Red Hat JBoss Enterprise Application Platform 8.0

在 JBoss EAP 中使用单点登录

使用单点登录为 JBoss EAP 上部署的应用程序添加身份验证指南

Last Updated: 2024-06-30

Red Hat JBoss Enterprise Application Platform 8.0 在 JBoss EAP 中使用单点登录

使用单点登录为 JBoss EAP 上部署的应用程序添加身份验证指南

法律通告

摘要

使用单点登录为 JBoss EAP 上部署的应用添加身份验证指南。

目录

提供有关 JBOSS EAP 文档的反馈

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

流程

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

使开源包含更多

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

第 1 章 JBOSS EAP 中的单点登录

单点登录(SSO)是从中央身份提供程序的多个客户端验证身份的过程。例如，用户只需要一组登录凭据才能登录使用同一 SSO 提供程序的不同应用。

JBoss EAP 支持以下 SSO 协议：

OpenID Connect(OIDC)

OpenID Connect 是基于 RFC 6749 和 RFC 6750 中指定的规范的 OAuth 2.0 框架的身份验证协议。

安全断言标记语言 v2 (SAML v2)

SAML 是一种数据格式和协议，它允许在双方之间交换身份验证和授权信息，通常是身份提供程序和服务提供商。此信息以包含断言的 SAML 令牌的形式交换，并由身份提供程序发布，以向服务提供程序进行身份验证。主题可以重复使用由带有多个服务提供商的身份提供程序发布的 SAML 令牌，支持 SAML v2 中的基于浏览器的单点登录。

您可以使用 SSO 来保护在裸机上运行的 JBoss EAP 上部署的应用程序，以及 Red Hat OpenShift Container Platform 上运行的 JBoss EAP。有关使用 SSO 保护在 Red Hat OpenShift Container Platform 上运行的 JBoss EAP 上运行的应用程序的信息，请参阅在 OpenShift Container Platform 上使用 JBoss EAP。

第 2 章 使用单点登录保护在 JBOSS EAP 上部署的应用程序

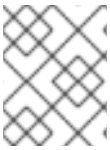
您可以使用单点登录(SSO)保护应用，将身份验证委派给 SSO-provider，如红帽构建的 Keycloak。您可以使用 OpenID Connect (OIDC)或安全断言标记语言 v2 (SAML v2)作为 SSO 协议。

要使用 SSO 保护应用程序，请按照以下步骤执行：

- 创建一个示例应用，以使用单点登录进行安全 ：使用此流程创建一个简单的 web-application 以通过 SSO 保护。如果您已有使用 SSO 保护的应用，请跳过这一步。
- 在 Red Hat build of Keycloak 中创建一个 realm 和用户
- 使用 OIDC 或 SAML 协议使用 SSO 保护应用程序：
  - 使用 OIDC 安全应用程序
  - 使用 SAML 保护应用程序

2.1. 创建示例应用程序以使用单点登录安全

创建一个 web 应用程序以在 JBoss EAP 上部署，并使用带有 OpenID Connect (OIDC)或安全断言标记语言(SAML)的单点登录(SSO)加以保护。



注意

以下流程仅作为示例提供。如果您已有一个要保护的应用程序，您可以跳过这些，并直接进入 在红帽构建的 Keycloak 中创建域和用户。

2.1.1. 为 web 应用程序开发创建一个 Maven 项目

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

重要

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

先决条件

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

流程

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

   语法

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

Example

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

2. 进入到应用程序根目录：

语法

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

Example

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

3. 将生成的 **pom.xml** 文件的内容替换为以下文本：

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

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

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

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

    <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <maven.compiler.source>11</maven.compiler.source>
      <maven.compiler.target>11</maven.compiler.target>
      <version.maven.war.plugin>3.4.0</version.maven.war.plugin>
    </properties>

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

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

验证

- 在应用程序根目录中，输入以下命令：

```
$ mvn install
```

  您会看到类似如下的输出：

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

后续步骤

- 创建一个 Web 应用程序

2.1.2. 创建一个 Web 应用程序

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

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

先决条件

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

流程

1. 创建一个用于存储 Java 文件的目录。

语法

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

Example

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

2. 前往新目录。

语法

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

Example

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

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

```
package com.example.app;

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

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

/**
 * A simple secured HTTP servlet. It returns the user name of obtained
 * from the logged-in user's Principal. If there is no logged-in user,
 * it returns the text "NO AUTHENTICATED USER".
 */

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

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        try (PrintWriter writer = resp.getWriter()) {
            writer.println("<html>");
            writer.println("  <head><title>Secured Servlet</title></head>");
            writer.println("  <body>");
            writer.println("    <h1>Secured Servlet</h1>");
            writer.println("    <p>");
            writer.print(" Current Principal '");
            Principal user = req.getUserPrincipal();
            writer.print(user != null ? user.getName() : "NO AUTHENTICATED USER");
            writer.print("'");
```

```
            writer.println("    </p>");
            writer.println("  </body>");
            writer.println("</html>");
        }
    }

}
```

4. 在应用程序根目录中，使用以下命令编译应用程序：

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

5. 部署应用。

```
$ mvn wildfly:deploy
```

验证

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

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

  因为没有添加验证机制，所以您可以访问应用程序。

后续步骤

- 在红帽构建的 Keycloak 中创建域和用户

2.2. 在红帽构建的 KEYCLOAK 中创建域和用户

红帽构建的 Keycloak 中的域等同于一个租户。每个域都允许管理员创建隔离的应用程序和用户组。

以下流程概述了使用红帽构建的 Keycloak 部署至 JBoss EAP 的应用程序以进行测试所需的最小步骤。有关详细配置，请参阅 红帽构建的 Keycloak 服务器管理指南。

注意

以下流程仅作为示例提供。如果您已经在 Red Hat build of Keycloak 中配置了 realm 和 用户，您可以跳过这个过程并直接进行安全应用程序。如需更多信息，请参阅：

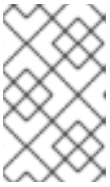
- 使用 OIDC 保护应用程序
- 使用 SAML 保护应用程序

先决条件

- 具有红帽构建的 Keycloak 的管理员访问权限。

流程

1. 在 8080 之外的端口启动红帽 Keycloak 服务器构建，因为 JBoss EAP 默认端口为 8080。



注意

start-dev 命令不适用于生产环境。如需更多信息，请参阅 Red Hat build of Keycloak Server Guide 中的 Trying Red Hat build of Keycloak in development mode。

语法

```
$ <path_to_rhbk>/bin/kc.sh start-dev --http-port <offset-number>
```

Example

```
$ /home/servers/rhbk-22.0/bin/kc.sh start-dev --http-port 8180
```

2. 登录到位于 **http://localhost:<port>/ 的管理控制台**。例如： **http://localhost:8180/**。
3. 创建 realm。
   a. 将鼠标悬停在 **Master** 上，然后单击 **Create Realm**。
   b. 输入 realm 的名称。例如： **example_realm**。
   c. 确保 **Enabled** 设置为 **ON**。
   d. 点 **Create**。

   如需更多信息，请参阅 红帽构建的 Keycloak 服务器管理指南中的创建域。
4. 创建用户。
   a. 单击 **Users**，然后单击 **Add user**,
   b. 输入用户名。例如，**user1**。
   c. 点 **Create**。

   如需更多信息，请参阅 Red Hat build of Keycloak Server Administration Guide 中的 创建用户。
5. 为用户设置凭据。
   a. 单击 **Credentials**。
   b. 设置用户的密码。例如，**passwordUser1**。将 **Temporary** 切换到 **OFF**，然后单击 **Set Password**。在确认提示中，单击 **Save**。

   如需更多信息，请参阅 Red Hat build of Keycloak Server Administration Guide 中的 定义用户凭证。

6. 创建角色。
   这是您在 JBoss EAP 中配置用于授权的角色名称。

   a. 单击 **Realm Roles**，然后单击 **Create role**。

   b. 输入角色名称，如 *Admin*。

   c. 点击 **Save**。

7. 将角色分配给用户。

   a. 点 **Users**。

   b. 单击要为其分配角色的用户。

   c. 单击 **Role Mapping**。

   d. 单击 **Assign role**。

   e. 选择要分配的角色。例如，*管理员*。单击 **Assign**。

   如需更多信息，请参阅红帽构建的 Keycloak 服务器管理指南中的创建域角色。

后续步骤

- 要使用此域来保护部署到 JBoss EAP 的应用，请按照以下步骤操作：
  - 使用 OIDC 保护应用程序
  - 使用 SAML 保护应用程序

其他资源

- 红帽构建的 Keycloak 入门指南

2.3. 使用 OIDC 保护应用程序

使用 JBoss EAP 原生 OpenID Connect (OIDC)客户端来利用外部 OpenID 供应商保护应用程序。OIDC 是一个身份层，它允许客户端（如 JBoss EAP）根据 OpenID 供应商执行的身份验证来验证用户的身份。例如，您可以使用红帽构建的 Keycloak 作为 OpenID 供应商来保护 JBoss EAP 应用程序。

要使用 OIDC 保护应用程序，请按照以下步骤执行：

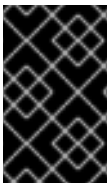
- 在 JBoss EAP 中创建 OIDC 客户端
- 使用 OpenID Connect 保护 Web 应用程序

2.3.1. 在 JBoss EAP 中使用 OpenID Connect 的应用安全性

当您使用 OpenID 供应商保护应用程序时，您不需要在本地配置任何安全域资源。**elytron-oidc-client** 子系统在 JBoss EAP 中提供原生 OpenID Connect (OIDC)客户端，以与 OpenID 提供程序(OP)连接。JBoss EAP 根据您的 OpenID 提供程序配置自动为您的应用程序创建虚拟安全域。**elytron-oidc-client** 子系统充当 Relying party (RP)。

注意

JBoss EAP 原生 OIDC 客户端不支持 RP-Initiated logout。



重要

建议您在 Red Hat build of Keycloak 中使用 OIDC 客户端。如果可将其他 OpenID 供应商配置为使用 JSON Web 令牌(JWT)的访问令牌，并可配置为使用 RS256、RS384、RS512、ES256、ES384 或 ES512 签名算法。

要启用 OIDC 的使用，您可以配置 **elytron-oidc-client** 子系统或应用程序本身。JBoss EAP 激活 OIDC 身份验证，如下所示：

- 当您将应用程序部署到 JBoss EAP 时，**elytron-oidc-client** 子系统会扫描部署，以检测是否需要 OIDC 身份验证机制。
- 如果子系统在 **elytron-oidc-client** 子系统或应用程序部署描述符中检测到部署的 OIDC 配置，JBoss EAP 为应用启用 OIDC 身份验证机制。
- 如果子系统在两个位置检测到 OIDC 配置，则 **elytron-oidc-client** 子系统 **secure-deployment** 属性中的配置优先于应用程序部署描述符中的配置。

部署配置

要使用部署描述符使用 OIDC 保护应用程序，请更新应用程序的部署配置，如下所示：

- 在应用程序部署描述符 **web.xml** 文件中将 **auth-method** 属性设置为 **OIDC**。

部署描述符更新示例

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

- 在 **WEB-INF** 目录中创建一个名为 **oidc.json** 的文件，其中包含 OIDC 配置信息。

  oidc.json 内容示例

```
{
  "client-id" : "customer-portal", (1)
  "provider-url" : "http://localhost:8180/realms/demo", (2)
  "ssl-required" : "external", (3)
   "credentials" : {
      "secret" : "234234-234234-234234" (4)
   }
}
```

(1) 使用 OpenID 供应商标识 OIDC 客户端的名称。

(2) OpenID 提供程序 URL。

(3) 对外部请求需要 HTTPS。

(4) 与 OpenID 供应商注册的客户端 secret。

子系统配置
您可以通过使用以下方法配置 **elytron-oidc-client** 子系统来保护使用 OIDC 的应用程序：

- 如果您为每个应用程序使用相同的 OpenID 供应商，请为多个部署创建一个单一配置。
- 如果您将不同的 OpenID 供应商用于不同的应用程序，请为每个部署创建不同的配置。

单个部署的 XML 配置示例：

```
<subsystem xmlns="urn:wildfly:elytron-oidc-client:1.0">
  <secure-deployment name="DEPLOYMENT_RUNTIME_NAME.war"> 1
    <client-id>customer-portal</client-id> 2
    <provider-url>http://localhost:8180/realms/demo</provider-url> 3
    <ssl-required>external</ssl-required> 4
    <credential name="secret" secret="0aa31d98-e0aa-404c-b6e0-e771dba1e798" /> 5
  </secure-deployment
</subsystem>
```

1. 部署运行时名称。
2. 使用 OpenID 供应商标识 OIDC 客户端的名称。
3. OpenID 提供程序 URL。
4. 对外部请求需要 HTTPS。
5. 与 OpenID 供应商注册的客户端 secret。

要使用同一 OpenID 供应商保护多个应用程序，请单独配置 **提供程序**，如下例所示：

```
<subsystem xmlns="urn:wildfly:elytron-oidc-client:1.0">
  <provider name="${OpenID_provider_name}">
    <provider-url>http://localhost:8080/realms/demo</provider-url>
    <ssl-required>external</ssl-required>
  </provider>
  <secure-deployment name="customer-portal.war"> 1
    <provider>${OpenID_provider_name}</provider>
    <client-id>customer-portal</client-id>
    <credential name="secret" secret="0aa31d98-e0aa-404c-b6e0-e771dba1e798" />
  </secure-deployment>
  <secure-deployment name="product-portal.war"> 2
    <provider>${OpenID_provider_name}</provider>
    <client-id>product-portal</client-id>
    <credential name="secret" secret="0aa31d98-e0aa-404c-b6e0-e771dba1e798" />
  </secure-deployment>
</subsystem>
```

1. 一个部署： **customer-portal.war**
2. 另一个部署： **product-portal.war**

其他资源

- OpenID Connect 规格
- **Elytron-oidc-client** 子系统属性
- OpenID Connect Libraries

2.3.2. 在红帽构建的 Keycloak 中创建 OIDC 客户端

在红帽构建的 Keycloak 中创建 OpenID Connect (OIDC)客户端，以用于 JBoss EAP 来保护应用程序。

以下流程概述了使用红帽构建的 Keycloak 部署至 JBoss EAP 的应用程序以进行测试所需的最小步骤。有关详细配置，请参阅红帽构建的 Keycloak 服务器 管理指南中的管理 OpenID Connect 客户端。

先决条件

- 您已在红帽构建的 Keycloak 中创建域并定义了用户。
  如需更多信息，请参阅在 JBoss EAP 中创建域和用户

流程

1. 导航到红帽构建的 Keycloak Admin 控制台。
2. 创建客户端。
   a. 单击 **Clients**，然后单击 **Create client**。
   b. 确保将 **Client type** 设置为 **OpenID Connect**。
   c. 输入客户端 ID。例如，**jbeap-oidc**。
   d. 点击 **Next**。
   e. 在 **Capability Config** 选项卡中，确保将 **Authentication Flow** 设置为 **Standard 流**，并且 **Direct access 授予**。
   f. 点击 **Next**。
   g. 在 **Login settings** 选项卡中，输入 **Valid redirect URIs** 的值。输入页面在成功身份验证后应重定向的 URL，例如 **http://localhost:8080/simple-webapp-example/secured/***。
   h. 点击 **Save**。
3. 查看适配器配置。
   a. 点 **Action**，然后点 **Download adapter config**。
   b. 选择 **Keycloak OIDC JSON** 作为 **Format Option** 来查看连接参数。

```
{
  "realm": "example_realm",
  "auth-server-url": "http://localhost:8180/",
  "ssl-required": "external",
  "resource": "jbeap-oidc",
  "public-client": true,
  "confidential-port": 0
}
```

在将 JBoss EAP 应用程序配置为使用红帽构建的 Keycloak 作为身份提供程序时，您可以使用参数，如下所示：

```
"provider-url" : "http://localhost:8180/realms/example_realm",
"ssl-required": "external",
"client-id": "jbeap-oidc",
"public-client": true,
"confidential-port": 0
```

后续步骤

- 使用 OpenID Connect 保护 Web 应用程序

其他资源

- 保护应用程序和服务指南

2.3.3. 使用 OpenID Connect 保护 Web 应用程序

您可以通过更新其部署配置或配置 **elytron-oidc-client 子系统来保护应用**。

如果您使用流程中创建的应用程序，创建 Web 应用程序，Principal 的值来自 OpenID 供应商中的 ID 令牌。默认情况下，Principal 是令牌中的 "sub" 声明的值。您还可以使用 "email", "preferred_username", "name", "given_name", "family_name", 或 "nickname" 声明作为主体。指定在以下位置之一将 ID 令牌中的声明值用作主体：

- **elytron-oidc-client** 子系统属性 **principal-attribute**。
- **oidc.json 文件**。

您可以将应用程序配置为使用 OIDC 的方法有两种：

- 通过配置 **elytron-oidc-client** 子系统。
  如果您不想在应用程序部署中添加配置，请使用此方法。
- 通过更新部署配置
  如果您不想向服务器添加配置，并更喜欢在应用程序部署中保留配置，则使用此方法。

先决条件

- 您已在 JBoss EAP 上部署了应用程序。

流程

1. 配置应用程序的 **web.xml** 以保护应用程序资源。

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

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">
```

```
    <security-constraint>
      <web-resource-collection>
        <web-resource-name>secured</web-resource-name>
        <url-pattern>/secured</url-pattern>
      </web-resource-collection>

      <auth-constraint>
        <role-name>Admin</role-name>
      </auth-constraint>
    </security-constraint>

    <security-role>
      <role-name>*</role-name>
    </security-role>
</web-app>
```

1 仅允许具有 **Admin** 角色的用户访问应用。要允许具有任何角色的用户访问应用程序，请使用通配符 ** 作为 **role-name** 的值。

2. 要使用 OpenID Connect 保护应用，可以更新部署配置或配置 **elytron-oidc-client** 子系统。

注意

如果您在部署配置和 **elytron-oidc-client** 子系统中配置 OpenID Connect，则 **elytron-oidc-client** 子系统 **secure-deployment** 属性中的配置优先于应用部署描述符中的配置。

- 更新部署配置。

  i. 在应用程序的 **web.xml** 中添加登录配置，并将验证方法指定为 OIDC。

```
<web-app>
...
    <login-config>
        <auth-method>OIDC</auth-method>
    </login-config>
...
</web-app>
```

1 使用 OIDC 保护应用程序的安全。

  ii. 在 **WEB-INF** 目录中创建 **oidc.json** 文件，如下所示：

```
{
  "provider-url" : "http://localhost:8180/realms/example_realm",
  "ssl-required": "external",
  "client-id": "jbeap-oidc",
  "public-client": true,
  "confidential-port": 0
}
```

- 配置 **elytron-oidc-client** 子系统：

- 要保护应用程序，请使用以下管理 CLI 命令：

```
/subsystem=elytron-oidc-client/secure-deployment=simple-oidc-example.war/:add(client-id=jbeap-oidc,provider-url=http://localhost:8180/realms/example_realm,public-client=true,ssl-required=external)
```

3. 在应用程序根目录中，使用以下命令编译应用程序：

```
$ mvn package
```

4. 部署应用。

```
$ mvn wildfly:deploy
```

验证

1. 在浏览器中，导航到 **http://localhost:8080/simple-webapp-example/secured**。
   您将被重定向到红帽构建的 Keycloak 登录页面。
2. 您可以使用您在红帽构建的 Keycloak 中定义的用户凭证登录。

您的应用程序现在使用 OIDC 进行保护。

其他资源

- **Elytron-oidc-client** 子系统属性

2.4. 使用 SAML 保护应用程序

您可以使用 Keycloak SAML 适配器功能 pack 提供的 Galleon 层使用安全断言标记语言(SAML)来保护 Web 应用程序。

有关 Keycloak SAML 适配器功能 pack 的详情，请参考 Keycloak SAML 适配器功能 pack 来使用 SAML 保护应用程序。

要使用 SAML 保护应用程序，请按照以下步骤执行：

- 使用 SAML 保护 Web 应用程序

2.4.1. 在 JBoss EAP 中使用 SAML 的应用程序安全性

Keycloak SAML 适配器 Galleon pack 是一个 Galleon 功能 pack，包含三个层： **keycloak-saml**、**keycloak-client-saml**、**keycloak-client-saml-ejb**。使用功能软件包中的层在 JBoss EAP 中安装必要的模块和配置，以使用红帽构建的 Keycloak 作为使用安全断言标记语言(SAML)的单点登录。

下表描述了每个层的用例。

layer	适用于	描述

layer	适用于	描述
keycloak-saml	OpenShift	将此层用于 Source 到镜像(s2i)，并自动注册 SAML 客户端。您必须将这个层与 **cloud-default-config** 层一起使用。
keycloak-client-saml	裸机、OpenShift	将此层用于裸机上的 Web 应用程序，以及 CLI 脚本或部署配置中提供的 **keycloak-saml** 子系统配置的 Source 到镜像(s2i)。
keycloak-client-saml-ejb	裸机	将这个层用于您要传播身份到 Jakarta Enterprise Beans 的应用。

要启用 SAML 的使用，您可以配置 **keycloak-saml** 子系统或应用程序本身。

部署配置
要使用部署描述符使用 SAML 保护应用程序，请更新应用程序的部署配置，如下所示：

- 在应用部署描述符 **web.xml** 文件中，将 **auth-method** 属性设置为 **SAML**。

  部署描述符更新示例

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

- 使用 SAML 配置信息，在 **WEB-INF** 目录中创建一个名为 **keycloak-saml.xml** 的文件。您可以从 SAML 供应商获取此文件。

  keycloak-saml.xml示例

```
<keycloak-saml-adapter>
    <SP entityID=""
        sslPolicy="EXTERNAL"
        logoutPage="SPECIFY YOUR LOGOUT PAGE!">
        <Keys>
            <Key signing="true">
                <PrivateKeyPem>PRIVATE KEY NOT SET UP OR KNOWN</PrivateKeyPem>
                <CertificatePem>...</CertificatePem>
            </Key>
        </Keys>
        <IDP entityID="idp"
             signatureAlgorithm="RSA_SHA256"
             signatureCanonicalizationMethod="http://www.w3.org/2001/10/xml-exc-c14n#">
            <SingleSignOnService signRequest="true"
                                 validateResponseSignature="true"
                                 validateAssertionSignature="false"
                                 requestBinding="POST"

bindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"/>
```

```
            <SingleLogoutService signRequest="true"
                                 signResponse="true"
                                 validateRequestSignature="true"
                                 validateResponseSignature="true"
                                 requestBinding="POST"
                                 responseBinding="POST"

postBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"

redirectBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"/>
        </IDP>
    </SP>
</keycloak-saml-adapter>
```

PrivateKeyPem 的值和 **CertificatePem** 对每个客户端都是唯一的。

子系统配置
您可以通过配置 **keycloak-saml** 子系统来保护使用 SAML 的应用程序。您可以从红帽构建的 Keycloak 获取包含子系统配置命令的客户端配置文件。如需更多信息，请参阅生成客户端适配器配置。

2.4.2. 在红帽构建的 Keycloak 中创建 SAML 客户端

在红帽构建的 Keycloak 中创建一个安全断言标记语言(SAML)客户端，以用于 JBoss EAP 来保护应用程序。

以下流程概述了使用红帽构建的 Keycloak 部署至 JBoss EAP 的应用程序以进行测试所需的最小步骤。有关详细配置，请参阅红帽构建的 Keycloak 服务器管理指南中的创建 SAML 客户端。

先决条件

- 您已在红帽构建的 Keycloak 中创建域并定义了用户。
  如需更多信息，请参阅在 JBoss EAP 中创建域和用户

流程

1. 导航到红帽构建的 Keycloak Admin 控制台。
2. 创建客户端。
   a. 单击 **Clients**，然后单击 **Create client**。
   b. 选择 **SAML** 作为客户端 类型。
   c. 输入您要保护为客户端 **ID** 的应用程序的 URL。例如： **http://localhost:8080/simple-webapp-example/secured/**。

重要

客户端 ID 必须与应用程序的 URL 完全匹配。如果客户端 ID 不匹配，您会收到类似如下的错误：

```
2023-05-17 19:54:31,586 WARN  [org.keycloak.events] (executor-thread-0) type=LOGIN_ERROR, realmId=eba0f106-389f-4216-a676-05fcd0c0c72e, clientId=null, userId=null, ipAddress=127.0.0.1, error=client_not_found, reason=Cannot_match_source_hash
```

d. 输入客户端名称。例如，**jbeap-saml**。

e. 点击 **Next**。

f. 输入以下信息：

- **根 URL** ：应用程序的 URL，例如 **http://localhost:8080/simple-webapp-example/**。
- **主页 URL** ：应用程序的 URL，例如 **http://localhost:8080/simple-webapp-example/**。

重要

如果您没有设置 Home URL，客户端配置中的 **SP entityID** 会保持空白，并导致错误。

- 如果使用管理 CLI 命令，您会收到以下错误：

```
Can't reset to root in the middle of the path @72
```

您可以通过在对应的配置文件中定义 **SP entityID** 的值来解决错误。

- **有效的 Redirect** URI：用户登录后允许的 URI，例如 **http://localhost:8080/simple-webapp-example/secured/***。
- **Master SAML 处理 URL**：应用程序的 URL 后跟 **saml**。例如：**http://localhost:8080/simple-webapp-example/saml**。

重要

如果您没有将 **saml** 附加到 URL，您会收到重定向错误。

如需更多信息，请参阅创建 SAML 客户端。

现在，您可以使用配置的客户端来保护 JBoss EAP 上部署的 Web 应用。如需更多信息，请参阅使用 SAML 保护 Web 应用程序。

后续步骤

- 使用 SAML 保护 Web 应用程序

其他资源

- 红帽构建的 Keycloak 服务器管理指南

2.4.3. 使用 SAML 保护 Web 应用程序

Keycloak SAML 适配器功能 pack 为非 OpenShift 部署提供两个层： **keycloak-client-saml** 和 **keycloak-client-saml-ejb**。使用 **keycloak-client-saml** 层来保护 servlet based-web 应用，以及 **keycloak-client-saml-ejb** 来保护 Jakarta Enterprise Beans 应用。

您可以将应用程序配置为使用 SAML 的方法有两种：

- 通过配置 **keycloak-saml** 子系统。
  如果您不想在应用程序部署中添加配置，请使用此方法。

- 通过更新部署配置
  如果您不想向服务器添加配置，并更喜欢在应用程序部署中保留配置，则使用此方法。

先决条件

- 红帽构建的 Keycloak 中已创建了 SAML 客户端。
  如需更多信息，请参阅在 红帽构建的 Keycloak 中创建 SAML 客户端。
- 使用 **jboss-eap-installation-manager** 已安装了 JBoss EAP。
  如需更多信息，请参阅 *Red Hat JBoss Enterprise Application Platform 安装方法指南中的使用* ***jboss-eap-installation-manager****安装 JBoss EAP 8.0*。

流程

1. 使用 **jboss-eap-installation-manager** 将所需的 Keycloak SAML 适配器层添加到服务器。以下是可用层的详情：
   - feature pack: **org.keycloak:keycloak-saml-adapter-galleon-pack**.
   - 层 ：
     - **keycloak-client-saml** ：使用此层来保护 servlet。
     - **keycloak-client-saml-ejb** ：使用此层将身份从 servlet 传播到 Jakarta Enterprise Beans。
       有关在 JBoss EAP 中添加功能包和层的信息，请参阅 *Red Hat JBoss Enterprise Application Platform 安装方法指南中的使用 jboss-eap-installation-manager 向现有 JBoss EAP 服务器* 添加功能包和层。
2. 配置应用程序的 **web.xml** 以保护应用程序资源。

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

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    metadata-complete="false">

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

        <auth-constraint>
            <role-name>Admin</role-name>  (1)
        </auth-constraint>
    </security-constraint>

    <security-role>
        <role-name>*</role-name>
    </security-role>
</web-app>
```

1 仅允许具有 **Admin** 角色的用户访问应用。要允许具有任何角色的用户访问应用程序，请使用通配符 ** 作为 **role-name** 的值。

3. 使用管理 CLI 或更新应用部署，使用 SAML 保护应用程序。

- 通过更新应用部署。

a. 在应用程序的 **web.xml** 中添加登录配置，并将身份验证方法指定为 SAML。

```
<web-app>
...
    <login-config>
        <auth-method>SAML</auth-method>  1
    </login-config>
...
</web-app>
```

1 使用 SAML 保护应用程序。

b. 从红帽构建的 Keycloak 下载配置 **keycloak-saml.xml** 文件，并将它保存到应用程序的 **WEB-INF/** 目录中。
如需更多信息，请参阅生成客户端适配器配置。

keycloak-saml.xml示例

```
<keycloak-saml-adapter>
    <SP entityID=""
        sslPolicy="EXTERNAL"
        logoutPage="SPECIFY YOUR LOGOUT PAGE!">
        <Keys>
            <Key signing="true">
                <PrivateKeyPem>PRIVATE KEY NOT SET UP OR
KNOWN</PrivateKeyPem>
                <CertificatePem>...</CertificatePem>
            </Key>
        </Keys>
        <IDP entityID="idp"
             signatureAlgorithm="RSA_SHA256"
             signatureCanonicalizationMethod="http://www.w3.org/2001/10/xml-exc-
c14n#">
            <SingleSignOnService signRequest="true"
                                 validateResponseSignature="true"
                                 validateAssertionSignature="false"
                                 requestBinding="POST"

bindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"/>
            <SingleLogoutService signRequest="true"
                                 signResponse="true"
                                 validateRequestSignature="true"
                                 validateResponseSignature="true"
                                 requestBinding="POST"
                                 responseBinding="POST"

postBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"
```

```
redirectBindingUrl="http://localhost:8180/realms/example_saml_realm/protocol/saml"
/>
        </IDP>
    </SP>
</keycloak-saml-adapter>
```

PrivateKeyPem 的值和 **CertificatePem** 对每个客户端都是唯一的。

- 通过使用管理 CLI。

  a. 从红帽构建的 Keycloak 下载客户端配置文件 **keycloak-saml-subsystem.cli**。
  如需更多信息，请参阅生成客户端适配器配置。

  keycloak-saml-subsystem.cli示例

```
/subsystem=keycloak-saml/secure-deployment=YOUR-WAR.war/:add

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-
example/"/:add(sslPolicy=EXTERNAL,logoutPage="SPECIFY YOUR LOGOUT
PAGE!"

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-
example/"/Key=KEY1:add(signing=true, \
PrivateKeyPem="...", CertificatePem="...")

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-example/"/IDP=idp/:add( \
    SingleSignOnService={ \
        signRequest=true, \
        validateResponseSignature=true, \
        validateAssertionSignature=false, \
        requestBinding=POST, \
        bindingUrl=http://localhost:8180/realms/example-saml-realm/protocol/saml}, \
    SingleLogoutService={ \
        signRequest=true, \
        signResponse=true, \
        validateRequestSignature=true, \
        validateResponseSignature=true, \
        requestBinding=POST, \
        responseBinding=POST, \
        postBindingUrl=http://localhost:8180/realms/example-saml-realm/protocol/saml,
\
        redirectBindingUrl=http://localhost:8180/realms/example-saml-
realm/protocol/saml} \
)

/subsystem=keycloak-saml/secure-deployment=YOUR-
WAR.war/SP="http://localhost:8080/simple-webapp-example/"/IDP=idp/:write-
attribute(name=signatureAlgorithm,value=RSA_SHA256)

/subsystem=keycloak-saml/secure-deployment=YOUR-
```

```
WAR.war/SP="http://localhost:8080/simple-webapp-example/"/IDP=idp/:write-attribute(name=signatureCanonicalizationMethod,value=http://www.w3.org/2001/10/xml-exc-c14n#)
```

PrivateKeyPem 的值和 **CertificatePem** 对每个客户端都是唯一的。

b. 使用应用程序 **WAR 的名称（如 simple-webapp-example.war ）**更新客户端配置文件中出现的每个 WAR.war。

注意

生成的 CLI 脚本在第二个语句的末尾缺少 ） ：

```
/subsystem=keycloak-saml/secure-deployment=YOUR-WAR.war/SP="/:add(sslPolicy=EXTERNAL,logoutPage="SPECIFY YOUR LOGOUT PAGE!"
```

您必须添加缺少的 ）

c. 使用管理 CLI 运行 **keycloak-saml-subsystem.cli** 脚本来配置 JBoss EAP。

```
$ <EAP_HOME>/bin/jboss-cli.sh -c --file=<path_to_the_file>/keycloak-saml-subsystem.cli
```

4. 部署应用。

```
$ mvn wildfly:deploy
```

验证

1. 在浏览器中，导航到应用 URL。例如： **http://localhost:8080/simple-webapp-example/secured**。
   您将被重定向到红帽构建的 Keycloak 登录页面。
2. 您可以使用您在红帽构建的 Keycloak 中定义的用户凭证登录。

您的应用程序现在使用 SAML 进行保护。

其他资源

- 用于使用 SAML 保护应用程序的 Keycloak SAML 适配器功能 pack

第 3 章 在使用 OIDC 时将身份从 SERVLET 传播到 JAKARTA ENTERPRISE BEAN

您可以通过两种方式将从 OpenID Connect (OIDC)供应商从 Servlet 获取的安全身份传播到 Jakarta Enterprise Beans：

- 使用相同的虚拟安全域来保护 Servlet 和 Jakarta Enterprise Beans。
- 将身份从与 Servlet 关联的虚拟安全域传播到保护 Jakarta Enterprise Beans 的安全域。

3.1. 使用 OIDC 时的身份传播到 JAKARTA ENTERPRISE BEANS

当您使用 OpenID Connect (OIDC)保护应用程序时，**elytron-oidc-client** 子系统会自动为您创建一个虚拟安全域。您可以将虚拟安全域中的安全身份（从 OIDC 供应商获取的）传播到应用程序调用的 Jakarta Enterprise Beans。

下表根据您使用的安全域以及应用的部署方式，演示了所需的配置。

用于保护 Jakarta Enterprise Beans 的安全域	Servlet 和 Jakarta Enterprise Beans 位于同一 WAR 或 EAR 中	Servlet 和 Jakarta Enterprise Beans 位于不同的 WAR 或 EAR 中
虚拟安全域	不需要配置。 虚拟安全域会自动将安全身份流传输到 Jakarta Enterprise Beans，只要没有为 Jakarta Enterprise Beans 明确指定安全域配置。	配置如下： ● 创建 **virtual-security-domain** 资源。 ● 向 Jakarta Enterprise Beans 添加 **@SecurityDomain** 注释，以引用 **virtual-security-domain** 资源的名称。 如需更多信息，请参阅使用虚拟安全域保护 Jakarta Enterprise Beans 应用程序。
不同的安全域	要将安全身份从虚拟安全域流传输到另一个安全域，您必须配置以下资源： ● **virtual-security-domain** ：指定由虚拟安全域建立的安全身份应自动流传输到其他安全域。 ● **security-domain**: 表示它应该信任您配置的虚拟安全域建立的安全身份。 如需更多信息，请参阅将身份从虚拟安全域传播到安全域。	

3.2. 使用虚拟安全域保护 JAKARTA ENTERPRISE BEANS 应用程序

您可以使用 **elytron-oidc-client** 子系统创建的虚拟安全域来保护 Jakarta Enterprise Beans，当 Jakarta Enterprise Beans 都位于与调用它的 Servlet 的同一部署中时，以及它们位于不同部署中时。

如果 Jakarta Enterprise Beans 位于与 Servlet 调用的同一部署中，则不需要将安全身份从 Servlet 流传输到 Jakarta Enterprise Beans。

按照以下步骤将安全身份从 Servlet 流传输到位于不同部署的 Jakarta Enterprise Beans。

先决条件

- 您已保护了使用 OpenID Connect (OIDC)供应商调用 Jakarta Enterprise Beans 的应用程序。
- 您已创建了 Jakarta Enterprise Beans 来保护。

流程

1. 创建一个 **virtual-security-domain** 资源引用 WAR，其中包含通过 OIDC 保护的 Servlet 或包含使用 OIDC 保护的子部署的 EAR。

   语法

   ```
   /subsystem=elytron/virtual-security-domain=<deployment_name>:add()
   ```

   Example

   ```
   /subsystem=elytron/virtual-security-domain=simple-ear-example.ear:add()
   ```

2. 在引用虚拟安全域资源的 Jakarta Enterprise Beans 应用中添加 **org.jboss.ejb3.annotation.SecurityDomain** 注解，以保护应用。

   语法

   ```
   @SecurityDomain("<deployment_name>")
   ```

   Example

   ```
   ...
   @SecurityDomain("simple-ear-example.ear")
   @Remote(RemoteHello.class)
   @Stateless
   public class RemoteHelloBean implements RemoteHello {

     @Resource
     private SessionContext context;

     @Override
     public String whoAmI() {
       return context.getCallerPrincipal().getName();
     }

   }
   ```

   如果您从带有 OIDC 的 Servlet 安全调用此 Jakarta Enterprise Beans，则 **whoAmI （）** 返回的主体将与从 OIDC 供应商获取的 Servlet 的主体匹配。

3. 部署 Jakarta Enterprise Beans。

Example

```
$ mvn wildfly:deploy
```

其他资源

- **virtual-security-domain** 属性

3.3. 将身份从虚拟安全域传播到安全域

您可以将安全身份从虚拟安全域传播，从 OpenID Connect (OIDC)供应商获取到不同的安全域。如果您希望安全身份的角色由将身份传播到而不是虚拟安全域来确定，您可能希望这样做。

当 Servlet 调用 Jakarta Enterprise Beans 和 Jakarta Enterprise Beans 在同一部署中以及它们位于单独的部署中时，以下步骤也会应用它们。

先决条件

- 您已保护了使用 OIDC 供应商调用 Jakarta Enterprise Beans 的应用程序。
- 您已创建了 Jakarta Enterprise Beans 来保护。
- 您可以使用安全域保护 Jakarta Enterprise Beans。

流程

1. 创建一个 **virtual-security-domain** 资源引用 WAR，其中包含通过 OIDC 保护的 Servlet 或包含使用 OIDC 保护的子部署的 EAR。

   #### 语法

   ```
   /subsystem=elytron/virtual-security-domain=<deployment_name>:add(outflow-security-domains=[<domain_to_propagate_to>])
   ```

   #### Example

   ```
   /subsystem=elytron/virtual-security-domain=simple-ear-example.ear:add(outflow-security-domains=[exampleEJBSecurityDomain])
   ```

2. 更新 Jakarta Enterprise Beans 的安全域配置，以信任 **virtual-security-domain**。

   #### 语法

   ```
   /subsystem=elytron/security-domain=<security_domain_name>:write-attribute(name=trusted-virtual-security-domains,value=[<deployment_name>])
   ```

   #### Example

   ```
   /subsystem=elytron/security-domain=exampleEJBSecurityDomain:write-attribute(name=trusted-virtual-security-domains,value=[simple-ear-example.ear])
   ```

3. 重新加载服务器。

```
reload
```

4. 部署 Jakarta Enterprise Beans。

   Example

   ```
   $ mvn wildfly:deploy
   ```

其他资源

- **security-domain** 属性
- **virtual-security-domain** 属性

第 4 章 使用 OPENID 供应商保护 JBOSS EAP 管理控制台

您可以使用 OIDC 使用外部身份提供程序（如红帽构建的 Keycloak）来保护 JBoss EAP 管理控制台。通过使用外部身份提供程序，您可以将身份验证委派给身份提供程序。

要使用 OIDC 保护 JBoss EAP 管理控制台，请按照以下步骤执行：

- 配置红帽构建的 Keycloak 以保护 JBoss EAP 管理控制台
- 使用 OpenID Connect 保护 JBoss EAP 管理控制台

4.1. 使用 OIDC 的 JBOSS EAP 管理控制台安全性

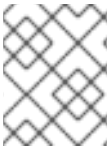
您可以通过配置 OIDC 供应商（如红帽构建的 Keycloak 和 **elytron-oidc-client** 子系统）来保护使用 OpenID Connect (OIDC)的 JBoss EAP 管理控制台。

重要

不支持保护作为带有 OIDC 的受管域运行的 JBoss EAP 的管理控制台。

使用 OIDC 的 JBoss EAP 管理控制台安全性可以正常工作：

- 当您在 **elytron-oidc-client** 子系统中配置 **secure-server** 资源时，JBoss EAP 管理控制台会重定向到用于登录的 OIDC 供应商登录页面。
- 然后，JBoss EAP 使用 **secure-deployment** 资源配置来通过 bearer 令牌身份验证来保护管理接口。



注意

OIDC 依赖于浏览器访问 Web 应用程序。因此，JBoss EAP 管理 CLI 无法通过 OIDC 进行保护。

RBAC 支持

您可以在 OIDC 供应商中配置和分配角色，以便为 JBoss EAP 管理控制台实施基于角色的访问控制 (RBAC)。JBoss EAP 包括或排除 JBoss EAP RBAC 配置中定义的 RBAC 的用户角色。有关 RBAC 的更多信息，请参阅 JBoss EAP 7.4 *安全架构指南中的* 基于角色的访问控制。

其他资源

- 配置红帽构建的 Keycloak 以保护 JBoss EAP 管理控制台
- 使用 OpenID Connect 保护 JBoss EAP 管理控制台

4.2. 配置红帽构建的 KEYCLOAK 以保护 JBOSS EAP 管理控制台

在 OpenID Connect (OIDC)供应商中配置所需的用户、角色和客户端，以保护 JBoss EAP 管理控制台。

使用 OIDC 保护管理控制台需要两个客户端。客户端必须配置如下：

- 为标准流配置的客户端。
- 配置为 bearer-only 客户端的客户端。

以下流程概述了使用 OIDC 为测试目的保护 JBoss EAP 管理控制台所需的最小步骤。有关详细配置，请参阅 红帽构建的 Keycloak 文档。

先决条件

- 具有红帽构建的 Keycloak 的管理员访问权限。
- 红帽构建的 Keycloak 正在运行。

流程

1. 使用红帽构建的 Keycloak 在 Keycloak 管理控制台中创建域，例如： **example_jboss_infra**。您将使用此域来创建所需的用户、角色和客户端。
   如需更多信息，请参阅创建域。
2. 创建用户。例如，**user1**。
   如需更多信息，请参阅 创建用户。
3. 为用户创建密码。例如，**passwordUser1**。
   如需更多信息，请参阅 为用户 设置密码。
4. 创建角色。例如，**管理员**。
   要在 JBoss EAP 中启用基于角色的访问控制(RBAC)，名称应为标准 RBAC 角色之一，如 **管理员**。有关 JBoss EAP 中的 RBAC 的更多信息，请参阅 JBoss EAP 7.4 *安全架构指南中的* 基于角色的访问控制。

   有关在红帽构建的 Keycloak 中创建角色的更多信息，请参阅 创建域角色。
5. 为用户分配角色。
   如需更多信息，请参阅 分配角色映射。
6. 创建 OpenID Connect 客户端，如 **jboss-console**。
   - 确保选中了以下能力配置值：
     - 标准流
     - 直接访问权限授予
   - 至少在 **登录设置页面中设置以下属性** ：
     - 将 **Valid Redirect URI** 设置为管理控制台 URI。例如： http://localhost:9990。
     - 将 **Web Origins** 设置为管理控制台 URI。例如： http://localhost:9990。
7. 创建另一个 OpenID Connect 客户端，如 **jboss-management**，作为仅 bearer 的客户端。
   - 在功能配置中，取消选择以下选项：
     - 标准流
     - 直接访问权限授予
   - 您不需要在 **Login settings** 页面中指定任何字段。

现在，您可以使用您定义的客户端来保护 JBoss EAP 管理控制台。如需更多信息，请参阅使用 OpenID Connect 保护 JBoss EAP 管理控制台。

其他资源

- 使用 OIDC 的 JBoss EAP 管理控制台安全性

4.3. 使用 OPENID CONNECT 保护 JBOSS EAP 管理控制台

当您使用 OpenID Connect (OIDC)保护 JBoss EAP 管理控制台时，JBoss EAP 会重定向到 OIDC 供应商，供用户登录管理控制台。

先决条件

- 您已在 OIDC 供应商中配置了所需的客户端。
  如需更多信息，请参阅配置红帽构建的 Keycloak 来保护 JBoss EAP 管理控制台 。

流程

1. 在 **elytron-oidc-client** 子系统中配置 OIDC 供应商。

   语法

   ```
   /subsystem=elytron-oidc-client/provider=keycloak:add(provider-url=<OIDC_provider_URL>)
   ```

   Example

   ```
   /subsystem=elytron-oidc-client/provider=keycloak:add(provider-
   url=http://localhost:8180/realms/example_jboss_infra)
   ```

2. 创建名为 **wildfly-management** 的 **secure-deployment** 资源，以保护管理界面。

   语法

   ```
   /subsystem=elytron-oidc-client/secure-deployment=wildfly-
   management:add(provider=<OIDC_provider_name>,client-
   id=<OIDC_client_name>,principal-attribute=<attribute_to_use_as_principal>,bearer-
   only=true,ssl-required=<internal_or_external>)
   ```

   Example

   ```
   /subsystem=elytron-oidc-client/secure-deployment=wildfly-
   management:add(provider=keycloak,client-id=jboss-management,principal-
   attribute=preferred_username,bearer-only=true,ssl-required=EXTERNAL)
   ```

3. OPTIONAL：您可以使用以下命令启用基于角色的访问控制(RBAC)。

   ```
   /core-service=management/access=authorization:write-attribute(name=provider,value=rbac)
   /core-service=management/access=authorization:write-attribute(name=use-identity-
   roles,value=true)
   ```

4. 创建名为 **wildfly-console** 的 **secure-server** 资源，该资源引用 **jboss-console** 客户端。

   语法

```
/subsystem=elytron-oidc-client/secure-server=wildfly-
console:add(provider=<OIDC_provider_name>,client-id=<OIDC_client_name>,public-
client=true)
```

Example

```
/subsystem=elytron-oidc-client/secure-server=wildfly-console:add(provider=keycloak,client-
id=jboss-console,public-client=true)
```

重要

JBoss EAP 管理控制台要求 **secure-server** 资源专门命名为 **wildfly-console**。

验证

1. 访问管理控制台。默认情况下，管理控制台位于 http://localhost:9990。
   您将被重定向到 OIDC 供应商。
2. 使用您在 OIDC 供应商中创建的用户凭证登录。

JBoss EAP 管理控制台现在通过 OIDC 进行保护。

其他资源

- 使用 OIDC 的 JBoss EAP 管理控制台安全性
- **Elytron-oidc-client** 子系统属性

第 5 章 参考

5.1. ELYTRON-OIDC-CLIENT 子系统属性

elytron-oidc-client 子系统提供属性来配置其行为。

表 5.1. Elytron-oidc-client 子系统属性

属性	描述
provider	配置 OpenID Connect 供应商。
secure-deployment	由 OpenID Connect 供应商保护的部署。
realm	配置红帽构建的 Keycloak 域。这为方便用户提供。您可以在 keycloak 客户端适配器中复制配置，并在此处使用。建议使用 **供应商**。

为以下目的使用三个 **elytron-oidc-client** 属性：

- **Provider**: 用于配置 OpenID Connect 供应商。如需更多信息，请参阅 **供应商 属性**。
- **secure-deployment** ：用于配置由 OpenID Connect 保护的部署。如需更多信息，请参阅 **secure-deployment 属性**
- **域** ：用于配置红帽构建的 Keycloak。如需更多信息，请参阅 **域 属性**。不建议使用 **realm**。它是为了方便使用。您可以在 keycloak 客户端适配器中复制配置，并在此处使用。建议使用 **provider** 属性。

表 5.2. 供应商 属性

属性	默认值	描述
allow-any-hostname	**false**	如果将值设为 **true**，则在与 OpenID 供应商通信时跳过主机名验证。这在测试时很有用。不要在生产环境中将其设置为 **true**。
always-refresh-token		如果设置为 **true**，则子系统会在每次应用收到 Web 请求时刷新令牌，并将新请求发送到 OpenID 供应商以获取新的访问令牌。
auth-server-url		红帽构建的 Keycloak 域授权服务器的基本 URL。如果使用此属性，还必须定义 **realm** 属性。 您还可以使用 **provider-url** 属性在单个属性中提供基本 URL 和 realm。

属性	默认值	描述
autodetect-bearer-only	**false**	设置是否自动检测 bearer-only 请求。 当收到 bearer-only 请求时，**autodetect-bearer-only** 被设置为 **true**，应用程序无法参与浏览器登录。 使用此属性根据 **X-Requested-With**、PP**Action 或 Accept 等标头自动检测简单对象访问协议(SOAP**)或 REST 客户端。
client-id		在 OpenID 提供程序中注册的 JBoss EAP 的客户端 ID。
client-key-password		如果指定了 **client-keystore**，请在此属性中指定密码。
client-keystore		如果您的应用通过 HTTPS 与 OpenID 提供程序通信，请在此属性中设置客户端密钥存储的路径。
client-keystore-password		如果指定了 **客户端密钥存储**，请提供用于在此属性中访问它的密码。
confidential-port	**8443**	指定 OpenID 供应商使用的机密端口(SSL/TLS)。
connection-pool-size		指定与 OpenID 供应商通信时使用的连接池大小。
connection-timeout-millis	**-1L**	指定与远程主机建立连接的超时时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**.**-1L** 表示该值未定义，这是默认值。
connection-ttl-millis	**-1L**	指定连接保留的时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**。**-1L** 表示值未定义，这是默认值。
cors-allowed-headers		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Allow-Headers** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-allowed-methods		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 Access-Control-Allow-Methods 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-exposed-headers		如果启用了 CORS，这将设置 Access-Control-Expose-Headers 标头的值。这应该是一个用逗号分开的字符串。这是 optinal。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-max-age		设置 Cross-Origin Resource Sharing (CORS) Max-Age 标头的值。该值可以在 **-1L** 和 **2147483647L** 之间。只有在 **enable-cors** 设为 **true** 时，此属性才会生效。
disable-trust-manager		指定在通过 HTTPS 与 OpenID 供应商通信时是否使用信任管理器。

属性	默认值	描述
enable-cors	**false**	启用红帽构建的 Keycloak Cross-Origin Resource Sharing (CORS) 支持。
expose-token	**false**	如果设置为 **true**，经过身份验证的浏览器客户端可以通过 Javascript HTTP 调用来获取签名的访问令牌，通过 URL **root/k_query_bearer_token**。这是可选的。这特定于红帽构建的 Keycloak。
ignore-oauth-query-parameter	**false**	禁用对 **access_token** 的查询参数解析。
principal-attribute		指定 ID 令牌中的声明值，用作身份的主体
provider-url		指定 OpenID 供应商 URL。
proxy-url		如果使用 HTTP 代理，请指定 HTTP 代理的 URL。
realm-public-key		指定域的公钥。
register-node-at-startup	**false**	如果设置为 **true**，则会将注册请求发送到红帽构建的 Keycloak。此属性仅在您的应用程序集群时才有用。
register-node-period		指定重新注册节点的频率。
socket-timeout-millis		以毫秒为单位指定等待数据的套接字超时。
ssl-required	**external**	指定与 OpenID 供应商的通信是否应该通过 HTTPS。该值可以是以下之一： • **All** - 所有通信都通过 HTTPS 进行。 • **external** - 只有与外部客户端的通信通过 HTTP 进行。 • **none** - 未使用 HTTP。
token-signature-algorithm	**RS256**	指定 OpenID 供应商使用的令牌签名算法。支持的算法有： • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**

属性	默认值	描述
token-store		为 auth-session 数据指定 Cookie 或会话存储。
truststore		指定用于客户端 HTTPS 请求的信任存储。
truststore-password		指定 truststore 密码。
verify-token-audience	**false**	如果设置为 **true**，则在仅 bearer 身份验证期间，如果令牌包含此客户端名称(**资源**)作为受众，则验证。

表 5.3. secure-deployment 属性

属性	默认值	描述
allow-any-hostname	**false**	如果将值设为 **true**，则在与 OpenID 供应商通信时跳过主机名验证。这在测试时很有用。不要将其设置为生产环境中的 **ture**。
always-refresh-token		如果设置为 **true**，JBoss EAP 会在每次 Web 请求上刷新令牌。
auth-server-url		红帽构建的 Keycloak 域授权服务器的基本 URL，也可以使用 **provider-url** 属性。
autodetect-bearer-only	**false**	设置是否自动检测 bearer-only 请求。当收到 bearer-only 请求时，**autodetect-bearer-only** 被设置为 **true**，应用程序无法参与浏览器登录。
bearer-only	**false**	把它设置为 **true**，以通过 Bearer Token 身份验证来保护应用。启用 Bearer Token 身份验证时，用户不会重定向到 OpenID 提供程序以进行登录；而 **elytron-oidc-client** 子系统会尝试验证用户的 bearer 令牌。
client-id		OpenID 提供程序中注册的客户端的唯一标识符。
client-key-password		如果指定了 **client-keystore**，请在此属性中指定其密码。

属性	默认值	描述
client-keystore		如果您的应用通过 HTTPS 与 OpenID 提供程序通信，请在此属性中设置客户端密钥存储的路径。
client-keystore-password		如果指定了 **客户端密钥存储**，请提供用于在此属性中访问它的密码。
confidential-port	**8443**	指定 OpenID 供应商使用的机密端口(SSL/TLS)。
connection-pool-size		指定与 OpenID 供应商通信时使用的连接池大小。
connection-timeout-millis	**-1L**	指定与远程主机建立连接的超时时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**。**-1L** 表示值未定义，这是默认值。
connection-ttl-millis	**-1L**	指定连接保留的时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**. -1L 表示该值未定义，这是默认值。
cors-allowed-headers		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Allow-Headers** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-allowed-methods		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Allow-Methods** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-exposed-headers		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Expose-Headers** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。

属性	默认值	描述
cors-max-age		设置 Cross-Origin Resource Sharing (CORS) Max-Age 标头的值。该值可以在 **-1L** 和 **2147483647L** 之间。只有在 **enable-cors** 设为 **true** 时，此属性才会生效。
credential		指定用于与 OpenID 供应商通信的凭证。
disable-trust-manager		指定在通过 HTTPS 与 OpenID 供应商通信时是否使用信任管理器。
enable-cors	**false**	启用红帽构建的 Keycloak Cross-Origin Resource Sharing (CORS) 支持。
enable-basic-auth	**false**	启用 Basic Authentication 以指定用于获取 bearer 令牌的凭证。
expose-token	**false**	如果设置为 **true**，经过身份验证的浏览器客户端可以通过 Javascript HTTP 调用来获取签名的访问令牌，通过 URL **root/k_query_bearer_token**。这是可选的。这特定于红帽构建的 Keycloak。
ignore-oauth-query-parameter	**false**	禁用对 **access_token** 的查询参数解析。
min-time-between-jwks-requests	如果子系统检测到由未知公钥签名的令牌，JBoss EAP 会尝试从 **elytron-oidc-client** 服务器下载新的公钥。属性指定 JBoss EAP 在后续下载尝试前等待的时间（以秒为单位）。该值可以在 **-1L** 和 **2147483647L** 之间。	principal-attribute
	指定 ID 令牌中的声明值，用作身份的主体	provider
	指定 OpenID 供应商。	provider-url
	指定 OpenID 供应商 URL。	proxy-url

属性	默认值	描述
	如果使用 HTTP 代理，请指定 HTTP 代理的 URL。	public-client
false	如果设置为 **true**，则在与 OpenID 提供程序通信时不会发送客户端凭证。这是可选的。	realm
	在 Red Hat build of Keycloak 中连接的域。	realm-public-key
	以 PEM 格式指定 OpenID 供应商的公钥。	redirect-rewrite-rule
	指定要应用到重定向 URI 的重写规则。	register-node-at-startup
false	如果设置为 **true**，则会将注册请求发送到红帽构建的 Keycloak。此属性仅在您的应用程序集群时才有用。	register-node-period
	指定重新注册节点的频率（以秒为单位）。	resource
	指定您要使用 OIDC 保护的应用程序名称。或者，您可以指定 **client-id**。	socket-timeout-millis
	以毫秒为单位指定等待数据的套接字超时。	ssl-required
external	指定与 OpenID 供应商的通信是否应该通过 HTTPS。该值可以是以下之一： • **All** - 所有通信都通过 HTTPS 进行。 • **external** - 只有与外部客户端的通信通过 HTTP 进行。 • **none** - 未使用 HTTP。	token-minimum-time-to-live
	如果当前令牌过期或是在您设定的时间（以秒为单位）内过期，则适配器会刷新令牌。	token-signature-algorithm

属性	默认值	描述
RS256	指定 OpenID 供应商使用的令牌签名算法。支持的算法有： • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**	token-store
	为 auth-session 数据指定 Cookie 或会话存储。	truststore
	指定用于适配器客户端 HTTPS 请求的信任存储。	truststore-password
	指定 truststore 密码。	turn-off-change-session-id-on-login
false	会话 ID 默认在成功登录时更改。将值设为 **true** 以将此关闭。	use-resource-role-mappings
false	使用从令牌获取的资源级别权限。	verify-token-audience

表 5.4. secure-server 属性

属性	默认值	描述
adapter-state-cookie-path		如果设置，这将定义子系统设置的 Cookie 中使用的路径。如果没有设置，则使用"" 作为路径。
allow-any-hostname	**false**	如果将值设为 **true**，则在与 OpenID 供应商通信时跳过主机名验证。这在测试时很有用。不要在生产环境中将其设置为 **true**。
always-refresh-token		如果设置为 **true**，则子系统会在每次应用收到 Web 请求时刷新令牌，并将新请求发送到 OpenID 供应商以获取新的访问令牌。

属性	默认值	描述
auth-server-url-for-backend-requests		指定仅用于直接调用 OpenID 供应商的后端请求的 URL，而无需通过负载均衡器或反向代理。
auth-server-url		红帽构建的 Keycloak 域授权服务器的基本 URL，也可以使用 **provider-url** 属性。
autodetect-bearer-only	**false**	设置是否自动检测 bearer-only 请求。 当收到 bearer-only 请求时，**autodetect-bearer-only** 被设置为 **true**，应用程序无法参与浏览器登录。 使用此属性根据 **X-Requested-With**、PP**Action 或 Accept 等标头自动检测简单对象访问协议(SOAP**)或 REST 客户端。
bearer-only	**false**	把它设置为 **true**，以通过 Bearer Token 身份验证来保护应用。 启用 Bearer Token 身份验证时，用户不会重定向到 OpenID 提供程序以进行登录；而 **elytron-oidc-client** 子系统会尝试验证用户的 bearer 令牌。
client-id		OpenID 提供程序中注册的客户端的唯一标识符。
client-key-password		如果指定了 **client-keystore**，请在此属性中指定其密码。
client-keystore-password		如果指定了 **客户端密钥存储**，请提供用于在此属性中访问它的密码。
client-keystore		通过 HTTPS 与 OpenID 提供程序通信时，在此属性中设置客户端密钥存储的路径。
confidential-port	**8443**	指定 OpenID 供应商使用的机密端口(SSL/TLS)。
connection-pool-size		指定与 OpenID 供应商通信时使用的连接池大小。

属性	默认值	描述
connection-timeout-millis	**-1L**	指定与远程主机建立连接的超时时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**。**-1L** 表示值未定义，这是默认值。
connection-ttl-millis	**-1L**	指定连接保留的时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**. -1L 表示该值未定义，这是默认值。
cors-allowed-headers		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Allow-Headers** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-allowed-methods		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Allow-Methods** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-exposed-headers		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Expose-Headers** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-max-age		设置 Cross-Origin Resource Sharing (CORS) Max-Age 标头的值。该值可以在 **-1L** 和 **2147483647L** 之间。只有在 **enable-cors** 设为 **true** 时，此属性才会生效。
credential		指定用于与 OpenID 供应商通信的凭证。

属性	默认值	描述
disable-trust-manager		指定在通过 HTTPS 与 OpenID 供应商通信时是否使用信任管理器。
enable-basic-auth	**false**	启用 Basic Authentication 以指定用于获取 bearer 令牌的凭证。
enable-cors	**false**	启用红帽构建的 Keycloak Cross-Origin Resource Sharing (CORS) 支持。
expose-token	**false**	如果设置为 **true**，经过身份验证的浏览器客户端可以通过 Javascript HTTP 调用来获取签名的访问令牌，通过 URL **root/k_query_bearer_token**。这是可选的。这特定于红帽构建的 Keycloak。
ignore-oauth-query-parameter	**false**	禁用对 **access_token** 的查询参数解析。
min-time-between-jwks-requests		如果子系统检测到由未知公钥签名的令牌，JBoss EAP 会尝试从 **elytron-oidc-client** 服务器下载新的公钥。但是，如果您已经尝试了小于这个值的值，则 JBoss EAP deosn 不会尝试下载新的公钥，以秒为单位。该值可以在 **-1L** 和 **2147483647L** 之间。
principal-attribute		指定 ID 令牌中的声明值，用作身份的主体
principal-attribute		指定 ID 令牌中的声明值，用作身份的主体。
provider		指定 OpenID 供应商。
provider-url		指定 OpenID 供应商 URL。
proxy-url		如果使用 HTTP 代理，请指定 HTTP 代理的 URL。
public-client	**false**	如果设置为 **true**，则在与 OpenID 提供程序通信时不会发送客户端凭证。这是可选的。

属性	默认值	描述
public-key-cache-ttl		两个请求之间检索新公钥的最大间隔（以秒为单位）。
realm-public-key		以 PEM 格式指定 OpenID 供应商的公钥。
realm		在 Red Hat build of Keycloak 中连接的域。
redirect-rewrite-rule		指定要应用到重定向 URI 的重写规则。
register-node-at-startup	**false**	如果设置为 **true**，则会将注册请求发送到红帽构建的 Keycloak。此属性仅在您的应用程序集群时才有用。
register-node-period		指定重新注册节点的频率（以秒为单位）。
resource		指定您要使用 OIDC 保护的应用程序名称。或者，您可以指定 **client-id**。
socket-timeout-millis		以毫秒为单位指定等待数据的套接字超时。
ssl-required	**external**	指定与 OpenID 供应商的通信是否应该通过 HTTPS。该值可以是以下之一： • **All** - 所有通信都通过 HTTPS 进行。 • **external** - 只有与外部客户端的通信通过 HTTP 进行。 • **none** - 未使用 HTTP。
token-minimum-time-to-live		如果当前令牌过期或是在您设定的时间（以秒为单位）内过期，则适配器会刷新令牌。

属性	默认值	描述
token-signature-algorithm	**RS256**	指定 OpenID 供应商使用的令牌签名算法。支持的算法有： • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**
token-store		为 auth-session 数据指定 Cookie 或会话存储。
truststore-password		指定 truststore 密码。
truststore		指定用于适配器客户端 HTTPS 请求的信任存储。
turn-off-change-session-id-on-login	**false**	会话 ID 默认在成功登录时更改。将值设为 **true** 以将此关闭。
use-resource-role-mappings	**false**	使用从令牌获取的资源级别权限。
verify-token-audience	**false**	如果设置为 **true**，则在仅 bearer 身份验证过程中，适配器会验证令牌是否包含这个客户端名称(**资源**)作为受众。

表 5.5. 域 属性

属性	默认值	描述
allow-any-hostname	**false**	如果将值设为 **true**，则在与 OpenID 供应商通信时跳过主机名验证。这在测试时很有用。不要将其设置为生产环境中的 **ture**。
always-refresh-token		如果设置为 **true**，则子系统会在每次应用收到 Web 请求时刷新令牌，并将新请求发送到 OpenID 供应商以获取新的访问令牌。

属性	默认值	描述
auth-server-url		红帽构建的 Keycloak 域授权服务器的基本 URL，也可以使用 **provider-url** 属性。
autodetect-bearer-only	**false**	设置是否自动检测 bearer-only 请求。当收到 bearer-only 请求时，**autodetect-bearer-only** 被设置为 **true**，应用程序无法参与浏览器登录。
client-key-password		如果指定了 **client-keystore**，请在此属性中指定密码。
client-keystore		如果您的应用通过 HTTPS 与 OpenID 提供程序通信，请在此属性中设置客户端密钥存储的路径。
client-keystore-password		如果指定了 **客户端密钥存储**，请提供用于在此属性中访问它的密码。
confidential-port	**8443**	指定红帽构建的 Keycloak 使用的机密端口(SSL/TLS)。
connection-pool-size		指定与红帽构建的 Keycloak 通信时使用的连接池大小。
connection-timeout-millis	**-1L**	指定与远程主机建立连接的超时时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**。**-1L** 表示值未定义，这是默认值。
connection-ttl-millis	**-1L**	指定连接保留的时间（以毫秒为单位）。最小值为 **-1L**，最大 **2147483647L**。**-1L** 表示值未定义，这是默认值。
cors-allowed-headers		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 Access-Control-Allow-Headers 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。

属性	默认值	描述
cors-allowed-methods		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Allow-Methods 标头的值**。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-exposed-headers		如果启用了 Cross-Origin Resource Sharing (CORS)，这将设置 **Access-Control-Expose-Headers** 标头的值。这应该是一个用逗号分开的字符串。这是可选的。如果没有设置，则不会在 CORS 响应中返回此标头。
cors-max-age		设置 Cross-Origin Resource Sharing (CORS) Max-Age 标头的值。该值可以在 **-1L** 和 **2147483647L** 之间。只有在 **enable-cors** 设为 **true** 时，此属性才会生效。
disable-trust-manager		指定在通过 HTTPS 与 OpenID 供应商通信时是否使用信任管理器。
enable-cors	**false**	启用红帽构建的 Keycloak Cross-Origin Resource Sharing (CORS) 支持。
expose-token	**false**	如果设置为 **true**，经过身份验证的浏览器客户端可以通过 Javascript HTTP 调用来获取签名的访问令牌，通过 URL **root/k_query_bearer_token**。这是可选的。
ignore-oauth-query-parameter	**false**	禁用对 **access_token** 的查询参数解析。
principal-attribute		指定 ID 令牌中的声明值，用作身份的主体
provider-url		指定 OpenID 供应商 URL。

属性	默认值	描述
proxy-url		如果使用 HTTP 代理，请指定 HTTP 代理的 URL。
realm-public-key		指定域的公钥。
register-node-at-startup	**false**	如果设置为 **true**，则会将注册请求发送到红帽构建的 Keycloak。此属性仅在您的应用程序集群时才有用。
register-node-period		指定重新注册节点的频率。
socket-timeout-millis		以毫秒为单位指定等待数据的套接字超时。
ssl-required	**external**	指定与 OpenID 供应商的通信是否应该通过 HTTPS。该值可以是以下之一： • **All** - 所有通信都通过 HTTPS 进行。 • **external** - 只有与外部客户端的通信通过 HTTP 进行。 • **none** - 未使用 HTTP。
token-signature-algorithm	**RS256**	指定 OpenID 供应商使用的令牌签名算法。支持的算法有： • **RS256** • **RS384** • **RS512** • **ES256** • **ES384** • **ES512**
token-store		为 auth-session 数据指定 Cookie 或会话存储。
truststore		指定用于客户端 HTTPS 请求的信任存储。
truststore-password		指定 truststore 密码。

属性	默认值	描述
verify-token-audience	**false**	如果设置为 **true**，则在仅 bearer 身份验证过程中，适配器会验证令牌是否包含这个客户端名称（资源）作为 audience。

其他资源

- 在 JBoss EAP 中使用 OpenID Connect 的应用安全性
- 使用 OpenID Connect 保护 Web 应用程序

5.2. SECURITY-DOMAIN 属性

您可以通过设置其属性来配置 **security-domain**。

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

属性	描述
role-mapper	对此域要使用的 RoleMapper 的引用。
security-event-listener	对安全事件的监听程序的引用。
trusted-security-domains	此安全域信任的安全域的列表。
trusted-virtual-security-domains	此安全域信任的虚拟安全域的列表。

5.3. VIRTUAL-SECURITY-DOMAIN ATTRIBUTES

您可以通过设置其属性来配置 **virtual-security-domain**。

表 5.6. virtual-security-domain attributes

属性	描述
outflow-anonymous	如果无法将安全身份流传输到安全域，则把此属性设置为 **true** 以出站匿名身份，这会在以下情况下发生： ● 要排除此虚拟域的域不信任这个虚拟域。 ● 那个域中不存在将流向域的身份 流出匿名身份对清除已为该域创建的任何身份有作用。 默认值为 **false**。
outflow-security-domains	此虚拟域中的安全身份应自动流向的安全域列表。