Red Hat Single Sign-On 7.6

保护应用程序和服务指南

适用于 Red Hat Single Sign-On 7.6

Last Updated: 2024-06-05

Red Hat Single Sign-On 7.6 保护应用程序和服务指南

适用于 Red Hat Single Sign-On 7.6

法律通告

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

摘要

本指南包含使用红帽单点登录 7.6 保护应用程序和服务的信息

目录

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 计划保护应用程序和服务

Red Hat Single Sign-On 支持 OpenID Connect（ OAuth 2.0 的扩展）和 SAML 2.0。在保护客户端和服务时，您需要决定您使用的两个服务是什么。如果您想还可以选择使用 OpenID Connect 和其它通过 SAML 保护部分 OpenID Connect。

为了保护客户端和服务，您还需要适配器或库用于您选择的协议。Red Hat Single Sign-On 附带了自己的适配器（用于所选平台），但也可以使用通用 OpenID Connect 重新以及 SAML Service Provider 库。

1.1. 客户端适配器

Red Hat Single Sign-On 客户端适配器是库，可让您轻松使用 Red Hat Single Sign-On 保护应用程序和服务。当它们提供了一个与底层平台和框架的紧密集成时，我们称其适配器而不是库。这样，我们的适配器易于使用，它们需要较少的样板代码，而不是库通常所需的代码。

1.2. 支持的平台

Red Hat Single Sign-On 可让您保护在不同平台上运行的应用程序，并使用 OpenID Connect 和 SAML 协议使用不同的技术堆栈。

1.2.1. OpenID Connect

1.2.1.1. Java

- JBoss EAP
- servlet Filter
- Spring Boot

1.2.1.2. JavaScript（客户端）

- JavaScript

1.2.1.3. Node.js (server-side)

- Node.js

1.2.2. SAML

1.2.2.1. Java

- JBoss EAP
- servlet 过滤器

1.2.2.2. Apache HTTP 服务器

- mod_auth_mellon

1.3. 支持的协议

Red Hat Single Sign-On 支持 OpenID Connect 和 SAML 协议。

1.3.1. OpenID Connect

OpenID Connect (OIDC)是一个身份验证协议，它是 OAuth 2.0 的扩展。虽然 OAuth 2.0 只是用于构建授权协议的框架，但主要不完整，但 OIDC 是一个功能齐全的身份验证和授权协议。OIDC 还使用 Json Web Token (JWT)标准集合。这些标准定义了身份令牌 JSON 格式，以及以紧凑和 Web 友好的方式对这些数据进行数字签名。

使用 OIDC 时，实际上有两种用例。第一个应用程序要求红帽单点登录服务器为其验证用户。成功登录后，应用将收到一个*身份令牌*和一个*访问令牌*。*身份令牌* 包含有关用户的信息，如用户名、电子邮件和其他配置集信息。*访问令牌* 由域数字签名，包含应用可使用的访问信息（如 user 角色映射），供应用程序用来决定用户被允许访问哪些资源。

第二种用例类型是希望获得远程服务访问权限的客户端。在这种情况下，客户端会要求红帽单点登录获取一个 *访问令牌*，供该用户用于对其他远程服务调用。Red Hat Single Sign-On 验证用户，然后要求用户同意授予请求它的客户端的访问权限。然后，客户端接收 *访问令牌*。此 *访问令牌* 由域数字签名。客户端可以使用此 *访问令牌* 在远程服务上发出 REST 调用。REST 服务提取 *访问令牌*，验证令牌的签名，然后根据令牌中的访问信息决定是否处理请求。

1.3.2. SAML 2.0

SAML 2.0 是与 OIDC 类似的规范，但有一个较老且更成熟的。它在 SOAP 中有根根和 WS-* 规格的 plethora，因此它往往会比 OIDC 更详细。SAML 2.0 主要是一种身份验证协议，可以在身份验证服务器和应用之间交换 XML 文档。XML 签名和加密用于验证请求和响应。

在红帽单点登录 SAML 中，提供两种用例：浏览器应用和 REST 调用。

使用 SAML 时，实际上有两种用例。第一个应用程序要求红帽单点登录服务器为其验证用户。成功登录后，应用将收到一个 XML 文档，其中包含一个名为 SAML 断言的内容，用于指定用户的各种属性。该 XML 文档由域数进行数字签名，包含应用可用于决定用户允许对应用程序访问哪些资源（如 user 角色映射）。

第二种用例类型是希望获得远程服务访问权限的客户端。在这种情况下，客户端要求红帽单点登录获得 SAML 断言，以便其用于代表用户调用其他远程服务。

1.3.3. OpenID Connect 与.SAML

在 OpenID Connect 和 SAML 间进行选择不只是使用较新的协议(OIDC)而不是旧的更成熟的协议(SAML)。

在大多数情况下，Red Hat Single Sign-On 建议使用 OIDC。

SAML 往往会比 OIDC 更详细。

除交换数据外，如果您比较的是，您会看到 OIDC 的设计旨在与 web 合作，而 SAML 被重新引入到 web 之上工作。例如，OIDC 更加适合 HTML5/JavaScript 应用程序，因为它比 SAML 更容易在客户端实现。由于令牌采用 JSON 格式，因此它们更容易被 JavaScript 使用。您还会发现一些在 Web 应用程序中实施安全性的附加功能。例如，请查看 iframe 技巧，说明规格用来轻松确定用户是否仍登录或未登录。

SAML 对其使用。当您看到 OIDC 规格的演变时，您会看到它们实现 SAML 多年的功能及更多功能。我们通常看到的是，用户选择 SAML 通过 OIDC，因为它比较成熟，同时还具有保护的现有应用程序。

1.4. 术语

本指南中使用这些术语：

- 客户端是 与红帽单点登录交互的实体，用于验证用户并获取令牌。大多数时候，客户端都是代表用户提供单点登录体验的用户，并使用服务器发布的令牌访问其他服务。客户端也可以只对获取令牌并自行代表访问其他服务的实体。
- 应用程序 包括适用于不同协议的特定平台的各种应用程序
- 客户端适配器 是库，可让您轻松使用红帽单点登录保护应用程序和服务。它们提供了一个与底层平台和框架的紧密集成。
- 创建客户端和注册客户端是相同的操作。创建客户端 是使用 Admin 控制台创建客户端的术语。 注册客户端是使用 **Red Hat Single Sign-On** 客户端注册 服务来注册客户端的术语。
- 服务帐户是 能够自行获取令牌的客户端类型。

1.5. 保护应用程序和服务的基本步骤

以下是在红帽单点登录中保护应用程序或服务的基本步骤。

1. 使用以下选项之一配置客户端：
   - Red Hat Single Sign-On 适配器
   - 通用 OpenID 连接或 SAML 库
2. 使用以下选项之一注册客户端：
   - Red Hat Single Sign-On Admin Console
   - 客户端注册服务
   - CLI

其他资源

- 本指南提供这些步骤的详细说明。相关信息请参考《 服务器管理指南》 中。本指南提供使用管理控制台创建客户端的说明。创建客户端与使用 Red Hat Single Sign-On 客户端注册服务注册客户端的任务相同。

第 2 章 使用 OPENID CONNECT 来保护应用程序和服务

本节介绍了如何使用 Red Hat Single Sign-On 适配器或通用 OpenID Connect 来保护使用 OpenID Connect 的应用程序和服务。

2.1. JAVA 适配器

Red Hat Single Sign-On 带有 Java 应用程序的不同适配器范围。选择正确的适配器取决于目标平台。

所有 Java 适配器共享一组常见配置选项，如 Java 适配器配置 章节中所述。

2.1.1. Java adapter 配置

Red Hat Single Sign-On 支持的每个 Java 适配器都可通过简单的 JSON 文件进行配置。这种情况可能如下所示：

```
{
  "realm" : "demo",
  "resource" : "customer-portal",
  "realm-public-key" : "MIGfMA0GCSqGSIb3D...31LwIDAQAB",
  "auth-server-url" : "https://localhost:8443/auth",
  "ssl-required" : "external",
  "use-resource-role-mappings" : false,
  "enable-cors" : true,
  "cors-max-age" : 1000,
  "cors-allowed-methods" : "POST, PUT, DELETE, GET",
  "cors-exposed-headers" : "WWW-Authenticate, My-custom-exposed-Header",
  "bearer-only" : false,
  "enable-basic-auth" : false,
  "expose-token" : true,
  "verify-token-audience" : true,
  "credentials" : {
    "secret" : "234234-234234-234234"
  },

  "connection-pool-size" : 20,
  "socket-timeout-millis" : 5000,
  "connection-timeout-millis" : 6000,
  "connection-ttl-millis" : 500,
  "disable-trust-manager" : false,
  "allow-any-hostname" : false,
  "truststore" : "path/to/truststore.jks",
  "truststore-password" : "geheim",
  "client-keystore" : "path/to/client-keystore.jks",
  "client-keystore-password" : "geheim",
  "client-key-password" : "geheim",
  "token-minimum-time-to-live" : 10,
  "min-time-between-jwks-requests" : 10,
  "public-key-cache-ttl" : 86400,
  "redirect-rewrite-rules" : {
    "^/wsmaster/api/(.*)$" : "/api/$1"
  }
}
```

您可以将 **${...}** containure 用于系统属性替换。例如： **${jboss.server.config.dir}** 替换为 **/path/to/Red Hat Single Sign-On**。也支持通过 **env** 前缀替换环境变量，如 **${env.MY_ENVIRONMENT_VARIABLE}**。

初始配置文件可以从 admin 控制台获得。这可以通过打开 admin 控制台来完成，从菜单中选择 **Clients** 并单击对应的客户端。打开客户端页面后，点 **Installation** 选项卡并选择 **Keycloak OIDC JSON**。

以下是每个配置选项的描述：

realm

域的名称。这是 *REQUIRED*.

resource

应用程序的 client-id。每个应用程序都有一个用于标识应用程序的 client-id。这是 *REQUIRED*.

realm-public-key

realm 公钥的 PEM 格式。您可以从 Admin Console 获取它。*这是选项*，我们不推荐对其进行设置。如果没有设置，则适配器会从红帽单点登录下载，并在需要时总是重新下载（例如，Red Hat Single Sign-On 轮转其密钥）。但是，如果设置了 realm-public-key，则适配器永远不会从 Red Hat Single Sign-On 下载新密钥，因此当红帽单点登录轮转它的密钥时，适配器可能会中断。

auth-server-url

红帽单点登录服务器的基本 URL。所有其他红帽单点登录页面和 REST 服务端点都源自于此。它通常是 **https://host:port/auth** 格式。这是 *REQUIRED*.

ssl-required

确保所有与 Red Hat Single Sign-On 服务器的通信均通过 HTTPS。在生产环境中，这应设置为 **all**。这是 *选项*。默认值为 *external*。这意味着外部请求默认需要 HTTPS。有效值为 'all'、'external' 和 'none'。

confidential-port

红帽单点登录服务器用于通过 SSL/TLS 进行安全连接的机密端口。这是 *选项*。默认值为 *8443*。

use-resource-role-mappings

如果设置为 true，则适配器会在令牌内部查找用户的应用程序级别的角色映射。如果为 false，它将查看用户角色映射的域级别。这是 *选项*。默认值为 *false*。

public-client

如果设置为 true，则适配器不会向 Red Hat Single Sign-On 发送客户端凭证。这是 *选项*。默认值为 *false*。

enable-cors

这可实现 CORS 支持。它将处理 CORS preflight 请求。它还将查看访问令牌来确定有效的来源。这是 *选项*。默认值为 *false*。

cors-max-age

如果启用了 CORS，这会设置 **Access-Control-Max-Age** 标头的值。这是 *选项*。如果没有设置，则这个标头不会在 CORS 响应中返回。

cors-allowed-methods

如果启用了 CORS，这会设置 **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 *选项*。如果没有设置，则这个标头不会在 CORS 响应中返回。

cors-allowed-headers

如果启用了 CORS，这会设置 **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 *选项*。如果没有设置，则这个标头不会在 CORS 响应中返回。

cors-exposed-headers

如果启用了 CORS，这会设置 **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 *选项*。如果没有设置，则这个标头不会在 CORS 响应中返回。

bearer-only

对于服务，这应设为 *true*。如果启用适配器不会试图验证用户，但只验证 bearer 令牌。这是 *选项*。默认值为 *false*。

仅自动探测器

如果您的应用程序同时充当 Web 应用程序和 Web 服务（如 SOAP 或 REST），则此项应设为 *true*。它允许您将 Web 应用的未经身份验证的用户重定向到 Red Hat Single Sign-On 登录页面，但向未经身份验证的 SOAP 或 REST 客户端发送 HTTP **401** 状态代码，因为它们不知道到登录页面的重定向。红帽单点登录根据典型的标头（如 **X-Requested-With**、**SOAPAction** 或 **Accept** ）自动探测 SOAP 或 REST 客户端。默认值为 *false*。

enable-basic-auth

这告知适配器也支持基本身份验证。如果启用了这个选项，则必须提供 *secret*。这是 *选项*。默认值为 *false*。

expose-token

如果为 **true**，则经过身份验证的浏览器客户端（通过 JavaScript HTTP 调用）可以通过 URL **root/k_query_bearer_token** 获取签名的访问令牌。这是 *选项*。默认值为 *false*。

credentials

指定应用程序的凭证。这是一个对象表示法，其中键是凭证类型，值是凭证类型的值。目前支持 password 和 jwt。这只适用于 *具有* "Confidential"访问类型的客户端。

connection-pool-size

此配置选项定义应池与红帽单点登录服务器的连接数量。这是 *选项*。默认值为 **20**。

socket-timeout-millis

在建立连接（以毫秒为单位）后，套接字等待数据的超时。在两个数据包之间不活跃的最大时间。超时值为零将解释为无限超时。负值代表为未定义（如果适用，系统默认值）。默认值为 **-1**。这是 *选项*。

connection-timeout-millis

与远程主机建立连接的超时（以毫秒为单位）。超时值为零将解释为无限超时。负值代表为未定义（如果适用，系统默认值）。默认值为 **-1**。这是 *选项*。

connection-ttl-millis

以毫秒为单位连接客户端到实时的连接时间。小于或等于零的值被解释为无限值。默认值为 **-1**。这是 *选项*。

disable-trust-manager

如果 Red Hat Single Sign-On 服务器需要 HTTPS，且这个配置选项被设置为 **true**，则不需要指定信任存储。此设置应只在开发期间使用，永远不会 在生产环境中使用，因为它将禁用 SSL 证书的验证。这是 *选项*。默认值为 **false**。

allow-any-hostname

如果 Red Hat Single Sign-On 服务器需要 HTTPS，且该配置选项被设置为 **true**，则红帽单点登录服务器的证书通过信任存储进行验证，但不会执行主机名验证。此设置应只在开发期间使用，永远不会 在生产环境中使用，因为它将禁用 SSL 证书的验证。此设置在测试环境中可能很有用，这是 *选项*。默认值为 **false**。

proxy-url

HTTP 代理的 URL（如果使用）。

truststore

值是信任存储文件的文件路径。如果您使用 **classpath:** 前缀，则信任存储将从部署的 classpath 获取。用于向 Red Hat Single Sign-On 服务器的传出 HTTPS 通信。进行 HTTPS 请求的客户端需要一种验证它们要通信的服务器主机的方法。这就是信任者的作用。该密钥存储包含一个或多个可信主机证

书或证书颁发机构。您可以通过提取红帽单点登录服务器的 SSL 密钥存储的公共证书来创建此信任存储。这是 *REQUIRED*，除非 **ssl-required** 为 **none** 或 **disable-trust-manager** 为 **true**。

truststore-password

truststore 的密码。如果设置了 **truststore** 且信任存储需要密码，则这是 *REQUIRED*。

client-keystore

这是密钥存储文件的文件路径。当适配器向 Red Hat Single Sign-On 服务器发出 HTTPS 请求时，此密钥存储包含双向 SSL 的客户端证书。这是 *选项*。

client-keystore-password

客户端密钥存储的密码。如果设置了 **client-keystore**，则这是 *REQUIRED*。

client-key-password

客户端密钥的密码。如果设置了 **client-keystore**，则这是 *REQUIRED*。

always-refresh-token

如果为 *true*，则适配器将在每个请求中刷新令牌。警告 - 启用后，将导致每个应用程序请求红帽单点登录请求。

register-node-at-startup

如果为 *true*，则适配器会向红帽单点登录发送注册请求。默认情况下为 *false*，仅在应用程序集群时使用。详情请参阅 应用程序集群

register-node-period

红帽单点登录重新注册适配器的时间。应用程序集群时会很有用。详情请参阅 应用程序集群

token-store

可能的值有 *session* 和 *cookie*。默认为 *session*，这意味着适配器在 HTTP Session 中存储帐户信息。替代 *Cookie* 意味着在 cookie 中存储信息。详情请参阅 应用程序集群

token-cookie-path

在使用 cookie 存储时，此选项会设置用于存储帐户信息的 cookie 的路径。如果是一个相对路径，则假定应用程序在上下文根目录中运行，并相对于该上下文 root 进行解释。如果是绝对路径，则使用绝对路径来设置 Cookie 路径。默认使用相对于上下文 root 的路径。

principal-attribute

OpenID Connect ID Token 属性，用于填充 UserPrincipal 名称：如果 token 属性为空，则默认为 **sub**。可能的值有 **sub**,**preferred_username**,**email**,**name**,**nickname**,**given_name**,**family_name**。

turn-off-change-session-id-on-login

在某些平台上成功登录时，会话 ID 被改变，以插入安全攻击向量。如果要关闭此操作，请将此项更改为 *true*。默认值为 *false*。

token-minimum-time-to-live

在过期之前，使用 Red Hat Single Sign-On 服务器预先刷新一个活跃的访问令牌的时间（以秒为单位）。当访问令牌发送到另一个 REST 客户端（在评估之前其过期时）时特别有用。这个值不应超过 realm 的访问令牌生命周期span。这是 *选项*。默认值为 **0** 秒，因此当访问令牌已过期时，适配器才会刷新访问令牌。

min-time-between-jwks-requests

以秒为单位指定两个请求之间的最短间隔，以便检索新的公钥。默认是 10 秒。当适配器识别带有未知 **kid** 的令牌时，适配器会始终尝试下载新的公钥。但是，每 10 秒（默认）不会尝试一次。当攻击者使用错误的 **kid** 强制适配器向 Red Hat Single Sign-On 发送大量请求时，为了避免 DoS。

public-key-cache-ttl

以秒为单位，指定两个请求到红帽单点登录之间的最大间隔，以检索新的公钥。默认为 86400 秒（1 天）。当适配器识别带有未知 **kid** 的令牌时，适配器会始终尝试下载新的公钥。如果它通过已知 **kid** 识别令牌，它仅使用之前下载的公钥。但是，每个配置的时间间隔至少一次（默认为 1 天）是新的公钥，即使 **kid** of token 已经已知。

ignore-oauth-query-parameter

默认为 **false**，如果设为 **true**，则会关闭处理 bearer 令牌处理的 **access_token** 查询参数。如果用户只通过了 **access_token**，用户将无法验证

redirect-rewrite-rules

如果需要，指定 Redirect URI 重写规则。这是一个对象表示法，其中键是要匹配重定向 URI 的正则表达式，值是替换 String。**$** 字符可用于替换 String 中的反向引用。

verify-token-audience

如果设置为 **true**，则在使用 bearer 令牌进行身份验证的过程中，适配器将验证令牌是否包含这个客户端名称（资源）作为使用者。选项对于服务来说，主要服务于由 bearer 令牌验证的请求。这会默认设置为 **false**，但为了提高安全性，建议启用它。如需了解更多有关受众支持的详细信息，请参阅受众支持。

2.1.2. JBoss EAP 适配器

您可以从 ZIP 文件或从 RPM 安装这个适配器。

- 从 ZIP 文件安装 JBOSS EAP 适配器
- 从 RPM 安装 JBoss EAP 7 Adapter
- 从 RPM 安装 JBoss EAP 6 Adapter

2.1.3. 从一个 ZIP 文件安装 JBOSS EAP 适配器

为了保护 JBoss EAP 上部署的 WAR 应用安全，您必须安装并配置 Red Hat Single Sign-On 适配器子系统。然后，有两个选项来保护 WAR。

- 您可以在 WAR 中提供适配器配置文件，并在 web.xml 中将 auth-method 更改为 KEYCLOAK。
- 或者，您不必修改所有 WAR，您可以通过配置文件中的 Red Hat Single Sign-On adapter 子系统配置（如 **standalone.xml** ）来保护它。

本节都介绍这两种方法。

适配器作为单独的归档提供，具体取决于您使用的服务器版本。

流程

1. 从 软件下载 站点安装适用于您的应用程序服务器的适配器。
   - 在 JBoss EAP 7 上安装：

     ```
     $ cd $EAP_HOME
     $ unzip rh-sso-7.6.9-eap7-adapter.zip
     ```

     此 ZIP 存档包含特定于 Red Hat Single Sign-On 适配器的 JBoss 模块。它还包含用于配置适配器子系统的 JBoss CLI 脚本。
2. 要配置适配器子系统，请执行适当的命令。
   - 如果服务器 没有运行，则在 JBoss EAP 7.1 或更新版本上安装。

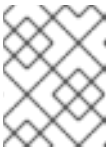
     ```
     $ ./bin/jboss-cli.sh --file=bin/adapter-elytron-install-offline.cli
     ```

注意

JBoss EAP 6.4 不可用的离线脚本

- 如果服务器在运行，则在 JBoss EAP 7.1 或更新版本上安装。

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



注意

在 JBoss EAP 7.1 或更新版本中可以使用旧的非 Elytron 适配器，这意味着您可以使用 **adapter-install-offline.cli**

注意

EAP 分别支持 OpenJDK 17 和 Oracle JDK 17 ， 自 7.4.CP7 和 7.4.CP8。请注意，新的 java 版本使 elytron 变体 compulsory，因此不要使用带有 JDK 17 的传统适配器。另外，在运行适配器 CLI 文件后，执行 EAP 提供的 **enable-elytron-se17.cli** 脚本。这两个脚本都需要配置 elytron 适配器并删除不兼容的 EAP 子系统。如需了解更多详细信息，请参阅此安全配置更改文章。

- 在 JBoss EAP 6.4 上安装

```
$ ./bin/jboss-cli.sh -c --file=bin/adapter-install.cli
```

2.1.3.1. JBoss SSO

JBoss EAP 内置了对部署到同一 JBoss EAP 实例的 Web 应用程序的单点登录支持。使用 Red Hat Single Sign-On 时不应该启用此功能。

2.1.3.2. 保护 WAR

这部分论述了如何通过在 WAR 软件包中添加配置和编辑文件来保护 WAR。

流程

1. 在 WAR 的 **WEB-INF** 目录下创建一个 **keycloak.json** 适配器配置文件。
这个配置文件的格式在 Java 适配器配置 部分中进行了描述。
2. 在 **web.xml** 中将 **auth-method** 设置为 **KEYCLOAK**。
3. 使用标准 servlet 安全性为您的 URL 指定角色基础限制。
例如：

```
<web-app 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_3_0.xsd"
    version="3.0">

  <module-name>application</module-name>

  <security-constraint>
```

```
        <web-resource-collection>
            <web-resource-name>Admins</web-resource-name>
            <url-pattern>/admin/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>admin</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Customers</web-resource-name>
            <url-pattern>/customers/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>KEYCLOAK</auth-method>
        <realm-name>this is ignored currently</realm-name>
    </login-config>

    <security-role>
        <role-name>admin</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

2.1.3.3. 通过适配器子系统保护 WAR

您不必修改 WAR 以便使用 Red Hat Single Sign-On 对其进行保护。您也可以通过 Red Hat Single Sign-On Adapter 子系统进行外部保护。虽然您不必将 KEYCLOAK 指定为 **auth-method**，但您仍然必须在 **web.xml** 中定义 **security-constraints**。但是，您不必创建 **WEB-INF/keycloak.json** 文件。元数据改为在 Red Hat Single Sign-On 子系统定义中的服务器配置(**standalone.xml**)中定义。

```
<extensions>
  <extension module="org.keycloak.keycloak-adapter-subsystem"/>
</extensions>

<profile>
  <subsystem xmlns="urn:jboss:domain:keycloak:1.1">
    <secure-deployment name="WAR MODULE NAME.war">
      <realm>demo</realm>
      <auth-server-url>http://localhost:8081/auth</auth-server-url>
      <ssl-required>external</ssl-required>
      <resource>customer-portal</resource>
```

```
        <credential name="secret">password</credential>
      </secure-deployment>
    </subsystem>
  </profile>
```

secure-deployment name 属性标识您要保护的 WAR。其值是 **web.xml** 中定义的 **module-name**，并附加 **.war**。其余的配置与在 Java 适配器配置 中定义的 **keycloak.json** 配置选项对应一个。

例外是 凭证 元素。

为了便于您了解，您可以进入 Red Hat Single Sign-On 管理控制台，并转至与这个 WAR 一致的应用程序的 Client/Installation 选项卡。它提供了一个可以剪切和粘贴的 XML 文件示例。

如果您有多个由同一域保护的部署，您可以在单独的元素中共享域配置。例如：

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
    <realm name="demo">
        <auth-server-url>http://localhost:8080/auth</auth-server-url>
        <ssl-required>external</ssl-required>
    </realm>
    <secure-deployment name="customer-portal.war">
        <realm>demo</realm>
        <resource>customer-portal</resource>
        <credential name="secret">password</credential>
    </secure-deployment>
    <secure-deployment name="product-portal.war">
        <realm>demo</realm>
        <resource>product-portal</resource>
        <credential name="secret">password</credential>
    </secure-deployment>
    <secure-deployment name="database.war">
        <realm>demo</realm>
        <resource>database-service</resource>
        <bearer-only>true</bearer-only>
    </secure-deployment>
</subsystem>
```

2.1.3.4. 安全域

安全上下文自动传播到 EJB 层。

2.1.4. 从 RPM 安装 JBoss EAP 7 适配器

注意

使用 Red Hat Enterprise Linux 7 时，术语频道被替换为术语仓库。这些说明中，仅使用术语库。

先决条件

在可以从 RPM 安装 JBoss EAP 7 适配器之前，您必须订阅 JBoss EAP 7.4 存储库。

- 确定使用 Red Hat Subscription Manager 将 Red Hat Enterprise Linux 系统注册到您的帐户。如需更多信息，请参阅 Red Hat Subscription Management 文档 。

- 如果您已经订阅了另一个 JBoss EAP 存储库，必须先退出该存储库。
  对于 Red Hat Enterprise Linux 6，7：使用 Red Hat Subscription Manager，通过以下命令订阅 JBoss EAP 7.4 存储库。根据您的 Red Hat Enterprise Linux 版本，将 <RHEL_VERSION> 替换为 6 或 7。

```
$ sudo subscription-manager repos --enable=jb-eap-7-for-rhel-<RHEL_VERSION>-server-rpms
```

  对于 Red Hat Enterprise Linux 8：使用 Red Hat Subscription Manager，使用以下命令订阅 JBoss EAP 7.4 存储库：

```
$ sudo subscription-manager repos --enable=jb-eap-7.4-for-rhel-8-x86_64-rpms --enable=rhel-8-for-x86_64-baseos-rpms --enable=rhel-8-for-x86_64-appstream-rpms
```

流程

1. 根据您的 Red Hat Enterprise Linux 版本，为 OIDC 安装 JBoss EAP 7 适配器。
   - 在 Red Hat Enterprise Linux 6 上安装 7:

```
$ sudo yum install eap7-keycloak-adapter-sso7_6
```

   - 在 Red Hat Enterprise Linux 8 中安装：

```
$ sudo dnf install eap7-keycloak-adapter-sso7_6
```

注意

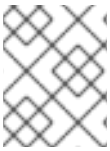
RPM 安装的默认 EAP_HOME 路径为 /opt/rh/eap7/root/usr/share/wildfly。

2. 为 OIDC 模块运行安装脚本。

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

您的安装已完成。

2.1.5. 从 RPM 安装 JBoss EAP 6 适配器



注意

使用 Red Hat Enterprise Linux 7 时，术语频道被替换为术语仓库。这些说明中，仅使用术语库。

您必须订阅 JBoss EAP 6 存储库，然后才能从 RPM 安装 EAP 6 适配器。

先决条件

- 确定使用 Red Hat Subscription Manager 将 Red Hat Enterprise Linux 系统注册到您的帐户。如需更多信息，请参阅 Red Hat Subscription Management 文档 。
- 如果您已经订阅了另一个 JBoss EAP 存储库，必须先退出该存储库。

使用 Red Hat Subscription Manager，使用以下命令订阅 JBoss EAP 6 存储库。根据您的 Red Hat Enterprise Linux 版本，将 <RHEL_VERSION> 替换为 6 或 7。

```
$ sudo subscription-manager repos --enable=jb-eap-6-for-rhel-<RHEL_VERSION>-server-rpms
```

流程

1. 使用以下命令为 OIDC 安装 EAP 6 适配器：

```
$ sudo yum install keycloak-adapter-sso7_6-eap6
```

注意

RPM 安装的默认 EAP_HOME 路径为 /opt/rh/eap6/root/usr/share/wildfly。

2. 为 OIDC 模块运行安装脚本。

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

您的安装已完成。

2.1.6. JBoss Fuse 6 adapter

Red Hat Single Sign-On 支持保护在 JBoss Fuse 6 中运行的 Web 应用程序。

警告

唯一支持的 Fuse 6 版本是最新的版本。如果您使用早期版本的 Fuse 6，则某些功能可能无法正常工作。特别是 Hawtio 集成将不适用于早期版本的 Fuse 6。

Fuse 支持以下项目的安全性：

- 使用 Pax Web War 扩展程序在 Fuse 上部署的经典 WAR 应用程序
- 使用 Pax Web Whiteboard Extender 在 Fuse 上部署的 servlets
- 使用 Camel Jetty 组件运行的 Apache Camel Jetty 端点
- 在自己的独立 Jetty 引擎上运行 Apache CXF 端点
- CXF servlet 提供的默认引擎上运行的 Apache CXF 端点
- SSH 和 JMX 管理访问权限
- Hawtio 管理控制台

2.1.6.1. 保护 Fuse 6 中的 Web 应用程序

您必须首先安装 Red Hat Single Sign-On Karaf 功能。接下来，您需要根据您要保护的应用程序类型执行这些步骤。所有引用的 Web 应用程序都需要将 Red Hat Single Sign-On Jetty 验证器注入底层 Jetty 服务器。实现此操作的步骤取决于应用程序类型。详情如下所述。

2.1.6.2. 安装 Keycloak 功能

您必须首先在 JBoss Fuse 环境中安装 **keycloak** 功能。keycloak 功能包括 Fuse 适配器和所有第三方依赖项。您可以从 Maven 存储库或从存档安装它。

2.1.6.2.1. 从 Maven 存储库安装

先决条件

- 您必须在线，并有权访问 Maven 存储库。
- 对于 Red Hat Single Sign-On，配置适当的 Maven 存储库，以便您可以安装工件。如需更多信息，请参阅 JBoss Enterprise Maven 存储库 页面。
  假设 Maven 存储库为 https://maven.repository.redhat.com/ga/，将以下内容添加到 **$FUSE_HOME/etc/org.ops4j.pax.url.mvn.cfg** 文件中，并将存储库添加到支持的存储库列表中。例如：

```
org.ops4j.pax.url.mvn.repositories= \
  https://maven.repository.redhat.com/ga/@id=redhat.product.repo
  http://repo1.maven.org/maven2@id=maven.central.repo, \
  ...
```

流程

1. 启动 JBoss Fuse 6.3.0 Rollup 12
2. 在 Karaf 终端类型中：

```
features:addurl mvn:org.keycloak/keycloak-osgi-features/18.0.14.redhat-00001/xml/features
features:install keycloak
```

3. 您可能还需要安装 Jetty 9 功能：

```
features:install keycloak-jetty9-adapter
```

4. 确保已安装功能：

```
features:list | grep keycloak
```

2.1.6.2.2. 从 ZIP 捆绑包安装

如果您离线或者不想使用 Maven 获取 JAR 文件和其他工件，则此安装选项很有用。

流程

1. 从 Sotware Downloads 站点下载 Red Hat Single Sign-On Fuse adapter ZIP 存档。
2. 将它解压缩到 JBoss Fuse 的根目录中。然后，依赖项会在 系统 目录下安装。您可以覆盖所有现有 jar 文件。

将其用于 JBoss Fuse 6.3.0 Rollup 12：

```
cd /path-to-fuse/jboss-fuse-6.3.0.redhat-254
unzip -q /path-to-adapter-zip/rh-sso-7.6.9-fuse-adapter.zip
```

3. 启动 Fuse 并在 fuse/karaf 终端中运行这些命令：

```
features:addurl mvn:org.keycloak/keycloak-osgi-features/18.0.14.redhat-00001/xml/features
features:install keycloak
```

4. 安装对应的 Jetty 适配器。由于工件直接在 JBoss Fuse 系统 目录中提供，因此您不需要使用 Maven 存储库。

2.1.6.3. 保护 Classic WAR 应用程序

流程

1. 在 **/WEB-INF/web.xml** 文件中，声明必要的：
   - <security-constraint> 元素中的安全限制
   - <login-config> 元素中的登录配置
   - <security-role> 元素中的安全角色。
     例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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_3_0.xsd"
         version="3.0">

    <module-name>customer-portal</module-name>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Customers</web-resource-name>
            <url-pattern>/customers/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
    </security-constraint>

    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>does-not-matter</realm-name>
    </login-config>
```

```
    <security-role>
      <role-name>admin</role-name>
    </security-role>
    <security-role>
      <role-name>user</role-name>
    </security-role>
  </web-app>
```

2. 将带有验证器的 **jetty-web.xml** 文件添加到 **/WEB-INF/jetty-web.xml** 文件。
例如：

```
<?xml version="1.0"?>
<!DOCTYPE Configure PUBLIC "-//Mort Bay Consulting//DTD Configure//EN"
 "http://www.eclipse.org/jetty/configure_9_0.dtd">
<Configure class="org.eclipse.jetty.webapp.WebAppContext">
  <Get name="securityHandler">
    <Set name="authenticator">
      <New class="org.keycloak.adapters.jetty.KeycloakJettyAuthenticator">
      </New>
    </Set>
  </Get>
</Configure>
```

3. 在 WAR 的 **/WEB-INF/** 目录中，创建一个新文件 keycloak.json。此配置文件的格式在 Java Adapters Config 部分中进行了描述。也可以使此文件在外部可用，如 配置外部适配器 中所述。

4. 确保您的 WAR 应用程序导入 **org.keycloak.adapters.jetty**，并在 **Import-Package** 标头下导入 **META-INF/MANIFEST.MF** 文件中的一些软件包。在项目中使用 **maven-bundle-plugin** 可以正确地生成清单中的 OSGI 标头。请注意，软件包的"*"解析不会导入 **org.keycloak.adapters.jetty** 软件包，因为它不供应用程序或 Blueprint 或 Spring 描述符使用，而是在 **jetty-web.xml** 文件中使用。
要导入的软件包列表可能类似如下：

```
org.keycloak.adapters.jetty;version="18.0.14.redhat-00001",
org.keycloak.adapters;version="18.0.14.redhat-00001",
org.keycloak.constants;version="18.0.14.redhat-00001",
org.keycloak.util;version="18.0.14.redhat-00001",
org.keycloak.*;version="18.0.14.redhat-00001",
*;resolution:=optional
```

2.1.6.3.1. 配置外部适配器

如果您不希望 **keycloak.json** 适配器配置文件捆绑在 WAR 应用程序内，但根据命名约定使外部和加载可用，请使用这个配置方法。

要启用这个功能，请将本节添加到您的 **/WEB_INF/web.xml** 文件中：

```
<context-param>
  <param-name>keycloak.config.resolver</param-name>
  <param-value>org.keycloak.adapters.osgi.PathBasedKeycloakConfigResolver</param-value>
</context-param>
```

该组件使用 **keycloak.config** 或 **karaf.etc** java 属性来搜索基本文件夹来定位配置。然后，在这些文件夹中搜索一个名为 < **your_web_context>-keycloak.json** 的文件。

例如，如果您的 Web 应用程序具有上下文 **my-portal**，则您的适配器配置会从 **$FUSE_HOME/etc/my-portal-keycloak.json** 文件加载。

2.1.6.4. 保护部署为 OSGI 服务的 servlet

如果您在 OSGI 捆绑的项目中有一个 servlet 类，但没有部署为典型的 WAR 应用，则可以使用这个方法。Fuse 使用 Pax Web Whiteboard expansioner 来部署如 web 应用程序的 servlet。

流程

1. Red Hat Single Sign-On 提供 **org.keycloak.adapters.osgi.undertow.PaxWebIntegrationService**，它允许注入 jetty-web.xml 并为应用程序配置安全限制。您需要在应用程序的 **OSGI-INF/blueprint/blueprint.xml** 文件中声明这些服务。请注意，您的 servlet 需要依赖于它。配置示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
           http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <!-- Using jetty bean just for the compatibility with other fuse services -->
    <bean id="servletConstraintMapping" class="org.eclipse.jetty.security.ConstraintMapping">
        <property name="constraint">
            <bean class="org.eclipse.jetty.util.security.Constraint">
                <property name="name" value="cst1"/>
                <property name="roles">
                    <list>
                        <value>user</value>
                    </list>
                </property>
                <property name="authenticate" value="true"/>
                <property name="dataConstraint" value="0"/>
            </bean>
        </property>
        <property name="pathSpec" value="/product-portal/*"/>
    </bean>

    <bean id="keycloakPaxWebIntegration"
class="org.keycloak.adapters.osgi.PaxWebIntegrationService"
          init-method="start" destroy-method="stop">
        <property name="jettyWebXmlLocation" value="/WEB-INF/jetty-web.xml" />
        <property name="bundleContext" ref="blueprintBundleContext" />
        <property name="constraintMappings">
            <list>
                <ref component-id="servletConstraintMapping" />
            </list>
        </property>
    </bean>

    <bean id="productServlet" class="org.keycloak.example.ProductPortalServlet" depends-
on="keycloakPaxWebIntegration">
    </bean>

    <service ref="productServlet" interface="javax.servlet.Servlet">
        <service-properties>
```

```
            <entry key="alias" value="/product-portal" />
            <entry key="servlet-name" value="ProductServlet" />
            <entry key="keycloak.config.file" value="/keycloak.json" />
        </service-properties>
    </service>

</blueprint>
```

- 您可能需要在项目中具有 **WEB-INF** 目录（即使您的项目不是 Web 应用程序），并创建 **/WEB-INF/jetty-web.xml** 和 **/WEB-INF/keycloak.json** 文件，如 Classic WAR 应用 部分中所示。请注意，您不需要 **web.xml** 文件，因为蓝图配置文件中声明 security-constraints。

2. **META-INF/MANIFEST.MF** 中的 **Import-Package** 必须至少包含这些导入：

```
org.keycloak.adapters.jetty;version="18.0.14.redhat-00001",
org.keycloak.adapters;version="18.0.14.redhat-00001",
org.keycloak.constants;version="18.0.14.redhat-00001",
org.keycloak.util;version="18.0.14.redhat-00001",
org.keycloak.*;version="18.0.14.redhat-00001",
*;resolution:=optional
```

2.1.6.5. 保护 Apache Camel 应用程序

您可以通过添加带有 **KeycloakJettyAuthenticator** 的 securityHandler 并注入了正确的安全限制，来保护使用 camel-jetty etty 组件实施的 Apache Camel 端点。您可以使用类似配置将 **OSGI-INF/blueprint/blueprint.xml** 文件添加到 Camel 应用程序。角色、安全约束映射和 Red Hat Single Sign-On 适配器配置可能会根据您的环境和需求稍有不同。

例如：

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

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:camel="http://camel.apache.org/schema/blueprint"
           xsi:schemaLocation="
      http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
      http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-
blueprint.xsd">

   <bean id="kcAdapterConfig" class="org.keycloak.representations.adapters.config.AdapterConfig">
      <property name="realm" value="demo"/>
      <property name="resource" value="admin-camel-endpoint"/>
      <property name="bearerOnly" value="true"/>
      <property name="authServerUrl" value="http://localhost:8080/auth" />
      <property name="sslRequired" value="EXTERNAL"/>
   </bean>

   <bean id="keycloakAuthenticator" class="org.keycloak.adapters.jetty.KeycloakJettyAuthenticator">
      <property name="adapterConfig" ref="kcAdapterConfig"/>
   </bean>

   <bean id="constraint" class="org.eclipse.jetty.util.security.Constraint">
      <property name="name" value="Customers"/>
```

```
        <property name="roles">
            <list>
                <value>admin</value>
            </list>
        </property>
        <property name="authenticate" value="true"/>
        <property name="dataConstraint" value="0"/>
    </bean>

    <bean id="constraintMapping" class="org.eclipse.jetty.security.ConstraintMapping">
        <property name="constraint" ref="constraint"/>
        <property name="pathSpec" value="/*"/>
    </bean>

    <bean id="securityHandler" class="org.eclipse.jetty.security.ConstraintSecurityHandler">
        <property name="authenticator" ref="keycloakAuthenticator" />
        <property name="constraintMappings">
            <list>
                <ref component-id="constraintMapping" />
            </list>
        </property>
        <property name="authMethod" value="BASIC"/>
        <property name="realmName" value="does-not-matter"/>
    </bean>

    <bean id="sessionHandler" class="org.keycloak.adapters.jetty.spi.WrappingSessionHandler">
        <property name="handler" ref="securityHandler" />
    </bean>

    <bean id="helloProcessor" class="org.keycloak.example.CamelHelloProcessor" />

    <camelContext id="blueprintContext"
                  trace="false"
                  xmlns="http://camel.apache.org/schema/blueprint">
        <route id="httpBridge">
            <from uri="jetty:http://0.0.0.0:8383/admin-camel-endpoint?
handlers=sessionHandler&amp;matchOnUriPrefix=true" />
            <process ref="helloProcessor" />
            <log message="The message from camel endpoint contains ${body}"/>
        </route>
    </camelContext>

</blueprint>
```

- **META-INF/MANIFEST.MF** 中的 **Import-Package** 需要包含以下导入：

```
javax.servlet;version="[3,4)",
javax.servlet.http;version="[3,4)",
org.apache.camel.*,
org.apache.camel;version="[2.13,3)",
org.eclipse.jetty.security;version="[9,10)",
org.eclipse.jetty.server.nio;version="[9,10)",
org.eclipse.jetty.util.security;version="[9,10)",
org.keycloak.*;version="18.0.14.redhat-00001",
```

```
org.osgi.service.blueprint,
org.osgi.service.blueprint.container,
org.osgi.service.event,
```

2.1.6.6. Camel RestDSL

Camel RestDSL 是一个 Camel 功能，用于以流畅的方式定义您的 REST 端点。但是，您仍必须使用特定的实施类，并提供了有关如何与 Red Hat Single Sign-On 集成的说明。

配置集成机制的方法取决于您为其配置 RestDSL 路由的 Camel 组件。

以下示例演示了如何使用 Jetty 组件配置集成，并引用前面 Blueprint 示例中定义的某些 Bean。

```
<bean id="securityHandlerRest" class="org.eclipse.jetty.security.ConstraintSecurityHandler">
   <property name="authenticator" ref="keycloakAuthenticator" />
   <property name="constraintMappings">
      <list>
         <ref component-id="constraintMapping" />
      </list>
   </property>
   <property name="authMethod" value="BASIC"/>
   <property name="realmName" value="does-not-matter"/>
</bean>

<bean id="sessionHandlerRest" class="org.keycloak.adapters.jetty.spi.WrappingSessionHandler">
   <property name="handler" ref="securityHandlerRest" />
</bean>


<camelContext id="blueprintContext"
              trace="false"
              xmlns="http://camel.apache.org/schema/blueprint">

   <restConfiguration component="jetty" contextPath="/restdsl"
                      port="8484">
      <!--the link with Keycloak security handlers happens here-->
      <endpointProperty key="handlers" value="sessionHandlerRest"></endpointProperty>
      <endpointProperty key="matchOnUriPrefix" value="true"></endpointProperty>
   </restConfiguration>

   <rest path="/hello" >
      <description>Hello rest service</description>
      <get uri="/{id}" outType="java.lang.String">
         <description>Just an helllo</description>
         <to uri="direct:justDirect" />
      </get>

   </rest>

   <route id="justDirect">
      <from uri="direct:justDirect"/>
      <process ref="helloProcessor" />
      <log message="RestDSL correctly invoked ${body}"/>
      <setBody>
         <constant>(__This second sentence is returned from a Camel RestDSL
```

```
endpoint__)</constant>
        </setBody>
    </route>

</camelContext>
```

2.1.6.7. 在单独的 Jetty 引擎上保护 Apache CXF 端点

流程

要在单独的 Jetty 引擎上运行由 Red Hat Single Sign-On 保护的 CXF 端点，请执行以下步骤。

1. 将 **META-INF/spring/beans.xml** 添加到应用程序中，并在其中声明 **httpj:engine-factory** with Jetty SecurityHandler with injected **KeycloakJettyAuthenticator**。CFX JAX-WS 应用的配置可能类似以下：

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

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xmlns:httpj="http://cxf.apache.org/transports/http-jetty/configuration"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd
        http://www.springframework.org/schema/osgi
http://www.springframework.org/schema/osgi/spring-osgi.xsd
        http://cxf.apache.org/transports/http-jetty/configuration
http://cxf.apache.org/schemas/configuration/http-jetty.xsd">

    <import resource="classpath:META-INF/cxf/cxf.xml" />

    <bean id="kcAdapterConfig"
class="org.keycloak.representations.adapters.config.AdapterConfig">
        <property name="realm" value="demo"/>
        <property name="resource" value="custom-cxf-endpoint"/>
        <property name="bearerOnly" value="true"/>
        <property name="authServerUrl" value="http://localhost:8080/auth" />
        <property name="sslRequired" value="EXTERNAL"/>
    </bean>

    <bean id="keycloakAuthenticator"
class="org.keycloak.adapters.jetty.KeycloakJettyAuthenticator">
        <property name="adapterConfig">
            <ref local="kcAdapterConfig" />
        </property>
    </bean>

    <bean id="constraint" class="org.eclipse.jetty.util.security.Constraint">
        <property name="name" value="Customers"/>
        <property name="roles">
            <list>
                <value>user</value>
            </list>
        </property>
```

```
        <property name="authenticate" value="true"/>
        <property name="dataConstraint" value="0"/>
    </bean>

    <bean id="constraintMapping" class="org.eclipse.jetty.security.ConstraintMapping">
        <property name="constraint" ref="constraint"/>
        <property name="pathSpec" value="/*"/>
    </bean>

    <bean id="securityHandler" class="org.eclipse.jetty.security.ConstraintSecurityHandler">
        <property name="authenticator" ref="keycloakAuthenticator" />
        <property name="constraintMappings">
            <list>
                <ref local="constraintMapping" />
            </list>
        </property>
        <property name="authMethod" value="BASIC"/>
        <property name="realmName" value="does-not-matter"/>
    </bean>

    <httpj:engine-factory bus="cxf" id="kc-cxf-endpoint">
        <httpj:engine port="8282">
            <httpj:handlers>
                <ref local="securityHandler" />
            </httpj:handlers>
            <httpj:sessionSupport>true</httpj:sessionSupport>
        </httpj:engine>
    </httpj:engine-factory>

    <jaxws:endpoint
                implementor="org.keycloak.example.ws.ProductImpl"
                address="http://localhost:8282/ProductServiceCF" depends-on="kc-cxf-endpoint"
/>

</beans>
```

对于 CXF JAX-RS 应用程序，唯一区别可能是取决于 engine-factory 的端点的配置：

```
<jaxrs:server serviceClass="org.keycloak.example.rs.CustomerService"
address="http://localhost:8282/rest"
    depends-on="kc-cxf-endpoint">
    <jaxrs:providers>
        <bean class="com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider" />
    </jaxrs:providers>
</jaxrs:server>
```

2. **META-INF/MANIFEST.MF** 中的 **Import-Package** 必须包含这些导入：

```
META-INF.cxf;version="[2.7,3.2)",
META-INF.cxf.osgi;version="[2.7,3.2)";resolution:=optional,
org.apache.cxf.bus;version="[2.7,3.2)",
org.apache.cxf.bus.spring;version="[2.7,3.2)",
org.apache.cxf.bus.resource;version="[2.7,3.2)",
org.apache.cxf.transport.http;version="[2.7,3.2)",
org.apache.cxf.*;version="[2.7,3.2)",
```

```
org.springframework.beans.factory.config,
org.eclipse.jetty.security;version="[9,10)",
org.eclipse.jetty.util.security;version="[9,10)",
org.keycloak.*;version="18.0.14.redhat-00001"
```

2.1.6.8. 在默认的 Jetty Engine 上保护 Apache CXF 端点

某些服务在启动时自动附带部署的 servlet。一个这样的服务是在 http://localhost:8181/cxf 上下文中运行的 CXF servlet。保护此类端点可能比较复杂。目前使用 Red Hat Single Sign-On 的一种方法是 ServletReregistrationService，它在启动时取消了一个内置 servlet，可让您在 Red Hat Single Sign-On 保护的上下文上重新部署。

应用程序中的配置文件 **OSGI-INF/blueprint/blueprint.xml** 可能类似以下。请注意，它会添加 JAX-RS **customerservice** 端点，该端点特定于您的应用程序，但更重要的是，可以保护整个 **/cxf** 上下文。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:jaxrs="http://cxf.apache.org/blueprint/jaxrs"
           xsi:schemaLocation="
    http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
    http://cxf.apache.org/blueprint/jaxrs http://cxf.apache.org/schemas/blueprint/jaxrs.xsd">

    <!-- JAXRS Application -->

    <bean id="customerBean" class="org.keycloak.example.rs.CxfCustomerService" />

    <jaxrs:server id="cxfJaxrsServer" address="/customerservice">
        <jaxrs:providers>
            <bean class="com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider" />
        </jaxrs:providers>
        <jaxrs:serviceBeans>
            <ref component-id="customerBean" />
        </jaxrs:serviceBeans>
    </jaxrs:server>


    <!-- Securing of whole /cxf context by unregister default cxf servlet from paxweb and re-register
    with applied security constraints -->

    <bean id="cxfConstraintMapping" class="org.eclipse.jetty.security.ConstraintMapping">
        <property name="constraint">
            <bean class="org.eclipse.jetty.util.security.Constraint">
                <property name="name" value="cst1"/>
                <property name="roles">
                    <list>
                        <value>user</value>
                    </list>
                </property>
                <property name="authenticate" value="true"/>
                <property name="dataConstraint" value="0"/>
            </bean>
        </property>
        <property name="pathSpec" value="/cxf/*"/>
    </bean>
```

```
    <bean id="cxfKeycloakPaxWebIntegration"
class="org.keycloak.adapters.osgi.PaxWebIntegrationService"
          init-method="start" destroy-method="stop">
        <property name="bundleContext" ref="blueprintBundleContext" />
        <property name="jettyWebXmlLocation" value="/WEB-INF/jetty-web.xml" />
        <property name="constraintMappings">
            <list>
                <ref component-id="cxfConstraintMapping" />
            </list>
        </property>
    </bean>

    <bean id="defaultCxfReregistration"
class="org.keycloak.adapters.osgi.ServletReregistrationService" depends-
on="cxfKeycloakPaxWebIntegration"
          init-method="start" destroy-method="stop">
        <property name="bundleContext" ref="blueprintBundleContext" />
        <property name="managedServiceReference">
            <reference interface="org.osgi.service.cm.ManagedService" filter="
(service.pid=org.apache.cxf.osgi)" timeout="5000"  />
        </property>
    </bean>

</blueprint>
```

因此，在默认 CXF HTTP 目标上运行的所有其他 CXF 服务也受到保护。同样，当应用程序取消部署时，整个 **/cxf** 上下文也会变得不安全。因此，为您的应用程序使用您自己的 Jetty 引擎，如 在单独的 Jetty Engine 上安全 CXF 应用程序 中所述，您可以更好地控制每个应用程序的安全性。

- **WEB-INF** 目录可能需要位于您的项目内（即使项目不是 Web 应用）。您可能还需要以类似 Classic WAR 应用 的方式编辑 **/WEB-INF/jetty-web.xml** 和 **/WEB-INF/keycloak.json** 文件。请注意，您不需要 **web.xml** 文件，因为蓝图配置文件中声明安全限制。

- **META-INF/MANIFEST.MF** 中的 **Import-Package** 必须包含以下导入：

```
META-INF.cxf;version="[2.7,3.2)",
META-INF.cxf.osgi;version="[2.7,3.2)";resolution:=optional,
org.apache.cxf.transport.http;version="[2.7,3.2)",
org.apache.cxf.*;version="[2.7,3.2)",
com.fasterxml.jackson.jaxrs.json;version="[2.5,3)",
org.eclipse.jetty.security;version="[9,10)",
org.eclipse.jetty.util.security;version="[9,10)",
org.keycloak.*;version="18.0.14.redhat-00001",
org.keycloak.adapters.jetty;version="18.0.14.redhat-00001",
*;resolution:=optional
```

2.1.6.9. 保护 Fuse 管理服务

2.1.6.9.1. 使用 SSH 身份验证到 Fuse Terminal

Red Hat Single Sign-On 主要解决 Web 应用程序身份验证的用例；但是，如果您的其他 Web 服务和应用程序通过红帽单点登录保护，则保护非 Web 管理服务（如使用红帽单点登录凭证的 SSH）是最佳选择。您可以使用 JAAS 登录模块进行此操作，该模块允许远程连接到 Red Hat Single Sign-On，并根据 Resource Owner Password 凭据验证凭证。

要启用 SSH 身份验证，请执行以下步骤。

流程

1. 在 Red Hat Single Sign-On 中创建一个客户端（例如，**ssh-jmx-admin-client**），它将用于 SSH 身份验证。此客户端需要选择 **Direct Access Grants Enabled** to **On**。
2. 在 **$FUSE_HOME/etc/org.apache.karaf.shell.cfg** 文件中，更新或指定此属性：

```
sshRealm=keycloak
```

3. 添加 **$FUSE_HOME/etc/keycloak-direct-access.json** 文件，其内容类似于以下内容（基于您的环境和 Red Hat Single Sign-On 客户端设置）：

```
{
    "realm": "demo",
    "resource": "ssh-jmx-admin-client",
    "ssl-required" : "external",
    "auth-server-url" : "http://localhost:8080/auth",
    "credentials": {
        "secret": "password"
    }
}
```

此文件指定客户端应用程序配置，供 **keycloak** JAAS 域中的 JAAS DirectAccessGrantsLoginModule 用于 SSH 身份验证。

4. 启动 Fuse 并安装 **keycloak** JAAS 域。最简单的方法是安装 **keycloak-jaas** 功能，该功能已预定义了 JAAS 域。您可以使用自己的 **keycloak** JAAS 域覆盖功能的预定义域。详情请查看 JBoss Fuse 文档。
在 Fuse 终端中使用这些命令：

```
features:addurl mvn:org.keycloak/keycloak-osgi-features/18.0.14.redhat-00001/xml/features
features:install keycloak-jaas
```

5. 在终端中输入以下内容，以 **admin** 用户身份使用 SSH 登录：

```
ssh -o PubkeyAuthentication=no -p 8101 admin@localhost
```

6. 使用密码 **password** 登录。

注意

在稍后的一些操作系统中，您可能还需要使用 SSH 命令的 -o 选项 **-o HostKeyAlgorithms=+ssh-dss** =+ssh-dss，因为之后的 SSH 客户端不允许使用 **ssh-dss** 算法。但是，默认情况下，它目前在 JBoss Fuse 6.3.0 Rollup 12 中使用。

请注意，用户需要具有 realm 角色 **admin** 来执行所有操作或其他角色才能执行操作子集（例如，**viewer** 角色来限制用户仅运行只读 Karaf 命令）。可用的角色在 **$FUSE_HOME/etc/org.apache.karaf.shell.cfg** 或 **$FUSE_HOME/etc/system.properties** 中配置。

2.1.6.9.2. 使用 JMX 身份验证

如果要使用 jconsole 或其他外部工具通过 RMI 远程连接到 JMX，则可能需要使用 JMX 身份验证。否则，最好使用 hawt.io/jolokia，因为 jolokia 代理默认安装在 hawt.io 中。如需了解更多详细信息，请参阅 Hawtio 管理控制台。

流程

1. 在 **$FUSE_HOME/etc/org.apache.karaf.management.cfg** 文件中，将 jmxRealm 属性更改为：

```
jmxRealm=keycloak
```

2. 安装 **keycloak-jaas** 功能并配置 **$FUSE_HOME/etc/keycloak-direct-access.json** 文件，如上面的 SSH 部分所述。

3. 在 jconsole 中，您可以使用 URL，例如：

```
service:jmx:rmi://localhost:44444/jndi/rmi://localhost:1099/karaf-root
```

和凭证：admin/password （基于您环境具有管理员特权的用户）。

2.1.6.10. 保护 Hawtio 管理控制台

要使用红帽单点登录来保护 Hawtio 管理控制台，请执行以下步骤。

流程

1. 将这些属性添加到 **$FUSE_HOME/etc/system.properties** 文件中：

```
hawtio.keycloakEnabled=true
hawtio.realm=keycloak
hawtio.keycloakClientConfig=file://${karaf.base}/etc/keycloak-hawtio-client.json
hawtio.rolePrincipalClasses=org.keycloak.adapters.jaas.RolePrincipal,org.apache.karaf.jaas.boot.principal.RolePrincipal
```

2. 在域的 Red Hat Single Sign-On 管理控制台中创建客户端。例如，在 Red Hat Single Sign-On 演示 域中，创建一个客户端 **hawtio-client**，将 **public** 指定为 Access Type，并指定指向 Hawtio: http://localhost:8181/hawtio/* 的重定向 URI。您还必须配置对应的 Web Origin （本例中为 http://localhost:8181）。

3. 使用类似以下示例的内容，在 **$FUSE_HOME/etc** 目录中创建 **keycloak-hawtio-client.json** 文件。根据您的 Red Hat Single Sign-On 环境，更改 **realm**、**resource** 和 **auth-server-url** 属性。**resource** 属性必须指向上一步中创建的客户端。此文件供客户端(Hawtio JavaScript 应用)侧使用。

```
{
  "realm" : "demo",
  "resource" : "hawtio-client",
  "auth-server-url" : "http://localhost:8080/auth",
  "ssl-required" : "external",
  "public-client" : true
}
```

1. 使用类似以下示例的内容，在 **$FUSE_HOME/etc** dicrectory 中创建 **keycloak-hawtio.json** 文件。根据您的 Red Hat Single Sign-On 环境更改 **realm** 和 **auth-server-url** 属性。此文件供服务器(JAAS Login 模块)侧使用。

```
{
  "realm" : "demo",
  "resource" : "jaas",
  "bearer-only" : true,
  "auth-server-url" : "http://localhost:8080/auth",
  "ssl-required" : "external",
  "use-resource-role-mappings": false,
  "principal-attribute": "preferred_username"
}
```

2. 启动 JBoss Fuse 6.3.0 Rollup 12 并安装 keycloak 功能（如果您还没有这样做）。Karaf 终端中的命令类似以下示例：

```
features:addurl mvn:org.keycloak/keycloak-osgi-features/18.0.14.redhat-00001/xml/features
features:install keycloak
```

3. 进入 http://localhost:8181/hawtio，以用户从 Red Hat Single Sign-On 域登录。
请注意，用户需要具有正确的 realm 角色才能成功向 Hawtio 进行身份验证。可用的角色在 **hawtio.roles** 中的 **$FUSE_HOME/etc/system.properties** 文件中配置。

2.1.6.10.1. 在 JBoss EAP 6.4 上保护 Hawtio

先决条件

设置 Red Hat Single Sign-On，如 保护 Hawtio 管理控制台 中所述。假设：

- 您有一个 Red Hat Single Sign-On realm **demo** 和 client **hawtio-client**
- 您的 Red Hat Single Sign-On 在 **localhost:8080**上运行
- 带有 Hawtio 的 JBoss EAP 6.4 服务器将在 **localhost:8181** 上运行。此服务器的目录在后续步骤中称为 **$EAP_HOME**。

流程

1. 将 **hawtio-wildfly-1.4.0.redhat-630396.war** 存档复制到 **$EAP_HOME/standalone/configuration** 目录。有关部署 Hawtio 的详情，请查看 Fuse Hawtio 文档。
2. 将带有上述内容的 **keycloak-hawtio.json** 和 **keycloak-hawtio-client.json** 文件复制到 **$EAP_HOME/standalone/configuration** 目录。
3. 将 Red Hat Single Sign-On adapter 子系统安装到 JBoss EAP 6.4 服务器，如 JBoss 适配器文档所述。
4. 在 **$EAP_HOME/standalone/configuration/standalone.xml** 文件中，配置系统属性，如下例所示：

```
<extensions>
...
</extensions>

<system-properties>
    <property name="hawtio.authenticationEnabled" value="true" />
    <property name="hawtio.realm" value="hawtio" />
```

```
    <property name="hawtio.roles" value="admin,viewer" />
    <property name="hawtio.rolePrincipalClasses"
value="org.keycloak.adapters.jaas.RolePrincipal" />
    <property name="hawtio.keycloakEnabled" value="true" />
    <property name="hawtio.keycloakClientConfig" value="${jboss.server.config.dir}/keycloak-
hawtio-client.json" />
    <property name="hawtio.keycloakServerConfig" value="${jboss.server.config.dir}/keycloak-
hawtio.json" />
</system-properties>
```

5. 将 Hawtio 域添加到 **security-domains** 部分中的同一文件中：

```
<security-domain name="hawtio" cache-type="default">
    <authentication>
        <login-module code="org.keycloak.adapters.jaas.BearerTokenLoginModule"
flag="required">
            <module-option name="keycloak-config-file"
value="${hawtio.keycloakServerConfig}"/>
        </login-module>
    </authentication>
</security-domain>
```

6. 将 **secure-deployment** 部分 **hawtio** 添加到 adapter 子系统。这样可确保 Hawtio WAR 可以找到 JAAS 登录模块类。

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
    <secure-deployment name="hawtio-wildfly-1.4.0.redhat-630396.war" />
</subsystem>
```

7. 使用 Hawtio 重启 JBoss EAP 6.4 服务器：

```
cd $EAP_HOME/bin
./standalone.sh -Djboss.socket.binding.port-offset=101
```

8. access Hawtio at http://localhost:8181/hawtio.它通过 Red Hat Single Sign-On 进行保护。

2.1.7. JBoss Fuse 7 Adapter

Red Hat Single Sign-On 支持保护在 JBoss Fuse 7 中运行的 Web 应用程序。

JBoss Fuse 7 利用 Undertow 适配器，与 JBoss EAP 7 适配器相同，JBoss Fuse 7.4.0 与 Undertow HTTP 引擎捆绑在一起，而 Undertow 则用于运行各种 Web 应用。

警告

唯一支持的 Fuse 7 版本是最新的版本。如果您使用早期版本的 Fuse 7，则某些功能可能无法正常工作。特别是，集成不适用于低于 7.12.0 的 Fuse 7 版本。

Fuse 支持以下项目的安全性：

- 使用 Pax Web War 扩展程序在 Fuse 上部署的经典 WAR 应用程序
- 使用 Pax Web Whiteboard Extender 在 Fuse 上部署的 servlets 作为 OSGI 服务，另外通过 org.osgi.service.http.HttpService#registerServlet （）进行注册，它是标准的 OSGi Enterprise HTTP Service
- 使用 Camel Undertow 组件运行的 Apache Camel Undertow 端点
- 在自己的独立 Undertow 引擎上运行 Apache CXF 端点
- CXF servlet 提供的默认引擎上运行的 Apache CXF 端点
- SSH 和 JMX 管理访问权限
- Hawtio 管理控制台

2.1.7.1. 在 Fuse 7 中保护您的 Web 应用程序

您必须首先安装 Red Hat Single Sign-On Karaf 功能。接下来，您需要根据您要保护的应用程序类型执行这些步骤。所有引用的 Web 应用都需要将 Red Hat Single Sign-On Undertow 身份验证机制注入底层 Web 服务器。实现此操作的步骤取决于应用程序类型。详情如下所述。

2.1.7.2. 安装 Keycloak 功能

您必须首先在 JBoss Fuse 环境中安装 **keycloak-pax-http-undertow** 和 **keycloak-jaas** 功能。**keycloak-pax-http-undertow** 功能包括 Fuse 适配器和所有第三方依赖项。 **keycloak-jaas** 包含用于 SSH 和 JMX 身份验证的域中使用的 JAAS 模块。您可以从 Maven 存储库或从存档安装它。

2.1.7.2.1. 从 Maven 存储库安装

先决条件

- 您必须在线，并有权访问 Maven 存储库。
- 对于 Red Hat Single Sign-On，配置适当的 Maven 存储库，以便您可以安装工件。如需更多信息，请参阅 JBoss Enterprise Maven 存储库 页面。
- 假设 Maven 存储库为 https://maven.repository.redhat.com/ga/，将以下内容添加到 **$FUSE_HOME/etc/org.ops4j.pax.url.mvn.cfg** 文件中，并将存储库添加到支持的存储库列表中。例如：

```
config:edit org.ops4j.pax.url.mvn
config:property-append org.ops4j.pax.url.mvn.repositories
,https://maven.repository.redhat.com/ga/@id=redhat.product.repo
config:update

feature:repo-refresh
```

流程

1. 启动 JBoss Fuse 7.4.0
2. 在 Karaf 终端中，键入：

```
feature:repo-add mvn:org.keycloak/keycloak-osgi-features/18.0.14.redhat-00001/xml/features
feature:install keycloak-pax-http-undertow keycloak-jaas
```

3. 您可能还需要安装 Undertow 功能：

```
feature:install pax-web-http-undertow
```

4. 确保已安装功能：

```
feature:list | grep keycloak
```

2.1.7.2.2. 从 ZIP 捆绑包安装

如果您离线或者不想使用 Maven 获取 JAR 文件和其他工件，这非常有用。

流程

1. 从 Sotware Downloads 站点下载 Red Hat Single Sign-On Fuse adapter ZIP 存档。
2. 将它解压缩到 JBoss Fuse 的根目录中。然后，依赖项会在 系统 目录下安装。您可以覆盖所有现有 jar 文件。
   将其用于 JBoss Fuse 7.4.0 ：

```
cd /path-to-fuse/fuse-karaf-7.z
unzip -q /path-to-adapter-zip/rh-sso-7.6.9-fuse-adapter.zip
```

3. 启动 Fuse 并在 fuse/karaf 终端中运行这些命令：

```
feature:repo-add mvn:org.keycloak/keycloak-osgi-features/18.0.14.redhat-00001/xml/features
feature:install keycloak-pax-http-undertow keycloak-jaas
```

4. 安装对应的 Undertow 适配器。由于工件直接在 JBoss Fuse 系统 目录中提供，因此您不需要使用 Maven 存储库。

2.1.7.3. 保护 Classic WAR 应用程序

流程

1. 在 **/WEB-INF/web.xml** 文件中，声明必要的：
   - <security-constraint> 元素中的安全限制
   - <login-config> 元素中的登录配置。确保 < **auth-method>** 是 **KEYCLOAK**。
   - <security-role> 元素中的安全角色
     例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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_3_0.xsd"
```

```
        version="3.0">

    <module-name>customer-portal</module-name>

    <welcome-file-list>
       <welcome-file>index.html</welcome-file>
    </welcome-file-list>

    <security-constraint>
       <web-resource-collection>
          <web-resource-name>Customers</web-resource-name>
          <url-pattern>/customers/*</url-pattern>
       </web-resource-collection>
       <auth-constraint>
          <role-name>user</role-name>
       </auth-constraint>
    </security-constraint>

    <login-config>
       <auth-method>KEYCLOAK</auth-method>
       <realm-name>does-not-matter</realm-name>
    </login-config>

    <security-role>
       <role-name>admin</role-name>
    </security-role>
    <security-role>
       <role-name>user</role-name>
    </security-role>
  </web-app>
```

2. 在 WAR 的 /**WEB-INF**/ 目录中，创建一个新文件 keycloak.json。此配置文件的格式在 Java Adapters Config 部分中进行了描述。也可以使此文件在外部可用，如 配置外部适配器 中所述。例如：

```
{
   "realm": "demo",
   "resource": "customer-portal",
   "auth-server-url": "http://localhost:8080/auth",
   "ssl-required" : "external",
   "credentials": {
      "secret": "password"
   }
}
```

3. 与 Fuse 6 适配器不同，MANIFEST.MF 中不需要特殊的 OSGi 导入。

2.1.7.3.1. 配置解析器

keycloak.json 适配器配置文件可以存储在捆绑包中，该捆绑包是默认的行为，或者存储在文件系统的目录中。要指定配置文件的实际源，请将 **keycloak.config.resolver** 部署参数设置为所需的配置解析器类。例如，在典型的 WAR 应用程序中，在 **web.xml** 文件中设置 **keycloak.config.resolver** 上下文参数，如下所示：

```
<context-param>
```

```
    <param-name>keycloak.config.resolver</param-name>
    <param-value>org.keycloak.adapters.osgi.PathBasedKeycloakConfigResolver</param-value>
</context-param>
```

以下解析器可用于 **keycloak.config.resolver** ：

org.keycloak.adapters.osgi.BundleBasedKeycloakConfigResolver

这是默认的解析器。该配置文件预期在 OSGi 捆绑包内受到保护。默认情况下，它会加载名为 **WEB-INF/keycloak.json** 的文件，但可通过 **configLocation** 属性配置此文件名。

org.keycloak.adapters.osgi.PathBasedKeycloakConfigResolver

此解析器在由 **keycloak.config** 系统属性指定的文件夹中搜索名为 & **lt;your_web_context>-keycloak.json** 的文件。如果没有设置 **keycloak.config**，则会使用 **karaf.etc** 系统属性。
例如，如果您的 Web 应用程序部署到上下文 **my-portal** 中，则您的适配器配置将从 **${keycloak.config}/my-portal-keycloak.json** 文件加载，或者从 **${karaf.etc}/my-portal-keycloak.json** 文件加载。

org.keycloak.adapters.osgi.HierarchicalPathBasedKeycloakConfigResolver

这个解析器与上述 **PathBasedKeycloakConfigResolver** 类似，其中给定 URI 路径，配置位置是从最到最具体的特定检查。
例如，对于 **/my/web-app/context** URI，会搜索以下配置位置是否存在，直到第一个位置存在：

- **${karaf.etc}/my-web-app-context-keycloak.json**
- **${karaf.etc}/my-web-app-keycloak.json**
- **${karaf.etc}/my-keycloak.json**
- **${karaf.etc}/keycloak.json**

2.1.7.4. 保护部署为 OSGI 服务的 servlet

如果您在 OSGI 捆绑的项目中有一个 servlet 类，但没有部署为典型的 WAR 应用，则可以使用这个方法。Fuse 使用 Pax Web Whiteboard expansioner 来部署如 web 应用程序的 servlet。

流程

1. Red Hat Single Sign-On 提供 **org.keycloak.adapters.osgi.undertow.PaxWebIntegrationService**，它允许为您的应用程序配置身份验证方法和安全限制。您需要在应用程序的 **OSGI-INF/blueprint/blueprint.xml** 文件中声明这些服务。请注意，您的 servlet 需要依赖于它。配置示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">

    <bean id="servletConstraintMapping"
class="org.keycloak.adapters.osgi.PaxWebSecurityConstraintMapping">
        <property name="roles">
            <list>
                <value>user</value>
            </list>
```

```
        </property>
        <property name="authentication" value="true"/>
        <property name="url" value="/product-portal/*"/>
    </bean>

    <!-- This handles the integration and setting the login-config and security-constraints
parameters -->
    <bean id="keycloakPaxWebIntegration"
class="org.keycloak.adapters.osgi.undertow.PaxWebIntegrationService"
          init-method="start" destroy-method="stop">
        <property name="bundleContext" ref="blueprintBundleContext" />
        <property name="constraintMappings">
            <list>
                <ref component-id="servletConstraintMapping" />
            </list>
        </property>
    </bean>

    <bean id="productServlet" class="org.keycloak.example.ProductPortalServlet" depends-
on="keycloakPaxWebIntegration" />

    <service ref="productServlet" interface="javax.servlet.Servlet">
        <service-properties>
            <entry key="alias" value="/product-portal" />
            <entry key="servlet-name" value="ProductServlet" />
            <entry key="keycloak.config.file" value="/keycloak.json" />
        </service-properties>
    </service>
</blueprint>
```

您可能需要在项目中具有 **WEB-INF** 目录（即使您的项目不是 Web 应用程序），并创建 **/WEB-INF/keycloak.json** 文件，如 Classic WAR application 部分所述。请注意，您不需要 **web.xml** 文件，因为蓝图配置文件中声明 security-constraints。

2. 与 Fuse 6 适配器不同，MANIFEST.MF 中不需要特殊的 OSGi 导入。

2.1.7.5. 保护 Apache Camel 应用程序

您可以通过蓝图注入正确的安全限制并将已使用组件更新至 **undertow-keycloak** 来保护使用 camel-undertow 组件的 Apache Camel 端点。您必须使用类似配置将 **OSGI-INF/blueprint/blueprint.xml** 文件添加到 Camel 应用程序。角色、安全约束映射和适配器配置可能会根据您的环境和需求稍有不同。

与标准的 **undertow** 组件相比，**undertow-keycloak** 组件添加了两个新属性：

- **configResolver** 是一个解析器 bean，提供 Red Hat Single Sign-On 适配器配置。可用的解析器列在 Configuration Resolvers 部分中。
- **allowedRoles** 是以逗号分隔的角色列表。访问该服务的用户必须至少具有允许访问的角色。

例如：

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:camel="http://camel.apache.org/schema/blueprint"
           xsi:schemaLocation="
```

```
      http://www.osgi.org/xmlns/blueprint/v1.0.0
http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
      http://camel.apache.org/schema/blueprint http://camel.apache.org/schema/blueprint/camel-
blueprint-2.17.1.xsd">

   <bean id="keycloakConfigResolver"
class="org.keycloak.adapters.osgi.BundleBasedKeycloakConfigResolver" >
       <property name="bundleContext" ref="blueprintBundleContext" />
   </bean>

   <bean id="helloProcessor" class="org.keycloak.example.CamelHelloProcessor" />

   <camelContext id="blueprintContext"
                 trace="false"
                 xmlns="http://camel.apache.org/schema/blueprint">

       <route id="httpBridge">
          <from uri="undertow-keycloak:http://0.0.0.0:8383/admin-camel-endpoint?
matchOnUriPrefix=true&amp;configResolver=#keycloakConfigResolver&amp;allowedRoles=admin"
/>
          <process ref="helloProcessor" />
          <log message="The message from camel endpoint contains ${body}"/>
       </route>

   </camelContext>

</blueprint>
```

- **META-INF/MANIFEST.MF** 中的 **Import-Package** 需要包含以下导入：

```
javax.servlet;version="[3,4)",
javax.servlet.http;version="[3,4)",
javax.net.ssl,
org.apache.camel.*,
org.apache.camel;version="[2.13,3)",
io.undertow.*,
org.keycloak.*;version="18.0.14.redhat-00001",
org.osgi.service.blueprint,
org.osgi.service.blueprint.container
```

2.1.7.6. Camel RestDSL

Camel RestDSL 是一个 Camel 功能，用于以流畅的方式定义您的 REST 端点。但是，您仍必须使用特定的实施类，并提供了有关如何与 Red Hat Single Sign-On 集成的说明。

配置集成机制的方法取决于您为其配置 RestDSL 路由的 Camel 组件。

以下示例演示了如何使用 **undertow-keycloak** 组件配置集成，并引用前面 Blueprint 示例中定义的一些 Bean。

```
<camelContext id="blueprintContext"
              trace="false"
              xmlns="http://camel.apache.org/schema/blueprint">

   <!--the link with Keycloak security handlers happens by using undertow-keycloak component -->
```

```
    <restConfiguration apiComponent="undertow-keycloak" contextPath="/restdsl" port="8484">
      <endpointProperty key="configResolver" value="#keycloakConfigResolver" />
      <endpointProperty key="allowedRoles" value="admin,superadmin" />
    </restConfiguration>

    <rest path="/hello" >
      <description>Hello rest service</description>
      <get uri="/{id}" outType="java.lang.String">
        <description>Just a hello</description>
        <to uri="direct:justDirect" />
      </get>

    </rest>

    <route id="justDirect">
      <from uri="direct:justDirect"/>
      <process ref="helloProcessor" />
      <log message="RestDSL correctly invoked ${body}"/>
      <setBody>
        <constant>(__This second sentence is returned from a Camel RestDSL
endpoint__)</constant>
      </setBody>
    </route>

</camelContext>
```

2.1.7.7. 在单独的 Undertow Engine 上保护 Apache CXF 端点

要在单独的 Undertow 引擎上运行由 Red Hat Single Sign-On 保护的 CXF 端点，请执行以下步骤。

流程

1. 将 **OSGI-INF/blueprint/blueprint.xml** 添加到应用程序中，并在其中添加类似于 Camel 配置的正确配置解析器 Bean。在 **httpu:engine-factory** 中，使用 camel 配置声明 **org.keycloak.adapters.osgi.undertow.CxfKeycloakAuthHandler** 处理程序。CFX JAX-WS 应用的配置可能类似以下：

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:jaxws="http://cxf.apache.org/blueprint/jaxws"
           xmlns:cxf="http://cxf.apache.org/blueprint/core"
           xmlns:httpu="http://cxf.apache.org/transports/http-undertow/configuration".
           xsi:schemaLocation="
      http://cxf.apache.org/transports/http-undertow/configuration
http://cxf.apache.org/schemas/configuration/http-undertow.xsd
      http://cxf.apache.org/blueprint/core http://cxf.apache.org/schemas/blueprint/core.xsd
      http://cxf.apache.org/blueprint/jaxws http://cxf.apache.org/schemas/blueprint/jaxws.xsd">

    <bean id="keycloakConfigResolver"
class="org.keycloak.adapters.osgi.BundleBasedKeycloakConfigResolver" >
        <property name="bundleContext" ref="blueprintBundleContext" />
    </bean>

    <httpu:engine-factory bus="cxf" id="kc-cxf-endpoint">
```

```
        <httpu:engine port="8282">
          <httpu:handlers>
            <bean class="org.keycloak.adapters.osgi.undertow.CxfKeycloakAuthHandler">
              <property name="configResolver" ref="keycloakConfigResolver" />
            </bean>
          </httpu:handlers>
        </httpu:engine>
      </httpu:engine-factory>

      <jaxws:endpoint implementor="org.keycloak.example.ws.ProductImpl"
                      address="http://localhost:8282/ProductServiceCF" depends-on="kc-cxf-
endpoint"/>

</blueprint>
```

对于 CXF JAX-RS 应用程序，唯一区别可能是取决于 engine-factory 的端点的配置：

```
<jaxrs:server serviceClass="org.keycloak.example.rs.CustomerService"
address="http://localhost:8282/rest"
   depends-on="kc-cxf-endpoint">
   <jaxrs:providers>
      <bean class="com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider" />
   </jaxrs:providers>
</jaxrs:server>
```

2. **META-INF/MANIFEST.MF** 中的 **Import-Package** 必须包含这些导入：

```
META-INF.cxf;version="[2.7,3.3)",
META-INF.cxf.osgi;version="[2.7,3.3)";resolution:=optional,
org.apache.cxf.bus;version="[2.7,3.3)",
org.apache.cxf.bus.spring;version="[2.7,3.3)",
org.apache.cxf.bus.resource;version="[2.7,3.3)",
org.apache.cxf.transport.http;version="[2.7,3.3)",
org.apache.cxf.*;version="[2.7,3.3)",
org.springframework.beans.factory.config,
org.keycloak.*;version="18.0.14.redhat-00001"
```

2.1.7.8. 在默认的 Undertow Engine 上保护 Apache CXF 端点

某些服务在启动时自动附带部署的 servlet。一个这样的服务是在 http://localhost:8181/cxf 上下文中运行的 CXF servlet。Fuse 的 Pax Web 支持通过配置管理员更改现有上下文。这可用于保护 Red Hat Single Sign-On 的端点。

应用程序中的配置文件 **OSGI-INF/blueprint/blueprint.xml** 可能类似以下。请注意，它会添加 JAX-RS **customerservice** 端点，该端点特定于您的应用。

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:jaxrs="http://cxf.apache.org/blueprint/jaxrs"
           xsi:schemaLocation="
  http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
  http://cxf.apache.org/blueprint/jaxrs http://cxf.apache.org/schemas/blueprint/jaxrs.xsd">
```

```
    <!-- JAXRS Application -->
    <bean id="customerBean" class="org.keycloak.example.rs.CxfCustomerService" />

    <jaxrs:server id="cxfJaxrsServer" address="/customerservice">
        <jaxrs:providers>
            <bean class="com.fasterxml.jackson.jaxrs.json.JacksonJsonProvider" />
        </jaxrs:providers>
        <jaxrs:serviceBeans>
            <ref component-id="customerBean" />
        </jaxrs:serviceBeans>
    </jaxrs:server>
</blueprint>
```

此外，您必须创建 **${karaf.etc}/org.ops4j.pax.web.context-*anyName*.cfg** 文件。它将被视为由 **pax-web-runtime** 捆绑包跟踪的工厂 PID 配置。这种配置可能包含与标准 **web.xml** 的一些属性对应的以下属性：

```
bundle.symbolicName = org.apache.cxf.cxf-rt-transports-http
context.id = default

context.param.keycloak.config.resolver =
org.keycloak.adapters.osgi.HierarchicalPathBasedKeycloakConfigResolver

login.config.authMethod = KEYCLOAK

security.cxf.url = /cxf/customerservice/*
security.cxf.roles = admin, user
```

有关配置管理文件中可用属性的完整描述，请参阅 Fuse 文档。以上属性的含义如下：

bundle.symbolicName and **context.id**

在 **org.ops4j.pax.web.service.WebContainer** 中识别捆绑包及其部署上下文。

context.param.keycloak.config.resolver

为捆绑包提供 **keycloak.config.resolver** 上下文参数的值，与 **web.xml** 中的 classic WARs 相同。在 Configuration Resolvers 部分中描述了可用的解析器。

login.config.authMethod

身份验证方法.必须是 **KEYCLOAK**。

security.*anyName*.url 和 **security.*anyName*.roles**

单个安全约束的属性值，就像在 **web.xml** 中的 **security-constraint/web-resource-collection/url-pattern** 和 **security-constraint/auth-constraint/role-name** 中设置一样。角色通过逗号和空格分开。***anyName*** 标识符可以是任意的，但必须为同一安全约束的独立属性匹配。

注意

有些 Fuse 版本包含一个程序漏洞，它要求角色用 **", "** (comma 和 single space)分开。确保您使用精确使用此表示法来分隔角色。

META-INF/MANIFEST.MF 中的 **Import-Package** 必须至少包含这些导入：

```
javax.ws.rs;version="[2,3)",
META-INF.cxf;version="[2.7,3.3)",
```

```
META-INF.cxf.osgi;version="[2.7,3.3)";resolution:=optional,
org.apache.cxf.transport.http;version="[2.7,3.3)",
org.apache.cxf.*;version="[2.7,3.3)",
com.fasterxml.jackson.jaxrs.json;version="${jackson.version}"
```

2.1.7.9. 保护 Fuse 管理服务

2.1.7.9.1. 使用 SSH 身份验证到 Fuse Terminal

Red Hat Single Sign-On 主要解决 Web 应用程序身份验证的用例；但是，如果您的其他 Web 服务和应用程序通过红帽单点登录保护，则保护非 Web 管理服务（如使用红帽单点登录凭证的 SSH）是最佳选择。您可以使用 JAAS 登录模块进行此操作，该模块允许远程连接到 Red Hat Single Sign-On，并根据 Resource Owner Password 凭据验证凭证。

要启用 SSH 身份验证，请执行以下步骤。

流程

1. 在 Red Hat Single Sign-On 中创建一个客户端（例如，**ssh-jmx-admin-client**），它将用于 SSH 身份验证。此客户端需要选择 **Direct Access Grants Enabled** to **On**。
2. 在 **$FUSE_HOME/etc/org.apache.karaf.shell.cfg** 文件中，更新或指定此属性：

   ```
   sshRealm=keycloak
   ```

3. 添加 **$FUSE_HOME/etc/keycloak-direct-access.json** 文件，其内容类似于以下内容（基于您的环境和 Red Hat Single Sign-On 客户端设置）：

   ```
   {
       "realm": "demo",
       "resource": "ssh-jmx-admin-client",
       "ssl-required" : "external",
       "auth-server-url" : "http://localhost:8080/auth",
       "credentials": {
           "secret": "password"
       }
   }
   ```

   此文件指定客户端应用程序配置，供 **keycloak** JAAS 域中的 JAAS DirectAccessGrantsLoginModule 用于 SSH 身份验证。

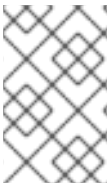
4. 启动 Fuse 并安装 **keycloak** JAAS 域。最简单的方法是安装 **keycloak-jaas** 功能，该功能已预定义了 JAAS 域。您可以使用自己的 **keycloak** JAAS 域覆盖功能的预定义域。详情请查看 JBoss Fuse 文档。
   在 Fuse 终端中使用这些命令：

   ```
   features:addurl mvn:org.keycloak/keycloak-osgi-features/18.0.14.redhat-00001/xml/features
   features:install keycloak-jaas
   ```

5. 在终端中输入以下内容，以 **admin** 用户身份使用 SSH 登录：

   ```
   ssh -o PubkeyAuthentication=no -p 8101 admin@localhost
   ```

6. 使用密码 **password** 登录。



注意

在稍后的一些操作系统中，您可能还需要使用 SSH 命令的 -o 选项 **-o HostKeyAlgorithms=+ssh-dss** =+ssh-dss，因为之后的 SSH 客户端不允许使用 **ssh-dss** 算法。但是，默认情况下，它目前在 JBoss Fuse 7.4.0 中使用。

请注意，用户需要具有 realm 角色 **admin** 来执行所有操作或其他角色才能执行操作子集（例如，**viewer** 角色来限制用户仅运行只读 Karaf 命令）。可用的角色在 **$FUSE_HOME/etc/org.apache.karaf.shell.cfg** 或 **$FUSE_HOME/etc/system.properties** 中配置。

2.1.7.9.2. 使用 JMX 身份验证

如果要使用 jconsole 或其他外部工具通过 RMI 远程连接到 JMX，则可能需要使用 JMX 身份验证。否则，最好使用 hawt.io/jolokia，因为 jolokia 代理默认安装在 hawt.io 中。如需了解更多详细信息，请参阅 Hawtio 管理控制台。

要使用 JMX 身份验证，请执行以下步骤。

流程

1. 在 **$FUSE_HOME/etc/org.apache.karaf.management.cfg** 文件中，将 jmxRealm 属性更改为：

```
jmxRealm=keycloak
```

2. 安装 **keycloak-jaas** 功能并配置 **$FUSE_HOME/etc/keycloak-direct-access.json** 文件，如上面的 SSH 部分所述。

3. 在 jconsole 中，您可以使用 URL，例如：

```
service:jmx:rmi://localhost:44444/jndi/rmi://localhost:1099/karaf-root
```

和凭证：admin/password （基于您环境具有管理员特权的用户）。

2.1.7.10. 保护 Hawtio 管理控制台

要使用红帽单点登录来保护 Hawtio 管理控制台，请执行以下步骤。

流程

1. 在域的 Red Hat Single Sign-On 管理控制台中创建客户端。例如，在 Red Hat Single Sign-On 演示 域中，创建一个客户端 **hawtio-client**，将 **public** 指定为 Access Type，并指定指向 Hawtio: http://localhost:8181/hawtio/* 的重定向 URI。配置对应的 Web Origin （本例中为 http://localhost:8181）。设置客户端范围映射，以在 **hawtio-client** 客户端详情的 *Scope* 选项卡中包括 *view-profile* 客户端角色。

2. *使用类似以下示例的内容，在 **$FUSE_HOME/etc** 目录中创建 **keycloak-hawtio-client.json** 文件。根据您的 Red Hat Single Sign-On 环境，更改 **realm**、**resource** 和 **auth-server-url** 属性。**resource** 属性必须指向上一步中创建的客户端。此文件供客户端(Hawtio JavaScript 应用)侧使用。*

```
{
  "realm" : "demo",
```

```
  "clientId" : "hawtio-client",
  "url" : "http://localhost:8080/auth",
  "ssl-required" : "external",
  "public-client" : true
}
```

3. 使用类似以下示例的内容，在 **$FUSE_HOME/etc** 目录中创建 **keycloak-direct-access.json** 文件。根据您的红帽单点登录环境更改 **realm** 和 **url** 属性。此文件供 JavaScript 客户端使用。

```
{
  "realm" : "demo",
  "resource" : "ssh-jmx-admin-client",
  "auth-server-url" : "http://localhost:8080/auth",
  "ssl-required" : "external",
  "credentials": {
    "secret": "password"
  }
}
```

4. 使用类似以下示例的内容，在 **$FUSE_HOME/etc** dicrectory 中创建 **keycloak-hawtio.json** 文件。根据您的 Red Hat Single Sign-On 环境更改 **realm** 和 **auth-server-url** 属性。此文件供服务器(JAAS Login 模块)侧使用。

```
{
  "realm" : "demo",
  "resource" : "jaas",
  "bearer-only" : true,
  "auth-server-url" : "http://localhost:8080/auth",
  "ssl-required" : "external",
  "use-resource-role-mappings": false,
  "principal-attribute": "preferred_username"
}
```

5. 启动 JBoss Fuse 7.4.0，安装 Keycloak 功能。然后，在 Karaf 终端中输入：

```
system:property -p hawtio.keycloakEnabled true
system:property -p hawtio.realm keycloak
system:property -p hawtio.keycloakClientConfig file:/\${karaf.base}/etc/keycloak-hawtio-client.json
system:property -p hawtio.rolePrincipalClasses org.keycloak.adapters.jaas.RolePrincipal,org.apache.karaf.jaas.boot.principal.RolePrincipal
restart io.hawt.hawtio-war
```

6. 进入 http://localhost:8181/hawtio，以用户从 Red Hat Single Sign-On 域登录。
请注意，用户需要具有正确的 realm 角色才能成功向 Hawtio 进行身份验证。可用的角色在 **hawtio.roles** 中的 **$FUSE_HOME/etc/system.properties** 文件中配置。

2.1.8. Spring Boot adapter

注意

Spring Boot Adapter 已被弃用，它不包括在 8.0 及更高版本和更高版本的 RH-SSO 版本中。此适配器将在 RH-SSO 7.x 生命周期中维护。用户应迁移到 Spring Security，以便将其 Spring Boot 应用程序与 RH-SSO 集成。

2.1.8.1. 安装 Spring Boot 适配器

为了保护 Spring Boot 应用程序的安全性，您必须将 Keycloak Spring Boot adapter JAR 添加到应用程序中。然后，您必须通过正常的 Spring Boot 配置(**application.properties**)提供一些额外的配置。

Keycloak Spring Boot 适配器利用 Spring Boot 的自动配置，因此所有您需要做的都是将这个适配器 Keycloak Spring Boot starter 添加到项目中。

流程

1. 要使用 Maven 将初学者添加到项目中，请添加到您的依赖项中：

```
<dependency>
  <groupId>org.keycloak</groupId>
  <artifactId>keycloak-spring-boot-starter</artifactId>
</dependency>
```

2. 添加 Adapter BOM 依赖项：

```
<dependencyManagement>
 <dependencies>
  <dependency>
   <groupId>org.keycloak.bom</groupId>
   <artifactId>keycloak-adapter-bom</artifactId>
   <version>18.0.14.redhat-00001</version>
   <type>pom</type>
   <scope>import</scope>
  </dependency>
 </dependencies>
</dependencyManagement>
```

目前，以下嵌入式容器被支持，在使用 Starter 时不需要任何额外的依赖项：

- tomcat
- Undertow
- Jetty

2.1.8.2. 配置 Spring Boot Adapter

使用流程将 Spring Boot 应用程序配置为使用 Red Hat Single Sign-On。

流程

1. 您不是 **keycloak.json** 文件，而是通过正常的 Spring Boot 配置为 Spring Boot 适配器配置域。例如：

```
keycloak.realm = demorealm
keycloak.auth-server-url = http://127.0.0.1:8080/auth
keycloak.ssl-required = external
keycloak.resource = demoapp
keycloak.credentials.secret = 11111111-1111-1111-1111-111111111111
keycloak.use-resource-role-mappings = true
```

您可以通过设置 **keycloak.enabled = false** 来禁用 Keycloak Spring Boot Adapter（例如在测试中）。

2. 要配置 Policy Enforcer，与 keycloak.json 不同，请使用 **policy-enforcer-config** 而不是只是 **policy-enforcer**。

3. 指定通常在 **web.xml** 中的 Jakarta EE 安全配置。
Spring Boot Adapter 将 **login-method** 设置为 **KEYCLOAK**，并在启动时配置 **security-constraints**。以下是配置示例：

```
keycloak.securityConstraints[0].authRoles[0] = admin
keycloak.securityConstraints[0].authRoles[1] = user
keycloak.securityConstraints[0].securityCollections[0].name = insecure stuff
keycloak.securityConstraints[0].securityCollections[0].patterns[0] = /insecure

keycloak.securityConstraints[1].authRoles[0] = admin
keycloak.securityConstraints[1].securityCollections[0].name = admin stuff
keycloak.securityConstraints[1].securityCollections[0].patterns[0] = /admin
```

警告

如果您计划将 Spring 应用程序部署为 WAR，则不应使用 Spring Boot Adapter，并将专用适配器用于应用服务器或 servlet 容器。您的 Spring Boot 还应包含 **web.xml** 文件。

2.1.9. Java servlet 过滤器适配器

如果要在没有选择使用 servlet 过滤器适配器的平台上部署 Java Servlet 应用程序。这个适配器的工作方式与其它适配器不同。您不会在 web.xml 中定义安全限制。反之，您可以使用 Red Hat Single Sign-On servlet 过滤器适配器定义一个过滤器映射来保护您要保护的 url 模式。

警告

Backchannel logout 与标准适配器不同。它将会话 ID 标记为已注销，而不是无效的 HTTP 会话。无法基于会话 ID 使 HTTP 会话无效。

```
<web-app 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_3_0.xsd"
    version="3.0">

 <module-name>application</module-name>

   <filter>
```

```
        <filter-name>Keycloak Filter</filter-name>
        <filter-class>org.keycloak.adapters.servlet.KeycloakOIDCFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>Keycloak Filter</filter-name>
        <url-pattern>/keycloak/*</url-pattern>
        <url-pattern>/protected/*</url-pattern>
    </filter-mapping>
</web-app>
```

在上面的代码段中有两个 url-patterns。/ protected/* 是我们需要保护的文件，而 /keycloak/* url-pattern 处理来自 Red Hat Single Sign-On 服务器的回调。

如果您需要排除配置的 **url-patterns** 下的一些路径，您可以使用 Filter init-param **keycloak.config.skipPattern** 配置正则表达式，该正则表达式描述了 keycloak 过滤器应立即委派给 filter-chain。默认情况下不配置 skipPattern。

在没有 **context-path** 的情况下，与 **requestURI** 匹配模式。假定上下文路径 **/myapp** 是一个对 **/myapp/index.html** 的请求，它会根据跳过特征匹配 **/index.html**。

```
<init-param>
    <param-name>keycloak.config.skipPattern</param-name>
    <param-value>^/(path1|path2|path3).*</param-value>
</init-param>
```

请注意，您应该在 Red Hat Single Sign-On Admin 控制台中配置您的客户端，其 Admin URL 指向过滤器的 url-pattern 所涵盖的安全部分。

Admin URL 将回调到 Admin URL，以便执行（如 backchannel logout）的操作。因此，本例中的 Admin URL 应为 **http[s]://hostname/{context-root}/keycloak**。

如果您需要自定义会话 ID 映射器，您可以在 Filter init-param keycloak.config.idMapper 中配置类的完全限定名称。会话 ID 映射器是一个映射用户 ID 和会话 ID 的映射程序。默认配置了 org.keycloak.adapters.spi.InMemorySessionIdMapper。

```
<init-param>
    <param-name>keycloak.config.idMapper</param-name>
    <param-value>org.keycloak.adapters.spi.InMemorySessionIdMapper</param-value>
</init-param>
```

Red Hat Single Sign-On 过滤器具有与其他适配器相同的配置参数，但您必须将它们定义为 filter init 参数，而不是上下文参数。

要使用此过滤器，请在 WAR poms 中包括这个 maven 工件：

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-servlet-filter-adapter</artifactId>
    <version>18.0.14.redhat-00001</version>
</dependency>
```

2.1.10. 安全上下文

如果您需要直接访问令牌，可以使用 **KeycloakSecurityContext** 接口。如果您要从令牌（如用户配置集信息）检索其他详细信息，或者您想要调用受 Red Hat Single Sign-On 保护的 RESTful 服务，这可能很有用。

在 servlet 环境中，它作为 HttpServletRequest 中的属性在安全调用中可用：

```
httpServletRequest
    .getAttribute(KeycloakSecurityContext.class.getName());
```

或者，它位于 HttpSession 中不安全的请求：

```
httpServletRequest.getSession()
    .getAttribute(KeycloakSecurityContext.class.getName());
```

2.1.11. 错误处理

Red Hat Single Sign-On 对基于 servlet 的客户端适配器有一些错误处理功能。在身份验证过程中遇到错误时，红帽 Single Sign-On 将调用 **HttpServletResponse.sendError（）**。您可以在 **web.xml** 文件中设置一个错误页面来处理错误。红帽单点登录可以抛出 400、401、403 和 500 错误。

```
<error-page>
    <error-code>403</error-code>
    <location>/ErrorHandler</location>
</error-page>
```

Red Hat Single Sign-On 还设置可检索的 **HttpServletRequest** 属性。属性名称是 **org.keycloak.adapters.spi.AuthenticationError**，它应该被转换为 **org.keycloak.adapters.OIDCAuthenticationError**。

例如：

```
import org.keycloak.adapters.OIDCAuthenticationError;
import org.keycloak.adapters.OIDCAuthenticationError.Reason;
...

OIDCAuthenticationError error = (OIDCAuthenticationError) httpServletRequest
    .getAttribute('org.keycloak.adapters.spi.AuthenticationError');

Reason reason = error.getReason();
System.out.println(reason.name());
```

2.1.12. 退出

您可以通过多种方式从 Web 应用程序注销。对于 Jakarta EE servlet 容器，您可以调用 **HttpServletRequest.logout（）**。对于其他浏览器应用，您可以将浏览器重定向到 **http://auth-server/auth/realms/{realm-name}/protocol/openid-connect/logout**，如果用户使用其浏览器具有 SSO 会话，则将用户注销。当用户确认注销后，会进行实际注销。您可以选择包含 **id_token_hint**、**post_logout_redirect_uri**、**client_id** 等参数，如 OpenID Connect RP-Initiated Logout 中所述。因此，如果您在包含 **id_token_hint** 参数时，用户不需要显式确认该注销。注销后，只要提供用户，用户会自动重定向到指定的 **post_logout_redirect_uri**。请注意，在包含 **post_logout_redirect_uri** 时，您需要包含 **client_id** 或 **id_token_hint** 参数。

如果您想避免在注销过程中注销外部身份提供程序，您可以提供 **启动_idp 的参数**，并成为身份提供程序的身份（别名）问题。当作为外部身份提供程序发起的单个注销端点的一部分调用时，此参数很有用。

initiated **ing_idp** 是 Red Hat Single Sign-On logout 端点的支持参数，除了 RP-Initiated Logout 规范中描述的参数之外。

当使用 **HttpServletRequest.logout（）** 选项时，适配器对通过刷新令牌的红帽单点登录服务器执行后端 POST 调用。如果从未保护的页面（没有检查有效令牌的页面）执行方法，则刷新令牌将不可用，在这种情况下，适配器会跳过调用。因此，建议使用一个受保护的页面来执行 **HttpServletRequest.logout（）**，以便在需要时执行当前的令牌，并与红帽单点登录服务器交互。

2.1.13. 参数转发

Red Hat Single Sign-On 初始授权端点请求支持不同的参数。大多数参数都在 OIDC 规格 中所述。某些参数由基于适配器配置自动添加。但是，每个调用时还可添加一些参数。当您打开受保护的应用程序 URI 时，特定的参数将转发到 Red Hat Single Sign-On 授权端点。

例如，如果请求离线令牌，您可以使用 **scope** 参数打开受保护的应用程序 URI：

```
http://myappserver/mysecuredapp?scope=offline_access
```

参数 **scope=offline_access** 将自动转发到 Red Hat Single Sign-On 授权端点。

支持的参数有：

- 范围 - 使用以空格分隔的范围列表。空格分隔的列表通常是指特定 客户端上定义的客户端范围。请注意，范围 **openid** 将始终添加到适配器的范围列表中。例如，如果您输入范围选项 **地址电话**，则到 Red Hat Single Sign-On 的请求将包含 scope 参数 **scope=openid 地址电话**。
- 提示 - Red Hat Single Sign-On 支持这些设置：
  - **login** - SSO 将会被忽略，并且始终会显示红帽单点登录登录页面，即使用户已经通过身份验证
  - **consent** - 只适用于带有 **Consent Required** 的客户端。如果使用它，则 Consent 页面将始终显示，即使用户之前授予了这个客户端的同意。
  - **none** - 不显示登录页面；相反，如果用户还没有通过身份验证，该用户将被重定向到应用。此设置允许您在应用程序一侧创建 filter/interceptor，并向用户显示自定义错误页面。请参阅规格中的更多详情。
- max_age - 仅在用户已通过身份验证时才使用。指定验证可保留的最大允许时间，从用户验证身份衡量。如果用户进行身份验证的时间超过 **maxAge**，则 SSO 将被忽略，必须重新验证。
- login_hint - 用来预先填充登录表单上的用户名/电子邮件字段。
- kc_idp_hint - 用来告知 Red Hat Single Sign-On 跳过显示登录页面并自动重定向到指定的身份提供程序。身份提供程序文档中的更多信息。

大多数参数都包括在 OIDC 规格。唯一的例外是参数 **kc_idp_hint**，它特定于 Red Hat Single Sign-On，并包含要使用的身份提供程序的名称。如需更多信息，请参阅《服务器管理指南》 中的 **Identity Brokering** 章节。

警告

如果您使用附加参数打开 URL，如果应用程序中已经通过身份验证，则适配器不会重定向到 Red Hat Single Sign-On。例如，如果您已向应用程序 **mysecuredapp** 进行了身份验证，打开 http://myappserver/mysecuredapp?prompt=login 不会自动重定向到 Red Hat Single Sign-On 登录页面。以后可能会更改此行为。

2.1.14. 客户端身份验证

当机密 OIDC 客户端需要发送后台请求（例如，交换令牌的代码，或刷新令牌）时，它需要与红帽单点登录服务器进行身份验证。默认情况下，可以通过三种方式验证客户端：客户端 ID 和客户端 secret，使用签名 JWT 进行客户端身份验证，或者使用客户端 secret 使用签名 JWT 进行客户端身份验证。

2.1.14.1. 客户端 ID 和客户端 Secret

这是 OAuth2 规范中描述的传统方法。客户端有一个 secret，需要知道适配器（应用程序）和 Red Hat Single Sign-On 服务器。您可以在 Red Hat Single Sign-On Admin Console 中为特定客户端生成 secret，然后将此 secret 粘贴到应用程序端的 **keycloak.json** 文件中：

```
"credentials": {
    "secret": "19666a4f-32dd-4049-b082-684c74115f28"
}
```

2.1.14.2. 使用签名 JWT 进行客户端身份验证

这基于 RFC7523 规格。它以以下方式工作：

- 客户端必须具有私钥和证书。对于 Red Hat Single Sign-On，可以通过传统的 **密钥存储文件** 提供，该文件可在客户端应用的类路径或文件系统中某一位置提供。
- 客户端应用程序启动后，它允许使用 URL（如 http://myhost.com/myapp/k_jwks）以 JWKS 格式下载其公钥，假设 http://myhost.com/myapp 是客户端应用程序的基本 URL。红帽单点登录可以使用此 URL（请参阅以下）。
- 在身份验证过程中，客户端会生成 JWT 令牌，并使用其私钥对其进行签名，并将其发送到特定的后端请求（如 **client_assertion** 参数中的代码到令牌请求）。
- Red Hat Single Sign-On 必须具有客户端的公钥或证书，以便它能够在 JWT 上验证签名。在 Red Hat Single Sign-On 中，您需要为您的客户端配置客户端凭证。首先，您需要在管理控制台中的 **凭据** 中选择 **Signed JWT** 作为对客户端进行身份验证的方法。然后，您可以选择在 **键** 选项卡中选择：
  - 配置红帽单点登录可下载客户端公钥的 JWKS URL。这可以是一个 URL，如 http://myhost.com/myapp/k_jwks（请参阅上面的详细信息）。这个选项是最灵活的，因为客户端可以随时轮转其密钥，然后在需要时始终下载新密钥，而无需更改配置。更准确地，红帽单点登录会在看到由未知 **kid**（密钥 ID）签名的令牌时下载新密钥。
  - 以 PEM 格式、采用 JWK 格式或从密钥存储上传客户端的公钥或证书。使用此选项时，公钥是硬编码的，在客户端生成新密钥对时必须更改。如果没有可用的红帽单点登录管理控制台，您也可以从红帽单点登录管理控制台生成自己的密钥存储。有关如何设置红帽单点登录管理控制台的详情，请查看 服务器管理指南。

要在适配器中设置，您需要在 **keycloak.json** 文件中具有类似如下的内容：

```
"credentials": {
  "jwt": {
    "client-keystore-file": "classpath:keystore-client.jks",
    "client-keystore-type": "JKS",
    "client-keystore-password": "storepass",
    "client-key-password": "keypass",
    "client-key-alias": "clientkey",
    "algorithm": "RS256",
    "token-expiration": 10
  }
}
```

使用这个配置，密钥存储文件 **keystore-client.jks** 必须在您的 WAR 中的 classpath 上可用。如果不使用前缀 **classpath：** 您可以指向运行客户端应用程序的文件系统中的任何文件。

algorithm 字段指定用于签名 JWT 的算法，默认为 **RS256**。此字段应当与密钥对同步。例如，**RS256** 算法需要 RSA 密钥对，而 **ES256** 算法需要 EC 密钥对。如需更多信息，请参阅 Cryptographic Algorithms 用于数字签名和 MAC。

2.1.15. 多租户

在我们的环境中，多元意味着可以使用多个红帽单点登录域保护单一目标应用程序(WAR)。域可以位于同一红帽单点登录实例或不同的实例上。

实际上，这意味着应用程序需要有多个 **keycloak.json** 适配器配置文件。

您可以有多个 WAR 实例，它们不同的适配器配置文件部署到不同的上下文路径。但是，这可能很不便，您可能还希望根据上下文路径以外的其他内容选择域。

Red Hat Single Sign-On 使可以有一个自定义配置解析器，以便您可以选择每个请求的适配器配置。

要实现这一点，首先需要创建 **org.keycloak.adapters.KeycloakConfigResolver** 的实现。例如：

```
package example;

import org.keycloak.adapters.KeycloakConfigResolver;
import org.keycloak.adapters.KeycloakDeployment;
import org.keycloak.adapters.KeycloakDeploymentBuilder;

public class PathBasedKeycloakConfigResolver implements KeycloakConfigResolver {

  @Override
  public KeycloakDeployment resolve(OIDCHttpFacade.Request request) {
    if (path.startsWith("alternative")) {
      KeycloakDeployment deployment = cache.get(realm);
      if (null == deployment) {
        InputStream is = getClass().getResourceAsStream("/tenant1-keycloak.json");
        return KeycloakDeploymentBuilder.build(is);
      }
    } else {
      InputStream is = getClass().getResourceAsStream("/default-keycloak.json");
      return KeycloakDeploymentBuilder.build(is);
    }
```

```
    }

}
```

您还需要配置用于 **web.xml** 中的 **keycloak.config.resolver** 上下文-param 的 **KeycloakConfigResolver** 实现：

```
<web-app>
    ...
    <context-param>
        <param-name>keycloak.config.resolver</param-name>
        <param-value>example.PathBasedKeycloakConfigResolver</param-value>
    </context-param>
</web-app>
```

2.1.16. 应用程序集群

本章与支持部署到 JBoss EAP 的集群应用程序相关。

根据您的应用程序是可用的一些选项：

- 无状态或有状态
- 可分发（复制的 http 会话）或不可分发的
- 依赖负载均衡器提供的粘性会话
- 托管在与 Red Hat Single Sign-On 相同的域中

处理集群的过程就像常规的应用程序一样简单。主要的原因是浏览器和服务器端应用程序向红帽单点登录发送请求，因此不像在您的负载均衡器上启用粘性会话那样简单。

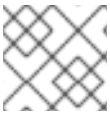
2.1.16.1. 无状态令牌存储

默认情况下，Red Hat Single Sign-On 保护的 Web 应用程序使用 HTTP 会话来存储安全上下文。这意味着您必须启用粘性会话或复制 HTTP 会话。

作为在 HTTP 会话中存储安全上下文的替代方法，可将适配器配置为将其存储在 cookie 中。如果您要使应用程序无状态，或者您不想将安全上下文存储在 HTTP 会话中，这很有用。

要使用 Cookie 存储来保存安全上下文，请编辑应用程序 **WEB-INF/keycloak.json** 并添加：

```
"token-store": "cookie"
```



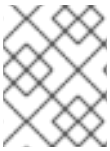
注意

token-store 的默认值为 **session**，它将安全上下文存储在 HTTP 会话中。

使用 cookie 存储的一个限制是，整个安全性上下文将针对每个 HTTP 请求在 cookie 中传递。这可能会影响性能。

另一个较小的限制是，对单点登录的支持有限。如果您从应用程序本身发起 servlet logout(HttpServletRequest.logout)，它没有问题，因为适配器将删除 KEYCLOAK_ADAPTER_STATE cookie。但是，从不同应用程序初始化的频道注销不会由 Red Hat Single Sign-On 传播到使用 cookie 存储

的应用程序。因此，建议对访问令牌超时使用简短值（例如1 分钟）。



注意

有些负载均衡器不允许配置粘性会话 Cookie 名称或内容，如 Amazon ALB。对于这些，建议将 **shouldAttachRoute** 选项设置为 **false**。

2.1.16.2. 相对 URI 优化

在部署情况下，Red Hat Single Sign-On 和应用程序托管在同一域中（通过反向代理或负载均衡器），它可以方便地在客户端配置中使用相对 URI 选项。

对于相对 URI，则根据用于访问 Red Hat Single Sign-On 的 URL 解析 URI。

例如，如果您的应用程序的 URL 是 **https://acme.org/myapp**，Red Hat Single Sign-On 的 URL 为 **https://acme.org/auth**，您可以使用 redirect-uri **/myapp** 而不是 **https://acme.org/myapp**。

2.1.16.3. 管理 URL 配置

可以在红帽单点登录管理控制台中配置特定客户端的管理 URL。红帽单点登录服务器用于向应用程序发送后端请求以获取各种任务，如注销用户或推送撤销策略。

例如，后端频道注销的工作方式为：

1. 用户从一个应用程序发送注销请求
2. 应用程序向 Red Hat Single Sign-On 发送注销请求
3. Red Hat Single Sign-On 服务器使用户会话无效
4. 然后，Red Hat Single Sign-On 服务器会向应用程序发送一个与会话关联的 admin url 的后方请求
5. 当应用程序收到注销请求时，它会使对应的 HTTP 会话无效

如果 admin URL 包含 **${application.session.host}**，它将被替换为与 HTTP 会话关联的节点的 URL。

2.1.16.4. 注册应用程序节点

上一节中介绍了 Red Hat Single Sign-On 如何发送注销请求到与特定 HTTP 会话关联的节点。然而，在某些情况下，管理员可能希望将管理任务传播到所有注册的集群节点，而不只是其中之一。例如，在策略到应用程序之前将新版不推送到应用程序，或从应用程序注销所有用户。

在这种情况下，Red Hat Single Sign-On 需要了解所有应用程序集群节点，因此可以将事件发送到所有应用程序。要做到这一点，我们支持自动发现机制：

1. 当新的应用程序节点加入集群时，它会向 Red Hat Single Sign-On 服务器发送注册请求
2. 根据配置的定期间隔，请求可能会与 Red Hat Single Sign-On 重新输入
3. 如果 Red Hat Single Sign-On 服务器没有在指定超时内收到重新注册请求，那么它会自动取消注册特定节点
4. 当节点发送未注册或应用程序取消部署时，节点也会在 Red Hat Single Sign-On 中取消注册。如果没有调用非部署监听程序，这可能无法正常工作，这会导致自动取消注册

默认情况下，发送启动注册和定期重新注册功能会被禁用，因为它只是一些集群应用程序所需的。

要启用功能，为您的应用程序编辑 **WEB-INF/keycloak.json** 文件并添加：

```
"register-node-at-startup": true,
"register-node-period": 600,
```

这意味着，适配器会在启动时发送注册请求，并每 10 分钟重新注册一次。

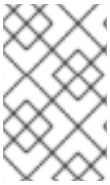
在 Red Hat Single Sign-On Admin Console 中，您可以指定最大节点 重新注册 超时（应该大于适配器配置中注册节点）。您还可以通过 Admin Console 手动添加和删除集群节点，如果您不想依赖自动注册功能，或者想要删除过时的应用程序节点（如果不使用自动取消注册功能），这会很有用。

2.1.16.5. 每个请求中的刷新令牌

默认情况下，应用程序适配器仅在访问令牌过期时刷新访问令牌。但是，您也可以将适配器配置为在每个请求中刷新令牌。这可能会对性能产生影响，因为您的应用程序将向红帽单点登录服务器发送更多请求。

要启用功能，为您的应用程序编辑 **WEB-INF/keycloak.json** 文件并添加：

```
"always-refresh-token": true
```



注意

这可能会对性能有显著影响。只有您无法依赖 backchannel 消息传播注销而不是在策略前启用此功能。需要考虑的另一个因素是，默认情况下访问令牌具有较短的到期时间，因此即使注销不会传播令牌在注销后才会过期。

2.2. JAVASCRIPT 适配器

Red Hat Single Sign-On 附带了一个客户端 JavaScript 库，可用于保护 HTML5/JavaScript 应用程序。JavaScript 适配器对 Cordova 应用程序进行了内置支持。

最好使用 NPM 或 Yarn 等软件包管理器在应用程序中包括 JavaScript 适配器。**keycloak-js** 软件包位于以下位置：

- NPM： https://www.npmjs.com/package/keycloak-js
- YARN ：https://yarnpkg.com/package/keycloak-js

另外，还可直接从位于 **/auth/js/keycloak.js** 的 Red Hat Single Sign-On 服务器检索库，也可以作为 ZIP 存档分发。

需要注意的是使用客户端应用程序，客户端必须是公共客户端，因为无法安全地将客户端凭据存储在客户端应用中。这需要确保您为客户端配置的重定向 URI 正确且尽量具体。

要使用 JavaScript 适配器，您必须首先在 Red Hat Single Sign-On Admin Console 中为您的应用程序创建客户端。确保为 **Access Type** 选择了 **public**。您可以通过切换 OFF 客户端身份验证切换来实现此功能。

您还需要配置 **有效的重定向 URI** 和 **Web Origins**。具体来说，因为无法执行此操作，可能会导致安全漏洞。

创建客户端后，点右上角的 **Action** 选项卡，然后选择 **Download adapter config**。为 **Format** 选项 选择 **Keycloak OIDC JSON**，然后点 **Download**。下载的 **keycloak.json** 文件应当托管在您的 web 服务器上，其位置与您的 HTML 页面相同。

另外，您可以跳过配置文件并手动配置适配器。

以下示例演示了如何初始化 JavaScript 适配器：

```
<html>
<head>
  <script src="keycloak.js"></script>
  <script>
    function initKeycloak() {
      const options = {};
      const keycloak = new Keycloak();
      keycloak.init(options)
        .then(function(authenticated) {
          console.log('keycloak:' + (authenticated ? 'authenticated' : 'not authenticated'));
        }).catch(function(error) {
          for (const property in error) {
            console.error(`${property}: ${error[property]}`);
          }
        });
    }
  </script>
</head>
<body onload="initKeycloak()">
    <!-- your page content goes here -->
</body>
</html>
```

如果 **keycloak.json** 文件位于您指定的不同位置：

```
const keycloak = new Keycloak('http://localhost:8080/myapp/keycloak.json');
```

或者，您可以使用所需配置在 JavaScript 对象中传递：

```
const keycloak = new Keycloak({
    url: 'http://keycloak-server$/auth',
    realm: 'myrealm',
    clientId: 'myapp'
});
```

默认情况下，为了验证您需要调用 **登录功能**。但是，有两个选项可供适配器自动进行身份验证。您可以将 **login-required** 或 **check-sso** 传递给 init 功能。如果用户登录到 Red Hat Single Sign-On，或显示登录页面时，**login-required** 将验证客户端。只有用户尚未登录时，**检查** 关联才会验证客户端。如果该浏览器没有登录，则将重新指向应用并保持未经身份验证的。

您可以配置一个静默的 **check-sso** 选项。启用此功能后，您的浏览器不会完全重定向到 Red Hat Single Sign-On 服务器，再重新指向应用程序，但此操作将在隐藏的 iframe 中执行，因此只有当应用程序初始化并再次从红帽单点登录重新指向您的应用程序后，您的应用程序资源才需要被浏览器被加载和解析。在 SPA(Single Page Applications)时，这特别有用。

要启用 ent **check-sso**，您必须在 init 方法中提供 **silentCheckSsoRedirectUri** 属性。此 URI 需要是应用中的有效端点（当然，在 Red Hat Single Sign-On Admin Console 中，作为客户端的有效重定向）：

```
keycloak.init({
    onLoad: 'check-sso',
    silentCheckSsoRedirectUri: window.location.origin + '/silent-check-sso.html'
})
```

在成功检查身份验证状态并从 Red Hat Single Sign-On 服务器检索令牌后，位于 ilent check-sso redirect uri 的页面会加载到 iframe 中。它没有其他任务，而不是将接收的令牌发送到主应用程序，应该只如下所示：

```
<html>
<body>
    <script>
        parent.postMessage(location.href, location.origin)
    </script>
</body>
</html>
```

请注意，此页面的指定位置必须由应用程序本身提供，不是 JavaScript 适配器的一部分。

警告

在某些现代浏览器中，静默的 **check-sso** 会有一些限制。请参阅带有跟踪保护部分的现代浏览器。

要在 **Load** to **login-required** 中将 **login-required** 设置为 login-required 并传递给 init 方法：

```
keycloak.init({
    onLoad: 'login-required'
})
```

在用户通过身份验证后，通过在 **Authorization** 标头中包含 bearer 令牌，向红帽单点登录保护的 RESTful 服务发出请求。例如：

```
const loadData = function () {
    document.getElementById('username').innerText = keycloak.subject;

    const url = 'http://localhost:8080/restful-service';

    const req = new XMLHttpRequest();
    req.open('GET', url, true);
    req.setRequestHeader('Accept', 'application/json');
    req.setRequestHeader('Authorization', 'Bearer ' + keycloak.token);

    req.onreadystatechange = function () {
        if (req.readyState == 4) {
            if (req.status == 200) {
                alert('Success');
            } else if (req.status == 403) {
                alert('Forbidden');
            }
        }
    }

    req.send();
};
```

要记住的一点是，访问令牌默认具有简短的到期时间，因此您可能需要在发送请求前刷新访问令牌。您可以通过 **updateToken** 方法进行此操作。**updateToken** 方法会返回保证，只有在令牌成功刷新并向用户显示错误时，才会轻松地调用该服务。例如：

```
keycloak.updateToken(30).then(function() {
    loadData();
}).catch(function() {
    alert('Failed to refresh token');
});
```

2.2.1. 会话状态 iframe

默认情况下，JavaScript 适配器会创建一个隐藏 iframe，用于检测是否发生了 Single-Sign Out。这不需要任何网络流量，而是通过查看特殊状态 cookie 来检索的状态。通过在传递到 **init** 方法的选项中设置 **checkLoginIframe: false** 来禁用这个功能。

您不应该依赖于直接查看这个 cookie。其格式可能会改变，它也与 Red Hat Single Sign-On 服务器的 URL 关联，而不是您的应用程序。

警告

在某些现代浏览器中，会话状态 iframe 功能有限。请参阅 带有跟踪保护部分的现代浏览器。

2.2.2. 隐式和混合流程

默认情况下，JavaScript 适配器使用授权代码流。

通过这个流程，Red Hat Single Sign-On 服务器会向应用返回授权代码，而不是身份验证令牌。在浏览器重新定向到应用程序后，JavaScript 适配器会交换访问令牌 **的代码** 和刷新令牌。

Red Hat Single Sign-On 还支持 Implicit 流，在使用红帽单点登录验证成功后立即发送访问令牌。这可能比标准流性能更好，因为没有额外的请求交换令牌代码，但它在访问令牌过期时会产生影响。

但是，在 URL 片段中发送访问令牌可以是安全漏洞。例如，令牌可以通过 Web 服务器日志和浏览器历史记录泄露。

要启用隐式流，您需要在 Red Hat Single Sign-On Admin Console 中为客户端启用 **Implicit Flow Enabled** 标志。您还需要传递带有值 **隐式** 到 **init** 方法的参数 **流** ：

```
keycloak.init({
    flow: 'implicit'
})
```

需要注意的一点是，只会提供访问令牌，且没有刷新令牌。这意味着，当访问令牌过期后，应用程序必须再次重定向到 Red Hat Single Sign-On，以获取新的访问令牌。

Red Hat Single Sign-On 还支持 混合 流程。

这要求客户端同时在管理控制台中启用 **Standard Flow** 和 **Implicit Flow Enabled** 标记。然后，Red Hat Single Sign-On 服务器会将代码和令牌发送到您的应用。访问令牌可以立即使用，而代码可以被交换用于访问和刷新令牌。与隐式流类似，混合流适合性能，因为访问令牌可以立即可用。但是，令牌仍然在 URL 中发送，前面提到的安全漏洞可能仍然适用。

混合流程的一个优点是，刷新令牌可供应用程序使用。

对于混合流程，您需要将参数 **流** 与值 **hybrid** 传递给 **init** 方法：

```
keycloak.init({
    flow: 'hybrid'
})
```

2.2.3. 带有 Cordova 的混合应用程序

Keycloak 支持通过 Apache Cordova 开发的混合移动应用程序。JavaScript 适配器具有两种模式：**cordova** 和 **cordova-native**。

默认为 cordova，如果还没有配置适配器类型且存在 window.cordova，则适配器会自动选择。登录时，它会打开一个 InApp Browser，允许用户与红帽单点登录互动，然后再通过重定向到 **http://localhost** 来返回到应用程序。因此，您必须在管理控制台的客户端配置部分中将这个 URL 列列为有效的 redirect-uri。

虽然此模式易于设置，但它也有一些缺点：

- InApp-Browser 是嵌入在应用程序中且不是电话的默认浏览器的浏览器。因此，它将有不同的设置，存储的凭据将不可用。
- InApp-Browser 可能也较慢，特别是渲染更复杂的它们时。
- 在使用此模式前需要考虑安全性问题，例如，应用程序可以访问用户凭证，因为它可以完全控制浏览器渲染登录页面，因此不允许其在应用中使用它。

使用此示例应用程序帮助您入门：
https://github.com/keycloak/keycloak/tree/master/examples/cordova

另一种模式 **cordova-native** 采用不同的方法。它使用系统的浏览器打开登录页面。在用户通过身份验证后，浏览器会使用特殊的 URL 重新指向应用程序。在那里，Red Hat Single Sign-On 适配器可以通过从 URL 读取代码或令牌来完成登录。

您可以通过将适配器类型 **cordova-native** 传递给 **init** 方法来激活原生模式：

```
keycloak.init({
    adapter: 'cordova-native'
})
```

这个适配器需要两个附加插件：

- Cordova-plugin-browsertab ：允许应用程序在系统浏览器中打开 Webpages
- Cordova-plugin-deeplinks ：允许浏览器通过特殊 URL 将重新重定向到您的应用程序

每个平台上链接到应用程序的技术细节各有不同，且需要特殊设置。如需更多详情，请参阅 deeplinks 插件文档中的 Android 和 iOS 部分。

用于打开应用程序的不同链接：自定义方案（如 **myapp://login** 或 **android-app://com.example.myapp/https/example.com/login**）和 Universal Links(iOS)）/ Deep

Links(Android)。虽然前者易于设置并更可靠工作，但之后会提供额外的安全性，因为它们是唯一的，并且只有域的所有者可以注册它们。custom-URLs 在 iOS 中已弃用。我们建议您使用通用链接和回退站点，并在其中使用 custom-url 链接来获得最佳的可靠性。

另外，我们推荐以下步骤提高与 Keycloak 适配器的兼容性：

- iOS 上的 Universal Links 似乎可以在将 **response-mode** 设置为 **query**时更可靠地工作
- 要防止 Android 在重定向时打开应用程序的新实例，请将以下代码片段添加到 **config.xml** 中：

```
<preference name="AndroidLaunchMode" value="singleTask" />
```

一个示例应用程序演示了如何使用 native-mode:
https://github.com/keycloak/keycloak/tree/master/examples/cordova-native

2.2.4. 自定义适配器

有时需要运行默认不支持的 JavaScript 客户端（如 Capacitor）。要在这些未知环境中使用 JavasScript 客户端，可以传递自定义适配器。例如，第三方库可能会提供这样的适配器，以便可以在没有问题的情况下运行 JavaScript 客户端：

```
import Keycloak from 'keycloak-js';
import KeycloakCapacitorAdapter from 'keycloak-capacitor-adapter';

const keycloak = new Keycloak();

keycloak.init({
    adapter: KeycloakCapacitorAdapter,
});
```

这个特定软件包不存在，但它会提供一个很好的示例来把这个适配器传递给客户端。

也可以让自己的适配器实现，因此您必须实现 **KeycloakAdapter** 接口中描述的方法。例如，以下 TypeScript 代码确保正确实施所有方法：

```
import Keycloak, { KeycloakAdapter } from 'keycloak-js';

// Implement the 'KeycloakAdapter' interface so that all required methods are guaranteed to be present.
const MyCustomAdapter: KeycloakAdapter = {
    login(options) {
        // Write your own implementation here.
    }

    // The other methods go here...
};

const keycloak = new Keycloak();

keycloak.init({
    adapter: MyCustomAdapter,
});
```

通过省略类型信息来确保正确实施接口将完全保留给您，那么在一定程度上，您也可以在不进行 TypeScript 的情况下执行此操作。

2.2.5. 早期浏览器

JavaScript 适配器依赖于 Base64（window.btoa 和 window.atob）、HTML5 History API 以及可选的 Promise API。如果您需要支持没有这些可用的浏览器（例如，IE9），则需要添加填充器。

polyfill 库示例：

- Base64 - https://github.com/davidchambers/Base64.js
- HTML5 History - https://github.com/devote/HTML5-History-API
- Promise - https://github.com/stefanpenner/es6-promise

2.2.6. 带有跟踪保护的现代浏览器

应用了各种浏览器的最新版本中的 Cookie 策略，以防止由第三方（如 Chrome 中的 SameSite）跟踪用户，或者完全阻止了第三方 Cookie。预计这些策略会随着时间的推移而变得更严格的限制，并由其他浏览器采用，最终导致浏览器不支持并阻止第三方上下文中的 Cookie。受此问题影响的适配器功能可能会在以后被弃用。

JavaScript 适配器依赖于 Session Status iframe、静默 **check-sso** 以及部分常规（非静默）**check-sso**。这些功能有限，或者完全禁用，具体取决于浏览器对 Cookie 的限制程度。适配器会尝试检测此设置并相应地做出反应。

2.2.6.1. 带有 "SameSite=Lax by Default" Policy 的浏览器

如果在 Red Hat Single Sign-On side 以及应用程序端配置 SSL / TLS 连接，则所有功能均被支持。受影响（例如，从版本 84 开始的 Chrome）。

2.2.6.2. 带有 Blocked 第三方 Cookies 的浏览器

不支持会话状态 iframe，如果 JS 适配器检测到这样的浏览器行为，则会自动禁用。这意味着适配器无法将会话 cookie 用于单点登录检测，且必须完全依赖令牌。这意味着，当用户在另一个窗口中注销时，使用 JavaScript 适配器的应用程序不会注销，直到它尝试刷新 Access Token。因此，建议您将 Access Token Lifespan 设置为相对较短的时间，因此会检测到注销，而不是早于以后的版本。请查看 会话和令牌超时。

不支持静默 **check-sso**，会默认回退到常规（非静默）**check-sso**。通过在传递到 **init** 方法的选项中设置 **silentCheckSsoFallback: false** 来更改这种行为。在这种情况下，如果检测到限制性浏览器行为，**check-sso** 会被完全禁用。

定期 **的检查** 也会受到影响。由于 Session Status iframe 不被支持，因此当适配器被初始化以检查用户的登录状态时，必须进行额外的重定向到红帽单点登录。当 iframe 用于识别用户是否登录时，这与标准行为不同，且仅在登出时执行重定向。

受影响的浏览器是 Safari 从版本 13.1 开始。

2.2.7. JavaScript Adapter 参考

2.2.7.1. Constructor

```
new Keycloak();
new Keycloak('http://localhost/keycloak.json');
new Keycloak({ url: 'http://localhost/auth', realm: 'myrealm', clientId: 'myApp' });
```

2.2.7.2. Properties

已验证

如果用户通过身份验证，则为 **true**，否则为 **false**。

token

base64 编码的令牌，可在请求到服务的 **Authorization** 标头中发送。

tokenParsed

解析的令牌作为一个 JavaScript 对象。

subject

用户 ID。

idToken

base64 编码的 ID 令牌。

idTokenParsed

解析的 id 令牌作为 JavaScript 对象。

realmAccess

与令牌关联的 realm 角色。

resourceAccess

与令牌关联的资源角色。

refreshToken

base64 编码的刷新令牌，可用于检索新令牌。

refreshTokenParsed

被解析的刷新令牌为 JavaScript 对象。

timeSkew

浏览器时间和红帽单点登录服务器之间的时间差差以秒为单位。这个值只是一个估算值，但在确定令牌是否过期时准确。

responseMode

以 init 传递的响应模式（默认值为片段）。

流

在 init 中传递的流程。

adapter

允许您覆盖重定向和其他浏览器相关功能的方式由该库处理。可用选项：

- "默认" - 库使用浏览器 api 进行重定向（这是默认设置）
- "Cordova" - 库将尝试使用 InAppBrowser cordova 插件加载 keycloak 登录/registration 页面（当库在 cordova 生态系统中工作时会自动使用）
- "Cordova-native" - 库尝试使用浏览器使用 BrowserTabs cordova 插件打开登录和注册页面。这需要额外的设置来重定向到应用程序（请参阅 第 2.2.3 节 "带有 Cordova 的混合应用程序"）。
- "custom" - 允许您实现自定义适配器（仅适用于高级用例）

responseType

使用登录请求发送到 Red Hat Single Sign-On 的响应类型。这根据初始化过程中所用的流值来确定，但可通过设置此值来覆盖。

2.2.7.3. Methods

init (options)

名为 初始化适配器。

选项是一个对象，其中：

- useNonce - 添加加密非密码以验证身份验证响应是否与请求匹配（默认为 **true**）。
- onLoad - 指定要对负载执行的操作。支持的值有 **login-required** 或 **check-sso**。
- silentCheckSsoRedirectUri - 在Load 上将 redirect uri for silent authentication check 设置为 'check-sso'。
- silentCheckSsoFallback - 当浏览器不支持 静默 **check-sso** 时（默认为 **true**），启用回到普通的 **check-sso**。
- token - 为令牌设置初始值。
- refreshToken - 为刷新令牌设置初始值。
- idToken - 为 id 令牌设置初始值（仅与令牌或刷新Token 一起）。
- Scope - 将默认 scope 参数设置为 Red Hat Single Sign-On 登录端点。使用以空格分隔的范围列表。它们通常引用 特定客户端中定义的客户端范围。请注意，scope **openid** 始终会添加到适配器的范围列表中。例如，如果您输入范围选项 **地址电话**，则到 Red Hat Single Sign-On 的请求将包含 scope 参数 **scope=openid 地址电话**。请注意，如果 **login （）** 选项明确指定范围，则这里指定的默认范围会被覆盖。
- timeSkew - 为本地时间和 Red Hat Single Sign-On 服务器（仅与令牌或刷新Token）之间设置一个初始值。
- checkLoginIframe - Set to enable/disable monitoring login status（默认为 **true**）。
- checkLoginIframeInterval - 设置用于检查登录状态的时间间隔（默认为 5 秒）。
- responseMode - 在登录请求时设置 OpenID Connect 响应模式发送到 Red Hat Single Sign-On 服务器。有效值为 **query** 或 **fragment**。默认值为 **片段**，这意味着在成功验证后，Red Hat Single Sign-On 会重定向到 JavaScript 应用，并在 URL 片段中添加 OpenID Connect 参数。这通常更安全，建议使用它而不是 **query**。
- 流 - 设置 OpenID Connect 流。有效值为 **标准**、**隐式** 或 **混合**。
- enableLogging - 启用从 Keycloak 到控制台的日志记录信息（默认为 **false**）。
- pkceMethod - 要使用的概念验证交换方法(PKCE)。配置此值可启用 PKCE 机制。可用选项：
  - "S256" - 基于 SHA256 的 PKCE 方法
- messageReceiveTimeout - 为等待 Keycloak 服务器的消息响应设置超时（毫秒）。这用于，例如，在第三方 Cookie 检查期间等待消息。默认值为 10000。

返回初始化完成后可以解决的保证。

login(options)

重定向至登录表单.

选项是一个可选对象，其中：

- redirecturi - 指定在登录后要重定向到的 uri。
- 提示 - 这个参数允许在 Red Hat Single Sign-On 服务器端稍自定义登录流。例如，使用值 登录时，强制显示 **登录屏幕**。如需 **提示** 参数的详细信息和所有可能值，请参阅 参数转发 部分。
- maxAge - 仅在用户已通过身份验证时才使用。指定用户验证发生以来的最大时间。如果用户已通过身份验证的时间超过 **maxAge**，则 SSO 将被忽略，而且需要重新验证。
- loginHint - 用来预先填充登录表单上的用户名/电子邮件字段。
- Scope - 覆盖 **init** 中配置的范围，这个特定登录的不同值。
- idpHint - 用来告诉 Red Hat Single Sign-On 跳过显示登录页面并自动重定向到指定的身份提供程序。身份提供程序文档中的更多信息。
- **acr** - 包含关于授权信息，该声明将在 **声明** 参数内部发送到 Red Hat Single Sign-On 服务器。典型的用法用于步骤验证。使用 **{ values: ["silver", "gold"], essential: true }** 的示例。如需了解更多详细信息，请参阅 OpenID Connect 规格和增强身份验证文档。
- action - 如果 **注册** 值，则将用户重定向到注册页面，如果值为 **UPDATE_PASSWORD**，则用户将重定向到重置密码页面（如果未验证，则首先向登录页面发送用户，并在身份验证后重定向到登录页面），否则重定向到登录页面。
- locale - 设置"ui_locales"查询 param，与 OIDC 1.0 规范的第 3.1.2.1 节 保持一致。
- cordovaOptions - 指定传递给 ordova in-app-browser（如果适用）的参数。选择 **hidden** 和 **location** 不受这些参数的影响。所有可用选项均在 https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-inappbrowser/ 中定义。使用示例 **：{ zoom: "no", hardwareback: "yes" }**;

createLoginUrl(options)

返回 URL to login 表单。

选项是一个可选对象，也支持与功能 **登录** 相同的选项。

logout (options)

重定向至注销.

选项是一个对象，其中：

- redirecturi - 指定在注销后要重定向到的 uri。

createLogoutUrl(options)

返回 URL 以注销用户。

选项是一个对象，其中：

- redirecturi - 指定在注销后要重定向到的 uri。

register (options)

重定向至注册表单。使用选项 action 进行登录的快捷方式 = 'register'

的选项与登录方法相同，但 'action' 设置为 'register'

createRegisterUrl(options)

返回 url to registration 页面。带有选项 action 的 createLoginUrl 的快捷方式 = 'register'

选项与 createLoginUrl 方法相同，但 'action' 设置为 'register'

accountManagement()

重定向到帐户管理控制台。

createAccountUrl(options)

返回帐户管理控制台的 URL。

选项是一个对象，其中：

- redirecturi - 指定在重定向到应用程序时要重定向到的 uri。

hasRealmRole(role)

如果令牌具有给定 realm 角色，则返回 true。

hasResourceRole (role, resource)

如果令牌具有资源的给定角色，则返回 true（如果不使用指定 clientId，资源是可选的）。

loadUserProfile()

加载用户配置文件。

返回可通过配置集解决的保证。

例如：

```
keycloak.loadUserProfile()
    .then(function(profile) {
        alert(JSON.stringify(profile, null, "  "))
    }).catch(function() {
        alert('Failed to load user profile');
    });
```

isTokenExpired(minValidity)

如果令牌在过期前的 minValidity 少于 minValidity 秒，则返回为 true（如果未指定 0 则是可选的）。

updateToken(minValidity)

如果令牌在 minValidity 秒内过期（如果未使用 5，则为minValidity 是可选的），则令牌会被刷新。如果 -1 作为 minValidity 传递，则令牌将被强制刷新。如果启用了会话状态 iframe，还会检查会话状态。

返回通过布尔值解析的指示是否刷新令牌的布尔值。

例如：

```
keycloak.updateToken(5)
```

```
    .then(function(refreshed) {
        if (refreshed) {
            alert('Token was successfully refreshed');
        } else {
            alert('Token is still valid');
        }
    }).catch(function() {
        alert('Failed to refresh the token, or the session has expired');
    });
```

clearToken()

清除身份验证状态，包括令牌。如果应用程序检测到会话已过期，例如更新令牌失败，这很有用。

调用此操作会导致被调用AuthLogout 回调监听程序。

2.2.7.4. 回调事件

适配器支持为特定事件设置回调监听程序。请记住，必须在调用 **init** 功能前设置它们。

例如：

```
keycloak.onAuthSuccess = function() { alert('authenticated'); }
```

可用的事件有：

- **onReady (authenticated)** - 在适配器初始化时调用。
- **onAuthSuccess** - 当用户成功验证时调用。
- **onAuthError** - 如果身份验证过程中出现错误，则调用。
- **onAuthRefreshSuccess** - 刷新令牌时调用。
- **onAuthRefreshError** - 如果尝试刷新令牌时出现错误，则调用。
- **onAuthLogout** - 只有启用了会话状态 iframe 或 Cordova 模式时，才会调用调用（只有在启用了会话状态 iframe 时）。
- **onTokenExpired** - 当访问令牌过期时调用。如果有可用的刷新令牌，可以使用 updateToken 刷新令牌，或者在没有（也就是说，带有隐式流）的情况下，您可以重定向到登录屏幕以获取新的访问令牌。

2.3. NODE.JS ADAPTER

Red Hat Single Sign-On 提供了一个在 Connect 上创建的 Node.js 适配器来保护服务器端 JavaScript 应用 - 目标足以与 Express.js 等框架集成。

要使用 Node.js 适配器，首先必须在 Red Hat Single Sign-On Admin Console 中为您的应用程序创建客户端。适配器支持公共、机密和只读访问类型。哪一种选择取决于用例方案。

创建客户端后，点 **Installation** 选项卡，选择 **Red Hat Single Sign-On OIDC JSON** for **Format option**，然后点 **Download**。下载的 **keycloak.json** 文件应当位于项目的根目录下。

2.3.1. 安装

假设您已经安装了 Node.js，并为您的应用程序创建一个文件夹：

```
mkdir myapp && cd myapp
```

使用 **npm init** 命令为您的应用程序创建 **package.json**。现在，在依赖列表中添加 Red Hat Single Sign-On 连接适配器：

```
"dependencies": {
    "keycloak-connect": "file:keycloak-connect-18.0.7.tgz"
}
```

2.3.2. 使用方法

实**例化 Keycloak 类**

Keycloak 类提供了一个中央点，用于配置并与您的应用程序集成。最简单的创建涉及任何参数。

在项目的根目录中，创建一个名为 **server.js** 的文件，并添加以下代码：

```
const session = require('express-session');
const Keycloak = require('keycloak-connect');

const memoryStore = new session.MemoryStore();
const keycloak = new Keycloak({ store: memoryStore });
```

安装 **express-session** 依赖项：

```
npm install express-session
```

要启动 **server.js** 脚本，请在 **package.json** 的"scripts"部分中添加以下命令：

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start" "node server.js"
},
```

现在，我们可以使用以下命令运行我们的服务器：

```
npm run start
```

默认情况下，这将查找名为 **keycloak.json** 的文件，以及应用程序的主要可执行文件，本例中为 root 文件夹，以初始化特定于密钥的设置，如公钥、域名和各种 URL。

在这种情况下，需要 Keycloak 部署来访问 Keycloak admin 控制台。

请访问链接如何使用 Podman 或 Docker 部署 Keycloak 管理控制台

现在，我们已准备好通过访问 Red Hat Single Sign-On Admin Console → client（左侧侧边栏）- → 选择您的 client → Installation → Format Option → Keycloak OIDC JSON → Download 来获取 **keycloak.json** 文件

将下载的 文件粘贴到我们项目的根目录中。

这个方法的实例化会导致使用所有合理的默认值。另外，也可以提供配置对象，而不是 **keycloak.json** 文件：

```
const kcConfig = {
  clientId: 'myclient',
  bearerOnly: true,
  serverUrl: 'http://localhost:8080/auth',
  realm: 'myrealm',
  realmPublicKey: 'MIIBIjANB...'
};

const keycloak = new Keycloak({ store: memoryStore }, kcConfig);
```

应用程序还可以使用以下方法将用户重定向到首选身份提供程序：

```
const keycloak = new Keycloak({ store: memoryStore, idpHint: myIdP }, kcConfig);
```

配置 web 会话存储

如果要使用 web 会话来管理服务器端身份验证的状态，则需要使用至少一个 **store** 参数初始化 **Keycloak(...)**，并传递 **express-session** 使用的实际会话存储。

```
const session = require('express-session');
const memoryStore = new session.MemoryStore();

// Configure session
app.use(
  session({
    secret: 'mySecret',
    resave: false,
    saveUninitialized: true,
    store: memoryStore,
  })
);

const keycloak = new Keycloak({ store: memoryStore });
```

传递自定义范围值

默认情况下，范围值 **openid** 作为参数传递给 Red Hat Single Sign-On 的登录 URL，但您可以添加额外的自定义值：

```
const keycloak = new Keycloak({ scope: 'offline_access' });
```

2.3.3. 安装中间件

实例化后，将中间件安装到连接功能的应用程序中：

为此，首先需要安装 Express：

```
npm install express
```

然后，我们的项目中需要 Express，如下所示：

```
const express = require('express');
const app = express();
```

并在以下代码中添加一个 Keycloak 中间件，并在 Express 中配置 Keycloak 中间件：

```
app.use( keycloak.middleware() );
```

最后，我们把服务器设置为在端口 3000 上侦听 HTTP 请求，方法是将以下代码添加到 **main.js** 中：

```
app.listen(3000, function () {
  console.log('App listening on port 3000');
});
```

2.3.4. 配置代理

如果应用程序在 代理的代理后面运行，则必须根据代理指南中的表达 来配置终止 SSL 连接 Express。使用不正确的代理配置可能会导致生成的重定向 URI。

配置示例：

```
const app = express();

app.set( 'trust proxy', true );

app.use( keycloak.middleware() );
```

2.3.5. 检查身份验证

要在访问资源之前检查用户进行身份验证，只需使用 **keycloak.checkSso（）**。只有在用户已经登录后，它将才进行身份验证。如果用户没有登录，浏览器将重新重定向到原先请求的 URL，并保持未经身份验证的：

```
app.get( '/check-sso', keycloak.checkSso(), checkSsoHandler );
```

2.3.6. 保护资源

简单验证

要强制在访问资源前必须验证用户，只需使用 **keycloak.protect（）** 的 no-argument 版本：

```
app.get( '/complain', keycloak.protect(), complaintHandler );
```

基于角色的授权

使用当前应用程序的应用程序角色保护资源：

```
app.get( '/special', keycloak.protect('special'), specialHandler );
```

使用不同应用程序的应用程序角色 **保护资源** ：

```
app.get( '/extra-special', keycloak.protect('other-app:special'), extraSpecialHandler );
```

使用 realm 角色保护资源：

```
app.get( '/admin', keycloak.protect( 'realm:admin' ), adminHandler );
```

基于资源的授权

借助基于资源的身份验证，您可以保护资源及其具体方法/操作，** 基于 Keycloak 中定义的一组策略，从而可以从应用程序外部化授权。这可以通过公开可用于保护资源的 **keycloak.enforcer** 方法来实现。*

```
app.get('/apis/me', keycloak.enforcer('user:profile'), userProfileHandler);
```

keycloak-enforcer 方法以两种模式运行，具体取决于 **response_mode** 配置选项的值。

```
app.get('/apis/me', keycloak.enforcer('user:profile', {response_mode: 'token'}), userProfileHandler);
```

如果将 **response_mode** 设为 **token**，则代表您的应用程序的 bearer 令牌代表的主体从服务器获得权限。在本例中，由由服务器授予权限的 Keycloak 发布新访问令牌。如果服务器没有响应具有预期权限的令牌，则请求被拒绝。使用此模式时，您应该能够从请求中获取令牌，如下所示：

```
app.get('/apis/me', keycloak.enforcer('user:profile', {response_mode: 'token'}), function (req, res) {
  const token = req.kauth.grant.access_token.content;
  const permissions = token.authorization ? token.authorization.permissions : undefined;

  // show user profile
});
```

当应用程序使用会话时，首选使用这个模式，并希望从服务器缓存以前的决定，并自动处理刷新令牌。对于作为客户端和服务器的应用程序，此模式特别有用。

如果将 **response_mode** 设为 **permissions**（默认模式），服务器只返回授予权限的列表，而不会发出新的访问令牌。除了不提供新令牌外，此方法还通过 **请求** 公开服务器授予的权限，如下所示：

```
app.get('/apis/me', keycloak.enforcer('user:profile', {response_mode: 'permissions'}), function (req, res) {
  const permissions = req.permissions;

  // show user profile
});
```

无论使用是否使用了 **response_mode**，cloak **.enforcer** 方法将首先尝试检查发送到应用程序的 bearer 令牌中的权限。如果 bearer 令牌已执行预期的权限，则不需要与服务器交互来获取决策。当您的客户端能够在访问受保护的资源前从具有预期权限的服务器获取访问令牌时，可以使用 Keycloak Authorization Services（如增量授权）提供的一些功能，并避免在 **keycloak.enforcer** 被强制访问资源时对服务器获取访问令牌。

默认情况下，策略 enforcer 将使用定义为应用程序的 **client_id**（例如，通过 **keycloak.json**）引用支持 Keycloak 授权服务的 Keycloak 中的客户端。在这种情况下，如果客户端实际上是资源服务器，客户端不能是公共的。

如果您的应用程序同时充当公共客户端(frontend)和资源服务器，您可以使用以下配置使用您要强制执行的策略引用 Keycloak 中的不同客户端：

```
keycloak.enforcer('user:profile', {resource_server_id: 'my-apiserver'})
```

建议您在 Keycloak 中使用不同的客户端来代表您的前端和后端。

如果使用 Keycloak 授权服务启用了保护应用程序，且您在 **keycloak.json** 中定义了客户端凭证，您可以将额外声明推送到服务器，并将其提供给您的策略以做出决策。为此，您可以定义一个 **claims** 配置选项，它需要一个会返回一个带有您要推送的声明的 JSON 的**函数**：

```
app.get('/protected/resource', keycloak.enforcer(['resource:view', 'resource:write'], {
  claims: function(request) {
    return {
      "http.uri": ["/protected/resource"],
      "user.agent": // get user agent  from request
    }
  }
 }), function (req, res) {
  // access granted
```

有关如何配置 Keycloak 以保护应用程序资源的详情，请查看授权服务 指南。

高级授权

要保护基于 URL 本身的部分资源，假设每个部分都存在角色：

```
function protectBySection(token, request) {
  return token.hasRole( request.params.section );
}

app.get( '/:section/:page', keycloak.protect( protectBySection ), sectionHandler );
```

高级登录配置：

默认情况下，所有未授权的请求都将被重定向到红帽单点登录登录页面，除非您的客户端只有 bearer-only。但是，机密或公共客户端可以托管可浏览和 API 端点。要防止对未经身份验证的 API 请求进行重定向并返回 HTTP 401，您可以覆盖 redirectToLogin 功能。

例如，这个覆盖会检查 URL 包含 /api/ 并禁用登录重定向：

```
Keycloak.prototype.redirectToLogin = function(req) {
const apiReqMatcher = /\/api\//i;
return !apiReqMatcher.test(req.originalUrl || req.url);
};
```

2.3.7. 其他 URL

显式用户触发注销

默认情况下，中间件捕获对 **/logout** 的调用，以通过 Red Hat Single Sign-On-centric logout 工作流向用户发送。这可以通过在 **middleware()** 调用中指定 **logout** 配置参数来更改：

```
app.use( keycloak.middleware( { logout: '/logoff' } ));
```

当调用查询参数 **redirect_url** 时，可以传递用户触发的 logout：

```
https://example.com/logoff?
redirect_url=https%3A%2F%2Fexample.com%3A3000%2Flogged%2Fout
```

然后，这个参数被用作 OIDC logout 端点的重定向 url，并将用户重定向到 **https://example.com/logged/out**。

Red Hat Single Sign-On Admin Callbacks

另外，中间件支持来自红帽单点登录控制台的回调，以注销单个会话或所有会话。默认情况下，这些管理员回调与 / 根 URL 相对进行的，但可通过向 Middleware () 调用提供 **admin** 参数来更改：

```
app.use( keycloak.middleware( { admin: '/callbacks' } );
```

2.3.8. 完整示例

使用 Node.js 适配器用法的完整示例可在 Node.js 的 Keycloak Quickstart 中找到

2.4. 其他 OPENID CONNECT 库

Red Hat Single Sign-On 由提供的适配器（通常更易于使用）进行保护，并可更好地与红帽单点登录集成。但是，如果适配器不适用于您的编程语言、框架或平台，您可能会选择使用通用 OpenID Connect Relying ware(RP)库。本章论述了特定于 Red Hat Single Sign-On 的详细信息，不包含特定的协议详情。如需更多信息，请参阅 OpenID Connect 规格 和 OAuth2 规格。

2.4.1. Endpoints

要理解的最重要端点是 **已知的** 配置端点。它列出了与 Red Hat Single Sign-On 中的 OpenID Connect 实施相关的端点和其他配置选项。端点是：

```
/realms/{realm-name}/.well-known/openid-configuration
```

要获得完整的 URL，请添加 Red Hat Single Sign-On 的基本 URL，并将 **{realm-name}** 替换为您的域的名称。例如：

http://localhost:8080/auth/realms/master/.well-known/openid-configuration

某些 RP 库从此端点检索所有必需的端点，但对于其他端点，您可能需要单独列出端点。

2.4.1.1. 授权端点

```
/realms/{realm-name}/protocol/openid-connect/auth
```

授权端点执行最终用户的身份验证。这可以通过将用户代理重定向到此端点来实现。

如需了解更多详细信息，请参阅 OpenID Connect 规格中的 Authorization Endpoint 部分。

2.4.1.2. 令牌端点

```
/realms/{realm-name}/protocol/openid-connect/token
```

令牌端点用于获取令牌。令牌可以通过交换授权代码来获取，或者直接提供凭证，具体取决于所使用的流。令牌端点还用于在令牌过期时获取新的访问令牌。

如需了解更多详细信息，请参阅 OpenID Connect 规格中的 Token Endpoint 部分。

2.4.1.3. UserInfo 端点

```
/realms/{realm-name}/protocol/openid-connect/userinfo
```

userinfo 端点返回有关经过身份验证的用户的标准声明，并由 bearer 令牌保护。

如需了解更多详细信息，请参阅 OpenID Connect 规格中的 Userinfo Endpoint 部分。

2.4.1.4. 退出端点

```
/realms/{realm-name}/protocol/openid-connect/logout
```

logout 端点会注销经过身份验证的用户。

用户代理可以重定向到端点，在这种情况下，活跃用户会话被注销。之后，用户代理会重新定向到应用程序。

端点也可以直接由应用调用。要直接调用此端点，需要包含刷新令牌以及验证客户端所需的凭据。

2.4.1.5. 证书端点

```
/realms/{realm-name}/protocol/openid-connect/certs
```

证书端点返回由域启用的公钥，并编码为 JSON Web Key(JWK)。根据域设置，可以启用一个或多个密钥来验证令牌。如需更多信息，请参阅 服务器管理指南 和 JSON Web 密钥规格。

2.4.1.6. 内省端点

```
/realms/{realm-name}/protocol/openid-connect/token/introspect
```

内省端点用于检索令牌的活跃状态。换句话说，您可以使用它来验证访问或刷新令牌。它只能由机密客户端调用。

有关如何在此端点上调用的详情，请参阅 OAuth 2.0 Token Introspection specification。

2.4.1.7. 动态客户端注册端点

```
/realms/{realm-name}/clients-registrations/openid-connect
```

动态客户端注册端点用于动态注册客户端。

如需了解更多详细信息，请参阅 客户端注册 章节 和 OpenID Connect Dynamic Client Registration 规格 。

2.4.1.8. 令牌撤销端点

```
/realms/{realm-name}/protocol/openid-connect/revoke
```

令牌撤销端点用于撤销令牌。此端点支持刷新令牌和访问令牌。

有关如何在此端点上调用的详情，请参阅 OAuth 2.0 Token Revocation 规格。

2.4.1.9. 设备授权端点

```
/realms/{realm-name}/protocol/openid-connect/auth/device
```

-

设备授权端点用于获取设备代码和用户代码。它可以通过机密或公共客户端调用。

有关如何在此端点上调用的详情，请参阅 OAuth 2.0 Device Authorization Grant 规格。

2.4.1.10. Backchannel 身份验证端点

```
/realms/{realm-name}/protocol/openid-connect/ext/ciba/auth
```

后端通道身份验证端点用于获取 auth_req_id，用于标识客户端提出的身份验证请求。它只能由机密客户端调用。

有关如何在此端点上调用的详情，请参阅 OpenID Connect Client Initiated Backchannel Authentication Flow 规格。

另请参阅 Red Hat Single Sign-On 文档中的其他部分，如 Client Initiated Backchannel Authentication Grant section of this guide 和 Server Administration Guide 中的 Client Initiated Backchannel Authentication Grant section 部分。

2.4.2. 验证访问令牌

如果您需要手动验证红帽单点登录发布的访问令牌，您可以调用 内省端点。这种方法的缺点是您必须使网络调用 Red Hat Single Sign-On 服务器。如果您同时出现太多验证请求，这可能很慢，可能会给服务器过载。红帽单点登录访问令牌是数字签名和编码的 JSON Web 令牌(JWT)，使用 JSON Web 签名(JWS)。由于它们采用这种方式编码，因此允许您使用发布域的公钥在本地验证访问令牌。您可以在验证代码中硬编码域的公钥，或使用嵌入在 JWS 中的密钥 ID (KID) 的证书端点 来缓存公钥。根据您的代码，有很多第三方库可以使用 JWS 验证。

2.4.3. 流

2.4.3.1. 授权代码

授权代码流将用户代理重定向到红帽单点登录。用户使用红帽单点登录验证后，会创建授权代码，用户代理会重定向到应用程序。然后，应用程序使用授权代码及其凭证从红帽单点登录获取访问令牌、刷新令牌和 ID 令牌。

流目标为 web 应用程序，但也建议用于原生应用程序，包括移动应用程序，其中可以嵌入用户代理。

更多详情请参阅 OpenID Connect 规格中的 授权代码流。

2.4.3.2. 隐式

Implicit 流重定向的工作方式与授权代码流类似，而是返回授权代码，而是返回 Access Token 和 ID Token。这可减少额外的调用来交换 Access Token 的授权代码的需求。但是，它不包括 Refresh Token。这会导致，需要一个长到期的 Access Tokens 中的一个问题，因为这很难使这些令牌无效。或 需要新的重定向以在初始访问令牌到期后获取新的访问令牌。如果应用程序只想验证用户并使用注销本身，则 Implicit 流很有用。

还有一个混合流程，其中返回 Access Token 和 Authorization Code。

需要注意的一点是，Implicit 流和混合流都存在潜在的安全风险，因为访问令牌可能会通过 Web 服务器日志和浏览器历史记录泄露。这可通过对访问令牌使用简短过期来缓解。

更多详情请参阅 OpenID Connect 规格中的Implicit 流。

2.4.3.3. 资源所有者密码凭证

资源所有者密码凭证（称为 Red Hat Single Sign-On 中的 Direct Grant）允许更改令牌的用户凭证。不建议使用这个流，除非您绝对需要。例如，这很有用的是传统应用程序和命令行界面。

使用这个流程有一些限制，包括：

- 用户凭证公开给应用程序
- 应用程序需要登录页面
- 应用程序需要了解验证方案
- 对身份验证流程的更改需要更改应用程序
- 不支持身份代理或社交登录
- 不支持流（用户自我注册、所需操作等）

要让客户端允许使用 Resource Owner Password Credentials 授权客户端启用了 **Direct Access Grants Enabled** 选项。

这个流没有包括在 OpenID Connect 中，而是包括在 OAuth 2.0 规格中。

如需了解更多详细信息，请参阅 OAuth 2.0 规格中的 Resource Owner Password Credentials Grant 章节。

2.4.3.3.1. 使用 CURL 的示例

以下示例演示了如何为 realm **master** 中的一个用户获得访问令牌，用户名 **user**，密码 **password**。该示例使用机密客户端 **myclient**：

```
curl \
  -d "client_id=myclient" \
  -d "client_secret=40cc097b-2a57-4c17-b36a-8fdf3fc2d578" \
  -d "username=user" \
  -d "password=password" \
  -d "grant_type=password" \
  "http://localhost:8080/auth/realms/master/protocol/openid-connect/token"
```

2.4.3.4. 客户端凭证

客户端（应用程序和服务）希望代表自己获取访问权限而不是代表用户时使用客户端凭证。例如，对于一般情况下，对系统应用更改的后台服务（而非特定用户）非常有用。

Red Hat Single Sign-On 支持客户端使用 secret 或公钥/私钥进行身份验证。

这个流没有包括在 OpenID Connect 中，而是包括在 OAuth 2.0 规格中。

如需了解更多详细信息，请参阅 OAuth 2.0 规格中的 客户端 凭证授予章节。

2.4.3.5. 设备授权授予

设备授权授权授权由在互联网连接的设备上运行的客户端使用，这些设备具有有限的输入功能，或缺少合适的浏览器。应用程序会要求红帽单点登录一个设备代码和用户代码。Red Hat Single Sign-On 创建设备代码和用户代码。Red Hat Single Sign-On 会返回包括设备代码和用户代码到应用程序的响应。然后，应

用程序为用户提供用户代码和验证 URI。用户使用另一个浏览器访问要通过验证 URI 进行身份验证。应用程序会重复轮询红帽单点登录，直到红帽单点登录完成用户授权。如果用户身份验证完成，应用程序会获取设备代码。然后，应用程序使用设备代码及其凭证从红帽单点登录获取访问令牌、刷新令牌和 ID 令牌。

如需了解更多详细信息，请参阅 OAuth 2.0 Device Authorization Grant 规格。

2.4.3.6. 客户端初始化的后台验证授予

客户端发起的后台身份验证 Grant 由希望通过直接与 OpenID Provider 通信来启动身份验证流的客户端使用，而无需重定向用户浏览器，如 OAuth 2.0 的授权代码授权代码授权。

客户端请求 Red Hat Single Sign-On 是一个 auth_req_id，用于标识客户端提出的身份验证请求。红帽单点登录创建 auth_req_id。

在收到此 auth_req_id 后，此客户端需要重复轮询红帽单点登录，以获取来自红帽单点登录的 Access Token、Refresh Token 和 ID 令牌，以返回 auth_req_id，直到用户被验证为止。

如果客户端使用 **ping** 模式，它不需要重复轮询令牌端点，但可以等待 Red Hat Single Sign-On 向指定的客户端通知端点发送的通知。可以在红帽单点登录管理控制台中配置客户端通知端点。在 CIBA 规格中介绍了客户端通知端点合同详情。

如需了解更多详细信息，请参阅 OpenID Connect Client Initiated Backchannel Authentication Flow 规格。

另请参阅 Red Hat Single Sign-On 文档的其他位置，如本指南的 Backchannel Authentication Endpoint 和 Server Administrationi ated Backchannel Authentication Grant 部分 （服务器管理指南）。有关 FAPI CIBA 合规性的详细信息，请参阅本指南的 FAPI 部分。

2.4.4. 重定向 URI

使用基于重定向的流时，将有效的重定向 uri 用于您的客户端非常重要。重定向 uri 应该尽量具体。这特别适用于客户端（公共客户端）应用程序。否则可能会导致：

- Open redirects - 攻击者可以利用这个方法创建伪链接，如下所示：
- 未授权条目 - 当用户已通过 Red Hat Single Sign-On 进行身份验证时，攻击者可以使用公共客户端（其中重定向 uris 尚未正确配置）通过重定向用户获得访问权限，而无需用户了解

在生产环境中，web 应用始终使用 **https** 作为所有重定向 URI。不要允许重定向到 http。

还有几个特殊的重定向 URI：

http://localhost

此重定向 URI 对原生应用很有用，并允许原生应用在随机端口上创建 Web 服务器，从而能获取授权代码。此重定向 uri 允许任何端口。

urn:ietf:wg:oauth:2.0:oob

如果无法在客户端上启动 Web 服务器（或浏览器不可用），则可以使用特殊的 **urn:ietf:wg:oauth:2.0:oob** redirect uri。当使用此重定向 uri 时，Red Hat Single Sign-On 在页面的标题和框中显示包含代码的页面。应用程序可以检测浏览器标题已更改，或者用户可以手动将代码复制到应用程序中。通过此重定向 uri，用户也可以使用其他设备获取代码，以转回应用程序。

2.5. 支持财务级 API(FAPI)

Red Hat Single Sign-On 可让管理员更轻松地确保其客户端符合这些规格：

- 财务级 API 安全配置文件 1.0 - 第 1 部分：基本信息
- 财务级 API 安全配置文件 1.0 - 第 2 部分：高级
- 财务级 API：客户端初始化的后台验证 配置文件(FAPI CIBA)

这种合规性意味着 Red Hat Single Sign-On 服务器将验证授权服务器的要求，这在规格中提到。Red Hat Single Sign-On 适配器没有对 FAPI 的任何特定支持，因此客户（应用程序）一侧所需的验证可能仍然需要手动或通过某些其他第三方解决方案完成。

2.5.1. FAPI 客户端配置集

为确保您的客户端兼容 FAPI，您可以在域中配置客户端策略，如 服务器管理指南 所述，并将它们链接到 FAPI 支持的全局客户端配置集，这些配置文件在每个域中自动可用。您可以基于客户端符合的 FAPI 配置集，使用 **fapi-1-baseline** 或 **fapi-1-advanced** 配置集。

如果要使用 Pushed Authorization Request(PAR)，建议您的客户端同时针对 PAR 请求使用 **fapi-1-baseline** 配置集和 **fapi-1-advanced**。具体来说，**fapi-1-baseline** 配置集包含 **pkce-enforcer** executor，以确保客户端使用带有安全 S256 算法的 PKCE。FAPI 高级客户端不需要此项，除非它们使用 PAR 请求。

如果要以 FAPI 兼容方式使用 CIBA，请确保您的客户端同时使用 **fapi-1-advanced** 和 **fapi-ciba** 客户端配置文件。需要使用 **fapi-1-advanced** 配置集，或者包含所请求执行器的其他客户端配置集，因为 **fapi-ciba** 配置集包含仅 CIBA 特定的 executors。当强制 FAPI CIBA 规格的要求时，需要满足机密客户端或证书界访问令牌的执行。

2.5.2. Open Banking Brasil financial-grade API Security Profile

红帽单点登录符合 Open Banking Brasil financial-grade API Security Profile 1.0 implementers Draft 2 。与 FAPI 1 高级 规格相比，这个要求更为严格，因此可能需要以 更严格的方式配置客户端 策略来强制实施某些要求。特别是：

- 如果您的客户端没有使用 PAR，请确保它使用加密的 OIDC 请求对象。这可以通过使用启用了 **Encryption** 的 **secure-request-object** executor 配置的客户端配置集来实现。
- 确保 对于 JWS，客户端使用 **PS256** 算法。对于 JWE，客户端应该使用 **RSA-OAEP** 和 **A256GCM**。这可能需要在适用这些算法的所有 客户端设置 中设置。

2.5.3. TLS 注意事项

随着机密信息交换，所有交互都应该通过 TLS(HTTPS)加密。此外，FAPI 规格中对使用的密码套件和 TLS 协议版本有一些要求。要满足这些要求，您可以考虑配置允许的密码。此配置可以在 Elytron 子系统中的 **KEYCLOAK_HOME/standalone/configuration/standalone-*.xml** 文件中完成。例如，此元素可以在 **tls → server-ssl-contexts** 下添加

```
<server-ssl-context name="kcSSLContext" want-client-auth="true" protocols="TLSv1.2" \
key-manager="kcKeyManager" trust-manager="kcTrustManager" \
cipher-suite-
filter="TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_128_GCM_SHA256,TLS_DHE_RSA_WITH_AES_256_GCM_SHA384" protocols="TLSv1.2" />
```

对 **kcKeyManager** 和 **kcTrustManager** 的引用引用引用对应的 Keystore 和 Truststore。有关更多详细信息以及 Red Hat Single Sign-On 文档，如 网络设置 部分或 X.509 身份验证 部分，请参阅 Wildfly Elytron 子系统文档。

第 3 章 使用 SAML 保护应用程序和服务

本节论述了如何使用 Red Hat Single Sign-On 客户端适配器或通用 SAML 供应商库来保护应用程序和服务。

3.1. JAVA 适配器

Red Hat Single Sign-On 带有 Java 应用程序的不同适配器范围。选择正确的适配器取决于目标平台。

3.1.1. 常规适配器配置

Red Hat Single Sign-On 支持的每个 SAML 客户端适配器均可通过简单的 XML 文本文件配置。这种情况可能如下所示：

```
<keycloak-saml-adapter xmlns="urn:keycloak:saml:adapter"
                       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                       xsi:schemaLocation="urn:keycloak:saml:adapter
https://www.keycloak.org/schema/keycloak_saml_adapter_1_10.xsd">
   <SP entityID="http://localhost:8081/sales-post-sig/"
       sslPolicy="EXTERNAL"
       nameIDPolicyFormat="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
       logoutPage="/logout.jsp"
       forceAuthentication="false"
       isPassive="false"
       turnOffChangeSessionIdOnLogin="false"
       autodetectBearerOnly="false">
       <Keys>
          <Key signing="true" >
             <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
                <PrivateKey alias="http://localhost:8080/sales-post-sig/" password="test123"/>
                <Certificate alias="http://localhost:8080/sales-post-sig/"/>
             </KeyStore>
          </Key>
       </Keys>
       <PrincipalNameMapping policy="FROM_NAME_ID"/>
       <RoleIdentifiers>
          <Attribute name="Role"/>
       </RoleIdentifiers>
       <RoleMappingsProvider id="properties-based-role-mapper">
          <Property name="properties.resource.location" value="/WEB-INF/role-mappings.properties"/>
       </RoleMappingsProvider>
       <IDP entityID="idp"
           signaturesRequired="true">
       <SingleSignOnService requestBinding="POST"
                            bindingUrl="http://localhost:8081/auth/realms/demo/protocol/saml"
                />

          <SingleLogoutService
                requestBinding="POST"
                responseBinding="POST"
                postBindingUrl="http://localhost:8081/auth/realms/demo/protocol/saml"
                redirectBindingUrl="http://localhost:8081/auth/realms/demo/protocol/saml"
                />
          <Keys>
             <Key signing="true">
```

```
            <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
              <Certificate alias="demo"/>
            </KeyStore>
          </Key>
        </Keys>
      </IDP>
    </SP>
</keycloak-saml-adapter>
```

其中一些配置切换可能特定于适配器，一些配置交换机在所有适配器中都很常见。对于 Java 适配器，您可以使用 **${...}** parture 作为系统属性替换。例如： **${jboss.server.config.dir}**

3.1.1.1. SP 元素

以下是 SP 元素属性的解释：

```
<SP entityID="sp"
    sslPolicy="ssl"
    nameIDPolicyFormat="format"
    forceAuthentication="true"
    isPassive="false"
    keepDOMAssertion="true"
    autodetectBearerOnly="false">
...
</SP>
```

entityID

这是此客户端的标识符。IdP 需要这个值来确定客户端与之通信的人员。此设置是 REQUIRED。

sslPolicy

这是适配器要强制执行的 SSL 策略。有效值为：**所有**、**EXTERNAL** 和 **NONE**。对于 **ALL**，所有请求都必须通过 HTTPS。对于 **EXTERNAL**，只有非专用 IP 地址必须通过 HTTPS 进行线路。对于 **NONE**，不需要通过 HTTPS 访问请求。此设置是 OPTIONAL。默认值为 **EXTERNAL**。

nameIDPolicyFormat

SAML 客户端可以请求特定的 NameID 主体格式。如果您希望特定格式，请填写此值。它必须是标准的 SAML 格式标识符：urn **:oasis:names:tc:SAML:2.0:nameid-format:transient** 此设置是 OPTIONAL。默认情况下，请求没有特殊格式。

forceAuthentication

SAML 客户端可以请求重新验证用户，即使他们已在 IdP 登录时。把它设置为 **true** 来启用。此设置是 OPTIONAL。默认值为 **false**。

isPassive

SAML 客户端可能会要求用户进行身份验证，即使他们没有在 IdP 登录时也如此。如果需要这样做，则将其设置为 **true**。不要与 **forceAuthentication** 一起使用，因为它们相反。此设置是 OPTIONAL。默认值为 **false**。

turnOffChangeSessionIdOnLogin

在某些平台上成功登录时，会话 ID 被默认更改，以插入安全攻击向量。将此更改为 **true** 以禁用此功能。建议您不要关闭它。默认值为 **false**。

autodetectBearerOnly

如果您的应用程序同时充当 Web 应用程序和 Web 服务（如 SOAP 或 REST），则此项应设为 true。它允许您将 Web 应用的未经身份验证的用户重定向到 Red Hat Single Sign-On 登录页面，但向未经身份验证的 SOAP 或 REST 客户端发送 HTTP **401** 状态代码，因为它们不知道到登录页面的重定向。红

帽单点登录根据典型的标头（如 **X-Requested-With**、**SOAPAction** 或 **Accept**）自动探测 SOAP 或 REST 客户端。默认值为 false。

logoutPage

这会将页面设置为在注销后显示。如果页面是一个完整的 URL，如 **http://web.example.com/logout.html**，则在使用 HTTP **302** 状态代码注销至该页面后，用户会被重定向到该页面。如果指定了没有方案部分的链接，如 **/logout.jsp**，则该页面会在注销后显示，无论它是否根据 web.xml 中的 **security-constraint** 声明显示在受保护区域中，并且页面会相对于部署上下文 root 解析。

keepDOMAssertion

此属性应设为 true，以便适配器将声明的 DOM 表示存储在与请求关联的 SamlPrincipal 的 **SamlPrincipal** 中。可以使用主体中的 **getAsertionDocument** 来检索 断言文档。这在重新显示签名的断言时特别有用。返回的文档是生成解析 Red Hat Single Sign-On 服务器收到的 SAML 响应的文件。此设置是 OPTIONAL，其默认值为 false （文件没有在主体内保存）。

3.1.1.2. 服务提供商密钥和关键元素

如果 IdP 需要客户端应用程序（或 SP）为其所有请求签名，或者，如果 IdP 会加密断言，则必须定义用于执行此操作的密钥。对于客户端签名文档，您必须定义用于签署文档的私钥和公钥或证书。为了进行加密，您只需要定义用于解密的私钥。

可以通过两种方式描述您的密钥。它们可存储在 Java KeyStore 中，或者您可以使用 PEM 格式直接将密钥复制到 **keycloak-saml.xml** 中。

```
<Keys>
    <Key signing="true" >
      ...
    </Key>
</Keys>
```

Key 元素有两个可选属性 **signing** 和 **encryption**。当设置为 true 时，它们告诉适配器将用于什么键。如果这两个属性都设为 true，则密钥将同时用于签名文档和解密加密的断言。您必须至少将其中一个属性设置为 true。

3.1.1.2.1. keystore 元素

在 **Key** 元素中，您可以从 Java 密钥存储加载您的密钥和证书。这在 **KeyStore** 元素中声明。

```
<Keys>
    <Key signing="true" >
        <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
            <PrivateKey alias="myPrivate" password="test123"/>
            <Certificate alias="myCertAlias"/>
        </KeyStore>
    </Key>
</Keys>
```

以下是使用 **KeyStore** 元素定义的 XML 配置属性。

file

密钥存储的文件路径。此选项是 OPTIONAL。必须设置 file 或 resource 属性。

resource

WAR 资源路径到 KeyStore。这是对 ServletContext.getResourceAsStream （）方法调用的路径。此选项是 OPTIONAL。必须设置 file 或 resource 属性。

password

KeyStore 的密码。此选项为 REQUIRED。

*如果要定义 SP 用于签名的密钥，还必须在 Java KeyStore 中指定您的私钥和证书的引用。上例中的 **PrivateKey** 和 **Certificate** 元素定义了一个 **alias**，指向密钥存储内密钥或证书。密钥存储需要额外密码才能访问私钥。在 **PrivateKey** 元素中，您必须在 **password** 属性中定义此密码。*

3.1.1.2.2. 密钥 PEMS

*在 **Key** 元素中，您可以使用子元素（ **PrivateKeyPem**、**PublicKeyPem** 和 **CertificatePem** ）来直接声明您的密钥和证书。这些元素中包含的值必须符合 PEM 密钥格式。如果您使用 **openssl** 或类似的命令行工具生成密钥，通常会使用这个选项。*

```
<Keys>
  <Key signing="true">
    <PrivateKeyPem>
      2341251234AB31234==231BB998311222423522334
    </PrivateKeyPem>
    <CertificatePem>
      211111341251234AB31234==231BB998311222423522334
    </CertificatePem>
  </Key>
</Keys>
```

3.1.1.3. SP PrincipalNameMapping 元素

*这个元素是可选的。创建您从 **HttpServletRequest.getUserPrincipal（）** 等方法获取的 Java Principal 对象时，您可以定义 **Principal.getName（）** 方法返回的名称。*

```
<SP ...>
  <PrincipalNameMapping policy="FROM_NAME_ID"/>
</SP>

<SP ...>
  <PrincipalNameMapping policy="FROM_ATTRIBUTE" attribute="email" />
</SP>
```

***policy** 属性定义用于填充该值的策略。此属性的可能值有：*

FROM_NAME_ID

此策略仅使用任何 SAML 主题值。这是默认设置

FROM_ATTRIBUTE

*这将从服务器收到的 SAML 断言中声明的属性中拉取值。您将需要指定要在 **attribute** XML 属性中使用的 SAML 断言属性的名称。*

3.1.1.4. RoleIdentifiers 元素

***RoleIdentifiers** 元素定义了从用户接收的断言中的 SAML 属性，用作用户的 Jakarta EE 安全上下文中的角色标识符。*

```
<RoleIdentifiers>
```

```
    <Attribute name="Role"/>
    <Attribute name="member"/>
    <Attribute name="memberOf"/>
</RoleIdentifiers>
```

默认情况下，**Role** 属性值转换为 Jakarta EE 角色。有些 IdP 使用 **member** 或 **memberOf** 属性断言角色。您可以定义一个或多个 **Attribute** 元素，以指定哪些 SAML 属性必须转换为角色。

3.1.1.5. RoleMappingsProvider 元素

RoleMappingsProvider 是一个可选元素，它允许规范 **org.keycloak.adapters.saml.RoleMappingsProvider** SPI 实现，供 SAML 适配器使用。

当 Red Hat Single Sign-On 用作 IDP 时，可以使用角色映射程序中的构建来映射任何角色，然后才能将它们添加到 SAML 断言。但是，SAML 适配器可用于发送 SAML 请求到第三方 IDP，在这种情况下，可能需要将断言中提取的角色映射到 SP 需要的不同角色。**RoleMappingsProvider** SPI 允许配置可用来执行必要映射的可插拔角色映射程序。

供应商的配置如下所示：

```
...
<RoleIdentifiers>
    ...
</RoleIdentifiers>
<RoleMappingsProvider id="properties-based-role-mapper">
    <Property name="properties.resource.location" value="/WEB-INF/role-mappings.properties"/>
</RoleMappingsProvider>
<IDP>
    ...
</IDP>
```

id 属性标识要使用哪个已安装的提供程序。**Property** 子元素可以多次使用来为提供程序指定配置属性。

3.1.1.5.1. 基于属性的角色映射提供程序

Red Hat Single Sign-On 包括一个 **RoleMappingsProvider** 实现，它使用 **properties** 文件来执行角色映射。此提供程序由 id **属性-role-mapper** 识别，并由 **org.keycloak.adapters.saml.PropertiesBasedRoleMapper** 类实施。

此提供程序依赖于两个配置属性，它们可用于指定要使用的 **属性** 文件的位置。首先，它会检查 **properties.file.location** 是否被设置，使用配置的值来查找文件系统中的 **properties** 文件。如果配置文件不位于，则提供程序会抛出 **RuntimeException**。以下片段演示了使用 **properties.file.configuration** 选项从文件系统中的 **/opt/mappers/** 目录中加载 **roles.properties** 文件的供应商示例：

```
    <RoleMappingsProvider id="properties-based-role-mapper">
        <Property name="properties.file.location" value="/opt/mappers/roles.properties"/>
    </RoleMappingsProvider>
```

如果没有设置 **properties.file.location**，则供应商会检查 **properties.resource.location** 属性，使用配置的值来从 **WAR** 资源中加载 **properties** 文件。如果该配置属性也不存在，则供应商将默认尝试从 **/WEB-INF/role-mappings.properties** 加载该文件。无法从资源加载文件将导致提供程序引发 **RuntimeException**。以下片段演示了使用 **properties.resource.location** 从应用程序的 **/WEB-INF/conf/** 目录中加载 **roles.properties** 文件的供应商示例：

```
<RoleMappingsProvider id="properties-based-role-mapper">
    <Property name="properties.resource.location" value="/WEB-INF/conf/roles.properties"/>
</RoleMappingsProvider>
```

属性 文件可以包含角色和主体作为键，以及以逗号分隔的零个角色列表作为值。调用后，如果存在映射，实施将迭代从断言和检查中提取的角色集合。如果角色映射到空角色，它将被丢弃。如果它映射到一个或多个不同的角色，则会在结果集中设置这些角色。如果没有为角色找到映射，则会包含在结果集中。

处理角色后，实施将检查从断言中提取的主体是否包含一个条目 **属性文件**。如果存在主体的映射，则所有作为值列出的角色都会添加到结果集中。这允许将额外角色分配给主体。

例如，假设供应商已经使用以下属性文件进行配置：

```
roleA=roleX,roleY
roleB=

kc_user=roleZ
```

如果主体 **kc_user** 从角色 **roleA**、**roleB** 和 **roleC** 中提取，则分配给主体的最终角色集将是 **roleC**、**roleX**、**roleY** 和 **roleZ**，因为 **roleA** 被映射到 **roleX** 和 **roleY**。 **roleB** 已映射到空角色 - 因此被丢弃，**roleC** 被用作，最后会添加额外的角色到 **kc_user** 主体(**roleZ**)。

注：要在角色名称中使用空格进行映射，使用 unicode 替换空间。例如，传入"角色 A"将如下所示：

```
role\u0020A=roleX,roleY
```

3.1.1.5.2. 添加您自己的角色映射供应商

要添加自定义角色映射提供程序，只需实施 **org.keycloak.adapters.saml.RoleMappingsProvider** SPI。如需了解更多详细信息，请参阅 服务器开发者指南中的 **SAML 角色映射 SPI** 部分。

3.1.1.6. IDP Element

IDP 元素中的所有内容描述了 SP 通信身份提供程序（身份验证服务器）的设置。

```
<IDP entityID="idp"
    signaturesRequired="true"
    signatureAlgorithm="RSA_SHA1"
    signatureCanonicalizationMethod="http://www.w3.org/2001/10/xml-exc-c14n#">
...
</IDP>
```

以下是您可以在 **IDP** 元素声明中指定的属性配置选项。

entityID

这是 IDP 的签发者 ID。此设置是 REQUIRED。

signaturesRequired

如果设置为 **true**，则客户端适配器都将为其发送到 IDP 的每个文档进行签名。另外，客户端会期望 IDP 将会对发送给它的所有文档进行签名。此切换为所有请求和响应类型设置默认值，但稍后您会看到您对此有非常精细的控制。此设置是 选项， 并且默认为 **false**。

signatureAlgorithm

这是 IDP 期望签名的文档使用的签名算法。允许的值有： ***RSA_SHA1***、***RSA_SHA256***、***RSA_SHA512*** *和* ***DSA_SHA1***。*此设置是 OPTIONAL，默认为* ***RSA_SHA256***。

signatureCanonicalizationMethod

这是 IDP 预期签名的文档要使用的签名规范方法。此设置是 OPTIONAL。默认值为 ***http://www.w3.org/2001/10/xml-exc-c14n#***，*它应该适用于大多数 IDP。*

metadataUrl

用于检索 IDP 元数据的 URL，目前它只用来定期获取签名和加密密钥，允许循环在 IDP 上不手动更改这些密钥。

3.1.1.7. IDP AllowedClockSkew 子元素

AllowedClockSkew *可选子元素定义 IDP 和 SP 之间的允许时钟偏移。默认值为 0。*

```
<AllowedClockSkew unit="MILLISECONDS">3500</AllowedClockSkew>
```

unit

可以定义附加到此元素值的时间范围。允许的值是 MICROSECONDS、MILLISECONDS、MINUTES、NANOSECONDS 和 SECONDS。这是 选项。默认值为 ***SECONDS***。

3.1.1.8. IDP SingleSignOnService 子元素

SingleSignOnService *子元素定义 IDP 的登录 SAML 端点。客户端适配器会在需要登录时通过此元素中的设置将请求发送到 IDP。*

```
<SingleSignOnService signRequest="true"
                     validateResponseSignature="true"
                     requestBinding="post"
                     bindingUrl="url"/>
```

以下是您可以在此元素上定义的配置属性：

signRequest

客户端签名身份验证是否请求？此设置是 OPTIONAL。默认为 IDP ***签名Required*** *元素值。*

validateResponseSignature

客户端是否希望 IDP 签署从uhtn 请求发回的断言响应文档？此设置是 OPTIONAL。默认为 IDP ***签名Required*** *元素值。*

requestBinding

这是用于与 IDP 通信的 SAML 绑定类型。此设置是 OPTIONAL。默认值为 ***POST***，*但您也可以将其设置为* ***REDIRECT***。

responseBinding

SAML 允许客户端请求使用什么绑定类型。这个值可以是 ***POST*** *或* ***REDIRECT***。*此设置是 OPTIONAL。默认值是客户端不会为响应请求特定绑定类型。*

assertionConsumerServiceUrl

断言使用者服务(ACS)的 URL，IDP 登录服务应向其中发送响应。此设置是 OPTIONAL。默认情况下，不设置它，它依赖于 IdP 中的配置。设置后，它必须以 ***/saml*** *结尾，例如* ***http://sp.domain.com/my/endpoint/for/saml***。*此属性的值以 SAML* ***AuthnRequest*** *消息的* ***AssertionConsumerServiceURL*** *属性中发送。此属性通常由* ***responseBinding*** *属性使用。*

bindingUrl

这是客户端将向其发送请求的 IDP 登录服务的 URL。此设置是 REQUIRED。

3.1.1.9. IDP SingleLogoutService 子元素

SingleLogoutService 子元素定义 IDP 的注销 SAML 端点。在希望注销时，客户端适配器将通过此元素中的设置将请求发送到 IDP。

```
<SingleLogoutService validateRequestSignature="true"
                     validateResponseSignature="true"
                     signRequest="true"
                     signResponse="true"
                     requestBinding="redirect"
                     responseBinding="post"
                     postBindingUrl="posturl"
                     redirectBindingUrl="redirecturl">
```

signRequest

客户端注销请求是否应该进入 IDP？此设置是 OPTIONAL。默认为 IDP **签名Required** 元素值。

signResponse

客户端注销响应是否应该发送到 IDP 请求？此设置是 OPTIONAL。默认为 IDP **签名Required** 元素值。

validateRequestSignature

客户端是否应该从 IDP 中签署注销请求文档？此设置是 OPTIONAL。默认为 IDP **签名Required** 元素值。

validateResponseSignature

客户端是否应该从 IDP 中签署注销响应文档？此设置是 OPTIONAL。默认为 IDP **签名Required** 元素值。

requestBinding

这是用于向 IDP 通信 SAML 请求的 SAML 绑定类型。此设置是 OPTIONAL。默认值为 **POST**，但您也可以将其设置为 REDIRECT。

responseBinding

这是用于向 IDP 通信 SAML 响应的 SAML 绑定类型。这个值可以是 **POST** 或 **REDIRECT**。此设置是 OPTIONAL。默认值为 **POST**，但您也可以将其设置为 **REDIRECT**。

postBindingUrl

这是使用 POST 绑定时 IDP 的注销服务的 URL。如果使用 **POST** 绑定，则此设置为 REQUIRED。

redirectBindingUrl

这是使用 REDIRECT 绑定时 IDP 注销服务的 URL。如果使用 REDIRECT 绑定，则此设置为 REQUIRED。

3.1.1.10. IDP Keys 子元素

IDP 的密钥子元素仅用于定义用于验证由 IDP 签署的证书或公钥。它定义方式与 SP 的 Keys 元素 相同。但是，您只需要定义一个证书或公钥引用。请注意，如果 Red Hat Single Sign-On 服务器和 adapter 分别意识到 IDP 和 SP，则不需要指定签名验证的密钥，请参阅以下。

可将 SP 配置为从已发布的证书中获取 IDP 签名验证的公钥，只要 Red Hat Single Sign-On 均实施 SP 和 IDP。这可以通过删除 Keys 子元素中的所有签名验证密钥声明来实现。如果 Keys 子元素保持为空，则可以完全省略它。然后，SP 从 SAML 描述符自动获取密钥，该位置派生自 IDP SingleSignOnService 子元素 中指定的 SAML 端点 URL。用于 SAML 描述符检索的 HTTP 客户端的设置通常需要额外的配置，但它可以在 IDP HttpClient 子元素 中配置。

也可以指定多个用于签名验证的密钥。这可以通过在 Keys 子元素内声明多个 Key 元素，并将 **signing** 属性设置为 **true**。例如，当 IDP 签名密钥被轮转时，这很有用：新的 SAML 协议消息和断言通常是一个转换周期，但仍应该接受之前密钥签名的断言。

无法将 Red Hat Single Sign-On 配置为自动获取签名验证的密钥并定义额外的静态签名验证密钥。

```
<IDP entityID="idp">
    ...
    <Keys>
        <Key signing="true">
            <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
                <Certificate alias="demo"/>
            </KeyStore>
        </Key>
    </Keys>
</IDP>
```

3.1.1.11. IDP HttpClient 子元素

HttpClient 可选子元素定义了 HTTP 客户端的属性，用于在 启用时 通过 IDP 的 SAML 描述符自动获取包含 IDP 签名验证的公钥的 HTTP 客户端的属性。

```
<HttpClient connectionPoolSize="10"
        disableTrustManager="false"
        allowAnyHostname="false"
        clientKeystore="classpath:keystore.jks"
        clientKeystorePassword="pwd"
        truststore="classpath:truststore.jks"
        truststorePassword="pwd"
        proxyUrl="http://proxy/"
        socketTimeout="5000"
        connectionTimeout="6000"
        connectionTtl="500" />
```

connectionPoolSize

此配置选项定义应池与红帽单点登录服务器的连接数量。这是 选项。默认值为 **10**。

disableTrustManager

如果 Red Hat Single Sign-On 服务器需要 HTTPS，且这个配置选项被设置为 **true**，则不需要指定信任存储。此设置应只在开发期间使用，**永远不会** 在生产环境中使用，因为它将禁用 SSL 证书的验证。这是 选项。默认值为 **false**。

allowAnyHostname

如果 Red Hat Single Sign-On 服务器需要 HTTPS，且该配置选项被设置为 **true**，则红帽单点登录服务器的证书通过信任存储进行验证，但不会执行主机名验证。此设置只应在开发期间使用，**永远不会** 在生产环境中使用，因为它将部分禁用 SSL 证书的验证。这集在测试环境中可能很有用。这是 选项。默认值为 **false**。

truststore

值是信任存储文件的文件路径。如果您使用 **classpath:** 前缀，则信任存储将从部署的 classpath 获取。用于向 Red Hat Single Sign-On 服务器的传出 HTTPS 通信。进行 HTTPS 请求的客户端需要一种验证它们要通信的服务器主机的方法。这就是信任者的作用。该密钥存储包含一个或多个可信主机证书或证书颁发机构。您可以通过提取红帽单点登录服务器的 SSL 密钥存储的公共证书来创建此信任存储。这是 REQUIRED，除非 **disableTrustManager** 为 **true**。

truststorePassword

truststore 的密码。如果设置了 **truststore** 且信任存储需要密码，则这是 REQUIRED。

clientKeystore

这是密钥存储文件的文件路径。当适配器向 Red Hat Single Sign-On 服务器发出 HTTPS 请求时，此密钥存储包含双向 SSL 的客户端证书。这是 选项。

clientKeystorePassword

客户端密钥存储的密码，以及客户端的密钥。如果设置了 **clientKeystore**，则这是 REQUIRED。

proxyUrl

用于 HTTP 连接的 HTTP 代理服务器的 URL。这是 选项。

socketTimeout

在建立连接（以毫秒为单位）后，套接字等待数据的超时。在两个数据包之间不活跃的最大时间。超时值为零将解释为无限超时。负值代表为未定义（如果适用，系统默认值）。默认值为 **-1**。这是 选项。

connectionTimeout

与远程主机建立连接的超时（以毫秒为单位）。超时值为零将解释为无限超时。负值代表为未定义（如果适用，系统默认值）。默认值为 **-1**。这是 选项。

connectionTtl

以毫秒为单位连接客户端到实时的连接时间。小于或等于零的值被解释为无限值。默认值为 **-1**。这是选项。

3.1.2. JBoss EAP 适配器

为了保护 JBoss EAP 上部署的 WAR 应用安全，您必须安装并配置 Red Hat Single Sign-On SAML Adapter subsystem。

然后，您可以在 WAR 中提供 keycloak 配置 **/WEB-INF/keycloak-saml.xml** 文件，并将 auth-method 更改为 web.xml 中的 KEYCLOAK-SAML。

您可以使用 ZIP 文件或 RPM 安装适配器。

- 从 ZIP 文件安装适配器
- 从 RPM 安装 JBoss EAP 7 适配器
- 从 RPM 安装 JBoss EAP 6 适配器

3.1.3. 从 ZIP 文件安装适配器

每个适配器是在 Red Hat Single Sign-On 下载站点单独下载。

流程

1. 从 Downloads 网站安装适用于您的应用程序服务器的适配器。
   - 在 JBoss EAP 7.x 上安装：

     ```
     $ cd $EAP_HOME
     $ unzip rh-sso-saml-eap7-adapter.zip
     ```

   - 在 JBoss EAP 6.x 上安装：

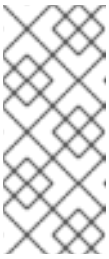
```
$ cd $EAP_HOME
$ unzip rh-sso-saml-eap6-adapter.zip
```

这些 ZIP 文件会创建特定于 JBoss EAP 分发中的 JBoss EAP SAML 适配器的新 JBoss 模块。

2. 使用 CLI 脚本在应用服务器的服务器配置中启用红帽单点登录 SAML subsystem： **domain.xml** 或 **standalone.xml**。
启动服务器并运行适用于您的应用服务器的脚本。

- 对 JBoss EAP 7.1 或更新版本使用此命令

```
$ cd $JBOSS_HOME
$ ./bin/jboss-cli.sh -c --file=bin/adapter-elytron-install-saml.cli
```



注意

EAP 分别支持 OpenJDK 17 和 Oracle JDK 17 ，自 7.4.CP7 和 7.4.CP8。请注意，新的 java 版本使 elytron 变体 compulsory，因此不要使用带有 JDK 17 的传统适配器。另外，在运行适配器 CLI 文件后，执行 EAP 提供的 **enable-elytron-se17.cli** 脚本。这两个脚本都需要配置 elytron 适配器并删除不兼容的 EAP 子系统。如需了解更多详细信息，请参阅此安全配置更改文章。

- 对于 JBoss EAP 7.0 和 EAP 6.4 使用此命令

```
$ cd $JBOSS_HOME
$ ./bin/jboss-cli.sh -c --file=bin/adapter-install-saml.cli
```

注意

也可以在 JBoss EAP 7.1 或更新版本中使用旧的非tron 适配器，这意味着您可以在这些版本中使用 **adapter-install-saml.cli**。但是，我们建议您使用较新的 Elytron 适配器。

该脚本将添加扩展、子系统以及可选的 security-domain，如下所述。

```
<server xmlns="urn:jboss:domain:1.4">

  <extensions>
    <extension module="org.keycloak.keycloak-saml-adapter-subsystem"/>
     ...
  </extensions>

  <profile>
    <subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1"/>
    ...
  </profile>
```

当您需要在受保护的 Web 层中创建的安全上下文被传播到您正调用的 EJB（其它 EE 组件）时，其中的 **keycloak** 安全域应当与 EJBs 和其他组件一起使用。否则此配置是可选的。

```
<server xmlns="urn:jboss:domain:1.4">
 <subsystem xmlns="urn:jboss:domain:security:1.2">
```

```
    <security-domains>
...
      <security-domain name="keycloak">
        <authentication>
          <login-module code="org.keycloak.adapters.jboss.KeycloakLoginModule"
                        flag="required"/>
        </authentication>
      </security-domain>
    </security-domains>
```

安全上下文自动传播到 EJB 层。

3.1.3.1. JBoss SSO

JBoss EAP 内置了对部署到同一 JBoss EAP 实例的 Web 应用程序的单点登录支持。使用 Red Hat Single Sign-On 时不应该启用此功能。

3.1.3.2. 为 JSESSIONID Cookie 设置 SameSite 值

浏览器计划将 Cookie 的 **SameSite** 属性的默认值设置为 **Lax**。此设置表示，仅当请求源自于同一域中时，cookies 才会发送到应用程序。这个行为可能会影响 SAML POST 绑定，这可能会变为无法正常工作。要保留 SAML 适配器的完整功能，我们建议将容器创建的 **JSESSIONID** cookie 的 **SameSite** 值设置为 **None**。不这样做可能会导致对每个请求重置容器的会话。

注意

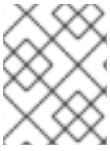
要避免将 **SameSite** 属性设置为 **None**，请考虑切换到 REDIRECT 绑定（如果可接受），或者使用这个临时解决方案的 OIDC 协议。

要将 Wildfly/EAP 中的 **JSESSIONID** cookie 的 **SameSite** 值设置为 **None**，请在应用程序的 **WEB-INF** 目录中添加 file **undertow-handlers.conf**。

```
samesite-cookie(mode=None, cookie-pattern=JSESSIONID)
```

对此配置的支持在 19.1.0 版本中的 Wildfly 中提供。

3.1.4. 从 RPM 安装 JBoss EAP 7 Adapter



注意

使用 Red Hat Enterprise Linux 7 时，术语频道被替换为术语仓库。这些说明中，仅使用术语库。

先决条件

您必须订阅 JBoss EAP 7 存储库，然后才能从 RPM 安装 EAP 7 适配器。

- 确定使用 Red Hat Subscription Manager 将 Red Hat Enterprise Linux 系统注册到您的帐户。如需更多信息，请参阅 Red Hat Subscription Management 文档。
- 如果您已经订阅了另一个 JBoss EAP 存储库，必须先退出该存储库。

对于 Red Hat Enterprise Linux 6，7：使用 Red Hat Subscription Manager，通过以下命令订阅 JBoss EAP 7.4 存储库。根据您的 Red Hat Enterprise Linux 版本，将 <RHEL_VERSION> 替换为 6 或 7。

```
$ sudo subscription-manager repos --enable=jb-eap-7-for-rhel-<RHEL_VERSION>-server-rpms
```

对于 Red Hat Enterprise Linux 8：使用 Red Hat Subscription Manager，使用以下命令订阅 JBoss EAP 7.4 存储库：

```
$ sudo subscription-manager repos --enable=jb-eap-7.4-for-rhel-8-x86_64-rpms --enable=rhel-8-for-x86_64-baseos-rpms --enable=rhel-8-for-x86_64-appstream-rpms
```

流程

1. 根据您的 Red Hat Enterprise Linux 版本，为 SAML 安装 EAP 7 适配器。
   - 在 Red Hat Linux 7 上安装：

     ```
     $ sudo yum install eap7-keycloak-saml-adapter-sso7_6
     ```

   - 在 Red Hat Enterprise Linux 8 中安装：

     ```
     $ sudo dnf install eap7-keycloak-adapter-sso7_6
     ```

     注意

     RPM 安装的默认 EAP_HOME 路径为 /opt/rh/eap7/root/usr/share/wildfly。

2. 为 SAML 模块运行安装脚本：

   ```
   $ $EAP_HOME/bin/jboss-cli.sh -c --file=$EAP_HOME/bin/adapter-install-saml.cli
   ```

您的安装已完成。

3.1.5. 从 RPM 安装 JBoss EAP 6 Adapter

注意

使用 Red Hat Enterprise Linux 7 时，术语频道被替换为术语仓库。这些说明中，仅使用术语库。

先决条件

您必须订阅 JBoss EAP 6 存储库，然后才能从 RPM 安装 EAP 6 适配器。

- 确定使用 Red Hat Subscription Manager 将 Red Hat Enterprise Linux 系统注册到您的帐户。如需更多信息，请参阅 Red Hat Subscription Management 文档。
- 如果您已经订阅了另一个 JBoss EAP 存储库，必须先退出该存储库。

- 使用 Red Hat Subscription Manager，使用以下命令订阅 JBoss EAP 6 存储库。根据您的 Red Hat Enterprise Linux 版本，将 <RHEL_VERSION> 替换为 6 或 7。

```
$ sudo subscription-manager repos --enable=jb-eap-6-for-rhel-<RHEL_VERSION>-server-rpms
```

流程

1. 使用以下命令为 SAML 安装 EAP 6 适配器：

```
$ sudo yum install keycloak-saml-adapter-sso7_6-eap6
```

注意

RPM 安装的默认 EAP_HOME 路径为 /opt/rh/eap6/root/usr/share/wildfly。

2. 为 SAML 模块运行安装脚本：

```
$ $EAP_HOME/bin/jboss-cli.sh -c --file=$EAP_HOME/bin/adapter-install-saml.cli
```

您的安装已完成。

3.1.5.1. 保护 WAR

这部分论述了如何通过在 WAR 软件包中添加配置和编辑文件来直接保护 WAR。

您首先在 WAR 的 **WEB-INF** 目录下创建一个 **keycloak-saml.xml** 适配器配置文件。常规 适配器配置部分描述了此配置文件的格式。

接下来，您必须在 **web.xml** 中将 **auth-method** 设置为 **KEYCLOAK-SAML**。您还必须使用标准 servlet 安全性为您的 URL 指定角色基础限制。以下是 web.xml 文件示例：

```
<web-app 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_3_0.xsd"
    version="3.0">

 <module-name>customer-portal</module-name>

   <security-constraint>
      <web-resource-collection>
         <web-resource-name>Admins</web-resource-name>
         <url-pattern>/admin/*</url-pattern>
      </web-resource-collection>
      <auth-constraint>
         <role-name>admin</role-name>
      </auth-constraint>
      <user-data-constraint>
         <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>
   </security-constraint>
   <security-constraint>
```

```
        <web-resource-collection>
            <web-resource-name>Customers</web-resource-name>
            <url-pattern>/customers/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <role-name>user</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>CONFIDENTIAL</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>KEYCLOAK-SAML</auth-method>
        <realm-name>this is ignored currently</realm-name>
    </login-config>

    <security-role>
        <role-name>admin</role-name>
    </security-role>
    <security-role>
        <role-name>user</role-name>
    </security-role>
</web-app>
```

除 **auth-method** 设置外的所有标准 servlet 设置。

3.1.5.2. 使用红帽单点登录 SAML 子系统保护 WAR

您不必打开 WAR 才能使用 Red Hat Single Sign-On 对其进行保护。另外，您还可以通过 Red Hat Single Sign-On SAML Adapter subsystem 对外部保护。虽然您不必将 KEYCLOAK-SAML 指定为 **auth-method**，但您仍必须在 **web.xml** 中定义 **security-constraints**。但是，您不必创建 **WEB-INF/keycloak-saml.xml** 文件。这个元数据会在服务器的 **domain.xml** 或 **standalone.xml** 子系统配置部分中的 XML 中定义。

```
<extensions>
  <extension module="org.keycloak.keycloak-saml-adapter-subsystem"/>
</extensions>

<profile>
  <subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1">
    <secure-deployment name="WAR MODULE NAME.war">
      <SP entityID="APPLICATION URL">
        ...
      </SP>
    </secure-deployment>
  </subsystem>
</profile>
```

secure-deployment name 属性标识您要保护的 WAR。其值是 **web.xml** 中定义的 **module-name**，并附加 **.war**。其余的配置使用与 General Adapter Config 中定义的 **keycloak-saml.xml** 配置相同的 XML 语法。

配置示例：

```
<subsystem xmlns="urn:jboss:domain:keycloak-saml:1.1">
  <secure-deployment name="saml-post-encryption.war">
    <SP entityID="http://localhost:8080/sales-post-enc/"
        sslPolicy="EXTERNAL"
        nameIDPolicyFormat="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
        logoutPage="/logout.jsp"
        forceAuthentication="false">
      <Keys>
        <Key signing="true" encryption="true">
          <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
            <PrivateKey alias="http://localhost:8080/sales-post-enc/" password="test123"/>
            <Certificate alias="http://localhost:8080/sales-post-enc/"/>
          </KeyStore>
        </Key>
      </Keys>
      <PrincipalNameMapping policy="FROM_NAME_ID"/>
      <RoleIdentifiers>
        <Attribute name="Role"/>
      </RoleIdentifiers>
      <IDP entityID="idp">
        <SingleSignOnService signRequest="true"
            validateResponseSignature="true"
            requestBinding="POST"
            bindingUrl="http://localhost:8080/auth/realms/saml-demo/protocol/saml"/>

        <SingleLogoutService
            validateRequestSignature="true"
            validateResponseSignature="true"
            signRequest="true"
            signResponse="true"
            requestBinding="POST"
            responseBinding="POST"
            postBindingUrl="http://localhost:8080/auth/realms/saml-demo/protocol/saml"
            redirectBindingUrl="http://localhost:8080/auth/realms/saml-demo/protocol/saml"/>
        <Keys>
          <Key signing="true" >
            <KeyStore resource="/WEB-INF/keystore.jks" password="store123">
              <Certificate alias="saml-demo"/>
            </KeyStore>
          </Key>
        </Keys>
      </IDP>
    </SP>
  </secure-deployment>
</subsystem>
```

3.1.6. Java Servlet 过滤器适配器

如果要将 SAML 与不属于 servlet 平台的适配器的 Java servlet 应用程序搭配使用，您可以选择使用 Red Hat Single Sign-On 拥有的 servlet 过滤器。这个适配器的工作方式与其它适配器不同。您仍然需要指定一个 **/WEB-INF/keycloak-saml.xml** 文件，如 General Adapter Config 部分中定义，但您不会在 web.xml 中定义安全限制。反之，您可以使用 Red Hat Single Sign-On servlet 过滤器适配器定义一个过滤器映射来保护您要保护的 url 模式。

注意

Backchannel logout 与标准适配器不同。它而不是无效的 http 会话，而是将会话 ID 标记为已注销。不只是一个基于会话 ID 对 http 会话进行无效的方法。

警告

当您有一个使用 SAML 过滤器的集群应用程序时，恢复通道注销将无法正常工作。

```
<web-app 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_3_0.xsd"
    version="3.0">

 <module-name>customer-portal</module-name>

   <filter>
      <filter-name>Keycloak Filter</filter-name>
      <filter-class>org.keycloak.adapters.saml.servlet.SamlFilter</filter-class>
   </filter>
   <filter-mapping>
      <filter-name>Keycloak Filter</filter-name>
      <url-pattern>/*</url-pattern>
   </filter-mapping>
</web-app>
```

Red Hat Single Sign-On 过滤器具有与与其他适配器相同的配置参数，但您必须将它们定义为 filter init 参数，而不是上下文参数。

如果您有不同的安全和不安全的 url 模式，可以定义多个过滤器映射。

警告

您必须有一个涵盖 **/saml** 的过滤器映射。此映射涵盖了所有服务器回调。

当将 SP 与 IdP 注册时，您必须将 **http[s]://hostname/{context-root}/saml** 注册为 Assert Consumer Service URL 和 Single Logout Service URL。

要使用此过滤器，请在 WAR poms 中包括这个 maven 工件：

```
<dependency>
  <groupId>org.keycloak</groupId>
  <artifactId>keycloak-saml-servlet-filter-adapter</artifactId>
```

```
    <version>18.0.14.redhat-00001</version>
</dependency>
```

要使用 Multi Tenancy，**keycloak.config.resolver** 参数应作为过滤器参数传递。

```
<filter>
    <filter-name>Keycloak Filter</filter-name>
    <filter-class>org.keycloak.adapters.saml.servlet.SamlFilter</filter-class>
    <init-param>
        <param-name>keycloak.config.resolver</param-name>
        <param-value>example.SamlMultiTenantResolver</param-value>
    </init-param>
</filter>
```

3.1.7. 使用身份提供程序注册

对于每个基于 servlet 的适配器，您注册了 assert 使用者服务 URL 的端点必须是 servlet 应用程序的基本 URL，附加有 **/saml**，即 **https://example.com/contextPath/saml**。

3.1.8. 退出

您可以通过多种方式从 web 应用程序注销。对于 Jakarta EE servlet 容器，您可以调用 **HttpServletRequest.logout（）**。对于任何其他浏览器应用程序，您可以将浏览器指向具有安全约束的 Web 应用的任何 url，并传递查询参数 GLO，例如 **http://myapp?GLO=true**。如果您通过浏览器具有 SSO 会话，这将注销。

3.1.8.1. 在集群环境中注销

在内部，SAML 适配器会在 SAML 会话索引、主体名称（已知）和 HTTP 会话 ID 之间存储映射。此映射可以在 JBoss 应用服务器系列（WildFly 10/11、EAP 6/7）中维护，以供分布的应用程序使用。作为条件，需要在应用程序的 **web.xml** 中使用 <distribut/> 标签分发 HTTP 会话（例如，应用程序标记为 **<distriable/>** 标签）。

要启用功能，请在 **/WEB_INF/web.xml** 文件中添加以下部分：

对于 EAP 7，WildFly 10/11：

```
<context-param>
    <param-name>keycloak.sessionIdMapperUpdater.classes</param-name>
    <param-
value>org.keycloak.adapters.saml.wildfly.infinispan.InfinispanSessionCacheIdMapperUpdater</para
m-value>
</context-param>
```

对于 EAP 6：

```
<context-param>
    <param-name>keycloak.sessionIdMapperUpdater.classes</param-name>
    <param-
value>org.keycloak.adapters.saml.jbossweb.infinispan.InfinispanSessionCacheIdMapperUpdater</pa
ram-value>
</context-param>
```

如果部署的会话缓存命名为 **deployment-cache**，用于 SAML 映射的缓存将命名为 **deployment-**

cache.ssoCache。缓存名称可以被 context 参数 **keycloak.sessionIdMapperUpdater.infinispan.cacheName** 覆盖。包含缓存的 cache 容器与包含部署会话缓存的 cache 容器相同，但可以被上下文参数 **keycloak.sessionIdMapperUpdater.infinispan.containerName** 覆盖。

默认情况下，SAML 映射缓存的配置将派生自会话缓存。可以手动覆盖与其它缓存相同的服务器的缓存配置部分。

目前，为了提供可靠的服务，建议将复制缓存用于 SAML 会话缓存。使用分布式缓存可能会导致 SAML 注销请求进入没有访问 SAML 会话索引到 HTTP 会话映射的节点，这会导致无法注销。

3.1.8.2. 退出跨站点情境

跨站点方案仅适用于 WildFly 10 和更高版本，且 EAP 7 及更高版本。

处理跨越多个数据中心的会话需要特殊处理。考虑以下情境：

1. 登录请求在数据中心 1 中处理。
2. 管理员发出特定 SAML 会话的注销请求，请求进入数据中心 2。

数据中心 2 必须注销数据中心 1 中（以及共享 HTTP 会话的所有其他数据中心）存在的所有会话。

要涵盖这种情况，上述 SAML 会话缓存只需要在单个集群中复制，而是在所有数据中心复制，例如通过独立 Infinispan/JDG 服务器：

1. 缓存必须添加到独立的 Infinispan/JDG 服务器中。
2. 必须将上一项中的缓存添加为对应 SAML 会话缓存的远程存储。

在部署期间找到了远程存储以在 SAML 会话缓存中存在后，则会监视更改，并相应地更新本地 SAML 会话缓存。

3.1.9. 获取断言属性

成功 SAML 登录后，您的应用程序代码可能希望获取通过 SAML 断言传递的属性值。**HttpServletRequest.getUserPrincipal（）** 返回 **可进入** Red Hat Single Sign-On 特定类（名为 **org.keycloak.adapters.saml.SamlPrincipal**）的主体对象。此对象允许您查看原始的断言，以及便利函数来查找属性值。

```
package org.keycloak.adapters.saml;

public class SamlPrincipal implements Serializable, Principal {
    /**
     * Get full saml assertion
     *
     * @return
     */
    public AssertionType getAssertion() {
        ...
    }

    /**
     * Get SAML subject sent in assertion
     *
     * @return
```

```
     */
    public String getSamlSubject() {
        ...
    }

    /**
     * Subject nameID format
     *
     * @return
     */
    public String getNameIDFormat() {
        ...
    }

    @Override
    public String getName() {
        ...
    }

    /**
     * Convenience function that gets Attribute value by attribute name
     *
     * @param name
     * @return
     */
    public List<String> getAttributes(String name) {
        ...

    }

    /**
     * Convenience function that gets Attribute value by attribute friendly name
     *
     * @param friendlyName
     * @return
     */
    public List<String> getFriendlyAttributes(String friendlyName) {
        ...
    }

    /**
     * Convenience function that gets first  value of an attribute by attribute name
     *
     * @param name
     * @return
     */
    public String getAttribute(String name) {
        ...
    }

    /**
     * Convenience function that gets first  value of an attribute by attribute name
     *
     *
     * @param friendlyName
     * @return
```

```
     */
    public String getFriendlyAttribute(String friendlyName) {
        ...
    }

    /**
     * Get set of all assertion attribute names
     *
     * @return
     */
    public Set<String> getAttributeNames() {
        ...
    }

    /**
     * Get set of all assertion friendly attribute names
     *
     * @return
     */
    public Set<String> getFriendlyNames() {
        ...
    }
}
```

3.1.10. 错误处理

Red Hat Single Sign-On 对基于 servlet 的客户端适配器有一些错误处理功能。在身份验证中遇到错误时，客户端适配器将调用 **HttpServletResponse.sendError（）**。您可以在 **web.xml** 文件中设置一个 **错误页面** 来处理错误。客户端适配器可能会抛出 400、401、403 和 500 错误。

```
<error-page>
    <error-code>403</error-code>
    <location>/ErrorHandler</location>
</error-page>
```

客户端适配器还设置可检索的 **HttpServletRequest** 属性。属性名称是 **org.keycloak.adapters.spi.AuthenticationError**。Typecast this object to: **org.keycloak.adapters.saml.SamlAuthenticationError**.此类可以告诉您发生的情况。如果没有设置此属性，则适配器不负责错误代码。

```
public class SamlAuthenticationError implements AuthenticationError {
    public static enum Reason {
        EXTRACTION_FAILURE,
        INVALID_SIGNATURE,
        ERROR_STATUS
    }

    public Reason getReason() {
        return reason;
    }
    public StatusResponseType getStatus() {
        return status;
    }
}
```

3.1.11. 故障排除

对问题进行故障排除的最佳方式是在客户端适配器和 Red Hat Single Sign-On Server 中打开 SAML 的调试。使用您的日志记录框架，将 **org.keycloak.saml** 软件包的日志级别设置为 **DEBUG**。启用这个端口可让您看到 SAML 请求和响应文档发送到服务器以及从服务器发送。

3.1.12. 多租户

SAML 提供与 Multi Tenancy 相同的功能，这意味着单个目标应用程序(WAR) 可以使用多个 Red Hat Single Sign-On 域进行保护。域可以位于同一红帽单点登录实例或不同的实例上。

要做到这一点，应用程序必须有多个 **keycloak-saml.xml** 适配器配置文件。

虽然您可以有多个 WAR 实例，但不同的适配器配置文件部署到不同的上下文路径，但这可能很不便，您可能还希望根据上下文路径以外的某些域选择域。

Red Hat Single Sign-On 使可以有一个自定义配置解析器，以便您可以选择每个请求使用哪些适配器配置。在 SAML 中，配置只在登录处理中非常感兴趣；一旦登录，会话会被身份验证，并且如果返回了 **keycloak-saml.xml** 并不重要。因此，为相同会话返回相同的配置是到达的正确方法。

为达成此目标，可创建 **org.keycloak.adapters.saml.SamlConfigResolver** 的实施。以下示例使用 **Host** 标头来定位正确的配置并加载它以及应用程序的 Java 类路径中的关联元素：

```
package example;

import java.io.InputStream;
import org.keycloak.adapters.saml.SamlConfigResolver;
import org.keycloak.adapters.saml.SamlDeployment;
import org.keycloak.adapters.saml.config.parsers.DeploymentBuilder;
import org.keycloak.adapters.saml.config.parsers.ResourceLoader;
import org.keycloak.adapters.spi.HttpFacade;
import org.keycloak.saml.common.exceptions.ParsingException;

public class SamlMultiTenantResolver implements SamlConfigResolver {

    @Override
    public SamlDeployment resolve(HttpFacade.Request request) {
        String host = request.getHeader("Host");
        String realm = null;
        if (host.contains("tenant1")) {
            realm = "tenant1";
        } else if (host.contains("tenant2")) {
            realm = "tenant2";
        } else {
            throw new IllegalStateException("Not able to guess the keycloak-saml.xml to load");
        }

        InputStream is = getClass().getResourceAsStream("/" + realm + "-keycloak-saml.xml");
        if (is == null) {
            throw new IllegalStateException("Not able to find the file /" + realm + "-keycloak-saml.xml");
        }

        ResourceLoader loader = new ResourceLoader() {
            @Override
            public InputStream getResourceAsStream(String path) {
                return getClass().getResourceAsStream(path);
```

```
            }
        };

        try {
            return new DeploymentBuilder().build(is, loader);
        } catch (ParsingException e) {
            throw new IllegalStateException("Cannot load SAML deployment", e);
        }
    }
}
```

*您还必须配置在 **web.xml** 中用于 **keycloak.config.resolver** context-param 的 **SamlConfigResolver** 实现：*

```
<web-app>
    ...
    <context-param>
        <param-name>keycloak.config.resolver</param-name>
        <param-value>example.SamlMultiTenantResolver</param-value>
    </context-param>
</web-app>
```

3.2. MOD_AUTH_MELLON APACHE HTTPD MODULE

mod_auth_mellon 模块是 SAML 的 Apache HTTPD 插件。如果您的语言/环境支持使用 Apache HTTPD 作为代理，则可使用 mod_auth_mellon 来使用 SAML 保护 Web 应用程序。有关此模块的详情请查看 mod_auth_mellon GitHub 仓库。

要配置 mod_auth_mellon，您需要：

- Identity Provider(IdP)实体描述符 XML 文件，描述与 Red Hat Single Sign-On 或其他 SAML IdP 的连接
- SP 实体描述符 XML 文件，描述您要保护的应用程序的 SAML 连接和配置。
- 私钥 PEM 文件，它是 PEM 格式的文本文件，用于定义应用用于签名文档的私钥。
- 证书 PEM 文件，这是为应用程序定义证书的文本文件。
- mod_auth_mellon 特定的 Apache HTTPD 模块配置。

3.2.1. 使用红帽单点登录配置 mod_auth_mellon

涉及两个主机：

- 运行 Red Hat Single Sign-On 的主机，它被称为 $idp_host，因为红帽单点登录是一个 SAML 身份提供程序(IdP)。
- 运行 web 应用的主机，其称为 $sp_host。在使用 IdP 的 SAML 应用程序中称为服务提供商(SP)。

以下所有步骤需要在带有 root 权限的 $sp_host 上执行。

3.2.1.1. 安装软件包

要安装所需的软件包，您需要：

- Apache Web 服务器(httpd)
- Apache 的 Mellon SAML SP 附加组件模块
- 创建 X509 证书的工具

要安装所需的软件包，请运行以下命令：

```
yum install httpd mod_auth_mellon mod_ssl openssl
```

3.2.1.2. 为 Apache SAML 创建配置目录

建议把配置文件与 Apache 使用 SAML 保存在一个位置。

在 Apache 配置根 /etc/httpd 下创建名为 saml2 的新目录：

```
mkdir /etc/httpd/saml2
```

3.2.1.3. 配置 Mellon Service Provider

Apache 附加组件模块的配置文件位于 /etc/httpd/conf.d 目录中，文件名扩展名为 .conf。您需要创建 /etc/httpd/conf.d/mellon.conf 文件，并在其中放置 Mellon 的配置指令。

Mellon 的配置指令可以大致分为两类信息：

- 要使用 SAML 身份验证保护的 URL
- 在引用受保护的 URL 时，将使用什么 SAML 参数。

Apache 配置指令通常遵循 URL 空间中的分级树结构，它们称为位置。您需要为 Mellon 指定一个或多个 URL 位置来保护。您在添加适用于每个位置的配置参数方面具有灵活性。您可以将所有必要的参数添加到位置块，或者您可以将 Mellon 参数添加到特定保护位置继承的 URL 位置分级（或这两者的某些组合）中。由于 SP 通常会以相同的方式运行：无论哪个位置触发 SAML 操作，因此这里使用的示例配置指令放在层次结构根中，然后使用 Mellon 对特定位置进行保护，那么可以使用最小的指令来定义特定位置。此策略可避免为每个受保护的位置重复相同的参数。

这个示例只有一个受保护的位置：https://$sp_host/private。

要配置 Mellon 服务提供程序，请执行以下步骤。

流程

1. 创建包含以下内容的文件 /etc/httpd/conf.d/mellon.conf：

```
<Location / >
  MellonEnable info
  MellonEndpointPath /mellon/
  MellonSPMetadataFile /etc/httpd/saml2/mellon_metadata.xml
  MellonSPPrivateKeyFile /etc/httpd/saml2/mellon.key
  MellonSPCertFile /etc/httpd/saml2/mellon.crt
  MellonIdPMetadataFile /etc/httpd/saml2/idp_metadata.xml
</Location>
<Location /private >
  AuthType Mellon
```

```
    MellonEnable auth
    Require valid-user
</Location>
```

注意

上述代码中引用的某些文件将在后续步骤中创建。

3.2.2. 为 mod_auth_mellon 使用的 cookie 设置 SameSite 值

浏览器计划将 Cookie 的 **SameSite** 属性的默认值设置为 **Lax**。此设置表示，仅当请求源自于同一域中时，cookies 才会发送到应用程序。这个行为可能会影响 SAML POST 绑定，这可能会变为无法正常工作。要保留 mod_auth_mellon 模块的完整功能，我们建议将 mod_auth_mellon 创建的 cookie 的 **SameSite** 值设置为 **None**。不这样做可能会导致无法使用 Red Hat Single Sign-On 进行登录。

要将 **SameSite** 值设为 **None**，请将以下配置添加到 **mellon.conf** 文件中的 **<Location** /> tag。

```
MellonSecureCookie On
MellonCookieSameSite none
```

对此配置的支持在 0.16.0 版的 mod_auth_mellon 模块中可用。

3.2.2.1. 创建服务提供商元数据

在 SAML IdP 和 SP 交换 SAML 元数据中，采用 XML 格式。元数据的 schema 是标准的，因此相关的 SAML 实体可以互相使用其元数据。您需要：

- SP 使用的 IdP 元数据
- 描述提供给 IdP 的 SP 元数据

SAML 元数据的一个组件是 X509 证书。这些证书用于两个目的：

- 为 SAML 信息签名，以便接收最终能够证明该消息源自预期方。
- 在传输期间加密消息（这些ldom 使用，因为 SAML 信息通常在 TLS 保护的传输中发生）

如果您已有证书颁发机构(CA)，或者您可以生成自签名证书，您可以使用自己的证书。为了便于本例中使用自签名证书。

因为 Mellon 的 SP 元数据必须反映安装的 mod_auth_mellon 版本的功能，所以必须是有效的 SP 元数据 XML，并且必须包含 X509 证书（除非熟悉了 X509 证书）,以便生成 SP 元数据，以便使用在 mod_auth_llme_llme_onme_onme_onme_onme_on_onme_onme_on_on_on_on_on_on_on_on_create_on_ 生成的元数据总是可以被编辑，因为它是一个文本文件。该工具还创建您的 X509 密钥和证书。

SAML IdP 和 SP 使用唯一名称标识自己，称为 EntityID。要使用 Mellon 元数据创建工具：

- EntityID（通常是 SP 的 URL），通常是 SP 的 URL，通常是可以检索 SP 元数据的 URL
- 用于 SP 的 SAML 消息的 URL 将被消耗，Mononon 调用 MellonEndPointPath。

要创建 SP 元数据，请执行以下步骤。

流程

1. 创建几个 helper shell 变量：

```
fqdn=`hostname`
mellon_endpoint_url="https://${fqdn}/mellon"
mellon_entity_id="${mellon_endpoint_url}/metadata"
file_prefix="$(echo "$mellon_entity_id" | sed 's/[^A-Za-z.]/_/g' | sed 's/__*/_/g')"
```

2. 运行以下命令调用 Mellon 元数据创建工具：

```
/usr/libexec/mod_auth_mellon/mellon_create_metadata.sh $mellon_entity_id
$mellon_endpoint_url
```

3. 将生成的文件移动到其目的地（参考在上面创建的 /etc/httpd/conf.d/mellon.conf 文件中）：

```
mv ${file_prefix}.cert /etc/httpd/saml2/mellon.crt
mv ${file_prefix}.key /etc/httpd/saml2/mellon.key
mv ${file_prefix}.xml /etc/httpd/saml2/mellon_metadata.xml
```

3.2.2.2. 将 Mellon Service Provider 添加到红帽单点登录身份提供程序

假设：Red Hat Single Sign-On IdP 已在 $idp_host 上安装。

Red Hat Single Sign-On 支持将所有用户、客户端等多个租户分组到称为域的域中。每个 realm 都是相互独立的。您可以在红帽单点登录中使用现有域，但本例演示了如何创建名为 test_realm 的新域，并使用该域。

所有这些操作都使用红帽单点登录管理控制台执行。您必须具有 $idp_host 的 admin 用户名和密码才能执行以下步骤。

流程

1. 打开 Admin Console 并通过输入 admin 用户名和密码登录。
   登录 Admin 控制台后，会有一个现有的 realm。当首先设置 Red Hat Single Sign-On 时，会创建根域 master。所有之前创建的域都在下拉列表中的 Admin Console 左上角列出。
2. 从 realm 下拉列表中，选择 **Add realm**。
3. 在 Name 字段中，键入 **test_realm**，再点击 **Create**。

3.2.2.2.1. 添加 Mellon Service Provider 作为域的客户端

在红帽单点登录 SAML SP 中，称为客户端。要添加 SP，我们必须在域的 Clients 部分中。

1. 单击左侧的 Clients 菜单项，再单击右上角的 **Create** 来创建新客户端。

3.2.2.2.2. 添加 Mellon SP 客户端

要添加 Mellon SP 客户端，请执行以下步骤。

流程

1. 将客户端协议设置为 SAML。
2. 从 Client Protocol 下拉列表中选择 **saml**。

3. 提供上述创建的 Mellon SP 元数据文件(/etc/httpd/saml2/mellon_metadata.xml)。
   根据您的浏览器运行位置，您可能需要将 SP 元数据从 $sp_host 复制到运行浏览器的计算机，以便浏览器可以找到该文件。

4. 点击 **Save**。

3.2.2.2.3. 编辑 Mellon SP 客户端

使用这个流程设置重要的客户端配置参数。

流程

1. 确保"Force POST Binding"为 On。
2. 将 paosResponse 添加到 Valid Redirect URIs 列表中：
3. 复制"Valid Redirect URI"中的 postResponse URL，并将它粘贴到"+"下面的空白添加文本字段。
4. 将"postResponse"更改为"paosResponse"。（SAML ECP 需要 paosResponse URL。）
5. 点底部的 **Save**。

许多 SAML SP 根据用户在组中的成员资格来确定授权。Red Hat Single Sign-On IdP 可以管理用户组信息，除非 IdP 配置为为其提供 SAML 属性，否则不会为用户提供组。

执行以下步骤将 IdP 配置为将用户组作为 SAML 属性提供。

流程

1. 点客户端的映射程序选项卡。
2. 在映射程序页面的右上角，点 **Create**。
3. 从 Mapper Type 下拉列表中选择 **组列表**。
4. 将 Name 设置为 "group list"。
5. 将 SAML 属性名称设置为 "groups"。
6. 点击 **Save**。

剩余的步骤在 $sp_host 上执行。

3.2.2.2.4. 检索身份提供程序元数据

现在，您已在 IdP 中创建域，您需要检索与其关联的 IdP 元数据，以便 Mellon SP 可以识别它。在之前创建的 /etc/httpd/conf.d/mellon.conf 文件中，MellonIdPMetadataFile 指定为 /etc/httpd/saml2/idp_metadata.xml，直到现在该文件在 $sp_host 上不存在。

使用这个流程从 IdP 检索该文件。

流程

1. 使用这个命令，用 $idp_host 的正确值替换：

```
curl -k -o /etc/httpd/saml2/idp_metadata.xml \
https://$idp_host/auth/realms/test_realm/protocol/saml/descriptor
```

Mellon 现在被完全配置。

2. 要对 Apache 配置文件运行语法检查，请使用以下命令：

```
apachectl configtest
```

注意

configtest 等同于 apachectl 的 -t 参数。如果配置测试显示任何错误，请在继续操作前进行更正。

3. 重启 Apache 服务器：

```
systemctl restart httpd.service
```

现在，您已将 Red Hat Single Sign-On 设置为 test_realm 和 mod_auth_mellon 中的 SAML IdP，作为 SAML SP 保护 URL $sp_host/ Protection（以及它下的所有内容）与 **$idp_host** IdP 进行身份验证。

第 4 章 配置 DOCKER 注册表以使用红帽单点登录

注意

Docker 身份验证默认为禁用。启用查看 配置集。

本节论述了如何配置 Docker 注册表，以使用 Red Hat Single Sign-On 作为其身份验证服务器。

有关如何设置和配置 Docker 注册表的更多信息，请参阅 Docker 注册表配置指南。

4.1. DOCKER REGISTRY 配置文件安装

对于具有更高级 Docker registry 配置的用户，通常建议提供自己的 registry 配置文件。Red Hat Single Sign-On Docker 供应商通过 Registry 配置文件格式选项支持这种 机制。选择这个选项将生成类似如下的输出：

```
auth:
  token:
    realm: http://localhost:8080/auth/realms/master/protocol/docker-v2/auth
    service: docker-test
    issuer: http://localhost:8080/auth/realms/master
```

然后可将这个输出复制到任何现有的 registry 配置文件中。如需有关如何设置该文件或以 基本示例 开始的更多信息，请参阅 registry 配置文件规格。

警告

别忘了为 **rootcertbundle** 字段配置 Red Hat Single Sign-On 域的公钥的位置。auth 配置在没有这个参数的情况下将无法工作。

4.2. DOCKER REGISTRY 环境变量覆盖安装

通常，最好使用简单的环境变量覆盖开发或 POC Docker 注册表。虽然此方法通常不建议用于生产环境，但当一个需要快速和 日志记录的方式来备份 registry 时，会很有帮助。使用客户端安装选项卡中的 Variable Override 格式选项，输出应类似于下面的内容：

```
REGISTRY_AUTH_TOKEN_REALM: http://localhost:8080/auth/realms/master/protocol/docker-v2/auth
REGISTRY_AUTH_TOKEN_SERVICE: docker-test
REGISTRY_AUTH_TOKEN_ISSUER: http://localhost:8080/auth/realms/master
```

警告

别忘了通过 Red Hat Single Sign-On 域的公钥配置 **REGISTRY_AUTH_TOKEN_ROOTCERTBUNDLE** 覆盖的 REGISTRY_AUTH_TOKEN_ROOTCERTBUNDLE 覆盖。auth 配置在没有这个参数的情况下将无法工作。

4.3. DOCKER 编译 YAML 文件

警告

这个安装方法旨在通过一种简单的方法使 docker 注册表与红帽单点登录服务器进行身份验证。它仅用于开发目的，不应用于生产环境或类似于生产环境的环境。

zip 文件安装机制为希望了解红帽单点登录服务器如何与 Docker 注册表交互的开发人员提供快速入门。配置：

流程

1. 从所需的域，创建客户端配置。此时，您不会有 Docker 注册表 - 快速入门将负责这一部分。
2. 从安装选项卡中选择"Docker Compose YAML"选项，并下载 .zip 文件
3. 将存档解压缩到所需位置，再打开 目录。
4. 使用 **docker-compose** 启动Docker registry

注意

建议您在 'master' 以外的域中配置 Docker registry 客户端，因为 HTTP Basic auth 流将不存在表单。

进行了上述配置后，密钥和 Docker registry 正在运行，docker 身份验证可以成功：

```
[user ~]# docker login localhost:5000 -u $username
Password: *******
Login Succeeded
```

第 5 章 使用客户端注册服务

要让应用程序或服务使用 Red Hat Single Sign-On，需要在 Red Hat Single Sign-On 中注册客户端。管理员可以通过管理控制台（或管理员 REST 端点）进行此操作，但客户端也可通过红帽单点登录客户端注册服务来注册。

客户端注册服务为红帽单点登录客户端代表、OpenID Connect Client Meta Data 和 SAML Entity Descriptors 提供内置支持。客户端注册服务端点为 **/auth/realms/<realm>/clients-registrations/<provider>**。

内置支持的供应商有：

- Default - Red Hat Single Sign-On Client Representation(JSON)
- install - Red Hat Single Sign-On Adapter Configuration(JSON)
- OpenID-connect - OpenID Connect 客户端元数据描述(JSON)
- saml2-entity-descriptor - SAML Entity Descriptor(XML)

以下小节将介绍如何使用不同的提供程序。

5.1. 身份验证

要调用客户端注册服务，通常需要令牌。令牌可以是 bearer 令牌、初始访问令牌或注册访问令牌。另外，还有在没有任何令牌的情况下注册新客户端的替代方法，但您需要配置客户端注册策略（请参阅以下）。

5.1.1. bearer 令牌

bearer 令牌可以代表用户或服务帐户发布。调用端点需要以下权限（请参阅 "服务器管理指南" 以了解更多详细信息）：

- create-client 或 manage-client - 要创建客户端
- view-client 或 manage-client - 要查看客户端
- manage-client - 要更新或删除客户端

如果您使用 bearer 令牌来创建客户端，建议仅使用具有 **create-client** 角色的服务帐户的令牌（详情请参阅 服务器管理指南）。

5.1.2. 初始访问令牌

注册新客户端的建议方法是使用初始访问令牌。初始访问令牌只能用于创建客户端，并且具有可配置的到期时间，以及可以创建客户端数的可配置限制。

通过 admin 控制台可以创建初始访问令牌。要创建新的初始访问令牌，首先在 admin 控制台中选择域，然后单击左侧菜单中的 **Realm Settings**，然后在页面中显示的标签页里单击 **Client Registration**。最后，单击 **Initial Access Tokens** 子选项卡。

现在，您可以看到任何现有的初始访问令牌。如果您有访问权限，可以删除不再需要的令牌。您只能在创建令牌时检索令牌值。要创建新令牌，请点 **Create**。现在，您可以选择性地添加令牌有效的时长，以及可以使用令牌创建多少个客户端。在点 **Save** the token 值后会显示。

现在，复制/粘贴此令牌非常重要，因为您将无法在以后检索它。如果您忘记复制/粘贴，请删除令牌并创建令牌。

在调用客户端注册服务时，令牌值用作标准 bearer 令牌，方法是将其添加到请求的 Authorization 标头中。例如：

```
Authorization: bearer eyJhbGciOiJSUz...
```

5.1.3. 注册访问令牌

当您通过客户端注册服务创建客户端时，响应将包含注册访问令牌。通过注册访问令牌，您可以在以后检索客户端配置，但也可更新或删除客户端。注册访问令牌会包含在请求中，其方式与 bearer 令牌或初始访问令牌相同。注册访问令牌只有效一次，使用响应时将包括新令牌。

如果在客户端注册服务之外创建客户端，它不会关联注册访问令牌。您可以通过 admin 控制台创建一个。如果您丢失了特定客户端的令牌，这也很有用。要在 admin 控制台中创建新令牌，并点击 **Credentials**。然后点击 **生成注册访问令牌**。

5.2. RED HAT SINGLE SIGN-ON 代表

默认的 客户端注册提供程序可用于创建、检索、更新和删除客户端。它使用 Red Hat Single Sign-On Client Representation 格式，它可根据配置客户端的支持，这可以通过管理控制台进行配置，例如配置协议映射程序。

要创建客户端代表(JSON)，然后对 **/auth/realms/<realm>/clients-registrations/default** 执行 HTTP POST 请求。

它将返回一个包括注册访问令牌的 Client Representation。如果要检索配置、更新或删除客户端，您应在一个地方保存注册访问令牌。

检索客户端代表执行 HTTP GET 请求到 **/auth/realms/<realm>/clients-registrations/default/<client id&** gt;。

它还会返回新的注册访问令牌。

要更新客户端代表，使用更新的客户端代表执行 HTTP PUT 请求： **/auth/realms/<realm>/clients-registrations/default/<client id&** gt;。

它还会返回新的注册访问令牌。

要删除客户端代表执行 HTTP DELETE 请求，请执行以下操作： **/auth/realms/<realm>/clients-registrations/default/<client id>**

5.3. RED HAT SINGLE SIGN-ON ADAPTER 配置

安装 客户端注册提供程序可用于检索客户端的适配器配置。除了令牌身份验证外，您还可以使用 HTTP 基本身份验证与客户端凭据进行身份验证。要做到这一点，请在请求中包含以下标头：

```
Authorization: basic BASE64(client-id + ':' + client-secret)
```

要检索适配器配置，然后对 **/auth/realms/<realm>/clients-registrations/install/<client id** > 执行 HTTP GET 请求。

公共客户端不需要身份验证。这意味着，对于 JavaScript 适配器，您可以使用以上 URL 直接从 Red Hat Single Sign-On 加载客户端配置。

5.4. OPENID CONNECT DYNAMIC CLIENT REGISTRATION

Red Hat Single Sign-On 实施 OpenID Connect Dynamic Client Registration，它扩展了 OAuth 2.0 Dynamic Client Registration Protocol 和 OAuth 2.0 Dynamic Client Registration Management Protocol。

在 Red Hat Single Sign-On 中注册客户端的端点是 **/auth/realms/<realm>/clients-registrations/openid-connect[/<client id>]**。

此端点也可以在域的 OpenID Connect Discovery 端点 **/auth/realms/<realm>/.well-known/openid-configuration** 中找到。

5.5. SAML 实体描述符

SAML 实体描述符端点仅支持使用 SAML v2 实体描述符来创建客户端。它不支持检索、更新或删除客户端。对于那些操作，应使用 Red Hat Single Sign-On 表示端点。在创建红帽单点登录客户端时，会返回包含创建客户端的详情，包括注册访问令牌的详情。

使用 SAML Entity Descriptor 创建客户端对 **/auth/realms/<realm>/clients-registrations/saml2-entity-descriptor** 来执行 HTTP POST 请求。

5.6. 使用 CURL 的示例

以下示例使用 CURL 使用 clientId **myclient** 创建客户端。您需要将 **eyJhbGciOiJSUz...** 替换为正确的初始访问令牌或 bearer 令牌。

```
curl -X POST \
    -d '{ "clientId": "myclient" }' \
    -H "Content-Type:application/json" \
    -H "Authorization: bearer eyJhbGciOiJSUz..." \
    http://localhost:8080/auth/realms/master/clients-registrations/default
```

5.7. 使用 JAVA 客户端注册 API 的示例

客户端注册 Java API 可让您轻松使用 Java 进行客户端注册服务。要使用的依赖项 **org.keycloak:keycloak-client-registration-api:>VERSION&** lt;。

有关使用客户端注册的完整说明，请参阅 JavaDocs。以下是创建客户端的示例。您需要将 **eyJhbGciOiJSUz...** 替换为正确的初始访问令牌或 bearer 令牌。

```
String token = "eyJhbGciOiJSUz...";

ClientRepresentation client = new ClientRepresentation();
client.setClientId(CLIENT_ID);

ClientRegistration reg = ClientRegistration.create()
    .url("http://localhost:8080/auth", "myrealm")
    .build();

reg.auth(Auth.token(token));
```

```
client = reg.create(client);

String registrationAccessToken = client.getRegistrationAccessToken();
```

5.8. 客户端注册策略

注意

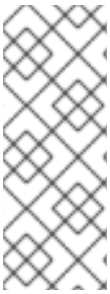
当前计划用于删除客户端注册策略，以替代《服务器管理指南》中描述的客户端策略。客户端策略更为灵活，支持更多用例。

Red Hat Single Sign-On 目前支持通过客户端注册服务注册新客户端的方法。

- 已验证的 requests - 注册新客户端的请求必须包含前面提到的 **Initial Access Token** 或 **Bearer Token**。
- 匿名请求 - 要注册新客户端的请求根本不需要包含任何令牌

匿名客户端注册请求非常有趣的功能，但您通常不希望任何人在没有限制的情况下注册新客户端。因此，我们有 **客户端注册策略 SPI** 提供了一种限制谁可以在哪些条件下注册新客户端的方法。

在 Red Hat Single Sign-On admin 控制台中，您可以点击 **Client Registration** 标签页，然后单击 **Client Registration Policies** 子tab。在这里，您将看到为匿名请求以及针对身份验证请求配置了哪些策略。



注意

仅允许新客户端创建（注册）匿名请求（请求，任何令牌）。因此，当您通过匿名请求注册新客户端时，响应将包含 Registration Access Token，该令牌必须用于特定客户端读取、更新或删除请求。但是，从匿名注册中使用此注册访问令牌将同样受到匿名策略的影响！这意味着，如果您有受信任的主机策略，对于更新客户端的请求也需要来自 **受信任的主机**。例如，更新客户端以及存在 **Consent Required** 策略时，不允许禁用 **Consent 所需**策略。

目前，我们有这些策略实现：

- 可信主机策略 - 您可以配置可信主机和可信域列表。只能从这些主机或域发送客户端注册服务请求。从某些不受信任的 IP 发送的请求将被拒绝。新注册客户端的 URL 也必须只使用这些可信主机或域。例如，不允许设置指向某些不可信主机的客户端的 **重定向 URI**。默认情况下，没有任何白名单的主机，因此匿名客户端注册被禁用。
- 批准策略 - 新注册的客户端将启用 **Consent Allowed** 开关。因此，在成功身份验证后，当他需要批准权限（客户端范围）时，用户将始终看到同意屏幕。这意味着，除非用户批准，客户端将无权访问用户的任何个人信息或权限。
- 协议映射程序策略 - 允许配置白名单协议映射程序实施列表。如果新客户端包含一些非列出的协议映射程序，则无法注册或更新。请注意，这个策略也用于经过身份验证的请求，因此即使经过身份验证的请求也有一些限制，可以使用协议映射程序。
- 客户端范围策略 - 允许将 **客户端范围** 列入白名单，可用于新注册或更新的客户端。默认没有白名单范围；只有客户端范围（定义为 **Realm Default Client Scopes**）默认列在白名单中。
- 完整的范围策略 - 注册的客户端将禁用 **全范围允许的** 交换机。这意味着它们没有其他客户端的任何有范围的域角色或客户端角色。

- *最大客户端策略 - 如果域中当前客户端数量比指定限制相同或大于指定限制，则拒绝注册。对于匿名注册，默认为 200。*
- *客户端禁用策略 - 新注册的客户端将被禁用。这意味着，admin 需要手动批准并启用所有新注册的客户端。即使匿名注册，这个策略也不使用。*

第 6 章 使用 CLI 自动注册客户端

客户端注册 CLI 是一个命令行界面(CLI)工具，供应用程序开发人员在与红帽单点登录集成时以自助服务方式配置新客户端。它旨在与红帽单点登录客户端注册 REST 端点交互。

需要为任何应用程序创建或获取客户端配置，才能使用 Red Hat Single Sign-On。您通常会为托管于唯一主机名的每个新应用程序配置新客户端。当应用程序与红帽单点登录交互时，应用会标识自身带有客户端 ID，以便红帽单点登录、单点登录(SSO)会话管理和其他服务。

您可以使用客户端注册 CLI 从命令行配置应用程序客户端，您可以在 shell 脚本中使用它。

要允许特定用户使用客户端注册 **CLI**，Red Hat Single Sign-On 管理员通常使用 Admin Console 配置具有适当角色的新用户，或配置新客户端和服务器 secret 以授予客户端注册 REST API 的访问权限。

6.1. 配置一个新的常规用户，用于客户端注册 CLI

流程

1. 以 **admin** 用户身份登录管理控制台（例如 http://localhost:8080/auth/admin）。
2. 选择要管理的域。
3. 如果要使用现有用户，请选择该用户进行编辑；否则，创建新用户。
4. 选择 **Role Mappings** > **Client Roles** > **realm-management**。如果您在 master 域中，请选择 **NAME-realm**，其中 **NAME** 是目标域的名称。您可以为 master 域中的用户授予对任何其他域的访问权限。
5. 选择 **Available Roles** > **manage-client** 来授予一组完整的客户端管理权限。另一个选择是选择只读的 **view-clients** 或 **create-client** 以创建新客户端。

注意

这些权限授予用户在不使用 Initial Access Token 或 Registration Access Token 的情况下执行操作的权限。

无法将任何 **realm-management** 角色分配给用户。在这种情况下，用户仍可使用客户端注册 CLI 登录，但没有 Initial Access Token。在不使用令牌的情况下尝试执行任何操作会导致 **403 Forbidden** 错误。

管理员可以通过 **Realm Settings** > **Client Registration** > **Initial Access Token** 菜单从 Admin Console 发出 **Initial Access** Tokens。

6.2. 配置客户端以用于客户端注册 CLI

默认情况下，服务器会将客户端注册 CLI 识别为 **admin-cli** 客户端，该客户端会自动为每个新域自动配置。使用用户名登录时，不需要额外的客户端配置。

流程

1. 如果要为客户端注册 CLI 使用单独的客户端配置，请创建一个客户端（如 **reg-cli**）。
2. 将 **Standard Flow Enabled** 设置切换为 **Off**。

3. 通过将 **客户端访问类型** 配置为 **Confidential**，然后选择 Credentials > ClientId 和 Secret 来增强安全性。

注意

您可以在 Credentials 标签页下配置客户端 Id 和 Secret 或 签名 JWT。

4. 如果要使用与客户端关联的服务帐户，请在 **Admin Console** 的 Clients 部分中选择要编辑的客户端，以启用服务帐户。

   a. 在 **Settings** 下，将 **Access Type** 更改为 **Confidential**，将 **Service Accounts Enabled** 设置切换为 **On**，然后单击 **Save**。

   b. 单击 **Service Account Roles** 并选择所需的角色，以配置服务帐户的访问权限。有关要选择什么角色的详情，请参考 第 6.1 节 "配置一个新的常规用户，用于客户端注册 CLI"。

5. 如果要使用常规用户帐户而不是服务帐户，请将 **Direct Access Grants Enabled** 设置切换为 **On**。

6. 如果客户端被配置为 **Confidential**，请使用 **--secret** 选项运行 **kcreg 配置凭证时提供** 配置的 secret。

7. 在运行 **kcreg config credentials** 时，指定要使用哪个 **clientId**（例如，**--client reg-cli**）。

8. 启用服务帐户后，您可以在运行 **kcreg config 凭证** 时省略指定用户，仅提供客户端 secret 或密钥存储信息。

6.3. 安装客户端注册 CLI

客户端注册 CLI 打包在红帽单点登录服务器分发中。您可以在 **bin** 目录中找到执行脚本。Linux 脚本名为 **kcreg.sh**，Windows 脚本名为 **kcreg.bat**。

设置客户端以便用于文件系统的任何位置时，将 Red Hat Single Sign-On 服务器目录添加到 **PATH** 中。

例如，在：

- Linux：

```
$ export PATH=$PATH:$KEYCLOAK_HOME/bin
$ kcreg.sh
```

- Windows：

```
c:\> set PATH=%PATH%;%KEYCLOAK_HOME%\bin
c:\> kcreg
```

KEYCLOAK_HOME 是指解包红帽单点登录服务器分发的目录。

6.4. 使用客户端注册 CLI

流程

1. 通过使用凭证登录来启动经过身份验证的会话。

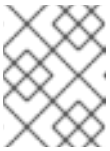
2. 在客户端注册 **REST 端点上** 运行命令。
例如，在：

- Linux：

```
$ kcreg.sh config credentials --server http://localhost:8080/auth --realm demo --user user
--client reg-cli
$ kcreg.sh create -s clientId=my_client -s 'redirectUris=["http://localhost:8980/myapp/*"]'
$ kcreg.sh get my_client
```

- Windows：

```
c:\> kcreg config credentials --server http://localhost:8080/auth --realm demo --user user
--client reg-cli
c:\> kcreg create -s clientId=my_client -s "redirectUris=[\"http://localhost:8980/myapp/*\"]"
c:\> kcreg get my_client
```



注意

在生产环境中，您必须使用 **https:** 访问 Red Hat Single Sign-On，以避免向网络嗅探器公开令牌。

3. 如果服务器的证书不是由 Java 默认证书信任存储中所含的可信证书颁发机构(CA)发布，准备信任存储 **。jks** 文件并指示客户端注册 CLI 使用它。
例如，在：

- Linux：

```
$ kcreg.sh config truststore --trustpass $PASSWORD ~/.keycloak/truststore.jks
```

- Windows：

```
c:\> kcreg config truststore --trustpass %PASSWORD%
%HOMEPATH%\.keycloak\truststore.jks
```

6.4.1. 登录

流程

1. 在使用客户端注册 CLI 登录时，指定服务器端点 URL 和域。

2. 指定一个用户名或客户端 id，这会导致使用特殊的服务帐户。使用用户名时，必须为指定用户使用密码。在使用客户端 ID 时，您可以使用客户端 secret 或 **签名 JWT** 而不是密码。

无论登录方法如何，登录的帐户需要适当的权限才能执行客户端注册操作。请记住，非主域中的任何帐户只能有一个权限管理同一域中客户端。如果需要管理不同的域，您可以在不同的域中配置多个用户，也可以在 **master** 域中创建单个用户，并在不同的域中管理客户端的角色。

您不能使用客户端注册 CLI 配置用户。使用 Admin Console Web 界面或 Admin Client CLI 配置用户。如需了解更多详细信息，请参阅 服务器管理指南。

当 **kcreg** 成功登录时，它会接收授权令牌并将其保存在私有配置文件中，以便令牌可用于后续调用。有关配置文件的更多信息，请参阅 第 6.4.2 节 "使用其他配置"。

有关使用客户端注册 CLI 的详情，请查看内置的帮助。

例如，在：

- Linux：

```
$ kcreg.sh help
```

- Windows：

```
c:\> kcreg help
```

有关启动经过身份验证的会话的更多信息，请参阅 **kcreg config credentials --help**。

6.4.2. 使用其他配置

默认情况下，客户端注册 CLI 会在用户主目录下自动维护默认位置 **./.keycloak/kcreg.config** 的配置文件。您可以使用 **--config** 选项指向其他文件或位置，以并行维护多个经过身份验证的会话。从单一线程执行与单个配置文件关联的操作是最安全的方法。

重要

不要使配置文件对系统的其他用户可见。配置文件包含应保留私有的访问令牌和 secret。

您可能需要通过将 **--no-config** 选项与所有命令一起使用来避免将 secret 存储在配置文件中，即使它较方便，但需要更多令牌请求来执行。使用每个 **kcreg** 调用指定所有身份验证信息。

6.4.3. 初始访问和注册访问令牌

在他们想使用的 Red Hat Single Sign-On 服务器上未配置帐户的开发人员可使用客户端注册 CLI。只有域管理员向开发人员发出 Initial Access Token 时，才能实现。它由域管理员决定如何和何时发布这些令牌。域管理员可以限制 Initial Access Token 的最长期限，以及可使用它创建的客户端总数。

当开发人员具有 Initial Access Token 后，开发人员就可以使用它来创建新客户端，而无需向 **kcreg 配置凭证** 进行身份验证。Initial Access Token 可以存储在配置文件中，或者指定为 **kcreg create** 命令的一部分。

例如，在：

- Linux：

```
$ kcreg.sh config initial-token $TOKEN
$ kcreg.sh create -s clientId=myclient
```

或者

```
$ kcreg.sh create -s clientId=myclient -t $TOKEN
```

- Windows：

```
c:\> kcreg config initial-token %TOKEN%
c:\> kcreg create -s clientId=myclient
```

或者

```
c:\> kcreg create -s clientId=myclient -t %TOKEN%
```

在使用 Initial Access Token 时，服务器响应包括新发布的注册访问令牌。该客户端的所有后续操作都需要通过与该令牌进行身份验证来执行，这仅对该客户端有效。

客户端注册 CLI 自动使用其专用配置文件来保存并使用其关联的客户端。只要所有客户端操作都使用相同的配置文件，开发人员不需要进行身份验证以这种方式创建、更新或删除创建的客户端。

有关初始访问和注册访问令牌的更多信息，请参阅客户端注册。

运行 **kcreg config initial-token --help** 和 **kcreg config registration-token --help** 命令以了解有关如何使用 Client Registration CLI 配置令牌的更多信息。

6.4.4. 创建客户端配置

在使用凭据进行身份验证或配置初始访问令牌后，第一项任务通常是创建新客户端。通常，您可能希望使用准备的 JSON 文件作为模板，并设置或覆盖某些属性。

以下示例演示了如何读取 JSON 文件，覆盖任何客户端 id，设置任何其他属性，并在成功创建后将配置打印到标准输出。

- Linux：

```
$ kcreg.sh create -f client-template.json -s clientId=myclient -s baseUrl=/myclient -s 'redirectUris=["/myclient/*"]' -o
```

- Windows：

```
C:\> kcreg create -f client-template.json -s clientId=myclient -s baseUrl=/myclient -s "redirectUris=[\"/myclient/*\"]" -o
```

运行 **kcreg create --help** 来获取有关 **kcreg create** 命令的更多信息。

您可以使用 **kcreg attrs** 来列出可用的属性。请记住，许多配置属性没有检查有效或一致性。您需要指定适当的值。请记住，您不应具有模板中的任何 id 字段，不应将它们指定为 **kcreg create** 命令的参数。

6.4.5. 检索客户端配置

您可以使用 **kcreg get** 命令检索现有客户端。

例如，在：

- Linux：

```
$ kcreg.sh get myclient
```

- Windows：

```
C:\> kcreg get myclient
```

您还可以作为适配器配置文件检索客户端配置，该文件可以与 Web 应用程序打包。

例如，在：

- Linux：

```
$ kcreg.sh get myclient -e install > keycloak.json
```

- Windows：

```
C:\> kcreg get myclient -e install > keycloak.json
```

运行 **kcreg get --help** 命令以了解有关 **kcreg get** 命令的更多信息。

6.4.6. 修改客户端配置

更新客户端配置有两种方法。

种方法是在获取当前配置后向服务器提交完整的新状态，将其保存到文件中，编辑该文件，并将它发回到服务器。

例如，在：

- Linux：

```
$ kcreg.sh get myclient > myclient.json
$ vi myclient.json
$ kcreg.sh update myclient -f myclient.json
```

- Windows：

```
C:\> kcreg get myclient > myclient.json
C:\> notepad myclient.json
C:\> kcreg update myclient -f myclient.json
```

第二个方法获取当前客户端，设置或删除它的字段，然后在一个步骤中重新发布它。

例如，在：

- Linux：

```
$ kcreg.sh update myclient -s enabled=false -d redirectUris
```

- Windows：

```
C:\> kcreg update myclient -s enabled=false -d redirectUris
```

您还可以使用仅包含要应用的更改的文件，因此不必将过多的值指定为参数。在这种情况下，指定 **--merge** 告知客户端注册 CLI，而不是将 JSON 文件视为完整的新配置，而是应该将其视为要通过现有配置应用的一组属性。

例如，在：

- Linux：

```
$ kcreg.sh update myclient --merge -d redirectUris -f mychanges.json
```

- Windows：

```
C:\> kcreg update myclient --merge -d redirectUris -f mychanges.json
```

运行 **kcreg update --help** 命令以了解有关 **kcreg update** 命令的更多信息。

6.4.7. 删除客户端配置

使用以下示例删除客户端。

- Linux：

```
$ kcreg.sh delete myclient
```

- Windows：

```
C:\> kcreg delete myclient
```

运行 **kcreg delete --help** 命令以了解有关 **kcreg delete** 命令的更多信息。

6.4.8. 刷新无效的注册访问令牌

当使用 **--no-config** 模式执行创建、读取、更新和删除(CRUD)操作时，客户端注册 CLI 无法为您处理注册访问令牌。在这种情况下，可能会丢失最近为客户端发出的注册访问令牌，这样就无法在该客户端上执行进一步的 CRUD 操作而无需与有 **管理客户端** 权限的帐户进行身份验证。

如果您有权限，您可以为客户端发布新的注册访问令牌，并将其打印到标准输出或保存到您选择的配置文件中。否则，您需要询问域管理员为您的客户端发布新的注册访问令牌并将其发送给您。然后，您可以通过 **--token** 选项将其传递给任何 CRUD 命令。您还可以使用 **kcreg config registration-token** 命令将新令牌保存在配置文件中，并使客户端注册 CLI 会自动为您处理它。

运行 **kcreg update-token --help** 命令以了解有关 **kcreg update-token** 命令的更多信息。

6.5. 故障排除

- 问：登录时，我收到一个错误：Parameter **client_assertion_type 缺少 [invalid_client]**。
  答：这个错误表示您的客户端配置了 **签名 JWT** 令牌凭证，这意味着您必须在登录时使用 **--keystore** 参数。

第 7 章 使用令牌交换

注意

令牌交换 **是技术预览，** 它不被完全支持。此功能默认为禁用。

要使用 **-Dkeycloak.profile=preview** 或 **-Dkeycloak.profile.feature.token_exchange=enabled** 来启用服务器启动。如需了解更多详细信息，请参阅 配置文件。

注意

要使用令牌交换，您还应启用 **token_exchange** 功能。请查看 配置文件。

7.1. 令牌交换如何工作

在红帽单点登录中，令牌交换是使用一组凭证或令牌集的过程，以获取完全不同的令牌。客户端可能想在不太可信的应用程序上调用，以便它希望降级其具有的当前令牌。客户端可能想要为已链接的社交帐户存储的令牌交换红帽单点登录令牌。您可能想信任被其他红帽单点登录域或外部 IDP 提供的外部令牌。客户端可能需要模拟用户。以下是 Red Hat Single Sign-On 令牌交换方面当前功能的简短摘要。

- 客户端可以交换为特定客户端为目标客户端创建的现有红帽单点登录令牌。
- 客户端可以为外部令牌交换现有红帽单点登录令牌，例如链接的 Facebook 帐户
- 客户端可以为红帽单点登录令牌交换外部令牌。
- 客户端可以模拟用户

Red Hat Single Sign-On 中的令牌交换是 IETF 上 OAuth 令牌交换 规范非常松散的实施。我们将其扩展了一个少，忽略了其中一些，并松散地解释该规范的其他部分。它是域的 OpenID Connect 令牌端点上的简单授权类型调用。

```
/auth/realms/{realm}/protocol/openid-connect/token
```

它接受表格参数（**application/x-www-form-**url 编码的 ）作为输入，输出取决于您请求交换的令牌类型。令牌交换是一个客户端端点，因此请求必须为调用客户端提供身份验证信息。公共客户端将其客户端标识符指定为表单参数。机密客户端也可以使用表单参数来传递其客户端 ID 和机密，或者您的管理员已在您的域中配置了客户端身份验证流。

7.1.1. 表单参数

client_id

必要可能。使用表单参数进行身份验证的客户端需要此参数。如果您使用 Basic Auth、客户端 JWT 令牌或客户端证书身份验证，则不要指定此参数。

client_secret

必要可能。使用表单参数进行身份验证并使用客户端 secret 作为凭证的客户端需要此参数。如果域中的客户端调用通过不同的方式进行身份验证，则不要指定此参数。

grant_type

必需。参数的值必须是 **urn:ietf:params:oauth:grant-type:token-exchange**。

subject_token

可选。代表发送请求者的身份的安全令牌。如果要交换新令牌的现有令牌，则需要此项。

subject_issuer

可选。标识 ***subject_token*** *的签发者。如果令牌来自当前域，或者签发者是否可以从* ***subject_token_type*** *中确定，则可以留空。否则需要指定它。有效值是为* ***您的域配置的身份提供程序*** *的别名。或由特定* ***身份提供程序配置的签发者声明标识符***。

subject_token_type

可选。此参数是使用 ***subject_token*** *参数传递的令牌的类型。如果* ***subject_token*** *来自域，则默认为* ***urn:ietf:params:oauth:token-type:access_token***，*则是一个访问令牌。如果是一个外部令牌，则根据* ***subject_issuer*** *的要求，可能不必指定此参数。*

requested_token_type

可选。此参数代表客户端要交换的令牌类型。目前只支持 oauth 和 OpenID Connect 令牌类型。这样做的默认值取决于它是 ***urn:ietf:params:oauth:token-type:refresh_token***，*在这种情况下，您会在响应中返回访问令牌和刷新令牌。其他适当的值有* ***urn:ietf:params:oauth:token-type:access_token*** *和* ***urn:ietf:params:oauth:token-type:id_token***

培训对象

可选。此参数指定您希望新令牌被 mint 的目标客户端。

requested_issuer

可选。此参数指定客户端希望由外部供应商报告令牌。它必须是在 ***域中配置的身份提供程序*** *的别名。*

requested_subject

可选。如果您的客户端希望模仿其他用户，则指定用户名或用户 id。

scope

未实施。此参数代表客户端正在请求的目标 OAuth 和 OpenID Connect 范围。此时不会实施，但一旦红帽单点登录能够更好地支持一般范围。

注意

我们目前仅支持 OpenID Connect 和 OAuth 交换。以后可能会根据用户需求添加对基于 SAML 的客户端和身份提供程序的支持。

7.1.2. 来自令牌交换请求的响应

来自交换调用的成功响应将返回 HTTP 200 响应代码，它带有依赖于 ***requested-token-type*** *和* ***requested_issuer*** *请求的内容类型。OAuth 请求的令牌类型将返回 JSON 文档，如 OAuth Token Exchange 规范中所述。*

```
{
  "access_token" : ".....",
  "refresh_token" : ".....",
  "expires_in" : "...."
 }
```

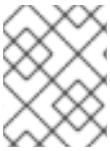
请求刷新令牌的客户端将在响应中返回访问和刷新令牌。仅请求访问令牌类型的客户端仅获得响应中的访问令牌。通过 ***requested_issuer*** paramater *请求外部签发者的客户端可能会包含过期信息。*

错误响应通常低于 400 HTTP 响应代码类别，但其他错误状态代码可以根据错误的严重性进行返回。错误响应可能包括内容取决于 ***requested_issuer***。*基于 OAuth 的交换可能会返回 JSON 文档，如下所示：*

```
{
  "error" : "...."
  "error_description" : "...."
```

```
}
```

可以根据交换类型返回额外的错误声明。例如，如果用户没有身份提供程序的链接，OAuth Identity Providers 可能包含额外的 **account-link-url** 声明。该链接可用于客户端发起链路请求。



注意

令牌交换设置需要了解管理员权限（请参见《服务器管理指南》以了解。您需要授予客户端权限才能交换。本章稍后将对此进行探讨。

本章的其余部分介绍了设置要求，并为不同的交换场景提供示例。为了简单起见，可以将当前域报告为 内部 令牌来报告令牌，以及由外部域或身份提供程序报告的令牌作为 外部 令牌。

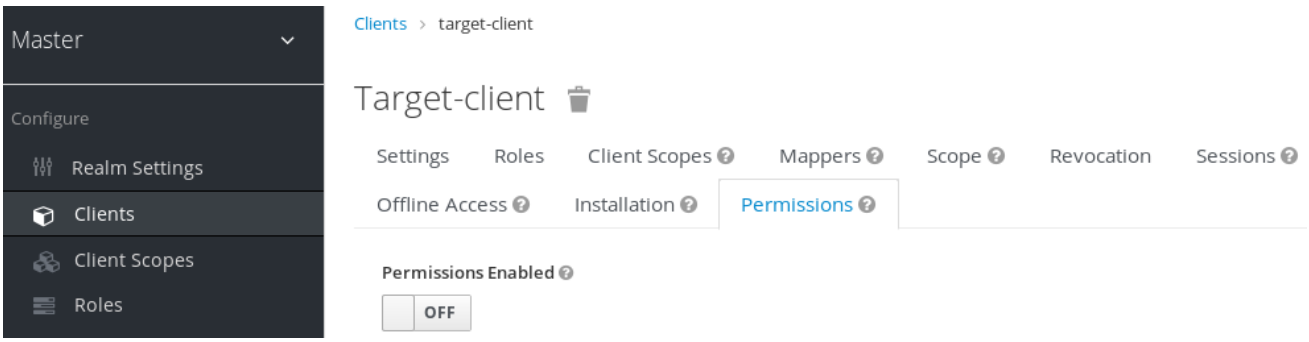
7.2. 内部令牌到内部令牌交换

使用内部令牌来对特定客户端进行令牌交换，您希望为不同的目标客户端交换这个令牌。您为什么会想进行此操作？当客户端本身有 mint 令牌时，通常会发生这种情况，需要为访问令牌中需要不同的声明和权限的其他应用程序创建额外的请求。如果您需要执行"权限级"，且应用程序需要在比较可信应用上调用并且不想传播您当前的访问令牌，则可能需要进行此类交换的其他原因。

7.2.1. 授予对交换的权限

希望为不同客户端交换令牌的客户端需要在管理控制台中授权。您需要在您要交换的目标客户端中定义 **token-exchange** fine grain 权限。

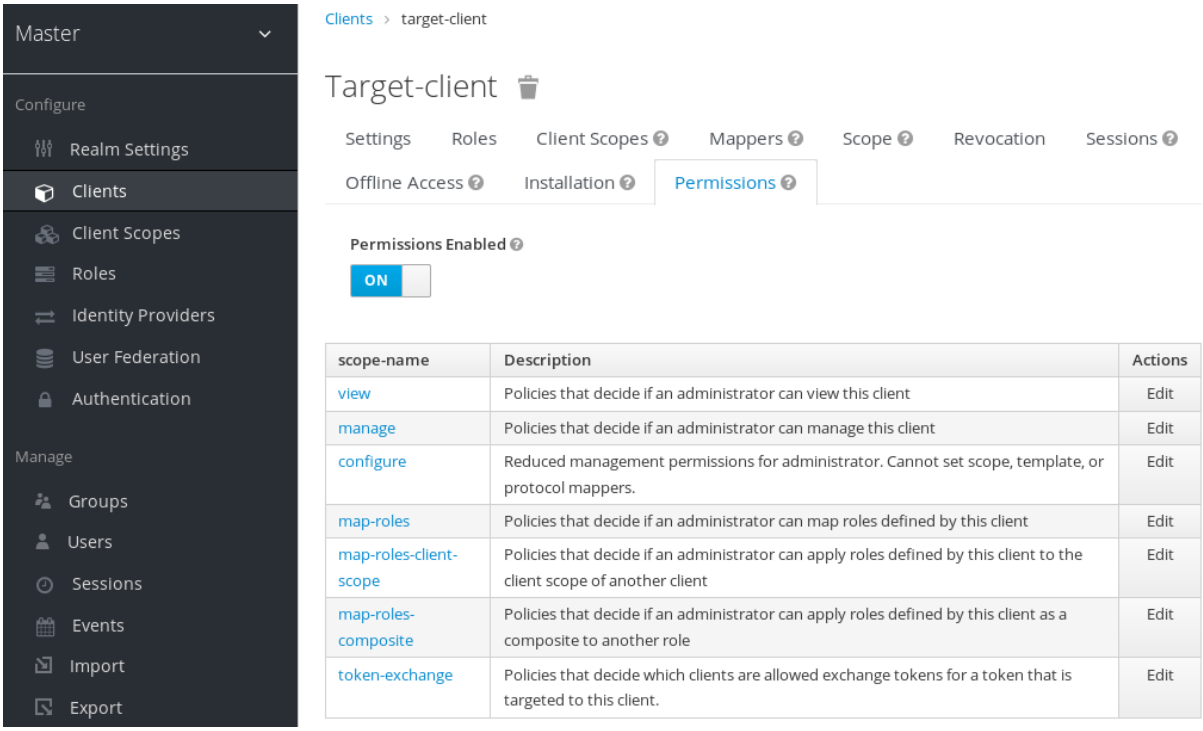
目标客户端权限



流程

1. 将启用 **的权限** 切换到 **ON**。

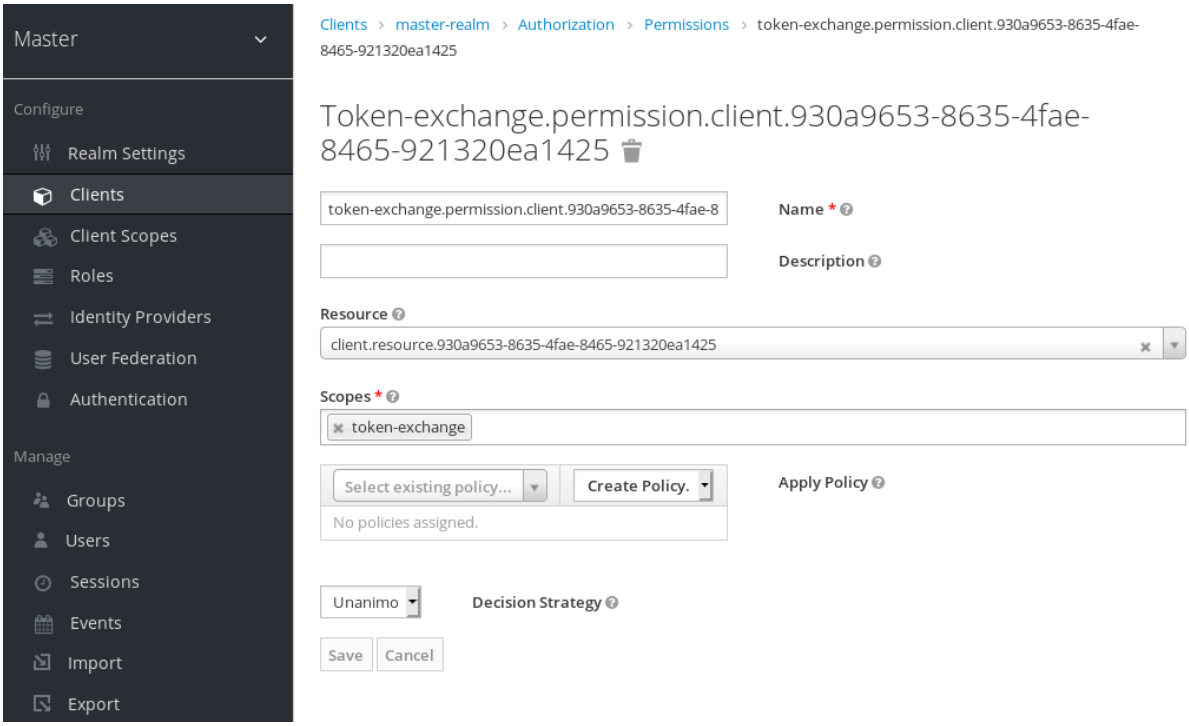
目标客户端权限



*该页面显示一个 **token-exchange** 链接。*

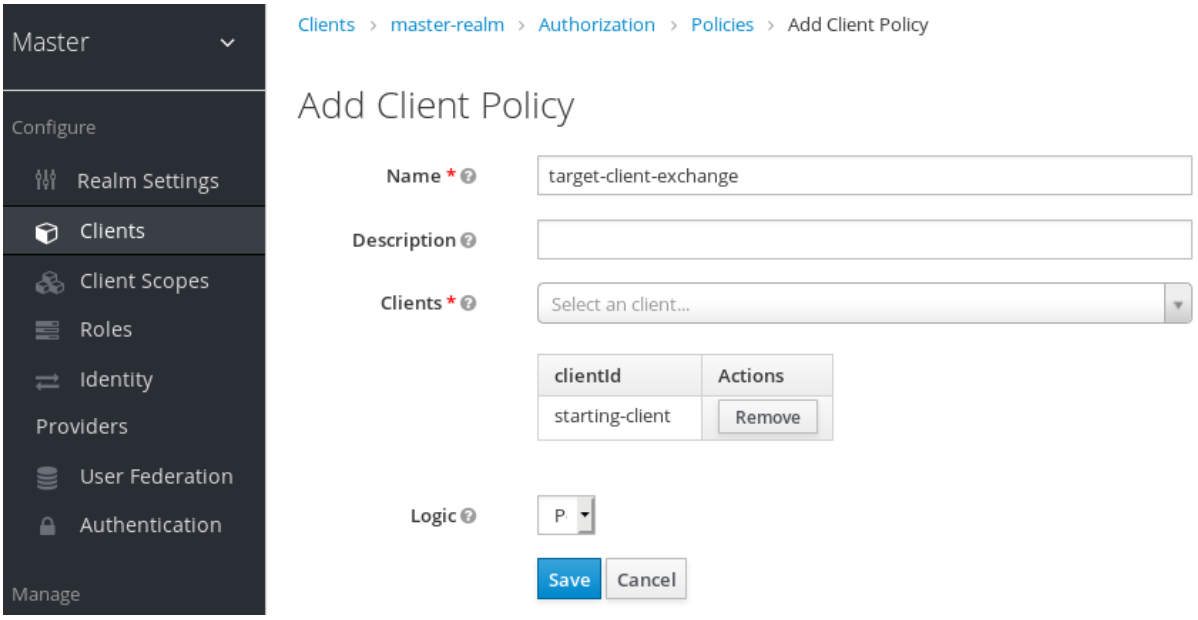
2. *点击该链接以开始定义权限。*
   这个设置页面会显示。

目标客户端交换权限设置



3. *点 **Authorization** 链接为这个权限定义策略*
4. *点 **Policies** 标签页。*
5. *创建 **客户端** 策略.*

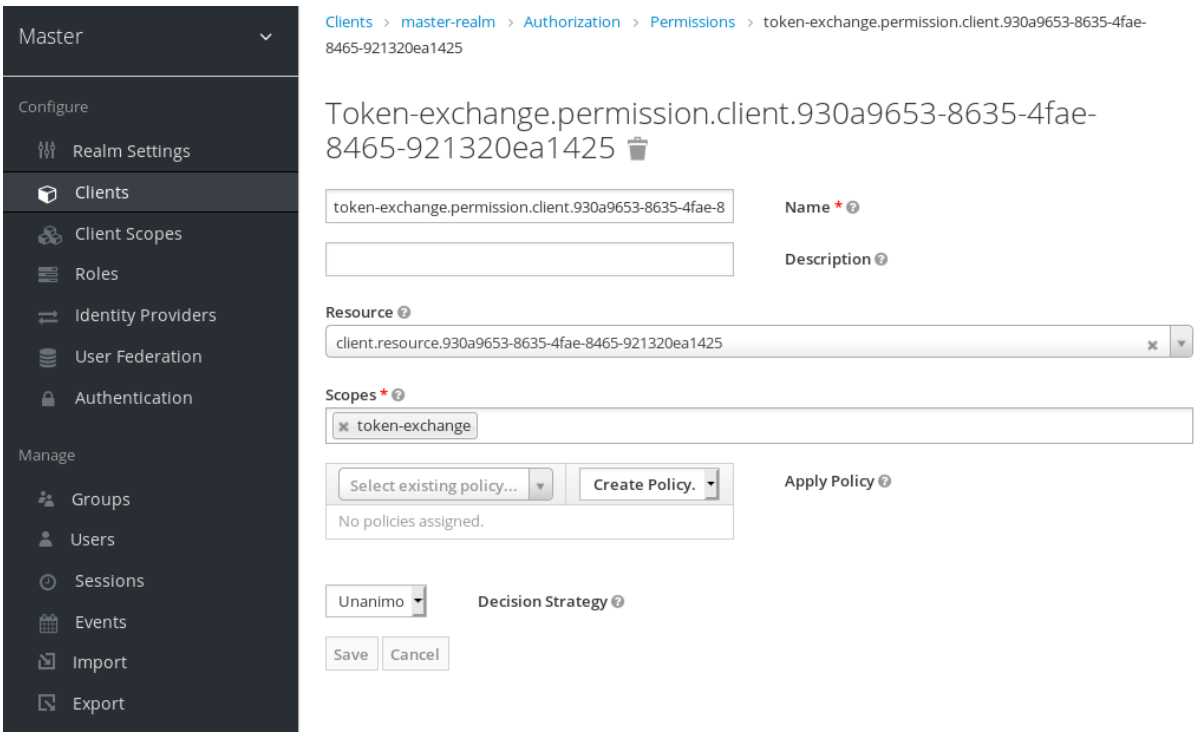
客户端策略创建



6. *在起始客户端中输入，作为请求令牌交换的经过身份验证的用户。*

7. *创建此策略后，返回目标客户端的* **token-exchange** *权限，再添加您刚刚定义的客户端策略。*

应用客户端策略



您的客户端现在有调用的权限。如果您没有正确执行此操作，如果您尝试进行交换，将得到一个 403 Forbidden 响应。

7.2.2. 发出请求

当客户端交换针对其他客户端的令牌的现有令牌时，请使用 **audience** *参数。此参数必须是您在 Admin 控制台中配置的目标客户端的客户端标识符。*

```
curl -X POST \
    -d "client_id=starting-client" \
    -d "client_secret=the client secret" \
```

```
--data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
-d "subject_token=...." \
--data-urlencode "requested_token_type=urn:ietf:params:oauth:token-type:refresh_token" \
-d "audience=target-client" \
http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token
```

subject_token 参数必须是目标域的访问令牌。如果您的 **requested_token_type** 参数是一个刷新令牌类型，则响应将同时包含访问令牌、刷新令牌和过期。以下是您从这个调用中获取的 JSON 响应示例：

如果没有设置 **使用者** 参数，则参数的值默认为生成令牌交换请求的客户端。

与机密客户端不同，公共客户端不允许使用来自其他客户端的令牌来执行令牌交换。如果您传递了 **subject_token**，发布令牌的（机密）客户端应与客户端发出请求匹配；如果向其他客户端发布，则发出请求的客户端应当位于使用者上。

如果您明确设置目标 **audience**（客户端与发出请求的客户端不同），您应确保为客户端集配置了 **token-exchange** 范围权限，设置 **audience** 权限以允许客户端成功完成交换。

```
{
  "access_token" : "....",
  "refresh_token" : "....",
  "expires_in" : 3600
}
```

7.3. 外部令牌交换的内部令牌

您可以从外部身份提供程序为 externl 令牌 mint 交换域令牌。此外部身份提供程序必须在管理控制台的 **Identity Provider** 部分中配置。目前只支持基于 OAuth/OpenID Connect 的外部身份提供程序，其中包括所有社交供应商。Red Hat Single Sign-On 不执行到外部供应商的后台交换。因此，如果帐户没有链接，您将无法获取外部令牌。要能够获取这些条件之一的外部令牌，必须满足以下条件：

- 用户必须至少使用外部身份提供程序登录
- 用户必须通过用户帐户服务与外部身份提供程序相关联
- 用户帐户使用 Client Initiated Account Linking API 通过外部身份提供程序链接。

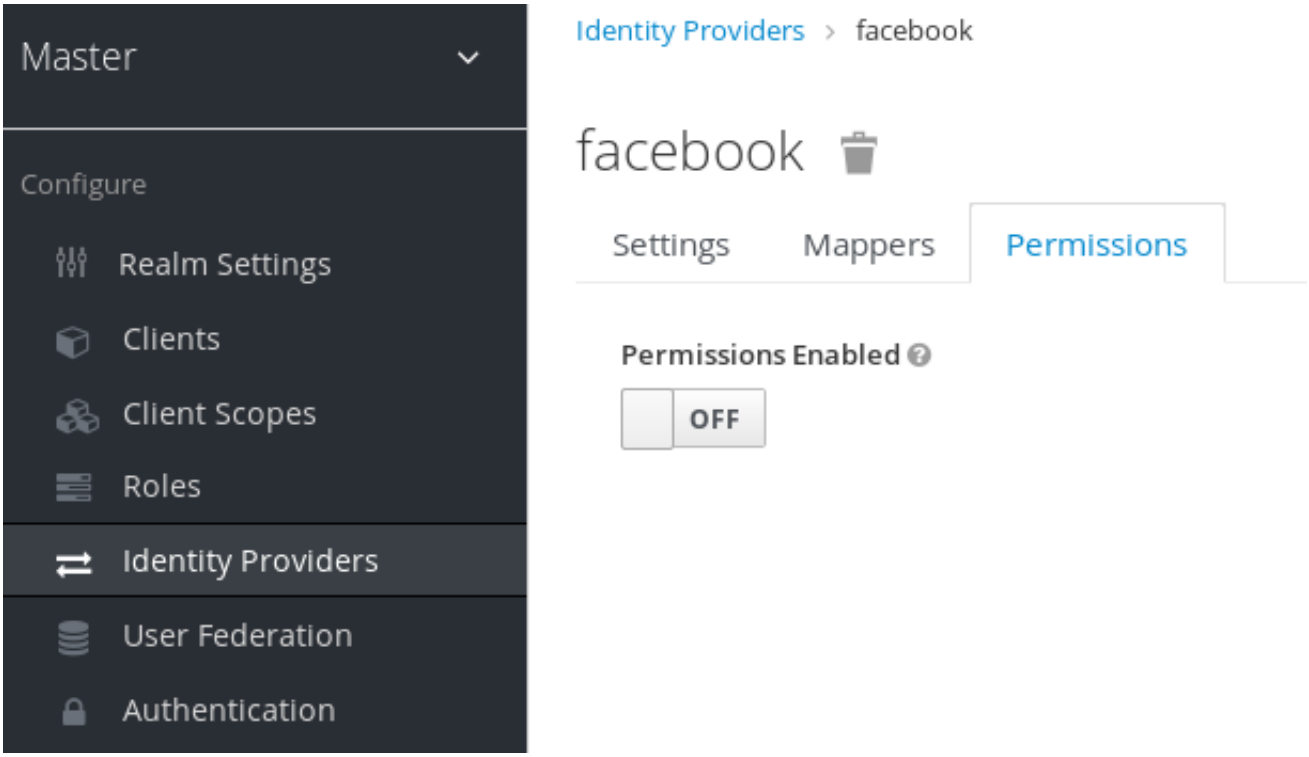
最后，必须将外部身份提供程序配置为存储令牌，或者，上述操作之一必须使用相同的用户会话和您要交换的内部令牌来执行。

如果帐户没有链接，则交换响应将包含一个可用于建立它的链接。这在进行 Request 部分进行 进一步讨论。

7.3.1. 授予对交换的权限

外部令牌交换请求内部将被拒绝使用 403, Forbidden 响应，直到您为调用客户端授予与外部身份提供程序交换令牌的权限。要向客户端授予权限，您需要进入身份提供程序的配置页面到 **Permissions** 选项卡。

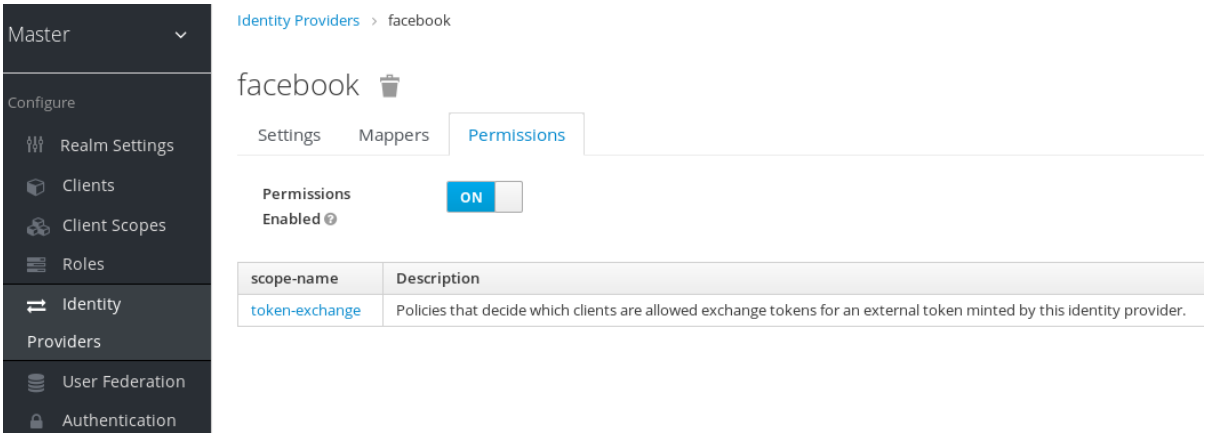
身份供应商权限



流程

1. 将启用 **的权限** 切换到 **ON**。

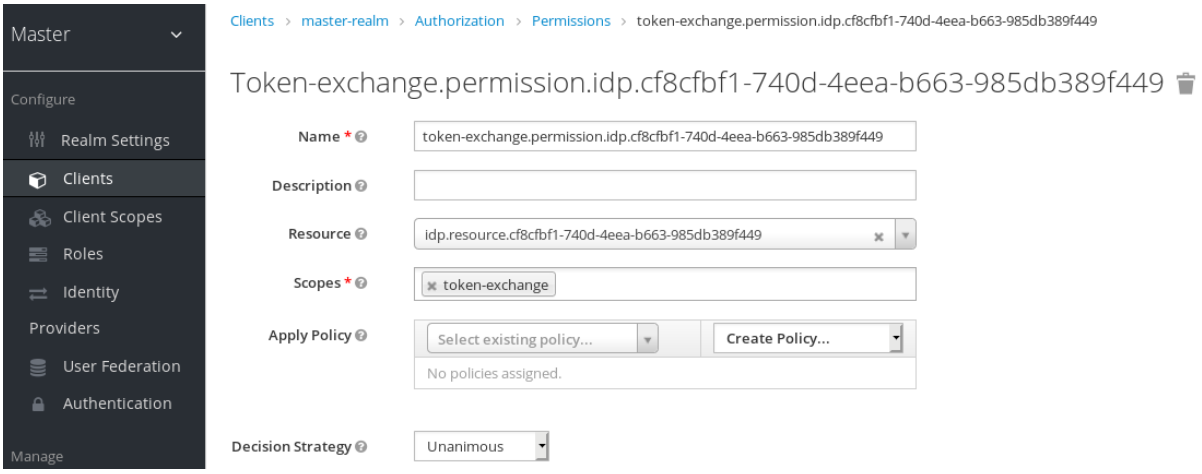
   身份供应商权限



   页面中显示 **token-exchange** 链接。

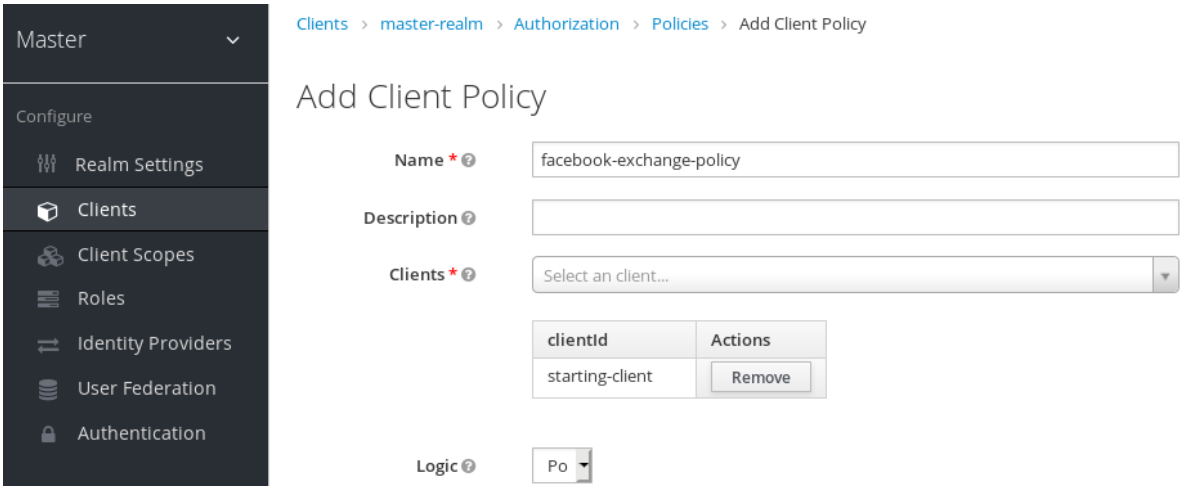
2. 点击链接以开始定义权限。
   这个设置页面会出现。

   身份供应商交换权限设置



3. *点 **Authorization** 链接，再进入 **Policies** 标签页来创建客户端策略。*

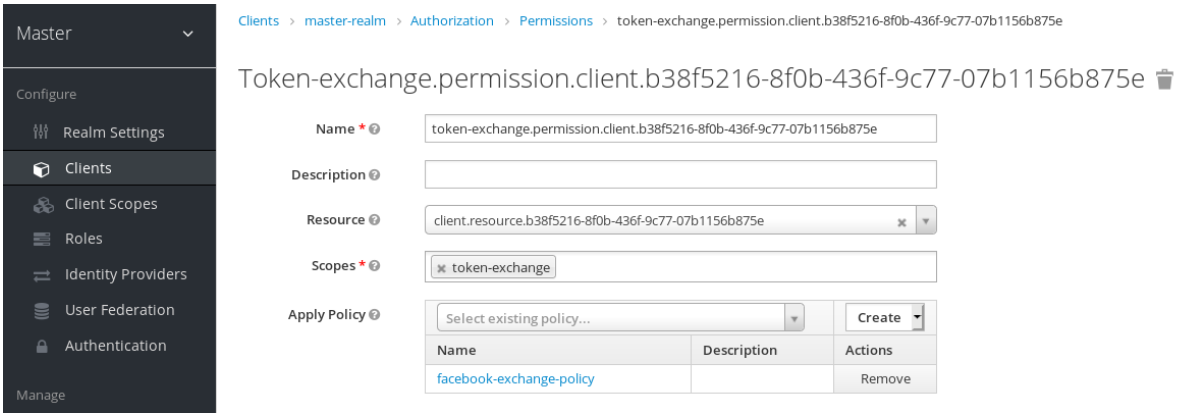
   ***客户端策略**创建*



4. *输入作为请求令牌交换的经过身份验证的用户的起始客户端。*

5. *返回到身份提供程序的 **token-exchange** 权限，再添加您刚刚定义的客户端策略。*

   *应**用客户端策略***



您的客户端现在有调用的权限。如果您没有正确执行此操作，如果您尝试进行交换，将得到一个 403 Forbidden 响应。

7.3.2. 发出请求

当您的客户端将现有的内部令牌切换到外部时，您要提供 **requested_issuer** 参数。参数必须是配置的身份提供程序的别名。

```
curl -X POST \
    -d "client_id=starting-client" \
    -d "client_secret=the client secret" \
    --data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
    -d "subject_token=...." \
    --data-urlencode "requested_token_type=urn:ietf:params:oauth:token-type:access_token" \
    -d "requested_issuer=google" \
    http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token
```

subject_token 参数必须是目标域的访问令牌。**requested_token_type** 参数必须是 **urn:ietf:params:oauth:token-type:access_token** 或 left blank。目前不支持其他请求的令牌类型。以下是您从这个调用中获取的 JSON 响应成功的示例。

```
{
   "access_token" : "....",
   "expires_in" : 3600
   "account-link-url" : "https://...."
}
```

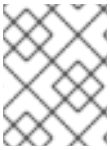
如果外部身份提供程序没有以任何原因链接，您可以使用此 JSON 文档获取 HTTP 400 响应代码：

```
{
   "error" : "....",
   "error_description" : "..."
   "account-link-url" : "https://...."
}
```

错误 声明将为 **token_expired** 或 **not_linked**。提供了 **account-link-url** 声明，客户端可以执行 客户端初始化帐户链接。大多数（如果不是全部）供应商都需要通过浏览器 OAuth 协议链接。使用 **account-link-url** 时，只需在其中添加 **redirect_uri** 查询参数，并可转发浏览器来执行链接。

7.4. 外部令牌到内部令牌交换

您可以信任并交换外部身份提供程序用于内部令牌的外部令牌。这可用于在域间桥接，或只用于从您的社交提供程序信任令牌。它的工作方式与身份提供程序浏览器登录类似，如果新用户不存在，则会将新用户导入到您的域中。



注意

外部令牌交换的当前限制是，如果外部令牌映射到现有用户，则不会允许交换，除非现有用户已具有外部身份提供程序的帐户链接。

当交换完成后，将在域内创建一个用户会话，您会收到访问或刷新令牌，具体取决于 **requested_token_type** 参数值。您应该注意，这个新用户会话会一直处于激活状态，直到被调用通过这个新访问令牌的域的注销端点为止。

这些类型的更改需要在管理控制台中需要配置的身份提供程序。

注意

目前不支持 SAML 身份提供程序。Twitter 令牌不能交换。

7.4.1. 授予对交换的权限

在进行外部令牌交换之前，您可以为调用客户端授予该交换的权限。这个权限的方式相同，与 授予外部权限的内部 权限相同。

如果还提供一个 **audience** 参数，其值指向调用者之外的不同客户端，还必须授予调用客户端的权限，以便与 **audience** 参数的特定目标客户端交换。本节前面将讨论如何 执行此操作。

7.4.2. 发出请求

subject_token_type 必须是 **urn:ietf:params:oauth:token-type:access_token** 或 **urn:ietf:params:oauth:token-type:jwt**。如果类型是 **urn:ietf:params:oauth:token-type:access_token**，您指定 **subject_issuer** 参数，它需要是配置的身份供应商的别名。如果类型为 **urn:ietf:params:oauth:token-type:jwt**，则提供程序将通过 JWT 中的 **签发者** 声明匹配，该提供商必须是提供程序的别名或提供程序配置中的注册签发者。

为进行验证，如果令牌是一个访问令牌，则会调用该提供程序的用户信息服务来验证令牌。成功调用意味着访问令牌有效。如果主体令牌是一个 JWT，如果提供程序启用了签名验证，则会试图进行尝试，否则它将默认在用户 info 服务上调用 以验证令牌。

默认情况下，Aded 内部令牌将使用调用客户端使用为调用客户端定义的协议映射程序来确定令牌中的内容。另外，您可以使用 **audience** 参数指定不同的目标客户端。

```
curl -X POST \
    -d "client_id=starting-client" \
    -d "client_secret=the client secret" \
    --data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
    -d "subject_token=...." \
    -d "subject_issuer=myOidcProvider" \
    --data-urlencode "subject_token_type=urn:ietf:params:oauth:token-type:access_token" \
    -d "audience=target-client" \
    http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token
```

如果您的 **requested_token_type** 参数是一个刷新令牌类型，则响应将同时包含访问令牌、刷新令牌和过期。以下是您从这个调用中获取的 JSON 响应示例：

```
{
  "access_token" : "....",
  "refresh_token" : "....",
  "expires_in" : 3600
}
```

7.5. 模拟（IMPERSONATION）

对于内部和外部令牌交换，客户端可以代表用户请求以模拟其他用户。例如，您可能有一个需要模拟用户的管理员应用程序，以便支持工程师能够调试问题。

7.5.1. 授予对交换的权限

subject 令牌代表的用户必须具有模拟其他用户的权限。有关如何启用这个权限的服务器 管理指南。它可以通过角色完成，也可以通过更精细的管理员权限完成。

7.5.2. 发出请求

根据其他章节中所述进行请求，但另外另外指定 **requested_subject** 参数。此参数的值必须是用户名或用户 id。

```
curl -X POST \
    -d "client_id=starting-client" \
    -d "client_secret=the client secret" \
    --data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
    -d "subject_token=...." \
    --data-urlencode "requested_token_type=urn:ietf:params:oauth:token-type:access_token" \
    -d "audience=target-client" \
    -d "requested_subject=wburke" \
    http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token
```

7.6. DIRECT NAKED IMPERSONATION

您可以在不提供 **subject_token** 的情况下发出内部令牌交换请求。这称为直接人类模拟，因为它将大量信任放在客户端中，因为该客户端可以模拟域中任何用户。对于无法获取主体才能交换的应用，您可能需要进行这个桥接。例如，您可能要集成一个直接与 LDAP 登录的遗留应用程序。在这种情况下，传统应用程序可以验证用户本身，但不能获取令牌。

警告

为客户端启用直接 naked impersonation 存在风险。如果客户端的凭据已被停止，则客户端可以模拟系统中的任何用户。

7.6.1. 授予对交换的权限

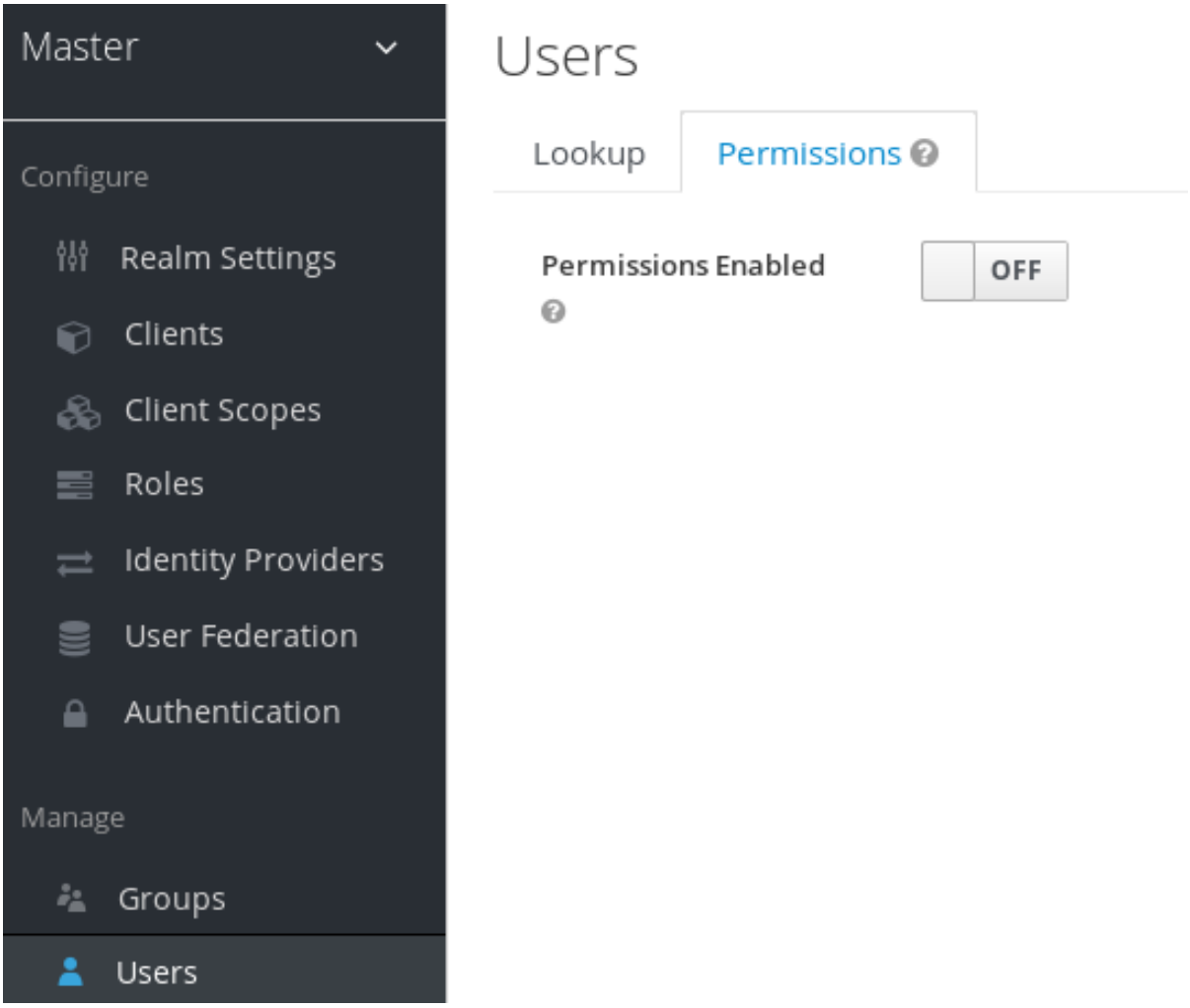
如果提供 **audience** 参数，则调用客户端必须具有与客户端交换的权限。本章前文中介绍了如何设置此设置。

另外，调用客户端必须被授予模拟用户的权限。在管理控制台中.

流程

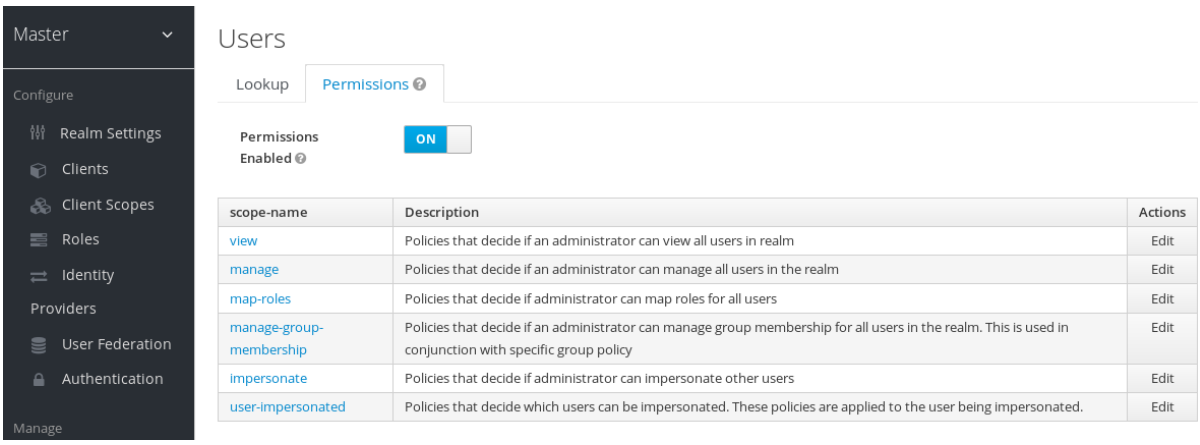
1. 点菜单中的 **Users**。
2. 单击**权限**选项卡。

用户权限



3. *将* ***权限已启用*** *为* true。

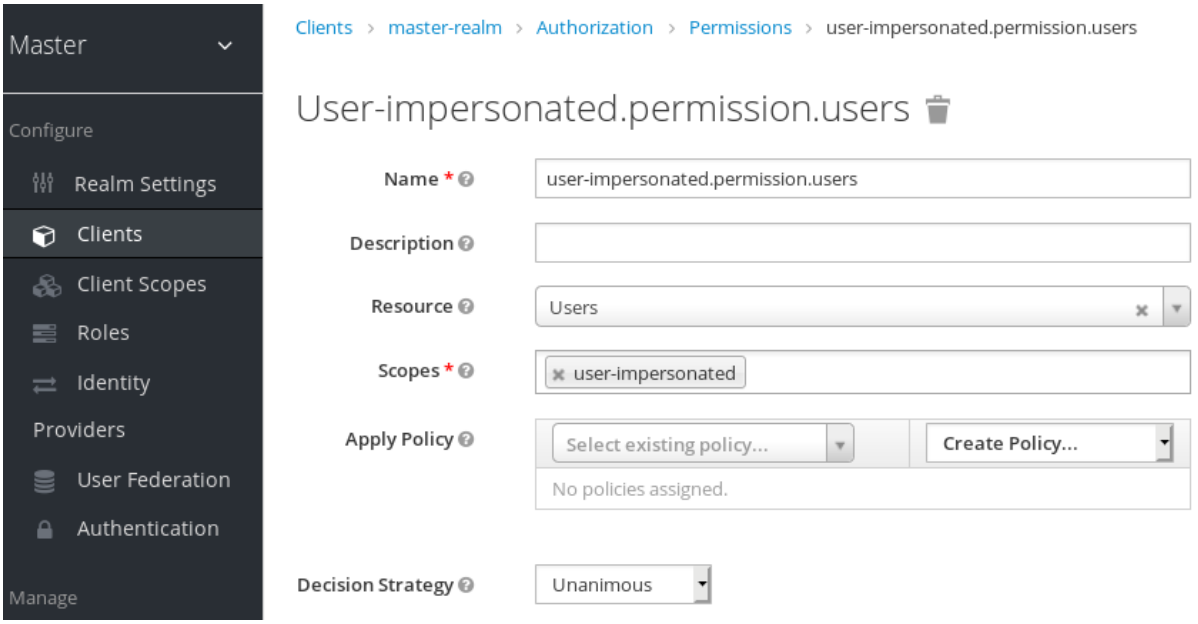
身份供应商权限



该页面显示 ***模仿*** *链接。*

4. *点击该链接以开始定义权限。*
这个设置页面会显示。

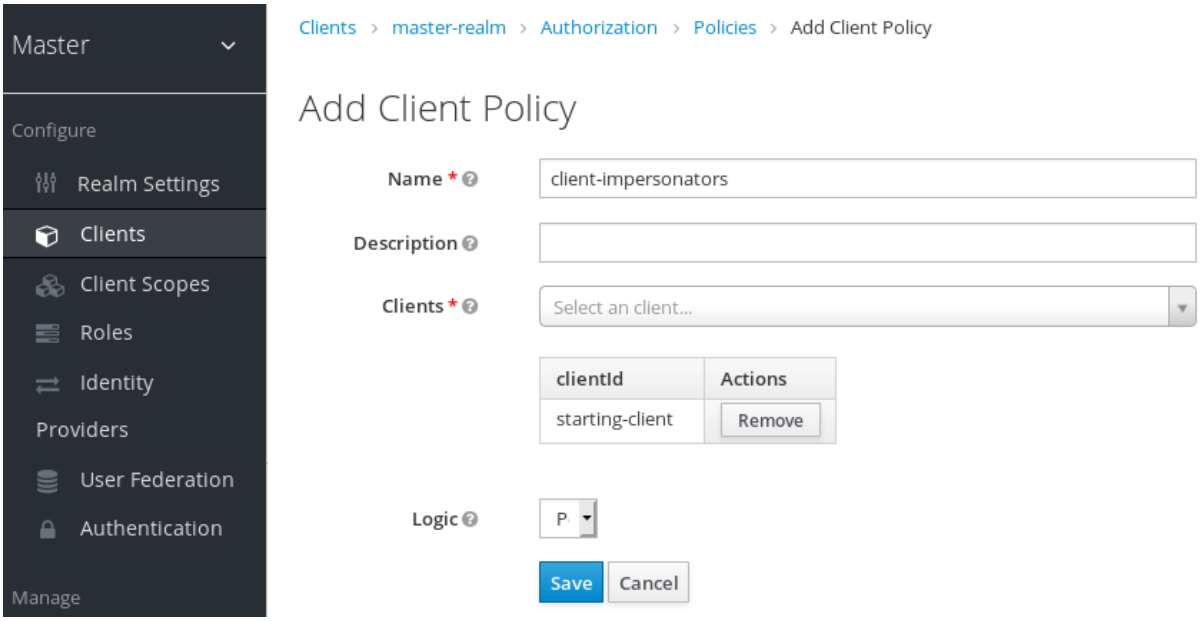
用户识别权限设置



5. *点 **Authorization** 链接*

6. *前往 **Policies** 选项卡，再创建客户端策略。*

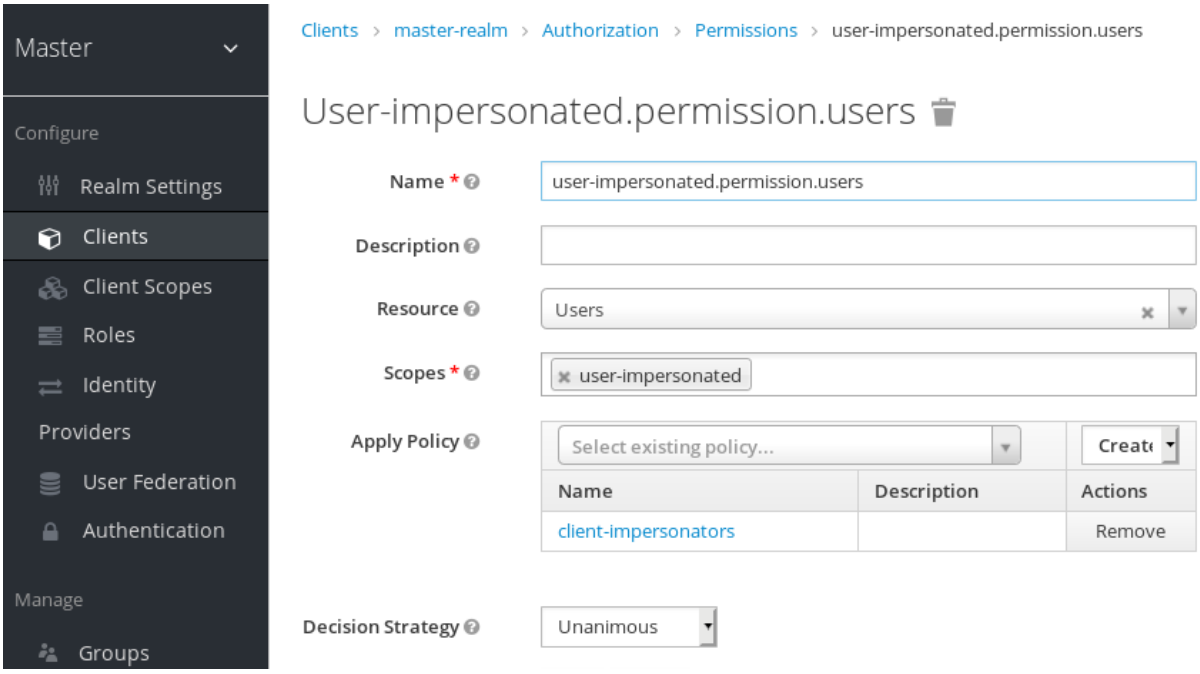
   ***客户端策略**创建*



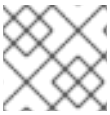
7. *输入作为请求令牌交换的经过身份验证的用户的起始客户端。*

8. *返回到用户的 **模拟** 权限，再添加您刚刚定义的客户端策略。*

   应用客户端策略



现在，您的客户端具有模拟用户的权限。如果您没有正确执行此操作，如果您尝试进行这类交换，您将得到一个 403 Forbidden 响应。



注意

不允许公共客户端直接生成模拟。

7.6.2. 发出请求

*要发出请求，只需指定 **requested_subject** 参数。这必须是有效用户的用户名或用户 id。如果您想要，还可以指定 **audience** 参数。*

```
curl -X POST \
    -d "client_id=starting-client" \
    -d "client_secret=the client secret" \
    --data-urlencode "grant_type=urn:ietf:params:oauth:grant-type:token-exchange" \
    -d "requested_subject=wburke" \
    http://localhost:8080/auth/realms/myrealm/protocol/openid-connect/token
```

7.7. 使用服务帐户扩展权限模型

*在授予客户端权限进行交换时，您不一定为每个客户端和每个客户端手动启用这些权限。如果客户端关联了服务帐户，您可以使用角色将权限分组到一起，并通过为客户端的服务帐户分配角色来分配交换权限。例如，您可以定义一个 **naked-exchange** 角色以及具有该角色的任何服务帐户都可以进行 naked 交换。*

7.8. EXCHANGE 漏洞

当您启动允许令牌交换时，您必须了解和小心。

第一个是公共客户端。公共客户端没有或要求客户端证书才能执行交换。任何具有有效令牌的人都可以 模拟 公共客户端，并执行允许公共客户端执行的交换。如果您的域管理有任何不信任客户端，则公共客户端可以在您的权限模型中打开漏洞。这就是为什么直接交换不会允许公共客户端，如果调用客户端为公共客户端，则中止并出错。

对于一个域令牌，可以交换 Facebook、Google 等的社交令牌。请注意，交换令牌允许交换令牌在这些社交 Web 站点上创建虚拟帐户，仔细和 vigilant。使用默认角色、组和身份提供程序映射程序来控制分配给外部社交用户的属性和角色。

直接交换非常危险。您会在调用的客户端中放入大量信任，它将不会泄漏其客户端凭证。如果这些凭证泄漏，thief 可以模拟您系统中的任何人。这与已具有现有令牌的机密客户端的直接对比。您有两个身份验证因素：访问令牌和客户端凭证，且您只处理一个用户。因此，使用直接交换器。