Red Hat JBoss Enterprise Application Platform 7.4

アイデンティティー管理の設定方法

LDAP ディレクトリーおよびその他のアイデンティティーストアを使用して Red Hat JBoss Enterprise Application Platform へのユーザーアクセスを管理する手順

Last Updated: 2021-10-10

Red Hat JBoss Enterprise Application Platform 7.4 アイデンティティー管理の設定方法

LDAP ディレクトリーおよびその他のアイデンティティーストアを使用して Red Hat JBoss Enterprise Application Platform へのユーザーアクセスを管理する手順

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

本ガイドでは、LDAP ディレクトリーおよびその他のアイデンティティーストアを使用して JBoss EAP 管理インターフェースおよびセキュリティードメインで使用する方法を考察します。本書は、JBoss EAP 『セキュリティーアーキテクチャーガイド』に記載されている概念が展開されており、管理者が LDAP の基本知識があり、JBoss EAP 内のセキュリティー概念を理解してから、このガイドを確認してください。

目次

RED HAT ドキュメントへのフィードバック

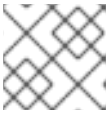
弊社のドキュメントに関するご意見やご感想をお寄せください。フィードバックをお寄せいただくには、ドキュメントのテキストを強調表示し、コメントを追加できます。以下の手順に従って、Red Hat ドキュメントのフィードバックをお寄せください。

要件

- Red Hat カスタマーポータルにログインします。
- Red Hat カスタマーポータルで、**Multi-page HTML** 形式のドキュメントを表示します。

手順

1. **フィードバック** をクリックして、既存のリーダーコメントを表示します。



注記

フィードバック機能は、**マルチページ HTML** 形式でのみ有効です。

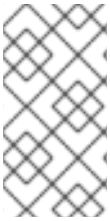
2. フィードバックを提供するドキュメントのセクションを強調表示します。
3. 選択したテキストの近くに表示されるプロンプトメニューで、**Add Feedback** をクリックします。
   ページの右側のフィードバックセクションにテキストボックスが開きます。
4. テキストボックスにフィードバックを入力し、**Submit** をクリックします。
   ドキュメントに関する問題を作成しました。
5. 問題を表示するには、フィードバックビューで問題トラッカーリンクをクリックします。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、弊社 の CTO、Chris Wright のメッセージを参照してください。

第1章 アイデンティティー管理の概要

さまざまなアイデンティティーストアでアプリケーションのセキュリティーを確保するための基本的なアイデンティティー管理の概要については、Red Hat JBoss Enterprise Application Platform (JBoss EAP)『**セキュリティーアーキテクチャー ガイド**』で説明しています。本ガイドでは、ファイルシステムや LDAP などのさまざまなアイデンティティーストアを設定し、アプリケーションを保護する方法を説明します。場合によっては、LDAP などの特定のアイデンティティーストアを承認局として使用することもできます。プリンシパルに関するさまざまなロールおよびアクセス情報は LDAP ディレクトリーに格納でき、JBoss EAP で直接使用したり、既存の JBoss EAP ロールにマップしたりできます。



注記

データベースや LDAP ディレクトリーなどの外部データストアでサポートされるアイデンティティーストアを使用すると、外部データストアと JBoss EAP インスタンス間のデータアクセスおよびトランスポートにより、認証および承認にパフォーマンスに影響する可能性があります。

第2章 ELYTRON サブシステム

2.1. ファイルシステムベースのアイデンティティーストアを使用した認証の設定

1. JBoss EAP で **filesystem-realm** を設定します。

```
/subsystem=elytron/filesystem-realm=exampleFsRealm:add(path=fs-realm-users,relative-to=jboss.server.config.dir)
```

   ディレクトリーが **jboss.server.config.dir** 外にある場合は、**path** と **relative-to** の値を適切に変更する 必要があります。

2. ユーザーを追加します。
   filesystem-realm を使用する場合は、管理 CLI を使用してユーザーを追加できます。

```
/subsystem=elytron/filesystem-realm=exampleFsRealm:add-identity(identity=user1)
/subsystem=elytron/filesystem-realm=exampleFsRealm:set-password(identity=user1, clear={password="password123"})
/subsystem=elytron/filesystem-realm=exampleFsRealm:add-identity-attribute(identity=user1, name=Roles, value=["Admin","Guest"])
```

3. **simple-role-decoder** を追加します。

```
/subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
```

   この **simple-role-decoder** は、**Roles** 属性からのプリンシパルのロールをデコードします。ロールが別の属性にある場合はこの値を変更できます。

4. **security-domain** を設定します。

```
/subsystem=elytron/security-domain=exampleFsSD:add(realms=[{realm=exampleFsRealm,role-decoder=from-roles-attribute}],default-realm=exampleFsRealm,permission-mapper=default-permission-mapper)
```

5. **undertow** サブシステムで **application-security-domain** を設定します。

```
/subsystem=undertow/application-security-domain=exampleApplicationDomain:add(security-domain=exampleFsSD)
```

注記

undertow サブシステムの **application-security-domain** は、**Configuration → Subsystems → Web (Undertow) → Application Security Domain** に移動して管理コンソールで設定できます。

6. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。
   JBoss EAP で設定した**application-security-domain** 使用するように、アプリケーションの **web.xml** および**jboss-web.xml** を更新する必要があります。この例については、Configure Web Applications to Use Elytron or Legacy Security for Authentication で参照できます。

アプリケーションは、認証にファイルシステムベースのアイデンティティーストアを使用しています。

2.2. プロパティーファイルベースのアイデンティティーストアを使用した認証の設定

1. プロパティーファイルを作成します。
   ユーザーがパスワードにマップされるプロパティーファイルと、ユーザーをロールにマッピングするプロパティーファイルを作成する必要があります。通常、これらのファイルは **jboss.server.config.dir** ディレクトリーにあり、命名規則 ***-users.properties** および ***-roles.properties**に従いますが、その他の場所や名前を使用することもできます。***-users.properties** ファイルには、次の手順で作成する **properties-realm** への参照が含まれる必要もあります。これは、次の手順で作成します。**$REALM_NAME=YOUR_PROPERTIES_REALM_NAME$**

   パスワードファイルの例: example-users.properties

   ```
   #$REALM_NAME=examplePropRealm$
   user1=password123
   user2=password123
   ```

   ロールファイルに対するユーザー例: example-roles.properties

   ```
   user1=Admin
   user2=Guest
   ```

2. JBoss EAP で **properties-realm** を設定します。

   ```
   /subsystem=elytron/properties-realm=examplePropRealm:add(groups-
   attribute=groups,groups-properties={path=example-roles.properties,relative-
   to=jboss.server.config.dir},users-properties={path=example-users.properties,relative-
   to=jboss.server.config.dir,plain-text=true})
   ```

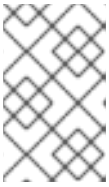
   properties-realm の名前は、**example-users.properties** ファイルの前の手順で使用した、**examplePropRealm** です。また、プロパティーファイルが **jboss.server.config.dir** 外にある場合は、**path** および **relative-to** の値を適切に変更する必要があります。

3. **security-domain** を設定します。

   ```
   /subsystem=elytron/security-domain=exampleSD:add(realms=
   [{realm=examplePropRealm,role-decoder=groups-to-roles}],default-
   realm=examplePropRealm,permission-mapper=default-permission-mapper)
   ```

4. **undertow** サブシステムで **application-security-domain** を設定します。

   ```
   /subsystem=undertow/application-security-domain=exampleApplicationDomain:add(security-
   domain=exampleSD)
   ```

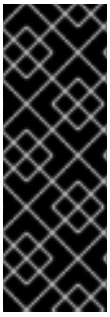
   注記

   undertow サブシステムの **application-security-domain** は、Configuration → Subsystems → Web (Undertow) → Application Security Domain に移動して管理コンソールで設定できます。

5. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。

JBoss EAP で設定した**application-security-domain** 使用するように、アプリケーションの **web.xml** および**jboss-web.xml** を更新する必要があります。この例については、Configure Web Applications to Use Elytron or Legacy Security for Authentication で参照できます。

アプリケーションは、認証にプロパティーファイルベースのアイデンティティーストアを使用しています。



重要

プロパティーファイルはサーバーの起動時のみ読み取られます。サーバーの起動後に、手動または **add-user** スクリプトを使用して追加したユーザーは、サーバーをリロードする必要があります。このリロードは、管理 CLI から **reload** コマンドを実行すると実行できます。

```
reload
```

2.3. データベースベースのアイデンティティーストアを使用した認証の設定

1. ユーザー名、パスワード、およびロールのデータベース形式を決定します。
アイデンティティーストアのデータベースを使用して認証を設定するには、ユーザー名、パスワード、およびロールがそのデータベースに保存される方法を決定する必要があります。この例では、以下のサンプルデータを含む単一つのテーブルを使用します。

username	password	roles
user1	password123	Admin
user2	password123	Guest

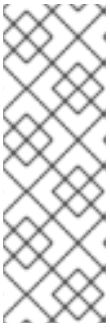
2. データソースを設定します。
JBoss EAP からデータベースに接続するには、適切なデータベースドライバーがディプロイされとデータソースが設定されている必要があります。以下の例は、PostgreSQL のドライバーをデプロイし、JBoss EAP でデータソースを設定する例になります。

```
deploy /path/to/postgresql-9.4.1210.jar

data-source add --name=examplePostgresDS --jndi-name=java:jboss/examplePostgresDS --driver-name=postgresql-9.4.1210.jar  --connection-url=jdbc:postgresql://localhost:5432/postgresdb --user-name=postgresAdmin --password=mysecretpassword
```

3. JBoss EAP で **jdbc-realm** を設定します。

```
/subsystem=elytron/jdbc-realm=exampleDbRealm:add(principal-query=[{sql="SELECT password,roles FROM eap_users WHERE username=?",data-source=examplePostgresDS,clear-password-mapper={password-index=1},attribute-mapping=[{index=2,to=groups}]}])
```



注記

上記の例は、単一の **principal-query** からパスワードとロールを取得する方法を示しています。また、ロールまたは追加の認証または承認情報を取得するために複数のクエリーが必要な場合には、**attribute-mapping** 属性で追加の **principal-query** を作成することもできます。

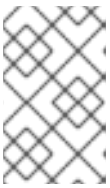
サポート対象のパスワードマッパーの一覧は、「パスワードマッパー」を参照してください。

4. **security-domain** を設定します。

```
/subsystem=elytron/security-domain=exampleDbSD:add(realms=
[{realm=exampleDbRealm,role-decoder=groups-to-roles}],default-
realm=exampleDbRealm,permission-mapper=default-permission-mapper)
```

5. **undertow** サブシステムで **application-security-domain** を設定します。

```
/subsystem=undertow/application-security-domain=exampleApplicationDomain:add(security-
domain=exampleDbSD)
```



注記

undertow サブシステムの **application-security-domain** は、Configuration → Subsystems → Web (Undertow) → Application Security Domain に移動して管理コンソールで設定できます。

6. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。
JBoss EAP で設定した**application-security-domain** 使用するように、アプリケーションの **web.xml** および**jboss-web.xml** を更新する必要があります。この例については、Configure Web Applications to Use Elytron or Legacy Security for Authentication で参照できます。

2.4. LDAPベースのアイデンティティーストアを使用した認証の設定

1. ユーザー名、パスワード、およびロールのLDAP形式を決定します。
アイデンティティーストアのLDAPサーバーを使用して認証を設定するには、ユーザー名、パスワード、およびロールが保存される方法を決定する必要があります。この例では、以下の構造を使用します。

```
dn: dc=wildfly,dc=org
dc: wildfly
objectClass: top
objectClass: domain

dn: ou=Users,dc=wildfly,dc=org
objectClass: organizationalUnit
objectClass: top
ou: Users

dn: uid=jsmith,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
```

```
cn: John Smith
sn: smith
uid: jsmith
userPassword: password123

dn: ou=Roles,dc=wildfly,dc=org
objectclass: top
objectclass: organizationalUnit
ou: Roles

dn: cn=Admin,ou=Roles,dc=wildfly,dc=org
objectClass: top
objectClass: groupOfNames
cn: Admin
member: uid=jsmith,ou=Users,dc=wildfly,dc=org
```

2. **dir-context** を設定します。
JBoss EAP から LDAP サーバーに接続するには、URL を指定する **dir-context** と、サーバーへの接続に使用するプリンシパルを設定する必要があります。

```
/subsystem=elytron/dir-context=exampleDC:add(url="ldap://127.0.0.1:10389",principal="uid=admin,ou=system",credential-reference={clear-text="secret"})
```

注記

Jakarta Management **ObjectName** を使用して LDAP 認証情報を復号化することはできません。代わりに、**JBoss EAP の サーバーセキュリティーの設定方法** で説明されているように、クレデンシャルストア を使用して認証情報のセキュリティーを確保できます。

3. JBoss EAP で **ldap-realm** を設定します。

```
/subsystem=elytron/ldap-realm=exampleLR:add(dir-context=exampleDC,identity-mapping={search-base-dn="ou=Users,dc=wildfly,dc=org",rdn-identifier="uid",user-password-mapper={from="userPassword"},attribute-mapping=[{filter-base-dn="ou=Roles,dc=wildfly,dc=org",filter="(&(objectClass=groupOfNames)(member={0}))",from="cn",to="Roles"}]})
```

警告

参照される LDAP サーバーの紹介にループが含まれる場合は、JBoss EAP サーバーで **java.lang.OutOfMemoryError** エラーが発生することがあります。

4. **simple-role-decoder** を追加します。

```
/subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
```

5. **security-domain** を設定します。

```
/subsystem=elytron/security-domain=exampleLdapSD:add(realms=[{realm=exampleLR,role-decoder=from-roles-attribute}],default-realm=exampleLR,permission-mapper=default-permission-mapper)
```

6. **undertow** サブシステムで **application-security-domain** を設定します。

```
/subsystem=undertow/application-security-domain=exampleApplicationDomain:add(security-domain=exampleLdapSD)
```

注記

undertow サブシステムの **application-security-domain** は、Configuration → Subsystems → Web (Undertow) → Application Security Domain に移動して管理コンソールで設定できます。

7. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。
JBoss EAP で設定した**application-security-domain** 使用するように、アプリケーションの **web.xml** および**jboss-web.xml** を更新する必要があります。この例については、Configure Web Applications to Use Elytron or Legacy Security for Authentication で参照できます。

重要

elytron サブシステムが LDAP サーバーを使用して認証を実行する場合、JBoss EAP は **500**または内部サーバーエラー (LDAP サーバーにアクセスできない場合) を返します。この動作は、同じ条件下で **401** または非承認のエラーコードを返すレガシー **security** サブシステムを使用して JBoss EAP の以前のバージョンとは異なります。

2.5. 証明書を使用した認証の設定

重要

証明書ベースの認証を設定する前に、双方向 SSL を設定する必要があります。双方向 SSL の設定に関する詳細は、『**サーバーセキュリティーの設定方法**』の「Elytron サブシステムを使用してアプリケーションに対して双方向 SSL/TLS を有効化する」を参照してください。

1. **key-store-realm** を設定します。

```
/subsystem=elytron/key-store-realm=ksRealm:add(key-store=twoWayTS)
```

このレルムは、クライアントの証明書が含まれるトラストストアで設定する必要があります。認証プロセスでは、双方向 SSL ハンドシェイク時にクライアントが提示するのと同じ証明書を使用します。

2. デコーダーを作成します。
証明書から取得したプリンシパルをデコードするには、**x500-attribute-principal-decoder** を作成する必要があります。以下の例では、最初の **CN** 値を基にしてプリンシパルをデコードします。

```
/subsystem=elytron/x500-attribute-principal-
decoder=CNDecoder:add(oid="2.5.4.3",maximum-segments=1)
```

たとえば、完全な **DN** が **CN=client,CN=client-certificate,DC=example,DC=jboss,DC=org** であった場合は、**CNDecoder** が **client** としてデコードします。このデコードされたプリンシパルは、**ksRealm** の値で設定したトラストストアの証明書を検索する **alias** 値として使用されます。

重要

デコードされたプリンシパル **MUST** は、クライアントの証明書に対してサーバーのトラストストアで設定した **alias** 値です。

- 必要に応じて、サブジェクトの別名エクステンションを使用してエビデンスデコーダーを設定し、サブジェクトの別名をプリンシパルとして使用することができます。詳細は、『**サーバーセキュリティーの設定方法**』ガイドの「サブジェクトの別名拡張を使用した X.509 証明書の Evidence Decoder の設定」を参照してください。

3. ロールを割り当てるには **constant-role-mapper** を追加します。
この例では、**constant-role-mapper** を使用して **ksRealm** からプリンシパルにロールを割り当てますが、他の方法を使用することもできます。

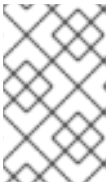
```
/subsystem=elytron/constant-role-mapper=constantClientCertRole:add(roles=[Admin,Guest])
```

4. **security-domain** を設定します。

```
/subsystem=elytron/security-domain=exampleCertSD:add(realms=[{realm=ksRealm}],default-
realm=ksRealm,permission-mapper=default-permission-mapper,principal-
decoder=CNDecoder,role-mapper=constantClientCertRole)
```

5. **undertow** サブシステムで **application-security-domain** を設定します。

```
/subsystem=undertow/application-security-domain=exampleApplicationDomain:add(security-
domain=exampleCertSD)
```



注記

undertow サブシステムの **application-security-domain** は、**Configuration → Subsystems → Web (Undertow) → Application Security Domain** に移動して管理コンソールで設定できます。

6. **server-ssl-context** を更新します。

```
/subsystem=elytron/server-ssl-context=twoWaySSC:write-attribute(name=security-
domain,value=exampleCertSD)
/subsystem=elytron/server-ssl-context=twoWaySSC:write-attribute(name=authentication-
optional, value=true)
reload
```

7. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。

JBoss EAP で設定した**application-security-domain** 使用するように、アプリケーションの **web.xml** および**jboss-web.xml** を更新する必要があります。この例については、Configure Web Applications to Use Elytron or Legacy Security for Authentication で参照できます。

さらに、**CLIENT-CERT** を認証方法として使用するように **web.xml** を更新する必要があります。

```
<login-config>
  <auth-method>CLIENT-CERT</auth-method>
  <realm-name>exampleApplicationDomain</realm-name>
</login-config>
```

2.6. 複数のアイデンティティーストアを使用した認証および承認の設定

異なるアイデンティティーストア間でアイデンティティーの属性を保存する場合は、**aggregate-realm** を使用して認証および承認のためにアイデンティティー属性を単一のセキュリティーレルムに読み込みます。

2.6.1. Elytron の集約レルム

Aggregate-realm を使用すると、認証と別のセキュリティーレルムにいずれかのセキュリティーレルムを使用したり、Elytron での承認に複数のセキュリティーレルムの集約を使用したりできます。たとえば、集約レルムを設定して、認証にプロパティーレルムを使用して、承認に JDBC リレムを使用することができます。

複数の承認レルムを集約するよう設定されたアグリゲートレルムでは、以下のようにアイデンティティーが作成されます。

- 承認に設定された各セキュリティーレルムの属性値はロードされます。
- 属性が 2 つ以上の承認レルムで定義されている場合は、最初に出現した属性の値が使用されます。

以下の例は、複数の承認レルムに同じアイデンティティー属性の定義が含まれる場合にアイデンティティーがどのように作成されるかを示しています。

例

レルム設定を集約します。

```
authentication-realm=properties-realm,
authorization-realms=[jdbc-realm,ldap-realm]
```

- JDBC レルムから取得した属性値:

```
e-mail: user@example.com
groups: Supervisor, User
```

- ldap レルムから取得した属性値:

```
e-mail: administrator@example.com
phone: 0000 0000 0000
```

その結果、集約レルムから取得されるアイデンティティーは以下のようになります。

```
e-mail: user@example.com
groups: Supervisor, User
phone: 0000 0000 0000
```

この例では、属性 **e-mail** が両方の承認レルムで定義されています。集約レルムは **authorization-realms=[jdbc-realm,ldap-realm]** のように承認レルムを集約するよう設定されていたため、JDBC レルムで定義された値は、生成される集約レルムの属性 **e-mail** に使用されます。

2.6.2. 集約レルムを使用した認証および承認の設定

集約レルムを使用して認証と承認を設定するには、集約レルムを作成し、セキュリティードメインとアプリケーションセキュリティードメインを集約レルムを使用するように設定します。

要件

- 集計されるセキュリティーレルムが設定されます。
  セキュリティーレルムの設定に関する詳細は、『**アイデンティティー管理の設定方法**』ガイドの「Elytron サブシステム」を参照してください。
- セキュリティードメインで使用されるロールデコーダーが設定されます。
  ロールデコーダーの詳細は、『**サーバーセキュリティーの設定方法**』の「Elytron ロールデコーダーの作成」を参照してください。

手順

1. 集約レルムを作成します。
   - 単一の承認レルムで集約レルムを作成するには、以下を実行します。

     ```
     /subsystem=elytron/aggregate-realm=exampleAggregateRealm:add(authentication-realm=__SECURITY_REALM_FOR_AUTHENTICATION__, authorization-realm=__SECURITY_REALM_FOR_AUTHORIZATION__)
     ```

   - 複数の認可レルムを使用して集約レルムを作成するには、以下を実行します。

     ```
     /subsystem=elytron/aggregate-realm=exampleAggregateRealm:add(authentication-realm=__SECURITY_REALM_FOR_AUTHENTICATION__, authorization-realms=[__SECURITY_REALM_FOR_AUTHORIZATION_1__,__SECURITY_REALM_FOR_AUTHORIZATION_2__,...,__SECURITY_REALM_FOR_AUTHORIZATION_N__])
     ```

2. **security-domain** を設定します。

   ```
   /subsystem=elytron/security-domain=exampleAggregateRealmSD:add(realms=[{realm=exampleAggregateRealm,role-decoder=__ROLE-DECODER__}],default-realm=exampleAggregateRealm,permission-mapper=default-permission-mapper)
   ```

3. **undertow** サブシステムで **application-security-domain** を設定します。

   ```
   /subsystem=undertow/application-security-domain=exampleAggregareRealmApplicationDomain:add(security-domain=exampleAggregateRealmSD)
   ```

4. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。

JBoss EAP で設定した**application-security-domain** 使用するように、アプリケーションの **web.xml** および**jboss-web.xml** を更新する必要があります。この例については、Configure Web Applications to Use Elytron or Legacy Security for Authentication で参照できます。

2.6.3. 集約レルムの例

単一の作成者レルムを使用した集約レルムの例

この例では、認証に **properties-realm** を使用し、承認に **jdbc-realm** を使用します。

以下のレルムを事前に設定する必要があります。

- examplPropertiesRealm という名前の **properties-realm**
- exampleJdbcRealm という名前の **jdbc-realm**

以下のコマンドを実行して、集約レルムを作成します。

```
/subsystem=elytron/aggregate-realm:exampleSimpleAggregateRealm:add(authentication-realm=examplPropertiesRealm,authorization-realm=exampleJdbcRealm)
```

2 つの承認レルムを使用した集約レルムの例

この例では、認証に **properties-realm** が使用され、承認に **ldap-realm** および **jdbc-realm** の集約が使用されます。

以下のレルムを事前に設定する必要があります。

- examplPropertiesRealm という名前の **properties-realm**
- exampleJdbcRealm という名前の **jdbc-realm**
- exampleLdapRealm という名前の **ldap-realm**

以下のコマンドを実行して、集約レルムを作成します。

```
/subsystem=elytron/aggregate-realm:exampleSimpleAggregateRealm:add(authentication-realm=examplPropertiesRealm,authorization-realms=[exampleJdbcRealm,exampleLdapRealm])
```

2.7. アプリケーションの認証設定の上書き

アプリケーションの認証設定を JBoss EAP で設定された設定で上書きできます。これを実行するには、**undertow** サブシステムの **application-security-domain** セクションで **override-deployment-configuration** プロパティーを使用します。

```
/subsystem=undertow/application-security-domain=exampleApplicationDomain:write-attribute(name=override-deployment-config,value=true)
```

注記

undertow サブシステムの **application-security-domain** は、**Configuration → Subsystems → Web (Undertow) → Application Security Domain** に移動して管理コンソールで設定できます。

たとえば、アプリケーションは **jboss-web.xml** の **exampleApplicationDomain** で **FORM** 認証を使用するように設定されます。

例: jboss-web.xml

```
<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>exampleApplicationDomain</realm-name>
</login-config>
```

override-deployment-configuration を有効にすることにより、**BASIC** や **DIGEST** などの異なる認証メカニズムを指定する新しい **http-authentication-factory** を作成できます。

例: http-authentication-factory

```
/subsystem=elytron/http-authentication-factory=exampleHttpAuth:read-resource()
{
    "outcome" => "success",
    "result" => {
        "http-server-mechanism-factory" => "global",
        "mechanism-configurations" => [{
            "mechanism-name" => "BASIC",
            "mechanism-realm-configurations" => [{"realm-name" => "exampleApplicationDomain"}]
        }],
        "security-domain" => "exampleSD"
    }
}
```

これにより、アプリケーションの **jboss-web.xml** に定義された認証方法が上書きされ、**FORM**ではなく **BASIC** を使用したユーザー認証が試行されます。

2.8. セキュリティーレルムのキャッシングの設定

Elytron は、セキュリティーレルムからクレデンシャルルックアップの結果をキャッシュできる **caching-realm** を提供します。たとえば、これを使用して LDAP またはデータベースからの認証情報のキャッシュを設定し、頻繁にクエリーされるユーザーのパフォーマンスを向上することができます。

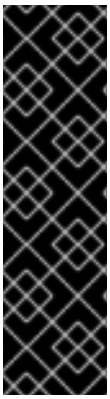
caching-realm は **LRU** または **Leastcently Used** キャッシュストラテジーを使用して **PasswordCredential** の認証情報をキャッシュします。キャッシュストラテジーでは、エントリーの最大数に達すると、アクセスが最も少ないエントリーが破棄されます。

caching-realm は以下のセキュリティーレルムで使用できます。

- **filesystem-realm**
- **jdbc-realm**
- **ldap-realm**
- カスタムセキュリティーレルム

JBoss EAP 以外で認証情報ソースを変更する場合は、それらの変更は基盤のセキュリティーレルムがリッスンに対応している場合にのみ JBoss EAP キャッシングレルムに伝搬されます。特に **ldap-realm** はリスニングをサポートしますが、**roles** 内の **roles** などのフィルターされた属性は対応しません。

キャッシュレルムでユーザーデータのキャッシュが正しいことを確認するには、認証情報のソースではなく、キャッシュレルムを使用してユーザー属性を変更することが推奨されます。また、キャッシュの消去を行うことも可能です。



重要

キャッシュレルムを使用してユーザーに変更を加えることは、テクノロジープレビューとしてのみ提供されます。テクノロジープレビューの機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。

テクノロジープレビュー機能のサポート範囲については、Red Hat カスタマーポータルの「テクノロジープレビュー機能のサポート範囲」を参照してください。

caching-realm を設定して使用するには、以下を実行します。

1. 既存のセキュリティーレルムを作成します。
caching-realm で使用するには、既存のセキュリティーレルムが必要です。たとえば、ファイルシステムベースのアイデンティティーストアを使用した認証の設定手順と同様の **filesystem-realm** を作成できます。

   Filesystem-realm の例

   ```
   /subsystem=elytron/filesystem-realm=exampleFsRealm:add(path=fs-realm-users, relative-to=jboss.server.config.dir)

   /subsystem=elytron/filesystem-realm=exampleFsRealm:add-identity(identity=user1)

   /subsystem=elytron/filesystem-realm=exampleFsRealm:set-password(identity=user1, clear={password="password123"})

   /subsystem=elytron/filesystem-realm=exampleFsRealm:add-identity-attribute(identity=user1,name=Roles,value=["Admin","Guest"])

   /subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
   ```

2. **caching-realm** を作成します。
キャッシュする既存のレルムがあることを確認したら、そのレルムを参照する **caching-realm** を作成します。

   exampleFsRealm を使用する caching-realm の例

   ```
   /subsystem=elytron/caching-realm=exampleCacheRealm:add(realm=exampleFsRealm)
   ```

3. **caching-realm** を使用します。
caching-realm を作成したら、他のセキュリティーレルムと同じように、セキュリティー設定で使用できます。たとえば、ファイルシステムベースのアイデンティティーストアを使用した認証の設定で **filesystem-realm** を使用するのと同じ場所で使用できます。

   Caching-realm を使用した設定例

   ```
   /subsystem=elytron/security-domain=exampleFsSD:add(realms=
   ```

```
[{realm=exampleCacheRealm, role-decoder=from-roles-attribute}], default-
realm=exampleCacheRealm, permission-mapper=default-permission-mapper)

/subsystem=elytron/http-authentication-factory=example-fs-http-auth:add(http-server-
mechanism-factory=global, security-domain=exampleFsSD, mechanism-configurations=
[{mechanism-name=BASIC, mechanism-realm-configurations=[{realm-
name=exampleApplicationDomain}]}])
```

caching-realm の **maximum-entries** および **maximum-age** 属性を使用すると、キャッシュサイズおよびアイテムの有効期限を制御できます。これらの属性に関する詳細は、『**サーバーセキュリティーの設定方法**』の「Elytron サブシステムのコンポーネントのリファレンス」を参照してください。

caching-realm キャッシュの削除

clear-cache コマンドを使用すると既存のキャッシュをクリアできます。キャッシュを消去すると、セキュリティーレルムの最新データを使用して強制的に再配置されます。

```
/subsystem=elytron/caching-realm=exampleCacheRealm:clear-cache
```

2.9. コンテナー管理のシングルサインオンを使用するようにアプリケーションを設定する

Elytron **FORM** 認証を使用して、アプリケーションのコンテナー管理シングルサインオンを使用するよう JBoss EAP を設定できます。これにより、ユーザーは 1 度だけ認証を行い、再認証せずに **FORM** 認証方法でセキュア化された他のリソースにアクセスできます。

以下の場合には、関連するシングルサインオンセッションが無効になります。

- アクティブなローカルセッションは残っていない。
- アプリケーションからログアウトしている。

重要

これらのインスタンスがクラスター内にある限り、異なる JBoss EAP インスタンスにデプロイされたアプリケーション間でシングルサインオンを使用できます。

1. **key-store** を作成します。
   SSO に参加している異なるサーバー間でセキュアな通信チャネルを設定するには、**key-store** が必要です。このチャネルは、シングルサインオンセッションが作成されたり破棄されたときに発生するイベントに関するメッセージを、ログイン中およびログアウト時に交換するために使用されます。

   elytron サブシステムで **key-store** を作成するには、以下のように Java KeyStore を作成します。

   ```
   keytool -genkeypair -alias localhost -keyalg RSA -keysize 1024 -validity 365 -keystore
   keystore.jks -dname "CN=localhost" -keypass secret -storepass secret
   ```

   keystore.jks ファイルが作成されたら、以下の管理 CLI コマンドを実行して Elytron に **key-store** 定義を作成します。

```
/subsystem=elytron/key-store=example-keystore:add(path=keystore.jks, relative-
to=jboss.server.config.dir, credential-reference={clear-text=secret}, type=JKS)
```

2. セキュリティーレルムを追加します。
以下の管理 CLI コマンドを使用して、ユーザーがローカルファイルシステムに保存されるアイデンティティーストアの **FileSystem** レルムを作成します。

```
/subsystem=elytron/filesystem-realm=example-realm:add(path=/tmp/example-realm)
```

3. 以下の管理 CLI コマンドを使用して **security-domain** を作成します。

```
/subsystem=elytron/security-domain=example-domain:add(default-realm=example-
realm,permission-mapper=default-permission-mapper,realms=[{realm=example-realm,role-
decoder=groups-to-roles}]
```

注記

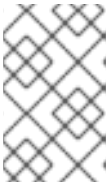
SSO を使用するアプリケーションは、通常はユーザーのログインページを提供する必要があるため、**HTTP FORM** 認証を使用する必要があります。

4. **undertow** サブシステムでアプリケーションセキュリティーを設定します。

注記

すでに **undertow** サブシステムで定義されている **application-security-domain** があり、アプリケーションへのシングルサインオンを有効にするためにこれを使用する場合は、この手順を省略できます。

```
/subsystem=undertow/application-security-domain=other:add(security-domain=example-
domain)
```



注記

デフォルトでは、アプリケーションが **jboss-web.xml** ファイルで特定のセキュリティドメインを定義しない場合、アプリケーションサーバーは **other** という名前でこれを選択します。

5. **undertow** サブシステムを更新してシングルサインオンを有効にし、キーストアを使用します。
シングルサインオンは **undertow** サブシステムの特定の **application-security-domain** 定義に対して有効化されます。アプリケーションのデプロイに使用するサーバーが同じ設定を使用していることが重要になります。

シングルサインオンを有効にするには、以下のように **undertow** サブシステムの既存の **application-security-domain** を変更します。

```
/subsystem=undertow/application-security-domain=other/setting=single-sign-on:add(key-
store=example-keystore, key-alias=localhost, domain=localhost, credential-reference={clear-
text=secret})
```

注記

undertow サブシステムの **application-security-domain** は、Configuration → Subsystems → Web (Undertow) → Application Security Domain に移動して管理コンソールで設定できます。

SSO 属性およびその定義の詳細は、Reference for Single Sign-on Attributes を参照してください。

6. アプリケーションの **web.xml** および **jboss-web.xml** ファイルを設定します。
JBoss EAP で設定した**application-security-domain** 使用するように、アプリケーションの **web.xml** および**jboss-web.xml** を更新する必要があります。この例については、Configure Web Applications to Use Elytron or Legacy Security for Authentication で参照できます。

JBoss EAP は、 **undertow** および **infinispan** サブシステムを使用する クラスター化された SSO とクラスター化されていない SSO に対して、追加設定なしでサポートを提供します。

2.10. ベアラートークンによる認証および承認の設定

2.10.1. ベアラートークン認証

BEARER_TOKEN 認証メカニズムを使用して、アプリケーションに送信された HTTP リクエストを承認できます。Web ブラウザーなどのクライアントが HTTP リクエストをアプリケーションに送信すると、**BEARER_TOKEN** メカニズムは要求の **Authorization** HTTP ヘッダーにベアラートークンがあることを確認します。

Elytron は、OpenID Connect ID トークンなどの JWT 形式のベアラートークン、または OAuth2 準拠の承認サーバーによって発行される不透明なトークンを使用して、認証をサポートします。追加リソースのセクションを参照してください。

以下の例は、Authorization HTTP ヘッダーに **the** mF **_9.B5f-4.1JqM** ベアラートークンが含まれることを示しています。

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

The **BEARER_TOKEN** メカニズムにより、ベアラートークンの文字列を抽出し、検証のために **token-realm** 実装に渡すことができます。実装がベアラートークンを正しく検証する場合、Elytron はトークンで表される情報に基づいてセキュリティーコンテキストを作成します。アプリケーションはこのセキュリティーコンテキストを使用して要求元についての情報を取得できます。その後、要求側が HTTP リソースへのアクセスを提供し、リクエストを満たすかどうかを決定できます。

要求担当者がベアラートークンを提供しない場合、The **BEARER_TOKEN** メカニズムは **401 HTTP** ステータスコードを返します。以下は例になります。

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example"
```

リクエスターがリソースにアクセスする権限がない場合は、**The BEARER_TOKEN メカニズムは、403 の HTTP** ステータスコードを返します。

```
HTTP/1.1 403 Forbidden
```

関連情報

- JWT 検証の詳細は、「JSON Web Tokens(JWT)認証の設定」 を参照してください。
- OAuth2 の検証に関する詳細は、「OAuth 2 対応の承認サーバーによって発行されるトークンによる認証の設定」を参照してください。
- **token-realm** で使用できる属性の詳細は、『 **サーバーセキュリティーの設定方法**』の「**表 A.93.token-realm 属性** 」 を参照してください。

2.10.2. JSON Web トークン(JWT)認証の設定

elytron サブシステムで **token-realm** を指定すると、JWT のサポートを有効にできます。

token-realm 内で、属性と **jwt** トークンバリデーターを指定できます。Elytron は以下のチェックを完了します。

- 自動有効期限は、**exp** claim および **nbf** 要求で指定された値をチェックします。
- オプション： 以下の方法のいずれかを使用して利用可能な公開鍵に基づいて、署名チェックを行います。
  - **public-key** または **certificate** 属性の使用
  - 名前付きの公開鍵でキーマップを使用します。
  - **client-ssl-context** 属性を使用して、jk **u** 要求で指定された URL からリモート JSON Web Key(JWK)を取得します。
- 必要に応じて、JWT をチェックし、これが **iss** および **aud** claim でサポートされる値のみを含むようにします。**発行者** および **オーディエンス** 属性を使用して、これらのチェックを実行できます。

以下の例は、**token-realm** をセキュリティーレルムとして、**principal-claim** を属性として示しています。**principal-claim** 属性は、**elytron** がプリンシパルの名前の取得に使用する要求の名前を定義します。**jwt** 要素は、トークンを JWT として検証する必要があることを指定します。

設定された token-realm の例

```
<token-realm name="${token_realm_name}" principal-claim="${principal_claim_key}">
   <jwt issuer="${issuer_name}"
       audience="${audience_name}"
       <public-key="${public_key_in_PEM_format}"/>
 </token-realm>
```

token-realm のキーマップを定義できます。次に、署名の検証に別の鍵のペアを使用し、この鍵のペアを簡単にローテーションすることができます。Elytron はトークンから **kid** 要求を取得し、検証に対応する公開鍵を使用します。

- **kid** クレームが JWT に存在しない場合は、**token-realm** は **jwt** の **public-key** 属性で指定された値を使用して署名を検証します。
- **kid** クレームが JWT に存在しない状態で **public-key を設定していない場合**、**token-realm** はトークンを無効にします。

token-realm の設定済みのキーマップの例。

```
<token-realm name="${token_realm_name}" principal-claim="${principal_claim_key}">
    <jwt issuer="${issuer_name}" audience="${audience_name}">
        <key kid="${key_ID_from_kid_claim}" public-key="${public_key_in_PEM_format}"/>
        <key kid="${another_key_ID_from_kid_claim}" public-key="${public_key_in_PEM_format}"/>
    </jwt>
</token-realm>
```

手順

1. **keytool** を使用して **key-store** を作成します。

   keytool を使用して key-store を作成する例。

   ```
   keytool -genkeypair -alias <alias_name> -keyalg <key_algorithm> -keysize <key_size> -
   validity <key_validity_in_days> -keystore <key_store_path> -dname
   <distinguished_name> -keypass <key_password> -storepass  <key_store_password>
   ```

   次に、**elytron** サブシステムに **key-store** 定義を追加します。

   elytron サブシステムで key-store 定義を追加する例。

   ```
   /subsystem=elytron/key-store=<key_store_name>:add(path=<key_store_path> ,
   credential-reference={clear-text=<key_store_password>}, type=<keystore_type>)
   ```

2. **elytron** サブシステムで **token-realm** を作成し、token **-realm に属性と jwt トークン** バリデーターを指定します。

   elytron サブシステムで token-realm を作成する例。

   ```
   /subsystem=elytron/token-realm=<token_realm_name>:add(jwt={issuer=
   [<issuer_name>],audience=[<audience_name>],key-
   store=<key_store_name>,certificate=<alias_name>},principal-
   claim=<principal_claim_key>)
   ```

次のステップ

- アプリケーションの認証を設定するには、「アプリケーションの 認証の設定」を参照してください。

関連情報

- **token-realm jwt** 属性の詳細は、『 **サーバーセキュリティーの設定方法**』の「 表 A.94. token-realm jwt 属性 」 を参照してください。

2.10.3. OAuth2 準拠の承認サーバーによって発行されたトークンを使用した認証の設定

Elytron は、OAuth2 準拠の承認サーバーによって発行されるベアラートークンをサポートします。事前定義された **oauth2-introspection** エンドポイントに対してトークンを検証するように、トークンレルムを設定できます。

手順

1. トークンレルムを作成します。

   elytron サブシステムを使用してセキュリティーレルムを作成する例：

   ```
   /subsystem=elytron/token-realm=<token_realm_name>:add(principal-claim=<principal_claim_key>, oauth2-introspection={client-id=<client_id>, client-secret=<client_secret>, introspection-url=<introspection_URL>})
   ```

   以下の例は、**token-realm** 要素に指定された **oauth2-introspection** 要素を示しています。このトークンレルムは、事前定義された **oauth2-introspection** エンドポイントに対してトークンを検証するように設定されます。**oauth2-introspection** エンドポイントは、**client-id** および **client-secret** 属性で指定された値を使用してクライアントを識別します。

   token-realm 要素内の oauth2-introspection 要素の例：

   ```
   <token-realm name="${token_realm_name}" principal-claim="${principal_claim_key}">
       <oauth2-introspection client-id="${client_id}"
                             client-secret="${client_secret}"
                             introspection-url="${introspection_URL}"
                             host-name-verification-policy="${hostname_verification_policy_value}"/>
   </token-realm>
   ```

次のステップ

- アプリケーションの認証を設定するには、「アプリケーションの 認証の設定」を参照してください。

関連情報

- トークンイントロスペクションエンドポイントについての詳細は、Table A.95 を参照してください。**token-realm oauth2-introspection 属性**

2.10.4. アプリケーションのベアラートークン認証の設定

アプリケーションの認証を設定するには、OpenID Connect ID トークンなどの JWT 形式でベアラートークンを使用するか、OAuth2 準拠の承認サーバーによって発行される不透明なトークンを使用します。

前提条件

- 必要な認証方法に応じて、以下の手順の 1 つを実行して **token-realm** を作成します。
  - JSON Web Tokens(JWT)認証の設定
  - OAuth2 準拠の承認サーバーによって発行されるトークンで認証を設定する。

手順

1. **elytron** サブシステムでセキュリティードメインを作成します。セキュリティードメインにトークンセキュリティーレルムを指定するようにしてください。

   elytron サブシステムでセキュリティードメインを作成する例。

```
/subsystem=elytron/security-domain=<security_domain_name>:add(realms=
[{realm=<token_realm_name>,role-decoder=<role_decoder_name>}],permission-
mapper=<permission_mapper_name>,default-realm=<token_realm_name>)
```

2. **BEARER_TOKEN** メカニズムを使用する **http-authentication-factory** を作成します。

   http-authentication-factory の作成例。

```
/subsystem=elytron/http-authentication-
factory=<authentication_factory_name>:add(security-
domain=<security_domain_name>,http-server-mechanism-factory=global,mechanism-
configurations=[{mechanism-name=BEARER_TOKEN,mechanism-realm-configurations=
[{realm-name=<token_realm_name>}]}])
```

3. **undertow** サブシステムで **application-security-domain** を設定します。

   undertow サブシステムで application-security-domain を設定する例。

```
/subsystem=undertow/application-security-
domain=<application_security_domain_name>:add(http-authentication-
factory=<authentication_factory_name>)
```

4. アプリケーションの **web.xml** および **jboss-web.xml** ファイルを設定します。アプリケーションの **web.xml** が **BEARER_TOKEN 認証メソッドを指定して** いることを確認する必要があります。さらに、jboss **-web.xml** が JBoss EAP で設定した **application-security-domain** を使用するようにしてください。

関連情報

- **BEARER_TOKEN** 認証メカニズムの詳細は、「 ベアラートークンの認証」を参照してください。
- **token-realm jwt** 属性の詳細は、『 **サーバーセキュリティーの設定方法**』の「**表 A.94. token-realm jwt 属性** 」 を参照してください。
- OAuth2 トークンエンドポイントで使用できる属性の詳細は、「 Table A.95」を参照してください。**token-realm oauth2-introspection** 属性
- アプリケーションの **web.xml および jboss-web.xml** の設定に関する詳細は、『 How to Configure Identity Management』ガイドの「Configure Web Applications to Use Elytron or Legacy Security for Authentication 」を参照してください。

第3章 レガシーセキュリティーサブシステム

3.1. LDAP を使用するよう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、ログインモジュールを使用して認証および承認に LDAP サーバーを使用するよう設定できます。セキュリティードメインおよびログインモジュールの基本は、JBoss EAP 『セキュリティーアーキテクチャー』ガイドで説明しています。**LdapExtended** は LDAP サーバー (Active Directory を含む) との統合に推奨されるログインモジュールですが、他のいくつかの LDAP ログインモジュールも使用できます。特に、LDAP を使用するようセキュリティードメインを設定するには、**Ldap**、**AdvancedLdap**、**AdvancedAdLdap** ログインモジュールを使用することもできます。本セクションでは、**LdapExtended** ログインモジュールを使用して、認証および承認に LDAP を使用するセキュリティードメインの作成方法を説明しますが、他の LDAP ログインモジュールを使用することもできます。他の LDAP ログインモジュールの詳細は、JBoss EAP『**ログインモジュールのリファレンス**』を参照してください。

重要

レガシー **security** サブシステムが LDAP サーバーを使用して認証を実行する場合、JBoss EAP は **500**または内部サーバーエラー (LDAP サーバーにアクセスできない場合) を返します。この動作は、同じ条件下で、**401**、または未承認のエラーコードを返す JBoss EAP の以前のバージョンとは異なります。

3.1.1. LdapExtended ログインモジュール

LdapExtended (**org.jboss.security.auth.spi.LdapExtLoginModule**) はログインモジュール実装で、LDAP サーバー上のバインドユーザーと関連付けられたロールの特定に使用されます。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。セキュリティードメインで LDAP を使用する場合、ほとんどの場合、特に Active Directory 以外の LDAP 実装では、LdapExtended ログインモジュールを使用する必要があります。LdapExtended ログインモジュールの設定オプションの完全リストは、JBoss EAP『**ログインモジュールのリファレンス**』の「LdapExtended ログインモジュール」を参照してください。

認証は以下のように行われます。

1. LDAP サーバーへの最初のバインドは、**bindDN** オプションおよび **bindCredential** オプションを使用して行われます。**BindDN** は LDAP ユーザーであり、ユーザーとロールの **baseCtxDN** および **rolesCtxDN** ツリーの両方を検索する機能があります。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. 生成されるユーザー DN は、**InitialLdapContext** 環境 **Context.SECURITY_PRINCIPAL** として使用してユーザー DN を使用し、LDAP サーバーにバインドして認証されます。**Context.SECURITY_CREDENTIALS** プロパティーはコールバックハンドラーによって取得される **String** パスワードに設定されます。

3.1.1.1. LdapExtended ログインモジュールを使用するようセキュリティードメインを設定

データの例 (LDIF 形式)

```
dn: uid=jduke,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: Java Duke
sn: duke
```

```
uid: jduke
userPassword: theduke
# ============================
dn: uid=hnelson,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: Horatio Nelson
sn: Nelson
uid: hnelson
userPassword: secret
# ============================
dn: ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: groups
# ============================
dn: uid=ldap,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: LDAP
sn: Service
uid: ldap
userPassword: randall
# ============================
dn: ou=Users,dc=jboss,dc=org
objectClass: top
objectClass: organizationalUnit
ou: Users
# ============================
dn: dc=jboss,dc=org
objectclass: top
objectclass: domain
dc: jboss
# ============================
dn: uid=GroupTwo,ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: groupOfNames
objectClass: uidObject
cn: GroupTwo
member: uid=jduke,ou=Users,dc=jboss,dc=org
uid: GroupTwo
# ============================
dn: uid=GroupThree,ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: GroupThree
uid: GroupThree
uniqueMember: uid=GroupOne,ou=groups,dc=jboss,dc=org
# ============================
dn: uid=HTTP,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
```

```
cn: HTTP
sn: Service
uid: HTTP
userPassword: httppwd
# ============================
dn: uid=GroupOne,ou=groups,dc=jboss,dc=org
objectClass: top
objectClass: groupOfUniqueNames
objectClass: uidObject
cn: GroupOne
uid: GroupOne
uniqueMember: uid=jduke,ou=Users,dc=jboss,dc=org
uniqueMember: uid=hnelson,ou=Users,dc=jboss,dc=org
```

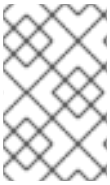
LdapExtended ログインモジュールを追加する CLI コマンド

```
/subsystem=security/security-domain=testLdapExtendedExample:add(cache-type=default)

/subsystem=security/security-domain=testLdapExtendedExample/authentication=classic:add

/subsystem=security/security-domain=testLdapExtendedExample/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://localhost:10389"),
("java.naming.security.authentication"=>"simple"),
("bindDN"=>"uid=ldap,ou=Users,dc=jboss,dc=org"), ("bindCredential"=>"randall"),
("baseCtxDN"=>"ou=Users,dc=jboss,dc=org"), ("baseFilter"=>"(uid={0})"),
("rolesCtxDN"=>"ou=groups,dc=jboss,dc=org"),     ("roleFilter"=>"(uniqueMember={1})"),
("roleAttributeID"=>"uid")])

reload
```



注記

ここで使用する管理 CLI コマンドは、JBoss EAP スタンドアロンサーバーを実行していることを仮定しています。JBoss EAP 管理対象ドメインの管理 CLI を使用する場合の詳細は『管理 CLI ガイド』を参照してください。

3.1.1.1.1. Active Directory に LdapExtended ログインモジュールを使用するようセキュリティードメインを設定

Microsoft Active Directory では、LdapExtended ログインモジュールを使用できます。

以下の例は、デフォルトの Active Directory 設定を示しています。Active Directory の設定によっては、通常のポート **389** ではなく、ポート **3268** でグローバルカタログを検索する必要がある場合があります。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。

デフォルト AD 設定用の LdapExtended ログインモジュールの設定例

```
/subsystem=security/security-domain=AD_Default:add(cache-type=default)

/subsystem=security/security-domain=AD_Default/authentication=classic:add

/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"=>"JBOSSsearchuser"),
("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")])

reload
```

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。

再帰検索を使用したデフォルトの AD 設定に対する LdapExtended ログインモジュールの設定例

```
/subsystem=security/security-domain=AD_Recursive:add(cache-type=default)

/subsystem=security/security-domain=AD_Recursive/authentication=classic:add

/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"=>"JBOSSsearchuser"), ("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")])

reload
```

3.2. データベースを使用するためのセキュリティードメインの設定

LDAP と同様に、セキュリティードメインはログインモジュールを使用して認証および承認にデータベースを使用するように設定できます。

3.2.1. Database ログインモジュール

Database ログインモジュールは、認証およびロールマッピングをサポートする JDBC ログインモジュールです。このログインモジュールは、ユーザー名、パスワード、およびロール情報がリレーショナルデータベースに格納される場合に使用されます。

このログインモジュールは、想定される形式のプリンシパルおよびロールが含まれる論理テーブルへの参照を提供して動作します。例を以下に示します。

```
Table Principals(PrincipalID text, Password text) Table Roles(PrincipalID text, Role text, RoleGroup
text)
```

Principals テーブルはユーザー **PrincipalID** を有効なパスワードに関連付けます。また、**Roles** テーブルはユーザー **PrincipalID** をそのロールセットに関連付けます。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。

テーブルは論理であるため、ログインモジュールが使用する SQL クエリーをユーザーが指定できます。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。

この概念を明確化するために、すでに宣言されているように**Principals** および **Roles** という 2 つのテーブルがあるデータベースを検討してください。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** テーブルの **echoman** というパワスワードを持つ **PrincipalID java**
- **Roles** テーブルの **RolesRoleGroup** の **Echo** という名前のロールを持つ **PrincipalID java**
- **Roles** テーブルの **CallerPrincipalRoleGroup** に **caller-java** という名前のロールを持つ **PrincipalID java**

Database ログインモジュールの設定オプションの完全リストは、JBoss EAP『**ログインモジュールのリファレンス**』の「Database ログインモジュール」を参照してください。

3.2.1.1. データベースログインモジュールを使用するようセキュリティードメインを設定

Database ログインモジュールを使用するようにセキュリティードメインを設定する前に、データソースを適切に設定する必要があります。

JBoss EAP でのデータソースの作成および設定に関する詳細は、JBoss EAP『**設定ガイド**』の「データソース管理」を参照してください。

データソースを適切に設定したら、セキュリティードメインがデータベースログインモジュールを使用するように設定できます。以下の例では、**MyDatabaseDS** という名前のデータソースが作成され、以下で構築されるデータベースで正しく設定されていることを前提としています。

```
CREATE TABLE Users(username VARCHAR(64) PRIMARY KEY, passwd VARCHAR(64))
CREATE TABLE UserRoles(username VARCHAR(64), role VARCHAR(32))
```

データベースログインモジュールを追加する CLI コマンド

```
/subsystem=security/security-domain=testDB:add

/subsystem=security/security-domain=testDB/authentication=classic:add

/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=?")])

reload
```

3.3. プロパティーファイルを使用するためのセキュリティードメインの設定

セキュリティードメインは、ログインモジュールを使用して認証および承認のアイデンティティーストアとしてファイルシステムを使用するように設定することもできます。

3.3.1. UsersRoles ログインモジュール

UsersRoles 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。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。デフォルトの username-to-password マッピングファイル名は **users.properties** で、デフォルトの username-to-roles マッピングファイル名は **roles.properties**です。

注記

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 Jakarta EE デプロイメントのクラスパス (WAR アーカイブの **WEB-INF/classes** フォルダーなど) またはサーバークラスパスの任意のディレクトリーに配置できます。

UsersRoles ログインモジュールの設定オプションの完全リストは、JBoss EAP『**Login Module Reference**』の「UsersRoles ログインモジュール」を参照してください。

3.3.1.1. UsersRoles ログインモジュールを使用するようにセキュリティードメインを設定

以下の例では、以下のファイルが作成され、アプリケーションのクラスパスで利用できることを前提としています。

- **sampleapp-users.properties**
- **sampleapp-roles.properties**

UserRoles ログインモジュールを追加する CLI コマンド

```
/subsystem=security/security-domain=sampleapp:add

/subsystem=security/security-domain=sampleapp/authentication=classic:add

/subsystem=security/security-domain=sampleapp/authentication=classic/login-
module=UsersRoles:add(code=UsersRoles,flag=required,module-options=
[("usersProperties"=>"sampleapp-users.properties"),("rolesProperties"=>"sampleapp-
roles.properties")])

reload
```

3.4. 証明書ベースの認証を使用するようにセキュリティードメインを設定する手順

JBoss EAP には、セキュリティードメインと証明書ベースの認証を使用して Web アプリケーションまたは Jakarta Enterprise Beans のセキュリティーを確保する機能があります。

重要

証明書ベースの認証を設定する前に、アプリケーションの双方向 SSL/TLS を有効化して設定する必要がありますが、これには、JBoss EAP インスタンスと、セキュリティードメインでセキュリティーを確保した Web アプリケーションまたは Jakarta Enterprise Beans の両方に X509 証明書を設定する必要があります。

証明書、トラストストア、および双方向 SSL/TLS を設定したら、証明書ベースの認証を使用するセキュリティードメインの設定、そのセキュリティードメインを使用するアプリケーションの設定、クライアント証明書を使用するクライアントの設定に進むことができます。

3.4.1. 証明書ベースの認証を使用したセキュリティードメインの作成

証明書ベースの認証を使用するセキュリティードメインを作成するには、トラストストアと Certificate ログインモジュールまたはそのサブクラスの 1 つを指定する必要があります。

トラストストアには、認証に使用される信頼できるクライアント証明書が含まれるか、クライアントの証明書の署名に使用される認証局の証明書が含まれている必要があります。ログインモジュールは、設定されたトラストストアを使用してクライアントが提示する証明書を認証するために使用されます。セキュリティードメイン全体として、認証後にロールをプリンシパルにマップする方法も指定する必要があります。Certificate ログインモジュール自体は、ロール情報をプリンシパルにマップしませんが、別のログインモジュールと組み合わせることもできます。または、Certificate ログインモジュールの 2 つのサブクラス (CertificateRoles および DatabaseCertificate) でも、認証後にロールをプリンシパルにマップできます。以下の例は、CertificateRoles ログインモジュールを使用して、証明書ベースの認証でセキュリティードメインを設定する方法を示しています。

警告

セキュリティードメインは認証を行うときに、双方向 SSL/TLS の設定時にクライアントが提示した証明書と同じ証明書を使用します。そのため、クライアントは双方向 SSL/TLS とアプリケーションまたは Jakarta Enterprise Beans を使用した証明書ベースの認証の **両方** に同じ証明書を使用する必要があります。

証明書ベースの認証を使用するセキュリティードメインの例

```
/subsystem=security/security-domain=cert-roles-domain:add

/subsystem=security/security-domain=cert-roles-domain/jsse=classic:add(truststore=
{password=secret, url="/path/to/server.truststore.jks"}, keystore={password=secret,
url="/path/to/server.keystore.jks"}, client-auth=true)

/subsystem=security/security-domain=cert-roles-domain/authentication=classic:add

/subsystem=security/security-domain=cert-roles-domain/authentication=classic/login-
module=CertificateRoles:add(code=CertificateRoles, flag=required, module-options=[
securityDomain="cert-roles-domain", rolesProperties="${jboss.server.config.dir}/cert-
roles.properties",password-stacking="useFirstPass",
verifier="org.jboss.security.auth.certs.AnyCertVerifier"])
```

注記

上記の例では、CertificateRoles ログインモジュールを使用して認証を処理し、ロールを認証済みプリンシパルにマッピングします。そのためには、**rolesProperties** 属性を使用してプロパティーファイルを参照します。このファイルは、以下の形式を使用してユーザー名およびロールを一覧表示します。

```
user1=roleA
user2=roleB,roleC
user3=
```

ユーザー名は、提供された証明書の DN として提示されます.たとえば、**CN=valid-client, OU=JBoss, O=Red Hat, L=Raleigh, ST=NC, C=US** と表示されているため、プロパティーファイルを使用する場合は、**=** などの特殊文字やスペースをエスケープする必要があります。

ロールプロパティーファイルの例

```
CN\=valid-client,\ OU\=JBoss,\ O\=Red\ Hat,\ L\=Raleigh,\ ST\=NC,\ C\=US=Admin
```

証明書の DN を表示するには、次のコマンドを実行します。

```
$ keytool -printcert -file valid-client.crt
Owner: CN=valid-client, OU=JBoss, O=Red Hat, L=Raleigh, ST=NC, C=US
...
```

3.4.2. 証明書ベースの認証で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する

他の形式の認証で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するのと同様に、**jboss-web.xml** ファイルおよび **web.xml** ファイルを適切に設定する必要があります。

Jboss-web.xml については、証明書ベースの認証に設定したセキュリティードメインへの参照を追加します。

jboss-web.xml ファイルのサンプル

```
<jboss-web>
  <security-domain>cert-roles-domain</security-domain>
</jboss-web>
```

web.xml の場合は、**<login-config>** の **<auth-method>** 属性を **CLIENT-CERT** に設定します。また、**<security-constraint>** と **<security-roles>** を定義する必要があります。

web.xml の例

```
<web-app>
  <!-- URL for secured portion of application-->
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>secure</web-resource-name>
      <url-pattern>/secure/*</url-pattern>
    </web-resource-collection>
```

```
    <auth-constraint>
      <role-name>All</role-name>
    </auth-constraint>
  </security-constraint>

  <!-- Security roles referenced by this web application -->
  <security-role>
    <description>The role that is required to log in to the application</description>
    <role-name>All</role-name>
  </security-role>

  <login-config>
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>cert-roles-domain</realm-name>
  </login-config>
</web-app>
```

3.4.3. クライアントの設定

クライアントが証明書ベースの認証でセキュア化されたアプリケーションに対して認証するには、クライアントは JBoss EAP インスタンスのトラストストアに含まれるクライアント証明書へのアクセスが必要になります。たとえば、ブラウザーを使用してアプリケーションにアクセスする場合、クライアントは信頼された証明書をブラウザーのトラストストアにインポートする必要があります。

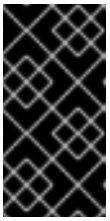
3.5. セキュリティードメインのキャッシングの設定

セキュリティードメインのキャッシュを指定して、認証チェックを高速化できます。デフォルトでは、セキュリティードメインは単純なマップをキャッシュとして使用します。このデフォルトキャッシュは、最大 1000 エントリーを持つ LRU (Least Recently Used) キャッシュです。または、Infinispan キャッシュを使用するようにセキュリティードメインを設定するか、キャッシュを完全に無効にします。

3.5.1. セキュリティードメインのキャッシュタイプの設定

要件

- セキュリティードメインが Infinispan キャッシュを使用するように設定している場合は、最初にセキュリティードメインが使用するデフォルトのキャッシュが含まれる **security** という名前の Infinispan キャッシュコンテナーを作成する必要があります。



重要

セキュリティードメインと使用する Infinispan キャッシュ設定のみを定義できます。Infinispan キャッシュを使用する複数のセキュリティードメインを使用することもできますが、各セキュリティードメインは 1 つの Infinispan キャッシュ設定から独自のキャッシュインスタンスを作成します。

キャッシュコンテナーの作成 に関する詳細は、JBoss EAP 『**設定ガイド**』を参照してください。

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のキャッシュタイプを設定できます。

- 管理コンソールを使用するには、以下の手順に従います。

  1. **Configuration → Subsystems → Security (Legacy)** の順に移動します。
  2. リストからセキュリティードメインを選択し、**View** をクリックします。
  3. **Edit** をクリックし、**Cache Type** フィールドで **default** または **infinspan** を選択します。
  4. **Save** をクリックします。

- 管理 CLI を使用する場合は、以下のコマンドを使用します。

```
/subsystem=security/security-domain=SECURITY_DOMAIN_NAME:write-
attribute(name=cache-type,value=CACHE_TYPE)
```

  たとえば、Infinispan キャッシュを使用するように **other** セキュリティードメインを設定するには、以下を実行します。

```
/subsystem=security/security-domain=other:write-attribute(name=cache-
type,value=infinispan)
```

3.5.2. プリンシパルの一覧表示およびフラッシュ

キャッシュでのプリンシパルの一覧表示

以下の管理 CLI コマンドを使用すると、セキュリティードメインのキャッシュに保存されているプリンシパルを確認できます。

```
/subsystem=security/security-domain=SECURITY_DOMAIN_NAME:list-cached-principals
```

キャッシュからのプリンシパルのフラッシュ

必要な場合は、セキュリティードメインのキャッシュからプリンシパルをフラッシュできます。

- 特定のプリンシパルをフラッシュするには、以下の管理 CLI コマンドを使用します。

```
/subsystem=security/security-domain=SECURITY_DOMAIN_NAME:flush-
cache(principal=USERNAME)
```

- キャッシュからすべてのプリンシパルをフラッシュするには、以下の管理 CLI コマンドを使用します。

```
/subsystem=security/security-domain=SECURITY_DOMAIN_NAME:flush-cache
```

3.5.3. セキュリティードメインのキャッシングの無効化

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のキャッシングを無効にできます。

- 管理コンソールを使用するには、以下の手順に従います。

  1. **Configuration → Subsystems → Security (Legacy)** の順に移動します。
  2. リストからセキュリティードメインを選択し、**View** をクリックします。
  3. **Edit** をクリックし、**Cache Type** の空白値を選択します 。

4. **Save** をクリックします。

- 管理 CLI を使用する場合は、以下のコマンドを使用します。

```
/subsystem=security/security-domain=SECURITY_DOMAIN_NAME:undefine-
attribute(name=cache-type)
```

第4章 アプリケーション設定ファイル

4.1. 認証に ELYTRON またはレガシーセキュリティーを使用するように WEB アプリケーションを設定

認証に **elytron** またはレガシー **security** サブシステムを設定した後は、アプリケーションを設定して使用する必要があります。

1. アプリケーションの **web.xml** を設定します。
   適切な認証方法を使用するようアプリケーションの **web.xml** を設定する必要があります。**elytron** サブシステムを使用する場合は、これは作成した **http-authentication-factory** に定義されています。レガシー **security** サブシステムを使用する場合、これはログインモジュールと設定する認証の種類によって異なります。

   BASIC 認証による web.xml のサンプル

   ```
   <web-app>
     <security-constraint>
       <web-resource-collection>
         <web-resource-name>secure</web-resource-name>
         <url-pattern>/secure/*</url-pattern>
       </web-resource-collection>
       <auth-constraint>
         <role-name>Admin</role-name>
       </auth-constraint>
     </security-constraint>
     <security-role>
       <description>The role that is required to log in to /secure/*</description>
       <role-name>Admin</role-name>
     </security-role>
     <login-config>
       <auth-method>BASIC</auth-method>
       <realm-name>exampleApplicationDomain</realm-name>
     </login-config>
   </web-app>
   ```

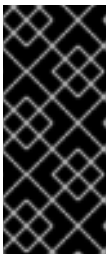
2. セキュリティードメインを使用するようにアプリケーションを設定します。
   アプリケーションの **jboss-web.xml** を設定して、認証に使用するセキュリティードメインを指定できます。**elytron** サブシステムを使用する場合、これは **application-security-domain** の作成時に定義されます。レガシー **security** サブシステムを使用する場合、これはレガシーセキュリティードメインの名前となります。

   例: jboss-web.xml

   ```
   <jboss-web>
     <security-domain>exampleApplicationDomain</security-domain>
   </jboss-web>
   ```

   Jboss-web.xml を使用すると、単一のアプリケーションに対してのみセキュリティードメインを設定できます。または、**undertow** サブシステムを使用してすべてのアプリケーションにデフォルトのセキュリティードメインを指定することもできます。これにより、**jboss-web.xml** を使用して各アプリケーションにセキュリティードメインを設定することを省略することができます。

```
/subsystem=undertow:write-attribute(name=default-security-domain,
value="exampleApplicationDomain")
```



重要

undertow サブシステムで **default-security-domain** を設定すると、**すべて**のアプリケーションに適用されます。**default-security-domain** が設定されており、アプリケーションのセキュリティードメインが **jboss-web.xml** ファイルで指定されている場合には、**jboss-web.xml** の設定が **undertow** サブシステムの **default-security-domain** よりも優先されます。

注記

Jakarta Enterprise Beans のセキュリティードメインは、**ejb3** サブシステム、**jboss-ejb3.xml** ファイルの Jakarta Enterprise Beans 記述子、または **@SecurityDomain** アノテーションを使用して Jakarta Enterprise Beans 設定で定義されます。

詳細は、『Developing Jakarta Enterprise Beans Applications』ガイドの「Jakarta Enterprise Beans Application Security」を参照してください。

サイレント BASIC 認証
elytron を設定して サイレント **BASIC** 認証を実行できます。サイレント認証を有効にすると、Web アプリケーションにアクセスする際にログインを求めるプロンプトが表示されません。代わりに代替の認証方法が使用されます。ユーザーの要求に Authorization ヘッダーが含まれる場合、**BASIC** 認証メカニズムが使用されます。

サイレント **BASIC** 認証を有効にするには、**auth-method** 属性の値を以下のように設定します。

```
<auth-method>BASIC?silent=true</auth-method>
```

Parallel での Elytron およびレガシーセキュリティーサブシステムの使用
これにより、**elytron** サブシステムとレガシーの **security** サブシステムの両方で認証を定義でき、両方のサブシステムを並行して使用できます。**undertow** サブシステムで **jboss-web.xml** と **default-security-domain** の両方を使用する場合、JBoss EAP は最初に **elytron** サブシステムで設定されたセキュリティードメインの一致を試行します。一致するものが見つからない場合、JBoss EAP はレガシー **security** サブシステムに設定されたセキュリティードメインと照合を試みます。**elytron** およびレガシー **security** サブシステムの両方に同じ名前のセキュリティードメインがある場合には、**elytron** セキュリティードメインが使用されます。

注記

1 つのセキュリティードメインを使用して Web サーブレットが定義されており、Jakarta Enterprise Beans 固有のセキュリティードメインを使用する別の EAR モジュールから Jakarta Enterprise Beans を呼び出す場合は、以下のいずれかが発生する可能性があります。

- 異なる Elytron セキュリティードメインに WAR と Jakarta Enterprise Beans をマッピングする場合に、アイデンティティーが別のデプロイメントドメインに伝播されるように、フローまたは信頼できるセキュリティードメインを設定する必要があります。この設定をしない限り、呼び出しが Jakarta Enterprise Beans に到達すると、アイデンティティーは匿名になります。認証にセキュリティーアイデンティティーを設定する方法は、「信頼されているセキュリティードメインアウトバウンドの設定」を参照してください。
- WAR と Jakarta Enterprise Beans が異なるセキュリティードメイン名を参照するにも拘らず、同じ Elytron セキュリティードメインにマップされた場合には、アイデンティティーは追加のステップなしで伝播されます。

移行時には、アプリケーション全体を移行することが推奨されます。Jakarta Enterprise Beans と WAR を別々に移行し、**elytron** サブシステムとレガシー **security** サブシステムの両方を並行して使用することは推奨されません。アプリケーションを移行して Elytron を使用する方法は、JBoss EAP『移行ガイド』の「Elytron への移行」を参照してください。

4.2. ELYTRON クライアントによるクライアント認証の設定

Jakarta Enterprise Beans などの JBoss EAP に接続するクライアントは、Elytron クライアントを使用して認証できます。Elytron クライアントは、リモートクライアントが Elytron で認証可能にするクライアント側フレームワークです。Elytron クライアントには以下のコンポーネントがあります。

認証設定の移行
: 認証設定には、ユーザー名、パスワード、許可された SASL メカニズムなどの認証情報と、ダイジェスト認証時に使用するセキュリティーレルムが含まれます。認証設定で指定される接続情報は、初期コンテキストの **PROVIDER_URL** に指定された値を上書きします。

MatchRule
: 使用する認証設定の決定に使用されるルール。

認証コンテキスト
: 接続を確立するためにクライアントで使用する一連のルールおよび認証設定。

接続が確立されると、クライアントは認証コンテキストを使用します。この認証コンテキストには、アウトバウンド接続ごとに使用する認証設定を選択するルールが含まれます。たとえば、**server1** に接続するときに、ある認証設定を使用するルールと、**server2** に接続するときに別の認証設定を使用するルールを設定できます。認証コンテキストは、一連の認証設定および接続の確立時の選択方法を定義するルールセットで構成されます。認証コンテキストは **ssl-context** を参照でき、ルールに一致できます。

接続を確立するときにセキュリティー情報を使用するクライアントを作成するには、以下を実行します。

- 1 つ以上の認証設定を作成します。
- ルールと認証設定のペアを作成して認証コンテキストを作成します。
- 接続を確立するために runnable を作成します。

- 認証コンテキストを使用して runnable を実行します。

接続を確立すると、Elytron Client は認証コンテキストによって提供されるルールセットを使用して、認証中に使用する正しい認証設定に一致させます。

クライアント接続の確立時に、以下のいずれかの方法でセキュリティー情報を使用できます。

重要

Elytron クライアントを使用して Jakarta Enterprise Beans 呼び出しを行う場合には、**javax.naming.InitialContext** の **Context.SECURITY_PRINCIPAL** 設定など、ハードコーディングされたプログラムによる認証情報が Elytron クライアント設定よりも優先されます。

4.2.1. 設定ファイルのアプローチ

設定ファイルのアプローチでは、認証設定、認証コンテキスト、および一致ルールを含む XML ファイルを作成します。

例: custom-config.xml

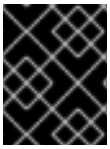
```
<configuration>
    <authentication-client xmlns="urn:elytron:client:1.2">
        <authentication-rules>
            <rule use-configuration="monitor">
                <match-host name="127.0.0.1" />
            </rule>
            <rule use-configuration="administrator">
                <match-host name="localhost" />
            </rule>
        </authentication-rules>
        <authentication-configurations>
            <configuration name="monitor">
                <sasl-mechanism-selector selector="DIGEST-MD5" />
                <providers>
                  <use-service-loader />
                </providers>
                <set-user-name name="monitor" />
                <credentials>
                    <clear-password password="password1!" />
                </credentials>
                <set-mechanism-realm name="ManagementRealm" />
            </configuration>

            <configuration name="administrator">
                <sasl-mechanism-selector selector="DIGEST-MD5" />
                <providers>
                  <use-service-loader />
                </providers>
                <set-user-name name="administrator" />
                <credentials>
                    <clear-password password="password1!" />
                </credentials>
                <set-mechanism-realm name="ManagementRealm" />
            </configuration>
```

```
        </authentication-configurations>
    </authentication-client>
</configuration>
```

クライアントの実行時にシステムプロパティーを設定すると、クライアントのコードでこのファイルを参照できます。

```
$ java -Dwildfly.config.url=/path/to/custom-config.xml ...
```



重要

programmatic approach を使用する場合、**wildfly.config.url** システムプロパティーが設定されていても、提供される設定ファイルは上書きされます。

ルールの作成時に、**hostname**、**port**、**protocol**、または **user-name** など、さまざまなパラメーターと一致するものを検索できます。**MatchRule** の完全なオプション一覧は Java ドキュメントで確認できます。ルールは、設定される順序で評価されます。

ルールに一致設定が含まれていない場合は、ルール全体が一致し、認証設定が選択されます。ルールに複数の一致設定が含まれる場合、認証設定を選択するには、すべてが一致する必要があります。

表4.1 一般的なルール

属性	説明
match-local-security-domain	一致するローカルセキュリティードメインを指定する単一の **name** 属性を取ります。
match-host	一致するホスト名を指定する単一の **name** 属性を取ります。たとえば、ホスト **127.0.0.1** は **http://127.0.0.1:9990/my/path** で一致します。
match-no-user	ユーザーのない URI に対して一致。
match-path	一致するパスを指定する単一の **name** 属性を取ります。たとえば、**/my/path/** would match on **http://127.0.0.1:9990/my/path** で一致します。
match-port	一致するポートを指定する単一の **name** 属性を取ります。たとえば、ポート **9990** は **http://127.0.0.1:9990/my/path** で一致します。
match-protocol	照合するプロトコルを指定する単一の **name** 属性を取ります。たとえば、プロトコル **http** は **http://127.0.0.1:9990/my/path** で一致します。
match-urn	照合する属性を指定する単一の **name** 属性を取ります。
match-user	一致する **user** を指定する単一の **user** 属性を取ります。

wildfly-config.xml ファイルのサンプルは、Example **wildfly-config.xml** にあります。**wildfly-config.xml** ファイルの設定方法に関する詳細は、JBoss EAP『**開発ガイド**』の「**wildfly-config.xml** ファイルを使用したクライアント設定」を参照してください。

4.2.2. プログラムによるアプローチ

プログラムによるアプローチでは、クライアントのコードにすべての Elytron クライアント設定を設定します。

```
//create your authentication configuration
AuthenticationConfiguration adminConfig =
    AuthenticationConfiguration.empty()
      .useProviders(() -> new Provider[] { new WildFlyElytronProvider() })
      .setSaslMechanismSelector(SaslMechanismSelector.NONE.addMechanism("DIGEST-MD5"))
      .useRealm("ManagementRealm")
      .useName("administrator")
      .usePassword("password1!");

//create your authentication context
AuthenticationContext context = AuthenticationContext.empty();
context = context.with(MatchRule.ALL.matchHost("127.0.0.1"), adminConfig);


//create your runnable for establishing a connection
Runnable runnable =
    new Runnable() {
      public void run() {
        try {
          //Establish your connection and do some work
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    };

//use your authentication context to run your client
context.run(runnable);
```

設定の詳細を **AuthenticationConfiguration** および **AuthenticationContext** に追加するとき、各メソッド呼び出しはそのオブジェクトの新しいインスタンスを返します。たとえば、異なるホスト名で接続する際に別の設定が必要な場合は、以下を実行できます。

```
//create your authentication configuration
AuthenticationConfiguration commonConfig =
    AuthenticationConfiguration.empty()
      .useProviders(() -> new Provider[] { new WildFlyElytronProvider() })
      .setSaslMechanismSelector(SaslMechanismSelector.NONE.addMechanism("DIGEST-MD5"))
      .useRealm("ManagementRealm");

AuthenticationConfiguration administrator =
    commonConfig
      .useName("administrator")
      .usePassword("password1!");
```

```
AuthenticationConfiguration monitor =
    commonConfig
      .useName("monitor")
      .usePassword("password1!");


//create your authentication context
AuthenticationContext context = AuthenticationContext.empty();
context = context.with(MatchRule.ALL.matchHost("127.0.0.1"), administrator);
context = context.with(MatchRule.ALL.matchHost("localhost"), monitor);
```

表4.2 一般的なルール

ルール	説明
matchLocalSecurityDomain(String name)	これは、設定ファイルアプローチの **match-domain** と同じです。
matchNoUser()	これは、設定ファイルアプローチの **match-no-user** と同じです。
matchPath(String pathSpec)	これは、設定ファイルアプローチの **match-path** と同じです。
matchPort(int port)	これは、設定ファイルアプローチの **match-port** と同じです。
matchProtocol(String protoName)	これは、設定ファイルアプローチの **match-port** と同じです。
matchPurpose(String purpose)	このルールと同じで、特定の目的名と一致する新規ルールを作成します。
matchUrnName(String name)	これは、設定ファイルアプローチの **match-urn** と同じです。
matchUser(String userSpec)	これは、設定ファイルアプローチの **match-userinfo** と同じです。

また、空の認証設定を開始する代わりに、**captureCurrent()** を使用して現在設定されている認証を開始することもできます。

```
//create your authentication configuration
AuthenticationConfiguration commonConfig = AuthenticationConfiguration.captureCurrent();
```

captureCurrent() を使用すると、以前に確立された認証コンテキストがキャプチャーされ、新しいベース設定として使用されます。認証コンテキストは、**run()**を呼び出してアクティベートすると確立されます。**captureCurrent()** が呼び出され、コンテキストが現在アクティブでない場合、利用可能な場合はデフォルト認証を試行し、使用します。詳細は、以下のセクションを参照してください。

- 設定ファイルの適用
- デフォルト設定の適用
- JBoss EAP にデプロイされたクライアントでの Elytron クライアントの使用

AuthenticationConfiguration.empty() は、上で設定を構築するベースとしてのみ使用し、それ自体では使用しないでください。JVM 全体で登録されたプロバイダーを使用し、匿名認証を有効にする設定を提供します。

AuthenticationConfiguration.empty() 設定でプロバイダーを指定する場合、カスタム一覧を指定できますが、ほとんどのユーザーは **WildFlyElytronProvider()** プロバイダーを使用する必要があります。

認証コンテキストを作成する場合は **context.with(...)** を使用することで、現在のコンテキストのルールと認証設定を、指定のルールと認証設定とマージさせる新しいコンテキストが作成されます。指定されるルールおよび認証設定は、現在のコンテキストのルールおよび認証設定の後に表示されます。

4.2.3. デフォルト設定の適用

デフォルト設定のアプローチは、Elytron Client によって提供される設定に完全に依存します。

```
//create your runnable for establishing a connection
Runnable runnable =
    new Runnable() {
      public void run() {
        try {
          //Establish your connection and do some work
        } catch (Exception e) {
          e.printStackTrace();
        }
      }
    };

// run runnable directly
runnable.run();
```

デフォルト設定を提供するために、Elytron クライアントはファイルシステム上の **wildfly-config.xml** ファイルの自動検出を試みます。以下の場所を探します。

- クライアントコード外で設定された **wildfly.config.url** システムプロパティーで指定される場所
- クラスパスルートディレクトリー
- クラスパスの **META-INF** ディレクトリー。
- 現在のユーザーのホームディレクトリー
- 現在の作業ディレクトリー。

以下の例は、クライアントの **wildfly-config.xml** ファイルの基本設定として使用できます。

基本的な wildfly-config.xml

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <authentication-rules>
      <rule use-configuration="default" />
    </authentication-rules>
    <authentication-configurations>
      <configuration name="default">
        <sasl-mechanism-selector selector="#ALL" />
        <set-mechanism-properties>
```

```
        <property key="wildfly.sasl.local-user.quiet-auth" value="true" />
      </set-mechanism-properties>
      <providers>
        <use-service-loader/>
      </providers>
    </configuration>
  </authentication-configurations>
 </authentication-client>
</configuration>
```

注記

ANONYMOUS メカニズムは、**non-anonymous** ユーザーの承認をサポートしません。そのため、**set-authorization-name** は Elytron クライアント設定ファイルの **set-anonymous** では機能しません。**set-authorization-name** を設定する場合は、承認されたアイデンティティーの **set-user-name** も指定する必要があります。

4.2.4. JBoss EAP にデプロイされたクライアントでの Elytron クライアントの使用

JBoss EAP にデプロイされたクライアントは Elytron クライアントを使用することもできます。**AuthenticationContext** は自動的に解析され、JBoss EAP 設定の **default-authentication-context** 設定から作成されます。**default-authentication-context** が設定されていないものの、デプロイメントに **wildfly-config.xml** ファイルが含まれるか、**wildfly.config.url** システムプロパティーを使用して設定されている場合、**AuthenticationContext** は自動的に解析され、そのファイルから作成されます。

例: デフォルト認証コンテキストの設定

```
/subsystem=elytron/authentication-context=AUTH_CONTEXT:add
/subsystem=elytron:write-attribute(name=default-authentication-context,value=AUTH_CONTEXT)
```

デプロイメント外で設定ファイルを読み込むには、**parseAuthenticationClientConfiguration(URI)** メソッドを使用します。このメソッドは、プログラムによるアプローチを使用してクライアントコードで使用できる programmatic approach を使用してクライアントコードで使用できる **AuthenticationContext** を返します。

さらに、クライアントは **elytron** サブシステムによって提供されるクライアント設定から **AuthenticationContext** を自動的に解析して作成します。**elytron** サブシステムのクライアント設定は、クレデンシャルストアなどの **elytron** サブシステムに定義された他のコンポーネントも利用できます。クライアント設定がデプロイメントと **elytron** サブシステムの両方によって提供される場合、**elytron** サブシステムの設定が使用されます。

注記

authentication-context が **elytron** サブシステムのデフォルトとして設定されている場合のみ、**elytron** サブシステムの **AuthenticationContext** を使用できます。

4.2.5. wildfly-config.xml ファイルを使用した Jakarta Management クライアントの設定

JBoss EAP 7.1 以降で、JConsole などの Jakarta Management クライアントは、**wildfly-config.xml** ファイルを使用して設定できます。Jakarta Management クライアントの起動時に **-Dwildfly.config.url** システムプロパティーを使用して、設定ファイルへのファイルパスを指定します。

```
-Dwildfly.config.url=path/to/wildfly-config.xml
```

注記

JConsole を使用する場合には、**-Dwildfly.config.url** システムプロパティーの前に **-J** を付ける必要があります。以下に例を示します。

```
-J-Dwildfly.config.url=path/to/wildfly-config.xml
```

詳細は、JBoss EAP『**開発ガイド**』の「**wildfly-config.xml** ファイルを使用したクライアント設定」を参照してください。

4.2.6. ElytronAuthenticator を使用したアイデンティティーの伝搬

警告

Java 8 の既知のクレデンシャル制限により、JBoss EAP での **ElytronAuthenticator** の使用はサポートされないか、または推奨されません。このクラスを使用してアイデンティティーを伝播する場合は、以下の制限に注意してください。

- Java 8 設計の制限により、保護されたサーブレットへの呼び出しではセキュリティーアイデンティティーは伝播されません。
- Jakarta Enterprise Beans などのサーバーで **ElytronAuthenticator** を使用しないでください。
- 認証情報のキャッシュは、スタンドアロンクライアントの JVM で使用する場合に影響を与える可能性があります。

JBoss EAP 7.1 には **ElytronAuthenticator** クラスが導入されました.これは現在のセキュリティーコンテキストを使用して認証を実行します。org.wildfly.security.auth.util.ElytronAuthenticator クラスは java.net.Authenticator の実装です。

- これには、新しいインスタンスを構成する **ElytronAuthenticator()** というコンストラクターがあります。
- これには、**PasswordAuthentication** インスタンスを返す **getPasswordAuthentication()** メソッドがあります。

以下は、**ElytronAuthenticator** クラスを作成して使用し、アイデンティティーをサーバーに伝播するクライアントコードの例です。

例: ElytronAuthenticator を使用したコード

```
// Create the authentication configuration
AuthenticationConfiguration httpConfig = AuthenticationConfiguration.empty().useName("bob");
```

```
// Create the authentication context
AuthenticationContext context = AuthenticationContext.captureCurrent().with(MatchRule.ALL,
httpConfig.usePassword(createPassword(httpConfig, "secret")));

String response = context.run((PrivilegedExceptionAction<String>) () -> {
    Authenticator.setDefault(new ElytronAuthenticator());
    HttpURLConnection connection = HttpURLConnection.class.cast(new URL("http://localhost:" +
SERVER_PORT).openConnection());
    try (InputStream inputStream = connection.getInputStream()) {
        return new BufferedReader(new InputStreamReader(inputStream)).lines().findFirst().orElse(null);
    }
});
```

4.3. 信頼されるセキュリティードメインのアウトフローの設定

すべてのセキュリティー呼び出しに対して、セキュリティードメインにセキュリティーアイデンティティーが確立されます。呼び出しが処理されると、**SecurityIdentity** は現在のスレッドに関連付けられます。同じセキュリティードメインの **getCurrentSecurityIdentity()** への後続の呼び出しでは、関連付けられたアイデンティティーが返されます。

アプリケーションサーバー内に、単一の呼び出しまたはスレッドに対して複数の **SecurityDomain** インスタンスが存在する可能性があります。各 **SecurityDomain** インスタンスは、異なる **SecurityIdentity** に関連付けることができます。セキュリティードメインの **getCurrentSecurityIdentity()** メソッドを呼び出しすると、正しいセキュリティーアイデンティティーが返されます。デプロイメントは、要求の処理中に他のデプロイメントを呼び出すことができます。各デプロイメントは単一のセキュリティードメインに関連付けられます。呼び出すデプロイメントが同じセキュリティードメインを使用する場合、現在のセキュリティーアイデンティティーを持つ単一セキュリティードメインの概念はそのままになります。ただし、各デプロイメントは独自のセキュリティードメインを参照できます。

次のセクションで説明されているように、セキュリティードメインに関連付けられたセキュリティーアイデンティティーを別のセキュリティードメインにインポートできます。

セキュリティーアイデンティティーのインポート

セキュリティードメインから別のセキュリティードメインにセキュリティーアイデンティティーをインポートして、このドメインのセキュリティーアイデンティティーを取得する場合、ほとんどの場合に 3 つの処理フローがあります。

同じセキュリティードメイン

セキュリティードメインは、独自のセキュリティー ID を常にインポートできます。この場合、セキュリティードメインは常に自身を信頼します。

一般的なセキュリティーレルム

インポートプロセス時に、セキュリティードメインはインポートされるセキュリティーアイデンティティーからプリンシパルを取得し、設定されたプリンシパルトランスフォーマーおよびレルムマッパー経由でそれを渡して、そのセキュリティードメイン内のアイデンティティーにマップします。アイデンティティーを作成したセキュリティードメインで使用されたのと同じセキュリティーレルムがセキュリティードメイン内で使用されている場合は、どちらも同じ基礎となるアイデンティティーによってサポートされ、インポートが許可されます。

信頼されているセキュリティードメイン

アイデンティティーが正常にマッピングされ、共通のセキュリティーレルムがない場合、インポートを処理するセキュリティードメインは、元のセキュリティードメインを信頼するかどうかをテストします。これが実行されると、インポートが許可されます。

注記

アイデンティティーは、インポートを処理するセキュリティードメインに存在する必要があります。セキュリティーアイデンティティー自体は信頼されません。

Outflow
セキュリティードメインは、セキュリティー ID を別のセキュリティードメインに自動的に移動するように設定できます。

セキュリティードメインでは、セキュリティーアイデンティティーが確立され、現在の呼び出しに使用される場合に、アウトフローのセキュリティードメインのリストが反復され、セキュリティードメインごとにセキュリティーアイデンティティーがインポートされます。

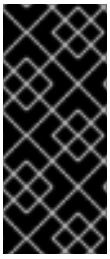
このモデルは、Web アプリケーションが共通のセキュリティードメインを使用して 5 つの異なる Jakarta Enterprise Beans. を呼び出す場合など、異なるセキュリティードメインを使用してデプロイメントに複数の呼び出しを行う場合に、より適しています。

第5章 LDAP での管理インターフェースのセキュリティー保護

管理インターフェースは、LDAP サーバー (Microsoft Active Directory を含む) に対して認証できます。これは、LDAP authenticator を使用して実行できます。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の接続 (アウトバウンド LDAP 接続を使用) 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。成功すると、クレデンシャルおよびユーザーによって提供されるパスワードとしてユーザーの DN で、LDAP サーバーへの新しい接続が確立されます。この 2 つ目の接続と 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、認証に成功しました。

注記

LDAP による管理インターフェースのセキュリティー保護により、認証がダイジェストから BASIC/Plain に変更されます。これによりデフォルトで、ネットワーク上にユーザー名とパスワードが暗号化されずに送信されます。アウトバウンド接続で SSL/TLS を有効にして、このトラフィックを暗号化し、この情報をクリアで送信しないようにすることができます。



重要

LDAP を使用した管理インターフェースのセキュア化など、レガシーセキュリティーレルムが LDAP サーバーを使用して認証を実行する場合、JBoss EAP は **500** または内部サーバーエラー (LDAP サーバーにアクセスできない場合) を返します。この動作は、同じ条件下で、**401**、または未承認のエラーコードを返す JBoss EAP の以前のバージョンとは異なります。

5.1. ELYTRON の使用

LDAP と **elytron** サブシステムを使用して、管理インターフェースをアイデンティティーストアと同じようにセキュアにすることができます。**elytron** サブシステムとのセキュリティーにアイデンティティーストアを使用する方法は、『**サーバーセキュリティーの設定方法**』の「新規アイデンティティーストアでの管理インターフェースのセキュア保護」を参照してください。たとえば、管理コンソールを LDAP でセキュアにするには、以下を実行します。

注記

Active Directory LDAP サーバーが使用される場合など、JBoss EAP サーバーにパスワードを読み取るパーミッションがない場合は、定義された LDAP レルム上で **direct-verification** を **true** に設定する必要があります。この属性を使用すると、JBoss EAP サーバーではなく LDAP サーバー上で直接検証を実行できます。

LDAP アイデンティティーストアの例

```
/subsystem=elytron/dir-context=exampleDC:add(url="ldap://127.0.0.1:10389",principal="uid=admin,ou=system",credential-reference={clear-text="secret"})

/subsystem=elytron/ldap-realm=exampleLR:add(dir-context=exampleDC,identity-mapping={search-base-dn="ou=Users,dc=wildfly,dc=org",rdn-identifier="uid",user-password-mapper={from="userPassword"},attribute-mapping=[{filter-base-dn="ou=Roles,dc=wildfly,dc=org",filter="(&(objectClass=groupOfNames)(member={0}))",from="cn",to="Roles"}]})

/subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
```

```
/subsystem=elytron/security-domain=exampleLdapSD:add(realms=[{realm=exampleLR,role-decoder=from-roles-attribute}],default-realm=exampleLR,permission-mapper=default-permission-mapper)

/subsystem=elytron/http-authentication-factory=example-ldap-http-auth:add(http-server-mechanism-factory=global,security-domain=exampleLdapSD,mechanism-configurations=[{mechanism-name=BASIC,mechanism-realm-configurations=[{realm-name=exampleApplicationDomain}]}])

/core-service=management/management-interface=http-interface:write-attribute(name=http-authentication-factory, value=example-ldap-http-auth)

reload
```

5.1.1. アウトバウンド LDAP 接続での双方向 SSL/TLS の Elytron の使用

LDAP を使用して管理インターフェースをセキュアにする場合、双方向 SSL/TLS を使用するようアウトバウンド LDAP 接続を設定できます。これを行うには、**ssl-context** を作成し、**ldap-realm** が使用する **dir-context** に追加します。双方向 SSL/TLS **ssl-context** の作成は、『**サーバーセキュリティーの設定方法**』の「Elytron サブシステムを使用してアプリケーションに対して双方向 SSL/TLS を有効化する手順」を参照してください。

警告

Red Hat では、影響す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 SSLv2、SSLv3、および TLSv1.0 を明示的に無効化することを推奨しています。

5.2. レガシーのコア管理認証の使用

LDAP ディレクトリーサーバーをレガシー **security** サブシステムを使用して管理インターフェースの認証ソースとして使用するには、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。
   アウトバウンド LDAP 接続を作成する目的は、セキュリティーレルム (および JBoss EAP インスタンス) が LDAP サーバーへの接続を確立できるようにすることです。これは、セキュリティードメインの **Database** ログインモジュールで使用するデータソースを作成する場合と同様です。

   LDAP アウトバウンド接続は以下の属性を許可します。

属性	必須	説明
url	○	ディレクトリーサーバーの URL アドレス。
search-dn	×	検索の実行が許可されているユーザーの完全識別名 (DN)。

属性	必須	説明
search-credential	×	検索を実行する権限のあるユーザーのパスワード。この要素によってサポートされる属性は次のとおりです。 ● **store**: 検索認証情報の取得元のクレデンシャルストアへの参照。 ● **alias**: 参照ストア内の認証情報のエイリアス。 ● **type**: クレデンシャルストアから取得する認証情報タイプの完全修飾クラス名。 ● **clear-text**: クレデンシャルストアを参照する代わりに、この属性を使用してクリアテキストのパスワードを指定できます。
initial-context-factory	×	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	×	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。
参考資料	×	検索時に参照が発生する場合の動作を指定します。有効なオプションは **IGNORE**、**FOLLOW**、および **THROW** です。 ● **ignore**: デフォルトのオプションです。リファレンスを無視します。 ● **FOLLOW**: 検索時に参照が見つかると、**DirContext** はその参照のフォローを試みます。ここでは、別のサーバーに接続するために同じ接続設定が使用できることを前提とし、参照機能で使用されている名前に到達できることを前提としています。 ● **THROW**: **DirContext** は例外 **LdapReferralException** をスローし、参照が必要であることを示します。セキュリティーレルムは、参照に使用する代替接続を処理し、特定しようとします。

属性	必須	説明
always-send-client-cert	×	デフォルトでは、ユーザーの認証情報を確認する際に、サーバーのクライアント証明書は送信されません。これを **true** に設定すると、常に送信されます。
handles-referrals-for	×	接続を処理できる参照情報を指定します。URI の一覧を指定する場合、それらはスペースで区切ります。これにより、複数の異なる認証情報をたどる必要がある場合に、接続プロパティーを持つ接続が定義され、使用されます。これは、別の認証情報が 2 つ目サーバーに対して認証する必要がある場合や、JBoss EAP インストールから到達できない参照表現にサーバーが名前を返し、代替のアドレスを送信できる場合に役に立ちます。

注記

search-dn および **search-credential** は、ユーザーが提供するユーザー名とパスワードとは異なります。ここで提供される情報は、JBoss EAP インスタンスと LDAP サーバー間の初期接続を確立するために特に提供されます。この接続により、JBoss EAP は認証を試みるユーザーの DN を後続の検索することができます。検索の結果となるユーザーの DN。この DN は認証を試み、入力したパスワードは認証プロセスを完了する際の別の接続を確立するために使用されます。

以下は LDAP サーバーの例になりますが、アウトバウンド LDAP 接続を設定する管理 CLI コマンドになります。

表5.1 LDAP サーバーの例

属性	値
url	127.0.0.1:389
search-credential	myPass
search-dn	cn=search,dc=acme,dc=com

アウトバウンド接続追加の CLI

```
/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")

reload
```

注記

これにより、JBoss EAP インスタンスと LDAP サーバー間に暗号化されていない接続が作成されます。SSL/TLS を使用して暗号化された接続を設定する方法は、Using SSL/TLS for the Outbound LDAP Connection を参照してください。

2. LDAP 対応のセキュリティーレルムを新規作成します。
アウトバウンド LDAP 接続を作成したら、使用する新しい LDAP 対応セキュリティーレルムを作成する必要があります。

   LDAP セキュリティーレルムには、以下の設定属性があります。

属性	説明
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。
base-dn	ユーザーの検索を開始するコンテキストの DN です。
recursive	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	DN を保持するユーザーの属性です。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。
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。デフォルト値は **false** 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。
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。この属性には、標準の LDAP 構文のフィルタークエリーが含まれます。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。詳細および **advanced-filter** の例は、「認証の LDAP と RBAC の組み合わせ」を参照してください。

警告

空の LDAP パスワードは、セキュリティ上の懸念が大きいため許可されていませんこの動作が環境内で特に必要でない限り、空のパスワードは許可されず、**allow-empty-passwords** は false のままになるようにしてください。

以下は、**ldap-connection** アウトバウンド LDAP 接続を使用して LDAP が有効なセキュリティーレルムを設定する管理 CLI コマンドです。

```
/core-service=management/security-realm=ldap-security-realm:add

/core-service=management/security-realm=ldap-security-realm/authentication=ldap:add(connection="ldap-connection", base-dn="cn=users,dc=acme,dc=com",username-attribute="sambaAccountName")

reload
```

3. 管理インターフェースの新しいセキュリティーレルムを参照します。セキュリティーレルムが作成され、アウトバウンド LDAP 接続を使用している場合、新しいセキュリティーレルムは管理インターフェースで参照される必要があります。

```
/core-service=management/management-interface=http-interface/:write-attribute(name=security-realm,value="ldap-security-realm")
```

注記

ここで使用する管理 CLI コマンドは、JBoss EAP スタンドアロンサーバーを実行していることを仮定しています。JBoss EAP 管理対象ドメインの管理 CLI を使用する場合の詳細は『管理 CLI ガイド』を参照してください。

5.2.1. アウトバウンド LDAP 接続に双方向 SSL/TLS を使用する

以下の手順に従って、SSL/TLS でセキュア化されたアウトバウンド LDAP 接続を作成します。

警告

Red Hat では、影響す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 SSLv2、SSLv3、および TLSv1.0 を明示的に無効化することを推奨しています。

1. 使用するアウトバウンド LDAP 接続のセキュリティーレルムを設定します。
セキュリティーレルムには、JBoss EAP サーバーがそれ自体と LDAP サーバー間の通信の復号化/暗号化に使用するキーで設定されたキーストアが含まれる必要があります。このキーストア

により、JBoss EAP インスタンスは LDAP サーバーに対して自己検証することもできます。セキュリティーレルムには、LDAP サーバーの証明書が含まれるトラストストア、または LDAP サーバーの証明書の署名に使用する認証局の証明書が含まれる必要もあります。キーストアとトラストストアの設定とこれらを使用するセキュリティーレルムの作成方法については、JBoss EAP『**サーバーセキュリティーの設定方法**』の「管理インターフェース向けの双方向 SSL/TLS の設定」を参照してください。

2. SSL/TLS URL およびセキュリティーレルムを使用してアウトバウンド LDAP 接続を作成します。
Using Legacy Core Management Authentication で定義されてりうプロセスと同様に、LDAP サーバーの SSL/TLS URL と SSL/TLS セキュリティーレルムを使用してアウトバウンド LDAP 接続を作成する必要があります。

   LDAP サーバーのアウトバウンド LDAP 接続および SSL/TLS セキュリティーレルムを作成したら、その情報でアウトバウンド LDAP 接続を更新する必要があります。

   SSL/TLS URL でアウトバウンド接続を追加する CLI の例

   ```
   /core-service=management/ldap-connection=ldap-connection/:add(search-credential=myPass, url=ldaps://LDAP_HOST:LDAP_PORT, search-dn="cn=search,dc=acme,dc=com")
   ```

   SSL/TLS 証明書を使用したセキュリティーレルムの追加

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

   reload
   ```

3. 管理インターフェースで使用するためにアウトバウンド LDAP 接続を使用する新しいセキュリティーレルムを作成します。
Using Legacy Core Management Authentication の手順の **Create a new LDAP-Enabled Security Realm** および **Reference the new security realm in the Management Interface**の手順に従います。

注記

ここで使用する管理 CLI コマンドは、JBoss EAP スタンドアロンサーバーを実行していることを仮定しています。JBoss EAP 管理対象ドメインの管理 CLI を使用する場合の詳細は『**管理 CLI ガイド**』を参照してください。

5.3. LDAP および 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 (ロール) を指定するメカニズムです。これにより、ユーザーは、無制限の完全なアクセスを許可することなく、異なる管理責任を付与できます。RBAC の詳細は、JBoss EAP『**セキュリティーアーキテクチャー**』の「ロールベースのアクセス制御」を参照してください。

RBAC は承認に使用され、認証は個別に処理されます。LDAP は認証および承認に使用できるため、JBoss EAP は以下の方法で設定できます。

- RBAC は承認専用に使用し、認証にのみ LDAP または別のメカニズムを使用します。
- RBAC と LDAP を組み合わせて、管理インターフェースで承認決定を行います。

5.3.1. LDAP および RBAC の使用による非依存

JBoss EAP では、認証および承認をセキュリティーレルムで個別に設定できます。これにより、LDAP を認証方法として設定でき、RBAC を承認メカニズムとして設定できます。この方法で設定された場合、ユーザーが管理インターフェースへのアクセス試行すると、最初に設定された LDAP サーバーを使用して認証されます。成功すると、LDAP サーバーにあるグループ情報に関係なく、そのロールのユーザーロールおよび設定されたパーミッションは RBAC のみを使用して決定されます。

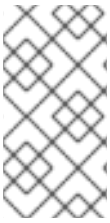
RBAC を管理インターフェースの承認メカニズムとして使用する方法は、JBoss EAP『**サーバーセキュリティーの設定方法**』を参照してください。管理インターフェースを使用した認証に LDAP を設定する方法は、前回のセクション を参照してください。

5.3.2. LDAP と RBAC の承認の統合

LDAP サーバーを使用して認証したユーザーまたはプロパティーファイルを使用したユーザーは、ユーザーグループのメンバーになります。ユーザーグループは、単一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 は、このグループ情報を使用してロールをユーザーに自動的に割り当てるか、ロールからユーザーを除外するように設定できます。

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 サーバー設定によっては、ユーザーエンティティーはユーザーが所属するグループを **memberOf** 属性を介してマップできます。また、グループエンティティーは、**uniqueMember** 属性によってユーザーが属するユーザーをマップできます。ユーザーが LDAP サーバーに正常に認証されると、グループ検索が実行され、そのユーザーのグループ情報が読み込まれます。使用中のディレクトリーサーバーによっては、グループ検索は SN を使用して実行できます。これは通常、認証で使用されるユーザー名か、ディレクトリー内のユーザーのエントリー DN を使用して実行できます。LDAP をセキュリティーレルムの承認メカニズムとして設定すると、グループ検索 (**username-to-dn**) およびユーザー名と識別名 (**username-to-dn**) 間のマッピングが設定されます。

ユーザーのグループメンバーシップ情報が LDAP サーバーから決定されると、RBAC 設定内のマッピングを使用して、ユーザーが所有するロールが決定されます。このマッピングは、グループおよび個々のユーザーを明示的に包含または除外するように設定されます。

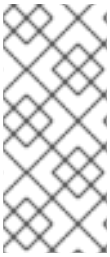


注記

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。セキュリティーレルムは、グループの読み込みステップで認証接続を再利用してこのプロセスを最適化します。

5.3.2.1. group-search の使用

グループメンバーシップ情報を検索する場合に使用できるスタイルには、**Principal to Group** および **Group to Principal** があります。Principal to Group には、 **memberOf** 属性を使用した、メンバーとなっているグループへの参照が含まれるユーザーのエントリーがあります。Group to Principal には、**uniqueMember** 属性を使用して、そのグループのメンバーであるユーザーへの参照が含まれるグループのエントリーがあります。

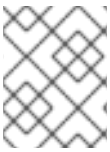


注記

JBoss EAP は Principal と Group to Principal の両方をサポートしますが、Group to Principal で Principal を使用することが推奨されます。グループへのプリンシパルが使用される場合、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、グループ情報を直接読み込むことができます。Group to Principal では、ユーザーを参照するすべてのグループを特定するために、幅広い検索が必要になります。

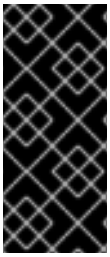
Principal to Group と Group to Principal はいずれも、以下の属性を含む **group-search** を使用します。

属性	説明
group-name	この属性は、ユーザーがメンバーとなるグループの一覧として返されたグループ名に使用され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、ユーザーがメンバーとなっているグループを特定した後、そのグループがメンバーとなっているグループを特定するために、グループに基づいて繰り返し検索を行う必要があるかどうかを示します。反復検索が有効になっていると、他のグループまたはサイクルが検出された場合に、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。
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	属性が単純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。



注記

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

反復検索が機能するには、グループエントリーがユーザーエントリーと同じである必要があります。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、グループメンバーシップ、クロス参照に使用される属性名の変更、または参照方向が変更された場合は利用できません。

グループ検索の Principal to Group (memberOf)

たとえば、**GroupOne** のメンバーの **TestUserOne** ユーザーと、**GroupOne** が次に **GroupFive** のメンバーとなる場合を考えます。グループメンバーシップは、メンバーレベルで **memberOf** 属性を使用することで表示されます。これは、**TestUserOne** によって、**memberOf** 属性が **GroupOne** の **dn** に設定されることになります。**GroupOne** は次に **memberOf** 属性を **GroupFive** の **dn** に設定します。

このタイプの検索を使用するために、**principal-to-group** 要素が **group-search** 要素に追加されます。

Principal to Group、 memberOf、 設定

```
/core-service=management/security-realm=ldap-security-realm:add
```

```
batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-realm/authorization=ldap/group-
search=principal-to-group:add(group-attribute="memberOf",iterative=true,group-dn-attribute="dn",
group-name="SIMPLE",group-name-attribute="cn")

run-batch
```

重要

上記の例では、**ldap-connection** がすでに定義されていることを前提としています。また、このセクションの前半で説明した認証メカニズムも設定する必要があります。

group-attribute 属性が **group-search=principal-to-group**. と使用されていることに注意してください。参考情報:

表5.2 principal-to-group

属性	説明
group-attribute	ユーザーがメンバーとなってい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の名前。デフォルトは **memberOf** です。
prefer-original-connection	この値は、参照に従う際に優先するグループ情報を示します。プリンシパルがロードされるたびに、各グループメンバーシップの属性がその後にロードされます。属性がロードされるたびに、最後の参照情報からの元の接続または接続のいずれかを使用できます。デフォルト値は **true** です。

Group to Principal, uniqueMember, Group Search

Principal to Group と同じ例を考えてみましょう。ここでは、**GroupOne** のメンバーであるユーザー**TestUserOne**、次に **GroupOne** が **GroupFive** のメンバーです。ただし、この場合、グループメンバーシップはグループレベルで設定された **uniqueMember** 属性を使用することで表示されます。これは **GroupFive** によって、**uniqueMember** が **GroupOne** の **dn** に設定されていることを意味します。**GroupOne** は次に、**uniqueMember** を **TestUserOne** の **dn** に設定します。

このタイプの検索を使用するために、**group-to-principal** 要素が **group-search** 要素に追加されます。

Group to Principal、uniqueMember、Configuration

```
/core-service=management/security-realm=ldap-security-realm:add

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)
```

```
/core-service=management/security-realm=ldap-security-realm/authorization=ldap/group-search=group-to-principal:add(iterative=true, group-dn-attribute="dn", group-name="SIMPLE", group-name-attribute="uid", base-dn="ou=groups,dc=group-to-principal,dc=example,dc=org", principal-attribute="uniqueMember", search-by="DISTINGUISHED_NAME")

run-batch
```

重要

上記の例では、**ldap-connection** がすでに定義されていることを前提としています。また、このセクションの前半で説明した認証メカニズムも設定する必要があります。

principal-attribute 属性は **group-search=group-to-principal** と使用されることに注意してください。**group-to-principal** は、ユーザーエントリーを参照するグループを検索する方法を定義するために使用されます。**principal-attribute** は、プリンシパルを参照するグループエントリーを定義するために使用されます。

参考情報:

表5.3 group-to-principal

属性	説明
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。
prefer-original-connection	この値は、参照に従う際に優先するグループ情報を示します。プリンシパルがロードされるたびに、各グループメンバーシップの属性がその後にロードされます。属性がロードされるたびに、最後の参照情報からの元の接続または接続のいずれかを使用できます。

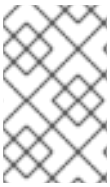
表5.4 membership-filter

属性	説明
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。

5.3.2.2. username-to-dn の使用

承認セクション内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。**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**任意**で、以下のいずれかが true の場合のみ必要になります。

- 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。
- グループ検索は、識別名を使用します。



注記

これは、セキュリティーレルムが LDAP および Kerberos 認証の両方をサポートし、認証中に検出された DN を実行している場合は、Kerberos に変換が必要な場合でも適用可能です。

これには、以下の属性が含まれます。

表5.5 username-to-dn

属性	説明
force	force 属性が **false** に設定されている場合は、認証中にユーザー名から識別名マッピングの検索がキャッシュされ、承認クエリー時に再利用されます。force が **true** の場合は、グループの読み込み中に承認中に検索が再度実行されます。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。

username-to-dn は以下のいずれかで設定できます。

username-is-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-is-dn の例

```
/core-service=management/security-realm=ldap-security-realm:add

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-realm/authorization=ldap/group-
search=group-to-principal:add(iterative=true, group-dn-attribute="dn", group-name="SIMPLE",
group-name-attribute="uid", base-dn="ou=groups,dc=group-to-principal,dc=example,dc=org",
principal-attribute="uniqueMember", search-by="DISTINGUISHED_NAME")

/core-service=management/security-realm=ldap-security-realm/authorization=ldap/username-to-
dn=username-is-dn:add(force=false)

run-batch
```

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

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-filter の例

```
/core-service=management/security-realm=ldap-security-realm:add

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-realm/authorization=ldap/group-
search=group-to-principal:add(iterative=true, group-dn-attribute="dn", group-name="SIMPLE",
group-name-attribute="uid", base-dn="ou=groups,dc=group-to-principal,dc=example,dc=org",
principal-attribute="uniqueMember", search-by="DISTINGUISHED_NAME")

/core-service=management/security-realm=ldap-security-realm/authorization=ldap/username-to-
dn=username-filter:add(force=false, base-dn="dc=people,dc=harold,dc=example,dc=com",
recursive="false", attribute="sn", user-dn-attribute="dn")

run-batch
```

属性	説明
base-dn	検索を開始するコンテキストの識別名。
recursive	検索がサブコンテキストに広がるかどうか。デフォルトは **false** です。
attribute	提供されたユーザー名と照合するユーザーのエントリーの属性。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取る属性。デフォルトは **5** です。

advanced-filter

このオプションは、カスタムフィルターを使用して、ユーザーの識別名を特定します。

advanced-filter の例

```
/core-service=management/security-realm=ldap-security-realm:add

batch

/core-service=management/security-realm=ldap-security-
realm/authorization=ldap:add(connection=ldap-connection)

/core-service=management/security-realm=ldap-security-realm/authorization=ldap/group-
search=group-to-principal:add(iterative=true, group-dn-attribute="dn", group-name="SIMPLE",
group-name-attribute="uid", base-dn="ou=groups,dc=group-to-principal,dc=example,dc=org",
```

```
principal-attribute="uniqueMember", search-by="DISTINGUISHED_NAME")

/core-service=management/security-realm=ldap-security-realm/authorization=ldap/username-to-dn=advanced-filter:add(force=true, base-dn="dc=people,dc=harold,dc=example,dc=com", recursive="false", user-dn-attribute="dn",filter="sAMAccountName={0}")

run-batch
```

username-filter の例にある属性に一致する属性の場合、意味とデフォルト値は同じです。追加の属性があります。

属性	説明
filter	ユーザー名が **{0}** プレースホルダーで置換されるユーザーのエントリーの検索に使用されるカスタムフィルター。

重要

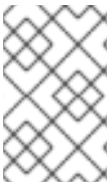
特殊文字 (**&** など) を使用する場合は、適切な形式が使用されていることを確認してください。たとえば **&** 文字の場合は **&** です。

5.3.2.3. LDAP グループ情報の RBAC ロールへのマッピング

LDAP サーバーへの接続が作成され、グループ検索が適切に設定されたら、LDAP グループと RBAC ロール間のマッピングを作成する必要があります。このマッピングは包括的かつ排他的で可能で、ユーザーはグループメンバーシップに基づいて自動的にロールを割り当てることができます。

警告

RBAC が設定されていない場合は、注意が必要です。、特に新規に作成された LDAP 対応レルムに切り替える場合は十分に注意してください。ユーザーとロールを適切に設定せずに RBAC を有効にすると、管理者が JBoss EAP 管理インターフェースにログインできなくなることがあります。



注記

ここで使用する管理 CLI コマンドは、JBoss EAP スタンドアロンサーバーを実行していることを仮定しています。JBoss EAP 管理対象ドメインの管理 CLI を使用する場合の詳細は『管理 CLI ガイド』を参照してください。

RBAC が有効になっており、設定されていることの確認

LDAP と RBAC ロール間のマッピングを使用するには、RBAC を有効化し、初期設定する必要があります。

```
/core-service=management/access=authorization:read-attribute(name=provider)
```

以下の結果が出されます。

```
{ "outcome" => "success", "result" => "rbac" }
```

RBAC の有効化および設定の詳細は、JBoss EAP『**サーバーセキュリティーの設定方法**』の「ロールベースアクセス制御の有効化」を参照してください。

既存のロール一覧の確認

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)
```

ロールの一覧を作成するもの

```
{
  "outcome" => "success",
  "result" =>
    [ "Administrator", "Deployer", "Maintainer", "Monitor", "Operator", "SuperUser" ]
}
```

また、ロールの既存のすべてのマッピングを確認できます。

```
/core-service=management/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"
      }
    }
  }
}
```

Role-Mapping エントリーの設定

ロールに **Role-Mapping** エントリーがない場合は、これを作成する必要があります。例:

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

```
{
  "outcome" => "failed",
  "failure-description" => "WFLYCTL0216: Management resource '[ (\"core-service\" =>
\"management\"), (\"access\" => \"authorization\"), (\"role-mapping\" => \"Auditor\") ]' not found"
}
```

ロールマッピングを追加する:

```
/core-service=management/access=authorization/role-mapping=Auditor:add()
```

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

検証する:

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

```
{
  "outcome" => "success",
  "result" => {
    "include-all" => false,
    "exclude" => undefined,
    "include" => undefined
  }
}
```

除外および除外のロールへのグループの追加

ロールに含める、またはロールから除外するためにグループを追加できます。

注記

除外マッピングが優先されるか、または包含マッピングが優先されます。

包含にグループを追加:

```
/core-service=management/access=authorization/role-mapping=Auditor/include=group-
GroupToInclude:add(name=GroupToInclude, type=GROUP)
```

除外するグループを追加:

```
/core-service=management/access=authorization/role-mapping=Auditor/exclude=group-
GroupToExclude:add(name=GroupToExclude, type=GROUP)
```

結果の確認:

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

```
{
  "outcome" => "success",
  "result" => {
    "include-all" => false,
    "exclude" => {
      "group-GroupToExclude" => {
        "name" => "GroupToExclude",
        "realm" => undefined,
        "type" => "GROUP"
      }
    },
    "include" => {
      "group-GroupToInclude" => {
        "name" => "GroupToInclude",
        "realm" => undefined,
        "type" => "GROUP"
      }
    }
  }
}
```

RBAC ロールグループの除外または包含からのグループの削除

グループを追加から削除するには、以下を使用します。

```
/core-service=management/access=authorization/role-mapping=Auditor/include=group-
GroupToInclude:remove
```

除外からグループを削除するには、以下を使用します。

```
/core-service=management/access=authorization/role-mapping=Auditor/exclude=group-
GroupToExclude:remove
```

5.4. キャッシュの有効化

セキュリティーレルムは、認証とグループローディングの両方で LDAP クエリーの結果をキャッシュする機能も提供します。これにより、特定の状況で複数の検索で異なるクエリーの結果を再利用できます (たとえば、グループのグループメンバーシップ情報を繰り返しクエリーするなど)。利用できるキャッシュは 3 種類あり、それぞれは個別に設定され、独立して動作します。

- authentication
- group-to-principal
- username-to-dn

5.4.1. キャッシュの設定

キャッシュが相互に独立している場合でも、3 つすべてのキャッシュは同じ方法で設定されます。各キャッシュは以下の設定オプションを提供します。

属性	説明
type	これは、キャッシュが準拠するエビクションストラテジーを定義します。オプションは **by-access-time** および **by-search-time** です。**by-access-time** は、最後のアクセスから一定期間経過した後にキャッシュから項目をエビクトします。**by-search-time** は、最後のアクセスに関係なく、キャッシュにある期間に基づいて項目をエビクトします。
eviction-time	これはストラテジーに応じてエビクションに使用される時間 (秒単位) を定義します。
cache-failures	これは、失敗した検索のキャッシングを有効/ 無効にするブール値です。これにより、LDAP サーバーが、失敗した同じ検索によって繰り返しアクセスされてしまうのを防げる可能性がありますが、存在しないユーザーの検索でキャッシュがいっぱいに成ってしまう可能性もあります。この設定は、認証キャッシュで特に重要になります。
max-cache-size	これは、キャッシュの最大サイズ (項目数) を定義します。これは、アイテムがエビクトされるタイミングを指定します。古い項目は、新規認証のスペースを確保するためにキャッシュからエビクトされ、必要に応じて検索が行われます。つまり、**max-cache-size** は新規認証の試行や検索を阻止しません。

5.4.2. 例

注記

この例では、**LDAPRealm** という名前のセキュリティーレルムが作成されていることを前提とします。これは既存の LDAP サーバーに接続し、認証および承認用に設定されます。現在の設定を表示するコマンドは、Reading the Current Cache Configuration で詳細に説明されています。LDAP を使用するセキュリティーレルムの作成に関する詳細は、Using Legacy Core Management Authentication を参照してください。

ベース設定の例

```
"core-service" : {
  "management" : {
    "security-realm" : {
      "LDAPRealm" : {
        "authentication" : {
          "ldap" : {
            "allow-empty-passwords" : false,
            "base-dn" : "...",
            "connection" : "MyLdapConnection",
            "recursive" : false,
            "user-dn" : "dn",
```

```
            "username-attribute" : "uid",
            "cache" : null
          }
        },
        "authorization" : {
          "ldap" : {
            "connection" : "MyLdapConnection",
            "group-search" : {
              "group-to-principal" : {
                "base-dn" : "...",
                "group-dn-attribute" : "dn",
                "group-name" : "SIMPLE",
                "group-name-attribute" : "uid",
                "iterative" : true,
                "principal-attribute" : "uniqueMember",
                "search-by" : "DISTINGUISHED_NAME",
                "cache" : null
              }
            },
            "username-to-dn" : {
              "username-filter" : {
                "attribute" : "uid",
                "base-dn" : "...",
                "force" : false,
                "recursive" : false,
                "user-dn-attribute" : "dn",
                "cache" : null
                }
              }
            }
          },
        }
      }
    }
  }
}
```

"cache" : null が表示されるすべてのエリアで、キャッシュを設定できます。

認証

認証時に、ユーザーの識別名はこの定義を使用して検出されます。また、LDAP サーバーへの接続が試行され、その ID がこれらの認証情報を使用して作成されます。

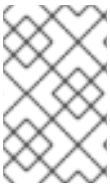
group-search 定義

グループ検索定義があります。この場合、上記のサンプル設定では **iterative** が **true** に設定されているため、反復検索になります。まず、ユーザーが直接メンバーとなっているすべてのグループを見つける検索が実行されます。その後、これらの各グループに検索が実行され、他のグループにメンバーシップがあるかどうかが特定されます。このプロセスは、cyclic 参照が検出されるまで、または最終グループが他のグループのメンバーではないまで継続されます。

グループ検索の username-to-dn 定義

グループ検索は、ユーザーの識別名の可用性に依存します。このセクションはすべての状況で使用されるわけではありませんが、ユーザーの識別名の検出試行に使用することができます。これは、ローカル認証など、別の形式の認証がサポートされている場合に役立ちます。

5.4.2.1. 現在のキャッシュ設定の読み取り



注記

本セクションおよび後続のセクションで使用される CLI コマンドは、セキュリティーレルムの名前に **LDAPRealm** を使用します。これは、設定する実際のレルムの名前に置き換えてください。

現在のキャッシュ設定を読み取る CLI コマンド

```
/core-service=management/security-realm=LDAPRealm:read-resource(recursive=true)
```

出力

```
{
  "outcome" => "success",
  "result" => {
    "map-groups-to-roles" => true,
    "authentication" => {
      "ldap" => {
        "advanced-filter" => undefined,
        "allow-empty-passwords" => false,
        "base-dn" => "dc=example,dc=com",
         "connection" => "ldapConnection",
         "recursive" => true,
         "user-dn" => "dn",
         "username-attribute" => "uid",
         "cache" => undefined
       }
     },
     "authorization" => {
       "ldap" => {
         "connection" => "ldapConnection",
         "group-search" => {
           "principal-to-group" => {
             "group-attribute" => "description",
             "group-dn-attribute" => "dn",
             "group-name" => "SIMPLE",
             "group-name-attribute" => "cn",
             "iterative" => false,
             "prefer-original-connection" => true,
             "skip-missing-groups" => false,
             "cache" => undefined
           }
         },
         "username-to-dn" => {
           "username-filter" => {
             "attribute" => "uid",
             "base-dn" => "ou=Users,dc=jboss,dc=org",
             "force" => true,
             "recursive" => false,
             "user-dn-attribute" => "dn",
             "cache" => undefined
           }
         }
       }
     },
```

```
    "plug-in" => undefined,
    "server-identity" => undefined
  }
}
```

5.4.2.2. キャッシュの有効化

注記

本セクションおよび後続のセクションで使用される管理 CLI コマンドは、セキュリティーレルムの authentication セクション (つまり **authentication=ldap**/) でキャッシュを設定します。承認セクションのキャッシュは、コマンドのパスを更新することで、同様の方法で設定することもできます。

キャッシュを有効にする管理 CLI コマンド

```
/core-service=management/security-realm=LDAPRealm/authentication=ldap/cache=by-access-time:add(eviction-time=300, cache-failures=true, max-cache-size=100)
```

このコマンドは、エビクション時間が 300 秒 (5 分) で最大キャッシュサイズが 100 の認証に、**by-access-time** キャッシュを追加します。さらに、失敗した検索はキャッシュされます。または、**by-search-time** キャッシュを設定することもできます。

```
/core-service=management/security-realm=LDAPRealm/authentication=ldap/cache=by-search-time:add(eviction-time=300, cache-failures=true, max-cache-size=100)
```

5.4.2.3. 既存キャッシュの検査

既存のキャッシュを調べる管理 CLI コマンド

```
/core-service=management/security-realm=LDAPRealm/authentication=ldap/cache=by-access-time:read-resource(include-runtime=true)

{
  "outcome" => "success",
  "result" => {
    "cache-failures" => true,
    "cache-size" => 1,
    "eviction-time" => 300,
    "max-cache-size" => 100
  }
}
```

include-runtime 属性は、キャッシュ内の現在のアイテム数を表示する **cache-size** を追加します。上記の出力では **1** です。

5.4.2.4. 既存のキャッシュの内容のテスト

既存のキャッシュの内容をテストする管理 CLI コマンド

```
/core-service=management/security-realm=LDAPRealm/authentication=ldap/cache=by-access-
```

```
time:contains(name=TestUserOne)

{
  "outcome" => "success",
  "result" => true
}
```

これは、**TestUserogg** のエントリーがキャッシュに存在することを示しています。

5.4.2.5. キャッシュのフラッシュ

キャッシュから単一の項目をフラッシュしたり、キャッシュ全体をフラッシュしたりできます。

単一アイテムを実行する管理 CLI コマンド

```
/core-service=management/security-realm=LDAPRealm/authentication=ldap/cache=by-access-time:flush-cache(name=TestUserOne)
```

キャッシュ全体のフラッシュのための管理 CLI コマンド

```
/core-service=management/security-realm=LDAPRealm/authentication=ldap/cache=by-access-time:flush-cache()
```

5.4.2.6. キャッシュの削除

キャッシュを削除する管理 CLI コマンド

```
/core-service=management/security-realm=LDAPRealm/authentication=ldap/cache=by-access-time:remove()

reload
```

第6章 セキュリティードメインがセキュリティーマッピングを使用するように設定する

セキュリティードメインにセキュリティーマッピングを追加すると、認証または承認の実行後、情報がアプリケーションに渡される前に認証および承認情報を組み合わせることができます。セキュリティーマッピングの詳細は、JBoss EAP『**セキュリティーアーキテクチャー**』の「セキュリティーマッピング」を参照してください。

既存のセキュリティードメインにセキュリティーマッピングを追加するには、**code**、**type**、および関連するモジュールオプションを設定する必要があります。**code** フィールドは、**SimpleRoles**、**PropertiesRoles**、**DatabaseRoles**、またはセキュリティーマッピングモジュールのクラス名などの短い名前です。**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指し、許可される値はプリンシパル、**principal**、**role**、**attribute**、または **credential** です。使用できるセキュリティーマッピングモジュールとそのモジュールオプションの完全リストは、JBoss EAP『**ログインモジュールのリファレンス**』の「セキュリティーマッピング」を参照してください。

例: SimpleRoles セキュリティーマッピングを既存のセキュリティードメインに追加する管理 CLI コマンド

```
/subsystem=security/security-domain=sampleapp/mapping=classic:add

/subsystem=security/security-domain=sampleapp/mapping=classic/mapping-
module=SimpleRoles:add(code=SimpleRoles,type=role,module-options=[("user1"=>"specialRole")])

reload
```

第7章 スタンドアロンサーバー VS管理対象ドメインに関する考慮事項

Microsoft Active Directory を含む LDAP サーバーでのアイデンティティー管理の設定は、スタンドアロンサーバーまたは管理対象ドメインで使用されるかどうかにかかわらず、基本的に同じです。一般的に、これはセキュリティーレルムとセキュリティードメインの両方でほとんどのアイデンティティーストアを設定するのにも適用されます。他の設定と同様に、スタンドアロン設定は **standalone.xml** ファイルにあり、管理対象ドメインの設定は **domain.xml** および **host.xml** ファイルにあります。

付録A リファレンス資料

A.1. 例: WILDFLY-CONFIG.XML

wildlfly-config.xml ファイルは、クライアントが Elytron Client を使用する方法のひとつです。これにより、クライアントは JBoss EAP に接続を行う際に、セキュリティー情報を使用できます。Elytron クライアントの使用に関する詳細は、「Configure Client Authentication with Elytron Client を参照してください。

例: custom-config.xml

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <authentication-rules>
      <rule use-configuration="monitor">
        <match-host name="127.0.0.1" />
      </rule>
      <rule use-configuration="administrator">
        <match-host name="localhost" />
      </rule>
    </authentication-rules>
    <authentication-configurations>
      <configuration name="monitor">
        <sasl-mechanism-selector selector="DIGEST-MD5" />
        <providers>
          <use-service-loader />
        </providers>
        <set-user-name name="monitor" />
        <credentials>
          <clear-password password="password1!" />
        </credentials>
        <set-mechanism-realm name="ManagementRealm" />
      </configuration>

      <configuration name="administrator">
        <sasl-mechanism-selector selector="DIGEST-MD5" />
        <providers>
          <use-service-loader />
        </providers>
        <set-user-name name="administrator" />
        <credentials>
          <clear-password password="password1!" />
        </credentials>
        <set-mechanism-realm name="ManagementRealm" />
      </configuration>
    </authentication-configurations>

    <net-authenticator/>

    <!-- This decides which SSL context configuration to use -->
    <ssl-context-rules>
      <rule use-ssl-context="mycorp-client">
        <match-host name="mycorp.com"/>
      </rule>
    </ssl-context-rules>
```

```
    <ssl-contexts>
      <default-ssl-context name="mycorp-context"/>
      <ssl-context name="mycorp-context">
        <key-store-ssl-certificate key-store-name="store1" alias="mycorp-client-certificate"/>
        <!-- This is an OpenSSL-style cipher suite selection string; this example is the expanded form of
DEFAULT to illustrate the format -->
        <cipher-suite selector="ALL:!EXPORT:!LOW:!aNULL:!eNULL:!SSLv2"/>
        <protocol names="TLSv1.2"/>
      </ssl-context>
    </ssl-contexts>
  </authentication-client>
</configuration>
```

wildfly-config.xml ファイルを使用したクライアントの設定方法に関する詳細は、JBoss EAP『**開発ガイド**』の「wildfly-config.xml ファイルを使用したクライアント設定」を参照してください。

A.2. シングルサインオン属性への参照

SSO 認証メカニズムの設定。

これは、**undertow** サブシステムの **application-security-domain** の **setting=single-sign-on** リソースの参照です。

A.2.1. シングルサインオン

表A.1 single-sign-on 属性

属性	説明
domain	使用されるクッキードメイン。
path	Cookie パス。
http-only	Cookie の **httpOnly** 属性の設定
secure	Cookie の **secure** 属性の設定
cookie-name	Cookie の名前。
key-store	プライベートキーエントリーが含まれるキーストアへの参照。
key-alias	バックチャネルログアウト接続の署名および検証に使用されるプライベートキーエントリーのエイリアス。

属性	説明
credential-reference	プライベートキーエントリーを復号化するための認証情報参照。 **credential-reference** には以下の属性があります。 ● **store**: 認証情報のエイリアスを保持するクレデンシャルストアの名前。 ● **alias**: ストアに保存されたシークレットまたは認証情報を示すエイリアス。 ● **type**: この参照がアノテーション付けている認証情報のタイプ。 ● **clear-text**: クリアテキストを使用して指定されるシークレットです。認証情報またはシークレットをサービスに提供するためのクレデンシャルストアの方法をチェックします。
client-ssl-context	バックチャネルログアウト接続のセキュア化に使用される SSL コンテキストへの参照。

A.3. パスワードマッパー

パスワードマッパーは、以下のアルゴリズムタイプのいずれかを使用して、データベースの複数のフィールドからパスワードを構成します。

- クリアテキスト
- シンプルダイジェスト
- ソルトシンプルダイジェスト
- bcrypt
- SCRAM
- モジュラークリプト (modular crypt)

パスワードマッパーには以下の属性があります。

注記

最初のコラムのインデックスは、すべてのマッパーに対して **1** になります。

表A.2 パスワードマッパー属性

Mapper 名	属性	暗号化の方法

Mapper 名	属性	暗号化の方法
clear-password-mapper	• **password-index** クリアテキストパスワードが含まれるコラムのインデックス。	暗号化なし
simple-digest	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **algorithm** 使用されるハッシュアルゴリズム。以下の値がサポートされています。 ◦ **simple-digest-md2** ◦ **simple-digest-md5** ◦ **simple-digest-sha-1** ◦ **simple-digest-sha-256** ◦ **simple-digest-sha-384** ◦ **simple-digest-sha-512** • **hash-encoding** 表現ハッシュを指定します。許可される値: ◦ **base64** (デフォルト) ◦ **hex**	簡単なハッシュメカニズムが使用されます。

Mapper 名	属性	暗号化の方法
salted-simple-digest	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **algorithm** 使用されるハッシュアルゴリズム。以下の値がサポートされています。 ◦ **password-salt-digest-md5** ◦ **password-salt-digest-sha-1** ◦ **password-salt-digest-sha-256** ◦ **password-salt-digest-sha-384** ◦ **password-salt-digest-sha-512** ◦ **salt-password-digest-md5** ◦ **salt-password-digest-sha-1** ◦ **salt-password-digest-sha-256** ◦ **salt-password-digest-sha-384** ◦ **salt-password-digest-sha-512** • **salt-index** ハッシュに使用される salt を含む列のインデックス。 • **hash-encoding** ハッシュの表示を指定します。許可される値: ◦ **base64** (デフォルト) ◦ **hex** • **salt-encoding** ソルトの表示を指定します。許可される値: ◦ **base64** (デフォルト) ◦ **hex**	シンプルなハッシュメカニズムがソルトに使用されます。

Mapper 名	属性	暗号化の方法
bcrypt-password-mapper	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **salt-index** ハッシュに使用される salt を含む列のインデックス。 • **iteration-count-index** 使用される反復の数を含む列のインデックス。 • **hash-encoding** ハッシュの表示を指定します。許可される値: ◦ **base64** (デフォルト) ◦ **hex** • **salt-encoding** ソルトの表示を指定します。許可される値: ◦ **base64** (デフォルト) ◦ **hex**	ハッシュに使用される Blowfish アルゴリズム。

Mapper 名	属性	暗号化の方法
scram-mapper	• **password-index** パスワードハッシュを含むコラムのインデックス。 • **algorithm** 使用されるハッシュアルゴリズム。以下の値がサポートされています。 ◦ **scram-sha-1** ◦ **scram-sha-256** ◦ **scram-sha-384** ◦ **scram-sha-512** • **salt-index** ソルトを含むコラムのインデックスは、ハッシュ処理に使用されます。 • **iteration-count-index** 使用される反復の数を含む列のインデックス。 • **hash-encoding** ハッシュの表示を指定します。許可される値: ◦ **base64** (デフォルト) ◦ **hex** • **salt-encoding** ソルトの表示を指定します。許可される値: ◦ **base64** (デフォルト) ◦ **hex**	Salted Challenge Response Authentication メカニズムがハッシュ化に使用されます。
modular-crypt-mapper	• **password-index** 暗号化したパスワードが含まれるコラムのインデックス。	Modular Crypt エンコーディングでは、パスワードタイプ、ハッシュまたはダイジェスト、ソルト (salt)、反復カウント (iteration count) などの複数の情報が単一の文字列でエンコードできるようになります。

Revised on 2021-10-10 12:51:52 +1000