



Red Hat JBoss Enterprise Application Platform 7.3

アイデンティティ管理の設定方法

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

Red Hat JBoss Enterprise Application Platform 7.3 アイデンティティ管理の設定方法

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

法律上の通知

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本ガイドでは、JBoss EAP 管理インターフェイスやセキュリティードメインで LDAP ディレクトリーなどのアイデンティティーストアを使用する方法について説明します。本ガイドは、JBoss EAP Security Architecture に説明されている概要にまで至ります。このため、管理者が LDAP に関する基本知識を持ち、JBoss EAP のセキュリティー概念を理解したら必ず確認してください。

目次

第1章 アイデンティティ管理の概要	3
第2章 ELYTRON サブシステム	4
2.1. ファイルシステムベースのアイデンティティストアでの認証設定	4
2.2. プロパティファイルベースのアイデンティティストアでの認証設定	5
2.3. データベースベースのアイデンティティストアでの認証設定	6
2.4. LDAP ベースのアイデンティティストアでの認証設定	7
2.5. 証明書による認証設定	9
2.6. 複数のアイデンティティストアを使用した認証および承認の設定	11
2.7. アプリケーションの認証設定の上書き	13
2.8. セキュリティーレルムのキャッシングの設定	14
2.9. コンテナ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手順	16
2.10. ベアラートークンを使用して認証と承認の設定	18
第3章 レガシーセキュリティーサブシステム	23
3.1. LDAP 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	23
3.2. データベース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	26
3.3. プロパティファイ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手順	27
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	28
3.5. セキュリティードメインのキャッシングの設定	31
第4章 アプリケーション設定	34
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 WEB アプリケーションを設定する手順	34
4.2. ELYTRON クライアントによるクライアント認証の設定	36
4.3. 信頼できるセキュリティードメインのアウトフロー設定	44
第5章 LDAP での管理インターフェイスのセキュリティー保護	46
5.1. ELYTRON の使用	46
5.2. レガシーのコア管理認証の使用	47
5.3. LDAP および RBAC	52
5.4. キャッシングの有効化	62
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手順	68
第7章 スタンドアロンサーバー VS.管理対象ドメインの考慮事項	69
付録A 参考資料	70
A.1. WILDFLY-CONFIG.XML の例	70
A.2. シングルサインオン属性の参照	71
A.3. パスワードマッパー	72

第1章 アイデンティティ管理の概要

さまざまなアイデンティティストアでアプリケーションのセキュリティを確保するための基本的なアイデンティティ管理の概要については、Red Hat JBoss Enterprise Application Platform (JBoss EAP) [セキュリティアーキテクチャー 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ストアを設定して、アプリケーションのセキュリティ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ストアを認証局として使用することもできます。プリンシパル関連のさまざまなロールおよびアクセス情報は、LDAP ディレクトリーに保存でき、JBoss EAP で直接使用することも、既存の JBoss EAP ロールにマッピングすることも可能です。



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を基盤とするアイデンティティ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が原因で、認証および承認の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ストアでの認証設定

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

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

6. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。
アプリケーションの **web.xml** および **jboss-web.xml** は、JBoss EAP で設定した **application-security-domain** を使用するように更新する必要があります。このサンプルは、[Configure Web Applications to use Elytron or Legacy Security for Authentication](#) で確認できます。

これで、お使いのアプリケーションでファイルシステムベースのアイデンティティストアを使用して認証を行えるようになりました。

2.2. プロパティファイルベースのアイデンティティストアでの認証設定

1. プロパティファイルを作成します。

ユーザーとパスワードのマッピング、ユーザーとロールのマッピングの2つのプロパティファイルを作成する必要があります。通常、このファイルは **jboss.server.config.dir** ディレクトリにあり、***-users.properties** および ***-roles.properties** の命名規則に従います。ただし、他の場所や名前を使用できます。***-users.properties** ファイルには、**properties-realm** への参照も含める必要があります。これについては次の手順で作成します (**#\$REALM_NAME=YOUR_PROPERTIES_REALM_NAME\$**)。

ユーザー/パスワードファイルの例: **example-users.properties**

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

ユーザー/ロールファイルの例: **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 **examplePropRealm** で、これは1つ前の手順の **example-users.properties** ファイルで使用しています。また、プロパティファイルが **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。

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

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

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

アプリケーションの **web.xml** および **jboss-web.xml** は、JBoss EAP で設定した **application-security-domain** を使用するように更新する必要があります。このサンプルは、[Configure Web Applications to use Elytron or Legacy Security for Authentication](#) で確認できます。

これで、お使いのアプリケーションで、プロパティファイルベースのアイデンティティストアを使用して認証が行えるようになりました。



重要

プロパティファイルは、サーバーの起動時にのみ読み込まれます。ユーザーを手作業か **add-user** スクリプトを使用して、サーバーの起動後に追加した場合には、サーバーのリロードが必要です。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ストアでの認証設定

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。
アイデンティティストアのデータベースを使用して認証を設定するには、このデータベースへのユーザー名、パスワード、およびロールの保存方法を決定する必要があります。この例では、以下のデータ例でテーブルを1つ使用します。

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をデプロイし、データソースを設定する必要があります。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方法を示しています。

```
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、1つの **principal-query** からパスワードおよびロールを取得する方法を示しています。ロールや、追加の認証または承認情報の取得に複数のクエリーが必要な場合には、**attribute-mapping** 属性を使用して追加の **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

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

6. アプリケーションの **web.xml** および **jboss-web.xml** を設定します。
アプリケーションの **web.xml** および **jboss-web.xml** は、JBoss EAP で設定した **application-security-domain** を使用するように更新する必要があります。このサンプルは、[Configure Web Applications to use Elytron or Legacy Security for Authentication](#) で確認できます。

2.4. LDAP ベースのアイデンティティーストアでの認証設定

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に 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、およびロールの保存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ています。

```
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

JMX **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

参照元の LDAP サーバーに参照のループが含まれる場合には、JBoss EAP サーバーで **java.lang.OutOfMemoryError** エラーが発生する可能性があります。

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

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

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

アプリケーションの **web.xml** および **jboss-web.xml** は、JBoss EAP で設定した **application-security-domain** を使用するように更新する必要があります。このサンプルは、[Configure Web Applications to use Elytron or Legacy Security for Authentication](#) で確認できます。



重要

elytron サブシステムで LDAP サーバーを使用して認証を実行しており、この LDAP サーバーに到達できない場合に、JBoss EAP は **500** または内部サーバーエラーを返します。この動作は、同じ状況下でレガシーの **security** サブシステムを使用する以前の JBoss EAP バージョンが **401** または不正アクセスのエラーコードを返す動作とは異なります。

2.5. 証明書による認証設定



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)
```

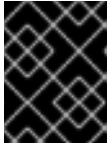
このレルムは、クライアントの証明書が含まれるトラストストアで設定する必要があります。認証プロセスでは、双方向 SSL のハンドシェイク時にクライアントが提示するものと同じ証明書を使用します。

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** 値に基づい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** の場合には、**CNDecoder** はプリンシパルを **クライアント** 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、**エイリアス** 値として、**ksRealm** に設定されたトラストストアの証明書の検索に使用します。



重要

デコードされたプリンシパルは、クライアントの証明書のトラストストアに設定した **エイリアス** 値に 指定しなければなりません。

- オプションで、サブジェクト別名エクステンションを使用する Evidence Decoder が、サブジェクト別名をプリンシパルとして使用するように設定できます。詳細は、[サーバーセキュリティ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. ロールの割り当て用に **constant-role-mapper** を追加します。
この例では、**constant-role-mapper** を使用して **ksRealm** のプリンシパルにロールを割り当てますが、他のアプローチ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

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

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。

アプリケーションの **web.xml** および **jboss-web.xml** は、JBoss EAP で設定した **application-security-domain** を使用するように更新する必要があります。このサンプルは、[Configure Web Applications to use Elytron or Legacy Security for Authentication](#) で確認できます。

さらに、**web.xml** が **CLIENT-CERT** を認証方法として使用するように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にアイデンティティの属性を保存する場合は、**aggregate-realm** を使用してアイデンティティ属性を1つのセキュリティーレルムにロードし、認証および承認を行います。

2.6.1. Elytron での集約レルム

aggregate-realm では、認証に1つのセキュリティーレルムを、Elytron の承認に別のセキュリティーレルム、または複数のセキュリティーレルムの集約を使用できます。たとえば、認証にプロパティレルムを、承認に JDBC レルムを使用するように、集約レルムを設定できます。

複数の承認レルムを集約するように設定された集約レルムでは、アイデンティティは以下のように作成されます。

- 承認用に設定された各セキュリティーレルムの属性値を読み込む。
- 複数の承認レルムで属性が定義されている場合は、最初に表示される属性の値を使用する。

以下の例では、複数の承認レルムに同じアイデンティティ属性の定義が含まれる場合のアイデンティティの作成方法を説明します。

例

集約レルムの設定:

```
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
```

集約レルムから取得した生成アイデンティティ:


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

この例では、**e-mail** 属性は両方の承認レルムで定義されます。JDBC レルムで定義された値は、集約レルムは、承認レルムを集約するように設定されているため (**authorization-realms=[jdbc-realm,ldap-realm]**)、生成された集約レルムの **e-mail** 属性に使用されます。

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および承認を設定するには、集約レルムを作成し、セキュリティドメインとアプリケーションセキュリティドメインが集約レルムを使用するように設定します。

前提条件

- 集約するセキュリティレルムを設定する。
セキュリティレルムの設定に関する詳細は、[アイデンティティ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。
- セキュリティドメインで使用するロールデコーダーを設定する。
ロールデコーダーの詳細は、[サーバーセキュリティの設定方法の 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。

- 1つ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-realms=
[__SECURITY_REALM_FOR_AUTHORIZATION_1__, __SECURITY_REALM_FOR_AU-
THORIZATION_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=exampleAggregateRealmApplicationDomain: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。

アプリケーションの **web.xml** および **jboss-web.xml** は、JBoss EAP で設定した **application-security-domain** を使用するように更新する必要があります。このサンプルは、[Configure Web Applications to use Elytron or Legacy Security for Authentication](#) で確認できます。

2.6.3. 集約レルムの例

承認レルムと集約レルムの例

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

以下のレルムを事前設定する必要があります。

- `examplePropertiesRealm` 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=examplePropertiesRealm,authorization-realm=exampleJdbcRealm)
```

承認レルム 2 つと集約レルムの例

この例では、認証に **properties-realm** を、承認に **ldap-realm** と **jdbc-realm** の集約を使用します。

以下のレルムを事前設定する必要があります。

- `examplePropertiesRealm` 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=examplePropertiesRealm,authorization-realms=[exampleJdbcRealm,exampleLdapRealm])
```

2.7. アプリケーションの認証設定の上書き

JBoss EAP で設定されたアプリケーションの認証設定を上書きできます。これには、**undertow** サブシステムの **application-security-domain** セクションにある **override-deployment-configuration** プロパティを使用します。

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



注記

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

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

jboss-web.xml の例

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

override-deployment-configuration を有効化すると、**BASIC** または **DIGEST** など、異なる認証メカニズムを指定する **http-authentication-factory** を新たに作成できます。

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"
  }
}
```

この設定では、アプリケーションの **jboss-web.xml** で定義された認証メカニズムを上書きし、**FORM** の代わりに **BASIC** を使用して認証を試行します。

2.8. セキュリティーレルムのキャッシングの設定

Elytron には **caching-realm** があり、セキュリティーレルムからの認証情報ルックアップの結果をキャッシュできます。たとえば、これを使用して LDAP またはデータベース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高めることができます。

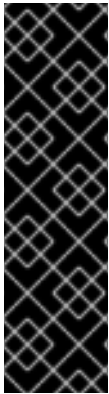
caching-realm は、LRU または **Least Recently Used** キャッシュストラテジーを使用して **PasswordCredential** 認証情報をキャッシュし、エントリーが最大数に達すると、アクセスが最も少ないエントリーが削除されます。

以下のセキュリティーレルムで **caching-realm** を使用できます。

- **filesystem-realm**
- **jdbc-realm**
- **ldap-realm**
- カスタムの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と、基盤のセキュリティーレルムでリスン機能がサポートされる場合に、この変更内容は JBoss EAP キャッシュレルムだけに伝播されます。特に、**ldap-realm** はリスンをサポートしますが、**ldap-realm** 内では **ロール** などのフィルターされた属性はサポートされません。

キャッシュレームにユーザーデータのキャッシュを正しく設定できるようにするには、認証情報ソースではなく、キャッシングレームを使用してユーザー属性を変更することを推奨します。または、キャッシュを [クリア](#) してください。



重要

キャッシュレームでのユーザーの変更は、テクノロジープレビュー機能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、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲は、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ームを作成します。
 caching-realm と合わせて使用する既存のセキュリティーレームが必要です。たとえば、 [ファイルシステムベースのアイデンティティストアを使用した認証の設定](#) の手順と似た **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。
キャッシュ先となる既存のレームがある場合には、そのレームを参照する **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 を作成すると、他のセキュリティーレームと同様にセキュリティー設定で使えるようになります。たとえば、 [ファイルシステムベースのアイデンティティストアでの認証設定](#) 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}]]])
```

cacheing-realm の **maximum-entries** と **maximum-age** 属性を使用すると、キャッシュサイズと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。

cacheing-realm キャッシュの消去

clear-cache コマンドを使用して既存のキャッシュをクリアできます。キャッシュをクリアすると、セキュリティレルムの最新データを使用して、強制的に再生成します。

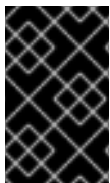
```
/subsystem=elytron/cacheing-realm=exampleCacheRealm:clear-cache
```

2.9. コンテナ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手順

Elytron **FORM** の認証メソッドを使用すると、アプリケーションにコンテナ管理のシングルサインオンを使用するように JBoss EAP を設定できます。この設定では、ユーザーを1回認証し、**FORM** 認証メソッドでセキュリティ保護された他のリソースに再認証する必要なくアクセスできます。

以下の場合に、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が残っていない。
- アプリケーションからログアウトする。



重要

さまざまな JBoss EAP インスタンスにデプロイされたアプリケーションには、このようなインスタンスが1つのクラスター内にある場合に限り、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 に参加するさまざまな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の作成時または破棄時 (ログイン時、またはログアウト時) に発生するイベントのメッセージ交換に使用します。

elytron サブシステムで **key-store** を作成するには、まず以下のように 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 ファイルを作成したら、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. セキュリティーレームを追加します。

以下の管理 CLI コマンドを使用して **FileSystem** レーム (ローカルファイルシステムにユーザーを保存するアイデンティティストア) 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、通常ユーザーにログインページを表示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。

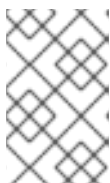
4. **undertow** サブシステムでアプリケーションのセキュリティードメインを設定します。



注記

undertow サブシステムに **application-security-domain** がすでに定義されており、このドメインを使用して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、お使いのアプリケーションの **jboss-web.xml** ファイルで特定のセキュリティードメインが定義されていない場合には、アプリケーションサーバーは **other** という名前のセキュリティードメインを選択します。

5. **undertow** サブシステムがシングルサインオンを有効にし、キーストアを使用するように更新します。
シングルサインオンは、**undertow** サブシステムの特定の **application-security-domain** 定義に対して有効になります。アプリケーションのデプロイに使用するサーバーで、同じ設定を使用することが重要です。

シングルサインオンを有効にするには、以下のように **undertow** サブシステムで既存の **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

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

SSO 属性とその定義に関する詳細は、[シングルサインオン属性のリファレンス](#) 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。
アプリケーションの **web.xml** および **jboss-web.xml** は、JBoss EAP で設定した **application-security-domain** を使用するように更新する必要があります。このサンプルは、[Configure Web Applications to use Elytron or Legacy Security for Authentication](#) で確認できます。

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. ベアラートークンを使用して認証と承認の設定

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る 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 要求をアプリケーションに送信した後、**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存在す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を使用するか、OAuth2 準拠の承認サーバーによって発行された不透明なトークンを使用することで認証をサポートします。関連情報のセクションを参照してください。

次の例は、**Authorization** HTTP ヘッダーに **mF_9.B5f-4.1JqM** ベアラートークンが含まれてい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
```

BEARER_TOKEN メカニズムは、ベアラトークン文字列を抽出し、検証のために **token-realm** 実装に渡すことができるようになりました。実装がベアラートークンの検証に成功すると、Elytron はトークンによって表される情報に基づいてセキュリティコンテキストを作成します。アプリケーションは、このセキュリティコンテキストを使用して、リクエスターに関する情報を取得できます。次に、リクエスターに HTTP リソースへのアクセスを提供することにより、リクエストを実行するかどうかを決定できます。

リクエスターがベアラートークンを提供しない場合、**BEARER_TOKEN** メカニズムは **401 HTTP** ステータスコードを返します。以下に例を示します。

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

リクエスターがリソースへのアクセスを許可されていない場合、**BEARER_TOKEN** メカニズムは **403 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 検証の詳細は、[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](#) を参照してください。
- **token-realm** で使用できる属性の詳細は、[サーバーのセキュリティーの設定方法 ガイドの 表 A.93.token-realm 属性](#) を参照してください。

2.10.2. JSON Web Token (JWT) 認証の設定

elytron サブシステムで **token-realm** を指定することで、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 は次のチェックを完了します。

- **exp** クレームおよび **nbf** クレームで指定された値の自動有効期限チェック。
- オプション: 次のいずれかの方法で提供される公開鍵に基づく署名チェック:
 - **public-key** または **certificate** 属性を使用
 - 名前付き公開鍵での鍵マップの使用
 - **client-ssl-context** 属性を使用して、**jku** クレームで指定された URL からリモート JSON Web キー (JWK) セットを取得します。
- オプション: JWT をチェックして、**iss** および **aud** クレームでサポートされている値のみが含まれていることを確認します。**issuer** 属性と **audience** 属性を使用して、これらのチェックを実行できます。

次の例は、セキュリティーレルムとして **token-realm** を示し、属性として **principal-claim** を示しています。**principal-claim** 属性は、**elytron** がプリンシパルの名前を取得するために使用するクレーム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、署名の検証にさまざまなキーペアを使用して、これらのキーペアを簡単にローテーションできます。Elytron はトークンから **kid** の主張を取得し、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、**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 に存在せず、**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 用に設定されたキーマップの例。


```
<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 サブシステムに **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** の属性と **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、[サーバーの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 準拠の認証サーバーによって発行された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トークン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>})
```

次の例は、**token-realm** 要素で指定された **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ています。**oauth2-introspection** エンドポイントは、**client-id** 属性と **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の詳細は、[表 A.95 token-realm oauth2-introspection Attributes](#) を参照してください。

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

Open ID Connect ID トークンなどの JWT 形式の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た不透明なトークンを使用して、アプリケーションの認証を設定できます。

前提条件

- 必要な認証方法に応じて、次のいずれかの手順を実行して **token-realm** を作成します。
 - [JSON Web Token \(JWT\)認証の設定](#)。
 - [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](#)。

手順

1. **elytron** サブシステムに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でトークンセキュリティードメインを指定していることを確認してください。

elytron サブシステムに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** で **BEARER_TOKEN** 認証方式が指定され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** を使用していることを確認します。

関連情報

- **BEARER_TOKEN** 認証メカニズムの詳細については、[ベアラートークン認証](#) を参照してください。
- **token-realm jwt** 属性の詳細は、[サーバーのセキュリティの設定方法](#) 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える属性の詳細は、[表 A.95 token-realm oauth2-introspection Attributes](#) を参照してください。
- アプリケーションの **web.xml** および **jboss-web.xml** 設定の詳細については、[アイデンティティ管理の設定方法 ガイド](#) の [認証に Elytron またはレガシーセキュリティを使用するよう Web アプリケーションを設定する手順](#) を参照してください。

第3章 レガシーセキュリティサブシステム

3.1. LDAP を使用するようにセキュリティドメインを設定する手順

セキュリティドメインは、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に設定できます。セキュリティ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、他にも使用可能な LDAP ログインモジュールが複数あります。具体的には Ldap、AdvancedLdap および AdvancedAdLdap ログインモジュールで、セキュリティドメインが LDAP を使用するように設定できます。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ドメインを作成する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

レガシーのセキュリティサブシステムで LDAP サーバーを使用して認証を実行しており、この LDAP サーバーに到達できない場合に、JBoss EAP は **500** または内部サーバーエラーを返します。この動作は、同じ状況下で JBoss EAP の以前のバージョンが **401** または不正アクセスのエラーコードを返す動作とは異なります。

3.1.1. LdapExtended ログインモジュール

LdapExtended (`org.jboss.security.auth.spi.LdapExtLoginModule`) は、検索を使用して LDAP サーバー上のバインドユーザーと関連のロールを特定するログインモジュール実装です。ロールは再帰的にクエリーを実行して DN に従い、階層的なロール構造を移動します。特に Active Directory ではない LDAP 実装の場合など、セキュリティドメインで LDAP を使用する場合はほぼ、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なります。

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、ユーザーとロールの **baseCtxDN** ツリーと **rolesCtxDN** ツリーの両方を検索する機能を備えた LDAP ユーザーです。認証するユーザー DN には、**baseFilter** 属性で指定されたフィルターを使用してクエリーが実行されます。
2. ユーザー DN を **InitialLdapContext** 環境の **Context.SECURITY_PRINCIPAL** として使用し、LDAP サーバーにバインドし、作成されたユーザー DN が認証されます。**Context.SECURITY_CREDENTIALS** プロパティは、コールバックハンドラーが取得した **String** パスワードに設定されます。

3.1.1.1. LdapExtended ログインモジュールを使用するようにセキュリティドメインを設定する手順

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 を使用する場合は詳細は [管理 CLI ガイド](#) を参照してください。

3.1.1.1.1. Active Directory の LdapExtended 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手順

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 設定です。一部の Active Directory 設定では、通常のポート **389** ではなく、**3268** のグローバルカタログに対する検索が必要になる場合があります。これは、Active Directory フォレストに複数のドメインが含まれる場合に必要になる可能性が高いです。

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_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"=>"JBOSSearchuser"),
("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 に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 の例の主な相違点は、ロールの検索から、ユーザーの DN を使用した member 属性の検索に変更になった点です。次に、ログインモジュールはロールの DN を使用して、グループが所属す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"=>"JBOSSearchuser"), ("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ようにセキュリティードメインを設定する手順

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、認証とロールマッピングをサポートする JDBC (Java Database Connectivity) ベースの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。以下に例を示します。

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

Principals テーブルは、**PrincipalID** ユーザーと有効なパスワードを、**Roles** テーブルは **PrincipalID** ユーザーとそのロールセットを関連付けます。ユーザーのパーミッションに使用するロールは、**Roles** の **roleGroup** コラムの値が含まれる行に追加する必要があります。

ユーザーがログインモジュールが使用する SQL クエリーを指定できる点で、これらのテーブルは論理的であるといえます。唯一の要件は、**java.sql.ResultSet** に、先程説明した **Principals** と **Roles** テーブルと同じ論理構造を使用する必要があります。テーブルとコラムの実際の名前は、コラムインデックスに基づいて結果にアクセスするため、重要ではありません。

この概念を明確化するには、すでに宣言済みの **Principals** と **Roles** の 2 つのテーブルが含まれるデータベースを思い浮かべてください。以下のステートメントでは、テーブルに以下のデータが投入されます。

- **Principals** テーブルには **PrincipalID** が **java** でパスワードが **echoman** のデータを投入
- **Roles** テーブルの **RolesRoleGroup** には **PrincipalID** が **java** でロールが **Echo** のデータを投入
- **Roles** テーブルの **CallerPrincipalRoleGroup** には **PrincipalID** が **java** でロールが **caller-java** のデータを投入

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. 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手順

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後に、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できます。以下の例は、**MyDatabaseDS** という名前のデータソースが作成済みで、以下で構築されるデータベースを使用して適切に設定されていることが前提です。

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

Database 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. プロパティファイ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手順

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可能です。

3.3.1. UsersRoles ログインモジュール

UsersRoles は、Java プロパティ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 マッピングのファイル名は **users.properties** で、デフォルトの username-to-roles マッピングのファイル名は **roles.properties** です。



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をサポートします。

プロパティファイルは、初期化メソッドスレッドのコンテキストクラスローダーを使用して、初期化中に読み込みます。つまり、プロパティファイルを Jakarta EE デプロイメントのクラスパス (例: WAR アーカイブの **WEB-INF/classes** フォルダー) またはサーバークラスパス上の任意のディレクトリに配置できます。

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 EAPLogin 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ドメインを設定する手順

次の例では、以下のファイルが作成されており、アプリケーションのクラスパスで利用可能であることを前提としています。

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

UsersRoles 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ドメインを設定する手順

JBoss EAP には、セキュリティ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 EJB のセキュリティ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を確保した Web アプリケーションまたは EJB 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、認証に使用する信頼できるクライアント証明書を含めるか、クライアントの証明書署名に使用する認証局の証明書を含める必要があります。このログインモジュールを使用して、設定済みのトラストストアでクライアントが提示する証明書を認証します。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ピングする方法を提供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ピングしませんが、別のログインモジュールと組み合わせることでこのマッピングが可能です。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紹介しています。



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 EJB 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、ロールを認証済みのプリンシパルにマップします。これには、**rolesProperties** 属性を使用してプロパティ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、ユーザー名とロールを一覧表示します。

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

ユーザー名は指定の証明書の DN として提示されるため、プロパティファイルを使用する場合は、`=` などの特殊文字をエスケープする必要があります (例: **CN=valid-client, OU=JBoss, O=Red Hat, L=Raleigh, ST=NC, C=US**)。

ロールプロパティファイルの例

```
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ドメインを使用するようにアプリケーションを設定する手順

他の形式の認証でセキュリティドメインを使用するようにアプリケーションを設定する場合と同様に、**jboss-web.xml** と **web.xml** ファイルの両方を適切に設定する必要があります。

jboss-web.xml の場合は、証明書ベース認証用に設定したセキュリティドメインへの参照を追加します。

jboss-web.xml の例

```
<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リティ保護したアプリケーションに対して認証するには、クライアントから 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にアクセスする必要があります。たとえば、ブラウザを使用してアプリケーションにアクセスする場合には、クライアントは信頼される証明書をブラウザのトラストストアにインポートする必要があります。

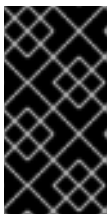
3.5. セキュリティードメインのキャッシングの設定

セキュリティードメインのキャッシュを指定して、認証チェックにかかる時間を短縮できます。デフォルトでは、セキュリティードメインは簡単なマップをキャッシュとして使用します。このデフォルトのキャッシュは、最大エントリー数が 1000 件の 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することができます。

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

前提条件

- セキュリティードメインを Infinispan キャッシュを使用するように設定する場合は、まず、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併用するには Infinispan キャッシュ設定は 1 つだけ定義できます。Infinispan キャッシュを使用するセキュリティードメインは複数設定できますが、セキュリティードメインごとに 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が作成されます。

[キャッシュコンテナの作成](#) 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場合は、以下を実行します。

1. **Configuration → Subsystems → Security (Legacy)** の順に移動します。
 2. 一覧からセキュリティドメインを選択し、**View** をクリックします。
 3. **Edit** をクリックし、**Cache Type** フィールドでは **default** または **infinispan** 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、**other** セキュリティドメインが Infinispan キャッシュを使用するように設定するには、次のコマンドを実行します。

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

3.5.2. プリンシパルの表示とフラッシュ

キャッシュのプリンシパルの一覧表示

以下の管理 CLI コマンドを使用して、セキュリティ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

必要に応じて、セキュリティ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ドメインのキャッシュを無効にできます。

- 管理コンソールを使用する場合は、以下を実行します。
 1. **Configuration → Subsystems → Security (Legacy)** の順に移動します。
 2. 一覧からセキュリティドメインを選択し、**View** をクリックします。
 3. **Edit** をクリックし、**Cache Type** には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

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

認証用に **elytron** またはレガシーの **security** サブシステムを設定した後に、お使いのアプリケーションでこのサブシステムを使用するように設定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に、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には、作成した **http-authentication-factory** 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には、ログインモジュールと設定する認証のタイプにより設定内容は異なります。

BASIC 認証を使用した web.xml の例

```
<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。

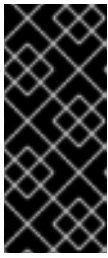
認証に使用するセキュリティドメインを指定するように、アプリケーションの **jboss-web.xml** を設定できます。**elytron** サブシステムを使用する場合には、これは **application-security-domain** の作成時に定義されます。レガシー **security** サブシステムを使用する場合には、これはレガシーセキュリティドメインの名前になります。

jboss-web.xml の例

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

jboss-web.xml を使用すると、セキュリティドメインを設定できるアプリケーションは1つだけです。または、**undertow** サブシステムを使用して、全アプリケーションのデフォルトのセキュリティドメインを指定できます。このように指定すると、**jboss-web.xml** を使用して個別のアプリケーションのセキュリティドメインを設定せずに済みます。

```
/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、全 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

EJB のセキュリティドメインは、EJB 設定 (**ejb3** サブシステムまたは **jboss-ejb3.xml** ファイルの EJB 記述子) または **@SecurityDomain** アノテーションを使用して定義します。

詳細は、*Developing EJB Applications* ガイドの [EJB Application Security](#) を参照してください。

サイレント BASIC 認証

elytron がサイレント **BASIC** 認証を実行するように設定できます。サイレント認証を有効にすると、Web アプリケーションへのアクセス時にログインのプロンプトが表示されません。代わりに別の認証メカニズムが使用されます。ユーザーの要求に Authorization ヘッダーが含まれる場合には、**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>
```

Elytron およびレガシーセキュリティサブシステムの併用

elytron サブシステムとレガシー **security** サブシステムの両方で認証を定義し、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には、JBoss EAP は最初に **elytron** サブシステムの設定済みのセキュリティドメインとの照合を開始します。一致するものが見つからない場合には、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ドメインとの照合を試行します。**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ドメインがある場合には、**elytron** セキュリティドメインが使用されます。

注記

1つのセキュリティドメインを使用して Web サブレットが定義されており、EJB 固有のセキュリティドメインを使用する別の EAR モジュールから EJB 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 セキュリティドメインに WAR と EJB をマッピングする場合に、アイデンティティ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 EJB に到達すると、アイデンティティは匿名になります。認証にセキュリティアイデンティティを設定する方法は、[信頼されているセキュリティ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 EJB が異なるセキュリティドメイン名を参照するにも拘らず、同じ Elytron セキュリティドメインにマップされた場合には、アイデンティティ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。EJB と WAR を別々に移行して、**elytron** サブシステムとレガシー **security** サブシステム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

EJB 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。

authentication-configuration

認証設定には、ユーザー名、パスワード、承認済みの 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レームが含まれます。認証設定で指定した接続情報は、初期コンテキストの **PROVIDER_URL** で指定した値よりも優先されます。

MatchRule

使用する認証設定の決定に使用されるルール。

authentication-context

接続の確立にクライアントで使用するルールおよび認証設定のセット。

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、各アウトバウンド接続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** への接続時に1つの認証設定を使用し、**server2** との接続時に別の認証設定を使用するルールを指定できます。認証コンテキストには、接続の確立時の選択方法を定義するルールや認証設定が含まれます。認証コンテキストは **ssl-context** も参照でき、ルールとの照合も可能です。

接続の確立時にセキュリティ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 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。

接続の確立時に、Elytron クライアントは認証コンテキストが提供するルールセットを使用して、認証時に使用する適切な認証設定を特定します。

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情報を使用できます。



重要

Elytron クライアントを使用して EJB 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めて XML ファイルを作成する必要があります。

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>
```

```

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

```

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

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



重要

[プログラムによるアプローチ](#)を使用する場合には、**wildfly.config.url** システムプロパティが設定されていても、指定した設定ファイルを上書きします。

ルールの作成時に **hostname**、**port**、**protocol** または **user-name** などのさまざまなパラメーターと合致するものを検索できます。**MatchRule** のオプションの完全リストは、Java ドキュメントにあります。ルールは、設定順に評価されます。

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。複数の一致設定をルールに追加する場合は、そのルールすべてが合致しない限り、認証設定は選択されません。

表4.1 共通ルール

属性	説明
match-local-security-domain	照合するローカルのセキュリティドメインを指定する name 属性を1つ設定できます。
match-host	照合するホスト名を指定する name 属性を1つ設定できます。たとえば、ホスト 127.0.0.1 は http://127.0.0.1:9990/my/path と一致します。
match-no-user	ユーザーのない URI に対して照合します。
match-path	照合するパスを指定する name 属性を1つ設定できます。たとえば、パス /my/path/ は http://127.0.0.1:9990/my/path と一致します。
match-port	照合するポートを指定する name 属性を1つ設定できます。たとえば、ポート 9990 は http://127.0.0.1:9990/my/path と一致します。
match-protocol	照合するプロトコルを指定する name 属性を1つ設定できます。たとえば、プロトコル http は http://127.0.0.1:9990/my/path と一致します。
match-urn	照合する URN を指定する name 属性を1つ設定できます。
match-user	照合する user を指定する user 属性を1つ設定できます。

wildfly-config.xml ファイルの例は、[Example wildfly-config.xml](#) を参照してください。**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内の全 Elytron クライアント設定を行います。

```
//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、各メソッド呼び出しはそのオブジェクトの新規インスタンスを返します。たとえば、別のホスト名で接続する場合に別個の設定が必要な場合は、以下を行うことができます。

```
//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 共通ルール

ルール	説明
matchLocalSecurityDomain(String name)	これは、設定ファイルのアプローチの match-domain と同じです。
matchNoUser()	これは、設定ファイルのアプローチの match-no-user と同じです。
matchPath(String pathSpec)	これは、設定ファイルのアプローチの match-path と同じです。
matchPort(int port)	これは、設定ファイルのアプローチの match-port と同じです。
matchProtocol(String protoName)	これは、設定ファイルのアプローチの match-port と同じです。
matchPurpose(String purpose)	このルールと同じ新しいルールを作成しますが、指定の目的名も合わせて照合します。
matchUrnName(String name)	これは、設定ファイルのアプローチの match-urn と同じです。
matchUser(String userSpec)	これは、設定ファイルのアプローチの match-userinfo と同じです。

また、空白の認証設定で開始せずに、**captureCurrent()** を使用して現在設定済みの認証設定で開始できます。

```

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

```

captureCurrent() を使用すると、以前に設定した認証コンテキストを取得し、そのコンテキストを新しい基本設定として使用します。認証コンテキストは、**run()** を呼び出してコンテキストをアクティブにすると確立されます。**captureCurrent()** を呼び出し、アクティブなコンテキストがない場合に、デフォルトの認証があれば、それを使用します。詳細は、以下のセクションを参照してください。

- [設定ファイルのアプローチ](#)

- デフォルト設定アプローチ
- 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、設定を構築するためのベースとしてだけ使用し、単独で使用するべきではありません。これは、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せんが、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. デフォルト設定アプローチ

デフォルト設定アプローチは、Elytron クライアントが指定す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();
```

Elytron クライアントはファイルシステムの **wildfly-config.xml** ファイルを自動検出して、デフォルト設定を使用します。以下の場所を探します。

- クライアントコード外に設定された **wildfly.config.url** システムプロパティーが指定する場所。
- クラスパスの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-client>
</configuration>
```

```

<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 メカニズムでは、匿名ユーザー以外のユーザーとしての認証はサポートしていません。つまり、**set-authorization-name** は、Elytron クライアントの設定ファイルの **set-anonymous** と合わせて使用できません。代わりに、**set-authorization-name** を設定する場合には、承認されたアイデンティティに **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** は解析され、JBoss EAP 設定の **default-authentication-context** 設定をもとに自動的に作成されます。**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場合や、**wildfly.config.url** システムプロパティを使用して設定している場合には、**AuthenticationContext** は自動的に解析され、このファイルをもとに作成されます。

例: デフォルトの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 **AuthenticationContext** を返すので、[programmatic approach](#) を使用してクライアントのコードで使用できます。

さらに、クライアントは **elytron** サブシステムが指定するクライアント設定をもとに **AuthenticationContext** を自動的に解析し、作成します。**elytron** サブシステムのクライアント設定は、認証ストアなど、**elytron** サブシステムで定義されている他のコンポーネントも活用できます。デプロイメントと **elytron** サブシステムの両方でクライアントの設定が指定されている場合には、**elytron** サブシステムの設定が使用されます。

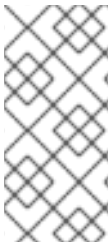
注記

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 JMX クライアントの設定

JBoss EAP 7.1 以降で、JConsole などの JMX 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。JMX クライアントの起動時に **-Dwildfly.config.url** システムプロパティを使用して、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の伝播



警告

Java 8 では認証情報で既知の制限があるため、JBoss EAP での **ElytronAuthenticator** の使用はサポートされておらず、推奨していません。このクラスを使用してアイデンティティを伝播する場合は、以下の制限に注意してください。

- Java 8 設計の制限により、保護されたサブレットへの呼び出しではセキュリティアイデンティティは伝播されません。
- EJB 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 では、現在のセキュリティコンテキストを使用して認証を実行する **ElytronAuthenticator** クラスが導入されました。[org.wildfly.security.auth.util.ElytronAuthenticator](#) クラスは [java.net.Authenticator](#) の実装です。

- このクラスには、新規インスタンスを構築するコンストラクター (**ElytronAuthenticator()**) が1つ含まれています。
- また、**PasswordAuthentication** を返すメソッド (**getPasswordAuthentication()**) が1つ含まれています。

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

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. 信頼できるセキュリティードメインのアウトフロー設定

セキュリティーが呼び出しされると、対象のセキュリティードメインに対してセキュリティーアイデンティティーが確立されます。呼び出しの処理時に、**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で、後続の **getCurrentSecurityIdentity()** が呼び出されると、関連する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は、1回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場合があります。**SecurityDomain** インスタンスはそれぞれ、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()** メソッドを呼び出すと、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、リクエスト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。デプロイメントごとに、1つのセキュリティードメインに関連付けられます。呼び出された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には、現在のセキュリティーアイデンティティーにセキュリティードメイン1つという概念は残ります。ただし、各デプロイメントは、そのデプロイメントが使用する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することが可能です。

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、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には、主に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

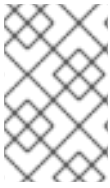
セキュリティードメインは、常に独自のセキュリティーアイデンティティーをインポートできます。独自のセキュリティーアイデンティティーをインポートする場合には、セキュリティードメインは常に自身を信頼します。

一般的なセキュリティーレルム

インポートプロセス中に、セキュリティードメインは、インポートされた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、設定済みのプリンシパルトランスフォーマーとレルムマッパーに渡して、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ピングします。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と同じセキュリティーレルムが使用される場合には、いずれも、基盤となる同じアイデンティティーがサポートするので、インポートが可能です。

信頼できるセキュリティードメイン

アイデンティティが正常にマップされたものの、共通のセキュリティーレلمがない場合には、インポートを処理するセキュリティードメインがテストされ、インポート元のセキュリティードメインが信頼できるかどうかを確認します。信頼できる場合には、インポートが可能です。



注記

このアイデンティティ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。このセキュリティーアイデンティティは、存在する間は信頼できません。

アウトフロー

セキュリティードメインは、セキュリティーアイデンティティを異なるセキュリティードメインに自動的に送信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 EJB 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イスのセキュリティ保護

管理インターフェイスは LDAP サーバー (Microsoft Active Directory など) に対して認証できます。これは、LDAP 認証システムを使用して実行できます。LDAP 認証システムは、最初にリモートディレクトリサーバーへの接続を確立します (アウトバウンド LDAP 接続を使用)。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、ユーザーが入力したパスワードとユーザーの DN を認証情報として使用し、新しい接続が確立されます。LDAP サーバーへの 2 つ目の接続と認証に成功すると、DN が有効であることが検証され、認証に成功します。



注記

LDAP で管理インターフェイスのセキュリティを保護すると、認証がダイジェストから BASIC/Plain に変更されますが、デフォルトでは、ユーザー名とパスワードが暗号化されずにネットワーク上で送信されてしまいます。送信接続で SSL/TLS を有効にして、このトラフィックを暗号化し、ユーザー名とパスワードが暗号化なしで送信されないようにします。



重要

LDAP を使用した管理インターフェイスのセキュリティ保護など、レガシーセキュリティレلمが LDAP サーバーを使用して認証を実行する場合には、JBoss EAP は LDAP サーバーに到達できないと **500** や内部サーバーのエラー、エラーコードを返します。この動作は、同じ状況下で JBoss EAP の以前のバージョンが **401** または不正アクセスのエラーコードを返す動作とは異なります。

5.1. ELYTRON の使用

管理インターフェイスは、任意のアイデンティティストアを使用するのと同じ方法で、**elytron** サブシステムで LDAP を使用してセキュリティ保護できます。**elytron** サブシステムとのセキュリティにアイデンティティストアを使用する方法は、[サーバーセキュリティの設定方法の 新規アイデンティティストアでの管理インターフェイスのセキュア保護](#) を参照してください。たとえば、管理コンソールを LDAP でセキュリティ保護するには、以下を実行します。



注記

JBoss EAP サーバーで Active Directory LDAP サーバーが使用する場合など、パスワードを読み取るパーミッションがない場合は、定義済みの LDAP レلم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 サーバーの代わりに LDAP サーバーで検証を直接実行できます。

LDAP アイデンティティストアの例

```
/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 接続の双方向 SSL/TLS での Elytron の使用

LDAP を使用して管理インターフェイスをセキュアにする場合は、双方向 SSL/TLS を使用するように アウトバウンド LDAP 接続を設定できます。これには、**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

レガシー **security** サブシステムを使用して管理インターフェイスの認証ソースとして LDAP ディレクトリーサーバーを使用するには、以下の手順を実行する必要があります。

- 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、セキュリティドメインの **Database** ログインモジュールと使用するデータソースを作成する場合と似ています。

LDAP 送信接続には、以下の属性を使用することができます。

属性	必須	説明
url	必要	ディレクトリーサーバーの URL アドレス。
search-dn	不要	検索の実行を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	不要	<p>検索の実行を許可されているユーザーのパスワード。この要素でサポートされる属性は次のとおりです。</p> <ul style="list-style-type: none"> ● store: 検索認証情報の取得元となるクレデンシャルストアへの参照。 ● alias: 参照ストア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	不要	<p>接続を確立するときに使用する最初の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。</p>
security-realm	不要	<p>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。</p>
referrals	不要	<p>検索の実行時に参照元が見つかったとき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。</p> <ul style="list-style-type: none"> ● IGNORE: デフォルトのオプション。参照元を無視します。 ● FOLLOW: 検索中に参照元が見つかったと、使用中の DirContext はその参照元の照会に従おうとします。この属性は、2 番目のサーバーに接続するときと同じ接続設定を使用でき、参照元で使用される名前に到達できることを前提としています。 ● THROW: DirContext が、参照元が必要であることを示す LdapReferralException の例外を送出します。セキュリティーレルムは、参照元に使用する別の接続を特定するように処理し、試行します。

属性	必須	説明
always-send-client-cert	不要	デフォルトではサーバーのクライアント証明書は、ユーザーの認証情報の確認中には送信されません。 true に設定すると、常に送信されます。
handles-referrals-for	不要	接続が処理できる参照元を指定します。URI の一覧を指定する場合には、URI はスペースで区切る必要があります。この属性で、接続プロパティのある接続を定義して、参照元の照会に従うのに別の認証情報が必要な場合に使用できます。これは、2 番目のサーバーの認証に異なる認証情報が必要な場合や、JBoss EAP のインストールから到達できない参照元の名前をサーバーが返す場合など、代替りのアドレスを使用できるので便利です。



注記

search-dn および **search-credential** は、ユーザーが指定した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 サーバー間の最初の接続を確立するためのものです。この接続を使用することで、JBoss EAP は、今後認証を試行するユーザーの DN を検索できます。認証を行うユーザーの DN (検索の結果) とユーザーが入力したパスワードを使用して、2 番目の接続を別に確立して、認証プロセスを完了します。

以下の LDAP サーバーの例の場合に、以下のような管理 CLI コマンドを使用して、アウトバウンド LDAP 接続を設定します。

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を追加する 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 サーバーとの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、[アウトバウンド LDAP 接続での SSL/TLS の使用](#) を参照してください。

- LDAP 対応のセキュリティーレلمを新たに作成します。
アウトバウンド LDAP 接続が作成されると、その接続を使用するために新しい LDAP 対応セキュリティーレلمを作成する必要があります。

LDAP セキュリティーレلمは次の設定属性を使用します。

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ためのコンテキストの DN。
recursive	LDAP ディレクトリーツリー全体にわたって再帰的に検索を行うか、指定の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。その後この情報を使用して、ユーザーの入力完了と同時に認証をテストします。デフォルトは dn です。
allow-empty-passwords	この属性は、空のパスワードを受け入れ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と、指定の属性を照合する簡単な検索を実行します。
advanced-filter	指定のユーザー ID をもとにしたユーザー検索に使用する完全に定義されたフィルター。この属性の標準の LDAP 構文内には、フィルタークエリーが含まれます。フィルターには、 {0} 形式の変数を含める必要があります。これは後で、ユーザーが指定したユーザー名に置き換えられます。詳細および advanced-filter の例は、 Combining LDAP and RBAC for Authorization section を参照してください。

**警告**

セキュリティの面で深刻な問題をもたらす可能性があるため、空白の LDAP パスワードは使用できないようにすることが重要です。この動作が環境で特に必要される場合を除き、空のパスワードは使用できず、**allow-empty-passwords** は **false** のままになります。

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. 管理インターフェイスの新しいセキュリティレルムを参照します。セキュリティレルムが作成されており、アウトバウンド LDAP 接続を使用している場合には、管理インターフェイスで新しいセキュリティレルムを参照する必要があります。

```
/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 接続での双方向 SSL/TLS の使用

以下の手順に従って、SSL/TLS でセキュリティを確保したアウトバウンド 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レルムを設定します。
セキュリティレルムには、JBoss EAP サーバーが LDAP サーバーとの間の通信の復号化/暗

号化に使用するキーを使用して設定したキーストアを含める必要があります。このキーストアを使用することで、JBoss EAP インスタンスは LDAP サーバーに対して自己検証もできるようになります。セキュリティレルムには、LDAP サーバーの証明書または、LDAP サーバーの証明書の署名に使用する認証局の証明書を含むトラストストアを含める必要があります。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の設定方法の管理インターフェイス向けの双方向 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レルムを使用してアウトバウンド LDAP 接続を作成します。
[レガシーのコア管理認証の使用](#) で定義されたプロセスと同様に、アウトバウンド LDAP 接続が作成されますが、LDAP サーバーと SSL/TLS セキュリティレルムに SSL/TLS URL を使用します。

LDAP サーバーのアウトバウンド LDAP 接続と SSL/TLS セキュリティレルムを作成したら、その情報を使用してアウトバウンド LDAP 接続を最新の情報に更新する必要があります。

SSL/TLS URL を使用して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. 管理インターフェイスでできるように、アウトバウンド LDAP 接続を使用する新しいセキュリティレルムを作成します。
[レガシーのコア管理認証の使用](#) の指示に含まれる [LDAP 対応のセキュリティレルムの新規作成](#) および [管理インターフェイスでの新規セキュリティレルムの参照](#) の手順を実行します。



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、無制限にすべてのアクセス権を付与することなく、さまざまな管理責任を付与できます。RBAC の詳細は、JBoss EAP [セキュリティ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 は承認にのみ使用され、認証は個別に処理されます。LDAP は認証と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。

- 承認にのみ RBAC を使用し、LDAP、または別のメカニズムは認証にのみ使用する。
- RBAC と LDAP を統合して使用し、管理インターフェイスで承認の意思決定を行う。

5.3.1. LDAP および RBAC の個別使用

JBoss EAP では、セキュリティレームで認証と承認を個別に分けて設定できます。そのため、LDAP を認証メカニズムとして、RBAC を承認メカニズムとして設定できます。このように分けて設定されている場合には、ユーザーが管理インターフェイスにアクセスしようとする、最初に設定済みの LDAP サーバーを使用して認証されます。成功すると、ユーザーのロールおよびそのロールに設定されたパーミッションは、LDAP サーバーにあるグループ情報とは独立して RBAC だけを使用し、決定されます。

RBAC を管理インターフェイ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の設定方法](#) を参照してください。管理インターフェイスで認証できるように LDAP を設定する方法は、[前述のセクション](#) を参照してください。

5.3.2. 承認用に LDAP と RBAC を統合する手順

LDAP サーバーまたはプロパティファイルを使用して認証されたユーザーは、ユーザーグループのメンバーにすることができます。ユーザーグループは、1つ以上のユーザーに割り当てることができる任意のラベルです。RBAC では、このグループ情報を使用して自動的にロールをユーザーに割り当てたり、ロールからユーザーを除外したりする設定が可能です。

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ており、このエントリーは属性によりクロス参照されます。LDAP サーバーの設定に応じて、ユーザーエンティティはユーザーが属するグループを **memberOf** 属性で、グループエンティティは **uniqueMember** 属性またはこの2つの組み合わせを使用してそのグループに所属するユーザーをマッピングできます。ユーザーが LDAP サーバーに正常に認証されると、グループ検索を実行してそのユーザーのグループ情報を読み込みます。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 (通常、認証で使用するユーザー名)、またはディレクトリー内のユーザーのエントリーの DN を使用して実行できます。グループ検索 (**group-search**) およびユーザー名と識別名 (**username-to-dn**) のマッピングは、セキュリティレームで LDAP を承認メカニズムとして設定する時に設定されます。

ユーザーのグループメンバーシップ情報が LDAP サーバーをもとに判断されると、RBAC 設定内のマッピングを使用して、ユーザーのロールが決まります。このマッピングを設定して、明示的にグループおよび個別ユーザーを包含または除外します。



注記

サーバーに接続するユーザーの認証ステップは常に最初に行われます。ユーザーが正常に認証されると、サーバーのグループが読み込まれます。認証ステップと承認ステップではいずれも、LDAP サーバーに接続する必要があります。セキュリティレームは、グループの読み込みステップの認証接続を再利用することで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の検索時に使用するスタイルは2つ (**Principal to Group** および **Group to Principal**) あります。Principal to Group には、所属グループへの参照を含むユーザーのエントリー (**memberOf** 属性を使用) が含まれます。Group to Principal には、対象のグループに所属するユーザーへの参照を含むグループのエントリー (**uniqueMember** を使用) が含まれます。



注記

JBoss EAP は、Principal to Group と Group to Principal の検索の両方をサポートしますが、Group to Principal よりも Principal to Group が推奨されます。Principal to Group を使用すると、検索を行わずに既知の識別名の属性を読み取り、直接グループ情報を読み込むことができます。Group to Principal には、ユーザーを参照する全グループを識別するための大規模な検索が必要です。

Principal to Group および 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	この属性を使用して、グループ名に使用すべきフォームを指定します。このグループ名は、対象のユーザーが所属するグループの一覧として返されます。これには、グループ名の単純形式またはグループの識別名のいずれかを使用できます。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	この属性を使用して、ユーザーが所属するグループを特定した後に、グループをもとに繰り返し検索を行い、グループに所属するグループを特定します。反復検索が有効な場合は、他のグループやサイクルが検出されると、メンバーではないグループに到達するまで継続されます。デフォルトは false です。
group-dn-attribute	属性が識別名であるグループのエントリ。デフォルトは dn です。
group-name-attribute	属性が簡単な名前のグループのエントリ。デフォルトは uid です。



注記

巡回群メンバーシップは問題ではありません。検索済みのグループを再び検索しないように、各検索の記録が保持されます。



重要

反復検索を機能させるには、グループエントリとユーザーエントリの内容を同じようにする必要があります。ユーザーが所属するグループの特定に使用するアプローチと同じアプローチを使用して、グループが所属するグループを特定します。グループ間のメンバーシップの場合に、クロス参照に使用する属性名が変更されるか、参照の方向が変更されると、このアプローチでグループが所属するグループは特定できません。

グループ検索の Principal to Group (memberOf)

GroupOne に所属するユーザー **TestUserOne** があり、**GroupOne** が **GroupFive** に所属する場合の例を見ていきます。グループメンバーシップは、メンバーレベルで **memberOf** 属性を使用すると表示されます。つまり、**TestUserOne** の **memberOf** 属性は、**GroupOne** の **dn** に設定されます。このような場合に **GroupOne** の **memberOf** 属性は **GroupFive** の **dn** に設定されます。

この種の検索を使用するには、**principal-to-group** 要素が **group-search** 要素に追加されます。

Principal to Group, memberOf の設定

```
/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。認証メカニズムも [このセクションで前述したように](#) 設定しておく必要があります。

group-attribute 属性が **group-search=principal-to-group** と合わせて使用されている点に注意してください。詳細は以下を参照してください。

表5.2 principal-to-group

属性	説明
group-attribute	ユーザーがメンバーであるグループの識別名と一致するユーザーエントリーの属性名。デフォルトは memberOf です。
prefer-original-connection	この値を使用して、参照元の照会情報に従う場合にどのグループ情報を優先するかを指定します。プリンシパルがロードされるたびに、続いて各グループメンバーシップの属性が読み込まれます。また、属性が読み込まれるたびに、元の接続または最後の参照元からの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 のグループ検索

GroupOne に所属するユーザー **TestUserOne** があり、**GroupOne** が **GroupFive** に所属する場合の Principal to Group と同様の例を見ていきます。ただし、今回の例では、グループのメンバーシップは、グループレベルで設定されている **uniqueMember** 属性を使用して表示されます。つまり、**GroupFive** の **uniqueMember** は **GroupOne** の **dn** に、**GroupOne** の **uniqueMember** は **TestUserOne** の **dn** に設定されます。

この種の検索を使用するには、**group-to-principal** 要素が **group-search** 要素に追加されます。

Group to Principal、uniqueMember の設定

```
/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=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。認証メカニズムも [このセクションで前述したように](#) 設定しておく必要があります。

principal-attribute 属性は **group-search=group-to-principal** と合わせて使用されている点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の検索方法を、**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のに使用します。

詳細は以下を参照してください。

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フォーム。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	この値を使用して、参照元の照会情報に従う場合にどのグループ情報を優先するかを指定します。プリンシパルがロードされるたびに、続いて各グループメンバーシップの属性が読み込まれます。また、属性が読み込まれるたびに、元の接続または最後の参照元からの接続のいずれかを使用できます。

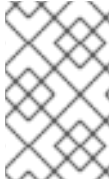
表5.4 membership-filter

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、以下の両方に該当する場合にのみ必要です。

- 認証および承認手順は異なる LDAP サーバーに対するものである。
- グループ検索が識別名を使用する。



注記

セキュリティレームが LDAP およびケルベロス認証の両方をサポートするインスタンスや、ケルベロス用に変換が必要なインスタンスにも該当します。LDAP 認証が実行された場合には、認証時に検出された DN を使用できます。

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	認証中のユーザー名から識別名のマッピング検索の結果はキャッシュされ、force 属性が false に設定されている場合の承認クエリー時に再利用されます。force が true の場合には、グループの読み込み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と承認を実行する場合に行われます。

username-to-dn は、以下のいずれかで設定できます。

username-is-dn

リモートユーザーによって入力されたユーザー名はユーザーの識別名であると指定します。

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
```

これは 1:1 マッピングを定義し、追加の設定はありません。

username-filter

入力されたユーザー名を照会し、指定の属性を検索します。

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	検索がコンテキストのサブコンテキストを拡張するかどうか。デフォルトは false です。
attribute	入力したユーザー名に対して照合を試行するユーザーのエントリーの属性です。デフォルトは uid です。
user-dn-attribute	ユーザーの識別名を取得するために読み込む属性です。デフォルトは dn です。

advanced-filter

このオプションは、カスタムのフィルターを使用してユーザーの識別名を見つけます。

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 の例と同じ属性は、意味とデフォルト値も同じです。属性がもう 1 つあります。

属性	説明
filter	ユーザーのエントリーの検索に使用するカスタムフィルター。ユーザー名は {0} のプレースホルダーに置き換えられます。



重要

これは、フィルターの定義後も有効でなければなりません。特殊文字を使用する場合には (例 **&**) 正しい形式を使用するようにしてください。たとえば、**&** 文字には **&** を使用します。

5.3.2.3. LDAP グループ情報の RBAC ロールへのマッピング

LDAP サーバーへの接続を作成してグループ検索を適切に設定したら、LDAP グループと RBAC ロールの間にマッピングを作成する必要があります。このマッピングには、包含と除外の設定両方があり、グループメンバーシップをもとに 1 つまたは複数のロールを自動的にユーザーに割り当てることができます。



警告

RBAC がまだ設定されていない場合は、特に新規作成した LDAP 対応のレルムに切り替える場合など、設定時には最新の注意を払ってください。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 管理インターフェイ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 を使用する場合は [管理 CLI ガイド](#) を参照してください。

RBAC が有効で設定されていることの確認

RBAC を有効にして初期設定してから、LDAP と RBAC ロールの間のマッピングを使用する必要があります。

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

この設定では、以下のような結果になります。

```
{ "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)
```

上記の設定で、以下のようにロールの一覧が生成されます。

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

さらに、ロールの既存のマッピングをすべて確認できます。

```
/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"
        }
      }
    }
}
```


ロールマッピングエントリーの設定

ロールに **Role-Mapping** エントリーがまだない場合は、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には、以下を実行します。

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

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

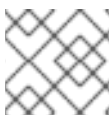
確認するには、以下を実行します。

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

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

ロールにグループを追加して包含または除外設定

ロールに対してグループを追加することで包含または除外設定できます。



注記

除外マッピングが優先され、それ以外は包含マッピングが優先されます。

包含するグループを追加するには、以下を実行します。

```
/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するには、以下を実行します。

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

結果を確認するには、以下を実行します。

```
/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 ロールグループから除外または包含されないようにグループを削除する手順

包含されていたグループを削除するには、以下を実行します。

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

除外されていたグループを削除するには、以下を実行します。

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

5.4. キャッシングの有効化

セキュリティーレルムには、認証とグループ読み込みの両方の LDAP クエリーの結果をキャッシュする機能もあります。この機能により、グループのグループメンバーシップ情報を反復的にクエリーする場合など、特定の状況で異なるユーザーが複数の検索で複数のクエリーの結果を再利用できます。3つの異なるキャッシュが利用でき、それぞれが個別に設定され、独立して動作します。

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

5.4.1. キャッシュ設定

キャッシュは相互に独立していますが、上記3つともすべて、同じ方法で設定されます。キャッシュごとに、以下の設定オプションがあります。

属性	説明
type	この属性は、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、最終アクセスから一定期間が経過すると、キャッシュからアイテムが退避されます。 by-search-time は、最終アクセスのタイミングに関係なく、アイテムがキャッシュに保存されていた期間をもとに退避されます。
eviction-time	これは、ストラテジーに応じてエビクションに費やす時間 (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ュを有効または無効にするブール値です。この設定により、検索に失敗しているにも拘らず、同じ検索が LDAP サーバーに反復的にアクセスしないようにできる可能性があります。存在しないユーザーの検索でキャッシュがいっぱいになってしまう可能性もあります。この設定は特に、認証キャッシュに重要です。
max-cache-size	この属性は、キャッシュの最大サイズ (アイテム数) を定義し、アイテムの退避を開始するタイミングを指定できます。必要に応じて新しい認証や検索で使用できるように、古いアイテムはキャッシュから退避されます。つまり max-cache-size を指定すると、新しい認証の試行や検索が行われなくなります。

5.4.2. 例



注記

この例では、**LDAPRealm** という名前のセキュリティレルムが作成されていることを前提としています。このセキュリティレルムは、既存の LDAP サーバーに接続して、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の詳細は [現在のキャッシュ設定の読み取り](#) を参照してください。LDAP を使用するセキュリティレルムの作成に関する詳細は、[レガシーコア管理認証の使用](#) 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 が指定されているエリアはすべて、キャッシュを設定できます。

認證

認証時に、この定義を使用してユーザーの識別名を検出し、LDAP サーバーへの接続を試行し、この認証情報を使用してアイデンティティーが作成されたことを確認します。

group-search 定義

グループ検索の定義です。今回は上記の設定例で **iterative** が **true** に指定されているので反復検索になっています。まず、ユーザーが直接所属するグループをすべて検索します。その後、これらのグループごとに検索が実行され、他のグループに所属しているかどうかを特定します。このプロセスは、循環参照が検出されるか、最後のグループが所属するグループがなくなるまで継続されます。

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有無に依存します。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を2回目に試行する時などに使用できます。これは、ローカル認証など、2つ目の認証形式がサポートされる場合に便利で、必要になる場合さえもあります。

5.4.2.1. 現在のキャッシュ設定の読み取り



注記

これ以降のセクションで使用する CLI コマンドでは、**セキュリティレルム** の名前に LDAPRealm を使用します。これは、実際に設定するレルムの名前に置き換える必要があります。

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

このセクション以降で使用する管理 CLI コマンドは、セキュリティレームの authentication セクション (**authentication=ldap/**) でキャッシュを設定します。また、承認セクションのキャッシュは、コマンドのパスの更新と同様の方法で設定できます。

キャッシュを有効化する管理 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)
```

このコマンドでは、退避時間 300 秒 (5 分)、最大キャッシュサイズ 100 個で、認証用の **by-access-time** キャッシュを追加します。さらに、検索に失敗した検索が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. 既存のキャッシュ検証

既存のキャッシュを確認する管理 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 **cache-size** を追加し、これでキャッシュの現在のアイテム数が表示されます。上記の出力では、このアイテム数は 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、**TestUserOne** 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1つ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することができます。

アイテム1つをフラッシュする管理 CLI コマンド

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

全キャッシュをフラッシュする管理 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手順

セキュリティーマッピングをセキュリティードメインに追加すると、認証または承認が済んでから、情報をアプリケーションに渡す前に認証情報と承認情報を統合できます。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、コード、タイプ、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**などのショートネーム、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を指定します。**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表します。このフィールドの許容値は **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: 既存のセキュリティードメインへの SimpleRoles セキュリティーマッピングを追加する管理 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.管理対象ドメインの考慮事項

Microsoft Active Directory などの LDAP サーバーを使用したアイデンティティ管理の設定は、スタンドアロンサーバーまたは管理対象ドメインで使われるかどうかに関係なく基本的に同じです。通常、セキュリティルールおよびセキュリティドメイン両方で、アイデンティティストアを設定する場合も当てはまります。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、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 参考資料

A.1. WILDFLY-CONFIG.XML の例

クライアントが Elytron Client を使用する 1 つの方法として **wildfly-config.xml** ファイルがあり、クライアントは JBoss EAP への接続時にセキュリティ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、[Elytron クライアントによるクライアント認証の設定](#) 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>
  </authentication-client>
</configuration>
```

```

<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の参照

SSO 認証メカニズムの設定。

これは、**undertow** サブシステムにある **application-security-domain** の **setting=single-sign-on** 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	使用するクッキードメイン。
path	クッキーのパス。
http-only	クッキーの httpOnly 属性を設定する場合。
secure	クッキーの secure 属性を設定する場合。
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	<p>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。</p> <p>credential-reference には以下の属性があります。</p> <ul style="list-style-type: none"> ● store: 認証情報へのエイリアスを格納するクレデンシャルストアの名前。 ● alias: ストアに保存されているシークレットまたは認証情報を示すエイリアス。 ● type: 参照がマークされ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するシークレット。クレデンシャルストアが認証情報またはシークレットをサービスに提供する方法を確認します。
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をもとにパスワードを作成します。

- 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
- モジュール暗号化

パスワードマッパーには以下の属性があります。



注記

どのマッパーも最初のコラムのインデックスは、**1** になります。

表A.2 パスワードマッパーの属性

マッパー名	属性	暗号化方法
-------	----	-------

マッパー名	属性	暗号化方法
clear-password-mapper	<ul style="list-style-type: none"> ● 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	<ul style="list-style-type: none"> ● 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 使用するハッシュアルゴリズム。以下の値がサポートされます。 <ul style="list-style-type: none"> ○ 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。使用できる値: <ul style="list-style-type: none"> ○ base64 (デフォルト) ○ hex 	簡単なハッシュメカニズムを使用します。

マッパー名	属性	暗号化方法
salted-simple-digest	<ul style="list-style-type: none"> ● 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 使用するハッシュアルゴリズム。以下の値がサポートされます。 <ul style="list-style-type: none"> ○ 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 ハッシュに使用するソルトを含むコラムのインデックス。 ● hash-encoding ハッシュの表現を指定します。使用できる値: <ul style="list-style-type: none"> ○ base64 (デフォルト) ○ hex ● salt-encoding ソルトの表現を指定します。使用できる値: <ul style="list-style-type: none"> ○ base64 (デフォルト) ○ hex 	ソルトには単純なハッシュメカニズムを使用します。

マッパー名	属性	暗号化方法
bcrypt-password-mapper	<ul style="list-style-type: none"> ● 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 ハッシュに使用するソルトを含むコラムのインデックス。 ● iteration-count-index 使用する反復数を含むコラムのインデックス。 ● hash-encoding ハッシュの表現を指定します。使用できる値: <ul style="list-style-type: none"> ○ base64 (デフォルト) ○ hex ● salt-encoding ソルトの表現を指定します。使用できる値: <ul style="list-style-type: none"> ○ base64 (デフォルト) ○ hex 	ハッシュ化に使用する Blowfish アルゴリズム。

マッパー名	属性	暗号化方法
scram-mapper	<ul style="list-style-type: none"> ● 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 使用するハッシュアルゴリズム。以下の値がサポートされます。 <ul style="list-style-type: none"> ○ 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に使用されます。 ● iteration-count-index 使用する反復数を含むコラムのインデックス。 ● hash-encoding ハッシュの表現を指定します。使用できる値: <ul style="list-style-type: none"> ○ base64 (デフォルト) ○ hex ● salt-encoding ソルトの表現を指定します。使用できる値: <ul style="list-style-type: none"> ○ base64 (デフォルト) ○ hex 	<p>Salted Challenge Response Authentication メカニズムはハッシュに使用します。</p>
modular-crypt-mapper	<ul style="list-style-type: none"> ● password-index 暗号化されたパスワードを含むコラムのインデックス。 	<p>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エンコードできるようになります。</p>

