Red Hat JBoss Enterprise Application Platform 6.4

セキュリティーガイド

非推奨。本ガイドでは、EAP サーバーインスタンスを強化し、Web アプリケーションをセキュア化するためのセキュリティー概念および手順について説明します。

Last Updated: 2023-02-21

Red Hat JBoss Enterprise Application Platform 6.4 セキュリティーガイド

非推奨。本ガイドでは、EAP サーバーインスタンスを強化し、Web アプリケーションをセキュア化するためのセキュリティー概念および手順について説明します。

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

非推奨。本ガイドでは、EAP サーバーインスタンスを強化し、Web アプリケーションをセキュア化するためのセキュリティー概念および手順について説明します。

目次

はじめに

1. ドキュメント規則

本ガイドでは、いくつかの規則を使用して特定の単語やフレーズを強調表示し、特定の情報への注意を促しています。

1.1. 誤字規則

特定の単語や句に注意を促すために 4 つの誤字規則を使用しています。これらの規則や、これらが適用される状況は以下のとおりです。

Mono-spaced Bold

シェルコマンド、ファイル名、パスなど、システム入力を強調表示するために使用されます。キーとキーの組み合わせを強調表示するためにも使用されます。以下に例を示します。

> 現在の作業ディレクトリーのファイル **my_next_bestselling_novel** の内容を表示するには、シェルプロンプトで **cat my_next_bestselling_novel** コマンドを入力し、**Enter** を押してコマンドを実行します。

上記には、ファイル名、シェルコマンドおよびキーが含まれます。これはすべて mono-spaced bold で示され、コンテキストにより区別可能なものになります。

キーの組み合わせは、キーの組み合わせの各パーツをつなげるプラス記号によって個別のキーと区別できます。以下に例を示します。

> **Enter** を押してコマンドを実行します。
>
> **Ctrl**+**Alt**+**F2** を押して、仮想端末に切り替えます。

最初の例では、押す特定のキーを強調表示しています。2 つ目の例は、同時に押す 3 つのキーのセットというキーの組み合わせを強調表示しています。

ソースコードについて記述では、クラス名、メソッド、関数、変数名、および段落内で記述された戻り値は、**mono-spaced bold**で示されます。以下に例を示します。

> ファイル関連のクラスには、**ファイル** システムのファイルシステム、**ファイルのファイル**、ディレクトリー用の **dir** が含まれます。各クラスには、独自の関連付けられたパーミッションセットがあります。

Proportional Bold

これは、アプリケーション名、ダイアログボックステキスト、ラベルが付いたボタン、チェックボックスおよびラジオボタン、メニュータイトルおよびサブメニュータイトルなど、システムで発生した単語またはフレーズを示します。以下に例を示します。

> メインメニューバーから System → Preferences → Mouse を選択して、Mouse Preferences を起動します。**Buttons** タブで、**Left-handed mouse** チェックボックスを選択し、**Close** をクリックして、左から右に主要なマウスボタンを切り替えます (左側のマウスは適切なマウスになります)。
>
> 特殊文字を gedit ファイルに挿入するには、メインメニューバーから Applications → Accessories → Character Map を選択します。次に、Character Map メニューバーから Search → Find... を選択し、**Search** フィールドに文字の名前を入力して **Next** をクリックします。目的の文字は **Character Table** に強調表示されます。この強調表示し

た文字をダブルクリックして、**Text to copy** フィールドに配置し、**Copy** ボタンをクリックします。次に、ドキュメントに戻り、**gedit** メニューバーから **Edit → Paste** を選択します。

上記のテキストにはアプリケーション名、システム全体のメニュー名および項目、アプリケーション固有のメニュー名、GUI インターフェース内のボタンおよびテキストなどがあります。すべては proportional bold で示され、コンテキストと区別できます。

Mono-spaced Bold Italic または *Proportional Bold Italic*

mono-spaced bold または proportional bold のいずれでも、イタリックを追加すると、置換または変数テキストが表示されます。イタリックは、状況に応じて変化するテキストや、文字を入力しないテキストを表します。以下に例を示します。

ssh を使用してリモートマシンに接続するには、シェルプロンプトで **ssh *username@domain.name*** と入力します。リモートマシンが **example.com** で、そのマシンのユーザー名が john の場合は、**ssh john@example.com** と入力します。

mount -o remount *file-system* コマンドは、名前付きのファイルシステムを再マウントします。たとえば、**/home** ファイルシステムを再マウントする場合、コマンドは **mount -o remount /home** になります。

現在インストールされているパッケージのバージョンを表示するには、**rpm -q *package*** コマンドを使用します。その結果が ***package-version-release*** と以下のように返されます。

上記の太字のイタリック体の用語、username、domain.name、file-system、package、version、および release に注意してください。各単語はプレースホルダーで、コマンドの発行時に入力するテキストまたはシステムによって表示されるテキストのどちらかになります。

作業のタイトルを示す標準的な使用法のほかに、イタリックは新用語と重要な用語の最初の使用を示します。以下に例を示します。

Publican は *DocBook* 公開システムです。

1.2. 引用規則

端末の出力およびソースコードの一覧は、周りのテキストから視覚的に表示されます。

端末に送信される出力は、**mono-spaced roman** に設定されるため、次のように表示されます。

```
books       Desktop   documentation  drafts  mss    photos   stuff  svn
books_tests  Desktop1  downloads      images  notes  scripts  svgs
```

ソースコードの一覧も **mono-spaced roman** に設定されますが、構文強調表示を以下のように追加します。

```
static int kvm_vm_ioctl_deassign_device(struct kvm *kvm,
                 struct kvm_assigned_pci_dev *assigned_dev)
{
        int r = 0;
        struct kvm_assigned_dev_kernel *match;

        mutex_lock(&kvm->lock);

        match = kvm_find_assigned_dev(&kvm->arch.assigned_dev_head,
```

```
                                assigned_dev->assigned_dev_id);
        if (!match) {
                printk(KERN_INFO "%s: device hasn't been assigned before, "
                  "so cannot be deassigned\n", __func__);
                r = -EINVAL;
                goto out;
        }

        kvm_deassign_device(kvm, match);

        kvm_free_assigned_device(kvm, match);

out:
        mutex_unlock(&kvm->lock);
        return r;
}
```

1.3. 注記および警告

最後に、3 つの視覚的スタイルを使用して、見落とす可能性のある情報に注意を促します。

注記

注記とは、タスクへのヒント、ショートカット、または代替アプローチです。注意を無視しても悪い結果を招くことはありませんが、便利なヒントを見逃してしまう可能性があります。

重要

見落としやすい詳細のある重要なボックス: 現行セッションにのみ適用される設定変更や、更新を適用する前に再起動が必要なサービスなどです。「Important」というラベルが付いたボックスを無視しても、データが失われることはありませんが、スムーズな操作が行えないことがあります。

警告

警告は無視すべきではありません。警告を無視すると、データが失われる可能性があります。

2. ヘルプの利用とフィードバック提供

2.1. ヘルプが必要ですか?

本ガイドで説明されている手順で問題が発生した場合は、Red Hat カスタマーポータル http://access.redhat.com にアクセスしてください。カスタマーポータルでは、以下を行うことができます。

- Red Hat 製品に関する技術サポート記事の知識ベースの検索または閲覧。

- Red Hat グローバルサポートサービス (GSS) へのサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

Red Hat は、Red Hat のソフトウェアおよびテクノロジーについて、多くの電子メーリングリストも提供しています。一般に公開されているメーリングリストの一覧は、https://www.redhat.com/mailman/listinfoを参照してください。メーリングリストの名前をクリックして、その一覧をサブスクライブするか、またはメーリングリストのアーカイブにアクセスします。

2.2. ご意見をお寄せください

本ガイドで誤字脱字を発見されたり、このマニュアルを改善するための提案をお持ちの場合は、弊社までご連絡ください。Red Hat JBoss Enterprise Application Platform の製品に対して、http://bugzilla.redhat.com/ の Bugzilla レポートを送信してください。

バグレポートを送信する際には、マニュアル識別子 『Security_Guide』を指定してください。

本ガイドを改善するためのご意見やご提案をお寄せいただく場合は、できるだけ具体的にご説明ください。エラーが見つかった場合は、簡単に確認できるように、セクション番号と前後のテキストを含めてください。

パート I. SECURITY FOR RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

第1章 はじめに

1.1. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 6

Red Hat JBoss Enterprise Application Platform 6(JBoss EAP 6)は、オープン標準に構築されたミドルウェアプラットフォームで、Java Enterprise Edition 6 仕様に準拠します。JBoss Application Server 7 と高可用性クラスタリング、メッセージング、分散キャッシングなどのテクノロジーを統合します。

JBoss EAP 6 には、必要な場合にだけサービスを有効にできる新しいモジュール構造が含まれます (サービスの起動時間が短縮されます)。

管理コンソールと管理コマンドラインインターフェースにより、XML 設定ファイルの編集が不必要になり、タスクをスクリプト化および自動化する機能が追加されました。

また、JBoss EAP 6 には、セキュアでスケーラブルな Java EE アプリケーションの迅速な開発を可能にする API と開発フレームワークが含まれます。

バグの報告

1.2. JBOSS EAP 6 のセキュリティー保護

コンピューターセキュリティーは、デジタル年を駆動する仮想環境のセキュリティーを保護する情報技術のフィールドに記載されているすべての用語です。これには、データ保護と整合性、アプリケーションセキュリティー、リスク、脆弱性評価、認証および認可プロトコルが含まれます。

コンピューターデータは、ほとんどの組織における重要なアセットです。データ保護は重要なものであり、ほとんどの企業の中核を形成します。JBoss EAP 6 は、すべての段階でデータを処理するためのセキュリティーに対する多層アプローチを提供します。

本当の安全性は、セキュリティーを主な機能を果たすことで、セキュリティーから設計されているシステムです。このようなシステムは、設計上、セキュリティーの原則を使用します。このようなシステムでは、悪意のある攻撃や侵入が、通常のセキュリティーアプリやアルファベーションの一部として受け入れられ、システムがそれを回避するように設計されています。

セキュリティーは、オペレーティングシステム、ミドルウェア、およびアプリケーションレベルで適用できます。RHEL に適用されるオペレーティングシステムレベルでのセキュリティーに関する詳細は、『Red Hat Enterprise Linux セキュリティーガイド』を参照してください。

後の章では、JBoss EAP 6 のさまざまなレベルのセキュリティーレベルとレイヤーについて説明します。これらのレイヤーは、プラットフォーム内のすべてのセキュリティー機能にインフラストラクチャーを提供します。

バグの報告

パート II. プラットフォームのセキュリティー保護

第2章 JAVA SECURITY MANAGER

2.1. JAVA SECURITY MANAGER について

Java Security Manager は、Java Virtual Machine (JVM) サンドボックスの外部境界を管理するクラスで、JVM 内で実行されるコードが JVM 外のリソースと対話する方法を制御します。Java Security Manager を有効にすると、Java API はさまざまな潜在的に危険な操作を実行する前に、セキュリティーマネージャーで承認を確認します。Java Security Manager はセキュリティーポリシーを使用して、特定のアクションが許可または拒否されるかどうかを判断します。

バグの報告

2.2. JAVA セキュリティーポリシー

Java Security ポリシーは、さまざまなクラスのコードに対する定義されたパーミッションのセットです。Java Security Manager は、アプリケーションによって要求されたアクションをセキュリティーポリシーと比較します。ポリシーがアクションを許可した場合、Security Manager はそのアクションの実行を許可します。ポリシーによりアクションが許可されない場合、セキュリティーマネージャーはそのアクションを拒否します。セキュリティーポリシーは、コードの場所、コードの署名、またはサブジェクトのプリンシパルに基づいてパーミッションを定義できます。

Java Development Kit(JDK)に含まれる **policytool** アプリケーションを使用してセキュリティーポリシーを作成できます。セキュリティーポリシーエントリーは、**policytool** を使用して設定可能な以下の設定要素で構成されます。

CodeBase

コードの起点となる URL の場所（ホストとドメイン情報を除く）。このパラメーターは任意です。

SignedBy

コードの署名に秘密鍵が使用された署名者を参照するキーストアで使用されるエイリアス。これは、単一の値またはコンマ区切りの値のリストになります。このパラメーターは任意です。省略した場合には、署名の有無に関わらず、Java Security Manager には影響がありません。

Principal

実行中のスレッドのプリンシパルセットに存在する必要がある ***principal_type*/*principal_name*** ペアの一覧。Principals エントリーは任意です。これを省略すると、実行中のスレッドのプリンシパルが Java Security Manager には影響しません。

Permissions

パーミッションは、コードに付与されるアクセスです。多くのパーミッションは、Java Enterprise Edition 6(Java EE 6)仕様の一部として提供されます。

バグの報告

2.3. JAVA セキュリティーポリシーの作成

ほとんどの JDK および JRE ディストリビューションに **policytool** と呼ばれるアプリケーションが含まれており、Java セキュリティーポリシーを作成および編集することを目的としています。**policytool** に関する詳細情報は、にリンクされてい http://docs.oracle.com/javase/6/docs/technotes/tools/ ます。テキストエディターを使用してセキュリティーポリシーを作成することもできます。

手順2.1 新しい Java Security Manager ポリシーの設定

1. **policytool を起動します。**
   policytool ツールを以下のいずれかの方法で起動します。

   - Red Hat Enterprise Linux
     GUI またはコマンドプロンプトで、**/usr/bin/policytool** を実行します。

   - Microsoft Windows Server
     Start メニューまたは Java インストールの **bin** から **policytool.exe** を実行します。ロケーションは異なる場合があります。

2. **ポリシーを作成します。**
   ポリシーを作成するには、**Add Policy Entry** を選択します。必要なパラメーターを追加してから、完了 をクリック し ます。

注記

VFS を使用して、JBoss EAP にデプロイされたアプリケーションのパスを指定します。Linux では、パスは **vfs:/content/*application.war*** になります。Microsoft Windows では、**vfs:/${user.dir}/content/*application.war*** です。

以下に例を示します。

```
grant codeBase "vfs:/content/application.war/-" {
permission java.util.PropertyPermission "*", "read";
};
```

3. **既存のポリシーを編集します。**

   既存のポリシーの一覧からポリシーを選択し、Edit Policy Entry ボタンを選択します。必要に応じてパラメーターを編集します。

4. **既存のポリシーを削除します。**

   既存のポリシーの一覧からポリシーを選択し、Remove Policy Entry ボタンを選択します。

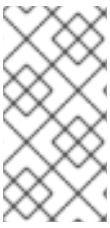
バグの報告

2.4. JAVA SECURITY MANAGER 内での JBOSS EAP 6 の実行

JBoss EAP 6.4 以降、Java Security Manager(JSM)内で JBoss EAP 6 を実行すると、secmgr オプションを使用します。

重要

-Djava.security.manager Java システムプロパティーの直接使用ができなくなりました。以前の JBoss EAP 6 で Java Security Manager を有効にするために使用されていたこの方法は、JBoss EAP 起動スクリプトのフォールバックメカニズムとしてのみサポートされています。



注記

JBoss EAP 6.4 以降では、カスタムセキュリティーマネージャーは使用できません。

以下の手順では、指定のセキュリティーポリシーを使用して Java Security Manager 内で実行するように JBoss EAP 6 インスタンスを設定する手順を説明します。

前提条件

- この手順を実行する前に、Java Development Kit(JDK)に含まれる policytool アプリケーションを使用してセキュリティーポリシーを記述する必要があります。テキストエディターを使用してセキュリティーポリシーを作成することもできます。

  パーミッションが必要なユーザーデプロイメントには、セキュリティーポリシーが必要です。この手順では、ポリシーが *EAP_HOME*/bin/server.policy にあることを前提としています。

- 設定ファイルを編集する前に、ドメインまたはスタンドアロンサーバーを完全に停止する必要があります。

管理対象ドメインで JBoss EAP 6 を使用している場合は、ドメインの各物理ホストまたはインスタンスで以下の手順を実行する必要があります。

手順2.2 JBoss EAP 6 の Java Security Manager の設定

1. 設定ファイルを開く

   設定ファイルを開いて編集します。編集が必要な設定ファイルは、管理対象ドメインまたはスタンドアロンサーバーとオペレーティングシステムのどちらを使用するかによって異なります。

- 管理対象ドメイン
  - For Linux: *EAP_HOME*/bin/domain.conf
  - Windows の場合： *EAP_HOME*\bin\domain.conf.bat
- スタンドアロンサーバー
  - For Linux: *EAP_HOME*/bin/standalone.conf
  - Windows の場合： *EAP_HOME*\bin\standalone.conf.bat

2. Java Security Manager の有効化

   以下のいずれかの方法を使用して Java Security Manager を有効にします。

   - JBoss EAP 6 のサーバー起動スクリプトで -secmgr オプションを使用します。
   - 設定ファイルの secmgr="true" 行のコメントを解除します。

     - Linux の場合:

       ```
       # Uncomment this to run with a security manager enabled
       SECMGR="true"
       ```

- **Windows の場合:**

```
rem # Uncomment this to run with a security manager enabled
set "SECMGR=true"
```

3. **Java セキュリティーポリシーの指定**

-Djava.security.policy を使用して、セキュリティーポリシーの場所を指定できます。これは、改行なしで 1 行のみに置く必要があります。-Djava.security.policy を設定する場合は == を使用すると、セキュリティーマネージャーは指定されたポリシーファイル のみ を使用するように指定します。= を使用すると、セキュリティーマネージャーは *JAVA_HOME*/lib/security/java.security の policy.url セクションに設定されたポリシー と組み合わせ て指定されたポリシーを使用するように指定します。

関連する JBoss EAP 6 設定ファイルでセキュリティーポリシー Java オプションを追加します。管理対象ドメインを使用している場合は、PROCESS_CONTROLLER_JAVA_OPTS および HOST_CONTROLLER_JAVA_OPTS が設定される前に挿入されていることを確認します。

- **Linux の場合:**

```
JAVA_OPTS="$JAVA_OPTS -
Djava.security.policy==$JBOSS_HOME/bin/server.policy -
Djboss.home.dir=$JBOSS_HOME"
```

- **Windows の場合:**

```
set "JAVA_OPTS=%JAVA_OPTS% -
Djava.security.policy==%JBOSS_HOME%\bin\server.policy -
Djboss.home.dir=%JBOSS_HOME%"
```

4. **ドメインまたはサーバーの起動**

ドメインまたはサーバーを通常どおり起動します。

バグの報告

2.5. IBM JDK および JAVA SECURITY MANAGER

IBM JDK のバージョンによっては、JBoss EAP セキュリティーポリシーと適切に動作しないデフォルトのポリシープロバイダーを使用します。IBM JDK を使用して Java Security Manager を有効にして JBoss EAP をホストする際に問題が発生した場合は、JRE 設定を変更して標準のポリシープロバイダーを使用する必要があります。

IBM JDK の JRE 設定を変更するには、JAVA_HOME/jre/lib/security/java.security ファイルを編集し、policy.provider の値を sun.security.provider.PolicyFile に設定します。

```
policy.provider=sun.security.provider.PolicyFile
```

バグの報告

2.6. SECURITY MANAGER ポリシーのデバッグ

デバッグ情報を有効にして、セキュリティーポリシー関連の問題のトラブルシューティングに役立ちます。java.security.debug オプションは、報告されたセキュリティー関連情報のレベルを設定します。java -Djava.security.debug=help コマンドは、デバッグオプションの全範囲でヘルプ出力を生成します。デバッグレベルを all に設定すると、原因が完全に不明ですが、一般的に使用する場合に、情報が過剰に生成されるセキュリティー関連の障害をトラブルシューティングするのに便利です。適切な一般的なデフォルトは access:failure です。

手順2.3 一般的なデバッグの有効化

- この手順を実行すると、セキュリティー関連デバッグ情報の一般的な機密レベルを有効にすることができます。

  次の行をサーバー設定ファイルに追加します。

  - JBoss EAP 6 インスタンスが管理対象ドメインで実行されている場合は、Linux の場合は bin/domain.conf ファイル、Windows の場合は bin\domain.conf.bat ファイルに追加されます。

  -

JBoss EAP 6 インスタンスがスタンドアロンサーバーとして実行されている場合、行は Linux の場合は bin/standalone.conf ファイル、Windows の場合は bin\standalone.conf.bat ファイルに追加されます。

Linux

```
JAVA_OPTS="$JAVA_OPTS -Djava.security.debug=access:failure"
```

Windows

```
set "JAVA_OPTS=%JAVA_OPTS% -Djava.security.debug=access:failure"
```

結果

セキュリティー関連デバッグ情報の一般的なレベルが有効になります。

バグの報告

第3章 セキュリティーレルム

3.1. セキュリティーレルム

セキュリティーレルム は、ユーザーとパスワード間の一連のマッピング、ユーザーとロール間のマッピングです。セキュリティーレルムは、EJB および Web アプリケーションに認証および承認を追加するメカニズムです。JBoss EAP 6 はデフォルトで 2 つのセキュリティーレルムを提供します。

- ManagementRealm は、管理 CLI および Web ベースの管理コンソールの機能を提供する Management API の認証情報を保存します。JBoss EAP 6 自体を管理するための認証システムを提供します。また、アプリケーションが Management API に使用するものと同じビジネスルールで認証する必要がある場合は、ManagementRealm を使用できます。

- ApplicationRealm は、Web アプリケーションおよび EJB のユーザー、パスワード、およびロール情報を保存します。

各レルムはファイルシステムの複数のファイルに保存されます。

- *REALM*-users.properties は、ユーザー名とハッシュ化されたパスワードを保存します。

- *REALM*-roles.properties は、ユーザーからロールへのマッピングを保存します。

- mgmt-groups.properties は、ManagementRealm のユーザーグループマッピングファイルを保存します。ロールベースアクセス制御(RBAC)が有効化されている場合にのみ使用されます。

プロパティーファイルは domain/configuration/ ディレクトリーおよび standalone/configuration/ ディレクトリーに保存されます。ファイルは add-user.sh または add-user.bat コマンドで同時に書き込みされます。コマンドを実行すると、最初に作成した決定は、新しいユーザーを追加するレルムになります。

バグの報告

3.2. 新しいセキュリティーレルムの追加

1. 管理 CLI を実行します。

jboss-cli.sh または jboss-cli.bat コマンドを起動し、サーバーに接続します。

2. **新しいセキュリティーレルム自体を作成します。**

   以下のコマンドを実行し、ドメインコントローラーまたはスタンドアロンサーバーに MyDomainRealm という名前の新しいセキュリティーレルムを作成します。

   ドメインインスタンスの場合は、以下のコマンドを使用します。

   ```
   /host=master/core-service=management/security-realm=MyDomainRealm:add()
   ```

   スタンドアロンインスタンスの場合には、以下のコマンドを使用します。

   ```
   /core-service=management/security-realm=MyDomainRealm:add()
   ```

3. **新規ロールについての情報を格納するプロパティーファイルへの参照を作成します。**

   以下のコマンドを実行して、新しいロールに関連するプロパティーが含まれる myfile.properties という名前のファイルを作成します。

   注記

   新規作成されたプロパティーファイルは、含まれる add-user.sh スクリプトおよび add-user.bat スクリプトで管理されません。これは外部で管理される必要があります。

   ドメインインスタンスの場合は、以下のコマンドを使用します。

   ```
   /host=master/core-service=management/security-realm=MyDomainRealm/authentication=properties:add(path=myfile.properties)
   ```

   スタンドアロンインスタンスの場合には、以下のコマンドを使用します。

   ```
   /core-service=management/security-realm=MyDomainRealm/authentication=properties:add(path=myfile.properties)
   ```

結果

新しいセキュリティーレルムが作成されます。この新しいレルムにユーザーおよびロールを追加す

ると、情報はデフォルトのセキュリティーレルムとは別のファイルに保存されます。この新しいファイルは、独自のアプリケーションまたは手順を使用して管理できます。

バグの報告

3.3. セキュリティーレルムへのユーザーの追加

1. add-user.sh または add-user.bat コマンドを実行します。

   ターミナルを開き、*EAP_HOME*/bin/ ディレクトリーに移動します。Red Hat Enterprise Linux または別の UNIX のようなオペレーティングシステムを実行している場合は、add-user.sh を実行します。Microsoft Windows Server を実行する場合は add-user.bat を実行します。

2. Management User または Application User を追加するかどうかを選択します。

   この手順では、b と入力してアプリケーションユーザーを追加します。

3. ユーザーを追加するレルムを選択します。

   デフォルトでは、利用可能なレルムは ApplicationRealm のみです。カスタムレルムを追加した場合は、代わりに名前を入力できます。

4. プロンプトが表示されたら、ユーザー名、パスワード、およびロールを入力します。

   プロンプトが表示されたら、希望のユーザー名、パスワード、および任意のロールを入力します。yes を入力して選択を確認するか、no と入力して変更をキャンセルします。変更は、セキュリティーレルムの各プロパティーファイルに書き込まれます。

バグの報告

第4章 ネットワークトラフィックの暗号化

4.1. JBOSS EAP 6 が使用するネットワークインターフェースの指定

概要

サービスを分離することで、必要なクライアントのみがアクセスできるようにすることで、ネットワークのセキュリティーが強化されます。JBoss EAP 6 には、デフォルト設定の 2 つのインターフェースが含まれ て います。インターフェースの 1 つは 管理 と呼ばれ、管理 コンソール、CLI、および API によって使用されます。もう 1 つは public と呼ばれ、アプリケーションをデプロイするために使用されます。これらのインターフェースは特別なものや重要ではありませんが、開始点として提供されます。

管理 インターフェースはデフォルトでポート 9990 および 9999 を使用し、パブリック インターフェースはポート 8080、HTTPS を使用する場合はポート 8443 を使用します。

管理インターフェース、パブリックインターフェース、またはその両方の IP アドレスを変更できます。

管理インターフェースを公開する場合は注意してください。

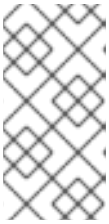
リモートホストからアクセス可能な他のネットワークインターフェースに管理インターフェースを公開する場合は、セキュリティーへの影響に注意してください。多くの場合、管理インターフェースにリモートアクセスを提供することは推奨されません。

1. JBoss EAP 6 を停止します。

   オペレーティングシステムに適した方法で割り込みを送信して JBoss EAP 6 を停止します。JBoss EAP 6 をフォアグラウンドアプリケーションとして実行している場合は、通常、Ctrl+C を押します。

2. バインドアドレスを指定して JBoss EAP 6 を再起動します。

   -b コマンドラインスイッチを使用して、特定のインターフェースで JBoss EAP 6 を起動します。



注記

以下の例では、**10.1.1.1** を使用した **IP** アドレスが利用可能である必要があります。**IP** アドレスを確認するには、**ifconfig** コマンドを使用します。

例4.1 パブリックインターフェースの指定

```
EAP_HOME/bin/domain.sh -b 10.1.1.1
```

例4.2 管理インターフェースの指定

```
EAP_HOME/bin/domain.sh -bmanagement=10.1.1.1
```

例4.3 各インターフェースへの異なるアドレスの指定

```
EAP_HOME/bin/domain.sh -bmanagement=127.0.0.1 -b 10.1.1.1
```

例4.4 すべてのネットワークインターフェースへのパブリックインターフェースのバインド

```
EAP_HOME/bin/domain.sh -b 0.0.0.0
```

XML 設定ファイルを直接編集して、デフォルトのバインドアドレスを変更できます。ただし、これを行うと、**-b** コマンドラインスイッチを使用して実行時に **IP** アドレスを指定できなくなるため、これは推奨されません。これを行う場合は、**XML** ファイルを編集する前に、必ず **JBoss EAP 6** を完全に停止してください。

バグの報告

4.2. JBOSS EAP 6 で動作可能なネットワークファイアウォールの設定

概要

ほとんどの実稼働環境では、ネットワークセキュリティー全体のストラテジーの一部としてファイアウォールを使用します。複数のサーバーインスタンスが相互に通信したり、**Web** サーバーやデータベースなどの外部サービスと通信したりする必要がある場合は、ファイアウォールにこれを考慮する必要があります。適切に管理されたファイアウォールは、操作に必要なポートのみを開き、ポートへのアクセスを特定の **IP** アドレス、サブネット、およびネットワークプロトコルに制限します。

本書では、ファイアウォールの完全な説明は範囲外になります。

前提条件

- 開く必要があるポートを判断します。
- ファイアウォールソフトウェアを理解していること。この手順では、Red Hat Enterprise Linux 6 の system-config-firewall コマンドを使用します。Microsoft Windows Server にはファイアウォールが組み込まれ、プラットフォームごとに複数のサードパーティーのファイアウォールソリューションを利用できます。Microsoft Windows Server では、PowerShell を使用してファイアウォールを設定できます。

前提条件

この手順では、以下を前提として環境のファイアウォールを設定します。

- オペレーティングシステムは Red Hat Enterprise Linux 6 です。
- JBoss EAP 6 はホスト 10.1.1.2 で実行さ れ ます。オプションで、サーバーには独自のファイアウォールがあります。
- ネットワークのファイアウォールサーバーは、ホスト 10. 1.1.1 のインターフェース eth0 で実行され、外部インターフェース eth1 があります。
- ポート 5445 のトラフィック（JMS で使用されるポート）を JBoss EAP 6 に転送する必要があります。ネットワークファイアウォールを介して他のトラフィックを許可しないでください。

手順4.1 ネットワークファイアウォールと JBoss EAP 6 が連携するための管理

1. 管理コンソールへのログイン

   管理コンソールへのログインデフォルトでは、で実行され http://localhost:9990/console/ ます。

2. ソケットバインディンググループが使用するソケットバインディングを決定します。

   a. 管理コンソールの上部にある Configuration ラベルをクリックします。

b.
General Configuration メニューを展開します。Socket Binding を選択します。

c.
Socket Binding Declarations 画面が表示されます。最初は、standard-sockets グループが表示されます。右側のコンボボックスからそのグループを選択して、別のグループを選択します。

注記

スタンドアロンサーバーを使用する場合は、1 つのソケットバインディンググループのみが存在します。

ソケット名とポートのリストが表示されます（ページごとに 8 つの値）。テーブルの下にある矢印ナビゲーションを使用すると、ページを移動できます。

3. 開く必要があるポートを判断します。

特定ポートの機能および環境の要件によっては、一部のポートをファイアウォール上で開く必要がある場合があります。

4. JBoss EAP 6 にトラフィックを転送するようファイアウォールを設定します。

以下の手順を実行して、必要なポートでトラフィックを許可するようネットワークファイアウォールを設定します。

a.
root ユーザーとしてファイアウォールマシンにログインし、コマンドプロンプトにアクセスします。

b.
system-config-firewall コマンドを実行してファイアウォール設定ユーティリティーを起動します。ファイアウォールシステムにログインする方法に応じて、GUI またはコマンドラインユーティリティーが起動します。このタスクでは、SSH でログインし、コマンドラインインターフェースを使用します。

c.
キーボードの TAB キーを使用して Customize ボタンに移動し、ENTER キーを押します。Trusted Services 画面が表示されます。

d.
値は変更しないでくださいが、TAB キーを使用して Forward ボタンに移動し、ENTER を押して次の画面に移動します。他のポート画面が 表示されます。

e.
TAB キーを使用して < Add> ボタンに 移動し、ENTER を押します。Port and Protocol 画面が表示されます。

f.
Port / Port Range フィールドに 5445 を入力し、TAB キーを使用して Protocol フィールドに移動し、tcp と入力します。TAB キーを使用して OK ボタンに移動し、ENTER を押します。

g.
TAB キーを使用して、Port Forwarding 画面に到達するまで Forward ボタンに移動します。

h.
TAB キーを使用して < Add> ボタン に移動し、ENTER キーを押します。

i.
以下の値を入力して、ポート 5445 のポート転送を設定します。

- 送信元インターフェース： eth1
- protocol: tcp
- ポート/ポート範囲： 5445
- 宛先 IP アドレス： 10 .1.1.2
- ポート/ポート範囲： 5445

TAB キーを使用して OK ボタンに移動し、ENTER を押します。

j.
TAB キーを使用して Close ボタンに移動し、ENTER を押します。

k.
TAB キーを使用して OK ボタンに移動し、ENTER を押します。変更を適用するには、警告を確認して Yes をクリックします。

5. JBoss EAP 6 ホストでファイアウォールを設定します。

   一部の組織では、JBoss EAP 6 サーバー自体でファイアウォールを設定し、操作に必要でないすべてのポートを閉じます。「JBoss EAP 6 により使用されるネットワークポート」 を参照して開くポートを決定してから、残りを閉じます。Red Hat Enterprise Linux 6 のデフォルト設定は、22 （Secure Shell(SSH)および 5353 （マルチキャスト DNS で使用される）以外のポートをすべて閉じます。ポートを設定する際には、誤ってロックアウトしないように、サーバーに物理アクセスがあることを確認します。

結果

ファイアウォールは、ファイアウォール設定で指定した方法で、トラフィックを内部 JBoss EAP 6 サーバーに転送するように設定されます。サーバーでファイアウォールを有効にすることを選択すると、アプリケーションの実行に必要なポートを除くすべてのポートが閉じられます。

手順4.2 PowerShell を使用した Microsoft Windows 上でのファイアウォールの設定

1. デバッグの目的でファイアウォールを停止し、現在のネットワークの挙動がファイアウォールの設定と関係しているかどうかを判断します。

```
Start-Process "$psHome\powershell.exe" -Verb Runas -ArgumentList '-command
"NetSh Advfirewall set allprofiles state off"'
```

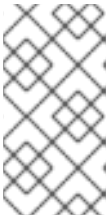
2. ポート 23364 で UDP 接続を許可します。以下に例を示します。

```
Start-Process "$psHome\powershell.exe" -Verb Runas -ArgumentList '-command
"NetSh Advfirewall firewall add rule name="UDP Port 23364" dir=in  action=allow
protocol=UDP localport=23364"'
Start-Process "$psHome\powershell.exe" -Verb Runas -ArgumentList '-command
"NetSh Advfirewall firewall add rule name="UDP Port 23364" dir=out action=allow
protocol=UDP localport=23364"'
```

手順4.3 mod_cluster アドバタイズを許可するよう Red Hat Enterprise Linux 7 でファイアウォールを設定

- Red Hat Enterprise Linux 7 で mod_cluster のアドバタイズを有効にするには、以下のようにファイアウォールで UDP ポートを有効にする必要があります。

```
firewall-cmd --permanent --zone=public --add-port=23364/udp
```



注記

UDP マルチキャストをアドバタイズする mod_cluster バランサーのデフォルトアドレスおよびポートは 224.0.1.105:23364 です。

バグの報告

4.3. JBOSS EAP 6 により使用されるネットワークポート

JBoss EAP 6 のデフォルト設定で使用されるポートは、次の複数の要因によって異なります。

- サーバーグループがデフォルトのソケットバインディンググループのいずれかを使用するか、またはカスタムグループを使用するかどうか。
- 個別デプロイメントの要件。

数値ポートオフセット

同じ物理サーバーで複数のサーバーを実行する際にポートの競合を軽減するために、数値ポートオフセットを設定できます。サーバーが数値ポートオフセットを使用する場合は、サーバーグループのソケットバインディンググループのデフォルトのポート番号にオフセットを追加します。たとえば、ソケットバインディンググループの HTTP ポートが 8080 で、サーバーが 100 のポートオフセットを使用する場合、HTTP ポートは 8180 になります。

特に指定がない限り、ポートは TCP プロトコルを使用します。

デフォルトのソケットバインディンググループ

- full-ha-sockets

- **full-sockets**
- **ha-sockets**
- **standard-sockets**

これらのソケットバインディンググループは domain.xml でのみ利用できます。スタンドアロンサーバープロファイルには、標準のソケットバインディンググループのみが含まれます。このグループは standalone.xml の standard-sockets、standalone-ha.xml のha-sockets、standalone-full.xml の場合はfull-sockets、standalone-full-ha.xml の場合は full-ha-sockets に対応します。スタンドアロンプロファイルには、management-{native,http,https} などのソケットバインディングが含まれます。

表4.1 デフォルトのソケットバインディングの参照

名前	ポート	マルチキャストポート	説明	full-ha-sockets	full-sockets	ha-socket	standard-socket
ajp	8009		Apache JServ プロトコル。HTTP クラスタリングおよび負荷分散に使用します。	はい	はい	はい	はい
http	8080		デプロイされた Web アプリケーションのデフォルトポート。	はい	はい	はい	はい
https	8443		デプロイされた Web アプリケーションとクライアント間の SSL 暗号化接続。	はい	はい	はい	はい
jacorb	3528		JTS トランザクションおよび他の ORB 依存サービス用の CORBA サービス。	はい	はい	いいえ	いいえ
jacorb-ssl	3529		SSL 暗号化 CORBA サービス。	はい	はい	いいえ	いいえ
jgroups-diagnostics		7500	マルチキャスト。HA クラスターでのピア検出に使用されます。管理インターフェースを使用して設定できません。	はい	いいえ	はい	いいえ

名前	ポート	マルチキャストポート	説明	full-ha-sockets	full-sockets	ha-socket	standard-socket
jgroups-mping		45700	マルチキャスト。HA クラスターでの初期メンバーシップの検出に使用されます。	はい	いいえ	はい	いいえ
jgroups-tcp	7600		TCP を使用した、HA クラスター内でのユニキャストピア検出。	はい	いいえ	はい	いいえ
jgroups-tcp-fd	57600		TCP を介した HA 障害検出に使用されます。	はい	いいえ	はい	いいえ
jgroups-udp	55200	45688	UDP を使用した、HA クラスター内でのマルチキャストピア検出。	はい	いいえ	はい	いいえ
jgroups-udp-fd	54200		UDP を介した HA 障害検出に使用されます。	はい	いいえ	はい	いいえ
messaging	5445		JMS サービス。	はい	はい	いいえ	いいえ
messaging-group			HornetQ JMS ブロードキャストと検出グループにより参照されます。	はい	はい	いいえ	いいえ
messaging-throughput	5455		JMS Remoting により使用されます。	はい	はい	いいえ	いいえ
mod_cluster		23364	JBoss EAP 6 と HTTP ロードバランサー間の通信に対するマルチキャストポート。	はい	いいえ	はい	いいえ
remoting	4447		リモート EJB の呼び出しに使用されます。	はい	はい	はい	はい

名前	ポート	マルチキャストポート	説明	full-ha-sockets	full-sockets	ha-socket	standard-socket
txn-recovery-environment	4712		JTA トランザクションリカバリーマネージャー。	はい	はい	はい	はい
txn-status-manager	4713		JTA / JTS トランザクションマネージャー。	はい	はい	はい	はい

管理ポート

ソケットバインディンググループの他に、各ホストコントローラーによって 2 つのポートが管理目的で開かれます。

- **9990 - Web 管理コンソールポート**
- **9999 - 管理コンソールと管理 API が使用するポート**

さらに、管理コンソールに対して HTTPS が有効になっている場合、9443 もデフォルトポートとして開かれます。

バグの報告

4.4. 暗号化について

暗号化 とは、数学的なアルゴリズムを適用して機密情報を難読化することを指します。暗号化とは、データ障害やシステム停止などのリスクからインフラストラクチャーを保護する基盤の 1 つです。

暗号化は、パスワードなどの単純な文字列データに適用できます。データ通信ストリームにも適用できます。たとえば、HTTPS プロトコルは、ある当事者から別のデータに転送する前に、すべてのデータを暗号化します。SSH(Secure Shell)プロトコルを使用してサーバーから別のサーバーに接続すると、すべての通信が暗号化された *トンネル* で送信されます。

バグの報告

4.5. SSL 暗号化について

Secure Sockets Layer(SSL)は、2 つのシステム間のネットワークトラフィックを暗号化します。2 つのシステム間のトラフィックは、接続の *ハンドシェイク* フェーズで生成され、この 2 つのシステムによってのみ認識される双方向キーを使用して暗号化されます。

双方向暗号化キーをセキュアに交換するために、SSL は、*キーペア* を使用する暗号化の手法である Public Key Infrastructure(PKI)を使用します。*キーペア* は、個別のキーで一致する暗号化キー（公開鍵と秘密鍵）の 2 つで構成されます。公開鍵は他の人と共有され、データの暗号化に使用され、秘密鍵は秘密を保持し、公開鍵を使用して暗号化されたデータの復号化に使用されます。

クライアントがセキュアな接続を要求すると、セキュアな通信の開始前にハンドシェイクフェーズが行われます。SSL ハンドシェイク時に、サーバーは公開鍵を証明書形式でクライアントに渡します。証明書には、サーバーの ID(its URL)、サーバーの公開鍵、および証明書を検証するデジタル署名が含まれます。その後、クライアントは証明書を検証し、証明書が信頼できるかどうかについて決定します。証明書が信頼できると、クライアントは SSL 接続の双方向暗号化鍵を生成し、サーバーの公開鍵を使用して暗号化し、サーバーに送り返します。サーバーは、双方向の暗号化鍵を復号し、秘密鍵を使用して、この接続上の 2 つのマシン間の通信は、双方向の暗号化キーを使用して暗号化されます。

警告

Red Hat は、影響を受けるすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSL を明示的に無効にすることを推奨します。

バグの報告

4.6. JBOSS EAP 6 WEB サーバーでの SSL 暗号化の実装

はじめに

多くの Web アプリケーションには、HTTPS 接続とも呼ばれるクライアントとサーバー間の SSL 暗号化接続が必要です。以下の手順を使用して、サーバーまたはサーバーグループで HTTPS を有効にすることができます。

警告

Red Hat は、影響を受けるすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSL を明示的に無効にすることを推奨します。

前提条件

- SSL 暗号化キーのセットと SSL 暗号化証明書。これらは、証明書署名認証局から購入したり、コマンドラインユーティリティーを使用して生成したりできます。Red Hat Enterprise Linux で利用可能なユーティリティーを使用して暗号鍵を生成するには、「SSL 暗号化キーおよび証明書の生成」 を参照してください。
- 特定の環境と設定に関する以下の詳細。
  - 証明書ファイルが保存されている完全なディレクトリー名。
  - 暗号化キーの暗号化パスワード。
- ドメインコントローラーまたはスタンドアロンサーバーに接続する稼働中の管理 CLI。
- 適切な暗号化スイートを選択します。

暗号化スイート

暗号化スイートを形成するためにビルディングブロックとして使用される暗号プリミティブが多数あります。最初の表には、推奨される暗号化プリミティブが記載されています。2 つ目は、既存のソフトウェアとの互換性のために使用できる暗号化プリミティブを一覧表示しています が、セキュリティーレベルは推奨として考慮されません。

警告

Red Hat は、cipher-suite に使用する強力な暗号のセットを選択的にホワイトリスト化することを推奨します。弱い暗号を有効にすると、セキュリティーリスクが大きくなります。互換性の問題が生じる可能性があるため、特定の暗号スイートに決定する前に JDK ベンダーのドキュメントを参照してください。

表4.2 推奨される暗号プリミティブ

2048 ビットキーと OAEP を使用する RSA
CBC モードの AES-128
SHA-256
HMAC-SHA-256
HMAC-SHA-1

表4.3 その他の暗号プリミティブ

1024 以上のキーサイズとレガシーパディングを使用する RSA
AES-192
AES-256
3DES (トリプル DES で、2 つまたは 3 つの 56 ビットキー)
RC4 (非推奨)
SHA-1
HMAC-MD5

コネクターの SSL プロパティーに設定できるパラメーターの完全リストは、「SSL コネクターリファレンス」 を参照してください。

注記

この手順では、管理対象ドメインを使用する JBoss EAP 6 設定に適切なコマンドを使用します。スタンドアロンサーバーを使用する場合は、管理 CLI コマンドの最初から /profile=default を削除して管理 CLI コマンドを変更し、jboss.domain.config.dir プロパティーのインスタンスを jboss.server.config.dir に置き換えます（ jboss.domain.config.dir はスタンドアロンモードでは使用できません）。

警告

Red Hat は、影響を受けるすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSL を明示的に無効にすることを推奨します。

手順4.4 JBoss Web Server が HTTPS を使用するよう設定

1. **新しい HTTPS コネクターを追加します。**

   https スキーム、https ソケットバインディング（デフォルトは 8443）を使用し、セキュアに設定されている *HTTPS* という名前のセキュアなコネクターを作成します。

```
/profile=default/subsystem=web/connector=HTTPS/:add(socket-
binding=https,scheme=https,protocol=HTTP/1.1,secure=true)
```

2. **SSL 暗号化証明書およびキーを設定します。**

   SSL 証明書を設定し、例に独自の値を置き換えます。この例では、キーストアがサーバー設定ディレクトリー（管理対象ドメインの *EAP_HOME*/domain/configuration/ ）にコピーされることを前提としています。

```
/profile=default/subsystem=web/connector=HTTPS/ssl=configuration:add(name=https,certificate
-key-file="${jboss.domain.config.dir}/keystore.jks",password=SECRET, key-
alias=KEY_ALIAS, cipher-suite=CIPHERS)
```

3. **プロトコルを TLSv1 に設定します。**

```
/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-
attribute(name=protocol,value=TLSv1)
```

4. **アプリケーションをデプロイします。**

   設定したプロファイルを使用するサーバーグループにアプリケーションをデプロイします。スタンドアロンサーバーを使用する場合は、サーバーにアプリケーションをデプロイしま

す。新しい SSL 暗号化接続を使用する HTTPS リクエスト。

バグの報告

4.7. SSL 暗号化キーおよび証明書の生成

SSL で暗号化された HTTP 接続(HTTPS)や、その他のタイプの SSL で暗号化された通信を使用するには、署名済み暗号化証明書が必要です。認証局(CA)から証明書を購入するか、自己署名証明書を使用できます。自己署名証明書は、多くのサードパーティーが信頼しているとは見なされませんが、内部テストの目的で適しています。

この手順を実行すると、Red Hat Enterprise Linux で利用可能なユーティリティーを使用して自己署名証明書を作成できます。

前提条件

- Java Development Kit 実装によって提供される keytool ユーティリティーが必要です。Red Hat Enterprise Linux の OpenJDK は、このコマンドを /usr/bin/keytool にインストールします。
- keytool コマンドの構文とパラメーターを理解します。この手順では、SSL 証明書または keytool コマンドの詳細な説明は本書の対象外であるため、非常に一般的な手順を使用します。

手順4.5 SSL 暗号化キーおよび証明書の生成

1. パブリックキーおよびプライベートキーとともにキーストアを生成します。<remark>Bugzilla のバグの内容を反映済み</remark>

   以下のコマンドを実行して、現在のディレクトリーにエイリアス jboss が含まれる server.keystore という名前のキーストアを生成します。

   ```
   keytool -genkeypair -alias jboss -keyalg RSA -keystore server.keystore -storepass mykeystorepass --dname "CN=jsmith,OU=Engineering,O=mycompany.com,L=Raleigh,S=NC,C=US"
   ```

   この keytool コマンドで使用されるパラメーターの説明は次のとおりです。

パラメーター	説明
-genkeypair	公開鍵と秘密鍵が含まれるキーペアを生成する **keytool** コマンド。
-alias	キーストアのエイリアス。この値は任意ですが、エイリアス **jboss** は JBoss Web サーバーによって使用されるデフォルトです。
-keyalg	キーペア生成アルゴリズム。この場合、これは **RSA** です。
-keystore	キーストアファイルの名前および場所。デフォルトの場所は、現在のディレクトリーです。選択する名前は任意です。この場合、ファイルの名前は **server.keystore** になります。
-storepass	このパスワードは、鍵を読み取りできるようにキーストアに対して認証するために使用されます。パスワードは 6 文字以上である必要があります。パスワードは、キーストアへのアクセス時に入力する必要があります。この例では、**mykeystorepass** を使用しています。このパラメーターを省略すると、コマンドの実行時に入力が求められます。
-keypass	実際の鍵のパスワードです。 **注記** 実装の制限により、ストアと同じパスワードを使用する必要があります。

パラメーター	説明
--dname	キーの識別名を記述する引用符で囲まれた文字列（例： "CN=jsmith,OU=Engineering,O=mycompany.com,L=Raleigh,C=US"）。この文字列は、以下のコンポーネントを連結したものです。 ● **CN**: 共通名またはホスト名。ホスト名が "jsmith.mycompany.com" の場合、**CN** は jsmith になります。 ● **OU** - 組織単位（例： Engineering） ● **O** - 組織名（例： mycompany.com）。 ● l - ローカリティー（例： Raleigh または「London」） ● **s - "NC" など、状態または 提案について参照してください。このパラメーターは任意です。** ● **c - 2 文字の国コード。たとえば、「US」または「UK」です。**

上記のコマンドを実行すると、次の情報が要求されます。

- **コマンドラインで *-storepass* パラメーターを使用しなかった場合は、キーストアパスワードを入力するように求められます。次のプロンプトで新しいパスワードを再入力します。**

- **コマンドラインで -keypass パラメーターを使用しない場合は、キーパスワードを入力するように求められます。Enter を押して、キーストアパスワードと同じ値に設定します。**

コマンドが完了すると、ファイル server.keystore にエイリアス jboss のある単一のキーが含まれるようになりました。

2. **鍵を確認します。**

   以下のコマンドを使用して、キーが正常に動作することを検証します。

```
keytool -list -keystore server.keystore
```

   キーストアパスワードの入力が求められます。キーストアの内容が表示されます（この場合は、jbossと呼ばれる単一のキー）。jboss キーのタイプ(PrivateKeyEntry)に注目してください。これは、キーストアにこのキーのパブリックおよびプライベートエントリーの両方が含まれることを示します。

3. **証明書の署名要求を生成します。**

   次のコマンドを実行し、手順 1 で作成したキーストアより公開鍵を使用して証明書署名要求を生成します。

```
keytool -certreq -keyalg RSA -alias jboss -keystore server.keystore -file certreq.csr
```

   キーストアに対して認証を行うためにパスワードが要求されます。次に、keytool コマンドは、現在の作業ディレクトリーに certreq.csr という新しい証明書署名要求を作成します。

4. **新しく生成された証明書署名要求をテストします。**

   以下のコマンドを使用して証明書の内容をテストします。

```
openssl req -in certreq.csr -noout -text
```

   証明書の詳細が表示されます。

5. **オプション: 証明書署名要求を認証局 (CA) に送信します。**

   認証局(CA)は証明書を認証できるため、サードパーティークライアントが信用できると見なされます。CA は署名済み証明書を提供し、必要に応じて 1 つ以上の中間証明書を提供します。

6. **オプション: キーストアからの自己署名証明書のエクスポート**

   テストまたは内部目的でのみ必要な場合は、自己署名証明書を使用できます。以下のように、手順 1 で作成したキーストアからエクスポートできます。

```
keytool -export -alias jboss -keystore server.keystore -file server.crt
```

キーストアに対して認証を行うためにパスワードが要求されます。現在の作業ディレクトリーに、**server.crt** という名前の自己署名証明書が作成されます。

7. 署名済み証明書を中間証明書とともにインポートします。

CA によって指示された順序で各証明書をインポートします。インポートする証明書ごとに、**intermediate.ca** または **server.crt** を実際のファイル名に置き換えます。証明書が個別のファイルとして提供されない場合は、各証明書に個別のファイルを作成し、その内容をファイルに貼り付けます。

注記

署名済み証明書および証明書キーは、有用なアセットです。サーバー間での転送方法に注意してください。

```
keytool -import -keystore server.keystore -alias intermediateCA -file intermediate.ca
```

```
keytool -importcert -alias jboss -keystore server.keystore -file server.crt
```

8. 証明書が正常にインポートされたことをテストします。

以下のコマンドを実行し、プロンプトが表示されたらキーストアのパスワードを入力します。キーストアの内容が表示され、証明書はリストに含まれます。

```
keytool -list -keystore server.keystore
```

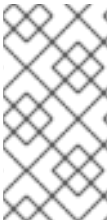
結果

署名済み証明書はキーストアに含まれ、**HTTPS Web** サーバー通信を含む **SSL** 接続を暗号化するために使用できます。

バグの報告

4.8. SSL コネクターリファレンス

JBoss Web コネクターには以下の **SSL** 設定属性を含めることができます。提供される **CLI** コマンドは、プロファイルの デフォルト を使用して管理対象ドメイン用に設計されています。プロファイル名を、管理対象ドメインに設定するものに変更するか、スタンドアロンサーバーのコマンドの **/profile=default** 部分を省略します。



注記

表に記載されている write-attribute CLI コマンドを使用する前に、ssl=configuration を追加する必要があります。

表4.4 SSL コネクター属性

属性	説明	CLI コマンド
name	SSL コネクターの表示名。	属性 **名** は読み取り専用です。

属性	説明	CLI コマンド
verify-client	HTTP/HTTPS コネクターまたはネイティブ APR コネクターが使用されているかによって、**verify-client** の可能な値は異なります。 **HTTP/HTTPS コネクター** 使用できる値は **true**、**false**、または **want** です。**true** に設定すると、接続を受け入れる前にクライアントから有効な証明書チェーンが必要になります。SSL スタックがクライアント証明書を要求し、提供しない場合は失敗しないように設定します。**クライアントが CLIENT-CERT 認証を使用するセキュリティー制約で保護されたリソースを要求しない限り、証明書チェーンを必要としないよう false（デフォルト）に設定します。** **write-attribute CLI コマンドを使用する前に、APR コネクターを追加する必要があります。** **ネイティブ APR コネクター** **使用できる値は オプション であり、required、optionalNoCA、および none（または noneと同じ効果のある他の文字列）です。これらの値は、認定を任意、必須、認証局なしの任意であるか、または全く必要ないかを決定します。デフォルトは none で、クライアントが証明書を送信する機会を持たないことを意味します。**	最初のコマンド例は HTTPS コネクターを使用します。 ``` /profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=verify-client,value=want) ``` 2 つ目のコマンド例は APR コネクターを使用します。 ``` /profile=default/subsystem=web/connector=APR/ssl=configuration/:write-attribute(name=verify-client,value=require) ```

属性	説明	CLI コマンド
verify-depth	クライアントが有効な証明を持たないと判断するまでにチェックされる中間証明書発行者の最大数。デフォルト値は **10** です。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=verify-depth,value=10)
certificate-key-file	署名済みサーバー証明書が保存されるキーストアの完全ファイルパスおよびファイル名。JSSE 暗号化では、この証明書ファイルは唯一のファイルとなり、OpenSSL は複数のファイルを使用します。デフォルト値は、JBoss EAP 6 を実行しているユーザーのホームディレクトリーにある **.keystore** ファイルです。**keystoreType** でファイルを使用しない場合は、パラメーターを空の文字列に設定します。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=certificate-key-file,value=../domain/configuration/server.keystore)
certificate-file	OpenSSL 暗号化を使用する場合は、このパラメーターの値を、サーバー証明書を含むファイルに対するパスに設定します。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=certificate-file,value=server.crt)
password	トラストストアとキーストアの両方のパスワード。以下の例では、*PASSWORD* を独自のパスワードに置き換えます。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=password,value=*PASSWORD*)

属性	説明	CLI コマンド
protocol	使用する SSL プロトコルのバージョン。サポートされる値は、下層の SSL 実装（JSSE または OpenSSL）によって異なります。Java SSE ドキュメント を参照してください。 また、プロトコルの組み合わせ（コンマ区切り）を指定することもできます。例： TLSv1、TLSv1.1、TLSv1.2 **警告** Red Hat は、影響を受けるすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSL を明示的に無効にすることを推奨します。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=protocol,value=ALL) /profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=protocol,value="TLSv1, TLSv1.1,TLSv1.2")

属性	説明	CLI コマンド
cipher-suite	許可される暗号の一覧。JSSE 構文については、コンマ区切りのリストが必要です。OpenSSL 構文については、コロン区切のリストが必要です。必ず 1 つの構文のみを使用してください。 デフォルトは **HIGH:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5** です。 この例では、2 つの暗号だけが一覧表示されますが、実際の例はより多くの暗号が使用されます。  **重要** 強度の低い暗号を使用すると、セキュリティが重大なリスクにさらされることになります。NIST で推奨される暗号化スイートは、を参照 http://www.nist.gov/manuscript-publication-search.cfm?pub_id=915295 してください。 利用可能な OpenSSL 暗号の一覧は、を参照してください https://www.openssl.org/docs/apps/ciphers.html#CIPHER_STRINGS。**@SECLEVEL**、**SUITEB128**、SUITEB128 のみ、SUITEB**128 のみ、SUITEB 192** はサポートされません。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=cipher-suite,value="TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA")
key-alias	キーストアのサーバー証明書に使用されるエイリアス。以下の例では、*KEY_ALIAS* を、証明書のエイリアスに置き換えます。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=key-alias,value=*KEY_ALIAS*)
truststore-type	トラストストアのタイプ。**PKCS12** や Java の標準 **JKS** など、さまざまなタイプのトラストストアが利用できます。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=truststore-type,value=jks)

属性	説明	CLI コマンド
keystore-type	キーストアのタイプ。**PKCS12** や Java の標準 **JKS** など、さまざまなタイプのキーストアが利用できます。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=keystore-type,value=jks)
ca-certificate-file	CA 証明書を含むファイル。これは、**truststoreFile**（JSSE の場合）で、キーストアと同じパスワードを使用します。**ca-certificate-file** ファイルは、クライアント証明書を検証するために使用されます。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=certificate-file,value=ca.crt)
ca-certificate-password	**ca-certificate-file** の証明書パスワード。以下の例では、*MASKED_PASSWORD* を、実際のマスクされたパスワードに置き換えます。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=ca-certificate-password,value=*MASKED_PASSWORD*)
ca-revocation-url	失効リストが含まれるファイルまたは URL。これは、JSSE の **crlFile**、SSL の場合は **SSLCARevocationFile** を参照します。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=ca-revocation-url,value=ca.crl)
session-cache-size	SSLSession キャッシュのサイズ。この属性は、JSSE コネクターにのみ適用されます。デフォルトは **0** で、無制限のキャッシュサイズを指定します。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=session-cache-size,value=100)
session-timeout	キャッシュされた SSLSession が期限切れになるまでの秒数。この属性は、JSSE コネクターにのみ適用されます。デフォルトは **86400** 秒（24 時間）です。	/profile=default/subsystem=web/connector=HTTPS/ssl=configuration/:write-attribute(name=session-timeout,value=43200)

注記

パフォーマンステストでは、明示的な暗号およびプロトコルを設定する必要があります。

バグの報告

4.9. FIPS 140-2 準拠の暗号化

4.9.1. FIPS 140-2 の準拠

FIPS 140-2(Federal Information Processing Standard 140-2)は、暗号化ソフトウェアモジュールの取り組みに関する米国政府コンピューターセキュリティー標準です。FIPS 140-2 コンプライアンスは、政府およびプライベートセクタービジネスが使用するソフトウェアシステムの要件であることがよくあります。

JBoss EAP 6 は外部モジュールによる暗号化を使用します。また、FIPS 140-2 に準拠する暗号化モジュールを使用するよう設定できます。

バグの報告

4.9.2. IBM JDK における FIPS 140-2 準拠暗号化

IBM JDK では、マルチプラットフォームの IBM® JCE(Java™ Cryptographic Extension)IBMJCEFIPS プロバイダーおよび IBM JSSE(Java Secure Sockets Extension)FIPS 140-2 Cryptographic Module(IBMJSSEFIPS)は、FIPS 140-2 準拠の暗号を提供します。

IBMJCEFIPS プロバイダーの詳細は、IBM JCEFIPS の IBM ドキュメント および NIST IBMJCEFIPS - セキュリティーポリシー を参照してください。

キーストレージ

IBM JCE はキーストアを提供しないことに注意してください。このキーはコンピューターに保存され、その物理境界は残しません。このキーがコンピューター間で移動している場合は暗号化する必要があります。

FIPS 準拠のモードで keytool を実行するには、以下のような各コマンドで -providerClass オプションを使用します。

```
keytool -list -storetype JCEKS -keystore mystore.jck -storepass mystorepass -providerClass
com.ibm.crypto.fips.provider.IBMJCEFIPS
```

FIPS プロバイダー情報の確認

サーバーが使用する IBMJCEFIPS に関する情報を確認するには、-Djavax.net.debug=true を standalone.conf または domain.conf に追加してデバッグレベルのロギングを有効にします。FIPS プロバイダーに関する情報は server.log に記録されます。以下に例を示します。

```
04:22:45,685 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using MessageDigest SHA from
provider IBMJCEFIPS version 1.7
04:22:45,689 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH KeyPairGenerator  from provider
from init IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using KeyFactory DiffieHellman from
provider IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using KeyAgreement DiffieHellman
from provider IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH KeyAgreement  from provider
IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH KeyAgreement  from provider
from initIBMJCEFIPS version 1.7
```

バグの報告

4.9.3. FIPS 140-2 準拠のパスワード

FIPS 準拠のパスワードは以下の条件を満たす必要があります。

1. **7 文字以上であること。**
2. **以下の文字クラスのうち、3 クラス以上の文字が含まれること。**
   - **ASCII 数字**
   - **小文字の ASCII**
   - **大文字の ASCII**

- 英数字以外の ASCII

- ASCII 以外の文字

パスワードの最初の文字が大文字の ASCII である場合、2. にある大文字の ASCII として見なされません。

パスワードの最後の文字が ASCII 数字である場合、制限 2 の ASCII 数字とは見なされません。

バグの報告

4.9.4. Red Hat Enterprise Linux 6 にて SSL の FIPS 140-2 準拠の暗号を有効化

ここでは、JBoss EAP 6 の Web コンテナー(JBoss Web)を SSL の FIPS 140-2 準拠の暗号化に設定する方法を説明します。ここでは、Red Hat Enterprise Linux 6 でこの手順のみを説明します。

このタスクでは、FIPS モードの Mozilla NSS ライブラリーを使用します。

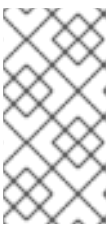
前提条件

- Red Hat Enterprise Linux 6 が FIPS 140-2 に準拠するよう設定されている必要があります。を参照して https://access.redhat.com/knowledge/solutions/137833 ください。

手順4.6 SSL に対して FIPS 140-2 準拠の暗号化を有効にする

1. データベースの作成

   jboss ユーザーが所有するディレクトリーに NSS データベースを作成します。



注記

jboss ユーザーは単なる例となります。オペレーティングシステムのユーザーに置き換える必要があります。

```
$ mkdir -p  /usr/share/jboss-as/nssdb
$ chown jboss /usr/share/jboss-as/nssdb
$ modutil -create -dbdir /usr/share/jboss-as/nssdb
```

2. **NSS 設定ファイルの作成**

以下の内容を含む /usr/share/jboss-as ディレクトリーに nss_pkcsll_fips.cfg という名前の新規テキストファイルを作成します。

```
name = nss-fips
nssLibraryDirectory=/usr/lib64
nssSecmodDirectory=/usr/share/jboss-as/nssdb
nssModule = fips
```

NSS 設定ファイルには以下が指定されている必要があります。

- **名前**
- **NSS ライブラリーが存在するディレクトリー**
- **手順 1 に従って作成された NSS データベースが存在するディレクトリー**

64 ビットバージョンの Red Hat Enterprise Linux 6 を実行していない場合は、/usr/lib 64 ではなく、nssLibraryDirectory を /usr/lib64 に設定します。

3. **SunPKCS11 プロバイダーの有効化**

JRE($JAVA_HOME/jre/lib/security/ java.security)の java.security設定ファイルを編集し、以下の行を追加します。

```
security.provider.1=sun.security.pkcs11.SunPKCS11  /usr/share/jboss-
as/nss_pkcsll_fips.cfg
```

この行に指定されている設定ファイルは手順 2 で作成されたファイルであることに注意してください。

このプロバイダーに優先順位が指定されるように、このファイルの他の security.provider.*X* 行では *X* の値を 1 つ増やす必要があります。

4. **NSS ライブラリーに対して FIPS モードを有効にする**

   以下のように modutil コマンドを実行して、FIPS モードを有効にします。

```
modutil -fips true -dbdir /usr/share/jboss-as/nssdb
```

   ここで指定するディレクトリーは手順 1 で作成したものであることに注意してください。

   この時点で、セキュリティーライブラリーエラーが発生し、NSS 共有オブジェクトの一部に対してライブラリー署名の再生成が必要になることがあります。

5. **FIPS トークンのパスワードの変更**

   以下のコマンドを使用して FIPS トークンにパスワードを設定します。トークンの名前は NSS FIPS 140-2 Certificate DB である必要があることに注意してください。

```
modutil -changepw "NSS FIPS 140-2 Certificate DB" -dbdir /usr/share/jboss-as/nssdb
```

   FIPS トークンに使用されるパスワードは、FIPS に準拠したパスワードである必要があります。

6. **NSS ツールを使用した証明書の作成**

   次のコマンドを入力し、NSS ツールを使用して証明書を作成します。

```
certutil -S -k rsa -n jbossweb  -t "u,u,u" -x -s "CN=localhost, OU=MYOU, O=MYORG,
L=MYCITY, ST=MYSTATE, C=MY" -d /usr/share/jboss-as/nssdb
```

7. **PKCS11 キーストアを使用するよう HTTPS コネクターを設定する**

   JBoss CLI ツールで次のコマンドを使用し、HTTPS コネクターを追加します。

```
/subsystem=web/connector=https/:add(socket-
binding=https,scheme=https,protocol=HTTP/1.1,secure=true)
```

   次に、以下のコマンドを使用して SSL 設定を追加します。PASSWORD を 手順 5 の FIPS 準拠のパスワードに置き換えます。

```
/subsystem=web/connector=https/ssl=configuration:add(name=https,password=PASSWORD,ke
ystore-type=PKCS11,
cipher-
suite="SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_S
```

```
HA,
TLS_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_128_
CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_3DES_EDE
_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_
CBC_SHA,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_RSA_WITH_AES_128_CBC_S
HA,
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_
SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_
SHA,
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_anon_WITH_AES_128_CBC_
SHA,
TLS_ECDH_anon_WITH_AES_256_CBC_SHA")
```

8. **検証**

   以下のコマンドを実行することで、JVM が PKCS11 キーストアから秘密鍵を読み取りできることを確認します。

```
keytool -list -storetype pkcs11
```

例4.5 FIPS 140-2 に準拠した HTTPS コネクターの XML 設定

```
<connector name="https" protocol="HTTP/1.1" scheme="https" socket-binding="https"
secure="true">
  <ssl name="https" password="*****"
      cipher-
suite="SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,

        TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
        TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,

TLS_DHE_DSS_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,

TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_128_CBC_
SHA,

TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC
_SHA,

TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_256_CBC
_SHA,

TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,


TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
,
```

```
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,

TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_anon_WITH_AES_128_CBC_SHA,

        TLS_ECDH_anon_WITH_AES_256_CBC_SHA"
    keystore-type="PKCS11"/>
</connector>
```

cipher-suite 属性には、読みやすくするために改行が挿入されている点に注意してください。

バグの報告

第5章 管理インターフェースのセキュア化

一般的な開発シナリオでは、セキュア化されていない JBoss EAP 6 を管理インターフェースで実行し、迅速に設定を変更できるようにします。

実稼働に向けた開発段階では、少なくとも以下の方法で管理インターフェースをセキュアにします。

- 「JBoss EAP 6 が使用するネットワークインターフェースの指定」
- 「JBoss EAP 6 で動作可能なネットワークファイアウォールの設定」

また、デフォルトのサイレントローカル認証モードを使用すると、ローカルクライアント（サーバーマシン上）はユーザー名やパスワードなしに管理 CLI に接続できます。これは、ローカルユーザーと管理 CLI スクリプトで便利です。これを無効にするには、「デフォルトセキュリティーレルムからのサイレント認証の削除」 を参照してください。

バグの報告

5.1. デフォルトのユーザーセキュリティー設定

はじめに

JBoss EAP 6 のすべての管理インターフェースはデフォルトで保護されます。このセキュリティーには、2 つの形式があります。

- ローカルインターフェースは、ローカルクライアントとサーバーとの間の SASL コントラクトによって保護されます。このセキュリティーメカニズムは、ローカルファイルシステムにアクセスできるクライアントの機能を基にしています。これは、ローカルファイルシステムへのアクセスにより、クライアントがユーザーを追加できるか、その他のセキュリティーメカニズムに設定を変更できるためです。これは、ファイルシステムへの物理アクセスが達成されると、他のセキュリティーメカニズムが一杯になるという原則に従います。このメカニズムは 4 つの手順で行われます。

注記

HTTP を使用してローカルホストへ接続する場合でも、HTTP のアクセスはリモートと見なされます。

1. **ローカル SASL メカニズムを用いて認証する要求が含まれるメッセージをクライアントがサーバーに送信します。**

2. **サーバーは 1 度限りのトークンを生成して、固有のファイルに書き込み、そのファイルのフルパスが含まれるメッセージをクライアントに送信します。**

3. **クライアントはファイルよりトークンを読み取り、サーバーへ送信し、ファイルシステムへローカルアクセスできるかを検証します。**

4. **サーバーはトークンを検証し、ファイルを削除します。**

- **ローカル HTTP クライアントを含むリモートクライアントはレルムベースのセキュリティーを使用します。管理インターフェースを使用して JBoss EAP 6 インスタンスをリモートで設定するパーミッションがあるデフォルトのレルムは ManagementRealm です。このレルム（または作成するレルム）にユーザーを追加できるスクリプトが提供されます。ユーザーの追加に関する詳細は、『JBoss EAP 6『管理および設定ガイド』の「ユーザー管理』 」の章を参照してください。『』ユーザーごとに、ユーザー名とハッシュ化されたパスワードはファイルに保存されます。**

管理対象ドメイン

***EAP_HOME*/domain/configuration/mgmt-users.properties**

スタンドアロンサーバー

***EAP_HOME*/standalone/configuration/mgmt-users.properties**

mgmt-users.properties の内容がマスクされますが、このファイルは機密ファイルとして扱う必要があります。ファイル所有者による読み取りおよび書き込みアクセス以外のアクセス権限を付与する 600 のファイルモードに設定することが推奨されます。

バグの報告

5.2. 管理インターフェースの詳細設定の概要

EAP_HOME /domain/configuration/host.xml または*EAP_HOME*/standalone/configuration/standalone.xml の管理インターフェース設定は、ホストコントローラープロセスのバインド先となるネットワークインターフェース、利用可能な管理インターフェースのタイプ、および各インターフェースのユーザーを認証するために使用される認証システムのタイプを制御します。このトピックでは、お使いの環境に適した管理インターフェースの設定方法について説明します。

管理サブシステムは、以下の 4 つの設定可能な子要素 が含まれる <management > 要素で構成されます。セキュリティーレルムとアウトバウンド接続はそれぞれ最初に定義され、管理インターフェースに属性として適用されます。

- <security-realms>
- <outbound-connections>
- <management-interfaces>
- <audit-log>

注記

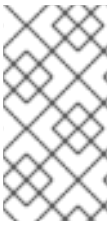
監査ロギングの詳細は、『『Administration and Configuration Guide』』の「『Management Interface Audit Logging』」セクションを参照してください。

セキュリティーレルム

セキュリティーレルムは、管理 API、管理 CLI、または Web ベースの管理コンソールを介して JBoss EAP 6 の管理を許可されているユーザーの認証と承認を行います。

デフォルトのインストールには、ManagementRealm と ApplicationRealm の 2 つの異なるファイルベースのセキュリティーレルムが含まれます。これらのセキュリティーレルムはそれぞれ -users.properties ファイルを使用してユーザーおよびハッシュ化されたパスワードを保存し、-

roles.properties を使用してユーザーとロール間のマッピングを保存します。LDAP 対応のセキュリティーレルムにもサポートが含まれています。



注記

独自のアプリケーションにセキュリティーレルムを使用することもできます。ここで説明するセキュリティーレルムは管理インターフェースに固有のものです。

送信接続

セキュリティーレルムには、LDAP サーバーなどの外部インターフェースに接続するものあります。アウトバウンド接続は、このような接続の確立方法を定義します。事前定義済みの接続タイプ ldap-connection で、LDAP サーバーに接続してクレデンシャルを検証するために必要な属性と任意の属性をすべて設定します。

LDAP 認証の設定方法の詳細は、「管理インターフェースに対する LDAP を使用した認証」 を参照してください。

管理インターフェース

管理インターフェースには JBoss EAP への接続および設定方法に関するプロパティーが含まれます。この情報には、名前付きのネットワークインターフェース、ポート、セキュリティーレルム、およびインターフェースに関するその他の設定可能な情報が含まれます。デフォルトのインストールには、以下の 2 つのインターフェースが含まれます。

- http-interface は、Web ベースの管理コンソールの設定です。
- ネイティブインターフェース は、コマンドライン管理 CLI および REST のような管理 API の設定です。

ホスト管理サブシステムの設定可能要素はそれぞれ関連しています。セキュリティーレルムはアウトバウンド接続を参照し、管理インターフェースはセキュリティーレルムを参照します。

関連情報は 5章 *管理インターフェースのセキュア化* を参照してください。

バグの報告

5.3. HTTP 管理インターフェースの無効化

管理対象ドメインでは、ドメインメンバーサーバーではなく、ドメインコントローラーの HTTP インターフェースへのアクセスのみが必要になります。さらに、実稼働環境では、Web ベースの管理コンソールを完全に無効にできます。

注記

JBoss Operations Network などの他のクライアントも HTTP インターフェースを使用して動作します。これらのサービスを使用し、管理コンソール自体を無効にする場合は、インターフェースを完全に無効にする代わりに、HTTP インターフェースの **console-enabled** 属性を **false** に設定します。

```
/host=master/core-service=management/management-interface=http-interface/:write-attribute(name=console-enabled,value=false)
```

HTTP インターフェースへのアクセスを無効にすると、Web ベースの管理コンソールへのアクセスも無効になるため、HTTP インターフェースを完全に削除して無効化できます。

次の JBoss CLI コマンドを使用すると、再度追加する場合に備えて HTTP インターフェースの現在の内容を読み込むことができます。

例5.1 HTTP インターフェースの設定の読み込み

```
/host=master/core-service=management/management-interface=http-interface/:read-resource(recursive=true,proxies=false,include-runtime=false,include-defaults=true)
{
    "outcome" => "success",
    "result" => {
        "console-enabled" => true,
        "interface" => "management",
        "port" => expression "${jboss.management.http.port:9990}",
        "secure-port" => undefined,
        "security-realm" => "ManagementRealm"
    }
}
```

HTTP インターフェースを削除するには、次のコマンドを実行します。

例5.2 HTTP インターフェースの削除

```
/host=master/core-service=management/management-interface=http-interface/:remove
```

アクセスを再度有効化するには、以下のコマンドを実行し、デフォルト値を使用して HTTP インターフェースを再作成します。

例5.3 HTTP インターフェースの再作成

```
/host=master/core-service=management/management-interface=http-interface:add(console-enabled=true,interface=management,port="${jboss.management.http.port:9990}",security-realm=ManagementRealm)
```

バグの報告

5.4. デフォルトセキュリティーレルムからのサイレント認証の削除

概要

JBoss EAP 6 のデフォルトのインストールには、ローカル管理 CLI ユーザーのサイレント認証メソッドが含まれています。これにより、ローカルユーザーは、ユーザー名またはパスワード認証なしで管理 CLI にアクセスできます。この機能は利便性のためと、管理 CLI スクリプトを実行しているローカルユーザー認証を不要にするために有効になっています。通常、ユーザーがローカル設定にアクセスできれば、独自のユーザー詳細を追加することもできるため便利な機能と見なされます。

ローカルユーザーのサイレント認証の利便性は、セキュリティー管理が長い場合に無効にすることができます。これは、設定ファイルの security-realm セクション内の local 要素を削除することで実現できます。これは、スタンドアロンサーバーインスタンスの standalone.xml、管理対象ドメインの host.xml の両方に適用されます。local 要素の削除は、特定のサーバー設定への影響を把握している場合にのみ考慮する必要があります。

推奨されるサイレント認証の削除方法は、管理 CLI を使用して、以下の例に表示されている local 要素を直接削除することです。

例5.4 security-realmの local 要素の例

```
<security-realms>
    <security-realm name="ManagementRealm">
        <authentication>
            <local default-user="$local"/>
            <properties path="mgmt-users.properties" relative-to="jboss.server.config.dir"/>
        </authentication>
    </security-realm>
    <security-realm name="ApplicationRealm">
        <authentication>
```

```
            <local default-user="$local" allowed-users="*"/>
            <properties path="application-users.properties" relative-
to="jboss.server.config.dir"/>
        </authentication>
        <authorization>
            <properties path="application-roles.properties" relative-
to="jboss.server.config.dir"/>
        </authorization>
    </security-realm>
</security-realms>
```

前提条件

- JBoss EAP 6 インスタンスを起動します。
- 管理 CLI を起動します。

手順5.1 デフォルトセキュリティーレルムからのサイレント認証の削除

- **管理 CLI でのサイレント認証の削除**

  必要に応じて、管理レルムとアプリケーションレルムから **local** 要素を削除します。

  a. 管理レルムから **local** 要素を削除します。

  - スタンドアロンの場合

    ```
    /core-service=management/security-
    realm=ManagementRealm/authentication=local:remove
    ```

  - 管理対象ドメインの場合

    ```
    /host=HOST_NAME/core-service=management/security-
    realm=ManagementRealm/authentication=local:remove
    ```

b.

Application Realm から local 要素を削除します。

- スタンドアロンの場合

```
/core-service=management/security-
realm=ApplicationRealm/authentication=local:remove
```

- 管理対象ドメインの場合

```
/host=HOST_NAME/core-service=management/security-
realm=ApplicationRealm/authentication=local:remove
```

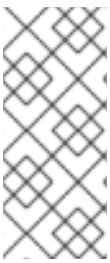
結果

サイレント認証モードは ManagementRealm および ApplicationRealm から削除されます。

バグの報告

5.5. JMX サブシステムへのリモートアクセスの無効化

JMX サブシステムへのリモートアクセスにより、JDK およびアプリケーション管理操作をリモートでトリガーできます。インストールをセキュアにするには、リモーティングコネクターを削除するか、または JMX サブシステムを削除してこの機能を無効にします。管理 CLI コマンドの例は、管理対象ドメインに適しています。スタンドアロンサーバーの場合は、コマンドから /profile=default 接頭辞を削除します。



注記

管理対象ドメインでは、リモーティングコネクターはデフォルトで JMX サブシステムから削除されます。このコマンドは、開発中に追加している場合など、お使いの情報用に提供されます。

例5.5 JMX サブシステムからのリモートコネクターの削除

```
/profile=default/subsystem=jmx/remoting-connector=jmx/:remove
```

例5.6 JMX サブシステムの削除

管理対象ドメインの場合は、各プロファイルに対してこのコマンドを実行します。

```
/profile=default/subsystem=jmx/:remove
```

バグの報告

5.6. 管理インターフェースのセキュリティーレルムの設定

管理インターフェースはセキュリティーレルムを使用して JBoss EAP 6 の認証および設定メカニズムへのアクセスを制御します。セキュリティーレルムは Unix グループに似ています。これは、ユーザーの認証に使用できるユーザー名とパスワードのデータベースです。

デフォルトの管理レルム

デフォルトでは、管理インターフェースは ManagementRealm セキュリティーレルムを使用するように設定されています。ManagementRealm は、ユーザーパスワードの組み合わせを mgmt-users.properties ファイルに保存します。

例5.7 デフォルトの ManagementRealm

```
/host=master/core-service=management/security-realm=ManagementRealm/:read-
resource(recursive=true,proxies=false,include-runtime=false,include-defaults=true)
{
    "outcome" => "success",
    "result" => {
        "map-groups-to-roles" => false,
        "authentication" => {
            "local" => {
                "allowed-users" => undefined,
                "default-user" => "$local"
            },
            "properties" => {
                "path" => "mgmt-users.properties",
                "plain-text" => false,
                "relative-to" => "jboss.domain.config.dir"
            }
        },
        "authorization" => {"properties" => {
            "path" => "mgmt-groups.properties",
```

```
            "relative-to" => "jboss.domain.config.dir"
        }},
        "plug-in" => undefined,
        "server-identity" => undefined
    }
}
```

セキュリティーレルムを新たに作成します。

以下のコマンドは **TestRealm** という新しいセキュリティーレルムを作成し、関連するプロパティーファイルのディレクトリーを設定します。

例5.8 セキュリティーレルムを新たに作成します。

```
/host=master/core-service=management/security-realm=TestRealm/:add
/host=master/core-service=management/security-realm=TestRealm/authentication=properties/:add(path=TestUsers.properties, relative-to=jboss.domain.config.dir)
```

既存のセキュリティードメインを用いたセキュリティーレルム認証の設定

セキュリティードメインを使用して管理インターフェースに対して認証を行うには、以下の手順に従います。

最初にセキュリティーレルムを作成します。次に、管理インターフェースの **security-realm** 属性の値として指定します。

例5.9 HTTP 管理インターフェースに使用するセキュリティーレルムの指定

```
/host=master/core-service=management/management-interface=http-interface/:write-attribute(name=security-realm,value=TestRealm)
```

バグの報告

5.7. HTTPS 用の管理コンソールの設定

HTTPS 経由で のみ 通信するように JBoss EAP 管理コンソールを設定すると、セキュリティーが強化されます。クライアント（Web ブラウザー）と管理コンソール間のネットワークトラフィックはすべて暗号化されるため、中間者攻撃などのセキュリティー攻撃のリスクが軽減されます。JBoss EAP インスタンスを管理するユーザーは、そのインスタンスに対して非特権ユーザーよりも多くの

パーミッションを持ち、HTTPS を使用すると、そのインスタンスの整合性と可用性の保護に役立ちます。

この手順では、JBoss EAP スタンドアロンインスタンスまたはドメインとの暗号化されていない通信を無効にします。これらの通信で使用されるパスワードは、JBoss EAP vault 機能を使用して暗号化され、設定ファイルで使用されるパスワードはマスクされます。

この手順は、スタンドアロン モードと ドメイン モードの設定の両方に適用されます。ドメインモードでは、管理 CLI コマンドの前にホストの名前（例： /host=master ）を付けます。

手順5.2

1. キーストアを作成して管理コンソールをセキュアにします。

注記

管理コンソールは JCEKS 形式のキーストアと互換性がないため、このキーストアは JKS 形式である必要があります。

端末エミュレーターで以下のコマンドを入力します。パラメーター *エイリアス*、*keypass*、*keystore*、*storepass*、および *dname* の場合は、サンプル値を任意の値に置き換えます。

パラメーター *validity* は、キーが有効な日数を指定します。730 の値は 2 年と等しくなります。

```
keytool -genkeypair -alias appserver -storetype jks -keyalg RSA -keysize 2048 -
keypass password1 -keystore EAP_HOME/standalone/configuration/identity.jks -
storepass password1 -dname "CN=appserver,OU=Sales,O=Systems
Inc,L=Raleigh,ST=NC,C=US" -validity 730 -v
```

2. 管理コンソールが HTTPS にバインドされていることを確認します。

- スタンドアロンモード

management-https 設定を追加し、management-http 設定を削除して、管理コンソールがインターフェースの HTTPS にバインドされていることを確認します。

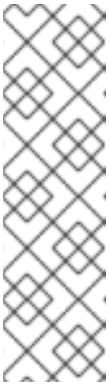
JBoss EAP インスタンスが稼働していることを確認してから、以下の管理 CLI コマンドを入力します。

```
/core-service=management/management-interface=http-interface:write-attribute(name=secure-socket-binding, value=management-https)
```

```
/core-service=management/management-interface=http-interface:undefine-attribute(name=socket-binding)
```

これらのコマンドの出力は以下のようになります。

```
{"outcome" => "success"}
```



注記

この時点で、JBoss EAP ログに以下のエラーメッセージが表示されることがあります。SSL 設定が完了していないため、これは想定されています。

```
JBAS015103: A secure port has been specified for the HTTP interface but no SSL configuration in the realm.
```

- ドメインモード

secure-port を追加し、ポート設定を削除して、management-interface セクション内の socket 要素を変更します。

JBoss EAP インスタンスが稼働していることを確認してから、以下の管理 CLI コマンドを入力します。

```
/host=master/core-service=management/management-interface=http-interface:write-attribute(name=secure-port,value=9443)
/host=master/core-service=management/management-interface=http-interface:undefine-attribute(name=port)
```

注記

この時点で、**JBoss EAP** ログに以下のエラーメッセージが表示されることがあります。**SSL** 設定が完了していないため、これは想定されています。

```
JBAS015103: A secure port has been specified for the HTTP
interface but no SSL configuration in the realm.
```

3. **オプション： カスタムの socket-binding グループ**

カスタムの ***socket-binding*** グループを使用している場合は、***management-https*** バインディングが定義されていることを確認します（デフォルトではポート **9443**にバインドされます）。マスター設定ファイル（例： **standalone.xml** ）を編集して以下と一致させます。

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
    <socket-binding name="management-native" interface="management"
port="${jboss.management.native.port:9999}"/>
    <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
    <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9443}"/>
```

4. **セキュリティーレルムを新たに作成します。**

以下のコマンドを入力して **ManagementRealmHTTPS** という名前の新規セキュリティーレルムを作成します。

```
/host=master/core-service=management/security-realm=ManagementRealmHTTPS/:add
/host=master/core-service=management/security-
realm=ManagementRealmHTTPS/authentication=properties/:add(path=ManagementUsers.pro
perties, relative-to=jboss.domain.config.dir)
```

5. **新しいセキュリティーレルムを使用するように管理インターフェースを設定**

以下のコマンドを実行します。

```
/host=master/core-service=management/management-interface=http-interface/:write-
attribute(name=security-realm,value=ManagementRealmHTTPS)
```

6. **キーストアを使用するように管理コンソールを設定します。**

以下の管理 CLI コマンドを入力します。パラメーター *ファイル* では、*パスワード* と *エイリアス* の値を手順からコピーして 『管理コンソールをセキュアにする』 必要があります。

```
/core-service=management/security-realm=ManagementRealmHTTPS/server-identity=ssl:add(keystore-path=identity.jks,keystore-relative-to=jboss.server.config.dir, keystore-password=password1, alias=appserver)
```

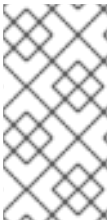
このコマンドからの出力は以下のようになります。

```
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```

7. **JBoss EAP サーバーを再起動します。**

サーバーの再起動時には、起動したサービスの数を示すテキストの直前にログが含まれるはずです。管理コンソールはポート 9443 でリッスンし、手順が成功したことを確認します。

```
14:53:14,720 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015962: Http management interface listening on https://127.0.0.1:9443/management
14:53:14,721 INFO  [org.jboss.as] (Controller Boot Thread) JBAS015952: Admin console listening on https://127.0.0.1:9443
```



注記

セキュリティー上の理由から、キーストアパスワードをマスクすることが推奨されます。これを行う方法は、「パスワード vault システム」 を参照してください。

バグの報告

5.8. 管理インターフェースへの HTTP および HTTPS 接続に DISTINCT インターフェースを使用

管理インターフェースは、HTTP および HTTPS 接続の異なるインターフェースでリッスンできます。1 つのシナリオでは、外部ネットワークの暗号化されたトラフィックをリッスンし、内部ネットワークで暗号化されていないトラフィックを使用します。

secure-interface 属性は、インターフェース属性で指定されたものとは異なるインターフェースを使用する場合は、HTTPS 管理通信のホストのソケットを開くネットワーク インターフェース を指定し

ます。指定されない場合は、interface 属性で指定された インターフェース が使用されます。

secure-port 属性が設定されていない場合は、secure-interface 属性は機能しません。

サーバーが 同じ インターフェースの HTTP および HTTPS トラフィックをリッスンすると、HTTP リスナーによって受信される HTTPS 要求は自動的に HTTPS ポートにリダイレクトされます。HTTP および HTTPS トラフィックに 個別 のインターフェースが使用される場合、HTTP リスナーが HTTPS 要求を受信する際にリダイレクトは実行されません。

以下は、HTTP トラフィックとは異なるインターフェースで HTTPS トラフィックをリッスンするように secure-interface 属性を設定する *EAP_HOME*/domain/configuration/host.xml 設定例になります。

```
<?xml version='1.0' encoding='UTF-8'?>

<host name="master" xmlns="urn:jboss:domain:3.0">

    <management>
        <security-realms>
            <security-realm name="ManagementRealm">
                <authentication>
                    <local default-user="$local" />
                    <properties path="mgmt-users.properties" relative-to="jboss.domain.config.dir"/>
                </authentication>
            </security-realm>
        </security-realms>
        <management-interfaces>
            <native-interface security-realm="ManagementRealm">
                <socket interface="management" port="${jboss.management.native.port:9999}"/>
            </native-interface>
            <http-interface security-realm="ManagementRealm">
                <socket interface="management" port="${jboss.management.http.port:9990}" secure-
port="${jboss.management.https.port:9943}" secure-interface="secure-management"/>
            </http-interface>
        </management-interfaces>
    </management>

    <domain-controller>
        <local/>
        <!-- Alternative remote domain controller configuration with a host and port -->
        <!-- <remote host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9999}"
security-realm="ManagementRealm"/> -->
    </domain-controller>

    <interfaces>
        <interface name="management">
            <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
        </interface>
        <interface name="secure-management">
```

```
            <inet-address value="${jboss.bind.address:10.10.64.1}"/>
        </interface>
    </interfaces>
</host>
```

バグの報告

5.9. 管理インターフェースと CLI での双方向 SSL の使用

クライアント認証 とも呼ばれる 2 方向 SSL 認証は、SSL 証明書を使用してクライアントとサーバーの両方を認証 します。そのため、サーバーの伝えるアイデンティティーだけでなく、クライアントの伝えるアイデンティティーも信頼できます。

このトピックでは、以下の規則を使用します。

HOST1

JBoss サーバーのホスト名。例： jboss.redhat.com

HOST2

クライアントに適した名前。例: myclientこれは必ずしも実際のホスト名ではないことに注意してください。

CA_HOST1

HOST1 証明書に使用する DN（識別名）です。たとえば、cn=jboss,dc=redhat,dc=com です。

CA_HOST2

HOST2 証明書に使用する DN（識別名）です。たとえば、cn=myclient,dc=redhat,dc=com です。

前提条件

- パスワード vault を使用してキーストアとトラストストアパスワードを保存する場合は（推奨）、パスワード vault はすでに作成されている必要があります。「パスワード vault システム」 を参照してください。

手順5.3

1. ストアを生成します。

```
keytool -genkeypair -alias HOST1_alias -keyalg RSA -keysize 1024 -validity 365 -
keystore host1.keystore.jks -dname "CA_HOST1" -keypass secret -storepass secret
```

```
keytool -genkeypair -alias HOST2_alias -keyalg RSA -keysize 1024 -validity 365 -
keystore host2.keystore.jks -dname "CA_HOST2" -keypass secret -storepass secret
```

2. 証明書をエクスポートします。

```
keytool -exportcert  -keystore HOST1.keystore.jks -alias HOST1_alias -keypass secret
-storepass secret -file HOST1.cer
```

```
keytool -exportcert  -keystore HOST2.keystore.jks -alias HOST2_alias -keypass secret
-storepass secret -file HOST2.cer
```

3. 逆のトラストストアに証明書をインポートします。

```
keytool -importcert -keystore HOST1.truststore.jks -storepass secret -alias
HOST2_alias -trustcacerts -file HOST2.cer
```

```
keytool -importcert -keystore HOST2.truststore.jks -storepass secret -alias
HOST1_alias -trustcacerts -file HOST1.cer
```

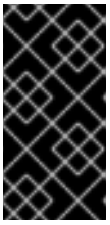
4. インストール（host.xml または standalone.xml）の設定で CertificateRealm を定義し、インターフェースを指定します。

これは、設定ファイルを手動で編集するか（推奨されません）、または以下のコマンドを使用して実行できます。

```
/core-service=management/security-realm=CertificateRealm:add()
```

```
/core-service=management/security-realm=CertificateRealm/server-
identity=ssl:add(keystore-path=/path/to/HOST1.keystore.jks,keystore-
password=secret, alias=HOST1_alias)
```

```
/core-service=management/security-
realm=CertificateRealm/authentication=truststore:add(keystore-
path=/path/to/HOST1.truststore.jks,keystore-password=secret)
```



重要

提供されるコマンドはスタンドアロンモードにのみ適用されます。ドメインモードの場合は、各コマンドの前に /host=master を追加します。

5. native-interface の *security-realm* を新しい証明書レルムに変更します。

```
/host=master/core-service=management/management-interface=native-interface:write-attribute(name=security-realm,value=CertificateRealm)
```

6. *EAP_HOME*/bin/jboss-cli.xml を設定ファイルとして使用する CLI の SSL 設定を追加します。パスワード vault を使用してキーストアおよびトラストストアパスワードを保存するか（推奨）、プレーンテキストに保存します。

- キーストアおよびトラストストアパスワードをパスワード vault に保存するには、以下を実行します。

EAP_HOME/bin/jboss-cli.xml を編集し、SSL 設定を追加します（変数に適切な値を使用）。また、vault 設定を追加して、各値を vault のものに置き換えます。

```
<ssl>
  <vault>
    <vault-option name="KEYSTORE_URL" value="path-to/vault/vault.keystore"/>
    <vault-option name="KEYSTORE_PASSWORD" value="MASK-5WNXs8oEbrs"/>
    <vault-option name="KEYSTORE_ALIAS" value="vault"/>
    <vault-option name="SALT" value="12345678"/>
    <vault-option name="ITERATION_COUNT" value="50"/>
    <vault-option name="ENC_FILE_DIR" value="path-to/jboss-eap/vault/"/>
  </vault>
  <alias>$HOST2alias</alias>
  <key-store>/path/to/HOST2.keystore.jks</key-store>
  <key-store-password>VAULT::VB::cli_pass::1</key-store-password>
  <key-password>VAULT::VB::cli_pass::1</key-password>
  <trust-store>/path/to/HOST2.truststore.jks</trust-store>
  <trust-store-password>VAULT::VB::cli_pass::1</trust-store-password>
  <modify-trust-store>true</modify-trust-store>
</ssl>
```

キーストアおよびトラストストアパスワードをプレーンテキストに保存するには、以下を行います。

EAP_HOME/bin/jboss-cli.xml を編集し、SSL 設定を追加します（変数に適切な値を使用します）。

```
<ssl>
  <alias>$HOST2alias</alias>
  <key-store>/path/to/HOST2.keystore.jks</key-store>
  <key-store-password>secret</key-store-password>
  <trust-store>/path/to/HOST2.truststore.jks</trust-store>
  <trust-store-password>secret</trust-store-password>
  <modify-trust-store>true</modify-trust-store>
</ssl>
```

バグの報告

5.10. JAAS による管理インターフェースのセキュア化

JAAS を使用して管理インターフェースに対して認証するには、以下を実行します。

まず、UsersRoles ログインモジュールでセキュリティードメインを作成します。

```
/subsystem=security/security-domain=UsersLMDomain:add(cache-type=default)
/subsystem=security/security-domain=UsersLMDomain/authentication=classic:add
/subsystem=security/security-domain=UsersLMDomain/authentication=classic/login-
module=UsersRoles:add()
```

次に、JAAS 認証でセキュリティーレルムを作成します。

```
/core-service=management/security-realm=SecurityDomainAuthnRealm:add
/core-service=management/security-
realm=SecurityDomainAuthnRealm/authentication=jaas:add(name=UsersLMDomain)
```

属性 assign-groups は、セキュリティーレルム内のグループ割り当てに、セキュリティードメインからロードされたユーザーメンバーシップ情報を使用するかどうかを決定します。true に設定すると、このグループ割り当ては RBAC(Role-Based Access Control)に使用されます。

この CLI コマンドでは assign-groups 属性を true に設定できます。

```
/core-service=management/security-realm=ManagementRealm/authentication=jaas:write-attribute(name=assign-groups,value=true)
```

バグの報告

5.11. LDAP

5.11.1. LDAP について

LDAP (Lightweight Directory Access Protocol) はネットワーク全体でディレクトリー情報を格納および分散するプロトコルです。このディレクトリー情報には、ユーザー、ハードウェアデバイス、アクセスロール、制限に関する情報などが含まれます。

LDAP の一般的な実装には OpenLDAP、Microsoft Active Directory、IBM Tivoli Directory Server、Oracle Internet Directory などがあります。

JBoss EAP 6 には、Web アプリケーションや EJB アプリケーションの認証承認権限として LDAP サーバーを使用できるようにする複数の認証および承認モジュールがあります。

バグの報告

5.11.2. 管理インターフェースに対する LDAP を使用した認証

管理コンソール、管理 CLI、管理 API の認証ソースとして LDAP ディレクトリーサーバーを使用するには、以下の手順を実行する必要があります。

1. LDAP サーバーへアウトバウンド接続を作成します。
2. LDAP 対応のセキュリティーレルムを作成します。
3. 管理インターフェースの新規セキュリティードメインを参照します。

LDAP サーバーへの送信接続の作成

LDAP 送信接続には、以下の属性を使用することができます。

表5.1 LDAP 送信接続の属性

属性	必須	説明
url	はい	ディレクトリーサーバーの URL アドレス。
search-dn	いいえ	検索の実行が許可されているユーザーの完全識別名 (DN)。
search-credentials	いいえ	検索を実行する権限のあるユーザーのパスワード。
initial-context-factory	いいえ	接続を確立するときに使用する初期コンテキストファクトリー。デフォルトは **com.sun.jndi.ldap.LdapCtxFactory** です。
security-realm	いいえ	接続の確立時に使用する設定済みの **SSLContext** を取得するために参照するセキュリティーレルム。

例5.10 LDAP 送信接続の追加

この例では、以下のプロパティーセットを使用して送信接続を追加します。

- **Search DN: cn=search,dc=acme,dc=com**
- **認証情報の検索： myPass**
- **URL: ldap://127.0.0.1:389**

最初のコマンドによってセキュリティーレルムが追加されます。

```
/host=master/core-service=management/security-realm=ldap_security_realm:add
```

2 つ目のコマンドによって、LDAP 接続が追加されます。

```
/host=master/core-service=management/ldap-connection=ldap_connection/:add(search-
credential=myPass,url=ldap://127.0.0.1:389,search-dn="cn=search,dc=acme,dc=com")
```

LDAP 対応セキュリティーレルムの作成

管理インターフェースは、デフォルトで設定されたプロパティーファイルベースのセキュリティーレルムの代わりに LDAP サーバーに対して認証できます。LDAP オーセンティケーターは、最初にリモートディレクトリーサーバーへ接続を確立します。その後、ユーザーが認証システムに渡すユーザー名を使用して検索を実行し、LDAP レコードの完全修飾識別名 (DN) を探します。ユーザーの DN を認証情報として使用し、ユーザーが指定したパスワードと、新しい接続が確立されます。LDAP サーバーへの認証に成功すると、DN が有効であることが検証されます。

LDAP セキュリティーレルムは次の設定属性を使用します。

connection

LDAP ディレクトリーへの接続に使用する *outbound-connections* で定義された接続の名前。

advanced-filter

提供されたユーザー ID に基づいてユーザーを検索するために使用される、完全に定義されたフィルター。フィルターには、{0} 形式の変数が含まれている必要があります。これは、ユーザーが指定したユーザー名で後ほど置き換えられます。

base-dn

ユーザー検索を開始するためのコンテキストの識別名

再帰

検索が LDAP ディレクトリーツリー全体で再帰的であるか、指定したコンテキストのみを検索するか。デフォルトは false です。

user-dn

識別名を保持するユーザーの属性。この後、ユーザー完了時の認証テストに使用されます。デフォルトは 5 です。

username-attribute

ユーザーを検索する属性の名前。このフィルターは、ユーザーが入力したユーザー名が指定した属性と一致する単純な検索を実行します。

allow-empty-passwords

この属性は、空のパスワードを許可するかどうかを決定します。この属性のデフォルト値は **false** です。

username-filter または advanced-filter のいずれかを指定する必要があります。

advanced-filter 属性には、標準の LDAP 構文にフィルタークエリーが含まれます。以下に例を示します。

```
(&(sAMAccountName={0})(memberOf=cn=admin,cn=users,dc=acme,dc=com))
```

例5.11 LDAP 対応のセキュリティーレルムを示す XML

この例には、以下のパラメーターを使用します。

- connection - ldap_connection
- base-dn - cn=users,dc=acme,dc=com.
- username-filter - attribute="sambaAccountName"

```
<security-realm name="ldap_security_realm">
  <authentication>
    <ldap connection="ldap_connection" base-dn="cn=users,dc=acme,dc=com">
      <username-filter attribute="sambaAccountName" />
    </ldap>
  </authentication>
</security-realm>
```

警告

空の LDAP パスワードを許可しないようにすることが重要になります。ご使用の環境で具体的に空の LDAP パスワードを許可したい場合を除き、深刻なセキュリティー上の問題となります。

EAP 6.1 には CVE-2012-5629 のパッチが含まれており、LDAP ログインモジュールの allowEmptyPasswords オプションが設定されていない場合は、そのオプションを false に設定します。以前のバージョンでは、このオプションを手動で設定する必要があります。

例5.12 LDAP セキュリティーレルムの追加

以下のコマンドは、 LDAP 認証をセキュリティーレルムに追加し、ドメインの master という名前のホストに対して属性を設定します。

```
/host=master/core-service=management/security-
realm=ldap_security_realm/authentication=ldap:add(base-dn="DC=mycompany,DC=org",
recursive=true, username-attribute="MyAccountName", connection="ldap_connection")
```

管理インターフェースへの新規セキュリティーレルムの適用

セキュリティーレルムの作成後、管理インターフェースの設定でそのレルムを参照する必要があります。管理インターフェースは HTTP ダイジェスト認証にセキュリティーレルムを使用します。

例5.13 HTTP インターフェースへのセキュリティーレルムの適用

この設定が有効になり、ホストコントローラーを再起動した後、Web ベースの管理コンソールは LDAP を使用してユーザーの認証を行います。

```
/host=master/core-service=management/management-interface=http-interface/:write-
attribute(name=security-realm,value=ldap_security_realm)
```

例5.14 セキュリティーレルムのネイティブインターフェースへの適用

以下のコマンドを使用して同じ設定をネイティブインターフェースに適用します。

```
/host=master/core-service=management/management-interface=native-interface/:write-attribute(name=security-realm,value=ldap_security_realm)
```

バグの報告

5.11.3. 管理インターフェースおよび CLI での 2 方向 SSL によるアウトバウンド LDAP の使用

JBoss EAP 6 は、管理インターフェースおよび CLI で認証に 2 方向 SSL を使用して LDAP サーバーへのアウトバウンド接続を使用するように設定できます。

前提条件

- LDAP 対応のセキュリティーレルムを作成する必要があります。セキュリティーレルムの作成に関する詳細は、「管理インターフェースに対する LDAP を使用した認証」 を参照してください。

手順5.4 2 方向 SSL を使用したアウトバウンド LDAP の設定

1. セキュリティーレルムキーストアおよびトラストストアを設定します。セキュリティーレルムには、JBoss EAP 6 サーバーが LDAP サーバーに対して認証するために使用するキーで設定されたキーストアが含まれている必要があります。セキュリティーレルムには、LDAP サーバーの証明書で設定されたトラストストアを含める必要があります。キーストアおよびトラストストアの設定方法については、「管理インターフェースと CLI での双方向 SSL の使用」を参照してください。

2. 設定されたセキュリティーレルムを指定して、アウトバウンド接続を LDAP サーバーに追加します。

```
/core-service=management/ldap-connection=LocalLdap:add(url="ldaps://LDAP_HOST:LDAP_PORT")

/core-service=management/ldap-connection=LocalLdap:write-attribute(name=security-realm,value="LdapSSLRealm")
```

3. 「管理インターフェースに対する LDAP を使用した認証」 に示されるように、セキュリティーレルムと管理インターフェース内に LDAP 認証を設定します。

バグの報告

第6章 ロールベースアクセス制御を用いた管理インターフェースのセキュア化

6.1. ロールベースアクセス制御 (RBAC)

ロールベースのアクセス制御 (RBAC) は管理ユーザーにパーミッションを指定するメカニズムです。JBoss EAP 6 サーバーを管理する責任を複数のユーザーに共有でき、各ユーザーは無制限のアクセスを必要としません。JBoss EAP 6 では、管理ユーザーの「職務の分離」を提供することで、不要な権限を付与せずに組織が個人やグループ間で責任を簡単に分散できます。これにより、設定、デプロイメント、および管理の柔軟性を維持しながらサーバーやデータのセキュリティーを可能な限り最大限にします。

JBoss EAP 6 のロールベースアクセス制御は、ロールパーミッションと制約の組み合わせにより動作します。

それぞれに異なる固定パーミッションを持つ 7 つの事前定義されたロールが提供されます。事前定義されたロールは Monitor、Operator、Maintainer、Deployer、Auditor、Administrator、および SuperUser です。各管理ユーザーには 1 つまたは複数のロールが割り当てられ、ユーザーがサーバーの管理時に許可されるものを指定します。

重要

プロバイダーを rbac に変更する前に、RBAC ロールのいずれかにマップするユーザーがあるようにしてください（Administrator または SuperUser ロールのいずれかが望ましい）。シャットダウンし、XML 設定を編集しない限り、インストールは管理できません。

JBoss EAP 6 に同梱される標準 XML 設定のいずれかで開始した場合、$local ユーザーは SuperUser ロールにマッピングされ、local 認証スキームが有効になります。これにより、JBoss EAP 6 プロセスと同じシステムで CLI を実行するユーザーに、完全な管理パーミッションが付与されます。リモート CLI ユーザーおよび Web ベースの管理コンソールユーザーにはパーミッションがありません。

プロバイダーを rbac に切り替える前に、$local 以外のユーザーをマッピングします。

バグの報告

6.2. 管理コンソールおよび CLI でのロールベースアクセス制御

ロールベースアクセス制御 (RBAC) が有効になっている場合、ユーザーに割り当てられたロールによって、ユーザーがアクセスできるリソースやリソースの属性を用いて実行できる操作が決定されます。

管理コンソール

管理コンソールでは、ユーザーに割り当てられたロールのパーミッションによっては、制御およびビューの一部が無効化 (灰色で表示) されたり、全く表示されないことがあります。

リソース属性の読み取りパーミッションがない場合、その属性はコンソールで空白で表示されます。たとえば、ほとんどのロールはデータソースのユーザー名およびパスワードフィールドを読み取ることができません。

リソース属性への書き込みパーミッションがない場合、その属性はリソースの編集フォームで無効化（表示）されます。リソースに書き込み権限がない場合は、リソースの編集ボタンは表示されません。

ユーザーがリソースまたは属性にアクセスするパーミッションを持たない場合（そのロールの「アドレス不可能」の場合）、そのユーザーのコンソールには表示されません。この例の 1 つは、アクセス制御システム自体で、デフォルトではいくつかのロールでのみ表示されます。

管理 CLI または API

RBAC が有効である場合、API での 管理 CLI や管理 API の動作が若干異なります。

読み取りできないリソースや属性は、結果からフィルター処理されます。フィルターされた項目がロールによってアドレス可能である場合、それらの名前は結果の filtered-attributes セクションに response-headers として一覧表示されます。ロールがリソースや属性をアドレス指定できない場合はリストされません。

アドレス指定できないリソースにアクセスしようとすると、リソースが見つかりません。

適切な書き込みまたは読み取りパーミッションがないものの、アドレス指定可能なリソースの読み取りまたは読み取りを試みると、Permission Denied エラーが返されます。

バグの報告

6.3. サポートされる認証スキーム

ロールベースアクセス制御は、JBoss EAP 6 に含まれる標準の認証プロバイダーと動作します。標準の認証プロバイダーは、ユーザー 名/パスワード、クライアント証明書、および ローカルユーザー です。

ユーザー名/パスワード

ユーザーは、mgmt-users.properties ファイルまたは LDAP サーバーに対して検証されるユーザー名とパスワードの組み合わせを使用して認証されます。

クライアント証明書

トラストストアを使用します。

Local User

jboss-cli.sh は同じマシンで実行されているサーバーの場合、自動的に Local User として認証されます。デフォルトでは、Local User は SuperUser グループのメンバーです。

使用されるプロバイダーに関係なく、JBoss EAP はロールをユーザーに割り当てます。ただし、mgmt-users.properties ファイルまたは LDAP サーバーで認証する場合、これらのシステムはユーザーグループ情報を提供できます。JBoss EAP はこの情報を使用してユーザーにロールを割り当てることもできます。

バグの報告

6.4. 標準のロール

JBoss EAP 6 は、Monitor、Operator、Maintainer、Deployer、Auditor、Administrator、および SuperUser の 7 つの事前定義されたユーザーロールを提供します。これらのロールにはそれぞれ異なるパーミッションセットがあり、特定のユースケース用に設計されています。Monitor、Operator、Maintainer、Administrator、および SuperUser ロールは相互にビルドされ、各パーミッションは直前のものよりも多くのパーミッションを持ちます。Auditor ロールと Deployer ロールは Monitor ロールと Maintainer ロールと似ていますが、特別なパーミッションと制限が追加されています。

Monitor

Monitor ロールのユーザーは最も少ないパーミッションを持ち、現在の設定とサーバーの状態のみを読み取りできます。このロールは、サーバーのパフォーマンスを追跡および報告する必要があるユーザーを対象としています。

Monitor はサーバー設定を変更したり、機密データおよび操作にアクセスしたりできません。

Operator

Operator ロールは、サーバーのランタイム状態を変更する機能を追加して Monitor ロールを拡張します。つまり、Operator はサーバーをリロードおよびシャットダウンし、JMS 宛先を一時停止および再開できます。Operator ロールは、必要時にサーバーを確実にシャットダウンおよび再起動できるように、アプリケーションサーバーの物理または仮想ホストを行うユーザーに適しています。

Operator はサーバー設定を変更したり、機密データおよび操作にアクセスしたりできません。

Maintainer

Maintainer ロールは、ランタイム状態と機密データおよび操作を除くすべての設定を表示および変更できます。Maintainer ロールは機密データおよび操作にアクセスできない汎用のロールです。Maintainer ロールを使用すると、パスワードやその他の機密情報へのアクセスを許可せずに、サーバー管理をほぼ完全なアクセス権限として付与できます。

Maintainer は機密データまたは操作へアクセスできません。

Administrator

Administrator ロールは、監査ロギングシステムを除くサーバーのすべてのリソースおよび操作に無制限にアクセスできます。Administrator ロールは機密のデータおよび操作にアクセスできます。また、このロールはアクセス制御システムも設定できます。Administrator ロールは、機密データを扱う場合やユーザーやロールを設定する場合のみ必要です。

Administrator は監査ロギングシステムへアクセスできず、Administrator 自身を Auditor または SuperUser ロールへ変更できません。

SuperUser

SuperUser ロールには制限がなく、監査ロギングシステムを含むサーバーのすべてのリソースおよび操作に完全アクセスできます。このロールは、以前のバージョンの JBoss EAP 6（6.0 および 6.1）の管理者ユーザーと同等です。RBAC が無効の場合、すべての管理ユーザーは SuperUser ロールと同等のパーミッションを持ちます。

Deployer

Deployer ロールは Monitor と同じパーミッションを持ちますが、デプロイメントの設定や状態を変更でき、アプリケーションリソースとして有効になっている他のリソースタイプも変更できます。

Auditor

Auditor ロールは Monitor ロールのパーミッションをすべて持ちます。また、機密データを表示（ただし変更できません）、監査ロギングシステムに完全アクセスできます。Auditor ロールは

SuperUser 以外のロールで、監査ロギングシステムにアクセスできます。

監査人は機密データやリソースを変更できません。読み込みアクセスのみが許可されています。

バグの報告

6.5. ロールパーミッション

各ロールの権限は、各ロールの権限で定義されます。すべてのロールにすべてのパーミッションがあるわけではありません。SuperUser にすべてのパーミッションがあり、Monitor のパーミッションは最も少なくなります。

各パーミッションは、リソースの単一のカテゴリーへの読み取りアクセスや書き込みアクセスを付与できます。

カテゴリーは、ランタイム状態、サーバー設定、機密データ、監査ログ、およびアクセス制御システムです。

表6.1「ロールパーミッション」 は、各ロールのパーミッションをまとめています。

表6.1 ロールパーミッション

	Monitor	Operator	Maintainer	Deployer	Auditor	Administrator	SuperUser
設定と状態の読み取り	X	X	X	X	X	X	X
機密データの読み取り [2]					X	X	X
機密データの変更 [2]						X	X
監査ログの読み取り/変更					X		X
ランタイム状態の変更		X	X	O[1]		X	X

永続化設定の変更			X	○[1]		X	X
アクセス制御の読み取り/変更						X	X

[1] パーミッションはアプリケーションリソースに制限されます。

[2] 機密性制約 (Sensitivity Constraint) を使用して「機密データ」として考慮されるリソースを設定します。

バグの報告

6.6. 制約

制約とは、指定のリソースリストに対するアクセス制御設定の名前付きセットです。RBAC システムは制約とロールパーミッションの組み合わせを使用して、特定ユーザーが管理操作を実行できるかどうかを決定します。

制約は、アプリケーション、機密性、および Vault 式の 3 つに分類されます。

アプリケーション制約

アプリケーション制約は、Deployer ロールのユーザーがアクセス可能なリソースおよび属性のセットを定義します。デフォルトでは、有効になっている唯一のアプリケーション制約は、デプロイメント、デプロイメントオーバーレイが含まれるコアです。アプリケーション制約には、データソース、ロギング、メール、メッセージング、ネーミング、resource-adapters、およびセキュリティーも含まれます（デフォルトでは有効になっていません）。これらの制約により、Deployer ユーザーはアプリケーションをデプロイできるだけでなく、これらのアプリケーションが必要とするリソースを設定および維持できます。

アプリケーション制約の設定は、Management API(/core-service=management/access=authorization/constraint=application-classification)にあります。

機密性制約

機密性制約は、「機密」とみなされるリソースのセットを定義します。通常、機密リソースとは、パスワードなどの秘密のリソースや、ネットワーキング、JVM 設定、システムプロパティーなどのサーバーの操作に重大な影響を与えるリソースのことです。アクセス制御システム自体も機密であると見なされます。

機密リソースへの書き込みが許可されるロールは Administrator と SuperUser ロールのみです。Auditor ロールは機密リソースの読み取りのみが許可されます。その他のロールは機密リソースにアクセスできません。

機密性制約の設定は、管理 API(/core-service=management/access=authorization/constraint=sensitivity-classification)にあります。

vault 式制約

Vault 式制約は、vault 式の読み取りまたは書き込みが機密操作として考慮されるかどうかを定義します。デフォルトでは、vault 式の読み書きは機密操作です。

Vault 式制約の設定は、管理 API(/core-service=management/access=authorization/constraint=vault-expression)にあります。

制約は管理コンソールでは設定できません。

バグの報告

6.7. JMX およびロールベースアクセス制御

ロールベースのアクセス制御は、3 つの方法で JMX に適用されます。

1. JBoss EAP 6 の管理 API は JMX 管理 Bean として公開されます。これらの管理 Bean は「コア mbeans」と呼ばれ、コア mbeans へのアクセスは基盤の管理 API と全く同じように制御およびフィルターされます。

2. JMX サブシステムは、書き込みパーミッションが「機密」であるように設定されています。そのため、Administrator および SuperUser ロールのユーザーのみがそのサブシステムを変更できます。Auditor ロールのユーザーは、このサブシステムの設定を読み取ることもできます。

3. デフォルトでは、デプロイされたアプリケーションおよびサービスによって登録された管理 Bean (コアでない MBean) へすべての管理ユーザーがアクセスできますが、Maintainer、Operator、Administrator、および SuperUser ロールのユーザーのみが書き込み可能です。

注記

ユーザーは、jconsole などの JMX クライアントから JMX 通知を受信できます。この機能は、ローカルの JMX 接続に制限されます。JMX クライアントは、アプリケーションサーバーと同じ JVM 内に接続するか、同じマシン上で接続し、Attach エージェントを使用してアプリケーションサーバーに接続します（jconsole と同様に）。MBean 登録/登録解除および属性値の変更に関する JMX 通知が、jboss.as ドメインおよび jboss.as.expr ドメインの MBean に対しても生成されるようになりました。

バグの報告

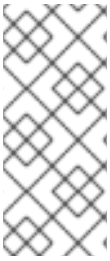
6.8. ロールベースアクセス制御の設定

6.8.1. RBAC 設定タスクの概要

RBAC が有効になっている場合、Administration および SuperUser ロールのユーザーのみがアクセス制御システムを表示し、変更を追加できます。

管理コンソールは、以下の一般的な RBAC タスクに対してインターフェースを提供します。

- 各ユーザーへ割り当てられた (または除外された) ロールの表示および設定
- 各グループへ割り当てられた (または除外された) ロールの表示および設定。
- ロールごとのグループおよびユーザーメンバーシップの表示。
- ロールごとのデフォルトメンバーシップの設定。
- スコープ指定されたロールの作成。



注記

管理コンソールを使用して RBAC を有効または無効にすることは できません。これらの設定は UI で公開されません。コマンドラインインターフェースを使用して、これらのタスクを実行します。

管理 CLI は、完全なアクセス制御システムへのアクセスを提供します。つまり、管理コンソールで実行できるすべてのことを実行できますが、管理コンソールでは実行できない管理 CLI では多くの追加タスクを実行することができます。

管理 CLI で実行できる追加のタスクは次のとおりです。

- RBAC の有効化および無効化。
- パーミッション組み合わせポリシーの変更
- アプリケーションリソースおよびリソース機密性の制約の設定

バグの報告

6.8.2. ロールベースのアクセス制御の有効化

デフォルトでは、Role-Based Access Control (RBAC) システムが無効になっています。有効にするには、provider 属性を simple から rbac に変更します。これは、管理 CLI を使用するか、サーバーがオフラインの場合はサーバー設定 XML ファイルを編集して実行できます。稼働中のサーバーで RBAC を無効化または有効化した場合、サーバー設定をリロードして変更を反映する必要があります。

有効にすると、無効にできるのは Administrator または SuperUser ロールのユーザーのみです。デフォルトでは、サーバーと同じマシンで実行される場合、管理 CLI は SuperUser ロールとして実行されます。

手順6.1 RBAC の有効化

- 管理 CLI で RBAC を有効にするには、アクセス承認リソースの write-attribute 操作を使用して provider 属性を rbac に設定します。

-

/core-service=management/access=authorization:write-attribute(name=provider, value=rbac)

```
[standalone@localhost:9999 /] /core-service=management/access=authorization:write-attribute(name=provider, value=rbac)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
[standalone@localhost:9999 /] /:reload
{
    "outcome" => "success",
    "result" => undefined
}
```

手順6.2 RBAC の無効化

- **管理 CLI で RBAC を無効にするには、アクセス承認リソースの write-attribute 操作を使用して provider 属性を simple に設定します。**

/core-service=management/access=authorization:write-attribute(name=provider, value=simple)

```
[standalone@localhost:9999 /] /core-service=management/access=authorization:write-attribute(name=provider, value=simple)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
[standalone@localhost:9999 /] /:reload
{
    "outcome" => "success",
    "result" => undefined
}
```

サーバーがオフラインの場合、XML 設定を編集して RBAC を有効または無効にすることができます。これには、管理要素の provider 要素の access-control 属性を編集します。この値を rbac に設定して有効にし、simple で無効にします。

```
<management>

    <access-control provider="rbac">
        <role-mapping>
```

```
            <role name="SuperUser">
              <include>
                <user name="$local"/>
              </include>
            </role>
          </role-mapping>
        </access-control>

      </management>
```

バグの報告

6.8.3. Permission Combination Policy の変更

Permission Combination Policy は、ユーザーが複数のロールが割り当てられているかどうかの判断方法を決定します。permissive または reject に設定できます。デフォルトは permissive です。

permissive に設定すると、アクションを許可するユーザーにロールが割り当てられていると、そのアクションが許可されます。

rejecting に設定すると、複数のロールがユーザーに割り当てられている場合、アクションは許可されません。つまり、ポリシーが rejecting に設定されていると、各ユーザーには 1 つのロールのみを割り当てる必要があります。ポリシーが rejecting に設定されている場合、複数のロールを持つユーザーは管理コンソールまたは管理 CLI を使用できません。

Permission Combination Policy は、permission-combination-policy 属性を permissive または reject のいずれかに設定して設定し ます。これは、管理 CLI を使用するか、サーバーがオフラインの場合はサーバー設定 XML ファイルを編集して実行できます。

手順6.3 パーミッション組み合わせポリシーの設定

- **アクセス承認リソースの write-attribute 操作を使用して、permission-combination-policy 属性を必要なポリシー名に設定します。**

  ```
  /core-service=management/access=authorization:write-attribute(name=permission-combination-policy, value=POLICYNAME)
  ```

  有効なポリシー名は rejecting と permissive です。

  ```
  [standalone@localhost:9999 /] /core-service=management/access=authorization:write-attribute(name=permission-combination-policy, value=rejecting)
  ```

```
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

サーバーがオフラインの場合、XML 設定を編集してパーミッションの組み合わせポリシー (permission combination policy) 値を変更できます。これには、access-control 要素の permission-combination-policy 属性を編集します。

```
<access-control provider="rbac" permission-combination-policy="rejecting">
  <role-mapping>
    <role name="SuperUser">
      <include>
        <user name="$local"/>
      </include>
    </role>
  </role-mapping>
</access-control>
```

バグの報告

6.9. ロールの管理

6.9.1. ロールメンバーシップ

RBAC (Role-Based Access Control) を有効にすると、管理ユーザーが許可されている内容は、ユーザーが割り当てられているロールによって決まります。JBoss EAP 6 は、ユーザーおよびグループメンバーシップの両方に基づいて包含と除外のシステムを使用して、ユーザーが属するロールを決定します。

以下の場合に、ロールがユーザーに割り当てられたとみなされます。

1. ユーザーが以下のいずれかである場合。
   - ロールに含まれるユーザーとしてリストに記載されている。
   - ロールに含まれるとリストに記載されたグループのメンバーである。
2. ユーザーが以下のいずれかに該当しない場合。

- ロールから除外されるユーザーとしてリストに記載されている。
- ロールから除外される一覧のグループのメンバーです。

除外は包含よりも優先されます。

ユーザーおよびグループに対するロールの include および exclude 設定は、管理コンソールと管理 CLI の両方を使用して設定できます。

SuperUser または Administrator ロールのユーザーのみがこの設定を行えます。

バグの報告

6.9.2. ユーザーロール割り当ての設定

include（含まれる）および exclude（除外）されるユーザーのロールは、管理コンソールおよび管理 CLI で設定できます。このトピックでは、管理コンソールの使用のみを説明します。

この設定を実行できるのは、SuperUser または Administrator ロールのユーザーのみです。

以下の手順で、管理コンソールのユーザーロール設定を確認します。

1. 管理コンソールへログインします。
2. Administration タブをクリックします。
3. Access Control メニューを展開し、Role Assignment を選択します。
4. USERS タブを選択します。

手順6.4 ユーザーの新しいロール割り当ての作成

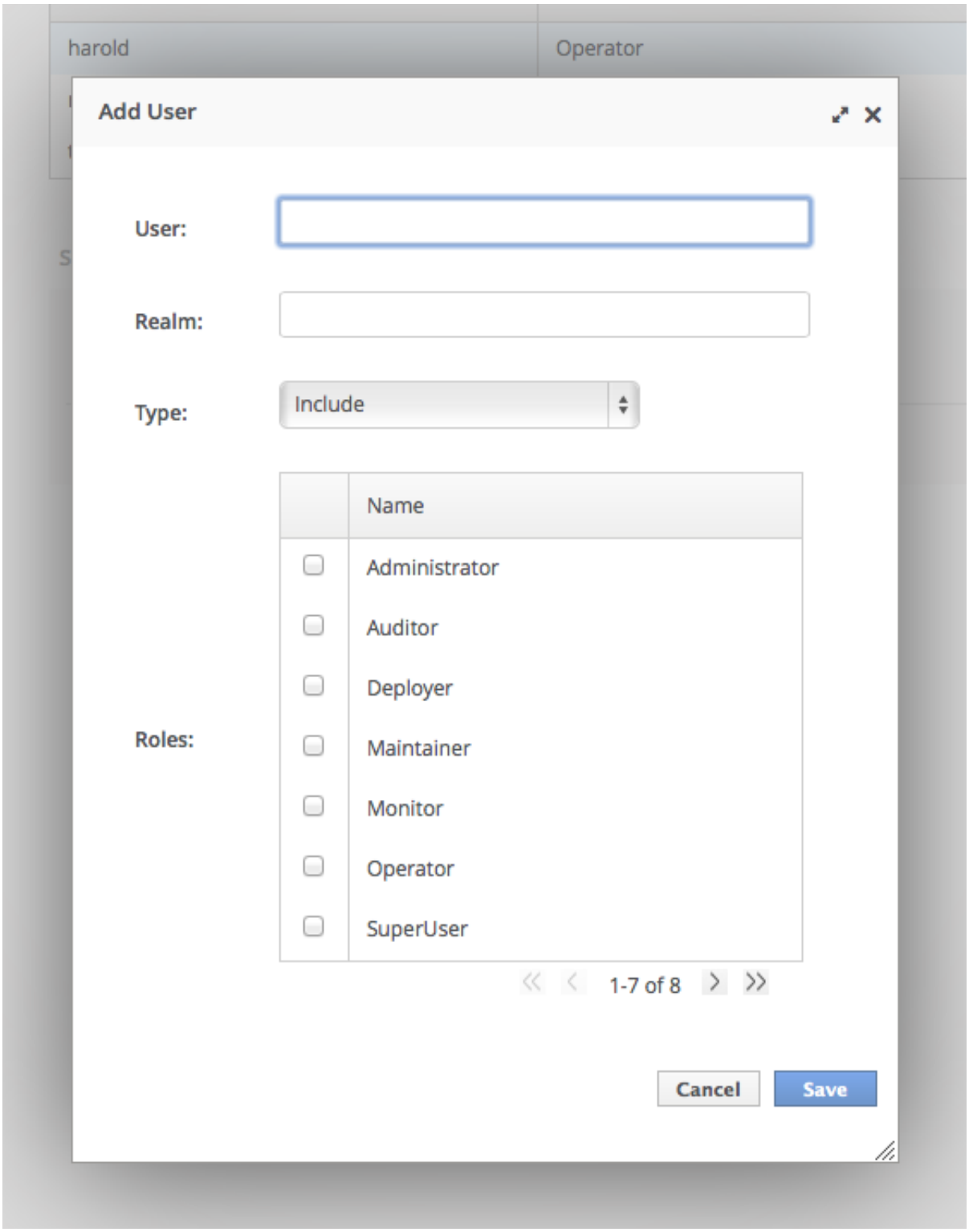
手順6.4 ユーザーの新しいロール割り当ての作成

1. 管理コンソールへログインします。

2. Role Assignment セクションの Users タブに移動します。

3. ユーザー一覧の右上にある Add ボタンをクリックします。Add User ダイアログが表示されます。

図6.1 Add User ダイアログ



4. **ユーザー名を指定し、任意でレルムを指定します。**

5. **Type メニューを include または exclude に設定します。**

6. **include または exclude するロールのチェックボックスをクリックします。複数の項目を確認するには、コントロールキー（OSX のコマンドキー）を保持します。**

7. **Save をクリックして終了します。**

成功すると、ユーザーの 追加 ダイアログが閉じ、ユーザーの一覧が更新され、変更内容が反映されます。Failed to save role assignment メッセージが表示されます。

手順6.5 ユーザーロール割り当ての更新

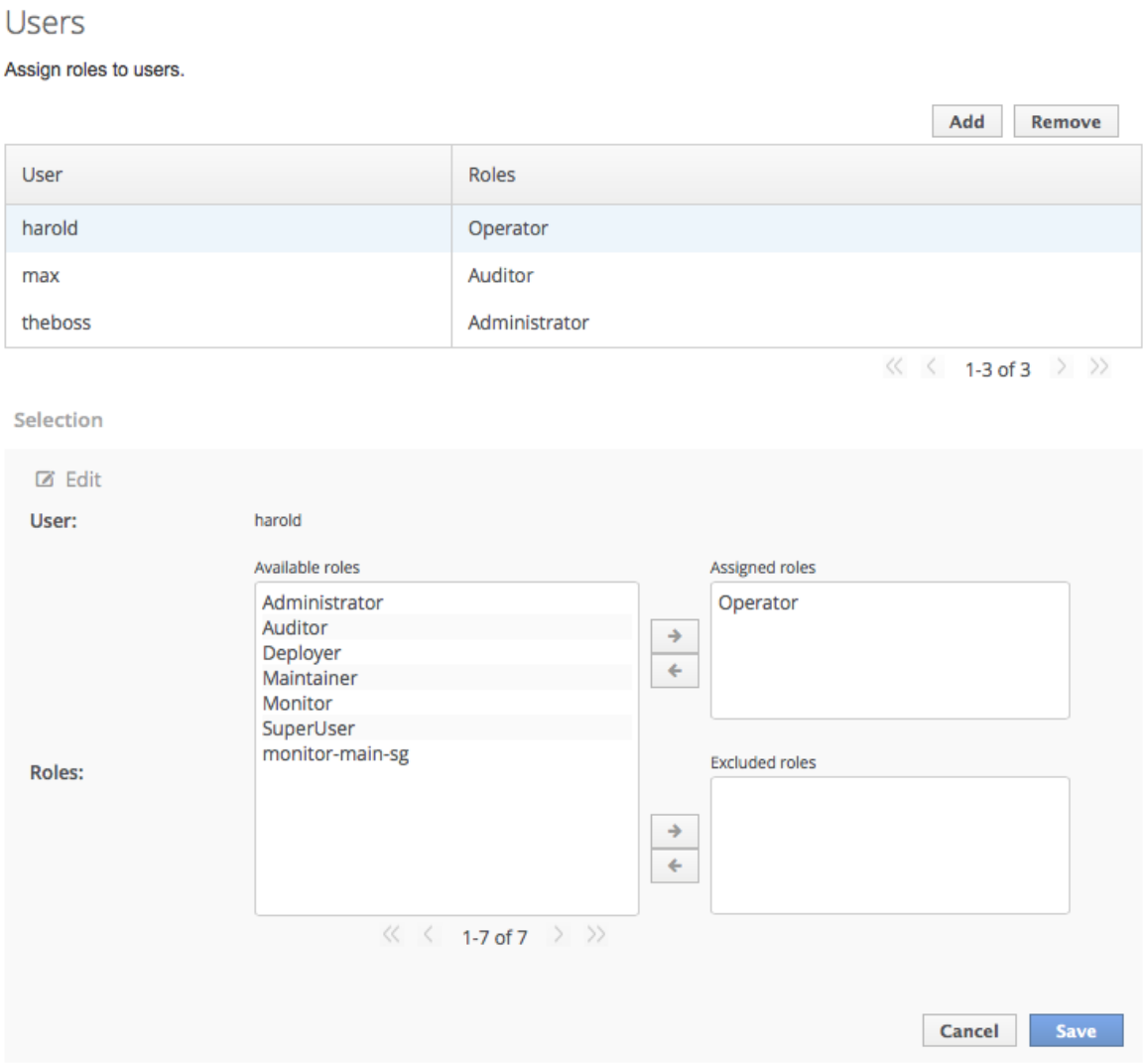
1. **管理コンソールへログインします。**

2. **Role Assignment セクションの Users タブに移動します。**

3. **リストよりユーザーを選択します。**

4. **Edit をクリックします。選択パネルは編集モードになります。**

図6.2 Selection Edit ビュー



ここで、ユーザーに割り当てられたロールや除外されたロールを追加および削除できます。

1. 割り当てられたロールを追加するには、左側の使用可能なロールのリストからロールを選択し、割り当てられたロールリストの横にある右矢印のボタンをクリックします。ロールは利用可能なリストから割り当てられたリストに移動します。

2. 割り当てられたロールを削除するには、割り当てられたロールのリストからロールを選択し、割り当てられたロールリストの横にある左矢印のボタンをクリックします。ロールは割り当てられたリストから利用可能なリストに移動します。

3. 除外されたロールを追加するには、左側の使用可能なロールのリストからロールを選択し、除外されたロールリストの横にある右矢印のボタンをクリックします。ロールは利用可能な一覧から除外されたリストに移動します。

4. 除外されたロールを削除するには、除外されたロールのリストからロールを選択し、除外されたロールリストの横にある左矢印のボタンをクリックします。ロールは除外されたリストから利用可能なリストに移動します。

5. Save をクリックして終了します。

正常に保存されると、編集ビューが閉じられ、ユーザーのリストが更新され、変更内容が反映されます。Failed to save role assignment メッセージが表示されます。

手順6.6 ユーザーのロール割り当ての削除

1. 管理コンソールへログインします。

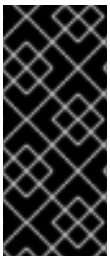
2. Role Assignment セクションの Users タブに移動します。

3. 一覧からユーザーを選択します。

4. Remove をクリックします。Remove Role Assignment 確認プロンプトが表示されます。

5. Confirm をクリックします。

正しく削除されると、ユーザーロール割り当てのリストにユーザーが表示されなくなります。



重要

ロール割り当てのリストからユーザーを削除しても、ユーザーはシステムから削除されず、ユーザーにロールが割り当てられなくなる保証はありません。ロールはグループメンバーシップから依然として割り当てられる可能性があります。

バグの報告

6.9.3. 管理 CLI を用いたユーザーロール割り当ての設定

include（含まれる）および exclude（除外）されるユーザーのロールは、管理コンソールおよび管理 CLI で設定できます。このトピックでは、管理 CLI の使用についてのみ説明します。

ユーザーおよびグループのロールへのマッピングの設定は、/core-service=management/access=authorization 要素として管理 API にあります。role-mapping

SuperUser または Administrator ロールのユーザーのみがこの設定を行えます。

コマンドへのアクセスを容易にするため、管理 CLI で /core-service=management/access=authorization の場所を変更します。

```
[standalone@localhost:9999] cd /core-service=management/access=authorization
```

手順6.7 ロール割り当て設定の表示

1. :read-children-names 操作を使用して、設定されたロールの完全リストを取得します。

```
/core-service=management/access=authorization:read-children-names(child-type=role-mapping)
```

```
[standalone@localhost:9999 access=authorization] :read-children-names(child-type=role-mapping)
{
    "outcome" => "success",
    "result" => [
        "Administrator",
        "Deployer",
        "Maintainer",
        "Monitor",
        "Operator",
        "SuperUser"
    ]
}
```

2. 指定した role-mapping の read-resource 操作を使用して、特定のロールの詳細を取得します。

```
/core-service=management/access=authorization/role-mapping=ROLENAME:read-resource(recursive=true)
```

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Administrator:read-
resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "include-all" => false,
        "exclude" => undefined,
        "include" => {
            "user-theboss" => {
                "name" => "theboss",
                "realm" => undefined,
                "type" => "USER"
            },
            "user-harold" => {
                "name" => "harold",
                "realm" => undefined,
                "type" => "USER"
            },
            "group-SysOps" => {
                "name" => "SysOps",
                "realm" => undefined,
                "type" => "GROUP"
            }
        }
    }
}
[standalone@localhost:9999 access=authorization]
```

手順6.8 新規ロールの追加

この手順では、ロールの role-mapping エントリーを追加する方法を説明します。これは、ロールを設定する前に行う必要があります。

- **add 操作を使用して、新しいロール設定を追加します。**

```
/core-service=management/access=authorization/role-mapping=ROLENAME:add
```

***ROLENAME* は、新しいマッピングに使用するロールの名前です。**

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor:add
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

手順6.9 ロールに include されるユーザーの追加

この手順では、ユーザーをロールの include されたリストに追加する方法を説明します。

ロールの設定が行われていない場合は、そのロールの role-mapping エントリーを最初に実行する必要があります。

- add 操作を使用して、ロールの追加一覧にユーザーエントリーを追加します。

```
/core-service=management/access=authorization/role-mapping=ROLENAME/include=ALIAS:add(name=USERNAME, type=USER)
```

ROLENAME は、設定されるロールの名前です。

ALIAS このマッピングの一意の名前です。Red Hat は、user-*USERNAME* などのエイリアスの命名規則を使用することを推奨します。

USERNAME は、include リストに追加されたユーザーの名前です。

```
 [standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/include=user-max:add(name=max, type=USER)
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

手順6.10 ロールに exclude されるユーザーの追加

この手順では、ロールの除外リストにユーザーを追加する方法を説明します。

ロールの設定が行われていない場合は、そのロールの role-mapping エントリーを最初に実行する必要があります。

- add 操作を使用して、ロールの除外一覧にユーザーエントリーを追加します。

```
/core-service=management/access=authorization/role-mapping=ROLENAME/exclude=ALIAS:add(name=USERNAME, type=USER)
```

ROLENAME は、設定されるロールの名前です。

USERNAME は、除外リストに追加されたユーザーの名前です。

ALIAS このマッピングの一意の名前です。**Red Hat** は、**user-*USERNAME*** などのエイリアスの命名規則を使用することを推奨します。

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/exclude=user-max:add(name=max, type=USER)
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

手順6.11 ユーザーロールの include 設定の削除

この手順では、ロールマッピングからユーザー包含エントリーを削除する方法を説明します。

- **remove 操作を使用してエントリーを削除します。**

```
/core-service=management/access=authorization/role-mapping=ROLENAME/include=ALIAS:remove
```

ROLENAME **は、設定されるロールの名前です。**

ALIAS このマッピングの一意の名前です。**Red Hat** は、**user-*USERNAME*** などのエイリアスの命名規則を使用することを推奨します。

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/include=user-max:remove
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

包含の一覧からユーザーを削除しても、ユーザーはシステムから削除されません。また、ロールがユーザーに割り当てられないようにします。ロールはグループメンバーシップに基づいて依然として割り当てられる可能性があります。

手順6.12 ユーザーロールの exclude 設定の削除

この手順では、ロールマッピングからユーザー除外エントリーを削除する方法を説明します。

- **remove 操作を使用してエントリーを削除します。**

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:remove
```

ROLENAME は、設定されるロールの名前です。

ALIAS このマッピングの一意の名前です。Red Hat は、user-*USERNAME* などのエイリアスの命名規則を使用することを推奨します。

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/exclude=user-
max:remove
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

除外の一覧からユーザーを削除しても、ユーザーはシステムから削除されません。また、そのロールが必ずそのユーザーに割り当てられるようになるわけではありません。依然としてロールはグループメンバーシップに基づいて除外される可能性があります。

バグの報告

6.9.4. ロールとユーザーグループ

mgmt-users.properties ファイルまたは LDAP サーバーを使用して認証されたユーザーは、ユーザーグループのメンバーである可能性があります。ユーザーグループは、1 人以上のユーザーに割り当てることのできる任意のラベルです。

RBAC システムは、所属するユーザーグループに応じて、自動的にロールをユーザーに割り当てるように設定できます。また、グループメンバーシップに基づいてロールからユーザーを除外することもできます。

mgmt-users.properties ファイルを使用する場合、グループ情報は mgmt-groups.properties ファイルに保存されます。LDAP を使用する場合、グループ情報は LDAP サーバーに保存され、LDAP サーバーに対応するものによって維持されます。

バグの報告

6.9.5. グループロール割り当ての設定

ユーザーグループのユーザーのメンバーシップを基に、ロールをユーザーに割り当てできます。

ロールに含まれる、またはロールから除外されるグループは、管理コンソールおよび管理 CLI で設定できます。このトピックでは、管理コンソールの使用のみを説明します。

この設定を実行できるのは、SuperUser または Administrator ロールのユーザーのみです。

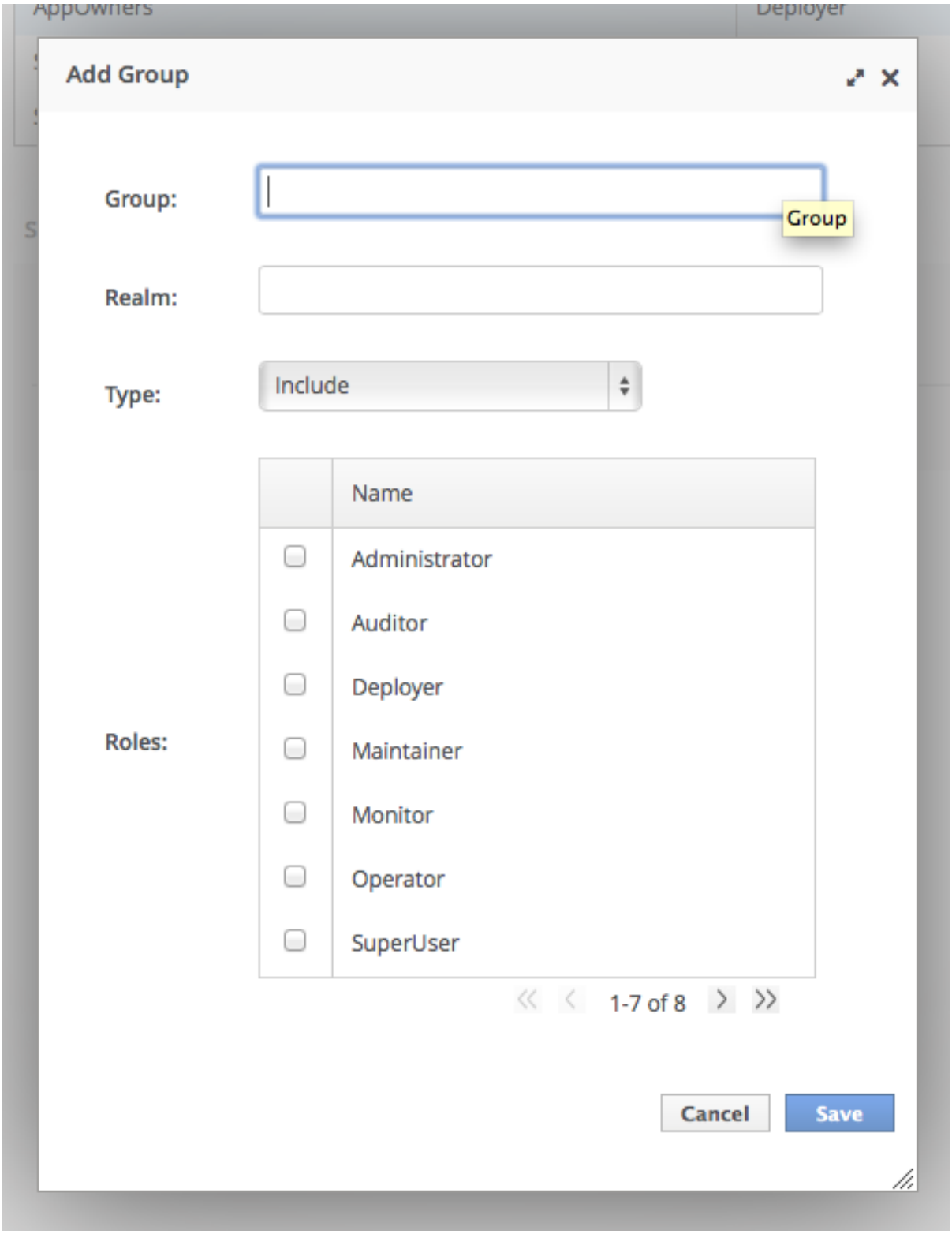
以下の手順で、管理コンソールのグループロール設定を確認します。

1. 管理コンソールへログインします。
2. Administration タブをクリックします。
3. Access Control メニューを展開し、Role Assignment を選択します。
4. GROUPS タブを選択します。

手順6.13 グループの新しいロール割り当ての作成

1. 管理コンソールへログインします。
2. Role Assignment セクションの GROUPS タブに移動します。
3. ユーザー一覧の右上にある Add ボタンをクリックします。Add Group ダイアログが表示されます。

図6.3 Add Group ダイアログ



4. **グループ名を指定し、任意でレルムを指定します。**

5. **Type メニューを include または exclude に設定します。**

6. **include または exclude するロールのチェックボックスをクリックします。複数の項目を**

確認するには、コントロールキー（OSX のコマンドキー）を保持します。

7. Save をクリックして終了します。

成功すると、グループの 追加 ダイアログが閉じ、グループの一覧が更新され、変更内容が反映されます。Failed to save role assignment メッセージが表示されます。

手順6.14 グループロール割り当ての更新

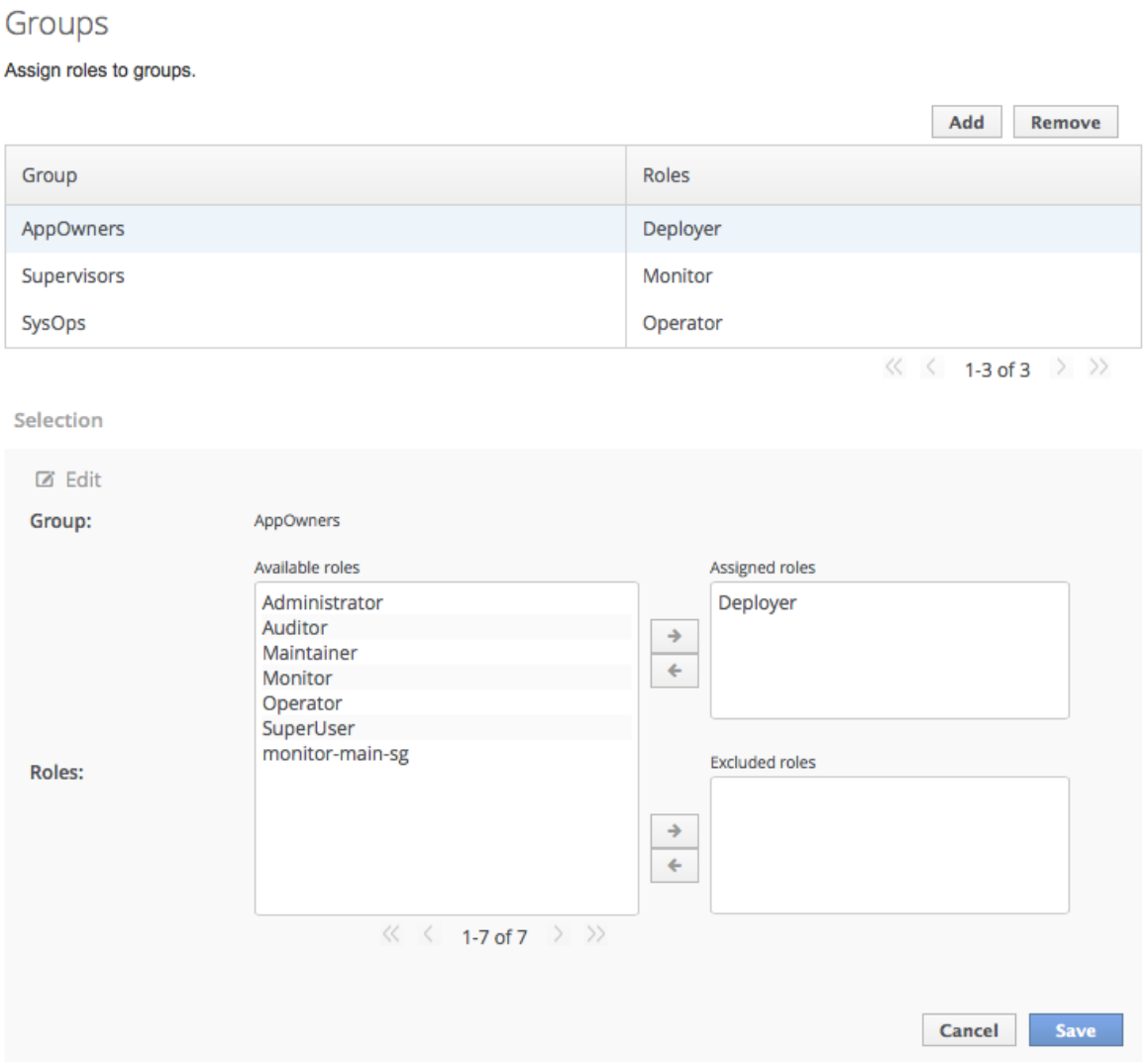
1. 管理コンソールへログインします。

2. Role Assignment セクションの GROUPS タブに移動します。

3. リストよりグループを選択します。

4. Edit をクリックします。Selection ビューが編集モードになります。

図6.4 Selection ビュー編集モード



ここで、グループに割り当てられたロールや除外されたロールを追加および削除できます。

- 割り当てられたロールを追加するには、左側の使用可能なロールのリストからロールを選択し、割り当てられたロールリストの横にある右矢印のボタンをクリックします。ロールは利用可能なリストから割り当てられたリストに移動します。

- 割り当てられたロールを削除するには、割り当てられたロールのリストからロールを選択し、割り当てられたロールリストの横にある左矢印のボタンをクリックします。ロールは割り当てられたリストから利用可能なリストに移動します。

- 除外されたロールを追加するには、左側の使用可能なロールのリストからロールを選択し、除外されたロールリストの横にある右矢印のボタンをクリックします。ロールは利用可能な一覧から除外されたリストに移動します。

- 除外されたロールを削除するには、除外されたロールのリストからロールを選択し、除外されたロールリストの横にある左矢印のボタンをクリックします。ロールは除外されたリストから利用可能なリストに移動します。

5. Save をクリックして終了します。

正常に保存されると、編集ビューが閉じられ、グループのリストが更新され、変更内容が反映されます。Failed to save role assignment メッセージが表示されます。

手順6.15 グループのロール割り当ての削除

1. 管理コンソールへログインします。
2. Role Assignment セクションの GROUPS タブに移動します。
3. リストよりグループを選択します。
4. Remove をクリックします。Remove Role Assignment 確認プロンプトが表示されます。
5. Confirm をクリックします。

正しく削除されると、グループロール割り当てのリストにロールが表示されなくなります。

ロール割り当てのリストからグループを削除しても、ユーザーグループがシステムから削除されず、そのグループのメンバーにロールが割り当てられなくなる保証はありません。各グループメンバーに直接ロールが割り当てられる可能性があります。

バグの報告

6.9.6. 管理 CLI を用いたグループロール割り当ての設定

ロールに含まれる、またはロールから除外されるグループは、管理コンソールおよび管理 CLI で設定できます。このトピックでは、管理 CLI の使用についてのみ説明します。

ユーザーおよびグループのロールへのマッピングの設定は、role-mapping 要素として /core-service=management/access=authorization の管理 API にあります。

この設定を行えるのは、SuperUser または Administrator ロールのユーザーのみです。

コマンドへのアクセスを容易にするため、管理 CLI で /core-service=management/access=authorization の場所を変更します。

```
[standalone@localhost:9999] cd /core-service=management/access=authorization
```

手順6.16 グループロール割り当て設定の表示

1. **read-children-names 操作を使用して、設定されたロールの完全なリストを取得します。**

   /core-service=management/access=authorization:read-children-names(child-type=role-mapping)

   ```
   [standalone@localhost:9999 access=authorization] :read-children-names(child-type=role-mapping)
   {
       "outcome" => "success",
       "result" => [
           "Administrator",
           "Deployer",
           "Maintainer",
           "Monitor",
           "Operator",
           "SuperUser"
       ]
   }
   ```

2. **指定した role-mapping の read-resource 操作を使用して、特定のロールの詳細を取得します。**

   /core-service=management/access=authorization/role-mapping=*ROLENAME*:read-resource(recursive=true)

   ```
   [standalone@localhost:9999 access=authorization] ./role-mapping=Administrator:read-resource(recursive=true)
   {
   ```

```
    "outcome" => "success",
    "result" => {
        "include-all" => false,
        "exclude" => undefined,
        "include" => {
            "user-theboss" => {
                "name" => "theboss",
                "realm" => undefined,
                "type" => "USER"
            },
            "user-harold" => {
                "name" => "harold",
                "realm" => undefined,
                "type" => "USER"
            },
            "group-SysOps" => {
                "name" => "SysOps",
                "realm" => undefined,
                "type" => "GROUP"
            }
        }
    }
}
[standalone@localhost:9999 access=authorization]
```

手順6.17 新規ロールの追加

この手順では、ロールの role-mapping エントリーを追加する方法を説明します。これは、ロールを設定する前に行う必要があります。

- **add 操作を使用して、新しいロール設定を追加します。**

```
/core-service=management/access=authorization/role-mapping=ROLENAME:add
```

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor:add
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

手順6.18 グループに include されるユーザーの追加

この手順では、グループをロールの include されたリストに追加する方法を説明します。

ロールの設定が行われていない場合は、そのロールの role-mapping エントリーを最初に実行する必要があります。

- add 操作を使用して、ロールの追加一覧にグループエントリーを追加します。

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:add(name=GROUPNAME, type=GROUP)
```

ROLENAME は、設定されるロールの名前です。

GROUPNAME は、include リストに追加されるグループの名前です。

ALIAS このマッピングの一意の名前です。Red Hat は、group-*GROUPNAME* などのエイリアスの命名規則を使用することを推奨します。

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/include=group-
investigators:add(name=investigators, type=GROUP)
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

手順6.19 ロールに exclude されるグループの追加

この手順では、グループをロールの exclude されたリストに追加する方法を説明します。

ロールの設定が行われていない場合は、そのロールの role-mapping エントリーを最初に作成する必要があります。

- add 操作を使用して、ロールの除外一覧にグループエントリーを追加します。

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:add(name=GROUPNAME, type=GROUP)
```

ROLENAME は、設定されるロールの名前です。

GROUPNAME は、include リストに追加されるグループの名前です。

ALIAS このマッピングの一意の名前です。Red Hat は、group-*GROUPNAME* などのエイリアスの命名規則を使用することを推奨します。

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/exclude=group-supervisors:add(name=supervisors, type=GROUP)
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

手順6.20 グループ－ロールの include 設定の削除

この手順では、ロールマッピングからグループ包含エントリーを削除する方法を説明します。

- **remove 操作を使用してエントリーを削除します。**

```
/core-service=management/access=authorization/role-mapping=ROLENAME/include=ALIAS:remove
```

***ROLENAME* は、設定されるロールの名前です。**

ALIAS このマッピングの一意の名前です。Red Hat は、group-*GROUPNAME* などのエイリアスの命名規則を使用することを推奨します。

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/include=group-investigators:remove
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

包含の一覧からグループを削除しても、グループはシステムから削除されず、ロールがこのグループのユーザーに割り当てられないようにします。このロールは、引き続きグループのユーザーに割り当てられます。

手順6.21 ユーザーグループの exclude エントリーの削除

この手順では、ロールマッピングからグループ除外エントリーを削除する方法を説明します。

- **remove 操作を使用してエントリーを削除します。**

```
/core-service=management/access=authorization/role-mapping=ROLENAME/exclude=ALIAS:remove
```

***ROLENAME* は、設定されるロールの名前です。**

ALIAS このマッピングの一意の名前です。**Red Hat** は、**group-*GROUPNAME*** などのエイリアスの命名規則を使用することを推奨します。

```
[standalone@localhost:9999 access=authorization] ./role-mapping=Auditor/exclude=group-supervisors:remove
{"outcome" => "success"}
[standalone@localhost:9999 access=authorization]
```

除外の一覧からグループを削除しても、システムからそのグループは削除されません。また、ロールがグループのメンバーに割り当てられることを保証する訳ではありません。依然としてロールはグループメンバーシップに基づいて除外される可能性があります。

バグの報告

6.9.7. LDAP での承認とグループローディング

LDAP ディレクトリーには、ユーザーアカウントおよびグループのエントリーが含まれ、属性によって参照されます。**LDAP** サーバーの設定によっては、ユーザーエンティティーは、ユーザーが **memberOf** 属性を介して所属するグループをマッピングでき、グループエンティティーは **uniqueMember** 属性を介して所属するユーザーをマッピングできます。または、両方のマッピングは **LDAP** サーバーで維持できます。

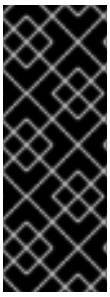
通常、ユーザーは簡単なユーザー名を使用してサーバーに対して認証を行います。グループメンバーシップ情報を検索する場合、使用中のディレクトリーサーバーによっては、この単純名またはディレクトリー内のユーザーのエントリーの識別名を使用して検索を実行できます。

サーバーに接続するユーザーの認証手順が常に最初に行われます。ユーザーの認証に成功すると、サーバーはユーザーのグループを読み込みます。認証のステップと承認の手順では、それぞれ **LDAP** サーバーへの接続が必要です。レルムは、グループの読み込みステップの認証接続を再利用してこのプロセスを最適化します。以下の設定手順のように、承認セクションでルールを定義して、ユーザーの簡単なユーザー名を識別名に変換できます。認証時の「ユーザー名から識別名へのマッピング」検索の結果はキャッシュされ、**force** 属性が **"false"** に設定されている場合の承認クエリー時に再利用されます。**force** が **true** の場合、承認中に検索が再度実行されます（グループのロード中）。これは通常、異なるサーバーが認証および承認を実行すると実行されます。

```
<authorization>
   <ldap connection="...">
    <!-- OPTIONAL -->
     <username-to-dn force="true">
        <!-- Only one of the following. -->
        <username-is-dn />
        <username-filter base-dn="..." recursive="..." user-dn-attribute="..." attribute="..." />
        <advanced-filter base-dn="..." recursive="..." user-dn-attribute="..." filter="..." />
```

```
        </username-to-dn>

        <group-search group-name="..." iterative="..." group-dn-attribute="..." group-name-attribute="..." >
            <!-- One of the following -->
            <group-to-principal base-dn="..." recursive="..." search-by="...">
                <membership-filter principal-attribute="..." />
            </group-to-principal>
            <principal-to-group group-attribute="..." />
        </group-search>
    </ldap>
</authorization>
```



重要

これらの例では、デフォルト値で一部の属性を指定します。これはデモ用に行われます。デフォルト値を指定する属性は、サーバーによって永続化される時に設定から削除されます。例外は **force** 属性です。これは、デフォルト値の **false** に設定されていても必要になります。

username-to-dn

username-to-dn 要素は、ユーザー名を **LDAP** ディレクトリー内のエントリーの識別名にマップする方法を指定します。この要素は、どちら も **true** の場合にのみ必要です。

- 認証および承認の手順は、異なる **LDAP** サーバーに対して行われます。
- グループ検索が識別名を使用する。

1:1 username-to-dn

リモートユーザーが入力するユーザー名がユーザーの識別名であることを指定します。

```
<username-to-dn force="false">
  <username-is-dn />
</username-to-dn>
```

これにより **1:1** マッピングが定義され、追加設定はありません。

username-filter

次のオプションは、認証手順の上記で説明した簡単なオプションと非常に似ています。指定された属性で、指定のユーザー名に対して一致するものが検索されます。

```
<username-to-dn force="true">
   <username-filter base-dn="dc=people,dc=harold,dc=example,dc=com" recursive="false"
attribute="sn" user-dn-attribute="dn" />
</username-to-dn>
```

設定可能な属性は次のとおりです。

- **base-dn:** 検索を開始するコンテキストの識別名。
- **recursive:** 検索がコンテキストのサブコンテキストを拡張するかどうか。デフォルトは **false** です。
- **attribute:** 入力したユーザー名と照合するユーザーエントリーの属性。デフォルトは **uid** です。
- **user-dn-attribute:** ユーザーの識別名を取得するために読み込む属性です。デフォルトは **5** です。

advanced-filter

詳細フィルターを指定するオプションです。認証セクションでは、カスタムフィルターを使用してユーザーの識別名を見つけられます。

```
<username-to-dn force="true">
   <advanced-filter base-dn="dc=people,dc=harold,dc=example,dc=com" recursive="false"
filter="sAMAccountName={0}" user-dn-attribute="dn" />
</username-to-dn>
```

username-filter の例と同じ属性は、意味とデフォルト値も同じです。新たな属性があります。

- **filter:** ユーザーのエントリーの検索に使用するカスタムフィルター。ユーザー名は **{0}** の場所に置き換え られます。

重要

フィルターの定義後も XML が有効である必要があるため、& などの特殊文字を使用する場合には、適切な形式が使用されるようにします。たとえば、& 文字の & です。

グループ検索

グループメンバーシップ情報の検索時に使用できるスタイルは 2 つあります。最初のスタイルは、ユーザーのエントリーに、ユーザーがメンバーであるグループを参照する属性が含まれる場所です。2 つ目のスタイルは、グループにユーザーエントリーを参照する属性が含まれる場所です。

使用するスタイルを選択する場合、Red Hat では、グループを参照するユーザーエントリーの設定を使用することを推奨します。これは、検索を実行せずに既知の識別名の属性を読み取ることで、グループ情報をロードできるからです。他のアプローチでは、ユーザーを参照するグループを特定するための大規模な検索が必要です。

設定を記述する前に、LDIF の例を見てみましょう。

例6.1 LDIF の例: プリンシパルからグループ

この例では、GroupOne に所属するユーザー TestUserOne があり、 GroupOne は GroupFive のメンバーになります。グループメンバーシップは、ユーザー（またはグループ）がメンバーであるグループの識別名に設定された memberOf 属性を使用すると表示されます。

ここには示されていませんが、ユーザーが直接メンバーであるグループごとに 1 つずつ、複数の memberOf 属性を設定することは可能です。

```
dn: uid=TestUserOne,ou=users,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
distinguishedName: uid=TestUserOne,ou=users,dc=principal-to-group,dc=example,dc=org
memberOf: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
memberOf: uid=Slashy/Group,ou=groups,dc=principal-to-group,dc=example,dc=org
userPassword::
e1NTSEF9WFpURzhLVjc4WVZBQUJNbEl3Ym96UVAva0RTNlFNWUpLOTdTMUE9PQ==
```

```
dn: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupOne
distinguishedName: uid=GroupOne,ou=groups,dc=principal-to-group,dc=example,dc=org
memberOf: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-group,dc=example,dc=org
objectClass: extensibleObject
objectClass: top
objectClass: groupMember
objectClass: group
objectClass: uidObject
uid: GroupFive
distinguishedName: uid=GroupFive,ou=subgroups,ou=groups,dc=principal-to-
group,dc=example,dc=org
```

例6.2 LDIF の例: グループからプリンシパル

この例では、GroupOne のメンバーである同じユーザー TestUserOne を示しています。これは GroupFive のメンバーですが、この例ではグループから相互参照に使用されるユーザーに uniqueMember 属性になります。

ここでも、グループメンバーシップの相互参照に使用される属性は繰り返すことができます。GroupFive を確認すると、ここに表示されない別のユーザー TestUserFive への参照もあります。

```
dn: uid=TestUserOne,ou=users,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: inetOrgPerson
objectClass: uidObject
objectClass: person
objectClass: organizationalPerson
cn: Test User One
sn: Test User One
uid: TestUserOne
userPassword::
e1NTSEF9SjR0OTRDR1ltaHc1VVZQOEJvbXhUYjl1dkFVd1lQTmRLSEdzaWc9PQ==

dn: uid=GroupOne,ou=groups,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group One
uid: GroupOne
```

```
uniqueMember: uid=TestUserOne,ou=users,dc=group-to-principal,dc=example,dc=org

dn: uid=GroupFive,ou=subgroups,ou=groups,dc=group-to-principal,dc=example,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: Group Five
uid: GroupFive
uniqueMember: uid=TestUserFive,ou=users,dc=group-to-principal,dc=example,dc=org
uniqueMember: uid=GroupOne,ou=groups,dc=group-to-principal,dc=example,dc=org
```

一般的なグループ検索

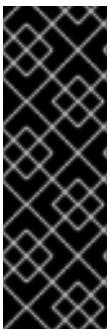
前述の 2 つの方法の例を確認する前に、両方の方法に共通する属性を定義する必要があります。

```
<group-search group-name="..." iterative="..." group-dn-attribute="..." group-name-attribute="..." >
   ...
</group-search>
```

- group-name: この属性は、ユーザーがメンバーであるグループの一覧として返されるグループ名に使用するフォームを指定するために使用されます。これは、グループ名の単純な形式か、グループの識別名のいずれかになります。識別名が必要な場合は、この属性を DISTINGUISHED_NAME に設定できます。デフォルトは SIMPLE です。

- iterative: この属性は、ユーザーが所属するグループを特定した後に、グループをもとに繰り返し検索を行い、グループに所属するグループを特定します。反復検索が有効な場合は、他のグループやサイクルが検出されると、メンバーではないグループに到達するまで継続されます。デフォルトは false です。

同時グループメンバーシップは問題ではありません。各検索の記録は、すでに検索されているグループが再度検索されないように保持されます。



重要

繰り返し検索を行うには、グループエントリーがユーザーエントリーと同じものを見える必要があります。ユーザーがメンバーとなっているグループを識別するのに使用するのと同じアプローチを使用して、グループがメンバーとなっているグループを特定します。これは、グループのグループメンバーシップが相互参照に使用される属性の名前が変更されたり、参照の方向が変更された場合に実行できません。

- group-dn-attribute: 属性が識別名であるグループのエントリー。デフォルトは 5 です。

- **group-name-attribute: 属性が簡単な名前であるグループのエントリー。デフォルトは uid です。**

例6.3 プリンシパルからグループへの設定例

上からの LDIF の例を基に、ユーザーのグループを繰り返しロードする設定の例は次のとおりです。相互参照に使用される属性はユーザーの memberOf 属性です。

```
<authorization>
  <ldap connection="LocalLdap">
    <username-to-dn>
      <username-filter base-dn="ou=users,dc=principal-to-group,dc=example,dc=org"
recursive="false" attribute="uid" user-dn-attribute="dn" />
    </username-to-dn>
    <group-search group-name="SIMPLE" iterative="true" group-dn-attribute="dn" group-name-
attribute="uid">
      <principal-to-group group-attribute="memberOf" />
    </group-search>
  </ldap>
</authorization>
```

この設定で最も重要なことは、principal-to-group 要素が単一の属性で追加されていることです。

- **group-attribute: ユーザーがメンバーであるグループの識別名と一致するユーザーエントリーの属性名。デフォルトは memberOf です。**

例6.4 グループからプリンシパルへの設定例

この例は、前述のグループからプリンシパルへの LDIF の例に対する繰り返し検索を示しています。

```
<authorization>
   <ldap connection="LocalLdap">
     <username-to-dn>
       <username-filter base-dn="ou=users,dc=group-to-principal,dc=example,dc=org"
recursive="false" attribute="uid" user-dn-attribute="dn" />
     </username-to-dn>
     <group-search group-name="SIMPLE" iterative="true" group-dn-attribute="dn" group-name-
attribute="uid">
       <group-to-principal base-dn="ou=groups,dc=group-to-principal,dc=example,dc=org"
recursive="true" search-by="DISTINGUISHED_NAME">
         <membership-filter principal-attribute="uniqueMember" />
       </group-to-principal>
```

```
        </group-search>
    </ldap>
</authorization>
```

ここでは、要素 group-to-principal が追加されます。この要素は、ユーザーエントリーを参照するグループの検索方法を定義するために使用されます。以下の属性が設定されます。

- base-dn: 検索を開始するために使用するコンテキストの識別名。
- recursive: サブコンテキストも検索されるかどうか。デフォルトは false です。
- search-by: 検索で使用するロール名の形式。有効な値は SIMPLE および DISTINGUISHED_NAME です。デフォルトは DISTINGUISHED_NAME です。

group-to-principal 要素には、相互参照を定義する membership-filter 要素があります。

- principal-attribute: ユーザーエントリーを参照するグループエントリーの属性の名前。デフォルトは member です。

バグの報告

6.9.8. スコープ指定ロール

スコープ指定されたロールは、指定された 1 つ以上のサーバーグループまたはホストに対してのみ、1 つの標準ロールのパーミッションを付与するユーザー定義のロールです。スコープ指定されたロールでは、管理ユーザーに、必要なサーバーグループまたはホストのみに制限されるパーミッションを付与することができます。

Administrator または SuperUser ロールが割り当てされたユーザーがスコープ指定ロールを作成できます。

スコープ指定ロールは 5 つの特性によって定義されます。

1. 一意名。

2. ベースになる標準ロール。

3. サーバーグループまたはホストへ適用されるかどうか。

4. 制限されたサーバーグループまたはホストの一覧。

5. すべてのユーザーが自動的に含まれるかどうか。デフォルトは **false** です。

作成すると、スコープ指定されたロールは標準ロールと同じ方法でユーザーおよびグループに割り当てることができます。

スコープ指定されたロールを作成しても、新しいパーミッションは定義できません。スコープ指定されたロールは、制限されたスコープ内で既存のロールのパーミッションを適用する場合にのみ使用できます。たとえば、単一のサーバーグループに制限される **Deployer** ロールに基づいてスコープ指定されたロールを作成できます。

ロールは、ホストとサーバーグループの **2** つのスコープのみに限定されます。

ホストスコープ指定ロール

ホストスコープ指定ロールは、そのロールのパーミッションは単一または複数のホストに制限されます。つまり、関連する **/host=*/** リソースツリーにアクセスが提供されますが、他のホストに固有のリソースは非表示になります。

サーバーグループスコープ指定ロール

サーバーグループスコープ指定ロールは、そのロールのパーミッションを **1** つ以上のサーバーグループに制限します。さらに、ロールパーミッションは、指定した **server-groups** に関連付けられたプロファイル、ソケットバインディンググループ、サーバー設定、およびサーバーリソースにも適用されます。サーバーグループに論理的に関連していないものの内部のサブリソースは、ユーザーには認識できません。

ホストおよびサーバーグループスコープ指定ロールは、その他の管理対象ドメイン設定に対して **Monitor** ロールのパーミッションを持ちます。

バグの報告

6.9.9. スコープ指定ロールの作成

スコープ指定されたロールは、指定された 1 つ以上のサーバーグループまたはホストに対してのみ、1 つの標準ロールのパーミッションを付与するユーザー定義のロールです。このトピックでは、スコープ指定ロールの作成方法について説明します。

この設定を実行できるのは、SuperUser または Administrator ロールのユーザーのみです。

管理コンソールのスコープ指定ロール設定は、以下の手順で確認できます。

1. 管理コンソールへログインします。
2. 管理 タブをクリックします。
3. Access Control メニューを展開し、Role Assignment を選択します。
4. ROLES タブを選択し、その中の Scoped Roles タブを選択します。

管理コンソールの スコープ指定 ロールセクションは、主に 2 つの領域、現在設定されているスコープ指定ロールのリストが含まれるテーブル、テーブルで現在選択されているロールの詳細を表示する Selection パネルで構成されます。

スコープ指定ロールの設定タスクを実行する手順は次のとおりです。

手順6.22 新しいスコープ指定されたロールの追加

1. 管理コンソールへログインします。
2. Roles タブの Scoped Roles エリアに移動します。

3. Add をクリックします。Add Scoped Role ダイアログが表示されます。

4. 以下の詳細を指定します。

- 名前（新しいスコープ指定ロールの一意の名前）。
- ベースロール: このロールがパーミッションをベースとするロール。
- このロールをホストまたはサーバーグループに制限するかどうかを 入力 します。
- スコープ は、ロールが制限されるホストまたはサーバーグループの一覧です。複数のエントリーを選択できます。
- All を含め ます。このロールには全ユーザーが自動的に含まれるはずです。デフォルトは no です。

5. Save をクリックすると、ダイアログが閉じられ、新規作成されたロールがテーブルに表示されます。

手順6.23 スコープ指定されたロールの編集

1. 管理コンソールへログインします。

2. Roles タブの Scoped Roles エリアに移動します。

3. テーブルで編集するスコープ指定ロールをクリックします。そのロールの詳細は、表の下にある Selection パネルに表示されます。

4. Selection パネルで Edit をクリックします。Selection パネルが編集モードになります。

5. 変更する必要のある詳細を更新して、Save ボタンをクリックします。Selection パネルが以前の状態に戻ります。Selection パネルとテーブルの両方に、新たに更新された詳細が表示

されます。

手順6.24 スコープ指定されたロールメンバーの表示

1. 管理コンソールへログインします。
2. Roles タブの Scoped Roles エリアに移動します。
3. Members を表示するテーブルでスコープ指定ロールをクリックし、Members をクリックします。ロールダイアログの Members が表示されます。ロールに含まれる、またはロールから除外されるユーザーおよびグループが表示されます。
4. この情報の確認が終了したら Done をクリックします。

手順6.25 スコープ指定ロールの削除

重要

ユーザーまたはグループが割り当てられている場合は、スコープ指定ロール を削除できません。最初にロールの割り当てを削除してから、スコープ指定ロールを削除します。

1. 管理コンソールへログインします。
2. Roles タブの Scoped Roles エリアに移動します。
3. テーブル上で、削除するスコープ指定ロールを選択します。
4. 削除 ボタンをクリックします。Remove Scoped Role ダイアログが表示されます。
5. Confirm.The ダイアログが閉じ、ロールが削除されます。

バグの報告

6.10. 制約の設定

6.10.1. 機密性制約の設定

各機密性制約は、「機密」とみなされるリソースのセットを定義します。通常、機密リソースとは、パスワードなどの秘密のリソースや、ネットワーキング、JVM 設定、システムプロパティーなどのサーバーに深刻な影響を与えるリソースのことです。アクセス制御システム自体も機密であると見なされます。リソースの機密性は、どのロールが特定のリソースの読み取り、書き込み、またはアドレス指定できるかを制限します。

機密性制約の設定は、管理 API(/core-service=management/access=authorization/constraint=sensitivity-classification)にあります。

管理モデル内で、各機密性制約は classification として識別されます。分類は types にグループ化されます。39 個のタイプには分類が含まれており、これらの分類は 13 タイプにグループ化されます。

機密性制約を設定するには、write-attribute 操作を使用して configured-requires-read、configured-requires-write、または configured-requires-addressable 属性を設定します。操作のタイプを機密に設定するには、属性の値を true に設定します。機密にしない場合は値を false に設定します。デフォルトでは、これらの属性は設定されず、default-requires-read、default-requires-write、および default-requires-addressable の値が使用されます。設定した属性が適用されると、デフォルトではなく、その値が使用されます。デフォルト値は変更できません。

例6.5 読み取りシステムプロパティーを機密操作にする

```
[domain@localhost:9999 /] cd /core-
service=management/access=authorization/constraint=sensitivity-
classification/type=core/classification=system-property
[domain@localhost:9999 classification=system-property] :write-attribute(name=configured-
requires-read, value=true)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => {"main-server-group" => {"host" => {"master" => {
        "server-one" => {"response" => {"outcome" => "success"}},
        "server-two" => {"response" => {"outcome" => "success"}}
    }}}}
}
[domain@localhost:9999 classification=system-property] :read-resource
{
    "outcome" => "success",
    "result" => {
        "configured-requires-addressable" => undefined,
```

```
        "configured-requires-read" => true,
        "configured-requires-write" => undefined,
        "default-requires-addressable" => false,
        "default-requires-read" => false,
        "default-requires-write" => true,
        "applies-to" => {
            "/host=master/system-property=*" => undefined,
            "/host=master/core-service=platform-mbean/type=runtime" => undefined,
            "/server-group=*/system-property=*" => undefined,
            "/host=master/server-config=*/system-property=*" => undefined,
            "/host=master" => undefined,
            "/system-property=*" => undefined,
            "/" => undefined
        }
    }
}
[domain@localhost:9999 classification=system-property]
```

これらの属性の設定に応じて、どのロールがどの操作を実行できるかは、表6.2「機密性制約の設定結果」 に要約されています。

表6.2 機密性制約の設定結果

値	requires-read	requires-write	requires-addressable
true	読み取りは機密です。 **Auditor**、**Administrator**、**SuperUser** のみを読み取ることができます。	書き込みは機密です。 **Administrator** および **SuperUser** のみが書き込み可能です。	アドレス指定は機密です。 **Auditor**、**Administrator**、**SuperUser** のみがアドレス指定できます。
false	読み取りは機密ではありません。 すべての管理ユーザーが読み取り可能です。	書き込みは機密ではありません。 **Maintainer**、**Administrator**、および **SuperUser** のみが書き込み可能です。また、**Deployer** は リソースをアプリケーションリソースとして記述することもできます。	アドレス指定は機密ではありません。 すべての管理ユーザーがアドレス指定できます。

バグの報告

6.10.2. アプリケーションリソース制約の設定

各アプリケーションリソース制約は、通常アプリケーションとサービスのデプロイメントに関連するリソース、属性、および操作のセットを定義します。アプリケーションリソース制約が有効化される

と、**Deployer** ロールの管理ユーザーに、適用されるリソースへのアクセスが付与されます。

アプリケーション制約の設定は、**/core-service=management/access=authorization/constraint=application-classification/** の管理モデルにあります。

管理モデル内で、各アプリケーションリソース制約は **classification** として識別されます。分類は **types** にグループ化されます。**14** 個の分類が含まれており、**8** タイプにグループ化されます。各分類には、分類設定が適用されるリソースパスパターンの一覧である **applies-to** 要素があります。

デフォルトでは、有効になっている唯一のアプリケーションリソースの分類は コア です。コアには、デプロイメント、デプロイメントオーバーレイ、およびデプロイメント操作が含まれます。

アプリケーションリソースを有効にするには、**write-attribute** 操作を使用して、分類の **configured-application attribute** を **true** に設定します。アプリケーションリソースを無効にするには、この属性を **false** に設定します。デフォルトでは、これらの属性は設定されず、**default-application attribute** の値が使用されます。デフォルト値は変更できません。

例6.6 logger-profile アプリケーションリソースの分類を有効にする

```
[domain@localhost:9999 /] cd /core-
service=management/access=authorization/constraint=application-
classification/type=logging/classification=logging-profile
[domain@localhost:9999 classification=logging-profile] :write-attribute(name=configured-
application, value=true)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => {"main-server-group" => {"host" => {"master" => {
        "server-one" => {"response" => {"outcome" => "success"}},
        "server-two" => {"response" => {"outcome" => "success"}}
    }}}}
}
[domain@localhost:9999 classification=logging-profile] :read-resource
{
    "outcome" => "success",
    "result" => {
        "configured-application" => true,
        "default-application" => false,
        "applies-to" => {"/profile=*/subsystem=logging/logging-profile=*" => undefined}
    }
}
[domain@localhost:9999 classification=logging-profile]
```

重要

Application Resource Constraints は、その設定に一致するすべてのリソースに適用されます。たとえば、Deployer ユーザーに、あるデータソースリソースへのアクセスを許可せず、別のデータソースリソースへのアクセスを拒否することはできません。このレベルの分離が必要な場合は、異なるサーバーグループでリソースを設定し、グループごとに異なるスコープ指定の Deployer ロールを作成することが推奨されます。

バグの報告

6.10.3. Vault 式制約の設定

デフォルトでは、vault 式の読み書きは機密操作です。Vault 式制約を設定すると、これらの操作のいずれかまたは両方を非機密に設定できます。この制約を変更すると、より多くのロールで vault 式の読み書きが可能になります。

vault 式制約は、/core-service=management/access=authorization/constraint=vault-expression の管理モデルにあります。

vault 式制約を設定するには、write-attribute 操作を使用して configured-requires-write および configured-requires-read の属性を true または false に設定します。デフォルトでは、これらは設定されず、default-requires-read および default-requires-write の値が使用されます。デフォルト値は変更できません。

例6.7 vault 式への書き込みを非機密操作にする

```
[domain@localhost:9999 /] cd /core-service=management/access=authorization/constraint=vault-expression
[domain@localhost:9999 constraint=vault-expression] :write-attribute(name=configured-requires-write, value=false)
{
    "outcome" => "success",
    "result" => undefined,
    "server-groups" => {"main-server-group" => {"host" => {"master" => {
        "server-one" => {"response" => {"outcome" => "success"}},
        "server-two" => {"response" => {"outcome" => "success"}}
    }}}}
}
[domain@localhost:9999 constraint=vault-expression] :read-resource
{
    "outcome" => "success",
    "result" => {
        "configured-requires-read" => undefined,
        "configured-requires-write" => false,
        "default-requires-read" => true,
```

```
        "default-requires-write" => true
    }
}
[domain@localhost:9999 constraint=vault-expression]
```

表6.3「vault 式制約の設定結果」 で、この設定に応じて Vault 式の読み書きが可能なロールの概要を示します。

表6.3 vault 式制約の設定結果

値	requires-read	requires-write
true	読み取り操作は機密です。 **Auditor**、**Administrator**、および **SuperUser** のみを読み取ることができます。	書き込み操作は機密です。 **Administrator** および **SuperUser** のみが書き込み可能です。
false	読み取り操作は機密ではありません。 すべての管理ユーザーは読み取りが可能です。	書き込み操作は機密ではありません。 **Monitor**、**Administrator**、および **SuperUser** は書き込み可能です。**Deployer** は、vault 式がアプリケーションリソースにある場合にも記述できます。

バグの報告

6.11. 制約の参照

6.11.1. アプリケーションリソース制約に関する参考情報

タイプ: core

分類: deployment-overlay

- **デフォルト: true**
- **PATH: /deployment-overlay=***
- **PATH: /deployment=***

- PATH: /

  操作:

  upload-deployment-stream、full-replace-deployment、 upload-deployment-url、upload-deployment-bytes

タイプ: datasources

分類: datasource

- デフォルト: false
- PATH: /deployment=*/subdeployment=*/subsystem=datasources/data-source=*
- PATH: /subsystem=datasources/data-source=*
- PATH: /subsystem=datasources/data-source=ExampleDS
- PATH: /deployment=*/subsystem=datasources/data-source=*

分類: jdbc-driver

- デフォルト: false
- PATH: /subsystem=datasources/jdbc-driver=*

分類: xa-data-source

- デフォルト: false
- PATH: /subsystem=datasources/xa-data-source=*

- PATH: /deployment=*/subsystem=datasources/xa-data-source=*
- PATH: /deployment=*/subdeployment=*/subsystem=datasources/xa-data-source=*

タイプ: logging

分類: logger

- デフォルト: false
- PATH: /subsystem=logging/logger=*
- PATH: /subsystem=logging/logging-profile=*/logger=*

分類: logging-profile

- デフォルト: false
- PATH: /subsystem=logging/logging-profile=*

タイプ: mail

分類: mail-session

- デフォルト: false
- PATH: /subsystem=mail/mail-session=*

タイプ: naming

分類: binding

- デフォルト: false

- PATH: /subsystem=naming/binding=*

タイプ: resource-adapters

分類: resource-adapters

- デフォルト: false

- PATH: /subsystem=resource-adapters/resource-adapter=*

タイプ: security

分類: security-domain

- デフォルト: false

- PATH: /subsystem=security/security-domain=*

バグの報告

6.11.2. 機密性制約に関する参考情報

タイプ: core

分類: access-control

- requires-addressable: true

- requires-read: true

- requires-write: true

- PATH: /core-service=management/access=authorization

- PATH: /subsystem=jmx ATTRIBUTE: non-core-mbean-sensitivity

分類: credential

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=mail/mail-session=*/server=pop3 ATTRIBUTE: username , password
- PATH: /subsystem=mail/mail-session=*/server=imap ATTRIBUTE: username , password
- PATH: /subsystem=datasources/xa-data-source=* ATTRIBUTE: user-name, recovery-username, password, recovery-password
- PATH: /subsystem=mail/mail-session=*/custom=* ATTRIBUTE: username, password
- PATH: /subsystem=datasources/data-source=*" ATTRIBUTE: user-name, password
- PATH: /subsystem=remoting/remote-outbound-connection=*" ATTRIBUTE: username
- PATH: /subsystem=mail/mail-session=*/server=smtp ATTRIBUTE: username, password
- PATH: /subsystem=web/connector=*/configuration=ssl ATTRIBUTE: key-alias, password

- PATH: /subsystem=resource-adapters/resource-adapter=*/connection-definitions=*" ATTRIBUTE: recovery-username, recovery-password

分類: domain-controller

- requires-addressable: false
- requires-read: false
- requires-write: true

分類: domain-names

- requires-addressable: false
- requires-read: false
- requires-write: true

分類: extensions

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /extension=*

分類: jvm

- requires-addressable: false

- requires-read: false
- requires-write: true
- PATH: /core-service=platform-mbean/type=runtime ATTRIBUTE: input-arguments, boot-class-path, class-path, boot-class-path-supported, library-path

分類: management-interfaces

- requires-addressable: false
- requires-read: false
- requires-write: true
- /core-service=management/management-interface=native-interface
- /core-service=management/management-interface=http-interface

分類: module-loading

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=module-loading

分類: patching

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=patching/addon=*
- PATH: /core-service=patching/layer=*"
- PATH: /core-service=patching

分類: read-whole-config

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: / OPERATION: read-config-as-xml

分類: security-domain

- requires-addressable: true
- requires-read: true
- requires-write: true

- PATH: /subsystem=security/security-domain=*

分類: security-domain-ref

- requires-addressable: true
- requires-read: true
- requires-write: true
- PATH: /subsystem=datasources/xa-data-source=* ATTRIBUTE: security-domain
- PATH: /subsystem=datasources/data-source=* ATTRIBUTE: security-domain
- PATH: /subsystem=ejb3 ATTRIBUTE: default-security-domain
- PATH: /subsystem=resource-adapters/resource-adapter=*/connection-definitions=* ATTRIBUTE: security-domain, recovery-security-domain, security-application, security-domain-and-application

分類: security-realm

- requires-addressable: true
- requires-read: true
- requires-write: true
- PATH: /core-service=management/security-realm=*

分類: security-realm-ref

- requires-addressable: true
- requires-read: true
- requires-write: true
- PATH: /subsystem=remoting/connector=* ATTRIBUTE: security-realm
- PATH: /core-service=management/management-interface=native-interface ATTRIBUTE: security-realm
- PATH: /core-service=management/management-interface=http-interface ATTRIBUTE: security-realm
- PATH: /subsystem=remoting/remote-outbound-connection=* ATTRIBUTE: security-realm

分類: security-vault

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=vault

分類: service-container

- requires-addressable: false

- requires-read: false
- requires-write: true
- PATH: /core-service=service-container

分類: snapshots

- requires-addressable: false
- requires-read: false
- requires-write: false
- PATH: / ATTRIBUTE: take-snapshot, list-snapshots, delete-snapshot

分類: socket-binding-ref

- requires-addressable: false
- requires-read: false
- requires-write: false
- PATH: /subsystem=mail/mail-session=*/server=pop3 ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=mail/mail-session=*/server=imap ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=remoting/connector=* ATTRIBUTE: socket-binding

- PATH: /subsystem=web/connector=* ATTRIBUTE: socket-binding
- PATH: /subsystem=remoting/local-outbound-connection=* ATTRIBUTE: outbound-socket-binding-ref
- PATH: /socket-binding-group=*/local-destination-outbound-socket-binding=* ATTRIBUTE: socket-binding-ref
- PATH: /subsystem=remoting/remote-outbound-connection=* ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=mail/mail-session=*/server=smtp ATTRIBUTE: outbound-socket-binding-ref
- PATH: /subsystem=transactions ATTRIBUTE: process-id-socket-binding, status-socket-binding, socket-binding

分類: socket-config

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /interface=* OPERATION: resolve-internet-address
- PATH: /core-service=management/management-interface=native-interface ATTRIBUTE: port, interface, socket-binding
- PATH: /socket-binding-group=*
-

PATH: /core-service=management/management-interface=http-interface ATTRIBUTE: port, secure-port, interface, secure-socket-binding, socket-binding

- PATH: / OPERATION: resolve-internet-address
- PATH: /subsystem=transactions ATTRIBUTE: process-id-socket-max-ports

分類: system-property

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /core-service=platform-mbean/type=runtime ATTRIBUTE: system-properties
- PATH: /system-property=*
- PATH: / OPERATION: resolve-expression

タイプ: datasources

分類: data-source-security

- requires-addressable: false
- requires-read: true
- requires-write: true
-

PATH: /subsystem=datasources/xa-data-source=* ATTRIBUTE: user-name, security-domain, password

- PATH: /subsystem=datasources/data-source=* ATTRIBUTE: user-name, security-domain, password

タイプ: jdr

分類: jdr

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /subsystem=jdr OPERATION: generate-jdr-report

タイプ: jmx

分類: jmx

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /subsystem=jmx

タイプ: mail

分類: mail-server-security

- requires-addressable: false
- requires-read: false
- requires-write: true
- PATH: /subsystem=mail/mail-session=*/server=pop3 ATTRIBUTE: username, tls, ssl, password
- PATH: /subsystem=mail/mail-session=*/server=imap ATTRIBUTE: username, tls, ssl, password
- PATH: /subsystem=mail/mail-session=*/custom=* ATTRIBUTE: username, tls, ssl, password
- PATH: /subsystem=mail/mail-session=*/server=smtp ATTRIBUTE: username, tls, ssl, password

タイプ: naming

分類: jndi-view

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=naming OPERATION: jndi-view

分類: naming-binding

- requires-addressable: false

- requires-read: false
- requires-write: false
- PATH: /subsystem=naming/binding=*

タイプ: remoting

分類: remoting-security

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=remoting/connector=* ATTRIBUTE: authentication-provider, security-realm
- PATH: /subsystem=remoting/remote-outbound-connection=* ATTRIBUTE: username, security-realm
- PATH: /subsystem=remoting/connector=*/security=sasl

タイプ: resource-adapters

分類: resource-adapter-security

- requires-addressable: false
- requires-read: true

- requires-write: true

- PATH: /subsystem=resource-adapters/resource-adapter=*/connection-definitions=* ATTRIBUTE: security-domain, recovery-username, recovery-security-domain, security-application, security-domain-and-application, recovery-password

タイプ: security

分類: misc-security

- requires-addressable: false

- requires-read: true

- requires-write: true

- PATH: /subsystem=security ATTRIBUTE: deep-copy-subject-mode

タイプ: web

分類: web-access-log

- requires-addressable: false

- requires-read: false

- requires-write: false

- PATH: /subsystem=web/virtual-server=*/configuration=access-log

分類: web-connector

- requires-addressable: false

- requires-read: false
- requires-write: false
- PATH: /subsystem=web/connector=*

分類: web-ssl

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=web/connector=*/configuration=ssl

分類: web-sso

- requires-addressable: false
- requires-read: true
- requires-write: true
- PATH: /subsystem=web/virtual-server=*/configuration=sso

分類: web-valve

- requires-addressable: false
- requires-read: false

- **requires-write: false**

- **PATH: /subsystem=web/valve=***

バグの報告

第7章 パスワード VAULT でのパスワードおよびその他の機密文字列のセキュリティー保護

7.1. パスワード VAULT システム

JBoss EAP および関連するアプリケーションの設定には、ユーザー名とパスワードなどの機密情報が必要になります。この機密情報をプレーンテキストとして設定ファイルに保存する代わりに、パスワード Vault 機能を使用してこの情報をマスクし、暗号化されたキーストアに保存します。

パスワードをプレーンテキストで設定ファイルに保存する代わりに、パスワード Vault 機能を使用してパスワード情報をマスクし、暗号化されたキーストアに保存できます。パスワードを保存したら、管理 CLI コマンドまたは独自のアプリケーションに参照を含めることができます。パスワード vault は Java キーストアをストレージメカニズムとして使用します。パスワード vault はストレージと鍵ストレージの 2 つの部分で構成されます。Java キーストアは、Vault ストレージで機密文字列を暗号化または復号化するために使用されるキーを保存するために使用されます。

バグの報告

7.2. パスワード VAULT の設定と使用

パスワード vault で提供されるマスクされたキーストアのパスワード機能は、JBoss EAP サーバーに保存されているパスワード vault からマスクされたキーストアパスワードを取得するオプションを提供します。パスワード vault は Java キーストアをストレージメカニズムとして使用します。

手順7.1 パスワード vault を設定および使用するための基本的な手順

1. パスワード暗号化のキーを保存するように Java キーストアを設定します。

   キーストアの作成に関する詳細は、「機密性が高い文字列を格納する Java キーストアの作成」 を参照してください。

2. パスワード vault を初期化します。

   パスワードをマスクし、パスワード値を初期化する方法は、「パスワード vault の初期化」 を参照してください。

3. パスワード vault を使用するように JBoss EAP 6 を設定します。

パスワード vault を使用するように EAP 6 を設定する方法は、「パスワード vault を使用するよう JBoss EAP 6 を設定」 を参照してください。

4.
パスワード vault に Sensitive 文字列を保存します。

パスワード vault に機密文字列を保存する方法は、「パスワード Vault に Sensitive 文字列を保存します。」 を参照してください。

5.
パスワード vault を使用するように JBoss EAP 6 を設定します。

パスワード vault を使用するよう JBoss EAP 6 を設定する方法は、「パスワード vault を使用するよう JBoss EAP 6 を設定」 を参照してください。カスタムの実装については、「パスワード Vault のカスタム実装を使用するよう JBoss EAP 6 を設定」 を参照してください。

注記

設定で暗号化された機密文字列を使用するには、「暗号化した機密文字列を設定で使用」 を参照してください。

アプリケーションで暗号化された機密文字列を使用するには、「アプリケーションで暗号化した機密文字列の使用」 を参照してください。

パスワード vault で機密文字列を確認するには、「機密文字列がパスワード vault 内にあるかどうかを確認します。」 を参照してください。

パスワード vault から機密文字列を削除するには、「パスワード vault からの機密文字列の削除」 を参照してください。

バグの報告

7.3. 機密性が高い文字列を格納する JAVA キーストアの作成

前提条件

-

Java Runtime Environment(JRE)によって提供される keytool ユーティリティー。ファイルのパスを見つけます。Red Hat Enterprise Linux では、/usr/bin/keytool です。

警告

JCEKS キーストアの実装は Java ベンダーごとに異なります。したがって、使用する JDK と同じベンダーの keytool ユーティリティーを使用してキーストアを生成する必要があります。

別のベンダーの JDK で実行されている JBoss EAP インスタンスで、あるベンダーの JDK からキーツールによって生成されたキーストアを使用すると、以下の例外が発生します。

```
java.io.IOException: com.sun.crypto.provider.SealedObjectForKeyProtector
```

手順7.2 Java キーストアの設定

1. キーストアと他の暗号化された情報を格納するディレクトリーを作成します。

   キーストアおよびその他の重要な情報を保存するディレクトリーを作成します。この手順では、ディレクトリーは *EAP_HOME*/vault/ であることを前提としています。このディレクトリーには機密情報が含まれるため、アクセスは、一部のユーザーに制限する必要があります。JBoss EAP を実行しているユーザーアカウントには少なくとも読み書き込みアクセスが必要です。

2. keytool ユーティリティーで使用するパラメーターを決定します。

   以下のパラメーターの値を決めます。

   alias

   エイリアスは、vault やキーストアに保存されている他のデータの一意識別子です。エイリアスは大文字と小文字を区別しません。

   storetype

   ストアタイプはキーストアのタイプを指定します。値は、jceks が推奨されます。

   keyalg

暗号化に使用するアルゴリズム。JRE およびオペレーティングシステムのドキュメントを参照して、利用可能な他の選択肢を確認します。

keysize

暗号化キーのサイズは、ブルートフォースでの暗号解除の難易度に影響します。適切な値の詳細は、keytool ユーティリティーとともに配布されるドキュメントを参照してください。

storepass

storepass の値は、キーストアに対する認証に使用されるパスワードであるため、キーを読み取ることができます。パスワードは 6 文字以上である必要があります。パスワードは、キーストアへのアクセス時に入力する必要があります。このパラメーターを省略すると、コマンドの実行時に入力が求められます。

keypass

keypass の値は、特定のキーへのアクセスに使用されるパスワードで、storepass パラメーターの値と一致する必要があります。

validity

validity の値は、鍵の有効期間（日数）です。

keystore

キーストア の値は、キーストアの値が保存されるファイルパスおよびファイル名です。キーストアファイルは、データが最初に追加されると作成されます。

正しいファイルパスセパレーターを使用するようにしてください。Red Hat Enterprise Linux や同様のオペレーティングシステムの場合は / （スラッシュ）、Microsoft Windows Server の場合は \ （バックスラッシュ）を使用してください。

Keytool ユーティリティー に は、その他の多くのオプションがあります。詳細は、JRE またはオペレーティングシステムのドキュメントを参照してください。

3. keytool コマンドを実行します。

オペレーティングシステムのコマンドラインインターフェースを起動し、収集した情報を指定して keytool ユーティリティーを実行します。

例7.1 Java キーストアの作成

```
$ keytool -genseckey -alias vault -storetype jceks -keyalg AES -keysize 128 -storepass vault22 -keypass vault22 -validity 730 -keystore EAP_HOME/vault/vault.keystore
```

結果

この例では、*EAP_HOME*/vault/vault.keystore ファイルにキーストアが作成されている。JBoss EAP では、パスワードなどの暗号化された文字列を保存するために使用されるエイリアス vault を持つ単一のキーを保存します。

バグの報告

7.4. パスワード VAULT の初期化

前提条件

- 「機密性が高い文字列を格納する Java キーストアの作成」

概要

パスワード vault は、各パラメーターの値の入力を求めるプロンプトが表示される場合にインタラクティブに初期化できます。または、すべてのパラメーターの値をコンmmand 行に指定する場合に非対話的に初期化できます。各メソッドで同じ結果が表示されるため、任意の方法を選択します。

いずれかの方法を使用する場合は、以下の一覧を参照してください。

keystore URL(KEYSTORE_URL)

キーストアファイルのファイルシステムパスまたは URI。この例では、*EAP_HOME*/vault/vault.keystore を使用します。

キーストアパスワード(KEYSTORE_PASSWORD)

キーストアのアクセスに使用されるパスワード。

Salt (SALT)

salt 値は、キーストアの内容を暗号化するために反復回数とともに使用される 8 文字のランダムな文字列です。

keystore Alias(KEYSTORE_ALIAS)

キーストア認識されているエイリアス。

Iteration Count (ITERATION_COUNT)

暗号化アルゴリズムの実行回数。

Directory to store encrypted files (ENC_FILE_DIR)

暗号化したファイルを保存するパス。これは通常、パスワード vault を含むディレクトリーです。

暗号化されたすべての情報をキーストアと同じ場所に格納することは便利ですが、必須ではありません。このディレクトリーには、制限のあるユーザーのみがアクセスできるようにする必要があります。JBoss EAP を実行しているユーザーアカウントには少なくとも読み書き込みアクセスが必要です。「機密性が高い文字列を格納する Java キーストアの作成」 に従っている場合、キーストアは *EAP_HOME*/vault/ ディレクトリーに置かれます。

注記

ディレクトリー名の末尾のバックスラッシュまたはスラッシュが必要です。正しいファイルパスセパレーターを使用するようにしてください。Red Hat Enterprise Linux や同様のオペレーティングシステムの場合は /（スラッシュ）、Microsoft Windows Server の場合は \（バックスラッシュ）を使用してください。

Vault Block (VAULT_BLOCK)

パスワード vault でこのブロックに与えられる名前。重要な値を選択します。

Attribute (ATTRIBUTE)

保存される属性に与えられる名前。重要な値を選択します。たとえば、データソースに関連付ける名前を選択できます。

Security Attribute (SEC-ATTR)

パスワード vault に保存されているパスワード。

手順7.3 パスワード vault コマンドを対話的に実行

各パラメーターの値の入力を求めるプロンプトを出す場合は、この方法を使用します。

1. **パスワード vault コマンドをインタラクティブに起動します。**

   オペレーティングシステムのコマンドラインインターフェースを起動し、*EAP_HOME*/bin/vault.sh （Red Hat Enterprise Linux および同様のオペレーティングシステム）または *EAP_HOME*\bin\vault.bat （Microsoft Windows Server 上）を実行します。新しい対話セッションを開始するには、0 (ゼロ) と入力します。

2. **要求パラメーターを入力します。**

   プロンプトに従って必要なパラメーターを入力します。

3. **マスクされたパスワード情報をメモします。**

   マスクされたパスワード、salt、iteration count (反復数) は、標準出力に出力されます。安全な場所にそれらを書き留めておきます。これらのエントリーは、パスワード Vault に追加する必要があります。キーストアファイルおよびこの値にアクセスすると、攻撃者はパスワード Vault の機密情報にアクセスできるようになります。

4. **対話式コンソールを終了します。**

   対話式コンソールを終了するには、3 （3 つ）と入力します。

例7.2 パスワード vault コマンドを対話的に実行する

```
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password: vault22
Enter Keystore password again: vault22
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Oct 17, 2014 12:58:11 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in AS7 config file:
*********************************************

...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
```

```
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
********************************************
Vault is initialized and ready for use
Handshake with Vault complete
```

手順7.4 パスワード vault コマンドを非対話的に実行

すべてのパラメーターの値を一度に指定する場合は、この方法を使用してください。

- **オペレーティングシステムのコマンドラインインターフェースを起動し、パスワード Vault コマンドを実行します。プレースホルダーの値を優先する値に置き換える概要 のリストを参照してください。**

  Red Hat Enterprise Linux または同様のオペレーティングシステムでは *EAP_HOME*/bin/vault.sh、Microsoft Windows Server では *EAP_HOME*\bin\vault.bat を使用します。

  ```
  vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD --alias
  KEYSTORE_ALIAS --vault-block VAULT_BLOCK --attribute ATTRIBUTE --sec-attr SEC-
  ATTR --enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT --salt SALT
  ```

  例7.3 パスワード vault コマンドを非対話的に実行

  ```
  vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password vault22 --alias
  vault --vault-block vb --attribute password --sec-attr 0penS3sam3 --enc-dir
  EAP_HOME/vault/ --iteration 120 --salt 1234abcd
  ```

  コマンド出力

  ```
  ================================================================
  =

   JBoss Vault

   JBOSS_HOME: EAP_HOME

   JAVA: java

  ================================================================
  =

  Oct 17, 2014 2:23:43 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
  ```

```
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Secured attribute value has been stored in vault.
Please make note of the following:
********************************************
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
VAULT::vb::password::1
********************************************
Vault Configuration in AS7 config file:
********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
********************************************
```

結果

設定ファイルやデプロイメントで使用するためにキーストアパスワードがマスキングされています。さらに、vault が初期化され、使用できる状態になります。

バグの報告

7.5. 外部ソースからのキーストアパスワードの取得

Java キーストアパスワードを取得するために、Vault 設定の EXT、EXTC、CMD、CMDC、または CLASS メソッドを使用することもできます。

```
<vault-option name="KEYSTORE_PASSWORD" value="[here]"
```

この方法を以下で説明します。

- **{EXT}...: 実際のコマンドを参照します。ここで、「...」は実際のコマンドです。たとえば {EXT}/usr/bin/getmypassword --section 1 --query company など、/usr/bin/getmypassword**

コマンドを実行します。このコマンドは、標準出力のパスワードを表示し、セキュリティー vault のキーストアのパスワードとして使用します。この例では、--section 1 と --query company のオプションを使用しています。

- {EXTC[:expiration_in_millis]}...: 実際のコマンドを参照します。ここで、'...' は、プラットフォームコマンドを実行する Runtime.exec(String)メソッドに渡される実際のコマンドラインです。コマンド出力の最初の行がパスワードとして使用されます。EXTC バリアントは expiration_in_millis ミリ秒のパスワードをキャッシュします。デフォルトのキャッシュの有効期限は 0（ゼロ）で、キャッシュ内のアイテムは期限切れになりません。例：{EXTC:120000}/usr/bin/getmypassword --section 1 --query company キャッシュに /usr/bin/getmypassword 出力が含まれるかどうかを確認し、出力が含まれる場合は使用します。出力がない場合は、コマンドを実行してこれをキャッシュに出力して使用します。この例では、キャッシュの有効期限は 2 分（120000 ミリ秒）です。

- {CMD}... または {CMDC[:expiration_in_millis]}...: 一般的なコマンドは ',' で区切られた文字列です。最初の部分は実際のコマンドであり、追加の部分はパラメーターを表します。コンマにバックスラッシュを付けることで、パラメーターの一部として維持することができます。以下は例になります。 {CMD}/usr/bin/getmypassword,--section,1,--query,company

- {CLASS[@jboss_module_spec]}classname[:ctorargs]: '[:ctorargs]' は、クラス名の ':' で区切ったオプションの文字列です。ctorargs は文字列のカンマ区切りの一覧です。（例：{CLASS@org.test.passwd}org.test.passwd.ExternamPassworProvider ）。この例では、org.test.passwd.ExternamPassworProvider モジュールから org.test.passwd クラスを読み込み、 toCharArray() メソッドを使用してパスワードを取得します。toCharArray() が利用できない場合は、toString() メソッドを使用してください。org.test.passwd.ExternamPassworProvider クラスにはデフォルトのコンストラクターが必要です。

バグの報告

7.6. パスワード VAULT を使用するよう JBOSS EAP 6 を設定

概要

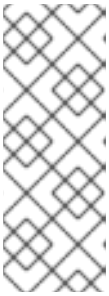
設定ファイル内でパスワードやその他の機密属性をマスクする前に、保存および復号化するパスワード vault を JBoss EAP 6 が認識するようにする必要があります。

前提条件

- 「パスワード vault の初期化」

手順7.5 パスワード vault の有効化

- 以下の管理 CLI コマンドを実行し、「パスワード vault の初期化」 のパスワード vault コマンドの出力からプレースホルダーの値を置換します。



注記

Microsoft Windows Server を使用する場合は、通常 1 つを使用するファイルパスでバックスラッシュ(\\)を使用します。例: C:\\data\\vault\\vault.keystoreこれは、単一のバックスラッシュ(\)が文字エスケープに使用されるためです。

```
/core-service=vault:add(vault-options=[("KEYSTORE_URL" => "PATH_TO_KEYSTORE"),
("KEYSTORE_PASSWORD" => "MASKED_PASSWORD"), ("KEYSTORE_ALIAS" =>
"ALIAS"), ("SALT" => "SALT"),("ITERATION_COUNT" => "ITERATION_COUNT"),
("ENC_FILE_DIR" => "ENC_FILE_DIR")])
```

例7.4 パスワード vault の有効化

```
/core-service=vault:add(vault-options=[("KEYSTORE_URL" =>
"EAP_HOME/vault/vault.keystore"), ("KEYSTORE_PASSWORD" => "MASK-
5dOaAVafCSd"), ("KEYSTORE_ALIAS" => "vault"), ("SALT" => "1234abcd"),
("ITERATION_COUNT" => "120"), ("ENC_FILE_DIR" => "EAP_HOME/vault/")])
```

結果

JBoss EAP 6 は、パスワード vault に保存されているマスクされた文字列を復号化するように設定されています。パスワード vault に文字列を追加して設定で使用するには、「パスワード Vault に Sensitive 文字列を保存します。」 を参照してください。

バグの報告

7.7. パスワード VAULT のカスタム実装を使用するよう JBOSS EAP 6 を設定

概要

SecurityVault の独自の実装を使用して、設定ファイルのパスワードやその他の機密属性をマスクできます。

前提条件

- 「パスワード vault の初期化」

手順7.6 パスワード vault のカスタム実装の使用

1. インターフェース SecurityVault を実装するクラスを作成します。

2. 前の手順のクラスが含まれるモジュールを作成し、インターフェースが org.picketbox である SecurityVault の依存関係を指定します。

3. 以下の属性を用いて vault 要素を追加して、JBoss EAP サーバー設定でカスタムのパスワード vault を有効にします。

   code

   SecurityVault を実装するクラスの完全修飾名。

   モジュール

   カスタムクラスが含まれるモジュールの名前。

   必要に応じて、vault-options パラメーターを使用して、パスワード Vault のカスタムクラスを初期化できます。

例7.5 vault-options パラメーターを使用してカスタムクラスを初期化

```
/core-service=vault:add(code="custom.vault.implementation.CustomSecurityVault",
module="custom.vault.module", vault-options=[("KEYSTORE_URL" =>
"PATH_TO_KEYSTORE"), ("KEYSTORE_PASSWORD" => "MASKED_PASSWORD"),
("KEYSTORE_ALIAS" => "ALIAS"), ("SALT" => "SALT"),("ITERATION_COUNT" =>
"ITERATION_COUNT"), ("ENC_FILE_DIR" => "ENC_FILE_DIR")])
```

結果

パスワード vault のカスタム実装を使用して、マスクされた文字列が復号化されるよう JBoss EAP 6 が設定されます。

バグの報告

7.8. パスワード VAULT に SENSITIVE 文字列を保存します。

概要

プレーンテキストの設定ファイルにパスワードやその他の機密文字列を含めると、セキュリティーリスクが伴います。これらの文字列は、パスワード vault に保存します。パスワード vault では、これらの文字列は、マスクされた形式で設定ファイル、管理 CLI コマンド、およびアプリケーションで参照できます。

機密文字列は、各パラメーターの値の入力を求める対話形式でパスワード vault に保存するか、コマンドにすべてのパラメーターの値を提供する非対話形式で保存できます。各メソッドで同じ結果が表示されるため、任意の方法を選択します。すべてのパラメーターの説明は、「パスワード vault の初期化」 を参照してください。

前提条件

- 「パスワード vault を使用するよう JBoss EAP 6 を設定」

手順7.7 対話式での機密文字列の保存

各パラメーターの値の入力を求めるプロンプトを出す場合は、この方法を使用します。

1. パスワード vault コマンドの実行

   オペレーティングシステムのコマンドラインインターフェースを起動し、パスワード Vault コマンドを実行します。Red Hat Enterprise Linux または同様のオペレーティングシステムでは *EAP_HOME*/bin/vault.sh、Microsoft Windows Server では *EAP_HOME*\bin\vault.bat を使用します。新しい対話セッションを開始するには、0 (ゼロ) と入力します。

2. パスワード vault に関するパラメーターの入力

   プロンプトに従って必要な認証パラメーターを入力します。これらの値は、パスワード vault の作成時に提供された値と一致している必要があります。

注記

キーストアパスワードは、マスク形式ではなく、プレーンテキスト形式で指定する必要があります。

3. **機密文字列に関するパラメーターの入力**

   機密文字列の保存を開始する場合は 0 (ゼロ) を入力します。プロンプトに従って必要なパラメーターを入力します。

4. **マスクされた文字列に関する情報を書き留めておきます。**

   メッセージは標準出力に出力され、vault ブロック、属性名、マスクされた文字列、設定で文字列の使用に関するアドバイスを表示します。この情報は、安全な場所にメモしておいてください。出力例を以下に示します。

```
Vault Block:ds_Example1
Attribute Name:password
Configuration should be done as follows:
VAULT::ds_Example1::password::1
```

5. **対話式コンソールを終了します。**

   対話式コンソールを終了するには、3 （3 つ）を入力します。

例7.6 対話式での機密文字列の保存

```
========================================================================

 JBoss Vault

 JBOSS_HOME: EAP_HOME/jboss-eap-6.4

 JAVA: java

========================================================================

**********************************
****  JBoss Vault  ***************
**********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:11:18:46,086 INFO  [org.jboss.security] (management-
handler-thread - 4) PBOX0
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Oct 21, 2014 11:20:49 AM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in AS7 config file:
```

```
**********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
**********************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::   0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
0
Task: Store a secured attribute
Please enter secured attribute value (such as password):
Please enter secured attribute value (such as password) again:
Values match
Enter Vault Block:ds_Example1
Enter Attribute Name:password
Secured attribute value has been stored in vault.
Please make note of the following:
**********************************************
Vault Block:ds_Example1
Attribute Name:password
Configuration should be done as follows:
VAULT::ds_Example1::password::1
**********************************************
Please enter a Digit::   0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
```

手順7.8 機密文字列を非対話的に保存する

すべてのパラメーターの値を一度に指定する場合は、この方法を使用してください。

1. **オペレーティングシステムのコマンドラインインターフェースを起動し、パスワード Vault コマンドを実行します。Red Hat Enterprise Linux または同様のオペレーティングシステムでは *EAP_HOME*/bin/vault.sh、Microsoft Windows Server では *EAP_HOME*\bin\vault.bat を使用します。**

   プレースホルダーの値を実際の値に置き換えます。パラメーター *KEYSTORE_URL*、*KEYSTORE_PASSWORD*、および *KEYSTORE_ALIAS* の値は、パスワード vault の作成時に提供された値と一致している必要があります。

注記

キーストアパスワードは、マスク形式ではなく、プレーンテキスト形式で指定する必要があります。

```
EAP_HOME/bin/vault.sh --keystore KEYSTORE_URL --keystore-password
KEYSTORE_PASSWORD --alias KEYSTORE_ALIAS --vault-block VAULT_BLOCK --
attribute ATTRIBUTE --sec-attr SEC-ATTR --enc-dir ENC_FILE_DIR --iteration
ITERATION_COUNT --salt SALT
```

2. **マスクされた文字列に関する情報を書き留めておきます。**

メッセージは標準出力に出力され、vault ブロック、属性名、マスクされた文字列、設定で文字列の使用に関するアドバイスを表示します。この情報は、安全な場所にメモしておいてください。出力例を以下に示します。

```
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
VAULT::vb::password::1
```

例7.7 パスワード vault コマンドを非対話的に実行

```
EAP_HOME/bin/vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password vault22
--alias vault --vault-block vb --attribute password --sec-attr 0penS3sam3 --enc-dir
EAP_HOME/vault/ --iteration 120 --salt 1234abcd
```

コマンド出力

```
=========================================================================

  JBoss Vault

  JBOSS_HOME: EAP_HOME

  JAVA: java

=========================================================================

Oct 22, 2014 9:24:43 AM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Secured attribute value has been stored in vault.
Please make note of the following:
********************************************
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
VAULT::vb::password::1
********************************************
```

```
Vault Configuration in AS7 config file:
************************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="vault22"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/vault/"/>
</vault><management> ...
************************************************
```

結果

機密文字列はパスワード Vault に保存され、マスクされた形式で設定ファイル、管理 CLI コマンド、およびアプリケーションで使用できます。

バグの報告

7.9. 暗号化した機密文字列を設定で使用

前提条件

- **「パスワード Vault に Sensitive 文字列を保存します。」**

暗号化されている機密文字列は、マスクされた形式で設定ファイルまたは管理 CLI コマンドで使用できます。式の指定も可能です。

特定のサブシステム内で式が許可されているかどうかを確認するには、そのサブシステムに対して以下の管理 CLI コマンドを実行します。

注記

管理対象ドメインのコマンドに、プレフィックス /host=*HOST_NAME* を追加します。

```
/core-service=SUBSYSTEM:read-resource-description(recursive=true)
```

例7.8 管理サブシステムのすべてのリソースの説明の一覧表示

```
/core-service=management:read-resource-description(recursive=true)
```

このコマンドの実行の出力から、**expressions-allowed** パラメーターの値を探します。**true** の場合、このサブシステムの設定内で式を使用できます。

以下の構文を使用して、プレーンテキスト文字列をマスクされたフォームに置き換えます。

```
${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::MASKED_STRING}
```

例7.9 マスクされたフォームでのパスワードを使用したデータソース定義

この例では、**vault** ブロックは **ds_ExampleDS** で、属性は **password** です。

```
...
  <subsystem xmlns="urn:jboss:domain:datasources:1.0">
    <datasources>
      <datasource jndi-name="java:jboss/datasources/ExampleDS" enabled="true" use-java-
context="true" pool-name="H2DS">
        <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
        <driver>h2</driver>
        <pool></pool>
        <security>
          <user-name>sa</user-name>
          <password>${VAULT::ds_ExampleDS::password::1}</password>
        </security>
      </datasource>
      <drivers>
        <driver name="h2" module="com.h2database.h2">
          <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
        </driver>
      </drivers>
    </datasources>
  </subsystem>
...
```

バグの報告

7.10. アプリケーションで暗号化した機密文字列の使用

前提条件

- 「パスワード Vault に Sensitive 文字列を保存します。」

パスワード vault に保存されている暗号化された文字列は、アプリケーションのソースコードで使用できます。

例7.10 vault されたパスワードを使用するサーブレット

この例は、サーブレットのソースコードの抜粋です。これは、プレーンテキストのパスワードではなく、データソース定義でのマスクされたパスワードの使用を示しています。プレーンテキストバージョンはコメントアウトされているため、違いがわかります。

```
/*@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "sa",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)*/
@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "VAULT::DS::thePass::1",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)
```

バグの報告

7.11. 機密文字列がパスワード VAULT 内にあるかどうかを確認します。

概要

パスワード vault に機密文字列を保存または使用する前に、その文字列がすでに保存されているかどうかを確認すると便利です。

このチェックは、各パラメーターの値の入力を求める対話形式または、すべてのパラメーターの値をコンmmand 行に提供する非対話形式のいずれかで実行できます。各メソッドで同じ結果が表示されるため、任意の方法を選択します。

手順7.9 対話式での機密文字列の確認

各パラメーターの値の入力を求めるプロンプトを出す場合は、この方法を使用します。

1. **パスワード vault コマンドの実行**

   オペレーティングシステムのコマンドラインインターフェースを起動し、パスワード Vault コマンドを実行します。Red Hat Enterprise Linux または同様のオペレーティングシステムでは *EAP_HOME*/bin/vault.sh、Microsoft Windows Server では *EAP_HOME*\bin\vault.bat を使用します。新しい対話セッションを開始するには、0 (ゼロ) と入力します。

2. **パスワード vault に関するパラメーターの入力**

   プロンプトに従って必要な認証パラメーターを入力します。これらの値は、パスワード vault の作成時に提供された値と一致している必要があります。

注記

キーストアパスワードは、マスク形式ではなく、プレーンテキスト形式で指定する必要があります。

3. **1 (one)を入力して「Check whether a secured attribute exists」を選択します。**

4. **機密文字列を保存する vault ブロックの名前を入力します。**

5. **チェックする機密文字列の名前を入力します。**

結果

機密文字列が指定された vault ブロックに保存されている場合は、以下のような確認メッセージが表示されます。

```
A value exists for (VAULT_BLOCK, ATTRIBUTE)
```

機密文字列が指定されたブロックに保存されて いない 場合は、以下のようなメッセージが出力されます。

```
No value has been store for (VAULT_BLOCK, ATTRIBUTE)
```

例7.11 対話式での機密文字列の確認

```
================================================================================

  JBoss Vault

  JBOSS_HOME: EAP_HOME

  JAVA: java

================================================================================

**********************************
****  JBoss Vault  ***************
**********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Oct 22, 2014 12:53:56 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in AS7 config file:
*********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
*********************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::   0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
```

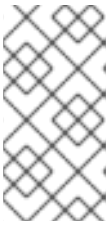
```
1
Task: Verify whether a secured attribute exists
Enter Vault Block:vb
Enter Attribute Name:password
A value exists for (vb, password)
Please enter a Digit::   0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
```

手順7.10 非対話的な文字列の確認

すべてのパラメーターの値を一度に指定する場合は、この方法を使用してください。すべてのパラメーターの説明は、「パスワード vault の初期化」 を参照してください。

- **オペレーティングシステムのコマンドラインインターフェースを起動し、パスワード Vault コマンドを実行します。Red Hat Enterprise Linux または同様のオペレーティングシステムでは *EAP_HOME*/bin/vault.sh、Microsoft Windows Server では *EAP_HOME*\bin\vault.bat を使用します。**

  プレースホルダーの値を実際の値に置き換えます。パラメーター *KEYSTORE_URL*、*KEYSTORE_PASSWORD-password*、および *KEYSTORE_ALIAS* の値は、パスワード vault の作成時に提供された値と一致している必要があります。

  注記

  キーストアパスワードは、マスク形式ではなく、プレーンテキスト形式で指定する必要があります。

```
EAP_HOME/bin/vault.sh --keystore KEYSTORE_URL --keystore-password
KEYSTORE_PASSWORD --alias KEYSTORE_ALIAS --check-sec-attr --vault-block
VAULT_BLOCK --attribute ATTRIBUTE --enc-dir ENC_FILE_DIR --iteration
ITERATION_COUNT --salt SALT
```

結果

機密文字列が指定された vault ブロックに保存されている場合は、以下のメッセージが出力されます。

```
Password already exists.
```

値が指定のブロックに保存されて いない 場合は、以下のメッセージが出力されます。

> Password doesn't exist.

バグの報告

7.12. パスワード VAULT からの機密文字列の削除

概要

セキュリティー上の理由から、機密文字列が不要になった場合は、パスワード vault から削除することが推奨されます。たとえば、アプリケーションの使用を停止する場合、データソース定義で使用される機密文字列を同時に削除する必要があります。

前提条件

パスワード vault から機密文字列を削除する前に、JBoss EAP の設定で使用されるかどうかを確認します。これを行う方法の 1 つとして、'grep' ユーティリティーを使用して、マスクされた文字列のインスタンスを検索する方法があります。Red Hat Enterprise Linux（および同様のオペレーティングシステム）では、grep はデフォルトでインストールされますが、Microsoft Windows Server の場合は手動でインストールする必要があります。

Password Vault ユーティリティーは、インタラクティブと非対話の 2 つのモードを提供します。インタラクティブモードでは、各パラメーターの値の入力を求めるプロンプトが出されます。この場合、非対話モードでは、1 つのコマンドですべてのパラメーターの値を提供する必要があります。

手順7.11 対話式での機密文字列の削除

各パラメーターの値の入力を求めるプロンプトを出す場合は、この方法を使用します。

1. **パスワード vault コマンドの実行**

   オペレーティングシステムのコマンドラインインターフェースを起動し、*EAP_HOME*/bin/vault.sh （Red Hat Enterprise Linux および同様のオペレーティングシステム）または *EAP_HOME*\bin\vault.bat （Microsoft Windows Server 上）を実行します。0（ゼロ）を入力して、新しい対話セッションを開始します。

2. **認証の詳細**

   プロンプトに従って必要な認証パラメーターを入力します。これらの値は、パスワード vault の作成時に提供された値と一致している必要があります。

注記

キーストアパスワードは、マスク形式ではなく、プレーンテキスト形式で指定する必要があります。

3. **2 を入力して、セキュアな属性を削除 するオプションを選択します。**

4. **機密文字列を保存する vault ブロックの名前を入力します。**

5. **削除する機密文字列の名前を入力します。**

結果

機密文字列が正常に削除されると、以下のような確認メッセージが表示されます。

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] has been successfully removed from vault
```

機密文字列が 削除されない場合は、以下のようなメッセージが出力されます。

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] was not removed from vault, check whether it exist
```

例7.12 対話式での機密文字列の削除

```
**********************************
****  JBoss Vault  ***************
**********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Dec 23, 2014 1:40:56 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
```

```
Vault Configuration in configuration file:
************************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
************************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::  0: Store a secured attribute  1: Check whether a secured attribute
exists  2: Remove secured attribute  3: Exit
2
Task: Remove secured attribute
Enter Vault Block:craft
Enter Attribute Name:password
Secured attribute [craft::password] has been successfully removed from vault
```

手順7.12 非対話的な文字列の削除

すべてのパラメーターの値を一度に指定する場合は、この方法を使用してください。すべてのパラメーターの説明は、「パスワード vault の初期化」 を参照してください。

- オペレーティングシステムのコマンドラインインターフェースを起動し、パスワード Vault コマンドを実行します。Red Hat Enterprise Linux または同様のオペレーティングシステムでは *EAP_HOME*/bin/vault.sh、Microsoft Windows Server では *EAP_HOME*\bin\vault.bat を使用します。

  プレースホルダーの値を実際の値に置き換えます。パラメーター *KEYSTORE_URL*、*KEYSTORE_PASSWORD*、および *KEYSTORE_ALIAS* の値は、パスワード vault の作成時に提供された値と一致している必要があります。

  注記

  キーストアパスワードは、マスク形式ではなく、プレーンテキスト形式で指定する必要があります。

```
EAP_HOME/bin/vault.sh --keystore KEYSTORE_URL --keystore-password
KEYSTORE_PASSWORD --alias KEYSTORE_ALIAS --remove-sec-attr --vault-block
VAULT_BLOCK --attribute ATTRIBUTE --enc-dir ENC_FILE_DIR --iteration
ITERATION_COUNT --salt SALT
```

結果

機密文字列が正常に削除されると、以下のような確認メッセージが表示されます。

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] has been successfully removed from vault
```

機密文字列が 削除されない場合は、以下のようなメッセージが出力されます。

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] was not removed from vault, check whether it exist
```

例7.13 非対話的な文字列の削除

```
./vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password vault22 --alias
vault --remove-sec-attr --vault-block craft --attribute password --enc-dir ../vault/ --iteration
120 --salt 1234abcd
================================================================================

  JBoss Vault

  JBOSS_HOME: EAP_HOME

  JAVA: java

================================================================================

Dec 23, 2014 1:54:24 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Secured attribute [craft::password] has been successfully removed from vault
```

バグの報告

パート III. セキュアなアプリケーションの開発

第8章 セキュリティーの概要

8.1. アプリケーションセキュリティー

アプリケーションのセキュリティー保護は、各アプリケーション開発者にとって多面的かつ重要な問題となります。JBoss EAP 6 は、以下の機能を含む、セキュアなアプリケーションの作成に必要なすべてのツールを提供します。

- 「認証」
- 「認可について」
- 「セキュリティー監査」
- 「セキュリティーマッピング」
- 「宣言型セキュリティー」
- 「EJB メソッドパーミッションについて」
- 「EJB セキュリティーアノテーションについて」

「アプリケーションでのセキュリティードメインの使用」 も参照してください。

バグの報告

8.2. 宣言型セキュリティー

宣言型セキュリティー は、コンテナーを使用してセキュリティーを管理することでアプリケーションコードからセキュリティー上の懸念を分離する方法です。コンテナーはファイルパーミッション、またはユーザー、グループ、およびロールを基に承認システムを提供します。通常、このアプローチはアプリケーション自体 *に* セキュリティーの責任をすべて与えるプログラムによるセキュリティーに適しています。

JBoss EAP 6 はセキュリティードメインを介して宣言型セキュリティーを提供します。

バグの報告

8.2.1. Java EE の宣言的セキュリティーの概要

Java EE セキュリティーモデルは、ビジネスコンポーネントにセキュリティーを埋め込むのではなく、標準の XML 記述子でセキュリティーロールおよびパーミッションを説明する宣言的です。これにより、セキュリティーは、コンポーネントのビジネスロジックの特有の要素よりもコンポーネントがデプロイされる機能が強化されるため、ビジネスレベルのコードからセキュリティーが分離されます。たとえば、銀行のアカウントへのアクセスに使用する Automation Teller Machine(ATM)について見てみましょう。セキュリティー要件、ロール、およびパーミッションは、BTM が置かれているアカウントの管理対象を基にして銀行のアカウントへのアクセス方法によって異なります。

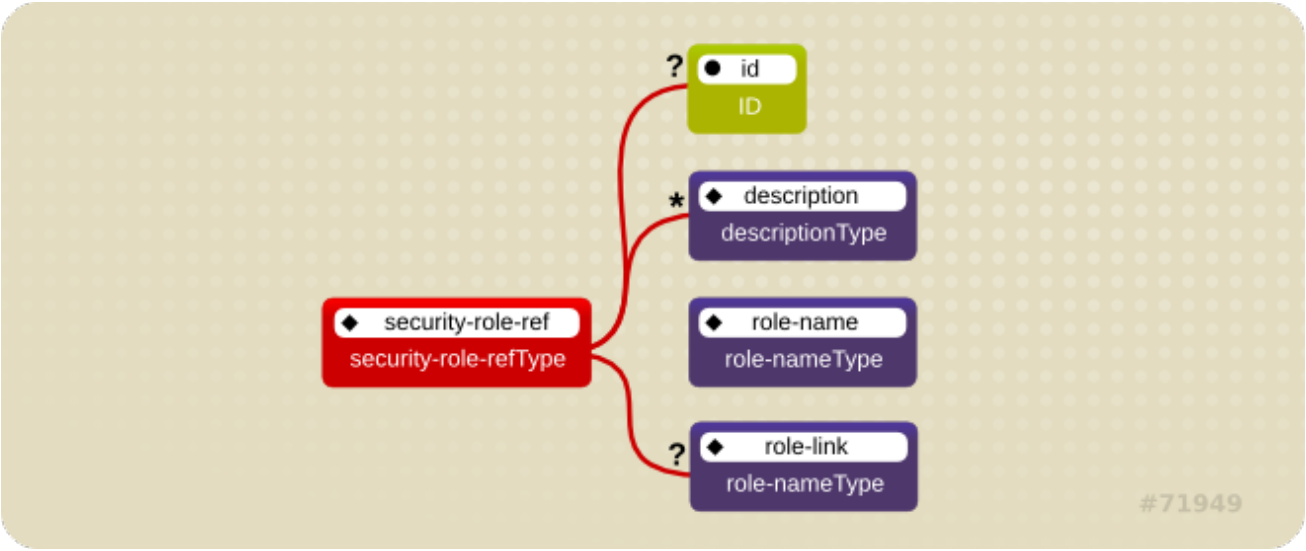
Java EE アプリケーションのセキュリティーを保護することは、標準の Java EE デプロイメント記述子を使用したアプリケーションセキュリティー要件の仕様に基づいています。ejb-jar.xml および web.xml デプロイメント記述子を使用して、エンタープライズアプリケーションの EJB および Web コンポーネントへのアクセスをセキュアにすることができます。

バグの報告

8.2.2. セキュリティーリファレンス

Enterprise Java Bean(EJB)とサーブレットの両方が、1 つ以上の <security-role-ref> 要素を宣言できます。

図8.1 セキュリティーロールのリファレンスモデル



この要素は、コンポーネントが <role-name> 要素の role-nameType 属性値を isCallerInRole(String) メソッドの引数として使用していることを宣言します。isCallerInRole メソッドを使用して、コンポーネントは <security-role-ref> または <role-name> 要素で宣言されたロールにあるかを確認できます。<role-name> 要素の値は、<security-role> 要素を介して <role-link> 要素にリンクする必要があります。isCallerInRole の一般的な用途は、ロールベースの <method-permissions> 要素を使用して定義できないセキュリティーチェックを実行することです。

例8.1 ejb-jar.xml descriptor fragment

```
<!-- A sample ejb-jar.xml fragment -->
 <ejb-jar>
  <enterprise-beans>
   <session>
    <ejb-name>ASessionBean</ejb-name>
    ...
    <security-role-ref>
     <role-name>TheRoleICheck<role-name>
      <role-link>TheApplicationRole</role-link>
    </security-role-ref>
   </session>
  </enterprise-beans>
 ...
 </ejb-jar>
```

注記

この抜粋は例です。デプロイメントでは、本セクションの要素にはロール名と、EJB デプロイメントに関連するリンクが含まれている必要があります。

例8.2 web.xml 記述子フラグメント

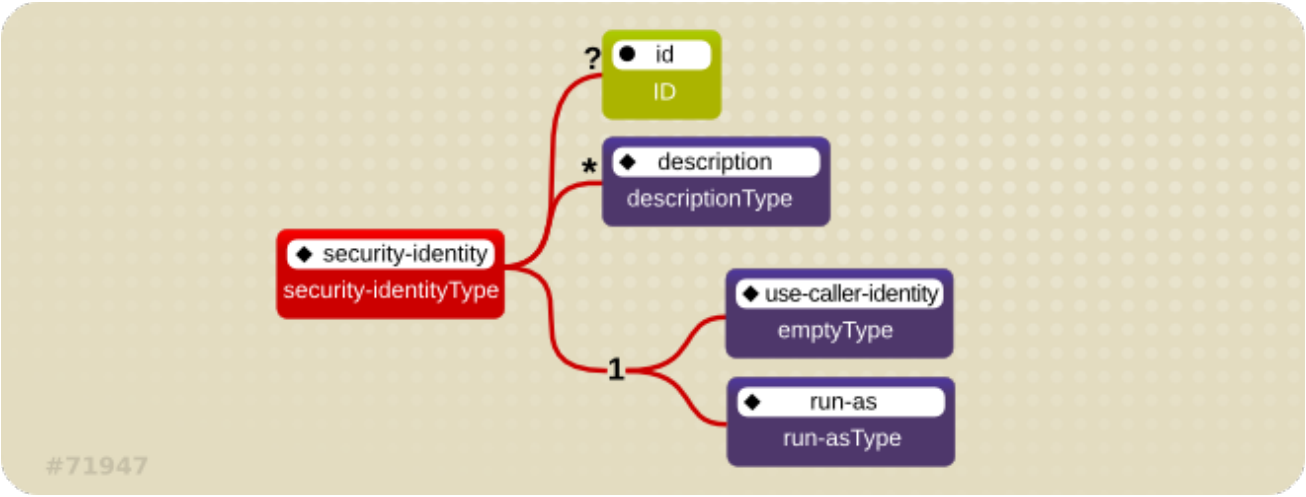
```
<web-app>
  <servlet>
    <servlet-name>AServlet</servlet-name>
    ...
    <security-role-ref>
      <role-name>TheServletRole</role-name>
      <role-link>TheApplicationRole</role-link>
    </security-role-ref>
  </servlet>
    ...
</web-app>
```

バグの報告

8.2.3. セキュリティーアイデンティティー

Enterprise Java Bean(EJB)は、<security-identity> 要素を使用してコンポーネントでメソッドを呼び出すときに別の EJB を使用する必要があります。

図8.2 Java EE Security Identity データモデル



呼び出しアイデンティティーは、現在の呼び出し元または特定のロールになります。アプリケーションアセンブラーは、<security-identity> 子要素で <use-caller-identity> 要素を使用します。これは、現在の呼び出し元のアイデンティティーを EJB によるメソッド呼び出しのセキュリティーアイデンティティーとして伝播する必要があることを示します。呼び出し元のアイデンティティーの伝搬は、明示的な <security-identity> 要素宣言がない場合に使用されるデフォルトです。

または、アプリケーションアセンブラーは <run-as> または <role-name> 子要素を使用し

て、<role-name> 要素値によって提供される特定のセキュリティーロールを EJB によるメソッド呼び出しのセキュリティーアイデンティティーとして使用する必要があると指定できます。

これにより、EJBContext.getCallerPrincipal() メソッドが示すように呼び出し元のアイデンティティーは変更されません。代わりに、呼び出し元のセキュリティーロールは、<run-as> または <role-name> 要素の値で指定された単一のロールに設定されます。

<run-as> 要素のユースケースの 1 つは、外部クライアントが内部 EJB にアクセスできないようにします。この動作は、外部クライアントに割り当てられないロールへのアクセスを制限する内部 EJB <method-permission> 要素を割り当てます。その後、内部 EJB を使用する EJB は、restricted ロールと同等の <run-as> または <role-name> で設定されます。以下の記述子フラグメントは、<security-identity> 要素の使用例を示しています。

```
<ejb-jar>
   <enterprise-beans>
      <session>
         <ejb-name>ASessionBean</ejb-name>
         <!-- ... -->
         <security-identity>
            <use-caller-identity/>
         </security-identity>
      </session>
      <session>
         <ejb-name>RunAsBean</ejb-name>
         <!-- ... -->
         <security-identity>
            <run-as>
               <description>A private internal role</description>
               <role-name>InternalRole</role-name>
            </run-as>
         </security-identity>
      </session>
   </enterprise-beans>
   <!-- ... -->
</ejb-jar>
```

<run-as> を使用して送信呼び出しに特定のロールを割り当てると、anonymous という名前のプリンシパルがすべて送信呼び出しに割り当てられます。呼び出しに別のプリンシパルを関連付ける場合は、<run-as-principal> を jboss-ejb3.xml ファイルの Bean に関連付ける必要があります。以下のフラグメントは、前の例の internal という名前のプリンシパルを RunAsBean に関連付けます。

```
<session>
   <ejb-name>RunAsBean</ejb-name>
   <security-identity>
      <run-as-principal>internal</run-as-principal>
```

```
    </security-identity>
</session>
```

<run-as> 要素は web.xml ファイルのサーブレット定義でも利用可能です。以下の例は、InternalRole ロールをサーブレットに割り当てる方法を示しています。

```
<servlet>
  <servlet-name>AServlet</servlet-name>
  <!-- ... -->
  <run-as>
     <role-name>InternalRole</role-name>
  </run-as>
</servlet>
```

このサーブレットからの呼び出しは、匿名 プリンシパル に関連付けられます。<run-as-principal> 要素は jboss-web.xml ファイルで利用でき、run-as ロールとともに特定のプリンシパルを割り当てます。以下の例は、internal という名前のプリンシパルを上記のサーブレットに関連付ける方法を示しています。

```
<servlet>
  <servlet-name>AServlet</servlet-name>
  <run-as-principal>internal</run-as-principal>
</servlet>
```

バグの報告

8.2.4. セキュリティーロール

security-role-ref または security-identity 要素によって参照されるセキュリティーロール名は、アプリケーションの宣言されたロールの 1 つにマップする必要があります。アプリケーションアセンブラーは、security-role 要素を宣言して論理セキュリティーロールを定義します。role-name の値は、Administrator、MarketedManager などの論理アプリケーションのロール名です。

Java EE 仕様では、デプロイメント記述子のセキュリティーロールを使用してアプリケーションの論理セキュリティービューを定義することが重要です。Java EE デプロイメント記述子で定義されたロールは、ターゲットのエンタープライズ環境に存在するユーザーグループ、ユーザー、プリンシパル、およびその他の概念と混同しないようにしてください。デプロイメント記述子ロールは、アプリケーションのドメイン固有の名前を使用してアプリケーションを構築します。たとえば、銀行取引アプリケーションは BankManager、Teler、Customer などのロール名を使用する場合があります。

JBoss EAP では、security-role 要素は、コンポーネントロールが参照する論理ロールに security-role-ref/role-name の値をマップするために使用されます。ユーザーに割り当てられたロールは、アプリケーションのセキュリティーマネージャーの動的な機能です。JBoss では、メソッドパーミッションを宣言するために security-role 要素の定義は必要ありません。ただし、security-role 要素の仕様は、アプリケーションサーバーとデプロイメント記述子のメンテナンスにおける移植性を確保するために依然として推奨されています。

例8.3 security-role 要素の使用を示す ejb-jar.xml 記述子フラグメント。

```
<!-- A sample ejb-jar.xml fragment -->
<ejb-jar>
   <assembly-descriptor>
      <security-role>
         <description>The single application role</description>
         <role-name>TheApplicationRole</role-name>
      </security-role>
   </assembly-descriptor>
</ejb-jar>
```

例8.4 security-role 要素の使用を示す web.xml 記述子のフラグメントの例。

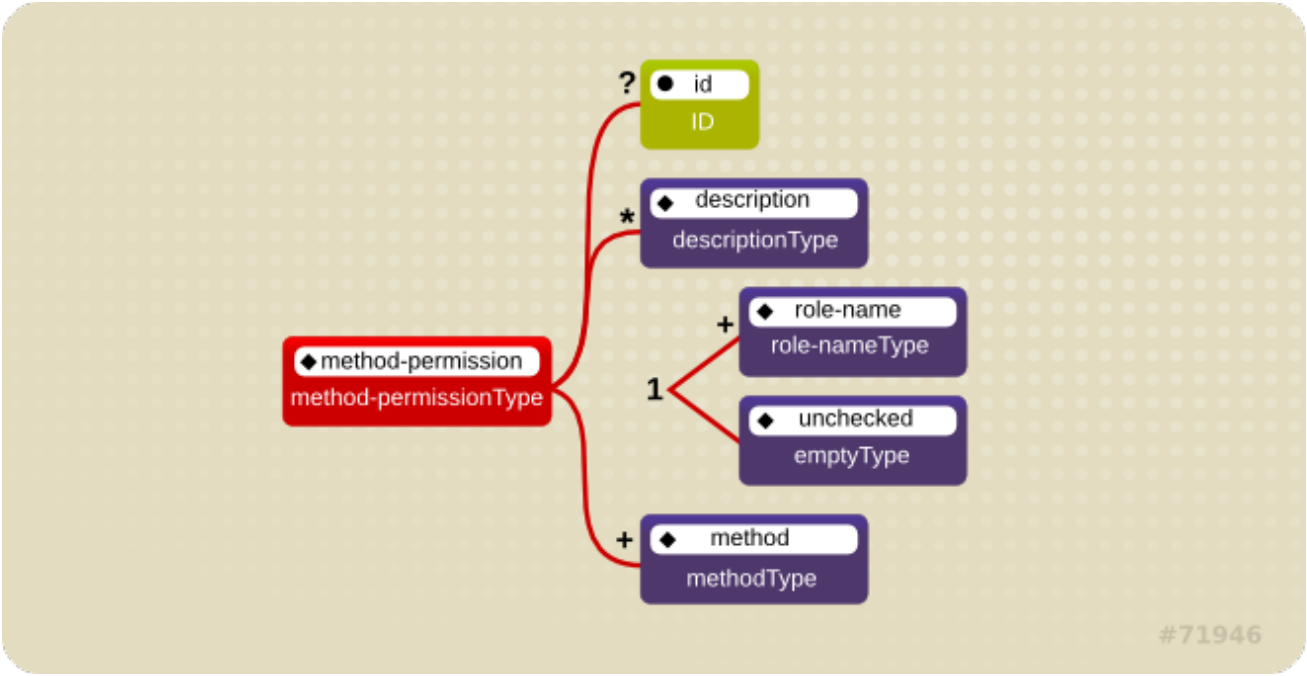
```
<!-- A sample web.xml fragment -->
<web-app>
  <security-role>
    <description>The single application role</description>
    <role-name>TheApplicationRole</role-name>
  </security-role>
</web-app>
```

バグの報告

8.2.5. EJB メソッドパーミッション

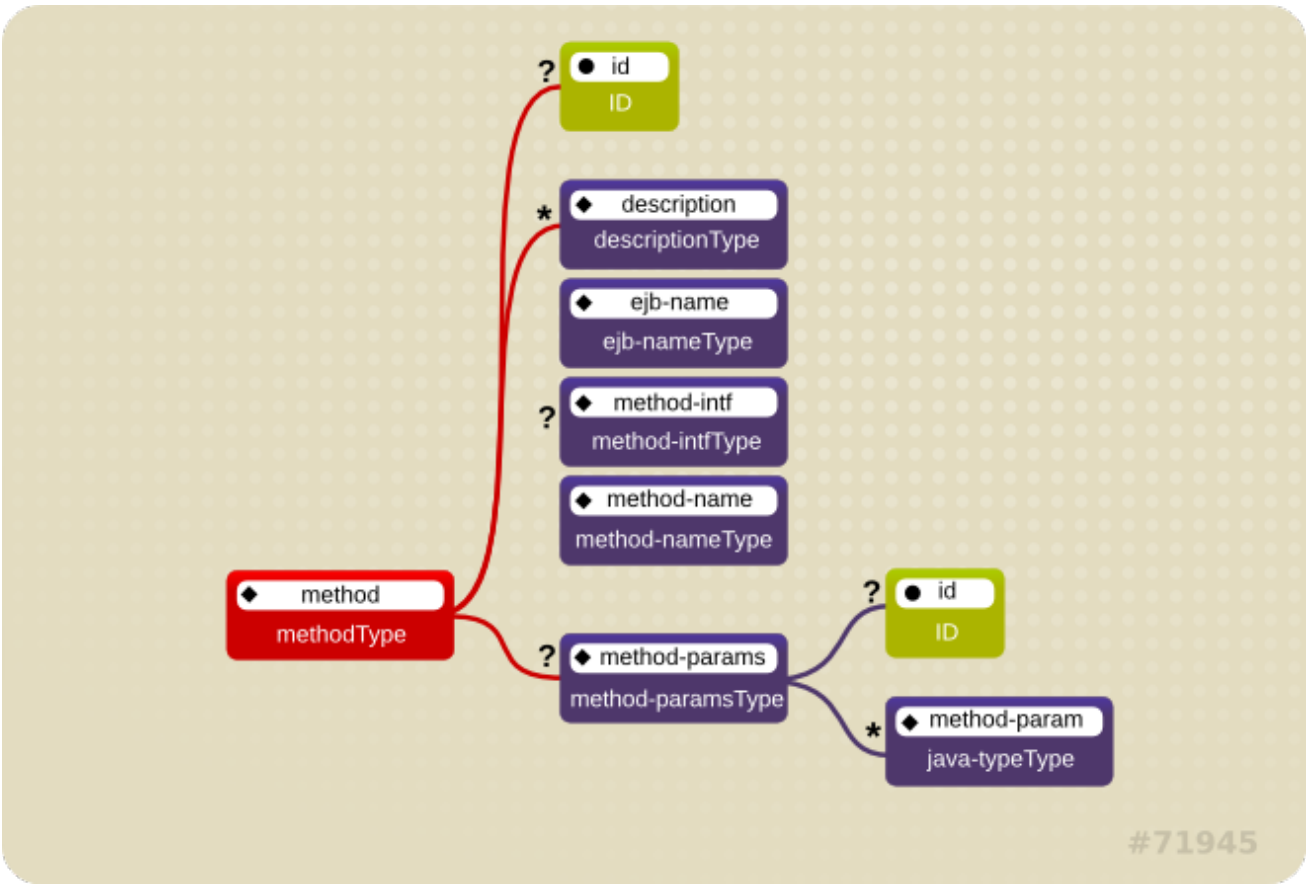
アプリケーションアセンブラーは、method-permission 要素宣言を使用して EJB のホームおよびリモートインターフェースメソッドを呼び出すことができるロールを設定できます。

図8.3 Java EE メソッドパーミッション要素



各 method-permission 要素には、メソッド子要素によって識別される EJB メソッドへのアクセスが許可される論理ロールを定義する 1 つ以上の role-name 子要素が含まれます。role-name 要素の代わりに 未 確認要素を指定して、認証されたユーザーがメソッド子要素で識別されるメソッドにアクセスできることを宣言することもできます。さらに、exclude-list 要素を持つメソッドにアクセスできないように宣言することもできます。EJB が method-permission 要素を使用してロールによってアクセス可能であることが宣言されていないメソッドがある場合、EJB メソッドはデフォルトで使用から除外されます。これは、exclude-list にメソッドのデフォルトと同等です。

図8.4 Java EE メソッド要素



メソッド要素宣言には、サポートされるスタイルが 3 つあります。

1 つ目は、名前付きエンタープライズ Bean のすべてのホームおよびコンポーネントインターフェースメソッドを参照するために使用されます。

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

2 つ目のスタイルは、名前付きエンタープライズ Bean のホームまたはコンポーネントインターフェースの特定メソッドを参照するために使用されます。

```
  <method>
    <ejb-name>EJBNAME</ejb-name>
    <method-name>METHOD</method-name>
  </method>
```

オーバーロード名が同じ複数のメソッドがある場合、このスタイルはオーバーロードされたすべてのメソッドを参照します。

3 つ目のスタイルは、オーバーロード名を持つメソッドのセット内で指定されたメソッドを参照するために使用されます。

```
<method>
  <ejb-name>EJBNAME</ejb-name>
  <method-name>METHOD</method-name>
  <method-params>
    <method-param>PARAMETER_1</method-param>
    <!-- ... -->
    <method-param>PARAMETER_N</method-param>
  </method-params>
</method>
```

メソッドは、指定されたエンタープライズ bean のホームまたはリモートインターフェースで定義する必要があります。method-param 要素値は、対応するメソッドパラメーター型の完全修飾名です。同じオーバーロードされた署名を持つ複数のメソッドがある場合、パーミッションは一致するオーバーロードされたすべてのメソッドに適用されます。

オプションの method-intf 要素を使用すると、エンタープライズ Bean のホームインターフェースとリモートインターフェースの両方で定義される名前と署名を持つメソッドを区別することができます。

例8.5「method-permission 要素の使用を示す ejb-jar.xml 記述子フラグメント。」 method-permission 要素の使用の完全な例を提供します。

例8.5 method-permission 要素の使用を示す ejb-jar.xml 記述子フラグメント。

```
<ejb-jar>
  <assembly-descriptor>
    <method-permission>
      <description>The employee and temp-employee roles may access any
        method of the EmployeeService bean </description>
      <role-name>employee</role-name>
      <role-name>temp-employee</role-name>
      <method>
        <ejb-name>EmployeeService</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
    <method-permission>
      <description>The employee role may access the findByPrimaryKey,
        getEmployeeInfo, and the updateEmployeeInfo(String) method of
        the AardvarkPayroll bean </description>
      <role-name>employee</role-name>
      <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>findByPrimaryKey</method-name>
```

```
            </method>
            <method>
               <ejb-name>AardvarkPayroll</ejb-name>
               <method-name>getEmployeeInfo</method-name>
            </method>
            <method>
               <ejb-name>AardvarkPayroll</ejb-name>
               <method-name>updateEmployeeInfo</method-name>
               <method-params>
                  <method-param>java.lang.String</method-param>
               </method-params>
            </method>
         </method-permission>
         <method-permission>
            <description>The admin role may access any method of the
               EmployeeServiceAdmin bean </description>
            <role-name>admin</role-name>
            <method>
               <ejb-name>EmployeeServiceAdmin</ejb-name>
               <method-name>*</method-name>
            </method>
         </method-permission>
         <method-permission>
            <description>Any authenticated user may access any method of the
               EmployeeServiceHelp bean</description>
            <unchecked/>
            <method>
               <ejb-name>EmployeeServiceHelp</ejb-name>
               <method-name>*</method-name>
            </method>
         </method-permission>
         <exclude-list>
            <description>No fireTheCTO methods of the EmployeeFiring bean may be
               used in this deployment</description>
            <method>
               <ejb-name>EmployeeFiring</ejb-name>
               <method-name>fireTheCTO</method-name>
            </method>
         </exclude-list>
      </assembly-descriptor>
   </ejb-jar>
```

バグの報告

8.2.6. エンタープライズ Bean セキュリティーアノテーション

エンタープライズ Bean はアノテーションを使用して、アプリケーションのセキュリティーやその他の側面に関する情報をデプロイヤーに渡します。アノテーションで指定された場合、デプロイヤーはアプリケーションに適切なエンタープライズ bean セキュリティーポリシーを設定できます（アノテーションで指定された場合）。

デプロイメント記述子に明示的に指定されるメソッドの値はすべて、アノテーションの値を上書きします。メソッドの値がデプロイメント記述子で指定されていない場合は、アノテーションを使用して設定された値が使用されます。上書きの粒度は、メソッドごとに決定されます。

セキュリティーに対応するアノテーションや、エンタープライズ Bean で使用できるアノテーションには以下が含まれます。

@DeclareRoles

コードで宣言された各セキュリティーロールを宣言します。ロールの設定に関する情報は、「『Java EE 6 チュートリアル』 で セキュリティーロールの宣言による承認ユーザーの指定」を参照してください。

@RolesAllowed、*@PermitAll*、および *@DenyAll*

アノテーション のメソッドパーミッションを指定します。アノテーションメソッドパーミッション の設定に関する詳細は、「『Java EE 6 チュートリアル』 による承認済みのユーザーの指定」を参照してください。

@RunAs

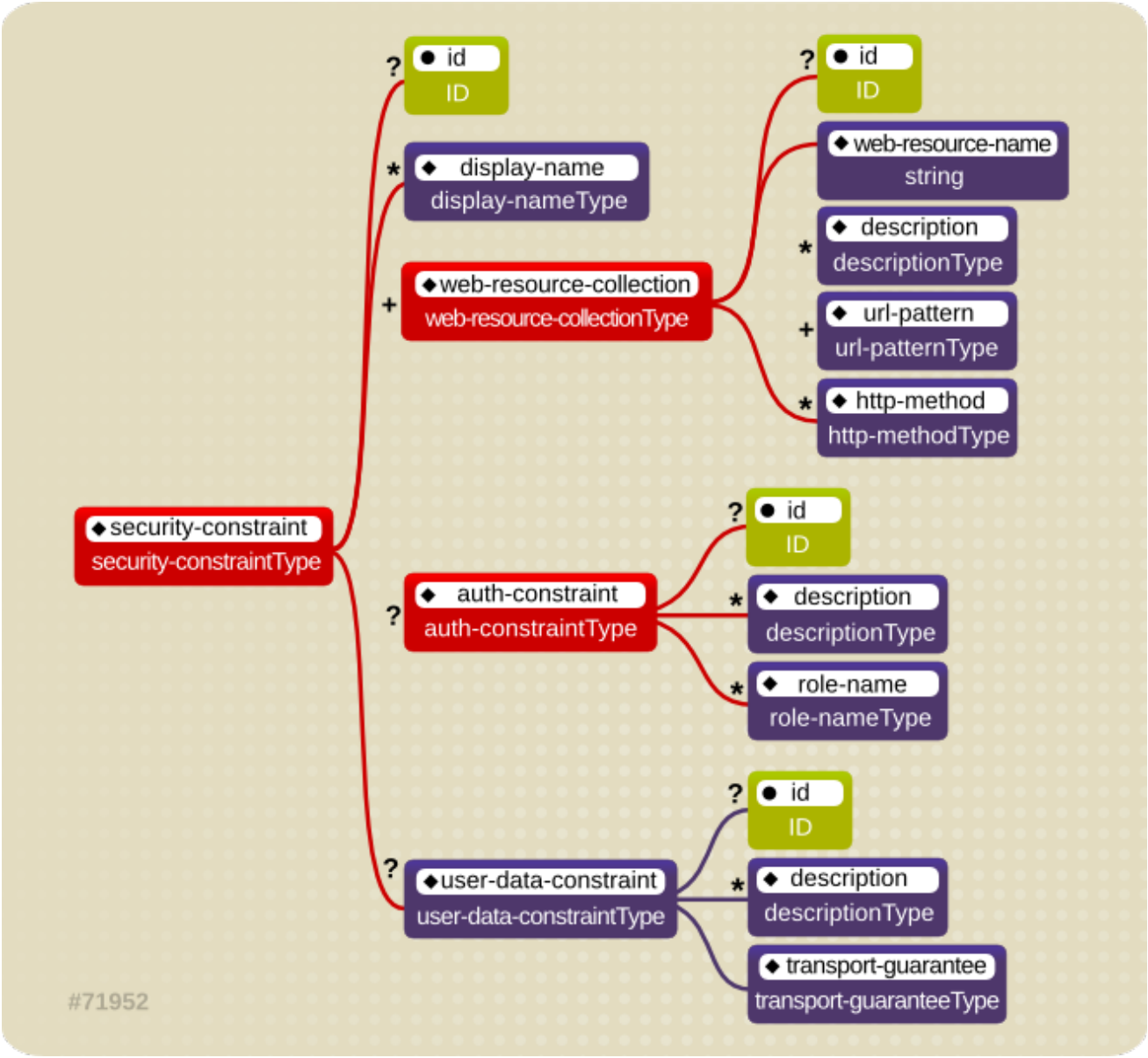
コンポーネントの伝播セキュリティーアイデンティティーを設定します。アノテーションを使用した伝播されたセキュリティーアイデンティティー の設定に関する詳細は、「『Java EE 6 チュートリアル』 によるセキュリティーアイデンティティー(Run-As) 」のチュートリアルを参照してください。

バグの報告

8.2.7. Web コンテンツのセキュリティー制約

Web アプリケーションでセキュリティーは、保護されているコンテンツを識別する URL パターンによるコンテンツへのアクセスが許可されるロールで定義されます。この一連の情報は、web.xml security-constraint 要素を使用して宣言されます。

図8.5 Web コンテンツのセキュリティー制約

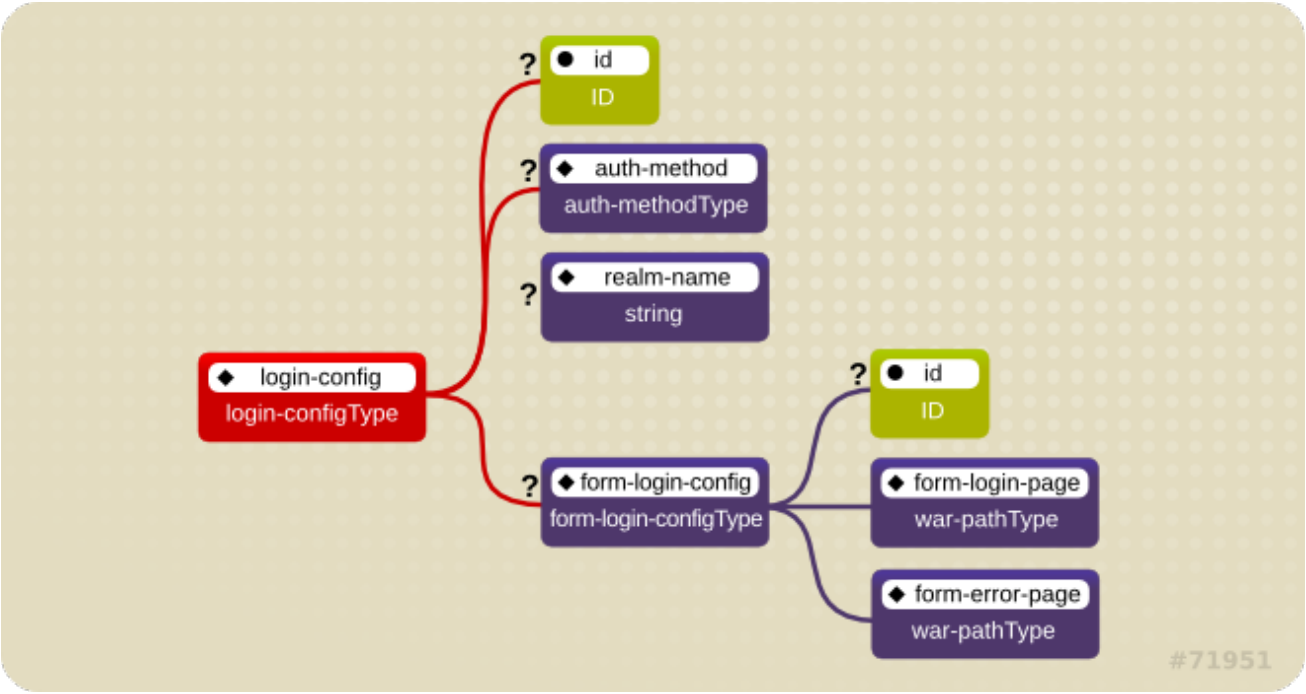


セキュリティー保護するコンテンツは、1 つ以上の <web-resource-collection> 要素を使用して宣言されます。各 <web-resource-collection> 要素には、任意の一連の <url-pattern> 要素が含まれ、その後に任意の一連の <http-method> 要素が含まれます。<url-pattern> 要素の値は、リクエストがセキュアなコンテンツへのアクセスを試みるためにリクエスト URL が一致する必要がある URL パターンを指定します。<http-method> 要素の値は、許可する HTTP 要求のタイプを指定します。

任意の <user-data-constraint> 要素は、クライアントのサーバー接続へのトランスポート層の要件を指定します。要件は、コンテンツの整合性（通信プロセスでデータの改ざんを生じさせる）や、機密性（トランジション中にローディング）を行う場合に役立ちます。<transport-guarantee> 要素の値は、クライアントとサーバー間の通信を保護するレベルを指定します。この値は NONE、INTEGRAL、CONFIDENTIAL です。NONE の値は、アプリケーションにトランスポート保証を必要としないことを意味します。INTEGRAL の値は、転送時に変更できないように、アプリケーションとサーバー間で送信されるデータを送信する必要があることを意味します。CONFIDENTIAL の値は、他のエンティティーが送信の内容を認識できないようにする方法でデータを送信する必要があることを意味します。ほとんどの場合、INTEGRAL または CONFIDENTIAL フラグがある場合は、SSL の使用が必要であることを示しています。

任意の <login-config> 要素は、使用する認証方法、アプリケーションに使用するレルム名、フォームログインメカニズムで必要な属性を設定するために使用されます。

図8.6 Web ログインの設定



<auth-method> 子要素は、Web アプリケーションの認証メカニズムを指定します。承認制約で保護される Web リソースへのアクセス権を取得する前提条件として、ユーザーは設定済みのメカニズムを使用して認証する必要があります。有効な <auth-method> 値は、BASIC、DIGEST、FORM、SPNEGO、および CLIENT-CERT です。<realm-name> 子要素は、HTTP Basic およびダイジェスト承認で使用するレルム名を指定します。<form-login-config> 子要素はログインと、フォームベースのログインで使用するエラーページを指定します。<auth-method> 値が FORM ではない場合、form-login-config とその子要素は無視されます。

以下の設定例は、Web アプリケーションの /restricted パスで強制される URL に AuthorizedUser ロールが必要であることを示しています。必要なトランスポートの保証はなく、ユーザー ID の取得に使用される認証方法は BASIC HTTP 認証です。

例8.6 web.xml 記述子のフェンシング

```
<web-app>
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>Secure Content</web-resource-name>
         <url-pattern>/restricted/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
         <role-name>AuthorizedUser</role-name>
      </auth-constraint>
      <user-data-constraint>
         <transport-guarantee>NONE</transport-guarantee>
```

```
        </user-data-constraint>
    </security-constraint>
    <!-- ... -->
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>The Restricted Zone</realm-name>
    </login-config>
    <!-- ... -->
    <security-role>
        <description>The role required to access restricted content </description>
        <role-name>AuthorizedUser</role-name>
    </security-role>
</web-app>
```

バグの報告

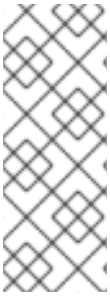
8.2.8. フォームベースの認証の有効化

フォームベースの認証では、ログイン用にカスタム JSP/HTML ページを柔軟に定義でき、ログイン時にエラーが発生する場合にユーザーにダイレクトする別のページを利用できます。

フォームベースの認証は、デプロイメント記述子 web.xml の <auth-method>FORM</auth-method> 要素に <login-config> を含めることで定義されます。ログインおよびエラーページは、以下のように <login-config> でも定義されています。

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

フォームベースの認証のある Web アプリケーションがデプロイされると、Web コンテナーは FormAuthenticator を使用してユーザーを適切なページに転送します。JBoss EAP はセッションプールを維持するため、各リクエストに認証情報を追加する必要はありません。FormAuthenticator がリクエストを受信すると、既存のセッションに対して org.apache.catalina.session.Manager をクエリーします。セッションが存在しない場合は、新しいセッションが作成されます。FormAuthenticator は、セッションの認証情報を検証します。



認証要求

各セッションは、ランダムな値から生成される 16 バイトの文字列であるセッション ID で識別されます。この値はデフォルトで /dev/urandom (Linux)から取得され、MD5 でハッシュ化されます。チェックはセッション ID の作成時に実行され、作成された ID が一意であることを確認します。

確認されると、セッション ID はクッキーの一部として割り当てられ、クライアントに戻ります。このクッキーは後続のクライアント要求で予想され、ユーザーセッションを識別するために使用されます。

クライアントに渡されるクッキーは、複数の任意の属性を持つ名前/値のペアです。識別子属性は JSESSIONID と呼ばれます。この値は、セッション ID の 16 進文字列です。このクッキーは永続的ではないように設定されています。つまり、クライアント側ではブラウザーが終了すると削除されます。サーバー側では、セッションが 30 分後に非アクティブになると有効期限が切れ、セッションオブジェクトとその認証情報情報が削除されます。

ユーザーがフォームベースの認証で保護されている Web アプリケーションにアクセスしようとするとします。FormAuthenticator はリクエストをキャッシュし、必要に応じて新しいセッションを作成し、ユーザーを login-config で定義されたログインページにリダイレクトします。（上記の例のコードでは、ログインページは login.html です。） 次に、ユーザーは提供された HTML フォームにユーザー名とパスワードを入力します。ユーザー名とパスワードは、j_security_check フォームアクションを介して FormAuthenticator に渡されます。

次に FormAuthenticator は、Web アプリケーションコンテキストに割り当てられたレルムに対してユーザー名とパスワードを認証します。JBoss Enterprise Application Platform では、レルムは JBossWebRealm です。認証に成功すると、FormAuthenticator はキャッシュから保存されたリクエストを取得し、ユーザーを元の要求にリダイレクトします。

注記

サーバーは、URI が /j_security_check で終わり、少なくとも j_username および j_password パラメーターが存在する場合のみ、フォーム認証要求を認識します。

バグの報告

第9章 アプリケーションセキュリティー

9.1. データソースセキュリティー

9.1.1. データソースセキュリティー

データソースセキュリティーとは、データソース接続のパスワードを暗号化したり分かりにくくすることを言います。これらのパスワードはプレーンテキストで設定ファイルに保存できますが、セキュリティーリスクが高くなります。

データソースセキュリティーの推奨ソリューションは、セキュリティードメインまたはパスワード vault を使用することです。以下には、各メソッドの例が含まれています。詳細は、セキュリティー『アーキテクチャー』 およびその他の JBoss EAP セキュリティードキュメントを参照してください。

例9.1 セキュリティードメインの例

```
 <security-domain name="DsRealm" cache-type="default">
  <authentication>
    <login-module code="ConfiguredIdentity" flag="required">
      <module-option name="userName" value="sa"/>
      <module-option name="principal" value="sa"/>
      <module-option name="password" value="sa"/>
    </login-module>
  </authentication>
</security-domain>
```

DsRealm ドメインはデータソースによって参照されます。

```
<datasources>
  <datasource jndi-name="java:jboss/datasources/securityDs"
    pool-name="securityDs">
    <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
      <driver>h2</driver>
      <new-connection-sql>select current_user()</new-connection-sql>
      <security>
        <security-domain>DsRealm</security-domain>
      </security>
    </datasource>
</datasources>
```

例9.2 パスワード vault の例

```
<security>
  <user-name>admin</user-name>
```

```
<password>${VAULT::ds_ExampleDS::password::N2NhZDYzOTMtNWE0OS00ZGQ0LWE4M
mEtMWNlMDMyNDdmNmI2TElORV9CUkVBS3ZhdWx0}</password>
</security>
```

バグの報告

9.2. EJB アプリケーションセキュリティー

9.2.1. セキュリティーアイデンティティー

9.2.1.1. EJB セキュリティーアイデンティティー

EJB は、他のコンポーネントでメソッドを呼び出す際に使用するアイデンティティーを指定できます。これは、EJB の *セキュリティーアイデンティティー*（*呼び出し*ID とも呼ばれます）です。

デフォルトでは、EJB は独自の呼び出し元アイデンティティーを使用します。アイデンティティーは、特定のセキュリティーロールに設定することもできます。特定のセキュリティーロールの使用は、セグメント化されたセキュリティーモデルを構築する場合に便利です。たとえば、内部 EJB のみへのコンポーネントセットへのアクセスを制限します。

バグの報告

9.2.1.2. EJB のセキュリティーアイデンティティーの設定

EJB のセキュリティーアイデンティティーは、セキュリティー設定の < security-identity> タグで指定します。

デフォルトでは、< security-identity> タグ が存在しない場合、EJB の独自の呼び出し元アイデンティティーが使用されます。

例9.3 EJB のセキュリティーアイデンティティーを呼び出し元と同じものに設定します。

この例では、EJB によって作成されたメソッド呼び出しのセキュリティーアイデンティティーを、現在の呼び出し元のアイデンティティーと同じに設定します。< security-identity > 要素宣言を指定しない場合は、この動作がデフォルトになります。

```
<ejb-jar>
```

```
  <enterprise-beans>
  <session>
  <ejb-name>ASessionBean</ejb-name>
  <!-- ... -->
  <security-identity>
    <use-caller-identity/>
  </security-identity>
  </session>
  <!-- ... -->
  </enterprise-beans>
</ejb-jar>
```

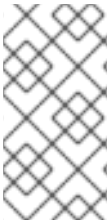
例9.4 EJB のセキュリティーアイデンティティーを特定のロールに設定します。

セキュリティーアイデンティティーを特定のロールに設定するには、< security -identity> タグ内の <run -as > および <role- name > タグを使用します。

```
<ejb-jar>
  <enterprise-beans>
  <session>
  <ejb-name>RunAsBean</ejb-name>
  <!-- ... -->
  <security-identity>
    <run-as>
    <description>A private internal role</description>
    <role-name>InternalRole</role-name>
    </run-as>
  </security-identity>
  </session>
  </enterprise-beans>
  <!-- ... -->
</ejb-jar>
```

デフォルトでは、< run-as > を使用すると、anonymous という名前のプリンシパルが発信呼び出しに割り当てられます。別のプリンシパルを割り当てるには、< run-as-principal> を使用し ます。

```
<session>
    <ejb-name>RunAsBean</ejb-name>
    <security-identity>
        <run-as-principal>internal</run-as-principal>
    </security-identity>
</session>
```



サーブレットでのセキュリティーアイデンティティーの指定

servlet 要素内で < run-as> および &lt ;run-as-principal> 要素を使用することもできます。

以下も参照してください。

- 「EJB セキュリティーアイデンティティー」
- 「EJB セキュリティーパラメーターのリファレンス」

バグの報告

9.2.2. EJB メソッドパーミッション

9.2.2.1. EJB メソッドパーミッションについて

EJB はメソッドへのアクセスを特定のセキュリティーロールに制限できます。

EJB &lt ;method-permission> 要素の宣言は、EJB のインターフェースメソッドを呼び出すことができるロールを指定します。以下の組み合わせのパーミッションを指定できます。

- 名前付き EJB のすべてのホームおよびコンポーネントインターフェースメソッド
- 名前付き EJB のホームまたはコンポーネントインターフェースの指定メソッド
- オーバーロード名を持つ一連のメソッドに指定されたメソッド

バグの報告

9.2.2.2. EJB メソッドパーミッションの使用

概要

&lt ;method-permission> 要素は、<method> 要素で定義された EJB メソッドへのアクセスが許可される論理ロールを定義 します。いくつかの例は、XML の構文を示しています。複数のメソッドパーミッションステートメントが存在する可能性があり、それらのステートメントには累積的影響があります。&lt ;method-permission> 要素は、< ejb -jar> 記述子の <assembly - descriptor> 要素の子です。

XML 構文は、EJB メソッドパーミッションのアノテーションを使用する代わりになります。

例9.5 ロールによる EJB のすべてのメソッドへのアクセスの許可

```
<method-permission>
  <description>The employee and temp-employee roles may access any method
  of the EmployeeService bean </description>
  <role-name>employee</role-name>
  <role-name>temp-employee</role-name>
  <method>
    <ejb-name>EmployeeService</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>
```

例9.6 ロールによる EJB の特定のメソッドのみへのアクセスを許可し、どのメソッドパラメーターを渡すかを制限します。

```
<method-permission>
  <description>The employee role may access the findByPrimaryKey,
  getEmployeeInfo, and the updateEmployeeInfo(String) method of
  the AcmePayroll bean </description>
  <role-name>employee</role-name>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>findByPrimaryKey</method-name>
  </method>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>getEmployeeInfo</method-name>
  </method>
  <method>
 <ejb-name>AcmePayroll</ejb-name>
 <method-name>updateEmployeeInfo</method-name>
 <method-params>
   <method-param>java.lang.String</method-param>
 </method-params>
  </method>
</method-permission>
```

例9.7 認証されたユーザーが EJB のメソッドにアクセスできるようにします。

< unchecked/ > 要素を使用すると、認証されたユーザーが指定されたメソッドを使用できます。

```
<method-permission>
 <description>Any authenticated user may access any method of the
 EmployeeServiceHelp bean</description>
 <unchecked/>
 <method>
<ejb-name>EmployeeServiceHelp</ejb-name>
<method-name>*</method-name>
 </method>
</method-permission>
```

例9.8 特定の EJB メソッドが使用されないように完全に除外

```
<exclude-list>
 <description>No fireTheCTO methods of the EmployeeFiring bean may be
 used in this deployment</description>
 <method>
<ejb-name>EmployeeFiring</ejb-name>
<method-name>fireTheCTO</method-name>
 </method>
</exclude-list>
```

例9.9 複数の &lt ; method-permission> ブロックを含む完全な <assembly-descriptor & gt;

```
<ejb-jar>
   <assembly-descriptor>
      <method-permission>
         <description>The employee and temp-employee roles may access any
            method of the EmployeeService bean </description>
         <role-name>employee</role-name>
         <role-name>temp-employee</role-name>
         <method>
            <ejb-name>EmployeeService</ejb-name>
            <method-name>*</method-name>
         </method>
      </method-permission>
      <method-permission>
         <description>The employee role may access the findByPrimaryKey,
            getEmployeeInfo, and the updateEmployeeInfo(String) method of
            the AcmePayroll bean </description>
         <role-name>employee</role-name>
         <method>
            <ejb-name>AcmePayroll</ejb-name>
            <method-name>findByPrimaryKey</method-name>
```

```
            </method>
            <method>
               <ejb-name>AcmePayroll</ejb-name>
               <method-name>getEmployeeInfo</method-name>
            </method>
            <method>
               <ejb-name>AcmePayroll</ejb-name>
               <method-name>updateEmployeeInfo</method-name>
               <method-params>
                  <method-param>java.lang.String</method-param>
               </method-params>
            </method>
         </method-permission>
         <method-permission>
            <description>The admin role may access any method of the
               EmployeeServiceAdmin bean </description>
            <role-name>admin</role-name>
            <method>
               <ejb-name>EmployeeServiceAdmin</ejb-name>
               <method-name>*</method-name>
            </method>
         </method-permission>
         <method-permission>
            <description>Any authenticated user may access any method of the
               EmployeeServiceHelp bean</description>
            <unchecked/>
            <method>
               <ejb-name>EmployeeServiceHelp</ejb-name>
               <method-name>*</method-name>
            </method>
         </method-permission>
         <exclude-list>
            <description>No fireTheCTO methods of the EmployeeFiring bean may be
               used in this deployment</description>
            <method>
               <ejb-name>EmployeeFiring</ejb-name>
               <method-name>fireTheCTO</method-name>
            </method>
         </exclude-list>
      </assembly-descriptor>
   </ejb-jar>
```

バグの報告

9.2.3. EJB セキュリティーアノテーション

9.2.3.1. EJB セキュリティーアノテーションについて

EJB javax.annotation.security アノテーションは JSR250 で定義されています。

EJB はセキュリティーアノテーションを使用してセキュリティーに関する情報をデプロイヤーに渡します。これらの型には次のようなものがあります。

@DeclareRoles

利用可能なロールを宣言します。

@RunAs

コンポーネントの伝播セキュリティーアイデンティティーを設定します。

バグの報告

9.2.3.2. EJB セキュリティーアノテーションの使用

概要

XML 記述子またはアノテーションのいずれかを使用して、Enterprise JavaBeans(EJB)でメソッドを呼び出すことのできるセキュリティーロールを制御できます。XML 記述子の使用方法は、「EJB メソッドパーミッションの使用」 を参照してください。

デプロイメント記述子に明示的に指定されるメソッドの値はすべて、アノテーションの値を上書きします。メソッドの値がデプロイメント記述子で指定されていない場合は、アノテーションを使用して設定された値が使用されます。上書きの粒度は、方法ごとに決定されます。

EJB のセキュリティーパーミッションを制御するアノテーション

@DeclareRoles

@DeclareRoles を使用して、パーミッションをチェックするセキュリティーロールを定義します。@DeclareRoles がない場合、この一覧は @RolesAllowed アノテーションから自動的にビルドされます。ロールの設定に関する情報は、「『Java EE 6 チュートリアル』 で セキュリティーロールの宣言による承認ユーザーの指定」を参照してください。

@RolesAllowed, @PermitAll, @DenyAll

@RolesAllowed を使用して、メソッドまたはメソッドへのアクセスを許可するロールを一覧表示します。メソッドまたはメソッドの使用をすべて許可または拒否するには、@PermitAll または @DenyAll を使用します。アノテーションメソッドパーミッション の設定に関する詳細は、「『Java EE 6 チュートリアル』 による承認済みのユーザーの指定」を参照してください。

@RunAs

@RunAs を使用して、アノテーション付きメソッドから呼び出しを行う際にメソッドが使用するロールを指定します。アノテーションを使用した伝播されたセキュリティーアイデンティティー の設定に関する詳細は、「『Java EE 6 チュートリアル』 によるセキュリティーアイデンティティー(Run-As) 」のチュートリアルを参照してください。

例9.10 セキュリティーアノテーションの例

```
@Stateless
@RolesAllowed({"admin"})
@SecurityDomain("other")
public class WelcomeEJB implements Welcome {
 @PermitAll
 public String WelcomeEveryone(String msg) {
  return "Welcome to " + msg;
 }
 @RunAs("tempemployee")
 public String GoodBye(String msg) {
    return "Goodbye, " + msg;
 }
 public String GoodbyeAdmin(String msg) {
  return "See you later, " + msg;
 }
}
```

このコードでは、すべてのロールがメソッド WelcomeEveryone にアクセスできます。GoodBye メソッドは、呼び出しの実行時に tempemployee ロールを使用します。admin ロールのみがメソッド GoodbyeAdmin にアクセスでき、セキュリティーアノテーションのないその他のメソッドにアクセスできます。

バグの報告

9.2.4. EJB へのリモートアクセス

9.2.4.1. リモートメソッドアクセス

JBoss Remoting は、EJB、JMX MBean、およびその他の同様のサービスへのリモートアクセスを提供するフレームワークです。SSL の使用に関わらず、以下のトランスポートタイプ内で動作します。

サポートされるトランスポートタイプ

- ソケット/セキュアソケット
- SSL 上の RMI / RMI
- HTTP / HTTPS
- Servlet / Secure Servlet
- bisocket / Secure Bisocket

警告

Red Hat は、影響を受けるすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSL を明示的に無効にすることを推奨します。

JBoss Remoting は、マルチキャストまたは JNDI を介して自動検出も提供します。

JBoss EAP 6 内のサブシステムの多くで使用され、クライアントが複数のトランスポートメカニズムでリモートで呼び出すことができるサービスを設計、実装、およびデプロイすることもできます。また、JBoss EAP 6 の既存のサービスにアクセスできます。

データマーシャリング

リモーティングシステムはデータマーシャリングおよびアンマーシャリングサービスも提供します。データマーシャリングとは、別のシステムが動作できるように、ネットワークおよびプラットフォームの境界間でデータを安全に移動できることを指します。その後、作業は元のシステムに戻され、ローカルで処理されているかのように動作します。

アーキテクチャーの概要

Remoting を使用するクライアントアプリケーションを設計する場合は、URL タイプ形式の単純な String である InvokerLocator と呼ばれる特殊なタイプのリソースロケーターを使用するようアプリケーションを設定し、アプリケーションに対してサーバーと通信するように指示します。サーバー

は、remoting サブシステムの一部として設定された コネクター のリモートリソースのリクエストをリッスンします。コネクター は設定済みの ServerInvocationHandler へのリクエストを処理します。各 ServerInvocationHandler はメソッド invoke(InvocationRequest) を実装し、リクエストの処理方法を認識します。

JBoss Remoting フレームワークには、クライアントとサーバー側で相互にミラーリングする 3 つの層が含まれます。

JBoss Remoting Framework レイヤー

- ユーザーは outer レイヤーと対話します。クライアント側では、外部レイヤーは、呼び出しリクエストを送信する Client クラスです。サーバー側では、ユーザーが実装し、呼び出しリクエストを受信する InvocationHandler です。
- トランスポートは invoker レイヤーによって制御されます。
- 下層のレイヤーには、データフォーマットをワイヤ形式に変換するマーシャラーとアンマーシャラーが含まれています。

バグの報告

9.2.4.2. リモーティングコールバック

Remoting クライアントがサーバーから情報を要求すると、サーバーが応答するのをブロックして待機できますが、通常これは理想的な動作ではありません。クライアントがサーバー上の非同期イベントをリッスンし、サーバーが要求を終了するのを待機している間に他の作業を継続できるようにするため、アプリケーションは終了した時点でサーバーに対し通知を送信するように要求します。これはコールバックと呼ばれます。あるクライアントも、別のクライアントの代わりに生成された非同期イベント用のリスナーとして追加できます。コールバックを受け取る方法は、プルコールバックまたはプッシュコールバックの2つの選択肢があります。クライアントはプルコールバックを同期的にチェックしますが、パッシブにプッシュコールバックをリッスンします。

基本的に、コールバックは InvocationRequest をクライアントに送信するサーバーによって動作します。サーバー側のコードは、コールバックが同期されているか非同期であるかに関わらず、同じように動作します。クライアントのみが違いを認識する必要があります。サーバーの InvocationRequest は responseObject をクライアントに送信します。これは、クライアントがリクエストしたペイロードです。これは、リクエストまたはイベント通知に直接応答する場合があります。

サーバーは、m_listeners オブジェクトを使用してリスナーも追跡します。これには、サーバーハンドラーに追加されたリスナーの一覧が含まれます。ServerInvocationHandler インターフェースに

は、この一覧を管理できるメソッドが含まれます。

クライアントは各種の方法でプルおよびプッシュコールバックを処理します。いずれの場合も、コールバックハンドラーを実装する必要があります。コールバックハンドラーは、コールバックデータを処理するインターフェース org.jboss.remoting.InvokerCallbackHandler の実装です。コールバックハンドラーを実装したら、コールバックのリスナーとして自身を追加するか、プッシュコールバックのコールバックサーバーを実装します。

プルコールバック

プル要求の場合、クライアントは Client.addListener() メソッドを使用してリスナーのリストに自身を追加します。その後、サーバーを定期的にポーリングしてコールバックデータの同期配信を行います。このポーリングは Client.getCallbacks() を使用して実行されます。

コールバックのプッシュ

プッシュコールバックでは、クライアントアプリケーションが独自の InvocationHandler を実行する必要があります。そのためには、クライアント自体で Remoting サービスを実行する必要があります。これは *コールバックサーバー* と呼ばれます。コールバックサーバーは、受信要求を非同期的に受け入れ、要求元（この場合はサーバー）に対して処理します。クライアントのコールバックサーバーをメインサーバーに登録するには、コールバックサーバーの InvokerLocator を 2 つ目の引数として addListener メソッドに渡します。

バグの報告

9.2.4.3. リモーティングサーバー検出

リモーティングサーバーとクライアントは、JNDI またはマルチキャストを使用して相互を自動的に検出できます。Remoting Detector がクライアントとサーバーの両方に追加され、NetworkRegistry がクライアントに追加されます。

サーバーサイドの Detector は定期的に InvokerRegistry をスキャンし、作成したすべてのサーバー呼び出しャーをプルします。この情報を使用して、各サーバー呼び出し元によってサポートされるロケーターおよびサブシステムが含まれる検出メッセージを公開します。このメッセージは、マルチキャストブロードキャストまたは JNDI サーバーへのバインディングを介して公開します。

クライアント側で、Detector はマルチキャストメッセージを受信するか、または定期的に JNDI サーバーをポーリングして検出メッセージを取得します。Detector が検出メッセージが新たに検出されたリモーティングサーバーであることを示す場合、これを NetworkRegistry に登録します。また、Detector は、サーバーが利用できなくなったことを検知した場合に NetworkRegistry も更新します。

バグの報告

9.2.4.4. Remoting サブシステムの設定

概要

JBoss Remoting には、ワーカースレッドプール、1 つ以上のコネクター、および一連のローカルおよびリモート接続 URI の 3 つのトップレベル設定可能な要素があります。このトピックでは、設定可能な各項目の説明、各項目の設定方法についての CLI コマンドの例、および完全に設定されたサブシステムの XML の例を紹介します。この設定はサーバーにのみ適用されます。独自のアプリケーションにカスタムコネクターを使用しない限り、ほとんどのユーザーは Remoting サブシステムをまったく設定する必要はありません。EJB などの Remoting クライアントとして動作するアプリケーションには、特定のコネクターに接続するための個別の設定が必要です。

注記

Remoting サブシステム設定は Web ベースの管理コンソールに公開されませんが、コマンドラインベースの管理 CLI から完全に設定可能です。手作業による XML の編集は推奨されません。

CLI コマンドの適合

CLI コマンドは、デフォルト のプロファイルの設定時に管理対象ドメインに対して式化されます。別のプロファイルを設定するには、その名前を置き換えます。スタンドアロンサーバーの場合は、コマンドの /profile=default 部分を省略します。

Remoting サブシステム外の設定

remoting サブシステム以外の設定側面がいくつかあります。

ネットワークインターフェース

remoting サブシステムによって使用されるネットワークインターフェースは、domain/configuration/domain.xml または standalone/configuration/standalone.xml で定義された パブリック インターフェースです。

```
<interfaces>
  <interface name="management"/>
  <interface name="public"/>
  <interface name="unsecure"/>
</interfaces>
```

パブリック インターフェースのホストごとの定義は、domain.xml または standalone.xml と同じディレクトリーの host.xml で定義されます。このインターフェースは、他のサブシステムによっても使用されます。変更時には注意が必要です。

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
  </interface>
  <interface name="unsecure">
    <!-- Used for IIOP sockets in the standard configuration.
         To secure JacORB you need to setup SSL -->
    <inet-address value="${jboss.bind.address.unsecure:127.0.0.1}"/>
  </interface>
</interfaces>
```

socket-binding

remoting サブシステムによって使用されるデフォルトの socket-binding は TCP ポート 4447 にバインドされます。これを変更する必要がある場合は、ソケットバインディングおよびソケットバインディンググループのドキュメントを参照してください。

EJB のリモーティングコネクターリファレンス

EJB サブシステムには、リモートメソッド呼び出しに対するリモーティングコネクターへの参照が含まれます。デフォルト設定は次のとおりです。

```
<remote connector-ref="remoting-connector" thread-pool-name="default"/>
```

セキュアなトランスポート設定

リモーティングトランスポートは、クライアントが要求した場合に StartTLS を使用してセキュアな（HTTPS、Secure Servlet など）接続を使用します。セキュアな接続とセキュアでない接続に同じソケットバインディング（ネットワークポート）が使用されるため、サーバー側の追加設定は必要ありません。クライアントは必要に応じてセキュアなトランスポートまたはセキュアでないトランスポートを要求します。EJB、ORB、JMS プロバイダーなどの Remoting を使用する JBoss EAP 6 コンポーネントは、デフォルトでセキュアなインターフェースを要求します。

警告： STARTTLS セキュリティーに関する考慮事項

StartTLS は、クライアントが要求した場合にセキュアな接続をアクティベートし、セキュアでない接続にデフォルト設定することで機能します。これは、攻撃者がクライアントの要求をインターセプト *してセキュアでない接続を要求するために Middle スタイルの不正使用における Man* に悪影響を与えることが本質的に行われます。セキュアでない接続が実際に適切なフォールバックである場合を除き、クライアントがセキュアな接続を受信しない場合、適切に失敗するよう記述する必要があります。

ワーカースレッドプール

ワーカースレッドプールは、Remoting コネクターを介して提供される作業の処理に使用できるスレッドのグループです。これは単一の要素 &lt ;worker-thread-pool> で、複数の属性を取ります。ネットワークタイムアウトの取得、スレッドの不足、またはメモリー使用量の制限が必要な場合には、これらの属性をチューニングします。特定の推奨事項は、特定の状況によって異なります。詳細は Red Hat グローバルサポートサービスまでお問い合わせください。

表9.1 ワーカースレッドプールの属性

属性	説明	CLI コマンド
read-threads	リモーティングワーカーに対して作成する読み取りスレッドの数。デフォルトは**1**です。	**/profile=default/subsystem=remoting/:write-attribute(name=worker-read-threads,value=1)**
write-threads	リモーティングワーカーに作成する書き込みスレッドの数。デフォルトは**1**です。	**/profile=default/subsystem=remoting/:write-attribute(name=worker-write-threads,value=1)**
task-keepalive	コアでないリモーティングワーカーのタスクスレッドにキープアライブを使用する期間 (ミリ秒単位)。デフォルトは **60** です。	**/profile=default/subsystem=remoting/:write-attribute(name=worker-task-keepalive,value=60)**
task-max-threads	リモーティングワーカーのタスクスレッドプールに対するスレッドの最大数。デフォルトは **16** です。	**/profile=default/subsystem=remoting/:write-attribute(name=worker-task-max-threads,value=16)**
task-core-threads	リモーティングワーカーのタスクスレッドプールに対するコアスレッドの数。デフォルトは **4** です。	**/profile=default/subsystem=remoting/:write-attribute(name=worker-task-core-threads,value=4)**

属性	説明	CLI コマンド
task-limit	許可するリモーティングワーカータスクの最大数。 この数を超えるリモーティングワーカータスクは拒否されます。デフォルトは **16384** です。	**/profile=default/subsystem=remoting/:write-attribute(name=worker-task-limit,value=16384)**

コネクター

コネクターは主な Remoting 設定要素です。複数のコネクターを設定できます。それぞれは複数のサブ要素を持つ & lt;connector > 要素といくつかの属性で構成されます。デフォルトのコネクターは、JBoss EAP 6 の複数のサブシステムによって使用されます。カスタムコネクターの要素や属性の設定はアプリケーションに依存するため、詳細は Red Hat グローバルサポートサービスにお問い合わせください。

表9.2 コネクターの属性

属性	説明	CLI コマンド
socket-binding	このコネクターに使用するソケットバインディングの名前。	**/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=socket-binding,value=remoting)**
authentication-provider	このコネクターで使用する JASPIC(Java Authentication Service Provider Interface for Containers)モジュール。モジュールはクラスパスにある必要があります。	**/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=authentication-provider,value=myProvider)**
security-realm	オプション。アプリケーションのユーザー、パスワード、およびロールが含まれるセキュリティーレルム。EJB または Web アプリケーションはセキュリティーレルムに対して認証できます。**ApplicationRealm** はデフォルトの JBoss EAP 6 インストールで利用できます。	**/profile=default/subsystem=remoting/connector=remoting-connector/:write-attribute(name=security-realm,value=ApplicationRealm)**

表9.3 コネクター要素

属性	説明	CLI コマンド
sasl	SASL(Simple Authentication and Security Layer)認証メカニズムの組み込み要素	**該当なし**

属性	説明	CLI コマンド
properties	1 つ以上の &lt **;property&gt**; 要素が含まれ、それぞれ **name** 属性と任意の **value** 属性が含まれます。	**/profile=default/subsystem=remoting/connector=remoting-connector/property=myProp/:add(value=myPropValue)**

送信接続

3 種類のアウトバウンド接続を指定することができます。

- **URI への送信接続。**
- **ローカルアウトバウンド接続： ソケットなどのローカルリソースに接続します。**
- **リモートアウトバウンド接続： リモートリソースに接続し、セキュリティーレルムを使用して認証します。**

すべてのアウトバウンド接続は < outbound-connections> 要素に囲まれて います。これらの各接続タイプは outbound-socket-binding-ref 属性を取ります。outbound-connection は uri 属性を取ります。リモートアウトバウンド接続は、承認に使用する任意の username および security-realm 属性を取ります。

表9.4 送信接続要素

属性	説明	CLI コマンド
outbound-connection	汎用アウトバウンド接続。	**/profile=default/subsystem=remoting/outbound-connection=my-connection/:add(uri=http://my-connection)**
local-outbound-connection	暗黙的な local:// URI スキームを使用した送信接続。	**/profile=default/subsystem=remoting/local-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting2)**

属性	説明	CLI コマンド
remote-outbound-connection	セキュリティーレルムで basic/digest 認証を使用した remote:// URI スキームの送信接続。	**/profile=default/subsystem=remoting/remote-outbound-connection=my-connection/:add(outbound-socket-binding-ref=remoting,username=myUser,security-realm=ApplicationRealm)**

SASL 要素

SASL 子要素を定義する前に、最初の SASL 要素を作成する必要があります。以下のコマンドを使用します。

```
/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:add
```

SASL 要素の子要素は以下の表で説明されています。

表9.5 SASL 子要素

属性	説明	CLI コマンド
include-mechanisms	SASL メカニズムのリストである **value** 属性が含まれます。	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=include-mechanisms,value=["DIGEST","PLAIN","GSSAPI"])`
qop	優先順で SASL Quality of 保護値のリストである **value** 属性が含まれます。	`/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=qop,value=["auth"])`

属性	説明	CLI コマンド
strength	**value** 属性が含まれます。これは SASL 暗号強度の値のリストで優先順で表されます。	/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=strength,value=["medium"])
reuse-session	ブール値である **value** 属性が含まれます。true の場合、セッションを再利用しようとします。	/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=reuse-session,value=false)
server-auth	ブール値である **value** 属性が含まれます。true の場合、サーバーはクライアントに対して認証します。	/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl:write-attribute(name=server-auth,value=false)
policy	ゼロ以上の要素が含まれるローカリング要素。それぞれが単一の **値** を取ります。 ● forward-secrecy: 前方機密性の実装にメカニズムが必要であるかどうか（1 つのセッションに分割しても、今後のセッションに侵入するための情報が自動的に提供されない） ● 非アクティブ - 辞書攻撃を受けやすいメカニズムを許可するかどうか。**false** を指定すると許可され、**true** は拒否されます。 ● no-anonymous: 匿名ログインを受け入れるメカニズムを許可するかどうか。**false** を指定すると許可され、**true** は拒否されます。 ● no-dictionary: パッシブ	/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:add /profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=forward-secrecy,value=true) /profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-active,value=false)

属性	説明	CLI コマンド
	辞書攻撃を受けやすいメカニズムを許可するかどうか。**false** を指定すると許可され、**true** は拒否されます。 ● no-plain-text: 単純なプレーンテキスト攻撃を受けやすいメカニズムを許可するかどうか。**false** を指定すると許可され、**true** は拒否されます。 ● pass-credentials: クライアントのクレデンシャルを渡すメカニズムを許可するかどうか。	/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-anonymous,value=false) /profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-dictionary,value=true) /profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=no-plain-text,value=false) /profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/sasl-policy=policy:write-attribute(name=pass-credentials,value=true)
properties	1 つ以上の &lt **;property&gt**; 要素が含まれ、それぞれ **name** 属性と任意の **value** 属性が含まれます。	/profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/property=myprop:add(value=1) /profile=default/subsystem=remoting/connector=remoting-connector/security=sasl/property=myprop2:add(value=2)

例9.11 設定例

以下の例は、JBoss EAP 6 に同梱されるデフォルトの remoting サブシステムを示していま

す。

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <connector name="remoting-connector" socket-binding="remoting" security-
realm="ApplicationRealm"/>
</subsystem>
```

この例には多くのハイフン的な値が含まれており、前述の要素と属性をコンテキストに配置するように提示されています。

```
<subsystem xmlns="urn:jboss:domain:remoting:1.1">
    <worker-thread-pool read-threads="1" task-keepalive="60" task-max-threads="16" task-core-
thread="4" task-limit="16384" write-threads="1" />
    <connector name="remoting-connector" socket-binding="remoting" security-
realm="ApplicationRealm">
        <sasl>
            <include-mechanisms value="GSSAPI PLAIN DIGEST-MD5" />
            <qop value="auth" />
            <strength value="medium" />
            <reuse-session value="false" />
            <server-auth value="false" />
            <policy>
                <forward-secrecy value="true" />
                <no-active value="false" />
                <no-anonymous value="false" />
                <no-dictionary value="true" />
                <no-plain-text value="false" />
                <pass-credentials value="true" />
            </policy>
            <properties>
                <property name="myprop1" value="1" />
                <property name="myprop2" value="2" />
            </properties>
        </sasl>
        <authentication-provider name="myprovider" />
        <properties>
            <property name="myprop3" value="propValue" />
        </properties>
    </connector>
    <outbound-connections>
        <outbound-connection name="my-outbound-connection" uri="http://myhost:7777/"/>
        <remote-outbound-connection name="my-remote-connection" outbound-socket-binding-
ref="my-remote-socket" username="myUser" security-realm="ApplicationRealm"/>
        <local-outbound-connection name="myLocalConnection" outbound-socket-binding-ref="my-
outbound-socket"/>
    </outbound-connections>
</subsystem>
```

設定イントロスペクションが文書化されていない

- JNDI およびマルチキャストの自動検出

バグの報告

9.2.4.5. リモート EJB クライアントでのセキュリティーレルムの使用

EJB をリモートで呼び出すクライアントにセキュリティーを追加する方法として、セキュリティーレルムを使用することができます。セキュリティーレルムは、ユーザー名/パスワードのペアとユーザー名/ロールのペアの単純なデータベースです。この用語は Web コンテナーのコンテキストでも使用され、意味が若干異なります。

EJB に対してセキュリティーレルムに存在する特定のユーザー名/パスワードのペアを認証するには、以下の手順に従います。

- 新しいセキュリティーレルムをドメインコントローラーまたはスタンドアロンサーバーに追加します。
- アプリケーションのクラスパスにある jboss-ejb-client.properties ファイルに以下のパラメーターを追加します。この例では、ファイルの他のパラメーターでコネクションが default と呼ばれることを前提としています。

```
remote.connection.default.username=appuser
remote.connection.default.password=apppassword
```

- 新しいセキュリティーレルムを使用するドメインまたはスタンドアロンサーバーにカスタム Remoting コネクターを作成します。
- カスタムリモーティングコネクターでプロファイルを使用するよう設定されたサーバーグループに EJB を追加します。または管理対象ドメインを使用していない場合はスタンドアロンサーバーに EJB をデプロイします。

バグの報告

9.2.4.6. 新しいセキュリティーレルムの追加

1. **管理 CLI を実行します。**

   jboss-cli.sh または jboss-cli.bat コマンドを起動し、サーバーに接続します。

2. **新しいセキュリティーレルム自体を作成します。**

   以下のコマンドを実行し、ドメインコントローラーまたはスタンドアロンサーバーに MyDomainRealm という名前の新しいセキュリティーレルムを作成します。

   ドメインインスタンスの場合は、以下のコマンドを使用します。

   ```
   /host=master/core-service=management/security-realm=MyDomainRealm:add()
   ```

   スタンドアロンインスタンスの場合には、以下のコマンドを使用します。

   ```
   /core-service=management/security-realm=MyDomainRealm:add()
   ```

3. **新規ロールについての情報を格納するプロパティーファイルへの参照を作成します。**

   以下のコマンドを実行して、新しいロールに関連するプロパティーが含まれる myfile.properties という名前のファイルを作成します。

   注記

   新規作成されたプロパティーファイルは、含まれる add-user.sh スクリプトおよび add-user.bat スクリプトで管理されません。これは外部で管理される必要があります。

   ドメインインスタンスの場合は、以下のコマンドを使用します。

   ```
   /host=master/core-service=management/security-realm=MyDomainRealm/authentication=properties:add(path=myfile.properties)
   ```

   スタンドアロンインスタンスの場合には、以下のコマンドを使用します。

   ```
   /core-service=management/security-realm=MyDomainRealm/authentication=properties:add(path=myfile.properties)
   ```

結果

新しいセキュリティーレルムが作成されます。この新しいレルムにユーザーおよびロールを追加すると、情報はデフォルトのセキュリティーレルムとは別のファイルに保存されます。この新しいファイルは、独自のアプリケーションまたは手順を使用して管理できます。

バグの報告

9.2.4.7. セキュリティーレルムへのユーザーの追加

1. add-user.sh または add-user.bat コマンドを実行します。

   ターミナルを開き、*EAP_HOME*/bin/ ディレクトリーに移動します。Red Hat Enterprise Linux または別の UNIX のようなオペレーティングシステムを実行している場合は、add-user.sh を実行します。Microsoft Windows Server を実行する場合は add-user.bat を実行します。

2. Management User または Application User を追加するかどうかを選択します。

   この手順では、b と入力してアプリケーションユーザーを追加します。

3. ユーザーを追加するレルムを選択します。

   デフォルトでは、利用可能なレルムは ApplicationRealm のみです。カスタムレルムを追加した場合は、代わりに名前を入力できます。

4. プロンプトが表示されたら、ユーザー名、パスワード、およびロールを入力します。

   プロンプトが表示されたら、希望のユーザー名、パスワード、および任意のロールを入力します。yes を入力して選択を確認するか、no と入力して変更をキャンセルします。変更は、セキュリティーレルムの各プロパティーファイルに書き込まれます。

バグの報告

9.2.4.8. SSL 暗号化を使用したリモート EJB アクセス

デフォルトでは、EJB2 および EJB3 Bean の Remote Method Invocation (RMI) のネットワークトラフィックは暗号化されません。暗号化が必要なインスタンスでは、クライアントとサーバー間の接続を暗号化する Secure Sockets Layer (SSL) を使用できます。また、SSL を使用すると、ファイアウォールの設定に応じて、ネットワークトラフィックが一部のファイアウォールを通過できるようになります。

警告

Red Hat は、影響を受けるすべてのパッケージで TLSv1.1 または TLSv1.2 を利用するために SSL を明示的に無効にすることを推奨します。

バグの報告

9.3. JAX-RS アプリケーションセキュリティー

9.3.1. RESTEasy JAX-RS Web サービスのロールベースのセキュリティーの有効化

概要

RESTEasy は JAX-RS メソッドで @RolesAllowed、@PermitAll、および @DenyAll アノテーションをサポートします。ただし、デフォルトではこれらのアノテーションを認識しません。以下の手順に従って web.xml ファイルを設定し、ロールベースのセキュリティーを有効にします。

警告

アプリケーションが EJB を使用する場合は、ロールベースのセキュリティーをアクティベートしないでください。EJB コンテナーは RESTEasy ではなく機能を提供します。

手順9.1 RESTEasy JAX-RS Web サービスのロールベースのセキュリティーの有効化

1. テキストエディターでアプリケーションの web.xml ファイルを開きます。

2. 以下の <context-param> を web-app タグ内のファイルに追加します。

```
<context-param>
  <param-name>resteasy.role.based.security</param-name>
  <param-value>true</param-value>
```

```
</context-param>
```

3. **<security-role> タグを使用して、RESTEasy JAX-RS WAR ファイル内で使用されているすべてのロールを宣言します。**

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

4. **すべてのロールに対して JAX-RS ランタイムが処理するすべての URL へのアクセスを承認します。**

```
<security-constraint>
   <web-resource-collection>
 <web-resource-name>Resteasy</web-resource-name>
 <url-pattern>/PATH</url-pattern>
   </web-resource-collection>
   <auth-constraint>
 <role-name>ROLE_NAME</role-name>
 <role-name>ROLE_NAME</role-name>
   </auth-constraint>
</security-constraint>
```

結果

ロールベースのセキュリティーは、定義されたロールとともにアプリケーション内で有効にされています。

例9.12 ロールベースのセキュリティーの設定例

```
<web-app>

   <context-param>
 <param-name>resteasy.role.based.security</param-name>
 <param-value>true</param-value>
   </context-param>

   <servlet-mapping>
 <servlet-name>Resteasy</servlet-name>
 <url-pattern>/*</url-pattern>
   </servlet-mapping>

   <security-constraint>
 <web-resource-collection>
    <web-resource-name>Resteasy</web-resource-name>
```

```
    <url-pattern>/security</url-pattern>
</web-resource-collection>
<auth-constraint>
    <role-name>admin</role-name>
    <role-name>user</role-name>
</auth-constraint>
  </security-constraint>

  <security-role>
<role-name>admin</role-name>
  </security-role>
  <security-role>
<role-name>user</role-name>
  </security-role>

</web-app>
```

バグの報告

9.3.2. アノテーションを使用した JAX-RS Web サービスのセキュア化

概要

このトピックでは、サポートされるセキュリティーアノテーションを使用して JAX-RS Web サービスのセキュリティーを保護する手順を説明します。

手順9.2 サポートされるセキュリティーアノテーションを使用した JAX-RS Web サービスのセキュア化

1. ロールベースのセキュリティーを有効にします。詳細は、以下を参照してください。「RESTEasy JAX-RS Web サービスのロールベースのセキュリティーの有効化」

2. JAX-RS Web サービスにセキュリティーアノテーションを追加します。RESTEasy は以下のアノテーションをサポートします。

   @RolesAllowed

   メソッドにアクセスできるロールを定義します。すべてのロールは web.xml ファイルに定義する必要があります。

   @PermitAll

   web.xml ファイルで定義されたすべてのロールがメソッドにアクセスできるようにします。

@DenyAll

メソッドへのすべてのアクセスを拒否します。

バグの報告

第10章 SECURITY サブシステム

10.1. セキュリティーサブシステム

security サブシステムはアプリケーションのセキュリティーインフラストラクチャーを提供します。このサブシステムは現在のリクエストに関連するセキュリティーコンテキストを使用して、認証マネージャー、承認マネージャー、監査マネージャー、およびマッピングマネージャーの性能を関係のあるコンテナーに公開します。

security サブシステムはデフォルトで事前設定されているため、セキュリティー要素を変更する必要はほとんどありません。変更が必要となる可能性がある唯一のセキュリティー要素は、*deep-copy-subject-mode* を使用するかどうかです。 ほとんどの場合、管理者は *セキュリティードメインの設定に重点を置いています。*

ディープコピーモード

ディープコピーサブジェクトモードの詳細は、「ディープコピーサブジェクトモード」 を参照してください。

セキュリティードメイン

セキュリティードメインは、認証、承認、監査、およびマッピングを制御するために 1 つ以上のアプリケーションが使用する *Java Authentication and Authorization Service(JAAS)* の宣言的セキュリティー設定のセットです。デフォルトでは jboss-ejb-policy、jboss-web-policy、および その他 のセキュリティードメインが含まれます。アプリケーション要件に応じて必要な数のセキュリティードメインを作成できます。

バグの報告

10.2. セキュリティーサブシステムの構造

security サブシステムは管理対象ドメインまたはスタンドアロン設定ファイルで設定されます。設定要素のほとんどは、Web ベースの管理コンソールまたはコンソールベースの管理 CLI を使用して設定できます。以下は、security サブシステムの例を表す XML です。

例10.1 セキュリティーサブシステムの設定例

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
 <security-management>
  ...
 </security-management>
 <security-domains>
```

```
        <security-domain name="other" cache-type="default">
            <authentication>
                <login-module code="Remoting" flag="optional">
                    <module-option name="password-stacking" value="useFirstPass"/>
                </login-module>
                <login-module code="RealmUsersRoles" flag="required">
                    <module-option name="usersProperties"
value="${jboss.domain.config.dir}/application-users.properties"/>
                    <module-option name="rolesProperties"
value="${jboss.domain.config.dir}/application-roles.properties"/>
                    <module-option name="realm" value="ApplicationRealm"/>
                    <module-option name="password-stacking" value="useFirstPass"/>
                </login-module>
            </authentication>
        </security-domain>
        <security-domain name="jboss-web-policy" cache-type="default">
            <authorization>
                <policy-module code="Delegating" flag="required"/>
            </authorization>
        </security-domain>
        <security-domain name="jboss-ejb-policy" cache-type="default">
            <authorization>
                <policy-module code="Delegating" flag="required"/>
            </authorization>
        </security-domain>
    </security-domains>
    <vault>
     ...
    </vault>
</subsystem>
```

&lt ;security-management >、< subject-factory >、< security-properties >、および < vault > 要素はデフォルト設定には存在しません。JBoss EAP 6.1 以降では、<subject -factory > および <security-properties > 要素が非推奨になりました。

バグの報告

10.3. SECURITY サブシステムの設定

10.3.1. セキュリティーサブシステムの設定

セキュリティーサブシステムの設定には、管理 CLI または Web ベースの管理コンソールを使用できます。

security サブシステム内の各トップレベル要素には、セキュリティー設定のさまざまな側面に関する情報が含まれます。セキュリティーサブシステム設定の例は、「セキュリティーサブシステムの構造」 を参照してください。

<security-management>

このセクションでは、security サブシステムのハイレベルな動作を上書きします。これには、追加のスレッド安全性用に、セキュリティートークンにコピーするか、またはリンクするかどうかを指定する任意の設定 deep-copy-subject-mode が含まれます。

<security-domains>

複数のセキュリティードメインを保持するコンテナー要素。セキュリティードメインには、認証、承認、マッピング、および監査モジュールに関する情報と、JASPI 認証および JSSE 設定が含まれる場合があります。アプリケーションはセキュリティードメインを指定してセキュリティー情報を管理します。

<security-properties>

java.security.Security クラスに設定されるプロパティーの名前と値が含まれます。

バグの報告

10.3.2. セキュリティー管理

10.3.2.1. ディープコピーサブジェクトモード

ディープコピーサブジェクトモード が無効の場合（デフォルト）、セキュリティーデータ構造をコピーすると、データ構造全体をコピーするのではなく、元のデータ構造への参照が作成されます。この動作は効率的ですが、同じアイデンティティーを持つ複数のスレッドがフラッシュまたはログアウト操作によってサブジェクトをクリアすると、データ破損が発生する可能性が高くなります。

ディープコピーサブジェクトモードでは、クローン可能とマークされている限り、データ構造の完全コピーと関連するデータが作成されます。これはよりスレッドセーフになりますが、効率が悪くなります。

ディープコピーサブジェクトモードは、セキュリティーサブシステムの一部として設定されます。

バグの報告

10.3.2.2. ディープコピーサブジェクトモードの有効化

Web ベースの管理コンソールまたは管理 CLI より、ディープコピーセキュリティーモードを有効にできます。

手順10.1 管理コンソールからディープコピーセキュリティーモードを有効化

1. 管理コンソールへのログイン

   詳細な手順は、カスタマーポータルの JBoss Enterprise Application Platform 6.x の『『Administration and Configuration Guide』』の「titled 『The Management Console』」セクションを参照 https://access.redhat.com/site/documentation/JBoss_Enterprise_Application_Platform/ してください。

2. 管理対象ドメイン: 適切なプロファイルを選択します。

   管理対象ドメインではプロファイルごとにセキュリティーサブシステムが設定されているため、各プロファイルに対して個別にディープコピーセキュリティーモードを有効または無効にできます。

   プロファイルを選択するには、画面上部の Configuration をクリックし、左上の Profile ドロップダウンボックスからプロファイルを選択します。

3. Security Subsystem 設定メニューを開きます。

   Security メニューを展開し、Security Subsystem を選択します。

4. Deep Copy Subject モードを有効にします。

   Edit をクリックします。Deep Copy Subjects の横にあるボックスにチェックを入れ、ディープコピーサブジェクトモードを有効にします。

管理 CLI を使用したディープコピーサブジェクトモードの有効化

管理 CLI を使用してこのオプションを有効にしたい場合、以下のコマンドの 1 つを使用します。

例10.2 管理対象ドメイン

```
/profile=full/subsystem=security/:write-attribute(name=deep-copy-subject-mode,value=TRUE)
```

例10.3 スタンドアロンサーバー

```
/subsystem=security/:write-attribute(name=deep-copy-subject-mode,value=TRUE)
```

バグの報告

10.3.3. セキュリティードメイン

10.3.3.1. セキュリティードメイン

セキュリティードメインは JBoss EAP 6 の security サブシステムの一部です。すべてのセキュリティー設定は、管理対象ドメインのドメインコントローラーまたはスタンドアロンサーバーによって集中管理されるようになりました。

セキュリティードメインは、認証、承認、セキュリティーマッピング、および監査の設定で構成されます。*JAAS(Java Authentication and Authorization Service)* の宣言的セキュリティーを実装します。

認証とは、ユーザーのアイデンティティーを検証することを指します。セキュリティー用語では、このユーザーは *プリンシパル* と呼ばれます。認証および承認は異なりますが、含まれる認証モジュールの多くは承認も処理します。

承認 は、認証されたユーザーが、システムまたは操作の特定リソースにアクセスするためのパーミッションまたは権限を持っているかどうかをサーバーが判断するプロセスです。

セキュリティーマッピング とは、情報をアプリケーションに渡す前にプリンシパル、ロール、または属性から情報を追加、変更、または削除する機能のことです。

監査マネージャーは、セキュリティーイベントの報告方法を制御するための *プロバイダーモジュール* の設定を可能にします。

セキュリティードメインを使用する場合は、アプリケーション自体から特定のセキュリティー設定をすべて削除できます。これにより、セキュリティーパラメーターを一元的に変更できます。このような設定構造を有効に活用できる一般的な例が、アプリケーションをテスト環境と本番環境の間で移動する処理です。

バグの報告

10.3.3.2. セキュリティードメインに関連する CLI 操作

例10.4 flush-cache

この CLI コマンドは、セキュリティードメインの認証キャッシュに保存されているエントリーを削除します。username を値としてプリンシパル引数を使用すると、単一のエントリーをフラッシュできます。操作に引数が渡されない場合、すべてのエントリーがフラッシュされます。この操作の詳細は、CLI コマンドを使用します。

```
/subsystem=security/security-domain=other:read-operation-description(name=flush-cache)
```

例10.5 list-cached-principals

この CLI コマンドは、このセキュリティードメインの認証キャッシュに保存されているプリンシパルを一覧表示します。この操作の詳細は、CLI コマンドを使用します。

```
/subsystem=security/security-domain=other:read-operation-description(name=list-cached-principals)
```

バグの報告

第11章 認証および承認

11.1. KERBEROS および SPNEGO 統合

11.1.1. Kerberos および SPNEGO 統合

Kerberos は、オープンネットワークコンピューティング環境向けに設計された認証方法です。チケットとオーセンティケーターに基づいて、ユーザーとサーバー両方のアイデンティティーを確立します。これは、セキュアではない環境で通信する 2 つのノードが、安全な方法で相互のアイデンティティーを確立するのに役立ちます。

SPNEGO は、クライアントアプリケーションによって使用される認証方法で、クライアントアプリケーション自体をサーバーに対して認証します。この技術は、クライアントアプリケーションと相互の通信を試みるサーバーが、もう一方がサポートする認証プロトコルを認識できない場合に使用されます。SPNEGO は、クライアントアプリケーションとサーバー間の共通の GSSAPI メカニズムを判断し、その後のセキュリティー操作をすべてディスパッチします。

Kerberos および SPNEGO 統合

一般的なセットアップでは、ユーザーは Active Directory ドメインによって管理されるデスクトップにログインします。その後、ユーザーは Firebox または Internet Explorer の web ブラウザーを使用して、JBoss EAP でホストされる JBoss Negotiation を使用する web アプリケーションにアクセスします。Web ブラウザーはデスクトップサインオン情報を Web アプリケーションに転送します。JBoss EAP は、Active Directory または Kerberos サーバーでバックグラウンドの GSS メッセージを使用し、ユーザーを検証します。これにより、ユーザーは web アプリケーションへのシームレスな SSO を実現することができます。

バグの報告

11.1.2. SPNEGO を使用したデスクトップ SSO

SPNEGO を使用してデスクトップ SSO を設定するには、以下を設定します。

- セキュリティードメイン
- システムプロパティー
- Web アプリケーション

手順11.1 SPNEGO を使用したデスクトップ SSO の設定

1. **セキュリティードメインの設定**

   サーバーのアイデンティティーを表現し、Web アプリケーションをセキュアにするようにセキュリティードメインを設定します。

例11.1 セキュリティードメインの設定

```
<security-domains>
  <security-domain name="host" cache-type="default">
   <authentication>
    <login-module code="Kerberos" flag="required">
     <module-option name="storeKey" value="true"/>
     <module-option name="useKeyTab" value="true"/>
     <module-option name="principal" value="host/testserver@MY_REALM"/>
     <module-option name="keyTab" value="/home/username/service.keytab"/>
     <module-option name="doNotPrompt" value="true"/>
     <module-option name="debug" value="false"/>
    </login-module>
   </authentication>
  </security-domain>

  <security-domain name="SPNEGO" cache-type="default">
   <authentication>
    <login-module code="SPNEGO"  flag="requisite">
     <module-option name="password-stacking" value="useFirstPass"/>
     <module-option name="serverSecurityDomain" value="host"/>
    </login-module>

    <!-- Login Module For Roles Search -->
   </security-domain>
```

2. システムプロパティーの設定

必要な場合は、ドメインモデルでシステムプロパティーを設定できます。

例11.2 システムプロパティーの設定

```
<system-properties>
    <property name="java.security.krb5.kdc" value="mykdc.mydomain"/>
    <property name="java.security.krb5.realm" value="MY_REALM"/>
  </system-properties>
```

3. Web アプリケーションの設定

オーセンティケーターをオーバーライドすることはできませんが、web アプリケーションを設定するために NegotiationAuthenticator を jboss-web.xml 記述子にバルブとして追加することもできます。

注記

バルブには、セキュアなリソースを決定するために使用されるため、security-constraint および login-config が web.xml ファイルに定義する必要があります。ただし、選択した auth-method はこのオーセンティケーターによって上書きされます。

例11.3 Web アプリケーションの設定

```
<!DOCTYPE jboss-web PUBLIC
 "-//JBoss//DTD Web Application 2.4//EN"
 "http://www.jboss.org/j2ee/dtd/jboss-web_4_0.dtd">

 <jboss-web>

   <security-domain>SPNEGO</security-domain>

   <valve>

    <class-name>org.jboss.security.negotiation.NegotiationAuthenticator</class-name>

   </valve>

 </jboss-web>
```

web アプリケーションでは、JBoss Negotiation クラスを配置できるように META-INF/MANIFEST.MF に依存関係を定義する必要もあります。

例11.4 META-INF/MANIFEST.MFへの依存関係の定義

```
Manifest-Version: 1.0

Build-Jdk: 1.6.0_24

Dependencies: org.jboss.security.negotiation
```

バグの報告

11.1.3. Microsoft Windows ドメインでの JBoss Negotiation の設定

ここでは、JBoss EAP が Active Directory ドメインの一部である Microsoft Windows サーバーで稼働している場合に JBoss Negotiation が使用されるために必要なアカウントを設定する方法を説明します。

ここでは、サーバーへのアクセスに使用されるホスト名は {hostname} と呼ばれます。レルムは {realm} と呼ばれ、ドメインは {domain} と呼ばれ、JBoss EAP インスタンスをホストするサーバーは {machine_name} と呼ばれます。

手順11.2 Microsoft Windows ドメインでの JBoss Negotiation の設定

1. 既存のサービスプリンシパルマッピングの消去

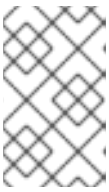
   Microsoft Windows ネットワークでは、一部のマッピングが自動作成されます。自動的に作成されたマッピングを削除し、ネゴシエーションが適切に行われるようにサーバーのアイデンティティーをサービスプリンシパルへマップします。マッピングにより、クライアントコンピューター上の Web ブラウザーがサーバーを信頼し、SPNEGO の実行を試みます。クライアントコンピューターは、HTTP{hostname} 形式のマッピングに対し、ドメインコントローラーを検証します。

   既存のマッピングを削除する手順は次のとおりです。

   - setspn -L {machine_name} コマンドを使用して、コンピューターのドメインに登録されているマッピングを一覧表示します。

- コマンド **setspn -D HTTP/{hostname} {machine_name}** および **setspn -D host/{hostname} {machine_name}** を使用して、既存のマッピングを削除します。

2. ホストユーザーアカウントを作成します。



注記

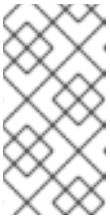
ホスト名が **{machine_name}** とは異なることを確認します。

これ以降では、ホストのユーザー名は **{user_name}** と呼ばれます。

3. **{user_name}** と **{hostname}** 間のマッピングを定義します。

- 以下のコマンドを実行して、サービスプリンシパルマッピング **ktpass -princ HTTP/{hostname}@{realm} -pass * -mapuser {domain}\{user_name}** を設定します。

- プロンプトが表示されたら、ユーザー名のパスワードを入力します。



注記

キータブのエクスポートの前提条件として、ユーザー名のパスワードをリセットします。

- 以下のコマンドを実行してマッピングを確認します。 **setspn -L {user_name}**

4. **EAP JBoss** がインストールされているサーバーに、ユーザーのキータブをエクスポートします。

以下のコマンドを実行してキータブ **ktab -k service.keytab -a HTTP/{hostname}@{realm}** をエクスポートします。

注記

このコマンドは、HTTP/{hostname} プリンシパルのチケットをキータブ service.keytab にエクスポートします。キータブは、JBoss でホストセキュリティードメインを設定するために使用されます。

5. セキュリティードメイン内で以下のようにプリンシパルを定義します。

```
<module-option name="principal">HTTP/{hostname}@{realm}</module-option>
```

バグの報告

11.1.4. PicketLink IDP の Kerberos 認証

以下のセクションでは、PicketLink IDP の Kerberos 認証を設定する方法を説明します。

手順11.3 JBoss EAP 6 のインストールおよび Kerberos の設定

1. JBoss EAP 6 をダウンロードし、インストールします。『インストールガイド』の「インストール手順」を参照してください。

2. Oracle Java または IBM Java を使用している場合でも、無制限の JCE を使用する必要があります。無制限の JCE がない場合、JBoss サーバーは適切な SPNEGO メカニズムタイプ（GSS_IAKERB_MECHANISMである 1.3.6.1.5.2.5 を使用して）でネゴシエートすることはできません。

3. 以下の例を使用して、必要な Java バージョンを使用するように JBoss を設定します。

```
export JAVA_HOME=JDK/JRE_directory
```

手順11.4 JBoss Negotiation Toolkit を使用した Kerberos 設定のテスト

1. Githubで利用可能な JBoss Negotiation Toolkit の使用

2. 設定ファイルを変更し、mvn clean install コマンドを使用してプロジェクトをビルドします。

3. jboss-negotiation-toolkit/target/jboss-negotiation-toolkit.war ファイルを $JBOSS_HOME/standalone/deployments/ にコピーします。

4. 3 つのセクションがすべて JBoss Negotiation Toolkit を通過していることを確認します。

手順11.5 PicketLink IDP の設定

idp.warへの変更

提供されている例では、idp.war アーカイブと employee.war アーカイブを使用し、PicketLink Quickstarts リポジトリーに配置します。以下のように idp.war のファイルを変更します。

1. IDP は JBoss Negotiation モジュールを使用するため、org.jboss.security.negotiation モジュールを $JBOSS_HOME/standalone/deployments/idp.war/META-INF/jboss-deployment-structure.xml に追加します。

```
<jboss-deployment-structure>
  <deployment>
  <!-- Add picketlink module dependency -->
    <dependencies>
      <module name="org.picketlink" />
      <module name="org.jboss.security.negotiation" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

2. SPNEGO の追加のバルブ org.jboss.security.negotiation.NegotiationAuthenticator を $JBOSS_HOME/standalone/deployments/idp.war/WEB-INF/jboss-web.xml に追加します。

3. 以下のように、$JBOSS_HOME/standalone/deployments/ idp.war/WEB-INF/jboss-web.xml の security-domain を idp から SPNEGO に変更します。

```
<jboss-web>
  <security-domain>SPNEGO</security-domain>
  <context-root>idp</context-root>
    <valve>
      <class-
name>org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOValve
</class-name>
      <param>
        <param-name>passUserPrincipalToAttributeManager</param-name>
        <param-value>true</param-value>
      </param>
    </valve>
    <valve>
      <class-name>org.jboss.security.negotiation.NegotiationAuthenticator</class-name>
    </valve>
</jboss-web>
```

4. **Kerberos サーバーセットアップにより、プリンシパルに追加された security-role を $JBOSS_HOME/standalone/deployments/idp.war/WEB-INF/web.xml に追加または変更します。**

5. **$JBOSS_HOME/standalone/deployments/idp.war/WEB-INF/picketlink.xml ファイルを以下のように変更します。**

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  <PicketLinkIDP xmlns="urn:picketlink:identity-federation:config:2.1">
  <IdentityURL>${idp.url::http://localhost:8080/idp/}</IdentityURL>
    <Trust>
     <Domains>redhat.com,localhost,amazonaws.com</Domains>
    </Trust>
  </PicketLinkIDP>
  <Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHandler" />
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler" />
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2AuthenticationHandler" />
      <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHandler" />
  </Handlers>
  <!-- The configuration bellow defines a token timeout and a clock skew. Both
configurations will be used during the SAML Assertion creation. This configuration is
optional. It is defined only to show you how to set the token timeout and clock skew
configuration. -->
  <PicketLinkSTS xmlns="urn:picketlink:identity-federation:config:1.0" TokenTimeout="5000"
ClockSkew="0">
    <TokenProviders>
    <TokenProvider
```

```
ProviderClass="org.picketlink.identity.federation.core.saml.v1.providers.SAML11AssertionTokenProvider"
      TokenType="urn:oasis:names:tc:SAML:1.0:assertion"
      TokenElement="Assertion" TokenElementNS="urn:oasis:names:tc:SAML:1.0:assertion"
/>
   <TokenProvider

ProviderClass="org.picketlink.identity.federation.core.saml.v2.providers.SAML20AssertionTokenProvider"
      TokenType="urn:oasis:names:tc:SAML:2.0:assertion"
      TokenElement="Assertion" TokenElementNS="urn:oasis:names:tc:SAML:2.0:assertion"
/>
   </TokenProviders>
  </PicketLinkSTS>
</PicketLink>
```

6. **IDP を実行しているサーバーのホスト名に一致するように IdentityURL を変更します。**

7. **Trust を、アイデンティティープロバイダーが信頼するドメイン名が含まれるように変更します。**

8. **employee.war を変更します。Kerberos サーバーセットアップにより、プリンシパルに追加された security-roles を $JBOSS_HOME/standalone/deployments/employee.war/WEB-INF/web.xml に追加または変更します。**

9. **$JBOSS_HOME/standalone/configuration/standalone.xml ファイルの セキュリティードメイン 設定を変更します。ロールマッピング設定は、通常のセキュリティードメイン設定と同じです。**

```
<security-domain name="host" cache-type="default">
  <authentication>
    <login-module code="Kerberos" flag="required">
      <module-option name="principal" value="HTTP/something.com@yourdomain.COM"/>
<module-option name="storeKey" value="true"/>
      <module-option name="useKeyTab" value="true"/>
      <module-option name="doNotPrompt" value="true"/>
      <module-option name="keyTab" value="/root/keytab"/>
    </login-module>
  </authentication>
</security-domain>
<security-domain name="SPNEGO" cache-type="default">
   <authentication>
```

```
    <login-module code="SPNEGO" flag="required">
      <module-option name="serverSecurityDomain" value="host"/>
    </login-module>
  </authentication>
</security-domain>
<security-domain name="sp" cache-type="default">
  <authentication>
    <login-module
    code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2LoginModule"
     flag="required" />
  </authentication>
</security-domain>
```

注記

IBM JDK を使用する場合、Kerberos モジュールのオプションは異なります。システムプロパティー jboss.security.disable.secdomain.option を true に設定する必要があります。詳細は、「セキュリティードメインでの認証の設定」 を参照してください。ログインモジュールを以下に更新します。

```
<login-module code="Kerberos" flag="required">
 <module-option name="principal" value="HTTP/something.com@yourdomain.COM"/>
 <module-option name="credsType" value="acceptor"/>
 <module-option name="useKeytab" value="file:///root/keytab"/>
</login-module>
```

手順11.6 PicketLink IDP の Kerberos 認証設定の確認

1. **$JBOSS_HOME/bin/standalone.sh を使用して JBoss EAP サーバーを起動します。**

2. **Kerberos システムを設定します。これにはいくつかの方法があります。たとえば、以下のオプションのいずれかを使用します。**

   - **freeipa: 複数の設定オプションがあります。設定に適したものを選択する必要があります。**
   - **ApacheDS**

3. Firefox などのブラウザーを Kerberos を使用するように設定します。「https://access.redhat.com/documentation/ja-JP/Red_Hat_Enterprise_Linux/5/html/Deployment_Guide/sso-config-firefox.html」に記載の手順に従ってください。

4. 上記のように設定した Firefox から http://YOUR_DOMAIN:8080/employee にアクセスできることを確認します。

バグの報告

11.1.5. PicketLink IDP で証明書でログイン

SSL をサポートする IDP の設定

PicketLink IDP が SSL をサポートするように設定できます。以下の手順は、SSL クライアント認証をサポートする IDP として Web アプリケーションを設定する方法を実証する例です。ユーザーを認証するように IDP を設定する方法は 2 つあります。

- SSL が使用されている場合、サーバーはクライアントに証明書を要求し、この証明書を使用してユーザーを認証するようにします。
- クライアントが証明書を提供していない場合は、フォームベースの認証が実行されます。

バグの報告

11.1.5.1. JBoss EAP 6 の SSL 設定

最初に証明書（設定手順全体で使用されるキーストアとトラストストア）を作成する必要があります。

手順11.7 サーバーの証明書、キーストア、およびトラストストアの作成

1. **サーバー用の証明書の作成**

   以下のコマンドを使用してサーバーの証明書を作成します。

```
keytool -genkey -alias server -keyalg RSA -keystore server.keystore -storepass
change_it -validity 365
```

システムは、追加情報を求めるプロンプトを出します。必要に応じて値を指定する必要があります。証明書の CN 名は DNS サーバー名と同じである必要があります。たとえば、ローカルホストの場合は、以下のコマンドを使用できます。

```
keytool -genkey -alias server -keystore server.keystore -storepass change_it -keypass
password -dname
"CN=localhost,OU=QE,O=example.com,L=Brno,C=CZ"
```

2. クライアント証明書の作成

このクライアント証明書を使用して、SSL 経由でリソースにアクセスするときにサーバーに対して認証します。

```
keytool -genkey -alias client -keystore client.keystore -storepass change_it -validity
365 -keyalg RSA -keysize 2048 -storetype pkcs12
```

3. トラストストアの作成

この証明書をインポートして、クライアントの証明書をエクスポートし、トラストストアを作成します。

```
keytool -exportcert -keystore client.keystore -storetype pkcs12 -storepass change_it -
alias client -keypass change_it -file client.keystore
keytool -import -file client.keystore -alias client -keystore client.truststore
```

4. JBoss EAP 6 Server インストールを SSL を有効にするように変更

以下のコネクターを Web サブシステムに追加して SSL を有効にします。

```
<connector name="https" protocol="HTTP/1.1" scheme="https" socket-
binding="https" enable-lookups="false" secure="true">
  <ssl name="localhost-ssl" key-alias="server" password="change_it"
  certificate-key-file="${jboss.server.config.dir}/server.keystore"
  protocol="TLSv1"
  verify-client="want"
  ca-certificate-file="${jboss.server.config.dir}/client.truststore"/>
</connector>
```

5. **サーバーの再起動**

   サーバーを再起動して、応答していることを確認します。 https://localhost:8443

6. **証明書の信頼**

   サーバー証明書を信頼するよう求められます。

ブラウザーでクライアント証明書の設定

アプリケーションにアクセスする前に、client.keystore をブラウザーにインポートする必要があります。このファイルはクライアント証明書を保持します。アプリケーションにアクセスすると、ブラウザーはサーバーとの認証に使用する証明書を選択するように求められます。必要な証明書を選択します。

セキュリティードメインの設定

以下のセキュリティードメインをサーバーインストールに追加します。スタンドアロンモードを使用している場合は、JBOSS_HOME/standalone/configuration/standalone.xml に追加する必要があります。

```
 <security-domain name="idp" cache-type="default">
   <authentication>
   <login-module code="CertificateRoles" flag="optional">
   <module-option name="password-stacking" value="useFirstPass"/>
   <module-option name="securityDomain" value="idp"/>
   <module-option name="verifier" value="org.jboss.security.auth.certs.AnyCertVerifier"/>
   </login-module>

   <login-module
code="org.picketlink.identity.federation.bindings.jboss.auth.RegExUserNameLoginModule"
flag="optional">
   <module-option name="regex" value="CN=(.*?),"/>
   </login-module>

   <login-module code="UsersRoles" flag="required">
   <module-option name="password-stacking" value="useFirstPass"/>
   <module-option name="usersProperties" value="users.properties"/>
   <module-option name="rolesProperties" value="roles.properties"/>
   </login-module>
   </authentication>

   <jsse keystore-password="change_it" keystore-url="${jboss.server.config.dir}" truststore-
password="change_it" truststore-url="${jboss.server.config.dir}" client-auth="true"/>
</security-domain>
```

上記の設定例は、指定した証明書を検証します。証明書が指定されていない場合や、認証に失敗した場合、この手順はユーザー/パスワードベースの認証にフォールバックします。

通常の式のユーザー名のログインモジュール

Certificate Login Modules の後に Regular Expression User Name Login モジュールを使用すると、ロールが LDAP から取得できるようにプリンシパル名からユーザー名、UID その他フィールドを抽出できます。モジュールには regex という名前のオプションがあります。プリンシパル名に適用される正規表現、後続のログインモジュールに渡される結果を指定します。

この例では、UID=007, EMAILADDRESS=something@something, CN=James Bond, O=SpyAgency の入力プリンシパル名である UID=007 が出力されます。

例11.5 正規表現のユーザー名のログインモジュールの例

```
<login-module
code="org.picketlink.identity.federation.bindings.jboss.auth.RegExUserNameLoginModule"
flag="required">
  <module-option name="password-stacking" value="useFirstPass"/>
  <module-option name="regex" value="UID=(.*?),"/>
</login-module>
```

正規表現の詳細は、java.util.regex.Pattern の http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html クラスドキュメント を参照してください。

バグの報告

11.2. AUTHENTICATION

11.2.1. 認証

認証とは、認証の対象を特定し、その識別情報の真正性を検証することを言います。最も一般的な認証メカニズムは、ユーザー名とパスワードの組み合わせです。その他の一般的な認証メカニズムは、共有鍵、スマートカード、またはフィンガープリントを使用します。認証に成功した結果は、Java Enterprise Edition の宣言的セキュリティーに関してプリンシパルと呼ばれます。

JBoss EAP 6 は、認証モジュールのプラグ可能なシステムを使用して、組織ですでに使用されている認証システムと柔軟性と統合を実現します。各セキュリティードメインには、1 つ以上の設定された

認証モジュールを含めることができます。各モジュールには、動作をカスタマイズするための追加の設定パラメーターが含まれています。認証サブシステムを設定する最も簡単な方法は、Web ベースの管理コンソールを使用することです。

認証は多くの場合リンクされますが、承認は同じではありません。含まれる認証モジュールの多くは承認も処理できます。

バグの報告

11.2.2. セキュリティードメインでの認証の設定

セキュリティードメインの認証を設定するには、管理コンソールにログインして以下の手順を実行します。

手順11.8 セキュリティードメインの認証設定

1. セキュリティードメインの詳細ビューを開きます。
   a. 管理コンソールの上部にある Configuration ラベルをクリックします。
   b. Profile ビューの左側にある Profile 選択ボックスから編集するプロファイルを選択します。
   c. Security メニューを展開し、Security Domains を選択します。
   d. 編集するセキュリティードメインの View リンクをクリックします。
2. 認証サブシステム設定に移動します。

   ビューの上部にある Authentication ラベルを選択していない場合は、これを選択します。

   設定領域は ログインモジュール と Details の 2 つの領域に分割されます。ログインモジュールは、設定の基本単位です。セキュリティードメインには複数のログインモジュールを含めることができ、各ログインモジュールには複数の属性およびオプションを含めることができます。
3. 認証モジュールを追加します。

Add をクリックして JAAS 認証モジュールを追加します。モジュールの詳細を入力します。

Code はモジュールのクラス名です。フラグ 設定は、モジュールが同じセキュリティードメイン内の他の認証モジュールにどのように関係するかを制御します。

フラグの説明

Java Enterprise Edition 6 仕様では、セキュリティーモジュールのフラグについて以下に説明します。以下の一覧は、から http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html#AppendixA 取得されます。詳細は、このドキュメントを参照してください。

フラグ	説明
required	LoginModule は成功する必要があります。成功または失敗しても、LoginModule リストの下方向に進み認証が続行されます。
requisite	LoginModule は成功する必要があります。成功すると、LoginModule リストの下方向に進み認証が続行されます。失敗すると、即座に制御がアプリケーションに戻されます（LoginModule リストの下方向に進まず、認証が続行されません）。
sufficient	LoginModule は成功する必要はありません。成功すると、即座に制御がアプリケーションに戻されます（LoginModule リストの下方向に進まず、認証が続行されません）。失敗すると、LoginModule リストの下方向に進み認証が続行されます。
任意	LoginModule は成功する必要はありません。成功または失敗しても、LoginModule リストの下方向に進み認証が続行されます。

4. 認証設定を編集します。

   モジュールを追加したら、画面の Details セクションで Edit をクリック して、コード またはフラグ を変更できます。Attributes タブが選択されていることを確認してください。

5. モジュールオプションを追加、編集、または削除します (任意)。

   モジュールにオプションを追加する必要がある場合は、Login Modules リストのエントリーをクリックし、ページの Details セクションの Module Options タブを選択します。追加ボタンをクリックして、オプションのキーと値を指定します。Remove ボタンを使用してオプションを削除します。

結果

認証モジュールが、セキュリティードメインに追加され、セキュリティードメインを使用するアプリケーションですぐに利用できます。

jboss.security.security_domain モジュールオプション

デフォルトでは、セキュリティードメインに定義された各ログインモジュールには、**jboss.security.security_domain** モジュールオプションが自動的に追加されます。このオプションは、既知のオプションのみが定義されるようにチェックを行うログインモジュールでは問題が発生します。**IBM Kerberos** ログインモジュールの **com.ibm.security.auth.module.Krb5LoginModule** は、その 1 つです。

このモジュールオプションを追加する挙動を無効にするには、**JBoss EAP 6** の起動時にシステムプロパティーを **true** に設定します。以下を起動パラメーターに追加します。

```
-Djboss.security.disable.secdomain.option=true
```

Web ベースの管理コンソールを使用してこのプロパティーを設定することもできます。スタンドアロンサーバーでは、設定の **Profile** セクションにシステムプロパティーを設定できます。管理対象ドメインでは、各サーバーグループにシステムプロパティーを設定できます。

バグの報告

11.3. JAAS: JAVA 認証および承認サービス

11.3.1. JAAS

JAAS は **Java Authentication and Authorization Service** です。これは **Java EE Spec** に含まれており、プラグ可能な認証および承認でセキュリティープロバイダーからアプリケーションを抽象化できます。

JAAS 1.0 API は、ユーザーの認証および承認用に設計された **Java** パッケージのセットで構成されます。**API** は標準的な **PAM(Pluggable Authentication Modules)**フレームワークの **Java** バージョンを実装し、**Java 2 Platform** アクセス制御アーキテクチャーを拡張し、ユーザーベースの承認をサポートします。

JAAS 認証はプラグ可能な方法で実行されます。これにより、**Java** アプリケーションは基礎となる認証技術から独立し、セキュリティーマネージャーは異なるセキュリティーインフラストラクチャーで動作できるようになります。セキュリティーマネージャーの実装を変更しなくてもセキュリティーイン

フラストラクチャーとの統合を実現できます。変更する必要があるのは、JAAS が使用する認証スタックの設定だけです。

JAAS の詳細は、Java EE JAAS ドキュメント を参照してください。

JBoss Enterprise Application Platform 6 の security サブシステムは JAAS API をベースにしています。

バグの報告

11.3.2. JAAS Core クラス

JAAS コアクラスは、共通、認証、および承認の 3 つのカテゴリーに分けることができます。以下のリストは、本章で説明している EAP security サブシステムの機能を実装するために使用される特定のクラスであるため、共通クラスと認証クラスのみを示しています。

以下は共通のクラスです。

- Subject (javax.security.auth.Subject)

以下は認証クラスです。

- Configuration (javax.security.auth.login.Configuration)
- LoginContext (javax.security.auth.login.LoginContext)

関連付けられたインターフェースは以下のとおりです。

- Principal (java.security.Principal)
- Callback (javax.security.auth.callback.Callback)

- CallbackHandler (javax.security.auth.callback.CallbackHandler)
- LoginModule (javax.security.auth.spi.LoginModule)

バグの報告

11.3.3. Subject クラスおよび Principal クラス

リソースへのアクセスを承認するには、アプリケーションが最初に要求のソースを認証する必要があります。JAAS フレームワークは、リクエストのソースを表すサブジェクトという用語を定義します。Subject クラスは JAAS の中心クラスです。Subject は、個人やサービスなどの単一のエンティティーの情報を表します。エンティティーのプリンシパル、パブリック認証情報、およびプライベートクレデンシャルをカバーします。JAAS API は既存の Java 2 java.security.Principal インターフェースを使用してプリンシパルを表します。これは基本的に型指定された名前です。

認証プロセス中に、サブジェクトには関連するアイデンティティーまたはプリンシパルが入力されます。サブジェクトには多くのプリンシパルが含まれる場合があります。たとえば、あるユーザーが名前プリンシパル(John Doe)、ソーシャルセキュリティー番号プリンシパル(123-45-6789)、およびユーザー名プリンシパル(johnd)を含めることで、サブジェクトを他のサブジェクトと区別するのに役立ちます。サブジェクトに関連付けられたプリンシパルを取得するには、以下の 2 つの方法を使用できます。

```
public Set getPrincipals() {...}
public Set getPrincipals(Class c) {...}
```

getPrincipals() サブジェクトに含まれるすべてのプリンシパルを返します。getPrincipals(Class c) クラス c のインスタンス、またはそのサブクラスの 1 つのインスタンスのみを返します。サブジェクトに一致するプリンシパルがない場合は、空のセットが返されます。

java.security.acl.Group インターフェースは java.security.Principal のサブインターフェースであるため、プリンシパルセット内のインスタンスは、他のプリンシパルまたはプリンシパルのグループの論理グループを表す可能性があることに注意してください。

バグの報告

11.3.4. 発行先認証

サブジェクト認証には JAAS ログインが必要です。JAAS ログイン設定ファイルの説明は、Java ド

キュメントの JAAS Login Configuration File を参照してください。

ログインプロセスは、以下のポイントで構成されます。

1. アプリケーションは LoginContext をインスタンス化し、ログイン設定の名前および CallbackHandler を渡して、設定 LoginModules の要件に応じて Callback オブジェクトを設定します。

2. LoginContext は、名前付きログイン 設定 に含まれるすべて の LoginModule をロードするよう設定を参照します。このような名前付き設定が存在しない場合は、他 の設定がデフォルトとして使用されます。

3. アプリケーションは LoginContext.login メソッドを呼び出します。

4. ログインメソッドは、ロードされた各 LoginModule を呼び出します。各 LoginModule がサブジェクトの認証を試みると、関連付けられた CallbackHandler で handle メソッドを呼び出し、認証プロセスに必要な情報を取得します。必要な情報は、callback オブジェクトの配列で handle メソッドに渡されます。成功すると、LoginModuleは関連するプリンシパルとクレデンシャルをサブジェクトに関連付けます。

5. LoginContext は認証ステータスをアプリケーションに返します。成功はログインメソッドからの戻り値によって表されます。失敗は、ログインメソッドによってスローされる LoginException によって表されます。

6. 認証に成功すると、アプリケーションは LoginContext.getSubject メソッドを使用して認証されたサブジェクトを取得します。

7. サブジェクト認証のスコープが完了したら、LoginContext.logout メソッドを呼び出して、ログイン メソッドによってサブジェクトに関連付けられたすべてのプリンシパルおよび関連情報を削除できます。

LoginContext クラスは、サブジェクトを認証する基本的なメソッドを提供し、基礎となる認証技術に依存しないアプリケーションを開発する方法を提供します。LoginContext は Configuration を確認し、特定のアプリケーションに設定された認証サービスを判別します。LoginModule クラスは認証サービスを表します。そのため、アプリケーション自体を変更せずに、異なるログインモジュールをアプリケーションにプラグインできます。以下のコードは、アプリケーションがサブジェクトを認証するために必要な手順を示しています。

■

```
CallbackHandler handler = new MyHandler();
LoginContext lc = new LoginContext("some-config", handler);

try {
   lc.login();
   Subject subject = lc.getSubject();
} catch(LoginException e) {
   System.out.println("authentication failed");
   e.printStackTrace();
}

// Perform work as authenticated Subject
// ...

// Scope of work complete, logout to remove authentication info
try {
   lc.logout();
} catch(LoginException e) {
   System.out.println("logout failed");
   e.printStackTrace();
}

// A sample MyHandler class
class MyHandler
   implements CallbackHandler
{
   public void handle(Callback[] callbacks) throws
      IOException, UnsupportedCallbackException
   {
      for (int i = 0; i < callbacks.length; i++) {
         if (callbacks[i] instanceof NameCallback) {
            NameCallback nc = (NameCallback)callbacks[i];
            nc.setName(username);
         } else if (callbacks[i] instanceof PasswordCallback) {
            PasswordCallback pc = (PasswordCallback)callbacks[i];
            pc.setPassword(password);
         } else {
            throw new UnsupportedCallbackException(callbacks[i],
                                          "Unrecognized Callback");
         }
      }
   }
}
```

開発者は LoginModule インターフェースの実装を作成して、認証テクノロジーと統合します。これにより、管理者は異なる認証技術をアプリケーションにプラグインできます。複数の LoginModuleをチェーンして、複数の認証テクノロジーが認証プロセスに参加できるようにすることができます。たとえば、LoginModule はユーザー名/パスワードベースの認証を実行できますが、別の LoginModule はスマートカードリーダーや biometric オーセンティケーターなどのハードウェアデバイスと対話できます。

LoginModule のライフサイクルは、クライアントがログインメソッドを作成し、発行する LoginContext オブジェクトによって実行されます。このプロセスは 2 つのフェーズで構成されます。

プロセスのステップは以下のとおりです。

- LoginContext は、パブリック no-arg コンストラクターを使用して設定済みの LoginModule を作成します。

- 各 LoginModule は、初期化メソッドへの呼び出しで初期化されます。Subject 引数は、常に null 以外であることが保証されます。initialize メソッドの署名は次のとおりです。public void initialize(Subject subject, CallbackHandler callbackHandler, Map sharedState, Map options)

- login メソッドは、認証プロセスを開始するために呼び出されます。たとえば、メソッドの実装では、ユーザーにユーザー名とパスワードの入力を求めるプロンプトを出し、NIS や LDAP などの命名サービスに保存されているデータに対して情報を確認することができます。別の実装では、スマートカードとバiometric デバイスへのインターフェースを設定したり、基礎となるオペレーティングシステムからユーザー情報を抽出することもできます。各 LoginModule によるユーザーアイデンティティーの検証は JAAS 認証のフェーズ 1 とみなされます。login メソッドの署名は boolean login() throws LoginException です。LoginException は失敗を示します。戻り値 true はメソッドが成功したことを示しますが、戻り値が false の場合は、ログインモジュールが無視されることを示します。

- LoginContext の 's overall authentication に成功すると、コミット は各 LoginModule で呼び出されます。LoginModule についてフェーズ 1 に成功すると、コミットメソッドはフェーズ 2 を続行し、関連するプリンシパル、パブリッククレデンシャル、またはプライベートクレデンシャルをサブジェクトに関連付けます。LoginModule でフェーズ 1 が失敗すると、コミット するとユーザー名やパスワードなどの以前に保存された認証状態が削除されます。commit メソッドの署名は boolean commit() throws LoginException です。コミットフェーズの完了に失敗した場合は、LoginException を発生させます。true を返すと、メソッドが成功したことを示しますが、false を返すと、ログインモジュールは無視されることを示します。

- LoginContext 's overall authentication fails を選択すると、各 LoginModule で abort メソッドが呼び出されます。abort メソッドは、ログインまたは初期化メソッドによって作成された認証状態を削除または破棄します。アボート メソッドの署名は boolean abort() throws LoginException です。中止 フェーズの完了に失敗した場合は、LoginException を発生させます。true を返すと、メソッドが成功したことを示しますが、false を返すと、ログインモジュールは無視されることを示します。

- ログインに成功した後に認証状態を削除するには、アプリケーションが LoginContext で logout を呼び出します。これにより、各 LoginModule で logout メソッドが呼び出されます。logout メソッドは、コミット 操作時に最初にサブジェクトに関連付けられたプリンシパルおよび認証情報を削除します。認証情報は削除時に破棄される必要があります。logout メソッドの署名は boolean logout() throws LoginException です。ログアウトプロセスの完了に失敗したため、LoginException が発生します。true を返すと、メソッドが成功したことを示しますが、false を返すと、ログインモジュールは無視されることを示します。

LoginModule がユーザーと通信して認証情報を取得する必要がある場合は、callbackHandler オブジェクトを 使用 します。アプリケーションが実装する CallbackHandler インターフェースを作成してそれを LoginContext に渡します。これにより、認証情報が基礎となるログインモジュールに直接送信されます。

ログインモジュールは CallbackHandler を使用して、パスワードやスマートカード PIN などのユーザーからの入力を収集し、ステータス情報などのユーザーに情報を提供します。アプリケーションが CallbackHandler を指定できるようにすることで、アプリケーションがユーザーと対話する方法から、基礎となる LoginModuleは独立しています。たとえば、CallbackHandler 's implementation for a GUI application の場合、ユーザー入力を要求するウィンドウが表示されることがあります。一方、アプリケーションサーバーなど、GUI 以外の環境の CallbackHandler 実装は、アプリケーションサーバー API を使用して認証情報情報を取得するだけです。その CallbackHandler インターフェースには、以下を実装する 1 つのメソッドがあります。

```
void handle(Callback[] callbacks)
   throws java.io.IOException,
        UnsupportedCallbackException;
```

Callback インターフェースは、最後に確認される認証クラスです。これは、前述の例で使用した NameCallback や PasswordCallback を含む、複数のデフォルト実装を提供するタグ付けインターフェースです。LoginModule は Callback を使用して、認証メカニズムで必要な情報を要求します。LoginModuleは、認証のログインフェーズ中に コールバックの配列を CallbackHandler.handle メソッドに直接渡します。callbackhandler が handle メソッドに渡される Callback オブジェクトの使用方法を理解していない場合は、UnsupportedCallbackException をスローし、ログイン呼び出しを中止します。

バグの報告

11.4. JASPI (JAVA AUTHENTICATION SPI FOR CONTAINERS)

11.4.1. JASPI (Java Authentication SPI for Containers) のセキュリティー

Java Authentication SPI for Containers (JASPI または JASPIC) は Java アプリケーションのプラグ可能なインターフェースであり、Java Community Process の JSR-196 に定義されています。JACC は、Java コミュニティープロセスの JSR-196 で定義されています。仕様の http://www.jcp.org/en/jsr/detail?id=196 詳細は、を参照してください。

バグの報告

11.4.2. JASPI (Java Authentication SPI for Containers) のセキュリティーの設定

JASPI プロバイダーに対して認証するには、< authentication-jaspi> 要素 をセキュリティードメインに追加します。設定は標準の認証モジュールと似ていますが、ログインモジュール要素は < login-module-stack> 要素で囲まれて います。設定の構成は次のとおりです。

例11.6 authentication-jaspi 要素の構造

```
<authentication-jaspi>
 <login-module-stack name="...">
  <login-module code="..." flag="...">
   <module-option name="..." value="..."/>
  </login-module>
 </login-module-stack>
 <auth-module code="..." login-module-stack-ref="...">
  <module-option name="..." value="..."/>
 </auth-module>
</authentication-jaspi>
```

ログインモジュール自体は標準的な認証モジュールと全く同じように設定されます。

Web ベースの管理コンソールは JASPI 認証モジュールの設定を公開しないため、設定を EAP_HOME /domain/configuration/domain.xml または*EAP_HOME*/standalone/configuration/standalone.xml に直接追加する必要があります。

バグの報告

11.5. 承認

11.5.1. 認可について

承認は、アイデンティティーに基づいてリソースへのアクセスを許可または拒否するメカニズムです。プリンシパルに追加できる宣言型セキュリティーロールのセットとして実装されます。

JBoss EAP 6 はモジュールシステムを使用して承認を設定します。各セキュリティードメインには、1 つ以上の承認ポリシーが含まれる場合があります。各ポリシーには、動作を定義する基本モジュールがあります。これは、特定のフラグおよび属性で設定されます。承認サブシステムを設定する最も簡単な方法は、Web ベースの管理コンソールを使用することです。

承認は認証とは異なり、通常は認証後に行われます。認証モジュールの多くは承認も処理します。

バグの報告

11.5.2. セキュリティードメインにおける承認の設定

セキュリティードメインの承認を設定するには、管理コンソールにログインして以下の手順を実行します。

手順11.9 セキュリティードメインの承認設定

1. セキュリティードメインの詳細ビューを開きます。
   a. 管理コンソールの上部にある Configuration ラベルをクリックします。
   b. 管理対象ドメインでは、左上の Profile ドロップダウンボックスから編集するプロファイルを選択します。
   c. Security メニュー項目を展開し、Security Domains を選択します。
   d. 編集するセキュリティードメインの View リンクをクリックします。
2. 承認サブシステム設定に移動します。

   画面上部の Authorization ラベルを選択します。

   設定領域は、Policy と Details の 2 つの領域に分割されます。ログインモジュールは、設定の基本単位です。セキュリティードメインには、複数の承認ポリシーを含めることができ、それぞれには複数の属性およびオプションを含めることができます。
3. ポリシーを追加します。

   Add をクリックして JAAS 認証ポリシーモジュールを追加します。モジュールの詳細を入力します。

   Code はモジュールのクラス名です。フラグは、 モジュールが同じセキュリティードメイン内で他の承認ポリシーモジュールにどのように関係するかを制御します。

   フラグの説明

Java Enterprise Edition 6 仕様では、セキュリティーモジュールのフラグについて以下に説明します。以下の一覧は、から http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html#AppendixA 取得されます。詳細は、このドキュメントを参照してください。

フラグ	説明
required	LoginModule は成功する必要があります。成功または失敗しても、LoginModule リストの下方向に進み承認が続行されます。
requisite	LoginModule は成功する必要があります。成功すると、LoginModule リストの下方向に進み承認が続行されます。失敗すると、即座に制御がアプリケーションに戻されます（LoginModule リストの下方向に進まず、承認が続行されません）。
sufficient	LoginModule は成功する必要はありません。成功すると、即座に制御がアプリケーションに戻されます（LoginModule リストの下方向に進まず、承認が続行されません）。失敗すると、LoginModule リストの下方向に進み承認が続行されます。
任意	LoginModule は成功する必要はありません。成功または失敗しても、LoginModule リストの下方向に進み承認が続行されます。

4. 承認設定を編集します。

   モジュールを追加したら、画面の Details セクションで Edit をクリック して、コード またはフラグ を変更できます。Attributes タブが選択されていることを確認してください。

5. モジュールオプションを追加、編集、または削除します (任意)。

   モジュールにオプションを追加する必要がある場合は、Policies リストのエントリーをクリックし、ページの Details セクションの Module Options タブを選択します。Add をクリックして、オプションのキーと値を指定します。Remove ボタンを使用してオプションを削除します。

結果

承認ポリシーモジュールが、セキュリティードメインに追加され、セキュリティードメインを使用するアプリケーションですぐに利用できます。

バグの報告

11.6. JAVA AUTHORIZATION CONTRACT FOR CONTAINERS (JACC)

11.6.1. JACC (Java Authorization Contract for Containers)

JACC (Java Authorization Contract for Containers) はコンテナーと承認サービスプロバイダー間のコントラクトを定義する規格であり、これによりコンテナーによって使用されるプロバイダーの実装が可能になります。これは JSR-115 で定義されています。これは、の Java Community Process Web サイトにあり http://jcp.org/en/jsr/detail?id=115 ます。Java EE バージョン 1.3 より、コア Java Enterprise Edition(Java EE)仕様の一部となっています。

JBoss EAP 6 はセキュリティーサブシステムのセキュリティー機能内に JACC のサポートを実装します。

バグの報告

11.6.2. JACC (Java Authorization Contract for Containers) のセキュリティーの設定

JACC(Java Authorization Contract for Containers)を設定するには、適切なモジュールでセキュリティードメインを設定し、jboss-web.xml を変更して正しいパラメーターが含まれるようにする必要があります。

セキュリティードメインへの JACC サポートの追加

セキュリティードメインに JACC サポートを追加するには、required フラグセットで JACC 承認ポリシーをセキュリティードメインの承認スタックへ追加します。以下は JACC サポートを持つセキュリティードメインの例です。ただし、セキュリティードメインは XML で直接ではなく、管理コンソールまたは管理 CLI で設定されます。

```
<security-domain name="jacc" cache-type="default">
    <authentication>
        <login-module code="UsersRoles" flag="required">
        </login-module>
    </authentication>
    <authorization>
        <policy-module code="JACC" flag="required"/>
    </authorization>
</security-domain>
```

JACC を使用するよう Web アプリケーションを設定

jboss-web.xml は デプロイメントの WEB-INF/ ディレクトリーにあり、Web コンテナーに対する追加の JBoss 固有の設定が含まれています。JACC が有効になっているセキュリティードメインを使用するには、< security-domain > 要素が含まれるようにし、さらに < use-jboss-authorization > 要素

を true に設定する必要があります。以下のアプリケーションは、上記の JACC セキュリティードメインを使用するよう適切に設定されます。

```
<jboss-web>
   <security-domain>jacc</security-domain>
   <use-jboss-authorization>true</use-jboss-authorization>
</jboss-web>
```

JACC を使用するよう EJB アプリケーションを設定

セキュリティードメインと JACC を使用するよう EJB を設定する方法は Web アプリケーションの場合とは異なります。EJB の場合、ejb-jar.xml 記述子にメソッドまたはメソッドのグループ上でメソッド *パーミッション* を宣言できます。< ejb-jar > 要素内では、すべての子 &lt ;method-permission> 要素に JACC ロールに関する情報が含まれます。詳細は、設定例を参照してください。EJBMethodPermission クラスは Java Enterprise Edition 6 API の一部であり、で http://docs.oracle.com/javaee/6/api/javax/security/jacc/EJBMethodPermission.html 説明されています。

例11.7 EJB の JACC メソッドパーミッション例

```
<ejb-jar>
  <assembly-descriptor>
    <method-permission>
      <description>The employee and temp-employee roles may access any method of the
EmployeeService bean </description>
      <role-name>employee</role-name>
      <role-name>temp-employee</role-name>
      <method>
        <ejb-name>EmployeeService</ejb-name>
        <method-name>*</method-name>
      </method>
    </method-permission>
  </assembly-descriptor>
</ejb-jar>
```

Web アプリケーションと同様にセキュリティードメインを使用して EJB の認証および承認メカニズムを指定することも可能です。セキュリティードメインは < security > 子要素の jboss-ejb3.xml 記述子で宣言されます。セキュリティードメインの他に、EJB が実行されるプリンシパルを変更する < run-as-principal > を指定することもできます。

例11.8 EJB におけるセキュリティードメイン宣言の例

```
<ejb-jar>
 <assembly-descriptor>
  <security>
    <ejb-name>*</ejb-name>
```

```
    <security-domain>myDomain</security-domain>
    <run-as-principal>myPrincipal</run-as-principal>
  </security>
  </assembly-descriptor>
</ejb-jar>
```

バグの報告

11.6.3. XACML を使用した粒度の細かい承認

11.6.3.1. 粒度の細かい承認および XACML

意思決定に関与する要件や、意思決定プロセスに関与する複数の変数に対する承認の細かい承認は、モジュールにアクセスするための承認のベースとなります。そのため、Fine Grained Authorization のプロセスは複雑です。

JBoss は XACML をメディアとして使用し、Fine Grained Authorization を実現します。XACML は、Fine Grained Authorization の複雑な性質に標準ベースのソリューションを提供します。XACML は、意思決定のポリシー言語やアークテクチャーを定義します。XACML アーキテクチャーには、通常のプログラムフローの要求をインターセプトする PEP(Policy Enforcement Point)が含まれ、PDP(Policy Decision Point)に、PDP に関連するポリシーに基づいてアクセスを決定するよう要求します。PDP は PEP によって作成された XACML リクエストを評価し、ポリシーを実行して以下のアクセスの決定を行います。

- PERMIT: アクセスは承認されます。
- DENY: アクセスは拒否されます。
- INDETERMINATE: PDP にエラーがあります。
- NOTAPPLICABLE - リクエストにない属性があるか、一致するポリシーがありません。

XACML の機能は次のとおりです。

- Oasis XACML v2.0 ライブラリー
- JAXB v2.0 ベースのオブジェクトモデル
- XACML ポリシーおよび属性を保存/取得するための ExistDB 統合

バグの報告

11.6.3.2. 粒度の細かい承認のための XACML の設定

XACML を設定する手順は次のとおりです。

手順11.10 XACML の設定

1. 単一の jar ファイルであるライブラリーをダウンロードします。
2. XACML のポリシーファイルを 1 つ以上作成
   - WEB-INF/classes で、すべてのポリシーを保存する policies ディレクトリーを作成します。
   - WEB-INF/classes ディレクトリーに policyConfig.xml を作成します。

     以下は、2 種類のポリシーセットを定義できます。
     - Role Permission Policy Sets(RPS)
     - パーミッションポリシーセット(PPS)

> 例11.9 Role Permission Policy Sets(RPS)
>
> Employee

```
    <PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    PolicySetId="RPS:employee:role"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-
algorithm:permit-overrides">
    <Target>
    <Subjects>
    <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
    <AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#anyURI">employee</AttributeValu
e>
    <SubjectAttributeDesignator
    AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
    </SubjectMatch>
    </Subject>
    </Subjects>
    </Target>
    <!-- Use permissions associated with the employee role -->
    <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
    </PolicySet>
```

Manager

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
PolicySetId="RPS:manager:role"
PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-
algorithm:permit-overrides">
<Target>
<Subjects>
<Subject>
<SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">manager</AttributeValu
e>
<SubjectAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
</SubjectMatch>
</Subject>
</Subjects>
</Target>
<!-- Use permissions associated with the manager role -->
```

```
<PolicySetIdReference>PPS:manager:role</PolicySetIdReference>
</PolicySet>
```

例11.10 パーミッションポリシーセット(PPS)

Employee

```
   <PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
      PolicySetId="PPS:employee:role"
      PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-
algorithm:permit-overrides">
      <Target />
      <!-- Permissions specifically for the employee role -->
      <Policy PolicyId="Permissions:specifically:for:the:employee:role"
         RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
         <Target />
         <!-- Permission to create a purchase order -->
         <Rule RuleId="Permission:to:create:a:purchase:order" Effect="Permit">
            <Target>
               <Resources>
                  <Resource>
                     <ResourceMatch
                        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">purchase order
                        </AttributeValue>
                        <ResourceAttributeDesignator
                           AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
id" DataType="http://www.w3.org/2001/XMLSchema#string" />
                     </ResourceMatch>
                  </Resource>
               </Resources>
               <Actions>
                  <Action>
                     <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">create</AttributeValue>
                        <ActionAttributeDesignator AttributeId="urn:action-id"
                           DataType="http://www.w3.org/2001/XMLSchema#string" />
                     </ActionMatch>
                  </Action>
               </Actions>
            </Target>
         </Rule>
      </Policy>
```

```
    <!-- HasPrivilegesOfRole Policy for employee role -->
  <Policy PolicyId="Permission:to:have:employee:role:permissions"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
    <Target />
    <!-- Permission to have employee role permissions -->
    <Rule RuleId="Permission:to:have:employee:permissions" Effect="Permit">
      <Condition>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-is-
in">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">employee</AttributeValu
e>
            <ResourceAttributeDesignator
              AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
          </Apply>
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-is-
in">
            <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">urn:oasis:names:tc:xac
ml:2.0:actions:hasPrivilegesOfRole
            </AttributeValue>
            <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
              DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
          </Apply>
        </Apply>
      </Condition>
    </Rule>
  </Policy>
  </PolicySet>
```

Manager

```
<PolicySet xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
    PolicySetId="PPS:manager:role"
    PolicyCombiningAlgId="urn:oasis:names:tc:xacml:1.0:policy-combining-
algorithm:permit-overrides">
    <Target />
    <!-- Permissions specifically for the manager role -->
    <Policy PolicyId="Permissions:specifically:for:the:manager:role"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
      <Target />
      <!-- Permission to sign a purchase order -->
      <Rule RuleId="Permission:to:sign:a:purchase:order" Effect="Permit">
```

```
            <Target>
               <Resources>
                  <Resource>
                     <ResourceMatch
                        MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">purchase order
                        </AttributeValue>
                        <ResourceAttributeDesignator
                           AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
id" DataType="http://www.w3.org/2001/XMLSchema#string" />
                     </ResourceMatch>
                  </Resource>
               </Resources>
               <Actions>
                  <Action>
                     <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                        <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">sign</AttributeValue>
                        <ActionAttributeDesignator AttributeId="urn:action-id"
                           DataType="http://www.w3.org/2001/XMLSchema#string" />
                     </ActionMatch>
                  </Action>
               </Actions>
            </Target>
         </Rule>
      </Policy>
      <!-- HasPrivilegesOfRole Policy for manager role -->
   <Policy PolicyId="Permission:to:have:manager:role:permissions"
      RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
algorithm:permit-overrides">
      <Target />
      <!-- Permission to have manager role permissions -->
      <Rule RuleId="Permission:to:have:manager:permissions" Effect="Permit">
         <Condition>
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-is-
in">
                  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">manager</AttributeValu
e>
                  <ResourceAttributeDesignator
                     AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
               </Apply>
               <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:anyURI-is-
in">
                  <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#anyURI">urn:oasis:names:tc:xac
ml:2.0:actions:hasPrivilegesOfRole
                  </AttributeValue>
                  <ActionAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                     DataType="http://www.w3.org/2001/XMLSchema#anyURI" />
               </Apply>
```

```
            </Apply>
          </Condition>
        </Rule>
      </Policy>
        <!-- Include permissions associated with employee role -->
        <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
      </PolicySet>
```

3. **XACML エンジンの設定ファイルを作成します。**

ロケーターを設定し、ポリシーが保存されるディレクトリーに言及するために設定ファイルが作成されます。

例11.11 設定ファイル

Directory Of the Policy Files のみを示す設定ファイル。

```
    <ns:jbosspdp xmlns:ns="urn:jboss:xacml:2.0">
      <ns:Policies>
       <ns:PolicySet>
          <ns:Location>test/policies/rbac/</ns:Location>
       </ns:PolicySet>
      </ns:Policies>
      <ns:Locators>
       <ns:Locator
Name="org.jboss.security.xacml.locators.JBossRBACPolicySetLocator"/>
      </ns:Locators>
    </ns:jbosspdp>
```

ポリシーセットの定義

```
<ns:jbosspdp xmlns:ns="urn:jboss:xacml:2.0">
  <ns:Policies>
    <ns:PolicySet>
      <ns:Location>test/policies/rbac/employee-PPS-policyset.xml</ns:Location>
    </ns:PolicySet>
    <ns:PolicySet>
      <ns:Location>test/policies/rbac/manager-PPS-policyset.xml</ns:Location>
```

```
    </ns:PolicySet>
    <ns:PolicySet>
      <ns:Location>test/policies/rbac/employee-RPS-policyset.xml</ns:Location>
    </ns:PolicySet>
    <ns:PolicySet>
      <ns:Location>test/policies/rbac/manager-RPS-policyset.xml</ns:Location>
    </ns:PolicySet>
  </ns:Policies>
  <ns:Locators>
    <ns:Locator
Name="org.jboss.security.xacml.locators.JBossRBACPolicySetLocator"/>
  </ns:Locators>
</ns:jbosspdp>
```

4. Policy Decision Point(PDP)を作成して、設定ファイルに渡します。

5. Policy Enforcement Point(PEP)で、コンテキストに基づいて XACML リクエストを作成します。XACML リクエストを PDP に渡して、以下のアクセス決定の 1 つを取得します。

- permit
- 却下
- Indeterminate
- 適用外

例11.12 Decisions へのアクセス

許可条件

```
<Request
    xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
      access_control-xacml-2.0-context-schema-os.xsd">
<Subject>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
 DataType="http://www.w3.org/2001/XMLSchema#string">
<AttributeValue>Anne</AttributeValue>
</Attribute>

<Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
 DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>manager</AttributeValue>
</Attribute>
</Subject>

<Resource>
<Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<AttributeValue>manager</AttributeValue>
</Attribute>
</Resource>

<Action>
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
 DataType="http://www.w3.org/2001/XMLSchema#anyURI">

<AttributeValue>urn:oasis:names:tc:xacml:2.0:actions:hasPrivilegesOfRole</Attrib
uteValue>
</Attribute>
</Action>
</Request>
```

パーミッションの拒否

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="urn:oasis:names:tc:xacml:2.0:context:schema:os
      access_control-xacml-2.0-context-schema-os.xsd">
   <Subject>
      <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
         DataType="http://www.w3.org/2001/XMLSchema#string">
         <AttributeValue>Anne</AttributeValue>
      </Attribute>

      <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
         DataType="http://www.w3.org/2001/XMLSchema#anyURI">
```

```
            <AttributeValue>manager</AttributeValue>
        </Attribute>
    </Subject>

    <Resource>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:2.0:subject:role"
            DataType="http://www.w3.org/2001/XMLSchema#anyURI">
            <AttributeValue>manager</AttributeValue>
        </Attribute>
    </Resource>

    <Action>
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            DataType="http://www.w3.org/2001/XMLSchema#anyURI">
            <AttributeValue>urn:nobody</AttributeValue>
        </Attribute>
    </Action>
</Request>
```

バグの報告

11.7. セキュリティー監査

11.7.1. セキュリティー監査

セキュリティー監査とは、security サブシステム内で発生するイベントに対応して、ログへの書き込みなどのイベントをトリガーすることを指します。監査方法は、認証、承認、およびセキュリティーマッピングの詳細とともにセキュリティードメインの一部として設定されます。

監査は、*プロバイダーモジュール* を使用します。含まれるもののいずれかを使用するか、独自の実装を行うことができます。

バグの報告

11.7.2. セキュリティー監査の設定

セキュリティードメインのセキュリティー監査を設定するには、管理コンソールにログインして以下の手順を実行します。

手順11.11 セキュリティードメインのセキュリティー監査の設定

1. **セキュリティードメインの詳細ビューを開きます。**

   a. **画面上部の Configuration をクリックします。**

   b. **管理対象ドメインでは、左上の Profile 選択ボックスから変更するプロファイルを選択します。**

   c. **Security メニューを展開し、Security Domains を選択します。**

   d. **編集するセキュリティードメインの 表示 をクリックします。**

2. **監査サブシステム設定に移動します。**

   画面上部の Audit タブを選択します。

   設定領域が Provider Modules と Details の 2 つの領域に分割されます。プロバイダーモジュールは、設定の基本単位です。セキュリティードメインには、属性およびオプションを含む複数のプロバイダーモジュールを含めることができます。

3. **プロバイダーモジュールを追加します。**

   Add をクリックします。Code セクションにプロバイダーモジュールのクラス名を入力します。

4. **モジュールの挙動を確認します。**

   監査モジュールの目的は、security サブシステムでイベントを監視する方法を提供することです。この監視は、ログファイル、メール通知、またはその他の測定可能な監査メカニズムに書き込む方法で実行できます。

   たとえば、JBoss EAP 6 にはデフォルトで LogAuditProvider モジュールが含まれます。上記の手順に従って有効にすると、この監査モジュールは、*EAP_HOME* ディレクトリー内のログサブフォルダーの audit. log ファイルにセキュリティー通知を書き込みます。

   上記の手順が LogAuditProvider のコンテキストで機能しているかどうかを確認するには、通知をトリガーする可能性のあるアクションを実行してから、監査ログファイルを確認します。

含まれるセキュリティー監査プロバイダーモジュールの完全リストは、以下を参照してください。 「含まれるセキュリティー監査プロバイダーモジュール」

5. モジュールオプションを追加、編集、または削除します (任意)。<remark>Bugzilla のバグの内容を反映済み</remark>

モジュールにオプションを追加するには、モジュール リストの エントリー をクリックし、ページの Details セクションの Module Options タブを選択します。Add をクリックして、オプションのキーと値を指定します。

すでに存在するオプションを編集するには、Remove をクリックして 削除 し、Add をクリックして、正しいオプションで再度追加します。

結果

セキュリティー監査モジュールは、セキュリティードメインに追加され、セキュリティードメインを使用するアプリケーションですぐに利用できます。

バグの報告

11.7.3. 新しいセキュリティープロパティー

JBoss EAP バージョン 6.2.2 以降のセキュリティー監査機能に新しいシステムプロパティーが追加されました。これらの新しいプロパティーは、特に BASIC または FORM ベースの認証に関連するシナリオにおいて、Web 要求コンポーネントのプレーンテキストのロギングに関連するセキュリティーの問題を軽減します。

新しいプロパティーを使用すると、Web 要求のどのコンポーネントを監査ログでキャプチャーするかをより詳細に制御できます（パラメーター、Cookie、ヘッダーまたは属性）。これらのコンポーネントは、新しいプロパティーを使用してマスクすることもできます。

新しいプロパティーは以下のとおりです。

表11.1 新しいセキュリティープロパティー

名前	説明	可能な値	動作	デフォルト

名前	説明	可能な値	動作	デフォルト
org.jboss.security.web.audit	このプロパティーは、Web 要求のセキュリティー監査の粒度を制御します。	**off**、**ヘッダー**、**Cookie**、**パラメーター**、**属性**	指定されたコンポーネント（またはコンポーネントのカンマ区切りグループ）は、Web 要求から監査されます。	**headers,parameters**
org.jboss.security.web.audit.mask	このプロパティーは、Web リクエストのヘッダー、パラメーター、Cookie、および属性と照合される文字列のリストを指定するために使用できます。指定されたマスクに一致する要素は、セキュリティー監査ロギングから除外されます。	ヘッダー、パラメーター、Cookie、および属性のキーを示すコンマ区切りの文字列。	現在、マスクのマッチングは厳格ではなくあいまいです。たとえば、承認のマスクは、**authorization** というヘッダーと custom_authorization と呼ばれるパラメーターの両方をマスクします。今後のリリースでは、厳密なマスクが導入される可能性があります。	j_password,authorization

バグの報告

11.8. セキュリティーマッピング

11.8.1. セキュリティーマッピング

セキュリティーマッピングを使用すると、認証または承認が実行された後、情報がアプリケーションに渡される前に認証情報と承認情報を組み合わせることができます。

プリンシパル (認証)、ロール (承認)、またはクレデンシャル (プリンシパルやロールでない属性) をマッピングすることが可能です。

ロールマッピングは、認証後にサブジェクトへロールを追加、置換、または削除するために使用されます。

プリンシパルマッピングは、認証後にプリンシパルを変更するために使用されます。

属性マッピングは、外部システムからの属性値をアプリケーションで使用するために変換したり、逆にそのような外部システムへ属性を変換したりするために使用されます。<remark>Bugzilla のフィードバックを反映済み</remark>

バグの報告

11.8.2. セキュリティードメインでのセキュリティーマッピングの設定

セキュリティードメインのセキュリティーマッピングを設定するには、管理コンソールにログインして以下の手順を実行します。

手順11.12 セキュリティードメインでのセキュリティーマッピングの設定

1. セキュリティードメインの詳細ビューを開きます。
   a. 管理コンソールの上部にある Configuration ラベルをクリックします。
   b. 管理対象ドメインでは、左上の Profile 選択ボックスからプロファイルを選択します。
   c. Security メニューを展開し、Security Domains を選択します。
   d. 編集するセキュリティードメインの 表示 をクリックします。
2. マッピングサブシステム設定に移動します。

   画面上部の Mapping ラベルを選択します。

   設定領域は、モジュール と Details の 2 つの領域に分割されます。マッピングモジュールは、設定の基本単位です。セキュリティードメインには複数のマッピングモジュールを含めることができ、各モジュールには複数の属性およびオプションを含めることができます。
3. セキュリティーマッピングモジュールを追加します。

   Add をクリックします。

   モジュールの詳細を入力します。Code はモジュールのクラス名です。Type フィールド

は、このモジュールが実行するマッピングのタイプを参照します。使用できる値は principal、role、attribute、または credential です。

4. セキュリティーマッピングモジュールを編集します。

   モジュールを追加したら、その コード または タイプを変更できます。

   a. Attributes タブを選択します。

   b. 画面の Details セクションで Edit をクリックします。

5. モジュールオプションを追加、編集、または削除します (任意)。<remark>Bugzilla のバグの内容を反映済み</remark>

   モジュールにオプションを追加するには、モジュール リストの エントリー をクリックし、ページの Details セクションの Module Options タブを選択します。Add をクリックして、オプションのキーと値を指定します。

   すでに存在するオプションを編集するには、Remove をクリックして 削除 し、新しい値で再度追加します。

   Remove ボタンを使用してオプションを削除します。

結果

セキュリティーマッピングモジュールは、セキュリティードメインに追加され、セキュリティードメインを使用するアプリケーションですぐに利用できます。

バグの報告

11.9. アプリケーションでのセキュリティードメインの使用

概要

アプリケーションでセキュリティードメインを使用するには、最初にサーバーの設定でセキュリティードメインを定義し、アプリケーションのデプロイメント記述子でアプリケーションに対して有効にする必要があります。次に、必要なアノテーションを使用する EJB に追加する必要があります。このトピックでは、アプリケーションでセキュリティードメインを使用するために必要な手順について説明します。

警告

アプリケーションが、認証キャッシュを使用するセキュリティードメインの一部である場合、セキュリティードメインの他のアプリケーションもそのアプリケーションのユーザー認証を使用できます。

手順11.13 セキュリティードメインを使用するようアプリケーションを設定

1. **セキュリティードメインの定義**

   サーバーの設定ファイルでセキュリティードメインを定義した後、アプリケーションの記述子ファイルでアプリケーションに対して有効にする必要があります。

   a. **サーバーの設定ファイルへセキュリティードメインを設定**

      セキュリティードメインは、サーバーの設定ファイルの **security** サブシステムで設定されます。**JBoss EAP 6** インスタンスが管理対象ドメインで稼働している場合、これは **domain/configuration/domain.xml** ファイルになります。**JBoss EAP 6** インスタンスがスタンドアロンサーバーとして実行されている場合、これは **standalone/configuration/standalone.xml** ファイルになります。

      他 のjboss-web-policy、および jboss-ejb-policy セキュリティードメインは、**JBoss EAP 6** ではデフォルトで提供されます。以下の **XML** の例は、サーバーの設定ファイルの **security** サブシステムからコピーされました。

      セキュリティードメインの **cache-type** 属性は、認証チェックをより高速にするためにキャッシュを指定します。許可される値は デフォルト で簡単なマップをキャッシュとして使用するか、**infinispan** が **Infinispan** キャッシュを使用します。

```
<subsystem xmlns="urn:jboss:domain:security:1.2">
   <security-domains>
      <security-domain name="other" cache-type="default">
         <authentication>
            <login-module code="Remoting" flag="optional">
               <module-option name="password-stacking" value="useFirstPass"/>
            </login-module>
            <login-module code="RealmDirect" flag="required">
               <module-option name="password-stacking" value="useFirstPass"/>
            </login-module>
         </authentication>
      </security-domain>
      <security-domain name="jboss-web-policy" cache-type="default">
         <authorization>
            <policy-module code="Delegating" flag="required"/>
```

```
            </authorization>
        </security-domain>
        <security-domain name="jboss-ejb-policy" cache-type="default">
            <authorization>
                <policy-module code="Delegating" flag="required"/>
            </authorization>
        </security-domain>
    </security-domains>
</subsystem>
```

管理コンソールまたは CLI を使用して、追加のセキュリティードメインを必要に応じて設定できます。

b. アプリケーションの記述子ファイルでのセキュリティードメインの有効化

セキュリティードメインは、アプリケーションの WEB-INF/jboss -web.xml ファイルの < jboss-web > 要素の <security -domain> 子要素に指定されます。以下の例は、my-domain という名前のセキュリティードメインを設定します。

```
<jboss-web>
    <security-domain>my-domain</security-domain>
</jboss-web>
```

これは、WEB-INF/jboss-web.xml 記述子で指定できる多くの設定の 1 つのみです。

2. EJB への必要なアノテーションの追加

@SecurityDomain および @RolesAllowed アノテーションを使用して EJB でセキュリティーを設定します。以下の EJB コードの例では、guest ロールのユーザーが 他 のセキュリティードメインへのアクセスを制限します。

```
package example.ejb3;

import java.security.Principal;

import javax.annotation.Resource;
import javax.annotation.security.RolesAllowed;
import javax.ejb.SessionContext;
import javax.ejb.Stateless;

import org.jboss.ejb3.annotation.SecurityDomain;

/**
```

```
 * Simple secured EJB using EJB security annotations
 * Allow access to "other" security domain by users in a "guest" role.
 */
@Stateless
@RolesAllowed({ "guest" })
@SecurityDomain("other")
public class SecuredEJB {

  // Inject the Session Context
  @Resource
  private SessionContext ctx;

  /**
   * Secured EJB method using security annotations
   */
  public String getSecurityInfo() {
    // Session context injected using the resource annotation
    Principal principal = ctx.getCallerPrincipal();
    return principal.toString();
  }
}
```

その他のコード例は、Red Hat カスタマーポータルで利用できる JBoss EAP 6 Quickstarts バンドルの ejb-security クイックスタートを参照してください。

バグの報告

第12章 シングルサインオン (SSO)

12.1. WEB アプリケーションのシングルサインオン(SSO)について

概要

シングルサインオン(SSO) は、1 つのリソースに認証を行い、他のリソースへのアクセスを暗黙的に許可します。

クラスター化および非クラスター化 SSO

非クラスター SSO は、同じ仮想ホストのアプリケーションへのアクセス情報の共有を制限します。また、ホストに障害が発生した場合の回復性はありません。クラスター化された SSO データは複数のホストのアプリケーション間で共有でき、フェイルオーバーに回復性があります。さらに、クラスター化された SSO はロードバランサーからリクエストを受信できます。

SSO の仕組み

リソースが保護されていない場合、ユーザーは認証を全く行いません。ユーザーが保護されたリソースにアクセスする場合は、ユーザーを認証する必要があります。

認証に成功すると、ユーザーに関連付けられたロールが保存され、関連するすべてのリソースの認証に使用されます。

ユーザーがアプリケーションからログアウトしたり、アプリケーションがプログラムでセッションを無効にする場合、永続化された認証データはすべて削除され、プロセスが引き継ぎされます。

他のセッションがまだ有効であれば、セッションのタイムアウトは SSO セッションを無効にしません。

バグの報告

12.2. WEB アプリケーションの CLUSTERED SINGLE SIGN ON(SSO)について

シングルサインオン(SSO)は、ユーザーが単一の Web アプリケーションに対して認証できる機能です。クラスター化された SSO は認証情報をクラスター化されたキャッシュに保存します。これにより、複数のサーバーのアプリケーションが情報を共有でき、情報に回復性を持たせることができます。

サポートされる SSO メカニズムの一部（Kerberos、PicketLink SAML など）は正常に機能するためにバルブが必要です。バルブにはサーブレットフィルターと同様の機能がありますが、それらはコンテナー管理認証の前に処理されます。Web アプリケーションのバルブは jboss-web.xml デプロイメント記述子に定義できます。

バグの報告

12.3. 右の SSO 実装の選択

JBoss EAP 6 は、Web アプリケーション、EJB アプリケーション、Web サービス、またはその他のタイプである Java Enterprise Edition(EE)アプリケーションを実行します。シングルサインオン(SSO)を使用すると、これらのアプリケーション間でセキュリティーコンテキストとアイデンティティー情報を伝播できます。利用できる SSO ソリューションがいくつかありますが、適切なソリューションを選択する方法は要件によって異なります。

クラスター化された web アプリケーションとクラスター化された SSO には異なる違いがあることに注意してください。クラスター化された web アプリケーションは、そのアプリケーションをホストするための負荷を分散するためにクラスターのノード全体で分散されます。分散可能とマークされた場合、新しいセッションはすべてクラスターの他のメンバーに複製されます。アプリケーションは、web.xml デプロイメント記述子の <distributable/> タグを使用してクラスターノード全体に分散可能とマークされます。クラスター化された SSO は、アプリケーション自体がクラスター化されているかどうかに関わらず、セキュリティーコンテキストとアイデンティティー情報のレプリケーションを可能にします。これらの技術は一緒に使用できますが、概念は異なります。

Kerberos ベースのデスクトップ SSO

組織が Microsoft Active Directory などの Kerberos ベースの認証および認可システムをすでに使用している場合、同じシステムを使用して JBoss EAP 6 で実行されているエンタープライズアプリケーションに対して透過的に認証できます。

非クラスター化 Web アプリケーション SSO

単一インスタンスで複数のアプリケーションを実行し、これらのアプリケーションの SSO セッションレプリケーションを有効にする必要がある場合は、クラスター化されていない SSO が要件を満たしている。

クラスター化された Web アプリケーション SSO

単一のアプリケーションまたは複数のアプリケーションのいずれかをクラスター全体で実行し、これらのアプリケーションの SSO セッションレプリケーションを有効にする必要がある場合は、クラスター化された SSO が要件を満たすようになります。

バグの報告

12.4. WEB アプリケーションでのシングルサインオン(SSO)の使用

概要

シングルサインオン(SSO)機能は web および Infinispan サブシステムによって提供されます。以下の手順に従って、Web アプリケーションで SSO を設定します。

前提条件

- 認証とアクセスを処理する設定済みのセキュリティードメイン。
- infinispan サブシステム。デフォルトでは、管理対象ドメインとスタンドアロンサーバーのすべてのプロファイルに存在します。
- web cache-container および SSO replicated-cache初期設定ファイルには web キャッシュコンテナーがすでに含まれ、一部の設定ファイルにも SSO replicated-cache がすでに含まれています。以下のコマンドを使用して、SSO replicated-cache をチェックし、有効にします。このコマンドは、管理対象ドメインの ha プロファイルを変更することに注意してください。コマンドを変更して別のプロファイルを使用するか、スタンドアロンサーバーでコマンドの /profile=ha 部分を削除できます。

> **例12.1 web cache-container の有無を確認します。**
>
> 上記のプロファイルと設定には、デフォルトで web キャッシュコンテナーが含まれます。以下のコマンドを使用して、存在を確認します。別のプロファイルを使用する場合は、ha の代わりに名前を置き換えます。
>
> ```
> /profile=ha/subsystem=infinispan/cache-container=web/:read-
> resource(recursive=false,proxies=false,include-runtime=false,include-defaults=true)
> ```
>
> 結果が 成功 した場合、サブシステムが存在します。それ以外の場合は、追加する必要があります。

> **例12.2 web キャッシュコンテナーの追加**
>
> 以下の 3 つのコマンドを使用して、web キャッシュコンテナーを設定に対して有効にします。適切なプロファイルの名前と、その他のパラメーターを変更します。このパラメー

ターは、デフォルト設定で使用されるパラメーターです。

```
/profile=ha/subsystem=infinispan/cache-container=web:add(aliases=["standard-session-cache"],default-cache="repl",module="org.jboss.as.clustering.web.infinispan")
```

```
/profile=ha/subsystem=infinispan/cache-container=web/transport=TRANSPORT:add(lock-timeout=60000)
```

```
/profile=ha/subsystem=infinispan/cache-container=web/replicated-cache=repl:add(mode="ASYNC",batching=true)
```

例12.3 SSO replicated-cache を確認します。

以下の管理 CLI コマンドを実行します。

```
/profile=ha/subsystem=infinispan/cache-container=web/:read-resource(recursive=true,proxies=false,include-runtime=false,include-defaults=true)
```

以下のような出力を探します。 "sso" => {

これが見つからない場合、SSO replicated-cache は設定に表示されません。

例12.4 SSO replicated-cache を追加します。

```
/profile=ha/subsystem=infinispan/cache-container=web/replicated-cache=sso:add(mode="SYNC", batching=true)
```

管理対象ドメインでのクラスター化された SSO の設定

web サブシステムは SSO を使用するように設定する必要があります。以下のコマンドは、default-host と呼ばれる仮想サーバーおよび cookie ドメイン domain.com で SSO を有効にします。キャッシュ名は sso で、再認証は無効になります。

```
/profile=ha/subsystem=web/virtual-server=default-host/sso=configuration:add(cache-container="web",cache-name="sso",reauthenticate="false",domain="domain.com")
```

SSO 情報を共有する各アプリケーションは、jboss-web.xml デプロイメント記述子と web.xml 設定ファイルで同じ <security-domain> を使用するよう設定する必要があります。

スタンドアロンサーバー向けのクラスター化された SSO または非クラスター化 SSO の設定

サーバープロファイルの web サブシステムの下に sso を設定します。属性 ClusteredSingleSignOn が存在する場合、cache-container バージョンが使用されます。それ以外の場合は、標準の SingleSignOn クラスが使用されます。

例12.5 非クラスター化 SSO 設定の例

```
/subsystem=web/virtual-server=default-host/sso=configuration:add(reauthenticate="false")
```

セッションの無効化

アプリケーションは、javax.servlet.http.HttpSession.invalidate() メソッドを呼び出すことで、プログラム的にセッションを無効にすることができます。

バグの報告

12.5. KERBEROS について

Kerberos は、クライアント/サーバーアプリケーションのネットワーク認証プロトコルです。これは、秘密鍵の対称暗号を使用して、安全な方法でセキュアでないネットワーク全体で認証を可能にします。

Kerberos はチケットと呼ばれるセキュリティートークンを使用します。セキュリティー保護されたサービスを使用するには、ネットワーク上のサーバーで実行しているサービスである TGT(Ticket Granting Service)からチケットを取得する必要があります。チケットの取得後、認証サービス(AS)からサービスチケット(ST)を要求します。これは、ネットワーク上で実行中の別のサービスです。次に、ST を使用して、使用するサービスへの認証を行います。TGS と AS はどちらも Key Distribution Center(KDC)と呼ばれるエンクロージングサービス内で実行されます。

Kerberos はクライアントサーバー環境で使用されるように設計されており、Web アプリケーションまたはシンクライアント環境でほとんど使用されません。ただし、多くの組織は、デスクトップ認証に Kerberos システムをすでに使用しているため、Web アプリケーション用に 2 つ目のシステムを作成するのではなく、既存のシステムを再利用します。Kerberos は Microsoft Active Directory には欠かせない要素で、Red Hat Enterprise Linux 環境の多くでも使用されています。

バグの報告

12.6. SPNEGO

SPNEGO(Simple and Protected GSS_API Negotiation Mechanism)は、Web アプリケーションで使用するために Kerberos ベースのシングルサインオン(SSO)環境を拡張するメカニズムを提供します。

web ブラウザーなどのクライアントコンピューター上のアプリケーションが web サーバーの保護ページにアクセスしようとすると、サーバーは承認が必要であることを伝えます。その後、アプリケーションは Kerberos Key Distribution Center(KDC)からサービスチケットを要求します。チケットの取得後、アプリケーションはこのチケットをSPNEGO 用にフォーマットされたリクエストにラップし、ブラウザーを介して Web アプリケーションに返信します。デプロイされた Web アプリケーションを実行している Web コンテナーはリクエストをアンパックし、チケットを認証します。認証に成功するとアクセスが許可されます。

SPNEGO は、Red Hat Enterprise Linux に含まれる Kerberos サービスや Microsoft Active Directory の必須部分である Kerberos サーバーなど、すべてのタイプの Kerberos プロバイダーと動作します。

バグの報告

12.7. MICROSOFT ACTIVE DIRECTORY について

Microsoft Active Directory は、Microsoft が開発したディレクトリーサービスで、Microsoft Windows ドメインでユーザーとコンピューターを認証します。Microsoft Windows Server の一部として含まれています。Microsoft Windows Server のコンピューターはドメインコントローラーと呼ばれます。Samba サービスを実行している Red Hat Enterprise Linux サーバーは、この種のネットワークでもドメインコントローラーとして機能します。

Active Directory は、連携する 3 つのコアテクノロジーに依存します。

- ユーザー、コンピューター、パスワード、およびその他のリソースに関する情報を格納する軽量 Directory Access Protocol(LDAP)。
- Kerberos: ネットワーク上でセキュアな認証を提供します。

- ネットワーク上のコンピューターおよびその他のデバイスの IP アドレスとホスト名間のマッピングを提供するドメインネームサービス(DNS)。

バグの報告

12.8. WEB アプリケーション用の KERBEROS または MICROSOFT ACTIVE DIRECTORY DESKTOP SSO の設定

はじめに

組織の既存の Kerberos ベースの認証および承認インフラストラクチャー（Microsoft Active Directory など）を使用して web または EJB アプリケーションを認証するには、JBoss EAP 6 に組み込まれている JBoss Negotiation 機能を使用できます。Web アプリケーションを適切に設定した場合には、正常なデスクトップまたはネットワークログインで Web アプリケーションに対する透過的な認証を実行できるため、追加のログインプロンプトは必要ありません。

以前のバージョンとプラットフォームの違い

JBoss EAP 6 と、これまでのバージョンには顕著な違いがいくつかあります。

- セキュリティードメインは、管理対象ドメインの各プロファイル、またはスタンドアロンサーバーごとに設定されます。これらはデプロイメント自体の一部ではありません。デプロイメントが使用するセキュリティードメインの名前は、デプロイメントの jboss-web.xml ファイルまたは jboss-ejb3.xml ファイルにあります。
- セキュリティープロパティーはセキュリティードメインの一部として設定されます。これらはデプロイメントの一部ではありません。
- デプロイメントの一部としてオーセンティケーターを上書きしなくなりました。ただし、NegotiationAuthenticator バルブを jboss-web.xml 記述子に追加して同じ効果を得ることができます。バルブを使用するには、& lt;security-constraint > 要素および < login-config > 要素を web.xml に定義する必要があります。これらの要素は、セキュアなリソースを決定するのに使用されます。しかし、選択した auth-method は jboss-web.xml の NegotiationAuthenticator バルブによって上書きされます。
- セキュリティードメインの CODE 属性は、完全修飾クラス名の代わりに単純な名前を使用するようになりました。以下の表は、JBoss Negotiation に使用されるクラスとそれらのクラスとの間のマッピングを示しています。

表12.1 ログインモジュールコードおよびクラス名

簡単な名前	クラス名	目的
Kerberos	com.sun.security.auth.module.Krb5LoginModule com.ibm.security.auth.module.Krb5LoginModule	Oracle JDK を使用する場合の Kerberos ログインモジュール IBM JDK を使用する場合の Kerberos ログインモジュール
SPNEGO	org.jboss.security.negotiation.spnego.SPNEGOLoginModule	Web アプリケーションが Kerberos 認証サーバーに認証できるようにするメカニズム。
AdvancedLdap	org.jboss.security.negotiation.AdvancedLdapLoginModule	Microsoft Active Directory 以外の LDAP サーバーで使用されます。
AdvancedAdLdap	org.jboss.security.negotiation.AdvancedADLoginModule	Microsoft Active Directory LDAP サーバーで使用されます。

JBoss Negotiation Toolkit

JBoss Negotiation Toolkit は、から https://community.jboss.org/servlet/JiveServlet/download/16876-2-34629/jboss-negotiation-toolkit.war ダウンロードできるデバッグツールです。これは、アプリケーションを実稼働環境にデプロイする前に認証メカニズムのデバッグおよびテストに役立つ追加のツールとして提供されます。これはサポートされていないツールですが、SPNEGO は web アプリケーションに対して設定するのが困難な可能性があるため、非常に便利なツールと見なされます。

手順12.1 Web または EJB アプリケーションの SSO 認証の設定

1. **サーバーのアイデンティティーを表す 1 つのセキュリティードメインを設定します。必要に応じてシステムプロパティーを設定します。**

   最初のセキュリティードメインは、コンテナー自体をディレクトリーサービスに対して認証します。実際のユーザーは関与していないため、静的ログインメカニズムのタイプを受け入れるログインモジュールを使用する必要があります。この例では、静的プリンシパルを使用し、認証情報が含まれるキータブファイルを参照します。

   ここでは明確にするために XML コードが提供されますが、管理コンソールまたは管理 CLI を使用してセキュリティードメインを設定する必要があります。

```
<security-domain name="host" cache-type="default">
  <authentication>
    <login-module code="Kerberos" flag="required">
      <module-option name="storeKey" value="true"/>
      <module-option name="useKeyTab" value="true"/>
      <module-option name="principal" value="host/testserver@MY_REALM"/>
      <module-option name="keyTab" value="/home/username/service.keytab"/>
      <module-option name="doNotPrompt" value="true"/>
      <module-option name="debug" value="false"/>
```

```
      </login-module>
    </authentication>
  </security-domain>
```

2. **2 つ目のセキュリティードメインを設定して、Web アプリケーションまたはアプリケーションをセキュアにします。必要に応じてシステムプロパティーを設定します。**

   2 つ目のセキュリティードメインは、個別のユーザーを Kerberos または SPNEGO 認証サーバーに対して認証するために使用されます。ユーザーの認証には少なくとも 1 つのログインモジュールが必要です。もう 1 つは、ユーザーに適用するロールを検索することです。以下の XML コードは、SPNEGO セキュリティードメインの例になります。これには、ロールを個々のユーザーにマップする承認モジュールが含まれます。認証サーバー自体でロールを検索するモジュールを使用することもできます。

```
<security-domain name="SPNEGO" cache-type="default">
  <authentication>
    <!-- Check the username and password -->
    <login-module code="SPNEGO"  flag="requisite">
      <module-option name="password-stacking" value="useFirstPass"/>
      <module-option name="serverSecurityDomain" value="host"/>
    </login-module>
    <!-- Search for roles -->
    <login-module code="UsersRoles" flag="required">
      <module-option name="password-stacking" value="useFirstPass" />
      <module-option name="usersProperties" value="spnego-users.properties" />
      <module-option name="rolesProperties" value="spnego-roles.properties" />
    </login-module>
  </authentication>
</security-domain>
```

3. **web.xmlに security-constraint および login-config を指定します。**

   web.xml 記述子には、セキュリティー制約およびログイン設定に関する情報が含まれます。以下は、それぞれの値の例になります。

```
<security-constraint>
  <display-name>Security Constraint on Conversation</display-name>
  <web-resource-collection>
    <web-resource-name>examplesWebApp</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
  <role-name>RequiredRole</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>SPNEGO</auth-method>
  <realm-name>SPNEGO</realm-name>
</login-config>

<security-role>
```

```
    <description> role required to log in to the Application</description>
    <role-name>RequiredRole</role-name>
  </security-role>
```

4. **jboss-web.xml 記述子にセキュリティードメインとその他の設定を指定します。**

   デプロイメントの jboss-web.xml 記述子にクライアント側のセキュリティードメインの名前（この例では 2 つ目のセキュリティードメイン）を指定し、アプリケーションがこのセキュリティードメインを使用するように設定します。

   オーセンティケーターを直接上書きできなくなりました。代わりに、必要に応じて NegotiationAuthenticator を jboss-web.xml 記述子にバルブとして追加できます。&lt ;jacc-star-role-allow > を使用すると、アスタリスク(*)文字を使用して複数のロール名を一致させることができます。これはオプションです。

```
<jboss-web>
  <security-domain>SPNEGO</security-domain>
  <valve>
    <class-name>org.jboss.security.negotiation.NegotiationAuthenticator</class-name>
  </valve>
  <jacc-star-role-allow>true</jacc-star-role-allow>
</jboss-web>
```

5. **依存関係をアプリケーションの MANIFEST.MF に追加し、Negotiation クラスを見つけます。**

   JBoss Negotiation クラスを見つけるには、web アプリケーションは org.jboss.security.negotiation クラスの依存関係をデプロイメントの META-INF/MANIFEST.MF マニフェストに追加する必要があります。以下は、適切にフォーマットされたエントリーを示しています。

```
Manifest-Version: 1.0
Build-Jdk: 1.6.0_24
Dependencies: org.jboss.security.negotiation
```

   - **あるいは、META-INF/jboss-deployment-structure.xml ファイルを編集して、依存関係をアプリケーションに追加します。**

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss-deployment-structure>
  <deployment>
    <dependencies>
 <module name='org.jboss.security.negotiation'/>
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

結果

Web アプリケーションは、Kerberos、Microsoft Active Directory、またはその他の SPNEGO 互換ディレクトリーサービスに対して、認証情報を受け入れて認証します。ユーザーが、ディレクトリーサービスにすでにログインしているシステムからアプリケーションを実行し、必要なロールがすでにユーザーに適用される場合、web アプリケーションは認証を要求しず、SSO 機能が実行されます。

バグの報告

12.9. フォーム認証への SPNEGO FALL バックの設定

以下の手順に従って、SPNEGO フォールバックを設定して認証を形成します。

手順12.2 認証を形成するためにフォールバックした SPNEGO セキュリティー

1. SPNEGO の設定

   で説明されている手順を参照してください。 「Web アプリケーション用の Kerberos または Microsoft Active Directory Desktop SSO の設定」

2. Modify web.xml

   login-config 要素をアプリケーションに追加し、web.xml のログインページおよびエラーページを設定します。

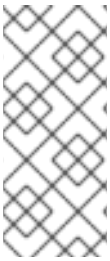
```
<login-config>
    <auth-method>SPNEGO</auth-method>
    <realm-name>SPNEGO</realm-name>
        <form-login-config>
            <form-login-page>/login.jsp</form-login-page>
            <form-error-page>/error.jsp</form-error-page>
        </form-login-config>
    </login-config>
```

3. Web コンテンツの追加

   login.html および error.html の参照を web.xml に追加します。これらのファイルは、web アプリケーションアーカイブに form-login-config 設定で指定された場所に追加されます。詳

細は、JBoss EAP 6 『『セキュリティーガイド』』 の「 『フォームベースの認証の有効化』」を参照してください。通常の login.html は以下のようになります。

```
<html>
   <head>
      <title>Vault Form Authentication</title>
   </head>
   <body>
      <h1>Vault Login Page</h1>
      <p>
      <form method="post" action="j_security_check">
      <table>
         <tr>
            <td>Username</td><td>-</td>
            <td><input type="text" name="j_username"></td>
         </tr>
         <tr>
            <td>Password</td><td>-</td>
            <td><input type="password" name="j_password"></td>
         </tr>
         <tr>
            <td colspan="2"><input type="submit"></td>
         </tr>
      </table>
      </form>
      </p>
      <hr>
   </body>
</html>
```



注記

FORM ロジックへのフォールバックは、SPNEGO（または NTLM）トークンが存在しない場合でのみ使用できます。その結果、ブラウザーが NTLM トークンを送信すると、ログインフォームがブラウザーに表示されません。

バグの報告

第13章 SAML を使用したシングルサインオン

13.1. セキュリティートークンサービス(STS)

Security Token Service は、セキュリティートークンを生成および管理します。特定のタイプのトークンを発行しません。代わりに、複数のトークンプロバイダーをプラグインできるようにする汎用インターフェースを定義します。そのため、各トークンタイプのトークンプロバイダーが存在する場合、さまざまなタイプのトークンに対応するよう設定することができます。また、セキュリティートークンのリクエストと応答の形式も指定します。

セキュリティートークン要求メッセージは、以下を指定します。

- Issue、Renew など、要求のタイプ。
- トークンのタイプ。
- 発行したトークンの有効期間。
- トークンを要求したサービスプロバイダーに関する情報。
- 生成されたトークンの暗号化に使用される情報。

備考

PKCS#11 トークンのサポートが、バージョン 6.3.0 から JBoss EAP に追加されました。

EAP セキュリティーレルムは、*provider* 属性を使用して PKCS#11 鍵とトラストストア定義を受け入れることができます。このパラメーターで指定した値は、関連する KeyStore.getInstance("PKCS11")呼び出しに 渡され、キーとトラストストアが初期化されます。

この新しいサポートの設定は、EAP ドキュメントの範囲外です。この機能を利用する場合は、PKCS#11 のハードウェアおよびソフトウェアの正しいインストールと、java.security ポリシーファイルに必要な正しいエントリーについて理解しておく必要があります。Oracle の 『Java PKCs#11 リファレンスガイド』 は、この情報に役立つリソースである場合があります。

トークン要求メッセージは、SOAP メッセージのボディーに送信されます。トークン要求に関連するすべての情報は、RequestSecurityToken 要素に囲まれています。サンプルリクエストには RequestType の他の 2 つの WS-Trust 要素が含まれます。この要素は、このリクエストが Issue リクエストであると指定し、発行するトークンのタイプを指定する TokenType です。

WS-Trust リクエストメッセージの例を以下に示します。

例13.1 WS-Trust security token request message

```
<S11:Envelope xmlns:S11=".." xmlns:wsu=".." xmlns:wst="..">
  <S11:Header>
    ...
  </S11:Header>
  <S11:Body wsu:Id="body">
    <wst:RequestSecurityToken Context="context">
      <wst:TokenType>http://www.tokens.org/SpecialToken</wst:TokenType>
      <wst:RequestType>
        http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue
      </wst:RequestType>
    </wst:RequestSecurityToken>
  </S11:Body>
</S11:Envelope>
```

以下は、セキュリティートークンの応答の例です。

例13.2 セキュリティートークンのレスポンスメッセージ

```
  <wst:RequestSecurityTokenResponse Context="context" xmlns:wst=".."
xmlns:wsu="..">
    <wst:TokenType>http://www.tokens.org/SpecialToken</wst:TokenType>
    <wst:RequestedSecurityToken>
      <token:SpecialToken xmlns:token="...">
        ARhjefhE2FEjneovi&@FHfeoveq3
      </token:SpecialToken>
    </wst:RequestedSecurityToken>
    <wst:Lifetime>
      <wsu:Created>...</wsu:Created>
      <wsu:Expires>...</wsu:Expires>
    </wst:Lifetime>
  </wst:RequestSecurityTokenResponse>
```

セキュリティートークンの応答の例では、TokenType 要素は発行されたトークンのタイプを指定しますが、RequestedSecurityToken 要素にはトークン自体が含まれます。トークンの形式は、トークンの種類によって異なります。Lifetime 要素は、トークンが作成されたタイミングおよび期限が切れるタイミングを指定します。

セキュリティートークンリクエストの処理

以下は、セキュリティートークンリクエストが処理される手順です。

- クライアントはセキュリティートークンリクエストを PicketLinkSTS に送信します。
- PicketLinkSTS 要求メッセージを解析し、JAXB オブジェクトモデルを生成します。
- PicketLinkSTS 設定ファイルを読み取り、必要に応じて STSConfiguration オブジェクトを作成します。設定から WSTrustRequestHandler への参照を取得し、リクエストの処理をハンドラーインスタンスに委譲します。

- リクエストハンドラーは、必要に応じて STSConfiguration を使用してデフォルト値を設定します（例： 要求がトークンの有効期間値を指定しない場合）。

- WSTrustRequestHandler は、WSTrustRequestContext リクエストオブジェクトと、JAXB から受け取った呼び出し元プリンシパルを設定します。PicketLinkSTS

- WSTrustRequestHandler は STSConfiguration を使用して SecurityTokenProvider を取得します。これは、要求されているトークンのタイプに基づいてリクエストを処理するために使用する必要があります。次にプロバイダーを呼び出し、構築された WSTrustRequestContext をパラメーターとして渡します。

- SecurityTokenProvider インスタンスはトークンリクエストを処理し、発行されたトークンをリクエストコンテキストに保存します。

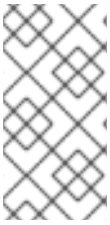
- WSTrustRequestHandler はコンテキストからトークンを取得し、必要に応じて暗号化し、セキュリティートークンが含まれる WS-Trust 応答オブジェクトを構築します。

- PicketLinkSTS リクエストハンドラーによって生成された応答を指示し、クライアントに返します。

バグの報告

13.2. セキュリティートークンサービス(STS)の設定

EAP Security Token Service(STS)は、拡張ポイントを提供する複数のインターフェースを定義します。設定を使用して実装をプラグインでき、設定を介してデフォルト値を一部のプロパティーに指定できます。すべての STS 設定は、デプロイされたアプリケーションの WEB-INF ディレクトリーにある picketlink.xml ファイルに指定されます。以下は、picketlink.xml ファイルで設定できる要素です。



注記

以下のテキストでは、*サービスプロバイダー* は、クライアントによるセキュリティートークンの表示を必要とする Web サービスを参照します。

- PicketLinkSTS: これはルート要素です。STS 管理者が以下のデフォルト値を設定できるようにするプロパティーを定義します。
  - STSName: セキュリティートークンサービスの名前を表す文字列。指定のない場合は、デフォルトの PicketLinkSTS 値が使用されます。
  - TokenTimeout: トークンの有効期間の値（秒単位）。指定されていない場合は、デフォルト値の 3600（1 時間）が使用されます。
  - EncryptToken: 発行されたトークンを暗号化するかどうかを指定するブール値。デフォルト値は false です。
- KeyProvider: この要素とそのすべてのサブ要素は、トークンに署名および暗号化するために PicketLink STS によって使用されるキーストアを設定するために使用されます。このセクションでは、キーストアの場所、パスワード、署名 (秘密鍵) のエイリアス、パスワードなどのプロパティーをすべて設定します。
- RequestHandler: この要素は、使用される WSTrustRequestHandler 実装の完全修飾名を指定します。指定のない場合は、デフォルトの org.picketlink.identity.federation.core.wstrust.StandardRequestHandler が使用されます。
- TokenProvider: このセクションは、各種類のセキュリティートークンを処理するために使用する必要がある TokenProvider 実装を指定します。この例では、2 つのプロバイダー（SpecialToken タイプのトークンと SAMLV2.0 タイプのトークンを処理するプロバイダー）があります。WSTrustRequestHandler は、getProviderForTokenTypeの STSConfiguration (String type)メソッドを呼び出して、適切な TokenProvider への参照を取得します。
-

TokenTimeout: これは、WS-Trust リクエストに lifetime が指定されていない場合に WSTrustRequestHandler によって使用されます。これは、作成時間として現在の時間が含まれる Lifetime インスタンスを作成し、指定の秒数後に有効期限が切れます。

- ServiceProviders: このセクションでは、各サービスプロバイダー（セキュリティートークンを必要とする Web サービス）に使用する必要があるトークンタイプを指定します。WS-Trust リクエストにトークンタイプが含まれていない場合、WSTrustRequestHandler はサービスプロバイダーエンドポイントを使用して発行する必要のあるトークンのタイプを検出する必要があります。

- EncryptToken: これは、発行されたトークンを暗号化する必要があるかどうかを判断するために WSTrustRequestHandler によって使用されます。true の場合、サービスプロバイダーの公開鍵証明書(PKC)を使用してトークンを暗号化します。

以下は STS 設定の例になります。

例13.3 STS 設定

```
<PicketLinkSTS xmlns="urn:picketlink:identity-federation:config:1.0"
        STSName="Test STS" TokenTimeout="7200" EncryptToken="true">
  <KeyProvider
ClassName="org.picketlink.identity.federation.bindings.tomcat.KeyStoreKeyManager">
    <Auth Key="KeyStoreURL" Value="keystore/sts_keystore.jks"/>
    <Auth Key="KeyStorePass" Value="testpass"/>
    <Auth Key="SigningKeyAlias" Value="sts"/>
    <Auth Key="SigningKeyPass" Value="keypass"/>
    <ValidatingAlias Key="http://services.testcorp.org/provider1" Value="service1"/>
    <ValidatingAlias Key="http://services.testcorp.org/provider2" Value="service2"/>
  </KeyProvider>
  <RequestHandler>org.picketlink.identity.federation.core.wstrust.StandardRequestHandler
</RequestHandler>
  <TokenProviders>
    <TokenProvider
ProviderClass="org.picketlink.test.identity.federation.bindings.wstrust.SpecialTokenProvid
er"
        TokenType="http://www.tokens.org/SpecialToken"/>
    <TokenProvider
ProviderClass="org.picketlink.identity.federation.api.wstrust.plugins.saml.SAML20TokenPr
ovider"
        TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV2.0"/>
  </TokenProviders>
  <ServiceProviders>
   <ServiceProvider Endpoint="http://services.testcorp.org/provider1"
TokenType="http://www.tokens.org/SpecialToken"
        TruststoreAlias="service1"/>
   <ServiceProvider Endpoint="http://services.testcorp.org/provider2"
TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV2.0"
        TruststoreAlias="service2"/>
```

```
</ServiceProviders>
</PicketLinkSTS>
```

バグの報告

13.3. PICKETLINK STS ログインモジュール

PicketLink ログインモジュールは通常、ユーザーの認証に Security Token Service を使用するように JEE コンテナーのセキュリティー設定の一部として設定されます。STS はログインモジュールと同じコンテナーに配置するか、または Web サービス呼び出しまたは他のテクノロジーを使用してリモートでアクセスすることができます。Picketlink ログインモジュールは、標準の WS-Trust 呼び出しを介して非 PicketLink STS 実装をサポートします。

STS ログインモジュールのタイプ

以下は、STS ログインモジュールのさまざまなタイプです。

STSIssuingLoginModule

- 設定された STS およびセキュリティートークンのリクエストを呼び出します。RequestedSecurityToken を正常に受信すると、認証が成功したとマークされます。
- STS への呼び出しには、通常認証が必要です。このログインモジュールは、以下のいずれかのソースからの認証情報を使用します。
  - useOptionsCredentials モジュールオプションが true に設定されている場合、そのプロパティーファイル。
  - password-stackingmodule オプションが useFirstPass に設定されている場合の以前のログインモジュールの認証情報
  - 名前とパスワードコールバックを指定して、設定した CallbackHandler から設定します。
- 認証に成功すると、SamlCredential は、同じ Assertion を持つ認証情報がまだ存在していない場合は、サブジェクトのパブリック認証情報に挿入されます。

STSValidatingLoginModule

- 設定済みの STS を呼び出して、利用可能なセキュリティートークンを検証します。
- STS への呼び出しには、通常認証が必要です。このログインモジュールは、以下のいずれかのソースからの認証情報を使用します。
  - useOptionsCredentials モジュールオプションが true に設定されている場合、そのプロパティーファイル。
  - password-stacking モジュールオプションが useFirstPass に設定されている場合の以前のログインモジュールの認証情報
  - 名前とパスワードコールバックを指定して、設定した CallbackHandler から設定します。
- 認証に成功すると、同じ Assertion を持つものがすでに存在していない場合には、SamlCredential が Subject のパブリック認証情報に挿入されます。

SAML2STSLoginModule

- このログインモジュールは、設定された ObjectCallback に CallbackHandler を提供し、SamlCredential オブジェクトを元に戻します。Assertion は、設定された STS に対して検証されます。
- ユーザー ID と SAML トークンが共有されている場合、このログインモジュールは、正常に認証される別のログインモジュールにスタックされた場合に検証をバイパスします。
- 認証に成功すると、SamlCredential は NameID と、それぞれユーザーの ID およびロールとして設定される多値ロール属性について検査されます。

SAML2LoginModule

- このログインモジュールは SAML 認証の他のコンポーネントとともに使用され、認証は実行されません。

- SPRedirectFormAuthenticator はこのログインモジュールを SAML v2 HTTP リダイレクトプロファイルの PicketLink の実装で使用します。

- Tomcat オーセンティケーターバルブは、アイデンティティープロバイダーへのリダイレクトと SAML アサーションの取得によって認証を実行します。

- このログインモジュールは、JAAS サブジェクトに入力されるようにユーザー ID とロールを JBoss セキュリティーフレームワークに渡すために使用されます。

バグの報告

13.4. STSISSUINGLOGINMODULE の設定

STSIssuingLoginModule はユーザー名とパスワードを使用して、トークンを取得して STS に対してユーザーを認証します。

例13.4 STSIssuingLoginModule の設定

```
<security-domain name="saml-issue-token">
    <authentication>
        <login-module
            code="org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLoginModule"
flag="required">          <module-option name="configFile">./picketlink-sts-
client.properties</module-option>
          <module-option name="endpointURI">http://security_saml/endpoint</module-
option>
        </login-module>
    </authentication>
    <mapping>
        <mapping-module

code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STSPrincipalMappin
gProvider"
            type="principal" />
        <mapping-module

code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STSGroupMappingPr
ovider"
            type="role" />
    </mapping>
</security-domain>
```

ほとんどの設定は、以下の例で示した設定に切り替えることができます。

- 宣言された security-domain の変更
- プリンシパルマッピングプロバイダーの指定
- RoleGroup マッピングプロバイダーの指定

指定の Principal マッピングプロバイダーと RoleGroup マッピングプロバイダーにより、認証サブジェクトが設定され、粒度の細かいロールベースの承認が可能になります。認証後、セキュリティートークンが利用可能になり、シングルサインオンにより他のサービスを呼び出すために使用できます。

バグの報告

13.5. CONFIGURE STSVALIDATINGLOGINMODULE

STSValidatingLoginModule は TokenCallback を使用して、トークンを取得して、設定した CallbackHandler に STS を要求します。

例13.5 Configure STSValidatingLoginModule

```
<security-domain name="saml-validate-token">
   <authentication>
      <login-module

code="org.picketlink.identity.federation.core.wstrust.auth.STSValidatingLoginModule"
flag="required">
         <module-option name="configFile">./picketlink-sts-client.properties</module-
option>
         <module-option name="endpointURI">http://security_saml/endpoint</module-
option>
      </login-module>
   </authentication>
   <mapping>
      <mapping-module

code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STSPrincipalMappin
gProvider"
         type="principal" />
      <mapping-module
```

```
code="org.picketlink.identity.federation.bindings.jboss.auth.mapping.STSGroupMappingPr
ovider"
            type="role" />
    </mapping>
</security-domain>
```

この例で引用された設定は、アプリケーションやサービスに対してシングルサインオンを有効にします。STS を直接接続するか、トークン発行ログインモジュールを介してトークンを発行したら、例で提供される設定を利用して複数のアプリケーションとサービスに対して認証できます。Principal マッピングプロバイダーと RoleGroup マッピングプロバイダーを指定すると、認証されたサブジェクトが設定され、粒度の細かいロールベースの承認が可能になります。認証後、セキュリティートークンが利用可能になり、シングルサインオンにより他のサービスを呼び出すために使用できます。

バグの報告

13.6. STS クライアントプール

PicketLink はサーバー上の STS クライアントのプールを提供します。これにより、STS クライアントの作成がボトルネックとして削除されます。

クライアントプーリングは、STS クライアントが SAML チケットの取得に必要なログインモジュールから利用できます。

STS クライアントプーリングを使用できるログインモジュール。

- org.picketlink.identity.federation.core.wstrust.auth.STSIssuingLoginModule
- org.picketlink.identity.federation.core.wstrust.auth.STSValidatingLoginModule
- org.picketlink.trust.jbossws.jaas.JBWSTokenIssuingLoginModule

各ログインモジュールのプール内のデフォルトのクライアント数は、initialNumberOfClients ログインモジュールオプションで設定されます。

STSClientPoolFactory クラス
org.picketlink.identity.federation.bindings.stspool.STSClientPoolFactory は、アプリケーションにクライアントプール機能を提供します。

STSClientPoolFactory の使用

STS クライアントは、その 設定 をキーとして使用してサブプールに挿入されます。STSClientPool インスタンスを取得してから、設定に基づいてサブプールを初期化します。任意で STS クライアントの初期数で、またはデフォルトの番号に依存します。

```
final STSClientPool pool = STSClientPoolFactory.getPoolInstance();
pool.createPool(20, stsClientConfig);
final STSClient client = pool.getClient(stsClientConfig);
```

クライアントを使用した作業が終了したら、以下のようにプールに戻すことができます。

```
pool.returnClient();
```

特定の設定についてサブプールがすでに存在しているかどうかを確認するには、次のコマンドを実行します。

```
if (! pool.configExists(stsClientConfig) {
    pool.createPool(stsClientConfig);
}
```

PicketLink Federation サブシステムを有効にすると、デプロイメント用に作成されたすべてのクライアントプールはアンデプロイプロセス時に自動的に破棄されます。プールを手動で破棄するには、以下を実行します。

```
pool.destroyPool(stsClientConfig);
```

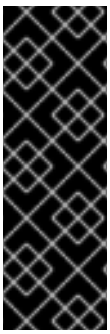
バグの報告

13.7. SAML WEB ブラウザーベースの SSO

13.7.1. SAML Web ブラウザーベースの SSO

JBoss EAP の picketlink は、フェデレーションされた ID ベースのサービスを実装するプラットフォームを提供します。これには、アプリケーションの ID サービスおよびシングルサインオン(SSO)が含まれます。

SAML プロファイルでは、アプリケーションの Web SSO を有効にするための集中型 ID サービスを含む HTTP/POST と HTTP/Redirect バインディングの両方がサポートされます。SAML v2 ベースの Web SSO のアーキテクチャーは、ID 管理のハブおよびスポークアーキテクチャーに従います。このアーキテクチャーでは、アイデンティティープロバイダー(IDP)は、すべてのアプリケーション（サービスプロバイダー）に対するアイデンティティーおよびロール情報の中央ソース（ハブ）として機能します。スポークはサービスプロバイダー(SP)です。



重要

両方が同じ IDP をポイントする SP が 2 つ以上ある場合、IDP は異なる SP を区別しません。同じ IDP を参照する別の SP にリクエストを行う場合、IDP は SP からの最新のリクエストを処理し、認証されたユーザーに関する SAML アサーションを送信します。古い SP リクエストに戻るには、ブラウザーで SP URL を再入力する必要があります。

バグの報告

13.7.2. SAML v2 ベースの Web SSO の設定

SAML v2 ベースの SSO を設定するには、以下を設定する必要があります。

- アイデンティティープロバイダー： ID プロバイダーはエンドユーザーを認証し、そのユーザーのアイデンティティーを信頼できる方法で信頼できるパートナーにアサートする信頼できるエンティティーです。
- サービスプロバイダー： サービスプロバイダーは、アイデンティティープロバイダーに依存してユーザーに関する情報を電子ユーザーの認証情報経由でアサートし、信頼できるユーザークレデンシャルのアサーションセットに基づいてアクセス制御と配信を管理します。

バグの報告

13.7.3. ID プロバイダーの設定

アイデンティティープロバイダー(IDP)は JBoss EAP サーバーインスタンスです。

手順13.1 アイデンティティープロバイダー(IDP)の設定

1. IDP の Web アプリケーションセキュリティーの設定

Web アプリケーションを ID プロバイダーとして設定します。

注記

FORM ベースの Web アプリケーションセキュリティーの使用が推奨されます。ログインページをカスタマイズすることができます。

web.xml 設定の例を以下に示します。

例13.6 IDP の web.xml 設定

```
<display-name>IDP</display-name>
<description>IDP</description>
<!-- Define a security constraint that gives unlimited access to images -->
<security-constraint>
  <web-resource-collection>
  <web-resource-name>Images</web-resource-name>
  <url-pattern>/images/*</url-pattern>
</web-resource-collection>
</security-constraint>
<!-- Define a Security Constraint on this Application -->
<security-constraint>
  <web-resource-collection>
  <web-resource-name>IDP</web-resource-name>
  <url-pattern>/*</url-pattern>
</web-resource-collection>
  <auth-constraint>
  <role-name>manager</role-name>
</auth-constraint>
</security-constraint>
<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>IDP Application</realm-name>
  <form-login-config>
    <form-login-page>/jsp/login.jsp</form-login-page>
    <form-error-page>/jsp/loginerror.jsp</form-error-page>
  </form-login-config>
</login-config>
<!-- Security roles referenced by this web application -->
<security-role>
 <description>
  The role that is required to log in to the IDP Application
 </description>
 <role-name>manager</role-name>
</security-role>
</web-app>
```

2. **IDP のセキュリティードメインの作成**

   IDP 向けに定義された認証および承認メカニズムを使用してセキュリティードメインを作成します。詳細は、「アプリケーションでのセキュリティードメインの使用」 を参照してください。

3. **IDP バルブの設定**

   IDP Web アプリケーションの WEB-INF ディレクトリーに jboss-web.xml ファイルを作成し、IDP のバルブを設定します。以下は jboss-web.xml ファイルの例になります。

例13.7 IDP バルブの jboss-web.xml ファイル設定

```
<jboss-web>
  <security-domain>idp</security-domain>
  <context-root>idp</context-root>
  <valve>
    <class-
name>org.picketlink.identity.federation.bindings.tomcat.idp.IDPWebBrowserSSOVa
lve</class-name>
  </valve>
</jboss-web>
```

4. **PicketLink 設定ファイル(picketlink.xml)の設定**

   以下は、picketlink.xml の設定例です。この設定ファイルで、送信 SAML2 アサーションの発行者としてサービスプロバイダーと IDP に発行者として追加される URL を提供します。

例13.8 picketlink.xml Configuration

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  <PicketLinkIDP xmlns="urn:picketlink:identity-federation:config:2.1">
    <IdentityURL>http://localhost:8080/idp/</IdentityURL>
  </PicketLinkIDP>
  <Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2IssuerTrustHan
dler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler
" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Authentication
Handler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHand
ler" />
  </Handlers>
</PicketLink>
```

デフォルトでは、picketlink.xml は IDP Web アプリケーションの WEB-INF ディレクトリーにあります。ただし、アプリケーションの外部にある picketlink.xml へのカスタムパスを設定できます。

a. オプション： カスタムパスを picketlink.xmlに設定する設定

アプリケーションの WEB-INF/jboss-web.xml の valve 要素に 2 つのパラメーターを追加します。configFile は、picketlink.xml へのパスに指定し、設定を再読み込みする間隔をミリ秒単位で指定する timerInterval です。以下に例を示します。

```
<valve>
  <class-name>...</class-name>
    <param>
      <param-name>timerInterval</param-name>
      <param-value>5000</param-value>
    </param>
    <param>
      <param-name>configFile</param-name>
      <param-value>path-to/picketlink.xml</param-value>
    </param>
</valve>
```

5. PicketLink モジュールの依存関係を宣言します（META-INF/MANIFEST.MF または jboss-deployment-structure.xml）。

Web アプリケーションには、PicketLink クラスを配置するために META-INF/MANIFEST.MF または jboss-deployment-structure.xml を定義する依存関係も必要です。

例13.9 META-INF/MANIFEST.MFへの依存関係の定義

```
Manifest-Version: 1.0
    Build-Jdk: 1.6.0_24
    Dependencies: org.picketlink
```

例13.10 META-INF/jboss-deployment-structure.xmlで依存関係を定義

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.picketlink" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

バグの報告

13.7.4. HTTP/REDIRECT バインディングを使用したサービスプロバイダーの設定

サービスプロバイダー(SP)は JBoss EAP サーバーインスタンスになります。

手順13.2 サービスプロバイダー(SP)の設定

1. **SP の Web アプリケーションセキュリティーの設定**

   SP として設定される web アプリケーションは、web.xml ファイルで FORM ベースのセキュリティーを有効にする必要があります。

例13.11 SP の web.xml 設定

```
<display-name>SP</display-name>
<description>SP</description>
<!-- Define a security constraint that gives unlimited access to images -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Images</web-resource-name>
    <url-pattern>/images/*</url-pattern>
  </web-resource-collection>
</security-constraint>
<!-- Define a Security Constraint on this Application -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>SP</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>
<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>SP Application</realm-name>
  <form-login-config>
    <form-login-page>/jsp/login.jsp</form-login-page>
    <form-error-page>/jsp/loginerror.jsp</form-error-page>
  </form-login-config>
</login-config>
<!-- Security roles referenced by this web application -->
<security-role>
    <description>
   The role that is required to log in to the SP Application
  </description>
```

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

2. **SP のセキュリティードメインの作成**

   SAML2LoginModule を使用するセキュリティードメインを作成します。以下は設定例です。

```
<security-domain name="sp" cache-type="default">
  <authentication>
    <login-module
code="org.picketlink.identity.federation.bindings.jboss.auth.SAML2LoginModule"
flag="required"/>
   </authentication>
</security-domain>
```

3. **SP バルブの設定**

   SP にバルブを設定するには、SP web アプリケーションの WEB-INF ディレクトリーに jboss-web.xml を作成します。

例13.12 SP バルブの jboss-web.xml ファイル設定

```
<jboss-web>
  <security-domain>sp</security-domain>
  <context-root>sales-post</context-root>
  <valve>
    <class-
name>org.picketlink.identity.federation.bindings.tomcat.sp.ServiceProviderAuthent
icator</class-name>
  </valve>
</jboss-web>
```

4. **PicketLink 設定ファイル(picketlink.xml)の設定**

   以下は、SP の picketlink.xml の設定例です。この設定ファイルでは、SP の URL および IDP の URL を SP に対応するハンドラーとともに提供します。

例13.13 picketlink.xml Configuration

```
<PicketLink xmlns="urn:picketlink:identity-federation:config:2.1">
  <PicketLinkSP xmlns="urn:picketlink:identity-federation:config:2.1"
ServerEnvironment="tomcat" BindingType="REDIRECT">
    <IdentityURL>${idp.url::http://localhost:8080/idp/}</IdentityURL>
    <ServiceURL>${sales-post.url::http://localhost:8080/sales-post/}</ServiceURL>
  </PicketLinkSP>
  <Handlers xmlns="urn:picketlink:identity-federation:handler:config:2.1">
```

```
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2LogOutHandler
" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.SAML2Authentication
Handler" />
    <Handler
class="org.picketlink.identity.federation.web.handlers.saml2.RolesGenerationHand
ler" />
  </Handlers>
</PicketLink>
```

デフォルトでは、picketlink.xml はアプリケーションの WEB-INF ディレクトリーにあります。ただし、アプリケーションの外部にある picketlink.xml へのカスタムパスを設定できます。

a. オプション： カスタムパスを picketlink.xmlに設定する設定

アプリケーションの WEB-INF/jboss-web.xml の valve 要素に 2 つのパラメーターを追加します。configFile は、picketlink.xml へのパスに指定し、設定を再読み込みする間隔をミリ秒単位で指定する timerInterval です。以下に例を示します。

```
<valve>
  <class-name>...</class-name>
    <param>
      <param-name>timerInterval</param-name>
      <param-value>5000</param-value>
    </param>
    <param>
      <param-name>configFile</param-name>
      <param-value>path-to/picketlink.xml</param-value>
    </param>
</valve>
```

5. PicketLink モジュールの依存関係を宣言します（META-INF/MANIFEST.MF または jboss-deployment-structure.xml）。

Web アプリケーションには、PicketLink クラスを配置するために META-INF/MANIFEST.MF または jboss-deployment-structure.xml を定義する依存関係も必要です。

例13.14 META-INF/MANIFEST.MFへの依存関係の定義

```
Manifest-Version: 1.0
    Build-Jdk: 1.6.0_24
    Dependencies: org.picketlink
```

例13.15 META-INF/jboss-deployment-structure.xmlで依存関係を定義

```
<jboss-deployment-structure>
  <deployment>
    <dependencies>
      <module name="org.picketlink" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

バグの報告

13.7.5. HTTP/POST バインディングを使用した SAML v2 ベースの Web SSO の設定

web ブラウザーベースの SSO を取得するために、HTTP/POST バインディングが推奨されます。

手順13.3 HTTP/POST バインディングを使用した SAML v2 ベースの Web SSO の設定

1. アイデンティティープロバイダー(IDP)を設定します。

   HTTP/POST バインディングの IDP を設定する手順は、HTTP/Redirect Binding と同じです。IDP の設定に関する詳細は、を参照してください。 「SAML v2 ベースの Web SSO の設定」

2. サービスプロバイダー(SP)の設定

   HTTP/POST バインディングの SP を設定する手順は、SP の picketlink.xml ファイルの 1 つのバリエーションを除き、HTTP/Redirect バインディングと同じです。BindingType="REDIRECT" を BindingType="POST" に変更します。

   SP の設定に関する詳細は、を参照してください。 「HTTP/REDIRECT バインディングを使用したサービスプロバイダーの設定」

バグの報告

13.7.6. サービスプロバイダーでの動的アカウント選択の設定

要件:

- 「ID プロバイダーの設定」

- 「HTTP/REDIRECT バインディングを使用したサービスプロバイダーの設定」

サービスプロバイダー(SP)が複数のアイデンティティープロバイダー(IDP)で設定されている場合、PicketLink を設定して、認証情報の認証に使用する IDP を選択するように指示できます。

手順13.4 サービスプロバイダーでの動的アカウント選択の設定

1. SP Web アプリケーションの WEB-INF ディレクトリーで jboss-web.xml でアカウント選択可能なバルブを設定します。

例13.16 SP アカウント選択の jboss-web.xml ファイル設定

```
<jboss-web>
  <security-domain>sp</security-domain>
  <context-root>accountchooser</context-root>
  <valve>
    <class-
name>org.picketlink.identity.federation.bindings.tomcat.sp.AccountChooserValve<
/class-name>
  </valve>
  <valve>
    <class-
name>org.picketlink.identity.federation.bindings.tomcat.sp.ServiceProviderAuthent
icator</class-name>
  </valve>
</jboss-web>
```

AccountChooserValve 設定可能なオプションは以下のとおりです。

DomainName

ユーザーのブラウザーに送信されるクッキーに使用されるドメイン名。

CookieExpiry

Cookie の有効期限 (秒単位)。デフォルトは -1です.これは、ブラウザーが閉じられると Cookie が期限切れすることを意味します。

AccountIDPMapProvider

IDP マッピングの実装の完全修飾名。デフォルトは、SP Web アプリケーションの WEB-INF ディレクトリーにあるプロパティーファイル idpmap.properties です。この実装は

org.picketlink.identity.federation.bindings.tomcat.sp.AbstractAccountChooserValve.AccountIDPMapProvider を実装する必要があります。

AccountChooserPage

異なる IDP アカウントを一覧表示する HTML/JSP ページの名前。デフォルトは /accountChooser.html です。

2. IDP のマッピングを定義します。デフォルトでは、これは SP Web アプリケーションの WEB-INF ディレクトリーにあるプロパティーファイル idpmap.properties です。

例13.17 idpmap.properties 設定

```
DomainA=http://localhost:8080/idp1/
DomainB=http://localhost:8080/idp2/
```

3. ユーザーが IDP を選択するための SP web アプリケーションに HTML ページを作成します。デフォルトでは、このファイルは accountChooser.html です。各 IDP の URL には、 idpmap.properties に一覧表示される IDP の名前を指定する idp パラメーターが必要です。

例13.18 accountChooser.html Configuration

```
<html>
  ...
  <a href="?idp=DomainA">DomainA</a>
  <hr/>
  <a href="?idp=DomainB">DomainB</a>
  ...
</html>
```

バグの報告

13.7.7. IDP によって開始される SSO の設定

要件:

- 「ID プロバイダーの設定」

- 「HTTP/REDIRECT バインディングを使用したサービスプロバイダーの設定」

通常 PicketLink では、SP は認証リクエストを IDP に送信してフローを開始します。これにより、有効なアサーションを持つ SP に SAML 応答が送信されます。このフローは SP 開始 SSO と呼ばれます。しかし、SAML 2.0 仕様は IDP 開始または未要求応答 SSO と呼ばれる別のフローも定義します。このシナリオでは、SP は認証フローを開始しず、IDP から SAML 応答を受け取ります。フローは IDP 側で開始し、認証されると、ユーザーはリストから特定の SP を選択し、その URL にリダイレクトされます。

手順

1. ユーザーが IDP にアクセスします。
2. IDP は SAML リクエストおよび応答がないことを確認し、SAML を使用する IDP の最初のシナリオを想定します。
3. IDP はユーザーの認証を行います。
4. 認証後、IDP は、すべての SP アプリケーションにリンクするページをユーザーが取得する、ホストされたセクションを表示します。
5. ユーザーは SP アプリケーションを選択します。
6. IDP は、クエリーパラメーターの SAML 応答に SAML アサーションを使用して、ユーザーをサービスプロバイダーにリダイレクトします。
7. SP は SAML アサーションをチェックし、アクセスを提供します。

設定

未要求のレスポンスを取得するためには特別な設定は必要ありません。通常通り IDP および SP を設定できます。IDP および SP の設定方法に関する詳細は、以下を参照してください。

- 「ID プロバイダーの設定」

- 「HTTP/REDIRECT バインディングを使用したサービスプロバイダーの設定」

使用方法

ユーザーが認証されると、IDP には、すべてのサービスプロバイダーアプリケーションへのリンクを持つページが表示されます。通常、リンクは以下のようになります。

```
<a href="http://localhost:8080/idp?SAML_VERSION=2.0&TARGET=http://localhost:8080/sales-post/">Sales</a>
```

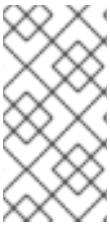
上記のリンクは、TARGET クエリーパラメーターを渡す IDP にユーザーをリダイレクトし、値はターゲット SP アプリケーションへ URL になります。ユーザーが上記のリンクをクリックすると、IDP はリクエストから TARGET パラメーターを抽出し、SAML v2.0 応答をビルドし、ユーザーをターゲット URL にリダイレクトします。ユーザーが SP に到達すると、自動的に認証されます。

SAML_VERSION クエリーパラメーターを使用して、IDP が SAML 応答を作成するのに必要な SAML バージョンを指定できます。SAML_VERSION パラメーターには 2.0 および 1.1 のオプションを使用できます。

バグの報告

13.8. SAML グローバルログアウトプロファイルの設定

1 つのサービスプロバイダーで開始されるグローバルログアウトは、アイデンティティープロバイダー(IDP)およびサービスプロバイダーからユーザーをログアウトします。



注記

グローバルログアウトが適切に機能するには、アイデンティティープロバイダーごとに最大 5 つのサービスプロバイダーのみがあることを確認してください。

手順13.5 グローバルログアウトの設定

1. picketlink-handlers.xml の設定

   picketlink-handlers.xml に SAML2LogOutHandler を追加します。

2. **サービスプロバイダーの Web ページの設定**

   サービスプロバイダーの Web ページの最後に GLO=true をリンクに追加します。

例13.19 グローバルログアウトへのリンク

```
<a href="?GLO=true">Click to Globally LogOut</a>
```

3. **logout.jsp ページを作成します。**

   ログアウトプロセスの一環として、PicketLink はユーザーを Service Provider アプリケーションのルートディレクトリーにある logout.jsp ページにリダイレクトします。このページが作成されていることを確認します。

バグの報告

第14章 ログインモジュール

バグの報告

14.1. モジュールの使用

JBoss EAP 6 には、ほとんどのユーザー管理のニーズに適したバンドルされたログインモジュールが複数含まれています。JBoss EAP 6 は、リレーショナルデータベース、LDAP サーバー、またはフラットファイルからユーザー情報を読み取ることができます。JBoss EAP 6 は、これらのコアログインモジュールの他に、非常にカスタマイズされたニーズに合わせてユーザー情報を提供するログインモジュールを提供します。

ログインモジュールとそのオプションの詳細は、「付録 A.1」を参照してください。

バグの報告

14.1.1. パスワードスタッキング

スタックでは複数のログインモジュールをチェーンでき、各ログインモジュールは認証中にクレデンシャルの検証とロールの割り当ての両方を提供します。これは多くのユースケースで機能しますが、クレデンシャルの検証とロールの割り当てが複数のユーザー管理ストアに分散されることがあります。

「Ldap ログインモジュール」 では、LDAP とリレーショナルデータベースを組み合わせて、ユーザーがいずれかのシステムで認証できるようにする方法を説明します。ユーザーは中央の LDAP サーバーで管理されますが、アプリケーション固有のロールはアプリケーションのリレーショナルデータベースに格納される場合を考えてみましょう。password-stacking モジュールオプションはこの関係をキャプチャーします。

パスワードスタッキングを使用するには、各ログインモジュールで <module-option> password-stacking 属性を useFirstPass に設定する必要があります。パスワードスタッキングに設定した以前のモジュールがユーザーを認証した場合、他のすべてのスタッキングモジュールがユーザーによって認証されたこととなり、承認の手順でロールの提供のみを行います。

password-stacking オプションを useFirstPass に設定すると、このモジュールは最初に、ログインモジュール共有状態マップで、プロパティー名 javax.security.auth.login.name と

javax.security.auth.login.password で共有ユーザー名とパスワードを検索します。

これらのプロパティーが見つかった場合、プリンシパル名とパスワードとして使用されます。見つからなかった場合、プリンシパル名とパスワードはこのログインモジュールによって設定され、プロパティー名 javax.security.auth.login.name と javax.security.auth.login.password の下にそれぞれ保存されます。

注記

パスワードスタッキングを使用する場合は、すべてのモジュールが必要になるように設定します。これにより、すべてのモジュールが考慮され、承認プロセスにロールを公開することができるようになります。

例14.1 パスワードスタッキングの例

この管理 CLI の例は、パスワードスタッキングがどのように使用されるかを示しています。

```
/subsystem=security/security-domain=pwdStack/authentication=classic/login-
module=Ldap:add( \
  code=Ldap, \
  flag=required, \
  module-options=[ \
    ("password-stacking"=>"useFirstPass"), \
    ... Ldap login module configuration
  ])
/subsystem=security/security-domain=pwdStack/authentication=classic/login-
module=Database:add( \
  code=Database, \
  flag=required, \
  module-options=[ \
    ("password-stacking"=>"useFirstPass"), \
    ... Database login module configuration
  ])
```

バグの報告

14.1.2. パスワードのハッシュ化

ログインモジュールのほとんどは、クライアントが提供するパスワードをユーザー管理システムに保存されたパスワードと比較する必要があります。通常、これらのモジュールはプレーンテキストのパスワードを使用しますが、プレーンテキストのパスワードがサーバー側に保存されないようにするため、ハッシュ化されたパスワードをサポートするよう設定できます。

重要

Red Hat JBoss Enterprise Application Platform Common criteria certified リリースは、パスワードのハッシュ化に SHA-256 のみをサポートしています。

例14.2 パスワードのハッシュ化

以下は、認証されていないユーザーにプリンシパル名 nobody を割り当て、usersb64.properties ファイルのパスワードの base64 でエンコードされた SHA-256 ハッシュが含まれるログインモジュール設定です。usersb64.properties ファイルはデプロイメントクラスパスの一部です。

```
/subsystem=security/security-domain=testUsersRoles:add
/subsystem=security/security-domain=testUsersRoles/authentication=classic:add
/subsystem=security/security-domain=testUsersRoles/authentication=classic/login-
module=UsersRoles:add( \
  code=UsersRoles, \
  flag=required, \
  module-options=[ \
    ("usersProperties"=>"usersb64.properties"), \
    ("rolesProperties"=>"test-users-roles.properties"), \
    ("unauthenticatedIdentity"=>"nobody"), \
    ("hashAlgorithm"=>"SHA-256"), \
    ("hashEncoding"=>"base64") \
  ])
```

hashAlgorithm

パスワードのハッシュ化に使用する java.security.MessageDigest アルゴリズムの名前。デフォルトがないため、ハッシュを有効にするには、このオプションを指定する必要があります。一般的な値は SHA-256、SHA-1、および MD5 です。

hashEncoding

base64、16 進数、または rfc2617 の 3 つのエンコーディングタイプのいずれかを指定する文字列。デフォルトは base64 です。

hashCharset

クリアテキストのパスワードをバイトアレイに変換するために使用されるエンコーディング文字セット。プラットフォームのデフォルトエンコーディングはデフォルトです。

hashUserPassword

ユーザーが送信するパスワードに適用する必要があるハッシュアルゴリズムを指定します。ハッシュ化されたユーザーパスワードは、ログインモジュール内の値と比較されます. これは、パスワードのハッシュです。デフォルトは true です。

hashStorePassword

サーバー側に保存されているパスワードに適用する必要があるハッシュアルゴリズムを指定します。これはダイジェスト認証に使用され、ユーザーが、比較するサーバーからの要求固有のトークンとともにユーザーパスワードのハッシュを送信します。ハッシュアルゴリズム（ダイジェストでは rfc2617）はサーバー側のハッシュを計算するために使用されます。これは、クライアントから送信されるハッシュ化された値と一致している必要があります。

コードでパスワードを生成する必要がある場合は、org.jboss.security.auth.spi.Util クラスは、指定されたエンコーディングを使用してパスワードをハッシュ化する静的ヘルパーメソッドを提供します。以下の例では、base64 でエンコードされた MD5 ハッシュされたパスワードを生成します。

```
String hashedPassword = Util.createPasswordHash("SHA-256",
 Util.BASE64_ENCODING, null, null, "password");
```

OpenSSL は、コマンドラインでハッシュ化されたパスワードを迅速に生成する代替方法を提供します。以下の例では、base64 でエンコードされた SHA-256 ハッシュされたパスワードも生成します。ここでは、プレーンテキストのパスワード - password - は OpenSSL ダイジェスト関数にパイプされ、base64 でエンコードされた形式に変換するために別の OpenSSL 関数にパイプされます。

```
echo -n password | openssl dgst -sha256 -binary | openssl base64
```

いずれの場合も、ハッシュ化されたバージョンのパスワードは同じです：XohImNooBHFR0OVvjcYpJ3NgPQ1qq73WKhHvch0VQtg=.この値は、上記の例のセキュリティードメイン(user b64.properties)に指定されたユーザーのプロパティー ファイルに保存する必要があります。

バグの報告

14.1.3. 認証されていない ID

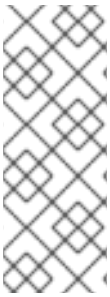
すべての要求が認証形式で受信される訳ではありません。unauthenticatedIdentity は、特定の ID（guest など）を、関連する認証情報を持たない要求に割り当てるログインモジュール設定オプションです。これを使用すると、保護されていないサーブレットは特定ロールを必要としない EJB でメソッドを呼び出すことができます。このようなプリンシパルには関連したロールがなく、セキュアでない EJB や、チェックされていないパーミッション制約と関連する EJB メソッドのみにアクセスできます。

- unauthenticatedIdentity: これは、認証情報のない要求に割り当てる必要のあるプリンシパル名を定義します。

バグの報告

14.1.4. Ldap ログインモジュール

LDAP ログインモジュールは、LDAP(Lightweight Directory Access Protocol)サーバーに対して認証を行う LoginModule 実装です。ユーザー名および認証情報が JNDI(Java Naming and Directory Interface)LDAP プロバイダーを使用してアクセス可能な LDAP サーバーに保存されている場合は、Ldap ログインモジュールを使用します。

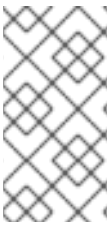


「ADVANCEDLDAP ログインモジュール」

SPNEGO 認証で LDAP を使用する場合や、LDAP サーバーの使用時に認証フェーズの一部を省略する場合は、SPNEGO ログインモジュールとチェーンされた AdvancedLdap ログインモジュールを使用することを検討するか、AdvancedLdap ログインモジュールのみを使用することを検討してください。

識別名(DN)

Lightweight Directory Access Protocol(LDAP)では、識別名はディレクトリー内のオブジェクトを一意に識別します。各識別名には、他のオブジェクトと区別するための一意名と場所が必要で、これには属性と値のペア (AVP) を使用します。AVP は、共通名、組織単位などの情報を定義します。LDAP に必要となる一意な文字列は、これらの値の組み合わせになります。



注記

このログインモジュールは、認証されていない ID およびパスワードスタックもサポートします。

LDAP 接続情報は、JNDI 初期コンテキストの作成に使用される環境オブジェクトに渡される設定オプションとして提供されます。使用される標準 LDAP JNDI プロパティーには以下が含まれます。

java.naming.factory.initial

InitialContextFactory 実装クラス名。デフォルトは Sun LDAP プロバイダー実装 com.sun.jndi.ldap.LdapCtxFactory です。

java.naming.provider.url

LDAP サーバーの LDAP URL。

java.naming.security.authentication

使用するセキュリティープロトコルレベル。使用できる値には none、simple、および strong が含まれます。プロパティーが定義されていない場合、動作はサービスプロバイダーによって決定されます。

java.naming.security.protocol

セキュアなアクセスに使用するトランスポートプロトコル。この設定オプションをサービスプロバイダーのタイプ（例： SSL）に設定します。プロパティーが定義されていない場合、動作はサービスプロバイダーによって決定されます。

java.naming.security.principal

サービスに対する呼び出し元を認証する Principal のアイデンティティーを指定します。これは、以下で説明するように他のプロパティーから構築されます。

java.naming.security.credentials

サービスに対して呼び出し元を認証する Principal の認証情報を指定します。認証情報は、ハッシュ化されたパスワード、クリアテキストパスワード、キー、または証明書の形式を取ることができます。プロパティーが定義されていない場合、動作はサービスプロバイダーによって決定されます。

Ldap ログインモジュール設定オプションの詳細は、「含まれる認証モジュール」 を参照してください。

注記

特定のディレクトリースキーマ（Microsoft Active Directory など）では、ユーザーオブジェクトのロール属性は単純な名前ではなく、ロールオブジェクトへの DN として保存されます。このスキーマタイプを使用する実装の場合は、roleAttributeIsDN を true に設定する必要があります。

ユーザー認証は、ログインモジュール設定オプションに基づいて LDAP サーバーに接続することで実行されます。LDAP サーバーへの接続は、このセクションで説明した LDAP JNDI プロパティーで構

成される環境を使用して InitialLdapContext を作成して行います。

Context.SECURITY_PRINCIPAL は、コールバックハンドラーが取得したユーザーの識別名に、principalDNPrefix および principalDNSuffix オプションの値と組み合わせて設定され、Context.SECURITY_CREDENTIALS プロパティーはそれぞれの String パスワードに設定されます。

認証が成功したら(initialLdapContext instance)、検索属性を roleAttributeName および uidAttributeName に設定して rolesCtxDN の場所で検索を実行すると、ユーザーのロールがクエリーされます。ロール名は、検索結果セットでロール属性の toString メソッドを呼び出すことで取得します。

例14.3 LDAP ログインモジュールのセキュリティードメイン

以下の管理 CLI の例は、セキュリティードメイン認証設定でパラメーターを使用する方法を示しています。

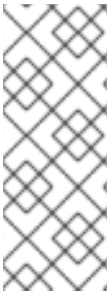
```
/subsystem=security/security-domain=testLDAP:add(cache-type=default)
/subsystem=security/security-domain=testLDAP/authentication=classic:add
/subsystem=security/security-domain=testLDAP/authentication=classic/login-
module=Ldap:add( \
  code=Ldap, \
  flag=required, \
  module-options=[ \
    ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"), \
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org:1389/"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("principalDNPrefix"=>"uid="), \
    ("principalDNSuffix"=>",ou=People,dc=jboss,dc=org"), \
    ("rolesCtxDN"=>"ou=Roles,dc=jboss,dc=org"), \
    ("uidAttributeID"=>"member"), \
    ("matchOnUserDN"=>true), \
    ("roleAttributeID"=>"cn"), \
    ("roleAttributeIsDN"=>false) \
  ])
```

testLDAP セキュリティードメイン設定の *java.naming.factory.initial* オプション、*java.naming.factory.url* オプション、および *java.naming.security* オプションは、以下の条件を示しています。

- Sun LDAP JNDI プロバイダー実装が使用されます。

- LDAP サーバーは、ポート 1389 のホスト ldaphost.jboss.org にあります。
- LDAP 簡易認証方法は、LDAP サーバーへの接続に使用されます。

ログインモジュールは、認証しようとしているユーザーを表す識別名(DN)を使用して LDAP サーバーへの接続を試みます。この DN は、上記のように、渡された principalDNPrefix、ユーザーのユーザー名、および principalDNSuffix から作成されます。例14.4「LDIF ファイルの例」 では、ユーザー名 jsmith は uid=jsmith,ou=People,dc=jboss,dc=org にマップされました。



注記

この例では、LDAP サーバーは、ユーザーのエントリーの userPassword 属性を使用してユーザーを認証します（この例ではtheduke ）。ほとんどの LDAP サーバーはこの方法で動作しますが、LDAP サーバーが認証ごとに異なる場合は、LDAP が実稼働環境の要件に応じて設定する必要があります。

認証に成功すると、uidAttributeID がユーザーに一致するエントリーに対して rolesCtxDN のサブツリー検索を実行して、承認に基づくロールを取得します。matchOnUserDN が true の場合、検索はユーザーの完全な DN を基にします。それ以外の場合は、入力した実際のユーザー名をもとに検索が行われます。この例では、uid=jsmith,ou=People,dc=jboss,dc=org と同等の メンバー 属性を持つエントリーの検索は ou=Roles,dc=jboss,dc=org の下にあります。検索は、roles エントリーの下に cn=JBossAdmin を見つけます。

検索は、roleAttributeID オプションで指定した属性を返します。この例では、属性は cn です。返される値は JBossAdmin であるため、jsmith ユーザーは JBossAdmin ロールに割り当てられます。

ローカル LDAP サーバーは多くの場合、アイデンティティーおよび認証サービスを提供しますが、認証サービスを使用することはできません。これは、アプリケーションロールが常に LDAP グループにマッピングされるわけではないので、LDAP 管理者は多くの場合、中央の LDAP サーバーで外部アプリケーション固有のデータを許可する傾向があります。LDAP 認証モジュールは、多くの場合、データベースログインモジュールなどの別のログインモジュールとペアになり、開発するアプリケーションによりより適したロールを提供できます。

このデータが動作するディレクトリーの構造を表す LDAP データ交換形式(LDIF)ファイルは 例14.4「LDIF ファイルの例」 に表示されます。

LDAP データ交換形式(LDIF)

LDAP ディレクトリーの内容を表し、リクエストを更新するために使用されるプレーンテキストデータ交換形式。ディレクトリーの内容は、各オブジェクトまたは更新要求に対して 1 つのレ

コードとして表されます。コンテンツは、要求の追加、変更、削除、および名前変更で構成されます。

例14.4 LDIF ファイルの例

```
dn: dc=jboss,dc=org
objectclass: top
objectclass: dcObject
objectclass: organization
dc: jboss
o: JBoss

dn: ou=People,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: People

dn: uid=jsmith,ou=People,dc=jboss,dc=org
objectclass: top
objectclass: uidObject
objectclass: person
uid: jsmith
cn: John
sn: Smith
userPassword: theduke

dn: ou=Roles,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=JBossAdmin,ou=Roles,dc=jboss,dc=org
objectclass: top
objectclass: groupOfNames
cn: JBossAdmin
member: uid=jsmith,ou=People,dc=jboss,dc=org
description: the JBossAdmin group
```

バグの報告

14.1.5. LdapExtended ログインモジュール

識別名(DN)

Lightweight Directory Access Protocol(LDAP)では、識別名はディレクトリー内のオブジェクトを一意に識別します。各識別名には、他のオブジェクトと区別するための一意名と場所が必要で、これには属性と値のペア (AVP) を使用します。AVP は、共通名、組織単位などの情報を定義します。LDAP に必要となる一意な文字列は、これらの値の組み合わせになります。

LdapExtended(org.jboss.security.auth.spi.LdapExtLoginModule) は、バインドするユーザーとその関連のロールを検索します。ロールは再帰的にクエリーを行い、DN に従って階層的なロール構造を移動します。

LoginModule オプションには、選択した LDAP JNDI プロバイダーがサポートするオプションがすべて含まれます。標準のプロパティー名の例は次のとおりです。

- Context.INITIAL_CONTEXT_FACTORY = "java.naming.factory.initial"
- Context.SECURITY_PROTOCOL = "java.naming.security.protocol"
- Context.PROVIDER_URL = "java.naming.provider.url"
- Context.SECURITY_AUTHENTICATION = "java.naming.security.authentication"
- Context.REFERRAL = "java.naming.referral"

ログインモジュールの実装ロジックは、以下の順序に従います。

1. 初期 LDAP サーバーバインドは、bindDN および bindCredential プロパティーを使用して認証されます。bindDN は、baseCtxDN ツリーと rolesCtxDN ツリーの両方を検索するパーミッションを持つユーザーです。認証する user DN は、baseFilter プロパティーで指定されたフィルターを使用してクエリーされます。
2. userDN を InitialLdapContext 環境 userDN として使用して、生成される Context.SECURITY_PRINCIPAL は LDAP サーバーにバインドされ、認証されます。Context.SECURITY_CREDENTIALS プロパティーは、コールバックハンドラーが取得した String パスワードに設定されます。
3. これに成功すると、関連付けられたユーザーロールは rolesCtxDN、roleAttributeID、roleAttributeIsDN、roleNameAttributeID、および roleFilter オプションを使用してクエリーされます。

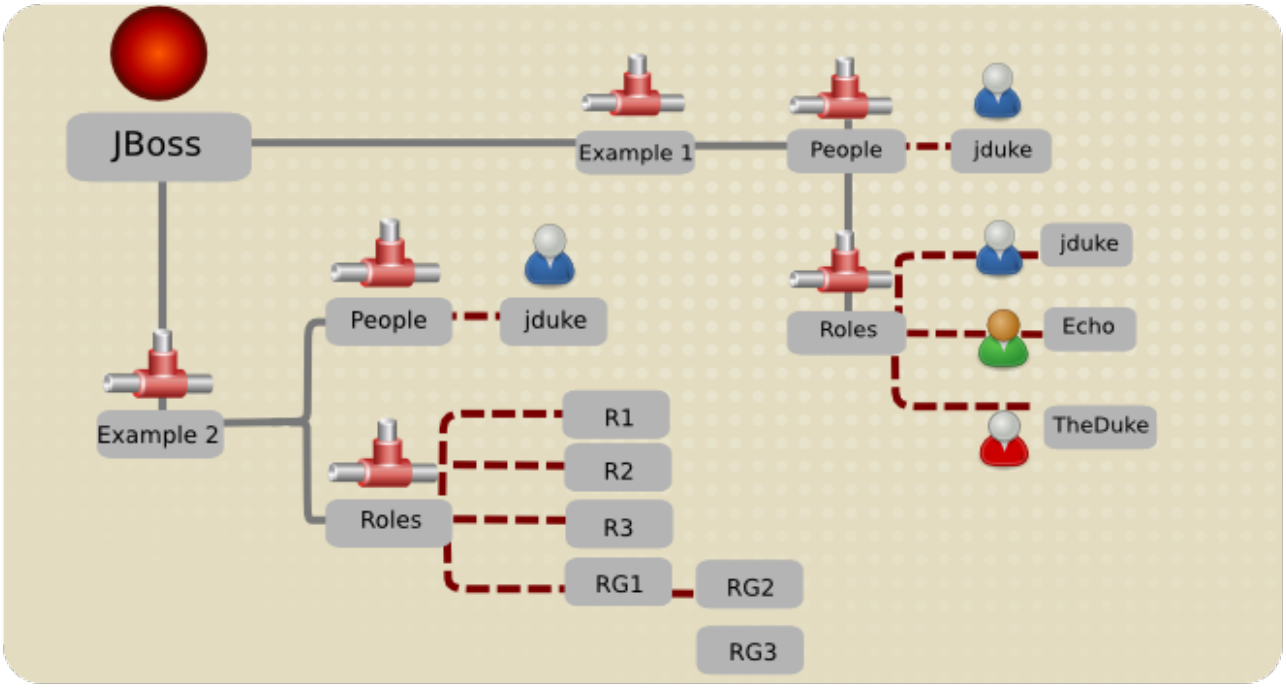
注記

AdvancedLDAP ログインモジュールは、以下の点で LdapExtended ログインモジュールとは異なります。

- 最上位のロールは roleAttributeID に対してのみクエリーされ、roleNameAttributeID にはクエリーされません。
- roleAttributeIsDN モジュールプロパティーを false に設定すると、recurseRoles モジュールオプションが true に設定されている場合でも再帰的なロール検索が無効になります。

LdapExtended ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

図14.1 LDAP 構造の例



[D]

例14.5 2 つの LDAP 設定の例

```
version: 1
dn: o=example2,dc=jboss,dc=org
```

```
objectClass: top
objectClass: organization
o: example2

dn: ou=People,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: People

dn: uid=jduke,ou=People,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: uidObject
objectClass: person
objectClass: inetOrgPerson
cn: Java Duke
employeeNumber: judke-123
sn: Duke
uid: jduke
userPassword:: dGhlZHVrZQ==

dn: uid=jduke2,ou=People,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: uidObject
objectClass: person
objectClass: inetOrgPerson
cn: Java Duke2
employeeNumber: judke2-123
sn: Duke2
uid: jduke2
userPassword:: dGhlZHVrZTI=

dn: ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Roles

dn: uid=jduke,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupUserEx
memberOf: cn=Echo,ou=Roles,o=example2,dc=jboss,dc=org
memberOf: cn=TheDuke,ou=Roles,o=example2,dc=jboss,dc=org
uid: jduke

dn: uid=jduke2,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupUserEx
memberOf: cn=Echo2,ou=Roles,o=example2,dc=jboss,dc=org
memberOf: cn=TheDuke2,ou=Roles,o=example2,dc=jboss,dc=org
uid: jduke2

dn: cn=Echo,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: Echo
description: the echo role
member: uid=jduke,ou=People,dc=jboss,dc=org
```

```
dn: cn=TheDuke,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: TheDuke
description: the duke role
member: uid=jduke,ou=People,o=example2,dc=jboss,dc=org

dn: cn=Echo2,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: Echo2
description: the Echo2 role
member: uid=jduke2,ou=People,dc=jboss,dc=org

dn: cn=TheDuke2,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: TheDuke2
description: the duke2 role
member: uid=jduke2,ou=People,o=example2,dc=jboss,dc=org

dn: cn=JBossAdmin,ou=Roles,o=example2,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: JBossAdmin
description: the JBossAdmin group
member: uid=jduke,ou=People,dc=jboss,dc=org
```

この LDAP 構造例のモジュール設定は、以下の管理 CLI コマンドで説明されています。

```
/subsystem=security/security-domain=testLdapExample2/authentication=classic/login-
module=LdapExtended:add( \
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
    ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"), \
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
    ("java.naming.security.authentication"=>"simple"), \
    ("bindDN"=>"cn=Root,dc=jboss,dc=org"), \
    ("bindCredential"=>"secret1"), \
    ("baseCtxDN"=>"ou=People,o=example2,dc=jboss,dc=org"), \
    ("baseFilter"=>"(uid={0})"), \
    ("rolesCtxDN"=>"ou=Roles,o=example2,dc=jboss,dc=org"), \
    ("roleFilter"=>"(uid={0})"), \
    ("roleAttributeIsDN"=>"true"), \
    ("roleAttributeID"=>"memberOf"), \
    ("roleNameAttributeID"=>"cn") \
  ])
```

例14.6 3 つの LDAP 設定の例

```
dn: o=example3,dc=jboss,dc=org
objectclass: top
objectclass: organization
o: example3

dn: ou=People,o=example3,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: People

dn: uid=jduke,ou=People,o=example3,dc=jboss,dc=org
objectclass: top
objectclass: uidObject
objectclass: person
objectClass: inetOrgPerson
uid: jduke
employeeNumber: judke-123
cn: Java Duke
sn: Duke
userPassword: theduke

dn: ou=Roles,o=example3,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Roles

dn: uid=jduke,ou=Roles,o=example3,dc=jboss,dc=org
objectClass: top
objectClass: groupUserEx
memberOf: cn=Echo,ou=Roles,o=example3,dc=jboss,dc=org
memberOf: cn=TheDuke,ou=Roles,o=example3,dc=jboss,dc=org
uid: jduke

dn: cn=Echo,ou=Roles,o=example3,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
cn: Echo
description: the JBossAdmin group
member: uid=jduke,ou=People,o=example3,dc=jboss,dc=org

dn: cn=TheDuke,ou=Roles,o=example3,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: TheDuke
member: uid=jduke,ou=People,o=example3,dc=jboss,dc=org
```

この LDAP 構造例のモジュール設定は、以下の管理 CLI コマンドで説明されています。

```
/subsystem=security/security-domain=testLdapExample3/authentication=classic/login-
module=LdapExtended:add( \
 code=LdapExtended, \
 flag=required, \
 module-options=[ \
  ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"), \
  ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
  ("java.naming.security.authentication"=>"simple"), \
  ("bindDN"=>"cn=Root,dc=jboss,dc=org"), \
  ("bindCredential"=>"secret1"), \
  ("baseCtxDN"=>"ou=People,o=example3,dc=jboss,dc=org"), \
  ("baseFilter"=>"(cn={0})"), \
  ("rolesCtxDN"=>"ou=Roles,o=example3,dc=jboss,dc=org"), \
  ("roleFilter"=>"(member={1})"), \
  ("roleAttributeID"=>"cn") \
 ])
```

例14.7 4 つの LDAP 設定の例

```
dn: o=example4,dc=jboss,dc=org
objectclass: top
objectclass: organization
o: example4

dn: ou=People,o=example4,dc=jboss,dc=org
objectclass: top
objectclass: organizationalUnit
ou: People

dn: uid=jduke,ou=People,o=example4,dc=jboss,dc=org
objectClass: top
objectClass: uidObject
objectClass: person
objectClass: inetOrgPerson
cn: Java Duke
employeeNumber: jduke-123
sn: Duke
uid: jduke
userPassword:: dGhlZHVrZQ==

dn: ou=Roles,o=example4,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Roles

dn: cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: RG1
member: cn=empty
```

```
dn: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: RG2
member: cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
member: uid=jduke,ou=People,o=example4,dc=jboss,dc=org

dn: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: RG3
member: cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R1,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R1
member: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R2,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R2
member: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R3,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R3
member: cn=RG2,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
member: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R4,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R4
member: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org

dn: cn=R5,ou=Roles,o=example4,dc=jboss,dc=org
objectClass: groupOfNames
objectClass: top
cn: R5
member: cn=RG3,cn=RG1,ou=Roles,o=example4,dc=jboss,dc=org
member: uid=jduke,ou=People,o=example4,dc=jboss,dc=org
```

この LDAP 構造例のモジュール設定は、コードサンプルで説明されています。

```
/subsystem=security/security-domain=testLdapExample4/authentication=classic/login-
module=LdapExtended:add( \
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
```

```
  ("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"), \
  ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
  ("java.naming.security.authentication"=>"simple"), \
  ("bindDN"=>"cn=Root,dc=jboss,dc=org"), \
  ("bindCredential"=>"secret1"), \
  ("baseCtxDN"=>"ou=People,o=example4,dc=jboss,dc=org"), \
  ("baseFilter"=>"(cn={0})"), \
  ("rolesCtxDN"=>"ou=Roles,o=example4,dc=jboss,dc=org"), \
  ("roleFilter"=>"(member={1})"), \
  ("roleRecursion"=>"1"), \
  ("roleAttributeID"=>"memberOf") \
 ])
```

例14.8 デフォルトの Active Directory 設定

以下の例は、デフォルトの Active Directory 設定を示しています。

一部の Active Directory 設定には、通常のポート 389 ではなく、ポート 3268 のグローバルカタログに対する検索が必要になる場合があります。これは、Active Directory フォレストに複数のドメインが含まれる場合によく見られます。

```
/subsystem=security/security-domain=AD_Default/authentication=classic/login-
module=LdapExtended:add( \
 code=LdapExtended, \
 flag=required, \
 module-options=[ \
  ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
  ("bindDN"=>"JBOSS\searchuser"), \
  ("bindCredential"=>"password"), \
  ("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
  ("baseFilter"=>"(sAMAccountName={0})"), \
  ("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
  ("roleFilter"=>"(sAMAccountName={0})"), \
  ("roleAttributeID"=>"memberOf"), \
  ("roleAttributeIsDN"=>"true"), \
  ("roleNameAttributeID"=>"cn"), \
  ("searchScope"=>"ONELEVEL_SCOPE"), \
  ("allowEmptyPasswords"=>"false") \
 ])
```

例14.9 再帰的なロールの Active Directory 設定

以下の例では、Active Directory 内で再帰的なロール検索を実装します。この例と、デフォルト

の Active Directory の例の主な違いは、ユーザーの DN を使用してメンバー属性を検索するためにロール検索が置き換えられている点です。ログインモジュールは、ロールの DN を使用して、グループがメンバーとなっているグループを検索します。

```
/subsystem=security/security-domain=AD_Recursive/authentication=classic/login-
module=LdapExtended:add( \
  code=LdapExtended, \
  flag=required, \
  module-options=[ \
    ("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), \
    ("java.naming.referral"=>"follow"), \
    ("bindDN"=>"JBOSS\searchuser"), \
    ("bindCredential"=>"password"), \
    ("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
    ("baseFilter"=>"(sAMAccountName={0})"), \
    ("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"), \
    ("roleFilter"=>"(member={1})"), \
    ("roleAttributeID"=>"cn"), \
    ("roleAttributeIsDN"=>"false"), \
    ("roleRecursion"=>"2"), \
    ("searchScope"=>"ONELEVEL_SCOPE"), \
    ("allowEmptyPasswords"=>"false") \
  ])
```

バグの報告

14.1.6. UsersRoles ログインモジュール

UsersRoles ログインモジュールは、Java プロパティーファイルからロードされる複数のユーザーおよびユーザーロールをサポートする簡単なログインモジュールです。デフォルトの username-to-password マッピングのファイル名は users.properties で、デフォルトの username-to-roles マッピングのファイル名は roles.properties です。

UsersRoles ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

このログインモジュールは、パスワードスタッキング、パスワードハッシュ、および認証されていないアイデンティティーをサポートします。

プロパティーファイルは、initialize メソッドスレッドコンテキストローダーを使用して初期化中にロードされます。つまり、これらのファイルは Java EE デプロイメントのクラスパス（例： WAR アー

カイブの WEB-INF/classes フォルダー）またはサーバークラスパス上の任意のディレクトリーに配置できます。このログインモジュールの主な目的は、アプリケーションとともにデプロイされたプロパティーファイルを使用して複数のユーザーおよびロールのセキュリティー設定を簡単にテストすることです。

例14.10 UsersRoles ログインモジュール

```
/subsystem=security/security-domain=ejb3-sampleapp/authentication=classic/login-
module=UsersRoles:add( \
  code=UsersRoles, \
  flag=required, \
  module-options=[ \
    ("usersProperties"=>"ejb3-sampleapp-users.properties"), \
    ("rolesProperties"=>"ejb3-sampleapp-roles.properties") \
  ])
```

例14.10「UsersRoles ログインモジュール」 では 、ejb3-sampleapp-users.properties ファイルは username=password 形式を使用します。

```
username1=password1
username2=password2
...
```

例14.10「UsersRoles ログインモジュール」 で参照される ejb3-sampleapp-roles.properties ファイルは、任意のグループ名の値で username=role1,role2 パターンを使用します。以下に例を示します。

```
username1=role1,role2,...
username1.RoleGroup1=role3,role4,...
username2=role1,role3,...
```

ejb3-sampleapp-roles.properties にある user name.XXX プロパティー名パターンは、特定の名前付きロールのグループにこのユーザー名のロールを割り当てます。ここで、プロパティー名の XXX 部分はグループ名です。user name=... フォームは、user name.Roles=... の省略形です。ここで、Roles グループ名は、JBossAuthorizationManager がユーザーのパーミッションを定義するロールが含まれることが想定される標準名です。

以下は、jduke ユーザー名の同等の定義です。

```
jduke=TheDuke,AnimatedCharacter
jduke.Roles=TheDuke,AnimatedCharacter
```

バグの報告

14.1.7. Database ログインモジュール

Database ログインモジュールは、認証とロールマッピングをサポートする JDBC(Java Database Connectivity-based)ログインモジュールです。ユーザー名、パスワード、およびロール情報をリレーショナルデータベースに保存している場合は、このログインモジュールを使用します。

注記

このモジュールは、パスワードスタッキング、パスワードハッシュ、および認証されていない ID をサポートします。

Database ログインモジュールは 2 つの論理テーブルに基づいています。

```
Table Principals(PrincipalID text, Password text)
Table Roles(PrincipalID text, Role text, RoleGroup text)
```

Principals テーブルは PrincipalID ユーザーと有効なパスワードと、Roles テーブルは PrincipalID ユーザーとそのロールセットを関連付けます。ユーザーパーミッションに使用されるロールは、Roles の RoleGroup コラムの値を持つ行に含まれる必要があります。

ログインモジュールが使用する SQL クエリーを指定できる点で、テーブルは論理的です。唯一の要件として、java.sql.ResultSet は前述の Principals および Roles と同じ論理構造を持ちます。テーブル名およびコラムの実際の名前は、コラムのインデックスに基づいてアクセスされるため、関係ありません。

この概念を明確にするために、すでに宣言済みの Principals と Roles の 2 つのテーブルが含まれるデータベースを考慮してください。以下のステートメントは、以下のデータをテーブルに追加します。

- Principals テーブルに PrincipalID が java でパスワードが echoman の java
- Roles テーブルの RolesRoleGroup には PrincipalID が java でロールが Echo のデータを投入
-

Roles テーブルの CallerPrincipalRoleGroup に PrincipalID が java でロールが caller_java である

```
INSERT INTO Principals VALUES('java', 'echoman')
INSERT INTO Roles VALUES('java', 'Echo', 'Roles')
INSERT INTO Roles VALUES('java', 'caller_java', 'CallerPrincipal')
```

Database ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

Database ログインモジュール 設定の例は、以下のように作成できます。

```
CREATE TABLE Users(username VARCHAR(64) PRIMARY KEY, passwd VARCHAR(64))
CREATE TABLE UserRoles(username VARCHAR(64), role VARCHAR(32))
```

セキュリティードメインに対応するログインモジュール設定：

```
/subsystem=security/security-domain=testDB/authentication=classic/login-
module=Database:add( \
 code=Database, \
 flag=required, \
 module-options=[ \
  ("dsJndiName"=>"java:/MyDatabaseDS"), \
  ("principalsQuery"=>"select passwd from Users where username=?"), \
  ("rolesQuery"=>"select role, 'Roles' from UserRoles where username=?") \
 ])
```

バグの報告

14.1.8. Certificate ログインモジュール

証明 書 ログインモジュールは、X509 証明書に基づいてユーザーを認証します。このログインモジュールの典型的なユースケースが、web 層の CLIENT-CERT 認証です。

このログインモジュールは認証のみを実行します。セキュアな Web または EJB コンポーネントへのアクセスを完全に定義するために、承認ロールを取得できる別のログインモジュールと組み合わせる必要があります。このログインモジュールの 2 つのサブクラスである CertRolesLoginModule と DatabaseCertLoginModule は動作を拡張し、プロパティーファイルまたはデータベースから承認ロールを取得します。

Certificate ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してくだ

さい。

Certificate ログインモジュールは、ユーザー検証を実行するには KeyStore が必要です。これは、以下の設定フラグメントに示されるように、リンクされたセキュリティードメインの JSSE 設定から取得されます。

```
/subsystem=security/security-domain=trust-domain:add
/subsystem=security/security-domain=trust-domain/jsse=classic:add( \
  truststore={ \
    password=>pass1234, \
    url=>/home/jbosseap/trusted-clients.jks \
  })

/subsystem=security/security-domain=testCert:add
/subsystem=security/security-domain=testCert/authentication=classic:add
/subsystem=security/security-domain=testCert/authentication=classic/login-
module=Certificate:add( \
  code=Certificate, \
  flag=required, \
  module-options=[ \
    ("securityDomain"=>"trust-domain"), \
  ])
```

手順14.1 証明書およびロールベースの承認を使用した Web アプリケーションのセキュア化

この手順では、クライアント証明書とロールベースの承認を使用して、user-app.war などの Web アプリケーションのセキュリティーを保護する方法を説明します。この例では、CertificateRoles ログインモジュールが認証および承認に使用されます。trusted-clients.keystore と app-roles.properties の両方には、クライアント証明書に関連付けられたプリンシパルにマップするエントリーが必要です。

デフォルトでは、例14.11「証明書の例」 で指定される DN などのクライアント証明書識別名を使用してプリンシパルが作成されます。

1. リソースおよびロールの宣言

   web.xml を変更し、認証および承認に使用される許可されるロールとセキュリティードメインと共にセキュアなリソースを宣言します。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

      <security-constraint>
```

```
            <web-resource-collection>
                <web-resource-name>Protect App</web-resource-name>
                <url-pattern>/*</url-pattern>
            </web-resource-collection>
            <auth-constraint>
                <role-name>Admin</role-name>
            </auth-constraint>
        </security-constraint>

        <login-config>
            <auth-method>CLIENT-CERT</auth-method>
            <realm-name>Secured area</realm-name>
        </login-config>

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

2. セキュリティードメインの指定

    jboss-web.xml ファイルで、必要なセキュリティードメインを指定します。

```
<jboss-web>
  <security-domain>app-sec-domain</security-domain>
</jboss-web>
```

3. ログインモジュールの設定

    管理 CLI を使用して指定した app-sec-domain ドメインのログインモジュール設定を定義します。

```
[
/subsystem=security/security-domain=trust-domain:add
/subsystem=security/security-domain=trust-domain/jsse=classic:add( \
  truststore={ \
    password=>pass1234, \
    url=>/home/jbosseap/trusted-clients.jks \
  })

/subsystem=security/security-domain=app-sec-domain:add
/subsystem=security/security-domain=app-sec-domain/authentication=classic:add
/subsystem=security/security-domain=app-sec-domain/authentication=classic/login-
module=CertificateRoles:add( \
  code=CertificateRoles, \
  flag=required, \
  module-options=[ \
    ("securityDomain"=>"trust-domain"), \
```

```
  ("rolesProperties"=>"app-roles.properties") \
])
```

例14.11 証明書の例

```
[conf]$ keytool -printcert -file valid-client-cert.crt
Owner: CN=valid-client, OU=Security QE, OU=JBoss, O=Red Hat, C=CZ
Issuer: CN=EAP Certification Authority, OU=Security QE, OU=JBoss, O=Red Hat, C=CZ
Serial number: 2
Valid from: Mon Mar 24 18:21:55 CET 2014 until: Tue Mar 24 18:21:55 CET 2015
Certificate fingerprints:
        MD5:  0C:54:AE:6E:29:ED:E4:EF:46:B5:14:30:F2:E0:2A:CB
        SHA1: D6:FB:19:E7:11:28:6C:DE:01:F2:92:2F:22:EF:BB:5D:BF:73:25:3D
        SHA256:
CD:B7:B1:72:A3:02:42:55:A3:1C:30:E1:A6:F0:20:B0:2C:0F:23:4F:7A:8E:2F:2D:FA:AF:55:3E:A7:9B
:2B:F4
        Signature algorithm name: SHA1withRSA
        Version: 3
```

trusted-clients.keystore では、CN=valid-client、OU=Security QE、OU=JBoss、O=Red Hat、C=CZ のエイリアスで 例14.11「証明書の例」 に保存されている証明書が必要です。app-roles.properties に同じエントリーが必要です。DN には通常区切り文字として扱われる文字が含まれるため、以下のようにバックスラッシュ('\')を使用して問題文字をエスケープする必要があります。

```
# A sample app-roles.properties file
CN\=valid-client,\ OU\=Security\ QE,\ OU\=JBoss,\ O\=Red\ Hat,\ C\=CZ
```

バグの報告

14.1.9. Identity ログインモジュール

Identity ログインモジュールは、ハードコーディングされたユーザー名をモジュールに対して認証されたサブジェクトに関連付ける簡単なログインモジュールです。プリンシパル オプションで指定した名前を使用して SimplePrincipal インスタンスを作成します。

注記

このモジュールは、パスワードスタッキングをサポートします。

このログインモジュールは、固定のアイデンティティーをサービスに提供する必要がある場合や、指定のプリンシパルと関連ロールに関連付けられたセキュリティーをテストする必要がある場合などに

便利です。

Identity ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

セキュリティードメイン設定の例を以下で説明します。すべてのユーザーを jduke という名前のプリンシパルとして認証し、TheDuke および AnimatedCharacter のロール名を割り当てます。

```
/subsystem=security/security-domain=testIdentity:add
/subsystem=security/security-domain=testIdentity/authentication=classic:add
/subsystem=security/security-domain=testIdentity/authentication=classic/login-
module=Identity:add( \
  code=Identity, \
  flag=required, \
  module-options=[ \
    ("principal"=>"jduke"), \
    ("roles"=>"TheDuke,AnimatedCharacter") \
  ])
```

バグの報告

14.1.10. RunAs ログインモジュール

RunAs ログインモジュールは、認証のログインフェーズの間に 実行をロールと してスタックにプッシュするヘルパーモジュールで、コミットまたはアボートフェーズでスタックからロール として実行 をポップします。

このログインモジュールの目的は、認証を実行するためにセキュアなリソースにアクセスする必要のある他のログインモジュール（セキュリティーが保護された EJB にアクセスするログインモジュールなど）にロールを提供することです。RunAs ログインモジュールは、run-as ロールの構築が必要なログインモジュールよりも先に設定する必要があります。

RunAs ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

バグの報告

14.1.10.1. RunAsIdentity Creation

JBoss EAP 6 が EJB メソッドへのアクセスをセキュアにするには、メソッド呼び出し時にユーザーのアイデンティティーを認識する必要があります。

サーバーのユーザーのアイデンティティーは、javax.security.auth.Subject インスタンスまたは org.jboss.security.RunAsIdentity インスタンスによって表されます。これらのクラスはいずれも、アイデンティティーを表す 1 つ以上のプリンシパルとアイデンティティーが所有するロールの一覧を保存します。javax.security.auth.Subject の場合、認証情報の一覧も保存されます。

ejb-jar.xml デプロイメント記述子の <assembly-descriptor> セクションで、ユーザーがさまざまな EJB メソッドにアクセスするために必要なロールを 1 つ以上指定します。これらの一覧の比較は、ユーザーに EJB メソッドへのアクセスに必要なロールの 1 つがあるかどうかを示します。

例14.12 org.jboss.security.RunAsIdentity Creation

ejb-jar.xml ファイルで、<session> 要素の子として定義された <security-identity> ロールで <run-as> 要素を指定します。

```
<session>
  ...
  <security-identity>
    <run-as>
      <role-name>Admin</role-name>
    </run-as>
  </security-identity>
  ...
</session>
```

この宣言は、Admin RunAsIdentity ロールを作成する必要があることを示します。

管理 ロールのプリンシパルに名前を付けるには、jboss-ejb3 .xml ファイルに <run-as-principal > 要素を定義します。

```
<jboss:ejb-jar
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
      xmlns:s="urn:security:1.1"
      version="3.1" impl-version="2.0">
   <assembly-descriptor>
      <s:security>
         <ejb-name>WhoAmIBean</ejb-name>
         <s:run-as-principal>John</s:run-as-principal>
      </s:security>
   </assembly-descriptor>
```

```
</jboss:ejb-jar>
```

jboss -ejb3.xml ファイルの ejb-jar.xml と < security > 要素の両方にある <security- identity > 要素はデプロイメント時に解析されます。& lt;run-as& gt; のロール名と < run-as-principal > 名は org.jboss.metadata.ejb.spec.SecurityIdentityMetaData クラスに保存されます。

例14.13 RunAsIdentity への複数ロールの割り当て

ロールを jboss-ejb3.xml デプロイメント記述子 < assembly- descriptor > 要素グループのプリンシパルにマッピングすることで、より多くのロールを RunAsIdentity に割り当てることができます。

```
<jboss:ejb-jar xmlns:sr="urn:security-role"
      ...>
      <assembly-descriptor>
         ...
            <sr:security-role>
                  <sr:role-name>Support</sr:role-name>
                  <sr:principal-name>John</sr:principal-name>
                  <sr:principal-name>Jill</sr:principal-name>
                  <sr:principal-name>Tony</sr:principal-name>
            </sr:security-role>
      </assembly-descriptor>
</jboss:ejb-jar>
```

例14.12「org.jboss.security.RunAsIdentity Creation」 で、John の <run-as-principal> が作成されました。この例の設定では、Support ロールを追加して Admin ロールを拡張します。新しいロールには、最初に定義したプリンシパル John を含む追加のプリンシパルが含まれます。

ejb -jar.xml ファイルおよび jboss-ejb3.xml ファイル両方の <security- role> 要素は、デプロイメント時に解析されます。<role-name& gt; および <principal-name > データは org.jboss.metadata.ejb.spec.SecurityIdentityMetaData クラスに保存されます。

バグの報告

14.1.11. Client ログインモジュール

クライアントログインモジュール(org.jboss.security. Client LoginModule)は、呼び出し元のアイデンティティーとクレデンシャルを確立するときに JBoss クライアントによって使用される LoginModule の実装です。これにより、新しい SecurityContext がプリンシパルとクレデンシャルに割り当て、SecurityContext を ThreadLocal セキュリティーコンテキストに設定します。

クライアント ログインモジュールは、クライアントが現在のスレッドの呼び出し元を確立するために唯一サポートされているメカニズムです。スタンドアロンクライアントアプリケーションとサーバー環境（セキュリティー環境が EAP security サブシステムを使用するよう透過的に設定されていない JBoss EJB クライアントとして機能するため）は、Client ログインモジュールを使用する必要があります。

このログインモジュールは認証を実行しないことに注意してください。サーバー上の後続の認証のために、提供されたログイン情報をサーバー EJB 呼び出しレイヤーにコピーすることもほとんどありません。ユーザーのクライアント側の認証を実行する必要がある場合は、Client ログインモジュールに加えて、別のログインモジュールを設定する必要があります。

Client ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

バグの報告

14.1.12. SPNEGO ログインモジュール

SPNEGO ログインモジュール(org.jboss.security.negotiation.spnego.SPNEGOLoginModule)は、KDC で呼び出し元のアイデンティティーとクレデンシャルを確立する LoginModule の実装です。モジュールは SPNEGO（Simple および Protected GSSAPI Negotiation メカニズム）を実装し、JBoss Negotiation プロジェクトの一部です。この認証は、AdvancedLdap ログインモジュールとチェーンされた設定で使用することができ、LDAP サーバーと連携できるようにします。

SPNEGO ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

JBoss Negotiation モジュールは、デプロイされたアプリケーションの標準依存関係として含まれません。プロジェクトで SPNEGO または AdvancedLdap ログインモジュールを使用するには、META-INF/jboss-deployment-structure.xml デプロイメント記述子ファイルを編集して依存関係を手動で追加する必要があります。

例14.14 JBoss Negotiation モジュールを依存関係として追加

```
<jboss-deployment-structure>
```

```
  <deployment>
    <dependencies>
      <module name="org.jboss.security.negotiation" />
    </dependencies>
  </deployment>
</jboss-deployment-structure>
```

バグの報告

14.1.13. RoleMapping ログインモジュール

RoleMapping ログインモジュールは、認証プロセスの最終結果であるロールを 1 つ以上の宣言的ロールへのマッピングをサポートします。たとえば、ユーザー "A" に "ldapAdmin" と "testAdmin" のロールがあり、web.xml または ejb-jar.xml ファイルで定義された宣言型ロールは admin であると認証プロセスが判断された場合、このログインモジュールは admin ロールをユーザー A にマッピングします。

RoleMapping ログインモジュールオプションの詳細は、「含まれる認証モジュール」 を参照してください。

RoleMapping ログインモジュールは、以前マップされたロールのマッピングを変更するため、ログインモジュール設定でオプションのモジュールとして定義する必要があります。

例14.15 マッピングされたロールの定義

```
/subsystem=security/security-domain=test-domain-2/:add
/subsystem=security/security-domain=test-domain-2/authentication=classic:add
/subsystem=security/security-domain=test-domain-2/authentication=classic/login-
module=test-2-lm/:add(\
flag=required,\
code=UsersRoles,\
module-options=[("usersProperties"=>"users.properties"),
("rolesProperties"=>"roles.properties")]\
)
/subsystem=security/security-domain=test-domain-2/authentication=classic/login-
module=test2-map/:add(\
flag=optional,\
code=RoleMapping,\
module-options=[("rolesProperties"=>"rolesMapping-roles.properties")]\
)
```

もう 1 つの例は、同じ結果を実現しますが、マッピングモジュールを使用します。これは、ロールマッピングの推奨される方法です。

例14.16 マップされたロールを定義するための推奨される方法

```
/subsystem=security/security-domain=test-domain-2/:add
/subsystem=security/security-domain=test-domain-2/authentication=classic:add
/subsystem=security/security-domain=test-domain-2/authentication=classic/login-
module=test-2-lm/:add(\
flag=required,\
code=UsersRoles,\
module-options=[("usersProperties"=>"users.properties"),
("rolesProperties"=>"roles.properties")]\
)
/subsystem=security/security-domain=test-domain-2/mapping=classic/mapping-
module=test2-map/:add(\
code=PropertiesRoles,type=role,\
module-options=[("rolesProperties"=>"rolesMapping-roles.properties")]\
)
```

例14.17 RoleMappingLoginModule によって使用されるプロパティーファイル

```
ldapAdmin=admin, testAdmin
```

認証されたサブジェクトに ldapAdmin ロールが含まれている場合は、replaceRole プロパティーの値に応じて admin ロールおよび testAdmin ロールが追加されたり、認証されたサブジェクトを置き換えたりします。

バグの報告

14.1.14. bindCredential モジュールオプション

bindCredential モジュールオプションは、DN の認証情報を保存するために使用され、複数のログインおよびマッピングモジュールで使用できます。パスワードを取得する方法は複数あります。

管理 CLI コマンドのプレーンテキストです。

bindCredential モジュールのパスワードは、管理 CLI コマンドでプレーンテキストで指定できます。たとえば、("bindCredential"=>"secret1") のようになります。セキュリティー上の理由から、パスワードは JBoss EAP vault のメカニズムを使用して暗号化する必要があります。

外部コマンドを使用する。

外部コマンドの出力からパスワードを取得するには、{EXT}... の形式を使用します。ここで、... は外部コマンドになります。コマンド出力の最初の行がパスワードとして使用されます。

パフォーマンスを向上させるために、{EXTC[:expiration_in_millis]} バリアントは指定された数のミリ秒のパスワードをキャッシュします。デフォルトでは、キャッシュされたパスワードは期限切れになりません。0 （ゼロ）の値を指定すると、キャッシュされた認証情報の有効期限はありません。

EXTC バリアントは LdapExtended ログインモジュールでのみサポートされます。

例14.18 外部コマンドからパスワードを取得します。

```
{EXT}cat /mysecretpasswordfile
```

例14.19 外部ファイルからパスワードを取得し、500 ミリ秒キャッシュします。

```
{EXTC:500}cat /mysecretpasswordfile
```

バグの報告

14.2. カスタムモジュール

EAP セキュリティーフレームワークにバンドルされたログインモジュールがセキュリティー環境で機能しない場合は、独自のカスタムログインモジュール実装を作成できます。AuthenticationManager では、Subject プリンシパルの特定の使用パターンが必要です。AuthenticationManager で動作するログインモジュールを作成するには、JAAS Subject クラスの情報ストレージ機能と、これらの機能の予想される使用方法を理解している。

ここではこの要件を調べ、カスタムログインモジュールの実装に役立つ 2 つの抽象ベース LoginModule 実装を紹介します。

以下の方法を使用して、Subject に関連するセキュリティー情報を取得できます。

```
java.util.Set getPrincipals()
java.util.Set getPrincipals(java.lang.Class c)
java.util.Set getPrivateCredentials()
java.util.Set getPrivateCredentials(java.lang.Class c)
java.util.Set getPublicCredentials()
java.util.Set getPublicCredentials(java.lang.Class c)
```

Subject identity および roles の場合、EAP は最も論理的に選択されています。これは、getPrincipals（）および getPrincipals (java.lang.Class) で取得したプリンシパルセットです。使用パターンは以下のとおりです。

- ユーザー ID（ユーザー名、ソーシャルセキュリティー番号、従業員 ID など）は、サブジェクトプリンシパル セットに java.security.Principal オブジェクトとして保存されます。ユーザー ID を表す Principal 実装は、プリンシパルの名前に対するベース比較と等価でなければなりません。org.jboss.security.SimplePrincipal クラスとして適切な実装を利用できます。他の プリンシパル インスタンスは、必要に応じて設定された SubjectPrincipals に追加できます。

- 割り当てられたユーザーロールは Principals セットに保存され、java.security.acl.Group インスタンスを使用して名前付きのロールセットにグループ化されます。Group インターフェースは Principals または Groupのコレクションを定義し、java.security.Principal のサブインターフェースです。

- サブジェクト には任意の数のロールセットを割り当てることができます。

- EAP セキュリティーフレームワークは、Roles と CallerPrincipal という名前の 2 つのよく知られたロールセットを使用します。

  - Roles グループは、Subject が認証されたアプリケーションドメインで既知の名前付きロールの Principalのコレクションです。このロールセットは、ejb.isCallerInRole(String)などのメソッドによって使用されます。 EJB は、現在の呼び出し元が名前付きのアプリケーションドメインロールに属するかどうかを確認するために使用できます。メソッドパーミッションチェックを実行するセキュリティーインターセプターロジックは、このロールセットも使用します。

- CallerPrincipal グループ は、アプリケーションドメインのユーザーに割り当てられる単一の Principal アイデンティティーで構成されます。gitops .getCallerPrincipal（）メソッドは CallerPrincipal を使用して、アプリケーションドメインのオペレーション環境アイデンティティーからアプリケーションに適したユーザー ID へのマッピングを許可します。Subject に CallerPrincipal Group がない場合、アプリケーションのアイデンティティーは動作する環境アイデンティティーと同じです。

バグの報告

14.2.1. Subject Usage パターンのサポート

「カスタムモジュール」 で説明されている Subject Usage パターンの正しい実装を簡素化するために、EAP には、認証された Subject に正しいサブジェクトの使用を強制するテンプレートパターンを設定するログインモジュールが含まれます。

AbstractServerLoginModule

この 2 つの最も一般的なものは org.jboss.security.auth.spi.AbstractServerLoginModule クラスです。

javax.security.auth.spi.LoginModule インターフェースの実装を提供し、オペレーション環境セキュリティーインフラストラクチャーに固有のキータスクに抽象メソッドを提供します。クラスの主要な詳細は、例14.20「AbstractServerLoginModule Class Fragment」 で強調表示されています。JavaDoc コメントでは、サブクラスの責任が詳細に説明されています。

重要

loginOk インスタンス変数はピボットです。これは、ログインに成功する場合は true に設定する必要があります。または、ログインメソッドを上書きするすべてのサブクラスで false に設定する必要があります。この変数が正しく設定されている場合、コミットメソッドはサブジェクトを正しく更新しません。

フェーズでログを追跡すると、ログインモジュールを制御フラグと共にチェーンできます。これらの制御フラグは、認証プロセスの一環としてログインモジュールを正常に実行する必要はありません。

例14.20 AbstractServerLoginModule Class Fragment

```
package org.jboss.security.auth.spi;
/**
 * This class implements the common functionality required for a JAAS
 * server-side LoginModule and implements the PicketBox standard
```

```
 * Subject usage pattern of storing identities and roles. Subclass
 * this module to create your own custom LoginModule and override the
 * login(), getRoleSets(), and getIdentity() methods.
 */
public abstract class AbstractServerLoginModule
   implements javax.security.auth.spi.LoginModule
{
   protected Subject subject;
   protected CallbackHandler callbackHandler;
   protected Map sharedState;
   protected Map options;
   protected Logger log;

   /** Flag indicating if the shared credential should be used */
   protected boolean useFirstPass;
   /**
    * Flag indicating if the login phase succeeded. Subclasses that
    * override the login method must set this to true on successful
    * completion of login
    */
   protected boolean loginOk;

   // ...
   /**
    * Initialize the login module. This stores the subject,
    * callbackHandler and sharedState and options for the login
    * session. Subclasses should override if they need to process
    * their own options. A call to super.initialize(...) must be
    * made in the case of an override.
    *
    * <p>
    * The options are checked for the <em>password-stacking</em> parameter.
    * If this is set to "useFirstPass", the login identity will be taken from the
    * <code>javax.security.auth.login.name</code> value of the sharedState map,
    * and the proof of identity from the
    * <code>javax.security.auth.login.password</code> value of the sharedState map.
    *
    * @param subject the Subject to update after a successful login.
    * @param callbackHandler the CallbackHandler that will be used to obtain the
    * the user identity and credentials.
    * @param sharedState a Map shared between all configured login module instances
    * @param options the parameters passed to the login module.
    */
   public void initialize(Subject subject,
                          CallbackHandler callbackHandler,
                          Map sharedState,
                          Map options)
   {
      // ...
   }


   /**
    * Looks for javax.security.auth.login.name and
    * javax.security.auth.login.password values in the sharedState
    * map if the useFirstPass option was true and returns true if
```

```
     * they exist. If they do not or are null this method returns
     * false.
     * Note that subclasses that override the login method
     * must set the loginOk var to true if the login succeeds in
     * order for the commit phase to populate the Subject. This
     * implementation sets loginOk to true if the login() method
     * returns true, otherwise, it sets loginOk to false.
     */
    public boolean login()
       throws LoginException
    {
       // ...
    }

    /**
     * Overridden by subclasses to return the Principal that
     * corresponds to the user primary identity.
     */
    abstract protected Principal getIdentity();

    /**
     * Overridden by subclasses to return the Groups that correspond
     * to the role sets assigned to the user. Subclasses should
     * create at least a Group named "Roles" that contains the roles
     * assigned to the user.  A second common group is
     * "CallerPrincipal," which provides the application identity of
     * the user rather than the security domain identity.
     *
     * @return Group[] containing the sets of roles
     */
    abstract protected Group[] getRoleSets() throws LoginException;
}
```

UsernamePasswordLoginModule

2 つ目の抽象ベースログインモジュールは、org.jboss.security.auth.spi.UsernamePasswordLoginModule です。

このログインモジュールは、文字列ベースのユーザー名をユーザー ID として、認証クレデンシャルとして char[] パスワードを強制することで、カスタムログインモジュール実装をさらに単純化します。また、匿名ユーザー（null ユーザー名およびパスワードによって指定される）とロールのないプリンシパルへのマッピングもサポートします。クラスの主要な詳細は、以下のクラスフラグメントで強調表示されています。JavaDoc コメントでは、サブクラスの責任が詳細に説明されています。

例14.21 UsernamePasswordLoginModule Class Fragment

```
package org.jboss.security.auth.spi;

/**
 * An abstract subclass of AbstractServerLoginModule that imposes a
 * an identity == String username, credentials == String password
```

```
 *  view on the login process. Subclasses override the
 *  getUsersPassword() and getUsersRoles() methods to return the
 *  expected password and roles for the user.
 */
public abstract class UsernamePasswordLoginModule
   extends AbstractServerLoginModule
{
   /** The login identity */
   private Principal identity;
   /** The proof of login identity */
   private char[] credential;
   /** The principal to use when a null username and password are seen */
   private Principal unauthenticatedIdentity;

   /**
    * The message digest algorithm used to hash passwords. If null then
    * plain passwords will be used. */
   private String hashAlgorithm = null;

   /**
    *  The name of the charset/encoding to use when converting the
    * password String to a byte array. Default is the platform's
    * default encoding.
    */
    private String hashCharset = null;

   /** The string encoding format to use. Defaults to base64. */
   private String hashEncoding = null;

   // ...

   /**
    *  Override the superclass method to look for an
    *  unauthenticatedIdentity property. This method first invokes
    *  the super version.
    *
    *  @param options,
    *  @option unauthenticatedIdentity: the name of the principal to
    *  assign and authenticate when a null username and password are
    *  seen.
    */
   public void initialize(Subject subject,
                          CallbackHandler callbackHandler,
                          Map sharedState,
                          Map options)
   {
      super.initialize(subject, callbackHandler, sharedState,
                       options);
      // Check for unauthenticatedIdentity option.
      Object option = options.get("unauthenticatedIdentity");
      String name = (String) option;
      if (name != null) {
         unauthenticatedIdentity = new SimplePrincipal(name);
      }
   }
```

```
    // ...

    /**
     *  A hook that allows subclasses to change the validation of the
     *  input password against the expected password. This version
     *  checks that neither inputPassword or expectedPassword are null
     *  and that inputPassword.equals(expectedPassword) is true;
     *
     *  @return true if the inputPassword is valid, false otherwise.
     */
    protected boolean validatePassword(String inputPassword,
                                       String expectedPassword)
    {
        if (inputPassword == null || expectedPassword == null) {
            return false;
        }
        return inputPassword.equals(expectedPassword);
    }

    /**
     *  Get the expected password for the current username available
     * via the getUsername() method. This is called from within the
     * login() method after the CallbackHandler has returned the
     * username and candidate password.
     *
     * @return the valid password String
     */
    abstract protected String getUsersPassword()
        throws LoginException;
}
```

ログインモジュールのサブクラス

AbstractServerLoginModule と UsernamePasswordLoginModule へのサブクラスの選択は、ログインモジュールを書き込む認証技術で文字列ベースのユーザー名と認証情報を使用できるかどうかに基づいています。文字列ベースのセマンティックが有効な場合は、サブクラスの UsernamePasswordLoginModule、他のサブクラス AbstractServerLoginModule。

サブクラスの手順

カスタムログインモジュールを実行する必要がある手順は、選択したベースログインモジュールクラスによって異なります。セキュリティーインフラストラクチャーと統合するカスタムログインモジュールを作成する場合は、EAP セキュリティーマネージャーが想定される形式でログインモジュールが認証された Principal 情報を提供できるように、AbstractServerLoginModule または UsernamePasswordLoginModule をサブクラス化して開始する必要があります。

AbstractServerLoginModule をサブクラス化する場合は、以下を上書きする必要があります。

-

void initialize(Subject, CallbackHandler, Map, Map): 解析するカスタムオプションがある場合は。

- boolean login（）: 認証アクティビティーを実行する。ログインに成功したら、loginOk インスタンス変数を true に設定してください。失敗した場合は false に設定します。

- プリンシパル getIdentity（）: log （）ステップで認証されたユーザーの Principal オブジェクトを返します。

- group[] getRoleSets（）: login（）時に認証された Principal に割り当てられたロールが含まれる Roles という名前の グループ を 1 つ以上返します。2 つ目の 共通グループ は CallerPrincipal という名前で、セキュリティードメインアイデンティティーではなくユーザーのアプリケーションアイデンティティーを提供します。

UsernamePasswordLoginModule をサブクラス化する場合、以下を上書きする必要があります。

- void initialize(Subject, CallbackHandler, Map, Map): 解析するカスタムオプションがある場合は。

- group[] getRoleSets（）: login（）時に認証された Principal に割り当てられたロールが含まれる Roles という名前の グループ を 1 つ以上返します。2 つ目の 共通グループ は CallerPrincipal という名前で、セキュリティードメインアイデンティティーではなくユーザーのアプリケーションアイデンティティーを提供します。

- string getUsersPassword（）: getUsername （）メソッドで利用可能な現在のユーザー名の想定されるパスワードを返します。getUsersPassword（） メソッドは、callbackhandler がユーザー名およびパスワードを返した後に login（） 内で呼び出されます。

バグの報告

14.2.2. カスタム LoginModule の例

以下の情報は、UsernamePasswordLoginModule を拡張し、JNDI ルックアップからユーザーのパスワードとロール名を取得するカスタム ログインモジュールサンプルの作成に役立ちます。

本セクションの最後に、password /<username> の名前を使用してコンテキストでルックアップを実行する場合に、ユーザーのパスワードを返すカスタム JNDI コンテキストログインモジュールが作成されます（<username > は認証される現在のユーザーです）。同様に、roles/<username> 形式のルックアップが 要求されたユーザーのロールを返します。例14.22「JndiUserAndPassLoginModule Custom Login Module」 では、JndiUserAndPassLoginModule カスタムログインモジュールのソースコードです。

これは JBoss UsernamePasswordLoginModule を拡張するため、JndiUserAndPassLoginModule は JNDI ストアからユーザーのパスワードとロールを取得することに注意してください。JndiUserAndPassLoginModule は JAAS LoginModule 操作と対話しません。

例14.22 JndiUserAndPassLoginModule Custom Login Module

```
package org.jboss.book.security.ex2;

import java.security.acl.Group;
import java.util.Map;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginException;
import org.jboss.logging.Logger;
import org.jboss.security.SimpleGroup;
import org.jboss.security.SimplePrincipal;
import org.jboss.security.auth.spi.UsernamePasswordLoginModule;
/**
 * An example custom login module that obtains passwords and roles for a user from a
JNDI lookup.
 *
 * @author Scott.Stark@jboss.org
 */
public class JndiUserAndPassLoginModule extends UsernamePasswordLoginModule {
  /** The JNDI name to the context that handles the password/username lookup */
  private String userPathPrefix;
  /** The JNDI name to the context that handles the roles/username lookup */
  private String rolesPathPrefix;
  private static Logger log = Logger.getLogger(JndiUserAndPassLoginModule.class);
  /**
   * Override to obtain the userPathPrefix and rolesPathPrefix options.
   */
  @Override
  public void initialize(Subject subject, CallbackHandler callbackHandler, Map sharedState,
Map options) {
    super.initialize(subject, callbackHandler, sharedState, options);
    userPathPrefix = (String) options.get("userPathPrefix");
    rolesPathPrefix = (String) options.get("rolesPathPrefix");
  }
  /**
   * Get the roles the current user belongs to by querying the rolesPathPrefix + '/' +
super.getUsername() JNDI location.
   */
```

```
  @Override
  protected Group[] getRoleSets() throws LoginException {
    try {
      InitialContext ctx = new InitialContext();
      String rolesPath = rolesPathPrefix + '/' + super.getUsername();
      String[] roles = (String[]) ctx.lookup(rolesPath);
      Group[] groups = { new SimpleGroup("Roles") };
      log.info("Getting roles for user=" + super.getUsername());
      for (int r = 0; r < roles.length; r++) {
        SimplePrincipal role = new SimplePrincipal(roles[r]);
        log.info("Found role=" + roles[r]);
        groups[0].addMember(role);
      }
      return groups;
    } catch (NamingException e) {
      log.error("Failed to obtain groups for user=" + super.getUsername(), e);
      throw new LoginException(e.toString(true));
    }
  }
  /**
   * Get the password of the current user by querying the userPathPrefix + '/' +
super.getUsername() JNDI location.
   */
  @Override
  protected String getUsersPassword() throws LoginException {
    try {
      InitialContext ctx = new InitialContext();
      String userPath = userPathPrefix + '/' + super.getUsername();
      log.info("Getting password for user=" + super.getUsername());
      String passwd = (String) ctx.lookup(userPath);
      log.info("Found password=" + passwd);
      return passwd;
    } catch (NamingException e) {
      log.error("Failed to obtain password for user=" + super.getUsername(), e);
      throw new LoginException(e.toString(true));
    }
  }
}
```

例14.23 新たに作成したカスタムログインモジュールでの security-ex2 セキュリティードメインの定義

```
/subsystem=security/security-domain=security-ex2/:add
/subsystem=security/security-domain=security-ex2/authentication=classic:add
/subsystem=security/security-domain=security-ex2/authentication=classic/login-
module=ex2/:add(\
flag=required,\
code=org.jboss.book.security.ex2.JndiUserAndPassLoginModule,\
module-options=[("userPathPrefix"=>"/security/store/password"),\
("rolesPathPrefix"=>"/security/store/roles")]\
)
```

ユーザーのサーバー側の認証に JndiUserAndPassLoginModule カスタムログインモジュールを使用することは、example セキュリティードメインのログイン設定によって決定されます。EJB JAR の META-INF/jboss-ejb3.xml 記述 子はセキュリティードメインを設定します。Web アプリケーションの場合は、WEB-INF/jboss-web.xml ファイルに含まれます。

例14.24 jboss-ejb3.xml の例

```
<?xml version="1.0"?>
<jboss:ejb-jar xmlns:jboss="http://www.jboss.com/xml/ns/javaee"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:s="urn:security" version="3.1" impl-
version="2.0">
  <assembly-descriptor>
    <s:security>
      <ejb-name>*</ejb-name>
      <s:security-domain>security-ex2</s:security-domain>
    </s:security>
  </assembly-descriptor>
</jboss:ejb-jar>
```

例14.25 jboss-web.xml example

```
<?xml version="1.0"?>
<jboss-web>
    <security-domain>security-ex2</security-domain>
</jboss-web>
```

バグの報告

第15章 アプリケーションのロールベースのセキュリティー

15.1. JAVA AUTHENTICATION AND AUTHORIZATION SERVICE(JAAS)

JAAS(Java Authentication and Authorization Service) は、ユーザーの認証および承認用に設計された Java パッケージのセットで構成されるセキュリティー API です。この API は 標準の PAM (Pluggable Authentication Modules) フレームワークの Java 実装です。Java Enterprise Edition アクセス制御アーキテクチャーを拡張し、ユーザーベースの承認をサポートします。

JBoss EAP 6 では、JAAS は宣言型のロールベースのセキュリティーのみを提供します。宣言型セキュリティーの詳細は、「宣言型セキュリティー」 を参照してください。

JAAS は、Kerberos や LDAP などの基礎となる認証技術からは独立しています。アプリケーションを変更せずに、基礎となるセキュリティー構造を変更できます。JAAS 設定のみを変更するだけです。

バグの報告

15.2. JAAS(JAVA AUTHENTICATION AND AUTHORIZATION SERVICE)

JBoss EAP 6 のセキュリティーアーキテクチャーは、セキュリティー設定サブシステムと、アプリケーション内の複数の設定ファイルに含まれるアプリケーション固有のセキュリティー設定で構成されています。

ドメイン、サーバーグループ、およびサーバー固有の設定

サーバーグループ（管理対象ドメイン）およびサーバー（スタンドアロンサーバー）にはセキュリティードメインの設定が含まれます。セキュリティードメインには、設定の詳細とともに認証、承認、マッピング、および監査モジュールの組み合わせに関する情報が含まれます。アプリケーションは、jboss-web.xml に名前で必要なセキュリティードメインを指定します。

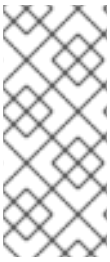
アプリケーション固有の設定

アプリケーション固有の設定は、以下の 4 つのファイルのいずれか 1 つ以上で実行されます。

表15.1 アプリケーション固有の設定ファイル

ファイル	説明

ファイル	説明
ejb-jar.xml	アーカイブの **META-INF** ディレクトリーにある Enterprise JavaBean(EJB)アプリケーションのデプロイメント記述子。**ejb-jar.xml** を使用してロールを指定し、アプリケーションレベルでプリンシパルにマッピングします。特定のメソッドおよびクラスを特定のロールに制限することもできます。また、セキュリティーとは関係のない他の EJB 固有の設定にも使用されます。
web.xml	Java Enterprise Edition(EE)Web アプリケーションのデプロイメント記述子。**web.xml** を使用して、許可される HTTP リクエストのタイプを制限するなど、アプリケーションのリソースおよびトランスポート制約を宣言します。このファイルで簡単な Web ベースの認証を設定することもできます。また、セキュリティーとは関係のない他のアプリケーション固有の設定にも使用されます。アプリケーションが認証および承認に使用するセキュリティードメインは **jboss-web.xml** で定義されます。
jboss-ejb3.xml	**ejb-jar.xml** 記述子への JBoss 固有のエクステンションが含まれます。
jboss-web.xml	**web.xml** 記述子に対する JBoss 固有の拡張が含まれます。



注記

ejb-jar.xml および web.xml は、Java Enterprise Edition(Java EE)仕様で定義されています。jboss-ejb3.xml は ejb-jar.xml に JBoss 固有の拡張機能を提供し、jboss-web.xml は web.xml の JBoss 固有の拡張を提供します。

バグの報告

15.3. サーブレットでのロールベースのセキュリティーの使用

セキュリティーをサーブレットに追加するには、各サーブレットを URL パターンにマッピングし、セキュリティー確保が必要な URL パターンにセキュリティー制約を作成します。セキュリティー制約は URL へのアクセスをロールに制限します。認証および承認は、WAR の jboss-web.xml で指定されたセキュリティードメインによって処理されます。

前提条件

サーブレットでロールベースのセキュリティーを使用する前に、アクセスの認証および承認に使用されるセキュリティードメインを JBoss EAP 6 コンテナーに設定する必要があります。

手順15.1 ロールベースのセキュリティーをサーブレットに追加

1. **サーブレットと URL パターン間のマッピングを追加します。**

   web.xml の < servlet-mapping > 要素を使用して、個別のサーブレットを URL パターンにマッピングします。以下の例では、DisplayOpResult というサーブレットを URL パターン /DisplayOpResult にマッピングします。

```
<servlet-mapping>
    <servlet-name>DisplayOpResult</servlet-name>
    <url-pattern>/DisplayOpResult</url-pattern>
</servlet-mapping>
```

2. **URL パターンにセキュリティー制約を追加します。**

   URL パターンをセキュリティー制約にマッピングするには、< security-constraint> を使用します。以下の例は、eap_admin ロールを持つユーザーがアクセスする URL パターン /DisplayOpResult からのアクセスを制限します。ロールはセキュリティードメインに存在する必要があります。

```
<security-constraint>
 <display-name>Restrict access to role eap_admin</display-name>
 <web-resource-collection>
  <web-resource-name>Restrict access to role eap_admin</web-resource-name>
  <url-pattern>/DisplayOpResult/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
  <role-name>eap_admin</role-name>
 </auth-constraint>
</security-constraint>

<security-role>
  <role-name>eap_admin</role-name>
</security-role>


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

   認証方法を指定する必要があります。BASIC、FORM、DIGEST、CLIENT-CERT、SPNEGO のいずれかです。この例では BASIC 認証を使用します。

3. **WAR の jboss-web.xmlでセキュリティードメインの指定**

サーブレットを設定済みのセキュリティードメインに接続するには、セキュリティードメインを WAR の jboss-web.xml に追加します。これは、セキュリティー制約に対してプリンシパルを認証および承認する方法を認識します。次の例では、acme_domain というセキュリティードメインを使用します。

```
<jboss-web>
 ...
 <security-domain>acme_domain</security-domain>
 ...
</jboss-web>
```

例15.1 ロールベースのセキュリティーが設定された web.xml の例

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
        version="3.0">

<display-name>Use Role-Based Security In Servlets</display-name>

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

<servlet-mapping>
    <servlet-name>DisplayOpResult</servlet-name>
    <url-pattern>/DisplayOpResult</url-pattern>
</servlet-mapping>

<security-constraint>
  <display-name>Restrict access to role eap_admin</display-name>
    <web-resource-collection>
      <web-resource-name>Restrict access to role eap_admin</web-resource-name>
      <url-pattern>/DisplayOpResult/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
        <role-name>eap_admin</role-name>
      </auth-constraint>
    </security-constraint>

    <security-role>
      <role-name>eap_admin</role-name>
    </security-role>

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

</web-app>
```

–

バグの報告

15.4. アプリケーションでのサードパーティーの認証システムの使用

サードパーティーセキュリティーシステムを JBoss EAP 6 と統合できます。通常、これらのタイプのシステムはトークンベースです。外部システムは認証を実行し、要求ヘッダーを介してトークンを Web アプリケーションに戻します。多くの場合、これは perimeter *認証と呼ば*れます。アプリケーションで perimeter 認証を設定するには、カスタム認証バルブを追加します。サードパーティーのプロバイダーからバルブがある場合は、クラスパスにあり、以下の例に従い、サードパーティーの認証モジュールのドキュメントと一緒に確認してください。

注記

JBoss EAP 6 ではバルブを設定する場所が変更になりました。context.xml デプロイメント記述子はなくなりました。バルブは jboss-web.xml 記述子で直接設定されます。context.xml は無視されるようになりました。

例15.2 Basic 認証バルブ

```
<jboss-web>
  <valve>
    <class-name>org.jboss.security.negotiation.NegotiationAuthenticator</class-name>
  </valve>
</jboss-web>
```

このバルブは Kerberos ベースの SSO に使用されます。また、Web アプリケーションのサードパーティーオーセンティケーターを指定する最も単純なパターンも表示します。

例15.3 ヘッダー属性が設定されたカスタムバルブ

```
<jboss-web>
  <valve>
    <class-name>org.jboss.web.tomcat.security.GenericHeaderAuthenticator</class-name>
    <param>
      <param-name>httpHeaderForSSOAuth</param-name>
      <param-value>sm_ssoid,ct-remote-user,HTTP_OBLIX_UID</param-value>
    </param>
    <param>
      <param-name>sessionCookieForSSOAuth</param-name>
      <param-value>SMSESSION,CTSESSION,ObSSOCookie</param-value>
    </param>
```

```
  </valve>
</jboss-web>
```

この例は、バルブにカスタム属性を設定する方法を示しています。オーセンティケーターはヘッダー ID とセッションキーの有無を確認し、ユーザー名とパスワードの値としてセキュリティー層を駆動する JAAS フレームワークに渡します。ユーザー名とパスワードを処理し、サブジェクトに適切なロールを入力できるカスタム JAAS ログインモジュールが必要です。設定された値にヘッダー値が一致しない場合は、通常のフォームベースの認証セマンティクスが適用されます。

カスタムオーセンティケーターの作成

独自のオーセンティケーターの作成については、本書では扱いません。ただし、以下の Java コードはサンプルとして提供されます。

例15.4 GenericHeaderAuthenticator.java

```
/*
 * JBoss, Home of Professional Open Source.
 * Copyright 2006, Red Hat Middleware LLC, and individual contributors
 * as indicated by the @author tags. See the copyright.txt file in the
 * distribution for a full listing of individual contributors.
 *
 * This is free software; you can redistribute it and/or modify it
 * under the terms of the GNU Lesser General Public License as
 * published by the Free Software Foundation; either version 2.1 of
 * the License, or (at your option) any later version.
 *
 * This software is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this software; if not, write to the Free
 * Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA
 * 02110-1301 USA, or see the FSF site: http://www.fsf.org.
 */

package org.jboss.web.tomcat.security;

import java.io.IOException;
import java.security.Principal;
import java.util.StringTokenizer;

import javax.management.JMException;
import javax.management.ObjectName;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import org.apache.catalina.Realm;
import org.apache.catalina.Session;
import org.apache.catalina.authenticator.Constants;
import org.apache.catalina.connector.Request;
import org.apache.catalina.connector.Response;
import org.apache.catalina.deploy.LoginConfig;
import org.jboss.logging.Logger;

import org.jboss.as.web.security.ExtendedFormAuthenticator;

/**
 * JBAS-2283: Provide custom header based authentication support
 *
 * Header Authenticator that deals with userid from the request header Requires
 * two attributes configured on the Tomcat Service - one for the http header
 * denoting the authenticated identity and the other is the SESSION cookie
 *
 * @author <a href="mailto:Anil.Saldhana@jboss.org">Anil Saldhana</a>
 * @author <a href="mailto:sguilhen@redhat.com">Stefan Guilhen</a>
 * @version $Revision$
 * @since Sep 11, 2006
 */
public class GenericHeaderAuthenticator extends ExtendedFormAuthenticator {
  protected static Logger log = Logger
      .getLogger(GenericHeaderAuthenticator.class);

  protected boolean trace = log.isTraceEnabled();

  // JBAS-4804: GenericHeaderAuthenticator injection of ssoid and
  // sessioncookie name.
  private String httpHeaderForSSOAuth = null;

  private String sessionCookieForSSOAuth = null;

  /**
   * <p>
   * Obtain the value of the <code>httpHeaderForSSOAuth</code> attribute. This
   * attribute is used to indicate the request header ids that have to be
   * checked in order to retrieve the SSO identity set by a third party
   * security system.
   * </p>
   *
   * @return a <code>String</code> containing the value of the
   *         <code>httpHeaderForSSOAuth</code> attribute.
   */
  public String getHttpHeaderForSSOAuth() {
    return httpHeaderForSSOAuth;
  }

  /**
   * <p>
   * Set the value of the <code>httpHeaderForSSOAuth</code> attribute. This
   * attribute is used to indicate the request header ids that have to be
   * checked in order to retrieve the SSO identity set by a third party
   * security system.
   * </p>
```

```
     *
     * @param httpHeaderForSSOAuth
     *          a <code>String</code> containing the value of the
     *          <code>httpHeaderForSSOAuth</code> attribute.
     */
    public void setHttpHeaderForSSOAuth(String httpHeaderForSSOAuth) {
      this.httpHeaderForSSOAuth = httpHeaderForSSOAuth;
    }

    /**
     * <p>
     * Obtain the value of the <code>sessionCookieForSSOAuth</code> attribute.
     * This attribute is used to indicate the names of the SSO cookies that may
     * be present in the request object.
     * </p>
     *
     * @return a <code>String</code> containing the names (separated by a
     *       <code>','</code>) of the SSO cookies that may have been set by a
     *       third party security system in the request.
     */
    public String getSessionCookieForSSOAuth() {
      return sessionCookieForSSOAuth;
    }

    /**
     * <p>
     * Set the value of the <code>sessionCookieForSSOAuth</code> attribute. This
     * attribute is used to indicate the names of the SSO cookies that may be
     * present in the request object.
     * </p>
     *
     * @param sessionCookieForSSOAuth
     *          a <code>String</code> containing the names (separated by a
     *          <code>','</code>) of the SSO cookies that may have been set by
     *          a third party security system in the request.
     */
    public void setSessionCookieForSSOAuth(String sessionCookieForSSOAuth) {
      this.sessionCookieForSSOAuth = sessionCookieForSSOAuth;
    }

    /**
     * <p>
     * Creates an instance of <code>GenericHeaderAuthenticator</code>.
     * </p>
     */
    public GenericHeaderAuthenticator() {
      super();
    }

    public boolean authenticate(Request request, HttpServletResponse response,
        LoginConfig config) throws IOException {
      log.trace("Authenticating user");

      Principal principal = request.getUserPrincipal();
      if (principal != null) {
        if (trace)
```

```
      log.trace("Already authenticated '" + principal.getName() + "'");
    return true;
   }

   Realm realm = context.getRealm();
   Session session = request.getSessionInternal(true);

   String username = getUserId(request);
   String password = getSessionCookie(request);

   // Check if there is sso id as well as sessionkey
   if (username == null || password == null) {
    log.trace("Username is null or password(sessionkey) is null:fallback to form auth");
    return super.authenticate(request, response, config);
   }
   principal = realm.authenticate(username, password);

   if (principal == null) {
    forwardToErrorPage(request, response, config);
    return false;
   }

   session.setNote(Constants.SESS_USERNAME_NOTE, username);
   session.setNote(Constants.SESS_PASSWORD_NOTE, password);
   request.setUserPrincipal(principal);

   register(request, response, principal, HttpServletRequest.FORM_AUTH,
      username, password);
   return true;
  }

  /**
   * Get the username from the request header
   *
   * @param request
   * @return
   */
  protected String getUserId(Request request) {
   String ssoid = null;
   // We can have a comma-separated ids
   String ids = "";
   try {
    ids = this.getIdentityHeaderId();
   } catch (JMException e) {
    if (trace)
      log.trace("getUserId exception", e);
   }
   if (ids == null || ids.length() == 0)
    throw new IllegalStateException(
       "Http headers configuration in tomcat service missing");

   StringTokenizer st = new StringTokenizer(ids, ",");
   while (st.hasMoreTokens()) {
    ssoid = request.getHeader(st.nextToken());
    if (ssoid != null)
      break;
```

```
    }
    if (trace)
      log.trace("SSOID-" + ssoid);
    return ssoid;
  }

  /**
   * Obtain the session cookie from the request
   *
   * @param request
   * @return
   */
  protected String getSessionCookie(Request request) {
    Cookie[] cookies = request.getCookies();
    log.trace("Cookies:" + cookies);
    int numCookies = cookies != null ? cookies.length : 0;

    // We can have comma-separated ids
    String ids = "";
    try {
      ids = this.getSessionCookieId();
      log.trace("Session Cookie Ids=" + ids);
    } catch (JMException e) {
      if (trace)
        log.trace("checkSessionCookie exception", e);
    }
    if (ids == null || ids.length() == 0)
      throw new IllegalStateException(
          "Session cookies configuration in tomcat service missing");

    StringTokenizer st = new StringTokenizer(ids, ",");
    while (st.hasMoreTokens()) {
      String cookieToken = st.nextToken();
      String val = getCookieValue(cookies, numCookies, cookieToken);
      if (val != null)
        return val;
    }
    if (trace)
      log.trace("Session Cookie not found");
    return null;
  }

  /**
   * Get the configured header identity id in the tomcat service
   *
   * @return
   * @throws JMException
   */
  protected String getIdentityHeaderId() throws JMException {
    if (this.httpHeaderForSSOAuth != null)
      return this.httpHeaderForSSOAuth;
    return (String) mserver.getAttribute(new ObjectName(
        "jboss.web:service=WebServer"), "HttpHeaderForSSOAuth");
  }

  /**
```

```
    * Get the configured session cookie id in the tomcat service
    *
    * @return
    * @throws JMException
    */
   protected String getSessionCookieId() throws JMException {
    if (this.sessionCookieForSSOAuth != null)
      return this.sessionCookieForSSOAuth;
    return (String) mserver.getAttribute(new ObjectName(
        "jboss.web:service=WebServer"), "SessionCookieForSSOAuth");
   }

   /**
    * Get the value of a cookie if the name matches the token
    *
    * @param cookies
    *          array of cookies
    * @param numCookies
    *          number of cookies in the array
    * @param token
    *          Key
    * @return value of cookie
    */
   protected String getCookieValue(Cookie[] cookies, int numCookies,
      String token) {
    for (int i = 0; i < numCookies; i++) {
      Cookie cookie = cookies[i];
      log.trace("Matching cookieToken:" + token + " with cookie name="
          + cookie.getName());
      if (token.equals(cookie.getName())) {
       if (trace)
        log.trace("Cookie-" + token + " value=" + cookie.getValue());
       return cookie.getValue();
      }
    }
    return null;
   }
  }
```

バグの報告

第16章 マイグレーション

16.1. アプリケーションセキュリティーの変更の設定

Basic 認証のセキュリティー設定

これまでのバージョンの JBoss EAP では、*EAP_HOME*/server/*SERVER_NAME*/conf/ ディレクトリーに置かれたプロパティーファイルはクラスパス上にあり、UsersRolesLoginModule によって簡単に確認できました。JBoss EAP 6 では、ディレクトリー構造が変更になりました。プロパティーファイルは、クラスパスで利用できるようにするためにアプリケーション内にパッケージ化する必要があります。

重要

サーバーの再起動後に変更が維持されるようにするには、サーバーを停止してからサーバー設定ファイルを編集する必要があります。

Basic 認証のセキュリティーを設定するには、security-domains の下に新しいセキュリティードメインを standalone/configuration/standalone.xml または domain/configuration/domain.xml サーバー設定ファイルに追加します。

```
<security-domain name="example">
    <authentication>
        <login-module code="UsersRoles" flag="required">
            <module-option name="usersProperties"
                    value="${jboss.server.config.dir}/example-users.properties"/>
            <module-option name="rolesProperties"
                    value="${jboss.server.config.dir}/example-roles.properties"/>
        </login-module>
    </authentication>
</security-domain>
```

JBoss EAP 6 インスタンスがスタンドアロンサーバーとして実行されている場合、${jboss.server.config.dir} は *EAP_HOME*/standalone/configuration/ ディレクトリーを参照します。インスタンスが管理対象ドメインで実行されている場合は、 ${jboss.server.config.dir} は *EAP_HOME*/domain/configuration/ ディレクトリーを参照します。

セキュリティードメイン名の変更

JBoss EAP 6 では、セキュリティードメインの名前にプレフィックス java:/jaas/ が使用されなくなりました。

- Web アプリケーションの場合、jboss-web.xml のセキュリティードメイン設定からこの接頭辞を削除する必要があります。

- エンタープライズアプリケーションの場合、jboss-ejb3.xml ファイルのセキュリティードメイン設定からこの接頭辞を削除する必要があります。このファイルは JBoss EAP 6 の jboss.xml に置き換えられました。

バグの報告

付録A 参照資料

A.1. 含まれる認証モジュール

以下の認証モジュールは JBoss EAP 6 に含まれています。これらの一部は承認と認証を処理します。これらには通常、Code の名前内に Role という単語が含まれます。

これらのモジュールを設定する場合は、Code 値またはフルネーム（パッケージ修飾）名を使用してモジュールを参照します。

認証モジュール

- 表A.39「AdvancedADLdap」
- 表A.40「UsersRoles」
- 表A.41「UsersRoles モジュールオプション」
- カスタム認証モジュール

表A.1 RealmDirect

コード	**RealmDirect**
クラス	**org.jboss.as.security.RealmDirectLoginModule**
説明	セキュリティーレルムと直接インターフェースするログインモジュール実装。このログインモジュールを使用すると、バッキングストアとのすべての対話をレルムに委譲できます。これにより、複製された同期された定義が必要なくなります。リモーティング呼び出しおよび管理インターフェースに使用されます。

表A.2 RealmDirect モジュールオプション

オプション	Type	デフォルト	説明
realm	string	**ApplicationRealm**	必要なレルムの名前。

表A.3 Client

コード	**Client**
クラス	**org.jboss.security.ClientLoginModule**
説明	このログインモジュールは、JBoss EAP 6 がクライアントとして動作するときに呼び出し元のアイデンティティーおよびクレデンシャルを確立するように設計されています。サーバー認証に使用されるセキュリティードメインの一部として使用しないでください。

表A.4 Client モジュールオプション

オプション	Type	デフォルト	説明
multi-threaded	**true** または **false**	**false**	各スレッドに独自のプリンシパルおよび認証情報ストレージがある場合は true に設定されます。仮想マシンのすべてのスレッドが同じアイデンティティーおよび認証情報を共有することを示すには、false に設定します。
password-stacking	**useFirstPass** or **false**	**false**	**UseFirstPass** に設定して、このログインモジュールがアイデンティティーとして使用する **LoginContext** に保存されている情報を検索することを示します。このオプションは、このログインモジュールと他のログインモジュールをスタックする際に使用できます。
restore-login-identity	**true** または **false**	**false**	**login()** メソッド開始時に表示される ID および認証情報が **logout()** メソッドの呼び出し後に復元される場合は true に設定します。

表A.5 Remoting

コード	**Remoting**
クラス	**org.jboss.as.security.remoting.RemotingLoginModule**
説明	このログインモジュールは、現在認証中の要求が Remoting 接続上で受信された要求であるかどうかを確認するために使用されます。Remoting 接続上で受信された要求である場合、Remoting 認証プロセス中に作成されたアイデンティティーが使用され、現在の要求に関連付けられます。リクエストが Remoting 接続に到達しなかった場合、このモジュールは何もせず、JAAS ベースのログインが次のモジュールに続行できるようにします。

表A.6 Remoting モジュールオプション

オプション	Type	デフォルト	説明

オプション	Type	デフォルト	説明
password-stacking	**useFirstPass** or **false**	**false**	**useFirstPass** の値は、このログインモジュールが、最初にアイデンティティーの **LoginContext** に保存されている情報を検索することを示しています。このオプションは、このログインモジュールと他のログインモジュールをスタックする際に使用できます。
principalClass	完全修飾クラス名	none	プリンシパル名に String 引数を取るコンストラクターが含まれる **Principal** 実装クラス。
unauthenticatedIdentity	プリンシパル名。	none	認証情報を含まない要求に割り当てられるプリンシパル名を定義します。これを使用すると、保護されていないサーブレットは特定ロールを必要としない EJB でメソッドを呼び出すことができます。このようなプリンシパルには関連付けられたロールがなく、**unchecked permission** 制約に関連付けられたセキュアでない EJB または EJB メソッドのみにアクセスできます。

表A.7 Certificate

コード	**Certificate**
クラス	**org.jboss.security.auth.spi.BaseCertLoginModule**
説明	このログインモジュールは、**X509 Certificates** に基づいてユーザーを認証するように設計されています。このユースケースは、web アプリケーションの **CLIENT-CERT** 認証です。

表A.8 Certificate モジュールオプション

オプション	Type	デフォルト	説明

オプション	Type	デフォルト	説明
securityDomain	string	**その他**	信頼できる証明書を保持するトラストストアの JSSE 設定を持つセキュリティードメインの名前。
verifier	class	none	ログイン証明書の検証に使用する **org.jboss.security.auth.certs.X509CertificateVerifier** のクラス名。

表A.9 CertificateRoles

コード	**CertificateRoles**
クラス	**org.jboss.security.auth.spi.CertRolesLoginModule**
説明	このログインモジュールは、Certificate ログインモジュールを拡張して、プロパティーファイルからロールマッピング機能を追加します。Certificate ログインモジュールと同じオプションをすべて取得し、以下のオプションを追加します。

表A.10 CertificateRoles モジュールオプション

オプション	Type	デフォルト	説明
rolesProperties	string	**roles.properties**	各ユーザーに割り当てるロールを含むリソースまたはファイルの名前です。ロールプロパティーファイルの形式は **username=role1,role2** である必要があります。username は証明書の DN、任意の **=**（等しい）およびスペース文字をエスケープします。正しい形式の例を以下に示します。 `CN\=unit-tests-client,\ OU\=Red\ Hat\ Inc.,\ O\=Red\ Hat\ Inc.,\ ST\=North\ Carolina,\ C\=US`

オプション	Type	デフォルト	説明
defaultRolesProperties	string	**defaultRoles.properties**	**rolesProperties** ファイルが見つからない場合にフォールバックするリソースまたはファイルの名前。
roleGroupSeparator	1 文字	**.**（1 ピリオド）	**rolesProperties** ファイルでロールグループセパレーターとして使用する文字。

表A.11 Database

コード	**Database**
クラス	**org.jboss.security.auth.spi.DatabaseServerLoginModule**
説明	認証およびロールマッピングをサポートする JDBC ベースのログインモジュール。これは、以下の定義を持つ 2 つの論理テーブルに基づいています。 ● **Principals: PrincipalID (text), Password (text)** ● **Roles: PrincipalID (text), Role (text), RoleGroup (text)**

表A.12 Database モジュールオプション

オプション	Type	デフォルト	説明
digestCallback	完全修飾クラス名	none	入力パスワードをハッシュ化するソルトなどのプレ/ポストダイジェストコンテンツを含む **DigestCallback** 実装のクラス名。**hashAlgorithm** が指定されている場合にのみ使用されます。
dsJndiName	JNDI リソース	**java:/DefaultDS**	認証情報を格納している JNDI リソースの名前。このオプションは必須です。
hashAlgorithm	文字列	プレーンパスワードを使用	パスワードをハッシュ化するために使用されるメッセージダイジェストアルゴリズム。サポートされるアルゴリズムは Java セキュリティープロバイダーによって異なりますが、**MD5**、**SHA-1**、および **SHA- 256** がサポートされます。
hashCharset	文字列	プラットフォームのデフォルトのエンコーディング	パスワードをバイトアレイに変換するときに使用する文字セット/エンコーディングの名前。これには、サポートされるすべての Java 文字セット名が含まれます。

オプション	Type	デフォルト	説明
hashEncoding	文字列	Base64	使用する文字列エンコーディング形式。
ignorePasswordCase	boolean	false	パスワードの比較で大文字と小文字を無視するかどうかを示すフラグ。
inputValidator	完全修飾クラス名	none	クライアントによって提供されるユーザー名およびパスワードを検証するために使用される InputValidator 実装のインスタンス。
principalsQuery	準備済み SQL ステートメント	**Principals から Password from PrincipalID=? を選択します。**	プリンシパルに関する情報を取得するための準備済み SQL クエリー。
rolesQuery	準備済み SQL ステートメント	none	ロールに関する情報を取得するための準備済み SQL クエリー。これは、**Role、RoleGroup from Roles where PrincipalID=? を選択** します。ここで、Role はロール名で、RoleGroup 列の値は常に大文字の **R** または **CallerPrincipal** を持つ **Roles** である必要があります。
storeDigestCallback	完全修飾クラス名	none	ストア/予想されるパスワードをハッシュ化するソルトなどのプレまたはポストダイジェストコンテンツを含む **DigestCallback** 実装のクラス名。**hashStorePassword** または **hashUserPassword** が **true** で、**hashAlgorithm** が指定されている場合にのみ使用されます。
suspendResume	boolean	true	データベースの操作中に既存の JTA トランザクションを一時停止するかどうか。
throwValidatorError	boolean	false	検証エラーがクライアントへ公開されるべきかどうかを示すフラグ。
transactionManagerJndiName	JNDI リソース	java:/TransactionManager	ログインモジュールによって使用されるトランザクションマネージャーの JNDI 名。

表A.13 DatabaseCertificate

コード	**DatabaseCertificate**
クラス	**org.jboss.security.auth.spi.DatabaseCertLoginModule**

説明	このログインモジュールは Certificate ログインモジュールを拡張して、データベーステーブルからロールマッピング機能を追加します。これには、同じオプションと、以下の追加オプションがあります。

表A.14 DatabaseCertificate モジュールオプション

オプション	Type	デフォルト	説明
dsJndiName	JNDI リソース	**java:/DefaultDS**	認証情報を格納している JNDI リソースの名前。このオプションは必須です。
rolesQuery	準備済み SQL ステートメント	**select Role,RoleGroup from Roles where PrincipalID=?**	ロールをマッピングするために実行される SQL の準備済みステートメント。これは、**Role、RoleGroup from Roles where PrincipalID=?** と同等である必要があります。ここで、Role はロール名で、RoleGroup 列の値は常に大文字の **R** または **CallerPrincipal** を持つ **Roles** である必要があります。
suspendResume	**true** または **false**	**true**	データベースの操作中に既存の JTA トランザクションを一時停止するかどうか。

表A.15 Identity

コード	**Identity**
クラス	**org.jboss.security.auth.spi.IdentityLoginModule**
説明	モジュールオプションで指定されたプリンシパルをモジュールに対して認証されたサブジェクトに関連付けます。使用される Principal クラスのタイプは **org.jboss.security.SimplePrincipal** です。プリンシパルオプションが指定されていない場合は、**guest** の名前を持つプリンシパルが使用されます。

表A.16 Identity モジュールオプション

オプション	Type	デフォルト	説明
principal	文字列	**guest**	プリンシパルに使用する名前。
roles	カンマ区切りの文字列リスト	none	サブジェクトに割り当てられるロールのカンマ区切りの一覧。

表A.17 Ldap

コード	**Ldap**
クラス	**org.jboss.security.auth.spi.LdapLoginModule**
説明	JNDI LDAP プロバイダーを使用してアクセス可能な LDAP サーバーにユーザー名とパスワードを保存する際に、LDAP サーバーに対して認証を行います。オプションの多くは LDAP プロバイダーまたは環境によって決定されるため、必須ではありません。

表A.18 Ldap モジュールオプション

オプション	Type	デフォルト	説明
java.naming.factory.initial	クラス名	**com.sun.jndi.ldap.LdapCtxFactory**	**InitialContextFactory** 実装クラス名。
java.naming.provider.url	**ldap://** URL	**java.naming.security.protocol** の値が **SSL**、**ldap://localhost:636** の場合は になります。**ldap://localhost:389**	LDAP サーバーの URL。
java.naming.security.authentication	**SASL** メカニズムの名前なし、**シンプル**、または SASL メカニズムの名前	**simple**	LDAP サーバーにバインドするために使用するセキュリティーレベル。
java.naming.security.protocol	トランスポートプロトコル	指定されていない場合、プロバイダーによって決定されます。	SSL や TLS などのセキュアなアクセスに使用するトランスポートプロトコル。
java.naming.security.principal	string	none	サービスへの呼び出し元を認証するためのプリンシパルの名前。これは、以下で説明されているその他のプロパティーから構築されます。

オプション	Type	デフォルト	説明
java.naming.security.credentials	認証情報のタイプ	none	認証スキームによって使用される認証情報のタイプ。ハッシュ化されたパスワード、クリアテキストのパスワード、キー、証明書などが例となります。このプロパティーが指定されていない場合、この動作はサービスプロバイダーによって決定されます。
principalDNPrefix	string		ユーザー DN を形成するユーザー名に追加されるプレフィックス。ユーザーに対しユーザー名を要求し、**principalDNPrefix** および **principalDNSuffix** を使用して完全修飾 DN を構築できます。
principalDNSuffix	string		ユーザー DN を形成するユーザー名に追加されるサフィックス。ユーザーに対しユーザー名を要求し、**principalDNPrefix** および **principalDNSuffix** を使用して完全修飾 DN を構築できます。
useObjectCredential	**true** または **false**	false	JAAS PasswordCallback を使用した **char[]** パスワードではなく Callback の **org.jboss.security.auth.callback.ObjectCallback** タイプを使用して、不透明なオブジェクトとしてクレデンシャルを取得するかどうか。これにより、**非char[]** 認証情報情報を LDAP サーバーに渡すことができます。
rolesCtxDN	完全修飾 DN	none	ユーザーロールを検索するコンテキストの完全修飾 DN。

オプション	Type	デフォルト	説明
userRolesCtxDNAttributeName	属性	none	ユーザーロールを検索するコンテキストの DN を含むユーザーオブジェクトの属性です。ユーザーのロールを検索するコンテキストはユーザーごとに一意となる可能性がある点で、**rolesCtxDN** とは異なります。
roleAttributeID	属性	**roles**	ユーザーロールを含む属性の名前。
roleAttributeIsDN	**true** または **false**	**false**	**roleAttributeID** にロールオブジェクトの完全修飾 DN が含まれるかどうか。false の場合、ロール名はコンテキスト名の **roleNameAttributeId** 属性の値から取得されます。Microsoft Active Directory などの特定のディレクトリースキーマでは、この属性を **true** に設定する必要があります。
roleNameAttributeID	属性	**name**	ロール名を含む **roleCtxDN** コンテキスト内の属性の名前。**roleAttributeIsDN** プロパティーが **true** に設定されている場合、このプロパティーはロールオブジェクトの名前属性の検索に使用されます。
uidAttributeID	属性	**uid**	ユーザー ID に対応する **UserRolesAttributeDN** の属性の名前。これは、ユーザーロールの特定に使用されます。
matchOnUserDN	**true** または **false**	**false**	ユーザーロールの検索がユーザーの完全識別 DN で一致するか、またはユーザー名のみに一致するかどうか。**true** の場合、完全なユーザー DN が match 値として使用されます。**false** の場合、**uidAttributeName** 属性に対する一致値としてユーザー名のみが使用されます。

オプション	Type	デフォルト	説明
allowEmptyPasswords	**true** または **false**	**false**	空のパスワードを許可するかどうか。ほとんどの LDAP サーバーは、空のパスワードを匿名ログイン試行として処理します。空のパスワードを拒否するには、これを **false** に設定します。

表A.19 LdapExtended

コード	**LdapExtended**
クラス	**org.jboss.security.auth.spi.LdapExtLoginModule**
説明	検索を使用してバインドユーザーと関連のロールを検索する別の LDAP ログインモジュール実装。ロールは再帰的にクエリーを行い、DN に従って階層的なロール構造を移動します。これは Ldap モジュールと同じ **java.naming** オプションを使用し、Ldap モジュールの他のオプションの代わりに以下のオプションを使用します。 認証は 2 つの手順で行われます。 1. LDAP サーバーへの最初のバインドは、bindDN および **bindCredential** オプションを使用して行われます。**bindDN** は、ユーザーとロールの baseCtxDN ツリーと **rolesCtxDN** ツリーの両方を検索できる LDAP ユーザーです。認証するユーザー DN は、**baseFilter** 属性で指定されたフィルターを使用してクエリーされます。 2. ユーザー DN を **InitialLdapContext** 環境 **Context.SECURITY_PRINCIPAL** として使用して、生成されるユーザー DN は LDAP サーバーにバインドされ、認証されます。**Context.SECURITY_CREDENTIALS** プロパティーは、コールバックハンドラーが取得した String パスワードに設定されます。

表A.20 LdapExtended モジュールオプション

オプション	Type	デフォルト	説明
baseCtxDN	完全修飾 DN	none	ユーザーの検索を開始するため、トップレベルのコンテキストの固定 DN です。

オプション	Type	デフォルト	説明
bindCredential	文字列、オプションで暗号化	none	詳細は、『JBoss EAP の『アプリケーションセキュリティーガイド』』を参照してください。
bindDN	完全修飾 DN	none	ユーザーおよびロールクエリーの LDAP サーバーに対してバインドするために使用される DN です。この DN には、**baseCtxDN** および **rolesCtxDN** の値に対する読み取りおよび検索パーミッションが必要です。
baseFilter	LDAP フィルター文字列	none	認証するユーザーのコンテキストを見つけるために使用される検索フィルター。**{0}** 式を使用しているフィルターに、入力ユーザー名またはログインモジュールコールバックから取得した **userDN** が置換されます。検索フィルターの一般的な例は **(uid={0})** です。
rolesCtxDN	完全修飾 DN	none	ユーザーロールを検索するためのコンテキストの固定 DN です。これは、実際のロールがである DN ではなく、ユーザーロールを含むオブジェクトがある DN です。たとえば、Microsoft Active Directory サーバーでは、これは、ユーザーアカウントが存在する DN です。

オプション	Type	デフォルト	説明
roleFilter	LDAP フィルター文字列	none	認証済みユーザーと関連付けられたロールを検索するために使用される検索フィルター。**{0}** 式を使用しているフィルターに、入力ユーザー名またはログインモジュールコールバックから取得した **userDN** が置換されます。**{1}** が使用されると、認証された **userDN** がフィルターに置き換わります。入力ユーザー名に一致する検索フィルター例は **(member={0})** です。認証済み **userDN** に一致する他の例は **(member={1})** です。
roleAttributeIsDN	**true** または **false**	**false**	**roleAttributeID** にロールオブジェクトの完全修飾 DN が含まれるかどうか。false の場合、ロール名はコンテキスト名の **roleNameAttributeId** 属性の値から取得されます。Microsoft Active Directory などの特定のディレクトリースキーマでは、この属性を **true** に設定する必要があります。
defaultRole	ロール名	none	認証された全ユーザーに対して含まれるロール
parseRoleNameFromDN	**true** または、以下を実行します。 **false**	**false**	クエリーによって返された DN に roleNameAttributeID が含まれるかどうかを示すフラグ。**true** に設定すると、DN は roleNameATtributeID に対してチェックされます。**false** に設定すると、DN は roleNameATtributeID に対してチェックされません。このフラグは LDAP クエリーのパフォーマンスを向上できます。

オプション	Type	デフォルト	説明
parseUsername	**true** または、以下を実行します。 **false**	**false**	DN がユーザー名に対して解析されるかどうかを示すフラグ。**true** に設定すると、DN はユーザー名に対して解析されます。**false** に設定すると、DN はユーザー名に対して解析されません。このオプションは、usernameBeginString および usernameEndString とともに使用されます。
usernameBeginString	string	none	ユーザー名を公開するため、DN の最初から削除される文字列を定義します。このオプションは、**usernameEndString** と併用されます。
usernameEndString	string	none	ユーザー名を公開するため、DN の最後から削除される文字列を定義します。このオプションは、**usernameBeginString** と併用されます。
roleNameAttributeID	属性	**name**	ロール名を含む **roleCtxDN** コンテキスト内の属性の名前。**roleAttributeIsDN** プロパティーが **true** に設定されている場合、このプロパティーはロールオブジェクトの名前属性の検索に使用されます。
distinguishedNameAttribute	属性	**distinguishedName**	ユーザーの DN を含むユーザーエントリーの属性の名前。これは、ユーザー自身の DN に正しいユーザーマッピングを妨げる特殊文字（バックスラッシュなど）が含まれる場合に必要になることがあります。属性が存在しない場合は、エントリーの DN が使用されます。
roleRecursion	integer	**0**	ロール検索の再帰レベルの数は、一致するコンテキストの下に続きます。再帰を無効にするには、これを **0** に設定します。

オプション	Type	デフォルト	説明
searchTimeLimit	integer	**10000** (10 秒)	ユーザーまたはロールの検索のタイムアウト (ミリ秒単位)。
searchScope	**OBJECT_SCOPE**、**ONELEVEL_SCOPE**、**SUBTREE_SCOPE**のいずれか。	**SUBTREE_SCOPE**	使用する検索範囲。
allowEmptyPasswords	**true** または **false**	**false**	空のパスワードを許可するかどうか。ほとんどの LDAP サーバーは、空のパスワードを匿名ログイン試行として処理します。空のパスワードを拒否するには、これを false に設定します。
referralUserAttributeIDToCheck	属性	none	紹介を使用しない場合、このオプションは無視することができます。リファーラルを使用し、ロールオブジェクトがリファーラル内部にある場合、このオプションは特定のロール（例：**member**）に対して定義されたユーザーが含まれる属性名を示します。ユーザーはこの属性名の内容に対して確認されます。このオプションが設定されていないとチェックは常に失敗するため、ロールオブジェクトはリファーラルツリーに保存できません。

表A.21 RoleMapping

コード	**RoleMapping**
クラス	**org.jboss.security.auth.spi.RoleMappingLoginModule**
説明	認証プロセスの最終結果であるロールを宣言型ロールにマッピングします。このモジュールは、セキュリティードメインに追加する際に、**任意** としてフラグ付けする必要があります。

表A.22 RoleMapping モジュールオプション

オプション	Type	デフォルト	説明
rolesProperties	プロパティーファイルまたはリソースの完全修飾ファイルパスまたは完全修飾ファイル名。	**none**	ロールを置換ロールにマップするプロパティーファイルまたはリソースの完全修飾ファイルパスまたはファイル名。フォーマットは、以下のようになります。**original_role=role1,role2,role3**
replaceRole	**true** または **false**	**false**	現在のロールを追加するか、マップされたロールに現在のロールを置き換えるか。**true** に設定した場合は、置き換えられます。

備考

rolesProperties モジュールオプションは RoleMapping に必要です。

表A.23 RunAs

コード	**RunAs**
クラス	**org.jboss.security.auth.spi.RunAsLoginModule**
説明	認証のログインフェーズの間に **run as** ロールをスタックにプッシュするヘルパーモジュール。コミットまたはアボートフェーズで **run as** ロールをスタックにポップします。このログインモジュールは、セキュアな EJB にアクセスするログインモジュールなど、セキュアなリソースにアクセスするためにセキュアなリソースにアクセスする必要のある他のログインモジュールのロールを提供します。**RunAsLoginModule run as** ロールを確立する必要があるログインモジュールの前に設定する必要があります。

表A.24 RunAs オプション

オプション	Type	デフォルト	説明
roleName	ロール名	**nobody**	ログインフェーズで **run as** ロールとして使用するロールの名前。

オプション	Type	デフォルト	説明
principalName	プリンシパル名	**nobody**	ログインフェーズで **run as** プリンシパルとして使用するプリンシパルの名前。指定されていない場合は、デフォルトの **nobody** が使用されます。
principalClass	完全修飾クラス名	none	プリンシパル名に String 引数を取るコンストラクターが含まれる **Principal** 実装クラス。

表A.25 Simple

コード	**Simple**
クラス	**org.jboss.security.auth.spi.SimpleServerLoginModule**
説明	テスト目的でセキュリティーを素早くセットアップするためのモジュール。以下の単純なアルゴリズムを実装します。 ● パスワードが null の場合、ユーザーを認証し **guest** のアイデンティティーと **guest** のロールを割り当てます。 ● パスワードがユーザーと同じ場合は、ユーザー名と **admin** と **guest** の両ロールに等しいアイデンティティーを割り当てます。 ● そうしないと、認証に失敗します。

Simple モジュールオプション

Simple モジュールにはオプションがありません。

表A.26 ConfiguredIdentity

コード	**ConfiguredIdentity**
クラス	**org.picketbox.datasource.security.ConfiguredIdentityLoginModule**
説明	モジュールオプションで指定されたプリンシパルをモジュールに対して認証されたサブジェクトに関連付けます。使用される Principal クラスのタイプは **org.jboss.security.SimplePrincipal** です。

表A.27 ConfiguredIdentity モジュールオプション

オプション	Type	デフォルト	説明
username	string	none	認証のユーザー名
password	暗号化文字列	""	認証に使用するパスワード。パスワードを暗号化するには、コマンドラインで直接モジュールを使用します。 java org.picketbox.datasource.security.SecureIdentityLoginModule *password_to_encrypt* このコマンドの結果をモジュールオプションの値フィールドに貼り付けます。デフォルト値は空の文字列です。
principal	プリンシパルの名前	**none**	モジュールに対して認証されたサブジェクトに関連付けられるプリンシパル。

表A.28 SecureIdentity

コード	**SecureIdentity**
クラス	**org.picketbox.datasource.security.SecureIdentityLoginModule**
説明	このモジュールはレガシー目的のために提供されます。これにより、パスワードを暗号化してから、暗号化されたパスワードを静的プリンシパルで使用できます。アプリケーションが **SecureIdentity** を使用する場合は、パスワード vault メカニズムを代わりに使用することを検討してください。

表A.29 SecureIdentity モジュールオプション

オプション	Type	デフォルト	説明
username	string	none	認証のユーザー名

オプション	Type	デフォルト	説明
password	暗号化文字列	""	認証に使用するパスワード。パスワードを暗号化するには、コマンドラインで直接モジュールを使用します。 `java org.picketbox.datasource.security.SecureIdentityLoginModule password_to_encrypt` このコマンドの結果をモジュールオプションの値フィールドに貼り付けます。デフォルト値は空の文字列です。
managedConnectionFactoryName	JCA リソース	none	データソースの JCA 接続ファクトリーの名前。

表A.30 PropertiesUsers

コード	**PropertiesUsers**
クラス	**org.jboss.security.auth.spi.PropertiesUsersLoginModule**
説明	プロパティーファイルを使用して、認証用のユーザー名とパスワードを保存します。承認（ロールマッピング）は提供されません。このモジュールは、テストにのみ適しています。

表A.31 SimpleUsers

コード	**SimpleUsers**
クラス	**org.jboss.security.auth.spi.SimpleUsersLoginModule**
説明	このログインモジュールは、***module-option*** を使用してユーザー名とクリアテキストのパスワードを保存します。***module-option*** 's***name*** and ***value*** 属性は、ユーザー名とパスワードを指定します。これはテスト用にのみ含まれており、実稼働環境には適していません。

表A.32 LdapUsers

コード	**LdapUsers**
クラス	**org.jboss.security.auth.spi.LdapUsersLoginModule**
説明	**LdapUsers** モジュールは、**ExtendedLDAP** モジュールおよび **AdvancedLdap** モジュールに置き換えられました。

表A.33 Kerberos

コード	**Kerberos**
クラス	**com.sun.security.auth.module.Krb5LoginModule**.IBM JDK では、クラス名は **com.ibm.security.auth.module.Krb5LoginModule** です。
説明	GSSAPI を使用して Kerberos ログイン認証を実行します。このモジュールは、Sun Microsystems によって提供される API のセキュリティーフレームワークの一部です。詳細は、を参照してください http://docs.oracle.com/javase/7/docs/jre/api/security/jaas/spec/com/sun/security/auth/module/Krb5LoginModule.html。このモジュールは、認証およびロールマッピングを処理する別のモジュールとペアにする必要があります。

表A.34 Kerberos モジュールオプション

オプション	Type	デフォルト	説明
storekey	**true** または **false**	false	**KerberosKey** をサブジェクトのプライベート認証情報に追加するかどうか。
doNotPrompt	**true** または **false**	false	**true** に設定すると、キャッシュ、キータブ、または共有状態から認証情報を取得できない場合、ユーザーはパスワードを要求されません。
useTicketCache	**true** または **false**のブール値 .	false	**true** の場合、TGT はチケットキャッシュから取得されます。**false** の場合、チケットキャッシュは使用されません。

オプション	Type	デフォルト	説明
ticketcache	Kerberos チケットキャッシュを表すファイルまたはリソース。これが設定され ***ている場合は、UseTicketCache*** も **true** に設定する必要があります。設定エラーが返されます。	デフォルトは使用するオペレーティングシステムによって異なります。 ● Red Hat Enterprise Linux / Solaris: **/tmp/krb5cc_*uid***（オペレーティングシステムの UID 数値を使用）。 ● Microsoft Windows Server の場合: Local Security Authority (LSA) API を使用してチケットキャッシュを見つけます。	チケットキャッシュの場所。
useKeyTab	**true** または **false**	false	キーテーブルファイルからプリンシパルのキーを取得するかどうか。
keytab	Kerberos keytab を表すファイルまたはリソース。	オペレーティングシステムの Kerberos 設定ファイルの場所、または **/home/*ユーザー*/krb5.keytab**の場所	キーテーブルファイルの場所。
principal	string	none	プリンシパルの名前。これは、単純なユーザー名または **host/testserver.acme.com** などのサービス名のいずれかになります。キーテーブルからプリンシパルを取得する場合、またはキーテーブルに複数のプリンシパルが含まれる場合には、このパラメーターを使用します。

オプション	Type	デフォルト	説明
useFirstPass	**true** または **false**	false	**javax.security.auth.login.name** および **javax.security.auth.login.password** をキーとして使用して、モジュールの共有状態からユーザー名とパスワードを取得するかどうか。認証に失敗すると、再試行は行われません。
tryFirstPass	**true** または **false**	false	**useFirstPass** と同じですが、認証に失敗した場合、モジュールは **CallbackHandler** を使用して新しいユーザー名とパスワードを取得します。2 つ目の認証に失敗すると、失敗は呼び出し元アプリケーションに報告されます。
storePass	**true** または **false**	false	ユーザー名とパスワードをモジュールの共有状態に保存するかどうか。これは、鍵が共有状態にあるか、または認証に失敗した場合に発生しません。
clearPass	**true** または **false**	false	認証の各フェーズが完了した後に、共有状態からユーザー名とパスワードを消去するには、**true** に設定します。

表A.35 SPNEGO

コード	**SPNEGO**
クラス	**org.jboss.security.negotiation.spnego.SPNEGOLoginModule**
説明	Microsoft Active Directory サーバーまたは SPNEGO をサポートするその他の環境への SPNEGO 認証を許可します。SPNEGO には Kerberos 認証情報を保持することもできます。このモジュールは、認証とロールマッピングを処理する別のモジュールとペアにする必要があります。

表A.36 SPNEGO モジュールオプション

オプション	Type	デフォルト	説明
serverSecurityDomain	**string**	**Null**.	Kerberos ログインモジュールを介してサーバーサービスの ID を取得するために使用されるドメインを定義します。このプロパティーを設定する必要があります。
removeRealmFromPrincipal	**boolean**	**false**	さらなる処理を行う前に Kerberos レルムをプリンシパルから削除する必要があることを指定します。
usernamePasswordDomain	**string**	**null**	Kerberos が失敗した場合にフェイルオーバーログインとして使用する必要のある設定内の別のセキュリティードメインを指定します。

表A.37 AdvancedLdap

コード	**AdvancedLdap**
クラス	**org.jboss.security.negotiation.AdvancedLdapLoginModule**
説明	SASL や JAAS セキュリティードメインの使用など、追加機能を提供するモジュール。

表A.38 AdvancedLdap モジュールオプション

オプション	Type	デフォルト	説明
bindAuthentication	string	none	ディレクトリーサーバーへのバインドに使用する SASL 認証のタイプ。
java.naming.provider.url	**string**	**java.naming.security.protocol** の値が **SSL**、**ldap://localhost:686** の場合は になります。**ldap://localhost:389**	ディレクトリーサーバーの URI。
baseCtxDN	完全修飾 DN	none	検索のベースとして使用する識別名。
baseFilter	LDAP 検索フィルターを表す文字列。	none	検索結果を絞り込むために使用するフィルター。

オプション	Type	デフォルト	説明
roleAttributeID	LDAP 属性を表す文字列値。	none	承認ロールの名前が含まれる LDAP 属性です。
roleAttributeIsDN	**true** または **false**	**false**	ロール属性が識別名 (DN) であるかどうか。
roleNameAttributeID	LDAP 属性を表す文字列。	none	実際のロール属性が含まれる **RoleAttributeId** 内に含まれる属性。
recurseRoles	**true** または **false**	**false**	**RoleAttributeId** でロールを再帰的に検索するかどうか。
referralUserAttributeIDToCheck	属性	none	紹介を使用しない場合、このオプションは無視することができます。リファーラルを使用し、ロールオブジェクトがリファーラル内部にある場合、このオプションは特定のロール（例：**member**）に対して定義されたユーザーが含まれる属性名を示します。ユーザーはこの属性名の内容に対して確認されます。このオプションが設定されていないとチェックは常に失敗するため、ロールオブジェクトはリファーラルツリーに保存できません。

表A.39 AdvancedADLdap

コード	**AdvancedADLdap**
クラス	**org.jboss.security.negotiation.AdvancedADLoginModule**
説明	このモジュールは **AdvancedLdap** ログインモジュールを拡張し、Microsoft Active Directory に関連する追加のパラメーターを追加します。

表A.40 UsersRoles

コード	**UsersRoles**
クラス	**org.jboss.security.auth.spi.UsersRolesLoginModul**

説明	2 つの異なるプロパティーファイルに格納された複数のユーザーおよびユーザーロールをサポートする簡単なログインモジュール。

表A.41 UsersRoles モジュールオプション

オプション	Type	デフォルト	説明
usersProperties	ファイルまたはリソースへのパス。	**users.properties**	ユーザー/パスワード間のマッピングが含まれるファイルまたはリソースです。ファイルの形式は ***username=password*** です。
rolesProperties	ファイルまたはリソースへのパス。	**roles.properties**	ユーザー/ ロール間のマッピングが含まれるファイルまたはリソースです。ファイルの形式は、以下のようになります。 **username=role1,role2,role3**
password-stacking	**useFirstPass** or **false**	**false**	**useFirstPass** の値は、このログインモジュールが ID の **LoginContext** に保存されている情報を参照することを示します。このオプションは、このログインモジュールと他のログインモジュールをスタックする際に使用できます。

オプション	Type	デフォルト	説明
hashAlgorithm	パスワードハッシュアルゴリズムを表す文字列。	**none**	パスワードのハッシュ化に使用する **java.security.MessageDigest** アルゴリズムの名前。デフォルトがないため、ハッシュを有効にするには、このオプションを明示的に設定する必要があります。**hashAlgorithm** を指定すると、**CallbackHandler** から取得したクリアテキストのパスワードが **UsernamePasswordLoginModule.validatePassword** 引数として **inputPassword** に渡される前にハッシュ化されます。**users.properties** ファイルに保存されているパスワードは正確にハッシュ化する必要があります。
hashEncoding	**base64** または **hex**	**base64**	hashAlgorithm も設定されている場合、ハッシュ化されたパスワードの文字列形式。
hashCharset	string	コンテナーのランタイム環境に設定されるデフォルトのエンコーディング	クリアテキストのパスワードをバイトアレイに変換するために使用されるエンコーディング。
unauthenticatedIdentity	プリンシパル名	none	認証情報を含まない要求に割り当てられるプリンシパル名を定義します。これを使用すると、保護されていないサーブレットは特定ロールを必要としない EJB でメソッドを呼び出すことができます。このようなプリンシパルには関連するロールがなく、**チェックされていないパーミッション** 制約に関連付けられたセキュアでない EJB または EJB メソッドのみにアクセスできます。

カスタム認証モジュール

認証モジュールは、javax.security.auth.spi.LoginModule の実装です。カスタム認証モジュールの作成に関する詳細は、API ドキュメントを参照してください。

バグの報告

A.2. 含まれる承認モジュール

以下のモジュールは承認サービスを提供します。

コード	クラス
DenyAll	org.jboss.security.authorization.modules.AllDenyAuthorizationModule
PermitAll	org.jboss.security.authorization.modules.AllPermitAuthorizationModule
Delegating	org.jboss.security.authorization.modules.DelegatingAuthorizationModule
web	org.jboss.security.authorization.modules.web.WebAuthorizationModule
JACC	org.jboss.security.authorization.modules.JACCAuthorizationModule
XACML	org.jboss.security.authorization.modules.XACMLAuthorizationModule

AllDenyAuthorizationModule

これは、承認要求を常に拒否する簡単な承認モジュールです。設定オプションは利用できません。

AllPermitAuthorizationModule

これは、常に承認要求を許可する簡単な承認モジュールです。設定オプションは利用できません。

DelegatingAuthorizationModule

意思決定を設定済みの delegate へ移譲するデフォルトの認証モジュールです。

WebAuthorizationModule

デフォルトの Tomcat 承認ロジック (permit all) を持つデフォルトの Web 認証モジュールです。

JACCAuthorizationModule

このモジュールは 2 つの delegate を使用して JACC セマンティクスを強制します（Web コンテナー承認リクエストの WebJACCPolicyModuleDelegate と、EJB コンテナーリクエストの EJBJACCPolicyModuleDelegate）。利用可能な設定オプションはありません。

XACMLAuthorizationModule

このモジュールは web および EJB コンテナーの 2 つの委譲（WebXACMLPolicyModuleDelegate および EJBXACMLPolicyModuleDelegate）を使用して XACML 承認を強制します。登録したポリシーに基づいて PDP オブジェクトを作成し、それに対して web または EJB リクエストを評価します。

AbstractAuthorizationModule

これは、上書きが必要なベース承認モジュールで、他の認可モジュールへ委任する機能を提供します。

バグの報告

A.3. 含まれるセキュリティーマッピングモジュール

以下のセキュリティーマッピングロールが JBoss EAP 6 で提供されます。

コード	クラス
PropertiesRoles	**org.jboss.security.mapping.providers.role.PropertiesRolesMappingProvider**
SimpleRoles	**org.jboss.security.mapping.providers.role.SimpleRolesMappingProvider**
DeploymentRoles	**org.jboss.security.mapping.providers.DeploymentRolesMappingProvider**
DatabaseRoles	**org.jboss.security.mapping.providers.role.DatabaseRolesMappingProvider**
LdapRoles	**org.jboss.security.mapping.providers.role.LdapRolesMappingProvider**

コード	クラス
LdapAttributes	**org.jboss.security.mapping.providers.attribute.LdapAttributeMappingProvider**

DeploymentRolesMappingProvider

jboss-web.xml および jboss-app.xml デプロイメント記述子で実行できるプリンシパルからロールへのマッピングを考慮するロールマッピングモジュール。

例A.1 例

```
<jboss-web>
...
  <security-role>
     <role-name>Support</role-name>
     <principal-name>Mark</principal-name>
     <principal-name>Tom</principal-name>
  </security-role>
...
</jboss-web>
```

org.jboss.security.mapping.providers.DeploymentRoleToRolesMappingProvider

jboss-web.xml および jboss-app.xml デプロイメント記述子で実行可能なプリンシパルからロールへのマッピングを考慮する Role to Roles マッピングモジュール。この場合、principal-name は他のロールをマップするロールを示します。

例A.2 例

```
 <jboss-web>
...
   <security-role>
     <role-name>Employee</role-name>
     <principal-name>Support</principal-name>
     <principal-name>Sales</principal-name>
   </security-role>
...
 </jboss-web>
```

これは、Support または Sales をロールに持つ各プリンシパルには Employee ロールも割り当てられることを意味します。

org.jboss.security.mapping.providers.OptionsRoleMappingProvider

オプションからロールを選択し、渡されたグループにロールを追加するロールマッピングプロバイダー。ロール名（キー）のプロパティースタイルのマッピングを、ロール（値）のコンマ区切りリストで取ります。

org.jboss.security.mapping.providers.principal.SimplePrincipalMappingProvider

SimplePrincipal を取り、異なるプリンシパル名で SimplePrincipal を変換するプリンシパルマッピングプロバイダー。

DatabaseRolesMappingProvider

データベースからロールを読み取る MappingProvider。

オプション:

- dsJndiName: ロールをユーザーにマップするために使用されるデータソースの JNDI 名。
- rolesQuery: このオプションは、「select RoleName from Roles where User=?」と同等の準備済みステートメントである必要があります。
- suspendResume: boolean - ロールの検索の実行中に、現在のスレッドに関連するトランザクションを一時停止および後で再開します。
- transactionManagerJndiName: トランザクションママガーの JNDI 名（デフォルトは java:/TransactionManager）

LdapRolesMappingProvider

ロールを検索するため LDAP サーバーを使用してロールを割り当てるマッピングプロバイダー。

オプション:

-

bindDN: ユーザーおよびロールクエリーのために LDAP サーバーに対してバインドするために使用される DN。この DN には、baseCtxDN および rolesCtxDN の値に読み取りおよび検索パーミッションが必要です。

- **bindCredential:** bindDN のパスワードこれは、jaasSecurityDomain が指定されている場合は暗号化できます。

- **rolesCtxDN:** ユーザーロールを検索するコンテキストの固定 DN。これは、実際のロールがである DN ではなく、ユーザーロールを含むオブジェクトがある DN です。たとえば、Microsoft Active Directory サーバーでは、これは、ユーザーアカウントが存在する DN です。

- **roleAttributeID:** 承認ロールの名前が含まれる LDAP 属性。

- **roleAttributeIsDN:** roleAttributeID にロールオブジェクトの完全修飾 DN が含まれるかどうか。False の場合、ロール名はコンテキスト名の roleNameAttributeId 属性の値から取得されます。Microsoft Active Directory などの特定のディレクトリースキーマでは、この属性を true に設定する必要があります。

- **roleNameAttributeID:** ロール名を含む roleCtxDN コンテキスト内の属性の名前。RoleAttributeIsDN プロパティーが true に設定されている場合、このプロパティーはロールオブジェクトの name 属性を見つけるために使用されます。

- **parseRoleNameFromDN:** クエリーによって返された DN に roleNameAttributeID が含まれるかどうかを示すフラグ。true に設定すると、DN は roleNameATtributeID についてチェックされます。false に設定すると、DN は roleNameAttributeID に対してチェックされません。このフラグは LDAP クエリーのパフォーマンスを向上できます。

- **roleFilter:** 認証済みユーザーと関連付けられたロールを検索するために使用される検索フィルター。{0} 式を使用しているフィルターに、入力ユーザー名またはログインモジュールコールバックから取得した userDN が置換されます。{1} が使用されると、認証された userDN がフィルターに置き換わります。入力ユーザー名に一致する検索フィルターの例は (member={0}) です。認証済み userDN に一致する他の例は (member=gitops) です。

- **roleRecursion:** ロール検索が一致するコンテキストで行われる再帰のレベル数。再帰を無効にするには、これを 0 に設定します。

- **searchTimeLimit:** ユーザー/ロール検索のタイムアウト（ミリ秒単位）。デフォルトは 10000（10 秒）です。

- searchScope: 使用する検索範囲。

PropertiesRolesMappingProvider

「username=role1,role2,...」という形式でプロパティーファイルからロールを読み取る MappingProvider。

オプション:

- rolesProperties: プロパティーフォーマットされたファイル名。JBoss 変数の拡張は、${ jboss.variable} の 形式で使用できます。

SimpleRolesMappingProvider

オプションマップからロールを読み取る簡単な MappingProvider。option 属性名はロールを割り当てるプリンシパルの名前で、属性値はプリンシパルに割り当てるコンマ区切りのロール名です。

例A.3 例

```
<module-option name="JavaDuke" value="JBossAdmin,Admin"/>
<module-option name="joe" value="Users"/>
```

org.jboss.security.mapping.providers.attribute.DefaultAttributeMappingProvider

モジュールをチェックし、マッピングコンテキストからプリンシパル名を見つけ、principalName + ".email" という名前のモジュールオプションから属性電子メールアドレスを作成し、それを指定のプリンシパルへマップします。

LdapAttributeMappingProvider

LDAP からサブジェクトに属性をマッピングします。オプションには、LDAP JNDI プロバイダーがサポートするオプションが含まれます。

例A.4 標準のプロパティー名の例

```
Context.INITIAL_CONTEXT_FACTORY = "java.naming.factory.initial"
Context.SECURITY_PROTOCOL = "java.naming.security.protocol"
Context.PROVIDER_URL = "java.naming.provider.url"
Context.SECURITY_AUTHENTICATION = "java.naming.security.authentication"
```

オプション:

- bindDN: ユーザーおよびロールクエリーのために LDAP サーバーに対してバインドするために使用される DN。この DN には、baseCtxDN および rolesCtxDN の値に読み取りおよび検索パーミッションが必要です。
- bindCredential: bindDN のパスワードこれは、jaasSecurityDomain が指定されている場合は暗号化できます。
- baseCtxDN: ユーザー検索を開始するコンテキストの固定 DN。
- baseFilter: 認証するユーザーのコンテキストの検索に使用する検索フィルター。{0} 式を使用しているフィルターに、入力ユーザー名またはログインモジュールコールバックから取得した userDN が置換されます。この置換動作は、標準の __DirContext.search(Name, String, Object[], SearchControls cons)__ メソッドから取得されます。一般的な検索フィルターの例は (uid={0}) です。
- searchTimeLimit: ユーザー/ロール検索のタイムアウト（ミリ秒単位）。デフォルトは 10000（10 秒）です。
- attributeList: ユーザーの属性のコンマ区切りリスト。たとえば、mail,cn,sn,employeeType,employeeNumber です。
- jaasSecurityDomain: java.naming.security.principal の復号化に使用する JaasSecurityDomain。パスワードの暗号化形式は JaasSecurityDomain#encrypt64(byte[]) メソッドによって返されます。org.jboss.security.plugins.PBEUtils を使用して、暗号化されたフォームを生成することもできます。

バグの報告

A.4. 含まれるセキュリティー監査プロバイダーモジュール

JBoss EAP 6 はセキュリティー監査プロバイダーを 1 つ提供します。

コード	クラス
LogAuditProvider	org.jboss.security.audit.providers.LogAuditProvider

バグの報告

A.5. JBOSS-WEB.XML 設定リファレンス

はじめに

jboss-web.xml および web.xml デプロイメント記述子はいずれもデプロイメントの WEB-INF ディレクトリーに配置されます。jboss-web.xml は JBoss EAP の Web アプリケーションデプロイメント記述子で、JBoss Web の追加機能の追加設定オプションが含まれます。この記述子を使用すると web.xml 記述子の設定を上書きし、JBoss EAP 固有の設定を設定できます。

グローバルリソースの WAR 要件へのマッピング

アプリケーションの web.xml で設定される利用可能な設定要件の多くをローカルリソースにマップします。web.xml 設定の説明は、を参照してください http://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html。

たとえば、web.xml に jdbc/MyDataSource が必要な場合、jboss-web.xml はグローバルデータソース java:/DefaultDS をマップしてこのニーズに対応する場合があります。WAR はグローバルデータソースを使用して、jdbc/MyDataSource の必要性を埋めます。

表A.42 jboss-web.xml の一般的な最上位属性

属性	説明
servlet	servlet 要素はサーブレット固有のバインディングを指定します。
max-active-sessions	許可されるアクティブなセッションの最大数を決定します。セッションマネージャーが管理するセッションの数がこの値を超え、パッシベーションが有効な場合には、設定された **passivation -min-idle-time**に基づいて超過がパッシベートされます。 -1 に設定すると、無制限を意味します。
replication-config	**replication-config** 要素は、**jboss-web.xml** ファイルでセッションレプリケーションを設定するために使用されます。

属性	説明
passivation-config	**passivation-config** 要素は、**jboss-web.xml** ファイルでセッションパッシベーションを設定するために使用されます。
distinct-name	**distinct-name** 要素は、Web アプリケーションの EJB 3 固有名を指定します。
data-source	**web.xml** が必要とする **データソース** へのマッピング。
context-root	アプリケーションのルートコンテキスト。デフォルト値は、**.war** サフィックスのないデプロイメントの名前です。
virtual-host	アプリケーションが要求を受け入れる HTTP virtual-host の名前。HTTP **Host** ヘッダーの内容を参照します。
アノテーション	アプリケーションによって使用されるアノテーションを記述します。詳細は、<annotation> を参照してください。
listener	アプリケーションによって使用されるリスナーを記述します。詳細は、<listener> を参照してください。
session-config	この要素は **web.xml** の **<session-config>** 要素と同じ関数を埋め、互換性のためにのみ含まれます。
バルブ	は、アプリケーションによって使用されるバルブを表しています。詳細は、<valve> を参照してください。
overlay	アプリケーションに追加するオーバーレイの名前。
security-domain	アプリケーションによって使用されるセキュリティードメインの名前。セキュリティードメイン自体は Web ベースの管理コンソールまたは管理 CLI で設定されます。
security-role	この要素は **web.xml** の **<security-role> 要素** と同じ関数を入力し、互換性のためにのみ含まれます。
jacc-star-role-allow	**jacc-star-role-allow** 要素は、jacc プロバイダーが承認をバイパスできるように、**Web** レイヤーでエージェントを生成する jacc パーミッションを生成する必要があるかどうかを指定します。

属性	説明
use-jboss-authorization	この要素が存在し、大文字と小文字を区別しない値「true」が含まれている場合は、JBoss Web 認証スタックが使用されます。これが存在しないか、「true」ではない値が含まれている場合は、Java Enterprise Edition 仕様で指定された承認メカニズムのみが使用されます。この要素は JBoss EAP 6 の新機能です。
disable-audit	このブール値要素を **false** に設定して有効にし、Web 監査を無効にするには **true** に設定します。Web セキュリティー監査は Java EE 仕様の一部ではありません。この要素は JBoss EAP 6 の新機能です。
disable-cross-context	**false** の場合、アプリケーションは別のアプリケーションコンテキストを呼び出すことができます。デフォルトは **true** です。
enable-websockets	**jboss-web.xml** でこの要素を **true** に設定して、Web アプリケーションに対して WebSocket アクセスを有効にするかどうかを指定します。

以下の要素にはそれぞれ子要素があります。

<annotation>

アプリケーションによって使用されるアノテーションを記述します。以下の表は、< annotation> の子要素を示してい ます。

表A.43 アノテーション設定要素

属性	説明
class-name	アノテーションのクラス名
servlet-security	サーブレットセキュリティーを表す **@ServletSecurity** などの要素。
run-as	run-as 情報を表す **@RunAs** などの要素。
multipart-config	マルチパート設定情報を表す **@MultiPart** などの要素。

<listener>

リスナーを記述します。以下の表は、< listener> の子要素を示してい ます。

表A.44 リスナー設定要素

属性	説明
class-name	リスナーのクラス名
listener-type	アプリケーションのコンテキストに追加するリスナーの種類を示す **条件** 要素の一覧。有効な選択肢は以下の通りです。 **CONTAINER** ContainerListener をコンテキストに追加します。 **ライフサイクル** LifecycleListener をコンテキストに追加します。 **SERVLET_INSTANCE** InstanceListener をコンテキストに追加します。 **SERVLET_CONTAINER** WrapperListener をコンテキストに追加します。 **SERVLET_LIFECYCLE** コンテキストに WrapperLifecycle を追加します。
モジュール	リスナークラスが含まれるモジュールの名前。
パラメーター	パラメーター。< **param-name> と <param-value> の 2 つの子要素が含まれ** ます。

<valve>

アプリケーションのバルブを示します。<listener> と同様に、Class-name、module、および param 要素があります。

バグの報告

A.6. EJB セキュリティーパラメーターのリファレンス

表A.45 EJB セキュリティーパラメーター要素

要素	説明
<security-identity>	EJB のセキュリティーアイデンティティーに関連する子要素が含まれます。
<use-caller-identity />	EJB が呼び出し元と同じセキュリティーアイデンティティーを使用することを示します。
<run-as>	**<role-name** > 要素が含まれます。
<run-as-principal>	存在する場合は、発信呼び出しに割り当てられたプリンシパルを示します。存在しない場合、発信呼び出しは **anonymous** という名前のプリンシパルに割り当てられます。
<role-name>	EJB を実行するロールを指定します。
<description>	では、<role **-name> という名前のロールを説明します。** .

例A.5 セキュリティーアイデンティティーの例

以下の例では、**表A.45「EJB セキュリティーパラメーター要素」** で説明されている各タグを示しています。**< session>** 内でも使用できます。

```
<ejb-jar>
  <enterprise-beans>
    <session>
      <ejb-name>ASessionBean</ejb-name>
      <security-identity>
        <use-caller-identity/>
      </security-identity>
    </session>
    <session>
      <ejb-name>RunAsBean</ejb-name>
      <security-identity>
        <run-as>
          <description>A private internal role</description>
          <role-name>InternalRole</role-name>
        </run-as>
      </security-identity>
    </session>
  <session>
  <ejb-name>RunAsBean</ejb-name>
  <security-identity>
  <run-as-principal>internal</run-as-principal>
  </security-identity>
```

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

バグの報告

付録B 改訂履歴

改訂 6.4.0-11	Tuesday April 14 2015	Lucas Costi
Red Hat JBoss Enterprise Application Platform 6.4.0.GA		