# Red Hat build of Keycloak 24.0

## 服务器管理指南

Last Updated: 2024-06-28

## 法律通告



## 摘要

本指南包含管理员配置红帽构建的 Keycloak 24.0 的信息。

# 目录

# 第 1 章 红帽构建的 KEYCLOAK 功能和概念

红帽构建的 Keycloak 是 Web 应用和 RESTful Web 服务的单点登录解决方案。红帽构建的 Keycloak 的目标是使安全简单，以便应用程序开发人员易于保护他们机构中部署的应用程序和服务。开发人员通常必须自己编写的安全功能是开箱即用的，可根据您的组织的个人要求轻松定制。红帽构建的 Keycloak 为登录、注册、管理和帐户管理提供了可自定义的用户界面。您还可以使用红帽构建的 Keycloak 作为集成平台，将其插入到现有的 LDAP 和 Active Directory 服务器中。您还可以将身份验证委托给 Facebook 和 Google 等第三方身份提供程序。

## 1.1. 功能

红帽构建的 Keycloak 提供以下功能：

- 浏览器应用程序的单点登录和单点登录。
- OpenID Connect 支持。
- OAuth 2.0 支持。
- SAML 支持。
- 身份代理 - 通过外部 OpenID Connect 或 SAML 身份提供程序进行身份验证。
- 社交登录 - 启用使用 Google、GitHub、Facebook、Twitter 和其他社交网络登录。
- 用户 Federation - 从 LDAP 和 Active Directory 服务器同步用户。
- Kerberos 网桥 - 自动验证登录到 Kerberos 服务器的用户。
- 用于集中管理用户、角色、角色映射、客户端和配置的管理控制台。
- 允许用户集中管理其帐户的帐户控制台。
- 主题支持 - 自定义所有用户面向用户的页面，以与您的应用程序和品牌集成。
- 双因素身份验证 - 通过 Google Authenticator 或 FreeOTP 支持 TOTP/HOTP。
- 登录流 - 可选用户自我注册、恢复密码、验证电子邮件、需要密码更新等。
- 会话管理 - 管理员和用户本身可以查看和管理用户会话。
- 令牌映射器 - 映射用户属性、角色等。如何融入令牌和语句中。
- 每个域、应用程序和用户前的撤销策略。
- CORS 支持 - 客户端适配器对 CORS 的内置支持。
- 用于 JavaScript 应用程序、JBoss EAP 等的客户端适配器。
- 支持具有 OpenID Connect Relying party 库或 SAML 2.0 Service Provider 库的任何平台/语言。

## 1.2. 基本红帽构建的 KEYCLOAK 操作

红帽构建的 Keycloak 是您在网络上管理的单独服务器。应用程序被配置为指向此服务器并加以保护。红帽构建的 Keycloak 使用 OpenID Connect 或 SAML 2.0 等开放协议标准来保护您的应用程序。浏览器应用程序会将用户的浏览器从应用程序重定向到红帽构建的 Keycloak 身份验证服务器，在其中输入其凭

证。这个重定向非常重要，因为用户完全与应用程序隔离，应用程序永远不会看到用户的凭证。相反，应用被授予一个身份令牌或断言，它经过加密签名。这些令牌可以有身份信息，如用户名、地址、电子邮件和其他配置集数据。它们也可以保存权限数据，以便应用程序可以做出授权决策。这些令牌也可用于在基于 REST 的服务上进行安全调用。

## 1.3. 核心概念和术语

在尝试使用红帽构建的 Keycloak 来保护 Web 应用程序和 REST 服务前，请考虑这些核心概念和术语。

**users**

用户是可以登录到您的系统的实体。它们可以有与自己关联的属性，如电子邮件、用户名、地址、电话号码和生日。可以为其分配组成员资格，并为其分配特定的角色。

**身份验证**

识别和验证用户的过程。

**授权**

授予用户访问权限的过程。

**credentials**

凭证是红帽构建的 Keycloak 用来验证用户身份的数据片段。某些示例包括密码、一次性密码、数字证书甚至指纹。

**roles**

角色标识用户的类型或类别。**Admin**, **user**, **manager**, 和 **employee** 是一个机构中的典型的职位。应用程序通常为特定角色分配访问权限和权限，而不是像处理用户那样的单独用户进行精细管理，难以管理。

**用户角色映射**

用户角色映射定义角色和用户之间的映射。用户可以与零个或多个角色关联。此角色映射信息可以封装到令牌和断言中，以便应用能够决定它们管理的各种资源的访问权限。

**复合角色**

复合角色是一个可以与其他角色关联的角色。例如，**超级用户** 复合角色可以与 **sales-admin** 和 **order-entry-admin** 角色关联。如果用户映射到 **超级用户** 角色，他们也会继承 **sales-admin** 和 **order-entry-admin** 角色。

**groups**

组管理用户组。可以为组定义属性。您还可以将角色映射到组。成为组成员的用户继承了该组定义的属性和角色映射。

**realms**

域管理一组用户、凭据、角色和组。用户从属于并登陆到的域。域彼此隔离，只能管理和验证其控制的用户。

**客户端**

客户端是可以请求红帽构建的 Keycloak 验证用户身份的实体。大多数情况下，客户端都是希望使用红帽构建的 Keycloak 来保护自身并提供单点登录解决方案的应用程序和服务。客户端也可以是仅请求身份信息或访问令牌的实体，以便可以在由红帽构建 Keycloak 保护的网络上安全地调用其他服务。

**客户端适配器**

客户端适配器是安装到应用程序环境中的插件，以便能够由红帽构建的 Keycloak 进行通信和保护。Red Hat build of Keycloak 对于您可以下载的不同平台，有多个适配器。此外，还有第三方适配器，您可以获得我们不涵盖的环境。

**consent**

当您作为管理员希望用户向客户端授予权限时，您才会获得同意，直到客户端能够参与身份验证过程。用户提供其凭证后，红帽构建的 Keycloak 将会弹出一个屏幕，标识客户端请求登录以及用户请求的身份信息。用户可以决定是否授予请求。

**客户端范围**

注册客户端时，您必须为该客户端定义协议映射程序和角色范围映射。存储客户端范围通常很有用，通过共享一些常见设置来更轻松地创建新客户端。这对于根据 **scope** 参数的值有条件地请求某些声明或角色也很有用。红帽构建的 Keycloak 提供了客户端范围的概念。

**客户端角色**

客户端可以定义特定于它们的角色。这基本上是一个专用于客户端的角色命名空间。

**身份令牌**

提供用户身份信息的令牌。OpenID Connect 规范的一部分。

**访问令牌**

作为 HTTP 请求的一部分提供的令牌，可授予对要调用的服务的访问权限。这是 OpenID Connect 和 OAuth 2.0 规范的一部分。

**assertion**

有关用户的信息。这通常与 SAML 身份验证响应中包含的 XML blob 相关，该响应提供有关经过身份验证的用户的身份元数据。

**服务帐户**

每个客户端都有一个内置服务帐户，允许它获取访问令牌。

**直接授权**

客户端通过 REST 调用代表用户获取访问令牌的方法。

**协议映射器**

对于每个客户端，您可以定制存储在 OIDC 令牌或 SAML 断言中的声明和断言。您可以通过创建和配置协议映射程序来为每个客户端执行此操作。

**会话**

当用户登录时，会创建一个会话来管理登录会话。会话包含用户登录的时间以及该会话中参与过单登录的应用程序等信息。管理员和用户可以查看会话信息。

**用户联邦供应商**

红帽构建的 Keycloak 可以存储和管理用户。通常，公司已经有存储用户和凭证信息的 LDAP 或 Active Directory 服务。您可以点红帽构建的 Keycloak 从这些外部存储验证凭证，并拉取身份信息。

**身份提供程序**

身份提供程序(IDP)是一个可以验证用户身份的服务。红帽构建的 Keycloak 是一个 IDP。

**身份提供程序联邦**

红帽构建的 Keycloak 可以被配置为将身份验证委派给一个或多个 IDP。通过 Facebook 或 Google+ 进行社交登录是身份提供程序联邦的示例。您还可以 hook 红帽构建的 Keycloak，将身份验证委托给任何其他 OpenID Connect 或 SAML 2.0 IDP。

**身份提供程序映射器**

在执行 IDP 联邦时，您可以将传入的令牌和断言映射到用户和会话属性。这有助于您将身份信息从外部 IDP 传播到请求身份验证的客户端。

**所需操作**

必要的操作是用户在身份验证过程中必须执行的操作。在这些操作完成前，用户无法完成身份验证过程。例如，管理员可以调度用户每月重置其密码。为所有这些用户设置 **更新密码** 必需操作。

**身份验证流**

身份验证流是用户在与系统某些方面交互时必须执行的操作。登录流程可以定义所需的凭证类型。注册流程定义了用户必须输入哪些配置集信息，以及是否需要使用 reCAPTCHA 来过滤 bots。凭证重置流程定义了用户在重置密码前必须执行的操作。

**events**

事件是管理员可以查看和 hook 的审计流。

**themes**

由红帽构建的 Keycloak 提供的每个屏幕都由一个主题支持。主题定义 HTML 模板和样式表，您可以根据需要覆盖它们。

# 第 2 章 创建第一个管理员

安装红帽构建的 Keycloak 后，您需要一个管理员帐户，该帐户可以充当具有完整权限来管理红帽构建的 Keycloak 的 *超级* 管理员。使用这个帐户，您可以登录到红帽构建的 Keycloak 管理控制台，您可以在其中创建域和用户，并注册由红帽构建 Keycloak 保护的应用程序。

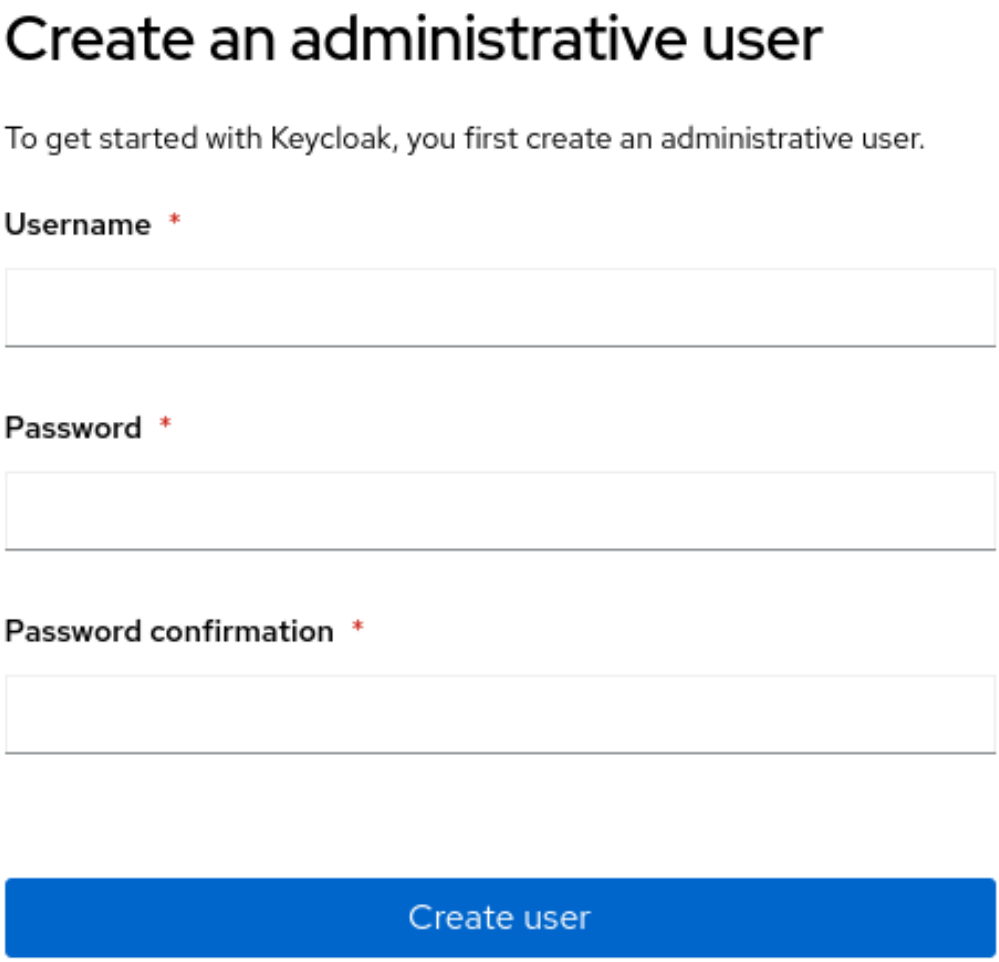
## 2.1. 在本地主机上创建帐户

如果可从 **本地主机** 访问您的服务器，请执行以下步骤。

**流程**

1. 在 Web 浏览器中，访问 http://localhost:8080 URL。
2. 提供可以重新调用的用户名和密码。

**欢迎页面**




## 2.2. 远程创建帐户

如果无法从 **localhost** 地址访问服务器，或者只想从命令行启动 Keycloak 的红帽构建，请使用 **KEYCLOAK_ADMIN** 和 **KEYCLOAK_ADMIN_PASSWORD** 环境变量来创建初始 admin 帐户。

例如：

```
export KEYCLOAK_ADMIN=<username>
export KEYCLOAK_ADMIN_PASSWORD=<password>

bin/kc.[sh|bat] start
```

# 第 3 章 配置域

具有管理控制台的管理帐户后，您可以配置域。realm 是您管理对象的空间，包括用户、应用程序、角色和组。用户从属于并登陆到的域。一个红帽构建的 Keycloak 部署可以定义、存储和管理与数据库中的空间一样多的域。

## 3.1. 使用管理控制台

您可以在红帽构建的 Keycloak 管理控制台中配置域并执行大多数管理任务。

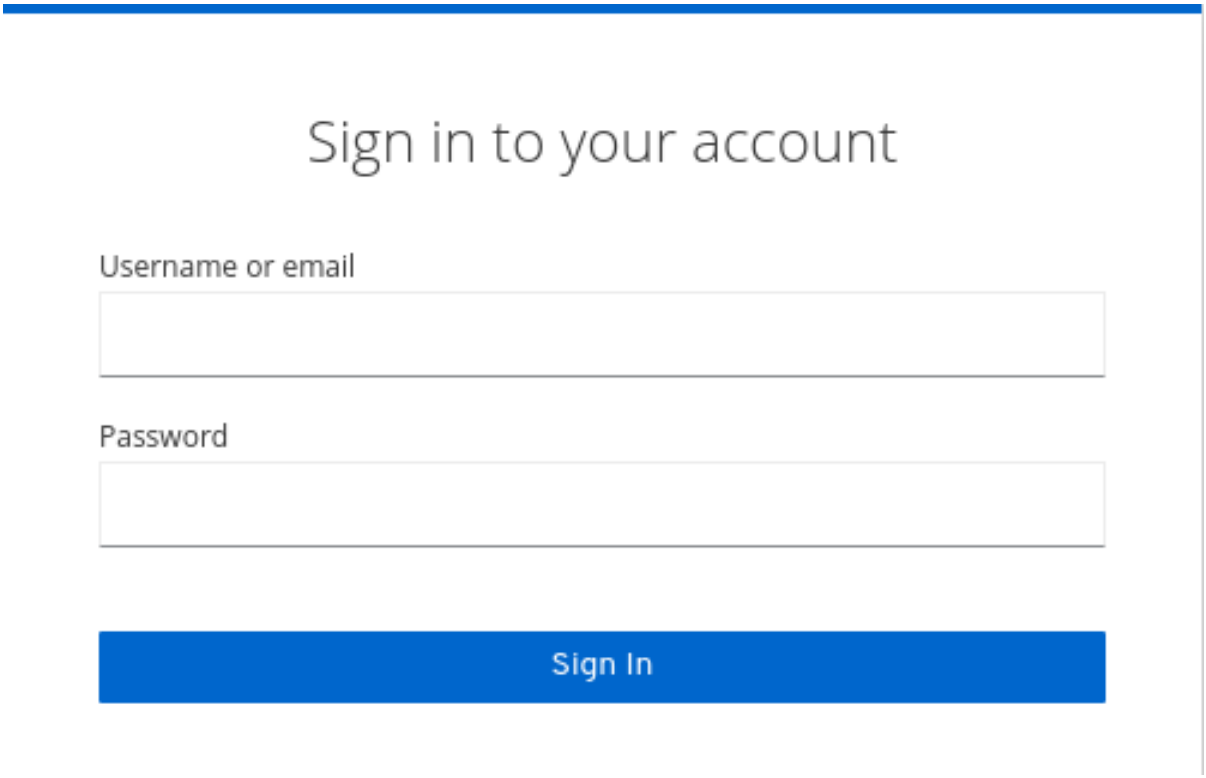
**先决条件**

- 您需要一个管理员帐户。请参阅创建第一个管理员。

**流程**

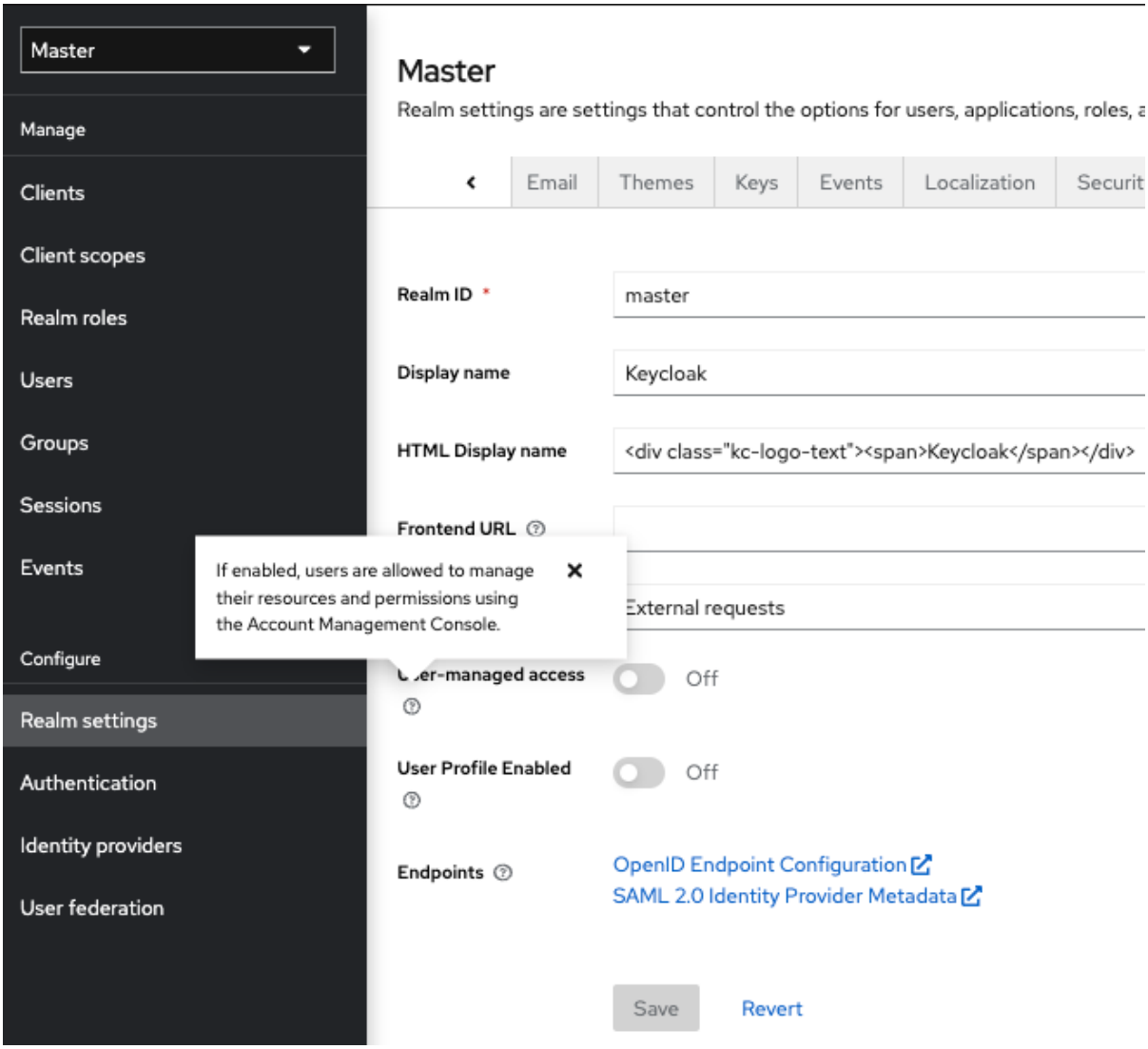
1. 前往管理控制台的 URL。
   例如，对于 localhost，请使用此 URL ：http://localhost:8080/admin/

   **登录页面**



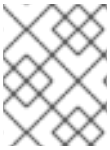

2. 根据 创建初始 admin 用户指南，输入您在 Welcome Page 或通过环境变量 中创建的用户名和密码。此操作显示 Admin 控制台。

   **管理控制台**




3. 请注意您可以使用的菜单和其他选项：

- 单击标有 **Master** 的菜单，以选择要管理的域或创建新域。
- 点右列表查看您的帐户或注销。
- 将鼠标悬停在问号 **?** 图标上，以显示描述该字段的工具提示文本。以上镜像显示了操作中的工具提示。
- 点问号 **?** 图标显示描述该字段的工具提示文本。以上镜像显示了操作中的工具提示。



**注意**

从管理控制台导出文件不适合在服务器之间备份或数据传输。只有引导时导出适合服务器之间备份或数据传输。

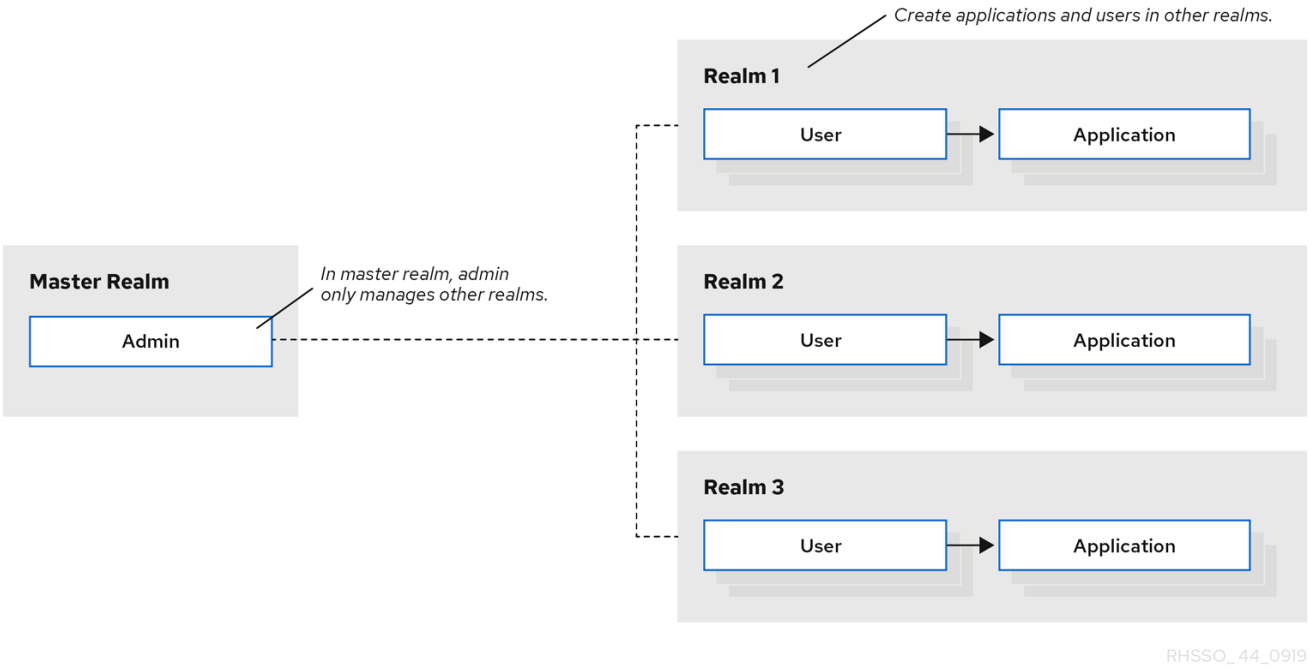
## 3.2. MASTER 域

在 Admin 控制台中，存在两种类型的域：

- **Master 域** - 首次启动红帽 Keycloak 构建时，已为您创建此域。它包含您在首次登录时创建的管理员帐户。仅使用 *master* 域来创建和管理系统中的域。

- **其他域** - 这些域由管理员在主域中创建。在这些域中，管理员将管理您机构中的用户及其所需的应用程序。应用程序归用户所有。

### realms 和应用程序




域彼此隔离，只能管理和验证其控制的用户。遵循此安全模型有助于防止意外更改，并遵循允许用户帐户访问其当前任务所需的特权和电源的原则。

### 其他资源

- 如果要禁用 *master* 域并在您创建的任何新域中定义管理员帐户，请参阅 Dedicated Realm Admin 控制台。每个域都有自己的专用管理控制台，您可以使用本地帐户登录该控制台。

## 3.3. 创建一个域

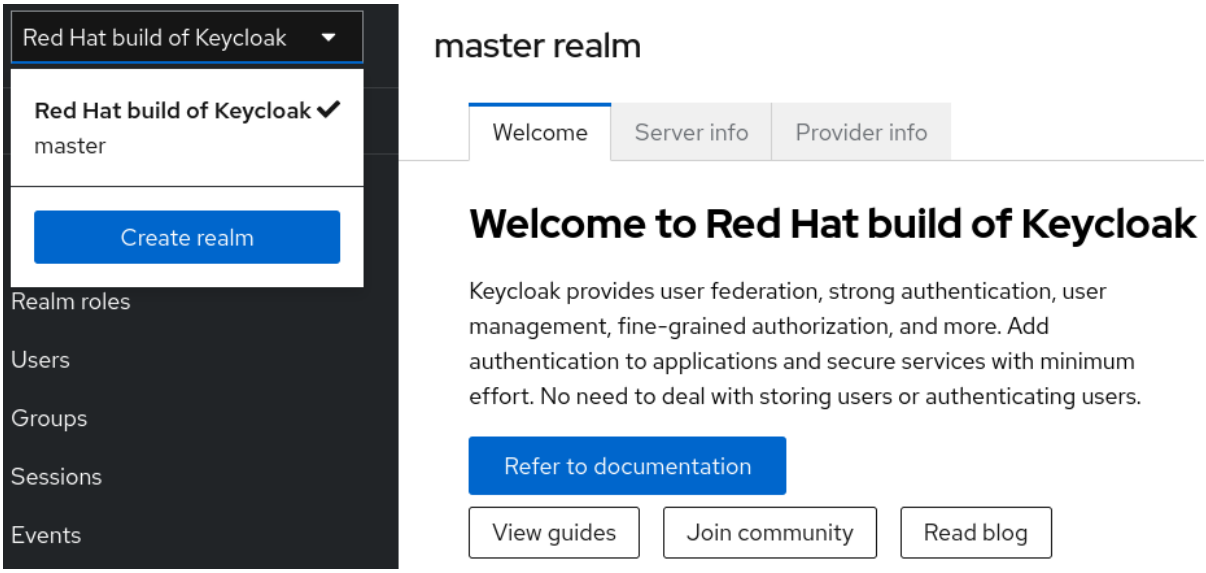
您可以创建一个域来提供管理空间，您可以在其中创建用户并授予他们使用应用程序的权限。首次登录时，您通常位于 *master* 域中，这是创建其他域的顶级域。

在决定您需要的域时，请考虑用户和应用程序所需的隔离类型。例如，您可以为公司员工和您客户的单独域创建一个域。您的员工将登录员工领域，只能访问内部公司应用程序。客户会登录客户领域，并且只能与面向客户的应用程序进行交互。

### 流程

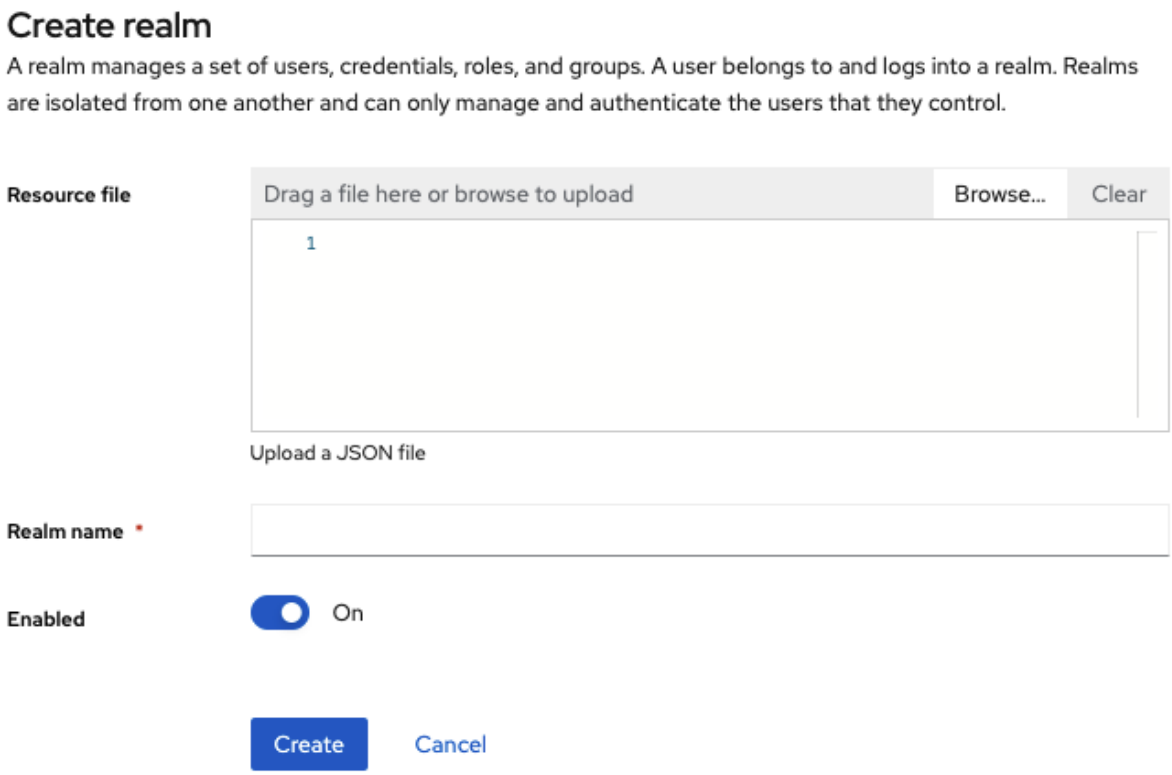
1. 单击 **master realm** 旁边的 **红帽构建的 Keycloak**，然后单击 **Create Realm**。

   **添加 realm 菜单**




2. 输入 realm 的名称。
3. 点 **Create**。

**创建 realm**




当前域现在设置为您刚才创建的域。您可以通过单击菜单中的 realm 名称在域之间切换。

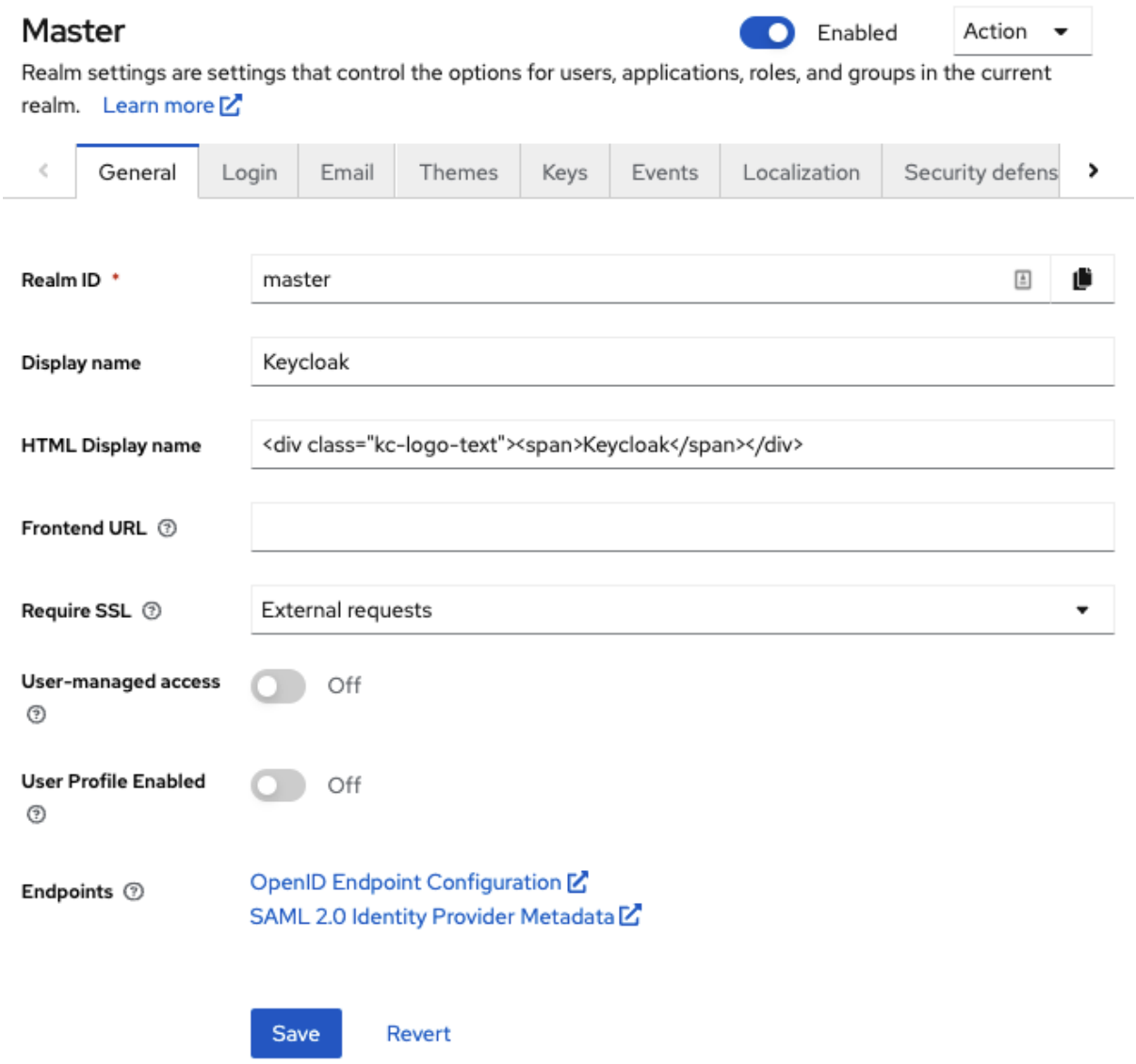
## 3.4. 为域配置 SSL

每个域都有一个关联的 SSL 模式，它定义了与域交互的 SSL/HTTPS 要求。与域交互的浏览器和应用程序遵循 SSL 模式定义的 SSL/HTTPS 要求，或者不能与服务器交互。

**流程**

1. 点菜单中的 **Realm settings**。
2. 点击**常规**标签。

**常规标签页**




3. 将 **Require SSL** 设置为以下 SSL 模式之一：
   - **外部请求用户可以在没有** SSL 的情况下与红帽构建的 Keycloak 交互，只要他们坚持使用私有 IP 地址，如 **localhost**、192.**127.0.0.1**、**10.x.x.x**、**192.168.x.x** 和 **172.16.x.x**。如果您试图在没有非专用 IP 地址的情况下访问红帽 Keycloak 构建，则会出现错误。
   - **没有** 红帽构建的 Keycloak 不需要 SSL。只有在您试验且不计划支持此部署时，此选择仅适用于开发。
   - **所有请求** 红帽构建的 Keycloak 需要 SSL 用于所有 IP 地址。

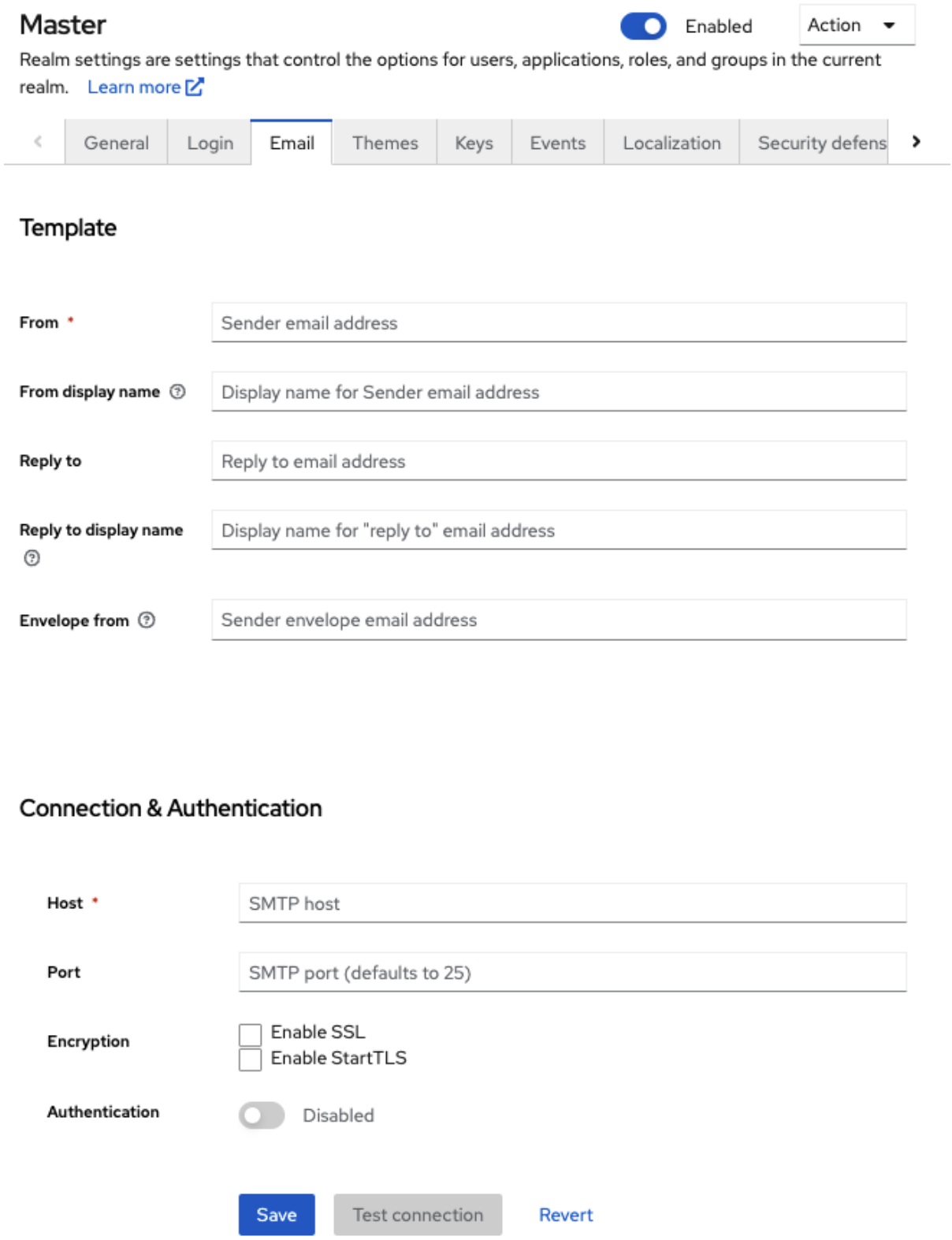
## 3.5. 为域配置电子邮件

红帽构建的 Keycloak 向用户发送电子邮件，以便在用户忘记密码时或需要收到有关服务器事件的通知时验证其电子邮件地址。要启用红帽构建的 Keycloak 来发送电子邮件，您可以使用 SMTP 服务器设置提供红帽构建的 Keycloak。

**流程**

1. 点菜单中的 **Realm settings**。
2. 单击 **Email** 选项卡。

   **电子邮件**标签页




3. 填写字段并根据需要切换交换机。

**模板**

from

From 表示用于发送的电子邮件的 From SMTP 标头使用的地址。

**从显示名称**

**从显示名称** 中，可以配置用户友好的电子邮件地址别名（可选）。如果没有设置 plain From 电子邮件地址，则会在电子邮件客户端中显示。

**回复**

回复表示用于 Reply-To SMTP-Header 用于邮件的地址（可选）。**如果没有设置 plain from email address，则会使用。**

**回复显示名称**

**回复显示名称 允许配置用户友好的电子邮件地址别名（可选）。如果没有设置 plain Reply To 电子邮件地址，则会显示。**

**信封来自**

envelope 表示用于发送邮件的 return-Path SMTP-Header 的 Bounce Address （可选）。

连接和验证

**主机**

host 表示用于发送电子邮件的 SMTP 服务器主机名。

**端口**

port 表示 SMTP 服务器端口。

Encryption

选择其中一个复选框来支持发送电子邮件以恢复用户名和密码，特别是在 SMTP 服务器位于外部网络中。您可能需要将端口更改为 465，这是 SSL/TLS 的默认端口。

**身份验证**

如果您的 SMTP 服务器需要身份验证，请将此开关设置为 ON。出现提示时，提供 Username 和 Password。Password 字段的值可以引用来自外部 密码库 的值。

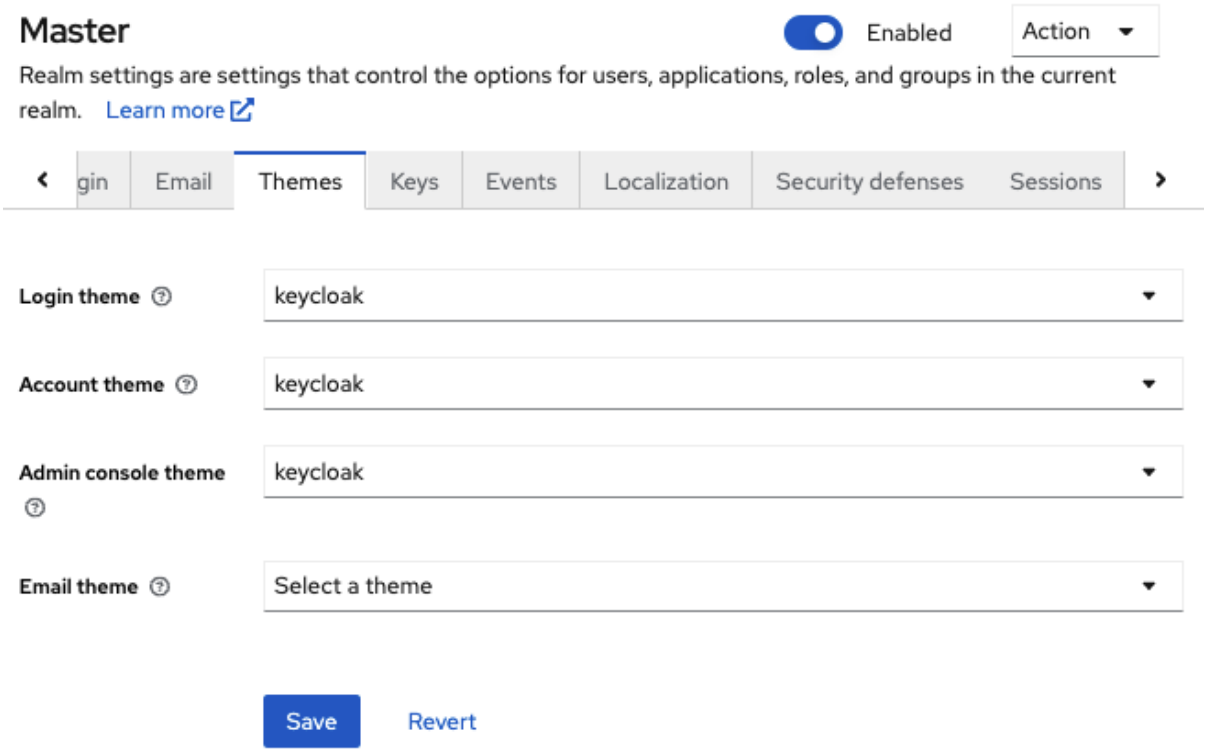
## 3.6. 配置主题

对于给定域，您可以使用它们更改红帽构建的 Keycloak 中任何 UI 的外观。

**流程**

1. 单击菜单中的 Realm settings。
2. 点 Themes 选项卡。

   themes 标签页




3. 选择您需要的每个 UI 类别的主题，然后单击保存。

登录主题

用户名密码条目、OTP 条目、新用户注册和其他与登录相关的类似屏幕。

帐户主题

用户用于管理其帐户的控制台。

管理控制台主题

红帽构建的 Keycloak 管理控制台的偏移。

电子邮件主题

每当红帽构建的 Keycloak 必须发送电子邮件，它会使用此主题中定义的模板来制作电子邮件。

**其他资源**

- 服务器开发人员指南 描述了如何创建新主题或修改现有的主题。

## 3.7. 启用国际化

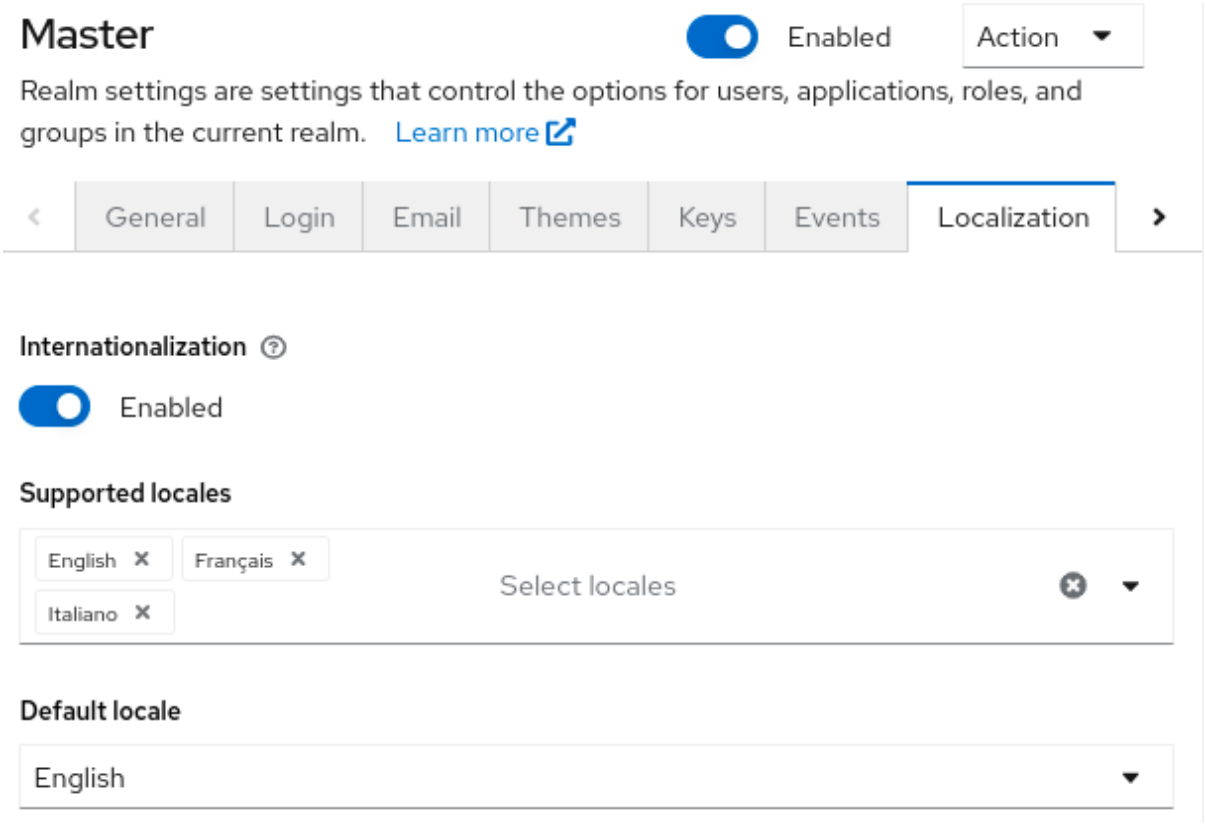
每个 UI 屏幕在 Red Hat build of Keycloak 中进行了国际化。默认语言是英语，但您可以选择您要支持的区域以及默认区域设置。

**流程**

1. 单击菜单中的 Realm Settings。
2. 点 Localization 选项卡。
3. 启用 国际化。

4. 选择支持的语言。

   本地化标签页




   用户下次登录时，该用户可以在登录页面上选择一个语言，以用于登录屏幕、帐户控制台和管理控制台。

其他资源

- 服务器开发人员指南 解释了如何提供额外的语言。主题提供的所有国际化文本可能会被 Localization 选项卡上域特定文本覆盖。

### 3.7.1. 用户区域设置选择

区域设置选择器供应商建议有关可用信息的最佳区域。但是，用户通常未知。因此，之前经过身份验证的用户的区域设置在持久的 Cookie 中记住。

选择区域设置的逻辑使用以下第一个可用：

- 用户选择 - 当用户使用下拉区域设置选择器选择了区域时
- 用户配置文件 - 当有经过身份验证的用户并且用户设置了首选区域设置时
- client selected - 由客户端使用 ui_locales 参数传递
- cookie - 浏览器中选择的最后一个区域设置
- 接受的语言 - 来自 Accept-Language 标头的区域设置
- 域默认值
- 如果以上都没有，请回退到英语

当用户通过身份验证时，会触发操作来更新前面提到的持久性 Cookie 中的区域设置。如果用户已在登录页面上通过区域设置主动切换区域，则此时也会更新用户区域设置。

如果要更改选择区域的逻辑，您可以选择创建自定义 LocaleSelectorProvider。详情请查看 服务器开发人员指南。

## 3.8. 控制登录选项

红帽构建的 Keycloak 包括几个内置登录页面功能。

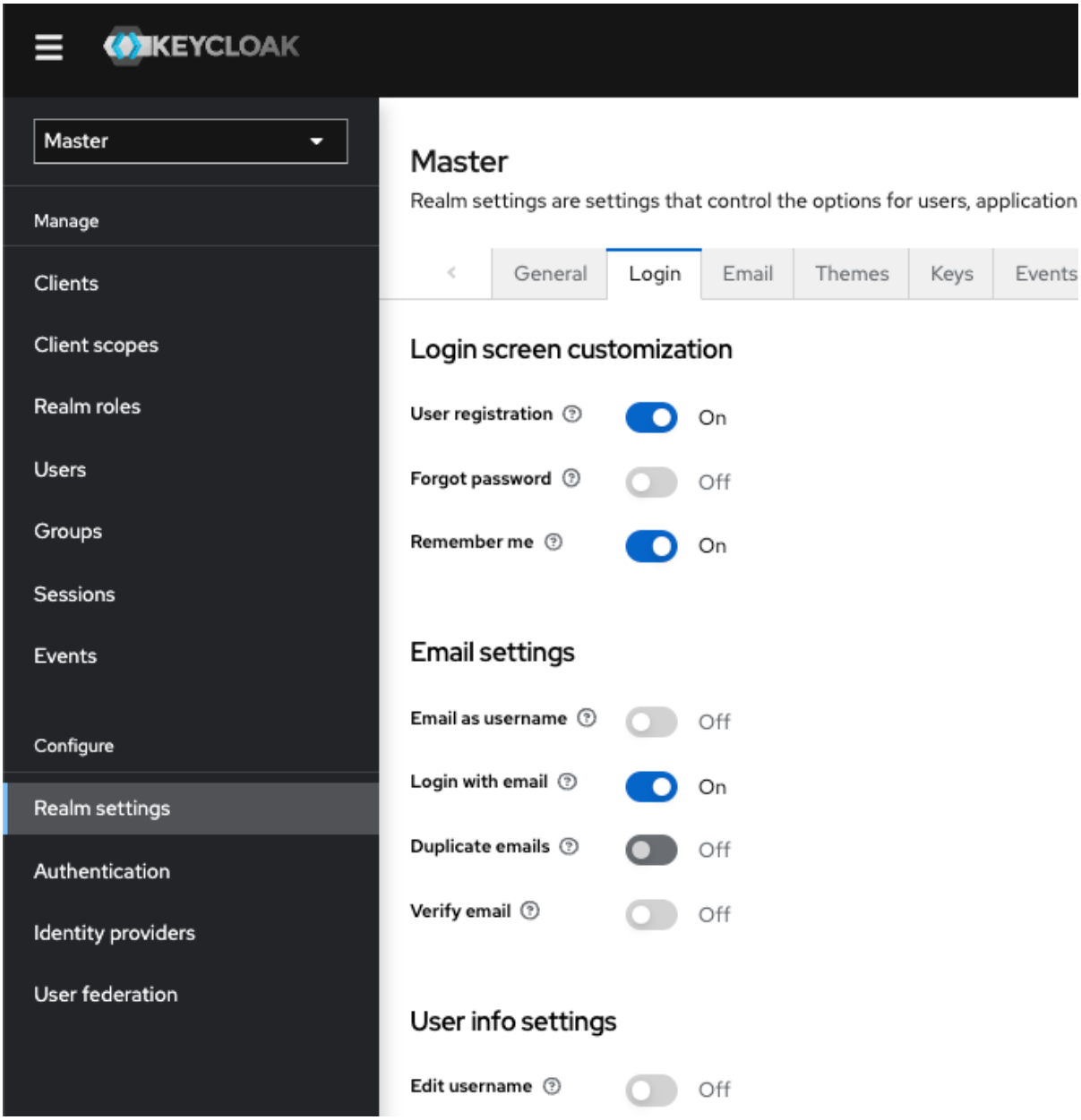
### 3.8.1. 启用忘记密码

如果启用 Forgot 密码，如果用户忘记了其密码或丢失其 OTP 生成器，用户可以重置其登录凭据。

**流程**

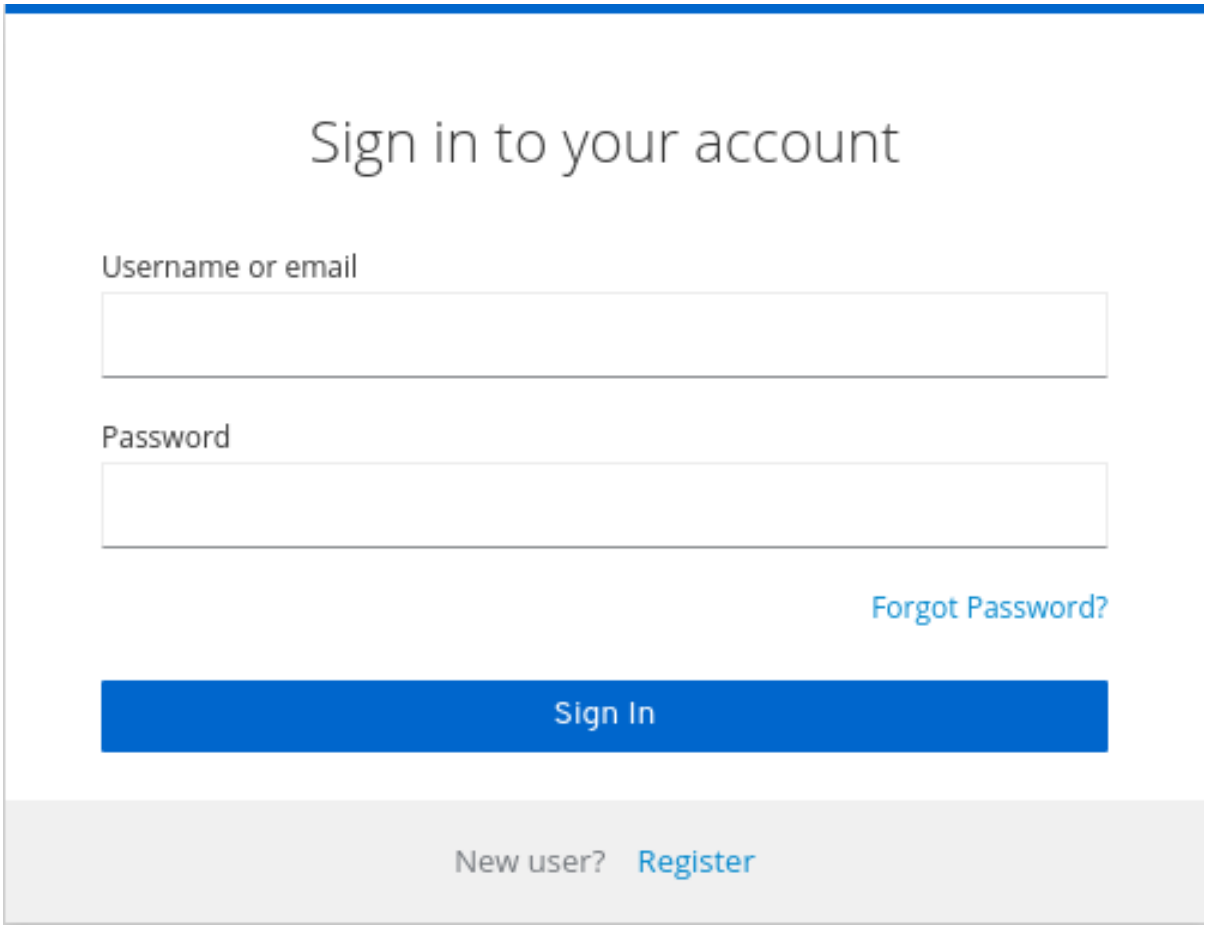
1. 点菜单中的 Realm settings。
2. 点 Login 选项卡。

Login 标签页



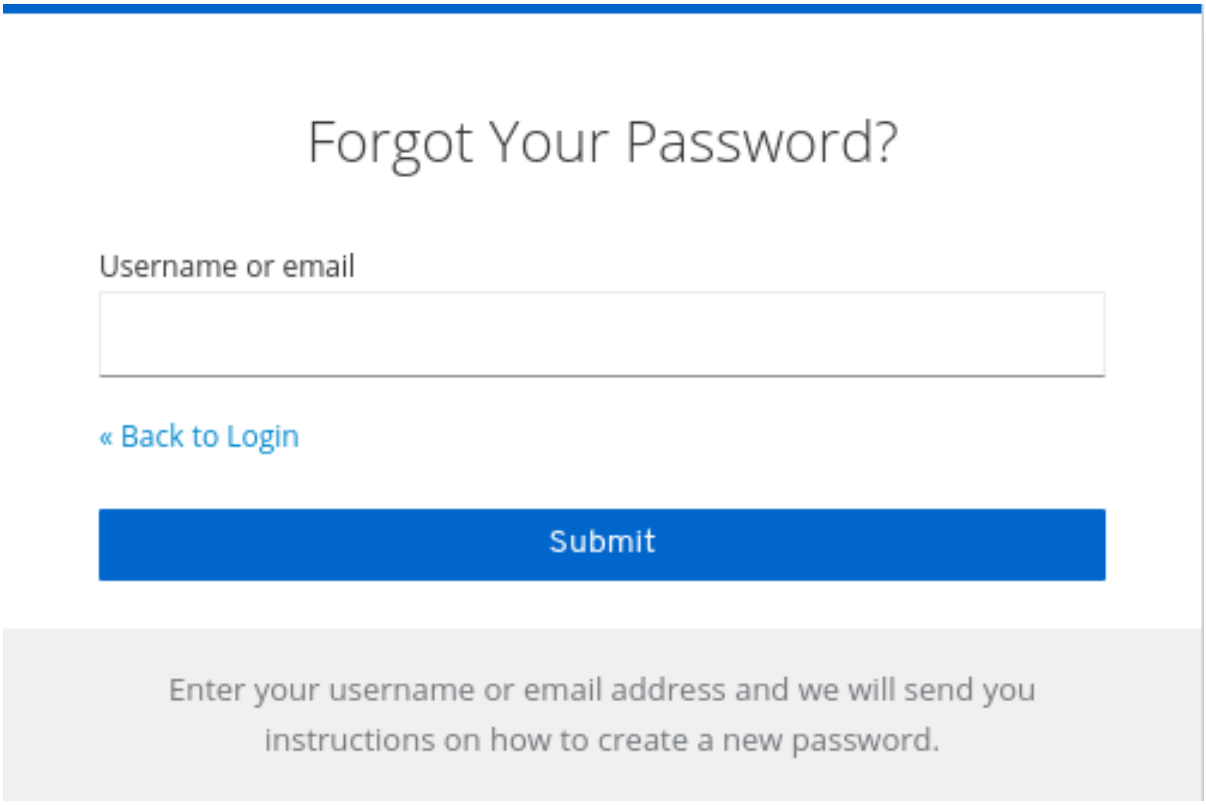

3. **将 Forgot 密码 切换为 ON。**
   **忘记密码？ 链接会显示在您的登录页面中。**

   **忘记密码链接**




4. **在 Email 选项卡中指定 Host 和 From，以便红帽构建的 Keycloak 能够发送重置电子邮件。**
5. **单击此链接，使用户可以输入其用户名或电子邮件地址，并收到一封电子邮件，以重置其凭据。**

**忘记密码页面**




**电子邮件中发送的文本可以配置。如需更多信息，请参阅 服务器开发人员指南。**

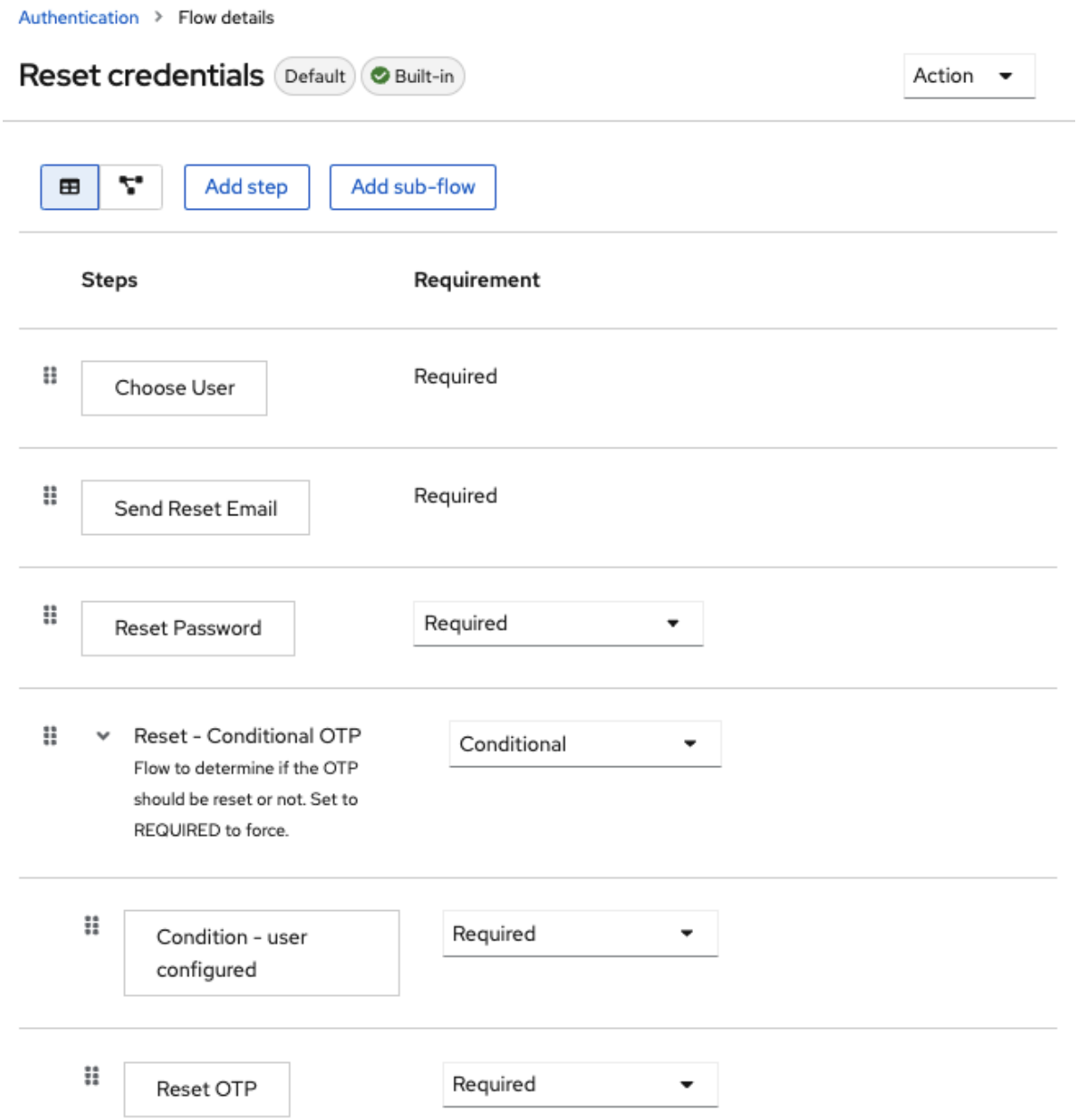
当用户点击电子邮件链接时，红帽构建的 Keycloak 会要求他们更新其密码，如果他们设置了 OTP 生成器，红帽构建的 Keycloak 会要求他们重新配置 OTP 生成器。根据您的机构的安全要求，您可能不希望用户通过电子邮件重置 OTP 生成器。

要更改此行为，请执行以下步骤：

**流程**

1. 点菜单中的 Authentication。
2. 点 Flows 选项卡。
3. 选择 Reset Credentials 流。

   **重置凭证流**

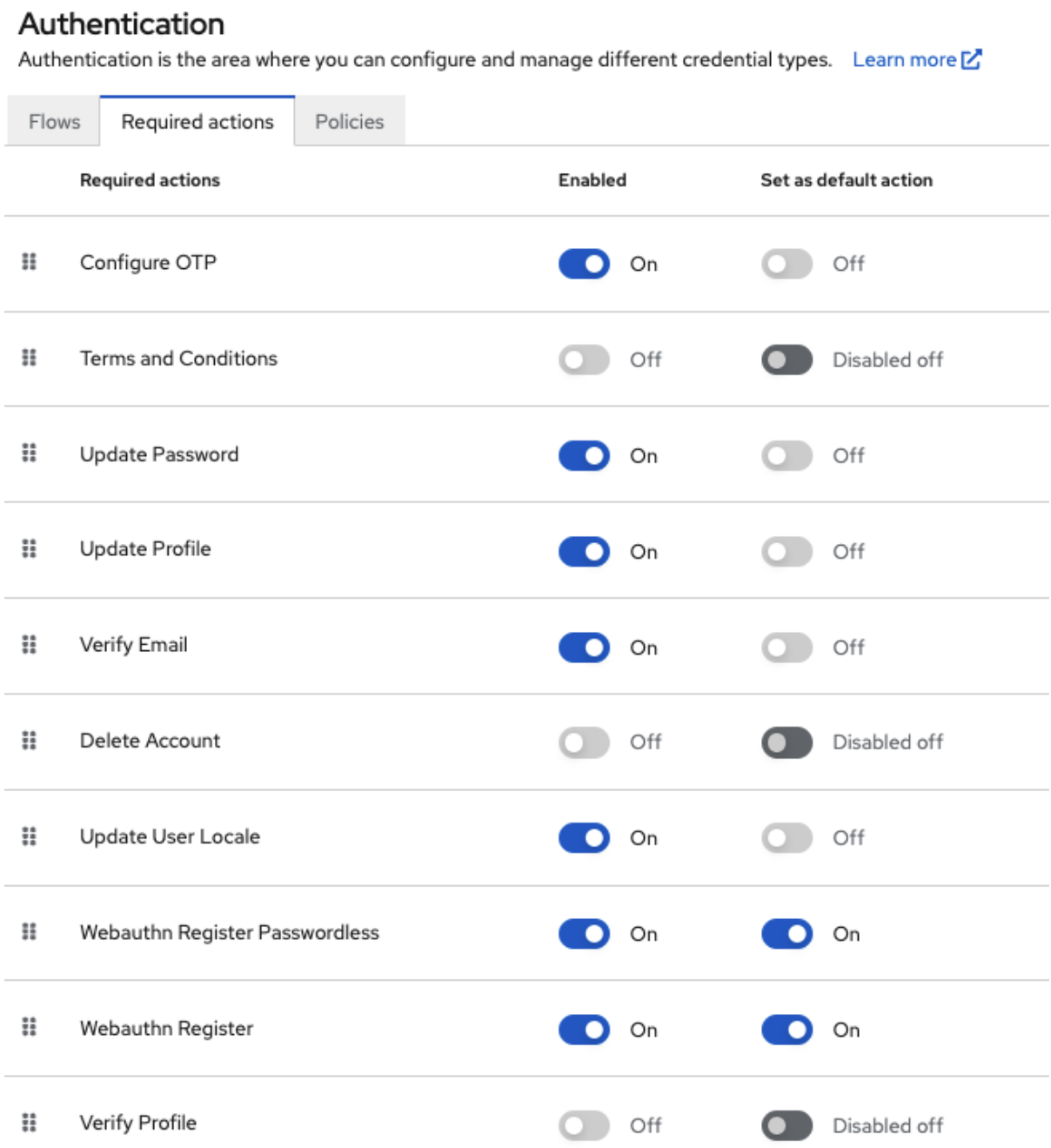



   如果您不想重置 OTP，请将 **Reset - Conditional OTP** 子流要求设置为 Disabled。

4. 点菜单中的 Authentication。

5. 点 Required actions 选项卡。
6. 确保启用了 Update Password。

**所需操作**




## 3.8.2. 启用 Remember Me

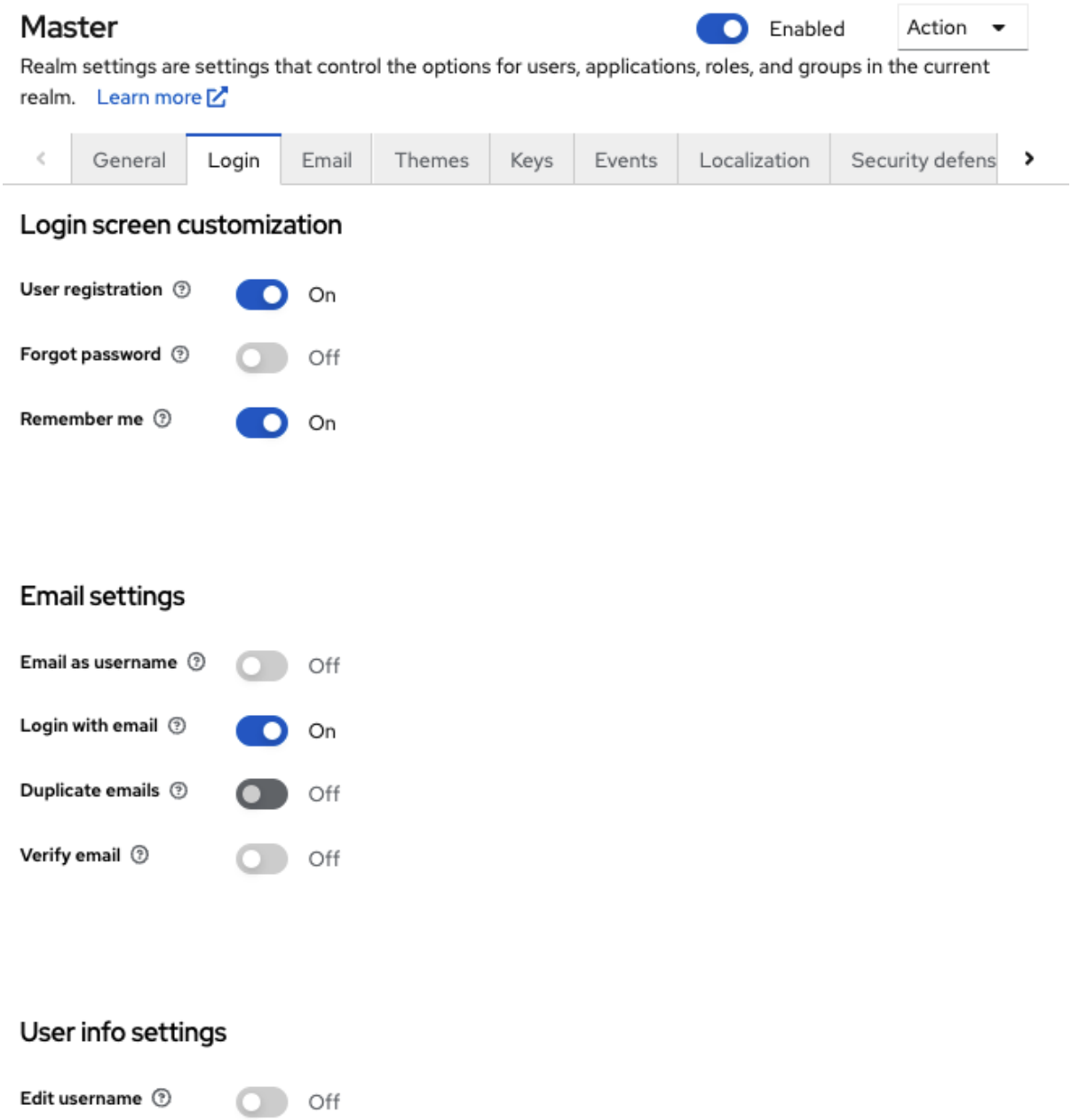
登录的用户关闭其浏览器会破坏其会话，并且该用户必须再次登录。如果该用户在登录时点 *Remember Me* 复选框，您可以设置 Red Hat build of Keycloak 来保持用户的登录会话打开。此操作将登录 Cookie 从仅会话 Cookie 变为持久性 Cookie。

**流程**

1. 点菜单中的 Realm settings。
2. 点 Login 选项卡。

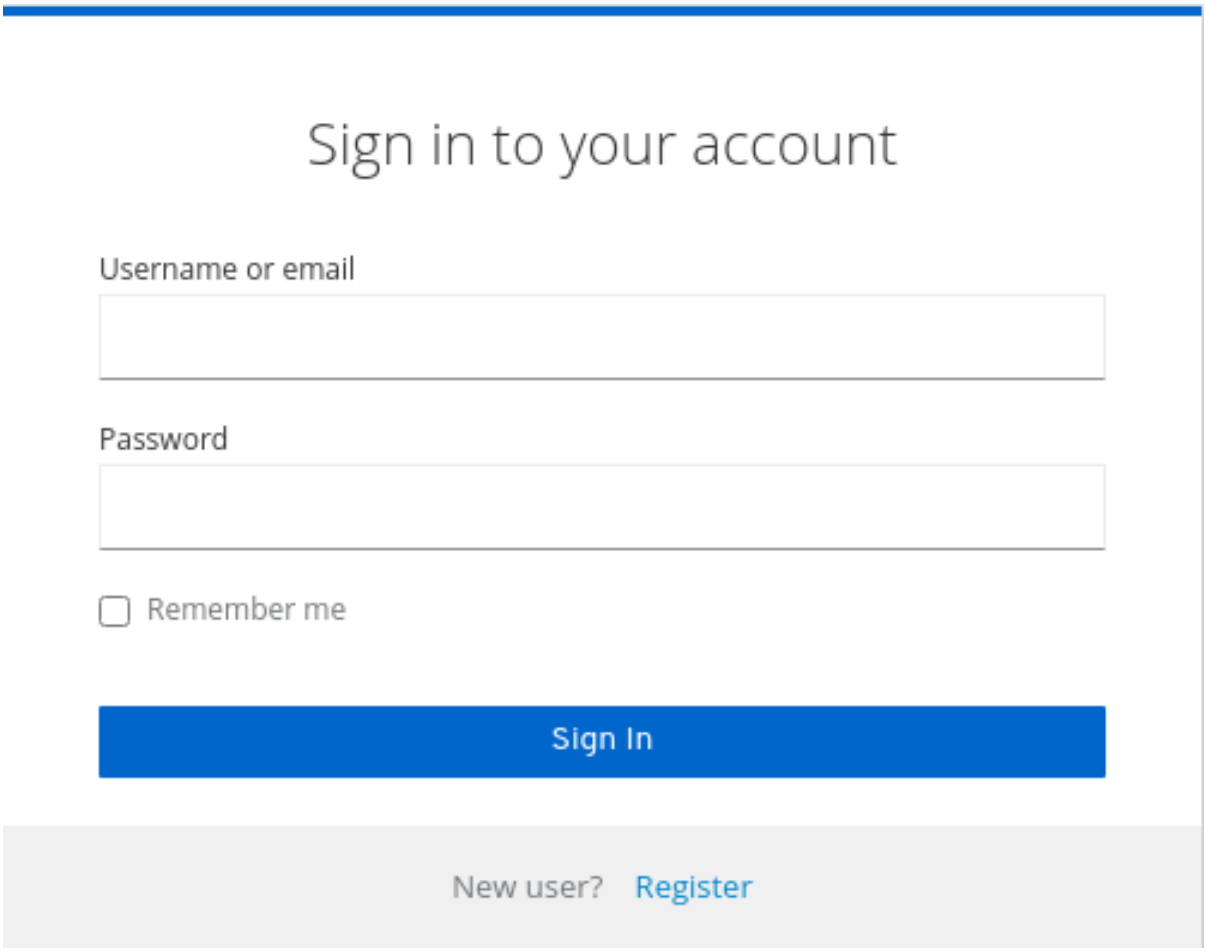
3. **将 Remember Me 开关切换为 On。**

Login 标签页




**在保存了此设置后，remember me 复选框显示在域的登录页面上。**

请记住我




### 3.8.3. ACR 到身份验证级别(LoA)映射

在域的登录设置中，您可以定义哪个 Authentication Context Class Reference (ACR) 值映射到哪个级别的 身份验证(LoA)。ACR 可以是任意值，而 LoA 必须是数字。acr 声明可以在 OIDC请求发送的声明 或 cr_values 参数中请求，它也包含在访问令牌和 ID 令牌中。映射的数字在身份验证流条件中使用。

如果特定客户端需要使用与 realm 不同的值，则映射也可以在客户端级别指定。但是，最佳实践是遵循域映射。


详情请查看 步骤验证和 官方 OIDC 规格。

### 3.8.4. 更新电子邮件工作流(UpdateEmail)

使用这个工作流，用户必须使用 UPDATE_EMAIL 操作来更改自己的电子邮件地址。

操作与单个电子邮件输入表相关联。如果域禁用了电子邮件验证，则此操作将允许在不验证的情况下更新电子邮件。如果域启用了电子邮件验证，则该操作会将电子邮件更新操作令牌发送到新的电子邮件地址，而不更改帐户电子邮件。只有操作令牌触发后才会完成电子邮件更新。

应用程序可以通过使用 UPDATE_EMAIL 作为 AIA （应用程序初始操作）来将其用户发送到电子邮件更新表单。

**注意**

UpdateEmail 是 技术预览，并不被支持。此功能默认为禁用。

使用 --features=preview 或 --features=update-email启动服务器

**注意**

如果您启用此功能，且您要从以前的版本迁移，请在域中启用 Update Email required 操作。否则，用户无法更新其电子邮件地址。

## 3.9. 配置 REALM 密钥

红帽构建的 Keycloak 使用的身份验证协议需要加密签名，有时加密。红帽构建的 Keycloak 使用非对称密钥对（一个私钥和公钥）来实现这一点。

红帽构建的 Keycloak 一次有一个活跃的密钥对，但也可以有多个被动密钥。主动密钥对用于创建新签名，而被动密钥对则可用于验证之前的签名。这样，可以定期轮转密钥，而无需向用户停机或中断。

创建域时，会自动生成密钥对和自签名证书。

**流程**

1. 点菜单中的 Realm settings。
2. 点 Keys。
3. 从过滤器下拉菜单中选择 Passive 键来查看被动密钥。
4. 从过滤器下拉菜单中选择 Disabled 键 来查看禁用的密钥。

密钥对可以具有 Active 状态，但仍然不能选择为该域的当前活动密钥对。所选用于签名的活动对会根据提供活跃密钥对的优先级排序的第一个密钥提供程序来选择。

### 3.9.1. 轮转密钥

我们建议您定期轮转密钥。首先，创建优先级高于现有活动密钥的新密钥。您可以使用相同的优先级创建新密钥，并将以前的密钥设置为被动。

有新密钥可用后，所有新令牌和 Cookie 将用新密钥签名。当用户向应用进行身份验证时，SSO cookie 将使用新签名进行更新。当刷新 OpenID Connect 令牌时，使用新密钥签名新令牌。最后，所有 Cookie 和令牌都使用新密钥，并在可以删除旧密钥时之后使用。

删除旧密钥的频率是安全性之间的权衡，并确保所有 Cookie 和令牌都已更新。考虑每三到六个月创建新密钥，并在创建新密钥后将旧密钥删除至两个月。如果用户在添加新密钥和删除的旧密钥之间的期间不活跃，则该用户必须重新验证。

轮转密钥也适用于离线令牌。为确保它们已更新，应用程序需要在删除旧密钥前刷新令牌。

### 3.9.2. 添加生成的密钥对

使用这个流程生成密钥对，包括自签名证书。

**流程**

1. 在管理控制台中选择域。
2. 点菜单中的 Realm settings。
3. 点 Keys 选项卡。
4. 点 Providers 选项卡。
5. 单击 Add provider，再选择 rsa 生成的。
6. 在 Priority 字段中输入数字。此数字决定了新密钥对是否成为活动密钥对。最高数字使密钥对处于活动状态。
7. 为 AES Key size 选择一个值。
8. 点击 Save。

更改供应商的优先级不会导致密钥被重新生成，但如果您想要更改密钥大小，则可以编辑提供程序，并将生成新密钥。

### 3.9.3. 通过提取证书来轮转密钥

您可以通过从 RSA 生成的密钥对中提取证书并在新密钥存储中使用该证书来轮转密钥。

**先决条件**

- 生成的密钥对

**流程**

1. 在管理控制台中选择域。
2. 单击 Realm Settings。
3. 点 Keys 选项卡。
   此时会出现 Active keys 列表。
4. 在带有 RSA 密钥的行中，单击 Public Keys 下的 Certificate。
   证书以文本形式出现。
5. 将证书保存到文件中，并将其放在这些行中。

```
----Begin Certificate----
<Output>
----End Certificate----
```

6. 使用 keytool 命令将密钥文件转换为 PEM 格式。
7. 从密钥存储中删除当前的 RSA 公钥证书。

```
keytool -delete -keystore <keystore>.jks -storepass <password> -alias <key>
```

8. 将新证书导入到密钥存储中

```
keytool -importcert -file domain.crt -keystore <keystore>.jks -storepass <password>  -
alias <key>
```

9. 重新构建应用。

```
mvn clean install wildfly:deploy
```

### 3.9.4. 添加现有的密钥对和证书

要添加密钥对和在其他位置获取的证书，请选择 **Providers**，然后从下拉菜单中选择 **rsa**。您可以更改优先级，以确保新密钥对成为活跃的密钥对。

**先决条件**

- 私钥文件。文件必须采用 PEM 格式。

**流程**

1. 在管理控制台中选择域。
2. 单击 Realm settings。
3. 点 Keys 选项卡。
4. 点 Providers 选项卡。
5. 单击 Add provider，再选择 rsa。
6. 在 Priority 字段中输入数字。此数字决定了新密钥对是否成为活动密钥对。
7. 点 Private RSA Key 旁边的 Browse... 来上传私钥文件。
8. 如果您的私钥有一个签名证书，请点击 Browse... 旁边的 X509 Certificate 来上传证书文件。如果没有上传证书，Red Hat build of Keycloak 会自动生成一个自签名证书。
9. 点击 Save。

### 3.9.5. 从 Java Keystore 中载入密钥

要添加存储在主机上的 Java 密钥存储文件中的密钥对和证书，请选择 **Providers**，然后从下拉菜单中选择 **java-keystore**。您可以更改优先级，以确保新密钥对成为活跃的密钥对。

要加载关联的证书链，必须将其导入到 Java 密钥存储文件中，并使用 相同的键 别名来加载密钥对。

**流程**

1. 在管理控制台中选择域。
2. 点菜单中的 Realm settings。
3. 点 Keys 选项卡。
4. 点 Providers 选项卡。
5. 单击 Add provider，再选择 java-keystore。

6. 在 Priority 字段中输入数字。此数字决定了新密钥对是否成为活动密钥对。
7. 为 Keystore 输入一个值。
8. 输入 Keystore Password 的值。
9. 输入 Key Alias 的值。
10. 输入 Key Password 的值。
11. 点 Save。

### 3.9.6. 使密钥被动

**流程**

1. 在管理控制台中选择域。
2. 点菜单中的 Realm settings。
3. 点 Keys 选项卡。
4. 点 Providers 选项卡。
5. 单击您要进行被动的密钥的供应商。
6. 将 Active 切换到 Off。
7. 点击 Save。

### 3.9.7. 禁用密钥

**流程**

1. 在管理控制台中选择域。
2. 点菜单中的 Realm settings。
3. 点 Keys 选项卡。
4. 点 Providers 选项卡。
5. 单击您要进行被动的密钥的供应商。
6. 将 Enabled 切换到 Off。
7. 点击 Save。

### 3.9.8. 被破坏的密钥

红帽构建的 Keycloak 具有仅本地存储的签名密钥，它们永远不会与客户端应用程序、用户或其他实体共享。但是，如果您认为您的域签名密钥已被破坏，您应当首先生成新密钥对，如上所述，然后立即删除被入侵的密钥对。

或者，您可以从 Providers 表中删除供应商。

**流程**

1. 点菜单中的 Clients。
2. 点 security-admin-console。
3. 向下滚动到 Access settings 部分。
4. 填写 Admin URL 字段。
5. 点 Advanced 选项卡。
6. 在 Revocation 部分中，点 Set to now。
7. 单击 Push。

推送 not-before 策略可确保客户端应用程序不接受由被入侵密钥签名的现有令牌。客户端应用程序被强制从红帽构建的 Keycloak 下载新密钥对，因此由被破坏的密钥签名的令牌无效。

**注意**

REST 和机密客户端必须设置 Admin URL，以便红帽构建的 Keycloak 能够向推送的 not-before 策略请求发送客户端。

# 第 4 章 使用外部存储

机构可以具有包含信息、密码和其他凭证的数据库。通常，您无法将现有数据存储迁移到红帽构建的 Keycloak 部署，以便红帽构建的 Keycloak 可以联合现有的外部用户数据库。红帽构建的 Keycloak 支持 LDAP 和 Active Directory，但您也可以使用红帽构建的 Keycloak User Storage SPI 对任何自定义用户数据库进行代码扩展。

当用户尝试登录时，红帽构建的 Keycloak 会检查该用户的存储以查找该用户。如果红帽构建的 Keycloak 找不到用户，红帽构建的 Keycloak 会迭代域的每个用户存储供应商，直到找到匹配项为止。然后，来自外部数据存储的数据会映射到红帽构建的 Keycloak 运行时使用的标准用户模型。然后，这个用户模型会映射到 OIDC 令牌声明和 SAML 断言属性。

外部数据库很少有支持红帽构建 Keycloak 的所有功能所需的数据，因此用户存储供应商可以选择在红帽构建的 Keycloak 用户数据存储中存储项目。提供商可以在本地导入用户，并定期与外部数据存储进行同步。这种方法取决于提供程序的功能和配置提供程序。例如，您的外部用户数据存储可能不支持 OTP。OTP 可以由红帽构建的 Keycloak 处理和存储，具体取决于供应商。

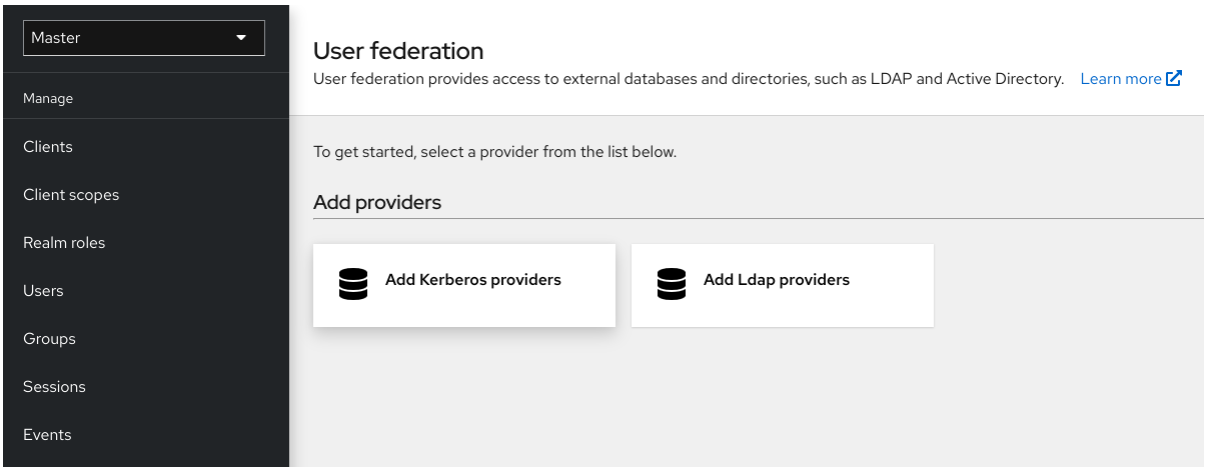
## 4.1. 添加供应商

要添加存储供应商，请执行以下步骤：

**流程**

1. 点菜单中的 User Federation。

   用户联邦




2. 从列出的卡中选择供应商类型卡。
   红帽构建的 Keycloak 会带您进入该供应商的配置页面。

## 4.2. 处理供应商失败

如果 User Storage Provider 失败，您可能无法在管理控制台中登录和查看用户。在使用存储供应商查找用户时，红帽构建的 Keycloak 不会检测到失败，因此它会取消调用。如果您的存储供应商具有在用户查找过程中失败的存储提供程序，则登录或用户查询会失败，并显示例外，且不会切换到下一个配置的供应商。

Red Hat build of Keycloak 会首先搜索本地的 Keycloak 用户数据库构建，以便在任何 LDAP 或自定义用户存储供应商前解析用户。考虑创建一个存储在本地红帽构建的 Keycloak 用户数据库中的管理员帐户，以防连接到 LDAP 和后端的问题。

每个 LDAP 和自定义用户存储提供程序在其 Admin Console 页面中都有一个 启用 切换。在执行查询时禁用 User Storage Provider 会跳过供应商，因此您可以使用优先级较低的不同供应商中的用户帐户查看和登录。如果您的供应商使用导入策略并被禁用，则导入的用户仍可以只读模式查找。

当 Storage Provider 查找失败时，红帽构建的 Keycloak 不会故障切换，因为用户数据库通常在它们之间有重复的用户名或重复电子邮件。重复用户名和电子邮件可能会导致问题，因为当管理员希望从另一个数据存储加载时，用户会从一个外部数据存储加载。

## 4.3. 轻量级目录访问协议(LDAP)和 ACTIVE DIRECTORY

红帽构建的 Keycloak 包括 LDAP/AD 供应商。您可以在一个红帽构建的 Keycloak 域中进行多个不同的 LDAP 服务器，并将 LDAP 用户属性映射到红帽构建的 Keycloak 通用用户模型中。

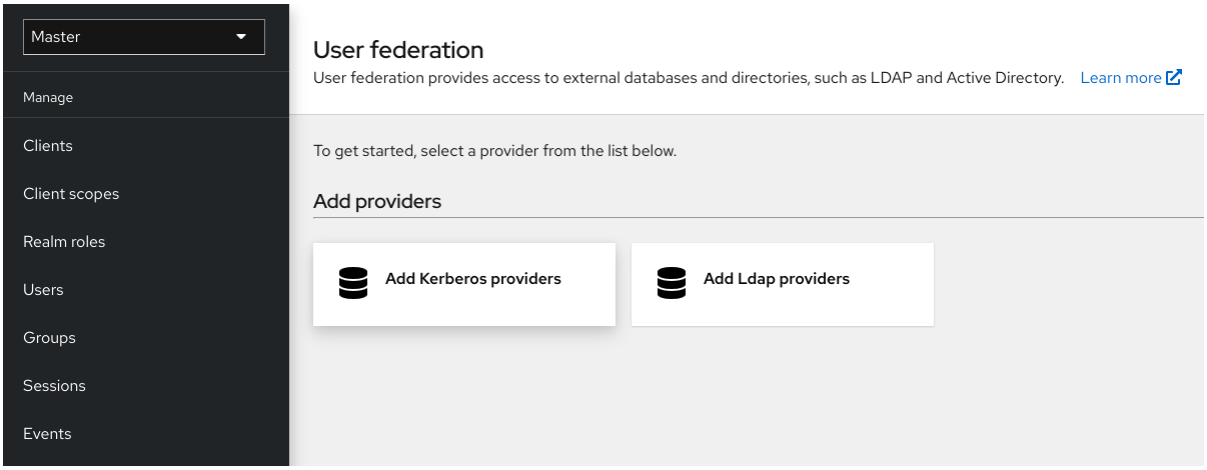
默认情况下，红帽构建的 Keycloak 会映射用户名、电子邮件、名字和用户帐户姓氏，但您也可以配置额外的 映射。红帽构建的 Keycloak 的 LDAP/AD 供应商支持使用 LDAP/AD 协议和存储、编辑和同步模式进行密码验证。

### 4.3.1. 配置联邦 LDAP 存储

**流程**

1. 点菜单中的 User Federation。

   用户联邦




2. 单击 Add LDAP provider。
   红帽构建的 Keycloak 会进入 LDAP 配置页面。

### 4.3.2. 存储模式

红帽构建的 Keycloak 将用户从 LDAP 导入到红帽 Keycloak 用户数据库的本地构建。用户数据库的这个副本会按需同步，或通过定期后台任务同步。同步密码存在一个例外。红帽构建的 Keycloak 永不导入密码。密码验证总是在 LDAP 服务器上发生。

同步的优点在于，所有红帽构建的 Keycloak 功能都高效工作，因为任何需要的额外用户数据都会存储在本地。缺点是，当红帽构建的 Keycloak 首次查询特定用户时，红帽构建的 Keycloak 都会执行对应的数据库插入。

您可以将导入与 LDAP 服务器同步。当 LDAP 映射器始终从 LDAP 而不是数据库读取特定属性时，导入同步是不必要的。

您可以在红帽构建的 Keycloak 中使用 LDAP，而无需将用户导入到红帽 Keycloak 用户数据库中。LDAP 服务器备份红帽构建的 Keycloak 运行时使用的通用用户模型。如果 LDAP 不支持红帽构建的 Keycloak 功能需要的数据，该功能将无法正常工作。这种方法的优点是，您没有将 LDAP 用户的副本导入并同步到红帽 Keycloak 用户数据库的资源使用情况。

LDAP 配置页面中的 Import Users 开关控制此存储模式。要导入用户，请将此开关切换到 ON。

注意

如果禁用 Import Users，则无法将用户配置集属性保存到红帽构建的 Keycloak 数据库中。另外，除了映射到 LDAP 的用户配置文件元数据外，您无法保存元数据。这个元数据可以包括角色映射、组映射和其他元数据，具体取决于 LDAP 映射器的配置。

当您试图更改非 LDAP 映射用户数据时，无法进行用户更新。例如，除非用户的 启用 标记映射到 LDAP 属性，否则您无法禁用 LDAP 映射的用户。

### 4.3.3. 编辑模式

用户和管理员可以通过 管理控制台 修改用户元数据、用户以及管理员。LDAP 配置页面中的 Edit Mode 配置定义了用户的 LDAP 更新特权。

READONLY

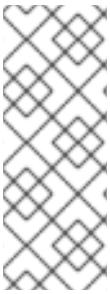
: 您无法更改用户名、电子邮件、名字、姓氏和其他映射的属性。红帽构建的 Keycloak 会在用户尝试更新这些字段时显示错误。不支持密码更新。

WRITABLE

: 您可以更改用户名、电子邮件、名字、姓氏和其他映射的属性和密码，并将它们自动与 LDAP 存储同步。

未同步

: 红帽构建的 Keycloak 存储了红帽构建的 Keycloak 本地存储中的用户名、电子邮件、名字、姓氏和密码，因此管理员必须将此数据同步回 LDAP。在这个模式中，红帽构建的 Keycloak 部署可以在只读 LDAP 服务器中更新用户元数据。这个选项也适用于将用户从 LDAP 导入到 Keycloak 用户数据库的本地红帽构建。



注意

当红帽构建的 Keycloak 创建 LDAP 供应商时，红帽构建的 Keycloak 还会创建一组初始 LDAP 映射器。红帽构建的 Keycloak 根据 Vendor、Edit Mode 和 Import Users 交换机的组合来配置这些映射程序。例如，当编辑模式是 UNSYNCED 时，红帽构建的 Keycloak 会将映射器配置为从数据库读取特定用户属性，而不是从 LDAP 服务器读取。但是，如果您稍后更改了编辑模式，则映射器的配置不会改变，因为无法检测 UNSYNCED 模式中的配置更改。在创建 LDAP 提供商时决定 Edit Mode本备注也适用于 Import Users 开关。

### 4.3.4. 其他配置选项

控制台显示名称

: 在管理控制台中显示的供应商名称。

优先级

: 查找用户或添加用户时供应商的优先级。

同步注册

: 如果您希望红帽构建的 Keycloak 创建的新用户添加到 LDAP，请将此切换切换到 ON。

允许 Kerberos 身份验证

使用从 LDAP 调配的用户数据，在域中启用 Kerberos/SPNEGO 身份验证。如需更多信息，请参阅 Kerberos 部分。

其他选项

将鼠标指针悬停在管理控制台中工具提示上，以查看这些选项的更多详细信息。

### 4.3.5. 通过 SSL 连接到 LDAP

当您配置到 LDAP 存储的安全连接 URL （例如ldaps://myhost.com:636）时，红帽构建的 Keycloak 使用 SSL 与 LDAP 服务器通信。在红帽构建的 Keycloak 服务器端配置信任存储，以便红帽构建的 Keycloak 可以信任到 LDAP 的 SSL 连接 - 请参阅 配置 Truststore 章节。

Use Truststore SPI 配置属性已弃用。它通常应保留为 Always。

### 4.3.6. 将 LDAP 用户同步到红帽构建的 Keycloak

如果您设置了 Import Users 选项，LDAP Provider 会处理将 LDAP 用户导入到红帽 Keycloak 本地数据库的红帽构建中。当用户第一次登录时，或作为用户查询的一部分返回（例如，使用管理控制台中的 search 字段），LDAP 供应商会将 LDAP 用户导入到红帽 Keycloak 数据库的构建中。在身份验证过程中，会验证 LDAP 密码。

如果要将所有 LDAP 用户同步到红帽构建的 Keycloak 数据库，请在 LDAP 供应商配置页面中配置和启用 Sync Settings。

存在两种类型的同步：

定期完全同步

这个类型将所有 LDAP 用户同步到红帽构建的 Keycloak 数据库。在红帽构建的 Keycloak 中已存在 LDAP 用户，但在 LDAP 中有所不同，直接在红帽构建的 Keycloak 数据库中更新。

定期更改的用户同步

同步时，红帽构建的 Keycloak 只会创建或更新用户在最后一次同步后创建或更新的用户。

同步的最佳方法是，在第一次创建 LDAP 提供程序时单击 Synchronize all users，然后设置更改的用户定期同步。

### 4.3.7. LDAP 映射器

LDAP 映射器是由 LDAP 提供程序触发 的监听程序。它们提供 LDAP 集成的另一个扩展点。在以下情况下触发 LDAP 映射器：

- 用户使用 LDAP 登录。
- 用户最初注册。
- 管理控制台查询用户。

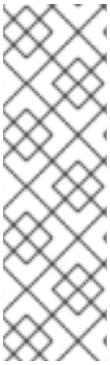
当您创建 LDAP Federation 供应商时，红帽构建的 Keycloak 会自动为 此 供应商提供一组映射程序。这个集合可由用户更改，用户也可以开发映射程序或更新/删除现有的映射程序。

用户属性映射程序

这个映射程序指定将哪些 LDAP 属性映射到红帽构建的 Keycloak 用户的属性。例如，您可以将 mail LDAP 属性配置为红帽构建的 Keycloak 数据库中的 email 属性。对于这个映射程序实现，始终有一个一对一的映射。

fullName Mapper

这个 mapper 指定用户的全名。红帽构建的 Keycloak 将名称保存在 LDAP 属性中（通常为 cn），并将名称映射到红帽构建的 Keycloak 数据库中的 firstName 和 lastname 属性。使用 cn 包含用户的完整名称对于 LDAP 部署很常见。



注意

当您在红帽构建的 Keycloak 和 Sync Registrations 中为 LDAP 供应商注册新用户时，fullName 映射器允许回退到用户名。在使用 Microsoft Active Directory (MSAD) 时，这个回退很有用。MSAD 的常见设置是将 cn LDAP 属性配置为 fullName，同时使用 cn LDAP 属性作为 LDAP 供应商配置中的RDN LDAP Attribute。使用这个设置时，红帽构建的 Keycloak 会返回用户名。例如，如果您创建了红帽构建的 Keycloak 用户 "john123"，并将 firstName 和 lastName 留空，那么 fullname 映射器会将 "john123" 保存为 LDAP 中的 cn 的值。当您为 firstName 和 lastName 输入 "John Doe" 时，fullname mapper 会将 LDAP cn 更新至 "John Doe" 值，因为回退到用户名是不必要的。

硬编码的属性映射程序

这个映射程序为与 LDAP 链接的每个红帽构建的 Keycloak 用户添加一个硬编码的属性值。这个映射程序也可以强制为 enabled 或 emailVerified 用户属性强制使用值。

角色映射程序

这个映射程序配置从 LDAP 到红帽构建的 Keycloak 角色映射的角色映射。单个角色映射器可以将 LDAP 角色（通常是 LDAP 树的特定分支中的组）映射到与指定客户端的域角色或客户端角色对应的角色。您可以为同一 LDAP 供应商配置更多角色映射程序。例如，您可以指定从 ou=main,dc=example,dc=org map 到 realm 角色映射下的组的角色映射，以及 ou=finance,dc=example,dc=org map 到客户端 finance 的客户端角色映射的角色映射。

硬编码的角色映射映射程序

这个映射程序向来自 LDAP 供应商的每个红帽构建的 Keycloak 用户授予指定的红帽构建的 Keycloak 角色。

组群映射程序

这个映射程序将 LDAP 组从 LDAP 树的分支映射到红帽构建的 Keycloak 中的组。这个映射程序还会将用户组映射从 LDAP 传播到红帽构建的 Keycloak 中的用户组映射中。

MSAD 用户帐户映射程序

这个映射程序特定于 Microsoft Active Directory (MSAD)。它可以将 MSAD 用户帐户状态集成到红帽构建的 Keycloak 帐户状态，如启用的帐户或过期密码。这个映射程序使用 userAccountControl、和 pwdLastSet LDAP 属性，特定于 MSAD，不是 LDAP 标准。例如，如果pwdLastSet 的值为 0，则红帽构建的 Keycloak 用户必须更新其密码。结果是添加到用户的 UPDATE_PASSWORD 所需的操作。如果 userAccountControl 的值为 514 （禁用帐户），则禁用 Keycloak 用户的红帽构建。

证书映射程序

这个映射器映射 X.509 证书。红帽构建的 Keycloak 将其与 X.509 身份验证和完整证书结合使用，以 PEM 格式 作为身份源。这个映射程序的行为与 User Attribute Mapper 类似，但红帽构建的 Keycloak 可以过滤存储 PEM 或 DER 格式的证书的 LDAP 属性。使用这个 映射程序启用 Always Read Value from LDAP。

将基本红帽构建的 Keycloak 用户属性（如用户名、名名、姓氏和电子邮件）映射到对应的 LDAP 属性的用户属性映射器。您可以扩展它们，并提供自己的额外属性映射。管理控制台提供了工具提示，以帮助配置对应的映射程序。

### 4.3.8. 密码散列

当红帽构建的 Keycloak 更新密码时，红帽构建的 Keycloak 以纯文本格式发送密码。此操作与更新内置红帽构建的 Keycloak 数据库中的密码不同，红帽构建的 Keycloak 哈希和 salt 密码会将其发送到数据库。对于 LDAP，红帽构建的 Keycloak 依赖于 LDAP 服务器来哈希和 salt 密码。

默认情况下，LDAP 服务器（如 MSAD、RHDS 或 FreeIPA 哈希）和 salt 密码。其他 OpenLDAP 或 ApacheDS 等 LDAP 服务器以纯文本形式存储密码，除非您使用 *LDAPv3 密码修改扩展操作*，如 RFC3062 所述。在 LDAP 配置页面中启用 LDAPv3 密码修改扩展操作。详情请查看 LDAP 服务器文档。

**警告**

使用 ldapsearch 和 base64 解码userPassword 属性值，始终通过检查更改的目录条目来验证用户密码是否被正确哈希，而不是以纯文本形式存储。

### 4.3.9. 故障排除

对于 org.keycloak.storage.ldap 类别，可以将日志级别增加到 TRACE。使用此设置时，许多日志记录消息都会发送到 TRACE 级别中的服务器日志，包括所有查询的日志记录到 LDAP 服务器以及用于发送查询的参数。在用户论坛或 JIRA 上创建任何 LDAP 问题时，请考虑使用启用的 TRACE 日志记录附加服务器日志。如果太大，良好的替代方案是将来自服务器日志的代码片段仅包含包含消息（在操作过程中添加到日志中），这会导致您出现问题。

- 当您创建 LDAP 供应商时，从以下位置开始，服务器日志中会出现一条信息：

```
Creating new LDAP Store for the LDAP storage provider: ...
```

它显示了您的 LDAP 供应商的配置。在询问问题或报告错误之前，最好包含此消息来显示您的 LDAP 配置。最终，可以用一些占位符值替换您不希望包含的一些配置更改。一个例子是 bindDn=some-placeholder。对于 connectionUrl，也可以自由替换它，但通常情况下包括使用的协议(ldap 和 ldaps)会很有用。同样，包含 LDAP 映射器配置的详情，这在 DEBUG 级别会显示如下信息：

```
Mapper for provider: XXX, Mapper name: YYY, Provider: ZZZ ...
```

请注意，这些消息仅显示启用了 DEBUG 日志记录。

- 要跟踪性能或连接池问题，请考虑将 LDAP 提供程序的属性 Connection Pool Debug Level 的值设置为值 all。这将通过包含的日志记录为 LDAP 连接池添加大量其他消息到服务器日志中。这可用于跟踪与连接池或性能相关的问题。

**注意**

更改连接池配置后，您可能需要重启红帽构建的 Keycloak 服务器，以强制实施 LDAP 供应商连接的重新初始化。

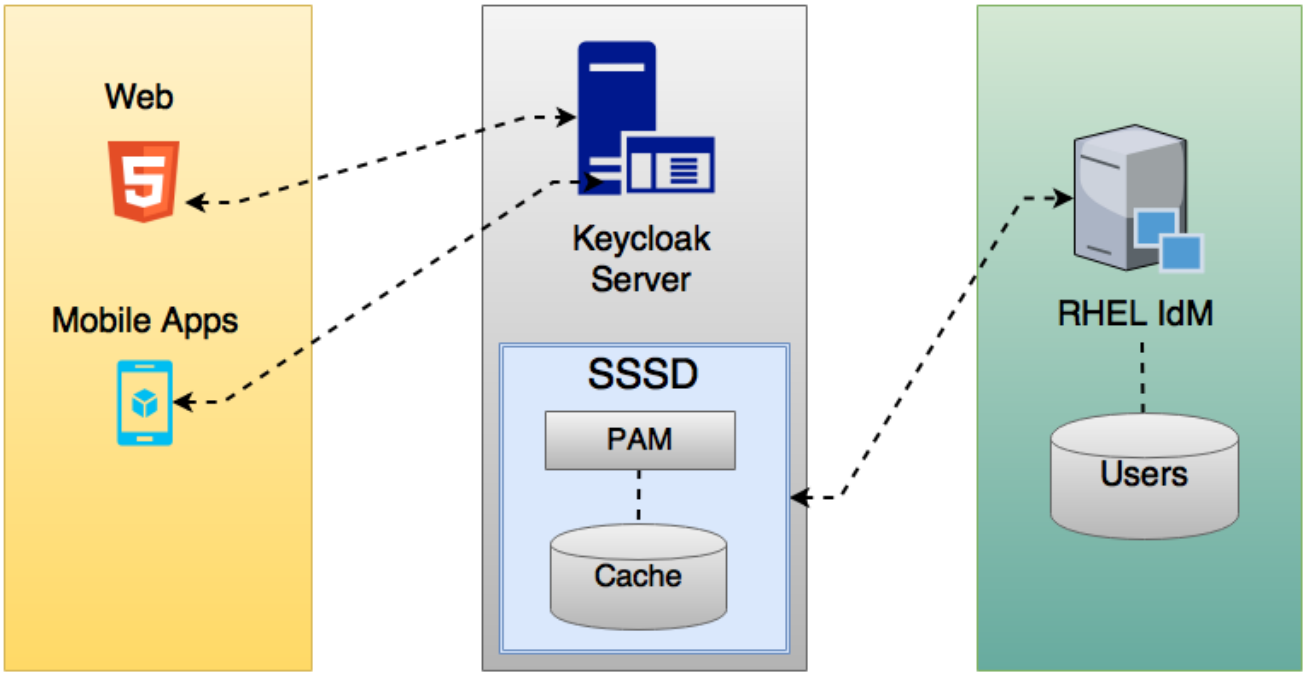
如果服务器重启后没有显示连接池的消息，这可能表示连接池无法用于您的 LDAP 服务器。

- 对于报告 LDAP 问题的情况，您可以考虑将部分 LDAP 树与目标数据附加，这会导致您的环境出现问题。例如，如果某些用户的登录需要很多时间，您可以考虑显示各种"group"条目的 成员属性 计数的 LDAP 条目。在这种情况下，如果这些组条目在红帽构建的 Keycloak 中映射到某些组 LDAP 映射器（或 Role LDAP Mapper）等，则添加这些组条目可能很有用。

## 4.4. SSSD 和 FREEIPA 身份管理集成

红帽构建的 Keycloak 包括 系统安全服务守护进程(SSSD) 插件。SSSD 是 Fedora 和 Red Hat Enterprise Linux (RHEL)的一部分，它提供对多个身份和身份验证提供程序的访问。SSSD 还提供故障转移和离线支持等优点。如需更多信息，请参阅 Red Hat Enterprise Linux Identity Management 文档。

SSSD 与 FreeIPA 身份管理(IdM)服务器集成，提供身份验证和访问控制。通过此集成，红帽构建的 Keycloak 可以针对特权访问管理(PAM)服务进行身份验证，并从 SSSD 检索用户数据。有关在 Linux 环境中使用 Red Hat Identity Management 的更多信息，请参阅 Red Hat Enterprise Linux Identity Management 文档。




红帽构建的 Keycloak 和 SSSD 通过只读 D-Bus 接口进行通信。因此，置备和更新用户的方式是使用 FreeIPA/IdM 管理界面。默认情况下，接口导入用户名、电子邮件、名字和姓氏。

**注意**

红帽构建的 Keycloak 将自动注册组和角色，但不会同步它们。在红帽构建的 Keycloak 管理员中，红帽构建的 Keycloak 管理员所做的任何更改都与 SSSD 同步。

### 4.4.1. FreeIPA/IdM server

FreeIPA 容器镜像位于 Quay.io。要设置 FreeIPA 服务器，请查看 FreeIPA 文档。

**流程**

1. 使用以下命令运行 FreeIPA 服务器：

```
docker run --name freeipa-server-container -it \
-h server.freeipa.local -e PASSWORD=YOUR_PASSWORD \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v /var/lib/ipa-data:/data:Z freeipa/freeipa-server
```

server.freeipa.local 的参数 -h 代表 FreeIPA/IdM 服务器主机名。将 YOUR_PASSWORD 更改为您自己的密码。

2. 容器启动后，将 /etc/hosts 文件改为包括：

```
x.x.x.x     server.freeipa.local
```

如果没有进行此更改，您必须设置 DNS 服务器。

3. 使用以下命令在 IPA 域中注册您的 Linux 服务器，以便 SSSD 联邦供应商在红帽构建的 Keycloak 上启动并运行：

```
ipa-client-install --mkhomedir -p admin -w password
```

4. 在客户端上运行以下命令验证安装是否正常工作：

```
kinit admin
```

5. 输入您的密码。

6. 使用以下命令将用户添加到 IPA 服务器：

```
$ ipa user-add <username> --first=<first name> --last=<surname> --email=<email
address> --phone=<telephoneNumber> --street=<street> --city=<city> --state=<state> -
-postalcode=<postal code> --password
```

7. 使用 kinit 强制设置用户的密码。

```
kinit <username>
```

8. 输入以下内容来恢复普通的 IPA 操作：

```
kdestroy -A
kinit admin
```

### 4.4.2. SSSD 和 D-Bus

联邦供应商使用 D-BUS 从 SSSD 获取数据。它使用 PAM 验证数据。

**流程**

1. 安装 sssd-dbus RPM。

```
$ sudo yum install sssd-dbus
```

2. 运行以下置备脚本：

```
$ bin/federation-sssd-setup.sh
```

该脚本也可以用作为红帽构建的 Keycloak 配置 SSSD 和 PAM 的指南。它对 /etc/sssd/sssd.conf 进行以下更改：

```
[domain/your-hostname.local]
...
ldap_user_extra_attrs = mail:mail, sn:sn, givenname:givenname,
telephoneNumber:telephoneNumber
...
```

```
[sssd]
services = nss, sudo, pam, ssh, ifp
...
[ifp]
allowed_uids = root, yourOSUsername
user_attributes = +mail, +telephoneNumber, +givenname, +sn
```

ifp 服务添加到 SSSD 中，并配置为允许 OS 用户通过这个接口对 IPA 服务器进行干预。

该脚本还会创建一个新的 PAM 服务 /etc/pam.d/keycloak 以通过 SSSD 验证用户：

```
auth    required   pam_sss.so
account required   pam_sss.so
```

3. 运行 dbus-send 以确保设置成功。

```
dbus-send --print-reply --system --dest=org.freedesktop.sssd.infopipe
/org/freedesktop/sssd/infopipe org.freedesktop.sssd.infopipe.GetUserAttr string:
<username> array:string:mail,givenname,sn,telephoneNumber

dbus-send --print-reply --system --dest=org.freedesktop.sssd.infopipe
/org/freedesktop/sssd/infopipe org.freedesktop.sssd.infopipe.GetUserGroups string:
<username>
```

如果设置成功，则每个命令分别显示用户的属性和组。如果有超时或错误，则在红帽构建的 Keycloak 上运行的联邦供应商无法检索任何数据。这个错误通常是因为服务器没有在 FreeIPA IdM 服务器中注册，或者没有访问 SSSD 服务的权限。

如果您没有访问 SSSD 服务的权限，请确保运行红帽构建的 Keycloak 服务器的用户位于以下部分的 /etc/sssd/sssd.conf 文件中：

```
[ifp]
allowed_uids = root, yourOSUsername
```

ipaapi 系统用户是在主机内创建的。ifp 服务需要此用户。检查用户是否已在系统中创建。

```
grep ipaapi /etc/passwd
ipaapi:x:992:988:IPA Framework User:/:/sbin/nologin
```

### 4.4.3. 启用 SSSD 联邦供应商

红帽构建的 Keycloak 使用 DBus-Java 项目与 D-Bus 和JNA 进行通信，以通过操作系统可插拔验证模块(PAM)进行身份验证。

虽然现在红帽构建的 Keycloak 包含运行 SSSD 供应商所需的所有库，但需要 JDK 版本 17。因此，只有在主机配置正确且 JDK 17 用于运行红帽构建的 Keycloak 时，才会显示 SSSD 供应商。

### 4.4.4. 配置联合 SSSD 存储

安装后，配置联合 SSSD 存储。

**流程**

1. 点菜单中的 User Federation。
2. 如果一切设置成功，则 Add Sssd providers 按钮将显示在页面中。点它。
3. 为新提供程序分配名称。
4. 点击 Save。

现在，您可以使用 FreeIPA/IdM 用户和凭证与红帽构建 Keycloak 进行身份验证。

## 4.5. 自定义供应商

红帽构建的 Keycloak 具有用于 User Storage Federation 的 Service Provider Interface (SPI)来开发自定义供应商。您可以在 服务器开发人员指南 中找到有关开发客户供应商的文档。

# 第 5 章 管理用户

在管理控制台中，您可以执行各种操作来管理用户。

## 5.1. 创建用户

您可以在域中创建用户，其中您要拥有这些用户所需的应用程序。避免在 master 域中创建用户，这仅用于创建其他域。

**前提条件**

- 您位于 master 域以外的域中。

**流程**

1. 点菜单中的 Users。
2. 单击 Add User。
3. 输入新用户的详情。

**注意**

Username 是唯一的必填字段。

4. 点 Save。保存详情后，会显示新用户的 Management 页面。

## 5.2. 管理用户属性

在红帽构建的 Keycloak 中，用户与一组属性关联。这些属性用于更好地描述和识别红帽构建的 Keycloak 中的用户，以及将有关它们的额外信息传递给应用程序。

用户配置文件定义了定义良好的模式，用于表示用户属性以及如何在域中管理。通过对用户信息提供一致的视图，管理员可以控制管理属性的不同方面，并更轻松地扩展红帽构建的 Keycloak 以支持其他属性。

虽然用户配置文件主要针对最终用户可以管理的属性（例如：名字和姓氏、电话等），它也用于管理您要与用户关联的任何其他元数据。

用户配置集还可让管理员：

- 为用户属性定义 schema
- 根据上下文信息定义是否需要属性（例如：仅需要用户或 admins，还是根据请求的范围）
- 定义用于查看和编辑用户属性的特定权限，以便遵守无法看到或更改某些属性（包括管理员）的强隐私要求。
- 动态强制用户配置文件合规性，以便始终更新用户信息，并遵守与属性关联的元数据和规则
- 利用内置验证器或编写自定义验证规则，以针对每个属性定义验证规则
- 根据属性定义，用户会在帐户控制台中动态呈现用户与交互的表单，如注册、更新配置集、代理和个人信息，而无需手动更改它们。

- 在管理控制台中自定义用户管理界面，以便根据用户配置集模式动态呈现属性

用户配置文件模式或配置使用 JSON 格式来表示属性及其元数据。在管理控制台中，您可以通过单击左侧菜单中的 Realm Settings 来管理配置，然后单击该页面上的 User Profile 选项卡。

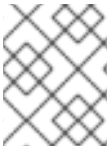
在下一部分中，我们将了解如何创建自己的用户配置文件模式或配置以及如何管理属性。

### 5.2.1. 了解默认配置

默认情况下，Red Hat build of Keycloak 提供了一个基本的用户配置集配置，涵盖了一些最常见的用户属性：

| Name | 描述 |
| --- | --- |
| **username** | 用户名 |
| **email** | 最终用户的首选电子邮件地址。 |
| **firstName** | 给定最终用户的名称或名字 |
| **LastName** | End-User 的 surname 或姓氏 |

在红帽构建的 Keycloak 中，username 和 email 属性都有一个特殊的处理，因为它们通常用于识别、验证和链接用户帐户。对于这些属性，您限于更改其设置，您无法删除它们。



**注意**

username 和 email 属性的行为会相应地更改为 域的登录 设置。例如，将 Email 改为 username 或 Edit username 设置将覆盖您在用户配置文件配置中设置的任何配置。

正如您在以下部分中看到的那样，您可以通过引入自己的属性或更改任何可用属性的设置以更好地满足您的需要，从而自由更改默认配置。

### 5.2.2. 了解用户配置文件上下文

在红帽构建的 Keycloak 中，用户通过不同的上下文管理：

- 注册
- 更新配置文件
- 通过代理或社交供应商进行身份验证时查看配置集
- 帐户控制台
- 管理（例如：管理控制台和管理 REST API）

除了管理上下文外，所有其他上下文都被视为最终用户上下文，因为它们与用户自助服务流相关。

了解这些上下文非常重要，了解您的用户配置文件配置在管理用户时将生效的位置。无论用户管理的

上下文如何，都将使用同一用户配置文件配置来呈现 UI 并验证属性值。

正如您在以下部分中看到的，您可以限制某些属性只能从管理上下文中可用，并为最终用户完全禁用它们。如果您不希望管理员访问某些用户属性，但只限于最终用户，则另一方面也是如此。

### 5.2.3. 了解受管和非受管属性

默认情况下，红帽构建的 Keycloak 只会识别用户配置集配置中定义的属性。服务器会忽略没有明确定义的任何其他属性。

通过严格了解哪些用户属性可以设置为您的用户，以及如何验证它们的值，红帽构建的 Keycloak 可以在您的域中添加另一个保护障碍，并帮助您防止与用户关联的意外属性和值。

这意味着，用户属性可以归类如下：

- 受管.这些是您的用户配置文件控制的属性，您要允许最终用户和管理员从任何用户配置文件上下文管理。对于这些属性，您希望完全控制它们的管理方式和时间。
- 非受管.这些是在用户配置集中没有显式定义的属性，以便默认情况下，Red Hat build of Keycloak 会完全忽略它们。

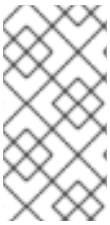
虽然默认情况下禁用非受管属性，但您可以使用不同的策略配置域来定义服务器如何处理它们。为此，请点击左侧菜单中的 Realm Settings，点 General 选项卡，然后从 Unmanaged Attributes 设置中选择以下选项之一：

- 禁用。这是默认策略，以便从所有用户配置集上下文禁用非受管属性。
- 启用。此策略为所有用户配置集上下文启用非受管属性。
- 管理员可以查看。此策略只从管理上下文以只读方式启用非受管属性。
- 管理员可以编辑。此策略只从管理上下文启用非受管属性，以进行读取和写入。

这些策略可让您精细控制服务器如何处理非受管属性。在通过管理上下文管理用户时，您可以选择完全禁用或仅支持非受管属性。

如果启用了非受管属性（即使部分），您可以在 User Details UI 中的 Attributes 选项卡中从管理控制台管理它们。如果策略被设置为 Disabled this tab 不可用。

作为安全建议，尝试尽可能遵循最严格的策略（例如：禁用或管理员 编辑）以防止在通过最终用户管理其配置集时将意外属性（和值）设置为您的用户。避免设置 Enabled 策略，并首选在用户配置文件配置下定义最终用户可以管理的所有属性。



**注意**

Enabled 策略针对于从以前的红帽构建的 Keycloak 版本迁移，并避免在使用自定义主题并使用自己的自定义用户属性扩展服务器时破坏行为。

正如您在以下部分中看到的那样，您还可以通过选择它是否应对用户和/或管理员可见或写入，从而限制属性的使用者。

对于非受管属性，最大长度为 2048 个字符。要指定不同的最小值或最大长度，请将 unmanaged 属性改为 managed 属性并添加 长度 验证器。

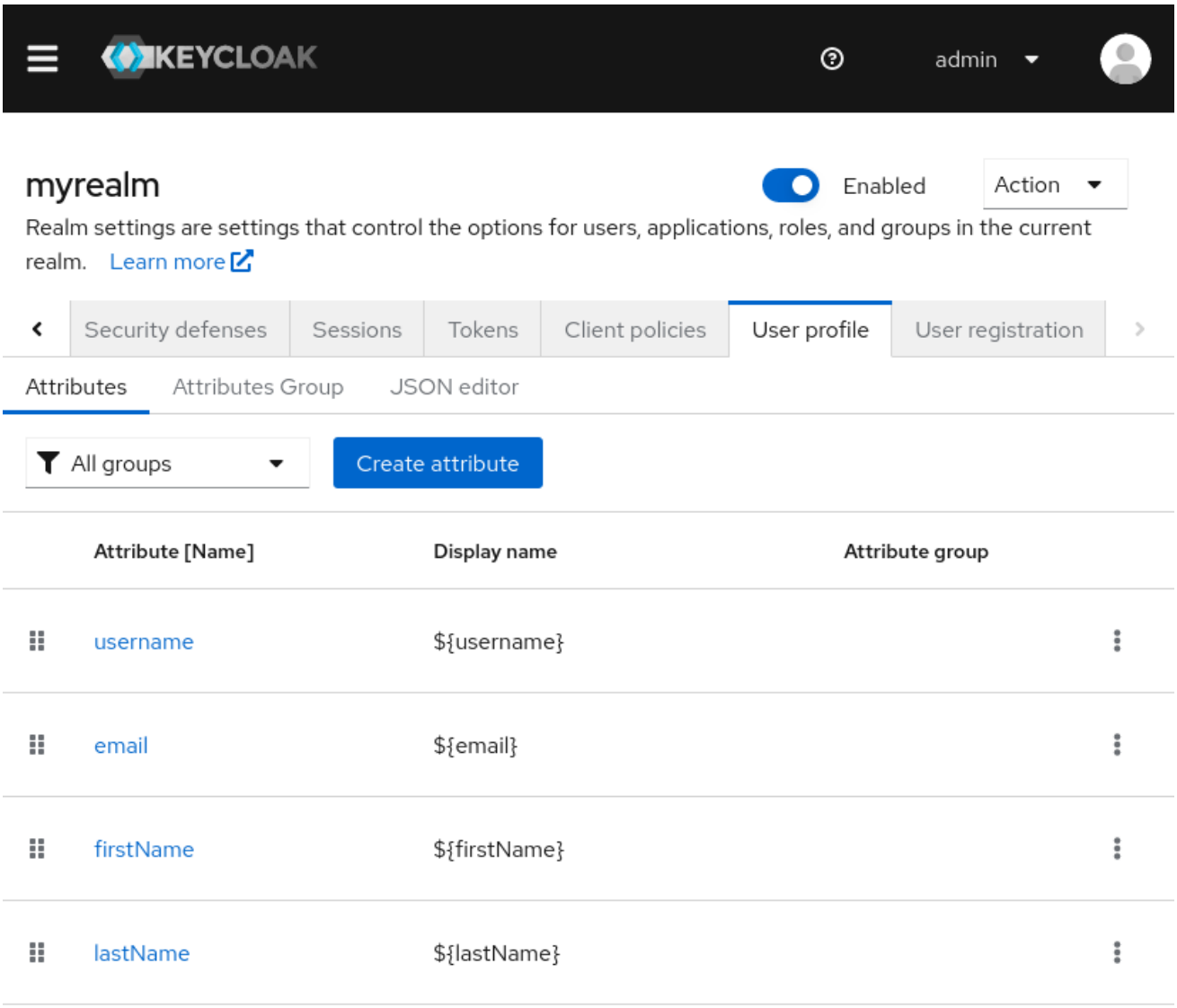
**警告**

红帽构建的 Keycloak 会缓存其内部缓存中与用户相关的对象。属性越长，缓存消耗的内存越多。因此，建议限制 length 属性的大小。考虑在红帽构建的 Keycloak 外部存储大型对象，并通过 ID 或 URL 引用它们。

### 5.2.4. 管理用户配置文件

用户配置集配置基于每个域进行管理。为此，请单击左侧菜单中的 Realm Settings 链接，然后单击 User Profile 选项卡。

用户配置文件选项卡




在 **Attributes** 子选项卡中，您将拥有所有受管属性的列表。

在 **Attribute Groups** 子选项卡中，您可以管理属性组。属性组允许您关联属性，以便在呈现面向用户的表单时显示它们。

在 **JSON Editor** 子选项卡中，您可以查看和编辑 JSON 配置。您可以使用此选项卡获取当前的配置，或者手动管理它。您对此选项卡所做的任何更改都会反映在其他标签页中，反之亦然。

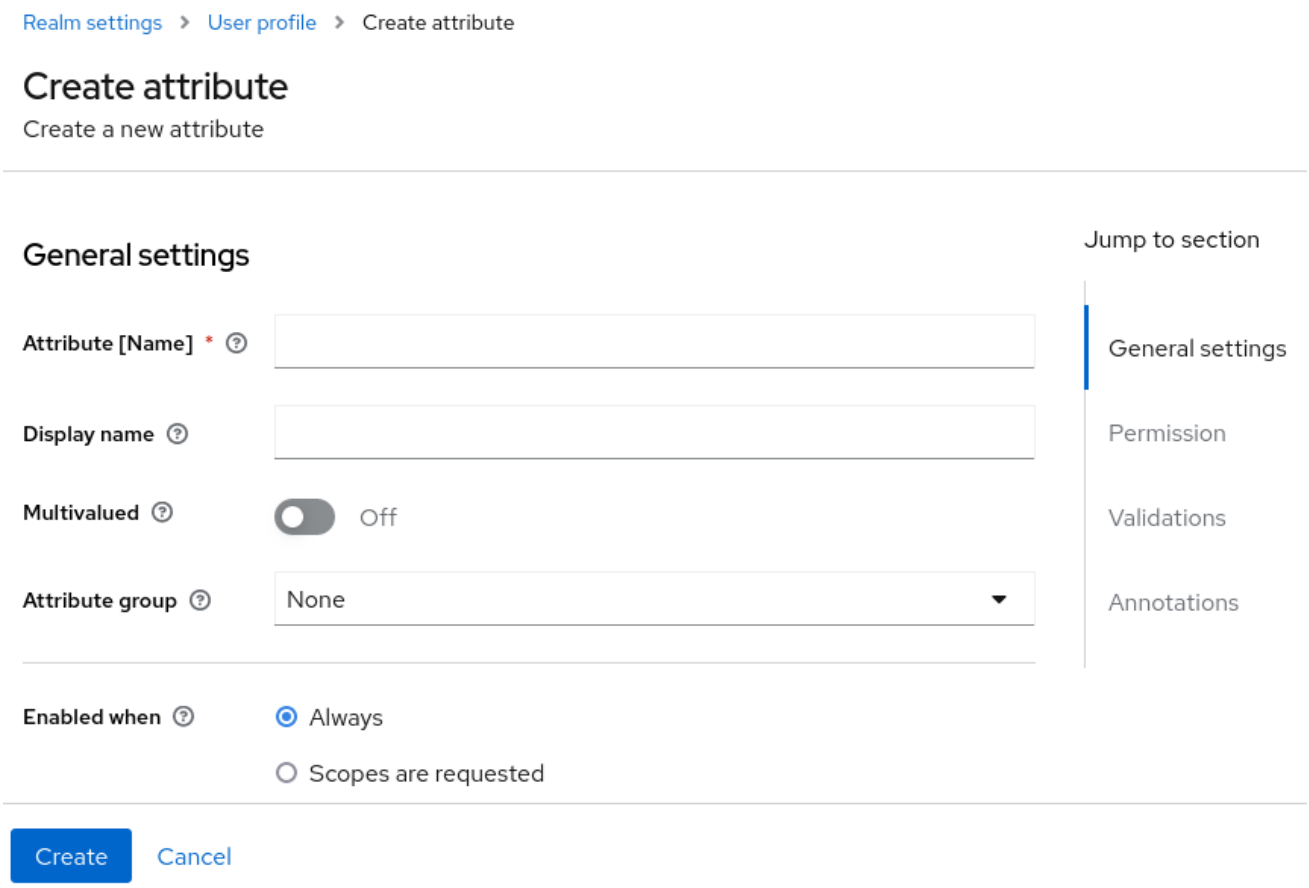
在下一部分中，您将了解如何管理属性。

### 5.2.5. 管理属性

在 属性 子选项卡中，您可以创建、编辑和删除受管属性。

要定义新属性并将其与用户配置文件关联，请单击 属性列表顶部的 Create 属性按钮。

### 属性配置




在配置属性时，您可以定义以下设置：

#### Name

属性的名称，用于唯一标识属性。

#### 显示名称

属性的用户友好名称，主要用于呈现面向用户的表单。它还支持 使用国际消息

#### Multivalued

如果启用，则属性支持多个值和 UI，以允许设置多个值。启用此设置时，请确保添加验证器来为

值数设置硬限制。

属性组

属性所属的属性组（若有）。

启用的时间

启用或禁用属性。如果设置为 Always，则属性可从任何用户配置文件上下文中可用。如果请求范围，则属性仅在代表用户的客户端请求一组或多个范围时才可用。您可以根据请求的客户端范围，使用这个选项来动态强制实施某些属性。对于帐户和管理控制台，不会评估范围，并且属性始终被启用。这是因为，根据范围过滤属性仅在运行身份验证流时正常工作。

必填

将条件设置为根据需要标记属性。如果禁用，则属性是可选的。如果启用，您可以设置所需的设置 来根据 用户配置集上下文将属性标记为必需，以便最终用户（通过最终用户上下文）或管理员（通过管理上下文）需要属性。您还可以在设置时设置 Required 来仅在请求一组或多个客户端范围时才将属性标记为 required。如果设置为 Always，则需要来自任何用户配置文件上下文的属性。如果请求 范围，则只有代表用户的客户端请求一组或多个范围时才需要属性。对于帐户和管理控制台，不会评估范围，且不需要属性。这是因为，根据范围过滤属性仅在运行身份验证流时正常工作。

权限

在本节中，您可以在从最终用户或管理上下文中管理属性时定义读写权限。Who 可以编辑 设置，将属性分别由 User 和/或 Admin 分别从最终用户和管理上下文中写入。Who 可以从 最终用户和管理上下文中分别将属性标记为只读。

验证

在本节中，您可以定义管理属性值时将执行的验证。Red Hat build of Keycloak 提供了一组内置验证器，您可以选择它们可能会自行添加。如需更多详细信息，请参阅 Validating Attributes 部分。

注解

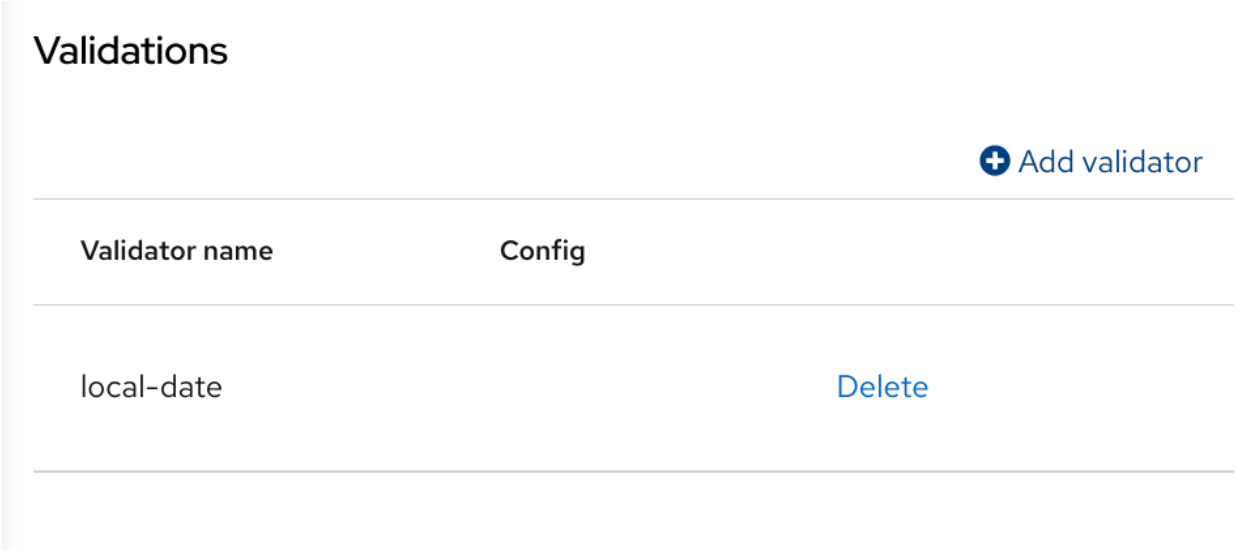
在本小节中，您可以将注解与属性关联。对于渲染目的，注解主要有助于将其他元数据传递给前端。如需了解更多详细信息，请参阅 定义 UI Annotations 部分。

当您创建属性时，属性只能从管理上下文中可用，以避免意外向最终用户公开属性。实际上，在通过最终用户上下文管理其配置文件时，最终用户无法访问该属性。您可以随时根据您的需要更改 Permissions 设置。

#### 5.2.6. 验证属性

您可以启用对受管属性的验证，以确保属性值符合特定的规则。为此，您可以在管理属性时从 Validations 设置中添加或删除验证器。

属性验证




在写入属性时，验证都会发生，它们可能会抛出在值验证失败时在 UI 中显示的错误。

为安全起见，用户可编辑的每个属性都应该有一个验证来限制用户输入的值的大小。如果没有指定 长度 验证器，Red Hat build of Keycloak 默认为最大 2048 个字符。

#### 5.2.6.1. 内置验证器

Red Hat build of Keycloak 提供了您可以从中选择的一些内置验证器，您也可以通过扩展 Validator SPI 来提供自己的验证器。

以下列表提供了所有内置验证器的列表：

| Name | 描述 | Configuration |
| --- | --- | --- |
| length | 根据最小和最大长度，检查字符串值的长度。 | Min: 定义允许的最小长度的整数。<br><br>max ：一个整数，用于定义最大允许长度。<br><br>trim-disabled ：一个布尔值，用于定义在验证前是否修剪该值。 |

| Name | 描述 | Configuration |
| --- | --- | --- |
| 整数 | 检查该值是否为整数，并在较低和/或大写范围内。如果没有定义范围，验证器只检查该值是否为有效数字。 | **Min**: 定义较低范围的整数。<br><br>**max** ：定义上限的整数。 |
| double | 检查该值是否为双倍，并在低和/或大写范围内。如果没有定义范围，验证器只检查该值是否为有效数字。 | **Min**: 定义较低范围的整数。<br><br>**max** ：定义上限的整数。 |
| uri | 检查该值是否为有效的 URI。 | None |
| pattern | 检查值是否与特定 RegEx 模式匹配。 | **Pattern** ：验证值时要使用的 RegEx 模式。<br><br>**error-message** ： i18n 捆绑包中错误消息的密钥。如果没有设置通用消息，则使用通用消息。 |
| email | 检查该值是否具有有效的电子邮件格式。 | **max-local-length** ：一个整数，用于定义电子邮件本地部分的最大长度。每个规格默认为 64。 |
| local-date | 检查值是否具有基于 realm 和/或用户区域设置的有效格式。 | None |
| person-name-prohibited-characters | 检查该值是否为有效的人员名称，作为脚本注入等攻击的额外障碍。验证基于默认的 RegEx 模式，用于阻止人员名称中不常见字符。 | **error-message** ： i18n 捆绑包中错误消息的密钥。如果没有设置通用消息，则使用通用消息。 |
| username-prohibited-characters | 检查该值是否为有效的用户名，作为脚本注入等攻击的额外障碍。验证基于默认的 RegEx 模式，用于阻止用户名中不常见的字符。 | **error-message** ： i18n 捆绑包中错误消息的密钥。如果没有设置通用消息，则使用通用消息。 |
| 选项 | 检查该值是否来自定义的允许值集合。可用于验证通过 select 和 multiselect 字段输入的值。 | **选项** ：包含允许值的字符串数组。 |
| up-username-not-idn-homograph | 该字段只能包含 latin 字符和常见 unicode 字符。这对于字段很有用，可以是 IDN 摄影攻击（通常是用户名）的主题。 | **error-message** ： i18n 捆绑包中错误消息的密钥。如果没有设置通用消息，则使用通用消息。 |

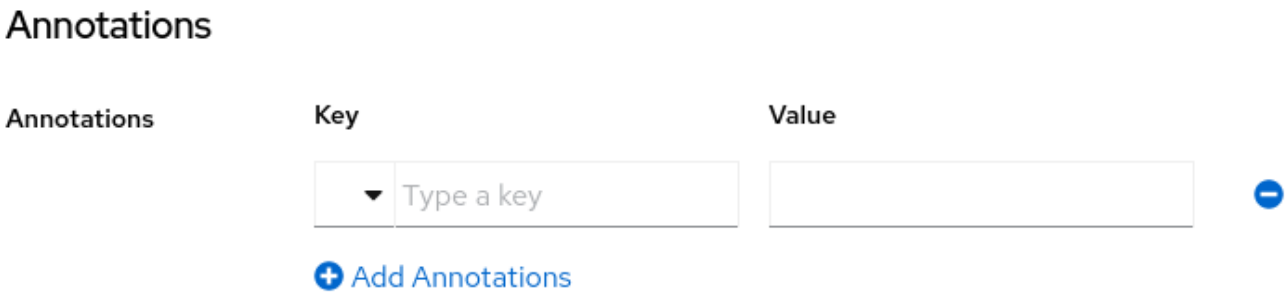
| Name | 描述 | Configuration |
| --- | --- | --- |
| 多值 | 验证多值属性的大小。 | Min: 整数，用于定义允许的属性值数。<br><br>max ：一个整数，用于定义允许的最大属性值数。 |

### 5.2.7. 定义 UI 标注

要将额外信息传递给前端，可以使用注解对属性进行解码，以决定如何呈现属性。当扩展红帽构建的 Keycloak 主题时，此功能主要有用，以根据与属性关联的注解动态呈现页面。

使用注解（例如，用于更改 属性的 HTML 类型 ）并更改 属性的 DOM 表示，如以下部分中所示。

**属性注解**




注解是与 UI 共享的键/值对，它们可以更改与属性对应的 HTML 元素是如何呈现的。只要您的 realm 使用的主题支持注解，就可以将您想要的注解设置为属性。

**注意**

您的唯一限制是避免在其密钥中使用 kc 前缀，因为这些注解使用此前缀保留给红帽构建的 Keycloak。

#### 5.2.7.1. 内置注解

红帽构建的 Keycloak 内置主题支持以下注解：

| Name | 描述 |
| --- | --- |
| inputType | 表单输入字段的类型。下表中描述了可用的类型。 |
| inputHelperTextBefore | 帮助程序文本在（左上）输入字段之前显示。此处可以使用直接文本或国际化模式（如 **${i18n.key}**）。文本在呈现到页面中时不会进行 html 转义，因此您可以使用此处的 html 标签格式化文本，但您还必须正确转义 html 控制字符。 |
| inputHelperTextAfter | 帮助程序文本在（在）输入字段后显示。此处可以使用直接文本或国际化模式（如 **${i18n.key}**）。文本在呈现到页面中时不会进行 html 转义，因此您可以使用此处的 html 标签格式化文本，但您还必须正确转义 html 控制字符。 |
| inputOptionsFromValidation | 选择和多选类型的注解。从中获取输入选项的可选自定义属性验证名称。请参见以下 的详细描述。 |
| inputOptionLabelsI18nPrefix | 选择和多选类型的注解。在 UI 中呈现选项的国际化密钥前缀。请参见以下 的详细描述。 |
| inputOptionLabels | 选择和多选类型的注解。可选映射为选项定义 UI 标签（直接或使用国际化）。请参见以下 的详细描述。 |
| inputTypePlaceholder | 应用到字段的 HTML 输入 **占位符** 属性 - 指定简短提示，用于描述输入字段的预期值（如示例值或预期格式的简短描述）。在用户输入值前，在输入字段中会显示简短提示。 |
| inputTypeSize | 应用到字段的 HTML 输入 **大小** 属性 - 指定单行输入字段的宽度（以字符为单位）。对于基于 HTML **选择类型的字段**，它指定了显示选项的行数。可能无法工作，具体取决于使用的 css！ |
| inputTypeCols | 应用于字段的 HTML 输入 **cols** 属性 - 指定宽度（对于 **textarea** 类型）。可能无法工作，具体取决于使用的 css！ |
| inputTypeRows | 应用到字段的 HTML 输入 **行** - 为 **textarea** 类型指定高亮、字符数。对于选择字段，它通过显示的选项指定行数。可能无法工作，具体取决于使用的 css！ |
| inputTypePattern | 应用到提供客户端侧验证的字段的 HTML 输入 **模式** 属性 - 指定检查输入字段值的正则表达式。对于单行输入很有用。 |
| inputTypeMaxLength | 应用到提供客户端验证的字段的 HTML 输入 **maxlength** 属性 - 可以在输入字段中输入的文本的最大长度。对于文本字段很有用。 |

| Name | 描述 |
| --- | --- |
| inputTypeMinLength | 应用到提供客户端验证的字段的 HTML 输入 **minlength** 属性 - 可以在输入字段中输入的最小文本长度。对于文本字段很有用。 |
| inputTypeMax | 应用到提供客户端侧验证的字段的 HTML 输入 **max** 属性 - 可以在输入字段中输入的最大值。对数字字段很有用。 |
| inputTypeMin | 应用到提供客户端侧验证的字段的 HTML 输入 **min** 属性 - 可以在输入字段中输入的最小值。对数字字段很有用。 |
| inputTypeStep | 应用到字段的 HTML 输入 **step** 属性 - 指定输入字段中的法律数字之间的间隔。对数字字段很有用。 |
| 数字格式 | 如果设置，**data-kcNumberFormat** 属性添加到字段中，以根据给定格式格式化值。此注解针对数字，其中格式基于确定的位置的预期数字数。例如，一个格式 **({2}){5}-{4}** 将字段值格式化为 **(00) 00000-0000**。 |
| number UnFormat | 如果设置，**data-kcNumberUnFormat** 属性将添加到字段中，以便在提交表单前根据给定格式格式化值。如果您不希望存储特定属性的任何格式，但仅在客户端一侧格式化该值，则此注解很有用。例如，如果当前值为 **(00) 00000-0000**，如果将值 **{11}** 设置为此注解，则该值将更改为 **00000000000**，或者指定一组或一组数字，则该值将更改为 00000000000。确保添加验证器以在存储值前执行服务器端验证。 |

**注意**

**字段类型使用 HTML 表单字段标签和应用于它们的属性 - 它们的行为取决于 HTML 规格和浏览器支持它们。**

**可视化渲染也取决于所使用的主题中应用的 cs 样式。**

#### 5.2.7.2. 更改 属性的 HTML 类型

**您可以通过设置 inputType 注解来更改 HTML5 输入元素 的类型。可用的类型有：**

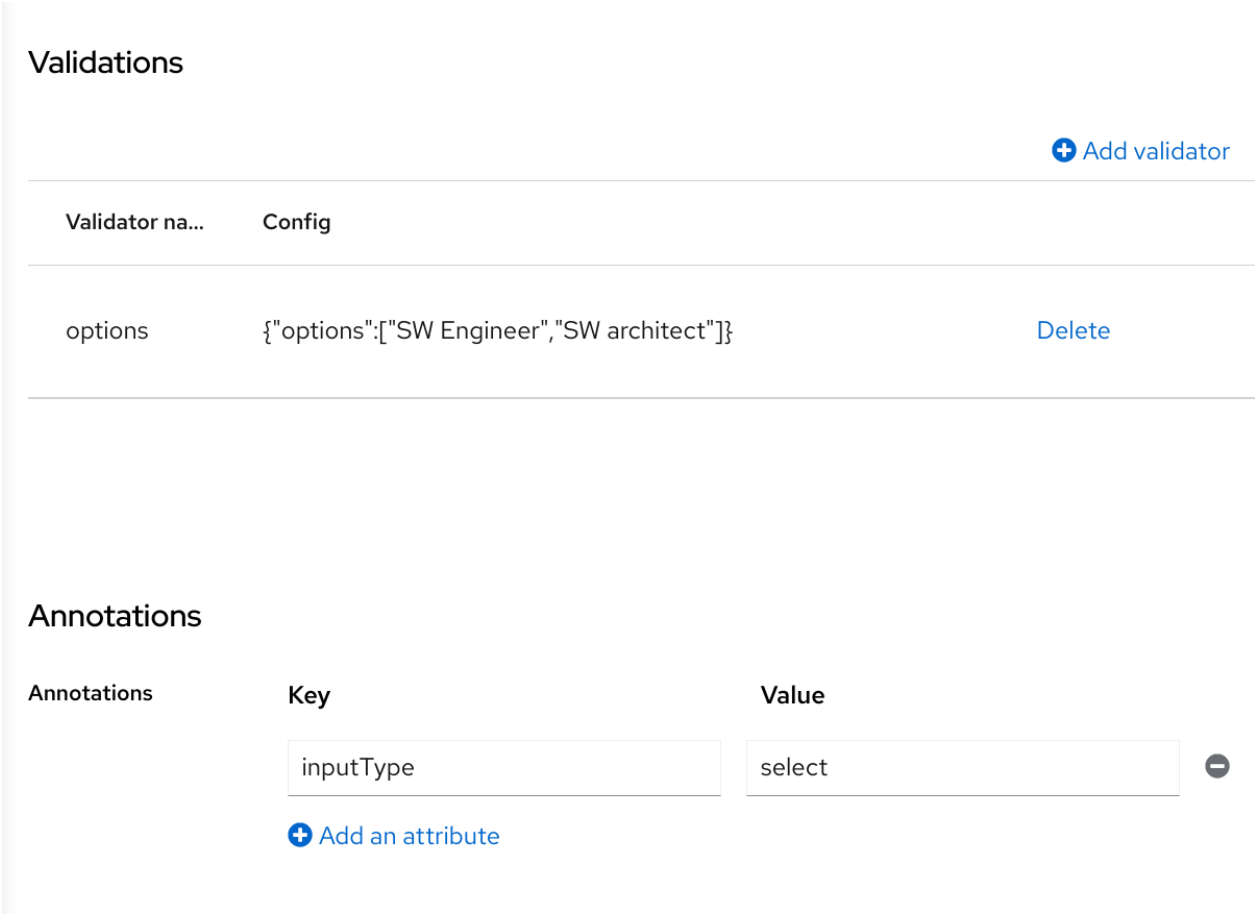
| Name | 描述 | 使用的 HTML 标签 |
| --- | --- | --- |
| text | 单行文本输入. | 输入 |
| textarea | 多行文本输入. | textarea |
| select | 常见单一选择输入。请参阅下面如何配置选项的描述。 | select |
| select-radiobuttons | 通过一组单选按钮选择输入。请参阅下面如何配置选项的描述。 | 输入组 |
| multiselect | 常见多选择输入。请参阅下面如何配置选项的描述。 | select |
| multiselect-checkboxes | 通过一组复选框进行多选输入。请参阅下面如何配置选项的描述。 | 输入组 |
| html5-email | 基于 HTML 5 规格的电子邮件地址的单行文本输入。 | 输入 |
| html5-tel | 基于 HTML 5 规格的电话号码的单行文本输入。 | 输入 |
| html5-url | 基于 HTML 5 spec 的用于 URL 的单行文本输入。 | 输入 |
| html5-number | 基于 HTML 5 规格的单行输入（整数或浮点，取决于 **step**） | 输入 |
| html5-range | 根据 HTML 5 规格输入数字的滑块。 | 输入 |
| html5-datetime-local | 基于 HTML 5 规格的日期时间输入。 | 输入 |
| html5-date | 基于 HTML 5 规格的日期输入。 | 输入 |
| html5-month | 基于 HTML 5 规格的月份输入。 | 输入 |
| html5-week | 基于 HTML 5 规格的周输入。 | 输入 |
| html5-time | 基于 HTML 5 规格的时间输入。 | 输入 |

#### 5.2.7.3. 定义 select 和 multiselect 字段的选项

**select 和 multiselect 字段的选项从应用到属性的验证中获取，以确保 UI 中显示的验证和字段选项始终一致。默认情况下，选项取自内置 选项 验证。**

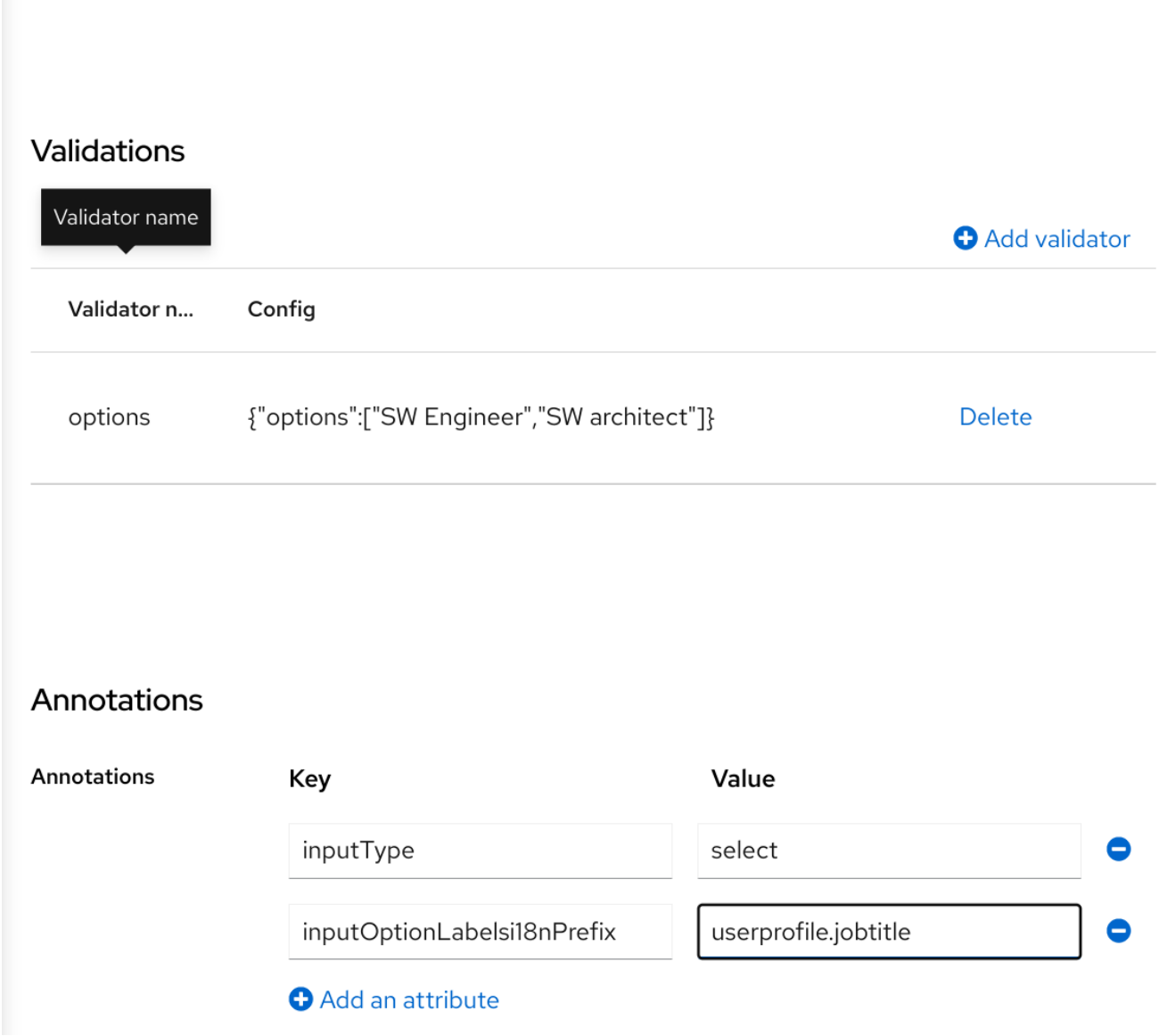
您可以使用各种方法为选择和多选选项提供 nice 人类可读的标签。最简单的情况是属性值与 UI 标签相同。在这种情况下不需要额外的配置。

选项值与 UI 标签相同




当属性值不是 UI 的 ID 类型时，您可以使用 inputOptionLabelsI18nPrefix 注解提供的简单国际化支持。它定义国际化键的前缀，选项值是附加到这个前缀的点。

使用 i18n 键前缀进行 UI 标签的简单国际化




选项值的本地化 UI 标签文本必须由 **userprofile.jobtitle.sweng** 和 **userprofile.jobtitle.swarch** 键提供，然后使用通用的本地化机制。

您还可以使用 **inputOptionLabels** 注解来为单个选项提供标签。它包含 option 的标签映射 - 映射中的键是 option 值（在验证中定义），映射中的值是 UI 标签文本本身或其国际化模式（如 ${i18n.key}）用于该选项。

**注意**

您必须使用 User Profile JSON Editor 输入映射作为 **inputOptionLabels** 注解值。

没有国际化的独立选项直接输入标签示例：

```
"attributes": [
<...
{
  "name": "jobTitle",
  "validations": {
    "options": {
      "options":[
        "sweng",
        "swarch"
      ]
    }
  },
  "annotations": {
    "inputType": "select",
    "inputOptionLabels": {
      "sweng": "Software Engineer",
      "swarch": "Software Architect"
    }
  }
}
...
]
```

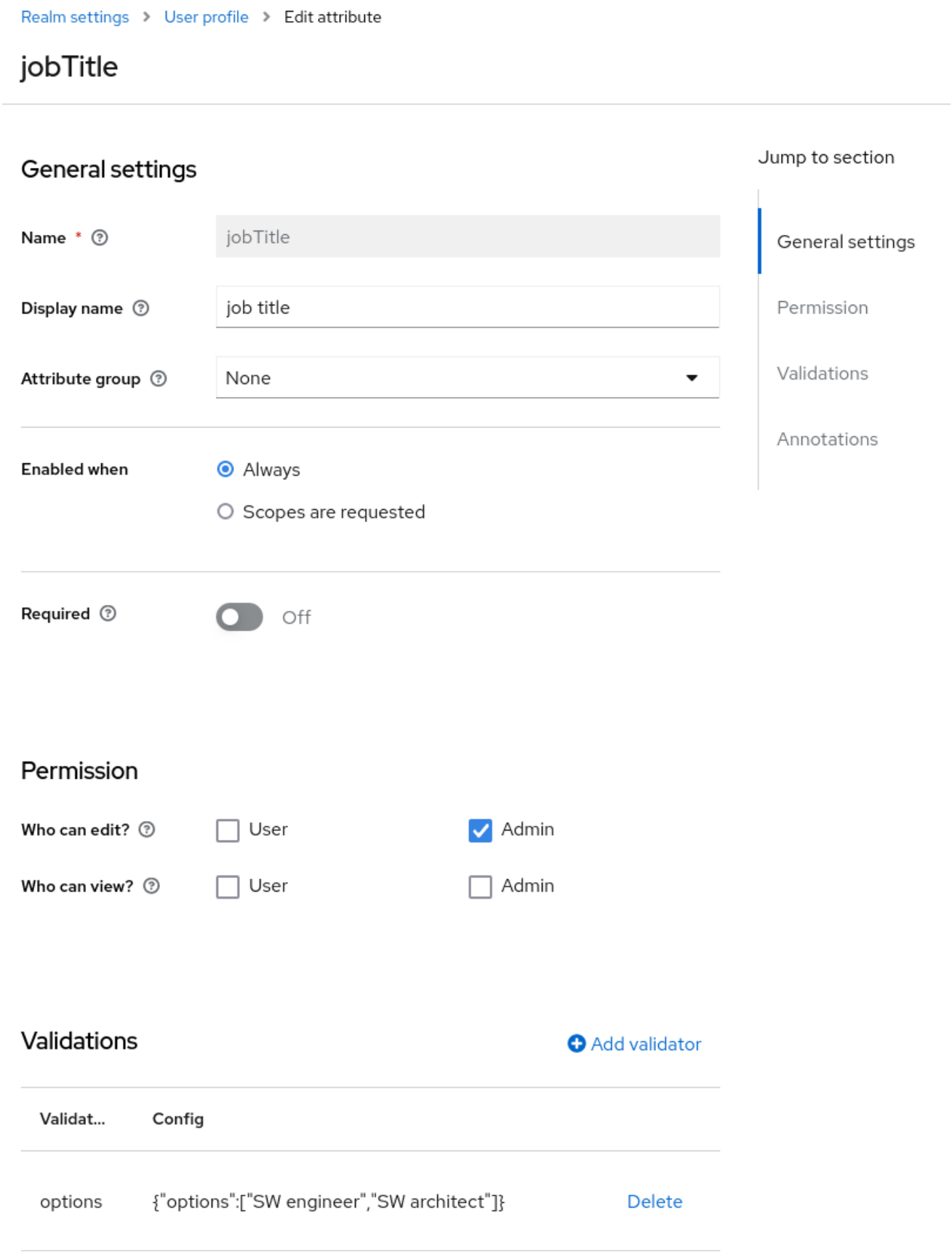
单个选项的国际化标签示例：

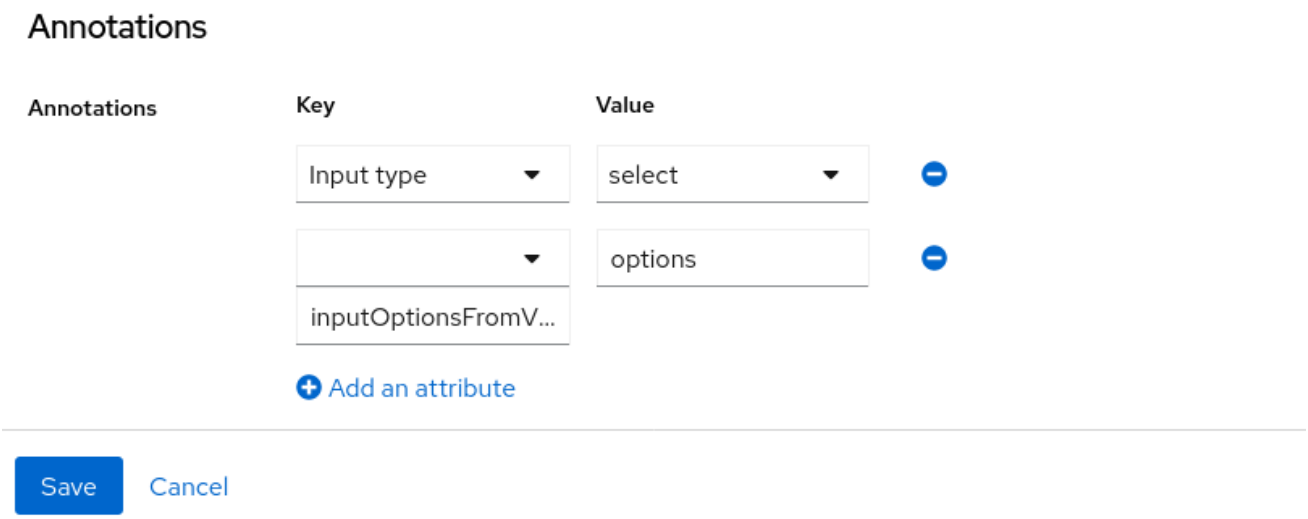
```
"attributes": [
...
{
  "name": "jobTitle",
  "validations": {
    "options": {
      "options":[
        "sweng",
        "swarch"
      ]
    }
  },
  "annotations": {
    "inputType": "select-radiobuttons",
    "inputOptionLabels": {
      "sweng": "${jobtitle.swengineer}",
      "swarch": "${jobtitle.swarchitect}"
    }
  }
}
...
]
```

本地化文本必须由 jobtitle.swengineer 和 jobtitle.swarchitect 键提供，使用常见的本地化机制。

自定义验证器可用于提供选项，因为 inputOptionsFromValidation 属性注解。此验证必须具有提供选项数组 的选项 配置。国际化的工作方式与内置选项验证 提供的选项 相同。

自定义验证器提供的选项






#### 5.2.7.4. 更改属性的 DOM 表示

您可以使用 kc 前缀设置注解来启用额外的客户端行为。这些注解将转换为属性的对应元素中的 HTML 属性，带有 data- 前缀，具有相同名称的脚本将被加载到动态页面，以便您可以根据自定义数据属性选择 DOM 中的元素，并通过修改其 DOM 表示来相应地分离它们。

例如，如果您向属性添加 kcMyCustomValidation 注解，则 HTML 属性 data-kcMyCustomValidation 会添加到属性的对应 HTML 元素中，并且从 < THEME TYPE>/resources/js/kcMyCustomValidation.js 的自定义主题加载 JavaScript 模块。有关如何将自定义 JavaScript 模块部署到主题的更多信息，请参阅 Server Developer Guide。

JavaScript 模块可以运行任何代码来自定义 DOM 以及每个属性呈现的元素。为此，您可以使用 userProfile.js 模块为自定义注解注册注解描述符，如下所示：

```
import { registerElementAnnotatedBy } from "./userProfile.js";

registerElementAnnotatedBy({
  name: 'kcMyCustomValidation',
  onAdd(element) {
    var listener = function (event) {
       // do something on keyup
    };

    element.addEventListener("keyup", listener);

    // returns a cleanup function to remove the event listener
    return () => element.removeEventListener("keyup", listener);
  }
});
```

registerElementAnnotatedBy 是一个注册注解描述符的方法。描述符是 具有名称、引用注解名称和一个 onAdd 函数的对象。每当页面被渲染或带有注解的属性添加到 DOM 时，都会调用 onAdd 函数，以便您可以自定义元素的行为。

onAdd 功能也可以返回执行清理的功能。例如，如果您要将事件监听程序添加到元素中，您可能希望在从 DOM 中删除该元素时删除它们。

另外，如果 userProfile.js 不足以满足您的需要，您还可以使用您想要的任何 JavaScript 代码：

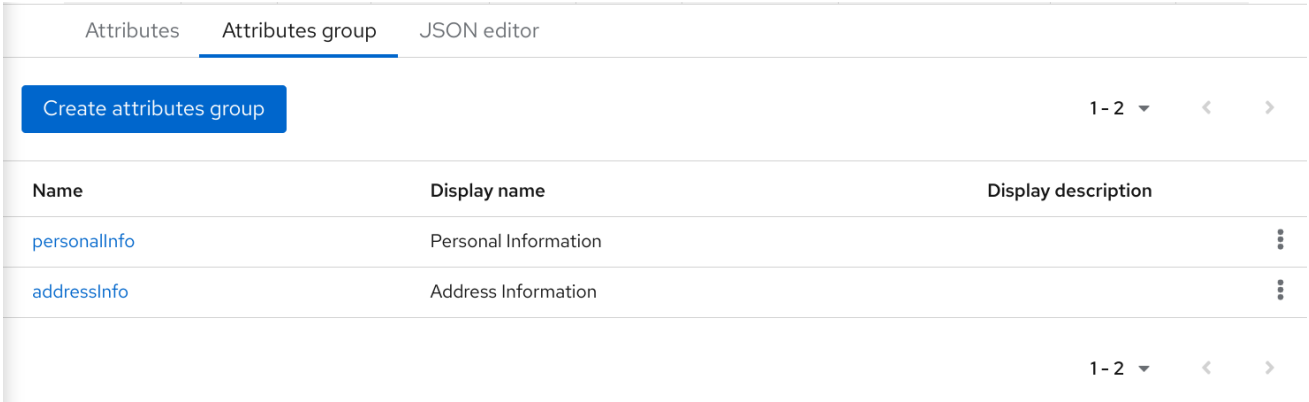
```
document.querySelectorAll(`[data-kcMyCustomValidation]`).forEach((element) => {
  var listener = function (evt) {
    // do something on keyup
  };

  element.addEventListener("keyup", listener);
});
```

### 5.2.8. 管理属性组

在 Attribute Groups 子选项卡中，您可以创建、编辑和删除属性组。属性组允许您为关联的属性定义容器，以便在用户表单上呈现容器。

属性组列表

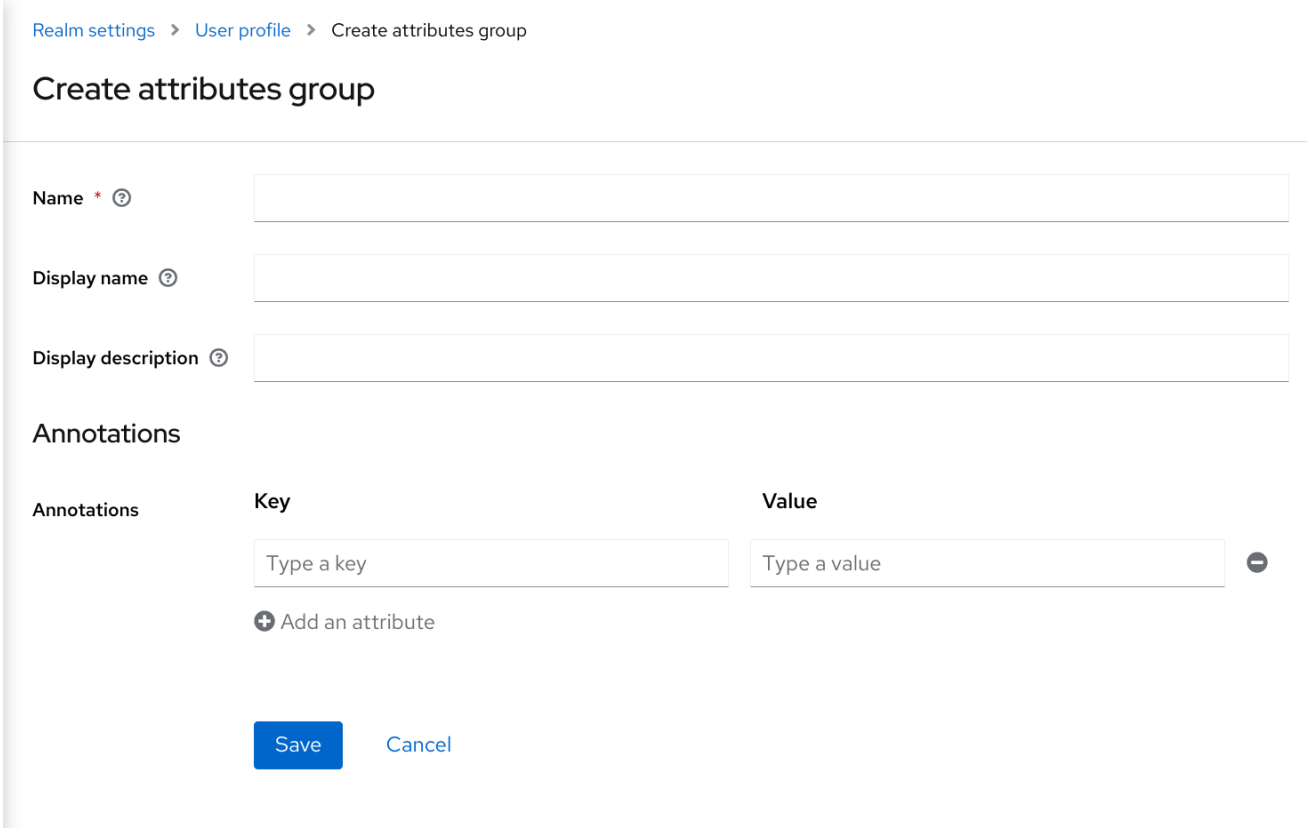



注意

您不能删除绑定到属性的属性组。为此，您应该首先更新属性以删除绑定。

若要创建新组，请单击属性组列表顶部的 Create attributes group 按钮。

属性组配置




在配置组时，您可以定义以下设置：

Name

属性的名称，用于唯一标识属性。

显示名称

属性的用户友好名称，主要用于呈现面向用户的表单。它还支持 使用国际消息

显示描述

在呈现面向用户的表单时，用户友好的文本将显示为工具提示。它还支持 使用国际消息

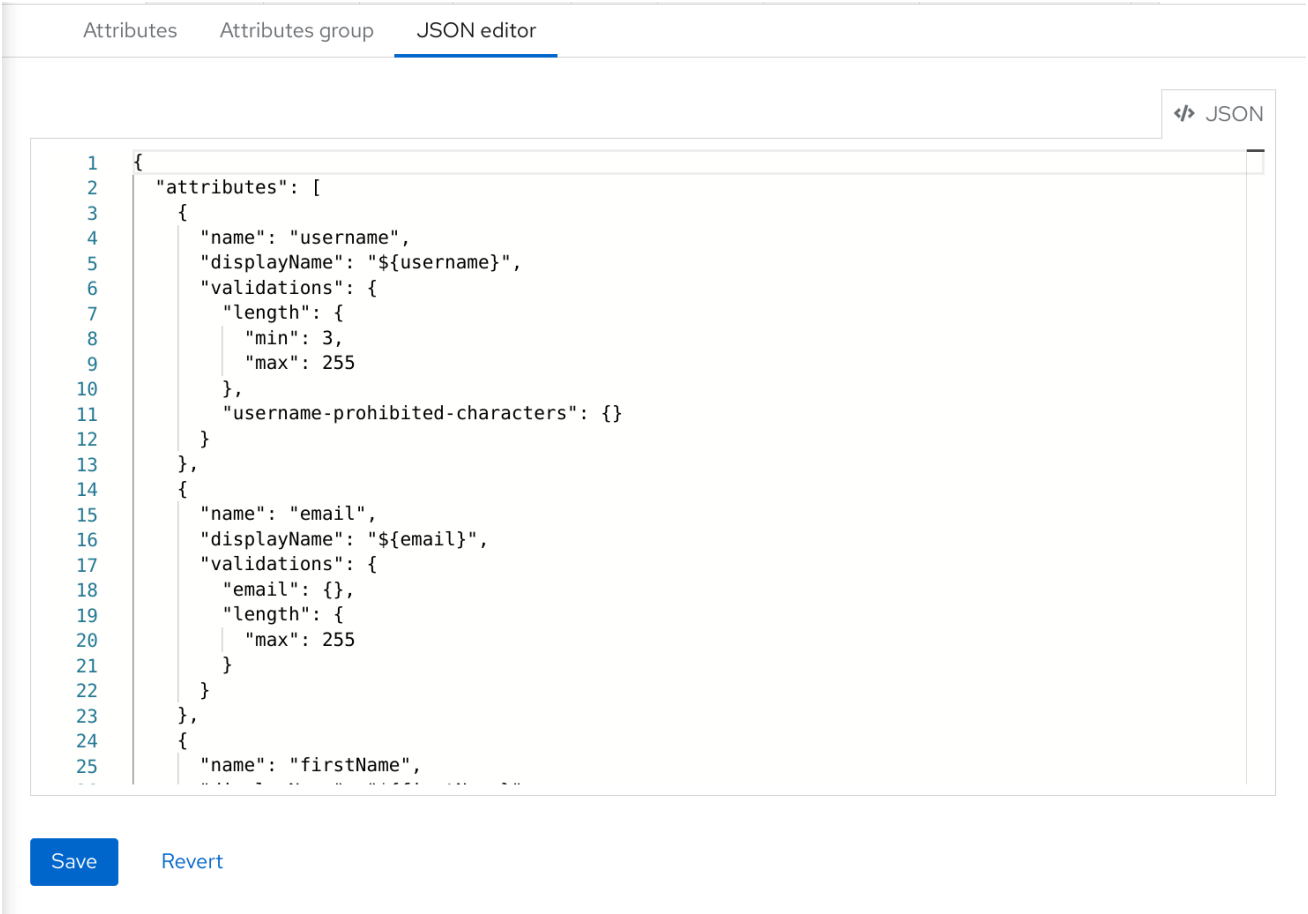
注解

在本小节中，您可以将注解与属性关联。对于渲染目的，注解主要有助于将其他元数据传递给前端。

### 5.2.9. 使用 JSON 配置

用户配置文件配置使用明确定义的 JSON 模式存储。您可以通过单击 JSON Editor 子选项卡直接从编辑用户配置文件配置中进行选择。

**JSON 配置**




JSON 模式定义如下：

```
{
  "unmanagedAttributePolicy": "DISABLED",
  "attributes": [
    {
      "name": "myattribute",
      "multivalued": false,
      "displayName": "My Attribute",
      "group": "personalInfo",
      "required": {
        "roles": [ "user", "admin" ],
```

```
      "scopes": [ "foo", "bar" ]
     },
     "permissions": {
      "view": [ "admin", "user" ],
      "edit": [ "admin", "user" ]
     },
     "validations": {
      "email": {
       "max-local-length": 64
      },
      "length": {
       "max": 255
      }
     },
     "annotations": {
      "myannotation": "myannotation-value"
     }
    }
   ],
   "groups": [
    {
     "name": "personalInfo",
     "displayHeader": "Personal Information",
     "annotations": {
      "foo": ["foo-value"],
      "bar": ["bar-value"]
     }
    }
   ]
}
```

该架构根据需要支持任意数量的属性和组。

**unmanagedAttributePolicy 属性通过设置以下值之一来定义非受管属性策略：如需了解更多详细信息，请参阅 了解受管和非受管属性。**

- **DISABLED**
- **ENABLED**
- **ADMIN_VIEW**
- **ADMIN_EDIT**

#### 5.2.9.1. 属性架构

对于每个属性，您应该定义一个 名称，并可以选择定义 所需的、权限 和 annotations 设置。

required 属性定义是否需要属性。红帽构建的 Keycloak 允许您根据不同的条件设置属性。

当将 required 属性定义为空对象时，始终需要属性。

```
{
  "attributes": [
    {
      "name": "myattribute",
      "required": {}
  ]
}
```

另一方面，您可以选择只为用户或管理员或两个都进行必要的属性。以及仅在 Red Hat build of Keycloak 进行身份验证时请求特定范围时，才需要标记属性。

要根据用户和/或管理员的要求标记属性，请设置 roles 属性，如下所示：

```
{
  "attributes": [
    {
      "name": "myattribute",
      "required": {
        "roles": ["user"]
      }
  ]
}
```

roles 属性需要一个数组，其值可以是 user 或 admin，具体取决于用户还是管理员是否要求属性。

同样，您可以选择在验证用户时请求一组或多个范围时所需的属性。为此，您可以使用 scopes 属性，如下所示：

```
{
  "attributes": [
    {
      "name": "myattribute",
      "required": {
```

```
      "scopes": ["foo"]
    }
  ]
}
```

scopes 属性是一个数组，其值可以是代表客户端范围的任何字符串。

attribute-level permissions 属性可用于定义属性的读取和写入权限。权限根据这些操作的设置，用户是否可以对属性执行，还是管理员或两者。

```
{
  "attributes": [
    {
      "name": "myattribute",
      "permissions": {
        "view": ["admin"],
        "edit": ["user"]
      }
  ]
}
```

view 和 edit 属性都预期值可以是 user 或 admin 的数组，具体取决于属性是否可以被用户或管理员编辑。

当授予edit权限时，而会获得 view 权限。

attribute-level 注解 属性可用于将其他元数据与属性关联。注解主要用于将属性的额外信息传递给基于用户配置集配置的前端渲染用户属性。每个注释都是键/值对。

```
{
  "attributes": [
    {
      "name": "myattribute",
      "annotations": {
        "foo": ["foo-value"],
        "bar": ["bar-value"]
      }
  ]
}
```

#### 5.2.9.2. 属性组架构

对于每个属性组，您应该定义一个 名称，以及可选的 注解 设置。

attribute-level 注解 属性可用于将其他元数据与属性关联。注解主要用于将属性的额外信息传递给基于用户配置集配置的前端渲染用户属性。每个注释都是键/值对。

### 5.2.10. 自定义 UI 的渲染方式

来自所有用户配置集上下文（包括管理控制台）的 UI 会动态呈现到您的用户配置集配置中。

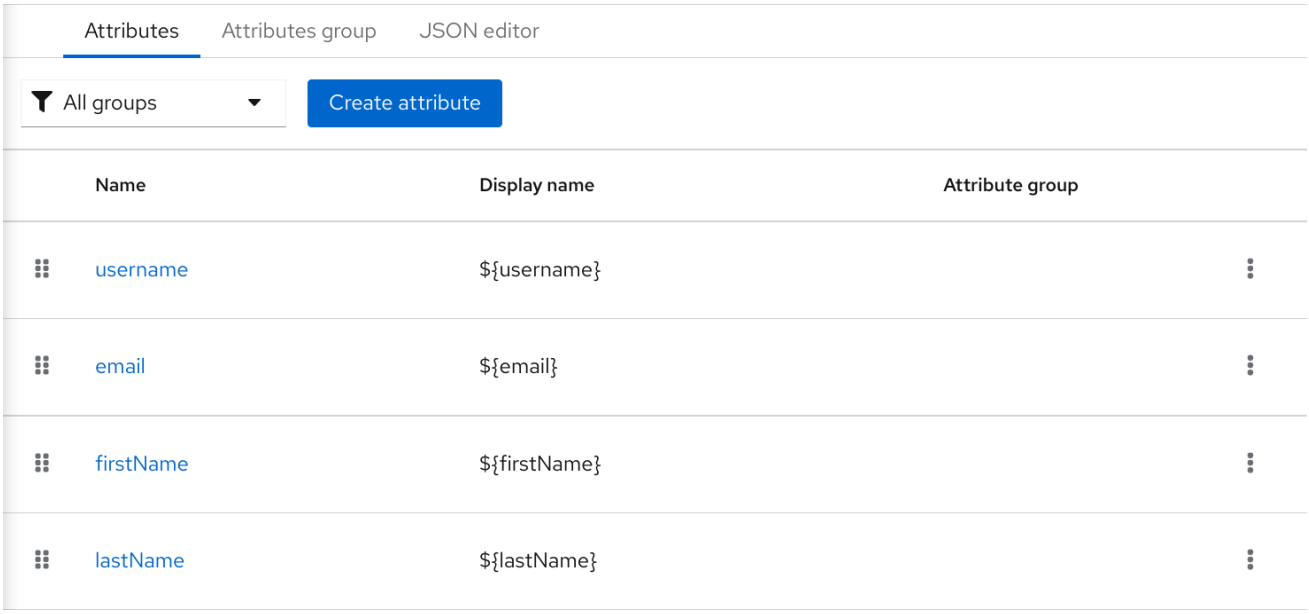
默认渲染机制提供以下功能：

- 根据设置为属性的权限显示或隐藏字段。
- 根据设置为属性的限制，呈现必填字段的标记。
- 将字段输入类型(text, date, number, select, multiselect)设置为属性。
- 根据权限设置为属性，将字段标记为只读。
- 根据将顺序设置为属性的订购字段。
- 属于同一属性组的组字段。
- 动态组属于同一属性组的字段。

#### 5.2.10.1. 排序属性

通过拖放属性行到属性列表页面中来设置属性顺序。

排序属性

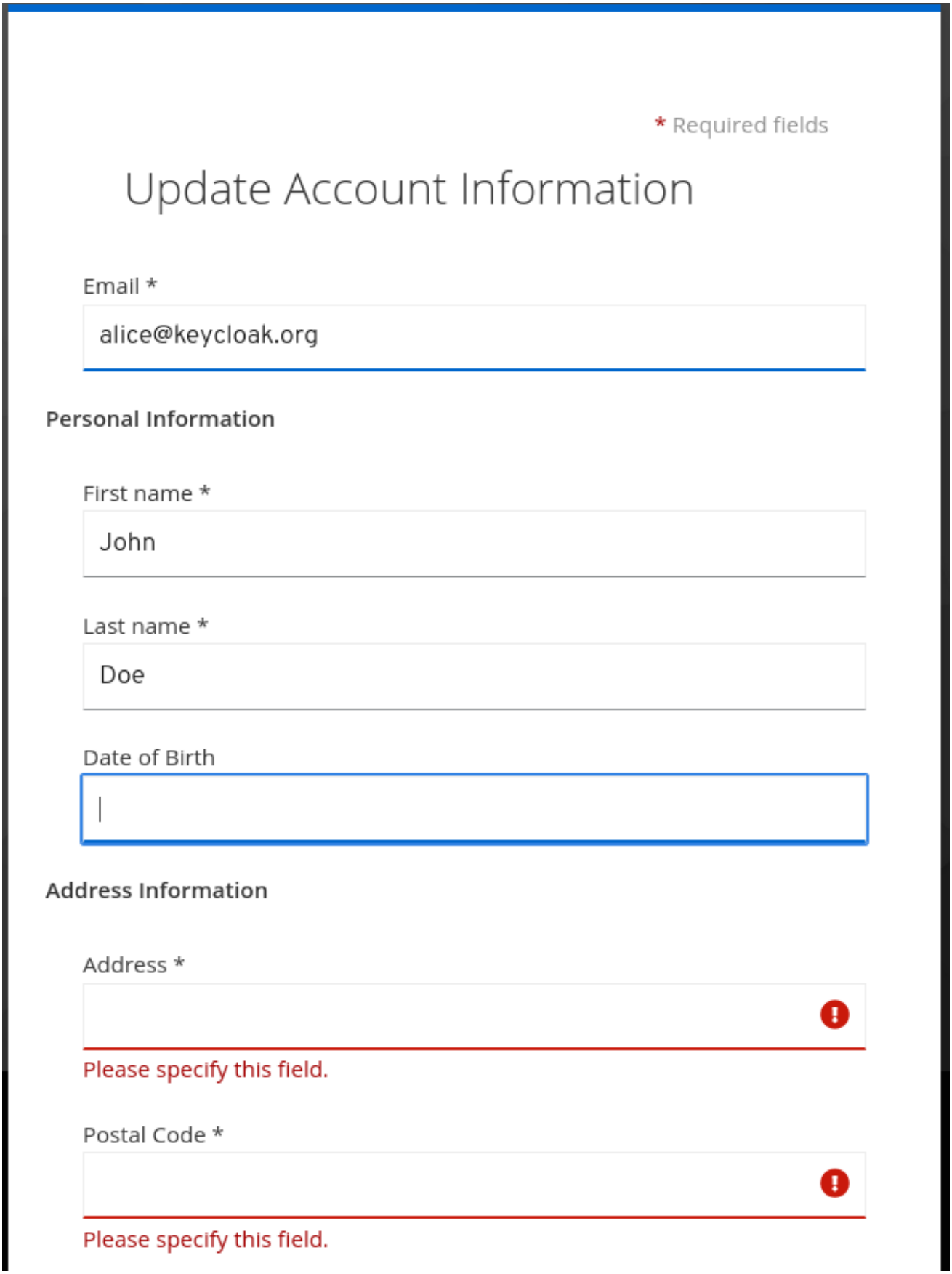



**当您以动态形式呈现字段时，在此页面中设置的顺序会被遵守。**

#### 5.2.10.2. 分组属性

**当呈现动态形式时，它们将尝试将属于同一属性组的属性分组在一起。**

动态更新配置文件表单


* Required fields
Update Account Information
Email *
alice@keycloak.org
Personal Information
First name *
John
Last name *
Doe
Date of Birth
Address Information
Address *
Please specify this field.
Postal Code *
Please specify this field.

**注意**

当属性链接到属性组时，属性顺序也很重要，以确保同一组中的属性在同一组标头中关闭。否则，如果组中的属性没有顺序，您可能有相同的组标头以动态形式多次呈现。

### 5.2.11. 启用 Progressive Profiling

为确保最终用户配置集符合配置，管理员可以使用 VerifyProfile 所需的操作，在向 Keycloak 进行身份验证时强制用户更新其配置集。

**注意**

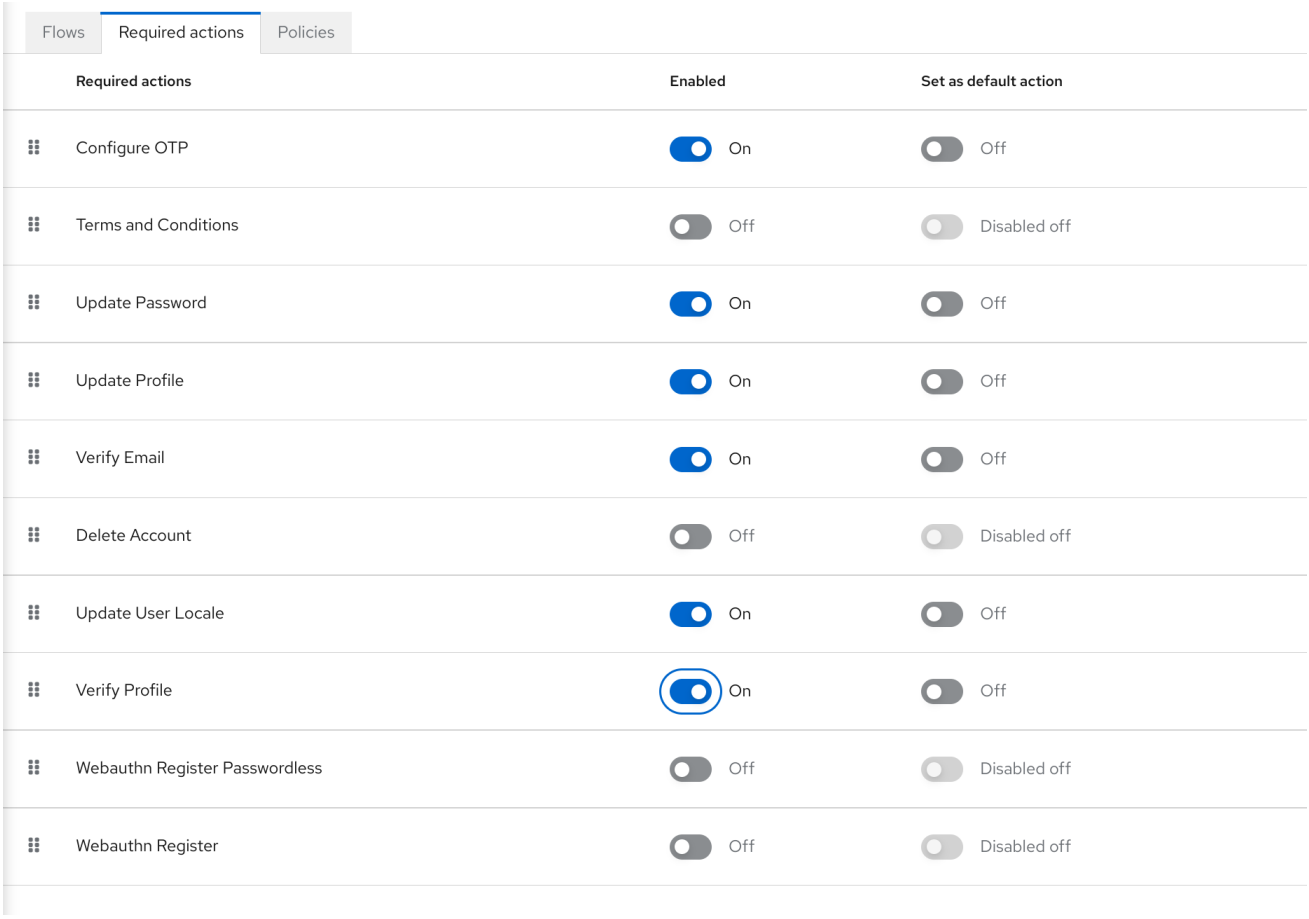
VerifyProfile 操作与 UpdateProfile 操作类似。但是，它利用 user 配置集提供的所有功能来自动强制遵守用户配置集配置。

启用后，在 验证用户 时将执行以下步骤：

- 检查 user 配置集是否与用户配置集配置为 realm 完全合规。这意味着运行验证并确保它们都成功。
- 如果没有，请在身份验证过程中执行额外的步骤，以便用户可以更新任何缺少的或无效属性。
- 如果用户配置文件与配置兼容，则不会执行额外的步骤，并且用户继续进行身份验证过程。

VerifyProfile 操作被默认启用。要禁用它，请单击左侧菜单中的 Authentication 链接，然后点 Required Actions 选项卡。在此选项卡中，使用 VerifyProfile 操作的 Enabled 开关来禁用它。

#### 注册 VerifyProfile Required Action




### 5.2.12. 使用国际消息

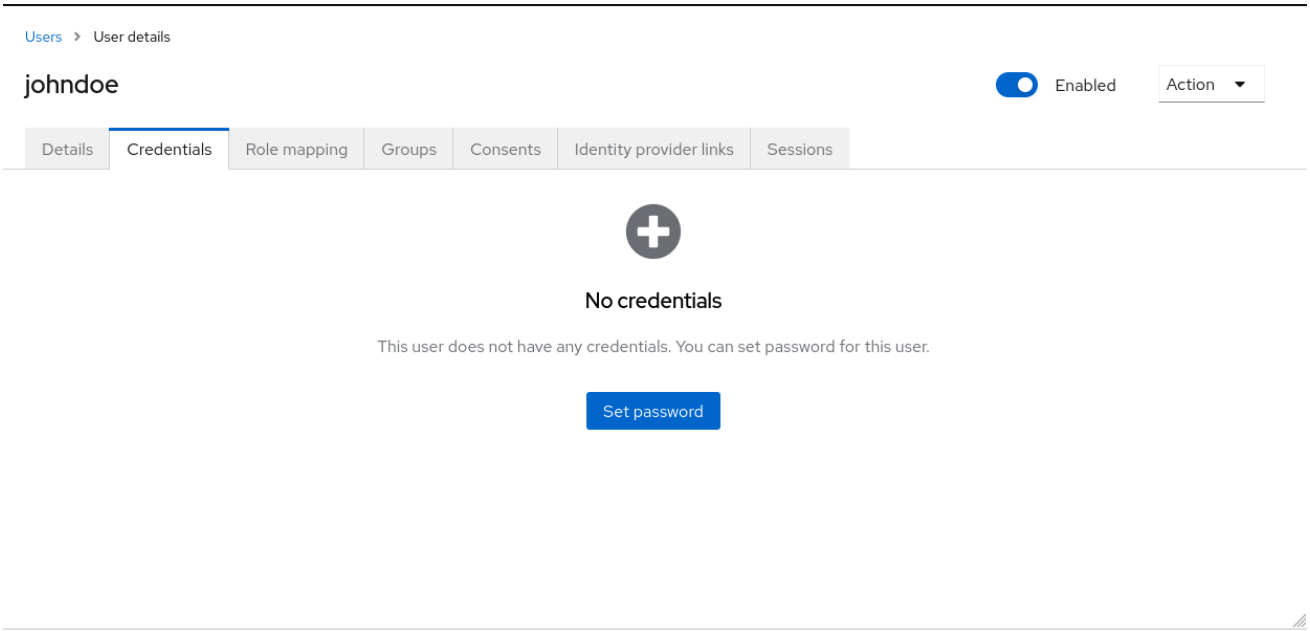
如果要在配置属性、属性组和注解时使用国际化消息，您可以使用从消息捆绑包转换为消息捆绑包的消息，设置其显示名称、描述和值。

为此，您可以使用占位符解析消息键，如 ${myAttributeName}，其中 myAttributeName 是消息捆绑包中消息的密钥。如需了解更多详细信息，请参阅 服务器开发人员指南，了解如何添加消息捆绑包到自定义主题。

## 5.3. 定义用户凭证

您可以在 Credentials 选项卡中管理用户凭证。

凭证管理




您可以通过拖放行来更改凭证的优先级。新顺序决定了该用户的凭据的优先级。最顶层的凭证具有最高优先级。优先级决定了用户在用户登录后首先显示哪个凭证。

类型

此列显示凭证类型，如密码或 OTP。

用户标签

这是一个可分配的标签，用于在登录时作为选择选项识别凭证。它可以设置为任何值来描述凭据。

**data**

这是有关凭证的非机密技术信息。默认情况下，它会被隐藏。您可以点 Show data... 来显示凭证的数据。

**Actions**

单击 Reset password 以更改用户的密码，再单击 Delete 以删除凭据。

您不能在管理控制台中为特定用户配置其他类型的凭证；该任务是用户的职责。

当用户丢失 OTP 设备或者凭证已被破坏时，您可以删除用户的凭证。您只能在 Credentials 选项卡中删除用户的凭证。

### 5.3.1. 为用户设置密码

如果用户没有密码，或者密码已被删除，则会显示 Set Password 部分。

如果用户已有密码，可以在 Reset Password 部分中重置它。

**流程**

1. 点菜单中的 Users。此时会显示 Users 页面。
2. 选择一个用户。
3. 点 Credentials 选项卡。
4. 在 Set Password 部分中输入新密码。
5. 单击 Set Password。

**注意**

如果 Temporary 为 ON，则用户必须在第一次登录时更改密码。要允许用户保留提供的密码，请将 Temporary 设置为 OFF。用户必须单击 Set Password 以更改密码。

### 5.3.2. 请求用户重置密码

您还可以请求用户重置密码。

**流程**

1. 点菜单中的 Users。此时会显示 Users 页面。
2. 选择一个用户。

3. 点 Credentials 选项卡。

4. 单击 Credential Reset。

5. 从列表中选择 Update Password。

6. 单击 Send Email。发送的电子邮件包含一个将用户定向到 Update Password 窗口的链接。

7. 另外，您可以设置电子邮件链接的有效性。这设置为 Realm Settings 中的 Tokens 选项卡中的默认预设置。

### 5.3.3. 创建 OTP

如果 OTP 在您的域中是条件，用户必须导航到红帽构建的 Keycloak 帐户控制台，以重新配置新的 OTP 生成器。如果需要 OTP，则用户在登录时必须重新配置一个新的 OTP 生成器。

或者，您可以向请求用户重置 OTP 生成器的用户发送电子邮件。如果用户已有 OTP 凭证，则应用以下步骤。

**前提条件**

- 已登陆到适当的域。

**流程**

1. 点主菜单中的 Users。此时会显示 Users 页面。

2. 选择一个用户。

3. 点 Credentials 选项卡。

4. 单击 Credential Reset。

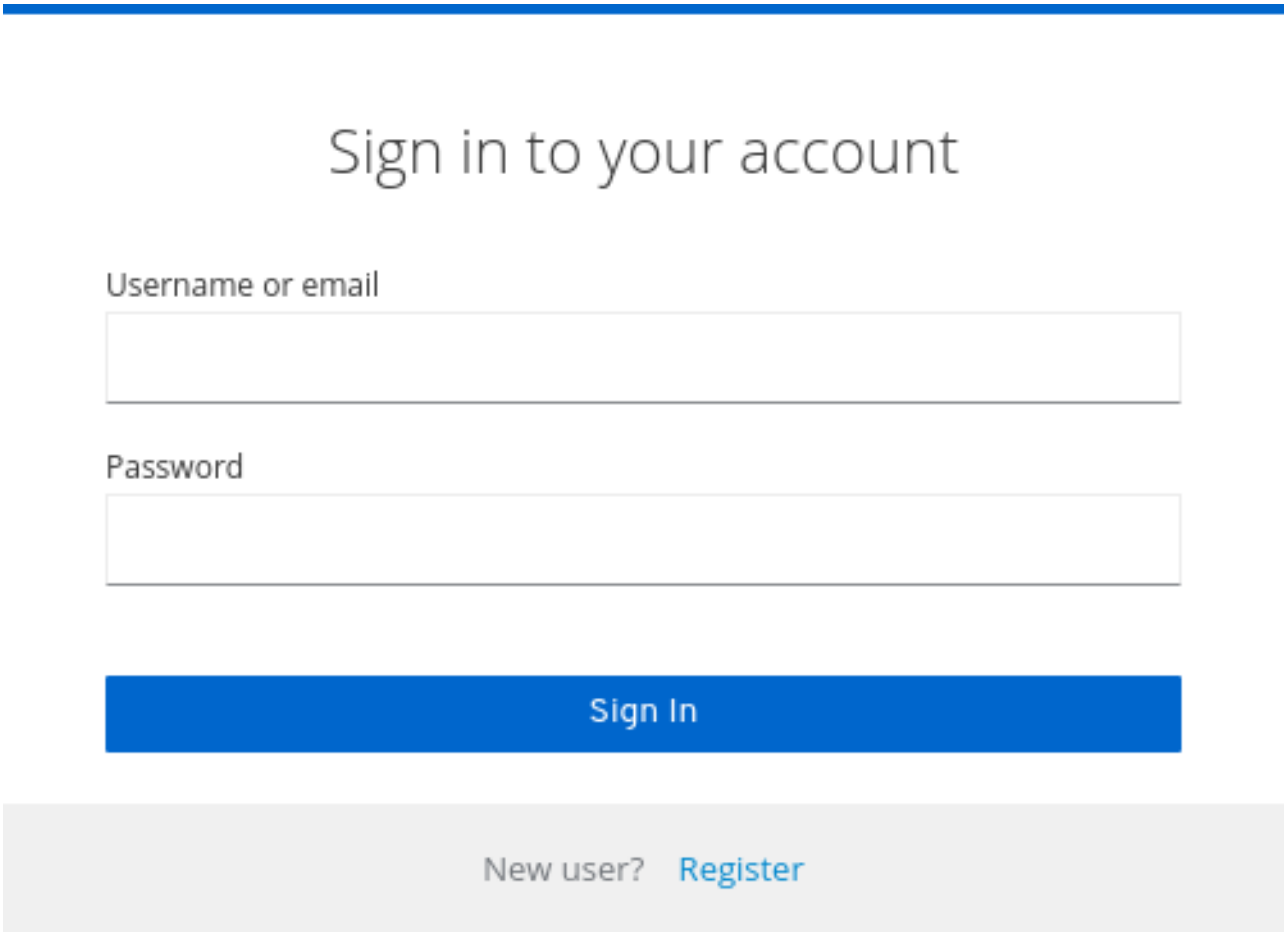
5. 将 **Reset Actions** 设置为 **Configure OTP**。

6. 单击 **Send Email**。发送的电子邮件包含一个将用户定向到 **OTP** 设置页面 的链接。

### 5.4. 允许用户自助注册

您可以使用红帽构建的 **Keycloak** 作为第三方授权服务器来管理应用程序用户，包括自助注册的用户。如果启用自我注册，登录页面会显示注册链接，以便用户可以创建帐户。

**注册链接**




用户必须向注册表中添加配置集信息才能完成注册。可以通过删除或添加用户必须完成的字段来自定义注册表单。

**阐明身份代理和管理 API**

即使禁用了自助注册，新用户仍然可以添加到 Keycloak 的红帽构建中：

- 管理员可以使用 admin 控制台（或 admin REST API）添加新用户
- 启用身份代理后，身份提供程序验证的新用户可在红帽构建的 Keycloak 存储中自动添加/注册。如需更多信息，请参阅 Identity Brokering 章节中的第一个 登录流 部分。

另外，当启用特定用户存储时，来自第三方用户存储的用户 （如 LDAP）会在红帽构建的 Keycloak 中自动提供

**其他资源**

- 有关自定义用户注册的更多信息，请参阅 服务器开发人员指南。

### 5.4.1. 启用用户注册

允许用户自助注册。

**流程**

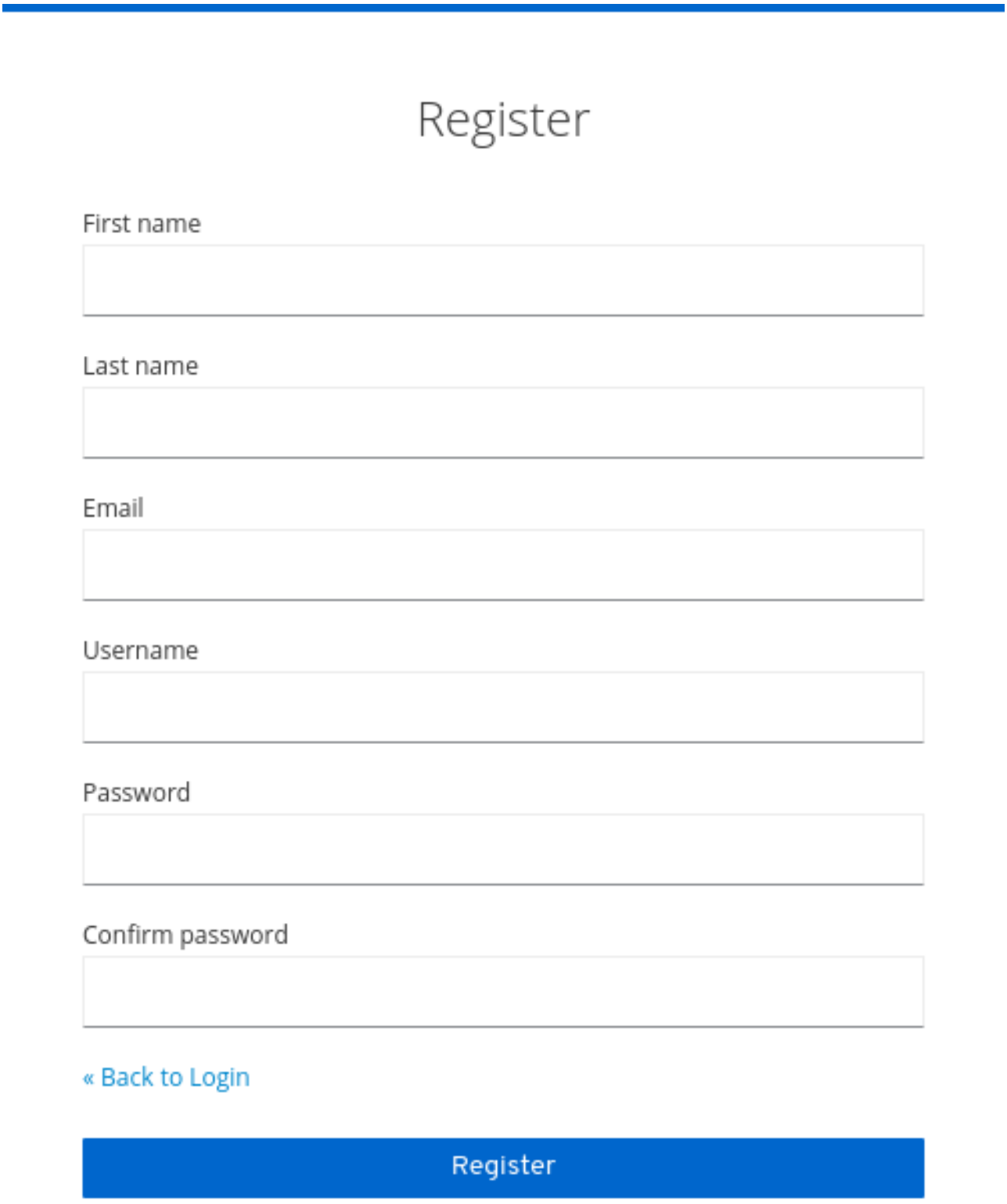
1. 在主菜单中，单击 **Realm Settings**。
2. 点 **Login** 选项卡。
3. 将用户 **注册** 切换为 **ON**。

启用此设置后，控制台控制台的登录页面上会显示 **Register** 链接。

### 5.4.2. 以新用户注册

作为新用户，您必须完成注册表单才能首次登录。您可以添加配置文件信息和要注册的密码。

**注册表单**




**前提条件**

- 启用了用户注册。

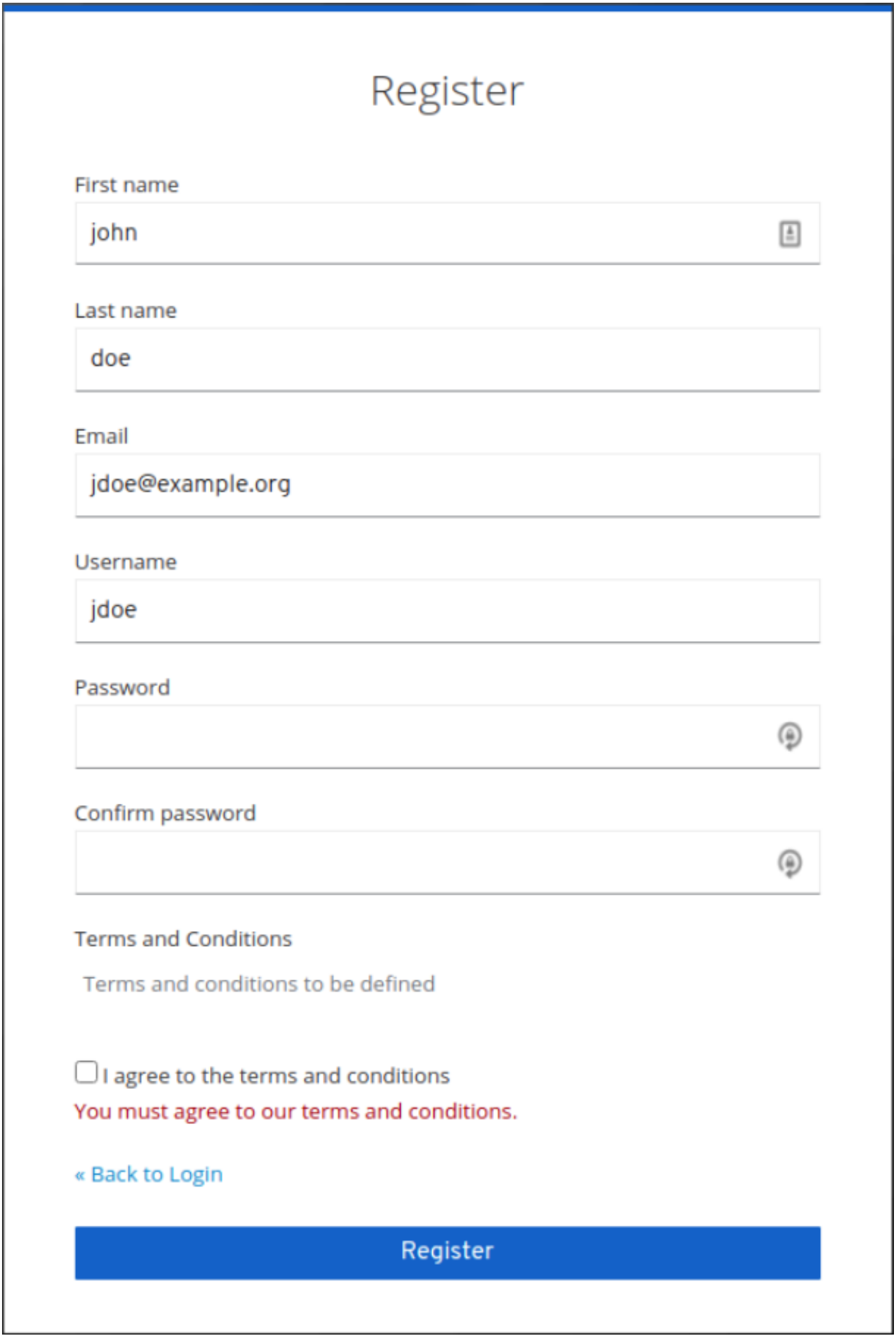
**流程**

1. 点登录页面中的 **Register** 链接。此时会显示注册页面。
2. 输入用户配置文件信息。
3. 输入新密码。
4. 点 **Register**。

### 5.4.3. 要求用户在注册过程中同意条款和条件

要注册用户，您需要同意您的条款和条件。

**符合所需条款和条件的注册表**


Register
First name
john
Last name
doe
Email
jdoe@example.org
Username
jdoe
Password
Confirm password
Terms and Conditions
Terms and conditions to be defined
I agree to the terms and conditions
You must agree to our terms and conditions.
« Back to Login
Register

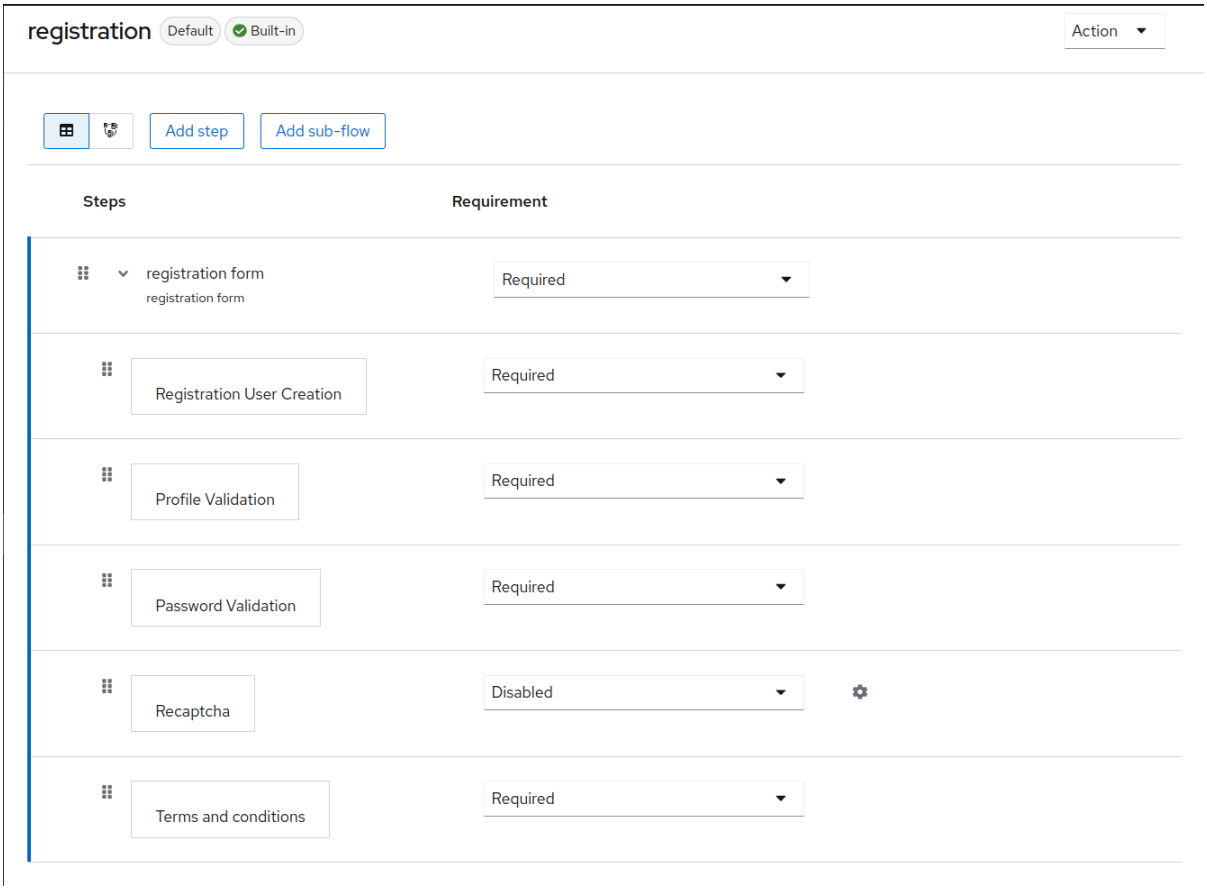
**前提条件**

- 启用了用户注册。
- 启用所需操作的条款和条件。

**流程**

1. 点菜单中的 **Authentication**。点 **Flows** 选项卡。
2. 点 **注册 流**。
3. 在 **Conditions** 和 **Conditions** 行中选择 **Required**。

**在注册时进行所需的条款和条件协议**



## 5.5. 定义登录时所需的操作

您可以设置用户在第一次登录时必须执行的操作。用户提供凭据后需要这些操作。第一次登录后，不再需要这些操作。您可以在该用户的 Details 标签页中添加所需的操作。

即使管理员没有明确添加到此用户，在登录期间会自动触发一些必要的操作。例如，如果以每 X 天更改用户密码的方式配置了 密码策略，则可以触发 更新密码 操作。或者，验证配置集 操作可能需要用户更新 User 配置集，只要某些用户属性根据用户配置集配置不匹配要求。

以下是所需操作类型的示例：

更新密码

用户必须更改密码。

配置 OTP

用户必须使用 Free OTP 或 Google Authenticator 应用程序在移动设备上配置一次性密码生成器。

验证电子邮件

用户必须验证其电子邮件帐户。将向用户发送一封电子邮件，其中包含必须单击的验证链接。成功完成此工作流后，用户就可以登录。

更新配置文件

用户必须更新配置文件信息，如名称、地址、电子邮件和电话号码。

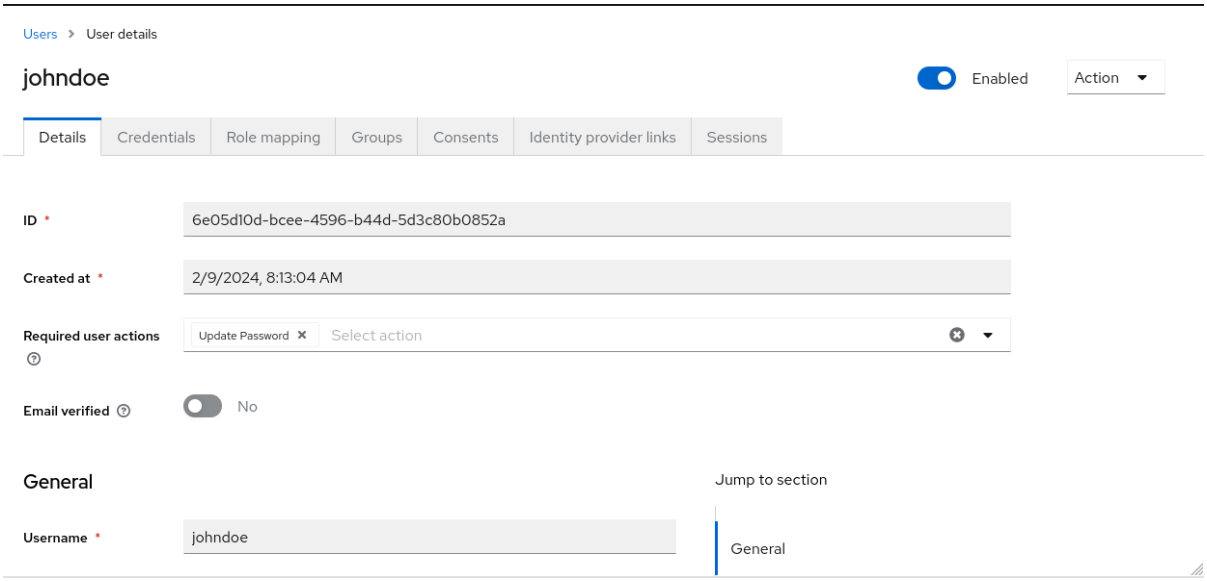
### 5.5.1. 为一个用户设置所需的操作

您可以设置任何用户所需的操作。

流程

1. 点菜单中的 Users。
2. 从列表中选择一个用户。

3. 导航到 **Required User Actions** 列表。




4. 选择您要添加到帐户的所有操作。

5. 点操作名称旁边的 **X** 将其删除。

6. 选择要添加的操作后，点 **Save**。

### 5.5.2. 为所有用户设置所需的操作

您可以指定在首次登录所有新用户前需要什么操作。该要求适用于由 **Users** 页面中的 **Add User** 按钮或登录页面中的 **Register** 链接创建的用户。

**流程**

1. 点菜单中的 **Authentication**。

2. 点 **Required Actions** 选项卡。

3. 对于一个或多个所需操作，点 **Set** 作为默认操作 列中的复选框。当新用户第一次登录时，必须执行所选操作。

### 5.5.3. 根据所需操作启用条款和条件

您可以启用所需的操作，在首次登录到红帽构建的 Keycloak 之前，必须接受条款和条件。

**流程**

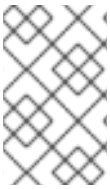
1. 点菜单中的 **Authentication**。
2. 点 **Required Actions** 选项卡。
3. 启用 **条款和条件** 操作。
4. 编辑基本登录主题中的 **terms.ftl** 文件。

**其他资源**

- 有关扩展和创建主题的更多信息，请参阅 服务器开发人员指南。

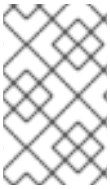
## 5.6. 应用程序启动的操作

应用程序启动的操作(AIA)允许客户端应用程序请求用户在红帽构建的 Keycloak 端执行操作。通常，当 OIDC 客户端应用程序希望用户登录时，它会将该用户重定向到登录 URL，如 OIDC 部分所述。登录后，用户将重新重定向到客户端应用程序。用户执行管理员 在上一节中 描述所需的操作，然后立即重定向到应用程序。但是，AIA 允许客户端应用程序在登录期间从用户请求一些必要的操作。即使用户已在客户端上进行身份验证并且具有活跃的 SSO 会话，也可以执行此操作。它通过将 kc_action 参数添加到 OIDC 登录 URL 中触发，其值包含所请求操作的值。例如 kc_action=UPDATE_PASSWORD 参数。



**注意**

kc_action 参数是红帽构建的 OIDC 规格不支持的 Keycloak 专有机制。



**注意**

应用程序启动的操作只支持 OIDC 客户端。

因此，如果使用 AIA，则示例流类似如下：

- 客户端应用程序使用附加参数将用户重定向到 OIDC 登录 URL，如 kc_action=UPDATE_PASSWORD
- 按照 Authentication flows 部分所述，始终触发 浏览器 流。如果用户没有经过身份验证，则该用户在正常登录过程中需要以 身份进行身份验证。如果用户已经通过了身份验证，该用户可能会被 SSO cookie 自动重新验证，而无需主动重新验证并再次提供凭据。在这种情况下，该用户将直接重定向到带有特定操作（本例中为更新密码）的屏幕。然而，在某些情况下，即使用户有 SSO cookie （详情请参阅 下面），也需要激活重新身份验证。
- 用户会显示具有特定操作的屏幕（本例中为 更新密码），以便用户需要执行特定操作
- 然后，用户被重定向到客户端应用程序

请注意，红帽构建的 Keycloak 帐户 控制台使用 AIA 来请求更新密码或重置 OTP 或 WebAuthn 等其他凭证。

警告

即使使用了 kc_action 参数，也不足以假定用户始终执行该操作。例如，用户可以从浏览器 URL 中手动删除 kc_action 参数。因此，在客户端请求 kc_action=CONFIGURE_TOTP 后，用户不会保证用户具有 OTP。如果要验证用户配置了双因素验证器，则客户端应用程序可能需要检查它是否已配置。例如，通过检查令牌中的 acr 等声明。

### 5.6.1. AIA 期间重新进行身份验证

如果用户因为活跃的 SSO 会话而已通过身份验证，则该用户通常不需要主动重新验证。但是，如果该用户在五分钟以前被主动验证的时间超过五分钟，客户端仍然可以在请求一些 AIA 时请求重新身份验证。本指南行中存在例外，如下所示：

- 操作 delete_account 将始终要求用户主动重新验证

- 操作 update_password 可能需要用户根据配置的 Maximum Authentication Age Password 策略 主动重新验证。如果没有配置策略，它将默认为五分钟。

- 如果要使用较短的重新身份验证，您仍可使用带有指定较短的值的 max_age 等参数查询参数，或者最终 提示=login，这始终需要用户来主动重新验证，如 OIDC 规范中所述。请注意，不支持使用 max_age 作为默认的五分钟（或者密码策略所描述的值）。目前，max_age 只能用于使值比默认的五分钟短。

### 5.6.2. 可能的操作

要查看所有可用的操作，请登录到 Admin 控制台，再进入右上角，点 Realm info → tab Provider info → Find provider required-action。但请注意，这可以根据在 Required 操作选项卡中为您的域启用哪些操作 进行进一步限制。

## 5.7. 搜索用户

搜索用户以查看用户的详细信息，如用户的组和角色。

### 前提条件

- 您位于用户所在的域中。

### 流程

1. 点主菜单中的 Users。此时会显示这个 Users 页面。

2. 在搜索框中输入您要搜索的用户的完整名称、姓氏、名或电子邮件地址。搜索返回符合您条件的所有用户。

   用于匹配用户的条件取决于搜索框中使用的语法：

   a. "somevalue" → 对字符串"somevalue"; 执行精确搜索；

   b. *somevalue* → 执行 infix 搜索, akin to a LIKE '%somevalue%' DB 查询;

c.
somevalue* 或 somevalue → 执行前缀搜索，类似于 LIKE 'somevalue%' DB 查询。

**注意**

在 Users 页面中执行搜索包括搜索红帽构建的 Keycloak 数据库和配置的用户联邦后端，如 LDAP。在联邦后端中找到的用户将导入到红帽构建的 Keycloak 数据库（如果它们尚不存在）。

**其他资源**

- 有关用户联邦的更多信息，请参阅用户联邦。

## 5.8. 删除用户

您可以删除不再需要应用程序的用户。如果用户被删除，则用户配置集和数据也会被删除。

**流程**

1. 点菜单中的 Users。此时会显示 Users 页面。
2. 点 View all users 来查找要删除的用户。

**注意**

或者，您可以使用搜索栏来查找用户。

3. 从您要删除的用户旁的操作菜单中，点 Delete 并确认删除。

## 5.9. 为用户启用删除帐户

如果您在管理控制台中启用了此功能，则最终用户和应用程序可以在帐户控制台中删除其帐户。启用这个功能后，您可以为特定用户提供该功能。

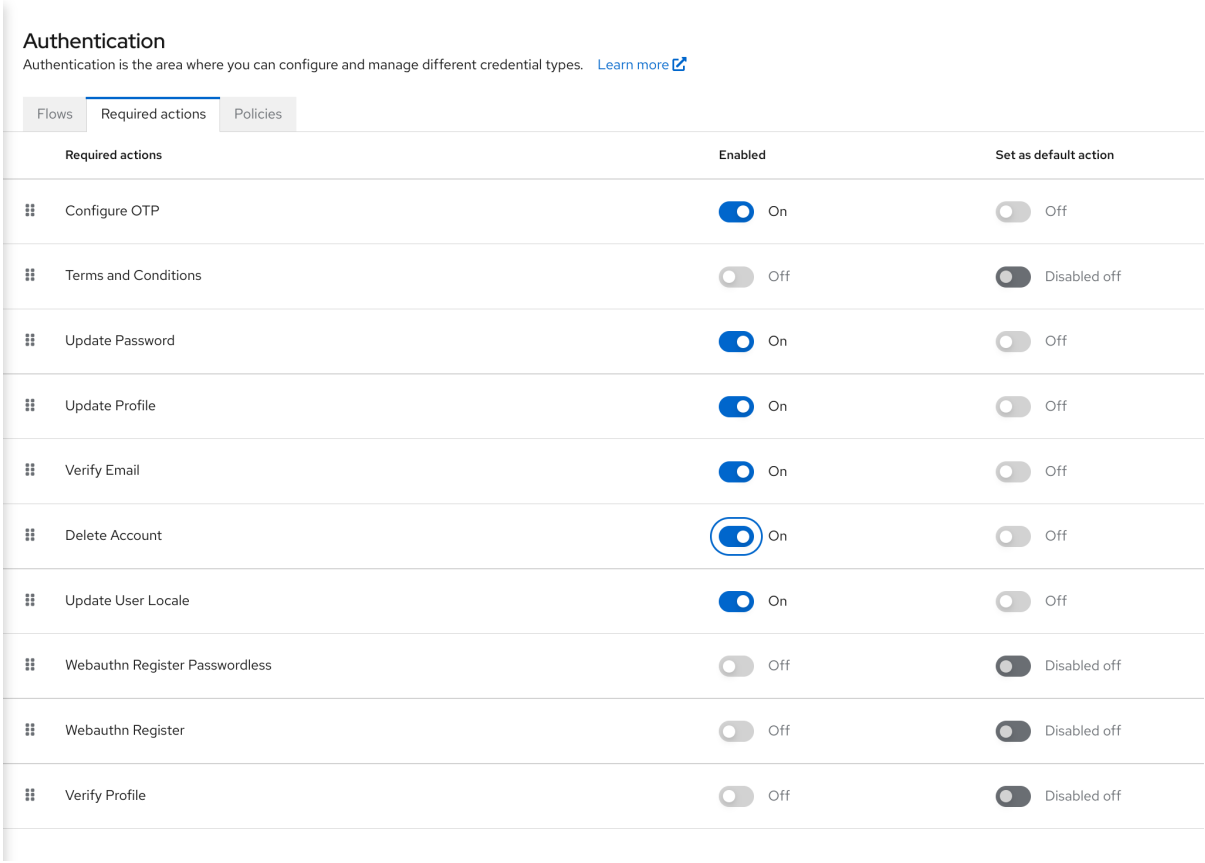
### 5.9.1. 启用删除帐户能力

您可以在 Required Actions 选项卡中启用此功能。

流程

1. 点菜单中的 Authentication。
2. 点 Required Actions 选项卡。
3. 在 Delete Account 行中选择 Enabled。

删除所需操作标签的帐户



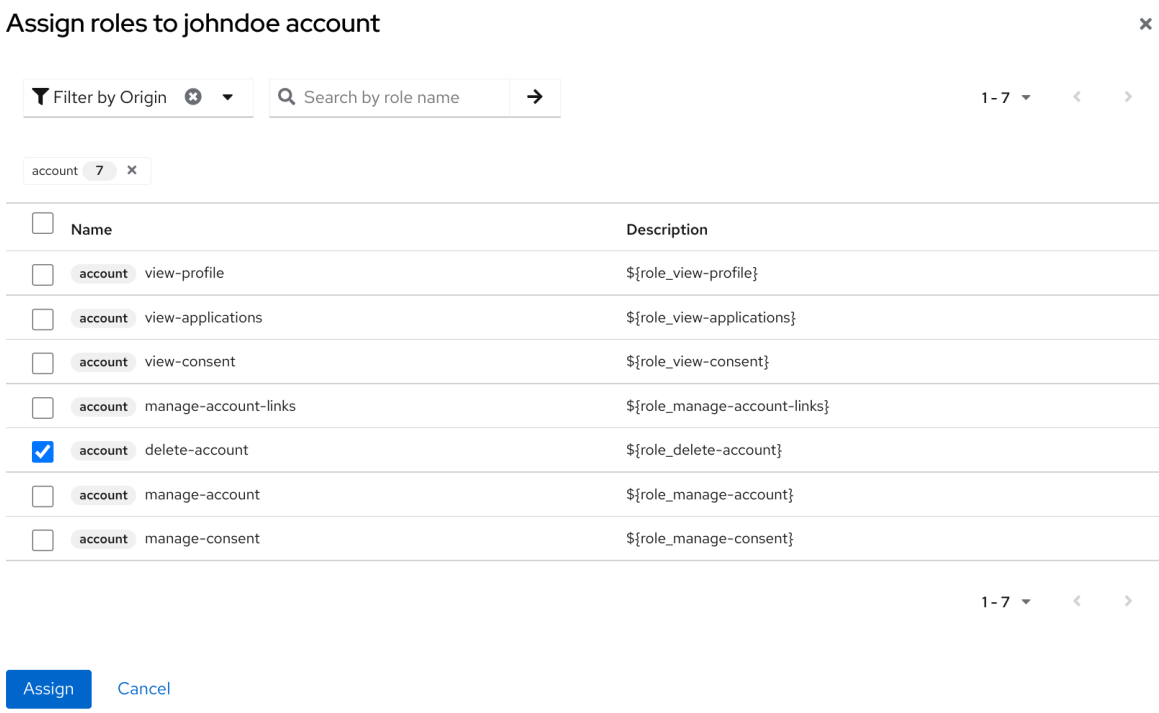

### 5.9.2. 为用户提供 delete-account 角色

您可以为特定用户赋予允许删除帐户的角色。

**流程**

1. **点菜单中的 Users。**
2. **选择一个用户。**
3. **点 Role Mappings 选项卡。**
4. **点 Assign role 按钮。**
5. **单击 account delete-account。**
6. **单击 Assign。**

**delete-account 角色**



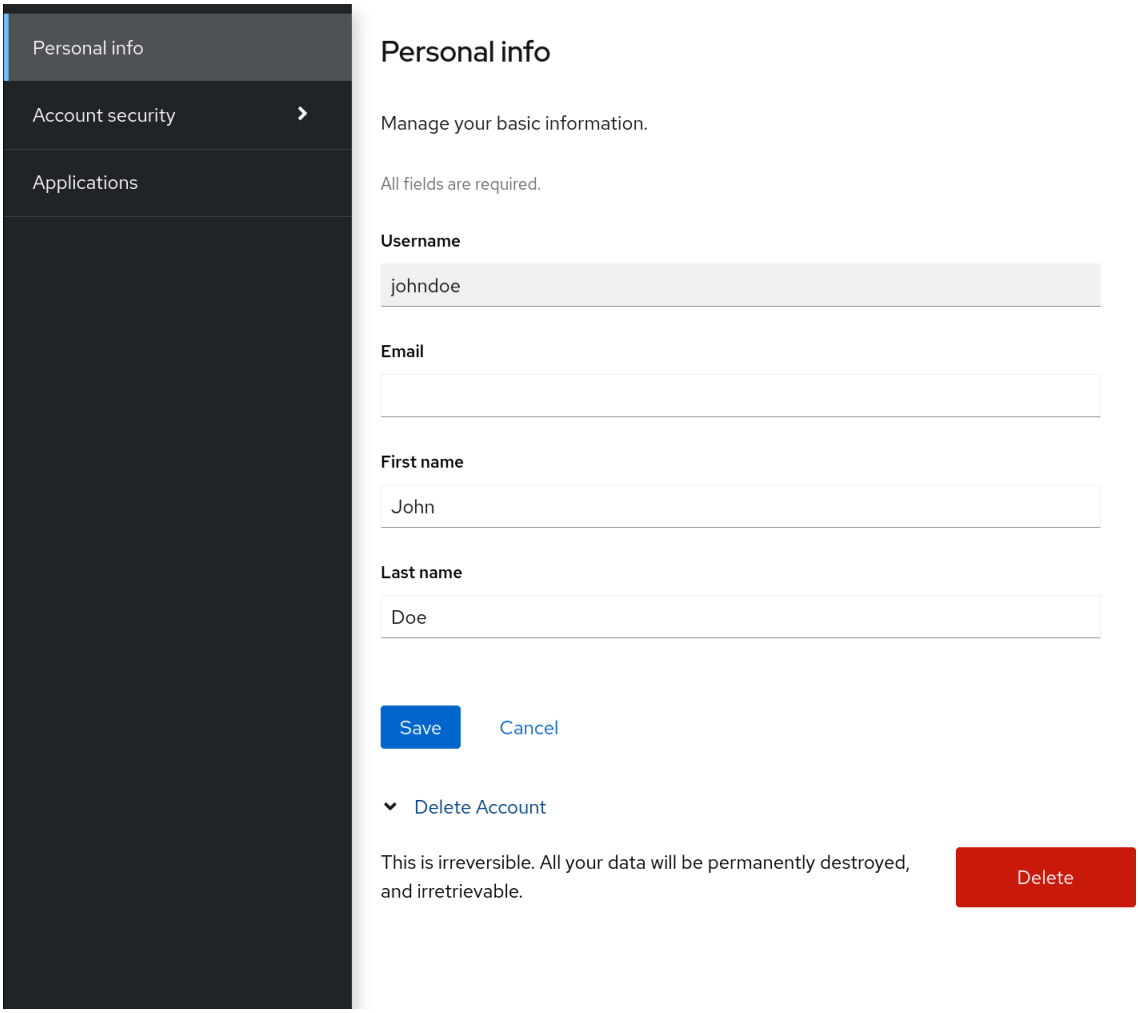

### 5.9.3. 删除您的帐户

具有 delete-account 角色后，您可以删除您自己的帐户。

1. 登录到帐户控制台。

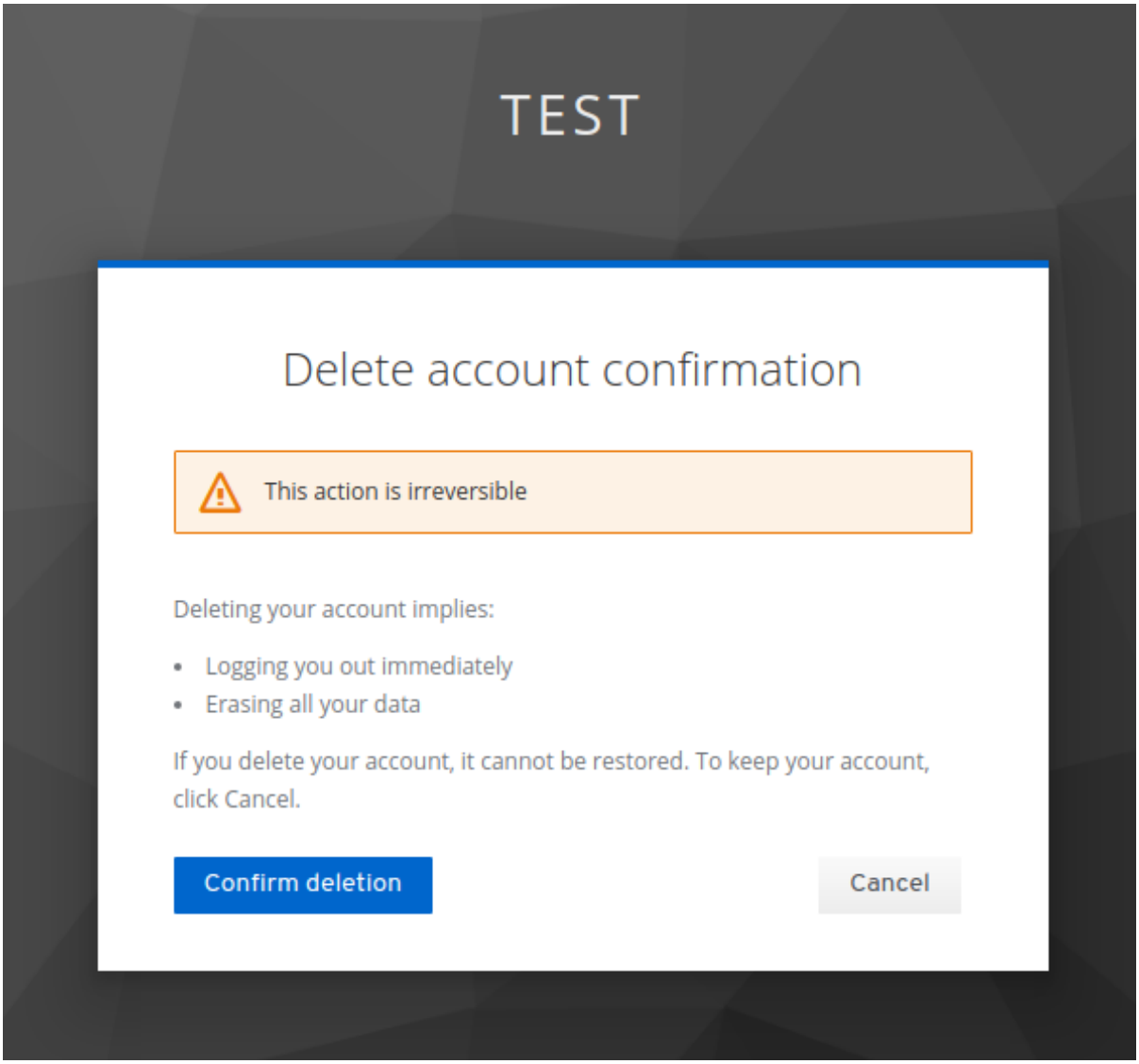
2. 在 Personal Info 页面的底部，单击 Delete Account。

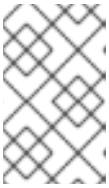
删除帐户页面




3. 输入您的凭证并确认删除。

删除确认



**注意**

**此操作不可逆。红帽构建的 Keycloak 中的所有数据都将被删除。**

## 5.10. 模拟用户

**具有适当权限的管理员可以模拟用户。例如，如果用户在应用程序中遇到错误，管理员可以模拟用户来调查或复制问题。**

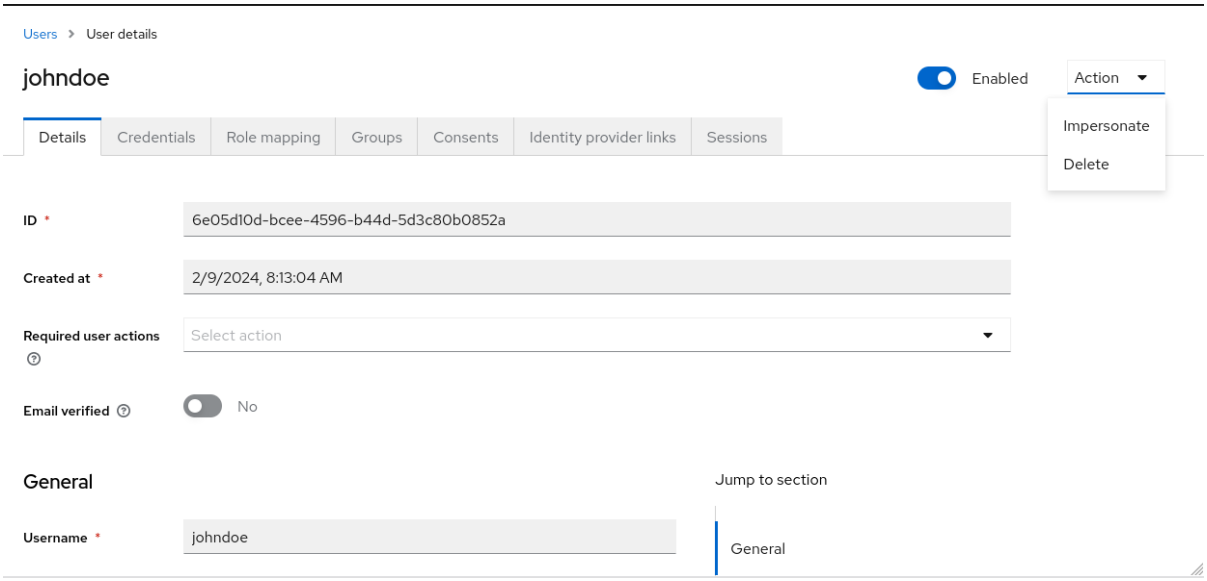
**域中具有 模拟 角色的任何用户都可以模拟用户。**

**流程**

1. 点菜单中的 Users。

2. 单击要模拟的用户。

3. 从 Actions 列表中，选择 Impersonate。




- 如果管理员和用户位于同一域中，则管理员将退出，并以模拟的用户身份自动登录。

- 如果管理员和用户位于不同的域中，管理员将保持登录，另外也将作为该用户的域中的用户登录。

在这两个实例中，会显示模拟用户的 Account Console。

### 其他资源

- 有关分配管理权限的更多信息，请参阅 管理控制台访问控制 章节。

## 5.11. 启用 RECAPTCHA

为了保护注册，红帽构建的 Keycloak 与 Google reCAPTCHA 集成。

启用 reCAPTCHA 后，您可以编辑登录主题中的 register.ftl，以配置注册页面上 reCAPTCHA 按钮的放置和样式。

**流程**

1. 在浏览器中输入以下 URL：

   ```
   https://developers.google.com/recaptcha/
   ```

2. 创建一个 API 密钥以获取您的 reCAPTCHA 站点密钥和 secret。请注意 reCAPTCHA 站点密钥和 secret，以备将来在此流程中使用。

   **注意**

   默认情况下，localhost 可以正常工作。您不必指定域。

3. 导航到红帽构建的 Keycloak 管理控制台。

4. 点菜单中的 Authentication。

5. 点 Flows 选项卡。

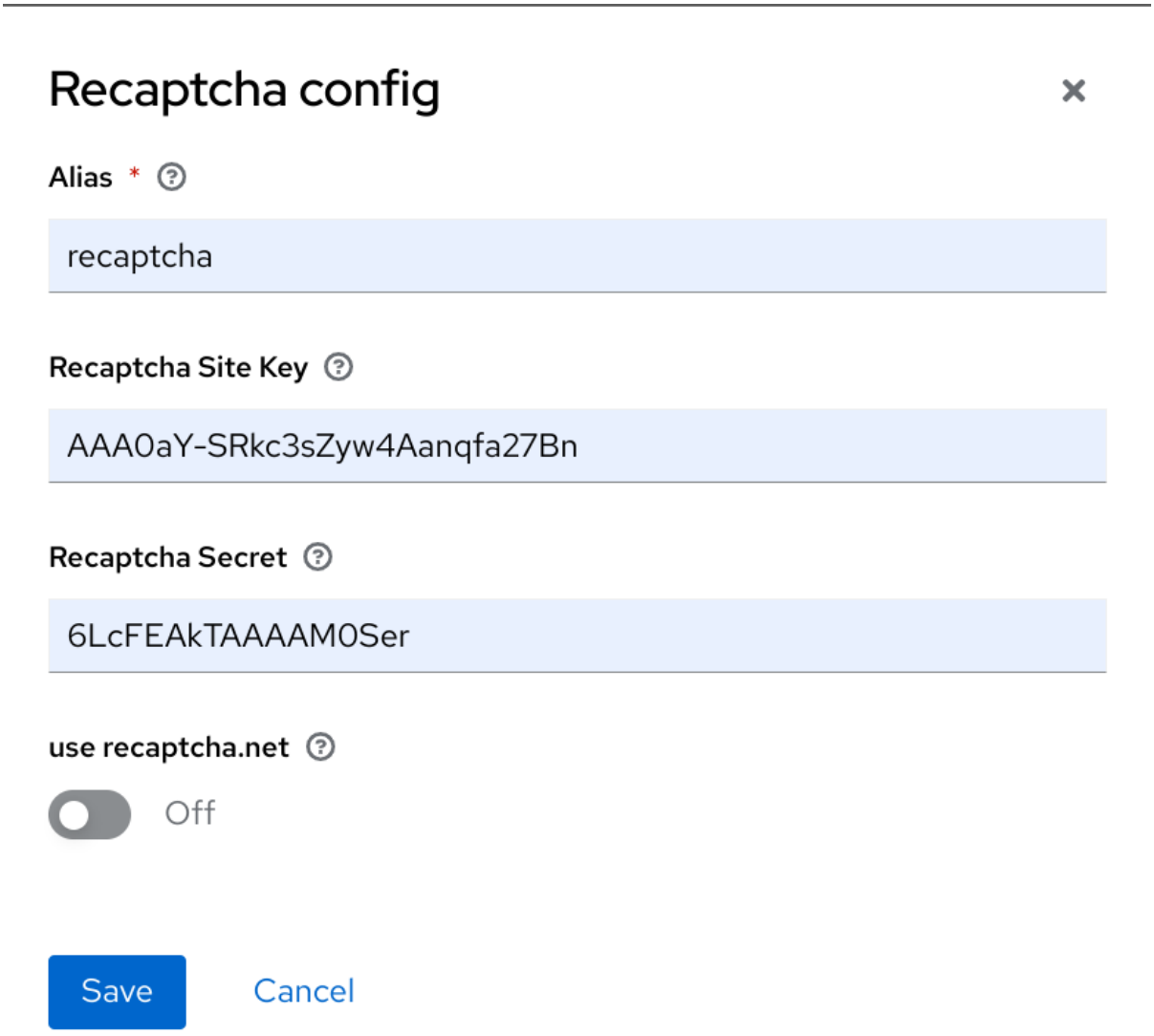
6. 从列表中选择 Registration。

7. 将 reCAPTCHA 要求设置为 所需的。这可启用 reCAPTCHA。

8. 点 reCAPTCHA 行上的 齿轮图标。

9. 单击 Config 链接。

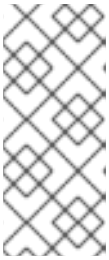
**reCAPTCHA 配置页面**




a. **输入 Google reCAPTCHA 网站生成的 Recaptcha Site Key。**

b. **输入由 Google reCAPTCHA 网站生成的 Recaptcha Secret。**

10. **授权 Google 使用注册页面作为 iframe。**



**注意**

在红帽构建的 Keycloak 中，网站不能在 iframe 中包含登录页面对话框。这个限制是防止点jacking 攻击。您需要更改红帽构建的 Keycloak 中设置的默认 HTTP 响应标头。

a. 单击菜单中的 **Realm Settings**。

b. 点 **Security Defenses** 选项卡。

c. 在 **X-Frame-Options** 标头的字段中输入 **https://www.google.com**。

d. 在 **Content-Security-Policy** 标头的字段中输入 **https://www.google.com**。

**其他资源**

- 有关扩展和创建主题的更多信息，请参阅 服务器开发人员指南。

## 5.12. 红帽构建的 KEYCLOAK 收集的个人数据

默认情况下，红帽构建的 Keycloak 会收集以下数据：

- 基本用户配置文件数据，如用户电子邮件、名字和姓氏。
- 使用社交账户时，用于社交帐户的基本用户配置文件数据，并在使用社交登录时引用社交帐户。
- 为审计和安全目的收集的设备信息，如 IP 地址、操作系统名称和浏览器名称。

红帽构建的 Keycloak 中收集的信息可高度自定义。在进行自定义时应用以下准则：

- 注册和帐户表单可以包含自定义字段，如 birthday、gender 和 nationality。管理员可以配

置红帽构建的 Keycloak，以从社交供应商或用户存储供应商（如 LDAP）检索数据。

- Red Hat build of Keycloak 会收集用户凭证，如 password、OTP Code 和 WebAuthn 公钥。这些信息会被加密并保存到数据库中，因此红帽构建的 Keycloak 管理员不可见。每种凭证类型都可以包含对管理员可见的非机密元数据，如用于哈希密码的算法以及用于哈希密码的哈希迭代数量。

- 启用授权服务和 UMA 支持后，红帽构建的 Keycloak 可以包含有关特定用户是所有者的一些对象的信息。

# 第 6 章 管理用户会话

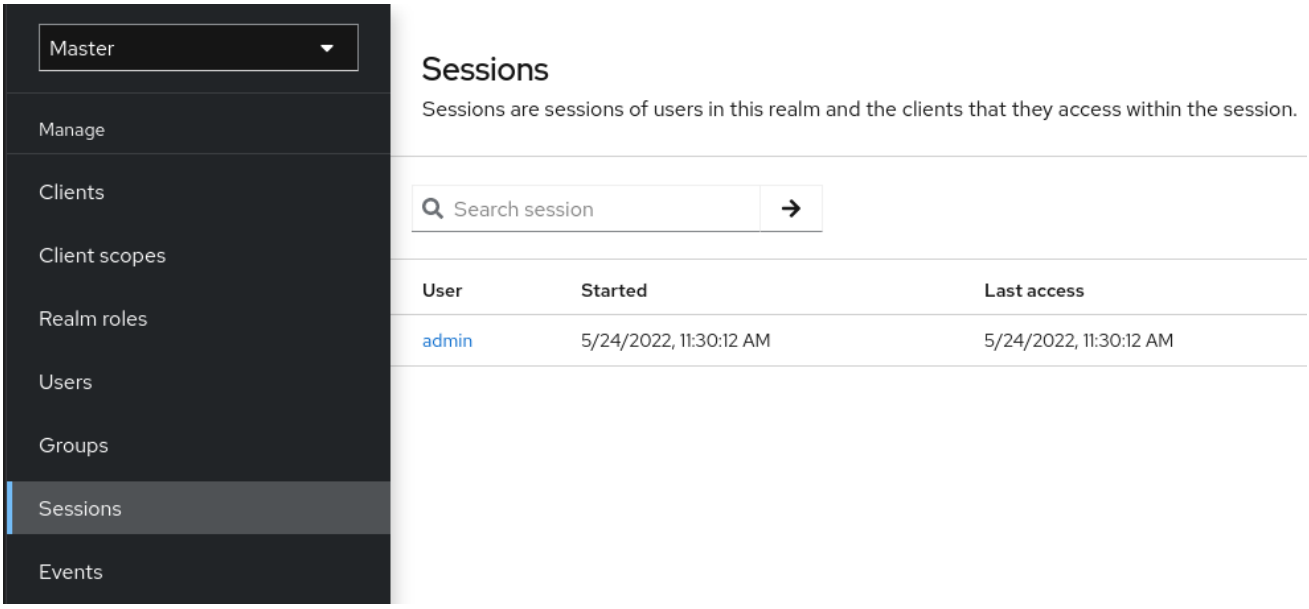
当用户登录到域时，红帽构建的 Keycloak 会为每个用户维护一个用户会话，并记住会话中用户访问的每个客户端。域管理员可以对每个用户会话执行多个操作：

- 查看域的登录统计信息。
- 查看活动的用户以及登录的位置。
- 从其会话注销用户。
- 撤销令牌。
- 设置令牌超时。
- 设置会话超时。

## 6.1. 管理会话

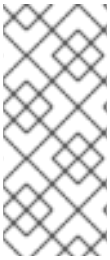
要查看红帽构建的 Keycloak 中活跃客户端和会话的顶级视图，请点击菜单中的 Session。

会话




### 6.1.1. 签名所有活跃的会话

您可以注销域中的所有用户。从 Action 列表中，选择 Sign out all active sessions。所有 SSO cookie 都无效。红帽构建的 Keycloak 通过使用红帽构建的 Keycloak OIDC 客户端适配器通知客户端。在活跃浏览器会话中请求身份验证的客户端必须再次登录。SAML 等客户端类型不会接收后备通道注销请求。



**注意**

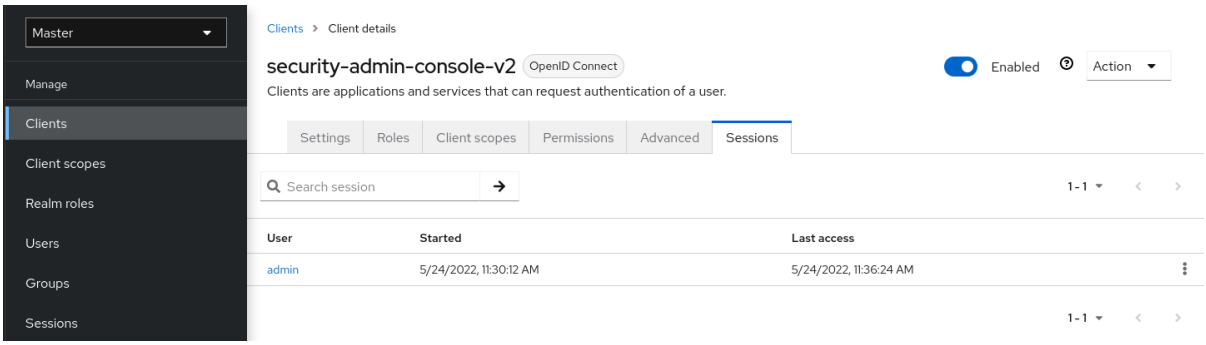
点 Sign out all active sessions 不会撤销未完成的访问令牌。未完成的令牌必须自然而过期。对于使用红帽构建的 Keycloak OIDC 客户端适配器的客户端，您可以推送 撤销策略来 撤销令牌，但这不适用于其他适配器。

### 6.1.2. 查看客户端会话

**流程**

1. 点菜单中的 Clients。
2. 点 Sessions 选项卡。
3. 点客户端查看该客户端的会话。

客户端会话

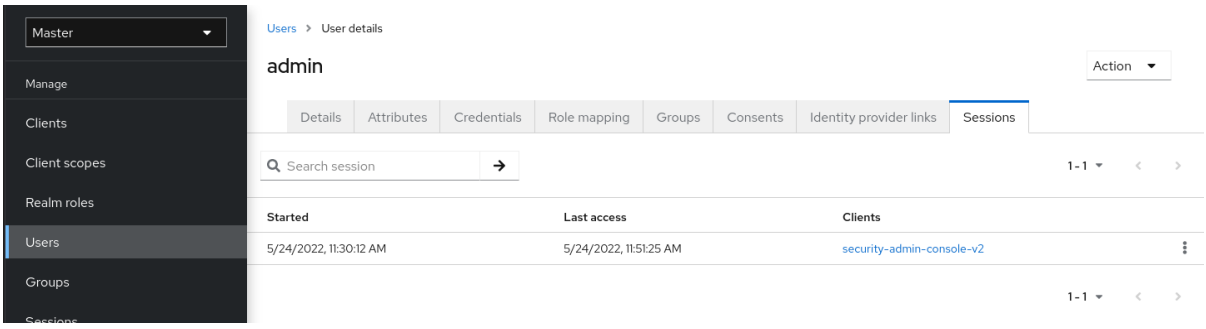



### 6.1.3. 查看用户会话

**流程**

1. **点菜单中的 Users。**
2. **点 Sessions 选项卡。**
3. **点一个用户查看该用户的会话。**

**用户会话**




## 6.2. 撤销活跃的会话

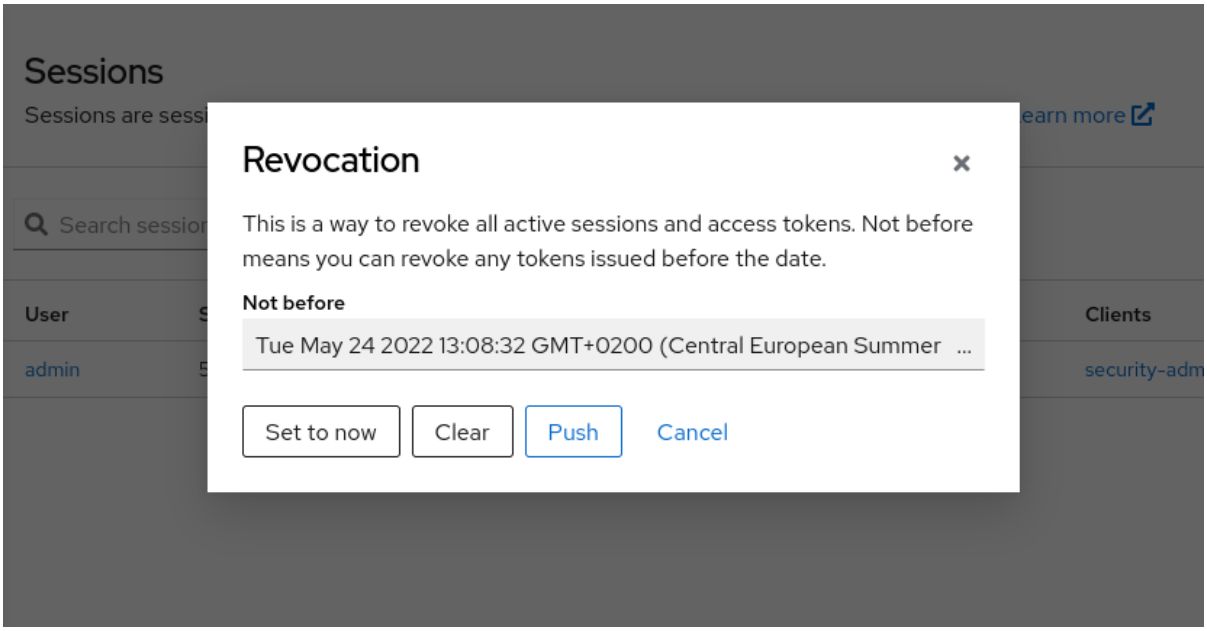
**如果您的系统被破坏，您可以撤销所有活跃的会话和访问令牌。**

**流程**

1. **点菜单中的 Sessions。**

2. **在 Actions 列表中，选择 Revocation。**

**吊销**



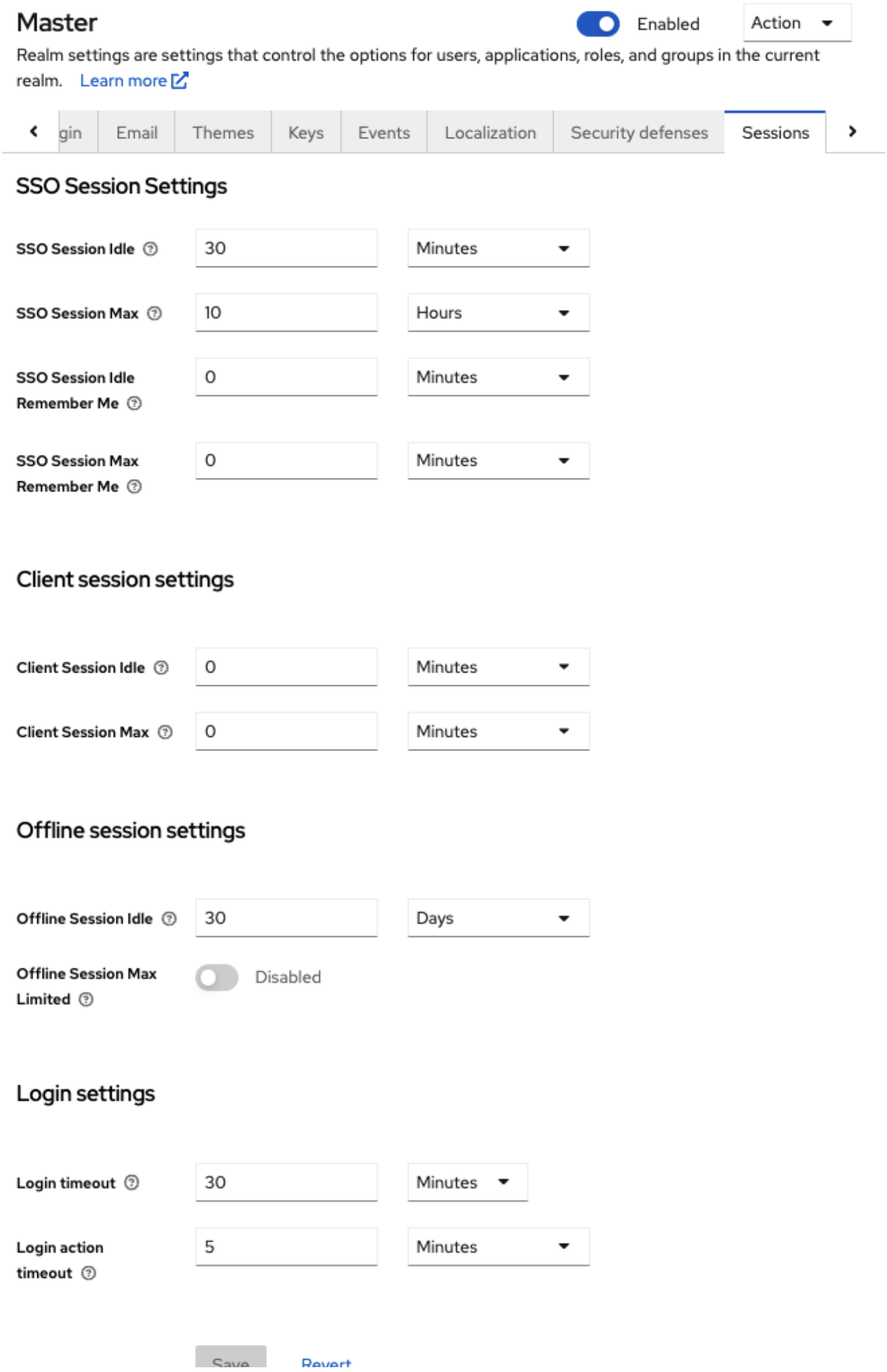

3. **指定使用此控制台前发布的会话或令牌无效的时间和日期。**

- **点 Set to now 将策略设置为当前时间和日期。**
- **点 Push 将这个撤销策略推送到使用红帽构建的 Keycloak OIDC 客户端适配器的任何注册的 OIDC 客户端。**

### 6.3. 会话和令牌超时

**红帽构建的 Keycloak 包括通过 Realm 设置 菜单中的 Sessions 和 Tokens 选项卡控制会话、Cookie 和令牌超时。**

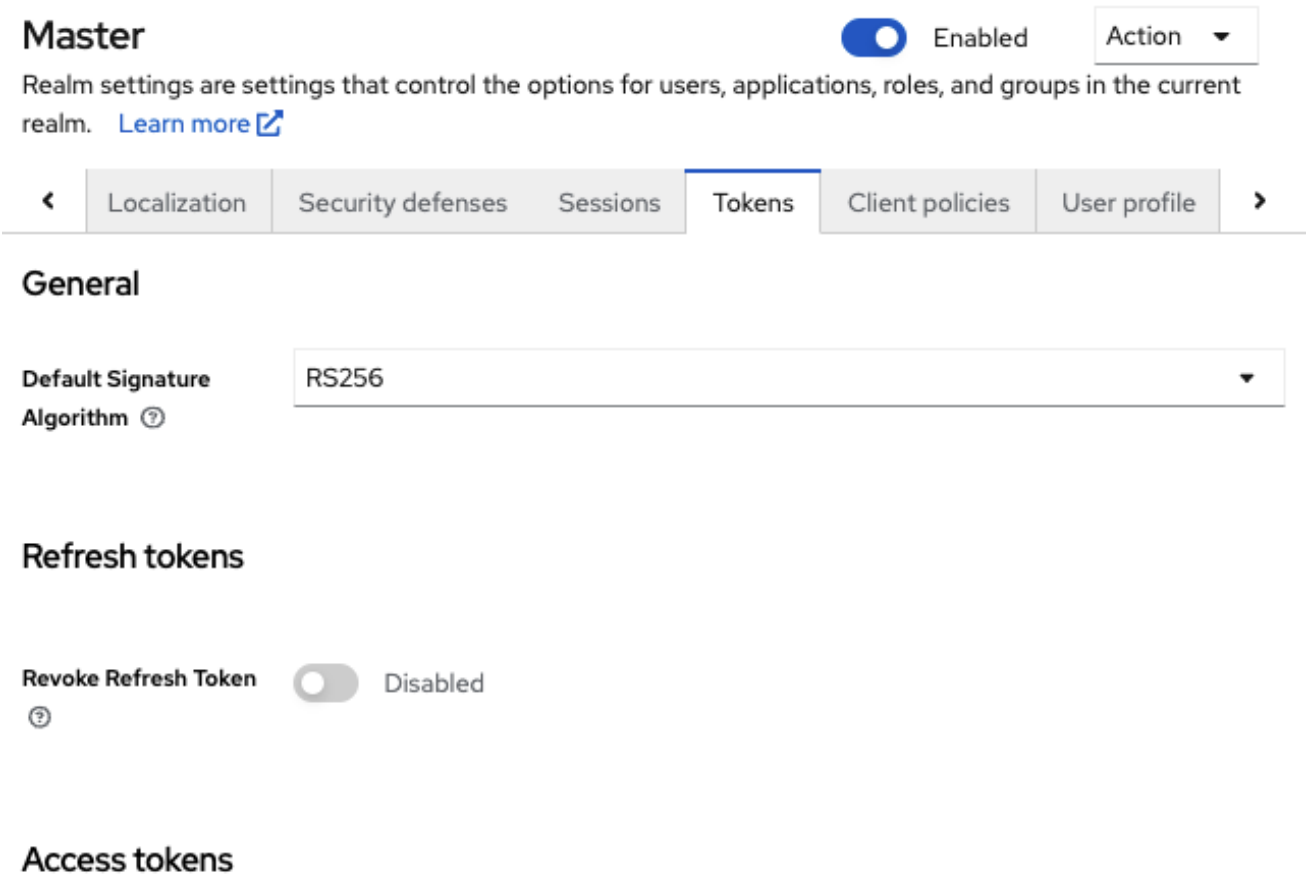
会话标签页

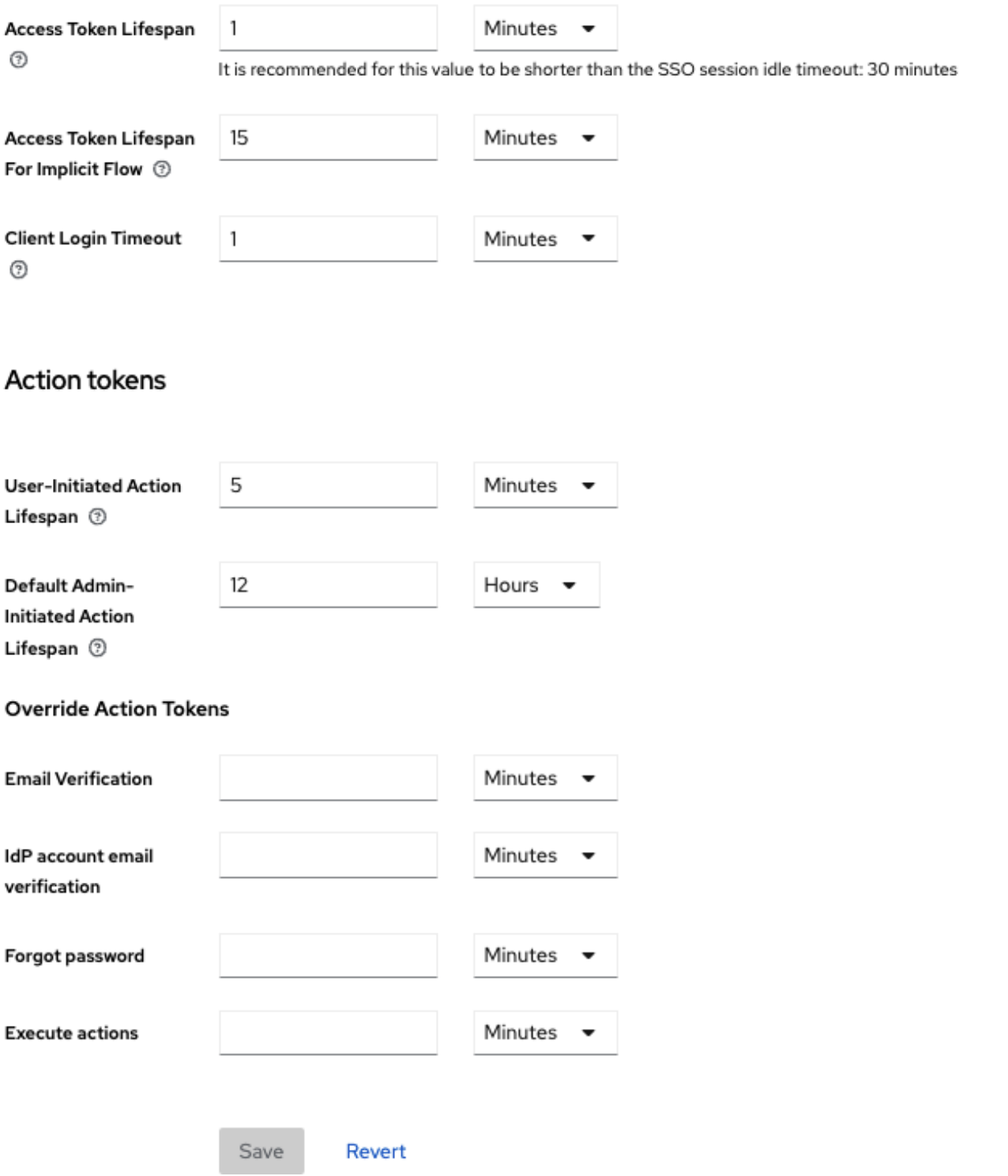

Master
Enabled
Action
Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. Learn more
gin
Email
Themes
Keys
Events
Localization
Security defenses
Sessions
SSO Session Settings
SSO Session Idle
30
Minutes
SSO Session Max
10
Hours
SSO Session Idle Remember Me
0
Minutes
SSO Session Max Remember Me
0
Minutes
Client session settings
Client Session Idle
0
Minutes
Client Session Max
0
Minutes
Offline session settings
Offline Session Idle
30
Days
Offline Session Max Limited
Disabled
Login settings
Login timeout
30
Minutes
Login action timeout
5
Minutes
Save
Revert

| Configuration | 描述 |
| --- | --- |
| SSO 会话空闲 | 此设置仅适用于 OIDC 客户端。如果用户不活跃的时间超过这个超时时间，则用户会话无效。当客户端请求身份验证或发送刷新令牌请求时，此超时值重置。Red Hat build of Keycloak 会在会话无效生效前为闲置超时时间添加一个时间窗。请参见本节后面 的备注。 |
| SSO Session Max | 用户会话过期前的最长时间。 |
| SSO 会话空闲重新成员 | 此设置与标准的 SSO 会话空闲配置类似，但特定于启用了 **Remember Me** 登录。当用户登录时点 **Remember Me** 时，可以指定较长的会话闲置超时。此设置是一个可选配置，如果其值没有大于零，它将使用与 SSO Session Idle 配置相同的空闲超时。 |
| SSO 会话最大重新成员 | 此设置与标准 SSO Session Max 类似，但特定于 **Remember Me** 登录。当用户登录时点 **Remember Me** 时，可以指定较长的会话。此设置是一个可选配置，如果其值大于零，它将使用与 SSO Session Max 配置相同的会话生命周期。 |
| 客户端会话空闲 | 客户端会话的闲置超时。如果用户不活跃的时间超过这个超时时间，客户端会话将无效，刷新令牌请求会禁止闲置超时。此设置永远不会影响一般的 SSO 用户会话，这是唯一的。请注意，SSO 用户会话是零个或多个客户端会话的父，会为用户登录的每个不同的客户端应用创建一个客户端会话。这个值应该指定比 **SSO Session Idle** 更短的空闲超时。用户可以在 **Advanced Settings** client 选项卡中覆盖单个客户端。此设置是一个可选配置，当设为零时，在 SSO Session Idle 配置中使用相同的闲置超时。 |
| 客户端会话最大数 | 客户端会话以及刷新令牌过期和无效前的最长时间。与上一个选项中所示，此设置永远不会影响 SSO 用户会话，并且应指定比 **SSO Session Max** 更短的值。用户可以在 **Advanced Settings** client 选项卡中覆盖单个客户端。此设置是一个可选配置，当设为零时，在 SSO Session Max 配置中使用相同的最大超时。 |
| 离线会话空闲 | 此设置用于 离线访问。在 Red Hat build of Keycloak 撤销其离线令牌前，会话保持闲置的时间长度。Red Hat build of Keycloak 会在会话无效生效前为闲置超时时间添加一个时间窗。请参见本节后面 的备注。 |

| Configuration | 描述 |
| --- | --- |
| 离线会话 Max Limited | 此设置用于 离线访问。如果此标志已启用，则 Offline Session Max 可以控制离线令牌保持活动状态的最长时间，而不考虑用户活动。**如果标志为 Disabled，则离线会话永远不会通过 lifespan 过期，仅通过 idle 过期。激活此选项后，可以配置 Offline Session Max（域级别上的global 选项）和 Client Offline Session Max ( 高级设置选项卡中的 特定客户端级别选项)。** |
| 离线会话最大 | 此设置用于 离线访问，这是红帽构建的 Keycloak 撤销对应的离线令牌前的最长时间。这个选项控制离线令牌激活的最大时间，无论用户活动如何。 |
| 登录超时 | 登录必须花费的总时间。如果身份验证的时间超过这个时间，用户必须再次启动身份验证过程。 |
| 登录操作超时 | 在身份验证过程中，用户可以将最大时间用于任意一个页面。 |

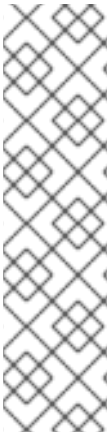
**Token 标签页**






| Configuration | 描述 |
| --- | --- |
| 默认签名算法 | 用于为域分配令牌的默认算法。 |
| 撤销刷新令牌 | **启用后**，红帽构建的 Keycloak 会撤销刷新令牌，并发出客户端必须使用的另一个令牌。此操作适用于执行刷新令牌流的 OIDC 客户端。 |

| Configuration | 描述 |
| --- | --- |
| 访问令牌生命周期 | 当红帽构建的 Keycloak 创建 OIDC 访问令牌时，这个值会控制令牌的生命周期。 |
| 访问令牌 Lifespan 用于 Implicit 流 | 使用 Implicit 流，红帽构建的 Keycloak 不提供刷新令牌。存在一个单独的超时，供 Implicit 流创建的访问令牌使用。 |
| 客户端登录超时 | 客户端必须在 OIDC 中完成授权代码流前的最长时间。 |
| 用户初始化的 Action Lifespan | 用户操作权限过期前的最长时间。缩短这个值，因为用户通常会快速响应自创建的操作。 |
| 默认 Admin-Initiated Action Lifespan | 管理员发送给用户的操作权限前的最长时间。保持这个值，以便管理员可以向离线用户发送电子邮件。管理员可以在发出令牌前覆盖默认超时。 |
| 电子邮件通知验证 | 指定用于电子邮件验证的独立超时。 |
| IdP 帐户电子邮件验证 | 为 IdP 帐户电子邮件验证指定独立超时。 |
| 忘记密码 | 指定 forgot 密码的独立超时。 |
| 执行操作 | 指定执行操作的独立超时。 |



**注意**

**对于空闲超时，会话处于活跃状态的两分钟时间窗。例如，当您将超时设置为 30 分钟时，会话过期前将为 32 分钟。**

**对于一些情况，在一些情况下，在过期前，在一个集群节点上刷新令牌需要这个操作，其他集群节点会错误地将会话视为过期，因为它们还没有收到从刷新节点成功刷新的消息。**

## 6.4. 离线访问

**在 离线访问 登录过程中，客户端应用程序会请求离线令牌，而不是刷新令牌。客户端应用保存此离线令牌，并在用户注销时将其用于以后登录。如果您的应用程序需要代表用户执行离线操作，即使用户没有在线，这个操作也会很有用。例如，常规数据备份。**

客户端应用程序负责将离线令牌保留在存储中，然后使用它从红帽构建的 Keycloak 服务器检索新的访问令牌。

刷新令牌和离线令牌之间的区别在于离线令牌永远不会过期，且不受 SSO Session Idle timeout 和 SSO Session Max lifespan 的影响。在用户注销或服务器重新启动后，离线令牌有效。您必须每 30 天或 Offline Session Idle 指定的天数使用离线令牌进行刷新令牌操作。

如果您启用 离线 Session Max Limited，则离线令牌会在 60 天后过期，即使您使用离线令牌进行刷新令牌操作。您可以在管理门户中更改这个值 Offline Session Max。

使用离线访问时，客户端闲置和最大超时可在 客户端级别 覆盖。在 client Advanced Settings 选项卡中，选项 Client Offline Session Idle and Client Offline Session Max 可让您为特定应用程序有较短的离线超时。请注意，客户端会话值也控制刷新令牌过期，但它们永远不会影响全局离线用户 SSO 会话。只有在 realm 级别中 Offline Session Max Limited 被设置为 Enabled 时，选项 Client Offline Session Max 才会被评估。

如果启用 Revoke Refresh Token 选项，则只能使用每个离线令牌一次。刷新后，您必须从刷新响应而不是上一个令牌保存新的离线令牌。

用户可以在用户帐户控制台中查看和撤销红帽构建的 Keycloak 的 离线令牌。管理员可以在 Consents 选项卡中为单个用户撤销离线令牌。管理员可以查看每个客户端的离线 访问 选项卡中发出的所有离线令牌。管理员可以通过设置 吊销策略 来撤销离线令牌。

要发出离线令牌，用户必须有 realm-level offline_access 角色的角色映射。客户端还必须在其范围内具有该角色。客户端必须添加一个 offline_access 客户端范围作为 Optional 客户端范围 到该角色，这默认为完成。

当向红帽构建的 Keycloak 发送授权请求时，客户端可以通过添加参数 scope=offline_access 来请求离线令牌。当您用来访问应用程序的安全 URL （如 http://localhost:8080/customer-portal/secured?scope=offline_access）时，红帽构建的 Keycloak OIDC 客户端适配器会自动添加此参数。如果您在身份验证请求正文中包含 scope=offline_access，则 Direct Access Grant 和 Service Accounts 支持离线令牌。

离线会话与存储在数据库中的 Infinispan 缓存旁边。每当红帽构建的 Keycloak 服务器被重启或离线会话都会从 Infinispan 缓存驱除时，它仍然在数据库中可用。一旦尝试访问离线会话，都将从数据库加载会话，还会将其导入到 Infinispan 缓存。为降低内存要求，我们引入了一个配置选项，用于缩短导入的离线会话的生命周期。这些会话将在指定生命周期后从 Infinispan 缓存驱除，但仍然在数据库中可用。这将降低内存消耗，特别是对于有大量离线会话的部署。目前，离线会话生命周期覆盖被默认禁用。要为离线用户会话指定生命周期覆盖，请使用以下参数启动红帽构建的 Keycloak 服务器：

-

```
--spi-user-sessions-infinispan-offline-session-cache-entry-lifespan-override=<lifespan-in-seconds>
```

与离线客户端会话类似：

```
--spi-user-sessions-infinispan-offline-client-session-cache-entry-lifespan-override=<lifespan-in-seconds>
```

### 6.5. 离线会话预加载

除了 Infinispan 缓存外，离线会话存储在数据库中，这意味着即使在服务器重启后也会可用。默认情况下，在服务器启动过程中，离线会话不会从数据库加载到 Infinispan 缓存中，因为如果很多离线会话被预加载，则这种方法存在缺陷。它可能会显著减慢服务器启动时间。因此，默认从数据库获取离线会话。

但是，红帽构建的 Keycloak 可以配置为在服务器启动期间将离线会话从数据库加载到 Infinispan 缓存中。这可以通过将 userSessions SPI 中的 preloadOfflineSessionsFromDatabase 属性设置为 true 来实现。这个功能当前已弃用，并将在以后的发行版本中删除。

以下示例演示了如何配置离线会话预加载。

```
bin/kc.[sh|bat] start --features-enabled offline-session-preloading --spi-user-sessions-infinispan-preload-offline-sessions-from-database=true
```

### 6.6. 临时会话

您可以在红帽构建的 Keycloak 中进行临时会话。当使用临时会话时，Red Hat build of Keycloak 不会在成功身份验证后创建用户会话。红帽构建的 Keycloak 创建一个临时的临时会话，用于成功验证用户的当前请求的范围。红帽构建的 Keycloak 可以在身份验证后使用临时会话运行 协议 映射程序。

当令牌通过临时会话发布时，令牌的 sid 和 session_state 通常为空。因此，在临时会话过程中，客户端应用程序无法刷新令牌或验证特定的会话。有时这些操作是不必要的，您可以避免额外的资源使用持久性用户会话。此会话可保存性能、内存和网络通信（在集群和跨数据中心环境）资源。

目前，临时会话会在禁用令牌刷新 的服务帐户身份验证 过程中自动使用。请注意，令牌刷新会在服务帐户身份验证过程中自动禁用，除非客户端交换机 使用刷新令牌为客户端凭证授予。

# 第 7 章 使用角色和组分配权限

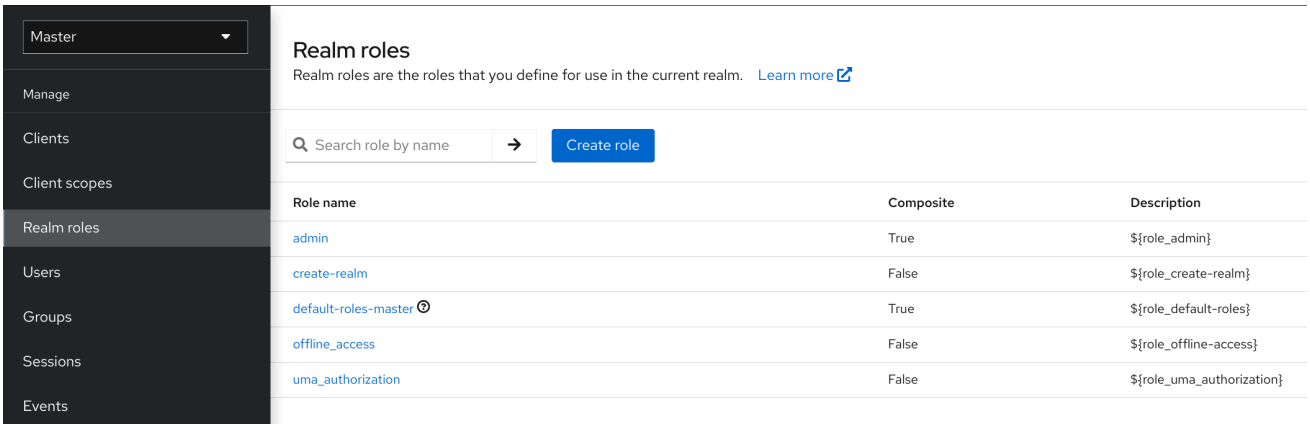
角色和组具有类似的用途，即授予用户使用应用的访问权限和权限。组是应用角色和属性的用户集合。角色定义特定的应用权限和访问控制。

角色通常适用于一种用户。例如，一个机构可能会包括 admin, user, manager, 和 employee 角色。应用程序可以为角色分配访问权限和权限，然后将多个用户分配给该角色，以便用户具有相同的访问权限和权限。例如，管理控制台具有授予用户访问管理控制台不同部分的权限的角色。

角色有一个全局命名空间，每个客户端也具有自己的专用命名空间，可以在其中定义角色。

## 7.1. 创建 REALM 角色

realm 级别角色是一个命名空间，用于定义您的角色。要查看角色列表，请单击菜单中的 Realm Roles。

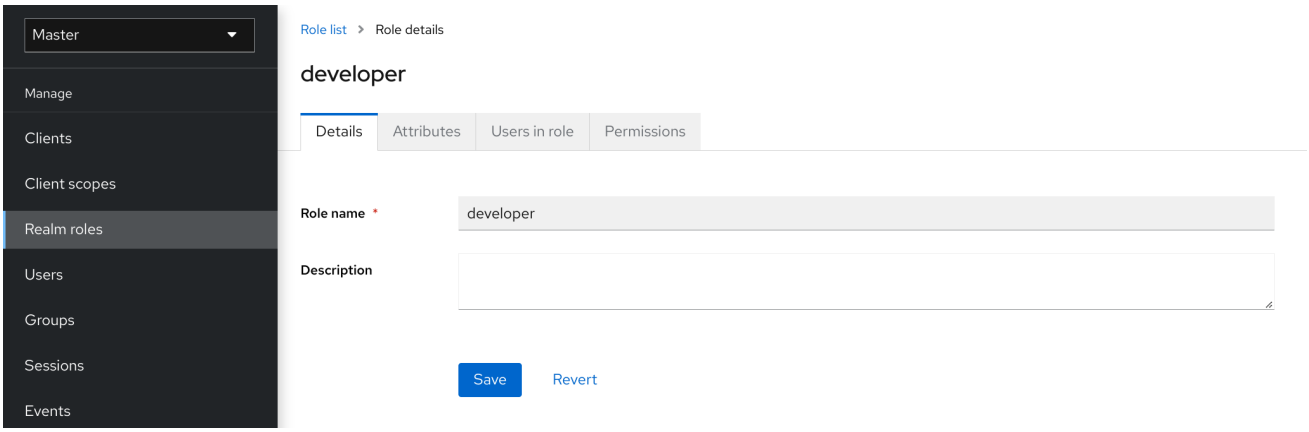



**流程**

1. 单击 **Create Role**。
2. 输入 角色名称。
3. 输入 描述。

4. 点 Save。

### 添加角色




description 字段可以通过使用 ${var-name} 字符串指定替换变量来本地化。localized 值被配置为 themes 属性文件中的主题。如需了解更多详细信息，请参阅 服务器开发人员指南。

## 7.2. 客户端角色

客户端角色是专用于客户端的命名空间。每个客户端获得自己的命名空间。客户端角色在各个客户端的 Roles 选项卡下进行管理。您与 UI 交互的方式与 realm 级别角色相同。

## 7.3. 将角色转换为复合角色

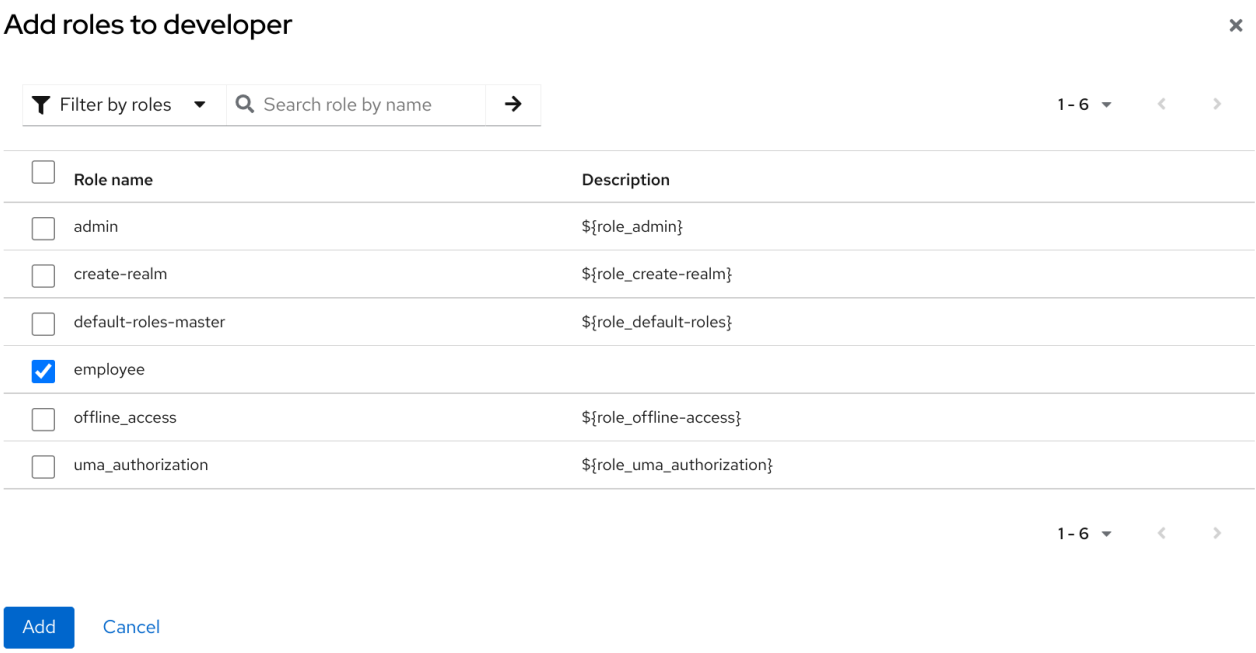
任何 realm 或 client level 角色都可以成为 *复合角色*。*复合角色是一个角色*，它关联有一个或多个额外的角色。当复合角色映射到用户时，用户会获得与复合角色关联的角色。这个继承是递归的，因此用户也会继承任何复合的复合。但我们建议不会过度使用复合角色。

### 流程

1. 单击菜单中的 Realm Roles。
2. 点您要转换的角色。

3. 从 Action 列表中，选择 Add associated roles。

复合角色




角色选择 UI 显示在页面中，您可以将域级别和客户端级别角色关联到您要创建的复合角色。

在本例中，员工 领域级角色 与开发人员 复合角色相关联。具有 developer 角色的任何用户也继承 员工 角色。

注意

在创建令牌和 SAML 断言时，任何复合也关联角色添加到发送到客户端的身份验证响应的声明中。

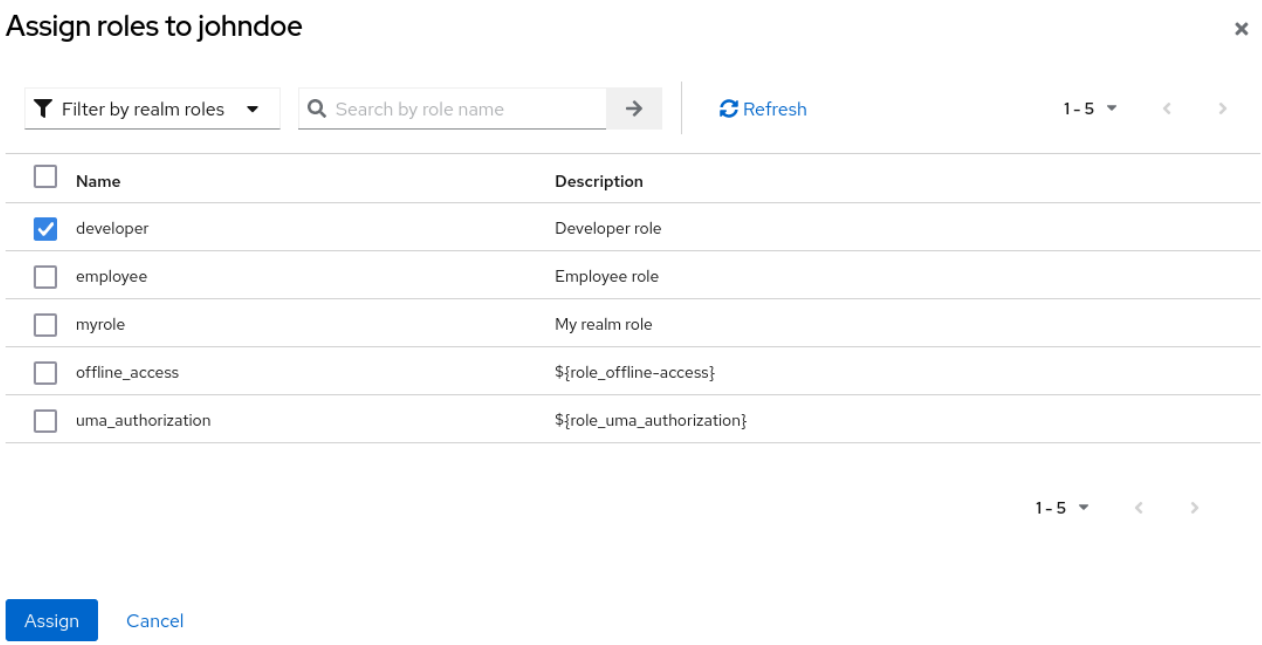
## 7.4. 分配角色映射

您可以通过该用户的 Role Mappings 选项卡为用户分配角色映射。

流程

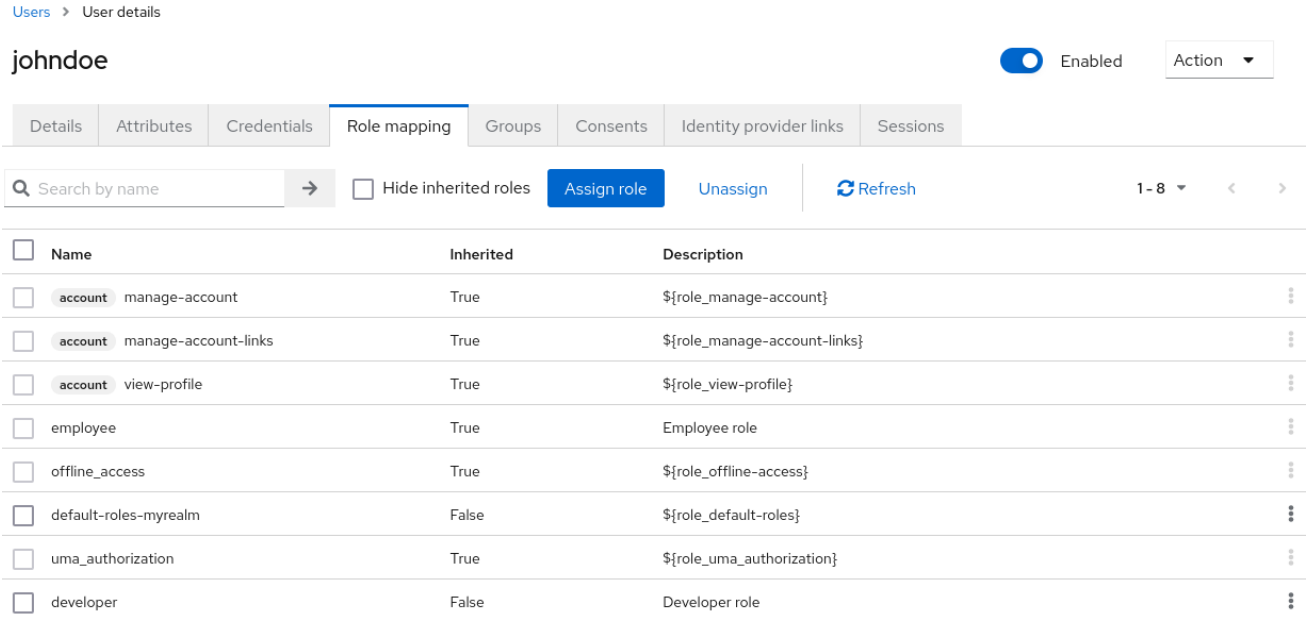
1. 点菜单中的 **Users**。
2. 点您要执行角色映射的用户。
3. 点 **Role mappings** 选项卡。
4. 单击 **Assign role**。
5. 从对话框中选择要分配给用户的角色。
6. 单击 **Assign**。

**角色映射**




在前面的示例中，我们将复合角色 开发人员 分配给用户。该角色是在 Composite Roles 主题中创建的。

**有效角色映射**




当分配 developer 角色时，与开发人员复合关联的 员工 角色会显示 Inherited "True"。继承 的角色是明确分配给从复合继承的用户和角色的角色。

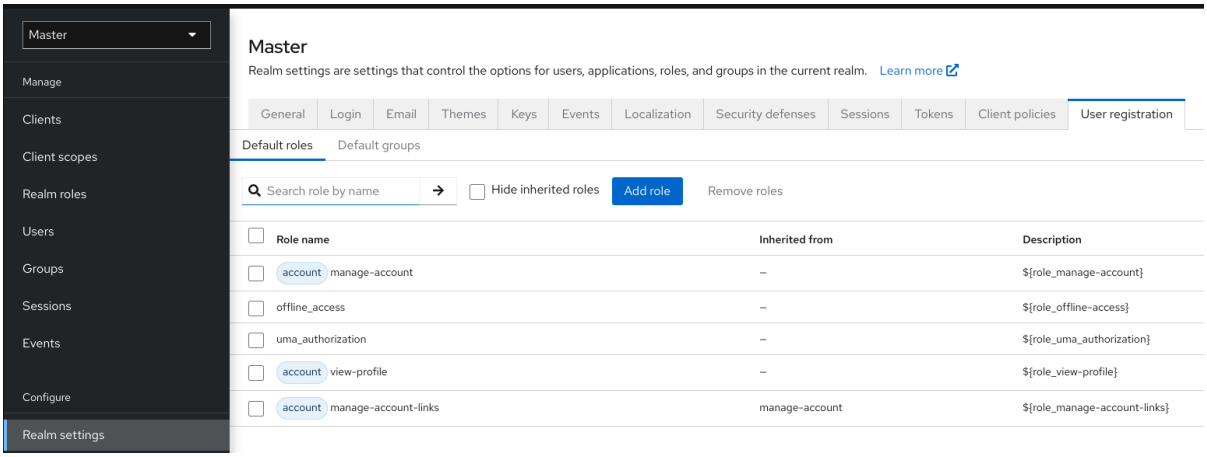
### 7.5. 使用默认角色

当用户通过 Identity Brokering 创建或导入时，使用默认角色自动分配用户角色映射。

**流程**

1. 点菜单中的 **Realm settings**。
2. 点 **User registration** 选项卡。

**默认角色**




此屏幕截图显示一些 *默认角色* 已经存在。

## 7.6. 角色范围映射

在创建 OIDC 访问令牌或 SAML 断言时，用户角色映射成为令牌或断言中的声明。应用程序使用这些声明来对应用程序控制的资源做出访问决策。Red Hat build of Keycloak 数字签名访问令牌和应用程序重新使用它们调用远程保护的 REST 服务。但是，这些令牌存在关联的风险。攻击者可以获取这些令牌，并使用其权限来破坏您的网络。要防止这种情况，请使用 *角色范围映射*。

*角色范围* 映射限制访问令牌内声明的角色。当客户端请求用户身份验证时，它们接收的访问令牌仅包含为客户端范围明确指定的角色映射。结果是，您可以限制每个独立访问令牌的权限，而不是授予客户端对所有用户权限的权限。

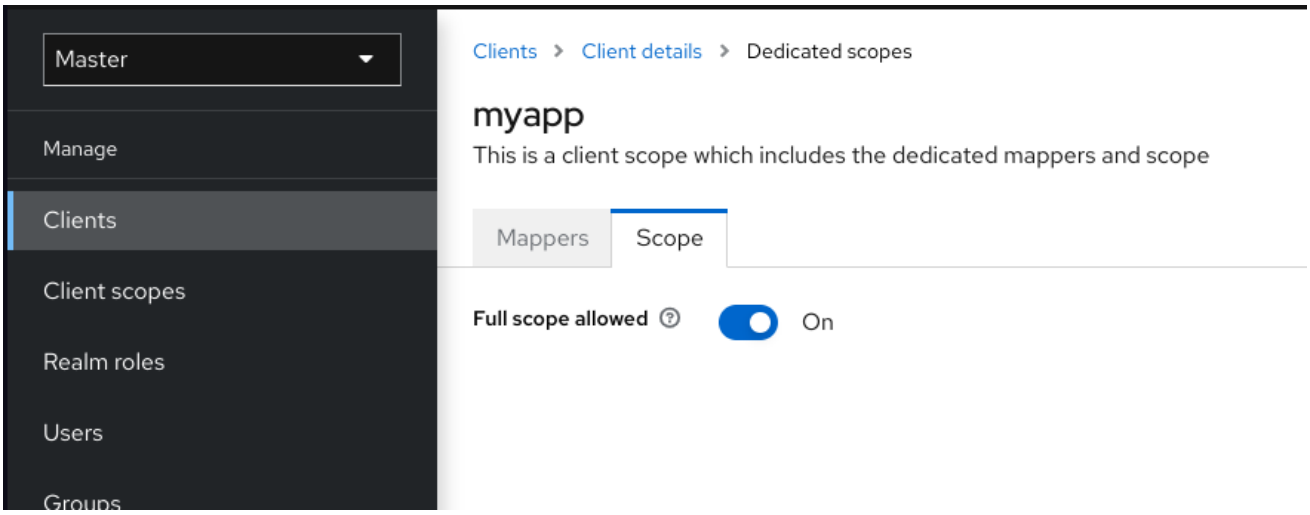
默认情况下，每个客户端获取用户的所有角色映射。您可以查看客户端的角色映射。

**流程**

1. 点菜单中的 **Clients**。
2. 单击客户端以进入详细信息。
3. 点 **Client scopes** 选项卡。
4. 点击 *此客户端的* ***Dedicated*** *范围和映射程序*所在行中的链接

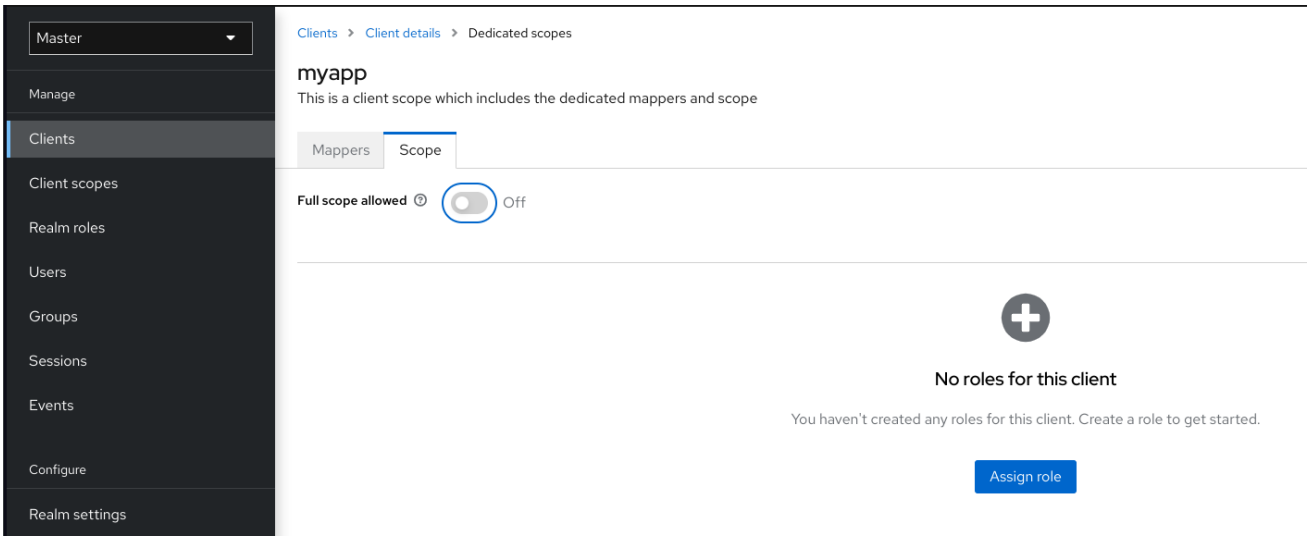
5. **点 Scope 选项卡。**

**完整范围**




**默认情况下，域中的每个声明的角色都是范围的有效角色。要更改此默认行为，请将 Full Scope Allowed 切换到 OFF，并声明每个客户端中所需的特定角色。您还可以使用 客户端 范围为一组客户端定义相同的角色范围映射。**

**部分范围**

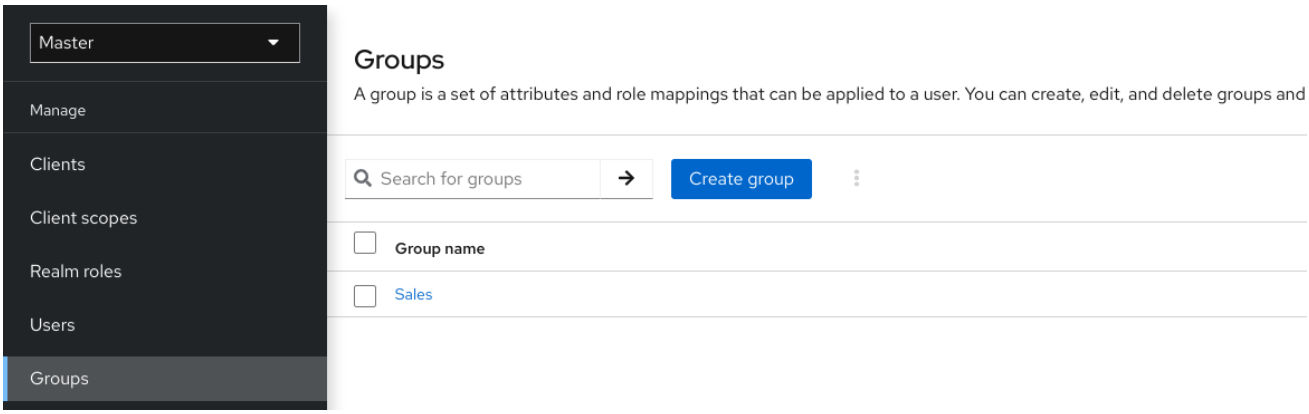



## 7.7. 组

红帽构建的 Keycloak 中的组为每个用户管理一组通用的属性和角色映射。用户可以是任意数量的组的成员，并继承分配给每个组的属性和角色映射。

要管理组，点菜单中的 Groups。

组




组是分层的。个组可以有多个子组，但一个组只能有一个父组。子组从其父级继承属性和角色映射。用户也从其父级继承属性和角色映射。

如果您有父组和子组，并且只属于子组的用户，子组中的用户将继承父组和子组的属性和角色映射。

以下示例包括顶级 销售组 和子 北美 子组。

添加组：

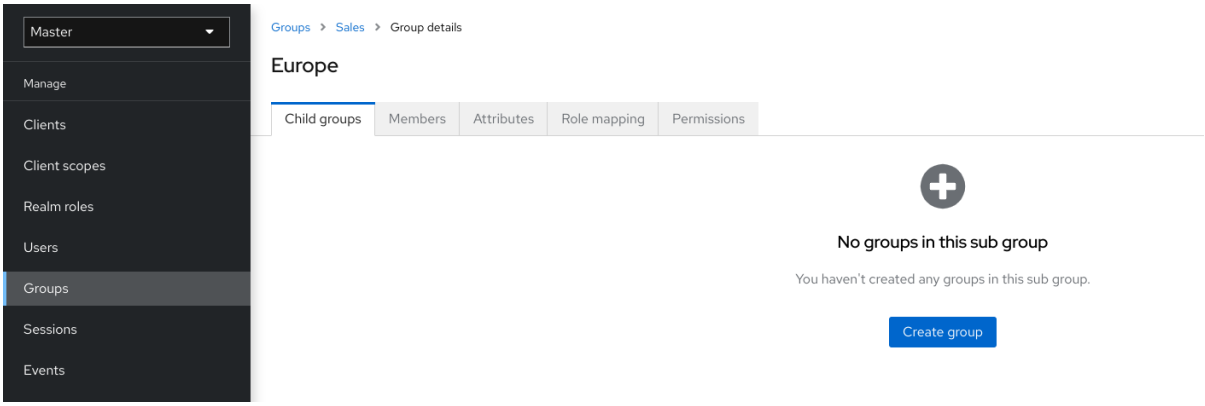
1. 点组。
2. 点 Create group。
3. 输入组名称。

4. **点 Create。**

5. **点组名称。**

   **此时会显示组管理页面。**

组




**您定义的属性和角色映射由属于组成员的组和用户继承。**

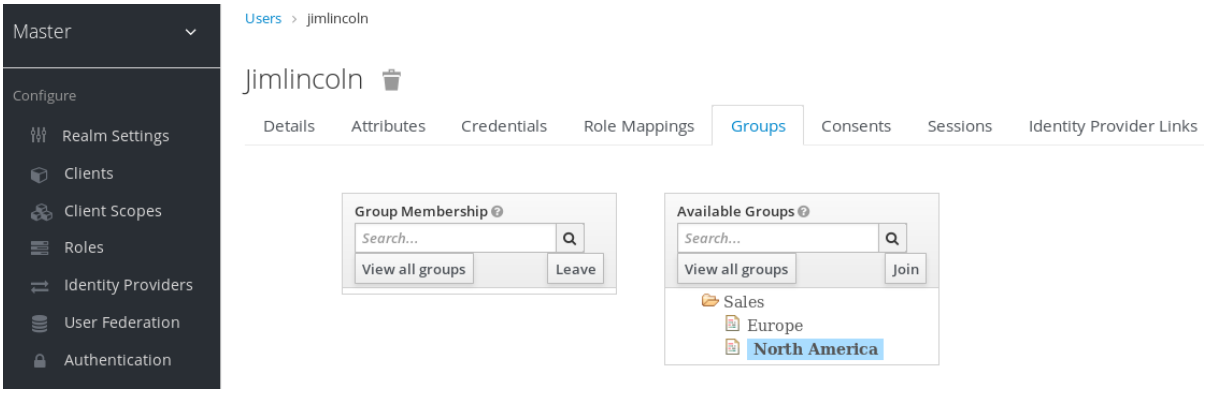
**将用户添加到组中：**

1. **点菜单中的 Users。**

2. **点您要执行角色映射的用户。如果没有显示用户，请单击 View all users。**

3. **点 Groups。**

   用户组




4. **点 Join Group。**

5. **从对话框中选择一个组。**

6. **从 Available Groups 树中选择一个组。**

7. **点 Join。**

**从用户中删除组：**

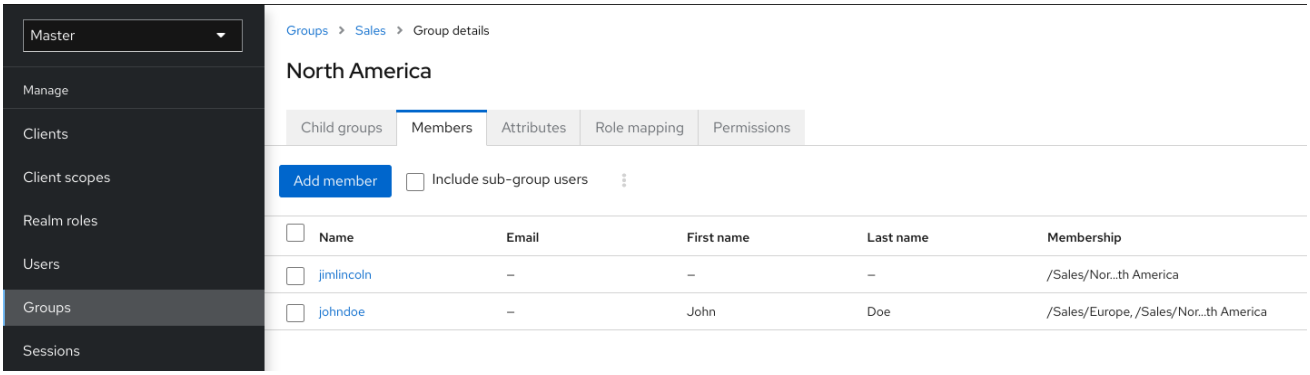
1. **点菜单中的 Users。**

2. **单击要从组中删除的用户。**

3. **点 组表行上的 Leave。**

**在本例中，用户 *jimlincoln* 位于 *北美* 组中。您可以在组的 Members 选项卡下显示 *jimlincoln*。**

组成员资格




### 7.7.1. 与角色相比的组

组和角色具有一些相似性和不同之处。在红帽构建的 Keycloak 中，组是应用角色和属性的用户集合。角色定义用户和应用程序的类型，为角色分配权限和访问控制。

复合角色 与组类似，因为它们提供相同的功能。它们之间的差别在于概念性。复合角色将权限模型应用到一组服务和应用程序。使用复合角色来管理应用程序和服务。

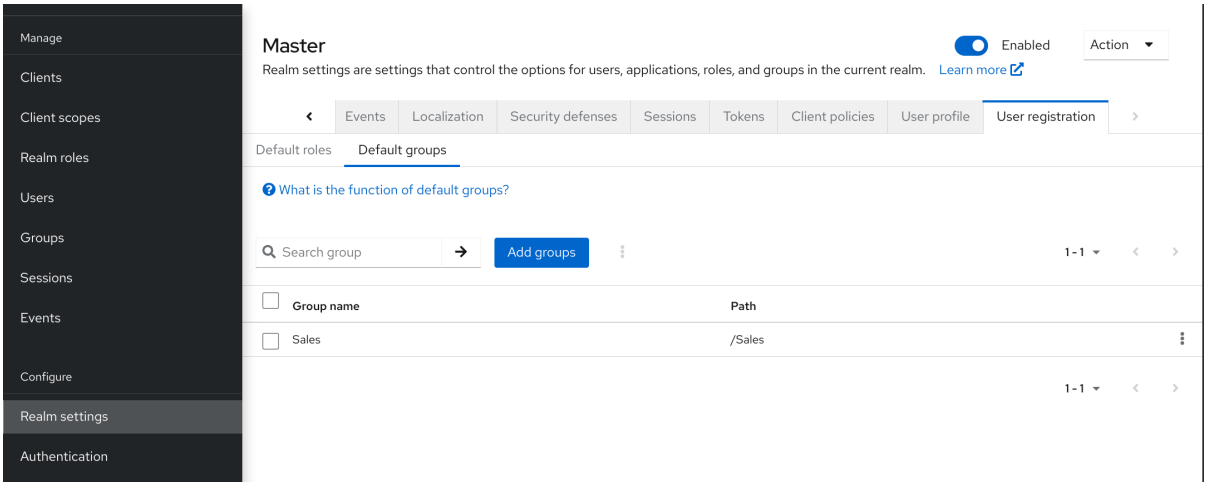
组专注于用户的集合及其在组织中的角色。使用组来管理用户。

### 7.7.2. 使用默认组

要自动将组成员资格分配给通过 Identity Brokering 导入的任何用户，您可以使用默认组。

1. 点菜单中的 **Realm settings**。
2. 点 **User registration** 选项卡。
3. 点 **Default Groups** 选项卡。

默认组




**此截屏显示一些 *默认的组* 已存在。**

# 第 8 章 配置身份验证

本章论述了几个身份验证主题。这些主题包括：

- 强制严格的密码和一次性密码(OTP)策略。
- 管理不同的凭证类型。
- 使用 Kerberos 登录。
- 禁用并启用内置凭证类型。

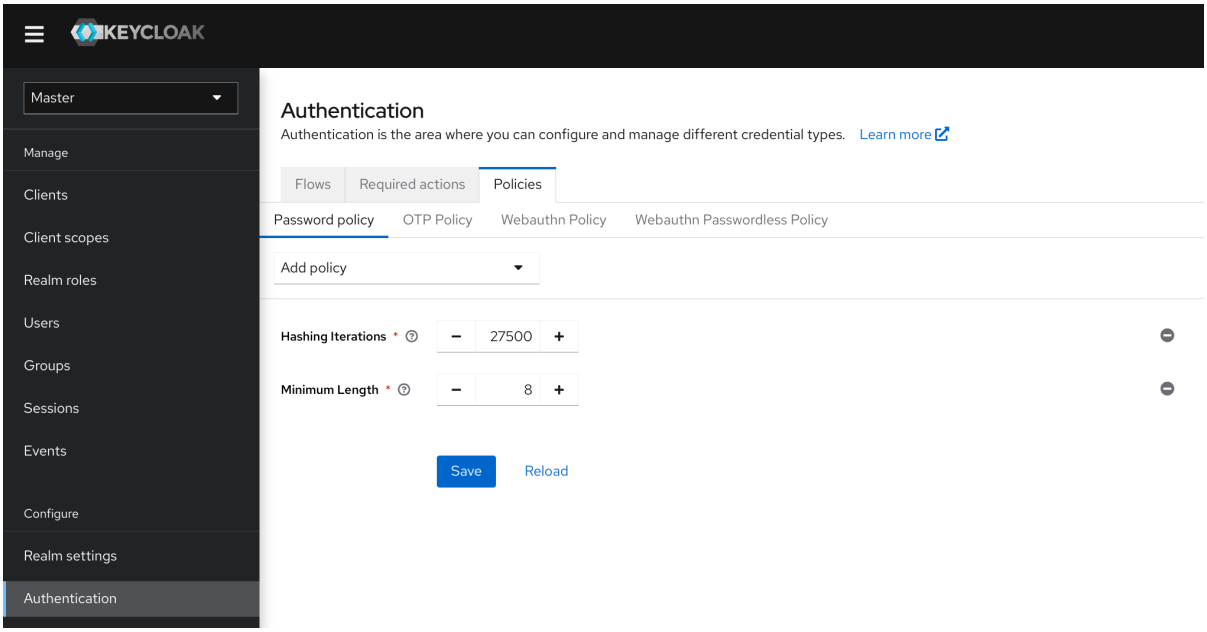
## 8.1. 密码策略

当红帽构建的 Keycloak 创建一个域时，它不会将密码策略与域关联。您可以设置简单的密码，使其长度、安全性或复杂性没有限制。在生产环境中，无法接受简单的密码。红帽构建的 Keycloak 通过管理控制台提供一组密码策略。

**流程**

1. 点菜单中的 **Authentication**。
2. 点 **Policies** 选项卡。
3. 选择要在 **Add policy** 下拉菜单中添加的策略。
4. 输入适用于所选策略的值。
5. 点击 **Save**。

密码策略




保存策略后，红帽构建的 Keycloak 会为新用户强制执行策略。

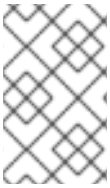
**注意**

新策略对现有用户无效。因此，请确保从域创建开始设置密码策略，或向现有用户添加"更新密码"，或者使用"Expire password"以确保用户在下次"N"天中更新其密码，实际上会调整到新的密码策略。

### 8.1.1. 密码策略类型

#### 8.1.1.1. HashAlgorithm

密码不会存储在明文中。在存储或验证之前，红帽使用标准哈希算法构建 Keycloak 哈希密码。PBKDF2 是唯一可用的内置和默认算法。有关如何添加您自己的哈希算法，请参阅 服务器开发人员指南。



**注意**

如果您更改了哈希算法，则存储中的密码哈希不会更改，直到用户登录为止。

#### 8.1.1.2. 哈希迭代

指定红帽在存储或验证前的 Keycloak 哈希密码构建的次数。如果 pbkdf2-sha512 用作哈希算法，则默认值为 210,000。如果使用'HashAlgorithm' 策略明确设置了其他哈希算法，则默认散列迭代数可能会

有所不同。例如，如果使用'pbkdf2-sha256' 算法，则默认为 600,000；如果 pbkdf2 算法（算法 pbkdf2 对应于 HMAC-SHA1，则默认为 1,300,000）。

红帽构建的 Keycloak 哈希密码，以确保可访问密码数据库的主机操作者无法通过反向工程读取密码。

注意

高哈希迭代值可能会影响性能，因为它需要更高的 CPU 电源。

#### 8.1.1.3. 数字

密码字符串中所需的数字数。

#### 8.1.1.4. 小写字符

密码字符串中所需的小写字母数。

#### 8.1.1.5. 大写字符

密码字符串中所需的大写字母数。

#### 8.1.1.6. 特殊字符

密码字符串中所需的特殊字符数。

#### 8.1.1.7. 不是 username

密码不能与用户名相同。

#### 8.1.1.8. Not email

密码不能与用户的电子邮件地址相同。

#### 8.1.1.9. 正则表达式

密码必须与一个或多个定义的 Java 正则表达式模式匹配。有关这些表达式的语法，请参阅 Java 的正则表达式文档。

#### 8.1.1.10. 过期的密码

密码有效的天数。当天数已过期时，用户必须更改其密码。

#### 8.1.1.11. 最近使用

用户不能使用密码。红帽构建的 Keycloak 存储了已用密码的历史记录。存储的旧密码数量可在红帽构建的 Keycloak 中进行配置。

#### 8.1.1.12. 密码黑名单

密码不能位于黑名单文件中。

- 黑名单文件是 UTF-8 纯文本文件，结尾 Unix 行。每行都代表列入黑名单的密码。
- 红帽构建的 Keycloak 以区分大小写的方式比较密码。黑名单中的所有密码都必须为小写。
- blacklist 文件的值必须是黑名单文件的名称，例如 100k_passwords.txt。
- 默认情况下，将针对 ${kc.home.dir}/data/password-blacklists/ 解析的文件列入黑名单。使用以下命令自定义此路径：
  - keycloak.password.blacklists.path 系统属性。
  - passwordBlacklist 策略 SPI 配置的 blacklistsPath 属性。要使用 CLI 配置黑名单文件夹，请使用 --spi-password-policy-password-blacklist-blacklists-path=/path/to/blacklistsFolder。

关于 False Positives 的备注

当前实施使用 BloomFilter 进行快速和内存效率控制检查，如给定密码是否包含在黑名单中，并可能误报。

- 默认情况下，使用 false 的正概率为 0.01%。

- 要通过 CLI 配置更改假的正概率，请使用 --spi-password-policy-password-blacklist-false-positive-probability=0.00001。

#### 8.1.1.13. 最大身份验证期限

指定用户身份验证的最长期限（以秒为单位），用户可在不重新身份验证的情况下更新密码。值 0 表示用户必须始终重新验证其当前密码，然后才能更新密码。有关此策略的详情，请参阅 AIA 部分。

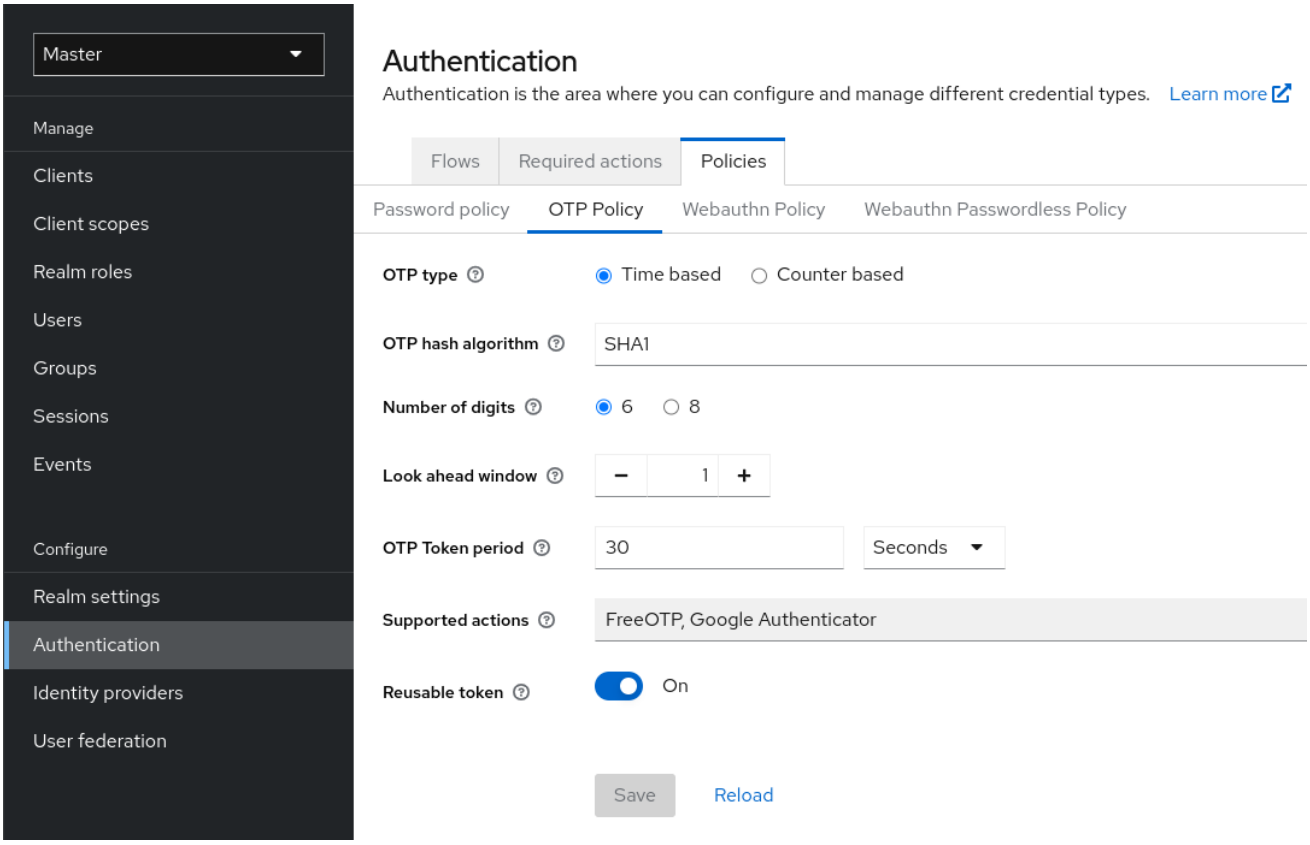
## 8.2. 一次性密码(OTP)策略

红帽构建的 Keycloak 有几个策略用于设置 FreeOTP 或 Google Authenticator 一次性密码生成器。

**流程**

1. 点菜单中的 Authentication。
2. 点 Policy 选项卡。
3. 点 OTP Policy 选项卡。

**OTP 策略**




**Red Hat build of Keycloak 根据 OTP Policy 选项卡中配置的信息，在 OTP 设置页面中生成一个 QR 代码。FreeOTP 和 Google Authenticator 在配置 OTP 时扫描 QR 代码。**

### 8.2.1. 基于时间的或基于计数器的一次性密码

**红帽为 OTP 生成器的 Keycloak 提供的算法基于时间和基于计数器的算法。**

**使用基于时间的一次性密码(TOTP)，令牌生成器会哈希当前时间和共享 secret。服务器通过将时间窗内的哈希与提交的值进行比较来验证 OTP。TOTP 在较短的时间内有效。**

**使用基于计数器的一次性密码(HOTP)，红帽构建的 Keycloak 使用共享计数器而不是当前的时间。红帽构建的 Keycloak 服务器会在每次成功 OTP 登录时递增计数器。有效 OTPs 在成功登录后改变。**

**TOTP 比 pollingP 更安全，因为 matchable OTP 在短时间内有效，而 OTP for housekeepingP 则有效，而在确定的时间方面有效。HOTP 比 TOTP 更友好，因为不存在时间限制来进入 OTP。**

**当服务器每次递增计数器时，ACMP 需要数据库更新。在这个版本中，在负载过重的过程中，身份验证服务器上的性能排空。为提高效率，TOTP 不记住所使用的密码，因此无需执行数据库更新。缺陷是可**

以在有效时间间隔中重新使用 TOTPs。

#### 8.2.2. TOTP 配置选项

##### 8.2.2.1. OTP 哈希算法

默认算法是 SHA1。其他更安全的选项包括 SHA256 和 SHA512。

##### 8.2.2.2. 数字数

OTP 的长度。短 OTP 是用户友好的，易于输入，更容易记住。OTP 比更短的 OTP 更安全。

##### 8.2.2.3. 查看窗口

服务器尝试与哈希匹配的间隔数。如果 TOTP 生成器或身份验证服务器的时钟不同步，则 Red Hat build of Keycloak 中存在这个选项。1 的默认值足够。例如，如果令牌的时间间隔是 30 秒，则默认值 1 表示它将在 90 秒窗口中接受有效令牌（时间间隔 30 秒 + 查看前 30 秒 + 查看后 30 秒）：这个值的每个递增会将有效窗口增加 60 秒（在 30 秒后 + 在 30 秒后查找）。

##### 8.2.2.4. OTP 令牌周期

服务器与哈希匹配的时间间隔（以秒为单位）。每次间隔通过时，令牌生成器都会生成一个 TOTP。

##### 8.2.2.5. 可重复使用的代码

确定 OTP 令牌是否可以在身份验证过程中重复使用，或者用户需要等待下一个令牌。默认情况下，用户无法重复使用这些令牌，管理员需要明确指定可以重复使用这些令牌。

#### 8.2.3. libpmemP 配置选项

##### 8.2.3.1. OTP 哈希算法

默认算法是 SHA1。其他更安全的选项包括 SHA256 和 SHA512。

##### 8.2.3.2. 数字数

OTP 的长度。短 OTP 是用户友好的，易于输入，更容易记住。较长的 OTP 比较短的 OTP 更安全。

#### 8.2.3.3. 查看窗口

服务器尝试与哈希匹配的前和以下间隔数。如果 TOTP 生成器或身份验证服务器的时钟不同步，则 Red Hat build of Keycloak 中存在这个选项。1 的默认值足够。当用户的计数器在服务器前面时，红帽构建的 Keycloak 中存在这个选项。

#### 8.2.3.4. 初始计数器

初始计数器的值。

## 8.3. 身份验证流

*身份验证流程* 是在登录、注册和其他红帽构建的 Keycloak 工作流期间验证、屏幕和操作的容器。

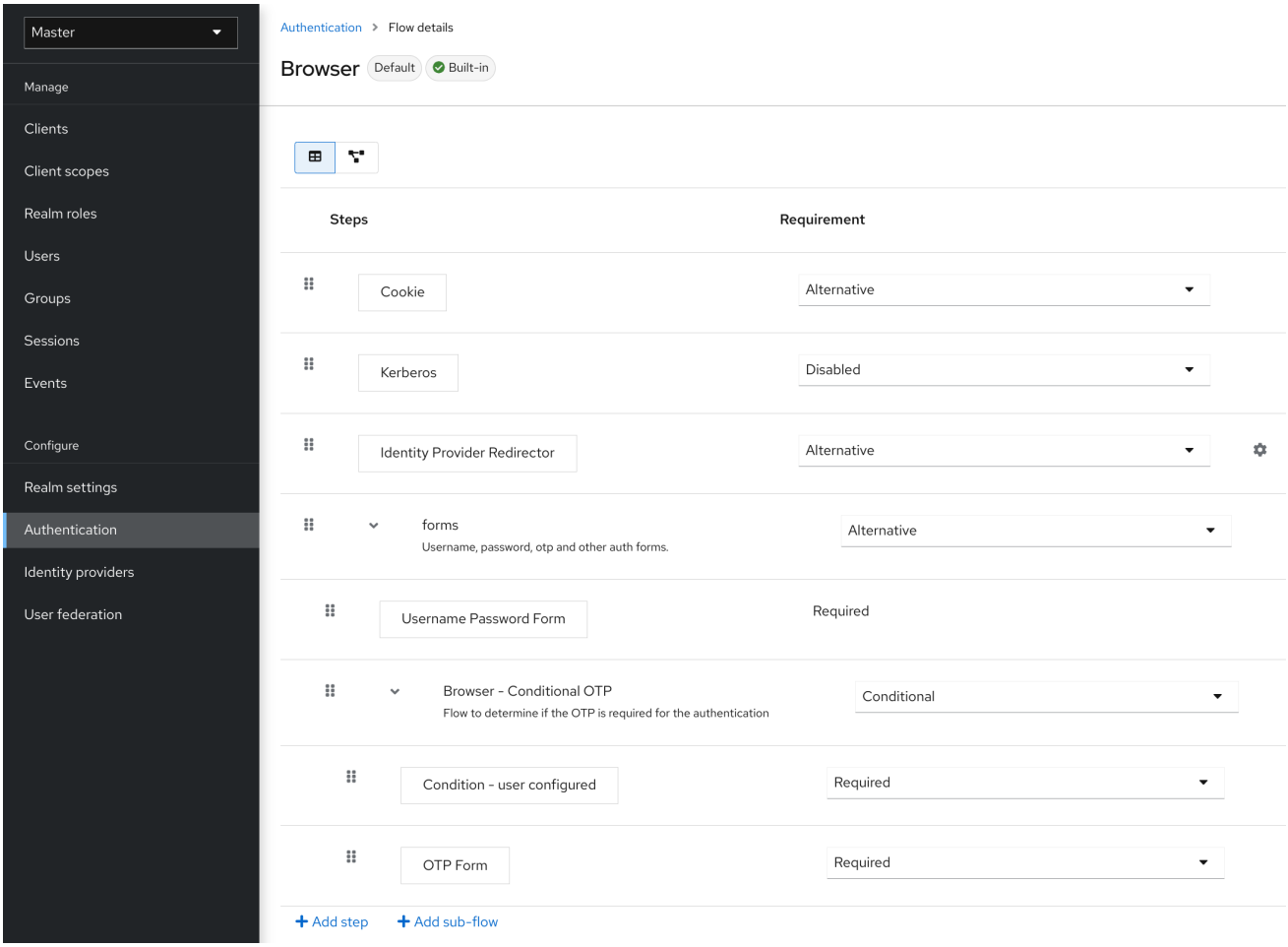
### 8.3.1. 内置流

红帽构建的 Keycloak 有几个内置流。您无法修改这些流，但您可以更改流的要求以满足您的需要。

**流程**

1. 点菜单中的 **Authentication**。
2. 单击列表中的 ***Browser*** 项，以查看详细信息。

浏览器流




#### 8.3.1.1. Auth 类型

**身份验证的名称或要执行的操作。如果身份验证缩进，则它在子流中。它可能或不执行，具体取决于其父级的行为。**

1. **cookie**

   **当用户成功登录时，红帽构建的 Keycloak 会设置一个会话 Cookie。如果已经设置了 Cookie，这个验证类型会成功。由于 Cookie 供应商返回成功，并且在这个流程级别上执行每个执行都是，因此红帽构建的 Keycloak 不会执行任何其他执行。*这会导致成功登录。***

2. ***Kerberos***

   *默认情况下，此验证器被禁用，并在浏览器流中跳过。*

3. *身份提供程序* ***Redirector***

*此操作通过 **Actions > Config** 链接进行配置。它重定向到另一个用于 身份代理的 **IdP**。*

4. *表单*

*由于此子流标记为 替代方案，因此如果 **Cookie** 身份验证类型通过，则不会执行它。此子流包含需要执行的额外验证类型。红帽构建的 **Keycloak** 会加载这个子流的执行并处理它们。*

*第一次执行是 **Username Password Form**，它是一个呈现用户名和密码页面的身份验证类型。它标记为 必需的，因此用户必须输入有效的用户名和密码。*

*第二个执行是 **Browser - Conditional OTP** 子流。这个子流是 条件的，并根据 **Condition - User Configured** 执行的结果执行。如果结果为 **true**，红帽构建的 **Keycloak** 会加载这个子流的执行并处理它们。*

*下一个执行是 **Condition - User Configured** 身份验证。此身份验证检查红帽构建的 **Keycloak** 是否在用户流中配置了其他执行。**Browser - Conditional OTP** 子流仅在用户配置了 **OTP** 凭证时才执行。*

*最后的执行是 **OTP Form**。红帽构建的 **Keycloak** 将这个执行标记为 需要，但只有用户在因为 条件子流中的设置而设置了 **OTP** 凭证时才运行。如果没有，用户不会看到 **OTP** 表单。*

#### *8.3.1.2. 要求*

*控制执行某个操作的一组单选按钮。*

##### *8.3.1.2.1. 必需*

*流中的所有 必需 元素都必须成功执行。如果一个必需的元素失败，则流会终止。*

##### *8.3.1.2.2. 替代方案*

*只有单个元素必须成功执行，才能评估流成功。因为 **Required** 流元素足以将流标记为成功，所以包含 **Required** 流元素的 **Alternative** 流元素将无法执行。*

##### *8.3.1.2.3. Disabled*

元素不计算将流标记为成功。

#### 8.3.1.2.4. 条件

这个要求类型只在子流中设置。

- 条件 子流包含执行。这些执行必须评估逻辑语句。
- 如果所有执行都评估为 **true**，则 **Conditional** 子流将充当 **Required**。
- 如果有任何执行评估为 **false**，则 **Conditional** 子流将充当 **Disabled**。
- 如果没有设置执行，**Conditional** 子流充当 **Disabled**。
- 如果流包含执行，且流没有设置为 **Conditional**，则红帽构建的 **Keycloak** 不会评估执行，执行被视为功能 禁用。

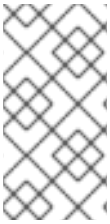
### 8.3.2. 创建流

在设计流时，会应用重要的功能和安全注意事项。

要创建流，请执行以下操作：

流程

1. 点菜单中的 **Authentication**。
2. 点 **Create flow**。



*注意*

*您可以复制并修改现有流。单击* ***"Action list"*** *（行末尾的三个点），单击* ***Duplicate,*** *然后为新流输入一个名称。*

*在创建新流时，您必须首先使用以下选项创建一个顶层流：*

***Name***

*流的名称。*

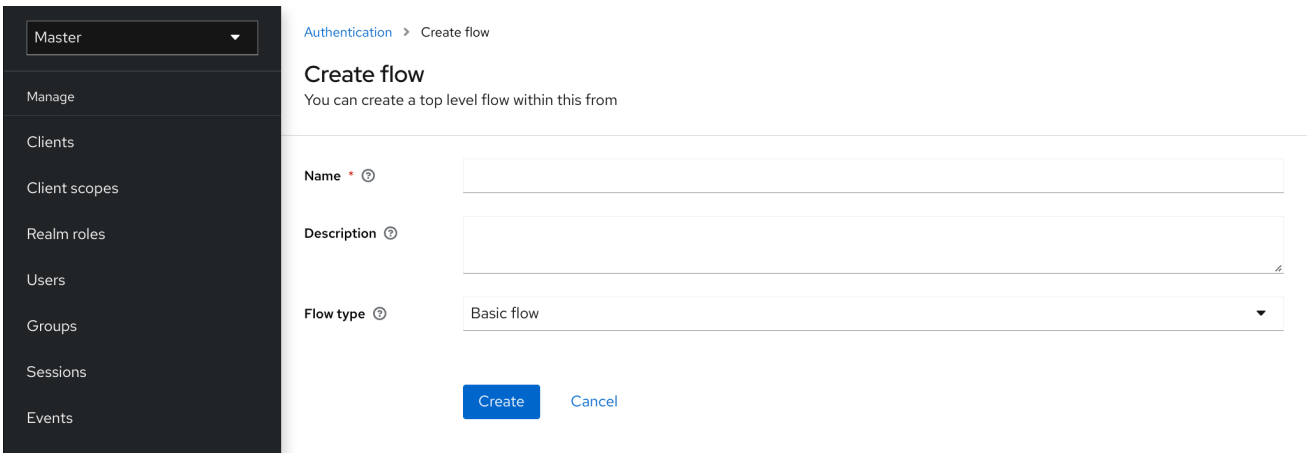
*描述*

*您可以设置为流的描述。*

*顶级流类型*

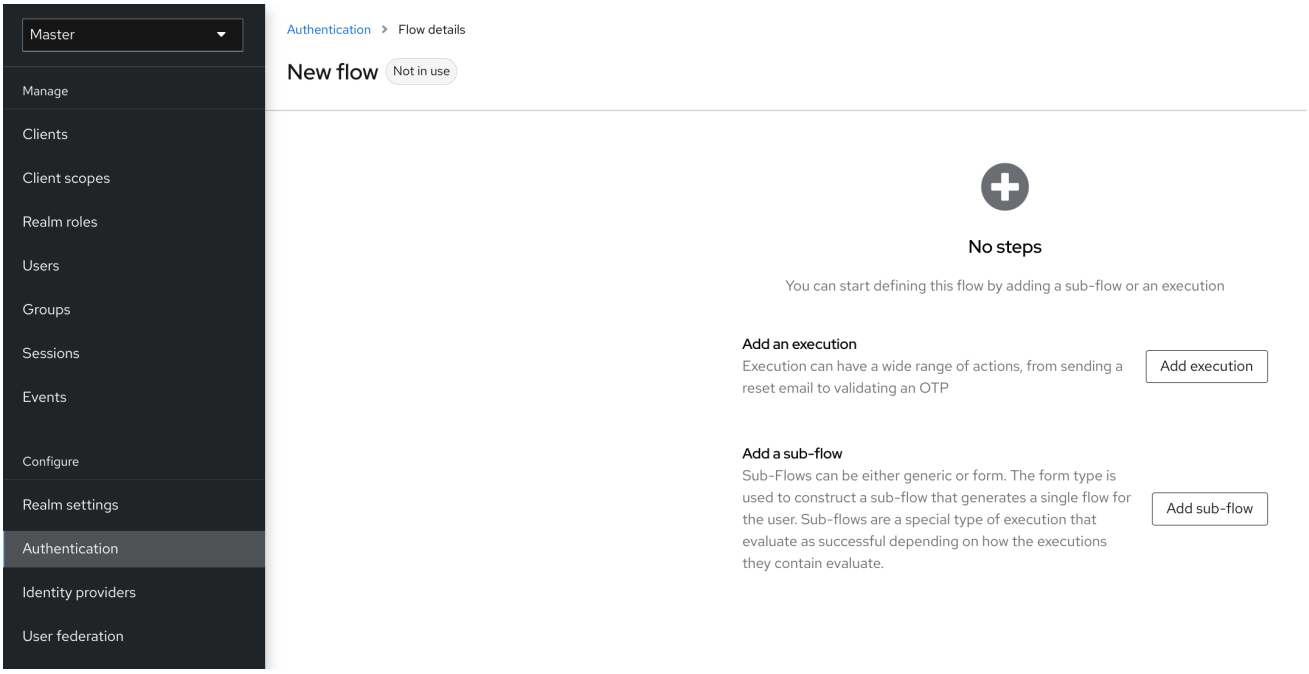
*流的类型。类型 客户端仅用于客户端 （应用程序）的身份验证。对于所有其他情况，请选择 基本。*

*创建顶层流*




*当红帽构建的* ***Keycloak*** *创建流时，* ***Red Hat build of Keycloak*** *会显示* ***Add step,*** *和* ***Add sub-flow*** *按钮。*

*一个空新流*

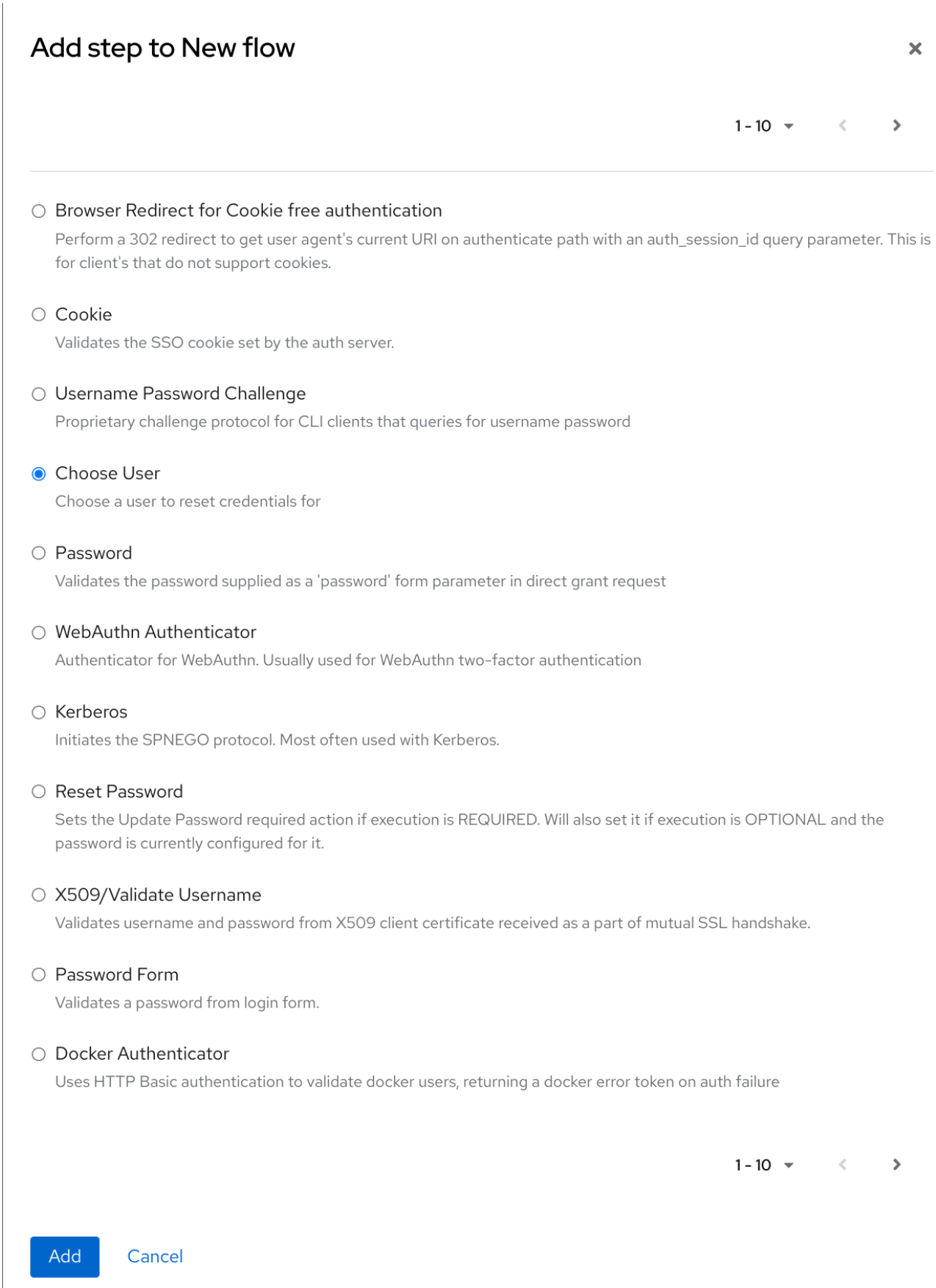



*三个因素决定了流和子流的行为。*

- *流和子流的结构。*
- *流中的执行*
- *子流和执行中设置的要求。*

*执行具有各种操作，从发送重置电子邮件以验证 **OTP**。使用 **Add step** 按钮添加执行。*

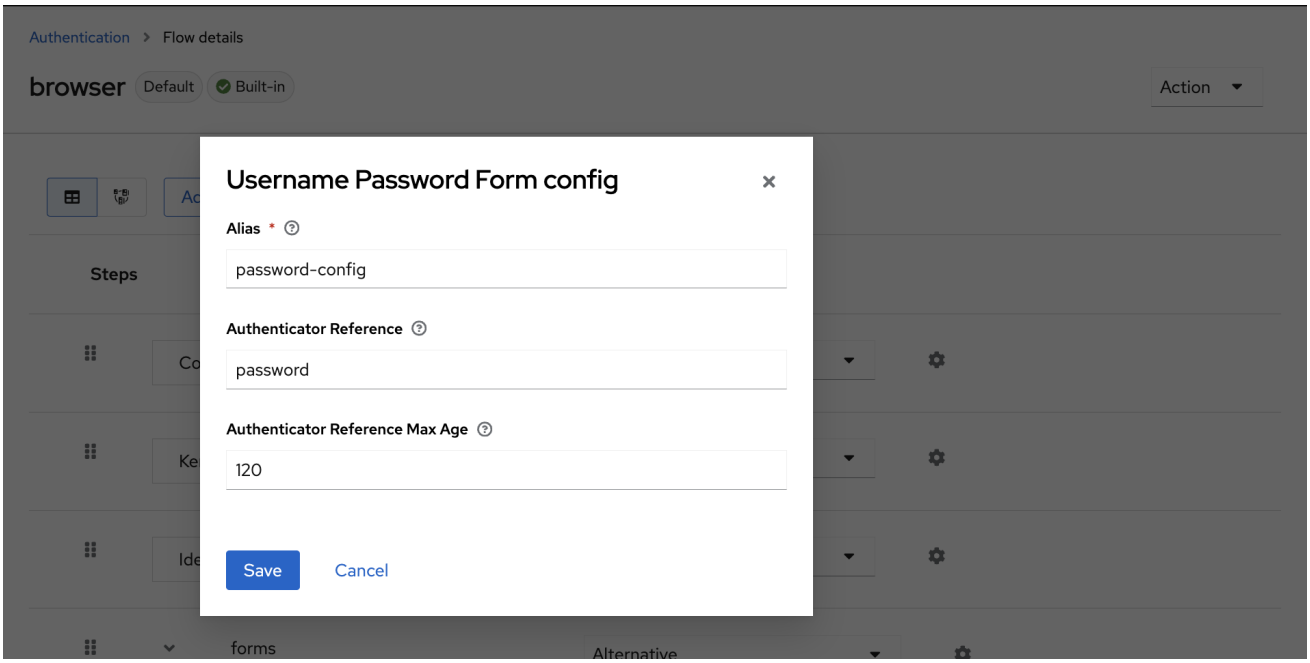
*添加身份验证执行*




*身份验证执行可以选择性地配置引用值。这可供 身份验证方法参考(**AMR**) 协议映射器用于填充 **OIDC** 访问和 **ID** 令牌（有关 **AMR** 声明的更多信息，请参阅 RFC-8176）。当为客户端配置了 身份验证方法引*

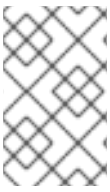
*用(AMR) 协议映射器时，它将使用用户在身份验证流中成功完成的任何验证器的参考值填充 **amr** 声明。*

*添加验证器引用值*




*存在两种类型的执行，自动执行 和交互式执行 。**Automatic executions** 与 **Cookie** 执行类似，并将在流中自动执行操作。交互式执行将暂停流以获取输入。执行操作成功将其状态设置为 **success**。要完成流，至少需要一个 成功状态 的执行。*

*您可以使用 **Add sub-flow** 按钮将子流添加到顶层流中。**Add sub-flow** 按钮显示 **Create Execution Flow** 页面。此页面与 **Create Top Level Form** 页面类似。区别在于 **Flow Type** 可以是 **basic** （默认）或 表单。表单 类型构建了一个子流，它为用户生成表单，类似于内置的 注册流程。子流成功取决于它们的执行评估，包括其包含子流。有关子流的工作方式的信息，请参阅执行要求部分。**???***



*注意*

*添加执行后，检查要求是否有正确的值。*

*流中的所有元素在元素旁边都有一个 **Delete** 选项。有些执行有一个 **HBAC** 菜单项（齿轮图标）来配置执行。也可以使用 **Add** 步骤和 **Add sub-flow** 链接将执行和子流 添加到子流中。*

*由于执行顺序很重要，因此您可以通过拖动名称来移动执行和子流。*

*警告*

*在配置身份验证流时，请确保正确测试您的配置，以确认设置中没有安全漏洞。我们建议您测试各种基本情况。例如，在身份验证前从用户帐户中删除各种凭证时，请考虑测试用户的身份验证行为。*

*例如，当 **2nd-factor authenticators**（如 **OTP Form** 或 **WebAuthn Authenticator**）被配置为 **REQUIRED**，且用户没有特定类型的凭证，用户可以在身份验证本身中设置特定的凭证。这种情况意味着，在身份验证过程中，用户不会与此凭证进行身份验证。因此，对于浏览器身份验证，请确保使用一些第一因素凭证（如 **Password** 或 **WebAuthn Passwordless Authenticator**）配置身份验证流。*

### *8.3.3. 创建无密码浏览器登录流*

*为了说明流程的创建，本节描述了创建高级浏览器登录流程。此流的目的是允许用户使用 **WebAuthn** 的无密码方式登录，或使用密码和 **OTP** 进行双因素身份验证。*

*流程*

1. *点菜单中的 **Authentication**。*
2. *点 **Flows** 选项卡。*
3. *点 **Create flow**。*
4. *输入 **Browser Password-less** 作为名称。*
5. *点 **Create**。*
6. *点 **Add execution**。*

7. *从列表中选择 **Cookie**。*

8. *点击 **Add**。*

9. *为 **Cookie** 验证类型选择 **Alternative**，以将其要求设置为替代。*

10. *点 **Add step**。*

11. *从列表中选择 **Kerberos**。*

12. *点击 **Add**。*

13. *点 **Add step**。*

14. *从列表中选择 **Identity Provider Redirector**。*

15. *点击 **Add**。*

16. *为 **Identity Provider Redirector** 身份验证类型选择 **Alternative** 来将其要求设置为替代。*

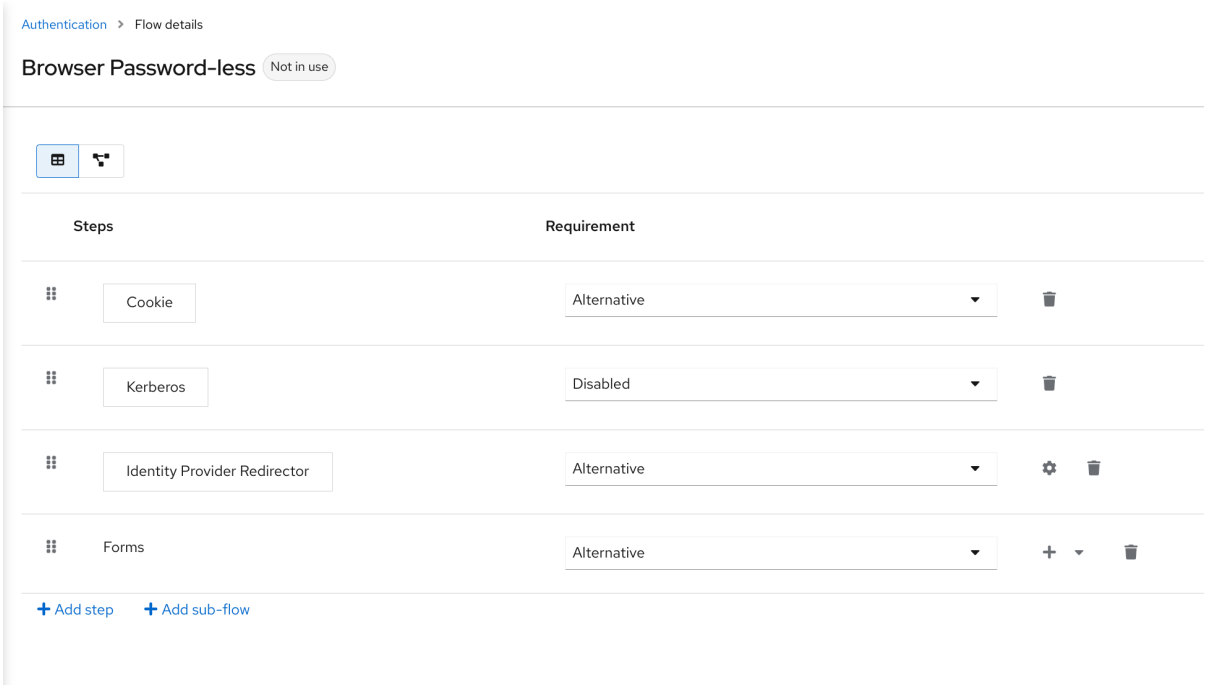
17. *点 **Add sub-flow**。*

18. *输入 **Forms** 作为名称。*

19. *点击 **Add**。*

20. *为 **Forms** 验证类型选择 **Alternative**，以将其要求设置为替代。*

*浏览器流的常见部分*




21. *点 **Forms** 执行的 **+** 菜单。*

22. *选择 **Add step**。*

23. *从列表中选择 **Username Form**。*

24. *点击 **Add**。*

*在这个阶段，表单需要一个用户名，但没有密码。我们必须启用密码身份验证以避免安全风险。*

1. *点 **Forms** 子流的 **+** 菜单。*

2. *点 **Add sub-flow**。*

3. *输入 **Authentication** 作为名称。*

4. 点击 **Add**。

5. 选择 **Authentication authentication type** 是必需的，以设置其要求。

6. 点 **Authentication** 子流的 **+** 菜单。

7. 点 **Add step**。

8. 从列表中选择 **WebAuthn Passwordless Authenticator**。

9. 点击 **Add**。

10. 为 **Webauthn Passwordless Authenticator** 身份验证类型选择替代方案，以将其要求设置为替代。

11. 点 **Authentication** 子流的 **+** 菜单。

12. 点 **Add sub-flow**。

13. 输入 **Password with OTP** 作为名称。

14. 点击 **Add**。

15. 选择 **Alternative for the Password with OTP authentication type**，将其要求设置为替代。

16. 点 带有 **OTP** 子流的 **Password** 的 **+** 菜单。

17. 点 **Add step**。

18. *从列表中选择 **Password Form**。*

19. *点击 **Add**。*

20. *选择 **Password Form authentication type** 是必需的，以将其要求设置为 **required**。*

21. *点 带有 **OTP** 子流的 **Password** 的 **+** 菜单。*

22. *点 **Add step**。*

23. *从列表中选择 **OTP Form**。*

24. *点击 **Add**。*

25. *点 **OTP Form** 身份验证类型的 **Required**，将其要求设置为 **required**。*

*最后，更改绑定。*

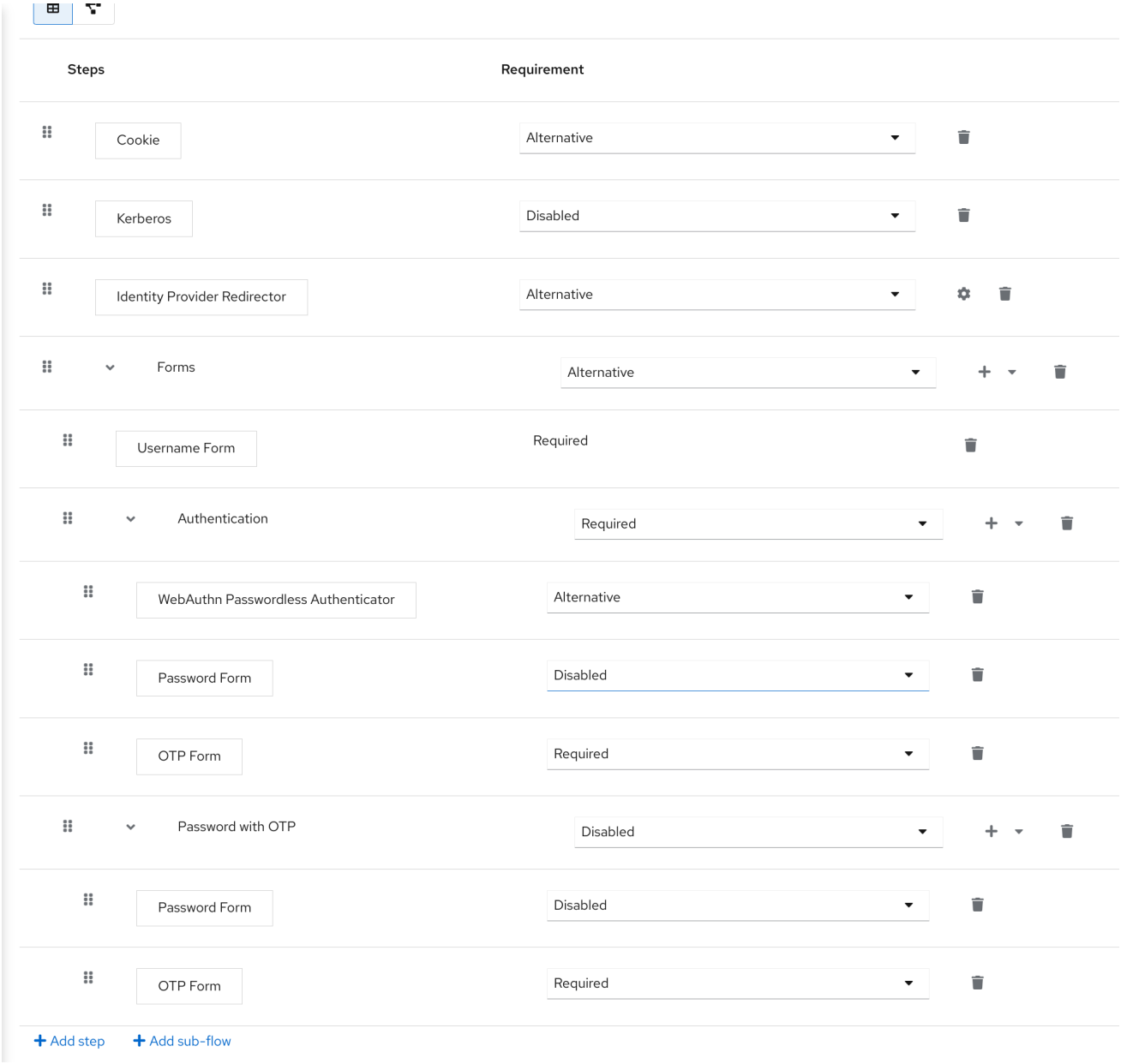
1. *单击屏幕顶部的 **Action** 菜单。*

2. *从菜单中选择 绑定流。*

3. *点 **Browser Flow** 下拉列表。*

4. *点击 **Save**。*

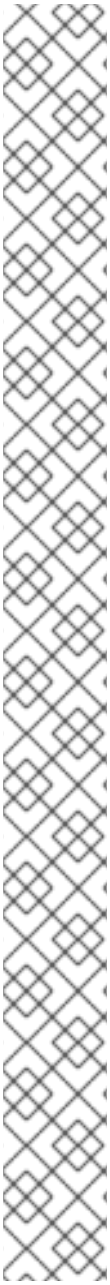
*无密码浏览器登录*




*输入用户名后，流可以正常工作：*

*如果用户记录了 **WebAuthn** 免密码凭据，他们可以使用这些凭证直接登录。这是免密码登录。用户也可以选择 **Password with OTP**，因为 **WebAuthn Passwordless** 执行和 **Password with OTP** 流被设置为 **Alternative**。如果它们被设置为 **Required**，用户必须输入 **WebAuthn**、**password** 和 **OTP**。*

*如果用户使用 **WebAuthn** 免密码身份验证 选择 **Try** 另一个 链接，用户可以在 **Password** 和 **Passkey (WebAuthn password)**之间进行选择。在选择密码时，用户需要继续并使用分配的 **OTP** 登录。如果用户没有 **WebAuthn** 凭据，用户必须输入密码，然后是 **OTP**。如果用户没有 **OTP** 凭证，则会要求记录一个凭证。*



注意

因为 **WebAuthn Passwordless** 执行被设置为 **Alternative** 而不是 **Required**，因此这个流永远不会要求用户注册 **WebAuthn** 凭证。要使用户具有 **Webauthn** 凭证，管理员必须为用户添加必要的操作。通过以下方法进行此操作：

1. 在域中启用 **Webauthn Register Passwordless required** 操作（请参阅 **WebAuthn** 文档）。
2. 使用用户凭据管理菜单的 **Credential Reset** 部分设置必要的操作。

创建高级流（如），这可能会产生副作用。例如，如果您启用了为用户重置密码的功能，则可以从密码表单访问。在默认的 **Reset Credentials** 流中，用户必须输入其用户名。由于用户已在 浏览器无密码 流中输入了用户名，因此对于用户体验，这个操作对于红帽构建的 **Keycloak** 和子优化是不需要的。要更正这个问题，您可以：

- 复制 重置凭据 流。将其名称设置为 无密码的重置凭据，例如：
- 单击 **Choose user** 步骤的 **Delete (trash icon)**。
- 在 **Action** 菜单中，选择 **Bind flow**，然后从下拉菜单中选择 **Reset credentials flow**，然后单击 **Save**

### 8.3.4. 使用步骤机制创建浏览器登录流程

本节论述了如何使用步骤机制创建高级浏览器登录流。步骤身份验证的目的是允许根据用户的特定身份验证级别访问客户端或资源。

流程

1. 点菜单中的 **Authentication**。
2. 点 **Flows** 选项卡。

3. 点 **Create flow**。

4. 输入 **Browser Incl Step up Mechanism** 作为名称。

5. 点击 **Save**。

6. 点 **Add execution**。

7. 从列表中选择 **Cookie**。

8. 点击 **Add**。

9. 为 **Cookie** 验证类型选择 **Alternative**，以将其要求设置为替代。

10. 点 **Add sub-flow**。

11. 输入 **Auth Flow** 作为名称。

12. 点击 **Add**。

13. 点 **Alternative**，**Auth Flow** 验证类型将它的 **requirement** 设置为 **alternative**。

现在，您可以为第一个身份验证级别配置流。

1. 点 **Auth Flow** 的 **+** 菜单。

2. 点 **Add sub-flow**。

3. *输入 **1st Condition Flow** 作为名称。*

4. *点击 **Add**。*

5. *点 **1st Condition Flow** 验证类型的 **Conditional** 将要求设置为条件。*

6. *点 **1st Condition Flow** 的 **+** 菜单。*

7. *单击 **Add condition**。*

8. *从列表中选择 **Conditional - Level Of Authentication**。*

9. *点击 **Add**。*

10. *点 **Conditional - Level Of Authentication authentication type** 来将其要求设置为 **required**。*

11. *点 **iwl iwl (gear** 图标)。*

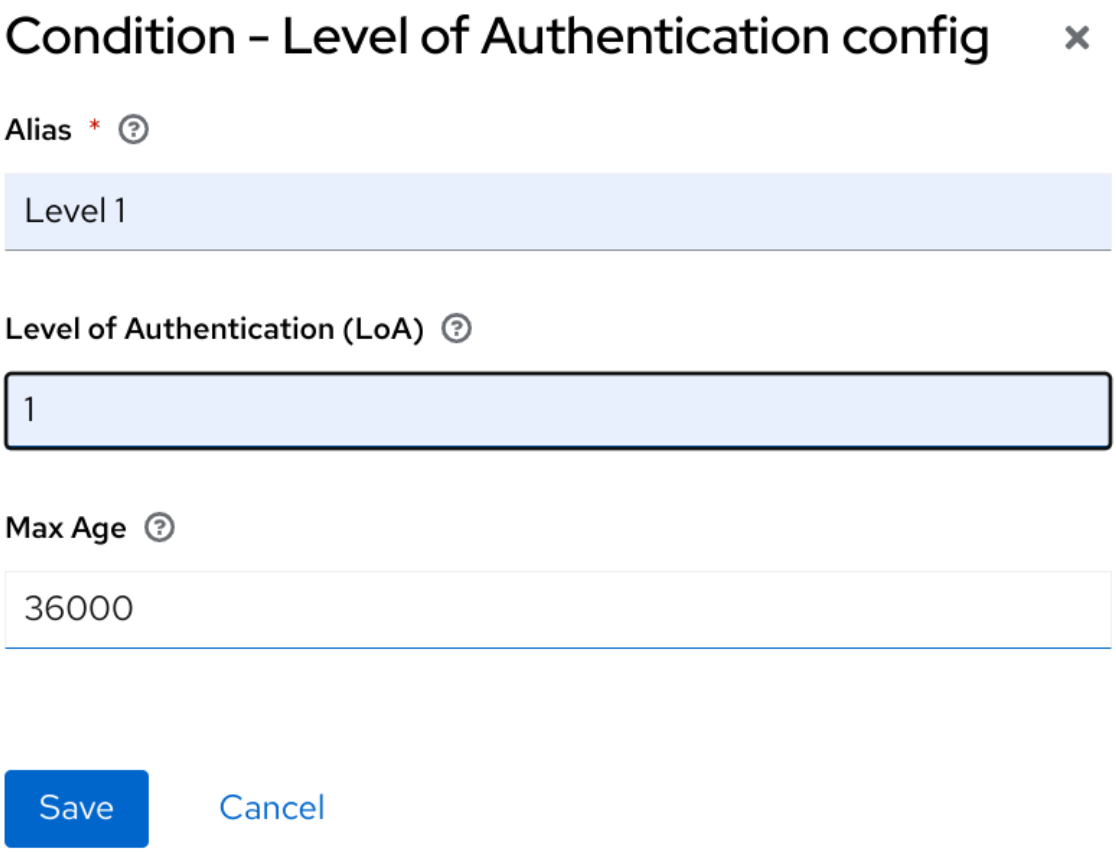
12. *输入 **Level 1** 作为别名。*

13. *在验证级别**(LoA)**输入 **1**。*

14. *将 **Max Age** 设置为 **36000**。这个值以秒为单位，相当于 **10** 小时，这是在 **realm** 中设置的默认 **SSO Session Max** 超时。因此，当用户使用此级别进行身份验证时，后续的 **SSO** 登录可以重新使用这个级别，用户不需要在用户会话结束前与这个级别进行身份验证，默认为 **10** 小时。*

15. *点 **Save***

*为第一个身份验证级别配置条件*




16. *点 **1st Condition Flow** 的 **+** 菜单。*

17. *点 **Add step**。*

18. *从列表中选择 **Username Password Form**。*

19. *点击 **Add**。*

*现在，您可以为第二个身份验证级别配置流。*

1. *点 **Auth Flow** 的 **+** 菜单。*

2. *点 **Add sub-flow**。*

3. *输入 **2nd Condition Flow** 作为一个别名。*

4. *点击 **Add**。*

5. *点 **2nd Condition Flow** 验证类型的 **Conditional** 将要求设置为条件。*

6. *点 **2nd Condition Flow** 的 **+** 菜单。*

7. *单击 **Add condition**。*

8. *从项目列表中选择 **Conditional - Level Of Authentication**。*

9. *点击 **Add**。*

10. *点 **Conditional - Level Of Authentication authentication type** 来将其要求设置为 **required**。*

11. *点 **iwl iwl (gear** 图标)。*

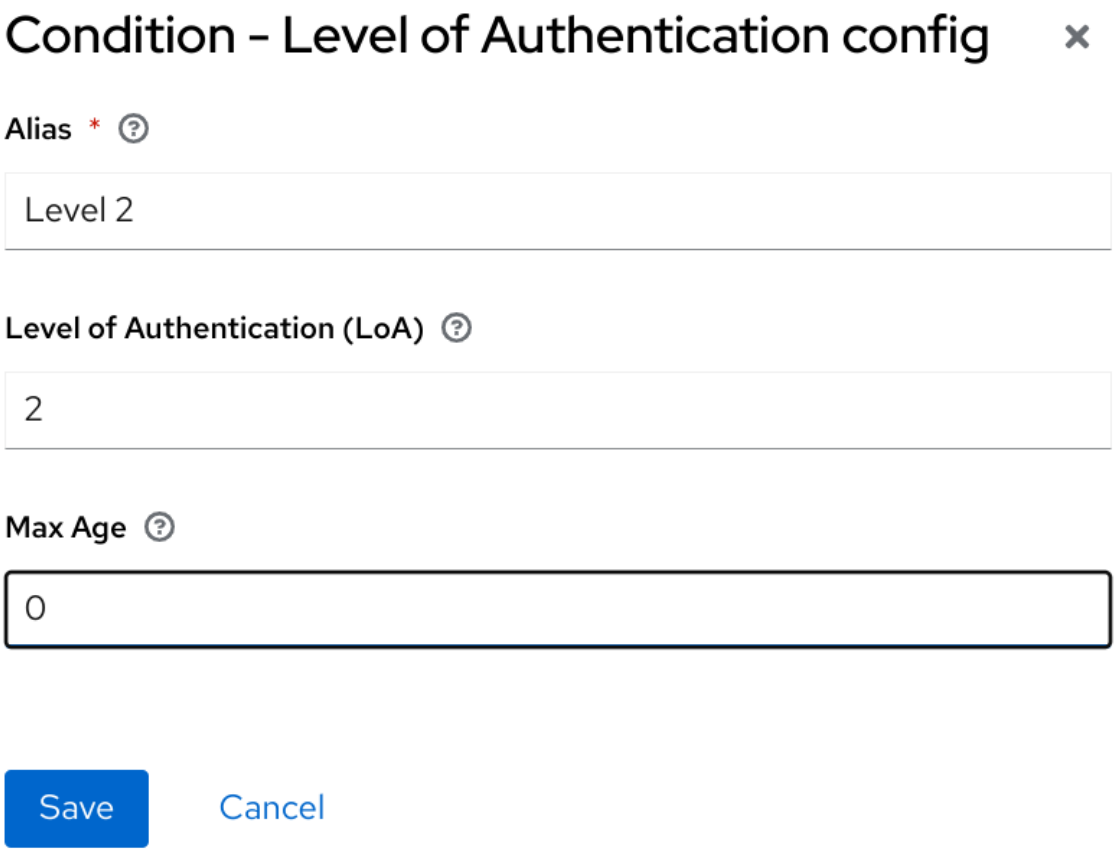
12. *输入 **Level 2** 作为别名。*

13. *在验证级别**(LoA)**输入 **2**。*

14. *将 **Max Age** 设置为 **0**。因此，当用户进行身份验证时，这个级别仅对当前身份验证有效，但不对后续任何 **SSO** 身份验证有效。因此，在请求此级别时，用户始终需要再次使用此级别进行身份验证。*

15. 点 **Save**

为第二个身份验证级别配置条件




16. 点 **2nd Condition Flow** 的 **+** 菜单。

17. 点 **Add step**。

18. 从列表中选择 **OTP Form**。

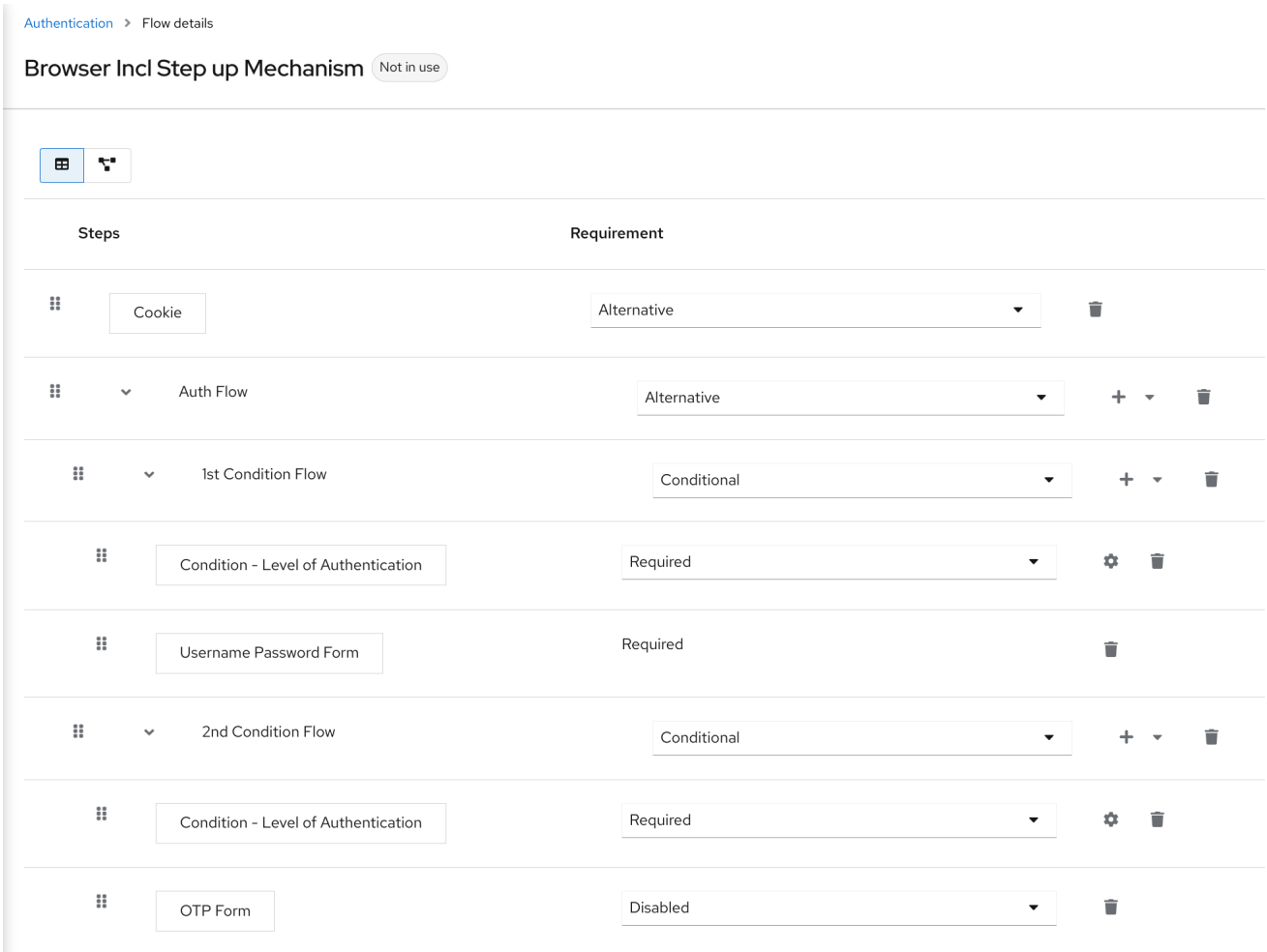
19. 点击 **Add**。

20. 点 **OTP Form** 身份验证类型的 **Required**，将其要求设置为 **required**。

*最后，更改绑定。*

1. *单击屏幕顶部的 **Action** 菜单。*
2. *从列表中选择 **Bind flow**。*
3. *从下拉菜单中选择 浏览器流。*
4. *点击 **Save**。*

*使用 **step-up** 机制的浏览器登录*




*请求特定的身份验证级别*

要使用 步骤机制，您可以在身份验证请求中指定请求的身份验证**(LoA)**级别。**claim** 参数用于此目的：

```
https://{DOMAIN}/realms/{REALMNAME}/protocol/openid-connect/auth?client_id={CLIENT-ID}&redirect_uri={REDIRECT-URI}&scope=openid&response_type=code&response_mode=query&nonce=exg16fxdjcu&claims=%7B%22id_token%22%3A%7B%22acr%22%3A%7B%22essential%22%3Atrue%2C%22values%22%3A%5B%22gold%22%5D%7D%7D%7D
```

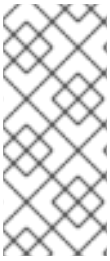
**claims** 参数以 **JSON** 指定：

```
claims= {
        "id_token": {
            "acr": {
                "essential": true,
                "values": ["gold"]
            }
        }
    }
```

红帽构建的 **Keycloak javascript** 适配器支持轻松构建此 **JSON** 并在登录请求中发送它。如需了解更多详细信息，请参阅 **Javascript** 适配器文档。

您还可以使用 **simpler** 参数 **acr_values** 而不是 **claims** 参数来请求特定级别，作为非**essential**。这在 **OIDC** 规格中提到。

您还可以为特定客户端配置默认级别，当参数 **acr_values** 或带有 **acr** 声明 的参数声明不存在时使用。如需了解更多详细信息，请参阅 客户端 **ACR** 配置。



注意

要请求 **acr_values** 作为文本（如 金级）而不是数字值，您可以在 **ACR** 和 **LoA** 之间配置映射。可以在域级别（推荐）或客户端级别进行配置。有关配置，请参阅 **ACR** 到 **LoA** 映射。

如需了解更多详细信息，请参阅官方 **OIDC** 规格。

流逻辑

上述配置的身份验证流程的逻辑如下：
如果客户端需要高的身份验证级别，即身份验证 **2** 级别**(LoA 2)**，用户必须执行完整的双因素身份验证：**Username/Password + OTP**。但是，如果用户已在红帽构建的 **Keycloak** 中已有会话，该会话已使用用户名和密码**(LoA 1)**登录，则用户只要求提供第二个身份验证因素**(OTP)**。

该条件中的选项 **Max Age** 决定后续身份验证级别有效的时长（以秒为单位）。此设置有助于确定用户是否在后续身份验证中再次显示身份验证因素。如果 声明 或 **cr_values** 参数请求了特定的级别 **X**，并且已使用级别 **X** 进行身份验证，但它已过期（例如，**max age** 被配置为 **300**，在 **310** 秒之前验证用户），则会要求用户再次与特定级别重新进行身份验证。但是，如果级别尚未过期，该用户将自动被视为使用该级别进行身份验证。

使用 **Max Age** 的值为 **0** 表示，该特定级别只针对这个单一身份验证有效。因此，要求该级别的每个重新身份验证都需要使用该级别再次进行身份验证。这对需要应用中安全性（如发送付款）且始终要求使用特定级别进行身份验证的操作很有用。

警告

请注意，当通过用户浏览器将登录请求从客户端发送到 **Keycloak** 的红帽构建时，**URL** 中的用户可能会更改 声明 或 **cr_ value** 等参数。如果客户端使用 **PAR**（推送的授权请求）、请求对象或其他阻止用户在 **URL** 中重写参数的机制，则可以缓解这种情况。因此，在身份验证后，建议客户端检查 **ID Token** 以仔细检查令牌中的 **acr** 是否与预期的级别对应。

如果参数没有请求显式级别，则红帽构建的 **Keycloak** 将需要使用身份验证流中找到的第一个 **LoA** 条件进行身份验证，如上例中的 **Username/Password**。当用户已使用该级别进行身份验证并且该级别已过期时，用户不需要重新进行身份验证，但令牌中的 **cr** 将具有值 **0**。结果被视为仅基于 长期浏览器 **cookie** 进行身份验证，如 **OIDC Core 1.0** 规格的第 **2** 节所述。

注意

当管理员指定了多个流时，可能会出现冲突的情况，为每个流设置不同的 **LoA** 级别，并将流分配给不同的客户端。但是，该规则始终相同：如果用户有一定级别，则它只需要具有该级别来连接到客户端。管理员最多确保 **LoA** 一致。

示例情境

1. *对于级别 **1** 条件，最大期限配置为 **300** 秒。*

2. *在不请求任何证书的情况下发送登录请求。将使用级别 **1**，用户需要使用用户名和密码进行身份验证。令牌将具有 **acr=1**。*

3. *另一个登录请求将在 **100** 秒后发送。由于 **SSO**，用户会自动进行身份验证，令牌将返回 **acr=1**。*

4. *另一登录请求在 **201** 秒后发出（自 **2** 中的身份验证后计算为 **301** 秒）由于 **SSO**，用户会自动进行身份验证，但令牌将返回 **cr=0**，因为级别 **1** 被视为过期。*

5. *发送另一个登录请求，但现在它会在 **claims** 参数中明确请求 **ACR** 级别 **1**。用户将被要求重新使用用户名/密码进行身份验证，然后在令牌中返回 **acr=1**。*

*令牌中的 **ACR** 声明*

***ACR** 声明通过在 **acr** 客户端范围中定义的 **acr loa level** 协议映射程序添加到令牌中。此客户端范围是 **realm** 默认客户端范围，因此将添加到域中所有新创建的客户端。*

*如果您不希望令牌内部的 **acr** 声明或需要一些自定义逻辑来添加它，您可以从客户端中删除客户端范围。*

*请注意，当登录请求请求一个带有 **claims** 参数作为 基本 声明时，红帽构建的 **Keycloak** 将始终返回指定级别之一。如果无法返回指定级别中的一个（例如，如果请求的级别未知或大于身份验证流中配置的条件），则红帽构建的 **Keycloak** 将会抛出错误。*

### *8.3.5. 客户端请求注册或重置凭证*

*通常，当用户从客户端应用程序重定向到 **Keycloak** 的红帽构建时，会触发 浏览器 流。如果启用了域注册，则此流程可以允许用户注册，用户点击登录屏幕上的 **Register**。另外，如果为域启用了 **Forget** 密码，用户可以在登录屏幕上点 **Forget** 密码，这会触发 **Reset credentials** 流，用户可以在电子邮件地址确认后重置凭证。*

*有时，客户端应用程序直接重定向到 注册屏幕或重置 凭据 流会很有用。当用户在正常登录屏幕上点 **Register** 或 **Forget** 密码时，生成的操作将与 的操作匹配。自动重定向到注册或重置凭证屏幕，如下所*

示：

- 当客户端希望用户直接重定向到注册时，**OIDC** 客户端应将 **OIDC** 登录 **URL** 路径(**/auth**)中的最后一个片断替换为 **/registrations**。因此，完整 **URL** 可能类似如下：**https://keycloak.example.com/realms/your_realm/protocol/openid-connect/registrations**。

- 当客户端希望用户直接重定向到 **Reset** 凭证流时，**OIDC** 客户端应使用 **/forgot-credentials** 替换 **OIDC** 登录 **URL** 路径(**/auth**)中的最后一个片断。

警告

前面的步骤是客户端唯一支持的方法来直接请求注册或重置凭证流。为了安全起见，不支持它，推荐客户端应用程序绕过 **OIDC**/**SAML** 流，并直接重定向到其他红帽构建的 **Keycloak** 端点（如 **/realms/realm_name/login-actions** 或 **/realms/realm_name/broker**）。

## 8.4. 用户会话限制

限制用户可以配置用户的会话数量。每个域或每个客户端可以限制会话。

要为流添加会话限制，请执行以下步骤。

1. 点流的 **Add step**。
2. 从 **item** 列表中选择 **User session count limiter**。
3. 点击 **Add**。
4. 为 **User Session Count Limiter** 验证类型点 **Required**，将它的 **requirement** 设置为 **required**。

5. 单击 **User Session Count Limiter** 的 **iwl** 齿轮(**gear** 图标)。

6. 输入此配置的别名。

7. 输入用户可以在这个域中具有所需的最大会话数。例如，如果 **2** 是值，则 **2 SSO** 会话是每个用户在此域中可以具有的最大值。如果 **0** 是值，则禁用此检查。

8. 输入用户可用于客户端所需的最大会话数。例如，如果 **2** 是值，则 **2** 个 **SSO** 会话是每个客户端在此域中的最大值。因此，当用户尝试对客户端 **foo** 进行身份验证时，但该用户已在 **2** 个 **SSO** 会话中向客户端 **foo** 进行身份验证，则身份验证将被拒绝，或者现有会话将根据配置的行为终止。如果使用值 **0**，则禁用此检查。如果启用了会话限制和客户端会话限制，则将客户端会话限制始终低于会话限制。每个客户端的限制永远不会超过此用户的所有 **SSO** 会话的限制。

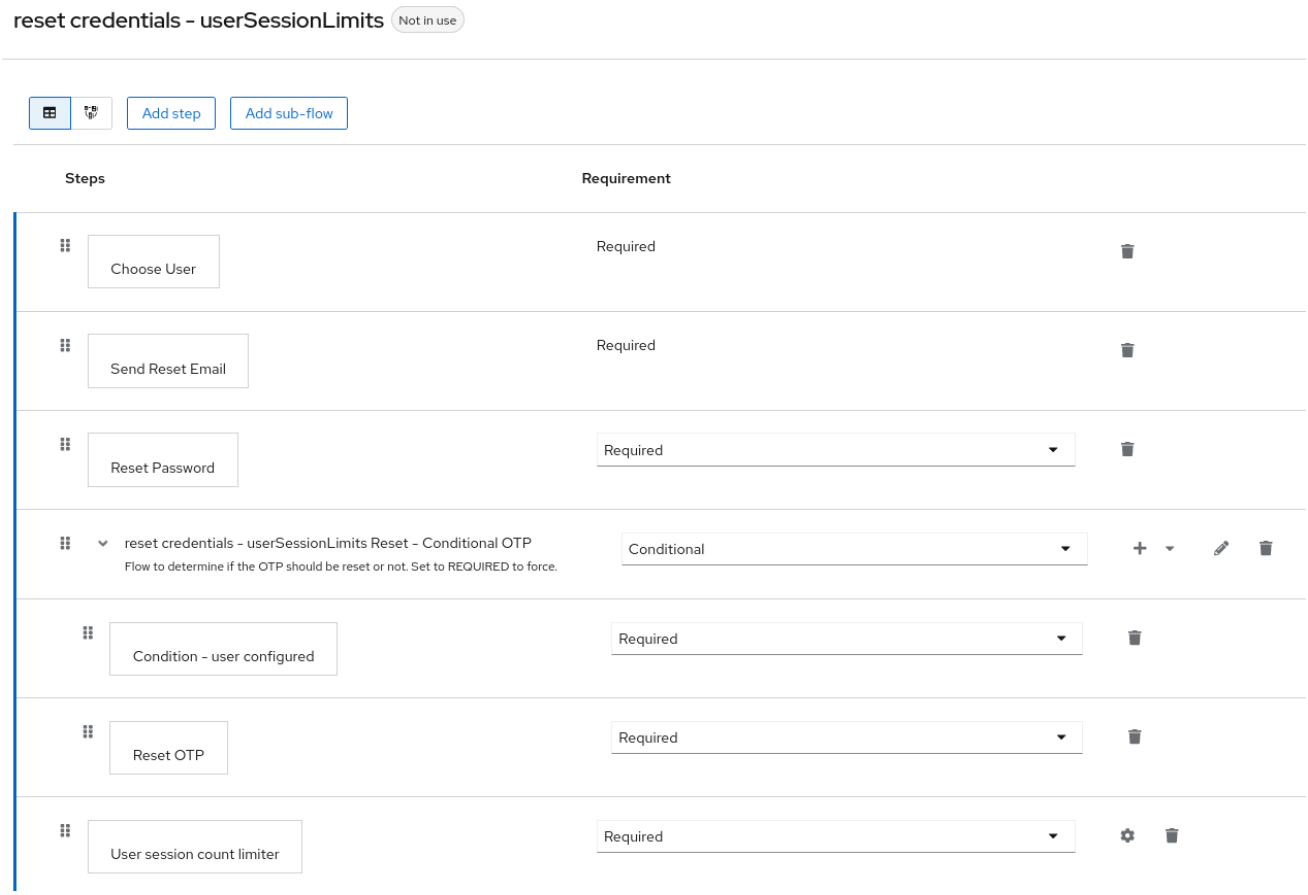
9. 选择用户在达到限制后尝试创建会话时所需的行为。可用的行为有：

   - 拒绝 新会话 **-** 当请求新会话并且达到会话限制时，无法创建新的会话。
   - 终止最旧的会话 **-** 当请求新会话并且达到会话限制时，将删除最旧的会话并创建新会话。

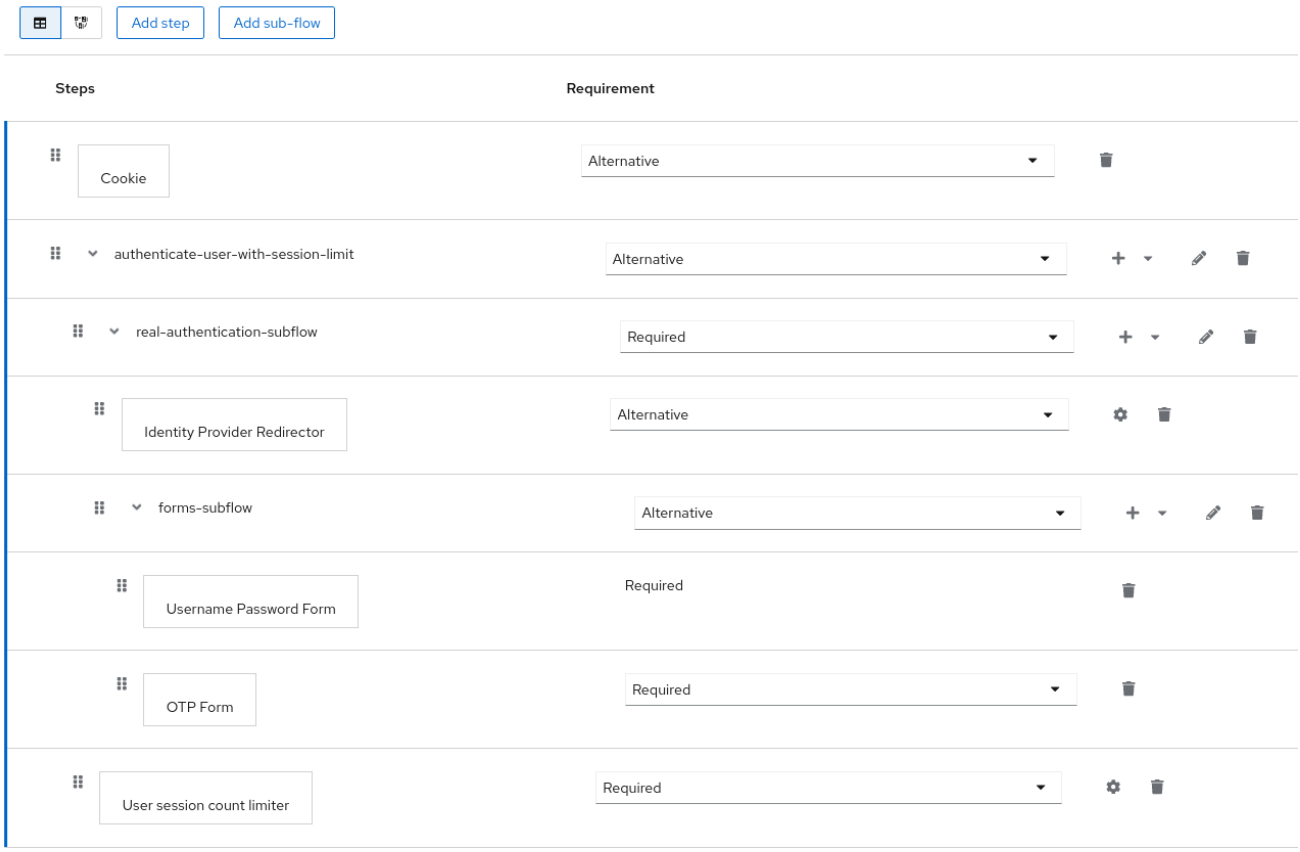
10. 另外，还可在达到限制时添加自定义错误消息。

请注意，用户会话限制应添加到您的绑定 浏览器流中，直接授予流，重置凭证，以及任何 **Post** 代理登录流。当用户已在身份验证过程中知道时（通常在身份验证流结束时）并且通常为 **REQUIRED** 时，应添加验证器。请注意，不能在同一级别上执行 **ALTERNATIVE** 和 **REQUIRED**。

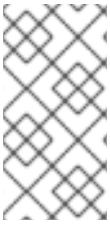
对于大多数验证器，如 **Direct grant flow,Reset credentials** 或 **Post broker login flow**，建议在身份验证流末尾将验证器作为 **REQUIRED** 添加。以下是 **Reset** 凭证流的示例 ：




*对于 浏览器 流，请考虑不要在顶级流中添加会话限制验证器。此建议是因为 **Cookie** 验证器，它根据 **SSO cookie** 自动重新验证用户。它是顶级，最好不要检查 **SSO** 重新身份验证期间的会话限制，因为用户会话已存在。因此，请考虑在相同级别（如 **Cookie** ）添加单独的 **ALTERNATIVE** 子流，如以下 **authentication-user-with-session-limit** 示例。然后，您可以在以下 **real-authentication-subflow'example** 中添加 **REQUIRED** 子流，作为 **'authenticate-user-with-session-limit** 的嵌套子流，并在同一级别上添加用户会话限制。在 **real-authentication-subflow** 中，您可以以类似默认浏览器流的方式添加真实验证器。以下示例流允许用户使用身份提供程序或密码和 **OTP** 进行身份验证：*




*关于 **Post Broker** 登录流，您可以添加 用户会话限制，作为身份验证流中的唯一验证器，只要您在与身份提供程序进行身份验证后没有触发的其他验证器。但是，请确保此流被配置为身份提供程序上的 **Post Broker** 流。需要此要求，以便与身份提供程序进行身份验证也参与会话限制。*



*注意*

*目前，管理员负责在不同配置之间保持一致性。因此，请确保您的所有流都使用与用户会话限制相同的配置。*

*注意*

***CIBA** 不提供用户会话限制功能。*

### *8.5. KERBEROS*

*红帽构建的 **Keycloak** 支持通过 **Simple** 和 **Protected GSSAPI Negotiation Mechanism (SPNEGO)** 协议登录。在用户验证会话后，**SPNEGO** 通过 **Web** 浏览器以透明的方式进行身份验证。对于非 **Web** 情况，或者在登录期间没有票据时，红帽构建的 **Keycloak** 支持使用 **Kerberos** 用户名和密码登录。*

*Web 身份验证的典型用例如下：*

1. *用户登录桌面。*
2. *用户使用浏览器访问由红帽构建的 Keycloak 保护的 Web 应用程序。*
3. *应用程序重定向到红帽 Keycloak 登录的构建。*
4. *Red Hat build of Keycloak 呈现 HTML 登录屏幕，其状态为 401 和 HTTP 标头 WWW-Authenticate: Negotiate*
5. *如果浏览器有来自桌面登录的 Kerberos 票据，浏览器会将桌面登录信息传送到标题 授权中的 Keycloak 的红帽构建： Negotiate 'spnego-token'。否则，它会显示标准登录屏幕，用户输入登录凭据。*
6. *红帽构建的 Keycloak 从浏览器验证令牌并验证用户。*
7. *如果使用带有 Kerberos 身份验证支持的 LDAPFederationProvider，红帽构建的 Keycloak 会置备来自 LDAP 的用户数据。如果使用 KerberosFederationProvider，红帽构建的 Keycloak 可让用户更新配置集并预先填充登录数据。*
8. *红帽构建的 Keycloak 返回应用程序。红帽构建的 Keycloak 和应用程序通过 OpenID Connect 或 SAML 消息进行通信。红帽构建的 Keycloak 充当 Kerberos/SPNEGO 登录的代理。因此，红帽通过 Kerberos 进行身份验证的 Keycloak 在应用程序中是隐藏的。*

*警告*

*默认情况下，**Negotiate** **www-authenticate** 模式允许 **NTLM** 作为 **Kerberos** 的回退，以及 **Windows NTLM** 中的一些 **Web** 浏览器支持。如果 **www-authenticate** 质询来自浏览器允许列表之外的服务器，用户可能会遇到 **NTLM** 对话框。用户需要在对话框中点击取消按钮才能继续，因为红帽构建的 **Keycloak** 不支持此机制。如果 **I intranet Web** 浏览器没有严格配置，或者红帽构建的 **Keycloak** 在 **I intranet** 和 **Internet** 中为用户提供了服务，则可能会出现这种情况。自定义验证器 可用于将 **Negotiate** 质询限制为主机白名单。*

*执行以下步骤设置 **Kerberos** 身份验证：*

1. *__Kerberos__ 服务器的设置和配置(KDC)。*
2. *红帽构建的 **Keycloak** 服务器的设置和配置。*
3. *客户端机器的设置和配置。*

### *8.5.1. 设置 Kerberos 服务器*

*设置 **Kerberos** 服务器的步骤取决于操作系统(OS)和 **Kerberos** 厂商。有关设置和配置 **Kerberos** 服务器的说明，请参阅 **Windows Active Directory**、**MIT Kerberos** 和您的操作系统文档。*

*在设置过程中，执行以下步骤：*

1. *在您的 **Kerberos** 数据库中添加一些用户主体。您还可以将您的 **Kerberos** 与 **LDAP** 集成，因此用户帐户从 **LDAP** 服务器置备。*
2. *为**"HTTP"**服务添加服务主体。例如，如果红帽构建的 **Keycloak** 服务器在 **www.mydomain.org** 上运行，请添加服务主体 **HTTP/www.mydomain.org@<kerberos realm>**。*

*在 **MIT Kerberos** 上，您将运行一个 **"kadmin"** 会话。在带有 **MIT Kerberos** 的机器上，您*

可以使用以下命令：

```
sudo kadmin.local
```

然后，使用如下命令添加 **HTTP** 主体并将其密钥导出到 **keytab** 文件中：

```
addprinc -randkey HTTP/www.mydomain.org@MYDOMAIN.ORG
ktadd -k /tmp/http.keytab HTTP/www.mydomain.org@MYDOMAIN.ORG
```

确保运行红帽构建的 **Keycloak** 的主机上可以访问 **keytab** 文件 **/tmp/http.keytab**。

### 8.5.2. 设置和配置红帽构建的 Keycloak 服务器

在您的机器上安装 **Kerberos** 客户端。

流程

1. 安装 **Kerberos** 客户端。如果您的机器运行 **Fedora**、**Ubuntu** 或 **RHEL**，请安装 **freeipa-client** 软件包，其中包含 **Kerberos** 客户端和其他工具。

2. 配置 **Kerberos** 客户端（在 **Linux** 中，配置设置位于 **/etc/krb5.conf** 文件中）。

   将您的 **Kerberos** 域添加到配置中，并配置服务器运行的 **HTTP** 域。

   例如，对于 **MYDOMAIN.ORG** 域，您可以配置 **domain_realm** 部分，如下所示：

   ```
   [domain_realm]
    .mydomain.org = MYDOMAIN.ORG
    mydomain.org = MYDOMAIN.ORG
   ```

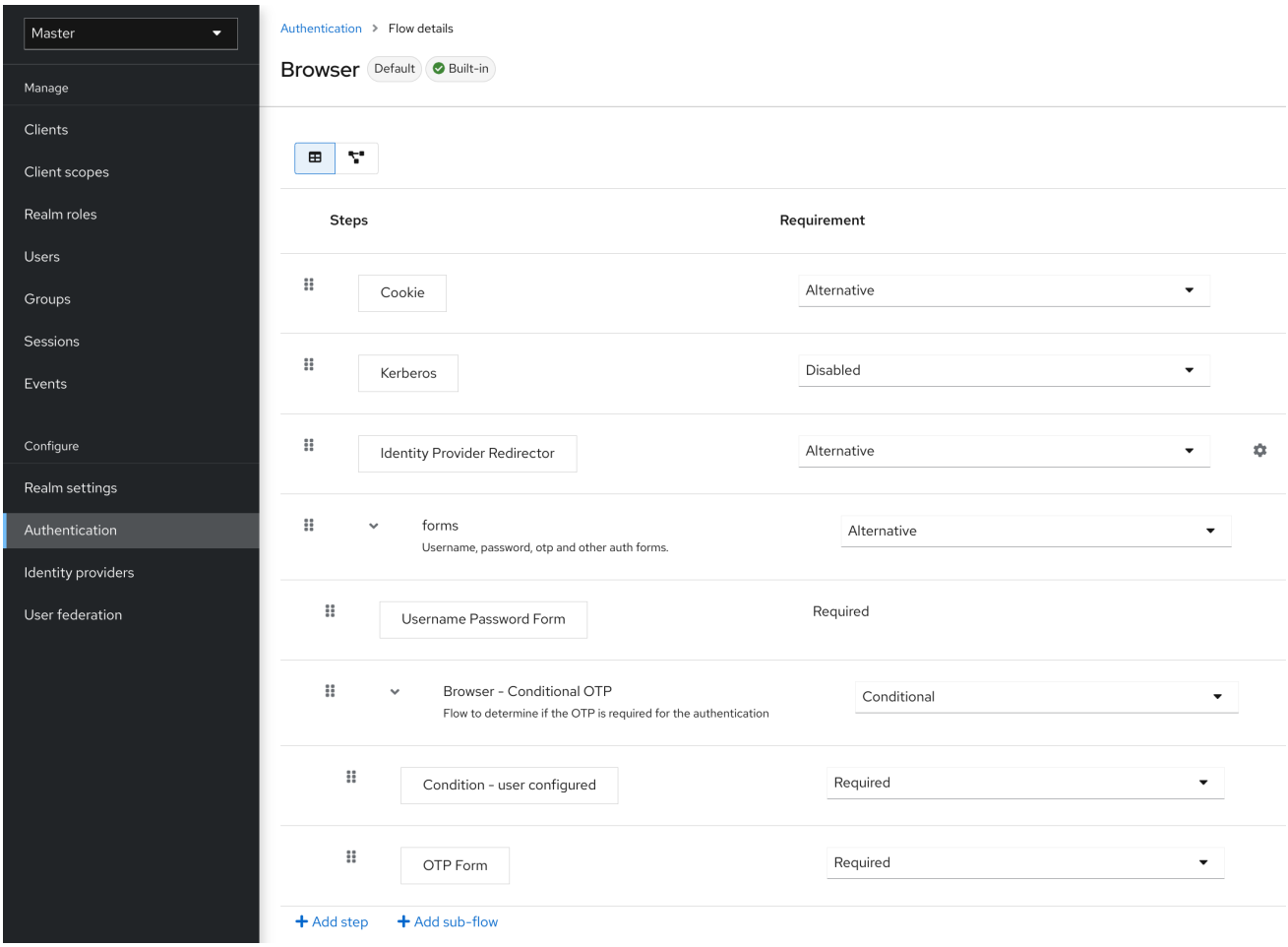
3. 使用 **HTTP** 主体导出 **keytab** 文件，并确保该文件可供运行红帽构建的 **Keycloak** 服务器的进程访问。对于生产环境，请确保该文件只对此过程可读。

   对于以上 **MIT Kerberos** 示例，我们将 **keytab** 导出到 **/tmp/http.keytab** 文件中。如果您的密钥分发中心(**KDC**) 和红帽构建的 **Keycloak** 运行在同一主机上运行，则该文件已可用。

#### 8.5.2.1. 启用 SPNEGO 处理

默认情况下，红帽构建的 **Keycloak** 禁用 **SPNEGO** 协议支持。要启用它，请转至 浏览器流 并启用 **Kerberos**。

浏览器流




将 禁用的 **Kerberos** 要求设置为 替代 (**Kerberos** 是可选的)或 必需 （浏览器必须启用 **Kerberos**）。如果您还没有将浏览器配置为使用 **SPNEGO** 或 **Kerberos**，则 **Keycloak** 的 **Red Hat build** 会返回常规登录屏幕。

#### 8.5.2.2. 配置 Kerberos 用户存储联邦供应商

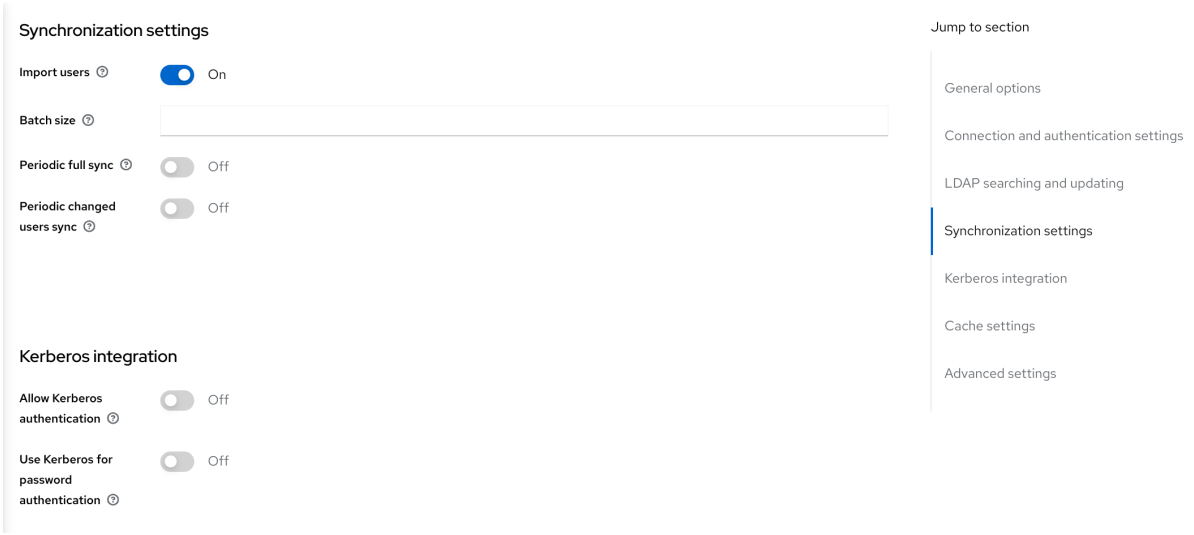
现在，您必须使用 User Storage Federation 来配置红帽构建的 **Keycloak** 如何解释 **Kerberos** 票据。有两个不同的联合供应商提供 **Kerberos** 身份验证支持。

要使用由 **LDAP** 服务器支持的 **Kerberos** 进行身份验证，请配置 LDAP Federation Provider。

*流程*

1. *进入 **LDAP** 供应商的配置页面。*

***LDAP kerberos** 集成*




2. *将 **Allow Kerberos** 身份验证 切换到 **ON***

*允许 **Kerberos** 身份验证 使红帽构建 **Keycloak** 使用 **Kerberos** 主体访问用户信息，以便信息可以导入到红帽构建的 **Keycloak** 环境中。*

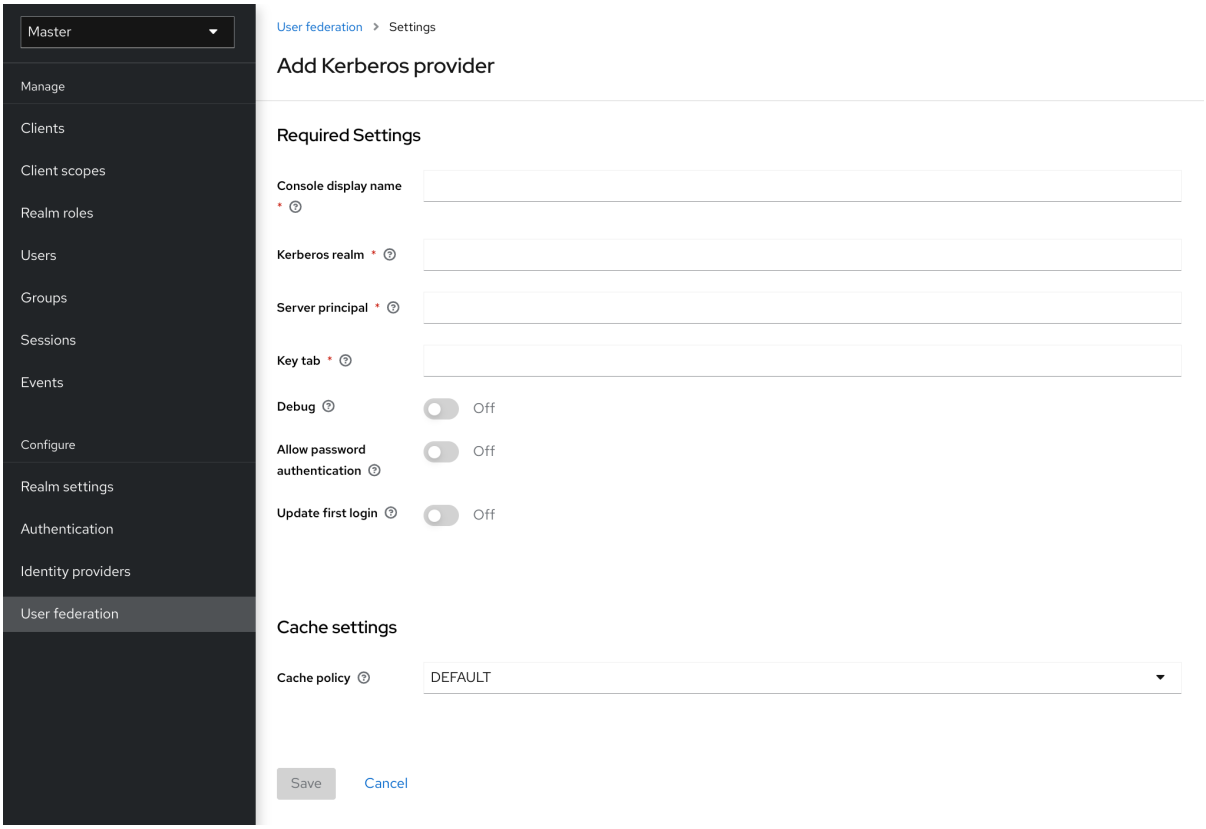
*如果 **LDAP** 服务器没有备份您的 **Kerberos** 解决方案，请使用 **Kerberos** 用户存储 **Federation** 提供程序。*

*流程*

1. *点菜单中的 **User Federation**。*

2. *从 **Add provider** 选择 **Kerberos**。*

***Kerberos** 用户存储供应商*




*Kerberos 供应商解析 Kerberos 票据以获取简单主体信息，并将信息导入到本地红帽 Keycloak 数据库构建中。没有置备用户配置文件信息，如名字、姓氏和电子邮件。*

### *8.5.3. 设置和配置客户端机器*

*客户端机器必须具有 Kerberos 客户端并设置 krb5.conf，如上所述。客户端机器还必须在其浏览器中启用 SPNEGO 登录支持。如果您使用 Firefox 浏览器，请参阅为 Kerberos 配置 Firefox。*

*.mydomain.org URI 必须位于 network.negotiate-auth.trusted-uris 配置选项中。*

*在 Windows 域中，客户端不需要调整其配置。Internet Explorer 和 Edge 可以参与 SPNEGO 身份验证。*

### *8.5.4. 凭证委托*

*Kerberos 支持凭据委派。应用程序可能需要访问 Kerberos 票据，以便他们可以重新使用它来与 Kerberos 保护的其他服务进行交互。因为红帽构建的 Keycloak 服务器处理 SPNEGO 协议，所以您必须将 GSS 凭证传播到 OpenID Connect 令牌声明或 SAML 断言属性中的应用程序。红帽构建的 Keycloak 从红帽构建的 Keycloak 服务器将其传送到您的应用程序。要将此声明插入到令牌或断言中，*

每个应用必须启用内置协议映射器 **gss** 委派凭据。这个映射程序位于应用程序客户端页面的 **Mappers** 选项卡中。如需了解更多详细信息，请参阅协议映射程序 章节。

在使用应用程序对其他服务进行 **GSS** 调用前，应用程序必须反序列化从红帽构建的 **Keycloak** 接收的声明。当您从访问令牌到 **GSSCredential** 对象时，请使用传递给 **GSSManager.createContext** 方法此凭证创建 **GSSContext**。例如：

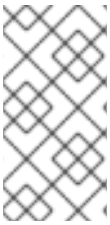
```
// Obtain accessToken in your application.
KeycloakPrincipal keycloakPrincipal = (KeycloakPrincipal) servletReq.getUserPrincipal();
AccessToken accessToken = keycloakPrincipal.getKeycloakSecurityContext().getToken();

// Retrieve Kerberos credential from accessToken and deserialize it
String serializedGssCredential = (String) accessToken.getOtherClaims().

get(org.keycloak.common.constants.KerberosConstants.GSS_DELEGATION_CREDENTIAL);

GSSCredential deserializedGssCredential =
org.keycloak.common.util.KerberosSerializationUtils.
   deserializeCredential(serializedGssCredential);

// Create GSSContext to call other Kerberos-secured services
GSSContext context = gssManager.createContext(serviceName, krb5Oid,
   deserializedGssCredential, GSSContext.DEFAULT_LIFETIME);
```



注意

在 **krb5.conf** 文件中配置 **forwardable Kerberos** 票据，并在浏览器中添加对授权凭证的支持。

警告

凭证委托存在安全隐患，因此仅在需要时才使用它。有关更多详细信息 和示例，请参阅本文档。

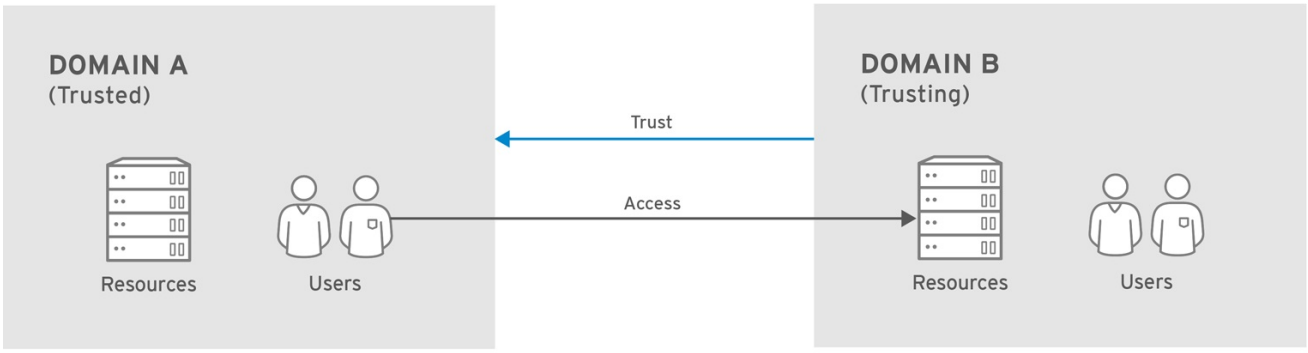
### 8.5.5. 跨域信任

在 **Kerberos** 协议中，**realm** 是一组 **Kerberos** 主体。这些主体的定义存在于 **Kerberos** 数据库中，通常是 **LDAP** 服务器。

**Kerberos** 协议允许跨域信任。例如，如果 **2** 个 **Kerberos** 域(**A** 和 **B**)存在，则跨域信任将允许 **realm**

*A 中的用户访问 realm B 的资源。realm B 信任域 A.*

*Kerberos 跨域信任*




*红帽构建的 Keycloak 服务器支持跨域信任。要实现此功能，请执行以下操作：*

- *为跨域信任配置 Kerberos 服务器。实施这个步骤取决于 Kerberos 服务器实现。此步骤是必需的，将 Kerberos 主体 krbtgt/B@A 添加到域 A 和 B 的 Kerberos 数据库中。这个主体必须在两个 Kerberos 域上具有相同的密钥。主体必须在两个域中具有相同的密码、密钥版本号和密码。详情请查看 Kerberos 服务器文档。*

*注意*

*默认情况下，跨域信任是单向的。您必须将主体 krbtgt/A@B 添加到 Kerberos 数据库中，以实现域 A 和域 B 之间的双向信任。但是，信任默认是传输的。如果 realm B 信任 realm A 和 realm C 信任域 B，则 realm C 信任域 A，且没有主体、krbtgt/C@A，可用。Kerberos 客户端可能需要额外的配置（如 capaths），以便客户端可以找到信任路径。详情请查看 Kerberos 文档。*

- *配置红帽构建的 Keycloak 服务器*
  - *当使用支持 Kerberos 的 LDAP 存储供应商时，为域 B 配置服务器主体，如下例所示： HTTP/mydomain.com@B。如果来自 realm A 的用户成功验证到红帽构建的 Keycloak，则 LDAP 服务器必须找到域 A 中的用户，因为红帽构建的 Keycloak 必须执行 SPNEGO 流，然后查找用户。*

*查找用户基于 LDAP 存储供应商选项 Kerberos 主体属性。当为实例配置了类似 userPrincipalName*

*的值的实例时，在用户 **john@A** 的 **SPNEGO** 身份验证后，红帽构建的 **Keycloak** 将尝试查找与 **john@A** 等效属性 **userPrincipalName** 的 **LDAP** 用户。如果 **Kerberos** 主体属性 留空，则 **Keycloak** 的红帽构建将根据其 **kerberos** 主体的前缀，并带有域省略的内容。例如，**Kerberos** 主体用户 **john@A** 必须在用户名 **john** 下的 **LDAP** 中可用，因此通常位于 **LDAP DN** 下，如 **uid=john,ou=People,dc=example,dc=com**。如果您希望域 **A** 和 **B** 中的用户进行身份验证，请确保 **LDAP** 可以从域 **A** 和 **B** 中找到用户。*

- *当使用 **Kerberos** 用户存储提供程序（通常没有 **LDAP** 集成）时，将服务器主体配置为 **HTTP/mydomain.com@B**，并且 **Kerberos** 域 **A** 和 **B** 的用户必须能够进行身份验证。*

*支持多个 **Kerberos** 域中的用户，因为每个用户都有属性 **KERBEROS_PRINCIPAL** 引用用于身份验证的 **kerberos** 主体，这用于进一步查找此用户。为了避免在 **kerberos** 域 **A** 和 **B** 中存在用户 **john** 时冲突，红帽构建的 **Keycloak** 用户的用户名可能包含 **kerberos** 域小写。例如，用户名将是 **john@a**。仅在 **realm** 与配置的 **Kerberos** 域 匹配时，可能会从生成的用户名中省略 **realm** 后缀。例如，只要 **Kerberos** 提供程序上配置了 **Kerberos** 域是 **A**，即 **Kerberos** 主体 **john @ A** 的 **john**。*

### *8.5.6. 故障排除*

*如果您有问题，请启用额外的日志记录来调试问题：*

- *在 **Kerberos** 或 **LDAP** 联邦供应商的管理控制台中启用 **Debug** 标志*
- *为类别 **org.keycloak** 启用 **TRACE** 日志记录，以便在服务器日志中接收更多信息*
- *添加系统属性 **-Dsun.security.krb5.debug=true** 和 **-Dsun.security.spnego.debug=true***

## *8.6. X.509 客户端证书用户身份验证*

*如果您将服务器配置为使用 **mutual SSL** 身份验证，**Red Hat build of Keycloak** 支持使用 **X.509** 客户端证书登录。*

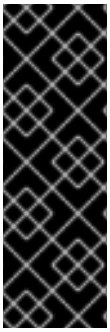
*一个典型的工作流：*

- *客户端通过 **SSL/TLS** 频道发送身份验证请求。*

- 在 **SSL/TLS** 握手期间，服务器和客户端会交换其 **x.509/v3** 证书。
- 容器(**JBoss EAP**)验证证书 **PKIX** 路径和证书过期日期。
- **X.509** 客户端证书验证器使用以下方法验证客户端证书：
  - 使用 **CRL** 或 **CRL** 分发点检查证书撤销状态。
  - 使用 **OCSP** （在线证书状态协议）检查证书撤销状态。
  - 验证证书中的密钥是否与预期的密钥匹配。
  - 验证证书中的扩展密钥是否与预期的扩展密钥匹配。
- 如果其中任何一个检查失败，则 **x.509** 身份验证会失败。否则，验证器提取证书身份并将其映射到现有用户。

当证书映射到现有用户时，行为会根据身份验证流进行不同：

- 在浏览器流中，服务器会提示用户确认其身份或使用用户名和密码登录。
- 在 **Direct Grant Flow** 中，服务器登录用户。



重要

请注意，**web** 容器负责验证证书 **PKIX** 路径。红帽构建的 **Keycloak** 端的 **X.509** 身份验证程序只提供检查证书过期、证书撤销状态和密钥用法的额外支持。如果您使用部署在反向代理后的 **Keycloak** 的红帽构建，请确保将反向代理配置为验证 **PKIX** 路径。如果您不使用反向代理和用户直接访问 **JBoss EAP**，则 **JBoss EAP** 您应非常正常，只要其配置了 **PKIX** 路径，就可以验证 **PKIX** 路径。

### 8.6.1. 功能

支持的证书身份源：

- 使用正则表达式匹配 **SubjectDN**
- **X500** 主题的电子邮件属性
- 来自 **Subject Alternative Name Extension (RFC822Name General Name)**的 **X500** 主题的电子邮件
- 来自 **Subject Alternative Name Extension** 的 **X500** 主题的其他名称。另一个名称是 **User Principal Name (UPN)**，通常为。
- **X500** 主题的通用名称属性
- 使用正则表达式匹配 **IssuerDN**
- 证书序列号
- 证书序列号和 **IssuerDN**
- **SHA-256** 证书指纹
- **PEM** 格式的完整证书

#### 8.6.1.1. 正则表达式

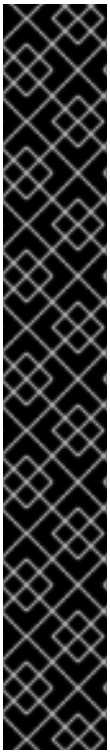
**Red Hat build of Keycloak** 通过使用正则表达式作为过滤器从主题 **DN** 或 **Issuer DN** 中提取证书身份。例如，这个正则表达式与 **email** 属性匹配：

```
emailAddress=(.*?)(?:,|$)
```

*如果将 **Identity Source** 设置为 **Match SubjectDN using regular expression** 或 **Match IssuerDN using regular expression**，会应用正则表达式过滤。*

#### *8.6.1.1.1. 将证书身份映射到现有用户*

*证书身份映射可将提取的用户身份映射到现有用户的用户名、电子邮件或自定义属性，其值与证书身份匹配。例如，在按 用户名或电子邮件搜索现有用户时，将 **Identity source** 设置为 **Subject** 或 **User mapping method to Username** 或 **email** 使 **X.509** 客户端证书验证器 使用 **email** 属性作为搜索条件。*



*重要*

- *如果您在 **realm** 设置中禁用 电子邮件登录，则相同的规则适用于证书身份验证。用户无法使用 **email** 属性登录。*
- *使用证书序列号和 **IssuerDN** 作为身份源，需要两个用于序列号和 **IssuerDN** 的自定义属性。*
- ***SHA-256** 证书指纹 是 **SHA-256** 证书指纹的小写十六进制表示。*
- *使用 **PEM** 格式的完整证书 作为身份源，仅限于映射到外部联邦源（如 **LDAP**）的自定义属性。由于长度限制，红帽构建的 **Keycloak** 无法将证书存储在数据库中，因此对于 **LDAP**，您必须启用 **Always Read Value from LDAP**。*

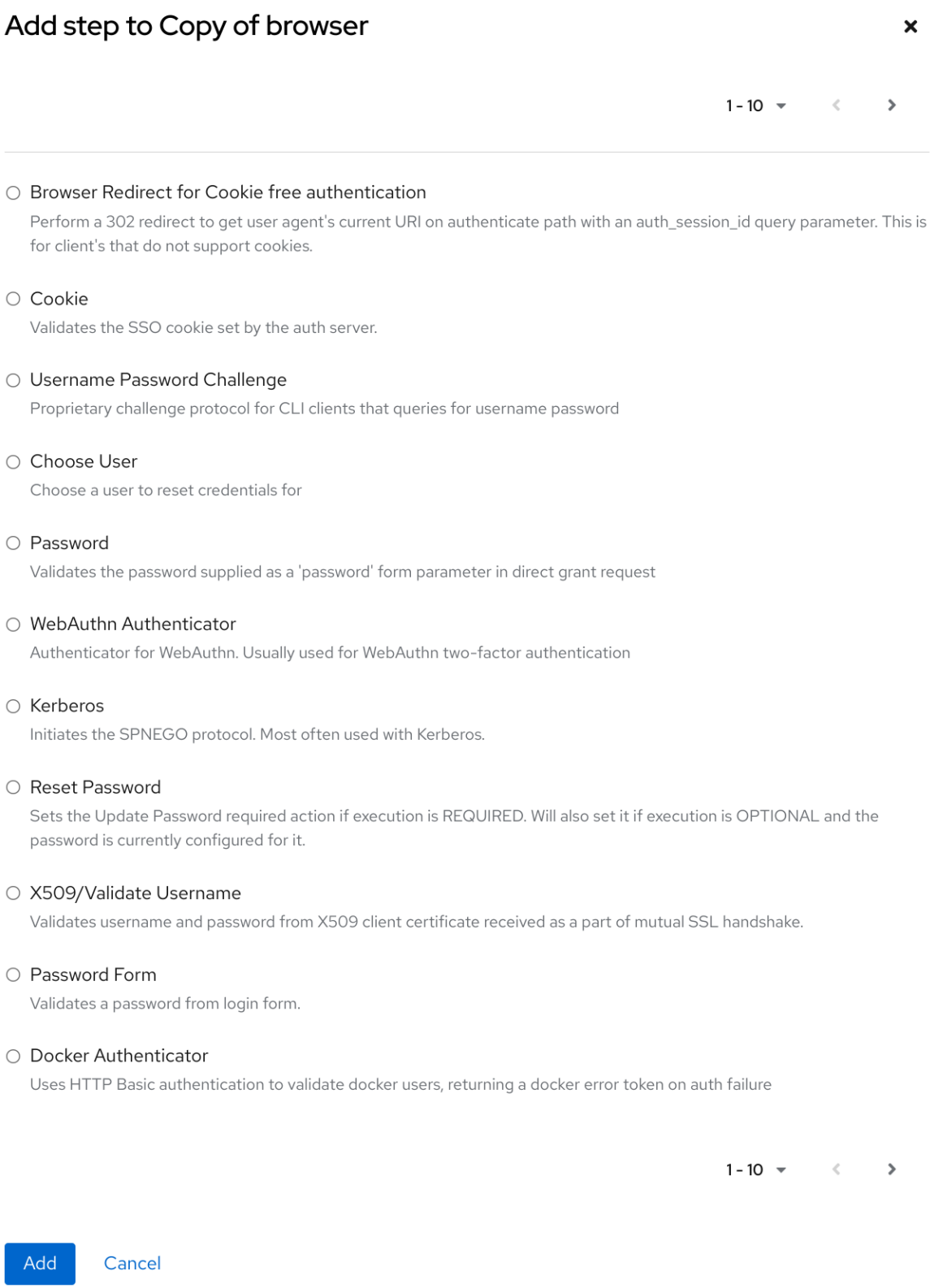
#### *8.6.1.1.2. 扩展证书验证*

- *使用 **CRL** 吊销状态检查。*
- *使用 **CRL/Distribution Point** 进行撤销状态检查。*
- *使用 **OCSP/Responder URI** 进行撤销状态检查。*
- *证书密钥用法验证.*
- *证书扩展密钥用法验证.*

### *8.6.2. 在浏览器流中添加 X.509 客户端证书身份验证*

1. *点菜单中的 **Authentication**。*
2. *单击 浏览器 流。*
3. *从 **Action** 列表中，选择 **Duplicate**。*
4. *输入副本的名称。*
5. *点 **Duplicate**。*
6. *点 **Add step**。*
7. *点 **"X509/Validate Username Form"**。*
8. *点击 **Add**。*

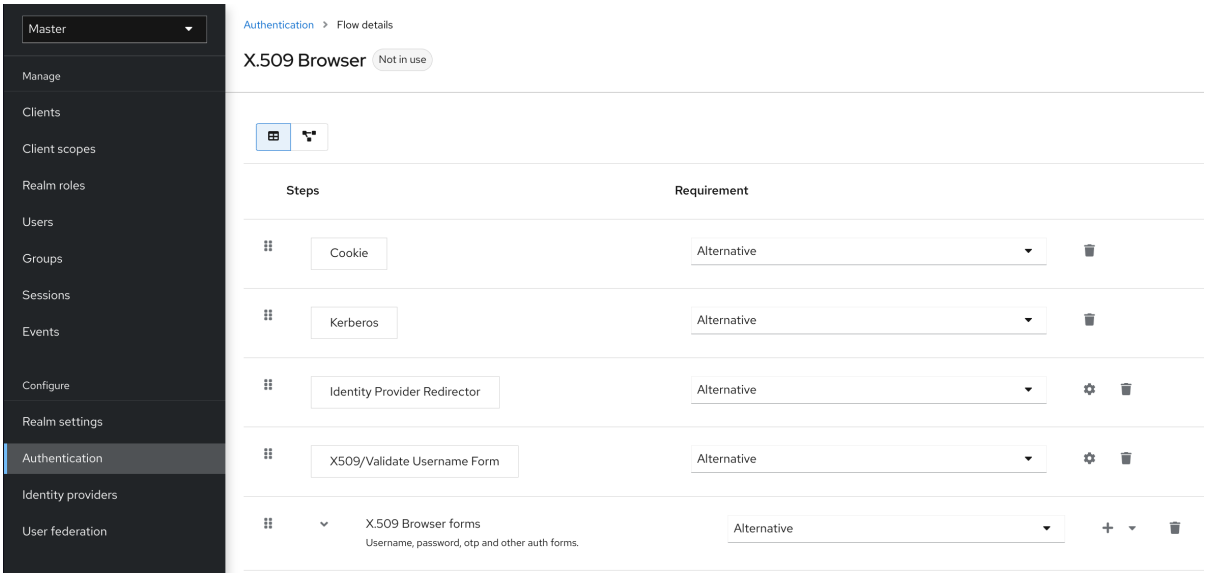
***X509** 执行*




9. *点并在**"Browser Forms"**执行中拖动 **"X509/Validate Username Form"**。*

10. *将要求设置为**"ALTERNATIVE"**。*

***X509 浏览器流***




11. *单击 **Action** 菜单。*

12. *单击 绑定流。*

13. *从下拉列表中选择 **Browser** 流。*

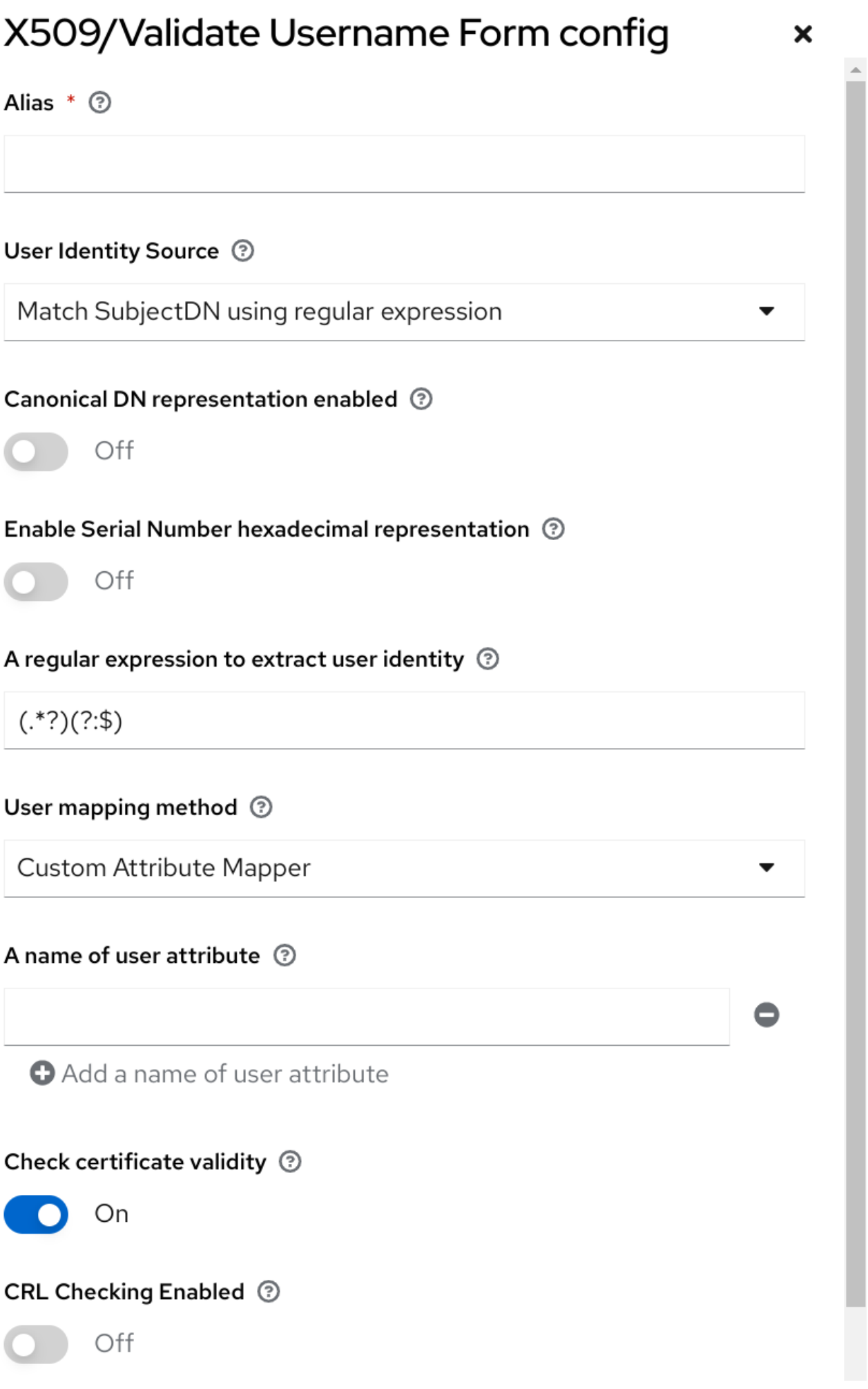
14. *点击 **Save**。*

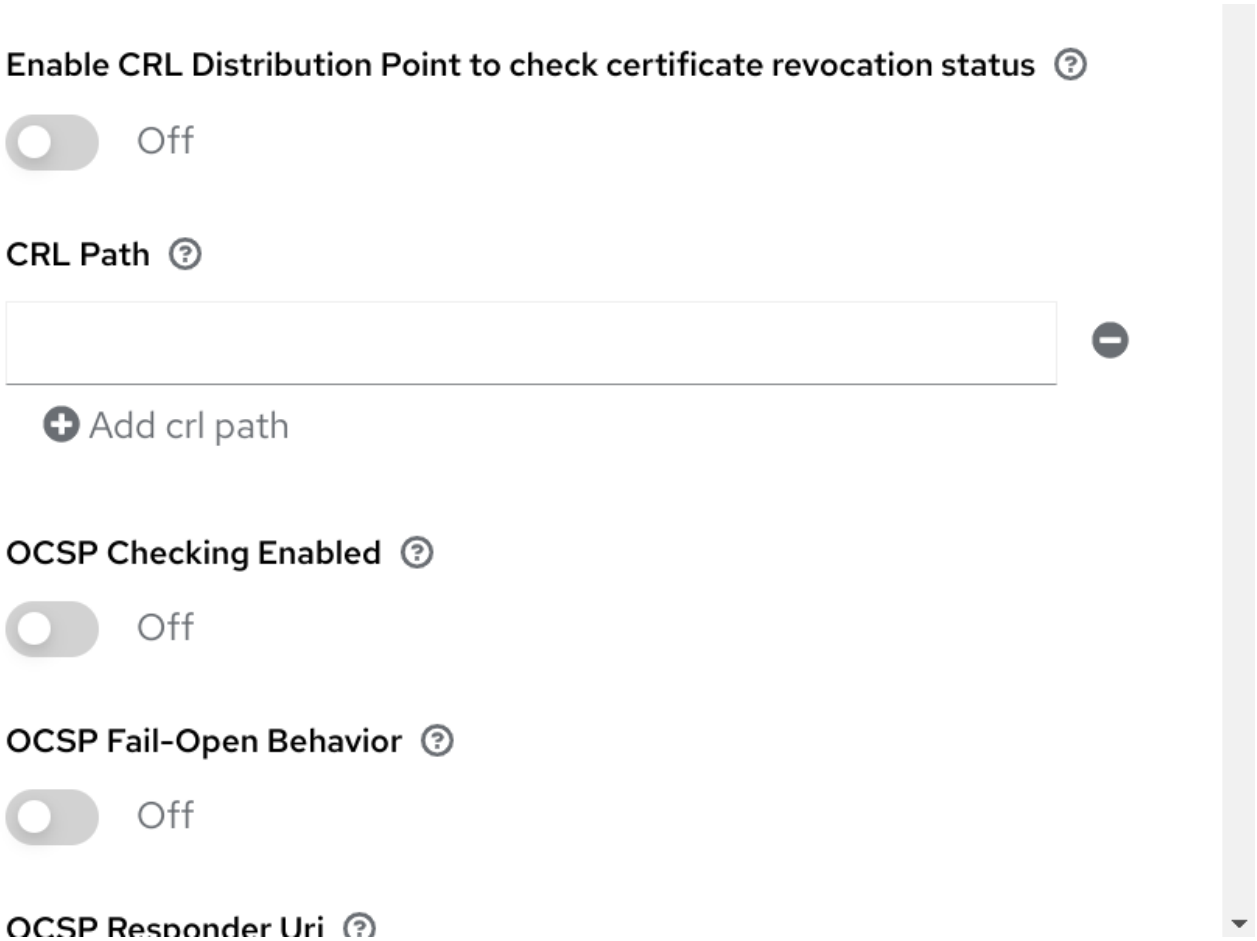
***X509 浏览器流绑定***

| | | |
|---|---|---|
| Docker auth Built-in | Docker auth | Used by Docker clients to authenticate against the IDP |
| X.509 Browser | Browser flow | browser based authentication |
| Client-flow | Specific clients | |

### *8.6.3. 配置 X.509 客户端证书身份验证*

***X509 配置***


X509/Validate Username Form config
Alias *
User Identity Source
Match SubjectDN using regular expression
Canonical DN representation enabled
Off
Enable Serial Number hexadecimal representation
Off
A regular expression to extract user identity
(.*?)(?:$)
User mapping method
Custom Attribute Mapper
A name of user attribute
Add a name of user attribute
Check certificate validity
On
CRL Checking Enabled
Off




*用户身份源*

*定义从客户端证书中提取用户身份的方法。*

*启用规范 **DN** 表示*

*定义是否使用规范格式来确定可分辨的名称。官方 **Java API** 文档描述了格式。这个选项会影响两个用户身份源 **Match SubjectDN**，只使用 正则表达式的 **Match IssuerDN**。设置新的 **Keycloak** 实例时，启用这个选项。禁用这个选项，以保持与现有红帽构建的 **Keycloak** 实例的向后兼容性。*

*启用序列号十六进制表示*

*以十六进制表示 序列号。符号位设为 **1** 的序列号必须保留为 **00** 八位。例如，根据 **RFC5280**，带有十进制值 **161** 或十六进制的 **a1** 的序列号被编码为 **00a1**。如需了解更多详细信息，请参阅 **RFC5280, appendix-B**。*

*正则表达式*

*用作提取证书身份的过滤器的正则表达式。表达式必须包含单个组。*

*用户映射方法*

*定义与现有用户匹配证书身份的方法。用户名或电子邮件按用户名或电子邮件 搜索现有用户。自定义属性映射程序 搜索具有与证书身份匹配的自定义属性的现有用户。自定义属性的名称可以配置。*

用户属性的名称

其值与证书身份匹配的自定义属性。当属性映射与多个值相关时使用多个自定义属性，例如 **'Certificate Serial Number** 和 **IssuerDN'**。

**CRL** 检查已启用

使用证书撤销列表检查证书的撤销状态。列表的位置在 **CRL** 文件路径 属性中定义。

启用 **CRL** 分发点以检查证书撤销状态

使用 **CDP** 检查证书撤销状态。大多数 **PKI** 授权在其证书中包含 **CDP**。

**CRL** 文件路径

包含 **CRL** 列表的文件的路径。如果启用了 **CRL Checking Enabled** 选项，则该值必须是到有效文件的路径。

启用 **OCSP** 检查

使用在线证书状态协议检查证书撤销状态。

**OCSP Fail-Open** 行为

默认情况下，**OCSP** 检查必须返回正响应，才能继续成功进行身份验证。但是，这个检查可能不方便：例如，**OCSP** 服务器可能会无法访问、过载，或者客户端证书可能不包含 **OCSP** 响应器 **URI**。当此设置打开 **ON** 时，只有在 **OCSP** 响应程序收到显式负响应并且证书被绝对撤销时，才会拒绝身份验证。如果没有可用的有效 **OCSP** 响应，则接受身份验证尝试。

**OCSP Responder URI**

覆盖证书中的 **OCSP** 响应器 **URI** 的值。

验证密钥使用

验证是否设置了证书的 **KeyUsage** 扩展位。例如，**"digitalSignature,KeyEncipherment"** 验证是否设置了 **KeyUsage** 扩展中的位 **0** 和 **2**。保留此参数为空以禁用 **Key Usage** 验证。如需更多信息，请参阅 RFC5280, Section-4.2.1.3。当发生密钥使用不匹配时，红帽构建的 **Keycloak** 会引发错误。

验证扩展的密钥用法

验证扩展密钥使用扩展中定义的一个或多个目的。如需更多信息，请参阅 RFC5280, Section-4.2.1.12。将此参数设置为空，以禁用扩展的密钥用法验证。当发布 **CA** 和发生密钥使用扩展不匹配时，**Red Hat build of Keycloak** 会抛出一个错误。

验证证书策略

*验证一个或多个策略 **OID**，如证书策略扩展中定义的。请参阅 RFC5280，Section-4.2.1.4。将参数留空，以禁用证书策略验证。应使用逗号分隔多个策略。*

*证书策略验证模式*

*当在 **Validate Certificate Policy** 设置中指定多个策略时，它会决定匹配是否应该检查要存在的所有请求策略，或者一个匹配是否足以成功进行身份验证。默认值为 **All**，表示客户端证书中所有请求的策略都应存在。*

*绕过身份确认*

*如果启用，**X.509** 客户端证书身份验证不会提示用户确认证书身份。身份验证成功后，红帽在用户构建 **Keycloak** 进行签名。*

*重新验证客户端证书*

*如果设置，则始终使用配置的信任存储中存在的证书在应用程序级别验证客户端证书信任链。如果底层 **Web** 服务器没有强制实施客户端证书链验证，这很有用，例如，因为它位于非评估负载均衡器或反向代理后面，或者允许的 **CA** 数量对于 **mutual SSL** 协商来说太大（大多数浏览器将最大 **SSL** 协商数据包大小上限为 **32767** 字节），它对应于 **200** 公告的 **CA**。默认情况下，这个选项为 **off**。*

### *8.6.4. 将 **X.509** 客户端证书身份验证添加到 **Direct Grant Flow***

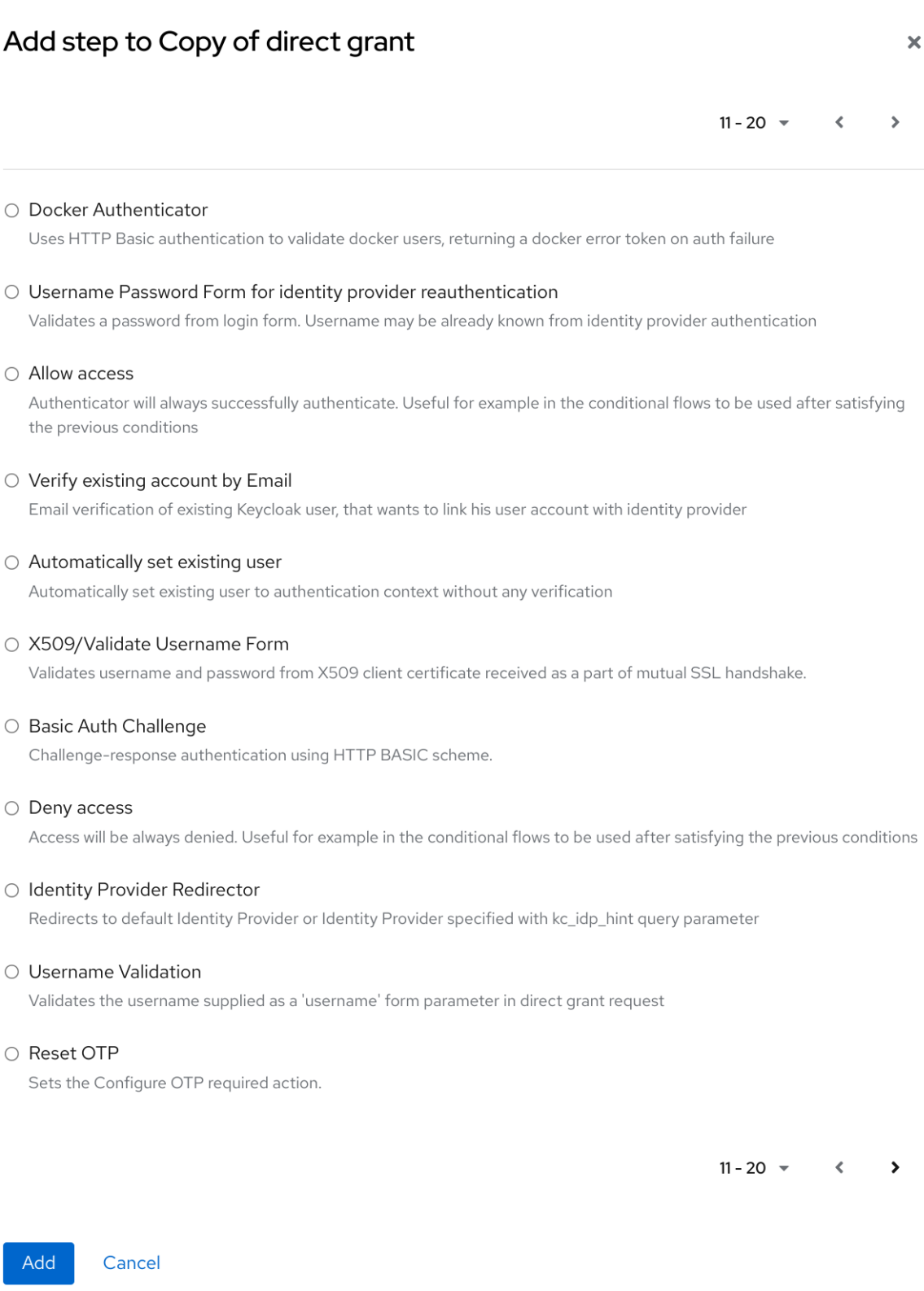
1. *点菜单中的 **Authentication**。*
2. *从 **"Action list"** 选择 **Duplicate** 来复制内置**"Direct grant"**流。*
3. *输入副本的名称。*
4. *点 **Duplicate**。*
5. *点创建的流。*
6. *点垃圾桶图标 **"Username Validation"** 的图标，然后点 **Delete**。*
7. *点击垃圾箱图标 **"Password"** 的图标，然后单击 **Delete**。*

8. 点 ***Add step***。

9. 点 ***"X509/Validate Username"***。

10. 点击 ***Add***。

***X509*** *直接授权执行*




11. *按照 **x509 Browser Flow** 部分中描述的步骤设置 **x509** 身份验证配置。*

12. *单击 **Bindings** 选项卡。*

13. 点 **Direct Grant Flow** 下拉列表。

14. 点新创建的 **"x509 Direct Grant"** 流。

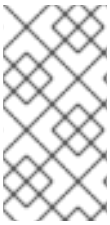
15. 点 **Save**。

**X509** 直接授权流绑定

| | | |
|---|---|---|
| X509 Direct grant | Direct grant flow | OpenID Connect Resource Owner Grant |

### 8.7. W3C WEB 身份验证(WEBAUTHN)

红帽构建的 **Keycloak** 支持 **W3C Web** 身份验证(WebAuthn)。红帽构建的 **Keycloak** 充当 **WebAuthn** 的 **Rely ing** 第三方(RP)。



注意

**Webauthn** 的操作成功取决于用户的 **WebAuthn** 支持验证器、浏览器和平台。确保您的验证器、浏览器和平台支持 **WebAuthn** 规格。

#### 8.7.1. 设置

对 **2FA** 的 **WebAuthn** 支持设置过程如下：

##### 8.7.1.1. 启用 WebAuthn authenticator 注册

1. 点菜单中的 **Authentication**。

2. 点 **Required Actions** 选项卡。

3. 将 **Webauthn Register** 开关切换到 **ON**。

*如果您希望所有新用户注册其 **WebAuthn** 凭据，请将 **Default Action** 开关切换为 **ON**。*

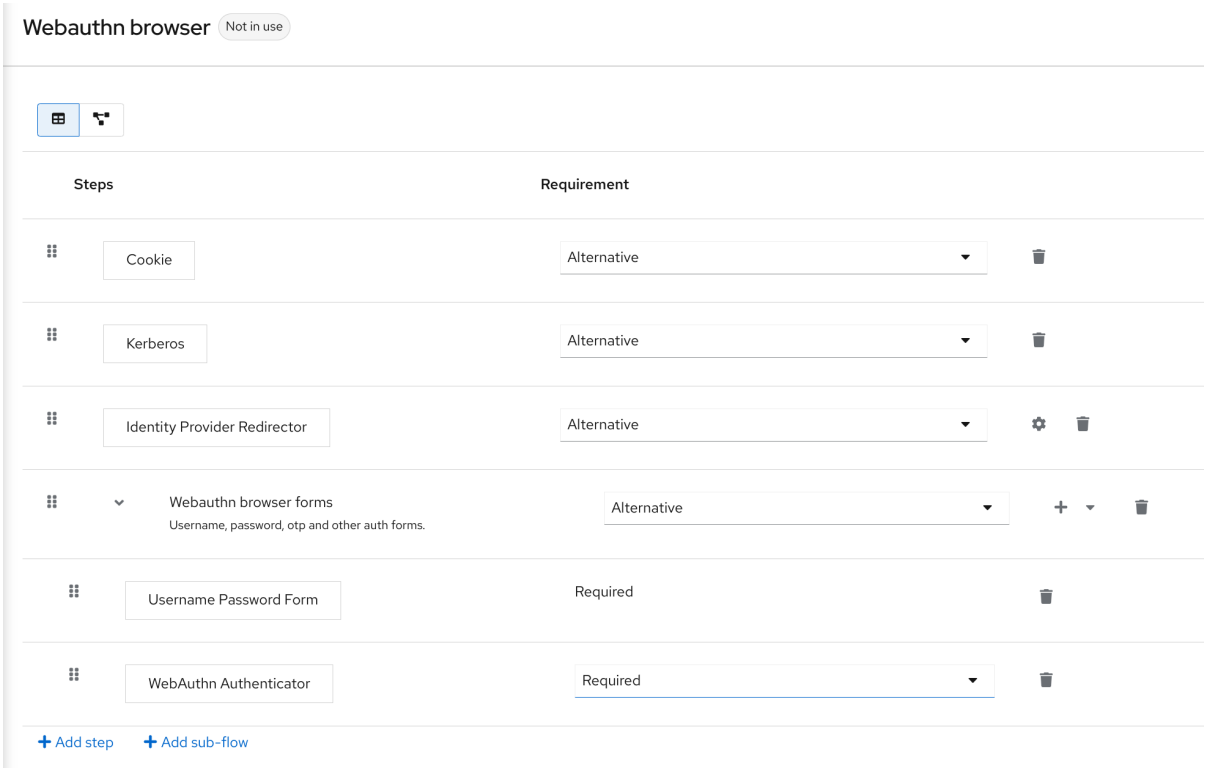
### *8.7.2. 在浏览器流中添加 WebAuthn 身份验证*

1. *点菜单中的 **Authentication**。*
2. *单击 浏览器 流。*
3. *从**"Action list"**中选择重复项，以复制内置 浏览器 流。*
4. *输入 **"WebAuthn Browser"** 作为副本的名称。*
5. *点 **Duplicate**。*
6. *点名称进入详情*
7. *单击**"WebAuthn Browser Browser"**的垃圾桶图标 **- Conditional OTP"**，然后单击 **Delete**。*

*如果所有用户需要 **WebAuthn** ：*

1. *单击 **WebAuthn Browser Forms** 的 **+** 菜单。*
2. *点 **Add step**。*
3. *点 **WebAuthn Authenticator**。*
4. *点击 **Add**。*

5. *选择 **WebAuthn Authenticator** 身份验证类型所需的，以设置其要求。*




6. *单击屏幕顶部的 **Action** 菜单。*

7. *从下拉列表中选择 **Bind flow**。*

8. *从下拉列表中选择 **Browser**。*

9. *点击 **Save**。*

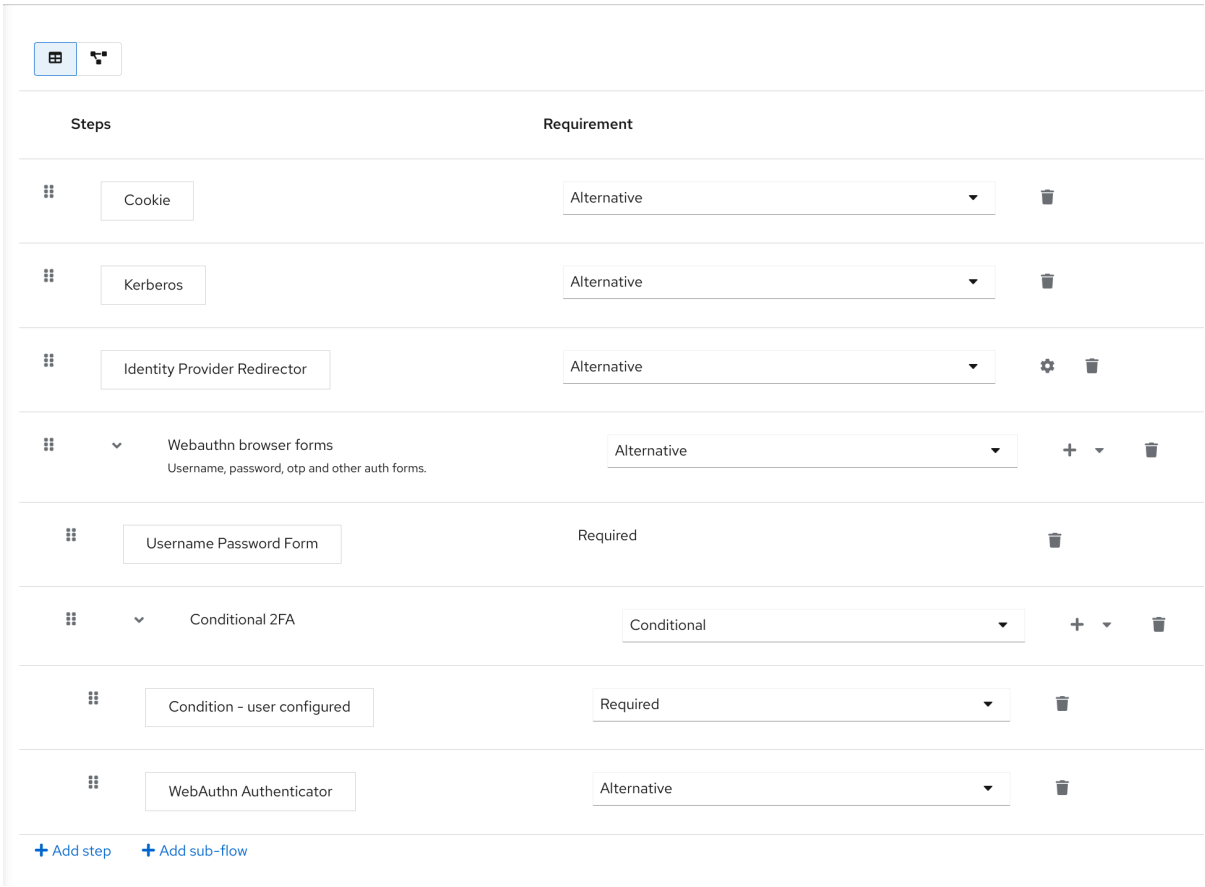
*注意*

*如果用户没有 **WebAuthn** 凭证，用户必须注册 **WebAuthn** 凭证。*

*如果用户只注册了 **WebAuthn** 凭证，则用户可以使用 **WebAuthn** 登录。因此，您可以：*

*流程*

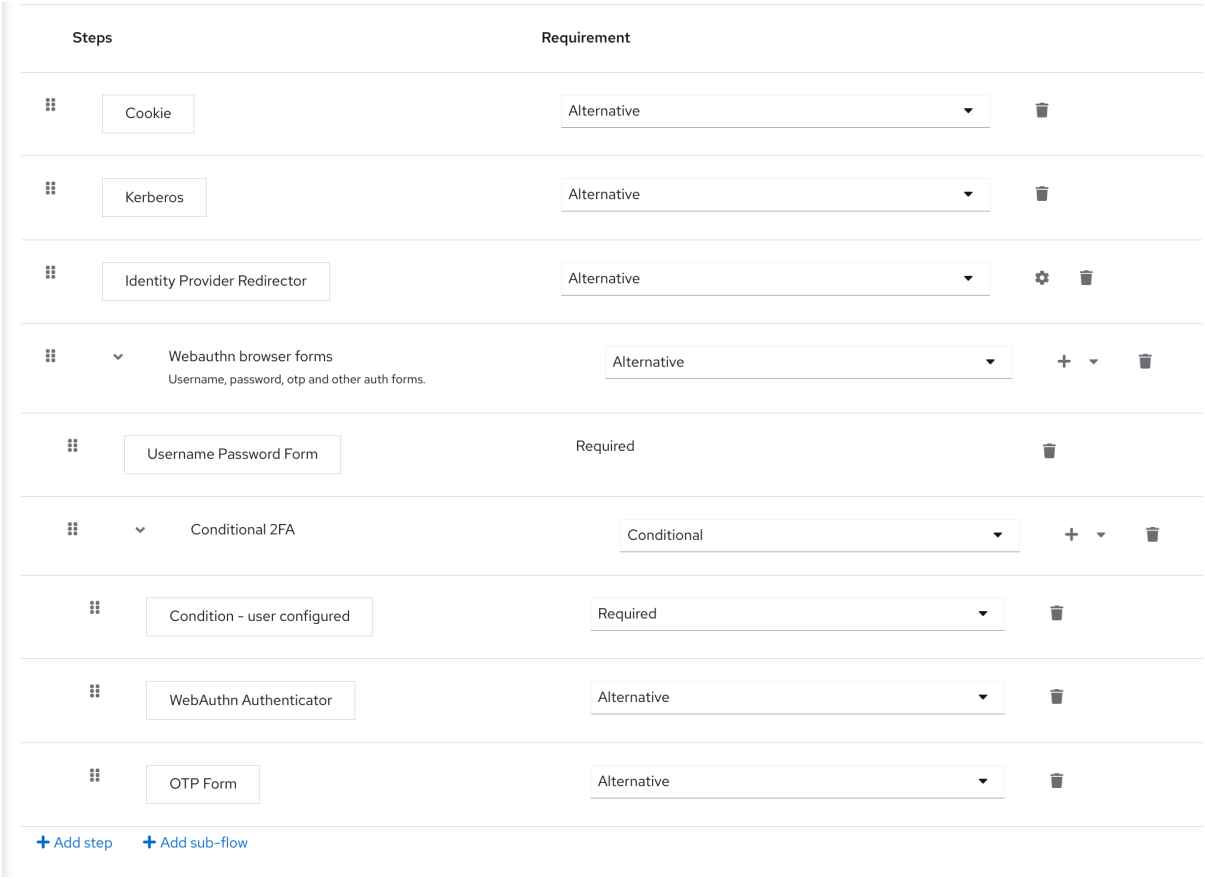
1. *单击 **WebAuthn Browser Forms** 行的 **+** 菜单。*
2. *点 **Add sub-flow**。*
3. *在 **name** 字段中输入 **"Conditional 2FA"**。*
4. *为 **Conditional 2FA** 选择 **Conditional** 条件，将其要求设置为条件。*
5. *在 **Conditional 2FA** 行中，单击加号 **+** 并选择 **Add condition**。*
6. *单击 **Add condition**。*
7. *选择 **Condition - User Configured**。*
8. *点击 **Add**。*
9. *选择 **Condition** 所需的 **- User Configured** 以将其要求设置为 **required**。*
10. *将 **WebAuthn Authenticator** 拖放到 条件 **2FA** 流*
11. *为 **WebAuthn Authenticator** 选择 替代方案，将其要求设置为替代方案。*




*对于第二个因素，用户可以在使用 **WebAuthn** 和 **OTP** 之间进行选择：*

*流程*

1. *在 **Conditional 2FA** 行中，单击加号 **+** 并选择 **Add step**。*
2. *从列表中选择 **OTP Form**。*
3. *点击 **Add**。*
4. *为 **OTP Form** 选择 **Alternative** 来设置它的替代要求。*




### *8.7.3. 使用 WebAuthn authenticator 进行身份验证*

*注册 **WebAuthn authenticator** 后，用户会执行以下操作：*

- *打开登录表单。用户必须使用用户名和密码进行身份验证。*
- *用户的浏览器要求用户使用其 **WebAuthn** 验证器进行身份验证。*

### *8.7.4. 以管理员身份管理 WebAuthn*

#### *8.7.4.1. 管理凭证*

***Red Hat build of Keycloak** 与用户凭证管理中的其他凭证类似 管理 **WebAuthn** 凭证 ：*

- ***Red Hat build of Keycloak** 为用户分配从 **Reset Actions** 列表中创建 **WebAuthn** 凭证所需的操作，然后选择 **Webauthn Register**。*

- *管理员可以点删除来删除 **WebAuthn** 凭据。*
- *管理员可以选择 **Show data...** 来查看凭据的数据，如 **AAGUID**。*
- *管理员可以通过在 **User Label** 字段中设置值并保存数据来为凭证设置标签。*

#### *8.7.4.2. 管理策略*

*管理员可以将 **WebAuthn** 相关的操作配置为每个域的 **WebAuthn Policy**。*

*流程*

1. *点菜单中的 **Authentication**。*
2. *点 **Policy** 选项卡。*
3. *点 **WebAuthn Policy** 选项卡。*
4. *在策略中配置项目（请参阅以下描述）。*
5. *点 **Save**。*

*可配置的项目及其描述如下：*

| Configuration | 描述 |
| --- | --- |
| 依赖的实体名称 | 可读的服务器名称作为 WebAuthn Relying party。此项目是必需的，适用于 WebAuthn 验证器的注册。默认设置为 "keycloak"。如需了解更多详细信息，请参阅 WebAuthn 规格。 |

| Configuration | 描述 |
| --- | --- |
| 签名算法 | 该算法告知 WebAuthn 验证器用于 公钥 凭据的签名算法。红帽构建的 Keycloak 使用公钥凭证来签名和验证身份验证。如果没有算法，则默认 ES256 被调整。ES256 是一个可选配置项，适用于 WebAuthn 身份验证器注册。如需了解更多详细信息，请参阅 WebAuthn 规格。 |
| 依赖第三方 ID | WebAuthn 重写的 ID，用于决定 公钥凭证 的范围。ID 必须是原始的有效域。此 ID 是应用于 WebAuthn 身份验证器的可选配置项。如果此条目为空，红帽构建的 Keycloak 会适应红帽构建的 Keycloak 基本 URL 的主机部分。如需了解更多详细信息，请参阅 WebAuthn 规格。 |
| attestation Conveyance Preference | 浏览器上的 WebAuthn API 实现(WebAuthn Client)是生成 Attestation 语句的优先方法。此首选项是一个可选配置项，适用于 WebAuthn 验证器的注册。如果没有选项，其行为与选择 "none" 的行为相同。如需了解更多详细信息，请参阅 WebAuthn 规格。 |
| 身份验证器附加 | WebAuthn 客户端的 WebAuthn authenticator 可接受的附加模式。这个模式是一个可选配置项，适用于 WebAuthn 验证器的注册。如需了解更多详细信息，请参阅 WebAuthn 规格。 |
| 需要可发现的凭证 | 要求 WebAuthn 验证器的选项将生成公钥凭据 作为客户端发现的凭据。这个选项适用于 WebAuthn 验证器的注册。如果留空，其行为与选择"No"的行为相同。如需了解更多详细信息，请参阅 WebAuthn 规格。 |
| 用户身份验证要求 | 要求 WebAuthn authenticator 选项确认用户的验证。这是一个可选配置项，适用于 WebAuthn 身份验证器的注册，以及用户通过 WebAuthn authenticator 进行身份验证。如果没有选项，其行为与选择"首选"的行为相同。如需了解更多详细信息，请参阅 用于注册 WebAuthn 验证器和 WebAuthn 规范的 WebAuthn 规范，以通过 WebAuthn 验证器验证用户。 |
| Timeout（超时） | 注册 WebAuthn 身份验证器并使用 WebAuthn 身份验证器验证用户的超时值（以秒为单位）。如果设置为零，则其行为取决于 WebAuthn 验证器的实现。默认值为 0。如需了解更多详细信息，请参阅 用于注册 WebAuthn 验证器和 WebAuthn 规范的 WebAuthn 规范，以通过 WebAuthn 验证器验证用户。 |
| 避免相同的身份验证器注册 | 如果启用，红帽构建的 Keycloak 无法重新注册已注册的 WebAuthn 验证器。 |

| Configuration | 描述 |
| --- | --- |
| 可接受的 AAGUID | WebAuthn 验证器必须注册的 AAGUID 的白板列表。 |

#### *8.7.5. attestation 语句验证*

*在注册 **WebAuthn authenticator** 时，红帽构建的 **Keycloak** 会验证由 **WebAuthn authenticator** 生成的 **attestation** 语句的可信度。**Red Hat build of Keycloak** 需要信任 **anchor** 的证书导入到 信任存储中。*

*要省略此验证，请禁用此信任存储，或将 **WebAuthn** 策略的配置项 **"Attestation Conveyance Preference"** 设置为 **"none"**。*

### *8.7.6. 以用户身份管理 WebAuthn 凭证*

#### *8.7.6.1. 注册 WebAuthn authenticator*

*注册 **WebAuthn authenticator** 的适当方法取决于用户在红帽构建的 **Keycloak** 上已注册了帐户。*

#### *8.7.6.2. 新用户*

*如果 **WebAuthn Register required** 操作在域中是 **Default Action**，则新用户必须在首次登录后设置 **Passkey**。*

*流程*

1. *打开登录表单。*
2. *点 **Register**。*
3. *填写表单上的项目。*
4. *点 **Register**。*

*成功注册后，浏览器要求用户输入其 **WebAuthn authenticator's** 标签的文本。*

#### *8.7.6.3. 现有用户*

*如果根据第一个示例所示设置 **WebAuthn Authenticator**，那么当现有用户尝试登录时，需要他们自动注册其 **WebAuthn** 验证器：*

*流程*

1. *打开登录表单。*
2. *输入表单上的项目。*
3. *点 **Save**。*
4. *单击 **Login**。*

*注册成功后，用户的浏览器会要求用户输入其 **WebAuthn** 验证器标签的文本。*

### *8.7.7. 免密码 WebAuthn 与 Two-Factor*

***Red Hat build of Keycloak** 使用 **WebAuthn** 进行双因素身份验证，但您可以使用 **WebAuthn** 作为第一因素身份验证。在这种情况下，具有 免密码 **WebAuthn** 凭证的用户可以在没有密码的情况下向红帽构建的 **Keycloak** 进行身份验证。红帽构建的 **Keycloak** 可以使用 **WebAuthn** 作为域上下文中的免密码和双因素身份验证机制，以及单个身份验证流。*

*管理员通常要求用户为 **WebAuthn** 免密码身份验证注册 **Passkeys** 满足不同的要求。例如，**Passkeys** 可能需要用户使用 **PIN** 对 **Passkey** 进行身份验证，或使用更强大的证书颁发机构进行 **Passkey attests**。*

*因此，红帽构建的 **Keycloak** 允许管理员配置单独的 **WebAuthn Passwordless Policy**。必需的 **Webauthn Register Passwordless action of type and separate authenticator of type WebAuthn Passwordless Authenticator.***

#### *8.7.7.1. 设置*

*设置 **WebAuthn** 免密码支持，如下所示：*

1. （如果不存在），为 **WebAuthn** 免密码支持注册一个新的所需操作。使用 **Enable WebAuthn Authenticator Registration** 中所述的步骤。注册 **Webauthn Register Passwordless** 操作。

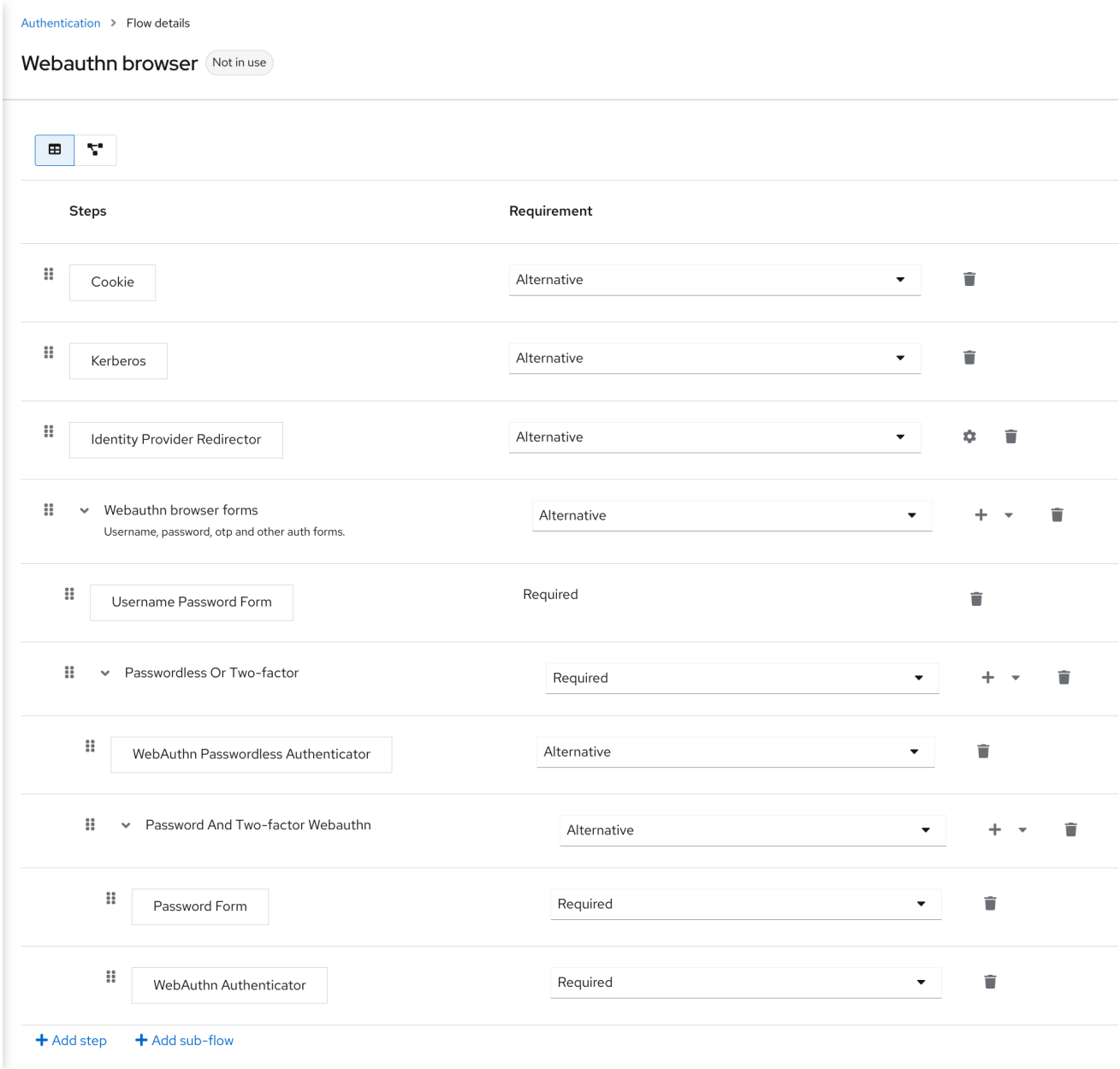
2. 配置策略。您可以使用管理策略 中描述的步骤和 配置选项。在 **WebAuthn Passwordless Policy** 选项卡中执行管理控制台中的配置。通常，**Passkey** 的要求比双因素策略更强。例如，您可以在配置免密码策略时将 **User Verification Requirement** 设置为 **Required**。

3. 配置身份验证流。使用 **WebAuthn Browser** 流，如 **Adding WebAuthn Authentication to a Browser Flow** 所述。配置流，如下所示：

   - **WebAuthn Browser Forms** 子流包含 **Username Form** 作为第一个验证器。删除默认 **Username Password Form authenticator**，再添加 **Username Form authenticator**。此操作要求用户提供用户名作为第一步。
   - 将有一个必需的子流，它可以命名为 **Passwordless Or Two-factor**，例如：此子流表示用户可以使用 **Passwordless WebAuthn** 凭证或双因素身份验证。
   - 流包含 **WebAuthn Passwordless Authenticator** 作为第一个替代方案。
   - 第二个替代方案将是名为 **Password and two-factor Webauthn** 的子流，例如：此子流包含 **Password Form** 和 **WebAuthn Authenticator**。

流的最终配置类似如下：

免密码流




*现在，您可以将 **WebAuthn Register Passwordless** 作为用户（已已知的 **Keycloak** ）添加必要操作来测试这一点。在第一次身份验证过程中，用户必须使用密码和第二因素 **WebAuthn** 凭证。如果使用 **WebAuthn Passwordless** 凭证，用户不需要提供密码和第二因素 **WebAuthn** 凭证。*

### *8.7.8. LoginLess WebAuthn*

***Red Hat build of Keycloak** 使用 **WebAuthn** 进行双因素身份验证，但您可以使用 **WebAuthn** 作为第一因素身份验证。在这种情况下，具有 免密码 **WebAuthn** 凭证的用户可以在不提交登录或密码的情况下向红帽构建的 **Keycloak** 进行身份验证。**Red Hat build of Keycloak** 可以使用 **WebAuthn** 作为域上下文中的无登录/密码以及双因素身份验证机制。*

*管理员通常要求用户为 **WebAuthn** 登录身份验证注册的 **Passkeys** 满足不同的要求。无登录身份验证要求用户向 **Passkey** 进行身份验证（例如，使用 **PIN** 代码或指纹），并且与无登录凭证关联的加密密钥*

*实际上存储在 **Passkey** 上。并非所有 **Passkeys** 都满足这一要求。如果您的设备支持 **'user verification'** 和 **'discoverable credential'**，则使用您的 **Passkey vendor** 进行检查。请参阅支持的 **Passkeys**。*

*红帽构建的 **Keycloak** 允许管理员以允许 无登录身份验证的方式配置 **WebAuthn** 无密码策略。请注意，无登录身份验证只能使用 **WebAuthn Passwordless Policy** 和 **WebAuthn Passwordless** 凭证进行配置。**Webauthn** 登录身份验证和 **WebAuthn** 免密码身份验证可以在同一域中配置，但将共享相同的策略 **WebAuthn Passwordless Policy**。*

#### *8.7.8.1. 设置*

*流程*

*设置 **WebAuthn** 无登录支持，如下所示：*

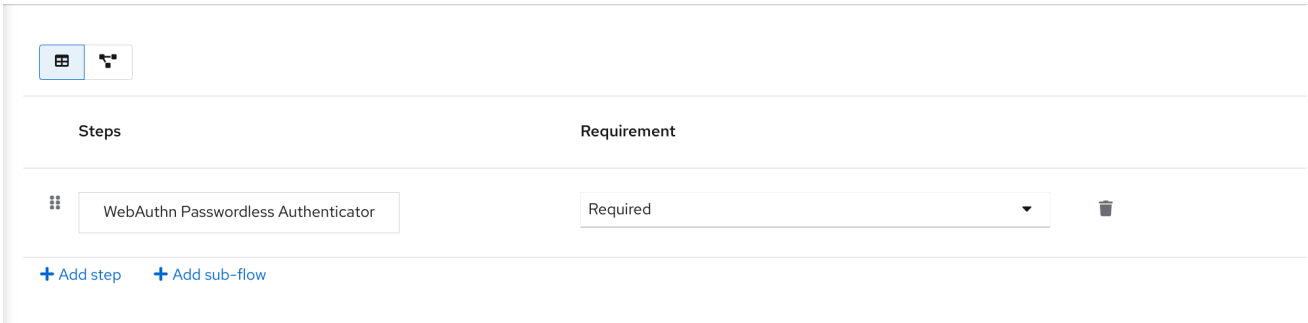
1. *（如果不存在），为 **WebAuthn** 免密码支持注册一个新的所需操作。使用 **Enable WebAuthn Authenticator Registration** 中所述的步骤。注册 **Webauthn Register Passwordless** 操作。*

2. *配置 **WebAuthn** 无密码策略。在 **Admin Console, Authentication** 部分中执行配置，在 **Policies → WebAuthn Passwordless Policy** 选项卡中执行。当您为无登录场景配置策略时，您必须将 **User Verification Requirement** 设置为 **required** 和 **Require Discoverable Credential to Yes**。请注意，由于没有专用的无登录策略，因此无法使用用户验证 **=no/discoverable credential=no** 和无登录场景（用户验证=**yes/discoverable credential=yes**）混合身份验证场景。存储容量通常对 **Passkeys** 非常有限，这意味着您无法将许多可发现的凭证存储在您的 **Passkey** 中。*

3. *配置身份验证流。创建新的身份验证流，添加 **"WebAuthn Passwordless"** 执行，并将执行的 **Requirement** 设置设置为 **Required***

*流的最终配置类似如下：*

***LoginLess** 流*




*现在，您可以将所需的操作 **WebAuthn Register Passwordless** 添加到一个用户（已在红帽构建的 **Keycloak** 中已知）以进行测试。配置有所需操作的用户必须进行身份验证（例如使用用户名/密码），然后提示您注册用于无登录身份验证的 **Passkey**。*

#### *8.7.8.2. 特定于供应商的标记*

##### *8.7.8.2.1. 兼容性检查列表*

*使用红帽构建的 **Keycloak** 进行无登录身份验证需要 **Passkey** 才能满足以下功能*

- *__FIDO2__ 合规性：不要与 **FIDO/U2F** 混淆*
- *用户验证： **Passkey** 能够验证用户（防止发现您的 **Passkey** 用户能够验证无登录和免密码）*
- *可发现的凭证： **Passkey** 能够存储与客户端应用程序关联的登录和加密密钥*

##### *8.7.8.2.2. Windows Hello*

*要使用基于 **Windows Hello** 的凭证来针对红帽构建的 **Keycloak** 进行身份验证，请将 **WebAuthn Passwordless Policy** 的 **Signature Algorithms** 设置配置为包含 **RS256** 值。请注意，一些浏览器不允许访问私有窗口内的平台 **Passkey**（如 **Windows Hello**）。*

##### *8.7.8.2.3. 支持的 Passkeys*

*以下 **Passkeys** 已通过红帽构建的 **Keycloak** 成功测试进行无登录身份验证：*

- *Windows Hello (Windows 10 21H1/21H2)*
- *yubico Yubikey 5 NFC*
- *Feitian ePass FIDO-NFC*

### *8.8. 恢复代码(RECOVERYCODES)*

*您可以通过在身份验证流中添加 **'Recovery Authentication Code Form'** 作为双因素验证器，为双因素身份验证程序配置恢复代码。有关配置此验证器的示例，请参阅 WebAuthn。*

*注意*

***RecoveryCodes** 是 技术预览，不被支持。此功能默认为禁用。*

*使用 **--features=preview** 或 **--features=recovery-codes**启动服务器*

### *8.9. 条件流中的条件*

*如 执行要求 中所述，**Condition** 执行只能包含在 条件 子流中。如果所有条件执行都评估为 **true**，则 **Conditional** 子流将充当 **Required**。您可以在 **Conditional** 子流中处理下一个执行。如果 **Conditional** 子流中包含的某些执行评估为 **false**，则整个子流将被视为 **Disabled**。*

#### *8.9.1. 可用条件*

***condition -** 用户角色*

*此执行能够确定用户是否有由 **User role** 字段定义的角色。如果用户具有所需的角色，则执行被视为 **true**，并评估其他执行。管理员必须定义以下字段：*

***Alias***

*描述执行的名称，该名称将显示在身份验证流中。*

*用户角色*

*用户应该执行此流的角色。要指定应用角色，语法为 **appname.approle** （如 **myapp.myrole**）。*

**condition -** *用户配置*

*这将检查是否为用户配置了流中的其他执行。* **Execution requirements** *部分包含* **OTP** *表单的示例。*

**condition -** *用户属性*

*这将检查用户是否已设置所需的属性：（可选）检查也可以评估组属性。可能需要对输出进行求值，这意味着用户不应具有属性。* **User Attributes** *部分演示了如何添加自定义属性。您可以提供这些字段：*

**Alias**

*描述执行的名称，该名称将显示在身份验证流中。*

*属性名称*

*要检查的属性的名称。*

*预期属性值*

*属性中预期的值。*

*包含组属性*

*如果* **On**，*条件检查任何加入组是否具有与配置的名称和值匹配的一个属性：此选项可能会影响性能*

**negate** *输出*

*您可以对输出进行求反。换句话说，属性不应存在。*

### 8.9.2. 在条件流中明确拒绝/允许访问

*您可以允许或拒绝访问条件流中的资源。两个 验证器 拒绝访问， 并允许根据条件访问资源。*

*允许访问*

*身份验证器将始终成功进行身份验证。此验证器不可配置。*

*拒绝访问*

*访问将始终被拒绝。您可以定义错误消息，它将向用户显示。您可以提供这些字段：*

***Alias***

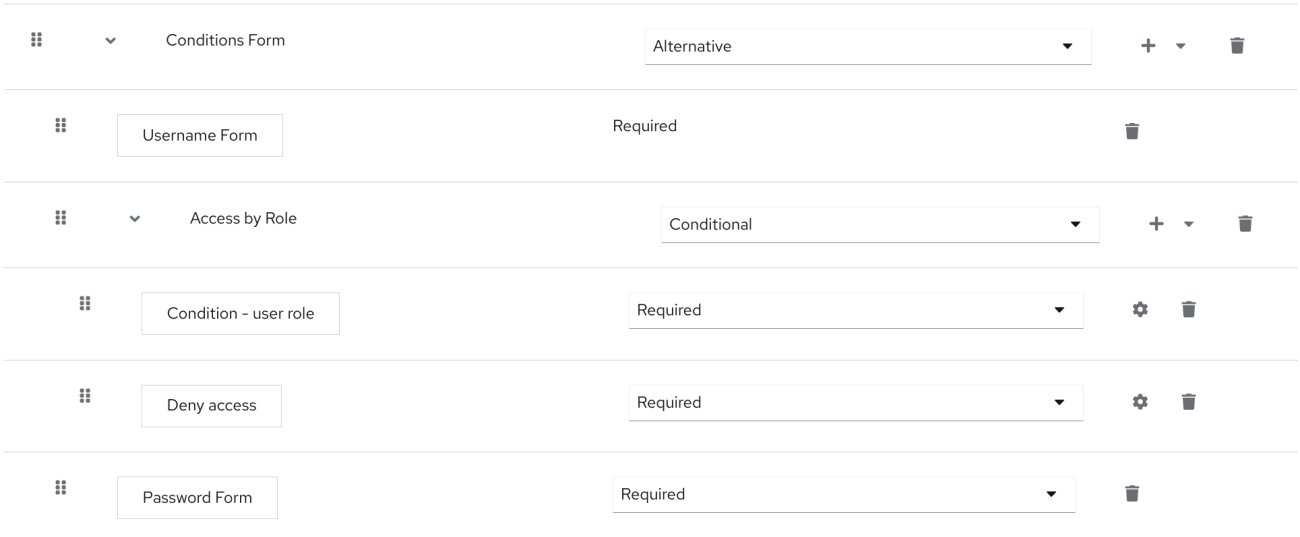
*描述执行的名称，该名称将显示在身份验证流中。*

*错误消息*

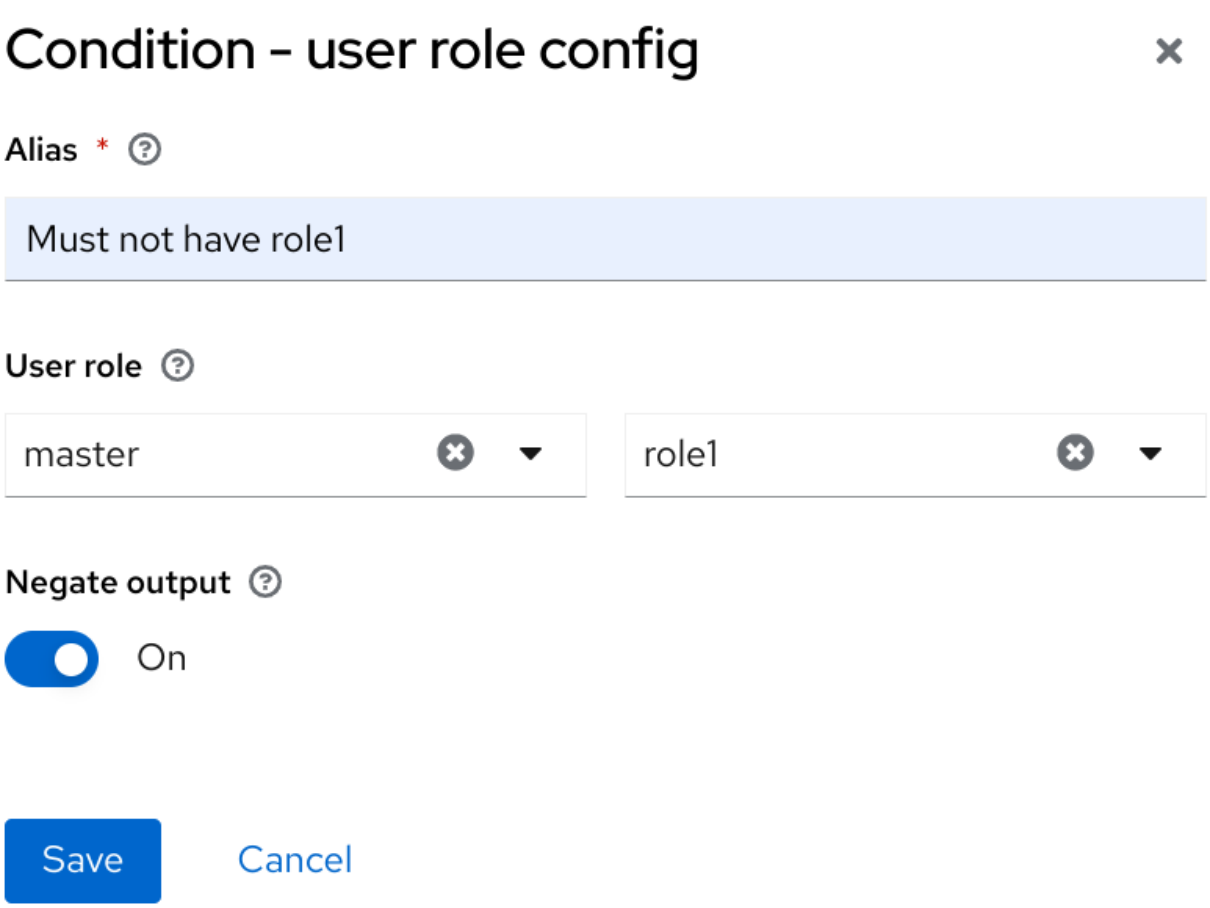
*向用户显示的错误消息。错误消息可以作为特定消息或属性提供，以便将其与本地化（例如."您没有角色 **'admin'**。在消息属性中，**my-property-deny** 会为默认消息留空，定义为属性 **access-denied**。*

*以下是如何拒绝对没有角色 **role1** 的所有用户的访问，并显示由属性 **deny-role1** 定义的错误消息。这个示例包括 **Condition - User Role** 和 **Deny Access executions**。*

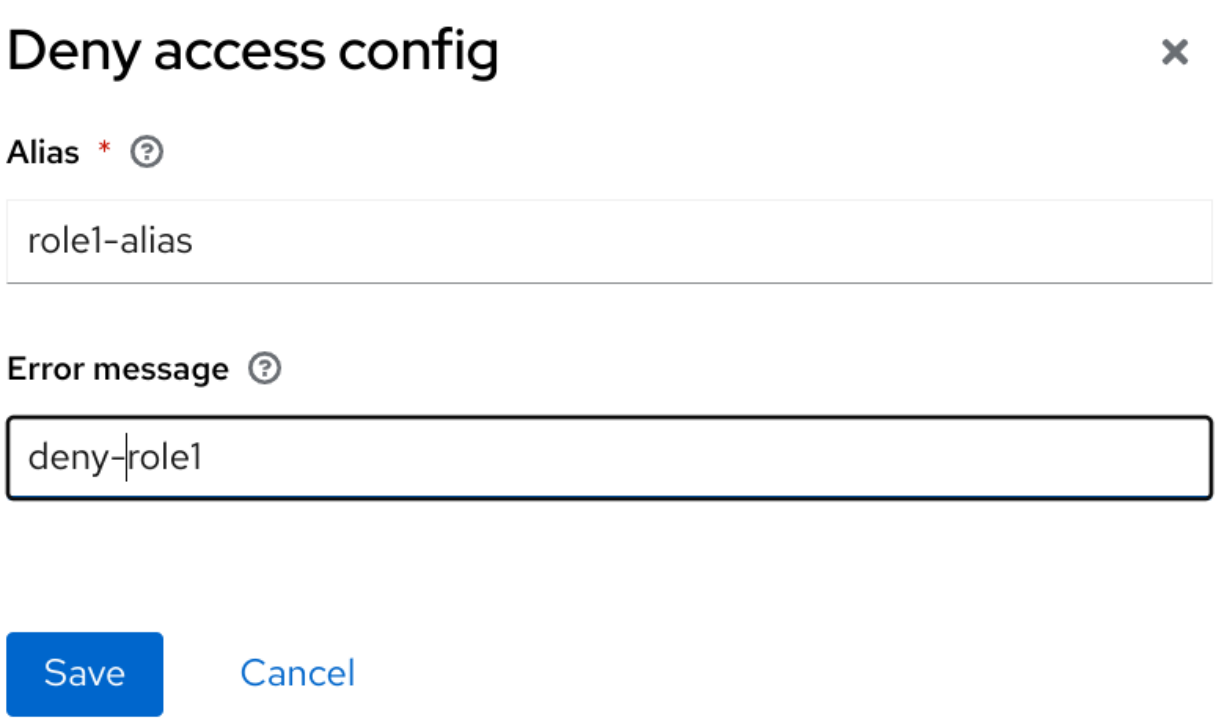
*浏览器流*




***condition -** 用户角色配置*




*配置 **Deny Access** 非常简单。您可以指定任意 **Alias** 和所需消息，如下所示：*




*最后的内容在 **login theme messages_en.properties** （用于英语）中定义带有错误消息的属性：*

```
deny-role1 = You do not have required role!
```

### *8.10. PASSKEYS*

*红帽构建的 **Keycloak** 为 **Passkeys** 提供预览支持。红帽构建的 **Keycloak** 充当 **Passkeys Relying party (RP)**。*

***Passkey** 注册和验证是由 **WebAuthn** 的功能实现的。因此，红帽构建的 **Keycloak** 用户可以通过现有 **WebAuthn** 注册和验证。*

*注意*

*同步的 **Passkeys** 和 **device-bound Passkeys** 都可用于 **Same-Device** 和 **Cross-Device Authentication (CDA)**。但是，**Passkeys** 操作成功取决于用户的环境。确保在 环境中 哪些操作可以成功。*

# 第 9 章 集成身份提供程序

**Identity Broker** 是一个中间服务，可将服务提供商与身份提供程序连接。身份代理创建与外部身份提供程序的关系，以使用提供程序的身份访问服务提供商公开的内部服务。

从用户的角度来看，身份代理提供了一种以用户为中心的、集中的方式来管理安全域和域的身份。您可以使用身份提供程序中的一个或多个身份链接帐户，或者根据其身份信息创建帐户。

身份提供程序从用于验证和向用户发送身份验证和授权信息的特定协议派生。它可以是：

- 社交供应商，如 **Facebook**、**Google** 或 **Twitter**。
- 用户需要访问您的服务的业务合作伙伴。
- 要集成的基于云的身份服务。

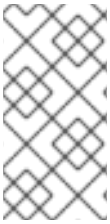
通常，红帽在以下协议上构建 **Keycloak** 基础供应商：

- **SAML v2.0**
- **OpenID Connect v1.0**
- **OAuth v2.0**

## 9.1. BROKERING 概述

当使用红帽构建的 **Keycloak** 作为身份代理时，红帽构建的 **Keycloak** 不会强制用户提供其凭证以便在特定域中进行身份验证。**Red Hat build of Keycloak** 显示可以进行身份验证的身份提供程序列表。

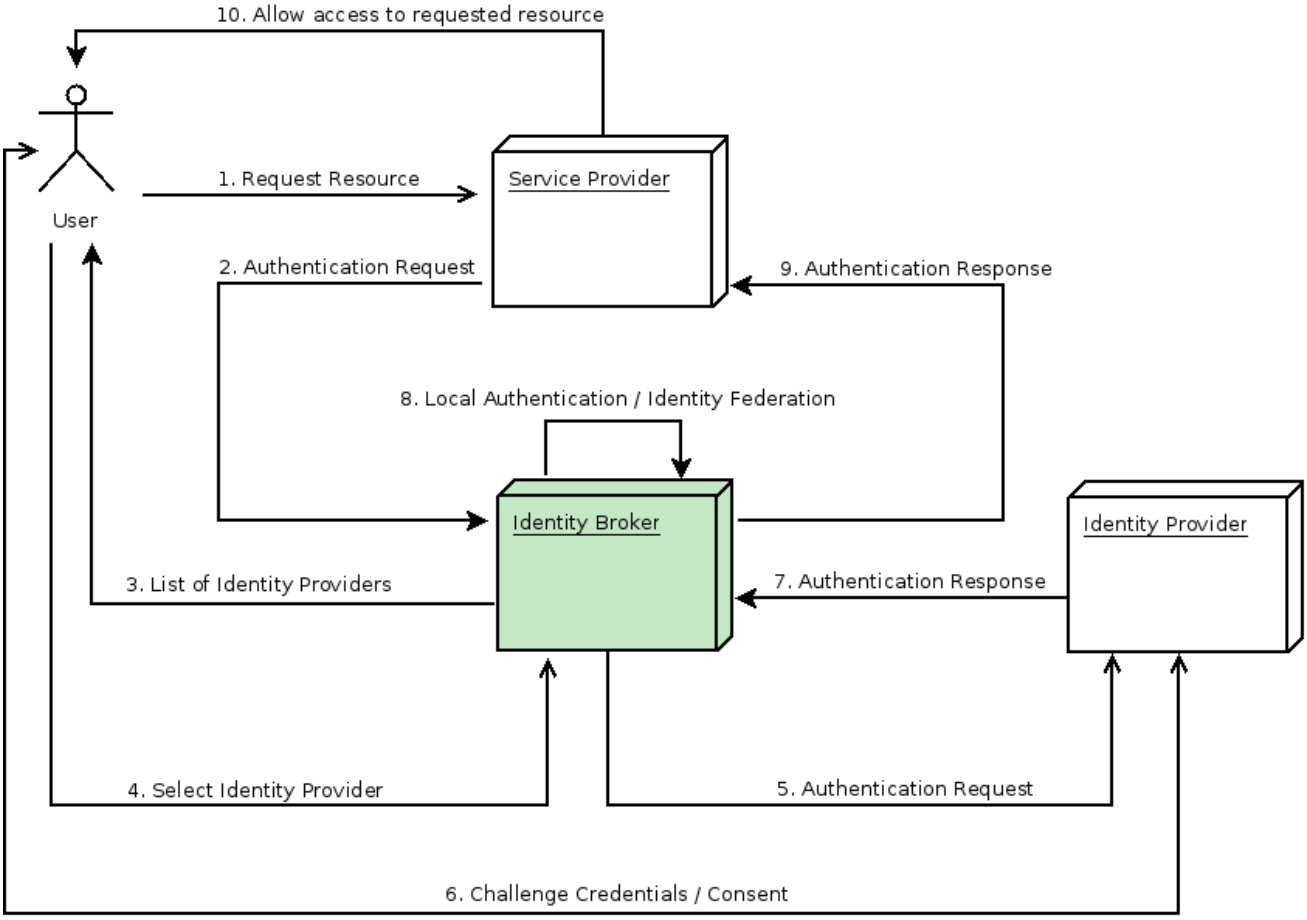
如果您配置默认身份提供程序，红帽构建的 **Keycloak** 会将用户重定向到默认供应商。



*注意*

*不同的协议可能需要不同的身份验证流。红帽构建的 **Keycloak** 支持的所有身份提供程序都使用以下流。*

*身份代理流*




1. *未经身份验证的用户在客户端应用程序中请求受保护的资源。*

2. *客户端应用程序将用户重定向到红帽构建的 **Keycloak** 进行验证。*

3. ***Red Hat build of Keycloak** 显示登录页面，其中包含在域中配置的身份提供程序列表。*

4. *用户通过单击其按钮或链接来选择其中一个身份提供程序。*

5.

*红帽构建的 **Keycloak** 会向请求身份验证的目标身份提供程序发出身份验证请求，并将用户重定向到身份提供程序的登录页面。管理员已经为管理控制台身份提供程序设置了连接属性和其他配置选项。*

6. *用户提供凭证或同意与身份提供程序进行身份验证。*

7. *当身份提供程序成功身份验证后，用户将使用身份验证响应重定向到红帽构建的 **Keycloak**。通常，响应包含由红帽构建的 **Keycloak** 用来信任身份提供程序的身份验证并检索用户信息的安全令牌。*

8. *红帽构建的 **Keycloak** 会检查来自身份提供程序的响应是否有效。如果有效，红帽构建的 **Keycloak** 导入，并在用户尚不存在时创建用户。如果令牌不包含该信息，红帽构建的 **Keycloak** 可能会询问身份提供程序以获取进一步的用户信息。此行为是 身份联邦。如果用户已存在，红帽构建的 **Keycloak** 可能会要求用户将现有帐户从身份提供程序返回的身份链接。此行为是 链接 的帐户。使用红帽构建的 **Keycloak**，您可以配置 帐户链接，并在第一个 登录流中指定它。在这一步中，红帽构建的 **Keycloak** 会验证用户并发出其令牌来访问服务提供商中请求的资源。*

9. *当用户验证时，**Red Hat build of Keycloak** 会将用户重定向到服务提供商，方法是在本地身份验证过程中发送之前发布的令牌。*

10. *服务供应商从红帽构建的 **Keycloak** 接收令牌，并允许访问受保护的资源。*

*此流的变体是可能的。例如，客户端应用程序可以请求特定的身份提供程序而不是显示它们列表，或者您可以设置 **Keycloak** 的红帽构建，以便在迭代身份前强制用户提供更多信息。*

*在身份验证过程结束时，红帽构建的 **Keycloak** 会将其令牌发送到客户端应用程序。客户端应用程序与外部身份提供程序分开，因此无法看到客户端应用程序的协议或它们如何验证用户身份。供应商只需要了解红帽构建的 **Keycloak**。*

### *9.2. 默认身份提供程序*

*红帽构建的 **Keycloak** 可以重定向到身份提供程序，而不是显示登录表单。启用此重定向：*

*流程*

1. *点菜单中的 **Authentication**。*

2. 单击 浏览器 流。

3. 点 **Identity Provider Redirector** 行上的齿轮图标。

4. 将 **Default Identity Provider** 设置为您要将用户重定向到的身份提供程序。

如果红帽构建的 **Keycloak** 找不到配置的默认身份提供程序，则会显示登录表单。

此验证器负责处理 **kc_idp_hint** 查询参数。如需更多信息，请参阅客户端推荐的身份提供程序 部分。

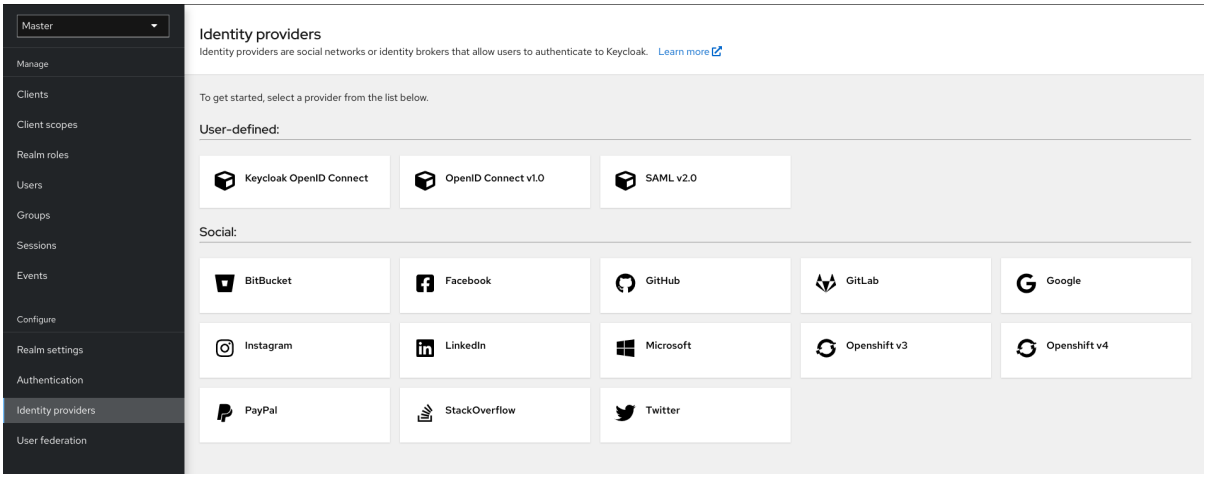
### 9.3. 常规配置

身份代理配置的基础是身份提供程序(IDP)。红帽构建的 **Keycloak** 为每个域创建身份提供程序，并默认为每个应用程序启用它们。当向应用程序签名时，域中的用户可以使用任何注册的身份提供程序。

流程

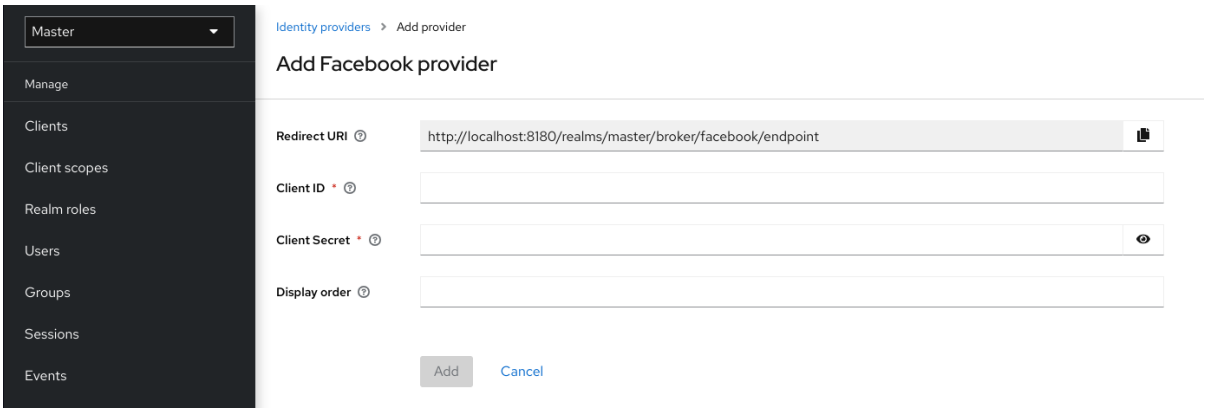
1. 单击菜单中的 **Identity Providers**。

身份供应商



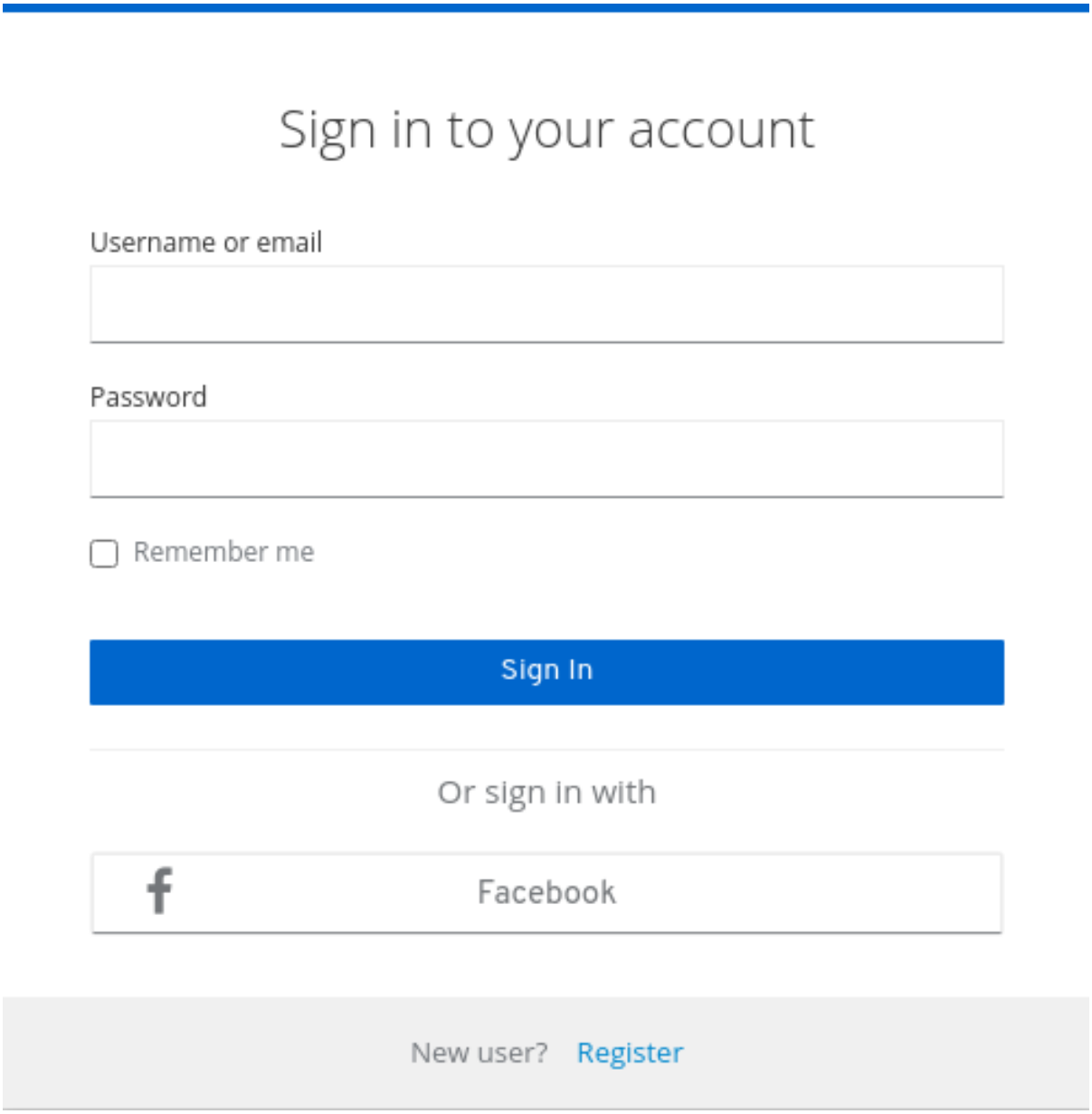

2. 选择身份提供程序。**Red Hat build of Keycloak** 显示您选择的身份提供程序的配置页面。

*添加 **Facebook** 身份提供程序*




*当您配置身份提供程序时，身份提供程序会出现在红帽构建的 **Keycloak** 登录页面中作为选项。您可以将自定义图标放在每个身份提供程序的登录屏幕上。如需更多信息，请参阅自定义图标。*

***IDP** 登录页面*




*社交*

*社交供应商在您的域中启用社交身份验证。通过红帽构建的 **Keycloak**，用户可以使用社交网络帐户登录到您的应用程序。支持的供应商包括 **Twitter**、**Facebook**、**Google**、**LinkedIn**、**Instagram**、**Microsoft**、**Pal**、**Openshift v3**、**GitLab**、**Bitbucket** 和 **Stack Overflow**。*

*基于协议*

*基于协议的供应商依赖特定协议来验证和授权用户。使用这些供应商，您可以连接到符合特定协议的任何身份提供程序。红帽构建的 **Keycloak** 支持 **SAML v2.0** 和 **OpenID Connect v1.0** 协议。您可以根据这些开放标准配置和代理任何身份提供程序。*

*虽然每种类型的身份提供程序都有其配置选项，但所有共享一个通用配置。可用的配置选项：*

*表 **9.1.** 常见配置*

| Configuration | 描述 |
| --- | --- |
| Alias | alias 是身份提供程序的唯一标识符，并引用内部身份提供程序。Red Hat build of Keycloak 使用别名来为需要重定向 URI 或回调 URL 的 OpenID Connect 协议构建重定向 URI。所有身份提供程序都必须有一个别名。别名示例包括 **facebook**、**google** 和 **idp.acme.com**。 |
| Enabled | 切换供应商 ON 或 OFF。 |
| 在登录页中隐藏 | 当 **ON** 时，Red Hat build of Keycloak 不会在登录页面中将此提供程序显示为登录选项。客户端可以使用 URL 中的 'kc_idp_hint' 参数来请求此提供程序。 |
| 仅限客户链接 | **ON** 时，红帽构建的 Keycloak 将现有帐户与这个供应商相关联。此供应商无法登录，红帽构建的 Keycloak 不会在登录页面上将这个供应商显示为选项。 |
| 存储令牌 | 在 **ON** 时，红帽构建的 Keycloak 存储来自身份提供程序的令牌。 |
| 存储的令牌可读 | 在 **ON** 时，用户可以检索存储的身份提供程序令牌。此操作也适用于 *代理* 客户端级别的角色 *读取令牌*。 |
| 信任电子邮件 | **ON** 时，红帽构建的 Keycloak 信任电子邮件地址来自身份提供程序。如果域需要电子邮件验证，则从此身份提供程序登录的用户不需要执行电子邮件验证过程。 |
| GUI 顺序 | 登录页面中可用身份提供程序的排序顺序。 |
| 验证基本声明 | 当 **ON** 时，身份提供程序发布的 ID 令牌必须具有特定的声明，否则用户无法通过这个代理进行身份验证 |
| 基本声明 | 当 **验证基本** 声明为 **ON** 时，要过滤的 JWT 令牌声明的名称（匹配区分大小写） |
| 基本声明值 | 当 **验证基本** 声明为 **ON** 时，要匹配的 JWT 令牌声明的值（支持正则表达式格式） |
| 第一个登录流 | 当用户使用此身份提供程序第一次登录到 Keycloak 的红帽构建时，Red Hat build of Keycloak 会触发。 |
| 后登录流 | 当用户完成使用外部身份提供程序登录时，Red Hat build of Keycloak 会触发。 |

| Configuration | 描述 |
| --- | --- |
| 同步模式 | 通过映射程序从身份提供程序更新用户信息的策略。在选择 **旧的** 时，红帽构建的 Keycloak 会使用当前的行为。**导入** 不会更新用户数据，**并强制** 更新用户数据。如需更多信息，请参阅身份提供程序映射程序。 |

### *9.4. 社交身份提供程序*

*社交身份提供程序可将身份验证委托给受信任、尊重的社交介质帐户。红帽构建的 **Keycloak** 包括对社交网络的支持，如 **Google**、**Facebook**、**Twitter**、**GitHub**、**LinkedIn**、**Microsoft** 和 **Stack Overflow**。*

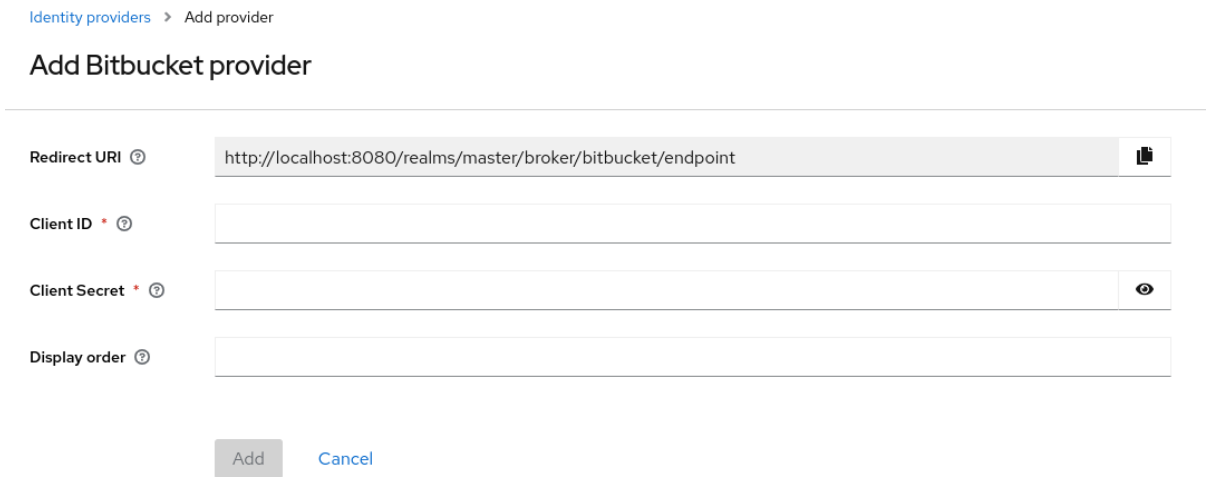
#### *9.4.1. Bitbucket*

*要使用 **Bitbucket** 登录，请执行以下步骤。*

*流程*

1. *单击菜单中的 **Identity Providers**。*
2. *从 **Add provider** 列表中，选择 **Bitbucket**。*

*添加身份提供程序*



3. *将 **Redirect URI** 的值复制到您的剪贴板。*

4. *在单独的浏览器选项卡中，在 **Bitbucket** 云进程上执行 **OAuth**。当您点 **Add Consumer** 时：*

   a. *将 **Redirect URI** 的值粘贴到 **Callback URL** 字段中。*

   b. *确保您在 帐户 部分中选择 **Email** 和 **Read**，以允许您的应用程序读取电子邮件。*

5. *请注意，在创建消费者时会显示 **Key** 和 **Secret** 值 **Bitbucket**。*

6. *在红帽构建的 **Keycloak** 中，将 **Key** 的值粘贴到 **Client ID** 字段中。*

7. *在红帽构建的 **Keycloak** 中，将 **Secret** 的值粘贴到 **Client Secret** 字段中。*

8. *点击 **Add**。*

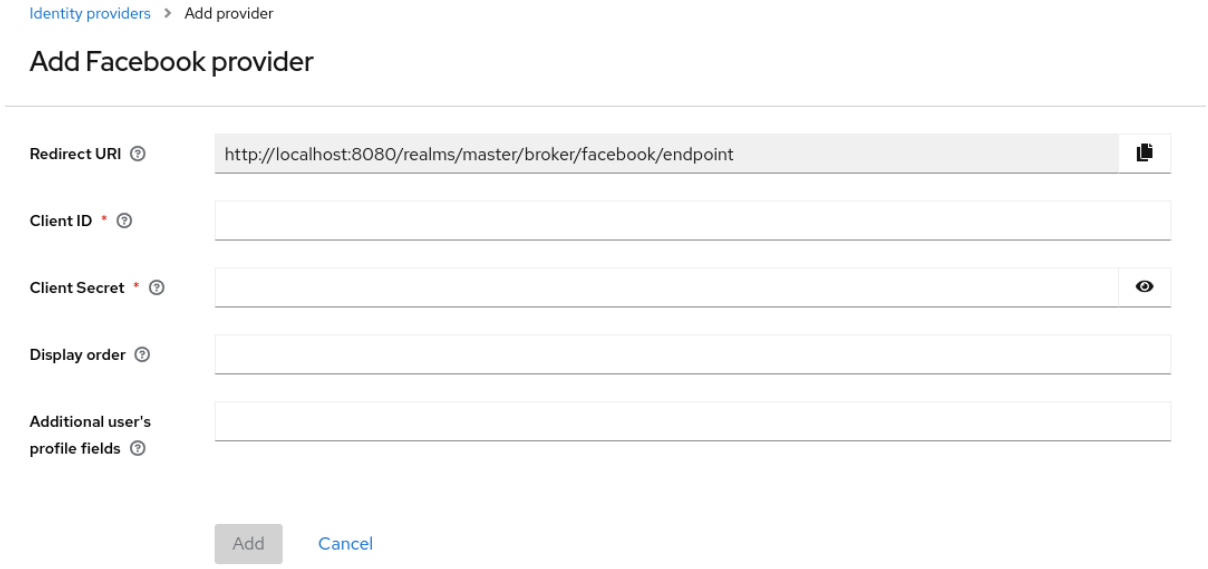
### *9.4.2. Facebook*

*流程*

1. *单击菜单中的 **Identity Providers**。*

2. *从 **Add provider** 列表中，选择 **Facebook**。*

*添加身份提供程序*



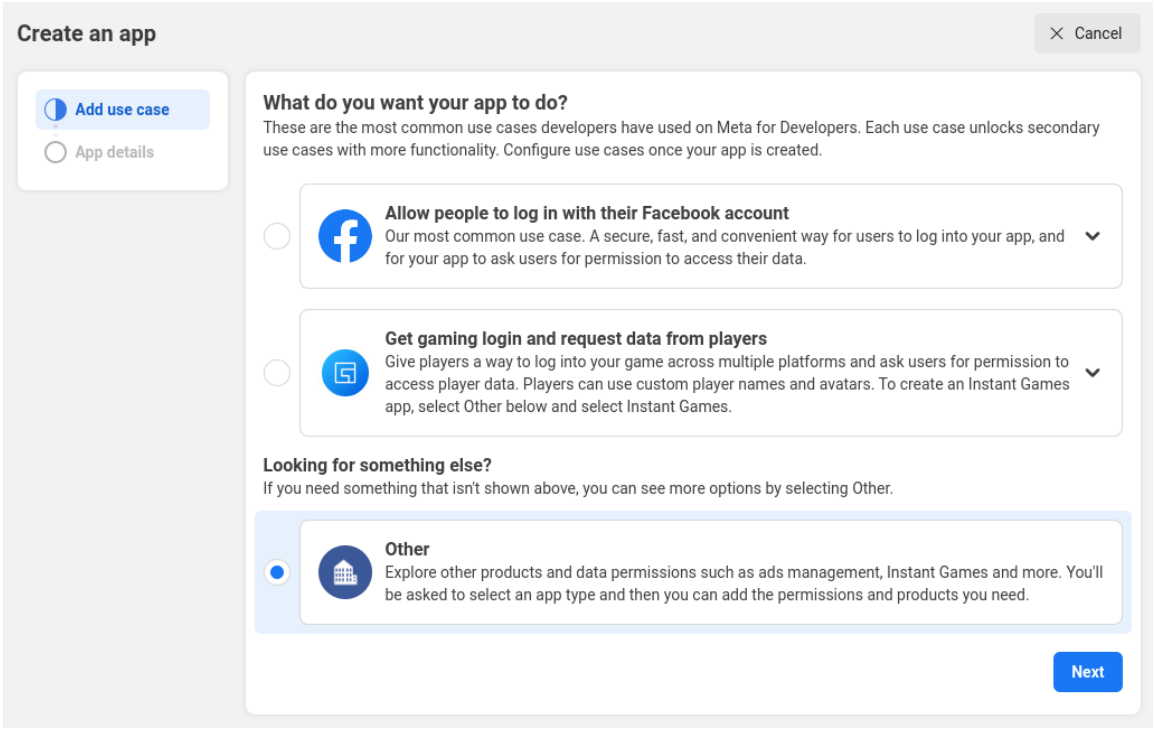

3. *将 **Redirect URI** 的值复制到您的剪贴板。*

4. *在单独的浏览器选项卡中，打开 **Developers** 的 **Meta**。*

   a. *单击 **My Apps**。*

   b. *选择 **Create App**。*

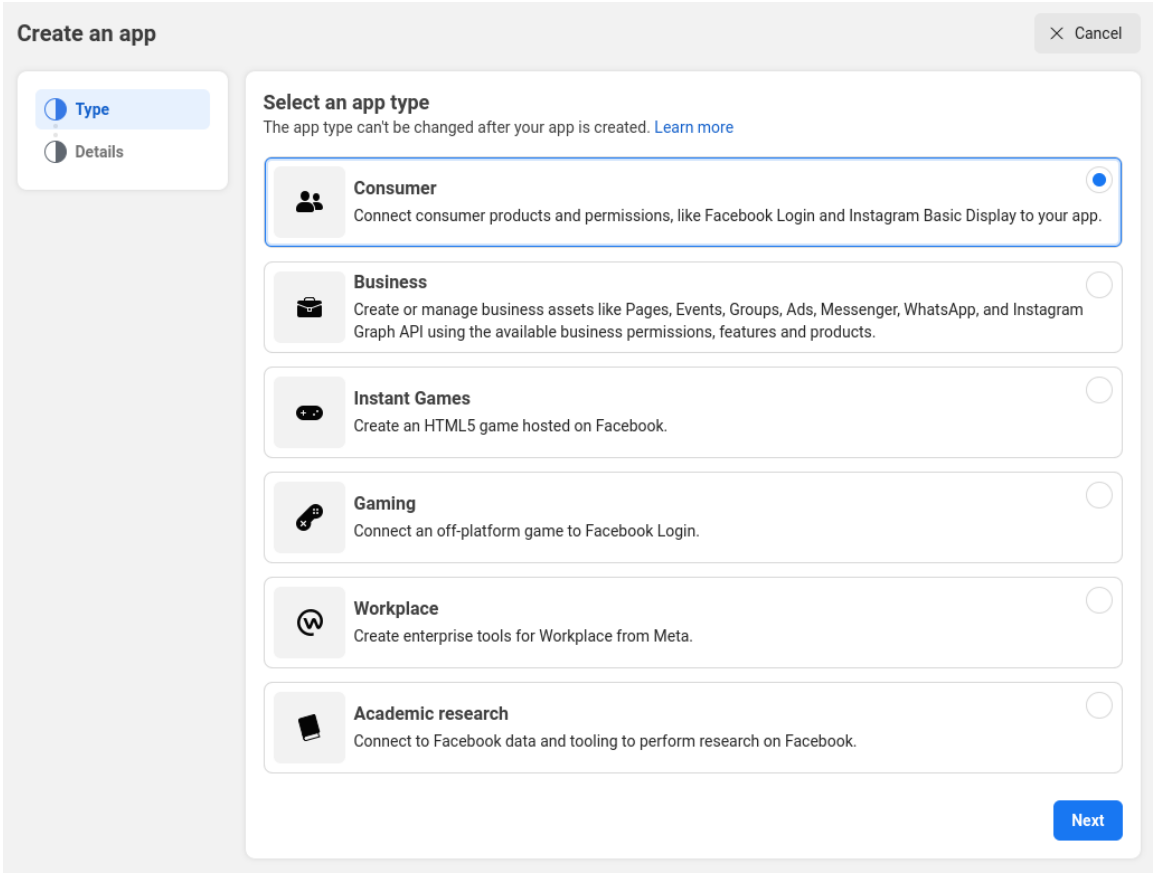
*添加用例*




c.

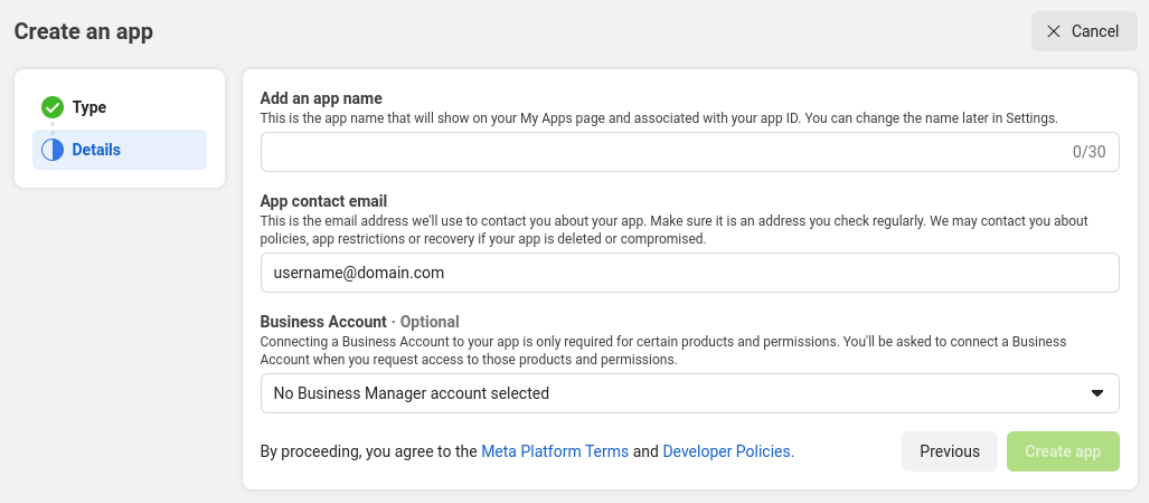
*选择 其他。*

*选择应用程序类型*



d. *选择 **Consumer**。*

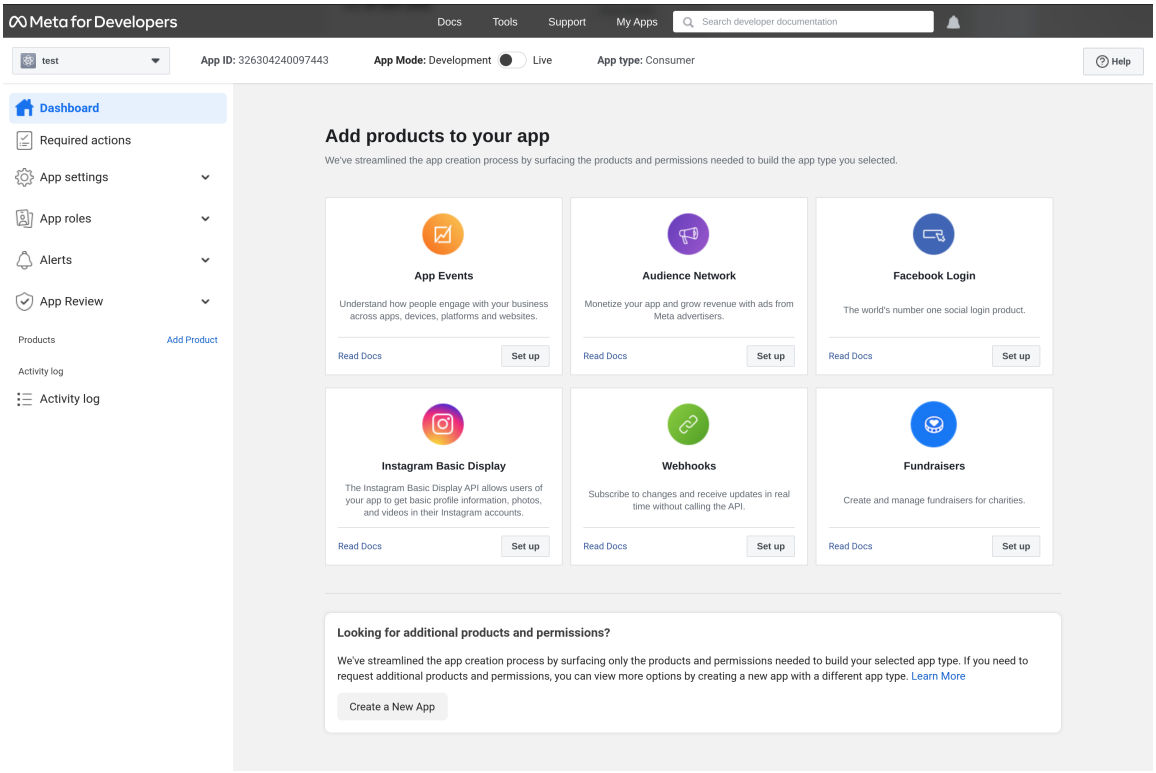
*创建一个应用程序*




e. *填写所有必填字段。*

f. *点 **Create app**。然后，**meta** 带您进入仪表板。*

*添加产品*




g. *在 **Facebook Login** 框中，单击 **Set Up**。*

h. *选择 **Web**。*

i. *在 **Site URL** 字段中输入 **Redirect URI** 的值，然后单击 **Save**。*

j. *在导航面板中，选择 **App settings - Basic**。*

k. *在 **App Secret** 字段中点 **Show**。*

l. *记录 **App ID** 和 **App Secret**。*

5. *在 **Red Hat build of Keycloak** 中的 **Client ID** 和 **Client Secret** 字段中输入您的 **Facebook app** 中的 **App ID** 和 **App Secret**值。*

6. *点 添加*

7. 在 **Default Scopes** 字段中输入所需的范围。默认情况下，红帽构建的 **Keycloak** 使用 电子邮件 范围。有关 **Facebook** 范围的更多信息，请参阅 **Graph API**。

红帽构建的 **Keycloak** 将配置集请求发送到 **graph.facebook.com/me?fields=id,name,email,first_name,last_name**。响应仅包含 **id, name, email, first_name,** 和 **last_name** 字段。要从 **Facebook** 配置集中获取其他字段，请在 **Additional user profile** 项项中添加相应 的范围并添加字段名称。

### 9.4.3. GitHub

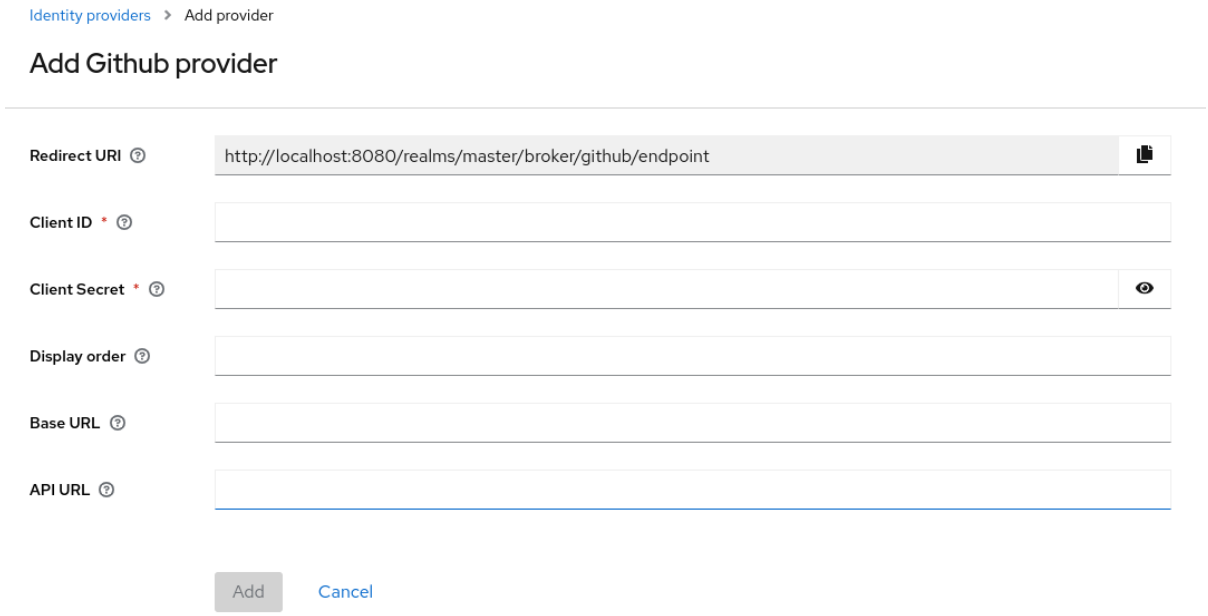
要使用 **GitHub** 登录，请执行以下步骤。

流程

1. 单击菜单中的 **Identity Providers**。

2. 从 **Add provider** 列表中，选择 **Github**。

添加身份提供程序




3. 将 **Redirect URI** 的值复制到您的剪贴板。

4. 在单独的浏览器选项卡中，创建 **OAUTH** 应用。

   a. 在创建应用程序时，在 **Authorization** 回调 **URL** 字段中输入 **Redirect URI** 的值。

   b. 请注意 **OAUTH** 应用程序的管理页面中的客户端 **ID** 和客户端 **secret**。

5. 在 **Red Hat build of Keycloak** 中，将 **Client ID** 的值粘贴到 **Client ID** 字段中。

6. 在 **Red Hat build of Keycloak** 中，将 **Client secret** 的值粘贴到 **Client Secret** 字段中。

7. 点击 **Add**。

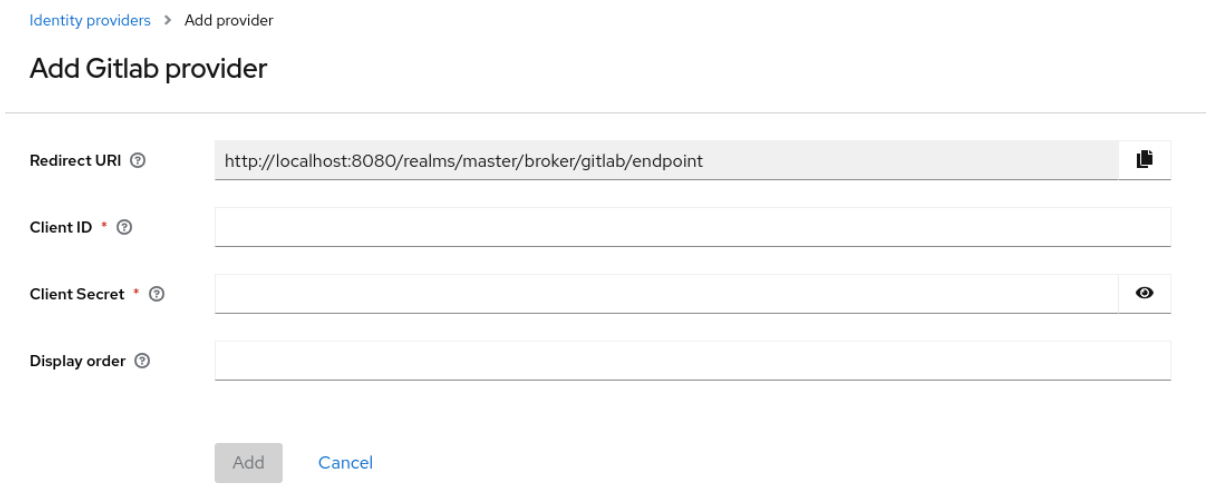
### 9.4.4. GitLab

流程

1. 单击菜单中的 **Identity Providers**。

2. 从 **Add provider** 列表中，选择 **GitLab**。

添加身份提供程序



3. 将 **Redirect URI** 的值复制到您的剪贴板。

4. 在单独的浏览器选项卡中，添加新的 **GitLab** 应用。

   a. 使用剪贴板中的 **Redirect URI** 作为 **Redirect URI**。

   b. 在保存应用程序时 记录应用程序 **ID** 和 **Secret**。

5. 在红帽构建的 **Keycloak** 中，将 **Application ID** 的值粘贴到 **Client ID** 字段中。

6. 在红帽构建的 **Keycloak** 中，将 **Secret** 的值粘贴到 **Client Secret** 字段中。

7. 点击 **Add**。

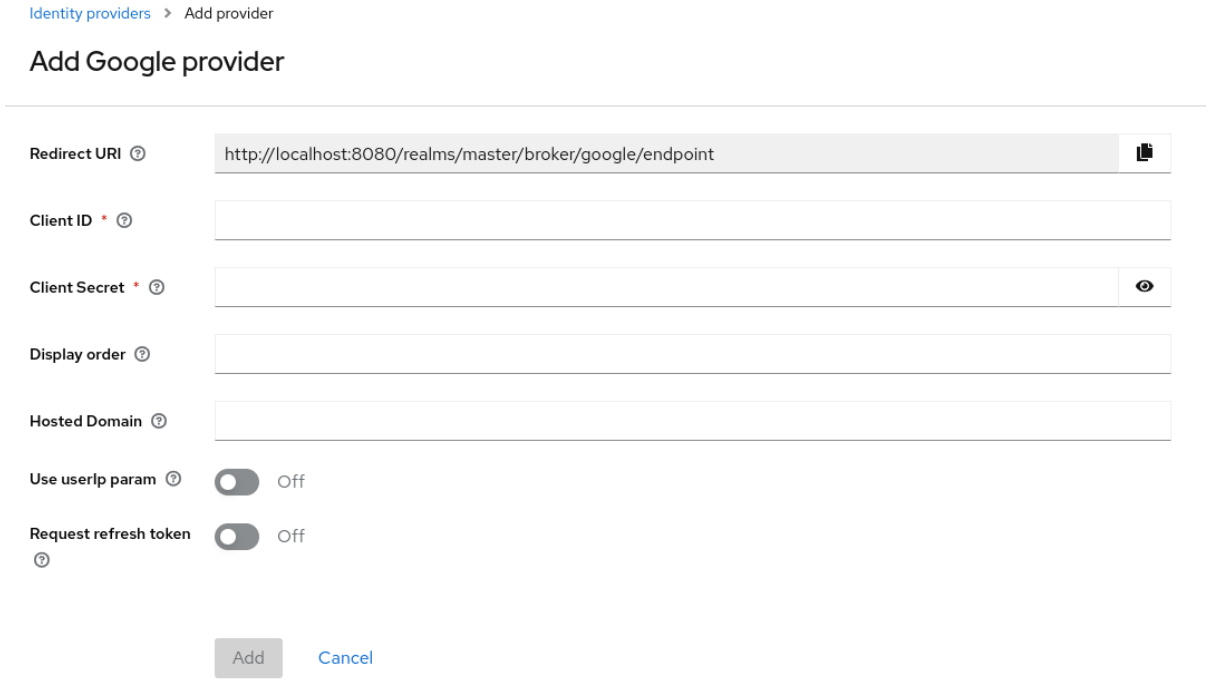
#### 9.4.5. Google

流程

1. 单击菜单中的 **Identity Providers**。

2. 从 **Add provider** 列表中，选择 **Google**。

添加身份提供程序




3. *将 **Redirect URI** 的值复制到您的剪贴板。*

4. *在单独的浏览器选项卡中，打开 **Google Cloud Platform** 控制台。*

5. *在 **Google** 应用程序的 **Google** 仪表板中，在左侧的 **Navigation** 菜单中，将鼠标悬停在 **API & Services** 上，然后点 **OAuth consent** 屏幕 选项。创建同意屏幕，确保同意屏幕的用户类型是外部。*

6. *在 **Google** 仪表板中：*

   a. *点 **Credentials** 菜单。*

   b. *单击 **CREATE CREDENTIALS - OAuth** 客户端 **ID**。*

   c. *从 **Application type** 列表中，选择 **Web application**。*

   d. *使用剪贴板中的 **Redirect URI** 作为 **Authorized** 重定向 **URI***

e. 点 **Create**。

f. 记录 您的客户端 **ID** 和 您的客户端 **secret**。

7. 在红帽构建的 **Keycloak** 中，将您的 客户端 **ID** 的值粘贴到 **Client ID** 字段中。

8. 在红帽构建的 **Keycloak** 中，将您的 客户端 **secret** 的值粘贴到 **Client Secret** 字段中。

9. 点 添加

10. 在 **Default Scopes** 字段中输入所需的范围。默认情况下，红帽构建的 **Keycloak** 使用以下范围： **openid** 配置集 电子邮件。如需 **Google** 范围列表，请参阅 **OAuth Playground**。

11. 要限制对 **GSuite** 组织成员的访问，请在 **Hosted Domain** 字段中输入 **G Suite** 域。

12. 点击 **Save**。

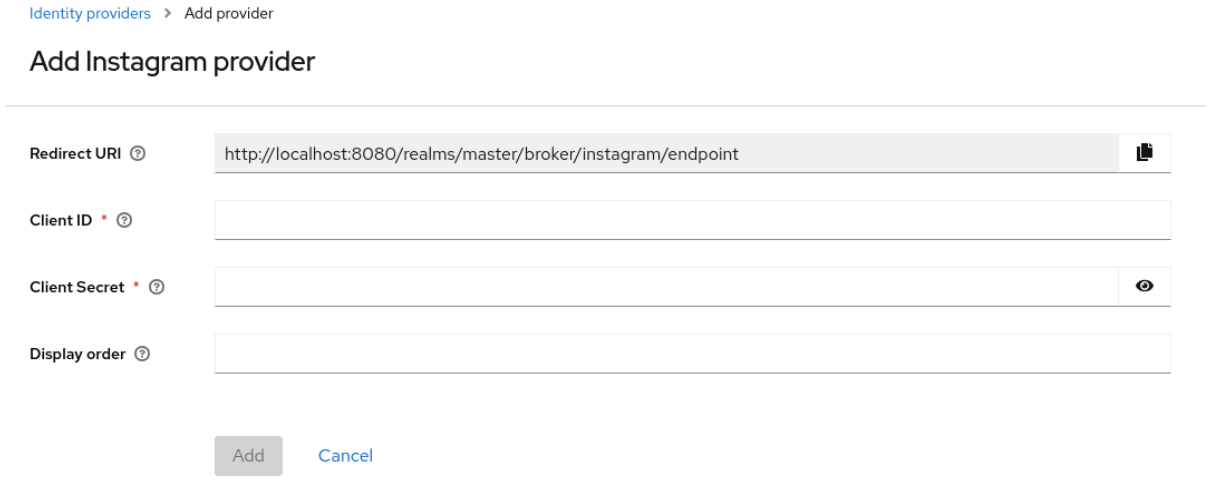
#### 9.4.6. Instagram

流程

1. 单击菜单中的 **Identity Providers**。

2. 从 **Add provider** 列表中，选择 **Instagram**。

添加身份提供程序



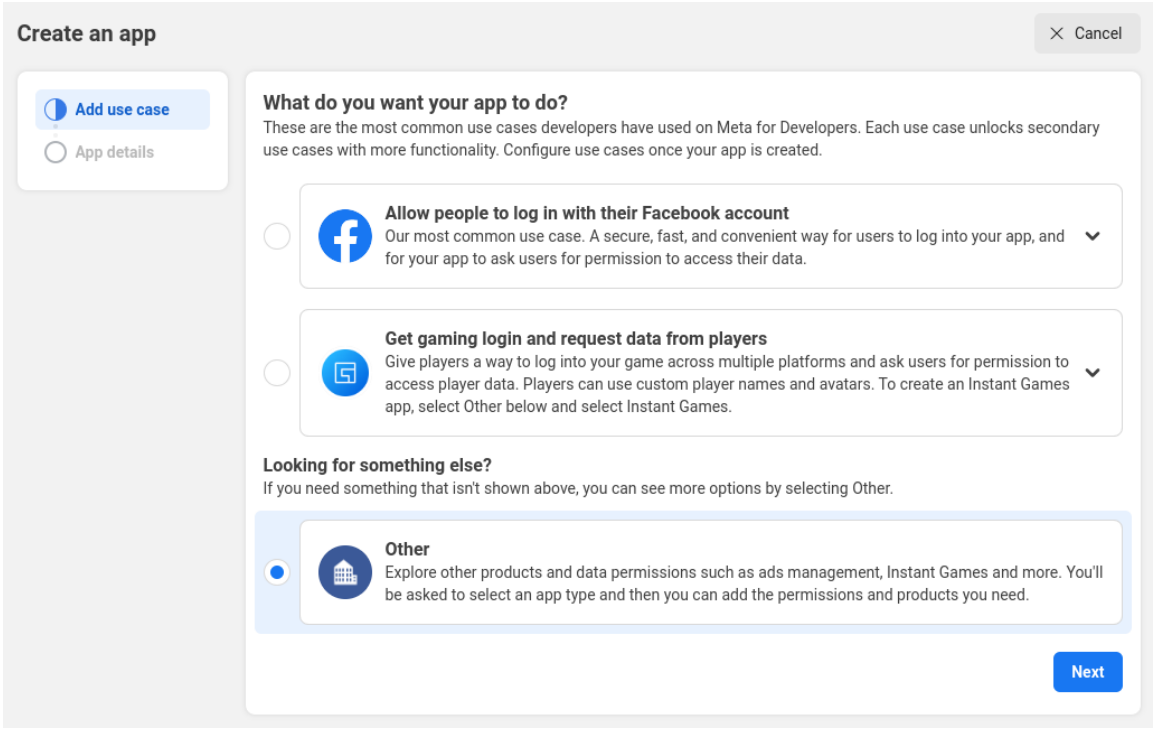

3. *将 **Redirect URI** 的值复制到您的剪贴板。*

4. *在单独的浏览器选项卡中，打开 **Developers** 的 **Meta**。*

   a. *单击 **My Apps**。*

   b. *选择 **Create App**。*

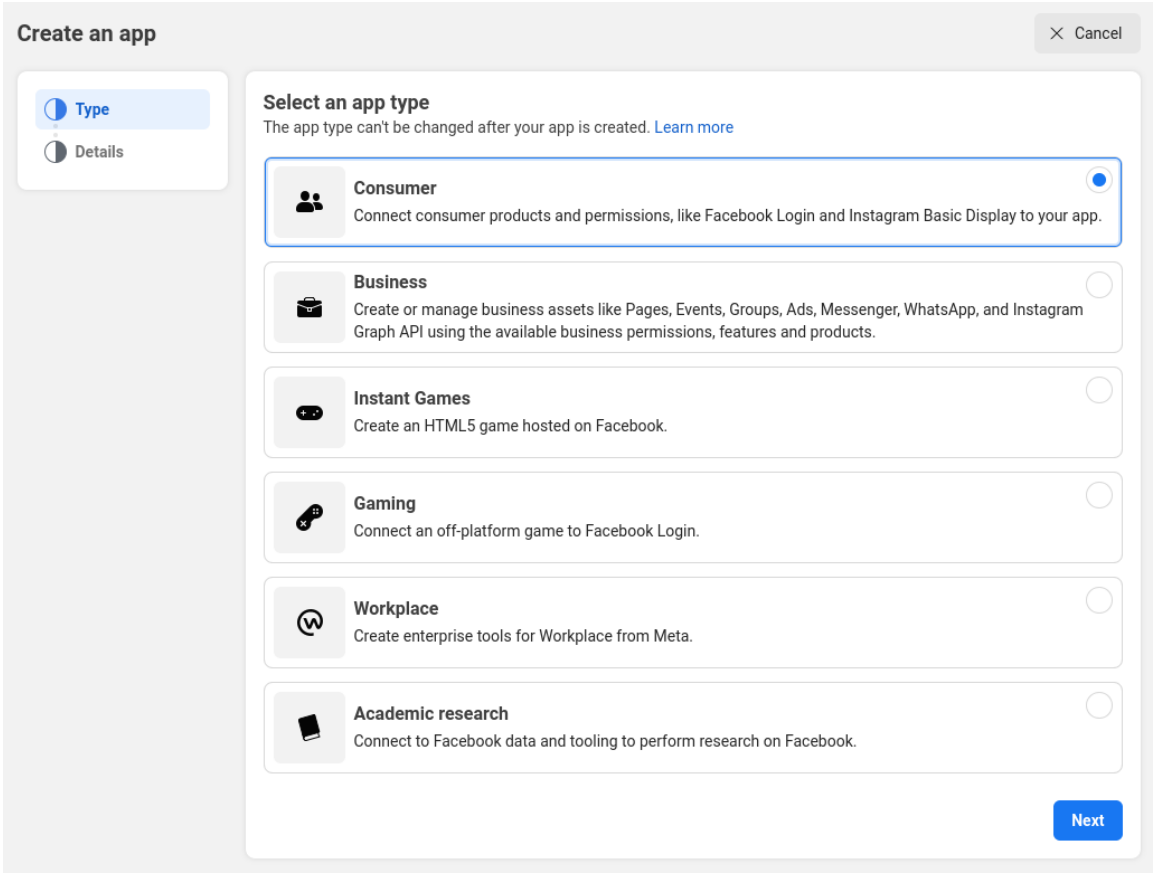
*添加用例*




c.

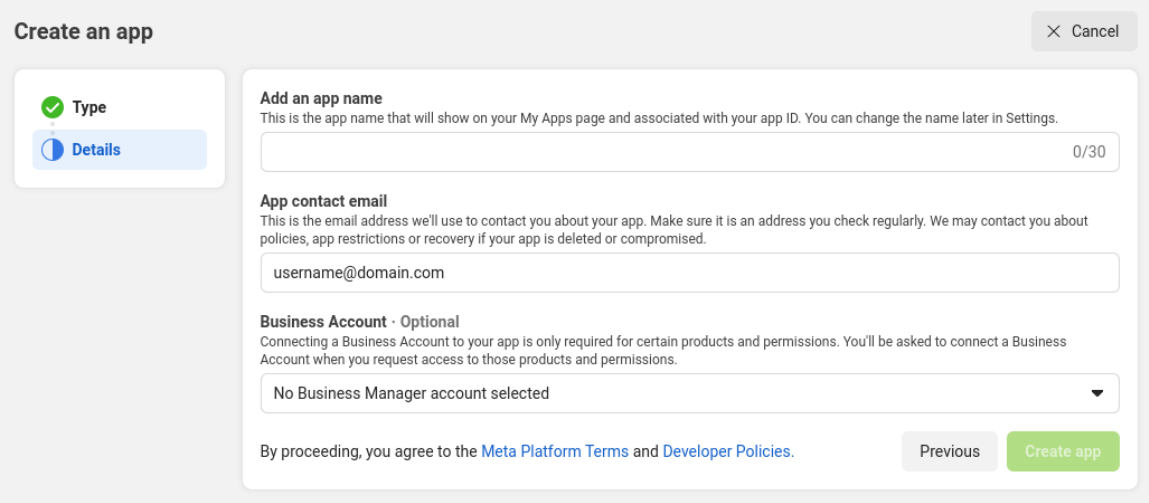
*选择 其他。*

*选择应用程序类型*



d. *选择 **Consumer**。*

*创建一个应用程序*




e. *填写所有必填字段。*

f. *点 **Create app**。然后，**meta** 带您进入仪表板。*

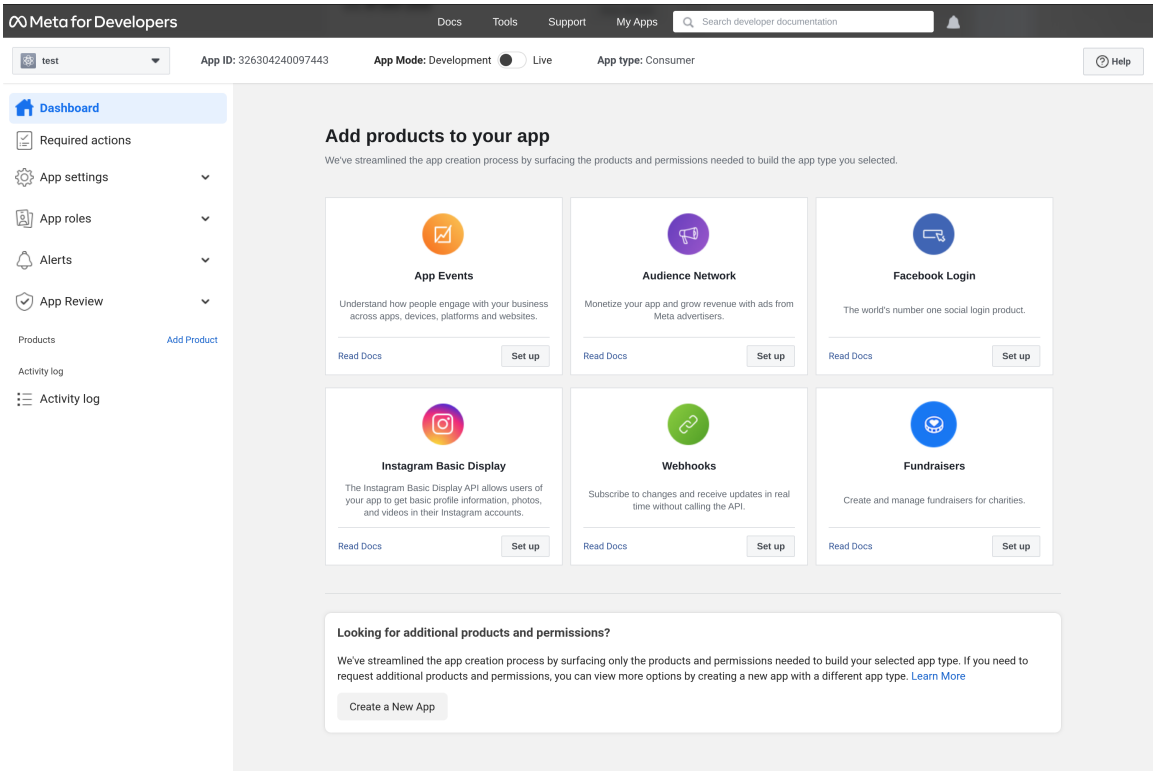
g. *在导航面板中，选择 **App settings - Basic**。*

h. *选择页面底部的 **+ Add Platform**。*

i. *单击 **[Website]**。*

j. *输入您的站点的 **URL**。*

*添加产品*

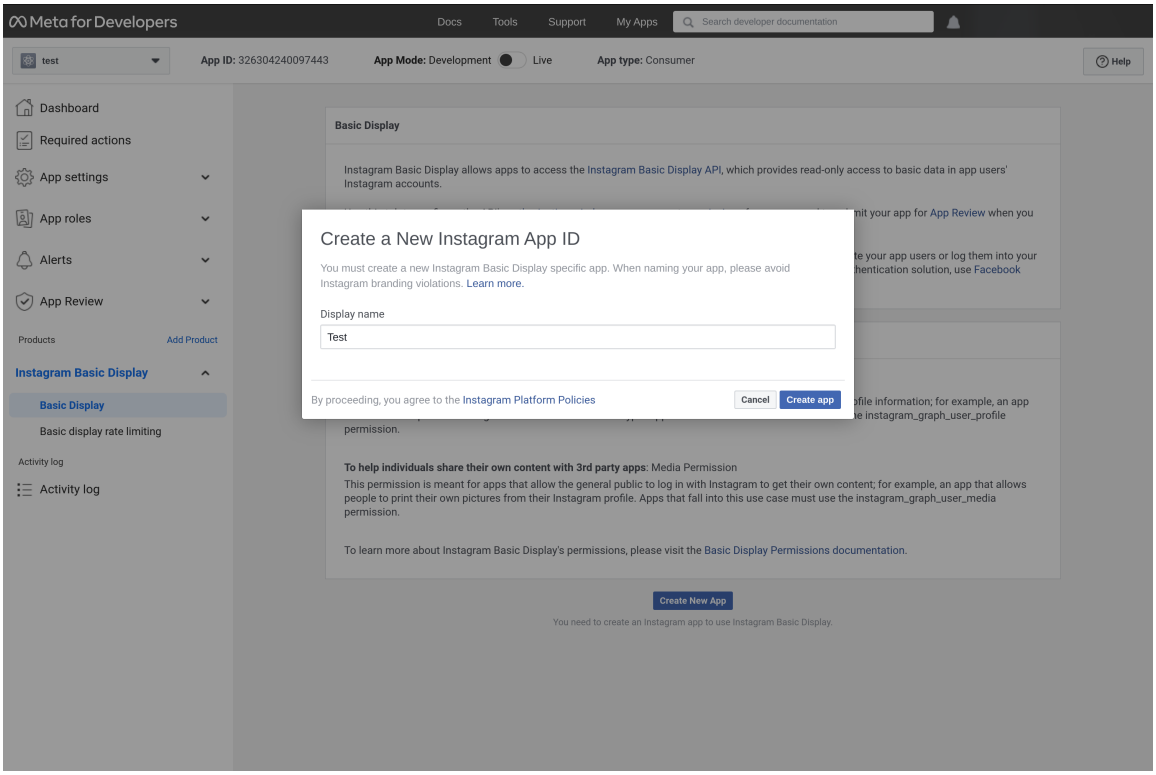



k. *从菜单中选择 **Dashboard**。*

l. *在 **Instagram Basic Display** 框中，单击 **Set Up**。*

m. *单击 **Create New App**。*

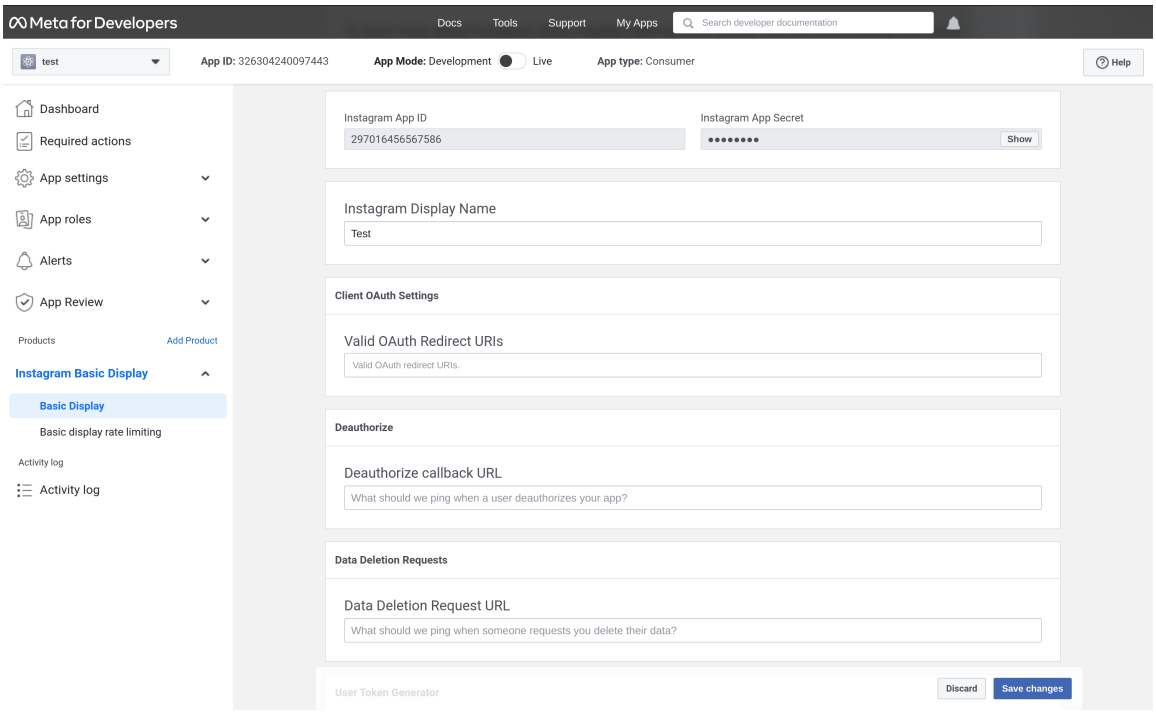
*创建一个新的 **Instagram App ID***




n.

*在 **Display name** 字段中输入值。*

*设置应用程序*




o.

将来自红帽构建的 **Keycloak** 的 重定向 **URL** 粘贴到 **Valid OAuth Redirect URIs** 字段中。

p. 将红帽构建的 **Keycloak** 的 重定向 **URL** 粘贴到 **Deauthorize Callback URL** 字段中。

q. 将红帽构建的 **Keycloak** 的 重定向 **URL** 粘贴到 **Data Deletion Request URL** 字段中。

r. 在 **Instagram App Secret** 字段中点 **Show**。

s. 注意 **Instagram App ID** 和 **Instagram App Secret**。

t. 点 **App Review - Requests**。

u. 按照屏幕上的说明进行操作。

5. 在红帽构建的 **Keycloak** 中，将 **Instagram App ID** 的值粘贴到 **Client ID** 字段中。

6. 在红帽构建的 **Keycloak** 中，将 **Instagram App Secret** 的值粘贴到 **Client Secret** 字段中。

7. 点击 **Add**。

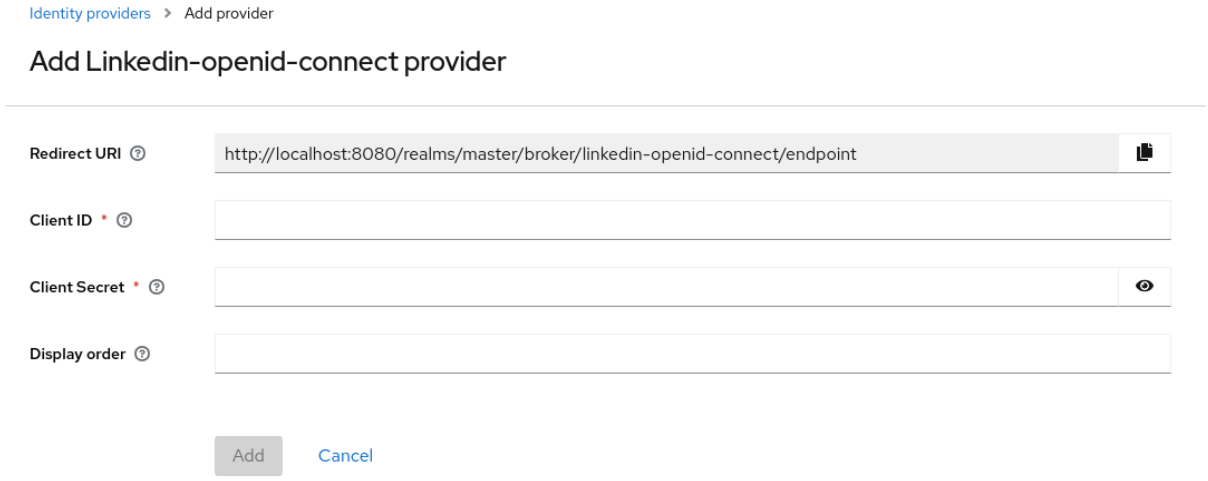
#### 9.4.7. LinkedIn

流程

1. 单击菜单中的 **Identity Providers**。

2. 从 **Add provider** 列表中，选择 **LinkedIn**。

添加身份提供程序




3. *将 **Redirect URI** 的值复制到您的剪贴板。*

4. *在单独的浏览器选项卡中，在 **LinkedIn** 开发人员门户创建一个应用程序。*

   a. *创建应用程序后，单击 **Auth** 选项卡。*

   b. *在 **Authorized redirect URLs for your app** 字段中输入 **Redirect URI** 的值。*

   c. *记录 您的客户端 **ID** 和 您的客户端 **Secret**。*

   d. *点 **Products** 选项卡，使用 **OpenID Connect** 产品对 **Sign In with LinkedIn** 的 **Request** 访问。*

5. *在 **Red Hat build of Keycloak** 中，将 **Client ID** 的值粘贴到 **Client ID** 字段中。*

6. *在红帽构建的 **Keycloak** 中，将 **Client Secret** 的值粘贴到 **Client Secret** 字段中。*

7. *点击 **Add**。*

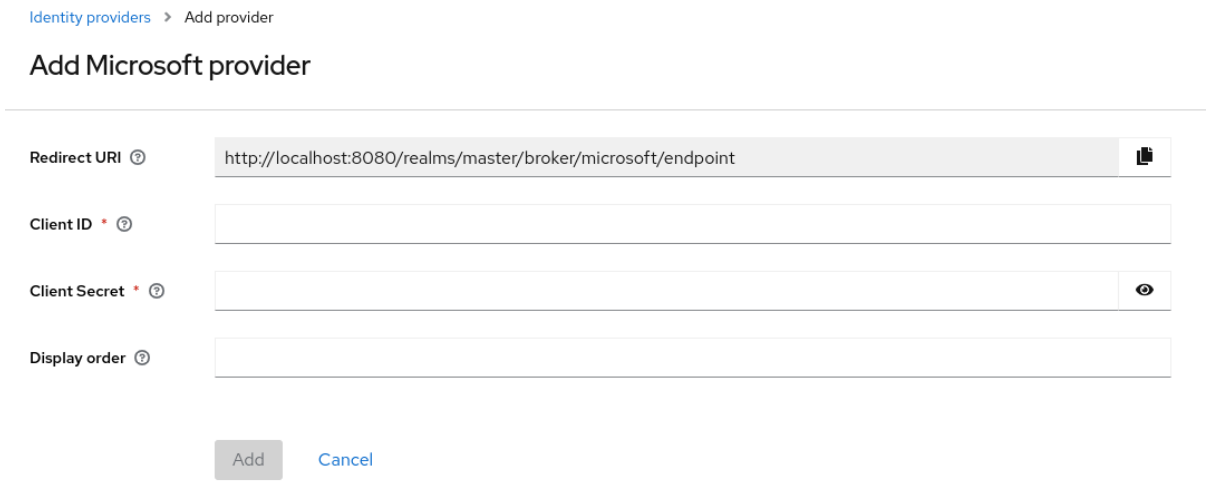
### *9.4.8. Microsoft*

*流程*

1. *单击菜单中的 **Identity Providers**。*

2. *从 **Add provider** 列表中，选择 **Microsoft**。*

*添加身份提供程序*




3. *将 **Redirect URI** 的值复制到您的剪贴板。*

4. *在单独的浏览器选项卡中，在 **App registrations** 下注册一个应用程序。*

   a. *在 **Redirect URI** 部分中，选择 **Web** 作为平台，并将 **Redirect URI** 的值粘贴到字段中。*

   b. *在 **App registrations** 下查找应用程序，并在 **Certificates & secrets** 部分中添加新客户端 **secret**。*

   c. *请注意创建的 **secret** 的值。*

   d. *请注意 **Overview** 部分中的 应用程序（客户端）**ID**。*

5. *在 **Red Hat build of Keycloak** 中，将 **Application (client) ID** 的值粘贴到 **Client ID** 字段中。*

6. *在红帽构建的 **Keycloak** 中，将 **secret** 的 **Value** 粘贴到 **Client Secret** 字段中。*

7. *点击 **Add**。*

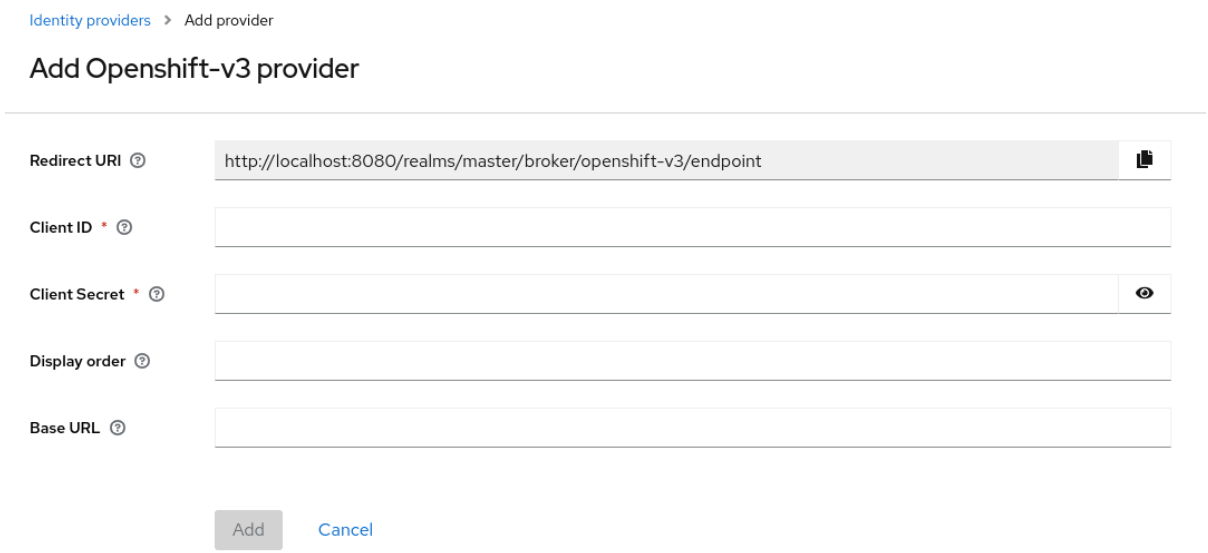
### *9.4.9. OpenShift 3*

*流程*

1. *单击菜单中的 **Identity Providers**。*

2. *从 **Add provider** 列表中，选择 **Openshift v3**。*

*添加身份提供程序*




3. *将 **Redirect URI** 的值复制到您的剪贴板。*

4. *使用 **oc** 命令行工具注册您的客户端。*

-

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: v1
metadata:
 name: kc-client 1
secret: "..." 2
redirectURIs:
 - "http://www.example.com/" 3
grantMethod: prompt 4
')
```

1

**OAuth** 客户端 的名称。在向 **<openshift_master> /oauth/ authorize** 和 **<openshift_master> / oauth/ token** 发出请求时，作为 **client_id request** 参数传递。

2

用于 **client_ secret** 请求参数的 **Keycloak** 的 **secret Red Hat build**。

3

对 **< openshift_master>/oauth/authorize** 和 **< openshift_master> /oauth/token** 的请求中指定的 **redirect_uri** 参数必须等于 **redirectURIs** 中的某一 **URI**（或作为前缀）。您可以从 **Identity Provider** 屏幕中的 **Redirect URI** 字段获取它

4

**grantMethod Red Hat build of Keycloak** 用来确定这个客户端请求令牌但没有被用户授予访问权限时的操作。

1. 在 **Red Hat build of Keycloak** 中，将 **Client ID** 的值粘贴到 **Client ID** 字段中。
2. 在红帽构建的 **Keycloak** 中，将 **Client Secret** 的值粘贴到 **Client Secret** 字段中。
3. 点击 **Add**。

#### 9.4.10. OpenShift 4

先决条件

1. 存储在红帽构建的 **Keycloak Truststore** 中的 **OpenShift 4** 实例的证书。

2. *配置红帽 **Keycloak** 服务器的红帽构建以使用信任存储。如需更多信息，请参阅配置 **Truststore** 章节。*

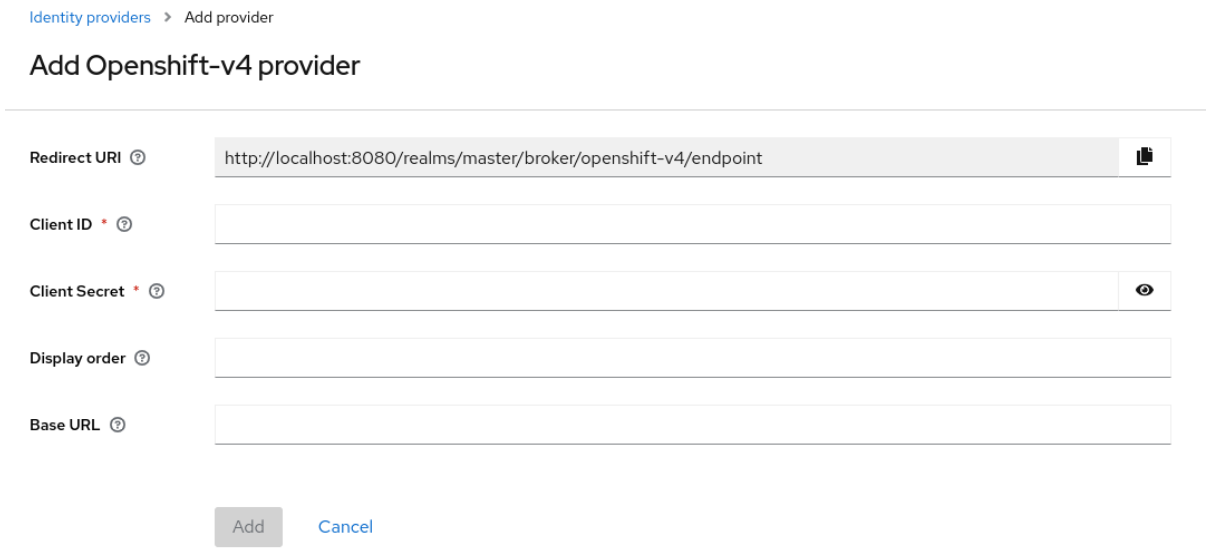
*流程*

1. *单击菜单中的 **Identity Providers**。*

2. *在 **Add provider** 列表中，选择 **Openshift v4**。*

3. *输入 **Client ID** 和 **Client Secret**，并在 **Base URL** 字段中输入 **OpenShift 4** 实例的 **API URL**。另外，您可以将 **Redirect URI** 复制到您的剪贴板。*

*添加身份提供程序*




4. *通过 **OpenShift 4** 控制台(**Home → API Explorer → OAuth Client → Instances**)或使用 **oc** 命令行工具注册您的客户端。*

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
 name: kc-client 1
secret: "..." 2
redirectURIs:
```

```
 - "<here you can paste the Redirect URI that you copied in the previous step>" 3
grantMethod: prompt 4
')
```

1

*OAuth 客户端 的名称。在向 <openshift_master> /oauth/ authorize 和 <openshift_master> / oauth/ token 发出请求时，作为 client_id request 参数传递。OAuthClient 对象中 name 参数必须相同，红帽构建的 Keycloak 配置。*

2

*红帽构建的 Keycloak 的 secret 用作 client_secret 请求参数。*

3

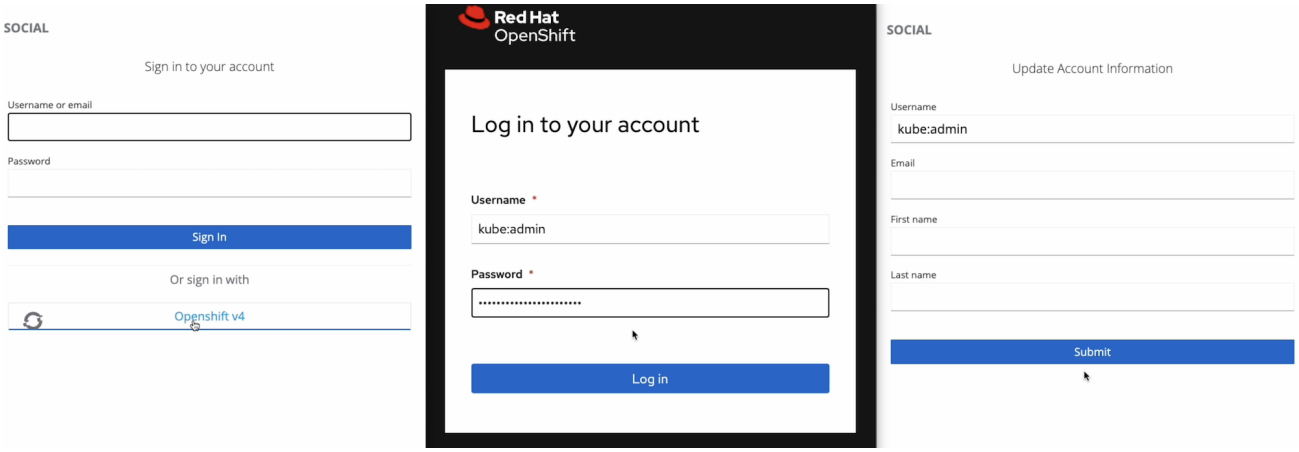
*对 < openshift_master>/oauth/authorize 和 < openshift_master> /oauth/token 的请求中指定的 redirect_uri 参数必须等于 redirectURIs 中的某一 URI （或作为前缀）。正确配置它的最简单方法是，从红帽构建的 Keycloak OpenShift 4 Identity Provider 配置页面(Redirect URI 字段)复制它。*

4

*grantMethod Red Hat build of Keycloak 用来确定这个客户端请求令牌但没有被用户授予访问权限时的操作。*

*最后，您应该在红帽构建的 Keycloak 实例的登录页面上看到 OpenShift 4 身份提供程序。在单击它后，您应当重定向到 OpenShift 4 登录页面。*

*结果*



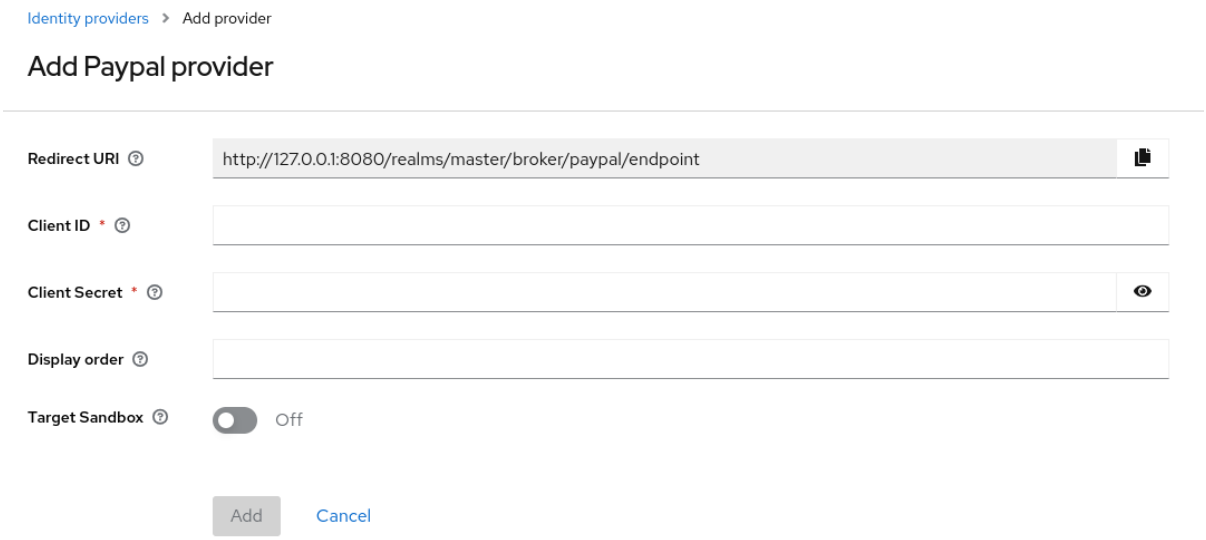
如需更多信息，请参阅官方 **OpenShift** 文档。

### 9.4.11. PayPal

流程

1. 单击菜单中的 **Identity Providers**。
2. 从 **Add provider** 列表中，选择 **PayPal**。

添加身份提供程序




3. 将 **Redirect URI** 的值复制到您的剪贴板。
4. 在单独的浏览器选项卡中，打开 **PayPal Developer** 应用程序区域。
   a. 点 **Create App** 创建 **PayPal** 应用。
   b. 记录 客户端 **ID** 和客户端 **Secret**。单击 **Show** 链接，以查看该机密。

c.

*确保选中 带有 **PayPal** 登录。*

d.

*在 **Log in with PayPal** 下点 **Advanced Settings**。*

e.

*将 **return URL** 字段的值设置为来自红帽 **Keycloak** 的 **Redirect URI** 的值。请注意，**URL** 不得包含 **localhost**。如果要在本地使用红帽 **Keycloak** 构建，请将 返回 **URL** 中的 **localhost** 替换为 **127.0.0.1**，然后在 **localhost** 的浏览器中使用 **127.0.0.1** 访问 **Keycloak** 的红帽构建。*

f.

*确保选中了 **Full Name** 和 **Email** 字段。*

g.

*单击 **Save**，然后单击 **Save Changes**。*

5.

*在 **Red Hat build of Keycloak** 中，将 **Client ID** 的值粘贴到 **Client ID** 字段中。*

6.

*在红帽构建的 **Keycloak** 中，将 **Secret 键 1** 的值粘贴到 **Client Secret** 字段中。*

7.

*点击 **Add**。*

### *9.4.12. 堆栈溢出*

*流程*

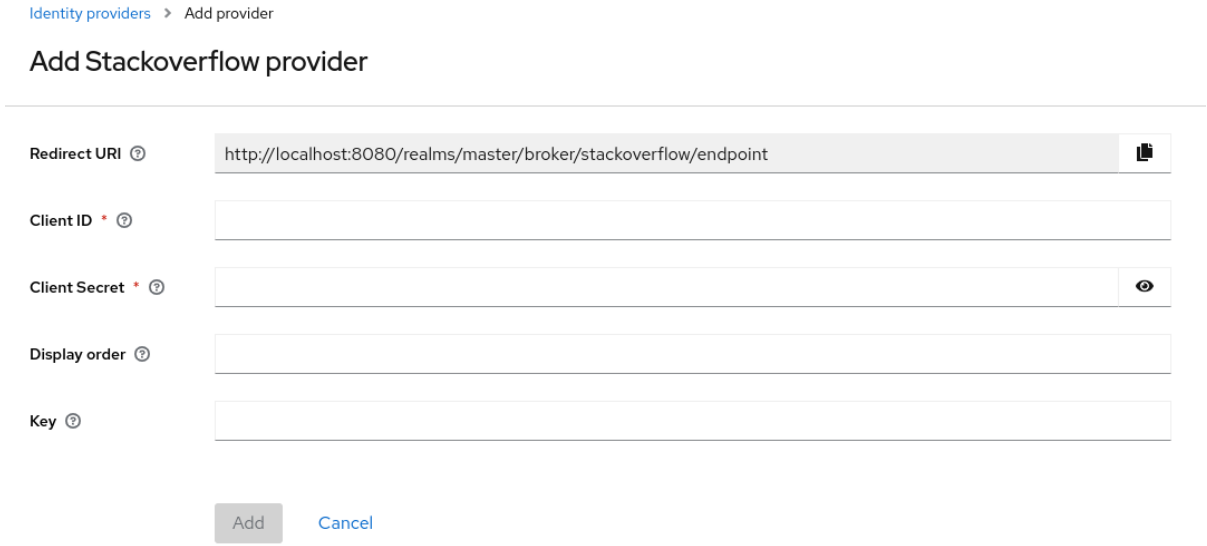
1.

*单击菜单中的 **Identity Providers**。*

2.

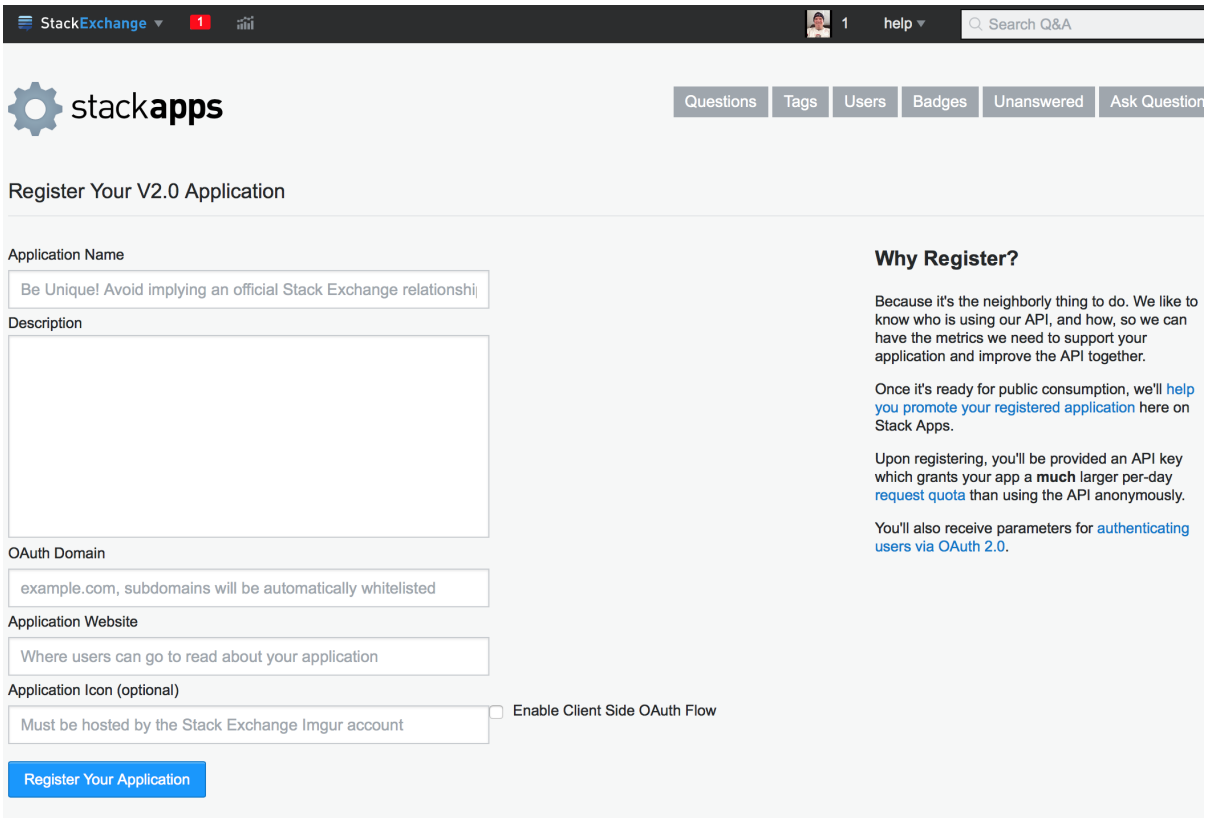
*从 **Add provider** 列表中，选择 **Stack Overflow**。*

*添加身份提供程序*




3. *在单独的浏览器选项卡中，登录 在 **Stack Apps** 上注册您的应用程序。*

*注册应用程序*

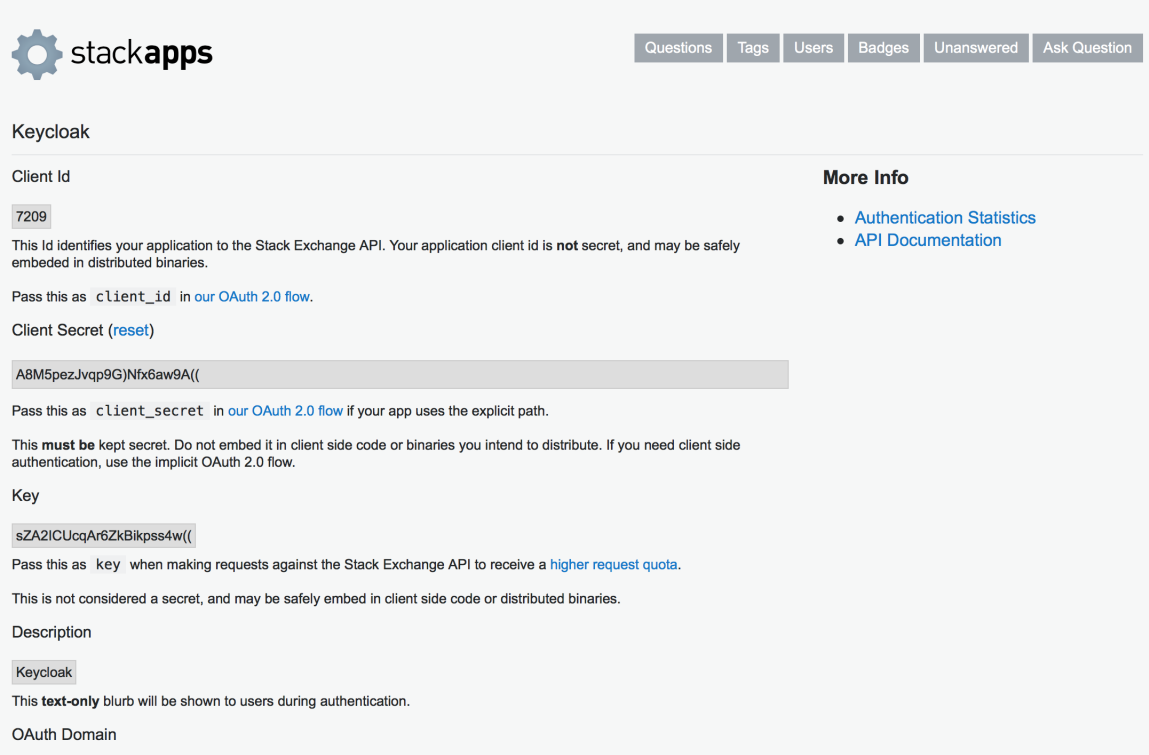



a. *在 **Application Name** 字段中输入您的应用程序名称。*

b. *在 **OAuth Domain** 字段中输入 **OAuth** 域。*

c. *点 **Register Your Application**。*

*设置*




4. *记录 客户端 **ID** 、 客户端 **Secret** 和密钥。*

5. *在 **Red Hat build of Keycloak** 中，将 **Client Id** 的值粘贴到 **Client ID** 字段中。*

6. *在红帽构建的 **Keycloak** 中，将 **Client Secret** 的值粘贴到 **Client Secret** 字段中。*

7. *在红帽构建的 **Keycloak** 中，将 **Key** 的值粘贴到 **Key** 字段中。*

8. *点击 **Add**。*

### 9.4.13. Twitter

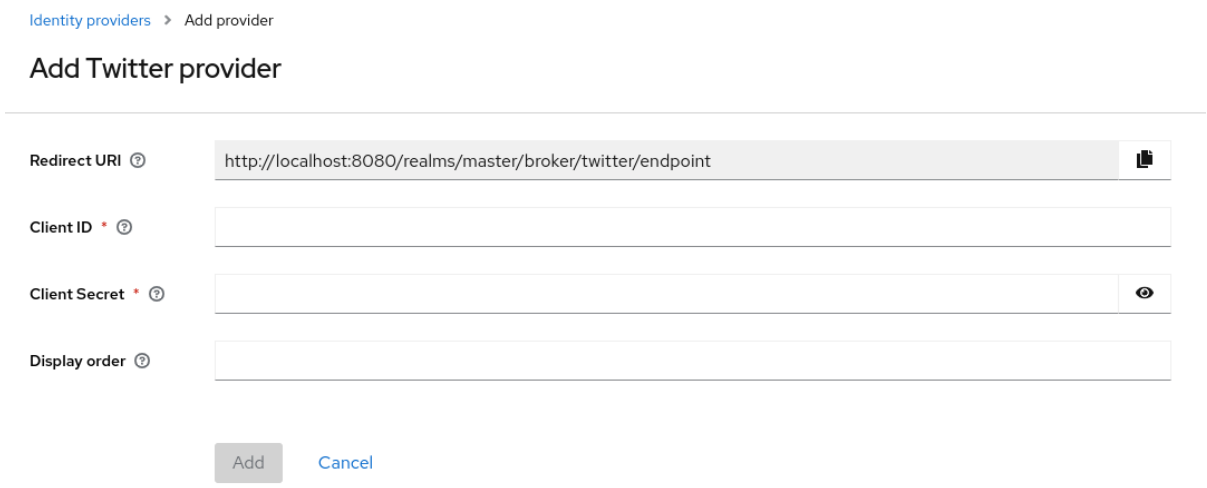
*先决条件*

1. ***Twitter** 开发人员帐户。*

*流程*

1. *单击菜单中的 **Identity Providers**。*
2. *从 **Add provider** 列表中，选择 **Twitter**。*

*添加身份提供程序*




3. *将 **Redirect URI** 的值复制到您的剪贴板。*
4. *在单独的浏览器选项卡中，在 **Twitter** 应用管理 中创建应用程序。*
   a. *输入 **App name** 并点 **Next**。*
   b. *请注意 **API Key** 和 **API Key Secret** 的值，再点 **App settings**。*

c. 在 **User authentication settings** 部分中，点 **Set up** 按钮。

d. 选择 **Web App** 作为 **App** 的 **Type**。

e. 将 **Redirect URL** 的值粘贴到 **Callback URI / Redirect URL** 字段中。

f. **Website URL** 的值可以是除 **localhost** 以外的任何有效的 **URL**。

g. 单击 **Save**，然后单击 **Done**。

5. 在 **Red Hat build of Keycloak** 中，将 **API Key** 的值粘贴到 **Client ID** 字段中。

6. 在红帽构建的 **Keycloak** 中，将 **API Key Secret** 的值粘贴到 **Client Secret** 字段中。

7. 点击 **Add**。

### 9.5. OPENID CONNECT V1.0 身份提供程序

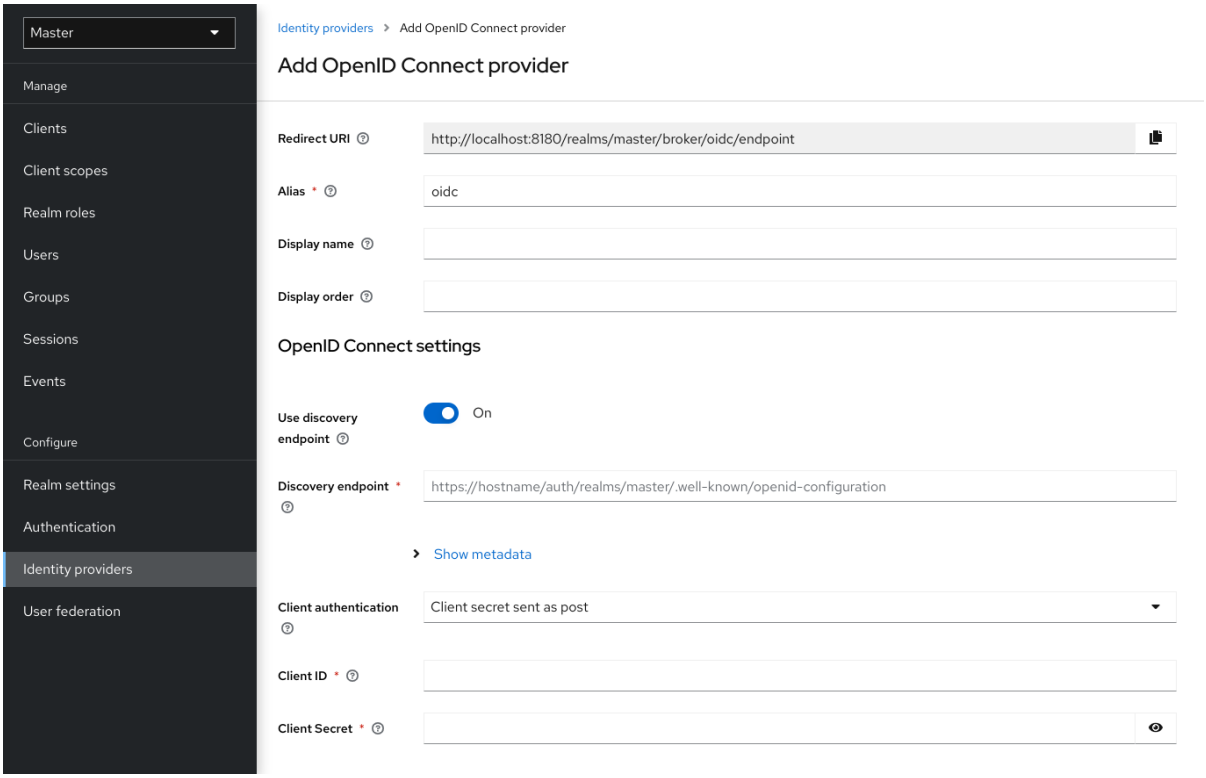
红帽根据 **OpenID Connect** 协议构建 **Keycloak** 代理身份提供程序。这些身份提供程序**(IDP)**必须支持规格中定义的 *授权代码流*，以验证用户和授权访问权限。

流程

1. 单击菜单中的 **Identity Providers**。

2. 从 **Add provider** 列表中，选择 **OpenID Connect v1.0**。

添加身份提供程序




3. *输入您的初始配置选项。有关配置选项的更多信息，请参阅 常规 IDP 配置。*

*表 9.2. OpenID 连接配置*

| Configuration | 描述 |
| --- | --- |
| 授权 URL | OIDC 协议所需的授权 URL 端点。 |
| 令牌 URL | OIDC 协议所需的令牌 URL 端点。 |
| 注销 URL | OIDC 协议中的注销 URL 端点。这个值是可选的。 |
| Backchannel Logout | 对 IDP 的后台、带外、REST 请求以注销用户。有些 IDP 仅通过浏览器重定向执行注销，因为它们可能会使用浏览器 Cookie 识别会话。 |
| 用户信息 URL | OIDC 协议定义的端点。此端点指向用户配置集信息。 |
| 客户端身份验证 | 定义红帽构建 Keycloak 与授权代码流相关的客户端验证方法。如果使用私钥签名的 JWT，Red Hat build of Keycloak 会使用 realm 私钥。在其他情况下，定义客户端 secret。如需更多信息，请参阅 客户端身份验证规格。 |

| Configuration | 描述 |
| --- | --- |
| 客户端 ID | 域充当外部 IDP 的 OIDC 客户端。如果您使用授权代码流与外部 IDP 交互，则域必须具有 OIDC 客户端 ID。 |
| Client Secret | 来自外部 密码库的 客户端机密。如果您使用授权代码流，则需要此 secret。 |
| 客户端断言签名算法 | 创建 JWT 断言作为客户端身份验证的签名算法。如果 JWT 使用私钥或客户端机密签名为 jwt，则需要它。如果没有指定算法，则会对以下算法进行调整。当 JWT 使用私钥签名时，**RS256** 会进行调整。当 Client secret 为 jwt 时，**HS256** 会进行调整。 |
| 客户端断言 Audience | 用于客户端断言的受众。默认值为 IDP 的令牌端点 URL。 |
| 发布者 | 红帽构建的 Keycloak 会在来自 IDP 的响应中根据这个值验证签发者声明。 |
| 默认范围 | OIDC 范围列表，红帽构建的 Keycloak 使用身份验证请求发送。默认值为 **openid**。一个空格来分隔每个范围。 |
| 提示 | OIDC 规格中的 prompt 参数。通过此参数，您可以强制重新验证和其他选项。详情请查看规格。 |

| Configuration | 描述 |
| --- | --- |
| 接受来自客户端的 prompt=none 转发 | 指定 IDP 是否接受包含 **prompt=none** 查询参数的转发验证请求。如果 realm 收到带有 **prompt=none** 的 auth 请求，则 realm 会检查用户当前是否经过身份验证，并在用户没有登录时返回 **login_required** 错误。当红帽构建的 Keycloak 为身份验证请求决定默认 IDP （使用 **kc_idp_hint** 查询参数或域具有默认 IDP）时，您可以将 **prompt=none** 的身份验证请求转发到默认的 IDP。默认 IDP 检查用户的身份验证。因为并非所有 IDP 都支持使用 **prompt=none** 的请求，所以红帽构建的 Keycloak 会使用这个切换来指示默认 IDP 在重定向身份验证请求前支持该参数。<br><br>如果用户在 IDP 中未经身份验证，客户端仍然会收到 **login_required** 错误。如果用户在 IDP 中是真实的，如果红帽构建的 Keycloak 必须显示需要用户交互的身份验证页面，客户端仍然可以收到 **interaction_required** 错误。此身份验证包括所需的操作（如密码更改）、同意屏幕，以及 **first broker login** 流或 **post broker login** 流显示的屏幕。 |
| 验证签名 | 指定红帽构建的 Keycloak 是否在这个 IDP 签名的外部 ID 令牌中验证签名。如果 **ON**，红帽构建的 Keycloak 必须知道外部 OIDC IDP 的公钥。为了性能，红帽构建 Keycloak 会缓存外部 OIDC 身份提供程序的公钥。 |
| 使用 JWKS URL | 如果 **Validate Signatures** 为 **ON**，则此交换机适用。如果使用 **JWKS URL** 为 **ON**，则红帽构建的 Keycloak 从 JWKS URL 下载 IDP 的公钥。当身份提供程序生成新密钥对时下载新密钥。如果 **OFF**，Red Hat build of Keycloak 使用来自其数据库的公钥（或证书），因此当 IDP 密钥对更改时，将新密钥导入到红帽构建的 Keycloak 数据库。 |
| JWKS URL | 指向 IDP JWK 密钥位置的 URL。如需更多信息，请参阅 JWK 规格。如果您使用外部红帽构建的 Keycloak 作为 IDP，如果您的代理红帽构建的 Keycloak 在 http://broker-keycloak:8180 上运行，您可以使用一个 URL，其 realm is **test**。http://broker-keycloak:8180/realms/test/protocol/openid-connect/certs |
| 验证公钥 | 红帽构建的 Keycloak 用来验证外部 IDP 签名的 PEM 格式的公钥。如果 **Use JWKS URL** 为 **OFF**，则应用此密钥。 |

| Configuration | 描述 |
| --- | --- |
| 验证公钥 Id | 如果 **Use JWKS URL** 为 **OFF**，则应用此设置。此设置以 PEM 格式指定公钥的 ID。因为没有从密钥计算密钥 ID 的标准方法，外部身份提供程序可以使用来自红帽构建的 Keycloak 使用的不同算法。如果没有指定此字段的值，Red Hat build of Keycloak 会为所有请求使用验证公钥，而不考虑外部 IDP 发送的密钥 ID。当 **ON** 时，此字段的值是红帽构建的 Keycloak 使用的密钥 ID 从供应商验证签名，且必须与 IDP 指定的密钥 ID 匹配。 |

*您可以通过提供指向 **OpenID** 提供程序元数据的 **URL** 或文件来导入所有这些配置数据。如果您连接到红帽构建的 **Keycloak** 外部 **IDP**，您可以从 **< root>/realms/{realm-name}/.well-known/openid-configuration** 导入 **IDP** 设置。这个链接是一个 **JSON** 文档，描述 **IDP** 的元数据。*

*如果要使用 **Json Web 加密(JWE)** **ID** 令牌或提供程序中的 **UserInfo** 响应，**IDP** 需要知道用于红帽构建的 **Keycloak** 的公钥。供应商使用为不同加密算法定义的 **realm 密钥** 来解密令牌。**Red Hat build of Keycloak** 提供了一个标准的 **JWKS 端点**，**IDP** 可用于自动下载密钥。*

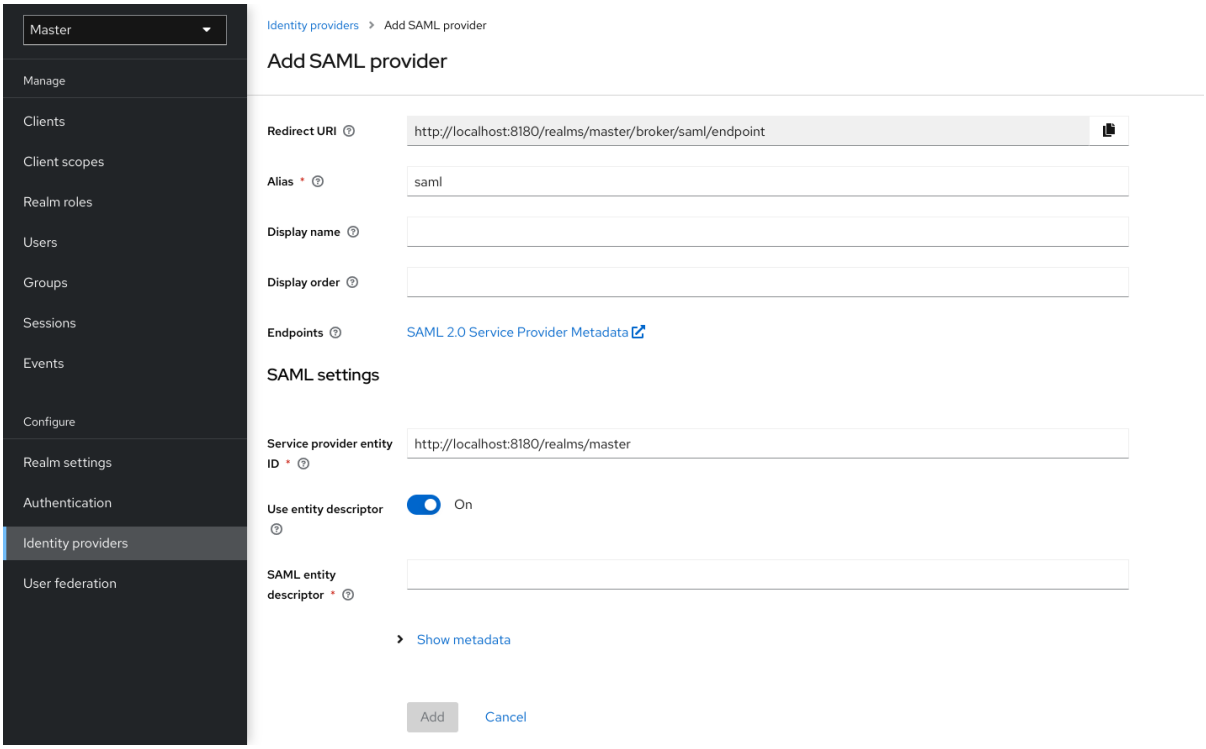
### *9.6. SAML V2.0 身份提供程序*

*红帽构建的 **Keycloak** 可以根据 **SAML v2.0** 协议代理身份提供程序。*

*流程*

1. *单击菜单中的 **Identity Providers**。*
2. *从 **Add provider** 列表中，选择 **SAML v2.0**。*

*添加身份提供程序*




3.

*输入您的初始配置选项。有关配置选项的更多信息，请参阅 常规 IDP 配置。*

*表 9.3. SAML 配置*

| 配置 | 描述 |
| --- | --- |
| 服务提供程序实体 ID | 远程身份提供程序用来识别来自此服务提供商请求的 SAML 实体 ID。默认情况下，此设置设置为 realm base URL < **root>/realms/{realm-name}**。 |
| 身份提供程序实体 ID | 用于验证收到 SAML 断言的颁发者的实体 ID。如果为空，则不会执行 Issuer 验证。 |
| 单点登录服务 URL | 启动身份验证过程的 SAML 端点。如果您的 SAML IDP 发布 IDP 实体描述符，则会在那里指定此字段的值。 |
| 单个 Logout Service URL | SAML 注销端点。如果您的 SAML IDP 发布 IDP 实体描述符，则会在那里指定此字段的值。 |
| Backchannel Logout | 如果您的 SAML IDP 支持返回频道注销，请将这个开关切换到 **ON**。 |
| NameID 策略格式 | 与名称标识符格式对应的 URI 引用。默认情况下，红帽构建的 Keycloak 把它设置为 **urn:oasis:names:tc:SAML:2.0:nameid-format:persistent**。 |

| 配置 | 描述 |
| --- | --- |
| 主体类型 | 指定 SAML 断言中的哪些部分将用于识别和跟踪外部用户身份。可以是 Subject NameID 或 SAML 属性（按名称或友好名称）。主题 NameID 值不能与 'urn:oasis:names:tc:SAML:2.0:nameid-format:transient' NameID Policy Format 值一起设置。 |
| 主体属性 | 如果主体类型是 non-blank，此字段指定识别属性的名称("Attribute [Name]")或 friendly name ("Attribute [Friendly Name]")。 |
| 允许创建 | 允许外部身份提供程序创建新标识符来代表主体。 |
| HTTP-POST 绑定响应 | 控制 SAML 绑定，以响应外部 IDP 发送的任何 SAML 请求。当 **OFF** 时，红帽构建的 Keycloak 使用重定向绑定。 |
| AuthnRequest 的 HTTP-POST Binding | 控制当从外部 IDP 请求身份验证时的 SAML 绑定。当 **OFF** 时，红帽构建的 Keycloak 使用重定向绑定。 |
| want AuthnRequests Signed | 在 **ON** 时，红帽构建的 Keycloak 使用域的密钥对签署发送到外部 SAML IDP 的请求。 |
| want Assertions Signed | 指明此服务提供商是否需要签名的 Assertion。 |
| want Assertions Encrypted | 指明此服务提供商是否需要加密的 Assertion。 |
| 签名算法 | 如果 **Want AuthnRequests Signed** 为 **ON**，则使用签名算法。请注意，基于 **SHA1** 的算法已弃用，并可能在以后的发行版本中删除。我们建议使用一些更安全的算法，而不是 ***_SHA1**。另外，使用 ***_SHA1** 算法时，如果 SAML 身份提供程序（如红帽构建的 Keycloak 实例在 Java 17 或更高版本上运行）时，验证签名无法正常工作。 |
| 加密算法 | SAML IDP 用于加密 SAML 文档、断言或 ID 的加密算法。解密 SAML 文档部分的对应解密密钥将根据这种配置的算法进行选择，并应该可用于加密(ENC)使用的域密钥。如果没有配置算法，则允许任何支持的算法，并根据 SAML 文档本身中指定的算法选择解密密钥。 |

| 配置 | 描述 |
| --- | --- |
| SAML 签名密钥名称 | 使用 POST 绑定发送的签名 SAML 文档包含 **KeyName** 元素中的签名密钥标识，该元素默认包含红帽 Keycloak 密钥 ID 的红帽构建。外部 SAML IDP 可以预期不同的密钥名称。此交换机控制 **KeyName** 是否包含：* **KEY_ID** - Key ID。* **CERT_SUBJECT** - 与 realm 键对应的证书的主题。Microsoft Active Directory Federation Services 期望 **CERT_SUBJECT**.* **NONE** - 红帽构建的 Keycloak 省略了 SAML 信息中的键名称提示。 |
| 强制身份验证 | 用户必须在外部 IDP 中输入其凭证，即使用户已经登录。 |
| 验证签名 | 当 **ON** 时，域要求 SAML 请求和来自外部 IDP 的响应进行数字签名。 |
| 元数据描述符 URL | 身份提供程序发布 **IDPSSODescriptor** 元数据的外部 URL。当点 **Reload** key 或 **Import key 操作时，此 URL 用于下载** 身份提供程序证书。 |
| 使用元数据描述符 URL | 在 **ON** 时，验证签名的证书会自动从 **元数据描述符 URL** 下载，并在红帽构建的 Keycloak 中缓存。SAML 提供程序可以通过两种不同的方式验证签名。如果请求特定的证书（通常在 **POST** 绑定中），且没有在缓存中，则证书会自动从 URL 中刷新。如果请求所有证书来验证签名(**REDIRECT** binding)，则刷新只在最大缓存时间后进行（请参阅所有供应商配置指南中的 public-key-storage spi 以了解有关缓存如何工作的更多信息）。也可以使用身份提供程序页面中的操作 **Reload Keys** 手动更新缓存。<br><br>当选项为 **OFF** 时，**验证 X509 证书中的证书** 用于验证签名。 |
| 验证 X509 证书 | 当使用 **元数据描述符 URL** 为 **OFF** 时，红帽构建的 Keycloak 用来验证 SAML 请求签名和来自外部 IDP 的响应。可以使用逗号分隔多个证书(、 )。证书可以从 **元数据描述符 URL** 重新导入，点击身份提供程序页面中的 **Import Keys** 操作。该操作会在 metadata 端点中下载当前证书，并将它们分配给同一选项的配置。您需要单击 **Save** 以绝对存储重新导入的证书。 |
| 签名服务提供商元数据 | 在 **ON** 时，红帽构建的 Keycloak 使用域的密钥对为 SAML 服务提供商元数据描述符 签名。 |
| 传递主题 | 控制红帽构建的 Keycloak 是否将 **login_hint** 查询参数转发到 IDP。Red Hat build of Keycloak 将此字段的值添加到 AuthnRequest 的 Subject 中的 login_hint 参数中，以便目标供应商可以预先填充其登录表单。 |

| 配置 | 描述 |
| --- | --- |
| 属性使用服务索引 | 标识设置为请求到远程 IDP 的属性。红帽构建的 Keycloak 会自动将身份提供程序配置中映射的属性添加到自动生成的 SP 元数据文档中。 |
| 属性使用服务名称 | 自动生成的 SP 元数据文档中公告的属性集合的描述性名称。 |

*您可以通过提供 **URL** 或指向外部 **IDP** 实体描述符的 **URL** 或文件来导入所有配置数据。如果您要连接到红帽构建的 **Keycloak** 外部 **IDP**，您可以从 **URL < root>/realms/{realm-name}/protocol/saml/descriptor** 导入 **IDP** 设置。这个链接是一个 **XML** 文档，描述了有关 **IDP** 的元数据。您还可以通过提供指向外部 **SAML IDP** 实体描述符的 **URL** 或 **XML** 文件来导入所有这些配置数据。*

### *9.6.1. 请求特定的 AuthnContexts*

*身份提供程序有助于客户端在验证用户身份的身份验证方法中指定限制。例如，询问 **MFA**、**Kerberos** 身份验证或安全要求。这些限制使用特定的 **AuthnContext** 标准。客户端可以请求一个或多个条件，并指定身份提供程序必须与请求的 **AuthnContext**、精确或满足其他等效条件匹配。*

*您可以通过在 **Requested AuthnContext Constraints** 部分添加 **ClassRefs** 或 **DeclRefs** 来列出您的服务提供商所需的条件。通常，您需要提供 **ClassRefs** 或 **DeclRefs**，因此请检查您的身份提供程序文档支持哪些值。如果没有 **ClassRefs** 或 **DeclRefs**，则身份提供程序不会强制实施额外的限制。*

*表 **9.4.** 请求的 **AuthnContext** 限制*

| Configuration | 描述 |
| --- | --- |
| 比较 | 身份提供程序用于评估上下文要求的方法。可用值有 **Exact**、**Minimum**、**Maximum**、或 **Better**。默认值为 **Exact**。 |
| AuthnContext ClassRefs | AuthnContext ClassRefs 描述所需条件。 |
| AuthnContext DeclRefs | AuthnContext DeclRefs 描述所需标准。 |

### *9.6.2. SP Descriptor*

*当您访问供应商的 **SAML SP** 元数据时，请在身份提供程序配置设置中查找 **Endpoints** 项。它包含一个 **SAML 2.0 Service Provider Metadata** 链接，用于为服务提供商生成 **SAML** 实体描述符。您可以下载描述符或复制其 **URL**，然后将其导入到远程身份提供程序。*

通过转至以下 **URL**，也可以公开提供此元数据：

```
http[s]://{host:port}/realms/{realm-name}/broker/{broker-alias}/endpoint/descriptor
```

确保您在访问描述符前保存任何配置更改。

### 9.6.3. 在 SAML 请求中发送主题

默认情况下，指向 **SAML** 身份提供程序的社交按钮将用户重定向到以下登录 **URL**：

```
http[s]://{host:port}/realms/${realm-name}/broker/{broker-alias}/login
```

在此 **URL** 中添加名为 **login_hint** 的查询参数，将参数的值作为 **Subject** 属性添加到 **SAML** 请求中。如果此查询参数为空，红帽构建的 **Keycloak** 不会对请求添加主题。

启用 **"Pass subject"** 选项，以在 **SAML** 请求中发送主题。

## 9.7. CLIENT-SUGGESTED 身份提供程序

**OIDC** 应用程序可以通过提示他们要使用的身份提供程序来绕过红帽构建的 **Keycloak** 登录页面。您可以通过在 **Authorization Code Flow authorization** 端点中设置 **kc_idp_hint** 查询参数来启用此功能。

使用红帽构建的 **Keycloak OIDC** 客户端适配器，您可以在访问应用程序中的安全资源时指定此查询参数。

例如：

```
GET /myapplication.com?kc_idp_hint=facebook HTTP/1.1
Host: localhost:8080
```

在这种情况下，您的域必须具有一个带有 **facebook** 别名的身份提供程序。如果此提供程序不存在，则会显示登录表单。

如果使用 **keycloak.js** 适配器，您也可以实现与以下相同的行为：

```
const keycloak = new Keycloak('keycloak.json');

keycloak.createLoginUrl({
 idpHint: 'facebook'
});
```

使用 **kc_idp_hint** 查询参数时，如果您为 **Identity Provider Redirector authenticator** 配置了一个，客户端可以覆盖默认身份提供程序。客户端可以通过将 **kc_idp_hint** 查询参数设置为空值来禁用自动重定向。

### 9.8. 映射声明和断言

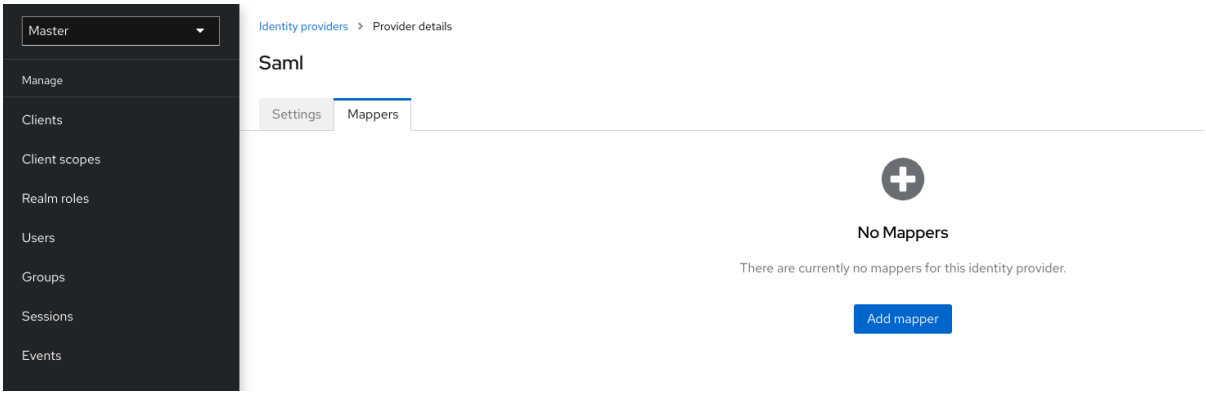
您可以将 **SAML** 和 **OpenID Connect** 元数据（由您要进行身份验证的外部 **IDP** 提供）导入到域中。导入后，您可以提取用户配置集元数据和其他信息，以便您可以将其提供给应用程序。

每个用户使用外部身份提供程序登录到您的域的用户在本地红帽构建的 **Keycloak** 数据库中有一个条目，具体取决于 **SAML** 或 **OIDC** 断言和声明的元数据。

流程

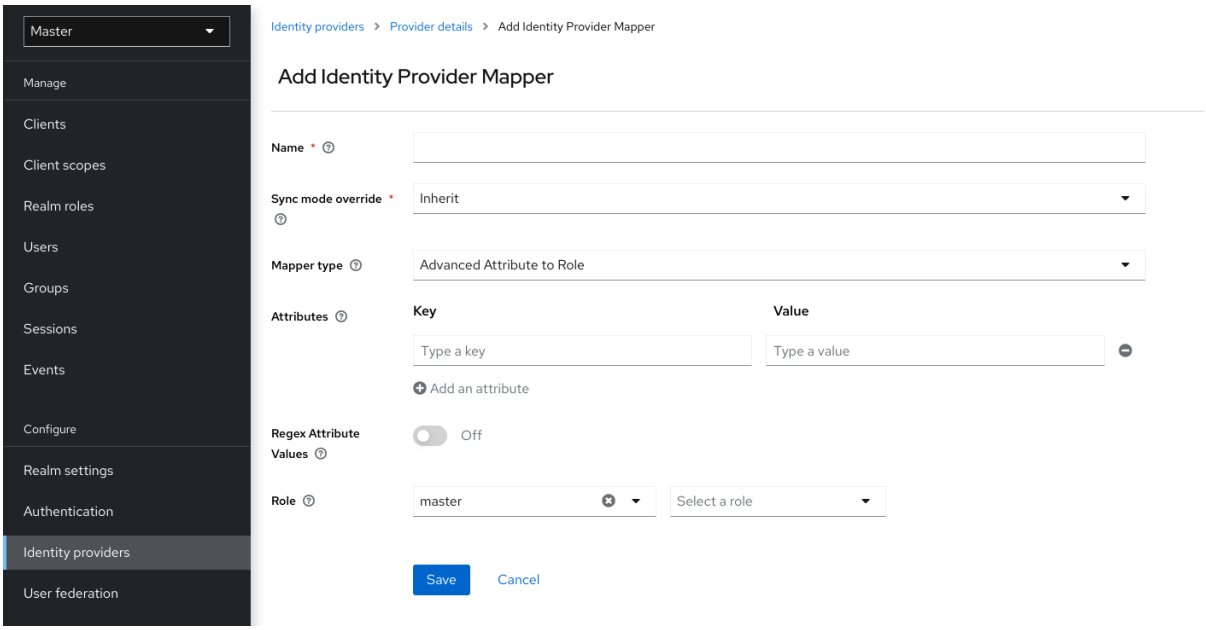
1. 单击菜单中的 **Identity Providers**。
2. 从列表中选择其中一个身份提供程序。
3. 点 **Mappers** 选项卡。

身份提供程序映射器



4. *单击 **Add mapper**。*

*身份提供程序映射器*




5. *为 **Sync Mode Override** 选择一个值。当用户根据此设置重复登录时，映射程序会更新用户信息。*

   a. *选择 **legacy** 来使用之前红帽构建的 **Keycloak** 版本的行为。*

   b. *选择在首次登录到带有特定身份提供程序的 **Keycloak** 的红帽构建时，从导入数据。*

   c. *选择 **force** 在每次用户登录时更新用户数据。*

   d. *选择 **inherit** 以使用在身份提供程序中配置的同步模式。所有其他选项将覆盖此同步模式。*

6. *从 **Mapper Type** 列表中选择一个 **mapper**。将鼠标悬停在 映射程序类型上，用于映射程序描述和配置要为映射程序输入。*

7. 点 **Save**。

对于基于 **JSON** 的声明，您可以使用点表示法来嵌套和方括号来访问按索引访问数组字段。例如，**contact.address[0].country**。

要调查社交供应商提供的用户配置文件 **JSON** 数据的结构，您可以在启动服务器时启用 **DEBUG** 级别日志记录器 **org.keycloak.social.user_profile_dump**。

### 9.9. 可用的用户会话数据

用户从外部 **IDP** 登录后，红帽构建的 **Keycloak** 存储用户会话会记录您可以访问的数据。此数据可以传播到请求使用令牌登录的客户端，或使用适当的客户端映射程序传递给客户端。

**identity_provider**

用于执行登录的代理的 **IDP** 别名。

**identity_provider_identity**

当前经过身份验证的用户的 **IDP** 用户名。通常，但并不总是与红帽构建的 **Keycloak** 用户名相同。例如，红帽构建的 **Keycloak** 可将用户 **john'** 链接到 **Facebook** 用户 **john123@gmail.com**。在这种情况下，用户会话记录的值为 **john123@gmail.com**。

您可以使用用户会话 *类型的协议 映射程序 将* 此信息传播到您的客户端。

### 9.10. 第一个登录流

当用户通过身份代理登录时，红帽构建的 **Keycloak** 导入，以及域本地数据库中用户的链接方面。当红帽构建的 **Keycloak** 通过外部身份提供程序成功验证用户时，可能会出现两种情况：

- 红帽构建的 **Keycloak** 已导入，并将用户帐户与经过身份验证的身份提供程序帐户相关联。在这种情况下，红帽构建的 **Keycloak** 以现有用户身份进行身份验证，并将重新重定向到应用程序。

- 红帽构建的 **Keycloak** 中没有此用户的帐户。通常，您将新帐户注册并导入到红帽构建的 **Keycloak** 数据库，但可能存在具有相同电子邮件地址的现有 **Keycloak** 帐户构建。将现有本地帐

户自动链接到外部身份提供程序是潜在的安全漏洞。您无法始终信任来自外部身份提供程序的信息。

在处理其中的一些情况时，不同的机构有不同的要求。使用红帽构建的 **Keycloak**，您可以使用 **IDP** 设置中的 **First Login Flow** 选项，选择首次从外部 **IDP** 登录的用户 的工作流。默认情况下，第一个 登录流选项指向 第一个代理登录流，但您可以将流或不同的流用于不同的身份提供程序。

流位于 身份验证选项卡下的管理控制台中。当您选择 **First Broker Login** 流时，您会看到默认使用的验证器。您可以重新配置现有的流。例如，您可以禁用一些验证器，将它们标记为 必需，或者配置一些验证器。

### 9.10.1. 默认第一次登录流验证器

查看配置文件

- 此验证器显示配置集信息页面，因此用户可以查看红帽从身份提供程序中检索 **Keycloak** 的配置集。
- 您可以在 **Actions** 菜单中设置 **Update Profile On First Login** 选项。
- 在 **ON** 时，用户会显示配置集页面，请求额外信息来联合用户的身份。
- 缺少 时，如果身份提供程序不提供强制信息（如电子邮件、名字或姓氏）时，用户会看到配置文件页面。
- 当 **OFF** 时，配置集页面不会显示，除非用户在 **Confirm Link Existing Account authenticator** 显示的页面中点 **Review profile info** 链接的后续阶段。

如果唯一，创建用户

此验证器检查是否已有一个已存在的 **Keycloak** 帐户，其电子邮件或用户名（如身份提供程序中的帐户一样）。如果没有，则验证器只创建一个新的本地红帽构建 **Keycloak** 帐户，并将其与身份提供程序链接，整个流程已完成。否则，它会进入下一个 **Handle Existing Account** 子流。如果始终要确保没有重复的帐户，您可以将此验证器标记为 **REQUIRED**。在这种情况下，如果已存在红帽构建 **Keycloak** 帐户，用户会看到错误页面，并且用户需要通过帐户管理链接身份提供程序帐户。

- 此验证器验证是否已有一个红帽构建的 **Keycloak** 帐户，其电子邮件或用户名与身份提供程序的帐户相同。

- 如果帐户不存在，则验证器会创建一个本地的红帽构建 **Keycloak** 帐户，将这个帐户与身份提供程序连接，并终止流。

- 如果存在帐户，则验证器将实施下一个 **Handle Existing Account** 子流。

- 要确保没有重复的帐户，您可以将此验证器标记为 **REQUIRED**。如果红帽构建的 **Keycloak** 帐户存在，用户会看到错误页面，用户必须通过帐户管理链接其身份提供程序帐户。

确认链接现有帐户

- 在信息页面中，用户会看到一个具有相同电子邮件的 **Keycloak** 帐户的红帽构建。用户可以再次查看其配置文件，并使用不同的电子邮件或用户名。流重启并返回到 **Review Profile** 验证器。

- 另外，用户可以确认他们希望将其身份提供程序帐户与其现有红帽构建的 **Keycloak** 帐户相关联。

- 如果您不希望用户看到此确认页面，请直接通过电子邮件验证或重新身份验证链接身份提供程序帐户，请禁用此验证器。

通过电子邮件验证现有帐户

- 默认情况下，此 验证器 是 **ALTERNATIVE**。如果域配置了 **SMTP** 设置，红帽构建的 **Keycloak** 会使用此验证器。

- 验证器向用户发送电子邮件，以确认他们希望将身份提供程序与其红帽构建的 **Keycloak** 帐户相关联。

- 如果您不想通过电子邮件确认链接，但希望用户使用其密码重新进行身份验证，请禁用此验证器。

通过重新身份验证验证现有帐户

- 如果电子邮件验证器不可用，请使用此验证器。例如，您尚未为您的域配置 **SMTP**。此验证器显示一个登录屏幕，供用户验证将其红帽构建的 **Keycloak** 帐户与身份提供程序相关

联。

- 用户也可以重新验证已链接到其红帽构建的 **Keycloak** 帐户的另一个身份提供程序。

- 您可以强制用户使用 **OTP**。否则，它是可选的，如果您为用户帐户设置了 **OTP**，则使用它。

### 9.10.2. 自动链接现有第一次登录流

警告

**AutoLink** 验证器在通用环境中是危险的，用户可以使用任意用户名或电子邮件地址注册自己。除非您仔细策展用户注册并分配用户名和电子邮件地址，否则不要使用此验证器。

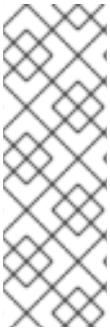
要配置在不提示的情况下自动链接用户的第一个登录流，请使用以下两个验证器创建新流：

如果唯一，创建用户

此验证器可确保红帽构建 **Keycloak** 处理唯一用户。将验证器要求设置为 **Alternative**。

自动设置现有用户

此验证器将现有用户设置为身份验证上下文，无需验证。将验证器要求设置为**"Alternative"**。



注意

此设置是可用的最简单的设置，但可以使用其他验证器。例如：如果您希望最终用户确认其配置集信息，您可以在流的开头添加 **Review Profile authenticator**。* 您可以在此流中添加验证机制，强制用户验证其凭证。添加验证机制需要复杂的流程。例如，您可以在 **"Alternative"** 子流中将 **"Automatically Set Existing User"** 和 **"Password Form"** 设置为 **"Required"**。

### 9.10.3. 禁用自动用户创建

**Default first login** 流查找与外部身份匹配的 **Keycloak** 帐户的红帽构建，并提供链接它们。如果没有匹配的红帽构建的 **Keycloak** 帐户，则流会自动创建一个。

对于某些设置，这个默认行为可能不适合。例如，如果您使用只读 **LDAP** 用户存储，其中所有用户都预先创建。在这种情况下，您必须关闭自动创建用户。

禁用用户创建：

流程

1. 点菜单中的 **Authentication**。
2. 从列表中选择 **First Broker Login**。
3. 设置 创建用户（如果唯一 ）到 **DISABLED**。
4. 将 **Confirm Link Existing Account** 设置为 **DISABLED**。

此配置还意味着红帽构建的 **Keycloak** 本身将无法确定哪个内部帐户会与外部身份对应。因此， **Verify Existing Account By Re-authentication authenticator** 将要求用户提供用户名和密码。

注意

通过身份提供程序启用或禁用用户创建在域用户 注册交换机上 完全独立。您可以由身份提供程序启用 **user-creation**，并在域登录设置中禁用了用户自助注册，反之亦然。

### 9.10.4. 检测现有用户第一次登录流

要配置第一个登录流，请执行以下操作：

- 只有已在此域中注册的用户可以登录，

- 在不提示的情况下，会自动链接用户。

使用以下两个验证器创建新流：

检测现有代理用户

此验证器可确保处理唯一的用户。将验证器要求设置为 **REQUIRED**。

自动设置现有用户

自动将现有用户设置为身份验证上下文，而无需验证。将验证器要求设置为 **REQUIRED**。

您必须将 身份提供程序配置的第一个登录流 设置为该流。如果要使用身份提供程序属性更新用户配置集**(Last Name, First Name…)**，您可以将 **Sync Mode** 设置为 **force**。

注意

如果要将身份委派给其他身份提供程序（如 **GitHub**、**Facebook …**），但您想要管理可以登录的用户，则可以使用这个流。

使用这个配置，红帽构建的 **Keycloak** 无法决定哪个内部帐户与外部身份对应。**Verify Existing Account By Re-authentication authenticator** 向提供程序询问用户名和密码。

### 9.11. 检索外部 IDP 令牌

使用红帽构建的 **Keycloak**，您可以使用 **IDP** 的设置页面中的 **Store Token** 配置选项存储来自外部 **IDP** 的身份验证流程的令牌和响应。

应用程序代码可以检索这些令牌和响应，以导入额外的用户信息或安全地请求外部 **IDP**。例如，应用可以使用 **Google** 令牌来使用其他 **Google** 服务和 **REST API**。要检索特定身份提供程序的令牌，请按如下所示发送请求：

```
GET /realms/{realm}/broker/{provider_alias}/token HTTP/1.1
Host: localhost:8080
Authorization: Bearer <KEYCLOAK ACCESS TOKEN>
```

应用程序必须通过红帽构建的 **Keycloak** 进行身份验证并接收访问令牌。此访问令牌必须具有 代理 客

*户端级别的角色 读取令牌，因此用户必须具有此角色的角色映射，并且客户端应用程序在其范围内必须具有该角色。在这种情况下，由于您在红帽构建的 **Keycloak** 中访问受保护的服务，请在用户身份验证过程中发送由红帽构建 **Keycloak** 发布的访问令牌。您可以通过将 **Stored Tokens Readable** 开关设置为 **ON**，将此角色分配给代理配置页面中的新用户。*

*这些外部令牌可以通过提供程序再次登录或使用客户端发起的帐户链接 **API** 来重新建立。*

### *9.12. 身份代理退出*

*注销时，红帽构建的 **Keycloak** 会向外部身份提供程序发送请求，该供应商最初用于登录，并从此身份提供程序注销。您可以跳过此行为，并避免从外部身份提供程序注销。如需更多信息，请参阅 适配器注销文档。*

# 第 10 章 SSO 协议

本节讨论红帽构建的 Keycloak 身份验证服务器，以及由红帽构建的 Keycloak 身份验证服务器保护应用程序的方式，并与这些协议进行交互。

## 10.1. OPENID CONNECT

OpenID Connect (OIDC)是一个身份验证协议，它是 OAuth 2.0 的扩展。

OAuth 2.0 是用于构建授权协议的框架，不完整。但是，OIDC 是使用 Json Web Token (JWT)标准的完整身份验证和授权协议。JWT 标准定义了身份令牌 JSON 格式和方法，以便以紧凑和 Web 友好的方式对数据进行数字签名和加密。

通常，OIDC 实现了两个用例。第一个情形是请求红帽构建的 Keycloak 服务器验证用户的应用程序。成功登录后，应用程序会收到 身份令牌和访问令牌。身份令牌 包含用户信息，包括用户名、电子邮件和配置文件信息。域以数字方式签署 访问令牌，其中包含应用程序用来确定用户可在应用中访问的资源信息（如用户角色映射）。

第二个用例是访问远程服务的客户端。

- 客户端从红帽构建的 Keycloak 请求访问令牌，代表用户对远程服务调用。
- 红帽构建的 Keycloak 验证用户，并要求用户同意授予请求客户端的访问权限。
- 客户端接收由域数字签名的访问令牌。
- 客户端使用 访问令牌 在远程服务上发出 REST 请求。
- 远程 REST 服务提取 访问令牌。
- 远程 REST 服务验证令牌签名。

- 远程 **REST** 服务根据令牌内的访问信息决定，以处理或拒绝请求。

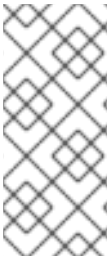
### 10.1.1. OIDC 身份验证流

**OIDC** 有几个方法或流，客户端或应用程序可用于验证用户并接收 身份和 访问令牌。该方法取决于请求访问的应用或客户端的类型。

#### 10.1.1.1. 授权代码流

授权代码流是一个基于浏览器的协议，适合验证和授权基于浏览器的应用程序。它使用浏览器重定向来获取身份和访问令牌。

1. 用户使用浏览器连接到应用程序。应用程序检测到用户没有登录到应用程序。
2. 应用程序将浏览器重定向到红帽构建的 **Keycloak** 进行验证。
3. 应用在浏览器重定向中作为查询参数传递回调 **URL**。红帽构建的 **Keycloak** 在身份验证成功后使用该参数。
4. 红帽构建的 **Keycloak** 验证用户，并创建一个一次性、短期的临时代码。
5. **Red Hat build of Keycloak** 使用回调 **URL** 重定向到应用程序，并在回调 **URL** 中添加临时代码作为查询参数。
6. 应用程序提取临时代码，并对红帽构建的 **Keycloak** 发出后台 **REST** 调用，以交换 身份和访问和 刷新令牌 的代码。为防止重播攻击，无法多次使用临时代码。



注意

在令牌的生命周期中，系统会受到 **stolen** 令牌的影响。出于安全性和可扩展性的原因，访问令牌通常设置为快速过期，因此后续令牌请求会失败。如果令牌过期，应用程序可以使用登录协议发送的额外 刷新令牌 来获取新的访问令牌。

机密 客户端在为令牌交换临时代码时提供客户端机密。不需要公共客户端来提供客户端 **secret**。当

*HTTPS 被严格强制时，公共客户端安全，并且严格控制为客户端注册的 URI。HTML5/JavaScript 客户端必须是 公共 客户端，因为无法安全地将客户端机密传输到 HTML5/JavaScript 客户端。如需了解更多详细信息，请参阅管理客户端 章节。*

*红帽构建的 Keycloak 还支持代码 交换规范的概念验证。*

#### *10.1.1.2. 隐式流*

*Implicit 流是一个基于浏览器的协议。它与 Authorization Code 流类似，但请求较少，且没有刷新令牌。*

*注意*

*当令牌通过重定向 URI 传输时，在浏览器历史记录中可能会泄漏访问令牌（请参阅以下）。*

*另外，这个流没有为客户端提供刷新令牌。因此，访问令牌必须长期有效，或者用户必须在过期时重新验证。*

*我们不推荐使用这个流。支持这个流，因为它位于 OIDC 和 OAuth 2.0 规格中。*

*协议按如下方式工作：*

1. *用户使用浏览器连接到应用程序。应用程序检测到用户没有登录到应用程序。*
2. *应用程序将浏览器重定向到红帽构建的 Keycloak 进行验证。*
3. *应用在浏览器重定向中作为查询参数传递回调 URL。红帽构建的 Keycloak 在成功身份验证时使用查询参数。*
4. *红帽构建的 Keycloak 验证用户，并创建一个 身份和 访问令牌。Red Hat build of Keycloak 使用回调 URL 重定向到应用程序，并额外在回调 URL 中添加 身份和 访问令牌 作为查询参数。*

5. *应用从回调 URL 中提取 身份和 访问令牌。*

#### *10.1.1.3. 资源所有者密码凭证授予（直接访问授予授予）*

*REST 客户端使用 直接访问授予授予 代表用户获取令牌。它是 HTTP POST 请求，其中包含：*

- *用户的凭据。凭据以表单参数的形式发送。*
- *客户端的 id。*
- *客户端机密（如果是机密客户端）。*

*HTTP 响应包含 身份、访问和 刷新令牌。*

#### *10.1.1.4. 客户端凭证授权*

*Client Credentials Grant 根据与客户端关联的服务帐户的元数据和权限创建一个令牌，而不是获取代表外部用户的令牌。REST 客户端使用客户端凭证授予。*

*如需更多信息，请参阅服务帐户章节。*

### *10.1.2. 刷新令牌授权*

*默认情况下，红帽构建的 Keycloak 在令牌响应中返回刷新令牌。一些例外是隐式流或客户端凭证授予上述。*

*刷新令牌与 SSO 浏览器会话的用户会话关联，并在用户会话的生命周期内有效。但是，该客户端应为每个指定间隔至少发送一次 refresh-token 请求。否则，会话可以被视为"idle"，并可过期。如需更多信息，请参阅 超时部分。*

*Red Hat build of Keycloak 支持 离线令牌，在客户端需要使用刷新令牌时，即使对应的浏览器 SSO 会话已经过期，也可以使用它。*

#### *10.1.2.1. 刷新令牌轮转*

*可以指定刷新令牌在被使用后被视为无效。这意味着，客户端必须始终从最后一次刷新响应保存刷新令牌，因为已经使用的旧刷新令牌，红帽构建的 **Keycloak** 不再被视为有效。这可以通过使用 **Revoke Refresh token** 选项设置，如 **timeout** 部分中指定的。**???***

*红帽构建的 **Keycloak** 还支持没有刷新令牌轮转的情况。在这种情况下，在登录期间返回刷新令牌，但刷新令牌请求的后续响应不会返回新的刷新令牌。建议在 **FAPI 2** 草案规格中 为实例提供这种方法。在红帽构建的 **Keycloak** 中，可以使用 客户端策略 跳过刷新令牌轮转。您可以将 **executor suppress-refresh-token-rotation** 添加到某些客户端配置集，并将客户端策略配置为触发配置集，这意味着将跳过刷新令牌轮转的客户端。*

#### *10.1.2.2. 设备授权授权*

*这供运行在有有限输入功能或缺少适当浏览器的互联网连接设备上使用。以下是协议的简要概述：*

1. *应用程序请求红帽构建 **Keycloak**，一个设备代码和用户代码。红帽构建的 **Keycloak** 创建一个设备代码和用户代码。红帽构建的 **Keycloak** 会向应用程序返回包括设备代码和用户代码的响应。*

2. *应用为用户提供用户代码和验证 **URI**。用户访问验证 **URI** 以供使用其他浏览器进行身份验证。您可以定义一个简短的 **verification_uri**，它将被重定向到红帽构建的 **Keycloak** 验证 **URI (/realms/realm_name/device) outside Red Hat build of Keycloak - fe on a proxy**。*

3. *应用程序会重复轮询红帽构建的 **Keycloak**，以查找用户是否已完成用户授权。如果完成了用户身份验证，应用程序会交换 身份、访问和刷新令牌的设备代码。*

#### *10.1.2.3. 客户端启动的后端通道身份验证授权*

*此功能供希望通过直接与 **OpenID** 提供程序通信来启动身份验证流的客户端使用，而无需通过用户浏览器（如 **OAuth 2.0** 的授权代码授权）进行重定向。以下是协议的简要概述：*

1. *客户端请求红帽构建的 **Keycloak** 是一个 **auth_req_id**，用于标识客户端发出的身份验证请求。红帽构建的 **Keycloak** 创建 **auth_req_id**。*

2. *收到这个 **auth_req_id** 后，此客户端会重复轮询红帽构建的 **Keycloak**，以获取来自红帽构建的 **Keycloak** 的访问令牌、刷新令牌和 **ID** 令牌，以返回 **auth_req_id**，直到用户被验证为止。*

*管理员可以为每个域将客户端初始通道身份验证(CIBA)相关操作配置为 **CIBA** 策略。*

*另请参阅，请参阅红帽构建的 **Keycloak** 文档的其他位置，如保护应用程序和服务指南的 **back channel Authentication Endpoint** 部分，以及保护应用程序和服务指南的 **Client Initiated Backchannel Authentication Grant** 部分。*

#### *10.1.2.3.1. CIBA 策略*

*管理员在 管理控制台 上执行以下操作：*

- *打开 **Authentication** → **CIBA Policy** 选项卡。*
- *配置项目并点 **Save**。*

*可配置的项目及其描述如下。*

| Configuration | 描述 |
| --- | --- |
| 后端令牌交付模式 | 指定 CD （持续设备）如何获取身份验证结果和相关令牌。有三种模式："poll", "ping" 和 "push"。红帽构建的 Keycloak 只支持 "poll"。默认设置为 "poll"。这个配置是必需的。如需了解更多详细信息，请参阅 CIBA 规范。 |
| 过期 | 收到身份验证请求的 "auth_req_id" 的过期时间（以秒为单位）。默认设置为 120。这个配置是必需的。如需了解更多详细信息，请参阅 CIBA 规范。 |
| Interval（间隔） | CD （跳过设备）在轮询请求到令牌端点之间需要等待的时间（以秒为单位）。默认设置为 5。此配置是可选的。如需了解更多详细信息，请参阅 CIBA 规范。 |
| 身份验证请求的用户 Hint | 识别请求身份验证的最终用户的方法。默认设置为 "login_hint"。有三种模式 "login_hint", "login_hint_token" 和 "id_token_hint"。红帽构建的 Keycloak 只支持 "login_hint"。这个配置是必需的。如需了解更多详细信息，请参阅 CIBA 规范。 |

#### *10.1.2.3.2. 供应商设置*

*CIBA 授权使用以下两个供应商。*

1. *身份验证频道供应商 ：提供红帽构建的 Keycloak 和实际通过 AD （身份验证设备）验证用户的实体之间的通信。*

2. *用户 Resolver Provider：从客户端提供的信息获取红帽构建的 Keycloak UserModel，以识别用户。*

*红帽构建的 Keycloak 具有这两个默认供应商。但是，管理员需要设置身份验证频道提供程序，如下所示：*

```
kc.[sh|bat] start --spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri=https://backend.internal.example.com
```

*可配置的项目及其描述如下。*

| Configuration | 描述 |
| --- | --- |
| http-authentication-channel-uri | 指定实际通过 AD （身份验证设备）验证用户的实体的 URI。 |

#### *10.1.2.3.3. 身份验证频道供应商*

*CIBA 标准文档没有指定 AD 如何验证用户。因此，它可能会在产品自由裁量实施。红帽构建的 Keycloak 将这个身份验证委托给外部身份验证实体。为了与身份验证实体通信，红帽构建的 Keycloak 提供身份验证频道提供程序。*

*其红帽构建的 Keycloak 实施假定身份验证实体由红帽构建的 Keycloak 控制下，以便红帽构建的 Keycloak 信任身份验证实体。不建议使用红帽构建的 Keycloak 无法控制的身份验证实体。*

*身份验证频道提供程序作为 SPI 供应商提供，以便红帽构建的 Keycloak 用户可以实施自己的供应商，以满足其环境。红帽构建的 Keycloak 提供名为 HTTP Authentication Channel Provider 的默认供应商，该提供程序使用 HTTP 与身份验证实体通信。*

*如果红帽构建的 Keycloak 用户希望使用 HTTP Authentication Channel Provider，则需要知道红帽构建的 Keycloak 和由以下两个部分组成的身份验证实体之间的合同。*

*身份验证委托请求/**Response***

*红帽构建的 **Keycloak** 将身份验证请求发送到身份验证实体。*

*身份验证结果通知/**ACK***

*身份验证实体会通知红帽构建的 **Keycloak** 身份验证的结果。*

*身份验证委托 **Request/Response** 由以下消息传递组成：*

*身份验证委托请求*

*该请求从红帽构建的 **Keycloak** 发送到身份验证实体，以要求它供 **AD** 进行身份验证。*

> *POST [delegation_reception]*

- ***Headers***

| Name | 值 | 描述 |
| --- | --- | --- |
| Content-Type | application/json | 消息正文格式为 json。 |
| 授权 | bearer [token] | 当身份验证实体通知红帽构建的 Keycloak 身份验证结果时，会使用 [token]。 |

- *参数*

| 类型 | Name | 描述 |
| --- | --- | --- |
| 路径 | delegation_reception | 身份验证实体提供的端点，以接收委托请求 |

- ***Body***

| Name | 描述 |
| --- | --- |

| Name | 描述 |
| --- | --- |
| login_hint | 它告知身份验证实体由 AD 进行身份验证。<br>默认情况下，它是用户的"username"。<br>此字段是必需的，并由 CIBA 标准文档定义。 |
| scope | 它告知身份验证实体从经过身份验证的用户获得同意的范围。<br>此字段是必需的，并由 CIBA 标准文档定义。 |
| is_consent_required | 它显示身份验证实体是否需要从经过身份验证的用户获得有关该范围的同意。<br>此字段是必需的。 |
| binding_message | 其值应该在 CD 和 AD 的 UI 中显示，以便用户识别 AD 的身份验证是由 CD 触发的。<br>此字段是可选的，并由 CIBA 标准文档定义。 |
| acr_values | 它告知从 CD 请求的身份验证上下文类参考。<br>此字段是可选的，并由 CIBA 标准文档定义。 |

*身份验证委托响应*

*响应从身份验证实体返回到红帽构建的 **Keycloak**，以通知身份验证实体从红帽构建的 **Keycloak** 收到身份验证请求。*

- *响应*

| HTTP 状态代码 | 描述 |
| --- | --- |
| 201 | 它通知红帽构建的 Keycloak 接收身份验证委托请求。 |

*身份验证结果通知/**ACK** 由以下消息传递组成：*

*身份验证结果通知*

*身份验证实体向红帽构建的 **Keycloak** 发送身份验证请求的结果。*

```
POST /realms/[realm]/protocol/openid-connect/ext/ciba/auth/callback
```

- ***Headers***

| Name | 值 | 描述 |
| --- | --- | --- |
| Content-Type | application/json | 消息正文格式为 json。 |
| 授权 | bearer [token] | [token] 必须是身份验证实体在 Authentication Delegation Request 中从红帽构建的 Keycloak 接收的身份验证实体。 |

- *参数*

| 类型 | Name | 描述 |
| --- | --- | --- |
| 路径 | realm | realm 名称 |

- ***Body***

| Name | 描述 |
| --- | --- |
| status | 它告知 AD 用户身份验证的结果。<br>它必须是以下状态之一：<br>SUCCEED : AD 的身份验证已成功完成。<br>UNAUTHORIZED : AD 的身份验证尚未完成。<br>CANCELLED：由 AD 的身份验证已被用户取消。 |

*身份验证结果 **ACK***

*从红帽构建的 **Keycloak** 返回响应到身份验证实体，以通知红帽构建的 **Keycloak** 收到来自身份验证实体的 **AD** 进行用户身份验证的结果。*

- *响应*

| HTTP 状态代码 | 描述 |
| --- | --- |
| 200 | 它通知了接收身份验证结果通知的身份验证实体。 |

#### 10.1.2.3.4. 用户 Resolver 提供程序

即使同一用户，其表示可能在每个 CD 中有所不同，Red Hat build of Keycloak 和身份验证实体也是如此。

对于 CD，红帽构建的 Keycloak 和身份验证实体用于识别同一用户，此用户 Resolver 提供程序会转换他们自己的用户表示形式。

用户 Resolver 提供程序作为 SPI 供应商提供，以便红帽构建的 Keycloak 用户可以实施自己的供应商来满足其环境。红帽构建的 Keycloak 提供名为 Default User Resolver Provider 的默认供应商，它具有以下特征：

- 只支持 login_hint 参数，用作默认参数。
- 红帽构建的 Keycloak 中的 UserModel 用户名 用于代表 CD 上的用户、红帽构建的 Keycloak 和身份验证实体。

### 10.1.3. OIDC Logout

OIDC 有四个与注销机制相关的规格：

1. 会话管理
2. RP-Initiated Logout
3. front-Channel Logout
4. Back-Channel Logout

同样，因为 OIDC 规格在 OIDC 规格中进行了描述，因此我们只会在此处提供简短概述。

#### 10.1.3.1. 会话管理

这是基于浏览器的注销。应用程序定期从红帽构建的 **Keycloak** 获取会话状态信息。当会话在红帽构建的 **Keycloak** 时终止时，应用程序会注意到并触发其自身的注销。

#### 10.1.3.2. RP-Initiated Logout

这也是基于浏览器的注销，退出开始将用户重定向到红帽构建的 **Keycloak** 上的特定端点。当用户点击一些应用程序页面中的 **Log Out** 链接时，通常会发生这个重定向，之前使用 **Red Hat build of Keycloak** 来验证用户。

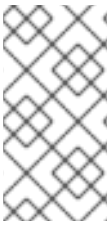
用户重定向到 **logout** 端点后，红帽构建的 **Keycloak** 将向客户端发送注销请求，从而使他们在本地用户会话无效，并在注销过程完成后可能会将用户重定向到一些 **URL**。当用户没有使用 **id_token_hint** 参数时，可能会有可选要求确认 **logout**。注销后，用户会自动重定向到指定的 **post_logout_redirect_uri**，只要它作为一个参数提供。请注意，如果包含 **post_logout_redirect_uri**，则需要包含 **client_id** 或 **id_token_hint** 参数。另外，**post_logout_redirect_uri** 参数需要与客户端配置中指定的 **Valid Post Logout Redirect URI** 之一匹配。

根据客户端配置，可以通过前端通道或通过后端通道向客户端发送注销请求。对于依赖于上一节中描述的会话管理前端浏览器客户端，红帽构建的 **Keycloak** 不需要向它们发送任何注销请求；这些客户端会自动检测浏览器中的 **SSO** 会话。

#### 10.1.3.3. front-channel Logout

要将客户端配置为通过前端通道接收注销请求，请查看 **Front-Channel Logout** 客户端设置。使用此方法时，请考虑以下几点：

- 注销由红帽构建的 **Keycloak** 发送到客户端的请求依赖浏览器，并在为注销页面呈现的嵌入式 **iframes**。
- 通过基于 **iframes**，前端通道注销可能会受到内容安全策略(**CSP**)的影响，并且注销请求可能会阻止。
- 如果用户在渲染 **logout** 页面或实际发送到客户端之前关闭浏览器，则客户端中的会话可能无法无效。



注意

考虑使用 **Back-Channel Logout**，因为它提供了一种更加可靠且安全的方法来注销用户并在客户端上终止其会话。

如果没有通过前端注销启用客户端，则红帽构建的 **Keycloak** 将首先尝试使用 **Back-Channel Logout URL** 通过 **back-channel** 发送注销请求。如果没有定义，服务器将回退到使用 **Admin URL**。

#### 10.1.3.4. Backchannel Logout

这是一个基于非浏览器的注销，它使用红帽构建的 **Keycloak** 和客户端之间的直接后备通道通信。**Red Hat build of Keycloak** 发送 **HTTP POST** 请求，其中包含一个注销令牌到登录到红帽构建的 **Keycloak** 的所有客户端。这些请求在 **Red Hat build of Keycloak** 中发送到注册的 **backchannel logout URL**，应该会在客户端触发注销。

### 10.1.4. 红帽构建的 Keycloak 服务器 OIDC URI 端点

以下是红帽构建的 **Keycloak** 发布的 **OIDC** 端点列表。当非红帽构建的 **Keycloak** 客户端适配器使用 **OIDC** 与身份验证服务器通信时，可以使用这些端点。它们都是相对 **URL**。**URL** 的根由 **HTTP (S)**协议、主机名和可选的路径组成：例如：

> https://localhost:8080

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

用于在授权代码流中获取临时代码，或使用 **Implicit Flow**、**Direct Grants** 或 **Client Grants** 获取令牌。

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

授权代码流使用将临时代码转换为令牌。

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

用于执行退出。

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

用于 **OIDC** 规格中描述的用户信息服务。

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

用于 **RFC7009** 中描述的 **OAuth 2.0** 令牌撤销。

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

用于 **JSON Web** 密钥集**(JWKS)**，其中包含用于验证任何 **JSON Web** 令牌**(jwks_uri)**的公钥。

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

用于设备授权授权，以获取设备代码和用户代码。

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

这是 **Client Initiated Backchannel Authentication Grant** 的 **URL** 端点，以获取一个 **auth_req_id**，用于标识客户端发出的身份验证请求。

**/realms/{realm-name}/protocol/openid-connect/logout/backchannel-logout**

这是执行 **OIDC** 规格中描述的 **backchannel logouts** 的 **URL** 端点。

在所有这些版本中，将 **{realm-name}** 替换为域的名称。

### 10.2. SAML

SAML 2.0 是与 **OIDC** 类似的规格，但更成熟。它源自 **SOAP** 和 **Web** 服务消息传递规格，因此通常比 **OIDC** 更详细。**SAML 2.0** 是一个身份验证协议，可在身份验证服务器和应用程序间交换 **XML** 文档。**XML** 签名和加密用于验证请求和响应。

通常，**SAML** 实施两个用例。

第一个用例是请求红帽构建的 **Keycloak** 服务器验证用户的应用程序。成功登录后，应用程序将收到 **XML** 文档。本文档包含一个 **SAML** 断言，用于指定用户属性。域以数字签名为包含访问信息（如用户角色映射）的文档，用于确定应用程序中允许用户访问的资源。

第二个用例是访问远程服务的客户端。客户端从红帽构建的 **Keycloak** 请求 **SAML** 断言，代表用户调用远程服务。

#### 10.2.1. SAML 绑定

红帽构建的 **Keycloak** 支持三种绑定类型。

##### 10.2.1.1. 重定向绑定

重定向 绑定使用一系列浏览器重定向 **URI** 来交换信息。

1. 用户使用浏览器连接到应用程序。应用程序检测到用户没有经过身份验证。

2. 应用程序生成 **XML** 身份验证请求文档，并在 **URI** 中将其编码为查询参数。**URI** 用于重定向到红帽构建的 **Keycloak** 服务器。根据您的设置，应用程序也可以数字签名 **XML** 文档，并将签名作为查询参数包含在将 **URI** 重定向到红帽 **Keycloak**。此签名用于验证发送请求的客户端。

3. 浏览器会重定向到红帽 **Keycloak** 的构建。

4. 服务器提取 **XML** 身份验证请求文档，并在需要时验证数字签名。

5. 用户输入其身份验证凭据。

6. 身份验证后，服务器会生成 **XML** 身份验证响应文档。文档包含一个 **SAML** 断言，其中包含有关用户的元数据，包括用户的名称、地址、电子邮件以及用户具有的任何角色映射。文档通常使用 **XML** 签名进行数字签名，也可以加密。

7. **XML** 身份验证响应文档在重定向 **URI** 中编码为查询参数。**URI** 使浏览器回到应用。数字签名也作为查询参数包含。

8. 应用程序接收重定向 **URI** 并提取 **XML** 文档。

9. 应用程序会验证域的签名，以确保它收到有效的身份验证响应。**SAML** 断言中的信息用于做出访问决策或显示用户数据。

#### *10.2.1.2. POST 绑定*

**POST** 绑定与 **Redirect** 绑定类似，但 **POST** 绑定使用 **POST** 请求而不是使用 **GET** 请求来交换 **XML** 文档。在交换文档时，**POST Binding** 使用 **JavaScript** 将 **POST** 请求发送到红帽构建的 **Keycloak** 服务器或应用程序。**HTTP** 以 **HTML** 文档响应，其中包含嵌入式 **JavaScript** 的 **HTML** 表单。当页面加载时，**JavaScript** 会自动调用表单。

建议因为两个限制而发布 **POST** 绑定：

- **security wagon-categoriesWith Redirect** 绑定，**SAML** 响应是 **URL** 的一部分。在日志中

*捕获响应是不太安全的。*

- ***HTTP** 有效负载中的文档 大小 是多于有限 **URL** 的大量数据范围。*

#### *10.2.1.3. ECP*

*增强的客户端或代理**(ECP)**是一个 **SAML v.2.0** 配置集，它允许在 **Web** 浏览器上下文外交换 **SAML** 属性。它通常由基于 **REST** 或 **SOAP** 的客户端使用。*

### *10.2.2. Red Hat build of Keycloak Server SAML URI Endpoints*

*红帽构建的 **Keycloak** 为所有 **SAML** 请求有一个端点。*

***http(s)://authserver.host/realms/{realm-name}/protocol/saml***

*所有绑定都使用此端点。*

## *10.3. OPENID CONNECT 与 SAML 相比*

*下面列出了在选择协议时要考虑的多个因素。*

*在大多数情况下，红帽构建的 **Keycloak** 建议使用 **OIDC**。*

***OIDC***

- ***OIDC** 专门设计用于 **Web**。*
- ***OIDC** 适用于 **HTML5/JavaScript** 应用程序，因为比 **SAML** 更容易在客户端实现。*
- ***OIDC** 令牌采用 **JSON** 格式，使其更易于使用 **Javascript**。*
-

OIDC 具有一些功能，可以更轻松地进行安全实施。例如，请参阅规格用来确定用户登录状态的 iframe 欺骗。

SAML

- SAML 设计为在 Web 之上工作的层。
- SAML 比 OIDC 更加详细。
- 用户通过 OIDC 选择了 SAML，因为存在非常成熟。
- 用户在 OIDC 间选择带有保护它的现有应用程序。

### 10.4. DOCKER REGISTRY V2 身份验证

注意

Docker 身份验证默认为禁用。要启用 docker 身份验证，请参阅 启用和禁用功能 章节。

Docker Registry V2 身份验证 是与 OIDC 类似的协议，它针对 Docker registry 验证用户。红帽构建的 Keycloak 实施允许 Docker 客户端使用红帽构建的 Keycloak 身份验证服务器与 registry 进行身份验证。这个协议使用标准令牌和签名机制，但它与真正的 OIDC 实现分离。它通过对请求和响应使用非常具体的 JSON 格式，并将存储库名称和权限映射到 OAuth 范围机制来分离。

#### 10.4.1. Docker 身份验证流程

身份验证流程在 Docker API 文档 中进行了描述。以下是红帽构建的 Keycloak 身份验证服务器的视角概述：

- 执行 Docker 登录。
- Docker 客户端从 Docker 注册表请求资源。如果资源受保护且没有请求中的身份验证令牌，Docker 注册表服务器会以 401 HTTP 消息响应，其中包含有关所需权限和授权服务器位置

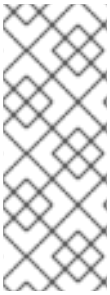
的信息。

- **Docker** 客户端基于 **Docker** 注册表中的 **401 HTTP** 消息构造身份验证请求。客户端使用本地缓存的凭证（来自 **docker login** 命令），作为红帽构建的 **Keycloak** 身份验证服务器的一部分。**https://datatracker.ietf.org/doc/html/rfc2617**

- 红帽构建的 **Keycloak** 身份验证服务器会尝试验证用户并返回包含 **OAuth** 样式 **Bearer** 令牌的 **JSON** 正文。

- **Docker** 客户端从 **JSON** 响应接收 **bearer** 令牌，并在授权标头中使用它请求受保护的资源。

- **Docker registry** 收到带有来自红帽构建的 **Keycloak** 服务器令牌的新请求。**registry** 验证令牌，并授予对请求资源的访问权限（如果适用）。



注意

红帽构建的 **Keycloak** 在通过 **Docker** 协议成功进行身份验证后不会创建浏览器 **SSO** 会话。浏览器 **SSO** 会话不使用 **Docker** 协议，因为它无法刷新令牌或从红帽构建的 **Keycloak** 服务器获取令牌或会话的状态；因此不需要浏览器 **SSO** 会话。如需了解更多详细信息，请参阅 临时会话 部分。

### 10.4.2. Red Hat build of Keycloak Docker Registry v2 Authentication Server URI Endpoints

红帽构建的 **Keycloak** 为所有 **Docker** 身份验证 **v2** 请求有一个端点。

**http(s)://authserver.host/realms/{realm-name}/protocol/docker-v2**

# 第 11 章 控制对管理控制台的访问

在红帽构建的 **Keycloak** 上创建的每个域都有一个专用的管理控制台，可从中管理该域。主域 是一个特殊的域，它允许管理员管理系统上多个域。本章介绍了本情况的所有情况。

## 11.1. MASTER 域访问控制

红帽构建的 **Keycloak** 中的 **master** 域是一个特殊的域，其处理与其他域不同。红帽构建的 **Keycloak master** 域中的用户可以被授予管理红帽构建的 **Keycloak** 服务器上部署零个或更多域的权限。创建域时，红帽构建的 **Keycloak** 会自动创建各种角色，授予精细权限来访问该新域。可以通过将这些角色映射到 **master** 域中的用户来控制对 **Admin Console** 和 **Admin REST** 端点的访问。可以创建多个超级用户，以及只能管理特定域的用户。

### 11.1.1. 全局角色

**master** 域中有两个 **realm** 级别的角色。这些是：

- **admin**
- **create-realm**

具有 **admin** 角色的用户是超级用户，具有管理服务器上任何域的完整访问权限。允许具有 **create-realm** 角色的用户创建新域。它们被授予对他们创建的任何新域的完整访问权限。

### 11.1.2. 域特定角色

**master** 域中的管理员用户可为系统中的一个或多个其他域授予管理特权。红帽构建的 **Keycloak** 中的每个域都由 **master** 域中的客户端表示。客户端的名称是 **< realm name>-realm**。这些客户端各自定义了客户端级角色，它们定义了不同的访问级别来管理单个域。

可用的角色有：

- **view-realm**

- ***view-users***
- ***view-clients***
- ***view-events***
- ***manage-realm***
- ***manage-users***
- ***create-client***
- ***manage-clients***
- ***manage-events***
- ***view-identity-providers***
- ***manage-identity-providers***
- *模拟*

*分配您想要的用户的角色，且他们只能使用管理控制台的特定部分。*

*重要*

*具有 **manage-users** 角色的管理员只能为自己拥有的用户分配 **admin** 角色。因此，如果管理员具有 **manage-users** 角色，但没有 **manage-realm** 角色，则他们将无法分配此角色。*

### *11.2. 专用域管理控制台*

*每个域都有一个专用的管理控制台，可以通过转至 **url /admin/{realm-name}/console** 来访问。该域中的用户可以通过分配特定的用户角色映射来授予域管理权限。*

*每个 **realm** 都有一个内置客户端，称为 **realm-management**。您可以通过转至域的 **Clients left** 菜单项来查看此客户端。此客户端定义指定可授予管理域的权限的客户端级角色。*

- ***view-realm***
- ***view-users***
- ***view-clients***
- ***view-events***
- ***manage-realm***
- ***manage-users***
- ***create-client***
- ***manage-clients***
- ***manage-events***
- ***view-identity-providers***
- ***manage-identity-providers***

- *模拟*

*分配您想要的用户的角色，且他们只能使用管理控制台的特定部分。*

# 第 12 章 管理 OPENID CONNECT 和 SAML 客户端

客户端是可以请求用户身份验证的实体。客户端采用两种形式：第一个客户端类型是希望参与单点登录的应用程序。这些客户端只需要使用红帽构建的 **Keycloak** 为其提供安全性。另一种类型的客户端是请求访问令牌，以便它能够代表经过身份验证的用户调用其他服务。本节讨论配置客户端的各个方面，以及进行它的各种方法。

## 12.1. 管理 OPENID CONNECT 客户端

OpenID Connect 是保护应用程序的建议协议。它设计为使用 **Web** 友好型，它最适合 **HTML5/JavaScript** 应用程序。

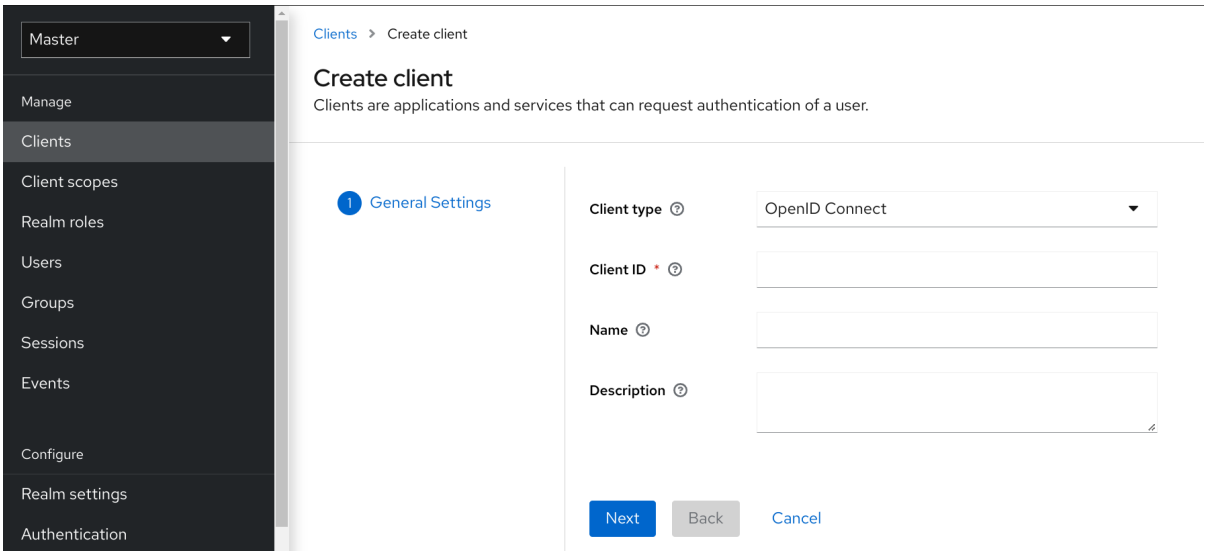
### 12.1.1. 创建 OpenID Connect 客户端

要保护使用 **OpenID** 连接协议的应用程序，您可以创建一个客户端。

流程

1. 点菜单中的 **Clients**。
2. 单击 **Create client**。

创建客户端



3. 将 **Client type** 设置为 **OpenID Connect**。

4. 输入 客户端 **ID**。

   此 **ID** 是一个字母数字字符串，用于 **OIDC** 请求以及红帽构建的 **Keycloak** 数据库来识别客户端。

5. 为客户端提供 名称。

   如果您计划本地化这个名称，请设置替换字符串值。例如，字符串值，如 **${myapp}**。如需更多信息，请参阅 服务器开发人员指南。

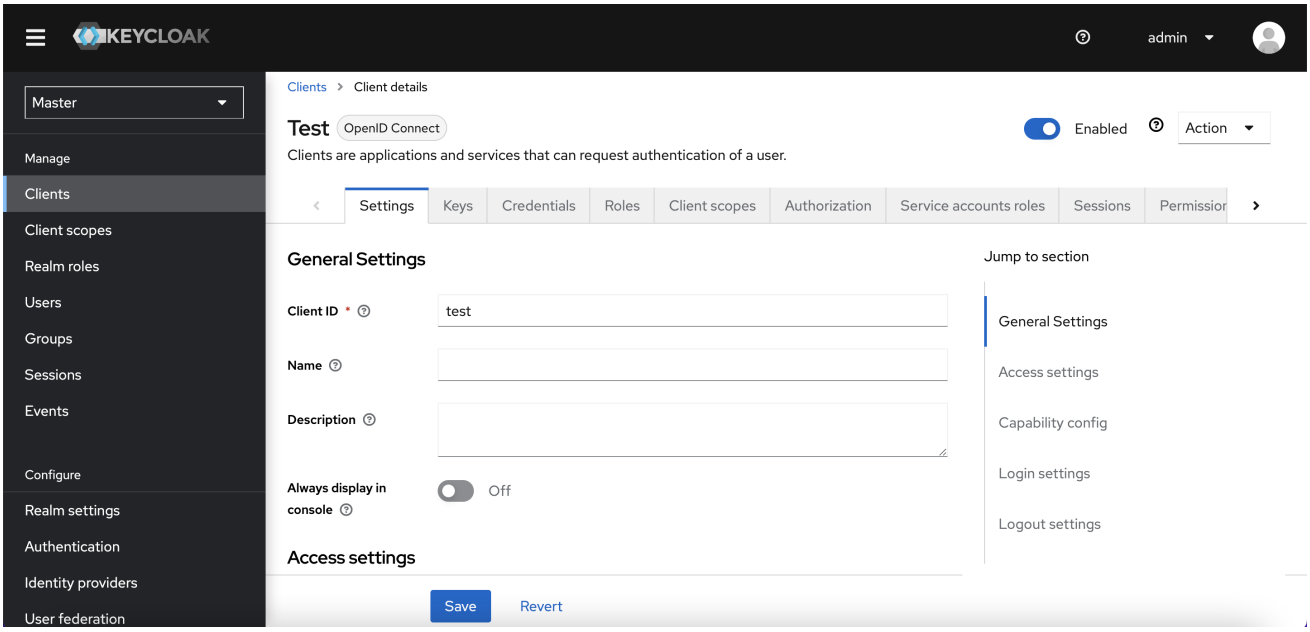
6. 点击 **Save**。

此操作会创建客户端，并将您带到 **Settings** 选项卡，您可以在其中 执行基本配置。

### 12.1.2. 基本配置

**Settings** 选项卡包含许多配置此客户端的选项。

设置标签页



#### 12.1.2.1. 常规设置

客户端 ID

OIDC 请求以及 Red Hat build of Keycloak 数据库用于识别客户端的字母数字字符 ID 字符串。

Name

红帽构建的 Keycloak UI 屏幕中的客户端名称。要本地化名称，请设置替换字符串值。例如，字符串值，如 ${myapp}。如需更多信息，请参阅 服务器开发人员指南。

描述

客户端的描述。此设置也可以是本地化。

始终在控制台中显示

在帐户控制台中始终列出此客户端，即使此用户没有活跃的会话。

#### 12.1.2.2. 访问设置

根 URL

如果红帽构建的 Keycloak 使用任何配置的相对 URL，则该值会添加到其中。

主页 URL

提供当 auth 服务器需要重定向或链接回客户端时的默认 URL。

有效的 Redirect URI

必填字段。输入 URL 模式，然后单击 + 以添加 和 - 以删除现有 URL，然后单击 Save。确切的（区分大小写）字符串匹配用于比较有效的重定向 URI。

您可以在 URL 模式的末尾使用通配符。例如 http://host.com/path/*。为避免安全问题，如果传递的重定向 URI 包含 userinfo 部分或其路径 来管理对 父目录的访问(/../)，则不会执行通配符比较，但标准和安全匹配的字符串匹配。

完整的通配符 * 有效重定向 URI 也可以配置为允许任何 http 或 https 重定向 URI。请不要在生产环境中使用它。

专用重定向 **URI** 模式通常更安全。如需更多信息，请参阅 **Unspecific Redirect URI**。

**Web Origins**

输入 **URL** 模式并点 **+** 添加和 **-** 删除现有 **URL**。点 **Save**。

此选项处理 跨资源共享(CORS)。如果浏览器 **JavaScript** 尝试对域域与 **JavaScript** 代码来自的服务器的 **AJAX HTTP** 请求，则请求必须使用 **CORS**。服务器必须处理 **CORS** 请求，否则浏览器不会显示或允许处理请求。这个协议可防止 **XSS**、**CSRF** 和其他基于 **JavaScript** 的攻击。

此处列出的域 **URL** 嵌入到发送到客户端应用程序的访问令牌中。客户端应用使用此信息决定是否允许在其上调用 **CORS** 请求。只有红帽构建的 **Keycloak** 客户端适配器支持此功能。如需更多信息，请参阅保护应用程序和服务指南。

管理 **URL**

客户端的回调端点。服务器使用此 **URL** 进行回调，如推送撤销策略、执行后备通道注销和其他管理操作。对于红帽构建的 **Keycloak servlet** 适配器，此 **URL** 可以是 **servlet** 应用程序的 **root URL**。如需更多信息，请参阅 保护应用程序和服务指南。

#### 12.1.2.3. 功能配置

客户端身份验证

**OIDC** 客户端的类型。

- **ON**

  对于执行浏览器登录并需要客户端 **secret** 的服务器端客户端，在生成访问令牌请求时。此设置应用于服务器端应用程序。

- **OFF**

  对于执行浏览器登录的客户端。因为无法确保 **secret** 可以与客户端内部客户端保持安全，因此务必要通过配置正确的重定向 **URI** 来限制访问。

授权

*启用或禁用对此客户端的细粒度授权支持。*

*标准流*

*如果启用，这个客户端可以使用 **OIDC** 授权代码流。*

*直接访问授予*

*如果启用，这个客户端可以使用 **OIDC Direct Access Grants**。*

*隐式流*

*如果启用，这个客户端可以使用 **OIDC Implicit Flow**。*

*服务帐户角色*

*如果启用，此客户端可以向红帽构建的 **Keycloak** 进行身份验证，并检索专用于此客户端的访问令牌。在 **OAuth2** 规范中，这启用了为这个客户端 授予 客户端凭证的支持。*

*身份验证 **2.0** 设备授权*

*如果启用，这个客户端可以使用 **OIDC** 设备授权。*

***OIDC CIBA Grant***

*如果启用，这个客户端可以使用 **OIDC Client Initiated Backchannel Authentication Grant**。*

#### *12.1.2.4. 登录设置*

*登录主题*

*用于登录、**OTP**、授予注册和忘记密码页面的主题。*

*需要同意*

*如果启用，用户必须同意客户端访问。*

*对于执行浏览器登录的客户端。因为无法确保 **secret** 可以与客户端内部客户端保持安全，因此务必要通过配置正确的重定向 **URI** 来限制访问。*

*在屏幕上显示客户端*

*如果 **Consent Required is Off**，则应用此开关。*

- 关

  **consent** 屏幕将仅包含与配置的客户端范围对应的同意。

- **On**

  在同意屏幕上也会有一个有关此客户端本身的项目。

客户端同意屏幕文本

如果启用了 **Consent required** 和 **Display client on screen**，则适用。包含此客户端有关权限的同意屏幕上的文本。

#### 12.1.2.5. 注销设置

前端频道注销

如果启用了 **Front Channel Logout**，应用程序应能够根据 OpenID Connect Front-Channel Logout 规格通过前端频道注销用户。如果启用，您还应提供 **Front-Channel Logout URL**。

前端通道注销 **URL**

红帽构建的 **Keycloak** 用于通过前端向客户端发送注销请求的 **URL**。

**Backchannel logout URL**

当一个注销请求发送到这个域（通过 **end_session_endpoint**）时，导致客户端自行注销的 **URL**。如果省略，则不会向客户端发送注销请求。

需要 **backchannel logout** 会话

指定在使用 **Backchannel Logout URL** 时，是否在 **Logout Token** 中包含会话 **ID Claim**。

**Backchannel logout revoke offline session**

指定在使用 **Backchannel Logout URL** 时，是否在 **Logout Token** 中包含 **revoke_offline_access** 事件。当收到带有此事件的 **Logout Token** 时，红帽构建的 **Keycloak** 将撤销离线会话。

### 12.1.3. 高级配置

完成 **Settings** 选项卡上的字段后，您可以使用其他选项卡来执行高级配置。

#### 12.1.3.1. 高级标签页

当您点 **Advanced** 选项卡时，会显示其他字段。有关特定字段的详情，点该字段的问号图标。但是，本节中详细介绍某些字段。

#### 12.1.3.2. 细粒度 OpenID Connect 配置

徽标 **URL**

引用客户端应用程序徽标的 **URL**。

策略 **URL**

**Relying party Client** 提供的 **URL** 提供给 **End-User**，以阅读有关如何使用配置集数据的 **URL**。

服务 **URL** 条款

过期客户端向最终用户提供 **URL**，以阅读有关重新处理服务条款的 **URL**。

签名和加密的 **ID** 令牌支持

红帽构建的 **Keycloak** 可以根据 **Json Web 加密(JWE)规范加密** **ID** 令牌。管理员确定是否为每个客户端加密 **ID** 令牌。

用于加密 **ID** 令牌的密钥是内容加密密钥**(CEK)**。红帽构建的 **Keycloak** 和客户端必须协商使用哪个 **CEK** 以及如何交付它。用于确定 **CEK** 的方法是密钥管理模式。红帽构建的 **Keycloak** 支持的密钥管理模式是密钥加密。

在密钥加密中：

1. *客户端生成非对称加密密钥对。*

2. *公钥用于加密 **CEK**。*

3. *红帽构建的 **Keycloak** 会为每个 **ID** 令牌生成一个 **CEK***

4. *红帽构建的 **Keycloak** 使用生成的 **CEK** 加密 **ID** 令牌*

5. *红帽构建的 **Keycloak** 使用客户端的公钥加密 **CEK**。*

6. *客户端使用其私钥解密此加密的 **CEK***

7. *客户端使用解密的 **CEK** 解密 **ID** 令牌。*

*客户端以外的任何方可以解密 **ID** 令牌。*

*客户端必须通过其公钥将 **CEK** 加密到红帽 **Keycloak** 的构建。红帽构建的 **Keycloak** 支持从客户端提供的 **URL** 下载公钥。客户端必须根据 **Json Web Keys (JWK)** 规范提供公钥。*

*该流程是：*

1. *打开客户端的 **Keys** 选项卡。*

2. *将 **JWKS URL** 切换到 **ON**。*

3. *在 **JWKS URL** 文本框中输入客户端的公钥 **URL**。*

*密钥加密算法在 **Json Web Algorithm (JWA)** 规范中定义。红帽构建的 **Keycloak** 支持：*

- ***RSAES-PKCS1-v1_5(RSA1_5)***
- ***RSAES OAEP** 使用默认参数(**RSA-OAEP**)*
- ***RSAES OAEP 256** 使用 **SHA-256** 和 **MFG1 (RSA-OAEP-256)***

*选择算法的步骤为：*

1. *打开客户端 的高级 选项卡。*
2. *打开 **Fine Grain OpenID Connect** 配置。*
3. *从 **ID Token Encryption Content Algorithm** 下拉菜单中选择算法。*

#### *12.1.3.3. OpenID Connect 兼容性模式*

*本节存在向后兼容。单击问号图标以获取有关每个字段的详细信息。*

*启用 **OAuth 2.0** 通用 **TLS** 证书绑定访问令牌*

*双向 **TLS** 将访问令牌和刷新令牌与客户端证书绑定，该证书在 **TLS** 握手过程中交换。这个绑定可防止攻击者使用 **stolen** 令牌。*

*这种类型的令牌是 **holder-of-key** 令牌。与 **bearer** 令牌不同，**holder-of-key** 令牌的接收者可以验证令牌的发送者是否合法。*

*如果此设置为 **on**，则工作流为：*

1. *令牌请求在授权代码流或混合流中发送到令牌端点。*

2. *红帽构建的 **Keycloak** 请求客户端证书。*

3. *红帽构建的 **Keycloak** 接收客户端证书。*

4. *红帽构建的 **Keycloak** 可以成功验证客户端证书。*

*如果验证失败，红帽构建的 **Keycloak** 会拒绝令牌。*

*在以下情形中，红帽构建的 **Keycloak** 将验证客户端发送访问令牌或刷新令牌：*

- *令牌刷新请求通过 **holder-of-key** 刷新令牌发送到令牌端点。*
- ***UserInfo** 请求被发送到带有 **holder-of-key** 访问令牌的 **UserInfo** 端点。*
- *通过 **holder-of-key** 刷新令牌向非**OIDC** 兼容红帽构建的 **Keycloak** 专有 **Logout** 端点发送一个注销请求。*

*如需了解更多详细信息，请参阅 **OAuth 2.0** 通用 **TLS** 客户端身份验证和证书绑定访问令牌 中的 **Mutual TLS** 客户端证书绑定访问令牌。**https://datatracker.ietf.org/doc/html/draft-ietf-oauth-mtls-08#section-3***

*注意*

*目前，红帽构建的 **Keycloak** 客户端适配器不支持拥有者的密钥令牌验证。红帽构建的 **Keycloak** 适配器将访问和刷新令牌视为 **bearer** 令牌。*

***OAuth 2.0** 在应用层(**DPoP**)上演示概念验证.*

***DPoP** 将访问令牌和刷新令牌与客户端的密钥对的公钥部分绑定。这个绑定可防止攻击者使用 **stolen** 令牌。*

*这种类型的令牌是 **holder-of-key** 令牌。与 **bearer** 令牌不同，**holder-of-key** 令牌的接收者可以验证令牌的发送者是否合法。*

*如果客户端切换 **OAuth 2.0 DPoP Bound Access Tokens Enabled**，则工作流为：*

1. *令牌请求在授权代码流或混合流中发送到令牌端点。*
2. *红帽构建的 **Keycloak** 请求 **DPoP** 证明。*
3. *红帽构建的 **Keycloak** 接收 **DPoP** 证明。*
4. *红帽构建的 **Keycloak** 可以成功验证 **DPoP** 证明。*

*如果验证失败，红帽构建的 **Keycloak** 会拒绝令牌。*

*如果交换机 **OAuth 2.0 DPoP Bound Access Tokens Enabled** 已关闭，客户端仍然可以在令牌请求中发送 **DPoP** 证明。在这种情况下，红帽构建的 **Keycloak** 将验证 **DPoP** 证明，并将 **thumbprint** 添加到令牌中。但是，如果交换机关闭，则此客户端的 **Keycloak** 服务器的红帽构建不会强制使用 **DPoP** 绑定。如果要确保特定客户端始终使用 **DPoP** 绑定，则建议在 上进行这个切换。*

*在以下情形中，红帽构建的 **Keycloak** 将验证客户端发送访问令牌或刷新令牌：*

- *令牌刷新请求通过 **holder-of-key** 刷新令牌发送到令牌端点。此验证仅对公共客户端完成，如 **DPoP** 规范中所述。对于机密客户端，不会作为具有正确客户端凭据的客户端身份验证进行验证，以确保请求来自合法客户端。对于公共客户端，访问令牌和刷新令牌都是 **DPoP** 绑定。对于机密客户端，只有访问令牌是 **DPoP** 绑定。*
- ***UserInfo** 请求被发送到带有 **holder-of-key** 访问令牌的 **UserInfo** 端点。*
- *一个注销请求会被发送到一个非 **OIDC** 兼容红帽构建的 **Keycloak** 专有 **logout** 端点，并带有拥有者**(holder-of-key refresh)**令牌。此验证仅对上述的公共客户端进行。*

*如需了解更多详细信息，请参阅 **OAuth 2.0 演示 Possession (DPoP)**。*

*注意*

*目前，红帽构建的 **Keycloak** 客户端适配器不支持 **DPoP** 持有者的令牌验证。红帽构建的 **Keycloak** 适配器将访问和刷新令牌视为 **bearer** 令牌。*

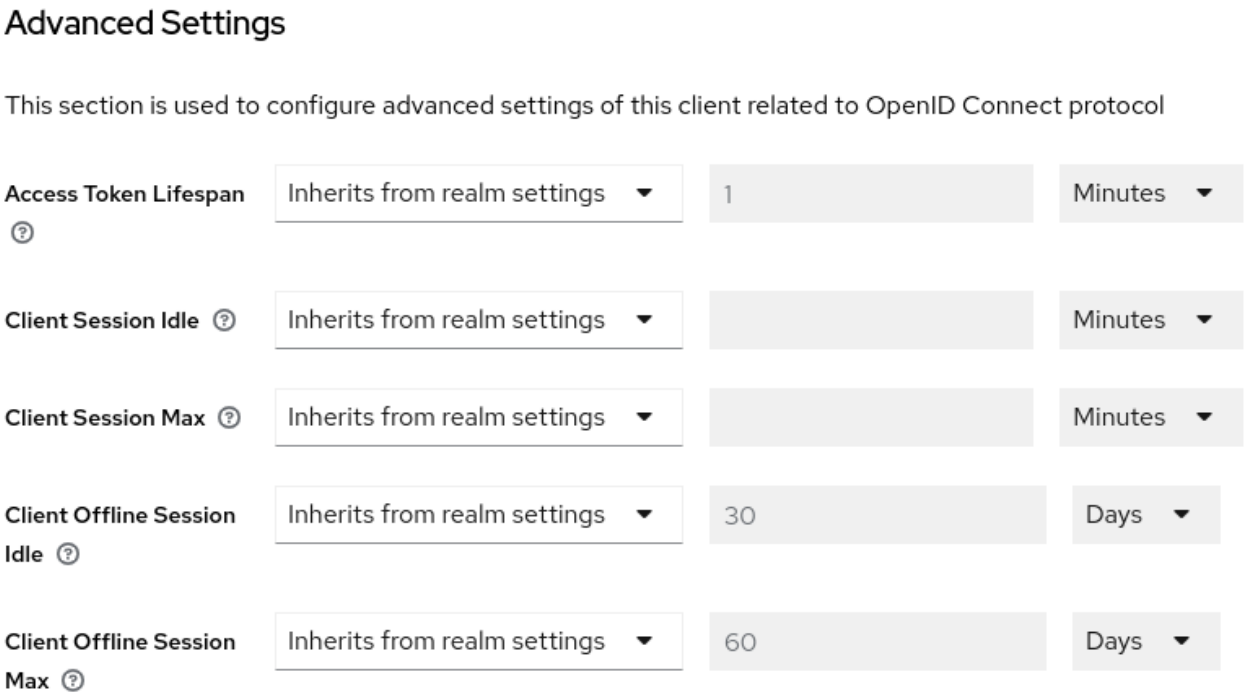
*注意*

***DPoP** 只是一个技术预览，并不被支持。此功能默认为禁用。*

*使用 **--features=preview** 或 **--features=dpop**启动服务器*

***OIDC** 的高级配置*

***OpenID Connect** 的 **Advanced Settings** 允许您在客户端级别上配置 会话和令牌超时的覆盖。*



| Configuration | 描述 |
| --- | --- |
| 访问令牌生命周期 | 该值会覆盖名称相同的 realm 选项。 |
| 客户端会话空闲 | 该值会覆盖名称相同的 realm 选项。该值应小于全局 SSO 会话空闲。 |
| 客户端会话最大数 | 该值会覆盖名称相同的 realm 选项。该值应小于全局 SSO Session Max。 |
| 客户端离线会话空闲 | 此设置允许您为客户端配置较短的离线会话闲置超时。超时是红帽构建 Keycloak 撤销其离线令牌前会话闲置的时间长度。如果没有设置，则使用 realm Offline Session Idle。 |
| 客户端离线会话 Max | 此设置允许您为客户端配置较短的离线会话最大生命周期。生命周期是红帽构建的 Keycloak 撤销对应的离线令牌前的最长时间。这个选项需要在域中全局启用 Session Max Limited，默认为 Offline Session Max。 |

*代码交换代码挑战方法的证明密钥*

*如果攻击者窃取合法客户端的授权代码，则代码交换的概念验证(PKCE)会阻止攻击者获得适用于代码的令牌。*

*管理员可以选择以下选项之一：*

*（空）*

*红帽构建的 **Keycloak** 不适用 **PKCE**，除非客户端向红帽构建的 **Keycloak** 授权端点发送适当的 **PKCE** 参数。*

***S256***

*红帽构建的 **Keycloak** 适用于代码质询方法为 **S256** 的客户端 **PKCE**。*

***plain***

*红帽构建的 **Keycloak** 适用于代码质询方法为 **plain** 的客户端 **PKCE**。*

*如需了解更多详细信息，请参阅 **OAuth** 公共客户端的代码交换的 **RFC 7636** 概念验证。*

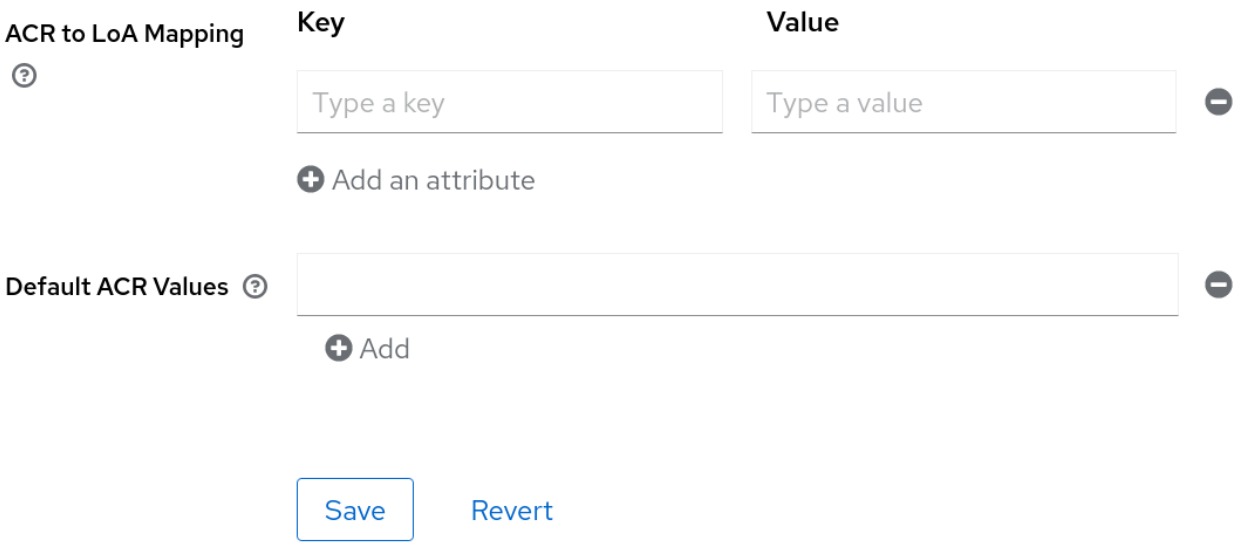
*ACR 到身份验证级别(LoA)映射*

*在客户端的高级设置中，您可以定义哪个 **Authentication Context Class Reference (ACR)** 值映射到哪个 级别的身份验证(**LoA**)。此映射也可以在域中指定，如 **ACR 到 LoA 映射** 中所述。最佳实践是在域级别上配置此映射，它允许在多个客户端间共享相同的设置。*

*当从此客户端发送到红帽构建的 **Keycloak** 时，**Default ACR** 值可以用来指定默认值，但没有附加了 **cr_values** 参数的 **claim** 参数。请参阅官方 **OIDC** 动态客户端注册规格。*

*警告*

*请注意，默认的 **ACR** 值用作默认级别，但它无法可靠地用于强制使用特定级别的登录。例如，假设您将 **Default ACR** 值 配置为级别 **2**。然后，默认情况下，用户需要与级别 **2** 进行身份验证。但是，当用户将参数显式附加到登录请求（如 **acr_values=1** ）时，将使用级别 **1**。因此，如果客户端实际需要 **2** 级，则鼓励客户端检查 **ID Token** 中是否存在 **acr** 声明，并再次检查它是否包含请求的级别 **2**。*

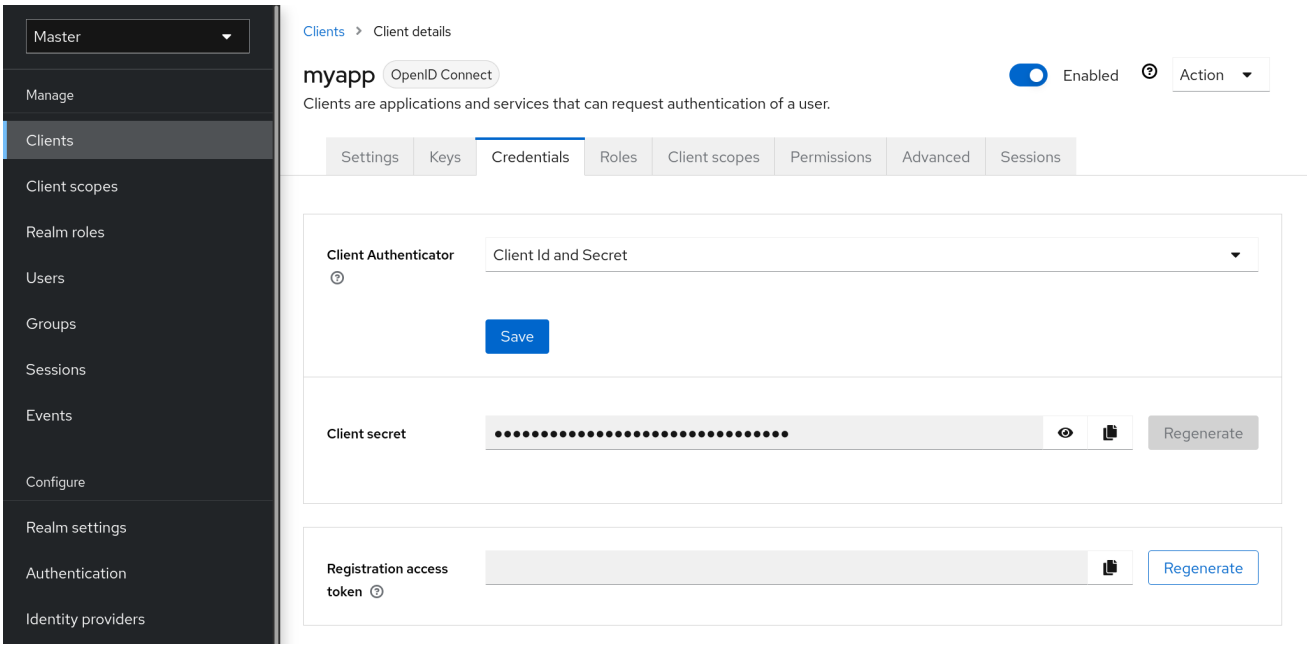



*详情请查看 步骤验证和 官方 **OIDC** 规格。*

### 12.1.4. 机密客户端凭证

*如果客户端的客户端身份验证设置为 **ON**，必须在 **Credentials** 选项卡下配置客户端的凭据。**???***

*凭证标签页*

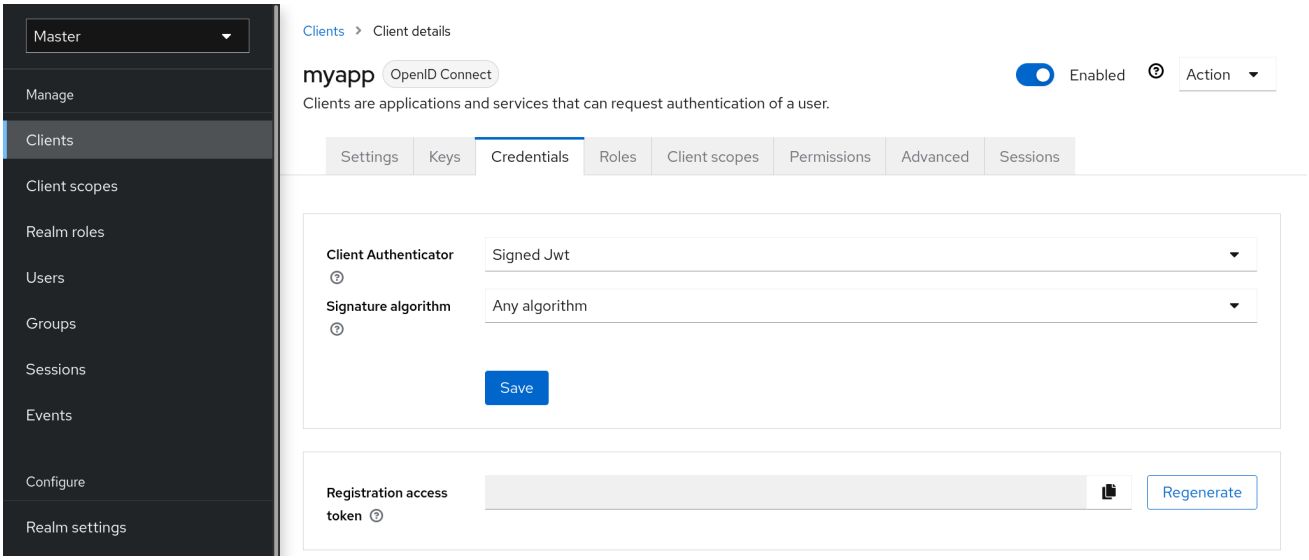



***Client Authenticator** 下拉列表指定用于客户端的凭证类型。*

*客户端 **ID** 和 **Secret***

*这个选择是默认设置。该 **secret** 会自动生成。如果需要，点 **Regenerate** 重新创建 **secret**。*

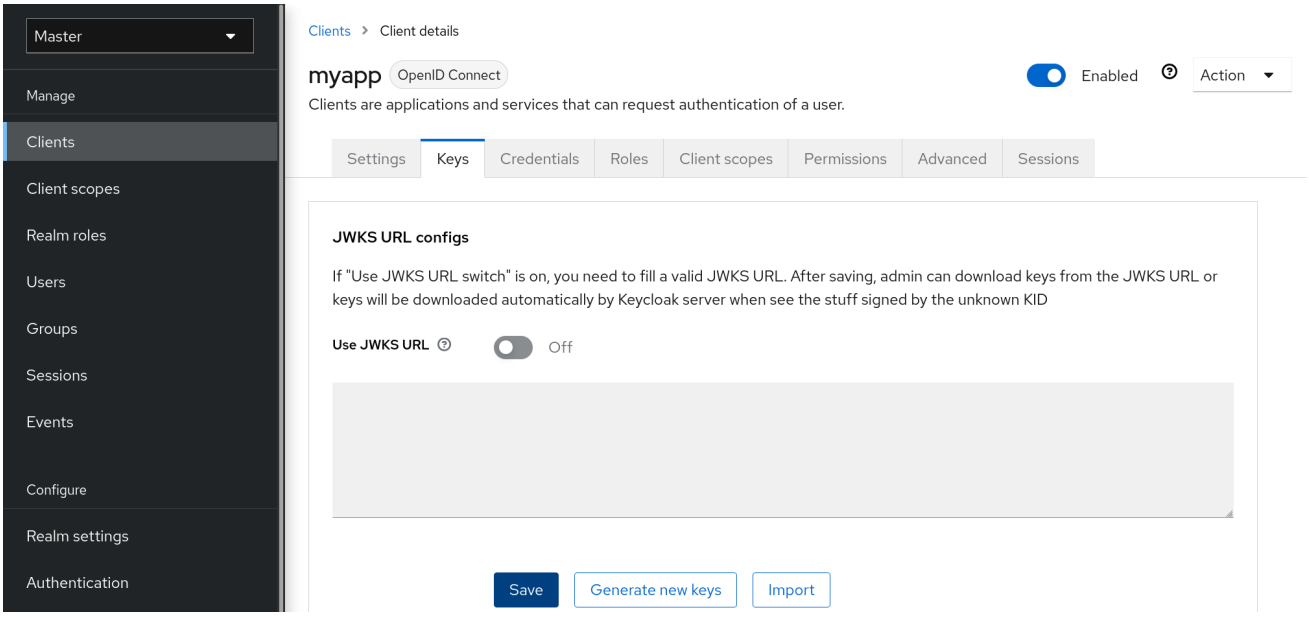
*签名的 **JWT***




*已签名的 **JWT** 是**"Signed Json Web Token"**。*

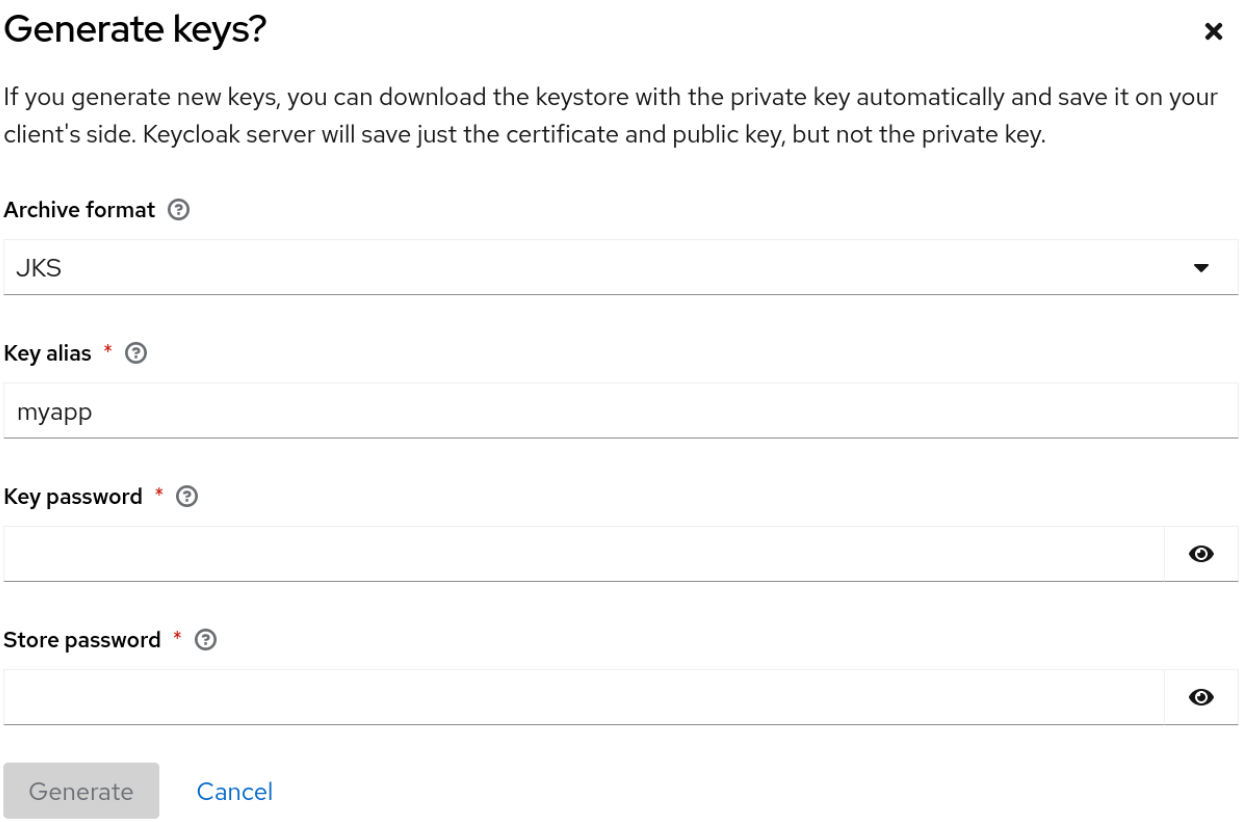
*在选择此凭证类型时，还必须在标签 **Keys** 中为客户端生成私钥和证书。私钥将用于签署 **JWT**，而证书则供服务器用于验证签名。*

***keys** 标签页*




*点 **Generate new keys** 按钮启动此过程。*

*生成密钥*

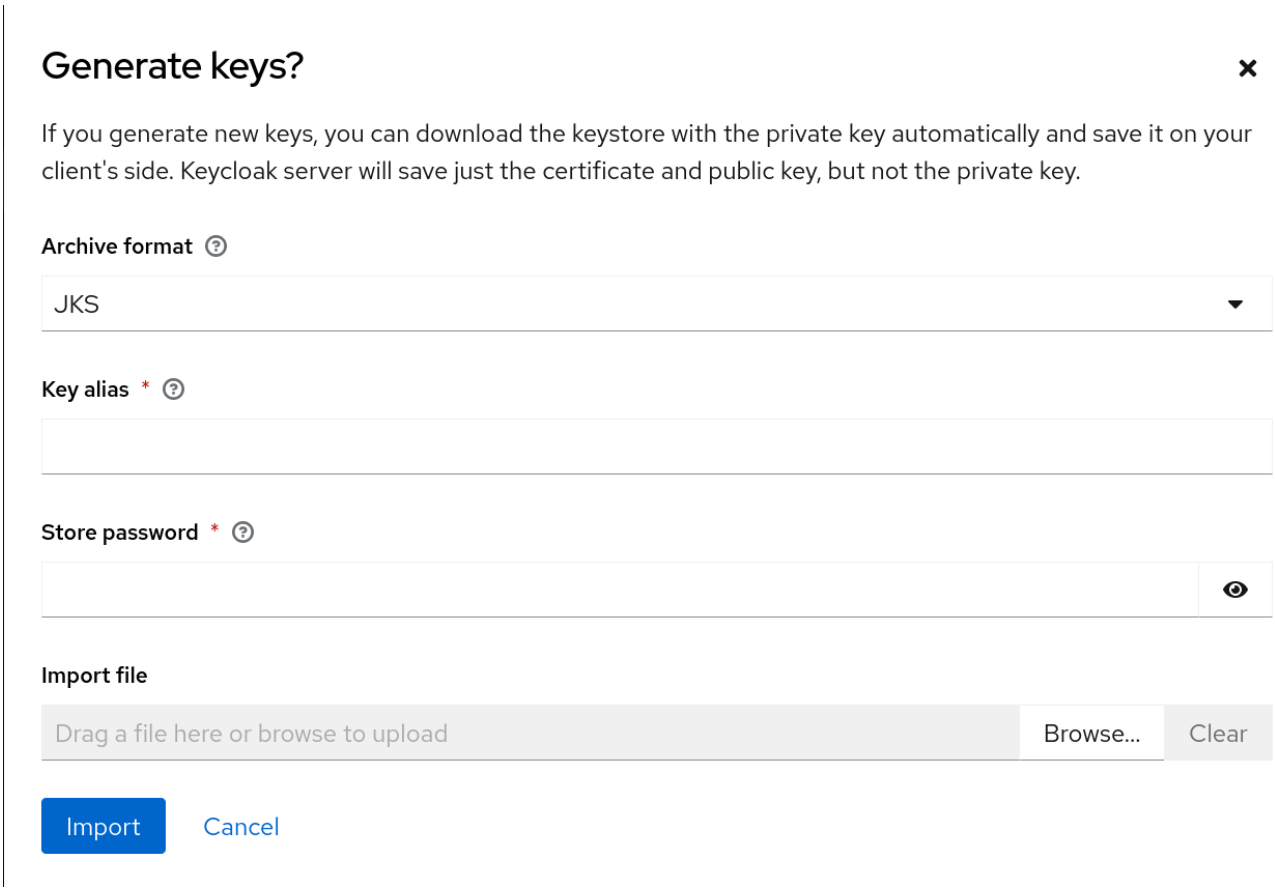



1. *选择要使用的归档格式。*
2. *输入 密钥密码。*
3. *输入 存储密码。*
4. *点 **Generate**。*

*当您生成密钥时，**Red Hat build of Keycloak** 将存储证书，并下载您的客户端的私钥和证书。*

*您还可以使用外部工具生成密钥，然后通过单击 **Import Certificate** 来导入客户端证书。*

*导入证书*




1. *选择证书的存档格式。*
2. *输入存储密码。*
3. *单击 **Import File** 来选择证书文件。*
4. *点 **Import**。*

*如果您点 **Use JWKS URL**，则需要导入证书。在这种情况下，您可以提供以 **JWK** 格式发布公钥的 **URL**。使用此选项时，如果密钥被改变，红帽构建的 **Keycloak** 会重新导入密钥。*

*如果您使用由红帽构建的 **Keycloak** 适配器保护的客户端，您可以使用此格式配置 **JWKS URL**，假设 **https://myhost.com/myapp** 是客户端应用程序的根 **URL**：*

> ***https://myhost.com/myapp/k_jwks***

*如需了解更多详细信息，请参阅 服务器开发人员指南。*

*使用客户端 **Secret** 签名的 **JWT***

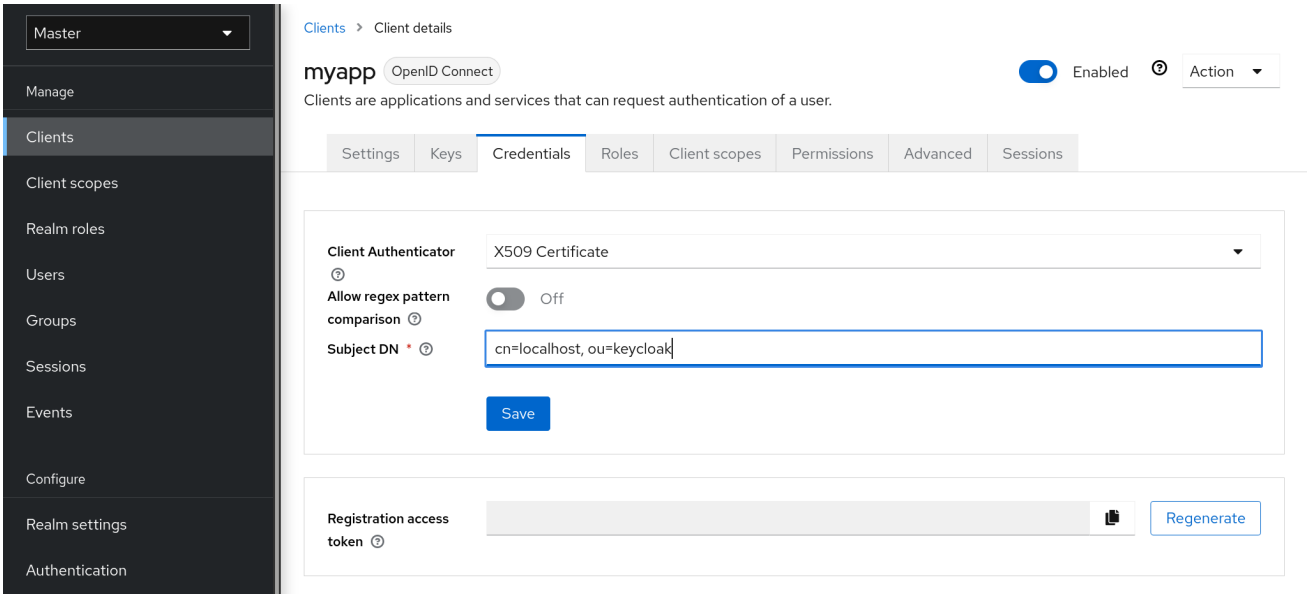
*如果选择了这个选项，您可以使用由客户端 **secret** 签名的 **JWT**，而不是私钥。*

*客户端机密将由客户端签名 **JWT**。*

***X509** 证书*

*红帽构建的 **Keycloak** 将验证客户端是否在 **TLS Handshake** 中使用正确的 **X509** 证书。*

***X509** 证书*




*验证器还使用配置的 **regexp** 验证表达式检查证书的 **Subject DN** 字段。对于某些用例，接受所有证书就足够了。在这种情况下，您可以使用 **(external?) (?:$)** 表达式。*

*红帽构建的 **Keycloak** 可以通过两种方式从请求获取客户端 **ID**：*

- 查询中的 **client_id** 参数（在 **OAuth 2.0** 规范的 **2.2** 节中规定）。
- 提供 **client_id** 作为表单参数。

#### 12.1.5. 客户端 Secret 轮转

重要

请注意，客户端 **Secret** 轮转支持正在开发中。以实验方式使用此功能。

对于具有 机密 客户端身份验证 红帽构建的 **Keycloak** 的客户端，支持通过客户端 策略 轮转客户端 **secret** 的功能。

客户端 **secret** 轮转策略提供了更高的安全性，以缓解问题，如 **secret** 泄漏。启用后，红帽构建的 **Keycloak** 支持每个客户端同时有两个活跃的 **secret**。策略根据以下设置管理轮转：

- **Secret expiration: [seconds] -** 当 **secret** 被轮转时，这是新 **secret** 的过期时间。添加到 **secret** 创建日期 中的数量（以秒为单位）。在策略执行时计算。
- 轮转 **secret expiration: [seconds] -** 当 **secret** 被轮转时，这个值是旧 **secret** 的剩余过期时间。这个值应该总是小于 **Secret** 过期。当值为 **0** 时，在客户端轮转过程中将立即删除旧 **secret**。添加到 **secret** 轮转日期 中的数量（以秒为单位）。在策略执行时计算。
- 更新期间轮转的剩余过期时间：**[秒] -** 更新动态客户端应执行客户端 **secret** 轮转时的时间。在策略执行时计算。

当发生客户端 **secret** 轮转时，会生成一个新的主 **secret**，旧的客户端主 **secret** 成为带有新的过期日期的二级 **secret**。

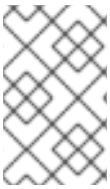
##### 12.1.5.1. 客户端 secret 轮转的规则

轮转不会自动发生，或通过后台进程进行。为了执行轮转，客户端上需要一个更新操作，通过 **Regenerate Secret**（在客户端凭证选项卡中或 **Admin REST API**）的功能，通过 **Red Hat build of Keycloak Admin Console**。在调用客户端更新操作时，**secret** 轮转会根据规则进行：

- 当 **Secret** 过期的 值小于当前日期时。

- 在动态客户端注册客户端更新请求时，如果更新 期间的 **Remaining** 过期时间 值与当前日期和 **Secret** 过期之间的周期匹配，客户端 **secret** 将自动轮转。

此外，管理员 **REST API** 可以随时强制进行客户端 **secret** 轮转。



注意

在创建新客户端时，如果客户端 **secret** 轮转策略处于活跃状态，则会自动应用行为。

警告

要将 **secret** 轮转行为应用到现有客户端，请在定义策略后更新该客户端，以便应用行为。

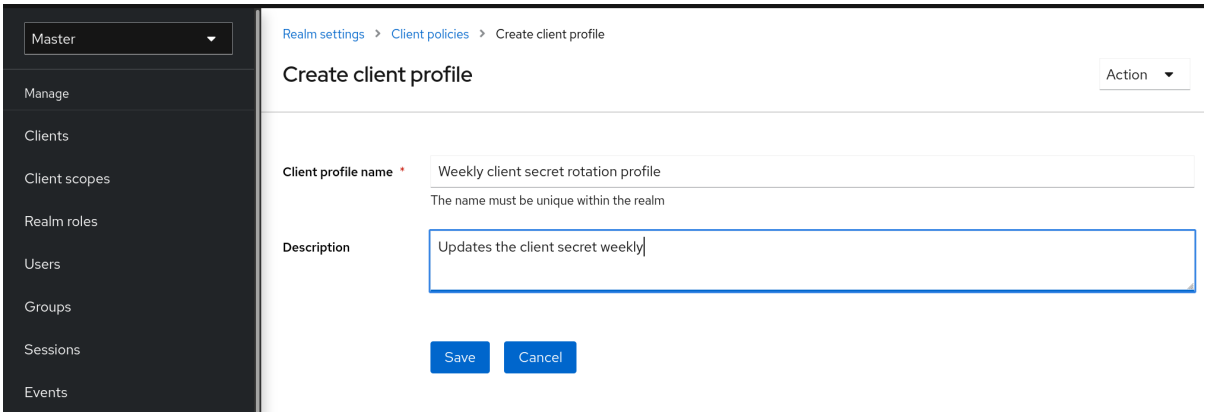
### 12.1.6. 创建 OIDC 客户端 Secret 轮转策略

以下是定义 **secret** 轮转策略的示例：

流程

1. 单击菜单中的 **Realm Settings**。
2. 点 **Client Policies** 选项卡。
3. 在 **Profiles** 页面上，单击 **Create client profile**。

创建配置集




4. *为 **Name** 输入任何名称。*

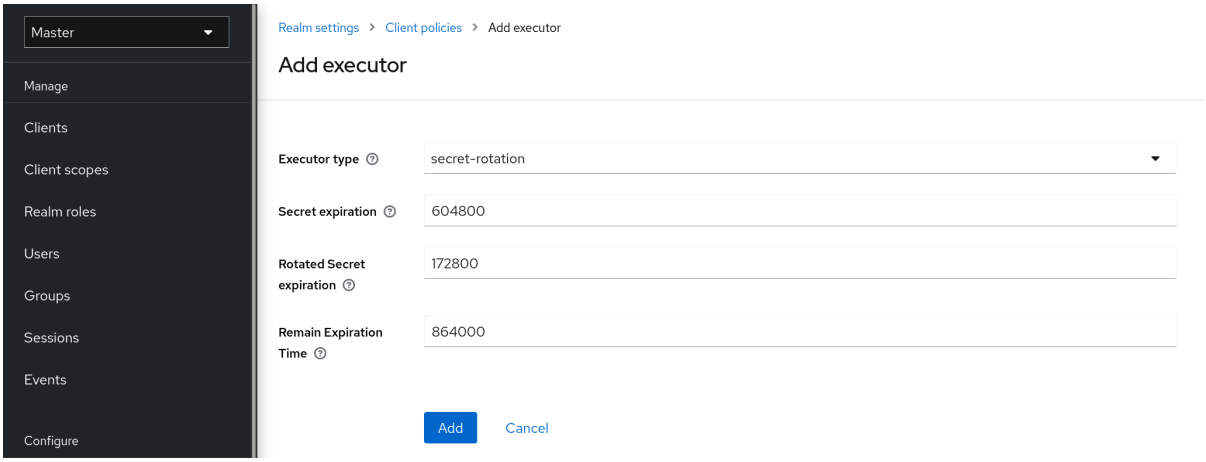
5. *输入描述帮助您识别 描述 的配置文件的用途。*

6. *点 **Save**。*

   *此操作会创建配置集，并可让您配置 **executors**。*

7. *点 **Add executor** 为这个配置集配置 **executor**。*

*创建配置集 **executor***




8. *选择 **Executor Type** 的 **secret-rotation**。*

9. 为 **Secret Expiration** 输入每个 **secret** 的最长持续时间时间（以秒为单位）。

10. 为 **Rotated Secret Expiration** 输入每个轮转 **secret** 的最长持续时间时间（以秒为单位）。

警告

请记住，**Rotated Secret Expiration** 值必须始终小于 **Secret Expiration**。

11. 输入时间（以秒为单位）后，任何更新操作都会为 **Remain Expiration Time** 更新客户端。

12. 点击 **Add**。

在上例中：

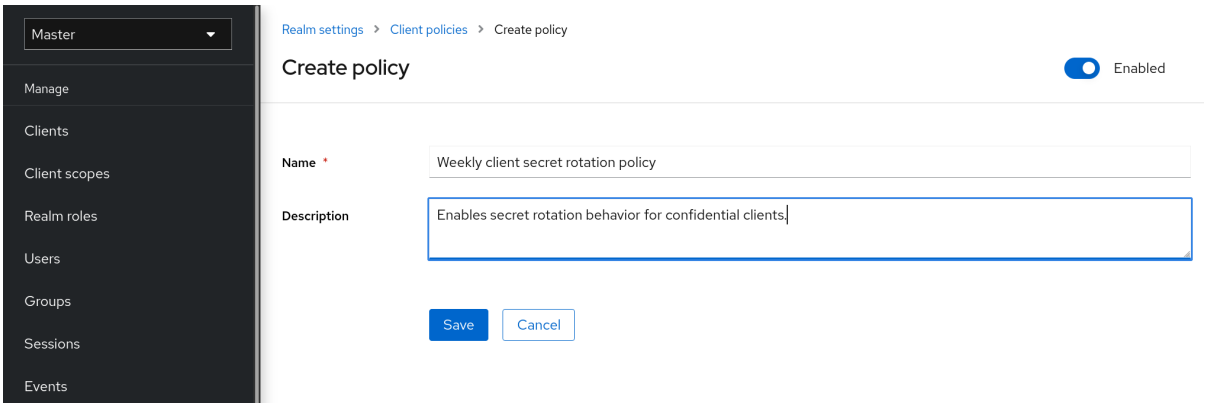
- 每个 **secret** 都在一周内有效。
- 轮转的 **secret** 在两天后过期。
- 更新动态客户端的窗口在 **secret** 过期前一天开始。

13. 返回到 **Client Policies** 选项卡。

14. 点 **Policies**。

15. 单击 **Create client policy**。

创建 **Client Secret Rotation** 策略




16. *为 **Name** 输入任何名称。*

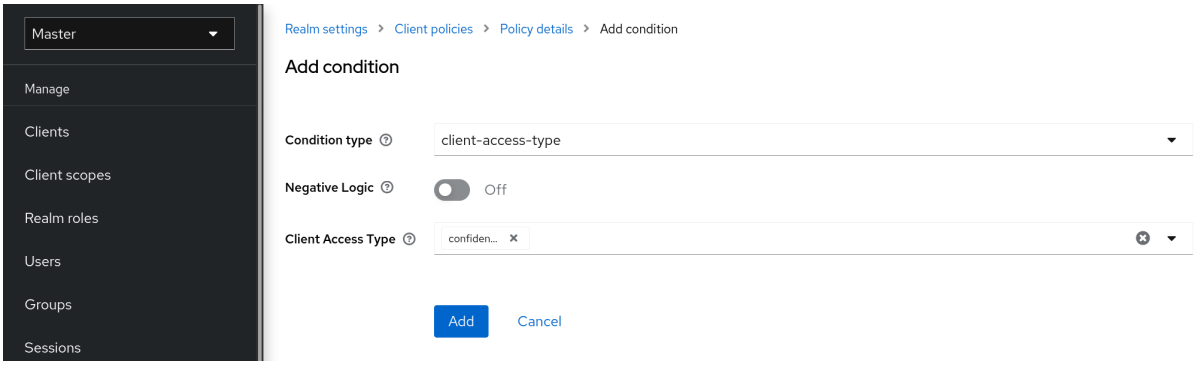
17. *输入描述，帮助您识别 描述 策略的用途。*

18. *点 **Save**。*

    *此操作会创建策略，并可让您将策略与配置集关联。它还允许您配置策略执行的条件。*

19. *在 **Conditions** 下，点 **Add condition**。*

***创建 Client Secret Rotation Policy Condition***




20. *要将行为应用到所有机密客户端，请在 **Condition Type** 字段中选择 **client-access-type***

*注意*

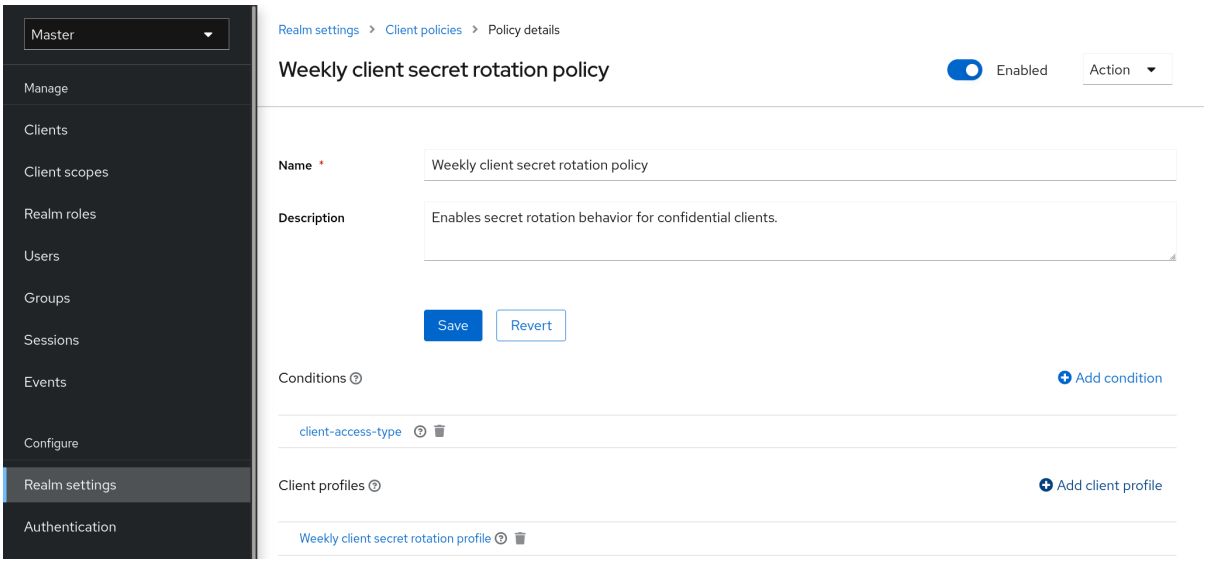
*要应用到特定的客户端组，另一种方法是选择* ***Condition Type*** *字段中的* ***client-roles*** *类型。这样，您可以创建特定的角色，并为每个角色分配自定义轮转配置。*

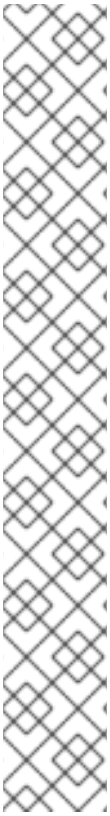
21. *将机密 添加到 字段* ***Client Access Type*** *中。*

22. *点击* ***Add***。

23. *返回到策略设置，在* ***Client Profiles*** *下，单击* ***Add client profile***，*然后从列表中选择* ***Weekly Client Secret Rotation Profile***，*然后点* ***Add***。

*客户端* ***Secret*** *轮转策略*





注意

要将 **secret** 轮转行为应用到现有客户端，请按照以下步骤执行：

使用管理控制台

1. 点菜单中的 **Clients**。
2. 点客户端。
3. 点 **Credentials** 选项卡。
4. 点客户端 **secret** 的 **Re-generate**。

使用客户端 **REST** 服务可以通过两种方式执行：

- 通过客户端上的更新操作
- 通过重新生成客户端 **secret** 端点

### 12.1.7. 使用服务帐户

每个 **OIDC** 客户端都有一个内置 服务帐户。使用 此服务帐户 获取访问令牌。

流程

1. 点菜单中的 **Clients**。
2. 选择您的客户端。

3. 点 **Settings** 选项卡。

4. 将 客户端身份验证 切换为 **On**。

5. 选择 服务帐户角色。

6. 点击 **Save**。

7. 配置客户端凭据。

8. 点 **Scope** 选项卡。

9. 验证您有角色，或者将 **Full Scope Allowed** 切换到 **ON**。

10. 点 **Service Account Roles** 选项卡

11. 为您的客户端配置此服务帐户可用的角色。

来自访问令牌的角色是以下交集：

- 客户端的角色范围映射与从链接的客户端范围继承的角色范围映射合并。
- 服务帐户角色.

要调用的 **REST URL** 是 **/realms/{realm-name}/protocol/openid-connect/token**。此 **URL** 必须作为 **POST** 请求调用，并要求您使用请求发布客户端凭证。

默认情况下，客户端凭证由 **Authorization: Basic** 标头中的客户端的 **clientId** 和 **clientSecret** 表示，但您也可以使用签名的 **JWT** 断言或客户端身份验证的任何其他自定义机制验证客户端。

您还需要根据 **OAuth2** 规范将 **grant_type** 参数设置为 **"client_credentials"**。

例如，检索服务帐户的 **POST** 调用类似如下：

```
POST /realms/demo/protocol/openid-connect/token
Authorization: Basic cHJvZHVjdC1zYS1jbGllbnQ6cGFzc3dvcmQ=
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

响应类似于来自 **OAuth 2.0** 规范的访问令牌 响应。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
    "access_token":"2YotnFZFEjr1zCsicMWpAA",
    "token_type":"bearer",
    "expires_in":60
}
```

默认仅返回访问令牌。不会返回刷新令牌，默认身份验证时不会在红帽构建的 **Keycloak** 端上创建用户会话。由于缺少刷新令牌，在访问令牌过期时需要重新进行身份验证。但是，这种情况并不意味着红帽构建的 **Keycloak** 服务器的额外开销，因为默认不会创建会话。

在这种情况下，不需要注销。但是，发布的访问令牌可以通过发送请求到 **OAuth2 Revocation Endpoint** 来撤销，如 **OpenID Connect Endpoints** 部分所述。

其他资源

如需了解更多详细信息，请参阅 客户端凭证授予。

#### 12.1.8. 受众支持

通常，部署了红帽构建的 **Keycloak** 的环境由一组使用红帽构建的 **Keycloak** 进行身份验证 的机密 或公共 客户端应用程序组成。

也可以使用 服务 ( **OAuth 2** 规范中的资源服务器 )为来自客户端应用程序的请求提供服务，并向这些

*应用提供资源。这些服务需要向它们发送访问令牌(**Bearer** 令牌)来验证请求。此令牌由 **frontend** 应用程序在登录到红帽 **Keycloak** 构建时获得。*

*在服务间信任较低的环境中，您可能会遇到这种情况：*

1. *前端客户端应用程序需要针对红帽构建的 **Keycloak** 进行身份验证。*
2. *红帽构建的 **Keycloak** 验证用户。*
3. *红帽构建的 **Keycloak** 向应用程序发布令牌。*
4. *应用使用令牌来调用不可信服务。*
5. *不受信任的服务将响应返回给应用程序。但是，它会保留应用令牌。*
6. *然后，不受信任的服务使用应用令牌调用可信服务。这会导致安全性问题，因为不受信任的服务可能会滥用令牌来代表客户端应用程序访问其他服务。*

*在服务间具有高度信任但信任低的环境中，这种情况不太可能。在某些环境中，此工作流可能正确，因为不受信任的服务可能需要从可信服务检索数据，才能将数据返回到原始客户端应用。*

*在服务间存在高级别信任时，有无限的受众很有用。否则，应限制受众。您可以限制受众，同时允许不受信任的服务从可信服务检索数据。在这种情况下，请确保不受信任的服务和可信服务被添加到令牌中。*

*为防止访问令牌滥用，请限制令牌的使用者并配置您的服务以验证令牌上的受众。流程将按如下方式更改：*

1. *前端应用程序针对红帽构建的 **Keycloak** 进行身份验证。*
2. *红帽构建的 **Keycloak** 验证用户。*

3. 红帽构建的 **Keycloak** 向应用程序发布令牌。应用程序知道它需要调用不受信任的服务，以便在发送到红帽构建的 **Keycloak** 的 身份验证请求中放置 **scope=<untrusted service >** （请参阅 **Client Scopes** 部分 以了解更多有关 **scope** 参数的详细信息）。

   向应用程序发布的令牌包含对其受众中的不受信任的服务的引用（**"audience": [ "<untrusted service>" ]**，它声明客户端使用此访问令牌来调用不受信任的服务。

4. 不受信任的服务调用带有令牌的可信服务。调用不成功，因为可信服务会检查令牌的受众，并发现其受众只针对不受信任的服务。这个行为是正常的，安全性不会被破坏。

如果客户端希望稍后调用可信服务，它必须通过使用 **scope =<trusted service>** 颁发 **SSO** 登录来获取另一个令牌。然后，返回的令牌将包含可信服务作为受众：

> **"audience": [ "<trusted service>" ]**

使用这个值调用 **< trusted service>**。

#### 12.1.8.1. 设置

设置受众检查时：

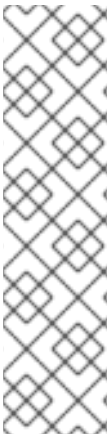
- 通过在 **adapter** 配置中添加标志 **verify-token-audience**，确保服务被配置为检查发送到它们的访问令牌的受众。详情请参阅适配器配置。
- 确保红帽构建的 **Keycloak** 发布的访问令牌包含所有必要的受众。可使用客户端角色添加受众，如 下一节 或硬编码所述。请参阅 硬编码的受众。

#### 12.1.8.2. 自动添加受众

在默认的客户端 范围角色 中定义 **Audience Resolve** 协议映射器。映射器会检查至少有一个客户端角色可用于当前令牌的客户端。然后，每个客户端的客户端 **ID** 会添加为受众，如果您的服务客户端依赖客户端角色时很有用。服务客户端通常是在没有启用任何流的情况下的客户端，这可能没有直接向其自身发布的令牌。它代表 **OAuth 2** 资源服务器。

例如，对于服务客户端和机密客户端，您可以使用为机密客户端发布的访问令牌来调用服务客户端 **REST** 服务。如果满足以下条件，服务客户端将自动作为受众添加到为机密客户端发布的访问令牌中：

- 服务客户端本身上定义了任何客户端角色。
- 目标用户至少分配了这些客户端角色之一。
- 机密客户端具有所分配的角色的角色范围映射。



注意

如果要确保不自动添加 **audience**，请不要直接在机密客户端上配置角色范围映射。相反，您可以创建一个专用的客户端范围，其中包含专用客户端范围的客户端角色范围映射。

假设将客户端范围作为可选客户端范围添加到机密客户端，如果 **scope=< trusted service >** 参数明确请求，客户端角色和 **audience** 将添加到令牌中。

注意

前端客户端本身不会自动添加到访问令牌受众中，因此允许在访问令牌和 **ID** 令牌之间轻松区分，因为访问令牌不包含作为受众发出令牌的客户端。

如果您需要客户端本身作为受众，请查看 *硬编码的 **audience*** 选项。但是，不建议使用与 **frontend** 和 **REST** 服务相同的客户端。

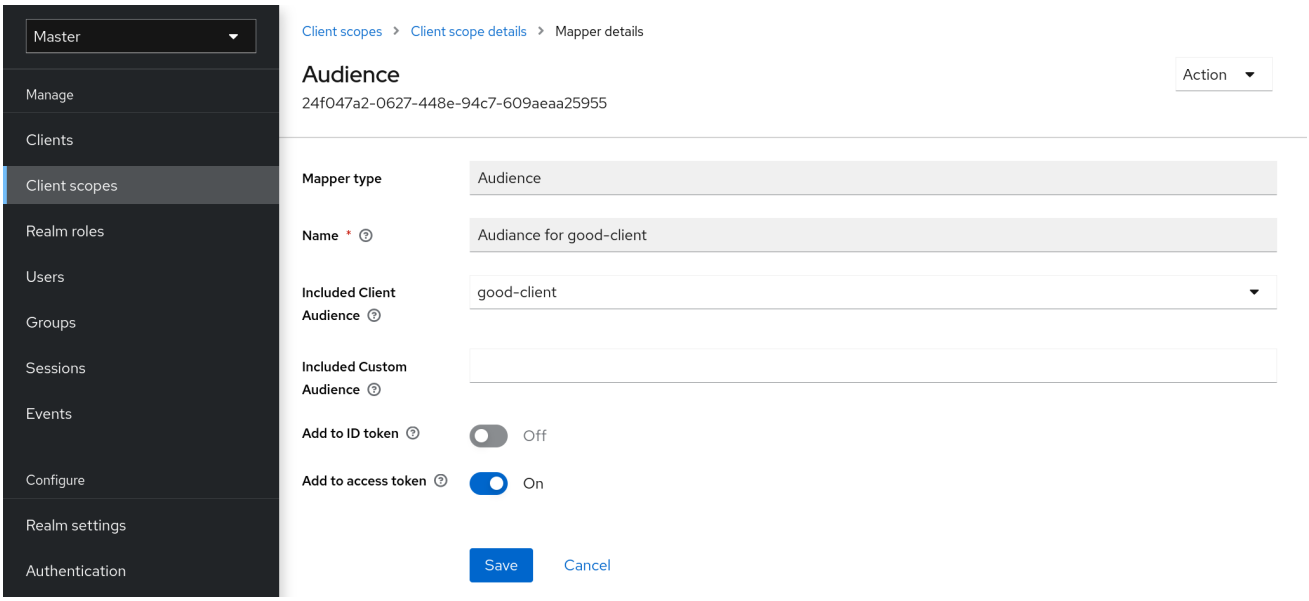
#### *12.1.8.3. 硬编码的受众*

当您的服务依赖于域角色或根本不依赖于令牌中的角色时，使用硬编码的受众会很有用。硬编码的受众是一个协议映射程序，它会将指定服务客户端的客户端 **ID** 添加为令牌的受众。如果要使用与客户端 **ID** 不同的受众，您可以使用任何自定义值，如 **URL**。

您可以将协议映射程序直接添加到前端客户端。如果直接添加协议映射程序，将始终添加受众。

要对协议映射程序进行更多控制，您可以在专用客户端范围内创建协议映射器，该范围将称为 **good-service**。

受众协议映射器




- *在 **good-service** 客户端的客户端 详情选项卡中，您可以生成适配器配置，并确认 **verify-token-audience** 已设置为 **true**。如果您使用此配置，这个操作会强制适配器验证受众。*

- *您需要确保机密客户端能够向其令牌中作为受众 提供良好的服务。*

  *在机密客户端中：*

  1. *点 **Client Scopes** 选项卡。*
  2. *将 **good-service** 分配为可选（或默认）客户端范围。*

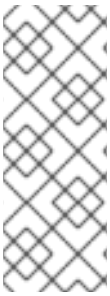
     *如需了解更多详细信息，请参阅客户端范围链接部分。*

- *您可以选择 评估客户端范围 并生成示例访问令牌。如果您将 **good-service** 指定为可选客户端范围时，如果您将 **good-service** 包含在 **scope** 参数中，则将添加到生成的访问令牌的受众中。*

- *在机密客户端应用中，确保使用了 **scope** 参数。当您要签发用于访问 **good-service** 的令牌时，必须包含值 **good-service**。*

*请参阅：*

- *如果您的应用程序使用 **servlet** 适配器，则需要转发部分。*
- *如果您的应用程序使用 **javascript** 适配器，则 **JavaScript adapter** 部分。*



*注意*

*默认情况下，**Audience** 和 **Audience Resolve** 协议映射程序都仅将受众添加到访问令牌中。**ID Token** 通常仅包含一个受众，即签发令牌的客户端 **ID**，这是 **OpenID Connect** 规格的要求。但是，访问令牌不一定具有客户端 **ID**，这是签发的令牌，除非使用者映射程序添加了它。*

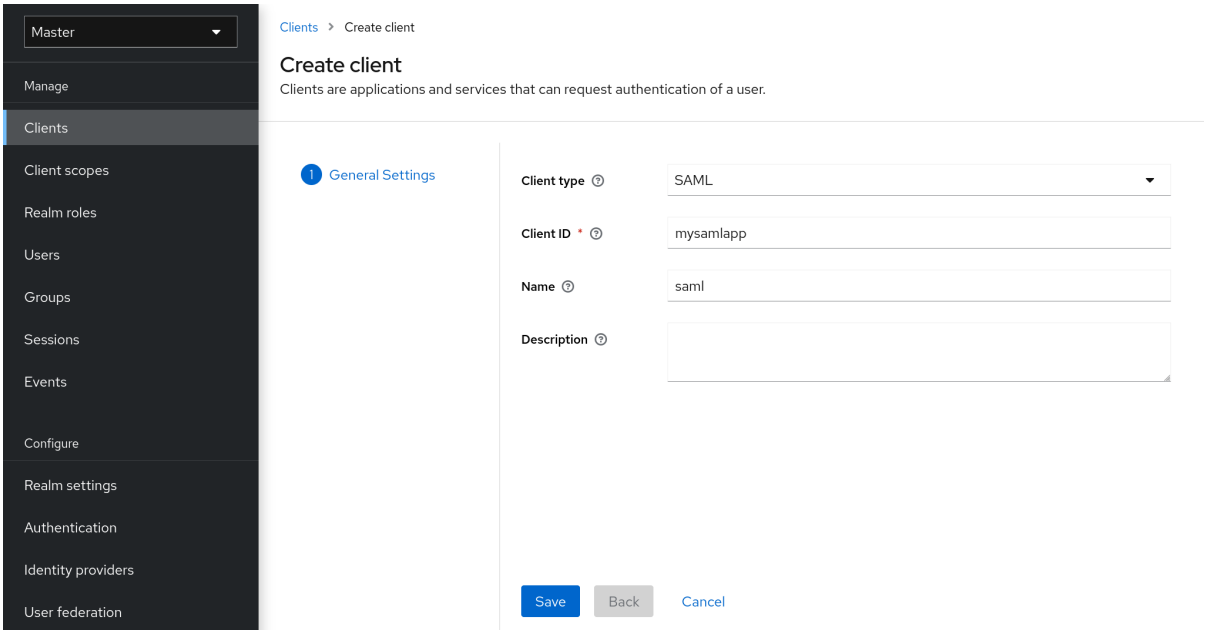
### *12.2. 创建 SAML 客户端*

*红帽构建的 **Keycloak** 支持 **SAML 2.0** 用于注册的应用程序。支持 **POST** 和重定向绑定。您可以选择需要客户端签名验证。您还可以让服务器签名和/或加密响应。*

*流程*

1. *点菜单中的 **Clients**。*
2. *单击 **Create client** 以进入 **Create client** 页面。*
3. *将 **Client type** 设置为 **SAML**。*

*创建客户端*




4. *输入客户端的客户端 **ID**。这通常是一个 **URL**，它是应用程序发送的 **SAML** 请求中预期的 签发者 值。*

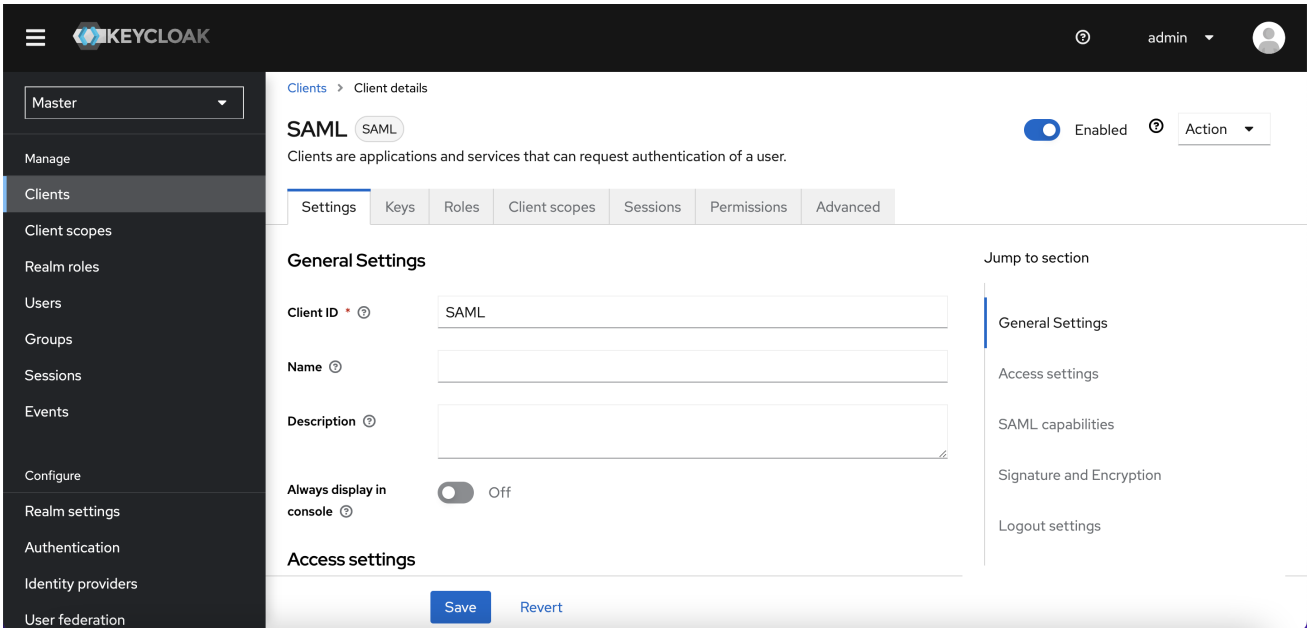
5. *点击 **Save**。此操作会创建客户端，并将您带到 **Settings** 选项卡。*

*以下小节描述了此选项卡上的每个设置。*

### *12.2.1. 设置标签页*

***Settings** 选项卡包含许多配置此客户端的选项。*

*客户端设置*




#### *12.2.1.1. 常规设置*

*客户端 ID*

> ***OIDC** 请求以及 **Red Hat build of Keycloak** 数据库用于识别客户端的字母数字字符 **ID** 字符串。这个值必须与使用 **AuthNRequests** 发送的签发者值匹配。**Red Hat build of Keycloak** 从 **Authn SAML** 请求中提取签发者，并根据这个值将其与客户端匹配。*

***Name***

> *红帽构建的 **Keycloak UI** 屏幕中的客户端名称。要本地化名称，请设置替换字符串值。例如，字符串值，如 **${myapp}**。如需更多信息，请参阅 服务器开发人员指南。*

*描述*

> *客户端的描述。此设置也可以是本地化。*

*始终在控制台中显示*

> *在帐户控制台中始终列出此客户端，即使此用户没有活跃的会话。*

#### *12.2.1.2. 访问设置*

*根 URL*

> *当红帽构建的 **Keycloak** 使用配置的相对 **URL** 时，这个值会添加到 **URL** 中。*

*主页 URL*

如果红帽构建的 **Keycloak** 需要链接到客户端，则会使用这个 **URL**。

有效的 **Redirect URI**

输入 **URL** 模式，然后单击要添加的 **+** 符号。点 **-** 符号删除。点 **Save** 保存这些更改。通配符值仅在 **URL** 的末尾允许。例如： **http://host.com/*$$**。当没有注册准确的 **SAML** 端点，而 **Red Hat build of Keycloak** 会从请求拉取 **Assertion Consumer URL** 时，会使用此字段。

**IDP-Initiated SSO URL** 名称

如果要执行 **IDP** 启动的 **SSO** 时引用客户端的 **URL** 片段名称。保留此空将禁用 **IDP Initiated SSO**。您将从浏览器引用的 **URL** 为： **server-root/realms/{realm}/protocol/saml/clients/{client-url-name}**

**IDP Initiated SSO Relay State**

当您想要执行 **IDP Initiated SSO** 时，您要使用 **SAML** 请求发送的中继状态。

**Master SAML** 处理 **URL**

此 **URL** 用于所有 **SAML** 请求，响应被定向到 **SP**。它被用作 **Assertion Consumer Service URL** 和 **Single Logout Service URL**。

如果登录请求包含 **Assertion Consumer Service URL**，则这些登录请求将具有优先权。此 **URL** 必须通过注册的 **Valid Redirect URI** 模式进行验证。

#### 12.2.1.3. SAML 功能

名称 **ID** 格式

主题的名称 **ID** 格式。如果请求中没有指定名称 **ID** 策略，或者使用此格式，或者 **Force Name ID Format** 属性设为 **ON**，则使用此格式。

强制名称 **ID** 格式

如果请求具有名称 **ID** 策略，请忽略它，并使用 **Admin Console** 下 **Name ID Format** 下配置的值。

强制 **POST** 绑定

默认情况下，红帽构建的 **Keycloak** 使用原始请求的初始 **SAML** 绑定进行响应。通过启用 强制 **POST Binding**，红帽构建的 **Keycloak** 会使用 **SAML POST** 绑定来响应，即使原始请求使用了重定向绑定。

强制工件绑定

如果启用，响应消息将通过 **SAML ARTIFACT** 绑定系统返回到客户端。

包括 **AuthnStatement**

**SAML** 登录响应可以指定所用的身份验证方法，如密码，以及登录和会话到期的时间戳。包括 **AuthnStatement** 会被默认启用，以便 **AuthnStatement** 元素将包含在登录响应中。把它设置为 **OFF** 可防止客户端确定最大会话长度，这可以创建不过期的客户端会话。

**include OneTimeUse Condition**

如果启用，则登录响应中包含 **OneTimeUse Condition**。

优化 **REDIRECT** 签名密钥查找

当设置为 **ON** 时，**SAML** 协议消息包括红帽构建的 **Keycloak** 原生扩展。此扩展包含签名密钥 **ID** 的提示。**SP** 使用扩展进行签名验证，而不是尝试使用密钥验证签名。

这个选项适用于在查询参数中传输签名的 **REDIRECT** 绑定，且这些信息不会在签名信息中找到。这与 **POST** 绑定消息（其中密钥 **ID** 始终包含在文档签名中）。

当红帽构建 **Keycloak** 服务器和适配器提供 **IDP** 和 **SP** 时，会使用这个选项。这个选项只有在 **Sign Documents** 设置为 **ON** 时才相关。

#### 12.2.1.4. 签名和加密

签署文档

当设置为 **ON** 时，红帽构建的 **Keycloak** 会使用域私钥为文档签名。

**sign Assertions**

断言已签名并嵌入在 **SAML XML Auth** 响应中。

签名算法

签名 **SAML** 文档中使用的算法。请注意，基于 **SHA1** 的算法已弃用，并可能在以后的发行版本中删除。我们建议使用一些更安全的算法，而不是 ***_SHA1**。另外，如果 **SAML** 客户端在 **Java 17** 或更高版本上运行，验证签名无法正常工作。

**SAML** 签名密钥名称

使用 **POST** 绑定发送的签名 **SAML** 文档包含 **KeyName** 元素中签名密钥的标识。此操作可由 **SAML** 签名密钥名称 选项控制。此选项控制 **Keyname** 的内容。

- *KEY_ID KeyName 包含密钥 ID。这个选项是默认选项。*

- *CERT_SUBJECT KeyName 包含与 realm 键对应的证书的主题。Microsoft Active Directory Federation Services 期望这个选项。*

- *NONE 在 SAML 消息中完全省略 KeyName hint。*

*规范方法*

*XML 签名的规范方法。*

#### *12.2.1.5. 登录设置*

*登录主题*

*用于登录、OTP、授予注册和忘记密码页面的主题。*

*需要同意*

*如果启用，用户必须同意客户端访问。*

*对于执行浏览器登录的客户端。因为无法确保 secret 可以与客户端内部客户端保持安全，因此务必要通过配置正确的重定向 URI 来限制访问。*

*在屏幕上显示客户端*

*如果 Consent Required is Off，则应用此开关。*

- *关*

  *consent 屏幕将仅包含与配置的客户端范围对应的同意。*

- *On*

  *在同意屏幕上也会有一个有关此客户端本身的项目。*

*客户端同意屏幕文本*

如果启用了 **Consent required** 和 **Display client on screen**，则适用。包含此客户端有关权限的同意屏幕上的文本。

#### 12.2.1.6. 注销设置

前端频道注销

如果启用了 **Front Channel Logout**，应用程序需要重定向浏览器才能执行注销。例如，应用程序可能需要重置 **Cookie**，该 **Cookie** 只能通过重定向来完成。如果禁用了 **Front Channel Logout**，红帽构建的 **Keycloak** 会调用后台 **SAML** 请求来注销应用程序。

### 12.2.2. keys 标签页

加密断言

使用 **realm** 私钥加密 **SAML** 文档中的断言。**AES** 算法的密钥大小为 **128** 位。

需要客户端签名

如果启用了 **Client Signature Required**，则来自客户端的文档会被签名。红帽构建的 **Keycloak** 将使用在 **Keys** 选项卡中设置的客户端公钥或证书来验证此签名。

允许 **ECP** 流

如果为 **true**，则允许此应用程序使用 **SAML ECP** 配置集进行身份验证。

### 12.2.3. 高级标签页

这个标签对于特定情况有很多字段。其他主题中将涵盖一些字段。有关其他字段的详情，点问号图标。

#### 12.2.3.1. 精细 Grain SAML 端点配置

徽标 **URL**

引用客户端应用程序徽标的 **URL**。

策略 **URL**

**Relying party Client** 提供的 **URL** 提供给 **End-User**，以阅读有关如何使用配置集数据的 **URL**。

服务 **URL** 条款

过期客户端向最终用户提供 **URL**，以阅读有关重新处理服务条款的 **URL**。

*Assertion Consumer Service POST Binding URL*

*Assertion Consumer Service 的 POST Binding URL。*

*Assertion Consumer Service Redirect Binding URL*

*重定向 Assertion Consumer Service 的绑定 URL。*

*注销服务 POST Binding URL*

*Logout 服务的 POST Binding URL。*

*Logout Service Redirect Binding URL*

*重定向 Logout 服务的绑定 URL。*

*注销 Service Artifact Binding URL*

*Logout Service 的工件绑定 URL。当与 Force Artifact Binding 选项一起设置时，Artifact 绑定会被强制用于登录和注销流程。除非设置了此属性，否则工件 绑定不会用于注销。*

*注销 Service SOAP Binding URL*

*重定向 Logout 服务的绑定 URL。仅在使用 back channel logout 时才适用。*

*工件绑定 URL*

*将 HTTP 工件消息发送到的 URL。*

*工件解析服务*

*将 ArtifactResolve 消息发送到的客户端 SOAP 端点的 URL。*

#### *12.2.4. IDP 启动登录*

*IDP Initiated Login 是一项功能，允许您在红帽构建的 Keycloak 服务器上设置一个端点，供您登录到特定的应用程序/客户端。在客户端的 Settings 选项卡中，您需要指定 IDP Initiated SSO URL Name。这是一个简单的字符串，其中没有空格。之后，您可以在以下 URL 中引用您的客户端：*
***root/realms/{realm}/protocol/saml/clients/{url-name}***

*IDP 启动登录实施首选 POST over REDIRECT 绑定（查看 saml 绑定 以获取更多信息）。因此，最终绑定和 SP URL 被选择如下：*

1. *如果定义了特定的 Assertion Consumer Service POST Binding URL （客户端设置 中的*

*SAML Endpoint Configuration 部分） POST 绑定通过该 URL 使用。*

2. *如果指定了常规 Master SAML Processing URL，则在此常规 URL 中再次使用 POST 绑定。*

3. *最后，如果配置了 Assertion Consumer Service Redirect Binding URL （在 Fine Grain SAML Endpoint Configuration配置 中） REDIRECT 绑定用于这个 URL。*

*如果您的客户端需要特殊的中继状态，也可以在 IDP Initiated SSO Relay State 字段中的 Settings 选项卡中进行配置。或者，浏览器可以在 RelayState 查询参数中指定中继状态，即 root/realms/{realm}/protocol/saml/clients/{url-name}?RelayState=thestate。*

*使用身份代理 时，可以从外部 IDP 为客户端设置 IDP 启动登录。实际客户端为代理 IDP 的 IDP 启动登录设置，如上所述。外部 IDP 必须为应用程序 IDP 启动登录设置客户端，指向指向代理的特殊 URL，并在代理 IDP 上为所选客户端代表 IDP Initiated Login 端点。这意味着，在外部 IDP 的客户端设置中：*

- *IDP Initiated SSO URL Name 设置为一个名称，该名称将作为 IDP Initiated Login 初始点发布，*

- *Fine Grain SAML Endpoint Configuration 部分中的 断言 Consumer Service POST Binding URL 必须设置为以下 URL： broker-root/realms/{broker-realm}/broker/{idp-name}/endpoint/clients/{client-id}，其中：*

  - *broker-root 是基本代理 URL*

  - *broker-realm 是声明外部 IDP 的代理中的域名称*

  - *IdP-name 是代理的外部 IDP 的名称*

  - *client-id 是代理上定义的 SAML 客户端的 IDP Initiated SSO URL Name 属性的值。这是此客户端，可供来自外部 IDP 的 IDP 启动登录提供。*

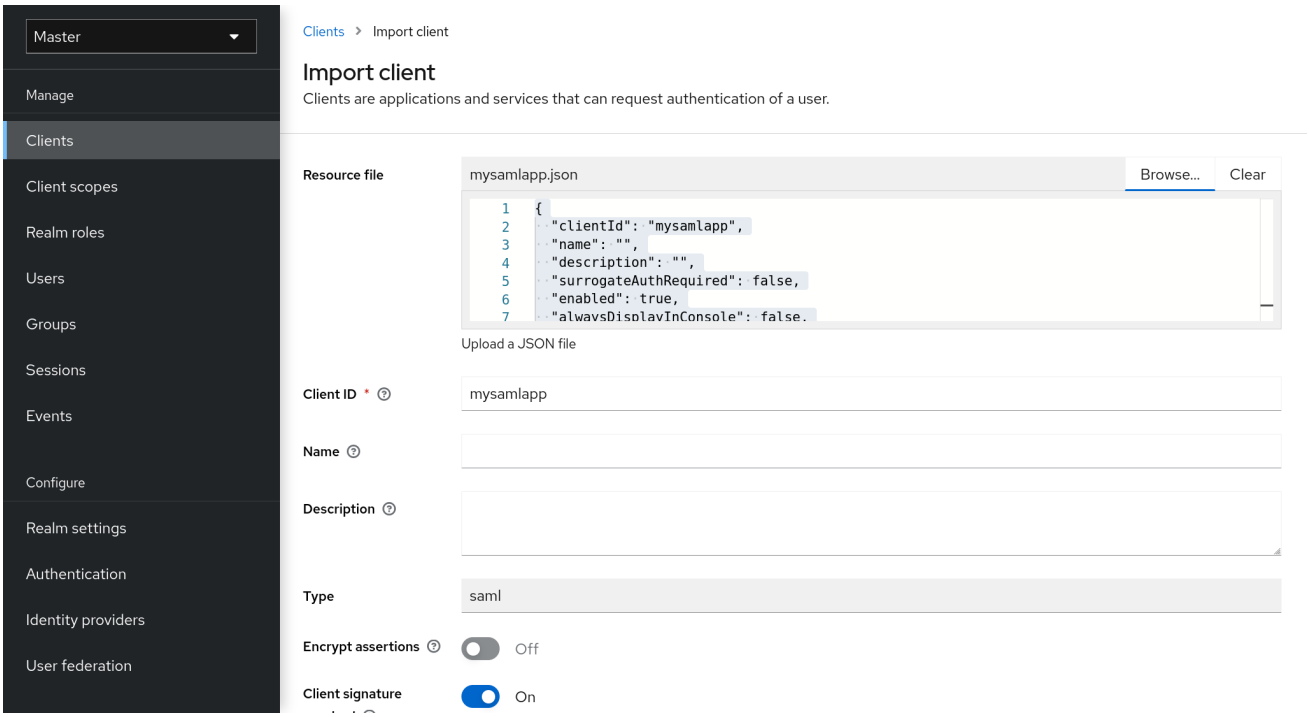
*请注意，您可以将来自代理 IDP 的基本客户端设置导入到外部 IDP 的客户端设置 - 仅使用代理 IDP 中的身份供应商的 SP Descriptor，并将 客户端/客户端-id 添加到端点 URL。*

### 12.2.5. 使用实体描述符创建客户端

您可以使用标准的 **SAML Entity Descriptor XML** 文件导入客户端，而不是手动注册 **SAML 2.0** 客户端。

**Client** 页面包含一个 **Import client** 选项。

添加客户端




流程

1. 点 **Browse**。
2. 加载包含 **XML** 实体描述符信息的文件。
3. 检查信息以确保一切设置正确。

有些 **SAML** 客户端适配器（如 **mod-auth-mellon** ）需要 **IDP** 的 **XML Entity Descriptor**。您可以通过进入这个 **URL** 来查找此描述符：

-

```
root/realms/{realm}/protocol/saml/descriptor
```

其中 **realm** 是您的客户端的域。

### 12.3. 客户端链接

为了从一个客户端链接到另一个客户端，红帽构建的 **Keycloak** 提供了一个重定向端点：**/realms/realm_name/clients/{client-id}/redirect**。

如果客户端使用 **HTTP GET** 请求访问此端点，红帽构建的 **Keycloak** 会在响应的 **Location** 标头中以 **HTTP 307 (Temporary Redirect)**的形式返回提供的 **Client** 和 **Realm** 的配置基本 **URL**。因此，客户端只需要知道 **Realm** 名称和客户端 **ID** 来链接到它们。这种间接性避免硬编码客户端基础 **URL**。

例如，给定 **realm master** 和 **client-id** 帐户 ：

```
http://host:port/realms/master/clients/account/redirect
```

这个 **URL** 临时重定向到 ：**http://host:port/realms/master/account**

### 12.4. OIDC 令牌和 SAML 断言映射

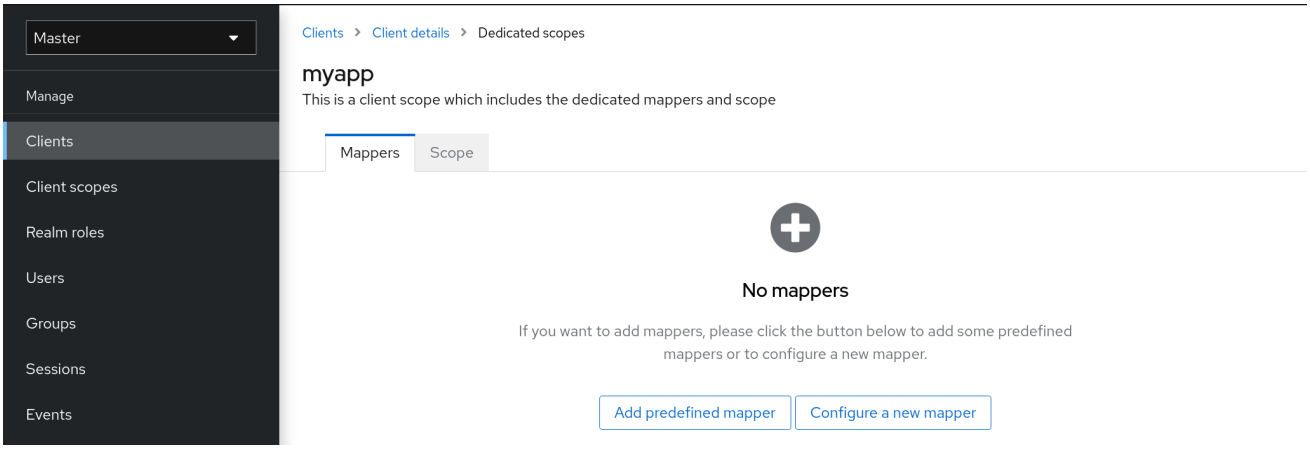
接收 **ID** 令牌、访问令牌或 **SAML** 断言的应用程序可能需要不同的角色和用户元数据。

您可以使用红帽构建的 **Keycloak** 进行以下操作：

- 硬编码角色、声明和自定义属性。
- 将用户元数据拉取到令牌或断言中。
- 重命名角色。

您可以在管理控制台的 **Mappers** 选项卡中执行这些操作。

***Mappers** 标签页*

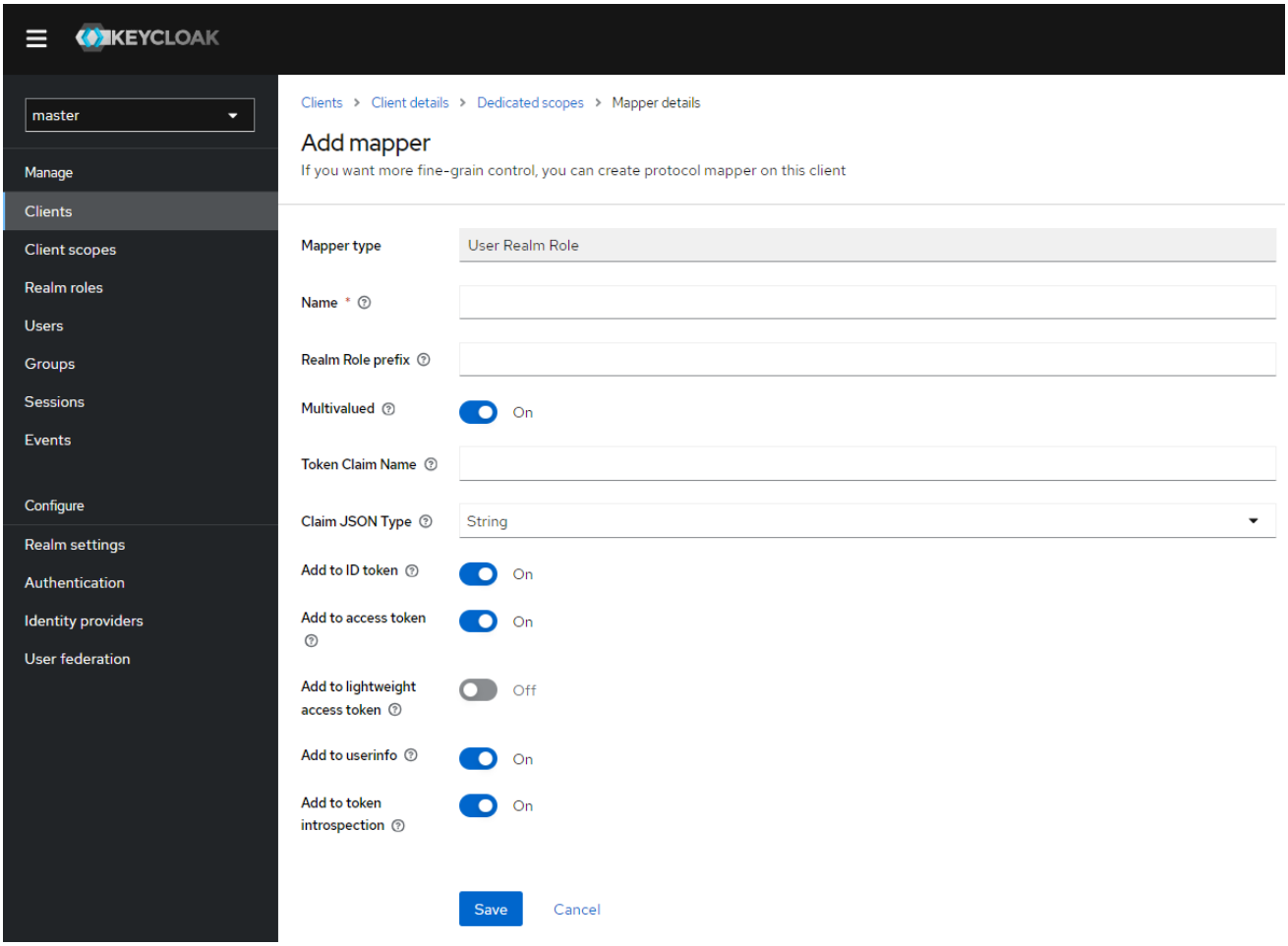



*新客户端没有内置映射程序，但可以从客户端范围继承一些映射程序。如需了解更多详细信息，请参阅客户端范围部分。*

*协议映射器将项目（如电子邮件地址）映射到身份和访问令牌中的特定声明。映射器的功能应该从其名称中自我解释。您可以通过单击 **Add Builtin** 来添加预先配置的映射程序。*

*每个映射程序都有一组通用设置。根据映射程序类型，提供了其他设置。点 **mapper** 旁边的 **Edit** 来访问配置屏幕来调整这些设置。*

*映射器配置*




*将鼠标悬停在工具提示上，可以查看每个选项的详情。*

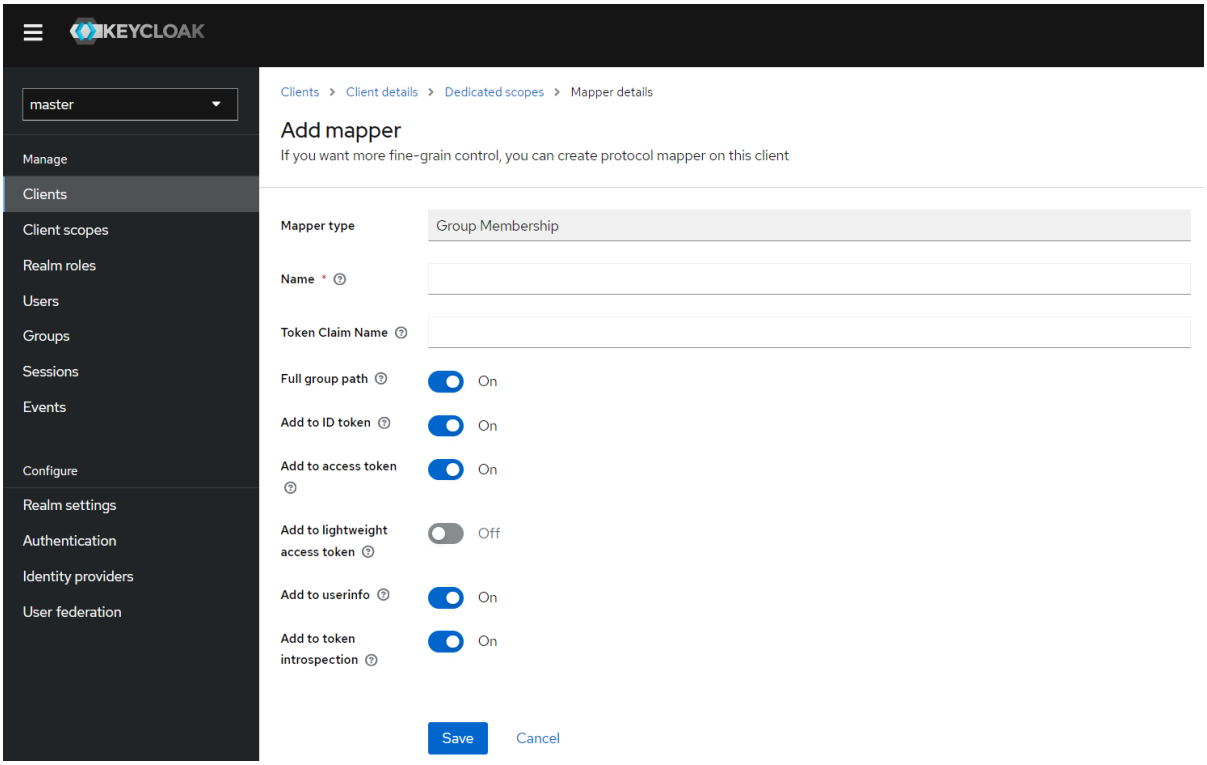
*您可以使用大多数 **OIDC** 映射器来控制声明的放置位置。您可以通过调整 **Add to ID** 令牌和 **Add to access token**，从 **id** 和 **access tokens** 中包括或排除声明。*

*您可以按照以下方法添加映射程序类型：*

*流程*

1. *转至 **Mappers** 选项卡。*
2. *单击 **Configure a new mapper**。*

*添加映射器*




3. *从列表框中选择一个 映射程序类型。*

### *12.4.1. 优先级顺序*

*映射程序实施具有 优先级顺序。优先级顺序 不是映射程序的配置属性。它是映射程序的 **concrete** 实现的属性。*

***Mappers** 按照映射程序列表中的顺序进行排序。令牌或断言中的更改会按照首先应用最低的顺序应用。因此，依赖于其他实现的实现会按照必要顺序进行处理。*

*例如，要计算令牌中包含的角色：*

1. *基于这些角色解决受众。*
2. *处理使用令牌中已提供的角色和受众的 **JavaScript** 脚本。*

### *12.4.2. OIDC 用户会话备注映射程序*

用户会话详情使用映射程序定义，并在客户端上启用功能时自动包含。点 **Add builtin**，使其包含会话详情。

模拟的用户会话提供以下详情：

- ***IMPERSONATOR_ID*** ：模拟用户的 **ID**。
- ***IMPERSONATOR_USERNAME*** ：模拟用户的用户名。

服务帐户会话提供以下详情：

- ***clientId*** ：服务帐户的客户端 **ID**。
- ***client_id*** ：服务帐户的客户端 **ID**。
- ***clientAddress*** ：服务帐户验证设备的远程主机 **IP**。
- ***clientHost*** ：服务帐户验证的设备的远程主机名。

### *12.4.3. 脚本映射器*

通过运行用户定义的 **JavaScript** 代码，使用 **Script Mapper** 将声明映射到令牌。有关向服务器部署脚本的更多详细信息，请参阅 **JavaScript Providers**。

当脚本部署时，您可以从可用映射器列表中选择部署的脚本。

### *12.4.4. 使用轻量级访问令牌*

红帽构建的 **Keycloak** 中的访问令牌包含敏感信息，包括个人 **Identifiable** 信息**(PII)**。因此，如果资源服务器不希望将此类信息披露给客户端等第三方实体，红帽构建的 **Keycloak** 支持从访问令牌中删除 **PII** 的轻量级访问令牌。另外，当资源服务器从访问令牌中删除了 **PII** 时，它可以通过向 **Keycloak** 的令牌内省端点的红帽构建发送访问令牌来获取 **PII**。

*无法从轻量级访问令牌中删除的信息*

*协议映射器可以控制哪些信息被放入访问令牌中，而轻量级访问令牌则使用协议映射器。因此，不能从轻量级访问中删除以下信息。*
***exp,iat,auth_time,jti,iss,sub,typ,azp,nonce,session_state,sid,scope cnf***

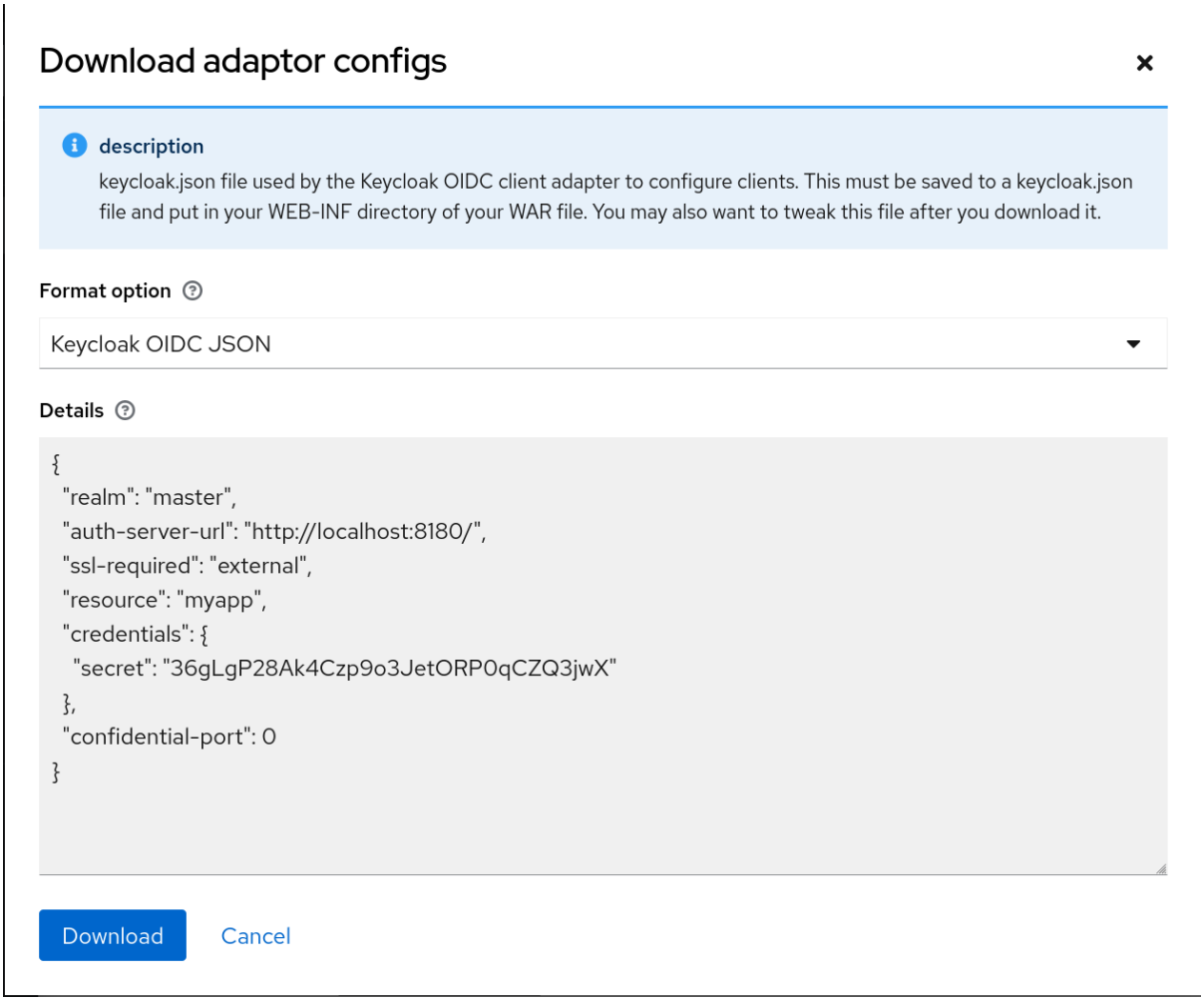
*在红帽构建的 **Keycloak** 中使用轻量级访问令牌*

*通过将 **use-lightweight-access-token executor** 应用到客户端，客户端可以接收轻量级访问令牌，而不是访问令牌。**???**轻量级访问令牌包含一个由协议映射器控制的声明，其设置 设为轻量级访问令牌（默认为 **OFF**）被打开。另外，通过将其设置 **Add** 切换到 协议映射器的令牌内省，客户端可以通过将访问令牌发送到红帽构建的 **Keycloak** 令牌内省端点来获取声明。*

### *12.5. 生成客户端适配器配置*

*红帽构建的 **Keycloak** 可以生成可在应用程序部署环境中安装客户端适配器的配置文件。**OIDC** 和 **SAML** 支持很多适配器类型。*

1. *点 **Action** 菜单并选择 **Download adapter** 配置选项*



2. *选择您要为其生成的格式 选项。*

*支持 **OIDC** 和 **SAML** 的所有红帽构建 **Keycloak** 客户端适配器。**SAML** 的 **mod-auth-mellon Apache HTTPD** 适配器以及标准 **SAML** 实体描述符文件被支持。*

## *12.6. 客户端范围*

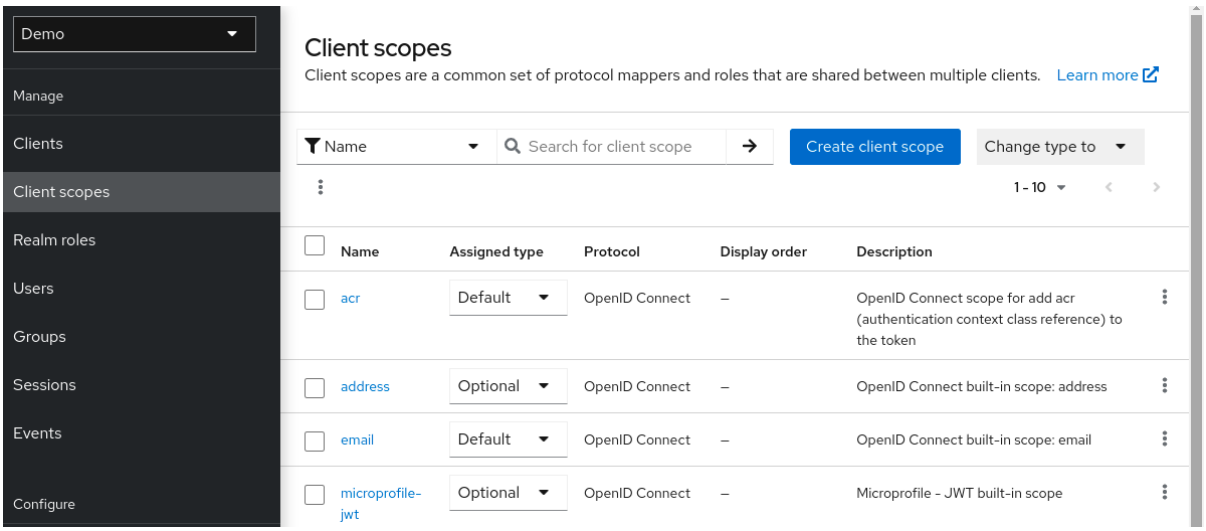
*使用红帽构建的 **Keycloak** 在名为 客户端范围 的实体中定义共享客户端配置。客户端范围为 多个客户端配置 协议映射器 和角色范围映射。*

*客户端范围也支持 **OAuth 2** 范围 参数。客户端应用程序使用此参数请求访问令牌中的声明或角色，具体取决于应用程序的要求。*

*要创建客户端范围，请按照以下步骤执行：*

1. *点菜单中的 **Client Scopes**。*

*客户端范围列表*



2. 点 **Create**。

3. 为您的客户端范围命名。

4. 点 **Save**。

客户端范围 与常规客户端有类似的标签页。您可以定义 协议映射器 和角色范围映射。这些映射可由其他客户端继承，并配置为继承此客户端范围。

### 12.6.1. 协议

在创建客户端范围时，选择 协议。在同一范围内链接的客户端必须具有相同的协议。

每个 **realm** 在菜单中都有一组预定义的内置客户端范围。

- **SAML** 协议： **role_list.**此范围包含 **SAML** 断言中角色列表的一个协议映射程序。
- **OpenID Connect** 协议：提供 **Several** 客户端范围：
  - **roles**

    此范围在 **OpenID Connect** 规格中未定义，不会自动添加到访问令牌中的 范围 声明中。此范围具有映射程序，用于将用户的角色添加到访问令牌中，并为至少有一个客户端角色的客户端添加受众。在 **Audience** 部分 更详细地描述了这些映射程序。
  - **web-origins**

    此范围也没有在 **OpenID Connect** 规格中定义，且不添加到声明访问令牌的 范围中。此范围用于将允许的 **Web** 来源添加到访问令牌 **allowed-origins** 声明中。
  - **microprofile-jwt**

*此范围处理 MicroProfile/JWT Auth 规范中定义的声明。此范围为 upn 声明定义用户属性映射器，以及 groups 声明的域角色映射器。这些映射程序可以更改，因此不同的属性可用于创建 MicroProfile/JWT 特定声明。*

- ***offline_access***

  *当客户端需要获取离线令牌时，可使用此范围。有关离线令牌的更多详细信息，请参阅离线令牌部分和 OpenID Connect 规格。*

- *配置集*

- ***email***

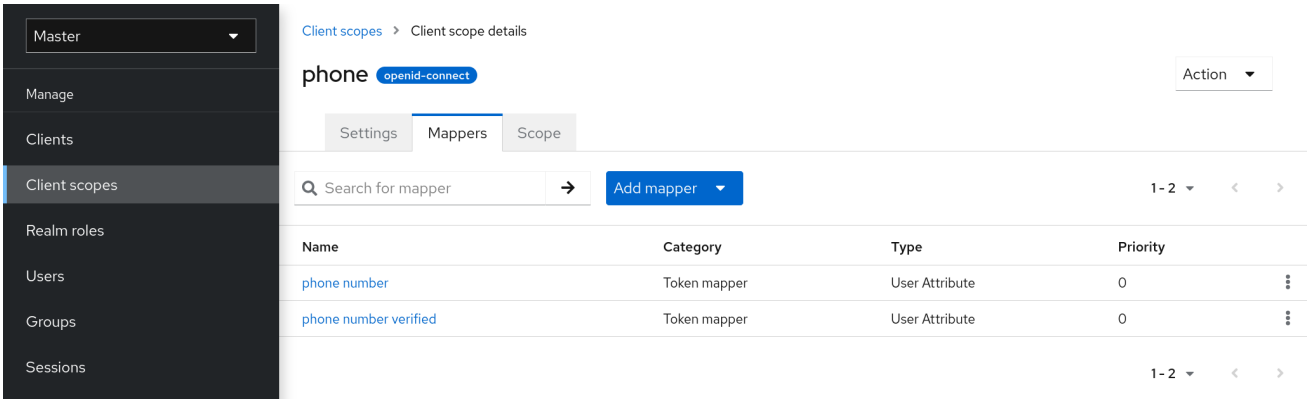
- ***address***

- *电话*

*客户端范围 配置文件、电子邮件、地址和 电话 在 OpenID Connect 规格中定义。这些范围没有定义任何角色范围映射，但它们定义了协议映射程序。这些映射程序与 OpenID Connect 规范中定义的声明对应。*

*例如，当您打开 电话 客户端范围并打开 Mappers 选项卡时，您将看到与范围 电话 规格中定义的协议映射器对应的协议映射程序。*

*客户端范围映射器*



*当 电话 客户端范围链接到客户端时，客户端会自动继承 手机 客户端范围中定义的所有协议映射程序。为此客户端发布的访问令牌包含有关用户的电话号码信息，假定用户有定义的电话号码。*

*内置客户端范围包含规格中定义的协议映射器。您可以自由编辑客户端范围，并创建、更新或删除任何协议映射程序或角色范围映射。*

#### *12.6.2. 同意相关设置*

*客户端范围包含与同意屏幕相关的选项。如果在客户端上启用了 **Consent Required**，则这些选项很有用。*

*显示 **Consent** 屏幕*

> *如果启用了 **Display On Consent Screen**，并且范围被添加到需要同意的客户端中，则会在同意屏幕上显示 **Consent Screen Text** 中指定的文本。当用户通过身份验证，并在用户从 **Keycloak** 的红帽构建重定向到客户端之前，会显示此文本。如果禁用 **Display On Consent Screen**，则不会在同意屏幕上显示此客户端范围。*

*同意屏幕文本*

> *当此客户端范围添加到客户端范围时，当同意需要默认为客户端范围名称时，在同意屏幕上显示的文本。此文本的值可以通过使用 **${var-name}** 字符串指定替换变量来自定义。自定义值在主题中的属性文件中配置。有关自定义的更多信息，请参阅 服务器开发人员指南。*

#### *12.6.3. 将客户端范围链接到客户端*

*客户端范围和客户端之间的链接在客户端的 **Client Scopes** 选项卡中配置。客户端范围和客户端之间链接有两种方法。*

*默认客户端范围*

> *此设置适用于 **OpenID Connect** 和 **SAML** 客户端。为客户端发出 **OpenID Connect** 令牌或 **SAML** 断言时应用默认客户端范围。客户端将继承客户端范围中定义的协议映射和角色范围映射。对于 **OpenID Connect** 协议，始终应用 **Mappers** 和 **Role Scope Mappings**，无论 **OpenID Connect** 授权请求中用于 **scope** 参数的值是什么。*

*可选的客户端范围*

> *此设置仅适用于 **OpenID Connect** 客户端。为这个客户端发布令牌时会应用可选客户端范围，但仅在 **OpenID Connect** 授权请求中的 **scope** 参数请求时应用可选客户端范围。*

##### *12.6.3.1. Example*

在本例中，假设客户端具有 配置文件，以及链接为默认客户端范围 的电子邮件，以及作为可选客户端范围链接的电话和 地址。客户端在发送请求到 **OpenID Connect** 授权端点时使用 **scope** 参数的值。

```
scope=openid phone
```

**scope** 参数包含字符串，范围值除以空格分开。值 **openid** 是所有 **OpenID Connect** 请求使用的 **meta-value**。令牌将包含来自默认客户端范围配置集和 电子邮件的 映射程序 和角色 范围映射，即 **scope** 参数请求的可选客户端范围。

### 12.6.4. 评估客户端范围

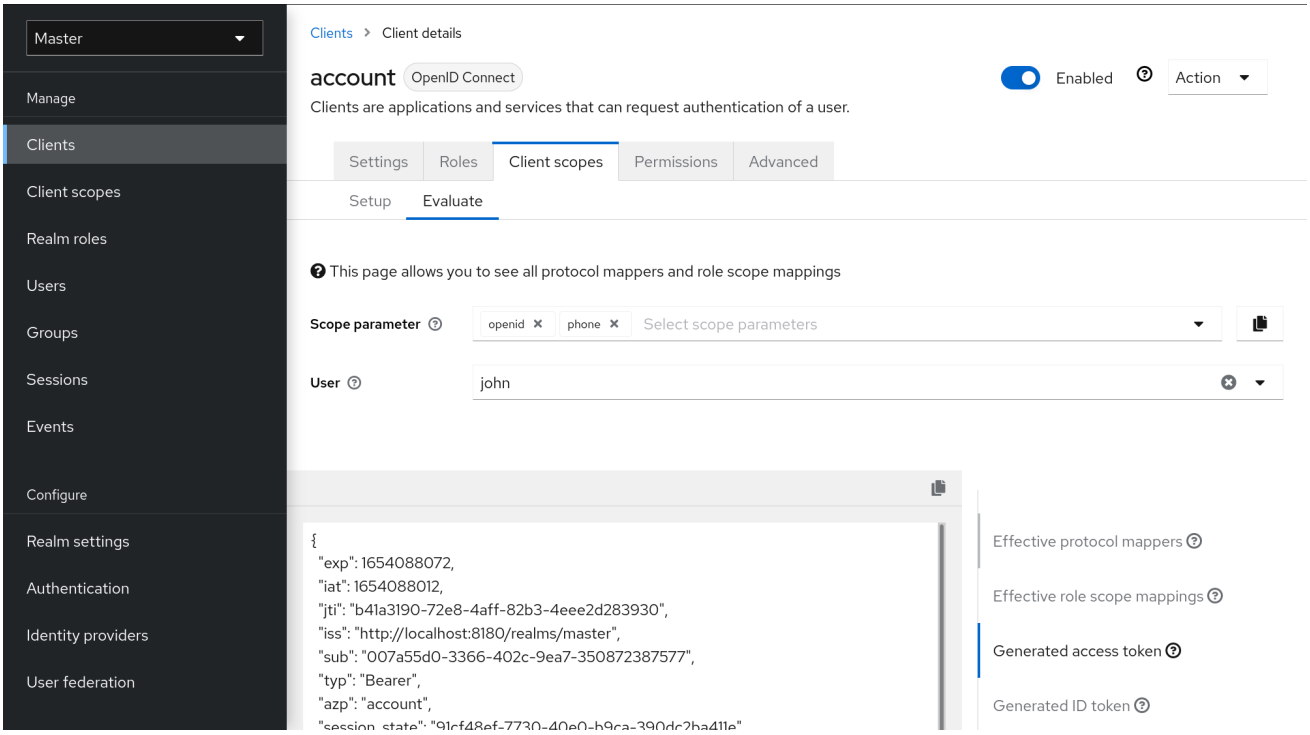
**Mappers** 选项卡包含协议映射程序，范围 选项卡包含为此客户端声明的角色范围映射。它们不包含从客户端范围继承的映射程序和范围映射。可以看到有效的协议映射程序（即客户端本身中定义的协议映射程序），以及从链接的客户端范围继承的有效角色范围映射，以及为客户端生成令牌时使用的有效角色范围映射。

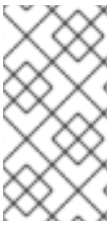
流程

1. 点客户端的 **Client Scopes** 选项卡。
2. 打开选项卡 评估。
3. 选择您要应用的可选客户端范围。

这还向您显示 **scope** 参数的值。此参数需要从应用程序发送到红帽构建的 **Keycloak OpenID Connect** 授权端点。

评估客户端范围



*注意*

*要从应用程序发送 **scope** 参数的自定义值，请参阅 **servlet** 适配器 的参数转发部分，或 **javascript** 适配器部分。*

*为特定用户生成所有示例，并为特定客户端发布，且指定的值为 **scope** 参数。示例包括所有声明和角色映射。*

### *12.6.5. 客户端范围权限*

*向用户发布令牌时，只有在允许用户使用令牌时才应用客户端范围。*

*当客户端范围没有定义任何角色范围映射时，允许每个用户使用此客户端范围。但是，当客户端范围定义了角色范围映射时，用户必须至少是其中一个角色的成员。用户角色和客户端范围的角色之间必须有一个交集。复合角色被计算为评估这个交集。*

*如果用户不允许使用客户端范围，则在生成令牌时不使用协议映射程序或角色范围映射。客户端范围不会出现在 令牌的范围 值中。*

### *12.6.6. 域默认客户端范围*

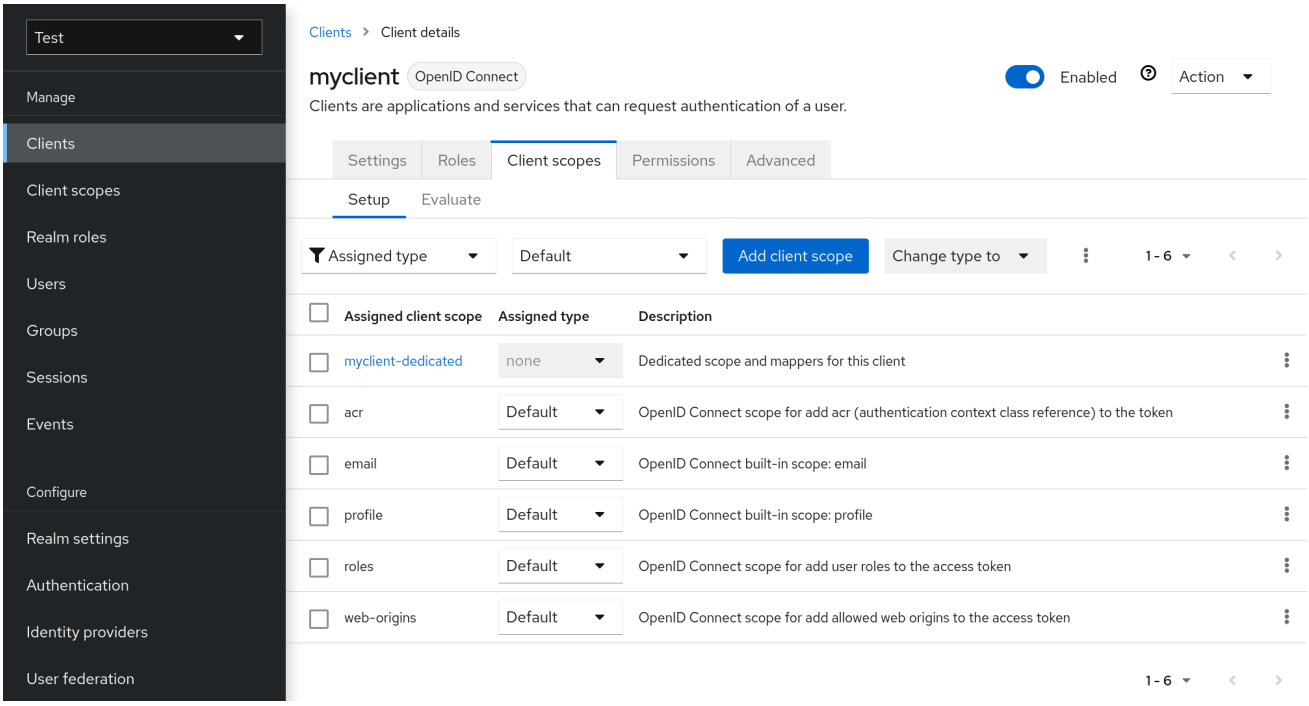
*使用 **Realm Default Client Scopes** 定义自动链接到新创建的客户端的客户端范围集合。*

*流程*

1. *点客户端的 **Client Scopes** 选项卡。*
2. *点 **Default Client Scopes**。*

*在这里，选择您要添加为 **Default Client Scopes** 的客户端范围到新创建的客户端和 **Optional Client Scopes**。*

*默认客户端范围*




*创建客户端时，如果需要，可以取消链接默认的客户端范围。这与删除 默认角色 类似。*

### *12.6.7. 范围解释*

*客户端范围*

*客户端范围是红帽构建的 **Keycloak** 中的实体，这些 **Keycloak** 在域级别配置，并可链接到客户端。当请求发送到红帽构建的 **Keycloak** 授权端点时，客户端范围会根据名称引用，其范围是 **scope***

参数的值。如需了解更多详细信息，请参阅 客户端范围链接 部分。

角色范围映射

这位于客户端或客户端范围的 **Scope** 选项卡下。使用角色范围映射 来限制访问令牌中可以使用的角色。如需了解更多详细信息，请参阅角色范围映射部分。

### 12.7. 客户端策略

为了便于保护客户端应用程序，以统一方式实现以下点很有用：

- 对客户端可以有的配置设置策略
- 验证客户端配置
- 遵守所需的安全标准和配置文件，如 **financial-grade API (FAPI)**和 **OAuth 2.1**

为了以统一的方式实现这些点，引入了 客户端策略 概念。

#### 12.7.1. 使用案例

客户端策略实现以下提到的点：

对客户端可以有的配置设置策略

客户端上的配置设置可以在客户端创建/更新期间由客户端策略实施，也可以在 **OpenID Connect** 请求到红帽 **Keycloak** 服务器（与特定客户端相关）期间强制执行。红帽构建的 **Keycloak** 还通过 保护应用程序和服务指南 中描述的客户端注册策略支持类似的操作。但是，客户端注册策略只能涵盖 **OIDC** 动态客户端注册。客户端策略不仅涵盖客户端注册策略可以执行的操作，还涵盖其他客户端注册和配置方式。当前计划用于客户端注册，以替换为客户端策略。

验证客户端配置

红帽构建的 **Keycloak** 支持验证客户端是否遵循代码交换、请求对象签名算法、保存密钥令牌等设置，等等。它们可以通过每个设置项（在管理控制台、切换、下拉菜单等）来指定。为了使客户端应用程序安全，管理员需要以适当的方式设置许多设置，从而让管理员难以保护客户端应用。客户端策略可以对上述客户端配置进行这些验证，它们也可用于自动配置一些客户端配置交换机来满足高级安全要求。以后，单个客户端配置设置可能会被 客户端策略直接执行所需的验证替代。

符合所需的安全标准和配置集，如 **FAPI** 和 **OAuth 2.1**

默认情况下，全局客户端配置集 是在红帽构建的 **Keycloak** 中预先配置的客户端配置集。它们预先配置为符合 **FAPI** 和 **OAuth 2.1** 等标准安全配置文件，因此管理员可以轻松地保护其客户端应用符合特定的安全配置文件。目前，红帽构建的 **Keycloak** 具有支持 **FAPI** 和 **OAuth 2.1** 规范的全局配置集。管理员只需要配置客户端策略，以指定哪些客户端应与 **FAPI** 和 **OAuth 2.1** 兼容。管理员可以配置客户端配置文件和客户端策略，以便红帽构建的 **Keycloak** 客户端可轻松与各种其他安全配置集兼容，如 **SPA**、**Native App**、**Open banking** 等。

#### 12.7.2. 协议

客户端策略概念独立于任何特定的协议。但是，红帽构建的 **Keycloak** 目前只支持 **OpenID Connect (OIDC)**协议。

#### 12.7.3. 架构

客户端策略由四个构建块组成：**Condition**、**Executor**、**Profile** 和 **Policy**。

##### 12.7.3.1. 状况

条件决定策略被采用的客户端以及何时采用。当在客户端请求过程中检查一些其他条件**(OIDC Authorization request, Token endpoint request** 等**)**，在客户端创建/更新时检查一些条件。该条件检查是否满足一个指定的条件。例如，一些条件检查客户端的访问类型是机密。

该条件不能自行使用。它可用于之后描述 的策略。

条件可以与其他可配置提供程序相同。可以配置的内容取决于每个条件的性质。

提供以下条件：

创建/更新客户端的方法

- 动态客户端注册（具有初始访问令牌或注册访问令牌的非匿名或授权）
- 管理 **REST API** （管理员控制台等）

例如，在创建客户端时，当没有初始访问令牌（非动态客户端注册）的情况下，可将条件配置为由 **OIDC Dynamic Client Registration** 创建为 **true**。因此，这个条件可用于确保通过 **OIDC** 动态客户端注

*册的所有客户端都兼容 **FAPI** 或 **OAuth 2.1**。*

*客户端作者（通过存在特定角色或组来检查）*

*在 **OpenID Connect** 动态客户端注册中，客户端的作者是经过身份验证以获取生成新客户端的访问令牌的最终用户，而不是实际通过访问令牌访问注册端点的现有客户端的服务帐户。在由 **Admin REST API** 注册时，客户端的作者是类似红帽构建的 **Keycloak** 管理员的最终用户。*

*客户端访问类型（机密、公共、仅 **bearer**）*

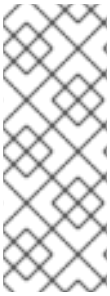
*例如，当客户端发送授权请求时，如果此客户端机密，则使用策略。当公共客户端禁用了客户端身份验证时，机密客户端启用了客户端身份验证。**bearer-only** 是已弃用的客户端类型。*

*客户端范围*

*如果客户端具有特定的客户端范围（为默认值或当前请求中使用的可选范围，则评估为 **true**）。例如，这可用于确保具有范围 **fapi-example-scope** 的 **OIDC** 授权请求需要兼容 **FAPI**。*

*客户端角色*

*适用于具有指定名称的客户端角色。通常，您可以创建指定名称的客户端角色来请求客户端，并将其用作"标记角色"，以确保将针对请求的客户端策略应用指定的客户端策略。*



*注意*

*对于需要特定客户端应用（如 **my-client-1** 和 **my-client-2**）的用例，通常存在 用例。实现此结果的最佳方法是在您的策略中使用 客户端角色 条件，然后创建指定名称的客户端角色来请求客户端。此客户端角色可用作"标记角色"，仅用于为特定客户端标记该特定客户端策略。*

*客户端域名、主机或 **IP** 地址*

*适用于客户端的特定域名。或者，当管理员从特定主机或 **IP** 地址注册/更新客户端时。*

*任何客户端*

*此条件始终评估为 **true**。例如，它可用于确保特定域中的所有客户端都兼容 **FAPI**。*

#### *12.7.3.2. executor*

***executor** 指定在采用策略的客户端上执行哪些操作。**executor** 执行一个或多个指定操作。例如，一些 **executor** 会检查授权请求中的参数 **redirect_uri** 的值是否与授权端点上预注册的重定向 **URI** 完全匹配，并在未授权端点上拒绝此请求。*

*executor 本身不能单独使用。它可用于之后描述 的配置集。*

*executor 可以与其他可配置供应商相同。可以配置的内容取决于每个 executor 的性质。*

*executor 会对各种事件执行操作。executor 的实现可以忽略某些类型的事件（例如，检查 OIDC 请求 对象的 executor 仅适用于 OIDC 授权请求）。事件是：*

- *创建客户端（包括通过动态客户端注册创建）*
- *更新客户端*
- *发送授权请求*
- *发送令牌请求*
- *发送令牌刷新请求*
- *发送令牌撤销请求*
- *发送令牌内省请求*
- *发送 userinfo 请求*
- *使用刷新令牌发送注销请求（请注意，使用刷新令牌退出时，任何规格不支持的 Keycloak 功能的专有红帽构建）。相反，建议您依赖 官方 OIDC 注销。*

*在每个事件中，executor 都可以在多个阶段工作。例如，在创建/更新客户端时，executor 可以通过自动配置特定的客户端设置来修改客户端配置。之后，executor 会在验证阶段验证此配置。*

*此 executor 的几个目的之一是实现客户端一致性配置集的安全要求，如 FAPI 和 OAuth 2.1。要做到*

这一点，需要以下 **executors**：

- 强制使用安全 客户端身份验证方法进行 客户端
- 使用 **enforce Holder-of-key** 令牌
- 使用 代码交换强制概念验证**(PKCE)**
- 使用 **Signed JWT** 客户端身份验证**(private-key-jwt)** 的安全签名算法
- 强制 **HTTPS** 重定向 **URI**，并确保配置的重定向 **URI** 不包含通配符
- 强制 **OIDC** 请求 对象满足高级别
- 对 **OIDC Hybrid Flow** 强制执行响应类型，包括用作 分离签名 的 **ID** 令牌，如 **FAPI 1** 规格中所述，这意味着从授权响应返回的 **ID Token** 不包含用户配置集数据
- 强制更安全的 状态 和非**ce** 参数处理以防止 **CSRF**
- 在客户端注册时强制实施更安全的签名算法
- **enforce binding_message** 参数用于 **CIBA** 请求
- 强制 客户端 **Secret** 轮转
- 强制客户端注册访问令牌
- 强制检查客户端是否为在启动授权代码流之前发出意图的用例中发布的意图，以获取英国 **OpenBanking** 等访问令牌

- 强制禁止隐式和混合流
- 强制检查 **PAR** 请求是否包含授权请求中包含的必要参数
- 使用强制 **DPoP-binding** 令牌 （在启用 **dpop** 功能时可用）
- 使用轻量级访问令牌强制
- 强制跳过 刷新令牌轮转，且不会从刷新令牌响应返回刷新令牌
- 强制 **OAuth 2.1** 规格所需的有效重定向 **URI**

#### 12.7.3.3. profile

配置集由多个 **executors** 组成，它可以实现一个安全配置集，如 **FAPI** 和 **OAuth 2.1**。配置集可以通过 **Admin REST API (Admin Console)**与其 **executors** 一起配置。有三个 全局配置集，它们默认在 **Keycloak** 的红帽构建中配置，并带有预配置的 **executors**，与 **FAPI 1 Baseline, FAPI 1 Advanced, FAPI CIBA, FAPI 2** 和 **OAuth 2.1** 规格兼容。安全应用程序和服务 指南的 **FAPI** 和 **OAuth 2.1** 部分中存在更多详细信息。

#### 12.7.3.4. policy

策略由多个条件和配置集组成。该策略可用于满足此策略的所有条件的客户端。该策略指的是多个配置集，这些配置集的所有 **executors** 会根据这个策略的客户端执行其任务。

### 12.7.4. Configuration

策略、配置集、条件、**executors** 可由 **Admin REST API** 配置，这意味着管理控制台。为此，有一个标签 **Realm → Realm Settings → Client Policies**，这意味着管理员可以为每个域具有客户端策略。

全局客户端配置集 在每个域中自动可用。但是，默认情况下不配置任何客户端策略。这意味着，如果管理员希望其域的客户端符合 **FAPI**，则始终需要创建任何客户端策略。全局配置集无法更新，但管理员可轻松将它们用作模板，并在需要在全局配置集配置中进行一些小修改的情况下创建自己的配置集。**Admin Console** 中提供了 **JSON** 编辑器，它简化了基于某些全局配置集的新配置集的创建。

#### 12.7.5. 向后兼容性

客户端策略可以替换 保护应用程序和服务指南 中描述的客户端注册 策略。但是，客户端注册政策仍然是共存的。这意味着，在动态客户端注册请求创建/更新客户端期间，会应用客户端策略和客户端注册策略。

当前计划是要删除的客户端注册策略功能，并且现有客户端注册策略将自动迁移到新的客户端策略中。

#### 12.7.6. 客户端 Secret 轮转示例

请参阅 客户端 secret 轮转的示例配置。

# 第 13 章 使用 VAULT 获取 SECRET

红帽构建的 Keycloak 目前提供 Vault SPI 的两个开箱即用实现：基于纯文本文件的 vault 和基于 Java KeyStore 的 vault。

要从密码库获取 secret，而不是直接输入它，请在适当的字段中输入以下特殊设计的字符串：

```
${vault.key}
```

其中，键是 密码库识别的 secret 的名称。

为了防止 secret 泄漏跨域，红帽构建的 Keycloak 会将域名与从 vault 表达式获取 的密钥 合并。此方法意味着 密钥 不会直接映射到密码库中的条目，而是根据用于组合键和域名称的算法创建最终条目名称。如果是基于文件的库，此类组合反映了特定文件名，对于基于 Java KeyStore 的库， 它是特定的别名名称。

您可以在以下字段从 vault 获取 secret：

**SMTP 密码**

在 realm SMTP 设置中

**LDAP 绑定凭证**

在基于 LDAP 的用户联邦的 LDAP 设置中。

**OIDC 身份提供程序 secret**

在身份提供程序 OpenID Connect Config中的 Client Secret 中

## 13.1. 密钥解析器

所有内置供应商都支持配置密钥解析器。密钥解析器实施算法或策略，用于将域名与从 ${vault.key} 表达式获取的键合并到用于从密码库检索 secret 的最终条目名称。Red Hat build of Keycloak 使用 keyResolvers 属性来配置供应商使用的解析器。该值是一个以逗号分隔的解析器名称列表。files-plaintext 供应商配置示例如下：

```
kc.[sh|bat] start --spi-vault-file-key-resolvers=REALM_UNDERSCORE_KEY,KEY_ONLY
```

*解析器会按照您在配置中声明的顺序运行相同的顺序。对于每个解析器，红帽构建的 **Keycloak** 使用解析器生成的最后一个条目名称，它将 **realm** 与 **vault** 密钥合并，以搜索 **vault** 的 **secret**。如果红帽构建的 **Keycloak** 找到 **secret**，它会返回 **secret**。如果没有，**Red Hat build of Keycloak** 会使用下一个解析器。这个搜索会一直进行，直到红帽构建的 **Keycloak** 找到非空的 **secret** 或不使用解析器。如果红帽构建的 **Keycloak** 找不到 **secret**，红帽构建的 **Keycloak** 会返回一个空的 **secret**。*

*在上例中，红帽构建的 **Keycloak** 首先使用 **REALM_UNDERSCORE_KEY** 解析器。如果红帽构建的 **Keycloak** 在使用该解析器的 **vault** 中查找条目，红帽构建的 **Keycloak** 会返回该条目。如果没有，红帽使用 **KEY_ONLY** 解析器再次搜索 **Keycloak**。如果红帽构建的 **Keycloak** 使用 **KEY_ONLY** 解析器查找条目，红帽构建的 **Keycloak** 会返回该条目。如果红帽构建的 **Keycloak** 使用所有解析器，红帽构建的 **Keycloak** 会返回一个空的 **secret**。*

*当前可用的解析器列表如下：*

| Name | 描述 |
|---|---|
| KEY_ONLY | 红帽构建的 Keycloak 忽略了域名，并使用 vault 表达式中的密钥。 |
| REALM_UNDERSCORE_KEY | Red Hat build of Keycloak 使用下划线字符组合 realm 和 key。红帽构建的 Keycloak 转义在 realm 或 key 中出现带有另一个下划线字符的下划线。例如，如果域名为 **master_realm** 且密钥是 **smtp_key**，则组合键是 **master__realm_smtp__key**。 |
| REALM_FILESEPARATOR_KEY | Red Hat build of Keycloak 使用平台文件分隔符字符组合 realm 和 key。 |

*如果您还没有为内置提供程序配置解析器，红帽构建的 **Keycloak** 将选择 **REALM_UNDERSCORE_KEY**。*

# 第 14 章 配置审计来跟踪事件

红帽构建的 Keycloak 包括一组审计功能。您可以记录每个登录和管理员操作，并在管理控制台中查看这些操作。红帽构建的 Keycloak 还包括一个 Listener SPI，它侦听事件并可触发操作。内置监听程序的示例包括日志文件并在发生事件时发送电子邮件。

## 14.1. 审计用户事件

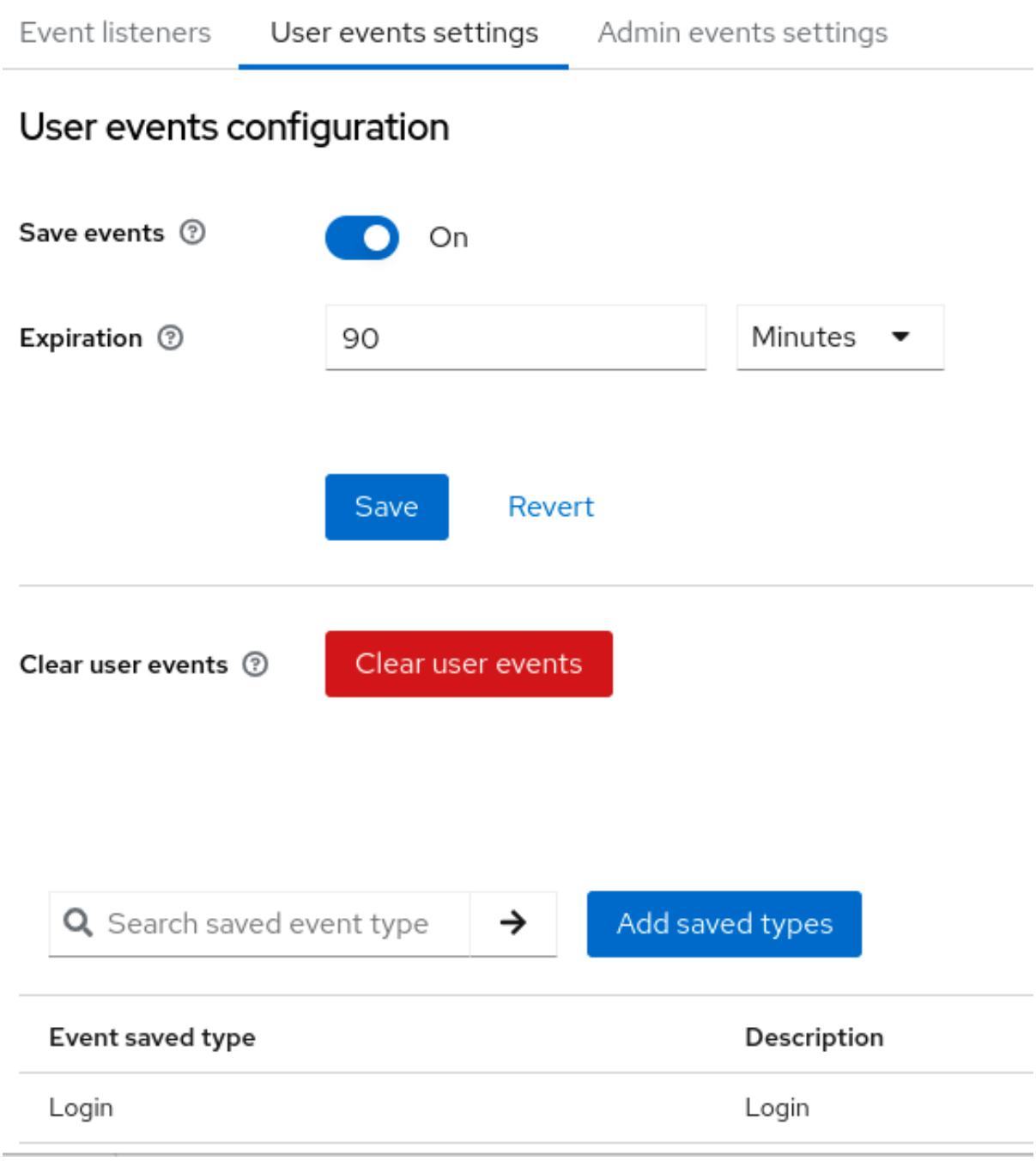
您可以记录和查看影响用户的每个事件。红帽构建的 Keycloak 会触发登录事件，如成功用户登录、输入错误密码或用户帐户更新等操作。默认情况下，红帽构建的 Keycloak 不存储或显示管理控制台中的事件。只有错误事件会记录到管理控制台和服务器的日志文件。

### 流程

使用这个流程开始审核用户事件。

1. 点菜单中的 **Realm settings**。
2. 单击 **Events** 选项卡。
3. 点 **User events settings** 选项卡。
4. 将 **Save** 事件 切换为 **ON**。

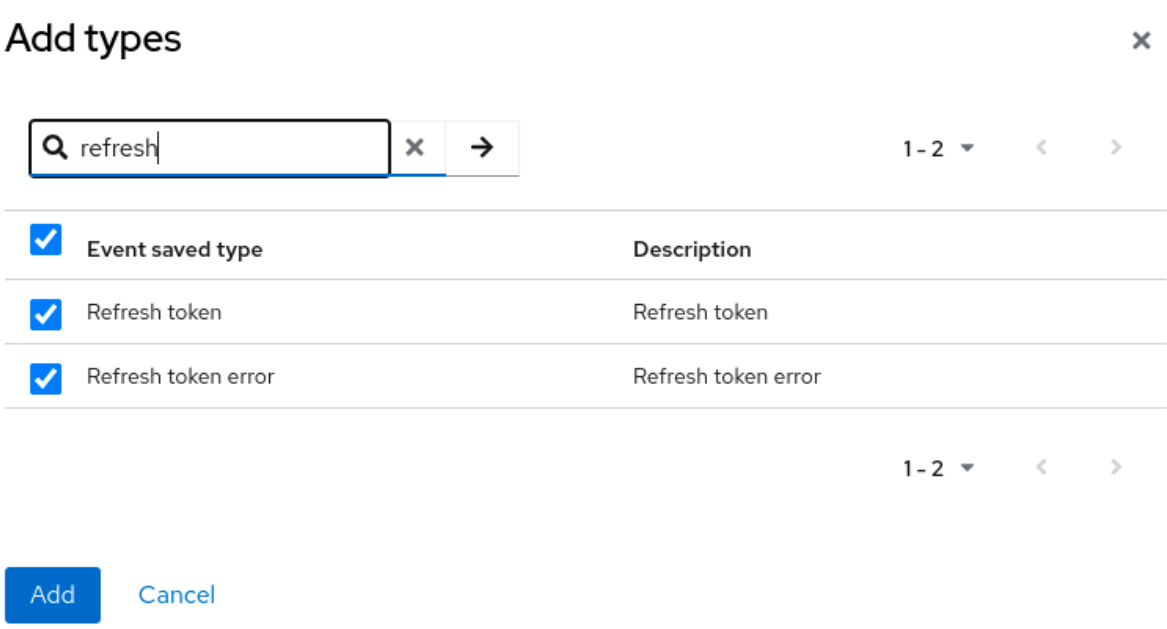
用户事件设置




5. *指定在 **Expiration** 字段中存储事件的时间长度。*

6. *点 **Add saved types** 查看您可以保存的其他事件。*

*添加类型*




7. *点击 **Add**。*

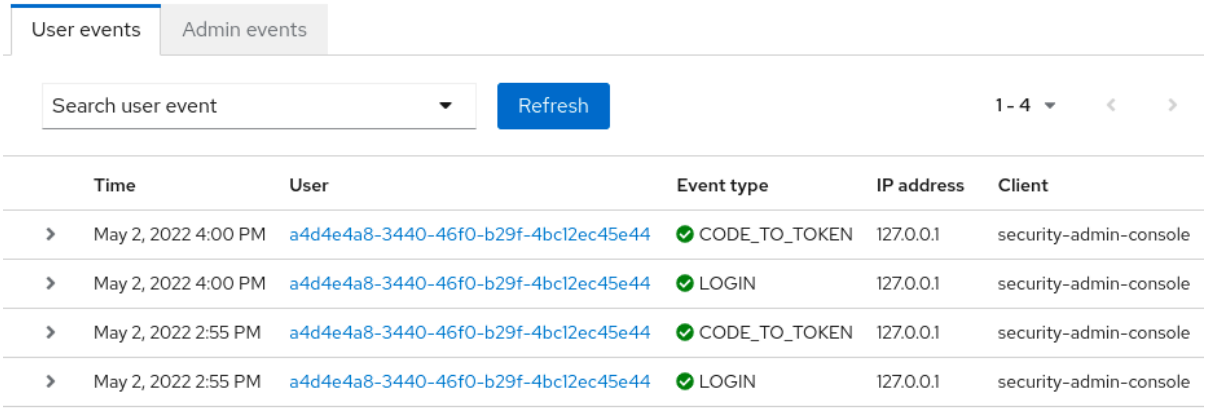
*当您要删除所有保存的事件时，点 **Clear user events**。*

*流程*

*现在，您可以查看事件。*

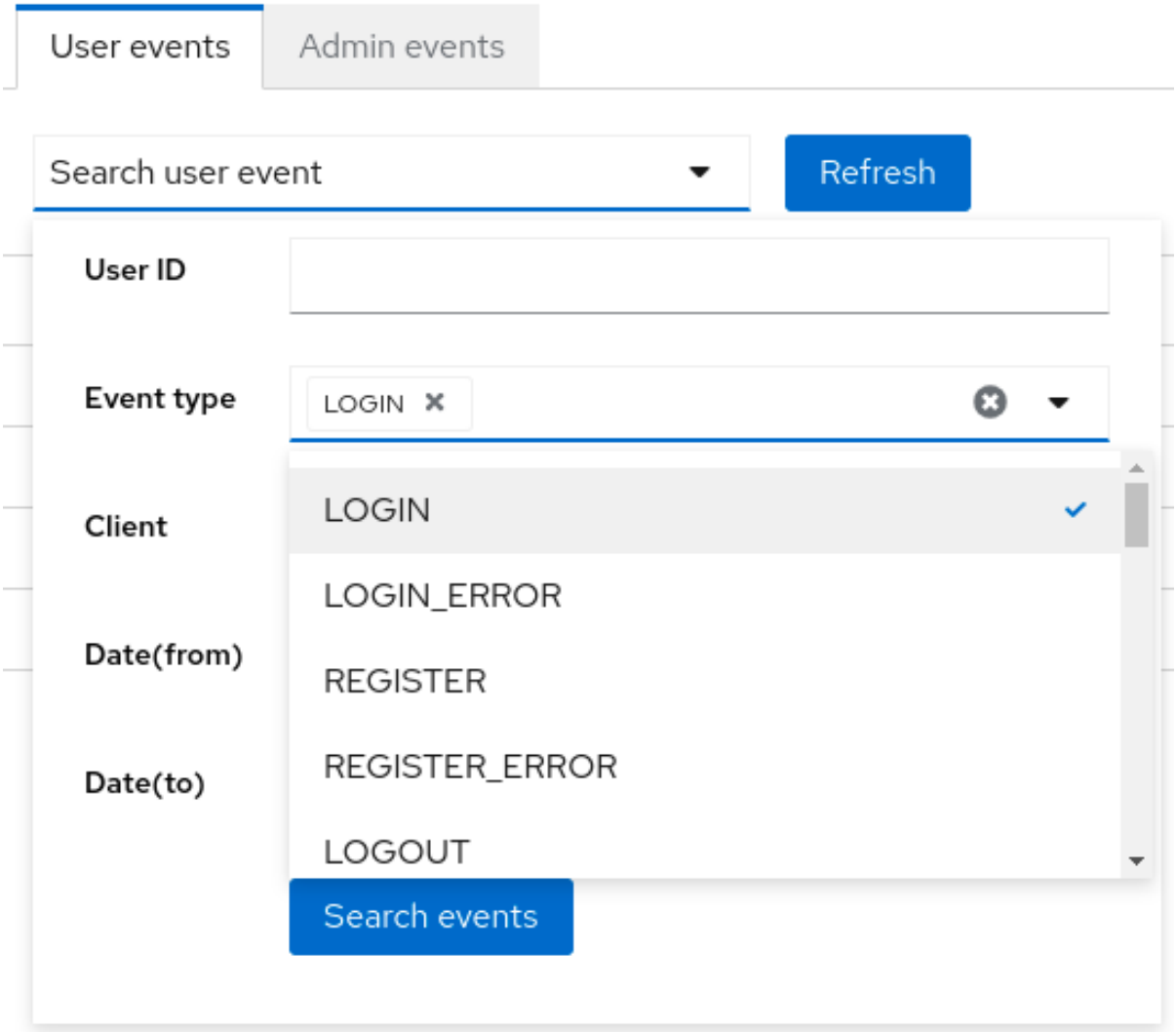
1. *单击菜单中的 **Events** 选项卡。*

*用户事件*



2. *要过滤事件，请点* ***Search user event****。*

*搜索用户事件*




### *14.1.1. 事件类型*

*登录事件：*

| 事件 | 描述 |
| --- | --- |
| 登录 | 用户登录。 |
| 注册 | 用户注册。 |

| 事件 | 描述 |
| --- | --- |
| 退出 | 用户注销。 |
| 令牌代码 | 应用或客户端交换令牌的代码。 |
| 刷新令牌 | 应用或客户端会刷新令牌。 |

***brute** 强制保护：*

| 事件 | 描述 |
| --- | --- |
| 由永久锁定禁用的用户 | 静默强制因为太多登录失败而永久禁用用户帐户。 |
| 通过临时锁定禁用的用户 | 静默强制因为太多登录失败而临时禁用用户帐户。 |

*帐户事件：*

| 事件 | 描述 |
| --- | --- |
| 社交链接 | 用户帐户链接到社会媒体提供商的链接。 |
| 删除交换链接 | 从社交介质帐户链接到用户帐户 severs。 |
| 更新电子邮件 | 帐户更改的电子邮件地址。 |
| 更新配置文件 | 帐户更改的配置集。 |
| 发送密码重置 | 红帽构建的 Keycloak 会发送密码重置电子邮件。 |
| 更新密码 | 帐户更改的密码。 |
| 更新 TOTP | 帐户更改的基于时间的一次性密码(TOTP)设置。 |
| 删除 TOTP | 红帽构建的 Keycloak 从帐户中删除 TOTP。 |
| 发送验证电子邮件 | 红帽构建的 Keycloak 发送电子邮件验证电子邮件。 |
| 验证电子邮件 | 红帽构建的 Keycloak 会验证帐户的电子邮件地址。 |

每个事件都有对应的错误事件。

#### 14.1.2. 事件监听程序

事件监听器侦听事件并根据该事件执行操作。红帽构建的 **Keycloak** 包括两个内置监听程序，即 **Logging Event Listener** 和 **Email Event Listener**。

##### 14.1.2.1. 日志记录事件监听程序

当启用 **Logging Event Listener** 时，此监听程序会在出现错误事件时写入日志文件。

**Logging Event Listener** 的日志消息示例：

```
11:36:09,965 WARN  [org.keycloak.events] (default task-51) type=LOGIN_ERROR, realmId=master,
        clientId=myapp,
        userId=19aeb848-96fc-44f6-b0a3-59a17570d374, ipAddress=127.0.0.1,
        error=invalid_user_credentials, auth_method=openid-connect, auth_type=code,
        redirect_uri=http://localhost:8180/myapp,
        code_id=b669da14-cdbb-41d0-b055-0810a0334607, username=admin
```

您可以使用 **Logging Event Listener** 来防止黑客 **bot** 攻击：

1. 解析 **LOGIN_ERROR** 事件的日志文件。
2. 提取失败的登录事件的 **IP** 地址。
3. 将 **IP** 地址发送到入侵防止软件框架工具。

**Logging Event Listener** 将事件记录到 **org.keycloak.events** 日志类别。默认情况下，**Red Hat build of Keycloak** 不会在服务器日志中包括调试日志事件。

在服务器日志中包括 **debug** 日志事件：

1. 更改 **org.keycloak.events** 类别的日志级别

2. 更改日志记录事件监听器使用的日志级别。

要更改日志记录事件监听程序使用的日志级别，请执行以下操作：

```
bin/kc.[sh|bat] start --spi-events-listener-jboss-logging-success-level=info --spi-events-listener-jboss-logging-error-level=error
```

日志级别的有效值为 **debug**、**info**、**warn**、**error**、**error** 和 **fatal**。

#### 14.1.2.2. 电子邮件事件 Listener

当事件发生并支持以下事件时，电子邮件事件 **Listener** 会向用户帐户发送电子邮件：

- 登录错误。
- 更新密码.
- 更新基于时间的一次性密码(TOTP)。
- 删除基于时间的一次性密码(TOTP)。

流程

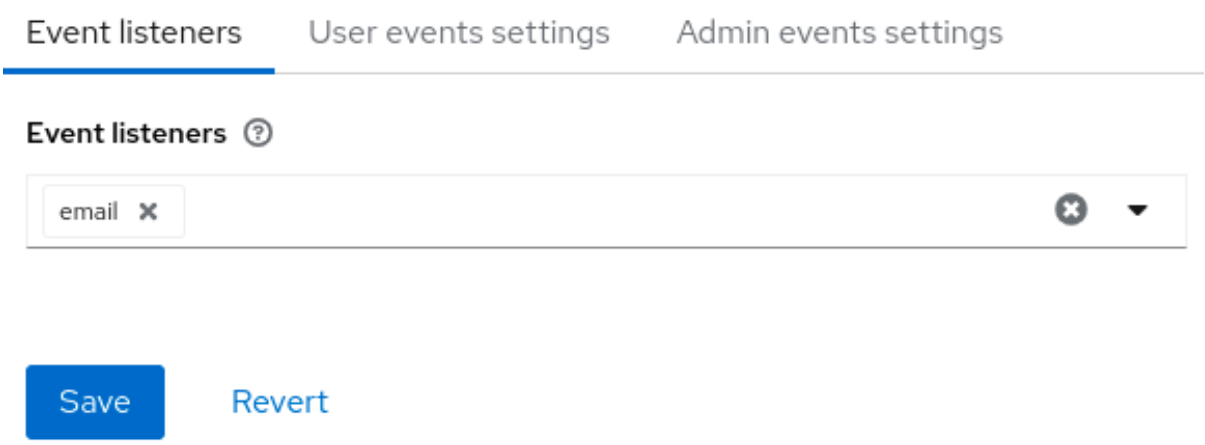
启用 **Email Listener**：

1. 点菜单中的 **Realm settings**。
2. 单击 **Events** 选项卡。

3. 点 **Event listeners** 字段。

4. 选择 电子邮件。

事件监听程序




您可以使用 **--spi-events-listener-email-exclude-events** 参数排除事件。例如：

```
kc.[sh|bat] --spi-events-listener-email-exclude-events=UPDATE_TOTP,REMOVE_TOTP
```

### 14.2. 审计管理事件

您可以在管理控制台中记录管理员执行的所有操作。管理控制台通过调用红帽构建的 **Keycloak REST** 接口和红帽构建的 **Keycloak** 审计来执行管理操作。您可以在控制台中查看生成的事件。

流程

使用这个流程开始审核管理操作。

1. 点菜单中的 **Realm settings**。

2. 单击 **Events** 选项卡。

3. *单击 **Admin events settings** 选项卡。*

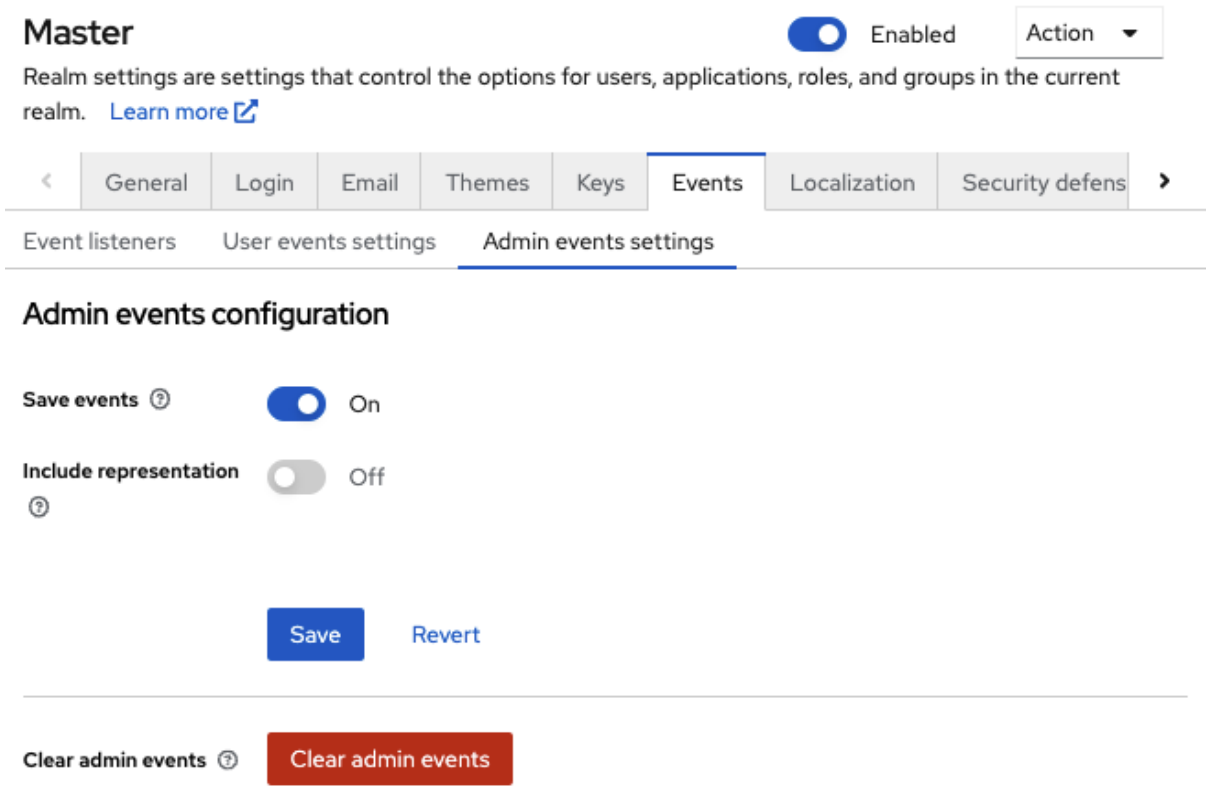
4. *将 **Save** 事件 切换为 **ON**。*

   ***Red Hat build of Keycloak** 显示 **Include** 表示 开关。*

5. *将 **Include** 表示 切换为 **ON**。*

   ***Include Representation** 开关包括通过 **admin REST API** 发送的 **JSON** 文档，以便您可以查看管理员操作。*

*管理事件设置*




6. *点击 **Save**。*

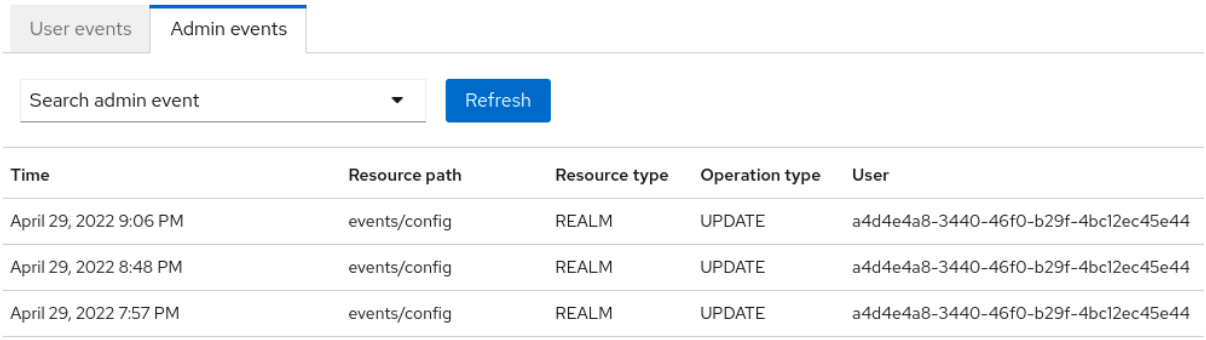
7. *要清除存储操作的数据库，请单击 **Clear admin events**。*

*流程*

*现在，您可以查看 **admin** 事件。*

1. *单击菜单中的 **Events**。*
2. *单击 **Admin events** 选项卡。*

*管理事件*




*当 **Include Representation** 开关为 **ON** 时，可能会导致将大量信息存储在数据库中。您可以使用 **--spi-events-store-jpa-max-field-length** 参数设置表示的最大长度。如果要遵循底层存储限制，则此设置很有用。例如：*

```
kc.[sh|bat] --spi-events-store-jpa-max-field-length=2500
```

# 第 15 章 缓解安全威胁

任何身份验证服务器中都存在安全漏洞。如需更多信息，请参阅互联网工程任务 Force's (IETF) OAuth 2.0 Threat Model 和 OAuth 2.0 Security best Current practice。

## 15.1. 主机

红帽构建的 Keycloak 以多种方式使用公共主机名，如令牌签发者字段和密码重置电子邮件中的 URL。

默认情况下，主机名从请求标头派生。不存在验证以确保主机名有效。如果您没有在 Red Hat build of Keycloak 中使用负载均衡器或代理，以防止无效的主机标头，请配置可接受的主机名。

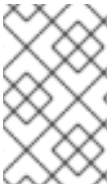
主机名的服务提供商接口(SPI)提供了一种为请求配置主机名的方法。您可以使用此内置供应商为前端请求设置固定 URL，同时根据请求 URI 允许后端请求。如果内置供应商没有所需的功能，您可以开发自定义供应商。

## 15.2. 管理端点和管理控制台

红帽构建的 Keycloak 会在与非管理员用户相同的端口上公开管理 REST API 和 Web 控制台。如果不需要外部访问，请不要向外部公开管理端点。

## 15.3. BRUTE 强制攻击

静默强制通过尝试多次登录来尝试猜测用户的密码。Red Hat build of Keycloak 具有 brute 强制检测功能，如果登录失败次数超过指定的阈值，可以临时禁用用户帐户。



注意

红帽构建的 Keycloak 默认禁用 brute 强制检测。启用此功能可防止防止暴力攻击。

流程

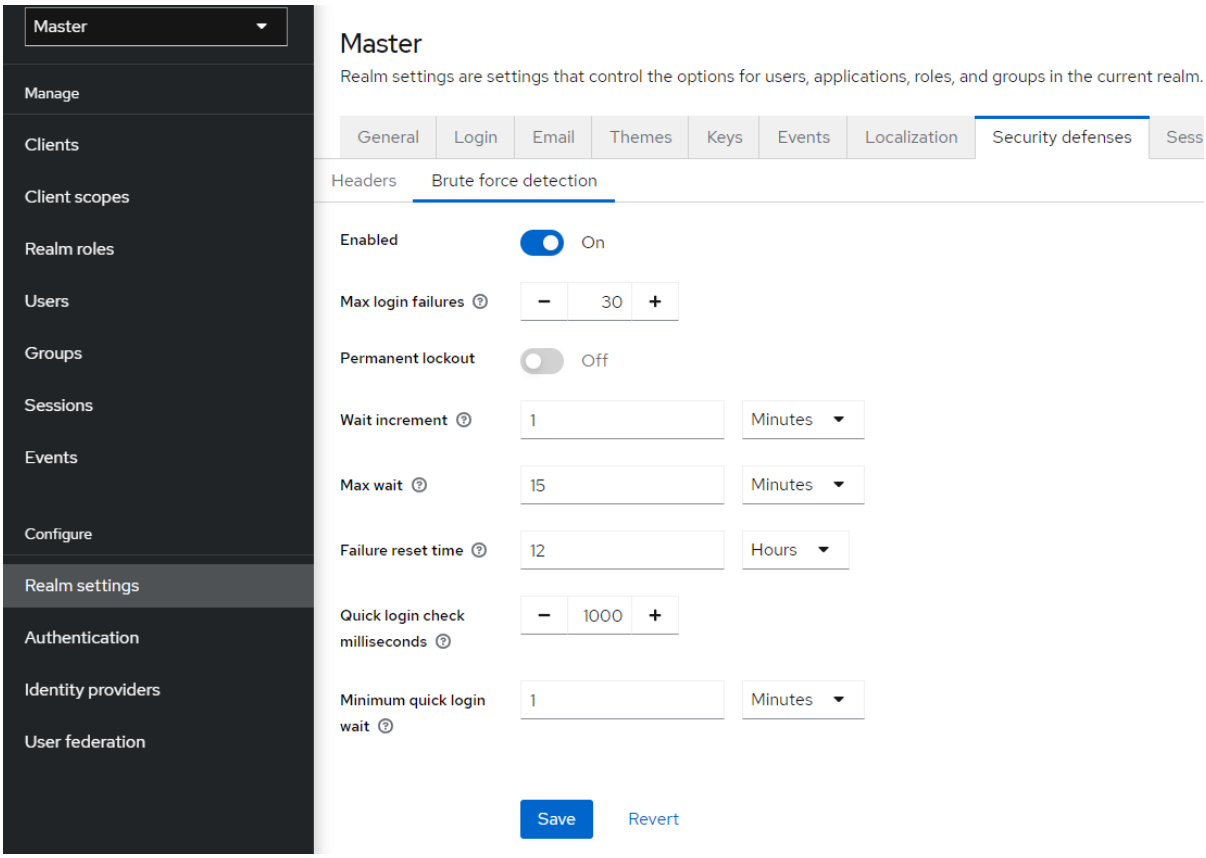
启用此保护：

1. *在菜单中点 **Realm Settings***

2. *点 **Security Defenses** 选项卡。*

3. *点 **Brute Force Detection** 选项卡。*

***brute** 强制检测*




*红帽构建的 **Keycloak** 可以在检测到攻击时部署永久锁定和临时锁定操作。永久锁定会禁用用户帐户，直到管理员重新启用它。临时锁定会禁用特定时间段内的用户帐户。当攻击持续增加时，帐户被禁用的时间周期会增加，后续的故障到达 **Max Login Failures** 的倍数。*

*注意*

*当用户被临时锁定并尝试登录时，红帽构建的 **Keycloak** 会显示默认的 **Invalid username** 或 **password** 错误消息。这个消息与为无效用户名或无效密码显示的消息相同，以确保攻击者不知道帐户已被禁用。*

*常见参数*

| Name | 描述 | 默认 |
| --- | --- | --- |
| 最大登录失败 | 登录失败的最大数量。 | 30 个故障。 |
| 快速登录检查 Milliseconds | 登录尝试之间的最短时间。 | 1000 毫秒。 |
| 最小快速登录等待 | 当登录尝试速度快于 *Quick Login Check Milliseconds* 时，用户会被禁用的最短时间。 | 1 分钟. |

*临时锁定参数*

| Name | 描述 | 默认 |
| --- | --- | --- |
| wait Increment | 当用户登录尝试超过 *Max Login* Failure 时，添加到用户的时间会被临时禁用。 | 1 分钟. |
| Max Wait | 用户暂时禁用的最长时间。 | 15 分钟. |
| 失败重置时间 | 失败计数重置的时间。计时器从上次失败的登录运行。确保这个数字始终大于 **Max wait**; 否则有效的等待时间永远不会达到您被设置为 **Max wait** 的值。 | 12 小时. |

*临时锁定算法*

1. *成功登录*
    a. *重置 计数*
2. *登录失败时*
    a. *如果此失败和最后一次失败之间的时间大于 **Failure Reset Time***

i.
*重置 计数*

b.
*增加 计数*

c.
*使用 **Wait Increment** * 计算等待 (计数 / **Max Login Failures**)。该部门是一个整数部门，舍入为一个整数*

d.
*如果 等待 等于 **0**，且最后一次失败的时间小于 **Quick Login Check Milliseconds**，请将 **wait to Minimum Quick Login Wait**。*

i.
*临时禁用用户 等待 和 **Max Wait** 秒的最小时间*

ii.
*增加临时锁定计数器*

*当临时禁用的帐户提交登录失败时，计数 不会递增。*

*例如，如果您将 **Max Login Failures** 设置为 **5**，**Wait Increment** 为 **30** 秒，则在多次验证尝试失败后将禁用帐户的有效时间：*

| 失败数 | wait Increment | 最大登录失败 | 有效等待时间 |
|---|---|---|---|
| 1 | 30 | 5 | 0 |
| 2 | 30 | 5 | 0 |
| 3 | 30 | 5 | 0 |
| 4 | 30 | 5 | 0 |
| **5** | **30** | 5 | **30** |
| 6 | 30 | 5 | 30 |
| 7 | 30 | 5 | 30 |
| 8 | 30 | 5 | 30 |

| | | | |
|---|---|---|---|
| 9 | 30 | 5 | 30 |
| 10 | 30 | 5 | 60 |

*请注意，**Effective Wait Time at the 5th failed trying** 将禁用帐户 **30** 秒。只有达到下一个 **Max Login Failure**（本例中为 **10**）后，将时间从 **30** 增加到 **60**。只有在达到 **Max Login Failures** 的倍数时，才会禁用该帐户的时间。*

*永久锁定参数*

| Name | 描述 | default |
|---|---|---|
| 最大临时锁定 | 在发生永久锁定前允许的最大临时锁定数。 | 0 |

*永久锁定流*

1. *遵循临时锁定流*
2. *如果临时锁定计数器超过 **Max temporary lockouts***
    a. *永久禁用用户*

*当红帽构建的 **Keycloak** 禁用用户时，用户无法登录，直到管理员启用了用户。启用帐户会重置 计数。*

*红帽构建的 **Keycloak brute** 检测是服务器易受拒绝服务攻击的影响。当实施拒绝服务攻击时，攻击者可以尝试通过猜测其知道的任何帐户的密码登录，并最终导致红帽构建的 **Keycloak** 禁用帐户。*

*考虑使用入侵防止软件**(IPS)**。红帽构建的 **Keycloak** 都会记录每个登录失败和客户端 **IP** 地址失败。您可以将 **IPS** 指向红帽构建的 **Keycloak** 服务器日志文件，**IPS** 可以修改防火墙来阻止这些 **IP** 地址的连接。*

### *15.3.1. 密码策略*

确保您有一个复杂的密码策略来强制用户选择复杂的密码。如需更多信息，请参阅 密码策略 章节。???通过将红帽构建的 **Keycloak** 服务器设置为使用一次性密码，防止密码猜测。

### 15.4. 只读用户属性

存储在红帽构建的 **Keycloak** 中的典型用户具有与其用户配置集相关的各种属性。此类属性包括 **email**、**firstName** 或 **lastName**。但是，用户也可能具有属性，它们不是典型的配置集数据，而是元数据。元数据属性通常为用户只读，而典型的用户从红帽构建的 **Keycloak** 用户界面或帐户 **REST API** 中更新这些属性。在使用 **Admin REST API** 创建或更新用户时，管理员也应有某些属性甚至只读。

元数据属性通常是来自这些组的属性：

- 与用户存储供应商相关的各种链接或元数据。例如，如果是 **LDAP** 集成，**LDAP_ID** 属性包含 **LDAP** 服务器中用户的 **ID**。
- **User Storage** 置备的元数据。例如，从 **LDAP** 置备的 **createdTimestamp** 应该始终是用户或管理员只读的。
- 与各种验证器相关的元数据。例如，**KERBEROS_PRINCIPAL** 属性可以包含特定用户的 **kerberos** 主体名称。同样的属性 **usercertificate** 可以包含与从 **X.509** 证书绑定数据相关的元数据，这通常在启用 **X.509** 证书验证时使用。
- 应用程序/客户端与用户识别器相关的元数据。例如，**saml.persistent.name.id.for.my_app** 可以包含 **SAML NameID**，供客户端应用程序 **my_app** 用作用户的标识符。
- 与授权策略相关的元数据，这些策略用于基于属性的访问控制(**ABAC**)。这些属性的值可用于授权决策。因此，用户无法更新这些属性非常重要。

从长期的角度来看，红帽构建的 **Keycloak** 将具有正确的用户配置文件 **SPI**，这将允许精细配置每个用户属性。目前，这个能力还没有完全可用。因此，**Red Hat build of Keycloak** 具有用户属性的内部列表，这些属性对于用户是只读的，管理员在服务器级别上配置只读。

这是只读属性的列表，由红帽构建的 **Keycloak** 默认供应商和功能在内部使用，因此始终是只读的：

- 对于用户：

KERBEROS_PRINCIPAL,LDAP_ID,LDAP_ENTRY_DN,CREATED_TIMESTAMP,createTimestamp,modifyTimestamp,userCertificate,saml.persistent.name.id.for <.> , ENABLED,EMAIL_VERIFIED

- 对于管理员：
  KERBEROS_PRINCIPAL,LDAP_ID,LDAP_ENTRY_DN,CREATED_TIMESTAMP,createTimestamp,modifyTimestamp

系统管理员有向此列表添加其他属性的方法。该配置目前位于服务器级别。

您可以使用 spi-user-profile-declarative-user-profile-read-only-attributes 和 'spi-user-profile-declarative-user-profile-admin-read-only-attributes 选项来添加此配置。例如：

```
kc.[sh|bat] start --spi-user-profile-declarative-user-profile-read-only-attributes=foo,bar*
```

在本例中，用户和管理员无法更新属性 foo。用户将无法编辑从 bar 开始的任何属性。例如，bar 或 barrier。配置不区分大小写，因此 FOO 或 BarRier 等属性也会被拒绝。通配符字符 * 仅在属性名称的末尾被支持，因此管理员可以有效地拒绝以指定字符开头的所有属性。属性中中部的 * 被视为普通字符。

### 15.5. 验证用户属性

通过 第 5.2 节 “管理用户属性” 中的功能，管理员可以限制用户输入的属性，例如在用户注册或帐户控制台中输入的数据用户。

管理员不应该允许用户非受管属性，以防止攻击者添加无限数量的属性。属性应该有一个验证，用于限制攻击者输入的数据量。

当使用正则表达式来验证用户属性时，请避免使用过量内存或 CPU 的正则表达式。详情请参阅 OWASP 的正则表达式拒绝服务。

### 15.6. 单击JACKING

单击jacking 是一种欺骗用户的技术，单击与用户不同的用户界面元素。恶意站点将目标站点加载为透明 iFrame，在一组虚拟按钮之上直接放置在目标站点上的重要按钮下。当用户点击可见按钮时，它们将点击隐藏页面上的按钮。攻击者可以利用这个方法窃取用户的身份验证凭据并访问其资源。

默认情况下，红帽构建的 Keycloak 的每个响应都会设置一些特定的 HTTP 标头，这些标头可能会阻止

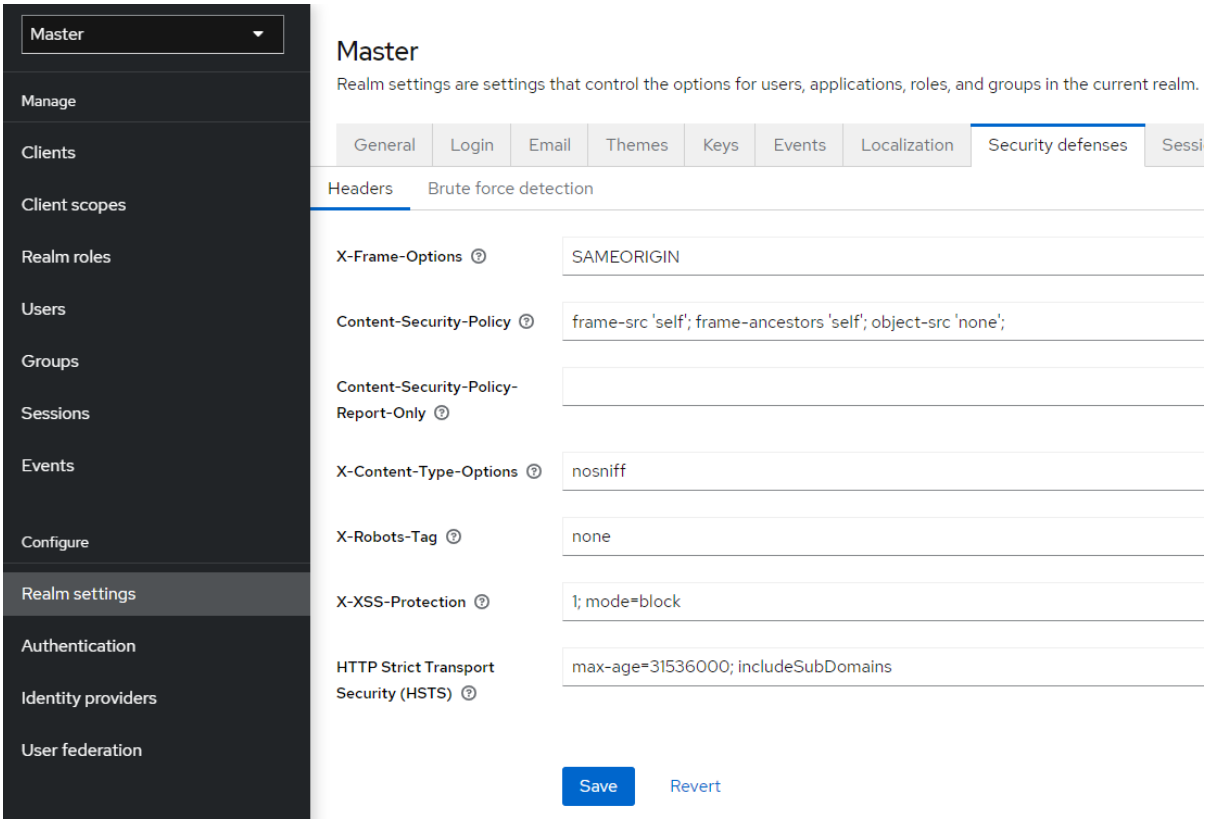
*发生这种情况。具体来说，它设置 **X-Frame-Options** 和 **Content-Security-Policy**。您应该查看这两个标头的定义，因为有许多精细的浏览器访问您可以控制。*

*流程*

*在管理门户中，您可以指定 **X-Frame-Options** 和 **Content-Security-Policy** 标头的值。*

1. *单击 **Realm Settings** 菜单项。*
2. *点 **Security Defenses** 选项卡。*

*安全国防*




*默认情况下，红帽构建的 **Keycloak** 只为 **iframes** 设置相同 的原始策略。*

### *15.7. SSL/HTTPS 要求*

***OAuth 2.0/OpenID Connect** 使用访问令牌来实现安全性。攻击者可以扫描您的网络以获取访问令牌，*

并使用它们执行令牌具有权限的恶意操作。这个攻击被称为中间人攻击。使用 SSL/HTTPS 在红帽构建的 Keycloak 身份验证服务器和红帽构建的 Keycloak 安全客户端之间的通信，以防止中间人攻击。

红帽构建的 Keycloak 有三个 SSL/HTTPS 模式。SSL 非常复杂，因此红帽构建的 Keycloak 允许通过私有 IP 地址（如 localhost、192.168.x.x 和其他专用 IP 地址）进行非 HTTPS 通信。在生产环境中，请确保为所有操作启用 SSL 和 SSL。

在 adapter/client-side 中，您可以禁用 SSL 信任管理器。信任管理器确保红帽构建 Keycloak 与 进行通信的身份是有效的，并确保 DNS 域名与服务器证书相关。在生产环境中，请确保每个客户端适配器都使用信任存储来防止 DNS man-in-the-middle 攻击。

### 15.8. CSRF 攻击

跨站点请求伪造(CSRF)攻击使用 Web 站点已经验证的用户的 HTTP 请求。任何使用基于 Cookie 的身份验证的站点都容易受到 CSRF 攻击。您可以通过匹配状态 Cookie 针对已发布的表单或查询参数来缓解这些攻击。

OAuth 2.0 登录规格要求状态 Cookie 与传输的状态参数匹配。红帽构建的 Keycloak 完全实现此规格的一部分，因此所有登录都受到保护。

Red Hat build of Keycloak Admin Console 是一个 JavaScript/HTML5 应用程序，可发出对红帽构建的 Keycloak 管理 REST API 的 REST 调用。这些调用需要 bearer 令牌身份验证，并由 JavaScript criu 调用组成，因此无法执行 CSRF。您可以配置 admin REST API 以验证 CORS 来源。

红帽构建的 Keycloak 中的帐户控制台可能会受到 CSRF 的攻击。为了防止 CSRF 攻击，红帽构建的 Keycloak 会设置一个状态 cookie，并将这个 Cookie 的值嵌入到隐藏形式字段或操作链接中的查询参数中。红帽构建的 Keycloak 会针对状态 Cookie 检查查询/信息参数，以验证同一用户是否进行了调用。

### 15.9. 不特定的重定向 URI

将您的注册的重定向 URI 尽量具体。注册授权代码流的重定向 URI 可让恶意客户端模拟具有更广泛的访问权限的另一个客户端。???例如，如果两个客户端位于同一域中，则可能会出现模拟。

您可以为域使用安全重定向 uris enforcer executor。结果可确保客户端管理员只能注册符合不同要求的特定重定向机构的客户端，如要求 URL 在上下文路径中具有通配符，或者仅限于指定的允许的域。有关如何使用特定 executor 配置客户端策略的详情，请参阅 客户端策略。???

### 15.10. FAPI 合规性

为确保红帽构建的 Keycloak 服务器将验证您的客户端更安全且符合 FAPI，您可以为 FAPI 支持配置客户端策略。有关保护应用程序和服务 指南的 FAPI 部分的详细信息。此外，这可确保上述一些安全最佳实践，如客户端所需的 SSL、使用安全重定向 URI 以及更多类似最佳实践的安全重定向 URI。

### 15.11. OAUTH 2.1 合规性

为确保红帽构建的 Keycloak 服务器将验证您的客户端更安全且 OAuth 2.1 兼容，您可以为 OAuth 2.1 支持配置客户端策略。如需更多信息，请参阅 保护应用程序和服务指南中的 OAuth 2.1 部分。

### 15.12. 被破坏的访问和刷新令牌

红帽构建的 Keycloak 包括一些操作，以防止恶意参与者窃取访问令牌并刷新令牌。关键操作是在红帽构建的 Keycloak 及其客户端和应用程序之间强制实施 SSL/HTTPS 通信。红帽构建的 Keycloak 默认不启用 SSL。

缓解泄漏访问令牌损坏的另一个操作是缩短令牌的生命周期。您可以在 超时页面中 指定令牌生命周期。访问令牌的短寿命强制客户端和应用程序在短时间内刷新其访问令牌。如果管理员检测到泄漏，管理员可以注销所有用户会话使这些刷新令牌无效或设置撤销策略。

确保刷新令牌始终保持对客户端私有且永远不会被传输。

您可以通过将令牌作为密钥令牌发布，从而缓解泄漏访问令牌和刷新令牌的问题。如需更多信息，请参阅 OAuth 2.0 通用 TLS 客户端证书 绑定访问令牌。

如果访问令牌或刷新令牌被破坏，访问管理控制台，并将 not-before revocation 策略推送到所有应用程序。推送一个 not-before 策略可确保该时间无效之前发出的任何令牌。推送新的 not-before 策略可确保应用程序必须从红帽构建的 Keycloak 下载新的公钥，并从受入侵的域签名密钥缓解损坏。如需更多信息，请参阅 密钥章节。

如果特定应用程序、客户端或用户被破坏，您可以禁用它们。

### 15.13. 被破坏的授权代码

对于 OIDC Auth Code 流，红帽构建的 Keycloak 为其授权代码生成一个强大的随机值。授权代码仅使用一次来获取访问令牌。

在 **Admin Console** 的 超时页面中，您可以指定授权代码有效的时长。确保时间长度小于 **10** 秒，这足够长，以便客户端能够从代码请求令牌。

您还可以通过将代码 交换(PKCE) 应用到客户端来防御泄漏授权代码。

### 15.14. OPEN REDIRECTORS

**Open redirector** 是一个端点，使用参数自动将用户代理重定向到参数值指定的位置，而无需验证。攻击者可以利用最终用户授权端点和重定向 **URI** 参数将授权服务器用作开放重定向器，使用授权服务器中的用户的信任来启动临时攻击。

**Red Hat build of Keycloak** 要求所有注册的应用程序和客户端都至少注册一个重定向 **URI** 模式。当红帽构建的 **Keycloak** 的客户端请求执行重定向时，**Red Hat build of Keycloak** 会根据有效注册 **URI** 模式列表检查重定向 **URI**。客户端和应用程序必须注册为特定的 **URI** 模式，以缓解开放的重定向器攻击。

如果应用程序需要一个非 **http (s)**自定义方案，它应该是验证模式的显式部分（如 **custom:/app8:0:1::**）。为了安全起见，常规模式（如 * ）并不涵盖非 **http**。

通过使用 客户端策略，管理员可以确保客户端无法注册打开的重定向 **URL**，如 *。

### 15.15. 密码数据库被破坏

红帽构建的 **Keycloak** 不会将密码存储在原始文本中，而是使用 **PBKDF2-HMAC-SHA512** 消息摘要算法作为哈希文本。红帽构建的 **Keycloak** 执行 **210,000** 哈希迭代，这是安全社区推荐的迭代数。这个哈希迭代数量可能会对性能造成负面影响，因为 **PBKDF2** 哈希使用了大量 **CPU** 资源。

### 15.16. 限制范围

默认情况下，新客户端应用程序具有无限 角色范围映射。该客户端的每个访问令牌都包含用户具有的所有权限。如果攻击者破坏客户端并获取客户端的访问令牌，则用户可访问的每个系统都会受到破坏。

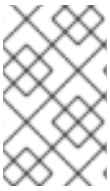
使用每个客户端的 **Scope** 菜单 限制访问令牌的角色。另外，您可以在 **Client Scope** 级别上设置角色范围映射，并使用 **Client Scope** 菜单将 **Client Scopes** 分配给客户端。

### 15.17. 限制令牌受众

在服务间具有低级别的信任的环境中，限制了令牌的受众。如需更多信息，请参阅 OAuth2 Threat Model 和 Audience Support 部分。

### 15.18. 限制身份验证会话

当在网页浏览器中第一次打开登录页面时，红帽构建的 **Keycloak** 会创建一个名为身份验证会话的对象，该对象存储了有关该请求的一些有用信息。每当从同一浏览器中的不同标签页打开新的登录页面时，**Red Hat build of Keycloak** 会创建一个名为 **authentication** 子会话的新记录，该记录存储在身份验证会话中。身份验证请求可能来自任何类型的客户端，如 **Admin CLI**。在这种情况下，也会创建一个新的身份验证会话，并带有一个身份验证子会话。请注意，除使用浏览器流外，还可以以其他方式创建身份验证会话。无论源流是什么，以下文本都适用。



注意

本节论述了使用 **Data Grid** 供应商进行身份验证会话的部署。

身份验证会话内部存储为 **RootAuthenticationSessionEntity**。每个 **RootAuthenticationSessionEntity** 可以有多个身份验证子会话存储在 **RootAuthenticationSessionEntity** 中，作为 **AuthenticationSessionEntity** 对象的集合。红帽构建的 **Keycloak** 将身份验证会话存储在专用 **Data Grid** 缓存中。每个 **Root AuthenticationSessionEntity** 的 **AuthenticationSessionEntity** 数量为每个缓存条目的大小贡献。身份验证会话缓存的内存占用总量由存储的 **RootAuthenticationSessionEntity** 数以及每个 **Root AuthenticationSessionEntity** 中的 **AuthenticationSessionEntity** 的数量决定。

维护的 **RootAuthenticationSessionEntity** 对象的数量对应于浏览器中未完成登录流的数量。要将 **RootAuthenticationSessionEntity** 的数量保持在控制下，建议使用高级防火墙控制来限制入口网络流量。

对于有很多活跃 **Root AuthenticationSessionEntity** 且有很多身份验证会话的部署，可能会进行更高的内存用量。如果负载均衡器不支持或没有为会话粘性配置，则集群中网络的负载可能会显著提高。此负载的原因是，位于没有适当身份验证会话的节点上的每个请求都需要检索和更新所有者节点中的身份验证会话记录，这涉及检索和存储的独立网络传输。

可以通过设置 **authSessionsLimit** 属性在 **authenticationSessions SPI** 中配置每个 **Root AuthenticationSessionEntity** 的最大 **AuthenticationSessionEntity** 数量。默认值为每个 **Root AuthenticationSessionEntity** 设置为 **300 AuthenticationSessionEntity**。当达到这个限制时，最旧的身份验证子会话将在新的身份验证会话请求后删除。

以下示例演示了如何将每个 **Root AuthenticationSessionEntity** 数量限制为 **100**。

```
bin/kc.[sh|bat] start --spi-authentication-sessions-infinispan-auth-sessions-limit=100
```

### 15.19. SQL 注入攻击

目前，红帽构建的 **Keycloak** 没有已知的 **SQL** 注入漏洞。

# 第 16 章 帐户控制台

红帽构建的 **Keycloak** 用户可以通过帐户控制台管理其帐户。它们可以配置其配置文件，添加双因素身份验证，包括身份提供程序帐户，以及查看设备活动。

其他资源

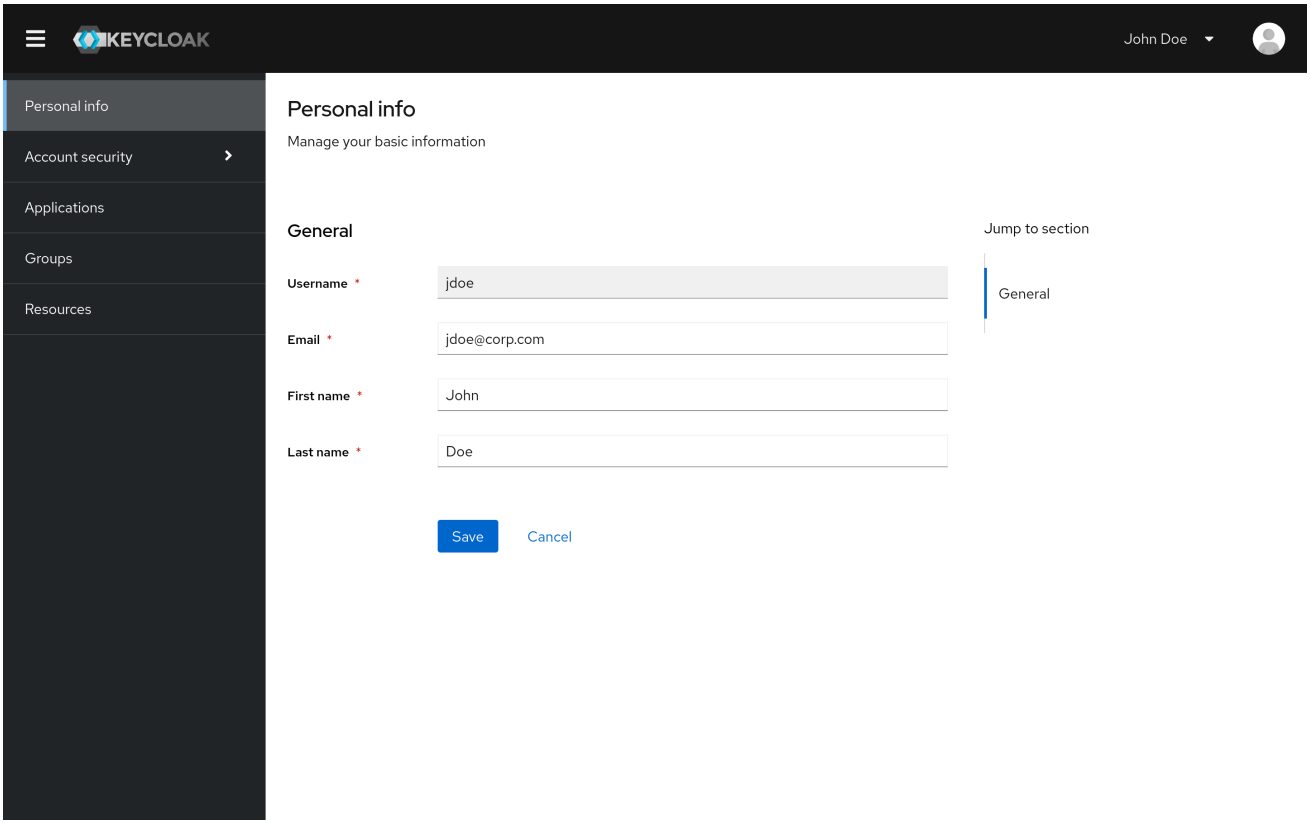
- 帐户控制台可以在外观和语言首选项方面进行配置。例如，在 **Personal info** 页面中添加额外的属性。如需更多信息，请参阅 服务器开发人员指南。

## 16.1. 访问帐户控制台

流程

1. 记录下您帐户存在的红帽构建的 **Keycloak** 服务器的域名和 **IP** 地址。
2. 在 **Web** 浏览器中，输入以下格式的 **URL**： **server-root/realms/{realm-name}/account**。
3. 输入您的登录名和密码。

帐户控制台




## *16.2. 配置登录方式*

*您可以使用基本身份验证（用户名和密码）或双因素身份验证来登录此控制台。对于双因素身份验证，请使用以下流程之一。*

### *16.2.1. 使用 OTP 进行双因素身份验证*

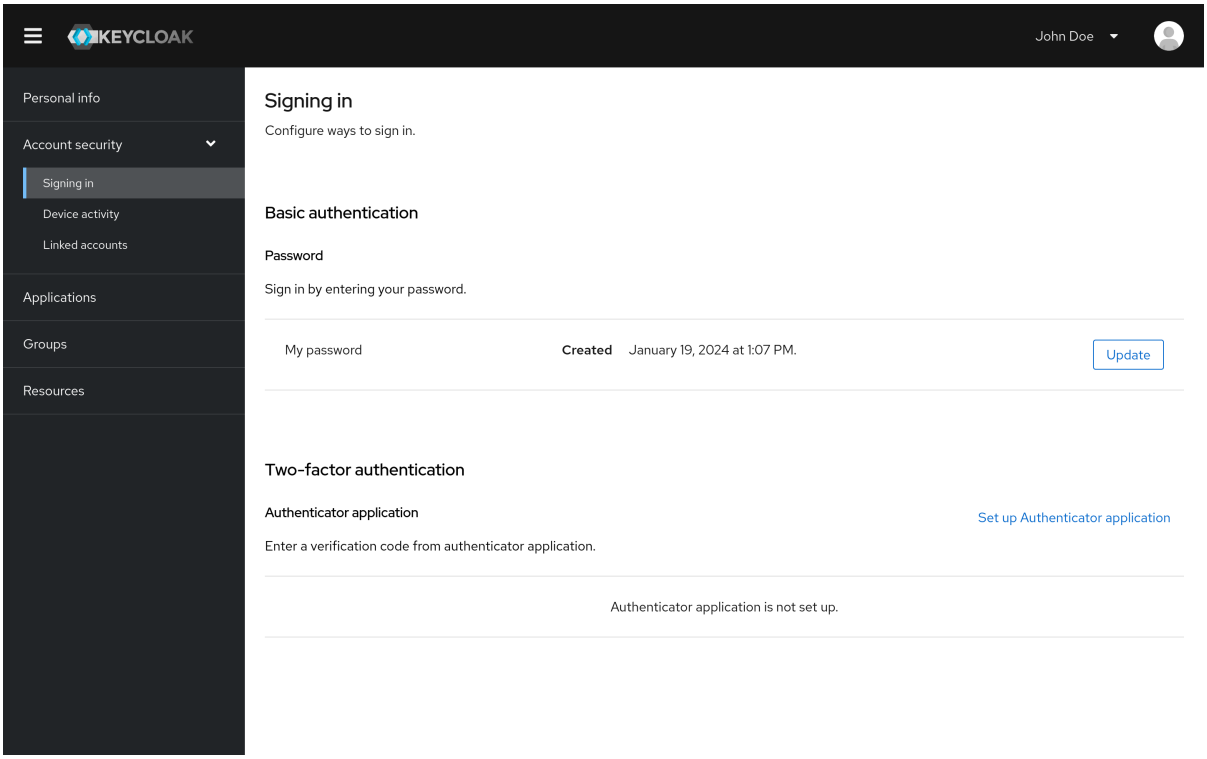
*先决条件*

- ***OTP** 是您的域的有效身份验证机制。*

*流程*

1. *点菜单中的 **Account security**。*
2. *在 中单击 **Signing**。*
3. *单击 **Set up Authenticator** 应用。*

*登陆*




4. *按照屏幕上出现的指示，将您的移动设备用作您的 **OTP** 生成器。*

5. *将屏幕截图中的 **QR** 代码扫描到您的移动设备上的 **OTP** 生成器中。*

6. *注销，然后再次登录。*

7. *输入在移动设备中提供的 **OTP** 来响应提示。*

### *16.2.2. 使用 WebAuthn 进行双因素身份验证*

*先决条件*

- ***Webauthn** 是您的域的有效双因素身份验证机制。详情请查看 **WebAuthn** 部分。*

*流程*

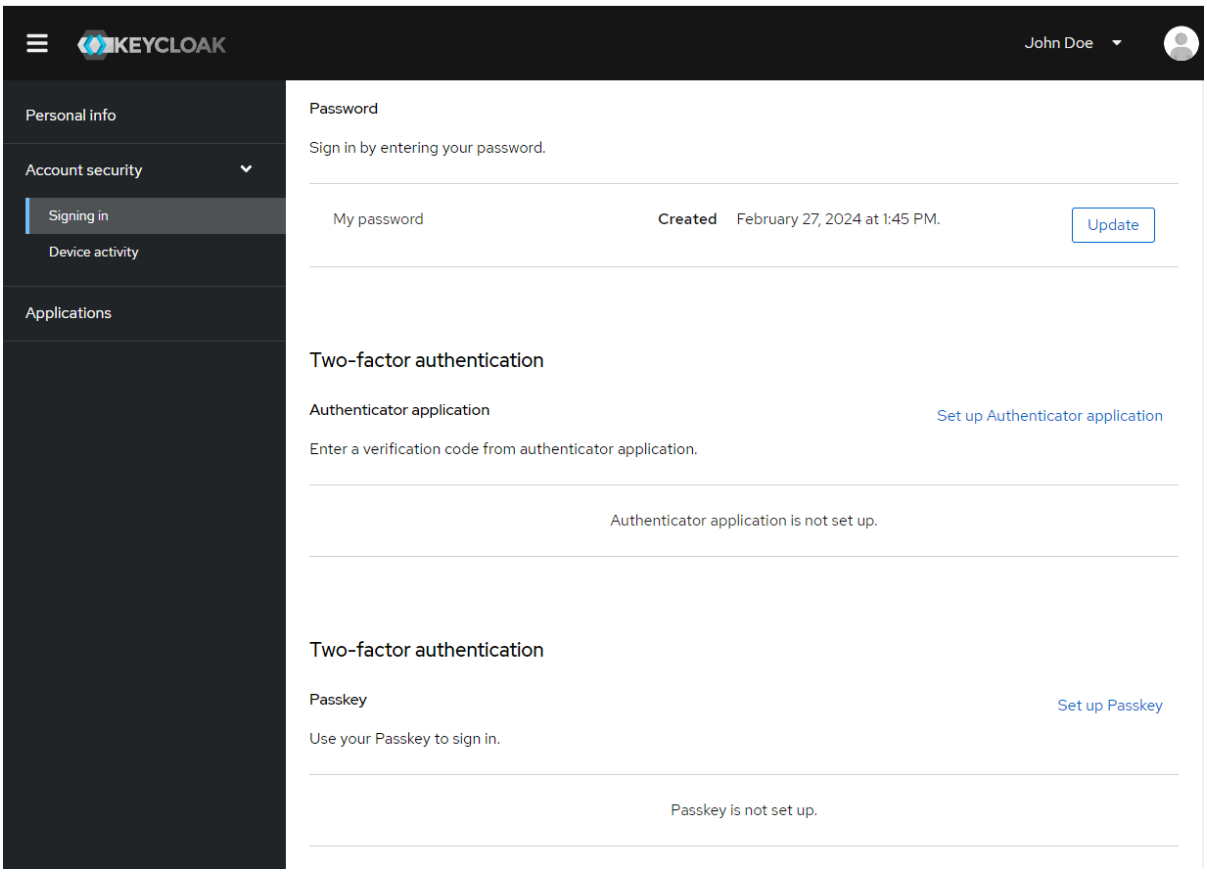
1.

1. *在菜单中点* ***Account Security****。*

2. *单击* ***Signing In****。*

3. *点* ***Set up a Passkey****。*

*签发登录*




4. *准备您的* ***Passkey****。如何准备此密钥取决于您使用的* ***Passkey*** *类型。例如，对于基于* ***USB*** *的* ***Yubikey****，您可能需要将密钥放在笔记本电脑上的* ***USB*** *端口中。*

5. *点* ***Register*** *注册您的* ***Passkey****。*

6. *注销，然后再次登录。*

7.

*假设正确设置了身份验证流，则会出现一条消息，要求您使用您的 **Passkey** 进行身份验证作为第二个因素。*

### *16.2.3. 使用 WebAuthn 进行免密码身份验证*

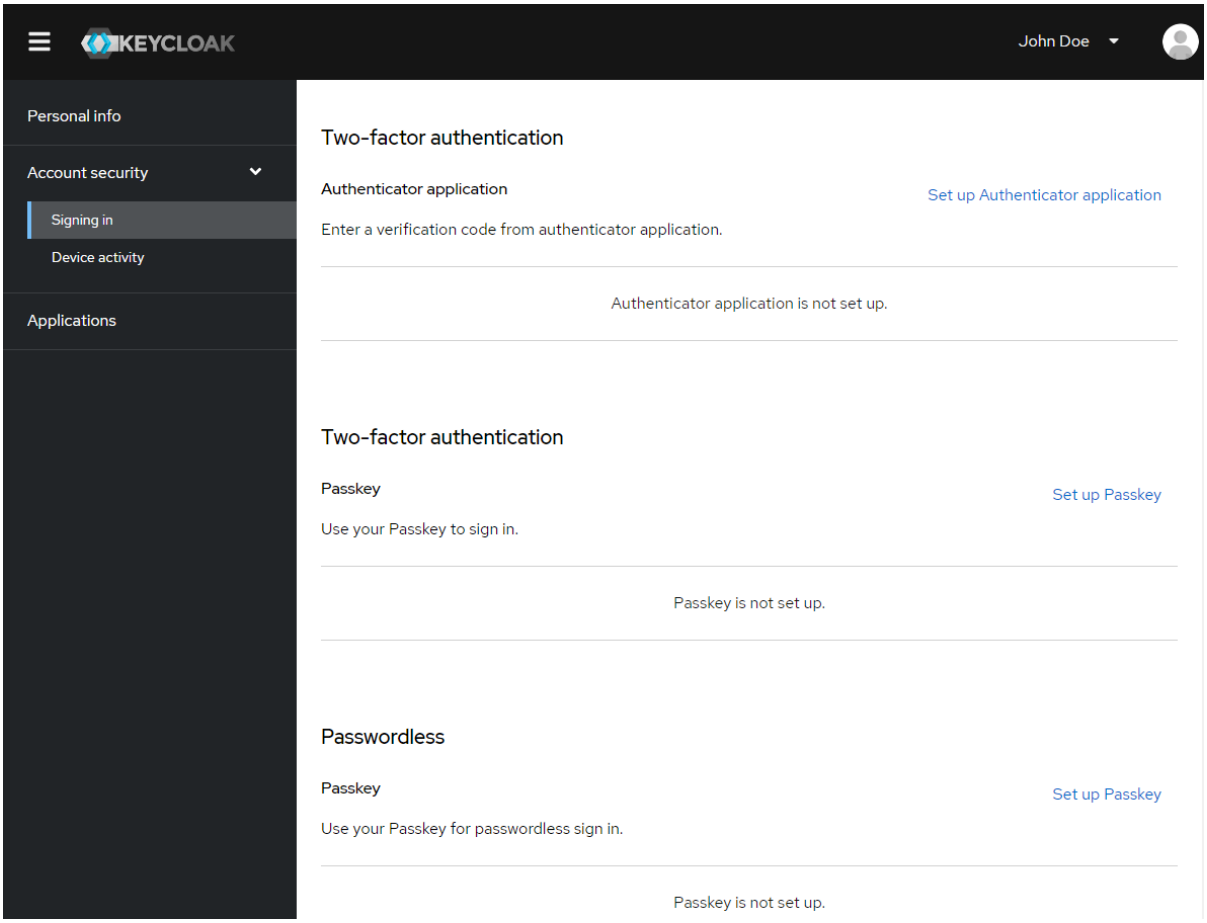
*先决条件*

- ***Webauthn** 是您的域的有效免密码身份验证机制。详情请查看 **Passwordless WebAuthn** 部分。*

*流程*

1. *在菜单中点 **Account Security**。*
2. *单击 **Signing In**。*
3. *在 **Passwordless** 部分中，点 **Set up a Passkey**。*

*签发登录*




4. *准备您的 **Passkey**。如何准备此密钥取决于您使用的 **Passkey** 类型。例如，对于基于 **USB** 的 **Yubikey**，您可能需要将密钥放在笔记本电脑上的 **USB** 端口中。*

5. *点 **Register** 注册您的 **Passkey**。*

6. *注销，然后再次登录。*

7. *假设正确设置了身份验证流，则会出现一条消息，要求您使用您的 **Passkey** 进行身份验证作为第二个因素。您不再需要提供您的密码才能登录。*

## *16.3. 查看设备活动*

*您可以查看登录到您的帐户的设备。*

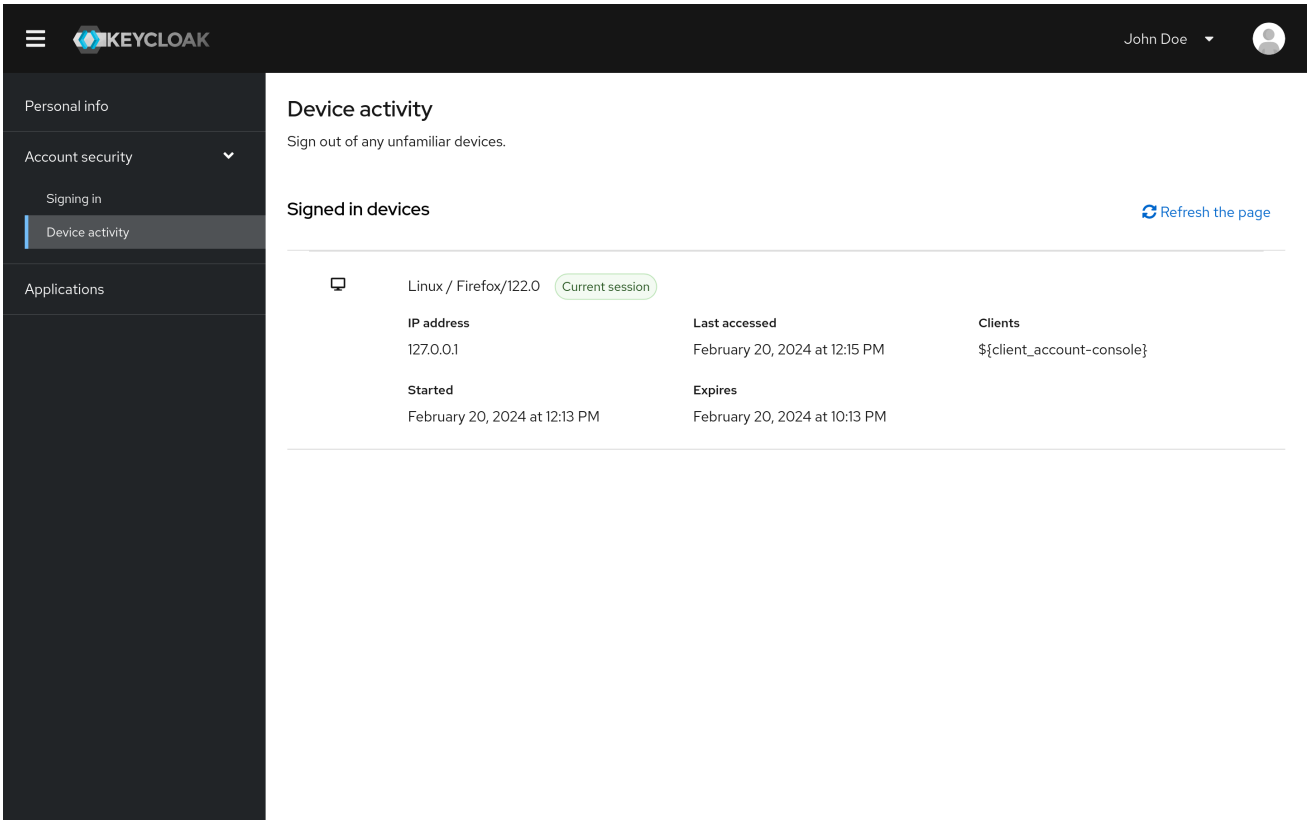
*流程*

1. *点菜单中的* ***Account security****。*

2. *点* ***Device activity****。*

3. *如果设备看起来可疑，请注销。*

***Devices***




### *16.4. 添加身份提供程序帐户*

*您可以使用身份代理链接您的帐户。* ***???****这个选项通常用于链接社交供应商帐户。*

*流程*

1. *登录* ***Admin*** *控制台。*

2. *单击菜单中的* ***Identity provider****。*

3. *选择一个供应商并填写字段。*

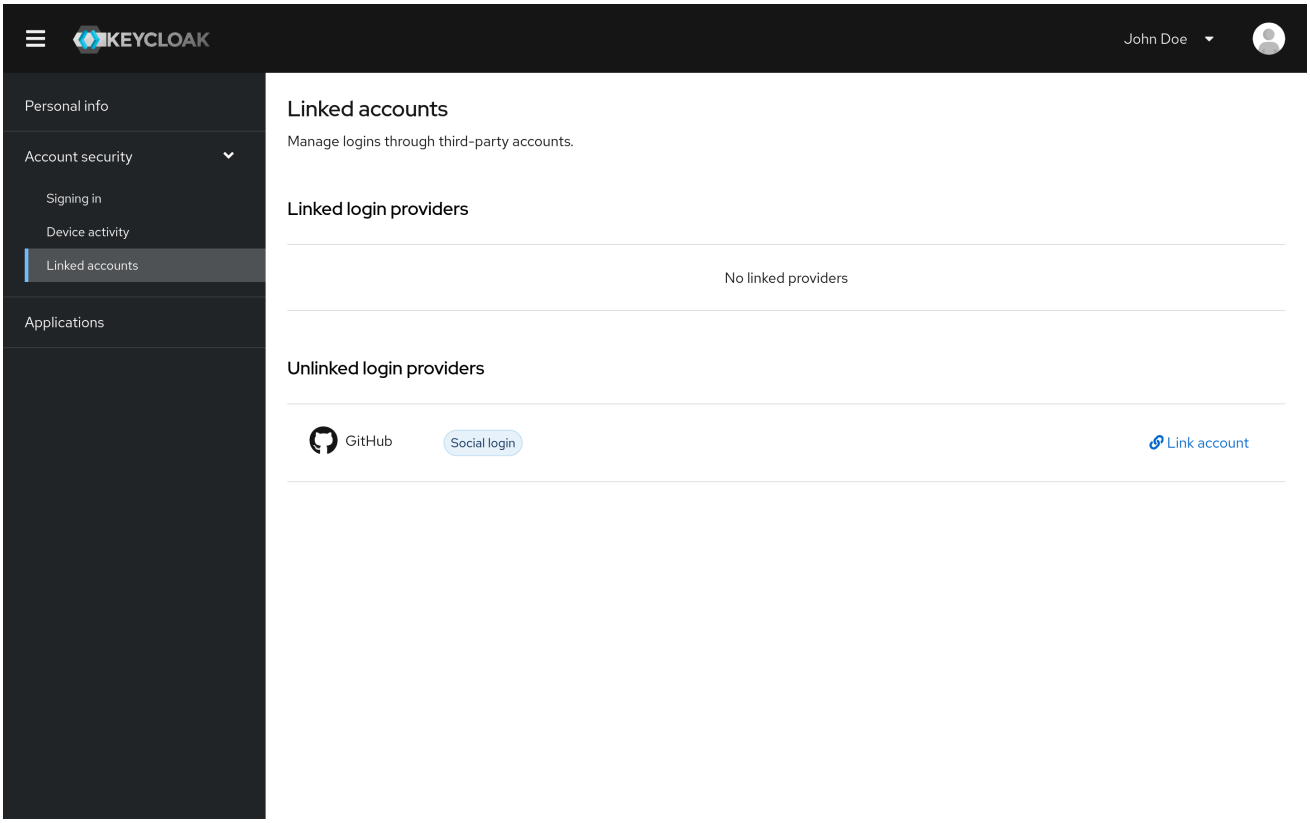
4. *返回到帐户控制台。*

5. *点菜单中的* ***Account security****。*

6. *单击* ***Linked accounts****。*

*您添加的身份提供程序会出现在此页面中。*

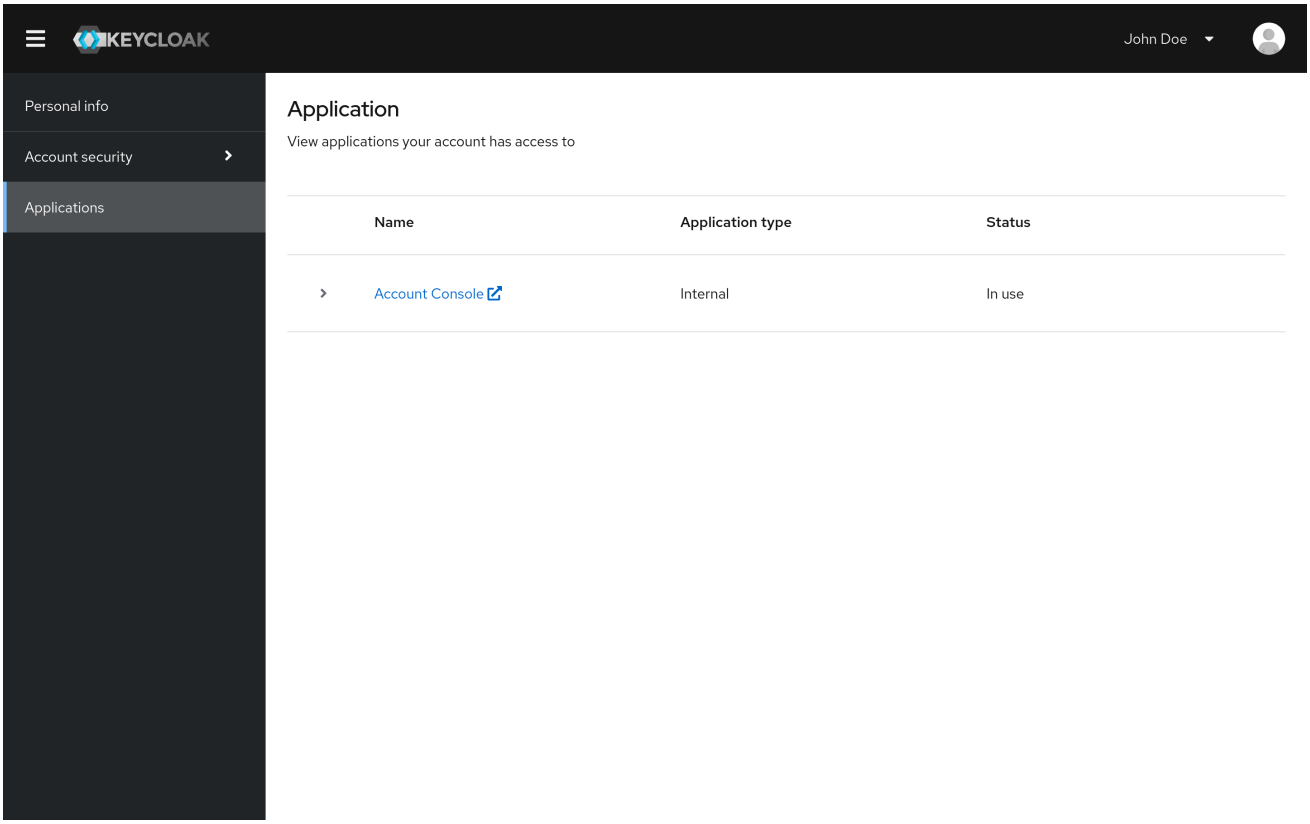
*链接的帐户*




## *16.5. 访问其他应用程序*

*Applications 菜单项显示您可以访问的应用程序的用户。在这种情况下，只有帐户控制台可用。*

*应用程序*



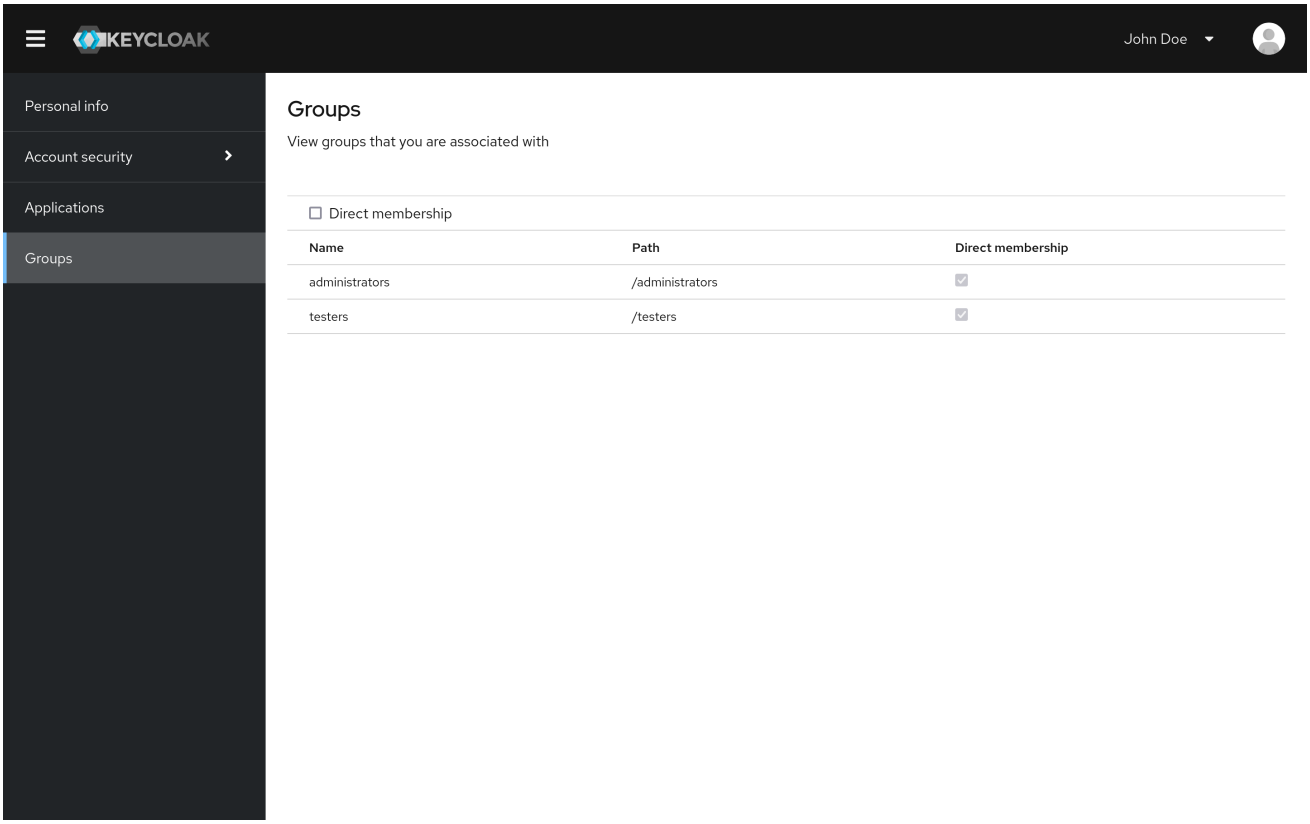

### *16.6. 查看组成员资格*

*您可以通过点 Groups 菜单来查看您关联的组。如果您选择 Direct membership 复选框，则只会看到您直接关联的组。*

*先决条件*

- *您需要具有 view-groups 帐户角色，才能查看 Groups 菜单。*

*查看组成员资格*


KEYCLOAK
John Doe
Personal info
Account security
Applications
Groups
Groups
View groups that you are associated with
Direct membership
Name
Path
Direct membership
administrators
/administrators
testers
/testers

# 第 17 章 管理 CLI

使用红帽构建的 **Keycloak**，您可以使用 **Admin CLI** 命令行工具从命令行界面**(CLI)**执行管理任务。

## 17.1. 安装 ADMIN CLI

**Red Hat build of Keycloak** 使用 **bin** 目录中的执行脚本来打包 **Admin CLI** 服务器分发。

**Linux** 脚本名为 **kcadm.sh**，**Windows** 的脚本称为 **kcadm.bat**。将红帽构建的 **Keycloak** 服务器目录添加到 **PATH** 中，以使用文件系统的任意位置的客户端。

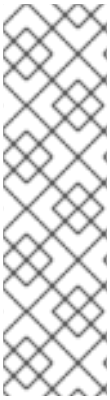
例如：

- **linux:**

```
$ export PATH=$PATH:$KEYCLOAK_HOME/bin
$ kcadm.sh
```

- **Windows**：

```
c:\> set PATH=%PATH%;%KEYCLOAK_HOME%\bin
c:\> kcadm
```



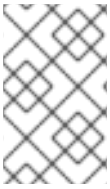
注意

您必须将 **KEYCLOAK_HOME** 环境变量设置为提取红帽构建的 **Keycloak** 服务器分发的路径。

为避免重复，本文档的剩余部分只使用 **Windows** 示例，其中 **CLI** 差别不仅限于 **kcadm** 命令名称。

## 17.2. 使用 ADMIN CLI

**Admin CLI** 向 **Admin REST** 端点发出 **HTTP** 请求。访问 **Admin REST** 端点需要身份验证。



*注意*

*有关特定端点的 **JSON** 属性的详细信息，请参阅 **Admin REST API** 文档。*

1. *通过登录来启动经过身份验证的会话。现在，可以执行创建、读取、更新和删除**(CRUD)**操作。*

   *例如：*

   - ***linux:***

     ```
     $ kcadm.sh config credentials --server http://localhost:8080 --realm demo --user admin --
     client admin
     $ kcadm.sh create realms -s realm=demorealm -s enabled=true -o
     $ CID=$(kcadm.sh create clients -r demorealm -s clientId=my_client -s 'redirectUris=
     ["http://localhost:8980/myapp/*"]' -i)
     $ kcadm.sh get clients/$CID/installation/providers/keycloak-oidc-keycloak-json
     ```

   - ***Windows：***

     ```
     c:\> kcadm config credentials --server http://localhost:8080 --realm demo --user admin --
     client admin
     c:\> kcadm create realms -s realm=demorealm -s enabled=true -o
     c:\> kcadm create clients -r demorealm -s clientId=my_client -s "redirectUris=
     [\"http://localhost:8980/myapp/*\"]" -i > clientid.txt
     c:\> set /p CID=<clientid.txt
     c:\> kcadm get clients/%CID%/installation/providers/keycloak-oidc-keycloak-json
     ```

2. *在生产环境中，使用 **https:** 访问 **Keycloak** 的红帽构建，以避免公开令牌。如果可信证书颁发机构（包含在 **Java** 的默认证书信任存储中）没有发布服务器证书，请准备 **truststore.jks** 文件并指示 **Admin CLI** 使用它。*

   *例如：*

   - ***linux:***

     ```
     $ kcadm.sh config truststore --trustpass $PASSWORD ~/.keycloak/truststore.jks
     ```

- ***Windows***：

```
c:\> kcadm config truststore --trustpass %PASSWORD%
%HOMEPATH%\.keycloak\truststore.jks
```

### *17.3. 身份验证*

*使用 **Admin CLI** 登录时，您可以指定：*

- *服务器端点 **URL***
- *一个 **realm***
- *用户名*

*另一个选项是只指定 **clientId**，它会为您创建唯一服务帐户。*

*使用用户名登录时，使用指定用户的密码。当使用 **clientId** 登录时，您只需要客户端 **secret**，而不是用户密码。您还可以使用 **Signed JWT** 而不是客户端 **secret**。*

*确保用于会话的帐户具有调用 **Admin REST API** 操作的正确权限。例如，**realm-management** 客户端的 **realm-admin** 角色可以管理用户的域。*

*有两种主要机制可用于身份验证。一个机制使用 **kcadm config** 凭证 来启动经过身份验证的会话。*

```
$ kcadm.sh config credentials --server http://localhost:8080 --realm master --user admin --password
admin
```

*此机制通过保存获取的访问令牌及其关联的刷新令牌，在 **kcadm** 命令调用之间维护经过身份验证的会话。它可以在私有配置文件中维护其他 **secret**。如需更多信息，请参见 下一章节。*

*第二种机制在调用期间验证每个命令调用。这种机制会增加服务器上的负载，以及往返获取令牌所需的时间。这种方法的好处在于，在调用之间不需要保存令牌，因此不会将令牌保存到磁盘。当指定 **--no-***

**config** 参数时，**Red Hat build of Keycloak** 会使用这个模式。

例如，在执行操作时，指定身份验证所需的所有信息。

```
$ kcadm.sh get realms --no-config --server http://localhost:8080 --realm master --user admin --password admin
```

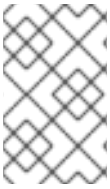
运行 **kcadm.sh help** 命令以获取关于使用 **Admin CLI** 的更多信息。

运行 **kcadm.sh config credentials --help** 命令以了解有关启动经过身份验证的会话的更多信息。

### 17.4. 使用其他配置

默认情况下，**Admin CLI** 维护名为 **kcadm.config** 的配置文件。红帽构建的 **Keycloak** 将此文件放在用户的主目录中。在基于 **Linux** 的系统中，完整路径名称为 **$HOME/.keycloak/kcadm.config**。在 **Windows** 中，完整路径名称为 **%HOMEPATH%\.keycloak\kcadm.config**。

您可以使用 **--config** 选项指向不同的文件或位置，以便您可以并行维护多个经过身份验证的会话。



注意

从单个线程执行与单个配置文件关联的操作。

确保配置文件对系统上的其他用户不可见。它包含必须私有的访问令牌和 **secret**。**Red Hat build of Keycloak** 会自动创建 **~/.keycloak** 目录及其内容，并带有正确的访问限制。如果目录已存在，红帽构建的 **Keycloak** 不会更新目录的权限。

可以避免将 **secret** 存储在配置文件中，但这样做不方便，并增加令牌请求的数量。将 **--no-config** 选项与所有命令搭配使用，并指定配置 凭证命令所需的 身份验证信息，并在每次调用 **kcadm** 时使用。

### 17.5. 基本操作和资源 URI

**Admin CLI** 通过简化特定任务的额外命令通常对 **Admin REST API** 端点执行 **CRUD** 操作。

此处列出了主要的使用模式：

```
$ kcadm.sh create ENDPOINT [ARGUMENTS]
$ kcadm.sh get ENDPOINT [ARGUMENTS]
$ kcadm.sh update ENDPOINT [ARGUMENTS]
$ kcadm.sh delete ENDPOINT [ARGUMENTS]
```

**create**、**get**、**update** 和 **delete** 命令分别映射到 **HTTP** 动词 **POST**、**GET**、**PUT** 和 **DELETE**。**ENDPOINT** 是一个目标资源 **URI**，可以绝对（从 **http:** 或 **https:**开始）或相对，红帽构建的 **Keycloak** 使用它来以以下格式编写绝对 **URL**：

```
SERVER_URI/admin/realms/REALM/ENDPOINT
```

例如：如果您对服务器 **http://localhost:8080** 进行身份验证且 **realm** 是 **master**，使用用户 **ENDPOINT** 会创建 **http://localhost:8080/admin/realms/master/users** 资源 **URL**。

如果将 **ENDPOINT** 设置为 客户端，则有效的资源 **URI** 为 **http://localhost:8080/admin/realms/master/clients**。

红帽构建的 **Keycloak** 有一个 域 端点，它是域的容器。它解析为：

```
SERVER_URI/admin/realms
```

红帽构建的 **Keycloak** 具有 **serverinfo** 端点。此端点独立于域。

当您以具有 **realm-admin** 电源的用户身份进行身份验证时，您可能需要对多个域执行命令。如果是，请指定 **-r** 选项，以告知 **CLI** 命令要明确针对其执行的域。命令不使用由 **kcadm.sh** 配置凭据 指定的 **--realm** 选项指定的 **REALM**，该命令使用 **TARGET_REALM**。

```
SERVER_URI/admin/realms/TARGET_REALM/ENDPOINT
```

例如：

```
$ kcadm.sh config credentials --server http://localhost:8080 --realm master --user admin --password
admin
$ kcadm.sh create users -s username=testuser -s enabled=true -r demorealm
```

在本例中，您将启动一个会话，作为 **master** 域中的 **admin** 用户进行身份验证。然后，您将针对资源 **URL** http://localhost:8080/admin/realms/demorealm/users 执行 **POST** 调用。

**create** 和 **update** 命令将 **JSON** 正文发送到服务器。您可以使用 **-f FILENAME** 从文件中读取预先可用的文档。当使用 **-f -** 选项时，红帽构建的 **Keycloak** 从标准输入中读取消息正文。您可以指定单独的属性及其值，如 创建用户示例中所示。红帽构建的 **Keycloak** 将属性组成 **JSON** 正文，并将它们发送到服务器。

红帽构建的 **Keycloak** 中提供了几种方法，以使用 **update** 命令更新资源。您可以确定资源的当前状态，并将其保存到文件中，编辑该文件并将其发送到服务器以进行更新。

例如：

```
$ kcadm.sh get realms/demorealm > demorealm.json
$ vi demorealm.json
$ kcadm.sh update realms/demorealm -f demorealm.json
```

此方法使用发送 **JSON** 文档中的属性更新服务器上的资源。

另一种方法是使用 **-s, --set** 选项来设置新值来执行实时更新。

例如：

```
$ kcadm.sh update realms/demorealm -s enabled=false
```

此方法将 **enabled** 属性设置为 **false**。

默认情况下，**update** 命令会执行 **get**，然后将新属性值与现有的值合并。在某些情况下，端点可能支持 **put** 命令，但不支持 **get** 命令。您可以使用 **-n** 选项执行 **no-merge** 更新，它会在不首先运行 **get** 命令的情况下执行 **put** 命令。

### 17.6. REALM 操作

创建新域

在 **realm** 端点上使用 **create** 命令来创建新的启用的域。将属性设置为 **realm** 并启用。

```
$ kcadm.sh create realms -s realm=demorealm -s enabled=true
```

红帽构建的 **Keycloak** 默认禁用域。您可以通过启用它来立即使用 **realm** 进行身份验证。

新对象的描述也可以是 **JSON** 格式。

```
$ kcadm.sh create realms -f demorealm.json
```

您可以直接向文件发送带有 **realm** 属性的 **JSON** 文档，或者将文档传送到标准输入。

例如：

- **linux:**

```
$ kcadm.sh create realms -f - << EOF
{ "realm": "demorealm", "enabled": true }
EOF
```

- **Windows**：

```
c:\> echo { "realm": "demorealm", "enabled": true } | kcadm create realms -f -
```

列出现有域

此命令返回所有域的列表。

```
$ kcadm.sh get realms
```

注意

红帽构建的 **Keycloak** 会过滤服务器上的域列表，以返回用户只能看到的域。

所有 **realm** 属性的列表都非常详细，大多数用户都对属性的子集感兴趣，如 **realm name** 和 **enabled realm** 的状态。您可以使用 **--fields** 选项指定要返回的属性。

```
$ kcadm.sh get realms --fields realm,enabled
```

-

您可以使用逗号分隔的值显示结果。

```
$ kcadm.sh get realms --fields realm --format csv --noquotes
```

获取特定域

将 **realm** 名称附加到集合 **URI**，以获取单个域。

```
$ kcadm.sh get realms/master
```

更新域

1. 当您不想更改所有域属性时，请使用 **-s** 选项为属性设置新值。

   例如：

   ```
   $ kcadm.sh update realms/demorealm -s enabled=false
   ```

2. 如果要将所有可写属性设置为新值：

   a. 运行 **get** 命令。

   b. 编辑 **JSON** 文件中的当前值。

   c. 重新提交。

   例如：

   ```
   $ kcadm.sh get realms/demorealm > demorealm.json
   $ vi demorealm.json
   $ kcadm.sh update realms/demorealm -f demorealm.json
   ```

删除域

运行以下命令以删除域：

```
$ kcadm.sh delete realms/demorealm
```

*打开域的所有登录页面选项*

*将控制特定功能的属性设置为 **true**。*

*例如：*

```
$ kcadm.sh update realms/demorealm -s registrationAllowed=true -s
registrationEmailAsUsername=true -s rememberMe=true -s verifyEmail=true -s
resetPasswordAllowed=true -s editUsernameAllowed=true
```

*列出 **realm** 密钥*

*对目标域的 **key** 端点使用 **get** 操作。*

```
$ kcadm.sh get keys -r demorealm
```

*生成新的 **realm** 密钥*

1. *在添加新的 **RSA** 生成的密钥对前，获取目标域的 **ID**。*

   *例如：*

   ```
   $ kcadm.sh get realms/demorealm --fields id --format csv --noquotes
   ```

2. *添加一个优先级高于现有提供程序的新密钥提供程序，如 **kcadm.sh get keys -r demorealm** 所示。*

   *例如：*

   - ***linux:***

     ```
     $ kcadm.sh create components -r demorealm -s name=rsa-generated -s providerId=rsa-
     generated -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-
     41d7-8359-6aa527fca0b0 -s 'config.priority=["101"]' -s 'config.enabled=["true"]' -s
     'config.active=["true"]' -s 'config.keySize=["2048"]'
     ```

   - ***Windows***：

     ```
     c:\> kcadm create components -r demorealm -s name=rsa-generated -s providerId=rsa-
     generated -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-
     ```

```
41d7-8359-6aa527fca0b0 -s "config.priority=[\"101\"]" -s "config.enabled=[\"true\"]" -s
"config.active=[\"true\"]" -s "config.keySize=[\"2048\"]"
```

3. 将 **parentId** 属性设置为目标域 **ID** 的值。

新添加的密钥现在是活跃的密钥，如 **kcadm.sh get keys -r demorealm** 所示。

从 **Java Key Store** 文件添加新的域密钥

1. 添加新密钥提供程序，以作为 **JKS** 文件预先准备添加新密钥对。

例如，在：

- **linux:**

```
$ kcadm.sh create components -r demorealm -s name=java-keystore -s providerId=java-
keystore -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-
41d7-8359-6aa527fca0b0 -s 'config.priority=["101"]' -s 'config.enabled=["true"]' -s
'config.active=["true"]' -s 'config.keystore=["/opt/keycloak/keystore.jks"]' -s
'config.keystorePassword=["secret"]' -s 'config.keyPassword=["secret"]' -s
'config.keyAlias=["localhost"]'
```

- **Windows**：

```
c:\> kcadm create components -r demorealm -s name=java-keystore -s providerId=java-
keystore -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-
41d7-8359-6aa527fca0b0 -s "config.priority=[\"101\"]" -s "config.enabled=[\"true\"]" -s
"config.active=[\"true\"]" -s "config.keystore=[\"/opt/keycloak/keystore.jks\"]" -s
"config.keystorePassword=[\"secret\"]" -s "config.keyPassword=[\"secret\"]" -s
"config.keyAlias=[\"localhost\"]"
```

2. 确保更改 **keystore**、**storePassword**、**keyPassword** 和 **alias** 的属性值，以匹配您的特定密钥存储。

3. 将 **parentId** 属性设置为目标域 **ID** 的值。

使密钥被动或禁用密钥

1. 识别您要进行被动的密钥。

-

```
$ kcadm.sh get keys -r demorealm
```

2. 使用密钥的 **providerId** 属性来构造端点 **URI**，如 **components/PROVIDER_ID**。

3. 执行 更新。

   例如：

   - **linux:**

     ```
     $ kcadm.sh update components/PROVIDER_ID -r demorealm -s 'config.active=["false"]'
     ```

   - **Windows**：

     ```
     c:\> kcadm update components/PROVIDER_ID -r demorealm -s "config.active=[\"false\"]"
     ```

     您可以更新其他关键属性：

   - 设置一个新的 **enabled** 值来禁用键，如 **config.enabled=["false "]**。

   - 设置一个新的 优先级值 以更改键的优先级，如 **config.priority=["110 "]**。

删除旧密钥

1. 确保您要删除的密钥不活跃，并已禁用它。此操作是防止应用程序和用户持有的现有令牌失败。

2. 识别要删除的密钥。

   ```
   $ kcadm.sh get keys -r demorealm
   ```

3. 使用密钥的 **providerId** 来执行删除。

   ```
   $ kcadm.sh delete components/PROVIDER_ID -r demorealm
   ```

–

*为域配置事件日志记录*

*对 **events/config** 端点使用 **update** 命令。*

***eventListener s** 属性包含 **EventListenerProviderFactory ID** 列表，指定接收事件的所有事件监听程序。可以使用属性来控制内置事件存储，因此您可以使用 **Admin REST API** 查询过去的事件。红帽构建的 **Keycloak** 对服务调用**(eventsEnabled)**和 **Admin REST API (adminEventsEnabled)**触发的审计事件有单独的控制。您可以将 **eventsExpiration** 事件设置为过期，以防止数据库填满。**Red Hat build of Keycloak** 将 **eventExpiration** 设置为 **time-to-live**，以秒为单位。*

*您可以设置内置事件监听程序，通过 **JBoss-logging** 接收所有事件，并记录事件。通过使用 **org.keycloak.events** 日志记录器，红帽构建的 **Keycloak** 会记录错误事件，作为 **WARN** 和其他事件作为 **DEBUG**。*

*例如：*

- ***linux:***

```
$ kcadm.sh update events/config -r demorealm -s 'eventsListeners=["jboss-logging"]'
```

- ***Windows**：*

```
c:\> kcadm update events/config -r demorealm -s "eventsListeners=[\"jboss-logging\"]"
```

*例如：*

*您可以为所有可用 **ERROR** 事件（不包括审计事件）打开存储，以便您可以通过 **Admin REST** 检索事件。*

- ***linux:***

```
$ kcadm.sh update events/config -r demorealm -s eventsEnabled=true -s
'enabledEventTypes=
["LOGIN_ERROR","REGISTER_ERROR","LOGOUT_ERROR","CODE_TO_TOKEN_ERRO
R","CLIENT_LOGIN_ERROR","FEDERATED_IDENTITY_LINK_ERROR","REMOVE_FEDE
RATED_IDENTITY_ERROR","UPDATE_EMAIL_ERROR","UPDATE_PROFILE_ERROR","U
PDATE_PASSWORD_ERROR","UPDATE_TOTP_ERROR","VERIFY_EMAIL_ERROR","RE
```

```
MOVE_TOTP_ERROR","SEND_VERIFY_EMAIL_ERROR","SEND_RESET_PASSWORD_E
RROR","SEND_IDENTITY_PROVIDER_LINK_ERROR","RESET_PASSWORD_ERROR","ID
ENTITY_PROVIDER_FIRST_LOGIN_ERROR","IDENTITY_PROVIDER_POST_LOGIN_ER
ROR","CUSTOM_REQUIRED_ACTION_ERROR","EXECUTE_ACTIONS_ERROR","CLIEN
T_REGISTER_ERROR","CLIENT_UPDATE_ERROR","CLIENT_DELETE_ERROR"]' -s
eventsExpiration=172800
```

- ***Windows***：

```
c:\> kcadm update events/config -r demorealm -s eventsEnabled=true -s
"enabledEventTypes=
[\"LOGIN_ERROR\",\"REGISTER_ERROR\",\"LOGOUT_ERROR\",\"CODE_TO_TOKEN_ER
ROR\",\"CLIENT_LOGIN_ERROR\",\"FEDERATED_IDENTITY_LINK_ERROR\",\"REMOVE_
FEDERATED_IDENTITY_ERROR\",\"UPDATE_EMAIL_ERROR\",\"UPDATE_PROFILE_ER
ROR\",\"UPDATE_PASSWORD_ERROR\",\"UPDATE_TOTP_ERROR\",\"VERIFY_EMAIL_E
RROR\",\"REMOVE_TOTP_ERROR\",\"SEND_VERIFY_EMAIL_ERROR\",\"SEND_RESET_
PASSWORD_ERROR\",\"SEND_IDENTITY_PROVIDER_LINK_ERROR\",\"RESET_PASSW
ORD_ERROR\",\"IDENTITY_PROVIDER_FIRST_LOGIN_ERROR\",\"IDENTITY_PROVIDE
R_POST_LOGIN_ERROR\",\"CUSTOM_REQUIRED_ACTION_ERROR\",\"EXECUTE_ACTI
ONS_ERROR\",\"CLIENT_REGISTER_ERROR\",\"CLIENT_UPDATE_ERROR\",\"CLIENT_
DELETE_ERROR\"]" -s eventsExpiration=172800
```

*您可以将存储的事件类型重置为 所有可用事件类型。将值设为空列表与枚举* ***all*** *相同。*

```
$ kcadm.sh update events/config -r demorealm -s enabledEventTypes=[]
```

*您可以启用审计事件存储。*

```
$ kcadm.sh update events/config -r demorealm -s adminEventsEnabled=true -s
adminEventsDetailsEnabled=true
```

*您可以获取最后* ***100*** *个事件。事件从最新到最旧的排序。*

```
$ kcadm.sh get events --offset 0 --limit 100
```

*您可以删除所有保存的事件。*

```
$ kcadm delete events
```

*刷新缓存*

1. *使用带有其中一个端点的* ***create*** *命令清除缓存：*

- ***clear-realm-cache***
- ***clear-user-cache***
- ***clear-keys-cache***

2. *将 **realm** 设置为与目标域相同的值。*

   *例如：*

```
$ kcadm.sh create clear-realm-cache -r demorealm -s realm=demorealm
$ kcadm.sh create clear-user-cache -r demorealm -s realm=demorealm
$ kcadm.sh create clear-keys-cache -r demorealm -s realm=demorealm
```

*从导出的 **.json** 文件导入域*

1. *在 **partialImport** 端点上使用 **create** 命令。*
2. *将 **ifResourceExists** 设置为 **FAIL**、**SKIP** 或 **OVERWRITE**。*
3. *使用 **-f** 提交导出的域 **.json** 文件。*

   *例如：*

```
$ kcadm.sh create partialImport -r demorealm2 -s ifResourceExists=FAIL -o -f
demorealm.json
```

   *如果域尚不存在，请首先创建它。*

   *例如：*

```
$ kcadm.sh create realms -s realm=demorealm2 -s enabled=true
```

### *17.7. 角色操作*

*创建 **realm** 角色*

*使用 **roles** 端点来创建 **realm** 角色。*

```
$ kcadm.sh create roles -r demorealm -s name=user -s 'description=Regular user with a limited set of permissions'
```

*创建客户端角色*

1. *确定客户端。*
2. *使用 **get** 命令列出可用的客户端。*

   ```
   $ kcadm.sh get clients -r demorealm --fields id,clientId
   ```

3. *使用 **clientId** 属性创建新角色来构造端点 **URI**，如 **clients/ID/roles**。*

   *例如：*

   ```
   $ kcadm.sh create clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles -r demorealm -s name=editor -s 'description=Editor can edit, and publish any article'
   ```

*列出 **realm** 角色*

*对 **roles** 端点使用 **get** 命令来列出现有的 **realm** 角色。*

```
$ kcadm.sh get roles -r demorealm
```

*您还可以使用 **get-roles** 命令。*

```
$ kcadm.sh get-roles -r demorealm
```

*列出客户端角色*

***Red Hat build of Keycloak** 有一个专用的 **get-roles** 命令，可简化域和客户端角色列表。命令是 **get** 命令的扩展，其行为与 **get** 命令相同，但具有列出角色的额外语义。*

*通过传递 **clientId (--cclientid)**选项或 **id (--c id )**选项，使用 **get-roles** 命令来识别客户端以列出客户端角色。*

例如：

```
$ kcadm.sh get-roles -r demorealm --cclientid realm-management
```

获取特定的域角色

使用 **get** 命令和角色名称，为特定域角色**(role /ROLE_NAME )**构造端点 **URI**，其中 **user** 是现有角色的名称。

例如：

```
$ kcadm.sh get roles/user -r demorealm
```

您可以使用 **get-roles** 命令，传递角色名称**(--rolename** 选项**)**或 **ID (--roleid** 选项**)**。

例如：

```
$ kcadm.sh get-roles -r demorealm --rolename user
```

获取特定的客户端角色

使用 **get-roles** 命令，传递 **clientId** 属性**(--cclientid** 选项**)**或 **ID** 属性**(--cid** 选项**)**来识别客户端，并传递角色名称**(--rolename** 选项**)**或角色 **ID** 属性**(--roleid)**来标识特定的客户端角色。

例如：

```
$ kcadm.sh get-roles -r demorealm --cclientid realm-management --rolename manage-clients
```

更新 **realm** 角色

使用 **update** 命令和您用来获取特定域角色的端点 **URI**。

例如：

```
$ kcadm.sh update roles/user -r demorealm -s 'description=Role representing a regular user'
```

更新客户端角色

使用 **update** 命令和您用来获取特定客户端角色的端点 **URI**。

例如：

```
$ kcadm.sh update clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles/editor -r demorealm -s
'description=User that can edit, and publish articles'
```

删除 **realm** 角色

使用 **delete** 命令和您用来获取特定域角色的端点 **URI**。

例如：

```
$ kcadm.sh delete roles/user -r demorealm
```

删除客户端角色

使用 **delete** 命令和您用来获取特定客户端角色的端点 **URI**。

例如：

```
$ kcadm.sh delete clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles/editor -r demorealm
```

列出复合角色的已分配、可用和有效域角色

使用 **get-roles** 命令列出为复合角色分配的、可用和有效的域角色。

1. 要列出复合角色的 分配的 域角色，请按名称(**--rname** 选项)或 **ID** （**--rid** 选项指定目标复合角色。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --rname testrole
   ```

2. 使用 **-- effective** 选项列出有效的域角色。

   例如：

```
$ kcadm.sh get-roles -r demorealm --rname testrole --effective
```

3. 使用 **--available** 选项列出您可以添加到复合角色的域角色。

   例如：

```
$ kcadm.sh get-roles -r demorealm --rname testrole --available
```

列出复合角色的已分配、可用和有效的客户端角色

使用 **get-roles** 命令列出为复合角色分配的、可用和有效的客户端角色。

1. 要列出为复合角色分配的客户端角色，您可以按名称(**--rname** 选项)或 **ID (--rid** 选项)和客户端(**--cclientid** 选项)或 **ID** （**--cid** 选项指定目标复合角色。

   例如：

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management
```

2. 使用 **-- effective** 选项列出有效的域角色。

   例如：

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management --effective
```

3. 使用 **--available** 选项列出您可以添加到目标复合角色的域角色。

   例如：

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management --
available
```

将 **realm** 角色添加到复合角色

红帽构建的 **Keycloak** 提供了一个 **add-roles** 命令，用于添加域角色和客户端角色。

本例将用户角色添加到复合角色 **testrole**。

```
$ kcadm.sh add-roles --rname testrole --rolename user -r demorealm
```

从复合角色中删除域角色

红帽构建的 **Keycloak** 提供了一个 **remove-roles** 命令，用于删除域角色和客户端角色。

以下示例从目标复合角色 **testrole** 中删除 用户角色。

```
$ kcadm.sh remove-roles --rname testrole --rolename user -r demorealm
```

将客户端角色添加到域角色

红帽构建的 **Keycloak** 提供了一个 **add-roles** 命令，用于添加域角色和客户端角色。

以下示例将客户端 **realm-management**、**create-client** 和 **view-users** 中定义的角色添加到 **testrole** 复合角色。

```
$ kcadm.sh add-roles -r demorealm --rname testrole --cclientid realm-management --rolename
create-client --rolename view-users
```

将客户端角色添加到客户端角色

1. 使用 **get-roles** 命令确定复合客户端角色的 **ID**。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --cclientid test-client --rolename operations
   ```

2. 假设存在一个名为 **test-client** 的 **clientId** 属性、名为 **support** 的客户端角色，以及名为 **operations** 的客户端角色，它成为复合角色，其 **ID** 为 **"fc400897-ef6a-4e8c-872b-1581b7fa8a71"**。

3. 使用以下示例将另一个角色添加到复合角色。

   ```
   $ kcadm.sh add-roles -r demorealm --cclientid test-client --rid fc400897-ef6a-4e8c-872b-
   1581b7fa8a71 --rolename support
   ```

4. 使用 **get-roles --all** 命令列出复合角色的角色。

例如：

```
$ kcadm.sh get-roles --rid fc400897-ef6a-4e8c-872b-1581b7fa8a71 --all
```

从复合角色中删除客户端角色

使用 **remove-roles** 命令从复合角色中删除客户端角色。

使用以下示例从 **testrole** 复合角色中删除客户端 **realm-management** 中定义的两个角色，即 **create-client** 角色和 **view-users** 角色。

```
$ kcadm.sh remove-roles -r demorealm --rname testrole --cclientid realm-management --rolename
create-client --rolename view-users
```

将客户端角色添加到组中

使用 **add-roles** 命令添加 **realm** 角色和客户端角色。

以下示例将客户端 **realm-management**、**create-client** 和 **view-users** 中定义的角色添加到 **Group** 组 **(--gname** 选项**)**。或者，您可以根据 **ID (--gid** 选项**)**指定组。

如需更多信息，请参阅 组操作。

```
$ kcadm.sh add-roles -r demorealm --gname Group --cclientid realm-management --rolename
create-client --rolename view-users
```

从组中删除客户端角色

使用 **remove-roles** 命令从组中删除客户端角色。

以下示例从组组中删除客户端域管理、**create-client** 和 **view-users** 中定义的两个角色。

如需更多信息，请参阅 组操作。

```
$ kcadm.sh remove-roles -r demorealm --gname Group --cclientid realm-management --rolename
create-client --rolename view-users
```

### 17.8. 客户端操作

*创建客户端*

1. *在客户端端点上运行 **create** 命令以创建新客户端。*

   *例如：*

   ```
   $ kcadm.sh create clients -r demorealm -s clientId=myapp -s enabled=true
   ```

2. *如果为适配器设置 **secret** 进行验证，请指定 **secret**。*

   *例如：*

   ```
   $ kcadm.sh create clients -r demorealm -s clientId=myapp -s enabled=true -s
   clientAuthenticatorType=client-secret -s secret=d0b8122f-8dfb-46b7-b68a-f5cc4e25d000
   ```

*列出客户端*

*在 **clients** 端点上使用 **get** 命令来列出客户端。*

*这个示例过滤输出以只列出 **id** 和 **clientId** 属性：*

```
$ kcadm.sh get clients -r demorealm --fields id,clientId
```

*获取特定客户端*

*使用客户端 **ID** 构建以特定客户端为目标的端点 **URI**，如 **client /ID**。*

*例如：*

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm
```

*为特定客户端获取当前 **secret***

*使用客户端 **ID** 构造端点 **URI**，如 **clients/ID/client-secret**。*

*例如：*

```
$ kcadm.sh get clients/$CID/client-secret -r demorealm
```

*为特定客户端生成新 **secret***

*使用客户端 **ID** 构造端点 **URI**，如 **clients/ID/client-secret**。*

*例如：*

```
$ kcadm.sh create clients/$CID/client-secret -r demorealm
```

*为特定客户端更新当前 **secret***

*使用客户端 **ID** 构造端点 **URI**，如客户端/**ID**。*

*例如：*

```
$ kcadm.sh update clients/$CID -s "secret=newSecret" -r demorealm
```

*获取特定客户端的适配器配置文件**(keycloak.json)***

*使用客户端 **ID** 构建以特定客户端为目标的端点 **URI**，如 **clients/ID/installation/providers/keycloak-oidc-keycloak-json**。*

*例如：*

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3/installation/providers/keycloak-oidc-keycloak-json -r demorealm
```

*为特定客户端获取 **WildFly** 子系统适配器配置*

*使用客户端 **ID** 构建以特定客户端为目标的端点 **URI**，如 **clients/ID/installation/providers/keycloak-oidc-jboss-subsystem**。*

*例如：*

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3/installation/providers/keycloak-oidc-jboss-subsystem -r demorealm
```

*获取特定客户端的 **Docker-v2** 示例配置*

*使用客户端 **ID** 构建以特定客户端为目标的端点 **URI**，如 **clients/ID/installation/providers/docker-v2-compose-yaml**。*

*响应采用 **.zip** 格式。*

*例如：*

```
$ kcadm.sh get http://localhost:8080/admin/realms/demorealm/clients/8f271c35-44e3-446f-8953-b0893810ebe7/installation/providers/docker-v2-compose-yaml -r demorealm > keycloak-docker-compose-yaml.zip
```

*更新客户端*

*使用带有您用来获取特定客户端的同一端点 **URI** 的 **update** 命令。*

*例如：*

- ***linux:***

```
$ kcadm.sh update clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm -s enabled=false -s publicClient=true -s 'redirectUris=["http://localhost:8080/myapp/*"]' -s baseUrl=http://localhost:8080/myapp -s adminUrl=http://localhost:8080/myapp
```

- ***Windows**：*

```
c:\> kcadm update clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm -s enabled=false -s publicClient=true -s "redirectUris=[\"http://localhost:8080/myapp/*\"]" -s baseUrl=http://localhost:8080/myapp -s adminUrl=http://localhost:8080/myapp
```

*删除客户端*

*使用 **delete** 命令和您用来获取特定客户端的同一端点 **URI**。*

*例如：*

```
$ kcadm.sh delete clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm
```

*为客户端服务帐户添加或删除角色*

*客户端的服务帐户是具有用户名 **service-account-CLIENT_ID** 的用户帐户。您可以作为常规帐户对此帐户执行相同的用户操作。*

### 17.9. 用户操作

创建用户

在 **users** 端点上运行 **create** 命令，以创建新用户。

例如：

```
$ kcadm.sh create users -r demorealm -s username=testuser -s enabled=true
```

列出用户

使用 **users** 端点列出用户。目标用户在下次登录时必须更改其密码。

例如：

```
$ kcadm.sh get users -r demorealm --offset 0 --limit 1000
```

您可以根据 用户名、**firstName**、**lastName** 或 **email** 过滤用户。

例如：

```
$ kcadm.sh get users -r demorealm -q email=google.com
$ kcadm.sh get users -r demorealm -q username=testuser
```

注意

过滤不使用完全匹配。本例与 **username** 属性的值与 ***testuser*** 模式匹配。

您可以通过指定多个 **-q** 选项来过滤多个属性。红帽构建的 **Keycloak** 返回仅与所有属性条件匹配的用户。

获取特定用户

使用用户 **ID** 编写端点 **URI**，如 **users/USER_ID**。

例如：

```
$ kcadm.sh get users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm
```

更新用户

使用带有您用来获取特定用户的同一端点 **URI** 的 **update** 命令。

例如：

- **linux:**

```
$ kcadm.sh update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm -s
'requiredActions=
["VERIFY_EMAIL","UPDATE_PROFILE","CONFIGURE_TOTP","UPDATE_PASSWORD"]'
```

- **Windows**：

```
c:\> kcadm update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm -s
"requiredActions=
[\"VERIFY_EMAIL\",\"UPDATE_PROFILE\",\"CONFIGURE_TOTP\",\"UPDATE_PASSWORD
\"]"
```

删除用户

使用 **delete** 命令和您用来获取特定用户的同一端点 **URI**。

例如：

```
$ kcadm.sh delete users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm
```

重置用户的密码

使用专用的 **set-password** 命令重置用户的密码。

例如：

```
$ kcadm.sh set-password -r demorealm --username testuser --new-password NEWPASSWORD --
temporary
```

此命令为用户设置临时密码。目标用户在下次登录时必须更改密码。

您可以使用 **--userid** 使用 **id** 属性来指定用户。

您可以通过在由您用来获取特定用户（如 **users/USER_ID/reset-password** ）组成的端点上的 **update** 命令实现相同的结果。

例如：

```
$ kcadm.sh update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2/reset-password -r demorealm -s
type=password -s value=NEWPASSWORD -s temporary=true -n
```

**n** 参数可确保红帽构建的 **Keycloak** 在 **PUT** 命令之前执行 **PUT** 命令，而无需执行 **GET** 命令。这是必要的，因为 **reset-password** 端点不支持 **GET**。

列出用户的已分配、可用和有效的域角色

您可以使用 **get-roles** 命令列出分配给用户的、可用和有效的域角色。

1. 根据用户名或 **ID** 指定目标用户，以列出 用户分配的域 角色。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --uusername testuser
   ```

2. 使用 **-- effective** 选项列出有效的域角色。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --uusername testuser --effective
   ```

3. 使用 **--available** 选项列出您可以添加到用户的域角色。

   例如：

-

```
$ kcadm.sh get-roles -r demorealm --uusername testuser --available
```

*列出分配给用户的、可用和有效的客户端角色*

*使用 **get-roles** 命令列出分配给用户的、可用和有效的客户端角色。*

1. *根据用户名(**--uusername** 选项)或 **ID (--uid** 选项)和客户端(**--cclientid** 选项)或 **ID (--cid** 选项)指定目标用户，以列出用户 分配的 客户端角色。*

   *例如：*

   ```
   $ kcadm.sh get-roles -r demorealm --uusername testuser --cclientid realm-management
   ```

2. *使用 **-- effective** 选项列出有效的域角色。*

   *例如：*

   ```
   $ kcadm.sh get-roles -r demorealm --uusername testuser --cclientid realm-management --effective
   ```

3. *使用 **--available** 选项列出您可以添加到用户的域角色。*

   *例如：*

   ```
   $ kcadm.sh get-roles -r demorealm --uusername testuser --cclientid realm-management --available
   ```

*向用户添加域角色*

*使用 **add-roles** 命令，向用户添加 **realm** 角色。*

*使用以下示例将用户角色添加到用户 **testuser** 中：*

```
$ kcadm.sh add-roles --uusername testuser --rolename user -r demorealm
```

*从用户中删除域角色*

*使用 **remove-roles** 命令从用户中删除 **realm** 角色。*

使用以下示例从用户 **testuser** 中删除用户角色：

```
$ kcadm.sh remove-roles --uusername testuser --rolename user -r demorealm
```

向用户添加客户端角色

使用 **add-roles** 命令，向用户添加客户端角色。

使用以下示例，将客户端域管理 中定义的两个角色( **create-client** 角色和 **view-users** 角色)添加到用户 **testuser**。

```
$ kcadm.sh add-roles -r demorealm --uusername testuser --cclientid realm-management --rolename
create-client --rolename view-users
```

从用户中删除客户端角色

使用 **remove-roles** 命令从用户中删除客户端角色。

使用以下示例删除域管理客户端中定义的两个角色：

```
$ kcadm.sh remove-roles -r demorealm --uusername testuser --cclientid realm-management --
rolename create-client --rolename view-users
```

列出用户的会话

1. 确定用户的 **ID**，
2. 使用 **ID** 编写端点 **URI**，如 **users/ID/sessions**。
3. 使用 **get** 命令检索用户会话的列表。

   例如：

   ```
   $ kcadm.sh get users/6da5ab89-3397-4205-afaa-e201ff638f9e/sessions -r demorealm
   ```

从特定会话注销用户

1. 如前面所述，确定会话的 **ID**。

2. 使用会话的 **ID** 来编写端点 **URI**，如 **sessions/ID**。

3. 使用 **delete** 命令使会话无效。

例如：

```
$ kcadm.sh delete sessions/d0eaa7cc-8c5d-489d-811a-69d3c4ec84d1 -r demorealm
```

从所有会话注销用户

使用用户的 **ID** 来构造端点 **URI**，如用户/**ID**/注销。

使用 **create** 命令在该端点 **URI** 上执行 **POST**。

例如：

```
$ kcadm.sh create users/6da5ab89-3397-4205-afaa-e201ff638f9e/logout -r demorealm -s
realm=demorealm -s user=6da5ab89-3397-4205-afaa-e201ff638f9e
```

### 17.10. 组操作

创建组

在 **groups** 端点上使用 **create** 命令来创建新组。

例如：

```
$ kcadm.sh create groups -r demorealm -s name=Group
```

列出组

对 **groups** 端点使用 **get** 命令来列出组。

例如：

```
$ kcadm.sh get groups -r demorealm
```

*获取特定组*

*使用组的 **ID** 来构造端点 **URI**，如 **groups/GROUP_ID**。*

*例如：*

```
$ kcadm.sh get groups/51204821-0580-46db-8f2d-27106c6b5ded -r demorealm
```

*更新组*

*使用带有您用来获取特定组的同一端点 **URI** 的 **update** 命令。*

*例如：*

```
$ kcadm.sh update groups/51204821-0580-46db-8f2d-27106c6b5ded -s 'attributes.email=
["group@example.com"]' -r demorealm
```

*删除组*

*使用 **delete** 命令和您用来获取特定组的同一端点 **URI**。*

*例如：*

```
$ kcadm.sh delete groups/51204821-0580-46db-8f2d-27106c6b5ded -r demorealm
```

*创建子组*

*通过列出组来查找父组的 **ID**。使用该 **ID** 构造端点 **URI**，如 **groups/GROUP_ID/children**。*

*例如：*

```
$ kcadm.sh create groups/51204821-0580-46db-8f2d-27106c6b5ded/children -r demorealm -s
name=SubGroup
```

*将组移动到另一个组下*

1. *查找现有父组的 **ID**，以及现有子组的 **ID**。*

2. 使用父组的 **ID** 来构造端点 **URI**，如 **groups/PARENT_GROUP_ID/children**。

3. 在此端点上运行 **create** 命令，并将子组的 **ID** 传递为 **JSON** 正文。

例如：

```
$ kcadm.sh create groups/51204821-0580-46db-8f2d-27106c6b5ded/children -r demorealm -s
id=08d410c6-d585-4059-bb07-54dcb92c5094 -s name=SubGroup
```

获取特定用户的组

使用用户的 **ID** 来确定用户在组中的成员资格，以编写端点 **URI**，如 **users/USER_ID/groups**。

例如：

```
$ kcadm.sh get users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups -r demorealm
```

将用户添加到组中

使用带有用户 **ID** 和组 **ID** （如用户 **/USER_ID/groups/GROUP_ID**）的端点 **URI** 和组 **ID** 的 **update** 命令，将用户添加到组中。

例如：

```
$ kcadm.sh update users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups/ce01117a-7426-4670-
a29a-5c118056fe20 -r demorealm -s realm=demorealm -s userId=b544f379-5fc4-49e5-8a8d-
5cfb71f46f53 -s groupId=ce01117a-7426-4670-a29a-5c118056fe20 -n
```

从组中删除用户

在用于将用户添加到组中的同一端点 **URI** 上使用 **delete** 命令，如 **users/USER_ID/groups/GROUP_ID**，从组中删除用户。

例如：

```
$ kcadm.sh delete users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups/ce01117a-7426-4670-a29a-
5c118056fe20 -r demorealm
```

列出组的已分配、可用和有效域角色

使用专用的 **get-roles** 命令列出为组分配的、可用和有效的域角色。

1. 根据名称(**--gname** 选项)、**path (--gpath** 选项)或 **ID (--gid** 选项)指定目标组，以列出组的 分配的 域角色。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --gname Group
   ```

2. 使用 **-- effective** 选项列出有效的域角色。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --gname Group --effective
   ```

3. 使用 **--available** 选项列出您可以添加到组中的域角色。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --gname Group --available
   ```

列出组的已分配、可用和有效的客户端角色

使用 **get-roles** 命令列出为组分配的、可用和有效的客户端角色。

1. 按名称指定目标组(**--gname** 选项)或 **ID (--gid** 选项)，

2. 通过 **clientId** 属性(**--cclientid** 选项)或 **ID** （**--id** 选项指定客户端，以列出用户 分配的 客户端角色。

   例如：

   ```
   $ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management
   ```

3. 使用 **-- effective** 选项列出有效的域角色。

例如：

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management --effective
```

4. 使用 **--available** 选项列出您仍然可以添加到组中的域角色。

例如：

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management --available
```

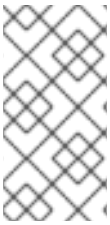
### 17.11. 身份提供程序操作

列出可用的身份提供程序

使用 **serverinfo** 端点列出可用的身份提供程序。

例如：

```
$ kcadm.sh get serverinfo -r demorealm --fields 'identityProviders(*)'
```



注意

**Red Hat build of Keycloak** 处理 **serverinfo** 端点与 **realm** 端点类似。红帽构建的 **Keycloak** 不会解析与目标域相关的端点，因为它存在于任何特定域之外。

列出配置的身份提供程序

使用 **identity-provider/instances** 端点。

例如：

```
$ kcadm.sh get identity-provider/instances -r demorealm --fields alias,providerId,enabled
```

获取特定的身份提供程序

使用身份提供程序的 **alias** 属性来构建端点 **URI**，如 **identity-provider/instances/ALIAS**，以获取特定的身份提供程序。

例如：

```
$ kcadm.sh get identity-provider/instances/facebook -r demorealm
```

删除特定的配置的身份提供程序

使用 **delete** 命令以及您用来获取特定配置的身份提供程序的同一端点 **URI**，以删除特定的身份提供程序。

例如：

```
$ kcadm.sh delete identity-provider/instances/facebook -r demorealm
```

配置 **Keycloak OpenID Connect** 身份提供程序

1. 在创建新身份提供程序实例时，使用 **keycloak-oidc** 作为 **providerId**。

2. 提供 **config** 属性： **authorizationUrl**、**tokenUrl**、**clientId** 和 **clientSecret**。

   例如：

   ```
   $ kcadm.sh create identity-provider/instances -r demorealm -s alias=keycloak-oidc -s
   providerId=keycloak-oidc -s enabled=true -s 'config.useJwksUrl="true"' -s
   config.authorizationUrl=http://localhost:8180/realms/demorealm/protocol/openid-connect/auth
   -s config.tokenUrl=http://localhost:8180/realms/demorealm/protocol/openid-connect/token -s
   config.clientId=demo-oidc-provider -s config.clientSecret=secret
   ```

配置 **OpenID Connect** 身份提供程序

配置通用 **OpenID Connect** 供应商的方式与配置 **Keycloak OpenID Connect** 供应商的方式相同，但将 **providerId** 属性值设置为 **oidc**。

配置 **SAML 2** 身份提供程序

1. 使用 **saml** 作为 **providerId**。

2. 提供 **config** 属性： **singleSignOnServiceUrl,nameIDPolicyFormat,** 和 **signatureAlgorithm**。

例如：

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=saml -s providerId=saml -s
enabled=true -s 'config.useJwksUrl="true"' -s
config.singleSignOnServiceUrl=http://localhost:8180/realms/saml-broker-realm/protocol/saml -s
config.nameIDPolicyFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent -s
config.signatureAlgorithm=RSA_SHA256
```

配置 **Facebook** 身份提供程序

1. 使用 **facebook** 作为 **providerId**。

2. 提供 **config** 属性： **clientId** 和 **clientSecret**。您可以在适用于应用程序的 **Facebook Developers** 应用程序配置页面中找到这些属性。如需更多信息，请参阅 **Facebook** 身份代理 页面。

   例如：

   ```
   $ kcadm.sh create identity-provider/instances -r demorealm -s alias=facebook -s
   providerId=facebook -s enabled=true  -s 'config.useJwksUrl="true"' -s
   config.clientId=FACEBOOK_CLIENT_ID -s
   config.clientSecret=FACEBOOK_CLIENT_SECRET
   ```

配置 **Google** 身份提供程序

1. 使用 **google** 作为 **providerId**。

2. 提供 **config** 属性： **clientId** 和 **clientSecret**。您可以在应用程序的 **Google Developers** 应用程序配置页面中找到这些属性。如需更多信息，请参阅 **Google** 身份代理 页面。

   例如：

   ```
   $ kcadm.sh create identity-provider/instances -r demorealm -s alias=google -s
   providerId=google -s enabled=true  -s 'config.useJwksUrl="true"' -s
   config.clientId=GOOGLE_CLIENT_ID -s config.clientSecret=GOOGLE_CLIENT_SECRET
   ```

配置 **Twitter** 身份提供程序

1. 使用 **twitter** 作为 **providerId**。

2.

*提供 **config** 属性 **clientId** 和 **clientSecret**。您可以在应用程序的 **Twitter** 应用程序配置页面中找到这些属性。如需更多信息，请参阅 **Twitter** 身份代理 页面。*

*例如：*

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=google -s
providerId=google -s enabled=true  -s 'config.useJwksUrl="true"' -s
config.clientId=TWITTER_API_KEY -s config.clientSecret=TWITTER_API_SECRET
```

*配置 **GitHub** 身份提供程序*

1. *使用 **github** 作为 **providerId**。*

2. *提供 **config** 属性 **clientId** 和 **clientSecret**。您可以在应用程序的 **GitHub Developer Application Settings** 页面中找到这些属性。如需更多信息，请参阅 **GitHub** 身份代理 页面。*

   *例如：*

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=github -s
providerId=github -s enabled=true  -s 'config.useJwksUrl="true"' -s
config.clientId=GITHUB_CLIENT_ID -s config.clientSecret=GITHUB_CLIENT_SECRET
```

*配置 **LinkedIn** 身份提供程序*

1. *使用 **linkedin** 作为 **providerId**。*

2. *提供 **config** 属性 **clientId** 和 **clientSecret**。您可以在应用程序的 **LinkedIn Developer Console** 应用程序页面中找到这些属性。如需更多信息，请参阅 **LinkedIn** 身份代理 页面。*

   *例如：*

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=linkedin -s
providerId=linkedin -s enabled=true  -s 'config.useJwksUrl="true"' -s
config.clientId=LINKEDIN_CLIENT_ID -s config.clientSecret=LINKEDIN_CLIENT_SECRET
```

*配置 **Microsoft Live** 身份提供程序*

1. *使用 **microsoft** 作为 **providerId**。*

2. *提供 **config** 属性 **clientId** 和 **clientSecret**。您可以在应用程序的 **Microsoft Application***

*Registration Portal 页面中找到这些属性。如需更多信息，请参阅 Microsoft 身份代理 页面。*

*例如：*

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=microsoft -s
providerId=microsoft -s enabled=true  -s 'config.useJwksUrl="true"' -s
config.clientId=MICROSOFT_APP_ID -s config.clientSecret=MICROSOFT_PASSWORD
```

*配置 Stack Overflow 身份提供程序*

1. *使用 stackoverflow 命令作为 providerId。*

2. *提供 config 属性 clientId、clientSecret 和 key。您可以在应用程序的 Stack Apps OAuth 页面中找到这些属性。如需更多信息，请参阅 Stack Overflow 身份代理 页面。*

   *例如：*

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=stackoverflow -s
providerId=stackoverflow -s enabled=true  -s 'config.useJwksUrl="true"' -s
config.clientId=STACKAPPS_CLIENT_ID -s
config.clientSecret=STACKAPPS_CLIENT_SECRET -s config.key=STACKAPPS_KEY
```

### *17.12. 存储供应商操作*

*配置 Kerberos 存储供应商*

1. *对 组件 端点使用 create 命令。*

2. *将 realm id 指定为 parentId 属性的值。*

3. *将 kerberos 指定为 providerId 属性的值，将 org.keycloak.storage.UserStorageProvider 指定为 providerType 属性的值。*

4. *例如：*

```
$ kcadm.sh create components -r demorealm -s parentId=demorealmId -s id=demokerberos
-s name=demokerberos -s providerId=kerberos -s
providerType=org.keycloak.storage.UserStorageProvider -s 'config.priority=["0"]' -s
'config.debug=["false"]' -s 'config.allowPasswordAuthentication=["true"]' -s 'config.editMode=
```

```
["UNSYNCED"]' -s 'config.updateProfileFirstLogin=["true"]' -s
'config.allowKerberosAuthentication=["true"]' -s 'config.kerberosRealm=["KEYCLOAK.ORG"]'
-s 'config.keyTab=["http.keytab"]' -s 'config.serverPrincipal=
["HTTP/localhost@KEYCLOAK.ORG"]' -s 'config.cachePolicy=["DEFAULT"]'
```

*配置 **LDAP** 用户存储供应商*

1. *对 组件 端点使用 **create** 命令。*

2. *将 **ldap** 指定为 **providerId** 属性的值，将 **org.keycloak.storage.UserStorageProvider** 指定为 **providerType** 属性的值。*

3. *提供 **realm ID** 作为 **parentId** 属性的值。*

4. *使用以下示例创建 **Kerberos** 集成的 **LDAP** 供应商。*

```
$ kcadm.sh create components -r demorealm -s name=kerberos-ldap-provider -s
providerId=ldap -s providerType=org.keycloak.storage.UserStorageProvider -s
parentId=3d9c572b-8f33-483f-98a6-8bb421667867  -s 'config.priority=["1"]' -s
'config.fullSyncPeriod=["-1"]' -s 'config.changedSyncPeriod=["-1"]' -s 'config.cachePolicy=
["DEFAULT"]' -s config.evictionDay=[] -s config.evictionHour=[] -s config.evictionMinute=[] -s
config.maxLifespan=[] -s 'config.batchSizeForSync=["1000"]' -s 'config.editMode=
["WRITABLE"]' -s 'config.syncRegistrations=["false"]' -s 'config.vendor=["other"]' -s
'config.usernameLDAPAttribute=["uid"]' -s 'config.rdnLDAPAttribute=["uid"]' -s
'config.uuidLDAPAttribute=["entryUUID"]' -s 'config.userObjectClasses=["inetOrgPerson,
organizationalPerson"]' -s 'config.connectionUrl=["ldap://localhost:10389"]'  -s
'config.usersDn=["ou=People,dc=keycloak,dc=org"]' -s 'config.authType=["simple"]' -s
'config.bindDn=["uid=admin,ou=system"]' -s 'config.bindCredential=["secret"]' -s
'config.searchScope=["1"]' -s 'config.useTruststoreSpi=["always"]' -s
'config.connectionPooling=["true"]' -s 'config.pagination=["true"]' -s
'config.allowKerberosAuthentication=["true"]' -s 'config.serverPrincipal=
["HTTP/localhost@KEYCLOAK.ORG"]' -s 'config.keyTab=["http.keytab"]' -s
'config.kerberosRealm=["KEYCLOAK.ORG"]' -s 'config.debug=["true"]' -s
'config.useKerberosForPasswordAuthentication=["true"]'
```

*删除用户存储供应商实例*

1. *使用存储提供程序实例的 **id** 属性来编写端点 **URI**，如 **components/ID**。*

2. *针对此端点运行 **delete** 命令。*

   *例如：*

```
$ kcadm.sh delete components/3d9c572b-8f33-483f-98a6-8bb421667867 -r demorealm
```

*为特定用户存储供应商触发所有用户同步*

1. *使用存储供应商的 **id** 属性来编写端点 **URI**，如 **user-storage/ID_OF_USER_STORAGE_INSTANCE/sync**。*

2. *添加 **action=triggerFullSync** 查询参数。*

3. *运行 **create** 命令。*

   *例如：*

```
$ kcadm.sh create user-storage/b7c63d02-b62a-4fc1-977c-947d6a09e1ea/sync?
action=triggerFullSync
```

*为特定用户存储供应商触发更改的用户同步*

1. *使用存储供应商的 **id** 属性来编写端点 **URI**，如 **user-storage/ID_OF_USER_STORAGE_INSTANCE/sync**。*

2. *添加 **action=triggerChangedUsersSync** 查询参数。*

3. *运行 **create** 命令。*

   *例如：*

```
$ kcadm.sh create user-storage/b7c63d02-b62a-4fc1-977c-947d6a09e1ea/sync?
action=triggerChangedUsersSync
```

*测试 **LDAP** 用户存储连接*

1. *在 **testLDAPConnection** 端点上运行 **get** 命令。*

2. *提供查询参数 **bindCredential**、**bindDn**、**connectionUrl** 和 **useTruststoreSpi**。*

3. 将 **action** 查询参数设置为 **testConnection**。

例如：

```
$ kcadm.sh create testLDAPConnection -s action=testConnection -s bindCredential=secret -s
bindDn=uid=admin,ou=system -s connectionUrl=ldap://localhost:10389 -s
useTruststoreSpi=always
```

测试 **LDAP** 用户存储身份验证

1. 在 **testLDAPConnection** 端点上运行 **get** 命令。

2. 提供查询参数 **bindCredential**、**bindDn**、**connectionUrl** 和 **useTruststoreSpi**。

3. 将 **action** 查询参数设置为 **testAuthentication**。

例如：

```
$ kcadm.sh create testLDAPConnection -s action=testAuthentication -s
bindCredential=secret -s bindDn=uid=admin,ou=system -s
connectionUrl=ldap://localhost:10389 -s useTruststoreSpi=always
```

### 17.13. 添加映射程序

添加硬编码的角色 **LDAP** 映射器

1. 在 组件 端点上运行 **create** 命令。

2. 将 **providerType** 属性设置为 **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**。

3. 将 **parentId** 属性设置为 **LDAP** 提供程序实例的 **ID**。

4. 将 **providerId** 属性设置为 **hardcoded-ldap-role-mapper**。确保提供 **role** 配置参数的值。

例如：

```
$ kcadm.sh create components -r demorealm -s name=hardcoded-ldap-role-mapper -s
providerId=hardcoded-ldap-role-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config.role=["realm-
management.create-client"]'
```

*添加 **MS Active Directory** 映射器*

1. *在 组件 端点上运行 **create** 命令。*
2. *将 **providerType** 属性设置为 **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**。*
3. *将 **parentId** 属性设置为 **LDAP** 提供程序实例的 **ID**。*
4. *将 **providerId** 属性设置为 **msad-user-account-control-mapper**。*

   *例如：*

```
$ kcadm.sh create components -r demorealm -s name=msad-user-account-control-mapper -
s providerId=msad-user-account-control-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea
```

*添加用户属性 **LDAP** 映射器*

1. *在 组件 端点上运行 **create** 命令。*
2. *将 **providerType** 属性设置为 **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**。*
3. *将 **parentId** 属性设置为 **LDAP** 提供程序实例的 **ID**。*
4. *将 **providerId** 属性设置为 **user-attribute-ldap-mapper**。*

   *例如：*

```
$ kcadm.sh create components -r demorealm -s name=user-attribute-ldap-mapper -s
providerId=user-attribute-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."user.model.attribute"=
["email"]' -s 'config."ldap.attribute"=["mail"]' -s 'config."read.only"=["false"]' -s
'config."always.read.value.from.ldap"=["false"]' -s 'config."is.mandatory.in.ldap"=["false"]'
```

*添加组 **LDAP** 映射器*

1. *在 组件 端点上运行 **create** 命令。*

2. *将 **providerType** 属性设置为 **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**。*

3. *将 **parentId** 属性设置为 **LDAP** 提供程序实例的 **ID**。*

4. *将 **providerId** 属性设置为 **group-ldap-mapper**。*

   *例如：*

```
$ kcadm.sh create components -r demorealm -s name=group-ldap-mapper -s
providerId=group-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."groups.dn"=[]' -s
'config."group.name.ldap.attribute"=["cn"]' -s 'config."group.object.classes"=
["groupOfNames"]' -s 'config."preserve.group.inheritance"=["true"]' -s
'config."membership.ldap.attribute"=["member"]' -s 'config."membership.attribute.type"=
["DN"]' -s 'config."groups.ldap.filter"=[]' -s 'config.mode=["LDAP_ONLY"]' -s
'config."user.roles.retrieve.strategy"=["LOAD_GROUPS_BY_MEMBER_ATTRIBUTE"]' -s
'config."mapped.group.attributes"=["admins-group"]' -s
'config."drop.non.existing.groups.during.sync"=["false"]' -s 'config.roles=["admins"]' -s
'config.groups=["admins-group"]' -s 'config.group=[]' -s 'config.preserve=["true"]' -s
'config.membership=["member"]'
```

*添加全名 **LDAP** 映射器*

1. *在 组件 端点上运行 **create** 命令。*

2. *将 **providerType** 属性设置为 **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**。*

3. 将 **parentId** 属性设置为 **LDAP** 提供程序实例的 **ID**。

4. 将 **providerId** 属性设置为 **full-name-ldap-mapper**。

例如：

```
$ kcadm.sh create components -r demorealm -s name=full-name-ldap-mapper -s
providerId=full-name-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."ldap.full.name.attribute"=
["cn"]' -s 'config."read.only"=["false"]' -s 'config."write.only"=["true"]'
```

### 17.14. 身份验证操作

设置密码策略

1. 将 **realm** 的 **passwordPolicy** 属性设置为包含特定策略供应商 **ID** 和可选配置的枚举表达式。

2. 使用以下示例将密码策略设置为默认值。默认值包括：

   - **210,000** 哈希迭代
   - 至少一个特殊字符
   - 至少一个大写字符
   - 至少一个数字字符
   - 不等于 用户的用户名
   - 至少 **8** 个字符长

```
$ kcadm.sh update realms/demorealm -s 'passwordPolicy="hashIterations and
specialChars and upperCase and digits and notUsername and length"'
```

3. 要使用与默认值不同的值，请在括号中传递配置。

4. 使用以下示例将密码策略设置为：

- **300,000** 个哈希迭代
- 至少两个特殊字符
- 至少两个大写字符
- 至少两个小写字符
- 至少两位数字
- 至少 **9** 个字符长
- 不等于 用户的用户名
- 至少重复四个更改

```
$ kcadm.sh update realms/demorealm -s 'passwordPolicy="hashIterations(300000) and specialChars(2) and upperCase(2) and lowerCase(2) and digits(2) and length(9) and notUsername and passwordHistory(4)"'
```

获取当前的密码策略

您可以通过过滤除 **passwordPolicy** 属性之外的所有输出来获取当前的 **realm** 配置。

例如，显示 **demorealm** 的 **passwordPolicy**。

```
$ kcadm.sh get realms/demorealm --fields passwordPolicy
```

列出身份验证流

*在 **authentication/flows** 端点上运行 **get** 命令。*

*例如：*

```
$ kcadm.sh get authentication/flows -r demorealm
```

*获取特定的身份验证流*

*在 **authentication/flows/FLOW_ID** 端点上运行 **get** 命令。*

*例如：*

```
$ kcadm.sh get authentication/flows/febfd772-e1a1-42fb-b8ae-00c0566fafb8 -r demorealm
```

*列出流的执行*

*在 **authentication/flows/FLOW_ALIAS/executions** 端点上运行 **get** 命令。*

*例如：*

```
$ kcadm.sh get authentication/flows/Copy%20of%20browser/executions -r demorealm
```

*在执行中添加配置*

1. *为流获取执行。*
2. *记录流的 **ID**。*
3. *在 **authentication/executions/{executionId}/config** 端点上运行 **create** 命令。*

*例如：*

```
$ kcadm.sh create "authentication/executions/a3147129-c402-4760-86d9-3f2345e401c7/config" -r demorealm -b '{"config":{"x509-cert-auth.mapping-source-selection":"Match SubjectDN using regular expression","x509-cert-auth.regular-expression":"(.*?)(?:$)","x509-cert-auth.mapper-selection":"Custom Attribute Mapper","x509-cert-auth.mapper-selection.user-attribute-name":"usercertificate","x509-cert-auth.crl-checking-enabled":"","x509-cert-auth.crldp-checking-
```

```
enabled":false,"x509-cert-auth.crl-relative-path":"crl.pem","x509-cert-auth.ocsp-checking-enabled":"","x509-cert-auth.ocsp-responder-uri":"","x509-cert-auth.keyusage":"","x509-cert-auth.extendedkeyusage":"","x509-cert-auth.confirmation-page-disallowed":""},"alias":"my_otp_config"}'
```

*获取执行配置*

1. *为流获取执行。*
2. *记下其 **authenticationConfig** 属性，其中包含配置 **ID**。*
3. *在 **authentication/config/ID** 端点上运行 **get** 命令。*

*例如：*

```
$ kcadm get "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r demorealm
```

*更新执行配置*

1. *获取流的执行。*
2. *获取流的 **authenticationConfig** 属性。*
3. *记下属性中的配置 **ID**。*
4. *在 **authentication/config/ID** 端点上运行 **update** 命令。*

*例如：*

```
$ kcadm update "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r demorealm -b '{"id":"dd91611a-d25c-421a-87e2-227c18421833","alias":"my_otp_config","config":{"x509-cert-auth.extendedkeyusage":"","x509-cert-auth.mapper-selection.user-attribute-name":"usercertificate","x509-cert-auth.ocsp-responder-uri":"","x509-cert-auth.regular-expression":"(.*?)(?:$)","x509-cert-auth.crl-checking-enabled":"true","x509-cert-auth.confirmation-page-disallowed":"","x509-cert-auth.keyusage":"","x509-cert-auth.mapper-selection":"Custom Attribute Mapper","x509-cert-auth.crl-relative-path":"crl.pem","x509-cert-auth.crldp-checking-enabled":"false","x509-cert-auth.mapping-source-selection":"Match SubjectDN using regular expression","x509-cert-auth.ocsp-checking-enabled":""}}'
```

*删除执行的配置*

1. *为流获取执行。*
2. *获取 **flows authenticationConfig** 属性。*
3. *记下属性中的配置 **ID**。*
4. *在 **authentication/config/ID** 端点上运行 **delete** 命令。*

*例如：*

```
$ kcadm delete "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r demorealm
```