Red Hat build of Keycloak 24.0

保护应用程序和服务指南

Last Updated: 2024-06-28

法律通告

摘要

本指南包含使用红帽构建的 Keycloak 24.0 保护应用程序和服务的信息。

目录

第 1 章 规划保护应用程序和服务

作为 OAuth2、OpenID Connect 和 SAML 兼容服务器，红帽构建的 Keycloak 可以保护任何应用程序和服务，只要它们使用的技术堆栈支持任何这些协议。有关红帽构建的 Keycloak 支持的安全协议的详情，请参考 服务器管理指南。

其中一些协议的大多数支持已经通过编程语言、框架或它们使用的反向代理获得。利用应用程序生态系统提供的支持是使您的应用程序完全符合安全标准和最佳实践的关键方面，以便您避免了厂商锁定。

对于某些编程语言，红帽构建的 Keycloak 提供库，试图填补缺少特定安全协议的支持或提供更丰富的与服务器集成。这些库是由 **Keycloak Client Adapters** 已知的，如果您无法依赖应用程序生态系统中的可用内容，则应该将它们用作最后的手段。

1.1. 保护应用程序和服务的基本步骤

以下是在红帽构建的 Keycloak 中保护应用程序或服务的基本步骤。

1. 使用以下选项之一将客户端注册到域中：
   - 红帽构建的 Keycloak 管理控制台
   - 客户端注册服务
   - CLI
2. 使用以下选项之一在应用程序中启用 OpenID Connect 或 SAML 协议：
   - 使用应用程序生态系统中的现有 OpenID Connect 和 SAML 支持
   - 使用红帽构建的 Keycloak 适配器

本指南提供了这些步骤的详细说明。您可以在 服务器管理指南 中找到有关如何通过管理控制台将客户端注册到 Keycloak 的红帽构建中的更多详细信息。

1.2. 开始使用

红帽构建的 Keycloak Quickstarts Repository 提供了如何使用不同编程语言和框架保护应用程序和服务的示例。通过其文档和代码库，您将了解应用程序和服务所需的裸机更改，以便使用红帽构建的 Keycloak 来保护它。

另外，请查看以下小节，以了解 OpenID Connect 和 SAML 协议的可信和知名客户端实现的建议。

1.2.1. OpenID Connect

1.2.1.1. JavaScript （客户端）

- JavaScript

1.2.1.2. Node.js （服务器端）

- Node.js

1.2.2. SAML

1.2.2.1. Java

- JBoss EAP

1.3. 术语

本指南中使用了这些术语：

- **客户端是** 与红帽构建的 Keycloak 交互的实体，以验证用户并获取令牌。大多数情况下，客户端是代表用户所代表的应用程序和服务，为用户提供单点登录体验，并使用服务器发布的令牌访问其他服务。客户端也可以是对获取令牌并代表其访问其他服务的实体感兴趣的实体。
- **应用程序** 包括各种应用程序，适用于每个协议的特定平台
- **客户端适配器** 是利用红帽构建的 Keycloak 轻松保护应用程序和服务的库。它们提供对底层平台和框架的紧密集成。
- **创建客户端**和**注册客户端**是相同的操作。**创建** 客户端是使用管理控制台创建客户端的术语。**注册客户端** 是使用红帽构建的 Keycloak 客户端注册服务注册客户端的术语。
- **服务帐户是一种** 客户端类型，它能够代表自己获取令牌。

第 2 章 使用 OPENID CONNECT 来保护应用程序和服务

本节论述了如何使用红帽构建的 Keycloak 保护带有 OpenID Connect 的应用程序和服务。

2.1. 可用的端点

作为完全兼容的 OpenID Connect 提供程序实现，红帽构建的 Keycloak 会公开一组端点，供应用程序和服务用于验证和授权其用户。

本节论述了在与 Red Hat build of Keycloak 交互时，您的应用程序和服务应使用的一些关键端点。

2.1.1. Endpoints

最重要的端点是 **已知的** 配置端点。它列出了红帽构建的 Keycloak 中与 OpenID Connect 实现相关的端点和其他配置选项。端点是：

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

要获取完整的 URL，请为红帽构建的 Keycloak 添加基本 URL，并将 **{realm-name}** 替换为您的域的名称。例如：

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

有些 RP 库从此端点检索所有必需的端点，但有些您可能需要单独列出端点。

2.1.1.1. 授权端点

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

授权端点执行最终用户的身份验证。此身份验证通过将用户代理重定向到此端点来实现。

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

2.1.1.2. 令牌端点

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

令牌端点用于获取令牌。令牌可以通过交换授权代码或直接提供凭据来获取，具体取决于所使用的流。令牌端点也用于在令牌过期时获取新的访问令牌。

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

2.1.1.3. userinfo 端点

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

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

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

2.1.1.4. 注销端点

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

logout 端点会注销经过身份验证的用户。

用户代理可以重定向到端点，这会导致活跃的用户会话被注销。然后，用户代理会重定向到应用程序。

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

2.1.1.5. 证书端点

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

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

2.1.1.6. 内省端点

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

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

有关如何在此端点上调用的更多详细信息，请参阅 OAuth 2.0 令牌内省规格。

2.1.1.7. 动态客户端注册端点

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

动态客户端注册端点用于动态注册客户端。

如需了解更多详细信息，请参阅 客户端注册 一章 和 OpenID Connect Dynamic Client Registration 规格。

2.1.1.8. 令牌撤销端点

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

令牌撤销端点用于撤销令牌。此端点支持刷新令牌和访问令牌。当撤销刷新令牌时，相应客户端的用户同意也会被撤销。

有关如何在此端点上调用的更多详细信息，请参阅 OAuth 2.0 令牌撤销规格。

2.1.1.9. 设备授权端点

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

设备授权端点用于获取设备代码和用户代码。它可以被机密或公共客户端调用。

有关如何在此端点上调用的更多详细信息，请参阅 OAuth 2.0 设备授权规格。

2.1.1.10. Backchannel Authentication 端点

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

backchannel 身份验证端点用于获取 auth_req_id，用于标识客户端发出的身份验证请求。它只能由机密客户端调用。

有关如何在此端点上调用的更多详细信息，请参阅 OpenID Connect Client Initiated Backchannel Authentication Flow 规格。

另请参阅 Red Hat build of Keycloak 文档的其他位置，如 Client Initiated Backchannel Authentication Grant section of this guide and Client Initiated Backchannel Authentication Grant section of Server Administration Guide。

2.2. 支持的授予类型

本节论述了可用于转发方的不同授权类型。

2.2.1. 授权代码

Authorization Code 流将用户代理重定向到红帽构建的 Keycloak。用户通过红帽构建的 Keycloak 成功进行身份验证后，会创建一个授权代码，用户代理将重新重定向到应用程序。然后，应用程序使用授权代码及其凭证从红帽构建的 Keycloak 获取访问令牌、刷新令牌和 ID 令牌。

该流程面向 Web 应用程序，但也建议用于原生应用程序，包括移动应用程序，在那里可以嵌入用户代理。

更多详情请参阅 OpenID Connect 规格中的 授权代码流。

2.2.2. 隐式

Implicit 流的工作方式与授权代码流类似，但不返回授权代码，而是返回访问令牌和 ID 令牌。这种方法减少了额外的调用来交换访问令牌的授权代码。但是，它不包括 Refresh Token。这会产生允许长时间过期的访问令牌；但是，这种方法并不实际，因为它很难使这些令牌无效。另外，您可以在初始访问令牌过期后需要新的重定向来获取新的访问令牌。如果应用程序只想验证用户并处理注销自身，则 Implicit 流很有用。

您可以使用返回 Access Token 和 Authorization Code 的混合流。

需要注意的一件事情是，Implicit 流和混合流都有潜在的安全风险，因为访问令牌可能会通过 Web 服务器日志和浏览器历史记录泄露。对于访问令牌，您可以使用简短的过期时间来缓解这个问题。

如需了解更多详细信息，请参阅 OpenID Connect 规格中的 Implicit 流。

根据当前的 OAuth 2.0 安全最佳实践，不应使用这个流。此流已从未来的 OAuth 2.1 规范 中删除。

2.2.3. 资源所有者密码凭证

在 Red Hat build of Keycloak 中，资源所有者密码凭证（称为 Direct Grant ）允许交换令牌的用户凭证。根据当前的 OAuth 2.0 安全最佳实践，不应使用这个流，首选使用 第 2.2.5 节 “设备授权” 或 第 2.2.1 节 “授权代码” 等替代方法。

使用这个流的限制包括：

- 用户凭证公开给应用程序
- 应用程序需要登录页面

- 应用程序需要了解身份验证方案
- 对身份验证流的更改需要更改应用程序
- 不支持身份代理或社交登录
- 不支持流（用户自助注册、所需操作等）

与此流程相关的安全问题包括：

- 涉及多个红帽构建的 Keycloak 处理凭证
- 增加了可能会发生凭证泄漏的问题
- 创建一个生态系统，用户信任另一个应用程序以输入其凭证，而不是红帽构建的 Keycloak

要使客户端被允许使用 Resource Owner Password Credentials grant，客户端必须启用 **Direct Access Grants Enabled** 选项。

此流不包括在 OpenID Connect 中，而是 OAuth 2.0 规范的一部分。它将从未来的 OAuth 2.1 规范 中删除。

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

2.2.3.1. 使用 CURL 的示例

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

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

2.2.4. 客户端凭证

当客户端（应用程序和服务）希望代表自己获取访问权限时，会使用客户端凭证，而不是代表用户获得访问权限。例如，这些凭据对于一般对系统而非特定用户应用更改的后台服务非常有用。

红帽构建的 Keycloak 支持客户端通过 secret 或公钥/私钥进行身份验证。

此流不包括在 OpenID Connect 中，而是 OAuth 2.0 规范的一部分。

如需了解更多详细信息，请参阅 OAuth 2.0 规格中的 客户端 凭证授予一章。

2.2.5. 设备授权

在互联网连接的设备中运行的客户端使用设备授权授予功能有限，或者缺少合适的浏览器。

1. 红帽构建的 Keycloak 的应用程序请求提供设备代码和用户代码。
2. 红帽构建的 Keycloak 创建一个设备代码和用户代码。

3. 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。
4. 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。
5. 应用程序会重复轮询红帽构建的 Keycloak，直到红帽构建的 Keycloak 完成用户授权。
6. 如果完成用户身份验证，应用程序会获取设备代码。
7. 应用程序使用设备代码及其凭证从红帽构建的 Keycloak 获取访问令牌、刷新令牌和 ID 令牌。

如需了解更多详细信息，请参阅 OAuth 2.0 设备授权规格。

2.2.6. 客户端初始频道身份验证授予

客户端初始通道身份验证授予由希望通过直接与 OpenID 提供程序通信来发起身份验证流的客户端使用，而无需通过 OAuth 2.0 的授权代码授权代码进行重定向。

来自红帽构建的 Keycloak 的客户端请求一个 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。

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。

如果客户端使用 **ping** 模式，则不需要重复轮询令牌端点，但可以等待红帽构建的 Keycloak 发送到指定的客户端通知端点。客户端通知端点可在红帽构建的 Keycloak 管理控制台中配置。客户端通知端点的合同详情请参考 CIBA 规格。

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

另请参阅 Red Hat build of Keycloak 文档的其他位置，如本指南的 backchannel Authentication Endpoint 和 Server Administration Guide 中的 Client Initiated Backchannel Authentication Grant 部分 。有关 FAPI CIBA 合规性的详情，请查看 本指南的 FAPI 部分。

2.3. 红帽构建的 KEYCLOAK JAVA 适配器

2.3.1. Red Hat JBoss Enterprise Application Platform

红帽构建的 Keycloak 不包括 Red Hat JBoss Enterprise Application Platform 的任何适配器。但是，对于部署到 Red Hat JBoss Enterprise Application Platform 的现有应用程序有替代方案。

2.3.1.1. 8.0 beta

Red Hat Enterprise Application Platform 8.0 Beta 通过 Elytron OIDC 客户端子系统提供原生 OpenID Connect 客户端。

如需更多信息，请参阅 Red Hat JBoss Enterprise Application Platform 文档 。

2.3.1.2. 6.4 和 7.x

部署到 Red Hat JBoss Enterprise Application Platform 6.4 和 7.x 的现有应用程序可以使用 Red Hat Single Sign-On 7.6 中的适配器，与红帽构建的 Keycloak 服务器结合使用。

如需更多信息，请参阅 Red Hat Single Sign-On 文档 。

2.3.2. Spring Boot adapter

红帽构建的 Keycloak 不包括 Spring Boot 的任何适配器。但是，使用 Spring Boot 构建的现有应用程序有替代方案。

Spring Security 为 OAuth 2 和 OpenID Connect 提供全面的支持。如需更多信息，请参阅 Spring 安全文档。

另外，对于 Spring Boot 2.x，Red Hat Single Sign-On 7.6 的 Spring Boot 适配器可与 Red Hat build of Keycloak 服务器结合使用。如需更多信息，请参阅 Red Hat Single Sign-On 文档 。

2.4. 红帽构建的 KEYCLOAK JAVASCRIPT 适配器

红帽构建的 Keycloak 附带一个名为 **keycloak-js** 的客户端 JavaScript 库，可用于保护 Web 应用程序。适配器还包括对 Cordova 应用程序的内置支持。

2.4.1. 安装

适配器以多种方式发布，但我们建议您从 NPM 安装 **keycloak-js** 软件包：

```
npm install keycloak-js
```

另外，该库可以直接从位于 **/js/keycloak.js** 的 Keycloak 服务器的红帽构建中检索，也可以作为 ZIP 存档分发。但是，我们考虑从 Keycloak 服务器直接包含适配器已被弃用，这个功能可能会在以后被删除。

2.4.2. 红帽构建的 Keycloak 服务器配置

要考虑使用客户端侧应用的一个重要事项是客户端必须是公共客户端，因为无法将客户端凭据存储在客户端侧应用中。考虑到这一点，请确保您为客户端配置的重定向 URI 正确且尽量具体。

要使用适配器，请在红帽构建的 Keycloak 管理控制台中为应用程序创建一个客户端。通过将客户端身份验证切换为 Off 在 **Capability 配置**页**面中使客户端**变为 **Off**。

您还需要配置 Valid Redirect URI 和 Web Origins。尽可能具体，因为无法这样做，可能会导致安全漏洞。

2.4.3. 使用适配器

以下示例演示了如何初始化适配器。确保将传递给 **Keycloak** 构造器的选项替换为**您配置的客户端。**

```
import Keycloak from 'keycloak-js';

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

try {
    const authenticated = await keycloak.init();
    console.log(`User is ${authenticated ? 'authenticated' : 'not authenticated'}`);
} catch (error) {
    console.error('Failed to initialize adapter:', error);
}
```

要进行身份验证，您需要调用 登录 功能。有两个选项以使适配器自动验证。您可以将 login-required 或 check-sso 传递给 init （） 函数。

- 如果用户登录到红帽构建的 Keycloak，则 login-required 验证客户端，如果用户没有登录，则会显示登录页面。
- check-sso 仅在用户已经登录时才验证客户端。如果用户没有登录，浏览器会被重定向到应用程序，并保持未经身份验证的。

您可以配置一个 *静默的* check-sso 选项。启用这个功能后，您的浏览器不会对红帽构建的 Keycloak 服务器执行完全重定向到您的应用程序，但此操作将在隐藏的 iframe 中执行。因此，您的应用程序资源只会由浏览器加载并解析一次，即在应用程序被初始化时，而不是在从红帽构建的 Keycloak 重新到应用程序后再次解析。对于 SPAs (Single Page Applications)，此方法特别有用。

要启用 *silent* check-sso，您可以在 init 方法中提供silentCheckSsoRedirectUri 属性。确保此 URI 是应用程序中的有效端点；它必须配置为红帽构建的 Keycloak 管理控制台中的客户端的有效重定向：

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

在成功检查身份验证状态并从红帽 Keycloak 服务器检索令牌后，silent check-sso redirect uri 会加载到 iframe 中。除了将接收的令牌发送到主应用程序外，它没有其他任务，应该只类似如下：

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

请记住，此页面必须由您的应用程序在 silentCheckSsoRedirectUri 中的指定位置提供，*且不是* 适配器的一部分。

警告

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

要在 Load to login-required 上启用 login-required 设置为 login-required，并传递给 init 方法：

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

在经过身份验证后，应用程序可以通过在 Authorization 标头中包含 bearer 令牌来向红帽构建的 Keycloak 安全的 RESTful 服务发出请求。例如：

```
async function fetchUsers() {
    const response = await fetch('/api/users', {
        headers: {
            accept: 'application/json',
            authorization: `Bearer ${keycloak.token}`
        }
    });

    return response.json();
}
```

请记住，访问令牌默认具有较短的过期时间，因此您可能需要在发送请求前刷新访问令牌。您可以通过调用 updateToken （） 方法来刷新此令牌。这个方法会返回 Promise，它可让您仅在成功刷新令牌时轻松调用该服务，并在用户未刷新时向用户显示一个错误。例如：

```
try {
    await keycloak.updateToken(30);
} catch (error) {
    console.error('Failed to refresh token:', error);
}

const users = await fetchUsers();
```

注意

访问和刷新令牌都存储在内存中，且不会保留在任何类型的存储中。因此，这些令牌永远不会被保留，以防止劫持攻击。

2.4.4. 会话状态 iframe

默认情况下，适配器会创建一个隐藏的 iframe，用于检测是否发生 Single-Sign Out。这个 iframe 不需要任何网络流量。相反，通过查看特殊的状态 cookie 来检索状态。通过在传递给 init （） 方法的选项中设置 checkLoginIframe: false 来禁用此功能。

您不应该直接查看这个 Cookie。其格式可能会改变，它还与红帽构建的 Keycloak 服务器的 URL 关联，而不是您的应用程序。

警告

会话状态 iframe 功能在某些现代浏览器中受到限制。请参阅 带有跟踪保护部分的现代浏览器。

2.4.5. 隐式和混合流

默认情况下，适配器使用 Authorization Code 流。

在这个版本中，红帽构建的 Keycloak 服务器会将授权代码而不是身份验证令牌返回给应用程序。JavaScript 适配器在浏览器重定向到应用程序后交换访问令牌的代码和刷新令牌。

红帽构建的 Keycloak 还支持 Implicit 流，在通过红帽构建的 Keycloak 身份验证成功后立即发送访问令牌。此流的性能可能比标准流更强，因为不存在额外的请求来交换令牌的代码，但在访问令牌过期时有影响。

但是，在 URL 片段中发送访问令牌可能是一个安全漏洞。例如，令牌可能会通过 Web 服务器日志或浏览器历史记录泄漏。

要启用隐式流，您可以在红帽构建的 Keycloak 管理控制台中为客户端启用 Implicit Flow Enabled 标志。您还将参数 流 传递值为 implicit 到 init 方法：

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

请注意，只提供访问令牌，且没有刷新令牌。这意味着，一旦访问令牌已过期，应用程序必须重新重定向到红帽构建的 Keycloak，以获取新的访问令牌。

红帽构建的 Keycloak 还支持 混合 流。

此流要求客户端在管理控制台中 同时启用 了标准流和 Implicit 流。然后，红帽构建的 Keycloak 服务器会将代码和令牌发送到您的应用程序。访问令牌可以立即使用，同时可以交换代码来访问和刷新令牌。与隐式流类似，混合流对性能很好，因为访问令牌会立即可用。但是，令牌仍然在 URL 中发送，前面提到的安全漏洞可能仍然适用。

混合流中的一个优点是刷新令牌可供应用程序使用。

对于混合流，您需要将带有值 hybrid 参数 流 传递给 init 方法：

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

2.4.6. Cordova 的混合应用程序

红帽构建的 Keycloak 支持使用 Apache Cordova 开发的混合移动应用程序。适配器具有两种模式：cordova 和 cordova-native ：

默认为 cordova，如果没有显式配置适配器类型，且存在 window.cordova，则适配器会自动选择它。登录时，它会打开一个 InApp Browser，允许用户与红帽构建的 Keycloak 交互，随后通过重定向到 http://localhost 来返回到应用程序。由于此行为，您可以在管理控制台的客户端配置部分中将此 URL 列为有效的 redirect-uri。

虽然此模式易于设置，但也有一些缺点：

- InApp-Browser 是嵌入在应用程序中的浏览器，不是电话的默认浏览器。因此，它具有不同的设置，并且存储的凭据将不可用。
- InApp-Browser 可能也很慢，特别是在渲染更复杂的主题时。
- 在使用此模式之前，需要注意一些安全性问题，例如，应用程序有可能获得用户凭证的访问权限，因为它对浏览器的渲染登录页面具有完全控制，因此不允许它在不信任的应用中使用。

另一种模式是'cordova-native'，它采用不同的方法。它使用系统的浏览器打开登录页面。用户通过身份验证后，浏览器将使用特殊 URL 重新重定向到应用。在这里，红帽构建的 Keycloak 适配器可以通过从 URL 读取代码或令牌来完成登录。

您可以通过将 adapter 类型 cordova-native 传递给 init （） 方法来激活原生模式：

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

这个适配器需要额外的插件：

- Cordova-plugin-browsertab ：允许应用程序在系统的浏览器中打开 webpages
- Cordova-plugin-deeplinks ：允许浏览器通过特殊 URL 重定向到应用程序

链接到应用程序的技术详情在每个平台上有所不同，需要特殊设置。具体步骤请查看 deeplinks 插件文档中的 Android 和 iOS 部分。

开放应用程序有不同类型的链接：* 自定义方案，如 myapp://login 或 android-app://com.example.myapp/https/example.com/login * Universal Links (iOS)）/ Deep Links (Android)。虽然前者更容易设置并倾向于更可靠工作，但后者会提供额外的安全性，因为它们是唯一的，且只有域的所有者可以注册它们。custom-URLs 在 iOS 上已弃用。为获得最佳可靠性，我们建议您将通用链接与使用 custom-url 链接的回退站点结合使用。

另外，我们建议以下步骤提高与适配器的兼容性：

- iOS 上的通用链接看似更可靠工作，并将 response-mode 设置为 query
- 要防止 Android 在重定向时打开应用程序的新实例，请将以下代码片段添加到 config.xml 中：

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

2.4.7. 自定义适配器

在某些情况下，您可能需要在不受默认支持的环境中运行适配器，如 Capacitor。要在这些环境中使用 JavasScript 客户端，您可以传递自定义适配器。例如，第三方库可以提供这样的适配器，以便可以可靠地运行适配器：

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

const keycloak = new Keycloak();

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

这个特定软件包不存在，但它是一个很好的例子，它是如何传递给客户端的适配器。

也可以自行创建适配器，您必须实施 KeycloakAdapter 接口中描述的方法。例如，以下 TypeScript 代码确保所有方法都被正确实现：

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

// Implement the 'KeycloakAdapter' interface so that all required methods are guaranteed to be present.
const MyCustomAdapter: KeycloakAdapter = {
   login(options) {
```

```
        // Write your own implementation here.
    }

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

const keycloak = new Keycloak();

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

通过省略类型信息，您也可以在没有 TypeScript 的情况下执行此操作，但确保正确实现接口将完全保留给您。

2.4.8. 带有跟踪保护的现代浏览器

在最新版本的某些浏览器中，会应用各种 Cookie 策略，以防止由第三方跟踪用户，如 Chrome 中的 SameSite 或完全阻止第三方 Cookie。这些策略可能会变得更严格并被其他浏览器随时间采用。最终，第三方上下文中的 Cookie 可能会完全不受支持，并由浏览器阻止。因此，受影响的适配器功能可能会最终被弃用。

适配器依赖于 Session Status iframe、*静默* check-sso 以及部分常规（非静默） check-sso。这些功能有有限的功能，或者根据浏览器对 Cookie 的限制而完全禁用。适配器会尝试检测此设置并相应地响应。

2.4.8.1. 带有 "SameSite=Lax by Default" 策略的浏览器

如果在红帽构建的 Keycloak 端以及应用程序端配置了 SSL / TLS 连接，则所有功能都被支持。例如：从版本 84 开始，Chrome 会受到影响。

2.4.8.2. 带有块第三方 Cookie 的浏览器

不支持会话状态 iframe，如果适配器检测到这样的浏览器行为，则会自动禁用。这意味着适配器无法将会话 Cookie 用于 Single Sign-Out 检测，且必须完全依赖令牌。因此，当用户在另一个窗口中登录时，使用适配器的应用程序不会登出，直到应用程序尝试刷新访问令牌为止。因此，请考虑将 Access Token Lifespan 设置为相对较短的时间，以便尽快检测到注销。如需了解更多详细信息，请参阅 会话和令牌超时。

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

常规 check-sso 也会受到影响。由于 Session Status iframe 不受支持，因此在适配器初始化以检查用户的登录状态时，必须进行到红帽构建的 Keycloak 的额外重定向。当 iframe 用于告知用户是否登录时，此检查与标准行为不同，并且仅在用户注销时才执行重定向。

受影响的浏览器是从版本 13.1 开始的 Safari 示例。

2.4.9. API 参考

2.4.9.1. Constructor

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

2.4.9.2. Properties

已验证

如果用户通过身份验证，则为 true，否则为 false。

token

可以在请求中的 Authorization 标头发送到服务的 base64 编码令牌。

tokenParsed

解析的令牌作为一个 JavaScript 对象。

subject

用户 ID。

idToken

base64 编码的 ID 令牌。

idTokenParsed

解析的 id 令牌作为 JavaScript 对象。

realmAccess

与令牌关联的 realm 角色。

resourceAccess

与令牌关联的资源角色。

refreshToken

base64 编码的刷新令牌，可用于检索新令牌。

refreshTokenParsed

解析的刷新令牌作为 JavaScript 对象。

timeSkew

浏览器时间和红帽构建的 Keycloak 服务器之间的预计时间差异（以秒为单位）。这个值只是一个估算，但在确定令牌是否过期时足够准确。

responseMode

在 init 中传递的响应模式（默认值为 片段）。

流

在 init 中传递的流程。

adapter

允许您覆盖重定向和其他浏览器相关功能的方式由该库处理。可用选项：

- "default" - 库使用浏览器 api 进行重定向（这是默认设置）
- "Cordova" - 库将尝试使用 InAppBrowser cordova 插件来加载 keycloak 登录/注册页面（当库在 cordova 生态系统中工作）
- "Cordova-native" - 库尝试使用浏览器Tabs cordova 插件使用电话的系统浏览器打开登录和注册页面。这需要额外的设置才能重新重定向到应用程序（请参阅 第 2.4.6 节 “Cordova 的混合应用程序”）。
- "custom" - 允许您实施自定义适配器（仅适用于高级用例）

responseType

发送到带有登录请求的红帽构建的 Keycloak 的响应类型。这根据初始化过程中使用的流值决定，但可通过设置这个值来覆盖。

2.4.9.3. Methods

init (options)

调用以初始化适配器。

选项是一个对象，其中：

- useNonce - 添加加密非ce 以验证身份验证响应是否与请求匹配（默认为 true）。
- onLoad - 指定加载时要执行的操作。支持的值是 login-required 或 check-sso。
- silentCheckSsoRedirectUri - 如果 onLoad 设为 'check-sso'，则设置重定向 uri for silent authentication check。
- silentCheckSsoFallback - 当浏览器不支持 *静默* check-sso 时（默认为 true），启用回到普通的 check-sso。
- token - 为令牌设置初始值。
- refreshToken - 为刷新令牌设置初始值。
- idToken - 为 id 令牌设置一个初始值（仅适用于 token 或 refreshToken）。
- Scope - 将默认范围参数设置为红帽构建的 Keycloak 登录端点。使用以空格分隔的范围列表。它们通常引用 在特定客户端上定义的客户端范围。请注意，范围 openid 将始终添加到适配器的范围列表中。例如，如果您输入范围选项 地址电话，则对红帽构建的 Keycloak 的请求将包含 scope=openid 地址电话。请注意，如果 login （） 选项明确指定范围，则此处指定的默认范围会被覆盖。
- timeSkew - 以秒为单位为 skew 和 Red Hat build of Keycloak 服务器设置初始值（仅与 token 或 refreshToken 一起）。

- checkLoginIframe - 设置为启用/禁用监控登录状态（默认为 true）。
- checkLoginIframeInterval - 设置检查登录状态的时间间隔（默认为 5 秒）。
- responseMode - 在登录请求时设置 OpenID Connect 响应模式向红帽构建的 Keycloak 服务器发送。有效值为 query 或 fragment。默认值为 fragment，这意味着，在成功身份验证后，红帽构建 Keycloak 重定向到 JavaScript 应用程序，并在 URL 片段中添加 OpenID Connect 参数。这通常更安全，建议使用它而不是 query。
- flow - 设置 OpenID Connect 流。有效值为 标准、隐式 或混合。
- enableLogging - 启用从 Keycloak 到控制台的日志记录信息（默认为 false）。
- pkceMethod - 要使用的概念验证代码交换(PKCE)的方法。配置这个值可启用 PKCE 机制。可用选项：
  - "S256" - 基于 SHA256 的 PKCE 方法（默认）
  - false - 禁用 PKCE。
- acrValues - 生成 acr_values 参数，该参数引用身份验证上下文类引用，并允许客户端声明所需的保证级别要求，如验证机制。请参阅 OpenID Connect MODRNA Authentication Profile 1.0 中的第 4 节 cr_values 请求值和保证级别。
- messageReceiveTimeout - 以毫秒为单位设置一个超时，用于等待 Keycloak 服务器的消息响应。这用于，例如，在第三方 Cookie 检查期间等待消息。默认值为 10000。
- locale - 当Load 为 'login-required' 时，设置 'ui_locales' 查询参数，以遵守 OIDC 1.0 规范的第 3.1.2.1 部分。

返回初始化完成后解决的承诺。

login (options)

重定向至登录表单。

选项是一个可选对象，其中：

- redirecturi - 指定要在登录后重定向到的 uri。
- prompt - 此参数允许在红帽构建的 Keycloak 服务器端稍微自定义登录流。例如，在值登录时，强制显示 登录屏幕。
- maxAge - 仅在用户已经验证时使用。指定发生用户身份验证后的最长时间。如果用户已经进行身份验证的时间超过 maxAge 的时间，则忽略 SSO，并且需要再次重新进行身份验证。
- loginHint - 用于在登录表单上预先填充 username/email 字段。
- scope - 覆盖在 init 中配置的范围，其范围使用此特定登录的值不同。
- idpHint - 告知红帽构建的 Keycloak 以跳过显示登录页面，并自动重定向到指定的身份提供程序。如需更多信息，请参阅 身份提供程序文档。
- acr - 包含有关 acr 声明的信息，该声明将在 声明 参数发送到红帽构建的 Keycloak 服务器。典型的用法是进行单步身份验证。使用 { 值的示例： ["silver", "gold"], essential: true }.如需了解更多详细信息，请参阅 OpenID Connect 规格和步骤 身份验证文档。
- acrValues - 生成 acr_values 参数，该参数引用身份验证上下文类引用，并允许客户端声明所需的保证级别要求，如验证机制。请参阅 OpenID Connect MODRNA Authentication Profile 1.0 中的第 4 节 cr_values 请求值和保证级别。
- action - 如果该值是 注册，则用户将重定向到注册页面。如需了解更多详细信息，请参阅客户端请求的注册部分。如果值为 UPDATE_PASSWORD 或另一个支持的必要操作，该用户将重定向到 reset password 页面或其他 required 操作页面。但是，如果用户未通过身份验证，该

用户将发送到登录页面并在身份验证后重定向。如需了解更多详细信息，请参阅应用程序初始操作部分。

- locale - 设置"ui_locales"查询参数，使其符合 OIDC 1.0 规范的第 3.1.2.1 部分。

- cordovaOptions - 指定传递给 Cordova in-app-browser （如果适用）的参数。选择 hidden 和 location 不受这些参数的影响。所有可用选项均在 https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-inappbrowser/ 中定义。使用示例 ：{ zoom: "no", hardwareback: "yes" };

createLoginUrl(options)

返回 URL 以登录表单。

选项是一个可选对象，它支持与函数 登录 相同的选项。

logout (options)

重定向 以注销。

选项是一个对象，其中：

- redirecturi - 指定要在退出后重定向到的 uri。

createLogoutUrl(options)

返回 URL 以注销用户。

选项是一个对象，其中：

- redirecturi - 指定要在退出后重定向到的 uri。

register (options)

重定向到注册表单。使用选项操作登录的快捷方式 = 'register'

选项与登录方法相同，但 'action' 被设置为 'register'

createRegisterUrl(options)

返回 url 以注册页面。使用选项 action = 'register' 的 createLoginUrl 的快捷方式

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

accountManagement()

重定向到帐户控制台。

createAccountUrl(options)

将 URL 返回到帐户控制台。

选项是一个对象，其中：

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

hasRealmRole(role)

如果令牌具有给定域角色，则返回 true。

hasResourceRole (role, resource)

如果令牌具有资源的给定角色（如果没有指定 clientId，则返回 true 是可选的）。

loadUserProfile()

加载 users 配置集。

返回与配置集解析的承诺。

例如：

```
try {
    const profile = await keycloak.loadUserProfile();
    console.log('Retrieved user profile:', profile);
} catch (error) {
    console.error('Failed to load user profile:', error);
}
```

isTokenExpired(minValidity)

如果令牌在过期前少于 minValidity 秒（使用minValidity 是可选的，则返回 true）。

updateToken(minValidity)

如果令牌在 minValidity 秒内过期(minValidity 是可选的，如果没有指定 5)，则刷新令牌。如果 -1 作为 minValidity 传递，则令牌将被强制刷新。如果启用了会话状态 iframe，也会检查会话状态。

返回一个使用布尔值解析的承诺，指示令牌是否已刷新。

例如：

```
try {
    const refreshed = await keycloak.updateToken(5);
    console.log(refreshed ? 'Token was refreshed' : 'Token is still valid');
} catch (error) {
    console.error('Failed to refresh the token:', error);
}
```

clearToken()

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

调用此功能会导致调用 onAuthLogout 回调监听程序。

2.4.9.4. 回调事件

适配器支持为特定事件设置回调监听程序。请记住，必须在调用 init （） 方法之前设置它们。

例如：

```
keycloak.onAuthSuccess = () => console.log('Authenticated!');
```

可用的事件有：

- onReady (authenticated) - 当适配器初始化时调用。
- onAuthSuccess - 当用户成功验证时调用。
- onAuthError - 如果身份验证过程中出现错误，则调用。
- onAuthRefreshSuccess - 当令牌被刷新时调用。

- onAuthRefreshError - 如果在尝试刷新令牌时出现错误。
- onAuthLogout - 如果用户已注销（仅在启用会话状态 iframe 或 Cordova 模式时调用）。
- onTokenExpired - 当访问令牌过期时调用。如果一个刷新令牌可用，可以使用 updateToken 刷新令牌，或者在没有（即隐式流）的情况下刷新令牌，您可以重定向到登录屏幕来获取新的访问令牌。

2.5. 红帽构建的 KEYCLOAK NODE.JS 适配器

Red Hat build of Keycloak 提供了一个在 Connect 上构建的 Node.js 适配器，以保护服务器端 JavaScript 应用 - 目标足够灵活，以便与 Express.js 等框架集成。

要使用 Node.js 适配器，您必须首先在红帽构建的 Keycloak 管理控制台中为应用程序创建一个客户端。适配器支持 public, confidential, 和 bearer-only 访问类型。哪个选项取决于用例场景。

创建客户端后，点右上角的 Action，然后选择 Download adapter config。对于 Format，请选择 *Keycloak OIDC JSON 并点 Download。下载的 keycloak.json 文件位于项目的根目录中。

2.5.1. 安装

假设您已经安装了 Node.js，请为您的应用程序创建一个文件夹：

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

使用 npm init 命令为应用程序创建 package.json。现在，在依赖项列表中添加红帽构建的 Keycloak 连接适配器：

```
"dependencies": {
  "keycloak-connect": "file:keycloak-connect-24.0.0+redhat-00001.tgz"
}
```

2.5.2. 使用方法

实例化 Keycloak 类

Keycloak 类为配置和与应用程序集成提供了一个中央点。最简单的创建涉及任何参数。

在项目的根目录中，创建一个名为 server.js 的文件并添加以下代码：

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

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

安装 express-session 依赖项：

```
npm install express-session
```

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

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

现在，我们可以通过以下命令运行服务器：

```
npm run start
```

默认情况下，这将找到一个名为 keycloak.json 的文件，以及应用程序的主可执行文件（在根文件夹上），以初始化红帽构建的 Keycloak 特定设置，如公钥、域名、各种 URL。

在这种情况下，红帽构建的 Keycloak 部署需要访问红帽构建的 Keycloak 管理控制台。

请访问有关如何使用 Podman 或 Docker部署红帽构建的 Keycloak 管理控制台的链接

现在，我们已准备好通过访问 Red Hat build of Keycloak Admin Console → client (left sidebar)→ 选择您的 client → Installation → Format Option → Keycloak OIDC JSON → Download 。

将下载的文件粘贴到项目的根目录上。

使用此方法实例化会导致使用所有合理的默认值。另外，也可以提供配置对象，而不是 keycloak.json

文件：

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

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

应用程序也可以使用以下方法将用户重定向到首选身份提供程序：

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

配置 Web 会话存储

如果要使用 Web 会话管理服务器端状态进行身份验证，则需要使用至少 存储 参数初始化 Keycloak (...)，从而传递 express-session 使用的实际会话存储。

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

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

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

传递自定义范围值

默认情况下，范围值 openid 作为查询参数传递给红帽构建的 Keycloak 登录 URL，但您可以添加额外的自定义值：

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

2.5.3. 安装中间件

实例化后，将中间件安装到支持连接的应用程序中：

要做到这一点，必须首先安装 Express：

```
npm install express
```

然后，需要我们的项目中的 Express，如下所示：

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

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

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

最后，我们通过将以下代码添加到 main.js，将服务器设置为在端口 3000 上侦听 HTTP 请求：

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

2.5.4. 配置代理配置

如果应用程序在终止 SSL connection Express 的代理后面运行，则必须根据 代理后面的表达 配置 SSL connection Express。使用不正确的代理配置可能会导致生成无效的重定向 URI。

配置示例：

```
const app = express();

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

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

2.5.5. 保护资源

简单身份验证

要强制用户在访问资源前必须经过身份验证，只需使用 keycloak.protect （） 的 no-argument 版本：

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

基于角色的授权

使用当前应用程序的应用程序角色来保护资源：

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

使用不同应用程序的应用程序角色来 保护资源 ：

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

使用 realm 角色保护资源：

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

基于资源的授权

基于资源的授权允许您根据 Keycloak 中定义的一组策略来保护资源及其特定方法/操作，从而从应用程序外部授权。这可以通过公开 keycloak.enforcer 方法来实现，您可以使用该方法来保护 resources metric

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

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

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

如果将 response_mode 设置为 令牌，则会代表发送到应用程序的 bearer 令牌代表从服务器获取权限。在这种情况下，Keycloak 会发布一个新的访问令牌，其权限由服务器授予。如果服务器没有响应具有预期权限的令牌，则请求将被拒绝。使用此模式时，您应能够从请求获取令牌，如下所示：

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

    // show user profile
});
```

当应用程序使用会话且您想要缓存来自服务器的之前决策时，首选此模式，并自动处理刷新令牌。此模式对于充当客户端和资源服务器的应用程序特别有用。

如果 response_mode 设为 permissions （默认模式），服务器仅返回已授予权限的列表，而不发出新的访问令牌。除了没有发布新令牌外，此方法还会公开服务器通过 请求 授予的权限，如下所示：

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

    // show user profile
});
```

无论使用中的 response_mode 是什么，keycloak.enforcer 方法都会首先尝试检查发送到应用程序的 bearer 令牌中的权限。如果 bearer 令牌已经执行了预期的权限，则不需要与服务器交互来获取决策。当您的客户端能够在访问受保护的资源前从具有预期权限的服务器获取访问令牌时，这特别有用，因此他们可以使用 Keycloak 授权服务（如增量授权服务）提供的一些功能，并避免对服务器的额外的请求。

默认情况下，策略实施器将使用定义为应用程序的 client_id （例如，通过 keycloak.json）来引用支持 Keycloak Authorization Services 的 Keycloak 中的客户端。在这种情况下，客户端不能是公共的，它实际上是一个资源服务器。

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

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

建议您在 Keycloak 中使用不同的客户端来代表您的前端和后端。

如果您使用 Keycloak 授权服务启用保护的应用程序，并在 keycloak.json 中定义客户端凭证，您可以将其他声明推送到服务器，并使其可用于您的策略以做出决策。为此，您可以定义一个 claims 配置选项，它需要一个会返回一个带有您要推送的声明的 JSON 的函数：

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

有关如何配置 Keycloak 以保护应用程序资源的详情，请参考 授权服务指南。

高级授权

要根据 URL 本身的部分来保护资源，假设每个部分都有一个角色：

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

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

高级登录配置：

默认情况下，除非您的客户端为 bearer-only，否则所有未授权的请求都会被重定向到红帽构建的 Keycloak 登录页面。但是，机密或公共客户端可以同时托管可浏览和 API 端点。要防止对未经身份验证的 API 请求进行重定向，并返回 HTTP 401，您可以覆盖 redirectToLogin 功能。

例如，此覆盖检查 URL 是否包含 /api/ 并禁用登录重定向：

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

2.5.6. 其他 URL

显式用户触发的退出

默认情况下，中间件会捕获对 /logout 的调用，以便通过以 Keycloak 为中心的注销工作流向用户发送。这可以通过在 middleware() 调用中指定 logout 配置参数来更改：

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

当调用 user-triggered logout 时，可以传递查询参数 redirect_url ：

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

然后，此参数被用作 OIDC 注销端点的重定向 URL，用户将重定向到

https://example.com/logged/out。

红帽构建的 Keycloak 管理回调

另外，中间件还支持来自红帽构建的 Keycloak 控制台的回调，以注销单个会话或所有会话。默认情况下，这些类型的管理回调相对于 / 的根 URL，但可以通过向 middleware （） 调用提供 admin 参数来更改：

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

2.5.7. 完成示例

使用 Node.js 适配器使用的完整示例可在 Node.js 的 Keycloak 快速入门中找到

2.6. 金融级 API (FAPI)支持

红帽构建的 Keycloak 可让管理员更容易确保其客户端符合这些规格：

- 财务级 API 安全配置文件 1.0 - 第 1 部分：基础
- 财务级 API 安全配置文件 1.0 - 第 2 部分：高级
- 财务级 API：客户端发起的后端通道身份验证配置文件 (FAPI CIBA)
- FAPI 2.0 安全配置文件(Draft)
- FAPI 2.0 消息签名(Draft)

此合规性意味着红帽构建的 Keycloak 服务器将验证授权服务器的要求，这在规格中提到。红帽构建的 Keycloak 适配器对 FAPI 没有任何特定支持，因此客户端（应用程序）端所需的验证可能需要手动或通过其他第三方解决方案完成。

2.6.1. FAPI 客户端配置集

为确保您的客户端兼容 FAPI，您可以在域中配置客户端策略，如 服务器管理指南 中所述，并将它们

链接到 FAPI 支持的全局客户端配置文件，这些配置文件在每个域中自动可用。您可以根据需要客户端符合 FAPI 配置集，使用 fapi-1-baseline 或 fapi-1-advanced 配置集。您还可以使用配置文件 fapi-2-security-profile 或 fapi-2-message-signing 来遵守 FAPI 2 Draft 规范。

如果要使用 Pushed Authorization Request (PAR)，建议您的客户端使用 fapi-1-baseline 配置文件和 fapi-1-advanced 用于 PAR 请求。具体来说，fapi-1-baseline 配置集包含 pkce-enforcer executor，这样可确保客户端使用带有安全 S256 算法的 PKCE。FAPI 高级客户端不需要此功能，除非它们使用 PAR 请求。

如果要以 FAPI 兼容方式使用 CIBA，请确保您的客户端同时使用 fapi-1-advanced 和 fapi-ciba 客户端配置集。需要使用 fapi-1-advanced 配置集，或其他包含请求的 executors 的客户端配置文件，因为 fapi-ciba 配置集仅包含 CIBA 特定的 executors。当强制实施 FAPI CIBA 规范的要求时，需要更多要求，如强制机密客户端或证书绑定访问令牌。

2.6.2. Open Finance Brasil financial-grade API Security Profile

红帽构建的 Keycloak 与 Open Finance Brasil financial-grade API Security Profile 1.0 实施器 Draft 3 兼容。与 FAPI 1 高级 规格相比，这个要求严格严格，因此可能需要以更严格的方式配置客户端策略 来强制实施某些要求。特别是：

- 如果您的客户端没有使用 PAR，请确保它使用加密的 OIDC 请求对象。这可以通过使用带有启用了 Encryption 的 secure-request-object executor 的客户端配置文件来实现。
- 确保用于 JWS，客户端使用 PS256 算法。对于 JWE，客户端应使用带有 A256GCM 的 RSA-OAEP。这可能需要在这些算法 的所有客户端设置 中设置。

2.6.3. 澳大利亚消费者数据权利(CDR)安全配置文件

红帽构建的 Keycloak 与 Australia Consumer Data Right Security Profile 兼容。

如果要应用 Australia CDR 安全配置集，您需要使用 fapi-1-advanced 配置集，因为 Australia CDR 安全配置集基于 FAPI 1.0 高级安全配置文件。如果您的客户端也适用 PAR，请确保客户端应用适用于代码交换的 RFC 7637 概念验证(PKCE)，因为 Australia CDR 安全配置集要求您在应用 PAR 时应用 PKCE。这可以通过使用带有 pkce-enforcer executor 的客户端配置集来实现。

2.6.4. TLS 注意事项

由于机密信息正在交换，所有交互都应与 TLS (HTTPS)加密。此外，FAPI 规格和使用的 TLS 协议版本有一些要求。要匹配这些要求，您可以考虑配置允许的密码。可以通过设置 https-protocols 和 https-

cipher-suites 选项来完成此配置。红帽构建的 Keycloak 默认使用 TLSv1.3，因此可能不需要更改默认设置。但是，如果您需要因为某种原因回退到较低 TLS 版本，可能需要调整密码。如需了解更多详细信息，请参阅配置 TLS 章节。

2.7. OAUTH 2.1 支持

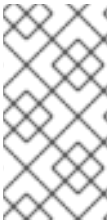
红帽构建的 Keycloak 可让管理员更容易确保其客户端符合这些规格：

- OAuth 2.1 授权框架 - 草案规格

此合规性意味着红帽构建的 Keycloak 服务器将验证授权服务器的要求，这在规格中提到。红帽构建的 Keycloak 适配器对 OAuth 2.1 没有任何特定支持，因此客户端（应用程序）端所需的验证可能需要手动或通过其他第三方解决方案完成。

2.7.1. OAuth 2.1 客户端配置集

为确保您的客户端兼容 OAuth 2.1，您可以在域中配置客户端策略，如 服务器管理指南 中所述，并将它们链接到 OAuth 2.1 支持的全局客户端配置文件，这些配置文件在每个域中自动可用。您可以将 oauth-2-1-for-confidential-client 配置集用于机密客户端，或将 oauth-2-1-for-public-client 配置集用于公共客户端。



注意

OAuth 2.1 规范仍是一个草案，未来可能会改变。因此，红帽构建的 Keycloak 内置 OAuth 2.1 客户端配置集也可以改变。

注意

将 OAuth 2.1 配置集用于公共客户端时，建议按照 服务器管理指南中所述 使用 DPoP preview 功能，因为 DPoP 将访问令牌与客户端密钥对的公钥部分绑定在一起。这个绑定可防止攻击者使用 stolen 令牌。

2.8. 建议

本节介绍了使用 Red Hat build of Keycloak 保护应用程序时的一些建议。

2.8.1. 验证访问令牌

如果您需要手动验证红帽构建的 Keycloak 发布的访问令牌，您可以调用 Introspection Endpoint。这种方法的不足之处在于，您必须进行一个网络调用红帽 Keycloak 服务器的构建。如果您同时存在太多验证请求，则这可能会减慢服务器的速度，并可能会过载。红帽构建的 Keycloak 发布访问令牌是 JSON Web 令牌(JWT)，使用 JSON Web 签名(JWS) 进行数字签名和编码。由于以这种方式编码了它们，因此您可以使用发布域的公钥在本地验证访问令牌。您可以在验证代码中硬编码域的公钥，或使用嵌入在 JWS 中的密钥 ID (KID) 的证书端点 查找并缓存公钥。根据您编码的语言，有很多第三方库，它们可帮助您进行 JWS 验证。

2.8.2. 重定向 URI

在使用基于重定向的流时，请确保为您的客户端使用有效的重定向 uri。redirect uris 应尽量具体。这特别适用于客户端（公共客户端）应用程序。无法这样做可能会导致：

- 打开重定向 - 这样可让攻击者创建像来自您的域的欺骗链接
- 未授权条目 - 当用户已使用红帽构建的 Keycloak 进行身份验证时，攻击者可以使用公共客户端，其中重定向 uris 没有正确配置，以便在没有了解用户的情况下重定向用户访问权限

在用于 Web 应用的生产环境中，始终将 https 用于所有重定向 URI。不允许重定向到 http。

还有几个特殊的重定向 URI：

http://127.0.0.1

此重定向 URI 可用于原生应用，并允许原生应用在可用于获取授权代码的随机端口上创建 Web 服务器。这个 redirect uri 允许任何端口。请注意，每个 OAuth 2.0 用于原生 Apps，不建议使用 localhost，应该改为使用 IP 字面 127.0.0.1。

urn:ietf:wg:oauth:2.0:oob

如果无法在客户端（或浏览器不可用）中启动 Web 服务器，您可以使用特殊的 urn:ietf:wg:oauth:2.0:oob redirect uri。当使用此重定向 uri 时，Red Hat build of Keycloak 会在标题和页面的框中显示带有代码的页面。应用可以检测浏览器标题已更改，或者用户可以手动将代码复制并粘贴到应用中。使用这个重定向 uri，用户可以使用不同的设备来获取要粘贴至应用程序的代码。

第 3 章 使用 SAML 保护应用程序和服务

本节论述了如何使用红帽构建的 Keycloak 客户端适配器或通用 SAML 供应商库保护带有 SAML 的应用程序和服务。

3.1. 红帽构建的 KEYCLOAK JAVA 适配器

红帽构建的 Keycloak 附带 Java 应用程序的不同适配器。选择正确的适配器取决于目标平台。

3.1.1. Red Hat JBoss Enterprise Application Platform

3.1.1.1. 8.0 beta

Red Hat build of Keycloak 为 Red Hat Enterprise Application Platform 8.0 Beta 提供 SAML 适配器。但是，文档目前还不可用，并将在不久的将来添加。

3.1.1.2. 6.4 和 7.x

部署到 Red Hat JBoss Enterprise Application Platform 6.4 和 7.x 的现有应用程序可以使用 Red Hat Single Sign-On 7.6 中的适配器，与红帽构建的 Keycloak 服务器结合使用。

如需更多信息，请参阅 Red Hat Single Sign-On 文档。

第 4 章 将 DOCKER REGISTRY 配置为使用红帽构建的 KEYCLOAK

注意

Docker 身份验证默认为禁用。要启用 启用和禁用功能，请参阅 启用和禁用功能 章节。

本节论述了如何将 Docker registry 配置为使用红帽构建的 Keycloak 作为其身份验证服务器。

有关如何设置和配置 Docker registry 的更多信息，请参阅 Docker Registry 配置指南。

4.1. DOCKER REGISTRY 配置文件安装

对于具有更高级的 Docker registry 配置的用户，通常建议提供您自己的 registry 配置文件。红帽构建的 Keycloak Docker 供应商通过 *Registry 配置文件格式选项支持* 此机制。选择这个选项将生成类似如下的输出：

```
auth:
  token:
    realm: http://localhost:8080/realms/master/protocol/docker-v2/auth
    service: docker-test
    issuer: http://localhost:8080/realms/master
```

然后可将此输出复制到任何现有 registry 配置文件中。有关如何设置 文件或以 基本示例 开始的更多信息，请参阅 registry 配置文件规格。

警告

不要忘记为 rootcertbundle 字段配置红帽构建的 Keycloak 域公钥的位置。如果没有此参数，auth 配置将无法工作。

4.2. DOCKER REGISTRY 环境变量覆盖安装

通常，它适合对开发或 POC Docker registry 使用简单的环境变量覆盖。虽然通常不建议在生产环境中使用这个方法，但当一个需要快速和-dirty 的方式备份 registry 时，这很有用。只需使用客户端详情中

的 *Variable Override* Format 选项，输出应类似如下：

```
REGISTRY_AUTH_TOKEN_REALM: http://localhost:8080/realms/master/protocol/docker-v2/auth
REGISTRY_AUTH_TOKEN_SERVICE: docker-test
REGISTRY_AUTH_TOKEN_ISSUER: http://localhost:8080/realms/master
```

警告

不要忘记使用红帽构建的 Keycloak 域公钥的位置配置 REGISTRY_AUTH_TOKEN_ROOTCERTBUNDLE 覆盖。如果没有此参数，auth 配置将无法工作。

4.3. DOCKER COMPOSE YAML 文件

警告

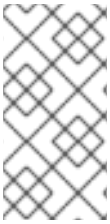
这个安装方法是为了获得针对红帽构建的 Keycloak 服务器验证 docker registry 的简单方法。它仅用于开发目的，不应在生产环境中使用。

zip 文件安装机制为希望了解如何红帽构建的 Keycloak 服务器与 Docker registry 交互的开发人员提供了一个快速入门。要配置：

流程

1. 从所需的域中，创建客户端配置。此时您没有 Docker registry - Quickstart 将负责该部分。
2. 从 *Action* 菜单中选择 "Docker Compose YAML" 选项，然后选择 Download adapter 配置选项来下载 ZIP 文件。
3. 将存档解压缩到所需位置，然后打开 目录。

4. 使用 docker-compose 启动Docker registry



注意

建议您在 'master' 以外的域中配置 Docker registry 客户端，因为 HTTP 基本身份验证流不存在表单。

在进行了上述配置后，keycloak 服务器和 Docker registry 正在运行，docker 身份验证应该成功：

```
[user ~]# docker login localhost:5000 -u $username
Password: *******
Login Succeeded
```

第 5 章 使用客户端注册服务

为了让应用程序或服务使用红帽构建的 Keycloak，它必须在红帽构建的 Keycloak 中注册客户端。管理员可以通过管理控制台（或管理 REST 端点）执行此操作，但客户端还可以通过红帽构建的 Keycloak 客户端注册服务来注册。

客户端注册服务为红帽构建的 Keycloak Client Representations、OpenID Connect Client Meta Data 和 SAML Entity Descriptors 提供内置的支持。客户端注册服务端点为 /realms/<realm>/clients-registrations/<provider>。

内置支持的 供应商 有：

- 默认 - 红帽构建的 Keycloak Client Representation (JSON)
- install - 红帽构建的 Keycloak Adapter 配置(JSON)
- openid-connect - OpenID Connect Client Metadata Description (JSON)
- saml2-entity-descriptor - SAML Entity Descriptor (XML)

以下小节介绍了如何使用不同的供应商。

5.1. 身份验证

若要调用您通常需要令牌的客户端注册服务。令牌可以是 bearer 令牌、初始访问令牌或注册访问令牌。另外，也可以注册新客户端而无需令牌，但需要配置客户端注册策略（请参阅以下）。

5.1.1. bearer 令牌

bearer 令牌可以代表用户或服务帐户发布。调用端点需要以下权限(更多详情请参阅 服务器管理指南) :

- create-client 或 manage-client - 要创建客户端

- view-client 或 manage-client - 查看客户端
- manage-client - 要更新或删除客户端

如果您使用 bearer 令牌创建客户端，建议只使用 create-client 角色的服务帐户中的令牌（更多详情请参阅 服务器管理指南 ）。

5.1.2. 初始访问令牌

注册新客户端的建议方法是使用初始访问令牌。初始访问令牌只能用于创建客户端，并具有可配置的过期时间，以及可创建客户端数量的可配置限制。

可以通过管理控制台创建初始访问令牌。若要创建新的初始访问令牌，首先在 admin 控制台中选择域，然后单击左侧菜单中的 Client，然后在页面中显示的选项卡中显示 Initial access token。

现在，您可以看到任何现有的初始访问令牌。如果您有访问权限，可以删除不再需要的令牌。您只能在创建令牌时检索令牌的值。要创建新令牌，请点 Create。现在，您可以选择添加令牌应有效的时长，也可以使用令牌创建客户端数量。点击 Save the token 值后会显示。

现在，您复制/粘贴此令牌非常重要，因为您现在无法在以后检索它。如果您忘记复制/粘贴它，请删除令牌并创建另一个令牌。

令牌值在调用客户端注册服务时用作标准 bearer 令牌，方法是将它添加到请求中的 Authorization 标头中。例如：

```
Authorization: bearer eyJhbGciOiJSUz...
```

5.1.3. 注册访问令牌

通过客户端注册服务创建客户端时，响应将包括注册访问令牌。通过注册访问令牌，可以稍后检索客户端配置，但也可用于更新或删除客户端。注册访问令牌包含在请求中，其方式与 bearer 令牌或初始访问令牌相同。

默认情况下启用注册访问令牌轮转。这意味着注册访问令牌只有效一次。使用令牌时，响应将包含新令牌。请注意，可以使用 客户端策略 禁用注册访问令牌 轮转。

如果在客户端注册服务之外创建了客户端，它不会关联有注册访问令牌。您可以通过 admin 控制台创建一个。如果您丢失了特定客户端的令牌，这也很有用。若要创建新令牌，可在管理控制台中找到客户端，然后单击 凭据。然后单击 Generate registration access token。

5.2. 红帽构建的 KEYCLOAK 代表

默认 客户端注册提供程序可用于创建、检索、更新和删除客户端。它使用红帽构建的 Keycloak Client Representation 格式，它支持通过管理控制台配置客户端，包括配置协议映射器。

要创建客户端，请创建一个客户端代表(JSON)，然后对 /realms/<realm>/clients-registrations/default 执行 HTTP POST 请求。

它将返回一个包括注册访问令牌的 Client Representation。如果要在以后检索配置、更新或删除客户端，您应该在此处保存注册访问令牌。

要检索 Client Representation，请执行对 /realms/<realm>/clients-registrations/default/<client id> 的 HTTP GET 请求。

它还将返回新的注册访问令牌。

要更新 Client Representation，请执行带有更新的 Client Representation to: /realms/<realm>/clients-registrations/default/<client id > 的 HTTP PUT 请求。

它还将返回新的注册访问令牌。

要删除客户端代表执行 HTTP DELETE 请求： /realms/<realm>/clients-registrations/default/<client id>

5.3. 红帽构建的 KEYCLOAK 适配器配置

安装 客户端注册供应商可用于检索客户端的适配器配置。除了令牌身份验证外，您还可以使用 HTTP 基本身份验证与客户端凭证进行身份验证。要做到这一点在请求中包含以下标头：

```
Authorization: basic BASE64(client-id + ':' + client-secret)
```

要检索适配器配置，然后对 /realms/<realm>/clients-registrations/install/<client id > 执行 HTTP GET 请求。

公共客户端不需要身份验证。这意味着，对于 JavaScript 适配器，您可以使用上述 URL 直接从红帽构建的 Keycloak 加载客户端配置。

5.4. OPENID CONNECT DYNAMIC CLIENT REGISTRATION

Red Hat build of Keycloak 实现 OpenID Connect Dynamic Client Registration，它扩展了 OAuth 2.0 Dynamic Client Registration Protocol 和 OAuth 2.0 Dynamic Client Registration Management Protocol。

在 Keycloak 的红帽构建中注册客户端的端点是 /realms/<realm>/clients-registrations/openid-connect[/<client id>]。

此端点也可以在域 /realms/<realm>/.well-known/openid-configuration 的 OpenID Connect Discovery 端点中找到。

5.5. SAML 实体描述符

SAML Entity Descriptor 端点只支持使用 SAML v2 Entity Descriptors 来创建客户端。它不支持检索、更新或删除客户端。对于这些操作，应使用红帽构建的 Keycloak 表示端点。在创建客户端时，红帽构建的 Keycloak Client Representation 会被返回，其中包含有关创建的客户端的详细信息，包括注册访问令牌。

要创建客户端，请执行带有 SAML Entity Descriptor 的 HTTP POST 请求到 /realms/<realm>/clients-registrations/saml2-entity-descriptor。

5.6. 使用 CURL 的示例

以下示例使用 CURL 创建一个带有 clientId myclient 的客户端。您需要将 eyJhbGciOiJSUz... 替换为正确的初始访问令牌或 bearer 令牌。

```
curl -X POST \
    -d '{ "clientId": "myclient" }' \
    -H "Content-Type:application/json" \
    -H "Authorization: bearer eyJhbGciOiJSUz..." \
    http://localhost:8080/realms/master/clients-registrations/default
```

5.7. 使用 JAVA 客户端注册 API 的示例

客户端注册 Java API 可让您使用 Java 轻松使用客户端注册服务。要使用包括依赖项 org.keycloak:keycloak-client-registration-api:>VERSION< from Maven。

有关使用客户端注册的完整说明，请参考 JavaDocs。以下是创建客户端的示例。您需要将 eyJhbGciOiJSUz... 替换为正确的初始访问令牌或 bearer 令牌。

```
String token = "eyJhbGciOiJSUz...";

ClientRepresentation client = new ClientRepresentation();
client.setClientId(CLIENT_ID);

ClientRegistration reg = ClientRegistration.create()
    .url("http://localhost:8080", "myrealm")
    .build();

reg.auth(Auth.token(token));

client = reg.create(client);

String registrationAccessToken = client.getRegistrationAccessToken();
```

5.8. 客户端注册策略

注意

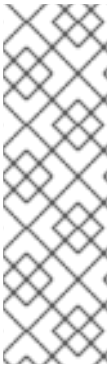
当前计划用于删除客户端注册策略，取代 服务器管理指南 中描述的客户端策略。客户端策略更灵活，支持更多用例。

红帽构建的 Keycloak 目前支持通过客户端注册服务注册新客户端的方法。

- 已验证请求 - 注册新客户端的请求必须包含上面提到的 Initial Access Token 或 Bearer Token。
- 匿名请求 - 请求注册新客户端不需要包含任何令牌

匿名客户端注册请求非常有趣且强大的功能，但您通常不希望任何人在没有任何限制的情况下注册新客户端。因此，我们有 客户端注册策略 SPI，它提供了一种限制谁可以注册新客户端以及条件下的方法。

在 Red Hat build of Keycloak 管理控制台中，您可以点 Client Registration 选项卡，然后点 Client Registration Policies 子选项卡。在这里，您将看到匿名请求默认配置了哪些策略，以及为经过身份验证的用户配置了哪些策略。



注意

允许使用匿名请求（不带任何令牌的请求）来创建（注册）新客户端。因此，当您通过匿名请求注册新客户端时，响应将包含注册访问令牌，该令牌必须用于特定客户端的 Read、Update 或 Delete 请求。但是，从匿名注册中使用此注册访问令牌也会受到匿名策略的影响！这意味着，对于更新客户端的示例请求，如果您有受信任的主机策略，则需要来自 受信任的主机 主机。例如，在更新客户端时以及存在 Consent Required 策略时，不允许禁用 Consent Required Required。

目前，我们有以下策略实现：

- 可信主机策略 - 您可以配置可信主机和可信域的列表。对客户端注册服务的请求只能从那些主机或域发送。从某些不受信任的 IP 发送的请求将被拒绝。新注册的客户端的 URL 还必须仅使用那些可信主机或域。例如，不允许设置指向某些不可信主机的客户端的 重定向 URI。默认情况下，没有任何白名单的主机，因此匿名客户端注册被禁用。

- consent Required Policy - 新注册的客户端将启用 Consent Allowed 交换机。因此，在身份验证成功后，当用户需要批准权限（客户端范围）时，用户总是看到同意的屏幕。这意味着，除非用户批准，否则客户端无法访问任何个人信息或用户的权限。

- 协议映射程序策略 - 允许配置白名单协议映射程序实现的列表。如果新客户端包含一些非转换协议映射器，则无法注册或更新新客户端。请注意，此策略也用于经过身份验证的用户，因此即使经过身份验证的用户也有一些可以使用协议映射器的限制。

- Client Scope Policy - 允许将 客户端范围 列入白名单，它们可用于新注册或更新的客户端。默认没有白名单范围；默认情况下，只有客户端范围定义为 Realm Default Client Scopes。

- 完整范围策略 - 新注册的客户端将禁用 完全范围允许 开关。这意味着它们没有范围的域角色或其他客户端的客户端角色。

- 最大客户端策略 - 如果域中当前客户端数量与指定限制相同或大于指定限制，则拒绝注册。对于匿名注册，默认为 200。

-

客户端禁用策略 - 将禁用新注册的客户端。这意味着管理员需要手动批准并启用所有新注册的客户端。即使匿名注册，默认不使用此策略。

第 6 章 使用 CLI 自动化客户端注册

客户端注册 CLI 是应用程序开发人员的命令行界面(CLI)工具，用于在与红帽构建的 Keycloak 集成时以自助服务方式配置新客户端。它旨在与红帽构建的 Keycloak 客户端注册 REST 端点交互。

需要为任何应用程序创建或获取客户端配置才能使用红帽构建的 Keycloak。您通常会为托管在唯一主机名上的每个新应用配置新客户端。当应用程序与红帽构建的 Keycloak 交互时，应用程序会使用客户端 ID 标识自己，以便红帽构建的 Keycloak 可以提供登录页面、单点登录(SSO)会话管理和其他服务。

您可以使用客户端注册 CLI 从命令行配置应用程序客户端，您可以在 shell 脚本中使用它。

要允许特定用户使用 客户端注册 CLI，红帽构建的 Keycloak 管理员通常使用 Admin 控制台配置具有正确角色的新用户，或配置新客户端和客户端 secret 以授予对客户端注册 REST API 的访问权限。

6.1. 配置新的常规用户以用于客户端注册 CLI

流程

1. 以 admin 用户身份登录管理控制台（例如 http://localhost:8080/admin）。
2. 选择要管理的域。
3. 如果要使用现有用户，请选择该用户进行编辑；否则，创建一个新用户。
4. 选择 Role Mapping,Assign role。从选项列表中，单击 Filter by client。在搜索栏中，键入 manage-clients。选择角色，或者如果您在 master 域中，使用 NAME-realm 选择一个，其中 NAME 是目标域的名称。您可以为 master 域中的用户授予对任何其他域的访问权限。
5. 单击 Assign 以授予一组完整的客户端管理权限。另一种选项是为只读选择 view-clients 或 create-client 来创建新客户端。
6. 选择 Available Roles,manage-client 以授予一组完整的客户端管理权限。另一种选项是为只读选择 view-clients 或 create-client 来创建新客户端。

注意

这些权限授予用户在不使用 Initial Access Token 或 Registration Access Token 的情况下执行操作的能力。

无法为用户分配任何 realm-management 角色。在这种情况下，用户仍然可以使用客户端注册 CLI 登录，但在没有 Initial Access Token 的情况下无法使用它。试图在没有令牌的情况下执行任何操作会导致 403 Forbidden 错误。

管理员可以从 Initial Access Token 选项卡上的 Clients 区域发出 Initial Access Tokens。

6.2. 配置客户端以用于客户端注册 CLI

默认情况下，服务器将客户端注册 CLI 识别为 admin-cli 客户端，该客户端会自动为每个新域配置。使用用户名登录时不需要额外的客户端配置。

流程

1. 如果要为客户端注册 CLI 使用单独的客户端配置，请创建一个客户端（如 reg-cli）。
2. 取消选中 Standard Flow Enabled。
3. 通过将 客户端身份验证切换为 On 来增强安全性。
4. 选择要使用的帐户类型。
   a. 如果要使用与客户端关联的服务帐户，请检查 服务帐户角色。
   b. 如果您希望使用常规用户帐户，请检查 直接访问权限授予。
5. 点击 Next。

6. 点击 Save。

7. 点 Credentials 选项卡。

   配置 Client Id 和 Secret 或 Signed JWT。

8. 如果使用服务帐户角色，请点 Service Account Roles 选项卡。

   选择角色来为服务帐户配置访问权限。有关要选择的角色的详情，请参考 第 6.1 节 “配置新的常规用户以用于客户端注册 CLI”。

9. 点击 Save。

运行 kcreg 配置凭证 时，请使用 --secret 选项提供配置的 secret。

- 在运行 kcreg config credentials 时，指定要使用哪个 clientId (例如，--client reg-cli)。
- 启用服务帐户后，您可以在运行 kcreg 配置凭证时 省略指定用户，并仅提供客户端 secret 或密钥存储信息。

6.3. 安装客户端注册 CLI

客户端注册 CLI 打包在红帽构建的 Keycloak 服务器分发中。您可以在 bin 目录中找到执行脚本。Linux 脚本称为 kcreg.sh，Windows 脚本名为 kcreg.bat。

在设置用于从文件系统的任何位置使用的客户端时，将红帽 Keycloak 服务器目录添加到 PATH 中。

例如，在：

- linux:

```
$ export PATH=$PATH:$KEYCLOAK_HOME/bin
$ kcreg.sh
```

- **Windows：**

```
c:\> set PATH=%PATH%;%KEYCLOAK_HOME%\bin
c:\> kcreg
```

KEYCLOAK_HOME 是指红帽构建的 Keycloak 服务器分发被解包的目录。

6.4. 使用客户端注册 CLI

流程

1. **使用您的凭证登录，启动经过身份验证的会话。**

2. **在 客户端注册 REST 端点上 运行命令。**

   例如，在：

   - **linux:**

     ```
     $ kcreg.sh config credentials --server http://localhost:8080 --realm demo --user user --
     client reg-cli
     $ kcreg.sh create -s clientId=my_client -s 'redirectUris=["http://localhost:8980/myapp/*"]'
     $ kcreg.sh get my_client
     ```

   - **Windows：**

     ```
     c:\> kcreg config credentials --server http://localhost:8080 --realm demo --user user --
     client reg-cli
     c:\> kcreg create -s clientId=my_client -s "redirectUris=[\"http://localhost:8980/myapp/*\"]"
     c:\> kcreg get my_client
     ```

注意

在生产环境中，红帽构建的 Keycloak 必须通过 https: 访问，以避免向网络嗅探器公开令牌。

3. 如果服务器的证书不是由 Java 默认证书信任存储中包含的可信证书颁发机构(CA)发布，请准备 信任存储.jks 文件并指示客户端注册 CLI 使用它。

例如，在：

- linux:

```
$ kcreg.sh config truststore --trustpass $PASSWORD ~/.keycloak/truststore.jks
```

- Windows：

```
c:\> kcreg config truststore --trustpass %PASSWORD%
%HOMEPATH%\.keycloak\truststore.jks
```

6.4.1. 登录

流程

1. 使用客户端注册 CLI 登录时指定服务器端点 URL 和域。

2. 指定用户名或客户端 id，这会导致使用特殊服务帐户。使用用户名时，您必须为指定用户使用密码。在使用客户端 ID 时，您可以使用客户端 secret 或 Signed JWT 而不是密码。

无论登录方法如何，登录的帐户都需要适当的权限才能执行客户端注册操作。请记住，非 master 域中的任何帐户都只能具有在同一域中管理客户端的权限。如果您需要管理不同的域，您可以在不同的域中配置多个用户，也可以在 master 域中创建一个用户，并在不同的域中添加管理客户端的角色。

您不能使用客户端注册 CLI 配置用户。使用 Admin Console Web 界面或 Admin Client CLI 配置用户。如需了解更多详细信息，请参阅 服务器管理指南。

当 kcreg 成功登录时，它会接收授权令牌并将其保存在私有配置文件中，以便令牌可用于后续调用。有关配置文件的详情，请参考 第 6.4.2 节 “使用其他配置”。

有关使用客户端注册 CLI 的更多信息，请参阅内置帮助。

例如，在：

- linux:

```
$ kcreg.sh help
```

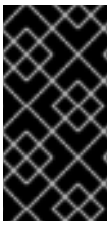
- Windows：

```
c:\> kcreg help
```

有关启动经过身份验证的会话的更多信息，请参阅 kcreg config credentials --help。

6.4.2. 使用其他配置

默认情况下，客户端注册 CLI 会在用户的主目录下自动维护一个配置文件 ./.keycloak/kcreg.config。您可以使用 --config 选项指向不同的文件或位置，以并行维护多个经过身份验证的会话。从单个线程执行绑定到单个配置文件的操作是安全的。



重要

不要使配置文件对系统上的其他用户可见。配置文件包含应保持私有的访问令牌和 secret。

您可能希望通过将 --no-config 选项与所有命令搭配使用，避免将 secret 存储在配置文件中，即使它不太方便，且需要更多令牌请求来执行此操作。使用每个 kcreg 调用指定所有身份验证信息。

6.4.3. 初始访问和注册访问令牌

在 Red Hat build of Keycloak 服务器中没有配置帐户的开发人员可以使用 Client Registration CLI。只有当域管理员向开发人员发出 Initial Access Token 时，才能实现。域管理员最多可决定如何以及何时发布这些令牌。域管理员可以限制 Initial Access Token 的最长期限以及可在其中创建的客户端总数。

开发人员有初始访问令牌后，开发人员可以使用它来创建新客户端，而无需通过 kcreg 配置凭据进行身份验证。Initial Access Token 可以存储在配置文件中，也可以作为 kcreg create 命令的一部分指定。

例如，在：

- **linux:**

```
$ kcreg.sh config initial-token $TOKEN
$ kcreg.sh create -s clientId=myclient
```

或者

```
$ kcreg.sh create -s clientId=myclient -t $TOKEN
```

- **Windows：**

```
c:\> kcreg config initial-token %TOKEN%
c:\> kcreg create -s clientId=myclient
```

或者

```
c:\> kcreg create -s clientId=myclient -t %TOKEN%
```

使用初始访问令牌时，服务器响应包括新发布的注册访问令牌。该客户端的任何后续操作都需要通过该令牌进行身份验证来执行，这仅对该客户端有效。

客户端注册 CLI 会自动使用其专用配置文件来保存此令牌并将其与关联的客户端一起使用。只要所有客户端操作使用相同的配置文件，开发人员不需要通过身份验证来读取、更新或删除以这种方式创建的客户端。

有关初始访问和注册访问令牌的更多信息，请参阅客户端注册。

运行 **kcreg config initial-token --help** 和 **kcreg config registration-token --help** 命令以了解有关如何使用客户端注册 CLI 配置令牌的更多信息。

6.4.4. 创建客户端配置

使用凭证或配置 Initial Access Token 进行身份验证后，第一项任务通常是创建新客户端。通常，您

可能希望将准备好的 JSON 文件用作模板，并设置或覆盖某些属性。

以下示例演示了如何读取 JSON 文件，覆盖它可能包含的任何客户端 ID，设置任何其他属性，并在成功创建后将配置输出到标准输出。

- **linux:**

```
$ kcreg.sh create -f client-template.json -s clientId=myclient -s baseUrl=/myclient -s 'redirectUris=["/myclient/*"]' -o
```

- **Windows：**

```
C:\> kcreg create -f client-template.json -s clientId=myclient -s baseUrl=/myclient -s "redirectUris=[\"/myclient/*\"]" -o
```

运行 **kcreg create --help** 以了解有关 **kcreg create** 命令的更多信息。

您可以使用 **kcreg attrs** 来列出可用的属性。请记住，不会检查许多配置属性以获得有效性或一致性。您要指定正确的值。请记住，您模板中不应包含任何 **id** 字段，不应将它们指定为 **kcreg create** 命令的参数。

6.4.5. 检索客户端配置

您可以使用 **kcreg get** 命令检索现有的客户端。

例如，在：

- **linux:**

```
$ kcreg.sh get myclient
```

- **Windows：**

```
C:\> kcreg get myclient
```

您还可以将客户端配置作为适配器配置文件检索，您可以使用 web 应用程序进行打包。

例如，在：

- linux:

```
$ kcreg.sh get myclient -e install > keycloak.json
```

- Windows：

```
C:\> kcreg get myclient -e install > keycloak.json
```

运行 kcreg get --help 命令以了解有关 kcreg get 命令的更多信息。

6.4.6. 修改客户端配置

可以通过两种方法更新客户端配置。

一种方法是在获取当前配置后向服务器提交完整的新状态，将其保存到文件，编辑该文件，并将它重新发布到服务器。

例如，在：

- linux:

```
$ kcreg.sh get myclient > myclient.json
$ vi myclient.json
$ kcreg.sh update myclient -f myclient.json
```

- Windows：

```
C:\> kcreg get myclient > myclient.json
C:\> notepad myclient.json
C:\> kcreg update myclient -f myclient.json
```

第二种方法获取当前的客户端，设置或删除它上的字段，并将其重新置于一个步骤中。

例如，在：

- **linux:**

```
$ kcreg.sh update myclient -s enabled=false -d redirectUris
```

- **Windows：**

```
C:\> kcreg update myclient -s enabled=false -d redirectUris
```

您还可以使用仅包含要应用的更改的文件，因此您不必将太多值指定为参数。在这种情况下，指定 **--merge** 来告知客户端注册 CLI 而不是将 JSON 文件视为完整的新配置，而是将其视为一组要应用到现有配置的属性。

例如，在：

- **linux:**

```
$ kcreg.sh update myclient --merge -d redirectUris -f mychanges.json
```

- **Windows：**

```
C:\> kcreg update myclient --merge -d redirectUris -f mychanges.json
```

运行 **kcreg update --help** 命令以了解有关 **kcreg update** 命令的更多信息。

6.4.7. 删除客户端配置

使用以下示例删除客户端。

- **linux:**

```
$ kcreg.sh delete myclient
```

- **Windows：**

```
C:\> kcreg delete myclient
```

运行 **kcreg delete --help** 命令以了解有关 **kcreg delete** 命令的更多信息。

6.4.8. 刷新无效的注册访问令牌

使用 **--no-config** 模式执行创建、读取、更新和删除(CRUD)操作时，客户端注册 **CLI** 无法为您处理注册访问令牌。在这种情况下，可能会丢失客户端最近发布的注册访问令牌的跟踪，这样就无法在该客户端上执行进一步的 **CRUD** 操作，而无需使用具有 管理客户端 权限的帐户进行身份验证。

如果您有权限，您可以为客户端发布新的注册访问令牌，并将其打印到标准输出或保存到您选择的配置文件中。否则，您必须要求域管理员为您的客户端发布新的注册访问令牌，并将其发送给您。然后，您可以通过 **--token** 选项将其传递给任何 **CRUD** 命令。您还可以使用 **kcreg config registration-token** 命令将新令牌保存到配置文件中，并让客户端注册 **CLI** 会自动处理它。

运行 **kcreg update-token --help** 命令以了解有关 **kcreg update-token** 命令的更多信息。

6.5. 故障排除

- 问：登录时，收到一个错误： **Parameter client_assertion_type is missing [invalid_client].**

  答：此错误意味着您的客户端配置了 **Signed JWT** 令牌凭证，这意味着您在登录时必须使用 **--keystore** 参数。