Red Hat JBoss Enterprise Application Platform 8.0

複数のアイデンティティーストアを使用したアプリケーションと管理インターフェイスの保護

ファイルシステム、データベース、LDAP (Lightweight Directory Access Protocol) などの複数のアイデンティティーストア、またはカスタムアイデンティティーストアを併用して JBoss EAP 管理インターフェイスおよびデプロイされたアプリケーションをセキュアにするためのガイド

Last Updated: 2024-02-09

Red Hat JBoss Enterprise Application Platform 8.0 複数のアイデンティティーストアを使用したアプリケーションと管理インターフェイスの保護

ファイルシステム、データベース、LDAP (Lightweight Directory Access Protocol) などの複数のアイデンティティーストア、またはカスタムアイデンティティーストアを併用して JBoss EAP 管理インターフェイスおよびデプロイされたアプリケーションをセキュアにするためのガイド

法律上の通知

概要

ファイルシステム、データベース、LDAP (Lightweight Directory Access Protocol) などの複数のアイデンティティーストア、またはカスタムアイデンティティーストアを併用して JBoss EAP 管理インターフェイスおよびデプロイされたアプリケーションをセキュアにするためのガイド。

目次

JBOSS EAP ドキュメントへのフィードバック (英語のみ)

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウントにログインし、課題を送信してください。Red Hat Jira アカウントをお持ちでない場合は、アカウントを作成するように求められます。

手順

1. **このリンクをクリック** してチケットを作成します。
2. **Summary** に課題の簡単な説明を入力します。
3. **Description** に課題や機能拡張の詳細な説明を入力します。問題があるドキュメントのセクションへの URL を含めてください。
4. **Submit** をクリックすると、課題が作成され、適切なドキュメントチームに転送されます。

多様性を受け入れるオープンソースの強化

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

第1章 ID ストアの設定

1.1. 集約レルムの作成

1.1.1. Elytron の集約レルム

aggregate-realm を使用すると、認証と別のセキュリティーレルムにいずれかのセキュリティーレルムを使用したり、Elytron での認可に複数のセキュリティーレルムの集約を使用したりできます。

たとえば、認証に **ldap-realm** を使用するように **aggregate-realm** を設定し、認可に **filesystem-realm** と **ldap-realm** を集約できます。

アイデンティティーは、次の方法で、複数の認可レルムで設定された集約レルム内に作成されます。

- 各認可レルムの属性値がロードされます。
- 属性が複数の認可レルムで定義されている場合は、最初に出現した属性の値が使用されます。

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

集約レルム設定の例

```
/subsystem=elytron/aggregate-realm=exampleSecurityRealm:add(authentication-
realm=exampleLDAPRealm,authorization-realms=[exampleLDAPRealm,exampleFileSystemRealm])
```

この例では、設定された **aggregate-realm** は、LDAP レルムである "exampleLDAPRealm" とファイルシステムレルムである "exampleFilesystemRealm" の 2 つの既存のセキュリティーレルムを参照します。

- LDAP レルムから取得した属性値:

```
mail: administrator@example.com
telephoneNumber: 0000 0000
```

- ファイルシステムレルムから取得した属性値:

```
mail: user@example.com
website: http://www.example.com/
```

集約レルムから取得した生成アイデンティティー:

```
mail: administrator@example.com
telephoneNumber: 0000 0000
website: http://www.example.com/
```

LDAP レルムはファイルシステムレルムの前に参照されるため、**aggregate-realm** の例では、LDAP レルムで定義された属性 **mail** の値を使用します。

関連情報

- Elytron での **aggregate-realm** の作成

1.1.2. 集約レルムに必要なセキュリティーレルムの作成例

以下の例では、**ldap-realm** および **filesystem-realm** を作成します。これらのセキュリティー レルムは **aggregate-realm** で参照できます。

1.1.2.1. Elytron で ldap-realm を作成する例

Lightweight Directory Access Protocol (LDAP) ID ストアでサポートされる Elytron セキュリティーレルムを作成して、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護します。

この手順の例では、以下の LDAP Data Interchange Format (LDIF) が使用されます。

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

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: passwordUser1
mail: administrator@example.com
telephoneNumber: 0000 0000

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=user1,ou=Users,dc=wildfly,dc=org
```

この例で使用されている LDAP 接続パラメーターは以下のとおりです。

- LDAP URL: **ldap://10.88.0.2**
- LDAP 管理者パスワード: **secret**
  これは、Elytron が LDAP サーバーに接続するために必要です。
- LDAP 管理者識別名 (DN): **(cn=admin,dc=wildfly,dc=org)**
- LDAP 組織: **wildfly**
  組織名が指定されていない場合、デフォルトで **Example Inc** になります。
- LDAP ドメイン: **wildfly.org**
  これは、プラットフォームが LDAP 検索リファレンスを受信したときに照合される名前です。

前提条件

- LDAP ID ストアを設定している。
- JBoss EAP が実行されている。

手順

1. LDAP サーバーへの接続に使用される URL とプリンシパルを提供するディレクトリーコンテキストを設定します。

```
/subsystem=elytron/dir-context=<dir_context_name>:add(url="<LDAP_URL>",principal="<principal_distinguished_name>",credential-reference=<credential_reference>)
```

例

```
/subsystem=elytron/dir-context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",credential-reference={clear-text="secret"})
{"outcome" => "success"}
```

2. ディレクトリーコンテキストを参照する LDAP レルムを作成します。検索ベース DN とユーザーのマッピング方法を指定します。

構文

```
/subsystem=elytron/ldap-realm=<ldap_realm_name>add:(dir-context=<dir_context_name>,identity-mapping=search-base-dn="ou=<organization_unit>,dc=<domain_component>",rdn-identifier="<relative_distinguished_name_identifier>",user-password-mapper={from=<password_attribute_name>},attribute-mapping=[{filter-base-dn="ou=<organization_unit>,dc=<domain_component>",filter="<ldap_filter>",from="<ldap_attribute_name>",to="<identity_attribute_name>"}]})
```

例

```
/subsystem=elytron/ldap-realm=exampleLDAPRealm:add(dir-context=exampleDirContext,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={1}))",from="cn",to="Roles"},{from="mail",to="mail"},{from="telephoneNumber",to="telephoneNumber"}]})
{"outcome" => "success"}
```

これで、このレルムを使用してセキュリティードメインを作成したり、**failover-realm**、**distributed-realm** または **aggregate-realm** で別のレルムと組み合わせたりすることができます。

1.1.2.2. Elytron で filesystem-realm を作成する例

ファイルシステムベースの ID ストアにサポートされる Elytron セキュリティーレルムを作成して、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護します。

前提条件

- JBoss EAP が実行されている。

手順

1. Elytron で **filesystem-realm** を作成します。

   構文

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>)
   ```

   例

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add(path=fs-realm-
   users,relative-to=jboss.server.config.dir)
   {"outcome" => "success"}
   ```

2. レルムにユーザーを追加し、ユーザーのロールを設定します。

   a. ユーザーを追加します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-
      identity(identity=<user_name>)
      ```

      例

      ```
      /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-
      identity(identity=user1)
      {"outcome" => "success"}
      ```

   b. ユーザーのロールを設定します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-
      attribute(identity=<user_name>,name=<roles_attribute_name>, value=
      [<role_1>,<role_N>])
      ```

      例

      ```
      /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-
      attribute(identity=user1, name=Roles, value=["Admin","Guest"])
      {"outcome" => "success"}
      ```

   c. ユーザーの属性を設定します。

      構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-attribute(identity=<user_name>,name=<attribute_name>, value=[<attribute_value>])
```

例

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-attribute(identity=user1, name=mail, value=["user@example.com"])
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-attribute(identity=user1, name=website, value=["http://www.example.com/"])
```

これで、このレルムを使用してセキュリティードメインを作成したり、**failover-realm**、**distributed-realm** または **aggregate-realm** で別のレルムと組み合わせたりすることができます。

1.1.3. Elytron での aggregate-realm の作成

Elytron で、認証用に 1 つのセキュリティーレルムを使用し、認可用に複数のセキュリティーレルムを集約する **aggregate-realm** を作成します。**aggregate-realm** を使用してセキュリティードメインを作成し、管理インターフェイスおよびデプロイされたアプリケーションに認証および認可を追加します。

前提条件

- JBoss EAP が実行されている。
- 集約レルムから参照するレルムを作成しました。

手順

1. 既存のセキュリティーレルムから **aggregate-realm** を作成します。

   構文

   ```
   /subsystem=elytron/aggregate-realm=<aggregate_realm_name>:add(authentication-realm=<security_realm_for_authentication>, authorization-realms=[<security_realm_for_authorization_1>,<security_realm_for_authorization_2>,...,<security_realm_for_authorization_N>])
   ```

   例

   ```
   /subsystem=elytron/aggregate-realm=exampleSecurityRealm:add(authentication-realm=exampleLDAPRealm,authorization-realms=[exampleLDAPRealm,exampleFileSystemRealm])
   {"outcome" => "success"}
   ```

2. 属性をロールにマップするロールデコーダーを作成します。

   構文

   ```
   /subsystem=elytron/simple-role-decoder=<role_decoder_name>:add(attribute=<attribute>)
   ```

   例

```
/subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
{"outcome" => "success"}
```

3. **aggregate-realm** およびロールデコーダを参照するセキュリティードメインを作成します。

構文

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

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm,role-decoder="from-roles-attribute"}])
{"outcome" => "success"}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **aggregate-realm** 属性
- **security-domain** 属性
- **simple-role-decoder** 属性

1.2. キャッシングレルムの作成

1.2.1. Elytron のキャッシングレルム

Elytron には、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する **caching-realm** があります。**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。

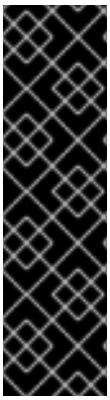
以下のセキュリティーレルムで **caching-realm** を使用できます。

- **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ッ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。**ldap-realm** のみがリッスンをサポートします。ただし、**ldap-realm** 内の **roles** などのフィルタリングされた属性は、リスニングをサポートしていません。

キャッシングレルムにユーザーデータの正しいキャッシュがあることを確認するには、以下を必ず行います。

- 認証情報ソースでユーザー属性を変更した後、**caching-realm** キャッシュをクリアします。
- 認証情報ソースではなく、キャッシングレルムを通じてユーザー属性を変更します。



重要

キャッシュレル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。

関連情報

- Elytron での **caching-realm** の作成
- **caching-realm** 属性
- **caching-realm** キャッシュのクリア

1.2.2. Elytron での caching-realm の作成

JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護するために、レルムを参照する **caching-realm** およびセキュリティードメインを作成します。

注記

キャッシングレルムとして設定された **ldap-realm** は、Active Directory をサポートしていません。詳細は、Changing LDAP/AD User Password via JBossEAP CLI for Elytron を参照してください。

前提条件

- セキュリティーレルムをキャッシュするように設定している。

手順

1. キャッシュするセキュリティーレルムを参照する **caching-realm** を作成します。

   構文

   ```
   /subsystem=elytron/caching-realm=<caching_realm_name>:add(realm=<realm_to_cache>)
   ```

   例

   ```
   /subsystem=elytron/caching-realm=exampleSecurityRealm:add(realm=exampleLDAPRealm)
   ```

2. **caching-realm** を参照するセキュリティードメインを作成します。

構文

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

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm}])
{"outcome" => "success"}
```

検証

- Elytron が **caching-realm** で参照されているセキュリティーレルムから **caching-realm** にデータをロードできるか確認するには、次のコマンドを使用します。

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:read-
identity(name=<username>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:read-identity(name=user1)
{
    "outcome" => "success",
    "result" => {
        "name" => "user1",
        "attributes" => {"Roles" => ["Admin"]},
        "roles" => ["Admin"]
    }
}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **caching-realm** 属性
- **caching-realm** キャッシュのクリア
- **security-domain** 属性

1.2.3. caching-realm キャッシュのクリア

caching-realm キャッシュをクリアすると、Elytron は、キャッシュするように設定されているセキュリティーレルムからの最新データを使用してキャッシュを再入力します。

前提条件

- **caching-realm** を設定している。

手順

- **caching-realm** キャッシュをクリアします。

  構文

  ```
  /subsystem=elytron/caching-realm=<caching_realm_name>:clear-cache
  ```

  例

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

関連情報

- **caching-realm** 属性

1.3. 分散レルムの作成

1.3.1. Elytron での分散レルム

分散レルムを使用すると、既存のセキュリティーレルムを参照して、異なるアイデンティティーストア全体で検索できます。取得した ID は、認証と認可の両方に使用されます。Elytron は、**distributed-realm** リソースで定義した順序で分散レルムのセキュリティーレルムを呼び出します。

distributed-realm の設定例

```
/subsystem=elytron/distributed-realm=exampleSecurityRealm:add(realms=
[exampleLDAPRealm,exampleFilesystemRealm])
```

この例では、設定された **distributed-realm** は、LDAP レルムである "exampleLDAPRealm" とファイルシステムレルムである "exampleFilesystemRealm" の 2 つの既存のセキュリティーレルムを参照します。Elytron は、参照されているセキュリティーレルムを以下のとおり順番に検索します。

- Elytron は最初に LDAP レルムに一致する ID を検索します。
- Elytron が一致を見つけた場合、認証は成功します。
- 一致するものが見つからない場合、Elytron は filesystem レルムを検索します。

デフォルトでは、アイデンティティーが一致する前にアイデンティティーストアへの接続が失敗した場合、認証が例外 **RealmUnavailableException** で失敗し、レルムはそれ以上検索されません。この動作を変更するには、属性 **ignore-unavailable-realms** を **true** に設定します。**ignore-unavailable-realms** が **true** に設定されていると、アイデンティティーストアへの接続が失敗した場合に Elytron が残りのレルムの検索を続けます。

ignore-unavailable-realms が **true** に設定されていると、**emit-events** がデフォルトで **true** に設定されているため、クエリーされたレルムのいずれかが利用できない場合に **SecurityEvent** が発行されます。これをオフにするには、**emit-events** を **false** に設定します。

関連情報

- Elytron での **distributed-realm** の作成

1.3.2. 分散レルムに必要なセキュリティーレルムの作成例

以下の例では、**ldap-realm** および **filesystem-realm** を作成します。これらのセキュリティーレルムは **distributed-realm** で参照できます。

1.3.2.1. Elytron で ldap-realm を作成する例

Lightweight Directory Access Protocol (LDAP) ID ストアでサポートされる Elytron セキュリティーレルムを作成して、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護します。

この手順の例では、以下の LDAP Data Interchange Format (LDIF) が使用されます。

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

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: userPassword1

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=user1,ou=Users,dc=wildfly,dc=org
```

この例で使用されている LDAP 接続パラメーターは以下のとおりです。

- LDAP URL: **ldap://10.88.0.2**
- LDAP 管理者パスワード: **secret**
  これは、Elytron が LDAP サーバーに接続するために必要です。
- LDAP 管理者識別名 (DN): **(cn=admin,dc=wildfly,dc=org)**

- LDAP 組織: **wildfly**
  組織名が指定されていない場合、デフォルトで **Example Inc** になります。

- LDAP ドメイン: **wildfly.org**
  これは、プラットフォームが LDAP 検索リファレンスを受信したときに照合される名前です。

前提条件

- LDAP ID ストアを設定している。
- JBoss EAP が実行されている。

手順

1. LDAP サーバーへの接続に使用される URL とプリンシパルを提供するディレクトリーコンテキストを設定します。

   ```
   /subsystem=elytron/dir-context=<dir_context_name>:add(url="<LDAP_URL>",principal="<principal_distinguished_name>",credential-reference=<credential_reference>)
   ```

   例

   ```
   /subsystem=elytron/dir-context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",credential-reference={clear-text="secret"})
   {"outcome" => "success"}
   ```

2. ディレクトリーコンテキストを参照する LDAP レルムを作成します。検索ベース DN とユーザーのマッピング方法を指定します。

   構文

   ```
   /subsystem=elytron/ldap-realm=<ldap_realm_name>add:(dir-context=<dir_context_name>,identity-mapping=search-base-dn="ou=<organization_unit>,dc=<domain_component>",rdn-identifier="<relative_distinguished_name_identifier>",user-password-mapper={from=<password_attribute_name>},attribute-mapping=[{filter-base-dn="ou=<organization_unit>,dc=<domain_component>",filter="<ldap_filter>",from="<ldap_attribute_name>",to="<identity_attribute_name>"}]})
   ```

   例

   ```
   /subsystem=elytron/ldap-realm=exampleLDAPRealm:add(dir-context=exampleDirContext,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={1}))",from="cn",to="Roles"}]})
   {"outcome" => "success"}
   ```

これで、このレルムを使用してセキュリティードメインを作成したり、**failover-realm**、**distributed-realm** または **aggregate-realm** で別のレルムと組み合わせたりすることができます。**ldap-realm** の

caching-realm を設定して、ルックアップの結果をキャッシュし、パフォーマンスを向上させることもできます。

1.3.2.2. Elytron で filesystem-realm を作成する例

ファイルシステムベースの ID ストアにサポートされる Elytron セキュリティーレルムを作成して、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護します。

前提条件

- JBoss EAP が実行されている。

手順

1. Elytron で **filesystem-realm** を作成します。

   構文

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>)
   ```

   例

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add(path=fs-realm-
   users,relative-to=jboss.server.config.dir)
   {"outcome" => "success"}
   ```

2. レルムにユーザーを追加し、ユーザーのロールを設定します。

   a. ユーザーを追加します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-
      identity(identity=<user_name>)
      ```

      例

      ```
      /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-
      identity(identity=user2)
      {"outcome" => "success"}
      ```

   b. ユーザーのパスワードを設定します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:set-
      password(identity=<user_name>, clear={password=<password>})
      ```

      例

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:set-
password(identity=user2, clear={password="passwordUser2"})
{"outcome" => "success"}
```

c. ユーザーのロールを設定します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-
attribute(identity=<user_name>, name=<roles_attribute_name>, value=
[<role_1>,<role_N>])
```

例

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-
attribute(identity=user2, name=Roles, value=["Admin","Guest"])
{"outcome" => "success"}
```

これで、このレルムを使用してセキュリティードメインを作成したり、**failover-realm**、**distributed-realm** または **aggregate-realm** で別のレルムと組み合わせたりすることができます。

1.3.3. Elytron での distributed-realm の作成

ID を検索するために、既存のセキュリティーレルムを参照する **distributed-realm** を Elytron に作成します。**distributed-realm** を使用してセキュリティードメインを作成し、管理インターフェイスおよびサーバーにデプロイされたアプリケーションに認証および認可を追加します。

前提条件

- JBoss EAP が実行されている。
- **distributed-realm** で参照するレルムを作成している。

手順

1. 既存のセキュリティーレルムを参照する **distributed-realm** を作成します。

   構文

   ```
   /subsystem=elytron/distributed-realm=<distributed_realm_name>:add(realms=
   [<security_realm_1>, <security_realm_2>, ..., <security_realm_N>])
   ```

   例

   ```
   /subsystem=elytron/distributed-realm=exampleSecurityRealm:add(realms=
   [exampleLDAPRealm, exampleFileSystemRealm])
   {"outcome" => "success"}
   ```

2. 属性をロールにマップするロールデコーダーを作成します。

   構文

```
/subsystem=elytron/simple-role-decoder=<role_decoder_name>:add(attribute=<attribute>)
```

例

```
/subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
{"outcome" => "success"}
```

3. **distributed-realm** およびロールデコーダを参照するセキュリティードメインを作成します。

構文

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

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm,role-decoder="from-roles-attribute"}])
{"outcome" => "success"}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **distributed-realm** 属性
- **security-domain** 属性
- **simple-role-decoder** 属性

1.4. フェイルオーバーレルムの作成

1.4.1. Elytron でのフェイルオーバーレルム

フェイルオーバーセキュリティーレルム (**failover-realm**) を Elytron で設定することができます。これは、2 つの既存のセキュリティーレルムを参照し、一方のセキュリティーレルムに障害が発生した場合に、Elytron がもう一方をバックアップとして使用するようにすることができます。

Elytron の **failover-realm** は、以下の 2 つのセキュリティーレルムを参照します。

- **delegate-realm**: 使用する主要なセキュリティーレルム。
- **failover-realm**: バックアップとして使用するセキュリティーレルム。

failover-realm 設定の例

```
/subsystem=elytron/failover-realm=exampleSecurityRealm:add(delegate-
realm=exampleLDAPRealm,failover-realm=exampleFileSystemRealm)
```

この例では、**ldap-realm** である **exampleLDAPRealm** がデリゲートレルムとして使用され、**filesystem-realm** である **exampleFileSystemRealm** が **failover-realm** として使用されています。**ldap-realm** に障害が発生した場合、Elytron は認証と認可に **filesystem-realm** を使用します。

注記

failover-realm では、**delegate-realm** に障害が発生した場合にのみ **failover-realm** が呼び出されます。**delegate-realm** への接続は成功したけれど、必要な ID が見つからない場合は、**fail-over** レルムは呼び出されません。複数のセキュリティーレルム間で ID を検索するには、**distributed-realm** を使用します。

1.4.2. フェイルオーバーレルムに必要なセキュリティーレルムの作成例

以下の例では、**ldap-realm** および **filesystem-realm** を作成します。これらのセキュリティーレルムは、**failover-realm** で参照できます。

1.4.2.1. Elytron で ldap-realm を作成する例

Lightweight Directory Access Protocol (LDAP) ID ストアでサポートされる Elytron セキュリティーレルムを作成して、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護します。

この手順の例では、以下の LDAP Data Interchange Format (LDIF) が使用されます。

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

dn: uid=user1,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: user1
sn: user1
uid: user1
userPassword: userPassword1

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=user1,ou=Users,dc=wildfly,dc=org
```

この例で使用されている LDAP 接続パラメーターは以下のとおりです。

- LDAP URL: **ldap://10.88.0.2**
- LDAP 管理者パスワード: **secret**
  これは、Elytron が LDAP サーバーに接続するために必要です。

- LDAP 管理者識別名 (DN): **(cn=admin,dc=wildfly,dc=org)**
- LDAP 組織: **wildfly**
  組織名が指定されていない場合、デフォルトで **Example Inc** になります。
- LDAP ドメイン: **wildfly.org**
  これは、プラットフォームが LDAP 検索リファレンスを受信したときに照合される名前です。

前提条件

- LDAP ID ストアを設定している。
- JBoss EAP が実行されている。

手順

1. LDAP サーバーへの接続に使用される URL とプリンシパルを提供するディレクトリーコンテキストを設定します。

```
/subsystem=elytron/dir-context=<dir_context_name>:add(url="<LDAP_URL>",principal="<principal_distinguished_name>",credential-reference=<credential_reference>)
```

例

```
/subsystem=elytron/dir-context=exampleDirContext:add(url="ldap://10.88.0.2",principal="cn=admin,dc=wildfly,dc=org",credential-reference={clear-text="secret"})
{"outcome" => "success"}
```

2. ディレクトリーコンテキストを参照する LDAP レルムを作成します。検索ベース DN とユーザーのマッピング方法を指定します。

構文

```
/subsystem=elytron/ldap-realm=<ldap_realm_name>add:(dir-context=<dir_context_name>,identity-mapping=search-base-dn="ou=<organization_unit>,dc=<domain_component>",rdn-identifier="<relative_distinguished_name_identifier>",user-password-mapper={from=<password_attribute_name>},attribute-mapping=[{filter-base-dn="ou=<organization_unit>,dc=<domain_component>",filter="<ldap_filter>",from="<ldap_attribute_name>",to="<identity_attribute_name>"}]})
```

例

```
/subsystem=elytron/ldap-realm=exampleLDAPRealm:add(dir-context=exampleDirContext,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={1}))",from="cn",to="Roles"}]})
{"outcome" => "success"}
```

これで、このレルムを使用してセキュリティードメインを作成したり、**failover-realm**、**distributed-realm** または **aggregate-realm** で別のレルムと組み合わせたりすることができます。**ldap-realm** の **caching-realm** を設定して、ルックアップの結果をキャッシュし、パフォーマンスを向上させることもできます。

1.4.2.2. Elytron で filesystem-realm を作成する例

ファイルシステムベースの ID ストアにサポートされる Elytron セキュリティーレルムを作成して、JBoss EAP サーバーインターフェイスまたはサーバーにデプロイされたアプリケーションを保護します。

前提条件

- JBoss EAP が実行されている。

手順

1. Elytron で **filesystem-realm** を作成します。

   構文

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>)
   ```

   例

   ```
   /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add(path=fs-realm-users,relative-to=jboss.server.config.dir)
   {"outcome" => "success"}
   ```

2. レルムにユーザーを追加し、ユーザーのロールを設定します。

   a. ユーザーを追加します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity(identity=<user_name>)
      ```

      例

      ```
      /subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity(identity=user1)
      {"outcome" => "success"}
      ```

   b. ユーザーのパスワードを設定します。

      構文

      ```
      /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:set-password(identity=<user_name>, clear={password=<password>})
      ```

      例

      -

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:set-
password(identity=user1, clear={password="passwordUser1"})
{"outcome" => "success"}
```

c. ユーザーのロールを設定します。

構文

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add-identity-
attribute(identity=<user_name>,name=<roles_attribute_name>, value=
[<role_1>,<role_N>])
```

例

```
/subsystem=elytron/filesystem-realm=exampleFileSystemRealm:add-identity-
attribute(identity=user1, name=Roles, value=["Admin","Guest"])
{"outcome" => "success"}
```

これで、このレルムを使用してセキュリティードメインを作成したり、**failover-realm**、**distributed-realm** または **aggregate-realm** で別のレルムと組み合わせたりすることができます。

1.4.3. Elytron での failover-realm の作成

Elytron で、既存のセキュリティーレルムをデリゲートレルム、使用するデフォルトのレルム、およびフェイルオーバーレルムとして参照するフェイルオーバーセキュリティーレルムを作成します。Elytron は、デリゲートレルムに障害が発生した場合、設定されたフェイルオーバーレルムを使用します。セキュリティーレルムを使用してセキュリティードメインを作成し、管理インターフェイスまたはサーバーにデプロイされたアプリケーションに認証と認可を追加します。

前提条件

- JBoss EAP が実行されている。
- デリゲートおよびフェイルオーバーレルムとして使用するレルムを作成している。

手順

1. 既存のセキュリティーレルムから **failover-realm** を作成します。

構文

```
/subsystem=elytron/failover-realm=<failover_realm_name>:add(delegate-
realm=<realm_to_use_by_default>,failover-realm=<realm_to_use_as_backup>)
```

例

```
/subsystem=elytron/failover-realm=exampleSecurityRealm:add(delegate-
realm=exampleLDAPRealm,failover-realm=exampleFileSystemRealm)
{"outcome" => "success"}
```

2. 属性をロールにマップするロールデコーダーを作成します。

構文

```
/subsystem=elytron/simple-role-decoder=<role_decoder_name>:add(attribute=<attribute>)
```

例

```
/subsystem=elytron/simple-role-decoder=from-roles-attribute:add(attribute=Roles)
{"outcome" => "success"}
```

3. **failover-realm** とロールデコーダーを参照するセキュリティードメインを作成します。

構文

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

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleSecurityRealm,role-decoder="from-roles-attribute"}])
{"outcome" => "success"}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **failover-realm** 属性
- **security-domain** 属性
- **simple-role-decoder** 属性

1.5. JAAS レルムの作成

1.5.1. Elytron の JAAS レルム

Java Authentication and Authorization Service (JAAS) レルム **jaas-realm** は、ユーザーの認証情報の検証とユーザーのロールの割り当てのために **elytron** サブシステムでカスタムログインモジュールを設定するために使用できるセキュリティーレルムです。

jaas-realm を使用して、JBoss EAP 管理インターフェイスとデプロイされたアプリケーションの両方を保護できます。

JAAS レルムは、JAAS 設定ファイルで指定されたログインモジュールを使用する **javax.security.auth.login.LoginContext** を初期化することにより、ユーザーの認証情報を検証します。

ログインモジュールは、**javax.security.auth.login.LoginContext.LoginModule** インターフェイスの実装です。これらの実装を JBoss EAP モジュールとしてサーバーに追加し、JAAS 設定ファイルで指定します。

JAAS 設定ファイルの例

```
test { (1)
    loginmodules.CustomLoginModule1 optional; (2)
    loginmodules.CustomLoginModule2 optional myOption1=true myOption2=exampleOption; (3)
};
```

(1) **jaas-realm** を設定するときに使用するエントリーの名前。

(2) 任意のフラグを持つログインモジュール。JAAS で定義されているすべてのフラグを使用できます。詳細は、Oracle Java SE ドキュメントの JAAS ログイン設定ファイル を参照してください。

(3) 任意のフラグおよびオプションを含むログインモジュール。

ログインモジュールの属性マッピングおよびロールの関連付けへのサブジェクトのプリンシパル

サブジェクト のプリンシパルを利用して、ログインモジュールから取得した ID に属性を追加できます。**サブジェクト** は認証されるユーザーであり、プリンシパルはサブジェクト内に含まれるユーザー名などの識別子です。

Elytron は、次のように ID を取得してマッピングします。

- ログインモジュールは、**javax.security.auth.Subject** を使用して、認証されるユーザー、**サブジェクト** を表します。
- **サブジェクト** には、**java.security.Principal**、principal の複数のインスタンスを関連付けることができます。
- Elytron は、**org.wildfly.security.auth.server.SecurityIdentity** を使用して認証されたユーザーを表します。**SecurityIdentity** の **対象となる** Elytron マップ。

サブジェクトの **プリンシパル** は、以下のルールでセキュリティーアイデンティティーの属性にマッピングされます。

- 属性の **key** は、**principal.getClass().getSimpleName()** 呼び出しによって取得された principal の単純なクラス名です。
- **value** は、**principal.getName()** 呼び出しによって取得された**プリンシパル**の名前です。
- 同じタイプの **プリンシパル** の場合、値は属性キーの下のコレクションに追加されます。

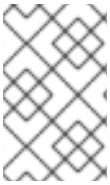
関連情報

- カスタム JAAS ログインモジュールの開発
- Elytron での **jaas-realm** の作成
- **Subject** クラスの Javadocs
- **Principal** クラスの Javadocs

1.5.2. カスタム JAAS ログインモジュールの開発

カスタム Java Authentication and Authorization Service (JAAS) ログインモジュールを作成して、カスタム認証および認可機能を実装できます。

Elytron サブシステムの **jaas-realm** を介してカスタム JAAS ログインモジュールを使用して、JBoss EAP 管理インターフェイスおよびデプロイされたアプリケーションをセキュア化できます。ログインモジュールはデプロイメントの一部ではなく、JBoss EAP モジュールとして含まれます。



注記

以下の手順は例としてのみ提供されています。保護する必要があるアプリケーションがすでにある場合は、以下の手順をスキップして、アプリケーションへの認証と認可の追加 に直接進むことができます。

1.5.2.1. JAAS ログインモジュール開発の Maven プロジェクトの作成

カスタム JavaAuthenticationand Authorization Service (JAAS) ログインモジュールを作成するには、必要な依存関係とディレクトリー構造を使用して Maven プロジェクトを作成します。

前提条件

- Maven がインストールされている。詳細は、Downloading Apache Maven を参照してください。

手順

1. CLI で **mvn** コマンドを使用して Maven プロジェクトを設定します。このコマンドは、プロジェクトのディレクトリー構造と **pom.xml** 設定ファイルを作成します。

   構文

   ```
   $ mvn archetype:generate \
   -DgroupId=<group-to-which-your-application-belongs> \
   -DartifactId=<name-of-your-application> \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-simple \
   -DinteractiveMode=false
   ```

   例

   ```
   $ mvn archetype:generate \
   -DgroupId=com.example.loginmodule \
   -DartifactId=example-custom-login-module \
   -DarchetypeGroupId=org.apache.maven.archetypes \
   -DarchetypeArtifactId=maven-archetype-simple \
   -DinteractiveMode=false
   ```

2. アプリケーションのルートディレクトリーに移動します。

   構文

   ```
   $ cd <name-of-your-application>
   ```

   例

```
$ cd example-custom-login-module
```

3. 生成された **pom.xml** ファイルの内容を、以下のテキストに置き換えます。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>custom.loginmodules</groupId>
    <artifactId>custom-login-modules</artifactId>
    <version>1.0</version>
    <dependencies>
        <dependency>
            <groupId>org.wildfly.security</groupId>
            <artifactId>wildfly-elytron</artifactId>
            <version>1.17.2.Final</version>
        </dependency>
        <dependency>
            <groupId>jakarta.security.enterprise</groupId>
            <artifactId>jakarta.security.enterprise-api</artifactId>
            <version>3.0.0</version>
        </dependency>
    </dependencies>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
    </properties>

</project>
```

4. この例ではディレクトリーは必要ないため、ディレクトリー **site** を削除して **test** します。

```
$ rm -rf src/site/
$ rm -rf src/test/
```

検証

- アプリケーションのルートディレクトリーで、次のコマンドを入力します。

```
$ mvn install
```

次のような出力が得られます。

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.404 s
[INFO] Finished at: 2022-04-28T13:55:18+05:30
[INFO] ------------------------------------------------------------------------
```

カスタム JAAS ログインモジュールを作成できるようになりました。

1.5.2.2. カスタム JAAS ログインモジュールの作成

javax.security.auth.spi.LoginModule インターフェイスを実装するクラスを作成して、カスタム Java Authentication and Authorization Service (JAAS) ログインモジュールを作成します。さらに、カスタムログインモジュールのフラグとオプションを含む JAAS 設定ファイルを作成します。

この手順では、**<application_home>** は、アプリケーションの **pom.xml** 設定ファイルが含まれるディレクトリーを参照します。

前提条件

- Maven プロジェクトを作成している。
  詳細は、JAAS ログインモジュール開発用の Maven プロジェクトの作成 を参照してください。

手順

1. Java ファイルを保存するディレクトリーを作成します。

   構文

   ```
   $ mkdir -p src/main/java/<path_based_on_artifactID>
   ```

   例

   ```
   $ mkdir -p src/main/java/com/example/loginmodule
   ```

2. ソースファイルを含むディレクトリーに移動します。

   構文

   ```
   $ cd src/main/java/<path_based_on_groupID>
   ```

   例

   ```
   $ cd src/main/java/com/example/loginmodule
   ```

3. 生成されたファイル **App.java** を削除します。

   ```
   $ rm App.java
   ```

4. カスタムログインモジュールソースの **ExampleCustomLoginModule.java** ファイルを作成します。

   ```
   package com.example.loginmodule;

   import org.wildfly.security.auth.principal.NamePrincipal;

   import javax.security.auth.Subject;
   import javax.security.auth.callback.Callback;
   import javax.security.auth.callback.CallbackHandler;
   ```

```
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import java.io.IOException;
import java.security.Principal;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;


public class ExampleCustomLoginModule implements LoginModule {

    private final Map<String, char[]> usersMap = new HashMap<String, char[]>();
    private Principal principal;
    private Subject subject;
    private CallbackHandler handler;

    /**
     * In this example, identities are created as fixed Strings.
     *
     * The identities are:
     *    user1 has the password passwordUser1
     *    user2 has the password passwordUser2
     *
     * Use these credentials when you secure management interfaces
     * or applications with this login module.
     *
     * In a production login module, you would get the identities
     * from a data source.
     *
     */

    @Override
    public void initialize(Subject subject, CallbackHandler callbackHandler, Map<String, ?>
sharedState, Map<String, ?> options) {
        this.subject = subject;
        this.handler = callbackHandler;
        this.usersMap.put("user1", "passwordUser1".toCharArray());
        this.usersMap.put("user2", "passwordUser2".toCharArray());
    }

    @Override
    public boolean login() throws LoginException {
        // obtain the incoming username and password from the callback handler
        NameCallback nameCallback = new NameCallback("Username");
        PasswordCallback passwordCallback = new PasswordCallback("Password", false);
        Callback[] callbacks = new Callback[]{nameCallback, passwordCallback};
        try {
            this.handler.handle(callbacks);
        } catch (UnsupportedCallbackException | IOException e) {
            throw new LoginException("Error handling callback: " + e.getMessage());
        }

        final String username = nameCallback.getName();
```

```
        this.principal = new NamePrincipal(username);
        final char[] password = passwordCallback.getPassword();

        char[] storedPassword = this.usersMap.get(username);
        if (!Arrays.equals(storedPassword, password)) {
            throw new LoginException("Invalid password");
        } else {
            return true;
        }
    }

    /**
     * user1 is assigned the roles Admin, User and Guest.
     * In a production login module, you would get the identities
     * from a data source.
     *
     */

    @Override
    public boolean commit() throws LoginException {
        if (this.principal.getName().equals("user1")) {
            this.subject.getPrincipals().add(new Roles("Admin"));
            this.subject.getPrincipals().add(new Roles("User"));
            this.subject.getPrincipals().add(new Roles("Guest"));
        }
        return true;
    }

    @Override
    public boolean abort() throws LoginException {
        return true;
    }

    @Override
    public boolean logout() throws LoginException {
        this.subject.getPrincipals().clear();
        return true;
    }

    /**
     * Principal with simple classname 'Roles' will be mapped to the identity's attribute with
name 'Roles'.
     */

    private static class Roles implements Principal {

        private final String name;

        Roles(final String name) {
            this.name = name;
        }

        /**
         * @return name of the principal. This will be added as a value to the identity's attribute
which has a name equal to the simple name of this class. In this example, this value will be
added to the attribute with a name 'Roles'.
```

```
        */

        public String getName() {
            return this.name;
        }
    }
}
```

5. <application_home> ディレクトリーに、JAAS 設定ファイル **JAAS-login-modules.conf** を作成します。

```
exampleConfiguration {
 com.example.loginmodule.ExampleCustomLoginModule optional;
};
```

- **exampleConfiguration** はエントリー名です。
- **com.example.loginmodule.ExampleCustomLoginModule** はログインモジュールです。
- **optional** はフラグです。

6. ログインモジュールをコンパイルします。

```
$ mvn package
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.321 s
[INFO] Finished at: 2022-04-28T14:16:03+05:30
[INFO] ------------------------------------------------------------------------
```

これで、ログインモジュールを使用して JBoss EAP 管理インターフェイスとデプロイされたアプリケーションを保護できます。

1.5.3. Elytron での jaas-realm の作成

Java Authentication and Authorization Service (JAAS) 互換のカスタムログインモジュールに基づく Elytron セキュリティーレルムを作成して、JBoss EAP サーバーインターフェイスまたはデプロイされたアプリケーションを保護します。セキュリティーレルムを使用して、セキュリティードメインを作成します。

前提条件

- カスタムログインモジュールを JAR としてパッケージ化しました。
  ログインモジュールの例は、カスタム JAAS ログインモジュールの開発 を参照してください。
- JBoss EAP が実行されている。

手順

1. 管理 CLI を使用してログインモジュール JAR をモジュールとして JBoss EAP に追加します。

   構文

   -

```
module add --name=<name_of_the_login_moudle> --
resources=<path_to_the_login_module_jar> --dependencies=org.wildfly.security.elytron
```

例

```
module add --name=exampleLoginModule  --resources=<path_to_login_module>/custom-
login-modules-1.0.jar --dependencies=org.wildfly.security.elytron
```

2. ログインモジュールと JAAS ログイン設定ファイルから **jaas-realm** を作成します。

構文

```
/subsystem=elytron/jaas-realm=<jaas_realm_name>:add(entry=<entry-
name>,path=<path_to_module_config_file>,module=<name_of_the_login_module>,callb
ack-handler=<name_of_the_optional_callback_handler>)
```

例

```
/subsystem=elytron/jaas-
realm=exampleSecurityRealm:add(entry=exampleConfiguration,path=<path_to_login_mod
ule>/JAAS-login-modules.conf,module=exampleLoginModule)
```

3. **jaas-realm** を参照するセキュリティードメインを作成します。

構文

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

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleSecurityRealm,realms=[{realm=exampleSecurityRealm}],permission-
mapper=default-permission-mapper)
{"outcome" => "success"}
```

これで、作成したセキュリティードメインを使用して、管理インターフェイスとアプリケーションに認証と認可を追加できるようになりました。詳細は、管理インターフェイスとアプリケーションの保護 を参照してください。

関連情報

- **module** コマンド引数
- **jaas-realm** 属性
- **security-domain** 属性

第2章 管理インターフェイスとアプリケーションのセキュリティー保護

2.1. 管理インターフェイスへの認証および認可の追加

管理インターフェイスに認証と認可を追加し、セキュリティードメインを使用して管理インターフェイスを保護できます。認証と認可を追加した後に管理インターフェイスにアクセスするには、ユーザーはログイン認証情報を入力する必要があります。

以下のように JBoss EAP 管理インターフェイスを保護できます。

- 管理 CLI
  sasl-authentication-factory の設定による保護。
- 管理コンソール
  http-authentication-factory の設定による保護。

前提条件

- セキュリティーレルムを参照するセキュリティードメインを作成しました。
- JBoss EAP が実行されている。

手順

1. **http-authentication-factory** または **sasl-authentication-factory** を作成します。
   - **http-authentication-factory** を作成します。

     構文

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

     例

     ```
     /subsystem=elytron/http-authentication-factory=exampleAuthenticationFactory:add(http-server-mechanism-factory=global, security-domain=exampleSecurityDomain, mechanism-configurations=[{mechanism-name=BASIC, mechanism-realm-configurations=[{realm-name=exampleSecurityRealm}]}])
     {"outcome" => "success"}
     ```

   - **sasl-authentication-factory** を作成します。

     構文

     ```
     /subsystem=elytron/sasl-authentication-factory=<sasl_authentication_factory_name>:add(security-domain=<security_domain>,sasl-server-factory=configured,mechanism-configurations=
     ```

```
[{mechanism-name=<mechanism-name>,mechanism-realm-configurations=[{realm-name=<realm_name>}]}])
```

例

```
/subsystem=elytron/sasl-authentication-factory=exampleSaslAuthenticationFactory:add(security-domain=exampleSecurityDomain,sasl-server-factory=configured,mechanism-configurations=[{mechanism-name=PLAIN,mechanism-realm-configurations=[{realm-name=exampleSecurityRealm}]}])
{"outcome" => "success"}
```

2. 管理インターフェイスを更新します。

- **http-authentication-factory** を使用して管理コンソールを保護します。

  構文

  ```
  /core-service=management/management-interface=http-interface:write-attribute(name=http-authentication-factory, value=<authentication_factory_name>)
  ```

  例

  ```
  /core-service=management/management-interface=http-interface:write-attribute(name=http-authentication-factory, value=exampleAuthenticationFactory)
  {
      "outcome" => "success",
      "response-headers" => {
          "operation-requires-reload" => true,
          "process-state" => "reload-required"
      }
  }
  ```

- **sasl-authentication-factory** を使用して管理 CLI を保護します。

  構文

  ```
  /core-service=management/management-interface=http-interface:write-attribute(name=http-upgrade,value={enabled=true,sasl-authentication-factory=<sasl_authentication_factory>})
  ```

  例

  ```
  /core-service=management/management-interface=http-interface:write-attribute(name=http-upgrade,value={enabled=true,sasl-authentication-factory=exampleSaslAuthenticationFactory})
  {
      "outcome" => "success",
      "response-headers" => {
          "operation-requires-reload" => true,
          "process-state" => "reload-required"
      }
  }
  ```

3. サーバーをリロードします。

```
reload
```

検証

- 管理コンソールに認証と認可が必要であることを確認するには、**http://127.0.0.1:9990/console/index.html** から管理コンソールに移動します。ユーザー名とパスワードの入力を求められます。
- 管理 CLI に認証と認可が必要であることを確認するには、次のコマンドを使用して管理 CLI を起動します。

```
$ bin/jboss-cli.sh --connect
```

ユーザー名とパスワードの入力を求められます。

関連情報

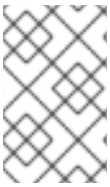
- **http-authentication-factory** 属性
- **sasl-authentication-factory** 属性

2.2. セキュリティードメインを使用したアプリケーションユーザーの認証と認可

セキュリティーレルムを参照するセキュリティードメインを使用して、アプリケーションユーザーを認証および認可します。アプリケーションを開発するための手順は、例としてのみ提供されています。

2.2.1. 集約レルム用の単純な Web アプリケーションの開発

簡単な Web アプリケーションを作成して、セキュリティーレルムの設定例に従うことができます。



注記

以下の手順は例としてのみ提供されています。保護する必要があるアプリケーションがすでにある場合は、以下の手順をスキップして、アプリケーションへの認証と認可の追加 に直接進むことができます。

2.2.1.1. web アプリケーション開発用の maven プロジェクトの作成

web-application を作成するには、必要な依存関係とディレクトリー構造で Maven プロジェクトを作成します。

前提条件

- Maven がインストールされている。詳細は、Downloading Apache Maven を参照してください。

手順

1. **mvn** コマンドを使用して Maven プロジェクトを設定します。このコマンドは、プロジェクトのディレクトリー構造と **pom.xml** 設定ファイルを作成します。

構文

```
$ mvn archetype:generate \
-DgroupId=${group-to-which-your-application-belongs} \
-DartifactId=${name-of-your-application} \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

例

```
$ mvn archetype:generate \
-DgroupId=com.example.app \
-DartifactId=simple-webapp-example \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DarchetypeArtifactId=maven-archetype-webapp \
-DinteractiveMode=false
```

2. アプリケーションのルートディレクトリーに移動します。

構文

```
$ cd <name-of-your-application>
```

例

```
$ cd simple-webapp-example
```

3. 生成された **pom.xml** ファイルの内容を、以下のテキストに置き換えます。

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

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.app</groupId>
  <artifactId>simple-webapp-example</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>simple-webapp-example Maven Webapp</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <dependencies>
```

```
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.0.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.wildfly.security</groupId>
      <artifactId>wildfly-elytron-auth-server</artifactId>
      <version>1.19.0.Final</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>${project.artifactId}</finalName>
    <plugins>
      <plugin>
        <groupId>org.wildfly.plugins</groupId>
        <artifactId>wildfly-maven-plugin</artifactId>
        <version>2.1.0.Final</version>
      </plugin>
    </plugins>
  </build>

</project>
```

検証

- アプリケーションのルートディレクトリーで、次のコマンドを入力します。

```
$ mvn install
```

次のような出力が得られます。

```
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.795 s
[INFO] Finished at: 2022-04-28T17:39:48+05:30
[INFO] ------------------------------------------------------------------------
```

web-application を作成できるようになりました。

2.2.1.2. Web アプリケーションの作成

ログインユーザーのプリンシパルおよび属性から取得したユーザー名を返すサーブレットを含む Web アプリケーションを作成します。ログインしているユーザーがないと、サーブレットは「NO AUTHENTICATED USER」のテキストを返します。

前提条件

- Maven プロジェクトを作成している。

- JBoss EAP が実行されている。

手順

1. Java ファイルを保存するディレクトリーを作成します。

   構文

   ```
   $ mkdir -p src/main/java/<path_based_on_artifactID>
   ```

   例

   ```
   $ mkdir -p src/main/java/com/example/app
   ```

2. 新しいディレクトリーに移動します。

   構文

   ```
   $ cd src/main/java/<path_based_on_artifactID>
   ```

   例

   ```
   $ cd src/main/java/com/example/app
   ```

3. 以下の内容で **SecuredServlet.java** ファイルを作成します。

   ```
   package com.example.app;

   import java.io.IOException;
   import java.io.PrintWriter;
   import java.security.Principal;
   import java.util.ArrayList;
   import java.util.Collection;
   import java.util.Iterator;
   import java.util.List;
   import java.util.Set;

   import jakarta.servlet.ServletException;
   import jakarta.servlet.annotation.WebServlet;
   import jakarta.servlet.http.HttpServlet;
   import jakarta.servlet.http.HttpServletRequest;
   import jakarta.servlet.http.HttpServletResponse;
   import org.wildfly.security.auth.server.SecurityDomain;
   import org.wildfly.security.auth.server.SecurityIdentity;
   import org.wildfly.security.authz.Attributes;
   import org.wildfly.security.authz.Attributes.Entry;
   /**
    * A simple secured HTTP servlet. It returns the user name and
    * attributes obtained from the logged-in user's Principal. If
    * there is no logged-in user, it returns the text
    * "NO AUTHENTICATED USER".
    */
   ```

```
@WebServlet("/secured")
public class SecuredServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {

         Principal user = req.getUserPrincipal();
         SecurityIdentity identity = SecurityDomain.getCurrent().getCurrentSecurityIdentity();
         Attributes identityAttributes = identity.getAttributes();
         Set <String> keys = identityAttributes.keySet();
         String attributes = "<ul>";

         for (String attr : keys) {
          attributes += "<li> " +  attr + " : " + identityAttributes.get(attr).toString() + "</li>";
         }

         attributes+="</ul>";
         writer.println("<html>");
         writer.println("  <head><title>Secured Servlet</title></head>");
         writer.println("  <body>");
         writer.println("    <h1>Secured Servlet</h1>");
         writer.println("    <p>");
         writer.print(" Current Principal '");
         writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
         writer.print("'");
         writer.print(user != null ? "\n" + attributes : "");
         writer.println("    </p>");
         writer.println("  </body>");
         writer.println("</html>");
        }
    }

}
```

4. アプリケーションのルートディレクトリーで、次のコマンドを使用してアプリケーションをコンパイルします。

```
$ mvn package
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.015 s
[INFO] Finished at: 2022-04-28T17:48:53+05:30
[INFO] ------------------------------------------------------------------------
```

5. アプリケーションをデプロイします。

```
$ mvn wildfly:deploy
```

検証

- ブラウザーで **http://localhost:8080/simple-webapp-example/secured** に移動します。

以下のメッセージが表示されます。

```
Secured Servlet
Current Principal 'NO AUTHENTICATED USER'
```

認証メカニズムが追加されないため、アプリケーションにアクセスできます。

これで、セキュリティードメインを使用してこのアプリケーションを保護し、認証されたユーザーのみがアクセスできるようになります。

2.2.2. アプリケーションへの認証と認可の追加

Web アプリケーションに認証と認可を追加して、セキュリティードメインを使用してアプリケーションを保護できます。認証と認可を追加した後に Web アプリケーションにアクセスするには、ユーザーはログイン認証情報を入力する必要があります。

前提条件

- セキュリティーレルムを参照するセキュリティードメインを作成しました。
- JBoss EAP にアプリケーションをデプロイしました。
- JBoss EAP が実行されている。

手順

1. **undertow subsystem** で **application-security-domain** を設定します。

   構文

   ```
   /subsystem=undertow/application-security-domain=<application_security_domain_name>:add(security-domain=<security_domain_name>)
   ```

   例

   ```
   /subsystem=undertow/application-security-domain=exampleApplicationSecurityDomain:add(security-domain=exampleSecurityDomain)
   {"outcome" => "success"}
   ```

2. アプリケーションの **web.xml** を設定して、アプリケーションリソースを保護します。

   構文

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

   <web-app>

    <!-- Define the security constraints for the application resources.
        Specify the URL pattern for which a challenge is -->

    <security-constraint>
   ```

```
        <web-resource-collection>
            <web-resource-name><!-- Name of the resources to protect --></web-resource-name>
            <url-pattern> <!-- The URL to protect  --></url-pattern>
        </web-resource-collection>

        <!-- Define the role that can access the protected resource -->
        <auth-constraint>
            <role-name> <!-- Role name as defined in the security domain --></role-name>
            <!-- To disable authentication you can use the wildcard *
              To authenticate but allow any role, use the wildcard **. -->
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>
         <!-- The authentication method to use. Can be:
          BASIC
          CLIENT-CERT
          DIGEST
          FORM
          SPNEGO
          -->
        </auth-method>

        <realm-name><!-- The name of realm to send in the challenge  --></realm-name>
    </login-config>
</web-app>
```

例

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

<web-app>

 <!-- Define the security constraints for the application resources.
     Specify the URL pattern for which a challenge is -->

 <security-constraint>
        <web-resource-collection>
            <web-resource-name>all</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>

        <!-- Define the role that can access the protected resource -->
        <auth-constraint>
            <role-name>Admin</role-name>
            <!-- To disable authentication you can use the wildcard *
              To authenticate but allow any role, use the wildcard **. -->
        </auth-constraint>
    </security-constraint>

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

```
        <realm-name>exampleSecurityRealm</realm-name>
    </login-config>
</web-app>
```

注記

別の **auth-method** を使用できます。

3. アプリケーションで **jboss-web.xml** ファイルを作成するか、**undertow** サブシステムでデフォルトのセキュリティードメインを設定することにより、セキュリティードメインを使用するようにアプリケーションを設定します。

   - **application-security-domain** を参照するアプリケーションの **WEB-INF** ディレクトリーに **jboss-web.xml** ファイルを作成します。

     構文

     ```
     <jboss-web>
       <security-domain> <!-- The security domain to associate with the application --
     ></security-domain>
     </jboss-web>
     ```

     例

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

   - アプリケーションの **undertow** サブシステムでデフォルトのセキュリティードメインを設定します。

     構文

     ```
     /subsystem=undertow:write-attribute(name=default-security-
     domain,value=<application_security_domain_to_use>)
     ```

     例

     ```
     /subsystem=undertow:write-attribute(name=default-security-
     domain,value=exampleApplicationSecurityDomain)
     {
         "outcome" => "success",
         "response-headers" => {
             "operation-requires-reload" => true,
             "process-state" => "reload-required"
         }
     }
     ```

4. サーバーをリロードします。

   ```
   reload
   ```

検証

1. アプリケーションのルートディレクトリーで、次のコマンドを使用してアプリケーションをコンパイルします。

```
$ mvn package
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.015 s
[INFO] Finished at: 2022-04-28T17:48:53+05:30
[INFO] ------------------------------------------------------------------------
```

2. アプリケーションをデプロイします。

```
$ mvn wildfly:deploy
```

3. ブラウザーで **http://localhost:8080/simple-webapp-example/secured** に移動します。アプリケーションにアクセスするには認証が必要であることを確認するログインプロンプトが表示されます。

これで、アプリケーションはセキュリティードメインで保護され、ユーザーは認証後にのみログインできます。さらに、指定されたロールを持つユーザーのみがアプリケーションにアクセスできます。

第3章 ELYTRON の監査ロギングの設定

Elytron を使用し、トリガーとなるイベントのセキュリティー監査を完了することができます。セキュリティー監査とは、認可または認証の試行に応答して、ログへの書き込みなどのイベントをトリガーすることを指します。

イベントに対して行われるセキュリティー監査の種類は、セキュリティーレルムの設定によって異なります。

3.1. ELYTRON 監査ロギング

elytron サブシステムで監査ロギングを有効にすると、アプリケーションサーバー内の Elytron の認証および認可イベントをログに記録できるようになります。Elytron は、監査ログエントリーを **JSON** または **SIMPLE** 形式で保存します。人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。

Elytron の監査ロギングは、JBoss EAP 管理インターフェイスの監査ロギングなど、他のタイプの監査ロギングとは異なります。

Elytron の監査ロギングはデフォルトでは無効ですが、次のログハンドラーのいずれかを設定することで監査ロギングを有効にできます。ログハンドラーをセキュリティードメインに追加することもできます。

- ファイル監査ロギング
  詳細は、Elytron のファイル監査ロギングの有効化 を参照してください。
- 定期的なローテーションファイル監査ロギング
  詳細は、Elytron の定期ローテーションファイル監査ロギングの有効化 を参照してください。
- サイズローテーションファイル監査ロギング
  詳細は、Elytron のサイズローテーションファイル監査ロギングの有効化 を参照してください。
- **syslog** 監査ロギング
  詳細は、Elytron の syslog 監査ロギングの有効化 を参照してください。
- カスタム監査ロギング
  詳細は、Elytron でのカスタムセキュリティーイベントリスナーの使用 を参照してください。

aggregate-security-event-listener リソース を使用すると、ロガーなどのより多くの宛先にセキュリティーイベントを送信することができます。**aggregate-security-event-listener リソース** は、全イベントをアグリゲートリスナー定義で指定されたリスナーすべてに配信します。

3.2. ELYTRON のファイル監査ロギングの有効化

ファイル監査ロギングは、監査ログメッセージをファイルシステムにあるファイル 1 つに保存します。

デフォルトでは、Elytron はファイル監査ロガーとして **local-audit** を指定します。

local-audit を有効にして、スタンドアロンサーバーでは **EAP_HOME/standalone/log/audit.log** に、マネージドドメインでは **EAP_HOME/domain/log/audit.log** に、Elytron 監査ログを書き込めるようにする必要があります。

前提条件

- アプリケーションをセキュリティー保護している。
  詳細は、Elytron での **aggregate-realm** の作成 を参照してください。

手順

1. ファイルの監査ログを作成します。

   構文

   ```
   /subsystem=elytron/file-audit-log=<audit_log_name>:add(path="<path_to_log_file>",format=<format_type>,synchronized=<whether_to_log_immediately>)
   ```

   例

   ```
   /subsystem=elytron/file-audit-log=exampleFileAuditLog:add(path="file-audit.log",relative-to=jboss.server.log.dir,format=SIMPLE,synchronized=true)
   ```

2. ファイル監査ログをセキュリティードメインに追加します。

   構文

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=security-event-listener,value=<audit_log_name>)
   ```

   例

   ```
   /subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-event-listener,value=exampleFileAuditLog)
   ```

検証

1. ブラウザーで、セキュリティー保護されたアプリケーションにログインします。
   たとえば、セキュリティードメインを使用したアプリケーションユーザーの認証と認可 で作成したアプリケーションにログインするには、http://localhost:8080/simple-webapp-example/secured に移動してログインします。

2. 監査ログを保存するように設定されたディレクトリーに移動します。この手順のコマンド例を使用する場合、ディレクトリーは **EAP_HOME**/standalone/log です。
   file-audit.log というファイルが作成されることに注意してください。これには、アプリケーションへのログインによってトリガーされたイベントのログが含まれています。

   file-audit.log ファイルの例

   ```
   2023-10-24 23:31:04,WARNING,{event=SecurityPermissionCheckSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true,permission=[type=org.wildfly.security.auth.permission.LoginPermission,actions=,name=]}
   2023-10-24 23:31:04,WARNING,{event=SecurityAuthenticationSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true}
   ```

関連情報

- **file-audit-log** 属性

3.3. ELYTRON の定期ローテーションファイル監査ロギングの有効化

elytron サブシステムを使用して、スタンドアロンサーバーまたはマネージドドメインとして実行されているサーバーの定期ローテーションファイル監査ロギングを有効にできます。

定期ローテーションファイル監査ロギングでは、設定されたスケジュールに基づいて監査ログファイルを自動的にローテーションします。定期ローテーションファイル監査ロギングは、デフォルトのファイル監査ロギングに似ていますが、定期ローテーションファイル監査ロギングには、**suffix** という追加の属性が含まれています。

suffix 属性の値は、**java.time.format.Date Time Formatter** 形式で指定された日付 **(.yyyy-MM-dd** など) です。Elytron は、その接尾辞で指定された値から自動的にローテーションの周期を計算します。**elytron** サブシステムは、ログファイル名の最後に接尾辞を付加します。

前提条件

- アプリケーションをセキュリティー保護している。
  詳細は、Elytron での **aggregate-realm** の作成 を参照してください。

手順

1. 定期的なローテーションファイル監査ログを作成します。

   構文

   ```
   /subsystem=elytron/periodic-rotating-file-audit-log=<periodic_audit_log_name>:add(path="<periodic_audit_log_filename>",format=<record_format>,synchronized=<whether_to_log_immediately>,suffix="<suffix_in_DateTimeFormatter_format>")
   ```

   例

   ```
   /subsystem=elytron/periodic-rotating-file-audit-log=examplePreiodicFileAuditLog:add(path="periodic-file-audit.log",relative-to=jboss.server.log.dir,format=SIMPLE,synchronized=true,suffix="yyyy-MM-dd")
   ```

2. 定期的なローテーションファイル監査ロガーをセキュリティードメインに追加します。

   構文

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=security-event-listener,value=<periodic_audit_log_name>)
   ```

   例

   ```
   /subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-event-listener,value=examplePreiodicFileAuditLog)
   ```

検証

1. ブラウザーで、セキュリティー保護されたアプリケーションにログインします。
たとえば、セキュリティードメインを使用したアプリケーションユーザーの認証と認可 で作成したアプリケーションにログインするには、http://localhost:8080/simple-webapp-example/secured に移動してログインします。

2. 監査ログを保存するように設定されたディレクトリーに移動します。この手順のコマンド例を使用する場合、ディレクトリーは **EAP_HOME**/standalone/log です。
periodic-file-audit.log というファイルが作成されることに注意してください。これには、アプリケーションへのログインによってトリガーされたイベントのログが含まれています。

periodic-file-audit.log ファイルの例

```
2023-10-24 23:31:04,WARNING,{event=SecurityPermissionCheckSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true,permission=[type=org.wildfly.security.auth.permission.LoginPermission,actions=,name=]}
2023-10-24 23:31:04,WARNING,{event=SecurityAuthenticationSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true}
```

関連情報

- **periodic-rotating-file-audit-log** 属性

3.4. ELYTRON のサイズローテーションファイル監査ロギングの有効化

elytron サブシステムを使用して、スタンドアロンサーバーまたはマネージドドメインとして実行されているサーバーのサイズローテーションファイル監査ロギングを有効にできます。

サイズローテーションファイル監査ロギングでは、ログファイルが設定されたファイルサイズに達すると、監査ログファイルが自動的にローテーションされます。サイズローテーションファイル監査ロギングは、デフォルトのファイル監査ロギングに似ていますが、サイズローテーションファイル監査ロギングには、追加の属性が含まれています。

ログファイルのサイズが **rotate-size** 属性で定義された制限を超えると、Elytron は現在のファイルの末尾に接尾辞 **.1** を追加し、新しいログファイルを作成します。Elytron は既存のログファイルの接尾辞を 1 ずつ増やします。たとえば、Elytron は **audit_log.1** の名前を **audit_log.2** に変更します。Elytron は、ログファイルの数が **max-backup-index** で定義されたログファイルの最大数に達するまで数を増やし続けます。ログファイルが **max-backup-index** 値を超えると、Elytron はファイルを削除します。たとえば、**max-backup-index の値として "98" が定義されている場合、audit_log.99 ファイルはこの制限を超えることになります。**

前提条件

- **アプリケーションをセキュリティー保護している。**

  詳細は、Elytron での aggregate-realm の作成 を参照してください。

手順

1. **サイズローテーションファイルの監査ログを作成します。**

構文

```
/subsystem=elytron/size-rotating-file-audit-log=<audit_log_name>:add(path="
<path_to_log_file>",format=<record_format>,synchronized=<whether_to_log_immediately
>,rotate-size="<max_file_size_before_rotation>",max-backup-
index=<max_number_of_backup_files>)
```

例

```
/subsystem=elytron/size-rotating-file-audit-log=exampleSizeFileAuditLog:add(path="size-file-
audit.log",relative-to=jboss.server.log.dir,format=SIMPLE,synchronized=true,rotate-
size="10m",max-backup-index=10)
```

2. **サイズローテーション監査ロガーをセキュリティードメインに追加します。**

構文

```
/subsystem=elytron/security-domain=<domain_size_logger>:write-attribute(name=security-
event-listener,value=<audit_log_name>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-
event-listener,value=exampleSizeFileAuditLog)
```

検証

1. ブラウザーで、セキュリティー保護されたアプリケーションにログインします。

   たとえば、セキュリティードメインを使用したアプリケーションユーザーの認証と認可 で作成したアプリケーションにログインするには、http://localhost:8080/simple-webapp-example/secured に移動してログインします。

2. 監査ログを保存するように設定されたディレクトリーに移動します。この手順のコマンド例を使用する場合、ディレクトリーは EAP_HOME/standalone/log です。

   size-file-audit.log というファイルが作成されることに注意してください。これには、アプリケーションへのログインによってトリガーされたイベントのログが含まれています。

   size-file-audit.log ファイルの例

```
2023-10-24 23:31:04,WARNING,{event=SecurityPermissionCheckSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true,permission=[type=org.wildfly.security.auth.permission.LoginPermission,actions=,name=]}
2023-10-24 23:31:04,WARNING,{event=SecurityAuthenticationSuccessfulEvent,event-time=2023-10-24 23:31:04,security-identity=[name=user1,creation-time=2023-10-24 23:31:04],success=true}
```

関連情報

- size-rotating-file-audit-log 属性

3.5. ELYTRON の SYSLOG 監査ロギングの有効化

elytron サブシステムを使用して、スタンドアロンサーバーまたはマネージドドメインとして実行されているサーバーの syslog 監査ロギングを有効にできます。syslog 監査ロギングを使用すると、ロギング結果を syslog サーバーに送信するため、ローカルファイルへのロギングよりもセキュリティーオプションが多くなります。

syslog ハンドラーは、syslog サーバーのホスト名や syslog サーバーがリッスンするポートなど、syslog サーバーへの接続に使用するパラメーターを指定します。複数の syslog ハンドラーを定義し、それらを同時にアクティブにすることができます。

対応するログ形式は、RFC5424 と RFC3164 です。伝送プロトコルは、UDP、TCP、TCP with SSL に対応しています。

最初のインスタンスの syslog を定義すると、次の例に示すメッセージを含む INFORMATIONAL 優先度イベントが、ロガーによって syslog サーバーに送信されます。

```
"Elytron audit logging enabled with RFC format: <format>"
```

<format> は監査ロギングハンドラー用に設定された Request for Comments (RFC) 形式を示します。デフォルトは RFC5424 です。

前提条件

- アプリケーションをセキュリティー保護している。

  詳細は、Elytron での aggregate-realm の作成 を参照してください。

手順

1. syslog ハンドラーを追加します。

   構文

   ```
   /subsystem=elytron/syslog-audit-log=<syslog_audit_log_name>:add(host-
   name=<record_host_name>,port=<syslog_server_port_number>,server-
   address=<syslog_server_address>,format=<record_format>,
   transport=<transport_layer_protocol>)
   ```

   また、TLS 経由で syslog サーバーにログを送信することもできます。

TLS 経由でログを送信するための syslog 設定の構文

```
/subsystem=elytron/syslog-audit-log=<syslog_audit_log_name>:add(transport=SSL_TCP,server-address=<syslog_server_address>,port=<syslog_server_port_number>,host-name=<record_host_name>,ssl-context=<client_ssl_context>)
```

2. **セキュリティードメインに syslog 監査ロガーを追加します。**

構文

```
/subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=security-event-listener,value=<syslog_audit_log_name>)
```

例

```
/subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-event-listener,value=exampleSyslog)
```

関連情報

- **syslog-audit-log 属性**
- **client-ssl-context の使用**
- **The Syslog Protocol**

- The BSD syslog Protocol

3.6. ELYTRON でのカスタムセキュリティーイベントリスナーの使用

Elytron を使用して、カスタムイベントリスナーを定義できます。カスタムイベントリスナーは、受信したセキュリティーイベントを処理します。イベントリスナーは、カスタム監査ロギングに使用することも、内部 ID ストレージに対してユーザーを認証するために使用することもできます。

重要

module 管理 CLI コマンドを使用してモジュールを追加および削除する機能は、テクノロジープレビュー機能としてのみ提供されています。module コマンドは、マネージドドメインで使用する場合や、リモート管理 CLI を使用して接続する場合には適していません。実稼働環境ではモジュールを手動で追加または削除する必要があります。

テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、Red Hat では実稼働環境での使用を推奨していません。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。

前提条件

- アプリケーションをセキュリティー保護している。

  詳細は、Elytron での aggregate-realm の作成 を参照してください。

手順

1. java.util.function.Consumer<org.wildfly.security.auth.server.event.SecurityEvent> インターフェイスを実装するクラスを作成します。

   指定のインターフェイスを使用する Java クラスの作成例

```
public class MySecurityEventListener implements Consumer<SecurityEvent> {
```

```
    public void accept(SecurityEvent securityEvent) {
        if (securityEvent instanceof SecurityAuthenticationSuccessfulEvent) {
            System.err.printf("Authenticated user \"%s\"\n",
securityEvent.getSecurityIdentity().getPrincipal());
        } else if (securityEvent instanceof SecurityAuthenticationFailedEvent) {
            System.err.printf("Failed authentication as user \"%s\"\n",
((SecurityAuthenticationFailedEvent)securityEvent).getPrincipal());
        }
    }
}
```

この例の Java クラスは、ユーザー認証に成功または失敗するたびに、メッセージを表示します。

2. **カスタムイベントリスナーをモジュールとして提供する JAR ファイルを、JBoss EAP に追加します。**

以下に、カスタムイベントリスナーをモジュールとして Elytron に追加する管理 CLI コマンドの例を示します。

module コマンドを使用して、カスタムイベントリスナーを Elytron にモジュールとして追加した例

```
/subsystem=elytron/custom-security-event-
listener=<listener_name>:add(module=<module_name>, class-name=<class_name>)
```

3. **セキュリティードメインのカスタムイベントリスナーを参照します。**

Application Domain でカスタムイベントリスナーを参照する例

```
/subsystem=elytron/security-domain=<domain_name>:write-attribute(name=security-event-
listener, value=<listener_name>)
```

4. サービスを再起動します。

```
$ reload
```

イベントリスナーは、指定されたセキュリティードメインからセキュリティーイベントを受信します。

関連情報

- カスタムモジュールの手動作成
- 手作業によるカスタムモジュールの削除
- カスタムコンポーネントの Elytron への追加

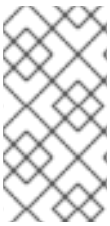
第4章 参照

4.1. AGGREGATE-REALM 属性

属性を設定することで、aggregate-realm を設定できます。

表4.1 aggregate-realm 属性

属性	説明
authentication-realm	認証手順に使用するセキュリティーレルムへの参照。これは、認証情報の取得または検証に使用されます。
authorization-realm	認可手順のアイデンティティーの読み込みに使用するセキュリティーレルムへの参照。
authorization-realms	認可手順のアイデンティティーをロードするために集約するセキュリティーレルムへの参照。属性が複数の認可レルムで定義されている場合は、最初に出現した属性の値が使用されます。
principal-transformer	認証用のアイデンティティーのロードと認可用のアイデンティティーのロードの間に適用するプリンシパルトランスフォーマーへの参照。



注記

authorization-realm と authorization-realms 属性は互いに排他的なものです。レルムでは、2 つの属性いずれかのみを定義します。

4.2. CACHING-REALM 属性

属性を設定することで caching-realm を設定できます。

表4.2 caching-realm 属性

属性	説明
maximum-age	項目がキャッシュ内に留まることができる時間 (ミリ秒単位)。**-1** の値では、無制限で項目を維持します。デフォルトは **-1** です。
maximum-entries	キャッシュに保持するエントリーの最大数。デフォルトは **16** です。

属性	説明
realm	**jdbc-realm**、**ldap-realm**、**filesystem-realm** またはカスタマーセキュリティーレリムなどのキャッシュ可能なセキュリティーレリムへの参照。

4.3. DISTRIBUTED-REALM 属性

属性を設定することで、distributed-realm を設定できます。

表4.3 distributed-realm 属性

属性	説明
emit-events	レルムが利用できないことを示す **SecurityEvent** を発行するかどうか。**ignore-unavailable-realms** 属性が **true** に設定されている場合にのみ適用されます。デフォルト値は **true** です。
ignore-unavailable-realms	アイデンティティーストアへの接続が失敗した場合に、後続のレルムをチェックする必要があるかどうか。後続のレルムをチェックするには、値を **true** に設定します。デフォルト値は **false** です。 値が **true** に設定されている場合、デフォルトでは、アイデンティティーストアへの接続が失敗すると **SecurityEvent** が発行されます。
realms	検索するセキュリティーレルムのリスト。セキュリティーレルムは、この属性で指定された順序で順番に呼び出されます。

4.4. FAILOVER-REALM 属性

属性を設定することにより、failover-realm を設定できます。

表4.4 failover-realm 属性

属性	説明
delegate-realm	デフォルトで使用するセキュリティーレルム。
emit-events	**delegate-realm** が使用できないことを示すタイプ **SecurityEvent** のセキュリティーイベントを出力するかどうかを指定します。有効にすると、これらのイベントを監査ログにキャプチャーすることができます。デフォルト値は **true** です。

属性	説明
failover-realm	**delegate-realm** が使用できない場合に使用するセキュリティーレルム。

4.5. FILE-AUDIT-LOG 属性

表4.5 file-audit-log 属性

属性	説明
autoflush	監査イベントが発生するたびに、出力ストリームのフラッシュが必要かどうかを指定します。属性を定義しない場合、**synchronized** 属性値がデフォルトになります。
encoding	監査ファイルのエンコーディングを指定します。デフォルトは **UTF-8** です。指定可能な値は次のとおりです。 • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**
format	デフォルト値は **SIMPLE** です。人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。
path	ログファイルの保存場所を定義します。
relative-to	任意の属性です。ログファイルの保存場所を定義します。
synchronized	デフォルト値は **true** です。監査イベントが発生するたびにファイル記述子が同期されることを指定します。

4.6. HTTP-AUTHENTICATION-FACTORY 属性

http-authentication-factory を設定するには、属性を設定します。

表4.6 http-authentication-factory 属性

属性	説明
http-server-mechanism-factory	このリソースに関連付ける **HttpServerAuthenticationMechanismFactory**。
mechanism-configurations	メカニズム固有の設定のリスト。
security-domain	リソースに関連付けるセキュリティードメイン。

表4.7 http-authentication-factory mechanism-configurations 属性

属性	説明
credential-security-factory	メカニズムで必要な認証情報の取得に使用するセキュリティーファクトリー。
final-principal-transformer	このメカニズムレルムに適用する最終のプリンシパルトランスフォーマー。
host-name	この設定が適用されるホスト名。
mechanism-name	この設定は、名前を指定したメカニズムが使用される場合にのみ適用されます。この属性を省略すると、メカニズム名に一致します。
mechanism-realm-configurations	メカニズムが理解するレルム名の定義のリストです。
pre-realm-principal-transformer	レルムが選択される前に適用するプリンシパルトランスフォーマー。
post-realm-principal-transformer	レルムの選択後に適用するプリンシパルトランスフォーマー。
protocol	この設定が適用されるプロトコル。
realm-mapper	メカニズムによって使用されるレルムマッパー。

表4.8 http-authentication-factory mechanism-configurations mechanism-realm-configurations 属性

属性	説明
final-principal-transformer	このメカニズムレルムに適用する最終のプリンシパルトランスフォーマー。

属性	説明
post-realm-principal-transformer	レルムの選択後に適用するプリンシパルトランスフォーマー。
pre-realm-principal-transformer	レルムが選択される前に適用するプリンシパルトランスフォーマー。
realm-mapper	メカニズムによって使用されるレルムマッパー。
realm-name	メカニズムにより提示されるレルムの名前。

4.7. JAAS-REALM 属性

属性を設定して jaas-realm を設定できます。entry 以外の属性はすべて任意です。

表4.9 jaas-realm 属性

属性	説明
callback-handler	ログインコンテキストで使用するコールバックハンドラー。代わりに、セキュリティープロパティー **auth.login.defaultCallbackHandler** を使用できます。これらのいずれも定義されていない場合は、レルムのデフォルトのコールバックハンドラーが使用されます。
entry	**LoginContext** の初期化に使用するエントリー名。
module	カスタム **LoginModules** およびd **CallbackHandler** クラスを持つモジュール。
path	JAAS 設定ファイルへのオプションのパス。java システムプロパティー **java.security.auth.login.config** または java セキュリティープロパティー **login.config.url** を使用して場所を指定することもできます。
relative-to	**relative-to** が指定されている場合、**path** 属性の値は relative-to 属性によって指定されるファイルパスの相対値となります。

4.8. MODULE コマンド引数

module コマンドで異なる引数を使用できます。

表4.10 モジュールコマンド引数

引数	説明
--absolute-resources	この引数を使用して、**module.xml** ファイルから参照するファイルシステムの絶対パスのリストを指定します。指定されるファイルはモジュールディレクトリーにはコピーされません。 区切り文字の詳細は **--resource-delimiter** を参照してください。
--allow-nonexistent-resources	この引数を使用して、**--resources** によって指定された存在しないリソースの空のディレクトリーを作成します。存在しないリソースがある場合、この引数を使用しないと **module add** コマンドの実行に失敗します。
--dependencies	この引数を使用して、このモジュールが依存するモジュール名のコンマ区切りリストを提供します。
--export-dependencies	この引数を使用して、エクスポートされた依存関係を指定します。 module add --name=com.mysql --resources=/**path**/**to**/{MySQLDriverJarName} --export-dependencies=wildflyee.api,java.se
--main-class	この引数を使用して、モジュールのメインメソッドを宣言する完全修飾クラス名を指定します。
--module-root-dir	デフォルトの **EAP_HOME**/**modules**/ ディレクトリーの代わりに、外部の JBoss EAP モジュールディレクトリーを定義した場合は、この引数を使用します。 module add --module-root-dir=/**path**/**to**/my-external-modules/ --name=com.mysql --resources=/**path**/**to**/{MySQLDriverJarName} --dependencies=wildflyee.api,java.se
--module-xml	この引数を使用して、この新規モジュールに使用する **module.xml** へのファイルシステムパスを提供します。ます。この引数を指定しないと、**module.xml** ファイルがモジュールディレクトリーに生成されます。
--name	この引数を使用して、追加するモジュールの名前を提供します。この引数は必須です。
--properties	この引数を使用して、モジュールプロパティーを定義する **PROPERTY_NAME=PROPERTY_VALUE** ペアのコンマ区切りリストを提供します。

引数	説明
--resource-delimiter	この引数を使用して、**--resources** または **absolute-resources** 引数に提供されたリソースリストの、ユーザー定義のファイルパス区切り文字を設定します。設定されていない場合、ファイルパスの区切り文字は Linux ではコロン (**:**) 、Windows ではセミコロン (**;**) になります。
--resources	この引数を使用して、ファイルシステムパスのリストを提供してこのモジュールのリソースを指定します。ファイルは直接このモジュールディレクトリーにコピーされ、その **module.xml** ファイルから参照されます。ディレクトリーへのパスを提供した場合、ディレクトリーとその内容はモジュールディレクトリーにコピーされます。シンボリックリンクは保持されず、リンクされたリソースはモジュールディレクトリーにコピーされます。**--absolute-resources** または **--module-xml** が提供されている場合を除き、この引数は必須です。 区切り文字の詳細は **--resource-delimiter** を参照してください。
--slot	この引数を使用して、デフォルトの **main** スロット以外のスロットにモジュールを追加します。 module add --name=com.mysql --slot=8.0 --resources=/**path**/**to**/{MySQLDriverJarName} --dependencies=wildflyee.api,java.se

4.9. PERIODIC-ROTATING-FILE-AUDIT-LOG 属性

表4.11 periodic-rotating-file-audit-log 属性

属性	説明
autoflush	監査イベントが発生するたびに、出力ストリームのフラッシュが必要かどうかを指定します。属性を定義しない場合、**synchronized** 属性値がデフォルトになります。
encoding	監査ファイルのエンコーディングを指定します。デフォルトは **UTF-8** です。指定可能な値は次のとおりです。 • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**

属性	説明
format	人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。
path	ログファイルの保存場所を定義します。
relative-to	任意の属性です。ログファイルの保存場所を定義します。
接尾辞	任意の属性です。ローテーションされたログに日付の接尾辞を追加します。**java.time.format.DateTimeFormatter** の形式を使用する必要があります。例: **.yyyy-MM-dd**。
synchronized	デフォルト値は **true** です。監査イベントが発生するたびにファイル記述子が同期されることを指定します。

4.10. SASL-AUTHENTICATION-FACTORY 属性

属性を設定して sasl-authentication-factory を設定できます。

表4.12 sasl-authentication-factory 属性

属性	説明
mechanism-configurations	メカニズム固有の設定のリスト。
sasl-server-factory	このリソースに関連付ける SASL サーバーファクトリー。
security-domain	このリソースに関連付けるセキュリティードメイン。

表4.13 sasl-authentication-factory mechanism-configurations 属性

属性	説明
credential-security-factory	メカニズムで必要な認証情報の取得に使用するセキュリティーファクトリー。
final-principal-transformer	このメカニズムレルムに適用する最終のプリンシパルトランスフォーマー。
host-name	この設定が適用されるホスト名。

属性	説明
mechanism-name	この設定は、名前を指定したメカニズムが使用される場合にのみ適用されます。この属性を省略すると、メカニズム名に一致します。
mechanism-realm-configurations	メカニズムが理解するレルム名の定義のリストです。
protocol	この設定が適用されるプロトコル。
post-realm-principal-transformer	レルムの選択後に適用するプリンシパルトランスフォーマー。
pre-realm-principal-transformer	レルムが選択される前に適用するプリンシパルトランスフォーマー。
realm-mapper	メカニズムによって使用されるレルムマッパー。

表4.14 sasl-authentication-factory mechanism-configurations mechanism-realm-configurations 属性

属性	説明
final-principal-transformer	このメカニズムレルムに適用する最終のプリンシパルトランスフォーマー。
post-realm-principal-transformer	レルムの選択後に適用するプリンシパルトランスフォーマー。
pre-realm-principal-transformer	レルムが選択される前に適用するプリンシパルトランスフォーマー。
realm-mapper	メカニズムによって使用されるレルムマッパー。
realm-name	メカニズムにより提示されるレルムの名前。

4.11. SECURITY-DOMAIN 属性

属性を設定することにより、security-domain を設定できます。

属性	説明
default-realm	このセキュリティードメインに含まれるデフォルトのレルム。
evidence-decoder	このドメインで使用される EvidenceDecoder への参照。

属性	説明
outflow-anonymous	この属性は、セキュリティードメインへのアウトフローが不可能な場合に、匿名アイデンティティーを使用するかどうかを指定します。アウトフローが不可能な状況は、次の場合に発生します。 ● アウトフロー先のドメインがこのドメインを信頼していない。 ● ドメインにアウトフローするアイデンティティーが当該ドメインに存在しない。 匿名アイデンティティーをアウトフローすると、そのドメインに対して以前に確立されたアイデンティティーがすべて消去されます。
outflow-security-domains	このドメインのセキュリティーアイデンティティーの自動アウトフロー先となるセキュリティードメインのリスト。
permission-mapper	このドメインで使用される PermissionMapper への参照。
post-realm-principal-transformer	指定のアイデンティティー名でレルムが動作した後に適用するプリンシパルトランスフォーマーへの参照。
pre-realm-principal-transformer	レルムが選択される前に適用するプリンシパルトランスフォーマーへの参照。
principal-decoder	このドメインで使用される PrincipalDecoder への参照。
realm-mapper	このドメインで使用される RealmMapper への参照。
realms	このセキュリティードメインに含まれるレルムのリスト。
role-decoder	このドメインで使用される RoleDecoder への参照。
role-mapper	このドメインで使用される RoleMapper への参照。
security-event-listener	セキュリティーイベントのリスナーへの参照。
trusted-security-domains	このセキュリティードメインによって信頼されているセキュリティードメインのリスト。
trusted-virtual-security-domains	このセキュリティードメインによって信頼されている仮想セキュリティードメインのリスト。

4.12. SIMPLE-ROLE-DECODER 属性

属性を設定することにより、単純なロールデコーダーを設定できます。

表4.15 simple-role-decoder 属性

属性	説明
attribute	直接ロールにマップするアイデンティティーの属性名。

4.13. SIZE-ROTATING-FILE-AUDIT-LOG 属性

表4.16 size-rotating-file-audit-log 属性

属性	説明
autoflush	監査イベントが発生するたびに、出力ストリームのフラッシュが必要かどうかを指定します。属性を定義しない場合、**synchronized** 属性値がデフォルトになります。
encoding	監査ファイルのエンコーディングを指定します。デフォルトは **UTF-8** です。指定可能な値は次のとおりです。 • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**
format	デフォルト値は **SIMPLE** です。人間が判読できるテキスト形式の場合は **SIMPLE** を使用し、個々のイベントを **JSON** に保存する場合は **JSON** を使用します。
max-backup-index	ローテーション時にバックアップするファイルの最大数です。デフォルト値は **1** です。
path	ログファイルの保存場所を定義します。
relative-to	任意の属性です。ログファイルの保存場所を定義します。

属性	説明
rotate-on-boot	デフォルトでは、サーバーの再起動時に新しいログファイルは Elytron により作成されません。この属性を **true** に設定すると、サーバーの再起動時にログを回転させることができます。
rotate-size	Elytron がログをローテーションする前にログファイルが到達する最大サイズ。デフォルトは **10m**(10メガバイト) です。また、ログの最大サイズを k、g、b、t の単位で定義することもできます。単位の指定は、大文字でも小文字でも可能です。
接尾辞	任意の属性です。ローテーションされたログに日付の接尾辞を追加します。**java.text.format.DateTimeFormatter** の形式を使用する必要があります。たとえば、**.yyyy-MM-dd-HH**。
synchronized	デフォルト値は **true** です。監査イベントが発生するたびにファイル記述子が同期されることを指定します。

4.14. SYSLOG-AUDIT-LOG 属性

表4.17 syslog-audit-log 属性

属性	説明
format	監査イベントが記録される形式。 サポートされる値: • **JSON** • **SIMPLE** デフォルト値: • **SIMPLE**
host-name	syslog サーバーに送信されるすべてのイベントに埋め込まれるホスト名。
port	**syslog** サーバーのリスニングポート。

属性	説明
reconnect-attempts	接続を閉じる前に Elytron が **syslog** サーバーへの連続メッセージの送信を試行する最大回数。この属性の値は、使用される送信プロトコルが UDP の場合にのみ有効です。 サポートされる値: ● 正の **integer** 値 ● **-1** は、再接続が無限に試行されることを示します。 デフォルト値: ● **0**
server-address	**syslog** サーバーの IP アドレス、または Java の **InetAddress.getByName()** メソッドで解決できる名前。
ssl-context	**syslog** サーバーへの接続時に使用する SSL コンテキスト。この属性は、**transport** が **SSL_TCP** に設定されている場合にのみ必要です。
syslog-format	監査イベントの記述に使用される RFC 形式。 サポートされる値: ● **RFC3164** ● **RFC5424** デフォルト値: ● **RFC5424**
transport	**syslog** サーバーへの接続に使用するトランスポート層プロトコル。 サポートされる値: ● **SSL_TCP** ● **TCP** ● **UDP** デフォルト値: ● **TCP**