Red Hat JBoss Enterprise Application Platform 8.0

使用多个身份存储保护应用程序和管理界面

通过组合多个身份存储（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 管理界面和部署的应用程序的指南

Last Updated: 2024-02-09

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

通过组合多个身份存储（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 管理界面和部署的应用程序的指南

法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

摘要

通过组合多个身份存储（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. 创建聚合域

1.1.1. Elytron 中的聚合域

有了聚合域 **aggregate-realm**，您可以对身份验证以及另一个安全域使用安全域，或对 Elytron 中的授权使用多个安全域的聚合。

例如，您可以将 **aggregate-realm** 配置为对身份验证使用 **ldap-realm**，对授权使用 **filesystem-realm** 和 **ldap-realm** 的聚合。

在配置了多个授权域的聚合域中创建的一个身份，如下所示：

- 来自每个授权域的属性值被加载。
- 如果属性在多个授权域中定义，则将使用该属性第一次出现的值。

以下示例演示了当多个授权域包含同一身份属性的定义时身份是如何创建的。

聚合域配置示例

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

在示例中，配置的 **aggregate-realm** 引用两个现有的安全域："exampleLDAPRealm"，它是一个 LDAP 域，以及"exampleFilesystemRealm"，它是文件系统域。

- 从 LDAP 域中获取的属性值：

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

- 从文件系统域中获取的属性值：

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

从聚合域中获取生成的身份：

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

示例 **aggregate-realm** 使用 LDAP 域中定义的属性 **mail** 的值，因为 LDAP 域在文件系统域之前被引用。

其他资源

- 在 Elytron 中创建一个 **aggregate-realm**

1.1.2. 创建聚合域所需的安全域的示例

以下示例演示了创建 **ldap-realm** 和 **filesystem-realm**。您可以在 **aggregate-realm** 中引用这些安全域。

1.1.2.1. 在 Elytron 示例中创建一个 ldap-realm

创建由轻量级目录访问协议(LDAP)身份存储支持的 Elytron 安全域，来保护 JBoss EAP 服务器接口或在服务器上部署的应用程序。

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: 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
```

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。

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

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=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"}
```

现在，您可以使用这个域来创建一个安全域，或与 **failover-realm**、**distributed-realm** 或 **aggregate-realm** 中的另一个域合并。

1.1.2.2. 在 Elytron 示例中创建一个 filesystem-realm

创建一个由基于文件系统的身份存储支持的 Elytron 安全域，来保护 JBoss EAP 服务器接口或在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=exampleFileSystemRealm: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=exampleFileSystemRealm:add-
      identity(identity=user1)
      {"outcome" => "success"}
      ```

   b. 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=exampleFileSystemRealm:add-identity-
      attribute(identity=user1, name=Roles, value=["Admin","Guest"])
      {"outcome" => "success"}
      ```

   c. 设置用户的属性。

      语法

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

      Example

      ```
      /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/"])
```

现在，您可以使用这个域来创建一个安全域，或与 **failover-realm**、**distributed-realm** 或 **aggregate-realm** 中的另一个域合并。

1.1.3. 在 Elytron 中创建一个 aggregate-realm

在 Elytron 中创建一个 **aggregate-realm**，它使用一个安全域进行身份验证，使用多个安全域的聚合进行授权。使用 **aggregate-realm** 创建一个安全域来添加身份验证和授权，以管理界面和部署的应用程序。

先决条件

- JBoss EAP 正在运行。
- 您已创建了一个要从聚合域中引用的域。

流程

1. 从现有的安全域创建一个 **aggregate-realm**。

   语法

   ```
   /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_rea
   lm_for_authorization_N>])
   ```

   Example

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

2. 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)
   {"outcome" => "success"}
   ```

3. 创建一个引用 **aggregate-realm** 和角色解码器的安全域。

   语法

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

Example

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

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

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

1.2. 创建缓存域

1.2.1. Elytron 中的缓存域

Elytron 提供 **caching-realm** 来从缓存安全域中查找的凭证的结果。**caching-realm** 使用 *LRU* 或 *Least Recently Used* 缓存策略缓存 **PasswordCredential** 凭证，在达到最大条目数时，将丢弃最少访问的条目。

您可以使用带有以下安全域的 **caching-realm** ：

- **filesystem-realm**
- **jdbc-realm**
- **ldap-realm**
- 自定义安全域

如果您在 JBoss EAP 之外更改凭证源，如果底层的安全域支持侦听，则这些更改只会传播到 JBoss EAP 缓存域。只有 **ldap-realm** 支持侦听。但是，过滤的属性，如 **roles**，在 **ldap-realm** 中不支持侦听。

要确保您的缓存域有正确的用户数据的缓存，请确保以下内容：

- 在修改了凭证源处的用户属性后，请清除 **caching-realm** 缓存。
- 通过缓存域而不是在凭证源处修改用户属性。

重要

通过缓存域进行用户更改仅作为技术预览提供。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，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。

其他资源

- 在 Elytron 中创建一个 **cache-realm**
- **cache-realm** 属性
- 清除 **caching-realm** 缓存

1.2.2. 在 Elytron 中创建一个 cache-realm

创建一个 **cache-realm** 以及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。

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 。

先决条件

- 您已将安全域配置为缓存。

流程

1. 创建一个将安全域引用为缓存的 **caching-realm**。

   语法

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

   Example

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

2. 创建一个引用 **caching-realm** 的安全域。

   语法

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

   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 是否可以将 **caching-realm** 中引用的安全域中的数据加载到 **caching-realm**，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"]
      }
  }
  ```

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

- **cache-realm** 属性
- 清除 **caching-realm** 缓存
- **security-domain** 属性

1.2.3. 清除 caching-realm 缓存

清除 **caching-realm** 缓存会强制 Elytron 使用安全域的最新数据重新填充缓存，Elytron 被配置为缓存这些数据。

先决条件

- **caching-realm** 已配置。

流程

- 清除 **caching-realm** 缓存。

  语法

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

示例

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

其他资源

- **cache-realm** 属性

1.3. 创建分布式域

1.3.1. Elytron 中的分布式域

有了分布式域，您可以通过引用现有的安全域来跨不同的身份存储进行搜索。获取的身份用于身份验证和授权。Elytron 在分布式域中跟俊您在 **distributed-realm** 资源中定义的顺序调用安全域。

distributed-realm 配置示例

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

在示例中，配置的 **distributed-realm** 引用两个现有的安全域："exampleLDAPRealm"，它是一个 LDAP 域，以及"exampleFilesystemRealm"，它是一个文件系统域。Elytron 按如下顺序搜索引用的安全域：

- Elytron 首先搜索 LDAP 域以匹配身份。
- 如果 Elytron 找到匹配项，则身份验证会成功。
- 如果 Elytron 没有找到匹配项，它会搜索文件系统域。

默认情况下，如果在匹配身份前与任何身份存储的连接失败，则身份验证会失败，但有一个例外的 **RealmUnavailableException**，且不会搜索更多域。您可以通过将属性 **ignore-unavailable-realms** 设置为 **true** 来更改此行为。当 **ignore-unavailable-realms** 设为 **true** 时与身份存储的连接失败，Elytron 会继续搜索剩余的域。

当 **ignore-unavailable-realms** 设置为 **true** 时，**emit-events** 默认被设置为 **true**，因此当任何查询的域不可用时会发出 **SecurityEvent**。您可以通过将 **emit-events** 设置为 **false** 来关闭此关闭。

其他资源

- 在 Elytron 中创建一个 **distributed-realm**

1.3.2. 分布式域所需的创建安全域的示例

以下示例演示了创建 **ldap-realm** 和 **filesystem-realm**。您可以在 **distributed-realm** 中引用这些安全域。

1.3.2.1. 在 Elytron 示例中创建一个 ldap-realm

创建由轻量级目录访问协议(LDAP)身份存储支持的 Elytron 安全域，来保护 JBoss EAP 服务器接口或在服务器上部署的应用程序。

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。

```
/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",c
redential-reference={clear-text="secret"})
{"outcome" => "success"}
```

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_attr
ibute_name>",to="<identity_attribute_name>"}]})
```

Example

```
/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"}
```

现在，您可以使用这个域来创建安全域，或与 **failover-realm**、**distributed-realm** 或 **aggregate-realm** 中的其它域合并。您还可以为 **ldap-realm** 配置一个 **caching-realm** 来缓存查找的结果，并提高性能。

1.3.2.2. 在 Elytron 示例中创建一个 filesystem-realm

创建一个由基于文件系统的身份存储支持的 Elytron 安全域，来保护 JBoss EAP 服务器接口或在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=exampleFileSystemRealm: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=exampleFileSystemRealm:add-identity(identity=user2)
   {"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=exampleFileSystemRealm:set-password(identity=user2, clear={password="passwordUser2"})
   {"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=exampleFileSystemRealm:add-identity-attribute(identity=user2, name=Roles, value=["Admin","Guest"])
   {"outcome" => "success"}
   ```

现在，您可以使用这个域来创建一个安全域，或与 **failover-realm**、**distributed-realm** 或 **aggregate-realm** 中的另一个域合并。

1.3.3. 在 Elytron 中创建一个 distributed-realm

在 Elytron 中创建一个 **distributed-realm**，以引用现有的安全域来搜索身份。使用 **distributed-realm** 创建一个安全域，来向管理界面或部署在服务器上的应用程序添加身份验证和授权。

先决条件

- JBoss EAP 正在运行。
- 您已创建了要在 **distributed-realm** 中引用的域。

流程

1. 创建一个引用现有安全域的 **distributed-realm**。

   语法

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

   Example

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

2. 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)
   {"outcome" => "success"}
   ```

3. 创建一个引用 **distributed-realm** 的安全域，以及角色解码器。

   语法

   ```
   /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>)
   ```

   Example

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

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

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

1.4. 创建故障转移域

1.4.1. Elytron 中的故障转移域

您可以在 Elytron 中配置故障转移安全域 **failover-realm**，其引用两个现有的安全域，以便在一个安全域失败时，Elytron 使用另一个作为备份。

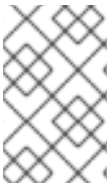
Elytron 中的 **failover-realm** 引用两个安全域：

- **delegate-realm** ：要使用的主安全域。
- **failover-realm** ：要用作备份的安全域。

failover-realm 配置示例

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

在示例中，**exampleLDAPRealm** 是一个 **ldap-realm**，用作委派域， **exampleFileSystemRealm** 是一个 **filesystem-realm**，用作 **failover-realm**。如果 **ldap-realm** 失败，Elytron 将使用 **filesystem-realm** 进行身份验证和授权。



注意

在 **failover-realm** 中，只有当 **delegate-realm** 失败时才会调用 **failover-realm**。如果与 **delegate-realm** 的连接成功但没找到所需的身份，则不会调用 **fail-over** 域。要搜索跨多个安全域的身份，请使用 **distributed-realm**。

1.4.2. 创建故障转移域所需的安全域的示例

以下示例演示了创建 **ldap-realm** 和 **filesystem-realm**。您可以在 **failover-realm** 中引用这些安全域。

1.4.2.1. 在 Elytron 示例中创建一个 ldap-realm

创建由轻量级目录访问协议(LDAP)身份存储支持的 Elytron 安全域，来保护 JBoss EAP 服务器接口或在服务器上部署的应用程序。

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。

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

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_attr
ibute_name>",to="<identity_attribute_name>"}]})
```

Example

```
/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"}
```

现在，您可以使用这个域来创建安全域，或与 **failover-realm**、**distributed-realm** 或 **aggregate-realm** 中的其它域合并。您还可以为 **ldap-realm** 配置一个 **caching-realm** 来缓存查找的结果，并提高性能。

1.4.2.2. 在 Elytron 示例中创建一个 filesystem-realm

创建一个由基于文件系统的身份存储支持的 Elytron 安全域，来保护 JBoss EAP 服务器接口或在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=exampleFileSystemRealm: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=exampleFileSystemRealm: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=exampleFileSystemRealm: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=exampleFileSystemRealm:add-identity-
attribute(identity=user1, name=Roles, value=["Admin","Guest"])
{"outcome" => "success"}
```

现在，您可以使用这个域来创建一个安全域，或与 **failover-realm**、**distributed-realm** 或 **aggregate-realm** 中的另一个域合并。

1.4.3. 在 Elytron 中创建一个 failover-realm

在 Elytron 中创建一个故障转移安全域，它将现有安全域引用为委派域、要使用的默认域以及故障转移域。Elytron 在委派域失败时使用配置的故障转移域。使用安全域创建安全域，来向管理接口或服务器上部署的应用程序添加身份验证和授权。

先决条件

- JBoss EAP 正在运行。
- 您已创建了域来用作委派和故障转移域。

流程

1. 从现有的安全域创建一个 **failover-realm**。

   语法

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

   Example

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

2. 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)
   {"outcome" => "success"}
   ```

3. 创建一个引用 **failover-realm** 和角色解码器的安全域。

   语法

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

   Example

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

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

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

1.5. 创建 JAAS 域

1.5.1. Elytron 中的 JAAS 域

Java 认证和授权服务(JAAS)域 **jaas-realm** 是一个安全域，您可以用来在 **elytron** 子系统中为用户的凭证验证和分配用户角色配置自定义登录模块。

您可以使用 **jaas-realm** 来保护 JBoss EAP 管理界面和已部署的应用。

JAAS 域通过初始化 **javax.security.auth.login.LoginContext** （其使用 JAAS 配置文件中指定的登录模块）来验证用户凭据，。

登录模块是 **javax.security.auth.login.LoginContext.LoginModule** 接口的实现。将这些实现作为 JBoss EAP 模块添加到服务器中，并在 JAAS 配置文件中指定它们。

JAAS 配置文件示例

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

1. 配置 **jaas-realm** 时使用的条目的名称。
2. 使用其可选标记登录模块。您可以使用 JAAS 定义的所有标志。如需更多信息，请参阅 Oracle Java SE 文档中的 JAAS 登录配置文件。
3. 使用其可选标志和选项登录模块。

在登录模块中与属性映射和角色关联的主题的主体
您可以使用 *主题* 的主体将属性添加到从登录模块获取的身份中。*主题* 是正被身份验证的用户，主体是标识符，如用户名，包含在主体内。

Elytron 获取和映射身份，如下所示：

- 登录模块使用 **javax.security.auth.Subject** 代表用户，正被身份验证的 *主题*。
- *主题* 可以有多个 **java.security.Principal** 实例 *principal* 与之关联。
- Elytron 使用 **org.wildfly.security.auth.server.SecurityIdentity** 来代表经过身份验证的用户。Elytron 将 *主题* 映射到**SecurityIdentity** 。

主题的 *主体* 被映射到具有以下规则的安全身份的属性：

- 属性的 **key** 是 *主体* 的简单类名称，可通过 **principal.getClass().getSimpleName()** 调用获得。
- **value** 是 *主体* 的名称，可通过 **principal.getName()** 调用获得。
- 对于同一类型的 *主体*，这些值会附加到属性键下的集合中。

其他资源

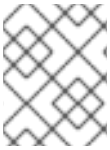
- 开发自定义 JAAS 登录模块

- 在 Elytron 中创建一个 **jaas-realm**
- **主题** 类 Javadocs
- **主体** 类 Javadocs

1.5.2. 开发自定义 JAAS 登录模块

您可以创建自定义 Java 身份验证和授权服务(JAAS)登录模块来实现自定义身份验证和授权功能。

您可以通过 Elytron 子系统中的 **jaas-realm**，使界面和部署的应用程序。登录模块不是部署的一部分，您可以将它们作为 JBoss EAP 模块包含在内。



注意

以下流程仅作为示例提供。如果您已有要保护的应用程序，则可以跳过这些步骤，直接来到 向应用程序添加身份验证和授权。

1.5.2.1. 为 JAAS 登录模块开发创建一个 Maven 项目

要创建自定义 Java 身份验证和授权服务(JAAS)登录模块，请创建一个具有所需依赖项和目录结构的 Maven 项目。

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. 在 CLI 中使用 **mvn** 命令来建立一个 Maven 项目。此命令为项目创建目录结构，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-simple \
   -DinteractiveMode=false
   ```

   Example

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

2. 导航到应用程序根目录。

   语法

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

Example

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

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>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. 删除目录 **site** 和 **test**，因为本例不需要它们。

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

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: 1.404 s
[INFO] Finished at: 2022-04-28T13:55:18+05:30
[INFO] ------------------------------------------------------------------------
```

现在，您可以创建自定义的 JAAS 登录模块。

1.5.2.2. 创建自定义的 JAAS 登录模块

通过创建实现 **javax.security.auth.spi.LoginModule** 接口的类来创建自定义的 Java 身份验证和授权服务(JAAS)登录模块。此外，为自定义登录模块创建具有志和选项的 JAAS 配置文件。

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 JAAS 登录模块开发创建一个 Maven 项目 。

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/loginmodule
   ```

2. 导航到包含源文件的目录。

   语法

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

   Example

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

3. 删除生成的文件 **App.java**。

   ```
   $ rm App.java
   ```

4. 为自定义登录模块源创建一个文件 **ExampleCustomLoginModule.java**。

   ```
   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. 在 *<application_home>* 目录中创建 JAAS 配置文件 **JAAS-login-modules.conf**。

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

- **exampleConfiguration** 是条目名称。
- **com.example.loginmodule.ExampleCustomLoginModule** 是登录模块。
- **optional** 是标志。

6. 编译登录模块。

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

现在，您可以使用登录模块来保护 JBoss EAP 管理界面和部署的应用程序。

1.5.3. 在 Elytron 中创建一个 jaas-realm

创建由 Java 认证和授权服务(JAAS)兼容自定义登录模块支持的 Elytron 安全域，来保护 JBoss EAP 服务器界面或部署的应用程序。使用安全域来创建安全域。

先决条件

- 您已将自定义登录模块打包为 JAR。
  如需示例登录模块，请参阅 开发自定义的 JAAS 登录模块。
- JBoss EAP 正在运行。

流程

1. 使用管理 CLI，将登录模块 JAR 作为模块添加到 JBoss EAP。

   语法

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

Example

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

2. 从登录模块和 JAAS 登录配置文件创建 **jaas-realm**。

语法

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

Example

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

3. 创建一个引用 **jaas-realm** 的安全域。

语法

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

Example

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

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

- **module** 命令参数
- **jaas-realm** 属性
- **security-domain** 属性

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. 为 aggregate-realm 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。如果您已有要保护的应用程序，则可以跳过这些步骤，直接来到 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，请创建一个具有所需依赖项和目录结构的 Maven 项目。

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>
  </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>
```

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和属性中获取的用户名。如果没有登录的用户，servlet 将返回文本"NO AUTHENTICATED USER"。

先决条件

- 您已创建了一个 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 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. 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。

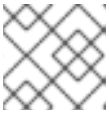
   语法

   ```
   <!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 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。

3.1. 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。

3.2. 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 **aggregate-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

3.3. 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 **aggregate-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

3.4. 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 **aggregate-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

3.5. 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 **aggregate-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

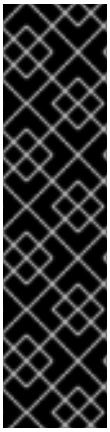
   ```
   /subsystem=elytron/security-domain=exampleSecurityDomain:write-attribute(name=security-event-listener,value=exampleSyslog)
   ```

其他资源

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

3.6. 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 **aggregate-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

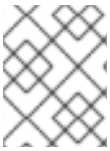
第 4 章 参考

4.1. AGGREGATE-REALM 属性

您可以通过设置其属性来配置 **aggregate-realm**。

表 4.1. aggregate-realm 属性

属性	描述
authentication-realm	用于身份验证步骤的安全域的引用。这用于获取或验证凭证。
authorization-realm	用于加载授权步骤身份的安全域的引用。
authorization-realms	引用安全领域，以聚合加载授权步骤的身份。如果属性在多个授权域中定义，则将使用该属性第一次出现的值。
principal-transformer	引用主体转换器，以在加载身份验证的身份和加载授权的身份之间应用。



注意

authorization-realm 和 **authorization-realms** 属性是互斥的。在域中仅定义两个属性中的一个。

4.2. CACHE-REALM 属性

您可以通过设置其属性来配置 **caching-realm**。

表 4.2. cache-realm 属性

属性	描述
maximum-age	项目可以保留在缓存中的时间（毫秒）。**-1** 代表无限期保留项目。默认值为 **-1**。
maximum-entries	在缓存中保留的最大条目数。默认值为 **16**。
realm	对可缓存的安全域的引用，如 **jdbc-realm**、**ldap-realm**、**system-realm** 或自定义安全域。

4.3. DISTRIBUTED-REALM 属性

您可以通过设置其属性来配置 **distributed-realm**。

表 4.3. distributed-realm 属性

属性	描述
emit-events	是否应发出 **SecurityEvent** 表示域不可用。仅在 **ignore-unavailable-realms** 属性设置为 **true** 时才适用。默认值为 **true**。
ignore-unavailable-realms	如果与任何身份存储的连接失败，则是否应检查后续域。将值设为 **true** 以检查后续域。默认值为 **false**。 当值设为 **true** 时，如果与任何身份存储的连接失败，则会发出 **SecurityEvent**。
realms	要搜索的安全域的列表。安全域按照其在此属性中提供的顺序调用。

4.4. FAILOVER-REALM 属性

您可以通过设置其属性来配置 **failover-realm**。

表 4.4. failover-realm 属性

属性	描述
delegate-realm	默认要使用的安全域。
emit-events	指定是否应发出表示 **delegate-realm** 不可用的 **SecurityEvent** 类型的安全事件。启用后，您可以在审计日志中捕获这些事件。默认值为 **true**。
failover-realm	在 **delegate-realm** 不可用时要使用的安全域。

4.5. FILE-AUDIT-LOG 属性

表 4.5. 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。

属性	描述
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。

4.6. HTTP-AUTHENTICATION-FACTORY 属性

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

表 4.6. 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 4.7. 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。

属性	描述
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 4.8. 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。

4.7. JAAS-REALM 属性

您可以通过设置其属性来配置 **jaas-realm**。除 **entry** 外的所有属性都是可选的。

表 4.9. jaas-realm 属性

attribute	description
callback-handler	与登录上下文一起使用的回调处理程序。可以使用安全属性 **auth.login.defaultCallbackHandler** 替代。如果未定义，则使用域的默认回调处理程序。
entry	用于初始化 **LoginContext** 的条目名称。

attribute	description
module	带有自定义 **LoginModules** 和 **CallbackHandler** 类的模块。
path	JAAS 配置文件的可选路径。您还可以使用 java 系统属性 **java.security.auth.login.config** 或 java 安全属性 **login.config.url** 指定位置。
relative-to	如果您提供了 **relative-to**，则 **path** 属性的值将被视为相对于此属性指定的路径。

4.8. MODULE 命令参数

您可以在 **module** 命令中使用不同的参数。

表 4.10. module 命令参数

参数	描述
--absolute-resources	使用这个参数指定要从 **module.xml** 文件引用的绝对文件系统路径的列表。指定的文件没有复制到模块目录中。 有关分隔符的详情，请参阅 **--resource-delimiter**。
--allow-nonexistent-resources	使用此参数为 **--resources** 指定的不存在的资源创建空目录。如果存在不存在的资源且没有使用此参数，则**module add** 命令将失败。
--dependencies	使用此参数提供此模块依赖的逗号分隔的模块名称的列表。
--export-dependencies	使用此参数指定导出的依赖项。 `module add --name=com.mysql --resources=/path/to/{MySQLDriverJarName} --export-dependencies=wildflyee.api,java.se`
--main-class	使用此参数指定声明模块主方法的完全限定类名称。
--module-root-dir	如果您定义了要使用的外部 JBoss EAP 模块目录，而不是默认的 ***EAP_HOME*/modules/** 目录，则使用此参数。 `module add --module-root-dir=/path/to/my-external-modules/ --name=com.mysql --resources=/path/to/{MySQLDriverJarName} --dependencies=wildflyee.api,java.se`
--module-xml	使用此参数为此新模块提供 **module.xml** 的文件系统路径。此文件被复制到模块目录中。如果没有指定此参数，则会在模块目录中生成一个 **module.xml** 文件。

参数	描述
--name	使用此参数提供要添加的模块的名称。此参数是必需的。
--properties	使用此参数提供逗号分隔的定义模块属性的 ***PROPERTY_NAME=PROPERTY_VALUE*** 对的列表。
--resource-delimiter	使用此参数为提供给 **--resources** 或 **absolute-resources** 参数的资源列表设置用户定义的文件路径分隔符。如果没有设置，则文件路径分隔符是 Linux 的冒号(**:**) 和 Windows 的分号(**;**)。
--resources	使用此参数通过提供文件系统路径列表来指定此模块的资源。这些文件被复制到此模块目录中，并从其 **module.xml** 文件引用。如果您提供了目录的路径，则目录及其内容会被复制到模块目录中。不保留符号链接；链接的资源被复制到模块目录中。这个参数是必需的，除非提供了 **--absolute-resources** 或 **--module-xml**。 有关分隔符的详情，请参阅 **--resource-delimiter**。
--slot	使用这个参数将模块添加到默认 **main** 插槽以外的插槽中。 `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	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。

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

4.10. SASL-AUTHENTICATION-FACTORY 属性

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

表 4.12. 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 4.13. 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。

属性	描述
realm-mapper	机制使用的 realm mapper。

表 4.14. 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。

4.11. 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。

属性	描述
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。

4.12. 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 4.15. 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。

4.13. SIZE-ROTATING-FILE-AUDIT-LOG 属性

表 4.16. 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。

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

4.14. SYSLOG-AUDIT-LOG ATTRIBUTES

表 4.17. syslog-audit-log attributes

属性	描述

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。
syslog-format	用于描述审计事件的 RFC 格式。 支持的值： • **RFC3164** • **RFC5424** 默认值： • **RFC5424**

属性	描述
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**