Red Hat JBoss Enterprise Application Platform 8.0

使用身份存储保护应用程序和管理界面

使用身份存储,如文件系统、数据库、轻量级目录访问协议(LDAP)或自定义身份存储来保护 JBoss EAP 管理接口和部署的应用程序的指南

Last Updated: 2024-02-10

Red Hat JBoss Enterprise Application Platform 8.0 使用身份存储保护应用程序和管理界面

使用身份存储,如文件系统、数据库、轻量级目录访问协议(LDAP)或自定义身份存储来保护 JBoss EAP 管理接口和部署的应用程序的指南

法律通告

摘要

使用身份存储，如文件系统、数据库、轻量级目录访问协议(LDAP)或自定义身份存储来保护 JBoss EAP 管理接口和部署的应用程序的指南。

目录

提供有关 JBOSS EAP 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

流程

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 配置身份存储

1.1. 创建 FILESYSTEM-REALM

1.1.1. Elytron 中的文件系统域

有了文件系统安全域 **filesystem-realm**，您可以在 Elytron 中使用基于文件系统的身份存储来存储用户凭证和属性。Elytron 将每个身份以及关联的凭证和属性存储在文件系统的 XML 文件中。XML 文件的名称是身份的名称。您可以将多个凭证和属性与每个身份关联。

默认情况下，身份存储在文件系统中，如下所示：

- Elytron 对存储身份的目录结构应用两级目录哈希。例如，名为 "user1" 的身份存储在 **u/s/user1.xml**。
  这样做是为了克服某些文件系统对您可以在单个目录中存储的文件数量的限制，以及出于性能方面的原因。

  使用 **levels** 属性配置要应用的目录哈希的级数。

- 在被用作文件名之前，身份名称是 Base32 编码的。这样做的原因是某些文件系统区分大小写，或者可能会限制文件名中允许的字符集。
  您可以通过将属性 **encoded** 设为 **false** 来关闭编码。

有关其他属性及其默认值的详情，请参考 **filesystem-realm 属性**。

Encryption
当将身份存储在身份文件中时，**filesystem-realm** 对明文密码、散列密码和属性使用 Base64 编码。为提高安全性，您可以使用存储在凭证存储中的 secret 密钥来加密明文密码、散列密码和属性。secret 密钥用于加密和解密密码和属性。

完整性检查
要确保使用 **filesystem-realm** 创建的身份不会被篡改，您可以通过在创建过程中引用 **filesystem-realm** 中的密钥对来启用对 **filesystem-realm** 的完整性检查。

完整性检查在 **filesystem-realm** 中可以正常工作，如下所示：

- 当您在启用了完整性检查的 **filesystem-realm** 中创建身份时，Elytron 会创建身份文件，并为其生成签名。

- 每当读取身份文件时，例如，当为身份验证更新身份或加载身份时，Elytron 会针对签名验证身份文件内容，以确保文件自上次授权写以来没有被篡改。

- 当您更新具有关联签名的现有身份时，Elytron 会更新内容并在原始内容通过验证后生成新的签名。
  如果验证失败，您会收到以下失败信息：

```
{
    "outcome" => "failed",
    "failure-description" => "WFLYCTL0158: Operation handler
failed:java.lang.RuntimeException: WFLYELY01008: Failed to obtain the authorization
identity.",
    "rolled-back" => true
}
```

其他资源

- **filesystem-realm** 属性
- 在 Elytron 中创建一个 **filesystem-realm**
- 在 Elytron 中创建一个加密的 **filesystem-realm**
- 在 Elytron 中创建一个完整性的 **filesystem-realm**

1.1.2. 在 Elytron 中创建一个 filesystem-realm

创建一个 **filesystem-realm** 和一个引用域的安全域，来保护 JBoss EAP 服务器界面或服务器上部署的应用程序。

先决条件

- JBoss EAP 正在运行。

流程

1. 在 Elytron 中创建一个 **filesystem-realm**。

   语法

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

   Example

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

2. 将用户添加到域，并配置用户的角色。

   a. 添加用户。

      语法

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

      Example

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

   b. 设置用户的密码。

      语法

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

Example

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

c. 为用户设置角色。

语法

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

Example

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

3. 创建一个引用 **filesystem-realm** 的安全域。

语法

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

Example

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

验证

- 要验证 Elytron 是否可以从 **filesystem-realm** 加载身份，请使用以下命令：

语法

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

Example

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

```
            "Admin",
            "Guest"
        ]},
        "roles" => [
            "Guest",
            "Admin"
        ]
    }
}
```

现在，您可以使用创建的安全域来向管理界面和应用程序添加身份验证和授权。如需更多信息，请参阅 保护管理界面和应用程序。

其他资源

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

1.1.3. 在 Elytron 中创建一个加密的 filesystem-realm

创建一个加密的 **filesystem-realm** 来保护 JBoss EAP 应用程序或服务器接口，并确保用户凭据已加密，因此是安全的。

1.1.3.1. 为独立服务器创建一个 secret-key-credential-store

使用管理 CLI 创建一个 **secret-key-credential-store**。当您创建 **secret-key-credential-store** 时，JBoss EAP 默认会生成一个 secret 密钥。生成的密钥的名称是 **key**，其大小为 256 位。

先决条件

- JBoss EAP 正在运行。
- 您为在其下运行 JBoss EAP 的用户帐户至少提供了对包含 **secret-key-credential-store** 的目录的读/写权限。

流程

- 使用以下命令，使用管理 CLI 来创建一个 **secret-key-credential-store** ：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:add(path="<path_to_the_credential_store>", relative-to=<path_to_store_file>)
  ```

  Example

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:add(path=examplePropertiesCredentialStore.cs, relative-to=jboss.server.config.dir)
  {"outcome" => "success"}
  ```

1.1.3.2. 创建一个加密的 filesystem-realm

创建一个加密的 **filesystem-realm** 和一个引用域的安全域，来保护 JBoss EAP 服务器接口或部署在服务器上的应用程序。

先决条件

- JBoss EAP 正在运行。
- 您已创建了一个 **secret-key-credential-store**。
  如需更多信息，请参阅 为独立服务器创建一个 **secret-key-credential-store** 。

流程

1. 在 Elytron 中创建一个加密的 **filesystem-realm**。

   语法

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>,credential-store=<name_of_credential_store>,secret-key=<key>)
   ```

   Example

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

2. 将用户添加到域，并配置用户的角色。

   a. 添加用户。

   语法

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

   Example

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

   b. 设置用户的密码。

   语法

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

   Example

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

c. 为用户设置角色。

语法

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

Example

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

3. 创建一个引用 **filesystem-realm** 的安全域。

语法

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

Example

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

验证

- 要验证 Elytron 是否可以从加密的 **filesystem-realm** 加载身份，请使用以下命令：

语法

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

Example

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

```
        ]},
        "roles" => [
            "Guest",
            "Admin"
        ]
    }
}
```

现在，您可以使用创建的安全域来向管理界面和应用程序添加身份验证和授权。

其他资源

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

1.1.4. 在 Elytron 中创建一个支持完整性的 filesystem-realm

创建一个支持完整性的 **filesystem-realm** 来保护 JBoss EAP 应用或服务器接口，并确保用户凭据未被篡改。

1.1.4.1. 使用管理 CLI 创建密钥对

在 Elytron 中使用密钥对创建密钥存储。

先决条件

- JBoss EAP 正在运行。

流程

1. 创建密钥存储。

   语法

   ```
   /subsystem=elytron/key-store=<key_store_name>:add(path=<path_to_key_store_file>,credential-reference={<password>})
   ```

   Example

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

2. 在密钥对存储中创建密钥对。

   语法

   ```
   /subsystem=elytron/key-store=<key_store_name>:generate-key-pair(alias=<alias>,algorithm=<key_algorithm>,key-
   ```

```
size=<size_of_key>,validity=<validity_in_days>,distinguished-
name="<distinguished_name>")
```

Example

```
/subsystem=elytron/key-store=exampleKeystore:generate-key-
pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,distinguished-
name="CN=localhost")
{"outcome" => "success"}
```

3. 将密钥对持久化到密钥存储文件。

语法

```
/subsystem=elytron/key-store=<key_store_name>:store()
```

Example

```
/subsystem=elytron/key-store=exampleKeystore:store()
{
    "outcome" => "success",
    "result" => undefined
}
```

其他资源

- **key-store** 属性

1.1.4.2. 创建支持完整性的 filesystem-realm

创建一个支持完整性的 **filesystem-realm**，以及一个引用域的安全域，来保护 JBoss EAP 服务器接口或服务器上部署的应用程序。

先决条件

- JBoss EAP 正在运行。
- 您已创建了一个 **secret-key-credential-store**。
  如需更多信息，请参阅使用管理 CLI 创建密钥对。

流程

1. 在 Elytron 中创建 **filesystem-realm**。

语法

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:add(path=<file_path>,key-
store=<key_store_name>,key-store-alias=<key_store_alias>)
```

Example

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:add(path=fs-realm-
```

```
users,relative-to=jboss.server.config.dir, key-store=exampleKeystore, key-store-
alias=localhost)
{"outcome" => "success"}
```

2. 将用户添加到域，并配置用户的角色。

   a. 添加用户。

      语法

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

      Example

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

   b. 设置用户的密码。

      语法

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

      Example

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

   c. 为用户设置角色。

      语法

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

      Example

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

3. 创建一个引用 **filesystem-realm** 的安全域。

   语法

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

Example

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

验证

- 要验证 Elytron 是否可以从 **filesystem-realm** 加载身份，请使用以下命令：

语法

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

Example

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

现在，您可以使用创建的安全域来向管理界面和应用程序添加身份验证和授权。如需更多信息，请参阅 保护管理界面和应用程序。

其他资源

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

1.1.4.3. 在启用了支持完整性的现有 filesystem-realm 中更新密钥对

当现有密钥被泄露的时，您可以更新在启用了支持完整性的 **filesystem-realm** 中引用的密钥对。另外，轮转密钥是一个好的做法。

先决条件

- 您已生成了一个密钥对。

- 您已创建了一个启用了完整性检查的 **filesystem-realm**。
  如需更多信息，请参阅 创建一个支持完整性的 **filesystem-realm** 。

流程

1. 在现有密钥存储中创建密钥对。

   语法

   ```
   /subsystem=elytron/key-store=<key_store_name>:generate-key-pair(alias=<alias>,algorithm=<key_algorithm>,key-size=<size_of_key>,validity=<validity_in_days>,distinguished-name="<distinguished_name>")
   ```

   Example

   ```
   /subsystem=elytron/key-store=exampleKeystore:generate-key-pair(alias=localhost2,algorithm=RSA,key-size=1024,validity=365,distinguished-name="CN=localhost")
   {"outcome" => "success"}
   ```

2. 将密钥对持久化到密钥存储文件。

   语法

   ```
   /subsystem=elytron/key-store=<key_store_name>:store()
   ```

   Example

   ```
   /subsystem=elytron/key-store=exampleKeystore:store()
   {
       "outcome" => "success",
       "result" => undefined
   }
   ```

3. 更新密钥存储别名以引用新密钥对。

   语法

   ```
   /subsystem=elytron/filesystem-realm=<filesystem_realm_name>:write-attribute(name=key-store-alias, value=<key_store_alias>)
   ```

   Example

   ```
   /subsystem=elytron/filesystem-realm=exampleSecurityRealm:write-attribute(name=key-store-alias, value=localhost2)
   {
       "outcome" => "success",
       "response-headers" => {
           "operation-requires-reload" => true,
   ```

```
        "process-state" => "reload-required"
    }
}
```

4. 重新加载服务器。

```
reload
```

5. 使用新密钥对，用新签名更新 **filesystem-realm** 中的文件。

语法

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:update-key-pair()
```

Example

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:update-key-pair()
{"outcome" => "success"}
```

验证

- 使用以下管理 CLI 命令验证 **filesystem-realm** 中引用的密钥对是否已被更新：

语法

```
/subsystem=elytron/filesystem-realm=<filesystem_realm_name>:read-resource()
```

Example

```
/subsystem=elytron/filesystem-realm=exampleSecurityRealm:read-resource()
{
    "outcome" => "success",
    "result" => {
        "credential-store" => undefined,
        "encoded" => true,
        "hash-charset" => "UTF-8",
        "hash-encoding" => "base64",
        "key-store" => "exampleKeystoreFSRealm",
        "key-store-alias" => "localhost2",
        "levels" => 2,
        "secret-key" => undefined,
        "path" => "fs-realm-users",
        "relative-to" => "jboss.server.config.dir"
    }
}
```

filesystem-realm 中引用的密钥对已被更新。

其他资源

- **filesystem-realm** 属性

1.1.5. 加密一个未加密的 filesystem-realm

如果您在 Elytron 中配置了一个 **filesystem-realm**，您可以使用 WildFly Elytron 工具对其添加加密。

1.1.5.1. 为独立服务器创建一个 secret-key-credential-store

使用管理 CLI 创建一个 **secret-key-credential-store**。当您创建 **secret-key-credential-store** 时，JBoss EAP 默认会生成一个 secret 密钥。生成的密钥的名称是 **key**，其大小为 256 位。

先决条件

- JBoss EAP 正在运行。
- 您为在其下运行 JBoss EAP 的用户帐户至少提供了对包含 **secret-key-credential-store** 的目录的读/写权限。

流程

- 使用以下命令，使用管理 CLI 来创建一个 **secret-key-credential-store** ：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:add(path="<path_to_the_credential_store>", relative-to=<path_to_store_file>)
  ```

  Example

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:add(path=examplePropertiesCredentialStore.cs, relative-to=jboss.server.config.dir)
  {"outcome" => "success"}
  ```

1.1.5.2. 将未加密的 filesystem-realm 转换为加密的 filesystem-realm

您可以使用 WildFly Elytron 工具 **filesystem-realm - encrypt 命令将未加密的 filesystem-realm** 转换为加密的。

先决条件

- 您有一个现有的 **filesystem-realm**。
  如需更多信息，请参阅 在 Elytron 中创建一个 **filesystem-realm** 。
- 您已创建了一个 **secret-key-credential-store**。
  如需更多信息，请参阅 为独立服务器创建一个 **secret-key-credential-store** 。
- JBoss EAP 正在运行。

流程

1. 将未加密的 **filesystem-realm** 转换为加密的。

   语法

```
$ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-encrypt --input-location
<existing_filesystem_realm_name> --output-location
JBOSS_HOME/standalone/configuration/<target_filesystem_realm_name> --credential-store
<path_to_credential_store>/<credential_store>
```

Example

```
$ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-encrypt --input-location
JBOSS_HOME/standalone/configuration/fs-realm-users --output-location
JBOSS_HOME/standalone/configuration/fs-realm-users-enc --credential-store
JBOSS_HOME/standalone/configuration/examplePropertiesCredentialStore.cs

Creating encrypted realm for: JBOSS_HOME/standalone/configuration/fs-realm-users
Found credential store and alias, using pre-existing key
```

WildFly Elytron 命令 **filesystem-realm-encrypt** 创建一个使用 **--output-location** 参数指定的 **filesystem-realm**。它还会在 **filesystem-realm** 的根目录处创建一个 CLI 脚本，您可以使用它来在 **elytron** 子系统中添加 **filesystem-realm** 资源。

提示

使用 **--summary** 选项查看命令执行的摘要。

2. 使用生成的 CLI 脚本，在 **elytron** 子系统中添加 **filesystem-realm** 资源。

语法

```
$ JBOSS_HOME/bin/jboss-cli.sh --connect --
file=<target_filesystem_realm_directory>/<target_filesystem_realm_name>.cli
```

Example

```
$ JBOSS_HOME/bin/jboss-cli.sh --connect --file=JBOSS_HOME/standalone/configuration/fs-
realm-users-enc/encrypted-filesystem-realm.cli
```

您可以使用加密的 **filesystem-realm** 创建一个引用域的安全域，来保护 JBoss EAP 服务器接口或部署在服务器上的应用程序。

其他资源

- **filesystem-realm** 属性
- **secret-key-credential-store** 属性
- 在 Elytron 中创建一个加密的 **filesystem-realm**
- 如需有关 WildFly Elytron 工具 **filesystem-realm-encrypt** 命令的更多信息，请运行 **filesystem-realm-encrypt --help** 命令：

```
$ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-encrypt --help
```

1.1.6. 在现有 **filesystem-realm**中添加完整性支持

如果您在 Elytron 中配置了 **filesystem-realm**，您可以使用 WildFly Elytron 工具使用密钥对签名，以启用完整性检查。

1.1.6.1. 使用管理 CLI 创建密钥对

在 Elytron 中使用密钥对创建密钥存储。

先决条件

- JBoss EAP 正在运行。

流程

1. 创建密钥存储。

   语法

   ```
   /subsystem=elytron/key-store=<key_store_name>:add(path=<path_to_key_store_file>,credential-reference={<password>})
   ```

   Example

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

2. 在密钥对存储中创建密钥对。

   语法

   ```
   /subsystem=elytron/key-store=<key_store_name>:generate-key-pair(alias=<alias>,algorithm=<key_algorithm>,key-size=<size_of_key>,validity=<validity_in_days>,distinguished-name="<distinguished_name>")
   ```

   Example

   ```
   /subsystem=elytron/key-store=exampleKeystore:generate-key-pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,distinguished-name="CN=localhost")
   {"outcome" => "success"}
   ```

3. 将密钥对持久化到密钥存储文件。

   语法

   ```
   /subsystem=elytron/key-store=<key_store_name>:store()
   ```

   Example

   ```
   /subsystem=elytron/key-store=exampleKeystore:store()
   ```

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

其他资源

- **key-store** 属性

1.1.6.2. 为 filesystem-realm启用完整性检查

您可以使用 WildFly Elytron 工具 **filesystem-realm -integrity** 命令从现有的非空 **filesystem-realm** -realm 创建一个带有完整性检查的 filesystem-realm。

您可以将 **filesystem-realm-integrity** 命令用于以下用例：

- 从现有 **filesystem-realm** 创建一个带有完整性检查的新 **filesystem-realm**。
- 向现有的 **filesystem-realm** 添加完整性检查。

先决条件

- 您有一个现有的 **filesystem-realm**。
  如需更多信息，请参阅 在 Elytron 中创建一个 **filesystem-realm** 。
- 您已生成了一个密钥对。
  如需更多信息， 请参阅使用管理 CLI 创建密钥对。
- JBoss EAP 正在运行。

流程

1. 使用现有的 **filesystem-realm** 并使用密钥对签名具有完整性的 **filesystem-realm**。
   要为现有 **filesystem-realm** 添加完整性支持，请使用以下命令省略 **--output-location** 和 **--realm-name** 选项。如果您指定了 **--output-location** 和 **--realm-name** 选项，命令会创建一个带有完整性检查的新 **filesystem-realm**，而无需更新现有检查。

   语法

   ```
   $ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-integrity  --input-location <path_to_existing_filesystem_realm> --keystore <path_to_key_store_file> --password <keystore_password> --key-pair <key_pair_alias> --output-location <path_for_new_filesystem_realm> --realm-name <name_of_new_filesystem_realm>
   ```

   Example

   ```
   $ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-integrity --input-location JBOSS_HOME/standalone/configuration/fs-realm-users/ --keystore JBOSS_HOME/standalone/configuration/keystore --password secret --key-pair localhost --output-location JBOSS_HOME/standalone/configuration/fs-realm-users --realm-name exampleRealmWithIntegrity
   ```

   示例输出

   ```
   
   ```

```
Creating filesystem realm with integrity verification for:
JBOSS_HOME/standalone/configuration/fs-realm-users
```

WildFly Elytron 命令 **filesystem-realm-integrity** 创建一个使用 **--output-location** 参数指定的 **filesystem-realm**。它还会在 **filesystem-realm** 的根目录处创建一个 CLI 脚本，您可以使用它来在 **elytron** 子系统中添加 **filesystem-realm** 资源。

提示

使用 **--summary** 选项查看命令执行的摘要。

2. 使用生成的 CLI 脚本，在 **elytron** 子系统中添加 **filesystem-realm** 资源。

语法

```
$ JBOSS_HOME/bin/jboss-cli.sh --connect --
file=<target_filesystem_realm_directory>/<target_filesystem_realm_name>.cli
```

Example

```
$ JBOSS_HOME/bin/jboss-cli.sh --connect --file=JBOSS_HOME/standalone/configuration/fs-
realm-users/exampleRealmWithIntegrity.cli
```

您可以使用 **filesystem-realm** 创建一个引用域的安全域，来保护 JBoss EAP 服务器接口或服务器上部署的应用程序。

其他资源

- **filesystem-realm 属性**
- 在 Elytron 中创建一个支持完整性的 filesystem-realm
- 有关 WildFly Elytron 工具 **filesystem-realm-integrity** 命令的更多信息，请运行 **filesystem-realm-integrity --help** 命令：

```
$ JBOSS_HOME/bin/elytron-tool.sh filesystem-realm-integrity --help
```

1.2. 创建 JDBC 域

1.2.1. 在 Elytron 中创建一个 jdbc-realm

创建一个 **jdbc-realm** 和一个引用域的安全域，来保护 JBoss EAP 服务器接口或部署在服务器上的应用程序。

此流程中的示例使用了 PostgreSQL 数据库，其配置如下：

- 数据库名称：postgresdb
- 数据库登录凭证：
  - 用户名: postgres

- 密码: postgres

- 表名称：example_jboss_eap_users
- example_jboss_eap_users 内容:

用户名	密码	角色
user1	passwordUser1	Admin
user2	passwordUser2	Guest

先决条件

- 您已配置了包含用户的数据库。
- JBoss EAP 正在运行。
- 您已下载了合适的 JDBC 驱动程序。

流程

1. 使用管理 CLI为数据库部署数据库驱动程序。

   语法

   ```
   deploy <path_to_jdbc_driver>/<jdbc-driver>
   ```

   Example

   ```
   deploy PATH_TO_JDBC_DRIVER/postgresql-42.2.9.jar
   ```

2. 将数据库配置为数据源。

   语法

   ```
   data-source add --name=<data_source_name> --jndi-name=<jndi_name> --driver-name=<jdbc-driver>  --connection-url=<database_URL> --user-name=<database_username> --password=<database_username>
   ```

   Example

   ```
   data-source add --name=examplePostgresDS --jndi-name=java:jboss/examplePostgresDS --driver-name=postgresql-42.2.9.jar  --connection-url=jdbc:postgresql://localhost:5432/postgresdb --user-name=postgres --password=postgres
   ```

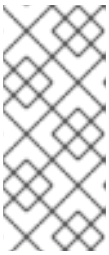
3. 在 Elytron 中创建一个 **jdbc-realm**。

   语法

   ```
   /subsystem=elytron/jdbc-realm=<jdbc_realm_name>:add(principal-query=[<sql_query_to_load_users>])
   ```

Example

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



注意

示例演示了如何从单个 **principal-query** 中获取密码和角色。如果您需要多个查询来获取角色或额外的身份验证或授权信息，您还可以创建具有额外 **attribute-mapping** 属性的 **principal-query**。

有关支持的密码映器的列表，请参阅 密码映射器。

4. 创建一个引用 **jdbc-realm** 的安全域。

语法

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

Example

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

验证

- 要验证 Elytron 是否可以从数据库加载数据，请使用以下命令：

语法

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

Example

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

输出确认 Elytron 可以从数据库加载数据。

现在，您可以使用创建的安全域来向管理界面和应用程序添加身份验证和授权。如需更多信息，请参阅 保护管理界面和应用程序。

其他资源

- **jdbc-realm** 属性
- 密码映射器
- **security-domain** 属性

1.3. 创建一个 LDAP 域

1.3.1. Elytron 中的 LDAP 域

Elytron 中的轻量级目录访问协议(LDAP)域 **ldap-realm** 是一个可用于加载 LDAP 身份存储中身份的安全域。

以下示例演示了 LDAP 中的身份如何与 JBoss EAP 中的 Elytron 身份进行映射。

LDAP 数据交换格式(LDIF)文件的示例

```
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
```

创建 LDAP 域的命令示例

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

```
/subsystem=elytron/ldap-realm=exampleSecurityRealm: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"}]})
```

命令会产生以下配置：

```
<ldap-realm name="exampleLDAPRealm" dir-context="exampleDirContext"> 1
   <identity-mapping rdn-identifier="uid" search-base-dn="ou=Users,dc=wildfly,dc=org"> 2
      <attribute-mapping> 3
         <attribute from="cn" to="Roles" filter="(&amp;(objectClass=groupOfNames)(member={1}))" filter-base-dn="ou=Roles,dc=wildfly,dc=org"/> 4
      </attribute-mapping>
      <user-password-mapper from="userPassword"/> 5
   </identity-mapping>
</ldap-realm>
```

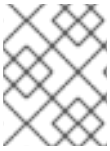
1. realm 定义。
   - **name** 是 **ldap-realm** 域名。
   - **dir-context** 是连接到 LDAP 服务器的配置。
2. 定义身份是如何映射的。
   - **rdn-identifier** 是主体可分辨名称(DN)的相对可分辨名称(DN)，用于从 LDAP 条目获取主体的名称。在示例 LDIF 中，**uid** 配置为代表来自基础 **DN=ou=Users,dc=wildfly,dc=org** 的主体名称。
     search-base-dn 是用于搜索身份的基础 DN。在示例 LDIF 中，它被定义为 **dn: ou=Users,dc=wildfly,dc=org**。
3. 定义 LDAP 属性到身份的属性的映射。
4. 配置如何将特定的 LDAP 属性映射为 Elytron 身份属性。
   - **from** 是要映射的 LDAP 属性。如果没有定义，则使用条目的 DN。
   - **to** 是从 LDAP 属性映射的身份属性的名称。如果没有提供，则属性的名称与 **from** 中定义的属性名称相同。如果 **from** 也未定义，则使用条目的 DN。
   - **filter** 是一个过滤器，用于获取特定属性的值。字符串 '{0}' 被 username 替换， '{1}' 被用户身份 DN 替换。
     - **objectClass** 是要使用的 LDAP 对象类。在示例 LDIF 中，要使用的对象类定义为 **groupOfNames**。
     - **member** 是要映射的成员。**{0}** 被用户名替换，**{1}** 被用户身份 DN 替换。在这个示例中，**{1}** 用于将 **member** 映射为 **user1**。
   - **filter-base-dn** 是应该应用过滤器的上下文的名称。
     示例过滤器的结果是用户 **user1** 使用 **Admin** 角色进行映射。
5. **user-password-mapper** 定义从中获取身份密码的 LDAP 属性。在示例中，它被配置为 **userPassword**，它在 LDIF 中被定义为 **userPassword1** 。

其他资源

- 在 Elytron 中创建一个 **ldap-realm**
- **ldap-realm** 属性

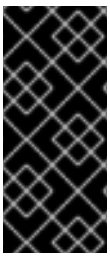
1.3.2. 在 Elytron 中创建一个 ldap-realm

创建一个由轻量级目录访问协议(LDAP)身份存储支持的 Elytron 安全域。使用安全域创建一个安全域，向管理界面或服务器上部署的应用程序添加身份验证和授权。



注意

配置为缓存域的 **ldap-realm** 不支持活动目录。如需更多信息，请参阅 通过 Elytron 的 JBossEAP CLI 修改 LDAP/AD 用户密码 。



重要

在 **elytron** 子系统使用 LDAP 服务器执行身份验证的情况，如果该 LDAP 服务器无法访问，JBoss EAP 将返回 **500** 错误码或内部服务器错误。

为确保使用 LDAP 域保护的管理界面和应用程序可以访问，即使使用故障切换域使 LDAP 服务器变得可用。如需更多信息，请参阅 创建故障转移域。

对于此过程中的示例，使用了以下 LDAP 数据交换格式(LDIF)：

```
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
```

用于示例的 LDAP 连接参数如下：

- LDAP URL ：**ldap://10.88.0.2**

- LDAP admin 密码：**secret**
  您需要此内容来使 Elytron 与 LDAP 服务器连接。
- LDAP admin 可分辨名称(DN): **(cn=admin,dc=wildfly,dc=org)**
- LDAP 机构：**wildfly**
  如果没有指定机构名称，则默认为 **Example Inc**。
- LDAP 域：**wildfly.org**
  这是当平台收到 LDAP 搜索参考时匹配的名称。

先决条件

- 您已配置了一个 LDAP 身份存储。
- JBoss EAP 正在运行。

流程

1. 配置一个提供 URL 的目录上下文，以及用于连接到 LDAP 服务器的主体。

   语法

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

   Example

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

2. 创建一个引用目录上下文的 LDAP 域。指定搜索基础 DN 以及用户的映射方式。

   语法

   ```
   /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>"}]})
   ```

   Example

   ```
   /subsystem=elytron/ldap-realm=exampleSecurityRealm: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"}]})
   ```

如果您在 LDIF 文件中存储了哈希密码，您可以指定以下属性：

- **hash-encoding** ：如果密码没有以纯文本存储，则此属性指定密码的字符串格式。默认情况下，它被设置为 **base64** 编码，但也支持 **hex**。
- **hash-charset** ：此属性指定在将密码字符串转换为字节数组时要使用的字符集。默认设为 **UTF-8**。

警告

如果引用的 LDAP 服务器在引用中包含了一个循环，则可能会在 JBoss EAP 中导致 **java.lang.OutOfMemoryError** 错误。

3. 创建一个角色解码器，来将属性映射到角色。

语法

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

Example

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

4. 创建一个引用 LDAP 域和角色解码器的安全域。

语法

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

Example

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

现在，您可以使用创建的安全域来向管理界面和应用程序添加身份验证和授权。如需更多信息，请参阅 保护管理界面和应用程序。

其他资源

- **ldap-realm** 属性
- **security-domain** 属性

1.4. 创建一个属性域

1.4.1. 在 Elytron 中创建一个引用 properties-realm 的安全域

创建一个 **properties-realm** 和一个引用域的安全域，来保护您的 JBoss EAP 管理接口或您在服务器上部署的应用程序。

先决条件

- JBoss EAP 正在运行。
- 您有一个授权用户和一个现有的具有正确域的遗留属性文件，该域写在 **users.properties** 文件中注释掉的行中：

  示例 $EAP_HOME/standalone/configuration/my-example-users.properties

  ```
  #$REALM_NAME=exampleSecurityRealm$
  user1=078ed9776d4b8e63b6e51135ec45cc75
  ```

  - **user1** 的密码为 **userPassword1**。密码被哈希成文件 **HEX (MD5 (user1:exampleSecurityRealm:userPassword1))**。
- **users.properties** 文件中列出的授权用户在 **groups.properties** 文件中有一个角色：

  示例 $EAP_HOME/standalone/configuration/my-example-groups.properties

  ```
  user1=Admin
  ```

流程

1. 在 Elytron 中创建一个 **properties-realm**。

   语法

   ```
   /subsystem=elytron/properties-realm=<properties_realm_name>:add(users-properties=
   {path=<file_path>},groups-properties={path=<file_path>})
   ```

   Example

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

2. 创建一个引用 **properties-realm** 的安全域。

   语法

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

   Example

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

验证

- 要验证 Elytron 是否可以从属性文件中加载数据，请使用以下命令：

  语法

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

  Example

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

  输出确认了 Elytron 可以从属性文件加载数据。

现在，您可以使用创建的安全域来向管理界面和应用程序添加身份验证和授权。如需更多信息，请参阅 保护管理界面和应用程序。

其他资源

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

1.5. 创建一个自定义域

1.5.1. 在 Elytron 中添加一个 custom-realm 安全域

您可以使用 **custom-realm** 创建一个为您的用例量身定制的 Elytron 安全域。当现有 Elytron 安全域不适合您的用例时，您可以添加一个 **custom-realm**。

先决条件

- JBoss EAP 已安装并运行。
- 已安装 Maven。
- 您有一个实现的自定义域 java 类。

流程

1. 实现一个自定义域 java 类，并将其打包为 **JAR** 文件。

```
$ mvn package
```

2. 添加一个包含自定义域实现的模块。

语法

```
module add --name=<name_of_your_wildfly_module>
--resources=<path_to_custom_realm_jar> --dependencies=org.wildfly.security.elytron
```

Example

```
module add --name=com.example.customrealm --resources=EAP_HOME/custom-realm.jar -
-dependencies=org.wildfly.security.elytron
```

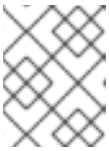
3. 创建 **custom-realm**。

语法

```
/subsystem=elytron/custom-
realm=<name_of_your_custom_realm>:add(module=<name_of_your_wildfly_module>,class-
name=<class_name_of_custom_realm_>,configuration=
{<configuration_option_1>=<configuration_value_1>,<configuration_option_2>=<configuratio
n_value_2>})
```

Example

```
/subsystem=elytron/custom-realm=example-
realm:add(module=com.example.customrealm,class-
name=com.example.customrealm.ExampleRealm,configuration=
{exampleConfigOption1=exampleConfigValue1,exampleConfigOption2=exampleConfigValue2})
```



注意

本例希望实现的自定义域具有类名称 **com.example.customrealm.ExampleRealm**。

注意

您可以使用 **configuration** 属性将 **key**/**value** 配置传给 **custom-realm**。**configuration** 属性是可选的。

4. 根据您创建的域定义一个安全域。

语法

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

Example

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(realms=[{realm=example-realm}],default-realm=example-realm,permission-mapper=default-permission-mapper)
```

现在，您可以使用创建的安全域来向管理界面和应用程序添加身份验证和授权。如需更多信息，请参阅 保护管理界面和应用程序。

其他资源

- **custom-realm** 属性
- **security-domain** 属性
- 要了解更多有关 **module add** 命令的信息，您可以在 JBoss EAP 管理 CLI 中运行 **--help** 命令：

```
module add --help
```

第 2 章 保护管理界面和应用程序

2.1. 向管理界面添加身份验证和授权

您可以为管理界面添加身份验证和授权，以便使用安全域来保护它们。要在添加身份验证和授权后访问管理界面，用户必须输入登录凭证。

您可以按照以下方法保护 JBoss EAP 管理界面：

- 管理 CLI
  通过配置一个 **sasl-authentication-factory**。
- 管理控制台
  通过配置一个 **http-authentication-factory**.

先决条件

- 您已创建了一个引用安全域的安全域。
- JBoss EAP 正在运行。

流程

1. 创建一个 **http-authentication-factory** 或 **sasl-authentication-factory**。
   - 创建一个 **http-authentication-factory**。

     语法

     ```
     /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>}]}])
     ```

     Example

     ```
     /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"}
     ```

   - 创建一个 **sasl-authentication-factory**。

     语法

     ```
     /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>}]}])
     ```

     Example

```
/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. 更新管理界面。

- 使用 **http-authentication-factory** 来保护管理控制台。

  语法

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

  Example

  ```
  /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"
      }
  }
  ```

- 使用 **sasl-authentication-factory** 来保护管理 CLI。

  语法

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

  Example

  ```
  /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. 重新加载服务器。

```
reload
```

验证

- 要验证管理控制台是否需要身份验证和授权，请导航到位于 **http://127.0.0.1:9990/console/index.html** 的管理控制台。会提示您输入用户名和密码。
- 要验证管理 CLI 是否需要身份验证和授权，请使用以下命令启动管理 CLI：

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

会提示您输入用户名和密码。

其他资源

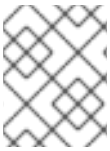
- **http-authentication-factory** 属性
- **sasl-authentication-factory** 属性

2.2. 使用安全域对应用程序用户进行身份验证和授权

使用引用安全域的安全域来验证和授权应用用户。开发应用程序的流程仅作为示例提供。

2.2.1. 开发一个简单的 Web 应用程序

您可以创建一个简单的 Web 应用来遵循配置安全域示例。

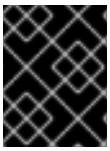


注意

以下流程仅作为示例提供。如果您已有一个要保护的应用程序，您可以跳过这些，并直接进入 向应用程序添加身份验证和授权。

2.2.1.1. 为 web 应用程序开发创建一个 Maven 项目

要创建一个 web 应用程序，请创建一个具有所需依赖项和目录结构的 Maven 项目。



重要

以下流程仅作为示例提供，不应在生产环境中使用。有关为 JBoss EAP 创建应用程序的详情，请参考 为 JBoss EAP 部署开发应用程序。

先决条件

- 您已安装了 Maven。如需更多信息，请参阅 下载 Apache Maven。

流程

1. 使用 **mvn** 命令建立一个 Maven 项目。该命令创建项目的目录结构以及 **pom.xml** 配置文件。

   语法

```
$ 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
```

Example

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

2. 进入到应用程序根目录：

语法

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

Example

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

3. 将生成的 **pom.xml** 文件的内容替换为以下文本：

```
<?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>
      <version.maven.war.plugin>3.4.0</version.maven.war.plugin>
    </properties>

    <dependencies>
      <dependency>
        <groupId>jakarta.servlet</groupId>
        <artifactId>jakarta.servlet-api</artifactId>
        <version>6.0.0</version>
```

```
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <finalName>${project.artifactId}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>${version.maven.war.plugin}</version>
            </plugin>
            <plugin>
                <groupId>org.wildfly.plugins</groupId>
                <artifactId>wildfly-maven-plugin</artifactId>
                <version>4.2.2.Final</version>
            </plugin>
        </plugins>
    </build>
</project>
```

验证

- 在应用程序根目录中，输入以下命令：

```
$ mvn install
```

您会看到类似如下的输出：

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

现在，您可以创建一个 web 应用程序。

2.2.1.2. 创建一个 Web 应用程序

创建一个 Web 应用程序，其中包含一个 servlet，它将返回从登录的用户主体中获取的用户名。如果没有登录的用户，servlet 将返回文本"NO AUTHENTICATED USER"。

在此过程中，*<application_home>* 指向包含应用程序 **pom.xml** 配置文件的目录。

先决条件

- 您已创建了一个 Maven 项目。
  如需更多信息，请参阅 为 web 应用程序开发创建一个 Maven 项目 。
- JBoss EAP 正在运行。

流程

1. 创建一个用于存储 Java 文件的目录。

   语法

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

   Example

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

2. 前往新目录。

   语法

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

   Example

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

3. 使用以下内容创建一个 **securedServlet.java** 文件：

```
package com.example.app;

import java.io.IOException;
import java.io.PrintWriter;
import java.security.Principal;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

/**
 * A simple secured HTTP servlet. It returns the user name of 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()) {
            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 '");
            Principal user = req.getUserPrincipal();
```

```
            writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
            writer.print("""");
            writer.println("    </p>");
            writer.println("  </body>");
            writer.println("</html>");
        }
    }

}
```

4. 在应用程序根目录中，使用以下命令编译应用程序：

```
$ 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. 部署应用。

```
$ mvn wildfly:deploy
```

验证

- 在浏览器中，导航到 **http://localhost:8080/simple-webapp-example/secured**。
  您会收到以下信息：

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

  因为没有添加验证机制，所以您可以访问应用程序。

现在，您可以使用安全域来保护这个应用程序，这样只有经过身份验证的用户才可以访问它。

2.2.2. 向应用程序添加身份验证和授权

您可以使用安全域向 Web 应用程序添加身份验证和授权，以对其进行保护。要在添加身份验证和授权后访问 web 应用程序，用户必须输入登录凭证。

先决条件

- 您已创建了一个引用安全域的安全域。
- 您已在 JBoss EAP 上部署了应用程序。
- JBoss EAP 正在运行。

流程

1. 在 **undertow 子系统** 中配置一个**application-security-domain** ：

语法

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

Example

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

2. 配置应用程序的 **web.xml** 以保护应用程序资源。

语法

```
<!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>
```

Example

```
<!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>
```

注意

您可以使用不同的 **auth-method**。

3. 通过在应用程序中创建一个 **jboss-web.xml** 文件，或者在 **undertow** 子系统中设置默认安全域，将应用程序配置为使用安全域。

   - 在应用程序的 **WEB-INF** 目录中创建引用 **application-security-domain** 的 **jboss-web.xml** 文件。

     语法

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

     Example

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

- 在 **undertow** 子系统中，为应用程序设置默认安全域。

  语法

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

  Example

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

4. 重新加载服务器。

   ```
   reload
   ```

验证

1. 在应用程序根目录中，使用以下命令编译应用程序：

   ```
   $ 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. 部署应用。

   ```
   $ mvn wildfly:deploy
   ```

3. 在浏览器中，导航到 **http://localhost:8080/simple-webapp-example/secured**。您会收到一个登录提示，确认现在需要身份验证才能访问应用程序。

现在，您的应用程序已使用安全域进行了保护，用户可以在身份验证后登录。另外，只有具有指定角色的用户才可以访问应用程序。

第 3 章 配置具有身份域的 ELYTRON，以允许轻松验证和授权本地用户

您可以使用 Elytron 提供的 **identity-realm** 来允许本地用户连接到 JBoss EAP 管理界面。

JBoss EAP 管理 CLI 被预先配置为使用名为 *local* 的 **identity-realm**。这允许本地用户连接，而不用提供凭证。身份域只能与 *JBOSS-LOCAL-USER* 机制一起使用。

3.1. 保护带有身份域的管理界面

您可以使用带有 *JBOSS-LOCAL-USER* 机制的 **identity-realm** 安全域来保护管理界面。

先决条件

- JBoss EAP 正在运行。

流程

1. 创建一个本地 **identity-realm**。

   语法

   ```
   /subsystem=elytron/identity-realm=
   <local_identity_realm_name>:add(identity="$local",attribute-
   name=<attribute_name>,attribute-values=<attribute_value>)
   ```

   Example

   ```
   /subsystem=elytron/identity-
   realm=exampleLocalIdentityRealm:add(identity="$local",attribute-
   name=AttributeName,attribute-values=Value)
   ```

   a. **可选** 如果要为 *$local* 以外的本地 **identity-realm** 使用名称，请修改 **configurable-sasl-server-factory=*<sasl_server_factory>*** 中的 **wildfly.sasl.local-user.default-user** 属性的值。

      语法

      ```
      /subsystem=elytron/configurable-sasl-server-factory=<sasl_server_factory>:write-
      attribute(name=properties,value={"wildfly.sasl.local-user.default-user" =>
      "<new_local_username>", "wildfly.sasl.local-user.challenge-path" => expression
      "${jboss.server.temp.dir}/auth"})
      ```

      Example

      ```
      /subsystem=elytron/configurable-sasl-server-factory=configured:write-
      attribute(name=properties,value={"wildfly.sasl.local-user.default-user" => "john",
      "wildfly.sasl.local-user.challenge-path" => expression "${jboss.server.temp.dir}/auth"})
      ```

2. 创建一个引用您创建的 **identity-realm** 的安全域。

   语法

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

Example

```
/subsystem=elytron/security-domain=exampleSecurityDomain:add(default-
realm=exampleLocalIdentityRealm,permission-mapper=default-permission-mapper,realms=
[{realm=exampleLocalIdentityRealm}])
```

3. 添加 SASL 身份验证工厂。

语法

```
/subsystem=elytron/sasl-authentication-factory=<sasl_auth_factory_name>:add(security-
domain=<security_domain_name>,sasl-server-factory=configured,mechanism-
configurations=[{mechanism-name=JBOSS-LOCAL-USER}])
```

Example

```
/subsystem=elytron/sasl-authentication-
factory=exampleSaslAuthenticationFactory:add(security-
domain=exampleSecurityDomain,sasl-server-factory=configured,mechanism-configurations=
[{mechanism-name=JBOSS-LOCAL-USER}])
```

4. 为您的管理界面启用 SASL 身份验证工厂。

语法

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

Example

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

5. 重新加载您的管理界面。

```
$ reload
```

其他资源

- **identity-realm** 属性
- **security-domain** 属性
- **sasl-authentication-factory** 属性

第 4 章 在 ELYTRON 中配置审计日志记录

您可以使用 Elytron 在触发事件时完成安全审核。安全审计指的是触发事件，如写入日志以响应授权或身份验证尝试。

对事件执行的安全审计类型取决于您的安全域配置。

4.1. ELYTRON 审计日志记录

在使用 **elytron** 子系统启用审计日志记录后，您可以在应用服务器中记录 Elytron 身份验证和授权事件。Elytron 以 **JSON** 或 **SIMPLE** 格式存储审计日志条目。使用 **SIMPLE** 进行人类可读的文本格式或 **JSON**，以 **JSON** 形式存储各个事件。

Elytron 审计日志记录与其他类型的审计日志记录不同，如 JBoss EAP 管理接口的审计日志记录。

但是，Elytron 默认禁用审计日志记录，您可以通过配置以下任何日志处理程序来启用审计日志记录。您还可以将日志处理程序添加到安全域。

- 文件审计日志记录
  如需更多信息，请参阅在 Elytron 中启用文件审计日志记录。
- 定期轮转文件审计日志记录
  如需更多信息，请参阅在 Elytron 中启用定期轮转文件审计日志记录。
- 大小轮转文件审计日志记录
  如需更多信息，请参阅在 Elytron 中启用大小轮转文件审计日志记录。
- **syslog** 审计日志记录
  如需更多信息，请参阅在 Elytron 中启用 syslog 审计日志记录。
- 自定义审计日志记录
  如需更多信息，请参阅在 Elytron 中使用自定义安全事件监听程序。

您可以使用 **aggregate-security-event-listener 资源** 将安全事件发送到更多目的地，如日志记录器。**aggregate-security-event-listener 资源** 将所有事件发送到聚合监听器定义中指定的所有监听程序。

4.2. 在 ELYTRON 中启用文件审计日志记录

文件审计日志记录将审计日志消息存储在文件系统中的一个文件中。

默认情况下，Elytron 将 **local-audit** 指定为文件审计日志记录器。

您必须启用 **local-audit**，以便它可以将 Elytron 审计日志写入单机服务器上的 **EAP_HOME/standalone/log/audit.log**，或者用于受管域的 **EAP_HOME/domain/log/audit.log**。

先决条件

- 您已保护了应用程序。
  如需更多信息，请参阅 在 Elytron 中创建一个 **filesystem-realm** 。

流程

1. 创建文件审计日志。

   语法

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

Example

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

2. 将文件审计日志添加到安全域。

语法

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

Example

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

验证

1. 在浏览器中，登录您的安全应用。
例如，要登录到 使用安全域创建的应用程序来验证和授权应用程序用户，请导航到 http://localhost:8080/simple-webapp-example/secured 并登录。

2. 导航到配置为存储审计日志的目录。如果您在流程中使用示例命令，则目录为 *EAP_HOME*/standalone/log。
请注意，创建一个名为 **file-audit.log** 的文件。它包含通过登录到应用程序来触发的事件的日志。

file-audit.log 文件示例

```
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}
```

其他资源

- **file-audit-log** 属性

4.3. 在 ELYTRON 中启用定期轮转文件审计日志记录

您可以使用 **elytron** 子系统为单机服务器或作为受管域运行的服务器启用定期轮转文件审计日志记录。

定期轮转文件审计日志记录会根据您配置的调度自动轮转审计日志文件。定期轮转文件审计日志记录与默认文件审计日志记录类似，但定期轮转文件审计日志记录包含额外的属性： **suffix**。

suffix 属性的值是一个使用 **java.time.format.DateTimeFormatter** 格式指定的日期，如 **.yyyy-MM-dd**。Elytron 自动从后缀提供的值计算轮转周期。**elytron** 子系统在日志文件名称末尾附加后缀。

先决条件

- 您已保护了应用程序。
  如需更多信息，请参阅 在 Elytron 中创建一个 **filesystem-realm** 。

流程

1. 创建定期轮转文件审计日志。

   语法

   ```
   /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>")
   ```

   Example

   ```
   /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. 将定期轮转文件审计日志记录器添加到安全域。

   语法

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

   Example

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

验证

1. 在浏览器中，登录您的安全应用。
   例如，要登录到 使用安全域创建的应用程序来验证和授权应用程序用户，请导航到 http://localhost:8080/simple-webapp-example/secured 并登录。

2. 导航到配置为存储审计日志的目录。如果您在流程中使用示例命令，则目录为 *EAP_HOME*/standalone/log。
   请注意，创建一个名为 **periodic-file-audit.log** 的文件。它包含通过登录到应用程序来触发的事件的日志。

   periodic-file-audit.log 文件示例

   ```
   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}
```

其他资源

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

4.4. 在 ELYTRON 中启用大小轮转文件审计日志记录

您可以使用 **elytron** 子系统为单机服务器或作为受管域运行的服务器启用大小轮转文件审计日志记录。

当日志文件达到配置的文件大小时，大小轮转文件审计日志记录会自动轮转审计日志文件。大小轮转文件审计日志记录与默认文件审计日志记录类似，但大小轮转文件审计日志记录包含额外的属性。

当日志文件大小超过 **rotate-size** 属性定义的限制时，Elytron 会将后缀 **.1** 附加到当前文件的末尾，并创建新的日志文件。对于每个现有日志文件，Elytron 将后缀递增一。例如，Elytron 将 **audit_log.1** 重命名为 **audit_log.2**。Elytron 继续递增，直到日志文件数量达到最大日志文件数，如 **max-backup-index** 定义。当日志文件超过 **max-backup-index** 值时，Elytron 会删除该文件。例如，如果 **max-backup-index** 将 "98" 定义为 **max-backup-index** 值，则 **audit_log.99** 文件将超过限制。

先决条件

- 您已保护了应用程序。
  如需更多信息，请参阅 在 Elytron 中创建一个 **filesystem-realm** 。

流程

1. 创建大小轮转文件审计日志。

   语法

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

   Example

   ```
   /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. 将大小轮转审计日志记录器添加到安全域。

   语法

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

   Example

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

验证

1. 在浏览器中，登录您的安全应用。
   例如，要登录到 使用安全域创建的应用程序来验证和授权应用程序用户，请导航到 http://localhost:8080/simple-webapp-example/secured 并登录。

2. 导航到配置为存储审计日志的目录。如果您在流程中使用示例命令，则目录为 *EAP_HOME*/standalone/log。
   请注意，创建一个名为 **size-file-audit.log** 的文件。它包含通过登录到应用程序来触发的事件的日志。

   size-file-audit.log 文件示例

```
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}
```

其他资源

- **size-rotating-file-audit-log** 属性

4.5. 在 ELYTRON 中启用 SYSLOG 审计日志记录

您可以使用 **elytron** 子系统为单机服务器或作为受管域运行的服务器启用 **syslog** 审计日志记录。使用 **syslog** 审计日志记录时，您可以将日志记录结果发送到 **syslog** 服务器，它比记录到本地文件提供了更多的安全选项。

syslog 处理程序指定用于连接 **syslog** 服务器的参数，如 **syslog** 服务器的主机名和 **syslog** 服务器侦听的端口。您可以定义多个 **syslog** 处理程序，并同时激活它们。

支持的日志格式包括 **RFC5424** 和 **RFC3164**。支持的传输协议包括 UDP、TCP 和 TCP （使用 SSL）。

当您为第一个实例定义 **syslog** 时，日志记录器会将包含消息的 **INFORMATIONAL** 优先级事件发送到 **syslog** 服务器，如下例所示：

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

<format > 指的是为审计日志处理器配置的 Request for Comments (RFC)格式，默认为 **RFC5424**。

先决条件

- 您已保护了应用程序。
  如需更多信息，请参阅 在 Elytron 中创建一个 **filesystem-realm** 。

流程

1. 添加 **syslog** 处理程序。

   语法

   ```
   /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>)
   ```

   您还可以通过 TLS 将日志发送到 **syslog** 服务器：

   syslog 配置通过 TLS 发送日志的语法

   ```
   /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. 将 **syslog** 审计日志记录器添加到安全域。

   语法

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

   Example

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

其他资源

- **syslog-audit-log** attributes
- 使用 **client-ssl-context**
- Syslog 协议
- BSD syslog 协议

4.6. 在 ELYTRON 中使用自定义安全事件监听程序

您可以使用 Elytron 定义自定义事件监听程序。自定义事件监听程序处理传入的安全事件。您可以使用事件监听程序进行自定义审计日志记录，也可以使用事件监听程序针对内部身份存储验证用户。

重要

使用 **module** 管理 CLI 命令添加和删除模块的功能仅作为技术预览功能提供。**module** 命令不适合在受管域中使用，或者在连接远程管理 CLI 时使用。您必须在生产环境中手动添加或删除模块。

红帽产品服务级别协议(SLA)不支持技术预览功能，且其功能可能并不完善，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

如需有关技术预览功能支持范围的信息，请参阅红帽客户门户网站上的 技术预览功能支持范围。

先决条件

- 您已保护了应用程序。
  如需更多信息，请参阅 在 Elytron 中创建一个 **filesystem-realm** 。

流程

1. 创建一个实施 **java.util.function.Consumer<org.wildfly.security.auth.server.event.SecurityEvent>** 接口的类。

   创建使用指定接口的 Java 类示例：

   ```
   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());
           }
       }
   }
   ```

   当用户成功或失败身份验证时，示例中的 Java 类会打印一条消息。

2. 添加 JAR 文件，它将自定义事件监听程序作为 JBoss EAP 模块提供。
   以下是管理 CLI 命令的示例，它将自定义事件监听程序作为模块添加到 Elytron 中。

   使用 module 命令将自定义事件监听程序作为模块添加到 Elytron 的示例：

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

3. 引用安全域中的自定义事件监听程序。

   在 ApplicationDomain 中引用自定义事件监听程序的示例：

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

4. 重新启动服务器。

```
$ reload
```

事件监听器从指定的安全域接收安全事件。

其他资源

- 手动创建自定义模块
- 手动删除自定义模块
- 在 Elytron 中添加自定义组件

第 5 章 参考

5.1. CUSTOM-REALM 属性

您可以通过设置其属性来配置 **custom-realm**。

表 5.1. custom-realm 属性

属性	描述
class-name	自定义域实现的完全限定类名称。
configuration	自定义域的可选的 **key/value** 配置。
module	用于加载自定义域的模块的名称。

5.2. FILESYSTEM-REALM 属性

您可以通过设置其属性来配置 **filesystem-realm**。

表 5.2. filesystem-realm 属性

属性	描述
credential-store	引用包含 secret 密钥的凭证存储，以加密和解密明文密码、散列密码以及域中的属性。使用此属性时，还必须通过在 **secret-key** 属性中定义它来指定要使用的 secret 密钥。
encoded	指示身份名称是否应该以编码(Base32)的方式存储在文件名中的属性。默认值为 **true**。
hash-charset	在将密码字符串转换为字节数组时会使用字符集。默认为 UTF-8。
hash-encoding	没有以纯文本形式存储的密码的字符串格式。它可以是以下之一： • base64 • hex 默认为 base64。
key-store	使用对包含密钥对的密钥存储的引用来验证完整性。在定义此属性时，还必须在 **key-store-alias** 属性中指定密钥存储别名。
key-store-alias	使用标识密钥存储中私钥条目的别名来验证完整性。如果您通过定义 **key-store** 属性添加了对密钥存储的引用，请使用此属性。
levels	要应用的目录哈希的级别数。默认值为 **2**。

属性	描述
path	包含域的目录的路径。
relative-to	与 **path** 一起使用的预定义的相对路径。例如 **jboss.server.config.dir**。
secret-key	加密和解密明文密码、散列密码和域中属性的 secret 密钥的别名。如果您通过定义 **credential-store** 属性添加了对凭证存储的引用，请使用此属性。

5.3. FILE-AUDIT-LOG 属性

表 5.3. file-audit-log 属性

属性	描述
Autoflush	指定输出流是否在每个审计事件后清除。如果没有定义属性，则 **同步** 的属性值是默认值。
编码	指定审计文件编码。默认值为 **UTF-8**。可能的值如下： • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**
格式	默认值为 **SIMPLE**。使用 **SIMPLE** 进行人类可读的文本格式或 **JSON**，以 **JSON** 形式存储各个事件。
path	定义日志文件的位置。
relative-to	可选属性。定义日志文件的位置。
synchronized	默认值为 **true**。指定文件描述符在每次审计事件后同步。

5.4. HTTP-AUTHENTICATION-FACTORY 属性

您可以通过设置其属性来配置 **http-authentication-factory**。

表 5.4. http-authentication-factory 属性

属性	描述
http-server-mechanism-factory	**HttpServerAuthenticationMechanismFactory** 与这个资源关联。
mechanism-configurations	特定于机制的配置的列表。
security-domain	与资源关联的安全域。

表 5.5. http-authentication-factory mechanism-configurations 属性

属性	描述
credential-security-factory	用于获取机制所要求的凭证的安全因素。
final-principal-transformer	应用于此机制域的最后一个主体转换器。
host-name	此配置应用到的主机名。
mechanism-name	此配置仅应用于使用指定名称机制的情况。如果省略此属性，则它将匹配任何机制名称。
mechanism-realm-configurations	机制理解的域名称的定义的列表。
pre-realm-principal-transformer	在选择域之前要应用的主体转换器。
post-realm-principal-transformer	选择域之后要应用的主体转换器。
protocol	此配置适用的协议。
realm-mapper	机制使用的 realm mapper。

表 5.6. http-authentication-factory mechanism-configurations mechanism-realm-configurations 属性

属性	描述
final-principal-transformer	应用于此机制域的最后一个主体转换器。
post-realm-principal-transformer	选择域之后要应用的主体转换器。
pre-realm-principal-transformer	在选择域之前要应用的主体转换器。
realm-mapper	机制使用的 realm mapper。
realm-name	按机制要显示的域的名称。

5.5. IDENTITY-REALM 属性

您可以通过设置其属性来配置 **identity-realm**。

表 5.7. identity-realm 属性

属性	描述
attribute-name	与此身份关联的属性的名称。
attribute-values	与身份属性关联的值的列表。
identity	安全域中提供的身份。

5.6. JDBC-REALM 属性

您可以通过设置其属性来配置 **jdbc-realm**。

表 5.8. jdbc-realm 属性

属性	描述
hash-charset	在将密码字符串转换为字节数组时会使用字符集。默认为 UTF-8。
principal-query	根据特定密钥类型用来验证用户身份的身份验证查询的列表。

表 5.9. JDBC-realm principal-query 属性

属性	描述
attribute-mapping	为这个资源定义的属性映射的列表。
bcrypt-mapper	将从 SQL 查询返回的列映射到 **Bcrypt** 密钥类型的密钥映射器。
clear-password-mapper	将从 SQL 查询返回的列映射到明文密钥类型的密钥映射器。它具有 **password-index** 子元素，这是列入代表用户密码的身份验证查询的索引。
data-source	用来连接到数据库的数据源的名称。
salted-simple-digest-mapper	将从 SQL 查询返回的列映射到 **Salted Simple Digest** 密钥类型的密钥映射器。
scram-mapper	将从 SQL 查询返回的列映射到 **SCRAM** 密钥类型的密钥映射器。

属性	描述
simple-digest-mapper	将从 SQL 查询返回的列映射到 **Simple Digest** 密钥类型的密钥映射器。
sql	用于获取特定用户表列的密钥的 SQL 语句，并使用其类型对其进行映射。

表 5.10. jdbc-realm principal-query attribute-mapping 属性

属性	描述
index	表示映射属性的 SQL 查询中的列索引。
to	从 SQL 查询返回的列映射的身份属性的名称。

其他资源

- Password mapper 属性

5.7. KEY-STORE 属性

您可以通过设置其属性来配置 **key-store**。

表 5.11. key-store 属性

属性	描述
alias-filter	要应用到从密钥存储返回的别名的过滤器，可以是逗号分隔的别名列表，可以返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**  **注意** **alias-filter** 属性区分大小写。由于使用混合情况或大写字母别名（如 **elytronAppServer** ）可能无法被某些密钥存储供应商识别，因此建议使用小写别名，如 **elytronappserver**。
credential-reference	用于访问密钥存储的密码。这可以以明文形式指定，或作为对存储在 **credential-store** 中的凭据的引用。
path	密钥存储文件的路径。
provider-name	用于加载密钥存储的提供程序名称。设置此属性时，搜索可以创建指定类型的密钥存储的第一个供应商被禁用。

属性	描述
providers	对用于获取要搜索的供应商实例列表的提供程序的引用。如果没有指定，则使用全局供应商列表。
relative-to	这个存储相对于的基本路径。这可以是完整路径或预定义路径，如 **jboss.server.config.dir**。
required	如果设置为 **true**，则引用的密钥存储文件必须在密钥存储服务启动时存在。默认值为 **false**。
type	密钥存储的类型，如 **JKS**。 **注意** 以下密钥存储类型会被自动检测到： • **JKS** • **JCEKS** • **PKCS12** • **BKS** • **BCFKS** • **UBER** 您必须手动指定其他密钥存储类型。 有关密钥存储类型的完整列表，请参阅 Oracle JDK 文档中的 JDK 11 的 Java Cryptography Architecture Standard Algorithm Name 文档。

5.8. LDAP-REALM 属性

您可以通过设置其属性来配置 **ldap-realm**。

表 5.12. ldap-realm 属性

属性	描述
allow-blank-password	此域是否支持空白密码直接验证。如果没有设置此属性，则会拒绝空密码尝试。
dir-context	用于连接到 LDAP 服务器的 **dir-context** 的名称。

属性	描述
direct-verification	如果此属性设为 **true**，则该域支持通过直接连接到 LDAP 作为被身份验证的帐户来验证凭据。否则，从 LDAP 服务器检索密码，并在 JBoss EAP 中进行验证。如果启用了，JBoss EAP 服务器必须能够从客户端获取纯文本用户密码，这需要使用 **PLAIN** SASL 或 **BASIC** HTTP 机制进行身份验证。默认值为 **false**。
hash-charset	在将密码字符串转换为字节数组时会使用字符集。默认为 UTF-8。
hash-encoding	没有以纯文本形式存储的密码的字符串格式。它可以是以下之一： • base64 • hex 默认为 base64。
identity-mapping	定义主体如何是映射到底层 LDAP 服务器中对应条目的配置选项。

表 5.13. ldap-realm identity-mapping 属性

属性	描述
attribute-mapping	为这个资源定义的属性映射列表。
filter-name	用于按名称获取身份的 LDAP 过滤器。
iterator-filter	用于迭代域身份的 LDAP 过滤器。
new-identity-attributes	新创建的身份的属性的列表。它是域的可修改性所必需的。这是**名称**和**值**对对象的列表。
new-identity-parent-dn	新创建的身份的父的 DN。域的可修改性所需要的。
otp-credential-mapper	OTP 凭证的凭证映射。
rdn-identifier	用于从 LDAP 条目获取主体名称的主体 DN 的 RDN 部分。这在创建新身份时也使用它。
search-base-dn	用于搜索身份的基础 DN。
use-recursive-search	如果此属性设为 **true**，则身份搜索查询是递归的。默认值为 **false**。
user-password-mapper	凭证的凭证映射，类似于 userPassword。

属性	描述
x509-credential-mapper	使用 LDAP 作为 X509 凭证存储的配置。如果未定义 **-from** 子属性，则此配置将被忽略。如果定义了多个 **-from** 子属性，则用户证书必须与所有定义的标准匹配。

表 5.14. ldap-realm identity-mapping attribute-mapping 属性

属性	描述
extract-rdn	在原始格式的值为 X.500 格式的情况下，用作属性值的 RDN 键。
filter	用于获取特定属性的值的过滤器。字符串 **{0}** 将被 username 替换，'{1}' 将被用户身份 DN 替换。
filter-base-dn	应该在其中执行过滤器的上下文的名称。
from	映射到身份属性的 LDAP 属性的名称。如果没有定义，则使用条目的 DN。
reference	包含要从中获取值的 DN 的 LDAP 属性的名称。
role-recursion	递归角色分配的最大深度。使用 **0** 来指定递归。默认为 **0**。
role-recursion-name	确定角色条目的 LDAP 属性，在搜索角色角色时，在 **filter-name** 中替换 "{0}"。
search- recursive	如果 **true** 属性 LDAP 搜索查询为递归。默认值为**true**。
to	从特定的 LDAP 属性映射的身份属性的名称。如果没有提供，则属性的名称与 **from** 中定义的属性名称相同。如果 **from** 未定义，则使用 **dn** 值。

表 5.15. ldap-realm identity-mapping user-password-mapper 属性

属性	描述
from	映射到身份属性的 LDAP 属性的名称。如果没有定义，则使用条目的 DN。
verifiable	如果 **true** 密码可用来验证用户。默认值为 **true**。
writable	如果更改了 **true** 密码。默认值为 **false**。

表 5.16. ldap-realm identity-mapping otp-credential-mapper 属性

属性	描述
algorithm-from	OTP 算法的 LDAP 属性的名称。
hash-from	OTP 哈希函数的 LDAP 属性的名称。
seed-from	OTP 种子 的 LDAP 属性的名称。
sequence-from	OTP 序列号的 LDAP 属性的名称。

表 5.17. ldap-realm identity-mapping x509-credential-mapper 属性

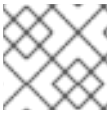
属性	描述
certificate-from	映射到编码的用户证书的 LDAP 属性的名称。如果没有定义，则不会检查编码的证书。
digest-algorithm	摘要算法，它是哈希函数，用于计算用户证书的摘要。只有定义了 **digest-from**，才会使用它。
digest-from	映射到用户证书摘要的 LDAP 属性的名称。如果没有定义，则不会检查证书摘要。
serial-number-from	映射到用户证书序列号的 LDAP 属性的名称。如果没有定义，则不会检查序列号。
subject-dn-from	映射到用户证书的主题 DN 的 LDAP 属性的名称。如果未定义，则不会检查主题 DN。

5.9. PASSWORD MAPPER 属性

密码映射器使用以下算法类型之一从数据库中的多个字段构造密码：

- 明文
- 简单摘要
- salt 简单摘要
- bcrypt
- SCRAM
- 模块加密

密码映射器具有以下属性：



注意

对于所有映射器，第一列的索引为 **1**。

表 5.18. password mapper 属性

映射器名称	属性	加密方法
clear-password-mapper	• **password-index** 包含明文密码的列的索引。	无加密。
simple-digest	• **password-index** 包含密码哈希的列的索引。 • **algorithm** 使用的哈希算法。支持以下值： ◦ **simple-digest-md2** ◦ **simple-digest-md5** ◦ **simple-digest-sha-1** ◦ **simple-digest-sha-256** ◦ **simple-digest-sha-384** ◦ **simple-digest-sha-512** • **hash-encoding** 指定表示哈希。允许的值： ◦ **base64**（默认） ◦ **hex**	使用简单哈希机制。

映射器名称	属性	加密方法
salted-simple-digest	• **password-index** 包含密码哈希的列的索引。 • **algorithm** 使用的哈希算法。支持以下值： ◦ **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** 包含用于哈希的 salt 的列的索引。 • **hash-encoding** 指定哈希的表示。允许的值： ◦ **base64**（默认） ◦ **hex** • **salt-encoding** 指定 salt 的表示。允许的值： ◦ **base64**（默认） ◦ **hex**	简单的散列机制与 salt 一起使用。

映射器名称	属性	加密方法
bcrypt-password-mapper	• **password-index** 包含密码哈希的列的索引。 • **salt-index** 包含用于哈希的 salt 的列的索引。 • **iteration-count-index** 包含使用的迭代数的列的索引。 • **hash-encoding** 指定哈希的表示。允许的值： ◦ **base64**（默认） ◦ **hex** • **salt-encoding** 指定 salt 的表示。允许的值： ◦ **base64**（默认） ◦ **hex**	用于哈希的 Blowfish 算法。
scram-mapper	• **password-index** 包含密码哈希的列的索引。 • **algorithm** 使用的哈希算法。支持以下值： ◦ **scram-sha-1** ◦ **scram-sha-256** ◦ **scram-sha-384** ◦ **scram-sha-512** • **salt-index** 包含 salt 的列的索引用于哈希。 • **iteration-count-index** 包含使用的迭代数的列的索引。 • **hash-encoding** 指定哈希的表示。允许的值： ◦ **base64**（默认） ◦ **hex** • **salt-encoding** 指定 salt 的表示。允许的值： ◦ **base64**（默认） ◦ **hex**	salt 挑战响应验证机制用于哈希。

映射器名称	属性	加密方法
modular-crypt-mapper	• **password-index** 包含加密密码的列的索引。	模块加密编码支持多个信息片段在单个字符串中编码。该信息可包括以下内容： • 密码类型 • hash 或 digest • salt • 迭代计数

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

表 5.19. periodic-rotating-file-audit-log 属性

属性	描述
Autoflush	指定输出流是否在每个审计事件后清除。如果没有定义属性，则 **同步** 的属性值是默认值。
编码	指定审计文件编码。默认值为 **UTF-8**。可能的值如下： • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**
格式	使用 **SIMPLE** 进行人类可读的文本格式或 **JSON**，以 **JSON** 形式存储各个事件。
path	定义日志文件的位置。
relative-to	可选属性。定义日志文件的位置。
suffix	可选属性。在轮转的日志中添加日期后缀。您必须使用 **java.time.format.DateTimeFormatter** 格式。例如 **.yyyy-MM-dd**。

属性	描述
synchronized	默认值为 **true**。指定文件描述符在每次审计事件后同步。

5.11. PROPERTIES-REALM 属性

您可以通过设置其属性来配置 **properties-realm**。

表 5.20. properties-realm 属性

属性	描述
groups-attribute	返回的 **AuthorizationIdentity** 中的属性名称，其应包含身份的组成员信息。
groups-properties	包含用户及其组的属性文件。
hash-charset	指定在将客户端提供的密码字符串转换为哈希计算的字节数组时要使用的字符集的名称。默认设置为 **UTF-8**。
hash-encoding	如果密码没有以纯文本形式存储，则指定哈希密码的字符串格式。它可以指定以下两种之一：**hex** 或 **base64**。对于 **properties-realm**，默认设置为 **hex**。
users-properties	包含用户及其密码的属性文件。

表 5.21. properties-realm users-properties 属性

属性	描述
digest-realm-name	如果未在属性文件中发现，用于摘要密码的默认域名称。
path	包含用户及其密码的文件的路径。该文件应包含 realm 名称声明。
plain-text	如果为 **true**，则属性文件中的密码以纯文本形式存储。如果为 **false**，它们被预哈希，格式为 **HEX (MD5 (username ':' realm ':' password))**。默认值为 **false**。
relative-to	路径相对的预定义的路径。

表 5.22. properties-realm groups-properties 属性

属性	描述
path	包含用户及其组的文件的路径。

属性	描述
relative-to	路径相对的预定义的路径。

5.12. SASL-AUTHENTICATION-FACTORY 属性

您可以通过设置其属性来配置 **sasl-authentication-factory**。

表 5.23. sasl-authentication-factory 属性

属性	描述
mechanism-configurations	特定于机制的配置列表。
sasl-server-factory	SASL 服务器与这个资源关联。
security-domain	与此资源关联的安全域。

表 5.24. sasl-authentication-factory mechanism-configurations 属性

属性	描述
credential-security-factory	用于获取机制所要求的凭证的安全因素。
final-principal-transformer	应用于此机制域的最后一个主体转换器。
host-name	此配置应用到的主机名。
mechanism-name	此配置仅应用于使用指定名称机制的情况。如果省略此属性，则它将匹配任何机制名称。
mechanism-realm-configurations	机制理解的域名称的定义的列表。
protocol	此配置适用的协议。
post-realm-principal-transformer	选择域之后要应用的主体转换器。
pre-realm-principal-transformer	在选择域之前要应用的主体转换器。
realm-mapper	机制使用的 realm mapper。

表 5.25. sasl-authentication-factory mechanism-configurations mechanism-realm-configurations 属性

属性	描述
final-principal-transformer	应用于此机制域的最后一个主体转换器。
post-realm-principal-transformer	选择域之后要应用的主体转换器。
pre-realm-principal-transformer	在选择域之前要应用的主体转换器。
realm-mapper	机制使用的 realm mapper。
realm-name	按机制要显示的域的名称。

5.13. SECRET-KEY-CREDENTIAL-STORE 属性

您可以通过设置其属性来配置 **secret-key-credential-store**。

表 5.26. secret-key-credential-store 属性

属性	描述
create	如果您不希望 Elytron 在其不存在时创建它，请将该值设为 **false**。默认为 **true**。
default-alias	默认生成的密钥的别名名称。默认值为 **key**。
key-size	生成密钥的大小。默认大小为 256 位。您可以将值设为以下之一： • **128** • **192** • **256**
path	凭证存储的路径。
populate	如果凭证存储不包含 **default-alias**，此属性表示 Elytron 是否应该创建一个。默认值是 **true**。
relative-to	对之前定义的属性 **path** 相对的路径的引用。

5.14. SECURITY-DOMAIN 属性

您可以通过设置其属性来配置 **security-domain**。

属性	描述
default-realm	此安全域中包含的默认域。
evidence-decoder	对此域所使用的 EvidenceDecoder 的引用。
outflow-anonymous	此属性指定在无法流向安全域时应使用匿名身份，这在以下情况下发生： • 要排除此域的域不信任这个域。 • 那个域中不存在将流向域的身份 出版匿名身份会清除之前为该域建立的任何身份。
outflow-security-domains	来自此域的安全身份应自动流向的安全域的列表。
permission-mapper	对此域使用的 PermissionMapper 的引用。
post-realm-principal-transformer	在域对提供的身份名称执行操作后，要应用的主体转换器的引用。
pre-realm-principal-transformer	在选择域之前对要应用的主体转换器的引用。
principal-decoder	对此域要使用的 PrincipalDecoder 的引用。
realm-mapper	对此域要使用的 RealmMapper 的引用。
realms	此安全域包含的域的列表。
role-decoder	对此域要使用的 RoleDecoder 的引用。
role-mapper	对此域要使用的 RoleMapper 的引用。
security-event-listener	对安全事件的监听程序的引用。
trusted-security-domains	此安全域信任的安全域的列表。
trusted-virtual-security-domains	此安全域信任的虚拟安全域的列表。

5.15. SIMPLE-ROLE-DECODER 属性

您可以通过设置其属性来配置简单的角色解码器。

表 5.27. simple-role-decoder 属性

属性	描述
attribute	身份直接映射到角色的属性的名称。

5.16. SIZE-ROTATING-FILE-AUDIT-LOG 属性

表 5.28. size-rotating-file-audit-log 属性

属性	描述
Autoflush	指定输出流是否在每个审计事件后清除。如果没有定义属性，则 **同步** 的属性值是默认值。
编码	指定审计文件编码。默认值为 **UTF-8**。可能的值如下： • **UTF-8** • **UTF-16BE** • **UTF-16LE** • **UTF-16** • **US-ASCII** • **ISO-8859-1**
格式	默认值为 **SIMPLE**。使用 **SIMPLE** 进行人类可读的文本格式或 **JSON**，以 **JSON** 形式存储各个事件。
max-backup-index	轮转时要备份的最大文件数。默认值为：**1**。
path	定义日志文件的位置。
relative-to	可选属性。定义日志文件的位置。
rotate-on-boot	默认情况下，当您重新启动服务器时，Elytron 不会创建新的日志文件。将此属性设置为 **true** 以轮转服务器重启时的日志。
rotate-size	日志文件在 Elytron 轮转日志前可以达到的最大大小。默认值为 **10m**，*10* MB。您还可以使用 k、g、b 或 t 单元定义日志的最大大小。您可以使用大写或小写字符指定单位。
suffix	可选属性。在轮转的日志中添加日期后缀。您必须使用 **java.text.format.DateTimeFormatter** 格式。例如 **.yyyy-MM-dd-HH**。

属性	描述
synchronized	默认值为 **true**。指定文件描述符在每次审计事件后同步。

5.17. SYSLOG-AUDIT-LOG ATTRIBUTES

表 5.29. syslog-audit-log attributes

属性	描述
格式	记录审计事件的格式。 支持的值： • **JSON** • **SIMPLE** 默认值： • **SIMPLE**
host-name	要嵌入到发送到 syslog 服务器的所有事件的主机名。
port	**syslog** 服务器上的监听端口。
reconnect-attempts	Elytron 在关闭连接前尝试向 **syslog** 服务器发送连续消息的次数上限。此属性的值仅在使用传输协议是 UDP 时有效。 支持的值： • 任何 **正整数** 值。 • **-1** 表示无限重新连接尝试。 默认值： • **0**
server-address	**syslog** 服务器的 IP 地址或名称，可由 Java 的 **InetAddress.getByName （）** 方法解析。
ssl-context	连接到 **syslog** 服务器时要使用的 SSL 上下文。只有在 **传输** 设置为 **SSL_TCP** 时，才需要此属性。

属性	描述
syslog-format	用于描述审计事件的 RFC 格式。 支持的值： • **RFC3164** • **RFC5424** 默认值： • **RFC5424**
传输	用于连接到 **syslog** 服务器的传输层协议。 支持的值： • **SSL_TCP** • **TCP** • **UDP** 默认值： • **TCP**