

JEUS セキュリティガイド

JEUS v8.0



Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

Copyright Notice

Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 13613, South Korea

Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features. This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

このソフトウェア(JEUS®)マニュアルの内容とプログラムは、日本国の著作権法および国際条約によって保護されています。マニュアルの内容とプログラムは、TmaxSoft Co., Ltd.との使用許諾契約書の下でのみ使用することができ、マニュアルは使用許諾契約で許可されている範囲を除いては、配布または複製することができません。TmaxSoftの書面による事前の承諾を得ることなく、このマニュアルの全部または一部を電子的または機械的な方法を問わず、転送、複製、配布したり、または二次的著作物を作成する等の行為を一切禁じます。

このソフトウェアのマニュアルとプログラムの使用許諾契約は、いかなる場合においても、マニュアル及びプログラムと関連する知的財産権(登録の有無を問わず)を譲渡するものと解釈されず、TmaxSoftのブランド、ロゴ、商標等の使用権限を与えるものではありません。マニュアルは、情報を提供する目的でのみ提供しており、これに伴う契約上の直接的ないしは間接的な責任を負わず、マニュアルの内容は法律上もしくは商業的な特定の条件が満たされることを保証しません。マニュアルの内容は、製品のアップグレード及び修正により、その内容が予告なく変更されることがあり、内容上の誤りがないことを保証しません。

Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

JEUS®は、TmaxSoft Co., Ltd.の登録商標です。

Java and Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Java、Solarisは、Oracle Corporation及びその子会社、関連会社の登録商標です。

Microsoft, Windows, and Windows NT are registered trademarks or trademarks of Microsoft Corporation.

Microsoft、Windows、Windows NTは、Microsoft Corporationの登録商標または商標です。

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

HP-UXは、Hewlett Packard Enterprise Companyの登録商標です。

AIX is a registered trademark of International Business Machines Corporation.

AIXは、International Business Machines Corporationの登録商標です。

UNIX is a registered trademark of X/Open Company, Ltd.

UNIXは、X/Open Company, Ltd.の登録商標です。

Linux is a registered trademark of Linus Torvalds.

Linuxは、Linus Torvaldsの登録商標です。

Other products and company names are trademarks or registered trademarks of their respective owners.

その他、記載されている会社名、製品名などは、各社の商号、商標または登録商標です。

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (TM, ®).

本マニュアルに記載されている会社名、システム名、製品名などには必ずしも商標表示(TM、®)を付記しておりません。

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory : \${INSTALL_PATH}/lib/licenses

この製品の一部ファイルまたはモジュールは、APACHE2.0、CDDL1.0、EDL1.0、OPEN SYMPHONY SOFTWARE1.1、TRILEAD-SSH2、Bouncy Castle、BSD、MIT、SIL OPEN FONT1.1のライセンスに準拠します。

文書情報

文書名: JEUS セキュリティガイド

発行日: 2016年10月14日

ソフトウェアバージョン: JEUS v8.0

ガイドバージョン: v2.1.1

目次

このガイドについて	xv
第1章 セキュリティー・システムの紹介	1
1.1. 概要	1
1.2. 主要特徴	2
1.3. システム構造	3
1.4. 主要概念	4
1.4.1. ログイン	5
1.4.2. 認証	5
1.4.3. 権限チェック(付与)	7
1.4.4. セキュリティー監査	12
1.4.5. サービスとSPI	12
1.4.6. ドメイン	14
1.5. 性能改善とセキュリティー・レベルの向上	16
1.5.1. JEUS性能の改善	16
1.5.2. セキュリティー・レベルの向上	16
第2章 セキュリティー・システムの設定	19
2.1. 概要	19
2.2. セキュリティー・ドメインの定義	21
2.2.1. WebAdminを使用した設定	21
2.2.2. XML編集による設定	22
2.2.3. ユーザー・アカウントおよびセキュリティー・ポリシーの設定	23
2.3. セキュリティー・ドメイン構成要素の設定	24
2.3.1. WebAdminを使用した設定	24
2.3.2. XML編集による設定	28
2.4. セキュリティー・サービスの設定	30
2.4.1. WebAdminを使用した設定	30
2.4.2. XML編集による設定	44
2.5. セキュリティー・システムのユーザー情報の設定	49
2.5.1. WebAdminを使用した設定	49
2.5.2. XML編集による設定	52
2.5.3. データベースを使用した設定	53
2.5.4. パスワードのセキュリティー設定	55
2.5.5. ログイン情報のキャッシュ機能	59
2.6. セキュリティー・システム・ポリシーの設定	60
2.6.1. WebAdminを使用した設定	60
2.6.2. XML編集による設定	63
2.6.3. データベースを利用した設定	70
2.7. 追加項目の設定	71
2.7.1. JavaSE SecurityManagerの設定	71
2.7.2. JACCプロバイダーの設定	72

2.7.3.	アイデンティティー付与のための情報の設定	72
2.7.4.	アイデンティティーに対する証明書情報の設定	74
第3章	アプリケーションとモジュールのセキュリティ設定	77
3.1.	概要	77
3.1.1.	モジュール・デプロイ対アプリケーション・デプロイ	77
3.1.2.	Role-to-Resourceマッピング	77
3.1.3.	Principal-to-Roleマッピング	80
3.1.4.	ユーザー設定	82
3.2.	EJBモジュールのセキュリティ設定	82
3.2.1.	ejb-jar.xmlの設定	83
3.2.2.	jeus-ejb-dd.xmlの設定	86
3.3.	Webモジュールのセキュリティ設定	89
3.3.1.	web.xmlの設定	89
3.3.2.	jeus-web-dd.xmlの設定	94
3.4.	J2EEアプリケーションのセキュリティ設定	96
3.4.1.	application.xmlの設定	96
3.4.2.	jeus-application-dd.xmlの設定	98
3.5.	例	100
第4章	セキュリティ・システムAPIプログラミング	103
4.1.	概要	103
4.2.	Java SEパーミッションの設定	103
4.3.	基本API	104
4.4.	リソースAPI	105
4.5.	SPIクラス	106
4.6.	例	106
第5章	カスタム・セキュリティ・サービスの開発	109
5.1.	概要	109
5.2.	サービス・クラス	109
5.3.	カスタム・セキュリティ・サービスの実装パターン	112
5.4.	SPIクラス	113
5.4.1.	SubjectValidationService SPI	114
5.4.2.	SubjectFactoryService SPI	114
5.4.3.	AuthenticationService SPI	115
5.4.4.	AuthenticationRepositoryService SPI	116
5.4.5.	IdentityAssertionService SPI	116
5.4.6.	CredentialMappingService SPI	116
5.4.7.	CredentialVerificationService SPI	116
5.4.8.	AuthorizationService SPI	117
5.4.9.	AuthorizationRepositoryService SPI	118
5.4.10.	EventHandlingService SPI	118
5.4.11.	SPI実装間の依存関係	119
5.5.	セキュリティ・サービスの設定	119

第6章 JACCプロバイダーの使用	121
6.1. 概要	121
6.2. JACCの規約	121
6.2.1. プロバイダー設定の規約	122
6.2.2. ポリシー設定の規約	122
6.2.3. ポリシー決定および執行の規約	123
6.3. JACCプロバイダーの開発	124
6.3.1. JACCプロバイダーの実装	124
6.3.2. JACCプロバイダーのパッケージング	126
6.3.3. デフォルトのJACCプロバイダー	126
6.4. JEUSセキュリティ・システムとJACCプロバイダーの統合	126
第7章 JAASの使用	131
7.1. 概要	131
7.2. JEUSとLDAPを連携するためのLoginModuleの実装	132
7.3. LDAP JAAS LoginModuleサービスの設定	138
7.4. データベース連携のためのLoginModuleの実装	140
7.5. データベースのLoginModuleサービスの設定	147
付録 A. セキュリティ・イベント・サービス	151
A.1. 概要	151
A.2. イベント	151
付録 B. JEUSサーバーのパーミッション	157
B.1. 概要	157
B.2. JEUSシステム・リソース名	157
B.3. jeusadminコマンドの権限設定	158
索引	161

図目次

[図 1.1]	セキュリティ・システムの構造	3
[図 1.2]	スタック・ベースのログイン・メカニズム	5
[図 1.3]	サブジェクトのUMLダイアグラム	6
[図 1.4]	ロール・ベースのパーミッション・チェック	7
[図 1.5]	午前1時30分のロール	8
[図 1.6]	午前10時30分のロール	9
[図 1.7]	ポリシーとパーミッション・マップのUMLダイアグラム	10
[図 1.8]	1つのPrincipal-to-Roleマップと2つのRole-to-Resourceマップを持つポリシーの例	10
[図 1.9]	サービスの2つの状態	13
[図 1.10]	サービス・クラスとSPIのサブクラス	14
[図 1.11]	異なるアプリケーションと異なるサブジェクト・リポジトリを使った2つのドメイン	15
[図 2.1]	セキュリティ・マネージャーのメイン画面	22
[図 2.2]	セキュリティ・ドメインの設定	25
[図 2.3]	[Security Service] - [Cache Config]	26
[図 2.4]	[Security Manager] - [Key Store]	27
[図 2.5]	[Security Manager] - [Custom Service]	28
[図 2.6]	[Security Service] - [Authentication]	31
[図 2.7]	[Security Service] - [Authentication] - [Repository Service]	32
[図 2.8]	[Security Service] - [Authentication] - [Jaas Login Config]	34
[図 2.9]	[Security Service] - [Authentication] - [Custom Authentication Service]	35
[図 2.10]	[Security Service] - [Authorization] - [Basic]	36
[図 2.11]	[Security Service] - [Authorization] - [Custom Authorization Service]	38
[図 2.12]	[Security Service] - [Identity Assertion]	39
[図 2.13]	[Security Service] - [Credential Mapping]	41
[図 2.14]	[Security Service] - [Credential Verification]	42
[図 2.15]	[Security Service] - Audit	43
[図 2.16]	[Security Service] - [Subject Validation]	44
[図 2.17]	Accounts設定画面	50
[図 2.18]	Accounts - ユーザーの登録	50
[図 2.19]	Accounts - パスワードの設定	51
[図 2.20]	Accounts - パスワードの暗号化	51
[図 2.21]	サブジェクトのユーザー情報を保存するためのデータベース・テーブルの構造	55
[図 2.22]	WebAdminでパスワードの検証を設定する画面	57
[図 2.23]	ポリシー設定メイン画面	60
[図 2.24]	ポリシーの設定 - ロール・パーミッションの登録	61
[図 2.25]	ポリシーの設定 - リソース・パーミッションの登録(1)	62
[図 2.26]	ポリシーの設定 - リソース・パーミッションの登録(2)	62
[図 2.27]	ポリシーの設定 - リソース・パーミッションの登録(3)	63
[図 2.28]	ポリシーを保存するためのデータベース・テーブルの構造	71
[図 3.1]	Principal-to-Roleマッピング	80

[図 3.2]	ログイン画面	100
[図 3.3]	トップ画面	101
[図 5.1]	サービス・クラスのダイアグラム	110
[図 5.2]	サービス・クラスのステート・チャート	111
[図 5.3]	デフォルトのセキュリティー・システムの実装におけるSPI実装クラス	119
[図 6.1]	JACCプロバイダー・クラス	124

表目次

[表 1.1]	多様な権限チェックのクエリーと結果の例	12
[表 5.1]	jeus.security.spiパッケージ内の標準SPIクラス	113

例目次

[例 2.1]	<<JEUS_HOME/domains/<domain name>/config/domain.xml>>	22
[例 2.2]	セキュリティー・システムのサービス設定 : <<domain.xml>>	28
[例 2.3]	セキュリティー・システムのサービスの設定 : <<domain.xml>>	44
[例 2.4]	セキュリティー・システムのユーザー情報の設定 : <<accounts.xml>>	52
[例 2.5]	データベースを使用したユーザーの設定 : <<domain.xml>>	53
[例 2.6]	JEUS JDBCを使用しない場合の設定 : <<domain.xml>>	54
[例 2.7]	パスワードの妥当性検査をDefaultPasswordValidatorに設定 : <<domain.xml>>	55
[例 2.8]	パスワードの妥当性検査をCustomPasswordValidatorに設定した例 : <<domain.xml>>	56
[例 2.9]	保存されたログイン情報 : <<.jeuspasswd>>	59
[例 2.10]	セキュリティー・システムのポリシーの設定 : <<policies.xml>>	63
[例 2.11]	カスタム・パーミッション・クラス : <<TimeConstrainedRolePermission.java>>	66
[例 2.12]	カスタム・パーミッション・クラス : <<policies.xml>>	69
[例 2.13]	データベースを利用したポリシーの設定 : <<domain.xml>>	70
[例 2.14]	Java SE SecurityManagerの設定 : <<policy>>	72
[例 2.15]	アイデンティティー付与のための情報の設定 : <<cert-user-map.xml>>	73
[例 2.16]	アイデンティティーに対する証明書情報の設定 : <<user-cert-map.xml>>	74
[例 3.1]	EJBモジュールに設定されているセキュリティー制約 : <<ejb-jar.xml>>	78
[例 3.2]	Principal-to-Roleマッピング : <<jeus-ejb-dd.xml>>	81
[例 3.3]	セキュリティーの設定 : <<ejb-jar.xml>>	83
[例 3.4]	セキュリティーの設定 : <<jeus-ejb-dd.xml>>	87
[例 3.5]	Webモジュールのセキュリティーの設定 : <<web.xml>>	92
[例 3.6]	Webモジュールのセキュリティーの設定 : <<jeus-web-dd.xml>>	95
[例 3.7]	J2EEアプリケーションのセキュリティー設定 : <<application.xml>>	97
[例 3.8]	J2EEアプリケーションのセキュリティーの設定 : <<jeus-application-dd.xml>>	99
[例 6.1]	JACCセキュリティー設定ファイルの設定 : <<domain.xml>>	127
[例 6.2]	JACCに対するJavaシステム属性の設定<<domain.xml>>	128
[例 7.1]	<<jeus.security.impl.login.LdapLoginModule>>	132
[例 7.2]	ドメイン・サービスの設定 : <<domain.xml>>	138
[例 7.3]	<<jeus.security.impl.login.DBRealmLoginModule>>	140
[例 7.4]	ドメイン・サービスの設定 : <<domain.xml>>	147
[例 B.1]	セキュリティー・システムのポリシー設定 : <<policies.xml>>	159

このガイドについて

対象読者

本書は、JEUS[®](以下、JEUS)のセキュリティに関連する設定方法、動作方法、カスタマイジング方法などを説明したガイドであり、JEUSシステム管理者を対象としています。

前提知識

本書を理解するためには、以下の内容についての知識が必要です。

- J2SEセキュリティ・アーキテクチャーと一般的なセキュリティ技術(SSL/TLS、デジタル認証、ロール・ベースのユーザー認証モデル)
- JEUSセキュリティ・アーキテクチャー

JEUSの基本的な使用方法と製品を理解するには、以下のガイドについてあらかじめ熟知することをお勧めします。

- JEUS 紹介ガイド
- JEUS インストール & スタートガイド
- JEUS サーバガイド

本書のすべてのサンプルと環境構成は、UNIXスタイルに準拠します。Microsoft Windows[™](以下、Windows)など他の環境で作業を行う場合は、次のような事項を考慮してください。たとえば、Windowsプラットフォームでは、ディレクトリー区切り子をUNIXスタイルのスラッシュ(/)からWindowsスタイルのバックスラッシュ(\)に変えて使用してください。また、環境変数もWindowsスタイル(%%)に変更して使用してください。本書で触れているJEUS_HOMEは、JEUSがインストールされているディレクトリーです。

制限事項

本書の内容は、Java標準に準拠して作成されていますが、本書で触れているJava EEやJava仕様については詳しく取り上げていません。関連内容についてはJava関連ドキュメントを参照してください。

本書の構成

本書は、計7章と2つの付録で構成されています。

- 「第1章 セキュリティー・システムの紹介」

JEUSセキュリティー・システムの概要を説明します。

- 「第2章 セキュリティー・システムの設定」

セキュリティー・ドメイン、セキュリティー・サービス、サブジェクト、ポリシーといった主なセキュリティー要素の設定方法について説明します。

- 「第3章 アプリケーションとモジュールのセキュリティー設定」

セキュリティー・システムでJ2EEアプリケーション、EJBモジュール、Webモジュールにセキュリティーを設定する方法について説明します。

- 「第4章 セキュリティー・システムAPIプログラミング」

セキュリティー・システムAPIを使うアプリケーション(たとえば、サーブレット)の開発方法について説明します。

- 「第5章 カスタム・セキュリティー・サービスの開発」

JEUSでカスタム・セキュリティー・プロバイダーの開発方法について説明します。

- 「第6章 JACCプロバイダーの使用」

JEUSセキュリティー・システムでJACCプロバイダーを開発および統合する方法について説明します。

- 「第7章 JAASの使用」

JEUSセキュリティー・システムでJAASを開発および統合する方法について説明します。

- 「付録 A. セキュリティー・イベント・サービス」

セキュリティー・システムで検知される標準セキュリティー・イベントについて説明します。

- 「付録 B. JEUSサーバーのパーミッション」

JEUSサーバーで行われるパーミッション・チェックに必要なセキュリティー・イベントのリファレンスについて説明します。

表記上の規則

表記	意味
<<AaBbCc123>>	プログラム・ソースコードのファイル名
<Ctrl>+C	CtrlキーとCキーを同時に押す
[Button]	GUIのボタン、メニュー名
太字	強調
「」、『』（鍵カッコ）	関連文書、あるいはガイド内の他の章および節の表示
「入力項目」	画面UI上の入力項目
ハイパーリンク	メール・アカウント、Webサイト
>	メニューの実行順
+----	下位ディレクトリー/ファイル有り
----	下位ディレクトリー/ファイル無し
<div>参考</div>	参照/注意事項
<div>注</div>	注意事項
[図 1.1]	図の名前
[表 1.1]	表の名前
AaBbCc123	Javaコード、XMLドキュメント
[<i>command argument</i>]	オプション・パラメータ
< xyz >	「<」と「>」の間の内容は実際に使用される特定の名前または値で置き換えられる
	構文の中の相互に排他的な選択項目の選択肢を示す 例) A B: AとBのいずれかを選択
...	パラメータ、値、または他の情報が繰り返される
\${ }	環境変数

システム要件

	要求事項
プラットフォーム	Solaris 9, 10, 11
	HP-UX 11.x, 11i, 11iV2
	IBM AIX 5L, 6L, AIX 7L
	MS Windows 2008, 2012, Vista, 7, 8
ハードウェア	最小2GB以上、推奨20GBのハードディスク容量
	推奨1GB以上のメモリー容量
JDK	JDK 7, JDK 8

関連文献

ガイド	説明
JEUS 紹介ガイド	JEUSサーバーについて全般的に紹介し、JEUSのアーキテクチャーを含む各構成要素について記述しています
JEUS インストール&スタートガイド	JEUSについて紹介し、JEUSのインストールおよび開始方法について記述しています
JEUS サーバガイド	JEUSシステムおよびサーバーの概要とシステムの管理方法について記述しています
JEUS EJBガイド	JEUS EJBエンジンおよびEJBモジュールのデプロイについて記述しています
JEUS Webエンジンガイド	JEUS Webエンジンの管理方法、Java EE WARアーカイブとサーブレット/JSPの管理およびデプロイ方法について記述しています
JEUS ノードマネージャガイド	JEUSノードマネージャの概念と動作方式を理解するための基本的な内容について記述しています
JEUS WebAdminガイド	JEUSのWeb管理ツールであるWebAdminを利用したJEUSの設定および制御、モニタリング、クラスタリング、リソースの設定および管理について記述しています

参考文献

- Java Authorization Contract for Containers Specification Version 1.0
JACCプロバイダーに関する情報を提供します。
- J2EE 6 Specification
JEUSを含むJ2EEサーバーに適用される基本的なセキュリティ・アーキテクチャーに関する情報を提供します。
- EJB 3.1 Specification
EJBセキュリティ・モデルに関する情報を提供します。
- Servlet 3.0 Specification
サーブレット・セキュリティ・モデルに関する情報を提供します。
- Javadoc for J2SE 6 packages java.security, javax.security.auth J2EE 6 package javax.security.jacc
J2SE/J2EEセキュリティに関連する基本的なクラスの詳細情報を提供します。

- Javadoc JEUS API

[JEUS_HOME/docs/api/jeusapi/index.html](#)

- XML Reference - セキュリティー・サービスのdomain.xmlの設定

[JEUS_HOME/docs/reference/schema/index.html](#)

お問合せ先

Korea

TmaxSoft Co., Ltd.
45, Jeongjail-ro, Bundang-gu,
Seongnam-si, Gyeonggi-do, 13613
South Korea
Tel: +82-31-8018-1000
Fax: +82-31-8018-1115
Email: info@tmax.co.kr
Web (Korean): <http://www.tmaxsoft.com>
TechNet: <http://technet.tmaxsoft.com>

USA

TmaxSoft Inc.
101 North Wacker Drive, Suite 2014,
Chicago, IL 60606
U.S.A
Tel: +1-312-525-8330
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/us_en/home

Japan

TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073
Japan
Tel: +81-3-5765-2550
Fax: +81-3-5765-2567
Email: info@tmaxsoft.co.jp
Web (Japanese): <http://www.tmaxsoft.co.jp>

China

Beijing TmaxSoft System Software Co., Ltd.
Room103, No.2 Huizhong Building, Seven Street Shangdi,
Haidian District, Beijing, 100085
P.R.China
Tel: +86-10-6298-8827
Email: info@tmaxsoft.com.cn
Web (Chinese): http://www.tmaxsoft.com/cn_en/home_cn_en

Brazil

Tmax Brasil Sistemas e Serviços Ltda.
Av. Copacabana, 177, sala 32~35 Empresarial 18 do Fortel
Alphaville Barueri, Sao Paulo, 06472-001
Brazil
Tel: +55-11-4191-3100
Fax: +55(11) 4191-3705 (extension#112)
Email: info.bra@tmaxsoft.com
Web (Portuguese): http://www.tmaxsoft.com/br_en/home_br_en

Russia

Tmax Rus L.L.C.
Leninsky prospekt, 113/1 (Park Place Moscow),
Office 318e, Moscow, 117198
Russia
Tel: +7(495)970-01-35
Email: info.rus@tmaxsoft.com
Web (Russian): http://www.tmaxsoft.com/ru_ru/home_ru_ru

Singapore

Tmax Singapore Pte. Ltd.
430 Lorong 6, Toa Payoh #10-02,
OrangeTee Building, 319402
Singapore
Tel: +65-6259-7223
Fax: +65-6258-7112
Email: info.sg@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/sg_en/home_sg_en

United Kingdom

TmaxSoft UK Ltd.
215 Knyvett House, Watermans Business Park,
The Causeway, Staines TW18 3BAB
United Kingdom
Tel: +44-1784-895005
Email: info.uk@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/gb_en/home_gb_en

Canada

TmaxSoft Canada, Inc.
2425 Matheson Blvd East, 8th floor,
Unit 824 Mississauga, ON, L4W 5K4
Canada
Tel: +1-905-361-2888
Email: info.canada@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/ca_en/home_ca_en

Australia

TmaxSoft Proprietary Limited
L32, 101 Miller Street, North Sydney 2060
Australia
Tel: +91-9845-330-704
Email: info.aus@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/au_en/home_au_en

India

TmaxSoft Technologies Private Limited
Sobha Alexander Plaza, 3rd Floor,
16/2 Commissariat Road, Bangalore-560025
India
Tel: +91-9845-330-704
Email: info.india@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/in_en/home_in_en

Turkey

TmaxSoft Co., Ltd. Turkey Liaison Office
Windowist Tower. Eski Buyukdere Cad. No:26,
Maslak 34467 Istanbul
Turkey
Tel: +90-544-553-6045
Email: cslee@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/tr_en/home_tr_en

第1章 セキュリティー・システムの紹介

本章では、セキュリティー・システムの概要、主要特徴および構造について説明します。

1.1. 概要

普段、セキュリティーという概念は、重要な何かに対し、外部もしくは内部からの侵入や毀損の恐れがある場合に、それを前もって防ぐことを意味します。このセキュリティーの概念は、コンピュータやエンタープライズの環境で、ユーザーの情報を扱ってサービスを提供することにも適用できます。

セキュリティーの対象になるのはユーザーの名前やパスワード、アドレスなどの個人情報、そしてサービスの提供やシステムの動作に必要なリソースなどです。各システムでは、重要な情報やリソースを保護するために、セキュリティー関連のソフトウェアを使うか、ハードウェア・レベルのセキュリティー・システムを構築します。

JEUSのセキュリティー・サービスは基本的にSPI(Service Provider Interface)に準拠しており、JEUSに登録されているリソースやユーザーの情報を保護するための多様なサービスを提供します。

本書では、JEUSセキュリティーの構造とセキュリティー情報管理の詳細について説明します。

JEUSセキュリティー・システムは以下のような目標の下で設計されました。

- 柔軟かつ交替しやすいフレームワーク

既存のサード・パーティーのセキュリティー・システムを搭載したセキュリティー・システムに効率よく統合されて多様なデータ・ストレージをサポートする。

- 機密性を保持する

有能なハッカーが悪意を持ってセキュリティー・システムのソースを逆コンパイルしたとしても、権限のないシステムにはアクセスできないようにする。

- 一定レベルの性能を維持する

システムとアプリケーションのコードにセキュリティー・システムが関与しても、全体のパフォーマンスへの影響は最小限に抑える。通常、セキュリティーが強化されれば、性能は低下してしまう嫌いがあるけれど、セキュリティー機能を最大に提供しつつも処理スピードが低下しないようにする。

- メンテナンスを容易にし、サード・パーティーのセキュリティー・サービスを簡単に作成できるように、簡潔かつ明瞭なAPIとSPIを提供する

- データを保全し、整合性を保証する

- 標準を遵守する

JEUSセキュリティ・システムは前述した設計上の目的を相当満たしています。

1.2. 主要特徴

以下は、JEUSセキュリティ・システムの主な特徴です。

- オープン・アーキテクチャーと柔軟なフレームワークを持ち、従来使用していたサード・パーティーのセキュリティ・システムに有効に統合できます。

- 動的なプリンシパルとロール(Principal-to-Role)のマッピング、ロールとリソース(Role-to-Resource)のマッピングをサポートします。

動的マッピングは、権限付与マッピング(Principal-to-Role、Role-to-Resourceマッピング)がランタイムに適用されることを意味します。

動的マッピングは、例えば次のようなセキュリティ・ポリシーを可能にします。

- ユーザーUに対して、月曜日から金曜日の間、勤務時間の9時から5時まで、Rというロールを与える
- 全員にRというロールを与える
- 誰にもRというロールを与えない

- セキュリティ・ドメインの概念をサポートします。ドメインを導入することで、異なるJ2EEアプリケーションが各々の特性に適合する別途のセキュリティ・システムを利用できるようになります。

JEUSシステムでは、ドメインとセキュリティ・ドメインは異なる概念であり、混同しないように注意してください。システムでのドメインはサーバーの管理単位であり、セキュリティ・ドメインはセキュリティの管理単位です。

- 認証、権限付与、リポジトリ、監査、クラスタリングなどの重要なセキュリティ機能をデフォルトで実装しています。

- 柔軟なイベント・ハンドリング・モデルを使ってセキュリティ監査のメカニズムをサポートします。

- サブレット、EJB、アプリケーションのソースコードにセキュリティ機能を直接追加できます。ただし、この機能は、サブジェクトやポリシーを追加するといった簡単なセキュリティ機能に限られます。

- JDK 7、JDK 8とJACC 1.5の仕様に対応しています。

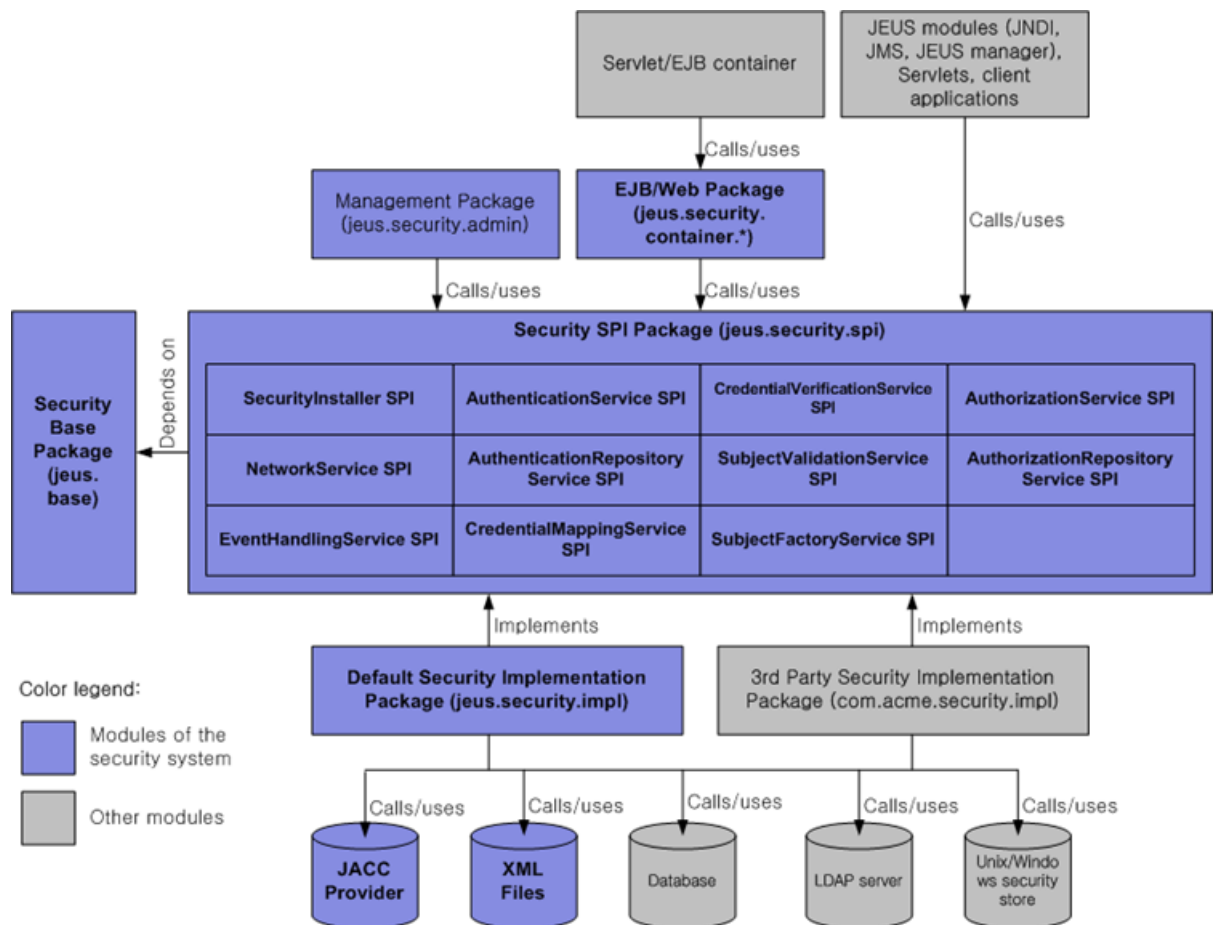
- このマニュアルの他に、JavaDocでも文書化を十分にサポートしています。

- 他のJEUSモジュールと独立しているため、JEUSシステムでセキュリティー・システムのみを切り出し、他のコンテキストに適用することが比較的簡単です。

1.3. システム構造

以下は、セキュリティー・システムの基本アーキテクチャーです。

[図 1.1] セキュリティー・システムの構造



以下では、上図に示したアーキテクチャーの主要コンポーネントについて説明します。

- jeus.security.spi

中央の大型のボックスにまとめられた部分で、セキュリティー・システムの中核をなすセキュリティーSPIのクラスです。この抽象クラスにはサード・パーティー製品で実装する抽象メソッドだけでなく、JEUSコンテナを含む、ユーザー・コードで呼び出すメソッドも含まれています。

現在使用できるSPIクラスは11個で、それぞれ認証、権限付与など、セキュリティーの中核機能を担当しています。

- jeus.security.base

左側の小型のボックスで、セキュリティSPIが参照するインターフェースと実装クラスが含まれています。このパッケージで重要な2つのクラスはサブジェクトとポリシーです。

- jeus.security.impl.*

下部のボックスで、JEUSが提供するSPIクラスのデフォルトの実装クラスが含まれています。現在のところ、デフォルトの実装クラスはXMLファイルのリポジトリとJACCプロバイダーに対応しています。

- jeus.security.admin, jeus.security.container

上部のボックスで、SPIクラスを呼び出すコードを含んでいます。

- リポジトリと外部セキュリティ・メカニズム

上図の最下部にあるのはリポジトリと外部セキュリティ・メカニズムです。

リポジトリにはデータベース、LDAPサーバー、XMLファイルがあります。リポジトリはサブジェクトやポリシーなどのセキュリティ・コンポーネントを永久に格納するために使います。

外部セキュリティ・メカニズムは認証や権限付与が実行されるメカニズムを意味します。外部セキュリティ・メカニズムの代表例はJACCプロバイダーです。通常、リポジトリとセキュリティ・メカニズムの境界を明確に区分するのは難しいです。SPI実装クラス(ここでは、jeus.security.impl.*パッケージに含まれているクラス)が、実際に適用するリポジトリとセキュリティ・メカニズムを決めます。

1.4. 主要概念

本節では、セキュリティ・システムを理解するために必須の、アーキテクチャーの重要な概念について説明します。

- [ログイン](#)
- [認証](#)
- [権限チェック\(付与\)](#)
- [セキュリティ監査](#)
- [サービスとSPI](#)
- [ドメイン](#)

1.4.1. ログイン

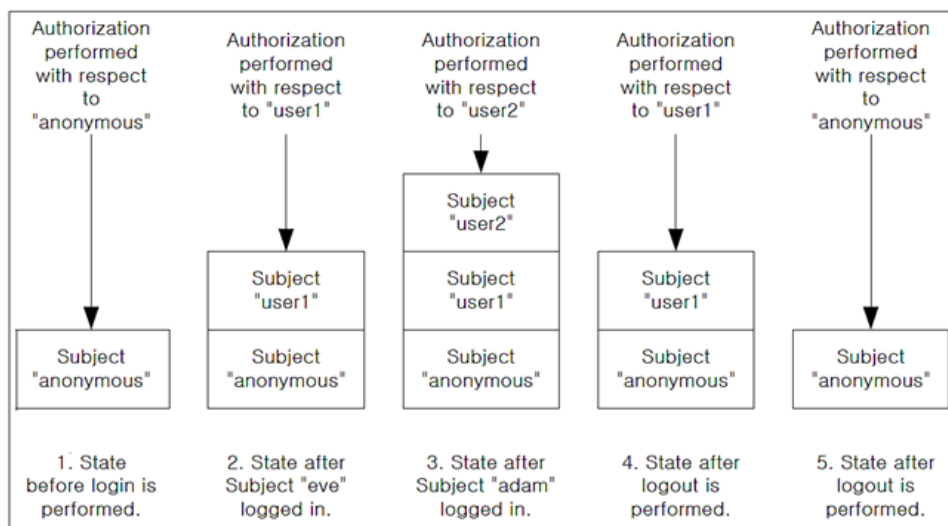
JEUSセキュリティ・システムにおいて、ログインはサブジェクトを実行スレッド(Javaスレッド)に関連付けることを意味します。これは認証とは異なります。通常、認証はサブジェクトが実行スレッドに関連付けられる前に行われます。認証に成功すればログインは正常に行われ、サブジェクトがスレッドに関連付けられますが、認証に失敗するとログインはそれ以上進みません。

サブジェクトがログインに成功すれば、そのサブジェクトに対して権限チェックが行われます。つまり、ログインは認証と権限チェックの両方に関わりがあります。ログインの反対概念としてログアウトがあり、現行のサブジェクトを実行スレッドから切り離すことを意味します。

ログインのメカニズムは、次のようなスタック・ベースのオペレーションに基づきます。複数の異なるサブジェクトでログインすると、最後にログインしたサブジェクトがスタックの一番上に追加されます。その後、権限チェックでは最後に追加されたサブジェクト(最後にログインしたサブジェクト)を使います。サブジェクトがログアウトするとスタックから除去され、その直前にログインしたサブジェクトがエクスポートされ、活性化します。

以下は、このメカニズムを図に表したものです。

[図 1.2] スタック・ベースのログイン・メカニズム



JEUSセキュリティ・システムでは、1つのJavaスレッドは1つのログイン・スタックを持ちます。ログイン・メカニズムは、jeus.security.impl.login.CommonLoginServiceによって実装されます。

1.4.2. 認証

認証は、呼び出し側の身元を確認する作業として、以降の権限チェックのために使用されます。

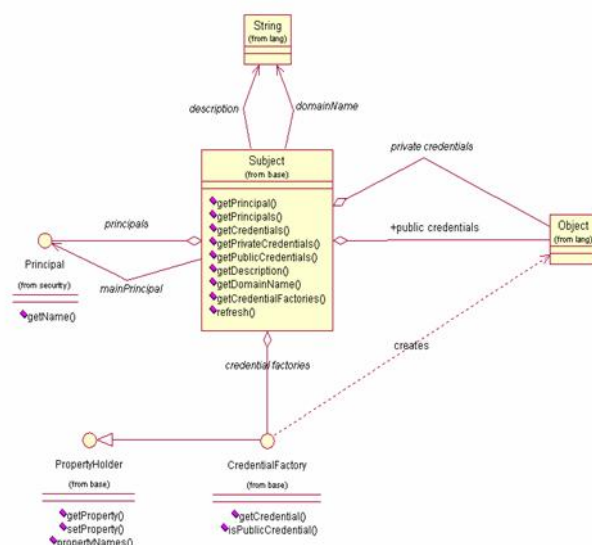
JEUSセキュリティ・システムにおける身元は、java.security.Principalインターフェースで定義したプリンシパルです。サブジェクトはプリンシパルを属性値として格納しています。サブジェクトは、jeus.security.base.Subjectクラスで表され、ログインの際に実行スレッドと関連付けられる主体でもあります。常にプリンシパルと資格証明をセキュリティ属性として含んでいます。

参考

JEUSセキュリティー・システムのサブジェクトは、`javax.security.auth.Subject`クラスが定義するJAASのサブジェクトとは異なります。しかし、この2つは多くの共通点があり、相互転換が可能です。また、一部の情報を失っても指定されたエイリアスを使用できます。

JEUSサブジェクトは、UMLを利用して以下のように表記することができます。

[図 1.3] サブジェクトのUMLダイアグラム



サブジェクトは一意的IDであるメイン・プリンシパルを保持します。メイン・プリンシパルの他に、複数の付加的なプリンシパルを持つこともできますが、このようなプリンシパルは他のサブジェクトと共有することができます。(そのため、付加的なプリンシパルをグループ・プリンシパルともいいます。)

サブジェクトはパブリックまたはプライベートの資格証明を持つことができます。資格証明は、通常、サブジェクトの身元を確認するか、特定情報を送信するために使われる証憑資料です。代表的なプライベート資格証明にはパスワードがあり、パブリック資格証明には電子証明書があります。

サブジェクトは資格証明ファクトリー(`CredentialFactory`)を使って資格証明を作成します。サブジェクトの`refresh()`メソッドが`CredentialFactory`から資格証明を取得して、それをパブリックまたはプライベートの資格証明セットに追加します。

注

サブジェクトは必ず1つのドメインに属するということに注意してください。つまり、ドメインAに属するプリンシパルの「user1」はドメインBに属するプリンシパル「user1」とは別のプリンシパルです。

1.4.3. 権限チェック(付与)

JEUSセキュリティー・システムにおける権限チェックは、すでに認証済みのサブジェクトが特定のアクションを実行する権限があるかどうかを確認することを意味します。

権限チェックは、普段、システム・レベルで行われます。たとえば、特定のサブジェクト(一般にadministrators)がJEUSサーバーを起動または終了する権限を持っているかどうかをチェックすることがその例です。アプリケーション・レベルでも権限チェックが行われることがあります。たとえば、リモートの呼び出し側が特定アプリケーション・コンポーネントにアクセス(特定のEJBメソッドの実行、特定のサーブレットの呼び出し)できるかどうかをJEUSエンジンで確認することがその例になります。

J2EEと同様に、JEUSセキュリティー・システムにおける権限付与はロール・ベース(Role-Based)のメカニズムに従います。つまり、J2EEアプリケーションのアセンブラや開発者がロールに従ってセキュリティーを設定(security restriction)すると、システム管理者がアプリケーションをデプロイするとき、実際のシステムのプリンシパルを論理的なロールにマッピングします。

注

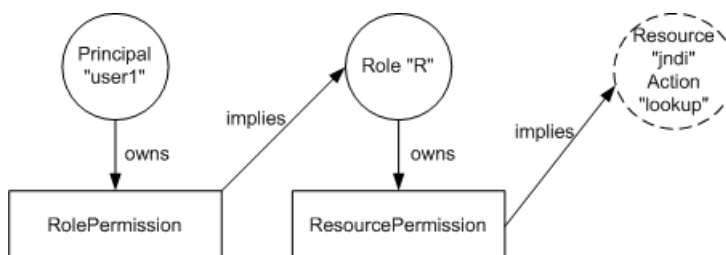
ロール・ベースのアプローチはJ2EEアプリケーションだけでなく、JEUSシステムに関連する権限チェックにも使われることに注意してください。

さらに、JEUSセキュリティー・システムでの各々の権限マッピングをパーミッション(java.security.Permissionのサブクラス)といいます。たとえば、プリンシパル「user1」が「R」というロールにアクセスできるロール・パーミッションを保持するとします。(このことを「user1」はロール「R」に含まれているといいます。)

また、ロール「R」は「jndi」リソースにアクセスし、「lookup」アクションを実行できるリソース・パーミッションを保持するとします。

以下は、上記の概念を図で表したものです。

【図 1.4】 ロール・ベースのパーミッション・チェック



上図のように、プリンシパルとロールだけが物理的なエンティティーとしてモデリングされます。上図において、リソースを囲んでいる円は点線で表されています。これは、リソースが権限付与システムにおいて物理的にモデリングされる部分ではなく、暗示される部分であるからです。

「RolePermission」と「ResourcePermission」は、それぞれjeus.security.resource.RolePermissionクラスとjeus.security.resource.ResourcePermissionクラスのインスタンスを示します。この2つのクラスは、java.security.Permissionの抽象クラスから拡張された代表的なクラスで、このほかにも多様なパーミッション・クラスのサブクラスがあります。

前述した内容をまとめると、プリンシパルは複数のロール・パーミッションを保持し、各々のロール・パーミッションは特定のロールがプリンシパルを含むことを許容します。また、ロールも複数のリソース・パーミッションを持っていますが、各々のリソース・パーミッションはロールが特定のリソースに対し特定のアクションをとることを許容します。したがって、プリンシパル、ロール、リソースを関連づけると、あるプリンシパルが特定のリソースに対し特定のアクションを実行できるかどうかを判断できます。

上図で、プリンシパルの「user1」は実線でロール・パーミッションを所有する(owns)と表示されています。ところが、ロール・パーミッションは対角線方向にロールの「R」を暗示している(implies)と表示されています。「暗示する」とは、両者の関係が静的でないという意味で、マッピングされることもあれば、そうでないこともあり得ることをいいます。つまり、ロール・パーミッションがロールを暗示するかどうかはランタイムに動的に決まります。

実際には、ロール・パーミッションのimplies(Permission p)メソッドによって決まります。implies()メソッドがtrueを返すとロール・パーミッションがロールの「R」を暗示し、falseを返すと暗示しません。したがって、前者の場合は、プリンシパル「user1」がロール「R」に含まれますが、後者の場合は含まれません。同じ論理がロール「R」とリソース「jndi」の間でも成立します。

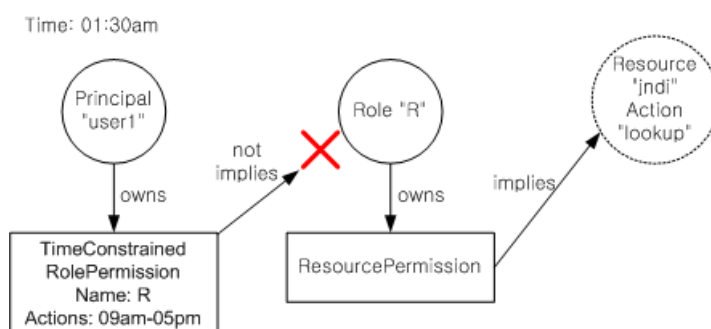
参考

パーミッションの詳細については、J2SE JavaDocのjava.security.Permission部分を参照してください。

上記の情報に基づいて動的なマッピングを実装することはさほど難しくありません。単にimplies()メソッドの実装を変更するだけで、特定の条件下でのみパーミッションがロールまたはリソースを暗示するようにすることができます。たとえば、動的なマッピングを利用して、プリンシパルの「user1」が勤務時間(9時から5時まで)の間だけ、ロール「R」に含まれるように実装することができます。方法は、ロール・パーミッションのサブクラス(以下、TimeConstrainedRolePermission)を新規作成し、implies()メソッド内で時間制約に関するコードを作成すればいいです。

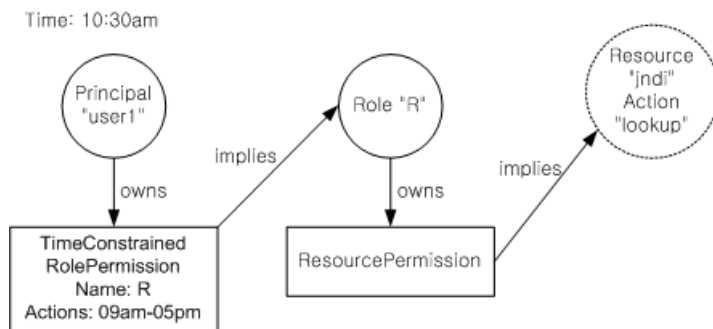
その例として、[\[図 1.5\]](#)と[\[図 1.6\]](#)を参照してください。

[図 1.5] 午前1時30分のロール



[\[図 1.5\]](#)で示すように、午前1時30分にはプリンシパル「user1」はロール「R」の権限がありません。

[図 1.6] 午前10時30分のロール



TimeConstrainedRolePermissionはNameとActionsの2つの値を持ちます。Nameはターゲット・ロールの名前である「R」であり、Actionsは有効な時間(9時から5時まで)を示します。「Name」と「Actions」はほとんどのjava.security.Permissionの実装クラスの変数で宣言されます。

基本的なパーミッション・ベースのマッピングとは別に、権限付与システムにおけるすべてのパーミッションは以下の3つのカテゴリのいずれかに属します。

区分	説明
Excluded Permission	<p>誰も当該パーミッションに対する許可を保持することができません。</p> <p>たとえば、リソース「RSC」へのアクセスをExcluded Permissionに設定すると、誰も「RSC」リソースにアクセスできなくなります。</p> <p>Excluded Permissionは、Unchecked Permissionより優先順位が高いです</p>
Unchecked Permission	<p>誰もが当該パーミッションへの許可を持つことができます。</p> <p>たとえば、リソース「RSC」に対しUnchecked Permissionを設定すると、ロールに関係なく誰もが「RSC」リソースにアクセスできます。</p> <p>Unchecked PermissionはExcluded Permissionより優先順位が低いですが、Checked Permissionよりは優先順位が高いです</p>
Checked Permission	<p>特定のプリンシパルやロールにのみ許可されたパーミッションであり、これまで説明したパーミッションに当たりません</p>

以下では、パーミッションに関する例を紹介します。

JEUSセキュリティー・システムですべてのパーミッション・マップは、jeus.security.base.PermissionMapクラスを実装したクラスです。PermissionMapクラスはjeus.security.base.Policyクラスに含まれます。

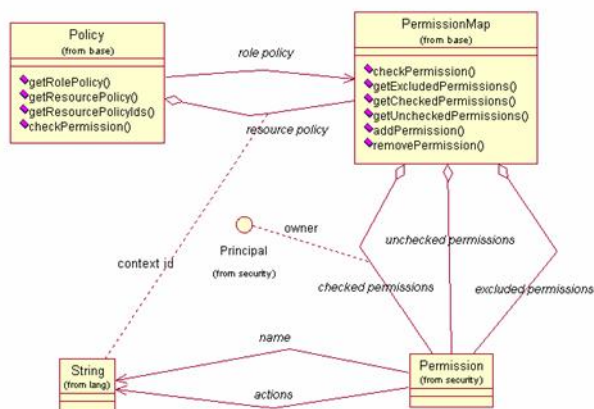
ポリシーは2種類のパーミッション・マップを持っています。

- Principal-to-Roleマップ(Role Policy)
- Role-to-Resourceマップ(Resource Policy)

これらは、それぞれPrincipal-to-RoleマッピングとRole-to-Resourceマッピングに使用されます。さらに、各パーミッション・マップは上記の3種類(Excluded、Unchecked、Checked)のパーミッションを含んでいます。Checked Permissionは、パーミッションとロール・パーミッション・マップ(RolePermissionMap)がプリンシパルまたはロールを媒介に関連付けられており、ポリシーとリソース・パーミッション・マップ(ResourcePermissionMap)は、コンテキストID(権限チェックが行われる範囲を示す)により関連付けられています。

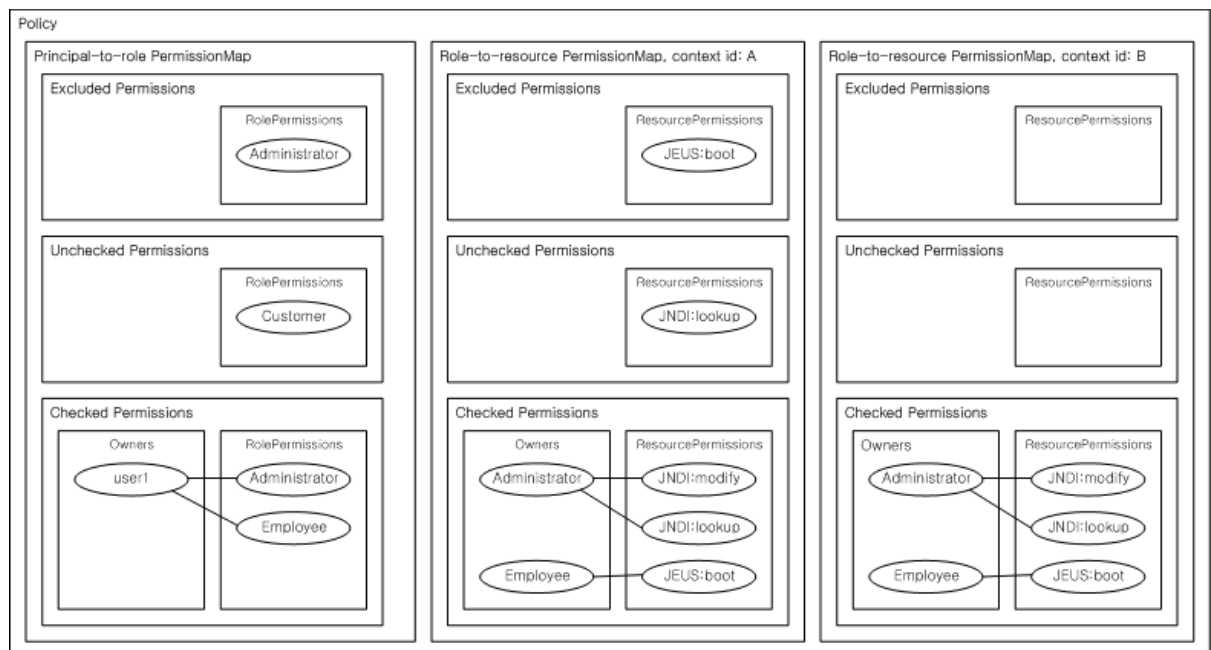
以下は、ポリシーとパーミッション・マップのUMLダイアグラムです。

【図 1.7】 ポリシーとパーミッション・マップのUMLダイアグラム



下の図は、上記内容の概略を示したものです。

【図 1.8】 1つのPrincipal-to-Roleマップと2つのRole-to-Resourceマップを持つポリシーの例



このポリシーは1つのPrincipal-to-Roleマップ(必須項目)と、各々のコンテキストIDが「A」と「B」の2つのRole-to-Resourceマップを含んでいます。3つのパーミッション・マップは、以下のようなパーミッション・セットを持っています。

- Excludedパーミッション
- Uncheckedパーミッション
- Checkedパーミッション

ボックス内の楕円形は、名前(Name)とアクション(Action)を持つ実際のパーミッション・インスタンス(Name:Action)を示しています。

コンテキストIDが「A」で、プリンシパルが「user1」のサブジェクトが「JNDI」にアクセスして「modify」というアクションを実行しようとするかと仮定します。「user1」はアクションを実行できるでしょうか。

権限付与システムはこの要求事項を以下のような方法で解決します。

1. 名前は「JNDI」で、アクションが「modify」の新しいリソース・パーミッション(RSP)を作成します。
2. RSPはコンテキストIDの「A」とプリンシパルの「user1」とともにポリシーに渡されます。
3. コンテキストID「A」のRole-to-Resourceパーミッション・マップがポリシーで選択されます。
4. ポリシーはパーミッション・マップ「A」のExcludedパーミッションにRSPが含まれるかをチェックします。チェックの結果、RSPはExcludedパーミッションではありません。
5. ポリシーはパーミッション・マップ「A」のUncheckedパーミッションにRSPが含まれるかをチェックします。チェックの結果、RSPはUncheckedパーミッションではありません。
6. ポリシーはパーミッション・マップ「A」のCheckedパーミッションにRSPが含まれるかをチェックします。チェックの結果、RSPはCheckedパーミッションのうちの1つです。
7. ポリシーはRSPを暗示するパーミッションの所有者を特定します。「Administrator」というロールがその所有者です。
8. ポリシーは「Administrator」という名前を持つロール・パーミッション(RLP)を作成します。
9. ポリシーはPrincipal-to-Roleパーミッション・マップを取得します。
10. ポリシーはPrincipal-to-RoleパーミッションのExcludedパーミッションに含まれるかどうかをチェックします。チェックの結果、RLPはExcludedパーミッションです。

11. ポリシーは「Administrator」というロールがExcludedパーミッションであるため、誰もアクセスできないという結論を出すことができます。したがって、権限チェックの全過程は終了し、「DENIED」という結果が返されます。

「user1」は「Administrator」のロールにマッピングされているけれど、「Administrator」ロール・パーミッションがExcludedパーミッションであるため、「user1」も含めて誰も「Administrator」のロールにアクセスすることができません。これは、ExcludedパーミッションがUncheckedパーミッションとCheckedパーミッションより優先順位が高いためです。

上記のとおり段階を踏むと、次の権限チェックのクエリーがどう解決されるかが分かります。

[表 1.1] 多様な権限チェックのクエリーと結果の例

Principal	Operation	Context	Outcome
user1	JNDI:lookup	A	GRANTED
user1	JNDI:modify	A	DENIED
user1	JEUS:boot	A	DENIED
user1	JEUS:boot	B	GRANTED
Anonymous	JNDI:lookup	A	GRANTED
Anonymous	JNDI:lookup	B	DENIED
Anonymous	JEUS:boot	A	DENIED
Anonymous	JEUS:boot	B	DENIED

1.4.4. セキュリティー監査

セキュリティ監査は、セキュリティ関連のイベント(認証の失敗、ランタイム例外の発生)を探索して適切に処理することで、セキュリティの全体レベルを向上させることです。

JEUSセキュリティ・システムの特徴は、セキュリティ・イベントをベースにした簡単かつ柔軟なセキュリティ監査のメカニズムにあります。主なイベント(認証の失敗、権限付与の失敗、関心度の高い他イベント)が発生するたびに、イベントは予め登録されたイベント・ハンドラーに関連づけられます。カスタム・ハンドラーの実装クラスはこれに適切に対応します。

たとえば、数回連続して認証に失敗する場合、関連サブジェクトの資格証明にロックをかけることもできます。

1.4.5. サービスとSPI

セキュリティ・システムにおいて実際のセキュリティ機能を実装しているクラスをサービスといいます。つまり、サービスは特定のセキュリティ機能(認証、権限付与、ネットワーキングなど)を提供するSPIを実装したクラスです。

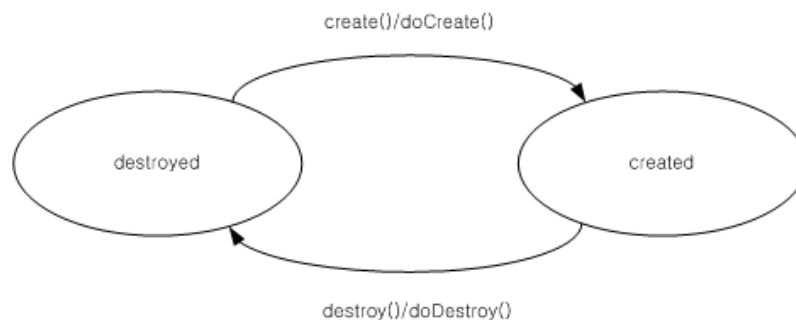
SPIは、jeus.security.spiパッケージに含まれている抽象クラスです。SPIクラスは多様に拡張することができ、SPIクラスの拡張によってカスタム・セキュリティ機能を実装できます。SPIを拡張したサブクラスが、サービスになります。

セキュリティ・システムにおけるサービス・インスタンスは一つ一つが特定のセキュリティ機能を提供する独立したエンティティとみなされます。しかし、サービスは往々にして他のSPIを呼び出すため、サービス間に一定レベルの依存性が存在します。サービスを初期化するために、すべてのサービスは「key-value」の組の属性値を渡されます。こうしたデータは設定ファイルを使って格納しておくこともできます。

すべてのサービスはオプション項目であり、JEUS管理システムに報告するJMX Beanを定義することもあります。通常、JMX MBeanはService.getMBeanInfo()メソッドから返されるMBeanInfoに基づいて作成されます。

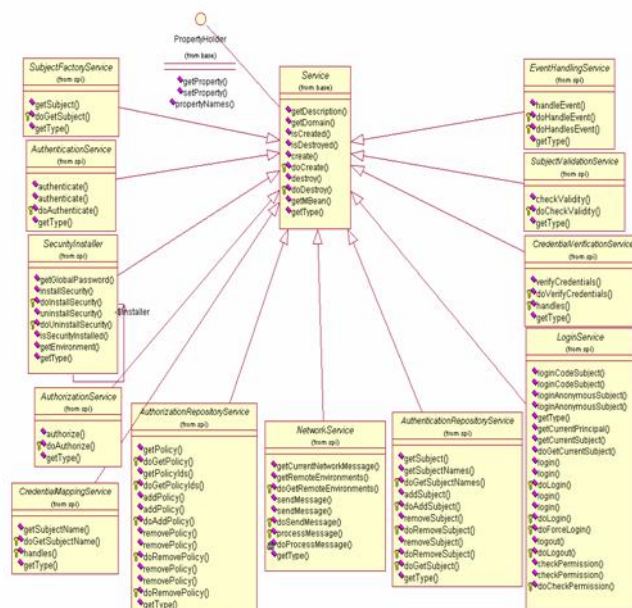
また、すべてのサービスは生成(created)と消滅(destroyed)状態を持ちます。この2つの状態間の切り替えはcreate()とdestroy()が呼び出されるときに発生します。このメソッドを呼び出すと実際の抽象メソッドであるdoCreate()とdoDestroy()メソッドが呼び出されます。これらの抽象メソッドはサービスのサブクラスで実装され、サービスを初期化し、整理する作業を含みます。

【図 1.9】 サービスの2つの状態



以下は、jeus.security.spiパッケージに含まれている多様なSPIクラスとサービス・クラスをクラス・ダイアグラムで示した図です。

[図 1.10] サービス・クラスとSPIのサブクラス



サービス・インスタンスは、ランタイムに、SecurityInstallerシングルトン・クラスによって生成されます。単純にSecurityInstallerを実装するとすれば、プログラムでサービスを直接生成するコードを作成するだけで済みますが、より柔軟かつ精巧にサービスを作成するには、サービスのクラス名と属性値を特定の設定ファイルに保管しておいて、ファイルから設定情報を読み込み、サービス・インスタンスを作って初期化する方法もあります。デフォルトのSecurityInstaller実装クラスは後者のプロセスを導入したため、簡単かつ柔軟な方法でサービスをセキュリティ・システムに追加することができます。

注

単純にJEUSセキュリティ・システム機能を使うときは、サービスやSPIアーキテクチャーをすべて理解する必要はありません。

1.4.6. ドメイン

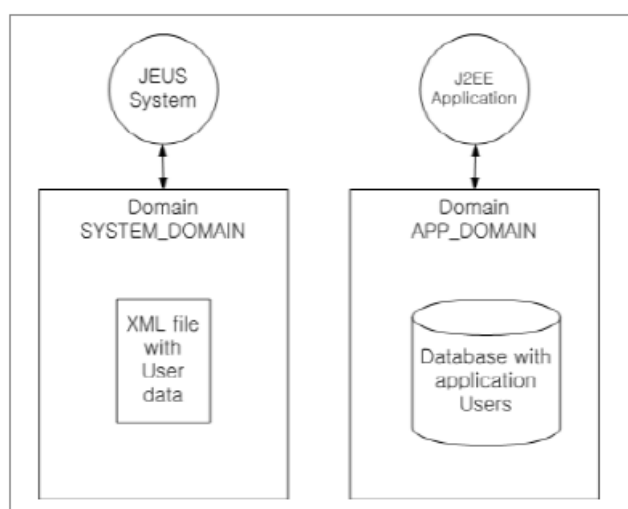
セキュリティ・ドメイン(Security Domain)は、セキュリティ・サービスのインスタンスの集合といえます。1つのセキュリティ・システムに複数のドメインが同時に存在することができます。ドメイン概念の背景には、異なるアプリケーションやJEUSサブシステムがそれぞれ別途のセキュリティ・サービスを使えるようにするという考え方があります。ドメインはセキュリティ・サービスをアプリケーション別に切り離す役割をします。

たとえば、JEUSシステムでのみ使える特別なドメインを設定することができます。JEUSが使うドメインをSYSTEM_DOMAINといいます。このドメインはJEUSを管理するために使うサブジェクトとポリシーを含んでいます。SYSTEM_DOMAINにサーバーを起動または終了できる「administrator」というメイン・サブジェクトのユーザー情報を設定することができます。

また別のドメインとして、デプロイされた特定のJ2EEアプリケーションでのみ使えるドメインを設定することもできます。これをAPP_DOMAINといいます。APP_DOMAINも「administrator」というユーザー情報を設定できますが、JEUSの「administrator」とはパーミッションが異なります。また、ドメイン別にそれぞれ異なるリポジトリのメカニズムを設定することもできます。たとえば、SYSTEM_DOMAINはユーザー情報をXMLファイルに保存し、APP_DOMAINはリモートのデータベースを使ってサブジェクトのユーザー定義を読み書きできるように設定することができます。

以下は、ドメインの概念を図で示したものです。

[図 1.11] 異なるアプリケーションと異なるサブジェクト・リポジトリを使った2つのドメイン



ドメインを別々に設定するかどうかはオプションです。ドメインを別途に設定しない場合、SYSTEM_DOMAINが使われます。

注

ドメイン名はすべて大文字で書き、通常「_DOMAIN」で終わります。これは必須の命名規則ではなく、推奨される命名規則です。この文書でいうドメインは、すべてセキュリティ・ドメインを表し、J2EEサーバー・ドメインとは関連のない別の概念です。

以下のように、JEUSは基本的にSYSTEM_DOMAINとSHARED_DOMAINの2つのドメインを使用します。

- SYSTEM_DOMAIN

標準管理ツールを使ってJEUSサーバーを管理するために使われるドメインです。起動と終了といったJEUSサーバー管理を行う「administrator」アカウントとパーミッションを含んでいます。基本的にはJ2EEアプリケーションもこのドメインを使用しますが、設定により変更できます。

- SHARED_DOMAIN

特殊なドメインであり、SHARED_DOMAINに設定されたセキュリティ・サービスは他のすべてのドメインで共有されます。したがって、SHARED_DOMAINはすべてのドメインに共通して適用可能なセキュリティ・サービスを含んでいる必要があります。

アプリケーションでJEUSシステムとは異なるセキュリティ・サービスを使用するには、別途のドメインを作成して、domain.xmlにその内容を記述する必要があります。セキュリティ・ドメインの作成および設定方法については、「[第2章 セキュリティ・システムの設定](#)」を参照してください。

1.5. 性能改善とセキュリティ・レベルの向上

セキュリティを強化すれば性能が低下する傾向があり、性能とセキュリティは常に折り合いが必要です。セキュリティ・システムが持つべき特徴に反する部分がないように、セキュリティと性能のレベルを調整し、各業務の特性に合わせて合理的に設定するようにします。

1.5.1. JEUS性能の改善

本節では、セキュリティ・システムを使いつつも、JEUSサーバーの性能の低下を最小限にとどめる方法を紹介します。

以下は、JEUSセキュリティ・システムの全般的な性能を向上させる方法です。

- **JEUSドメインでセキュア・コネクションを使わない**

セキュア・コネクション(Secured Connection)は汎用ソケットに暗号化されたパケットを格納して使います。そのため、普通のソケット通信を行うときより多くのデータを転送することになり、ネットワークの使用量が増加します。

JEUSドメインでは、セキュア・コネクションを使わないことが性能を向上させます。

- **JEUSセキュリティ・マネージャーを使わない**

セキュリティ・マネージャーを使わないと不要な演算を実行せずに済みます。セキュリティを使わないということは、権限付与の機能を使わないことを意味します(セキュリティ認証は使います)。

- **非ブロッキングI/Oを使う**

非ブロッキングI/O(Non-Blocking I/O)を使ってネットワークのブロックを軽減します。非ブロッキングI/Oは、ネットワークを通じてI/Oが発生するとき、次のI/Oが発生するのを待つブロックをなくします。

そのため、コネクションごとにI/O用のスレッドを必要とせず、CPUの使用量やFDの数も減ります。JEUSは基本的に非ブロッキングI/Oを使用しているため、ブロッキングI/Oより高い性能を発揮します。

1.5.2. セキュリティ・レベルの向上

本節では、多少性能を犠牲にしても、セキュリティ・システムのセキュリティ・レベルを向上させる方法について説明します。

以下は、JEUSセキュリティ・レベルを向上させる方法です。

- **グローバル・システム・パスワードを設定する**

- **JEUSセキュリティ・ドメインでセキュア・コネクションを使う**

ネットワーク・セキュリティ・サービスがセキュア・コネクション(通常、SSL/TLSで保護されるコネクション)を使うように設定します。デフォルトのネットワーク・サービスを利用するとき、セキュア・コネクションをどう設定するかについては、[参考文献](#)を参照してください。

- **格納されているサブジェクトとポリシーを権限のないアクセスから保護する**

サブジェクトのユーザー情報とポリシーのリポジトリを、権限のないアクセスから保護することは欠かせないことです。保護方法は、使用するリポジトリによって異なります。

- データベースを使う場合、サブジェクトのユーザー情報とポリシーを格納しているテーブルは、データベース側に設定された認証と権限付与のプロセスによって保護されます。また、JEUSセキュリティ・リポジトリとデータベース間のコネクションをSSLなどを使って保護します。この設定方法については使用するデータベースの文書を参照してください。
- セキュリティ関連の情報をaccounts.xml、policies.xmlなどのXMLファイルに保存する場合、これらのファイルにアクセスできるパーミッションを別途に作成して、JEUSセキュリティ・システムとadministratorだけがこのファイルにアクセスできるように設定する必要があります。もしファイル・リポジトリに関係なく暗号化をサポートする場合は、暗号化を使用します。ファイルにセキュリティを適用する方法は、リポジトリに関する文書を参照してください。ファイルに対して読み書きの権限を設定する方法については、使用するオペレーティング・システムのマニュアルを参照してください。

- **セキュリティ監査のメカニズムを導入する**

システムで発生したセキュリティ・イベントをログに記録し、定期的にそのログ内容をチェックすることで、全体のセキュリティ・レベルを向上させることができます。JEUSセキュリティ監査のメカニズムは通常、jeus.security.spi.EventHandling Service SPIクラスを使って実装されます。ログファイルは権限のないアクセスから保護される必要があります。セキュリティ監査の多様なメカニズムについては、[参考文献](#)を参照してください。

- **Java SE SecurityManagerを使う。**

JEUSシステムの堅牢性を高め、悪意のあるEJBとサーブレットのコードから保護するには、Java SE SecurityManagerを設定します。詳細については、「[2.7.1. JavaSE SecurityManagerの設定](#)」を参照してください。

- **サード・パーティーのセキュリティ・メカニズムを使う**

大切な情報が扱われる実環境(銀行業務向けアプリケーションなど)では、JEUS本来のセキュリティ・システム以外に、ファイアウォールやVPNs(Virtual Private Networks)といった追加セキュリティ要素をインストールすることをお勧めします。

- **オペレーティング・システム自体にセキュリティを設定する**

JEUSが動作するオペレーティング・システムの種類とは無関係に、オペレーティング・システムを保護するいくつかのガイドラインがあります。

- JEUSがインストールされているマシンへの物理的なアクセスを制限します。一例に、マシンを一箇所に置いてロックをかけます。
- 信頼が確保された管理者(administrator)のみがJEUSファイルとプロセスにアクセスするようにプロセスの権限とファイルの権限を設定します。これについては使用するオペレーティング・システムのマニュアルを参照してください。
- 最新のセキュリティ・パッチを適用してオペレーティング・システムをアップグレードします。
- アンチ・ウィルス・ソフトを周期的に動作させます。
- セキュリティを要する文書はすべて安全に保管します。たとえば、パスワードが書かれている文書はロックのかかっているキャビネットに保管します。

第2章 セキュリティー・システムの設定

本章では、JEUSでセキュリティー・システムをインストールおよび設定する方法について説明します。本章で説明するサービス以外のサービスをドメインに設定する方法は、[参考文献](#)を参照してください。

2.1. 概要

本節ではセキュリティー・システムの設定について簡単に説明します。

JEUSのセキュリティー・システムは、セキュリティー・インストーラー(Security Installer)によりサーバーを実行することで開始されます。セキュリティー・インストーラーは、domain.xmlに定義されている情報を基にセキュリティー・ドメインを作成し、セキュリティー・サービスを開始してJEUSを安全に保護します。デフォルトのセキュリティー・システムでは、accounts.xmlとpolicies.xmlに定義されているユーザー・アカウントとセキュリティー・ポリシーが適用されます。

セキュリティー・ドメインに対し、以下の2つを設定できます。

- セキュリティー・ドメインとセキュリティー・サービスの定義
- ドメインのアカウントおよびポリシーの設定

以下の手順でデフォルトのセキュリティー・システムを設定します。

1. セキュリティー・ドメインを設定します。
2. ドメイン別に構成要素とセキュリティー・サービスを設定します。
3. ドメインのサブジェクト(認証データ)のユーザー情報を設定します。
4. ドメインのポリシー(セキュリティー・ポリシー・データ)を設定します。
5. サブジェクトとポリシー以外の追加事項を設定します。
6. Java SEセキュリティー・マネージャーを設定します。(オプション)
7. JACCプロバイダーを設定します。(オプション)

デフォルトのセキュリティ・システムのディレクトリー構造

以下は、デフォルトのセキュリティ・システムのディレクトリー構造です。各ディレクトリーにはセキュリティ・システムで使う設定ファイルが含まれています。

```
JEUS_HOME/domains/<domain name>
|--config
    |--security
        |--security.key
        |--policy
        |--SYSTEM_DOMAIN
            |--accounts.xml
            |--policies.xml
```

以下は、ディレクトリーとファイルについての説明です。

security

区分	説明
security.key	対称鍵暗号化アルゴリズムのキーを保存するファイルです。最初に暗号化を実行するときにファイルが生成されます
policy	Javaパーミッションの設定ファイルです。JEUSのセキュリティ・システムとは別に、Java SEセキュリティ・マネージャーで使用されます
SYSTEM_DOMAIN	JEUSサーバーがユーザー認証と権限チェックのために使用するドメインです。このドメインはJEUSを起動および終了するためのパーミッションとJEUSシステムの管理者(administrator)アカウントを含んでいます – accounts.xml : ユーザー情報が保存されます – policies.xml : セキュリティ・ポリシー・データ(権限付与データ)が保存されます

参考

アプリケーションにJEUSシステムと異なるセキュリティ・ポリシーを適用するには、別途のセキュリティ・ドメイン・ディレクトリーを作成し、関連設定を適用した後で使用する必要があります。

2.2. セキュリティー・ドメインの定義

ドメインの設定は、WebAdminを使う方法と設定ファイルのdomain.xmlを編集する方法があります。ユーザー・アカウントとセキュリティ・ポリシーの設定を除く他の事項は動的に変更できません。したがって、セキュリティ・ドメインを追加したり、セキュリティ・サービスの設定を修正したりした場合などは、設定を適用するためにサーバーを再起動する必要があることに注意してください。

注

セキュリティに関連するすべての設定は、JEUSドメインに属するすべてのサーバーに同一に適用されます。したがって、ユーザー・アカウントとセキュリティ・ポリシーの設定を除いては、動作しているすべてのサーバーを再起動しなければ、セキュリティ設定を適用できません。

セキュリティ・ドメインを設定するためには、Domain Admin Server(以下、DAS)が起動中である必要があり、設定を適用するためには、DASを含むサーバーの再起動が必要です。本節では、WebAdminとXML編集によりセキュリティ・ドメインを定義する方法について説明します。

2.2.1. WebAdminを使用した設定

WebAdminの左のメニューから[Security]を選択すると、現在構成されているセキュリティ・ドメインの情報が表示されます。構成されているセキュリティ・ドメインのうち、アプリケーションで使用するドメインの名前を「**Default Application Domain**」項目に設定します。セキュリティ・ドメインを追加、削除することも可能です。

「**Connect Retries**」項目を設定することで、JEUSセキュリティ・ネットワーク・サービスの接続リトライ回数(connection retry count)を定義できます。

【図 2.1】セキュリティ・マネージャーのメイン画面

jeus_domain

Domain
Session
Clusters
Servers
Applications
Security
Resources
Monitoring
Console

システム状態

0 Failed
0 Standby
2 Running
0 Shutdown
0 Suspended
0 Other

Runtime Info
LOCK & EDIT

指定したドメインの項目を変更、追加、削除する機能です。

運用マニュアル
Domain
Server もっと見る

Security Manager

HISTORY

JEUSのセキュリティ情報管理についての説明です。1つのドメイン内における共通のセキュリティ情報を管理するときに設定します。

ヘルプ

動的設定 必須項目 このページの設定を変更したい場合は、左のメニューから[Lock & Edit]ボタンをクリックしてください。TIP

Connect Retries	0
セキュリティネットワークのサービス中に再接続を試行する回数を指定します。	
Default Application Domain	SYSTEM_DOMAIN
[デフォルト: SYSTEM_DOMAIN] アプリケーションのセキュリティに使用されるセキュリティドメインの名前を指定します。	

詳細設定

すべてを閉く

Password Validator

Default Validator

MinLength	
MaxLength	
Force Special Character	<input type="checkbox"/>
Force Digit	<input type="checkbox"/>
Force Capital Letter	<input type="checkbox"/>
Force Small Letter	<input type="checkbox"/>
Deny Username	<input type="checkbox"/>

Custom Validator

Class Name	
------------	--

このページの設定を変更したい場合は、左のメニューから[Lock & Edit]ボタンをクリックしてください。TIP

Security Domains

Name	The Number of Accounts	
SYSTEM_DOMAIN	3	Add Delete

2.2.2. XML編集による設定

XMLを直接編集するには、JEUS_HOME/domains/<domain name>/config/domain.xml内の以下のタグを利用します。

XMLファイルでドメインを設定する方法は、JEUS_HOME/lib/schemas/jeusディレクトリー内のjeus-domain.xsdとjeus-security.xsdのXMLスキーマに定義されています。上位タグは<security-manager>です。このタグはドメインの設定に関連する以下の下位タグを含みます。下位タグは、0個以上を設定できます。

【例 2.1】<<JEUS_HOME/domains/<domain name>/config/domain.xml>>

```
<security-manager>
  <security-domains>
    <default-application-domain>SYSTEM_DOMAIN</default-application-domain>
    <security-domain>
      <name>SYSTEM_DOMAIN</name>
      ...
    
```

```
</security-domain>
<security-domains>
</security-manager>
```

以下は、設定タグについての説明です。

タグ	説明
<connect-retries>	JEUSセキュリティ・ネットワーク・サービスの接続リトライ回数(connection retry count)値を定義します (デフォルト値：10)
<default-application-domain>	J2EEアプリケーションで使用するデフォルトのセキュリティ・ドメインの名前を定義します (デフォルト値：SYSTEM_DOMAIN)
<security-domains>	JEUSに登録するセキュリティ・ドメインを設定します。<security-domains>のセキュリティ設定ファイルの例題については、 「2.3. セキュリティ・ドメイン構成要素の設定」 を参照してください

2.2.3. ユーザー・アカウントおよびセキュリティ・ポリシーの設定

本節では、デフォルトで提供されるXMLを利用した設定方法について説明します。

ユーザー・アカウントとセキュリティ・ポリシーの設定は、XMLを利用する方法だけでなく、データベースにユーザー・アカウント情報とセキュリティ・ポリシー情報を保存して使用方法もあります。この方法については、[「2.5.3. データベースを使用した設定」](#)と[「2.6.3. データベースを利用した設定」](#)を参照してください。

以下は、ユーザー・アカウントとセキュリティ・ポリシーを設定する手順です。

1. セキュリティ・ドメインを新規設定するには、JEUS_HOME/domains/<domain name>/config/securityの配下に新しいディレクトリーを作成する必要があります。新規ディレクトリーには作成しようとするセキュリティ・ドメインと同じ名前を付けます。一般にセキュリティ・ドメイン名は大文字で記述し、「_DOMAIN」で終わらせます。たとえば、「DEFAULT_APPLICATION_DOMAIN」のようにネーミングします。

DEFAULT_APPLICATION_DOMAINドメインを生成するには、コマンド・プロンプトで、以下のコマンドを実行します。

```
$ mkdir ${JEUS_HOME}/domains/domain1/config/security/DEFAULT_APPLICATION_DOMAIN
```

参考

domain1は実際のJEUSDメイン名で代替します。

2. 新しいドメイン・ディレクトリーを作成し、そのディレクトリーの中に設定ファイルを作成します。

既存ドメインのディレクトリーから設定ファイルをコピーした後、必要に応じて設定ファイルを変更すると便利です(以下のコマンドは改行せずに連続して1ラインに書きます)。設定ファイルについては次節で説明します。

```
$ cp ${JEUS_HOME}/domains/domain1/config/security/SYSTEM_DOMAIN/*.*  
  
${JEUS_HOME}/domains/domain1/config/security/DEFAULT_APPLICATION_DOMAIN
```

JEUS_HOME/domains/<domain name>/securityの配下にSYSTEM_DOMAINがデフォルトで存在します。SYSTEM_DOMAINは、JEUSサーバーがユーザー認証と権限チェックのために使用するドメインです。このドメインは、JEUSを起動および終了するためのパーミッションとJEUSシステムの管理者(administrator)アカウントを含んでいます。

3. 新しく生成されたドメインを適用するには、JEUSを再起動する必要があります。

セキュリティ・ドメインのSYSTEM_DOMAINは、domain.xmlの設定にかかわらず、常に含まれます。基本的にJEUS_HOME/domains/<domain name>/config/securityパスの下にドメインと同じ名前のディレクトリーを作成し、ドメイン別に定義しようとするリポジトリのアカウント情報(account.xsd)とセキュリティ・ポリシー情報(policies.xsd)を定義します。ドメイン別のセキュリティ・サービスは、domain.xmlを利用して登録できます。

注

デフォルト・セキュリティ・システムでWebAdminを使ってセキュリティ・ドメインを追加した場合、前述した作業を行わなければ、SYSTEM_DOMAINに存在するアカウント情報とセキュリティ・ポリシー情報を共有します。セキュリティ・ドメインによってアカウント情報とセキュリティ・ポリシー情報を分離するためには、必ず前述した作業を行います。

2.3. セキュリティ・ドメイン構成要素の設定

本節では、セキュリティ・サービスを除く、セキュリティ・ドメインの構成要素の設定方法について説明します。

2.3.1. WebAdminを使用した設定

WebAdminの左のメニューから[Security]を選択すると、**Security Manager**画面が表示されます。Security設定画面で設定を変更するセキュリティ・ドメインを選択します。セキュリティ・ドメインは、[「2.2. セキュリティ・ドメインの定義」](#)で説明したとおり定義されている必要があります。設定を変更するには、[LOCK & EDIT]ボタンをクリックして設定変更モードに切り替えます。

[図 2.2] セキュリティー・ドメインの設定

Security Manager

HISTORY

JEUSのセキュリティ情報管理についての説明です。1つのドメイン内における共通のセキュリティ情報を管理するときに設定します。

ヘルプ

動的設定 必須項目

このページの設定を変更したい場合は、左のメニューから[Lock & Edit]ボタンをクリックしてください。

TIP

Connect Retries	0	セキュリティネットワークのサービス中に再接続を試行する回数を指定します。
Default Application Domain	SYSTEM_DOMAIN	[デフォルト: SYSTEM_DOMAIN] アプリケーションのセキュリティに使用されるセキュリティドメインの名前を指定します。

詳細設定

すべてを開く

Password Validator

Default Validator

MinLength

MaxLength

Force Special Character

Force Digit

Force Capital Letter

Force Small Letter

Deny Username

Custom Validator

Class Name

このページの設定を変更したい場合は、左のメニューから[Lock & Edit]ボタンをクリックしてください。

TIP

Security Domains

Name	The Number of Accounts	Add
SYSTEM_DOMAIN	3	Delete

Security Domainsリストから設定するドメイン名をクリックすると、セキュリティ設定画面が表示されます。セキュリティ設定画面は以下のタブで構成されます。各タブについての詳細は、以下の説明と各節の内容を参照してください。

タブ	説明
[Cache Config]	ドメインに適用されたセキュリティ・サービスで適用するキャッシュ・ポリシー値を定義します
[Key Store]	ドメインのセキュリティ・サービスに適用するキーストア・ファイルの情報を定義します
[Security Service]	セキュリティ・サービスを設定します

タブ	説明
[Custom Service]	JEUSセキュリティ・フレームワークで提供するサービスの属性に関係なく、JEUS Security APIを実装したサービス属性を持つ別のセキュリティ・サービスを登録できます。詳細については、「 2.4. セキュリティ・サービスの設定 」を参照してください
[Accounts & Policies Management]	ユーザー・アカウントとセキュリティ・ポリシーを設定します。詳細については、「 2.5. セキュリティ・システムのユーザー情報の設定 」と「 2.6. セキュリティ・システム・ポリシーの設定 」を参照してください

[Cache Config]

Cache Config画面でドメインに適用されたセキュリティ・サービスで適用するキャッシュ・ポリシーを定義できます。設定したキャッシュ値は、セキュリティ・サービスの1つのリポジトリ・サービスに適用されます。

[図 2.3] [Security Service] - [Cache Config]

[Key Store]

Keystore Config画面でドメインのセキュリティ・サービスに適用するキーストア・ファイル情報を定義します。

キーストア・ファイルに含まれている証明書情報を使って、ユーザー認証が行われます。下位の設定値がない場合、「-Djeus.ssl.*」または「-Djavax.net.ssl.*」が設定されているか確認します。何も設定されていない場合は、デフォルト値が適用されます。

[🔒 2.4] [Security Manager] - [Key Store]

Keystore Config

[HISTORY](#)

キーストアファイル及びトラストストアファイルを設定します。

Name	Cache Config	Key Store	Security Service	Custom Service	Accounts & Policies Management
<div style="float: left;">動的設定 * 必須項目</div> <div style="float: right;"> <input type="button" value="確認"/> <input type="button" value="再設定"/> <input type="button" value="削除"/> </div>					
Keystore Path	<input style="width: 100%;" type="text"/> 現在のドメインに適用するキーストアファイルのパスを定義します。				
Keystore Alias	<input style="width: 100%;" type="text" value="administrator"/> [Default: changeit] キーストアファイルのKeyEntryタイプの証明書が複数ある場合、エイリアス値を明示的に指定して該当するサーバの認証に必要な証明書を指し示します。				
Keystore Password	<input style="width: 100%;" type="password" value="....."/> <input type="button" value="入力"/> EX {DES}FQrLbQ/D8O1IDV571L28rw== [Default: changeit] 現在のドメインに適用するキーストアファイルのパスワードを設定します。パスワードを暗号化して保存するときは、{algorithm}ciphertextの形式で記述します。				
Keystore Keypassword	<input style="width: 100%;" type="password" value="....."/> <input type="button" value="入力"/> EX {DES}FQrLbQ/D8O1IDV571L28rw== 現在のドメインに適用するキーストアファイルのキーパスワードを設定します。パスワードを暗号化して保存するときは、{algorithm}ciphertextの形式で記述します。				
Truststore Path	<input style="width: 100%;" type="text"/> 現在のドメインに適用するトラストストアファイルのパスを定義します。				
Truststore Password	<input style="width: 100%;" type="password" value="....."/> <input type="button" value="入力"/> EX {DES}FQrLbQ/D8O1IDV571L28rw== [Default: changeit] 現在のドメインに適用するトラストストアファイルのパスワードを定義します。パスワードを暗号化して保存するときは、{algorithm}ciphertextの形式で記述します。				

以下は、各項目についての説明です。

項目	説明
Keystore Path	キーストア・ファイルのパスを設定します (\$JEUS_HOME)/domains/<domain name>/config/security/keystore)
Keystore Alias	キーストア・ファイルのKeyEntryタイプの証明書が複数ある場合、alias値を明示することでサーバー認証に必要な証明書を指定します
Keystore Password	キーストア・ファイルのパスワードを設定します (changeit)
Keystore Keypassword	キーストア・ファイルのキー・パスワードを設定します (デフォルト値 : <keystore-password>値と同一)
Truststore Path	トラストストア・ファイルのパスを設定します (\$JEUS_HOME)/domains/<domain name>/config/security/truststore)
Truststore Password	トラストストア・ファイルのパスワードを設定します (changeit)

[Custom Service]

Custom Service画面では、JEUSセキュリティ・フレームワークで提供するサービスの属性に関係なく、JEUS Security APIを実装したサービス属性を持つ別のセキュリティ・サービスを登録できます。JEUSで提供するセキュリティ・サービスの種類とカスタム・セキュリティ・サービスの開発に関する内容については、「[第5章 カスタム・セキュリティ・サービスの開発](#)」を参照してください。

[図 2.5] [Security Manager] - [Custom Service]



2.3.2. XML編集による設定

セキュリティ・ドメインでロードされるセキュリティ・サービスは、**domain.xml**に以下のように設定されています。

XMLファイルでサービスを設定する方法は、JEUS_HOME/lib/schemas/jeusディレクトリー内のjeus-security.xsdのXMLスキーマに定義されています。

<security-domains>タグの下位に、サービスの設定と関連する**<security-domain>**を設定します。**<security-domain>**は1つ以上設定することができ、それぞれのタグはJEUSで使用するセキュリティ・ドメインを定義します。

以下は、セキュリティ設定ファイルの例です。

[例 2.2] セキュリティ・システムのサービス設定 : <<domain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    . . .
    <security-manager>

<default-application-domain>DEFAULT_APPLICATION_DOMAIN</default-application-domain>
```

```

    <security-domains>
      <security-domain>
        <name>SYSTEM_DOMAIN</name>
        . . . . .
      </security-domain>
      <security-domain>
        <name>DEFAULT_APPLICATION_DOMAIN</name>
      </security-domain>
    </security-domains>
  </security-manager>
  . . .
</domain>

```

参考

`<security-domain>`で他のサービス・プロバイダーの情報を設定せずに`<name>`情報だけ設定すると、該当するドメインでデフォルトで提供されるセキュリティ・サービスが動作します。

以下は、`<security-domain>`の下位タグについての説明です。

- **`<name>`** (必須)

セキュリティ・ドメインの名前です。

- **`<cache-config>`** (0個以上、選択事項)

ドメインのセキュリティ・リポジトリ・サービスで適用するキャッシュのポリシー値を定義します。

タグ	説明
<code><min></code>	リポジトリ・サービスに適用するキャッシュ・エントリーの最小サイズを設定します
<code><max></code>	リポジトリ・サービスに適用するキャッシュ・エントリーの最大サイズを設定します
<code><timeout></code>	リポジトリ・サービスに適用するキャッシュ・エントリーのタイムアウトを設定します

- **`<keystore-config>`** (0個以上、選択事項)

ドメインのセキュリティ・サービスに適用するキーストア・ファイルの情報を定義します。

下位の設定値がない場合、「`-Djeus.ssl.*`」または「`-Djavax.net.ssl.*`」が設定されているかどうかを確認します。該当する値がない場合、デフォルト値が適用されます。

タグ	説明
<keystore-path>	キーストア・ファイルのパスを設定します (\${JEUS_HOME}/domains/<domain name>/config/security/keystore)
<keystore-alias>	キーストア・ファイルのKeyEntryタイプの証明書が複数ある場合、alias値を明示することでサーバー認証に必要な証明書を指定します
<keystore-password>	キーストア・ファイルのパスワードを設定します(changeit)
<keystore-keypassword>	キーストア・ファイルのキー・パスワードを設定します (デフォルト値：<keystore-password>値と同一)
<truststore-path>	トラストストア・ファイルのパスを設定します (\${JEUS_HOME}/domains/<domain name>/config/security/truststore)
<truststore-password>	トラストストア・ファイルのパスワードを設定します(changeit)

2.4. セキュリティー・サービスの設定

JEUSセキュリティ・システムは、プラグ(plug)型の認証(Authentication)サービスと権限チェック(Authorization)サービスをサポートします。本節では、セキュリティ・サービスを含むセキュリティ・ドメインの構成要素を、WebAdminまたはXML編集を通じて設定する方法について説明します。

2.4.1. WebAdminを使用した設定

[Security Service]メニューを選択すると、セキュリティ・サービスの設定画面が表示されます。セキュリティに関連する作業を行うセキュリティ・サービスはドメイン別に定義できます。セキュリティ・サービスによって、該当するセキュリティ・ドメインでの認証、権限確認などの内部動作が異なってきます。

[図 2.6] [Security Service] - [Authentication]

Repository Service

HISTORY

リポジトリサービスを設定します。

ヘルプ

NameCache ConfigKey StoreSecurity ServiceCustom ServiceAccounts & Policies Management

Authentication
Authorization
Identity Assertion
Credential Mapping
Credential Verification
Audit
Subject Validation

Repository Service
Jaas Login Config
Custom Authentication Service

動的設定
必須項目

確認

再設定

削除

Xml File Repository

認証または承認サービスのためのXMLリポジトリサービスを設定します。

Config File

XMLリポジトリファイル情報を設定します。

Filename

accounts.xml

ユーザまたはグループのセキュリティ情報を含むXML設定ファイルの名前を指定します。

Filepath

\${JEUS_HOME}/domains/jeus_domain/config/security/

ユーザまたはグループのセキュリティ情報を含むXML設定ファイルのパスを指定します。

メニュー	説明
[Authentication]	ユーザー認証と関連するセキュリティ・サービスを修正できます
[Authorization]	ユーザーの権限チェックと関連するセキュリティ・サービスを修正できます
[Identity Assertion]	証明書からユーザーの名前を取得するサービスを設定できます
[Credential Mapping]	X509証明書のトラストストア・ファイルのパスとパスワードのパスを設定できます
[Credential Verification]	ユーザーの資格証明を検査するサービスを選択できます
[Audit]	JEUSセキュリティ・フレームワークで発生するイベント情報を収集するサービスを設定できます
[Subject Validation]	ユーザー(Subject)の有効性チェックを行えます

[Authentication]

Security Serviceの[Authentication]タブを選択すると、ユーザー認証と関連するセキュリティー・サービスを修正できます。

- [Repository Service]

認証リポジトリを設定できます。

[図 2.7] [Security Service] - [Authentication] - [Repository Service]

The screenshot shows the 'Repository Service' configuration page within the 'Security Service' interface. The page has a header with 'Repository Service' and a 'HISTORY' dropdown. Below the header is a search bar and a 'ヘルプ ?' button. A message bar states 'リポジトリサービスを設定します。'. The main content area has tabs for 'Name', 'Cache Config', 'Key Store', 'Security Service' (selected), 'Custom Service', and 'Accounts & Policies Management'. Under 'Security Service', there are sub-tabs: 'Authentication' (selected), 'Authorization', 'Identity Assertion', 'Credential Mapping', 'Credential Verification', 'Audit', and 'Subject Validation'. Below these are 'Repository Service', 'Jaas Login Config', and 'Custom Authentication Service'. A legend indicates that blue icons represent '動的設定' (Dynamic Settings) and orange icons represent '必須項目' (Required Fields). Action buttons '確認' (Confirm), '再設定' (Reset), and '削除' (Delete) are at the bottom right. The 'Xml File Repository' section is active, with a description: '認証または承認サービスのためのXMLリポジトリサービスを設定します。'. A 'Config File' section follows, with the instruction 'XMLリポジトリファイル情報を設定します。'. It contains two input fields: 'Filename' with the value 'accounts.xml' and a description 'ユーザまたはグループのセキュリティ情報を含むXML設定ファイルの名前を指定します。', and 'Filepath' with the value '\${JEUS_HOME}/domains/jeus_domain/config/security/' and a description 'ユーザまたはグループのセキュリティ情報を含むXML設定ファイルのパスを指定します。'.

Name	Cache Config	Key Store	Security Service	Custom Service	Accounts & Policies Management	
Authentication	Authorization	Identity Assertion	Credential Mapping	Credential Verification	Audit	Subject Validation
Repository Service	Jaas Login Config	Custom Authentication Service				

動的設定 必須項目

確認 再設定 削除

Xml File Repository

認証または承認サービスのためのXMLリポジトリサービスを設定します。

▼ Config File

XMLリポジトリファイル情報を設定します。

Filename	accounts.xml ユーザまたはグループのセキュリティ情報を含むXML設定ファイルの名前を指定します。
Filepath	\${JEUS_HOME}/domains/jeus_domain/config/security/ ユーザまたはグループのセキュリティ情報を含むXML設定ファイルのパスを指定します。

Database Repository

認証または承認サービスのためのDBリポジトリサービスを設定します。

<input checked="" type="radio"/> Datasource Id *	<input type="text" value="datasource1"/>	リポジトリとして使用するデータソースのIDを指定します。すべてのサーバに指定したIDのデータソースがバインドされます。
<input checked="" type="radio"/> Dbdriver Config		
リポジトリとして使用するデータベースのドライバを指定します。		
Vendor *	<input type="text"/>	DBリポジトリサービスのためのDBベンダを指定します。
Driver *	<input type="text"/>	DBリポジトリサービスのためのDBドライバを指定します。
Url *	<input type="text"/>	DBリポジトリサービスのためのDBのURLを指定します。
Username *	<input type="text"/>	DBリポジトリサービスのためにDBに接続時に適用するユーザ名を指定します。
Password *	<input type="password"/> <input type="button" value="入力"/>	DBリポジトリサービスのためにDBに接続時に適用するパスワードを指定します。

Custom Repository

カスタムリポジトリサービスを設定します。 jeus.security.spi.AuthenticationRepositoryServiceまたは jeus.security.spi.AuthorizationRepositoryServiceを実装したリポジトリサービスを登録すると、当該サービスの作成時に必要なプロパティ値を設定できます。

Classname *	<input type="text" value="mypackage.MyAutenticationService"/> <small>EX mypackage.MyAutenticationService</small>	jeus.security.spiのクラスの中から1つを拡張したJavaクラス名を設定します。
Property	<input type="text" value="name=value"/> <small>EX name=value</small>	選択事項であり、該当するセキュリティサービスに名前と値(Name-Value)をペアにして属性を定義します。定義できる属性と、同属性についての説明はサービスクラスに関する文書を参照してください。

項目	説明
Xml File Repository	デフォルト値としてXMLを使用する Xml File Repository が選択されています。 Config File 領域にファイル名とパスを指定して使用します
Database Repository	チェックボックスを選択すると、データベースを認証リポジトリとして使用します。Resourcesに定義されている「 Datasource id 」を指定します
Dbdriver Config	データベースのドライバーを設定します。データベースのテーブルの構成に関する内容は、「 2.5.3. データベースを使用した設定 」を参照してください
Custom Repository	jeus.security.spi.AuthenticationRepositoryServiceを実装したカスタム・サービスを設定します

- [Jaas Login Config]

JAASと関連する設定を行えます。「**Callback Handler Classname**」にコールバック・ハンドラー・ファクトリーのクラス名を定義できます。**[Add]**ボタンをクリックして、ログイン・モジュールを追加できます。JAASに関する詳細は「[第7章 JAASの使用](#)」を参照してください。

[図 2.8] [Security Service] - [Authentication] - [Jaas Login Config]

JAAS Login Config

HISTORY

JAASログインモジュールを使用して認証を行うときに設定します。

ヘルプ

Name Cache Config Key Store **Security Service** Custom Service Accounts & Policies Management

Authentication Authorization Identity Assertion Credential Mapping Credential Verification Audit Subject Validation

Repository Service **Jaas Login Config** Custom Authentication Service

動的設定 必須項目 確認 再設定 削除

Callback Handler Classname

JAASコールバック・ハンドラ・ファクトリ・クラス名を設定します。

確認 再設定 削除

JAAS Login Config

Login Module Classname	Control Flag
該当する内容が存在しません。	

- [Custom Authentiction Service]

ユーザー認証を行うセキュリティー・サービスのクラス名を定義します。jeus.security.spi.AuthenticationServiceを実装したクラス名を設定します。デフォルト値としてJEUSで提供する「DefaultAuthenticationService」が設定されます。実装クラスによってプロパティを入力します。

[図 2.9] [Security Service] - [Authentication] - [Custom Authentication Service]

Credential Mapping

HISTORY

資格マッピングサービスを設定します。 ヘルプ

Name Cache Config Key Store **Security Service** Custom Service Accounts & Policies Management

Authentication | Authorization | Identity Assertion | **Credential Mapping** | Credential Verification | Audit | Subject Validation

動的設定 必須項目 確認 再設定 削除

Classname * jeus.security.impl.atn.DefaultAuthenticationService EX mypackage.MyAutenticationService
jeus.security.spiのクラスの中から1つを拡張したJavaクラス名を設定します。

Property EX name=value

選択事項であり、該当するセキュリティーサービスに名前と値(Name-Value)をペアにして属性を定義します。定義できる属性と、同属性についての説明はサービスクラスに関する文書を参照してください。

確認 再設定 削除

[Authorization]

Security Serviceの[Authorization]タブを選択すると、ユーザーの権限チェックと関連するセキュリティサービスを設定できます。

- [Basic]

認証リポジトリと同様に、**Xml File Repository**、**Database Repository**、**Custom Repository**から1つを選択できます。

[図 2.10] [Security Service] - [Authorization] - [Basic]

Authorization HISTORY

承認サービスを設定します。 ヘルプ

Name Cache Config Key Store **Security Service** Custom Service Accounts & Policies Management

Authentication | **Authorization** | Identity Assertion | Credential Mapping | Credential Verification | Audit | Subject Validation

Basic Custom Authorization Service

動的設定 * 必須項目 確認 再設定 削除

☐ Jacc Service ☐ JACCサービスを使用するか否かを指定します。

☒ **Repository Service**
デフォルトのリポジトリサービスを設定します。

☒ **Xml File Repository**
認証または承認サービスのためのXMLリポジトリサービスを設定します。

Filename policies.xml
ユーザまたはグループのセキュリティ情報を含むXML設定ファイルの名前を指定します。

Filepath \${JEUS_HOME}/domains/jeus_domain/config/security/
ユーザまたはグループのセキュリティ情報を含むXML設定ファイルのパスを指定します。

☒ **Database Repository**
認証または承認サービスのためのDBリポジトリサービスを設定します。

☒ **Datasource Id** *
リポジトリとして使用するデータソースのIDを指定します。すべてのサーバに指定したIDのデータソースがバインドされます。

☒ **Dbdriver Config**
リポジトリとして使用するデータベースのドライバを指定します。

Vendor *
DBリポジトリサービスのためのDBベンダを指定します。

Driver *
DBリポジトリサービスのためのDBドライバを指定します。

Url *
DBリポジトリサービスのためのDBのURLを指定します。

Username *
DBリポジトリサービスのためにDBに接続時に適用するユーザ名を指定します。

Password *
DBリポジトリサービスのためにDBに接続時に適用するパスワードを指定します。 入力

● ▼ Custom Repository

カスタムリポジトリサービスを設定します。 jeus.security.spi.AuthenticationRepositoryServiceまたは jeus.security.spi.AuthorizationRepositoryServiceを実装したリポジトリサービスを登録すると、当該サービスの作成時に必要なプロパティ値を設定できます。

Classname *	<input type="text" value="jeus.security.spi.AuthenticationRepositoryService"/> jeus.security.spiのクラスの中から1つを拡張したJavaクラス名を設定します。	EX mypackage.MyAutenticationService
Property	<input type="text" value="name=value"/> 選択事項であり、該当するセキュリティサービスに名前と値(Name-Value)をペアにして属性を定義します。定義できる属性と、同属性についての説明はサービスクラスに関する文書を参照してください。	EX name=value


確認 再設定 削除

項目	説明
Jacc Service	JACCサービスの使用の有無を設定します。 JACCを使用すると、JACCのJEUS実装のjeus.security.impl.azn.JACC AuthorizationServiceとjeus.security.impl.aznrep.JACCAuthorization RepositoryServiceが権限チェックおよびリポジトリ・サービスとして設定され、動作します。 JACCに関連する詳細は、「 第6章 JACCプロバイダーの使用 」を参照してください
Repository Service	権限チェック・リポジトリを設定します
Database Repository	データベースのテーブル構成に関する内容は、「 2.6.3. データベースを利用した設定 」を参照してください
Custom Repository	jeus.security.spi.AuthorizationRepositoryServiceを実装したカスタム・サービスのみ設定できます

- [Custom Authorization Service]

ユーザーの権限チェックを行うセキュリティー・サービスのクラス名を定義します。
jeus.security.spi.AuthorizationServiceを実装したクラス名を設定します。デフォルト値としてJEUSで提供するDefaultAuthorizationServiceが設定されます。実装クラスによってプロパティを設定できます。

[図 2.11] [Security Service] - [Authorization] - [Custom Authorization Service]



Custom Authorization Service

カスタム承認サービスを設定します。

Navigation: Name | Cache Config | Key Store | **Security Service** | Custom Service | Accounts & Policies Management

Authentication | **Authorization** | Identity Assertion | Credential Mapping | Credential Verification | Audit | Subject Validation

Basic | **Custom Authorization Service**

動的設定 * 必須項目

Classname * jeus.security.impl.azn.DefaultAuthorizationService EX mypackage.MyAutenticationService
jeus.security.spiのクラスの中から1つを拡張したJavaクラス名を設定します。

Property EX name=value

選択事項であり、該当するセキュリティーサービスに名前と値(Name-Value)をペアにして属性を定義します。定義できる属性と、同属性についての説明はサービスクラスに関する文書を参照してください。

確認 再設定 削除

[Identity Assertion]

Identity Assertion画面で証明書からユーザーの名前を取得するサービスを設定できます。jeus.security.spi.IdentityAssertionServiceを実装したクラスのうち1つを選択します。

[図 2.12] [Security Service] - [Identity Assertion]

Identity Assertion

HISTORY

IDアサーションサービスを設定します。

ヘルプ

NameCache ConfigKey StoreSecurity ServiceCustom ServiceAccounts & Policies Management

AuthenticationAuthorizationIdentity AssertionCredential MappingCredential VerificationAuditSubject Validation

動的設定 必須項目 確認 再設定 削除

Default Identity Assertion Service

現在のドメインにデフォルトで適用するIDアサーションサービスを設定します。

X509 Identity Assertion

X509証明書を使用したIDアサーションサービスを設定します。

Filename

ユーザまたはグループのセキュリティ情報を含むXML設定ファイルの名前を指定します。

Filepath

ユーザまたはグループのセキュリティ情報を含むXML設定ファイルのパスを指定します。

Default User Mapper

X509証明書トークンに対するユーザマッピングを行うための情報を設定します。

Cert Attr Type

資格マッピングのためのプロパティ値を設定します。

Attribute Type

資格マッピングのためのプロパティ値を設定します。

Attribute Value Delimiter

資格マッピングのためのプロパティ値を設定します。

Custom User Mapper

X509証明書トークンに対するユーザマッピングを行うための追加属性を設定します。

Property

name=value

資格マッピングのためのプロパティ値を設定します。

Custom Identity Assertion Service

証明書とユーザ間のマッピング情報を定義した設定ファイルのパスを指定します。

Classname

mypackage.MyAutenticationService

jeus.security.spiのクラスの中から1つを拡張したJavaクラス名を設定します。

Property

name=value

選択事項であり、該当するセキュリティサービスに名前と値(Name-Value)をペアにして属性を定義します。定義できる属性と、同属性についての説明はサービスクラスに関する文書を参照してください。

Kerberos Identity Assertion

ケルベロス認証を使用する場合に設定します。

Kerberos Realm Name	<input type="text"/>	サーバが属するケルベロス領域の名前を指定します。指定していない場合は、java.security.krb5.realmで設定された名前、またはkrb5.confファイルに設定された名前を使用します。
Service Principal	<input type="text"/>	サーバサービスのプリンシパルを設定します。デフォルト値は「krbtgt/{realm-name}@{realm-name}」です。
Service Password *	<input type="password"/>	設定したサーバサービスのプリンシパルのパスワードを指定します。

項目	説明
Default Identity Assertion Service	デフォルト値としてJEUSが提供するDefaultIdentityAssertionServiceの設定を変更できます。このサービスはX509証明書をベースに動作します。 「Filename」、「Filepath」設定により証明書とユーザー名のマッピング情報が含まれているXMLファイルを指定できます
Custome Identity Assertion Service	カスタムIDアサーション・サービスを定義します。SPIを直接実装したクラスを定義します
Kerberos Identity Assertion	Kerberosプロトコルを利用したアサーションを設定できます。 設定を行うと、JEUSで提供するログイン・モジュール実装のうち、jeus.security.impl.login.KerberosSharedStateLoginModuleが有効になります

[Credential Mapping]

Credential Mapping画面でX509証明書のトラストストアとパスワードのパスを指定できます。

デフォルト値として`jeus.security.spi.CredentialMappingService`の基本実装である`jeus.security.impl.credmap.JKSCertificateCredentialMappingService`サービスが有効になります。

【図 2.13】 [Security Service] - [Credential Mapping]

Credential Mapping HISTORY

資格マッピングサービスを設定します。 ヘルプ

Name Cache Config Key Store **Security Service** Custom Service Accounts & Policies Management

Authentication | Authorization | Identity Assertion | **Credential Mapping** | Credential Verification | Audit | Subject Validation

動的設定 * 必須項目 確認 再設定 削除

Default Credential Mapping Service
デフォルトの資格マッピングサービスを設定します。

X509 Credential Mapping
X509証明書を使用したIDアサーションサービスを設定します。

Truststore Path
現在のドメインに適用するトラストストアファイルのパスを指定します。

Truststore Password 入力
[Default: changeit] 現在のドメインに適用するトラストストアファイルのパスワードを指定します。パスワードを暗号化し
存する場合は、{algorithm}ciphertextの形式で記述します。例) (DI) (DES)FQrLbQ/D801IDVS71L28rw==

確認 再設定 削除

Custom Credential Mapping Service
カスタム資格マッピングサービスを設定します。

Classname Add

該当する内容が存在しません。

[Credential Verification]

Credential Verification画面でユーザーの資格証明を検査するサービスを選択できます。

デフォルト値として「**Password Verification**」がチェックされ、ユーザーのパスワードを検査します。「**Jeus Certificate Verification**」を選択すると、X509証明書の検査を有効にできます。

Credential Verificationリストからjeus.security.spi.CredentialVerificationServiceを実装したクラスを追加して使用することもできます。

[図 2.14] [Security Service] - [Credential Verification]

Credential Verification HISTORY

資格検証サービスを設定します。 ヘルプ ?

Name Cache Config Key Store **Security Service** Custom Service Accounts & Policies Management

Authentication | Authorization | Identity Assertion | Credential Mapping | **Credential Verification** | Audit | Subject Validation

動的設定 * 必須項目 確認 再設定 削除

Default Credential Verification Service
デフォルトの資格検証サービスを設定します。

Password Verification	<input checked="" type="checkbox"/> PasswordFactoryクラスを使用した資格情報の検証を設定します。
Jeus Certificate Verification	<input type="checkbox"/> X509証明書を使用した資格情報の検証を設定します。

確認 再設定 削除

Custom Credential Verification Service
カスタム資格検証サービスを設定します。

Classname	Add
該当する内容が存在しません。	

[Audit]

Audit画面でJEUSセキュリティ・フレームワークで発生するイベント情報を収集するサービスを設定できます。

「Filename」、「Filepath」にログを残すファイル名とパスを設定でき、「Audit Level」にログレベルを設定できます。基本実装のjeus.security.impl.auditlog.BasicAuditLogFileServiceが使用されます。

Custom Audit Serviceリストからjeus.security.spi.EventHandlingServiceを実装したクラスを追加して使用することもできます。

[図 2.15] [Security Service] - Audit

Audit HISTORY

監査サービスを設定します。

ヘルプ ?

Name Cache Config Key Store **Security Service** Custom Service Accounts & Policies Management

Authentication | Authorization | Identity Assertion | Credential Mapping | Credential Verification | **Audit** | Subject Validation

動的設定 必須項目 確認 再設定 削除

Default Audit Service
デフォルトの監査サービスを設定します。

Audit Level
監査ログレベルを設定します。

Config File
監査ログファイル情報を設定します。

Filename
ユーザまたはグループのセキュリティ情報を含むXML設定ファイルの名前を指定します。

Filepath
ユーザまたはグループのセキュリティ情報を含むXML設定ファイルのパスを指定します。

確認 再設定 削除

Custom Audit Service
カスタム監査サービスを設定します。

Classname Add

該当する内容が存在しません。

[Subject Validation]

Subject Validation画面でユーザー(Subject)の有効性チェックの有無を設定できます。「**Default Subject Validation Service**」を選択すると、jeus.security.impl.expiration.SubjectExpirationValidationServiceサービスが有効になり、ExpiryTimeの設定に従ってユーザーの有効性チェックを実行します。

Custom Subject Validation Serviceリストからjeus.security.spi.SubjectValidationServiceを実装したクラスを追加して使用することもできます。

[図 2.16] [Security Service] - [Subject Validation]

The screenshot shows the 'Subject Validation' configuration page. At the top, there's a 'HISTORY' dropdown and a search bar. Below that, a message says 'サブジェクト認証サービスを設定します。' (Set subject authentication service). A navigation bar includes 'Name', 'Cache Config', 'Key Store', 'Security Service' (selected), 'Custom Service', and 'Accounts & Policies Management'. A sub-navigation bar lists various services, with 'Subject Validation' selected. Below this, there are tabs for '動的設定' (Dynamic Settings) and '必須項目' (Required Items). The main section has a 'Default Subject Validation Service' checkbox, which is currently unchecked, with a note 'デフォルトのサブジェクト認証サービスを設定します。' (Set the default subject authentication service). To the right are '確認' (Check), '再設定' (Reset), and '削除' (Delete) buttons. Below this is the 'Custom Subject Validation Service' section, which has a message 'カスタムサブジェクト認証サービスを設定します。' (Set custom subject authentication service). It features a table with a 'Classname' column and an 'Add' button. A message '該当する内容が存在しません。' (No matching content exists) is displayed below the table.

2.4.2. XML編集による設定

セキュリティー・ドメインでロードされるセキュリティー・サービスは、**domain.xml**に以下のように設定されています。

以下は、セキュリティー設定ファイルの例です。

[例 2.3] セキュリティー・システムのサービスの設定 : <<domain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  .
  .
  .
  <security-manager>
    <default-application-domain>
      DEFAULT_APPLICATION_DOMAIN
    </default-application-domain>
  </security-manager>
</domain>
```

```

    </default-application-domain>
    <security-domains>
      <security-domain>
        <name>SYSTEM_DOMAIN</name>
        <authentication>
          <repository-service>
            <xml-file-repository>
              <config-file>
                <filename>accounts.xml</filename>
                <filepath>
                  ${JEUS_HOME}/domains/domain1/config/security/
                </filepath>
              </config-file>
            </xml-file-repository>
          </repository-service>
        </authentication>
      </security-domain>
      <security-domain>
        <name>DEFAULT_APPLICATION_DOMAIN</name>
      </security-domain>
    </security-domains>
  </security-manager>
  . . .
</domain>

```

以下は、<security-domain>の下位タグについての説明です。

- **<name>** (必須設定事項)

セキュリティー・ドメインの名前です。

- **<authentication>** (0個以上、選択事項)

ドメインで適用する認証サービスを定義します。

- **<repository-service>**

認証のためのユーザー情報のリポジトリ・タイプに合わせてサービスを定義します。

タグ	説明
<xml-file-repository>	ユーザー情報がXMLファイルに定義されている場合に設定します
<database-repository>	ユーザー情報がデータベースに定義されている場合に設定します
<custom-repository>	jeus.security.spi.AuthenticationRepositoryService SPIを継承してユーザー情報をロードする方式を拡張し、リポジトリ・サービスを実装して適用する場合に設定します

– **<jaas-login-config>**

ドメインに適用するJAASサービスを登録します。

- **<callback-handler-class>** : JAASコールバック・ハンドラー・ファクトリーのクラス名を定義します。
- **<login-module>** : ログイン・モジュール関連内容を設定します。

タグ	説明
<login-module-classname>	ログイン・モジュールを実装したパッケージを含むクラス名を定義します
<control-flag>	次の4つの属性のうち1つを定義して全般的に認証スタック(authentication stack)を調整します <ul style="list-style-type: none">– required– requisite– sufficient– optional
<option>(0個以上)	ログイン・モジュールを初期化する場合に適用する属性値を定義します

– **<custom-authentication-service>**

jeus.security.spi.AuthenticationService SPIを継承してデフォルトで提供される認証サービスを拡張して適用したい場合に定義します。設定については、「[カスタム・セキュリティ・サービス](#)」を参照してください。

• **<authorization>** (0個以上、選択事項)

ドメインに適用する権限付与サービスを定義します。

– **<repository-service>**

権限付与のためのポリシー情報のリポジトリ・タイプに合わせてサービスを定義します。

タグ	説明
<xml-file-repository>	ポリシー情報がXMLファイルに定義されている場合に設定します
<database-repository>	ポリシー情報がデータベースに定義されている場合に設定します
<custom-repository>	jeus.security.spi.AuthorizationRepositoryService SPIを継承してポリシー情報をロードする方式を拡張し、リポジトリ・サービスを実装して適用する場合に設定します

– **<custom-authorization-service>**

jeus.security.spi.AuthorizationService SPIを継承してデフォルトで提供される認証サービスを拡張適用したい場合に定義します。設定については、「[カスタム・セキュリティ・サービス](#)」を参照してください。

– **<jacc-service>**

JACC 1.5ベースの権限付与サービスを適用する場合に設定します。

● **<identity-assertion>** (0個以上、選択事項)

ドメインに適用するIDアサーション(IdentityAssertion)サービスを定義します。

– **<default-identity-assertion-service>**

JEUSセキュリティ・フレームワークで提供するデフォルトのIDアサーション・サービスを定義します。

- **<x509-identity-assertion>** : X509証明書トークンのIDアサーション・サービスをサポートします。

タグ	説明
<config-file>	X509証明書トークンに対し、ユーザー・マッピングをするための情報を定義したファイルの場所を指定します (デフォルト値: DOMAINフォルダーの下位に位置するuser-cert-map.xml)
<default-user-mapper>	X509証明書トークン値に属性タイプ(cert-attr-type)または属性タイプ(attribute-type)、属性値に対するデリミタ(attribute-value-delimiter)値を定義します

– **<custom-identity-assertion-service>**

jeus.security.spi.IdentityAssertionService SPIを継承したIDアサーション・サービスを拡張実装して定義します。設定については、「[カスタム・セキュリティ・サービス](#)」を参照してください。

● **<credential-mapping>** (0個以上、選択事項)

ドメインに適用する資格証明マッピング(CredentialMapping)サービスを定義します。

– **<default-credential-mapping-service>**

JEUSセキュリティ・フレームワークで提供するデフォルトの資格証明マッピング・サービスをサポートします。

- **<x509-credential-mapping>** : X509証明書の資格証明マッピング・サービスをサポートします。

タグ	説明
<truststore-path>	現ドメインに適用するトラストストア・ファイルのパスを定義します
<truststore-password>	現ドメインに適用するトラストストア・ファイルのパスワードを定義します

– <custom-credential-mapping-service>

jeus.security.spi.CredentialMappingService SPIを継承した資格証明マッピング・サービスを拡張実装して定義します。設定については、「[カスタム・セキュリティ・サービス](#)」を参照してください。

● <credential-verification> (0個以上、選択事項)

ドメインに適用する資格証明検証(CredentialVerification)サービスを定義します。

– <default-credential-verification-service>

JEUSセキュリティ・フレームワークで提供するデフォルトの資格証明検証サービスをサポートします。

タグ	説明
<password-verification>	PasswordFactoryクラスに対する検証サービスを定義します
<jeus-certificate-verification>	X509証明書に対する検証サービスを定義します

– <custom-credential-verification-service>

jeus.security.spi.CredentialVerificationService SPIを継承した資格証明検証サービスを拡張実装して定義します。設定については、「[カスタム・セキュリティ・サービス](#)」を参照してください。

● <audit> (0個以上、選択事項)

JEUSセキュリティ・フレームワークで発生するイベントの情報を収集できるようにドメインに適用するイベント・ハンドリング・サービス(EventHandlingService)を定義します。

タグ	説明
<default-audit-service>	イベントの情報を記録するファイルのパスとイベントログのレベルを定義します – config-file : ログファイルのパス – audit-level : ログレベル
<custom-audit-service>	jeus.security.spi.EventHandlingServiceService SPIを継承した監査サービスを拡張実装して定義します。設定は、「 カスタム・セキュリティ・サービス 」の設定方式に従います

カスタム・セキュリティ・サービス

以下は、カスタム・セキュリティ・サービスを設定する方法です。

- **<classname>** (必須)

カスタム・セキュリティ・サービスを実装したJavaのクラス名です。このクラスは、パラメータが存在しない、デフォルトのpublicコンストラクターを持ち、jeus.security.spiパッケージのSPIクラスを継承するか、jeus.security.base.Serviceクラスを直接継承する必要があります。

- **<property>** (0個以上)

jeus.security.base.PropertyHolderインターフェース(jeus.security.base.Serviceクラスが実装)を使ってセキュリティ・サービスにname-value組の属性を設定することができます。属性は各セキュリティ・サービスの初期化に使用されます。

以下の2つの下位タグを持ちます。

タグ	説明
<name>	属性名を設定します
<value> (選択事項)	属性名に該当するString属性値を設定します

JEUSがデフォルトで提供するセキュリティ・サービスの種類とカスタム・セキュリティ・サービスの開発に関連する内容は、「[第5章 カスタム・セキュリティ・サービスの開発](#)」を参照してください。

2.5. セキュリティ・システムのユーザー情報の設定

デフォルトのセキュリティ設定でユーザー・データはaccounts.xmlファイルから読み込みます。

以下は、ファイルを保存するパスです。

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/accounts.xml
```

<domain name>はドメイン名であり、<security domain name>はユーザーが管理されるセキュリティ・ドメインの名前です。

2.5.1. WebAdminを使用した設定

以下は、WebAdminを使ってユーザー情報を設定する手順です。

1. **[Accounts & Policies Management] > [accounts]**メニューを選択してユーザー情報を設定できます。ユーザー情報は動的に設定を変更できる項目です。**[Add]**、**[Delete]**ボタンをクリックしてユーザーを追加、削除できます。

[図 2.17] Accounts設定画面

Accounts

HISTORY

アカウントの設定を定義します。このページでユーザとグループを定義することができます。

ヘルプ

Name Cache Config Key Store Security Service Custom Service Accounts & Policies Management

Accounts > Policies

Users

Name	Password	Group	Description	Command
administrator	***** ***	Administrators		<input type="button" value="Lock User"/> <input type="button" value="Unlock User"/> <input type="button" value="Delete"/>

Groups

Name	Subgroup	Description
Administrators		A group for administrators

Groupsリストでグループを追加、削除することができます。定義されたグループにユーザーを含めるには、**Users**項目で編集します。**[Add]**、**[Delete]**ボタンをクリックしてユーザーを追加、削除できます。

2. **User**リストで**[Add]**ボタンをクリックすると、**User**画面が表示されます。当画面では、JEUSで提供するデフォルトのセキュリティ・システムに使用されるaccounts.xmlファイルの修正のみできます。

[図 2.18] Accounts - ユーザーの登録

User

HISTORY

ユーザを定義します。

ヘルプ

Name Cache Config Key Store Security Service Custom Service Accounts & Policies Management

Accounts > Policies

動的設定 * 必須項目

確認 再設定

Name * user1
ユーザ名を定義します。ユーザのプライマリIDとして一意である必要があります。

Password
ユーザのパスワードを設定します。

Description
ユーザについて記述します。

Group
☐ Administrators
ユーザが属しているグループのIDを定義します。このグループは<groups><group><name></name></group></groups>に定義されている必要があります。

確認 再設定

3. 「Password」項目の右の[入力]ボタンをクリックすると、以下のようにダイアログが表示され、パスワードを設定できます。暗号化アルゴリズムを選択し、パスワードを設定した後、[確認]ボタンをクリックします。

【図 2.19】Accounts - パスワードの設定

パスワードの設定が完了すると、以下の画面でユーザー・パスワードが暗号化アルゴリズムによって変更されたことが確認できます。暗号化に関する詳細は、「2.5.4. パスワードのセキュリティ設定」を参照してください。

【図 2.20】Accounts - パスワードの暗号化

4. 「Group」項目でユーザーが属するグループを指定した後、[確認]ボタンをクリックします。

2.5.2. XML編集による設定

accounts.xmlのXMLスキーマは**accounts.xsd**であり、JEUS_HOME/lib/schemas/jeusのパスにあります。

accounts.xmlファイルには、最上位タグの<accounts>下に<users>および<groups>タグがあり、0個以上の<user>および<group>タグが含まれています。それぞれはユーザーとグループを示します。

[例 2.4] セキュリティー・システムのユーザー情報の設定 : <accounts.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>user1</name>
      <password>{AES}mnG6ItxF0/WQ1E2YzIZ7sA==</password>
      <group>group1</group>
    </user>
    <user>
      <name>user2</name>
      <password>{DES}7dJ0KDTGQNpSnQAPYBNmA==</password>
    </user>
    <user>
      <name>user3</name>
      <password>{DES}7dJ0KDTGQNpSnQAPYBNmA==</password>
      <group>nested_group</group>
    </user>
    <user>
      <name>user4</name>
      <password>{SEED}dl2EePMAcnxPYbIyknuZkA==</password>
      <group>nested_group</group>
    </user>
  </users>
  <groups>
    <group>
      <description>Group1</description>
      <name>group1</name>
      <subgroup>nested_group</subgroup>
    </group>
    <group>
      <description>For NestedGroup</description>
      <name>nested_group</name>
    </group>
  </groups>
</accounts>
```

以下は、設定タグについての説明です。

- <user>

各<user>タグは以下のような下位のタグを持ちます。

タグ	説明
<description> (選択事項)	<user>についての説明です(文字列)
<name>(必須)	<user>の名前を示します(例: ユーザー名、ユーザーID)
<password>(0個以上、選択事項)	<user>のパスワードを示します。 <password>はある特定のアルゴリズムを使用してエンコードされた値を定義します。<password>タグに適用されるアルゴリズムまたはエンコード方式を「{ }」ブロックの間に記入します。エンコード方式の詳細については、「 2.5.4. パスワードのセキュリティ設定 」を参照してください
<group>(0個以上、選択事項)	<user>が含まれたグループを示します。1ユーザーは複数のグループに含まれることができ、以下の<group>タグで定義されたグループを指す必要があります

- <group>

各<group>タグは以下のような下位タグを持ち、グループを定義します。

タグ	説明
<description>(選択事項)	<group>の説明です(文字列)
<name>(必須事項)	<group>の名前を示します
<subgroup> (0個以上、選択事項)	当該グループのロールを同一に付与できるように、ネストされたグループの名前を定義します

2.5.3. データベースを使用した設定

本節では、JEUSが定義したデータベースのテーブルを利用した認証(authentication)を設定する方法について説明します。

以下の例題は、JEUSに定義したデータソースを利用する際に設定する方法です。

[例 2.5] データベースを使用したユーザーの設定 : <<domain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    . . .
    <security-manager>
        <seuciry-domains>
            <security-domain>
```

```

        <name>MY_DOMAIN</name>
        <authentication>
            <repository-service>
                <database-repository>
                    <datasource-id>auth</datasource-id>
                </database-repository>
            </repository-service>
        </authentication>
        . . .
    </security-domain>
</security-domains>
</security-manager>
. . .
</domain>

```

上記の例で、**<datasource-id>**には認証(authentication)に使用するdomain.xmlに登録したデータソースのIDを指定する必要があります。または、ドライバー・マネージャー(DriverManager)を使って、JEUSで提供するJDBCを介さずに直接データベースと通信しながら認証を利用することもできます。この方法を利用する場合は、上記の例で**<datasource-id>**タグを以下のように変更します。

以下の例は、JEUS JDBCを利用しない場合に、データベース・リポジトリを設定する方法です。

[例 2.6] JEUS JDBCを使用しない場合の設定 : <<domain.xml>>

```

<security-manager>
    ...
    <repository-service>
        <dbdriver-config>
            <vendor>oracle</vendor>
            <driver>oracle.jdbc.OracleDriver</driver>
            <url>jdbc:oracle:thin:@127.0.0.1:1521:ORA9I</url>
            <username>scott</username>
            <password>{base64}dGlnZXI=</password>
        </dbdriver-config>
    </repository-service>
    ...
</security-manager>

```

参考

この方法を利用すると、コネクションの接続/切断が頻繁に行われるため、性能に影響する恐れがあります。そのため、JEUS JDBC(<datasource-id>)を利用する方法を推奨します。

データベースで使われるテーブルは以下のように構成されます。

[図 2.21] サブジェクトのユーザー情報を保存するためのデータベース・テーブルの構造

jeus_group_members		
PK	domain	VARCHAR(30)
PK	groupname	VARCHAR(100)
PK	username	VARCHAR(30)

jeus_users		
PK	domain	VARCHAR(30)
PK	username	VARCHAR(30)
	password	VARCHAR(100)

jeus_groups		
PK	domain	VARCHAR(30)
PK	groupname	VARCHAR(100)
	subgroups	VARCHAR(100)
	description	VARCHAR(100)

データベースにテーブルが存在しない場合、JEUS_HOME/templates/security/dbrealm.sql.templateファイルを使ってデータベースにテーブルを作成すると、ユーザー名とパスワードが「jeus」のユーザーが新規作成されます。

2.5.4. パスワードのセキュリティ設定

本節では、パスワードのセキュリティ設定のうち、パスワードの妥当性検査、暗号化アルゴリズム、SecretKeyファイルの管理方法について説明します。

パスワードの妥当性検査

ユーザーが新規アカウントを作成するためにパスワードを入力するか、既存のパスワードを変更する場合、パスワードの妥当性を検査して制約を与えることができます。一般的に使われる制約(アルファベット、数字、特殊文字の使用可否やIDとパスワードを同一にしないことなど)を利用してパスワードのセキュリティを強化することができます。

ドメイン管理者(Domain administrator)は、基本的に提供されるDefaultPasswordValidatorまたはユーザーが直接実装するCustomPasswordValidatorのいずれかを選択して、**domain.xml**に設定します。

1. <DefaultPasswordValidator>

以下は、パスワードの妥当性検査をDefaultPasswordValidatorに設定した例です。

[例 2.7] パスワードの妥当性検査をDefaultPasswordValidatorに設定 : <<domain.xml>>

```
<domain>
  . . .
  <password-validator>
    <default-validator>
      <minLength>4</minLength>
      <maxLength>10</maxLength>
      <force-special-character>true</force-special-character>
      <force-digit>true</force-digit>
    
```

```

        <force-capital-letter>true</force-capital-letter>
        <force-small-letter>true</force-small-letter>
        <deny-username>true</deny-username>
    </default-validator>
</password-validator>
. . .
</domain>

```

- <minLength> : パスワードの最小文字数を設定
- <maxLength> : パスワードの最大文字数を設定
- <force-special-character> : パスワードに必ず特殊文字を使用するように設定
- <force-digit> : パスワードに必ず数字を使用するように設定
- <force-capital-letter> : パスワードに必ず大文字を使用するように設定
- <force-small-letter> : パスワードに必ず小文字を使用するように設定
- <deny-username> : パスワードとIDが異なるように設定

2. <CustomPasswordValidator>

CustomPasswordValidatorは、ユーザーがPasswordValidatorインターフェースを直接実装して、目的の機能が含まれたクラスを実装した後、jarファイルで圧縮して「\$JEUS_TARGET/jeus/lib/system」に格納する必要があります。

以下は、ユーザーが実装するPasswordValidatorインターフェースです。

```

public interface PasswordValidator {
    boolean validatePassword(String id, String password);
}

```

PasswordValidatorインターフェースを実装したクラスのvalidatePassword(String id, String password)メソッドは、アカウントの作成やパスワードの変更を試みるユーザーのIDとパスワードを引数として受け取り、パスワードの妥当性検査を実行した後、基準に適した場合はtrueを、そうでない場合はfalseを返します。

以下は、パスワードの妥当性検査をCustomPasswordValidatorに設定した例です。

[例 2.8] パスワードの妥当性検査をCustomPasswordValidatorに設定した例 : <<domain.xml>>

```

<domain>
. . .
<password-validator>

```



```

        <custom-validator>
            <class-name>this.is.MyPasswordValidator</class-name>
        </custom-validator>
    </password-validator>
    . . .
</domain>

```

- <class-name> : ユーザーがPasswordValidatorインターフェースを実装した後、jarファイルで圧縮して「\$JEUS_TARGET/jeus/lib/system」に格納したクラスの名前

上記のXML編集設定は、WebAdminを使用して設定することも可能です。WebAdminのメイン画面の**[Security]**タブを選択して、**Password Validator画面**で設定します。ただし、変更内容を適用するためには、サーバーを再起動する必要があります。

[図 2.22] WebAdminでパスワードの検証を設定する画面

The screenshot shows the 'Password Validator' configuration page. It has a header with a warning icon and the text '詳細設定' (Detailed Settings) and a link 'すべてを開く' (Open all). The main content area is divided into two sections: 'Default Validator' and 'Custom Validator'. The 'Default Validator' section contains several settings: 'MinLength' and 'MaxLength' (text input fields), 'Force Special Character', 'Force Digit', 'Force Capital Letter', 'Force Small Letter', and 'Deny Username' (checkboxes). The 'Custom Validator' section contains a 'Class Name' (text input field). At the bottom right, there are three buttons: '確認' (Check), '再設定' (Reset), and '削除' (Delete).

暗号化アルゴリズム

ユーザー設定にはパスワードの設定が不可欠です。パスワードをplain-text形式で保存することも可能ですが、暗号化アルゴリズムにより暗号化して保護するのが望ましいです。

参考

安定性が立証された対称鍵アルゴリズムのAESを使用することを推奨します。

jeusadminの**set-password**コマンド、またはWebAdminを使ってユーザー別にパスワードを暗号化することができ、JEUS_HOME/bin/encryptionを使って直接暗号化することもできます。

直接暗号化した場合には、パスワードを設定するとき以下の形式で設定します。

{アルゴリズム}暗号化された文字列

対称鍵暗号化アルゴリズムの鍵値は、JEUS_HOME/domains/<domain name>/config/security/security.keyファイルに保存されます。

以下は、パスワード暗号化でサポートするアルゴリズムについての説明です。

項目	説明
AES/DES/DESeed/SEED/Blowfish	対称鍵暗号化アルゴリズムです
base64	エンコード・アルゴリズムです。Base64でエンコードされた情報は誰でも簡単にデコードすることができ、セキュリティ面で安全ではありません
SHA	ハッシュ・アルゴリズムです。復号化が不可能です

暗号化アルゴリズムを使用する際、鍵のサイズを指定することができます。鍵のサイズは管理者がシステム・プロパティとして管理し、システム全体に統一された鍵のサイズが適用されます。

鍵サイズのデフォルト値は256ビットであり、システム・プロパティとしてjeus.security.keylengthオプションを指定してサイズを変更することができます。たとえば、「-Djeus.security.keylength=256」オプションを指定すると、256ビットの鍵サイズを持つ暗号化アルゴリズムを使用することができます。

システム・プロパティとして設定した鍵のサイズが暗号化アルゴリズムがサポートする最大サイズより大きい場合は、暗号化アルゴリズムがサポートできる最大値で設定されます。たとえば、AESは鍵のサイズとして128、192、256ビットのみサポートするので、AES512のような設定はAES256で適用されます。

指定した鍵のサイズが暗号化アルゴリズムがサポートできる最大サイズ以下ではあるがサポート外である場合は、EncryptionException(暗号化例外)が発生します。たとえば、AESは鍵のサイズとして128、192、256ビットのみサポートするので、AES200のような設定はEncryptionExceptionが発生します。

注

新しくシステム・プロパティを設定した後は、必ずパスワードを初期化してください。

マスター・パスワードによるSecretKeyファイルの管理

暗号化ツール(JEUS_HOME/binに存在)を使ってパスワードを暗号化する場合、その暗号化ファイルに適用される秘密鍵の情報をsecurity.keyファイルに暗号化アルゴリズム別に保存して管理します。マスター・パスワードの入力を受け、このパスワードでsecret.keyファイルを暗号化して保存することができます。

security.keyファイルは以下のパスにあります。ドメイン環境を構成する場合、当該security.keyファイルを一緒に他のノードに移す必要があります。

```
JEUS_HOME/domains/<domain name>/config/security/security.key
```

JEUSに登録するデータベースのパスワードを暗号化した場合、クライアントではこれを復号化(decrypt)するために鍵が必要です。この際、SecretKeyファイルのパスをシステム・プロパティを利用して設定できます。

また、security.keyファイルにマスター・パスワードが設定された場合、マスター・パスワードもシステム・プロパティを利用して設定できます。キー・パスを指定するプロパティ名はjeus.security.keypathであり、マスター・パスワードを設定するプロパティ名はjeus.security.masterです。このプロパティはJEUSを起動するときにも利用できます。ただ、セキュリティ上の理由からマスター・パスワードはプロンプト(スタンダード入力)で入力することを推奨します。

2.5.5. ログイン情報のキャッシュ機能

JEUSスクリプトを使ってサーバーを起動するときや、jeusadminを使ってサーバーに接続するときは、ユーザー情報の入力が必要になりますが、ユーザー情報を毎回入力せずに、ログイン情報をキャッシュして使用する機能を提供します。

注

この情報はUSER_HOME/.jeusadmin/.jeuspasswdファイルにBase64でエンコードされ、保存されます。ファイルにはユーザーの重要な情報であるIDとパスワードが保存されますが、Base64エンコーディングはセキュリティ面で安全ではないため、本機能の使用は推奨しません。

ログイン情報は、一度でも認証が行われたらファイルにキャッシュされます。

jeusadminの**connect**コマンドを実行するとき、ユーザー情報と一緒に**-cachelogin**オプションを追加すると、ログイン情報がファイルに保存されます。JEUSスクリプトの場合は、**<domain name>:<server name>**をキーとしてログイン情報が保存されます。

後でJEUSスクリプトが実行されるときに、ユーザー情報を入力しなくても、ドメイン名とサーバー名が同じログイン情報があれば、自動でユーザー情報が使用されます。キャッシュされたログイン情報が存在しても、ユーザーが直接ユーザー情報を入力した場合には、キャッシュされたログイン情報は無視されます。jeusadminの場合には、接続するサーバーの<host>:<port>をキーとしてログイン情報を保存します。

以下の例は保存されたログイン情報を示します。

[例 2.9] 保存されたログイン情報 : <<.jeuspasswd>>

```
#Warning: We don't recommend to use this on Production Environment.
localhost:9736 amV1czpqZXVz
domain1:server1 amV1czpqZXVz
```

保存されたログイン情報は、jeusadminのオフライン・コマンドの**remove-login-cache**コマンドで削除できます。

2.6. セキュリティー・システム・ポリシーの設定

デフォルト・セキュリティーを設定するとき、ポリシー・データ(権限付与データ)は**policies.xml**ファイルから読み込みます。

このファイルは以下のパスに存在します。

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/policies.xml
```

<domain name>はドメインの名前を、<security domain name>はポリシーが適用されるセキュリティー・ドメインの名前を示します。

2.6.1. WebAdminを使用した設定

[Accounts & Policies Management]タブで**[policies]**メニューを選択すると、セキュリティー・ポリシーに関する設定を行えます。Role Permissions領域では、プリンシパルとロールのマッピングを定義できます。1つのロールに複数のプリンシパルをマッピングすることができ、マッピングされたプリンシパルは該当するロールに対する権限を持ちます。

[Add]ボタンをクリックしてプリンシパルとロールのマッピングを追加できます。

[図 2.23] ポリシー設定メイン画面

Policy

HISTORY

JEUSのポリシーを定義します。

ヘルプ ?

Name Cache Config Key Store Security Service Custom Service Accounts & Policies Management

Accounts | Policies

Role Permissions

Role	Principal	Actions	Classname	Add
AdministratorsRole	administrator			Delete
jndiUser	anonymous			Delete

Resource Permissions

Context ID	Add
default	Delete

注

ユーザー情報の設定と同じく、この画面ではJEUSで提供するデフォルトのセキュリティー・システムに使用されるpolicies.xmlファイルの修正だけ提供されることに注意してください。

ロール・パーミッションの登録

Role Permissionsリストで[Add]ボタンをクリックすると、ロール・パーミッションを登録できます。

「Role」に名前を指定した後、そのロールの権限を付与するプリンシパルを選択します。誰もロールにアクセスできないようにするには「Excluded」にチェックを入れ、誰でもロールにアクセスできるようにするには「Unchecked」を選択します。情報の登録が完了すると、[確認]ボタンをクリックします。

[図 2.24] ポリシーの設定 - ロール・パーミッションの登録

Role Permissions

HISTORY

ポリシーのプリンシパル-ロールマッピングを定義します。

ヘルプ

Name Cache Config Key Store Security Service Custom Service Accounts & Policies Management

Accounts Policies

動的設定 必須項目 確認 再設定

Role *	<input type="text" value="deployRole"/> プリンシパルに付与するロール名を指定します。
Principal *	<input type="checkbox"/> administrator <input checked="" type="checkbox"/> user1 <input type="checkbox"/> Administrators ロールに該当するユーザプリンシパルを指定します。
Actions	<input type="text"/> ロールパーミッションオブジェクトのアクションを定義します。基本的に使用されるロールパーミッションには、決められたアクションがありません。
Classname	<input type="text"/> 使用するロールパーミッションのクラス名を指定します。指定していない場合は、JEUSがデフォルトで提供するクラスが使用されます。
Excluded	<input type="checkbox"/> ロールの使用を無効にします。
Unchecked	<input type="checkbox"/> 何のチェックも行わずにロールが使用できるように設定します。

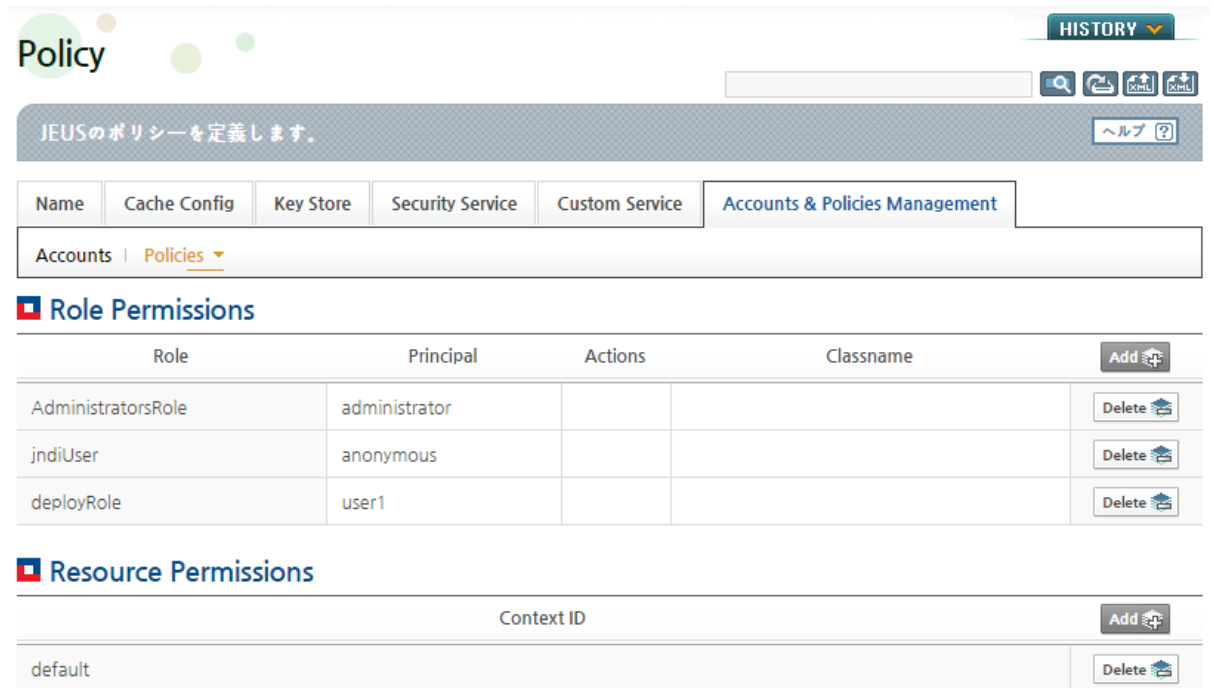
確認 再設定

リソース・パーミッションの登録

以下は、リソース・パーミッションを登録する手順です。

1. ロールに実際の動作権限であるリソース権限を与えるには、**Resource Permissions**領域で「default」を選択します。コンテキストIDはセキュリティー・ポリシーを区分するIDとして使用され、「default」はJEUSのデフォルトのセキュリティー・システムで使用するセキュリティー・ポリシーのコンテキストIDです。

[図 2.25] ポリシーの設定 - リソース・パーミッションの登録(1)



Policy

JEUSのポリシーを定義します。

Accounts | Policies

Role Permissions

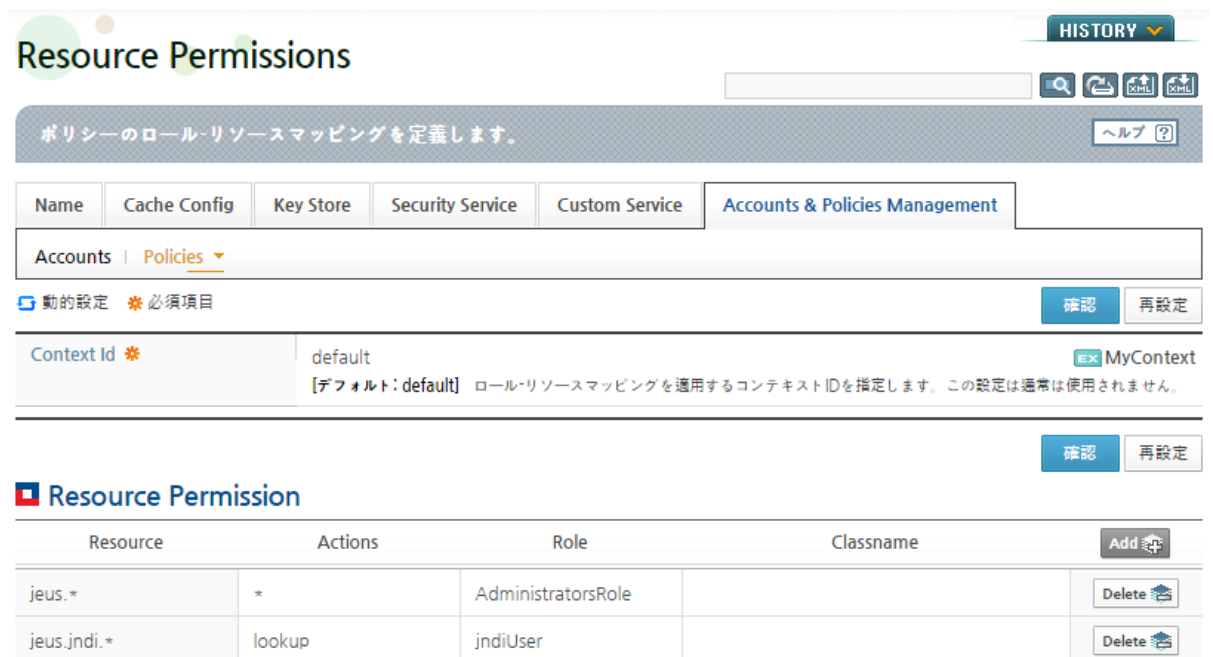
Role	Principal	Actions	Classname
AdministratorsRole	administrator		
jndiUser	anonymous		
deployRole	user1		

Resource Permissions

Context ID
default

2. 「default」コンテキストIDでJEUSシステムに対するリソース権限を設定します。**Resource Permissions**リストで[Add]ボタンをクリックしてロールとリソースのマッピングを追加できます。

[図 2.26] ポリシーの設定 - リソース・パーミッションの登録(2)



Resource Permissions

ポリシーのロール-リソースマッピングを定義します。

Accounts | Policies

動的設定 * 必須項目

Context ID * default [デフォルト: default] ロール-リソースマッピングを適用するコンテキストIDを指定します。この設定は通常は使用されません。

Resource Permission

Resource	Actions	Role	Classname
jeus.*	*	AdministratorsRole	
jeus.jndi.*	lookup	jndiUser	

3. **Resource Permissions**画面でリソースの名前とアクションを設定した後、リソースに対する権限を付与するロールを選択し、**[Add]**ボタンをクリックします。ロール・パーミッションの設定と同じく、誰もリソースにアクセスできないようにするには「**Excluded**」にチェックを入れ、誰でもリソースにアクセスできるようにするには「**Unchecked**」を選択します。JEUSのリソース権限のリストについては、「[付録B. JEUSサーバーのパーミッション](#)」を参照してください。

[図 2.27] ポリシーの設定 - リソース・パーミッションの登録(3)

The screenshot shows the 'Resource Permissions' configuration page. At the top, there's a 'HISTORY' button and a search bar. Below that, a message says 'ポリシーのロール-リソースマッピングを定義します。' (Define the role-resource mapping for the policy). There are tabs for 'Name', 'Cache Config', 'Key Store', 'Security Service', 'Custom Service', and 'Accounts & Policies Management'. The 'Accounts & Policies Management' tab is active, showing 'Accounts' and 'Policies' sub-tabs. Below the tabs, there are buttons for '動的設定' (Dynamic Settings), '必須項目' (Required Items), '確認' (Confirm), and '再設定' (Reset). The main form has several sections:

- Resource**: A text field containing 'jeus.*' with a hint 'jeus.server.*'. Below it, a note says 'ロールをマッピングするリソースを指定します。' (Specify the resource to map the role to).
- Actions**: A text field containing 'deployapplication' with a hint 'boot,down'. Below it, a note says 'ResourcePermissionクラスのコンストラクタに渡すアクション値を指定します。' (Specify the action value to pass to the constructor of the ResourcePermission class).
- Role**: A list of roles with checkboxes: 'AdministratorsRole', 'jndiUser', and 'deployRole' (which is checked). A hint 'Administrator' is shown. Below it, a note says 'リソースにマッピングされるロールを指定します。' (Specify the role to be mapped to the resource).
- Classname**: A text field with a hint 'jeus.security.resource.TimeConstrainedResourcePermission'. Below it, a note says '[Default: jeus.security.resource.ResourcePermission] java.security.Permissionを継承したJavaクラス名を指定します。このクラスはリソースパーミッションに使用されます。' (Specify the Java class name that inherits from java.security.Permission. This class is used for resource permissions).
- Excluded**: A checkbox that is unchecked. Below it, a note says 'この設定を使用した場合、リソースへのアクセスが拒否されます。' (If this setting is used, access to the resource is denied).
- Unchecked**: A checkbox that is unchecked. Below it, a note says 'この設定を使用した場合、リソースへのパーミッションがチェックされず、誰でもリソースにアクセスできます。' (If this setting is used, the permission for the resource is not checked, and anyone can access the resource).

At the bottom right, there are '確認' (Confirm) and '再設定' (Reset) buttons.

2.6.2. XML編集による設定

policies.xmlのXMLスキーマは**policies.xsd**で、JEUS_HOME/lib/schemas/jeusディレクトリー内にあります。

policies.xmlは0個以上の<policy>タグで構成され、各タグは個別ポリシー(権限付与データ)を示します。

[例 2.10] セキュリティー・システムのポリシーの設定 : <<policies.xml>>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<policies xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
```

```

<role-permissions>
  <role-permission>
    <principal>user1</principal>
    <role>administrator</role>
  </role-permission>
  <role-permission>
    <principal>user1</principal>
    <role>teller</role>
    <actions>9:00-17:00</actions>
    <classname>
      jeus.security.resource.TimeConstrainedRolePermission
    </classname>
  </role-permission>
</role-permissions>
<resource-permissions>
  <context-id>default</context-id>
  <resource-permission>
    <role>teller</role>
    <resource>bankdb</resource>
    <actions>select, update</actions>
  </resource-permission>
  <resource-permission>
    <role>administrator</role>
    <resource>jndi</resource>
    <actions>modify, delete</actions>
  </resource-permission>
  <resource-permission>
    <resource>jndi</resource>
    <actions>lookup</actions>
    <unchecked/>
  </resource-permission>
  <resource-permission>
    <role>administrator</role>
    <resource>jeus.*</resource>
    <actions>*</actions>
  </resource-permission>
</resource-permissions>
</policy>
</policies>

```

以下は、設定タグについての説明です。

- **<role-permissions>** (0または1)

Principal-to-Roleマッピングの情報を提供します。下位のタグとして0個以上の<role-permission>タグを持っています。

<role-permission>タグは以下のタグで構成されています。

タグ	説明
<principal> (0個以上)	現在<role-permission>を所有しているプリンシパルの名前です
<role>(1個、必須事項)	ロールの名前を示します
<actions>(選択事項)	ロールのアクションを示します
<classname>(選択事項)	ロール・パーミッションとして使われるjava.security.Permissionを実装したJavaクラス名です。これを省略すると、jeus.security.resource.Role Permissionがデフォルトで使われます。カスタム・パーミッションを設定および実装する方法については、「 カスタム・パーミッションの実装および設定 」を参照してください。
<excluded>(選択事項)	空のタグ(<xxx/>)です。このタグを設定すれば、該当するロール・パーミッションが排除されます。つまり、パーミッションが暗示するロールに誰もアクセスできません
<unchecked>(選択事項)	空のタグです。このタグを設定すれば、ロール・パーミッションがチェックされません。つまり、パーミッションが暗示するロールに誰もがアクセスできます

- **<resource-permissions>** (0個以上)

Role-to-Resourceマッピングの情報は以下のタグで構成されています。

- **<context-id>** (選択事項)

文字列でコンテキストIDを示します。コンテキストとは権限チェックが行われる範囲です。JEUSシステムがJEUSリソースに対して使用するデフォルトのコンテキストIDは「default」です。

- **<resource-permission>** (0個以上)

以下の項目で構成されています。

タグ	説明
<role> (0個以上)	現在リソース・パーミッションを所有しているロールの名前です
<resource>(1個、必須事項)	リソースの名前を示します
<actions>(選択事項)	リソースに対するアクションを示します
<classname>(選択事項)	リソース・パーミッションとして使われるjava.security.Permissionを実装したクラス名です。これを省略すると、jeus.security.resource.Resource Permissionがデフォルトで使われます。カスタム・パーミッションを生成および設定する方法については、 カスタム・パーミッションの実装および設定 を参照してください。
<excluded>(選択事項)	空のタグです。このタグを設定すれば、該当するリソース・パーミッションが排除されます。つまり、パーミッションが暗示するリソースに誰もアクセスできません

タグ	説明
<unchecked>(選択事項)	空のタグです。このタグを設定すれば、該当するリソース・パーミッションをチェックしません。つまり、パーミッションが暗示するリソースに誰もがアクセスできます

通常、policies.xmlはJEUSサーバー・リソースのパーミッション(JNDI、JMS、セキュリティー・サーバーなど)を設定するために使われます。一方、J2EEアプリケーションとJ2EEモジュールのパーミッションの設定には使用されません。J2EEアプリケーションとモジュールにパーミッションを設定するためには、多様なDDファイルを使います。詳細については、「[第3章 アプリケーションとモジュールのセキュリティー設定](#)」を参照してください。

カスタム・パーミッションの実装および設定

パーミッション(ロール・パーミッション、またはリソース・パーミッション)をpolicies.xmlファイルに追加するたびに、パーミッションを示すJavaクラス名を設定する必要があります。このクラスはjava.security.Permissionの抽象クラスを拡張したものです。自分だけのjava.security.Permission実装クラスを作成し、それをpolicies.xmlの<classname>タグに設定することで、カスタム・パーミッションを直接作成することができます。

以下の要求事項を満たしている、新しいカスタム・パーミッションを実装する方法について説明します。新しいロール・パーミッションの場合、以下のような2つの条件が満たされる中では、本来のロール以外のロールを暗示できます。

- 新規のロールも本来のロールと同一の名前を持ちます。
- 現在の時間が特定の時間の範囲(午前9時から午後5時まで)内に属するときのみ、他ロールを暗示することができます。

たとえば、バンキング・アプリケーションで「user2」というプリンシパルが午前9時から午後5時までの勤務時間の間だけ「teller」のロールを持つようにすることができます。

作成方法は以下のとおりです。

1. カスタム・パーミッションのクラスを実装します。クラスの実装が完了すると、「javac」でコンパイルし、JEUSサーバーのクラスパスに該当するクラスパスを設定します。

以下は、上記の要求事項を満たすカスタム・パーミッション・クラスを実装したものです。詳細なコードは省略します。

[例 2.11] カスタム・パーミッション・クラス : <<TimeConstrainedRolePermission.java>>

```
package jeus.security.resource;

import java.security.Permission;
import java.util.Calendar;
import java.util.StringTokenizer;
```

```

/**
 * A time-constrained Role permission.
 * <p>
 * With this permission implementation you can express
 * things such as "X can only be in the role Y between
 * 09:00 AM to 05:00 PM".
 */
public class TimeConstrainedRolePermission extends RolePermission {
    private String timeConstraint = "";
    private int startTime = Integer.MIN_VALUE;
    private int endTime = Integer.MAX_VALUE;

    public TimeConstrainedRolePermission(String roleName) {
        this(roleName, "");
    }

    public TimeConstrainedRolePermission(String roleName, String timeConstraint)
    {
        super(roleName);
        if (timeConstraint != null) {
            this.timeConstraint = timeConstraint;
            parseTimeConstraint();
        }
    }

    private void parseTimeConstraint() {
        . . .
    }

    public boolean equals(Object anotherObject) {
        . . .
    }

    public int hashCode() {
        . . .
    }

    public String getActions() {
        return timeConstraint;
    }

    public boolean implies(Permission anotherPermission) {
        if (this.timeConstraint.equals("")) {
            return super.implies(anotherPermission);
        } else {

```

```

        Calendar cal = Calendar.getInstance();
        int curHour = cal.get(Calendar.HOUR_OF_DAY);
        int curMinute = cal.get(Calendar.MINUTE);
        int now = curHour * 60 + curMinute;
        if (now >= this.startTime && now <= this.endTime) {
            return super.implies(anotherPermission);
        } else {
            return false;
        }
    }
}

```

当該クラスはjeus.security.resource.RolePermissionを継承します。そして、jeus.security.resource.RolePermissionはjava.security.Permissionを継承しています。これはコードの再活用性を高めるための構造です。

TimeConstrainedRolePermissionを継承して他のjava.security.Permissionを作成することも可能です。

上記のソースには、2つのタイプのコンストラクターがあります。

- 一つ目は、(role name)1個のパラメータのみを受信します。
- 二つ目は、(role name, time constraint)2個のパラメータを受信します。

java.security.Permissionにおいて、一番目のパラメータは「name」を意味し、二番目は「actions」を意味します。いくつかのパーミッションの実装クラスでは、「actions」パラメータを受信する二番目のタイプのコンストラクターが省略されることもあります。

– 「actions」パラメータはパーミッションの有効時間であり、「09:00-17:00」の値を持ちます。

– 「name」とは「administrator」または「teller」のようなロール名です。

ソースの核心は「implies(Permission anotherPermission)」メソッドです。システムの現時間が与えられた有効時間内にあるかどうかをチェックした後、その時間内にあればsuper.implies()メソッドを呼び出し、そうでなければ「false」を返します。すべてのimplies()メソッドはブーリアン値を返しますが、これは、当該パーミッションがパラメータとして渡されたパーミッションを暗示するかどうかを示します。

参考

1. implies()メソッドとjava.security.Permission抽象クラスの詳細については、JavaSE JavaDoc文書を参照してください。
 2. jeus.security.resource.RolePermission、jeus.security.resource.TimeConstrainedRolePermission、jeus.security.resource.ResourcePermissionの詳細については、[参考文献](#)とJEUS JEUS Security JavaDocでjeus.security.resourceパッケージを参照してください。
-

2. policies.xmlに該当パーミッションを使用するように設定します。

[例 2.12] カスタム・パーミッション・クラス : <<policies.xml>>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<policies xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <role-permissions>
      <role-permission>
        <principal>user2</principal>
        <role>administrator</role>
        <actions>9:00-17:00</actions>
        <classname>
          jeus.security.resource.TimeConstrainedRolePermission
        </classname>
      </role-permission>
    </role-permissions>
    <resource-permissions>
      <context-id>default</context-id>
      <resource-permission>
        <role>administrator</role>
        <resource>jeus.*</resource>
        <actions>*</actions>
      </resource-permission>
    </resource-permissions>
  </policy>
  . . .
</policies>
```

「user2」というユーザーは「administrator」ロールを午前9時から午後5時までの勤務時間の間だけ付与されます。「administrator」のロールは、すべてのJEUSリソース(jeus.*)に対してすべてのアクション(*)を実行できる権限を持ちます。したがって、user2は勤務時間に限って、全JEUSリソースに対し、すべての権限を行使することができます。

上記のスキーマはJ2EEアプリケーションとモジュールでJEUS DDファイルを設定する際もそのまま適用され、例題で示したPrincipal-to-Roleパーミッションに適用され、Role-to-Resourceパーミッションにも同様に適用されます。JEUS DDでカスタム・パーミッションを使用する方法については、「[第3章 アプリケーションとモジュールのセキュリティー設定](#)」を参照してください。

2.6.3. データベースを利用した設定

データベースを使ってポリシーを設定するには、以下のように`domain.xml`でセキュリティー・ドメイン・サービスを指定する必要があります。認証(Authentication)と同様、ドライバー・マネージャーを直接利用したい場合は、`<datasource-id>`の代わりに`<dbdriver-config>`タグを追加します。ただし、一般的には、なるべくJEUS JDBCデータソースを利用することを推奨します。

[例 2.13] データベースを利用したポリシーの設定 : `<<domain.xml>>`

```
<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    ...
    <security-manager>
        <security-domains>
            <security-domain>
                <name>MY_DOMAIN</name>
                <authorization>
                    <repository-service>
                        <database-repository>
                            <datasource-id>auth</datasource-id>
                        </database-repository>
                    </repository-service>
                </authorization>
                . . .
            </security-domain>
        </security-domains>
    </security-manager>
    . . .
</domain>
```

データベースを使う認証リポジトリ・サービス(AuthenticationRepositoryService)を利用するには、データベースにアクセスするJDBCドライバーと、アクセス情報のURL、ユーザー名、パスワードが必要です。

この例題では、オラクルにアクセスするための情報として、パスワードにbase64でエンコードされた文字が入力されます。特定の暗号化アルゴリズム、もしくはエンコーディング方式が適用されたパスワード値を記入する場合、`accounts.xml`のユーザー・パスワードと同様の方式で記入します。

データベースで使われるテーブルは以下のように構成されます。

[図 2.28] ポリシーを保存するためのデータベース・テーブルの構造

jeus_role_principal		
PK	domain	VARCHAR(30)
PK	rolepermissionid	VARCHAR(30)
PK	username	VARCHAR(30)

jeus_seqno		
PK	tablename	VARCHAR(30)
PK	currentseqno	VARCHAR(30)

jeus_role_permissions		
PK	domain	VARCHAR(30)
PK	rolepermissionid	VARCHAR(30)
	description	VARCHAR(100)

jeus_resource_permissions		
PK	domain	VARCHAR(30)
PK	contextid	VARCHAR(30)
PK	resourcepermissionid	VARCHAR(30)
	description	VARCHAR(100)

jeus_role_permission		
PK	rolepermissionid	VARCHAR(30)
PK	role	VARCHAR(30)
	actions	VARCHAR(100)
	classname	VARCHAR(100)
	excluded	VARCHAR(5)
	unchecked	VARCHAR(5)

jeus_resource_permission		
PK	resourcepermissionid	VARCHAR(30)
PK	res	VARCHAR(30)
	actions	VARCHAR(100)
	classname	VARCHAR(100)
	excluded	VARCHAR(5)
	unchecked	VARCHAR(5)

jeus_resource_role		
PK	domain	VARCHAR(30)
PK	rolepermissionid	VARCHAR(30)
PK	resourcepermissionid	VARCHAR(30)

データベースにテーブルが存在しない場合は、JEUS_HOME/templates/security/dbrealm.sql.templateファイルを使ってデータベースにテーブルを作成します。新しく生成されたテーブルの基本ポリシーは、「SYSTEM_DOMAIN」が「administrator」ロールを持ち、「jeus.*」のリソース権限を持ちます。「administrator」のロールに含まれたプリンシパルは、「[2.5.3. データベースを使用した設定](#)」で説明したとおり、jeusです。

2.7. 追加項目の設定

本節では、サブジェクトとポリシー以外の追加項目を設定する方法について説明します。

2.7.1. JavaSE SecurityManagerの設定

JEUSでJava SE SecurityMangerを使うと、プラットフォームの堅牢性は確保できても、性能は低下してしまいます。一般にJava SE SecurityMangerでは、JEUSの核心コードのみならず、JEUSにデプロイされているJ2EEアプリケーションおよびモジュールも完璧に信頼されるコードとみなされます。そのため、あえてセキュリティ・マネージャーを使って負荷をもたらす必要はありません。セキュリティ・マネージャーを使わないモードがJEUSのデフォルト・モードです。

ところが、Java SE SecurityMangerを使って堅牢性を高めるほうが、性能より重要な場合があります。

たとえば、システム管理者が不安定なコードを含んでいる恐れのあるJava EEアプリケーションをJEUSにデプロイする必要があるとします。この場合のように、性能の低下を甘受してもコード・レベルのセキュリティを強化してホストを保護する必要があるれば、Java SE SecurityMangerを動作させます。

Java SE SecurityMangerをJEUSと一緒に動作させるため、以下のようにdomain.xmlに特定のサーバーに対してjvm-optionを定義します。

```
-Djava.security.manager  
-Djava.security.policy=${JEUS_HOME}/domains/domain1/config/security/policy(UNIX基  
準)
```

ポリシー・ファイルはJEUS_HOME/domains/<domain name>/config/security/policyであり、内容は以下のとおりです。

[例 2.14] Java SE SecurityManagerの設定 : <<policy>>

```
grant codeBase "file:${jeus.home}/lib/system/*" {  
    permission java.security.AllPermission;  
};  
  
grant {  
    permission java.net.SocketPermission "127.0.0.1:1024-",  
        "connect, accept, connect, listen, resolve";  
    permission java.security.SecurityPermission "runTrustedLogin", "read";  
    permission java.security.SecurityPermission "loginCodeSubject", "read";  
    permission java.security.SecurityPermission "putProviderProperty.*";  
    permission java.security.SecurityPermission "insertProvider.*";  
    permission javax.management.MBeanPermission "*", "*";  
};
```

Java SE SecurityMangerはJEUSのセキュリティー・システムとは完全に別物です。JEUSセキュリティー・システムはコード・レベルの保護(コードを呼び出せるパーミッションの設定)ではなく、ユーザー・レベルの保護(ログインしたユーザーの特定と、そのユーザーのリソース・パーミッション保持の確認)が主です。

二者間の唯一の接点は、JEUSがJava SE SecurityMangerを使ってコード・レベルのパーミッションをチェックすることで悪意のあるサブレットやEJBコードからJEUSを保護する特殊な場合だけです。

2.7.2. JACCプロバイダーの設定

JEUSにおけるJACC 1.5の設定については、[「第6章 JACCプロバイダーの使用」](#)で詳しく説明します。その詳細については、当該節を参照してください。

2.7.3. アイデンティティー付与のための情報の設定

アイデンティティー・アサーション・サービス(IdentityAssertionService)をサポートする際は、証明書とユーザー間のマッピング情報が格納されているcert-user-map.xmlファイルから読み込みます。このファイルは以下のパスに存在します。


```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/
```

<domain name>はドメインの名前で、<security domain name>はユーザーが属するセキュリティ・ドメインの名前です。cert-user-map.xmlのXMLスキーマはcert-user-map.xsdであり、JEUS_HOME/lib/schemas/jeusのパスに存在します。

このファイルには最上位タグの<cert-user>の下に<user>および<cert>タグがあり、その下位に0個以上の<cert-user>タグが含まれています。それぞれはユーザーの証明書のマッピングに必要な属性情報を示します。

[例 2.15] アイデンティティ付与のための情報の設定 : <cert-user-map.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<cert-user-map xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <cert-user>
    <username>user1</username>
    <cert>
      <subjectDN>user1DN</subjectDN>
    </cert>
  </cert-user>
</cert-user-map>
```

それぞれの<cert-user>タグは、以下のような下位タグを持っています。

- <username> (必須事項)

証明書属性値にマッピングされるユーザーの名前を示します。(例: ユーザー名、ユーザーID)

- <cert>

各<cert>タグは以下のような下位タグを持っており、ユーザーに対する証明書のマッピング情報を定義します。これはトラストストア・ファイルに含まれた証明書に対し、下位の属性値がユニークなIDを保証することで、アイデンティティを付与できるように、ユーザー名とマッピングされる値を定義します。

タグ	説明
<alias>(選択事項)	キーストア内の証明書のエイリアスを定義します(文字列)
<subjectDN>(選択事項)	キーストア内の証明書のサブジェクトDNを定義します(文字列)
<SKI>(選択事項)	キーストア内の証明書のSKIを定義します(文字列)
<issuer>(選択事項)	キーストア内の証明書の発行人(issuer)を定義します(文字列)
<serialNo>(選択事項)	キーストア内の証明書のシリアル番号を定義します(文字列)

2.7.4. アイデンティティに対する証明書情報の設定

JEUSセキュリティ・システムでは、証明書で認証されたアイデンティティ・ユーザーに対し、そのユーザーの証明書情報を取得できるAPIを提供するUserCertMappingServiceを提供します。UserCertMappingServiceをサポートするときは、ユーザー別の証明書ファイルを取得するためのマッピング情報を含むuser-cert-map.xmlファイルから読み込みます。このファイルは以下のパスに存在します。

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/
```

<domain name>はドメインの名前で、<security domain name>はユーザーが属するセキュリティ・ドメインの名前です。user-cert-map.xmlのXMLスキーマはuser-cert-map.xsdで、JEUS_HOME/lib/schmas/jeusのパスに存在します。

このファイルには、最上位タグの<user-cert-map>タグの下に0個以上の<user-cert>タグが含まれています。それぞれはキーストア・ファイルからユーザーの証明書の取得に必要な属性情報を示します。

[例 2.16] アイデンティティに対する証明書情報の設定 : <<user-cert-map.xml>>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user-cert-map xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <user-cert>
    <username>user1</username>
    <alias>alias1</alias>
    <keypassword>changeit</keypassword>
    <secretkey>
      <keyname>testkeypass</keyname>
      <keyalgorithm>AES</keyalgorithm>
      <keyvalue>bjhTUjJvSXRTOGlkVEdlNHJnM2N3VnljSDZXV0JkYz0=</keyvalue>
    </secretkey>
  </user-cert>
</user-cert-map>
```

各<secretkey>タグは、以下のような下位タグを持っています。

- <username>(必須事項)

キーストア内の証明書に対するユーザー名を定義します。「primary」アイデンティティであり、唯一でなければなりません(例: ユーザー名、ユーザーID)

- <alias>(必須事項)

キーストア内の証明書に対するエイリアスを定義します。

- <keypassword>(選択事項)

キーストア内の証明書に対する秘密鍵を取得するためのキー・パスワード(Keypassword)を定義します(例: changeit)

- **<secretkey>(選択事項)**

秘密鍵を定義します。

各<secretkey>タグは、以下のような下位タグを持っています。

タグ	説明
<keyname>(必須事項)	秘密鍵の名前を定義します。<user-cert-map>タグの下位に存在するkeyname値は唯一でなければなりません(文字列)
<keyalgorithm>(必須事項)	秘密鍵のキー・アルゴリズムを示します(文字列)
<keyvalue>(必須事項)	秘密鍵の値をBase64形式で表します

第3章 アプリケーションとモジュールのセキュリティ設定

本章では、J2EEアプリケーションとEJBモジュール、Webモジュールでセキュリティを設定する方法について説明します。

3.1. 概要

本節では、アプリケーションとモジュールのセキュリティ設定に関連する基本的な内容について説明します。

J2EEアプリケーション、EJBモジュール、Webモジュールの基本的なセキュリティ設定方法は以下のとおりです。

1. モジュールごとに、Role-to-Resourceマッピングを設定します。
2. アプリケーションごとに、Principal-to-Roleマッピングを設定します。

3.1.1. モジュール・デプロイ対アプリケーション・デプロイ

JEUSにおけるデプロイメント方法は大きく下記の2つがあります。

- EJBモジュール(.jarファイル)、Webモジュール(.warファイル)をそれぞれ1つずつデプロイする方法
- 複数のEJBモジュール、Webモジュール、コネクタ・モジュール(.rar)をEARで圧縮してJ2EEアプリケーション(.ear)としてデプロイする方法

3.1.2. Role-to-Resourceマッピング

アセンブラは、EJBモジュールまたはWebモジュールをデプロイする前に、モジュールにRole-to-Resourceマッピングを設定する必要があります。Role-to-Resourceマッピングはセキュリティ制約(security constraint)ともいい、論理的なロールにモジュールのリソースをマッピングします。

- EJBモジュールにおけるリソースはEJBのメソッドを意味し、META-INF/ejb-jar.xmlに設定します。
- WebモジュールにおけるリソースはサーブレットURLを意味し、WEB-INF/web.xmlに設定します。

以下は、EJBモジュールに設定されているセキュリティ制約の内容で、META-INF/ejb-jar.xmlの設定について説明します。

[例 3.1] EJBモジュールに設定されているセキュリティ制約 : <<ejb-jar.xml>>

```
<?xml version="1.0"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee">
  <display-name>product</display-name>
  <enterprise-beans>
    <entity>
      ❶      <ejb-name>product</ejb-name>
      . . .
      ❷      <security-role-ref>
                <role-name>cust</role-name>
                <role-link>customer</role-link>
            </security-role-ref>
    </entity>
  </enterprise-beans>
  <assembly-descriptor>
    . . .
    ❸      <security-role>
                <role-name>administrator</role-name>
            </security-role>
    ❹      <security-role>
                <role-name>customer</role-name>
            </security-role>
    ❺      <method-permission>
                <role-name>administrator</role-name>
                <method>
                    <ejb-name>product</ejb-name>
                    <method-intf>Remote</method-intf>
                    <method-name>getSecretKey</method-name>
                    <method-params>
                        <method-param>java.lang.Integer</method-param>
                    </method-params>
                </method>
            </method-permission>
    ❻      <method-permission>
                <unchecked/>
                <method>
                    <ejb-name>product</ejb-name>
                    <method-name>doSomeAdmin</method-name>
                    <method-params>
                        <method-param>java.lang.String</method-param>
                    </method-params>
                </method>
            </method-permission>
```

```

⑦      <method-permission>
          <role-name>customer</role-name>
          <method>
              <ejb-name>product</ejb-name>
              <method-name>test1</method-name>
          </method>
      </method-permission>
⑧      <exclude-list>
          <method>
              <ejb-name>product</ejb-name>
              <method-intf>Remote</method-intf>
              <method-name>getCustomerProfile</method-name>
              <method-params></method-params>
          </method>
      </exclude-list>
  </assembly-descriptor>
</ejb-jar>

```

上記のDDファイルに設定されたセキュリティ制約の内容は以下のとおりです。

- ❶ 「product」というエンティティーEJBがあります。
- ❷ Role-to-Role Referenceマッピングが宣言されています。実際に宣言された「customer」ロールは「cust」というロール・リファレンスで参照することができます。つまり、EJBソース内の「cust」という名前のロールは、実際には「customer」ロールに対応することを意味します。
- ❸ 論理的なロールの「administrator」が宣言されています。
- ❹ 論理的なロールの「customer」が宣言されています。
- ❺ Role-to-Resourceマッピングが宣言されています。「administrator」のロールがリモートEJBインターフェースのgetSecreteKeyメソッドを呼び出せるように許可しています。このマッピングによって、「administrator」ロールに属するプリンシパルだけがgetSecreteKeyメソッドを呼び出せます。
- ❻ doSomeAdminメソッドがUncheckedで宣言されており、ロールに関係なく誰でもこのメソッドを呼び出せます。
- ❼ 「customer」ロールに属するプリンシパルは、「product」EJBの「test1」メソッドを呼び出せます。
- ❽ getCustomerProfileメソッドがExcludedリストに含まれており、どのロールもこのメソッドを呼び出すことができません。

Webモジュールに対しても、EJBと同様の設定が適用されます。ただ、EJBではロールがEJBメソッドを呼び出せるかどうかの問題になるのに対し、WebモジュールではロールがサーブレットURLにアクセスできるかどうかの問題になります。Webモジュールのセキュリティ設定については、[「3.3. Webモジュールのセキュリティ設定」](#)を参照してください。

上記で触れた2つのロールの「administrator」と「customer」は論理的な概念で、実際にデプロイされる環境を意味するわけではありません。そのため、論理的なロールを実際の環境のユーザーやユーザー・グループにマッピングする作業が必要です。この作業をPrincipal-to-Roleマッピングといいます。詳細については、「3.1.3. Principal-to-Roleマッピング」を参照してください。

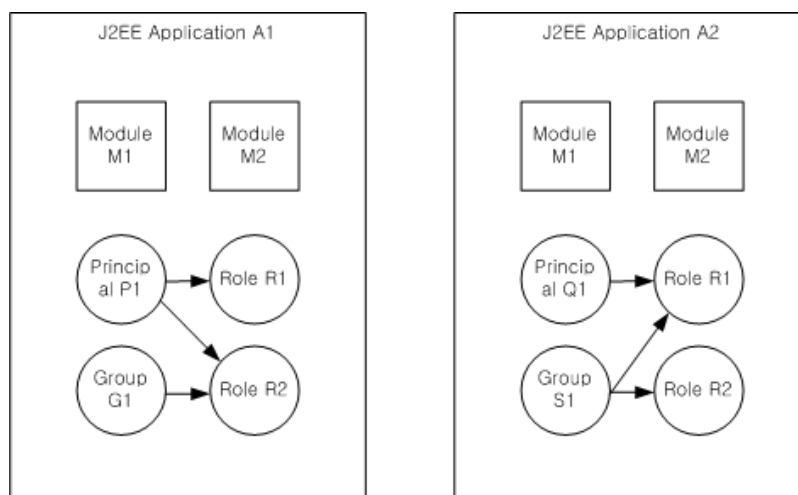
3.1.3. Principal-to-Roleマッピング

通常、デプロイヤー(Deployer)は、J2EEモジュール、またはアプリケーションに定義されているロールを実際の環境のユーザーやユーザー・グループにマッピングする作業を行います。プリンシパルを論理的なロールにマッピングする際には、スコープがアプリケーション・レベルになることに注意してください。

たとえば、モジュールM1とM2が、A1というアプリケーションに含まれているとしたら、Principal-to-RoleマッピングはM1とM2で共有されます。さらに、A2という別のアプリケーションはA1とは完全に異なるPrincipal-to-Roleマッピングを持ちます。A1とA2で同じ名前のロールがあっても、異なるロールと認識され、互いに共有されることはありません。

以下の図は、こうしたスコープに関連する概念を表したものです。

【図 3.1】 Principal-to-Roleマッピング



上の図のように、異なるアプリケーション(A1, A2)は異なるPrincipal-to-Roleマッピングを保持し、それぞれのアプリケーションに含まれているモジュール(M1, M2)はそのマッピングを共有します。

JEUSにおけるPrincipal-to-Roleマッピングは以下のファイルに設定されます。

- JEUS DDのjeus-ejb-dd.xml(EJBモジュール)
- jeus-web-dd.xml(Webモジュール)
- jeus-application-dd.xml(J2EEアプリケーション)

以下は、ejb-jar.xmlに対するjeus-ejb-dd.xmlの設定例です。

[例 3.2] Principal-to-Roleマッピング : <<jeus-ejb-dd.xml>>

```
<?xml version="1.0"?>

<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    ❶ <role-permission>
      <principal>user1</principal>
      <role>administrator</role>
      <classname>mypackage.MyRolePermission</classname>
    </role-permission>
    ❷ <role-permission>
      <principal>user2</principal>
      <role>customer</role>
    </role-permission>
    ❸ <role-permission>
      <role>roleA</role>
      </excluded>
    </role-permission>
    ❹ <role-permission>
      <role>roleB</role>
      </unchecked>
    </role-permission>
    ❺ <unspecified-method-permission>
      <role>administrator</role>
    </unspecified-method-permission>
  </module-info>
  <beanlist>
    . . .
  </beanlist>
</jeus-ejb-dd>
```

上記の例は以下のようなセキュリティ情報を含んでいます。

- ❶ 「user1」というプリンシパル(ユーザー)に「administrator」ロールを許可するロール・パーミッションが宣言されています。つまり、「user1」は「administrator」ロールのすべての権限を持つことになります。

上の例で見るように、カスタム・ロール・パーミッション(Cusotom Role Permission)は<classname>タグで定義します。カスタム・ロール・パーミッションの詳細については、「[第2章 セキュリティー・システムの設定](#)」を参照してください。

- ❷ 「user2」というプリンシパル(ユーザー)に「customer」ロールを付与するロール・パーミッションが宣言されています。つまり、「user2」は「customer」ロールが持つすべての権限を持つことになります。
<classname>タグが設定されていないので、デフォルトのロール・パーミッションが設定されます。
- ❸ 「roleA」というロールはすべてのプリンシパルから排除されています(excluded)。そのため、どのプリンシパルも「roleA」に含まれません。

- ④ 「roleB」というロールは権限付与システムでチェックされません(unchecked)。そのため、すべてのプリンシパルは自動で「roleB」に含まれます。
- ⑤ <unspecified-method-permission>は「unspecified」EJBメソッドを基本的にどう扱うかを決めます。「unspecified」EJBメソッドは、ejb-jar.xmlで<method-permission>タグで言及されていないメソッドを意味します。

<unspecified-method-permission>は3つの方法で処理できます。ロールにマッピングするか(例では「administrator」ロールにマッピングされている)、「excluded」で扱うか(誰もアクセスできない)、「unchecked」で扱います(誰でもアクセスできる)。デフォルト値は「unchecked」です。

参考

Webモジュール(jeus-web-dd.xml)の設定は、<unspecified-method-permission>の設定がないことを除けば、EJBモジュールの設定と同じです。

3.1.4. ユーザー設定

ユーザー設定の方法は大きく2つあります。

- アプリケーションで使用するドメイン自体のユーザー設定を利用する方法として、以下のファイルに設定します。(「[2.5. セキュリティー・システムのユーザー情報の設定](#)」を参照)

```
JEUS_HOME/domains/<domain name>/config/security/<security domain name>/accounts.xml
```

- EJBモジュール(.jarファイル)、Webモジュール(.warファイル)、J2EEアプリケーション(.ear)のユーザー設定方式として、WEB-INF(Webモジュール)またはMETA-INF(EJBモジュールおよびEAR)ディレクトリー内の**accounts.xml**を設定します。

参考

設定方法に関係なく、ユーザー情報のスコープはアプリケーションのセキュリティー・ドメイン・レベルになります。つまり、1つのセキュリティー・ドメイン内のユーザーは、同じセキュリティー・ドメインを使用するすべてのアプリケーションで共有されます。

3.2. EJBモジュールのセキュリティー設定

本節では、EJBモジュールでセキュリティーを設定する方法について説明します。本節で取り上げないCSIのようなセキュリティー設定については、『JEUS EJBガイド』を参照してください。

EJBモジュールのセキュリティー設定は大きく2つの方法で構成されます。

- [ejb-jar.xmlの設定](#)
- [jeus-ejb-dd.xmlの設定](#)

3.2.1. ejb-jar.xmlの設定

ejb-jar.xmlには以下のようなセキュリティ要素を設定することができます。

- Role-to-role referenceマッピング
- セキュリティー・アイデンティティー(security identity)の設定
- 論理的なロールの宣言
- EJBメソッドのパーミッション設定

以下の例は、上記のセキュリティ設定要素を含むejb-jar.xmlです。

[例 3.3] セキュリティーの設定 : <<ejb-jar.xml>>

```
<?xml version="1.0"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/j2ee">
  <display-name>product</display-name>
  <enterprise-beans>
    <entity>
      <ejb-name>product</ejb-name>
      . . .
      <security-role-ref>
        <role-name>cust</role-name>
        <role-link>customer</role-link>
      </security-role-ref>
      <security-identity>
        <run-as>
          <role-name>administrator</role-name>
        </run-as>
      </security-identity>
    </entity>
  </enterprise-beans>
  <assembly-descriptor>
    . . .
    <security-role>
      <role-name>administrator</role-name>
    </security-role>
    <security-role>
      <role-name>customer</role-name>
    </security-role>
  </assembly-descriptor>
</ejb-jar>
```

```

</security-role>
<method-permission>
  <role-name>administrator</role-name>
  <method>
    <ejb-name>product</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>getSecretKey</method-name>
    <method-params>
      <method-param>java.lang.Integer</method-param>
    </method-params>
  </method>
</method-permission>
<method-permission>
  <unchecked/>
  <method>
    <ejb-name>product</ejb-name>
    <method-name>doSomeAdmin</method-name>
    <method-params>
      <method-param>java.lang.String</method-param>
    </method-params>
  </method>
</method-permission>
<method-permission>
  <role-name>customer</role-name>
  <method>
    <ejb-name>product</ejb-name>
    <method-name>test1</method-name>
  </method>
</method-permission>
<exclude-list>
  <method>
    <ejb-name>product</ejb-name>
    <method-intf>Remote</method-intf>
    <method-name>getCustomerProfile</method-name>
    <method-params></method-params>
  </method>
</exclude-list>
</assembly-descriptor>
</ejb-jar>

```

以下は、ejb-jar.xmlに設定できるセキュリティー要素についての説明です。

● Role-to-role referenceマッピング

Role-to-role referenceマッピングは、**<security-role-ref>**タグで定義します。論理的なロールに対する参照(リファレンス)を設定します。EJBソースでは実際のロールの代わりにロール・リファレンスを使うことになります。

以下は、<security-role-ref>タグ内で使用するタグです。

タグ	説明
<role-name>	実際のロールに対するリファレンス名です。EJBコードでは実際のロールの名前の代わりにリファレンス名を使います
<role-link>	security-roleタグで宣言した実際のロール名です。ロール・リファレンスはロール・リンク(role-link)が指すロールにマッピングされます

● セキュリティー・アイデンティティーの設定

モジュール内にある各EJBは、他のEJBをリモートで呼び出す際に、自分のセキュリティ・アイデンティティーを伝播します。EJBのセキュリティ・アイデンティティーは、<ejb-jar> <enterprise-beans> <type> <security-identity>タグで設定します。

以下のような下位タグを持っています。

タグ	説明
<use-caller-identity>	エンプティ・タグです。このタグを設定すると、EJBを呼び出す際に、呼び出し側自身がセキュリティ・アイデンティティーとして使われます
<run-as>	EJBが呼び出す際に、<role-name>に設定されているロールに該当するプリンシパルをセキュリティ・アイデンティティーとして使用します。jeus-ejb-dd.xmlに<run-as-identity>タグがあれば、この設定は無視されます

● 論理的なロールの宣言

<assembly-descriptor>の下位タグである<security-role>タグは、EJBモジュールに適用される論理的なロールを宣言するタグです。ejb-jar.xmlに記述されているすべてのロール名は、このタグ内に宣言されている必要があります。

● EJBメソッドのパーミッション設定

<assembly-descriptor>の下位タグである<method-permission>タグは、EJBメソッドにセキュリティ制約を設定するタグです。

以下の下位タグを設定します。

– <role-name> (選択事項)

<security-role>タグに宣言されている論理的なロール名で、下に定義されたメソッドにアクセスできます。

– <unchecked> (選択事項)

値を持たないエンプティ・タグです。このタグを設定すると、下に定義されたメソッドは権限チェックされません。つまり、ロールに関係なく誰でもメソッドにアクセスできます。

– **<method>** (1個以上)

<role-name>タグまたは**<unchecked>**タグが適用されるメソッドで、下位タグは以下のとおりです。

タグ	説明
<ejb-name>	メソッドを含んでいるEJBの名前です
<method-ntf>	メソッドを含んでいるEJBのインターフェース・タイプです。省略すると、ローカル・インターフェースとリモート・インターフェースの両方を示します
<method-name>	EJBのメソッド名です
<method-params> (選択事項)	0個以上の <method-param> タグを持つことができます。 <method-param> タグは、EJBメソッドの各パラメータを示します。タグが省略された場合、メソッド名が同一であれば、パラメータに関係なく同じパーミッションが適用されます。EJBでは、パラメータは異なり、メソッド名は同一の複数のメソッドが存在できます。それらのメソッドにこのメソッド・パーミッションが共通して適用されます。 <method-param> タグ値はパッケージ名を含む完全なJavaクラス名になるか、またはintのようなJavaプリミティブ(Primitive)タイプになれます。 <method-param> をエンプティ・タグに設定すると(<method-params/>)、媒介変数が存在しないことを意味します

– **<exclude-list>** (1個)

ただ1つの**<exclude-list>**を指定できます。排除される(Excluded)EJBメソッドを定義します。つまり、誰もアクセスできないメソッドです。下位タグとして複数の**<method>**タグを持つことができます。

参考

例についての説明は、「[3.1.2. Role-to-Resourceマッピング](#)」を参照してください。各タグの詳細については、EJB仕様を参照してください。またJEUSでEJBをデプロイする方法については、『JEUS EJBガイド』を参照してください。

3.2.2. jeus-ejb-dd.xmlの設定

jeus-ejb-dd.xmlには以下のようなセキュリティ要素を設定することができます。

- セキュリティー関連設定
- ejb-jar.xmlに設定されていないEJBメソッドの設定

- Run-as Identityを使用するEJBプリンシパルの設定

EJBでRole-to-Methodマッピングはほとんどejb-jar.xmlに設定されていますが、ejb-jar.xmlに設定されていないメソッドや、実際のPrincipal-to-RoleマッピングはEJB.jarファイルのMETA-INFディレクトリーに存在するjeus-ejb-dd.xmlファイルで設定します。

以下の例は、上記のセキュリティー設定の要素を含むjeus-ejb-dd.xmlです。

[例 3.4] セキュリティーの設定 : <<jeus-ejb-dd.xml>>

```
<?xml version="1.0"?>
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    <role-permission>
      <principal>user1</principal>
      <role>administrator</role>
      <classname>mypackage.MyRolePermission</classname>
    </role-permission>
    <role-permission>
      <principal>user2</principal>
      <role>customer</role>
    </role-permission>
    <role-permission>
      <role>roleA</role>
      </excluded>
    </role-permission>
    <role-permission>
      <role>roleB</role>
      </unchecked>
    </role-permission>
    <unspecified-method-permission>
      <role>administrator</role>
    </unspecified-method-permission>
  </module-info>
  <beanlist>
    <jeus-bean>
      . . .
      <run-as-identity>
        <principal-name>user1</principal-name>
      </run-as-identity>
    </jeus-bean>
    . . .
  </beanlist>
</jeus-ejb-dd>
```

jeus-ejb-dd.xmlには以下のようなセキュリティー要素を設定できます。

- セキュリティー関連設定

jeus-ejb-dd.xmlでセキュリティー関連設定は、ほとんど<jeus-ejb-dd>タグの下にある<module-info>タグ内に設定されます。

– <role-permission> (0個以上)

以下は、<role-permission>の下位タグについての説明です。各タグはロール(principal-to-roleマッピング)を定義しています。

タグ	説明
<principal>(0個以上)	ロールにマッピングされるプリンシパルの名前です
<role> (必須事項)	1つのみ許可されます。必須項目であり、実際の論理的なロール名です。 ロール・パーミッションを実装するJavaクラスのコンストラクターに1番目のパラメータとして渡されます。ロール名はejb-jar.xmlに宣言されたロール名と一致する必要があります
<actions>(選択事項)	ロール・パーミッションの付加情報を提供します。このタグを設定すると、アクション・データはロール・パーミッションを実装するJavaクラスの2番目のパラメータとして渡されます
<classname> (選択事項)	ロール・パーミッションを実装するクラスで、パッケージ名を含む完全な名前を設定する必要があります。このクラスはjava.security.Permissionクラスのサブクラスで、少なくとも1つのパラメータ(ロール名)を受信する、publicコンストラクターを持ちます。 省略すると、jeus.security.resource.RolePermissionが使われます
<excluded> (選択事項)	エンプティ・タグです。このタグを設定すると、当該ロールは排除された(excluded)ものとして扱われます(誰もロール・パーミッションで言及したロールにアクセスできません)。 <principal>、<unchecked>タグより優先順位が高いです
<unchecked> (選択事項)	エンプティ・タグです。このタグを設定すると、当該ロールは権限をチェックしない(unchecked)ものとして扱われます(誰もがロール・パーミッションで言及したロールにアクセスできます)。 <principal>タグより優先順位が高いです

● ejb-jar.xmlに設定されていないEJBメソッドの設定

ejb-jar.xmlに設定されていないEJBメソッドを設定します。<module-info>の<unspecified-method-permission>タグ内に定義されます。

設定可能な下位タグは以下のとおりです。

タグ	説明
<role>(選択事項)	このタグを設定すると、設定されていないすべてのメソッドはこのロールにマッピングされます
<excluded>(選択事項)	このタグを設定すると、設定されていないすべてのメソッドは排除された(excluded)ものとして扱われます(誰もアクセスできません)。 <role>や<unchecked>タグより優先順位が高いです。
<unchecked>(選択事項)	このタグを設定すると、設定されていないすべてのメソッドは権限をチェックしない(unchecked)ものとして扱われます(誰もがアクセスできます)。 <role>タグより優先順位が高いです

● Run-as Identityを使用するEJBプリンシパルの設定

Run-as Identityを使用するEJBのプリンシパルの設定には、<jeus-bean> <run-as-identity> <principal-name>タグ内に与えられたプリンシパル名が使われます。プリンシパルはejb-jar.xmlの<security-identity> <run-as> <role-name>タグ内に設定されているロールに含まれている必要があります。

参考

1. 例の詳細については、「[3.1.2. Role-to-Resourceマッピング](#)」を参照してください。
2. 各タグの詳細情報については、『JEUS EJBガイド』を参照してください。

3.3. Webモジュールのセキュリティ設定

本節では、Webモジュール(サーブレット・モジュール)でセキュリティを設定する方法について説明します。

Webモジュールのセキュリティ設定は、大きく2つの方法で構成されます。

- [web.xmlの設定](#) : Role-to-Web Resourceマッピングの設定
- [jeus-web-dd.xmlの設定](#) : Principal-to-Roleマッピングの設定

3.3.1. web.xmlの設定

web.xmlはWebアーカイブ(WARファイル)のメインDDファイルです。標準のJ2EE DDファイルで、WARファイルのWEB-INFディレクトリ内に存在します。

web.xmlには以下のようなセキュリティ要素を設定することができます。

- Run-as Identity

Webモジュール内にあるサーブレットは、EJBをリモートで呼び出す際に、自分のセキュリティ・アイデンティティを伝播します。

サーブレットのセキュリティ・アイデンティティは<web-app><servlet><run-as>タグ内に設定され、以下のような下位タグを持ちます。

タグ	説明
<role-name>(選択事項)	サーブレットを実行するロールで、省略すると、呼び出し側のセキュリティ・アイデンティティが使われます

- Role-to-role referenceマッピング

Role-to-role referenceマッピングは<security-role-ref>タグで設定し、ロール・リファレンスを実際に論理的なロールにマッピングします。ロール・リファレンスは、サーブレット・コードでロールを表すときに使われます。

web.xmlで<security-role-ref>タグは<servlet>タグ内に設定され、以下のような下位タグを持ちます。

タグ	説明
<role-name>	サーブレット・コードに使われるロール名です
<role-link>	<security-role>タグで定義した実際のロール名で、<role-name>に設定されているロール・リファレンスを実際のロールにマッピングします

- サーブレットURLへのアクセス許可 (Security constraints, セキュリティ制約)

サーブレットURLへのアクセスを制限する目的で、<web-app>タグの下にある<security-constraint>タグを使います。<security-constraint>は以下のような下位タグを持ちます。

- <web-resource-collection> (1個以上)

アクセス制限が設定されているWebリソースの一覧です。

タグ	説明
<web-resource-name>	Webリソースの名前です
<url-patterns>(0個以上)	Webリソースを示すURL(コンテキスト・ルートの相対パスで指定)のパターンです。 以下は、URLパターンの例です <ul style="list-style-type: none">– /mywebapp/* : mywebappが含まれているすべてのURL– /something : 正確なURL– *.jsp : JSPで終わるすべてのリソース

タグ	説明
<http-method>(0個以上)	Webリソースに適用されるHTTPメソッドです <ul style="list-style-type: none"> – GET – POST – PUT

– <auth-constraint> (選択事項)

<security-constraint>タグ内に定義されたWebリソースにアクセスできるロールを設定します。下位タグには0個以上の<role-name>タグを設定できます。それぞれのタグはアクセスが許可されたロールを示します。

ロール名を設定せずに、エンプティ・タグ(<auth-constraint/>)で設定する場合は、誰も該当するWebリソースにアクセスできません。(これはWebリソースが排除される(excluded)ことと同じです。)

<auth-constraint>タグが完全に省略されている場合は、Webリソースがチェックされない(unchecked)ことを意味し、ロールに関係なく誰もがWebリソースにアクセスできます。

– <user-data-constraint> (選択事項)

Webリソースに確立されるコネクションに対して転送保証(transport guarantee)をするかどうかを宣言するタグです。

下位タグとして<transport-guarantee>タグを持ち、Webリソースへのコネクションを保証するレベルを示します。

設定値	説明
NONE	コネクションを保証しません
INTEGRAL	メッセージの整合性(つまり、転送されたメッセージは変更されていないオリジナルであること)を保証するコネクションであることを意味します
CONFIDENTIAL	メッセージがハッキングされるのを防止するために暗号化されて転送されることを意味します

● 論理的なロールの宣言

<security-constraint>タグの下には、論理的なロールを示す<security-role>タグを設定できます。各タグはすでにDDに記述された論理的なロール名を宣言する<role-name>タグを持ちます。

以下は、web.xmlのセキュリティ設定の例です。

[例 3.5] Webモジュールのセキュリティの設定 : <<web.xml>>

```
<?xml version="1.0"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>HelloWorld</servlet-name>
    . . .

❶    <run-as>
      <role-name>R1</role-name>
    </run-as>

❷    <security-role-ref>
      <role-name>adminRef</role-name>
      <role-link>R1</role-link>
    </security-role-ref>
    . . .

  </servlet>

❸    <security-constraint>
      <web-resource-collection>
        <web-resource-name>A</web-resource-name>
        <url-pattern>/a/*</url-pattern>
        <url-pattern>/b/*</url-pattern>
        <url-pattern>/a</url-pattern>
        <url-pattern>/b</url-pattern>
        <http-method>DELETE</http-method>
        <http-method>PUT</http-method>
      </web-resource-collection>
      <web-resource-collection>
        <web-resource-name>B</web-resource-name>
        <url-pattern>*.asp</url-pattern>
      </web-resource-collection>
      <auth-constraint/>
    </security-constraint>

❹    <security-constraint>
      <web-resource-collection>
        <web-resource-name>C</web-resource-name>
        <url-pattern>/a/*</url-pattern>
        <url-pattern>/b/*</url-pattern>
        <http-method>GET</http-method>
      </web-resource-collection>
      <web-resource-collection>
        <web-resource-name>D</web-resource-name>
        <url-pattern>/b/*</url-pattern>
        <http-method>POST</http-method>
      </web-resource-collection>
```

```

    <auth-constraint>
      <role-name>R1</role-name>
    </auth-constraint>
    <user-data-constraint>
      <transport-guarantee>
        CONFIDENTIAL
      </transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  ⑤ <security-role>
    <role-name>R1</role-name>
  </security-role>
  <security-role>
    <role-name>R2</role-name>
  </security-role>
  <security-role>
    <role-name>R3</role-name>
  </security-role>
</web-app>

```

上記の例は以下のようなセキュリティ情報を含んでいます。

- ① Run-as-Identityがロール「R1」に設定されているということは、サーブレットのセキュリティ・アイデンティティーが「R1」に属するプリンシパルであるという意味です。実際のプリンシパルはJEUS DDファイルで設定されます。
- ② <security-role-ref>タグは実際のロールである「R1」をロール・リファレンスの「adminRef」にマッピングしています。実際のサーブレット・ソース内では、ロールの代わりにロール・リファレンスの「adminRef」を使います。
- ③ 1番目のセキュリティ制約の設定は、「/a」、「/b」、「/a/*」、「/b/*」のURLパターンを持ちながら、HTTPメソッドがDELETE、PUTのURLと「*.asp」で終わるURLには誰もアクセスできないことを意味します。
- ④ 2番目のセキュリティ制約の設定は、「/a/*」、「/b/*」のURLパターンを持つGET方式のURLと、「/b/*」パターンを持つPOST方式のURLには、「R1」ロールに含まれたプリンシパルのみアクセスできることを意味します。そして、該当する接続はCONFIDENTIALを保証します。
- ⑤ 「R1」、「R2」、「R3」の3つの論理的なロールを宣言しています。

Webリソース(URLパターン+HTTPメソッド)がweb.xmlに別途設定されていない場合、そのリソースはロールに関係なく誰でもアクセスできます。これは、J2EEでWebアプリケーションへのアクセスのデフォルト値です。

参考

web.xmlに使用されるすべてのタグの詳細については、サーブレットの仕様を参照してください。また、JEUSでサーブレットをデプロイすることについての追加情報は、『JEUS Webエンジンガイド』を参照してください。

3.3.2. jeus-web-dd.xmlの設定

WebモジュールのJEUS DDファイルであるjeus-web-dd.xmlでセキュリティーを設定する方法を紹介します。

● Principal-to-Roleマッピングの定義

jeus-web-dd.xmlでPrincipal-to-Roleマッピングを定義するには、**<jeus-web-dd><role-mapping>**タグを使用します。

以下は、**<role-mapping>**タグの下位タグです。

– **<role-permission>** (0個以上)

ロールに対するパーミッション(一般にPrincipal-to-Roleマッピング)を以下の下位タグに設定します。

タグ	説明
<principal> (0個以上)	ロールにマッピングされるプリンシパルの名前です
<role>(1個、必須事項)	論理的なロール名です。 ロール名はロール・パーミッションを実装するJavaクラスのコンストラクターに1番目のパラメータとして渡されます。ロール名はweb.xmlに与えられた論理的なロール名と同一に設定します
<actions>(選択事項)	ロール・パーミッションに付加情報を提供します。このタグを設定すると、ロール・パーミッションを実装するJavaクラスのコンストラクターの2番目のパラメータとして渡されます
<classname>(選択事項)	ロール・パーミッションを実装するJavaクラス名です。 このクラスはjava.security.Permissionの実装クラスであり、少なくとも1つのパラメータ(String rolename)を受信する、修飾子がpublicのコンストラクターを持ちます。 省略すると、デフォルトでjeus.security.resource.RolePermissionクラスが使われます
<excluded>(選択事項)	エンプティ・タグです。このタグを設定すると、ロールは排除された(excluded)ものとして扱われます。つまり、どのプリンシパルにも当該ロールを付与できません。 <principal>、<unchecked>タグより優先順位が高いです
<unchecked>(選択事項)	エンプティ・タグです。このタグを設定すると、ロールは権限をチェックしない(unchecked)ものとして扱われます。つまり、すべてのプリンシパルにロールを付与できます。 <principal>タグより優先順位が高いです

● サブレットで<run-as-principal>を設定

サブレットで<run-as-principal>を設定するには、<servlet>タグを宣言し、<run-as><principal-name>タグを追加します。<principal-name>タグ値は、web.xmlの<run-as><role-name>に宣言されたロールに属するプリンシパルである必要があります。

注

現在は、web.xmlに記述されていないサブレットURLをどう扱うかを定義するタグがjeus-web-dd.xmlに存在しません。サブレットの仕様では定義されていないすべてのURLを常に権限をチェックしない(Unchecked)ものとして扱います。つまり、誰もがアクセスできます。

以下は、jeus-web-dd.xmlのセキュリティ設定の例です。

[例 3.6] Webモジュールのセキュリティの設定 : <<jeus-web-dd.xml>>

```
<?xml version="1.0"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <context-path>/tutorial</context-path>
    <docbase>Tutorial</docbase>
    . . .
    <role-mapping>
        ❶ <role-permission>
            <principal>user1</principal>
            <principal>user2</principal>
            <principal>customerGroup</principal>
            <role>R1</role>
        </role-permission>
        ❷ <role-permission>
            <role>R2</role>
            </excluded>
        </role-permission>
        ❸ <role-permission>
            <role>R3</role>
            </unchecked>
        </role-permission>
        ❹ <role-permission>
            <principal>tellerGroup</principal>
            <role>R4</role>
            <actions>09:00-17:00</actions>
            <classname>
                jeus.security.resource.TimeConstrainedRolePermission
            </classname>
        </role-permission>
    </role-mapping>
```

```

⑤      <servlet>
          <servlet-name>HelloWorld</servlet-name>
          <run-as-identity>
              <principal-name>user1</principal-name>
          </run-as-identity>
          . . .
      </servlet>
      . . .
  </jeus-web-dd>

```

上記の例は以下のようなセキュリティ情報を含んでいます。

- ❶ 「user1」、「user2」、「customerGroup」プリンシパルは「R1」ロールに属します。
- ❷ 「R2」ロールは排除(Excluded)されています。つまり、誰も「R2」ロールを付与されることができません。
- ❸ 「R3」ロールはチェックしません(Unchecked)。つまり、誰でも「R3」ロールを付与されることができます。
- ❹ 「tellerGroup」プリンシパルは午前9時から午後5時まで、ロール「R4」を付与されます。
jeus.security.resource.TimeConstrainedRolePermissionクラスのimplies()メソッドで実装されます。
- ❺ run-as-principalとして「user1」が使われています。つまり、サーブレットでEJBを呼び出す際、「user1」がセキュリティ・アイデンティティとして渡されます。デプロイヤーは「user1」がweb.xmlに定義された「R1」ロールに属するかどうかを必ず確認する必要があります。

3.4. J2EEアプリケーションのセキュリティ設定

本節では、J2EEアプリケーション(EARファイル)のセキュリティを設定する方法について説明します。

基本的な設定方法は以下のとおりです。

- [application.xmlの設定](#) : 論理的なロールを宣言
- [jeus-application-dd.xmlの設定](#) : デプロイするセキュリティ・ドメインとPrincipal-to-Roleマッピングを設定

3.4.1. application.xmlの設定

J2EEアプリケーションは、.ear拡張子を持つアーカイブ・ファイルです。EARアーカイブは、EJBモジュール、Webモジュール、コネクタ・モジュール、そしてモジュールで参照しているクラスで構成されています。META-INFディレクトリはapplication.xml設定ファイルを含み、アプリケーション全体に関する様々な情報を提供します。

本節では、application.xmlのセキュリティ関連部分を取り上げて説明します。

基本的にapplication.xmlでセキュリティに関連する部分は、論理的なロールを設定する部分しかありません。ロールの宣言は、EARファイル内にあるすべてのEJBモジュールとWebモジュールに共通して適用され、アプリケーションのスコープを持ちます。

[例 3.7] J2EEアプリケーションのセキュリティ設定 : <<application.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<application version="1.4" . . .>
  <description>Application description</description>
  <display-name>myApp</display-name>
  <module>
    <web>
      <web-uri>myWebApp.war</web-uri>
      <context-root>myWebApp</context-root>
    </web>
  </module>
  <module>
    <ejb>myEjbApp.jar</ejb>
  </module>
  ❶ <security-role>
    <role-name>Administrator</role-name>
  </security-role>
  ❷ <security-role>
    <role-name>Customer</role-name>
  </security-role>
</application>
```

上記の例では「Administrator」と「Customer」の2つのロールが宣言されています。

2つのロールは、Role-to-Resourceのマッピングを定義するために、EJBモジュールとWebモジュールのDDファイルで使われます。EJBモジュールとWebモジュールのセキュリティについては、[「3.2. EJBモジュールのセキュリティ設定」](#)と[「3.3. Webモジュールのセキュリティ設定」](#)ですでに説明しています。

アプリケーションをJEUSサーバーにデプロイする前に、論理的なロールが実際にプリンシパルにマッピングされているかどうかを確認する必要があります。それぞれのケースで、以下のファイルにマッピングされます。

- アプリケーションの場合 : jeus-application-dd.xml
- EJBモジュールの場合 : jeus-ejb-dd.xml(Principal-to-Roleの設定は、[「3.2.2. jeus-ejb-dd.xmlの設定」](#)を参照)
- Webモジュールの場合 : jeus-web-dd.xml(Principal-to-Roleの設定は、[「3.3.2. jeus-web-dd.xmlの設定」](#)を参照)

アプリケーション全体でロールとPrincipal-to-Roleのマッピングを共有するため、J2EEアプリケーションに含まれているすべてのモジュールを知っていることが求められます。たとえば、application.xmlで「Administrator」

というロールを設定し、jeus-application-dd.xmlで「user2」というプリンシパルを割り当てたとします。このロール・マッピングはアプリケーションに含まれているすべてのEJBモジュールとWebモジュールに適用されます。

同様に、ロールとPrincipal-to-Roleのマッピングを特定のEJBのjeus-ejb-dd.xmlにだけ定義したとしても、設定されたロールとPrincipal-to-Roleのマッピングは.earに含まれているすべてのモジュールで共有されることができます。

注

J2EEアプリケーションにおけるすべてのPrincipal-to-Roleマッピングは、それがjeus-application-dd.xmlに設定されているか、もしくは特定のjeus-ejb-dd.xmlやjeus-web-dd.xmlに設定されているかに関係なく、アプリケーション・スコープを持ちます。

3.4.2. jeus-application-dd.xmlの設定

アプリケーション・レベルでのPrincipal-to-Roleマッピングは、jeus-application-dd.xmlに設定します。

Principal-to-Roleマッピングを定義するために、jeus-application-dd.xmlの<application>タグの下に以下のタグを追加します。

- <role-permission> (選択事項)

<role-permission>タグはPrincipal-to-Roleマッピングを定義し、オプションとしてExcluded、Uncheckedロールを設定できます。このタグはjeus-web-dd.xml、jeus-ejb-dd.xml、policies.xmlの場合と同じく動作します。

以下のような下位タグを持ちます。

タグ	説明
<principal>(0個以上)	ロールにマッピングされるプリンシパルの名前です
<role>(1個、必須事項)	ロール名を示します。これはロール・パーミッションを実装するJavaクラスのコンストラクターに1番目のパラメータとして渡されます。 ロール名はapplication.xmlに宣言された論理的なロール名と同一に設定します
<actions>(選択事項)	ロール・パーミッションの付加情報を提供します。このタグを設定すると、ロール・パーミッションを実装するJavaクラスのコンストラクターに2番目のパラメータとして渡されます
<classname>(選択事項)	ロール・パーミッションを実装しているJavaクラス名です。このクラスはjava.security.Permissionクラスのサブクラスであり、少なくとも1つのパラメータ(String rolename)を受信する、publicコンストラクターを持っている必要があります。

タグ	説明
	省略すると、デフォルトでjeus.security.resource.RolePermissionクラスが使われます
<excluded>(選択事項)	<p>エンプティ・タグです。このタグを設定すると、ロールは排除された(Excluded)ものとして扱われます。つまり、どのプリンシパルにも当該ロールを付与できません。</p> <p><principal>や<unchecked>タグより優先順位が高いです</p>
<unchecked>(選択事項)	<p>エンプティ・タグです。このタグを設定すると、ロールをチェックしない(Unchecked)ものとして扱われます。つまり、すべてのプリンシパルに当該ロールを付与できます。</p> <p><principal>タグより優先順位が高いです</p>

<application>タグの下にアプリケーションがデプロイされるセキュリティ・ドメインを設定する必要があります。デフォルト値として「SYSTEM_DOMAIN」が使われますが、アプリケーションが特別なセキュリティ・サービスを必要とする場合は、そのセキュリティ・サービスを含む新規ドメインを作成してデプロイします。

jeusadminの**deploy**コマンドでセキュリティ・ドメイン名を設定できます。

[例 3.8] J2EEアプリケーションのセキュリティの設定 : <<jeus-application-dd.xml>>

```

<application>
  . . .
❶  <role-permission>
    <principal>user2</principal>
    <role>Administrator</role>
  </role-permission>
❷  <role-permission>
    <role>Customer</role>
    </unchecked>
  </role-permission>
  . . .
</application>

```

上記の例は、以下のようなJ2EEアプリケーションのセキュリティ情報を含んでいます。

- ❶ 「user2」というプリンシパルは「Administrator」ロールにマッピングされており、デフォルトのロール・パーミッションを使います。
- ❷ すべてのプリンシパルに「Customer」ロールを付与します。(ロールがuncheckedになっているため)

参考

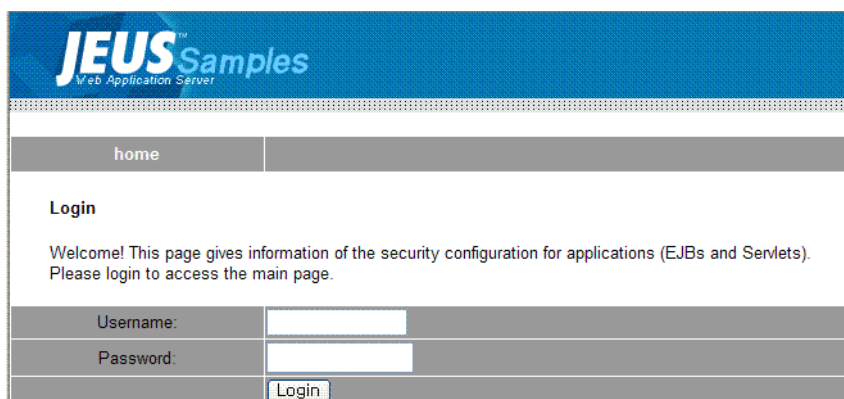
1. セキュリティー・ドメインを設定する方法については、「[2.2. セキュリティー・ドメインの定義](#)」を参照してください。
 2. カスタム・パーミッションを開発および設定する方法については、「[2.6. セキュリティー・システム・ポリシーの設定](#)」を参照してください。
 3. JEUSにJ2EEアプリケーションをデプロイする方法については、『JEUSサーバガイド』を参照してください。
-

3.5. 例

本節では、上記で説明した設定に基づいて作成した簡単な例を紹介します。

以下の画面はトップページに最初にアクセスするときにロードされるログイン画面です。該当するURLにChecked/パーミッションが設定されている場合にのみ表示されます。Unchecked/パーミッションが設定された場合、ログイン画面は省略されます。Excluded/パーミッションの場合には、認証に失敗してエラー画面が表示されます。

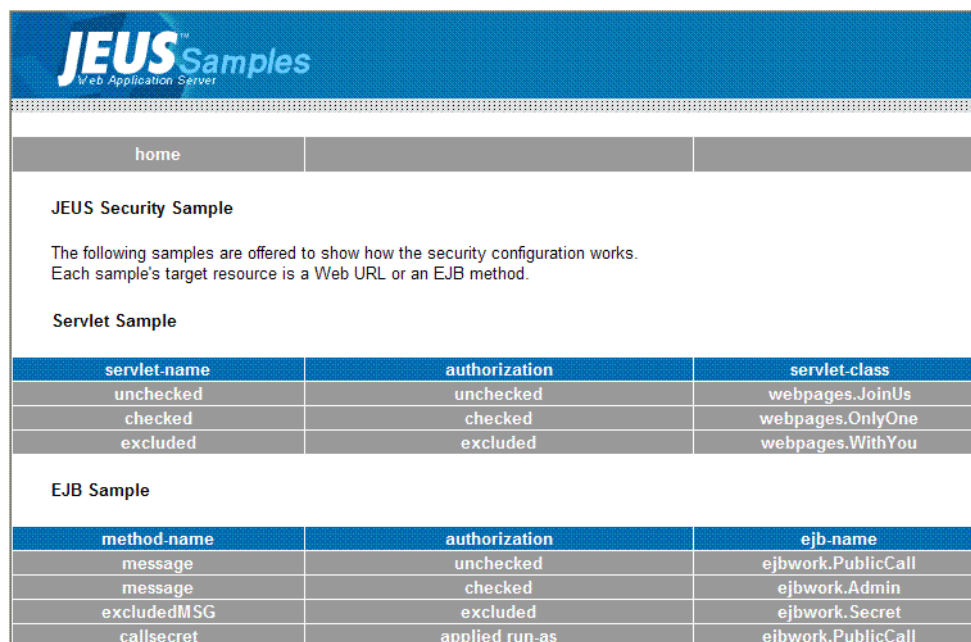
[図 3.2] ログイン画面



JEUS Samples Web Application Server	
home	
Login	
Welcome! This page gives information of the security configuration for applications (EJBs and Servlets). Please login to access the main page.	
Username:	<input type="text"/>
Password:	<input type="password"/>
	<input type="button" value="Login"/>

認証に失敗したか、権限のないアカウントでログインすると、エラーページが表示されます。権限を持つアカウントでログインに成功すると、以下のような画面が表示されます。

【図 3.3】トップ画面



home

JEUS Security Sample

The following samples are offered to show how the security configuration works.
Each sample's target resource is a Web URL or an EJB method.

Servlet Sample

servlet-name	authorization	servlet-class
unchecked	unchecked	webpages.JoinUs
checked	checked	webpages.OnlyOne
excluded	excluded	webpages.WithYou

EJB Sample

method-name	authorization	ejb-name
message	unchecked	ejbwork.PublicCall
message	checked	ejbwork.Admin
excludedMSG	excluded	ejbwork.Secret
callsecret	applied run-as	ejbwork.PublicCall

この例は、トップページで3つのサーブレットにリンクがかかっている、3つのEJBメソッドを呼び出します。ログインしたアカウントと設定ファイルによってアクセスできるものもあれば、できないものもあるので、色々設定を行い、複数のアカウントでログインして直接確認してみてください。

第4章 セキュリティー・システムAPIプログラミング

本章では、セキュリティー・システムAPIでプログラミングする方法について説明します。

4.1. 概要

セキュリティー・システムのAPIを使うと、ユーザー・アプリケーションに特別なセキュリティー機能を追加することができます。アプリケーション・レベルで登録したユーザーを自動でJEUSセキュリティー・システムに登録するレジストレーション・サーブレット(auto-registrationともいう)がその一例です。

アプリケーション開発者はセキュリティー・システムを開発する前に、そのセキュリティー・システムが標準のJ2EEセキュリティー・モデルとJEUSのセキュリティー・サービスが必要とするセキュリティー機能を提供しているか確認します。セキュリティーAPIを使ってプログラムを開発すると、J2EEサーバー同士の互換性が低下します。J2EEサーバー間の互換性を維持するために、一般には標準J2EEセキュリティー・インターフェースのみを使用することを推奨します。

4.2. Java SEパーミッションの設定

悪意のあるユーザー・コード(サーブレット、EJB)からJEUSシステムを保護するために、セキュリティーAPIを使うことができます。

ユーザー・コードにセキュリティーAPIを使うのは、Java SEセキュリティー・マネージャーを使用するか、もしくはソースコード(サーブレット、EJBなど)がSecurityCommonService.loginDefault(Subject)を使って、ログインに成功した場合です。このときのサブジェクトは、ターゲットのセキュリティー・ドメインのaccounts.xmlにあらかじめ設定されたユーザーのサブジェクトで、policies.xmlに設定されたJEUSパーミッションを保持しています。

各ファイルの設定方法については、以下を参照してください。

- Java SEセキュリティー・マネージャーとJava SEポリシー・ファイル : 「[2.7.1. JavaSE SecurityManager の設定](#)」
- accounts.xml : 「[2.5. セキュリティー・システムのユーザー情報の設定](#)」
- JEUSセキュリティー・システムのpolicies.xml : 「[2.6. セキュリティー・システム・ポリシーの設定](#)」と参考文献

4.3. 基本API

アプリケーションをプログラミングのレベルでセキュリティー・システムと連動する際には、**jeus.security.base** パッケージのクラスが重要な役割をします。

クラス	説明
jeus.security.base.Subject	ユーザーを表します。 サブジェクトはただ1つのメイン・プリンシパルを保持し、メイン・プリンシパルがサブジェクトのID(ユーザー名)として扱われます。 jeus.security.base.Subjectクラスに複数のString属性値が渡されることもあります
jeus.security.base.CredentialFactory	サブジェクト・クラスのメンバー変数であり、サブジェクトの実際の資格証明を作成するときに使われます。 たとえば、PasswordFactoryクラスはパスワードの資格証明インスタンスを作成し、JKSCertificateFactoryクラスはJKSキーストアから証明書を取得して、証明書の資格証明インスタンスを作成します
jeus.security.base.Policy	1つのPrincipal-to-Roleマッピングと複数のRole-to-Resourceマッピングを表します。PermissionMapsをメンバー変数として持ちます
jeus.security.base.PermissionMap	java.security.Permissionインスタンスのコンテナに当たり、Policyクラスのメンバー変数です
jeus.security.base.Role	論理的なロールを示すインターフェースです。 Role-to-Resourceパーミッション・マップではロール・インスタンスがリソース・パーミッションにマッピングされます。同様にPrincipal-to-Roleパーミッション・マップではプリンシパルがロール・パーミッションにマッピングされます
jeus.security.base.SecurityCommonService	サブジェクトを認証し、サブジェクトに対するパーミッションをチェックします
jeus.security.base.SecurityException	ログイン失敗、認証失敗、権限チェックの失敗など、セキュリティーを違反したときに発生する例外です
jeus.security.base.ServiceException	セキュリティー・システムで深刻なランタイム・エラーが起きたときに発生する例外です

4.4. リソースAPI

リソースと関連しては、jeus.security.baseパッケージの基本クラス以外に、**jeus.security.resource**パッケージのクラスも重要な役割をします。

クラス	説明
jeus.security.resource.PrincipallImpl	java.security.Principalインターフェースの実装クラスです
jeus.security.resource.Group PrincipallImpl	グループ・プリンシパルを表すPrincipallImplのサブクラスです。ネストされたグループのメンバーを管理するjava.security.acl.Groupインターフェースの実装クラスです
jeus.security.resource.Password	単純なパスワード資格証明インスタンスです。PasswordFactoryによって生成されます
jeus.security.resource.PasswordFactory	パスワード資格証明を作成するCredentialFactoryです
jeus.security.resource.Lock	一種の資格証明で、サブジェクトに対してロックをかけます。 ロックのかかったサブジェクトは常にログインに失敗します
jeus.security.resource.LockFactory	ロック資格証明(Lock Credential)を作成するCredentialFactoryです
jeus.security.resource.ExpiryTime	一種の資格証明で、サブジェクトの有効期限を設定します。 期限切れになったサブジェクトによるログインはすべて失敗となります
jeus.security.resource.ExpiryTimeFactory	有効期限を作成するCredentialFactoryです
jeus.security.resource.RoleImpl	ロール・インターフェースを実装したクラスです
jeus.security.resource.RolePermission	特定のプリンシパルが特定のロールに属することを示すjava.security.Permissionのサブクラスです。ロール・パーミッションはPrincipal-to-Roleのマッピングを表す際に使用されます
jeus.security.resource. TimeConstrainedRolePermission	ロール・パーミッションのサブクラスです。現在時間が、設定した時間の範囲内にあるときにのみ、プリンシパルがこのクラスのスーパー・クラスが示すロールに属します
jeus.security.resource.Resource Permission	java.security.Permissionのサブクラスです。このクラスは、ロールがリソースにアクセスして特定のアクションを実行できることを意味します。 したがって、リソース・パーミッションはRole-to-Resourceのマッピングに使用されます

参考

各クラスの詳細については、Javadocと[参考文献](#)を参照してください。

4.5. SPIクラス

セキュリティー・システムのサービスの作業を行うには、**jeus.security.spi** パッケージのSPIクラスを使います。

クラス	説明
jeus.security.spi.AuthenticationRepositoryService	サブジェクト・リポジトリからサブジェクトを追加、削除、照会する際に使われます。このAPIを使用すると、サブジェクト(ユーザー)をプログラム内に追加することができます
jeus.security.spi.AuthorizationRepositoryService	ポリシー・リポジトリからポリシー・データを追加、削除、照会する際に使われます。このAPIを使用すると、プログラム内でパーミッションを追加することができます

参考

各クラスの詳細については、Javadocと[参考文献](#)を参照してください。SPIクラスの詳細情報は、「[第5章 カスタム・セキュリティー・サービスの開発](#)」を参照してください。

4.6. 例

以下の例は、セキュリティーAPIを使ったプログラムの一部です。

```
// Login the CodeSubject so that security checks are
// disabled (so that we can modify the Subject and Policy
// stores)
SecurityCommonService.loginCodeSubject();

// Make Subject with Principal "pete"
Principal petePrincipal = new PrincipalImpl("pete");
Subject pete = new Subject(petePrincipal);

// Make password "petepw" for Subject "pete"
PasswordFactory pf = new PasswordFactory("petepw");
pete.getCredentialFactories().add(pf);

// Add new Subject to the Subject store
AuthenticationRepositoryService.addSubject(pete);

// Make a new Policy
Policy policy = new Policy();

// Make role "someRole"
Role someRole = new RoleImpl("someRole");
```

```

// Make a RolePermission for role "someRole"
Permission rolePermission = new RolePermission(someRole);

// Add the RolePermission for "someRole" to the Policy
policy.getRolePolicy().addPermission(
    rolePermission, new Object[] {petePrincipal}, false, false);

// Create a ResourcePermission for resource "rsc1" with actions
// "action1" and "action2"
Permission rscPermission =
new ResourcePermission("rsc1", "action1,action2");

// Add the ResourcePermission to the Policy using
// context id "ctx1"
policy.getResourcePolicy("ctx1", true).addPermission(
    rscPermission, new Object[] {someRole}, false, false);

// Add the new Policy to the Policy store
AuthorizationRepositoryService.addPolicy(policy);

// Logout the CodeSubject so that security checks are
// enabled again
SecurityCommonService.logout();

// Make a Subject to be logged in
Subject pete2 = Subject.makeSubject("pete", "petepw");

// Login Subject "pete" (should succeed since we added
// "pete" earlier)
SecurityCommonService.loginDefault(pete2);

// Check ResourcePermission "rsc1" for current Subject ("pete")
// Should succeed since we added Policy for this above
SecurityCommonService.checkPermission(
    "ctx1", new ResourcePermissin("rsc1", "action2");

// Print the name of the current Subject ("pete")
System.out.println(
SecurityCommonService.getCurrentSubject().getPrincipal().getName());

// Logout "pete"
SecurityCommonService.logout();

```


第5章 カスタム・セキュリティ・サービスの開発

本章では、カスタム・セキュリティ・サービスを開発する方法について説明します。カスタム・セキュリティ・サービスはJEUSセキュリティ・システムの主な特徴でもあります。

5.1. 概要

カスタム・セキュリティ・サービスを利用すると、JEUSセキュリティ・システムと様々な外部のセキュリティ・システム、セキュリティ・データのリポジトリを容易に統合することができます。

以下は、カスタム・セキュリティ・サービスの開発に関連する重要な概念です。詳しくは各節で説明します。

- [サービス・クラス](#)
- [実装の基本パターン](#)
- [11のSPIクラス](#)
- [カスタム・セキュリティ・サービスの設定方法](#)

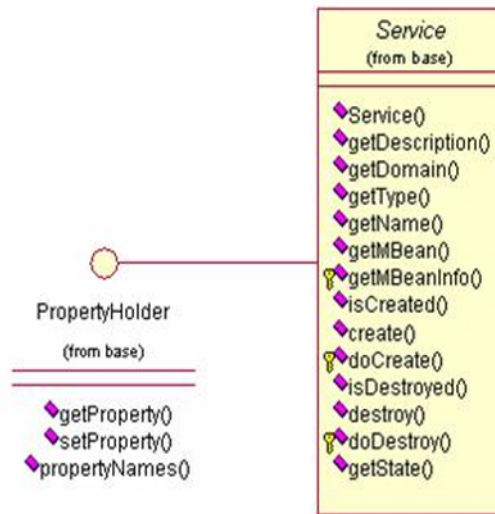
5.2. サービス・クラス

プラグブル(pluggable)なセキュリティ・アーキテクチャーにおいて、もっとも基本になるクラスは `jeus.security.base.Service` クラスです。

サービス・クラスは、セキュリティ・サービスを実装しようとするクラスを拡張するための抽象クラスです。現在、`jeus.security.spi` パッケージにあるすべてのSPIクラスもこのクラスを拡張したものです。

以下は、サービス・クラスのクラス・ダイアグラムです。

【図 5.1】 サービス・クラスのダイアグラム



サービス・クラスはすべてのセキュリティ・サービスに共通して適用される以下の項目を含んでいます。

- **Service Description**

サービス・クラスは、提供するサービスの内容について簡単に説明する、String型のディスクリプション (description)を提供します。サービスについての説明は、**getDescription()**メソッドで取得できます。

- **Service Domain**

ランタイムに生成されたインスタンスは特定のドメインに割り当てられます。したがって、ドメインは本質的にサービス・インスタンスの集合体といえます。サービスのドメインは、**getDomain()**メソッドで取得できます。

- **Service Type**

サービスの実装クラスはただ1つのタイプを持ちます。タイプとは、サービス・セット(Service set)から特定のサービスのインスタンスを取得するためのマークです。

たとえば、特定のサービス実装クラスが認証機能を提供するとしたら、他のセキュリティ・サービスでこの認証サービス・クラスを使うために、ドメインにそのクラスをリクエストする必要があります。リクエスト時に、サービス・タイプを渡すことで、ドメインが正しいサービス・インスタンスを探索して返すことができます。

サービス・タイプは**getType()**メソッドで取得できます。ただ、getType()メソッドは抽象メソッドであるため、サブクラスで実装する必要があります。サービス・タイプは実際にはjava.lang.Classタイプで、jeus.security.spiパッケージに含まれているSPIクラスのクラス・インスタンスです。各SPIクラスでは、getType()メソッドをファイナル(final)メソッドとして実装しています。つまり、SPIのサブクラスではgetType()メソッドを再定義することができません。

- **Service**

サービスの名前は**getName()**メソッドで取得できます。

● Service MBean

サービス・クラスは、サービス・インスタンスの管理時に使われるJMX MBeanと関連付けられています。

MBeanは`getMBean()`メソッドで取得できます。サブクラスでデフォルトのMBeanでなく他のMBeanを返すには、このメソッドを再定義する必要があります。

基本的にMBeanは、保護(protected)メソッドで宣言された`getMBeanInfo()`から返されたMBeanInfoに基づいて作成されます。サブクラスはこのメソッドを再定義して、デフォルト以外のMBeanInfoインスタンスを返すようにすることができます。メソッドを再定義するとき、スーパー・クラスのMBeanInfoを必ず含むようにします。

● Service State

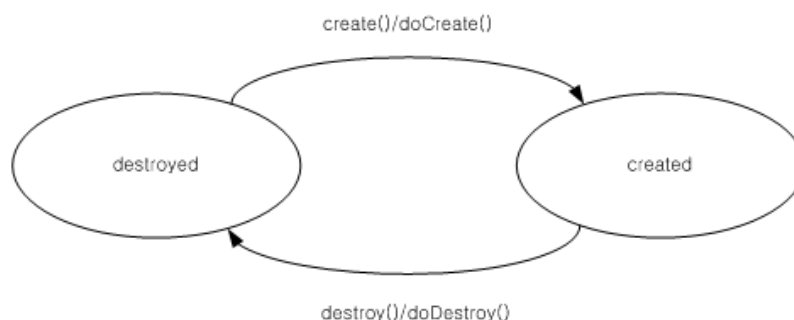
サービスの実装クラスには、createdとdestroyedの2つの状態があり、`create()`メソッドと`destroy()`メソッドを呼び出して状態を切り替えることができます。この2つのメソッドは抽象メソッドの`doCreate()`と`doDestroy()`を呼び出して実際の業務を任せます。したがって、サービスの実装クラスは`doCreate()`メソッドと`doDestroy()`メソッドを必ず実装する必要があります。

`doCreate()`メソッドと`doDestroy()`メソッド内には、サービスを初期化し消滅するコードが含まれています。典型的な`doCreate()`メソッドにはデータベース・コネクションのようなリソースを取得するコードが含まれており、`doDestroy()`メソッドには取得したリソースを解除するコードが含まれています。

サービス・クラスのカレント・ステートは、`isCreate()`メソッドと`isDestroyed()`メソッドを呼び出すことで確認できます。また、カレント・ステートをString型で返す`getState()`メソッドも持っています。

以下の図は、サービス・クラスのステート・チャートです。

[図 5.2] サービス・クラスのステート・チャート



● Service Properties

サービス・クラスはname-value組の属性を持ちます。この属性の設定および取得には、`jeus.security.base.PropertyHolder`インターフェースを使用します。そのため、サービスの実装クラスはこのインターフェースを実装する必要があります。

属性はサービス・インスタンスの初期化に使われます。サービス・インスタンスが生成されて、`create()`メソッドが呼び出される前に、属性が設定されます。その後、`doCreate()`メソッドを呼び出すと、設定した属性値をクエリーで取得し、サービス・インスタンスを適切に初期化します。

5.3. カスタム・セキュリティ・サービスの実装パターン

本節では、カスタム・セキュリティ・サービスの実装方法について簡単に説明します。

カスタム・セキュリティ・サービスを実装するには、通常、以下の手順に従います。

1. セキュリティ・システムとセキュリティ・システムのアーキテクチャーを予め十分理解している必要があります。
2. カスタム・セキュリティ・サービスが提供するセキュリティ・サービスを確認します。
3. 該当する属性をもっているSPIクラスを選択します。主要なSPIクラスの詳細については、「[5.4. SPIクラス](#)」を参照してください。
4. 該当するSPIクラスに関する文書を理解します。(Javadoc、[参考文献](#)、「[5.4. SPIクラス](#)」の説明を参照します。)
5. SPIクラスのサブクラスを作成し、サブクラスで以下のメソッドを実装します。そして、必ずパラメータを持たない、publicコンストラクターを提供します。
 - a. 選択的にサービスの初期化に使われる属性を定義します。各属性はpublic static final String型で、それぞれが何を意味するかを文書化します。
 - b. doCreate()メソッドを実装します。このメソッドはセキュリティ・サービスの開始時に1回呼び出され、リソース割り当てのような一般的な初期化を行います。このメソッドはgetProperty()メソッドを使って設定値を読み込みます。getProperty()メソッドのパラメータは、以前の段階で設定した属性名です。
 - c. doDestroy()メソッドを実装します。このメソッドはサービスが終了する前に1回だけ呼び出されます。このメソッドは、doCreate()メソッドで割り当てられたリソースを解除したり、ファイルにログを残すなど、通常のクリーンアップ作業を実行します。
 - d. SPIクラスのすべての抽象クラスをJavadocの説明どおり実装します。
 - e. JMXを使ってサービス管理用のメソッドを選択的に実装できます。getMBean()メソッドまたはgetMBeanInfo()メソッドを実装します。
6. SPIを実装したクラスをコンパイルします。
7. 「[第2章 セキュリティ・システムの設定](#)」の説明に従って、新規のセキュリティ・サービスをJEUSに登録します。

5.4. SPIクラス

本節では、jeus.security.spiパッケージに含まれている多様なSPIクラスについて説明します。

以下は、jeus.security.spiパッケージ内に定義されているSPIクラスのリストです。カーディナリティ(Cardinality、濃度)は、各ドメインに該当するSPIクラスが何個まで存在できるかを意味します。

[表 5.1] jeus.security.spiパッケージ内の標準SPIクラス

クラス名	目的	カーディナリティ
SecurityInstaller	セキュリティー・システムをインストールおよびアンインストールします。 全体JVMに対してただ1つだけ存在する必要があります	1
SubjectValidationService	ログイン前にサブジェクトの資格証明が有効かどうかをチェックします	0個以上
SubjectFactoryService	ログイン前にカスタム・サブジェクトを生成します	1
AuthenticationService	ログイン前にサブジェクトを認証します	1
AuthenticationRepositoryService	サブジェクトをサブジェクトのリポジトリに追加、削除、照会します	1
IdentityAssertionService	cert-user-map.xmlファイル情報を利用して資格証明をサブジェクトにマッピングします	0個以上
CredentialMappingService	トラストストア情報を利用して資格証明をサブジェクトにマッピングします	0個以上
CredentialVerificationService	サブジェクトの持っている資格証明のうち最低1つでも有効であるかを確認します	0個以上
AuthorizationService	サブジェクトが特定のロールまたはリソースにアクセスできる権限があるかをチェックします	1
AuthorizationRepositoryService	ポリシーをリポジトリに追加、削除、照会します。	1
EventHandlingService	セキュリティー・イベントに対するカスタム・イベントのハンドラー、多様なセキュリティー監査を実装します	0個以上

参考

SPIクラスの詳細については、[参考文献](#)を参照してください。

5.4.1. SubjectValidationService SPI

jeus.security.spi.SubjectValidationService SPIは、サブジェクトの資格証明(Credential)が有効かどうかをチェックします。資格証明は有効でない場合もあります。資格証明が有効でないということは、サブジェクトが有効でないことを意味し、そのようなサブジェクトはログインができません。

SubjectValidationServiceの典型例は、サブジェクトが「lock」資格証明を持っているかどうかをチェックすることです。もし「lock」資格証明を持っていれば、サブジェクトはロックがかかったものとして扱われ、それ以上のログイン・プロセスは行われません。

SubjectValidationService.checkValidity(Subject)メソッドは通常、ログインの際に呼び出されます。もし、このメソッドでSecurityExceptionが発生すると、すべてのログイン・プロセスは失敗となります。サブジェクトに「lock」資格証明を自動で設定するには、EventHandlingServiceを使用します。詳細については、[「5.4.10. EventHandlingService SPI」](#)を参照してください。

認証(パラメータとして渡されたサブジェクトが実際にそのサブジェクトかどうかを検証すること)とサブジェクトの有効性チェックは別物です。ログイン・プロセスでは2つのサービスが両方使われます。

各ドメインには0個以上のSubjectValidationServiceインスタンスを設定できます。全体のSubjectValidationServiceにおいてSecurityExceptionが1つも発生しない場合、そのサブジェクトは有効と判断され、ログイン・プロセスが引き続き進行されます。しかし、SecurityExceptionが1つでも発生すると、全体の有効性チェックは失敗とみなされ、ログイン・プロセスはそれ以上進みません。

注意

SubjectValidationService SPIはCredentialVerificationServiceと似ていますが、その機能には若干の差があります。

SubjectValidationServiceは資格証明をチェックし、サブジェクトが有効かどうかを判断するのに対し、CredentialVerificationServiceは該当するサブジェクトが実際のサブジェクトであることを証明する資格証明を少なくとも1つ以上持っているかどうかをチェックします。したがって、SubjectValidationServiceはサブジェクトの認証に成功した後に使われる反面、CredentialVerificationServiceはAuthenticationServiceの認証の途中で使われます。

5.4.2. SubjectFactoryService SPI

jeus.security.spi.SubjectFactoryService SPIクラスは外部からの情報に頼らずにサブジェクトを作成する特別なSPIです。

SubjectFactoryService SPIは、一般的にサブジェクトや資格証明をパラメータとして取得せずに、SubjectFactoryServiceを使ってサブジェクトを作成するときに使われます。たとえば、典型的な実装クラスは、プロンプトにユーザー名とパスワードを入力すると、その情報をベースにサブジェクトを生成して返します。

SubjectFactoryServiceは、ログイン・メカニズムにおいてパスワード以外の資格証明タイプもサポートするために生成されたSPIです。

5.4.3. AuthenticationService SPI

`jeus.security.spi.AuthenticationService`はサブジェクトを認証するためのクラスです。つまり、パラメータとして渡されたサブジェクトがサブジェクトのリポジトリに登録されている実際のサブジェクトと一致するかを検証します。

認証手順は以下のとおりです。

1. `jeus.security.spi.CredentialMappingService.getSubjectName(Object)`メソッドを呼び出してサブジェクトの資格証明がどのユーザーにマッピングされるかを確認します。
2. `jeus.security.spi.AuthenticationRepository.getSubject(String username)`メソッドを呼び出して、実際に登録されているサブジェクトをサブジェクトのリポジトリからローカルにコピーします。これをローカル・サブジェクトといいます。
3. ローカル・サブジェクトがヌルでない場合、ローカル・サブジェクトの資格証明と、認証したいサブジェクトの資格証明が一致するかを比較します。この比較は`jeus.security.spi.CredentialVerificationService.verifyCredentials(Subject, Subject)`メソッドを呼び出して実行されます。

ケースによっては`equals(Object)`を使って資格証明を比較します。しかし、これは柔軟な方法とはいえません。

4. 上記の手順がすべて成功すると、ローカル・サブジェクトが`authenticate(Subject)`メソッドから返され、そのサブジェクトを利用して`SecurityCommonService`にログインします。

ドメイン当たり1つ以上の`AuthenticationService`を設定することができます。設定された`AuthenticationService`のうち、1つでもサブジェクトの認証に成功すると、すべての`AuthenticationService`が認証されたものとみなします。つまり、ただ1つの`AuthenticationService`でもサブジェクトの認証に成功したら、他の`AuthenticationService`にはクエリーを投げません。

5.4.4. AuthenticationRepositoryService SPI

`jeus.security.spi.AuthenticationRepositoryService` SPIは、サブジェクトのリポジトリからサブジェクトを追加、削除、照会するメソッドを保持しています。このSPIは`AuthenticationService`を実装するクラスで使われる他、J2EEアプリケーション・コード内で直接使われます。

たとえば、Webサイトを通じて新規のユーザーが登録するたびに自動でサブジェクトをサブジェクト・リポジトリに追加するコードを作成するにはこのSPIが必要です。

本来、`AuthenticationRepositoryService`の最大の目的は、ランタイムに`jeus.security.base.Subject`インスタンスを特定のリポジトリに格納することです。典型的なリポジトリとしてXMLファイルやデータベースが挙げられます。

通常は、`AuthenticationRepositoryService` SPIを別途に実装する必要はありません。しかし、デフォルトの`AuthenticationService`を使うか(ユーザーの認証プロセスにおいて`Authentication`

RepositoryService.getSubject(String)メソッドを呼び出すためです)、J2EEアプリケーション・コードで直接使うためには必ず実装する必要があります。

ドメイン当たり1つのAuthenticationRepositoryServiceインスタンスのみ設定できます。(現在、複数のAuthenticationRepositoryServiceを設定することはできません。)

5.4.5. IdentityAssertionService SPI

jeus.security.spi.IdentityAssertionService SPIはサブジェクトのユーザー名(username)が提供されない場合に必要です。つまり、メイン・プリンシパルがヌルの場合です。この場合にはサブジェクトから資格証明を取得して、cert-user-map.xmlの情報をベースにその資格証明がどのユーザー名にマッチするかを確認することになります。マッチング済みのユーザー名はそれ以降の認証プロセスで使われます。

典型例としてjava.security.Certificateインスタンスが挙げられます。このインスタンスは証明書資格証明(Certificate Credential)を使うため、メイン・プリンシパルを保持しません。この場合、認証途中に、サブジェクトから証明書資格証明を取得し、IdentityAssertionService.getIdentity(Object)メソッドにパラメータとして渡します。

このメソッドは再びIdentityAssertionService.doGetIdentity(Object)を呼び出します。このメソッドの内部でcert-user-map.xmlに定義された証明書の属性キーの値に対応するユーザー名をバック・トラッキングし、マッチするユーザー名を返します。

5.4.6. CredentialMappingService SPI

jeus.security.spi.CredentialMappingServiceはサブジェクトのユーザー名(username)が提供されない場合に必要です。つまり、メイン・プリンシパルがヌルの場合です。この場合にはサブジェクトから資格証明を取得し、その資格証明がどのユーザー名にマッチするかを確認します。マッチング済みのユーザー名はそれ以降の認証プロセスで使われます。

典型例としてjava.security.Certificateインスタンスが挙げられます。このインスタンスは証明書資格証明を使うため、メイン・プリンシパルを保持しません。この場合、認証途中に、サブジェクトから証明書資格証明を取得し、CredentialMappingService.getSubjectName(Object)メソッドにパラメータとして渡します。

このメソッドは再びCredentialMappingService.doGetSubjectName(Object)を呼び出します。このメソッドの内部で証明書リポジトリの証明書を使ってユーザー名をバック・トラッキングし、マッチするユーザー名を返します。

5.4.7. CredentialVerificationService SPI

jeus.security.spi.CredentialVerificationService SPIは新しいタイプのプルーフ資格証明(Proof Credential)をサポートするために使われます。

プルーフ資格証明(Proof Credential)は、パラメータとして渡されたサブジェクトが実際に存在するサブジェクトかを検証するために使われる資格証明です。プルーフ資格証明の典型例がパスワードであり、`jeus.security.resource.Password`クラスで実装されます。

`CredentialVerificationService` SPIは、サブクラスが必ず実装しなければならない唯一のメソッドである`doVerifyCredentials(Subject, Subject)`を宣言しています。このメソッドのシグネチャで最初に出るサブジェクトはリファレンス・サブジェクト(reference Subject)といい、サブジェクトのリポジトリに登録されている実際のサブジェクトの資格証明を保持しています。2番目に出るサブジェクトはプルーフ・サブジェクト(proof Subject)といい、プルーフ資格証明を持っています。`doVerifyCredentials(Subject, Subject)`メソッドは2つのサブジェクトの資格証明を比較して、1つでも一致するものがあるかを確認します。

資格証明は多様な方法でマッチされます(一般に`equals(Object)`メソッドだけでは不十分です)。もし、両方のサブジェクトの間で1つでも一致する資格証明があれば、メソッドは返されます。そうでない場合は、`jeus.security.base.SecurityException`が発生します。

注意

ケースによっては、リファレンス・サブジェクトの情報がなくても済む場合があります。つまり、特定のプルーフ資格証明の場合、プルーフ・サブジェクトの情報だけで資格証明を検証することができます。たとえば、特定の証明書資格証明の場合がそれに当たります。

`jeus.security.resource.Password`をマッチする`CredentialVerificationService`は、`jeus.security.impl.verification.PasswordVerificationService`クラスです。

ドメイン当たり0個以上の`CredentialVerificationService`を設定することができます。最低1つの`CredentialVerificationService`でマッチが確認されれば、全体の`CredentialVerificationService`が成功したとみなします。そうでない場合は、`SecurityException`が発生し、全体の検証が失敗とみなされ、結果的に認証プロセスも失敗します。

5.4.8. AuthorizationService SPI

`jeus.security.spi.AuthorizationService` SPIはセキュリティー・システムにおける権限チェックに関連するメソッドを定義しています。このSPIの目的は、あるサブジェクトが特定のアクションを実行する権限を持つかどうかを確認することです。

典型的な実装クラスは`jeus.security.spi.AuthorizationRepositoryService.getPolicy(contextId)`メソッドを呼び出して、その結果として返された`jeus.security.base.Policy`を分析して確認を行います。`AuthorizationRepositoryService` SPIを使わない、他の方法による実装も可能です。

ドメイン当たり1つ以上の`AuthorizationService`を設定することができます。最低1つの`AuthorizationService`で正数値が返されれば、すべての権限チェックのプロセスで成功したとみなします。つまり、最低1つの`AuthorizationService`でパーミッションが許可されれば、他の`AuthorizationService`に追加で権限を確認するクエリーは投げません。

5.4.9. AuthorizationRepositoryService SPI

jeus.security.spi.AuthorizationRepositoryService SPIはポリシー・リポジトリにjeus.security.base.Policyインスタンスを追加、削除、照会するメソッドを含んでいます。このSPIはAuthorizationService実装クラスで使われます。また、J2EEアプリケーション・コード内で特定のイベントが発生する際に、直接ポリシーをポリシー・リポジトリに追加または削除するために使用します。

本来、AuthorizationServiceの最大の目的はランタイムにjeus.security.base.Policyを特定タイプのポリシー・リポジトリに保存することです。典型的な実装クラスは、XMLファイルやデータベース、LDAPサーバーにポリシーを保存します。

AuthorizationRepositoryService SPIを実装して使用するのは選択事項です。ただし、デフォルトのAuthorizationServiceサービスを使う場合は、このサービスがAuthorizationRepositoryService.getPolicy(String)メソッドを呼び出すため、必ずこのSPIを実装する必要があります。

ドメイン当たり1つのAuthorizationRepositoryServiceのみ設定できます。現在、複数のAuthorizationRepositoryServiceをドメインに設定することはできません。

5.4.10. EventHandlingService SPI

EventHandlingService SPIは各種のセキュリティ・イベントをキャプチャして処理するインターフェースです。

EventHandlingService SPIを使用すると、セキュリティ・イベントの発生時にログを残し、マネージャーに自動でメールを送り、サブジェクトにロックをかけるなどの作業を簡単に実装することができます。

セキュリティ・サービスで発生したjeus.security.base.Eventタイプのイベントは、同じセキュリティ・ドメインに設定されているすべてのEventHandlingServiceに通知されます。これにより、EventHandlingServiceは、イベントの内容によってそれぞれ異なる方式で処理します。

EventHandlingServiceの例として、サブジェクトにロックをかける場合を想定します。AuthenticationService(認証サービス)が失敗するたびに、security.authentication.failedタイプのイベントが発生します。すると、lockout EventHandlingServiceがイベントをキャッチし、handleEvent(Event)メソッドを実行させます。このメソッドの内部ではログイン回数をカウントして、その回数が特定の値(たとえば3回)以上になると、該当するサブジェクトに「ロック」資格証明を設定します。

以降、SubjectValidationServiceはサブジェクトにロックがかかっていることを確認し、結果的にこのサブジェクトのログインの試みをすべて失敗させます。SubjectValidationServiceとEventHandlingServiceを結合すると、一度に3回以上ログインに失敗したサブジェクトにロックをかけ、それ以上はシステムにログインを試みられないようにする機能を追加できます。

EventHandlingService SPIがもっとも頻繁に使われるのはイベントを記録するロガーを実装する場合です。イベントは一般のテキストやXMLファイルに記録されたりもしますが、ときには否認防止(non-repudiation)のために暗号化されたファイルを使うこともあります。

ドメイン当たり0個以上のEventHandlingServiceを設定することができます。

5.4.11. SPI実装間の依存関係

SPIクラス間では依存が存在し得ます。たとえば、AuthenticationServiceは、AuthenticationRepositoryServiceに依存します。

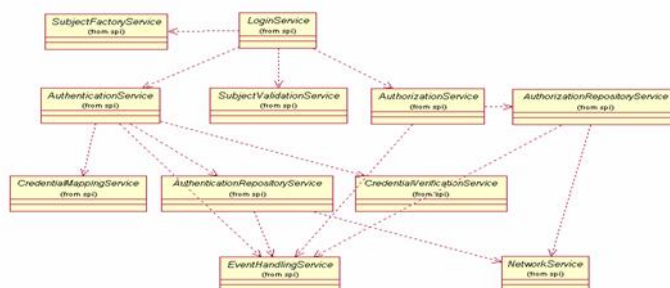
参考

依存とは、1つのSPI実装クラスが他のSPI実装クラスのstaticメソッドを呼び出すことです。AuthenticationServiceの実装クラスがAuthenticationServiceImplであるとする、このクラスはAuthenticationRepositoryService.getSubject()を呼び出してサブジェクトの情報を取得します。

SPIクラス間で依存が存在し得るということは、依存が存在しないこともあり得ることを意味します。つまり、依存が存在するかどうかは、SPIクラスをどう実装するかにかかっています。どのSPIクラスも直接他SPIクラスのメソッドを呼び出しません(いくつかの些細な例外はあります)。一般的には、SPIを実装したクラスで他のSPI実装クラスのメソッドを呼び出すことで依存が成立します。

以下の図は、デフォルトのセキュリティ・システムの実装におけるSPIの実装クラス同士の依存を表すもので、クラス間の関係を単純化して示しています。

[図 5.3] デフォルトのセキュリティ・システムの実装におけるSPI実装クラス



5.5. セキュリティ・サービスの設定

新しいSPI実装クラスをコンパイルした後、そのクラスをJEUSセキュリティ・システムに登録する必要があります。登録方法については、「[2.3. セキュリティ・ドメイン構成要素の設定](#)」を参照してください。

第6章 JACCプロバイダーの使用

本章では、JACCの目的を紹介し、カスタムJACCプロバイダーを実装してJEUSセキュリティー・システムで使用方法について説明します。

6.1. 概要

JACC(Java Authorization Contract for Containers)は、J2EEバージョン1.4で初めて紹介され、現在のバージョンは1.5です。

JACCの背景には、以下の2つの基本目的があります。

- EJBとサーブレットの権限をチェックする際に標準のSPIを提供する。
- 従来のJ2SEセキュリティー・モデルとJ2EEセキュリティー・モデル間の調和を追求する。

JACCはEJBとサーブレットのアクセス権限を定義して処理する標準的な方法を提示しています。したがって、JACCプロバイダーを作成しておけば、JACCをサポートするJ2EEサーバーで使用できます。

6.2. JACCの規約

JACCの仕様は次の3つの細部規約で構成されています。

- プロバイダー設定の規約
- ポリシー設定の規約
- ポリシー決定および執行の規約

参考

JACC仕様(JSR-115)の詳細については、<http://java.sun.com/j2ee>のJACC 1.5仕様文書を参照してください。

6.2.1. プロバイダー設定の規約

プロバイダー設定の規約では、アプリケーション・サーバーのランタイム環境にJACCプロバイダーを統合する方法について記述しています。つまり、この仕様はJACCプロバイダーをJ2EEサーバーに登録する方法を定義します。通常、このプロセスは非常に簡単です。

1. `javax.security.jacc.policy.provider`システム属性にカスタムの`java.security.Policy`クラス名を設定します。JACCプロバイダーはカスタム`java.security.Policy`を含んでいます。
2. J2EEサーバーはこのクラス名を読み込んでインスタンスを作成した後、`java.security.Policy`タイプでキャストします。
3. 上記の手順が正常に完了すると、`java.security.Policy.setPolicy()`メソッドを呼び出して、デフォルトJ2SEポリシーのクラスを新規のJACCポリシーのクラスに交替します。
4. すべての権限チェックは新規登録されたJACCポリシーによって行われます。`java.security.Policy.getPolicy()`メソッドを使って現在のJACCポリシーを取得することができます。

参考

JACCプロバイダーの設定の規約については、JACC仕様を参照してください。

6.2.2. ポリシー設定の規約

ポリシー設定の規約では、J2EE DDファイル(`ejb-jar.xml`, `web.xml`)に設定されたセキュリティ制限(`security constraints`)を`javax.security.jacc`パッケージに定義された`java.security.Permission` setにマッピングする方法と、このようなパーミッションをJACCプロバイダー(カスタム`java.security.Policy`)に追加する方法について記述しています。

この規約は、EJBとサーブレットでJACCプロバイダーが権限付与に関連する決定を下せるようにします。

参考

ポリシー設定の規約は、Principal-to-Roleマッピングについては触れていません。単にJ2EE DDファイルに基づいて、Role-to-Resourceマッピングをどう行うかについて定義しているだけです。Principal-to-RoleマッピングはJACCプロバイダーを供給するベンダーごとに異なる方法で定義されています。

ポリシーの設定手順を簡単に説明すると、以下のとおりです。

1. システム属性(-D属性で設定)の`javax.security.jacc.PolicyConfigurationFactory.provider`に、カスタム`javax.security.jacc.PolicyConfigurationFactory`クラス名を設定します。

JACCプロバイダーはカスタム`javax.security.jacc.PolicyConfigurationFactory`を含んでいます。

2. J2EEサーバーはこの属性値を読み込み、当該クラスのインスタンス(PCF)を生成します。
3. サブレットとEJBモジュールをデプロイします。
4. デプロイメント・コードは、web.xmlとejb-jar.xml DDファイルを読み込んで設定されたセキュリティ制限項目をJACCパーミッションのインスタンスに変換します。このパーミッション・クラスはjavax.security.jaccパッケージに含まれています。この際の各パーミッション・インスタンスは、サブレットとEJBに設定されているRole-to-Resourceマッピングを示します。
5. デプロイメント・コードはPCF.getPolicyConfiguration()メソッドを呼び出し、javax.security.jacc.PolicyConfigurationタイプのインスタンス(PC)が返されます。
6. デプロイメント・コードは手順4で生成されたRole-to-ResourceパーミッションをPCの多様なメソッドを使ってPCに追加します。
7. すべてのパーミッションをPCに追加してから、PCのcommit()メソッドを呼び出します。

サブレット・モジュールとEJBモジュールがアンデプロイされるときに、PCのdelete()メソッドが呼び出されます。これにより、サブレット・モジュールとEJBモジュールに対するパーミッションが削除されます。

参考

ポリシー設定プロセスの詳細については、JACC仕様を参照してください。

6.2.3. ポリシー決定および執行の規約

ポリシー決定および執行の規約は、EJBとサブレットでアクセス権限に関連する決定をどう下し、執行するかについて記述しています。これは前述した2つの細部規約が満たされた後に適用されます。

ポリシーの決定および執行は以下のような手順で行われます。本節では、サブレットを例に挙げて説明しますが、EJBの場合も同様に行われます。

1. サブレットのページに対するリクエストを受信します。
2. サブレット・コンテナはjavax.security.jacc.PolicyContextクラスを使ってJACCコンテキスト情報を設定します。
3. サブレット・コンテナは2種類のJACC Webパーミッション(javax.security.jaccパッケージに定義されている)を生成します。このパーミッション・インスタンスは、現在リクエストしたサブレットに設定されているパーミッションを示します。
4. サブレット・コンテナは、サブレットのリクエスト側が上記の2種類のパーミッションを保持しているかを確認するために、JACCプロバイダーにクエリーを投げます。クエリーを解釈する方法は複数あります。

たとえば、`Policy.implies()`メソッドで確認できます。この際のパラメータには、リクエスト側のプリンシパルで初期化された`java.security.ProtectionDomain`がチェックするすべてのパーミッションが使われます。

5. JACCプロバイダーは、手順2で設定されたコンテキスト情報、手順3で生成したパーミッション・インスタンス、リクエスト側のプリンシパル、Principal-to-Roleマッピング、Role-to-Resourceマッピングをすべて使って、現在のリクエスト側がサーブレットのページにアクセスする権限を持っているか否かを判断します。
6. 権限チェックの結果が正数値であれば、サーブレット・コンテナはリクエスト側がサーブレット・ページにアクセスするように許可します。そうでない場合は、権限チェックに失敗したというエラーメッセージを返します。

参考

ポリシー決定および執行の規約の詳細については、JACC仕様を参照してください。

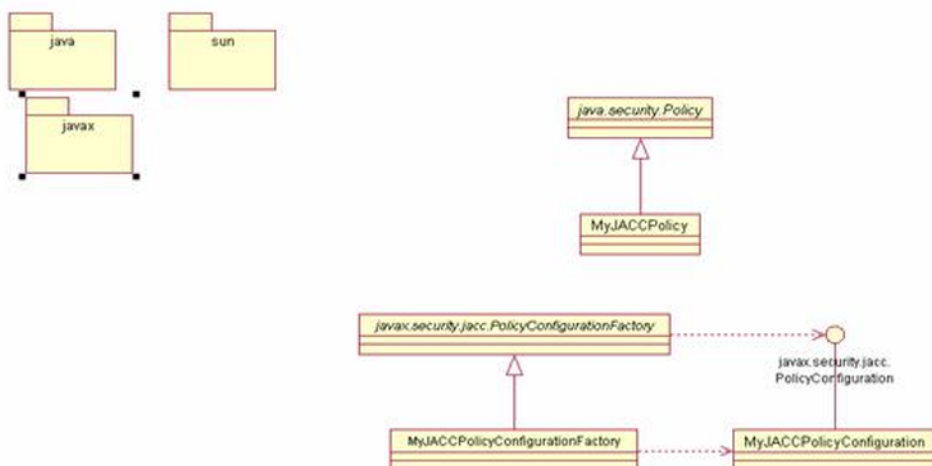
6.3. JACCプロバイダーの開発

本節では、カスタムJACCプロバイダーの開発方法と開発指針について説明します。

6.3.1. JACCプロバイダーの実装

以下は、カスタムJACCプロバイダーを実装するために必要なクラス間の関係を簡素化して表現したクラス・ダイアグラムです。このダイアグラムではクラス名を「MYJACC」で始めていますが、どう名づけても問題はありません。

【図 6.1】 JACCプロバイダー・クラス



JACCプロバイダーの実装時に必要なクラスは以下のとおりです。

- **java.security.Policy**

JACCプロバイダーで重要なことは、java.security.Policyのサブクラスを作成することです。このクラスは、JACCで実際に権限をチェックするために使われます。

java.security.Policyのサブクラスを作成するには、java.security.Policyを継承する新しいクラスを定義する必要があります(これをMyJACCPolicyとします)。その後、「[6.2.3. ポリシー決定および執行の規約](#)」で説明したポリシー決定および執行の規約を実装するために、implies()メソッドを再定義します。実装の際、権限チェックのクエリーを解釈するには、PolicyConfigurationインターフェースにより追加されたパーミッションとPrincipal-to-Roleマッピングを考慮する必要があります。

MyJACCPolicyはパラメータを持たないpublicコンストラクターを提供し、J2EEサーバーがインスタンスを容易に作成できるようにする必要があります。

- **javax.security.jacc.PolicyConfigurationFactory**

ポリシー設定の規約を実装するために、アプリケーション・サーバーはパーミッション・インスタンスをJACCのjava.security.Policyに追加する必要があります。この作業にはjavax.security.jacc.PolicyConfigurationインターフェースを使用します。また、PolicyConfigurationインスタンスを作成するためには、抽象クラスのjavax.security.jacc.PolicyConfigurationFactoryが必要です。

PolicyConfigurationFactoryのサブクラスを作成するには、javax.security.jacc.PolicyConfigurationFactoryを継承する新しいクラスを定義する必要があります(これをMyJACCPolicyConfigurationFactoryとします)。このクラスはgetPolicyConfiguration()を必ず再定義する必要があります。

MyJACCPolicyConfigurationFactoryクラスはパラメータを持たないpublicコンストラクターを提供して、J2EEサーバーが当該クラスのインスタンスを簡単に作成できるようにします。

- **javax.security.jacc.PolicyConfiguration**

ポリシー設定の規約を実装するために、アプリケーション・サーバーはパーミッション・インスタンスをJACCのjava.security.Policyに追加する必要があります。この作業にはjavax.security.jacc.PolicyConfigurationインターフェースを使用します。このインターフェースのインスタンスは、上述したようにjavax.security.jacc.PolicyConfigurationFactoryにより生成されます。

JACCプロバイダーはjavax.security.jacc.PolicyConfigurationを実装したクラスを含みます(この実装クラスをMyJACCPolicyConfigurationとします)。

カスタムjavax.security.jacc.PolicyConfigurationFactoryクラスであるMyJACCPolicyConfigurationFactoryは、常にMyJACCPolicyConfigurationインスタンスを返す必要があります。

参考

本節では各クラスの実装方法について簡単に説明しました。実装方法の詳細については、JACC仕様のポリシー決定および執行の規約と各クラスに関するJ2EE Javadocを参照してください。

6.3.2. JACCプロバイダーのパッケージング

通常、JACCプロバイダーとプロバイダーが参照するクラスをJARにまとめて、アプリケーション・サーバーに配置します。(これを「MYJACCProvider.jar」とします。)

アプリケーション・サーバーのパスにJACC Provider JARファイルのパスを含め、必要な場合は特定システムの属性値を設定します。このプロセスはアプリケーション・サーバーごとに若干の差があります。JEUSで設定する方法については、「[6.4. JEUSセキュリティ・システムとJACCプロバイダーの統合](#)」で説明します。

6.3.3. デフォルトのJACCプロバイダー

JEUSセキュリティ・システムは非常に単純なデフォルトのJACCプロバイダーを提供しています。一般にこのデフォルト・プロバイダーをJACC仕様のテスト以外の用途で使うことは推奨しません。前章で説明したデフォルトの承認プロバイダー(Authorization Provider)を使うことをお勧めします。

デフォルト以外のJACCプロバイダーを使用するには、商用化された製品は提供されていないため、JACC JARアーカイブを直接作成して、そのアーカイブにJACCプロバイダーのファイルを含める必要があります。

6.4. JEUSセキュリティ・システムとJACCプロバイダーの統合

本節では、JEUSセキュリティ・システムとJACCプロバイダーを統合する手順を説明します。

1. Principal-to-Roleマッパーを実装します。

JACCインターフェースはPrincipal-to-Roleマッピングについては何も触れておらず、Role-to-Resourceマッピングだけ定義しています。Principal-to-Roleマッピングを行うためには別途のインターフェースが必要であり、JEUSはjeus.security.impl.aznrep.JACCPPrincipalRoleMapperというインターフェースを提供しています。このインターフェースはただ1つのaddPrincipalRoleMapping(PermissionMap map, String policyId)メソッドを含んでいます。このメソッドはPrincipal-to-Roleマッピングを、policyIdに代表されるPolicyConfigurationに追加します。

注

Principal-to-RoleマッピングはJ2EEにおいてアプリケーションのスコープを持つことに注意してください。これは、同じJ2EEアプリケーションに属するすべてのPrincipal-to-Roleマッピングは1つのマップに統合されるためです。PermissionMapクラスとPermissionMapインスタンスをマージするために使われるadd()メソッドの詳細については、PermissionMapクラスのAPI文書を参照してください。

JJACCPPrincipalRoleMapperインターフェースを実装したクラスはパラメータを持たないpublicコンストラクターを提供します。そして、必ずJACC Provider JARの中に追加される必要があります。JACCPPrincipalRoleMapperインターフェースの詳細については、[参考文献](#)とJavadocを参照してください。

JACCPPrincipalRoleMapperがprincipal-to-roleマッピングを生成する手順は以下のとおりです。

- a. jeus.security.jacc.principalRoleMapperシステムの属性にjeus.security.jacc.principalRoleMapperを実装したクラス名を設定します。
- b. 設定が完了すると、jeus.security.impl.aznrep.JACCAuthorizationRepositoryServiceを実装したクラスがその属性値を読み込み、Class.forName(mapper Classname).newInstance()メソッドで該当するインスタンスを作成します。
- c. 生成されたインスタンスのaddPrincipalRoleMapping()メソッドを呼び出して、JEUS DDファイルに明示されたPrincipal-to-Roleマッピングを作成して追加します。

2. セキュリティー設定ファイルを設定します。

JEUSセキュリティー・システムは2つのアダプター・クラスを使って、JEUS native authorization APIとJACC authorization APIを関連付けます。

この2つのアダプター・クラスとそれぞれの役割は以下のとおりです。

- jeus.security.impl.azn.JACCAuthorizationService

コンテナでポリシーの決定および執行の規約を実装した部分です。権限チェックのためにjava.security.Policy.getPolicy().implies()メソッドを呼び出します。

- jeus.security.impl.aznrep.JACCAuthorizationRepositoryService

ポリシー設定の規約を実装した部分です。コンテナが作成したjeus.security.base.PolicyインスタンスをJACCプロバイダーの構成要素であるPolicyConfigurationインスタンスに追加する役割をします。

JACCを動作するには、この2つのセキュリティー・サービスを、domain.xmlのドメイン・サービスの定義に設定する必要があります。

[例 6.1] JACCセキュリティー設定ファイルの設定 : <<domain.xml>>

```
<?xml version="1.0"?>

<security-domains>
  <default-application-domain>JACC_DOMAIN</default-application-domain>
  ...
  <security-domain>
    <name>JACC_DOMAIN</name>
    <authorization>
      <jacc-service/>
    </authorization>
  </security-domain>
  . . .
</security-domains>
```

3. システム・パスにJACC Provider JARのパスを追加します。

システム・パスにJACC Provider JARファイルを追加するために、以下のディレクトリーにJARファイルを配置します。

```
JEUS_HOME/lib/system
```

4. Javaシステムの属性を設定します。

JACC規約およびJEUSでは、アプリケーション・サーバーがJACCプロバイダーを認識できるように、次の3つのJavaシステムの属性を規定しています。

- javax.security.jacc.policy.provider

JACCプロバイダーを示します。java.security.Policyを実装したクラス名です。

- javax.security.jacc.PolicyConfigurationFactory.provider

PolicyConfigurationインスタンスを生成およびロードする、PolicyConfigurationFactoryを実装したクラス名です。

- jeus.security.jacc.principalRoleMapper

jeus.security.impl.aznrep.JACCPrincipalRoleMapperインターフェースを実装したクラス名であり、JEUS DDファイルからPrincipal-to-Roleマッピングを作成します。

この3つのシステム属性は、すべてのサーバーに対し、domain.xmlの<jvm-option>に設定されている必要があります。

[例 6.2] JACCに対するJavaシステム属性の設定<<domain.xml>>

```
<?xml version="1.0"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <servers>
    <server>
      <name>server1</name>
      <!-- server JVM option -->

      <jvm-config>
        . . .
        <jvm-option>
          -Djavax.security.jacc.policy.provider=
            myprovider.MyJACCPolicy
        </jvm-option>
        <jvm-option>
          ?Djavax.security.jacc.PolicyConfigurationFactory.provider=
            myprovider.MyJACCPolicyConfigurationFactory</jvm-option>
        <jvm-option>
```



```
        ?Djeus.security.jacc.principalRoleMapper=  
        myprovider.MyJACCPrincipalToRoleMapper  
    </jvm-option>  
</jvm-config>  
    . . .
```

デフォルトのJACCプロバイダーのクラス名

デフォルトのJACCプロバイダーのクラス名は以下のとおりです。

区分	クラス名
Policy	jeus.security.impl.jacc.JACCPolicyWrapper
PolicyConfigurationFactory	jeus.security.impl.jacc.JACCPolicyConfigurationFactoryImpl
JACCPrincipalRoleMapper	jeus.security.impl.jacc.JACCDefaultPrincipalRoleMapper

上記のクラスは以下の場所にパッケージングされています。

```
JEUS_HOME/lib/system/jeus.jar
```


第7章 JAASの使用

本章では、JAASの概念を紹介し、JEUSセキュリティー・システムでSun ONE Directory ServerとJAASを連携する方法について簡単に説明します。

7.1. 概要

Java™認証および承認サービスのJAAS(Java Authentication and Authorization Service)は、Javaバージョンの標準PAM(Pluggable Authentication Module)を実装してユーザー・ベースの認証をサポートし、アクセス制御を認証および実行のための標準ベースのサービス・パッケージです。

JAASは、Java™ 2 SDK, Standard Edition(J2SDK) v 1.3のオプション・パッケージ(拡張機能)でしたが、J2SDK v 1.4以降に統合されました。

JAASの背景には、以下の2つの基本目的があります。

- ユーザーを**認証**します。

コードがアプリケーション、アプレット、Bean、またはサーブレットのいずれかに関係なく、Javaコードを実行しているユーザーを信頼するように、安全な方法で確認します。

- ユーザーを**承認**します。

動作の実行に必要なアクセス制御の権限(アクセス権)をユーザーが保持していることを確認します。

Java 2は、以前からも、コードソース・ベースのアクセス制御(コードの出所および署名者に基づくアクセス制御)を提供してきました。しかし、コードの実行者に基づくアクセス制御を追加実行する機能は不十分であったため、JAASがJava 2セキュリティー・アーキテクチャーを拡張してサポートするようにします。

JAAS認証は**プラグイン機能**の方法で実行されます。したがって、アプリケーションはベースとなる認証技術から独立して機能することになります。アプリケーション内では新規または更新された認証技術をプラグインとして使うことができ、アプリケーションを変更する必要はありません。

アプリケーションはLoginContextオブジェクトをインスタンス化することで、認証プロセスを有効にします。LoginContextオブジェクトはコンフィギュレーションを参照して使用する認証テクノロジー、もしくはLoginModuleを決めます。通常のLoginModuleはユーザー名およびパスワードの入力を要求して入力されたものを検証しますが、音声や指紋の解読、オブジェクトで検証を行うものもあります。

コードを実行するユーザーが認証されると、JAAS承認コンポーネントはコアJavaアクセス制御モデルと連携し、リソースへのアクセスを保護します。コードの場所およびコードの署名者(CodeSource)のみに基づいてアクセス制御の可否が決まるJ2SDK v 1.3と違って、J2SDK v1.4からは実行コードのコードソースおよびコードを実行するサブジェクト・オブジェクトで表せるユーザーまたはサービスに基づいてアクセス制御を行います。

す。認証に成功すると、ログイン・モジュールは関連するプリンシパルおよび資格証明を使ってサブジェクトを更新します。

参考

JAAS仕様の詳細については、<http://download.oracle.com/javase/6/docs/technotes/guides/security/jaas/tutorials/index.html>を参照してください。

基本的に、JAAS認証メカニズムが適用されたLDAPとデータベース、JEUSセキュリティー・システム間の連携のために、LoginModuleを拡張実装します。また、JEUSセキュリティー・システムではログインと承認に関連するサービスを拡張提供します。

本章では、JEUSセキュリティー・システムにJAAS関連のコア・クラスおよびインターフェースを適用する、LDAPとデータベースの連携例について説明します。

- LDAP LoginModuleの連携例
- DBRealm LoginModuleの連携例

7.2. JEUSとLDAPを連携するためのLoginModuleの実装

javax.security.auth.login.LoginContextインターフェース・クラスは被認証者の認証に使われる基本的なメソッドを提供するクラスです。このクラスを使用すれば、アプリケーションにログインする方法として、ベースとなる認証テクノロジーに頼らずに特定の認証タイプを提供するアプリケーションを開発することができます。

LDAPとの連携のために、LoginModuleを拡張してJAASの認証メカニズムをサポートします。LoginModuleインターフェースを継承したLdapLoginModuleを使って認証メカニズムを実行しています。

JEUSセキュリティー・システムとの連携を考慮してLdapLoginModuleを以下のように拡張実装します。

[例 7.1] <<jeus.security.impl.login.LdapLoginModule>>

```
package jeus.security.impl.login;

import jeus.security.base.Domain;
import jeus.security.resource.Password;
import jeus.security.resource.PrincipalImpl;
import jeus.security.resource.RolePrincipalImpl;
import jeus.util.logging.JeusLogger;

import javax.security.auth.Subject;
import javax.security.auth.callback.*;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import java.security.Principal;
import java.text.MessageFormat;
```

```

import java.util.ArrayList;
import java.util.Iterator;
import java.util.Map;

public class LdapLoginModule implements LoginModule {
    protected static final JeusLogger logger =
        (JeusLogger) JeusLogger.getLogger("jeus.security.loginmodule");

    // initial state
    private Subject subject;
    private CallbackHandler callbackHandler;

    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    private String username;
    private String password;
    private String domain;

    private Principal userPrincipal;
    private Password userCredential;

    private SunOneLdapAuthenticator authenticator;

    private static final String INITIAL_CONTEXT_FACTORY =
        "initialContextFactory"; // optional
    private static final String PROVIDER_URL = "providerURL";
    private static final String CONNECTION_USERNAME = "connectionUsername";
    private static final String CONNECTION_PASSWORD = "connectionPassword";
    private static final String USER_BASE = "userBase";
    private static final String USER_SEARCH_MAPPING = "userSearchMapping";
    private static final String USER_PASSWORD_ATTR = "userPasswordAttr";
    private static final String USER_ROLE_ATTR = "userRoleAttr";
    private static final String ROLE_BASE = "roleBase";
    private static final String ROLE_NAME_ATTR = "roleNameAttr";
    private static final String ROLE_SEARCH_MAPPING = "roleSearchMapping";

    private final String SUN_JDK_LDAP_CONTEXT_FACTORY =
        "com.sun.jndi.ldap.LdapCtxFactory";

    public void initialize(Subject subject, CallbackHandler callbackHandler,
        Map sharedState, Map options) {

        this.subject = subject;
        this.callbackHandler = callbackHandler;
        this.domain = Domain.SYSTEM_DOMAIN_NAME;
    }

```

```

        authenticator = new SunOneLdapAuthenticator();
        initAuthenticator(authenticator, options);
    }

    private void initAuthenticator(SunOneLdapAuthenticator authenticator,
                                   Map options) {
        // INITIAL_CONTEXT_FACTORY
        String value = (String) options.get(INITIAL_CONTEXT_FACTORY);
        if (value == null) {
            authenticator.setContextFactory(SUN_JDK_LDAP_CONTEXT_FACTORY);
        } else {
            authenticator.setContextFactory(value);
        }

        authenticator.setProviderUrl((String) options.get(PROVIDER_URL));
        authenticator.setConnectionUsername(
            (String) options.get(CONNECTION_USERNAME));
        authenticator.setConnectionPassword(
            (String) options.get(CONNECTION_PASSWORD));

        value = (String) options.get(USER_BASE);
        if (value != null) {
            authenticator.setUserBase(value);
        } else {
            String msg = "LoginMoulde initialization failed. " +
                "userBase option missing.";
            logger.warning(msg);
            throw new IllegalArgumentException(msg);
        }

        authenticator.setUserPasswordAttr((String) options.get(USER_PASSWORD_ATTR));

        value = (String) options.get(USER_SEARCH_MAPPING);
        if (value != null) {
            authenticator.setUserSearchMapping(new MessageFormat(value));
        } else {
            String msg = "LoginMoulde initialization failed. " +
                "userSearchMapping option missing.";
            logger.warning(msg);
            throw new IllegalArgumentException(msg);
        }

        authenticator.setUserRoleAttr((String) options.get(USER_ROLE_ATTR));

        value = (String) options.get(ROLE_BASE);
        if (value != null) {

```

```

        authenticator.setRoleBase(value);
    } else {
        String msg = "LoginMoulde initialization failed. " +
            "roleBase option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }

    value = (String) options.get(ROLE_NAME_ATTR);
    if (value != null) {
        authenticator.setRoleNameAttr(value);
    } else {
        String msg = "LoginMoulde initialization failed. " +
            "roleNameAttr option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }

    value = (String) options.get(ROLE_SEARCH_MAPPING);
    if (value != null) {
        authenticator.setRoleSearchMapping(new MessageFormat(value));
    } else {
        String msg = "LoginMoulde initialization failed. " +
            "roleSearchMapping option missing.";
        logger.warning(msg);
        throw new IllegalArgumentException(msg);
    }
}

public boolean commit() throws LoginException {
    if (succeeded == false) {
        return false;
    } else {
        userPrincipal = new PrincipalImpl(username);
        if (!subject.getPrincipals().contains(userPrincipal))
            subject.getPrincipals().add(userPrincipal);

        ArrayList roles = authenticator.getRoles();
        for (Iterator i = roles.iterator(); i.hasNext(); ) {
            String roleName = (String) i.next();
            logger.fine("Adding role to subject : username = " + username +
                ", roleName = " + roleName);
            subject.getPrincipals().add(new RolePrincipalImpl(roleName));
        }

        userCredential = new Password(password);
        subject.getPrivateCredentials().add(userCredential);
    }
}

```

```

        username = null;
        password = null;
        domain = null;
        commitSucceeded = true;
        return true;
    }
}

public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        succeeded = false;
        username = null;
        password = null;
        domain = null;
        userPrincipal = null;
        userCredential = null;
    } else {
        logout();
    }
    return true;
}

public boolean logout() throws LoginException {
    subject.getPrincipals().remove(userPrincipal);
    subject.getPrivateCredentials().remove(userCredential);
    succeeded = false;
    succeeded = commitSucceeded;
    username = null;
    password = null;
    domain = null;
    userPrincipal = null;
    userCredential = null;
    return true;
}

public boolean login() throws LoginException {
    Callback[] callbacks = null;

    // prompt for a user name and password
    if (callbackHandler == null)
        throw new LoginException("Error: no CallbackHandler available " +
            "to garner authentication information from the user");

    callbacks = new Callback[3];

```



```

callbacks[0] = new NameCallback("user name: ");
callbacks[1] = new PasswordCallback("password: ", false);
callbacks[2] = new TextInputCallback("domain: ");
try {
    callbackHandler.handle(callbacks);
    username = ((NameCallback) callbacks[0]).getName();
    char[] tmpPassword = ((PasswordCallback) callbacks[1]).getPassword();

    if (tmpPassword == null) {
        tmpPassword = new char[0];
    }
    password = new String(tmpPassword);
    ((PasswordCallback) callbacks[1]).clearPassword();
    domain = ((TextInputCallback) callbacks[2]).getText();

    if (!authenticator.authenticate(username, password)) {
        throw new LoginException("LDAP authentication failed.");
    }
    succeeded = true;
} catch (UnsupportedCallbackException uce) {
    uce.printStackTrace();
    LoginException le = new LoginException(
        "Error: " + uce.getCallback().toString() +
        " not available to garner authentication information " +
        "from the user");
    le.initCause(uce);
    throw le;
} catch (Exception e) {
    e.printStackTrace();    // todo. logging
    if (e instanceof LoginException) {
        throw (LoginException) e;
    } else {
        LoginException le = new LoginException(e.toString());
        le.initCause(e);
        throw le;
    }
}
return succeeded;
}
}

```

注

ランタイムに追加されるユーザーとロール情報がJEUSセキュリティー・システムに適用される必要がある
ので、RolePrincipalImpl情報をJEUSのRolePrincipalImplタイプに変換します。

7.3. LDAP JAAS LoginModuleサービスの設定

JEUSセキュリティ・システムで特定のドメインにJAAS LoginModuleを適用してログイン・サービスを提供するには、ドメインのセキュリティ・サービスを設定するとき、<jaas-login-config>を設定します。詳細については、「[2.3. セキュリティ・ドメイン構成要素の設定](#)」のlogin-service項目を参照してください。

DEFAULT_APPLICATION_DOMAINにLDAP LoginModuleのセキュリティ・サービスを提供するために、ログイン・サービスと承認サービスを以下のように設定します。

[例 7.2] ドメイン・サービスの設定 : <<domain.xml>>

```
<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
. . .
  <security-domain>
    <name>DEFAULT_APPLICATION_DOMAIN</name>
    <authentication>
      <jaas-login-config>
        <login-module>
          <login-module-classname>
            jeus.security.impl.login.LdapLoginModule
          </login-module-classname>
          <control-flag>required</control-flag>
          <option>
            <name>initialContextFactory</name>
            <value>com.sun.jndi.ldap.LdapCtxFactory</value>
          </option>
          <option>
            <name>providerURL</name>
            <value>ldap://192.168.1.63:389</value>
          </option>
          <option>
            <name>connectionUsername</name>
            <value>cn=Directory Manager</value>
          </option>
          <option>
            <name>connectionPassword</name>
            <value>adminadmin</value>
          </option>
          <option>
            <name>userBase</name>
            <value>ou=People,dc=sample,dc=com</value>
          </option>
          <option>
            <name>userSearchMapping</name>
            <value>(uid={0})</value>
          </option>
        </login-module>
      </jaas-login-config>
    </authentication>
  </security-domain>
</domain>
```

```

        <option>
            <name>roleBase</name>
            <value>ou=Groups,dc=sample,dc=com</value>
        </option>
        <option>
            <name>roleNameAttr</name>
            <value>cn</value>
        </option>
        <option>
            <name>roleSearchMapping</name>
            <value>(uniqueMember={0})</value>
        </option>
    </login-module>
</jaas-login-config>
</authentication>
<authorization>
    <repository-service>
        <custom-repository>
            <classname>
jeus.security.impl.aznrep.CustomPolicyFileRealmAuthorizationRepositoryService
            </classname>
            <property>
                <name>PolicyClassName</name>
                <value>jeus.security.base.CustomJeusPolicy</value>
            </property>
            <property>
                <name>UserPrincipalClassName</name>
                <value>jeus.security.resource.PrincipalImpl</value>
            </property>
            <property>
                <name>RolePrincipalClassName</name>
                <value>jeus.security.resource.RolePrincipalImpl</value>
            </property>
        </custom-repository>
    </repository-service>
</authorization>
<security-domain>
. . .
</domain>

```

以下は、各クラスについての説明です。

- `jeus.security.impl.callback.JAASUsernamePasswordCallbackHandler`

JEUSセキュリティー・システムでLoginModuleに認証情報((username, password, etc..))を取得する基本メカニズムを提供するCallbackHandlerクラスです。

- jeus.security.impl.login.LdapLoginModule

JEUSセキュリティー・システムでオプション値で定義されたLDAPの属性値に基づいて、LoginModuleをサポートするLoginModuleの実装クラスです。

- jeus.security.impl.aznrep.CustomPolicyFileRealmAuthorizationRepositoryService

JEUSセキュリティー・システムでユーザーが定義したUserPrincipalClassName/RolePrincipalClassNameタイプを適用して承認(Authorization)サービスを提供するクラスです。

- jeus.security.base.CustomJeusPolicy

JEUSセキュリティー・システムでランタイムにjeus-web-dd.xmlが存在せず、LoginModuleでプリンシパルに対するRolePrincipalImplを定義する必要がある場合、Principal-to-Roleマッピングをサポートするjeus.security.base.Policyを拡張実装したクラスです。

設定が完了すると、JEUSを起動して、DEFAULT_APPLICATION_DOMAINにデプロイされたアプリケーションへのログイン・サービスを開始します。

7.4. データベース連携のためのLoginModuleの実装

データベースとの連携のために、LoginModuleを拡張してJAASの認証メカニズムをサポートします。

LoginModuleインターフェースを継承したDBRealmLoginModuleを使って認証メカニズムを実行しています。JEUSのセキュリティー・システムとの連携を考慮して、以下のようにDBRealmLoginModuleを拡張実装します。

[例 7.3] <<jeus.security.impl.login.DBRealmLoginModule>>

```
package jeus.security.impl.login;

import jeus.security.base.Domain;
import jeus.security.base.ServiceException;
import jeus.security.resource.Password;
import jeus.security.resource.PrincipalImpl;
import jeus.security.resource.RolePrincipalImpl;
import jeus.util.logging.JeusLogger;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.security.auth.Subject;
import javax.security.auth.callback.*;
import javax.security.auth.login.FailedLoginException;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.sql.DataSource;
import java.security.Principal;
```

```

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

/**
 * User: choco
 * Date: 2007. 4. 13
 * Time: 4:35:57
 */
public class DBRealmLoginModule implements LoginModule {
    protected static final JeusLogger logger =
        (JeusLogger) JeusLogger.getLogger("jeus.security.login");
    protected String dsExportName;
    protected String principalsQuery =
        "select password from jeus_users where username=?";
    protected String rolesQuery = "select role from jeus_roles where username=?";

    private String username;
    private String password;
    private String domain;

    private Subject subject;
    private CallbackHandler callbackHandler;
    private boolean succeeded = false;
    private boolean commitSucceeded = false;

    protected Map options;
    private Principal userPrincipal;
    private Password userCredential;

    public void initialize(Subject subject,
        CallbackHandler callbackHandler, Map sharedState, Map options) {
        this.subject = subject;
        this.callbackHandler = callbackHandler;
        this.options = options;

        try {
            domain = Domain.getCurrentDomain().getName();
        } catch (ServiceException e) {
            domain = Domain.SYSTEM_DOMAIN_NAME;
        }
    }

```

```

dsExportName = (String) options.get("exportName");
if (dsExportName == null) {
    String msg = "LoginMoulde initialization failed. " +
        "exportName option missing.";
    logger.warning(msg);
    throw new IllegalArgumentException(msg);
}

Object tmp = options.get("principalsQuery");
if (tmp != null)
    principalsQuery = tmp.toString();
tmp = options.get("rolesQuery");
if (tmp != null)
    rolesQuery = tmp.toString();
logger.debug("DBRealmLoginModule, export name : " + dsExportName);
logger.debug("principalsQuery=" + principalsQuery);
logger.debug("rolesQuery=" + rolesQuery);
logger.debug("initialize successfully");
}

public boolean login() throws LoginException {
    Callback[] callbacks = null;

    // prompt for a user name and password
    if (callbackHandler == null)
        throw new LoginException("Error: no CallbackHandler available " +
            "to garner authentication information from the user");

    callbacks = new Callback[3];
    callbacks[0] = new NameCallback("user name: ");
    callbacks[1] = new PasswordCallback("password: ", false);
    callbacks[2] = new TextInputCallback("domain: ");
    try {
        callbackHandler.handle(callbacks);
        username = ((NameCallback) callbacks[0]).getName();
        char[] tmpPassword = ((PasswordCallback) callbacks[1]).getPassword();

        if (tmpPassword == null) {
            tmpPassword = new char[0];
        }
        password = new String(tmpPassword);
        ((PasswordCallback) callbacks[1]).clearPassword();
        domain = ((TextInputCallback) callbacks[2]).getText();

        String expectedPassword = getUsersPassword();
        if (validatePassword(password, expectedPassword) == false) {

```

```

        throw new LoginException("DBRealm authentication failed.");
    }
    succeeded = true;
} catch (UnsupportedCallbackException uce) {
    uce.printStackTrace();
    LoginException le = new LoginException(
        "Error: " + uce.getCallback().toString() +
        " not available to garner authentication information " +
        "from the user");
    le.initCause(uce);
    throw le;
} catch (Exception e) {
    e.printStackTrace();    // todo. logging
    if (e instanceof LoginException) {
        throw (LoginException) e;
    } else {
        LoginException le = new LoginException(e.toString());
        le.initCause(e);
        throw le;
    }
}
return succeeded;
}

private String getUsersPassword() throws LoginException {
    String password = null;
    Connection conn = null;
    PreparedStatement ps = null;
    ResultSet rs = null;

    try {
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup(dsExportName);
        conn = ds.getConnection();
        ps = conn.prepareStatement(principalsQuery);
        ps.setString(1, username);
        rs = ps.executeQuery();
        if (rs.next() == false)
            throw new FailedLoginException("No matching username found");

        password = rs.getString(1);
    }
    catch (NamingException ex) {
        throw new LoginException(ex.toString(true));
    }
    catch (SQLException ex) {
        logger.debug("Query failed", ex);
    }
}

```

```

        throw new LoginException(ex.toString());
    }
    finally {
        if (rs != null) {
            try {
                rs.close();
            }
            catch (SQLException e) {
            }
        }
        if (ps != null) {
            try {
                ps.close();
            }
            catch (SQLException e) {
            }
        }
        if (conn != null) {
            try {
                conn.close();
            }
            catch (SQLException ex) {
            }
        }
    }
    return password;
}

public boolean logout() throws LoginException {
    subject.getPrincipals().remove(userPrincipal);
    subject.getPrivateCredentials().remove(userCredential);
    succeeded = false;
    succeeded = commitSucceeded;
    username = null;
    password = null;
    domain = null;
    // trusted = false;
    userPrincipal = null;
    userCredential = null;
    return true;
}

public boolean commit() throws LoginException {
    if (succeeded == false) {
        return false;
    } else {
        userPrincipal = new PrincipalImpl(username);
    }
}

```



```

        if (!subject.getPrincipals().contains(userPrincipal))
            subject.getPrincipals().add(userPrincipal);

        ArrayList roles = getRoleSets();
        for (Iterator i = roles.iterator(); i.hasNext();) {
            String roleName = (String) i.next();
            logger.debug("Adding role to subject : username = " + username +
                ", roleName = " + roleName);
            subject.getPrincipals().add(new RolePrincipalImpl(roleName));
        }

        userCredential = new Password(password);
        subject.getPrivateCredentials().add(userCredential);

        username = null;
        password = null;
        domain = null;
        commitSucceeded = true;
        return true;
    }
}

public boolean abort() throws LoginException {
    if (succeeded == false) {
        return false;
    } else if (succeeded == true && commitSucceeded == false) {
        succeeded = false;
        username = null;
        password = null;
        domain = null;
        userPrincipal = null;
        userCredential = null;
    } else {
        logout();
    }
    return true;
}

protected ArrayList getRoleSets() throws LoginException {
    Connection conn = null;
    HashMap setsMap = new HashMap();
    PreparedStatement ps = null;
    ResultSet rs = null;
    ArrayList roles = new ArrayList();
    try {
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup(dsExportName);
    }
}

```

```

        conn = ds.getConnection();
        ps = conn.prepareStatement(rolesQuery);
        try {
            ps.setString(1, username);
        }
        catch (ArrayIndexOutOfBoundsException ignore) {
        }
        rs = ps.executeQuery();

        while (rs.next()) {
            String rolename = rs.getString(1);
            roles.add(rolename);
        }
    }
    catch (NamingException ex) {
        throw new LoginException(ex.toString(true));
    }
    catch (SQLException ex) {
        logger.debug("SQL failure", ex);
        throw new LoginException(ex.toString());
    }
    finally {
        if (rs != null) {
            try {
                rs.close();
            }
            catch (SQLException e) {
            }
        }
        if (ps != null) {
            try {
                ps.close();
            }
            catch (SQLException e) {
            }
        }
        if (conn != null) {
            try {
                conn.close();
            }
            catch (Exception ex) {
            }
        }
    }

    return roles;
}

```

```

        protected boolean validatePassword(String inputPassword, String
expectedPassword) {
            if (inputPassword == null || expectedPassword == null)
                return false;
            return inputPassword.equals(expectedPassword);
        }
    }
}

```

注

ランタイムに追加されるユーザーとロールの情報がJEUSセキュリティ・システムに適用される必要があるので、RolePrincipalImpl情報をJEUSのRolePrincipalImplタイプに変換します。

7.5. データベースのLoginModuleサービスの設定

JEUSセキュリティ・システムで特定のドメインにJAAS LoginModuleを適用してログイン・サービスを提供するには、ドメイン・セキュリティ・サービスを<jaas-login-config>に設定します。詳細については、「[2.3. セキュリティ・ドメイン構成要素の設定](#)」の「**authentication**」項目を参照してください。

DEFAULT_APPLICATION_DOMAINにデータベースのLoginModuleセキュリティ・サービスを提供するためには、ログイン・サービスと承認サービスを以下のように設定します。

[例 7.4] ドメイン・サービスの設定 : <<domain.xml>>

```

<?xml version="1.0" encoding="UTF-8"?>
<domain xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    . . .
    <security-domain>
        <name>DEFAULT_APPLICATION_DOMAIN</name>
        <authentication>
            <jaas-login-config>
                <login-module>
                    <login-module-classname>
                        jeus.security.impl.login.DBRealmLoginModule
                    </login-module-classname>
                    <control-flag>required</control-flag>
                    <option>
                        <name>exportName</name>
                        <value>dbrealmtest</value>
                    </option>
                    <option>
                        <name>principalsQuery</name>
                        <value>
                            select password from

```

```

DEFAULT_APPLICATION_DOMAIN_Principals
    where username=?
    </value>
</option>
<option>
    <name>rolesQuery</name>
    <value>
        select role from DEFAULT_APPLICATION_DOMAIN_roles where
username=?
    </value>
</option>
</login-module>
</jaas-login-config>
</authentication>
<authorization>
    <repository-service>
        <custom-repository>
            <classname>
                jeus.security.impl.aznrep.
CustomPolicyFileRealmAuthorizationRepositoryService
            </classname>
            <property>
                <name>PolicyClassName</name>
                <value>jeus.security.base.CustomJeusPolicy</value>
            </property>
            <property>
                <name>UserPrincipalClassName</name>
                <value>jeus.security.resource.PrincipalImpl</value>
            </property>
            <property>
                <name>RolePrincipalClassName</name>
                <value>jeus.security.resource.RolePrincipalImpl</value>
            </property>
        </custom-repository>
    </repository-service>
</authorization>
</security-domain>
. . .
</domain>

```

以下は、各クラスについての説明です。

- jeus.security.impl.login.DBRealmLoginModule

JEUSセキュリティー・システムでオプション値で定義されたデータベースのエクスポート名 (exportname)、プリンシパル (principal)、ロール (role) に対するクエリー値に基づいて、LoginModuleをサポートする LoginModuleの実装クラスです。

以下は、オプション項目についての説明です。

項目	説明
exportName	domain.xmlに定義されたデータベースのexport-nameを設定します
principalsQuery	主キーが1つで、プリンシパルのパスワード値を取得できるクエリーを定義します
rolesQuery	主キーが1つで、プリンシパルのロール値を取得できるクエリーを定義します

- jeus.security.impl.aznrep.CustomPolicyFileRealmAuthorizationRepositoryService

JEUSセキュリティー・システムでユーザーが定義したUserPrincipalClassName/RolePrincipalClassNameタイプを適用して承認(Authorization)サービスを提供するクラスです。

- jeus.security.base.CustomJeusPolicy

JEUSセキュリティー・システムでランタイムにjeus-web-dd.xmlが存在せず、LoginModuleでプリンシパルに対するRolePrincipalImplを定義する必要がある場合、Principal-to-Roleマッピングをサポートするjeus.security.base.Policyを拡張実装したクラスです。

参考

設定が完了すると、JEUSを起動して、DEFAULT_APPLICATION_DOMAINにデプロイされたアプリケーションへのログイン・サービスを開始します。

付録 A. セキュリティー・イベント・サービス

本付録では、セキュリティー・イベント・サービスについて説明します。

A.1. 概要

SPIクラスとデフォルトのセキュリティー・サービスの実装クラスからEventHandlingServiceに送信される標準のセキュリティー・イベントについて紹介します。jeus.security.spi.EventHandlingService SPIを実装して独自のイベント・ハンドリングを開発する際にこの章の内容を参照してください。

リストの様式は以下のとおりです。

```
G.2.X <Event type> = イベントのタイプ
    Source Class: イベントが発生したクラス
    Event Type: イベントのタイプ
    Event Level: イベントのレベル (FATAL、SERIOUS、WARNING、INFORMATION、DEBUG)
    Event Context: イベント・コンテキストに対するkey-value組の集合
    Emitted When? イベントが発生する条件
```

通常、イベントはイベントソースと同じドメインにあるEventHandlingServiceに送信されます。ただし、security.install.successfulイベントとsecurity.uninstall.attemptイベントは例外的にセキュリティー・システムに設定されているすべてのドメインに送信されます。

参考

jeus.security.base.Eventクラスとjeus.security.spi.EventHandlingServiceクラスの詳細については、Javadocを参照してください。

A.2. イベント

以下は、標準セキュリティー・イベントのリストです。

security.validation.failed

Source Class	jeus.security.spi.SubjectValidationService
Event Type	security.validation.failed
Event Level	WARNING

Event Context	<ul style="list-style-type: none"> – Key: “subject” – Value: 有効性検証に失敗したjeus.security.base.Subject
Emitted When	SubjectValidationServiceがSecurityExceptionを発生させるたびに

security.authentication.failed

Source Class	jeus.security.spi.AuthenticationService
Event Type	security.authentication.failed
Event Level	WARNING
Event Context	<ul style="list-style-type: none"> – Key: “subject” – Value: 有効性検証に失敗したjeus.security.base.Subject
Emitted When	サブジェクトへのユーザー認証が失敗するたびに

security.authorization.failed

Source Class	jeus.security.spi.AuthorizationService
Event Type	security.authentication.failed
Event Level	WARNING
Event Context	<ul style="list-style-type: none"> – Key: “contextid” Value: パーミッション・チェックが行われたコンテキストID – Key: “permission” Value: チェックされるjava.security.Permission – Key: “subject” Value: ユーザー認証に失敗したjeus.security.base.Subject
Emitted When	ユーザー認証に失敗するたびに

security.authentication.repository.subject.added

Source Class	jeus.security.spi.AuthenticationRepositoryService
---------------------	---

Event Type	security.authentication.repository.subject.added
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> – Key: “subject” – Value: 追加されたjeus.security.base.Subject
Emitted When	AuthenticationRepositoryServiceへのサブジェクトの追加に成功するたびに

security.authentication.repository.subject.removed

Source Class	jeus.security.spi.AuthenticationRepositoryService
Event Type	security.authentication.repository.subject.removed
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> – Key: “subject” – Value: 削除されたjeus.security.base.Subject
Emitted When	AuthenticationRepositoryServiceからサブジェクトの削除に成功するたびに

security.authentication.repository.subject.removed.complete

Source Class	jeus.security.spi.AuthenticationRepositoryService
Event Type	security.authentication.repository.subject.removed.complete
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> – Key: “name” – Value: 削除に成功したサブジェクト
Emitted When	AuthenticationRepositoryServiceからサブジェクトの削除に成功するたびに

security.authorization.repository.policy.added

Source Class	jeus.security.spi.AuthorizationRepositoryService
Event Type	security.authorization.repository.policy.added
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> – Key: “policy”

	– Value: 追加されたjeus.security.base.Policy
Emitted When	AuthorizationRepositoryServiceに新規のポリシーが追加されるたびに

security.authorization.repository.policy.removed

Source Class	jeus.security.spi.AuthorizationRepositoryService
Event Type	security.authorization.repository.policy.removed
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> – Key: “policy” – Value: 削除されたjeus.security.base.Policy
Emitted When	AuthorizationRepositoryServiceからポリシー・データが削除されるたびに

security.authorization.repository.policy.removed.complete

Source Class	jeus.security.spi.AuthorizationRepositoryService
Event Type	security.authorization.repository.policy.removed.complete
Event Level	INFORMATION
Event Context	<ul style="list-style-type: none"> – Key: “contextid” – Value: リポジトリから削除されたjava.lang.StringタイプのコンテキストID
Emitted When	AuthorizationRepositoryServiceからコンテキストIDが削除されるたびに

security.install.successful

Source Class	jeus.security.spi.SecurityInstaller
Event Type	security.install.successful
Event Level	INFORMATION
Event Context	None
Emitted When	セキュリティー・システムのインストールに成功した後

security.uninstall.attempt

Source Class	jeus.security.spi.SecurityInstaller
Event Type	security.uninstall.attempt
Event Level	INFORMATION
Event Context	None
Emitted When	セキュリティー・システムが除去される前

付録 B. JEUSサーバーのパーミッション

本付録では、標準パーミッションのリソース名とリソース・アクションについて説明します。

B.1. 概要

標準パーミッションのリソース名とリソース・アクションは、JEUSサーバー・モジュール(JNDI、JMS、マネージャー、セキュリティなど)がリソースへのアクセス権限を持っているかチェックするために使われます。リソースのパーミッション・チェックに関連するパーミッション・タイプはjeus.security.resource.ResourcePermissionであり、コンテキストIDは常に「default」です。

権限チェックは常にリソースの名前とアクションの組み合わせにより発生します。リソースの名前はアクセスするターゲットを示し、リソースのアクションはターゲットに対してどんな動作を行うかを示します。権限設定が正しく行われていない場合には、DASあるいはサーバー・ログを確認して定義されているパーミッションを確認して追加します。

B.2. JEUSシステム・リソース名

以下では、JEUSのデフォルト・セキュリティ・システムで提供する重要なリソースの名前を簡単に紹介します。

リソース名	説明
jeus.*	JEUSシステムのすべてのリソース名にアクセスできます
jeus.server.<server-name>.*	JEUSシステムの特定のサーバーのすべてのリソース名にアクセスできます。デフォルトのセキュリティ・システムを使用する場合に、サーバー側でチェックするパーミッションのリストです。 リソース・アクションによって以下のように分けられます – boot : サーバーを起動するとき – down : サーバーを終了するとき – deploy : サーバーにアプリケーションをデプロイするとき – ftp : ファイルの転送にFTPを使用するとき
jeus.server.<server-name>app.<application-name>	JEUSシステムの特定のサーバーの特定のアプリケーションのリソース名にアクセスできます

リソース名	説明
jeus.cluster.<cluster-name>.*	JEUSシステムの特定のクラスターのすべてのリソース名にアクセスできます
jeus.domain.<domain-name>	<p>JEUSシステムの動的な設定変更権限のリソース名です。ここでドメインはセキュリティー・ドメインではなく、JEUSシステム・ドメインを示します。</p> <p>リソース・アクションによって以下のように分けられます</p> <ul style="list-style-type: none"> – dynamicConfiguration : WebAdminまたはjeusadminを使ってドメインの設定を動的に変更するとき
jeus.jndi	<p>JEUSシステムのJNDI動作権限のリソース名です。</p> <p>リソース・アクションによって以下のように分けられます</p> <ul style="list-style-type: none"> – lookup : JNDIを利用してオブジェクトをルックアップする – modify : bind/unbind/renameでJNDIリポジトリ・オブジェクトを追加、削除、変更する – list : JNDIリポジトリに保存されているオブジェクト・リストを取得する
jeus.node.<node-name>	<p>JEUSシステムの特定のノードのリソース名です。ノードを追加・削除するときや、ノードにJEUSをインストールまたはアンインストールするときに使用します。ノードについては、『JEUS サーバガイド』または『JEUS ノードマネージャガイド』を参照してください。</p> <p>リソース・アクションによって以下のように分けられます</p> <ul style="list-style-type: none"> – edit : ノードを追加または削除する – install : ノードにJEUSをインストールする – uninstall : ノードでJEUSをアンインストールする

B.3. jeusadminコマンドの権限設定

デフォルトのセキュリティー・システムを基準に、jeusadminのコマンド単位で権限を設定することができます。

jeusadminのコマンド単位で権限を設定した場合、該当するコマンドに対する権限確認だけ行われ、その他の内部権限に対してはチェックせずにスキップします。

コマンド権限のリソース名はコマンドのオプションと関連付けられますが、server、servers、cluster、clusters、nodeオプションに対して前述したリソース名を保持します。その他のすべてのオプションに対しては、jeus.domain.<domain-name>リソース名を保持します。

コマンド権限のリソース・アクションはコマンドの名前で定義されます。コマンドのエイリアスに対しては提供されないため、help <command-name>で確認できる実際のコマンド名を入力してください。コマンド情報については、『JEUS リファレンスガイド』のPart II, 「コンソール・コマンドとツール」を参照してください。

以下は、「user1」に管理者権限を、「user2」には「server2」にdeployコマンドを実行できる権限を与える例です。「user2」には「server2」に対する権限のみを与えるため、リソース名にjeus.server.server2.*を入力し、リソース・アクションにはjeusadminでdeployコマンドの実際の名前であるdeploy-applicationを入力します。

[例 B.1] セキュリティー・システムのポリシー設定 : <<policies.xml>>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<policies xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <role-permissions>
      <role-permission>
        <principal>user1</principal>
        <role>adminRole</role>
      </role-permission>
      <role-permission>
        <principal>user2</principal>
        <role>server2DeployRole</role>
      </role-permission>
    </role-permissions>
    <resource-permissions>
      <context-id>default</context-id>
      <resource-permission>
        <role>adminRole</role>
        <resource>jeus.*</resource>
        <actions>*</actions>
      </resource-permission>
      <resource-permission>
        <role>server2DeployRole</role>
        <resource>jeus.server.server2.*</resource>
        <actions>deploy-application</actions>
      </resource-permission>
    </resource-permissions>
  </policy>
</policies>
```


索引

シンボル

- <application>
 - <role-permission>, 98
- <assembly-description>
 - <security-role>, 85
- <cert-user>
 - <alias>, 74
 - <keypassword>, 74
 - <secretkey>, 75
 - <username>, 74
- <cert-user>>
 - <cert>, 73
 - <user>, 73
- <group>, 53
 - <description>, 53
 - <name>, 53
 - <subgroup>, 53
- <method-permission>
 - <exclude-list>, 86
 - <method-intf>, 86
 - <method-name>, 86
 - <method-params>, 86
 - <method>, 86
 - <role-name>, 85
 - <unchecked>, 85
- <method>
 - <ejb-name>, 86
- <module-info>
 - <role-permission>, 88
- <password>
 - AES/DES/DESede/SEED/Blowfish, 58
 - base64, 58
 - SHA, 58
- <PasswordValidator>
 - <CustomPasswordValidator>, 56
 - <DefaultPasswordValidator>, 55

- <policy>
 - <resource-permissions>, 65
 - <role-permissions>, 64
- <role-mapping>
 - <role-permission>, 94
- <role-permission>
 - <actions>, 88, 94, 98
 - <classname>, 88, 94, 98
 - <excluded>, 88, 94, 99
 - <principal>, 88, 94, 98
 - <role>, 88, 94, 98
 - <unchecked>, 88, 94, 99
- <run-as>
 - <role-name>, 90
- <security-constraint>, 90
 - <auth-constraint>, 91
 - <user-data-constraint>, 91
 - <web-resource-collection>, 90
- <security-domain>
 - <audit>, 48
 - <authentication>, 45
 - <authorization>, 46
 - <cache-config>, 29
 - <connect-retries>, 23
 - <credential-mapping>, 47
 - <credential-verification>, 48
 - <default-application-domain>, 23
 - <identity-assertion>, 47
 - <keystore-config>, 29
 - <name>, 29, 45
 - <security-domains>, 23
- <security-identity>
 - <role-name>, 85
 - <run-as>, 85
- <security-role-ref>, 90
 - <role-link>, 85, 90
 - <role-name>, 85, 90
- <security-role>, 91
- <transport-guarantee>設定値
 - CONFIDENTIAL, 91
 - INTEGRAL, 91
 - NONE, 91
- <unspecified-method-permission>, 88

- <excluded>, 89
- <role>, 89
- <unchecked>, 89
- <user>, 52
 - <description>, 53
 - <group>, 53
 - <name>, 53
 - <password>, 53
- <web-resource-collection>
 - <http-method>, 91
 - <url-patterns>, 90
 - <web-resource-name>, 90

A

- accounts.xml, 20
- AuthenticationRepositoryService, 113, 115
- AuthenticationService, 113, 115
- AuthorizaionService, 152
- AuthorizationRepositoryService, 113, 118
- AuthorizationService, 113, 117

C

- Checked permission, 9
- Credential Factory, 105
- CredentialFactory, 6, 105
- CredentialMappingService, 113, 116
- CredentialVerificationService, 113, 116

E

- EventHandlingService, 113, 114, 151
- Excluded permission, 9
- ExpiryTime, 105

I

- IdentityAssertionService, 113, 116

J

- JACC Provider
 - java.security.Policy, 125
 - javax.security.jacc.PolicyConfiguration, 125
 - javax.security.jacc.PolicyConfigurationFactory, 125
- java.security.manager, 72

- java.security.policy, 72
- jeus-security.xsd, 28
- jeus.security.admin, 4
- jeus.security.admin, jeus.security.container, 4
- jeus.security.base, 4
- jeus.security.baseパッケージ・クラス
 - jeus.security.base.CredentialFactory, 104
 - jeus.security.base.PermissionMap, 104
 - jeus.security.base.Policy, 104
 - jeus.security.base.Role, 104
 - jeus.security.base.SecurityCommonService, 104
 - jeus.security.base.SecurityException, 104
 - jeus.security.base.ServiceException, 104
 - jeus.security.base.Subject, 104
- jeus.security.impl.*, 4
- jeus.security.resourceパッケージ・クラス
 - jeus.security.resource.ExpiryTime, 105
 - jeus.security.resource.ExpiryTimeFactory, 105
 - jeus.security.resource.GroupPrincipalImpl, 105
 - jeus.security.resource.Lock, 105
 - jeus.security.resource.LockFactory, 105
 - jeus.security.resource.Password, 105
 - jeus.security.resource.PasswordFactory, 105
 - jeus.security.resource.PrincipalImpl, 105
 - jeus.security.resource.ResourcePermission, 105
 - jeus.security.resource.RoleImpl, 105
 - jeus.security.resource.RolePermission, 105
 - jeus.security.resource.TimeConstrainedRolePermission, 105
- jeus.security.spi, 3

N

- Non-Blocking I/O, 16

P

- Password Credential, 105
- PermissionMap, 9, 10, 11
 - Checked Permissions, 11
 - Excluded Permissions, 11
 - Unchecked Permissions, 11
- policies.xml, 20, 60
- policy, 20

Principal-to-Role, 80
PrincipallImpl, 105, 106
Proof Credential, 116
PropertyHolder, 49

R

Role-to-Resource, 77
RoleImpl, 106
run-as identity, 89

S

Secured Connection, 16, 17
SecuritException, 117
security constraint, 77
Security Installer, 19
security.key, 20
SecurityException, 114, 117, 152
SecurityInstaller, 14, 113, 154, 155
SecurityManager, 16, 71
SHARED_DOMAIN, 15
SPI, 13
Subject, 5
SubjectFactoryService, 113, 114
SubjectValidationService, 113, 114, 152
SYSTEM_DOMAIN, 15, 20

T

TimeConstrainedRolePermission, 63, 69, 95
TimeConstraintRolePermission, 9

U

Unchecked permission, 9

か

権限チェック, 7
カスタム・パーミッション, 66

さ

資格証明, 6
サービス, 13
サービス・クラス

Service, 110
Service Description, 110
Service Domain, 110
Service MBean, 111
Service Properties, 111
Service State, 111
Service Type, 110

セキュリティー・ドメイン, 14
セキュリティー制約, 77
セキュリティー監査, 12

た

デフォルトJACCプロバイダー, 129
デフォルト・セキュリティー, 60

な

認証, 5

は

パーミッション, 7
プリンシパル, 80

ら

リポジトリと外部セキュリティー・メカニズム, 4
ログイン, 5
ロール・ベースのメカニズム, 7

