Red Hat JBoss Enterprise Application Platform 7.4

如何配置身份管理

使用 LDAP 目录和其他身份存储管理用户访问红帽 JBoss 企业应用平台的说明.

Last Updated: 2024-02-11

Red Hat JBoss Enterprise Application Platform 7.4 如何配置身份管理

使用 LDAP 目录和其他身份存储管理用户访问红帽 JBoss 企业应用平台的说明.

法律通告

摘要

本指南将学习如何使用 LDAP 目录和其他身份存储，以用于 JBoss EAP 管理接口和安全域。本指南扩展了 JBoss EAP 安全架构 指南中提供的概念，在管理员对 LDAP 的基本知识以及 JBoss EAP 中的安全概念有了充分了解后，应进行检查。

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. 请包含 **文档 URL、章节编号 并描述问题**。
3. 在 **Summary** 中输入问题的简短描述。
4. 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。
5. 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

红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始： master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 CTO Chris Wright 信息。

第 1 章 IDENTITY MANAGEMENT 概述

红帽 JBoss 企业应用平台(JBoss EAP)安全架构 *指南介绍了使用各种身份存储保护应用安全* 的基本身份管理概念。本指南将向您介绍如何配置各种身份存储（如文件系统或 LDAP）来保护应用程序的安全。在某些情况下，您还可以将某些身份存储（如 LDAP）用作授权授权。关于主体的各种角色和访问信息可以存储在 LDAP 目录中，然后直接供 JBoss EAP 使用或映射到现有的 JBoss EAP 角色。

注意

使用由数据库或 LDAP 目录等外部数据存储支持的身份存储可能会对身份验证和授权产生性能影响，因为外部数据存储和 JBoss EAP 实例之间的数据访问和传输可能会对身份验证和授权造成性能影响。

第 2 章 ELYTRON 子系统

2.1. 使用基于文件系统的身份存储配置身份验证

1. 在 JBoss EAP **中配置文件系统** 域：

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

如果您的目录位于 **jboss.server.config.dir** 之外，则需要相应地更改 **路径** 和 **相对值**。

2. 添加用户：
使用 **filesystem-realm** 时，您可以使用管理 CLI 添加用户。

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

3. 添加 **simple-role-decoder** ：

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

此 **simple-role-decoder** 从 **Roles** 属性解码主体的角色。如果您的角色在不同的属性中，您可以更改此值。

4. 配置 **security-domain** ：

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

5. 在 **undertow** 子系统中配置 **application-security-domain** ：

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

注意

可以通过导航到 **Configuration → Subsystems → Web(Under tow)→ Application Security Domain，使用管理控制台配置 undertow** 子系统中的 **application-security-domain**。

6. 配置应用程序的 **web.xml** 和 **jboss-web.xml** ：
您的 **web.xml** 和 **jboss-web.xml** 必须更新为使用您在 JBoss EAP 中配置的 **application-security-domain**。例如，将 Web 应用程序配置为使用 Elytron 或传统安全性进行身份验证。

您的应用程序现在使用基于文件系统的身份存储进行身份验证。

2.2. 使用基于文件的属性身份存储配置身份验证

1. 创建属性文件：
   您必须创建两个属性文件：一个用于将用户映射到密码，另一个将用户映射到角色。通常，这些文件位于 **jboss.server.config.dir** 目录中，并遵循命名约定 ***-users.properties** 和 ***-roles.properties**，但也可以使用其他位置和名称。***-users.properties** 文件还必须包含对 **properties-realm** 的引用，您要在下一步中创建：
   #$REALM_NAME=*YOUR_PROPERTIES_REALM_NAME*$

   密码文件示例：example-users.properties

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

   角色文件示例：example-roles.properties

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

2. 在 JBoss EAP **中配置 properties-realm** ：

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

   properties-realm 的名称为 **examplePropRealm**，它用于 **example-users.properties** 文件中的上一步中。此外，如果您的属性文件位于 **jboss.server.config.dir** 之外，您必须相应地更改 **路径** 和 **相对值**。

3. 配置 **security-domain** ：

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

4. 在 **undertow** 子系统中配置 **application-security-domain** ：

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

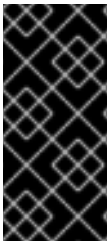
   

   注意

   可以通过导航到 **Configuration → Subsystems → Web(Under tow)→ Application Security Domain，使用管理控制台配置 undertow** 子系统中的 **application-security-domain**。

5. 配置应用程序的 **web.xml** 和 **jboss-web.xml** ：
   您的 **web.xml** 和 **jboss-web.xml** 必须更新为使用您在 JBoss EAP 中配置的 **application-security-domain**。例如，将 Web 应用程序配置为使用 Elytron 或传统安全性进行身份验证。

您的应用现在使用基于文件的属性身份存储进行身份验证。



重要

属性文件仅在服务器启动时读取。在服务器启动后添加的任何用户（手动或使用 **add-user** 脚本）都需要重新加载服务器。此重新加载通过从管理 CLI 运行 **reload** 命令来完成。

```
reload
```

2.3. 使用基于数据库的身份存储配置身份验证

1. 确定用户名、密码和角色的数据库格式：
要使用身份存储的数据库来设置身份验证，您需要确定用户名、密码和角色如何存储在该数据库中。在本例中，我们使用带有以下示例数据的单个表：

username	password	roles
user1	password123	Admin
user2	password123	guest

2. 配置数据源：
若要从 JBoss EAP 连接数据库，您必须已部署适当的数据库驱动程序，并且配置了数据源。本例演示了为 PostgreSQL 部署驱动程序并在 JBoss EAP 中配置数据源：

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

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

3. 在 JBoss EAP 中配置 a **jdbc-realm**:

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

注意

上例演示了如何从单个 **主体查询获取密码和角色。**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，您也可以 **使用 attribute-mapping** 属性创建额外的 principal- **query**。

有关支持的密码映射程序列表，请参阅 密码映射程序。

4. 配置 **security-domain** ：

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

5. 在 **undertow** 子系统中配置 **application-security-domain** ：

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

注意

可以通过导航到 **Configuration → Subsystems → Web(Under tow)→ Application Security Domain，使用管理控制台配置 undertow** 子系统中的 **application-security-domain**。

6. 配置应用程序的 **web.xml** 和 **jboss-web.xml** ：
您的 **web.xml** 和 **jboss-web.xml** 必须更新为使用您在 JBoss EAP 中配置的 **application-security-domain**。例如，将 Web 应用程序配置为使用 Elytron 或传统安全性进行身份验证。

2.4. 使用基于 LDAP 的身份存储配置身份验证

1. 确定用户名、密码和角色的 LDAP 格式：
要使用 LDAP 服务器设置身份存储身份验证，您需要确定用户名、密码和角色的存储方式。在本例中，我们使用以下结构：

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

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

dn: uid=jsmith,ou=Users,dc=wildfly,dc=org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
cn: John Smith
sn: smith
uid: jsmith
userPassword: password123

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

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

2. 配置 a **dir-context**:

若要从 JBoss EAP 连接到 LDAP 服务器，您需要配置一个 **dir-context**，它提供 URL 以及用于连接服务器的主体。

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

注意

无法使用 Jakarta 管理 对**象名称来** 解密 LDAP 凭证。相反，可以通过如何使用凭据 存储来保护凭据，如 *如何为 JBoss EAP 配置服务器* 安全性中所述。

3. 在 JBoss EAP **中配置 ldap-realm**:

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

警告

如果引用的 LDAP 服务器包含循环引用，则可能会导致 JBoss EAP **服务器上的 java.lang.OutOfMemoryError** 错误。

4. 添加 **simple-role-decoder** ：

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

5. 配置 **security-domain** ：

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

6. 在 **undertow** 子系统中配置 **application-security-domain** ：

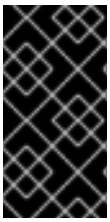
```
/subsystem=undertow/application-security-domain=exampleApplicationDomain:add(security-domain=exampleLdapSD)
```

注意

可以通过导航到 **Configuration → Subsystems → Web(Under tow)→ Application Security Domain，使用管理控制台配置 undertow** 子系统中的 **application-security-domain**。

7. 配置应用程序的 **web.xml** 和 **jboss-web.xml** ：

您的 **web.xml** 和 **jboss-web.xml** 必须更新为使用您在 JBoss EAP 中配置的 **application-security-domain**。例如，将 Web 应用程序配置为使用 Elytron 或传统安全性进行身份验证。



重要

如果 **elytron** 子系统使用 LDAP 服务器来执行身份验证，JBoss EAP 将返回 500 或内部服务器错误， 如果该 LDAP 服务器无法访问，则返回错误代码。此行为与使用传统 **security** 子系统的旧版 JBoss EAP 不同，后者在相同的条件下返回 **401** 或未授权的错误代码。

2.5. 使用证书配置身份验证

重要

您必须先配置双向 SSL，然后才能设置基于证书的身份验证。有关配置双向 SSL 的详情，请参考《如何 *配置服务器 安全性指南* 》的 *Elytron Subsystem 部分，为应用启用双向 SSL/TLS*。

1. 配置 **密钥存储域.**

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

您必须为这个域配置一个包含客户端证书的信任存储。身份验证过程使用客户端在双向 SSL 握手期间提供的相同证书。

2. 创建解码器。
您需要创建一个 **x500-attribute-principal-decoder** 来解码您从证书中获得的主体。以下示例将根据第一个 **CN** 值对主体进行解码。

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

例如，如果完整 **DN** 为 **CN=client,CN=client=certificate,DC=example,DC=jboss,DC=org**，**CNDecoder** 会将该主体解码为 **客户端**。此解码主体用作 **别名** 值，用于在 **ksRealm** 中配置的信任存储中查找证书。

重要

已解码的主体 MUST 是您在服务器信任存储中为客户端证书设置的 **别名** 值。

- 另外，您还可以使用主题替代名称扩展来配置一个证据解码器，以将主题替代名称用作主体。如需更多信息，请参阅 *如何配置服务器安全指南中的 X.509 证书配置* 带有主题备用名称扩展的 X.509 证书。

3. 添加用于分配角色的 **常量角色映射器**。
例如，使用 constant **-role-mapper** 将角色分配给 **ksRealm** 中的主体，但也可以使用其他方法。

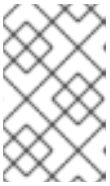
```
/subsystem=elytron/constant-role-mapper=constantClientCertRole:add(roles=[Admin,Guest])
```

4. 配置 **安全域.**

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

5. 在 **undertow** 子系统中配置 **application-security-domain**。

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



注意

可以通过导航到 **Configuration → Subsystems → Web(Under tow)→ Application Security Domain，使用管理控制台配置 undertow** 子系统中的 **application-security-domain**。

6. 更新 **server-ssl-context.**

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

7. 配置应用程序的 **web.xml** 和 **jboss-web.xml**。
   您的 **web.xml** 和 **jboss-web.xml** 必须更新为使用您在 JBoss EAP 中配置的 **application-security-domain**。例如，将 Web 应用程序配置为使用 Elytron 或传统安全性进行身份验证。

   此外，您需要更新 **web.xml**，以使用 **CLIENT-CERT** 作为其身份验证方法。

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

2.6. 使用多个身份服务配置身份验证和授权

如果您在不同身份存储之间存储身份属性，则请使用 **aggregate-realm** 将身份属性加载到单个安全域中以进行身份验证和授权。

2.6.1. 在 Elytron 中聚合 Realm

通过 **聚合域**，您可以使用一个安全域进行身份验证，也可以使用另一个安全域聚合多个安全域，从而在 Elytron 中进行身份验证。例如，您可以将聚合域配置为使用属性域来进行身份验证，并使用 JDBC 域来进行身份验证。

在配置为聚合多个授权域的聚合域中，会创建一个身份，如下所示：

- 加载为授权配置的每个安全域的属性值。
- 如果在多个授权域中定义了属性，则使用属性第一次出现的值。

以下示例演示了如何在多个授权域包含相同身份属性的定义时创建身份。

示例

聚合 realm 配置：

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

- 从 JDBC 域获取的属性值：

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

- 从 ldap 域获取的属性值：

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

从聚合域获取的身份：

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

在示例中，两个授权域中都定义了 **e-mail** 属性。JDBC 域中定义的值用于生成的聚合域中的 **电子邮件** 属性，因为聚合域配置为聚合授权域，**即 authorization-realms=[jdbc-realm,ldap-realm]**。

2.6.2. 使用聚合域配置身份验证和授权

要使用聚合域配置身份验证和授权，请创建一个聚合域，并配置安全域和应用安全域以使用聚合域。

先决条件

- 配置要聚合的安全域。
  有关配置安全域的详情，请参考 *如何配置身份管理* 指南中的 Elytron subsystem。
- 配置要在安全域中使用的角色解码器。
  有关角色解码器的详情，请参阅 *如何配置服务器安全* 指南中的创建 Elytron Role Decoder。

流程

1. 创建聚合域：
   - 使用一个授权域创建聚合域：

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

   - 创建具有多个授权域的聚合域：

```
/subsystem=elytron/aggregate-realm=exampleAggregateRealm:add(authentication-
realm=__SECURITY_REALM_FOR_AUTHENTICATION__, authorization-realms=
[__SECURITY_REALM_FOR_AUTHORIZATION_1__,__SECURITY_REALM_FOR_AU
THORIZATION_2__,...,__SECURITY_REALM_FOR_AUTHORIZATION_N__])
```

2. 配置 **security-domain** ：

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

3. 在 **undertow** 子系统中配置 **application-security-domain** ：

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

4. 配置应用程序的 **web.xml** 和 **jboss-web.xml** ：
   您的 **web.xml** 和 **jboss-web.xml** 必须更新为使用您在 JBoss EAP 中配置的 **application-security-domain**。例如，将 Web 应用程序配置为使用 Elytron 或传统安全性进行身份验证。

2.6.3. Aggregate Realms 示例

使用单个授权域聚合域示例

在本例中，s **properties-realm** 用于身份验证，aj **dbc-realm** 用于授权。

您必须预配置以下域：

- **properties-realm** named examplPropertiesRealm
- **jdbc-realm** named exampleJdbcRealm

使用以下命令创建一个聚合域：

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

包含两个授权域的聚合域示例

在本例中，**properties-realm** 用于身份验证，并且 **ldap-realm** 和 **jdbc-realm** 的聚合用于授权。

您必须预配置以下域：

- **properties-realm** named examplPropertiesRealm
- **jdbc-realm** named exampleJdbcRealm
- **ldap-realm** named exampleLdapRealm

使用以下命令创建一个聚合域：

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

2.7. 覆盖应用的身份验证配置

您可以使用在 JBoss EAP 中配置的应用，覆盖应用的身份验证配置。要做到这一点，使用 **undertow** 子系统的 **application -security-domain 部分中的 override- deployment-configuration** 属性：

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

注意

可以通过导航到 **Configuration → Subsystems → Web(Under tow)→ Application Security Domain，使用管理控制台配置 undertow** 子系统中的 **application-security-domain**。

例如，应用已配置为使用 FORM **身份验证** 及其 **jboss-web.xml** 中的 **exampleApplicationDomain**。

jboss-web.xml示例

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

通过启用 **覆盖部署配置，** 您可以创建一个新的 **http-authentication-factory**，以指定不同的身份验证机制，如 **BASIC** 或 **DIGEST**。

http-authentication-factory示例

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

这将覆盖应用的 **jboss-web.xml** 中定义的身份验证机制，并尝试使用 **BASIC 而不是 FORM** 验证用户 **。**

2.8. 为安全 REALMS 设置缓存

Elytron 提供了一个 **缓存域**，允许您从安全域缓存凭据查找的结果。例如，您可以使用它为来自 LDAP 或数据库的凭证配置缓存，以提高经常查询用户的性能。

caching-realm 使用 *LRU* 或 *Least Recently Used* 缓存策略来缓存 **PasswordCredential** 凭证，该策略中访问的条目在达到最大条目数时将被丢弃。

您可以将 **缓存域用于以下** 安全域：

- **filesystem-realm**

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

如果您在 JBoss EAP 外部对凭据来源进行更改，则这些更改只有在底层安全域支持时才会传播到 JBoss EAP 缓存域。特别是 **ldap-realm** 支持侦听，但 **ldap-realm 内过滤的属性（ 如 角色 ）** 不支持。

为确保缓存域具有正确的用户数据缓存，建议您通过缓存域而不是您的凭据来源来修改用户属性。或者，您也可以 清除缓存。

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中的技术预览功能支持范围。

配置和使用 **缓存域** ：

1. 创建现有的安全域。
   您需要一个现有的安全域来 **用于缓存域**。例如，您可以创建一个 **类似于使用基于文件系统** 的身份存储配置身份验证中的步骤的文件系统 域。

   filesystem-realm 示例

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

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

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

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

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

2. 创建 **缓存域.**
   具有要缓存的现有域后，创建一个引用它的 **缓存域**。

   使用 exampleFsRealm 的缓存域示例

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

3. 使用 **缓存域.**
   创建 **缓存域** 后，您可以像任何其他安全域一样在安全配置中使用它。例如，您可以在相同位置使用 **文件系统域，通过基于文件系统的** 身份服务在配置身份验证时使用文件系统域。

   使用 cache-realm 配置示例

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

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

您可以使用 caching- **realm** 的 **maximum-entries 和 maximum- age** 属性来控制缓存大小和项目过期时间。有关这些属性的详情，请参阅 *如何配置服务器 安全性的 Elytron Subsystem 组件参考* 一节。

清除 缓存域 缓存

您可以使用 **clear-cache** 命令清除现有的缓存。清除缓存会强制它使用来自安全域的最新数据进行重复。

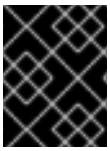
```
/subsystem=elytron/caching-realm=exampleCacheRealm:clear-cache
```

2.9. 配置应用程序以使用容器管理的单点登录

您可以通过 Elytron FORM **身份验证** 方法将 JBoss EAP 配置为将容器管理的单点登录用于应用。这允许用户进行身份验证一次，并访问 FORM **身份验证** 方法保护的其他资源，而无需重新进行身份验证。

在以下情况下，相关的单点登录会话无效：

- 没有有效的本地会话。
- 从应用注销。



重要

只要这些实例在集群中，就可以在不同的 JBoss EAP 实例上部署的应用之间使用单点登录。

1. 创建 **密钥存储**。
   需要 **密钥存储** 以在参与 SSO 的不同服务器之间配置安全通信通道。此通道用于交换创建或销毁单点登录会话时发生的事件的消息，分别登录和注销。

   要在 **elytron** 子系统中创建 **密钥** 存储，请首先创建一个 Java KeyStore，如下所示：

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

   创建 **密钥store.jks** 文件后，执行以下管理 CLI 命令在 Elytron 中创建 **密钥存储** 定义：

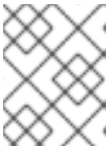
   ```
   /subsystem=elytron/key-store=example-keystore:add(path=keystore.jks, relative-
   to=jboss.server.config.dir, credential-reference={clear-text=secret}, type=JKS)
   ```

2. 添加安全域。
   使用以下管理 CLI 命令创建文件系统 **域**，这是用户在本地文件系统中存储的身份存储：

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

3. 使用以下管理 CLI 命令创建 **安全域** ：

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



注意

使用 SSO 的应用应当使用 **HTTP FORM 身份验证**，因为它们通常需要为用户提供登录页面。

4. 在 **undertow** 子系统中创建应用安全域。

注意

如果您已在 **undertow** 子系统中定义了 **application-security-domain**，只需将它用于为您的应用启用单点登录，您可以跳过这一步。

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

注意

默认情况下，如果您的应用未在 **jboss-web.xml** 文件中定义任何特定的安全域，应用服务器将以 **其他** 名称选择一个。

5. 更新 **undertow** 子系统，以启用单点登录并使用密钥存储：
单点登录启用到 **undertow** 子系统中的特定 **application-security-domain** 定义。您用于部署应用的服务器必须使用相同的配置。

   要启用单点登录，只需更改 **undertow** 子系统中的现有 **application-security-domain**，如下所示：

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

注意

可以通过导航到 **Configuration → Subsystems → Web(Under tow)→ Application Security Domain，使用管理控制台配置** undertow 子系统中的 **application-security-domain**。

   如需有关 SSO 属性及其定义的更多信息，请参阅单点登录属性 的参考资料。

6. 配置应用的 **web.xml** 和 **jboss-web.xml** 文件。
您的 **web.xml** 和 **jboss-web.xml** 必须更新为使用您在 JBoss EAP 中配置的 **application-security-domain**。例如，将 Web 应用程序配置为使用 Elytron 或传统安全性进行身份验证。

JBoss EAP 为 使用 **undertow** 和 **infinispan** 子系统的集群和非集群 SSO 提供开箱即用的支持。

2.10. 使用 BEARER 令牌配置身份验证和授权

2.10.1. bearer 令牌身份验证

您可以使用 **BEARER_TOKEN** 身份验证机制授权发送到应用程序的 HTTP 请求。在 Web 浏览器等客户端向应用程序发送 HTTP 请求后，**BEARER_TOKEN** 机制验证请求 **的授权 HTTP** 标头中是否存在 bearer 令牌。

Elytron 支持使用 bearer 令牌（如 OpenID Connect ID 令牌）或利用 OAuth2 兼容授权服务器发布的不透明令牌进行身份验证。请参阅其他资源部分。

以下示例显示 **Authorization** HTTP 标头包含 **mF_9.B5f-4.1JqM** bearer 令牌：

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

BEARER_TOKEN 机制现在可以提取 bearer 令牌字符串，并将它传递到 **token-realm** 实施以进行验证。如果实施成功验证 bearer 令牌，Elytron 将根据令牌代表的信息创建一个安全性上下文。应用可以使用此安全上下文来获取请求者的信息。然后，它可以通过提供请求者对 HTTP 资源的访问权限来决定是否满足请求。

如果请求者不提供 bearer 令牌，**BEARER_TOKEN** 机制会返回 **401 HTTP** 状态代码。例如：

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

如果请求者无权访问资源，**BEARER_TOKEN** 机制将返回 **403 HTTP** 状态代码。

```
HTTP/1.1 403 Forbidden
```

其他资源

- 有关 JWT 验证的详情，请参阅配置 JSON Web 令牌(JWT)身份验证。
- 有关 OAuth2 验证的详情，请参考使用兼容 OAuth2 授权服务器 发布的令牌配置身份验证。
- 有关可用于 **token-realm** 的属性的更多信息，请参阅 *如何配置服务器安全* 指南中 的表 A.93.**token-realm** Attributes。

2.10.2. 配置 JSON Web 令牌(JWT)身份验证

您可以通过在 **elytron** 子系统 **中指定令牌** 域来启用对 JWT 的支持。

在 **token-realm** 中，您可以指定属性和 **jwt** 令牌验证器。Elytron 完成以下检查：

- 自动过期检查 **exp** 声明和 **nbf** 声明中指定的值。
- 可选：根据以下方法之一提供的公钥进行签名检查：
  - 使用 **公钥或** 证书 属性.
  - 使用带命名公钥的密钥映射.

- 使用 **client-ssl-context** 属性从 **jku** claim 中指定的 URL 检索远程 JSON Web 密钥(JWK)。

- 可选：检查 JWT，以确保它只包含 **中** 支持的值和 **aud** 声明。您可以使用 签发 **者和使用者** 属性来执行这些检查。

下例演示了 **token-realm** 作为安全域，而 **principal-claim** 作为属性。**principal-claim** 属性定义 **elytron 用于** 获取主体名称的声明的名称。**jwt** 元素指定令牌必须被验证为 JWT。

配置的 token-realm示例

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

您可以为您的 **token-realm** 定义密钥映射。然后，您可以使用不同的密钥对来验证签名并轻松地轮转这些密钥对。Elytron 从令牌获取一名 **孩子** 索赔，并使用对应的公钥进行验证。

- 如果 JWT 中不存在 **孩子** 声明，则 **token-realm** 使用 **jwt** 的 **public-key** 属性中指定的值来验证签名。

- 如果 JWT 中不存在 **孩子** 声明，并且您尚未配置 **公钥，则 token -realm** 将使令牌失效。

为 **token-realm 配置的密钥映射示例。**

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

流程

1. 使用 **keytool 创建密钥存储。**

   使用 keytool 创建密钥存储 的示例：

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

   接下来，在 **elytron** 子系统中添加 **密钥存储** 定义。

   在 elytron 子系统中添加 密钥存储 定义的示例：

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

2. 在 **elytron** 子系统中创建 **token-realm**，并为您的 **token-realm** 指定属性和 **jwt** 令牌验证器。

   在 elytron 子系统中创建 令牌域 的示例：

   -

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

后续步骤

- 要为应用程序配置身份验证，请参阅为应用程序配置身份验证。

其他资源

- 有关 **token-realm jwt** 属性的更多信息，请参阅 *如何配置服务器安全* 指南中 的表 A.94. token-realm jwt Attributes。

2.10.3. 使用兼容 OAuth2 的授权服务器发布的令牌配置身份验证

Elytron 支持由 OAuth2 兼容的授权服务器发布的 bearer 令牌。您可以配置令牌域，以针对预定义的 **oauth2-introtion** 端点验证令牌。

流程

1. 创建令牌域。

   使用 elytron 子系统创建令牌域示例：

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

   以下示例显示了 token **- realm 元素中指定的 oauth2-introspection** 元素。此令牌域配置为针对预定义的 **oauth2-introtion** 端点验证令牌。**oauth2-introspection** 端点使用 **client-id 和 client - secret** 属性中指定的值来识别客户端。

   token -realm 元素中的 oauth2-introspection 元素示例：

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

后续步骤

- 要为应用程序配置身份验证，请参阅为应用程序配置身份验证。

其他资源

- 如需有关令牌内省端点的更多信息，请参阅 Table A.95。**token-realm oauth2-introspection** Attributes。

2.10.4. 为应用程序配置 bearer 令牌身份验证

您可以使用 JWT 格式的 bearer 令牌（如 OpenID Connect ID 令牌）或者使用 OAuth2 兼容授权服务器发布的不透明令牌来配置应用身份验证。

先决条件

- 根据您需要的身份验证方法，完成以下步骤之一来创建 **token-realm** ：
  - 配置 JSON Web 令牌(JWT)身份验证。
  - 使用由 OAuth2 兼容授权服务器 发布的令牌配置身份验证。

流程

1. 在 **elytron** 子系统中创建安全域。确保您在安全域中指定您的令牌安全域。

   在 elytron 子系统中创建安全域的示例.

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

2. 创建使用 **BEARER_TOKEN** 机制的 **http-authentication-factory**。

   创建 http-authentication-factory 的示例.

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

3. 在 **undertow** 子系统中配置 **application-security-domain**。

   在 undertow 子系统中配置 application-security-domain 的示例：

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

4. 配置应用的 **web.xml** 和 **jboss-web.xml** 文件。您必须确保应用程序的 **web.xml** 指定 **BEARER_TOKEN** 身份验证方法。此外，确保 **jboss-web.xml** 使用您在 JBoss EAP 中配置的 **application-security-domain**。

其他资源

- 有关 **BEARER_TOKEN** 身份验证机制的更多信息，请参阅 Bearer 令牌身份验证。
- 有关 **token-realm jwt** 属性的更多信息，请参阅 *如何配置服务器安全* 指南中 的表 A.94. token-realm jwt Attributes。
- 有关可用于 OAuth2 令牌端点的属性的更多信息，请参阅 Table A.95。 **token-realm oauth2-introspection** Attributes。
- 有关配置应用程序的 web.xml 和 **jboss- web.xml** 的更多信息，请参阅《 *如何配置身份管理* 指南》中的 配置 Web 应用程序使用 Elytron 或 Legacy Security for Authentication 。

第 3 章 旧安全子系统

3.1. 配置安全域以使用 LDAP

安全域可以配置为使用 LDAP 服务器进行身份验证和授权，方法是使用登录模块。JBoss EAP 安全架构 *指南* 介绍了安全域和登录模块的基础知识。*LdapExtended 是* 与 LDAP 服务器（包括 Active Directory）集成的首选登录模块，但也可以使用其他几个 LDAP 登录模块。特别是，*Ldap*、*Ada* p 和 *AdvancedAdap 登录* 模块也可用于配置安全域以使用 LDAP。本节使用 *LdapExtended 登录* 模块来演示如何创建使用 LDAP 进行身份验证和授权的安全域，但也可以使用其他 LDAP 登录模块。有关其他 LDAP 登录模块的更多详细信息，请参阅 JBoss EAP *登录模块参考*。

重要

如果传统 **安全** 子系统使用 LDAP 服务器来执行身份验证，如果该 LDAP 服务器无法访问 ， JBoss EAP 将返回 500 或内部服务器错误。此行为与之前版本的 JBoss EAP 不同，后者在相同的条件下返回 **401** 或未授权的错误代码。

3.1.1. LdapExtended Login 模块

LdapExtended(**org.jboss.security.auth.spi.LdapExtLoginModule**)是一个登录模块，它使用搜索来查找 LDAP 服务器上的 bind 用户和相关角色。角色以递归方式查询 DN，以浏览分层角色结构。对于将 LDAP 与安全域搭配使用时的绝大多数情形中，应使用 LdapExtended 登录模块，特别是对于非 Active Directory 的 LDAP 实施。有关 LdapExtended login 模块配置选项的完整列表，请参见 JBoss EAP *登录模块参考*中的 LdapExtended login 模块部分 。

验证过程如下：

1. 使用 bind **DN** 和 **bindCredential** 选项实现到 LDAP 服务器的初始绑定。 **bindDN** 是一个 LDAP 用户，能够同时搜索 **baseCtxDN** 和 **rolesCtxDN** 树以用户和角色搜索。要使用 **baseFilter** 属性指定的过滤器查询要对其进行身份验证的用户 DN。

2. 生成的用户 DN 通过使用用户 DN 作为 **InitialLdapContext** 环境 上下文.**SECURITY_PRINCIPAL** 绑定到 LDAP 服务器来验证。 **Context.SECURITY_CREDENTIALS** 属性设置为回调处理程序获取的 **String** 密码。

3.1.1.1. 配置安全域以使用 LdapExtended Login 模块

数据示例（LDIF 格式）

```
dn: uid=jduke,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: Java Duke
sn: duke
uid: jduke
userPassword: theduke
# ============================
dn: uid=hnelson,ou=Users,dc=jboss,dc=org
objectClass: inetOrgPerson
objectClass: person
objectClass: top
cn: Horatio Nelson
sn: Nelson
```

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

```
cn: GroupOne
uid: GroupOne
uniqueMember: uid=jduke,ou=Users,dc=jboss,dc=org
uniqueMember: uid=hnelson,ou=Users,dc=jboss,dc=org
```

用于添加 LdapExtended Login 模块的 CLI 命令

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

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

/subsystem=security/security-domain=testLdapExtendedExample/authentication=classic/login-
module=LdapExtended:add(code=LdapExtended, flag=required, module-options=[
("java.naming.factory.initial"=>"com.sun.jndi.ldap.LdapCtxFactory"),
("java.naming.provider.url"=>"ldap://localhost:10389"),
("java.naming.security.authentication"=>"simple"),
("bindDN"=>"uid=ldap,ou=Users,dc=jboss,dc=org"), ("bindCredential"=>"randall"),
("baseCtxDN"=>"ou=Users,dc=jboss,dc=org"), ("baseFilter"=>"(uid={0})"),
("rolesCtxDN"=>"ou=groups,dc=jboss,dc=org"),     ("roleFilter"=>"(uniqueMember={1})"),
("roleAttributeID"=>"uid")])

reload
```

注意

显示的管理 CLI 命令假定您在运行 JBoss EAP 单机服务器。有关将管理 CLI 用于 JBoss EAP 受管域的更多详细信息，请参见 JBoss EAP *管理 CLI 指南*。

3.1.1.1.1. 配置安全域以使用 LdapExtended Login 模块进行 Active Directory

对于 Microsoft Active Directory，可以使用 LdapExtended 登录模块。

以下示例显示了默认 Active Directory 配置的配置。某些 Active Directory 配置可能需要在端口 **3268** 上针对 Global Catalog 进行搜索，而不是通常的端口 **389**。这很可能是因为 Active Directory 林中包含多个域。

用于默认 AD 配置的 LdapExtended Login 模块配置示例

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

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

/subsystem=security/security-domain=AD_Default/authentication=classic/login-
module=LdapExtended:add(code=LdapExtended,flag=required,module-options=[
("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"),("bindDN"=>"JBOSSsearchuser"),
("bindCredential"=>"password"), ("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"), ("baseFilter"=>"
(sAMAccountName={0})"), ("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"), ("roleFilter"=>"
(sAMAccountName={0})"), ("roleAttributeID"=>"memberOf"), ("roleAttributeIsDN"=>"true"),
("roleNameAttributeID"=>"cn"), ("searchScope"=>"ONELEVEL_SCOPE"),
("allowEmptyPasswords"=>"false")])

reload
```

以下示例在 Active Directory 中实施递归角色搜索。本例与默认 Active Directory 示例之间的主要区别在于，角色搜索已被替换，以使用用户的 DN 搜索 member 属性。然后，登录模块使用角色的 DN 来查找该组所属的组。

使用递归搜索进行默认 AD 配置的 LdapExtended Login 模块配置示例

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

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

/subsystem=security/security-domain=AD_Recursive/authentication=classic/login-
module=LdapExtended:add(code=LdapExtended,flag=required,module-options=
[("java.naming.provider.url"=>"ldap://ldaphost.jboss.org"), ("java.naming.referral"=>"follow"),
("bindDN"=>"JBOSSsearchuser"), ("bindCredential"=>"password"),
("baseCtxDN"=>"CN=Users,DC=jboss,DC=org"), ("baseFilter"=>"(sAMAccountName={0})"),
("rolesCtxDN"=>"CN=Users,DC=jboss,DC=org"), ("roleFilter"=>"(member={1})"),
("roleAttributeID"=>"cn"), ("roleAttributeIsDN"=>"false"), ("roleRecursion"=>"2"),
("searchScope"=>"ONELEVEL_SCOPE"), ("allowEmptyPasswords"=>"false")])

reload
```

3.2. 配置安全域以使用数据库

与 LDAP 类似，安全域可以配置为使用数据库通过登录模块进行身份验证和授权。

3.2.1. 数据库登录模块

Database 登录模块是 JDBC 登录模块，支持身份验证和角色映射。如果用户名、密码和角色信息存储在关系数据库中，则使用此登录模块。

这可以通过以预期格式提供对逻辑表的引用，包含主体和角色。例如：

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

Principals 表将用户 **PrincipalID** 与有效密码关联，PrincipalID 将用户 **PrincipalID 与其** 角色集 **相关联**。用于用户权限的角色必须包含在 **Roles** 的 **RoleGroup** 列值的一行中。

表是逻辑的，用户可以指定登录模块使用的 SQL 查询。唯一的要求是 **java.sql.ResultSet** 具有与前面描述的 **Principals** 和 **Roles** 表相同的逻辑结构。表和列的实际名称无关，因为根据列索引访问结果。

为了说明此概念，请考虑一个包含已声明的表格 （**Principals** 和 **Roles** ）的数据库。以下语句使用以下数据填充表：

- **PrincipalID java**，在 **Principals** 表中密码为 **echoman**
- **PrincipalID java**，角色在 Roles **表** 中的 **RolesRoleGroup** 中名为 **Echo**
- **PrincipalID java**，角色在 **Roles** 表中的 **CallerPrincipalRoleGroup** 中具有名为 caller **-java** 的角色

有关 Database 登录模块配置选项的完整列表，请参阅 JBoss EAP 登录模块 *参考中的 Database 登录模块*部分。

3.2.1.1. 配置安全域以使用数据库登录模块

在将安全域配置为使用 Database 登录模块之前，必须正确配置数据源。

有关在 JBoss EAP 中创建和配置数据源的更多信息，请参见 JBoss EAP *配置指南*中的数据源管理部分。

正确配置了数据源后，可以将安全域配置为使用 Database 登录模块。以下示例假定已创建了名为 **MyDatabaseDS** 的数据源，并使用以下数据库进行了正确配置：

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

用于添加数据库登录模块的 CLI 命令

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

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

/subsystem=security/security-domain=testDB/authentication=classic/login-
module=Database:add(code=Database,flag=required,module-options=
[("dsJndiName"=>"java:/MyDatabaseDS"),("principalsQuery"=>"select passwd from Users where
username=?"),("rolesQuery"=>"select role, 'Roles' from UserRoles where username=?")])

reload
```

3.3. 配置安全域以使用属性文件

安全域也可以配置为使用文件系统作为身份存储，通过使用登录模块进行身份验证和授权。

3.3.1. UsersRoles 登录模块

UsersRoles 是一种简单的登录模块，支持从 Java 属性文件加载的多个用户和用户角色。此登录模块的主要用途是利用应用部署的属性文件，轻松测试多个用户和角色的安全设置。默认的用户名至密码映射文件名为 **users.properties**，默认的用户名至角色映射文件名是 **roles.properties**。

注意

此登录模块支持密码堆栈、密码散列和未经身份验证的身份。

属性文件在初始化过程中使用初始化方法线程上下文类加载程序。这意味着这些文件可以放在 Jakarta EE 部署的类路径上（例如，放入 WAR 存档中的 **WEB-INF/classes** 文件夹）或服务器类路径上的任何目录中。

有关 UsersRoles login 模块配置选项的完整列表，请参阅 JBoss EAP *Login 模块参考中的 UsersRoles login 模块*部分。

3.3.1.1. 配置安全域以使用 UsersRoles 登录模块

以下示例假设创建了以下文件，并在应用程序的类路径上可用：

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

用于添加 UserRoles 登录模块的 CLI 命令

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

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

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

reload
```

3.4. 配置安全域以使用基于证书的身份验证

JBoss EAP 让您能够通过安全域使用基于证书的身份验证来保护 Web 应用或 Jakarta 企业 Beans。

重要

在配置基于证书的身份验证之前，您需要启用并配置 用于应用的双Way SSL/TLS，这需要为 JBoss EAP 实例以及访问 Web 应用或安全域保护的 Jakarta Enterprise Beans 配置 X509 证书。

在配置了证书、信任存储和双向 SSL/TLS 后，您可以继续配置使用基于证书的身份验证的安全域，配置应用使用该安全域，并将客户端配置为使用客户端证书。

3.4.1. 使用基于证书的身份验证创建安全域

要创建使用基于证书的身份验证的安全域，您需要指定信任存储以及 证书 登录模块或其子类。

truststore 必须包含用于身份验证的任何可信客户端证书，或者必须包含用于为客户端的证书签名的证书颁发机构的证书。login 模块用于利用已配置的信任存储来验证客户端提供的证书。整个安全域还必须提供一种方式，在角色通过身份验证后将角色映射到主体。证书登录模块本身不会将任何角色信息映射到主体，但也可与另一个登录模块结合使用。另外，证书登录模块的两个子类（ CertificateRoles 和 DatabaseCertificate ）提供了在角色验证后将角色映射到主体的方法。以下示例演示了如何使用 CertificateRoles 登录模块为安全域配置基于证书的身份验证。

警告

在进行身份验证时，安全域将使用客户端在建立双向 SSL/TLS 时提供的相同证书。因此，客户端必须使用相同的 **BOTH** 双向 SSL/TLS 证书，并使用应用程序或 Jakarta Enterprise Beans 进行基于证书的身份验证。

使用基于证书的身份验证的安全域示例

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

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

```
url="/path/to/server.keystore.jks"}, client-auth=true)

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

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

注意

上例使用 CertificateRoles login 模块来处理身份验证并将角色映射到经过身份验证的主体。它通过使用 **rolesProperties** 属性引用属性文件。此文件使用以下格式列出了用户名和角色：

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

由于提供的证书中的用户名作为 DN 显示，例如 **CN=valid-client、OU=JBoss、O=红帽、L=Raleigh、ST=NC、C=US**，您必须在使用属性文件时转义特殊字符，如 **=** 和空格：

角色属性文件示例

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

要查看证书的 DN：

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

3.4.2. 配置应用以使用带基于证书的身份验证的安全域

与将应用配置为使用其他形式身份验证的安全域类似，您需要同时正确配置 **jboss-web.xml** 和 **web.xml** 文件。

对于 **jboss-web.xml**，添加对您为基于证书的身份验证配置的安全域的引用。

jboss-web.xml 示例

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

对于 **web.xml**，将 **< login-config> 中的 <auth-method >** 属性设置为 **CLIENT-CERT**。您还需要定义 **<security-constraint> 和 <security-roles>**。

web.xml 示例

```
<web-app>
```

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

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

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

3.4.3. 配置客户端

客户端若要针对通过基于证书的身份验证保护的应用进行身份验证，客户端需要访问 JBoss EAP 实例的信任存储中包含的客户端证书。例如，如果使用浏览器访问应用，客户端需要将可信证书导入到浏览器的信任存储中。

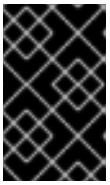
3.5. 配置安全域的缓存

您可以为安全域指定缓存，以加快身份验证检查。默认情况下，安全域使用简单的 map 作为缓存。这个默认缓存是最大 1000 个条目的 Least Recently Used(LRU)缓存。或者，您可以将安全域设置为使用 Infinispan 缓存，或者完全禁用缓存。

3.5.1. 为安全域设置缓存类型

先决条件

- 如果要将安全域配置为使用 Infinispan 缓存，您必须首先创建一个名为 security 的 Infinispan 缓存容器， 其中包含安全域将使用的默认缓存。



重要

您只能定义一个用于安全域的 Infinispan 缓存配置。虽然您可以有多个使用 Infinispan 缓存的安全域，但每个安全域都从 Infinispan 缓存配置中创建自己的缓存实例。

有关 创建缓存容器 的更多信息，请参阅 JBoss EAP *配置指南*。

您可以使用管理控制台或管理 CLI 来设置安全域的缓存类型。

- 使用管理控制台：

1. 导航到 **Configuration → Subsystems → Security(Legacy)**。
2. 从列表中选择安全域，然后单击 **View**。
3. 单击 **Edit**，然后在 **Cache Type** 字段中选择 **default** 或 **infinspan**。
4. 点 **Save**。

- 要使用管理 CLI，请使用以下命令：

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

例如，将 **其他** 安全域设置为使用 Infinispan 缓存：

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

3.5.2. 列出和查找主体

列出 Cache 中的 Principal

您可以使用以下管理 CLI 命令查看存储在安全域缓存中的主体：

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

从缓存中清空 Principal

如果需要，您可以从安全域的缓存中清空主体。

- 要刷新特定的主体，请使用以下管理 CLI 命令：

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

- 要从缓存中清除所有主体，请使用以下管理 CLI 命令：

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

3.5.3. 禁用安全域的缓存

您可以使用管理控制台或管理 CLI 为安全域禁用缓存。

- 使用管理控制台：
  1. 导航到 **Configuration → Subsystems → Security(Legacy)**。
  2. 从列表中选择安全域，然后单击 **View**。
  3. 点击 **Edit**，再选择 **Cache Type** 的空白值。
  4. 点 **Save**。
- 要使用管理 CLI，请使用以下命令：

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

第 4 章 应用程序配置

4.1. 配置 WEB 应用程序以使用 ELYTRON 或传统安全性进行身份验证

在配置了 **elytron 或** 传统 **安全** 子系统以进行身份验证后，您需要将应用配置为使用它。

1. 配置应用程序的 **web.xml**。
   您的 **web.xml** 需要配置为使用适当的身份验证方法。使用 **elytron** 子系统时，这在您创建的 **http-authentication-factory** 中定义。在使用旧 **安全** 子系统时，这取决于您的登录模块以及您要配置的身份验证类型。

   使用 BASIC 身份验证的 web.xml 示例

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

2. 将您的应用配置为使用安全域。
   您可以配置应用程序的 **jboss-web.xml**，以指定您要用于身份验证的安全域。使用 **elytron** 子系统时，这在您创建 **application-security-domain** 时定义。在使用传统 **安全** 子系统时，这是传统安全域的名称。

   jboss-web.xml示例

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

   使用 **jboss-web.xml**，您可以仅为单个应用配置安全域。或者，您也可使用 **undertow** 子系统为所有应用指定默认安全域：这样，您可以省略使用 **jboss-web.xml** 来配置各个应用的安全域。

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

重要

在 **undertow** 子系统中设置 **default-security-domain** 将应用到 **所有** 应用。如果设置了 **default-security-domain**，并且应用在 **jboss-web.xml** 文件中指定了安全域，则 **jboss-web.xml** 中的配置将覆盖 **undertow** 子系统中的 **default-security-domain**。

注意

Jakarta Enterprise Beans 的安全域在 Jakarta Enterprise Beans 配置中定义（在 **ejb3** 子系统中）、**jboss-ejb3.xml** 文件中的 Jakarta 企业 Bean 的描述符，或使用 **@SecurityDomain** 注释。

如需更多信息，请参阅开发 Jakarta 企业 Bean 应用程序指南中的 *Jakarta 企业 Beans 应用程序*安全性。

静默 BASIC 身份验证

您可以将 **elytron** 配置为执行静默的 **BASIC** 身份验证。启用静默身份验证后，不会提示用户登录以访问该 Web 应用。改为使用另一种身份验证机制。如果用户的请求包含授权标头，则使用 **BASIC** 身份验证机制。

要启用静默 **BASIC** 身份验证，请将 **auth-method** 属性的值设置为以下内容：

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

在 Parallel 中使用 Elytron 和传统安全子系统

您可以在 **elytron** 和旧 **安全** 子系统中定义身份验证，并并行使用它们。如果您在 **undertow** 子系统中使用 **jboss -web.xml 和 default-security-domain**，则 JBoss EAP 将首先尝试匹配 **elytron** 子系统中配置的安全域。如果未找到匹配项，则 JBoss EAP 将尝试将安全域与传统 **安全性** 子系统中配置的安全域匹配。如果 **elytron** 和传统 **安全** 子系统各自具有名称相同的安全域，则使用 **elytron** 安全域。

注意

如果您使用一个安全域定义了 Web servlet，并且您从另一个使用 Jakarta Enterprise Beans 特定安全域的另一个 EAR 模块调用 Jakarta Enterprise Beans，则可能会出现以下情况之一：

- 如果 WAR 和 Jakarta Enterprise Beans 映射到不同的 Elytron 安全域，您需要配置流流或可信安全域，以便其身份从一个部署域传播到下一个部署域。除非这样做，否则一旦呼叫到达 Jakarta Enterprise Beans，其身份将变为匿名身份。有关如何为身份验证配置安全身份的更多信息，请参阅配置可信安全域流。
- 如果 WAR 和 Jakarta Enterprise Beans 引用了不同的安全域名，但是它们映射到相同的 Elytron 安全域，则它们的身份将传播，而无需额外的步骤。

迁移时，最好迁移整个应用程序。不建议单独迁移 Jakarta Enterprise Beans 和 WAR，不建议并行使用 **elytron** 和旧版 **安全** 子系统。有关如何将应用迁移到使用 Elytron 的更多信息，请参阅 JBoss EAP 迁移指南中的 Migrating to Elytron。

4.2. 使用 ELYTRON 客户端配置客户端身份验证

连接 JBoss EAP 的客户端（如 Jakarta Enterprise Beans）可以使用 Elytron 客户端进行身份验证。Elytron 客户端是一种客户端框架，使远程客户端能够使用 Elytron 进行身份验证。Elytron Client 有以下组件：

身份验证配置

身份验证配置包含身份验证信息，如用户名、密码、允许 SASL 机制，以及在摘要身份验证期间要使用的安全域。身份验证配置中指定的连接信息会覆盖初始上下文的 **PROVIDER_URL** 中指定的任何值。

MatchRule

此规则用于决定要使用的身份验证配置。

身份验证上下文

用于建立连接的一组规则和身份验证配置。

建立连接时，客户端使用身份验证上下文。此身份验证上下文包含用于每个出站连接的身份验证配置的规则。例如，您可以在连接 server **2** 时具有在连接 **server1** 时使用一种身份验证配置的规则，以及另一个身份验证配置。身份验证上下文包含一组身份验证配置和一组规则，用于定义在建立连接时如何选择它们。身份验证上下文也可以引用 **ssl-context**，并可与规则匹配。

在建立连接时创建使用安全信息的客户端：

- 创建一个或多个身份验证配置。
- 通过创建规则和身份验证配置对创建身份验证上下文。
- 为建立连接创建一个可运行。
- 使用您的身份验证上下文运行您的可运行。

当您建立连接时，Elytron 客户端将使用身份验证上下文提供的一组规则与身份验证期间要使用的正确身份验证配置匹配。

您可以在建立客户端连接时使用以下方法之一：

重要

当使用 Elytron Client 进行 Jakarta Enterprise Beans 调用时，任何硬编码的程序身份验证信息（如在 **javax .naming.InitialContext 中设置 context. SECURITY_PRINCIPAL** ）都会覆盖 Elytron 客户端配置。

4.2.1. 配置文件方法

配置文件方法包括使用您的身份验证配置、身份验证上下文和匹配规则创建 XML 文件。

示例： custom-config.xml

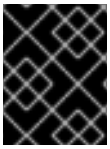
```
<configuration>
    <authentication-client xmlns="urn:elytron:client:1.2">
        <authentication-rules>
            <rule use-configuration="monitor">
                <match-host name="127.0.0.1" />
            </rule>
            <rule use-configuration="administrator">
                <match-host name="localhost" />
            </rule>
        </authentication-rules>
        <authentication-configurations>
            <configuration name="monitor">
                <sasl-mechanism-selector selector="DIGEST-MD5" />
```

```
            <providers>
              <use-service-loader />
            </providers>
            <set-user-name name="monitor" />
            <credentials>
               <clear-password password="password1!" />
            </credentials>
            <set-mechanism-realm name="ManagementRealm" />
          </configuration>

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

然后，您可以通过在运行客户端时设置系统属性，在客户端的代码中引用该文件。

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



重要

如果您使用 编程方法，它将覆盖任何提供的配置文件，即使设置了 **wildfly.config.url** 系统属性。

在创建规则时，您可以查找各种参数（如 **主机名**、**端口** 、 协议或 **user-name**）**的** 匹配项。Javadocs 中提供了 **MatchRule** 的完整选项列表。规则按照配置的顺序进行评估。

如果规则中没有包含匹配设置，则整个规则匹配并选择身份验证配置。如果规则中包含多个匹配设置，则所有匹配设置都必须匹配才能选择身份验证配置。

表 4.1. 通用规则

属性	描述
match-local-security-domain	**取单个 name 属性来** 指定要匹配的本地安全域。
match-host	**取单个 name 属性来** 指定要匹配的主机名。例如：主机 **127.0.0.1 将与 http://127.0.0.1:9990/my/path** 上的匹配。
match-no-user	匹配没有用户的 URI。

属性	描述
match-path	**取单个 name** 属性来指定要匹配的路径。例如：路径 **/my/path/** 匹配 **http://127.0.0.1:9990/my/path**。
match-port	**取单个 name 属性来** 指定要匹配的端口。例如：端口 **9990 将与 http://127.0.0.1:9990/my/path** 上的匹配。
match-protocol	**取单个 name 属性来** 指定要匹配的协议。例如：协议 **http** 在 **http://127.0.0.1:9990/my/path** 上匹配。
match-urn	**取单个 name 属性来** 指定要匹配的 URN。
match-user	**取单个 name 属性来** 指定要匹配的 **用户**。

wildfly-config.xml 文件示例可在 示例 **wildfly-config.xml** 中找到。有关如何配置 **wildfly-config.xml** 文件的更多信息，请参阅使用 **wildfly-config.xml** 文件进行客户端配置。

4.2.2. 编程方法

编程方法在客户端代码中配置所有 Elytron 客户端配置：

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

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


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

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

在 **AuthenticationConfiguration 和 Authentication Context** 中添加配置详情时，每个方法调用都会返回该对象的新实例。例如，如果您在使用不同主机名连接时需要单独的配置，您可以执行以下操作：

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

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


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


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

表 4.2. 通用规则

规则	描述
matchLocalSecurityDomain(String name)	这与 配置文件方法中的 **match-domain** 相同。
matchNoUser()	这与 配置文件方法 **中的 match-no-user** 相同。
matchPath(String pathSpec)	这与 配置文件方法中的 **match-path** 相同。
matchPort(int port)	这与 配置文件方法中的 **match-port** 相同。
matchProtocol(String protoName)	这与 配置文件方法中的 **match-port** 相同。
matchPurpose（String 目的）	创建一个新规则，它与此规则相同，但也与给定目的名称匹配。
matchUrnName(String name)	这与 配置文件方法中的 **match-urn** 相同。
matchUser(String userSpec)	这与 配置文件方法 **中的 match-userinfo** 相同。

另外，您可以使用 **captureCurrent（）** 从当前配置的配置开始，而不是从空身份验证配置开始。

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

使用 **captureCurrent（）** 将捕获任何之前建立的身份验证上下文，并将其用作您的新基础配置。通过 **调用 run（）** 激活验证上下文后，就会建立验证上下文。如果名为 **captureCurrent（）**，并且当前没有活跃上下文，它将尝试使用默认验证（如果可用）。您可以在以下部分找到有关此问题的更多详细信息：

- 配置文件方法
- 默认配置方法
- 将 Elytron 客户端与部署到 JBoss EAP 的客户端搭配使用

AuthenticationConfiguration.empty（） 应当仅用作在 上构建配置的基础，不应自行使用。它提供了一个配置，它使用 JVM 范围注册的提供程序并启用匿名身份验证。

在 **AuthenticationConfiguration.empty（）** 配置上指定提供程序时，您可以指定自定义列表，但大多数用户应使用 **WildFlyElytronProvider（）** 提供程序。

在创建身份验证上下文时，使用 **context.with(...)** 将创建一个新上下文，该上下文会将当前上下文中的规则和身份验证配置与所提供的规则和身份验证配置合并。提供的规则和身份验证配置将显示在当前上下文中的后面。

4.2.3. 默认配置方法

默认配置方法完全依赖于 Elytron Client 提供的配置：

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

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

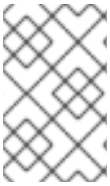
为提供默认配置，Elytron 客户端会尝试自动发现文件系统上的 **wildfly-config.xml** 文件。它位于以下位置：

- 由 **wildfly.config.url** 系统属性在客户端代码之外设置的位置。
- classpath 根目录。
- 类路径上的 **META-INF** 目录。
- 当前用户的主目录。
- 当前工作目录。

您可以使用以下示例作为客户端 **wildfly-config.xml** 文件的基本配置。

Basic wildfly-config.xml

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <authentication-rules>
      <rule use-configuration="default" />
    </authentication-rules>
    <authentication-configurations>
      <configuration name="default">
        <sasl-mechanism-selector selector="#ALL" />
        <set-mechanism-properties>
          <property key="wildfly.sasl.local-user.quiet-auth" value="true" />
        </set-mechanism-properties>
        <providers>
          <use-service-loader/>
        </providers>
      </configuration>
    </authentication-configurations>
  </authentication-client>
</configuration>
```



注意

ANONYMOUS 机制不支持 **非匿名用户** 的授权。这意味着 **set-authorization-name** 不可用于 Elytron 客户端配置文件中的 **set-anonymous**。相反，如果您配置 **set-authorization-name**，还必须为授权身份指定 **set-user-name**。

4.2.4. 将 Elytron 客户端与部署到 JBoss EAP 的客户端搭配使用

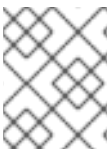
部署到 JBoss EAP 的客户端还可以利用 Elytron 客户端。**AuthenticationContext** 会自动从 JBoss EAP 配置 **中的 default-authentication-context** 设置解析和创建。如果没有配置 **default-authentication-context**，但您的部署中包含 **wildfly-config.xml** 文件，或使用 wildfly.config.url 系统属性设置 **wildfly.config.url** 系统属性，则 **AuthenticationContext** 会自动从该文件解析和创建。

示例：设置默认身份验证上下文

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

若要在部署外加载配置文件，您可以使用 **parseAuthenticationClientConfiguration(URI)** 方法。此方法返回 **AuthenticationContext**，您可以使用 编程方法 在客户端代码中使用它。

此外，客户端还将从 **elytron** 子系统提供的客户端配置中自动解析和创建 **AuthenticationContext**。**elytron** 子系统中的客户端配置还可以利用 **elytron** 子系统中定义的其他组件，如凭据存储。如果客户端配置同时由部署和 **elytron** 子系统提供，则使用 **elytron** 子系统的配置。



注意

只有在此 **身份验证-context** 设为 **elytron** 子系统的默认值时，才能使用 **elytron** 子系统中的 **AuthenticationContext**。

4.2.5. 使用 wildfly-config.xml 文件配置 Jakarta 管理客户端

从 JBoss EAP 7.1 开始，可使用 **wildfly-config.xml** 文件配置 Jakarta 管理客户端，包括 JConsole。在启动 Jakarta 管理客户端时，您可以使用 **-Dwildfly.config.url** 系统属性指定配置文件的文件路径。

-

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

注意

使用 JConsole 时，**-Dwildfly.config.url** 系统属性必须加上 **-J** 前缀，例如：

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

如需更多信息，请参阅 JBoss EAP *开发指南* **中的使用 wildfly-config.xml** 文件进行客户端配置。

4.2.6. 使用 ElytronAuthenticator 传播身份

警告

由于 Java 8 中已知的凭证限制，不支持或推荐在 JBoss EAP 中使用 **ElytronAuthenticator**。在使用此类传播身份时，请注意以下限制：

- 由于 Java 8 设计的限制，安全身份传播不适用于调用受保护 servlet。
- 不要在服务器上使用 **ElytronAuthenticator**，如 Jakarta Enterprise Beans。
- 凭据缓存可能会影响其在独立客户端 JVM 中的使用。

JBoss EAP 7.1 引入了 **ElytronAuthenticator** 类，它使用当前安全上下文来执行身份验证。org.wildfly.security.auth.util.ElytronAuthenticator 类是 java.net.Authenticator 实施。

- 它有一个构造器 **ElytronAuthenticator（）**，用于构建新实例。
- 它有一个方法 **getPasswordAuthentication（）**，它返回 **PasswordAuthentication** 实例。

以下是客户端代码示例，它创建并使用 **ElytronAuthenticator** 类向服务器传播身份：

示例：使用 ElytronAuthenticator 代码

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

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

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

```
            return new BufferedReader(new InputStreamReader(inputStream)).lines().findFirst().orElse(null);
        }
    });
```

4.3. 配置可信安全域流

对于任何安全调用，都为安全域建立了安全身份。在处理调用时，Sec **ureIdentity** 与当前的线程关联。对于同一安全域中 **getCurrentSecurityIdentity（）** 的后续调用，将返回关联的身份。

在应用服务器中，可以有多个 **SecurityDomain** 实例用于单个调用或线程。每个 **SecurityDomain** 实例都可以与不同的 **SecurityIdentity** 关联。当您调用该安全域的 **getCurrentSecurityIdentity（）** 方法时，将返回正确的安全身份。部署可以在请求处理期间调用其他部署。每一部署与单个安全域关联。如果调用的部署使用相同的安全域，则当前安全身份的单一安全域的概念仍然存在。但是，每一部署都可以引用自己的安全域。

可以将与安全域关联的安全身份导入到另一个安全域，如下一节所述。

导入安全身份

要从安全域导入安全身份到另一个安全域，以获取此域的安全身份，有三种处理流程：

相同的安全域

安全域始终可以导入自己的安全身份。在这种情况下，安全域始终信任自身。

Common Security Realm

在导入过程中，安全域从所导入的安全身份中提取主体，通过其配置的主体转换器和域映射器传递，并将其映射到该安全域中的身份。如果在创建身份的安全域中使用相同的安全域，则它们都使用相同的底层身份支持，并且导入也被接受。

可信安全域

如果身份映射成功，但没有通用的安全域，则将测试处理导入的安全域，以查看它是否信任原始的安全域。如果这样做，则接受导入。

注意

该身份必须存在于处理导入的安全域中。安全身份绝对不会被完全信任。

流

安全域可以配置为自动将其安全身份输出到其他安全域。

在安全域中，如果建立安全身份并用于当前调用，则会迭代溢出安全域列表，并且为每个域导入安全身份。

此模型更适合使用不同安全域对部署的多次调用，例如，当 Web 应用使用通用安全域调用五个不同的 Jakarta 企业 Beans 时。

第 5 章 使用 LDAP 保护管理接口

管理接口可以针对 LDAP 服务器（包括 Microsoft Active Directory）进行身份验证。这通过使用 LDAP 身份验证器来完成。LDAP 身份验证器首先通过建立与远程目录服务器的连接（使用出站 LDAP 连接）运行。然后，它将使用用户传递给身份验证系统的用户名执行搜索，以查找 LDAP 记录的完全限定可分辨名称(DN)。如果成功，将使用用户的 DN 作为凭证和用户提供的密码来创建新的连接。如果这第二个连接和 LDAP 服务器的身份验证成功，则 DN 被验证为有效并且身份验证成功。

注意

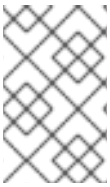
使用 LDAP 保护管理接口将身份验证从摘要更改为 BASIC/Plain，默认情况下，会导致用户名和密码通过网络进行未加密发送。可以在出站连接上启用 SSL/TLS 来加密此流量，并避免在明文中发送此信息。

重要

如果传统安全域使用 LDAP 服务器来执行身份验证，例如使用 LDAP 保护管理接口，JBoss EAP 将返回 500 或内部服务器错误， 如果该 LDAP 服务器无法访问，则返回错误代码。此行为与之前版本的 JBoss EAP 不同，后者在相同的条件下返回 **401** 或未授权的错误代码。

5.1. 使用 ELYTRON

您可以使用 LDAP 和 **elytron** 子系统来保护管理接口，其方式与使用任何身份存储相同。有关将身份存储用于 **elytron** 子系统安全性的信息，请参见 *《如何使用 配置服务器安全性的新身份存储》的安全管理接口*。例如，使用 LDAP 保护管理控制台：



注意

如果 JBoss EAP 服务器没有读取密码的权限，例如使用 Active Directory LDAP 服务器时，需要在定义的 LDAP 域中 **将直接验证** 设置为 **true**。此属性允许直接在 LDAP 服务器上执行验证，而非 JBoss EAP 服务器。

LDAP 身份服务示例

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

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

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

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

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

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

reload
```

5.1.1. 将 Elytron 用于出站 LDAP 连接的双向 SSL/TLS

使用 LDAP 保护管理接口时，您可以将出站 LDAP 连接配置为使用双向 SSL/TLS。为此，请创建一个 **ssl-context**，并将其添加到您的 **ldap -realm 使用的dir- context** 中。在如何使用 *配置服务器安全性 的 Elytron Subsystem 部分为应用启用双向 SSL/TLS 中介绍了 创建 双向 SSL/TLS。*

警告

红帽建议显式禁用 SSLv2、SSLv3 和 TLSv1.0，以便在所有受影响的软件包中明确禁用 TLSv1.1 或 TLSv1.2。

5.2. 使用传统核心管理身份验证

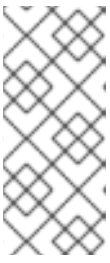
要使用 LDAP 目录服务器作为使用旧 **安全** 子系统管理接口的身份验证源，必须执行以下步骤：

1. 创建与 LDAP 服务器的出站连接。
   创建出站 LDAP 连接的目的是允许安全域（和 JBoss EAP 实例）建立与 LDAP 服务器的连接。这类似于创建数据源以用于安全域中的 **Database** 登录模块。

   LDAP 出站连接允许以下属性：

属性	必填	描述
url	是	目录服务器的 URL 地址。
search-dn	否	被授权执行搜索的用户的完全可分辨名称(DN)。
search-credential	否	有权执行搜索的用户密码。此元素支持的属性有： • **store** - 对凭据存储的引用，以获取搜索凭据。 • **alias** - 引用存储中的凭证别名。 • **type** - 要从凭证存储获取的凭证类型的完全限定类名称。 • **clear-text** - 不引用凭据存储，此属性可用于指定明文密码。

属性	必填	描述
initial-context-factory	否	建立连接时要使用的初始上下文工厂。默认为 **com.sun.jndi.ldap.LdapCtxFactory**。
security-realm	否	要引用的安全域，以获取在建立连接时要使用的已配置 **SSLContext**。
引用	否	指定在进行搜索时遇到引用时的行为。有效的选项包括 **IGNORE**、FO**LL**OW 和 **THROW**。 ● **IGNORE** ：默认选项。忽略引用。 ● **FOLLOW** ：搜索过程中遇到参考时，使用的 **DirContext** 将尝试紧随该引用。这假设相同的连接设置可用于连接第二台服务器，并且引用中使用的名称可被访问。 ● **THROW** ： **DirContext** 将抛出异常 **LdapReferralException**，以指示需要引用。安全域将处理并尝试识别要用于引用的替代连接。
always-send-client-cert	否	默认情况下，验证用户凭据时不会发送服务器的客户端证书。如果设置为 **true**，它将始终被发送。
handles-referrals-for	否	指定连接可处理的引用。如果指定 URI 列表，它们应当用空格分开。这允许在需要不同凭证后续引用时定义和使用与连接属性的连接。如果需要不同的凭据才能对第二服务器进行身份验证，或者服务器返回无法从 JBoss EAP 安装访问的引用中的名称，并且可以替换替代地址，这非常有用。



注意

search-dn 和 **search-credential** 与用户的用户名和密码不同。此处提供的信息专门用于在 JBoss EAP 实例和 LDAP 服务器之间建立初始连接。此连接允许 JBoss EAP 对尝试进行身份验证的用户的 DN 执行后续搜索。用户的 DN 是搜索的结果，它试图进行身份验证，并且他们提供的密码用于建立用于完成身份验证流程的单独第二个连接。

根据以下示例 LDAP 服务器，以下是用于配置出站 LDAP 连接的管理 CLI 命令：

表 5.1. LDAP 服务器示例

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

用于添加出站连接的 CLI

```
/core-service=management/ldap-connection=ldap-connection/:add(search-
credential=myPass,url=ldap://127.0.0.1:389,search-dn="cn=search,dc=acme,dc=com")

reload
```

注意

这会在 JBoss EAP 实例和 LDAP 服务器之间创建未加密的连接。有关使用 SSL/TLS 设置加密连接的详情，请参考对出站 LDAP 连接使用 SSL/TLS。

2. 创建一个新的启用 LDAP 的安全域。
创建出站 LDAP 连接后，必须创建一个新的支持 LDAP 的安全域才能使用它。

LDAP 安全域具有以下配置属性：

属性	描述
连接	用于连接 LDAP 目录的出站连接中定义的连接名称。
base-dn	用于开始搜索用户的上下文的 DN。
递归	搜索应在整个 LDAP 目录树中递归，还是仅搜索指定的上下文。默认值为 **false**。
user-dn	包含 DN 的用户属性。这随后用于测试用户能够完成的身份验证。默认值为 to **dn**。
allow-empty-passwords	此属性确定是否接受空密码。默认值为 **false**。
username-attribute	要搜索用户的属性名称。此过滤器执行简单的搜索，其中用户输入的用户名与指定的属性匹配。

属性	描述
advanced-filter	用于根据提供的用户 ID 搜索用户的完全定义过滤器。此属性包含标准 LDAP 语法中的过滤器查询。过滤器必须包含以下格式的变量：**{0}**。之后，这会被用户提供的用户名替代。如需更多详细信息和 **advanced-filter** 示例，请参阅 组合 LDAP 和 RBAC for Authorization 部分。

警告

确保不允许空 LDAP 密码非常重要，因为它是一个严重的安全问题。除非环境中特别需要此行为，否则请确保不允许空密码，并且 *allow-empty-passwords* 始终为 false。

以下是用于使用 **ldap-connection** 出站 LDAP 连接配置支持 LDAP 的安全域的管理 CLI 命令。

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

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

reload
```

3. 在管理界面中引用新的安全域。创建安全域并使用出站 LDAP 连接后，管理接口必须引用新的安全域。

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

注意

显示的管理 CLI 命令假定您在运行 JBoss EAP 单机服务器。有关将管理 CLI 用于 JBoss EAP 受管域的更多详细信息，请参见 JBoss EAP *管理 CLI 指南*。

5.2.1. 将双向 SSL/TLS 用于出站 LDAP 连接

按照以下步骤创建由 SSL/TLS 保护的出站 LDAP 连接：

警告

红帽建议显式禁用 SSLv2、SSLv3 和 TLSv1.0，以便在所有受影响的软件包中明确禁用 TLSv1.1 或 TLSv1.2。

1. 配置要使用的出站 LDAP 连接的安全域。
   安全域必须包含一个密钥存储，配置有 JBoss EAP 服务器将用于解密自身和 LDAP 服务器之间通信的密钥存储。此密钥存储还允许 JBoss EAP 实例根据 LDAP 服务器验证其自身。安全域还必须包含包含 LDAP 服务器证书的信任存储，或者用于签署 LDAP 服务器证书的证书颁发机构的证书。有关配置密钥存储和信任存储以及创建使用它们的安全域的说明，*请参阅 JBoss EAP 如何配置服务器安全* 指南中的双向 SSL/TLS。

2. 使用 SSL/TLS URL 和安全域创建一个出站 LDAP 连接。
   与使用传统 核心管理身份验证 中定义的进程类似，应创建出站 LDAP 连接，但是将 SSL/TLS URL 用于 LDAP 服务器和 SSL/TLS 安全域。

   为 LDAP 服务器创建出站 LDAP 连接和 SSL/TLS 安全域后，需要利用该信息更新出站 LDAP 连接。

   使用 SSL/TLS URL 添加出站连接的 CLI 示例

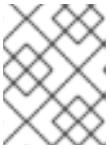
   ```
   /core-service=management/ldap-connection=ldap-connection/:add(search-credential=myPass, url=ldaps://LDAP_HOST:LDAP_PORT, search-dn="cn=search,dc=acme,dc=com")
   ```

   使用 SSL/TLS 证书添加安全域

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

   reload
   ```

3. 创建一个新的安全域，它使用出站 LDAP 连接供管理接口使用。
   按照 *创建新 LDAP 启用安全域* 的步骤，并根据 *使用传统 核心管理身份验证中的流程引用管理 界面中的新安全域*。



注意

显示的管理 CLI 命令假定您在运行 JBoss EAP 单机服务器。有关将管理 CLI 用于 JBoss EAP 受管域的更多详细信息，请参见 JBoss EAP *管理 CLI 指南*。

5.3. LDAP 和 RBAC

RBAC（基于角色的访问控制）是一种为管理用户指定一组权限（角色）的机制。这允许用户获得不同的管理职责，而无需给予他们完全、不受限制的访问权限。有关 RBAC 的详情，请参阅 JBoss EAP 安全架构 *指南* 中的基于角色的访问控制小节。

RBAC 仅用于授权，身份验证需单独处理。由于 LDAP 可以用于身份验证和授权，因此可使用以下方式配置 JBoss EAP：

- 仅将 RBAC 用于授权，并且仅将 LDAP 或其他机制用于身份验证。
- 使用 RBAC 和 LDAP 在管理界面中做出授权决策。

5.3.1. 独立使用 LDAP 和 RBAC

JBoss EAP 允许在安全域中独立配置身份验证和授权。这样，可以将 LDAP 配置为身份验证机制和 RBAC，并配置为授权机制。如果以这种方式配置，用户尝试访问管理界面时，将首先使用配置的 LDAP 服务器进行身份验证。如果成功，则用户的角色和配置权限将仅使用 RBAC，而不考虑 LDAP 服务器中的任何组信息。

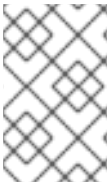
有关仅将 RBAC 用作管理接口的授权机制的更多详细信息，请参阅 *如何为 JBoss EAP 配置服务器* 安全性。有关配置 LDAP 以使用管理接口进行身份验证的更多详细信息，请参见 上一节。

5.3.2. 组合 LDAP 和 RBAC 进行授权

使用 LDAP 服务器或使用属性文件进行身份验证的用户可以是用户组的成员。用户组只是可分配给一个或多个用户的任意标签。RBAC 可以配置为使用此组信息自动为用户分配角色或将用户从角色中排除。

LDAP 目录包含用户帐户和组的条目，可通过属性来引用。根据 LDAP 服务器配置，用户实体可以映射用户通过 **memberOf** 属性属于的组；组实体可以通过 **uniqueMember** 属性或两者的组合来映射属于该用户的组。用户通过 LDAP 服务器成功身份验证后，将执行组搜索来加载该用户的组信息。根据使用的目录服务器，组搜索可使用其 SN（通常是身份验证中使用的用户名），或者通过使用目录中用户条目的 DN 执行。将 LDAP 设置为安全域中的授权机制时，会配置组搜索(**group-search)以及用户名和可分辨名称（用户名到dn**）之间的映射。

从 LDAP 服务器确定用户组成员资格信息后，将在 RBAC 配置中使用映射来确定用户具有哪些角色。此映射被配置为明确包含或排除组以及单个用户。

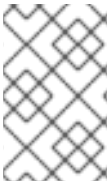


注意

用户连接到服务器的身份验证步骤始终先发生。成功验证用户后，服务器将加载用户的组。身份验证步骤和授权步骤各自都需要连接 LDAP 服务器。安全域通过为组加载步骤重复利用身份验证连接来优化此过程。

5.3.2.1. 使用 group-search

搜索组成员资格信息时可使用两种不同的样式：组先于组，*组为 Principal*。组的主体具有用户的条目，其中包含使用 **memberOf** 属性对其所属组的引用。组到 Principal 具有组的条目，其条目包含对属于其成员的用户的引用（使用 **uniqueMember** 属性）。



注意

JBoss EAP 支持组以及主要搜索的组(Principal)支持，但建议将组 Principal 改为 Principal。如果使用主要到组，则可以通过读取已知区分名称的属性直接加载组信息，而无需执行任何搜索。组到 Principal 需要大量的搜索来识别所有引用用户的组。

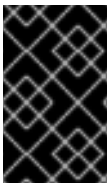
Principal 到 Group 和 Group to Principal 使用 **group-search**，它包含以下属性：

属性	描述

属性	描述
group-name	此属性用于指定应当用于作为用户所属组列表返回的组名的格式。这可以是组名的简单形式，也可以是组的可分辨名称。如果区分名称是必需的，可将此属性设置为 **DISTINGUISHED_NAME**。默认值为 **SIMPLE**。
迭代	此属性用于指示在确定用户所属组后，它也应该根据组反复搜索，以识别这些组所属的组。如果启用了迭代搜索，它将持续到检测到任何其他组或循环时到达非成员的组。默认值为 **false**。
group-dn-attribute	在组的条目中，属性是其可分辨的名称。默认值为 to **dn**。
group-name-attribute	在组的条目中，属性是其简单名称。默认为 **uid**。

注意

cyclic 组成员身份不是问题。将保留每个搜索的记录，以防止已搜索的组再次被搜索。



重要

若要使迭代搜索起作用，组条目需要与用户条目相同。然后，使用相同方法识别用户所属组。如果组成员身份的组、用于交叉引用的属性名称或参考方向有变化时，则无法做到这一点。

组搜索的组主体(memberOf)

例如，一个示例是 Group **One** 的成员，而 **GroupOne** 又是 **GroupFive** 的成员。组成员身份将通过在成员级别使用 **memberOf** 属性来显示。这意味着，**TestUserOne** 会将 **memberOf** 属性设置为 **GroupOne** 的 the **dn**。反过来，**GroupOne** 会将 **memberOf** 属性设置为 **GroupFive** 的 The **dn**。

要使用这种类型的搜索，将 **principal-to-group** 元素添加到 **group-search** 元素中：

组、memberOf、配置主体

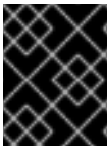
```
/core-service=management/security-realm=ldap-security-realm:add

batch

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

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

run-batch
```



重要

上面的示例假定您已定义了 **ldap-connection**。您还需要配置 本节前面 涵盖的身份验证机制。

注意 **group-attribute** 属性与 **group-search=principal-to-group** 一起使用。作为参考：

表 5.2. principal-to-group

属性	描述
group-attribute	用户条目上与用户所属组的可分辨名称匹配的属性名称。默认值为 **memberOf**。
preference-original-connection	这个值用于指示在引用后应该使用的组信息。每次加载主体时，来自每个组成员资格的属性都会随后被加载。每次加载属性时，都可以使用来自最后一次引用的原始连接或连接。默认值为 **true**。

Group to Principal, uniqueMember, group Search

考虑与 Principal 相同的示例，其中 user **TestUserOne** 是 **GroupOne** 的成员，而 **GroupOne** 又是 **GroupFive** 的成员。不过，在这种情况下，组成员资格将通过在组级别上设置的 **uniqueMember** 属性来显示。这意味着 **GroupFive** 会将 **uniqueMember** 设置为 **GroupOne** 的 The **dn**。反过来，**GroupOne** 会将 **uniqueMember** 设置为 **Test UserOne 的值**。

要使用这种类型的搜索，在 **group- search 元素中添加 group-to- principal** 元素：

Group to Principal, uniqueMember, Configuration

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

batch

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

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

run-batch
```

重要

上面的示例假定您已定义了 **ldap-connection**。您还需要配置 本节前面 涵盖的身份验证机制。

注意 **principal-attribute** 属性用于 **group-search=group-to-principal**。**Group-to-principal** 用于定义如何搜索引用用户条目的组，而 **principal-attribute** 则用于定义引用主体的组条目。

作为参考：

表 5.3. group-to-principal

属性	描述
base-dn	用于开始搜索的可分辨上下文名称。
递归	是否也搜索子上下文。默认值为 **false**。
search-by	搜索中使用的角色名称的形式。有效值为 **SIMPLE** 和 **DISTINGUISHED_NAME**。默认为 **DISTINGUISHED_NAME**。
preference-original-connection	这个值用于指示在引用后应该使用的组信息。每次加载主体时，来自每个组成员资格的属性都会随后被加载。每次加载属性时，都可以使用来自最后一次引用的原始连接或连接。

表 5.4. membership-filter

属性	描述
principal-attribute	引用用户条目的组条目上的 属性名称。默认为 **member**。

5.3.2.2. 使用用户名to-dn

可以在授权部分中定义规则，将用户的简单用户名转换为其可分辨名称。**username-to-dn** 元素指定如何将用户名映射到 LDAP 目录中条目的可分辨名称。这个元素 **是可选的**，只有在以下两者都满足时才需要：

- 身份验证和授权步骤针对不同的 LDAP 服务器。
- 组搜索使用可分辨名称。

注意

如果已执行 LDAP 身份验证，可以使用身份验证期间发现的 DN，这也适用于安全域支持 LDAP 和 Kerberos 身份验证以及 Kerberos 需要转换的实例。

它包含以下属性：

表 5.5. username-to-dn

属性	描述
force	当 force 属性设置为 **false** 时，在授权查询期间用于区分名称映射的用户名的结果会被缓存，并在授权查询期间重复使用。当强制为 **true** 时，加载组时会在授权期间再次执行搜索。这通常在不同服务器执行身份验证和授权时执行。

使用以下方法之一配置 **用户名至dn** ：

username-is-dn

这将指定远程用户输入的用户名是用户的可分辨名称。

username-is-dn 示例

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

batch

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

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

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

run-batch
```

这将定义 1:1 映射，并且没有额外的配置。

username-filter

将搜索指定属性与提供的用户名匹配。

username-filter 示例

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

batch

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

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

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

run-batch
```

属性	描述
base-dn	要开始搜索的可分辨上下文名称。

属性	描述
递归	搜索是否扩展至子上下文。默认值为 **false**。
attribute	用户条目的属性，以尝试和匹配提供的用户名。默认为 **uid**。
user-dn-attribute	要读取的属性以获取用户的可分辨名称。默认值为 to **dn**。

advanced-filter

此选项使用自定义过滤器来查找用户的可分辨名称。

advanced-filter 示例

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

batch

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

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

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

run-batch
```

对于与 **username-filter 示例中** 的那些属性，其含义和默认值相同。还有一个额外的属性：

属性	描述
filter	用于搜索用户条目的自定义过滤器，用户名将在 **{0}** 占位符中替换。

重要

这必须在过滤器定义后保持有效，以便在使用任何特殊字符（如 **&**）时确保使用正确的格式。例如 **和 字符： 和 字符**。

5.3.2.3. 将 LDAP 组信息映射到 RBAC 角色

创建与 LDAP 服务器的连接并正确配置了组搜索后，需要在 LDAP 组和 RBAC 角色之间创建映射。此映射可以是包含的，也可以是独占性的，并且能够根据用户的组成员资格自动分配给用户一个或多个角色。

警告

如果尚未配置 RBAC，请在这样做时密切关注，特别是切换到新创建的 LDAP 域时。在没有正确配置用户和角色的情况下启用 RBAC 可能会导致管理员无法登录 JBoss EAP 管理界面。

注意

显示的管理 CLI 命令假定您在运行 JBoss EAP 单机服务器。有关将管理 CLI 用于 JBoss EAP 受管域的更多详细信息，请参见 JBoss EAP *管理 CLI 指南*。

确保已启用并配置 RBAC

在可以使用 LDAP 和 RBAC 角色之间的映射前，RBAC 必须启用并初始配置。

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

它应产生以下结果：

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

有关启用和配置 RBAC 的更多信息，请参阅 *如何为 JBoss EAP 配置服务器安全性* 中的 基于角色的访问控制 。

验证现有角色列表

使用 **read-children-names** 操作获取配置的角色的完整列表：

```
/core-service=management/access=authorization:read-children-names(child-type=role-mapping)
```

这应该会产生一个角色列表：

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

另外，可以检查角色的所有现有映射：

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

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

```
  {
    "include-all" => false,
    "exclude" => undefined,
    "include" => {
      "user-theboss" => {
        "name" => "theboss",
        "realm" => undefined,
        "type" => "USER"
      },
      "user-harold" => {
        "name" => "harold",
        "realm" => undefined,
        "type" => "USER"
      },
      "group-SysOps" => {
        "name" => "SysOps",
        "realm" => undefined,
        "type" => "GROUP"
      }
    }
  }
}
```

配置 Role-Mapping 条目

如果角色还没有 **Role-Mapping** 条目，则需要创建一个。例如：

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

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

添加角色映射：

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

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

验证：

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

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

```
    "include" => undefined
  }
}
```

将组添加到 Role for Inclusion 和 Exclusion

可以添加组以包含或从角色中排除。

注意

排除映射具有优先权或包含映射。

为包含添加组：

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

为排除添加组：

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

检查结果：

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

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

删除组，使其无法排除或包含在 RBAC 角色绑定中

从包含中删除组：

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

要删除组，请排除：

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

5.4. 启用缓存

安全域还提供针对身份验证和组加载缓存 LDAP 查询结果的功能。这在某些情况下允许不同用户在多个搜索之间重复使用不同查询的结果，例如反复查询组的组成员资格信息。有三个不同的缓存，每个缓存都是单独配置并独立运行：

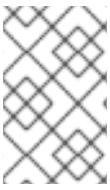
- 身份验证
- group-to-principal
- username-to-dn

5.4.1. 缓存配置

尽管缓存彼此独立，但所有三个缓存的配置方式都相同。每个缓存都提供以下配置选项：

属性	描述
type	这定义了缓存将遵循的驱除策略。选项为 **by-access-time** 和 **by-search-time**。**by-access-time** 在上次访问后经过了一定时间后从缓存中驱除项目。**按搜索时间** 根据它们在缓存中的项目的时间（不论它们上次访问权限如何）驱除项目。
eviction-time	根据策略，这定义用于驱除的时间（以秒为单位）。
cache-failures	这是一个布尔值，可启用/禁用失败搜索的缓存。这有可能防止 LDAP 服务器被同一失败搜索重复访问，但也有可能在缓存中填入缓存中搜索不存在的用户。此设置对于身份验证缓存尤为重要。
max-cache-size	这会定义缓存的最大大小（项目数），以内向中指示项目何时开始被驱除。旧项目从缓存中驱除，以便为新身份验证和搜索腾出空间，这意味着 **max-cache-size** 不会阻止新的身份验证尝试或搜索。

5.4.2. 示例



注意

此示例假定已创建了名为 **LDAPRealm** 的安全域。它连接到现有的 LDAP 服务器，并配置为进行身份验证和授权。用于显示当前 配置的命令包括在阅读当前缓存 配置中。有关创建使用 LDAP 的安全域的更多详细信息，请参阅使用传统核心管理身份验证。

基本配置示例

```
"core-service" : {
  "management" : {
    "security-realm" : {
      "LDAPRealm" : {
        "authentication" : {
          "ldap" : {
            "allow-empty-passwords" : false,
            "base-dn" : "...",
            "connection" : "MyLdapConnection",
            "recursive" : false,
            "user-dn" : "dn",
            "username-attribute" : "uid",
            "cache" : null
          }
        },
        "authorization" : {
          "ldap" : {
            "connection" : "MyLdapConnection",
            "group-search" : {
              "group-to-principal" : {
                "base-dn" : "...",
                "group-dn-attribute" : "dn",
                "group-name" : "SIMPLE",
                "group-name-attribute" : "uid",
                "iterative" : true,
                "principal-attribute" : "uniqueMember",
                "search-by" : "DISTINGUISHED_NAME",
                "cache" : null
              }
            },
            "username-to-dn" : {
              "username-filter" : {
                "attribute" : "uid",
                "base-dn" : "...",
                "force" : false,
                "recursive" : false,
                "user-dn-attribute" : "dn",
                "cache" : null
              }
            }
          },
        }
      }
    }
  }
}
```

在所有出现 **"cache": null** 的区域中，可以配置一个缓存：

身份验证

身份验证期间，使用此定义发现用户的可分辨名称，并尝试连接到 LDAP 服务器并验证其身份是否使用这些凭据来实现。

group-search 定义

有组搜索定义。在本例中，它是迭代搜索，因为在上面的示例配置中，它 被设置为 **true**。首先，将执行搜索以查找用户是以下直接成员的所有组：之后，将对每个组执行搜索，以确定它们是否与其他组成员资格。此过程将继续，直到检测到循环引用或最终组不属于任何其他组。

组搜索 中的用户名到dn 定义

组搜索取决于用户的可分辨名称的可用性。本节并不适用于所有情况，但可以作为第二次尝试来发现用户的可分辨名称。当支持第二种身份验证形式（如本地身份验证）时，这可能很有用，甚至也可以是必需的。

5.4.2.1. 阅读 Current Cache 配置

注意

本节和后续小节中使用的 CLI 命令使用 **LDAPRealm** 作为安全域的名称。这应当替换为正在配置的实际域的名称。

读取当前缓存配置的 CLI 命令

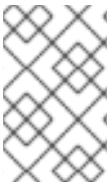
```
/core-service=management/security-realm=LDAPRealm:read-resource(recursive=true)
```

输出。

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

```
            "username-filter" => {
              "attribute" => "uid",
              "base-dn" => "ou=Users,dc=jboss,dc=org",
              "force" => true,
              "recursive" => false,
              "user-dn-attribute" => "dn",
              "cache" => undefined
            }
          }
        }
      },
      "plug-in" => undefined,
      "server-identity" => undefined
    }
  }
```

5.4.2.2. 启用缓存



注意

本节和后续小节中使用的管理 CLI 命令在安全域的身份验证部分中配置缓存，换句话说是 **authentication=ldap**/。授权部分中的缓存也可以通过更新命令的路径以类似的方式进行配置。

用于启用缓存的管理 CLI 命令

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

这个命令为验证添加 **by-access-time** 缓存，驱除时间为 300 秒（5 分钟），最大缓存大小为 100 个项目。此外，还将缓存失败的搜索。另外，也可以配置 **by-search-time** 缓存：

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

5.4.2.3. 检查现有缓存

用于内省现有缓存的管理 CLI 命令

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

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

include-runtime 属性添加 **cache-size**，它显示缓存中的当前项目数。它在上面的输出中是 **1**。

5.4.2.4. 测试现有缓存的内容

用于测试现有缓存内容的管理 CLI 命令

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

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

这表明 **TestUserOne** 的条目存在于缓存中。

5.4.2.5. 清空缓存

您可以从缓存中清空单个项目，或者清空整个缓存。

用于清理单项的管理 CLI 命令

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

用于查找 Entire Cache 的管理 CLI 命令

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

5.4.2.6. 删除缓存

用于删除缓存的管理 CLI 命令

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

reload
```

第 6 章 安全映射到安全域

当您向安全域添加安全映射时，这允许您在身份验证或授权发生后组合身份验证和授权信息，但在信息传递给应用之前。

要向现有安全域添加安全映射，必须配置 **代码**、**类型** 和相关模块选项。

示例：管理 CLI 命令，用于将 SimpleRoles 安全映射添加到现有的安全域

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

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

reload
```

code 字段是短名称，如 **SimpleRoles**、**PropertiesRoles**、**DatabaseRoles** 或安全映射模块的类名称。

type 字段指的是模块执行的映射类型，允许的值为 **主体**、**role**、**属性** 或 **凭证**。

其他资源

- 有关安全映射的更多信息，请参阅 JBoss EAP 中的安全映射。
- 有关可用安全映射模块及其模块选项的完整列表，请参阅 JBoss EAP 中的安全映射模块。

第 7 章 单机服务器和受管域中的服务器注意事项

使用 LDAP 服务器（包括 Microsoft Active Directory）设置身份管理基本上与它在单机服务器中使用，还是用于受管域中的服务器。通常，这也适用于利用安全域和安全域设置大多数身份存储。

与任何其他配置设置一样，独立配置驻留在 **standalone.xml** 文件中，受管域的配置则驻留在 **domain.xml** 和 **host.xml** 文件中。

附录 A. 参考资料

A.1. WILDFLY-CONFIG.XML 示例

wildlfly-config.xml 文件是客户端使用 Elytron Client 的一种方法，它允许客户端在连接 JBoss EAP 时使用安全信息。

示例： custom-config.xml

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

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

    <net-authenticator/>

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

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

其他资源

- 有关使用 Elytron 客户端的详情，请参阅使用 Elytron 客户端配置客户端身份验证。
- 有关如何使用 **wildfly-config.xml** 文件配置客户端的更多信息，请参阅使用 wildfly-config.xml 文件进行 客户端配置。

A.2. 单点登录属性

单点登录(SSO)身份验证机制配置。

下表提供了 **undertow** 子系统中 **application-security-domain** 的 **setting=single-sign-on** 资源的属性描述。

A.2.1. 单点登录

表 A.1. 单点登录 属性

属性	描述
client-ssl-context	对用来保护后端通道注销连接的 SSL 上下文的引用。
cookie-name	Cookie 的名称。默认值为 **JSESSIONIDSSO**。
credential-reference	凭据引用用于解密私钥条目。 **凭证引用** 具有以下属性： • **Ali** as：表示存储中的机密或凭证的别名。 • **clear-text** ：使用明文指定的机密。检查凭证存储向服务提供凭证或 secret 的方式。 • **存储** ：将别名存放到凭据的凭据存储的名称。 • **类型** ：此引用所指代的凭证类型。
domain	要使用的 Cookie 域。
http-only	设置 Cookie 的 **httpOnly** 属性：默认值为：**false**。

属性	描述
key-alias	用于签名并验证 back-channel 注销连接的私钥条目别名。
key-store	对含有私钥条目的密钥存储的引用。
路径	Cookie 路径.默认值为 /。
安全	用于设置 Cookie **的安全** 属性：默认值为：**false**。

其他资源

- 有关使用 **client-ssl-context** 的更多信息，请参阅使用 client-ssl-context。
- 有关 **凭证存储的更多信息，请参阅** Elytron 中的凭据存储。
- 有关如何创建密钥存储的更多信息，**请参阅创建** Elytron Keystore。

A.3. 密码映射器

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中构造密码：

- 清除文本
- 简单摘要
- salted 简单摘要
- bcrypt
- SCRAM
- 模块加密

密码映射器具有以下属性：

注意

第一列的索引是所有映射器的 **1**。

表 A.2. 密码映射器属性

映射程序名称	属性	加密方法
clear-password-mapper	• **password-index** 包含明文密码的列索引。	无加密.

映射程序名称	属性	加密方法
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**	使用简单的哈希机制。

映射程序名称	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** 指定哈希表示法。允许的值： ◦ **base64**（默认） ◦ **hex** • **salt-encoding** 指定 salt 的表示法。允许的值： ◦ **base64**（默认） ◦ **hex**	简单的哈希机制与 salt 一起使用。

映射程序名称	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** 包含所用迭代数量的列索引。 • **hash-encoding** 指定哈希表示法。允许的值： ◦ **base64**（默认） ◦ **hex** • **salt-encoding** 指定 salt 的表示法。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** 包含所用迭代数量的列索引。 • **hash-encoding** 指定哈希表示法。允许的值： ◦ **base64**（默认） ◦ **hex** • **salt-encoding** 指定 salt 的表示法。允许的值： ◦ **base64**（默认） ◦ **hex**	交换挑战响应身份验证机制用于哈希处理。

映射程序名称	属性	加密方法
modular-crypt-mapper	• **password-index** 包含加密密码的列的索引。	模块加密编码允许使用单一字符串对多段信息进行编码，如密码类型、散列或摘要、salt 和迭代计数。

更新于 2024-02-10