Red Hat JBoss Enterprise Application Platform 7.4

如何配置服务器安全性

保护红帽 JBoss 企业应用平台及其管理界面的说明。

Last Updated: 2024-02-10

Red Hat JBoss Enterprise Application Platform 7.4 如何配置服务器安全性

保护红帽 JBoss 企业应用平台及其管理界面的说明。

法律通告

摘要

本文档的目的是提供保护 Red Hat JBoss Enterprise Application Platform (JBoss EAP) 的实用指南。更具体地说，本指南详细介绍了如何在 JBoss EAP 上确保所有管理接口。在阅读本指南前，用户应阅读 Red Hat JBoss Enterprise Application Platform 的 安全架构 文档，并充分了解 JBoss EAP 如何处理安全性。本文档还利用了 JBoss EAP CLI 接口来执行配置更改。 完成本文档后，读者应该有一个可靠、了解如何保护 JBoss EAP 的安全。

目录

提供有关 JBOSS EAP 文档的反馈 3

使开源包含更多 4

第 1 章 保护服务器及其接口 5
1.1. 构建块 5
1.2. 如何保护管理接口 16
1.3. 为旧安全域配置安全审核 52
1.4. 使用 ELYTRON 进行安全审核 53
1.5. 为应用程序配置单向和双向 SSL/TLS 58
1.6. 使用 CLI 安全性命令为应用程序启用 HTTP 验证 70
1.7. SASL 身份验证机制 71
1.8. ELYTRON 与 MODCLUSTER 子系统集成 74
1.9. ELYTRON 与 JGROUPS 子系统集成 75
1.10. ELYTRON 与 REMOTING SUBSYSTEM 集成 75
1.11. 用于 SSL/TLS 的额外 ELYTRON 组件 82

第 2 章 保护托管域 99
2.1. 使用 ELYTRON为域控制器配置密码身份验证 99
2.2. 使用旧的内核管理验证为域控制器配置密码身份验证 100
2.3. 使用 ELYTRON 为域控制器配置 SSL 或 TLS 101
2.4. 使用 ELYTRON 在域模式中配置 SSL 105
2.5. 为旧的核心管理验证机制配置 SSL 或 TLS 106

第 3 章 保护服务器的用户及其管理界面 109
3.1. 使用 ELYTRON 进行用户身份验证 109
3.2. 使用 ELYTRON 进行身份传播和转发 117
3.3. 使用 ELYTRON 进行身份切换 126
3.4. 使用传统核心管理身份验证进行用户身份验证 127
3.5. 基于角色的访问控制 128

第 4 章 凭证的安全存储 145
4.1. 凭证存储在 ELYTRON 中 145
4.2. 密码 VAULT 184

第 5 章 JAVA 安全管理器 198
5.1. 关于 JAVA 安全管理器 198
5.2. 关于 JAVA 安全策略 198
5.3. 使用 JAVA 安全管理器运行 JBOSS EAP 199
5.4. 从之前的版本进行迁移前的注意事项 201

附录 A. 参考资料 202
A.1. ELYTRON 子系统组件参考 202
A.2. 配置您的环境以使用 BOUNCYCASTLE 供应商 240
A.3. SASL 身份验证机制参考 241
A.4. 安全授权参数 245
A.5. ELYTRON CLIENT SIDE ONE WAY 示例 246
A.6. ELYTRON 客户端方双向示例 247

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 保护服务器及其接口

1.1. 构建块

1.1.1. 接口和套接字绑定

JBoss EAP 使用其主机的接口，如 **inet-address** 和 **nic**，以及用于 Web 应用程序及其管理界面的通信端口。这些接口和端口通过 JBoss EAP 中的 **interfaces** 和 **socket-binding-groups** 设置进行定义和配置。

有关如何定义和配置 **interfaces** 和 **socket-binding-groups** 的更多信息，请参阅 JBoss EAP *配置指南*中的套接字绑定章节。

示例：接口

```
<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:127.0.0.1}"/>
  </interface>
</interfaces>
```

示例：Socket Binding Group

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
    <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
    <socket-binding name="management-https" interface="management"
port="${jboss.management.https.port:9993}"/>
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
        <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
</socket-binding-group>
```

1.1.2. Elytron 子系统

1.1.2.1. 启用跨服务器的 Elytron 安全

可以通过简单的方法在服务器间启用 Elytron。JBoss EAP 7.1 引入了一个示例配置脚本，它允许 Elytron 作为安全供应商。此脚本驻留在服务器安装中的 *EAP_HOME*/docs/examples 目录中。

执行以下命令，以启用服务器中的 Elytron 安全性。

```
$ EAP_HOME/bin/jboss-cli.sh --file=EAP_HOME/docs/examples/enable-elytron.cli
```

1.1.2.2. 创建 Elytron 安全域

elytron 子系统中的安全域与安全域结合使用，以及用于与应用进行核心管理身份验证。

重要

部署仅限于在每个部署中使用一个 Elytron 安全域。现在，可能需要多个旧安全域的情况可以使用单个 Elytron 安全域来完成。

使用管理 CLI 添加安全域

```
/subsystem=elytron/security-domain=domainName:add(realms=[{realm=realmName,role-
decoder=roleDecoderName}],default-realm=realmName,permission-
mapper=permissionMapperName,role-mapper=roleMapperName,...)
```

使用管理控制台添加安全域

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Other Settings**，再点击 **View**。
3. 选择 **SSL → Security Domain** 并使用 **Add** 按钮配置新的安全域。

1.1.2.3. 创建 Elytron Security Realm

elytron 子系统中的安全性域与安全域结合使用时，可用于核心管理身份验证以及与应用进行身份验证。安全域也根据其身份存储进行特别键入，例如 **jdbc-realm**、**filesystem-realm** 和**properties-realm** 等等。

使用管理 CLI 添加安全域

```
/subsystem=elytron/type-of-realm=realmName:add(....)
```

可以在上一节中找到添加特定域的示例，如 **jdbc-realm**、**filesystem-realm** 和 **properties-realm**。

使用管理控制台添加安全域

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Security Realms**，再点击 **View**。
3. 从 **Security Realm** 选项卡中选择适当的安全域类型，然后单击 **Add** 以配置新的安全域。

1.1.2.4. 创建 Elytron Role Decoder

角色解码器将安全域提供的身份属性转换为角色。角色解码器也专门根据其功能进行键入，如 **empty-role-decoder**、**simple-role-decoder** 和 **custom-role-decoder**。

使用管理 CLI 添加角色依赖项

```
/subsystem=elytron/ROLE-DECODER-TYPE=roleDeoderName:add(....)
```

使用管理控制台添加角色依赖项

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Mappers / Decoders**，再点击 **View**。
3. 点 **Role Decoder**，选择适当的角色解码器类型，然后点 **Add** 配置新角色解码器。

1.1.2.5. 将 source-address-role-decoder 添加到 elytron 子系统

您可以使用管理 CLI 或管理控制台将 **source-address-role-decoder** 角色解码器添加到 **elytron** 子系统中。通过在 **mappers** 元素中配置此角色解码器，您可以在做出授权决策时利用客户端的 IP 地址。

source-address-role-decoder 会提取客户端的 IP 地址，并检查是否与 **pattern** 属性或 **source-address** 属性中指定的 IP 地址匹配。如果客户端的 IP 地址与任一属性中指定的 IP 地址匹配，那么 **elytron** 使用 **roles** 属性将角色分配给用户。

注意

该流程使用管理 CLI 将 **source-address-role-decoder** 添加到 **elytron** 子系统中的 **mappers** 元素中。如果要使用管理控制台完成此任务，*请参阅附加资源* 部分提供的链接。

先决条件

- 记下服务器客户端的 IP 地址。

流程

1. 在 **elytron** 子系统中，使用管理 CLI 添加 **source-address-role-decoder**。对于 **source-address-role-decoder**，您必须为用户指定一个 IP 地址和至少一个角色。

   将 source-address-role-decoder 添加到 mappers 元素：

   ```
   /subsystem=elytron/source-address-role-decoder=decoder1:add(source-address="10.10.10.10", roles=["Administrator"])
   ```

   示例中显示了一个配置的 **source-address-role-decoder**，名为 **decoder1**。当客户端尝试连接到服务器时，**elytron** 子系统使用 **source-address-role-decoder** 检查客户端的 IP 地址是否与 **pattern** 属性或 **source-address** 属性中指定的 IP 地址匹配。在上例中，**source-address-role-decoder** 会检查客户端的 IP 地址是否为 **10.10.10.10**。如果客户端的 IP 地址为 **10.10.10.10**，则 **elytron** 使用 **roles** 属性将 **Administrator** 角色分配给用户。

   注意

   您可以配置 **source-address-role-decoder**，将特定的角色分配给需要从不同网络建立连接的用户。

2. 在 **security-domain** 中，引用 **role-decoder 属性中的已配置 source-** address- **role-decoder**。这可确保 Elytron 安全域在做出授权决策时使用 **source-address-role-decoder**。

   在 role-decoder 属性中引用配置的 source-address-role-decoder、解码r1 的示例：

   ```
   /subsystem=elytron/security-domain=domainName:add(role-decoder=decoder1,default-realm=realmName,realms=[{realm=realmName}])
   ```

其他资源

- 有关使用管理控制台添加角色解码器的详情，请参考 Elytron subsystem。
- 有关 **elytron** 子系统的详情，请参考 *安全架构*指南中的 Elytron 子系统。

1.1.3. 配置 elytron 子系统的 aggregate-role-decoder

aggregate-role-decoder 由两个或多个角色解码器组成。您可以使用 **aggregate-role-decoder** 来汇总从每个角色解码器返回的角色。

先决条件

- 在 **elytron** 子系统中配置至少两个角色解码器。

流程

- 将至少两个角色解码器添加到 **aggregate-role-decoder** 角色解码器中。

  将 decoder1 和 decoder2 添加到 aggregate-role-decoder 角色解码器中的示例：

  ```
  /subsystem=elytron/aggregate-role-decoder=aggregateDecoder:add(role-decoders=
  [decoder1, decoder2])
  ```

其他资源

- 有关 **elytron** 子系统中可用角色解码器的信息，*请参阅安全架构指南中的* Elytron 子系统资源。
- 有关创建角色解码器的详情，请参考 Elytron subsystem。

1.1.3.1. 创建 Elytron Rolemapper

角色映射程序在被解码到其他角色后映射角色。例如，在被解码后，角色名称或从主体中添加或删除特定角色。角色映射程序也根据功能进行输入，例如 **add-prefix-role-mapper**、**add-suffix-role-mapper** 以及 **constant-role-mapper**。

添加角色映射程序获取常规表单

```
/subsystem=elytron/ROLE-MAPPER-TYPE=roleMapperName:add(...)
```

使用管理控制台添加角色映射程序

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Mappers / Decoders**，再点击 **View**。
3. 单击 **Role Mapper**，选择适当的角色映射程序类型，然后单击 **Add** 以配置新角色映射程序。

1.1.3.2. 创建 Elytron 权限集

权限集可用于为身份分配权限。

使用管理 CLI 添加权限集

```
/subsystem=elytron/permission-set=PermissionSetName:add(permissions=[{class-name="...",
module="...", target-name="...", action="..."}...])
```

permissions 参数由一组权限组成，每个权限都具有以下属性：

- **class-name** 是权限的完全限定域名。这是唯一需要的权限属性。
- **module** 是一个可选模块，用于加载权限。
- **target-name** 是一个在被创建时传递给权限的一个可选目标名称。
- **action** 是传递到权限（组成）的可选操作。

1.1.3.3. 创建 Elytron 权限映射程序

除了分配给身份的角色外，也可以分配权限。权限映射程序将权限分配给身份。权限映射程序也根据其功能进行特别键入，如 **logical-permission-mapper**、简单**-permission-mapper** 以及 **custom-permission-mapper**。

使用管理 CLI 添加权限映射

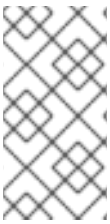
```
/subsystem=elytron/simple-permission-mapper=PermissionMapperName:add(...)
```

使用管理控制台添加权限映射程序

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Mappers / Decoders**，再点击 **View**。
3. 点 **Principal Decoder**，选择适当的主体解码器类型，然后点 **Add** 来配置新的主体解码器。

1.1.3.4. 创建身份验证配置

身份验证配置包含连接时要使用的凭证。如需有关身份验证配置的更多信息，请参阅 JBoss EAP 的 *How to Configure Identity Management* 中的 Configure Client Authentication with Elytron Client。



注意

您可以将 Elytron 安全域配置为使用访问用户的凭据，而不是凭据存储。例如，安全域可与 Kerberos 结合使用，以验证传入的用户。按照 配置 Elytron subsystem 中的说明：*如何使用 Kerberos 为 JBoss EAP 设置 SSO，并在 Kerberos* 安全工厂中设置 **get-kerberos-ticket=true**。

使用管理 CLI 添加身份验证配置

```
/subsystem=elytron/authentication-
configuration=AUTHENTICATION_CONFIGURATION_NAME:add(authentication-
name=AUTHENTICATION_NAME, credential-reference={clear-text=PASSWORD})
```

使用管理控制台添加身份验证配置

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Other Settings**，再点击 **View**。

3. 点 **Authentication → Authentication Configuration**，再点 **Add** 来配置新的身份验证配置。

有关 **authentication-configuration** 属性的完整列表，请参阅 Elytron 子系统组件参考。

1.1.3.5. 创建身份验证上下文

验证上下文包含一组规则以及用来建立连接的验证配置或 SSL 上下文。有关验证上下文的更多信息，请参阅 JBoss EAP 的 *How to Configure Identity Management* 中的 Configure Client Authentication with Elytron Client。

使用管理 CLI 添加身份验证上下文
可以使用以下管理 CLI 命令创建身份验证上下文：

```
/subsystem=elytron/authentication-context=AUTHENTICATION_CONTEXT_NAME:add()
```

通常，验证上下文将包含一组规则以及验证配置或 SSL 上下文。以下 CLI 命令演示了如何定义仅在主机名为 **localhost** 时的功能的身份验证上下文。

```
/subsystem=elytron/authentication-context=AUTHENTICATION_CONTEXT_NAME:add(match-rules=[{authentication-configuration=AUTHENTICATION_CONFIGURATION_NAME, match-host=localhost}])
```

使用管理控制台添加身份验证上下文

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Other Settings**，再点击 **View**。
3. 点 **Authentication → Authentication Context**，然后点击 **Add** 来配置新的身份验证上下文。

有关 **authentication-context** 属性的完整列表，请参阅 Elytron 子系统组件参考。

1.1.3.6. 创建 Elytron 身份验证工厂

身份验证工厂是用于特定验证机制的验证策略。身份验证工厂专门基于身份验证机制，如 **http-authentication-factory**、**sasl-authentication-factory** 和 **kerberos-security-factory**。

使用管理 CLI 添加身份验证工厂

```
/subsystem=elytron/AUTH-FACTORY-TYPE=authFactoryName:add(....)
```

使用管理控制台添加身份验证工厂

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Factories / Transformers**，再单击 **View**。
3. 单击 **HTTP Factories**、**SASL** 或 **Other factorsies**，选择相应的工厂类型，然后单击 **Add** 来配置新的工厂。

1.1.3.7. 创建 Elytron Keystore

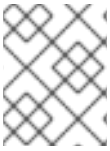
key-store 是密钥存储或信任存储的定义，包括密钥存储类型、其位置以及用于访问它的凭据。

要生成用于 **elytron** 子系统的示例密钥存储，请使用以下命令：

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

使用管理 CLI 添加密钥存储
要在 Elytron 中定义引用新密钥存储的 **key-store**，可执行以下管理 CLI 命令：此命令指定与提供的文件系统路径相关的密钥存储路径、用于访问密钥存储的凭据引用以及密钥存储类型。

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



注意

以上命令使用 **relative-to** 来引用密钥存储文件的位置。或者，您也可以指定 **路径中的** 密钥存储的完整路径，省略 **relative-to**。

使用管理控制台添加密钥存储

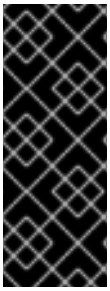
1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Other Settings**，再点击 **View**。
3. 点 **Stores → Key Store**，再点 **Add** 来配置一个新的密钥存储。

1.1.3.8. 创建 Elytron Key Manager

key-manager 引用 **key-store**，与 SSL 上下文结合使用。

使用管理 CLI 添加密钥管理器
以下命令指定要引用的底层密钥存储、初始化密钥管理器时使用的算法以及用于访问底层密钥存储中的条目的凭据引用。

```
/subsystem=elytron/key-manager=newKeyManager:add(key-store=KEY_STORE,credential-
reference={clear-text=secret})
```



重要

红帽没有指定上一命令中的 algorithm 属性，因为 Elytron 子系统使用 **KeyManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些关键管理器算法。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **SunX509** 作为密钥管理器算法属性。

使用管理控制台添加密钥管理器

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Other Settings**，再点击 **View**。
3. 点 **SSL → Key Manager** 并点 **Add** 来配置新的密钥管理器。

1.1.3.9. 创建 Elytron Truststore

要在 Elytron 中创建信任存储，请执行以下 CLI 命令：

```
/subsystem=elytron/key-store=default-trust-store:add(type=JKS, relative-to=jboss.server.config.dir,
path=application.truststore, credential-reference={clear-text=password})
```

要成功执行上述命令，必须在 ***EAP_HOME*/standalone/configuration** 目录中有一个 **application.truststore** 文件。如果端点的证书由 CA 签名，则信任存储必须包含与端点或证书链关联的证书。

红帽建议避免使用自签名证书。理想情况下，证书应由 CA 签名，您的信任存储应包含代表您的 **ROOT** 和中间 CA 的证书链。

1.1.3.10. 创建 Elytron Trust Manager

要在 Elytron 中定义信任管理器，请执行以下 CLI 命令：

```
/subsystem=elytron/trust-manager=default-trust-manager:add(key-store=TRUST-STORE-NAME)
```

这会将定义的信任存储设置为应用服务器信任的证书源。

1.1.3.11. 使用开箱即用的 Elytron 组件

JBoss EAP 提供一组在 **elytron** 子系统中配置的默认 Elytron 组件。您可以在 *安全架构*指南的 Out of Box 部分中找到有关这些预先配置的组件的详细信息。

1.1.3.11.1. 保护管理接口

您可以通过启用 JBoss EAP 来使用开箱即用的 Elytron 组件来保护 通过 Elytron 部分中的管理接口保护管理界面的详细信息。

1.1.3.11.2. 保护应用程序

elytron 子系统为 **http-authentication-factory** 提供 **application-http-authentication**，默认可用于保护应用。有关如何配置 **application-http-authentication** 的更多信息，请参阅 *安全架构指南*中的开箱即用部分。

要将应用程序配置为使用 **application-http-authentication**，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 安全 进行身份验证。您还可以使用 JBoss EAP *如何配置身份管理指南中的* 覆盖应用的身份验证配置 部分中的步骤来覆盖所有应用的默认行为。

1.1.3.11.3. 使用 SSL/TLS

JBoss EAP 使用传统的核心管理身份验证提供默认的单向 SSL/TLS 配置，但不在 **elytron** 子系统中提供。您可以在以下部分中找到有关使用 **elytron** 子系统配置 SSL/TLS 的更多详细信息，以及以下部分中的应用程序：

- 使用 Elytron subsystem 为管理接口启用单向 SSL/TLS
- 使用 Elytron subsystem 为管理接口启用两个 SSL/TLS
- 使用 Elytron subsystem 为应用程序启用单向 SSL/TLS
- 使用 Elytron subsystem 为应用程序启用两个 SSL/TLS

1.1.3.11.4. 将 Elytron 与其他子系统搭配使用

除了保护应用程序和管理界面外，Elytron 还与 JBoss EAP 中的其他子系统集成。

batch-jberet

您可以使用 Elytron 安全域配置 **batch-jberet** 子系统来运行批处理作业。如需更多信息，请参阅*配置指南*中的为批处理作业配置安全性。

datasources

您可以使用凭证存储或 Elytron 安全域在数据源定义中提供身份验证信息。如需更多信息，请参阅*配置指南*中的数据源安全性。

ejb3

您可以在 **ejb3** 子系统中为 Elytron 安全域创建映射，供部署引用。如需更多信息，请参阅 *开发 Jakarta Enterprise Beans* 应用中的使用 EJB 子系统集成。

iiop-openjdk

您可以使用 **elytron** 子系统使用 **iiop-openjdk** 子系统在客户端和服务器间配置 SSL/TLS。如需更多信息，请参阅 *配置指南*中的将 IIOP 配置为在 Elytron 子系统中使用 SSL/TLS 。

jca

您可以使用 **elytron-enabled** 属性为工作管理器启用 Elytron 安全性。有关更多信息，请参阅配置 *指南中的配置* JCA 子系统。

jgroups

您可以配置 **SYM_ENCRYPT** 和 **ASYM_ENCRYPT** 协议来引用 **elytron** 子系统中定义的密钥存储或凭证引用。如需更多信息，请参阅*配置指南*中的保护集群。

mail

您可以使用凭据存储在 **邮件** 子系统的服务器定义中提供身份验证信息。如需更多信息，请参阅*配置指南*中的使用凭据存储密码。

messaging-activemq

您可以保护与 **messaging-activemq** 子系统使用的远程连接的安全。如需更多信息，请参阅 *配置消息*中的使用 Elytron 子系统部分。

modcluster

您可以使用 Elytron 客户端 **ssl-context** 与使用 SSL/TLS 的负载均衡器通信。如需更多信息，请参阅 Elytron 与 ModCluster subsystem 集成。

remoting

您可以在 **remoting** 子系统中配置入站和出站连接，以引用 **elytron** 子系统中定义的验证上下文、SAS 身份验证工厂和 SSL 上下文。有关配置每种连接类型的完整详情，请参阅 Elytron 与 Remoting subsystem 集成。

resource-adapters

您可以使用 Elytron 保护到资源适配器的连接。在提交工作由工作管理器执行时，您可以启用安全性 inflow 建立安全凭证。如需更多信息，请参阅*配置指南*中的配置资源适配器以使用 Elytron 子系统。

undertow

您可以使用 **elytron** 子系统配置 SSL/TLS 和应用身份验证。有关配置应用程序身份验证的更多信息，请参阅 *如何配置身份管理*中的使用 SSL/TLS 和配置 Web 应用程序以使用 Elytron 或为应用程序使用传统的 Security 子系统。

1.1.3.12. 启用和禁用 Elytron 子系统

elytron 子系统使用默认 JBoss EAP 配置文件以及传统的 **Security** 子系统预先配置。

如果您使用尚未配置 **elytron** 子系统的配置集，您可以通过添加 **elytron** 扩展并启用 **elytron** 子系统来添加它。

添加 **elytron** 子系统所需的 **elytron** 扩展：

```
/extension=org.wildfly.extension.elytron:add()
```

在 JBoss EAP 中启用 **elytron** 子系统：

```
/subsystem=elytron:add

reload
```

在 JBoss EAP 中禁用 **elytron** 子系统：

```
/subsystem=elytron:remove

reload
```

重要

JBoss EAP 中的其他子系统可能对 **elytron** 子系统具有依赖性。如果在禁用这些依赖项前没有解决，则启动 JBoss EAP 时会看到错误。

1.1.4. 旧安全子系统

1.1.4.1. 禁用安全子系统

您可以通过执行**删除**子系统的操作来禁用 JBoss EAP 中的 **security** 子系统。

流程

- 在 JBoss EAP 中禁用 **Security** 子系统：

```
/subsystem=security:remove
```

重要

JBoss EAP 中的其他子系统可能对 **security** 子系统具有依赖性。如果在禁用这些依赖项前没有解决，则启动 JBoss EAP 时会看到错误。

1.1.4.2. 启用安全子系统

您可以通过执行子系统的 **add** 操作，在 JBoss EAP 中启用 **安全** 子系统。

流程

- 在 JBoss EAP 中启用 **Security** 子系统：

```
/subsystem=security:add
```

1.1.5. 旧安全域

JBoss EAP 使用安全域来定义身份验证和授权机制，如本地的 LDAP 属性，然后供管理接口使用。

示例：安全性域

```
<security-realms>
  <security-realm name="ManagementRealm">
    <authentication>
      <local default-user="$local" skip-group-loading="true"/>
      <properties path="mgmt-users.properties" relative-to="jboss.server.config.dir"/>
    </authentication>
    <authorization map-groups-to-roles="false">
      <properties path="mgmt-groups.properties" relative-to="jboss.server.config.dir"/>
    </authorization>
  </security-realm>
  <security-realm name="ApplicationRealm">
    <authentication>
      <local default-user="$local" allowed-users="*" skip-group-loading="true"/>
      <properties path="application-users.properties" relative-to="jboss.server.config.dir"/>
    </authentication>
    <authorization>
      <properties path="application-roles.properties" relative-to="jboss.server.config.dir"/>
    </authorization>
  </security-realm>
</security-realms>
```

注意

除了更新现有安全域外，JBoss EAP 还允许您创建新的安全域。您可以通过管理控制台创建新的安全域，并通过管理 CLI 调用以下命令：

```
/core-service=management/security-realm=<new_realm_name>:add()
```

如果创建新的安全域，并希望使用属性文件进行身份验证或授权，则必须创建一个新的属性文件，专门用于新的安全域。JBoss EAP 不重复使用其他安全域使用的现有文件，也不在配置中指定的新文件（如果它们不存在）。

其他资源

- 有关安全域的更多信息，请参阅 Security Realms。

1.1.6. 使用身份验证和套接字绑定来保护管理界面

您可以使用 **socket-binding**、**http-authentication-factory** 和 **http-upgrade** 的组合来保护使用 **elytron** 子系统的管理接口。另外，您还可以将 **socket-binding** 与 **security-realm** 搭配使用来保护使用旧核心管理身份验证的管理接口。您还可以禁用管理接口，并将接口的用户配置为拥有不同的角色和访问权限。

默认情况下，JBoss EAP 定义了一个 **http-interface** 来连接管理界面。

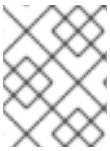
流程

- 显示服务器管理接口设置：

```
[standalone@localhost:9990 /] /core-service=management:read-resource(recursive=true)
{
    "outcome" => "success",
    "result" => {
        "access" => {...},
        "ldap-connection" => undefined,
        "management-interface" => {"http-interface" => {
            "allowed-origins" => undefined,
            "console-enabled" => true,
            "http-authentication-factory" => "management-http-authentication",
            "http-upgrade" => {
                "enabled" => true,
                "sasl-authentication-factory" => "management-sasl-authentication"
            },
            "http-upgrade-enabled" => true,
            "sasl-protocol" => "remote",
            "secure-socket-binding" => undefined,
            "security-realm" => undefined,
            "server-name" => undefined,
            "socket-binding" => "management-http",
            "ssl-context" => undefined
        }},
        "security-realm" => {...},
        "service" => undefined
    }
}
```

1.2. 如何保护管理接口

以下小节介绍了如何执行与保护 JBoss EAP 管理接口和相关子系统相关的各种操作。



注意

显示的管理 CLI 命令假定您正在运行 JBoss EAP 单机服务器。有关为 JBoss EAP 受管域使用管理 CLI 的详情，请查看 JBoss EAP *管理 CLI 指南*。

使用管理 CLI 集成 Elytron
可以使用来自 **elytron** 子系统的资源来保护管理界面，其方式与旧的安全域相同。

连接的 SSL 配置来自以下位置之一：

- 特定于 CLI 配置中的任何 SSL 配置。
- 自动提示用户接受服务器证书的默认 SSL 配置。
- java 系统属性。

可使用 **wildfly-config.xml** 文件修改客户端配置。

注意

如果设置 **-Dwildfly.config.url** 属性，客户端都可以使用任何文件进行配置。

1.2.1. 配置联网和端口

根据主机的配置，可将 JBoss EAP 配置为使用各种网络接口和端口。这使得 JBoss EAP 能够处理不同的主机、网络和防火墙要求。

其他资源

- 有关 JBoss EAP 使用的联网和端口的更多信息，以及如何配置这些设置，请参阅 JBoss EAP 配置 *指南中的网络和端口* 配置 部分。

1.2.2. 禁用管理控制台

其他客户端（如 JBoss 运营网络）操作使用 HTTP 接口来管理 JBoss EAP。为了继续使用这些服务，可能只禁用基于 Web 的管理控制台本身。这可以通过将 **console-enabled** 属性设置为 **false** 来完成。

流程

- 在 JBoss EAP 中禁用基于 Web 的管理控制台：

```
/core-service=management/management-interface=http-interface/:write-
attribute(name=console-enabled,value=false)
```

1.2.3. 禁用对 JMX 的远程访问

通过远程访问 **jmx** 子系统，可以远程触发 JDK 和应用程序管理操作。

流程

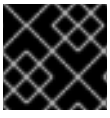
- 要在 JBoss EAP 中禁止远程访问 JMX，请删除 **jmx** 子系统中的补救连接器：

```
/subsystem=jmx/remoting-connector=jmx/:remove
```

1.2.4. silent 身份验证

JBoss EAP 的默认安装包含本地管理 CLI 用户的内部身份验证方法。这允许本地用户在不进行用户名或密码身份验证的情况下访问管理 CLI。可以启用此功能，以便本地用户在不需要身份验证的情况下运行管理 CLI 脚本。它被视为一个有用的功能，因为访问本地配置通常也让用户能够添加他们自己的用户详情或者禁用安全检查。

可在需要更大的安全控制的情况下禁用静默身份验证。这可以通过删除配置文件的 **security-realm** 属性中的 local 元素来实现。这适用于独立实例和受管域。



重要

只有在对 JBoss EAP 实例的影响及其配置被完全理解时，才应执行本地元素删除。

流程

1. 使用 **elytron** 子系统时删除静默身份验证：

```
[standalone@localhost:9990 /] /subsystem=elytron/sasl-authentication-factory=managenet-
sasl-authentication:read-resource
{
```

```
    "outcome" => "success",
    "result" => {
        "mechanism-configurations" => [
            {
                "mechanism-name" => "JBOSS-LOCAL-USER",
                "realm-mapper" => "local"
            },
            {
                "mechanism-name" => "DIGEST-MD5",
                "mechanism-realm-configurations" => [{"realm-name" => "ManagementRealm"}]
            }
        ],
        "sasl-server-factory" => "configured",
        "security-domain" => "ManagementDomain"
    }
}

[standalone@localhost:9990 /] /subsystem=elytron/sasl-authentication-factory=managenet-
sasl-authentication:list-remove(name=mechanism-configurations, index=0)

[standalone@localhost:9990 /] reload
```

2. 使用旧安全域时移除静默身份验证：

```
/core-service=management/security-realm=<realm_name>/authentication=local:remove
```

1.2.5. 使用 Elytron 子系统进行管理接口的单向 SSL/TLS

在 JBoss EAP 中，您可以使用 JBoss EAP 管理 CLI 或管理控制台为管理接口启用单向 SSL/TLS。

在管理 CLI 中，可以通过两种方式启用单向 SSL/TLS：

- 使用安全命令.
- 使用 **elytron** 子系统命令。

在管理控制台中，可以启用单向 SSL/TLS，如下所示：

- 使用管理控制台

1.2.5.1. 使用安全命令启用单向 SSL/TLS

security enable-ssl-management 命令可以用来为管理接口启用单向 SSL/TLS。

流程

- 在 CLI 中输入 **security enable-ssl-management --interactive** 命令。

  示例

```
security enable-ssl-management --interactive

Please provide required pieces of information to enable SSL:
Key-store file name (default management.keystore): keystore.jks
Password (blank generated): secret
```

```
What is your first and last name? [Unknown]: localhost
What is the name of your organizational unit? [Unknown]:
What is the name of your organization? [Unknown]:
What is the name of your City or Locality? [Unknown]:
What is the name of your State or Province? [Unknown]:
What is the two-letter country code for this unit? [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
correct y/n [y]?
Validity (in days, blank default): 365
Alias (blank generated): localhost
Enable SSL Mutual Authentication y/n (blank n): n

SSL options:
key store file: keystore.jks
distinguished name: CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown,
C=Unknown
password: secret
validity: 365
alias: localhost
Server keystore file keystore.jks, certificate file keystore.pem and keystore.csr file
will be generated in server configuration directory.
Do you confirm y/n :y
```

注意

- 执行命令后，管理 CLI 将重新加载服务器并重新连接该服务器。
- 您可以使用 **disable-ssl-management** 命令为管理接口禁用单向 SSL/TLS。

  ```
  security disable-ssl-management
  ```

  此命令不会删除 Elytron 资源。它将系统配置为使用 **ApplicationRealm** legacy security realm 进行其 SSL 配置。

1.2.5.2. 使用 Elytron 子系统命令启用单向 SSL/TLS

您可以使用 **elytron** 子系统命令为管理接口启用单向 SSL/TLS。

流程

1. 配置 **密钥存储**。

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

注意

以上命令使用 **relative-to** 来引用密钥存储文件的位置。或者，您也可以指定 **路径中的** 密钥存储的完整路径，省略 **relative-to**。

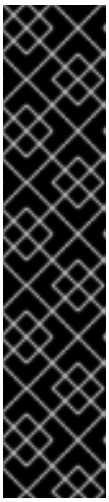
如果密钥存储文件尚不存在，则可以使用下列命令来生成示例密钥对：

```
/subsystem=elytron/key-store=httpsKS:generate-key-
```

```
pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,credential-reference={clear-text=secret},distinguished-name="CN=localhost")

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

2. 创建 **key-manager** 和 **server-ssl-context**。

```
/subsystem=elytron/key-manager=httpsKM:add(key-store=httpsKS,credential-reference={clear-text=secret})

/subsystem=elytron/server-ssl-context=httpsSSC:add(key-manager=httpsKM,protocols=["TLSv1.2"])
```



重要

红帽没有指定上一命令中的 algorithm 属性，因为 Elytron 子系统使用 **KeyManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些关键管理器算法。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **SunX509** 作为密钥管理器算法属性。

您还需要确定您要支持的 HTTPS 协议。上面的示例命令使用 **TLSv1.2**。

您可以使用 **cipher-suite-filter** 指定密码套件，**使用-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 **security** 子系统行为不同，它默认为遵循客户端密码套件顺序。

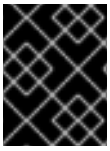
3. 在管理界面中启用 HTTPS。

```
/core-service=management/management-interface=http-interface:write-attribute(name=ssl-context, value=httpsSSC)

/core-service=management/management-interface=http-interface:write-attribute(name=secure-socket-binding, value=management-https)
```

4. 重新加载 JBoss EAP 实例。

```
reload
```

现在为管理界面启用了单向 SSL/TLS。



重要

如果您*同时*定义了 **security-realm** 和 **ssl-context**，JBoss EAP 将使用 **ssl-context** 提供的 SSL/TLS 配置。

其他资源

- key-store 属性
- key-manager 属性
- server-ssl-context 属性

1.2.5.3. 使用管理控制台启用单向 SSL/TLS

您可以使用管理控制台中的 SSL 向导为管理控制台中使用的管理界面启用 SSL。

流程

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 进入到 **Runtime**，单击适当的服务器名称。
3. 点服务器名称旁边的 **View**。
4. 点 **HTTP Manageme...** 以打开 **HTTP Management Interface** 配置页面。
5. 点**启用 SSL** 以启动向导。
   向导可以帮助您在以下场景中启用 SSL：
   - 您需要创建证书存储并生成自签名证书。
   - 您需要从 Let 的加密证书颁发机构获取证书。
   - 您已将证书存储在文件系统中，但没有密钥存储配置。
   - 您已经拥有使用有效证书存储的密钥存储配置。

使用向导，您可以选择性地为 mutual 验证创建信任存储。

1.2.6. 使用 Elytron 子系统进行管理接口的双向 SSL/TLS

在 JBoss EAP 中，可以使用安全命令或使用 **elytron** 子系统命令启用用于管理接口的双向 SSL/TLS。

要启用双向 SSL/TLS，首先您必须获取或生成客户端证书。您可以按照以下流程生成客户端证书：

- 生成客户端证书

然后您可以使用以下方法之一为管理接口启用双向 SSL/TLS：

- 使用安全命令启用双向 SSL/TLS
- 使用 Elytron 子系统命令启用双向 SSL/TLS

1.2.6.1. 生成客户端证书

您可以在 CLI 中使用 keytool 命令生成客户端证书。

流程

1. 生成客户端证书：

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

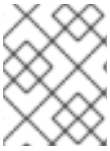
2. 导出客户端证书：

```
$ keytool -exportcert  -keystore client.keystore.jks -alias client -keypass secret -storepass
secret -file /path/to/client.cer
```

-

1.2.6.2. 使用安全命令启用双向 SSL/TLS

security enable-ssl-management 命令可以用来为管理接口启用双向 SSL/TLS。



注意

以下示例没有信任链的验证证书。如果您使用可信证书，则在没有问题的情况下可以验证客户端证书。

先决条件

- 您已配置了客户端密钥存储。
- 您已导出了服务器信任存储的证书。
  如需更多信息，请参阅生成客户端证书。

流程

- 在 CLI 中输入 **security enable-ssl-management --interactive** 命令。

  示例

```
security enable-ssl-management --interactive

Please provide required pieces of information to enable SSL:
Key-store file name (default management.keystore): server.keystore.jks
Password (blank generated): secret
What is your first and last name? [Unknown]: localhost
What is the name of your organizational unit? [Unknown]:
What is the name of your organization? [Unknown]:
What is the name of your City or Locality? [Unknown]:
What is the name of your State or Province? [Unknown]:
What is the two-letter country code for this unit? [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
correct y/n [y]?
Validity (in days, blank default): 365
Alias (blank generated): localhost
Enable SSL Mutual Authentication y/n (blank n): y
Client certificate (path to pem file): /path/to/client.cer
Validate certificate y/n (blank y): n
Trust-store file name (management.truststore): server.truststore.jks
Password (blank generated): secret

SSL options:
key store file: server.keystore.jks
distinguished name: CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown,
C=Unknown
password: secret
validity: 365
alias: localhost
client certificate: /path/to/client.cer
trust store file: server.trustore.jks
trust store password: secret
Server keystore file server.keystore.jks, certificate file server.pem and server.csr file will be
```

```
generated in server configuration directory.
Server truststore file server.trustore.jks will be generated in server configuration directory.
Do you confirm y/n: y
```

注意

- 执行命令后，管理 CLI 将重新加载服务器并尝试重新连接该服务器。
  要完成双向 SSL/TLS 身份验证，您需要将服务器证书导入到客户端信任存储中，并将您的客户端配置为显示客户端证书。
- 您可以使用 **disable-ssl-management** 命令为管理接口禁用双向 SSL/TLS。

  ```
  security disable-ssl-management
  ```

  此命令不会删除 Elytron 资源。它将系统配置为使用 **ApplicationRealm** legacy security realm 进行其 SSL 配置。

1.2.6.3. 使用 Elytron 子系统命令启用双向 SSL/TLS

您可以使用 **elytron** 子系统命令为管理接口启用双向 SSL/TLS。

先决条件

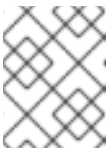
- 您已导出了服务器信任存储的证书。
  如需更多信息，请参阅生成客户端证书。

流程

1. 获取或生成密钥存储。
   在 JBoss EAP 中启用单向 SSL/TLS 之前，您必须获取或生成您计划使用的密钥存储、信任存储和证书。若要生成一组密钥存储、truststores 和证书的示例，可使用以下命令：

   a. 配置 **密钥存储**。

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

   /subsystem=elytron/key-store=twoWayKS:generate-key-
   pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,credential-reference=
   {clear-text=secret},distinguished-name="CN=localhost")

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

   注意

   以上命令使用 **relative-to** 来引用密钥存储文件的位置。或者，您也可以指定 **路径中的** 密钥存储的完整路径，省略 **relative-to**。

   b. 导出您的服务器证书。

   ```
   /subsystem=elytron/key-store=twoWayKS:export-
   certificate(alias=localhost,path=/path/to/server.cer,pem=true)
   ```

c. 为服务器信任 **存储** 创建一个密钥存储，并将客户端证书导入到服务器信任存储中。

注意

以下示例没有信任链的验证证书。如果您使用可信证书，则在没有问题的情况下可以验证客户端证书。

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

/subsystem=elytron/key-store=twoWayTS:import-certificate(alias=client,path=/path/to/client.cer,credential-reference={clear-text=secret},trust-cacerts=true,validate=false)

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

2. 为服务器密钥存储和 truststore 配置 **key-manager**、**trust-manager** 和 **server-ssl-context**。

```
/subsystem=elytron/key-manager=twoWayKM:add(key-store=twoWayKS,credential-reference={clear-text=secret})

/subsystem=elytron/trust-manager=twoWayTM:add(key-store=twoWayTS)

/subsystem=elytron/server-ssl-context=twoWaySSC:add(key-manager=twoWayKM,protocols=["TLSv1.2"],trust-manager=twoWayTM,want-client-auth=true,need-client-auth=true)
```

重要

红帽没有在前面的命令中指定 algorithm 属性，因为 Elytron 子系统使用 **KeyManagerFactory.getDefaultAlgorithm（）** 和 **TrustManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些关键管理器算法。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **SunX509** 作为密钥管理器算法属性，将 **PKIX** 指定为信任管理器算法属性。

您还需要确定您要支持的 HTTPS 协议。上面的示例命令使用 **TLSv1.2**。

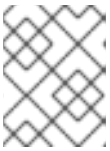
您可以使用 **cipher-suite-filter** 指定密码套件，**使用-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 **security** 子系统行为不同，它默认为遵循客户端密码套件顺序。

3. 在管理界面中启用 HTTPS。

```
/core-service=management/management-interface=http-interface:write-attribute(name=ssl-context, value=twoWaySSC)

/core-service=management/management-interface=http-interface:write-attribute(name=secure-socket-binding, value=management-https)
```

4. 重新加载 JBoss EAP 实例。

```
reload
```



注意

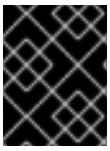
要完成双向 SSL/TLS 身份验证，您需要将服务器证书导入到客户端信任存储中，并将您的客户端配置为显示客户端证书。

5. 将您的客户端配置为使用客户端证书。
   您需要将客户端配置为向服务器提供可信客户端证书，以完成双向 SSL/TLS 身份验证。例如，如果使用浏览器，您需要将可信证书导入到浏览器的信任存储中。

   这会导致强制双向 SSL/TLS 身份验证，而不将原始身份验证改为服务器管理。

   如果要更改原始身份验证方法，请参阅 JBoss EAP 的 *How to Configure Identity Management* 中的 Configure Authentication with Certificates 部分。

现在为管理界面启用了双向 SSL/TLS。



重要

如果您*同时*定义了 **security-realm** 和 **ssl-context**，JBoss EAP 将使用 **ssl-context** 提供的 SSL/TLS 配置。

其他资源

- key-store 属性
- key-manager 属性
- server-ssl-context 属性
- trust-manager 属性

1.2.7. 使用 CLI 安全性命令进行管理接口的 SASL 身份验证

您可以使用 CLI **security** 命令为管理界面启用和禁用 SASL 身份验证。您还可以使用命令对 SASL 机制重新排序。

启用 SASL 身份验证
在 JBoss EAP 中，使用 elytron SASL 身份验证程序（使用 elytron SASL 身份验证工厂）可使用 **security enable-sasl-management** 命令启用管理界面。此命令创建配置身份验证所需的所有非现有资源。默认情况下，此命令将包含的 SASL 工厂与 **http-interface** 相关联。

示例：启用 SASL 身份验证

```
security enable-sasl-management

Server reloaded.
Command success.
Authentication configured for management http-interface
sasl authentication-factory=management-sasl-authentication
security-domain=ManagementDomain
```

注意

执行命令后，管理 CLI 将重新加载服务器并重新连接该服务器。

如果 SASL 工厂已存在，则工厂将更新为使用 **--mechanism** 参数定义的机制。

如需参数列表，请参阅授权安全参数。

禁用 SASL 身份验证
要删除活动 SASL 身份验证工厂，请使用以下命令：

```
security disable-sasl-management
```

另外，要从活跃的 SASL 身份验证工厂中删除特定机制，请使用以下命令：

```
security disable-sasl-management --mechanism=MECHANISM
```

重新排序 SASL 机制
定义 SASL 机制的顺序规定服务器如何对请求进行身份验证，并提供第一个匹配机制发送到客户端。

您可以通过将逗号分隔传递给 **安全 reorder-sasl-management** 命令来改变这个顺序，例如：

```
security reorder-sasl-management --mechanisms-order=MECHANISM1,MECHANISM2,...
```

其他资源

- 安全授权参数

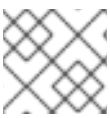
1.2.8. 使用 CLI 安全性命令对管理接口进行 HTTP 身份验证

您可以使用 CLI **security** 命令为管理界面启用和禁用 HTTP 身份验证。

启用 HTTP 验证
在 JBoss EAP 中，可通过 **安全性 enable-http-auth-management** 命令，为管理接口启用 HTTP 身份验证。这个命令只能将 **http-interface** 作为目标，并且没有附加参数，其中包含的 HTTP 身份验证工厂将与此接口相关联。

示例：启用 HTTP 身份验证

```
security enable-http-auth-management

Server reloaded.
Command success.
Authentication configured for management http-interface
http authentication-factory=management-http-authentication
security-domain=ManagementDomain
```



注意

执行命令后，管理 CLI 将重新加载服务器并重新连接该服务器。

如果 HTTP 工厂已存在，则工厂将更新为使用 **--mechanism** 参数定义的机制。

如需参数列表，请参阅授权安全参数。

禁用 HTTP 身份验证
要删除活动的 HTTP 身份验证工厂，请使用以下命令。

```
security disable-http-auth-management
```

或者，您可以使用以下命令从活跃的 HTTP 验证工厂中删除特定机制。

```
security disable-http-auth-management --mechanism=MECHANISM
```

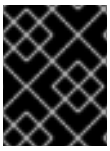
其他资源

- 授权安全参数

1.2.9. 使用旧的内核管理验证为单向 SSL/TLS 配置管理接口

仅使用单向 SSL/TLS 为通信配置 JBoss EAP 管理接口可以提供更高的安全性。客户端和管理界面之间的所有网络流量都是加密的，这降低了安全攻击的风险，比如中间人攻击。

在此过程中，禁用与 JBoss EAP 实例的未加密通信。此流程适用于独立服务器和受管域配置。对于受管域，使用主机名称作为管理 CLI 命令前缀，例如： **/host=master**。



重要

在在管理界面中启用单向 SSL/TLS 的步骤时，除非明确指示，否则不要重新加载配置。这样做可能会导致您被锁定在管理界面中。

1. 创建密钥存储来保护管理接口。
   如需更多信息，请参阅创建密钥存储来保护管理界面。
2. 确保管理接口绑定到 HTTPS。
   如需更多信息，请参阅确保管理接口绑定到 HTTPS。
3. **可选：** 实施自定义 **socket-binding-group**。
   如需更多信息，请参阅自定义 socket-binding-group。
4. 创建新的安全域。
   如需更多信息，请参阅创建新的安全域。
5. 配置管理接口以使用新的安全域。
   如需更多信息，请参阅配置管理界面以使用安全域。
6. 配置管理接口以使用密钥存储。
   如需更多信息，请参阅配置管理接口以使用密钥存储。
7. 更新 **jboss-cli.xml**。
   如需更多信息，请参阅更新 jboss-cli.xml 文件。

1.2.9.1. 创建密钥存储来保护管理接口

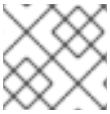
创建密钥存储来保护管理接口。

此密钥存储必须采用 JKS 格式，因为管理接口与 JCEKS 格式的密钥存储不兼容。

流程

- 使用以下 CLI 命令创建密钥存储：
  将参数的示例值（如 **别名**,**keypass**,**keystore**,**storepass** 和 **dname** ）替换为环境的正确值。

```
$ keytool -genkeypair -alias appserver -storetype jks -keyalg RSA -keysize 2048 -keypass password1 -keystore EAP_HOME/standalone/configuration/identity.jks -storepass password1 -dname "CN=appserver,OU=Sales,O=Systems Inc,L=Raleigh,ST=NC,C=US" -validity 730 -v
```



注意

参数 **validity** 指定密钥有效的天数。**730** 等于两年。

1.2.9.2. 确保管理接口绑定到 HTTPS

配置 JBoss EAP 以确保管理界面绑定到 HTTPS。

流程

- 运行独立服务器时的配置
  要确保管理接口绑定到 HTTPS，您必须添加 **management-https** 配置并移除 **management-http** 配置。

  使用以下 CLI 命令，将管理接口绑定到 HTTPS：

```
/core-service=management/management-interface=http-interface:write-attribute(name=secure-socket-binding, value=management-https)

/core-service=management/management-interface=http-interface:undefine-attribute(name=socket-binding)
```

- 运行受管域时的配置
  通过添加 **secure-port** 和删除端口配置来更改 **management-interface** 属性中的 socket 元素。

  使用以下命令将管理接口绑定到 HTTPS：

```
/host=master/core-service=management/management-interface=http-interface:write-attribute(name=secure-port,value=9993)

/host=master/core-service=management/management-interface=http-interface:undefine-attribute(name=port)
```

1.2.9.3. 自定义 socket-binding-group

如果要使用自定义 **socket-binding-group**，您必须确保定义了 **management-https** 绑定，默认绑定到端口 **9993**。您可以从服务器配置文件的 **socket-binding-group** 属性或使用管理 CLI 验证这一点：

```
/socket-binding-group=standard-sockets/socket-binding=management-https:read-resource(recursive=true)

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

```
        "client-mappings" => undefined,
        "fixed-port" => false,
        "interface" => "management",
        "multicast-address" => undefined,
        "multicast-port" => undefined,
        "name" => "management-https",
        "port" => expression "${jboss.management.https.port:9993}"
    }
}
```

1.2.9.4. 创建新的安全域

创建新的安全域。

在此过程中，使用 HTTPS（**ManagementRealmHTTPS**）的新安全 realm 会使用位于 ***EAP_HOME*/standalone/configuration/** 目录中的名为 **https-mgmt-users.properties** 的属性文件来存储用户名和密码。

流程

1. 创建用于存储用户名和密码的属性文件。
   可以稍后向文件中添加用户名和密码，但现在您需要创建一个名为 **https-mgmt-users.properties** 的空文件，并将它保存到该位置。以下示例显示使用 **touch** 命令，但也可以使用其他机制，如文本编辑器。

   示例：使用 touch 命令创建一个空文件

   ```
   $ touch EAP_HOME/standalone/configuration/https-mgmt-users.properties
   ```

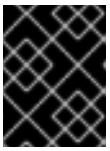
2. 接下来，使用以下管理 CLI 命令创建名为 **ManagementRealmHTTPS** 的新安全域：

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

   /core-service=management/security-realm=ManagementRealmHTTPS/authentication=properties:add(path=https-mgmt-users.properties,relative-to=jboss.server.config.dir)
   ```

3. 将用户添加到属性文件中。
   此时，您已创建一个新的安全域，并将其配置为使用属性文件进行身份验证。现在，您必须使用 **add-user** 脚本将用户添加到属性文件，该文件位于 ***EAP_HOME*/bin/** 目录中。在运行 **add-user** 脚本时，您必须分别使用 **-up** 和 **-r** 选项指定属性文件和安全域。在那里，**add-user** 脚本将以互动方式提示您输入用户名和密码信息，以便存储在 **https-mgmt-users.properties** 文件中。

   ```
   $ EAP_HOME/bin/add-user.sh -up EAP_HOME/standalone/configuration/https-mgmt-users.properties -r ManagementRealmHTTPS
   ...
   Enter the details of the new user to add.
   Using realm 'ManagementRealmHTTPS' as specified on the command line.
   ...
   Username : httpUser
   Password requirements are listed below. To modify these restrictions edit the add-user.properties configuration file.
    - The password must not be one of the following restricted values {root, admin, administrator}
   ```

```
 - The password must contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
 - The password must be different from the username
...
Password :
Re-enter Password :
About to add user 'httpUser' for realm 'ManagementRealmHTTPS'
...
Is this correct yes/no? yes
..
Added user 'httpUser' to file 'EAP_HOME/configuration/https-mgmt-users.properties'
...
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
yes/no? no
```



重要

在配置使用属性文件存储用户名和密码的安全域时，建议每个域都使用一个不与其他域共享的不同属性文件。

1.2.9.5. 配置管理接口以使用安全域

您可以使用管理 CLI 命令将管理接口配置为使用安全域。

流程

- 使用以下管理 CLI 命令：

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

1.2.9.6. 配置管理接口以使用密钥存储

利用管理 CLI 命令，将管理接口配置为使用密钥存储。

流程

1. 使用以下管理 CLI 命令，将管理接口配置为使用密钥存储：
   对于参数文件，密码和别名值必需是来自创建一个密钥存储来保护管理接口安全步骤。

```
/core-service=management/security-realm=ManagementRealmHTTPS/server-identity=ssl:add(keystore-path=identity.jks,keystore-relative-to=jboss.server.config.dir,keystore-password=password1, alias=appserver)
```

注意

要更新密钥存储密码，请使用以下 CLI 命令：

```
/core-service=management/security-realm=ManagementRealmHTTPS/server-identity=ssl:write-attribute(name=keystore-password,value=newpassword)
```

2. 重新载入服务器的配置：

```
reload
```

重新载入服务器配置后，日志应包含以下内容，只有在文本前显示启动的服务数量：

```
13:50:54,160 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0061: Http
management interface listening on https://127.0.0.1:9993/management
13:50:54,162 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0052: Admin console
listening on https://127.0.0.1:9993
```

管理接口现在侦听端口 **9993**，这确认过程是否成功。

重要

此时，CLI 将断开连接，并且因为端口绑定已更改后无法重新连接。

继续下一步，以更新 **jboss-cli.xml** 文件，以允许管理 CLI 重新连接。

1.2.9.7. 更新 jboss-cli.xml 文件

如果使用管理 CLI 执行管理操作，您必须更新 ***EAP_HOME*/bin/jboss-cli.xml** 文件。

流程

- 更新 ***EAP_HOME*/bin/jboss-cli.xml** 文件，如下所示：
  - 将 **<default-protocol>** 的值更新为 **https-remoting**。
  - 在 **<default-controller>** 中，将 **<protocol>** 的值更新为 **https-remoting**。
  - 在 **<default-controller>** 中，将 **<port>** 的值更新为 **9993**。

示例： jboss-cli.xml

```
<jboss-cli xmlns="urn:jboss:cli:2.0">
    <default-protocol use-legacy-override="true">https-remoting</default-protocol>
    <!-- The default controller to connect to when 'connect' command is executed w/o
arguments -->
    <default-controller>
        <protocol>https-remoting</protocol>
        <host>localhost</host>
        <port>9993</port>
    </default-controller>
...
```

下次使用管理 CLI 连接到管理界面时，您必须接受服务器证书并根据 **ManagementRealmHTTPS** 安全域进行身份验证：

示例：接受服务器证书和身份验证

```
$ ./jboss-cli.sh -c
Unable to connect due to unrecognised server certificate
```

```
Subject    - CN=appserver,OU=Sales,O=Systems Inc,L=Raleigh,ST=NC,C=US
Issuer     - CN=appserver, OU=Sales, O=Systems Inc, L=Raleigh, ST=NC, C=US
Valid From - Tue Jun 28 13:38:48 CDT 2016
Valid To   - Thu Jun 28 13:38:48 CDT 2018
MD5 : 76:f4:81:8b:7e:c3:be:6d:ee:63:c1:7a:b7:b8:f0:fb
SHA1 : ea:e3:f1:eb:53:90:69:d0:c9:69:4a:5a:a3:20:8f:76:c1:e6:66:b6

Accept certificate? [N]o, [T]emporarily, [P]ermenantly : p
Authenticating against security realm: ManagementRealmHTTPS
Username: httpUser
Password:
[standalone@localhost:9993 /]
```

重要

如果您*同时*定义了 **security-realm** 和 **ssl-context**，JBoss EAP 将使用 **ssl-context** 提供的 SSL/TLS 配置。

1.2.10. 为使用旧核心管理身份验证的管理接口设置双向 SSL/TLS

双向 SSL/TLS 身份验证（也称为 *客户端身份验证*）使用 SSL/TLS 证书验证客户端和服务器。这与 为单向 SSL/TLS 部分配置管理接口 不同，其中的客户端和服务器都有证书。这提供了保证，不仅显示它的服务器，而且客户端也表示它是什么。

本节使用以下惯例：

HOST1
: JBoss 服务器主机名。例如： **jboss.redhat.com**。

HOST2
: 适合客户端的名称。例如： **myclient**。请注意，这不一定是实际的主机名。

CA_HOST1
: 用于 HOST1 证书的 DN（区分名称）。例如： **cn=jboss,dc=redhat,dc=com**。

CA_HOST2
: 用于 HOST2 证书的 DN（区分名称）。例如： **cn=myclient,dc=redhat,dc=com**。

注意

如果使用了密码 vault 来存储密钥存储和信任存储密码，则应当已经创建密码 vault。有关密码库的更多信息，请参阅 Red Hat JBoss Enterprise Application Platform 7 *安全架构指南中的* Password Vault 部分以及 Password Vault System 部分。

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。

流程

1. 生成密钥存储。

```
$ keytool -genkeypair -alias HOST1_alias -keyalg RSA -keysize 1024 -validity 365 -keystore HOST1.keystore.jks -dname "CA_HOST1" -keypass secret -storepass secret

$ keytool -genkeypair -alias HOST2_alias -keyalg RSA -keysize 1024 -validity 365 -keystore HOST2.keystore.jks -dname "CA_HOST2" -keypass secret -storepass secret
```

2. 导出证书。

```
$ keytool -exportcert  -keystore HOST1.keystore.jks -alias HOST1_alias -keypass secret -storepass secret -file HOST1.cer

$ keytool -exportcert  -keystore HOST2.keystore.jks -alias HOST2_alias -keypass secret -storepass secret -file HOST2.cer
```

3. 将证书导入到而非信任存储中。

```
$ keytool -importcert -keystore HOST1.truststore.jks -storepass secret -alias HOST2_alias -trustcacerts -file HOST2.cer

$ keytool -importcert -keystore HOST2.truststore.jks -storepass secret -alias HOST1_alias -trustcacerts -file HOST1.cer
```

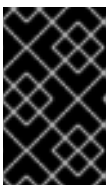
4. 定义证书。
在服务器（**host.xml** 或 **standalone.xml**）的配置中定义 CertificateRealm，并将接口指向该服务器。这可以通过以下命令完成：

```
/core-service=management/security-realm=CertificateRealm:add()

/core-service=management/security-realm=CertificateRealm/server-identity=ssl:add(keystore-path=/path/to/HOST1.keystore.jks, keystore-password=secret,alias=HOST1_alias)

/core-service=management/security-realm=CertificateRealm/authentication=truststore:add(keystore-path=/path/to/HOST1.truststore.jks,keystore-password=secret)
```

5. 将 **http-interface** 的 **security-realm** 更改为新的 CertificateRealm。

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

6. 为 CLI 添加 SSL/TLS 配置。



重要

除了添加双向 SSL/TLS 外，还必须将管理接口配置为绑定到 HTTPS。详情请参阅在标题为使用旧核心管理身份验证的 单向 SSL/TLS 中配置管理接口部分的管理接口（如网络接口绑定）部分的 HTTPS。

为 CLI 添加 SSL/TLS 配置，该配置使用 ***EAP_HOME*/bin/jboss-cli.xml** 作为设置文件。

要将密钥存储和信任存储密码以纯文本形式存储，请编辑 ***EAP_HOME*/bin/jboss-cli.xml** 并使用变量的适当值添加 SSL/TLS 配置：

示例：jboss-cli.xml 存储密钥存储和 Truststore Passwords（明文形式）

```
<ssl>
  <alias>HOST2_alias</alias>
  <key-store>/path/to/HOST2.keystore.jks</key-store>
  <key-store-password>secret</key-store-password>
  <trust-store>/path/to/HOST2.truststore.jks</trust-store>
  <trust-store-password>secret</trust-store-password>
  <modify-trust-store>true</modify-trust-store>
</ssl>
```

要使用存储在密码库中的密钥存储和信任存储密码，您需要将 vault 配置和适当的 vault 值添加到 ***EAP_HOME*/bin/jboss-cli.xml** 中：

示例：jboss-cli.xmltoring Keystore 和 Truststore Passwords（明文形式）

```
<ssl>
  <vault>
    <vault-option name="KEYSTORE_URL" value="path-to/vault/vault.keystore"/>
    <vault-option name="KEYSTORE_PASSWORD" value="MASK-5WNXs8oEbrs"/>
    <vault-option name="KEYSTORE_ALIAS" value="vault"/>
    <vault-option name="SALT" value="12345678"/>
    <vault-option name="ITERATION_COUNT" value="50"/>
    <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
  </vault>
  <alias>HOST2_alias</alias>
  <key-store>/path/to/HOST2.keystore.jks</key-store>
  <key-store-password>VAULT::VB::cli_pass::1</key-store-password>
  <key-password>VAULT::VB::cli_pass::1</key-password>
  <trust-store>/path/to/HOST2.truststore.jks</trust-store>
  <trust-store-password>VAULT::VB::cli_pass::1</trust-store-password>
  <modify-trust-store>true</modify-trust-store>
</ssl>
```

重要

如果您*同时*定义了 **security-realm** 和 **ssl-context**，JBoss EAP 将使用 **ssl-context** 提供的 SSL/TLS 配置。

1.2.11. HTTPS Listener 参考

如需可用于 HTTPS 侦听器的属性的完整列表，请参阅 JBoss EAP *配置指南*中的 Undertow 子系统属性小节。

1.2.11.1. 关于 Cipher Suites

您可以配置允许的加密密码列表。对于 JSSE 语法，它必须用逗号分开的列表。对于 OpenSSL 语法，它必须是以冒号分隔的列表。确定只使用一个语法。默认为 JVM 默认。

重要

使用弱密码是一个重大安全风险。有关加密套件的 NIST 建议，请参阅 NIST 指南。

有关可用 OpenSSL 密码列表，请查看 OpenSSL 文档。请注意，不支持以下内容：

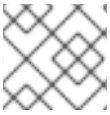
- *@SECLEVEL*
- *SUITEB128*
- *SUITEB128ONLY*
- *SUITEB192*

有关 标准 JSSE 密码列表， 请参见 Java 文档。

要更新启用的密码套件的列表，请在 **undertow** 子系统中使用 HTTPS 侦听器的 *enabled-cipher-suites* 属性。

示例：更新 Enabled Cipher Suites 列表的管理 CLI 命令

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=enabled-cipher-
suites,value="TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA")
```



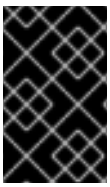
注意

这个示例只列出两个可能的密码，但实际示例可能会使用更多。

1.2.12. 使用 OpenSSL 供应商启用对 TLS 1.3 协议的支持

您可以通过在 **ssl-context** 配置中配置 **cipher-suite-names** 属性，通过为 TLS 启用对 TLS 1.3 协议的支持。选择以下方法之一将 JBoss EAP 配置为使用 OpenSSL TLS 供应商：

- 将 Elytron 子系统配置为使用 OpenSSL TLS 供应商。
- 配置 **server-ssl-context** 组件的 **provider** 属性，或将 **client-ssl-context** 组件配置为使用 OpenSSL TLS 供应商。



重要

与 TLS 1.2 相比，在使用 JDK 11 运行 TLS 1.3 时您可能会体验到较低的性能。当客户端向服务器发出大量 TLS 1.3 请求时会出现这种情况。系统升级到较新的 JDK 版本可以提高性能。在生产环境中启用前，使用 TLS 1.3 测试您的设置是否出现性能下降的问题。

先决条件

- 为应用程序启用单向 SSL/TLS 或双向 SSL/TLS。

流程

1. 选择以下一种方法之一将 JBoss EAP 7.4 实例配置为使用 OpenSSL TLS 供应商：

a. 将 **elytron** 子系统配置为使用 OpenSSL TLS 供应商。要做到这一点，删除默认的 **final-providers** 配置，它会在所有全局注册的供应商后注册 OpenSSL TLS 供应商。

```
/subsystem=elytron:undefine-attribute(name=final-providers)
reload
```

接下来，在所有全局注册的供应商前注册 OpenSSL TLS 供应商。

```
/subsystem=elytron:write-attribute(name=initial-providers, value=combined-providers)
```

b. 配置 **server-ssl-context** 或 **client-ssl-context** 的 **providers** 属性以使用 OpenSSL TLS 供应商。

为名为 serverSSC 的现有 server-ssl-context 设置 providers 属性的示例。

```
/subsystem=elytron/server-ssl-context=serverSSC:write-
attribute(name=providers,value=openssl)
reload
```

2. *可选：* 如果您将 **ssl-context** 配置为使用 TLS 1.3 协议以外的协议，您必须在 **ssl-context** 中配置 **protocol** 属性使其包含 TLS 1.3 协议：

```
/subsystem=elytron/server-ssl-context=serverSSC:write-attribute(name=protocols,value=
[TLSv1.3])
```

3. 通过在 **ssl-context** 配置中配置 password **-suite-names** 属性，启用对 OpenSSL 供应商的 TLS 1.3 协议的支持。以下示例将 **TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256** 设置为 password **-suite-names** 属性的值：

```
/subsystem=elytron/server-ssl-context=serverSSC:write-attribute(name=cipher-suite-
names,value=TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AE
S_128_GCM_SHA256)
```

4. 重新载入 JBoss EAP 实例：

```
reload
```

5. *可选：* 测试您可以使用 TLS 1.3 协议和 TLS 1.3 密码套件成功与服务器建立 SSL 加密连接。使用 **curl** 等工具检查配置的输出：

```
curl -v https://<ip_address>:<ssl_port>
```

显示 TLS_AES_256_GCM_SHA384 带有 TLS 1.3 协议来保护 SSL 连接的输出示例。

```
SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
*  subject: C=Unknown; ST=Unknown; L=Unknown; O=Unknown; OU=Unknown;
CN=localhost
*  start date: Oct  6 14:58:16 2020 GMT
*  expire date: Nov  5 15:58:16 2020 GMT
```

```
* issuer: C=Unknown; ST=Unknown; L=Unknown; O=Unknown; OU=Unknown;
CN=localhost
* SSL certificate verify result: self signed certificate (18), continuing anyway.
```

其他资源

- 有关为应用程序启用单向 SSL/TLS 或双向 SSL/TLS 的信息，请参阅 使用 Elytron subsystem 为应用程序启用单向 SSL/TLS。
- 有关 **client-ssl-context** 的详情请参考 使用 **client-ssl-context**。
- 有关 **server-ssl-context** 的详情请参考 使用 **server-ssl-context**。

1.2.13. FIPS 140-2 Compliant Cryptography

可使用以下任一方法在 Red Hat Enterprise Linux 上配置 FIPS 140-2 兼容加密。

- 使用带有 NSS 数据库的 SunPKCS11 供应商
- 使用第三方 BouncyCastle 供应商

1.2.13.1. 在 Red Hat Enterprise Linux 7 及之后的版本中为 SSL/TLS 启用 FIPS 140-2 Cryptography

您可以将 Undertow 配置为将 FIPS 140-2 兼容加密用于 SSL/TLS。此配置示例的范围仅限于 Red Hat Enterprise Linux 7 及之后的版本，在 FIPS 模式中使用 Mozilla NSS 库。

重要

安装的 Red Hat Enterprise Linux 必须已经配置为符合 FIPS 140-2。如需更多信息，请参阅标题为 How can make RHEL 6 或 RHEL 7 FIPS 140-2 的解决方案（在红帽客户门户网站中）。

警告

在以 FIPS 模式运行时使用 TLS 1.2 协议可能会导致 **NoSuchAlgorithmException**。有关此问题的更多详细信息，请参阅标题为 NoSuchAlgorithmException: no such algorithm: SunTls12MasterSecret，它位于红帽客户门户网站。

因此，无法在 FIPS 模式中配置 HTTP/2，因为 HTTP/2 需要 TLS 1.2 协议。FIPS 模式(PKCS11)支持 TLS 1 和 TLS 1.1 协议，以便您可以使用：

- Oracle/OpenJDK：TLS 1.1
- IBM java：TLS 1

要将 Undertow 配置为对 SSL/TLS 使用 FIPS 140-2 兼容加密，您必须执行以下操作：

- 配置 NSS 数据库。

- 为 SSL/TLS 配置 FIPS 140-2 兼容加密的管理 CLI 。
- 将 **undertow** 子系统配置为使用 Elytron 或 旧核心管理身份验证。

注意

OpenSSL 供应商需要一个私钥，但无法从 PKCS11 存储中检索私钥。FIPS 不允许从 FIPS 兼容的加密模块导出未加密的密钥。因此，对于 **elytron** 子系统和旧的安全性，在 FIPS 模式中无法将 OpenSSL 供应商用于 TLS。

配置 NSS 数据库

1. 创建由适当用户拥有的目录，以存放 NSS 数据库。

   创建 NSS 数据库目录的命令示例

   ```
   $ mkdir -p /usr/share/jboss-as/nssdb
   $ chown jboss /usr/share/jboss-as/nssdb
   $ modutil -create -dbdir /usr/share/jboss-as/nssdb
   ```

   注意

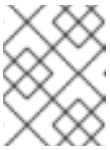
   - DBM 文件格式，RHEL 7 及更早版本的默认数据库格式已被弃用。NSS 现在默认使用 SQL。
   - *jboss* 用户只是一个示例。使用操作系统上的活跃用户替换它来运行 JBoss EAP。

2. 创建 NSS 配置文件： **/usr/share/jboss-as/nss_pkcsll_fips.cfg**。
   它必须指定：

   - 一个名称
   - NSS 库所在的目录
   - 上一步中创建的 NSS 数据库的目录

   示例： nss_pkcsll_fips.cfg

   ```
   name = nss-fips
   nssLibraryDirectory=/usr/lib64
   nssSecmodDirectory=/usr/share/jboss-as/nssdb
   nssDbMode = readOnly
   nssModule = fips
   ```

   注意

   如果您没有运行 64 位版本的 Red Hat Enterprise Linux 6，那么 **nss1028Directory** 设为 **/usr/lib**，而不是 **/usr/lib64**。

3. 编辑 Java 安全配置文件。此配置文件会影响整个 JVM，并可使用以下任一方式进行定义。

- JDK 中提供了默认配置文件 **java.security**。如果没有指定其他安全配置文件，则使用此文件。有关此文件的位置，请查看 JDK 供应商的文档。

- 定义自定义 Java 安全配置文件，并使用 **-Djava.security.properties=*/path/to/*java.security.properties** 来引用。以这种方式引用时，它会覆盖默认安全文件中的设置。在需要不同安全设置的同一主机上运行的多个 JVM 时，此选项很有用。
  在您的 Java 安全配置文件中添加以下行：

  示例： java.security

  ```
  security.provider.1=sun.security.pkcs11.SunPKCS11 /usr/share/jboss-as/nss_pkcsll_fips.cfg
  ```

  注意

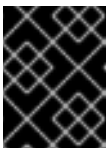
  上述行中指定的 **nss_pkcsll_fips.cfg** 配置文件是上一步中创建的文件。

  您还需要从以下位置更新配置文件中的以下链接：

  ```
  security.provider.5=com.sun.net.ssl.internal.ssl.Provider
  ```

  至

  ```
  security.provider.5=com.sun.net.ssl.internal.ssl.Provider SunPKCS11-nss-fips
  ```

  重要

  此文件中的任何其他 **security.provider.X** 行（如 **security.provider.2**）都必须增加其 X 的值，以确保此 provider 具有优先权。

4. 在您在上一步中创建的 NSS 数据库目录上运行 **modutil** 命令以启用 FIPS 模式。

   ```
   modutil -fips true -dbdir /usr/share/jboss-as/nssdb
   ```

   注意

   此时您可能会收到一个安全库错误，要求您为某些 NSS 共享对象重新生成库签名。

5. 在 FIPS 令牌中设置密码。
   令牌的名称*必须*是 *NSS FIPS 140-2 Certificate DB*。

   ```
   modutil -changepw "NSS FIPS 140-2 Certificate DB" -dbdir /usr/share/jboss-as/nssdb
   ```

   重要

   用于 FIPS 令牌的密码必须是 FIPS 兼容密码。如果密码不够强，您可能会收到错误： *ERROR: Unable to change password on token "NSS FIPS 140-2 Certificate DB"*。

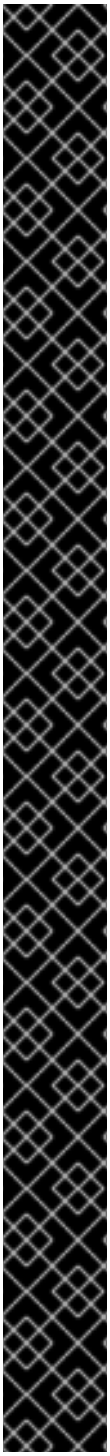
6. 使用 NSS 工具创建证书。

示例命令

```
$ certutil -S -k rsa -n undertow -t "u,u,u" -x -s "CN=localhost, OU=MYOU, O=MYORG,
L=MYCITY, ST=MYSTATE, C=MY" -d /usr/share/jboss-as/nssdb
```

7. 运行以下命令，验证 JVM 是否可以从 PKCS11 密钥存储读取私钥：

```
$ keytool -list -storetype pkcs11
```



重要

启用 FIPS 后，您可能会在启动 JBoss EAP 时看到以下错误：

```
10:16:13,993 ERROR [org.jboss.msc.service.fail] (MSC service thread 1-1)
MSC000001: Failed to start service
jboss.server.controller.management.security_realm.ApplicationRealm.key-manager:
org.jboss.msc.service.StartException in service
jboss.server.controller.management.security_realm.ApplicationRealm.key-manager:
WFLYDM0018: Unable to start service
 at
org.jboss.as.domain.management.security.AbstractKeyManagerService.start(AbstractKe
yManagerService.java:85)
 at
org.jboss.msc.service.ServiceControllerImpl$StartTask.startService(ServiceControllerImp
l.java:1963)
 at
org.jboss.msc.service.ServiceControllerImpl$StartTask.run(ServiceControllerImpl.java:18
96)
 at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
 at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
 at java.lang.Thread.run(Thread.java:745)
Caused by: java.security.KeyStoreException: FIPS mode: KeyStore must be from
provider SunPKCS11-nss-fips
 at
sun.security.ssl.KeyManagerFactoryImpl$SunX509.engineInit(KeyManagerFactoryImpl.j
ava:67)
 at javax.net.ssl.KeyManagerFactory.init(KeyManagerFactory.java:256)
 at
org.jboss.as.domain.management.security.AbstractKeyManagerService.createKeyMana
gers(AbstractKeyManagerService.java:130)
 at
org.jboss.as.domain.management.security.AbstractKeyManagerService.start(AbstractKe
yManagerService.java:83)
 ... 5 more
```

如果您已配置了任何现有的密钥管理器，如旧核心管理身份验证中的默认密钥管理器（不使用 FIPS 140-2 加密），则将显示此消息。

为 **FIPS 140-2 Compliant Cryptography 配置管理 CLI**
您必须将 JBoss EAP 管理 CLI 配置为在启用了 SSL/TLS 的 FIPS 140-2 兼容加密的环境中工作。默认情

况下，如果您尝试在此类环境中使用管理 CLI，则会抛出以下异常：
org.jboss.as.cli.CliProcessException: java.security.KeyManagementException: FIPS 模式： 只能使用 SunJSSE TrustManagers。

- 如果您使用传统的 **security** 子系统：
  更新 **jboss-cli.sh** 文件中的 **javax.net.ssl.keyStore** 和 **javax.net.ssl.trustStore** 系统属性，如下所示：

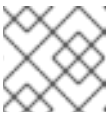
```
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStore=NONE -
Djavax.net.ssl.trustStoreType=PKCS11"
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.keyStore=NONE -
Djavax.net.ssl.keyStoreType=PKCS11 -Djavax.net.ssl.keyStorePassword=P@ssword123"
```

- 如果您使用 **elytron** 子系统：

  1. 使用以下内容为管理 CLI 创建 XML 配置文件：

     示例： cli-wildfly-config.xml

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <key-stores>
      <key-store name="truststore" type="PKCS11">
        <key-store-clear-password password="P@ssword123"/>
      </key-store>
    </key-stores>
    <ssl-contexts>
      <ssl-context name="client-cli-context">
        <trust-store key-store-name="truststore"/>
        <cipher-suite selector="${cipher.suite.filter}"/>
        <protocol names="TLSv1.1"/>
      </ssl-context>
    </ssl-contexts>
    <ssl-context-rules>
      <rule use-ssl-context="client-cli-context"/>
    </ssl-context-rules>
  </authentication-client>
</configuration>
```



注意

如果使用 IBM JDK，请参阅 IBM 管理 CLI 配置示例 以了解所需的具体配置。

  2. 启动管理 CLI 时，使用 **-Dwildfly.config.url** 属性将配置文件传递给管理 CLI 脚本。例如：

```
$ jboss-cli.sh -Dwildfly.config.url=cli-wildfly-config.xml
```

配置 Elytron 和 Undertow 子系统

1. 添加 FIPS 140-2 兼容加密密钥 **存储**、**key-manager** 和 **ssl-context**。

```
/subsystem=elytron/key-store=fipsKS:add(type=PKCS11,provider-name="SunPKCS11-nss-
fips",credential-reference={clear-text="P@ssword123"})
```

```
/subsystem=elytron/key-manager=fipsKM:add(key-
store=fipsKS,algorithm="SunX509",provider-name=SunPKCS11-nss-fips,credential-
reference={clear-text="P@ssword123"})

/subsystem=elytron/server-ssl-context=fipsSSC:add(key-manager=fipsKM,protocols=
["TLSv1.1"])
```

2. 更新 **undertow** 子系统，以使用新的 **ssl-context**。

注意

https-listener 需要始终配置了 **security-realm** 或 **ssl-context**。在两个配置之间更改时，命令必须作为一个批处理来执行，如下所示。

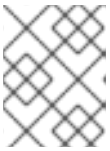
```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-
attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context,value=fipsSSC)
run-batch

reload
```

在 **elytron** 子系统中，OpenJDK 和 Oracle JDK（在 FIPS 模式中）会限制任何基于提供自定义 **KeyManager** 或 **TrustManager** 实现的高级功能的使用。以下配置属性无法在服务器上工作：

- **server-ssl-context.security-domain**
- **trust-manager.certificate-revocation-list**

使用传统核心管理身份验证配置 Undertow
另外，您仍然可以使用传统的核心管理身份验证而不是 **elytron** 子系统完成对 SSL/TLS 的 FIPS 140-2 兼容加密设置：

1. 配置 Undertow 以使用 SSL/TLS。



注意

以下命令必须在批处理模式下运行，或者在添加 *ssl* 服务器身份后重新加载服务器。下例中使用了批处理模式。

```
batch

/core-service=management/security-realm=HTTPSRealm:add

/core-service=management/security-realm=HTTPSRealm/server-identity=ssl:add(keystore-
provider=PKCS11, keystore-password="strongP@ssword1")

/subsystem=undertow/server=default-server/https-listener=https:add(socket-binding=https,
security-realm=HTTPSRealm, enabled-protocols="TLSv1.1")

run-batch
```

为应用设置 SSL/TLS部分介绍了将 Undertow 配置为 SSL/TLS 的基本信息。

2. 配置 Undertow 使用的加密套件。
配置了 SSL/TLS 后，您需要将 https 侦听程序和安全域配置为启用了特定的密码套件：

所需的加密套件

```
SSL_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_anon_WITH_AES_128_CBC_SHA,
TLS_ECDH_anon_WITH_AES_256_CBC_SHA
```

关于为 https 侦听程序启用加密套件的基础知识，请参阅关于加密套件。在 https 监听器上启用密码套件：

在 Https Listener 上启用 Cipher Suites 的命令示例

```
/subsystem=undertow/server=default-server/https-listener=https:write-
attribute(name=enabled-cipher-
suites,value="SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL_DHE_RSA_WITH_3DES_EDE_
CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_S
HA,TLS_DHE_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA,TLS
_DHE_DSS_WITH_AES_256_CBC_SHA,TLS_DHE_RSA_WITH_AES_256_CBC_SHA,TLS_
ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_ECDSA_WITH_AES_128_CBC_
SHA,TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,TLS_ECDHE_ECDSA_WITH_3DES_
EDE_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_ECDSA_
WITH_AES_256_CBC_SHA,TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDH_R
SA_WITH_AES_128_CBC_SHA,TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,TLS_ECDHE
_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_
ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
,TLS_ECDH_anon_WITH_AES_128_CBC_SHA,TLS_ECDH_anon_WITH_AES_256_CBC_S
HA")
```

3. 在安全域中启用密码套件。

在 Security Realm 上启用 Cipher Suites 的命令示例

```
/core-service=management/security-realm=HTTPSRealm/server-identity=ssl:write-
attribute(name=enabled-cipher-suites, value=[SSL_RSA_WITH_3DES_EDE_CBC_SHA,
```

```
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDH_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA,
TLS_ECDH_anon_WITH_AES_128_CBC_SHA,
TLS_ECDH_anon_WITH_AES_256_CBC_SHA])
```

1.2.13.2. 使用 Bouncy Castle 启用 FIPS 140-2 Cryptography for SSL/TLS

您可以将 Undertow 配置为将 FIPS 140-2 兼容加密用于 SSL/TLS。此配置示例的范围仅限于 Red Hat Enterprise Linux 7 及更新的版本。红帽不提供 Bouncy Castle JARs，它必须直接从 Bouncy Castle 获取。

先决条件

- 确保您的环境 已配置为使用 **BouncyCastle** 提供商。
- 服务器上必须存在 Bouncy Castle 密钥存储。如果不存在，您可以使用以下命令创建。

```
$ keytool -genkeypair -alias ALIAS -keyalg RSA -keysize 2048 -keypass PASSWORD -
keystore KEYSTORE -storetype BCFKS -storepass STORE_PASSWORD
```

使用 Elytron 为 FIPS 140-2 Cryptography 配置管理 CLI
您必须将 JBoss EAP 管理 CLI 配置为在启用了 SSL/TLS 的 FIPS 140-2 兼容加密的环境中工作。

1. 使用以下内容为管理 CLI 创建 XML 配置文件：

   示例： cli-wildfly-config.xml

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <key-stores>
      <key-store name="truststore" type="BCFKS">
        <file name="${truststore.location}" />
        <key-store-clear-password password="${password}" />
      </key-store>
      <key-store name="keystore" type="BCFKS">
        <file name="${keystore.location}" />
        <key-store-clear-password password="${password}" />
      </key-store>
    </key-stores>
```

```
    <ssl-contexts>
      <ssl-context name="client-cli-context">
        <key-store-ssl-certificate algorithm="PKIX" key-store-name="keystore">
          <key-store-clear-password password="${password}" />
        </key-store-ssl-certificate>
        <trust-store key-store-name="truststore"/>
        <trust-manager algorithm="PKIX">
        </trust-manager>
        <cipher-suite
selector="TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA,TLS_DHE_DSS_WITH_AES_128_
CBC_SHA,TLS_DHE_DSS_WITH_AES_128_CBC_SHA256,TLS_DHE_DSS_WITH_AES_25
6_CBC_SHA,TLS_DHE_DSS_WITH_AES_256_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_
3DES_EDE_CBC_SHA,TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_EC
DSA_WITH_AES_128_CBC_SHA256,TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,TL
S_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,TLS_ECDHE_RSA_WITH_3DES_EDE_
CBC_SHA,TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_
128_CBC_SHA256,TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_3DES
_EDE_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_SHA,TLS_RSA_WITH_AES_128_CBC_
SHA256,TLS_RSA_WITH_AES_256_CBC_SHA,TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_RSA_WITH_AES_256_CCM,TLS_RSA_WITH_AES_128_CCM"/>
        <protocol names="TLSv1.2"/>
      </ssl-context>
    </ssl-contexts>
    <ssl-context-rules>
      <rule use-ssl-context="client-cli-context"/>
    </ssl-context-rules>
  </authentication-client>
</configuration>
```

2. 启动管理 CLI 时，使用 **-Dwildfly.config.url** 属性将配置文件传递给管理 CLI 脚本。例如：

```
$ jboss-cli.sh -Dwildfly.config.url=cli-wildfly-config.xml
```

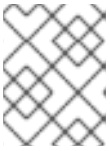
配置 Elytron 和 Undertow 子系统

1. 添加 FIPS 140-2 兼容加密密钥 **存储**、**key-manager** 和 **ssl-context**。在定义密钥存储时，类型必须是 **BCFKS**。

```
/subsystem=elytron/key-store=fipsKS:add(path=KEYSTORE,relative-
to=jboss.server.config.dir,credential-reference={clear-
text=STORE_PASSWORD},type="BCFKS")

/subsystem=elytron/key-manager=fipsKM:add(key-store=fipsKS,algorithm="PKIX",credential-
reference={clear-text=PASSWORD})

/subsystem=elytron/server-ssl-context=fipsSSC:add(key-manager=fipsKM,protocols=
["TLSv1.2"],cipher-suite-filter="TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA,
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256,
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA,
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384,
```

```
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA, TLS_RSA_WITH_3DES_EDE_CBC_SHA,
TLS_RSA_WITH_AES_128_CBC_SHA, TLS_RSA_WITH_AES_128_CBC_SHA256,
TLS_RSA_WITH_AES_256_CBC_SHA,
TLS_RSA_WITH_AES_256_CBC_SHA256,TLS_RSA_WITH_AES_256_CCM,TLS_RSA_WITH_AES_128_CCM")
```

2. 更新 **undertow** 子系统，以使用新的 **ssl-context**。



注意

https-listener 需要始终配置了 **security-realm** 或 **ssl-context**。在两个配置之间更改时，命令必须作为一个批处理来执行，如下所示。

```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-context,value=fipsSSC)
run-batch

reload
```

1.2.14. IBM JDK 上的 FIPS 140-2 Compliant Cryptography

在 IBM JDK 上，IBM Java Cryptographic Extension(JCE)IBMJCEFIPS 供应商和 IBM Java Secure Sockets Extension(JSSE) FIPS 140-2 Cryptographic 模块(IBMJSSE2)提供 FIPS 140-2 兼容加密。

有关 IBMJCEFIPS 供应商的更多信息，请参阅 IBM Documentation for IBM JCEFIPS 和 NIST IBMJCEFIPS – Security Policy。有关 IBMJSSE2 的更多信息，请参阅 在 FIPS 模式中运行 IBMJSSE2 。

1.2.14.1. 密钥存储

IBM JCE 不提供密钥存储。密钥存储在计算机上，不会离开其物理边界。如果密钥在必须加密的计算机间移动。

要在 FIPS 兼容模式下运行 **keytool**，在每个命令中使用 **-providerClass** 选项，如下所示：

```
keytool -list -storetype JCEKS -keystore mystore.jck -storepass mystorepass -providerClass com.ibm.crypto.fips.provider.IBMJCEFIPS
```

1.2.14.2. 管理 CLI 配置

要在 IBM JDK 上为 FIPS 140-2 兼容加密配置管理 CLI，您必须使用专门用于 IBM JDK 的管理 CLI 配置文件，如下所示：

示例： cli-wildfly-config-ibm.xml

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    <key-stores>
```

```
      <key-store name="truststore" type="JKS">
        <file name="/path/to/truststore"/>
        <key-store-clear-password password="P@ssword123"/>
      </key-store>
    </key-stores>
    <ssl-contexts>
      <ssl-context name="client-cli-context">
        <trust-store key-store-name="truststore"/>
        <cipher-suite selector="${cipher.suite.filter}"/>
        <protocol names="TLSv1"/>
      </ssl-context>
    </ssl-contexts>
    <ssl-context-rules>
      <rule use-ssl-context="client-cli-context"/>
    </ssl-context-rules>
  </authentication-client>
</configuration>
```

1.2.14.3. 检查 FIPS 提供者信息

要检查服务器使用的 IBMJCEFIPS 的信息，请通过将 **-Djavax.net.debug=true** 添加到 **standalone.conf** 或 **domain.conf** 文件来启用调试级别日志。有关 FIPS 供应商的信息记录到 **server.log** 文件，例如：

```
04:22:45,685 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using MessageDigest SHA from
provider IBMJCEFIPS version 1.7
04:22:45,689 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH KeyPairGenerator  from provider
from init IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using KeyFactory DiffieHellman from
provider IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) JsseJCE:  Using KeyAgreement DiffieHellman
from provider IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH KeyAgreement  from provider
IBMJCEFIPS version 1.7
04:22:45,754 INFO  [stdout] (http-/127.0.0.1:8443-1) DHCrypt:  DH KeyAgreement  from provider
from initIBMJCEFIPS version 1.7
```

1.2.15. 当 JVM 在 FIPS 模式中运行时启动受管域（Managed Domain）

更新每个主机控制器和主域控制器，以使用 SSL/TLS 进行通信。

先决条件
在开始之前，请确定您已完成以下先决条件。

- 您已实施了一个受管域。
  有关配置受管域的详情，请参考 JBoss EAP *配置指南中*的 域管理部分。
- 您已配置了 FIPS。
  有关配置 FIPS 的详情，请参考在 Red Hat Enterprise Linux 7 及之后的版本中为 SSL/TLS 启用 FIPS 140-2 Cryptography。
- 您已创建了所有必要的证书，并将域控制器的证书导入到每个控制器的信任存储中。

警告

红帽建议明确禁用 SSLv2、SSLv3 和 TLSv1.0，而不是在所有受影响的软件包中替代 TLSv1.1。

1. 在主域控制器上，创建一个 SSL/TLS 安全域，该域配置为使用 NSS 数据库作为 PKCS11 供应商。

   示例：主域控制器上的安全域控制器

```
<security-realm name="HTTPSRealm">
    <server-identities>
        <ssl>
            <engine enabled-protocols="TLSv1.1"/>
            <keystore provider="PKCS11" keystore-password="strongP@ssword1"/>
        </ssl>
    </server-identities>
    <authentication>
        <local default-user="\$local"/>
        <properties path="https-users.properties" relative-to="jboss.domain.config.dir"/>
    </authentication>
</security-realm>
```

2. 在每个主机控制器中，创建一个使用 SSL/TLS 信任存储进行身份验证的安全域。

   示例：每个主机控制器上的安全性 Realm

```
<security-realm name="HTTPSRealm">
  <authentication>
    <truststore provider="PKCS11" keystore-password="strongP@ssword1"/>
  </authentication>
</security-realm>
```

注意

在每个主机上重复此过程。

3. 使用您刚才创建的安全域，保护 master 域控制器上的 HTTP 接口。

   示例：HTTP 接口

```
<management-interfaces>
    <http-interface security-realm="HTTPSRealm">
        <http-upgrade enabled="true"/>
        <socket interface="management" port="${jboss.management.http.port:9990}"/>
    </http-interface>
</management-interfaces>
```

4. 使用每个主机控制器上的 SSL/TLS 域连接到主域控制器。

更新用于连接主域控制器的安全域。在服务器没有运行时，修改主机控制器的配置文件，如 **host.xml** 或 **host-slave.xml**。

示例：主机操作系统配置文件

```
<domain-controller>
  <remote security-realm="HTTPSRealm">
    <discovery-options>
      <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

5. 更新每个服务器如何连接其主机控制器。

示例：服务器配置

```
<server name="my-server" group="my-server-group">
  <ssl ssl-protocol="TLS" trust-manager-algorithm="PKIX" truststore-type="PKCS11"
truststore-password="strongP@ssword1"/>
</server>
```

6. 在受管域中配置双向 SSL/TLS。
要启用双向 SSL/TLS，在 master 域控制器的 SSL/TLS 安全域中添加信任存储身份验证方法，请执行以下管理 CLI 命令：

```
/host=master/core-service=management/security-
realm=HTTPSRealm/authentication=truststore:add(keystore-provider="PKCS11",keystore-
password="strongP@ssword1")

reload --host=master
```

您还需要更新每个主机控制器的安全域使其包含 SSL 服务器身份，请执行以下管理 CLI 命令：

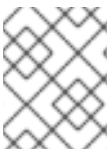
```
/host=host1/core-service=management/security-realm=HTTPSRealm/server-
identity=ssl:add(keystore-provider=PKCS11, keystore-
password="strongP@ssword1",enabled-protocols=["TLSv1.1"])

reload --host=host1
```

重要

您还需要确保将每个主机的证书导入到域控制器的信任存储中。

1.2.16. 使用红帽单点登录保护管理控制台

您也可以使用 Red Hat Single Sign-On 来保护使用 **elytron** 子系统的 JBoss EAP 管理控制台。



注意

只有运行单机服务器时，此功能才可用，且在运行受管域时不支持此功能。不支持使用 Red Hat Single Sign-On 保护管理 CLI。

使用以下步骤设置 Red Hat Single Sign-On 以针对 JBoss EAP 管理控制台验证用户身份。

1. 配置用于 JBoss EAP 管理的 Red Hat Single Sign-On 服务器 。
2. 在 JBoss EAP 上安装 Red Hat Single Sign-On 客户端适配器 。
3. 配置 JBoss EAP 以使用 Red Hat Single Sign-On 。

为 **JBoss EAP 管理配置 Red Hat Single Sign-On 服务器**

1. 下载并安装 Red Hat Single Sign-On 服务器。有关基本说明，请参阅 Red Hat Single Sign-On *入门指南*。
2. 启动 Red Hat Single Sign-On 服务器。
   这个步骤假设您使用端口偏移 **100** 启动服务器。

   ```
   $ RHSSO_HOME/bin/standalone.sh -Djboss.socket.binding.port-offset=100
   ```

3. 登录位于 http://localhost:8180/auth/ 的 Red Hat Single Sign-On 管理控制台。
   如果这是您第一次访问红帽单点登录管理控制台，系统将提示您创建初始管理用户。
4. 创建一个名为 **wildfly-infra** 的新域。
   a. 从域名旁边的下拉菜单中，点 **Add realm**，在 **Name** 字段中输入 **wildfly-infra**，然后点 **Create**。
5. 创建名为 **wildfly-console** 的客户端应用程序。

   重要

   此客户端应用程序的名称*必须*是 **wildfly-console**。

   a. 选择 **Clients** 并点 **Create**。
   b. 在 **Client ID** 字段中输入 **wildfly-console**，然后点 **Save**。
   c. 在出现的 **Settings** 屏幕中，将 **Access Type** 设置为 **public**，将**Valid Redirect URIs** 设置为 **http://localhost:9990/console/***，**Web Origins** 设为 **http://localhost:9990**，然后单击 **Save**。
6. 创建名为 **wildfly-management** 的客户端应用程序。
   a. 选择 **Clients** 并点 **Create**。
   b. 在 **Client ID** 字段中输入 **wildfly-management**，然后点 **Save**。
   c. 在出现的 **Settings** 屏幕中，将 **Access Type** 设置为 **bearer-only** 并点 **Save**。
7. 创建角色来授予 JBoss EAP 管理控制台的访问权限。
   a. 选择 **Roles**，再单击 **Add Role**。
   b. 在 **Role Name** 字段中输入 **ADMINISTRATOR**，然后点击 **Save**。
   此流程使用 **ADMINISTRATOR** 角色，但支持其他角色。如需更多信息，请参阅 JBoss EAP *安全架构*中的基于角色的访问控制部分。

8. 创建一个用户，并为他们分配 **ADMINISTRATOR** 角色。

   a. 选择 **Users** 并点 **Add user**。

   b. 在 **Username** 字段中输入 **jboss**，然后点 **Save**。

   c. 选择 **Credentials** 选项卡并为这个用户设置密码。

   d. 选择 **Role Mappings** 选项卡，选择 **ADMINISTRATOR** 并点 **Add selected** 向此用户添加角色。

在 JBoss EAP 上安装 Red Hat Single Sign-On Client Adapter

1. 从 软件下载页面，下载 JBoss EAP 7 的 Red Hat Single Sign-On 客户端适配器。
2. 将此文件解压缩到 JBoss EAP 安装的根目录中。
3. 执行 **adapter-elytron-install-offline.cli** 脚本来配置您的 JBoss EAP 安装。

```
$ EAP_HOME/bin/jboss-cli.sh --file=adapter-elytron-install-offline.cli
```

重要

此脚本将 **keycloak** 子系统以及 **elytron** 和 **undertow** 子系统中的其他资源添加到 **standalone.xml**。如果您需要使用不同的配置文件，请根据需要修改脚本。

配置 JBoss EAP 以使用 Red Hat Single Sign-On

1. 在 ***EAP_HOME*/bin/** 目录中，创建包含以下内容的文件 **protect-eap-mgmt-services.cli**。

```
# Create a realm for both JBoss EAP console and mgmt interface
/subsystem=keycloak/realm=wildfly-infra:add(auth-server-
url=http://localhost:8180/auth,realm-public-key=REALM_PUBLIC_KEY)

# Create a secure-deployment in order to protect mgmt interface
/subsystem=keycloak/secure-deployment=wildfly-management:add(realm=wildfly-
infra,resource=wildfly-management,principal-attribute=preferred_username,bearer-
only=true,ssl-required=EXTERNAL)

# Protect HTTP mgmt interface with Keycloak adapter
/core-service=management/management-interface=http-interface:undefine-
attribute(name=security-realm)
/subsystem=elytron/http-authentication-factory=keycloak-mgmt-http-
authentication:add(security-domain=KeycloakDomain,http-server-mechanism-factory=wildfly-
management,mechanism-configurations=[{mechanism-name=KEYCLOAK,mechanism-
realm-configurations=[{realm-name=KeycloakOIDCRealm,realm-mapper=keycloak-oidc-
realm-mapper}]}])
/core-service=management/management-interface=http-interface:write-attribute(name=http-
authentication-factory,value=keycloak-mgmt-http-authentication)
/core-service=management/management-interface=http-interface:write-attribute(name=http-
upgrade, value={enabled=true, sasl-authentication-factory=management-sasl-
authentication})

# Enable RBAC where roles are obtained from the identity
/core-service=management/access=authorization:write-attribute(name=provider,value=rbac)
/core-service=management/access=authorization:write-attribute(name=use-identity-
```

```
roles,value=true)

# Create a secure-server in order to publish the JBoss EAP console configuration via mgmt interface
/subsystem=keycloak/secure-server=wildfly-console:add(realm=wildfly-infra,resource=wildfly-console,public-client=true)

# reload
reload
```

2. 在本文件中，将 ***REALM_PUBLIC_KEY*** 替换为公钥的值。要获取这个值，请登录 Red Hat Single Sign-On 管理控制台，选择 **wildfly-infra** realm，导航到 **Realm Settings → Keys**，然后点击 **Public key**。

3. 启动 JBoss EAP。

```
$ EAP_HOME/bin/standalone.sh
```

重要

如果在安装 Red Hat Single Sign-On 客户端适配器时使用除 **standalone.xml** 以外的配置文件，则修改 **adapter-** elytron-install-offline.cli 脚本将使用该配置启动 JBoss EAP。

4. 执行 **protect-eap-mgmt-services.cli** 脚本。

```
$ EAP_HOME/bin/jboss-cli.sh --connect --file=protect-eap-mgmt-services.cli
```

现在，当您访问位于 http://localhost:9990/console/ 的 JBoss EAP 管理控制台时，您会被重定向到 Red Hat Single Sign-On 以登录，然后在成功验证后重新重定向到 JBoss EAP 管理控制台。

1.3. 为旧安全域配置安全审核

您可以使用 audit 模块来监控 **security** 子系统中的事件。审计使用提供程序模块、自定义实施或两者来监控事件。

监控事件后，audit 模块会写入日志文件，读取电子邮件通知，或者使用其他可测量审计机制。

使用管理控制台为安全域配置安全审核设置。

流程

1. 点 **Configuration** 选项卡。
2. 进入 **Subsystems → Security(Legacy)**。
3. 选择可编辑的安全域并点击 **View**。
4. 选择 **Audit** 选项卡，然后按 **Add** 来添加新的审计模块。
5. 为模块设置名称，并使用供应商模块的类名称填写 **Code** 字段。
6. *可选：* 编辑模块并在 **Module Options** 字段中添加键/值对来添加模块选项。按 Enter 键添加新值，然后按 Backspace 删除现有值。

1.4. 使用 ELYTRON 进行安全审核

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

在事件上执行的安全审计类型取决于您的安全域配置。

1.4.1. Elytron 审计日志记录

在使用 **elytron** 子系统启用审计日志记录后，您可以在应用服务器中记录 Elytron 身份验证和授权事件。Elytron 在 **JSON** 中存储审计日志条目，用于存储单个事件或 **SIMPLE** 用于人类可读的文本格式。

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。

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

- 文件审计日志记录
- 定期轮转文件审计日志记录
- 大小轮转文件审计日志记录
- **syslog** 审计日志记录
- 自定义审计日志记录

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。

您可以使用 audit 模块来监控旧安全域的事件。您可以使用管理控制台为旧安全域配置安全审核设置。

其他资源

- 有关使用旧安全系统配置审计的详情，请参考为 旧安全域 配置安全审核。
- 有关管理界面审计日志记录选项的更多信息，请参阅配置*指南中*的管理审计日志记录。
- 有关文件审计日志记录的更多信息，请参阅启用文件审计日志记录。
- 有关定期轮转文件审计日志记录的更多信息，请参阅定期轮转文件审计日志记录。
- 有关大小轮转文件审计日志记录的更多信息，请参阅 大小轮转文件审计日志记录。
- 有关 **syslog** 审计日志记录的更多信息，请参阅 **syslog** 审计日志记录。
- 有关自定义审计日志记录的更多信息，请参阅在 Elytron 中使用自定义安全事件监听程序。

1.4.2. 启用文件审计日志记录

您可以使用 **elytron** 子系统为单机服务器或受管域中的服务器启用文件审计日志记录。

文件审计日志记录在您的文件系统中的单个文件中存储审计日志信息。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**。

流程

1. 创建文件审计日志。

   使用 elytron 子系统创建文件审计日志的示例：

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

2. 将文件审计日志添加到安全域。

   在安全域中添加文件审计日志的命令示例

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

其他资源

- 有关文件审计日志记录器属性的更多信息，请参阅 文件审计日志记录器属性。

1.4.3. 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域域中的服务器启用定期轮转文件审计日志记录。

根据您配置的调度，定期轮转文件审计日志记录自动轮转审计日志。定期轮转文件审计日志记录与默认的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一个额外的属性： **后缀**。

suffix 属性的值是一个使用 **java.time.format.DateTimeFormatter** 格式指定的日期，如 **.yyyyy-MM-dd**。Elytron 会自动从后缀提供的值计算轮转的时间。**elytron** 子系统将后缀附加到日志文件名称的末尾。

流程

1. 创建定期轮转文件审计日志。

   在 elytron 子系统中创建定期轮转文件审计日志的示例

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

2. 将定期轮转文件审计日志记录器添加到安全域。

   在安全域中添加定期轮转文件审计日志记录示例

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

其他资源

- 有关定期轮转文件审计日志记录器属性的信息，请参阅 periodic-rotating-file-audit-log Attributes 表。

1.4.4. 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域中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的文件审计日志记录类似，但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，而 Elytron 会创建新的日志文件。Elytron 从现有日志文件递增后缀。例如，Elytron 将 **audit_log.1** 重命名为 **audit_log.2**。Elytron 继续增加，直到日志文件数量达到最大日志文件数，由 **max-backup-index** 定义。当日志文件超过 **max-backup-index** 值时，Elytron 会移除该文件，例如 **audit_log.99**，即超过限制的文件。

流程

1. 创建大小轮转文件审计日志。

   使用 elytron 子系统创建大小轮转文件审计日志的示例：

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

2. 将大小轮转审计日志记录器添加到安全域。

   使用 elytron 子系统启用大小轮转文件审计日志的示例：

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

其他资源

- 有关大小轮转文件审计日志属性的信息，请参阅 大小轮转文件审计日志属性 表。

1.4.5. 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域中的服务器启用 **syslog** 审计日志记录。使用 **syslog** 审计日志记录时，您要将日志记录结果发送到 **syslog** 服务器，它提供了比登录到本地文件更多的安全选项。

syslog 处理程序指定用于连接 **syslog** 服务器的参数，如 **syslog** 服务器的主机名以及 **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，日志记录器会向 syslog 服务器发送一个 **INFORMATIONAL** 事件到 **syslog** 服务器，如下例所示：

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

<format> 指的是为审计日志日志记录处理器配置的 RFC 格式，它默认为 **RFC5424** 值

流程

1. 添加 **syslog** 处理程序。

   使用 elytron 子系统添加 syslog 处理程序示例：

   ```
   /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：

   通过 TLS 发送日志的 syslog 配置示例

   ```
   /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。

   将 syslog audit logger 添加到安全域的示例

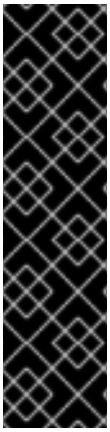
   ```
   /subsystem=elytron/security-domain=<security_domain_name>:write-
   attribute(name=security-event-listener, value=<syslog_audit_log_name>)
   ```

其他资源

- 有关 **syslog-audit-log** 属性的详情，请查看 **syslog-audit-log** Attributes 表。
- 有关通过设置 **ssl-context** 配置启用对 TLS 的支持的更多信息，请参阅使用 **client-ssl-context**。
- 有关 **RFC5424** 的更多信息，请参阅 Syslog 协议。
- 有关 **RFC3164** 的更多信息，请参阅 BSD syslog 协议。

1.4.6. 在 Elytron 中使用自定义安全事件监听程序

您可以使用 Elytron 定义自定义事件监听程序。自定义事件监听器管理传入的安全事件。您可以将事件监听程序用于自定义审计日志记录，也可以使用事件监听程序来针对内部身份存储验证用户。



重要

使用 **module** 管理 CLI 命令来添加和删除模块，仅作为技术预览提供。**module** 命令不适用于在受管域中使用，或者在与远程管理 CLI 进行连接时使用。您必须在生产环境中手动添加或删除模块。

红帽产品服务等级协议(SLA)不支持技术预览功能，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。

流程

1. 创建一个实施 **java.util.function.Consumer<org.wildfly.security.auth.server.event.SecurityEvent>** 接口的类。例如，每当用户成功或身份验证失败时，以下内容会显示一条消息。

   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());
         }
      }
   }
   ```

   示例中的 Java 类会在用户成功或失败时打印消息。

2. 添加 JAR，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添加为 Elytron 的模块。

   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. 在安全域中引用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. 重启服务器。

   ```
   $ reload
   ```

事件监听程序从指定的安全域接收安全事件。

其他资源

- 有关在生产环境中手动添加或删除模块的详情，请参考《*配置指南*》中的 手动创建 自定义模块和手动删除自定义模块。
- 有关添加自定义事件监听程序作为模块的详情，请参考在 Elytron 中添加自定义组件。

1.5. 为应用程序配置单向和双向 SSL/TLS

1.5.1. 面向应用程序的自动自签名证书创建

使用传统安全域时，JBoss EAP 提供了自动生成自签名证书以用于开发目的。

示例：服务器日志显示自签名证书创建

```
15:26:09,031 WARN  [org.jboss.as.domain.management.security] (MSC service thread 1-7)
WFLYDM0111: Keystore /path/to/jboss/standalone/configuration/application.keystore not found, it will
be auto generated on first use with a self signed certificate for host localhost
...
15:26:10,076 WARN  [org.jboss.as.domain.management.security] (MSC service thread 1-2)
WFLYDM0113: Generated self signed certificate at /path/to/jboss/configuration/application.keystore.
Please note that self signed certificates are not secure, and should only be used for testing purposes.
Do not use this self signed certificate in production.
SHA-1 fingerprint of the generated key is
00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff:00:11:22:33
SHA-256 fingerprint of the generated key is
00:11:22:33:44:55:66:77:88:99:00:aa:bb:cc:dd:ee:ff:00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee
...
```

此证书是为测试目的创建的，被分配给应用程序使用的 HTTPS 接口。如果文件在首次访问 HTTPS 接口时不存在，将生成包含证书的密钥存储。

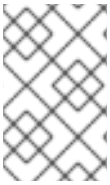
示例：使用自签名证书的默认 ApplicationRealm

```
<security-realm name="ApplicationRealm">
  <server-identities>
    <ssl>
      <keystore path="application.keystore" relative-to="jboss.server.config.dir" keystore-
password="password" alias="server" key-password="password" generate-self-signed-certificate-
host="localhost"/>
    </ssl>
  </server-identities>
  ...
</security-realm>
```

示例：默认 HTTPS 接口配置

```
<subsystem xmlns="urn:jboss:domain:undertow:10.0">
    ...
    <server name="default-server">
        ...
```

```
        <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true"/>
        <host name="default-host" alias="localhost">
        ...
```



注意

如果要禁用自签名证书创建，则需要从服务器密钥存储配置中移除 **generate-self-signed-certificate-host="localhost"**。**generate-self-signed-certificate-host** 属性保存应生成自签名证书的主机名。

警告

此自签名证书仅用于测试目的，不应在生产环境中使用。有关使用 Elytronon 为应用程序配置 SSL/TLS 的更多信息，请参阅使用 Elytron subsystem 为应用程序 启用双向 SSL/TLS 以及 为使用 Elytron 子系统 的应用启用双向 SSL/TLS。有关使用传统安全性为应用程序配置 SSL/TLS 的更多信息，请参阅 为应用程序 启用单向 SSL/TLS，以及使用传统安全性 Realms 为应用程序 启用双向 SSL/TLS，以及使用 Legacy Security Realms 的应用程序启用双向 SSL/TLS 部分。

1.5.2. 使用 Elytron

1.5.2.1. 使用 Elytron subsystem 为应用程序启用单向 SSL/TLS

在 JBoss EAP 中，您可以使用 JBoss EAP 管理 CLI 或管理控制台为已部署的应用程序启用单向 SSL/TLS。

在管理 CLI 中，可以通过两种方式启用单向 SSL/TLS：

- 使用安全命令.
- 使用 **elytron** 子系统命令.

使用安全命令
security enable-ssl-http-server 命令可用于为已部署的应用程序启用单向 SSL/TLS。

示例：向导使用

```
security enable-ssl-http-server --interactive

Please provide required pieces of information to enable SSL:
Key-store file name (default default-server.keystore): keystore.jks
Password (blank generated): secret
What is your first and last name? [Unknown]: localhost
What is the name of your organizational unit? [Unknown]:
What is the name of your organization? [Unknown]:
What is the name of your City or Locality? [Unknown]:
What is the name of your State or Province? [Unknown]:
What is the two-letter country code for this unit? [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown correct y/n
```

```
[y]?
Validity (in days, blank default): 365
Alias (blank generated): localhost
Enable SSL Mutual Authentication y/n (blank n): n

SSL options:
key store file: keystore.jks
distinguished name: CN=localhost, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown,
C=Unknown
password: secret
validity: 365
alias: localhost
Server keystore file keystore.jks, certificate file keystore.pem and keystore.csr file
will be generated in server configuration directory.
Do you confirm y/n: y
```

注意

执行命令后，管理 CLI 将重新加载服务器。

现在应用程序启用了单向 SSL/TLS。

使用 Elytron 子系统命令
在 JBoss EAP 中，您可以使用 **elytron** 子系统和 **undertow** 子系统来为已部署的应用程序启用单向 SSL/TLS。

1. 在 **JBoss EAP** 中配置密钥存储。

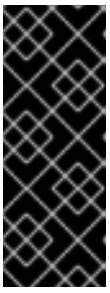
```
/subsystem=elytron/key-store=httpsKS:add(path=/path/to/keystore.jks, credential-reference=
{clear-text=secret}, type=JKS)
```

如果密钥存储文件尚不存在，则可以使用下列命令来生成示例密钥对：

```
/subsystem=elytron/key-store=httpsKS:generate-key-
pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,credential-reference={clear-
text=secret},distinguished-name="CN=localhost")

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

2. 配置一个 **key-manager**，它引用您的 **key-store**。

```
/subsystem=elytron/key-manager=httpsKM:add(key-store=httpsKS,credential-reference=
{clear-text=secret})
```



重要

红帽没有指定上一命令中的 algorithm 属性，因为 Elytron 子系统使用 **KeyManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些关键管理器算法。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **SunX509** 作为密钥管理器算法属性。

3. 配置引用您的 **key-manager** 的 **server-ssl-context**。

```
/subsystem=elytron/server-ssl-context=httpsSSC:add(key-manager=httpsKM, protocols=
["TLSv1.2"])
```

重要

您需要确定您要支持的 SSL/TLS 协议。上面的示例命令使用 **TLSv1.2**。您可以使用 **cipher-suite-filter** 参数指定允许哪些密码套件，**使用-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 **security** 子系统行为不同，它默认为遵循客户端密码套件顺序。

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。

4. 检查并查看 **https-listener** 是否已配置为使用旧安全域进行 SSL 配置。

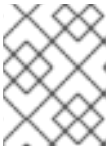
```
/subsystem=undertow/server=default-server/https-listener=https:read-
attribute(name=security-realm)
{
    "outcome" => "success",
    "result" => "ApplicationRealm"
}
```

以上命令显示 **https-listener** 被配置为将 **ApplicationRealm** 传统安全 realm 用于其 SSL 配置。Undertow 无法同时引用传统的安全域和 Elytron 中的 **ssl-context**，因此您必须删除对旧安全域的引用。

注意

如果结果 **未定义**，则不需要在下一步中删除对安全域的引用。

5. 删除对旧安全域的引用，并更新 **https-listener** 以使用 Elytron 中的 **ssl-context**。



注意

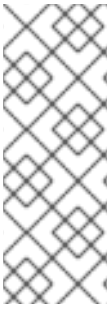
https-listener 需要始终配置了 **security-realm** 或 **ssl-context**。在两个配置之间更改时，命令必须作为一个批处理来执行，如下所示。

```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-
attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context, value=httpsSSC)
run-batch
```

6. 重新加载服务器。

```
reload
```

现在应用程序启用了单向 SSL/TLS。



注意

您可以使用 **disable-ssl-http-server** 命令为已部署的应用程序禁用单向 SSL/TLS。

```
security disable-ssl-http-server
```

此命令不会删除 Elytron 资源。它将系统配置为使用 **ApplicationRealm** legacy security realm 进行其 SSL 配置。

使用管理控制台
您可以使用管理控制台中的 SSL 向导配置 **undertow** 子系统，为应用启用 SSL。

访问向导：

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Web(Undertow) → Server**。
3. 单击要配置的服务器的名称。
4. 点 **View**。
5. 导航到 **Listener → HTTPS Listener**。
6. 选择要启用 SSL 的监听程序，然后点**启用 SSL** 以启动向导。
   向导可以帮助您在以下场景中启用 SSL：
   - 您需要创建证书存储并生成自签名证书。
   - 您需要从 Let 的加密证书颁发机构获取证书。
   - 您已将证书存储在文件系统中，但没有密钥存储配置。
   - 您已经拥有使用有效证书存储的密钥存储配置。

使用向导，您可以选择性地为 mutual 验证创建信任存储。

1.5.2.2. 使用 Elytron subsystem 为应用程序启用双向 SSL/TLS

1. 获取或生成您的客户端密钥存储：

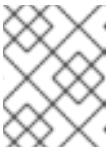
```
$ keytool -genkeypair -alias client -keyalg RSA -keysize 1024 -validity 365 -keystore client.keystore.jks -dname "CN=client" -keypass secret -storepass secret
```

2. 导出客户端证书：

```
keytool -exportcert  -keystore client.keystore.jks -alias client -keypass secret -storepass secret -file /path/to/client.cer
```

3. 为已部署的应用程序启用双向 SSL/TLS。
   在 JBoss EAP 中，可以使用安全命令或使用 **elytron** 子系统命令启用部署应用程序的双向 SSL/TLS。

   a. 使用安全命令.
   security enable-ssl-http-server 命令可用于为已部署的应用程序启用双向 SSL/TLS。

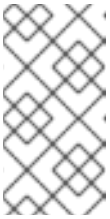


注意

以下示例没有信任链的验证证书。如果您使用可信证书，则在没有问题的情况下可以验证客户端证书。

示例：向导使用

```
security enable-ssl-http-server --interactive

Please provide required pieces of information to enable SSL:
Key-store file name (default default-server.keystore): server.keystore.jks
Password (blank generated): secret
What is your first and last name? [Unknown]: localhost
What is the name of your organizational unit? [Unknown]:
What is the name of your organization? [Unknown]:
What is the name of your City or Locality? [Unknown]:
What is the name of your State or Province? [Unknown]:
What is the two-letter country code for this unit? [Unknown]:
Is CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
correct y/n [y]?
Validity (in days, blank default): 365
Alias (blank generated): localhost
Enable SSL Mutual Authentication y/n (blank n): y
Client certificate (path to pem file): /path/to/client.cer
Validate certificate y/n (blank y): n
Trust-store file name (management.truststore): server.truststore.jks
Password (blank generated): secret

SSL options:
key store file: server.keystore.jks
distinguished name: CN=localhost, OU=Unknown, O=Unknown, L=Unknown,
ST=Unknown, C=Unknown
password: secret
validity: 365
alias: localhost
client certificate: /path/to/client.cer
trust store file: server.trustore.jks
trust store password: secret
Server keystore file server.keystore.jks, certificate file server.pem and server.csr file will
be generated in server configuration directory.
Server truststore file server.trustore.jks will be generated in server configuration
directory.
Do you confirm y/n: y
```



注意

执行命令后，管理 CLI 将重新加载服务器。

要完成双向 SSL/TLS 身份验证，您需要将服务器证书导入到客户端信任存储中，并将您的客户端配置为显示客户端证书。

b. 使用 elytron 子系统命令.
在 JBoss EAP 中，您还可以使用 **elytron** 子系统和 **undertow** 子系统，为已部署的应用启用双向 SSL/TLS。

i. 获取或生成密钥存储。
在 JBoss EAP 中启用双向 SSL/TLS 之前，您必须获取或生成您计划使用的密钥存储、信任存储和证书。

A. 创建服务器密钥存储：

```
/subsystem=elytron/key-store=twoWayKS:add(path=/PATH/TO/server.keystore.jks,credential-reference={clear-text=secret},type=JKS)

/subsystem=elytron/key-store=twoWayKS:generate-key-pair(alias=localhost,algorithm=RSA,key-size=1024,validity=365,credential-reference={clear-text=secret},distinguished-name="CN=localhost")

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

注意

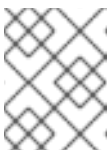
以上命令使用了到密钥存储的绝对路径。或者，您可以使用 **relative-to** 属性来指定基础目录变量，以及 **path** 指定相对路径。

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

B. 导出服务器证书：

```
/subsystem=elytron/key-store=twoWayKS:export-certificate(alias=localhost,path=/path/to/server.cer,pem=true)
```

ii. 为服务器信任存储创建密钥存储，并将客户端证书导入到服务器信任存储中。



注意

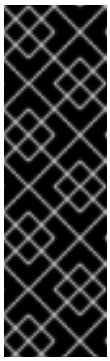
以下示例没有信任链的验证证书。如果您使用可信证书，则在没有问题的情况下可以验证客户端证书。

```
/subsystem=elytron/key-store=twoWayTS:add(path=/path/to/server.truststore.jks,credential-reference={clear-text=secret},type=JKS)
```

```
/subsystem=elytron/key-store=twoWayTS:import-
certificate(alias=client,path=/path/to/client.cer,credential-reference={clear-
text=secret},trust-cacerts=true,validate=false)

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

iii. 配置 **key-manager**，它引用您的密钥存储 **key-store**。

```
/subsystem=elytron/key-manager=twoWayKM:add(key-store=twoWayKS, credential-
reference={clear-text=secret})
```



重要

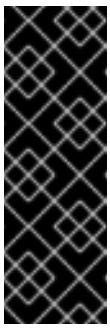
红帽没有指定上一命令中的 algorithm 属性，因为 Elytron 子系统使用 **KeyManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些关键管理器算法。

例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **SunX509** 作为密钥管理器算法属性。

iv. 配置一个 **trust-manager**，它引用您的信任存储 **key-store**。

```
/subsystem=elytron/trust-manager=twoWayTM:add(key-store=twoWayTS)
```



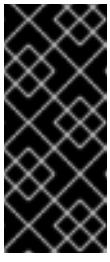
重要

红帽没有在上一个命令中指定 algorithm 属性，因为 Elytron 子系统使用 **TrustManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些信任管理器算法。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **PKIX** 作为信任管理器算法属性。

v. 配置一个 **server-ssl-context**，它引用您的 **key-manager**、**trust-manager** 并启用客户端身份验证：

```
/subsystem=elytron/server-ssl-context=twoWaySSC:add(key-manager=twoWayKM,
protocols=["TLSv1.2"], trust-manager=twoWayTM, need-client-auth=true)
```



重要

您需要确定您要支持的 SSL/TLS 协议。上面的示例命令使用 **TLSv1.2**。您可以使用 **cipher-suite-filter** 参数指定允许哪些密码套件，**使用-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 **security** 子系统行为不同，它默认为遵循客户端密码套件顺序。

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。

vi. 检查并查看 **https-listener** 是否已配置为使用旧安全域进行 SSL 配置。

```
/subsystem=undertow/server=default-server/https-listener=https:read-
attribute(name=security-realm)
{
    "outcome" => "success",
    "result" => "ApplicationRealm"
}
```

以上命令显示 **https-listener** 被配置为将 **ApplicationRealm** 传统安全 realm 用于其 SSL 配置。Undertow 无法同时引用 **elytron** 子系统中的传统安全域和 **ssl-context**。因此，您必须移除对旧安全域的引用。

注意

如果结果 **未定义**，则不需要在下一步中删除对安全域的引用。

vii. 删除对旧安全域的引用，并更新 **https-listener** 以使用 Elytron 中的 **ssl-context**。

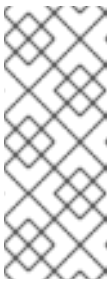
注意

https-listener 需要始终配置了 **security-realm** 或 **ssl-context**。在两个配置之间更改时，命令必须作为一个批处理来执行，如下所示。

```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-
attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-
attribute(name=ssl-context, value=twoWaySSC)
run-batch
```

viii. 重新加载服务器。

```
reload
```



注意

要完成双向 SSL/TLS 身份验证，您需要将服务器证书导入到客户端信任存储中，并将您的客户端配置为显示客户端证书。

```
$ keytool -importcert -keystore client.truststore.jks -storepass secret -
alias localhost -trustcacerts -file /path/to/server.cer
```

ix. 将您的客户端配置为使用客户端证书。
您需要将客户端配置为向服务器提供可信客户端证书，以完成双向 SSL/TLS 身份验证。例如，如果使用浏览器，您需要将可信证书导入到浏览器的信任存储中。

此流程强制双向 SSL/TLS，但不更改应用程序的原始验证方法。

如果要更改原始身份验证方法，请参阅 JBoss EAP 的 *How to Configure Identity Management* 中的 Configure Authentication with Certificates 部分。

现在应用程序启用了双向 SSL/TLS。

注意

您可以使用 **disable-ssl-http-server** 命令为已部署的应用程序禁用双向 SSL/TLS。

```
security disable-ssl-http-server
```

此命令不会删除 Elytron 资源。它将系统配置为使用 **ApplicationRealm** legacy security realm 进行其 SSL 配置。

1.5.3. 在 Elytron 中使用 CRL 配置证书撤销

配置用于启用双向 SSL/TLS 的信任管理器，以将证书撤销列表(CRL)用于 Elytron 中的证书撤销。

先决条件

- 信任管理器被配置为使用双向 SSL/TLS。
- 信任管理器包含用于检查的证书链来撤销。

流程

1. 配置信任管理器，以使用从证书中引用的发布点中获取的 CRL。

```
/subsystem=elytron/trust-manager=twoWayTM:write-attribute(name=certificate-revocation-list,value={})
```

2. 覆盖从证书中引用的发布点中获取的 CRL。

```
/subsystem=elytron/trust-manager=twoWayTM:write-attribute(name=certificate-revocation-list.path, value=intermediate.crl.pem)
```

3. 配置 **trust-manager**，以使用 CRL 进行证书撤销。
   - 如果同时也为证书撤销配置了 OCSP 响应器，在信任管理器中添加值为 **true** 的属性 **ocsp.prefer-crls**，以针对证书撤销使用 CRL：

```
/subsystem=elytron/trust-manager=twoWayTM:write-attribute(name=ocsp.prefer-crls,value="true")
```

   - 如果没有为证书撤销配置 OCSP 响应程序，则配置完成。

其它信息

- 如需 CRL 属性的完整列表，请参阅 trust-manager Attributes。

1.5.4. 在 Elytron 中配置使用 OCSP 的认证撤销

配置用于启用双向 SSL/TLS 的信任管理器，以使用在线证书状态协议(OCSP)响应程序进行证书撤销。OCSP 在 RFC6960 中定义。

当为证书撤销配置 OCSP 响应程序和 CRL 时，则默认调用 OCSP 响应程序。

先决条件

- 信任管理器被配置为使用双向 SSL/TLS。

流程

1. 将信任管理器配置为使用证书中定义的 OCSP 响应程序进行证书撤销。

```
/subsystem=elytron/trust-manager=twoWayTM:write-attribute(name=ocsp,value={})
```

2. 覆盖证书中定义的 OCSP 响应程序。

```
/subsystem=elytron/trust-manager=twoWayTM:write-
attribute(name=ocsp.responder,value="http://example.com/ocsp-responder")
```

其它信息

- 有关属性的完整列表，请参阅 online-certificate-status Attributes。

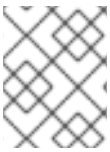
1.5.5. 使用旧的安全性 Realms

重要

作为先决条件，应创建 SSL/TLS 加密密钥和证书并将其放置在可访问的目录中。此外，也应可以访问密钥存储别名和密码等相关信息，以及所需的密码套件。有关生成 SSL/TLS 密钥和证书的示例，请参考为管理接口 设置双向 SSL/TLS 中的前两个步骤。有关 HTTPS 侦听器（包括密码套件）的更多信息，请参阅 HTTPS Listener Reference 部分。

1.5.5.1. 为使用旧安全 Realms 的应用启用单向 SSL/TLS

本例假定密钥存储 **identity.jks** 已复制到服务器配置目录中，并且配置了给定的属性。管理员应该用自己的示例值替换它们。



注意

显示的管理 CLI 命令假定您正在运行 JBoss EAP 单机服务器。有关为 JBoss EAP 受管域使用管理 CLI 的详情，请查看 JBoss EAP *管理 CLI 指南*。

1. 首先添加和配置 HTTPS 安全域。配置 HTTPS 安全域后，在引用安全域的 **undertow** 子系统中配置 **https-listener** ：

```
batch
```

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

/core-service=management/security-realm=HTTPSRealm/server-identity=ssl:add(keystore-path=identity.jks, keystore-relative-to=jboss.server.config.dir, keystore-password=password1, alias=appserver)

/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=security-realm, value=HTTPSRealm)

run-batch
```

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。

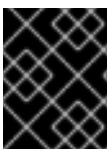
2. 重启 JBoss EAP 实例以使更改生效。

1.5.5.2. 为使用旧安全 Realms 的应用启用双向 SSL/TLS

为应用程序设置双向 SSL/TLS 遵循为 管理接口 设置双向 SSL/TLS 中的许多 相同步骤。要为应用程序设置双向 SSL/TLS，您需要执行以下操作：

1. 为客户端和服务器生成存储
2. 导出客户端和服务器的证书
3. 将证书导入到而非信任存储中
4. 在使用服务器的密钥存储和信任存储的服务器上定义安全域，如 **CertificateRealm**
5. 更新 **undertow** 子系统，以使用安全域并要求客户端验证

为管理接口设置双向 SSL/TLS 中介绍了前四个步骤。



重要

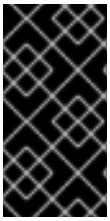
如果服务器尚未重新加载，因为添加了新的安全域，您必须重新加载服务器，然后执行下一步。

更新 Undertow 子系统
创建了密钥存储、证书、信任存储和安全域后，您需要将 HTTPS 侦听器添加到 **undertow** 子系统，使用您创建的安全域并需要客户端验证：

```
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=security-realm, value=CertificateRealm)

/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=verify-client, value=REQUIRED)
```

重要

您必须重新加载服务器，以便这些更改生效。



重要

任何为应用启用了双向 SSL/TLS 的客户端连接到 JBoss EAP 实例都必须有权访问客户端证书或密钥存储，而该证书包含在服务器信任存储中的客户端密钥存储中。如果客户端使用浏览器连接 JBoss EAP 实例，您需要将该证书或密钥存储导入到浏览器的证书管理器中。

注意

除了使用双向 SSL/TLS 和应用的双向 SSL/TLS 一起，请参阅 JBoss EAP *如何* 配置安全域以使用基于证书的身份验证一节中的更多有关在应用中使用基于证书的身份验证 的详细信息。

1.6. 使用 CLI 安全性命令为应用程序启用 HTTP 验证

在 JBoss EAP 中，可通过 **安全性 enable-http-auth-http-server** 命令，为 undertow 安全域启用 HTTP 身份验证。默认情况下，此命令会将应用 HTTP 工厂与指定的安全域关联。

示例：在 Undertow 安全域上启用 HTTP 身份验证

```
security enable-http-auth-http-server --security-domain=SECURITY_DOMAIN

Server reloaded.
Command success.
Authentication configured for security domain SECURITY_DOMAIN
http authentication-factory=application-http-authentication
security-domain=SECURITY_DOMAIN
```

注意

执行命令后，管理 CLI 将重新加载服务器并重新连接该服务器。

如果 HTTP 工厂已存在，则工厂将更新为使用 **--mechanism** 参数定义的机制。

1.6.1. 为管理接口禁用 HTTP 验证

此流程描述了如何为管理界面禁用 HTTP 验证。

流程

- 要删除活动的 HTTP 身份验证工厂，请使用以下命令。

```
security disable-http-auth-http-server --security-domain=SECURITY_DOMAIN
```

- 或者，您可以使用以下命令从活动 SASL 身份验证工厂中删除特定的机制：

```
security disable-http-auth-http-server --mechanism=MECHANISM --security-
domain=SECURITY_DOMAIN
```

1.7. SASL 身份验证机制

使用 简单的身份验证和安全层(SASL) 验证机制来定义使用 **elytron** 子系统验证与 JBoss EAP 服务器的连接的机制，以及用于连接到服务器的客户端。客户端可以是其他 JBoss EAP 实例，也可以是 Elytron 客户端。JBoss EAP 中的 SASL 身份验证机制也显著地用于 Elytron 与 Remoting subsystem 集成。

1.7.1. 选择 SASL 身份验证机制

注意

虽然 JBoss EAP 和 Elytron 客户端可处理各种 SASL 身份验证机制，但必须确保支持以下机制。有关 SASL 验证机制的支持级别请查看此列表。

您所使用的验证机制取决于您的环境和所需的身份验证方法。以下列表总结了一些支持的 SASL 验证机制的使用：

匿名

未经身份验证的客户端访问。

DIGEST-MD5

使用 HTTP 摘要身份验证作为 SASL 机制。

EXTERNAL

使用在请求上下文中表示的身份验证凭据。例如： IPsec 或 TLS 身份验证。

从 GS开始的机制

使用 Kerberos 进行身份验证。

JBOSS-LOCAL-USER

通过测试客户端具有与运行 JBoss EAP 服务器的本地用户相同的文件访问来提供身份验证。这对在同一计算机上运行的其他 JBoss EAP 实例有用。

OAUTHBEARER

使用 OAuth 提供的身份验证作为 SASL 机制。

PLAIN

纯文本用户名和密码身份验证。

以 SCRAM开头的机制

使用指定散列功能的 salt 挑战响应机制 (SCRAM)

以 -PLUS结尾的机制

表示特定身份验证机制的频道绑定变体。当底层连接使用 SSL/TLS 时，您应该使用这些变体。

有关各个 SASL 身份验证机制的更多信息，请参阅 IANA SASL 机制列表和单个机制 memos。

1.7.2. 在服务器范围内配置 SASL 身份验证机制

在服务器端配置 SASL 验证机制是使用 SASL 身份验证工厂完成的。

需要两种配置：

- 指定身份验证机制的 **sasl-authentication-factory**。
- **可配置-sasl-server-factory**，用于聚合一个或多个 **sasl-authentication-factory**，并配置机制属性，并选择性地应用过滤器来启用或禁用某些验证机制。

以下示例演示了创建一个新的 **configurable-sasl-server-factory** 和 **sasl-authentication-factory**，它使用 DIGEST-MD5 作为应用程序客户端的 SASL 身份验证机制。

```
/subsystem=elytron/configurable-sasl-server-factory=mySASLServerFactory:add(sasl-server-factory=elytron)

/subsystem=elytron/sasl-authentication-factory=mySASLAuthFactory:add(sasl-server-factory=mySASLServerFactory,security-domain=ApplicationDomain,mechanism-configurations=[{mechanism-name=DIGEST-MD5,mechanism-realm-configurations=[{realm-name=ApplicationRealm}]}])
```

1.7.3. 在客户端方中指定 SASL 身份验证机制

客户端使用的 SASL 验证机制通过 **sasl-mechanism-selector** 指定。您可以指定由客户端连接的服务器公开的任何支持的 SASL 身份验证机制。

sasl-mechanism-selector 在配置了身份验证的 Elytron 配置中定义：

- 在 **elytron** 子系统中，这是 **authentication-configuration** 的属性。例如：

  ```
  /subsystem=elytron/authentication-configuration=myAuthConfig:write-attribute(name=sasl-mechanism-selector,value="DIGEST-MD5")
  ```

  在配置 SSL 或使用 **elytron**的 TLS 中，可以看到将 **身份验证** 与 **sasl-mechanism-selector** 搭配使用 的示例。

- 对于 Elytron Client，它在客户端配置文件的 **authentication- configuration s** 配置元素下指定，通常称为 **wildfly-config.xml**。例如：

  ```
  <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" />
        ...
        </configuration>
      </authentication-configurations>
    </authentication-client>
  </configuration>
  ```

有关使用 Elytron 客户端配置客户端身份验证的更多信息，请参阅*如何配置身份管理*。

SASL-mechanism-selector Grammar
sasl-mechanism-selector 的选择器字符串有特定的 grammar。

在简单形式中，通过按顺序列出各个机制的名称（用空格分隔）来指定各个机制。例如，要将 DIGEST-MD5、SCRAM-SHA-1 和 SCRAM-SHA-256 指定为允许的验证机制，请使用以下字符串：**DIGEST-MD5 SCRAM-SHA-1 SCRAM-SHA-256**。

对 grammar 的更多高级用法可使用以下特殊令牌：

- **#ALL** ：所有机制。

- **#FAMILY(*NAME*)**: Mechanisms 属于指定机制系列。例如，系列可以是 DIGEST、EAP、GS2、SCRAM 或 IEC-ISO-9798。
- **#PLUS**: 使用频道绑定的机制。例如： SCRAM-SHA-*XXX*-PLUS 或 GS2-*XXX*-PLUS.
- **#MUTUAL**: Mechanisms 以某种方式验证服务器，例如使服务器能够证明服务器知道该密码。**#MUTUAL** 包括 系列，如 **#FAMILY(SCRAM)** 和 **#FAMILY(GS2)**。
- **#HASH(*ALGORITHM*)**: 使用指定的哈希算法的机制机制。例如，算法可以是 MD5、SHA-1、SHA-256、SHA-384 或 SHA-512。

以上令牌和名称也可以用于以下操作和 predicates：

- **-**: 表示禁止
- **!**: 反转
- **&&**: 和
- ||: 或
- **==**: 等于
- **?**: : if if
- **#TLS**: 在 TLS 处于活跃状态时是 true，否则为 false。

以下是 **sasl-mechanism-selector** 字符串及其含义的一些示例：

- **#TLS && !#MUTUAL** : 当 TLS 处于活跃状态时，所有没有相互身份验证的机制。
- **#ALL -ANONYMOUS** ：除 ANONYMOOUS 以外的所有机制。
- **SCRAM-SHA-1 SCRAM-SHA-256** ：按顺序添加这两种机制。
- **(SCRAM-SHA-1 || SCRAM-SHA-256)** ：根据供应商或服务器提供的顺序添加两种机制。
- **!#HASH(MD5)**:不使用 MD5 哈希算法的任何机制。
- **#FAMILY(DIGEST)** ：任何摘要机制。

1.7.4. 配置 SASL 身份验证机制属性

您可以在服务器端和客户端上配置身份验证机制属性。

- 在服务器端，您可以在 **configurable-sasl-server-factory** 中定义身份验证机制属性。以下示例定义了 **com.sun.security.sasl.digest.utf8** 属性，值设为 **false**。

```
/subsystem=elytron/configurable-sasl-server-factory=mySASLServerFactory:map-
put(name=properties,key=com.sun.security.sasl.digest.utf8,value=false)
```

- 在客户端，您可以在客户端的身份验证配置中定义验证机制属性：
  - 在 **elytron** 子系统中，定义 **身份验证配置中的验证机制** 属性。以下示例定义了 **wildfly.sasl.local-user.quiet-auth** 属性，值设为 **true**。

```
/subsystem=elytron/authentication-configuration=myAuthConfig:map-
put(name=mechanism-properties,key=wildfly.sasl.local-user.quiet-auth,value=true)
```

- 对于 Elytron 客户端，身份验证机制属性在客户端配置文件中的 **authentication-configuration s** 配置元素下指定，通常命名为 **wildfly-config.xml**。例如：

```
...
<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>
    ...
  </configuration>
</authentication-configurations>
...
```

您可以在 Java 文档 中看到标准 Java SASL 验证机制属性列表 。其他 JBoss EAP 特定的 SASL 身份验证机制属性列在身份验证机制参考 中。

1.8. ELYTRON 与 MODCLUSTER 子系统集成

elytron 子系统公开的其中一个安全功能是客户端 **ssl-context**，可用于配置 **modcluster** 子系统以使用 SSL/TLS 与负载平衡器通信。

在保护应用服务器和负载均衡器之间的通信时，您需要定义一个客户端 **ssl-context**，以便：

- 定义保存证书链的信任存储，该链将用于验证负载均衡器的证书。
- 定义信任管理器，针对负载均衡器的证书执行验证。

1.8.1. 定义客户端 SSL 上下文并配置 ModCluster subsystem

以下流程要求配置信任存储和信任管理器。有关创建这些的详情，请参阅创建 Elytron Truststore 和 Create a Elytron Trust Manager。

1. 创建客户端 SSL 上下文。
   在使用 SSL/TLS 连接到负载均衡器时，**modcluster** 子系统将使用此 SSL 上下文：

```
/subsystem=elytron/client-ssl-context=modcluster-client-ssl-context:add(trust-
manager=default-trust-manager)
```

2. 使用以下选项之一引用新创建的客户端 SSL 上下文：

   - 通过设置 **ssl-context** 来配置 **modcluster** 子系统。

```
/subsystem=modcluster/mod-cluster-config=configuration:write-attribute(name=ssl-
context, value=modcluster-client-ssl-context)
```

   - 通过定义 **mod-cluster** 过滤器的 **ssl-context** 属性来配置 **undertow** 子系统。

```
/subsystem=undertow/configuration=filter/mod-cluster=modcluster:write-attribute(name=ssl-context,value=modcluster-client-ssl-context)
```

3. 重新加载服务器。

```
reload
```

要配置 **modcluster** 子系统 和使用双向身份验证，还需要配置主管理器。

1. 创建密钥存储。

```
/subsystem=elytron/key-store=twoWayKS:add(path=/path/to/client.keystore.jks, credential-reference={clear-text=secret},type=JKS)
```

2. 配置密钥管理器。

```
/subsystem=elytron/key-manager=twoWayKM:add(key-store=twoWayKS, algorithm="SunX509", credential-reference={clear-text=secret})
```

3. 创建客户端 SSL 上下文。

```
/subsystem=elytron/client-ssl-context=modcluster-client-ssl-context:add(trust-manager=default-trust-manager, key-manager=twoWayKM)
```

注意

如果您已有客户端 SSL 上下文，您可以按照以下方法将 **key-manager** 添加到其中：

```
/subsystem=elytron/client-ssl-context=modcluster-client-ssl-context:write-attribute(name=key-manager, value=twoWayKM)
```

4. 重新加载服务器。

```
reload
```

1.9. ELYTRON 与 JGROUPS 子系统集成

在 **jgroups** 子系统中定义授权或加密协议时，可能会引用 **elytron** 子系统中的组件。有关配置这些协议的完整说明，请参阅*配置指南*中的保护集群部分。

1.10. ELYTRON 与 REMOTING SUBSYSTEM 集成

1.10.1. Elytron 与补救连接器集成

补救连接器由 SASL 身份验证工厂、套接字绑定和可选的 SSL 上下文指定。特别是，连接器的属性如下：

sasl-authentication-factory

对 SASL 身份验证工厂的引用，用于向这个连接器验证请求。有关创建此工厂的更多信息，请参阅创建 Elytron Authentication Factory。

socket-binding

对套接字绑定的引用，详细描述了连接器应该侦听传入请求的接口和端口。

ssl-context

此连接器使用的服务器端 SSL 上下文的可选引用。SSL 上下文包含要使用的服务器密钥管理器和信任管理器，应在需要 SSL 的实例上定义。

例如，可以添加连接器，如下所示，***SASL_FACTORY_NAME*** 是已经定义的身份验证工厂，***SOCKET_BINDING_NAME*** 是现有的套接字绑定。

```
/subsystem=remoting/connector=CONNECTOR_NAME:add(sasl-authentication-factory=SASL_FACTORY_NAME,socket-binding=SOCKET_BINDING_NAME)
```

如果需要 SSL，可以使用 **ssl-context** 属性来引用预配置的 **server-ssl-context**，如下所示。

```
/subsystem=remoting/connector=CONNECTOR_NAME:add(sasl-authentication-factory=SASL_FACTORY_NAME,socket-binding=SOCKET_BINDING_NAME,ssl-context=SSL_CONTEXT_NAME)
```

1.10.1.1. 启用单向 SSL/TLS 使用 elytron 子系统重新移动连接器

以下 SASL 机制支持频道绑定到外部安全频道，如 SSL/TLS：

- GS2-KRB5-PLUS
- SCRAM-SHA-1-PLUS
- SCRAM-SHA-256-PLUS
- SCRAM-SHA-384-PLUS
- SCRAM-SHA-512-PLUS

要使用任何这些机制，您可以配置自定义 SASL 工厂，或修改预定义的 SASL 身份验证因素之一。可以在客户端使用 SASL 机制选择器来指定适当的 SASL 机制。

先决条件

- 配置了 **密钥存储**。
- 配置了 **key-manager**。
- 配置了 **server-ssl-context**，它引用定义的 **key-manager**

流程

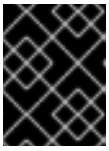
1. 为连接器创建 **socket-binding**。以下命令定义了在端口 **11199** 上侦听的 **oneWayBinding** 绑定。

```
/socket-binding-group=standard-sockets/socket-binding=oneWayBinding:add(port=11199)
```

2. 创建引用 SASL 身份验证工厂、之前创建的套接字绑定和 SSL 上下文的连接器。

-

```
/subsystem=remoting/connector=oneWayConnector:add(sasl-authentication-
factory=SASL_FACTORY,socket-binding=oneWayBinding,ssl-context=SSL_CONTEXT)
```



重要

如果您*同时*定义了 **security-realm** 和 **ssl-context**，JBoss EAP 将使用 **ssl-context** 提供的 SSL/TLS 配置。

3. 配置客户端以信任服务器证书。Elytron Client Side One Way Example 提供了一个常规的示例客户端。这个示例使用客户端 **trust-store** 来配置 **ssl-context**。

其他资源

- **key-store**
- **key-manager**
- **server-ssl-context**

1.10.1.2. 启用双向 SSL/TLS 使用 elytron 子系统重新移动连接器

以下 SASL 机制支持频道绑定到外部安全频道，如 SSL/TLS：

- GS2-KRB5-PLUS
- SCRAM-SHA-1-PLUS
- SCRAM-SHA-256-PLUS
- SCRAM-SHA-384-PLUS
- SCRAM-SHA-512-PLUS

要使用任何这些机制，您可以配置自定义 SASL 工厂，或修改预定义的 SASL 身份验证因素之一以提供这些机制。可以在客户端使用 SASL 机制来指定适当的 SASL 机制。

先决条件

- 为客户端和服务器证书配置单独的 **key-store** 组件。
- 为服务器 **key-store** 配置了一个 **key-manager**。
- 为服务器 **trust-store** 配置了一个 **trust-manager**。
- 配置了 **server-ssl-context**，它引用了定义的 **key-manager** 和 **trust-manager**。

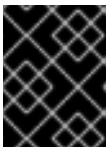
流程

1. 为连接器创建 **socket-binding**。以下命令定义了侦听端口 **11199** 的 **twoWayBinding** 绑定。

```
/socket-binding-group=standard-sockets/socket-binding=twoWayBinding:add(port=11199)
```

2. 创建引用 SASL 身份验证工厂、之前创建的套接字绑定和 SSL 上下文的连接器。

```
/subsystem=remoting/connector=twoWayConnector:add(sasl-authentication-
factory=SASL_FACTORY,socket-binding=twoWayBinding,ssl-context=SSL_CONTEXT)
```



重要

如果您*同时*定义了 **security-realm** 和 **ssl-context**，JBoss EAP 将使用 **ssl-context** 提供的 SSL/TLS 配置。

3. 将您的客户端配置为信任服务器证书，并将其证书提供给服务器。
   您需要将客户端配置为向服务器提供可信客户端证书，以完成双向 SSL/TLS 身份验证。例如，如果使用浏览器，您需要将可信证书导入到浏览器的信任存储中。Elytronon Client Side Two way Example 包括了一个常规的示例客户端。这个示例使用客户端 **trust-store** 和 **key-store** 来配置 **ssl-context**。

现在，在 remoting connector 上启用了双向 SSL/TLS。

其他资源

- **key-store**
- **key-manager**
- **trust-manager**
- **trust-store**
- **server-ssl-context**
- SASL 机制选择器

1.10.2. Elytron 与补救 HTTP 连接器集成

通过引用 **undertow** 子系统中的连接器和 **elytron** 子系统中定义的 SASL 身份验证工厂来指定远程 HTTP 连接。HTTP 连接器为基于 HTTP 升级的补救连接器提供配置，并连接到通过 **connector-ref** 属性指定的 HTTP 监听程序。

http-connector 的属性如下：

connector-ref

对预定义的 **undertow** 侦听器的引用。

sasl-authentication-factory

对 SASL 身份验证工厂的引用，用于向这个连接器验证请求。有关创建此工厂的更多信息，请参阅创建 Elytron Authentication Factory。

例如，可以添加 **http-connector**，其中 ***CONNECTOR_NAME*** 引用 **undertow** 侦听器，***SASL_FACTORY_NAME*** 是 **elytron** 子系统中已经定义了的身份验证工厂。

```
/subsystem=remoting/http-connector=HTTP_CONNECTOR_NAME:add(connector-
ref=CONNECTOR_NAME,sasl-authentication-factory=SASL_FACTORY_NAME)
```

1.10.2.1. 在 remoting HTTP 连接器中启用单向 SSL

以下 SASL 机制支持频道绑定到外部安全频道，如 SSL/TLS：

- GS2-KRB5-PLUS
- SCRAM-SHA-1-PLUS
- SCRAM-SHA-256-PLUS
- SCRAM-SHA-384-PLUS
- SCRAM-SHA-512-PLUS

要使用上述任何机制，可以配置自定义 SASL 工厂，或者可以修改其中一个预定义的 SASL 身份验证因素来提供这些机制。可以在客户端使用 SASL 机制选择器来指定适当的 SASL 机制。

先决条件

- 配置了 **密钥存储**。
- 配置了 **key-manager**。
- 配置了 **server-ssl-context**，它引用了定义的 **key-manager**。

流程

1. 检查 **https-listener** 是否已配置为使用旧安全域进行 SSL 配置。

```
/subsystem=undertow/server=default-server/https-listener=https:read-
attribute(name=security-realm)
{
    "outcome" => "success",
    "result" => "ApplicationRealm"
}
```

以上命令显示 **https-listener** 被配置为将 **ApplicationRealm** 传统安全 realm 用于其 SSL 配置。Undertow 无法同时引用传统的安全域和 Elytron 中的 **ssl-context**，因此您必须删除对旧安全域的引用。

注意

如果结果 **未定义**，则不需要在下一步中删除对安全域的引用。

2. 删除对旧安全域的引用，并更新 **https-listener** 以使用 Elytron 中的 **ssl-context**。

注意

https-listener 需要始终配置了 **security-realm** 或 **ssl-context**。在两个配置之间更改时，命令必须作为一个批处理来执行，如下所示。

```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-
attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context, value=SERVER_SSL_CONTEXT)
run-batch
```

3. 创建 HTTP 连接器来引用 HTTPS 侦听器和 SASL 身份验证工厂。

```
/subsystem=remoting/http-connector=ssl-http-connector:add(connector-ref=https,sasl-authentication-factory=SASL_FACTORY)
```

4. 重新加载服务器。

```
reload
```

5. 配置客户端以信任服务器证书。例如，如果使用浏览器，您需要将可信证书导入到浏览器的信任存储中。

其他资源

- **key-store**
- **key-manager**
- **server-ssl-context**
- 自定义 SASL 工厂

1.10.2.2. 在 remoting HTTP 连接器中启用双向 SSL/TLS

以下 SASL 机制支持频道绑定到外部安全频道，如 SSL/TLS：

- GS2-KRB5-PLUS
- SCRAM-SHA-1-PLUS
- SCRAM-SHA-256-PLUS
- SCRAM-SHA-384-PLUS
- SCRAM-SHA-512-PLUS

要使用上述任何机制，可以配置自定义 SASL 工厂，或者可以修改其中一个预定义的 SASL 身份验证因素来提供这些机制。可以在客户端使用 SASL 机制选择器来指定适当的 SASL 机制。

先决条件

- 为客户端和服务器证书配置单独的 **key-store** 组件。
- 为服务器 **key-store** 配置了一个 **key-manager**。
- 为服务器 **trust-store** 配置 **trust-manager**。
- 配置了 **server-ssl-context**，它引用了定义的 **key-manager** 和 **trust-manager**。

流程

1. 检查 **https-listener** 是否已配置为使用旧安全域进行 SSL 配置。

```
/subsystem=undertow/server=default-server/https-listener=https:read-attribute(name=security-realm)
```

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

以上命令显示 **https-listener** 被配置为将 **ApplicationRealm** 传统安全 realm 用于其 SSL 配置。Undertow 无法同时引用传统的安全域和 Elytron 中的 **ssl-context**，因此您必须删除对旧安全域的引用。

注意

如果结果 **未定义**，则不需要在下一步中删除对安全域的引用。

2. 删除对旧安全域的引用，并更新 **https-listener** 以使用 Elytron 中的 **ssl-context**。

注意

https-listener 需要始终配置了 **security-realm** 或 **ssl-context**。在两个配置之间更改时，命令必须作为一个批处理来执行，如下所示。

```
batch
/subsystem=undertow/server=default-server/https-listener=https:undefine-attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-context, value=SERVER_SSL_CONTEXT)
run-batch
```

3. 创建 HTTP 连接器来引用 HTTPS 侦听器和 SASL 身份验证工厂。

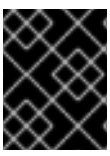
```
/subsystem=remoting/http-connector=ssl-http-connector:add(connector-ref=https,sasl-authentication-factory=SASL_FACTORY)
```

4. 重新加载服务器。

```
reload
```

5. 将您的客户端配置为信任服务器证书，并将其证书提供给服务器。
通过将客户端配置为向服务器提供可信客户端证书，完成双向 SSL/TLS 身份验证。例如，如果使用浏览器，您需要将可信证书导入到浏览器的信任存储中。

现在，在 remoting HTTP 连接器上启用了双向 SSL/TLS。



重要

如果您*同时*定义了 **security-realm** 和 **ssl-context**，JBoss EAP 将使用 **ssl-context** 提供的 SSL/TLS 配置。

其他资源

- **key-store**
- **key-manager**

- **trust-manager**
- **trust-store**
- **server-ssl-context**
- 自定义 SASL 工厂

1.10.3. Elytron 与远程出站连接器集成

远程出站连接通过出站套接字绑定和验证上下文指定。验证上下文提供连接所需的所有安全信息。特别是 **remote-outbound-connection** 的属性如下：

- **出站套接字-binding-ref** - 出站套接字绑定的名称，用于确定连接的目标地址和端口。
- **authentication-context** - 对验证上下文的引用，该上下文包含验证配置和定义的 SSL 上下文（如果存在，则为连接需要）。有关定义验证上下文的详情，请参考 创建身份验证上下文。

例如，可以添加 **remote-outbound-connection**，其中 *OUTBOUND_SOCKET_BINDING_NAME* 是已经定义的 **出站套接字-binding** 和 *AUTHENTICATION_CONTEXT_NAME*，已在 **elytron** 子系统配置中定义的 **验证上下文**。

```
/subsystem=remoting/remote-outbound-
connection=OUTBOUND_CONNECTION_NAME:add(authentication-
context=AUTHENTICATION_CONTEXT_NAME, outbound-socket-binding-
ref=OUTBOUND_SOCKET_BINDING_NAME)
```

其他资源

- 创建身份验证上下文

1.11. 用于 SSL/TLS 的额外 ELYTRON 组件

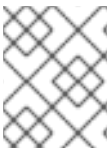
以下介绍了配置单向 SSL/TLS 和双向 SSL/TLS 的基本概念：

- 使用 Elytron subsystem 为应用程序启用单向 SSL/TLS
- 使用 Elytron subsystem 为应用程序启用双向 SSL/TLS
- 使用 Elytron subsystem 为管理接口启用单向 SSL/TLS
- 使用 Elytron subsystem 为管理接口启用双向 SSL/TLS

Elytron 还提供用于配置 SSL/TLS 的一些其他组件。

1.11.1. 使用 ldap-key-store

ldap-key-store 允许您使用存储在 LDAP 服务器中的密钥存储。您可以像使用密钥存储一样使用 **ldap-key-store** 。



注意

无法使用 Jakarta Management ObjectName 来解密 LDAP 凭证。反之，可以使用凭证存储来保护凭证。有关凭证存储的详情，请参考 Elytron 中的凭证存储。

要创建并使用 **ldap-key-store** ：

1. 配置 **dir-context**。
   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。

   示例：dir-context

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

2. 配置 **ldap-key-store**。
   当您配置 **ldap-key-store** 时，您需要指定用于连接 LDAP 服务器的 **dir-context**，以及如何定位存储在 LDAP 服务器中的密钥存储。至少，这需要您指定 **search-path**。

   示例： ldap-key-store

   ```
   /subsystem=elytron/ldap-key-store=ldapKS:add(dir-context=exampleDC, search-
   path="ou=Keystores,dc=wildfly,dc=org")
   ```

3. 使用 **ldap-key-store**。
   定义了 **ldap- key-store** 后，您可以在可以使用密钥存储的同一位置使用它。例如，您可以在为应用程序配置 单向 SSL/TLS 和双向 SSL/TLS 时使用 **ldap-key-store**。

有关 **ldap-key-store** 和其他 Elytron 组件的完整列表，请参阅 Elytron subsystem 组件参考。

1.11.2. 使用 filtering-key-store

借助 **filtering- key-store** ，您可以从现有密钥存储 中公开别名的子集，并将其在同一位置使用 **密钥存储**。例如，如果密钥存储包含 **alias1**、 **alias2** 和 **alias3**，但您只想公开 **alias1** 和 **alias3**，则 **filtering-key-store** 可以为您提供多种操作方式。

创建 **filtering-key-store** ：

1. 配置 **密钥存储**。

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

2. 配置 **filtering-key-store**。
   当您配置 **filtering-key-store** 时，您可以指定要过滤的键值和用于过滤 **key - store** 中的别名的 **key-store**。过滤器可使用以下格式之一指定：

   - **alias1,alias3**，它是一个以逗号分隔的别名列表。
   - **ALL:-alias2**，它公开密钥存储中的所有别名，除了列出的别名之外。
   - **NONE:+alias1:+alias3**，该密钥存储中不会公开密钥存储中的别名，唯一的原因除外。
     这个示例使用逗号分隔的列表来公开 **alias1** 和 **alias3**。

     ```
     /subsystem=elytron/filtering-key-store=filterKS:add(key-store=myKS, alias-
     filter="alias1,alias3")
     ```

注意

alias-filter 属性区分大小写。由于使用混合案例或大写别名（如 **elytronAppServer** ）可能无法被某些密钥存储提供程序识别，因此建议使用小写别名，如 **elytronappserver**。

3. 使用 **filtering-key-store**。
   定义了 **filtering-key-store** 后，您可以在可以使用密钥存储的 **同一位置** 使用它。例如，您可以在为应用程序配置 单向 SSL/TLS 和双向 SSL/TLS 时，使用 **filtering-key-store**。

有关 **filtering-key-store** 和其他 Elytron 组件的完整列表，请参阅 Elytron subsystem 组件参考。

1.11.3. 重新加载密钥存储

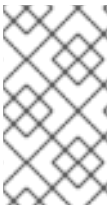
您可以从管理 CLI 重新加载 JBoss EAP 中配置的密钥存储。当您对密钥存储引用的证书进行了更改时，这很有用。

重新加载密钥存储：

```
/subsystem=elytron/key-store=httpsKS:load
```

1.11.4. 重新初始化密钥管理器

您可以从管理 CLI 重新初始化在 JBoss EAP 中配置的 **key-manager**。当您在密钥存储资源提供的证书中进行了更改，并且您希望在不重启服务器的情况下将此更改应用到新的 SSL 连接时，这非常有用。



注意

如果 **key-store** 基于文件，则必须首先载入它。

```
/subsystem=elytron/key-store=httpsKS:load()
```

要重新初始化 **key-manager**:

```
/subsystem=elytron/key-manager=httpsKM:init()
```

1.11.5. 重新初始化信任管理器

您可以从管理 CLI 或管理控制台重新初始化在 JBoss EAP 中配置的 **trust-manager**。当您对密钥存储资源提供的证书进行更改并且想要在不重启服务器的情况下将更改应用到新的 SSL 连接时，这将非常有用。

从管理 CLI 重新初始化 Trust Manager

注意

如果 **key-store** 基于文件，则必须首先载入它。

```
/subsystem=elytron/key-store=httpsKS:load()
```

要重新初始化 **trust-manager** ：

```
/subsystem=elytron/trust-manager=httpsTM:init()
```

从管理控制台重新初始化 Trust Manager

1. 导航到管理控制台，再单击 **Runtime** 选项卡。
2. 在 **monitor** 列中，点 **Security(Elytron)**。
3. 在 **Security** 列中，点 **SSL → View**。
4. 在导航窗格中，点 **Trust Manager**。
5. 单击屏幕右上角的 **Initialize** 以重新初始化 **trust-manager**。

1.11.6. Keystore Alias

alias 表示存储中存储的 secret 或凭证。如果您使用 **key-store** 组件向 **elytron** 子系统添加密钥存储，您可以使用与 **key-store** 操作相关的**别名**。

别名操作的不同操作包括：

- **read-alias** - 从密钥存储读取别名。
- **read-aliases** - 从密钥存储读取别名。
- **remove-alias** - 从密钥存储中删除别名。

例如，要读取别名：

```
/subsystem=elytron/key-store=httpsKS/:read-alias(alias=localhost)
```

1.11.7. 使用 client-ssl-context

当 JBoss EAP 实例创建作为客户端的 SSL 连接（如使用 SSL）时，使用 **client-ssl-context** 提供 SSL 上下文。

创建 **client-ssl-context** ：

1. 根据需要，创建 **key-store**、**key-manager** 和 **trust-manager** 组件。
   如果建立双向 SSL/TLS 连接，您需要为客户端和服务器证书分别创建单独的 **key-store** 组件，一个 **key-manager** 用于客户端 **key-store**，一个 **trust-manager** 用于服务器 **key-store**。或者，如果您要进行单向 SSL/TLS 连接，则需要为 **服务器证书和引用** 它的 **trust-manager** 创建密钥存储。使用 Elytron Subsystem 为应用程序启用双向 SSL/TLS 部分中包括了创建 keystores 和 truststores 的示例。
2. 创建 **client-ssl-context**。
   创建 **client-ssl-context**，引用密钥存储，truststores 以及任何其他必要的配置选项。

   示例： client-ssl-context

   ```
   /subsystem=elytron/client-ssl-context=exampleCSC:add(key-manager=clientKM, trust-manager=clientTM, protocols=["TLSv1.2"])
   ```
3. 引用 **client-ssl-context**。

有关 **client-ssl-context** 和其他 Elytron 组件的属性的完整列表，请参阅 Elytron subsystem 组件参考。

1.11.8. 使用 server-ssl-context

server-ssl-context 用于提供服务器端 SSL 上下文。除了通常的 SSL 上下文配置外，还可以配置额外的项目，如密码套件和协议。SSL 上下文会将配置的任何其他项包装。

1. 根据需要，创建 **key-store**、**key-manager** 和 **trust-manager** 组件。
   如果建立双向 SSL/TLS 连接，您需要为客户端和服务器证书分别创建单独的 **key-store** 组件，一个 **key-manager** 用于服务器 **key-store**，一个 **trust-manager** 用于服务器 **trust-store**。或者，如果您要进行单向 SSL/TLS 连接，则需要为 **服务器证书和引用** 它的 **key-manager** 创建密钥存储。使用 Elytron Subsystem 为应用程序启用双向 SSL/TLS 部分中包括了创建 keystores 和 truststores 的示例。

2. 创建 **server-ssl-context**。
   使用下面概述的选项之一，创建一个 **server-ssl-context**，引用密钥管理器、信任管理器或其他所需的配置选项。

使用管理 CLI 添加服务器 SSL 上下文

```
/subsystem=elytron/server-ssl-context=newServerSSLContext:add(key-
manager=KEY_MANAGER,protocols=["TLSv1.2"])
```

重要

您需要确定将支持哪些 HTTPS 协议。上面的示例命令使用 **TLSv1.2**。您可以使用 **cipher-suite-filter** 参数指定允许哪些密码套件，**使用-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 **security** 子系统行为不同，它默认为遵循客户端密码套件顺序。

使用管理控制台添加服务器 SSL 上下文

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。

2. 导航到 **Configuration → Subsystems → Security(Elytron) → Other Settings**，再点击 **View**。

3. 点击 **SSL → Server SSL 上下文**，并点击 **Add** 来配置新的服务器 SSL 上下文。

有关 **server-ssl-context** 和其他 Elytron 组件的属性的完整列表，请参阅 Elytron subsystem 组件参考。

1.11.9. 使用 server-ssl-sni-context

server-ssl-sni-context 用于提供服务器端的 SNI 匹配。它提供了匹配的规则，用于将主机名与 SSL 上下文关联，如果不提供主机名都不匹配，则为默认值。SSL SNI 上下文可以代替标准服务器 SSL 上下文，例如在 **undertow** 子系统中定义上下文时。

1. 根据需要，创建 **key-store**、**key-manager**、**trust-manager** 和 **server-ssl-context** 组件。必须定义一个服务器 SSL 上下文来创建 **server-ssl-sni-context**。

2. 创建一个 **server-ssl-sni-context**，为 **server-ssl-context** 元素提供匹配信息。必须使用 **default-ssl-context** 属性指定默认 SSL 上下文，如果没有找到匹配的主机名，则将使用该上下文。**host-context-map** 接受以逗号分隔的主机名列表，以匹配各种 SSL 上下文。

```
/subsystem=elytron/server-ssl-sni-context=SERVER_SSL_SNI_CONTEXT:add(default-ssl-context=DEFAULT_SERVER_SSL_CONTEXT,host-context-map={HOSTNAME=SERVER_SSL_CONTEXT,...})
```

以下内容将用于定义默认为 **serverSSL** SSL 上下文的 **server-ssl-sni-context**，并将 **www.example.com** 的传入请求与 **示例SSL** 上下文匹配。

```
/subsystem=elytron/server-ssl-sni-context=exampleSNIContext:add(default-ssl-context=serverSSL,host-context-map={www\\.example\\.com=exampleSSL})
```

注意

主机匹配的属性值用作正则表达式，因此请确保转义用于限定域名的任何句点(.)。

使用管理控制台配置 server-ssl-sni-context

1. 访问管理控制台。有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Configuration → Subsystems → Security(Elytron) → Other Settings**，再点击 **View**。
3. 点 **SSL → Server SSL SNI 上下文**配置所需的 **ssl-sni-context**。

有关 Elytron 组件的属性列表，请参阅 Elytron subsystem 组件参考。

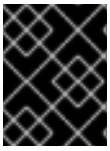
1.11.10. 自定义 SSL 组件

在 **elytron** 子系统中配置 SSL/TLS 时，您可以提供并使用以下组件的自定义实现：

- **key-store**
- **key-manager**
- **trust-manager**
- **client-ssl-context**
- **server-ssl-context**

警告

不建议为 **trust-manager** 以外的任何组件提供自定义实现，而无需了解 Java 安全套接字扩展(JSSE)。



重要

当使用 FIPS 时，无法使用自定义信任管理器或密钥管理器，因为 FIPS 需要这些管理器嵌入到 JDK 中。可以通过实施验证 X509 证据的 **SecurityRealm** 实现类似行为。

在创建 Elytron 组件的定制实施时，它们必须提供相应的功能和要求。有关功能和要求的详情，请查看 JBoss EAP *安全架构*指南中的功能与要求部分。每个组件的实施详情都由 JDK 供应商提供。

1.11.10.1. 在 Elytron 中添加自定义组件

以下步骤描述了在 Elytron 中添加自定义组件。

1. 将含有自定义组件提供程序的 JAR 作为模块添加到 JBoss EAP 中，声明任何需要的依赖项，如 **javax.api** ：

```
module add --name=MODULE_NAME --resources=FACTORY_JAR --
dependencies=javax.api,DEPENDENCY_LIST
```

重要

使用 **module** 管理 CLI 命令仅作为技术预览提供和删除模块。此命令不适用于在受管域中使用，或者在远程连接到管理 CLI 时使用。在生产环境中应手动添加模块并移除模块。如需更多信息，请参阅 JBoss EAP *配置指南*中的手动创建自定义模块和手动删除自定义模块部分。

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。

2. 当组件添加到 **elytron** 子系统时，**java.util.ServiceLoader** 将用于发现该提供程序。另外，可以通过定义 **provider-loader** 来提供对提供程序的引用。创建加载器的方法有两种，每个组件只能实施一个。

   - 在定义 **provider-loader** 时直接引用该提供程序：

```
/subsystem=elytron/provider-loader=LOADER_NAME:add(class-names=
[CLASS_NAME],module=MODULE_NAME)
```

   - 在 **META-INF/services/java.security.Provider** 中包含对提供程序的引用。在 **org.kohsuke.metainf-services** 中使用 **@MetaInfServices** 注解时，会自动创建此引用。当使用这个方法时，只有 **provider-loader** 需要使用该模块，如下所示：

```
/subsystem=elytron/provider-loader=LOADER_NAME:add(module=MODULE_NAME)
```

3. 将自定义组件添加到 Elytron 的配置中，使用适当的元素来添加和引用任何定义的提供程序。

```
/subsystem=elytron/COMPONENT_NAME=NEW_COMPONENT:add(providers=LOADER_N
AME,...)
```

例如，要定义信任管理器，需要使用 **trust-manager** 元素，如以下命令所示：

示例：添加自定义信任管理器

```
/subsystem=elytron/trust-
manager=newTrustManager:add(algorithm=MyX509,providers=customProvider,key-
store=sampleKeystore)
```

-

4. 定义之后，可以从其他元素引用该组件。

其他资源

- 如需更多信息，请参阅模块和依赖项。

1.11.10.2. 在自定义 Elytron 组件中包含参数

如果您的类实施 **初始化** 方法，您可以在自定义组件中包含参数，如下所示。

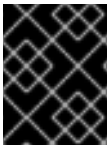
```
void initialize(final Map<String, String> configuration);
```

此方法允许自定义类在定义时接收一组配置字符串。它们在定义组件时使用 **configuration** 属性传递。例如，以下示例定义了名为 **myAttribute** 的属性，值设为 **myValue**。

```
/subsystem=elytron/COMPONENT_NAME=NEW_COMPONENT:add(class-
name=CLASS_NAME,module=MODULE_NAME,configuration={myAttribute="myValue"}
```

1.11.10.3. 使用带有 Elytron 的自定义信任管理器

通过实施自定义信任管理器，可以在 Undertow 中使用 HTTPS 时扩展证书验证、在 **dir-context** 中的 LDAPS 或将 Elytron 用于 SSL 连接的任何位置。此组件负责为服务器做出信任决策，强烈建议您在使用自定义信任管理器时实现这些实施。



重要

当使用 FIPS 时，无法使用自定义信任管理器，因为 FIPS 要求将此管理器嵌入到 JDK 中。可以通过实施验证 X509 证据的 **SecurityRealm** 实现类似行为。

实施自定义信任管理器的要求
使用自定义信任管理器时，必须实现以下内容：

- 实现 **X509ExtendedTrustManager** 接口的信任管理器。
- 扩展 **TrustManagerFactorySpi** 的信任管理器工厂。
- 信任管理器工厂的供应商。

此提供程序必须包含在 JAR 文件中，才能添加到 JBoss EAP 中。任何实施类都必须作为模块包含在 JBoss EAP 中。 不需要一个模块中的类，可以从模块依赖项加载。

实施示例
以下示例演示了一个提供程序，它把自定义信任管理器工厂注册为服务。

示例： Provider

```
import org.kohsuke.MetaInfServices;
import javax.net.ssl.TrustManagerFactory;
import java.security.Provider;
import java.util.Collections;
import java.util.List;
import java.util.Map;
```

```
@MetaInfServices(Provider.class)
public class CustomProvider extends Provider {

    public CustomProvider() {
        super("CustomProvider", 1.0, "Demo provider");

        System.out.println("CustomProvider initialization.");

        final List<String> emptyList = Collections.emptyList();
        final Map<String, String> emptyMap = Collections.emptyMap();

        putService(new Service(this, TrustManagerFactory.class.getSimpleName(),"CustomAlgorithm",
CustomTrustManagerFactorySpi.class.getName(), emptyList, emptyMap));
    }

}
```

以下示例演示了一个自定义信任管理器。这个信任管理器包含用于检查客户端或服务器是否信任的方法。

示例： TrustManager

```
import javax.net.ssl.SSLEngine;
import javax.net.ssl.X509ExtendedTrustManager;
import java.net.Socket;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

public class CustomTrustManager extends X509ExtendedTrustManager {

    public void checkClientTrusted(X509Certificate[] x509Certificates, String s, Socket socket) throws
CertificateException {
        // Insert your code here
    }

    public void checkServerTrusted(X509Certificate[] x509Certificates, String s, Socket socket) throws
CertificateException {
        // Insert your code here
    }

    public void checkClientTrusted(X509Certificate[] x509Certificates, String s, SSLEngine sslEngine)
throws CertificateException {
        // Insert your code here
    }

    public void checkServerTrusted(X509Certificate[] x509Certificates, String s, SSLEngine sslEngine)
throws CertificateException {
        // Insert your code here
    }

    public void checkClientTrusted(X509Certificate[] x509Certificates, String s) throws
CertificateException {
        // Insert your code here
    }

    public void checkServerTrusted(X509Certificate[] x509Certificates, String s) throws
CertificateException {
```

```
        // Insert your code here
    }

    public X509Certificate[] getAcceptedIssuers() {
        // Insert your code here
    }

}
```

以下示例用于返回信任管理器的实例。

示例：TrustManager factorySpi

```
import javax.net.ssl.ManagerFactoryParameters;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactorySpi;
import java.security.InvalidAlgorithmParameterException;
import java.security.KeyStore;
import java.security.KeyStoreException;

public class CustomTrustManagerFactorySpi extends TrustManagerFactorySpi {

    protected void engineInit(KeyStore keyStore) throws KeyStoreException {
        // Insert your code here
    }

    protected void engineInit(ManagerFactoryParameters managerFactoryParameters) throws
InvalidAlgorithmParameterException {
        // Insert your code here
    }

    protected CustomTrustManager[] engineGetTrustManagers() {
        // Insert your code here
    }

}
```

添加自定义信任管理器
创建提供商和信任管理器后，通过使用将自定义 组件添加到 Elytron 中的步骤将它们添加到 **elytron** 子系统中。

1.11.11. 默认 SSLContext

部署中使用的许多库可能需要 SSL 配置进行它们建立的连接。这些库通常由调用器进行配置。如果没有提供配置，则会对进程使用默认 **SSLContext**。

默认 **SSLContext** 有以下方法调用可用：

```
javax.net.ssl.SSLContext.getDefault();
```

默认情况下，这个 **SSLContext** 使用系统属性进行配置。但是，在 **elytron** 子系统中，可以指定哪些配置的上下文应关联并用作默认值。

要使用这个功能，请正常配置您的 **SSLContext**。然后，可使用下列命令来指定应将哪些 **SSLContext** 用作默认值。

```
/subsystem=elytron:write-attribute(name=default-ssl-context, value=client-context)
```

当现有服务和部署可以在设置前缓存默认 **SSLContext** 时，需要重新加载以确保在激活部署前设置了默认 SSLContext。

```
:reload
```

如果 **default-ssl-context** 属性变为 **未定义**，则标准 API 不会提供任何机制来恢复默认值。在这种情况下，需要重启 Java 进程。

```
/subsystem=elytron:undefine-attribute(name=default-ssl-context)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-restart" => true,
        "process-state" => "restart-required"
    }
}
```

1.11.12. 使用证书撤销列表

如果要根据证书撤销列表(CRL)验证证书，您可以使用 **elytron** 子系统中的信任管理器的 **certificate-revocation-list** 属性来配置它。例如：

```
/subsystem=elytron/trust-manager=TRUST_MANAGER:write-attribute(name=certificate-revocation-list,value={path=/path/to/CRL_FILE.crl.pem}
```

有关信任管理器可用属性的更多信息，请参阅 **trust-manager 属性表**。

注意

您的信任存储必须包含证书链，以便检查证书撤销列表和证书的有效性。信任存储不应该包含终止证书，只包含证书颁发机构和中间证书。

您可以使用 **reload-certificate-revocation-list** 操作指示信任管理器重新载入证书撤销列表。

```
/subsystem=elytron/trust-manager=TRUST_MANAGER:reload-certificate-revocation-list
```

1.11.13. 使用证书颁发机构管理签名证书

您可以使用 JBoss EAP 管理 CLI 和管理控制台获取和管理签名证书。这样，您可以直接从 CLI 或控制台创建签名证书，然后将其导入到所需的密钥存储中。

注意

本节中的许多命令都有一个可选的 **staging** 参数，指明是否应使用证书颁发机构的暂存 URL。这个值默认为 **false**，旨在协助测试。在生产环境中不应该启用此参数。

配置 Let 的加密帐户

自 JBoss EAP 7.4 起，Let 的加密是唯一支持的证书颁发机构。要管理签名证书，该帐户必须使用证书颁发机构创建，并提供以下信息：

- 包含证书颁发机构帐户密钥的别名的密钥存储。
- 证书颁发机构的别名。如果给定密钥存储中没有提供的别名，则会创建一个，并作为私钥条目存储。
- 可选 URL 列表，如电子邮件地址，证书颁发机构可以在任何问题时联系。

```
/subsystem=elytron/certificate-authority-account=CERTIFICATE_ACCOUNT:add(key-store=KEYSTORE,alias=ALIAS,contact-urls=[mailto:EMAIL_ADDRESS])
```

使用证书颁发机构创建帐户
配置了帐户后，可以通过同意其服务条款，使用证书颁发机构创建该帐户。

```
/subsystem=elytron/certificate-authority-account=CERTIFICATE_ACCOUNT:create-account(agree-to-terms-of-service=true)
```

使用证书颁发机构更新帐户
证书颁发机构帐户选项可以使用 **update-account** 命令更新。

```
/subsystem=elytron/certificate-authority-account=CERTIFICATE_ACCOUNT:update-account(agree-to-terms-of-service=true)
```

更改与认证机构关联的帐户密钥
可以使用 **change-account-key** 命令更改与证书颁发机构帐户关联的密钥。

```
/subsystem=elytron/certificate-authority-account=CERTIFICATE_ACCOUNT:change-account-key()
```

使用证书颁发机构取消激活帐户
如果不再需要帐户，则使用 **deactivate-account** 命令取消激活该帐户。

```
/subsystem=elytron/certificate-authority-account=CERTIFICATE_ACCOUNT:deactivate-account()
```

获取与认证机构关联的元数据
帐户的元数据可通过 **get-metadata** 命令查询。这提供了以下信息：

- 服务条款的 URL。
- 证书颁发机构网站的 URL。
- 证书颁发机构帐户列表。
- 是否需要外部帐户。

```
/subsystem=elytron/certificate-authority-account=CERTIFICATE_ACCOUNT:get-metadata()
```

使用管理控制台配置 Let 的加密帐户
使用管理控制台配置 Let 的 Encrypt 帐户：

1. 访问管理控制台。
   有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Runtime** → Host → **Security(Elytron)** → **SSL** 并点 **View**。
3. 点 **Certificate Auth...** 打开 **证书颁发机构帐户** 页面。

4. 您可以点带有标签的按钮来为所选别名执行以下配置：

   - **创建**
     使用证书颁发机构创建帐户。
   - **取消激活**
     取消激活所选证书颁发机构帐户。
   - **Update（更新）**
     使用证书颁发机构更新所选帐户。
   - **获取元数据**
     查看有关证书颁发机构帐户的以下信息：
     - 关联的别名
     - 证书颁发机构名称
     - 联系详情
     - 密钥存储名称
     - 证书颁发机构详情
   - **更改帐户密钥**
     - 使用证书颁发机构更改关联的密钥。

1.11.14. Keystore Manipulation Operations

您可以使用管理 CLI 和管理控制台在 Elytron **key-store** 资源上执行各种密钥存储操作。

使用管理 CLI 密钥存储操作操作
通过使用管理 CLI，可以执行以下密钥存储操作：

- 生成密钥对。
  generate-key-pair 命令生成密钥对，并将生成的公钥嵌套在自签名 X.509 证书中。生成的私钥和自签名证书将添加到密钥存储中。

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

/subsystem=elytron/key-store=httpsKS:generate-key-pair(alias=example,algorithm=RSA,key-size=1024,validity=365,credential-reference={clear-text=secret},distinguished-name="CN=www.example.com")
```

- 生成证书签名请求。
  generate-certificate-signing-request 命令使用密钥存储中的 **PrivateKeyEntry** 生成 PKCS #10 证书签名请求。生成的证书签名请求将写入到文件中。

```
/subsystem=elytron/key-store=httpsKS:generate-certificate-signing-request(alias=example,path=server.csr,relative-to=jboss.server.config.dir,distinguished-name="CN=www.example.com",extensions=[{critical=false,name=KeyUsage,value=digitalSignature}],credential-reference={clear-text=secret})
```

- 导入证书或证书链。
  import-certificate 命令将文件中的证书或证书链导入到密钥存储中的条目中。

```
/subsystem=elytron/key-store=httpsKS:import-
certificate(alias=example,path=/path/to/certificate_or_chain/file,relative-
to=jboss.server.config.dir,credential-reference={clear-text=secret},trust-cacerts=true)
```

- 导出证书。
  export-certificate 命令将证书从密钥存储中的条目导出到文件。

```
/subsystem=elytron/key-store=httpsKS:export-
certificate(alias=example,path=serverCert.cer,relative-to=jboss.server.config.dir,pem=true)
```

- 更改别名。
  change-alias 命令将现有的密钥存储条目移到新的别名。

```
/subsystem=elytron/key-store=httpsKS:change-alias(alias=example,new-
alias=newExample,credential-reference={clear-text=secret})
```

- 存储对密钥存储所做的更改。
  store 命令保留对备份密钥存储的文件所做的任何更改。

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

使用管理控制台管理密钥操作操作
使用管理控制台执行操作：

1. 访问管理控制台。
   有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Runtime → Security(Elytron) → Stores**，然后点击 **View**。
3. 点 **Key Store** 以打开密钥存储定义页面。
4. 点所需的密钥存储名称。
   您可以点击带有标签的按钮来为所选密钥存储执行以下操作：
   - **Load**
     加载或重新加载密钥存储。
   - **Store**
     对提供支持密钥存储的文件进行永久性更改。
   - **生成密钥对**
     生成密钥对，将公钥包装到自签名 X.509 证书，并将私钥和证书添加到密钥存储中。
   - **导入证书**
     将证书链从文件导入到密钥存储。
   - **获取**
     从证书颁发机构获取签名证书，并将它存储在密钥存储中。

1.11.14.1. 密钥存储证书颁发机构操作

在配置 Let 的加密帐户后，您可以在密钥存储上执行以下操作。

注意

本节中的许多命令都有一个可选的 **staging** 参数，指明是否应使用证书颁发机构的暂存 URL。这个值默认为 **false**，旨在协助测试。在生产环境中不应该启用此参数。

使用管理 CLI 密钥存储证书颁发机构操作
通过使用管理 CLI，可以执行以下密钥存储证书颁发机构操作：

- 获取签名证书。
  为密钥存储定义了证书颁发机构帐户后，您可以使用 **get-certificate** 命令获取签名证书并将其存储在密钥存储中。如果具有证书颁发机构的帐户不存在，则它会被自动创建。

```
/subsystem=elytron/key-store=KEYSTORE:obtain-certificate(alias=ALIAS,domain-names=[DOMAIN_NAME],certificate-authority-account=CERTIFICATE_ACCOUNT,agree-to-terms-of-service=true,algorithm=RSA,credential-reference={clear-text=secret})
```

- 撤销签名证书。
  revoke-certificate 命令撤销证书颁发机构发布的证书。

```
/subsystem=elytron/key-store=KEYSTORE:revoke-certificate(alias=ALIAS,certificate-authority-account=CERTIFICATE_ACCOUNT)
```

- 检查签名证书是否到期了续订。
  should-renew-certificate 命令确定证书是否在到期需要续订。如果证书过期的时间少于指定天数，则命令返回为 **true**，否则为 **false**。

  以下命令可确定证书是否在接下来的 7 天后过期。

```
/subsystem=elytron/key-store=KEYSTORE:should-renew-certificate(alias=ALIAS,expiration=7)
```

使用管理控制台密钥存储证书颁发机构操作
使用管理控制台执行操作：

1. 访问管理控制台。
   有关更多信息，请参阅 JBoss EAP *配置指南*中的管理控制台部分。
2. 导航到 **Runtime → Security(Elytron) → Stores**，然后点击 **View**。
3. 点 **Key Store** 以打开密钥存储定义页面。
4. 点所需密钥存储名称旁边的 **Aliases**。
5. 点所需别名。
   您可以点带有标签的按钮来为所选别名执行以下操作：
   - **更改别名**
     更改条目的别名。
   - **导出证书**
     将密钥存储条目中的证书导出到文件。

- **生成 CSR**
  生成证书签名请求。
- **删除别名**
  从密钥存储中删除所选别名。
- **详情**
  查看与别名关联的证书详情。
- **Revoke**
  撤销与别名关联的证书。
- **验证续订**
  确定关联的证书是到期续订。

1.11.15. 使用 Subject Alternative Name Extension 为 X.509 证书配置 Evidence Decoder

默认情况下，与 Elytron 中 X.509 证书关联的主体是证书中的主题名称，与 X.509 证书链相关联的主体是证书链中第一个证书的主题名称。您可以配置 **X509-subject-alt-name-evidence-decoder** 在 X.509 证书中使用主题备用名称扩展作为主体。

X.509 证书和 X.509 证书链的主题备用名称扩展规格在 RFC 5280 中定义。

先决条件

- 您知道客户端证书的预期格式，或者有客户端证书在本地可用。

流程

1. 确定要使用的主题备用名称扩展。
   如果您的客户端证书在本地，可以使用 **keytool** 命令查看主题备用名称扩展：

```
keytool -printcert -file /path/to/certificate/certificate.cert
```

   subject alternative name 扩展列为：

```
SubjectAlternativeName [
  DNS:one.example.org
  IP Address:127.0.0.1
]
```

2. 创建一个 **x509-subject-alt-name-evidence-decoder** 以使用指定的主题备用名称：

```
/subsystem=elytron/x509-subject-alt-name-evidence-decoder=exampleDnsDecoder:add(alt-name-type=__EXTENSION_TO_USE__)
```

   - 要使用证据解码器，请在 security-domain 中引用它：

```
/subsystem=elytron/security-domain=__Security_Domain_Name__:write-attribute(name="evidence-decoder",value="exampleDnsDecoder")
```

其他资源

- x509-subject-alternative-name-evidence-decoder Attributes

1.11.16. 配置 Aggregate Evidence Decoder

您可以配置聚合的证据解码器来合并两个或更多个解码器。证据解码器按配置的顺序应用，直到有证据解码器返回非null 主体，或直到没有更多数量的解码器才可以尝试。

先决条件

- 对要聚合的解码器进行配置。
  有关配置证据解码器的详情，请参考使用 Subject Alternative Name Extension 为 X.509 证书配置 Evidence Decoder 。

流程

1. 从现有解码器创建聚合的解码器：

```
/subsystem=elytron/aggregate-evidence-decoder=aggregateDecoder:add(evidence-decoders=[__DECODER_1__,__DECODER_2__,...,__DECODER_N__])
```

   - 要使用证据解码器，请在安全域中引用它：

```
/subsystem=elytron/security-domain=__SECURITY_DOMAIN__:write-attribute(name="evidence-decoder",value="aggregateDecoder")
```

1.11.17. 配置 X.500 Subject Evidence Decoder

配置 **x500-subject-evidence-decoder**，以从证书链中第一个证书中提取主题。

流程

- 创建 x.500 主题的解码器：

```
/subsystem=elytron/x500-subject-evidence-decoder=exampleSubjectDecoder:add()
```

1.11.18. 使用自定义 Evidence Decoder 实现

您可以通过在 Elytron 中使用自定义 **org.wildfly.security.auth.server.EvidenceDecoder** 实现，方法是将其添加为 JBoss EAP 的模块。

流程

1. 将自定义实施类打包为 Java 归档(JAR)。
2. 添加模块到含有 JAR 的 JBoss EAP。
   有关添加模块到 JBoss EAP 的详情，请参考 *配置指南*中的创建自定义模块部分。
3. 将自定义验证解码器添加到 Elytron:

```
/subsystem=elytron/custom-evidence-decoder=myCustomEvidenceDecoder:add(module=__MODULE_NAME__, class-name=__FULLY_QUALIFIED_CLASS_NAME__)
```

第 2 章 保护托管域

您可以保护受管域控制器与其主机控制器之间的通信。

2.1. 使用 ELYTRON 为域控制器配置密码身份验证

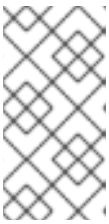
您需要将用户添加到主域控制器，以便从控制器能够以用户身份进行身份验证。从属控制器尝试在主域控制器的 HTTP 接口进行身份验证。

流程

1. 在 master 域控制器上添加用户。使用 **add-user** 实用程序添加用户名、密码和其他配置。如果 HTTP 接口通过 **ManagementRealm** Elytron 安全域进行保护，则必须添加用户到 **ManagementRealm**。

注意

如果您使用基于文件的默认用户和组身份验证机制，请运行 **EAP_HOME/bin/add-user.sh** 脚本。



注意

使用 **add-user** 实用程序添加用户信息后，服务器会将属性文件的内容缓存在内存中。但是，服务器会检查每个身份验证请求上属性文件修改的时间，并在更新时间时重新加载。这意味着 **add-user** 实用程序所做的所有更改都会立即应用到任何正在运行的服务器。

注意

管理用户的域的默认名称为 **ManagementRealm**。当 **add-user** 实用程序提示输入域名时，您必须接受默认域名；即，除非您切换到不同的域。

2. 在从控制器中添加一个**身份验证配置**。以下示例演示了使用用户 slave 和 **password1!** 来添加一个名为 **slave** 的新 **身份验证配置** ：

```
/host=slave/subsystem=elytron/authentication-configuration=slave:add(authentication-name=slave, credential-reference={clear-text=password1!})
```

3. 在从控制器中添加 **authentication-context**，如下例所示：

```
/host=slave/subsystem=elytron/authentication-context=slave-context:add(match-rules=[{authentication-configuration=slave}])
```

4. 指定从属控制器中的域控制器位置和 **authentication-context**，如下例所示：

```
<domain-controller>
  <remote protocol="remote" host="localhost" port="9990" authentication-context="slave-context"/>
</domain-controller>
```

其他资源

- 有关受管域工作模式的概念和常规配置的详情，请参考 JBoss EAP *配置指南*中的域管理 部分。
- 有关管理用户的信息，请参阅 JBoss EAP *配置指南*中的管理用户部分。

2.2. 使用旧的内核管理验证为域控制器配置密码身份验证

默认情况下，Red Hat JBoss Enterprise Application Platform 配置主域控制器，要求从连接到主域控制器的每个从属控制器进行身份验证。

使用适当的凭证配置从属控制器。

流程

1. 使用 **add-user** 脚本将用户添加到主域控制器。
   a. 检查用户是否已添加到同一个域中，master 用来保护其管理界面，默认是 **ManagementRealm**。
   b. 如以下示例所示，添加一个从（slave）用户。对于 *是否将这个新用户用于一个 AS 进程连接到另一个 AS 进程，请选择* ***yes*** *？*

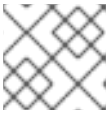
```
$ EAP_HOME/bin/add-user.sh

What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): a

Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : slave-user
Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.
 - The password should be different from the username
 - The password should not be one of the following restricted values {root, admin, administrator}
 - The password should contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
Password :
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[  ]:
About to add user 'slave-user' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'slave-user' to file '/home/user/EAP-7.4.0/standalone/configuration/mgmt-users.properties'
Added user 'slave-user' to file '/home/user/EAP-7.4.0/domain/configuration/mgmt-users.properties'
Added user 'slave-user' with groups  to file '/home/user/EAP-7.4.0/standalone/configuration/mgmt-groups.properties'
Added user 'slave-user' with groups  to file '/home/user/EAP-7.4.0/domain/configuration/mgmt-groups.properties'
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
```

```
yes/no? yes
To represent the user add the following to the server-identities definition <secret
value="ABCzc3dv11Qx" />
```

重要

添加用户后，脚本会输出一个 **<secret>** 元素。在下一步中，您需要使用此元素。

2. 配置从属控制器以使用该凭据。在主域控制器上创建用户后，您必须更新每个从属控制器，以便在主机配置文件中使用该凭证。例如，**host.xml** 或 **host-slave.xml**。
以下示例显示了在域控制器配置中的 < **remote&gt**; 元素中添加用户名。另外，示例显示了将 **<secret>** 添加到用于保护 **<remote>** 元素的域的 **server-identities** 中。



注意

用户名和 &lt ;*secret*& gt; 均通过在上一步中向 master 域控制器添加用户来获取。

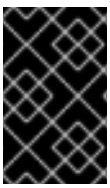
```
...
<security-realm name="ManagementRealm">
   <server-identities>
      <!-- Replace this with either a base64 password of your own, or use a vault with a vault
expression -->
      <secret value="ABCzc3dv11Qx"/>
   </server-identities>
...
<domain-controller>
  <remote security-realm="ManagementRealm" username="slave-user">
     <discovery-options>
        <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9990}"/>
     </discovery-options>
  </remote>
</domain-controller>
```

其他资源

- 有关受管域工作模式的概念和常规配置的详情，请参考 JBoss EAP *配置指南*中的域管理 部分。
- 有关管理用户的信息，请参阅 JBoss EAP *配置指南*中的管理用户部分。

2.3. 使用 ELYTRON 为域控制器配置 SSL 或 TLS

在受管域中配置 JBoss EAP 实例，以在 master 域控制器和主机控制器之间相互通信时，使用安全套接字层(SSL)或传输层(TLS)。



重要

当您将 SSL 或 TLS 配置为在受管域中 JBoss EAP 实例之间使用时，每个实例可以具有客户端或服务器角色，具体取决于交互。这包括所有主机控制器和域控制器。为获得最佳结果，请在端点之间设置双向 SSL 或 TLS。

先决条件

- 生成并配置了所有必要的证书和密钥存储。要为管理接口启用双向 SSL/TLS，请参阅使用安全命令启用双向 SSL/TLS，或使用 Elytron 子系统命令启用双向 SSL/TLS。

注意

要在端点之间设置双向 SSL 或 TLS，您需要为主域控制器和每一主机控制器生成和配置证书和密钥存储。

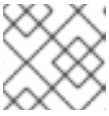
另外，您必须将 master 域控制器的证书导入到每个主机控制器密钥存储中。此外，将每个主机控制器证书导入到主域控制器密钥存储中。

流程

1. 在 master 域控制器上添加用户。如果您重新使用基于文件的用户和组身份验证机制，请运行 ***EAP_HOME*/bin/add-user.sh** 脚本。提示时，添加用户名、密码和其他配置。

注意

管理用户的域的默认名称为 **ManagementRealm**。当 **add-user** 实用程序提示输入域名时，您必须接受默认域名；即，除非您切换到不同的域。



注意

您必须在主域控制器上添加用户，以便从控制器进行身份验证。

注意

服务器将属性文件的内容缓存在内存中。但是，服务器会检查每个身份验证请求上属性文件修改的时间，并在更新时间时重新加载。因此，**add-user** 实用程序的任何更改都会立即应用到任何正在运行的服务器。

2. 将主域控制器配置为使用 SSL 或 TLS。以下示例显示了为服务器密钥存储和信任存储配置域控制器 **密钥**、**key-manager**、**trust-manager** 和 **server-ssl-context** 的命令。

```
/host=master/subsystem=elytron/key-store=twoWayKS:add(path=/path/to/server.keystore.jks,credential-reference={clear-text=secret},type=JKS)

/host=master/subsystem=elytron/key-store=twoWayTS:add(path=/path/to/server.truststore.jks,credential-reference={clear-text=secret},type=JKS)

/host=master/subsystem=elytron/key-manager=twoWayKM:add(key-store=twoWayKS,credential-reference={clear-text=secret})

/host=master/subsystem=elytron/trust-manager=twoWayTM:add(key-store=twoWayTS)

/host=master/subsystem=elytron/server-ssl-context=twoWaySSC:add(key-manager=twoWayKM,protocols=["TLSv1.2"],trust-manager=twoWayTM,want-client-auth=true,need-client-auth=true)
```

```
/host=master/core-service=management/management-interface=http-interface:write-attribute(name=ssl-context, value=twoWaySSC)
```

重要

红帽没有在前面的命令中指定 algorithm 属性，因为 Elytron 子系统使用 **KeyManagerFactory.getDefaultAlgorithm（）** 和 **TrustManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些关键管理器算法。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **SunX509** 作为密钥管理器算法属性，将 **PKIX** 指定为信任管理器算法属性。

此外，您需要确定您要支持的 HTTPS 协议。此流程中的示例使用 **TLSv1.2**。

您可以使用 **cipher-suite-filter** 指定密码套件，**使用-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 **security** 子系统行为不同，它默认为遵循客户端密码套件顺序。

3. 在每个从属主机控制器上配置身份验证上下文和域控制器位置。以下示例配置显示了 **localhost** 上存在的域控制器。

注意

您必须为您的环境指定正确的管理用户、密码和域控制器位置。

```
/host=slave1/subsystem=elytron/authentication-context=slaveHostSSLContext:add()

/host=slave1/subsystem=elytron/authentication-configuration=slaveHostSSLConfiguration:add()

/host=slave1/subsystem=elytron/authentication-configuration=slaveHostSSLConfiguration:write-attribute(name=sasl-mechanism-selector,value=DIGEST-MD5)

/host=slave1/subsystem=elytron/authentication-configuration=slaveHostSSLConfiguration:write-attribute(name=authentication-name,value=slave)

/host=slave1/subsystem=elytron/authentication-configuration=slaveHostSSLConfiguration:write-attribute(name=realm,value=ManagementRealm)

/host=slave1/subsystem=elytron/authentication-configuration=slaveHostSSLConfiguration:write-attribute(name=credential-reference,value={clear-text=password1!})

/host=slave1/subsystem=elytron/authentication-context=slaveHostSSLContext:write-attribute(name=match-rules,value=[{match-host=localhost,authentication-configuration=slaveHostSSLConfiguration}]
```

```
/host=slave1:write-remote-domain-
controller(host=localhost,port=9990,protocol=remote,authentication-
context=slaveHostSSLContext)
```

4. 将每个从属主机控制器配置为使用 SSL 或 TLS。以下命令为服务器密钥存储和信任存储配置从属主机控制器 **key-store**、**key-manager**、**trust-manager**、**client-ssl-context** 和 **authentication-context**。此外，示例中显示了一个在 **localhost** 上存在的域控制器。

注意

您必须为您的环境指定正确的域控制器位置。

```
/host=slave1/subsystem=elytron/key-
store=twoWayKS:add(path=/path/to/client.keystore.jks,credential-reference={clear-
text=secret},type=JKS)

/host=slave1/subsystem=elytron/key-
store=twoWayTS:add(path=/path/to/client.truststore.jks,credential-reference={clear-
text=secret},type=JKS)

/host=slave1/subsystem=elytron/key-manager=twoWayKM:add(key-
store=twoWayKS,credential-reference={clear-text=secret})

/host=slave1/subsystem=elytron/trust-manager=twoWayTM:add(key-store=twoWayTS)

/host=slave1/subsystem=elytron/client-ssl-context=twoWayCSC:add(key-
manager=twoWayKM,protocols=["TLSv1.2"],trust-manager=twoWayTM)

/host=slave1/subsystem=elytron/authentication-context=slaveHostSSLContext:write-
attribute(name=match-rules,value=[{match-host=localhost,authentication-
configuration=slaveHostSSLConfiguration,ssl-context=twoWayCSC}])
```

重要

红帽没有在前面的命令中指定 algorithm 属性，因为 Elytron 子系统使用 **KeyManagerFactory.getDefaultAlgorithm（）** 和 **TrustManagerFactory.getDefaultAlgorithm（）** 来确定算法。但是，您可以指定 algorithm 属性。要指定 algorithm 属性，您需要知道您使用的 JDK 提供了哪些关键管理器算法。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。

在上一命令中，您可以指定 **SunX509** 作为密钥管理器算法属性，将 **PKIX** 指定为信任管理器算法属性。

此外，您需要确定您要支持的 HTTPS 协议。此流程中的示例使用 **TLSv1.2**。

您可以使用 **cipher-suite-filter** 指定密码套件，**使用-cipher-suites-order** 参数来遵循服务器密码套件顺序。**use-cipher-suites-order** 属性默认设置为 **true**。这与旧的 **security** 子系统行为不同，它默认为遵循客户端密码套件顺序。

5. 重新加载您受管域中的所有 JBoss EAP 主机。

其他资源

- 有关受管域工作模式的概念和常规配置的详情，请参考 JBoss EAP *配置指南*中的域管理 部分。

- 有关管理用户的信息，请参阅 JBoss EAP *配置指南*中的管理用户部分。

2.4. 使用 ELYTRON 在域模式中配置 SSL

在 JBoss EAP 7.1 或更高版本中，您可以使用 Elytron 在域模式中配置 SSL。

先决条件

- JBoss EAP 7.1 或更高版本.
- Elytron

流程

1. 创建自签名证书以启用 SSL：

```
keytool -genkey -alias jboss -keysize 2048 -validity 365 -keyalg RSA -sigalg
SHA256withRSA -keystore jboss.jks -storepass jboss@123 -keypass jboss@123 -dname
"CN=example.com, OU=JavaEE, O=Red Hat, C=IN"
```

2. 使用管理 CLI 创建密钥存储、key-manager 和 ssl-context。

```
#Configure a keystore
/profile=<profile-name>/subsystem=elytron/key-
store=httpsKS:add(path="${jboss.home.dir}/ssl/jboss.jks", credential-reference={clear-
text=jboss@123}, type=JKS)

#Create a new key-manager
/profile=<profile-name>/subsystem=elytron/key-manager=httpsKM:add(key-
store=httpsKS,algorithm="SunX509",credential-reference={clear-text=jboss@123})

#Configure new server-ssl-context reference with protocol and ciphers
/profile=<profile-name>/subsystem=elytron/server-ssl-context=httpsSSC:add(key-
manager=httpsKM,protocols=["TLSv1.2"])
```

3. 配置 **undertow** 子系统，以映射 Elytron **ssl-context** ：

```
batch
/profile=<profile-name>/subsystem=undertow/server=default-server/https-
listener=https:undefine-attribute(name=security-realm)
/profile=<profile-name>/subsystem=undertow/server=default-server/https-
listener=https:write-attribute(name=ssl-context,value=httpsSSC)
run-batch
```

4. **可选：** 保护 **management-interface** 以使用同一 **ssl-context** ：
host-*.xml 文件定义域控制器和主机控制器的配置，该配置包含管理接口。要确保 SSL 配置成功，您必须在主机上再次定义 **ssl-context**。

```
#Configure a keystore on the master DC host
/host=<host-name>/subsystem=elytron/key-
store=httpsKS:add(path="${jboss.home.dir}/ssl/jboss.jks", credential-reference={clear-
text=jboss@123}, type=JKS)

#Create a new key-manager on the master DC host
```

```
/host=<host-name>/subsystem=elytron/key-manager=httpsKM:add(key-store=httpsKS,algorithm="SunX509",credential-reference={clear-text=jboss@123})

#Configure new server-ssl-context reference with protocol and ciphers on the master DC host
/host=<host-name>/subsystem=elytron/server-ssl-context=httpsSSC:add(key-manager=httpsKM,protocols=["TLSv1.2"])

#Configure the secure-port and ssl-context for management-http interface  on the master DC host
/host=<host-name>/core-service=management/management-interface=http-interface:write-attribute(name=ssl-context,value=httpsSSC)

/host=<host-name>/core-service=management/management-interface=http-interface:write-attribute(name=secure-port,value=9993)
```

5. 确保正确配置了信任存储，使远程主机控制器能够通过 SSL 连接到域控制器。如需更多信息，请参阅使用 Elytron 配置域和主机控制器之间的 SSL/TLS。

6. 重新载入服务器以确保更改有效：

```
reload --host=<host-name>
```

验证

- 使用浏览器或在 Red Hat Enterprise Linux 命令行中打开SSL 来验证 TLS 连接：

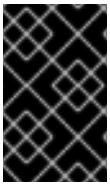
```
openssl s_client -connect host:8443
```

输出中显示有关证书和使用的 TLS 版本的信息。

```
SSL-Session:
Protocol: TLSv1.2
```

2.5. 为旧的核心管理验证机制配置 SSL 或 TLS

在受管域中配置 JBoss EAP 实例，以在 master 域控制器和主机控制器之间相互通信时，使用安全套接字层(SSL)或传输层(TLS)。

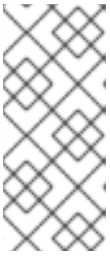


重要

当您将 SSL 或 TLS 配置为在受管域中 JBoss EAP 实例之间使用时，每个实例可以具有客户端或服务器角色，具体取决于交互。这包括所有主机控制器和域控制器。为获得最佳结果，请在端点之间设置双向 SSL 或 TLS。

先决条件

- 生成并配置了所有必要的证书和密钥存储。要为管理接口启用双向 SSL/TLS，请参阅使用旧 核心管理身份验证为管理接口设置双向 SSL/TLS。



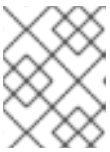
注意

要在端点之间设置双向 SSL 或 TLS，您需要为主域控制器和每一主机控制器生成和配置证书和密钥存储。

另外，您必须将 master 域控制器的证书导入到每个主机控制器密钥存储中。此外，将每个主机控制器证书导入到主域控制器密钥存储中。

流程

1. 将主域控制器配置为使用 SSL 或 TLS，如以下示例中所示。当您配置了所有证书和密钥存储后，您需要配置安全域以使用双向 SSL/TLS。您可以通过将安全域配置为使用 SSL/TLS 来达到此目的。配置的安全域将保护用于在主机控制器和主域控制器之间连接的管理接口。



注意

在批处理模式或服务器上运行以下命令。在添加 *ssl* 服务器身份后，您必须重新加载服务器。此流程中的示例以批处理模式运行。

```
batch

/host=master/core-service=management/security-realm=CertificateRealm:add()

/host=master/core-service=management/security-realm=CertificateRealm/server-identity=ssl:add(alias=domaincontroller,keystore-relative-to=jboss.domain.config.dir,keystore-path=domaincontroller.jks,keystore-password=secret)

/host=master/core-service=management/security-realm=CertificateRealm/authentication=truststore:add(keystore-relative-to=jboss.domain.config.dir,keystore-path=domaincontroller.jks,keystore-password=secret)

/host=master/core-service=management/security-realm=CertificateRealm/authentication=local:add(default-user=\$local)

/host=master/core-service=management/security-realm=CertificateRealm/authentication=properties:add(relative-to=jboss.domain.config.dir,path=mgmt-users.properties)

/host=master/core-service=management/management-interface=http-interface:write-attribute(name=security-realm,value=CertificateRealm)

run-batch
```

2. 将所有主机控制器配置为使用 SSL 或 TLS，如下例所示：

```
batch

/host=instance1/core-service=management/security-realm=CertificateRealm:add()

/host=instance1/core-service=management/security-realm=CertificateRealm/server-identity=ssl:add(alias=instance1,keystore-relative-to=jboss.domain.config.dir,keystore-path=instance1.jks,keystore-password=secret)

/host=instance1/core-service=management/security-realm=CertificateRealm/authentication=truststore:add(keystore-relative-
```

```
to=jboss.domain.config.dir,keystore-path=instance1.jks,keystore-password=secret)

/host=instance1/core-service=management/security-
realm=CertificateRealm/authentication=local:add(default-user="\$local")

/host=instance1/core-service=management/security-
realm=CertificateRealm/authentication=properties:add(relative-
to=jboss.domain.config.dir,path=mgmt-users.properties)

/host=instance1/core-service=management/management-interface=http-interface:write-
attribute(name=security-realm,value=CertificateRealm)

run-batch
```

3. 更新连接主域控制器时使用的安全域。当服务器没有运行时，您必须对主机控制器配置文件进行这个更新。例如，**host.xml** 或 **host-slave.xml**。

```
<domain-controller>
  <remote security-realm="CertificateRealm" username="slave-user">
    <discovery-options>
      <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9990}"/>
    </discovery-options>
  </remote>
</domain-controller>
```

其他资源

- 有关受管域工作模式的概念和常规配置的详情，请参考 JBoss EAP *配置指南*中的域管理 部分。
- 有关设置双向 SSL 或 TLS 的过程的详情，请参考 *配置服务器安全* 指南中的使用 传统内核管理 身份验证设置双向 SSL/TLS 的信息。

第 3 章 保护服务器的用户及其管理界面

3.1. 使用 ELYTRON 进行用户身份验证

3.1.1. 默认配置

默认情况下，JBoss EAP 管理接口通过传统的核心管理身份验证进行保护。

示例：默认配置

```
/core-service=management/management-interface=http-interface:read-resource()
{
    "outcome" => "success",
    "result" => {
        "allowed-origins" => undefined,
        "console-enabled" => true,
        "http-authentication-factory" => undefined,
        "http-upgrade" => {"enabled" => true},
        "http-upgrade-enabled" => true,
        "sasl-protocol" => "remote",
        "secure-socket-binding" => undefined,
        "security-realm" => "ManagementRealm",
        "server-name" => undefined,
        "socket-binding" => "management-http",
        "ssl-context" => undefined
    }
```

JBoss EAP 在 **elytron** 子系统中提供 **management-http-authentication** 和 **management-sasl-authentication**，以保护管理接口。

更新 JBoss EAP 以使用默认 Elytron 组件：

1. 设置 **http-authentication-factory** 以使用 **management-http-authentication** ：

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

2. 设置 **sasl-authentication-factory** 以使用 **management-sasl-authentication** ：

```
/core-service=management/management-interface=http-interface:write-attribute(name=http-upgrade.sasl-authentication-factory, value=management-sasl-authentication)
```

3. 取消定义 **security-realm** ：

```
/core-service=management/management-interface=http-interface:undefine-attribute(name=security-realm)
```

4. 重新加载 JBoss EAP 以使更改生效：

```
reload
```

管理接口现在使用 **elytron** 子系统提供的默认组件进行保护。

3.1.1.1. 默认 Elytron HTTP 验证配置

当您通过 http 访问管理界面时，例如在使用基于 Web 的管理控制台时，JBoss EAP 将使用 **management-http-authentication** http-authentication-factory。

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

management-http-authentication http-authentication-factory 配置为使用 **ManagementDomain** 安全域。

```
/subsystem=elytron/security-domain=ManagementDomain:read-resource()
{
    "outcome" => "success",
    "result" => {
        "default-realm" => "ManagementRealm",
        "permission-mapper" => "default-permission-mapper",
        "post-realm-principal-transformer" => undefined,
        "pre-realm-principal-transformer" => undefined,
        "principal-decoder" => undefined,
        "realm-mapper" => undefined,
        "realms" => [
            {
                "realm" => "ManagementRealm",
                "role-decoder" => "groups-to-roles"
            },
            {
                "realm" => "local",
                "role-mapper" => "super-user-mapper"
            }
        ],
        "role-mapper" => undefined,
        "trusted-security-domains" => undefined
    }
}
```

ManagementDomain 安全域由 **ManagementRealm** Elytron 安全域支持，它是基于属性的域。

重要

基于属性的域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）都要求重新加载服务器。此重新加载是通过从管理 CLI 运行 **reload** 命令来完成的。

```
reload
```

```
/subsystem=elytron/properties-realm=ManagementRealm:read-resource()
{
    "outcome" => "success",
    "result" => {
        "groups-attribute" => "groups",
        "groups-properties" => {
            "path" => "mgmt-groups.properties",
            "relative-to" => "jboss.server.config.dir"
        },
        "plain-text" => false,
        "users-properties" => {
            "path" => "mgmt-users.properties",
            "relative-to" => "jboss.server.config.dir"
        }
    }
}
```

3.1.1.2. 默认 Elytron 管理 CLI 身份验证

默认情况下，管理 CLI(**jboss-cli.sh**)配置为通过 **remote+http** 进行连接。

示例：默认 jboss-cli.xml

```
<jboss-cli xmlns="urn:jboss:cli:3.1">

    <default-protocol use-legacy-override="true">remote+http</default-protocol>

    <!-- The default controller to connect to when 'connect' command is executed w/o arguments -->
    <default-controller>
        <protocol>remote+http</protocol>
        <host>localhost</host>
        <port>9990</port>
    </default-controller>
```

这将通过 HTTP 建立连接，并使用 HTTP 升级将通信协议更改为 **Remoting**。HTTP 升级连接在 **http-interface** 的 **http-upgrade** 部分中使用 **sasl-authentication-factory** 进行了保护。

示例：使用默认组件配置

```
/core-service=management/management-interface=http-interface:read-resource()
{
    "outcome" => "success",
    "result" => {
        "allowed-origins" => undefined,
        "console-enabled" => true,
```

```
        "http-authentication-factory" => "management-http-authentication",
        "http-upgrade" => {
            "enabled" => true,
            "sasl-authentication-factory" => "management-sasl-authentication"
        },
        "http-upgrade-enabled" => true,
        "sasl-protocol" => "remote",
        "secure-socket-binding" => undefined,
        "security-realm" => undefined,
        "server-name" => undefined,
        "socket-binding" => "management-http",
        "ssl-context" => undefined
    }
}
```

默认的 sasl-authentication-factory 是 **management-sasl-authentication**。

```
/subsystem=elytron/sasl-authentication-factory=management-sasl-authentication:read-resource()
{
    "outcome" => "success",
    "result" => {
        "mechanism-configurations" => [
            {
                "mechanism-name" => "JBOSS-LOCAL-USER",
                "realm-mapper" => "local"
            },
            {
                "mechanism-name" => "DIGEST-MD5",
                "mechanism-realm-configurations" => [{"realm-name" => "ManagementRealm"}]
            }
        ],
        "sasl-server-factory" => "configured",
        "security-domain" => "ManagementDomain"
    }
}
```

management-sasl-authentication sasl-authentication-factory 指定 **JBOSS-LOCAL-USER** 和 **DIGEST-MD5** 机制。

ManagementRealm Elytron 安全域（在 **DIGEST-MD5** 中使用的域）与 **management-http-authentication** http-authentication-factory 中使用的 realm 相同。

示例：JBOSS-LOCAL-USER Realm

```
/subsystem=elytron/identity-realm=local:read-resource()
{
    "outcome" => "success",
    "result" => {
        "attribute-name" => undefined,
        "attribute-values" => undefined,
        "identity" => "$local"
    }
}
```

本地 Elytron 安全域用于处理本地用户的静默身份验证。

3.1.2. 使用新身份存储保护管理接口

1. 为您的身份存储创建安全域以及任何支持的安全域、解码器或映射程序。
   JBoss EAP 的 *如何配置身份管理指南*中的 Elytron subsystem 部分中介绍了此过程。例如，如果要使用基于文件系统的身份存储保护管理接口，则需要遵循使用基于文件系统的身份存储 配置身份验证 中的步骤。

2. 创建 **http-authentication-factory** 或 **sasl-authentication-factory**。

   示例：http-authentication-factory

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

   示例：sasl-authentication-factory

   ```
   /subsystem=elytron/sasl-authentication-factory=example-sasl-auth:add(sasl-server-factory=configured, security-domain=exampleSD, mechanism-configurations=[{mechanism-name=DIGEST-MD5, mechanism-realm-configurations=[{realm-name=exampleManagementRealm}]}])
   ```

3. 将 pattern-filter 添加到 **配置的 configurable-sasl-server-factory** 中。

   示例：在 Configured configurable-sasl-server-factory 中添加 GSSAPI

   ```
   /subsystem=elytron/configurable-sasl-server-factory=configured:list-add(name=filters, value={pattern-filter=GSSAPI})
   ```

   这是可选步骤。当客户端尝试连接到 HTTP 管理接口时，JBoss EAP 会向 HTTP 响应发送一个状态代码 **401 Unauthorized**，以及一组列出支持的验证机制（如 Digest、GSSAPI 等）的标头。如需更多信息，请参阅 JBoss EAP *安全架构指南*中的本地和远程客户端身份验证章节。

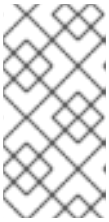
4. 更新管理接口以使用 **http-authentication-factory** 或 **sasl-authentication-factory**。

   示例：更新 http-authentication-factory

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

   reload
   ```

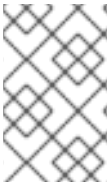
   示例：更新 sasl-authentication-factory

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

   reload
   ```



注意

在使用旧的核心管理身份验证时，您只能使用单一传统安全域来保护 http 管理接口。这会强制 HTTP 和 SASL 配置出现在单一的旧安全域中。使用 **elytron** 子系统时，您可以单独配置 **http-authentication-factory** 和 **sasl-authentication-factory**，从而使用不同的安全域来保护 http 管理界面的 HTTP 和 SASL 机制。



注意

如果在传统安全性和 Elytron 中分别使用相似实施的两个不同属性，则在管理界面中仅配置 Elytron 相关配置。例如，如果对旧安全和 Elytron 的 **security-realm** 进行了配置，则由 **http-authentication-factory 配置来处理身份验证**。

注意

当管理界面同时包含 **http-authentication-factory** 或 **sasl-authentication-factory**（用于 HTTP 接口）以及 **security-realm**，并且 **ssl-context** 属性没有使用 ssl-context 属性时，身份验证由 Elytron 处理，并且 SSL 由旧的安全域处理。

当管理界面同时包含 **security-realm** 和 **ssl-context** 时，以及 **http-authentication-factory** 或 **sasl-authentication-factory** 用于 HTTP 接口时，不使用 HTTP 接口的身份验证时，由旧安全域和 SSL 处理。

3.1.3. 添加 Silent 身份验证

默认情况下，JBoss EAP 提供本地用户的验证机制，也 **通过本地** 安全域来作为静默身份验证。更多信息请参阅 Silent 身份验证 部分。

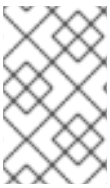
静默身份验证必须添加到 **sasl-authentication-factory** 中。

在现有的 **sasl-authentication-factory** 中添加静默身份验证：

```
/subsystem=elytron/sasl-authentication-factory=example-sasl-auth:list-add(name=mechanism-configurations, value={mechanism-name=JBOSS-LOCAL-USER, realm-mapper=local})

reload
```

使用 silent 身份验证创建新 **sasl-server-factory** ：

```
/subsystem=elytron/sasl-authentication-factory=example-sasl-auth:add(sasl-server-factory=configured,security-domain=ManagementDomain,mechanism-configurations=[{mechanism-name=DIGEST-MD5,mechanism-realm-configurations=[{realm-name=exampleManagementRealm}]},{mechanism-name=JBOSS-LOCAL-USER, realm-mapper=local}])

reload
```



注意

上面的示例使用现有的 **ManagementDomain** 安全域，但您也可以创建和使用其他安全域。您可以在 JBoss EAP *如何配置身份管理* 指南的 Elytron subsystem 部分中找到更多创建安全域的示例。

重要

如果使用了 Elytron 安全性，并且身份验证尝试尝试使用 **JBOSS-LOCAL-USER SASL** 机制及与实际身份不匹配的身份验证名称，身份验证会失败。

在传统的 **security** 子系统中，可以为 **JBOSS-LOCAL-USER** 选择自定义用户名。身份验证通过将用户名映射到 *特殊身份来继续*。

3.1.4. Authenticated Management 用户映射身份

使用 **elytron** 子系统保护管理接口时，您可以为通过身份验证用户身份映射提供安全域。这允许经过身份验证的用户在登录到管理界面时显示适当的身份。

应用程序服务器会公开多个管理界面。每种接口都可以与 **独立的身份验证关联**，以处理该接口的身份验证要求。

为做出授权决策，当前安全身份从安全域获得。返回的安全身份根据该安全域中定义的规则映射和权限分配。

注意

在大多数情况下，通用安全域用于所有管理；为了对管理接口进行身份验证，以及获取用于授权决策的安全身份。在这些情况下，安全域与管理接口的身份验证工厂相关联，并且不需要定义任何特殊的 **access=identity**。

在某些情况下，会使用不同的安全域来获取授权决策的身份。此处定义了 **access=identity** 资源。它包含对安全域的引用，以获取授权的身份。

以下示例假定您已使用 **exampleSD** Elytron 安全域保护了管理接口，并将其公开为 **exampleManagementRealm**。

要定义身份映射，请将 **身份资源** 添加到管理界面。

示例：添加 身份资源

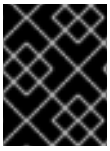
```
/core-service=management/access=identity:add(security-domain=exampleSD)
```

添加 **身份资源** 后，访问管理界面时将会出现经过身份验证的用户的身份。如果没有添加 **identity** 资源，则使用用于身份验证的安全域的身份。

例如，如果您以 **user1** 身份登录管理 CLI，您的身份将正确显示。

示例：从管理 CLI 显示 Authenticated User 的身份

```
:whoami
{
    "outcome" => "success",
    "result" => {"identity" => {"username" => "user1"}}
}
```



重要

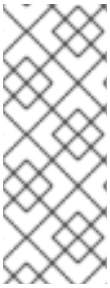
如果添加了 **identity** 资源，并且使用旧安全域来保护管理界面，经过身份验证的用户将始终具有 **匿名** 身份。删除 **identity** 资源后，通过旧安全域验证的用户会出现适当的身份。

管理操作的授权始终使用安全域，它是 **access=identity** 中指定的域。如果未指定，则这是用于身份验证的域。任何角色映射始终位于安全域的上下文中。

当前请求的 **identity** 资源将使用 Elytron 配置返回一组角色，作为映射。当使用基于 RBAC 的角色映射定义时，**identity** 资源的角色将作为组并放入管理 **角色映射** 以获取当前请求的管理角色。

表 3.1. 用于不同场景的身份验证

场景	没有 access=identity 定义	access=identity 引用 Elytron **security-domain**
使用旧的 **security-realm**的 HTTP 管理接口	来自连接的身份。	不支持或匿名身份。
使用由 **security-domain**支持的 **elytron** HTTP 身份验证工厂进行 HTTP 管理接口	来自连接的身份。	如果成功使用 **security-domain**，则来自引用的 security-domain 的身份。
原生管理，包括通过 HTTP 升级，使用旧的 **security-realm** 接口	来自连接的身份。	不支持或匿名身份。
原生管理，包括通过 HTTP 升级，使用 **elytron** SASL 验证因 **security-domain**支持的接口	来自连接的身份。	如果成功使用 **security-domain**，则来自引用的 security-domain 的身份。



注意

如果 **identity** 资源中使用的安全域不信任来自身份验证的安全域，则会使用匿名身份。

当两者都使用相同的安全域时，**身份资源** 中使用的安全域不需要信任来自身份验证的安全域。

可信安全域不是传输。

如果未定义 **access=identity** 资源，则使用针对管理接口身份验证过程中建立的身份。在这种情况下，通过 **remoting** 子系统或使用应用程序使用连接建立的身份。

如果定义了 **access=identity** 资源，但管理接口使用的安全域不同，且未列在域列表中，不会建立身份。将使用在身份验证过程中建立的身份尝试 inflow。使用 **remoting** subsystem 或使用应用程序建立的连接建立的身份不会以这种方式流处理。

重要

如果管理接口使用旧的安全域进行保护，其身份将无法在不同的安全域之间被控制。在这种情况下，不应定义 **access=identity** 资源。因此，可以在身份验证过程中建立的身份直接使用。因此，**identity** 资源不支持使用 PicketBox 保护的应用程序。

3.1.5. 通过管理 CLI 使用 Elytron 客户端

您可以将管理 CLI 配置为使用 Elytron 客户端在连接到 JBoss EAP 时提供安全信息。

1. 使用 Elytron 保护管理接口。
   要将 Elytron 客户端与管理 CLI 搭配使用，您必须使用 Elytron 保护管理界面。您可以使用 Elytron 在 用户身份验证中使用 Elytron 保护管理接口的详细信息。

2. 创建 Elytron 客户端配置文件。
   您需要创建一个 Elytron 客户端配置文件，该文件中包含您的身份验证配置以及使用该配置的规则。您可以在 JBoss EAP *如何配置身份管理指南*的配置文件方法一节中找到创建身份验证配置的更多详细信息。

   示例： custom-config.xml

```
<configuration>
    <authentication-client xmlns="urn:elytron:client:1.2">
        <authentication-rules>
            <rule use-configuration="configuration1">
                <match-host name="localhost" />
            </rule>
        </authentication-rules>
        <authentication-configurations>
            <configuration name="configuration1">
                <sasl-mechanism-selector selector="DIGEST-MD5" />
                <providers>
                  <use-service-loader />
                </providers>
                <set-user-name name="user1" />
                <credentials>
                    <clear-password password="password123" />
                </credentials>
                <set-mechanism-realm name="exampleManagementRealm" />
             </configuration>
        </authentication-configurations>
    </authentication-client>
</configuration>
```

3. 通过管理 CLI 脚本使用 Elytron 客户端配置文件。

```
$ ./jboss-cli.sh -c  -Dwildfly.config.url=/path/to/custom-config.xml
```

3.2. 使用 ELYTRON 进行身份传播和转发

3.2.1. 为远程调用传播安全识别信息

JBoss EAP 7.1 引入了轻松配置服务器和应用的功能，以将安全身份从客户端传播到服务器以重新移动调用。您还可以配置服务器组件，以便在给定用户的安全身份内运行。

本节中的示例演示了如何转发安全身份凭证。它将客户端的安全身份和 Jakarta Enterprise Beans 传播到远程 Jakarta Enterprise Bean。它返回一个字符串，其中包含名为远程 Jakarta Enterprise Beans 的 **Principal** 名称以及用户的授权角色信息。示例由以下组件组成。

- 安全的 Jakarta Enterprise Beans 包含单一方法，可供所有用户访问，后者返回有关调用者的授权信息。
- 含有单一方法的中间 Jakarta Enterprise Beans。它利用远程连接，并调用受保护 Jakarta Enterprise Beans 的方法。

- 调用中间 Jakarta Enterprise Beans 的远程独立客户端应用。
- **META-INF/wildfly-config.xml** 文件，其中包含用于身份验证的身份信息。

您必须首先通过 配置服务器来启用安全身份传播。接下来 查看使用 **WildFlyInitialContext** onnectionFactoryy 查找并调用远程 Jakarta Enterprise Bean 的示例应用程序代码。

为**安全传播配置服务器**

1. 将 **ejb3** 子系统配置为使用 Elytron **ApplicationDomain**。

```
/subsystem=ejb3/application-security-domain=quickstart-domain:add(security-
domain=ApplicationDomain)
```

这会将以下 **application-security-domain** 配置添加到 **ejb3** 子系统：

```
<subsystem xmlns="urn:jboss:domain:ejb3:5.0">
    ....
    <application-security-domains>
        <application-security-domain name="quickstart-domain" security-
domain="ApplicationDomain"/>
    </application-security-domains>
</subsystem>
```

2. 添加 **PLAIN** 身份验证配置，以发送纯文本用户名和密码，以及用于出站连接的验证上下文。有关支持 身份传播的机制列表，请参阅支持安全 身份传播的机制列表。

```
/subsystem=elytron/authentication-configuration=ejb-outbound-configuration:add(security-
domain=ApplicationDomain,sasl-mechanism-selector="PLAIN")
/subsystem=elytron/authentication-context=ejb-outbound-context:add(match-rules=
[{authentication-configuration=ejb-outbound-configuration}])
```

这会在 **elytron** 子系统中添加以下 **authentication-client** 配置。

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-
providers="OracleUcrypto">
    <authentication-client>
        <authentication-configuration name="ejb-outbound-configuration" security-
domain="ApplicationDomain" sasl-mechanism-selector="PLAIN"/>
        <authentication-context name="ejb-outbound-context">
            <match-rule authentication-configuration="ejb-outbound-configuration"/>
        </authentication-context>
    </authentication-client>
    ....
</subsystem>
```

3. 将远程目的地出站套接字绑定添加到 **standard-sockets** 套接字绑定组。

```
/socket-binding-group=standard-sockets/remote-destination-outbound-socket-binding=ejb-
outbound:add(host=localhost,port=8080)
```

这会添加以下 **ejb-outbound** 出站套接字绑定到 **standard-sockets** 套接字绑定组。

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
```

```
offset="${jboss.socket.binding.port-offset:0}">
    ....
    <outbound-socket-binding name="ejb-outbound">
        <remote-destination host="localhost" port="8080"/>
    </outbound-socket-binding>
</socket-binding-group>
```

4. 添加远程出站连接，并在 HTTP 连接器中设置 SASL 身份验证工厂。

```
/subsystem=remoting/remote-outbound-connection=ejb-outbound-connection:add(outbound-socket-binding-ref=ejb-outbound, authentication-context=ejb-outbound-context)
/subsystem=remoting/http-connector=http-remoting-connector:write-attribute(name=sasl-authentication-factory,value=application-sasl-authentication)
```

这会将以下 **http-remoting-connector** 和 **ejb-outbound-connection** 配置添加到 **remoting** 子系统。

```
<subsystem xmlns="urn:jboss:domain:remoting:4.0">
    ....
    <http-connector name="http-remoting-connector" connector-ref="default" security-realm="ApplicationRealm" sasl-authentication-factory="application-sasl-authentication"/>
    <outbound-connections>
        <remote-outbound-connection name="ejb-outbound-connection" outbound-socket-binding-ref="ejb-outbound" authentication-context="ejb-outbound-context"/>
    </outbound-connections>
</subsystem>
```

5. 将 Elytron SASL 身份验证配置为使用 **PLAIN** 机制。

```
/subsystem=elytron/sasl-authentication-factory=application-sasl-authentication:write-attribute(name=mechanism-configurations,value=[{mechanism-name=PLAIN},{mechanism-name=JBOSS-LOCAL-USER,realm-mapper=local},{mechanism-name=DIGEST-MD5,mechanism-realm-configurations=[{realm-name=ApplicationRealm}]}])
```

这会将以下 **application-sasl-authentication** 配置添加到 **elytron** 子系统：

```
<subsystem xmlns="urn:wildfly:elytron:4.0" final-providers="combined-providers" disallowed-providers="OracleUcrypto">
    ....
    <sasl>
      ....
      <sasl-authentication-factory name="application-sasl-authentication" sasl-server-factory="configured" security-domain="ApplicationDomain">
          <mechanism-configuration>
              <mechanism mechanism-name="PLAIN"/>
              <mechanism mechanism-name="JBOSS-LOCAL-USER" realm-mapper="local"/>
              <mechanism mechanism-name="DIGEST-MD5">
                  <mechanism-realm realm-name="ApplicationRealm"/>
              </mechanism>
          </mechanism-configuration>
      </sasl-authentication-factory>
  </sasl>
  ....
</subsystem>
```

服务器现在已配置为启用以下示例应用程序的安全性传播。

查看应用程序代码示例(pagates a Security Identity)
在服务器配置中启用了安全身份传播后，Jakarta Enterprise Beans 客户端应用可以使用 **WildFlyInitialContextonnectionFactory** y 查找并调用 Jakarta Enterprise Beans 代理。Jakarta Enterprise Beans 调用为在客户端示例中身份验证的用户，如下所示。以下缩写代码示例来自 JBoss EAP 7.4 附带的 **ejb-security-context-propagation** Quickstart。有关安全身份传播的完整工作示例，请参阅快速入门。

要将 Jakarta Enterprise Beans 作为其他用户调用，您可以在上下文属性中设置 **Context.SECURITY_PRINCIPAL** 和 **Context.SECURITY_CREDENTIALS**。

示例：远程客户端

```
public class RemoteClient {

    public static void main(String[] args) throws Exception {
        // invoke the intermediate bean using the identity configured in wildfly-config.xml
        invokeIntermediateBean();

        // now lets programmatically setup an authentication context to switch users before invoking the
intermediate bean
        AuthenticationConfiguration superUser =
AuthenticationConfiguration.empty().setSaslMechanismSelector(SaslMechanismSelector.NONE.addMe
chanism("PLAIN")).
                useName("superUser").usePassword("superPwd1!");
        final AuthenticationContext authCtx = AuthenticationContext.empty().
                with(MatchRule.ALL, superUser);

        AuthenticationContext.getContextManager().setThreadDefault(authCtx);
        invokeIntermediateBean();
    }

    private static void invokeIntermediateBean() throws Exception {
        final Hashtable<String, String> jndiProperties = new Hashtable<>();
        jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.wildfly.naming.client.WildFlyInitialContextFactory");
        jndiProperties.put(Context.PROVIDER_URL, "remote+http://localhost:8080");
        final Context context = new InitialContext(jndiProperties);
        IntermediateEJBRemote intermediate = (IntermediateEJBRemote) context.lookup("ejb:/ejb-
security-context-propagation/IntermediateEJB!"
                + IntermediateEJBRemote.class.getName());
        // Call the intermediate EJB
        System.out.println(intermediate.makeRemoteCalls());
    }
}
```

示例：Intermediate Jakarta Enterprise Beans

```
@Stateless
@Remote(IntermediateEJBRemote.class)
@SecurityDomain("quickstart-domain")
@PermitAll
public class IntermediateEJB implements IntermediateEJBRemote {
```

```
    @EJB(lookup="ejb:/ejb-security-context-
propagation/SecuredEJB!org.jboss.as.quickstarts.ejb_security_context_propagation.SecuredEJBRemot
e")
    private SecuredEJBRemote remote;

    @Resource
    private EJBContext context;

    public String makeRemoteCalls() {
        try {
            StringBuilder sb = new StringBuilder("** ").
                    append(context.getCallerPrincipal()).
                    append(" * * \n\n");
            sb.append("Remote Security Information: ").
                    append(remote.getSecurityInformation()).
                    append("\n");

            return sb.toString();
        } catch (Exception e) {
            if (e instanceof RuntimeException) {
                throw (RuntimeException) e;
            }
            throw new RuntimeException("Teasting failed.", e);
        }
    }
}
```

示例：Example: Secured Jakarta Enterprise Beans

```
@Stateless
@Remote(SecuredEJBRemote.class)
@SecurityDomain("quickstart-domain")
public class SecuredEJB implements SecuredEJBRemote {

    @Resource
    private SessionContext context;

    @PermitAll
    public String getSecurityInformation() {
        StringBuilder sb = new StringBuilder("[");
        sb.append("Principal=[").
                append(context.getCallerPrincipal().getName()).
                append("], ");
        userInRole("guest", sb).append(", ");
        userInRole("user", sb).append(", ");
        userInRole("admin", sb).append("]");
        return sb.toString();
    }
}
```

示例： wildfly-config.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <authentication-client xmlns="urn:elytron:client:1.2">
```

```
        <authentication-rules>
            <rule use-configuration="default"/>
        </authentication-rules>
        <authentication-configurations>
            <configuration name="default">
                <set-user-name name="quickstartUser"/>
                <credentials>
                    <clear-password password="quickstartPwd1!"/>
                </credentials>
                <sasl-mechanism-selector selector="PLAIN"/>
                <providers>
                    <use-service-loader />
                </providers>
            </configuration>
        </authentication-configurations>
    </authentication-client>
</configuration>
```

3.2.2. 使用授权转发模式

除了凭证转发外，Elytron 还支持在对等点之间使用身份。在以下情况下很有用。

- 要求是，您无法通过线路来发送密码。
- 身份验证类型不支持凭证转发。
- 环境需要限制哪些系统可以接收传播的请求。

要使用授权转发，您首先在 转发服务器上配置身份验证客户端，然后将 接收服务器配置为接受和处理授权。

在转发服务器上配置身份验证客户端
要启用授权转发，您必须在转发服务器配置中配置身份验证客户端配置。

以下管理 CLI 命令创建了默认的身份验证客户端配置，以启用身份验证转发。如果需要，您可以配置更加高级的基于规则的选择。

示例：管理 CLI 命令以创建身份验证客户端配置

```
/subsystem=elytron/authentication-configuration=forwardit:add(authentication-
name=theserver1,security-domain=ApplicationDomain,realm=ApplicationRealm,forwarding-
mode=authorization,credential-reference={clear-text=thereallysecretpassword})
/subsystem=elytron/authentication-context=forwardctx:add(match-rules=[{authentication-
configuration=forwardit,match-no-user=true}])
```

这些命令将以下 **authentication-configuration** 和 **authentication-context** 配置添加到 **elytron** 子系统中。

示例：身份验证客户端配置

```
<authentication-client>
    <authentication-configuration name="forwardit" authentication-name="theserver1" security-
domain="ApplicationDomain" forwarding-mode="authorization" realm="ApplicationRealm">
        <credential-reference clear-text="thereallysecretpassword"/>
    </authentication-configuration>
```

```
    <authentication-context name="forwardctx">
        <match-rule match-no-user="true" authentication-configuration="forwardit"/>
    </authentication-context>
</authentication-client>
```

当转发服务器联系接收服务器时，不使用默认的基于身份验证的用户名和凭证，它使用预定义的服务器登录名 **theserver1** 来建立信任关系。

在接收服务器上配置授权转发
要使转发成功完成，需要使用与转发服务器传递的用户身份配置接收服务器配置。在这种情况下，您必须在接收服务器上配置名为 **theserver1** 的用户并使用正确的凭据。

您还必须在 **elytron** 子系统中配置"RunAs"权限映射，以允许从转发服务器传递的 **server1** 身份的身份切换。有关权限映射的更多信息，请参阅 *如何为 JBoss EAP 配置服务器安全性* 中的创建一个 Elytron 权限映射程序。

下面的命令添加一个名为 **auth-forwarding-permission-mapper** 的 **simple-permission-mapper**，它包括以下配置。

- 用户 **anonymous** 的权限映射。此用户没有权限，这可以防止匿名用户可以登录。
- 用户 **theserver1** 的权限映射。此用户被分配了 * 的 **RunAsPrincipalPermission** 权限，这可让此用户全局权限作为任何身份运行。如果您愿意，您可以将权限限制为特定的身份。
- 所有其他用户的权限映射。

示例：管理 CLI 命令创建 Simple Permission Mapper

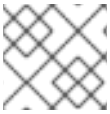
```
/subsystem=elytron/permission-set=run-as-principal-permission:add(permissions=[{class-
name="org.wildfly.security.auth.permission.RunAsPrincipalPermission",target-name="*"}])

/subsystem=elytron/simple-permission-mapper=auth-forwarding-permission-mapper:add(permission-
mappings=[{principals=["anonymous"]},{principals=["theserver1"],permission-sets=[{permission-
set=login-permission},{permission-set=default-permissions},{permission-set=run-as-principal-
permission}]},{match-all=true,permission-sets=[{permission-set=login-permission},{permission-
set=default-permissions}]}]
```

此命令将以下 **simple-permission-mapper** 配置添加到 **elytron** 子系统：

示例：简单权限映射配置

```
<mappers>
    <simple-permission-mapper name="auth-forwarding-permission-mapper">
        <permission-mapping>
            <principal name="anonymous"/>
            <!-- No permissions: Deny any permission to anonymous! -->
        </permission-mapping>
        <permission-mapping>
            <principal name="theserver1"/>
            <permission-set name="login-permission"/>
            <permission-set name="default-permissions"/>
            <permission-set name="run-as-principal-permission"/>
        </permission-mapping>
        <permission-mapping match-all="true">
            <permission-set name="login-permission"/>
```

```
            <permission-set name="default-permissions"/>
        </permission-mapping>
    </simple-permission-mapper>
</mappers>
<permission-sets>
    <permission-set name="login-permission">
        <permission class-name="org.wildfly.security.auth.permission.LoginPermission"/>
    </permission-set>
    <permission-set name="default-permissions">
        <permission class-name="org.wildfly.extension.batch.jberet.deployment.BatchPermission"
module="org.wildfly.extension.batch.jberet" target-name="*"/>
        <permission class-name="org.wildfly.transaction.client.RemoteTransactionPermission"
module="org.wildfly.transaction.client"/>
        <permission class-name="org.jboss.ejb.client.RemoteEJBPermission" module="org.jboss.ejb-
client"/>
    </permission-set>
    <permission-set name="run-as-principal-permission">
        <permission class-name="org.wildfly.security.auth.permission.RunAsPrincipalPermission" target-
name="*"/>
    </permission-set>
</permission-sets>
```



注意

默认配置中已存在 **login-permission** 和 **default-permissions** 权限集。

在转发授权后使用主体转换器时，这些转换程序都会在验证和授权主体中应用。

3.2.3. 创建 case-principal-transformer 以更改主体用户名的问题单字符

elytron 子系统包括 **case-principal-transformer** 主体转换程序。您可以使用这个主体转换器将主体的用户名更改为大写或小写字符。

case-principal-transformer 主体转换器包括默认设置为 **true** 的 **upper-case** 属性。

要演示 **case-principal-transformer** 的用例，请考虑您使用身份验证机制将主体映射到安全域。realm mapper 操控映射的主体，以识别安全域并加载其身份之一。身份验证机制会将身份传递到 post-realm 映射阶段和最终主体转换阶段。随后，验证机制会验证身份以进行身份验证。您可以使用 **case-principal-transformer** 主体转换器来转换映射主体的字符大小写格式。

该流程示例中在安全域上下文中使用 **case-principal-transformer**。您还可以在以下验证策略中使用内联主体转换器：

- **http-authentication-factory**
- **sasl-authentication-factory**
- **ssl-context**
- **aggregate-realm**

流程

1. 将 **case-principal-transformer** 添加到 **elytron** 子系统，然后选择用户名的字符大小写。

a. 要将转换器的用户名更改为大写字符，不要更改默认的 **大写** 属性值。

显示添加到 elytron 子系统中的 <transformer_name > 示例，并定义了默认的 大写 属性设置：

```
/subsystem=elytron/case-principal-transformer=<transformer_name>:add(upper-case="true")
```

或者，您可以截断命令语法以使用默认的 **大写** 属性值：

```
/subsystem=elytron/case-principal-transformer=<transformer_name>:add()
```

b. 要将转换器的用户名更改为小写字符，请将 **upper-case** 属性设置为 **false**。

显示添加到 elytron 子系统中的 <transformer_name> 示例，并将 upper-case 属性设置为 false ：

```
/subsystem=elytron/case-principal-transformer=<transformer_name>:add(upper-case="false")
```

2. *可选：*使用 **elytron** 子系统配置主体转换器。以下示例配置了一个主体转换器到由 **elytron** 子系统提供的默认 **ApplicationDomain** 配置。Elytron 将默认的 **ApplicationDomain** 配置应用到 **pre-realm-principal-transformer** ：

```
/subsystem=elytron/security-domain=ApplicationDomain:write-attribute(name=pre-realm-principal-transformer,value=<transformer_name>)
```

注意

您可以将 **post-realm-principal-transformer** 配置为在域映射程序标识了安全域后使用 **ApplicationDomain** 配置。

其他资源

- 有关 **upper-case** 属性的详情，请查看 表 A.26 **case-principal-transformer** 属性。

3.2.4. 检索安全身份凭证

在某些情况下，您可能需要检索用于传出调用的身份凭据，例如，由 HTTP 客户端使用。以下示例演示了如何以编程方式检索安全凭证。

```
import org.wildfly.security.auth.server.IdentityCredentials;
import org.wildfly.security.auth.server.SecurityDomain;
import org.wildfly.security.auth.server.SecurityIdentity;
import org.wildfly.security.credential.PasswordCredential;
import org.wildfly.security.password.interfaces.ClearPassword;

SecurityIdentity securityIdentity = null;
ClearPassword password = null;

// Obtain the SecurityDomain for the current deployment.
// The calling code requires the
```

```
//     org.wildfly.security.permission.ElytronPermission("getSecurityDomain") permission
//     if running with a security manager.
SecurityDomain securityDomain = SecurityDomain.getCurrent();
if (securityDomain != null) {
    // Obtain the current security identity from the security domain.
    // This always returns an identity, but it could be the representation
    //     of the anonymous identity if no authenticated identity is available.
    securityIdentity = securityDomain.getCurrentSecurityIdentity();
    // The private credentials can be accessed to obtain any credentials delegated to the identity.
    // The calling code requires the
    //     org.wildfly.security.permission.ElytronPermission("getPrivateCredentials")
    //     permission if running with a security manager.
    IdentityCredentials credentials = securityIdentity.getPrivateCredentials();
    if (credentials.contains(PasswordCredential.class)) {
        password =
credentials.getCredential(PasswordCredential.class).getPassword(ClearPassword.class);
    }
}
```

3.2.5. 支持安全身份传播机制

以下 SASL 机制支持传播安全身份：

- **PLAIN**
- **OAUTHBEARER**
- **GSSAPI**
- **GS2-KRB5**

以下 HTTP 机制支持传播安全身份：

- **FORM** [1]
- **BASIC**
- **BEARER_TOKEN**
- **SPNEGO**

[1] **FORM** 身份验证不会由 Web 浏览器自动处理。因此，无法在 HA 集群中运行的使用 **FORM** 身份验证的 Web 应用程序传播。其他机制，如 **BASIC** 和 **SPNEGO**，在 HA 集群环境中支持身份传播。

3.3. 使用 ELYTRON 进行身份切换

3.3.1. 在 Server-to-server Jakarta Enterprise Beans 调用中切换身份

默认情况下，当您对部署到应用服务器的 Jakarta Enterprise Beans 进行远程调用时，用于身份验证的身份与源服务器中使用的身份相同。在某些情况下，您可能想要在不同身份的安全上下文中运行远程安全 Jakarta Enterprise Beans。

您可以使用 Elytron API 将 server-to-server Jakarta Enterprise Beans 调用中的身份切换。当您这样做时，使用在 API 调用中编程指定的身份，通过连接接收的请求作为新请求执行。

以下代码示例演示了在远程 Jakarta Enterprise Beans 上切换用于身份验证的身份。**securityDomain.authenticate（）** 方法传递的 **remoteUsername** 和 **remotePassword** 参数是用于目标服务器上进行身份验证的身份凭证。

示例：在 Server-to-server Jakarta Enterprise Beans 调用中切换身份

```
SecurityDomain securityDomain = SecurityDomain.getCurrent();
Callable<T> forwardIdentityCallable = () -> {
   return AuthenticationContext.empty()
          .with(MatchRule.ALL,
                AuthenticationConfiguration.empty()
                .setSaslMechanismSelector(SaslMechanismSelector.ALL)
                .useForwardedIdentity(securityDomain))
          .runCallable(callable);
};

securityDomain.authenticate(remoteUsername, new
PasswordGuessEvidence(remotePassword.toCharArray())).runAs(forwardIdentityCallable);
```

3.4. 使用传统核心管理身份验证进行用户身份验证

3.4.1. 默认用户配置

JBoss EAP 中的所有管理界面默认是安全的，用户可以以两种不同的方式访问它们：本地接口和远程接口。JBoss EAP 安全 *架构指南* 的"默认 安全性和 JBoss EAP" 部分介绍了这两种身份验证机制的基础知识。默认情况下，在 *Management Realm* 安全域中配置对这些接口的访问。最初，本地接口已启用，需要访问运行 JBoss EAP 实例的主机计算机。也启用了远程访问，并配置为使用基于文件的身份存储。默认情况下，它使用 **mgmt-users.properties** 文件来存储用户名和密码，**mgmt-groups.properties** 来存储用户组信息。

使用 ***EAP_HOME*/bin/** 目录中的 included **adduser** 脚本添加到这些文件中。

通过 **adduser** 脚本添加用户：

1. 运行 **add-user.sh** 或 **add-user.bat** 命令。
2. 选择是否添加管理用户或应用程序用户。
3. 选择用户要添加到的域。默认情况下，唯一可用的域是 **ManagementRealm** 和 **ApplicationRealm**。如果添加了自定义域，则可以手动输入其名称。
4. 提示时输入所需的用户名、密码和可选角色。更改将写入到安全域的每个属性文件中。

3.4.2. 通过 LDAP 添加身份验证

JBoss EAP 还支持使用 LDAP 身份验证来保护管理界面。LDAP 及其工作原理与 JBoss EAP 的基础知识、使用 LDAP 以及管理界面的使用 LDAP 以及在红帽 JBoss 企业应用平台 7 *安全架构指南* 的 ManagementRealm 部分使用 LDAP 的基础知识。有关如何使用 LDAP 身份验证保护管理接口的更多信息，请参阅 JBoss EAP *如何配置身份管理* 指南中的使用 LDAP 保护管理接口 部分。

3.4.3. 使用 JAAS 来保护管理界面

JAAS 是一种声明性安全 API，供 JBoss EAP 用于管理安全性。有关 JAAS 和声明安全性的更多详情和背景信息，请参阅 Red Hat JBoss Enterprise Application Platform 安全架构指南中的 Declative *Security and JAAS* 部分。

注意

当 JBoss EAP 实例配置为以 **ADMIN_ONLY** 模式运行时，不支持使用 JAAS 来保护管理接口。有关 **ADMIN_ONLY** 模式的更多信息，请参阅 JBoss EAP *配置指南中的* ADMIN_ONLY Mode 一节中的运行 JBoss EAP。

要使用 JAAS 对管理接口进行身份验证，必须执行以下步骤：

1. 创建安全域。
   在这个示例中，使用 **UserRoles** 登录模块创建了一个安全域，但也可以使用其他登录模块：

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

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

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

2. 创建具有 JAAS 身份验证的安全域。

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

/core-service=management/security-realm=SecurityDomainAuthnRealm/authentication=jaas:add(name=UsersLMDomain)
```

3. 更新 **http-interface** 管理界面，以使用新的安全域。

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

4. **可选：** 分配组成员资格。
   属性 **assign-groups** 决定在安全域中加载的用户成员资格信息是否用于安全域中的组分配。当设置为 **true** 时，这个组分配用于基于角色的访问控制(RBAC)。

```
/core-service=management/security-realm=SecurityDomainAuthnRealm/authentication=jaas:write-attribute(name=assign-groups,value=true)
```

3.5. 基于角色的访问控制

基于角色的访问控制的基础知识涵盖了 基于角色的访问控制，以及将 RBAC 添加到 JBoss EAP *安全架构指南中的* 管理界面部分。

3.5.1. 启用基于角色的访问控制

默认情况下禁用基于角色的访问控制(RBAC)系统。它通过将 **provider** 属性从 **simple** 改为 **rbac** 来启用。**provider** 是 **management** 元素的 **access-control** 元素的属性。这可通过管理 CLI 或编辑服务器配

置 XML 文件（如果服务器离线）完成。当在运行的服务器上禁用或启用 RBAC 时，必须重新加载服务器配置才能生效。

警告

在将供应商更改为 **rbac** 之前，请确保您的配置具有映射到其中一个 RBAC 角色的用户，最好拥有至少一个 **Administrator** 或 **SuperUser** 角色。除关闭并编辑 XML 配置外，否则您的安装将无法管理。如果您使用 JBoss EAP 附带的一个标准 XML 配置启动，**$ local** 用户将映射到 **SuperUser** 角色，并且将启用本地身份验证方案。这将允许用户在与 JBoss EAP 进程相同的系统上运行 CLI，具有完整的管理权限。远程 CLI 用户和基于 Web 的管理控制台用户将没有权限。

建议在将提供程序切换到 **rbac** 之前，至少映射一个用户，而不是 **$local**。您可以执行与 **rbac** 提供程序关联的所有配置，即使提供程序设置为 **simple**。

启用后，只能被具有 **Administrator** 或 **SuperUser** 角色的用户禁用。默认情况下，如果管理 CLI 与服务器在同一计算机上运行，则作为 **SuperUser** 角色运行。

启用 RBAC 的 CLI

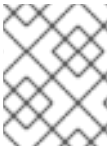
要使用管理 CLI 启用 RBAC，请使用访问授权资源的 **write-attribute** 操作将 **provider** 属性设置为 **rbac**。

```
/core-service=management/access=authorization:write-attribute(name=provider, value=rbac)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}

reload
```

在受管域中，访问控制配置是域范围内的配置的一部分，因此资源地址与上述相同，但管理 CLI 连接到主域控制器。

```
/core-service=management/access=authorization:write-attribute(name=provider,value=rbac)
{
    "outcome" => "success",
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    },
    "result" => undefined,
    "server-groups" => {"main-server-group" => {"host" => {"master" => {
        "server-one" => {"response" => {
            "outcome" => "success",
            "response-headers" => {
                "operation-requires-reload" => true,
                "process-state" => "reload-required"
            }
```

```
        }},
        "server-two" => {"response" => {
            "outcome" => "success",
            "response-headers" => {
                "operation-requires-reload" => true,
                "process-state" => "reload-required"
            }
        }}
    }}}}
}

reload --host=master
```



注意

与单机服务器一样，需要重新加载或重启才能使更改生效。在受管域中，域中的所有主机和服务器都需要重新加载或重启，自主域控制器开始。

管理 CLI 命令以禁用 RBAC
要使用管理 CLI 禁用 RBAC，请使用访问授权资源的 **write-attribute** 操作将 **provider** 属性设置为 **simple**。

```
/core-service=management/access=authorization:write-attribute(name=provider, value=simple)
```

用于启用或禁用 RBAC 的 XML 配置
如果服务器离线，则编辑 XML 配置以启用或禁用 RBAC。为此，请编辑 management 元素的 access-control 元素的 provider 属性。将值设为 **rbac** 以启用，而 **simple** 设置为 disable。

示例：启用或禁用 RBAC 的 XML 配置

```
<management>
  <access-control provider="rbac">
    <role-mapping>
      <role name="SuperUser">
        <include>
          <user name="$local"/>
        </include>
      </role>
    </role-mapping>
  </access-control>
</management>
```

3.5.2. 更改权限组合策略

权限组合策略决定如何确定用户是否分配多个角色。这可设置为 **permissive** 或 **reject**。默认值为 **permissive**。

当设置为 **permissive** 时，如果将任何角色分配给允许某一操作的用户，则允许该操作。

当设置为 **拒绝** 时，如果为某个用户分配多个角色，则不操作。这意味着，当将策略设置为拒绝每个用户时，应仅分配一个角色。当策略设置为拒绝时，具有多个角色的用户无法使用管理控制台或管理 CLI。

权限组合策略通过将 **permission-combination-policy** 属性设置为 **permissive** 或 **rejecting** 来进行配置。这可通过管理 CLI 或编辑服务器配置 XML 文件（如果服务器离线）完成。**permission-**

combination-policy 属性是 **access-control** 元素的一部分，而 **access-control** 元素可在 **management** 元素中找到。

设置权限组合策略
使用访问授权资源的 write-attribute 操作将 permission-combination-policy 属性设置为所需的策略名称。

```
/core-service=management/access=authorization:write-attribute(name=permission-combination-policy, value=POLICYNAME)
```

有效的策略名称是 **reject** 和 **permissive**。

示例：拒绝权限组合策略的管理 CLI 命令

```
/core-service=management/access=authorization:write-attribute(name=permission-combination-policy, value=rejecting)
```

如果服务器离线，可以编辑 XML 配置以更改权限组合策略值。为此，请编辑 **access-control** 元素的 **permission-combination-policy** 属性。

示例：拒绝权限组合策略的 XML 配置

```
<access-control provider="rbac" permission-combination-policy="rejecting">
  <role-mapping>
    <role name="SuperUser">
      <include>
        <user name="$local"/>
      </include>
    </role>
  </role-mapping>
</access-control>
```

3.5.3. 管理角色

启用基于角色的访问控制(RBAC)时，允许什么管理用户由向用户分配的角色决定。JBoss EAP 7 使用 中的系统 includes 和 excludes，它基于用户和组成员资格来确定用户所属的角色。

如果用户被视为一个角色，则用户被视为被分配给角色：

- 列出为要包含在角色中的用户，或者
- 列在角色中的组的成员。

如果用户不是，则用户也被视为被分配给角色：

- 列为从角色中排除的用户，或者
- 列出角色中排除的组的成员。

排除的优先级高于包含项。

可以使用管理控制台和管理 CLI 配置用户和组的角色包含和排除设置。

只有 **SuperUser** 或 **Administrator** 角色的用户才能执行此配置。

3.5.3.1. 使用管理 CLI 配置用户角色分配

将用户和组映射到角色的配置位于： **/core-service=management/access=authorization** 作为 **role-mapping** 元素。

只有 **SuperUser** 或 **Administrator** 角色的用户才能执行此配置。

查看角色分配配置
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)
{
    "outcome" => "success",
    "result" => [
        "Administrator",
        "Deployer",
        "Maintainer",
        "Monitor",
        "Operator",
        "SuperUser"
    ]
}
```

使用指定 role-mapping 的 **read-resource** 操作获取特定角色的完整详情：

```
/core-service=management/access=authorization/role-mapping=ROLENAME: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"
            }
        }
    }
}
```

添加新角色
此流程演示了如何为角色添加 role-mapping 条目。这必须在配置角色前完成。

使用 **add** 操作来添加新角色配置。

```
/core-service=management/access=authorization/role-mapping=ROLENAME:add
```

ROLENAME 是新映射的角色的名称，如 **Auditor**。

示例：管理CLI命令以进行新角色配置

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

添加角色中包括的用户
此流程演示了如何将用户添加到包含角色列表中。

如果没有进行任何角色配置，则必须首先执行角色映射条目。

使用 **add** 操作向包含角色列表添加用户条目。

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:add(name=USERNAME, type=USER)
```

- *ROLENAME* 是所配置的角色的名称，如 **Auditor**。
- *ALIAS* 是该映射的唯一名称。红帽建议将命名规则用于别名，如 **user-*USERNAME***（例如 **user-max**）。
- *USERNAME* 是添加到 include 列表中的用户的名称，如 **max**。

示例：角色中包含的用户管理 CLI 命令

```
/core-service=management/access=authorization/role-mapping=Auditor/include=user-
max:add(name=max, type=USER)
```

将用户添加到角色中已排除用户
此流程演示了如何将用户添加到角色的排除列表中。

如果没有进行任何角色配置，则必须首先执行角色映射条目。

使用 **add** 操作将用户条目添加到角色的 excludes 列表中。

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:add(name=USERNAME, type=USER)
```

- *ROLENAME* 是配置的角色的名称，如 **Auditor**。
- *USERNAME* 是添加到 exclude 列表中的用户的名称，如 **max**。
- *ALIAS* 是该映射的唯一名称。红帽建议将命名规则用于别名，如 **user-*USERNAME***（例如 **user-max**）。

示例：在一个角色中没有管理 CLI 命令用户

```
/core-service=management/access=authorization/role-mapping=Auditor/exclude=user-
max:add(name=max, type=USER)
```

删除用户角色包含配置
此流程演示了如何删除用户包含角色映射的条目。

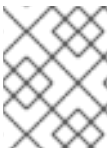
使用 **remove** 操作来删除该条目。

```
/core-service=management/access=authorization/role-mapping=ROLENAME/include=ALIAS:remove
```

- *ROLENAME* 是所配置的角色的名称，如 **Auditor**。
- *ALIAS* 是该映射的唯一名称。红帽建议将命名规则用于别名，如 **user-*USERNAME***（例如 **user-max**）。

示例：删除用户角色的管理 CLI 命令包含配置

```
/core-service=management/access=authorization/role-mapping=Auditor/include=user-max:remove
```



注意

从 includes 列表中删除用户不会从系统中删除用户，也不保证该角色不会分配给该用户。该角色可能仍然根据组成员资格来分配。

删除用户角色独到配置
此流程演示了如何从角色映射中删除用户排除条目。

使用 remove 操作来删除该条目。

```
/core-service=management/access=authorization/role-mapping=ROLENAME/exclude=ALIAS:remove
```

- *ROLENAME* 是所配置的角色的名称，如 **Auditor**。
- *ALIAS* 是该映射的唯一名称。红帽建议将命名规则用于别名，如 **user-*USERNAME***（例如 **user-max**）。

```
/core-service=management/access=authorization/role-mapping=Auditor/exclude=user-max:remove
```

注意

从 excludes 列表中删除用户不会从系统中删除用户，也不保证该角色将被分配给该用户。角色可能仍然会根据组成员资格排除。

3.5.4. 使用 Elytron subsystem 配置用户角色分配

除了直接为用户添加角色映射外，如 管理角色 部分所述，您还可以配置 RBAC 角色，以直接从 **elytron** 子系统提供的身份获取。

配置 RBAC 系统以使用 **elytron** 子系统提供的角色：

```
/core-service=management/access=authorization:write-attribute(name=use-identity-roles,value=true)
```

重要

必须启用 RBAC 来使用这个功能，并且主体必须具有 RBAC 角色。

3.5.5. 角色和用户组

用户组是一个任意标签，可分配给一个或多个用户。使用管理接口进行身份验证时，会根据管理界面如何保护用户，从 **elytron** 子系统或核心管理身份验证中分配组。RBAC 系统可以配置为根据他们所属的用户组自动为用户分配角色。它还可根据组成员资格从角色中排除用户。

3.5.6. 使用管理 CLI 配置组角色分配

在管理控制台和管理 CLI 中，可以在角色中包含或排除的组。本主题仅显示使用管理 CLI。

将用户和组映射到角色的配置位于管理 API 中，地址为：**/core-service=management/access=authorization** 作为 **role-mapping** 元素。

只有 **SuperUser** 或 **Administrator** 角色的用户才能执行此配置。

查看组角色分配配置
使用 **read- Child-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)
{
    "outcome" => "success",
    "result" => [
        "Administrator",
        "Deployer",
        "Maintainer",
        "Monitor",
        "Operator",
        "SuperUser"
    ]
}
```

使用指定 role-mapping 的 **read-resource** 操作获取特定角色的完整详情：

```
/core-service=management/access=authorization/role-mapping=ROLENAME: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"
```

```
        }
      }
    }
}
```

添加新角色
此流程演示了如何为角色添加 role-mapping 条目。这必须在配置角色前完成。

使用 **add** 操作来添加新角色配置。

```
/core-service=management/access=authorization/role-mapping=ROLENAME:add
```

添加角色中包括的组
此流程演示了如何在角色包含列表中添加组。

如果没有进行任何角色配置，则必须首先执行角色映射条目。

使用 **add** 操作向包含角色列表添加组条目。

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/include=ALIAS:add(name=GROUPNAME, type=GROUP)
```

- *ROLENAME* 是所配置的角色的名称，如 **Auditor**。
- *GROUPNAME* 是添加到 include 列表中的组的名称，如 **investigators**。
- *ALIAS* 是该映射的唯一名称。红帽建议为您的别名使用命名规则，如 **group-*GROUPNAME*** （例如： **group-investigators**）。

示例：将组添加到角色中的管理 CLI 命令

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

将组添加到角色中除外
此流程演示了如何在角色排除列表中添加组。

如果没有进行任何角色配置，则必须首先创建角色映射条目。

使用 **add** 操作将组条目添加到角色的 excludes 列表中。

```
/core-service=management/access=authorization/role-
mapping=ROLENAME/exclude=ALIAS:add(name=GROUPNAME, type=GROUP)
```

- *ROLENAME* 是所配置的角色的名称，如 **Auditor**。
- *GROUPNAME* 是添加到 include 列表中的组的名称，如 **supervisors**。
- *ALIAS* 是该映射的唯一名称。红帽建议为您的别名使用命名规则，如 **group-*GROUPNAME*** （例如： **group-supervisors**）。

示例：将组添加到角色中的管理 CLI 命令

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

删除组角色包含配置
此流程演示了如何删除组包括角色映射的条目。

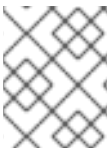
使用 **remove** 操作来删除该条目。

```
/core-service=management/access=authorization/role-mapping=ROLENAME/include=ALIAS:remove
```

- *ROLENAME* 是所配置的角色的名称，如 **Auditor**。
- *ALIAS* 是该映射的唯一名称。红帽建议为您的别名使用命名规则，如 **group-*GROUPNAME*** （例如： **group-investigators**）。

示例：删除组角色的管理 CLI 命令包含配置

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



注意

从 includes 列表中删除组不会从系统中删除组，也不保证该角色不会分配给此组中的用户。该角色可能仍然单独分配给组中的用户。

删除 User Group Exclude Entry
此流程演示了如何从角色映射中删除组排除条目。

使用 **remove** 操作来删除该条目。

```
/core-service=management/access=authorization/role-mapping=ROLENAME/exclude=ALIAS:remove
```

- *ROLENAME* 是所配置的角色的名称，如 **Auditor**。
- *ALIAS* 是该映射的唯一名称。红帽建议为您的别名使用命名规则，如 **group-*GROUPNAME*** （例如： **group-supervisors**）。

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

注意

从 excludes 列表中删除组不会从系统中删除组。它也不保证该角色将被分配给组的成员。角色可能仍然会根据组成员资格排除。

3.5.7. 通过 LDAP 使用 RBAC

介绍了将 RBAC 与 LDAP 搭配使用的基础知识，以及如何将 JBoss EAP 与 LDAP 搭配使用 RBAC 阐述在 JBoss EAP *如何配置身份管理指南中的* LDAP 和 RBAC 部分。

3.5.8. 有范围角色

有作用域角色是用户定义的角色，可授予其中一个标准角色的权限，但仅对 JBoss EAP 受管域中的一个或多个指定的服务器组或主机授予。通过有作用域角色，可以允许管理用户将权限授予限制那些仅需要那些服务器组或主机的权限。

重要

可为分配了 **Administrator** 或 **SuperUser** 角色的用户创建作用域角色。

它们由五个特征定义：

- 唯一的名称。
- 它要基于的标准角色。
- 如果它适用于服务器组或主机。
- 仅限于的服务器组或主机的列表。
- 如果所有用户被自动包含。默认值为 **false**。

创建范围的角色后，可以按照与标准角色相同的方式分配给用户和组。

创建全范围角色不允许定义新权限。有作用域角色只能用于在有限范围内应用现有角色的权限。例如，可以基于 restricted 到单个服务器组的 **Deployer** 角色创建范围的角色。

角色只能限制以下两范围：

主机范围内的角色

主机作用域的角色将该角色的权限限制为一个或多个主机。这意味着，可以访问相关的 **/host=*/** 资源树，但特定于其他主机的资源会被隐藏。

server-group-scoped 角色

即 server-group 范围的角色将该角色的权限限制为一个或多个服务器组。此外，角色权限也适用于与指定 *server-groups* 关联的 profile、套接字绑定组、服务器配置和服务器资源。任何与 server-group 相关的子资源都不会对用户可见。

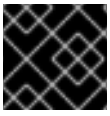
重要

有些资源无法被 **server-group** 和 **host** 范围角色提供简化的管理视图，以提高可用性。这与无法被寻址的资源不同，从而保护敏感数据。

对于 **主机** 范围的角色，这意味着如果它们与角色指定的服务器组无关，则管理模型中的 **/host=*** 部分中的资源将不可见。

对于 **server-group** scoped 角色，这意味着 **配置文件** 中的资源、**socket-binding-group**、**部署**、**Deployment-overlay**、**server-group**、**server-config** 和 **server** 部分（如果与为角色指定的服务器组无关）将无法看到。

3.5.8.1. 通过管理 CLI 配置范围角色



重要

只有 **SuperUser** 或 **Administrator** 角色的用户才能执行此配置。

添加新的范围角色

要添加新的范围的角色，必须执行以下操作：

```
/core-service=management/access=authorization/role-mapping=NEW-SCOPED-ROLE:add
```

```
/core-service=management/access=authorization/server-group-scoped-role=NEW-SCOPED-ROLE:add(base-role=BASE-ROLE, server-groups=[SERVER-GROUP-NAME])
```

将 *NEW-SCOPED-ROLE*、*BASE-ROLE* 和 *SERVER-GROUP-NAME* 替换为正确的信息。

查看并编辑范围角色映射
使用以下命令可以查看有范围的角色详情，包括成员：

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

用正确的信息替换 *NEW-SCOPED-ROLE*。

要编辑有范围的角色详情，可以使用 **write-attribute** 命令。例如：

```
/core-service=management/access=authorization/role-mapping=NEW-SCOPED-ROLE:write-attribute(name=include-all, value=true)
```

用正确的信息替换 *NEW-SCOPED-ROLE*。

删除范围角色

```
/core-service=management/access=authorization/role-mapping=NEW-SCOPED-ROLE:remove
```

```
/core-service=management/access=authorization/server-group-scoped-role=NEW-SCOPED-ROLE:remove
```

用正确的信息替换 *NEW-SCOPED-ROLE*。

重要

如果分配了用户或组，则无法删除有范围的角色。首先删除角色分配，然后删除它。

添加和删除用户
向范围角色添加和删除用户与 添加和删除标准角色 的过程相同。

3.5.8.2. 从管理控制台配置范围角色

重要

只有 **SuperUser** 或 **Administrator** 角色的用户才能执行此配置。

可以通过下列步骤在管理控制台中找到作用域角色配置：

1. 登录到管理控制台。
2. 点 **Access Control** 选项卡。

3. 单击 **Roles** 以查看所有角色，包括范围角色。

以下流程描述了如何为有范围角色执行配置任务。

添加新的范围角色

1. 登录到管理控制台。
2. 点 **Access Control** 选项卡。
3. 选择 **Roles**，点 Add (+) 按钮。
4. 选择**主机范围角色或 服务器组范围角色**。
5. 指定以下详情：
   - **Name** ：新 scoped 角色的唯一名称。
   - **基本角色** ：此角色将对其权限进行基础的角色。
   - **Host** 或 **Server Groups** ：根据添加的范围角色类型，角色限制的主机或服务器组列表。可以选择多个条目。
   - **Include All**: 此角色应该自动包含所有用户。默认值为 **OFF**。
6. 点 **Add** 以创建新角色。

编辑范围角色

1. 登录到管理控制台。
2. 点 **Access Control** 选项卡。
3. 点左侧的 **Roles** 菜单。
4. 点所需范围的角色来编辑，然后点 **编辑**。
5. 更新所需详情以更改并单击 **保存按钮**。

查**看范围角色成**员

1. 登录到管理控制台。
2. 点 **Access Control** 选项卡。
3. 点左侧的 **Roles** 菜单。
4. 单击所需范围的角色，以查看包含和排除的成员。

删除范围角色

1. 登录到管理控制台。
2. 点 **Access Control** 选项卡。
3. 点左侧的 **Roles** 菜单。
4. 点所需范围的角色，从下拉列表中点击 **Remove**。

5. 点 Yes 以删除角色及其所有分配。

添加和删除用户
向范围角色添加和删除用户与添加和删除标准角色的过程相同。更新用户的作用域角色：

1. 登录到管理控制台。
2. 点 Access Control 选项卡。
3. 点左侧的 Roles 菜单，再点所需范围角色。
4. 选择 Plus(+)按钮，使其包含成员或要排除成员的减号(-)按钮。

3.5.9. 配置限制

3.5.9.1. 配置敏感限制

每个敏感度约束都定义了一组被视为 *敏感的* 资源。*敏感的* 资源通常应该是 secret（如密码）或对服务器有严重影响的资源，如网络、JVM 配置或系统属性。访问控制系统本身也被视为敏感。资源敏感度限制哪些角色能够读取、写入或处理特定资源。

sensitivity 约束配置为 **/core-service=management/access=authorization/constraint=sensitivity-classification**。

在管理模型中，每个敏感度约束都被识别为分类。然后分类被分成不同的类型。每个分类都有 **应用到** 元素，它是分类配置适用的路径模式列表。

要配置敏感度约束，请使用 **write-attribute** 操作设置 **configured-requires-read**、configure-**requires-write** 或 **configured-requires-addressable** 属性。要使该操作类型敏感，请将属性的值设置为 **true**，否则将其非敏感性设置为 **false**。默认情况下，这些属性不会被设置，并且使用 **default-requires-read**、**default-requires-write** 和 **default-requires-addressable** 的值。设置配置的属性后，它会使用该值而不是默认值。无法更改默认值。

示例：让读取系统属性成为敏感操作

```
/core-service=management/access=authorization/constraint=sensitivity-classification/type=core/classification=system-property:write-attribute(name=configured-requires-read,value=true)
```

示例：Result

```
/core-service=management/access=authorization/constraint=sensitivity-classification/type=core/classification=system-property:read-resource
```

```
{
    "outcome" => "success",
    "result" => {
        "configured-requires-addressable" => undefined,
        "configured-requires-read" => true,
        "configured-requires-write" => undefined,
        "default-requires-addressable" => false,
        "default-requires-read" => false,
        "default-requires-write" => true,
        "applies-to" => {
```

```
            "/core-service=platform-mbean/type=runtime" => undefined,
            "/system-property=*" => undefined,
            "/" => undefined
        }
    }
}
```

角色以及它们能够执行的对应操作取决于属性的配置。下表总结如下：

表 3.2. Sensitivity Constraint Configuration Outcomes

值	requires-read	requires-write	requires-addressable
true	读对敏感。只有 **Auditor**,**管理员**,**SuperUser** 可以读取。	写入是敏感的。只有 **Administrator** 和 **SuperUser** 可以写入。	寻址是敏感的。只有 **Auditor**,**管理员**,**SuperUser** 可以地址。
false	读取并不敏感。任何管理用户都可以读取。	写不敏感。只有 **Maintainer**, **Administrator** 和 **SuperUser** 可以写。**部署器** 也可以编写资源是应用程序资源。	寻址并不敏感。任何管理用户都可以解决。

3.5.9.2. 列出敏感限制

您可以使用以下管理 CLI 命令，直接从 JBoss EAP 管理模型中看到可用敏感度限制列表：

```
/core-service=management/access=authorization/constraint=sensitivity-classification:read-resource(include-runtime=true,recursive=true)
```

3.5.9.3. 配置应用程序资源限制

每个应用资源约束都定义了一组资源、属性和操作，它们通常与应用程序和服务部署相关。当应用程序资源限制被启用 **Deployer** 角色的管理用户时，会被授予对它应用到的资源的访问权限。

应用程序约束配置位于 **/core-service=management/access=authorization/constraint=application-classification/**。

每个应用程序资源限制都被标识为分类。然后分类被分成不同的类型。每个分类都有 **应用到** 元素，它是分类配置适用的路径模式列表。

默认情况下，唯一启用的应用程序资源分类是 core。核心包括部署、部署覆盖和部署操作。

要启用应用程序资源，使用 **write-attribute** 操作将分类的 **configured-application** 属性设置为 **true**。要禁用应用程序资源，请将此属性设置为 **false**。默认情况下不设置这些属性，并且使用 **default-application** 属性的值。无法更改默认值。

示例：启用日志记录器的应用程序资源类别

```
/core-service=management/access=authorization/constraint=application-classification/type=logging/classification=logging-profile:write-attribute(name=configured-application,value=true)
```

示例：Result

```
/core-service=management/access=authorization/constraint=application-classification/type=logging/classification=logging-profile:read-resource
```

```
{
    "outcome" => "success",
    "result" => {
        "configured-application" => true,
        "default-application" => false,
        "applies-to" => {"/subsystem=logging/logging-profile=*" => undefined}
    }
}
```

重要

应用程序资源限制适用于其配置的所有资源。例如，无法授予 **Deployer** 用户访问一个数据源资源，但不能访问另一个数据源资源。如果需要这种隔离级别，则建议在不同的服务器组中配置资源，并为每个组创建不同的范围 **Deployer** 角色。

3.5.9.4. 列出应用程序资源限制

您可以使用以下管理 CLI 命令，直接从 JBoss EAP 管理模型中看到可用应用程序资源限制列表：

```
/core-service=management/access=authorization/constraint=application-classification:read-resource(include-runtime=true,recursive=true)
```

3.5.9.5. 配置 Vault 表达式约束

默认情况下，读取和写入 vault 表达式是敏感操作。配置 vault 表达式约束可允许将这些操作或两个操作都设置为非敏感。更改此约束可让更多的角色读写 vault 表达式。

vault 表达式约束位于 **/core-service=management/access=authorization/constraint=vault-expression**。

要配置 vault 表达式约束，请使用 **write-attribute** 操作设置 **configured-requires-write** 和 **configured-requires-read** 设为 **true** 或 **false** 的属性。默认情况下，不设置这些设置，并且使用 **default-requires-read** 和 **default-requires-write** 的值。无法更改默认值。

示例：使 Vault 表达式成为非敏感操作

```
/core-service=management/access=authorization/constraint=vault-expression:write-attribute(name=configured-requires-write,value=false)
```

示例：Result

```
/core-service=management/access=authorization/constraint=vault-expression:read-resource
```

-

```
{
    "outcome" => "success",
    "result" => {
        "configured-requires-read" => undefined,
        "configured-requires-write" => false,
        "default-requires-read" => true,
        "default-requires-write" => true
    }
}
```

角色以及对应的 vault 表达式，它们将能够读取和写入，这取决于属性的配置。下表总结如下：

表 3.3. Vault Expression Constraint Configuration Outcomes

值	requires-read	requires-write
true	读操作是敏感的。只有 **Auditor**、**管理员** 和 **SuperUser** 可以读取。	写入操作是敏感的。只有 **Administrator** 和 **SuperUser** 可以写入。
false	读操作不敏感。所有管理用户都可以读取。	写入操作不敏感。**监控**、**管理员** **和** 超级用户可以写入。**Deployer** 也可以写入 vault 表达式在应用程序资源中。

第 4 章 凭证的安全存储

JBoss EAP 允许在配置文件外对敏感字符串进行加密。这些字符串可以存储在密钥存储中，然后用于应用和验证系统。敏感字符串可以存储在以下任一位置：

- Credential Store - 在 JBoss EAP 7.1 中引入，通过加密存储到存储文件中，凭据存储可以安全地保护敏感和纯文本字符串。每个 JBoss EAP 服务器都可以包含多个凭据存储。
- Password Vault - 通常在旧配置中使用，密码 vault 使用 Java 密钥存储来存储配置文件外的敏感字符串。每个 JBoss EAP 服务器都只能包含一个密码 vault。

***EAP_HOME*/standalone/configuration/** 和 ***EAP_HOME*/domain/configuration/** 中的所有配置文件都可以默认读取。强烈建议您不要将明文密码存储在配置文件中，而将这些凭据放在 凭证存储 或者 密码 vault 中。

如果您决定将纯文本密码放在配置文件中，则仅限受限用户访问这些文件。至少，运行 JBoss EAP 7 的用户帐户需要读写访问。

4.1. 凭证存储在 ELYTRON 中

4.1.1. Elytron 提供的凭证存储

Elytron 提供两个默认凭证类型，您可以使用它来保存您的凭证：KeyStoreCredentialStore 和 PropertiesCredentialStore。您可以使用 JBoss EAP 管理 CLI 管理凭据存储，也可以使用 WildFly Elytron 工具管理它们脱机管理。除了两种默认存储类型外，您还可以创建、使用和管理您自己的自定义凭据存储。

4.1.1.1. KeyStoreCredentialStore/credential-store

您可以将所有 Elytron 凭证类型存储在 KeyStoreStore 中。**elytron** 子系统中的 KeyStore 的资源名称为 **credentials-store**。KeyStoreCredentialStore 使用 Java Development Kit(JDK)中的 KeyStore 实现提供的机制来保护您的凭证。

在管理 CLI 中访问 KeyStoreCredentialStore，如下所示：

```
/subsystem=elytron/credential-store
```

其他资源

- Elytron 中的凭证类型
- 使用 JBoss EAP 管理 CLI 的凭据存储操作
- 使用 WildFly Elytron 工具进行凭证存储操作
- KeyStore Javadoc
- **credential-store** 属性

4.1.1.2. PropertiesCredentialStore/secret-key-credential-store

为了正确启动，JBoss EAP 需要初始密钥来解锁某些安全资源。使用 **secret-key-credential-store** 提供此主密钥来解锁这些必要的服务器资源。您还可以使用 PropertiesCredentialStore 存储 SecretKeyCredential，它支持存储高级加密标准(AES)secret 密钥。使用文件系统权限来限制对凭据存储

的访问。理想情况下，您应该只向应用服务器授予访问权限，以限制访问此凭据存储。
PropertiesCredentialStore 的 **elytron** 子系统中的资源名称是 **secret-key-credential-store**，您可以按照如下所示在管理 CLI 中访问它：

```
/subsystem=elytron/secret-key-credential-store
```

有关创建和提供初始密钥的信息，请参阅 向 JBoss EAP 提供初始密钥来解锁安全资源。另外，您还可以从外部来源获取 master 密钥或密码。有关从外部源获取密码的信息，请参阅 获取来自外部来源的凭据存储的密码。

其他资源

- Elytron 中的凭证类型
- 使用 JBoss EAP 管理 CLI 的凭据存储操作
- 使用 WildFly Elytron 工具进行凭证存储操作
- **secret-key-credential-store** Attributes

4.1.2. Elytron 中的凭证类型

Elytron 提供了以下三种凭证类型来满足您的各种安全需求，您可将这些凭据存储在 Elytron 的凭据存储中。

PasswordCredential

使用这个凭证类型，您可以安全地存储纯文本或未加密的密码。对于需要密码的 JBoss EAP 资源，请使用指向 PasswordCredential 而不是纯文本密码的引用来维护密码的保密。

连接到数据库的示例

```
data-source add ... --user-name=db_user --password=StrongPassword
```

在这个示例数据库连接命令中，您可以看到密码： **StrongPassword**。这意味着其他人也可以在服务器配置文件中看到。

使用密码凭证连接到数据库的示例

```
data-source add ... --user-name=db_user --credential-reference=
{store=exampleKeyStoreCredentialStore, alias=passwordCredentialAlias}
```

当使用凭证引用而不是密码连接到数据库时，其它只能看到配置文件中的凭证引用，而不是您的密码

KeyPairCredential

您可以将 Secure Shell(SSH)和公钥 Cryptography Standards(PKCS)密钥对用作 KeyPairCredential。密钥对包括共享公钥和只有给定用户所知的私钥。
您只能使用 WildFly Elytron 工具管理 KeyPairCredential。

SecretKeyCredential

SecretKeyCredential 是高级加密标准(AES)密钥，可用于在 Elytron 中创建加密的表达式。有关加密表达式的详情，请参考 Elytron 中的加密表达式。

其他资源

- Elytron 提供的凭证存储
- 凭证存储支持的凭证类型
- Elytron 中的加密表达式

4.1.3. Elytron 凭证存储支持的凭证类型

下表描述了哪个凭证存储支持哪些凭证类型：

凭证类型	KeyStoreCredentialStore/credential-store	PropertiesCredentialStore/secret-key-credential-store
PasswordCredential	是	否
KeyPairCredential	是	否
SecretKeyCredential	是	是

其他资源

- Elytron 中的凭证类型
- Elytron 提供的凭证存储

4.1.4. 使用 JBoss EAP 管理 CLI 的凭据存储操作

要在运行的 JBoss EAP 服务器中管理 JBoss EAP 凭据，请使用提供的管理 CLI 操作。您可以使用 JBoss EAP 管理 CLI 管理 **PasswordCredential** 和 **SecretKeyCredential**。

注意

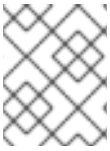
您只能对可修改的凭证存储进行这些操作。所有凭据存储类型默认为可修改。

4.1.4.1. 为独立服务器创建 KeyStore/credential-store

为在文件系统的任意目录中作为单机服务器运行的 JBoss EAP 创建一个 KeyStoreStore。为安全起见，只有有限用户才可以访问包含存储的目录。

先决条件

- 您至少提供对运行 JBoss EAP 的用户帐户的 KeyStoreCredentialStore 的目录的读/写权限。



注意

您不能与凭证存储和 **secret-key- credential- store** 的名称相同，因为它们实施相同的 Elytron 功能： **org.wildfly.security.credential-store**。

流程

- 使用以下管理 CLI 命令，创建一个 KeyStoreCredentialStore：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:add(path="<path_to_store_file>", relative-to=<base_path_to_store_file>, credential-reference={clear-text=<store_password>}, create=true)
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:add(path="exampleKeyStoreCredentialStore.jceks", relative-to=jboss.server.data.dir, credential-reference={clear-text=password}, create=true)
  {"outcome" => "success"}
  ```

其他资源

- 使用 JBoss EAP 管理 CLI 的凭据存储操作
- **credential-store** 属性

4.1.4.2. 为受管域创建 KeyStore/credential-store

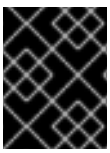
您可以在受管集群中创建 KeyStoreCredentialStore，但必须首先使用 WildFly Elytron 工具准备您的 KeyStoreCredentialStore。如果一个受管域中有多个主机控制器，请选择以下选项之一：

- 在每个主机控制器中创建 KeyStoreCredentialStore，并为每个 KeyStoreStore 添加凭证。
- 将填充的 KeyStoreStore 从一个主机操作系统复制到所有其他主机控制器。
- 将 KeyStoreCredentialStore 文件保存到网络文件系统(NFS)中，然后将该文件用于您创建的所有 KeyStoreCredentialStore 资源。

或者，您可以使用主机控制器上的凭证创建 KeyStoreCredentialStore 文件，而无需使用 WildFly Elytron 工具。

注意

您不必在每台服务器上定义一个 KeyStore 资源，因为同一配置集中的每台服务器都包含您的 KeyStoreCredentialStore 文件。您可以在服务器 **数据** 目录中找到 KeyStoreCredentialStore 文件，**相对-to=jboss.server.data.dir**。



重要

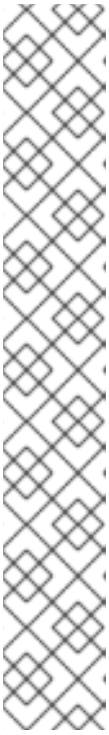
您不能与凭证存储和 **secret-key- credential- store** 的名称相同，因为它们实施相同的 Elytron 功能： **org.wildfly.security.credential-store**。

以下流程描述了如何使用 NFS 向所有主机控制器提供 KeyStoreCredentialStore 文件。

流程

1. 使用 WildFly Elytron 工具创建 KeyStoreCredentialStore 存储文件。有关此问题的更多信息，请参阅使用 WildFly Elytron 工具 存储操作 。

2. 分发存储文件。例如，使用 **scp** 命令将它分配给每个主机控制器，或者将其存储在 NFS 中，并将其用于所有 KeyStoreCredentialStore 资源。



注意

为了保持一致性，对于多个资源和主机操作系统使用的 KeyStoreStore 文件，您必须以只读模式使用 KeyStoreStore。另外，请确保为您的 KeyStoreStore 文件提供绝对路径。

语法

```
/profile=<profile_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<absolute_path_to_store_keystore>,credential-reference={clear-text="<store_password>"},create=false,modifiable=false)
```

示例

```
/profile=full-ha/subsystem=elytron/credential-store=exampleCredentialStoreDomain:add(path=/usr/local/etc/example-cred-store.cs,credential-reference={clear-text="password"},create=false,modifiable=false)
```

3. *可选：* 如果您需要在配置集中定义 **credential-store** 资源，请使用存储文件来创建资源。

语法

```
/profile=<profile_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_store_file>,credential-reference={clear-text="<store_password>"})
```

示例

```
/profile=full-ha/subsystem=elytron/credential-store=exampleCredentialStoreHA:add(path=/usr/local/etc/example-cred-store-ha.cs,credential-reference={clear-text="password"})
```

4. *可选：* 为主机控制器创建 KeyStoreStore 资源。

语法

```
/host=<host_controller_name>/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_store_file>,credential-reference={clear-text="<store_password>"})
```

示例

```
/host=master/subsystem=elytron/credential-store=exampleCredentialStoreHost:add(path=/usr/local/etc/example-cred-store-host.cs, credential-reference={clear-text="password"})
```

其他资源

- KeyStoreCredentialStore/credential-store
- 使用 WlidFly Ely Ely Ely Elytron 工具进行凭证存储操作
- **credential-store** 属性

4.1.4.3. 为独立服务器创建 PropertiesCredentialStore/secret-key-credential-store

使用管理 CLI 创建 PropertiesCredentialStore。在创建 PropertiesCredentialStore 时，JBoss EAP 默认生成 secret key。生成的密钥的名称是 **key**，其大小为 256 位。

先决条件

- 您至少提供对运行 JBoss EAP 的用户帐户的 PropertiesCredentialStore 的目录的读/写权限。

流程

- 使用以下命令，使用管理 CLI 创建 PropertiesCredentialStore：

  语法

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

  示例

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

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- 使用 JBoss EAP 管理 CLI 的凭据存储操作
- **secret-key-credential-store** Attributes

4.1.4.4. 将密码凭证添加到 KeyStoreStore/credential-store

为那些需要一个 PasswordCredential 到 KeyStore 的资源添加纯文本密码，以便在配置文件中隐藏该密码。然后，您可以引用该存储的凭证来访问这些资源，而无需公开您的密码。

先决条件

- 您已创建了 KeyStoreStore。
  有关创建 KeyStore 的详情，请参阅为独立服务器创建 KeyStoreCredentialStore/credential-store。

流程

- 将新的 PasswordCredential 添加到 KeyStoreCredentialStore：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:add-alias(alias=<alias>,
  secret-value=<secret-value>)
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:add-
  alias(alias=passwordCredentialAlias, secret-value=StrongPassword)
  {"outcome" => "success"}
  ```

验证

- 发出以下命令以验证 PasswordCredential 是否已添加到 KeyStoreCredentialStore 中：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => ["passwordcredentialalias"]
  }
  ```

其他资源

- KeyStoreCredentialStore/credential-store
- 在 JBoss EAP 配置中使用密码凭证
- **credential-store** 属性

4.1.4.5. 在 KeyStoreStore/credential-store 中生成 SecretKeyCredential

在 KeyStoreCredentialStore 中生成 SecretKeyCredential。默认情况下，Elytron 创建 256 位密钥。如果需要不同的大小，您可以在 **key-size** 属性中指定 128 位或 192 位键。

先决条件

- 您已创建了 KeyStoreStore。

有关创建 KeyStore 的详情，请参阅为独立服务器创建 KeyStoreCredentialStore/credential-store。

流程

1. 使用以下管理 CLI 命令，在 KeyStoreCredential 中生成 SecretKeyCredential：

   语法

   ```
   /subsystem=elytron/credential-store=<name_of_credential_store>:generate-secret-key(alias=<alias>, key-size=<128_or_192>)
   ```

   示例

   ```
   /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:generate-secret-key(alias=secretKeyCredentialAlias)
   ```

验证

- 发出以下命令，以验证 Elytron 将您的 SecretKeyCredential 存储在 KeyStoreCredentialStore 中：

  语法

  ```
  /subsystem=elytron/credential-store=<credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => [
          "secretkeycredentialalias"
      ]
  }
  ```

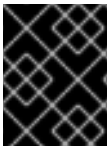
其他资源

- KeyStoreCredentialStore/credential-store
- 在 Elytron 中创建加密的表达式
- **credential-store** 属性

4.1.4.6. 在 PropertiesCredentialStore/secret-key-credential-store 中生成 SecretKeyCredential

在 PropertiesCredentialStore 中生成 SecretKeyCredential。默认情况下，Elytron 创建 256 位密钥。如果需要不同的大小，您可以在 **key-size** 属性中指定 128 位或 192 位键。

当您生成 SecretKeyCredential 时，Elytron 生成一个新的随机 secret 密钥，并将其保存为 SecretKeyCredential。您可以使用 PropertiesCredentialStore 上的 export 操作来查看凭证的内容。



重要

确保您创建 PropertiesCredentialStore、SecretKeyCredential 或两者的备份，因为 JBoss EAP 无法解密或检索丢失的 Elytron 凭据。

您可以使用 PropertiesCredentialStore 上的 **export** 操作来获取 SecretKeyCredential 的值。然后您可以将这个值保存为备份。如需更多信息，请参阅从 PropertiesCredentialStore/secret-key-credential-store 导出 SecretKey Credential。

先决条件

- 您已创建了 PropertiesCredentialStore。
  有关创建 PropertiesCredentialStore 的详情，请参考为单机服务器创建 PropertiesCredentialStore/secret-key-credential-store。

流程

- 使用以下管理 CLI 命令，在 PropertiesCredentialStore 中生成 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_the_properties_credential_store>:generate-secret-key(alias=<alias>, key-size=<128_or_192>)
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:generate-secret-key(alias=secretKeyCredentialAlias)
  {"outcome" => "success"}
  ```

验证

- 发出以下命令以验证 Elytron 创建了 SecretKeyCredential:

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_the_properties_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => [
          "secretkeycredentialalias",
          "key"
      ]
  }
  ```

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- 在 Elytron 中创建加密的表达式
- **secret-key-credential-store** Attributes

4.1.4.7. 将 SecretKeyCredential 导入到 PropertiesCredentialStore/secret-key-credential-store

您可以将在 PropertiesCredentialStore 之外创建的 SecretKeyCredential 导入到 Elytron PropertiesCredentialStore 中。假设您从另一个凭证存储中导出了一个 SecretKeyCredential，例如，De-jaxba KeyStoreStore，您可以将其导入到 PropertiesCredentialStore。

先决条件

- 您已创建了 PropertiesCredentialStore。
  有关创建 PropertiesCredentialStore 的详情，请参考为单机服务器创建 PropertiesCredentialStore/secret-key-credential-store。
- 您已导出了一个 SecretKeyCredential。
  有关导出 SecretKeyCredential 的信息，请参阅从 PropertiesCredentialStore/secret-key-credential-store 中导出 SecretKey Credential。

流程

1. 使用以下命令在管理 CLI 中禁用命令缓存：

   重要

   如果不禁用缓存，则可以访问管理 CLI 历史记录文件的任何人都可以看到 secret 密钥。

   ```
   history --disable
   ```

2. 使用以下管理 CLI 命令导入 secret 密钥：

   语法

   ```
   /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:import-secret-key(alias=<alias>, key="<secret_key>")
   ```

   示例

   ```
   /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:import-secret-key(alias=imported,
   key="RUxZAUs+Y1CzEPw0g2AHHOZ+oTKhT9osSabWQtoxR+O+42o11g==")
   ```

3. 使用以下管理 CLI 命令重新启用命令缓存：

   ```
   history --enable
   ```

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- **secret-key-credential-store** Attributes

4.1.4.8. 列出 KeyStoreStore/credential-store 中的凭证

要查看 KeyStoreCredentialStore 中存储的所有凭据，您可以使用管理 CLI 列出它们。

流程

- 使用以下管理 CLI 命令，列出存储在 KeyStoreStore 中的凭证：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => [
          "passwordcredentialalias",
          "secretkeycredentialalias"
      ]
  }
  ```

其他资源

- KeyStoreCredentialStore/credential-store
- **credential-store** 属性

4.1.4.9. 列出 PropertiesCredentialStore/secret-key-credential-store 中的凭证

要查看 PropertiesCredentialStore 中存储的所有凭据，您可以使用管理 CLI 列出这些凭证。

流程

- 使用以下管理 CLI 命令，列出存储在 PropertiesCredentialStore 中的凭证：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:read-aliases()
  {
      "outcome" => "success",
  ```

```
    "result" => [
        "secretkeycredentialalias",
        "key"
    ]
}
```

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- **secret-key-credential-store** Attributes

4.1.4.10. 从 KeyStoreCredential/credential-store 导出 SecretKeyCredential

您可以从 KeyStoreCredential 导出现有的 SecretKeyCredential，以使用 SecretKeyCredential 或 创建 SecretKeyCredential 的备份。

先决条件

- 您已生成了一个 SecretKeyCredential 作为 KeyStore。
  有关在 KeyStoreCredential 中生成 SecretKeyCredential 的信息，请参阅在 KeyStoreCredentialStore/credential-store 中生成 SecretKeyCredential。

流程

- 使用以下管理 CLI 命令，从 KeyStoreCredential 导出 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:export-secret-key(alias=<alias>)
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:export-secret-key(alias=secretKeyCredentialAlias)
  {
      "outcome" => "success",
      "result" => {"key" =>
  "RUxZAUui+8JkoDCE6mFyA3cCIbSAZaXq5wgYejj1scYgdDqWiw=="}
  }
  ```

其他资源

- KeyStoreCredentialStore/credential-store
- **credential-store** 属性

4.1.4.11. 从 PropertiesCredentialStore/secret-key-credential-store 导出 SecretKeyCredential

您可以从 PropertiesCredentialStore 导出现有的 SecretKeyCredential，以使用 SecretKeyCredential 或 创建 SecretKeyCredential 的备份。

先决条件

- 您在 PropertiesCredentialStore 中生成了一个 SecretKeyCredential，或将其导入。
有关在 PropertiesCredentialStore 中生成 SecretKeyCredential 的信息，请参阅在 PropertiesCredentialStore/secret-key-credential-store 中生成 SecretKey Credential。

  有关将 SecretKeyCredential 导入到 PropertiesCredentialStore 的信息，请参阅导入 SecretKeyCredential to PropertiesCredentialStore/secret-key-credential-store

流程

- 使用以下管理 CLI 命令，从 PropertiesCredentialStore 导出 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:export-secret-key(alias=<alias>)
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:export-secret-key(alias=secretkeycredentialalias)
  {
      "outcome" => "success",
      "result" => {"key" => "RUxZAUtxXcYvz0aukZu+odOynIr0ByLhC72iwzlJsi+ZPmONgA=="}
  }
  ```

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- **secret-key-credential-store** Attributes

4.1.4.12. 从 KeyStoreCredentialStore/credential-store 中删除凭证

您可以将每个凭证类型存储在 KeyStoreStore 中，但默认情况下，当您删除凭证时，Elytron 会假定它是 PasswordCredential。如果要删除其他凭证类型，请在 **entry-type** 属性中指定它。

流程

- 使用以下管理 CLI 命令，从 KeyStoreStore 中删除凭证：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:remove-alias(alias=<alias>, entry-type=<credential_type>)
  ```

  删除密码凭证示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:remove-alias(alias=passwordCredentialAlias)
  {
      "outcome" => "success",
  ```

```
    "response-headers" => {"warnings" => [{
        "warning" => "Update dependent resources as alias 'passwordCredentialAlias' does not
exist anymore",
        "level" => "WARNING",
        "operation" => {
            "address" => [
                ("subsystem" => "elytron"),
                ("credential-store" => "exampleKeyStoreCredentialStore")
            ],
            "operation" => "remove-alias"
        }
    }]}
}
```

删除 SecretKeyCredential 示例

```
/subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:remove-
alias(alias=secretKeyCredentialAlias, entry-type=SecretKeyCredential)
{
    "outcome" => "success",
    "response-headers" => {"warnings" => [{
        "warning" => "Update dependent resources as alias 'secretKeyCredentialAl
ias' does not exist anymore",
        "level" => "WARNING",
        "operation" => {
            "address" => [
                ("subsystem" => "elytron"),
                ("credential-store" => "exampleKeyStoreCredentialStore")
            ],
            "operation" => "remove-alias"
        }
    }]}
}
```

验证

- 发出以下命令以验证 Elytron 删除了凭证：

  语法

  ```
  /subsystem=elytron/credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/credential-store=exampleKeyStoreCredentialStore:read-aliases()
  {
      "outcome" => "success",
      "result" => []
  }
  ```

  您删除的凭证不会被列出。

其他资源

- KeyStoreCredentialStore/credential-store
- **credential-store 属性**

4.1.4.13. 从 PropertiesCredentialStore/secret-key-credential-store 中删除凭证

您只能将 SecretKeyCredential 类型存储在 PropertiesCredentialStore 中。这意味着，当您从 PropertiesCredentialStore 中删除凭证时，您不必指定 **entry-type**。

流程

- 使用以下命令，从 PropertiesCredentialStore 中删除 SecretKeyCredential：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:remove-alias(alias=<alias>)
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:remove-alias(alias=secretKeyCredentialAlias)
  {
      "outcome" => "success",
      "response-headers" => {"warnings" => [{
          "warning" => "Update dependent resources as alias 'secretKeyCredentialAlias' does not exist anymore",
          "level" => "WARNING",
          "operation" => {
              "address" => [
                  ("subsystem" => "elytron"),
                  ("secret-key-credential-store" => "examplePropertiesCredentialStore")
              ],
              "operation" => "remove-alias"
          }
      }]}
  }
  ```

验证

- 发出以下命令以验证 Elytron 删除了凭证：

  语法

  ```
  /subsystem=elytron/secret-key-credential-store=<name_of_credential_store>:read-aliases()
  ```

  示例

  ```
  /subsystem=elytron/secret-key-credential-store=examplePropertiesCredentialStore:read-aliases()
  {
  ```

```
    "outcome" => "success",
    "result" => []
}
```

您删除的凭证不会被列出。

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- **secret-key-credential-store** Attributes

4.1.5. 使用 WildFly Elytron 工具进行凭证存储操作

4.1.5.1. 使用 WildFly Elytron 工具创建 KeyStore/credential-store

在 Elytron 中，您可以创建一个 KeyStoreCredentialStore，您可以在其中保存所有凭证类型。

流程

- 使用以下命令，使用 WildFly Elytron 工具创建一个 KeyStoreStore：

  语法

  ```
  $ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "<path_to_store_file>" --password <store_password>
  ```

  示例

  ```
  $ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "../cred_stores/example-credential-store.jceks" --password storePassword
  Credential Store has been successfully created
  ```

  如果您不想将存储密码包含在命令中，请省略该参数，然后在提示符下手动输入密码。您还可以使用由 WildFly Elytron 工具生成的已屏蔽密码。有关生成屏蔽密码的详情，请参考使用 WildFly Elytron 工具生成屏蔽的加密字符串。

其他资源

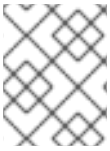
- KeyStoreCredentialStore/credential-store
- 使用 WildFly Elytron 工具生成屏蔽的加密字符串

4.1.5.2. 使用 Bouncy Castle 供应商创建 KeyStoreStore/credential-store

使用 Bouncy Castle 供应商创建一个 KeyStoreCredentialStore。

先决条件

- 确保您的环境已配置为使用 Bouncy Castle。
  如需更多信息，请参阅配置您的环境以使用 Bouncy Castle Provider。



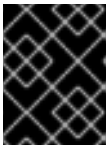
注意

您不能与凭证存储和 **secret-key- credential- store** 的名称相同，因为它们实施相同的 Elytron 功能： **org.wildfly.security.credential-store**。

流程

1. 定义 Bouncy Castle FIPS 密钥存储(**BCFKS**)密钥存储。FIPS 代表联邦信息处理标准。如果您已有，请转到下一步。

```
$ keytool -genkeypair -alias <key_pair_alias> -keyalg <key_algorithm> -keysize <key_size>
-storepass <key_pair_and_keystore_password> -keystore <path_to_keystore> -storetype
BCFKS -keypass <key_pair_and_keystore_password>
```



重要

确保密钥存储 **keypass** 和 **storepass** 属性相同。如果没有，**elytron** 子系统中的 **BCFKS** 密钥存储将无法定义它们。

2. 为 KeyStoreCredentialStore 生成 secret 密钥。

```
$ keytool -genseckey -alias <key_alias> -keyalg <key_algorithm> -keysize <key_size> -
keystore <path_to_keystore> -storetype BCFKS -storepass <key_and_keystore_password> -
keypass <key_and_keystore_password>
```

3. 使用以下命令，使用 WildFly Elytron 工具定义 KeyStoreCredentialStore：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store -c -a <alias> -x <alias_password> -p
<key_and_keystore_password> -l <path_to_keystore> -u
"keyStoreType=BCFKS;external=true;keyAlias=<key_alias>;externalPath=<path_to_credenti
al_store>"
```

其他资源

- KeyStoreCredentialStore/credential-store
- WildFly Elytron 工具 KeyStoreCredentialStore/credential-store 操作

4.1.5.3. 使用 WildFly Elytron 工具创建 PropertiesCredentialStore/secret-key-credential-store

在 Elytron 中，您可以创建一个 PropertiesCredentialStore 离线，您可以在其中保存 SecretKeyCredential 实例。

流程

- 使用以下命令，使用 WildFly Elytron 工具创建 PropertiesCredentialStore：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --create --location "<path_to_store_file>" --
type PropertiesCredentialStore
```

示例

```
$ bin/elytron-tool.sh credential-store --create --location=standalone/configuration/properties-credential-store.cs --type PropertiesCredentialStore
Credential Store has been successfully created
```

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- WildFly Elytron 工具 PropertiesCredentialStore/secret-key-credential-store 操作

4.1.5.4. WildFly Elytron 工具 KeyStoreCredentialStore/credential-store 操作

您可以使用 WildFly Elytron 工具执行各种 KeyStore 的任务，其中包括：

添加密码凭证

您可以使用以下 WildFly Elytron 工具命令将密码Credential 添加到 KeyStoreStore 中：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password <store_password> --add <alias> --secret <sensitive_string>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-store.jceks" --password storePassword --add examplePasswordCredential --secret
speci@l_db_pa$$_01
Alias "examplePasswordCredential" has been successfully stored
```

如果您不想将 secret 放在命令中，请省略该参数，然后在提示时手动输入 secret。

生成 SecretKeyCredential

您可以使用以下 WildFly Elytron 工具命令将 SecretKeyCredential 添加到 KeyStoreCredentialStore 中：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location=<path_to_the_credential_store> --password <store_password>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location "../cred_stores/example-credential-store.jceks" --password storePassword
Alias "example" has been successfully stored
```

如果您不想将 secret 放在命令中，请省略该参数，然后在提示时手动输入 secret。

默认情况下，当您在 JBoss EAP 中创建 SecretKeyCredential 时，您可以创建一个 256 位 secret 密钥。如果要更改大小，您可以指定 **--size=128** 或 **--size=192** 分别创建 128 位或 192 位键。

导入 SecretKeyCredential

您可以使用以下 WildFLy Elytron 工具命令导入 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --
location=<path_to_credential_store> --password=<store_password>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --
location=../cred_stores/example-credential-store.jceks --password=storePassword
```

输入您要导入的 secret 密钥。

列出所有凭证

您可以使用以下 WildFly Elytron 工具命令列出 KeyStore 中的凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --aliases
```

例如：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --aliases
Credential store contains following aliases: examplepasswordcredential example
```

检查是否存在别名

使用以下命令检查凭证存储中是否存在别名：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --exists <alias>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --exists examplepasswordcredential
Alias "examplepasswordcredential" exists
```

导出 SecretKeyCredential

您可以使用以下命令从 KeyStoreCredential 导出 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=<alias> --
location=<path_to_credential_store> --password=storePassword
```

-

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=example --
location=../cred_stores/example-credential-store.jceks --password=storePassword
Exported SecretKey for alias
example=RUxZAUtBiAnoLP1CA+i6DtcbkZHfybBJxPeS9mlVOmEYwjjmEA==
```

删除凭证

您可以使用以下命令从凭证存储中删除凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --password
<store_password> --remove <alias>
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "../cred_stores/example-credential-
store.jceks" --password storePassword --remove examplepasswordcredential
Alias "examplepasswordcredential" has been successfully removed
```

其他资源

- KeyStoreCredentialStore/credential-store
- Elytron 中的凭证类型

4.1.5.5. WildFly Elytron 工具 PropertiesCredentialStore/secret-key-credential-store 操作

您可以使用 WildFly Elytron 工具为 SecretKeyCredential 执行以下 PropertiesCredentialStore 操作：

生成 SecretKeyCredential

您可以使用以下 WildFly Elytron 工具在 PropertiesCredentialStore 中生成 **SecteKeyCredential** ：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location
"<path_to_the_credential_store>" --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --generate-secret-key=example --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
Alias "example" has been successfully stored
```

导入 SecretKeyCredential

您可以使用以下 WildFLy Elytron 工具命令导入 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --
location=<path_to_credential_store> --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-secret-key=imported --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
```

列出所有凭证

您可以使用以下 WildFly Elytron 工具命令列出 PropertiesCredentialStore 中的凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>"  --aliases --type
PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store  --location "standalone/configuration/properties-
credential-store.cs" --aliases --type PropertiesCredentialStore
Credential store contains following aliases: example
```

导出 SecretKeyCredential

您可以使用以下命令从 PropertiesCredentialStore 中导出 SecretKeyCredential：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=<alias> --location
"<path_to_credential_store>"  --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --export-secret-key=example --location
"standalone/configuration/properties-credential-store.cs" --type PropertiesCredentialStore
Exported SecretKey for alias
example=RUxZAUt1EZM7PsYRgMGypkGirSel+5Eix4aSgwop6jfxGYUQaQ==
```

删除凭证

您可以使用以下命令从凭证存储中删除凭证：

语法

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "<path_to_store_file>" --remove
<alias> --type PropertiesCredentialStore
```

示例

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location "standalone/configuration/properties-credential-store.cs"  --remove example --type PropertiesCredentialStore
Alias "example" has been successfully removed
```

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- Elytron 中的凭证类型

4.1.5.6. 将使用 WildFly Elytron 工具创建的凭据存储添加到 JBoss EAP 服务器

使用 WildFly Elytron 工具创建了凭据存储后，您可以将其添加到正在运行的 JBoss EAP 服务器中。

先决条件

- 您已使用 WildFly Elytron 工具创建了凭据存储。
  如需更多信息，请参阅使用 WildFly Elytron 工具创建 KeyStoreCredentialStore/credential-store。

流程

- 使用以下管理 CLI 命令，将凭证存储添加到正在运行的 JBoss EAP 服务器中：

  ```
  /subsystem=elytron/credential-store=<store_name>:add(location="<path_to_store_file>",credential-reference={clear-text=<store_password>})
  ```

  例如：

  ```
  /subsystem=elytron/credential-store=my_store:add(location="../cred_stores/example-credential-store.jceks",credential-reference={clear-text=storePassword})
  ```

将凭据存储添加到 JBoss EAP 配置后，您可以使用 credentials **-reference** 属性引用存储在凭据存储中的密码或敏感字符串。

如需更多信息，请使用 ***EAP_HOME*/bin/elytron-tool.sh credential-store --help** 命令获得可用选项的详细列表。

其他资源

- 在 JBoss EAP 配置中使用密码凭证
- **credential-store** 属性

4.1.5.7. WildFly Elytron 工具密钥对管理操作

您可以使用以下参数来操作 **elytron-tool.sh** 来处理凭据存储，例如生成可在凭证存储中存储的新密钥对。

生成密钥对

使用 **generate-key-pair** 命令创建一个密钥对。然后您可以在凭证存储的别名下存储密钥对。以下示例显示了创建 *RSA* 密钥对，其分配大小为 *3072* 位，存储在为凭据存储指定的位置。提供给密钥对的别名是 **example**。

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location=<path_to_store_file> --generate-key-pair example --algorithm RSA --size 3072
```

导入密钥对

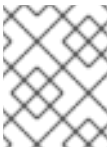
使用 **import-key-pair** 命令将现有的 SSH 密钥对导入到带有指定别名的凭据存储中。以下示例导入一个密钥对，其别名 *example* 来自 */home/user/.ssh/id_rsa* 文件，该文件包含采用 OpenSSH 格式的私钥：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --import-key-pair example --private-key-location /home/user/.ssh/id_rsa --location=<path_to_store_file>
```

导出密钥对

使用 **export-key-pair-public-key** 命令显示密钥对的公钥。公钥具有 OpenSSH 格式的指定别名。以下示例显示了别名 *示例* 的公钥：

```
$ EAP_HOME/bin/elytron-tool.sh credential-store --location=<path_to_store_file> --export-key-pair-public-key example

Credential store password:
Confirm credential store password:
ecdsa-sha2-nistp256
AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMfncZuHmR7uglb0M96ieArRFtp42xPn9+ugukbY8dyjOXoi
cZrYRyy9+X68fylEWBMzyg+nhjWkxJlJ2M2LAGY=
```



注意

在发出 **export-key-pair-public-key** 命令后，会提示您输入凭证存储密码短语。如果不存在密码短语，请将提示留空。

4.1.5.8. 在 Elytron 配置文件中使用存储的密钥对示例

密钥对包含两个独立的，但匹配，加密密钥：公钥和私钥。您需要在凭证存储中存储密钥对，然后才能在 **elytron** 配置文件中引用密钥对。然后，您可以提供对 Git 的访问权限，以管理您的单机服务器配置数据。

以下示例引用了 **elytron** 配置文件的 **<credential-stores>** 元素中的凭证存储及其属性。**<credential>** 元素引用凭证存储和别名，用于存储密钥对。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.6">

    <credential-stores>
      <credential-store name="${credential_store_name}">
        <protection-parameter-credentials>
          <clear-password password="${credential_store_password}"/>
        </protection-parameter-credentials>
```

```
        <attributes>
          <attribute name="path" value="${path_to_credential_store}"/>
        </attributes>
      </credential-store>
    </credential-stores>

    <authentication-rules>
      <rule use-configuration="${configuration_file_name}"/>
    </authentication-rules>

    <authentication-configurations>
      <configuration name="${configuration_file_name}">
        <credentials>
          <credential-store-reference store="${credential_store_name}" alias="${alias_of_key_pair}"/>
        </credentials>
      </configuration>
    </authentication-configurations>

  </authentication-client>
</configuration>
```

配置 **elytron** 配置文件后，密钥对可用于 SSH 身份验证。

其他资源

- WildFly Elytron 工具密钥对管理操作

4.1.5.9. 使用 WildFly Elytron 工具生成屏蔽的加密字符串

您可以使用 WildFly Elytron 工具来生成与 PicketBox 兼容的 **MASK** 加密字符串，而不使用凭据存储的纯文本密码。

流程

- 要生成屏蔽的字符串，请使用以下命令，并为 salt 和迭代计数提供值：

  ```
  $ EAP_HOME/bin/elytron-tool.sh mask --salt <salt> --iteration <iteration_count> --secret <password>
  ```

  例如：

  ```
  $ EAP_HOME/bin/elytron-tool.sh mask --salt 12345678 --iteration 123 --secret supersecretstorepassword

  MASK-8VzWsSNwBaR676g8ujiIDdFKwSjOBHCHgnKf17nun3v;12345678;123
  ```

  如果您不想在 命令中提供 secret，可以省略该参数，系统将提示您使用标准输入输入 secret。

如需更多信息，请使用 ***EAP_HOME*/bin/elytron-tool.sh mask --help** 命令来获取可用选项的详细列表。

4.1.6. Elytron 中的加密表达式

要保持敏感字符串的保密性，您可以在服务器配置文件中使用加密表达式而不是敏感字符串。

加密的表达式是结果，从加密字符串与 SecretKeyCredential 进行加密，然后将它与其编码前缀和解析器名称合并。编码前缀告知 Elytron 表达式是一个加密的表达式。解析器将加密表达式映射到凭据存储中对应的 SecretKeyCredential。

Elytron 中的 **expression=encryption** 资源使用加密表达式来解码它在运行时内的加密字符串。通过在配置文件中使用加密表达式而不是敏感字符串本身，您可以保护字符串的保密。加密表达式采用以下格式：

使用特定解析器的语法

```
${ENC::RESOLVER_NAME:ENCRYPTED_STRING}
```

ENC 是表示加密表达式的前缀。

RESOLVER_NAME 是解析程序用来解密加密的字符串。

示例

```
${ENC::initialresolver:RUxZAUMQE+L5zx9LmCRLyh5fjdfl1WM7lhfthKjeoEU+x+RMi6s=}
```

如果您使用默认解析器创建加密表达式，如下所示：

使用默认解析器的语法

```
${ENC::ENCRYPTED_STRING}
```

示例

```
${ENC::RUxZAUMQE+L5zx9LmCRLyh5fjdfl1WM7lhfthKjeoEU+x+RMi6s=}
```

在这种情况下，Elytron 使用您在 **expression=encryption** 资源中定义的默认解析器来解密表达式。您可以在支持它的任何资源属性中使用加密表达式。要找出某个属性是否支持加密表达式，请使用 **read-resource-description** 操作，例如：

有关 mail/mail-session的 read-resource-description 示例

```
/subsystem=mail/mail-session=*/:read-resource-description(recursive=true,access-control=none)
{
  "outcome"=>"success",
  "result"=>[{
  ...
    "from"=>{
      ...
      "expression-allowed"=>true,
      ...
   }]
}
```

在本例中，属性 **from** 支持加密表达式。这意味着，您可以通过加密表达式在 **from** 字段中隐藏您的电子邮件地址。

其他资源

- 在 Elytron 中创建加密的表达式

- **expression=encryption 属性**

4.1.7. 在 Elytron 中创建加密的表达式

从敏感字符串和 SecretKeyCredential 创建加密表达式。使用此加密表达式而不是管理模型中的敏感字符串 - 服务器配置文件，维护敏感字符串的保密性。

先决条件

- 您已在一些凭证存储中生成了 secret 密钥。
  有关在 **KeyStoreCredentialStore** 中创建 secret 密钥的信息，请参阅在 KeyStoreCredentialStore/credential-store 中生成 SecretKeyCredential

  有关在 PropertiesCredentialStore 中创建 secret 密钥的信息，请参阅在 **PropertiesCredentialStore** /secret-key-credential-store 中生成 SecretKey Credential

流程

1. 使用以下管理 CLI 命令，创建一个在凭证存储中引用现有 SecretKeyCredential 的别名的解析器：

   语法

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=[{name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-key=<secret_key_alias>}])
   ```

   示例

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=[{name=exampleResolver, credential-store=examplePropertiesCredentialStore, secret-key=key}])
   ```

   如果显示与重复资源相关的错误消息，请使用 **list-add** 操作而不是 **add**，如下所示：

   语法

   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers, value={name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-key=<secret_key_alias>})
   ```

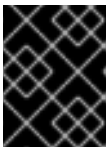
   示例

   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers,value={name=exampleResolver, credential-store=examplePropertiesCredentialStore, secret-key=key})
   {
       "outcome" => "success",
       "response-headers" => {
           "operation-requires-reload" => true,
           "process-state" => "reload-required"
       }
   }
   ```

2. 使用以下管理 CLI 命令重新载入服务器：

```
reload
```

3. 在管理 CLI 中禁用命令缓存：



重要

如果不禁用缓存，则可以访问管理 CLI 历史记录文件的任何人都可以看到 secret 密钥。

```
history --disable
```

4. 使用以下管理 CLI 命令创建一个加密表达式：

语法

```
/subsystem=elytron/expression=encryption:create-expression(resolver=<existing_resolver>,
clear-text=<sensitive_string_to_protect>)
```

示例

```
/subsystem=elytron/expression=encryption:create-expression(resolver=exampleResolver,
clear-text=TestPassword)
{
    "outcome" => "success",
    "result" => {"expression" =>
"${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurI=}"}
}
```

${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurI=} 是您在管理模型中使用的加密表达式，而不是在管理模型中使用 **TestPassword**。

如果您在不同的位置使用相同的纯文本，则每次使用加密的表达式而不是该位置中的纯文本前都会重复这个命令。当您为同一纯文本重复同一命令时，您可以获得同一键的不同结果，因为 Elytron 为每个调用使用唯一的初始化向量。

通过使用不同的加密表达式，您必须确保字符串中的一个加密表达式受到某种程度的破坏，用户无法发现任何其他加密表达式也可能会包含相同的字符串。

5. 使用以下管理 CLI 命令重新启用命令缓存：

```
history --enable
```

其他资源

- 使用加密表达式来保护 KeyStore/credential-store
- **expression=encryption** 属性

4.1.8. 在 JBoss EAP 配置中使用密码凭证

要引用存储在凭证存储中的密码或敏感字符串，请使用 JBoss EAP 配置中的 **credentials-reference** 属性。您可以使用 **credential-reference** 作为在 JBoss EAP 配置中大多数位置提供密码或其他敏感字符串的替代选择。

先决条件

- 您已将 PasswordCredential 添加到 KeyStoreStore。
  有关将密码Credential 添加到 KeyStoreStore 的详情，请参考 Adding PasswordCredential to a KeyStoreCredentialStore。

流程

- 在 **credential-reference** 属性中引用现有 KeyStore 和 PasswordCredential 的别名：

  语法

  ```
  credential-reference={store=<store_name>, alias=<alias>}
  ```

  示例

  ```
  data-source add --name=example_data_source --jndi-name=java:/example_data_source --
  driver-name=h2 --connection-url=jdbc:h2:mem:test;DB_CLOSE_DELAY=-
  1;DB_CLOSE_ON_EXIT=FALSE --user-name=db_user --credential-reference=
  {store=exampleKeyStoreCredentialStore, alias=passwordCredentialAlias}
  16:17:23,024 INFO  [org.jboss.as.connector.subsystems.datasources] (MSC service thread
  1-2) WFLYJCA0001: Bound data source [java:/example_data_source]
  ```

  在本例中，使用 KeyStoreCredentialStore **exampleKeyStoreCredentialStore** 中的别名为 **passwordCredentialAlias** 的一个已存在的 PasswordCredential，而不是使用明文形式的密码。这样可以包括数据库的密码。

其他资源

- 从外部来源获取凭据存储的密码。
- Elytron 中的凭证类型

4.1.9. 使用加密表达式来保护 KeyStore/credential-store

您可以使用加密表达式来保护 KeyStore。

先决条件

- 您已创建了加密的表达式。
  有关创建加密表达式的详情，请参考在 Elytron 中创建加密表达式。

流程

- 创建一个 KeyStoreCredentialStore，它使用加密表达式作为 **明文** ：

  语法

```
/subsystem=elytron/credential-
store=<name_of_credential_store>:add(path=<path_to_the_credential_store>, create=true,
modifiable=true, credential-reference={clear-text=<encrypted_expression>})
```

示例

```
/subsystem=elytron/credential-
store=secureKeyStoreCredentialStore:add(path="secureKeyStoreCredentialStore.jceks",
relative-to=jboss.server.data.dir, create=true, modifiable=true, credential-reference={clear-
text=${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2u
GurI=}})
{"outcome" => "success"}
```

其他资源

- **expression-encryption** 属性
- **credential-store** 属性

4.1.10. 自动更新凭证存储在凭证存储中

如果您有凭证存储，则不需要添加凭证或更新现有凭证，然后才能从凭证引用凭证引用它们。Elytron 实现此过程的自动化。配置凭证引用时，请同时指定 **store** 和 **clear-text** 属性。Elytron 会在 **store** 属性指定的凭证存储中自动添加或更新凭证。另外，您还可以指定 **alias** 属性。

Elytron 更新凭证存储，如下所示：

- 如果您指定了别名：
  - 如果存在别名的条目，现有凭证会使用指定的明文密码替换。
  - 如果别名的条目不存在，则使用指定别名添加新条目，以及明文密码。
- 如果您没有指定别名，Elytron 生成一个别名，并使用生成的别名和指定的明文密码添加新条目。

当凭证存储被更新时，**clear-text** 属性会从管理模型中删除。

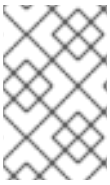
以下示例演示了如何创建用来指定 **store**, **clear-text**, 和 **alias** 属性的凭证引用：

```
/subsystem=elytron/key-store=exampleKS:add(relative-to=jboss.server.config.dir,
path=example.keystore, type=JCEKS, credential-reference=
{store=exampleKeyStoreCredentialStore, alias=myNewAlias, clear-text=myNewPassword})
{
    "outcome" => "success",
    "result" => {"credential-store-update" => {
        "status" => "new-entry-added",
        "new-alias" => "myNewAlias"
    }}
}
```

您可以使用以下命令将 **myNewAlias** 条目的凭证添加到之前定义的凭证存储中：

```
/subsystem=elytron/key-store=exampleKS:write-attribute(name=credential-reference.clear-
text,value=myUpdatedPassword)
```

```
{
    "outcome" => "success",
    "result" => {"credential-store-update" => {"status" => "existing-entry-updated"}},
    "response-headers" => {
        "operation-requires-reload" => true,
        "process-state" => "reload-required"
    }
}
```



注意

如果包含 **credentials-reference** 参数的操作失败，则不会发生自动凭证存储更新。

由 **credentials-reference** 属性指定的凭据存储不会改变。

4.1.11. 定义 FIPS 140-2 兼容凭证存储

您可以使用网络安全服务(NSS)数据库，或使用 Bouncy Castle 提供者来定义联邦信息处理标准(FIPS)140-2 兼容凭证存储。

4.1.11.1. 使用 NSS 数据库定义 FIPS 140-2 兼容凭证存储

要获得联邦信息处理标准(FIPS)兼容密钥存储，请使用 Sun PKCS#11（PKCS 代表公共密钥 Cryptography Standards）供应商访问网络安全服务(NSS)数据库。有关定义数据库的步骤，请参阅配置 NSS 数据库。

流程

1. 创建要在凭据存储中使用的 secret 密钥。

注意

要使 **keytool** 命令正常工作，在 **nss_pkcsll_fips.cfg** 文件中，您必须将 **nssDbMode** 属性分配为 **readWrite**。

```
$ keytool -keystore NONE -storetype PKCS11 -storepass <keystore_password> -genseckey
-alias <key_alias> -keyalg <key_algorithm> -keysize <key_size>
```

2. 创建外部凭据存储。外部凭据存储在 PKCS#11 密钥存储中包含一个 secret 密钥，并使用上一步中定义的别名访问此密钥存储。然后，该密钥存储用于解密 Java Cryptography Extension Keystore(JCEKS)密钥存储中的凭据。除了 **credential-store** 属性外，Elytron 使用 **credential-store KeyStore** 实现属性来配置外部凭据存储。

```
/subsystem=elytron/credential-store=<store_name>:add(modifiable=true, implementation-
properties={"keyStoreType"=>"PKCS11", "external"=>"true", "keyAlias"=>"<key_alias>",
externalPath="<path_to_JCEKS_file>"}, credential-reference={clear-
text="<keystore_password>"}, create=true)
```

3. 创建后，凭证存储可以正常存储别名。

```
/subsystem=elytron/credential-store=<store_name>:add-alias(alias="<alias>", secret-
value="<sensitive_string>")
```

4. 从凭证存储中读取，确认已成功添加别名。

```
/subsystem=elytron/credential-store=<store_name>:read-aliases()
```

其他资源

- 配置 NSS 数据库
- **credential-store** 属性
- **credential-store KeyStoreCredentialStore** 实现属性

4.1.11.2. 使用 Bouncy Castle 供应商定义 FIPS 140-2 兼容凭证存储

使用 Bouncy Castle 供应商定义联邦信息处理标准(FIPS)140-2 兼容凭证存储。

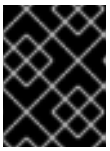
先决条件

- 确保您的环境已配置为使用 **BouncyCastle** 供应商。
  如需更多信息，请参阅配置 您的环境以使用 **BouncyCastle** Provider 。

流程

1. 创建要在凭据存储中使用的 secret 密钥。

```
$ keytool -genseckey -alias<key_alias> -keyalg <key_algorithm> -keysize <key_size> -
keystore <path_to_keystore> -storetype BCFKS -storepass <key_and_keystore_password> -
keypass <key_and_keystore_password>
```



重要

密钥存储的 **keypass** 和 **storepass** 必须相同，才能在 **elytron** 子系统中定义 FIPS 凭证存储。

2. 创建外部凭据存储。外部凭据存储在 BCFKS 密钥存储中保存机密密钥，并使用上一步中定义的别名访问此密钥存储。然后，该密钥存储用于解密 JCEKS 密钥存储中的凭据。**credential-store KeyStoreCredentialStore** implementation properties 用于配置外部凭据存储。

```
/subsystem=elytron/credential-store=<BCFKS_credential_store>:add(relative-
to=jboss.server.config.dir,credential-reference={clear-
text=<key_and_keystore_password>},implementation-properties=
{keyAlias=<key_alias>,external=true,externalPath=<path_to_credential_store>,keyStoreType
=BCFKS},create=true,location=<path_to_keystore>,modifiable=true)
```

3. 创建后，凭证存储可以正常存储别名。

```
/subsystem=elytron/credential-store=<BCFKS_credential_store>:add-alias(alias="<alias>",
secret-value="<sensitive_string>")
```

4. 从凭证存储中读取，确认已成功添加别名。

```
/subsystem=elytron/credential-store=<BCFKS_credential_store>:read-aliases()
```

其他资源

- **credential-store** 属性
- **credential-store KeyStoreCredentialStore** 实现属性

4.1.12. 使用凭证存储的自定义实现

使用凭据存储的自定义实现。

流程

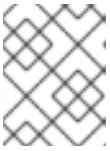
1. 创建一个扩展 Service Provider Interface(SPI) **CredentialStoreSpi** 抽象类的类。
2. 创建实施 Java Security **Provider** 的类。该提供程序必须将自定义凭据存储类添加为服务。
3. 创建包含凭据存储和提供程序类的模块，并将它添加到 JBoss EAP 中，并依赖于 **org.wildfly.security.elytron**。例如：

```
module add --name=org.jboss.customcredstore --
resources=/path/to/customcredstoreprovider.jar --dependencies=org.wildfly.security.elytron --
slot=main
```

4. 为您的供应商创建一个供应商加载程序。例如：

```
/subsystem=elytron/provider-loader=myCustomLoader:add(class-names=
[org.wildfly.security.mycustomcredstore.CustomElytronProvider],module=org.jboss.customcreds
tore)
```

5. 使用自定义实施创建凭据存储。



注意

确保指定正确的 **providers** 和 **type** 值。**type** 的值是供应商类中使用的，它为添加自定义凭据存储类添加为服务。

例如：

```
/subsystem=elytron/credential-
store=my_store:add(providers=myCustomLoader,type=CustomKeyStorePasswordStore,locatio
n="cred_stores/my_store.jceks",relative-to=jboss.server.data.dir,credential-reference={clear-
text=supersecretstorepassword},create=true)
```

或者，如果您创建了多个供应商，您可以使用 **other-providers** 的另一个供应商加载程序来指定额外的供应商。这可让您对新类型凭证有其他的实现。这些指定的其他供应商可以在自定义凭据存储 **初始化** 方法中自动访问，作为 **Provider[]** 参数。例如：

```
/subsystem=elytron/credential-store=my_store:add(providers=myCustomLoader,other-
providers=myCustomLoader2,type=CustomKeyStorePasswordStore,location="cred_stores/my_
store.jceks",relative-to=jboss.server.data.dir,credential-reference={clear-
text=supersecretstorepassword},create=true)
```

4.1.13. 从外部来源获取凭证存储的密码

您可以选择使用伪凭证存储来提供密码，而不必以明文格式提供密码。

您可以使用以下提供密码的选项：

EXT

使用 **java.lang.Runtime#exec(java.lang.String)** 的外部命令。您可以使用空格分隔的字符串列表为命令提供参数。外部命令引用来自操作系统的任何可执行文件，如 shell 脚本或可执行的二进制文件。Elytron 从您运行的命令的标准输出中读取密码。

示例

```
credential-reference={clear-text="{EXT}/usr/bin/getThePasswordScript.sh par1 par2",
type="COMMAND"}
```

CMD

使用 **java.lang.ProcessBuilder** 的外部命令.您可以使用以逗号分隔的字符串列表为命令提供参数。外部命令引用来自操作系统的任何可执行文件，如 shell 脚本或可执行的二进制文件。Elytron 从您运行的命令的标准输出中读取密码。

示例

```
credential-reference={clear-text="{CMD}/usr/bin/getThePasswordScript.sh par1,par2",
type="COMMAND"}
```

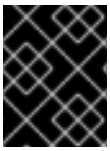
掩码

使用 PBE 或基于密码的加密屏蔽密码。它必须采用以下格式，其中包括 **SALT** 和 **ITERATION** 值：

```
credential-reference={clear-text="MASK-MASKED_VALUE;SALT;ITERATION"}
```

示例

```
credential-reference={clear-text="MASK-NqMznhSbL3lwRpDmyuqLBW==;12345678;123"}
```



重要

EXT、**CMD** 和 **MASK** 提供与提供外部密码的传统安全库样式的向后兼容性。对于 **MASK**，您必须使用以上格式，包括 **SALT** 和 **ITERATION** 值。

您还可以使用位于另一凭证存储中的密码作为新凭据存储的密码。

使用来自 Another Credential Store 的密码创建的凭证示例

```
/subsystem=elytron/credential-store=exampleCS:add(location="cred_stores/exampleCS.jceks",
relative-to=jboss.server.data.dir, create=true, credential-reference={store=cred-store, alias=pwd})
```

其他资源

- 为 JBoss EAP 提供初始密钥以解锁安全资源

4.1.14. 为 JBoss EAP 提供初始密钥以解锁安全资源

为安全起见，一些 JBoss EAP 组件会受 PasswordCredential in KeyStore 来保护。此 KeyStoreStore 由存储在 JBoss EAP 外部的 secret 密钥进行保护。这称为 *master* 密钥。JBoss EAP 在启动过程中 *使用这个* 主密钥来解锁 KeyStoreCredentialStore，以获取存储在 KeyStoreCredentialStore 中的 PasswordCredential。

您可以使用 Elytron 中的 PropertiesCredentialStore 来提供 *master* 密钥。另外，您还可以从外部来源获取 master 密钥或密码。有关从外部源获取密码的信息，请参阅 获取来自外部来源的凭据存储的密码。

4.1.14.1. 为独立服务器创建 PropertiesCredentialStore/secret-key-credential-store

使用管理 CLI 创建 PropertiesCredentialStore。在创建 PropertiesCredentialStore 时，JBoss EAP 默认生成 secret key。生成的密钥的名称是 **key**，其大小为 256 位。

先决条件

- 您至少提供对运行 JBoss EAP 的用户帐户的 PropertiesCredentialStore 的目录的读/写权限。

流程

- 使用以下命令，使用管理 CLI 创建 PropertiesCredentialStore：

  语法

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

  示例

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

其他资源

- PropertiesCredentialStore/secret-key-credential-store
- 使用 JBoss EAP 管理 CLI 的凭据存储操作
- **secret-key-credential-store** Attributes

4.1.14.2. 在 Elytron 中创建加密的表达式

从敏感字符串和 SecretKeyCredential 创建加密表达式。使用此加密表达式而不是管理模型中的敏感字符串 - 服务器配置文件，维护敏感字符串的保密性。

先决条件

- 您已在一些凭证存储中生成了 secret 密钥。
  有关在 **KeyStoreCredentialStore** 中创建 secret 密钥的信息，请参阅在 KeyStoreCredentialStore/credential-store 中生成 SecretKeyCredential

有关在 PropertiesCredentialStore 中创建 secret 密钥的信息，请参阅在 **PropertiesCredentialStore** /secret-key-credential-store 中生成 SecretKey Credential

流程

1. 使用以下管理 CLI 命令，创建一个在凭证存储中引用现有 SecretKeyCredential 的别名的解析器：

   语法

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=[{name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-key=<secret_key_alias>}])
   ```

   示例

   ```
   /subsystem=elytron/expression=encryption:add(resolvers=[{name=exampleResolver, credential-store=examplePropertiesCredentialStore, secret-key=key}])
   ```

   如果显示与重复资源相关的错误消息，请使用 **list-add** 操作而不是 **add**，如下所示：

   语法

   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers, value={name=<name_of_the_resolver>, credential-store=<name_of_credential_store>, secret-key=<secret_key_alias>})
   ```

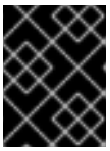
   示例

   ```
   /subsystem=elytron/expression=encryption:list-add(name=resolvers,value={name=exampleResolver, credential-store=examplePropertiesCredentialStore, secret-key=key})
   {
       "outcome" => "success",
       "response-headers" => {
           "operation-requires-reload" => true,
           "process-state" => "reload-required"
       }
   }
   ```

2. 使用以下管理 CLI 命令重新载入服务器：

   ```
   reload
   ```

3. 在管理 CLI 中禁用命令缓存：

   重要

   如果不禁用缓存，则可以访问管理 CLI 历史记录文件的任何人都可以看到 secret 密钥。

   ```
   history --disable
   ```

4. 使用以下管理 CLI 命令创建一个加密表达式：

语法

```
/subsystem=elytron/expression=encryption:create-expression(resolver=<existing_resolver>,
clear-text=<sensitive_string_to_protect>)
```

示例

```
/subsystem=elytron/expression=encryption:create-expression(resolver=exampleResolver,
clear-text=TestPassword)
{
    "outcome" => "success",
    "result" => {"expression" =>
"${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurI=}"}
}
```

${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurI=} 是您在管理模型中使用的加密表达式，而不是在管理模型中使用 **TestPassword**。

如果您在不同的位置使用相同的纯文本，则每次使用加密的表达式而不是该位置中的纯文本前都会重复这个命令。当您为同一纯文本重复同一命令时，您可以获得同一键的不同结果，因为 Elytron 为每个调用使用唯一的初始化向量。

通过使用不同的加密表达式，您必须确保字符串中的一个加密表达式受到某种程度的破坏，用户无法发现任何其他加密表达式也可能会包含相同的字符串。

5. 使用以下管理 CLI 命令重新启用命令缓存：

```
history --enable
```

其他资源

- 使用加密表达式来保护 KeyStore/credential-store
- **expression=encryption** 属性

4.1.14.3. 使用加密表达式来保护 KeyStore/credential-store

您可以使用加密表达式来保护 KeyStore。

先决条件

- 您已创建了加密的表达式。
  有关创建加密表达式的详情，请参考在 Elytron 中创建加密表达式。

流程

- 创建一个 KeyStoreCredentialStore，它使用加密表达式作为 **明文** ：

语法

```
/subsystem=elytron/credential-store=<name_of_credential_store>:add(path=<path_to_the_credential_store>, create=true, modifiable=true, credential-reference={clear-text=<encrypted_expression>})
```

示例

```
/subsystem=elytron/credential-store=secureKeyStoreCredentialStore:add(path="secureKeyStoreCredentialStore.jceks", relative-to=jboss.server.data.dir, create=true, modifiable=true, credential-reference={clear-text=${ENC::exampleResolver:RUxZAUMQgtpG7oFlHR2j1Gkn3GKlHff+HR8GcMX1QXHvx2uGurl=}})
{"outcome" => "success"}
```

其他资源

- **expression-encryption** 属性
- **credential-store** 属性

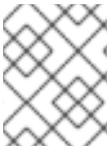
使用加密表达式保护 KeyStoreCredentialStore 后，您可以在 KeyStoreCredentialStore 中生成 **SecretKeyCredential**，并使用 secret 密钥来创建另一个加密表达式。然后，您可以使用这个新加密表达式而不是管理模型中的敏感字符串 - 服务器配置文件。您可以创建整个凭证存储链以提高安全性。这样链可以更难以猜测敏感字符串，因为字符串会按照以下方式进行保护：

- 第一个加密表达式为 KeyStoreCredentialStore 的安全。
- 另一个加密表达式保护敏感字符串。
- 要解码敏感字符串，您需要解密加密的表达式。

随着加密表达式的链变得较长，解密敏感字符串会变得比较困难。

4.1.15. 将密码库转换为凭证存储

您可以使用 WildFly Elytron 工具将密码 vault 转换为凭据存储。要将密码库转换为凭据存储，您需要在初始化 vault 时使用的 vault 值。



注意

在转换密码 vault 时，新凭证存储中的别名根据对应的密码 vault 块和属性名称命名：::AULT _***BLOCK ::ATTRIB*** UTE_***NAME***。

4.1.15.1. 使用 WildFly Elytron 工具将单个密码库转换为凭据存储

使用 WildFly Elytron 工具将单个密码 vault 转换为凭据存储。

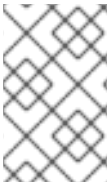
流程

- 使用以下命令将密码 vault 转换为凭证存储：

```
$ EAP_HOME/bin/elytron-tool.sh vault --keystore "<path_to_vault_file>" --keystore-password <vault_password> --enc-dir "<path_to_vault_directory>" --salt <salt> --iteration <iteration_count> --alias <vault_alias>
```

例如，您也可以使用 **--location** 参数指定新凭证存储的文件名和位置：

```
$ EAP_HOME/bin/elytron-tool.sh vault --keystore ../vaults/vault.keystore --keystore-password vault22 --enc-dir ../vaults/ --salt 1234abcd --iteration 120 --alias my_vault --location ../cred_stores/my_vault_converted.cred_store
```



注意

您还可以使用 **--summary** 参数显示用于转换的管理 CLI 命令的摘要。请注意，即使使用了纯文本密码，它也会在摘要输出中屏蔽。使用默认的 **salt** 和 **迭代** 值，除非在命令中指定。

4.1.15.2. 使用 WildFly Elytron 工具批量将密码 vault 转换为凭证存储

批量将多个密码库转换为凭据存储。

流程

1. 将您要转换为描述文件的库的详情采用以下格式：

```
keystore:<path_to_vault_file>
keystore-password:<vault_password>
enc-dir:<path_to_vault_directory>
salt:<salt> (1)
iteration:<iteration_count>
location:<path_to_converted_cred_store> (2)
alias:<vault_alias>
properties:<parameter1>=<value1>;<parameter2>=<value2>; (3)
```

(1) 如果您为 vault 提供纯文本密码，则可以省略 **salt** 和 **iteration**。

(2) 指定转换凭证存储的位置和文件名。

(3) 可选：指定以分号(;)分隔的可选参数列表。如需可用参数的列表，请参阅 ***EAP_HOME*/bin/elytron-tool.sh vault --help**。

例如：

```
keystore:/vaults/vault1/vault1.keystore
keystore-password:vault11
enc-dir:/vaults/vault1/
salt:1234abcd
iteration:120
location:/cred_stores/vault1_converted.cred_store
alias:my_vault

keystore:/vaults/vault2/vault2.keystore
keystore-password:vault22
enc-dir:/vaults/vault2/
salt:abcd1234
iteration:130
location:/cred_stores/vault2_converted.cred_store
alias:my_vault2
```

2. 使用上一步中的描述文件运行批量转换命令：

```
$ EAP_HOME/bin/elytron-tool.sh vault --bulk-convert vaultdescriptions.txt
```

如需更多信息，请使用 ***EAP_HOME*/bin/elytron-tool.sh vault --help** 命令来获取可用选项的详细列表。

4.1.16. 使用 Elytron 客户端的凭证存储示例

连接到 JBoss EAP（如 Jakarta Enterprise Bean）的客户端可以使用 Elytron 客户端进行身份验证。无法访问正在运行的 JBoss EAP 服务器的用户可以使用 WildFly Elytron 工具创建和修改凭据存储，然后客户端可以使用 Elytron 客户端访问凭据存储中的敏感字符串。

以下示例演示了如何在 Elytron 客户端配置文件中使用凭据存储。

带有 Credential Store 的 custom-config.xml 示例

```
<configuration>
  <authentication-client xmlns="urn:elytron:client:1.2">
    ...
    <credential-stores>
      <credential-store name="my_store"> 1
        <protection-parameter-credentials>
          <credential-store-reference clear-text="pass123"/> 2
        </protection-parameter-credentials>
        <attributes>
          <attribute name="location" value="/path/to/my_store.jceks"/> 3
        </attributes>
      </credential-store>
    </credential-stores>
    ...
    <authentication-configurations>
      <configuration name="my_user">
        <set-host name="localhost"/>
        <set-user-name name="my_user"/>
        <set-mechanism-realm name="ManagementRealm"/>
        <use-provider-sasl-factory/>
        <credentials>
          <credential-store-reference store="my_store" alias="my_user"/> 4
        </credentials>
      </configuration>
    </authentication-configurations>
    ...
  </authentication-client>
</configuration>
```

1. 用于在 Elytron 客户端配置文件中使用的凭证存储名称。
2. 凭据存储的密码。
3. 凭证存储文件的路径。
4. 存储在凭据存储中的敏感字符串的凭据引用。

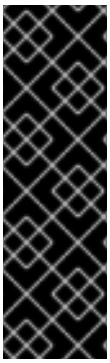
其他资源

- 使用 Elytron Client 配置客户端
- 通过 WildFly Elytron 工具创建和修改凭据存储。

4.2. 密码 VAULT

配置 JBoss EAP 和相关应用需要潜在的敏感信息，如用户名和密码。密码不以纯文本形式存储在配置文件中，可以使用密码 vault 功能来屏蔽密码信息并将其存储在加密的密钥存储中。密码存储之后，可将引用包含在管理 CLI 命令中或部署到 JBoss EAP 的应用中。

密码库使用 Java 密钥存储作为其存储机制。密码库由两个部分组成：存储和密钥存储。Java 密钥存储用于存储密钥，该密钥用于在 Vault 存储中加密或解密敏感字符串。



重要

此步骤中使用了由 Java Runtime Environment(JRE)提供的 keytool 程序。找到 Red Hat Enterprise Linux 上文件的路径，它是 **/usr/bin/keytool**。

JCEKS 密钥存储实施与 Java 供应商之间的不同，因此必须使用与所使用的 JDK 供应商中的 keytool 实用程序生成密钥存储。使用在 JDK 上运行的 JBoss EAP 7 实例中一个供应商的 JDK 生成的密钥存储会导致以下异常： **java.io.IOException: com.sun.crypto.provider.provider.provider.provider.SealedObjectForKeyProtectorc tor**

4.2.1. 设置密码 Vault

按照以下步骤设置和使用密码 Vault。

1. 创建用于存储密钥存储和其他加密信息的目录。
   此流程的其余部分假定目录是 ***EAP_HOME*/vault/**。由于该目录将包含敏感信息，因此只能被有限用户访问。至少运行 JBoss EAP 的用户帐户需要读写访问。

2. 确定要与 keytool 程序一起使用的参数。
   决定以下参数的值：

   alias

   别名是密码库或密钥存储中存储的其他数据的唯一标识符。别名不区分大小写。

   storetype

   storetype 指定密钥存储类型。建议使用 **jceks** 值。

   keyalg

   用于加密的算法。使用 JRE 和操作系统的文档来查看哪些其它选择。

   keysize

   加密密钥的大小会影响通过 brute 解密的难度。有关适当的值的信息，请查看使用 keytool 程序发布的文档。

   storepass

   storepass 的值是用于向密钥存储进行身份验证的密码，以便可以读取该密钥。密码必须至少为 6 个字符，且必须在访问密钥存储时提供。如果省略此参数，则执行该命令后将提示输入 keytool 程序。

   keypass

keypass 的值是用于访问特定密钥的密码，且必须与 storepass 参数的值匹配。

有效期

有效期的值是密钥有效的句点（以天为单位）。

keystore

密钥存储的值是要存储密钥存储的值的文件路径和文件名。先将数据添加到它时创建密钥存储文件。确定使用正确的文件路径分隔符：用于 Red Hat Enterprise Linux 和类似操作系统的 /（正斜杠），\（反斜杠）用于 Windows Server。
keytool 程序有许多其他选项。如需了解更多详细信息，请参阅 JRE 或操作系统的文档。

3. 运行 keytool 命令，确保 **keypass** 和 **storepass** 包含相同的值。

```
$ keytool -genseckey -alias vault -storetype jceks -keyalg AES -keysize 128 -storepass vault22 -keypass vault22 -keystore EAP_HOME/vault/vault.keystore
```

这将生成已在 ***EAP_HOME*/vault/vault.keystore** 中创建的密钥存储。它存储一个密钥，其别名为 vault，它将用于存储加密字符串（如密码）以用于 JBoss EAP。

4.2.2. 初始化密码 Vault

密码 vault 可以以交互方式初始化，其中会提示您每个参数的值，或者非交互方式进行，其中所有参数值均能在命令行中提供。每种方法都会出现相同的结果，因此可以使用其中任一个。

需要以下参数：

Keystore URL(KEYSTORE_URL)

密钥存储文件的文件系统路径或 URI。示例使用 ***EAP_HOME*/vault/vault.keystore**。

keystore password (KEYSTORE_PASSWORD)

用于访问密钥存储的密码。

Salt (SALT)

salt 值是一个随机字符串，它使用八个字符以及迭代计数来加密密钥存储的内容。

keystore Alias (KEYSTORE_ALIAS)

已知密钥存储的别名。

Iteration Count (ITERATION_COUNT)

加密算法运行的次数。

用于存储加密文件的目录(ENC_FILE_DIR)

要存储加密的文件的路径。这通常是包含密码库的目录。它很方便，但不要强制将所有加密信息存储在与密钥存储相同的位置。该目录应只能被限制用户访问。至少运行 JBoss EAP 7 的用户帐户需要读写访问。该密钥存储应位于您设置密码 vault 时创建的目录中。请注意，需要在目录名上末尾的反斜杠或正斜杠。确定使用正确的文件路径分隔符：用于 Red Hat Enterprise Linux 和类似操作系统的 /（正斜杠），\（反斜杠）用于 Windows Server。

Vault Block(VAULT_BLOCK)

在密码库中提供给此块的名称。

属性(ATTRIBUTE)

要给所存储属性的名称。

安全属性(SEC-ATTR)

存储在密码库中的密码。

要以非交互方式运行密码库命令，***EAP_HOME*/bin/** 中的 **vault** 脚本可使用相关信息的参数来调用：

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD --alias
KEYSTORE_ALIAS --vault-block VAULT_BLOCK --attribute ATTRIBUTE --sec-attr SEC-ATTR --
enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT --salt SALT
```

示例：初始化密码 Vault

```
$ vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password vault22 --alias vault --
vault-block vb --attribute password --sec-attr 0penS3sam3 --enc-dir EAP_HOME/vault/ --iteration 120
--salt 1234abcd
```

示例：输出

```
=========================================================================

  JBoss Vault

  JBOSS_HOME: EAP_HOME

  JAVA: java

=========================================================================

Nov 09, 2015 9:02:47 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX00361: Default Security Vault Implementation Initialized and Ready
WFLYSEC0047: Secured attribute value has been stored in Vault.
Please make note of the following:
********************************************
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
VAULT::vb::password::1
********************************************
WFLYSEC0048: Vault Configuration in WildFly configuration file:
********************************************

</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
********************************************
```

要以互动方式运行 vault 命令，需要执行以下步骤：

1. 以交互方式启动 password vault 命令。
   在 Red Hat Enterprise Linux 和类似操作系统或*EAP_HOME*\bin **vault.bat 上运行 EAP_HOME /bin/vault. sh**。键入 **0**（零）启动新的交互式会话。

2. 完成提示参数。
   按照提示输入所需的参数。

3. 记录已屏蔽的密码信息。
   屏蔽的密码、salt 和迭代数将打印到标准输出。在安全的位置记录它们。它们需要将条目添加到 Password Vault 中。访问密钥存储文件，这些值可能允许攻击者获得密码 Vault 中敏感信息的访问权限。

4. 退出交互式控制台
   键入 **2**（两）以退出交互式控制台。

示例：输入和输出

```
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password: vault22
Enter Keystore password again: vault22
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in AS7 config file:
********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
********************************************
Vault is initialized and ready for use
Handshake with Vault complete
```

+ 密钥存储密码已被屏蔽，以用于配置文件和部署。此外，库会被初始化并可供使用。

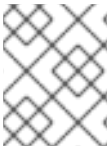
4.2.3. 使用密码 Vault

在可以屏蔽并使用密码和其他敏感属性前，必须先了解 JBoss EAP 7 存储和解密它们的密码库。

以下命令可将 JBoss EAP 7 配置为使用密码 vault：

```
/core-service=vault:add(vault-options=[("KEYSTORE_URL" => PATH_TO_KEYSTORE),
("KEYSTORE_PASSWORD" => MASKED_PASSWORD),("KEYSTORE_ALIAS" => ALIAS),("SALT"
=> SALT),("ITERATION_COUNT" => ITERATION_COUNT),("ENC_FILE_DIR" => ENC_FILE_DIR)])
```

```
/core-service=vault:add(vault-options=[("KEYSTORE_URL" => "EAP_HOME/vault/vault.keystore"),
("KEYSTORE_PASSWORD" => "MASK-5dOaAVafCSd"),("KEYSTORE_ALIAS" => "vault"),("SALT"
=> "1234abcd"),("ITERATION_COUNT" => "120"),("ENC_FILE_DIR" => "EAP_HOME/vault/")])
```



注意

如果使用 Microsoft Windows Server，请在文件路径中使用两个反斜杠(\\)，而不是使用一个。例如： **C:\\data\\vault\\vault.keystore**。这是因为单个反斜杠字符(\)用于字符转义。

4.2.4. 在 Password Vault 中存储敏感字符串

在纯文本配置文件中包括密码和其他敏感字符串存在安全风险。将这些字符串保存在 Password Vault 中以提高安全性，然后可以在配置文件中引用它们，使用其屏蔽的形式管理 CLI 命令和应用程序。

敏感字符串可以以互动方式存储在 Password Vault 中，其中工具会提示每个参数的值，或者非交互地在命令行中提供所有参数值。每种方法都会出现相同的结果，因此可以使用其中任一个。这两种方法均可使用 **vault** 脚本调用。

要以非交互方式运行 vault 命令，**vault** 脚本（位于 ***EAP_HOME*/bin/**中）可以使用相关信息的参数来调用：

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD --alias
KEYSTORE_ALIAS --vault-block VAULT_BLOCK --attribute ATTRIBUTE --sec-attr SEC-ATTR --
enc-dir ENC_FILE_DIR --iteration ITERATION_COUNT --salt SALT
```

注意

密钥存储密码必须以纯文本形式提供，而不是屏蔽的格式。

```
$ vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password vault22 --alias vault --
vault-block vb --attribute password --sec-attr 0penS3sam3 --enc-dir EAP_HOME/vault/ --iteration 120
--salt 1234abcd
```

示例：输出

```
=========================================================================

  JBoss Vault

  JBOSS_HOME: EAP_HOME

  JAVA: java

=========================================================================

Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX00361: Default Security Vault Implementation Initialized and Ready
WFLYSEC0047: Secured attribute value has been stored in Vault.
Please make note of the following:
********************************************
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
```

```
VAULT::vb::password::1
************************************************
WFLYSEC0048: Vault Configuration in WildFly configuration file:
************************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="../vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="../vault/"/>
</vault><management> ...
************************************************
```

在调用 **vault** 脚本后，消息会在标准输出中打印到标准输出，显示 vault 块、属性名称、屏蔽字符串以及配置中有关字符串的建议。在安全的位置记录此信息。示例输出的提取如下：

```
Vault Block:vb
Attribute Name:password
Configuration should be done as follows:
VAULT::vb::password::1
```

要以互动方式运行 vault 命令，需要执行以下步骤：

1. 以交互方式启动 Password Vault 命令。
   启动操作系统命令行界面，并运行 ***EAP_HOME*/bin/vault.sh** （在 Red Hat Enterprise Linux 和类似操作系统）或 ***EAP_HOME*\bin\vault.bat** （在 Microsoft Windows Server 上）。键入 **0**（零）启动新的交互式会话。

2. 完成提示参数。
   按照提示输入所需的参数。这些值必须与创建密码 Vault 时提供的匹配。

   注意

   密钥存储密码必须以纯文本形式提供，而不是屏蔽的格式。

3. 完成有关敏感字符串的提示参数。
   输入 **0**（零）开始存储敏感字符串。按照提示输入所需的参数。

4. 记录有关已屏蔽字符串的信息。
   条消息将打印到标准输出，显示 vault 块、属性名称、屏蔽字符串以及配置中使用字符串的建议。在安全的位置记录此信息。示例输出的提取如下：

   ```
   Vault Block:ds_Example1
   Attribute Name:password
   Configuration should be done as follows:
   VAULT::ds_Example1::password::1
   ```

5. 退出交互式控制台。
   键入 **2**（两）退出交互式控制台。

示例：输入和输出

```
 ===========================================================================
  JBoss Vault
  JBOSS_HOME: EAP_HOME
  JAVA: java
 ===========================================================================
 *********************************
 ****  JBoss Vault  ***************
 *********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in AS7 config file:
 ********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
 ********************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::  0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
0
Task: Store a secured attribute
Please enter secured attribute value (such as password):
Please enter secured attribute value (such as password) again:
Values match
Enter Vault Block:ds_Example1
Enter Attribute Name:password
Secured attribute value has been stored in vault.
Please make note of the following:
 ********************************************
Vault Block:ds_Example1
Attribute Name:password
Configuration should be done as follows:
VAULT::ds_Example1::password::1
 ********************************************
Please enter a Digit::  0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
```

4.2.5. 在配置中使用加密敏感字符串

任何已加密的敏感字符串都可以以屏蔽的形式用于配置文件或管理 CLI 命令，从而提供表达式。

要确认特定子系统内是否允许表达式，请对该子系统运行以下管理 CLI 命令：

```
/subsystem=SUBSYSTEM:read-resource-description(recursive=true)
```

在运行此命令的输出中，查找 expressions **-allowed 参数的值**。如果这是 **true**，那么可在此子系统配置中使用表达式。

使用以下语法将任何纯文本字符串替换为掩码形式。

```
${VAULT::VAULT_BLOCK::ATTRIBUTE_NAME::MASKED_STRING}
```

示例：使用 Masked Form 中的密码的数据源定义

```
...
  <subsystem xmlns="urn:jboss:domain:datasources:5.0">
    <datasources>
      <datasource jndi-name="java:jboss/datasources/ExampleDS" enabled="true" use-java-
context="true" pool-name="H2DS">
        <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-1</connection-url>
        <driver>h2</driver>
        <pool></pool>
        <security>
          <user-name>sa</user-name>
          <password>${VAULT::ds_ExampleDS::password::1}</password>
        </security>
      </datasource>
      <drivers>
        <driver name="h2" module="com.h2database.h2">
          <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
        </driver>
      </drivers>
    </datasources>
  </subsystem>
...
```

4.2.6. 在应用程序中使用加密敏感字符串

存储在密码 vault 中的加密字符串可以在应用程序的源代码中使用。以下示例是 servlet 的源代码的提取，说明在数据源定义中使用掩码的密码，而不是纯文本密码。纯文本版本已注释掉，以便您看到不同的差异。

示例：使用 Vault 密码的 Servlet

```
@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "VAULT::DS::thePass::1",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)
```

```
/*old (plaintext) definition
@DataSourceDefinition(
        name = "java:jboss/datasources/LoginDS",
        user = "sa",
        password = "sa",
        className = "org.h2.jdbcx.JdbcDataSource",
        url = "jdbc:h2:tcp://localhost/mem:test"
)*/
```

4.2.7. 检查敏感字符串是否在 Password Vault 中

在尝试存储或使用密码 Vault 中的敏感字符串之前，请先确认它是否已存储。

此检查可以交互方式完成，其中提示用户输入每个参数的值，或者非交互方式进行，其中所有参数的值都在命令行中都提供了。每种方法都会出现相同的结果，因此可以使用其中任一个。这两种方法均可使用 **vault** 脚本调用。

使用非操作方法一次性提供所有参数的值。有关所有参数的描述，请参阅 Initialize the Password Vault。要以非交互方式运行密码库命令，***EAP_HOME*/bin/** 中的 **vault** 脚本可使用相关信息的参数来调用：

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD --alias
KEYSTORE_ALIAS --check-sec-attr --vault-block VAULT_BLOCK --attribute ATTRIBUTE --enc-dir
ENC_FILE_DIR --iteration ITERATION_COUNT --salt SALT
```

使用实际值替换占位符值。*KEYSTORE_URL*、*KEYSTORE_PASSWORD* 和 *KEYSTORE_ALIAS* 的值必须与创建密码 vault 时提供的值匹配。

注意

密钥存储密码必须以纯文本形式提供，而不是屏蔽的格式。

如果敏感字符串存储在指定 vault 块中，则会显示以下消息：

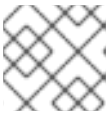
```
Password already exists.
```

如果值没有存储在指定块中，则会显示以下信息：

```
Password doesn't exist.
```

要以互动方式运行 vault 命令，需要执行以下步骤：

1. 以交互方式启动 password vault 命令。
   运行 ***EAP_HOME*/bin/vault.sh** （在 Red Hat Enterprise Linux 和类似操作系统）或 ***EAP_HOME*\bin\vault.bat** （在 Windows Server 上）。键入 **0**（零）启动新的交互式会话。
2. 完成提示参数。按照提示输入所需的身份验证参数。这些值必须与创建密码库时提供的匹配。



注意

提示输入身份验证时，密钥存储密码必须以纯文本形式提供，而不是屏蔽的形式。

- 输入 **1**（一） 选择检查**是否存在安全属性**。

- 输入存储敏感字符串的 vault 块的名称。
- 输入要检查的敏感字符串的名称。

如果敏感字符串存储在指定 vault 块中，类似以下内容的确认消息将输出：

> A value exists for (VAULT_BLOCK, ATTRIBUTE)

如果敏感字符串不在指定块中，类似以下内容的消息将输出如下：

> No value has been store for (VAULT_BLOCK, ATTRIBUTE)

示例：检查敏感字符串以主动的形式

```
 ========================================================================
  JBoss Vault
  JBOSS_HOME: EAP_HOME
  JAVA: java
 ========================================================================
 *********************************
 ****  JBoss Vault  ***************
 *********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Nov 09, 2015 9:24:36 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in AS7 config file:
 ********************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
 ********************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::  0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
1
Task: Verify whether a secured attribute exists
```

```
Enter Vault Block:vb
Enter Attribute Name:password
A value exists for (vb, password)
Please enter a Digit::  0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
```

4.2.8. 从 Password Vault 中删除敏感字符串

出于安全考虑，最好在不再需要密码 Vault 时从密码 Vault 中删除敏感字符串。例如，如果应用程序被停用，在数据源定义中使用的任何敏感字符串都应同时删除。

重要

作为先决条件，请先从密码 Vault 中删除敏感字符串，请确认它在 JBoss EAP 的配置中使用。

此操作可以交互方式完成，其中提示用户输入每个参数的值，或者非交互方式进行，其中所有参数的值都会在命令行中提供。每种方法都会出现相同的结果，因此可以使用其中任一个。这两种方法均可使用 **vault** 脚本调用。

使用非操作方法一次性提供所有参数的值。有关所有参数的描述，请参阅 Initialize the Password Vault。要以非交互方式运行 vault 命令，**vault** 脚本（位于 ***EAP_HOME*/bin/**中）可以使用相关信息的参数来调用：

```
$ vault.sh --keystore KEYSTORE_URL --keystore-password KEYSTORE_PASSWORD --alias
KEYSTORE_ALIAS --remove-sec-attr --vault-block VAULT_BLOCK --attribute ATTRIBUTE --enc-dir
ENC_FILE_DIR --iteration ITERATION_COUNT --salt SALT
```

使用实际值替换占位符值。*KEYSTORE_URL*、*KEYSTORE_PASSWORD* 和 *KEYSTORE_ALIAS* 的值必须与创建密码 vault 时提供的值匹配。

注意

密钥存储密码必须以纯文本形式提供，而不是屏蔽的格式。

如果成功删除了敏感字符串，则会显示类似如下的确认信息：

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] has been successfully removed from vault
```

如果没有删除敏感字符串，则会显示类似如下的消息：

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] was not removed from vault, check whether it exist
```

示例：输出

```
$ ./vault.sh --keystore EAP_HOME/vault/vault.keystore --keystore-password vault22 --alias vault --
remove-sec-attr --vault-block vb --attribute password --enc-dir EAP_HOME/vault/ --iteration 120 --salt
1234abcd
 ========================================================================
  JBoss Vault
  JBOSS_HOME: EAP_HOME
  JAVA: java
```

```
 ========================================================================
Dec 23, 2015 1:54:24 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Secured attribute [vb::password] has been successfully removed from vault
```

以交互方式删除敏感 String
要以互动方式运行密码 vault 命令，需要执行以下步骤：

1. 以交互方式启动 password vault 命令。
   运行 ***EAP_HOME*/bin/vault.sh** （在红帽企业 Linux 和类似操作系统上）或 ***EAP_HOME*\bin\vault.bat** （在 Microsoft Windows Server 上）。键入 **0**（零）启动新的交互式会话。

2. 完成提示的参数。
   按照提示输入所需的身份验证参数。这些值必须与创建密码库时提供的值匹配。

注意

提示进行身份验证时，必须以纯文本形式提供密钥存储密码，而不是屏蔽的格式。

- 输入 **2**（两），选择删除保护的属性。
- 输入存储敏感字符串的 vault 块的名称。
- 输入要删除的敏感字符串的名称。

如果成功删除了敏感字符串，则会显示类似如下的确认信息：

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] has been successfully removed from vault
```

如果没有删除敏感字符串，则会显示类似如下的消息：

```
Secured attribute [VAULT_BLOCK::ATTRIBUTE] was not removed from vault, check whether it exist
```

示例：输出

```
 *********************************
 ****  JBoss Vault  ***************
 *********************************
Please enter a Digit::   0: Start Interactive Session  1: Remove Interactive Session  2: Exit
0
Starting an interactive session
Enter directory to store encrypted files:EAP_HOME/vault/
Enter Keystore URL:EAP_HOME/vault/vault.keystore
Enter Keystore password:
Enter Keystore password again:
Values match
Enter 8 character salt:1234abcd
Enter iteration count as a number (Eg: 44):120
Enter Keystore Alias:vault
Initializing Vault
Dec 23, 2014 1:40:56 PM org.picketbox.plugins.vault.PicketBoxSecurityVault init
INFO: PBOX000361: Default Security Vault Implementation Initialized and Ready
Vault Configuration in configuration file:
```

```
 ************************************************
...
</extensions>
<vault>
  <vault-option name="KEYSTORE_URL" value="EAP_HOME/vault/vault.keystore"/>
  <vault-option name="KEYSTORE_PASSWORD" value="MASK-5dOaAVafCSd"/>
  <vault-option name="KEYSTORE_ALIAS" value="vault"/>
  <vault-option name="SALT" value="1234abcd"/>
  <vault-option name="ITERATION_COUNT" value="120"/>
  <vault-option name="ENC_FILE_DIR" value="EAP_HOME/vault/"/>
</vault><management> ...
 ************************************************
Vault is initialized and ready for use
Handshake with Vault complete
Please enter a Digit::  0: Store a secured attribute  1: Check whether a secured attribute exists  2:
Remove secured attribute  3: Exit
2
Task: Remove secured attribute
Enter Vault Block:vb
Enter Attribute Name:password
Secured attribute [vb::password] has been successfully removed from vault
```

4.2.9. 配置 Red Hat JBoss Enterprise Application Platform 平台，以使用密码 Vault 的自定义实施

除了使用提供的密码库实施外，还可以使用对 **SecurityVault** 的自定义实施。

重要

作为前提条件，请确保已初始化密码库。如需更多信息，请参阅 Initialize the Password Vault。

对密码库使用自定义实施：

1. 创建一个实施接口 **SecurityVault** 的类。
2. 创建一个包含上一步中的类的模块，并指定接口为 **SecurityVault** 的 **org.picketbox** 的依赖关系。
3. 通过添加 vault 元素及以下属性，在 JBoss EAP 配置中启用自定义密码库：
   - **code** - 实施 **SecurityVault** 的全限定类名称。
   - **module** - 包含自定义类的模块名称。

（可选） **vault-options** 参数可用于初始化密码库的自定义类。

示例：使用 vault-options 参数初始化自定义类

```
/core-service=vault:add(code="custom.vault.implementation.CustomSecurityVault",
module="custom.vault.module", vault-options=[("KEYSTORE_URL" => PATH_TO_KEYSTORE),
("KEYSTORE_PASSWORD" => MASKED_PASSWORD), ("KEYSTORE_ALIAS" => ALIAS),("SALT"
=> SALT),("ITERATION_COUNT" => ITERATION_COUNT),("ENC_FILE_DIR" => ENC_FILE_DIR)])
```

4.2.10. 从外部源获取密钥存储密码

可在 vault 配置中使用 **EXT**, **EXTC**, **CMD**, **CMDC** 或 **CLASS** 方法来获取 Java 密钥存储密码。

```
<vault-option name="KEYSTORE_PASSWORD" value="METHOD_TO_OBTAIN_PASSWORD"/>
```

方法的描述列为：

{EXT}...

指的是确切的命令，其中 **...** 是确切的命令。例如： **{EXT}/usr/bin/getmypassword --section 1 --query company** 运行 **/usr/bin/getmypassword** 命令，该命令可在标准输出上显示密码，并将它用作 Security Vault 的密钥存储的密码。在本例中，命令使用两个选项： **第 1 部分** 和 **--query company**。

{EXTC[:expiration_in_millis]}...

指的是确切的命令，其中 **...** 是传递给 **Runtime.exec(String)** 方法的确切命令行，以执行平台命令。命令输出的第一行用作密码。EXTC 变体缓存 **expiration_in_millis** 毫秒的密码。默认缓存到期时间为 **0 = infinity**。例如： **{EXTC:120000}/usr/bin/getmypassword --section 1 --query company** 验证缓存是否包含 **/usr/bin/getmypassword** 输出，如果它包含输出，然后将其使用。如果它不包含输出，请运行 命令将其输出到缓存并使用它。在本例中，缓存在 2 分钟后过期，该缓存为 120000 毫秒。

{CMD}... or {CMDC[:expiration_in_millis]}...

常规命令是用 **(** comma)分隔的字符串，其中第一部分是实际命令，而其他部分则表示参数。逗号可以反斜杠，使它成为 参数的一部分。例如： **{CMD}/usr/bin/getmypassword,--section,1,--query,company**。

{CLASS[@jboss_module_spec]}classname[:ctorargs]

其中 **[:ctorargs]** 是用以下分隔的可选字符串 **：**（冒号）将类名称传递到类名称器 **。ctorargs** 是一个以逗号分隔的字符串列表。例如 **{CLASS@org.test.passwd}org.test.passwd.ExternamPassworProvider**。在本例中，**org.test.passwd.ExternamPassworProvider** 类是从 **org.test.passwd** 模块加载的，并使用 **toCharArray()** 方法获取密码。如果 **toCharArray**（） 不可用，则使用 **toString**（） 方法。**org.test.passwd.ExternamPassworProvider** 类必须具有默认的构造器。

第 5 章 JAVA 安全管理器

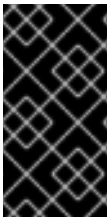
通过定义 Java 安全策略，您可以配置 Java 安全管理器来管理 Java 虚拟机(JVM)的外部边界。

5.1. 关于 JAVA 安全管理器

Java 安全管理器是管理 Java 虚拟机(JVM)沙盒外部边界的类，控制 JVM 中执行的代码如何与 JVM 外部的资源交互。激活 Java 安全管理器时，Java API 会与安全管理器检查是否批准，然后再执行大量潜在的不安全操作。Java 安全管理器使用安全策略来确定是允许还是拒绝给定的操作。

5.2. 关于 JAVA 安全策略

Java 安全策略是为不同类型的代码定义的一组权限。Java 安全管理器将应用程序请求的操作与安全策略进行比较。如果策略允许某个操作，则安全管理器将允许执行该操作。如果策略不允许该操作，则安全管理器将拒绝该操作。



重要

较早版本的 JBoss EAP 使用外部文件定义了策略，如 ***EAP_HOME*/bin/server.policy**。JBoss EAP 7 通过两种方式定义了 Java 安全策略：**security-manager** 子系统，以及各个部署中的 XML 文件。**security-manager** 子系统定义 *ALL* 部署的最低和最大权限，而 XML 文件则指定各个部署请求的权限。

5.2.1. 关于在安全管理器子系统中定义策略

security-manager 子系统允许您为所有部署定义共享或常见权限。这可以通过定义最小和最大权限集来实现。所有部署将获得至少在最小权限中定义的所有权限。如果部署流程请求了超过最大权限集中定义的权限的权限，部署过程将失败。

示例：用于更新最小权限集的管理 CLI 命令

```
/subsystem=security-manager/deployment-permissions=default:write-attribute(name=minimum-permissions, value=[{class="java.util.PropertyPermission", actions="read", name="*"}])
```

示例：更新最大权限集的管理 CLI 命令

```
/subsystem=security-manager/deployment-permissions=default:write-attribute(name=maximum-permissions, value=[{class="java.util.PropertyPermission", actions="read,write", name="*"},
{class="java.io.FilePermission", actions="read,write", name="/-"}])
```

注意

如果未定义最大权限集，则其值默认为 **java.security.AllPermission**。

其他资源

- 您可以在 JBoss EAP *配置指南* 中找到 **security-manager** 子系统的完整参考。

5.2.2. 关于在部署中定义策略

在 JBoss EAP 7 中，您可以在部署中添加 **META-INF/permissions.xml**。此文件允许您指定部署所需的权限。

如果在 **security-manager** 子系统中定义了最小权限，并且将 **META-INF/permissions.xml** 添加至您的部署中，则将授予这些权限的联合。如果 **permissions.xml** 中请求的权限超过 **security-manager** 子系统中定义的最大策略，其部署将无法成功。如果 **META-INF/permissions.xml** 和 **META-INF/jboss-permissions.xml** 都显示在部署中，则只授予 **META-INF/jboss-permissions.xml** 中请求的权限。

该规范规定 **permissions.xml** 涵盖了整个应用或顶级部署模块。如果您要为子部署定义特定权限，您可以使用 JBoss EAP 特定的 **META-INF/jboss-permissions.xml**。它采用与 **permissions.xml** 完全相同的格式，并且仅应用到声明它的部署模块。

示例：Sample permissions.xml

```
<permissions version="7">
  <permission>
    <class-name>java.util.PropertyPermission</class-name>
    <name>*</name>
    <actions>read</actions>
  </permission>
</permissions>
```

其他资源

- JSR 342 **META-INF/permissions.xml** 文件。

5.2.3. 关于在模块中定义策略

您可以通过在 **module.xml** 文件中添加 **<permissions>** 元素来限制模块的权限。**<permissions>** 元素包含零个或多个 **<grant>** 元素，它定义了授予该模块的权限。每个 **<grant>** 元素包含以下属性：

权限
要授予的权限的合格类名称。

name
向权限类构造器提供的权限名称。

操作
某些权限类型所需的（可选）操作列表。

示例：带有定义策略的 module.xml

```
<module xmlns="urn:jboss:module:1.5" name="org.jboss.test.example">
  <permissions>
    <grant permission="java.util.PropertyPermission" name="*" actions="read,write" />
    <grant permission="java.io.FilePermission" name="/etc/-" actions="read" />
  </permissions>
  ...
</module>
```

如果存在 **<permissions>** 元素，则该模块将限制为您列出的权限。如果没有 **<permissions>** 元素，则模块不会有限制。

5.3. 使用 JAVA 安全管理器运行 JBOSS EAP

您可以通过两种不同的方式使用 Java 安全性管理器运行 JBoss EAP：运行 Java 安全管理器的方法有两种：

- 使用带有启动配置脚本的 **-secmgr** 标志。
- 使用启动配置文件.

重要

之前版本的 JBoss EAP 允许使用 **-Djava.security.manager** Java 系统属性和自定义安全管理器。JBoss EAP 7 中不支持这两者。此外，Java 安全管理器策略现在在 **security-manager** 子系统中定义，这意味着 JBoss EAP 7 不支持外部策略文件和 **-Djava.security.policy** Java 系统属性。

重要

在启用 Java 安全管理器的情况下启动 JBoss EAP 之前，您需要确保 **security-manager** 子系统中定义了所有安全策略。

5.3.1. 使用带有启动配置脚本的 -secmgr 标志。

您可以使用 Java 安全性管理器运行 JBoss EAP。为此，请在启动时使用 **secmgr** 选项。

流程

- 启动 JBoss EAP 实例时，包含 **-secmgr** 标志。

  如何包含 -secmgr 标记的示例

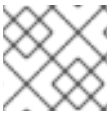
  ```
  ./standalone.sh -secmgr
  ```

5.3.2. 使用启动配置文件

您可以使用 Java 安全性管理器运行 JBoss EAP。为此，您必须修改启动配置文件。

重要

在编辑任何配置文件之前，必须完全停止域或单机服务器。



注意

如果您在受管域中使用 JBoss EAP，则必须对域中的每个物理主机或实例执行下列步骤。

流程

- 使用启动配置文件启用 Java Security Manager，您需要编辑 **standalone.conf** 或 **domain.conf** 文件，具体取决于您运行一个独立的实例或受管域。如果在 Windows 中运行，则改为使用 **standalone.conf.bat** 或 **domain.conf.bat** 文件。
  在配置文件中取消注释 **SECMGR="true"** 行：

  standalone.conf 或 domain.conf 示例

  ```
  # Uncomment this to run with a security manager enabled
  SECMGR="true"
  ```

standalone.conf.bat 或 domain.conf.bat 示例

```
rem # Uncomment this to run with a security manager enabled
set "SECMGR=true"
```

5.4. 从之前的版本进行迁移前的注意事项

将应用从 JBoss EAP 的旧版本移至已启用 Java 安全管理器运行的 JBoss EAP 7 时，您需要了解如何定义策略以及 JBoss EAP 配置和部署所需的配置。以下是您应该了解的变更：

- 在之前的 JBoss EAP 版本中，策略在外部配置文件中定义。在 JBoss EAP 7 中，使用 **security-manager** 子系统以及部署中包含的 **permissions.xml** 或 **jboss-permissions.xml** 来定义策略。
- 在 JBoss EAP 的早期版本中，您可以使用 **-Djava.security.manager** 和 **-Djava.security.policy** Java 系统属性。它们不再被支持，而应使用 **secmgr** 标志来启用 JBoss EAP 与 Java 安全管理器一起运行。
- JBoss EAP 7 不支持自定义安全管理器。

其他资源

- 定义 Java 安全策略.
- 如何使用 Java 安全性管理器运行 JBoss EAP。

附录 A. 参考资料

A.1. ELYTRON 子系统组件参考

表 A.1. add-prefix-role-mapper Attributes

属性	描述
prefix	要添加到每个角色的前缀。

表 A.2. add-suffix-role-mapper Attributes

属性	描述
suffix	要添加到各个角色的后缀。

表 A.3. aggregate-http-server-mechanism-factory Attributes

属性	描述
http-server-mechanism-factories	要聚合的 HTTP 服务器工厂列表。

表 A.4. aggregate-principal-decoder Attributes

属性	描述
principal-decoders	要聚合的主体解码器列表。

表 A.5. aggregate-principal-transformer Attributes

属性	描述
principal-transformers	要聚合的主体转换器列表。

表 A.6. aggregate-providers Attributes

属性	描述
providers	要聚合的引用 **Provider[]** 资源的列表。

表 A.7. aggregate-realm Attributes

属性	描述
authentication-realm	引用用于身份验证步骤的安全域。这用于获取或验证凭证。

属性	描述
authorization-realm	引用用于加载授权步骤的身份的安全域。
authorization-realms	引用安全域，以聚合用于加载授权步骤的身份。 有关使用多个授权域的详情，请参考 *如何配置身份管理指南*中的使用多个身份存储配置身份验证和授权。

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。

表 A.8. aggregate-role-mapper 属性

属性	描述
role-mappers	要聚合的角色映射器列表。

表 A.9. aggregate-sasl-server-factory Attributes

属性	描述
sasl-server-factories	要聚合的 SASL 服务器工厂列表。

表 A.10. 身份验证配置属性

属性	描述
匿名	如果允许 **true** 匿名身份验证。默认值为 **false**。
authentication-name	要使用的身份验证名称。
authorization-name	要使用的授权名称。
credential-reference	用于身份验证的凭据。这可以是明文，也可以是对存储在凭据存储中的 **凭据的引用。**
extends	扩展的现有身份验证配置。
主机	要使用的主机。
kerberos-security-factory	参考用于获取 GSS kerberos 凭证的 Kerberos 安全工厂。
mechanism-properties	SASL 身份验证机制的配置属性。

属性	描述
port	要使用的端口。
protocol	要使用的协议.
realm	要使用的域。
sasl-mechanism-selector	SASL 机制选择器字符串。有关 **sasl-mechanism-selector** 所需要的 getmmar 的更多信息，请参阅 JBoss EAP *配置服务器* 安全性的**sasl -mechanism-selector Grammar**。
security-domain	引用安全域以获取转发身份。

表 A.11. authentication-context 属性

属性	描述
extends	要扩展的现有身份验证上下文。
match-rules	针对此身份验证上下文与 匹配的规则。

表 A.12. authentication-context match-rules Attributes

属性	描述
authentication-configuration	引用用于成功匹配的身份验证配置。
match-abstract-type	要匹配的抽象类型。
match-abstract-type-authority	要匹配的抽象类型权威。
match-host	要匹配的主机。
match-local-security-domain	要匹配的本地安全域。
match-no-user	如果为 **true**，则规则将匹配任何用户。
match-path	要匹配的补丁。
match-port	要匹配的端口。
match-protocol	要匹配的协议。
match-urn	要匹配的 URN。

属性	描述
match-user	要匹配的用户。
ssl-context	引用用于成功匹配的 **ssl-context**.

表 A.13. cache-realm 属性

属性	描述
maximum-age	项目可在缓存中保留的时间（毫秒）。**-1** 代表无限期保留项目。默认值为 **-1**。
maximum-entries	要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。

表 A.14. case-principal-transformer 属性

属性	描述
upper-case	可选属性，将主转换器的名称转换为大写字符（设置为 **true** ），这是默认设置。将 属性设置为 **false**，将主体转换器的名称转换为小写字符。

表 A.15. certificate-authority-account Attributes

属性	描述
alias	密钥存储中证书颁发机构帐户密钥的别名。如果密钥存储中尚不存在别名，则会自动生成证书颁发机构帐户密钥，并将其存储为别名下的 **PrivateKeyEntry**。
certificate-authority	要使用的证书颁发机构的名称。默认值为 **LetsEncrypt**。
contact-urls	证书认证机构可以就与此帐户相关的问题联系的 URL 列表。
credential-reference	访问证书颁发机构帐户密钥时使用的凭证。
key-store	包含证书颁发机构帐户密钥的密钥存储。

表 A.16. chained-principal-transformer 属性

属性	描述
principal-transformers	到链的主要转换器列表.

表 A.17. client-ssl-context 属性

属性	描述
cipher-suite-filter	要应用的过滤器来指定启用的密码套件。此过滤器取以冒号、逗号或空格分隔的项目列表。每个项目可以是 OpenSSL 样式的密码套件名称、标准 SSL/TLS 密码套件名称，也可以是关键字，如 **TLSv1.2** 或 **DES**。完整的关键字列表以及创建过滤器的详情，请参考 **CipherSuiteSelector** 类的 Javadoc 中。默认值为 **DEFAULT**，它对应于所有没有 **NULL** 加密的已知密码套件，并排除任何没有身份验证的密码套件。
key-manager	引用 **SSLContext** 中使用的 **key-manager**。
协议	启用的协议。允许的选项：**SSLv2**、**SSLv3**、**TLSv1**、**TLSv1.1**、**TLSv1.2**、**TLSv1.3.** 默认为启用 **TLSv1**, **TLSv1.1**, **TLSv1.2**, 和 **TLSv1.3**。 **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。
provider-name	要使用的供应商的名称。如果未指定，则来自提供程序的所有提供程序都将传递到 SSLContext。
providers	要获取用于加载 **SSLContext** 的 **Provider[]** 的提供程序名称。
session-timeout	SSL 会话的超时时间。
trust-manager	引用 **SSLContext** 中使用的 **trust-manager**。

表 A.18. Concatenating-principal-decoder Attributes

属性	描述
joiner	用于加入 **principal-decoders** 属性中值的字符串。
principal-decoders	要连接的主要解码器列表。

表 A.19. 可配置-http-server-mechanism-factory 属性

属性	描述
过滤器	要应用的过滤器列表，以便根据名称启用或禁用机制。
http-server-mechanism-factory	引用要包装的 http 服务器工厂。
属性	要传递给 HTTP 服务器工厂调用的自定义属性。

表 A.20. 可配置-http-server-mechanism-factory 过滤器属性

属性	描述
pattern-filter	基于正则表达式模式进行过滤.
启用	如果为 **true**，则如果机制匹配，则会启用过滤器。默认值为 **true**。

表 A.21. 可配置-sasl-server-factory 属性

属性	描述
过滤器	要按顺序评估的过滤器列表，并使用 **or**。
属性	要传递给 SASL 服务器工厂调用的自定义属性。
protocol	创建机制时，协议传递到工厂。
sasl-server-factory	参考要包装的 SASL 服务器工厂.
server-name	创建机制时传递到工厂的服务器名称。

表 A.22. 可配置-sasl-server-factory 过滤器属性

属性	描述
启用	如果为 **true**，则如果工厂匹配，则会启用过滤器。默认值为 **true**。
predefined-filter	用于过滤机制名称的预定义过滤器。允许的值有 **HASH_MD5**、**HASH_SHA**、**HASH_SHA_256**、**HASH_SHA_384**、**HASH_SHA_512**、**GS2**、**SCRAM**、**DIGEST**、**IEC_ISO_9798**、**EAP**、**MUTUAL**、**BINDING** 和 **RECOMMENDED**.
pattern-filter	基于正则表达式的机制名称的过滤器。

表 A.23. 恒定权限映射器属性

属性	描述
permission-sets	匹配项时要分配的权限集。权限集可用于为身份分配权限。 **permission-set** 可以接受以下属性： ● **permission-set** 权限集的引用。  **注意** **permissions** 属性已弃用，它被 **permission-sets** 替代。

表 A.24. constant-principal-decoder Attributes

属性	描述
恒定	主体解码器将始终返回的恒定值。

表 A.25. constant-principal-transformer Attributes

属性	描述
恒定	这个主要转换器始终会返回的恒定值。

表 A.26. constant-realm-mapper Attributes

属性	描述
realm-name	对将返回的域的引用。

表 A.27. constant-role-mapper Attributes

属性	描述
roles	将返回的角色列表。

表 A.28. credential-store 属性

属性	描述
create	指定凭证存储是否应在不存在时创建存储。默认值为 **false**。

属性	描述
credential-reference	对用于创建保护参数的凭据的引用。这可以是明文，也可以是对存储在凭据存储中的 **凭据的引用。**
实施属性	凭据的映射存储特定于实施的属性。
modifiable	能否修改凭证存储。默认值为 **true**。
other-providers	要获取供应商的名称，在凭证存储中创建所需的 Jakarta Connectors 对象。这只适用于基于密钥存储的凭据存储。如果未指定，则改为使用全局供应商列表。
path	凭证存储的文件名。
provider-name	用于实例化 **CredentialStoreSpi** 的提供程序名称。如果没有指定提供程序，则使用找到的第一个可创建指定类型的实例的供应商。
providers	要获取供应商的名称，以搜索可创建所需凭据存储类型。如果未指定，则改为使用全局供应商列表。
relative-to	此凭证存储路径相对于的基本路径。
type	凭据存储的类型，如 **KeyStoreCredentialStore**。

表 A.29. credential-store 别名

属性	描述
entry-type	存储在凭据存储中的凭证条目类型。
secret-value	secret 值，如 password。

表 A.30. credential-store KeyStoreCredentialStore 实施属性

属性	描述
cryptoAlg	用于加密外部存储上的条目的密码算法名称。此属性仅在启用了 **external** 时有效。默认值为 **AES**。
external	指定数据是否存储到外部存储并由 **keyAlias** 加密。默认值为 **false**。
externalPath	指定到外部存储的路径。此属性仅在启用了 **external** 时有效。

属性	描述
keyAlias	凭据存储中的机密密钥别名，用于存储用于加密或解密数据到外部存储。
keyStoreType	密钥存储类型，如 **PKCS11**。默认为 **KeyStore.getDefaultType( )**。

表 A.31. custom-credential-security-factory Attributes

属性	描述
class-name	实施自定义安全工厂的类名称。
配置	自定义安全工厂的可选键和值配置。
module	用于加载自定义安全工厂的模块。

表 A.32. custom-modable-realm 属性

属性	描述
class-name	自定义域实施的类名称。
配置	自定义域的可选键和值配置。
module	用于加载自定义域的模块。

表 A.33. custom-permission-mapper Attributes

属性	描述
class-name	权限映射器的完全限定类名称。
配置	权限映射器的可选键和值配置。
module	用于加载权限映射器的模块名称。

表 A.34. custom-principal-decoder Attributes

属性	描述
class-name	主体解码器的完全限定类名称。
配置	主体解码器的可选键和值配置。

属性	描述
module	用于加载主体解码器的模块名称。

表 A.35. custom-principal-transformer Attributes

属性	描述
class-name	主体转换器的完全限定类名称。
配置	主体转换器的可选键和值配置。
module	用于载入主体转换器的模块名称。

表 A.36. custom-realm 属性

属性	描述
class-name	自定义域的完全限定域名。
配置	自定义域的可选键和值配置。
module	用于加载自定义域的模块名称。

表 A.37. custom-realm-mapper 属性

属性	描述
class-name	域映射程序的完全限定域名。
配置	域映射程序的可选键和值配置。
module	用于加载 realm 映射器的模块名称。

表 A.38. custom-role-decoder Attributes

属性	描述
class-name	角色解码器的完全限定类名称。
配置	角色解码器的可选键和值配置。
module	用于加载角色解码器的模块名称。

表 A.39. custom-role-mapper 属性

属性	描述
class-name	角色映射程序的完全限定域名。
配置	角色映射程序的可选键和值配置。
module	用于加载角色映射程序的模块名称。

表 A.40. dir-context 属性

属性	描述
authentication-context	获取用于连接 LDAP 服务器的登录凭据的身份验证上下文。如果 **authentication-level** 是 **none**，则可以省略，这等同于匿名身份验证。
authentication-level	要使用的身份验证级别，即安全级别或身份验证机制。对应于 **SECURITY_AUTHENTICATION** 或 **java.naming.security.authentication** 环境属性。允许的值为 **none**, **simple** 和 sasl_mech 格式。sasl_mech 格式是 SASL 机制名称的空格分隔列表。
connection-timeout	以毫秒为单位连接到 LDAP 服务器的超时。
credential-reference	用于进行身份验证并连接到 LDAP 服务器的凭据引用。如果 **authentication-level** 是 **none**，则可以省略它，这等同于匿名身份验证。
enable-connection-pooling	如果启用了 **true** 连接池。默认值为 **false**。
module	用作类加载基础的模块名称。
主体	验证并连接到 LDAP 服务器的主体。如果 **authentication-level** 为 **none** 则这个会被忽略，这等同于匿名身份验证。
属性	**DirContext** 的额外连接属性。
read-timeout	LDAP 操作的读取超时（以毫秒为单位）。
referral-mode	用于确定是否应遵循引用的模式。允许的值是 FOLLOW、**IGNORE** 和 **THROW**。默认值为 **IGNORE**。
ssl-context	用于安全连接到 LDAP 服务器的 SSL 上下文的名称。
url	连接 URL。

表 A.41. expression=encryption 属性

属性	描述
default-resolver	可选属性。当在没有加密表达式的情况下，要使用的解析器使用。例如，如果您将 "exampleResolver" 设置为 **默认的解析程序**，并且创建带有命令 **/subsystem=elytron/expression=encryption:create-expression(clear-text=TestPassword)** 的加密表达式，则 Elytron 使用"exampleResolver"作为该加密表达式的解析程序。
prefix	在加密表达式中使用的前缀。默认为 **ENC**。为可能已经定义 **ENC** 的情形中提供此属性。除非与已定义的 **ENC** 前缀冲突，否则不应更改这个值。
解析器	定义的解析器列表。解析器有以下属性： • **name** - 用于引用它的独立配置的名称。 • **credential-store** - 对包含此解析器使用的 secret 密钥的凭证存储实例的引用。 • **secret-key** - 从给定凭证存储中使用 secret 键的别名。

表 A.42. filesystem-realm 属性

属性	描述
编码	身份名称是否应以文件名形式存储编码(Base32)。
级	要应用的目录散列数量。默认值为 **2**。
path	包含域的文件的路径。
relative-to	与 **path** 一起使用的预定义相对路径。例如 **jboss.server.config.dir**。

表 A.43. filtering-key-store 属性

属性	描述

属性	描述
alias-filter	用于应用到从 **key-store** 返回的别名的过滤器。它可以是逗号分隔的别名列表，用于返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**  **注意** **alias-filter** 属性区分大小写。由于使用混合案例或大写别名（如 **elytronAppServer** ）可能无法被某些密钥存储提供程序识别，因此建议使用小写别名，如 **elytronappserver**。
key-store	对要过滤的 **key-store** 的引用。

表 A.44. generate-key-pair 属性

属性	描述
算法	指定加密算法，如 RSA、DSA 或 EC。默认值为 RSA。
size	以位为单位指定私钥的大小。密钥对类型的默认大小值如下：RSA 为 **2048** ；DSA 为 **2048**，而 EC 为 **256**。

表 A.45. HTTP-authentication-factory 属性

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

表 A.46. HTTP-authentication-factory mechanism-configurations 属性

属性	描述
credential-security-factory	用于根据机制要求获取凭证的安全因素。

属性	描述
final-principal-transformer	应用于此机制域的最后一个主要转换器。
host-name	此配置应用到的主机名。
mechanism-name	此配置仅应用使用指定名称的机制。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 realm 名称的定义列表。
pre-realm-principal-transformer	在选择了域前应用主体转换器。
post-realm-principal-transformer	选择域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 A.47. 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要转换器。
post-realm-principal-transformer	选择域后要应用的主体转换器。
pre-realm-principal-transformer	在选择了域前应用主体转换器。
realm-mapper	机制使用的 realm mapper。
realm-name	要按机制显示的域的名称。

表 A.48. identity-realm 属性

属性	描述
attribute-name	与这个身份关联的属性名称。
attribute-values	与 identity 属性关联的值列表。
identity	安全域中提供的身份。

表 A.49. import-key-pair 属性

属性	描述
key-passphrase	可选属性。设置密码短语来解密私钥。
private-key-location	包含私钥的文件的路径。只有您尚未指定 **private-key-string** 属性时指定。
private-key-string	将私钥设置为字符串。只有您尚未指定 **private-key-location** 属性，才指定。
public-key-location	如果私钥采用 OpenSSH 以外的任何格式，则需要此项。包含公钥的文件的路径。只有您尚未指定 **public-key-string** 属性时指定。
public-key-string	如果私钥采用 OpenSSH 以外的任何格式，则需要此项。将公钥设置为字符串。只有您尚未指定 **public-key-location** 属性，才指定。

表 A.50. jaspi-configuration 属性

属性	描述
application-context	在将此配置注册到 **AuthConfigFactory** 时使用。可以省略以允许通配符匹配。
description	用于向 **AuthConfigFactory** 提供描述。
层	在将此配置注册到 **AuthConfigFactory** 时使用。可以省略以允许通配符匹配。
name	允许在管理模型中引用资源的名称。

表 A.51. jaspi-configuration server-auth-module Attributes

属性	描述
class-name	**ServerAuthModule** 的完全限定域名。
flag	指示此模块如何与其他模块相关的控制标记。
module	从中加载 **ServerAuthModule** 的模块。
options	在初始化时要传递到 **ServerAuthModule** 的配置选项。

表 A.52. JDBC-realm 属性

属性	描述
principal-query	用于根据特定密钥类型验证用户身份的身份验证查询列表。

表 A.53. JDBC-realm principal-query 属性

属性	描述
attribute-mapping	为这个资源定义的属性映射列表。
bcrypt-mapper	密钥映射器，用于将从 SQL 查询返回的列映射到 **Bcrypt** 键类型。
clear-password-mapper	密钥映射程序将从 SQL 查询返回的列映射到明文密钥类型。它具有 **password-index** 子元素，这是列入代表用户密码的身份验证查询的索引。
data-source	用于连接到数据库的数据源的名称。
salted-simple-digest-mapper	密钥映射器，用于将从 SQL 查询返回的列映射到 **Salted Simple Digest** 键类型。
scram-mapper	密钥映射器，用于将从 SQL 查询返回的列映射到 **SCRAM** 密钥类型。
simple-digest-mapper	键映射映射从 SQL 查询返回的列到 **简单 Digest** 键类型。
sql	用于获取特定用户表列的密钥的 SQL 语句，并使用其类型对其进行映射。

表 A.54. jdbc-realm principal-query attribute-mapping 属性

属性	描述
index	列索引中代表映射属性的查询。
至	从从 SQL 查询返回的一列映射的身份属性的名称。

表 A.55. JDBC-realm principal-query bcrypt-mapper 属性

属性	描述
iteration-count-index	列索引中代表密码迭代计数（如果受支持）。
password-index	列从代表用户密码的身份验证查询的索引。

属性	描述
salt-index	列从代表密码 salt 的身份验证查询的索引（如果受支持）。

表 A.56. JDBC-realm principal-query salted-simple-digest-mapper 属性

属性	描述
算法	特定密码密钥映射器的算法。允许的值是 **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**, 和 **salt-password-digest-512**.默认为 **password-salt-digest-md5**。
password-index	列从代表用户密码的身份验证查询的索引。
salt-index	列从代表密码 salt 的身份验证查询的索引（如果受支持）。

表 A.57. JDBC-realm principal-query simple-digest-mapper 属性

属性	描述
算法	特定密码密钥映射器的算法。允许的值是 **simple-digest-md2**、**simple-digest-md5**、**simple-digest-sha-1**、**simple-digest-sha-256**、**simple-digest-sha-384** 和 **simple-digest-sha-512**。默认值为 **simple-digest-md5**。
password-index	列从代表用户密码的身份验证查询的索引。

表 A.58. jdbc-realm principal-query scram-mapper 属性

属性	描述
算法	特定密码密钥映射器的算法。允许的值有 **scram-sha-1** 和 **scram-sha-256**。默认值为 **scram-sha-256**。
iteration-count-index	列索引中代表密码迭代计数（如果受支持）。
password-index	列从代表用户密码的身份验证查询的索引。
salt-index	列从代表密码 salt 的身份验证查询的索引（如果受支持）。

表 A.59. Kerberos-security-factory 属性

属性	描述
debug	如果为 **true**，则获取凭证的 JAAS 步骤将启用 debug 日志记录。默认值为 **false**。
mechanism-names	凭据应使用的机制名称。名称将转换为 OID，并与 **mechanism-oids** 属性中的 OID 一起使用。
mechanism-oids	凭证应当可供使用的机制 OID 列表。
minimum-remaining-lifetime	在重新创建缓存凭证前，缓存凭证的时间（以秒为单位）。
obtain-kerberos-ticket	如果 **KerberosTicket** 也被获取并与凭证相关联。在将凭据委派给服务器时，需要满足此要求。
options	**Krb5LoginModule** 其他选项。
path	要加载的 keytab 的路径，以获取凭证。
主体	keytab 代表的主体。
relative-to	到 keytab 的相对路径。
request-lifetime	为新创建的凭证请求多少生命周期。
required	当服务启动时，带有适当的主体的 keytab 文件是否需要存在。
server	如果为 **true**，则这个因素用于 Kerberos 验证的服务器端。如果为 **false**，它将用于客户端。默认值为 **true**
wrap-gss-credential	是否应该换行 GSS 凭证以防止不当使用。

表 A.60. key-manager 属性

属性	描述
算法	用于创建底层的 **KeyManagerFactory** 的算法名称。这是 JDK 提供的。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。有关 SunJSSE 的详细信息，请参阅 Java 安全套接字扩展(JSSE)参考指南。
alias-filter	适用于从密钥存储返回的别名的过滤器。这可以是逗号分隔的别名列表，用于返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**

属性	描述
credential-reference	用于解密密钥存储项目的凭据引用。这可以以明文形式指定，也可以作为存储在 credentials **-store 中的凭证** 的引用。这不是密钥存储的密码。
key-store	对用于初始化底层 **KeyManageronnectionFactory** 的 **key-store** 的引用。
provider-name	用于创建底层的 **KeyManagerFactory** 的供应商名称。
providers	参考以获取创建底层 **KeyManagerFactory** 时要使用的 **Provider[]**。

表 A.61. key-store 属性

属性	描述
alias-filter	要应用到密钥存储返回的别名的过滤器，可以是逗号分隔的别名列表，可以返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**  **注意** **alias-filter** 属性区分大小写。由于使用混合案例或大写别名（如 **elytronAppServer**）可能无法被某些密钥存储提供程序识别，因此建议使用小写别名，如 **elytronappserver**。
credential-reference	用于访问密钥存储的密码。这可以以明文形式指定，也可以作为存储在 credentials **-store 中的凭证** 的引用。
path	密钥存储文件的路径。
provider-name	用于加载密钥存储的供应商名称。设置此属性将禁用搜索可以创建指定类型的密钥存储的第一个提供程序。
providers	对应该用来获取搜索的供应商实例列表的引用。如果未指定，则改为使用全局供应商列表。
relative-to	这个存储相对于的基本路径。这可以是完整路径或预定义的路径，如 **jboss.server.config.dir**。
required	如果为 **true**，则密钥存储服务启动时需要存在引用的密钥存储文件。默认值为 **false**。

属性	描述
type	密钥存储的类型，如 **JKS**。 **注意** 以下密钥存储类型会自动检测到： • **JKS** • **JCEKS** • **PKCS12** • **BKS** • **BCFKS** • **UBER** 您必须手动指定其他密钥存储类型。 在 JDK 8 的 Java Cryptography 架构标准 Algorithm Name 文档中找到完整的密钥存储类型列表。

表 A.62. key-store-realm 属性

属性	描述
key-store	引用用于支持此安全域的密钥存储。

表 A.63. LDAP-key-store 属性

属性	描述
alias-attribute	存储项目别名的 LDAP 属性的名称。
certificate-attribute	存储证书的 LDAP 属性的名称。
certificate-chain-attribute	存储证书链的 LDAP 属性的名称。
certificate-chain-encoding	证书链的编码。
certificate-type	证书的类型。
dir-context	用于与 LDAP 服务器通信的 **dir-context** 的名称。
filter-alias	用于按别名获取密钥存储中的项目的 LDAP 过滤器。
filter-certificate	用于根据证书获取密钥存储中的项目的 LDAP 过滤器。

属性	描述
filter-iterate	用于迭代密钥存储所有项目的 LDAP 过滤器。
key-attribute	存储密钥的 LDAP 属性的名称。
key-type	以序列化方式保存在 LDAP 属性中的密钥存储类型。例如，**JKS**。在 JDK 8 的 Java Cryptography 架构标准 Algorithm Name 文档中找到完整的密钥存储类型列表。
new-item-template	用于创建项目的配置。这将定义新创建的密钥存储项的 LDAP 条目将如何查找。
search-path	将搜索密钥存储项的 LDAP 中的路径。
search- recursive	如果 LDAP 搜索应该递归。
search-time-limit	从 LDAP 获取密钥存储项目的时间限值（毫秒）。默认值为 **10000**。

表 A.64. LDAP-key-store new-item-template Attributes

属性	描述
new-item-attributes	为新创建的项目设置的 LDAP 属性。这取一个带有**名称**和**值**对的项目列表。
new-item-path	将存储新创建的密钥存储项的 LDAP 中的路径。
new-item-rdn	新创建的项目的 LDAP RDN 的名称。

表 A.65. LDAP-realm 属性

属性	描述
allow-blank-password	此域是否支持空白密码直接验证。否则，空白密码尝试将被拒绝。
dir-context	用于连接到 LDAP 服务器的 **dir-context** 的名称。
direct-verification	如果为 **true**，这个域支持通过直接连接到 LDAP 作为被身份验证的帐户来验证凭据；否则，密码从 LDAP 服务器检索并在 JBoss EAP 中验证。如果启用，JBoss EAP 服务器必须能够从客户端获取纯文本，这需要 **PLAIN** SASL 或 **BASIC** HTTP 机制进行身份验证。默认值为 **false**。
identity-mapping	定义主体如何映射到底层 LDAP 服务器中的对应条目的配置选项。

表 A.66. LDAP-realm 身份映射属性

属性	描述
attribute-mapping	为这个资源定义的属性映射列表。
filter-name	用于按名称获取身份的 LDAP 过滤器。
iterator-filter	用于迭代域身份的 LDAP 过滤器。
new-identity-attributes	新创建的身份的属性列表，域中的 modifiability 需要。这是**名称**和**值**对对象的列表。
otp-credential-mapper	OTP 凭证的凭证映射。
new-identity-parent-dn	新创建的身份的父用户的 DN。该域的 modifiability 需要此项。
rdn-identifier	用于从 LDAP 条目获取主体 DN 的 RDN 部分。这在创建新身份时也使用它。
search-base-dn	用于搜索身份的基本 DN。
use-recursive-search	如果为 **true** 身份搜索查询是递归的。默认值为 **false**。
user-password-mapper	与 userPassword 类似的凭证的凭证映射。
x509-credential-mapper	允许使用 LDAP 作为 X509 凭证的存储配置。如果未定义 **-from** 子属性，则此配置将被忽略。如果定义了多个 **-from** 子属性，则用户证书必须与所有定义的标准匹配。

表 A.67. LDAP-realm identity-mapping 属性

属性	描述
extract-rdn	用作属性的值的 RDN 键，以 X.500 格式表示值。
filter	用于获取特定属性的值的过滤器。
filter-base-dn	执行该过滤器的上下文名称。
from	映射到身份属性的 LDAP 属性的名称。如果没有定义，则使用条目的 DN。
reference	包含要从中获取值的条目 DN 的 LDAP 属性名称。
role-recursion	递归角色分配的最大深度。使用 **0** 来指定递归。默认值为 **0**。

属性	描述
role-recursion-name	确定角色条目的 LDAP 属性，在搜索角色角色时，在 **filter-name** 中替换 "{0}"。
search- recursive	如果 **true** 属性 LDAP 搜索查询为递归。默认值为**true**。
至	从特定 LDAP 属性映射的身份属性的名称。如果没有提供，则属性的名称与 **from** 中的定义相同。如果 **from** 尚未定义，则使用值 **dn**。

表 A.68. LDAP-realm identity-mapping user-password-mapper 属性

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

表 A.69. LDAP-realm identity-mapping otp-credential-mapper Attributes

属性	描述
algorithm-from	OTP 算法 LDAP 属性的名称。
hash-from	OTP hash 功能的 LDAP 属性的名称。
seed-from	OTP seed 的 LDAP 属性的名称。
sequence-from	OTP 序列号的 LDAP 属性的名称。

表 A.70. LDAP-realm identity-mapping x509-credential-mapper 属性

属性	描述
certificate-from	映射到编码的用户证书的 LDAP 属性的名称。如果没有定义，则不会检查编码的证书。
digest-algorithm	摘要算法，即哈希功能，用于计算用户证书摘要。仅在定义了 **digest-from** 时使用。

属性	描述
digest-from	映射到用户证书摘要的 LDAP 属性的名称。如果没有定义，则不会检查证书摘要。
serial-number-from	映射到用户证书的序列号的 LDAP 属性的名称。如果未定义，则不会检查序列号。
subject-dn-from	映射到用户证书的主题 DN 的 LDAP 属性的名称。如果未定义，则不会检查主题 DN。

表 A.71. 逻辑权限映射程序属性

属性	描述
left	请参考用于操作左侧的权限映射程序。
逻辑协作	用于组合权限映射映射的逻辑操作。允许的值为 **and**, **or**, **xor**, 和 **unless**。
right	请参考用于操作右侧的权限映射程序。

表 A.72. logical-role-mapper 属性

属性	描述
left	对要在操作左侧使用的角色映射器的引用。
逻辑协作	要对角色映射执行的逻辑操作。允许的值有：**和**、**减去**、**或**、和 **xor**。
right	对要在操作右侧使用的角色映射器的引用。

表 A.73. mapped-regex-realm-mapper 属性

属性	描述
delegate-realm-mapper	如果没有使用 模式匹配的域映射程序。
pattern	正则表达式，必须至少包含一个捕获组才能从名称中提取域。
realm-map	使用正则表达式提取的域名称到定义的域名。

表 A.74. mechanism-provider-filtering-sasl-server-factory Attributes

属性	描述
启用	如果 **true** 没有启用供应商加载的机制，除非与其中一个过滤器匹配。默认值为 **true**。
过滤器	比较提供程序机制时要应用的过滤器列表。当所有指定值与机制和提供程序对匹配时，过滤器会匹配。
sasl-server-factory	对该定义嵌套到 SASL 服务器工厂的引用。

表 A.75. mechanism-provider-filtering-sasl-server-factory 过滤器属性

属性	描述
mechanism-name	此过滤器与 SASL 机制的名称匹配。
provider-name	此过滤器匹配的供应商名称。
provider-version	比较提供程序版本时使用的版本。
version-comparison	评估供应商版本时使用的同等程度。允许的值为 **less-than** 和 **greater-than**。默认值为 **less-than**。

表 A.76. online-certificate-status-protocol Attributes

属性	描述
responder	覆盖从证书解析的 OCSP Responder URI。
responder-certificate	如果未定义 **responder-keystore**，则响应者证书包包括在 **responseer-keystore** 或 **trust-manager** 密钥存储中。
responder-keystore	响应者证书的替代密钥存储。必须定义 **responder-certificate**。
prefer-crls	当同时配置 OCSP 和 CRL 机制时，会首先调用 OCSP 机制。当将 **prefer-crls** 设为 **true** 时，会首先调用 CRL 机制。

表 A.77. 权限设置权限属性

属性	描述
action	构成权限时要传递给权限的操作。
class-name	权限的完全限定域名。

属性	描述
module	用于加载权限的模块。
target-name	构造后要传递给权限的目标名称。

表 A.78. periodic-rotating-file-audit-log Attributes

属性	描述
autoflush	指定每个审计事件后是否输出流需要刷新。如果没有定义属性，则 **synchronized** 的值为默认值。
格式	使用 **SIMPLE** 进行人类可读的文本格式，或者使用 **JSON** 存储单个事件。
path	定义日志文件的位置。
relative-to	可选属性。定义日志文件的位置。
suffix	可选属性。在轮转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**。
同步	默认值为 **true**。指定文件描述符在每个审计事件后同步。

表 A.79. properties-realm 属性

属性	描述
groups-attribute	返回的 **AuthorizationIdentity** 中的属性名称，它应包含身份的组成员资格信息。
groups-properties	包含用户及其组的属性文件。
users-properties	包含用户及其密码的属性文件。

表 A.80. properties-realm 用户-properties 属性

属性	描述
digest-realm-name	如果属性文件中未发现，用于摘要密码的默认 realm 名称。
path	包含用户及其密码的 文件的路径。该文件应包含 realm 名称声明。

属性	描述
plain-text	如果为 **true**，则属性文件中的密码以纯文本形式存储。如果为 **false**，它们预先会进行哈希处理，格式为 **HEX(MD5(username \":\" realm \":\" password)))**。默认值为 **false**。
relative-to	路径相对于的预定义路径。

表 A.81. properties-realm groups-properties Attributes

属性	描述
path	包含用户及其组的文件的路径。
relative-to	路径相对于的预定义路径。

表 A.82. provider-http-server-mechanism-factory Attributes

providers	**用于定位因素的供应商。如果未指定，将使用全局注册的供应商列表。**

表 A.83. provider-loader Attributes

属性	描述
参数	作为 **Provider** 实例化而要传递给构造器的参数。
class-names	要加载的供应商的完全限定类名称列表。这些会在服务加载器发现的供应商后加载，并跳过所有重复数据。
配置	要传递至提供程序的键和值配置，以初始化它。
module	要从中加载提供程序的模块名称。
path	用于初始化提供程序的文件的路径。
relative-to	配置文件的基本路径。

表 A.84. provider-sasl-server-factory Attributes

属性	描述
providers	用于定位因素的供应商。如果未指定，将使用全局注册的供应商列表。

表 A.85. regex-principal-transformer Attributes

属性	描述
pattern	用于定位要替换的名称部分的正则表达式。
replace-all	如果为 **true**，将替换所有出现的模式匹配。如果为 **false**，则仅替换第一个出现的。默认值为 **false**。
替换	用作替换的值。

表 A.86. regex-role-mapper 属性

属性	描述
pattern	用于匹配角色的正则表达式。如果您想在替换中使用一部分原始角色，您可以使用组捕获。例如，要在角色（如 "app-admin", "batch-admin" 等角色后面捕获字符串），请使用模式 **.*-([a-z]*)$**。
替换	替换匹配项的字符串。您可以使用固定字符串，或引用从 **pattern** 属性中指定的正则表达式捕获的组。例如，在上面的模式中，您可以使用 **$1** 来指代第一个捕获的 group the group the the string "admin"，角色为 "app-admin" 和 "batch-admin"。
keep-non-mapped	将值设为 **true** 以保留与 **pattern** 属性中指定的正则表达式不匹配的角色。

表 A.87. regex-validating-principal-transformer Attributes

属性	描述
匹配	如果为 **true**，该名称必须与给定模式匹配才能成功验证。如果为 **false**，该名称必须与给定模式不匹配才能成功验证。默认值为 **true**。
pattern	主体转换器使用的正则表达式。

表 A.88. 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 A.89. SASL-authentication-factory mechanism-configurations 属性

属性	描述
credential-security-factory	用于根据机制要求获取凭证的安全因素。
final-principal-transformer	应用于此机制域的最后一个主要转换器。
host-name	此配置应用到的主机名。
mechanism-name	此配置仅应用使用指定名称的机制。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 realm 名称的定义列表。
protocol	此配置适用的协议。
post-realm-principal-transformer	选择域后要应用的主体转换器。
pre-realm-principal-transformer	在选择了域前应用主体转换器。
realm-mapper	机制使用的 realm mapper。

表 A.90. 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要转换器。
post-realm-principal-transformer	选择域后要应用的主体转换器。
pre-realm-principal-transformer	在选择了域前应用主体转换器。
realm-mapper	机制使用的 realm mapper。
realm-name	要按机制显示的域的名称。

表 A.91. secret-key-credential-store Attributes

属性	描述
create	如果您不希望 ELytron 创建它不存在，则将值设为 **false**。默认值为 **true**。
default-alias	默认生成的密钥的别名名称。默认值为 **key**。

属性	描述
key-size	生成密钥的大小。默认大小为 256 位。您可以将值设为以下之一： • **128** • **192** • **256**
path	凭证存储的路径。
populate	如果凭证存储不包含 **default-alias**，则此属性指示 Elytron 是否应该创建。默认值是 **true**。
relative-to	对之前定义的路径的引用，**该属性** 路径相对于.

表 A.92. server-ssl-context 属性

属性	描述
authentication-optional	**如果为** 安全域拒绝客户端证书，则不会防止连接。这样，当安全域拒绝客户端证书时，回退到使用其他验证机制，如表单登录。这只有在设置安全域时才会生效。默认值为 **false**。
cipher-suite-filter	要应用的过滤器指定启用的密码套件。此过滤器采用以冒号分隔、逗号或空格分隔的项目列表。每个项目可以是 OpenSSL 样式的密码套件名称、标准 SSL/TLS 密码套件名称，也可以是关键字，如 **TLSv1.2** 或 **DES**。完整的关键字列表以及创建过滤器的详情，请参考 **CipherSuiteSelector** 类的 Javadoc 中。默认值为 **DEFAULT**，它对应于所有没有 **NULL** 加密的密码套件，并排除没有身份验证的任何加密套件。
final-principal-transformer	应用于此机制域的最后一个主要转换器。
key-manager	参考在 **SSLContext** 中使用的主要管理器。
maximum-session-cache-size	要缓存的最大 SSL/TLS 会话数量。
need-client-auth	如果为 **true**，在 SSL 握手中需要客户端证书。没有可信客户端证书的连接将被拒绝。默认值为 **false**。
post-realm-principal-transformer	选择域后要应用的主体转换器。
pre-realm-principal-transformer	在选择了域前应用主体转换器。

属性	描述
协议	启用的协议。允许的选项有 **SSLv2**、**SSLv3**、 **TLSv1**、**TLSv1.1**、**TLSv1.2**、**TLSv1.3**。默认为启用 **TLSv1**, **TLSv1.1**, **TLSv1.2**, 和 **TLSv1.3**。 **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。
provider-name	要使用的供应商的名称。如果未指定，则来自提供程序的所有提供程序都将传递到 **SSLContext**。
providers	要获取用于加载 **SSLContext** 的 **Provider[]** 的提供程序名称。
realm-mapper	用于 SSL 验证的域映射程序。
security-domain	在 SSL/TLS 会话建立过程中用于身份验证的安全域。
session-timeout	SSL/TLS 会话超时。
trust-manager	对 SSLContext 中使用的 **trust-manager** 的引用。
use-cipher-suites-order	如果为 **true**，则将使用服务器中定义的密码套件顺序。如果为 **false**，将使用客户端提供的密码套件顺序。默认值为 **true**。
want-client-auth	如果为 **true**，则需要在 SSL 握手中请求客户端证书但不是必需的。如果引用安全域并支持 X509 证据，则会自动将其设置为 **true**。当设置了 **need-client-auth** 时，这会被忽略。默认值为 **false**。
wrap	如果为 **true**，则返回的 **SSLEngine**、**SSLSocket** 和 **SSLServerSocket** 实例将被嵌套，以防止进一步修改。默认值为 **false**。

注意

server-ssl-context 的 realm mapper 和主要转换器属性仅适用于 SASL EXTERNAL 机制，其中的证书由信任管理器验证。HTTP CLIENT-CERT 身份验证设置在 **http-authentication-factory** 中配置。

表 A.93. service-loader-http-server-mechanism-factory Attributes

属性	描述
module	用于获取加载的类加载器的模块。如果没有指定要加载资源的类加载加载的类加载程序。

表 A.94. service-loader-sasl-server-factory Attributes

属性	描述
module	用于获取加载的类加载器的模块。如果没有指定要加载资源的类加载加载的类加载程序。

表 A.95. simple-permission-mapper 属性

属性	描述
mapping-mode	在多个匹配项时应使用的映射模式。允许的值为 **are**, **and**, **or**, **xor**, **unless**, 和 **first**。默认值是 **第一个**。
权限映射	定义的权限映射列表。

表 A.96. simple-permission-mapper 权限属性

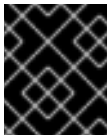
属性	描述
permission-sets	匹配时要分配的权限集。权限集可用于为身份分配权限。 **permission-sets** 可以采用以下属性： • **permission-set** 权限集的引用。  **重要** **permissions** 属性已弃用，并被 **permission-sets** 替换。
主体	在映射权限时要比较的主体列表，如果身份主体与列表中任何一个匹配，则要比较的主体列表。
roles	在映射权限时要比较的角色列表，如果身份是列表中某个对象的成员匹配。

表 A.97. simple-regex-realm-mapper 属性

属性	描述
delegate-realm-mapper	如果没有使用 模式匹配的域映射程序。
pattern	正则表达式，必须至少包含一个捕获组才能从名称中提取域。

表 A.98. simple-role-decoder Attributes

属性	描述
attribute	身份要直接映射到角色的属性名称。

表 A.99. source-address-role-decoder Attributes

属性	描述
pattern	指定客户端 IP 地址或要匹配的客户端 IP 地址的正则表达式。
source-address	指定客户端的 IP 地址。
roles	提供要分配给用户的角色列表，如果客户端的 IP 地址与 **pattern** 属性或 **source-address** 属性指定的值匹配。

注意

您必须在 **source-address** 属性或 **pattern** 属性中至少指定一个 IP 地址。否则，您无法根据客户端的 IP 地址来做出授权决策。

表 A.100. syslog-audit-log 属性

属性	描述
格式	审计事件应该记录的格式。 支持的值： • **JSON** • **SIMPLE** 默认值： • **SIMPLE**
host-name	要嵌入到发送到 syslog 服务器的所有事件中的主机名。
port	syslog 服务器上的监听端口。

属性	描述
reconnect-attempts	Elytron 将在关闭连接前尝试向 syslog 服务器发送连续消息的次数上限。只有使用的传输协议是 UDP 时，此属性的值才有效。 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 地址或一个可由 Java 的 **InetAddress.getByName()** 方法解析的名称。
ssl-context	连接到 syslog 服务器时使用的 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**
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**

表 A.101. 文件审计日志记录属性

属性	描述

属性	描述
autoflush	指定每个审计事件后是否输出流需要刷新。如果没有定义属性，则 **synchronized** 的值为默认值。
格式	默认值为 **SIMPLE**。使用 **SIMPLE** 进行人类可读的文本格式，或者使用 **JSON** 存储单个事件。
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
同步	默认值为 **true**。指定文件描述符在每个审计事件后同步。

表 A.102. 大小轮转文件审计日志属性

属性	描述
autoflush	指定每个审计事件后是否输出流需要刷新。如果没有定义属性，则 **synchronized** 的值为默认值。
格式	默认值为 **SIMPLE**。使用 **SIMPLE** 进行人类可读的文本格式，或者使用 **JSON** 存储单个事件。
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** 以在服务器重启时轮转日志。
rotate-size	日志文件在轮转日志前可访问的最大大小。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。您可以在大写或小写字符中指定单位。
suffix	可选属性。在轮转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**。
同步	默认值为 **true**。指定文件描述符在每个审计事件后同步。

表 A.103. token-realm 属性

属性	描述
jwt	用于与基于令牌的域结合使用的令牌验证器，该域根据 JWT/JWS 标准处理安全令牌。
oauth2-introspection	用于与基于令牌的域结合使用的令牌验证器，该域处理 OAuth2 访问令牌，并使用与 RFC-7662 OAuth2 令牌 Introspection 规范兼容的端点进行验证。
principal-claim	应该用来获取主体名称的声明名称。默认为 **用户名**。

表 A.104. token-realm jwt 属性

属性	描述
培训对象	代表此配置支持的听众的字符串列表。在验证 JWT 令牌时，必须有一个 **aud** 声明，其中包含此处定义的值之一。
certificate	带有公钥从密钥存储加载的证书名称，该密钥存储由 **key-store** 属性定义。
client-ssl-context	用于远程 JSON Web 密钥(JWK) 的 SSL 上下文。这可让您使用 **jku** (JSON Key URL)标头参数的 URL 获取令牌验证的公钥。
host-name-verification-policy	一个策略，用于定义在使用远程 JSON Web Keys 时如何验证主机名的策略。您可以为属性设置以下值之一： • **ANY**，禁用主机名验证。 • **DEFAULT**，它可以拒绝从证书和连接主机间的证书不匹配的连接。
issuer	代表此配置支持的签发者的字符串列表。在验证 JWT 令牌时，必须具有一个 **iss** 声明，其中包含此处定义的值之一。
key-store	应从中加载带有公钥的证书的密钥存储。此属性以及 **certificate** 属性也可用作 **公钥** 的替代选择。
public-key	PEM 格式的公钥。在验证过程中，如果提供了公钥，将基于此属性提供的键值来验证签名。 或者，**您可以定义一个密钥存储 和证书** 来配置公钥。这个替代密钥用来在没有 **kid** （密钥 ID）声明的情况下验证令牌。

表 A.105. token-realm oauth2-introspection Attributes

属性	描述
client-id	OAuth2 授权服务器上的客户端的标识符。
client-secret	客户端的机密。
client-ssl-context	如果内省端点使用 HTTPS，则将使用 SSL 上下文。
host-name-verification-policy	定义在使用 HTTPS 时如何验证主机名的策略。您可以为属性设置以下值之一： • **ANY**，禁用主机名验证。 • **DEFAULT**，它可以拒绝从证书和连接主机间的证书不匹配的连接。
introspection-url	令牌内省端点的 URL。

表 A.106. trust-manager 属性

属性	描述
算法	用于创建底层 **TrustManager factory 的算法名称**。这是 JDK 提供的。例如，使用 SunJSSE 的 JDK 提供了 **PKIX** 和 **SunX509** 算法。有关 SunJSSE 的详细信息，请参阅 Java 安全套接字扩展(JSSE)参考指南。
alias-filter	适用于从密钥存储返回的别名的过滤器。这可以是逗号分隔的别名列表，用于返回或以下格式之一： • **ALL:-alias1:-alias2** • **NONE:+alias1:+alias2**
certificate-revocation-list	启用信任管理器可以检查的证书撤销列表。**certificate-revocation-list** 的属性有： • **maximum-cert-path** - 认证路径中可存在的最大非签发的中间证书。默认值为 **5**。（已弃用。在 **trust-manager**中使用 **maximum-cert-path**。 • **path** - 用于初始化提供程序的配置文件的路径。 • **relative-to** - 证书撤销列表文件的基本路径。 如需更多信息，请参阅使用证书撤销列表。
key-store	对用于初始化底层 **TrustManagerFactory** 的 **key-store** 的引用。

属性	描述
maximum-cert-path	认证路径中可以存在的最大非签发的中间证书。默认值为 **5**。 此属性已从 JBoss EAP 7.3 中的 **trust-manager** 内的 **certificate-revocation-list** 移到 **trust-manager** 中。为了向后兼容，属性也会出现在 **certificate-revocation-list** 中。下一步，在 **trust-manager** 中使用 **maximum-cert-path**。  **注意** 在 **trust-manager** 或 **certificate-revocation-list** 中没有定义两者中的 **maximum-cert-path**。
only-leaf-cert	只检查叶证书的撤销状态。这是一个可选属性。默认值为 **false**。
provider-name	用于创建底层 **TrustManager factory 的供应商名称**。
providers	参考以获取创建底层 **TrustManagerFactory** 时要使用的 **Provider[]**。
soft-fail	当设置为 **true** 时，接受带有未知撤销状态的证书。这是一个可选属性。默认值为 **false**。

表 A.107. x500-attribute-principal-decoder Attributes

属性	描述
attribute-name	要映射的 X.500 属性的名称。这也可使用 **oid** 属性来定义。
convert	当设置为 **true** 时，如果主体解码器还没有将主体转换为 **X500Principal**，则主体解码器将尝试转换该主体。如果转换失败，原始值将用作主体。
joiner	加入的字符串。默认值为句点 (**.**)。
最大数据量	要映射的属性的最大数量。默认值为 **2147483647**。
oid	要映射的 X.500 属性的 OID。这也可通过 **attribute-name** 属性来定义。
required-attributes	主体中必须存在的属性名称列表
required-oids	主体中必须存在的属性的 OID 列表。
reverse	如果为 **true**，则属性值将按照相反的顺序处理并返回。默认值为 **false**。

属性	描述
start-segment	您要映射的属性的开头。这使用基于零的索引，默认值为 **0**。

表 A.108. x509-subject-alternative-name-evidence-decoder Attributes

属性	描述
alt-name-type	主题替代名称类型。必须是以下主题替代名称之一： • **directoryName** • **dNSName** • **iPAddress** • **registeredID** • **rfc822Name** • **uniformResourceIdentifier** 这是必需属性。
片段	• 基于 0 出现的主题备用名称进行映射。 • 如果给定类型有多个主题名称，则使用此属性。默认值为 **0**。

A.2. 配置您的环境以使用 BOUNCYCASTLE 供应商

您可以将 JBoss EAP 安装配置为使用 **BouncyCastle** 供应商。红帽不提供 Bouncy Castle JARs，它必须直接从 Bouncy Castle 获取。

重要

当指定 **BouncyCastle** 提供者时，必须使用 Java 8，因为 BouncyCastle API 只能被认证为 Java 8。

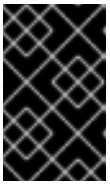
1. 在您的 JDK 的 classpath 中包含 BouncyCastle JARs，以 **bc-fips** 和 **bctls-fips** 开头。对于 Java 8，这通过将 JAR 文件放在 **$JAVA_HOME/lib/ext** 中来实现。
2. 使用以下任一方法，在您的 Java 安全配置文件中包括 **BouncyCastle** 供应商：
   - JDK 中提供了一个默认配置文件 **java.security**，并可更新为包含 **BouncyCastle** 提供程序。如果没有指定其他安全配置文件，则使用此文件。有关此文件的位置，请查看 JDK 供应商的文档。
   - 定义自定义 Java 安全配置文件，并通过添加 **-Djava.security.properties==*/path/to/*java.security.properties** 系统属性来引用它。使用两个等号引用时，默认策略将被覆盖，并且仅使用引用文件中定义的提供程序。使用一个等号时，如在 **-Djava.security.properties=*/path/to/*java.security.properties** 中一样，则提供程序会附加到默认安全文件中，首选使用在这两个文件中指定密钥时传递的文件。在需

要不同安全设置的同一主机上运行的多个 JVM 时，此选项很有用。

下面显示了一个定义这些提供程序的示例配置文件。

示例：BuncyCastle 安全策略

```
# We can override the values in the JRE_HOME/lib/security/java.security
# file here.  If both properties files specify values for the same key, the
# value from the command-line properties file is selected, as it is the last
# one loaded.  We can reorder and change security providers in this file.
security.provider.1=org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider
security.provider.2=org.bouncycastle.jsse.provider.BouncyCastleJsseProvider fips:BCFIPS
security.provider.3=sun.security.provider.Sun
security.provider.4=com.sun.crypto.provider.SunJCE

# This is a comma-separated list of algorithm and/or algorithm:provider
# entries.
#
securerandom.strongAlgorithms=DEFAULT:BCFIPS
```



重要

如果更新默认配置文件，则此文件中的所有其他 **security.provider.X** 行（例如 **security.provider**.2）必须提高其 **X** 的值，以确保该提供程序具有优先权。每个供应商都必须具有唯一优先级。

3. 将 **elytron** 子系统配置为独占使用 **BouncyCastle** 提供程序。默认情况下，系统配置为使用 **elytron** 和 **openssl** 提供程序。因为还包括一个 TLS 的实现，建议禁用 OpenSSL 供应商以确保使用 Bouncy Castle 的 TLS 实现。

```
/subsystem=elytron:write-attribute(name=final-providers,value=elytron)
```

4. 重新加载服务器以使更改生效。

```
reload
```

A.3. SASL 身份验证机制参考

A.3.1. 支持 SASL 身份验证机制的级别

名称	支持等级	注释
匿名	支持	
DIGEST-SHA-512	技术预览	支持，但目前未注册 IANA 名称。
DIGEST-SHA-256	技术预览	支持，但目前未注册 IANA 名称。
DIGEST-SHA	技术预览	支持，但目前未注册 IANA 名称。

名称	支持等级	注释
DIGEST-MD5	支持	
EXTERNAL	支持	
GS2-KRB5	支持	
GS2-KRB5-PLUS	支持	
GSSAPI	支持	
JBOSS-LOCAL-USER	支持	支持，但目前未注册 IANA 名称。
OAUTHBEARER	支持	
OTP	不支持	
PLAIN	支持	
SCRAM-SHA-1	支持	
SCRAM-SHA-1-PLUS	支持	
SCRAM-SHA-256	支持	
SCRAM-SHA-256-PLUS	支持	
SCRAM-SHA-384	支持	
SCRAM-SHA-384-PLUS	支持	
SCRAM-SHA-512	支持	
SCRAM-SHA-512-PLUS	支持	
9798-U-RSA-SHA1-ENC	不支持	
9798-M-RSA-SHA1-ENC	不支持	
9798-U-DSA-SHA1	不支持	
9798-M-DSA-SHA1	不支持	
9798-U-ECDSA-SHA1	不支持	
9798-M-ECDSA-SHA1	不支持	

A.3.2. SASL 身份验证机制属性

您可以在 Java 文档 中看到标准 Java SASL 验证机制属性列表 。下表中列出了其他 JBoss EAP 专用 SASL 身份验证机制属性。

表 A.109. SASL 属性在 SASL 机制或验证交换过程中使用

属性	客户端/服务器	描述
com.sun.security.sasl.digest.realm	Server	某些 SASL 机制使用（包括大多数 Oracle JDK 提供的 DIGEST-MD5 算法）为机制提供可能的服务器域列表。每个 realm 名称必须用空格字符(**U+0020**)分隔。
com.sun.security.sasl.digest.utf8	client, server	某些 SASL 机制使用，包括大多数 Oracle JDK 提供的 DIGEST-MD5 算法，以指明信息交换应使用 UTF-8 字符编码而非默认的 Latin-1/ISO-8859-1 编码。默认值为 **true**。
wildfly.sasl.authentication-timeout	Server	服务器应在多长时间内终止身份验证尝试的时间（以秒为单位）。默认值为 150 秒。
wildfly.sasl.channel-binding-required	client, server	表示需要支持频道绑定的机制。值为 **true** 表示需要频道绑定。任何其它值或缺少这个属性，表示不需要频道绑定。
wildfly.sasl.digest.alternative_protocols	Server	提供替代协议的单独列表，这些协议可在从客户端接收的响应中接受。列表可以是空格、逗号、制表符或以分开的新行。
wildfly.sasl.gssapi.client.delegate-credential	客户端	指定 GSSAPI 机制是否支持凭证委托。如果设置为 **true**，则凭据从客户端委派给服务器。 如果使用 **javax.security.sasl.credentials** 属性提供 **GSSCredential**，则此属性默认为 **true**。否则，默认值为 **false**。
wildfly.sasl.gs2.client.delegate-credential	客户端	指定 GS2 机制是否支持凭证委派。如果设置为 **true**，则凭据从客户端委派给服务器。 如果使用 **CredentialCallback** 提供 **GSSCredential**，则此属性默认为 **true**。否则，默认值为 **false**。
wildfly.sasl.local-user.challenge-path	Server	指定服务器在其中生成质询文件的目录。默认值为 **java.io.tmpdir** 系统属性。
wildfly.sasl.local-user.default-user	Server	用于 silent 身份验证的用户名。

属性	客户端/服务器	描述
wildfly.sasl.local-user.quiet-auth	客户端	为本地用户启用 silent 身份验证。默认值为 **true**。 请注意，如果未明确定义此属性并且客户端配置中指定了回调处理器或用户名，Jakarta Enterprise Beans 客户端和服务器会禁用静默本地身份验证。
wildfly.sasl.local-user.use-secure-random	Server	指定服务器在创建质询时是否使用安全的随机数字生成器。默认值为 **true**。
wildfly.sasl.mechanism-query-all	client, server	指明应返回所有可能支持的机制名称，无论是否存在其他任何属性。 此属性仅对 **SaslServerFactory#getMechanismNames(Map)** 或 **SaslClientonnectionFactoryy#getMechanismNames(Map)** 调用，对于 Elytron-provided SASL 工厂。
wildfly.sasl.otp.alternate-dictionary	客户端	为 OTP SASL 机制提供备用字典。每个字典必须用空格字符(**U+0020**)分隔。
wildfly.sasl.relax-compliance	Server	SASL 机制的规格要求某些行为，并验证该行为在相反的连接端。当与其它 SASL 机制交互时，其中一些要求将同样解释。如果此属性设为 **true**，则查询了检查，在规格解释中有所不同。默认值为 **false**。
wildfly.sasl.scram.min-iteration-count	client, server	用于 SCRAM 的最小迭代计数。默认值为 **4096**。
wildfly.sasl.scram.max-iteration-count	client, server	用于 SCRAM 的最大迭代计数。默认值为 **32786**。
wildfly.sasl.secure-rng	client, server	要使用的 **SecureRandom** 实施的算法名称。使用此属性可以提高性能。
wildfly.security.sasl.digest.ciphers	client, server	直接限制 SASL 机制支持的密码集的逗号分隔列表。

表 A.110. 身份验证后使用 SASL 属性

属性	客户端/服务器	描述
wildfly.sasl.principal	客户端	包含成功 SASL 客户端验证后的协商客户端主体。

属性	客户端/服务器	描述
wildfly.sasl.security-identity	Server	包含在成功 SASL 服务器端身份验证后协商的安全身份。

A.4. 安全授权参数

JBoss EAP 中 **安全** 命令的参数由定义的机制决定。每个机制都需要不同的属性，建议使用 tab 补全来检查定义的机制的各种要求。

表 A.111. Universal 参数

属性	描述
--mechanism	指定启用或禁用的机制。目前，支持 SASL 身份验证机制的支持 SASL 机制 列表和 **BASIC**、**CLIENT_CERT**、DIGEST、**DIGEST**、**DIGEST-SHA-256** 和 **FORM** HTTP 身份验证机制。
--no-reload	如果指定，服务器不会在安全命令完成后重新加载。

特定于机制的属性
以下属性仅符合特定机制：它们根据其功能分组到下方。

表 A.112. key-store Realm

属性	描述
--key-store-name	信任存储作为现有密钥存储的名称。如果 **--key-store-realm-name** 不用于 **EXTERNAL** SASL 机制或 **CLIENT_CERT** HTTP 机制，则必须指定它。
--key-store-realm-name	信任存储作为现有密钥存储域的名称。如果 **--key-store-name** 不用于 **EXTERNAL** SASL 机制或 **CLIENT_CERT** HTTP 机制，则必须指定它。
--roles	定义与当前身份关联的以逗号分隔的角色列表的可选参数。如果没有现有角色映射程序包含指定角色列表，则将生成并分配一个角色映射程序。

表 A.113. 文件系统 Realm

属性	描述
--exposed-realm	公开给用户的域。
--file-system-realm-name	文件系统域的名称。

属性	描述
--user-role-decoder	用于从用户存储库提取角色的角色解码器的名称。只有在指定了 **--file-system-realm-name** 时，此属性才会使用。

表 A.114. Realm 属性

属性	描述
--exposed-realm	公开给用户的域。这个值必须与用户属性文件中定义的 **realm-name** 匹配。
--groups-properties-file	到包括 **groups** 属性（对于管理操作）或 **roles**（对于 undertow 服务器的属性文件的路径。
--properties-realm-name	现有属性域的名称。
--relative-to	调整 **--group-properties-file** 和 **--users-properties-file** 的路径，使其相对于系统属性。
--users-properties-file	包含用户详情的属性文件的路径。

表 A.115. 其他属性

属性	描述
--management-interface	用于配置管理身份验证命令的管理接口。默认为 **http-interface**。
--new-auth-factory-name	用于指定身份验证工厂的名称。如果没有定义，则自动创建名称。
--new-realm-name	用于指定属性文件 realm 资源的名称。如果没有定义，则自动创建名称。
--new-security-domain	用于指定安全域的名称。如果没有定义，则自动创建名称。
--super-user	使用超级用户权限配置本地用户。可用于 **JBOSS-LOCAL-USER** 机制。

A.5. ELYTRON CLIENT SIDE ONE WAY 示例

配置服务器 SSL 上下文后，如果可能，务必要测试配置。Elytron 客户端 SSL 上下文可以放在配置文件中，然后从管理 CLI 执行，从而能够测试服务器配置。这些步骤假定已完成服务器端配置，并且服务器已重新加载（如有必要）。

1. 如果服务器密钥存储已存在，则继续下一步；否则，创建服务器密钥存储。

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

2. 如果已经导出了服务器证书，则继续下一步；否则，导出服务器证书。

```
$ keytool -exportcert  -keystore server.keystore.jks -alias localhost -keypass secret -
storepass secret -file server.cer
```

3. 将服务器证书导入到客户端的信任存储中。

```
$ keytool -importcert -keystore client.truststore.jks -storepass secret -alias localhost -
trustcacerts -file server.cer
```

4. 在 **example-security.xml** 中定义客户端 SSL 上下文。此配置文件包含一个 Elytron **authentication-client**，它定义了用于出站连接的身份验证和 SSL 配置。以下 文件演示了定义客户端 SSL 上下文和密钥存储：

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

<configuration>
    <authentication-client xmlns="urn:elytron:client:1.2">
        <key-stores>
            <key-store name="clientStore" type="jks" >
                <file name="/path/to/client.truststore.jks"/>
                <key-store-clear-password password="secret" />
            </key-store>
        </key-stores>
        <ssl-contexts>
            <ssl-context name="client-SSL-context">
                <trust-store key-store-name="clientStore" />
            </ssl-context>
        </ssl-contexts>
        <ssl-context-rules>
            <rule use-ssl-context="client-SSL-context" />
        </ssl-context-rules>
    </authentication-client>
</configuration>
```

5. 使用管理 CLI，引用新创建的文件并尝试访问服务器。以下命令访问管理界面并执行 **whoami** 命令。

```
$ EAP_HOME/bin/jboss-cli.sh -c --controller=remote+https://127.0.0.1:9993 -
Dwildfly.config.url=/path/to/example-security.xml :whoami
```

A.6. ELYTRON 客户端方双向示例

配置服务器 SSL 上下文后，如果可能，务必要测试配置。Elytron 客户端 SSL 上下文可以放在配置文件中，然后从管理 CLI 执行，从而能够测试服务器配置。这些步骤假定已完成服务器端配置，并且服务器已重新加载（如有必要）。

1. 如果服务器和客户端密钥存储已存在，则继续下一步；否则，创建服务器和客户端密钥存储。

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

2. 如果已经导出了服务器和客户端证书，则继续下一步；否则，导出服务器和客户端证书。

```
$ keytool -exportcert  -keystore server.keystore.jks -alias localhost -keypass secret -
storepass secret -file server.cer
$ keytool -exportcert  -keystore client.keystore.jks -alias client -keypass secret -storepass
secret -file client.cer
```

3. 将服务器证书导入到客户端的信任存储中。

```
$ keytool -importcert -keystore client.truststore.jks -storepass secret -alias localhost -
trustcacerts -file server.cer
```

4. 将客户端证书导入到服务器的信任存储中。

```
$ keytool -importcert -keystore server.truststore.jks -storepass secret -alias client -
trustcacerts -file client.cer
```

5. 在 **example-security.xml** 中定义客户端 SSL 上下文。此配置文件包含一个 Elytron **authentication-client**，它定义了用于出站连接的身份验证和 SSL 配置。以下 文件演示了定义客户端 SSL 上下文和密钥存储：

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

<configuration>
    <authentication-client xmlns="urn:elytron:client:1.2">
        <key-stores>
            <key-store name="clientStore" type="jks" >
                <file name="/path/to/client.truststore.jks"/>
                <key-store-clear-password password="secret" />
            </key-store>
        </key-stores>
        <key-store name="clientKeyStore" type="jks" >
            <file name="/path/to/client.keystore.jks"/>
            <key-store-clear-password password="secret" />
        </key-store>
        <ssl-contexts>
            <ssl-context name="client-SSL-context">
                <trust-store key-store-name="clientStore" />
                <key-store-ssl-certificate key-store-name="clientKeyStore" alias="client">
                    <key-store-clear-password password="secret" />
                </key-store-ssl-certificate>
            </ssl-context>
        </ssl-contexts>
        <ssl-context-rules>
            <rule use-ssl-context="client-SSL-context" />
        </ssl-context-rules>
    </authentication-client>
</configuration>
```

6. 使用管理 CLI，引用新创建的文件并尝试访问服务器。以下命令访问管理界面并执行 **whoami** 命令。

```
$ EAP_HOME/bin/jboss-cli.sh -c --controller=remote+https://127.0.0.1:9993 -
Dwildfly.config.url=/path/to/example-security.xml :whoami
```

更新于 2024-02-09