設定および管理したりする管理者のことです。アプリケーション管理者は、セキュリティの概念や Java セキュリティ アーキテクチャの一般的な知識を持っています。アプリケーション管理者は、Java、XML、デプロイメント記述子を理解し、サーバ ログおよび監査ログでセキュリティ イベントを特定できます。

  管理者は通常、アプリケーションを実際に稼働させるときに、Administration Console を使用してアプリケーションのデプロイ、コンフィグレーション、および管理を行いますが、アプリケーション開発者も Administration Console を使用して実際の稼働前にアプリケーションをテストする場合があります。アプリケーションをテストするときには、最低でも、アプリケーションがデプロイされてコンフィグレーションされている必要があります。このマニュアルではセキュリティに関連している場合は管理の側面にも言及しますが、Administration Console を使用してセキュリティ タスクを行う方法の説明については『WebLogic Security の管理』、『WebLogic リソースのセキュリティ』、Administration Console オンライン ヘルプなどのマニュアルを参照先として示します。

# e-docs Web サイト

BEA 製品のドキュメントは、BEA の Web サイトで入手できます。BEA のホーム ページで [ 製品のドキュメント ] をクリックします。

# このマニュアルの印刷方法

Web ブラウザの [ ファイル | 印刷 ] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 章ずつ印刷できます。

このマニュアルの PDF 版は、WebLogic Server の Web サイトで入手できます。PDF を Adobe Acrobat Reader で開くと、マニュアルの全体 ( または一部分 ) を書籍の形式で印刷できます。PDF を表示するには、WebLogic Server ドキュメントのホーム ページを開き、[ ドキュメントのダウンロード ] をクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader は Adobe の Web サイト (http://www.adobe.co.jp) で無料で入手できます。

# 関連情報

このマニュアルの他、以下のマニュアルでも WebLogic Security サービス情報について説明しています。

- 『WebLogic Security の紹介』- このマニュアルでは、WebLogic Security サービスの特徴についてまとめ、そのアーキテクチャと機能の概要を示します。WebLogic Security サービスを理解する上で基本となるマニュアルです。
- 『プロダクション環境のロックダウン』- このマニュアルでは、WebLogic Server をプロダクション環境にデプロイする前に検討すべき重要なセキュリティ対策について説明します。

- 『WebLogic Security サービスの開発』- このマニュアルでは、セキュリティ ベンダおよびアプリケーション開発者を対象に、WebLogic Server 用のカスタム セキュリティ プロバイダを開発するのに必要な情報を提供します。
- 『WebLogic Security の管理』- このマニュアルでは、WebLogic Server のセキュリティをコンフィグレーションする方法と、互換性セキュリティの使い方について説明します。
- 『WebLogic リソースのセキュリティ』- このマニュアル では、さまざまなタイプの WebLogic リソースを紹介し、WebLogic Server を使用してそれらのリソースを保護するための情報を提供します。
- 『BEA WebLogic Server 7.0 へのアップグレード』の「WebLogic Server 6.x からバージョン 7.0 へのアップグレード」の章の「セキュリティのアップグレード」の節 - この節では、以前のバージョンの BEA WebLogic Server を WebLogic Server 7.0 にアップグレードするために必要な手順およびその他の情報を示します。また、アプリケーションを WebLogic Server の以前のバージョンからバージョン 7.0 に移動する方法についても説明します。
- Administration Console オンライン ヘルプ - このマニュアルでは、Administration Console を使用してセキュリティ タスクを実行する方法について説明します。
- 『BEA WebLogic Server 7.0 の FAQ』の「FAQ: セキュリティ」- このマニュアルでは、WebLogic Server のセキュリティに関してよくある質問 (FAQ) に回答しています。
- WebLogic クラスに関する Javadoc のページ - このページでは、WebLogic Server 7.0 ソフトウェアで提供およびサポートされている WebLogic セキュリティ パッケージのリファレンスを提供します。

# サポート情報

BEA のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで docsupport-jp@beasys.com までお送りください。寄せられた意見については、ドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、ご使用のソフトウェアの名前とバージョン、およびドキュメントのタイトルと日付をお書き添えください。本バージョンの BEA WebLogic Server について不明な点がある場合、または BEA WebLogic Server のインストールおよび動作に問題がある場合は、BEA WebSupport (www.bea.com) を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

# 表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

| 表記法 | 適用 |
|---|---|
| 〔Ctrl〕+ 〔Tab〕 | 複数のキーを同時に押すことを示す。 |
| *斜体* | 強調または書籍のタイトルを示す。 |

| 表記法 | 適用 |
| --- | --- |
| 等幅テキスト | コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。<br>例：<br>`import java.util.Enumeration;`<br>`chmod u+w *`<br>`config/examples/applications`<br>`.java`<br>`config.xml`<br>`float` |
| *斜体の等幅テキスト* | コード内の変数を示す。<br>例：<br>`String CustomerName;` |
| すべて大文字のテキスト | デバイス名、環境変数、および論理演算子を示す。<br>例：<br>LPT1<br>BEA_HOME<br>OR |
| { } | 構文の中で複数の選択肢を示す。 |
| [ ] | 構文の中で任意指定の項目を示す。<br>例：<br>`java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]` |
| \| | 構文の中で相互に排他的な選択肢を区切る。<br>例：<br>`java weblogic.deploy [list\|deploy\|undeploy\|update] password {application} {source}` |

| 表記法 | 適用 |
| --- | --- |
| . . . | コマンドラインで以下のいずれかを示す。<br>■ 引数を複数回繰り返すことができる。<br>■ 任意指定の引数が省略されている。<br>■ パラメータや値などの情報を追加入力できる。 |
| .<br>.<br>. | コード サンプルまたは構文で項目が省略されていることを示す。 |

# 1 WebLogic Security のプログラミングの概要

この節では、以下のトピックについて説明します。


- 1-1 ページ「このガイドの対象読者」
- 1-3 ページ「セキュリティとは」
- 1-5 ページ「WebLogic Server でサポートされているセキュリティのタイプ」
- 1-6 ページ「セキュリティ API」
- 1-10 ページ「Administration Console とセキュリティ」
- 1-11 ページ「セキュリティ タスクおよびコード例」


## このガイドの対象読者

このマニュアルは、以下の読者を対象としています。

- アプリケーション開発者

  クライアント アプリケーションの開発と、Web アプリケーションおよびエンタープライズ JavaBean (EJB) へのセキュリティ機能の付加を主な業務とする Java プログラマのことです。他のエンジニアリング チームや品質保証 (QA) チーム、データベース チームと連携して、セキュリティ機能を実装します。アプリケーション開発者は、サーブレット、JSP、JSEE などの J2EE コンポーネントを含む Java、および Java セキュリティについて実用的かつ深い知識を備えています。

  アプリケーション開発者は、WebLogic.Security と Java 2 セキュリティ アプリケーション プログラミング インタフェース (API) を使用してアプリケーションのセキュリティを確保します。したがって、このマニュアルは Web アプリケーション、Java アプリケーション、およびエンタープライズ

JavaBeans (EJB) のセキュリティを確保するためにそれらの API を使用する手順を紹介します。

- セキュリティ開発者

  WebLogic Server に統合されるセキュリティ製品のシステム アーキテクチャとインフラストラクチャの定義と、WebLogic Server で使用するカスタム セキュリティ プロバイダの開発を主な業務とする開発者のことです。アプリケーション設計者と連携して、セキュリティ アーキテクチャを確実に設計に従って、セキュリティ ホールが発生しないように実装します。また、セキュリティが確実に正しくコンフィグレーションされるよう、サーバ管理者とも連携します。セキュリティ開発者は、認証、認可、監査 (AAA)、Java Management eXtension (JMX) などの Java に対する深い知識、および WebLogic Server とセキュリティ プロバイダの機能に対する実践的な知識をはじめとしたセキュリティ概念をしっかりと理解しています。

  セキュリティ開発者は、セキュリティ サービス プロバイダ インタフェース (SSPI) を使用して WebLogic Server で使用するカスタム セキュリティ プロバイダを開発します。しかし、このマニュアルではこのタスクについては扱いません。SSPI を使用してカスタム セキュリティ プロバイダを開発する方法については、『WebLogic Security サービスの開発』を参照してください。

- サーバ管理者

  アプリケーション設計者と密接に連携しながら、サーバおよびサーバ上で動作するアプリケーションのセキュリティ方式の設計、潜在的なセキュリティ リスクの特定、およびセキュリティ上の問題を防止するコンフィグレーションの提案を行う管理者のことです。関連する責務として、重要なプロダクション システムの保守、セキュリティ レルムのコンフィグレーションと管理、サーバ リソースとアプリケーション リソースへの認証および認可方式の実装、セキュリティ機能のアップグレード、およびセキュリティ プロバイダのデータベースの保守などが含まれる場合もあります。サーバ管理者は、Web アプリケーションと EJB のセキュリティ、公開鍵セキュリティ、および SSL を含む、Java セキュリティ アーキテクチャについて深い知識を備えています。

- アプリケーション管理者

  サーバ管理者と共同でセキュリティ コンフィグレーション、認証および認可方式を実装および管理したり、定義されたセキュリティ レルムでデプロイされているアプリケーション リソースへのアクセスを設定および管理したりする管理者のことです。アプリケーション管理者は、セキュリティの概念や Java セキュリティ アーキテクチャの一般的な知識を持っています。アプリケーション管理者は、Java、XML、デプロイメント記述子を理解し、サーバ ログおよび監査ログでセキュリティ イベントを特定できます。

  管理者は通常、アプリケーションを実際に稼働させるときに、Administration Console を使用してアプリケーションのデプロイ、コンフィグレーション、および管理を行いますが、アプリケーション開発者も Administration Console を使用して実際の稼働前にアプリケーションをテストする場合があります。アプリケーションをテストするときには、最低でも、アプリケーションがデプロイされてコンフィグレーションされている必要があります。このマニュアルではセキュリティに関連している場合は管理の側面にも言及しますが、Administration Console を使用してセキュリティ タスクを行う方法の説明については『WebLogic Security の管理』、『WebLogic リソースのセキュリティ』、Administration Console オンライン ヘルプなどのマニュアルを参照先として示します。

**注意：** このマニュアルでは、WebLogic セキュリティ プロバイダとカスタム セキュリティ プロバイダのコンフィグレーション方法については説明していません。WebLogic セキュリティ プロバイダとカスタム セキュリティ プロバイダのコンフィグレーションの詳細については、『WebLogic Security の管理』を参照してください。

**注意：** このマニュアルは、WebLogic Server で使用するカスタム セキュリティ プロバイダを記述しようと考えている開発者向けではありません。カスタム セキュリティ プロバイダの記述方法については説明していません。カスタム セキュリティ プロバイダの開発方法については、『WebLogic Security サービスの開発』を参照してください。

# セキュリティとは

セキュリティとは、コンピュータに保存されているデータまたはコンピュータ間でやりとりされるデータが危険にさらされないことを保証する技術です。ほとん

どのセキュリティ対策は、証明データとデータ暗号化を利用します。一般に証明データは、ユーザに特定のアプリケーションまたはシステムへのアクセスを許可する秘密の単語または句です。データ暗号化とは、その秘密の単語または句を保持しているか提供しなければ解釈できないような形式にデータを変換することです。

電子商取引 (e コマース ) 向けアプリケーションなどの分散アプリケーションでは、悪意のある何者かがデータを横取りし、処理を混乱させ、不正な入力を行う起点となるような多数のアクセス ポイントが提供されます。ビジネスの分散化が進むにつれて、セキュリティが侵害される可能性も大きくなります。したがって、アプリケーションの分散に伴い、その基盤となる分散コンピューティング ソフトウェアによってセキュリティを実現することがますます重要になります。

アプリケーション サーバは、エンド ユーザと貴重なデータやリソースとの間の重要なレイヤに位置しています。WebLogic Server は、リソースの保護に関して認証および暗号化サービスを提供します。しかし、こうしたサービスでは、デプロイメント環境の弱点を見つけ出して悪用することでアクセスを取得した侵入者から、リソースを守ることはできません。

したがって、インターネットまたはイントラネット上で WebLogic Server をデプロイする場合には、独立したセキュリティ専門家に依頼して、セキュリティ プランと手順を検討してもらい、インストール済みシステムの監査を受け、改善点のアドバイスを受けるとよいでしょう。

セキュリティ問題と適切なセキュリティ対策についてできる限り知識を増やすことも重要です。マニュアル『プロダクション環境のロックダウン』では、WebLogic Server をプロダクション環境にデプロイする前に検討すべき重要なセキュリティ対策について説明してあります。マニュアル『WebLogic リソースのセキュリティ』は、さまざまなタイプの WebLogic リソースを紹介し、WebLogic Server を使用してそれらのリソースを保護するための情報を提供します。Web サーバのセキュリティ対策の最新情報については、カーネギー メロン大学が運営する CERT (TM) Coordination Center が公開している「Security Improvement Modules, Security Practices, and Technical Implementations」にも目を通すことをお勧めします。

BEA の「security advisories」ページで推奨されている対策は是非、実行してください。BEA 製品に関して問題が発生した場合には、BEA から、その報告と適切な対策を示した指示が配信されます。サイトのセキュリティ問題を担当されている方は、今後、BEA からセキュリティ関連の問題の通知を受信できるよう、登録を行ってください。BEA では、BEA 製品に関するセキュリティ問題をご報告いただくための電子メール アドレス (`security-report@bea.com`) も用意して

います。また、リリースされている各サービス パックの適用もお勧めします。サービス パックには、製品の各バージョンおよび以前にリリースされた各サービス パックのすべてのバグの修正が含まれています。

さらに、WebLogic Server のプロダクション環境の保護に役に立つ、パートナ製品もあります。詳細については、BEA パートナのページを参照してください。

# WebLogic Server でサポートされているセキュリティのタイプ

WebLogic Server は以下のセキュリティ メカニズムをサポートしています。


- 1-5 ページ「認証」
- 1-6 ページ「認可」
- 1-6 ページ「J2EE セキュリティ」


## 認証

認証とは、呼び出し側とサービス プロバイダが、特定のユーザまたはシステムの代わりに動作していることを証明する際に使用するメカニズムのことです。認証は、資格を使用して「あなたは誰」という問いに答えます。証明が双方向で行われる場合、相互認証と呼ばれます。

WebLogic Server は、ユーザ名およびパスワードによる認証と証明書による認証をサポートしています。WebLogic Server は、証明書認証について一方向と双方向の SSL 認証を両方ともサポートしています。双方向の SSL 認証は、相互認証の一形態です。

WebLogic Server で、認証プロバイダはユーザまたはシステム プロセスの ID を証明するために使用します。認証プロバイダでは、ID 情報を記憶したり、転送したり、その情報が必要な場合に ( サブジェクトを通じて ) システムのさまざまなコンポーネントで利用できるようにしたりします。認証プロバイダは、Web アプリケーションおよび EJB デプロイメント記述子ファイル、または Administration Console、あるいはその両方を使用してコンフィグレーションできます。

# 認可

認可とは、ユーザの ID などの情報に基づいて、ユーザと WebLogic リソースとのやり取りを管理するプロセスのことです。言い換えれば、認可とは「自分は何にアクセスできますか」という質問に答えるものです。

WebLogic Server では、WebLogic 認可プロバイダはユーザと WebLogic リソースとの対話を制限して、整合性、機密性、および可用性を確保します。認可プロバイダは、Web アプリケーションおよび EJB デプロイメント記述子ファイル、または Administration Console、あるいはその両方を使用してコンフィグレーションできます。

WebLogic Server では、プログラムによる認可 ( このマニュアルではプログラムによるセキュリティともいう ) を使用してユーザと WebLogic リソースとの対話を制限することもできます。

# J2EE セキュリティ

ユーザ認証とユーザ認可を実装および使用するために、BEA WebLogic Server では、Java 2 Platform、Enterprise Edition (J2EE) の SDK バージョン 1.3 のセキュリティ サービスが利用されます。他の J2EE コンポーネントと同様、セキュリティ サービスも標準化されたモジュール コンポーネントに基づいています。BEA WebLogic Server は、標準に従ってこれらの Java セキュリティ サービス メソッドを実装し、細かなアプリケーションの動作をプログラミングを必要とせずに自動的に処理する拡張を追加します。

# セキュリティ API

ここでは、WebLogic Server が実装およびサポートしているセキュリティ関連のパッケージとクラスについて説明します。これらのパッケージを使用して、WebLogic Server と、クライアント アプリケーション、エンタープライズ JavaBeans (EJB)、および Web アプリケーションとの間の対話を保護します。

**注意：** WebLogic セキュリティ パッケージ、クラス、およびメソッドの中には、WebLogic Server のこのリリースでは非推奨となっているものがあります。非推奨のパッケージおよびクラスの詳細については、付録 A「非推奨のセキュリティ API」を参照してください。

この章では、以下の内容について説明します。


- 1-7 ページ「JAAS クライアント アプリケーション API」
- 1-8 ページ「SSL クライアント アプリケーション API」
- 1-9 ページ「その他の API」


# JAAS クライアント アプリケーション API

JAAS 認証を使用するクライアント アプリケーションを作成するには、Java API および WebLogic API を使用します。

この章では、以下の内容について説明します。


- 1-7 ページ「Java JAAS クライアント アプリケーション API」
- 1-8 ページ「WebLogic JAAS クライアント アプリケーション API」


## Java JAAS クライアント アプリケーション API

JAAS クライアント アプリケーションの作成には、以下の Java API を使用します。

- `javax.naming`
- `javax.security.auth`
- `javax.security.auth.Callback`
- `javax.security.auth.login`
- `javax.security.auth.SPI`

これらの API を使用する方法については、3-4 ページ「JAAS 認証 API」を参照してください。

## WebLogic JAAS クライアント アプリケーション API

JAAS クライアント アプリケーションの作成には、以下の WebLogic API を使用します。

- `weblogic.security`
- `weblogic.security.auth`
- `weblogic.security.auth.callback`

これらの API を使用する方法については、3-4 ページ「JAAS 認証 API」を参照してください。

# SSL クライアント アプリケーション API

SSL 認証を使用するクライアント アプリケーションを作成するには、Java API および WebLogic API を使用します。

この章では、以下の内容について説明します。

- 1-8 ページ「Java SSL クライアント アプリケーション API」
- 1-9 ページ「WebLogic SSL クライアント アプリケーション API」

## Java SSL クライアント アプリケーション API

SSL クライアント アプリケーションの作成には、以下の Java API を使用します。

- `java.security.cert`
- `java.security.cert`
- `javax.crypto`
- `javax.naming`
- `javax.net`
- `javax.security`
- `javax.servlet`
- `javax.servlet.http`

WebLogic Server はまた、javax.net.SSL API もサポートしていますが、WebLogic Server で SSL を使用する際には `weblogic.security.SSL` パッケージを使用することをお勧めします。

これらの API を使用する方法については、4-4 ページ「SSL 認証 API」を参照してください。

## WebLogic SSL クライアント アプリケーション API

SSL クライアント アプリケーションの作成には、以下の WebLogic API を使用します。

- `weblogic.net.http`
- `weblogic.security.SSL`

これらの API を使用する方法については、4-4 ページ「SSL 認証 API」を参照してください。

# その他の API

さらに、以下の API が WebLogic Server アプリケーションの開発に使用されます。

- `weblogic.security.net`

  この API はネットワーク接続フィルタの実装に使用されるインタフェースおよびクラスを提供します。ネットワーク接続フィルタは、ネットワーク接続を開始したクライアントの IP アドレス、ドメイン、またはプロトコルなどの属性を基に WebLogic Server への接続を許可または拒否します。この API の使い方の詳細については、6-1 ページ「ネットワーク接続フィルタの使い方」を参照してください。

- `weblogic.security.service`

  この API には、セキュリティ プロバイダをサポートするインタフェース、クラス、および例外が含まれます。WebLogic Security フレームワークは、この API が提供するインタフェース、クラス、および例外で構成されます。この API のインタフェース、クラス、および例外は、`weblogic.security.spi` パッケージのインタフェース、クラス、および例

外と関連付けて使用します。この API の使い方の詳細については、『WebLogic Security サービスの開発』を参照してください。

- `weblogic.security.services`

  この API はサーバサイドの認証クラスを提供します。このクラスは、サーバへのローカル ログインの実行に使用されます。ユーザを認証し、デフォルトのセキュリティ レルムを使用して資格を返すための CallbackHandler と共に使用されるログイン メソッドを提供します。

- `weblogic.security.spi`

  このパッケージはセキュリティ サービス プロバイダ インタフェース (SSPI) を提供します。SSPI はカスタム セキュリティ プロバイダの開発に使用されるインタフェース、クラス、および例外を提供します。多くの場合、これらのインタフェース、クラス、および例外は、`weblogic.security.service` API のインタフェース、クラス、および例外と関連付けて使用します。このパッケージのインタフェース、クラス、および例外を実装することで、セキュリティ プロバイダの実行時クラスを作成できます。この SSPI の使い方の詳細については、『WebLogic Security サービスの開発』を参照してください。

- `weblogic.servlet.security`

  この API は、サーブレット アプリケーション内からプログラムによる認証をサポートするサーバサイド API を備えています。この API の使い方の詳細については、2-44 ページ「プログラムによる認証 API の使用」を参照してください。

# Administration Console とセキュリティ

セキュリティに関して Web アプリケーション、EJB、J2EE コネクタ、およびエンタープライズ アプリケーションのデプロイメント記述子を定義および編集するには、Administration Console を使用できます。このマニュアル (『WebLogic Security プログラマーズ ガイド』) では、Administration Console を使用したセキュリティのコンフィグレーション方法については説明しません。Administration Console を使用してデプロイメント記述子を定義および編集する方法については、『WebLogic リソースのセキュリティ』および『WebLogic Security の管理』を参照してください。

# セキュリティ タスクおよびコード例

マニュアル内のセキュリティ タスクおよびコード例では、カスタム セキュリティ プロバイダではなく、WebLogic Server 配布キットの WebLogic セキュリティ プロバイダを使用することを前提にしています。カスタム セキュリティ プロバイダを使用する場合、WebLogic セキュリティ API の使い方は同じですが、カスタム セキュリティ プロバイダの管理手順が異なります。

**注意：** このマニュアルでは、WebLogic セキュリティ プロバイダまたはカスタム セキュリティ プロバイダの包括的なコンフィグレーション方法については説明していません。WebLogic セキュリティ プロバイダとカスタム セキュリティ プロバイダのコンフィグレーションの詳細については、『WebLogic Security の管理』を参照してください。

# 2 Web アプリケーションのセキュリティ対策

WebLogic Server は Web アプリケーションを保護する J2EE アーキテクチャ セキュリティ モデルをサポートします。これには、宣言による認可 ( このマニュアルでは宣言によるセキュリティともいう ) およびプログラムによる認可 ( このマニュアルではプログラムによるセキュリティともいう ) のサポートが含まれます。

ここでは以下のトピックについて説明します。


- 2-2 ページの「J2EE セキュリティ モデル」
- 2-5 ページの「Web ブラウザでの認証」
- 2-10 ページの「複数の Web アプリケーション、クッキー、および認証」
- 2-11 ページの「セキュアな Web アプリケーションの開発」
- 2-28 ページの「Web アプリケーションでの宣言によるセキュリティの使用」
- 2-28 ページの「Web アプリケーションのセキュリティ関連のデプロイメント記述子」
- 2-41 ページの「Web アプリケーションでのプログラムによるセキュリティの使用」
- 2-44 ページの「プログラムによる認証 API の使用」


**注意：** Web アプリケーションの保護には、デプロイメント記述子ファイルと Administration Console を使用できます。このマニュアルでは、デプロイメント記述子ファイルの使用方法について説明します。Administration Console を使用しての Web アプリケーションの保護については、『WebLogic リソースのセキュリティ』を参照してください。

# J2EE セキュリティ モデル

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3 Authorization」に、以下のような記述があります。

> 「J2EE アーキテクチャにおいて、コンテナはホストするコンポーネントとその呼び出し側との間の認可境界として機能します。認可境界は、コンテナの認証境界内に存在するので、認可は正常に実行される認証との関連で考慮されます。着信呼び出しの場合、コンテナは呼び出し側の資格のセキュリティ属性と、対象コンポーネントのアクセス制御ルールを比較します。ルール要件が満たされていれば、呼び出しは許可されます。それ以外の場合、この呼び出しは拒否されます。」
>
> 「アクセス制御ルールの定義には、能力とパーミッションという 2 つの基本的アプローチが存在します。能力は呼び出し側が実行できる操作、パーミッションは操作を実行できるユーザを焦点とします。J2EE アプリケーション プログラミング モデルは、パーミッションを焦点とします。J2EE アーキテクチャにおいて、デプロイヤの役割はアプリケーションのパーミッション モデルをその操作環境におけるユーザの能力にマップすることです。」

次にこのマニュアルでは、J2EE アーキテクチャ、宣言による認可、およびプログラムによる認可を使用してアプリケーション リソースへのアクセスを制御する 2 つの方法について説明します。

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html` からオンラインで入手できます。

## 宣言による認可

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3.1 Authorization」に、以下のような記述があります。

> 「デプロイヤは、J2EE アプリケーションと関連のコンテナによって実施されるアクセス制御ルールを設定します。デプロイヤは、デプロイメント ツール

を使用して、通常はアプリケーション アセンブラによって供給されるアプリケーション パーミッション モデルを、操作環境に固有のポリシーおよびメカニズムにマップします。アプリケーション パーミッション モデルは、デプロイメント記述子で定義されます。」

WebLogic Server は Web アプリケーションにおいて宣言による認可を実装するためのデプロイメント記述子の使用をサポートしています。

**注意：** 宣言による認可は、このマニュアルでは宣言によるセキュリティとも呼ばれます。

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html` からオンラインで入手できます。

# プログラムによる認可

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3.2 Programmatic Authorization」に、以下のような記述があります。

「J2EE コンテナは、コンポーネントにメソッド呼び出しをディスパッチする前に、アクセス制御の決定を行います。コンポーネントの論理または状態は、これらのアクセス決定を考慮に入れません。しかし、コンポーネントは `EJBContext.isCallerInRole` ( エンタープライズ Bean コード用 ) および `HttpServletRequest.isUserInRole` (Web コンポーネント用 ) の 2 つのメソッドを使用して、きめ細かいアクセス制御を行うことが可能です。コンポーネントはこれらのメソッドを使用して、呼び出しのパラメータ、コンポーネントの初期状態、または呼び出しの時間などその他の要因に基づきコンポーネントによって選択された特権が、呼び出し側に付与されているかどうかを判断します。」

「これらの機能の 1 つを呼び出すコンポーネントのアプリケーション コンポーネント プロバイダは、すべての呼び出しで使用されるあらゆる個別の roleName 値を宣言する必要があります。これらの宣言は、デプロイメント記述子では `security-role-ref` 要素として登場します。各 `security-role-ref` 要素は、アプリケーションに roleName として組み込まれた特権名をセキュリティ ロールにリンクします。最終的には、デプロイヤはアプリケーションに組み込まれた特権名と、デプロイメント記述子で定義

されたセキュリティ ロールの間のリンクを確立します。特権名とセキュリティ ロールの間のリンクは、同じアプリケーション内でもコンポーネントごとに異なる場合があります。」

「特定の特権をテストするだけでなく、アプリケーション コンポーネントでは `EJBContext.getCallerPrincipal` または `HttpServletRequest.getUserPrincipal` を使用して取得した呼び出し側 ID と、作成時のコンポーネントの状態に組み込まれている識別された呼び出し側 ID を比較できます。呼び出し側 ID が識別された呼び出し側のものに等しければ、コンポーネントは呼び出し側に処理の続行を許可できます。等しくなければ、コンポーネントは呼び出し側がそれ以上の対話を行わないようにできます。コンテナによって返された呼び出し側のプリンシパルは、呼び出し側が使用した認証メカニズムによって決まります。また、同じメカニズムによる同じユーザ認証に対してでも、異なったベンダからのコンテナは異なったプリンシパルを返す場合があります。プリンシパルの形式の変動性を考慮に入れるため、コンポーネントのアクセス決定に識別された呼び出し側の状態を適用することを選択した開発者は、同一ユーザを表す複数の識別された呼び出し側 ID がコンポーネントと関連付けられることを許容する必要があります。これは特に、アプリケーションの融通性や移植性が優先される場合に、推奨されます。

WebLogic Server は、Web アプリケーションにおけるプログラムによる認可を実装するための `HttpServletRequest.isUserInRole` メソッドと `HttpServletRequest.getUserPrincipal` メソッドの使用、および `security-role-ref` 要素の使用をサポートしています。

**注意：** プログラムによる認可は、このマニュアルではプログラムによるセキュリティとも呼ばれます。

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html` からオンラインで入手できます。

# 宣言による認可とプログラムによる認可

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3.3 Declarative Versus Programmatic Authorization」に、以下のような記述があります。

「デプロイヤによってコンフィグレーションされる外部アクセス制御ポリシーと、コンポーネント プロバイダによってアプリケーションに組み込まれた内部ポリシーの間にはトレードオフが存在します。アプリケーションが作成された後では、外部ポリシーのほうが高い融通性を持ちます。アプリケーションが記述されている過程では、内部ポリシーのほうが融通性の高い機能を提供します。また、外部ポリシーはデプロイヤにとって透過的で完全に理解可能であるのに対し、内部ポリシーはアプリケーションに埋め込まれており、これを完全に理解できるのはアプリケーション開発者のみである可能性があります。ある特定のコンポーネントとメソッドについて認可モデルを選択する際には、これらのトレードオフを検討する必要があります。」

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html` からオンラインで入手できます。

# Web ブラウザでの認証

Web ブラウザは、HyperText Transfer Protocol (HTTP) ポートまたは HTTP with SSL (HTTPS) ポートを介して WebLogic Server に接続できます。HTTP ポートではなく HTTPS ポートを利用するメリットは 2 つあります。HTTPS 接続を利用するメリットは以下のとおりです。

- Web ブラウザとサーバの間で行われるすべての通信が暗号化される。ユーザ名とパスワードを使用した通信がクリア テキストで行われることはありません。
- 最低限の認証要件として、サーバは身元を証明するために Web ブラウザ クライアントに対してデジタル証明書を提示する必要がある。

双方向 SSL 認証を行うようにサーバがコンフィグレーションされた場合は、サーバとクライアントの両方が互いにデジタル証明書を提示して、身元を証明する必要があります。

# ユーザ名およびパスワードの認証

WebLogic Server では、ユーザが Web ブラウザを使用して HTTP ポート経由でサーバに接続するときにユーザ名とパスワードの認証が行われます。その場合、ブラウザと WebLogic Server のインスタンスは次のように対話してユーザを認証します ( 図 2-1 を参照 )。

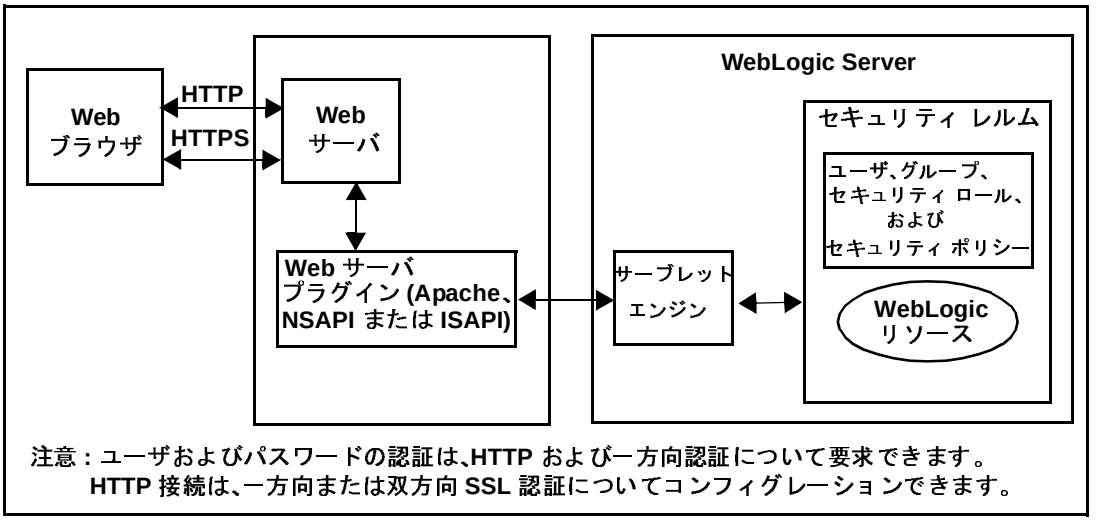
1. ユーザは、Web ブラウザでリソースの URL を入力して、WebLogic Server の WebLogic リソースを呼び出します。その URL では、たとえば `http://myserver:7001` のように HTTP および HTTP リスン ポートを指定します。
2. WebLogic Server の Web サーバがリクエストを受け取ります。

   **注意：** WebLogic Server では独自の Web サーバを用意していますが、Apache Server、Microsoft Internet Information Server、および Netscape Enterprise Server も Web サーバとして使用できます。
3. Web サーバは、要求された WebLogic リソースがセキュリティ ポリシーによって保護されているかどうかチェックします。WebLogic リソースが保護されている場合、Web サーバは確立されている HTTP 接続を使用して、ユーザのユーザ名とパスワードを要求します。
4. ユーザの Web ブラウザが Web サーバからのリクエストを受け取ると、ユーザに対してユーザ名とパスワードを要求します。
5. Web ブラウザは、Web サーバに対してユーザ名とパスワードと一緒にリクエストを再送信します。
6. Web サーバはリクエストを Web サーバ プラグインに転送します。WebLogic Server では、Web サーバ用に以下のプラグインを提供します。
   - Apache-WebLogic Server プラグイン
   - Netscape Server アプリケーション プログラミング インタフェース (NSAPI)
   - Internet Information Server アプリケーション プログラミング インタフェース (ISAPI)

   Web サーバ プラグインは、HTTP プロトコルを使用して、ユーザから受け取った認証データ ( ユーザ ID とパスワード ) と一緒に認証リクエストを WebLogic Server に送信することで認証を行います。

7. 認証が正常に行われると、WebLogic Server は処理を続行してユーザが WebLogic リソースへのアクセスを認可されているかどうかを判断します。
8. WebLogic リソースに対してメソッドを呼び出す前に、WebLogic Server インスタンスはセキュリティ認可チェックを実行します。サーバは、このチェックで、ユーザの資格をセキュリティ コンテキストから取り出し、ユーザのセキュリティ ロールを判定し、そのユーザのセキュリティ ロールを、要求された WebLogic リソースのセキュリティ ポリシーと照らし合わせて、ユーザが WebLogic リソースに対してメソッドを呼び出す権限があることを確認します。
9. 認可が成功すると、サーバがリクエストを遂行します。

**図 2-1 Web ブラウザのセキュア ログイン**




# デジタル証明書の認証

WebLogic Server は、Web ブラウザのユーザが HTTPS ポート経由でサーバに接続する場合に暗号化とデジタル証明書の認証を使用します。その場合、ブラウザと WebLogic Server インスタンスは次のように対話してユーザを認証および認可します ( 図 2-1 を参照 )。

1. ユーザは、Web ブラウザでリソースの URL を入力して、WebLogic Server の WebLogic リソースを呼び出します。その URL では、たとえば `https://myserver:7002` のように SSL リスン ポートと HTTPS スキーマを指定します。
2. WebLogic Server の Web サーバがリクエストを受け取ります。

   **注意：** WebLogic Server では独自の Web サーバを用意していますが、Apache Server、Microsoft Internet Information Server、および Netscape Enterprise Server も Web サーバとして使用できます。
3. Web サーバは、要求された WebLogic リソースがセキュリティ ポリシーによって保護されているかどうかチェックします。WebLogic リソースが保護されている場合、Web サーバは確立されている HTTPS 接続を使用して、ユーザのユーザ名とパスワードを要求します。
4. ユーザの Web ブラウザが WebLogic Server からのリクエストを受け取ると、ユーザに対してユーザ名とパスワードを要求します。( この手順は省略可能です。)
5. Web ブラウザは、ユーザ名とパスワードと一緒にリクエストを再送信します ( サーバから要求された場合のみ )。
6. WebLogic Server は、Web ブラウザにデジタル証明書を提示します。
7. Web ブラウザは、URL で使用されているサーバの名前 (`myserver` など ) がデジタル証明書の名前と一致すること、そしてデジタル証明書が信頼できる第三者 ( 信頼性のある CA) によって発行されたものであることをチェックします。
8. 双方向 SSL 認証を使用するよう設定されている場合、サーバはクライアントのデジタル証明書を要求します。

   **注意：** Web サーバ プロキシ プラグインとの完全な双方向 SSL ハンドシェークを実施するように WebLogic Server でコンフィグレーションできなくても、プロキシ プラグインは必要に応じてサーバにクライアント証明書を提供するようにコンフィグレーションできます。そのためには、WebLogic Server への HTTP ヘッダでクライアント証明書をエクスポートするようにプロキシ プラグインをコンフィグレーションします。WebLogic Server にクライアント証明書をエクスポートするようにプロキシ プラグインをコンフィグレーションする方法については、『WebLogic Server における Web サーバ プラグインの使い方』の特定プラグインのコンフィグレーション情報を参照してください。

9. Web サーバは、リクエストを Web サーバ プラグインに転送します。セキュアなプロキシが設定されていれば (HTTPS プロトコルが使用されている場合には設定されている )、Web サーバ プラグインはまた、HTTPS プロトコルを介して、ユーザから受け取った認証データ ( ユーザ名とパスワード ) と一緒にリクエストを WebLogic Server の WebLogic リソースに送信することによって認証を実行します。

   **注意：** 双方向 SSL 認証を使用していれば、クライアントの証明書に基づいて ID アサーションを行うようにサーバをコンフィグレーションすることもできます。この場合、ユーザ名とパスワードを供給するのではなく、サーバはクライアントの証明書からユーザ名とパスワードを取り出します。

10. 認証が正常に行われると、WebLogic Server は処理を続行してユーザが WebLogic リソースへのアクセスを認可されているかどうかを判定します。

11. WebLogic リソースに対してメソッドを呼び出す前に、サーバはセキュリティ認可チェックを実行します。サーバは、このチェックで、ユーザの資格をセキュリティ コンテキストから取り出し、ユーザのセキュリティ ロールを判定し、そのユーザのセキュリティ ロールを、要求された WebLogic リソースのセキュリティ ポリシーと照らし合わせて、ユーザが WebLogic リソースに対してメソッドを呼び出す権限があることを確認します。

12. 認可が成功すると、サーバがリクエストを遂行します。

詳細については、以下のマニュアルを参照してください。

- 『WebLogic Security の管理』
- 「Apache HTTP Server プラグインのインストールとコンフィグレーション」
- 「Microsoft Internet Information Server (IIS) プラグインのインストールとコンフィグレーション」
- 「Netscape Enterprise Server (NES) プラグインのインストールとコンフィグレーション」

# 複数の Web アプリケーション、クッキー、および認証

デフォルトでは、WebLogic Server はすべての Web アプリケーションに同じクッキー名 (JSESSIONID) を割り当てます。どの種類の認証を使用する場合でも、同じクッキー名を使用する Web アプリケーションでは、認証用にシングル サインオンを使用します。ユーザが認証されると、その認証は、同じクッキー名を使用するすべての Web アプリケーションへのリクエストに対して有効になります。ユーザは再び認証を要求されることはありません。

Web アプリケーションごとに個別の認証が必要な場合は、Web アプリケーションにユニークなクッキー名またはクッキー パスを指定できます。CookieName パラメータでクッキー名を指定し、CookiePath パラメータでクッキー パスを指定します。これらのパラメータは、<session-descriptor> 要素の WebLogic 固有のデプロイメント記述子 weblogic.xml で定義されています。詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』の「session-descriptor 要素」を参照してください。

クッキー名を保持しつつ Web アプリケーションごとに別々の認証が必要な場合は、Web アプリケーションごとにクッキー パラメータ (CookiePath) を変えることができます。

サービス パック 3 では、セッション データを失うことなく、HTTP を使用して開始されたセッションで HTTPS リソースにユーザが安全にアクセスできるようにする新機能が追加されました。この新機能を有効にするには、config.xml の WebServer 要素に AuthCookieEnabled="true" を追加します。

```
<WebServer Name="myserver" AuthCookieEnabled="true"/>
```

AuthCookieEnabled を true に設定すると、HTTPS 接続を介して認証するときに、WebLogic Server インスタンスはブラウザに新しいセキュアなクッキーを送信します。一度セキュアなクッキーを設定すると、セッションはクッキーがブラウザから送信された場合しかセキュリティ制約のある他の HTTPS リソースにアクセスできません。

**注意：** 普通の HTTP で認証する場合、HTTPS リソースでセキュアなクッキーは設定されず、必要ともされません。保護されていない HTTPS リソースにアクセスする場合、クッキーは検証されません ( ブラウザから送信されないため )。このため、ブラウザはユーザのログインなしで保護されていない HTTPs リソースにアクセスできます。

# セキュアな Web アプリケーションの開発

WebLogic Server は、Web ブラウザに関して以下の 3 タイプの認証をサポートしています。

- BASIC
- FORM
- CLIENT-CERT

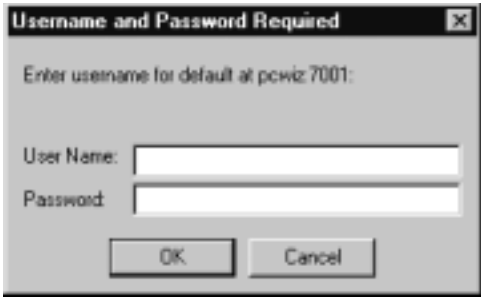
以降の節では、次のトピックについて説明します。


- 2-11 ページの「BASIC 認証 Web アプリケーションの開発」
- 2-17 ページの「FORM 認証 Web アプリケーションの開発」
- 2-24 ページの「ID アサーションを使用した Web アプリケーションの認証」
- 2-24 ページの「双方向 SSL を使用した Web アプリケーションの認証」
- 2-26 ページの「Swing ベース認証 Web アプリケーションの開発」
- 2-26 ページの「Web アプリケーションのデプロイメント」


# BASIC 認証 Web アプリケーションの開発

BASIC 認証の場合、Web ブラウザは WebLogic リソースの要求に応じてログイン画面を表示します。そのログイン画面は、ユーザ名とパスワードの入力をユーザに要求します。図 2-2 は典型的なログイン画面です。

**図 2-2　BASIC 認証のログイン画面**




BASIC 認証を提供する Web アプリケーションを開発するには、次の手順を行います。

1. `web.xml` デプロイメント記述子を作成します。このファイルでは、以下の情報を定義します ( コード リスト 2-1 を参照 )。

   a. ウエルカム ファイルを定義します。ウエルカム ファイルの名前は `welcome.jsp` です。

   b. 保護する予定の Web アプリケーション リソース、つまり URL リソースの各セットについてセキュリティ制約を定義します。リソースの各セットは共通の URL を共有します。HTML ページ、JSP、サーブレットなどは最も一般的に保護される URL リソースですが、他のタイプの URL リソースもサポートされています。コード リスト 2-1 では、URL パターンは Web アプリケーションの最上位ディレクトリに位置する `welcome.jsp` ファイルを指しており、URL リソースへのアクセスが許可される HTTP メソッドは POST と GET、セキュリティ ロールは `webuser` (`<auth-constraint>` で指定 ) となっています。

   **注意：** <auth-constraint> タグは、正確に理解し使用してください。そうしないと、必要な保護をアーカイブできません。<auth-constraint> タグの詳細については、2-29 ページの「auth-constraint」を参照してください。セキュリティ ロール名を指定する場合、以下の規約と制限に従ってください。

   - セキュリティ ロール名の適切な構文は、Web 上 (`http://www.w3.org/TR/REC-xml#NT-Nmtoken`) で閲覧可能な XML (Extensible Markup Language) 勧告で `Nmtoken` に関して定義されているとおりです。

- スペース、カンマ、ハイフン、\t、< >、#、|、&、~、?、( )、{ } を使用しないでください。

- セキュリティ ロール名では大文字 / 小文字を区別します。

- BEA が提唱する命名規約では、セキュリティ ロール名は単数形です。

c. 使用する認証のタイプとセキュリティ制約が適用されるセキュリティ レルムの定義には <login-config> を使用します。コード リスト 2-1 では BASIC タイプが指定されており、レルムはデフォルトのレルムとなっています。つまり、セキュリティ制約は WebLogic Server インスタンスの起動時にアクティブなセキュリティ レルムに適用されます。

**注意：** WebLogic Server のこのリリースでは、<login-config> タグおよび <realm-name> サブタグで定義されたレルム名は無視されます。

d. 1 つまたは複数のセキュリティ ロールを定義し、それらをセキュリティ制約にマップします。サンプルでは、セキュリティ制約で定義されているセキュリティ ロールは webuser 1 つだけなので、定義されているセキュリティ ロール名は 1 つだけです ( コード リスト 2-1 の <security-role> タグを参照 )。ただし、セキュリティ ロールは必要なだけ定義できます。

**コード リスト 2-1 BASIC 認証の web.xml ファイル**

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

    <web-app>
          <welcome-file-list>
               <welcome-file>welcome.jsp</welcome-file>
          </welcome-file-list>

          <security-constraint>
                <web-resource-collection>
                    <web-resource-name>Success</web-resource-name>
                      <url-pattern>/welcome.jsp</url-pattern>
                     <http-method>GET</http-method>
                     <http-method>POST</http-method>
                </web-resource-collection>
                <auth-constraint>
                    <role-name>webuser</role-name>
                </auth-constraint>
          </security-constraint>

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

```
            <realm-name>default</realm-name>
        </login-config>

        <security-role>
            <role-name>webuser</role-name>
        </security-role>
    </web-app>
```

2. weblogic.xml デプロイメント記述子を作成します。このファイルでは、セキュリティ ロール名をユーザおよびグループにマップします。コード リスト 2-2 は、対となる web.xml ファイルの <security-role> タグで定義されている webuser セキュリティ ロールを myGroup という名前のグループにマップするサンプルの weblogic.xml ファイルです。プリンシパルは、ユーザにもグループにもできるため、<principal-tag> はどちらにも使用できます。このコンフィグレーションでは、WebLogic Server は myGroup のユーザだけに保護されている URL リソース welcome.jsp へアクセスすることを許可します。ただし、Administration Console を使用すると、他のグループも保護リソースへのアクセスが許可されるように Web アプリケーションのセキュリティ ロールを修正できます。

   **注意：** weblogic.xml デプロイメント記述子の作成は省略可能です。このファイルを含めなかった場合、またはファイルを含めたがすべてのセキュリティ ロールのマッピングは含めなかった場合、マップされていないセキュリティ ロールはすべてデフォルトで、ロール名に一致する名前を持つユーザまたはグループに設定されます。たとえば、セキュリティ ロールに「SampleTester」という名前を付けると、「SampleTester」という名前を持つユーザまたはグループがそのセキュリティ ロールに含まれます。

**コード リスト 2-2 BASIC 認証の weblogic.xml ファイル**

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 7.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic700-web-jar.dtd">

<weblogic-web-app>

    <security-role-assignment>
        <role-name>webuser</role-name>
        <principal-name>myGroup</principal-name>
    </security-role-assignment>

</weblogic-web-app>
```

3. ユーザがユーザ名とパスワードを入力してアクセス権を付与されたときに表示されるウエルカム画面を生成するファイルを作成します。コード リスト 2-3 は、サンプルの welcome.jsp ファイルを示します。図 2-3 は、ウエルカム画面を示します。

**コード リスト 2-3 BASIC 認証の welcome.jsp ファイル**

```
<html>
  <head>
   <title>Browser Based Authentication Example Welcome Page</title>
  </head>
  <h1> Browser Based Authentication Example Welcome Page </h1>

  <p> Welcome <%= request.getRemoteUser() %>!

  </blockquote>
  </body>
</html>
```

**注意：** コード リスト 2-3 において、JSP がログインしたユーザの名前を取得するために API (`request.getRemoteUser()`) を呼び出していることに留意してください。代わりに、別の API `weblogic.security.Security.getCurrentSubject()` を使用することもできます。

**図 2-3　ウエルカム画面**


4. WebLogic Server を起動し、URL リソースにアクセス可能なユーザおよびグループを定義します。`weblogic.xml` ファイル ( コード リスト 2-2) では、`<principal-name>` タグで、`welcome.jsp` にアクセス可能なグループとして `myGroup` が定義されています。したがって、Administration Console を使用して `myGroup` グループを定義し、ユーザを定義して、そのユーザを `myGroup` グループに追加します。ユーザおよびグループの追加については、『WebLogic リソースのセキュリティ』の「ユーザとグループ」を参照してください。
5. Web アプリケーションをデプロイし、前の手順で定義したユーザを使用して保護 URL リソースにアクセスします。
   a. デプロイメントの手順については、2-26 ページの「Web アプリケーションのデプロイメント」を参照してください。
   b. Web ブラウザを開き、次の URL を入力します。

      ```
      http://localhost:7001/basicauth/welcome.jsp
      ```

   c. ユーザ名とパスワードを入力します。ウエルカム画面が表示されます。

# FORM 認証 Web アプリケーションの開発

Web アプリケーションで FORM 認証を使用する場合は、Web アプリケーション リソースの要求に応じて Web ブラウザが表示するカスタム ログイン画面とログインが失敗した場合に表示されるエラー画面を用意します。ログイン画面は HTML ページ、JSP またはサーブレットを使用して生成できます。フォームベースのログインのメリットは、これらの画面を完全に管理できるので、アプリケーションまたは会社の方針 / ガイドラインの要件にあわせてそれらを設計できることです。

そのログイン画面は、ユーザ名とパスワードの入力をユーザに要求します。図 2-4 は、JSP を使用して生成される典型的なログイン画面を、コード リスト 2-4 はソース コードを示します。

**図 2-4 フォームベースのログイン画面 (login.jsp)**


**コード リスト 2-4 フォームベースのログイン画面のソース コード (login.jsp)**

```
<html>
  <head>)
    <title>Security WebApp login page</title>
  </head>
  <body bgcolor="#cccccc">
```

```
  <blockquote>
  <img src=BEA_Button_Final_web.gif align=right>
  <h2>Please enter your user name and password:</h2>
  <p>
  <form method="POST" action="j_security_check">
  <table border=1>
    <tr>
      <td>Username:</td>
      <td><input type="text" name="j_username"></td>
    </tr>
    <tr>
      <td>Password:</td>
      <td><input type="password" name="j_password"></td>
    </tr>
    <tr>
      <td colspan=2 align=right><input type=submit
                                       value="Submit"></td>
    </tr>
  </table>
  </form>
  </blockquote>
  </body>
</html>
```

図 2-5 は、HTML を使用して生成される典型的なログイン エラー画面を、コード リスト 2-5 はソース コードを示します。

**図 2-5　ログイン エラー画面**

**コード リスト 2-5 ログイン エラー画面ソース コード**

```
<html>
  <head>
    <title>Login failed</title>
  </head>
  <body bgcolor=#ffffff>
  <blockquote>
  <img src=/security/BEA_Button_Final_web.gif align=right>
  <h2>Sorry, your user name and password were not recognized.</h2>
  <p><b>
  <a href="/security/welcome.jsp">Return to welcome page</a> or
           <a href="/security/logout.jsp">logout</a>
  </b>
  </blockquote>
  </body>
</html>
```

FORM 認証を提供する Web アプリケーションを開発するには、次の手順を行います。

1. `web.xml` デプロイメント記述子を作成します。このファイルでは、以下の情報を定義します ( コード リスト 2-6 を参照 )。

   a. ウエルカム ファイルを定義します。ウエルカム ファイルの名前は `welcome.jsp` です。

   b. 保護する予定の Web アプリケーション リソース、つまり URL リソースの各セットについてセキュリティ制約を定義します。URL リソースの各セットは共通の URL を共有します。HTML ページ、JSP、サーブレットなどは最も一般的に保護される URL リソースですが、他のタイプの URL リソースもサポートされています。コード リスト 2-6 では、URL パターンは `/admin/edit.jsp` を指しており ( したがって、Web アプリケーションの `admin` サブディレクトリに配置された `edit.jsp` ファイルが保護される )、URL リソースへのアクセスが許可される HTTP メソッド (`GET`) が定義され、セキュリティ ロール名 `admin` (`<auth-constraint>` タグで指定 ) が定義されています。

   **注意：** <auth-constraint> タグは、正確に理解し使用してください。そうしないと、必要な保護をアーカイブできません。<auth-constraint> タグの詳細については、2-29 ページの「auth-constraint」を参照してください。セキュリティ ロール名を指定する場合、以下の規約と制限に従ってください。

- セキュリティ ロール名の適切な構文は、Web 上 (http://www.w3.org/TR/REC-xml#NT-Nmtoken) で閲覧可能な XML (Extensible Markup Language) 勧告で Nmtoken に関して定義されているとおりです。

- スペース、カンマ、ハイフン、\t、< >、#、|、&、~、?、( )、{ } を使用しないでください。

- セキュリティ ロール名では大文字 / 小文字を区別します。

- BEA が提唱する命名規約では、セキュリティ ロール名は単数形です。

c. 使用する認証のタイプとセキュリティ制約が適用されるセキュリティ レルムを定義します。この場合は、FORM タイプが指定されており、レルムは指定されていないのでデフォルトのレルムになります。つまり、セキュリティ制約は WebLogic Server インスタンスの起動時にアクティブなセキュリティ レルムに適用されます。

d. 1 つまたは複数のセキュリティ ロールを定義し、それらをセキュリティ制約にマップします。サンプルでは、1 つのセキュリティ ロール admin のみがセキュリティ制約で定義されているので、ここでは 1 つのセキュリティ ロール名のみ定義されています。ただし、セキュリティ ロールは必要なだけ定義できます。

**コード リスト 2-6　FORM 認証の web.xml ファイル**

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <welcome-file-list>
        <welcome-file>welcome.jsp</welcome-file>
    </welcome-file-list>

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>AdminPages</web-resource-name>
            <description>
                These pages are only accessible by authorized
                 administrators.
            </description>
            <url-pattern>/admin/edit.jsp</url-pattern>
            <http-method>GET</http-method>
        </web-resource-collection>
        <auth-constraint>
```

```
            <description>
                These are the roles who have access.
            </description>
            <role-name>
                admin
            </role-name>
        </auth-constraint>
        <user-data-constraint>
            <description>
                This is how the user data must be transmitted.
            </description>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>FORM</auth-method>
        <form-login-config>
            <form-login-page>/login.jsp</form-login-page>
            <form-error-page>/fail_login.html</form-error-page>
        </form-login-config>
    </login-config>

    <security-role>
        <description>
            An administrator
        </description>
        <role-name>
            admin
        </role-name>
    </security-role>
</web-app>
```

2. weblogic.xml デプロイメント記述子を作成します。このファイルでは、セキュリティ ロール名をユーザおよびグループにマップします。コード リスト 2-7 は、対となる web.xml ファイルの <security-role> タグで定義されている admin セキュリティ ロールをグループ supportGroup にマッピングするサンプルの weblogic.xml ファイルです。このコンフィグレーションの場合、WebLogic Server は supportGroup グループのユーザのみに保護 WebLogic リソースへのアクセスを許可します。ただし、Administration Console を使用すると、他のグループも保護 WebLogic リソースへのアクセスが許可されるように Web アプリケーションのセキュリティ ロールを修正できます。

**コード リスト 2-7 FORM 認証の weblogic.xml ファイル**

```
<!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web
Application 7.0//EN"
"http://www.bea.com/servers/wls700/dtd/weblogic700-web-jar.dtd">

<weblogic-web-app>

     <security-role-assignment>
         <role-name>admin</role-name>
         <principal-name>supportGroup</principal-name>
     </security-role-assignment>

</weblogic-web-app>
```

3. ユーザが URL を入力して保護 Web アプリケーション リソースを要求したときにウエルカム画面を生成する Web アプリケーション ファイルを作成します。コード リスト 2-8 はサンプルの welcome.jsp ファイルです。図 2-3 はウエルカム画面です。

**コード リスト 2-8 FORM 認証の welcome.jsp ファイル**

```
<html>
  <head>
    <title>Security login example</title>
  </head>

  <%
    String bgcolor;
   if ((bgcolor=(String)application.getAttribute("Background")) ==
       null)
    {
       bgcolor="#cccccc";
    }
  %>

  <body bgcolor=<%="\""+bgcolor+"\""%>>

  <blockquote>
  <img src=BEA_Button_Final_web.gif align=right>
  <h1> Security Login Example </h1>

  <p> Welcome <%= request.getRemoteUser() %>!

  <p> If you are an administrator, you can configure the background
  color of the Web Application.
  <br> <b><a href="admin/edit.jsp">Configure background</a></b>.
```

```
  <% if (request.getRemoteUser() != null) { %>
    <p> Click here to <a href="logout.jsp">logout</a>.
  <% } %>

  </blockquote>
  </body>
</html>
```

**注意：** コード リスト 2-8 において、JSP がログインしたユーザの名前を取得するために API (`request.getRemoteUser()`) を呼び出していることに留意してください。代わりに、別の API `weblogic.security.Security.getCurrentSubject()` を使用することもできます。

4. WebLogic Server を起動し、URL リソースにアクセス可能なユーザおよびグループを定義します。`weblogic.xml` ファイル ( コード リスト 2-7) では、`<role-name>` タグで `admin` が `edit.jsp` ファイルにアクセス可能なグループとして定義され、ユーザ `joe` がそのグループのメンバーとして定義されています。したがって、Administration Console を使用して `admin` グループを定義し、ユーザ `joe` を定義して、`joe` を `admin` グループに追加します。他のユーザを定義してグループに追加することもでき、その追加ユーザも保護 WebLogic リソースにアクセスすることができます。ユーザおよびグループの追加については、『WebLogic リソースのセキュリティ』の「ユーザとグループ」を参照してください。
5. Web アプリケーションをデプロイし、前の手順で定義したユーザを使用して保護 Web アプリケーション リソースにアクセスします。
   a. デプロイメントの手順については、2-26 ページの「Web アプリケーションのデプロイメント」を参照してください。
   b. Web ブラウザを開き、次の URL を入力します。

      `http://`*`hostname`*`:7001/security/welcome.jsp`

   c. ユーザ名とパスワードを入力します。ウエルカム画面が表示されます。

# ID アサーションを使用した Web アプリケーションの認証

Web アプリケーションで ID アサーションを使用すると、認証を目的にクライアントの ID を検証できます。ID アサーションを使用するときには、以下の要件を満たす必要があります。

1. 認証タイプを CLIENT-CERT に設定する必要があります。
2. サーバに ID アサーション プロバイダがコンフィグレーションされている必要があります。Web ブラウザまたは Java クライアントがセキュリティ ポリシーで保護された WebLogic Server リソースを要求する場合、WebLogic Server は Web ブラウザまたは Java クライアントが ID を持つことを要求します。WebLogic ID アサーション プロバイダは、Web ブラウザまたは Java クライアントからのトークンを WebLogic Server セキュリティ レルムのユーザに対応付けます。ID アサーション プロバイダのコンフィグレーション方法については、「WebLogic ID アサーション プロバイダのコンフィグレーション」を参照してください。
3. トークンの値に対応するユーザは、サーバのセキュリティ レルムで定義されている必要があります。定義されていないと、クライアントは保護されている WebLogic リソースにアクセスできません。サーバ上のユーザの詳細については、『WebLogic リソースのセキュリティ』の「ユーザの作成」を参照してください。

# 双方向 SSL を使用した Web アプリケーションの認証

Web アプリケーションで双方向 SSL を使用すると、クライアントがその主張どおりの存在であることを検証できます。双方向 SSL を使用するときには、以下の要件を満たす必要があります。

1. 認証タイプを CLIENT-CERT に設定する必要があります。

2. サーバを双方向 SSL 向けにコンフィグレーションする必要があります。SSL とデジタル証明書の使用については、4-1 ページの「Java クライアントでの SSL 認証の使用」を参照してください。サーバの SSL コンフィグレーションの詳細については、『WebLogic Security の管理』の「SSL のコンフィグレーション」を参照してください。
3. クライアントは、サーバの Web アプリケーションにアクセスするために HTTPS を使用する必要があります。
4. サーバに ID アサーション プロバイダがコンフィグレーションされている必要があります。Web ブラウザまたは Java クライアントがセキュリティ ポリシーで保護された WebLogic Server リソースを要求する場合、WebLogic Server は Web ブラウザまたは Java クライアントが ID を持つことを要求します。WebLogic ID アサーション プロバイダを使用すると、Web ブラウザまたは Java クライアントのデジタル証明書を WebLogic Server セキュリティ レルム内のユーザにマップするユーザ名マッパーをサーバで使用できます。ID アサーション プロバイダとユーザ名マッパーの使い方については、『WebLogic Security の管理』の「WebLogic ID アサーション プロバイダのコンフィグレーション」および「WebLogic ID アサーション プロバイダでのユーザ名マッパーの使用」を参照してください。
5. クライアントのデジタル証明書の Subject's Distinguished Name (SubjectDN) 属性に対応するユーザは、サーバのセキュリティ レルムで定義されている必要があります。定義されていないと、クライアントには保護された WebLogic リソースへのアクセスが許可されません。サーバ上のユーザの詳細については、『WebLogic リソースのセキュリティ』の「ユーザの作成」を参照してください。

**注意：** SSL 認証を使用する場合、サーバの SSL コンフィグレーションを Administration Console で指定するので、`web.xml` および `weblogic.xml` ファイルを使用してサーバのコンフィグレーションを指定する必要はありません。サーバの SSL コンフィグレーションの詳細については、『WebLogic Security の管理』の「SSL のコンフィグレーション」を参照してください。

# Swing ベース認証 Web アプリケーションの開発

Web ブラウザでは、Swing コンポーネントを使用して開発されたグラフィカル ユーザ インタフェース (GUI) を操作することもできます。Java Foundation Classes (JFC) の一部である Swing コンポーネントは、JDK 1.1 または Java 2 プラットフォームで使用できます。

Swing コンポーネントを使用してアプリケーションおよびアプレットのグラフィカル ユーザ インタフェース (GUI) を作成する方法については、Sun Microsystems, Inc. 作成の「*Creating a GUI with JFC/Swing*」チュートリアル (Swing チュートリアルとも呼ばれる ) を参照してください。このチュートリアルは、`http://java.sun.com/docs/books/tutorial/uiswing/` でアクセスできます。

Swing ベースの GUI を開発したら、2-17 ページの「FORM 認証 Web アプリケーションの開発」を参照し、Swing ベースの画面を使用して、FORM 認証を提供する Web アプリケーションの開発に必要な手順を実行します。

**注意 :** Swing ベースの GUI を開発する場合、swing イベント スレッドの子スレッドに Java 仮想マシン全体のユーザを使用しないでください。これは J2EE に準拠していないので、通常はシン クライアントや IIOP では動作しません。代わりに、以下のいずれかの方法を用います。

- Swing アーティファクトの前に `InitialContext` を作成します。
- または、Java Authentication and Authorization Service (JAAS) を使用してログインしてから、Swing イベント スレッドとその子で `Security.runAs()` メソッドを使用します。

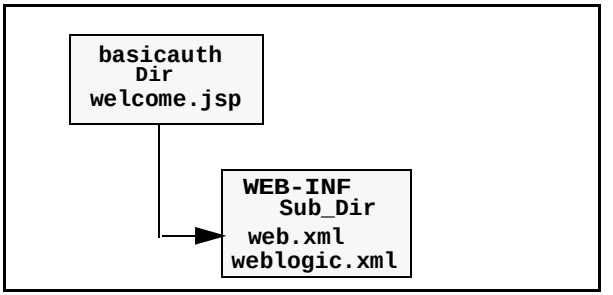
# Web アプリケーションのデプロイメント

開発モードで動作しているサーバに Web アプリケーションをデプロイするには、次の手順を行います。

**注意 :** 開発モードまたはプロダクション モードでの Web アプリケーションのデプロイの詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』の「Web アプリケーションのデプロイメント」を参照してください。

1. Web アプリケーションのファイルのディレクトリ構造を構築します。図 2-6 は、basicauth という Web アプリケーションのディレクトリ構造です。最上位ディレクトリには Web アプリケーションの名前を割り当て、サブディレクトリは WEB-INF という名前にする必要があります。

**図 2-6 Basicauth Web アプリケーションのディレクトリ構造**




2. Java アーカイブ (jar) 形式ではなく、展開ディレクトリ形式の Web アプリケーションをデプロイするには、ただ単純にディレクトリをサーバ上の applications ディレクトリに移動します。たとえば、basicauth Web アプリケーションは次の位置にデプロイします。

   *WL_HOME*\user_projects\mydomain\applications\basicauth

   WebLogic Server インスタンスが動作している場合、アプリケーションは自動デプロイされるはずです。Administration Console を使用すると、アプリケーションがデプロイされたことを確認できます。

   WebLogic Server インスタンスが動作していない場合は、サーバを起動すると Web アプリケーションは自動デプロイされます。

3. まだしていない場合は、Administration Console を使用して、Web アプリケーションにアクセスできるユーザおよびグループをコンフィグレーションします。保護 WebLogic リソースへのアクセスを許可されるユーザおよびグループを判別するには、weblogic.xml ファイルを調べます。たとえば、basicauth サンプルの weblogic.xml ファイル ( コード リスト 2-2) では、myGroup が welcome.jsp ファイルにアクセスできる唯一のグループとして定義されています。

セキュアな Web アプリケーションのデプロイの詳細については、『Web アプリケーションのアセンブルとコンフィグレーション』の「Web アプリケーションのデプロイメント」を参照してください。

# Web アプリケーションでの宣言によるセキュリティの使用

Web アプリケーションに宣言によってセキュリティを実装するには、デプロイメント記述子 (`web.xml` および `weblogic.xml`) を使用してセキュリティ要件を定義します。デプロイメント記述子は、アプリケーションの論理的なセキュリティ要件を実行時の定義にマップします。また、実行時には、サーブレット コンテナがセキュリティ定義を使って、要件を実施します。

デプロイメント記述子を使用した簡単な Web アプリケーションでのセキュリティのコンフィグレーションについては、2-11 ページの「セキュアな Web アプリケーションの開発」を参照してください。

セキュリティ関連のデプロイメント記述子については、2-28 ページの「Web アプリケーションのセキュリティ関連のデプロイメント記述子」を参照してください。

Administration Console を使用して Web アプリケーションのセキュリティをコンフィグレーションする方法については、『WebLogic リソースのセキュリティ』を参照してください。

# Web アプリケーションのセキュリティ関連のデプロイメント記述子

以下のトピックでは、Web アプリケーションのセキュリティ要件を定義するために `web.xml` および `weblogic.xml` ファイルで使用されるデプロイメント記述子の要素について説明します。


- 2-29 ページの「web.xml デプロイメント記述子」
- 2-36 ページの「weblogic.xml デプロイメント記述子」

# web.xml デプロイメント記述子

以下の web.xml のセキュリティ関連のデプロイメント記述子の要素は、WebLogic Server でセキュリティ要件を定義するために使用されます。


- 2-29 ページの「auth-constraint」
- 2-32 ページの「security-constraint」
- 2-33 ページの「security-role」
- 2-34 ページの「security-role-ref」
- 2-34 ページの「user-data-constraint」
- 2-35 ページの「web-resource-collection」


この節の情報は、Sun Microsystems, Inc. 提供の web.xml の文書型記述子 (DTD) に基づいています。web.xml の DTD は、http://java.sun.com/dtd/web-app_2_3.dtd にあります。

## auth-constraint

省略可能な auth-constraint 要素では、このセキュリティ制約で定義された Web リソースの集合にアクセスするグループまたはプリンシパルを定義します。

**注意：** 認可制約 (<auth-constraint> タグで定義 ) は、認証の要件を確立し、制約されたリクエストの実行を許可された認証ロール ( セキュリティ ロール ) を指定します。<auth-constraint> タグを使用して認可制約を定義する場合は、以下の点に注意してください。

- セキュリティ ロールを指定しない認可制約を定義した場合、コンテナは制約されたリクエストへのアクセスを絶対に許可しない。.
- リクエストに認可制約が適用されない場合、コンテナはユーザ認証なしにリクエストを受け入れなければならない。

以下の例は、<auth-constraint> タグの使い方を示しています。


- 2-30 ページの「例 1 : <auth-constraint> タグなしで <security-constraint> タグを使用する」
- 2-31 ページの「例 2 : ロールを指定せずに <auth-constraint> タグを使用する」

■ 2-31 ページの「例 3 : ロールを指定して <auth-constraint> タグを使用する」

<auth-constraint> タグの詳しい使い方については、http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html の Java サーブレット仕様バージョン 2.4 を参照してください。

次の表では、auth-constraint 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <description> | 省略可能 | このセキュリティ制約の説明文。 |
| <role-name> | 省略可能 | このセキュリティ制約で定義されたリソースにアクセスできるセキュリティ ロールを定義する。セキュリティ ロール名は、security-role-ref 要素を使用してプリンシパルにマップされる。2-34 ページの「security-role-ref」を参照。 |

## 使用する場所

auth-constraint 要素は、security-constraint 要素内で使用されます。

## 例 1 : <auth-constraint> タグなしで <security-constraint> タグを使用する

コード リスト 2-9 は、検証および保護されないリソースを示しています。この場合、<auth-constraint> タグが使用されないので、リソースは検証および保護されません。このタイプの保護はすべてが保護される ("/*") 場合は便利ですが、静的なファイルまたは画像への自由なアクセスを許可することも必要です。

**コード リスト 2-9 ケース 1 : 検証および保護されないリソース**

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>Unchecked-Resource</web-resource-name>
      <url-pattern>/foo/*</url-pattern>
    </web-resource-collection>
</security-constraint>
```

## 例 2 : ロールを指定せずに <auth-constraint> タグを使用する

コード リスト 2-10 では、ロールを指定しない <auth-constraint> タグが使用されています ( したがって、リソースには誰もアクセスできない )。このように <auth-constraint> タグを使用するのは、リソースへのダイレクトなアクセスを避ける場合には便利ですが、forwards、includes、servletContext.getResource()、getResourceAsStream() によるアクセスは許可する必要があります。

**注意 :** このように <auth-constraint> タグを使用する代替手段として、リソースを WEB-INF ディレクトリに配置することもできます。このディレクトリに配置されたリソースへのダイレクトなアクセス リクエストを WebLogic Server は許可しません。

**コード リスト 2-10 ロールが指定されない <auth-constraint> タグ**

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Excluded-Resource</web-resource-name>
    <url-pattern>/foo/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
  </auth-constraint>
</security-constraint>
```

## 例 3 : ロールを指定して <auth-constraint> タグを使用する

コード リスト 2-11 では、ロールを指定して <auth-constraint> タグが使用されています。この場合は、リソースが検証および保護され、adminrole ロールのユーザのみアクセスできるようになります。

**コード リスト 2-11 ロールが指定された <auth-constraint> タグ**

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Checked-Resource</web-resource-name>
    <url-pattern>/blah/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>adminrole</role-name>
```

```
    </auth-constraint>
  </security-constraint>
```

## security-constraint

security-constraint 要素は、web-resource-collection 要素で定義されたリソースの集合へのアクセス特権を定義するために web.xml ファイルで使用されます。

次の表では、security-constraint 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <web-resource-collection> | 必須 | このセキュリティ制約が適用される Web アプリケーションのコンポーネントを定義する。詳細については、2-35 ページの「web-resource-collection」を参照。 |
| <auth-constraint> | 省略可能 | このセキュリティ制約で定義される Web リソースの集合にアクセスするグループまたはプリンシパルを定義する。詳細については、2-29 ページの「auth-constraint」を参照。 |
| <user-data-constraint> | 省略可能 | クライアントとサーバ間でやり取りされるデータの保護方法を定義する。詳細については、2-34 ページの「user-data-constraint」を参照。 |

### 例

コード リスト 2-12 は、web.xml ファイルで security-constraint 要素を使用して SecureOrdersEast のセキュリティを定義する方法を示しています。

**コード リスト 2-12 セキュリティ制約の例**

```
web.xml entries:
<security-constraint>
     <web-resource-collection>
          <web-resource-name>SecureOrdersEast</web-resource-name>
          <description>
             Security constraint for
```

```
            resources in the orders/east directory
          </description>
          <url-pattern>/orders/east/*</url-pattern>
          <http-method>POST</http-method>
          <http-method>GET</http-method>
     </web-resource-collection>
     <auth-constraint>
          <description>
           constraint for east coast sales
          </description>
          <role-name>east</role-name>
          <role-name>manager</role-name>
     </auth-constraint>
 <user-data-constraint>
          <description>SSL not required</description>
          <transport-guarantee>NONE</transport-guarantee>
     </user-data-constraint>
</security-constraint>
...
```

## security-role

security-role 要素には、セキュリティ ロールの定義が指定されます。定義は、セキュリティ ロールの説明 ( 省略可能 ) とセキュリティ ロール名から成ります。

次の表では、security-role 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <description> | 省略可能 | セキュリティ ロールの説明文。 |
| <role-name> | 必須 | ロール名。ここで使用する名前は、WebLogic 固有のデプロイメント記述子 weblogic.xml で対応するエントリが必要になる。weblogic.xml によって、ロールはセキュリティ レルム内のプリンシパルにマップされる。詳細については、2-40 ページの「security-role-assignment」を参照。 |

### 例

web.xml ファイルでの security-role 要素の使用例については、コード リスト 2-1 を参照してください。

## security-role-ref

security-role-ref 要素は、<security-role> で定義されたセキュリティ ロール名を、サーブレットのロジックでハード コード化される代替ロール名にリンクします。この特別な抽象化レイヤによって、サーブレット コードを変更しなくてもデプロイメント時にサーブレットをコンフィグレーションできるようになります。

次の表では、security-role-ref 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <description> | 省略可能 | ロールの説明文。 |
| <role-name> | 必須 | サーブレット コード内で使用されるセキュリティ ロールまたはプリンシパルの名前を定義する。 |
| <role-link> | 必須 | 後にデプロイメント記述子内の <security-role> 要素で定義されるセキュリティ ロールの名前を定義する。 |

### 例

web.xml ファイルでの security-role-ref 要素の使用例については、コード リスト 2-17 を参照してください。

## user-data-constraint

user-data-constraint 要素は、クライアントとサーバ間でやり取りされるデータの保護方法を定義します。

次の表では、user-data-constraint 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <description> | 省略可能 | 説明文。 |

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <transport-guarantee> | 必須 | クライアントとサーバ間でやり取りされるデータのセキュリティ要件を指定する。<br>指定できる値：<br>■ NONE - 転送の保証が不要な場合に指定する。<br>■ INTEGRAL - クライアントとサーバ間で、転送中にデータが変更されない方法でデータを転送する必要がある場合に指定する。<br>■ CONFIDENTIAL - 転送中にデータの中味を覗かれないようにデータを転送する必要がある場合に指定する。<br>**注意：** INTEGRAL または CONFIDENTIAL の転送保証を使用してユーザが認証を受けた場合、WebLogic Server はセキュア ソケット レイヤ (SSL) 接続を確立する。 |

### 使用する場所

user-data-constraint 要素は、security-constraint 要素内で使用されます。

### 例

web.xml ファイルでの user-data-constraint 要素の使用例については、コード リスト 2-12 を参照してください。

## web-resource-collection

web-resource-collection 要素は、セキュリティ制約を適用する Web アプリケーションのリソースおよび HTTP メソッドのサブセットを識別するために使用されます。HTTP メソッドが指定されていない場合、セキュリティ制約はすべての HTTP メソッドに適用されます。

次の表では、web-resource-collection 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <web-resource-name> | 必須 | Web リソースの集合の名前。 |
| <description> | 省略可能 | Web リソースの説明文。 |
| <url-pattern> | 必須 | Web リソースの集合のマッピング、つまり場所。 |
| <http-method> | 省略可能 | クライアントが Web リソースの集合にアクセスしようとするときにセキュリティ制約を適用する HTTP メソッド。HTTP メソッドが指定されていない場合、セキュリティ制約はすべての HTTP メソッドに適用される。 |

### 使用する場所

web-resource-collection 要素は、security-constraint 要素内で使用されます。

### 例

web.xml ファイルでの web-resource-collection 要素の使用例については、コード リスト 2-12 を参照してください。

# weblogic.xml デプロイメント記述子

以下の weblogic.xml のセキュリティ関連のデプロイメント記述子の要素は、WebLogic Server のセキュリティ要件を定義するために使用されます。

- 2-37 ページの「global-role」
- 2-39 ページの「security-permission」
- 2-39 ページの「security-permission-spec」
- 2-40 ページの「security-role-assignment」

weblogic.xml デプロイメント記述子の詳細については、http://www.bea.com/servers/wls700/dtd/weblogic700-web-jar.dtd にある weblogic.xml の文書型記述子 (DTD) を参照してください。

# global-role

WebLogic Server 7.0 SP1 以降では、Administration Console で指定するマッピングを、対となる web.xml ファイルの role-name 要素で定義した特定のセキュリティ ロールで使用することを明示的に示すために、global-role 要素を使用します。この要素は、principal-name 要素の代わりに、プレースホルダとして使用されます。

global-role 要素を使用すると、特定の Web アプリケーションのデプロイメント記述子に定義されたセキュリティ ロールごとに特定のセキュリティ ロール マッピングを指定する必要がなくなります。代わりに、Administration Console を使用して定義済みの各ロールに対する特定のロール マッピングをいつでも指定および変更できます。さらに、この要素は一部のアプリケーションに対して使用できるので、セキュリティ レルムの [ 一般 ] タブの [ デプロイメント記述子内のセキュリティ データを無視 ] 属性を有効にする必要がありません。このため、同じセキュリティ レルムの中で、デプロイメント記述子を使用して一部のアプリケーションのセキュリティを指定および変更する一方、Administration Console を使用して他のアプリケーションのセキュリティを指定および変更できます。

**注意：** セキュリティ ロール名を指定する場合、以下の規約と制限に従ってください。

- セキュリティ ロール名の適切な構文は、Web 上 (http://www.w3.org/TR/REC-xml#NT-Nmtoken) で閲覧可能な XML (Extensible Markup Language) 勧告で Nmtoken に関して定義されているとおりです。

- スペース、カンマ、ハイフン、\t、< >、#、|、&、~、?、( )、{ } を使用しないでください。

- セキュリティ ロール名では大文字 / 小文字を区別します。

- BEA が提唱する命名規約では、セキュリティ ロール名は単数形です。

## 使用する場所

global-role 要素は、security-role-assignment 要素内で使用されます。

## 例

コード リスト 2-13 とコード リスト 2-14 は、weblogic.xml ファイルでの global-role 要素の使い方を比較により示しています。コード リスト 2-14 では、weblogic-ejb-jar.xml 内の「webuser」の global-role 要素は、セキュリティが getReceipts メソッドにおいて正しくコンフィグレーションされるためには、Administration Console で webuser に対応するプリンシパルが作成される必要があることを意味します。

**コード リスト 2-13 web.xml ファイルと weblogic.xml ファイルを使用したセキュリティ ロールとプリンシパルのセキュリティ レルムへのマップ**

**web.xml entries:**

```
<web-app>
           ...
           <security-role>
               <role-name>webuser</role-name>
           </security-role>
           ...

</web-app>
```

**<weblogic.xml entries:**

```
<weblogic-web-app>

     <security-role-assignment>
         <role-name>webuser</role-name>
         <principal-name>myGroup</principal-name>
         <principal-name>Bill</principal-name>
         <principal-name>Mary</principal-name>
     </security-role-assignment>

</weblogic-web-app>
```

**コード リスト 2-14 Web アプリケーションのデプロイメント記述子での global-role 要素の使い方**

**web.xml entries:**

```
<web-app>
           ...
           <security-role>
               <role-name>webuser</role-name>
           </security-role>
           ...
```

```
</web-app>

<weblogic.xml entries:

<weblogic-web-app>

     <security-role-assignment>
         <role-name>webuser</role-name>
         <global-role/>
     </security-role-assignment>
```

Administration Console を使用して Web アプリケーションのセキュリティをコンフィグレーションする方法については、『WebLogic リソースのセキュリティ』を参照してください。

# security-permission

security-permission 要素は、J2EE Sandbox と関連するセキュリティ パーミッションを指定します。

## 例

security-permission 要素の使用例については、コード リスト 2-15 を参照してください。

# security-permission-spec

security-permission-spec 要素は、セキュリティ ポリシー ファイル構文に基づいて単一のセキュリティ パーミッションを指定します。Sun のセキュリティ パーミッション仕様の実装については、次の URL を参照してください。

http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html#FileSyntax

**注意：** オプションの codebase および signedBy 句は無視してください。

## 使用する場所

security-permission-spec 要素は、security-permission 要素内で使用されます。

### 例

コード リスト 2-15 は、security-permission-spec 要素を使用して java.net.SocketPermission クラスにパーミッションを付与する方法を示しています。

**コード リスト 2-15 security-permission-spec 要素の例**

```
<weblogic-web-app>
   <security-permission>
     <description>Optional explanation goes here</description>
     <security-permission-spec>
<!–
http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html
#FileSyntax の構文に準拠する、「codebase」句や「signedBy」句を含まない単一
の grant 文をここに記述します。次に例を示します。
-->
      grant {
      permission java.net.SocketPermission "*", "resolve";
      };
     </security-permission-spec>
   </security-permission>
</weblogic-web-app>
```

コード リスト 2-15 では、permission java.net.SocketPermission はパーミッション クラス名を、"*" は対象名を、resolve (host/IP 名サービスのルックアップを解決する ) はアクションを示します。

## security-role-assignment

security-role-assignment 要素は、セキュリティ ロールと WebLogic Server セキュリティ レルムの 1 つまたは複数のプリンシパルとのマッピングを宣言します。

### 例

コード リスト 2-16 は、security-role-assignment 要素を使用してプリンシパルを PayrollAdmin ロールに割り当てる方法を示しています。

**コード リスト 2-16　security-role-assignment 要素の例**

```
<weblogic-web-app>
  <security-role-assignment>
    <role-name>PayrollAdmin</role-name>
    <principal-name>Tanya</principal-name>
    <principal-name>Fred</principal-name>
    <principal-name>system</principal-name>
  </security-role-assignment>
</weblogic-web-app>
```

# Web アプリケーションでのプログラムによるセキュリティの使用

サーブレットを記述して、そのサーブレット コードでプログラム的にユーザとセキュリティ ロールにアクセスできます。そのためには、サーブレットのコードで javax.servlet.http.HttpServletRequest.getUserPrincipal および javax.servlet.http.HttpServletRequest.isUserInRole(String role) メソッドを使用します。

## getUserPrincipal

getUserPrincipal() メソッドは、Web アプリケーションの現在のユーザを特定する場合に使用します。このメソッドは、1 ユーザが存在する場合に WLSUser Principal を返します。WLSUser Principal が複数の場合、メソッドは、Subject.getPrincipals().iterator() メソッドで指定された順序の 1 番目を返します。WLSUser Principal が存在しない場合、getUserPrincipal() メソッドは WLSGroup Principal 以外の 1 番目を返します。Principal が存在しない場合、またはすべての Principal が WLSGroup タイプの場合、このメソッドは null を返します。この動作は、weblogic.security.SubjectUtils.getUserPrincipal() メソッドのセマンティクスとまったく同じです。

getUserPrincipal() メソッドの使い方については、http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Security4.html を参照してください。

## isUserInRole

javax.servlet.http.HttpServletRequest.isUserInRole(String role) メソッドは、認証済みのユーザに指定された論理的セキュリティ「ロール」が付与されているかどうかを示すブール値を返します。ユーザが認証されていない場合、このメソッドは false を返します。

isUserInRole() メソッドは、セキュリティ ロールをセキュリティ レルムのグループ名にマップします。コード リスト 2-17 は、対となる web.xml ファイルでセキュリティ ロールを定義するために <servlet> 要素とともに使用される要素を示しています。

**コード リスト 2-17 IsUserInRole の web.xml および weblogic.xml の要素**

```
Begin web.xml entries:

...
<servlet>
        <security-role-ref>
             <role-name>user-rolename</role-name>
             <role-link>rolename-link</role-link>
        </security-role-ref>
</servlet>

<security-role>
         <role-name>rolename-link</role-name>
</security-role>
...

Begin weblogic.xml entries:

...
<security-role-assignment>
         <role-name>rolename-link</role-name>
         <principal-name>groupname</principal>
         <principal-name>username</principal>
</security-role-assignment>
...
```

文字列 role は、対となる web.xml デプロイメント記述子の <servlet> 宣言の <security-role-ref> 要素の中にネストされた <role-name> 要素で提供される名前にマップされます。<role-name> 要素は、サーブレット コードで使用されるセキュリティ ロールまたは principal ( ユーザまたはグループ ) の名前を定義します。<role-link> 要素は、weblogic.xml デプロイメント記述子の <security-role-assignment> 要素で定義された <role-name> に対応します。

**注意：** セキュリティ ロール名を指定する場合、以下の規約と制限に従ってください。

- セキュリティ ロール名の適切な構文は、Web 上 (`http://www.w3.org/TR/REC-xml#NT-Nmtoken`) で閲覧可能な XML (Extensible Markup Language) 勧告で `Nmtoken` に関して定義されているとおりです。

- スペース、カンマ、ハイフン、\t、< >、#、|、&、~、?、( )、{ } を使用しないでください。

- セキュリティ ロール名では大文字 / 小文字を区別します。

- BEA が提唱する命名規約では、セキュリティ ロール名は単数形です。

たとえば、クライアントが `manager` というセキュリティ ロールの `Bill` というユーザでログインに成功している場合、次のメソッドは true を返します。

```
request.isUserInRole("manager")
```

コード リスト 2-18 に例を示します。

**コード リスト 2-18 セキュリティ ロール マッピングの例**

```
Servlet code:
out.println("Is the user a Manager? " +
                        request.isUserInRole("manager"));

web.xml entries:

<servlet>
. . .
   <role-name>manager</role-name>
   <role-link>mgr</role-link>
. . .
</servlet>

<security-role>
   <role-name>mgr</role-name>
</security-role>

weblogic.xml entries:

<security-role-assignment>
   <role-name>mgr</role-name>
   <principal-name>bostonManagers</principal-name>
   <principal-name>Bill</principal-name>
   <principal-name>Ralph</principal-name>
</security-role-ref>
```

# プログラムによる認証 API の使用

一部のアプリケーションでは、プログラムによる認証を使用するのが適切です。

WebLogic Server は、サーブレット アプリケーション内からプログラムによる認証をサポートするサーバサイド API を備えています。

```
weblogic.servlet.security.ServletAuthentication
```

この API を使用すると、ユーザを認証し、そのユーザをログインさせ、デフォルトの ( アクティブな ) セキュリティ レルムで登録されるように現行セッションと関連付けるサーブレット コードを記述できます。いったんログインが完了すると、標準のメカニズムを使用してログインしたかのように感じられます。

`ServletAuthentication` API と一緒に WebLogic が提供する `weblogic.security.SimpleCallbackHandler` クラスまたは `weblogic.security.URLCallbackHandler` クラスのうちいずれかを使用できます。これらのクラスの詳細については、WebLogic クラスに関する Javadoc を参照してください。

コード リスト 2-19 は、`SimpleCallbackHandler` を使用した例です。コード リスト 2-20 は、`URLCallbackHandler` を使用した例です。

**コード リスト 2-19　SimpleCallbackHandler クラスを使用したプログラムによる認証コードの一部分**

```
CallbackHandler handler = new SimpleCallbackHandler(username, password);
Subject mySubject = weblogic.security.services.Authentication.login(handler);
weblogic.servlet.security.ServletAuthentication.runAs(mySubject, request);
```

`request` は `httpservletrequest` オブジェクト

**コード リスト 2-20　URLCallbackHandler クラスを使用したプログラムによる**

**認証コードの一部分**

```
CallbackHandler handler = new URLCallbackHandler(username, password);
Subject mySubject = weblogic.security.services.Authentication.login(handler);
weblogic.servlet.security.ServletAuthentication.runAs(mySubject, request);
```

request は httpservletrequest オブジェクト

# 3 Java クライアントでの JAAS 認証の使用

この節では、以下のトピックについて説明します。


- 3-1 ページ「JAAS と WebLogic Server」
- 3-3 ページ「JAAS 認証の開発環境」
- 3-12 ページ「JAAS 認証を使用するクライアント アプリケーションの作成」
- 3-22 ページ「JNDI 認証の使い方」
- 3-24 ページ「Java クライアントの JAAS 認証コード例」


## JAAS と WebLogic Server

Authentication and Authorization Service (JAAS) は、Java Software Development Kit バージョン 1.3 のセキュリティに対する標準拡張です。JAAS では、ユーザの ID に基づいてアクセス制御を実行できます。JAAS は、JNDI 認証メカニズムの代わりとして WebLogic Server で提供されています。

WebLogic Server クライアントは、標準の JAAS の認証部分のみを利用します。JAAS LoginContext は、コンフィグレーションされているすべての認証プロバイダの LoginModule インスタンスを順番どおりに実行し、コンフィグレーションされている各プロバイダの完了ステータスの管理を行います。

Java クライアントに対して JAAS 認証を使用する場合は、次のことに注意してください。

- WebLogic Server クライアントは認証に JNDI ログインを使うことも JAAS ログインを使うこともできますが、JAAS のほうが好ましい方法です。
- JAAS は認証方法として好ましい選択肢ですが、WebLogic 提供の LoginModule

(`weblogic.security.auth.login.UsernamePasswordLoginModule`) はユーザ名とパスワードの認証しかサポートしていません。したがって、クライアントの証明書認証 ( 相互 SSL 認証とも言う ) では JNDI を使用します。クライアント証明書の認証に JAAS を使用するには、証明書の認証を行うカスタム LoginModule を記述する必要があります。

**注意：** WebLogic Server クライアントで使用する LoginModule を自分で作成する場合は、`weblogic.security.auth.Authenticate.authenticate()` を呼び出してログインを実行するように記述する必要があります。

- リモートの Java クライアントから JAAS ログインを実行する ( つまり、Java クライアントが WebLogic Server クライアントではない ) 場合には、WebLogic 提供の LoginModule をログインの実行に使用できます。しかし、WebLogic 提供の LoginModule を使用せずに自分で作成することを選択した場合は、`weblogic.security.auth.Authenticate.authenticate()` メソッドを呼び出してログインを実行するように記述する必要があります。
- Security Assertion Markup Language (SAML) などリモートまたは境界のログイン システムを使用する場合は、`weblogic.security.auth.Authenticate.authenticate()` を呼び出す必要はありません。WebLogic Server を使用してログインを実行する場合は、`authenticate()` メソッドを呼び出すだけで済みます。

  **注意：** WebLogic Server は、JAAS 認証に対して完全なコンテナ サポートを提供し、アプリケーション コードにおける JAAS の認証と認可の完全な使用をサポートしています。

- WebLogic Server では、すべてのログインの実行に JAAS が呼び出されます。各認証プロバイダが、LoginModule を含みます。これは、JNDI または JAAS を介しての Java クライアントのログインだけでなく、サーブレットのログインについても当てはまります。WebLogic Server が JAAS ログオンを実行するために内部で呼び出すメソッドは、`weblogic.security.services.Authentication.authenticate()` です。認証クラスを使用する場合、ヘルパー クラスとしては `weblogic.security.SimpleCallbackHandler` が有用であると考えられます。
- WebLogic Server は、認証と認可に関して JAAS 1.0 参照実装を完全にサポートしています。WebLogic Server は JAAS 認可を使用してリソースを保護することはしませんが (WebLogic セキュリティを使用 )、アプリケーション

コードで JAAS 認可を使用してアプリケーションの独自のリソースを保護することはできます。

JAAS の詳細については、Web 上の `http://java.sun.com/security/jaas/doc/api.html` にある『Java Authentication and Authorization Service Developer's Guide』を参照してください。

# JAAS 認証の開発環境

クライアントがアプリケーション、アプレット、エンタープライズ JavaBean (EJB)、または認証が必要なサーブレットのどれであるかに関係なく、WebLogic Server は Java Authentication and Authorization Service (JAAS) のクラスを使用して安全かつ確実にそのサーバに対して認証を行います。JAAS は、プラグイン可能な認証モジュール (PAM) フレームワークの Java バージョンを実装します。このフレームワークにより、アプリケーションは基の認証技術から独立することができます。このため、PAM フレームワークを利用することで、Java アプリケーションに修正を加えることなく新しいまたはアップデート版の認証技術を使用することができます。

WebLogic Server は、リモートの Java クライアントの認証および内部の認証で JAAS を使用します。したがって、JAAS に直に関与する必要があるのは、カスタム認証プロバイダの開発者とリモートの Java クライアント アプリケーションの開発者だけです。Web ブラウザ クライアントのユーザまたはコンテナ内の Java クライアント アプリケーション ( サーブレットからエンタープライズ JavaBeans を呼び出すものなど ) の開発者は、JAAS を直接使用したり、その知識を身につけたりする必要はありません。

**注意：** Java Authentication and Authorization Service (JAAS) および Java Naming And Directory Interface (JNDI) は、いずれもセキュリティが確保された状態で WebLogic Server のインスタンスへログインするために WebLogic Server 上で実行される Java クライアントで使用できますが、JAAS のほうが好ましい方法です。

**注意：** WebLogic クライアントでセキュリティを実装するためには、Java クライアントで WebLogic Server ソフトウェア配布キットをインストールする必要があります。

この章では、以下の内容について説明します。


- 3-4 ページ「JAAS 認証 API」
- 3-9 ページ「JAAS クライアント アプリケーション コンポーネント」
- 3-11 ページ「WebLogic LoginModule 実装」
- 3-11 ページ「JVM 全体のデフォルト ユーザと runAs() メソッド」


# JAAS 認証 API

WebLogic Server で JAAS 認証を使用する Java クライアントを実装するには、Java SDK 1.3 アプリケーション プログラミング インタフェース (API) と WebLogic API を組み合わせて使用します。

表 3-1 では、JAAS 認証の実装に使用される Java SDK API パッケージについて説明します。表 3-1 の情報は Java SDK API のマニュアルから取られており、WebLogic Server 固有の情報がコメントで追加されています。Java SDK API の詳細については、http://java.sun.com/j2ee/sdk_1.3/techdocs/api/index.html の Javadoc を参照してください。

表 3-2 では、JAAS 認証の実装に使用される WebLogic API について説明します。詳細については、WebLogic クラスに関する Javadoc を参照してください。

**表 3-1 Java SDK JAAS API**

| Java SDK JAAS API | 説明 |
|---|---|
| `javax.security.auth.Subject` | `Subject` クラスは、リクエストのソースを表し、個々のユーザの場合もグループの場合もある。`Subject` オブジェクトは、ユーザのログインが正常に完了した後でのみ作成される。 |

**表 3-1 Java SDK JAAS API**

| Java SDK JAAS API | 説明 |
|---|---|
| javax.security.auth.login.LoginContext | LoginContext クラスは、Subjects の認証に使用される基本メソッドを記述し、基礎となる認証テクノロジから独立したアプリケーションを開発する方法を提供する。Configuration は、特定のアプリケーションで使用する認証テクノロジまたは LoginModule を指定する。したがって、アプリケーション自体はまったく変更せずに、異なる LoginModule をアプリケーションに組み込むことができる。<br>呼び出し側が LoginContext をインスタンス化すると、login メソッドを呼び出して Subject を認証する。この login メソッドは、呼び出し側で指定した名前についてコンフィグレーションされた LoginModule の各々からの login メソッドを呼び出す。<br>login メソッドが例外を送出することなく復帰すれば、認証は総合的に成功したことになる。呼び出し側はその後、getSubject メソッドを呼び出すことによって、新しく認証された Subject を取得できる。Subject と関連付けられたプリンシパルおよび資格は、Subject の getPrincipals、getPublicCredentials、および getPrivateCredentials メソッドをそれぞれ呼び出すことで取得できる。<br>Subject からログアウトするには、呼び出し側は logout メソッドを呼び出すだけでよい。login メソッドの場合と同じく、この logout メソッドはこの LoginContext についてコンフィグレーションされた各 LoginModule に対し logout メソッドを呼び出す。<br>このクラスの実装サンプルについては、コード リスト 3-4 を参照。 |

**表 3-1 Java SDK JAAS API**

| Java SDK JAAS API | 説明 |
|---|---|
| `javax.security.auth.login.Configuration` | これは、アプリケーションにおける LoginModule のコンフィグレーションを表す抽象クラスである。`Configuration` では、特定のアプリケーションについて、使用する LoginModule、および LoginModule を呼び出す順番を指定する。この抽象クラスは、実際のコンフィグレーションを読み取ってロードする実装を提供するようサブクラス化する必要がある。<br><br>WebLogic Server では、このクラスの代わりにログイン コンフィグレーション ファイルを使用する。サンプル コンフィグレーション ファイルについては、3-16 ページのコード リスト 3-3 「sample_jaas.config のコード例」を参照。デフォルトでは、WebLogic Server はコンフィグレーション ファイルから読み取りを行う、Sun Microsystems, Inc. によるコンフィグレーション クラスを使用する。 |
| `javax.security.auth.spi.LoginModule` | `LoginModule` では、認証テクノロジ プロバイダによって実装されるインタフェースを記述する。`LoginModule` は、特定のタイプの認証を提供するためにアプリケーションに組み込まれる。<br><br>アプリケーション開発者は `LoginContext` API に書き込み、API 認証テクノロジ プロバイダは LoginModule インタフェースを実装する。コンフィグレーションでは、特定のログイン アプリケーションで使用する LoginModule を指定する。したがって、アプリケーション自体はまったく変更せずに、異なる LoginModule をアプリケーションに組み込むことができる。<br><br>**注意：** WebLogic Server は、LoginModule の実装 (`weblogic.security.auth.login.UsernamePasswordLoginModule`) を提供する。WebLogic Server の Java クライアントでは、この JAAS 認証の実装の使用が望ましいが、独自の LoginModule を開発することも可能である。コード リスト 3-3 は、WebLogic Server の LoginModule を呼び出す方法を示す。 |

**表 3-1 Java SDK JAAS API**

| Java SDK JAAS API | 説明 |
|---|---|
| javax.security.auth.<br>callback.Callback | このインタフェースの実装は CallbackHandler に渡される。それにより、基礎となるセキュリティ サービスは呼び出し側アプリケーションと対話して、ユーザ名やパスワードなど特定の認証データを取得したり、エラーや警告メッセージなど特定の情報を表示できるようになる。<br>Callback 実装では、基礎となるセキュリティ サービスによって要求された情報の取得または表示は行わない。Callback 実装は、単にそのようなリクエストをアプリケーションに渡す手段と、場合に応じてアプリケーションが要求された情報を基礎となるセキュリティ サービスに返す手段を提供する。<br>このインタフェースの実装サンプルについては、コード リスト 3-2 を参照。 |
| javax.security.auth.<br>callback.CallbackHandler | アプリケーションは、CallbackHandler を実装し、それを基礎となるセキュリティ サービスに渡すことで、それらのサービスがアプリケーションと対話してユーザ名やパスワードなど特定の認証データを取得したり、エラーや警告メッセージなど特定の情報を表示したりできるようにする。<br>CallbackHandler は、アプリケーションに依存する形で実装される。<br>基礎となるセキュリティ サービスは、個々の Callbacks を CallbackHandler に渡すことによってさまざまなタイプの情報に対する要求を行う。CallbackHandler 実装は、渡された Callbacks に応じて情報を取得したり表示したりする方法を決定する。たとえば、基礎となるサービスがユーザを認証するためにユーザ名とパスワードを必要としていれば、NameCallback および PasswordCallback を使用する。その後、CallbackHandler は逐次的にユーザ名およびパスワードの入力を求めるか、または単一のウィンドウで両方の入力を求めるかを選択できる。<br>このインタフェースの実装サンプルについては、コード リスト 3-2 を参照。 |

**表 3-2 WebLogic JAAS API**

| WebLogic JAAS API | 説明 |
|---|---|
| weblogic.security.auth.Authenticate | ユーザ資格の認証に使用される認証クラス。<br>LoginModule の WebLogic Server の実装 (`weblogic.security.auth.login.UsernamePasswordLoginModule`) では、このクラスを使用してユーザを認証し、`Principals` を `Subject` に追加する。ユーザの記述による LoginModule も、同じ目的でこのクラスを使用する必要がある。 |
| `weblogic.security.auth.Callback.URLCallback` | 基礎となるセキュリティ サービスでは、このクラスを使用して `URLCallback` をインスタンス化し、`CallbackHandler` の `invokeCallback` メソッドに渡して、URL 情報を取得する。<br>`LoginModule` の WebLogic Server の実装 (`weblogic.security.auth.login.UsernamePasswordLoginModule`) ではこのクラスを使用する。<br>**注意：** アプリケーション開発者は、URL 情報の取得にこのクラスを使用すべきではない。代わりに、`weblogic.security.URLCallbackHandler` を使用する。 |
| `weblogic.security.Security` | このクラスは WebLogic Server クライアントの `runAs` メソッドを実装する。クライアント アプリケーションは、実行する `PrivilegedAction` または `PrivilegedExceptionAction` と `Subject` ID を、`runAs` メソッドを使用して関連付ける。<br>実装のサンプルについては、コード リスト 3-6 を参照。 |
| `weblogic.security.URLCallbackHandler` | アプリケーション開発者が `username`、`password`、および `URL` を返すために使用する `class`。アプリケーション開発者は、このクラスを使用して `URLCallback` を処理し、URL 情報を取得する。 |

# JAAS クライアント アプリケーション コンポーネント

最少でも、JAAS 認証クライアント アプリケーションは、以下のコンポーネントで構成されています。

- Java クライアント

  Java クライアントは `LoginContext` オブジェクトをインスタンス化し、オブジェクトの `login()` メソッドを呼び出すことでログインを呼び出します。`login()` メソッドは各 LoginModule 内のメソッドを呼び出して、ログインおよび認証を実行します。

  LoginContext はまた、新しい空の `javax.security.auth.Subject` オブジェクト ( 認証されているユーザまたはサービスを表す ) をインスタンス化し、コンフィグレーションされた LoginModule を作成し、それをこの新しい `Subject` および `CallbackHandler` で初期化します。

  LoginContext は、その後 LoginContext の `getSubject` メソッドを呼び出すことで、認証された Subject を取得します。LoginContext は `weblogic.security.Security` クラス `runAs()` メソッドを使って、ユーザ ID に代わって実行される `PrivilegedAction` または `PrivilegedExceptionAction` と `Subject` の ID を関連付けます。

- LoginModule

  LoginModule は `CallbackHandler` を利用して、ユーザ名およびパスワードを取得し、そのユーザ名とパスワードが予想どおりのものであるかどうかを確認します。

  認証に成功すると、LoginModule は、ユーザを表すプリンシパルを主体に格納します。LoginModule が主体に格納するプリンシパルは `Principal` のインスタンスであり、`java.security.Principal` インタフェースを実装するクラスです。

  LoginModule ファイルは、ユーザ名およびパスワードの認証や証明書の認証など、さまざまなタイプの認証を実行するように記述できます。クライアント アプリケーションに含める LoginModule は、1 つ ( 最低要件 ) でも複数でもかまいません。

  **注意：** WebLogic Server アプリケーションで JAAS `javax.security.auth.Subject.doAs` メソッドを使用する場合、サ

ブジェクトとクライアント アクションは関連付けられません。doAs メソッドを使用して WebLogic Server アプリケーションに J2SE セキュリティを実装することは可能ですが、この場合でも Security.runAs() メソッドを使用する必要があります。

- Callbackhandler

  CallbackHandler は、javax.security.auth.callback.CallbackHandler インタフェースを実装します。LoginModule は、CallbackHandler を使って、ユーザと対話し、ユーザ名やパスワードなどの要求された情報を取得します。

- コンフィグレーション ファイル

  このファイルでは、アプリケーションで使用する LoginModule をコンフィグレーションします。LoginModule の場所と、複数の LoginModule がある場合は、実行する順番を指定します。このファイルを使うと、Java アプリケーションは LoginModule を使って定義および実装される認証テクノロジから独立を保つことができます。

- アクション ファイル

  このファイルでは、クライアント アプリケーションが実行する処理を定義します。

- ant build script (build.xml)

  このスクリプトは、アプリケーションに必要なすべてのファイルをコンパイルし、それらを WebLogic Server のアプリケーション ディレクトリにデプロイします。

ここで説明しているコンポーネントを実装する完全な実践的 JAAS 認証クライアントについては、WebLogic Server で提供されている *SAMPLES_HOME*\server\src\examples\security\jaas ディレクトリの JAAS サンプル アプリケーションを参照してください。

JAAS 認証の基本の詳細については、Sun の『JAAS Authentication Tutorial』(http://java.sun.com/j2se/1.4/docs/guide/security/jaas/tutorials/GeneralAcnOnly.html) を参照してください。

# WebLogic LoginModule 実装

LoginModule クラスの WebLogic 実装は、WebLogic Server 配布キットにおいて、WL_HOME\server\lib ディレクトリの weblogic.jar ファイルで提供されています。

**注意：** WebLogic Server は、JAAS で定義されているすべてのコールバックのタイプだけではなく、JAAS 仕様を拡張するコールバックのタイプもすべてサポートしています。

WebLogic Server 製品に付属する UsernamePasswordLoginModule は、実行に先立ってシステム ユーザの認証定義が存在するかどうかを調べ、既に定義されている場合は何も行いません。

**注意：** WebLogic LoginModule が Java 仮想マシン全体のデフォルト ユーザに与える影響と、runAs() メソッドとの関連については、3-11 ページ「JVM 全体のデフォルト ユーザと runAs() メソッド」を参照してください。

JASS LoginModule の実装方法の詳細については、『Java Authentication and Authorization Service Developer's Guide』を参照してください。

# JVM 全体のデフォルト ユーザと runAs() メソッド

WebLogic Server の LoginModule 実装 (weblogic.security.auth.login.UsernamePasswordLoginModule) を初めて使用してログインすると、指定されたユーザが JVM (Java 仮想マシン ) に対するマシン全体のデフォルト ユーザとなります。

weblogic.security.Security.runAs() メソッドを実行すると、このメソッドは指定された Subject と現在のスレッドのアクセス パーミッションを関連付けた後、アクションを実行します。指定された Subject が特権を持たないユーザ ( グループに割り当てられていないユーザは特権を持たないと見なされる ) を表している場合、JVM 全体のデフォルト ユーザが使用されます。したがって、runAs() メソッドには必要な Subject を指定することが重要です。それには、以下のオプションがあります。

- オプション **1 :** クライアントが main() の制御を持つ場合、コード リスト 3-1 に示すラッパー コードをクライアント コードに実装します。

**コード リスト 3-1 runAs() メソッドのラッパー コード**

```
import java.security.PrivilegedAction;
import javax.security.auth.Subject;
import weblogic.security.Security;

public class client
{
  public static void main(String[] args)
  {
  Security.runAs(new Subject(),
    new PrivilegedAction() {
     public Object run() {
      //
      // クライアント コードに実装する場合、main() はここに記述する
      //
     return null;
     }
   });
  }
}
```

- オプション **2 :** クライアントが `main()` の制御を持たない場合、コード リスト 3-1 に示したラッパー コードを各スレッドの `run()` メソッドに実装します。

# JAAS 認証を使用するクライアント アプリケーションの作成

WebLogic Server Java クライアントにおいて JAAS を使って主体を認証するには、以下の手順を実行します。

1. WebLogic Server で使用する認証メカニズム用に `LoginModule` クラスを実装します。認証メカニズムのタイプごとに LoginModule クラスが必要となります。1 つの WebLogic Server デプロイメントに対して複数の LoginModule クラスを割り当てることができます。`LoginModule` クラスの実装方法については、`http://java.sun.com/security/jaas/doc/api.html` の『Java Authentication and Authorization Service (JAAS) 1.0 Developer's Guide 』を参照してください。

**注意：** ユーザ名およびパスワードの認証には、WebLogic Server 提供の LoginModule の実装 (`weblogic.security.auth.login.UsernamePasswordLoginModule`) を使用することをお勧めします。必要であれば、ユーザ名およびパスワードの認証用に独自の `LoginModule` を作成することもできます。しかし、WebLogic Server の `LoginModule` を修正して再利用しようとはしないでください。LoginModule を自分で作成する場合は、ログインを実行するために `weblogic.security.auth.Authenticate.authenticate()` メソッドを呼び出すように記述します。SAML などリモートのログイン システムを使用している場合は、`authenticate()` メソッドを呼び出す必要はありません。WebLogic Server を使用してログインを実行する場合は、`authenticate()` を呼び出すだけで済みます。

表 3-3 で説明されているように、`weblogic.security.auth.Authenticate` クラスは初期コンテキストとして JNDI Environment オブジェクトを使用します。

2. LoginModule を使用して、ユーザと対話し、ユーザ名やパスワードなどの要求された情報を取得する `CallbackHandler` クラスを実装します。WebLogic Server 配布キットに付属する JAAS クライアント サンプルで使われているサンプル `CallbackHandler` については、コード リスト 3-2 を参照してください。

   **注意：** 独自の `CallbackHandler` クラスを実装する代わりに、2 つの WebLogic 提供による `CallbackHandler` クラス、すなわち `weblogic.security.SimpleCallbackHandler` または `weblogic.security.URLCallbackHandler` のいずれかを使用できます。これらのクラスの詳細については、WebLogic クラスに関する Javadoc を参照してください。

**コード リスト 3-2 CallbackHandler インタフェースの実装**

```
package examples.security.jaas;

import java.io.*;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.callback.TextOutputCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.TextInputCallback;
```

```
import javax.security.auth.callback.NameCallback;
import weblogic.security.auth.callback.URLCallback;
import examples.utils.common.ExampleUtils;

/**
 * SampleCallbackHandler.java
 * CallbackHandler インタフェースの実装
 *
 * @author Copyright (c) 2000-2002 by BEA Systems, Inc. All Rights
 * Reserved.
 */
class SampleCallbackHandler implements CallbackHandler
{
  private String username = null;
  private String password = null;
  private String url = null;

  public SampleCallbackHandler() { }

  public SampleCallbackHandler(String pUsername, String pPassword,
                               String pUrl)
  {
    username = pUsername;
    password = pPassword;
    url = pUrl;
  }

  public void handle(Callback[] callbacks) throws IOException,
                                    UnsupportedCallbackException
  {
    for(int i = 0; i < callbacks.length; i++)
    {
      if(callbacks[i] instanceof TextOutputCallback)
      {
// 指定したタイプに従ってメッセージを表示
        TextOutputCallback toc = (TextOutputCallback)callbacks[i];
        switch(toc.getMessageType())
        {
        case TextOutputCallback.INFORMATION:
          ExampleUtils.log(toc.getMessage());
          break;
        case TextOutputCallback.ERROR:
          ExampleUtils.log("ERROR: " + toc.getMessage());
          break;
        case TextOutputCallback.WARNING:
          ExampleUtils.log("WARNING: " + toc.getMessage());
          break;
        default:
          throw new IOException("Unsupported message type: " +
                                toc.getMessageType());
        }
      }
      else if(callbacks[i] instanceof NameCallback)
      {
        // コマンドラインでユーザ名が未入力の場合は、
        // ユーザにユーザ名の入力を促す
        NameCallback nc = (NameCallback)callbacks[i];
```

```
    if (ExampleUtils.isEmpty(username)) {
      System.err.print(nc.getPrompt());
      System.err.flush();
      nc.setName((new BufferedReader(new
                  InputStreamReader(System.in))).readLine());
    }
    else {
      ExampleUtils.log("username: "+username);
      nc.setName(username);
    }
  }
  else if(callbacks[i] instanceof URLCallback)
  {
    // コマンドラインで URL が未入力の場合は、ユーザに URL の
    // 入力を促す
    // このサンプルでは URL が必要
    URLCallback uc = (URLCallback)callbacks[i];
    if (ExampleUtils.isEmpty(url)) {
      System.err.print(uc.getPrompt());\
      System.err.flush();
      uc.setURL((new BufferedReader(new
                  InputStreamReader(System.in))).readLine());\
    }
    else {
      ExampleUtils.log("URL: "+url);


      uc.setURL(url);
    }
  }
  else if(callbacks[i] instanceof PasswordCallback)
  {
    PasswordCallback pc = (PasswordCallback)callbacks[i];

    // コマンドラインでパスワードが未入力の場合は、
    // ユーザにパスワードの入力を促す
    if (ExampleUtils.isEmpty(password)) {
      System.err.print(pc.getPrompt());
      System.err.flush();

      // 注意 : JAAS の仕様では、パスワードは String ではなく
      // char[] である
      String tmpPassword = (new BufferedReader(new
                    InputStreamReader(System.in))).readLine();
      int passLen = tmpPassword.length();
      char[] passwordArray = new char[passLen];
      for(int passIdx = 0; passIdx < passLen; passIdx++)
        passwordArray[passIdx] = tmpPassword.charAt(passIdx);
      pc.setPassword(passwordArray);
    }
    else {
      String tPass = new String();
      for(int p = 0; p < password.length(); p++)
        tPass += "*";
      ExampleUtils.log("password: "+tPass);
      pc.setPassword(password.toCharArray());
```

```
        }
      }
      else if(callbacks[i] instanceof TextInputCallback)
      {
// ユーザ名の入力を要求
        TextInputCallback callback =
                              (TextInputCallback)callbacks[i];
        System.err.print(callback.getPrompt());
        System.err.flush();
        callback.setText((new BufferedReader(new
                      InputStreamReader(System.in))).readLine());
      }
      else
      {
        throw new UnsupportedCallbackException(callbacks[i],
                                          "Unrecognized Callback");
      }
    }
  }
}
```

3. WebLogic Server で使用する LoginModule クラス、および LoginModule クラスを呼び出す順序を指定するコンフィグレーション ファイルを記述します。WebLogic Server 配布キットに付属する JAAS クライアント サンプルで使われているサンプルのコンフィグレーション ファイルについては、コード リスト 3-3 を参照してください。

**コード リスト 3-3　sample_jaas.config のコード例**

```
/** JAAS サンプル アプリケーション用のログイン コンフィグレーション **/

Sample {
   weblogic.security.auth.login.UsernamePasswordLoginModule
         required debug=false;
};
```

4. Java クライアントにおいて、`LoginContext` をインスタンス化するコードを記述します。`LoginContext` は、コンフィグレーション ファイル `sample_jaas.config` を調べて、WebLogic Server 用にコンフィグレーションされているデフォルトの LoginModule をロードします。`LoginContext` のインスタンス化については、コード リスト 3-4 を参照してください。

**コード リスト 3-4 LoginContext のコードの一部分**

```
...
import javax.security.auth.login.LoginContext;
...
    LoginContext loginContext = null;

    try
    {
      // LoginContext を作成する。ユーザ名 / パスワード ログイン モジュール
を指定する
      loginContext = new LoginContext("Sample",
            new SampleCallbackHandler(username, password, url));
    }
```

**注意：** ID アサーション プロバイダや WebLogic Server のリモート インスタンスなどの他の手段を使ってユーザを認証する場合には、デフォルト LoginModule はリモート リソースによって決定されます。

5. `LoginContext` インスタンスの `login()` メソッドを呼び出します。`login()` メソッドによって、ロードされた LoginModule がすべて呼び出されます。各 LoginModule で主体の認証が試みられます。コンフィグレーションされているログイン条件が満たされない場合、`LoginContext` は `LoginException` を送出します。`login()` メソッドの例については、コード リスト 3-5 を参照してください。

**コード リスト 3-5 login() メソッドのコードの一部分**

```
...
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;
import javax.security.auth.login.FailedLoginException;
import javax.security.auth.login.AccountExpiredException;
import javax.security.auth.login.CredentialExpiredException;
...
 /**
     * 認証を試みる
     */
    try
    {
      // 例外が発生せずに復帰した場合は、認証に成功
      loginContext.login();

    }
```

```
catch(FailedLoginException fle)

{
  System.out.println("Authentication Failed, " +
                          fle.getMessage());
  System.exit(-1);
}
catch(AccountExpiredException aee)
{
 System.out.println("Authentication Failed: Account Expired");
  System.exit(-1);
}
catch(CredentialExpiredException cee)
{
  System.out.println("Authentication Failed: Credentials
                           Expired");
  System.exit(-1);
}
catch(Exception e)
{
  System.out.println("Authentication Failed: Unexpected
                         Exception, " + e.getMessage());
  e.printStackTrace();
  System.exit(-1);
}
```

6. Java クライアント内にコードを記述して、`javax.security.auth.Subject.getSubject()` メソッドを使用する `LoginContext` インスタンスから認証された `Subject` を取得し、`Subject` としてアクションを呼び出します。`Subject` の認証に成功したら、`weblogic.security.Security.runAs()` メソッドを呼び出すことで、その Subject に対してアクセス制御を設定できます。`runAs()` メソッドは、指定された `Subject` と現在のスレッドのアクセス パーミッションを関連付けた後、アクションを実行します。`getSubject()` メソッドと `runAs()` メソッドの実装例については、コード リスト 3-6 を参照してください。

   **注意：** WebLogic LoginModule が Java 仮想マシン全体のデフォルト ユーザに与える影響と、`runAs()` メソッドとの関連については、3-11 ページ「JVM 全体のデフォルト ユーザと runAs() メソッド」を参照してください。

   **注意：** WebLogic Server アプリケーションで JAAS `javax.security.auth.Subject.doAs` メソッドを使用する場合、`Subject` とクライアント アクションは関連付けられません。`doAs` メソッドを使用して WebLogic Server アプリケーションに J2SE セキュ

リティを実装することは可能ですが、この場合でも Security.runAs() メソッドを使用する必要があります。

**コード リスト 3-6 getSubject() メソッドと runAs() メソッドのコードの一部分**

```
...
/**
 * 認証された主体を取得し、主体として SampleAction を実行する
 */
   Subject subject = loginContext.getSubject();
   SampleAction sampleAction = new SampleAction(url);
   Security.runAs(subject, sampleAction);
   System.exit(0);

...
```

7. Subject が必要な特権を持っている場合にアクションを実行するコードを記述します。株式取引用の EJB を実行する javax.security.PrivilegedAction クラスのサンプル実装については、コード リスト 3-7 を参照してください。

**コード リスト 3-7 PrivilegedAction の実装例**

```
package examples.security.jaas;

import java.security.PrivilegedAction;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.util.Hashtable;
import javax.ejb.CreateException;
import javax.ejb.EJBException;
import javax.ejb.FinderException;
import javax.ejb.ObjectNotFoundException;
import javax.ejb.RemoveException;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import examples.ejb20.basic.statelessSession.TraderHome;
import examples.ejb20.basic.statelessSession.Trader;
import examples.utils.common.ExampleUtils;

/**
 * SampleAction.java
 *
 * JAAS サンプル PrivilegedAction の実装
```

```
 *
 * @author Copyright (c) 2000-2002 by BEA Systems, Inc. All Rights
 * Reserved.
 */
public class SampleAction implements PrivilegedAction
{
  private static final String JNDI_NAME =
            "ejb20-statelessSession-TraderHome";
  private String url;

  public SampleAction(String url)
  {
    this.url = url;
  }

  public Object run()
  {
    Object obj = null;

    try {
      callTraderEJB();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    return obj;
  }

  /**
   * Trader EJB を呼び出す
   */
  public void callTraderEJB()
    throws NamingException, CreateException, RemoteException,
              RemoveException
  {
    TraderHome home = lookupTraderHome();

    // Trader を作成する
    ExampleUtils.log("Creating a trader");
    Trader trader = (Trader)ExampleUtils.narrow(home.create(),
                      Trader.class);

    String [] stocks = {"BEAS", "MSFT", "AMZN", "HWP" };

      // 買いを実行する
    for (int i=0; i<stocks.length; i++) {
      int shares = (i+1) * 100;
      ExampleUtils.log("Buying "+shares+" shares of
                        "+stocks[i]+".");
      trader.buy(stocks[i], shares);
    }

    // 売りを実行する
    for (int i=0; i<stocks.length; i++) {
      int shares = (i+1) * 100;
      ExampleUtils.log("Selling "+shares+" shares of
                          "+stocks[i]+".");
```

```
      trader.sell(stocks[i], shares);
    }

    // Trader を削除する
    ExampleUtils.log("Removing the trader");
    trader.remove();
  }

  /**
   * JNDI を使って Bean のホーム インタフェースをルックアップする
   */
  private TraderHome lookupTraderHome()
    throws NamingException
  {
    Context ctx = ExampleUtils.getInitialContext(url);
    Object home = (TraderHome)ctx.lookup(JNDI_NAME);
   return (TraderHome)ExampleUtils.narrow(home, TraderHome.class);
  }
}
```

8. LoginContext インスタンスの logout() メソッドを呼び出します。logout() メソッドは、ユーザのセッションをクローズし、Subject をクリアします。logout() メソッドの例については、コード リスト 3-8 を参照してください。

**コード リスト 3-8 logout() メソッドのコード例**

```
...
import javax.security.auth.login.LoginContext;
...
try
    {
      System.out.println("logging out...");
      loginContext.logout();
    }
```

**注意：** LoginModule.logout() メソッドは、WebLogic 認証プロバイダまたはカスタム認証プロバイダでは呼び出されません。Principals を作成し Subject に取得すると、WebLogic セキュリティ フレームワークが Subject のライフサイクルを管理するためです。したがって、JAAS LoginContext を使用してログインし、Subject を取得する開発者によ

るユーザ コードでは、ログアウトする場合に LoginContext を呼び出す必要があります。LoginContext.logout() を呼び出すと、Subject から Principals がクリアされます。

# JNDI 認証の使い方

Java クライアントは、Java Naming and Directory Interface (JNDI) を使用して WebLogic Server に資格を渡します。Java クライアントは、JNDI InitialContext を取得して WebLogic Server との通信を確立します。その後、InitialContext を使用して、WebLogic Server JNDI ツリーで必要なリソースをルックアップします。

**注意：** JAAS は認証方法として好ましい選択肢ですが、WebLogic 認証プロバイダの LoginModule はユーザ名とパスワードの認証しかサポートしていません。したがって、クライアントの証明書認証 ( 相互 SSL 認証とも言う ) では JNDI を使用します。クライアント証明書の認証に JAAS を使用するには、LoginModule で証明書の認証を行うカスタム認証プロバイダを記述する必要があります。LoginModule の記述方法については、http://java.sun.com/j2se/1.4.1/docs/guide/security/jaas/JAASLMDevGuide.html を参照してください。

ユーザとユーザの資格を指定するには、表 3-3 で示されている JNDI プロパティを設定します。

**表 3-3 認証に使用される JNDI プロパティ**

| プロパティ | 意味 |
|---|---|
| INITIAL_CONTEXT_FACTORY | エントリ ポイントを WebLogic Server 環境に提供する。weblogic.jndi.WLInitialContextFactory クラスは WebLogic Server 用の JNDI SPI。 |
| PROVIDER_URL | ネーム サービスを提供する WebLogic Server のホストとポートを指定する。例 : t3://weblogic:7001. |

**表 3-3 認証に使用される JNDI プロパティ ( 続き )**

| プロパティ | 意味 |
|---|---|
| SECURITY_PRINCIPAL | ユーザがデフォルトの ( アクティブな ) セキュリティ レルムに対して認証されているときのユーザの ID を指定する。 |
| SECURITY_CREDENTIALS | ユーザがデフォルトの ( アクティブな ) セキュリティ レルムに対して認証されているときのユーザの資格を指定する。 |

これらのプロパティは、InitialContext コンストラクタに渡されるハッシュテーブルに格納されます。コード リスト 3-9 は、WebLogic Server で実行される Java クライアントでの JNDI 認証の使い方を示します。

**コード リスト 3-9 認証の例**

```
...
Hashtable env = new Hashtable();
     env.put(Context.INITIAL_CONTEXT_FACTORY,
               "weblogic.jndi.WLInitialContextFactory");
     env.put(Context.PROVIDER_URL, "t3://weblogic:7001");
     env.put(Context.SECURITY_PRINCIPAL, "javaclient");
     env.put(Context.SECURITY_CREDENTIALS, "javaclientpassword");
     ctx = new InitialContext(env);
```

JNDI コンテキストとスレッドの詳細と、JNDI コンテキストの問題を回避する方法については、『WebLogic JNDI プログラマーズ ガイド』の「JNDI コンテキストとスレッド」および「JNDI コンテキストの潜在的な問題を回避する方法」を参照してください。

# Java クライアントの JAAS 認証コード例

WebLogic Server 製品には、完全な実践的 JAAS 認証サンプルが付属しています。サンプルは *SAMPLES_HOME*\server\src\examples\security\jaas ディレクトリに置かれています。このサンプルの説明と、構築、コンフィグレーション、および実行の手順については、サンプル ディレクトリの package.html ファイルを参照してください。このコード例は、修正して再利用できます。

# 4 Java クライアントでの SSL 認証の使用

この節では、以下のトピックについて説明します。


- 4-1 ページ「JSSE および WebLogic Server」
- 4-2 ページ「JNDI 認証の使い方」
- 4-4 ページ「SSL 証明書認証の開発環境」
- 4-10 ページ「SSL を利用するアプリケーションの記述」
- 4-37 ページ「SSL クライアントのコード例」


## JSSE および WebLogic Server

JSSE は、SSL と TLS v1 プロトコルをサポートおよび実装し、それらの機能をプログラム的に利用可能にするパッケージのセットです。BEA WebLogic Server は、WebLogic Server クライアントとサーバ、Java クライアント、Web ブラウザ、および他のサーバの間で転送されるデータを暗号化するためにセキュア ソケット レイヤ (SSL) をサポートしています。

WebLogic クライアントでは WebLogic Server の JSSE (Java Secure Socket Extension) の実装を使用できますが、これは必須ではありません。他の JSSE の実装も、サーバの外側のクライアントサイドのコードで同じように使用できます。

WebLogic のサーバサイド アプリケーションで SSL を使用する際には、以下の制限が適用されます。

- WebLogic Server アプリケーションの開発における他の (サード パーティの) JSSE 実装の使用はサポートされていません。WebLogic Server が使用する SSL の実装はサーバのコンフィグレーションに対して固定であり、ユーザの

アプリケーションによって置き換えることはできません。WebLogic Server に他の JSSE 実装を組み込んで、SSL の実装を使用することはできません。

- JSSE の WebLogic 実装は JCE 暗号サービス プロバイダ (Cryptographic Service Providers : CSP) をサポートしていますが、プロバイダの JCE に対するサポートが一定していないため、テストされていないプロバイダがそのまま機能するかどうかは保証できません。nCipher JCE プロバイダでは WebLogic Server をテスト済みです。その他のプロバイダは、WebLogic Server で機能する可能性はありますが、テストされていないプロバイダの場合、おそらくそのままでは機能しません。WebLogic Server での nCipher JCE プロバイダの使用に関する詳細については、『WebLogic Security の管理』の「WebLogic Server での nCipher JCE プロバイダの使い方」を参照してください。

WebLogic Server では、SSL に HTTP ポートを使用します。このポートで使用できるのは SSL のみです。SSL は、ユーザ ID とパスワードがクリア テキストでは転送されないように、クライアントと WebLogic Server との間で転送されるデータを暗号化します。

**注意 :** WebLogic クライアントでセキュリティを実装するためには、Java クライアントで WebLogic Server ソフトウェア配布キットをインストールする必要があります。

# JNDI 認証の使い方

Java クライアントは、Java Naming and Directory Interface (JNDI) を使用して WebLogic Server に資格を渡します。Java クライアントは、JNDI `InitialContext` を取得して WebLogic Server との通信を確立します。その後、`InitialContext` を使用して、WebLogic Server JNDI ツリーで必要なリソースをルックアップします。

**注意 :** JAAS は認証方法として好ましい選択肢ですが、認証プロバイダの LoginModule はユーザ名とパスワードの認証しかサポートしていません。したがって、クライアントの証明書認証 ( 相互 SSL 認証とも言う ) では JNDI を使用します。クライアント証明書の認証に JAAS を使用するには、LoginModule で証明書の認証を行うカスタム認証プロバイダを記述する必要があります。

ユーザとユーザの資格を指定するには、表 4-1 で示されている JNDI プロパティを設定します。

**表 4-1 認証に使用される JNDI プロパティ**

| プロパティ | 意味 |
|---|---|
| INITIAL_CONTEXT_FACTORY | エントリ ポイントを WebLogic Server 環境に提供する。weblogic.jndi.WLInitialContextFactory クラスは WebLogic Server 用の JNDI SPI。 |
| PROVIDER_URL | ネーム サービスを提供する WebLogic Server のホストとポートを指定する。例 : t3s://weblogic:7002. |
| SECURITY_PRINCIPAL | ユーザがデフォルトの ( アクティブな ) セキュリティ レルムに対して認証されているときのユーザの ID を指定する。 |

これらのプロパティは、InitialContext コンストラクタに渡されるハッシュテーブルに格納されます。WebLogic Server 独自の SSL である t3s が使用されていることに注意してください。t3s は、WebLogic Server と Java クライアントの間の接続および通信を暗号化によって保護します。

コード リスト 4-1 は、Java クライアントで一方向 SSL 証明書認証を使用する方法を示しています。双方向 SSL 認証コード例は、4-22 ページのコード リスト 4-5 「JNDI を使用する双方向 SSL 認証クライアントの例」を参照してください。

**コード リスト 4-1 JNDI を使用する一方向 SSL 認証の例**

```
...
Hashtable env = new Hashtable();
      env.put(Context.INITIAL_CONTEXT_FACTORY,
              "weblogic.jndi.WLInitialContextFactory");
      env.put(Context.PROVIDER_URL, "t3s://weblogic:7002");
      env.put(Context.SECURITY_PRINCIPAL, "javaclient");
      env.put(Context.SECURITY_CREDENTIALS, "javaclientpassword");
      ctx = new InitialContext(env);
```

**注意：** JNDI コンテキストとスレッドの詳細と、JNDI コンテキストの問題を回避する方法については、『WebLogic JNDI プログラマーズ ガイド』の「JNDI コンテキストとスレッド」および「JNDI コンテキストの潜在的な問題を回避する方法」を参照してください。

# SSL 証明書認証の開発環境

この節では、以下のトピックについて説明します。


- 4-4 ページ「SSL 認証 API」
- 4-9 ページ「SSL クライアント アプリケーション コンポーネント」


# SSL 認証 API

WebLogic Server で SSL 認証を使用する Java クライアントを実装するには、Java SDK 1.3 アプリケーション プログラミング インタフェース (API) と WebLogic API を組み合わせて使用します。

表 4-2 では、証明書認証の実装に使用される Java SDK API パッケージについて説明します。表 4-2 の情報は Java SDK API のマニュアルから取られており、WebLogic Server 固有の情報がコメントで追加されています。Java SDK API の詳細については、`http://java.sun.com/j2ee/sdk_1.3/techdocs/api/index.html` の Javadoc を参照してください。

表 4-3 では、証明書認証の実装に使用される WebLogic API について説明します。詳細については、WebLogic クラスに関する Javadoc を参照してください。

**表 4-2 Java SDK 証明書 API**

| Java SDK 証明書 API | 説明 |
|---|---|
| javax.crypto | このパッケージは、暗号操作のためのクラスとインタフェースを提供する。このパッケージで定義されている暗号操作には、暗号化、鍵の生成と照合、および Message Authentication Code (MAC) の生成が含まれる。<br>暗号化のサポートには、対称、非対称、ブロック、およびストリームの各暗号化方式が含まれている。このパッケージは、セキュア ストリームおよび暗号化されたオブジェクトもサポートしている。<br>このパッケージで提供されるクラスの多くは、プロバイダベースである (java.security.Provider クラスを参照 )。クラス自体では、アプリケーションを記述できるプログラミング インタフェースが定義されている。独立したサードパーティ ベンダは、必要に応じて、実装自体を作成し、シームレスに組み込むことができる。したがって、アプリケーション開発者は、コードを追加したり書き直したりしなくても、プロバイダベースの実装をいくつでも利用できる。 |
| javax.net | このパッケージは、ネットワーク アプリケーション用のクラスを提供する。ソケット作成用のファクトリが含まれている。ソケット ファクトリを使うと、ソケットの作成とコンフィグレーションの動作をカプセル化できる。 |
| javax.net.SSL | このパッケージのクラスとインタフェースは WebLogic Server でサポートされているが、WebLogic Server で SSL を使用するときには weblogic.security.SSL パッケージの使用が望ましい。 |
| java.security.cert | このパッケージは、証明書、証明書失効リスト (CRL)、および証明書パスの解析と管理を行うためのクラスとインタフェースを提供する。X.509 v3 証明書および X.509 v2 CRL のサポートが含まれている。 |

**表 4-2 Java SDK 証明書 API ( 続き )**

| Java SDK 証明書 API | 説明 |
|---|---|
| `java.security.KeyStore` | このクラスは、鍵と証明書のメモリ内コレクションを表す。以下の 2 タイプのキーストア エントリの管理に使用される。<br>■ キー エントリ<br>このタイプのキーストア エントリは、非常に機密性の高い暗号鍵情報を保持している。これは不正アクセスを防ぐため、保護された形式で格納される。<br>通常、このタイプのエントリに格納された鍵は、対応する公開鍵の証明書チェーンを伴う秘密鍵またはプライベート キーである。<br>プライベート キーおよび証明書チェーンは、自動認証のための指定されたエンティティによって使用される。この認証のためのアプリケーションには、ソフトウェアのリリースやライセンス付与の一環として JAR ファイルに署名するソフトウェア配布組織が含まれる。<br>■ 信頼性のある証明書エントリ<br>このタイプのエントリには、別の相手に属する単一の公開鍵が含まれる。これは信頼性のある証明書と呼ばれる。その証明書内の公開鍵が、実際に証明書の主体 ( オーナ ) によって識別される ID に属するものであるとキーストアのオーナが信頼するからである。<br>このタイプのエントリは、他の相手を認証するのに使用できる。 |
| `java.security.PrivateKey` | プライベート キー。このインタフェースには、メッセージや定数が含まれない。すべてのプライベート キー インタフェースのグループ化 ( およびタイプ保証 ) を行う役割を持つのみである。<br>**注意 :** 専用のプライベート キー インタフェースは、このインタフェースを拡張したものである。たとえば、`java.security.interfaces` の `DSAPrivateKey` インタフェースを参照。 |

**表 4-2 Java SDK 証明書 API ( 続き )**

| Java SDK 証明書 API | 説明 |
|---|---|
| java.security.Provider | このクラスは、Java Security API の「暗号サービス プロバイダ」を表す。プロバイダは、以下のような Java Security の一部または全部を実装する。<br>■ アルゴリズム (DSA、RSA、MD5、SHA-1 など )<br>■ キーの生成、変換、および管理機能 ( アルゴリズム固有キーなど )<br>各プロバイダには名前とバージョン番号が付いており、インストールされる各実行時においてコンフィグレーションされる。<br>暗号サービスの実装を提供するには、開発者チームまたはサードパーティのベンダが実装コードを記述し、Provider クラスのサブクラスを作成する。 |
| javax.servlet.http.HttpServletRequest | このインタフェースは、ServletRequest インタフェースを拡張して、HTTP サーブレットのリクエスト情報を提供する。<br>サーブレット コンテナは、HttpServletRequest オブジェクトを作成し、サーブレットのサービス メソッド (doGet、doPost など ) に引数として渡す。 |
| javax.servlet.http.HttpServletResponse | このインタフェースは ServletResponse インタフェースを拡張して、応答送信の際に HTTP 固有の機能を提供する。たとえば、このインタフェースには HTTP ヘッダーおよびクッキーにアクセスする方法が含まれる。<br>サーブレット コンテナは、HttpServletRequest オブジェクトを作成し、サーブレットのサービス メソッド (doGet、doPost など ) に引数として渡す。 |
| javax.servlet.ServletOutputStream | このクラスは、バイナリ データをクライアントに送信するための出力ストリームを提供する。ServletOutputStream オブジェクトは通常、ServletResponse.getOutputStream() メソッドを通じて取得される。<br>これは、サーブレット コンテナが実装する抽象クラスである。このクラスのサブクラスは、java.io.OutputStream.write(int) メソッドを実装する必要がある。 |

**表 4-2 Java SDK 証明書 API ( 続き )**

| Java SDK 証明書 API | 説明 |
|---|---|
| javax.servlet.<br>ServletResponse | このクラスは、クライアントに応答を送信する際にサーブレットを支援するオブジェクトを定義する。サーブレット コンテナは、ServletResponse オブジェクトを作成し、サーブレットのサービス メソッド (doGet、doPost など ) に引数として渡す。 |

**表 4-3 WebLogic 証明書 API**

| WebLogic 証明書 API | 説明 |
|---|---|
| weblogic.net.http.<br>HTTPsURLConnection | このクラスは、リモートのオブジェクトに Hypertext Transfer Protocol with SSL (HTTPS) を提示するのに使用されるクラスを提供する。このクラスは、別の WebLogic Server に対するクライアントとして機能している WebLogic Server から発信 SSL 接続を確立するために使われる。 |
| weblogic.security.SSL.<br>HostnameVerifierJSSE | このインタフェースは、接続先のホストと証明書のサーバ名 SubjectDN が一致しなければならない場合を扱うポリシーをこのインタフェースの実装側で指定できるように、コールバック メカニズムを提供する。<br>サーバで使用するこのインタフェースのインスタンスを指定するには、Administration Console の [Server Configuration] の [SSL] タブの [SSL.HostName Verifier] フィールドを、このインタフェースを実装するクラスの名前に設定する。 |
| weblogic.security.SSL.<br>TrustManagerJSSE | このインタフェースを使うと、ユーザはピアの証明書チェーンでの何らかの検証エラーをオーバライドし、ハンドシェークを継続できる。また、ピア証明書チェーンに対しさらに検証を実行し、必要に応じてハンドシェークを中断することもできる。 |
| weblogic.security.SSL.<br>SSLContext | このクラスは、このコンテキスト下で作成されたすべてのソケットで共有されるすべての状態情報を保持する。 |
| weblogic.security.SSL.<br>SSLSocketFactory | このクラスは、SSL ソケットを作成するリクエストを、SSLSocketFactory に委託する。 |

# SSL クライアント アプリケーション コンポーネント

最少でも、SSL クライアント アプリケーションは、以下のコンポーネントで構成されています。

- Java クライアント

  通常、Java クライアントは、以下の機能を実行します。

  - クライアント ID、`HostnameVerifierJSSE`、`TrustManagerJSSE`、および `HandshakeCompletedListener` による `SSLContext` の初期化
  - キーストアの作成、およびプライベート キーと認証チェーンの取得
  - `SSLSocketFactory` の使用
  - WebLogic Server のインスタンスによって提供される JSP に接続するための HTTPS の使用

- HostnameVerifier

  HostnameVerifier は、`weblogic.security.SSL.HostnameVerifierJSSE` インタフェースを実装します。これは、接続先のホストと証明書のサーバ名 Subject Distinguished Name (SubjectDN) が一致しなければならない場合を扱うポリシーをこのインタフェースの実装側で指定できるように、コールバック メカニズムを提供します。

- HandshakeCompletedListener

  HandshakeCompletedListener は、`javax.net.ssl.HandshakeCompletedListener` インタフェースを実装します。これは、指定された SSL 接続に対する SSL ハンドシェークの完了について SSL クライアントが通知を受け取る方法を定義します。また、指定された SSL 接続で行われる SSL ハンドシェークの回数を定義します。

- TrustManager

  TrustManager は、`weblogic.security.SSL.TrustManagerJSSE` インタフェースを実装します。これは、信頼されるルートへの証明書のパスを作成し、これが検証可能であり、クライアント SSL 認証について信頼できるものであれば、true を返します。

- `ant` 構築スクリプト (`build.xml`)

このスクリプトは、アプリケーションに必要なすべてのファイルをコンパイルし、それらを WebLogic Server のアプリケーション ディレクトリにデプロイします。

ここで説明しているコンポーネントを実装する完全な実践的 SSL 認証クライアントについては、WebLogic Server で提供されている *SAMPLES_HOME*\server\src\examples\security\sslclient ディレクトリの SSLClient サンプル アプリケーションを参照してください。

JSEE 認証の詳細については、Sun の『Java Secure Socket Extension (JSSE) 1.0.3 API User's Guide』(http://java.sun.com/products/jsse/doc/guide/API_users_guide.html) を参照してください。

# SSL を利用するアプリケーションの記述

この節では以下のトピックについて説明します。


- 4-11 ページ「WebLogic Server 間の安全な通信」
- 4-11 ページ「SSL クライアントの作成」
- 4-20 ページ「双方向 SSL 認証の使用」
- 4-21 ページ「JNDI を使用した双方向 SSL 認証」
- 4-28 ページ「カスタム ホスト名検証の使い方」
- 4-30 ページ「トラスト マネージャの使い方」
- 4-33 ページ「SSLContext の使い方」
- 4-34 ページ「SSL サーバ ソケット ファクトリの使い方」
- 4-35 ページ「URL を使用した発信 SSL 接続」

# WebLogic Server 間の安全な通信

URL オブジェクトを使用すると、別の WebLogic Server インスタンスに対するクライアントとして機能している WebLogic Server インスタンスから発信 SSL 接続を確立することができます。`weblogic.net.http.HttpsURLConnection` クラスを使用して、プライベート キーとデジタル証明書を含む、クライアントのセキュリティ コンテキスト情報を指定できます。

`weblogic.net.http.HttpsURLConnection` クラスは、ネゴシエーションされた暗号スイートの判別、ホスト名検証の取得と設定、サーバの認証チェーンの取得、および新しい SSL ソケットを作成するための `SSLSocketFactory` の取得と設定を行うメソッドを提供します。

SSLClient のコード例では、`weblogic.net.http.HttpsURLConnection` クラスを使用して発信 SSL 接続を確立しています。SSLClient のコード例は、*`SAMPLES_HOME`*`\server\src\examples\security\sslclient` ディレクトリの `examples.security.sslclient` パッケージにあります。

# SSL クライアントの作成

ここでは、さまざまなタイプの SSL クライアントを作成する方法を、サンプルを示して説明します。以下のタイプの SSL クライアントについて説明します。

- 4-12 ページ「SSLClient サンプル」
- 4-16 ページ「SSLSocketClient サンプル」
- 4-19 ページ「SSLClientServlet サンプル」

**SSL クライアントのライセンス要件 :**WebLogic SSL クラス (`weblogic.security.SSL`) を使用してエンタープライズ JavaBean (EJB) を呼び出すスタンドアロン Java クライアントは、BEA ライセンス ファイルを使用する必要があります。クライアント アプリケーションを実行する際、コマンド ラインで次のシステム プロパティを設定します。

- *`bea.home=license_file_directory`*
- `java.protocol.handler.pkgs=com.certicom.net.ssl`

*license_file_directory* は BEA ライセンス ファイル (license.bea) が格納されているディレクトリを表します。以下に、ライセンス ファイルのデフォルトの場所 (c:\bea) を使用する実行コマンドの例を示します。

```
java -Dbea.home=c:\bea \
  -Djava.protocol.handler.pkgs=com.certicom.net.ssl  my_app
```

# SSLClient サンプル

SSLClient サンプルでは、URL オブジェクトおよび URLConnection オブジェクトを使用する発信 SSL 接続を WebLogic SSL ライブラリを使って確立する方法が示されています。スタンドアロンのアプリケーションからだけでなく、WebLogic Server 内のサーブレットからこの処理を行う方法も示されています。

**注意：** 発信 SSL 接続を確立するとき、WebLogic Server インスタンスはそのサーバの証明書を使用します。同じまたは別の WebLogic Server インスタンスと双方向の SSL で通信するときは、着信側の WebLogic Server インスタンスにおいて、発信側サーバの証明書がクライアントのルート CA リストと照合されます。

コード リスト 4-2 はサンプルの SSLClient です。このコードは、*SAMPLES_HOME*\server\src\examples\security\sslclient にある SSLClient.java ファイルからの抜粋です。

**コード リスト 4-2 SSL クライアントのサンプル コード**

```
package examples.security.sslclient;

import java.io.File;
import java.net.URL;
import java.io.IOException;
import java.io.InputStream;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Hashtable;
import java.security.Provider;
import javax.naming.NamingException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletOutputStream;
```

```
import weblogic.net.http.*;
import weblogic.jndi.Environment;

/** SSLClient は、WebLogic の SSL ライブラリを使用して外部への SSL 接続を
 * 行う方法を示す簡単なサンプル。これをスタンドアロン アプリケーションから
 * 行う場合と、WebLogic 内部 ( サーブレット内 ) から行う場合の両方を
 * 示す
 *
 *  WebLogic Server では、外部への SSL 接続を行う際に
 *  そのサーバの証明書のインスタンスを使用することに
 *  注意。双方向 SSL を使って、同じあるいは別の
 *  WebLogic Server と通信する場合、発信側サーバの証明書は、
 *  受信側の WebLogic Server でクライアント ルート CA リスト
 *  に照らし合わせて検証される
 *
 * @author Copyright (c) 1999-2002 by BEA Systems, Inc. All Rights Reserved.
 */

public class SSLClient {
  public void SSLClient() {}
  public static void main (String [] argv)
    throws IOException {
    if (((!((argv.length == 4) || (argv.length == 5))) ||
        (!(argv[0].equals("wls")))
        ) {
      System.out.println("example: java SSLClient wls
                  server2.weblogic.com 80 443 /examplesWebApp/SnoopServlet.jsp");
      System.exit(-1);
    }

    try {
      System.out.println("----");
      if (argv.length == 5) {
        if (argv[0].equals("wls"))
          wlsURLConnect(argv[1], argv[2], argv[3], argv[4], System.out);
      } else {  // クエリが null の場合、デフォルト ページが返される ...
        if (argv[0].equals("wls"))
          wlsURLConnect(argv[1], argv[2], argv[3], null, System.out);
      }
      System.out.println("----");
    } catch (Exception e) {
      e.printStackTrace();
      printSecurityProviders(System.out);
      System.out.println("----");
    }
  }
  private static void printOut(String outstr, OutputStream stream) {
    if (stream instanceof PrintStream) {
```

```
      ((PrintStream)stream).print(outstr);
      return;
    } else if (stream instanceof ServletOutputStream) {
      try {
        ((ServletOutputStream)stream).print(outstr);
        return;
      } catch (IOException ioe) {
        System.out.println(" IOException: "+ioe.getMessage());
      }
    }
    System.out.print(outstr);
  }
  private static void printSecurityProviders(OutputStream stream) {
    StringBuffer outstr = new StringBuffer();
    outstr.append(" JDK Protocol Handlers and Security Providers:\n");
    outstr.append("   java.protocol.handler.pkgs - ");
    outstr.append(System.getProperties().getProperty(
                                        "java.protocol.handler.pkgs"));
    outstr.append("\n");
    Provider[] provs = java.security.Security.getProviders();
    for (int i=0; i<provs.length; i++)
        outstr.append("   provider[" + i + "] - " + provs[i].getName() +
                      " - " + provs[i].getInfo() + "\n");
    outstr.append("\n");
    printOut(outstr.toString(), stream);
  }
  private static void tryConnection(java.net.HttpURLConnection connection,
                                    OutputStream stream)
    throws IOException {
      connection.connect();
      String responseStr = "\t\t" +
                           connection.getResponseCode() + " -- " +
                           connection.getResponseMessage() + "\n\t\t" +
                           connection.getContent().getClass().getName() + "\n";
      connection.disconnect();
      printOut(responseStr, stream);
  }
  /*
   * このメソッドは、URL オブジェクトと URLConnection オブジェクト
   * を使って新しい SSL 接続を作成する方法 (WebLogic SSL クライアント
   * クラスを使用) の例を示す
   */
  public static void wlsURLConnect(String host, String port,
                                   String sport, String query,
                                   OutputStream out) {
    try {
      if (query == null)
        query = "/examplesWebApp/index.jsp";
```

```
    // 以下に示したプロトコル登録は、WebLogic の通常の起動シーケンスで
    //  処理されるものである。ただし、これは、コンソールの SSL パネルを
    //  使って無効にできる
     //
    // ここでは、スタンドアロン Java アプリケーションにおける概念の実証として、
    //  それを再現する。URL オブジェクトを使用した WebLogic 内部での新しい
    //  接続の作成は期待どおりに機能する
     java.util.Properties p = System.getProperties();
     String s = p.getProperty("java.protocol.handler.pkgs");
     if (s == null) {
       s = "weblogic.net";
     } else if (s.indexOf("weblogic.net") == -1) {
       s += "|weblogic.net";
     }
     p.put("java.protocol.handler.pkgs", s);
     System.setProperties(p);
     printSecurityProviders(out);
     // プロトコル登録の終わり
     printOut(" Trying a new HTTP connection using WLS client classes -
               \n\thttp://" + host + ":" + port + query + "\n", out);
     URL wlsUrl = null;
   try {


       wlsUrl = new URL("http", host, Integer.valueOf(port).intValue(), query);
       weblogic.net.http.HttpURLConnection connection =
         new weblogic.net.http.HttpURLConnection(wlsUrl);
       tryConnection(connection, out);
     } catch (Exception e) {
       printOut(e.getMessage(), out);
       e.printStackTrace();
       printSecurityProviders(System.out);
       System.out.println("----");
     }
     printOut(" Trying a new HTTPS connection using WLS client classes -
                \n\thttps://" + host + ":" + sport + query + "\n", out);
     wlsUrl = new URL("https", host, Integer.valueOf(sport).intValue(), query);
     weblogic.net.http.HttpsURLConnection sconnection =
       new weblogic.net.http.HttpsURLConnection(wlsUrl);

     // 双方向 SSL 接続の場合、すなわち [SSL] タブでサーバに
     // ClientCertificateEnforced が選択されている場合にのみ、
     // 以下のプライベート キーとクライアント証明書チェーンが使用される

     File ClientKeyFile  = new File ("clientkey.pem");
     File ClientCertsFile  = new File ("client2certs.pem");
     if (!ClientKeyFile.exists() || !ClientCertsFile.exists())
{
```

```
   System.out.println("Error : clientkey.pem/client2certs.pem
                          is not present in this directory.");
   System.out.println("To create it run - ant createmycerts.");
   System.exit(0);
}
      InputStream [] ins = new InputStream[2];
      ins[0] = new FileInputStream("client2certs.pem");
      ins[1] = new FileInputStream("clientkey.pem");
      String pwd = "clientkey";
      sconnection.loadLocalIdentity(ins[0], ins[1], pwd.toCharArray());
      tryConnection(sconnection, out);
    } catch (Exception ioe) {
      printOut(ioe.getMessage(), out);
      ioe.printStackTrace();
    }
  }
}
```

## SSLSocketClient サンプル

SSLSocketClient サンプルでは、直接セキュア ポートから WebLogic Server のインスタンスによって提供される JSP に接続し、その接続の結果を表示する方法を示します。以下の機能を実装する方法が示されています。

- クライアント ID、`HostnameVerifierJSSE`、および `TrustManagerJSSE` による `SSLContext` の初期化
- キーストアの作成、およびプライベート キーと認証チェーンの取得
- `SSLSocketFactory` の使い方
- WebLogic Server によって提供される JSP に接続するための HTTPS の使い方
- `javax.net.ssl.HandshakeCompletedListener` インタフェースの実装
- サンプルが接続するサーバが目的のホスト上で稼働していることを確認するための、`weblogic.security.SSL.HostnameVerifierJSSE` クラスのダミー実装の作成

コード リスト 4-3 はサンプルの SSLSocketClient です。このコードは、`SAMPLES_HOME\server\src\examples\security\sslclient` にある `SSLSocketClient.java` ファイルからの抜粋です。

**コード リスト 4-3 SSLSocketClient サンプル**

```
package examples.security.sslclient;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.FileInputStream;
import java.util.Hashtable;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.Certificate;
import weblogic.security.SSL.HostnameVerifierJSSE;
import weblogic.security.SSL.SSLContext;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSession;
import weblogic.security.SSL.SSLSocketFactory;
import weblogic.security.SSL.TrustManagerJSSE;

  /**
  * この Java クライアントでは、WebLogic Server から配信される JSP に
  * セキュア ポートを使って接続する方法を示し、接続の結果を表示する
  *
 * @author Copyright (c) 1999-2002 by BEA Systems, Inc. All Rights Reserved.
  */

public class SSLSocketClient {
  public void SSLSocketClient() {}
  public static void main (String [] argv)
    throws IOException {
    if ((argv.length < 2) || (argv.length > 3)) {
      System.out.println("usage:   java SSLSocketClient host sslport
                                      <HostnameVerifierJSSE>");
      System.out.println("example: java SSLSocketClient server2.weblogic.com 443
                                      MyHVClassName");
      System.exit(-1);
    }
      try {
      System.out.println("\nhttps://" + argv[0] + ":" + argv[1]);
      System.out.println(" Creating the SSLContext");
      SSLContext sslCtx = SSLContext.getInstance("https");

      File KeyStoreFile  = new File ("mykeystore");
      if (!KeyStoreFile.exists())
{
   System.out.println("Keystore Error : mykeystore is not present in this
                           directory.");
```

```
   System.out.println("To create it run - ant createmykeystore.");
   System.exit(0);
}

      System.out.println(" Initializing the SSLContext with client\n" +
                         "  identity (certificates and private key),\n" +
                         "  HostnameVerifierJSSE, AND NulledTrustManager");

     // キーストアを開き、プライベート キーと認証チェーンを取得する
      KeyStore ks = KeyStore.getInstance("jks");
      ks.load(new FileInputStream("mykeystore"), null);
      PrivateKey key = (PrivateKey)ks.getKey("mykey", "testkey".toCharArray());
      Certificate [] certChain = ks.getCertificateChain("mykey");
      sslCtx.loadLocalIdentity(certChain, key);

      HostnameVerifierJSSE hVerifier = null;
      if (argv.length < 3)
        hVerifier = new NulledHostnameVerifier();
      else
        hVerifier = (HostnameVerifierJSSE) Class.forName(argv[2]).newInstance();
      sslCtx.setHostnameVerifierJSSE(hVerifier);
      TrustManagerJSSE tManager = new NulledTrustManager();
      sslCtx.setTrustManagerJSSE(tManager);
      System.out.println(" Creating new SSLSocketFactory with SSLContext");
      SSLSocketFactory sslSF = (SSLSocketFactory) sslCtx.getSocketFactoryJSSE();
      System.out.println(" Creating and opening new SSLSocket with
                              SSLSocketFactory");

      // createSocket(String hostname, int port) を使用する
      SSLSocket sslSock = (SSLSocket) sslSF.createSocket(argv[0],
                                            new Integer(argv[1]).intValue());
      System.out.println(" SSLSocket created");
      sslSock.addHandshakeCompletedListener(new MyListener());
      OutputStream out = sslSock.getOutputStream();

      // 簡単な HTTP リクエストを送信する
      String req = "GET /examplesWebApp/ShowDate.jsp HTTP/1.0\r\n\r\n";
      out.write(req.getBytes());

      // InputStream を取得し、HTTP 結果を読み込んで
      // コンソールに表示する
      InputStream    in = sslSock.getInputStream();
      byte buf[] = new byte[1024];
      try
      {
        while (true)
        {
          int amt = in.read(buf);
          if (amt == -1) break;
```

```
          System.out.write(buf, 0, amt);
        }
      }
      catch (IOException e)
      {
        return;
      }
      sslSock.close();
      System.out.println(" SSLSocket closed");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

## SSLClientServlet サンプル

SSLClientServlet サンプルとは、SSLClient サンプルの単純なサーブレット ラッパーです。

コード リスト 4-4 はサンプルの SSLClientServlet です。このコードは、SAMPLES_HOME\server\src\examples\security\sslclient にある SSLSClientServlet.java ファイルからの抜粋です。

**コード リスト 4-4 SSLClientServlet サンプル コード**

```
package examples.security.sslclient;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Enumeration;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * SSLClientServlet は、examples.security.sslclient.SSLClient の
 * 簡単なサーブレット ラッパーである
 *
 * @see SSLClient
 * @author Copyright (c) 1999-2002 by BEA Systems, Inc. All Rights Reserved.
 */
```

```
public class SSLClientServlet extends HttpServlet {
  public void service(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    response.setHeader("Pragma", "no-cache"); // HTTP 1.0
    response.setHeader("Cache-Control", "no-cache"); // HTTP 1.1
    ServletOutputStream out = response.getOutputStream();

    out.println("<br><h2>ssl client test</h2><br><hr>");
    String[] target = request.getParameterValues("url");
    try {

      out.println("<h3>wls ssl client classes</h3><br>");
      out.println("java SSLClient wls localhost 7001 7002
                    /examplesWebApp/SnoopServlet.jsp<br>");
      out.println("<pre>");
      SSLClient.wlsURLConnect("localhost", "7001", "7002",
                                "/examplesWebApp/SnoopServlet.jsp", out);
      out.println("</pre><br><hr><br>");

    } catch (IOException ioe) {
      out.println("<br><pre> "+ioe.getMessage()+"</pre>");
      ioe.printStackTrace();
    }
  }
}
```

# 双方向 SSL 認証の使用

証明書認証では、WebLogic Server は、リクエストを発したクライアントにデジタル証明書を送信します。クライアントは、デジタル証明書を調べて、本物かどうか、期限切れでないかどうか、認証元の WebLogic Server インスタンスに一致しているかどうかを確認します。

双方向 SSL 認証 ( 相互認証の一形態 ) を使用する場合は、リクエスト元のクライアントもデジタル証明書を WebLogic Server に提示します。双方向 SSL 認証をするよう WebLogic Server のインスタンスをコンフィグレーションすると、指定した認証局からデジタル証明書を送信するようにリクエスト元のクライアントに要求できるようになります。WebLogic Server では、指定した信頼性のある認証局のルート証明書によって署名されたデジタル証明書のみが受け付けられます。

双方向 SSL 認証をするための WebLogic Server のコンフィグレーション方法については、『WebLogic Security の管理』の「SSL のコンフィグレーション」を参照してください。

以降の節では、双方向 SSL 認証を WebLogic Server に実装するさまざまな方法について説明します。

- 4-21 ページ「JNDI を使用した双方向 SSL 認証」
- 4-25 ページ「WebLogic Server インスタンス間での双方向 SSL 認証の使用」
- 4-27 ページ「サーブレットでの双方向 SSL 認証の使用」

## JNDI を使用した双方向 SSL 認証

Java クライアントの双方向 SSL 認証で JNDI を使用する場合、WebLogic JNDI `Environment` クラスの `setSSLClientCertificate()` メソッドを使用します。このメソッドは、クライアント認証に対して、X.509 デジタル証明書のプライベート キーと認証チェーンを設定します。

デジタル証明書を JNDI に渡すには、DER エンコードされたデジタル証明書を格納するファイルで開かれている `InputStreams` の配列を作成し、JNDI ハッシュテーブルにその配列を設定します。配列内の最初の要素には、Java クライアントのプライベート キー ファイルで開かれている `InputStream` が格納されている必要があります。配列内の 2 番目の要素には、Java クライアントのデジタル証明書ファイルで開かれている `InputStream` が格納されている必要があります ( このファイルには Java クライアントの公開鍵が含まれています )。追加要素として、ルート認証局のデジタル証明書、認証チェーン内のデジタル証明書の署名者を格納できます。デジタル証明書が WebLogic Server キーストア ファイルの Java クライアントで登録された認証局によって直接発行されない場合、WebLogic Server は、認証チェーンを使用して Java クライアントのそのデジタル証明書を認証できます。

`weblogic.security.PEMInputStream` クラスを使用すると、Privacy Enhanced Mail (PEM) ファイルに保存されているデジタル証明書を読み込むことができます。このクラスでは、ベース 64 でエンコードされた DER 認証を PEM ファイルにデコードするフィルタが提供されます。

コード リスト 4-5 は、Java クライアントで双方向 SSL 認証を使用する方法を示しています。

**コード リスト 4-5 JNDI を使用する双方向 SSL 認証クライアントの例**

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import weblogic.jndi.Environment;
import weblogic.security.PEMInputStream;
import java.io.InputStream;
import java.io.FileInputStream;

public class SSLJNDIClient
{
  public static void main(String[] args) throws Exception
  {

    Context context = null;
    try {
      Environment env = new Environment();

      // 接続パラメータを設定
      env.setProviderUrl("t3s://localhost:7002");
      // 次の 2 つの set メソッドは、UserNameMapper インタフェースを
      // 使用している場合は省略可能
      env.setSecurityPrincipal("system");
      env.setSecurityCredentials("weblogic");

      InputStream key = new FileInputStream("certs/demokey.pem");
     InputStream cert = new FileInputStream("certs/democert.pem");

      // key/cert が pem ファイル内にある場合は入力ストリームをラップ
      key = new PEMInputStream(key);
      cert = new PEMInputStream(cert);

      env.setSSLClientCertificate(new InputStream[] { key, cert});

      env.setInitialContextFactory(Environment.
                  DEFAULT_INITIAL_CONTEXT_FACTORY);
      context = env.getInitialContext();

      Object myEJB = (Object) context.lookup("myEJB");
    }
    finally {
      if (context != null) context.close();
    }
  }
}
```

JNDI の getInitialContext() メソッドが呼び出されると、Java クライアントと WebLogic Server は、Web ブラウザが相互認証を実行してセキュリティ保護された Web サーバ接続を取得するのと同じ方法で相互認証を実行します。デジタル証明書を確認できない場合や Java クライアントのデジタル証明書をデフォル

トの ( アクティブな ) セキュリティ レルムで認証できない場合、例外が送出されます。認証されたユーザ オブジェクトは Java クライアントのサーバ スレッドに格納され、保護された WebLogic リソースへの Java クライアントのアクセスを管理するパーミッションのチェックに使用されます。

WebLogic JNDI `Environment` クラスを使用する場合、`getInitialContext()` メソッドの呼び出しごとに `Environment` オブジェクトを作成する必要があります。`User` オブジェクトとセキュリティ資格を一度指定すると、ユーザおよびユーザに関連する資格は `Environment` オブジェクトの設定に残ります。再設定を試みて JNDI `getInitialContext()` メソッドを呼び出した場合は、元のユーザと資格が使用されます。

Java クライアントによる双方向 SSL 認証を使用する場合、WebLogic Server は、クライアント JVM ごとにユニークな Java 仮想マシン (JVM) ID を取得し、Java クライアントと WebLogic Server の間の接続が切断されないようにします。処理がないために接続がタイムアウトになるまで、Java クライアントの JVM が実行されている間は接続が続行されます。Java クライアントが新しい SSL 接続を確実にネゴシエーションできる唯一の方法は、その JVM を停止して JVM の他のインスタンスを実行することです。

4-22 ページのコード リスト 4-5「JNDI を使用する双方向 SSL 認証クライアントの例」のコードでは、`weblogic.security.providers.authentication.UserNameMapper` インタフェースを実装する ID アサーション プロバイダの呼び出しが生成されます。`UserNameMapper` インタフェースを実装するクラスは、デジタル証明書が有効な場合にユーザ オブジェクトを返します。WebLogic Server では、この認証されたユーザ オブジェクトは WebLogic Server 内の Java クライアントのスレッドに格納されます。格納されたユーザ オブジェクトは、以降の認可リクエストで、スレッドがデフォルトの ( アクティブな ) セキュリティ レルムで保護されている WebLogic リソースにアクセスする際に使用されます。

**注意：** `weblogic.security.providers.authentication.UserNameMapper` インタフェースの実装は `CLASSPATH` に指定されている必要があります。

証明書ベースの認証を実行する ID アサーション プロバイダがコンフィグレーションされていない場合、SSL 接続の JVM で動作している Java クライアントは、新しい JNDI `InitialContext` を作成し、JNDI `SECURITY_PRINCIPAL` と `SECURITY_CREDENTIALS` プロパティで新しいユーザ名とパスワードを指定して WebLogic Server ユーザの ID を変更できます。SSL 接続後に Java クライアント

によって渡されたデジタル証明書は使用されません。新しい WebLogic Server ユーザは、最初のユーザのデジタル証明書でネゴシエーションした SSL 接続を使用し続けます。

証明書ベースの認証を実行する ID アサーション プロバイダがコンフィグレーションされている場合、WebLogic Server はデジタル証明書を Java クライアントから `UserNameMapper` インタフェースを実装するクラスへ渡し、`UserNameMapper` クラスがデジタル証明書を WebLogic Server ユーザ名にマップします。したがって、証明書ベースの ID アサーションを使用していて新しいユーザ ID を設定する場合、その ID は変更できません。この理由は、デジタル証明書は、`Environment` ごとに JVM からの最初の接続要求の時点でのみ処理されるからです。

**注意：** **制限：**双方向 SSL と JNDI を使用している場合、1 つのクライアント JVM から WebLogic Server に同時に複数のユーザがログインすることはできません。異なるスレッドで複数のログインが実行された場合、その結果は確認不能であり、片方のユーザのリクエストが別のユーザのログインで実行される恐れがあり、そうなればユーザが別のユーザのデータにアクセスできることになります。WebLogic Server では、1 つのクライアント JVM からの証明書ベースの複数の同時ログインがサポートされていません。JNDI コンテキストとスレッドの詳細と、JNDI コンテキストの問題を回避する方法については、『WebLogic JNDI プログラマーズ ガイド』の「JNDI コンテキストとスレッド」および「JNDI コンテキストの潜在的な問題を回避する方法」を参照してください。

## ユーザ名マッパーの作成

双方向 SSL を使用する場合、WebLogic Server は、SSL 接続を確立するときに Web ブラウザまたは Java クライアントのデジタル証明書を検証します。ただし、デジタル証明書は Web ブラウザまたは Java クライアントを WebLogic Server セキュリティ レルムのユーザとしては認識しません。Web ブラウザまたは Java クライアントがセキュリティ ポリシーで保護された WebLogic Server リソースをリクエストする場合、WebLogic Server は Web ブラウザまたは Java クライアントにユーザ名とパスワードを指定するように要求します。この要求を処理するために、WebLogic ID アサーション プロバイダでは、Web ブラウザまたは Java クライアントのデジタル証明書を WebLogic Server セキュリティ レルム内のユーザにマップするユーザ名マッパーを使用できます。ユーザ名マッパーは、

weblogic.security.providers.authentication.UserNameMapper インタフェースの実装でなければなりません。

WebLogic ID アサーション プロバイダは、以下の ID アサーション トークン タイプについて、UserNameMapper インタフェースの実装を呼び出すことができます。

- SSL ハンドシェークを通じて渡された X.509 デジタル証明書
- CSIv2 を通じて渡された X.509 デジタル証明書
- CSIv2 を通じて渡された X.501 識別名

その他のタイプの証明書をマップする場合、UserNameMapper インタフェースの独自の実装を記述する必要があります。

デジタル証明書をユーザ名にマップする UserNameMapper インタフェースを実装するには、次の処理を実行する UserNameMapper クラスを記述します。

1. UserNameMapper 実装クラスをインスタンス化します。
2. UserNameMapper インタフェース実装を作成します。
3. mapCertificateToUserName() メソッドを使用して、クライアントから提示された証明書チェーンに基づいて証明書をユーザ名にマップします。
4. 文字列属性タイプを対応する Attribute Value Assertion フィールド タイプにマップします。

## WebLogic Server インスタンス間での双方向 SSL 認証の使用

一方の WebLogic Server インスタンスがもう一方の WebLogic Server インスタンスのクライアントとして機能するサーバ間通信で双方向 SSL 認証を使用できます。サーバ間通信で双方向 SSL 認証を使用すると、一般的なクライアント / サーバ環境でない場合でも、信頼できる高度なセキュリティで保護した接続を利用することができます。

コード リスト 4-6 は、ある WebLogic Server インスタンスで動作するサーブレットから server2.weblogic.com という別の WebLogic Server インスタンスにセキュリティで保護された接続を確立する方法を示しています。

**コード リスト 4-6 他の WebLogic Server インスタンスへのセキュリティ保護された接続の確立**

```
FileInputStream [] f = new FileInputStream[3];
   f[0]= new FileInputStream("demokey.pem");
   f[1]= new FileInputStream("democert.pem");
   f[2]= new FileInputStream("ca.pem");

Environment e = new Environment ();
e.setProviderURL("t3s://server2.weblogic.com:443");
e.setSSLClientCertificate(f);
e.setSSLServerName("server2.weblogic.com");
e.setSSLRootCAFingerprints("ac45e2d1ce492252acc27ee5c345ef26");


e.setInitialContextFactory
       ("weblogic.jndi.WLInitialContextFactory");
Context ctx = new InitialContext(e.getProperties())
```

コード リスト 4-6 では、WebLogic JNDI `Environment` クラスは、以下のパラメータを格納するハッシュ テーブルを作成します。

- `setProviderURL` - SSL サーバとして機能する WebLogic Server インスタンスの URL を指定します。SSL クライアントとして機能する WebLogic Server インスタンスは、このメソッドを呼び出します。URL では、SSL プロトコルを基にした、WebLogic Server 独自のプロトコルである t3s プロトコルを指定します。SSL プロトコルは、2 つの WebLogic Server インスタンス間の接続および通信を保護します。
- `setSSLClientCertificate` - 接続に使用する認証チェーンを指定します。このメソッドを使用して、プライベート キー ( 配列内の最初の入力ストリーム ) と X.509 証明書のチェーン ( 配列内の残りの入力ストリームを構成する ) から成る入力ストリーム配列を指定します。証明書チェーン内の各証明書は、チェーン内の前の証明書の発行元です。
- `setSSLServerName` - SSL サーバとして機能する WebLogic Server インスタンスの名前を指定します。SSL サーバがデジタル証明書を SSL クライアントとして機能するサーバに提示すると、`setSSLServerName` メソッドを使用して指定された名前がデジタル証明書内の一般名と比較されます。ホスト名検証が成功するには、名前が一致する必要があります。このパラメータは、介在者の攻撃を防ぐために使用されます。

- `setSSLRootCAFingerprint` - 信頼性のある認証局のセットを表すデジタル コードを指定します。SSL サーバとして機能する WebLogic Server インスタンスから受け取った証明書チェーンのルート証明書は、このメソッドで指定されたフィンガープリントのいずれかと一致しないと信頼を確立できません。このパラメータは、介在者の攻撃を防ぐために使用されます。

**注意：** JNDI コンテキストとスレッドの詳細と、JNDI コンテキストの問題を回避する方法については、『WebLogic JNDI プログラマーズ ガイド』の「JNDI コンテキストとスレッド」および「JNDI コンテキストの潜在的な問題を回避する方法」を参照してください。

## サーブレットでの双方向 SSL 認証の使用

Java クライアントをサーブレット ( または他のサーバサイド Java クラス ) で認証するには、クライアントがデジタル証明書を提供したかどうかをチェックする必要があります。提供した場合は、証明書が信頼できる認証局で発行されたかどうかをさらにチェックします。サーブレットの開発者には、Java クライアントが有効なデジタル証明書を持っているかどうかを尋ねる役割があります。WebLogic Servlet API でサーブレットを開発する場合、`HTTPServletRequest` オブジェクトの `getAttribute()` メソッドで SSL 接続に関する情報にアクセスする必要があります。

以下の属性が、WebLogic Server サーブレットでサポートされています。

- `javax.servlet.request.X509Certificate` `java.security.cert.X509Certificate []` - X.509 証明書の配列を返します。
- `javax.servlet.request.cipher_suite` - HTTPS が使用する暗号スイートを表す文字列を返します。
- `javax.servlet.request.key_size` - 対称 ( バルク暗号化 ) キー アルゴリズムのビット サイズを表す整数 (0、40、56、128、168) を返します。
- `weblogic.servlet.request.SSLSession` `javax.net.ssl.session` - 暗号スイートを含む SSL セッション オブジェクト、および SSL Session オブジェクトが作成された日付と最後に使用された日付を返します。

デジタル証明書に定義されているユーザ情報にアクセスできます。`javax.servlet.request.X509Certificate` 属性を取得すると、情報は `java.security.cert` X.509 証明書の配列になっています。配列をそれにキャストして証明書を調べるだけで済みます。

デジタル証明書には、以下のような情報が指定されています。

- 主体 ( 保持者、オーナ ) の名前と、デジタル証明書を使用する Web サーバの URL や個人の電子メール アドレスなど、その主体の ID をユニークに確認するために必要なその他の情報
- 主体の公開鍵
- デジタル証明書を発行した認証局の名前
- シリアル番号
- デジタル証明書の有効期間 ( 開始日と終了日で定義 )

# カスタム ホスト名検証の使い方

ホスト名検証を使用すると、SSL 接続先のホストが予定していた通信先、または許可された通信先であることを確認できます。WebLogic クライアントまたは WebLogic Server インスタンスが別のアプリケーション サーバの SSL クライアントとして機能している場合には、介在者の攻撃を防ぐことができて便利です。

WebLogic Server の SSL ハンドシェーク機能としてのデフォルトの動作は、SSL サーバのデジタル証明書の Subject's Distinguished Name (SubjectDN) にある一般名と、SSL 接続の開始に使用する SSL サーバのホスト名を比較することです。これらの名前が一致しない場合は SSL 接続が中断されます。

SSL 接続の中断は、サーバのホスト名をデジタル証明書と照らし合わせて有効性を検証する SSL クライアントによって実行されます。デフォルト以外の動作が必要な場合は、ホスト名検証を無効にするか、カスタム ホスト名検証を登録します。ホスト名検証を無効にすると、SSL 接続は介在者の攻撃に対して無防備な状態になります。

**注意 :** ホスト名検証は、WebLogic Server に付属のデモ用デジタル証明書を使用しているときに無効にします。

ホスト名検証は、以下の方法で無効にできます。

- Administration Console で、サーバ ( たとえば myserver) の [SSL] タブで [ ホスト名検証を無視 ] ボックスをチェックします。
- SSL クライアントのコマンドラインで、次の引数を入力します。

  ```
  -Dweblogic.security.SSL.ignoreHostnameVerification=true
  ```

カスタム ホスト名検証を記述できます。`weblogic.security.SSL.HostnameVerifierJSSE` インタフェースではコールバック メカニズムが提供されるため、接続先のサーバ名がサーバのデジタル証明書の SubjectDN にあるサーバ名と一致しない場合を処理するためのポリシーを定義できます。

カスタム ホスト名検証を使用するには、`weblogic.security.SSL.HostnameVerifierJSSE` インタフェースを実装するクラスを作成し、サーバのセキュリティ ID に関する情報を取得するメソッドを定義します。

**注意：** このインタフェースは新しいスタイルの証明書を受け取り、このリリースの WebLogic Server では非推奨の `weblogic.security.SSL.HostnameVerifier` インタフェースに代わるものです。

カスタム ホスト名検証を使用する前に、以下の方法で実装するクラスを定義する必要があります。

- Administration Console で、サーバ ( たとえば myserver) の [SSL] タブで [ ホスト名の検証 ] フィールドのホスト名検証のクラスを定義します。
- コマンドラインに、次の引数を入力します。

  ```
  -Dweblogic.security.SSL.HostnameVerifierJSSE=hostnameverifier
  ```

  *hostnameverifier* は、カスタム ホスト名検証を実装するクラスの名前です。

カスタム ホスト名検証の例 ( コード リスト 4-7) については、*SAMPLES_HOME*`\server\src\examples\security\sslclient` ディレクトリの SSLclient のコード例を参照してください。このコード例には、比較のために常に true を返す `NulledHostnameVerifier` クラスが含まれています。このサンプルでは、WebLogic SSL クライアントは、サーバのホスト名とデジタル証明書の SubjectDN との比較に関係なく、どの SSL サーバにも接続できます。

**コード リスト 4-7 ホスト名検証のサンプル コード**

```
package examples.security.sslclient;

/**
 * HostnameVerifierJSSE にはコールバック メカニズムが用意されているため、
 * 接続先ホストと証明書の SubjectDN から得られるサーバ名とが一致しなければ
 * ならない場合の処理についてのポリシーを、このインタフェースの実装者が
 * 指定できるようになっている
 *
 * これは、そのクラスの null 版で、重大な問題のない WebLogic SSL
 * クライアント クラスを示す。たとえば、このサンプルでは、
 * クライアント コードは「localhost」のサーバに接続するが、
 * デモ用証明書の SubjectDN CommonName は「bea.com」なので、
 * デフォルトの WebLogic HostnameVerifierJSSE がこれら 2 つの
 * ホスト名に対して String.equals() を実行する
 *
 * @see HostnameVerifier#verify
 * @author Copyright (c) 1999-2002 by BEA Systems, Inc. All Rights
 * Reserved.
 */

public class NulledHostnameVerifier implements
                    weblogic.security.SSL.HostnameVerifierJSSE {
  public boolean verify(String urlHostname, String certHostname) {
    return true;
  }
}
```

# トラスト マネージャの使い方

weblogic.security.SSL.TrustManagerJSSE インタフェースを使用すると、ピアのデジタル証明書内での検証エラーをオーバライドし、SSL ハンドシェークを継続できます。また、サーバのデジタル証明書チェーンで付加的な検証を実行することで、SSL ハンドシェークを中止することもできます。

**注意：** このインタフェースは新しいスタイルの証明書を受け取り、このリリースの WebLogic Server では非推奨の weblogic.security.SSL.TrustManager インタフェースに代わるものです。

SSL クライアントが WebLogic Server のインスタンスに接続すると、サーバは認証のためにデジタル証明書チェーンをクライアントに提示します。提示されたチェーンに無効なデジタル証明書が含まれている場合もあります。SSL 仕様で

は、クライアントが無効な証明書を検出した場合、SSL 接続が中断されることになっています。しかし、Web ブラウザは、無効な証明書を無視するかどうかをユーザに確認し、証明書チェーン内の残りの証明書を使用して SSL サーバを認証できるかどうかを判別するため、チェーンの検証を継続します。TrustManagerJSSE インタフェースを使用すると、どのような場合に SSL 接続を継続するか ( または中止するか ) を制御でき、上記のような矛盾した動作をなくすことができます。トラスト マネージャを使用すると、SSL 接続を続行する前にカスタム検証を実行できます。たとえば、トラスト マネージャを使用して、特定の地域 ( 町、州、国など ) のユーザやその他の特殊な属性を持つユーザだけが SSL 接続を介してアクセスを取得できるように指定することができます。

トラスト マネージャを作成するには、weblogic.security.SSL.TrustManagerJSSE インタフェースを使用します。このインタフェースには、証明書検証で使用する一連のエラー コードが含まれています。また、必要に応じて、ピア証明書での付加的な検証を実行したり、SSL ハンドシェークを中断したりできます。デジタル証明書の検証が済むと、weblogic.security.SSL.TrustManagerJSSE インタフェースがコールバック関数を使用して、デジタル証明書の検証結果をオーバライドします。トラスト マネージャのインスタンスは、setTrustManagerJSSE() メソッドを使用して SSL コンテキストに関連付けることができます。

weblogic.security.SSL.TrustManagerJSSE インタフェースは、JSSE 仕様に準拠しています。トラスト マネージャはプログラムでのみ設定できます。その使用は、Administration Console やコマンドラインでは定義できません。

**注意：** 実行する検証によっては、トラスト マネージャを使用するとパフォーマンスに影響します。

トラスト マネージャの例 ( コード リスト 4-8) については、SAMPLES_HOME\server\src\examples\security\sslclient ディレクトリの SSLClient のコード例を参照してください。

**コード リスト 4-8 TrustManager のコード例**

```
package examples.security.sslclient;

import weblogic.security.SSL.TrustManagerJSSE;
import javax.security.cert.X509Certificate;

public class NulledTrustManagerJSSE implements TrustManagerJSSE{
```

```
  public boolean certificateCallback(X509Certificate[] o, int validateErr) {
   System.out.println(" --- Do Not Use In Production ---\n" + "  By using this " +
                      "NulledTrustManager, the trust in the server's identity "+
                      "is completely lost.\n --------------------------------");
    for (int i=0; i<o.length; i++)
      System.out.println(" certificate " + i + " -- " + o[i].toString());
    return true;
  }
}
```

SSLSocketClient サンプルでは、上で示したカスタム トラスト マネージャを使用します。SSLSocketClient では、トラスト マネージャで SSL コンテキストを使用して新しい SSL 接続を設定する方法が示されています。この例は、*SAMPLES_HOME*\server\src\examples\security\sslclient ディレクトリにもあります。

# ハンドシェーク完了リスナの使い方

javax.net.ssl.HandshakeCompletedListener インタフェースは、指定された SSL 接続に対する SSL ハンドシェークの完了について SSL クライアントが通知を受け取る方法を定義します。また、指定された SSL 接続で行われる SSL ハンドシェークの回数を定義します。コード リスト 4-9 は、HandshakeCompletedListener インタフェースのコード例です。この例は、*SAMPLES_HOME*\server\src\examples\security\sslclient ディレクトリにもあります。

**コード リスト 4-9 HandshakeCompletedListener のコード例**

```
package examples.security.sslclient;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.FileInputStream;
import java.util.Hashtable;
import javax.net.ssl.HandshakeCompletedListener;
import javax.net.ssl.SSLSession;

  public class MyListener implements HandshakeCompletedListener
  {
```

```
    public void handshakeCompleted(javax.net.ssl.
                                    HandshakeCompletedEvent event)
    {
      SSLSession session = event.getSession();
      System.out.println("Handshake Completed with peer " +
                         session.getPeerHost());
     System.out.println("   cipher: " + session.getCipherSuite());
      javax.security.cert.X509Certificate[] certs = null;
      try
      {
        certs = session.getPeerCertificateChain();
      }
      catch (javax.net.ssl.SSLPeerUnverifiedException puv)
      {
        certs = null;
      }
      if  (certs != null)
      {
        System.out.println("   peer certificates:");
        for (int z=0; z<certs.length; z++) System.out.
                println("certs["+z+"]: " + certs[z]);
      }
      else
      {
        System.out.println("No peer certificates presented");
      }
    }
  }
```

# SSLContext の使い方

SSLContext クラスを使用すると、SSL をプログラムによってコンフィグレーションし、SSL セッション情報を保持できます。たとえば、SSLContext クラスが提供するソケット ファクトリによって作成されたすべてのソケットは、SSL コンテキストに関連付けられたハンドシェーク プロトコルを使用することによってセッション ステートを一致させることができます。各インスタンスは、認証の実行に必要なキー、認証チェーン、および信頼されたルート認証局を使用してコンフィグレーションできます。これらのセッションはキャッシュされます。このため、同じ SSL コンテキストで作成された他のソケットは、その後もセッションを場合に応じて再利用できます。セッション キャッシングの詳細については、『WebLogic Security の管理』の「SSL セッションの動作」を参照してください。トラスト マネージャ クラスのインスタンスを SSL コンテキストと関連付けるには、weblogic.security.SSL.SSLContext.setTrustManagerJSSE() メソッドを使用します。

SSLContext は、プログラムでのみ設定できます。Administration Console やコマンドラインでは設定できません。SSLContext オブジェクトは、Java の new 式または SSLContext クラスの getInstance() メソッドで作成できます。getInstance() メソッドは静的で、指定されたセキュア ソケット プロトコルを実装する新しい SSLContext オブジェクトを生成します。SSLContext クラスの使用例については、SAMPLES_HOME\server\src\examples\security\sslclient ディレクトリの SSLSocketClient サンプルを参照してください。このサンプル (4-17 ページのコード リスト 4-3「SSLSocketClient サンプル」) では、SSLContext を使って新しい SSL ソケットを作成する新しい SSL ソケット ファクトリを作成する方法が示されています。

コード リスト 4-10 は、getInstance() メソッドを使用したインスタンス化のサンプルです。

**コード リスト 4-10 SSLContext のコード例**

```
import weblogic.security.SSL.SSLContext;
SSLcontext sslctx = SSLContext.getInstance ("https")
```

# SSL サーバ ソケット ファクトリの使い方

SSLServerSocketFactory クラスのインスタンスは、SSL ソケットを作成して返します。このクラスは、javax.net.SocketFactory を継承します。

コード リスト 4-11 は、このクラスのインスタンス化のサンプルです。4-17 ページのコード リスト 4-3 「SSLSocketClient サンプル」は、SSLServerSocketFactory クラスを使用するコード例です。このコード例は、SAMPLES_HOME\server\src\examples\security\sslclient にある SSLSocketClient.java ファイルからの抜粋です。

**コード リスト 4-11 SSLServerSocketFactory のコード サンプル**

```
import weblogic.security.SSL.SSLSocketFactory;
  SSLSocketFactory sslSF = (SSLSocketFactory) sslCtx.getSocketFactoryJSSE();
```

# URL を使用した発信 SSL 接続

URL オブジェクトを使用すると、別の WebLogic Server インスタンスに対するクライアントとして機能している WebLogic Server インスタンスから発信 SSL 接続を確立することができます。WebLogic Server は、発信 SSL 接続について一方向と双方向の SSL 認証を両方ともサポートしています。

一方向 SSL 認証の場合は、java.net.URL、java.net.URLConnection、および java.net.HTTPURLConnection を使って、URL オブジェクトを使用する発信 SSL 接続を確立します。コード リスト 4-12 は、HTTP および HTTPS の URL を双方ともサポートし、これらの Java クラスのみを使用する ( つまり WebLogic クラスは不要 ) simpleURL クラスを示します。WebLogic Server で一方向 SSL 認証 (HTTPS) に simpleURL クラスを使用するための必要条件は、java.protocols.handler.pkgs のシステム プロパティで「weblogic.net」が定義されていることだけです。

**注意：** コード リスト 4-12 の simpleURL サンプルは、デフォルトで信頼性とホスト名をチェックするため、このサンプルでは信頼されておりホスト名チェックに合格する本当の Web サーバにデフォルトで接続することが必要です。そうしない場合は、コマンド ライン上で信頼性とホスト名チェックをオーバライドする必要があります。

**コード リスト 4-12 Java クラスのみを使用する一方向 SSL 認証 URL 発信 SSL 接続クラス**

```
import java.net.URL;
import java.net.URLConnection;
import java.net.HttpURLConnection;
import java.io.IOException;

public class simpleURL
{
   public static void main (String [] argv)
   {
     if (argv.length != 1)
     {
       System.out.println("Please provide a URL to connect to");
       System.exit(-1);
     }
```

```
      setupHandler();
      connectToURL(argv[0]);
    }

    private static void setupHandler()
    {
      java.util.Properties p = System.getProperties();
      String s = p.getProperty("java.protocol.handler.pkgs");
      if (s == null)
        s = "weblogic.net";
      else if (s.indexOf("weblogic.net") == -1)
        s += "|weblogic.net";
      p.put("java.protocol.handler.pkgs", s);
      System.setProperties(p);
    }

    private static void connectToURL(String theURLSpec)
    {
      try
      {
        URL theURL = new URL(theURLSpec);
        URLConnection urlConnection = theURL.openConnection();
        HttpURLConnection connection = null;
        if (!(urlConnection instanceof HttpURLConnection))
        {
          System.out.println("The URL is not using HTTP/HTTPS: " +
                             theURLSpec);
          return;
        }
        connection = (HttpURLConnection) urlConnection;
        connection.connect();
        String responseStr = "\t\t" +
               connection.getResponseCode() + " -- " +
               connection.getResponseMessage() + "\n\t\t" +
                connection.getContent().getClass().getName() + "\n";
        connection.disconnect();
        System.out.println(responseStr);
      }
      catch (IOException ioe)
      {
       System.out.println("Failure processing URL: " + theURLSpec);
        ioe.printStackTrace();
      }
    }
}
```

双方向 SSL 認証の場合は、`weblogic.net.http.HttpsURLConnection` クラスを使用して、プライベート キーとデジタル証明書を含むクライアントのセキュリティ コンテキスト情報を指定できます。このクラスのインスタンスは、リモート オブジェクトに対する HTTPS 接続を表しています。

SSL クライアントのコード例 ( コード リスト 4-13) には、WebLogic URL オブジェクトを使用して、発信 SSL 接続を確立する方法が示されています。コード リスト 4-13 のコードは、*SAMPLES_HOME*\server\src\examples\security\sslclient ディレクトリの SSLClient.java ファイルからの抜粋です。

**コード リスト 4-13 WebLogic 双方向 SSL 認証 URL 発信 SSL 接続のコード例**

```
wlsUrl = new URL("https", host, Integer.valueOf(sport).intValue(),
                 query);
weblogic.net.http.HttpsURLConnection sconnection =
        new weblogic.net.http.HttpsURLConnection(wlsUrl);

InputStream [] ins = new InputStream[2];
      ins[0] = new FileInputStream("client2certs.pem");
      ins[1] = new FileInputStream("clientkey.pem");
      String pwd = "clientkey";
      sconnection.loadLocalIdentity(ins[0], ins[1],
                                    pwd.toCharArray());
```

# SSL クライアントのコード例

WebLogic Server 製品には、完全な実践的 SSL 認証サンプルが付属しています。サンプルは *SAMPLES_HOME*\server\src\examples\security\sslclient ディレクトリに置かれています。このサンプルの説明と、構築、コンフィグレーション、および実行の手順については、サンプル ディレクトリの package.html ファイルを参照してください。このコード例は、修正して再利用できます。

# 5 エンタープライズ JavaBean (EJB) のセキュリティ対策

WebLogic Server は エンタープライズ JavaBean (EJB) を保護する J2EE セキュリティ モデルをサポートします。これには、宣言による認可 ( このマニュアルでは宣言によるセキュリティと呼ぶ ) およびプログラムによる認可 ( このマニュアルではプログラムによるセキュリティと呼ぶ ) のサポートが含まれます。

この節では、以下のトピックについて説明します。


- 5-1 ページ「J2EE セキュリティ モデル」
- 5-5 ページ「EJB での宣言によるセキュリティの使用」
- 5-7 ページ「EJB のセキュリティ関連のデプロイメント記述子」
- 5-26 ページ「EJB でのプログラムによるセキュリティの使用」


**注意：** EJB の保護には、デプロイメント記述子ファイルと Administration Console を使用できます。Administration Console を使用しての EJB の保護については、「WebLogic リソースのセキュリティ」を参照してください。

## J2EE セキュリティ モデル

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3 Authorization」に、以下のような記述があります。

> 「J2EE アーキテクチャにおいて、コンテナはホストするコンポーネントとその呼び出し側との間の認可境界として機能します。認可境界は、コンテナの認証境界内に存在するので、認可は正常に実行される認証との関連で考慮されます。着信呼び出しの場合、コンテナは呼び出し側の資格のセキュリティ属性と、対象コンポーネントのアクセス制御ルールを比較します。ルール要

件が満たされていれば、呼び出しは許可されます。それ以外の場合、この呼び出しは拒否されます。」

「アクセス制御ルールの定義には、能力とパーミッションという 2 つの基本的アプローチが存在します。能力は呼び出し側が実行できる操作、パーミッションは操作を実行できるユーザを焦点とします。J2EE アプリケーション プログラミング モデルは、パーミッションを焦点とします。J2EE アーキテクチャにおいて、デプロイヤの役割はアプリケーションのパーミッション モデルをその操作環境におけるユーザの能力にマップすることです。」

次にこのマニュアルでは、J2EE アーキテクチャ、宣言による認可、およびプログラムによる認可を使用してアプリケーション リソースへのアクセスを制御する 2 つの方法について説明します。

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html` からオンラインで入手できます。

# 宣言による認可

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3.1 Authorization」に、以下のような記述があります。

「デプロイヤは、J2EE アプリケーションと関連のコンテナによって実施されるアクセス制御ルールを設定します。デプロイヤは、デプロイメント ツールを使用して、通常はアプリケーション アセンブラによって供給されるアプリケーション パーミッション モデルを、操作環境に固有のポリシーおよびメカニズムにマップします。アプリケーション パーミッション モデルは、デプロイメント記述子で定義されます。」

WebLogic Server は EJB において宣言による認可を実装するためのデプロイメント記述子の使用をサポートしています。

**注意：** 宣言による認可は、このマニュアルでは宣言によるセキュリティとも呼ばれます。

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html` からオンラインで入手できます。

# プログラムによる認可

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3.2 Programmatic Authorization」に、以下のような記述があります。

「J2EE コンテナは、コンポーネントにメソッド呼び出しをディスパッチする前に、アクセス制御の決定を行います。コンポーネントの論理または状態は、これらのアクセス決定を考慮に入れません。しかし、コンポーネントは `EJBContext.isCallerInRole` ( エンタープライズ Bean コード用 ) および `HttpServletRequest.isUserInRole` (Web コンポーネント用 ) の 2 つのメソッドを使用して、きめ細かいアクセス制御を行うことが可能です。コンポーネントはこれらのメソッドを使用して、呼び出しのパラメータ、コンポーネントの初期状態、または呼び出しの時間などその他の要因に基づきコンポーネントによって選択された特権が、呼び出し側に付与されているかどうかを判断します。」

「これらの機能の 1 つを呼び出すコンポーネントのアプリケーション コンポーネント プロバイダは、すべての呼び出しで使用されるあらゆる個別の roleName 値を宣言する必要があります。これらの宣言は、デプロイメント記述子では `security-role-ref` 要素として登場します。各 `security-role-ref` 要素は、アプリケーションに roleName として組み込まれた特権名をセキュリティ ロールにリンクします。最終的には、デプロイヤはアプリケーションに組み込まれた特権名と、デプロイメント記述子で定義されたセキュリティ ロールの間のリンクを確立します。特権名とセキュリティ ロールの間のリンクは、同じアプリケーション内でもコンポーネントごとに異なる場合があります。」

「特定の特権をテストするだけでなく、アプリケーション コンポーネントでは `EJBContext.getCallerPrincipal` または `HttpServletRequest.getUserPrincipal` を使用して取得した呼び出し側 ID と、作成時のコンポーネントの状態に組み込まれている識別された呼び出し側 ID を比較できます。呼び出し側 ID が識別された呼び出し側のものに等しければ、コンポーネントは呼び出し側に処理の続行を許可できます。等しくなければ、コンポーネントは呼び出し側がそれ以上の対話を行わないよう

にできます。コンテナによって返された呼び出し側のプリンシパルは、呼び出し側が使用した認証メカニズムによって決まります。また、同じメカニズムによる同じユーザ認証に対してでも、異なったベンダからのコンテナは異なったプリンシパルを返す場合があります。プリンシパルの形式の変動性を考慮に入れるため、コンポーネントのアクセス決定に識別された呼び出し側の状態を適用することを選択した開発者は、同一ユーザを表す複数の識別された呼び出し側 ID がコンポーネントと関連付けられることを許容する必要があります。これは特に、アプリケーションの融通性や移植性が優先される場合に、推奨されます。」

WebLogic Server は、EJB におけるプログラムによる認可を実装するための `EJBContext.isCallerInRole` メソッドと `EJBContext.getCallerPrincipal` メソッドの使用、および `security-role-ref` 要素の使用をサポートしています。

**注意：** プログラムによる認可は、このマニュアルではプログラムによるセキュリティとも呼ばれます。

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、`http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html` からオンラインで入手できます。

# 宣言による認可とプログラムによる認可

Sun Microsystems, Inc. のマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』の「Section 9.3.3 Declarative Versus Programmatic Authorization」に、以下のような記述があります。

「デプロイヤによってコンフィグレーションされる外部アクセス制御ポリシーと、コンポーネント プロバイダによってアプリケーションに組み込まれた内部ポリシーの間にはトレードオフが存在します。アプリケーションが作成された後では、外部ポリシーのほうが高い融通性を持ちます。アプリケーションが記述されている過程では、内部ポリシーのほうが融通性の高い機能を提供します。また、外部ポリシーはデプロイヤにとって透過的で完全に理解可能であるのに対し、内部ポリシーはアプリケーションに埋め込まれており、これを完全に理解できるのはアプリケーション開発者のみである可能性があります。ある特定のコンポーネントとメソッドについて認可モデルを選択する際には、これらのトレードオフを検討する必要があります。」

Sun Microsystems, Inc. の発行によるマニュアル『Designing Enterprise Applications with the J2EE Platform, Second Edition』は、http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/security/security4.html からオンラインで入手できます。

# EJB での宣言によるセキュリティの使用

EJB への宣言によってセキュリティを実装するには、デプロイメント記述子 (ejb-jar.xml および weblogic-ejb-jar.xml) を使用してセキュリティ要件を定義します。コード リスト 5-1 に、ejb-jar.xml および weblogic-ejb-jar.xml デプロイメント記述子を使用してセキュリティ ロール名をセキュリティ レルムにマップする例を示します。デプロイメント記述子は、アプリケーションの論理的なセキュリティ要件を実行時の定義にマップします。また、実行時には、EJB コンテナがセキュリティ定義を使って要件を実施します。

EJB デプロイメント記述子でセキュリティをコンフィグレーションするには、次の手順を実行します ( コード リスト 5-1 を参照 )。

1. テキスト エディタを使用して、ejb-jar.xml および weblogic-ejb-jar.xml デプロイメント記述子ファイルを作成します。
2. ejb-jar.xml ファイルに、セキュリティ ロール名、EJB 名、およびメソッド名を定義します ( 太字テキストを参照 )。

   **注意：** セキュリティ ロール名を指定する場合、以下の規約と制限に従ってください。

   - セキュリティ ロール名の適切な構文は、Web 上 (http://www.w3.org/TR/REC-xml#NT-Nmtoken) で閲覧可能な XML (Extensible Markup Language) 勧告で Nmtoken に関して定義されているとおりです。
   - スペース、カンマ、ハイフン、\t、< >、#、|、&、~、?、( )、{ } を使用しないでください。
   - セキュリティ ロール名では大文字 / 小文字を区別します。
   - BEA が提唱する命名規約では、セキュリティ ロール名は単数形です。

ejb-jar.xml ファイルでセキュリティをコンフィグレーションする方法の詳細については、Sun Microsystems のエンタープライズ JavaBeans 仕様 2.0 (http://java.sun.com/products/ejb/docs.html) を参照してください。

3. WebLogic 固有の EJB デプロイメント記述子ファイル weblogic-ejb-jar.xml にセキュリティ ロール名を定義し、それをセキュリティ レルム内の 1 つまたは複数のプリンシパル ( ユーザまたはグループ ) にリンクします。

   weblogic-ejb-jar.xml ファイルでセキュリティをコンフィグレーションする方法の詳細については、『WebLogic エンタープライズ JavaBeans プログラマーズ ガイド』の「weblogic-ejb-jar.xml デプロイメント 記述子要素」を参照してください。

**コード リスト 5-1　ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング**

**ejb-jar.xml** のエントリ **:**

```
          ...
<assembly-descriptor>
  <security-role>
        <role-name>manger</role-name>
  </security-role>
  <security-role>
        <role-name>east</role-name>
  </security-role>
  <method-permission>
        <role-name>manager</role-name>
        <role-name>east</role-name>
        <method>
          <ejb-name>accountsPayable</ejb-name>
          <method-name>getReceipts</method-name>
        </method>
  </method-permission>
  <method-permission>
        <unchecked/>
        <method>
          <ejb-name>vacationAccrued</ejb-name>
          <method-name>uncheckedGetCompanyTotal</method-name>
        </method>
  </method-permission>
          ...
</assembly-descriptor>
           ...
```

**weblogic-ejb-jar.xml** のエントリ **:**

```
<security-role-assignment>
    <role-name>manager</role-name>
    <principal-name>al</principal-name>
    <principal-name>george</principal-name>
    <principal-name>ralph</principal-name>
</security-role-assignment>
   ...
```

# EJB のセキュリティ関連のデプロイメント記述子

以下のトピックでは、EJB のセキュリティ要件を定義するために ejb-jar.xml および weblogic-ejb-jar.xml ファイルで使用されるデプロイメント記述子の要素について説明します。


- 5-7 ページ「ejb-jar.xml デプロイメント記述子」
- 5-14 ページ「weblogic-ejb-jar.xml デプロイメント記述子」


## ejb-jar.xml デプロイメント記述子

以下の ejb-jar.xml のデプロイメント記述子の要素は、WebLogic Server でセキュリティ要件を定義するために使用されます。


- 5-8 ページ「method」
- 5-9 ページ「method-permission」
- 5-10 ページ「role-name」
- 5-10 ページ「run-as」
- 5-11 ページ「security-identity」
- 5-11 ページ「security-role」
- 5-12 ページ「security-role-ref」

- 5-13 ページ「unchecked」
- 5-13 ページ「use-caller-identity」

この節の情報は、Sun Microsystems, Inc. 提供の `ejb-jar.xml` の文書型記述子 (DTD) に基づいています。`ejb-jar.xml` の DTD は、`http://java.sun.com/dtd/ejb-jar_2_0.dtd` にあります。

# method

`method` 要素は、エンタープライズ Bean のホームまたはコンポーネント インタフェースのメソッド、あるいはメッセージ駆動型 Bean の場合に Bean の onMessage メソッド ( またはメソッドのセット ) を示すために使用されます。

次の表では、`method` 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| `<description>` | 省略可能 | メソッドの説明文。 |
| `<ejb-name>` | 必須 | `ejb-jar.xml` ファイルで宣言されているエンタープライズ Bean のいずれかの名前を指定する。 |
| `<method-intf>` | 省略可能 | エンタープライズ Bean のホーム インタフェースとコンポーネント インタフェースの両方で定義された同じ署名を持つメソッドを識別できるようにする。 |
| `<method-name>` | 必須 | エンタープライズ Bean のメソッドの名前またはアスタリスク (*) を指定する。アスタリスクは、要素がエンタープライズ Bean のコンポーネントおよびホーム インタフェースのすべてのメソッドを表す場合に使用する。 |
| `<method-params>` | 省略可能 | Java タイプのメソッド パラメータの完全修飾名のリストが含まれる。 |

## 使用する場所

`method` 要素は、`method-permission` 要素内で使用されます。

### 例

method 要素の使用例については、5-6 ページのコード リスト 5-1「ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング」を参照してください。

## method-permission

method-permission 要素では、1 つまたは複数のエンタープライズ Bean メソッドの呼び出しを許可されている 1 つまたは複数のセキュリティ ロールを指定します。method-permission 要素は、説明 ( 省略可能 )、セキュリティ ロール名のリストまたはメソッドが認可に関してチェックされないことを示すインジケータ、およびメソッドの要素のリストから成ります。

method-permission 要素内のセキュリティ ロールは、デプロイメント記述子の security-role 要素で定義されており、メソッドは、エンタープライズ Bean のコンポーネントまたはホーム インタフェースで定義されているメソッドでなければなりません。

次の表では、method-permission 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <description> | 省略可能 | このセキュリティ制約の説明文。 |
| <role-name> または <unchecked> | 必須 | role-name 要素または unchecked 要素を指定する必要がある。<br>■ role-name 要素には、セキュリティ ロールの名前が含まれる。この名前は、NMTOKEN の命名規則に準拠しなければならない。<br>■ unchecked 要素には、メソッドがメソッド呼び出しの前にコンテナによる認可のチェックを受けないことを指定する。 |
| <method> | 必須 | エンタープライズ Bean のホームまたはコンポーネント インタフェースのメソッド、あるいはメッセージ駆動型 Bean の場合に Bean の onMessage メソッド ( またはメソッドのセット ) を指定する。 |

### 使用する場所

method-permission 要素は、assembly-descriptor 要素内で使用されます。

### 例

method-permission 要素の使用例については、5-6 ページのコード リスト 5-1「ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング」を参照してください。

## role-name

role-name 要素には、セキュリティ ロールの名前が含まれます。この名前は、NMTOKEN の命名規則に準拠しなければなりません。

### 使用する場所

role-name 要素は、method-permission、run-as、security-role、および security-role-ref 要素内で使用されます。

### 例

role-name 要素の使用例については、5-23 ページのコード リスト 5-7「run-as-identity-principal 要素の例」を参照してください。

## run-as

run-as 要素には、エンタープライズ Bean を実行するための run-as ID を指定します。この要素には、省略可能な説明とセキュリティ ロールの名前が含まれます。

### 使用する場所

run-as 要素は、security-identity 要素内で使用されます。

### 例

run-as 要素の使用例については、5-23 ページのコード リスト 5-7「run-as-identity-principal 要素の例」を参照してください。

# security-identity

security-identity 要素には、エンタープライズ Bean のメソッドを実行するために呼び出し側のセキュリティ ID を使用するか、または特定の run-as ID を使用するかを指定します。この要素には、省略可能な説明と使用するセキュリティ ID の指定が含まれます。

次の表では、security-identity 要素内で定義できる要素について説明します。

| 要素 | 必須 / 省略可能 | 説明 |
|---|---|---|
| <description> | 省略可能 | セキュリティ ID の説明文。 |
| <use-caller-identity> または <run-as> | 必須 | use-caller-identity 要素または run-as 要素を指定しなければならない。<br>■ use-caller-identity 要素には、エンタープライズ Bean のメソッドを実行するためのセキュリティ ID として、呼び出し側のセキュリティ ID を使用することを指定する。<br>■ run-as 要素には、エンタープライズ Bean を実行するための run-as ID を指定する。この要素には、省略可能な説明とセキュリティ ロールの名前が含まれる。 |

## 使用する場所

security-identity 要素は、entity、message-driven、および session 要素内で使用されます。

## 例

security-identity 要素の使用例については、5-14 ページのコード リスト 5-3「use-caller-identity 要素の例」と5-23 ページのコード リスト 5-7「run-as-identity-principal 要素の例」を参照してください。

# security-role

security-role 要素には、セキュリティ ロールの定義が指定されます。定義は、セキュリティ ロールの説明 ( 省略可能 ) とセキュリティ ロール名から成ります。

### 使用する場所

security-role 要素は、assembly-descriptor 要素内で使用されます。

### 例

assembly-descriptor 要素の使用例については、5-6 ページのコード リスト 5-1「ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング」を参照してください。

## security-role-ref

security-role-ref 要素には、エンタープライズ Bean のコード内のセキュリティ ロール参照の宣言が含まれます。この宣言は、省略可能な説明、コードで使用されているセキュリティ ロール名、およびセキュリティ ロールへのリンク ( 省略可能 ) から成ります。セキュリティ ロールが指定されていない場合、デプロイヤが適切なセキュリティ ロールを選択する必要があります。

role-name 要素の値は、EJBContext.isCallerInRole(String roleName) メソッドまたは HttpServletRequest.isUserInRole(String role) メソッドに対するパラメータとして使用される String でなければなりません。

### 使用する場所

security-role-ref 要素は、entity および session 要素内で使用されます。

### 例

security-role-ref 要素の使用例については、コード リスト 5-2 を参照してください。

**コード リスト 5-2　security-role-ref 要素の例**

```
<!DOCTYPE ejb-jar PUBLIC '-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 2.0//EN' 'http://java.sun.com/dtd/ejb-jar_2_0.dtd'>

<ejb-jar>
 <enterprise-beans>
   ...
   <session>
     <ejb-name>SecuritySLEJB</ejb-name>
     <home>weblogic.ejb20.security.SecuritySLHome</home>
```

```
      <remote>weblogic.ejb20.security.SecuritySL</remote>
      <ejb-class>weblogic.ejb20.security.SecuritySLBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <security-role-ref>
         <role-name>rolenamedifffromlink</role-name>
         <role-link>role121SL</role-link>
      </security-role-ref>
      <security-role-ref>
         <role-name>roleForRemotes</role-name>
         <role-link>roleForRemotes</role-link>
      </security-role-ref>
      <security-role-ref>
         <role-name>roleForLocalAndRemote</role-name>
         <role-link>roleForLocalAndRemote</role-link>
      </security-role-ref>
    </session>
     ...
  </enterprise-beans>
</ejb-jar>
```

# unchecked

unchecked 要素には、メソッドがメソッド呼び出しの前にコンテナによる認可のチェックを受けないことを指定します。

## 使用する場所

unchecked 要素は、method-permission 要素内で使用されます。

## 例

unchecked 要素の使用例については、5-6 ページのコード リスト 5-1「ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング」を参照してください。

# use-caller-identity

use-caller-identity 要素には、エンタープライズ Bean のメソッドを実行するためのセキュリティ ID として、呼び出し側のセキュリティ ID を使用することを指定します。

### 使用する場所

use-caller-identity 要素は、security-identity 要素内で使用されます。

### 例

use-caller-identity 要素の使用例については、コード リスト 5-3 を参照してください。

**コード リスト 5-3　use-caller-identity 要素の例**

```
<ejb-jar>
  <enterprise-beans>

    <session>
      <ejb-name>SecurityEJB</ejb-name>
      <home>weblogic.ejb20.SecuritySLHome</home>
      <remote>weblogic.ejb20.SecuritySL</remote>
      <local-home>
          weblogic.ejb20.SecurityLocalSLHome
      </local-home>
      <local>weblogic.ejb20.SecurityLocalSL</local>
      <ejb-class>weblogic.ejb20.SecuritySLBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
    </session>

    <message-driven>
      <ejb-name>SecurityEJB</ejb-name>
      <ejb-class>weblogic.ejb20.SecuritySLBean</ejb-class>
      <transaction-type>Container</transaction-type>
      <security-identity>
         <use-caller-identity/>
      </security-identity>
    </message-driven>

  </enterprise-beans>
</ejb-jar>
```

# weblogic-ejb-jar.xml デプロイメント記述子

以下の weblogic-ejb-jar.xml デプロイメント記述子の要素は、WebLogic Server でセキュリティ要件を定義するために使用されます。

- 5-15 ページ「client-authentication」

- 5-16 ページ「client-cert-authentication」
- 5-16 ページ「confidentiality」
- 5-17 ページ「global-role」
- 5-19 ページ「identity-assertion」
- 5-20 ページ「iiop-security-descriptor」
- 5-21 ページ「integrity」
- 5-21 ページ「principal-name」
- 5-22 ページ「role-name」
- 5-22 ページ「run-as-identity-principal」
- 5-24 ページ「security-permission」
- 5-24 ページ「security-permission-spec」
- 5-25 ページ「security-role-assignment」
- 5-26 ページ「transport-requirements」


# client-authentication

`client-authentication` 要素は、EJB がクライアント認証をサポートするか、または、必要とするかを指定します。

次の表に、指定可能な設定を定義してあります。

| 設定 | 定義 |
|---|---|
| なし | クライアント認証はサポートされない。 |
| `Supported` | クライアント認証はサポートされるが、必須ではない。 |
| `Required` | クライアント認証は必須となる。 |

### 例

`client-authentication` 要素の使用例については、5-20 ページのコード リスト 5-6 「iiop-security-descriptor 要素の例」 を参照してください。

## client-cert-authentication

`client-cert-authentication` 要素は、EJB が転送レベルでのクライアント証明書認証をサポートするか、または、必要とするかを指定します。

次の表に、指定可能な設定を定義してあります。

| 設定 | 定義 |
|---|---|
| なし | クライアント証明書認証はサポートされない。 |
| `Supported` | クライアント証明書認証はサポートされるが、必須ではない。 |
| `Required` | クライアント証明書認証は必須となる。 |

### 例

`client-cert-authentication` 要素の使用例については、5-26 ページのコード リスト 5-9 「transport-requirements 要素の例」 を参照してください。

## confidentiality

`confidentiality` 要素は、その EJB における転送の機密性の要件を指定します。`confidentiality` 要素を使用すると、他のエンティティに内容を見られることなく、クライアントとサーバ間でデータが送信されることが保証されます。

次の表に、指定可能な設定を定義してあります。

| 設定 | 定義 |
|---|---|
| なし | 機密性はサポートされない。 |
| `Supported` | 機密性はサポートされるが、必須ではない。 |
| `Required` | 機密性は必須となる。 |

### 例

confidentiality 要素の使用例については、5-26 ページのコード リスト 5-9「transport-requirements 要素の例」を参照してください。

## global-role

WebLogic Server 7.0 SP1 以降では、global-role 要素の weblogic-ejb-jar.xml デプロイメント記述子での使用がサポートされています。principal-name 要素の代わりにこの要素を使用すると、role-name 要素によってデプロイメント記述子に定義された weblogic-ejb-jar.xml ファイル内のセキュリティ ロールが Administration Console で指定したマッピングを使用するよう指定できます。

global-role 要素を使用すると、特定の EJB のデプロイメント記述子に定義されたセキュリティ ロールごとに特定のセキュリティ ロール マッピングを指定する必要がなくなります。代わりに、Administration Console を使用して定義済みの各ロールに対する特定のロール マッピングをいつでも指定および変更できます。さらに、この要素は一部の EJB に対して使用できるので、セキュリティ レルムの [ 一般 ] タブの [ デプロイメント記述子内のセキュリティ データを無視 ] 属性を有効にする必要がありません。このため、同じセキュリティ レルムの中で、デプロイメント記述子を使用して一部の EJB のセキュリティを指定および変更する一方、Administration Console を使用して他のアプリケーションのセキュリティを指定および変更できます。

**注意：** セキュリティ ロール名を指定する場合、以下の規約と制限に従ってください。

- セキュリティ ロール名の適切な構文は、Web 上 (http://www.w3.org/TR/REC-xml#NT-Nmtoken) で閲覧可能な XML (Extensible Markup Language) 勧告で Nmtoken に関して定義されているとおりです。
- スペース、カンマ、ハイフン、\t、< >、#、|、&、~、?、( )、{ } を使用しないでください。
- セキュリティ ロール名では大文字 / 小文字を区別します。
- BEA が提唱する命名規約では、セキュリティ ロール名は単数形です。

## 使用する場所

global-role 要素は、security-role-assignment 要素内で使用されます。

## 例

コード リスト 5-4 と コード リスト 5-5 は、weblogic-ejb-jar.xml デプロイメント記述子での global-role 要素の使い方を比較により示しています。コード リスト 5-5 では、weblogic-ejb-jar.xml 内の「manager」の global-role 要素は、セキュリティが getReceipts メソッドにおいて正しくコンフィグレーションされるためには、Administration Console で manager に対応するプリンシパルが作成される必要があることを意味します。

**コード リスト 5-4　ejb-jar.xml および weblogic-ejb-jar.xml デプロイメント記述子を使用しての EJB におけるセキュリティ ロールのマッピング**

**ejb-jar.xml** のエントリ **：**

```
          ...
<assembly-descriptor>
  <security-role>
        <role-name>manger</role-name>
  </security-role>
  <security-role>
        <role-name>east</role-name>
  </security-role>
  <method-permission>
        <role-name>manager</role-name>
        <role-name>east</role-name>
        <method>
          <ejb-name>accountsPayable</ejb-name>
          <method-name>getReceipts</method-name>
        </method>
  </method-permission>
          ...
</assembly-descriptor>
           ...
```

**weblogic-ejb-jar.xml** のエントリ **：**

```
  <security-role-assignment>
      <role-name>manager</role-name>
      <principal-name>joe</principal-name>
      <principal-name>Bill</principal-name>
      <principal-name>Mary</principal-name>
      ...
  </security-role-assignment>
    ...
```

**コード リスト 5-5 EJB デプロイメント記述子におけるロール マッピング用の <global-role/> タグの使用**

**ejb-jar.xml** のエントリ **:**

```
          ...
<assembly-descriptor>
  <security-role>
        <role-name>manger</role-name>
  </security-role>
  <security-role>
        <role-name>east</role-name>
  </security-role>
  <method-permission>
        <role-name>manager</role-name>
        <role-name>east</role-name>
        <method>
          <ejb-name>accountsPayable</ejb-name>
          <method-name>getReceipts</method-name>
        </method>
  </method-permission>
          ...
</assembly-descriptor>
           ...
```

**weblogic-ejb-jar.xml** のエントリ **:**

```
  <security-role-assignment>
      <role-name>manager</role-name>
      <global-role/>
      ...
  </security-role-assignment>
   ...
```

Administration Console を使用しての EJB のセキュリティのコンフィグレーションの詳細については、『WebLogic リソースのセキュリティ』を参照してください。

# identity-assertion

identity-assertion 要素は、EJB が ID アサーションをサポートするかどうかを指定します。

次の表に、指定可能な設定を定義してあります。

| 設定 | 定義 |
|---|---|
| なし | ID アサーションはサポートされない。 |
| Supported | ID アサーションはサポートされるが、必須ではない。 |
| Required | ID アサーションは必須となる。 |

### 使用する場所

identity-assertion 要素は、iiop-security-descriptor 要素内で使用されます。

### 例

identity-assertion 要素の使用例については、5-20 ページのコード リスト 5-6「iiop-security-descriptor 要素の例」を参照してください。

## iiop-security-descriptor

iiop-security-descriptor 要素は、Bean レベルのセキュリティ コンフィグレーション パラメータを指定します。これらのパラメータにより、インターオペラブル オブジェクト参照 (IOR) に含まれる IIOP セキュリティ情報が決定します。

### 例

iiop-security-descriptor 要素の使用例については、コード リスト 5-6 を参照してください。

**コード リスト 5-6　iiop-security-descriptor 要素の例**

```
<weblogic-enterprise-bean>
  <iiop-security-descriptor>
       <transport-requirements>
              <confidentiality>supported</confidentiality>
              <integrity>supported</integrity>
              <client-cert-authorization>
                  supported
```

```
            </client-cert-authentication>
        </transport-requirements>
        <client-authentication>supported<client-authentication>
        <identity-assertion>supported</identity-assertion>
  </iiop-security-descriptor>
</weblogic-enterprise-bean>
```

# integrity

integrity 要素は、EJB の転送の整合性の要件を指定します。integrity 要素を使用すると、クライアントとサーバ間で、データが途中で変化することなく転送されることが保証されます。

次の表に、指定可能な設定を定義してあります。

| 設定 | 定義 |
|---|---|
| なし | 整合性はサポートされない。 |
| Supported | 整合性はサポートされるが、必須ではない。 |
| Required | 整合性は必須となる。 |

## 使用する場所

integrity 要素は、transport-requirements 要素内で使用されます。

## 例

integrity 要素の使用例については、5-26 ページのコード リスト 5-9「transport-requirements 要素の例」を参照してください。

# principal-name

principal-name 要素は、security-role-assignment 要素で指定したロール名に適用する、WebLogic Server セキュリティ レルム内のプリンシパルの名前を指定します。security-role-assignment 要素には、少なくとも 1 つの principal が必要です。各ロール名に対しては、複数の principal-name を定義できます。

### 使用する場所

principal-name 要素は、security-role-assignment 要素内で使用されます。

### 例

principal-name 要素の使用例については、5-6 ページのコード リスト 5-1「ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング」を参照してください。

## role-name

role-name 要素は、EJB プロバイダが対となる ejb-jar.xml ファイルに指定したアプリケーションのロール名を示します。スタンザの次の principal-name 要素で、WebLogic Server のプリンシパルを、指定した role-name にマップします。

### 使用する場所

role-name 要素は、security-role-assignment 要素内で使用されます。

### 例

role-name 要素の使用例については、5-6 ページのコード リスト 5-1「ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング」を参照してください。

## run-as-identity-principal

run-as-identity-principal 要素は、セキュリティ プリンシパルを run-as ID として使用するかどうかを指定します。この要素は、ejb-jar.xml ファイルの run-as 要素で指定したロールが、セキュリティ レルム内のユーザを表す単一のセキュリティ プリンシパルにマップされない場合に必要です。これは以下の 2 つのケースで当てはまります。

- ケース 1 : ロールが複数のセキュリティ プリンシパルにマップされます。このケースでは、run-as-identity-principal 要素を使用して、どのセキュリティ プリンシパルを使用するかを指定する必要があります。

- ケース 2 : ロールが、セキュリティ レルム内のグループを表す単一のセキュリティ プリンシパルにマップされます。このケースでは、run-as-identity-principal 要素を使用して、セキュリティ レルム内の特定のユーザを指定する必要があります。このケースの例については、5-23 ページのコード リスト 5-7 「run-as-identity-principal 要素の例」を参照してください。

## 使用する場所

run-as-identity-principal 要素は、weblogic-enterprise-bean 要素内で使用されます。

## 例

run-as-identity-principal 要素の使用例については、コード リスト 5-7 を参照してください。

**コード リスト 5-7 run-as-identity-principal 要素の例**

**ebj-jar.xml:**

```
<ejb-jar>
 <enterprise-beans>
 <session>
   <ejb-name>Caller2EJB</ejb-name>
   <home>weblogic.ejb11.security.CallerBeanHome</home>
   <remote>weblogic.ejb11.security.CallerBeanRemote</remote>
   <ejb-class>weblogic.ejb11.security.CallerBean</ejb-class>
   <session-type>Stateful</session-type>
   <transaction-type>Container</transaction-type>
   <ejb-ref><ejb-ref-name>Callee2Bean</ejb-ref-name>
     <ejb-ref-type>Session</ejb-ref-type>
     <home>weblogic.ejb11.security.CalleeBeanHome</home>
     <remote>weblogic.ejb11.security.CalleeBeanRemote</remote>
   </ejb-ref>
   <security-role-ref>
     <role-name>users1</role-name>
     <role-link>users1</role-link>
   </security-role-ref>

   <security-identity>
    <run-as>
      <role-name>users2</role-name>
    </run-as>
   </security-identity>
  </session>
 </enterprise-beans>
</ejb-jar>
```

**woblogic-ejb-jar.xml:**

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>Caller2EJB</ejb-name>
    <reference-descriptor>
     <ejb-reference-description>
       <ejb-ref-name>Callee2Bean</ejb-ref-name>
        <jndi-name>security.Callee2Bean</jndi-name>
     </ejb-reference-description>
    </reference-descriptor>
    <run-as-identity-principal>wsUser3</run-as-identity-principal>
  </weblogic-enterprise-bean>
  <security-role-assignment>
    <role-name>user</role-name>
    <principal-name>wsUser2</principal-name>
    <principal-name>wsUser3</principal-name>
    <principal-name>wsUser4</principal-name>
  </security-role-assignment>
</weblogic-ejb-jar>
```

## security-permission

security-permission 要素は、J2EE Sandbox と関連するセキュリティ パーミッションを指定します。

### 例

security-permission 要素の使用例については、5-25 ページのコード リスト 5-8 「security-permission-spec 要素の例」を参照してください。

## security-permission-spec

security-permission-spec 要素は、セキュリティ ポリシー ファイル構文に基づいて単一のセキュリティ パーミッションを指定します。

詳細については、Sun によるセキュリティ パーミッション仕様の実装を参照してください。

http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html#FileSyntax

**注意：** オプションの codebase および signedBy 句は無視してください。

### 使用する場所

security-permission-spec 要素は、security-permission 要素内で使用されます。

### 例

security-permission-spec 要素の使用例については、コード リスト 5-8 を参照してください。

**コード リスト 5-8 security-permission-spec 要素の例**

```
<weblogic-ejb-jar>
  <security-permission>
     <description>Optional explanation goes here</description>
     <security-permission-spec>
<!—
http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html
#FileSyntax の構文に準拠する、「codebase」句や「signedBy」句を含まない単一
の grant 文をここに記述します。次に例を示します。
-->
      grant {
      permission java.net.SocketPermission "*", "resolve";
      };
     </security-permission-spec>
   </security-permission>
</weblogic-ejb-jar>
```

コード リスト 5-8 では、permission java.net.SocketPermission はパーミッション クラス名を、"*" は対象名を、resolve (host/IP 名サービスのルックアップを解決する ) はアクションを示します。

## security-role-assignment

security-role-assignment 要素は、ejb-jar.xml ファイル内のアプリケーション ロールを、WebLogic Server で使用可能なセキュリティ プリンシパル名にマップします。

### 例

security-role-assignment 要素の使用例については、5-6 ページのコード リスト 5-1 「ejb-jar.xml および weblogic-ejb-jar.xml ファイルを使用したセキュリティ ロール名とセキュリティ レルムのマッピング」を参照してください。

## transport-requirements

transport-requirements 要素は、EJB の転送の要件を定義します。

### 使用する場所

transport-requirements 要素は、iiop-security-descriptor 要素内で使用されます。

### 例

transport-requirements 要素の使用例については、コード リスト 5-9 を参照してください。

**コード リスト 5-9　transport-requirements 要素の例**

```
<weblogic-enterprise-bean>
  <iiop-security-descriptor>
       <transport-requirements>
              <confidentiality>supported</confidentiality>
              <integrity>supported</integrity>
              <client-cert-authorization>
                  supported
              </client-cert-authentication>
       </transport-requirements>
  </iiop-security-descriptor>
</weblogic-enterprise-bean>
```

# EJB でのプログラムによるセキュリティの使用

プログラムによるセキュリティを EJB に実装するには、javax.ejb.EJBContext.getCallerPrincipal() メソッドと javax.ejb.EJBContext.isCallerInRole() メソッドを使用します。

## getCallerPrincipal

getCallerPrincipal() メソッドは、EJB の呼び出し元を特定するために使用します。javax.ejb.EJBContext.getCallerPrincipal() メソッドは、呼び出し元のユーザの Subject に入っている場合に WLSUser Principal を返します。WLSUser Principal が複数の場合、メソッドは、Subject.getPrincipals().iterator() メソッドで指定された順序の 1 番目を返します。WLSUser Principal が存在しない場合、getCallerPrincipal() メソッドは WLSGroup Principal 以外の 1 番目を返します。Principal が存在しない場合、またはすべての Principal が WLSGroup タイプの場合、このメソッドは weblogic.security.WLSPrincipals.getAnonymousUserPrincipal() を返します。この動作は weblogic.security.SubjectUtils.getUserPrincipal() のセマンティクスとほぼ同じですが、EJBContext.getCallerPrincipal() は WLSPrincipals.getAnonmyousUserPrincipal() を返すのに対し、SubjectUtils.getUserPrincipal() は null を返す点が異なります。

getCallerPrincipal() メソッドの使い方については、http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Security5.html を参照してください。

## isCallerInRole

呼び出し元 ( 現在のユーザ ) に割り当てられているセキュリティ ロールにおいてその実行スレッドでの WebLogic Server リソースに対するアクションの実行が許可されているかどうかを判定するには、isCallerInRole() メソッドを使用します。たとえば、現在のユーザが admin 特権を持っている場合、javax.ejb.EJBContext.isCallerInRole("admin") メソッドは true を返します。

isCallerInRole() メソッドの使い方については、http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Security5.html を参照してください。

isCallerInRole() メソッドの Javadoc については、http://java.sun.com/products/ejb/javadoc-1.1/javax/ejb/EJBContext.html#isCallerInRole(java.lang.String) を参照してください。

# 6　ネットワーク接続フィルタの使い方

ここでは以下のトピックについて説明します。


- 6-1 ページ「ネットワーク接続フィルタを使用する利点」
- 6-2 ページ「ネットワーク接続フィルタ API」
- 6-5 ページ「接続フィルタ ルールの作成ガイドライン」
- 6-8 ページ「WebLogic 接続フィルタのコンフィグレーション」
- 6-8 ページ「カスタム接続フィルタの開発」
- 6-9 ページ「接続フィルタのサンプル」


## ネットワーク接続フィルタを使用する利点

セキュリティ ロールおよびセキュリティ ポリシーを使うと、WebLogic リソースをドメイン レベル、アプリケーション レベル、およびアプリケーション コンポーネント レベルで保護できますが、接続フィルタを使用すると、ネットワーク レベルでアクセス拒否ができます。したがって、ネットワーク接続フィルタを使用することで、セキュリティのレイヤを追加できます。接続フィルタを使うと、個々のサーバ、サーバのクラスタ、または全体的な内部ネットワークやインターネット上で、サーバ リソースを保護できます。

接続フィルタは、管理ポートを使用する場合に特に便利です。ネットワーク ファイアウォールのコンフィグレーションに応じて、接続フィルタを使用して管理アクセスの制限を強化することができます。一般的な使用方法は、管理ポートへのアクセスをドメイン内のサーバおよびマシンに制限することです。攻撃者がファイアウォール内部のマシンにアクセスできたとしても、管理操作を許可されたマシン上でなければ、その操作を実行することはできなくなります。

ネットワーク接続フィルタは、プロトコル、IP アドレス、および DNS ノード名に基づいてフィルタ処理するようコンフィグレーションできる点において一種のファイアウォールです。たとえば、ユーザの企業のネットワーク外部からの非 SSL 接続を拒否できます。これにより、インターネット上のシステムからのすべてのアクセスがセキュアであることが保証されます。

# ネットワーク接続フィルタ API

この節では、`weblogic.security.net` パッケージについて説明します。この API は、ネットワーク接続フィルタを開発するためのインタフェースとクラスを提供します。またこれには、ネットワーク接続フィルタのすぐに使える実装である、`ConnectionFilterImpl` クラスが含まれます。詳細については、WebLogic Server のこのリリースにおける WebLogic クラスの Javadoc を参照してください。

ここでは以下のトピックについて説明します。


- 6-2 ページ「接続フィルタのインタフェース」
- 6-4 ページ「接続フィルタのクラス」


## 接続フィルタのインタフェース

接続フィルタ処理を実装するには、接続フィルタ インタフェースを実装するクラスを記述します。接続フィルタの実装用に、以下の `weblogic.security.net` インタフェースが提供されます。


- 6-2 ページ「ConnectionFilter インタフェース」
- 6-3 ページ「ConnectionFilterRulesListener インタフェース」


### ConnectionFilter インタフェース

このインタフェースでは、接続フィルタ処理を実装するために使用する `accept()` メソッドが定義されています。接続フィルタ処理を実行するようにサーバをプログラミングするには、このインタフェースを実装するクラスをイン

スタンス化し、Administration Console でそのクラスをコンフィグレーションします。このインタフェースは、接続フィルタ処理に対する最低限の実装要件です。

**注意：** このインタフェースを実装しただけでは、Administration Console を使ってクライアント接続を制限するフィルタ処理ルールを入力および修正することはできません。他の方法 (Administration Console で定義するフラット ファイルなど ) を使ってルールを指定する必要があります。Administration Console を使ってフィルタ処理ルールを入力および修正するには、`ConnectionFilterRulesListener` インタフェースも実装する必要があります。`ConnectionFilterRulesListener` インタフェースについては、6-3 ページ「ConnectionFilterRulesListener インタフェース」を参照してください。

このインタフェースを使用する方法を示すコード例については、6-9 ページ「接続フィルタのサンプル」を参照してください。

## ConnectionFilterRulesListener インタフェース

サーバはこのインタフェースを使って、Administration Console の `ConnectionFilterRules` フィールドで指定したルールが有効であるかどうかを、`setRules()` メソッドおよび `checkRules()` メソッドに基づいて起動中および実行時に判断します。

**注意：** 必要に応じて、このインタフェースを実装することも、WebLogic Server 製品の一部として提供される WebLogic 接続フィルタ実装 `weblogic.security.net.ConnectionFilterImpl` を使用することもできます。

このインタフェースでは、接続フィルタ処理を実装するために使用する 2 つのメソッド `setRules()` と `checkRules()` が定義されています。`ConnectionFilter` インタフェースと共にこのインタフェースを実装すれば、クライアント接続を制限するフィルタ処理ルールを、Administration Console を使って入力することができます。

**注意：** Administration Console で接続フィルタ処理ルールを入力および編集するためには、`ConnectionFilterRulesListener` インタフェースを実装する必要があります。このインタフェースを実装しない場合は、フラット ファイルなどの他の方法を使用する必要があります。

このインタフェースを使用するコード例については、6-9 ページ「接続フィルタのサンプル」を参照してください。

# 接続フィルタのクラス

接続フィルタの実装用に、2 つの `weblogic.security.net` クラスが提供されます。

- 6-4 ページ「ConnectionFilterImpl クラス」
- 6-4 ページ「ConnectionEvent クラス」

## ConnectionFilterImpl クラス

このクラスは、WebLogic 接続フィルタの実装です。`ConnectionFilter` インタフェースおよび `ConnectionFilterRulesListener` インタフェースを実装します。いったん Administration Console を使ってコンフィグレーションされると、この接続フィルタはデフォルトですべての受信時接続を受け入れ、またサーバが現在の接続フィルタを取得できるようにする静的なファクトリ メソッドを提供します。この接続でアクセスを拒否するには、Administration Console を使用して接続フィルタ ルールを入力するだけです。

このクラスは、WebLogic Server 製品の一部として提供されます。このクラスをコンフィグレーションして使用するには、6-8 ページ「WebLogic 接続フィルタのコンフィグレーション」を参照してください。

## ConnectionEvent クラス

これは、すべてのイベント ステート オブジェクトが派生する元のクラスです。すべてのイベントは、そのオブジェクト、つまり特定のイベントが最初に発生したオブジェクトであると論理的に見なされるソースを基準に作成されます。アプリケーションは、このクラスで提供されるメソッド (`getLocalAddress()`、`getLocalPort()`、`getRemoteAddress()`、`getRemotePort()`、および `hashcode()`) を使用して、新しい `ConnectionEvent` インスタンスを作成します。

このクラスを使用するコード例については、6-11 ページのコード リスト 6-1「ネットワーク接続をフィルタ処理するサンプル コード」を参照してください。

# 接続フィルタ ルールの作成ガイドライン

この節では、接続フィルタ ルールの記述方法と評価方法について説明します。接続ルールが指定されていない場合は、すべての接続が受け入れられます。

接続フィルタ ルールの指定は、接続フィルタ処理の実装方法により、フラットファイルで記述したり、Administration Console で直接入力したりすることができます。ネットワーク接続フィルタのコード例 (*SAMPLES_HOME*\server\src\examples\security\net ディレクトリ ) では、2 つのメソッドが示されています。

以下の節では、接続フィルタ ルールを記述するために必要な情報とガイドラインを説明しています。


- 6-5 ページ「接続フィルタ ルールの構文」
- 6-6 ページ「接続フィルタ ルールの種類」
- 6-7 ページ「接続フィルタ ルールの評価方法」


## 接続フィルタ ルールの構文

接続フィルタ ルールの構文は以下のとおりです。

- 各ルールは、1 行に記述する必要がある。
- ルールのトークンは、ホワイト スペースで区切る必要がある。
- シャープ記号 (#) はコメント文字。シャープ記号から行末までの記述は無視される。
- ルールの前後のホワイト スペースは無視される。
- ホワイト スペースまたはコメントだけの行はスキップされる。

接続フィルタ ルールの形式は、フィルタ ルールを入力するフィルタ ファイルを使用するか、Administration Console でフィルタ ルールを入力するかによって異なります。

- Administration Console でフィルタ ルールを入力する場合、次の形式で入力します。

  ```
  targetAddress localAddress localPort action protocols
  ```

- フィルタ ファイルでルールを指定する場合、次の形式で入力します。

  ```
  targetAddress action protocols
  ```

各要素の説明は次のとおりです。

- `targetAddress` では、フィルタ処理を行うシステムを 1 つ以上指定する。
- `localAddress` では、WebLogic Server インスタンスのホスト アドレスを定義する。アスタリスク (*) を指定すると、すべてのローカル IP アドレスが返されます。
- `localPort` では、WebLogic Server インスタンスがリスンしているポートを定義する。アスタリスクを指定すると、サーバで利用可能なすべてのポートが返されます。
- `action` では、実行するアクションを指定する。この値は、`allow` または `deny` でなければなりません。
- `protocols` は一致するプロトコル名のリスト。指定できるプロトコルは、`http`、`https`、`t3`、`t3s`、`giop`、`giops`、`dcom`、`ftp` です。プロトコルを定義しないと、すべてのプロトコルがルールに一致します。

# 接続フィルタ ルールの種類

2 種類のフィルタ ルールが認識されます。

- ファスト ルール

  ファスト ルールは、ホスト名またはネットマスク ( オプション ) を含む IP アドレスに適用されます。ホスト名が複数の IP アドレスに対応する場合は、複数のルール ( 順序は不定 ) が生成されます。ネットマスクは、数値またはドットで区切った 4 つの値の形式で指定できます。次に例を示します。

```
dialup-555-1212.pa.example.net  127.0.0.1 7001 deny t3 t3s  # http(s) OK

192.168.81.0/255.255.254.0   127.0.0.1 8001 allow   # 23-bit netmask

192.168.0.0/16   127.0.0.1 8002 deny   # like /255.255.0.0
```

ファスト ルールに対してホスト名を指定すると、WebLogic Server インスタンスの起動時に 1 回だけルックアップされます。この方法は、接続時のオーバーヘッドの大幅な削減には有効ですが、ホスト名とアドレスの対応について古い情報をフィルタが取得する可能性があります。数値の IP アドレスを使用することをお勧めします。

- スロー ルール

  スロー ルールは、ドメイン名の一部に適用されます。一致検査を実行するにはクライアントサイドで接続時の DNS ルックアップが必要なので、スロー ルールはファスト ルールよりかなり遅くなる場合があります。また、スロー ルールは DNS スプーフィング攻撃の対象になる可能性があります。スロー ルールの指定方法は次のとおりです。

  ```
  *.script-kiddiez.org 127.0.0.1 7001 deny
  ```

  アスタリスクは、パターンの先頭に対してのみ使用できます。ルールの他の位置でアスタリスクを使用すると、パターンの一部として扱われます。アスタリスクはドメイン名の一部として認められていないので、このようなパターンはドメイン名と一致しなくなります。

# 接続フィルタ ルールの評価方法

クライアントが WebLogic Server に接続すると、記述されている順序でルールが評価されます。一致する最初のルールにより、接続の処理方法が決まります。一致するルールがない場合には、接続は許可されます。

サーバの保護をさらに厳しくし、特定のアドレスからの接続だけを許可する場合は、最後のルールを次のように指定します。

```
0.0.0.0/0  *  *  deny
```

最後のルールをこのように指定すると、それまでのルールで許可されている接続だけが認められて、他の接続はすべて拒否されます。たとえば次のルールを指定したとします。

```
Remote IP Address * * allow https
0.0.0.0/0  *  *  deny
```

Remote IP Address を持つマシンだけが、接続フィルタを実行する WebLogic Server のインスタンスへのアクセスを許可されます。その他のシステムはすべて、アクセスを拒否されます。

# WebLogic 接続フィルタのコンフィグレーション

接続フィルタのコンフィグレーション方法については、『WebLogic Security の管理』を参照してください。

# カスタム接続フィルタの開発

WebLogic の接続フィルタを使用せず、独自に開発する場合は、`weblogic.security.net` パッケージで提供されているアプリケーション プログラミング インタフェース (API) を使用できます。この API の詳細については、6-2 ページ「ネットワーク接続フィルタ API」を参照してください。

WebLogic Server でカスタム接続フィルタを開発するには、次の手順を実行します。

1. `ConnectionFilter` インタフェースを実装するクラスを記述します ( 最低限の要件 )。

   または、Administration Console を使用して接続フィルタ処理ルールを入力したり、既存のフィルタ処理ルールを直に編集したりする場合は、`ConnectionFilter` インタフェースおよび `ConnectionFilterRulesListener` インタフェースを実装するクラスを記述します。

2. `ConnectionFilter` インタフェースだけを実装して手順 1 の最低要件を満たす場合、接続フィルタ処理ルールをフラット ファイルで入力し、`ConnectionFilter` インタフェースを実装するクラスでそのファイルの場所を定義します。その後、Administration Console を使って WebLogic Server 内のクラスをコンフィグレーションします。Administration Console でクラスをコンフィグレーションする方法については、『WebLogic Security の管理』の「接続フィルタのコンフィグレーション」を参照してください。接続フィルタ処理を実装するためのフラット ファイルの使用例については、*`SAMPLES_HOME`*`\server\src\examples\security\net` ディレクトリの `SimpleConnectionFilter.java` ファイルを参照してください。

3. 手順 1 で 2 つのインタフェースを定義した場合、Administration Console を使用してクラスをコンフィグレーションし、接続フィルタ処理ルールを入力します。Administration Console でクラスをコンフィグレーションする方法については、『WebLogic Security の管理』の「接続フィルタのコンフィグレーション」を参照してください。接続フィルタ処理を実装するための `ConnectionFilterRulesListener` インタフェースの使用例については、*`SAMPLES_HOME`*`\server\src\examples\security\net` ディレクトリの `SimpleConnectionFilter2.java` ファイルを参照してください。

**注意：** Java クライアントまたは Web ブラウザ クライアントが WebLogic Server インスタンスへの接続を試行する際に接続フィルタ処理を実装すると、WebLogic Server インスタンスは `ConnectionEvent` オブジェクトを作成して、接続フィルタ クラスの `accept()` メソッドに渡します。接続フィルタ クラスは、`ConnectionEvent` オブジェクトを調べて、接続を受け付ける場合はオブジェクトを返し、接続を拒否する場合は `FilterException` を送出します。

実装されたクラス (`ConnectionFilter` インタフェースのみを実装するクラスと、`ConnectionFilter` インタフェースおよび `ConnectionFilterRulesListener` インタフェースの双方を実装するクラス ) は両方とも、クライアント接続に関する情報を収集後に、`accept()` メソッドを呼び出す必要があります。ただし、`ConnectionFilter` インタフェース だけを実装している場合、収集される情報にはリモート IP アドレスと接続プロトコル (HTTP、HTTPS、T3、T3S、GIOP、GIOPS、DCOM、または FTP) が含まれます。どちらのインタフェースも実装している場合、収集される情報には、リモート IP アドレス、リモート ポート番号、ローカル ポート番号、および接続プロトコルが含まれます。

# 接続フィルタのサンプル

WebLogic Server のソフトウェアには、2 つの接続フィルタ サンプルが付属しています。どちらにも、効率の良い一般的な接続フィルタが含まれています。どちらのサンプルも、ルールを解析し、接続フィルタ処理によって WebLogic Server 接続に付加されるオーバーヘッドが最小限になるようなルール照合アルゴリズムを設定します。必要に応じて、このコードを変更して再利用できます。たとえ

ば、ローカルまたはリモートのポート番号を、フィルタ、またはフィルタ処理のオーバーヘッドを軽減するサイト固有のアルゴリズムに合わせることができます。サンプルをビルド、コンフィグレーション、および実行する方法については、WebLogic Server の *SAMPLES_HOME*\server\src\examples\security\net ディレクトリにある package.html ファイルを参照してください。

ここでは以下のトピックについて説明します。

- 6-10 ページ「SimpleConnectionFilter サンプル」
- 6-10 ページ「SimpleConnectionFilter2 サンプル」
- 6-11 ページ「ネットワーク接続をフィルタ処理する場合の accept メソッドの例」

# SimpleConnectionFilter サンプル

examples.security.net.SimpleConnectionFilter サンプルは、WebLogic Server の *SAMPLES_HOME*\server\src\examples\security\net ディレクトリにあります。このサンプルは、ConnectionFilter インタフェースを実装し、フィルタ ファイルで定義されているルールを使って接続のフィルタ処理を行います。

# SimpleConnectionFilter2 サンプル

examples.security.net.SimpleConnectionFilter2 サンプルは、WebLogic Server の *SAMPLES_HOME*\server\src\examples\security\net ディレクトリにあります。このサンプルは、ConnectionFilter インタフェースと ConnectionFilterRulesListener インタフェースを実装し、Administration Console で定義されたルールを使って接続のフィルタ処理を行います。

# ネットワーク接続をフィルタ処理する場合の accept メソッドの例

コード リスト 6-1 では、WebLogic Server は、ConnectionEvent を指定した SimpleConnectionFilter.accept() メソッドを呼び出しています。SimpleConnectionFilter.accept() メソッドは、リモート アドレスとプロトコルを取得してプロトコルをビットマスクに変換し、ルール適合時に文字列が比較されることを防ぎます。次に、SimpleConnectionFilter.accept() メソッドは、ルールごとにリモート アドレスとプロトコルを比較して、一致するものを見つけます。

このコードは、*SAMPLES_HOME*\server\src\examples\security\net ディレクトリの SimpleConnectionFilter.java ファイルからの抜粋です。

**コード リスト 6-1　ネットワーク接続をフィルタ処理するサンプル コード**

```
public void accept(ConnectionEvent evt)
  throws FilterException
{
  InetAddress remoteAddress = evt.getRemoteAddress();
  String protocol = evt.getProtocol().toLowerCase();
  int bit = protocolToMaskBit(protocol);

  // この特殊なビットマスクは、
  // 認識されたプロトコルのいずれかが接続で使用されないことを
  // 示す
  if (bit == 0xdeadbeef)
  {
    bit = 0;
  }

  // 記述された順でルールをチェック
  for (int i = 0; i < rules.length; i++)
  {
    switch (rules[i].check(remoteAddress, bit))
    {
    case FilterEntry.ALLOW:
  return;
    case FilterEntry.DENY:
  throw new FilterException("rule " + (i + 1));
    case FilterEntry.IGNORE:
  break;
    default:
 throw new RuntimeException("connection filter internal error!");
    }
```

```
    }
    // 一致したルールがない場合でも、接続を成功させることができる
    return;
  }
```

# 7 Java セキュリティを使用しての WebLogic リソースの保護

この節では次のトピックについて説明します。


- 7-1 ページ「J2EE セキュリティを使用しての WebLogic リソースの保護」
- 7-2 ページ「Java セキュリティ マネージャを使用しての WebLogic リソースの保護」


## J2EE セキュリティを使用しての WebLogic リソースの保護

WebLogic Server では、Web、エンタープライズ JavaBean (EJB)、およびコネクタのコンポーネントを保護するために J2EE セキュリティを使用できます。さらに、WebLogic Server では、デプロイメント記述子を使って追加のセキュリティ ポリシーを指定するコネクタ モデルが、URL および EJB のコンポーネントに拡張されています。

**注意：** Connector 1.0 仕様 (11.2 節を参照 ) と同じように、J2EE では異なるアプリケーション タイプ (J2EE 1.3 仕様 6.2.2 節を参照 ) の Java 2 セキュリティのデフォルト パーミッションに対する要件が規定されています。これらの仕様は、`http://java.sun.com/j2ee/download.html#platformspec` で参照できます。

さらに、J2EE 仕様では、デプロイヤがこれらのセキュリティ ポリシーを追加できることが示唆されています。URL および EJB コンポーネントの場合、これはデプロイメント記述子のコメントを通して行われますが、仕様では「この仕様の将来のバージョンでは、アプリケーション コンポーネントに対するデプロイメント記述子の中でこれらのセキュリティ要件を指定することが認められる」と記

されています。コネクタ仕様では、既に <security-permission> タグを使って追加セキュリティ ポリシーを指定するデプロイメント記述子に対応しています ( コード リスト 7-1 を参照 )。

**コード リスト 7-1 security-permission タグのサンプル**

```
<security-permission>
<description> Optional explanation goes here </description>
<security-permission-spec>
<!—
http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html
#FileSyntax の構文に準拠する、「codebase」句や「signedBy」句を含まない単一
の grant 文をここに記述します。次に例を示します。
-->
grant {
permission java.net.SocketPermission "*", "resolve";
};
</security-permission-spec>
</security-permission>
```

<rar.xml> ファイルでの <security-permission> タグのサポートに加えて、WebLogic Server では weblogic.xml ファイルと weblogic-ejb-jar.xml ファイルに security-permission タグを追加しています。これは、コネクタ モデルを他の 2 つのアプリケーション タイプ (Web アプリケーションと EJB) に拡張し、すべてのコンポーネント タイプでセキュリティ ポリシーへのインタフェースを統一すると共に、将来の J2EE 仕様の変更に備えるためのものです。

# Java セキュリティ マネージャを使用しての WebLogic リソースの保護

Java セキュリティ マネージャと WebLogic Server を一緒に使用すると、Java 仮想マシン (JVM) 内で実行されているリソースのセキュリティを強化できます。Java セキュリティ マネージャは、任意のセキュリティ手順です。以下の節では、WebLogic Server で Java セキュリティ マネージャを使用する方法について説明します。

- 7-3 ページ「Java セキュリティ マネージャの設定」
- 7-7 ページ「レコーディング セキュリティ マネージャ ユーティリティの使い方」


Java セキュリティ マネージャの詳細については、`http://java.sun.com/j2se/1.4/docs/guide/security/index.html` の Java Security ページを参照してください。

# Java セキュリティ マネージャの設定

Java 2 (SDK 1.2 以降 ) 環境で WebLogic Server を実行する場合、WebLogic Server は Java 2 の Java セキュリティ マネージャを使用して、信頼性のないコードが Java セキュリティ ポリシー ファイルによって制限されているアクションを実行しないようにすることができます。

Java 仮想マシン (JVM) には、Java セキュリティ ポリシー ファイルでコードに制約を設定するセキュリティ メカニズムが組み込まれています。Java セキュリティ マネージャは、Java セキュリティ ポリシー ファイルを使用して一連のパーミッションをクラスに強制的に付与します。これらのパーミッションを使用すると、JVM のインスタンスで実行される指定されたクラスに特定の実行時処理を許可するかどうかを設定できます。多くの場合、脅威モデルでは悪意あるコードが JVM で実行されることを想定していないため、Java セキュリティ マネージャは必要ありません。しかし、信頼されていないサードパーティが WebLogic Server を使用し、信頼されていないクラスが実行される場合、Java セキュリティ マネージャが役立ちます。

WebLogic Server で Java セキュリティ マネージャを使用するには、WebLogic Server の起動時に `-Djava.security.policy` 引数と `-Djava.security.manager` 引数を指定します。`-Djava.security.policy` 引数は、Java 2 セキュリティ ポリシーを格納するファイル名を、相対パス名または絶対パス名で指定します。

WebLogic Server には、編集および使用可能なサンプル Java セキュリティ ポリシー ファイルが用意されています。このファイルは、*`WL_HOME`*`\server\lib\weblogic.policy` に格納されています。

セキュリティ ポリシー ファイルを指定しないで Java セキュリティ マネージャを有効にする場合、Java セキュリティ マネージャでは、*$JAVA_HOME*\jre\lib\security ディレクトリの java.policy ファイルに定義されるデフォルトのセキュリティ ポリシーを使用します。

Java セキュリティ マネージャのセキュリティ ポリシーは、以下のいずれかの方法で定義します。

- 7-4 ページ「weblogic.policy ファイルの修正」
- 7-5 ページ「アプリケーション型のセキュリティ ポリシーの設定」
- 7-6 ページ「アプリケーション固有のセキュリティ ポリシーの設定」

# weblogic.policy ファイルの修正

WebLogic Server デプロイメントで Java セキュリティ マネージャのセキュリティ ポリシー ファイルを使用するには、WebLogic Server の起動時に Java セキュリティ マネージャに weblogic.policy ファイルの場所を指定する必要があります。これを行うには、サーバの起動に使用する Java コマンド ラインで次の引数を設定します。

- java.security.manager - JVM に Java セキュリティ ポリシー ファイルを使用するよう指示します。
- java.security.policy - JVM に使用する Java セキュリティ ポリシー ファイルの場所を指示します。この引数は、Java セキュリティ ポリシーの完全修飾名 ( この場合は weblogic.policy) です。

次に例を示します。

```
java...-Djava.security.manager \
   -Djava.security.policy==c:\weblogic\weblogic.policy
```

**注意：** java.security.policy 引数を指定するときには、Java セキュリティ マネージャによって weblogic.policy ファイルだけが使用されるよう、= の代わりに == を使用します。== を使用すると、weblogic.policy ファイルはデフォルトのセキュリティ ポリシーをオーバライドします。単一の等号記号 (=) を使用した場合、weblogic.policy ファイルが既存のセキュリティ ポリシーに付加されます。

CLASSPATH に追加のディレクトリがある場合、または追加のディレクトリにアプリケーションをデプロイしている場合は、それらのディレクトリに対する特定のパーミッションを weblogic.policy ファイルに追加します。

weblogic.policy ファイルを使用する際には、次のような注意事項を考慮することをお勧めします。

- weblogic.policy ファイルのバックアップを作成し、安全な場所に保管します。
- オペレーティング システムを通じて、WebLogic Server デプロイメントの管理者は読み書き特権を持ち、それ以外のユーザはファイルへのアクセス権を持たないように、weblogic.policy ファイルにパーミッションを設定します。

**警告：** Java セキュリティ マネージャは、管理サーバと管理対象サーバの起動時に部分的に無効にされます。起動シーケンス中は、現在の Java セキュリティ マネージャが無効化され、checkRead() メソッドが無効化された Java セキュリティ マネージャに置き換えられます。このメソッドを無効化した場合、起動シーケンスのパフォーマンスは飛躍的に向上しますが、セキュリティは最低レベルに下がります。
WebLogic Server のスタートアップ クラスは、この部分的に無効にされた Java セキュリティ マネージャと一緒に実行されます。このため、スタートアップ クラスを十分にチェックして、セキュリティ ( ファイルの読み取りなど ) を検討する必要があります。

Java セキュリティ マネージャの詳細については、Web 上の http://java.sun.com/j2se/1.3/docs/api/index.html にある java.lang.SecurityManager クラスに関する Javadoc を参照してください。

# アプリケーション型のセキュリティ ポリシーの設定

サーブレット、EJB、および J2EE コネクタ リソース アダプタのデフォルト セキュリティ ポリシーを、Java セキュリティ ポリシー ファイルで設定します。サーブレット、EJB、およびリソース アダプタのデフォルト セキュリティ ポリシーは、以下のコードベースで Java セキュリティ ポリシー ファイルに定義します。

- サーブレット - "file:/weblogic/application/defaults/Web"
- EJB - "file:/weblogic/application/defaults/EJB"

- リソース アダプタ - `"file:/weblogic/application/defaults/Connectors"`

**注意：** これらのセキュリティ ポリシーは、WebLogic Server の特定のインスタンスにデプロイされるすべてのサーブレット、EJB、およびリソース アダプタに適用されます。

## アプリケーション固有のセキュリティ ポリシーの設定

特定のサーブレット、EJB、またはリソース アダプタのセキュリティ ポリシーを設定するには、セキュリティ ポリシーをそれらのデプロイメント記述子に追加します。デプロイメント記述子は、以下のファイルに定義されます。

- サーブレット - `weblogic.xml`
- EJB - `weblogic-ejb-jar.xml`
- リソース アダプタ - `rar.xml`

**注意：** リソース アダプタのセキュリティ ポリシーは J2EE 仕様に準拠し、サーブレットおよび EJB のセキュリティ ポリシーは WebLogic Server の J2EE 仕様拡張に準拠します。

コード リスト 7-2 はセキュリティ ポリシーをデプロイメント記述子に追加するための構文です。

**コード リスト 7-2　セキュリティ ポリシーの構文**

```
<security-permission>
  <description>
    //foo 操作を保護
  </description>
  <security-permission-spec>
    // grant 文
    grant {
      permission java.lang.RuntimePermission “foo”;
    }
  </security-permission-spec>
</security-permission>
```

**注意：** 現時点では、<security-permission-spec> タグは weblogic-application.xml ファイルに追加できません。このタグを使用できるのは、weblogic-ejb-jar.xml、rar.xml、および weblogic.xml ファイルの中だけです。また、<security-permission-spec> 属性では変数はサポートされていません。

# レコーディング セキュリティ マネージャ ユーティリティの使い方

レコーディング セキュリティ マネージャ ユーティリティを使用すると、WebLogic Server の起動時または動作中に発生するパーミッションの問題を検出できます。このユーティリティで出力されるパーミッションを Java セキュリティ ポリシー ファイルに追加して、発見されたパーミッションの問題を解決できます。レコーディング セキュリティ マネージャは BEA dev2dev Online で入手できます。

# A　非推奨のセキュリティ API

以下に挙げた WebLogic セキュリティ パッケージに含まれるセキュリティ インタフェース、クラス、および例外の中の一部またはすべては、WebLogic Server のこのリリースでは非推奨になりました。

- weblogic.security
- weblogic.security.acl
- weblogic.security.audit
- weblogic.security.SSL

各パッケージで非推奨になったインタフェース、クラス、および例外の詳細については、各パッケージについての WebLogic クラスに関する Javadoc を参照してください。

# 索引


## A

API メソッド
- accept() 6-9
- doAs() 3-10, 3-19
- getAttribute() 4-27
- getCallerPrincipal() 5-26
- getSubject() 3-9
- isCallerInRole() 5-26
- isUserInRole() 2-41, 2-42
- login() 3-17, 3-21
- runAs() 3-9, 3-18
- setTrustManager() 4-31

AuthCookieEnabled 2-10

## C

CallbackHandler 3-13
config.xml 2-10

## D

DCOM 6-6, 6-9

## E

e コマース 1-4
EJB
- 宣言によるセキュリティの実装 5-5

EJBContext
- getCallerPrincipal() 5-26
- isCallerInRole() 5-26

ejb-jar.xml
- EJB にセキュリティを実装するための使用 5-5

## F

FTP 6-6, 6-9

## G

GIOP 6-6
GIOPS 6-6
global-role タグ 2-37, 2-38, 5-17, 5-18, 5-19

## H

HTTP 6-6, 6-9
HTTPS 6-6, 6-9
- protocol 4-8
- リソースへの安全なアクセス 2-10

HttpServletRequest.isUserInRole(String role) 2-41, 2-42
HttpsURLConnection 4-36

## I

message URL http
- //jcp.org/aboutJava/communityprocess/final/jsr154/index.html 2-30

IIOP 6-9
IIOPS 6-9
IP アドレス 6-9

## J

J2EE 7-2
J2SE 3-10, 3-19
JAAS 3-4
- getSubject() メソッド 3-9
- LoginContext 3-1
- LoginModule 3-9
- サブジェクト 3-9

サポートされているバージョン 3-3
プリンシパル 3-9
Java 2 Platform, Standard Edition
J2SE を参照
java.security.Principal インタフェース 3-9
javax.security.auth.Subject
doAs() メソッド 3-10, 3-19
javax.servlet.request.cipher_suite 4-27
javax.servlet.request.key_size 4-27
javax.servlet.request.X509Certificate 4-27, 4-28
JNDI 3-1
JNDI パラメータ
setProviderURL 4-26
setSSLClientCertificate 4-26
setSSLRootCAFingerprint 4-27
setSSLServerName 4-26

## L

login() メソッド 3-17, 3-21
LoginContext 3-16
login() メソッド 3-17, 3-21
LoginException 3-17

## P

principal-name タグ 5-17
PrivilegedAction 3-9
PrivilegedExceptionAction 3-9

## S

sample_jaas.config 3-16
security-permission タグ 7-2
SSL
ハンドシェーク 4-30
SSL 使用の制限 4-1
SSLContext 4-33
設定 4-34
SubjectDN 4-28

## T

T3 6-6, 6-9
T3S 6-6, 6-9
TrustManagerJSSE インタフェース
使用 4-30

## U

UsernamePasswordLoginModule 3-11

## W

Web アプリケーション 2-28
Web ブラウザ
無効な証明書に対する応答 4-31
web.xml
Web アプリケーションにセキュリティを実装するための使用 2-28
WebLogic Server
JAAS 用のコンテナ サポート 3-2
JAAS のサポート 3-3
SSL 実装 4-2
WebLogic Server の JSSE 実装 4-1
WebLogic セキュリティ
非推奨のパッケージとクラス 1-7
weblogic.jar 3-11
weblogic.security.Security.runAs() メソッド 3-18
weblogic.servlet.request.SSLSession 4-27
weblogic.xml 7-2
EJB にセキュリティを実装するための使用 5-5
Web アプリケーションにセキュリティを実装するための使用 2-28
weblogic-ejb-jar.xml 7-2

## X

X.509 証明書 4-28

## あ

暗号化
サービス 1-4

## い

印刷、製品のマニュアル xii
インタフェース
CallbackHandler 3-10
ConnectionFilter 6-2
ConnectionFilterRulesListener 6-3, 6-8
HostnameVerifier 4-29
HostnameVerifierJSSE 4-29
java.security.Principal 3-9
TrustManager 4-30
TrustManagerJSSE 4-30, 4-31

## か

介在者の攻撃 4-28
カスタマ サポート情報 xiii

## く

クラス
HttpsURLConnection 4-36
javax.security.PrivilegedAction 3-19
JNDI Environment 4-26
weblogic.security.Security 3-9

## け

検証エラー
オーバライド 4-30

## こ

コード例
accept() メソッド 6-11
CallbackHandler インタフェース 3-13
ConnectionFilter インタフェース 6-8
ConnectionFilterRulesListener インタフェース 6-9
getSubject() メソッド 3-19
login() メソッド 3-17
LoginContext 3-17
PrivilegedAction 3-19
runAs() メソッド 3-19
Sample_jaas.config 3-16
SimpleConnectionFilter 6-10
SimpleConnectionFilter2 6-10
SSLClient 4-12
SSLClientServlet 4-19
SSLServerSocketFactory 4-34
SSLSocketClient 4-16, 4-32
URL による発信 SSL 接続 4-37
トラスト マネージャ 4-31
ネットワーク接続フィルタ 6-5, 6-10
ホスト名検証のカスタム バージョン 4-29
コンフィグレーション ファイル 3-16

## さ

サービス パック 1-5
サーブレット コンテナ 2-28
サブジェクト 3-18
サポート
技術情報 xiv

## し

主体の公開鍵 4-28
信頼性のある認証局 4-27

## せ

セキュリティ
API 1-6
サーブレットでのプログラム的な適用 2-43
パッケージ
非推奨の詳細 8-1
リスト 8-1
セキュリティ ポリシー 6-1
宣言によるセキュリティ

EJB への実装 5-5
Web アプリケーションへの実装 2-28

## そ

相互認証 4-20

## て

デジタル証明書 4-27, 4-28
検証エラーのオーバライド 4-30
チェーン 4-30
デモバージョン 4-28
内容 4-28
無効 4-31
デプロイメント記述子
EJB にセキュリティを実装するための使用 5-5
Web アプリケーションにセキュリティを実装するための使用 2-28
デプロイメント記述子ファイル
ejb-jar.xml 5-5
web.xml 2-28
weblogic.xml 2-28, 5-5

## と

トラスト マネージャ
SSLContext との関連付け 4-33
カスタム チェックの実行 4-31
作成 4-31
メリット 4-31

## ね

ネットワーク接続フィルタ
作成のガイドライン 6-5
スロー ルール 6-7
ファスト ルール 6-6
フラット ファイルの例 6-8
ルールの構文 6-5
ルールの評価 6-7

## ふ

プログラムによるセキュリティ
EJB への実装 5-26
Web アプリケーションへの実装 2-41
プログラムによるセキュリティの実装
EJB 5-26
Web アプリケーション 2-41

## ほ

ホスト名検証
カスタム バージョン 4-29
カスタム バージョンの使い方 4-29
カスタム バージョンを使用するための要件 4-29
コード例 4-29
無効化の影響 4-28
無効にする場合 4-28
無効にする方法 4-28

## ま

マニュアル、入手先 xii

## む

無効な証明書に対する応答
SSL 仕様の定義 4-31
Web ブラウザ 4-31

## れ

例外
LoginException 3-17