Red Hat build of Keycloak 24.0

授权服务指南

Last Updated: 2024-06-28

法律通告

摘要

本指南包含红帽构建的 Keycloak 24.0 的授权服务的信息。

目录

第 1 章 授权服务概述

红帽构建的 Keycloak 支持精细的授权策略，并可组合不同的访问控制机制，例如：

- **基于属性的访问控制(ABAC)**
- **基于角色的访问控制(RBAC)**
- **基于用户的访问控制(UBAC)**
- **基于上下文的访问控制(CBAC)**
- **基于规则的访问控制**
  - 使用 JavaScript
- **基于时间的访问控制**
- **通过服务提供商接口(SPI)支持自定义访问控制机制(ACM)**

Red Hat build of Keycloak 基于一组管理 UI 和 RESTful API，并提供为受保护的资源和范围创建权限所必需的方法，将这些权限与授权策略关联，并在您的应用程序和服务中强制实施授权决策。

资源服务器（应用程序或服务保护资源）通常依赖某些类型的信息来决定应将访问权限授予受保护的资源。对于基于 RESTful 的资源服务器，该信息通常从安全令牌获取，通常在每个请求上作为 bearer 令牌发送到服务器。对于依赖会话验证用户的 Web 应用程序，该信息通常存储在用户的会话中，并从那里检索每个请求。

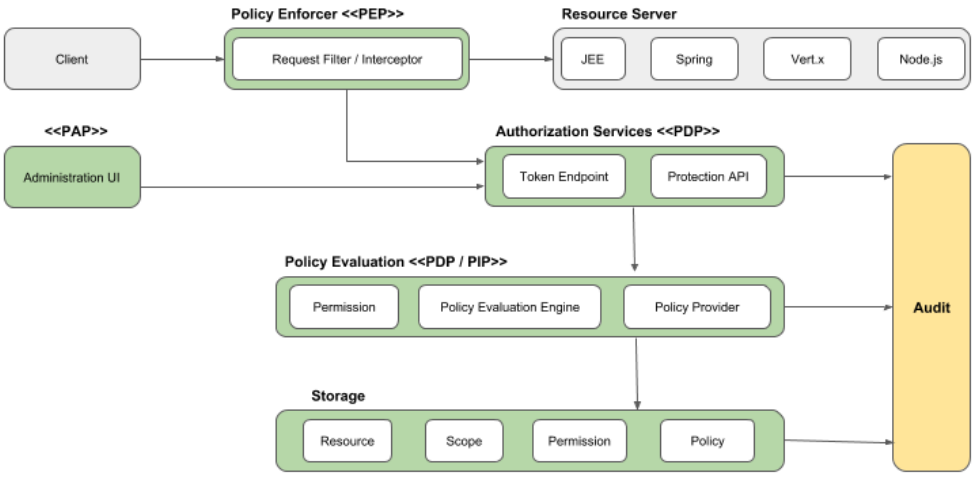
通常，资源服务器仅根据基于角色的访问控制(RBAC)执行授权决策，针对映射到这些相同资源的角色，检查授予访问受保护的资源的用户的角色。虽然角色对于应用程序非常有用且使用，但也有一些限制：

- 资源和角色与角色紧密耦合和更改（如添加、删除或更改访问上下文）可能会影响多个资源
- 对安全要求的更改会简化对应用程序代码的更改，以反映这些更改
- 根据您的应用程序大小，角色管理可能会变得困难，容易出错
- 它不是最灵活的访问控制机制。角色并不代表您是谁和缺少上下文信息。如果您被授予了角色，则至少具有一些访问权限。

请注意，现在我们需要考虑用户在不同区域分发的异构环境，使用不同的本地策略，使用不同的设备，以及对信息共享的高需求，红帽构建的 Keycloak 授权服务可以帮助您改进应用程序和服务的授权功能：

- 使用细粒度授权策略和不同的访问控制机制进行资源保护
- 集中式资源、权限和策略管理
- 集中式策略决策点
- 基于一组基于 REST 的授权服务的 REST 安全性
- 授权工作流和用户管理的访问
- 有助于避免跨项目进行代码复制（和重新部署）的基础架构，并快速适应安全要求的变化。

1.1. 架构



从设计的角度来看，授权服务基于一组明确定义的授权模式，提供以下功能：

- **策略管理点(PAP)**
  根据红帽构建的 Keycloak 管理控制台提供一组 UI，以管理资源服务器、资源、范围、权限和策略。部分也通过使用 保护 API 远程完成。

- **策略决策点(PDP)**
  提供可分布式策略决策点，指向发送授权请求的位置，并使用请求的权限进行相应评估。如需更多信息，请参阅 获取权限。

- **策略强制点(PEP)**
  为不同的环境提供实施，以便在资源服务器端实际强制执行授权决策。Red Hat build of Keycloak 提供了一些内置 Policy Enforcers。

- **策略信息点(PIP)**
  基于红帽构建的 Keycloak 身份验证服务器，您可以在评估授权策略时从身份和运行时环境获取属性。

1.1.1. 授权过程

三个主要流程定义了必要的步骤，了解如何使用红帽构建的 Keycloak 为应用程序启用精细的授权：

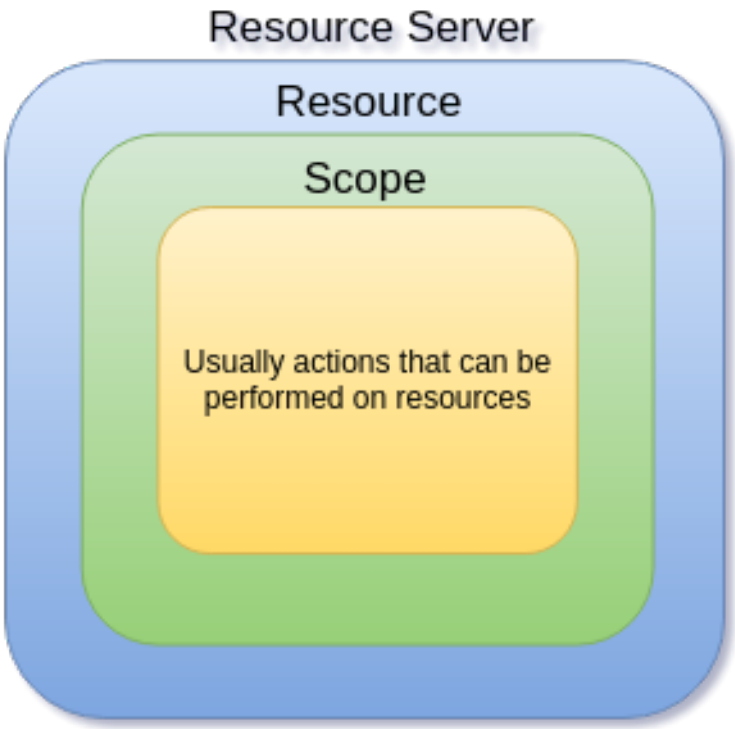
- **资源管理**
- **权限和策略管理**
- **策略强制**

1.1.1.1. 资源管理

资源管理 涉及定义所保护内容所需的所有步骤。

首先，您需要指定您要保护的 Keycloak 的红帽构建，这通常代表 web 应用程序或一组一个或多个服务。有关资源服务器的更多信息，请参阅 术语。

资源服务器使用红帽构建的 Keycloak 管理控制台进行管理。您可以使用任何注册的客户端应用程序作为资源服务器，并开始管理您要保护的资源和范围。



资源可以是网页、RESTFul 资源、您的文件系统中的文件、EJB 等。它们可以代表一组资源（就像 Java 中的类一样），或者可以代表单个和特定资源。

例如，您可能有一个代表所有 *银行帐户的 Account* 帐户资源，并使用它来定义所有银行帐户通用的授权策略。但是，您可能希望为 alice *帐户* 定义特定策略（属于某个客户的资源实例），其中只有所有者才能访问某些信息或执行操作。

可以使用红帽构建的 Keycloak 管理控制台或 保护 API 来管理资源。在后者的情况下，资源服务器能够远程管理其资源。

范围通常代表可以对资源执行的操作，但它们不仅限于这一操作。您还可以使用范围来代表资源中的一个或多个属性。

1.1.1.2. 权限和策略管理

在定义了资源服务器和您要保护的所有资源后，您必须设置权限和策略。

这个过程涉及实际定义管理您的资源的安全性和访问要求所需的所有步骤。

策略定义了必须满足访问或对资源（资源或范围）执行操作的条件，但它们没有与保护的内容相关联。它们是通用的，可以重复使用来构建权限甚至更复杂的策略。

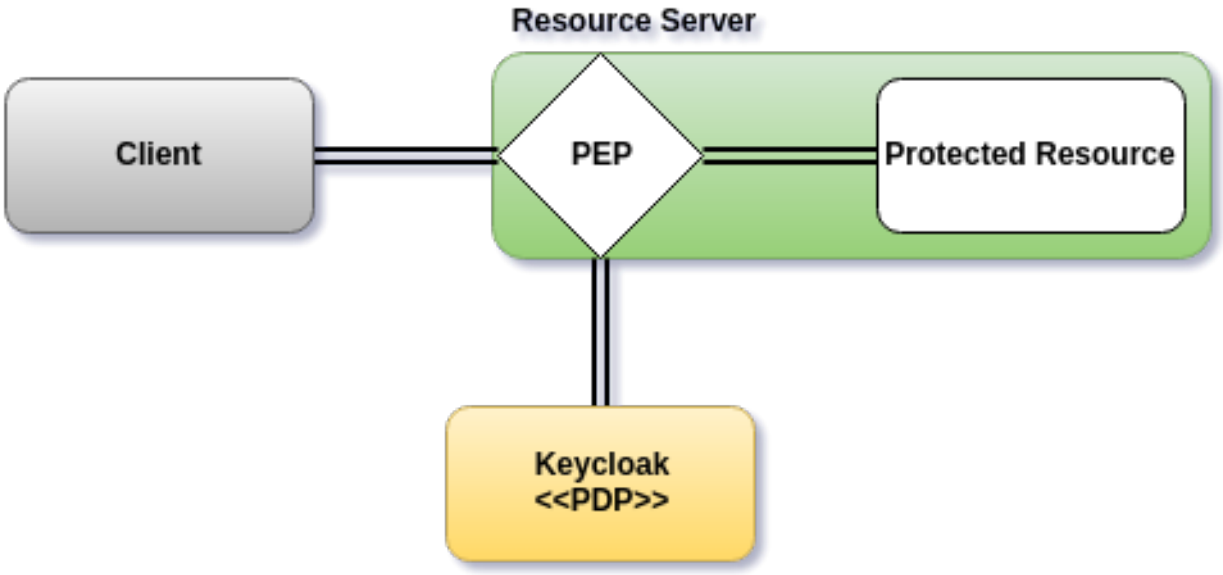
例如，要仅允许对赋予角色"用户高级"的用户访问一组资源，您可以使用 RBAC （基于角色的访问控制）。

Red Hat build of Keycloak 提供了几个内置策略类型（及其相应的策略供应商），涵盖了最常见的访问控制机制。您甚至可根据使用 JavaScript 编写的规则创建策略。

定义了策略后，您可以开始定义权限。权限与它们保护的资源相结合。在这里，您可以指定您要保护的内容（资源或范围）以及必须满足授权或拒绝权限的策略。

1.1.1.3. 策略强制

策略强制 涉及实际对资源服务器实施授权决策所需的步骤。这可以通过在能够与授权服务器通信的资源服务器上启用 **Policy Enforcement Point** 或 PEP，请求授权数据并根据服务器返回的决策和权限控制对受保护的资源的访问。



Red Hat build of Keycloak 提供了一些内置的 Policy Enforcers 实现，您可以根据它们运行的平台来保护应用程序。

1.1.2. 授权服务

授权服务由以下 RESTFul 端点组成：

- **令牌端点**
- **资源管理端点**
- **权限管理端点**

这些服务各自提供一个特定的 API，涵盖授权过程中涉及的不同步骤。

1.1.2.1. 令牌端点

OAuth2 客户端（如前端应用）可以使用令牌端点从服务器获取访问令牌，并使用这些相同的令牌来访问由资源服务器（如后端服务）保护的资源。同样，红帽构建的 Keycloak 授权服务为 OAuth2 提供扩展，以允许根据请求的资源或范围关联的所有策略来处理访问令牌。这意味着，资源服务器可以根据服务器授予的权限并由访问令牌持有来强制实施对其受保护的资源的访问。在红帽构建的 Keycloak 授权服务中，具有权限的访问令牌称为 请求第三方令牌或 RPT。

其他资源

- 获取权限

1.1.2.2. 保护 API

Protection API 是一组与 UMA 兼容的 端点置备操作，帮助它们管理与其关联的资源、范围、权限和策略。只有资源服务器才能访问此 API，这还需要 **uma_protection** 范围。

保护 API 提供的操作可以分为两个主要组：

- **资源管理**
  - 创建资源
  - 删除资源
  - 按 Id 查找
  - 查询
- **权限管理**
  - 问题权限提示

注意

默认情况下启用远程资源管理。您可以使用红帽构建的 Keycloak 管理控制台更改，只允许通过控制台进行资源管理。

使用 UMA 协议时，保护 API 的 Permission Tickets 的颁发是整个授权过程的一个重要部分。如后续小节中所述，它们代表客户端请求的权限，并发送到服务器以获取与请求的资源和范围关联的权限及策略期间授予的所有权限的最终令牌。

其他资源

- 保护 API

1.2. 术语

进一步之前，务必要了解红帽构建的 Keycloak 授权服务所引入的术语和概念。

1.2.1. 资源服务器

根据 OAuth2 术语，资源服务器是托管受保护的资源的服务器，能够接受和响应受保护的资源请求。

资源服务器通常依赖某种类型的信息来决定是否应该被授予对受保护的资源的访问。对于基于 RESTful 的资源服务器，这些信息通常在安全令牌中传输，通常作为 bearer 令牌发送，以及每个对服务器的请求。依赖会话验证用户的 Web 应用通常会将该信息存储在用户的会话中，并从那里检索每个请求。

在红帽构建的 Keycloak 中，任何 **机密** 客户端应用程序都可以充当资源服务器。此客户端的资源及其各自范围受一组授权策略保护和管理。

1.2.2. 资源

资源是应用程序和组织资产的一部分。它可以是一个或多个端点的集合，一个典型的 Web 资源，如 HTML 页面等。在授权策略术语中，资源是受保护的 *对象*。

每个资源都有一个唯一标识符，它可以代表单个资源或一组资源。例如，您可以管理代表并为所有 *银行帐户定义一组授权策略的银行帐户资源*。但是，您还可以有一个名为 alice *银行帐户* 的不同资源，它代表了单个客户拥有的单个资源，这些资源可以拥有自己的授权策略。

1.2.3. 影响范围

资源的范围是可以对资源执行的有一定程度的访问范围。在授权策略术语中，范围是可逻辑应用到资源的许多 *操作动词* 之一。

它通常表示可以使用给定资源完成什么操作。范围示例包括 view、edit、delete 等。但是，范围也可以与资源提供的特定信息相关。在这种情况下，您可以拥有项目资源和成本范围，其中成本范围用于为用户访问项目成本定义特定策略和权限。

1.2.4. 权限

考虑这个简单和非常常见的权限：

权限将受保护的对象与必须评估的策略相关联，以确定是否授予访问权限。

- X 可在资源 Z 上执行 Y
    - 其中 ...
        - X 代表一个或多个用户、角色或组，或者它们的组合。您也可以在此处使用声明和上下文。
        - Y 代表要执行的操作，例如 write、view 等等。
        - z 代表受保护的资源，例如 "/accounts"。

红帽构建的 Keycloak 提供了丰富的平台，用于构建一系列权限策略，范围从简单到非常复杂的、基于规则的动态权限。它提供灵活性并帮助：

- 减少代码重构和权限管理成本
- 支持更灵活的安全模型，帮助您轻松适应安全要求的变化
- 在运行时进行更改；应用程序仅关注被保护的资源和范围，而不关注它们如何保护。

1.2.5. 策略

策略定义了必须满足的条件，才能授予对对象的访问权限。与权限不同，您不指定对象受保护，而指定必须满足对给定对象（如资源、范围或两者）必须满足的条件。策略与可用于保护资源的不同访问控制机制(ACM)密切相关。借助策略，您可以为基于属性的访问控制(ABAC)、基于角色的访问控制(RBAC)、基于

上下文的访问控制或其中任何组合实施策略。

红帽构建的 Keycloak 利用策略的概念，以及如何通过提供聚合策略的概念来定义它们，您可以在其中构建"策略策略"，并仍然控制评估的行为。红帽构建的 Keycloak 授权服务中的策略实施不符合对给定资源的所有条件，而不是编写一个大型策略，而是遵循其划分的技术。也就是说，您可以创建单独的策略，然后使用不同的权限重复使用它们，并通过组合单独的策略来构建更复杂的策略。

1.2.6. 策略供应商

策略供应商是特定策略类型的实现。Red Hat build of Keycloak 提供内置的策略，由对应的策略供应商支持，您可以创建自己的策略类型来支持您的特定要求。

Red Hat build of Keycloak 提供了一个 SPI (Service Provider Interface)，可用于插入您自己的策略供应商实现。

1.2.7. 权限票据

权限票据是用户管理的访问(UMA)规范定义的特殊令牌类型，它提供由授权服务器决定的不透明结构。此结构代表了客户端请求的资源和/或范围，以及必须应用到授权数据请求的策略（请求方令牌 [RPT]）。

在 UMA 中，权限票据对于支持个人共享以及个人与组织共享至关重要。使用授权工作流的权限票据可简化一系列场景，其中资源所有者和资源服务器可根据管理这些资源访问的精细策略完全控制其资源。

在 UMA 工作流中，授权服务器向资源服务器发出权限票据，这会将权限票据返回到尝试访问受保护的资源的客户端。客户端收到票据后，它可以通过将票据发回到授权服务器来请求 RPT （包含授权数据的最终令牌）。

有关权限票据的更多信息，请参阅 用户管理的访问 和 UMA 规格。

第 2 章 开始使用

对于某些应用程序，您可以查看以下资源来快速开始使用红帽构建的 Keycloak 授权服务：

- 在 Wildfly 中保护 JakartaEE 应用程序
- 保护 Spring Boot 应用程序
- 保护 Quarkus 应用程序
- 保护 Node.js 应用程序

第 3 章 管理资源服务器

根据 OAuth2 规范，资源服务器是托管受保护资源的服务器，能够接受和响应受保护的资源请求。

在红帽构建的 Keycloak 中，资源服务器提供了一个丰富的平台，可为其受保护的资源启用精细的授权，其中可以根据不同的访问控制机制做出授权决策。

任何客户端应用程序都可以配置为支持细粒度权限。这样，您的概念将客户端应用程序转换为资源服务器。

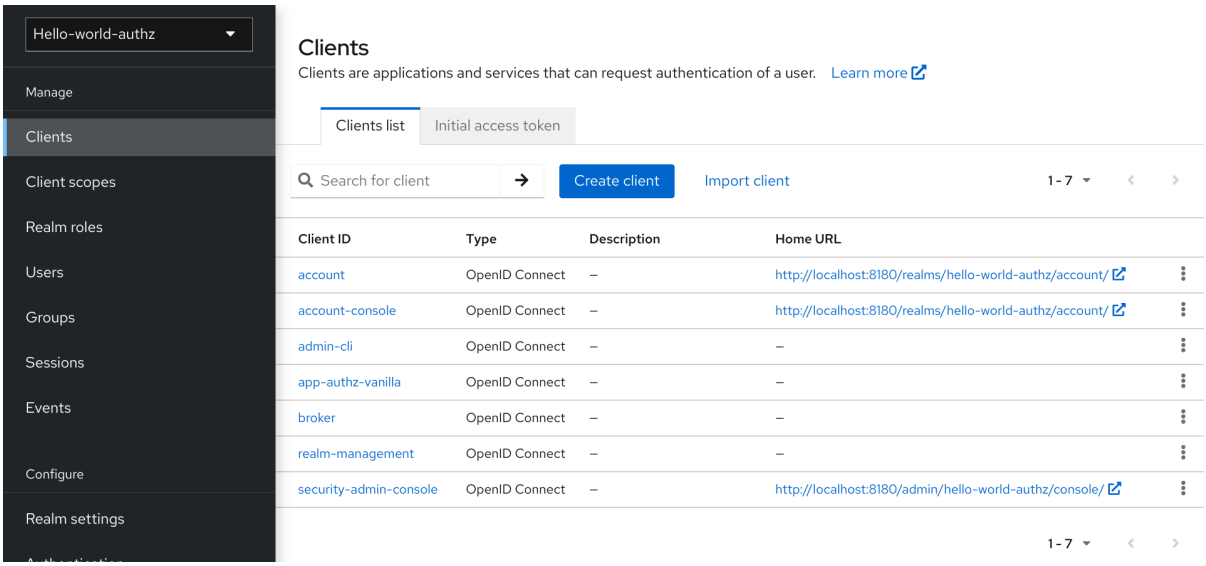
3.1. 创建客户端应用程序

启用红帽构建的 Keycloak 授权服务的第一步是创建您要切换到资源服务器的客户端应用程序。

流程

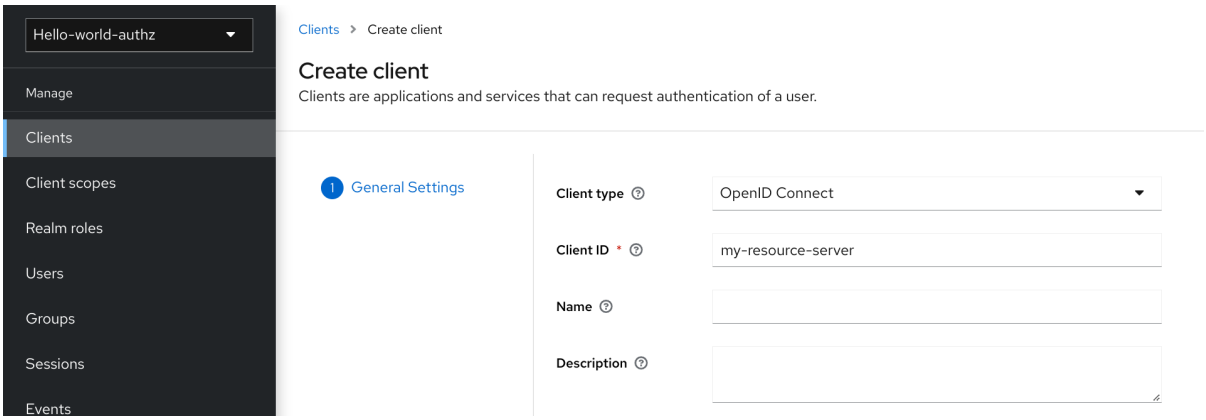
1. 点 **Clients**。

   客户端



2. 在此页面上，单击 **Create**。

   添加客户端



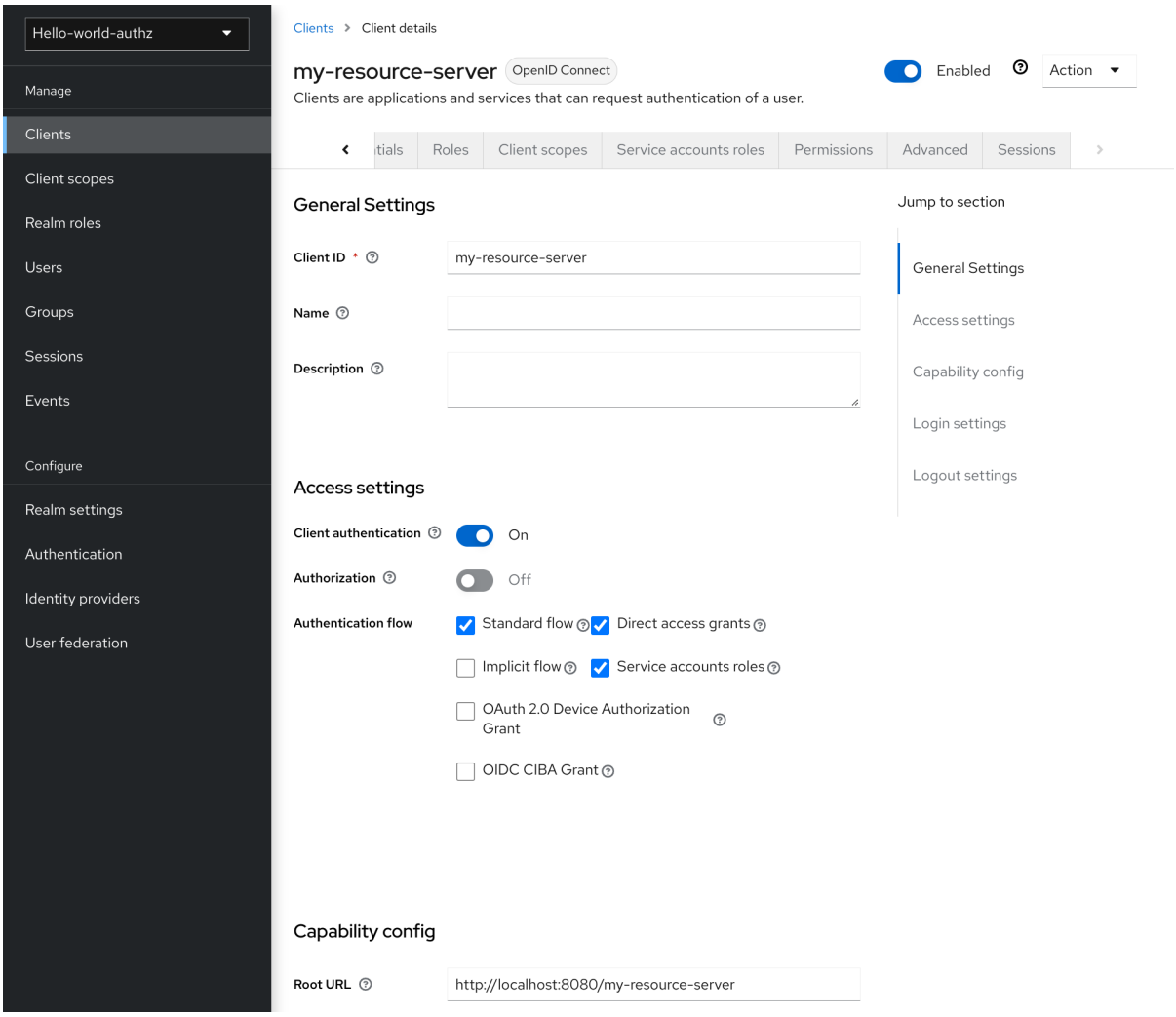
3. **键入客户端的客户端 ID**。例如，*my-resource-server*。

4. 输入应用程序的 **Root URL**。例如：

```
http://${host}:${port}/my-resource-server
```

5. 点 **Save**。创建客户端，并打开 client Settings 页面。此时会显示类似如下的页面：

客户端设置



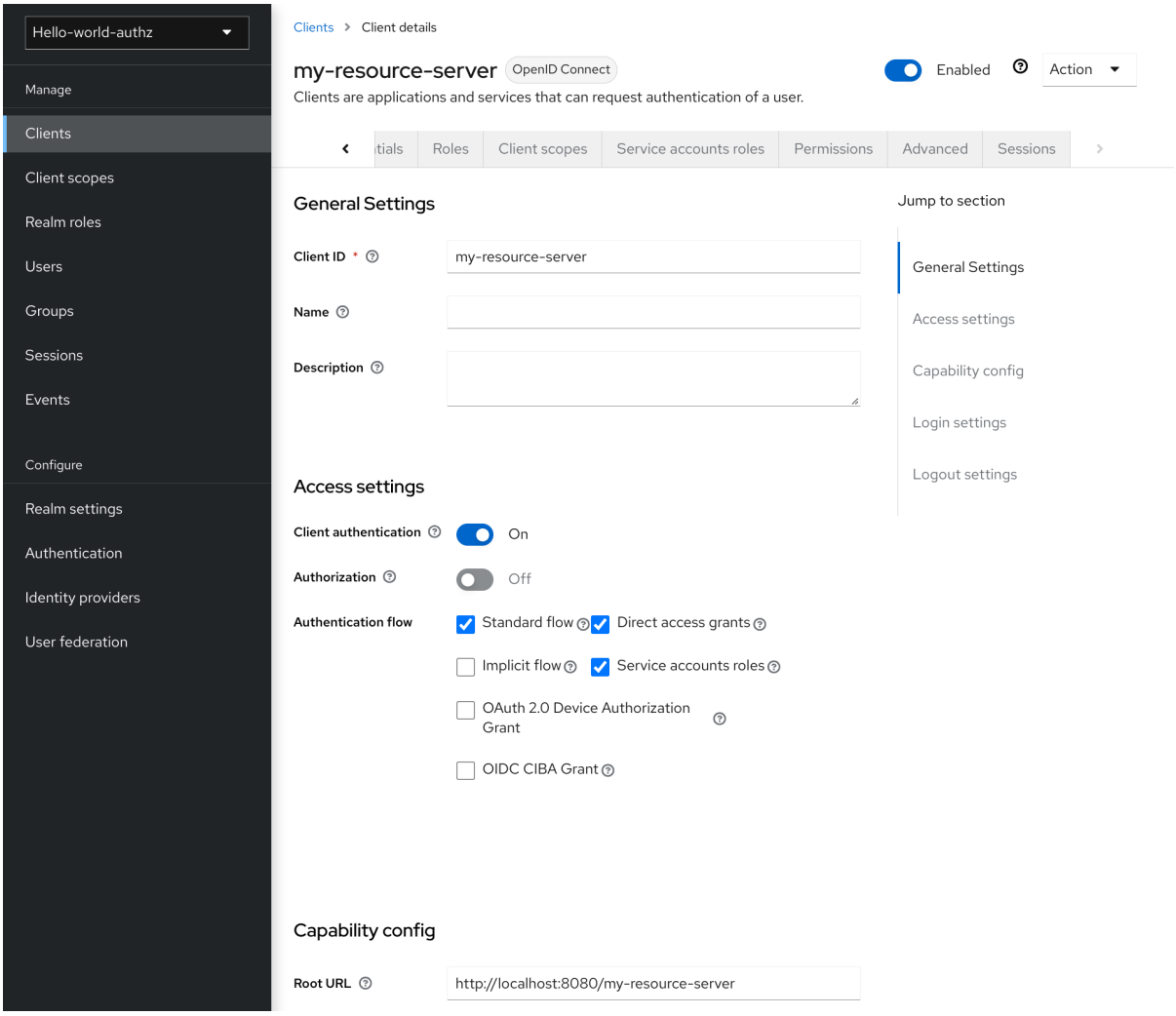
3.2. 启用授权服务

您可以将 OIDC 客户端转换为资源服务器并启用精细的授权。

流程

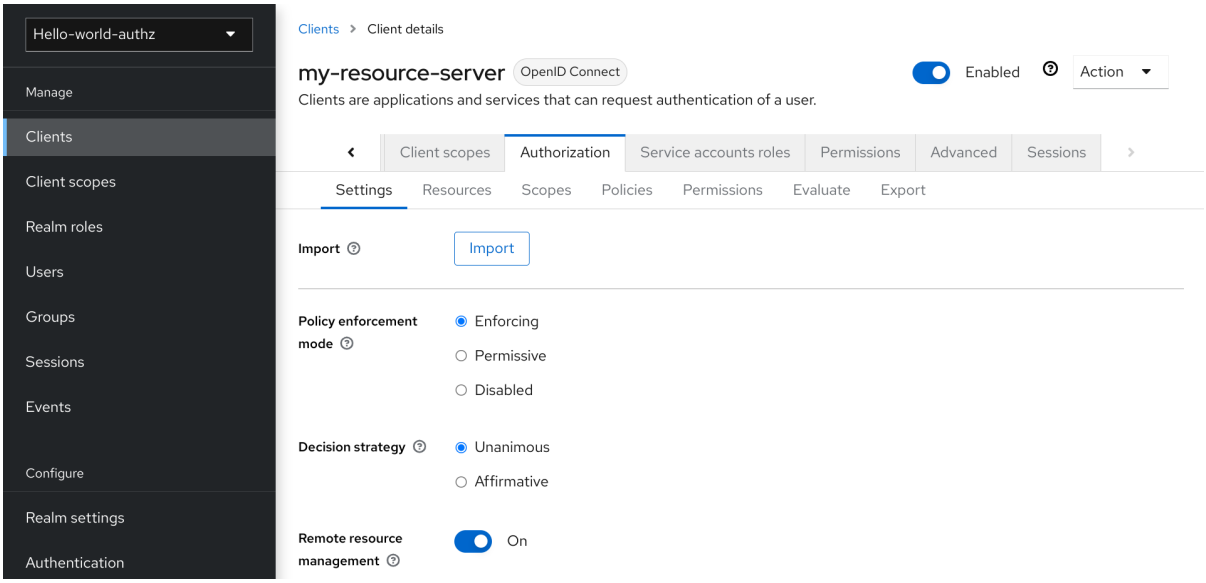
1. 将 **Authorization Enabled** 切换到 **'On**。
2. 点击 **Save**。

启用授权服务



此客户端会显示一个新的 Authorization 选项卡。点 **Authorization** 选项卡，并显示类似如下的页面：

资源服务器设置



Authorization 选项卡包含额外的子选项卡，涵盖了您必须遵循的不同步骤来实际保护应用程序的资源。每个标签页分别在本文档的特定主题中单独介绍。但是，这里是一个关于每个操作的快速描述：

- **设置**
  资源服务器的一般设置。有关此页面的详情，请参阅 Resource Server Settings 部分。

- **资源**
  在这个页面中，您可以管理 应用程序的资源。

- **授权范围**
  在这个页面中，您可以管理 范围。

- **策略（policy）**
  在这个页面中，您可以管理 授权策略，并定义必须满足的条件来授予权限。

- **权限**
  在这个页面中，您可以通过将受保护的资源和范围与您创建的策略链接来管理它们。???

- **评估**
  在这个页面中，您可以 模拟授权请求 并查看您定义的权限和授权策略的结果。

- **导出设置**
  在此页面中，您可以将授权设置 导出到 JSON 文件。

3.2.1. 资源服务器设置

在 Resource Server Settings 页面中，您可以配置策略执行模式、允许远程资源管理并导出授权配置设置。

- **策略强制模式**
  指定在处理发送到服务器的授权请求时如何强制实施策略。

  - Enforcing
    （默认模式）默认拒绝请求，即使没有与给定资源关联的策略。

  - Permissive
    即使没有与给定资源关联的策略，也允许请求。

  - Disabled
    禁用所有策略的评估，并允许访问所有资源。

- **决策策略**
  此配置会改变策略评估引擎如何根据所有评估权限的结果来决定资源或范围是否应被授予。**Affirmative** 表示，至少有一个权限必须评估为正决定，以便授予资源及其范围的访问权限。**不重要的** 意味着，所有权限都必须评估为正决定，以便最终决定也是正面的。例如，如果同一资源或范围的两个权限存在冲突（其中一个权限是授予访问权限，另一个是拒绝访问），则如果所选策略是 **Affirmative**，则授予资源或范围的权限。否则，来自任何权限的一个拒绝也会拒绝对资源或范围的访问。

- **远程资源管理**
  指定资源是否可以由资源服务器远程管理。如果为 false，则只能从管理控制台管理资源。

3.3. 默认配置

当您创建资源服务器时，红帽构建的 Keycloak 会为新创建的资源服务器创建一个默认配置。

默认配置包括：

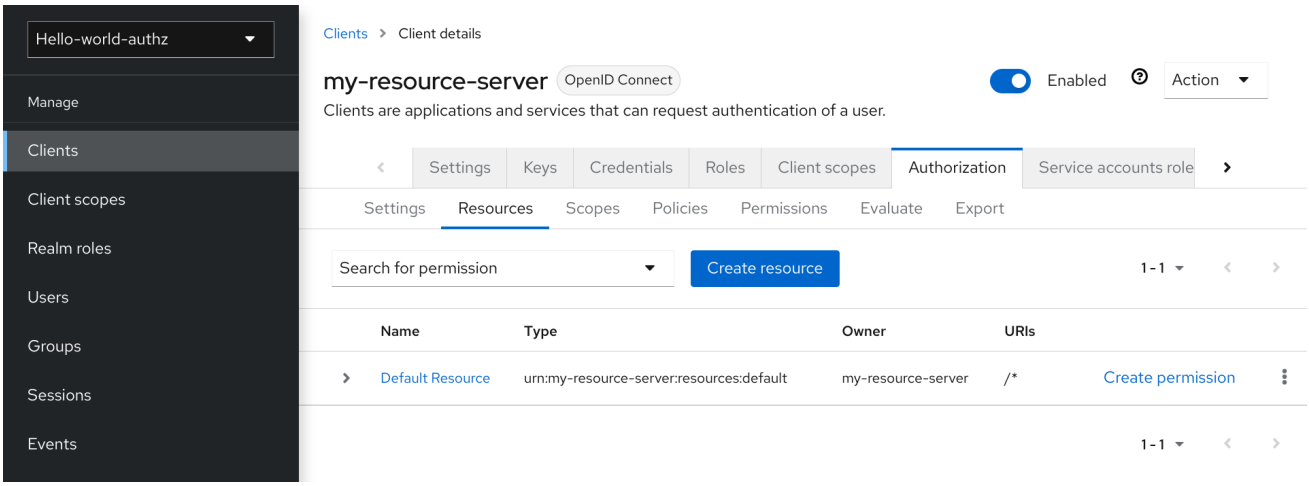
- 代表应用程序中的所有资源的默认保护资源。

- 策略始终授予对此策略保护的资源的访问权限。

- 监管根据默认策略对所有资源的访问的权限。

默认受保护的资源称为 **默认资源**，如果您导航到 **Resources** 选项卡，可以查看它。

默认资源

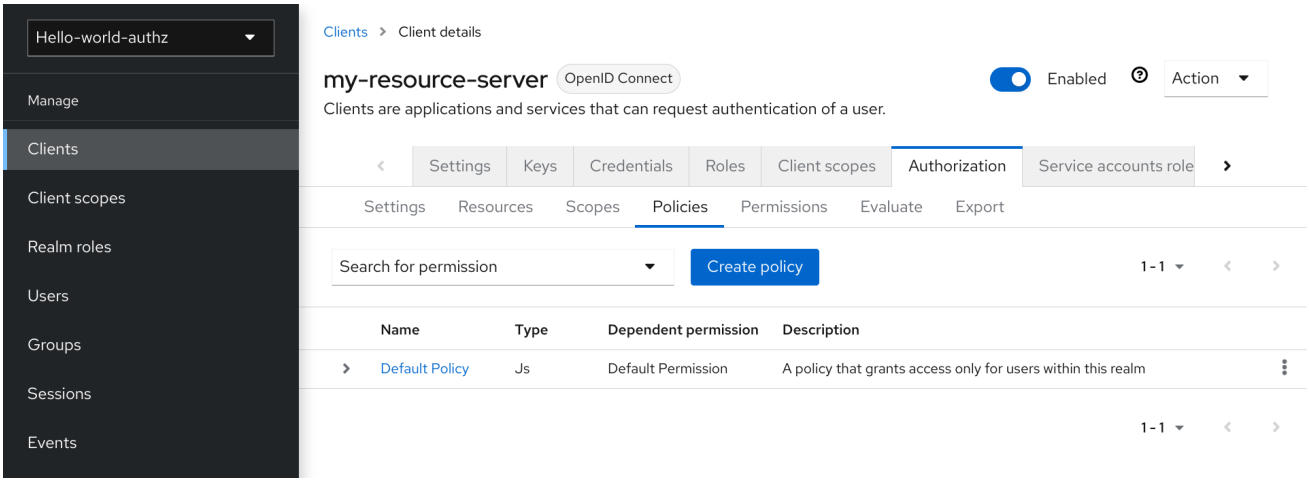


此资源定义了一个 **Type**，即 **urn:my-resource-server:resources:default** 和 **URI** Attr。**在这里，URI 字段定义了一个通配符模式，表示此资源的 Keycloak 代表应用程序中的所有路径。换句话说，当 为您的应用程序启用策略强制 时，将在授予访问权限前检查与该资源关联的所有权限。**

前面提到的 Type 定义了一个值，可用于创建 typed 资源权限，该帐户必须应用于默认资源或您使用相同类型创建的任何其他资源。

默认策略称为 唯一的域策略，如果您进入到 Policies 选项卡，可以查看该策略。

默认策略



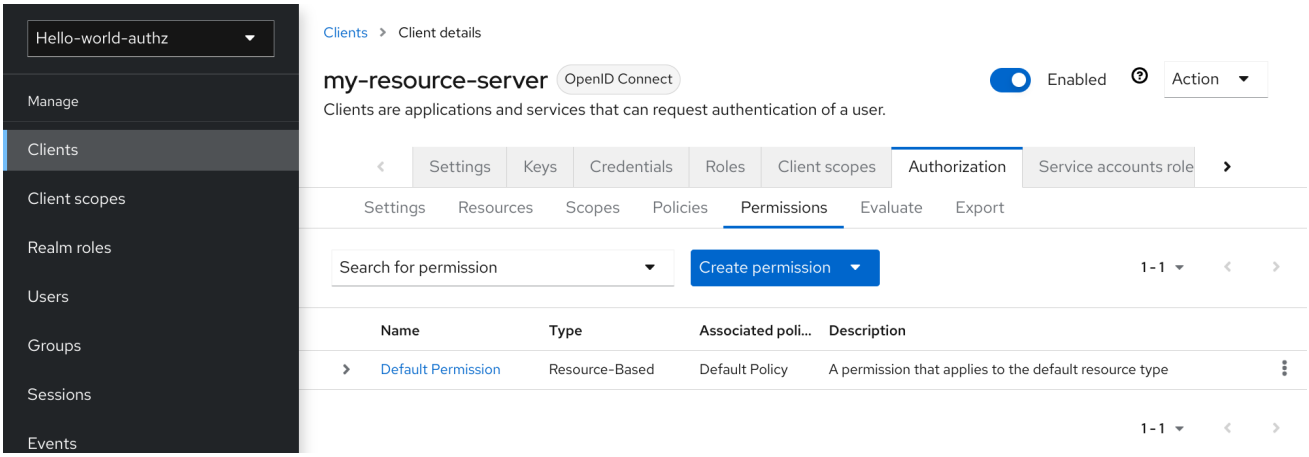
此策略是基于 JavaScript 的策略，定义一个条件，始终授予此策略保护的资源的访问权限。如果点此

策略，您可以看到它定义了一条规则，如下所示：

```
// by default, grants any permission associated with this policy
$evaluation.grant();
```

最后，默认权限称为默认权限，您可以通过 Permissions 标签页来查看它。

默认权限



此权限是基于资源的权限，定义一组适用于给定类型的所有资源的一个或多个策略。

3.3.1. 更改默认配置

您可以通过删除默认资源、策略或权限定义并自行创建自己的来更改默认配置。

默认资源使用 URI 创建，该 URI 使用 /* 模式映射到应用程序中的任何资源或路径。在创建自己的资源、权限和策略前，请确保默认配置不会与您自己的设置冲突。

注意

默认配置定义了一个资源，它映射到应用中的所有路径。如果要向您自己的资源写入权限，请确保删除 Default Resource 或将其 URIS 字段更改为应用程序中更具体的路径。否则，与默认资源关联的策略（默认始终授予访问权限）将允许红帽构建的 Keycloak 授予对任何受保护的资源的访问权限。

3.4. 导出和导入授权配置

可以导出和下载资源服务器（或客户端）的配置设置。您还可以为资源服务器导入现有的配置文件。在您要为资源服务器创建初始配置或更新现有配置时，导入和导出配置文件会很有用。配置文件包含以下的定义：

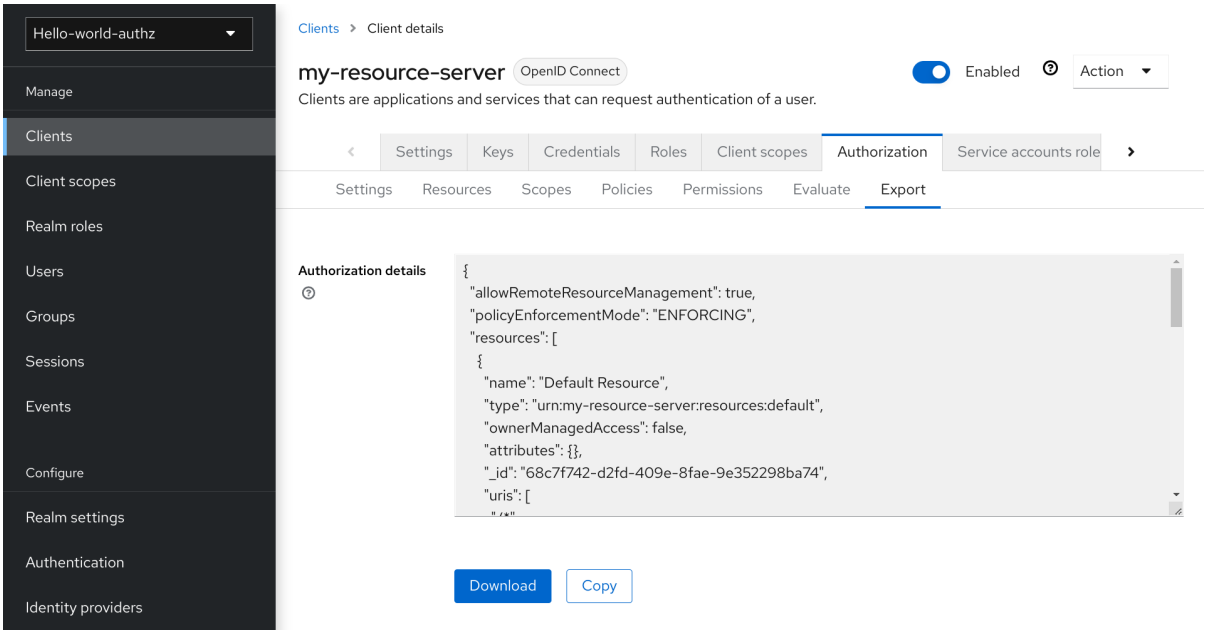
- 保护的资源和范围
- 策略（policy）
- 权限

3.4.1. 导出配置文件

流程

1. 点菜单中的 **Clients**。
2. 点您创建的客户端作为资源服务器。
3. 点 **Export** 选项卡。

导出设置



配置文件以 JSON 格式导出，并以文本区域显示，您可以从其中复制和粘贴。您还可以点 Download 下载配置文件并保存它。

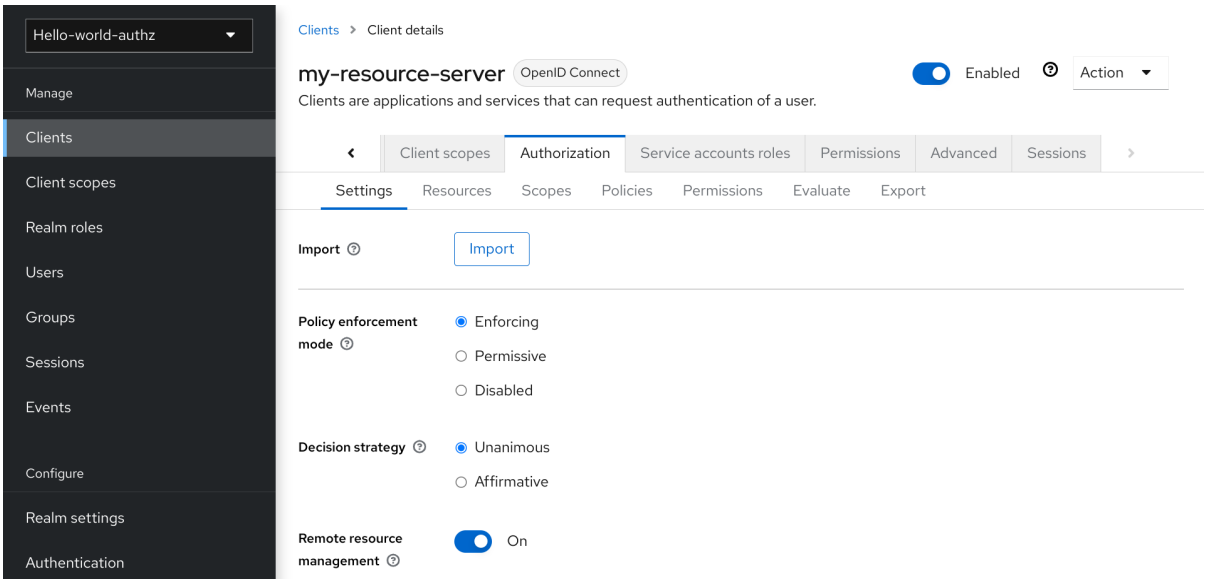
3.4.2. 导入配置文件

您可以为资源服务器导入配置文件。

流程

1. 导航到 Resource Server Settings 页面。

导入设置



2. 点 Import 并选择包含您要导入的配置的文件。

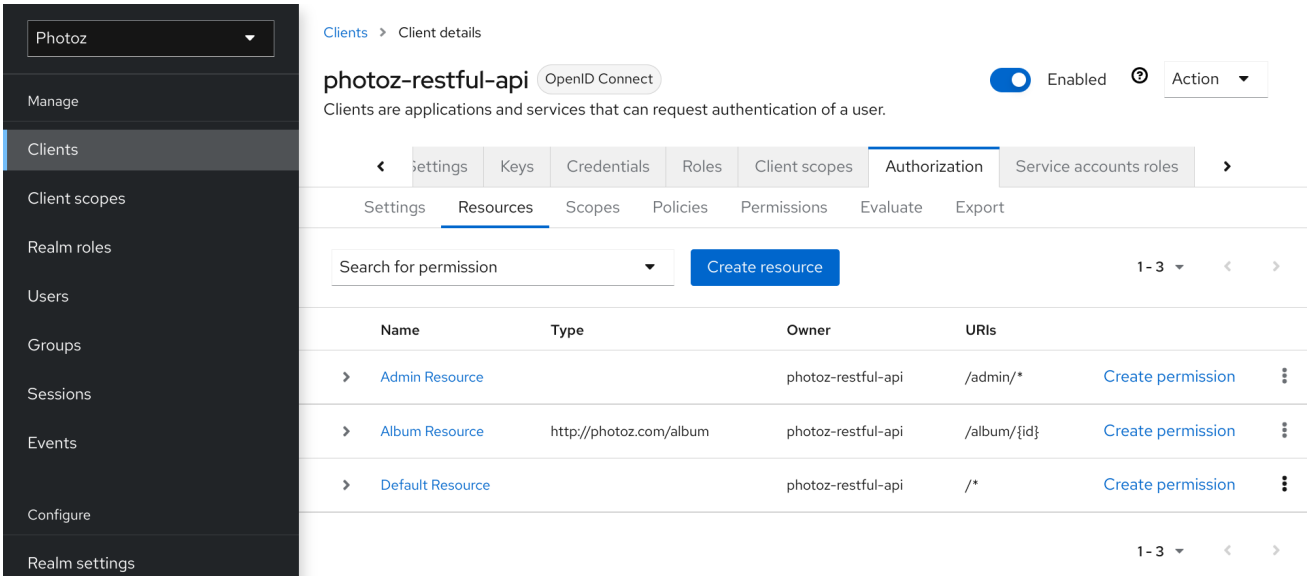
第 4 章 管理资源和范围

资源管理简单且通用。创建资源服务器后，您可以开始创建您要保护的资源和范围。可以通过分别导航到 Resource and Authorization Scopes 选项卡来管理资源和范围。

4.1. 查看资源

在 Resource 页面中，您会看到与资源服务器关联的资源列表。

Resources



资源列表提供有关受保护的资源的信息，例如：

- 类型
- URI
- 所有者
- 关联的范围，如果有

- 关联的权限

从此列表中，您还可以直接为您要为其创建权限的资源单击 **Create Permission** 来创建权限。

注意

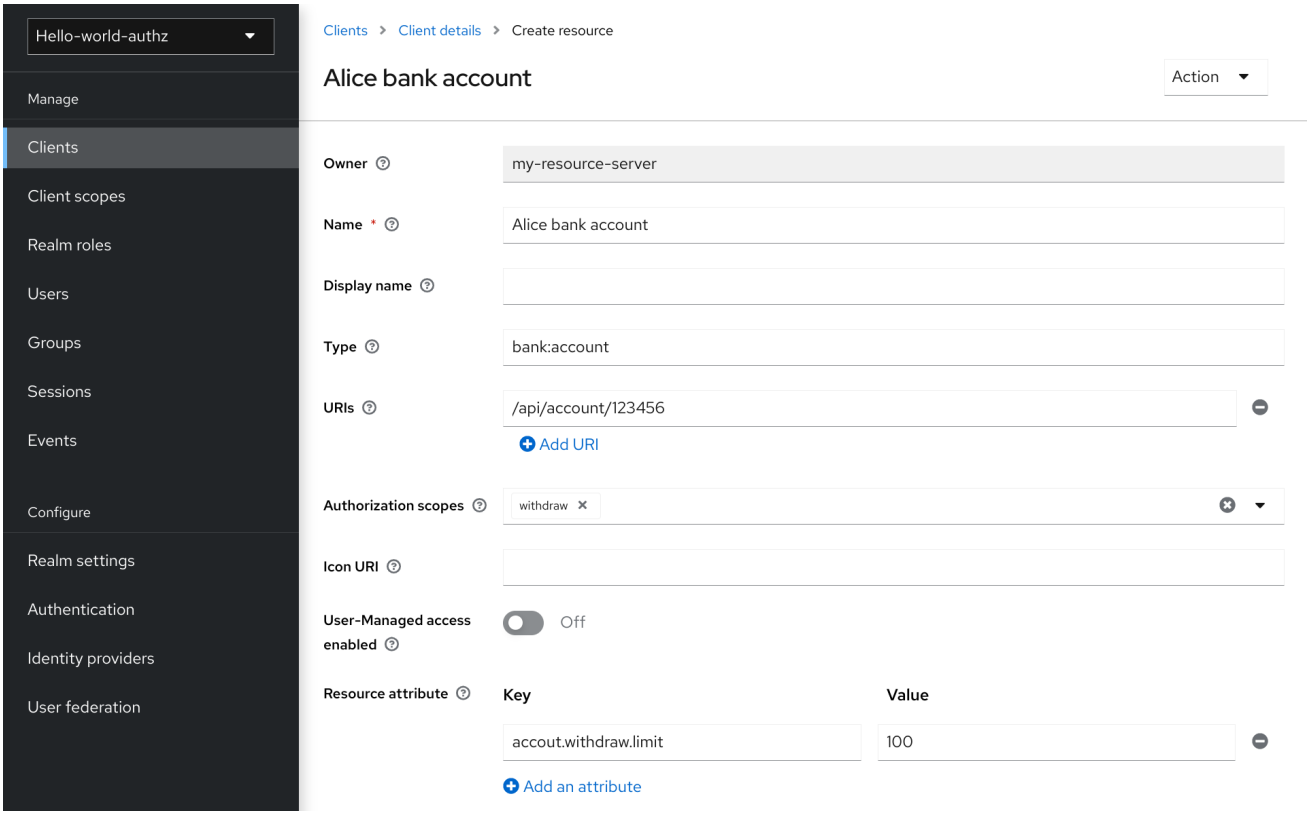
在为资源创建权限前，请确定您已定义了您要与权限关联的策略。

4.2. 创建资源

创建资源非常简单且通用。您的主要关注是您创建的资源的粒度。换句话说，可以创建资源来代表一组或多个资源，定义这些资源的方式对于管理权限至关重要。

若要创建新资源，请单击 **Create resource**。

添加资源



在红帽构建的 Keycloak 中，资源定义了一组对不同类型的资源通用的少量信息，例如：

- Name

  描述此资源的人类可读和唯一字符串。

- 类型

  唯一标识一组或多个资源的类型的字符串。类型是用于对不同资源实例进行分组的字符串。*例如，自动创建的默认资源类型为 urn:resource-server-name:resources:default*

- *URI*

  为资源提供 locations/addresses 的 URIS。对于 HTTP 资源，URIS 通常是用来为这些资源提供服务的相对路径。

- *范围*

  与资源关联的一个或多个范围。

4.2.1. 资源属性

资源可能关联有属性。这些属性可用于提供有关资源的额外信息，并在评估与资源关联的权限时为策略提供额外的信息。

每个属性是一个键值对，其中值可以是一组或多个字符串。通过用逗号分隔每个值，可以为属性定义多个值。

4.2.2. 类型的资源

资源的 type 字段可用于将不同的资源分组在一起，因此可使用一组通用权限进行保护。

4.2.3. 资源所有者

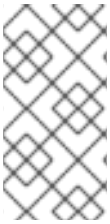
资源也具有所有者。默认情况下，资源由资源服务器所有。

但是，资源也可以与用户关联，因此您可以根据资源所有者创建权限。例如，只允许资源所有者删除或更新给定资源。

4.2.4. 远程管理资源

还通过 保护 API 公开资源管理，以允许资源服务器远程管理其资源。

在使用保护 API 时，可以实施资源服务器来管理其用户拥有的资源。在这种情况下，您可以指定用户标识符，将资源配置为属于特定用户。



注意

红帽构建的 Keycloak 提供对资源资源的完整控制。未来，我们应该能够控制自己的资源，并批准授权请求和管理权限，特别是在使用 UMA 协议时。

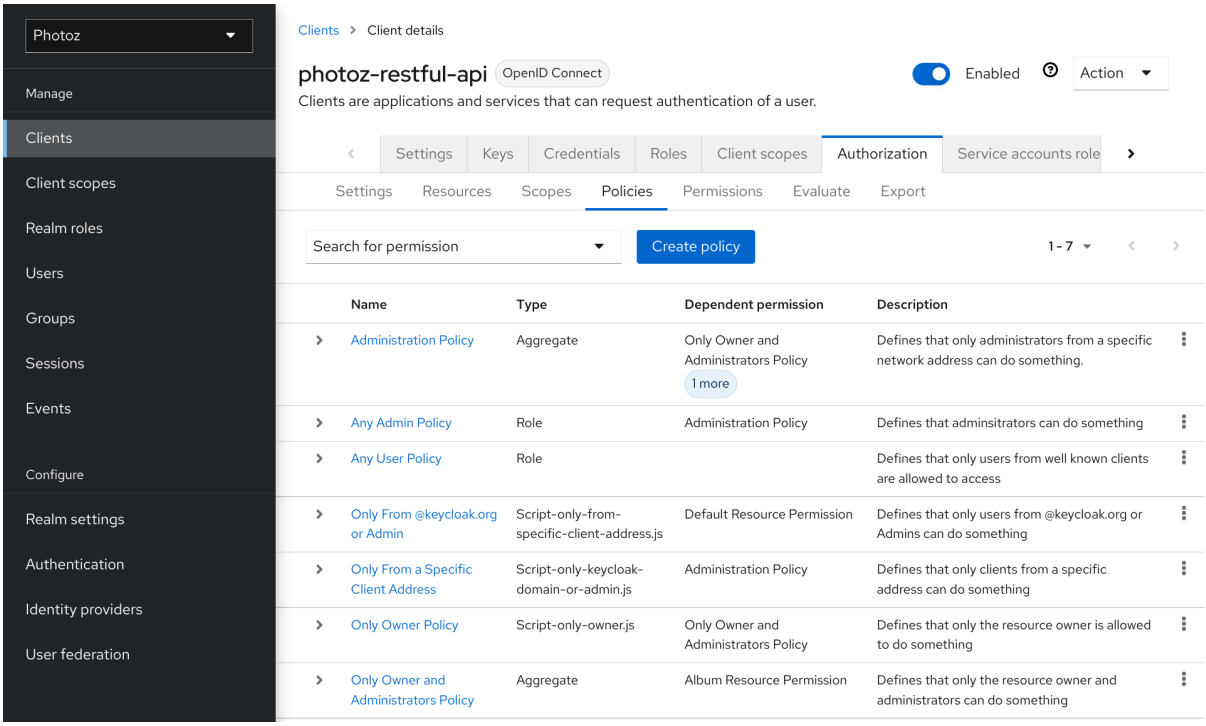
第 5 章 管理策略

如前文所述，策略定义了在授予对对象访问权限前必须满足的条件。

流程

1. 点 Policy 选项卡查看与资源服务器关联的所有策略。

策略（policy）



在此选项卡中，您可以查看之前创建的策略列表，以及创建和编辑策略。

若要创建新策略，请单击 Create policy，然后从列表中选择策略类型。

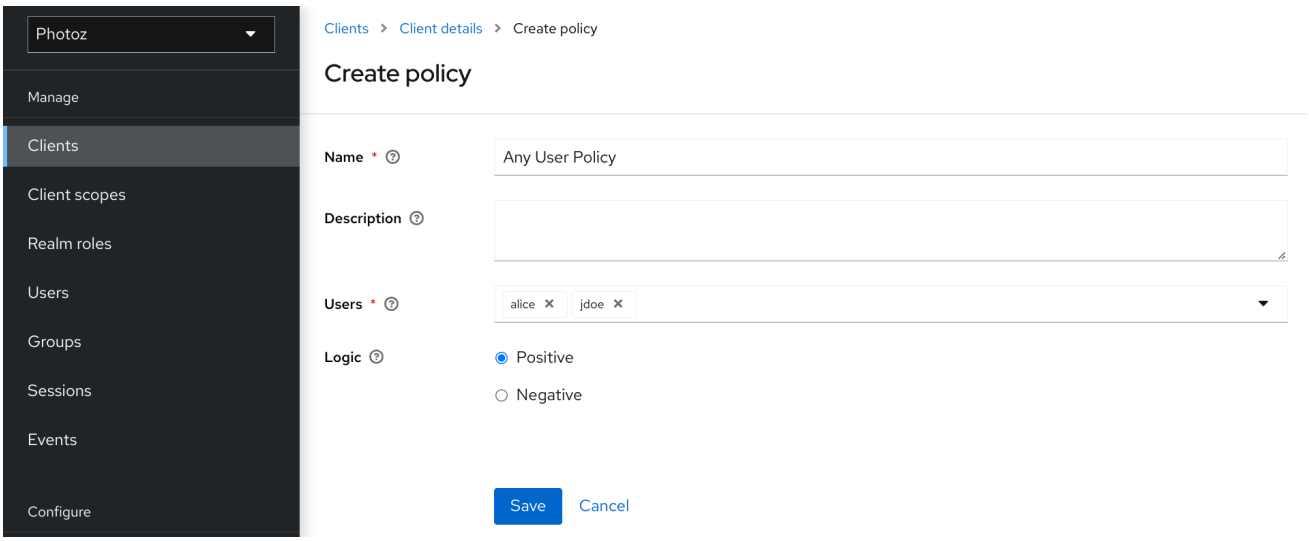
本节描述了每种策略类型的详情。

5.1. 基于用户的策略

您可以使用这类策略为您的权限定义允许一个或多个用户访问对象的条件。

要创建基于用户的新策略，请在策略列表右上角的项目列表中选择 **User**。

添加用户策略



5.1.1. Configuration

- **Name**

  标识策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**

  包含此策略详情的字符串。

- **用户**

  指定此策略授予哪些用户。

- **logic**

  评估其他条件后应用此策略的逻辑。

其他资源

- 正和负逻辑

5.2. 基于角色的策略

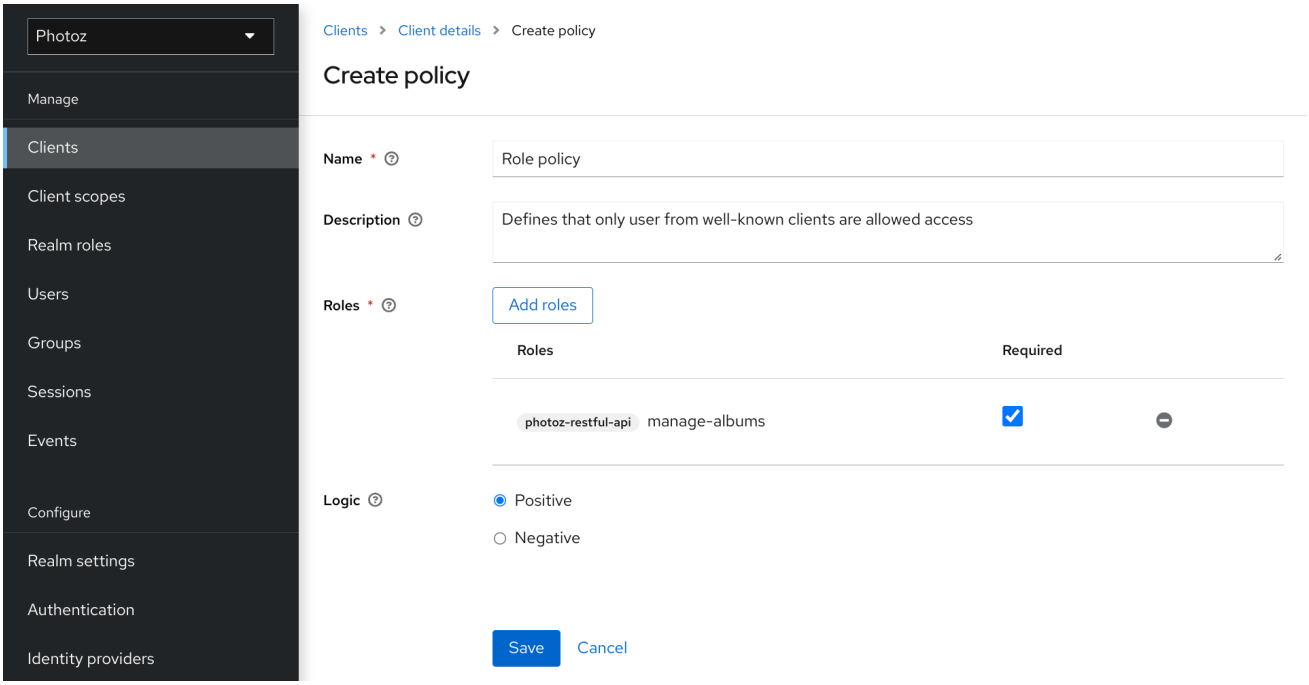
您可以使用这类策略为允许一组或多个角色访问对象的权限定义条件。

默认情况下，添加到此策略的角色没有根据需要指定，如果请求访问的用户被授予了任何这些角色，则策略将授予访问权限。但是，如果要强制执行特定的角色，您可以根据需要指定一个特定的角色为 **required**。您还可以组合所需的和非必需的角色，无论它们是 realm 或 client 角色。

当您需要更多限制基于角色的访问控制(RBAC)时，角色策略很有用，其中必须强制执行特定的角色来授予对对象的访问权限。例如，您可以强制用户必须同意允许客户端应用程序（代表用户执行操作）才能访问用户的资源。您可以使用红帽构建的 Keycloak Client Scope Mapping 来启用同意页面，甚至强制客户端在从红帽构建的 Keycloak 服务器获取访问令牌时显式提供范围。

要创建基于角色的新策略，请从策略类型列表中选择 Role。

添加角色策略



5.2.1. Configuration

- ***Name***

  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- ***描述***

  包含此策略详情的字符串。

- ***realm Roles***

  指定哪些 realm 角色被这个策略允许。

- ***客户端角色***

  指定哪些 client 角色被这个策略允许。若要启用此字段，必须首先选择一个 客户端。

- ***logic***

  评估其他条件后应用此策略的逻辑。

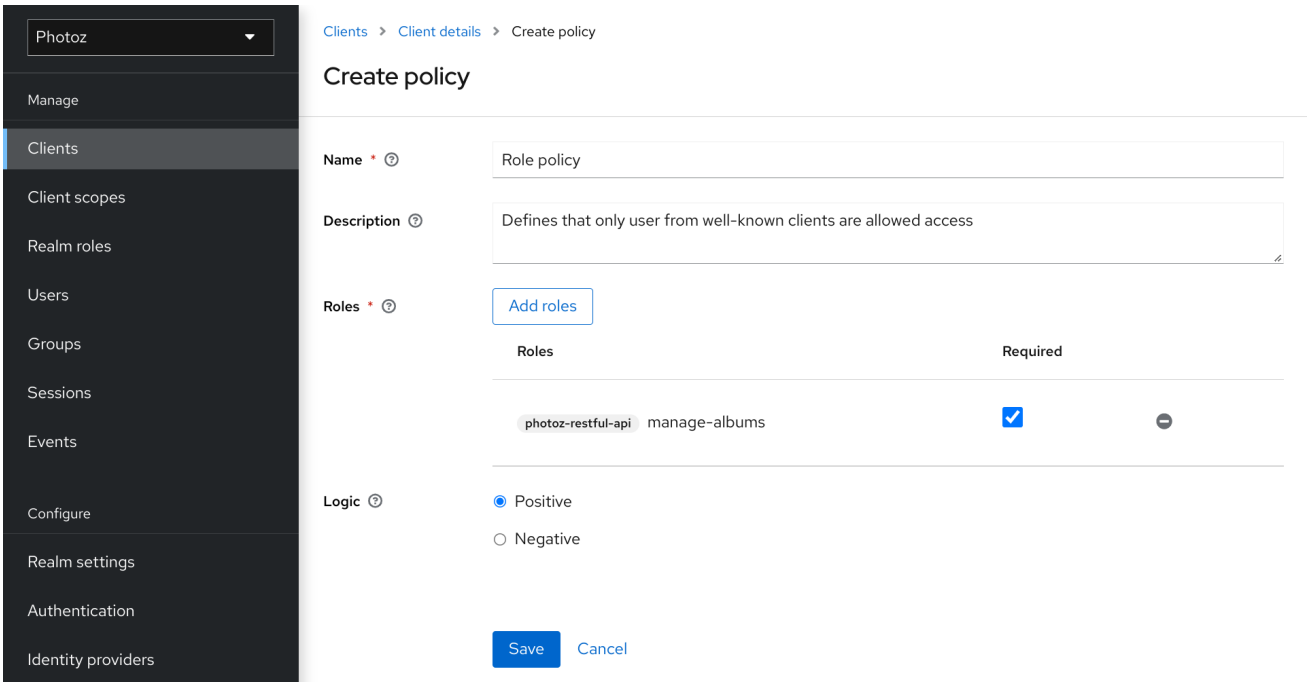
其他资源

- *正和负逻辑*

5.2.2. 根据需要定义角色

*在创建基于角色的策略时，您可以将特定的角色指定为 **Required**。当这样做时，只有在用户请求访问被授予了所有 **required** 角色时才会授予访问权限。**realm** 和 **client** 角色都可以配置，如：*

所需角色示例



要根据需要指定角色，请根据需要选择您要配置的角色 所需的 复选框。

*当策略定义了多个角色时，所需的角色很有用，但只有其中一个角色才是必需的。在这种情况下，您可以组合 **realm** 和 **client** 角色，以便为应用程序启用更加精细的基于角色的访问控制(RBAC)模型。例如，您可以拥有特定于客户端的策略，并且需要与该客户端关联的特定客户端角色。或者，您可以强制仅在存在特定域角色的情况下授予该访问权限。您还可以组合同一策略中的两个方法。*

5.3. 基于 JAVASCRIPT 的策略

警告

如果您的策略实现使用基于属性的访问控制(ABAC)，如以下示例所示，请确保用户无法编辑受保护的属性，并且对应的属性是只读的。请参阅 Threat 模型缓解 章节中的详细信息。

您可以使用这类策略使用 JavaScript 为权限定义条件。它是红帽构建的 Keycloak 支持的基于规则的策略类型，并提供根据 评估 API 编写任何策略的灵活性。

要创建基于 JavaScript 的新策略，请在策略列表右上角的 item 列表中选择 JavaScript。

注意

默认情况下，JavaScript 策略无法上传到服务器。您应该希望直接将 JS 策略部署到服务器，如 JavaScript Providers 所述。

5.3.1. 从部署的 JAR 文件创建 JS 策略

红帽构建的 Keycloak 允许您部署 JAR 文件，以便将脚本部署到服务器。请查看 JavaScript 提供者 以获取更多详细信息。

部署脚本后，您应能够从可用策略提供程序列表中选择部署的脚本。

5.3.2. 例子

5.3.2.1. 从评估上下文中检查属性

以下是使用基于 JavaScript 的策略的简单示例，它使用基于属性的访问控制(ABAC)根据从执行上下文获取的属性来定义条件：

```
const context = $evaluation.getContext();
const contextAttributes = context.getAttributes();
```

```
if (contextAttributes.containsValue('kc.client.network.ip_address', '127.0.0.1')) {
    $evaluation.grant();
}
```

5.3.2.2. 从当前身份检查属性

以下是基于 JavaScript 的策略的简单示例，它使用基于属性的访问控制(ABAC)根据与当前身份关联的属性来定义条件：

```
const context = $evaluation.getContext();
const identity = context.getIdentity();
const attributes = identity.getAttributes();
const email = attributes.getValue('email').asString(0);

if (email.endsWith('@keycloak.org')) {
    $evaluation.grant();
}
```

其中，这些属性从授权请求中使用的令牌中定义任何声明。

5.3.2.3. 检查授予当前身份的角色

您还可以在策略中使用基于角色的访问控制(RBAC)。在以下示例中，我们将检查用户是否被授予了 keycloak_user 域角色：

```
const context = $evaluation.getContext();
const identity = context.getIdentity();

if (identity.hasRealmRole('keycloak_user')) {
    $evaluation.grant();
}
```

或者，您可以检查用户是否被授予了 my-client-role 客户端角色，其中 my-client 是客户端应用程序的客户端 ID：

```
const context = $evaluation.getContext();
const identity = context.getIdentity();

if (identity.hasClientRole('my-client', 'my-client-role')) {
    $evaluation.grant();
}
```

5.3.2.4. 检查授予用户的角色

检查授予用户的域角色：

```
const realm = $evaluation.getRealm();

if (realm.isUserInRealmRole('marta', 'role-a')) {
    $evaluation.grant();
}
```

或 为用户授予了客户端角色：

```
const realm = $evaluation.getRealm();

if (realm.isUserInClientRole('marta', 'my-client', 'some-client-role')) {
    $evaluation.grant();
}
```

5.3.2.5. 检查授予组的角色

检查授予组的域角色：

```
const realm = $evaluation.getRealm();

if (realm.isGroupInRole('/Group A/Group D', 'role-a')) {
    $evaluation.grant();
}
```

5.3.2.6. 将任意声明推送到资源服务器

将任意声明推送到资源服务器，以提供有关如何强制实施权限的额外信息：

```
const permission = $evaluation.getPermission();

// decide if permission should be granted

if (granted) {
    permission.addClaim('claim-a', 'claim-a');
    permission.addClaim('claim-a', 'claim-a1');
    permission.addClaim('claim-b', 'claim-b');
}
```

5.3.2.7. 检查组成员资格

```
const realm = $evaluation.getRealm();
```

```
if (realm.isUserInGroup('marta', '/Group A/Group B')) {
    $evaluation.grant();
}
```

5.3.2.8. 混合不同的访问控制机制

您还可以使用多种访问控制机制的组合。以下示例演示了如何在同一策略中使用角色(RBAC)和 claims/attributes (ABAC)检查。在这种情况下，我们将检查用户是否被授予了 admin 角色，还是来自 keycloak.org 域的电子邮件：

```
const context = $evaluation.getContext();
const identity = context.getIdentity();
const attributes = identity.getAttributes();
const email = attributes.getValue('email').asString(0);

if (identity.hasRealmRole('admin') || email.endsWith('@keycloak.org')) {
    $evaluation.grant();
}
```

注意

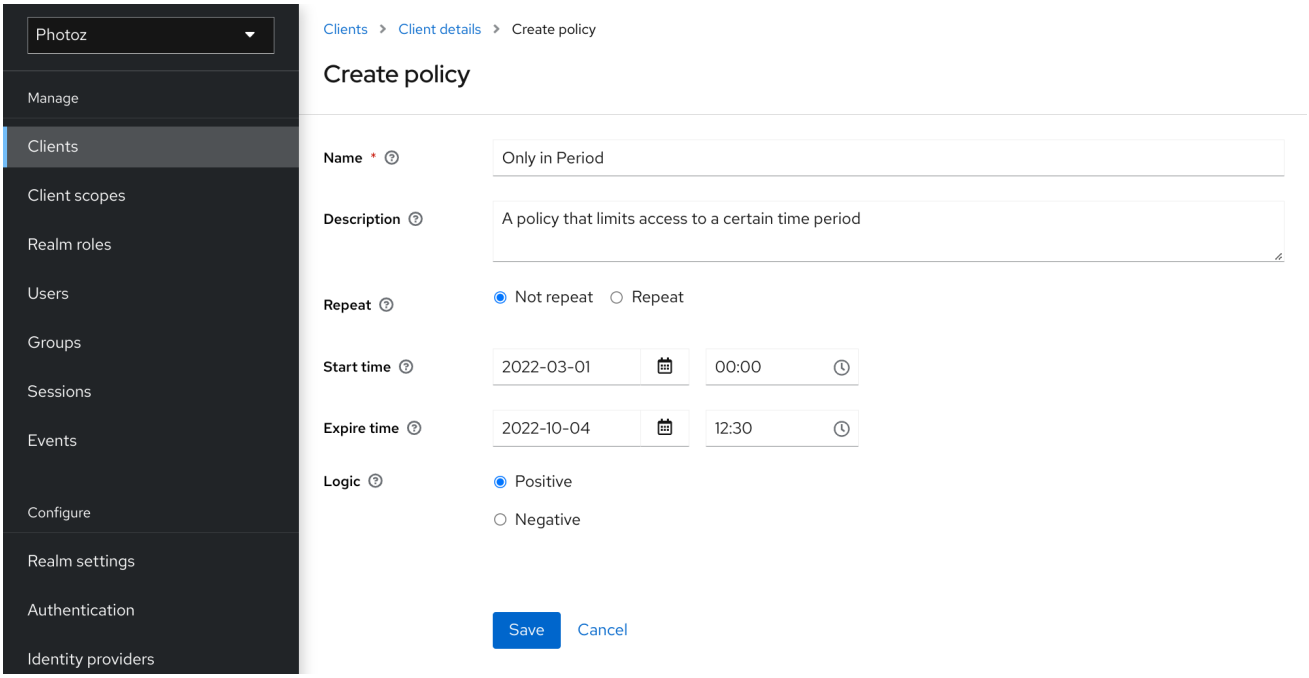
在编写自己的规则时，请记住 $evaluation 对象是实施 org.keycloak.authorization.policy.evaluation.Evaluation 的对象。有关可以从此接口访问的内容的更多信息，请参阅 评估 API。

5.4. 基于时间的策略

您可以使用这类策略为您的权限定义时间条件。

要创建基于时间的新策略，请在策略列表右上角的项目列表中选择 Time。

添加时间策略



5.4.1. Configuration

- ***Name***

  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- ***描述***

  包含此策略详情的字符串。

- ***开始时间***

  定义 不能 授予访问权限的时间。只有在当前日期/时间高于这个值或等于这个值时，才会授予权限。

- ***过期时间***

  定义 不能 授予访问权限的时间。只有在当前日期/时间早于或等于这个值时，才会授予权限。选择 Repeat 在指定日期(几号 ,Month, Year,Hour 或 Minute)上授予访问权限。

- *Month 几天*

  定义必须授予访问权限的月日。您还可以指定日期范围。在这种情况下，只有在月的当前日期介于或等于指定的值之间时，才会授予权限。

- *月*

  定义必须授予访问权限的月份。您还可以指定一系列月。在这种情况下，只有在当前月份介于或等于指定两个值时才授予权限。

- *年*

  定义必须授予访问权限的年份。您还可以指定年范围。在这种情况下，只有在当前年间或等于指定的值时，才会授予权限。

- *小时*

  定义必须授予访问权限的小时。您还可以指定小时范围。在这种情况下，只有在当前小时之间或等于指定两个值时才授予权限。

- *minute*

  定义必须授予访问权限的时间。您还可以指定分钟范围。在这种情况下，只有在当前分钟之间或等于指定的值时，才会授予权限。

- *logic*

  评估其他条件后应用此策略的逻辑。

只有在满足所有条件时才授予访问权限。红帽构建的 Keycloak 将根据每个状况的结果执行 AND。

其他资源

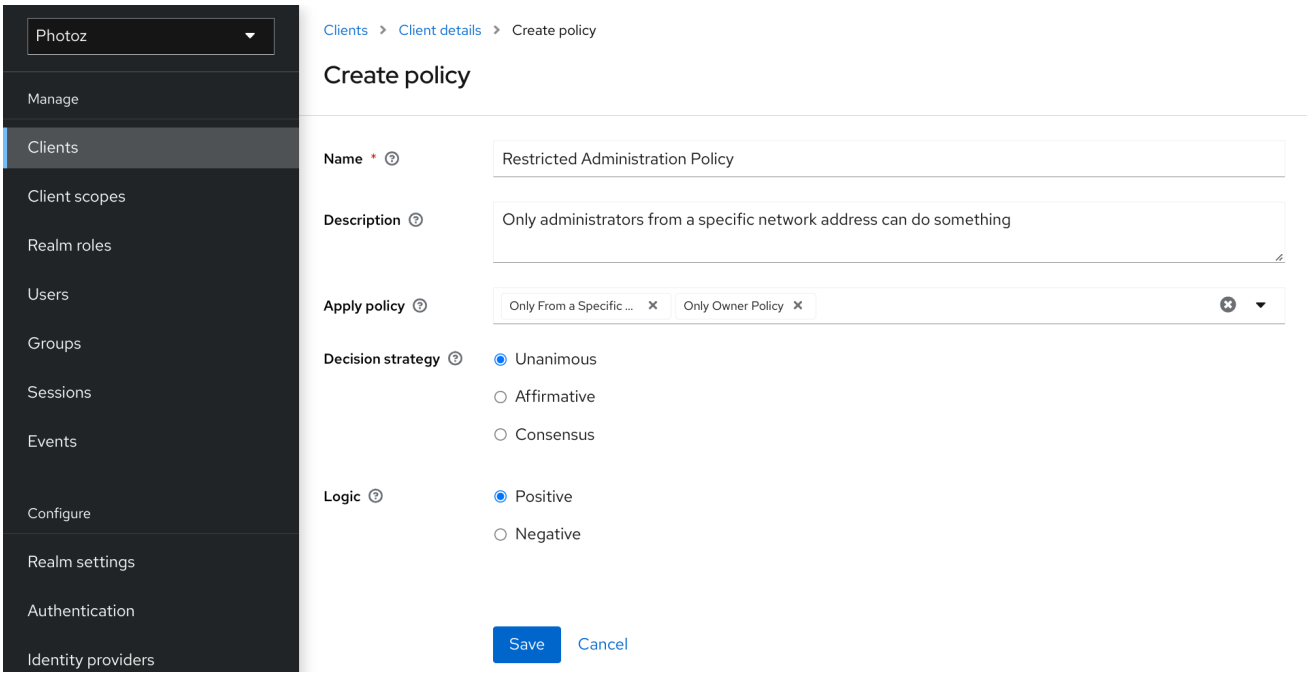
- *正和负逻辑*

5.5. 聚合策略

如前文所述，红帽构建的 Keycloak 允许您构建策略，它是一个称为策略聚合的概念。您可以使用策略聚合来重复使用现有策略来构建更复杂的策略，并使您的权限更加分离在处理授权请求期间评估的策略。

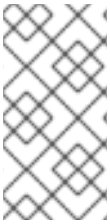
要创建新的聚合策略，请在策略列表右上角的 item 列表中选择 Aggregated。

添加一个聚合策略



假设有一个名为 Confidential 资源的资源，该资源只能由 keycloak.org 域中的用户和特定范围的 IP 地址访问。您可以使用这两个条件创建单一策略。但是，无论原始网络是什么，您想要重复使用此策略的域部分来应用到运行的权限。

您可以为域和网络条件创建单独的策略，并根据这两个策略的组合创建第三个策略。通过聚合策略，您可以自由组合其他策略，然后将新的聚合策略应用到您想要的任何权限。



注意

在创建聚合策略时，请注意，您不会引入策略之间的循环参考或依赖项。如果检测到循环依赖项，则无法创建或更新策略。

5.5.1. Configuration

- **Name**

  描述该策略的人类可读和唯一字符串。我们强烈建议您使用与您的业务和安全要求密切相关的名称，因此您可以更轻松地识别它们，同时了解它们的含义。

- **描述**

  一个字符串，其中包含有关此策略的更多详情。

- **应用策略**

  定义一组一个或多个与聚合策略关联的策略。要关联策略，您可以通过选择您要创建的策略类型来选择现有策略或创建新策略。

- **决策策略**

  此权限的决策策略。

- **logic**

  评估其他条件后应用此策略的逻辑。

其他资源

- 正和负逻辑

5.5.2. 聚合策略的决策策略

在创建聚合策略时，您还可以定义用于根据每个策略的结果确定最终决策的决策。

- **unanimous**

  如果没有提供任何策略，则默认策略。在这种情况下，所有策略都必须评估为最终决策的正决定。

- ***Affirmative***

  在这种情况下，至少一个 策略必须评估为正决定，以便最终决定也是正的。

- ***consensus***

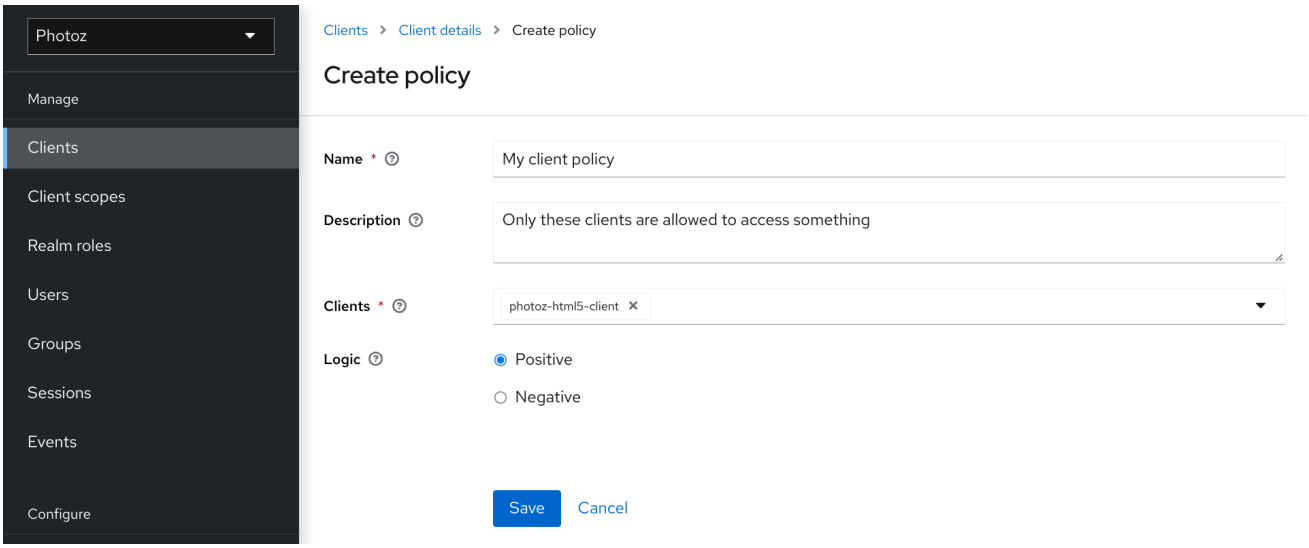
  在这种情况下，正决策的数量必须大于负决策的数量。如果正和负决策的数量相同，则最终决策将是负数。

5.6. 基于客户端的策略

您可以使用这类策略为允许一个或多个客户端访问对象的权限定义条件。

要创建基于客户端的新策略，请从策略类型列表中选择 Client。

添加客户端策略



5.6.1. Configuration

- ***Name***

标识策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- 描述

  包含此策略详情的字符串。

- 客户端

  指定此策略根据给定组策略访问哪些客户端。

- logic

  评估其他条件后应用此策略的逻辑。

其他资源

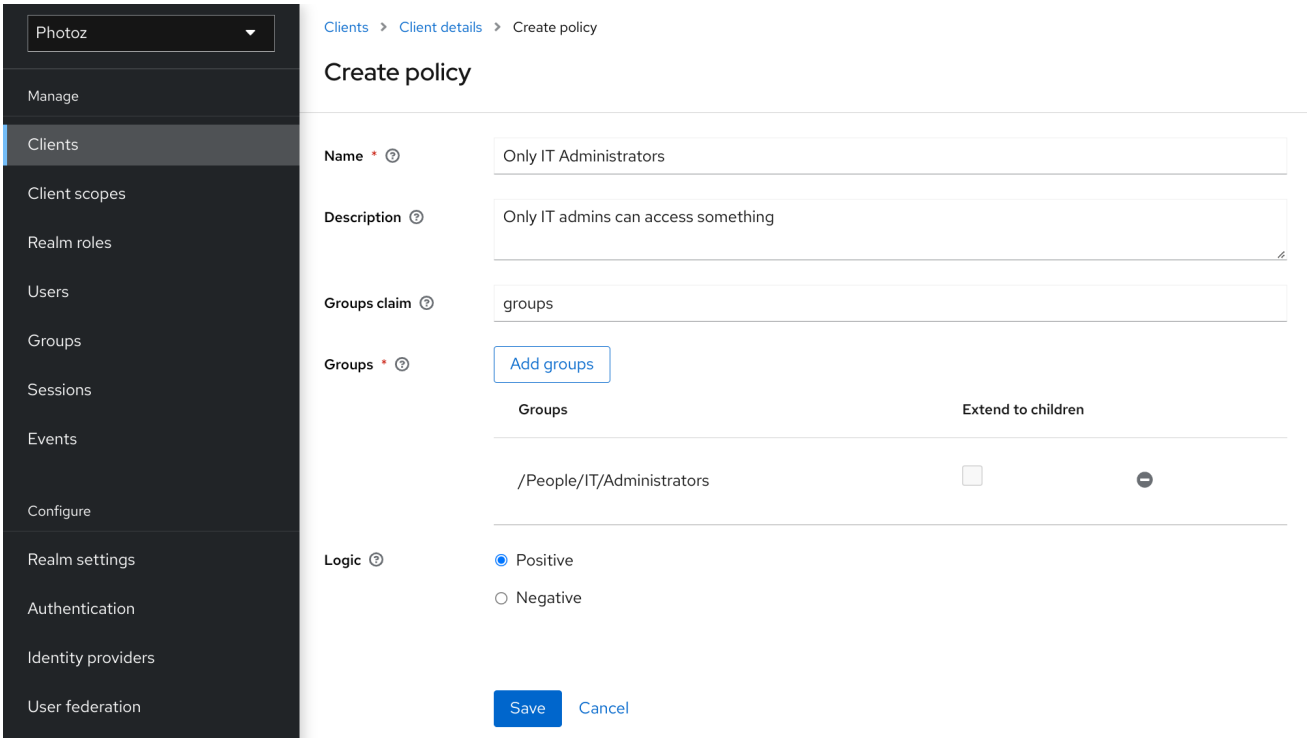
- 正和负逻辑

5.7. 基于组的策略

您可以使用这种类型的策略为您的权限定义一组一个或多个组（及其层次结构）的条件。

要创建基于组的新策略，请从策略类型列表中选择 Group。

组策略



5.7.1. Configuration

- ***Name***

  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- ***描述***

  包含此策略详情的字符串。

- *组声明*

  指定令牌中的声明名称，其中包含组名称和/或路径。通常，授权请求会根据之前向代表某些用户所发出的客户端的 ID 令牌或访问令牌进行处理。如果定义，令牌必须包含此策略要获取用户所属的组的声明。如果没有定义，则从您的域配置中获取用户的组。

- *组*

  允许您选择在评估权限时应由此策略实施的组。添加组后，您可以通过将复选框 扩展为 Children 来扩展对组 的子项的访问。如果没有标记，则访问限制仅适用于所选组。

- **logic**

  评估其他条件后应用此策略的逻辑。

其他资源

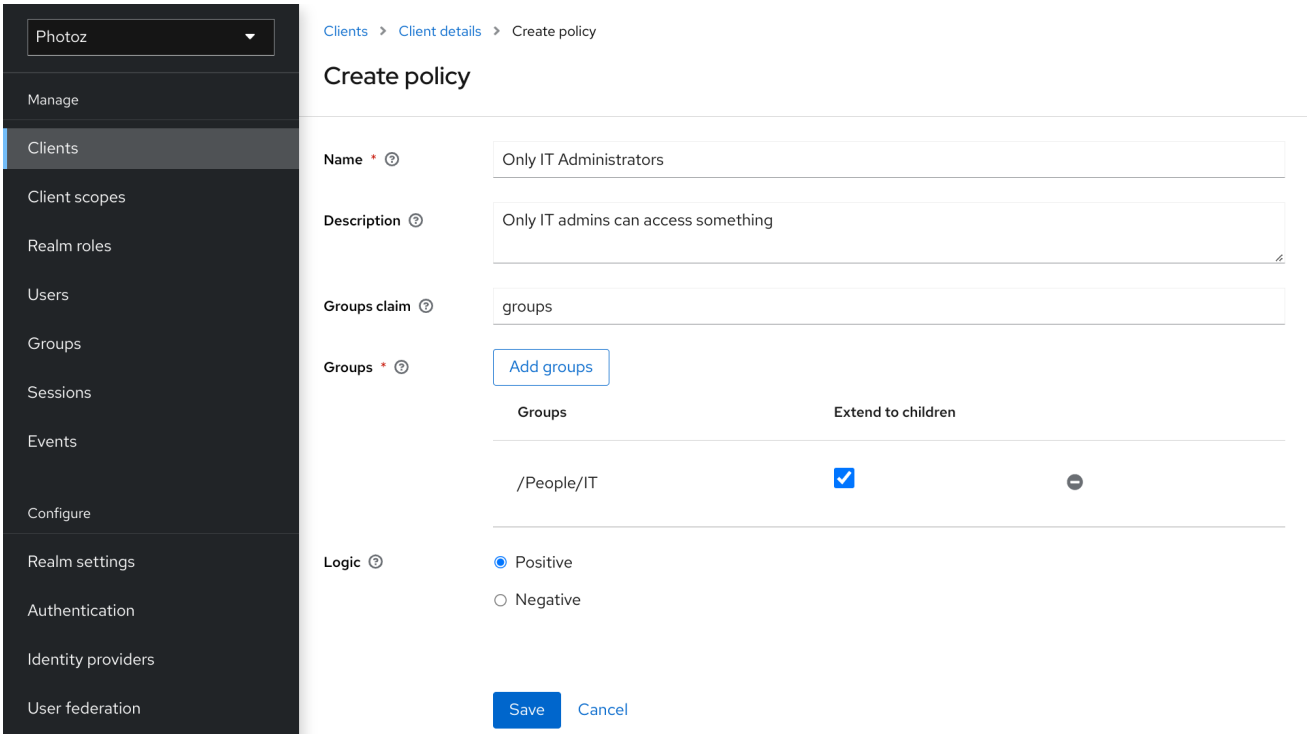
- 正和负逻辑

5.7.2. 扩展对子组的访问

默认情况下，当您向此策略添加组时，访问限制将仅适用于所选组的成员。

在某些情况下，可能需要仅允许访问组本身，而是允许访问层次结构中的任何子组。对于添加的任何组，您可以将复选框 扩展为 **Children**，以扩展对子组的访问。

扩展对子组的访问



在上例中，该策略被授予对任何 IT 用户成员或其任何子项的访问权限。

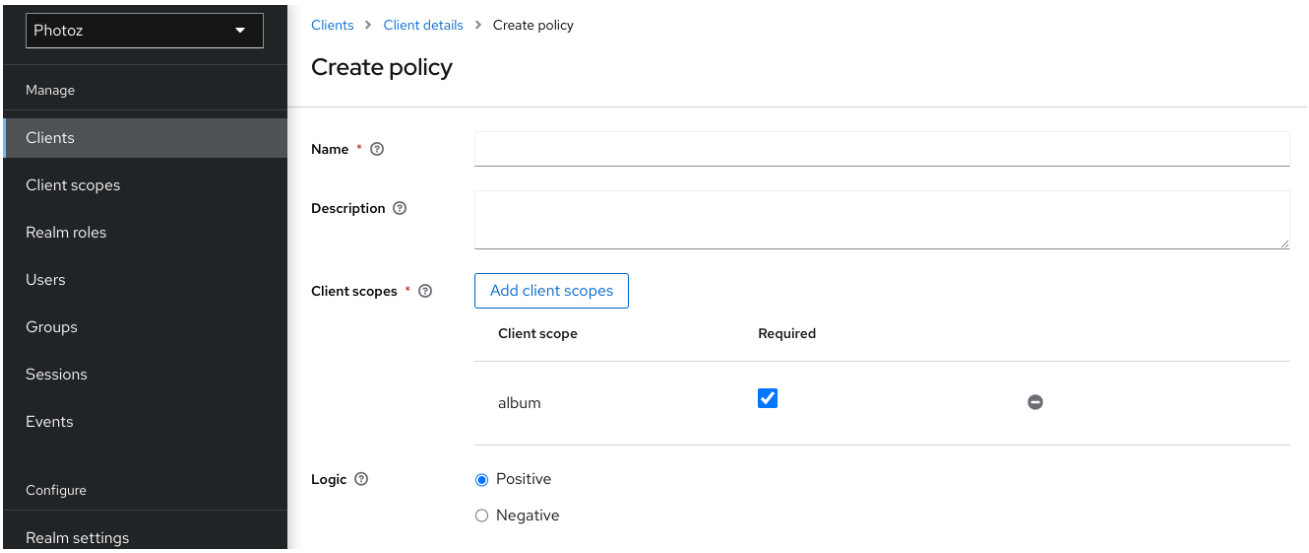
5.8. 基于客户端范围的策略

您可以使用这种类型的策略为您的权限定义允许一个或多个客户端范围访问对象的条件。

默认情况下，添加到此策略的客户端范围没有根据需要指定，如果请求访问的客户端被授予任何客户端范围，则策略将授予访问权限。但是，如果要强制执行特定的客户端范围，您可以指定一个特定的客户端范围为 **required**。

要创建新的客户端范围策略，请从策略类型列表中选择 **Client Scope**。

添加客户端范围策略



5.8.1. Configuration

- **Name**

  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**

包含此策略详情的字符串。

- 客户端范围

  指定此策略允许哪些客户端范围。

- logic

  评估其他条件后应用此策略的逻辑。

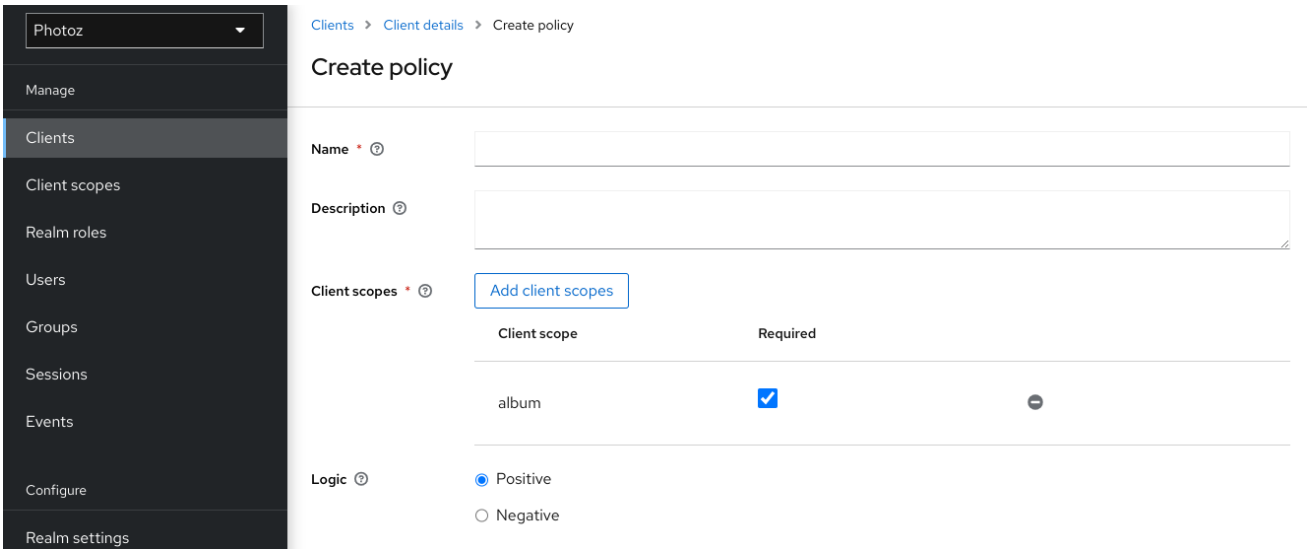
其他资源

- 正和负逻辑

5.8.2. 根据需要定义客户端范围

在创建基于客户端范围的策略时，您可以将特定的客户端范围指定为 **Required**。当这样做时，只有在客户端请求访问被授予了所有 **required** 客户端范围时才会授予访问权限。

所需的客户端范围示例



要根据需要指定客户端范围，请根据需要选择您要配置的客户端范围 所需的 复选框。

当您的策略定义了多个客户端范围时，所需的客户端范围很有用，但只有其中一个子集才是必需的。

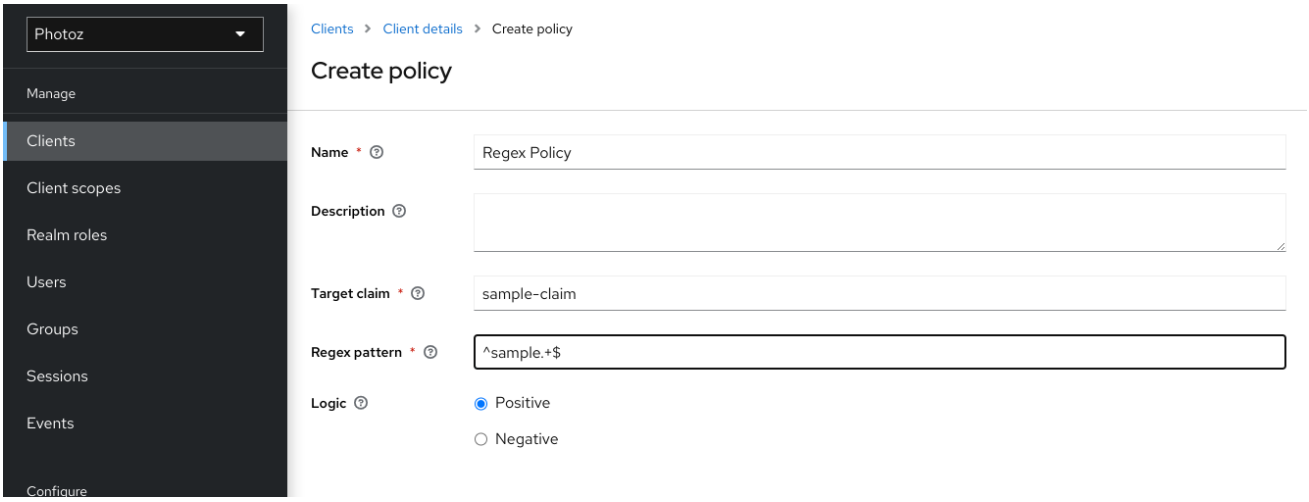
5.9. 基于正则表达式的策略

您可以使用这种类型的策略为您的权限定义正则表达式条件。

要创建基于 regex 的新策略，请从策略类型列表中选择 Regex。

此策略解析当前身份可用的属性。

添加 Regex 策略



5.9.1. Configuration

- **Name**

  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**

  包含此策略详情的字符串。

- 目标声明

  指定令牌中的目标声明的名称。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。如果目标声明引用 JSON 对象，则第一个路径（例如，联系）应映射到包含 JSON 对象的属性名称。

- 正则表达式模式

  指定正则表达式模式。

- logic

  评估完其他条件后应用此策略的 Logic。

5.10. 正和负逻辑

策略可以使用正或负逻辑进行配置。简而言之，您可以使用此选项来定义策略结果是否应保留原样还是负效果。

例如，假设您想创建一个策略，其中只有 未 授予特定角色的用户才被授予访问权限。在这种情况下，您可以使用该角色创建基于角色的策略，并将其 Logic 字段设置为 Negative。如果您保留 Positive （这是默认行为），策略结果将保留原样。

5.11. 策略评估 API

当使用 JavaScript 编写基于规则的策略时，红帽构建的 Keycloak 提供了一个评估 API，它提供有用的信息，以帮助确定是否应授予权限。

此 API 由几个提供您访问信息的接口组成，例如

- 正在评估的权限，代表正在请求的资源和范围。

- 与请求的资源关联的属性

- 运行时环境以及与执行上下文关联的任何其他属性
- 有关组成员资格和角色等用户的信息

主接口是 org.keycloak.authorization.policy.evaluation.Evaluation，它定义了以下合同：

```
public interface Evaluation {

    /**
     * Returns the {@link ResourcePermission} to be evaluated.
     *
     * @return the permission to be evaluated
     */
    ResourcePermission getPermission();

    /**
     * Returns the {@link EvaluationContext}. Which provides access to the whole evaluation
runtime context.
     *
     * @return the evaluation context
     */
    EvaluationContext getContext();

    /**
     * Returns a {@link Realm} that can be used by policies to query information.
     *
     * @return a {@link Realm} instance
     */
    Realm getRealm();

    /**
     * Grants the requested permission to the caller.
     */
    void grant();

    /**
     * Denies the requested permission.
     */
    void deny();
}
```

在处理授权请求时，红帽构建的 Keycloak 会在 评估 任何策略前创建一个评估实例。然后，此实例被传递给每个策略，以确定访问是 GRANT 还是 DENY。

策略通过调用 评估 实例的 grant （） 或 deny （） 方法来确定这一点。默认情况下，评估 实例的状态被拒绝，这意味着您的策略必须明确调用 grant （） 方法，以指示应授予其权限的策略评估引擎。

其他资源

- Java 文档文档.

5.11.1. 评估上下文

评估上下文在评估期间为策略提供有用的信息。

```
public interface EvaluationContext {

    /**
     * Returns the {@link Identity} that represents an entity (person or non-person) to which the permissions must be granted, or not.
     *
     * @return the identity to which the permissions must be granted, or not
     */
    Identity getIdentity();

    /**
     * Returns all attributes within the current execution and runtime environment.
     *
     * @return the attributes within the current execution and runtime environment
     */
    Attributes getAttributes();
}
```

在这个界面中，策略可以获取：

- 经过身份验证的 身份
- 有关执行上下文和运行时环境的信息

Identity 基于与授权请求一起发送的 OAuth2 访问令牌构建，此构造可以访问从原始令牌中提取的所有声明。例如，如果您使用 Protocol Mapper 在 OAuth2 访问令牌中包含自定义声明，您也可以从策略访问此声明，并使用它来构建您的条件。

EvaluationContext 还可让您访问与执行和运行时环境相关的属性。目前，只有几个内置属性。

表 5.1. 执行和运行时属性

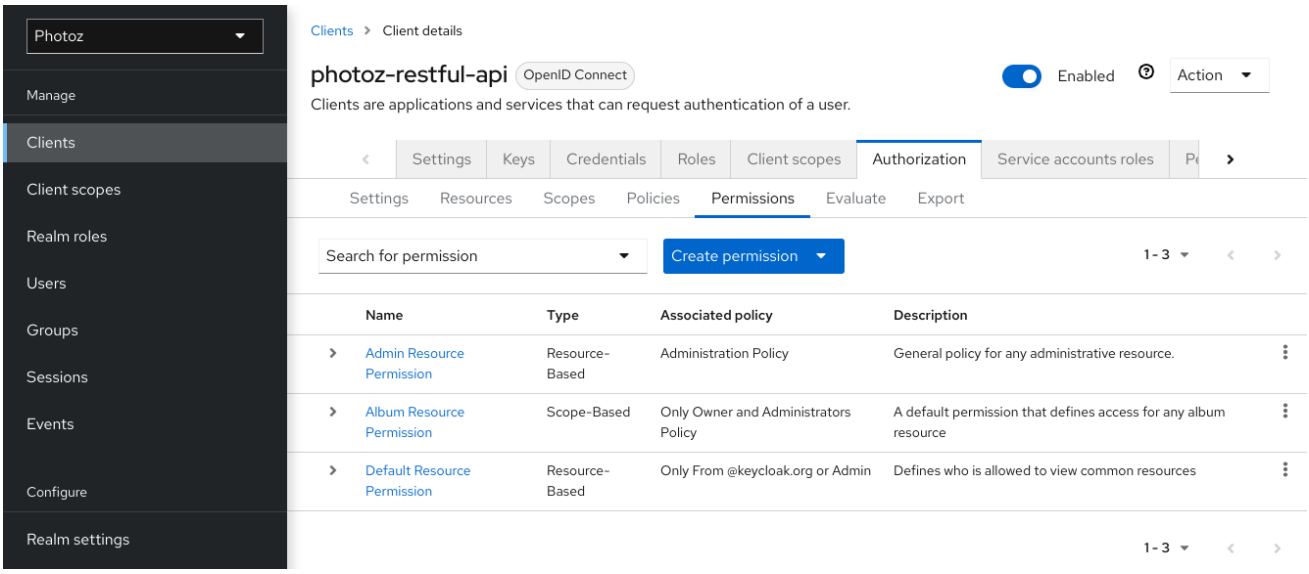
Name	描述	类型
kc.time.date_time	当前日期和时间	字符串.格式 **MM/dd/yyyy hh:mm:ss**
kc.client.network.ip_address	客户端的 IPv4 地址	字符串
kc.client.network.host	客户端的主机名	字符串
kc.client.id	客户端 ID	字符串
kc.client.user_agent	'User-Agent' HTTP 标头的值	String[]
kc.realm.name	域的名称	字符串

第 6 章 管理权限

权限关联正在保护的对象以及必须评估的策略，以决定是否授予访问权限。

在创建您要保护的资源以及您要用来保护这些资源的策略后，您可以开始管理权限。要管理权限，请在编辑资源服务器时单击 **Permissions** 选项卡。

权限



可以创建权限来保护两种主要对象：

- **Resources**
- 范围

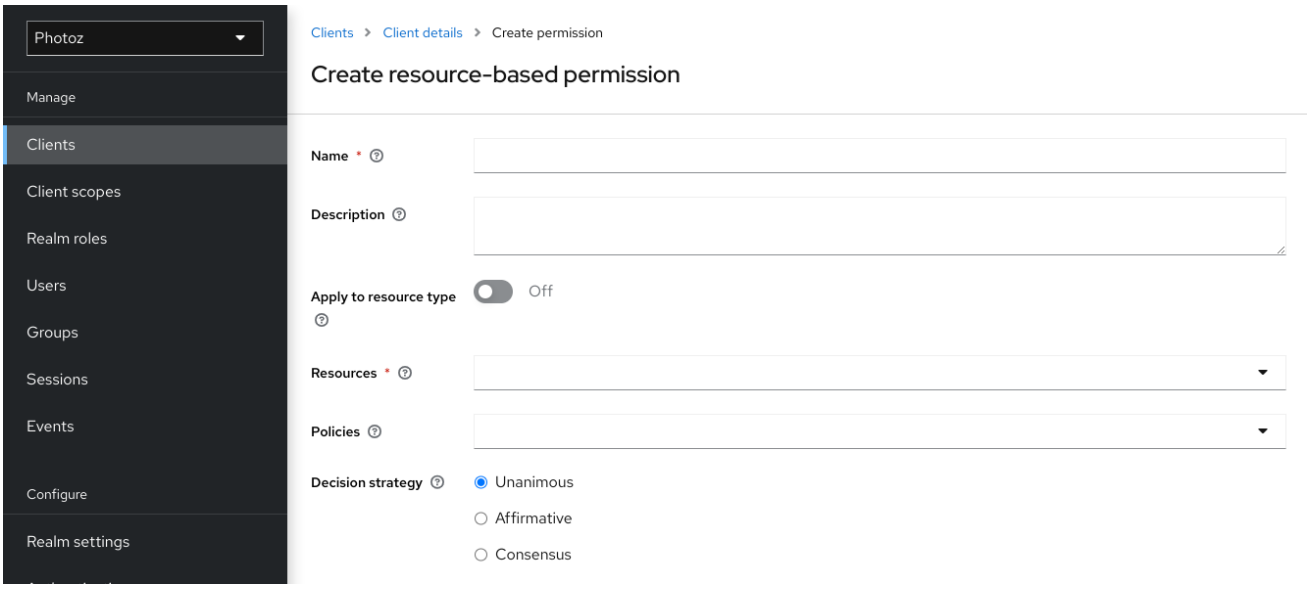
要创建权限，请从权限列表右上角的 item 列表中选择您要创建的权限类型。以下小节更详细地描述了这两类对象。

6.1. 创建基于资源的权限

基于资源的权限定义了一组一个或多个资源，以使用一组一个或多个授权策略来保护。

要创建基于资源的新权限，请从 Create permissions 下拉菜单中选择 Create resource-based permissions。

添加资源权限



6.1.1. Configuration

- **Name**

  描述权限的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**

  包含此权限详情的字符串。

- **Apply To Resource Type**

  指定权限是否应用到给定类型的所有资源。选择此字段时，会提示您输入要保护的资源类型。

  - 资源类型

定义要保护的资源类型。定义后，针对与该类型匹配的所有资源评估此权限。

- Resources

  定义要保护的一个或多个资源的集合。

- policy

  定义一组与权限关联的一个或多个策略。要关联策略，您可以通过选择您要创建的策略类型来选择现有策略或创建新策略。

- 决策策略

  此权限的决策策略。

6.1.2. 类型的资源权限

资源权限也可用于定义要应用到给定类型的所有资源的策略。当您有共享通用访问要求和约束的资源时，这种基于资源的权限形式很有用。

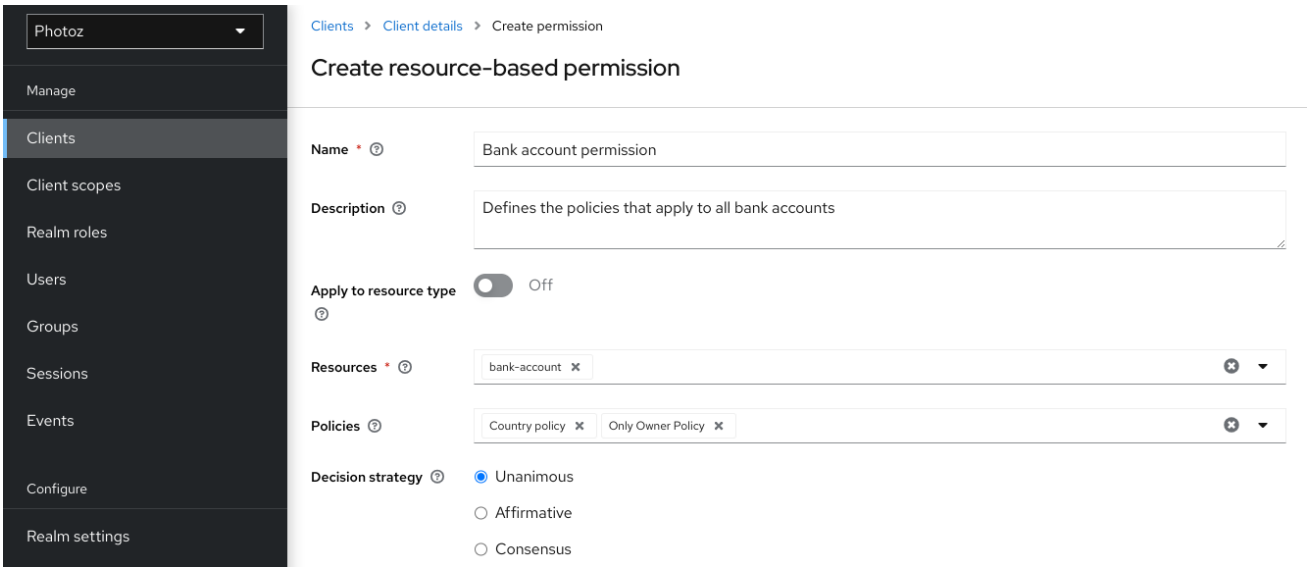
通常，应用中的资源可以根据它们封装的数据或提供的功能进行分类（或键入）。例如，一个金融应用程序可以管理不同的银行客户，各家都属于特定客户。虽然它们是不同的银行帐户，它们共享了银行组织范围定义的共同安全要求和限制。使用 typed 资源权限，您可以定义适用于所有银行帐户的通用策略，例如：

- 只有所有者可以管理其帐户
- 只允许从所有者的国家和/或区域进行访问
- 强制特定的验证方法

要创建类型的资源权限，请在创建新的基于资源的权限时点 Apply to Resource Type。将 Apply to Resource Type 设置为 On 时，您可以指定您要保护的类型以及要应用到的策略，以使用您指定的类型

管理对所有资源的访问。

类型的资源权限示例

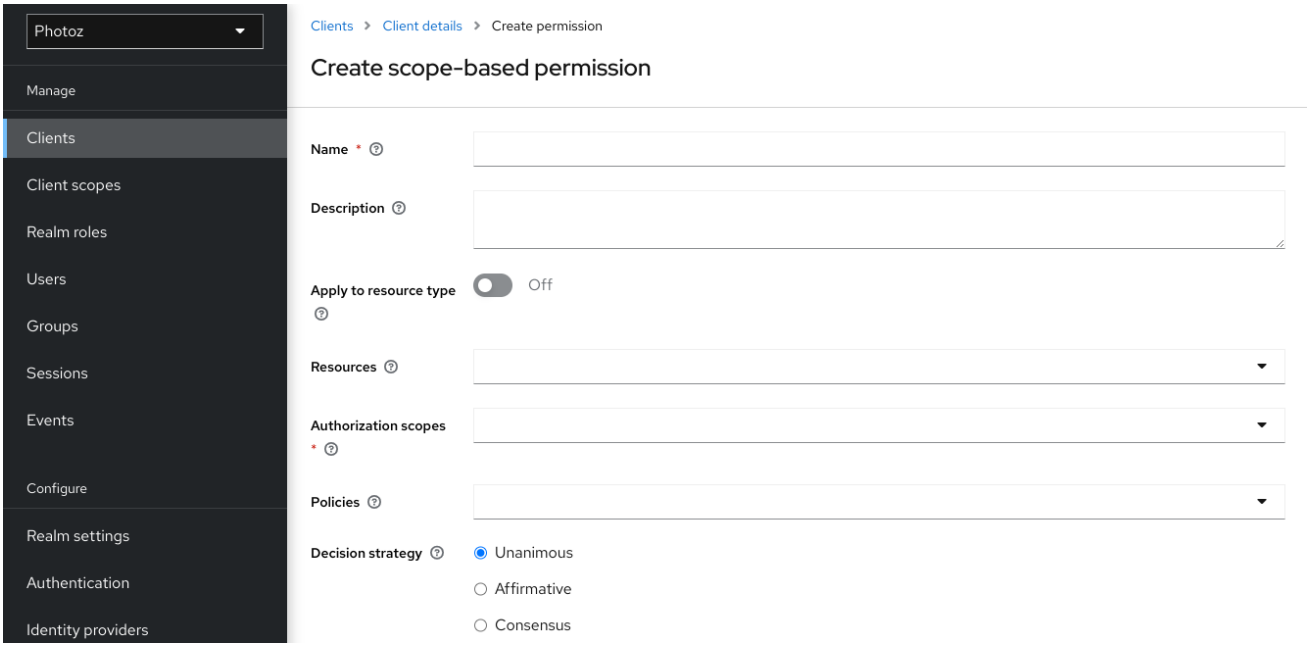


6.2. 创建基于范围的权限

基于范围的权限定义了一组一个或多个范围，以使用一组一个或多个授权策略来保护。与基于资源的权限不同，您可以使用此权限类型只为资源创建权限，也针对与它关联的范围创建权限，在定义管理资源的权限以及可以对其执行的操作时提供更粒度。

要创建基于范围的新权限，请从 **Create permissions** 下拉菜单中选择 **Create scope-based** 权限。

添加范围权限



6.2.1. Configuration

- ***Name***

  描述权限的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- ***描述***

  包含此权限详情的字符串。

- ***资源***

  将范围限制为与所选资源关联的范围。如果选择 none，则所有范围都可用。

- ***范围***

  定义要保护的一个或多个范围的集合。

- ***policy***

定义一组与权限关联的一个或多个策略。要关联策略，您可以通过选择您要创建的策略类型来选择现有策略或创建新策略。

- 决策策略

  此权限的决策策略。

6.3. 策略决策策略

将策略与权限关联时，您还可以定义决策策略，以指定如何评估相关策略的结果来确定访问。

- unanimous

  如果没有提供任何策略，则默认策略。在这种情况下，所有策略都 必须评估为最终决策的正决定。

- Affirmative

  在这种情况下，至少一个 策略必须评估为最终决策的正决定。

- consensus

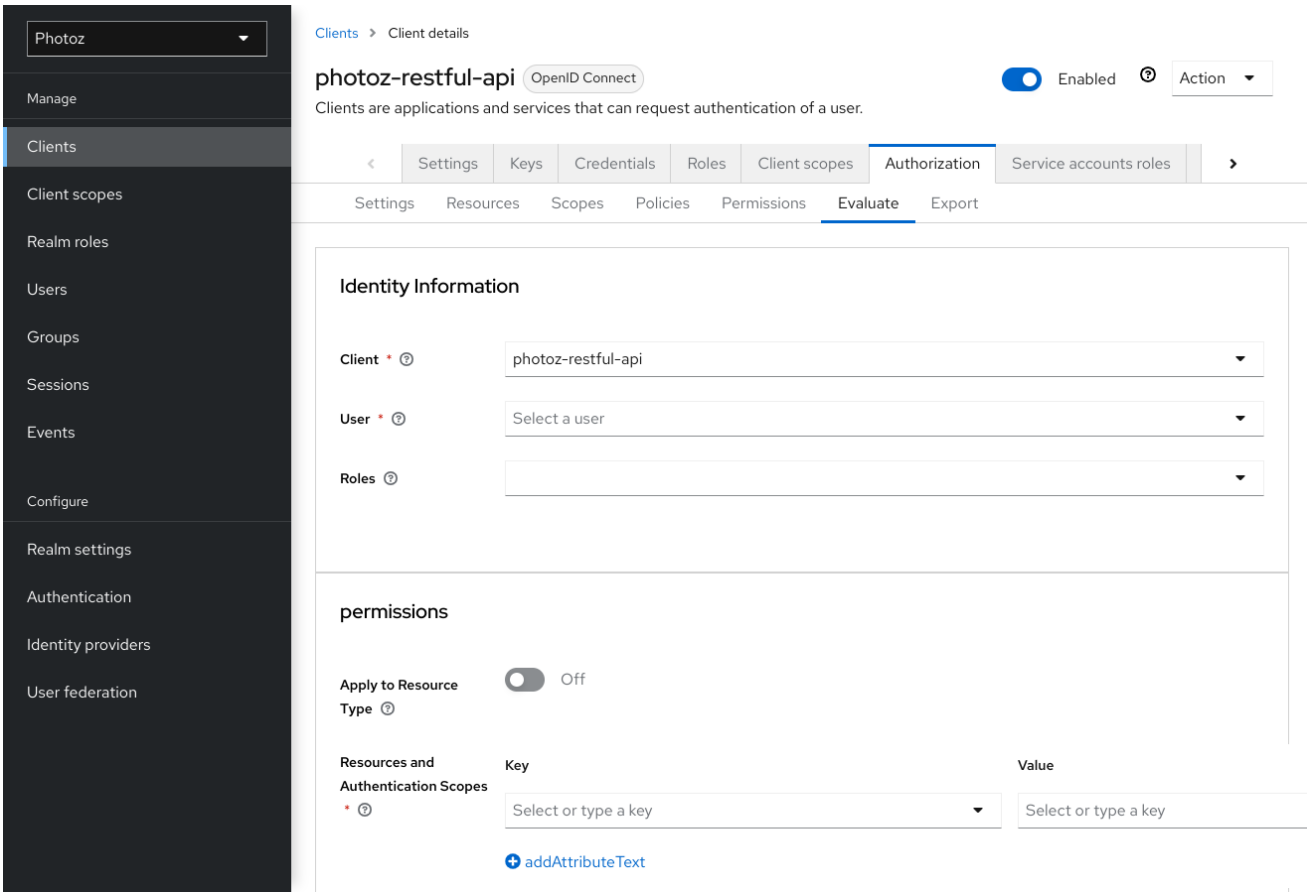
  在这种情况下，正决策的数量必须大于负决策的数量。如果正和负决策的数量相等，则最终决策将是负数。

第 7 章 评估和测试策略

在设计您的策略时，您可以模拟授权请求，以测试如何评估您的策略。

您可以在编辑资源服务器时单击 评估 选项卡来访问策略评估工具。您可以指定不同的输入来模拟真实授权请求并测试您的策略的影响。

策略评估工具



7.1. 提供身份信息

Identity Information 过滤器可用于指定请求权限的用户。

7.2. 提供上下文信息

上下文信息 过滤器可用于定义其他属性到评估上下文，以便策略可以获取这些相同的属性。

7.3. 授予权限

Permissions 过滤器可用于构建授权请求。您可以为一个或多个资源和范围请求权限。如果要根据所有受保护的资源和范围模拟授权请求，请点击 Add，而不指定任何资源或范围。

当您指定所需的值后，单击 Evaluate。

第 8 章 授权服务

红帽构建的 Keycloak 授权服务基于众所周知的标准构建，如 OAuth2 和用户管理的访问规范。

OAuth2 客户端（如前端应用）可以使用令牌端点从服务器获取访问令牌，并使用这些相同的令牌来访问由资源服务器（如后端服务）保护的资源。同样，红帽构建的 Keycloak 授权服务为 OAuth2 提供扩展，以允许根据请求的资源或范围关联的所有策略来处理访问令牌。这意味着，资源服务器可以根据服务器授予的权限并由访问令牌持有来强制实施对其受保护的资源的访问。在红帽构建的 Keycloak 授权服务中，具有权限的访问令牌称为 请求第三方令牌或 RPT。

除了 RPT 的颁发之外，红帽构建的 Keycloak 授权服务还提供了一组 RESTful 端点，允许资源服务器管理其受保护的资源、范围、权限和策略，帮助开发人员将这些功能扩展或集成到其应用程序中，以支持精细的授权。

8.1. 发现授权服务端点和元数据

Red Hat build of Keycloak 提供了一个发现文档，客户端可以从中获取与 Keycloak 授权服务（包括端点位置和功能）交互的所有必要信息。

发现文档可从以下位置获取：

```
curl -X GET \
  http://${host}:${port}/realms/${realm}/.well-known/uma2-configuration
```

其中 ${host}:${port} 是运行红帽构建的 Keycloak 的主机名（或 IP 地址）和端口，${realm} 是红帽构建的 Keycloak 中域名。

因此，您应该获得如下响应：

```
{

    // some claims are expected here

    // these are the main claims in the discovery document about Authorization Services
endpoints location
    "token_endpoint": "http://${host}:${port}/realms/${realm}/protocol/openid-connect/token",
    "token_introspection_endpoint": "http://${host}:${port}/realms/${realm}/protocol/openid-
connect/token/introspect",
    "resource_registration_endpoint":
"http://${host}:${port}/realms/${realm}/authz/protection/resource_set",
```

```
  "permission_endpoint":
"http://${host}:${port}/realms/${realm}/authz/protection/permission",
  "policy_endpoint": "http://${host}:${port}/realms/${realm}/authz/protection/uma-policy"
}
```

每个端点都公开一组特定的功能：

- token_endpoint

  支持 urn:ietf:params:oauth:grant-type:uma-ticket 授权类型的 OAuth2 兼容令牌端点。通过此端点客户端可以发送授权请求并获取 RPT，以及由红帽构建 Keycloak 授予的所有权限的 RPT。

- token_introspection_endpoint

  与 OAuth2 兼容令牌内省端点，客户端可以使用查询服务器来确定 RPT 的活动状态，并确定与令牌关联的任何其他信息，如红帽构建 Keycloak 授予的权限。

- resource_registration_endpoint

  UMA 兼容资源注册端点，哪些资源服务器可用于管理其受保护的资源和范围。此端点提供在红帽构建的 Keycloak 中创建、读取、更新和删除资源和范围的操作。

- permission_endpoint

  UMA 兼容权限端点，哪些资源服务器可用于管理权限票据。此端点提供在红帽构建的 Keycloak 中创建、读取、更新和删除权限票据的操作。

8.2. 获取权限

要从红帽构建的 Keycloak 获取权限，您需要向令牌端点发送授权请求。因此，红帽构建的 Keycloak 将评估与请求的资源和范围关联的所有策略，并使用服务器授予的所有权限发布 RPT。

客户端允许使用以下参数向令牌端点发送授权请求：

- *grant_type*

  这个参数是必需的。必须是 urn:ietf:params:oauth:grant-type:uma-ticket。

- *ticket*

  这个参数是可选的。客户端收到的最新权限票据作为 UMA 授权过程的一部分。

- *claim_token*

  这个参数是可选的。代表服务器在评估要请求的资源和范围时应考虑的额外声明的字符串。此参数允许客户端将声明推送到红帽构建的 Keycloak。有关所有支持的令牌格式的详情，请参阅 claim_token_format 参数。

- *claim_token_format*

  这个参数是可选的。代表 claim_token 参数中指定的令牌格式的字符串。红帽构建的 Keycloak 支持两种令牌格式： urn:ietf:params:oauth:token-type:jwt 和 https://openid.net/specs/openid-connect-core-1_0.html#IDToken。urn:ietf:params:oauth:token-type:jwt 格式表示 claim_token 参数引用访问令牌。https://openid.net/specs/openid-connect-core-1_0.html#IDToken 表示 claim_token 参数引用 OpenID Connect ID Token。

- *rpt*

  这个参数是可选的。之前发布的 RPT 还应在新配置中评估和添加权限。此参数允许拥有 RPT 的客户端在按需添加权限时执行增量授权。

- *权限*

  这个参数是可选的。代表一组一个或多个资源的字符串，以及客户端正在寻求访问的范围。这个参数可以被多次定义，以便请求多个资源和范围的权限。这个参数是 urn:ietf:params:oauth:grant-type:uma-ticket 授权类型的扩展，以便允许客户端在没有权限票据的情况下发送授权请求。字符串的格式必须是： RESOURCE_ID#SCOPE_ID。例如： Resource AcmScope A,Resource A;Scope A, Scope B, Scope C,Resource A,#Scope A.

- ***permission_resource_format***

  这个参数是可选的。代表 权限 参数中资源的格式的字符串。可能的值有 id 和 uri。id 表示格式为 RESOURCE_ID。URI 表示格式为 URI。如果未指定，则默认为 id。

- ***permission_resource_matching_uri***

  这个参数是可选的。指明在以 URI 格式代表 权限 参数的资源时是否要使用路径匹配的布尔值。如果没有指定，则默认为 false。

- ***受众***

  这个参数是可选的。客户端希望访问的资源服务器的客户端标识符。如果定义了 permission 参数，则此参数是必需的。它充当红帽构建的 Keycloak 的提示，以指示应评估哪些权限的上下文。

- ***response_include_resource_name***

  这个参数是可选的。指示服务器是否应当包含在 RPT 权限中的资源名称的布尔值。如果为 false，则仅包含资源标识符。

- ***response_permissions_limit***

  这个参数是可选的。为 RPT 可以具有的权限数量定义限制的整数 N。与 rpt 参数一起使用时，只有最后 N 个请求的权限保存在 RPT 中。

- ***submit_request***

  这个参数是可选的。指示服务器是否应该创建权限请求到权限票据引用的资源和范围的布尔值。这个参数只有在与 ticket 参数一起使用时作为 UMA 授权过程的一部分时才有效。

- ***response_mode***

  这个参数是可选的。一个字符串值，代表服务器应如何响应授权请求。当您主要关注总体决策或服务器授予的权限，而不是标准的 OAuth2 响应时，此参数特别有用。可能的值有：

- 决定

  表示来自服务器的响应应该只通过返回具有以下格式的 JSON 来代表总体决定：

  ```
  {
    'result': true
  }
  ```

  如果授权请求没有映射到任何权限，则返回 403 HTTP 状态代码。

- 权限

  通过返回具有以下格式的 JSON 来指示来自服务器的响应应包含服务器授予的任何权限：

  ```
  [
    {
      'rsid': 'My Resource'
      'scopes': ['view', 'update']
    },

    ...
  ]
  ```

  如果授权请求没有映射到任何权限，则返回 403 HTTP 状态代码。

当客户端希望访问由资源服务器保护的两个资源时，授权请求示例。

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "audience={resource_server_client_id}" \
  --data "permission=Resource A#Scope A" \
  --data "permission=Resource B#Scope B"
```

当客户端希望访问由资源服务器保护的任何资源和范围时，授权请求示例。注意：这不会评估所有资源的权限。相反，评估由资源服务器拥有的资源的权限（由请求用户拥有的），并明确授予其他所有者的请求用户的权限。

```
curl -X POST \
```

```
http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
-H "Authorization: Bearer ${access_token}" \
--data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
--data "audience={resource_server_client_id}"
```

当客户端在从资源服务器获得权限票据作为授权过程的一部分后，寻求访问 UMA 保护的资源时，授权请求示例：

```
curl -X POST \
http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
-H "Authorization: Bearer ${access_token}" \
--data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
--data "ticket=${permission_ticket}
```

如果红帽构建的 Keycloak 评估过程会导致权限无效，它会发出与权限关联的 RPT：

红帽构建的 Keycloak 使用 RPT 响应客户端

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
   "access_token": "${rpt}",
}
```

在使用某些其他授权类型时，服务器的响应与来自令牌端点的任何其他响应类似。RPT 可以从 access_token 响应参数获取。如果客户端未获得授权，红帽构建的 Keycloak 以 403 HTTP 状态代码进行响应：

红帽构建的 Keycloak 拒绝授权请求

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
...
{
   "error": "access_denied",
   "error_description": "request_denied"
}
```

8.2.1. 客户端验证方法

客户端需要向令牌端点进行身份验证以获取 RPT。当使用 urn:ietf:params:oauth:grant-type:uma-ticket 授权类型时，客户端可以使用以下任一身份验证方法：

- bearer 令牌

  客户端应将访问令牌作为 Bearer 凭据在 HTTP Authorization 标头中发送到令牌端点。

  示例：使用访问令牌向令牌端点进行身份验证的授权请求

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket"
```

  当客户端代表用户操作时，此方法特别有用。在这种情况下，bearer 令牌是由红帽构建的 Keycloak 签发的访问令牌，代表用户（或代表自身）成为某些客户端。将评估权限，考虑访问令牌代表的访问上下文。例如，如果向客户端 A 发出了访问令牌，则根据用户 A 有权访问的资源和范围，将授予权限。

- 客户端凭证

  客户端可以使用红帽构建的 Keycloak 支持的任何客户端验证方法。例如，client_id/client_secret 或 JWT。

  示例：使用客户端 id 和客户端 secret 向令牌端点进行身份验证的授权请求

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Basic cGhvdGg6L7Jl13RmfWgtkk==pOnNlY3JldA==" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket"
```

8.2.2. 推送声明

从服务器获取权限时，您可以推送任意声明，以便在评估权限时将这些声明提供给您的策略。

如果您要在没有使用权限票据(UMA 流) 的情况下 从服务器获取权限，您可以将授权请求发送到令牌端点，如下所示：

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "claim_token=ewoglCAib3JnYW5pemF0aW9uIjogWyJhY21lIl0KfQ==" \
  --data "claim_token_format=urn:ietf:params:oauth:token-type:jwt" \
  --data "client_id={resource_server_client_id}" \
  --data "client_secret={resource_server_client_secret}" \
  --data "audience={resource_server_client_id}"
```

claim_token 参数需要一个 BASE64 编码的 JSON，格式类似以下示例：

```
{
   "organization" : ["acme"]
}
```

格式需要一个或多个声明，每个声明的值必须是字符串数组。

8.2.2.1. 使用 UMA 推送声明

有关如何使用 UMA 和权限票据时如何推送声明的更多详细信息，请参阅 权限 API

8.3. 用户管理的访问

红帽构建的 Keycloak 授权服务基于用户管理的访问或 UMA。UMA 是一个以以下方式增强 OAuth2 功能的规格：

- 隐私性

  现在，用户隐私会变得很大的问题，因为更多的数据和设备可用并连接到云。通过 UMA 和红

帽构建的 Keycloak，资源服务器可以增强其功能，以改进其资源如何保护用户隐私，其中根据用户定义的策略授予权限。

- 第三方授权

  资源所有者（例如，常规最终用户）可以管理对资源的访问，并授权其他方（如常规最终用户）来访问这些资源。这与 OAuth2 不同，其同意代表一个用户，UMA 资源所有者能够以完全异步的方式同意对其他用户的访问。

- 资源共享

  资源所有者允许管理其资源的权限，并决定谁可以访问特定资源以及如何。然后，红帽构建的 Keycloak 可以充当共享管理服务，以供资源所有者管理其资源。

红帽构建的 Keycloak 是一个与 UMA 2.0 兼容的授权服务器，可提供大多数 UMA 功能。

例如，假设用户 alice （资源所有者）使用互联网银行服务（资源服务器）来管理她的银行帐户（资源）。第一天，Alice 决定打开她的银行客户，到 Bob （请求方），是财务专家。但是，Bob 应该只具有查看（范围）AXX 帐户的权限。

作为资源服务器，互联网银行服务必须能够保护 alice 银行帐户。为此，它依赖于红帽构建的 Keycloak 资源注册端点，来在代表 alice 的 bank 帐户的服务器中创建一个资源。

目前，如果 Bob 试图访问 alice 的 bank 帐户，则拒绝访问。互联网银行服务为银行帐户定义了几个默认策略。其中之一就是，只有所有者（本例中为 alice）可以访问她的银行帐户。

但是，与 alice 的隐私相关的互联网银行服务也使她能够更改银行帐户的特定策略。她可以更改这些政策之一就是定义允许哪些人查看她的银行帐户。为此，互联网银行服务依赖于红帽构建的 Keycloak 来为 alice 提供空间，其中她可以选择允许其访问的个人和操作（或数据）。在任何时候，Alice 可以撤销访问权限，或向 bob 授予其他权限。

8.3.1. 授权过程

在 UMA 中，当客户端尝试访问 UMA 保护的资源服务器时，授权过程会启动。

UMA 保护的资源服务器需要请求中的 bearer 令牌，其中令牌是 RPT。当客户端请求资源没有 RPT

时：

客户端请求受保护的资源而无需发送 RPT

```
curl -X GET \
  http://${host}:${port}/my-resource-server/resource/1bfdfe78-a4e1-4c2d-b142-fc92b75b986f
```

资源服务器将响应发送回具有权限 票据 的客户端，以及一个 as_uri 参数，并将红帽构建的 Keycloak 服务器的位置发送到其中，以获取 RPT。

资源服务器使用权限票据响应

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="${realm}",
   as_uri="https://${host}:${port}/realms/${realm}",
   ticket="016f84e8-f9b9-11e0-bd6f-0021cc6004de"
```

权限票据是红帽构建的 Keycloak 权限 API 发布的特殊令牌类型。它们代表正在请求的权限（如资源和范围），以及与请求关联的任何其他信息。只有资源服务器才能创建这些令牌。

现在，客户端有权限票据以及红帽构建的 Keycloak 服务器的位置，客户端可以使用发现文档来获取令牌端点的位置并发送授权请求。

客户端向令牌端点发送授权请求，以获取 RPT

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "ticket=${permission_ticket}
```

如果红帽构建的 Keycloak 评估过程会导致权限无效，它会发出与权限关联的 RPT：

红帽构建的 Keycloak 使用 RPT 响应客户端

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
   "access_token": "${rpt}",
}
```

在使用某些其他授权类型时，服务器的响应与来自令牌端点的任何其他响应类似。RPT 可以从 access_token 响应参数获取。如果客户端没有授权红帽构建 Keycloak 的权限以 403 HTTP 状态代码进行响应：

红帽构建的 Keycloak 拒绝授权请求

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
...
{
   "error": "access_denied",
   "error_description": "request_denied"
}
```

8.3.2. 提交权限请求

作为授权过程的一部分，客户端首先需要从 UMA 保护的资源服务器获取权限票据，以便在红帽构建的 Keycloak Token Endpoint 中与 RPT 进行交换。

默认情况下，红帽构建的 Keycloak 会以 403 HTTP 状态代码进行响应，并在无法通过 RPT 发布客户

端时发出 request_denied 错误。

红帽构建的 Keycloak 拒绝授权请求

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
...
{
   "error": "access_denied",
   "error_description": "request_denied"
}
```

这样的响应意味着红帽构建的 Keycloak 无法发布具有权限票据代表的 RPT。

在某些情况下，客户端应用程序可能希望启动异步授权流，并让请求的资源所有者决定是否应该授予访问权限。为此，客户端可以使用 commit _request 请求参数以及向令牌端点的授权请求：

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "ticket=${permission_ticket} \
  --data "submit_request=true"
```

使用 commit_request 参数时，红帽构建的 Keycloak 会为访问被拒绝的每个资源保留一个权限请求。创建后，资源所有者可以检查其帐户并管理其权限请求。

您可以将此功能视为应用程序中的 Request Access 按钮，其中用户可以要求其他用户访问其资源。

8.3.3. 管理对用户资源的访问

用户可以使用红帽构建的 Keycloak 帐户控制台来管理对其资源的访问。要启用此功能，您必须首先为您的域启用用户管理的访问。

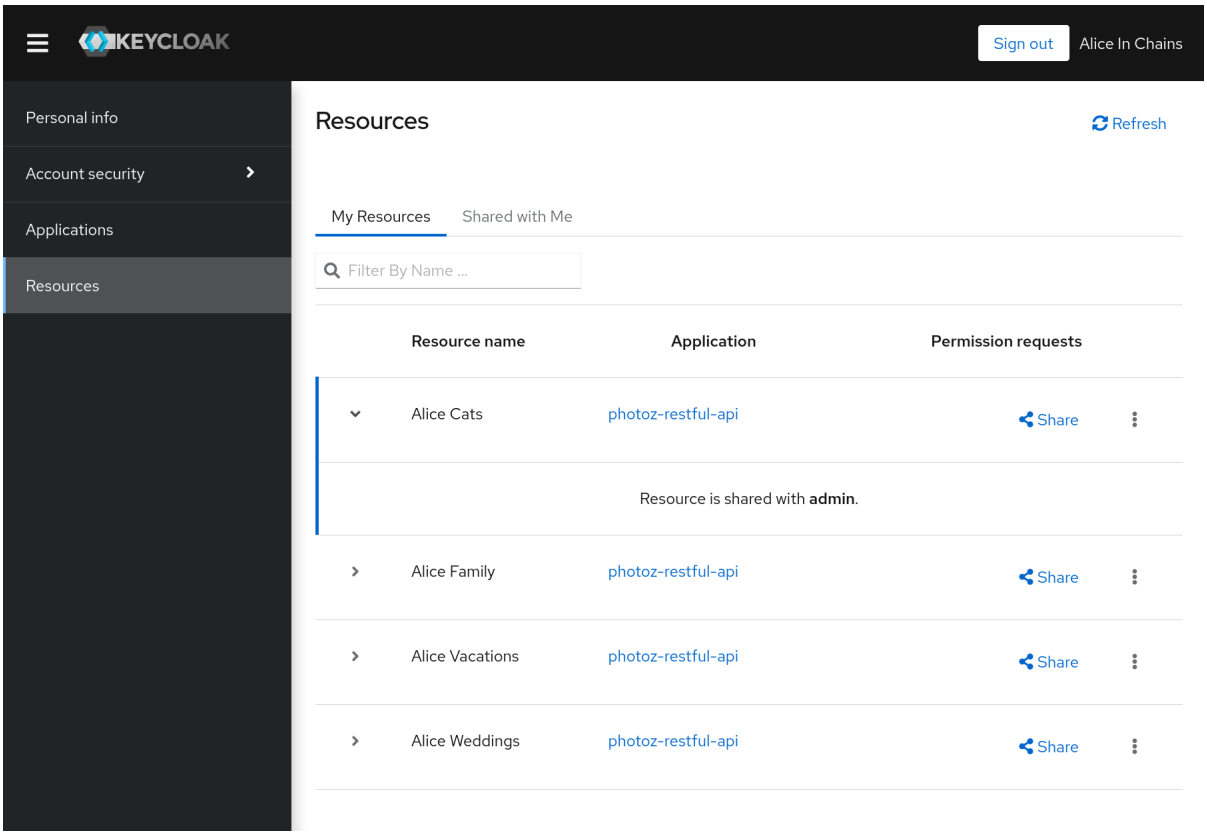
流程

1. 登录 Admin 控制台。

2. 单击菜单中的 Realm Settings。

3. 将 用户管理的访问切换为 ON。

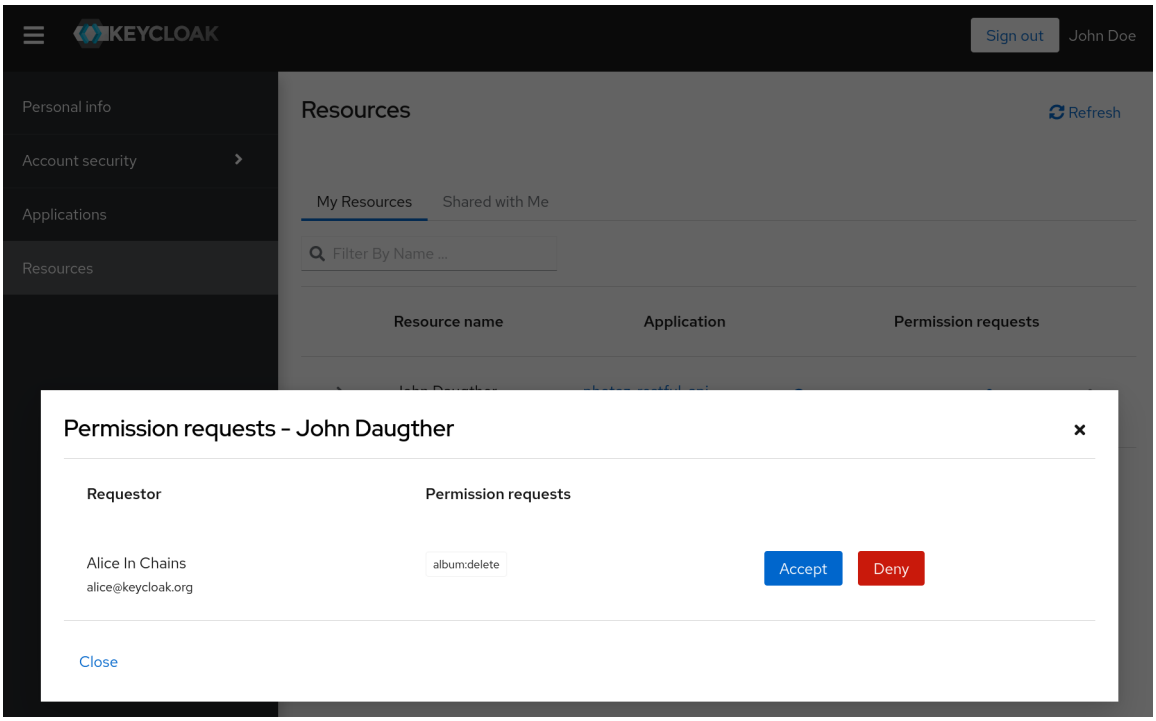
4. 单击管理控制台右上角的用户名，再选择 Manage Account。



5. 在菜单选项中，单击 My Resources。此时会显示一个带有以下选项的页面。

- 管理 我的资源

本节包含由用户拥有的所有资源的列表。用户可以单击资源以获取更多详细信息，并与他人共享资源。当某个权限请求等待批准时，将在资源名称旁边放置一个图标。这些请求连接到请求访问特定资源的方（用户）。用户可以批准或拒绝这些请求。您可以通过点图标来完成此操作



- *管理与我共享的资源*

  本节包含与用户共享的所有资源的列表。

- *通过访问此资源来管理人员*

  本节包含有权访问此资源的人员的列表。允许用户通过单击 Revoke 按钮或删除特定的权限 来撤销访问权限。

- *与其它资源共享资源*

  通过键入其他用户的用户名或电子邮件，用户可以共享该资源并选择希望授予访问权限的权限。

8.4. 保护 API

Protection API 提供了一组符合 UMA 的端点，提供：

- *资源管理*

  使用这个端点，资源服务器可以远程管理其资源，并启用 策略强制 者查询服务器以获取需要保护的资源。

- *权限管理*

  在 UMA 协议中，资源服务器访问此端点以创建权限票据。红帽构建的 Keycloak 还提供端点来管理权限和查询权限的状态。

- *Policy API*

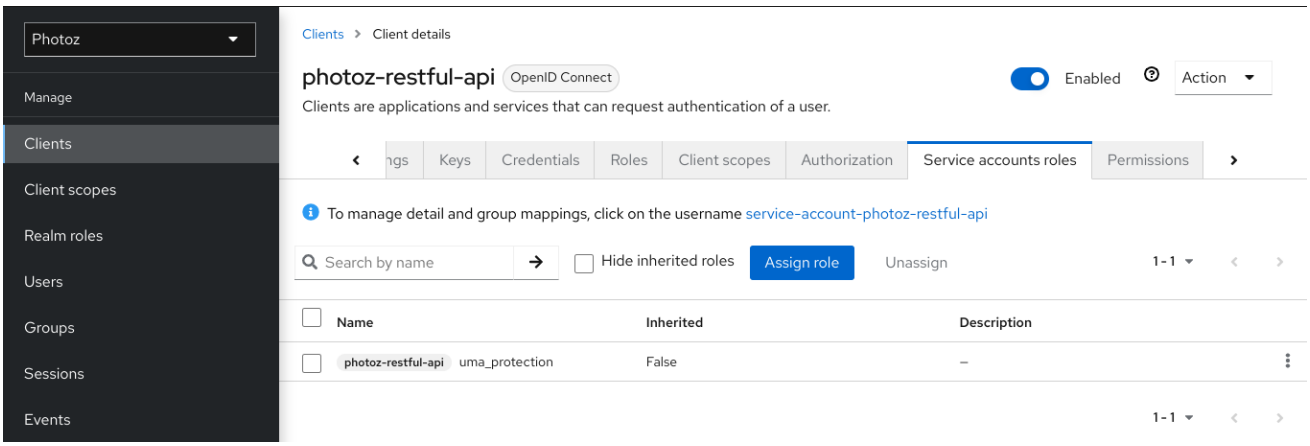
  红帽构建的 Keycloak 利用 UMA 保护 API，以允许资源服务器为其用户管理权限。除了 Resource 和 Permission API 外，红帽构建的 Keycloak 提供了一个 Policy API，其中的权限可由其用户的资源服务器设置为资源。

此 API 的一个重要要求是，只允许 资源服务器使用特殊的 OAuth2 访问令牌访问其端点，称为保护 API 令牌(PAT)。在 UMA 中，PAT 是具有范围 uma_protection 的令牌。

8.4.1. 什么是 PAT 以及如何获取它

保护 API 令牌 (PAT)是特殊的 OAuth2 访问令牌，范围定义为 uma_protection。当您创建资源服务器时，红帽构建的 Keycloak 会自动为对应的客户端应用程序创建一个角色 uma_protection，并将它与客户端的服务帐户关联。

使用 uma_protection 角色授予的服务帐户



与任何其他 OAuth2 访问令牌一样，资源服务器可以从红帽构建的 Keycloak 获取 PAT。例如，使用 curl：

```
curl -X POST \
   -H "Content-Type: application/x-www-form-urlencoded" \
   -d 'grant_type=client_credentials&client_id=${client_id}&client_secret=${client_secret}' \
   "http://${host}:${port}/realms/${realm_name}/protocol/openid-connect/token"
```

上面的示例使用 client_credentials 授权类型从服务器获取 PAT。因此，服务器会返回类似如下的响应：

```
{
 "access_token": ${PAT},
 "expires_in": 300,
 "refresh_expires_in": 1800,
 "refresh_token": ${refresh_token},
 "token_type": "bearer",
 "id_token": ${id_token},
 "not-before-policy": 0,
 "session_state": "ccea4a55-9aec-4024-b11c-44f6f168439e"
}
```

注意

红帽构建的 Keycloak 可以通过不同的方式验证您的客户端应用程序。为了简单起见，此处使用 client_credentials 授权类型，它需要 client_id 和 client_secret。您可以选择使用任何支持的身份验证方法。

8.4.2. 管理资源

资源服务器可以使用 UMA 兼容端点远程管理其资源。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set
```

此端点提供如下操作（为清晰起见省略的路径）：

- 创建资源设置描述： POST /resource_set
- read resource set description: GET /resource_set/{_id}

- 更新资源集描述： PUT /resource_set/{_id}
- 删除资源集描述： DELETE /resource_set/{_id}
- 列出资源集描述：GET /resource_set

有关这些操作的合同的更多信息，请参阅 UMA 资源注册 API。

8.4.2.1. 创建资源

要创建资源，您必须发送 HTTP POST 请求，如下所示：

```
curl -v -X POST \
 http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set \
 -H 'Authorization: Bearer '$pat \
 -H 'Content-Type: application/json' \
 -d '{
   "name":"Tweedl Social Service",
   "type":"http://www.example.com/rsrcs/socialstream/140-compatible",
   "icon_uri":"http://www.example.com/icons/sharesocial.png",
   "resource_scopes":[
      "read-public",
      "post-updates",
      "read-private",
      "http://www.example.com/scopes/all"
   ]
 }'
```

默认情况下，资源的所有者是资源服务器。如果要定义不同的所有者，如特定用户，您可以发送如下请求：

```
curl -v -X POST \
 http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set \
 -H 'Authorization: Bearer '$pat \
 -H 'Content-Type: application/json' \
 -d '{
   "name":"Alice Resource",
   "owner": "alice"
 }'
```

可以使用用户名或用户标识符设置属性 所有者。

8.4.2.2. 创建用户管理的资源

默认情况下，通过保护 API 创建的资源不能由资源所有者通过 帐户控制台 进行管理。

要创建资源并允许资源所有者管理这些资源，您必须设置 ownerManagedAccess 属性，如下所示：

```
curl -v -X POST \
 http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set \
 -H 'Authorization: Bearer '$pat \
 -H 'Content-Type: application/json' \
 -d '{
    "name":"Alice Resource",
    "owner": "alice",
    "ownerManagedAccess": true
 }'
```

8.4.2.3. 更新资源

要更新现有资源，请按如下所示发送 HTTP PUT 请求：

```
curl -v -X PUT \
 http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set/{resource_id} \
 -H 'Authorization: Bearer '$pat \
 -H 'Content-Type: application/json' \
 -d '{
    "_id": "Alice Resource",
    "name":"Alice Resource",
    "resource_scopes": [
       "read"
    ]
 }'
```

8.4.2.4. 删除资源

要删除现有资源，请按如下所示发送 HTTP DELETE 请求：

```
curl -v -X DELETE \
 http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set/{resource_id} \
 -H 'Authorization: Bearer '$pat
```

8.4.2.5. 查询资源

要按 id 查询资源，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set/{resource_id}
```

要查询给定名称的资源，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?name=Alice Resource
```

默认情况下，名称 过滤器将与给定模式的任何资源匹配。要将查询限制为仅返回具有完全匹配的资源，请使用：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?name=Alice Resource&exactName=true
```

要查询资源给出的 uri，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?uri=/api/alice
```

要查询给定 所有者 的资源，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?owner=alice
```

要查询给定类型的资源，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?type=albums
```

要查询给定范围的资源，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?scope=read
```

当查询服务器以获取权限时，首先使用 参数，最大 结果来限制结果。

8.4.3. 管理权限请求

使用 UMA 协议的资源服务器可以使用特定的端点来管理权限请求。此端点提供了一个与 UMA 兼容的流，用于注册权限请求并获取一个权限票据。

> http://${host}:${port}/realms/${realm_name}/authz/protection/permission

权限票据 是一个特殊的安全令牌类型，代表一个权限请求。根据 UMA 规格，权限票据是：

一个关联句柄，从授权服务器传到资源服务器、从资源服务器到客户端，最终从客户端返回到授权服务器，以使授权服务器能够评估正确的策略以应用到授权数据的请求。

在大多数情况下，您不需要直接处理此端点。红帽构建的 Keycloak 提供了一个 策略 保护程序，可为您的资源服务器启用 UMA，以便它可以从授权服务器获取权限票据，将此票据返回到客户端应用程序，并根据最终请求方令牌(RPT)强制实施授权决策。

从红帽构建的 Keycloak 获取权限票据的过程由资源服务器执行，而不是常规客户端应用程序，当客户端试图访问受保护的资源时获取权限票据，而无需必要授予访问资源。在使用 UMA 时，权限票据的颁发是一个重要方面，因为它允许资源服务器：

- 客户端与资源服务器保护的资源关联的数据抽象
- 在红帽构建的 Keycloak 授权请求中注册，之后可在工作流中使用这些请求，根据资源的所有者同意授予访问权限
- 将资源服务器与授权服务器分离，并允许它们使用不同的授权服务器保护和管理其资源

客户端明智，权限票据也具有值得注意的方面：

- 客户端不需要了解授权数据如何与受保护的资源关联。权限票据对客户端完全不透明。
- 客户端可以访问不同资源服务器上的资源，并由不同的授权服务器进行保护

这些仅仅是 UMA 其他方面的优势，其中 UMA 的其他方面都基于权限票据，特别考虑隐私和用户对资源控制的访问。

8.4.3.1. 创建权限票据

要创建权限票据，请按如下所示发送 HTTP POST 请求：

```
curl -X POST \
  http://${host}:${port}/realms/${realm_name}/authz/protection/permission \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '[
  {
    "resource_id": "{resource_id}",
    "resource_scopes": [
      "view"
    ]
  }
]'
```

在创建问题单时，您还可以推送任意声明，并将这些声明与票据相关联：

```
curl -X POST \
  http://${host}:${port}/realms/${realm_name}/authz/protection/permission \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '[
  {
    "resource_id": "{resource_id}",
    "resource_scopes": [
      "view"
    ],
    "claims": {
        "organization": ["acme"]
    }
  }
]'
```

在评估与权限票据关联的资源和范围时，您的策略可以使用这些声明。

8.4.3.2. 其他非 UMA 兼容端点

8.4.3.2.1. 创建权限票据

要将带有 id {resource_id} 的特定资源的权限授予 ID 为 {user_id} 的用户，作为资源的所有者发送 HTTP POST 请求，如下所示：

```
curl -X POST \
    http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket \
    -H 'Authorization: Bearer '$access_token \
    -H 'Content-Type: application/json' \
```

```
  -d '{
    "resource": "{resource_id}",
    "requester": "{user_id}",
    "granted": true,
    "scopeName": "view"
  }'
```

8.4.3.2.2. 获取权限票据

```
curl http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket \
  -H 'Authorization: Bearer '$access_token
```

您可以使用这些查询参数中的任何一个：

- scopeId
- resourceId
- owner
- requester
- 已授予
- returnNames
- 第一个
- max

8.4.3.2.3. 更新权限票据

```
curl -X PUT \
  http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket \
  -H 'Authorization: Bearer '$access_token \
  -H 'Content-Type: application/json' \
  -d '{
```

```
    "id": "{ticket_id}"
    "resource": "{resource_id}",
    "requester": "{user_id}",
    "granted": false,
    "scopeName": "view"
  }'
```

8.4.3.2.4. 删除权限票据

```
curl -X DELETE
http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket/{ticket_id} \
    -H 'Authorization: Bearer '$access_token
```

8.4.4. 使用 Policy API 管理资源权限

红帽构建的 Keycloak 利用 UMA 保护 API，以允许资源服务器为其用户管理权限。除了 Resource 和 Permission API 外，红帽构建的 Keycloak 提供了一个 Policy API，其中的权限可由其用户的资源服务器设置为资源。

Policy API 位于：

```
http://${host}:${port}/realms/${realm_name}/authz/protection/uma-policy/{resource_id}
```

此 API 通过 bearer 令牌进行保护，该令牌必须代表用户授予的同意，以便代表其管理权限。bearer 令牌可以是从令牌端点获取的常规访问令牌：

- 资源所有者密码凭证授予类型
- 令牌交换，为 audience 是资源服务器的令牌交换授予某些客户端（公共客户端）的访问令牌

8.4.4.1. 将权限与资源关联

要将权限与特定资源关联，您必须发送 HTTP POST 请求，如下所示：

```
curl -X POST \
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
  -H 'Authorization: Bearer '$access_token \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '{
 "name": "Any people manager",
```

```
 "description": "Allow access to any people manager",
 "scopes": ["read"],
 "roles": ["people-manager"]
}'
```

在上例中，我们创建了一个新权限，并将其与 resource_id 表示的资源关联，其中任何具有角色 people-manager 的用户都应被授予 读 范围。

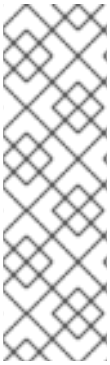
您还可以使用其他访问控制机制创建策略，比如使用组：

```
curl -X POST \
 http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
 -H 'Authorization: Bearer '$access_token \
 -H 'Cache-Control: no-cache' \
 -H 'Content-Type: application/json' \
 -d '{
 "name": "Any people manager",
 "description": "Allow access to any people manager",
 "scopes": ["read"],
 "groups": ["/Managers/People Managers"]
}'
```

或者一个特定的客户端：

```
curl -X POST \
 http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
 -H 'Authorization: Bearer '$access_token \
 -H 'Cache-Control: no-cache' \
 -H 'Content-Type: application/json' \
 -d '{
 "name": "Any people manager",
 "description": "Allow access to any people manager",
 "scopes": ["read"],
 "clients": ["my-client"]
}'
```

甚至使用 JavaScript 使用自定义策略：



注意

上传脚本 已弃用，并将在以后的发行版本中删除。此功能默认为禁用。

使用 -Dkeycloak.profile.feature.upload_scripts=enabled 来启用服务器。如需了解更多详细信息，请参阅 启用和禁用功能 章节。

```
curl -X POST \
 http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
 -H 'Authorization: Bearer '$access_token \
 -H 'Cache-Control: no-cache' \
 -H 'Content-Type: application/json' \
 -d '{
"name": "Any people manager",
"description": "Allow access to any people manager",
"scopes": ["read"],
"condition": "my-deployed-script.js"
}'
```

也可以设置这些访问控制机制的任意组合。

要更新现有权限，请按如下所示发送 HTTP PUT 请求：

```
curl -X PUT \
 http://localhost:8180/realms/photoz/authz/protection/uma-policy/{permission_id} \
 -H 'Authorization: Bearer '$access_token \
 -H 'Content-Type: application/json' \
 -d '{
   "id": "21eb3fed-02d7-4b5a-9102-29f3f09b6de2",
   "name": "Any people manager",
   "description": "Allow access to any people manager",
   "type": "uma",
   "scopes": [
      "album:view"
   ],
   "logic": "POSITIVE",
   "decisionStrategy": "UNANIMOUS",
   "owner": "7e22131a-aa57-4f5f-b1db-6e82babcd322",
   "roles": [
      "user"
   ]
}'
```

8.4.4.2. 删除权限

要删除与资源关联的权限，请按如下所示发送 HTTP DELETE 请求：

```
curl -X DELETE \
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{permission_id} \
  -H 'Authorization: Bearer '$access_token
```

8.4.4.3. 查询权限

要查询与资源关联的权限，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy?resource={resource_id}
```

要查询给定其名称的权限，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy?name=Any people
manager
```

要查询与特定范围关联的权限，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy?scope=read
```

要查询所有权限，请按如下所示发送 HTTP GET 请求：

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy
```

当查询服务器以获取权限时，首先使用 参数，最大 结果来限制结果。

8.5. 请求方令牌

请求方令牌 (RPT) 是使用 JSON web signature (JWS) 进行了数字签名的 JSON web token (JWT)。该令牌基于之前由红帽构建的 Keycloak 签发的 OAuth2 访问令牌构建，用于代表用户或其自有的特定客户端。

当您解码 RPT 时，您会看到类似如下的有效负载：

```
{
  "authorization": {
```

```
    "permissions": [
      {
        "resource_set_id": "d2fe9843-6462-4bfc-baba-b5787bb6e0e7",
        "resource_set_name": "Hello World Resource"
      }
    ]
  },
  "jti": "d6109a09-78fd-4998-bf89-95730dfd0892-1464906679405",
  "exp": 1464906971,
  "nbf": 0,
  "iat": 1464906671,
  "sub": "f1888f4d-5172-4359-be0c-af338505d86c",
  "typ": "kc_ett",
  "azp": "hello-world-authz-service"
}
```

在这个令牌中，您可以从 权限 声明中获取服务器授予的所有权限。

另请注意，权限直接与您要保护的资源/范围相关，并与用于实际授予和发出这些相同权限的访问控制方法完全分离。

8.5.1. 内省请求方令牌

有时，您可能希望内省请求方令牌(RPT)来检查其有效性，或者在令牌内获取权限，以对资源服务器端实施授权决策。

令牌内省可帮助您两个主要的用例：

- 当客户端应用程序需要查询令牌的有效性时，以获取具有相同或额外权限的新令牌
- 在资源服务器端强制授权决策时，特别是没有内置 策略强制器 适合您的应用程序

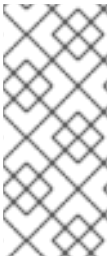
8.5.2. 获取有关 RPT 的信息

令牌内省基本上是一个 OAuth2令牌内省 - 兼容端点，您可以从中获取有关 RPT 的信息。

```
http://${host}:${port}/realms/${realm_name}/protocol/openid-connect/token/introspect
```

要使用此端点内省 RPT，您可以按如下方式向服务器发送请求：

```
curl -X POST \
  -H "Authorization: Basic aGVsbG8td29ybGQtYXV0aHotc2VydmljZTpzZWNyZXQ=" \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d 'token_type_hint=requesting_party_token&token=${RPT}' \
  "http://localhost:8080/realms/hello-world-authz/protocol/openid-connect/token/introspect"
```



注意

以上请求使用 HTTP BASIC 并传递客户端的凭据（客户端 ID 和 secret）来验证尝试内省令牌的客户端，但您可以使用红帽构建的 Keycloak 支持的任何其他客户端验证方法。

内省端点需要两个参数：

- **token_type_hint**

  使用 request_party_token 作为此参数的值，这表示您要内省 RPT。

- **token**

  在授权过程中，使用令牌字符串（由服务器返回）作为此参数的值。

因此，服务器响应是：

```
{
  "permissions": [
    {
      "resource_id": "90ccc6fc-b296-4cd1-881e-089e1ee15957",
      "resource_name": "Hello World Resource"
    }
  ],
  "exp": 1465314139,
  "nbf": 0,
  "iat": 1465313839,
  "aud": "hello-world-authz-service",
  "active": true
}
```

如果 RPT 未激活，则返回此响应：

```
{
  "active": false
}
```

8.5.3. 每次我想要内省 RPT 时，我是否需要调用服务器？

否。就像红帽构建的 Keycloak 服务器发布的常规访问令牌一样，RPT 也使用 JSON Web 令牌(JWT)规格作为默认格式。

如果要在不调用远程内省端点的情况下验证这些令牌，您可以解码 RPT，并在本地查询其有效性。解码令牌后，您也可以使用令牌中的权限来强制实施授权决策。

这基本上是策略执行者的作用。请确定：

- 验证 RPT 的签名（基于域的公钥）
- 基于它的 exp, iat, 和 aud 声明来查询令牌的有效性。

其他资源

- JSON Web 令牌(JWT)
- 策略实施器

8.6. 授权客户端 JAVA API

根据您的要求，资源服务器应能够远程管理资源，甚至以编程方式检查权限。如果使用 Java，您可以使用 Authorization Client API 访问红帽构建的 Keycloak 授权服务。

对于需要访问服务器提供的不同端点的资源服务器的目标，如 Token Endpoint、Resource 和 Permission 管理端点。

8.6.1. Maven 依赖项

```
<dependencies>
  <dependency>
```

```
        <groupId>org.keycloak</groupId>
        <artifactId>keycloak-authz-client</artifactId>
        <version>${KEYCLOAK_VERSION}</version>
    </dependency>
</dependencies>
```

8.6.2. Configuration

客户端配置在 keycloak.json 文件中定义，如下所示：

```
{
  "realm": "hello-world-authz",
  "auth-server-url" : "http://localhost:8080",
  "resource" : "hello-world-authz-service",
  "credentials": {
    "secret": "secret"
  }
}
```

- realm （必需）

  域的名称。

- auth-server-url （必需）

  红帽构建的 Keycloak 服务器的基本 URL。所有其他红帽构建的 Keycloak 页面和 REST 服务端点都源自此内容。它通常采用 https://host:port 格式。

- resource （必需）

  应用程序的客户端 ID。每个应用都有一个客户端 ID，用于识别应用。

- credentials （必需）

  指定应用程序的凭证。这是一个对象表示法，其中键是凭证类型，值是凭证类型的值。

配置文件通常位于应用程序的 classpath 中，来自客户端要尝试查找 keycloak.json 文件的默认位置。

8.6.3. 创建授权客户端

考虑在 classpath 中有一个 keycloak.json 文件，您可以创建一个新的 AuthzClient 实例，如下所示：

```
// create a new instance based on the configuration defined in a keycloak.json located in your classpath
AuthzClient authzClient = AuthzClient.create();
```

8.6.4. 获取用户权利

以下是如何获取用户权利的示例：

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// create an authorization request
AuthorizationRequest request = new AuthorizationRequest();

// send the entitlement request to the server in order to
// obtain an RPT with all permissions granted to the user
AuthorizationResponse response = authzClient.authorization("alice", "alice").authorize(request);
String rpt = response.getToken();

System.out.println("You got an RPT: " + rpt);

// now you can use the RPT to access protected resources on the resource server
```

以下是如何为一个或多个资源获取用户权利的示例：

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// create an authorization request
AuthorizationRequest request = new AuthorizationRequest();

// add permissions to the request based on the resources and scopes you want to check access
request.addPermission("Default Resource");

// send the entitlement request to the server in order to
// obtain an RPT with permissions for a single resource
AuthorizationResponse response = authzClient.authorization("alice", "alice").authorize(request);
String rpt = response.getToken();
```

```
System.out.println("You got an RPT: " + rpt);

// now you can use the RPT to access protected resources on the resource server
```

8.6.5. 使用保护 API 创建资源

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// create a new resource representation with the information we want
ResourceRepresentation newResource = new ResourceRepresentation();

newResource.setName("New Resource");
newResource.setType("urn:hello-world-authz:resources:example");

newResource.addScope(new ScopeRepresentation("urn:hello-world-authz:scopes:view"));

ProtectedResource resourceClient = authzClient.protection().resource();
ResourceRepresentation existingResource =
resourceClient.findByName(newResource.getName());

if (existingResource != null) {
    resourceClient.delete(existingResource.getId());
}

// create the resource on the server
ResourceRepresentation response = resourceClient.create(newResource);
String resourceId = response.getId();

// query the resource using its newly generated id
ResourceRepresentation resource = resourceClient.findById(resourceId);

System.out.println(resource);
```

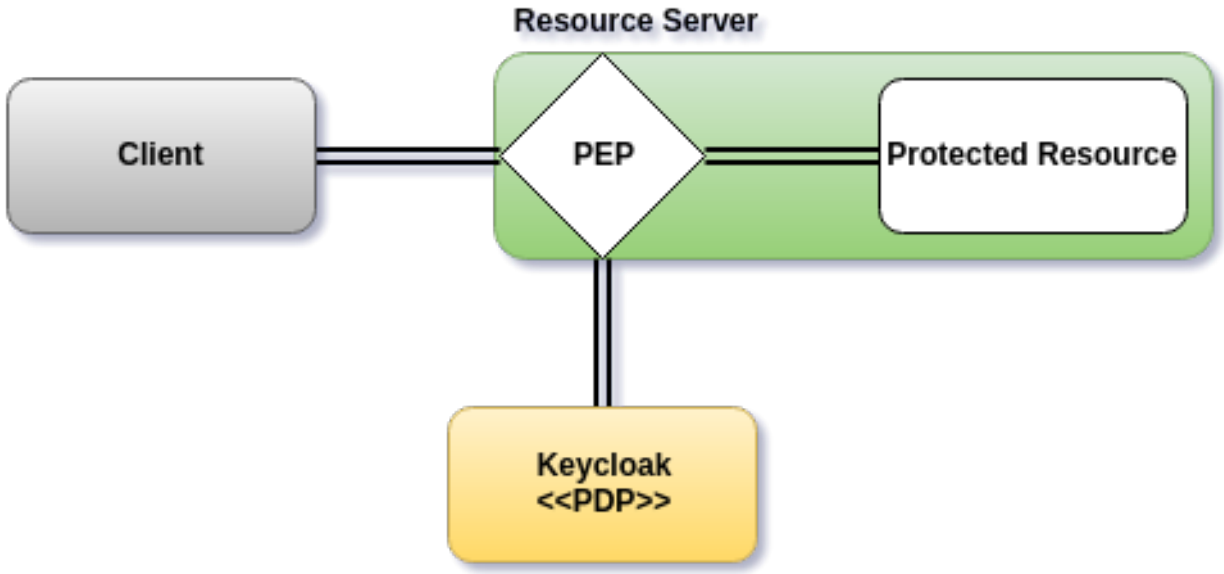
8.6.6. 内省 RPT

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// send the authorization request to the server in order to
// obtain an RPT with all permissions granted to the user
AuthorizationResponse response = authzClient.authorization("alice", "alice").authorize();
String rpt = response.getToken();

// introspect the token
TokenIntrospectionResponse requestingPartyToken =
authzClient.protection().introspectRequestingPartyToken(rpt);

System.out.println("Token status is: " + requestingPartyToken.getActive());
System.out.println("Permissions granted by the server: ");

for (Permission granted : requestingPartyToken.getPermissions()) {
    System.out.println(granted);
}
```

-

第 9 章 策略实施器

策略强制点(PEP)是一种设计模式，因此您可以使用不同的方式实现它。红帽构建的 Keycloak 提供了为不同平台、环境和编程语言实施 PEP 所需的所有方法。红帽构建的 Keycloak 授权服务提供了一个 RESTful API，并使用集中式授权服务器利用 OAuth2 授权功能进行精细的授权。



PEP 负责红帽构建的 Keycloak 服务器的访问决策，其中通过评估与受保护的资源关联的策略来实现这些决策。它充当您的应用中的过滤器或拦截器，根据这些决策授予的权限，检查对受保护的资源的特定请求是否可以满足。

Red Hat build of Keycloak 提供内置的支持，为启用红帽为 Java 应用程序 构建的 Keycloak 策略 Enforcer Enforcer 提供内置支持，以保护 JakartaEE 兼容的框架和 Web 容器。如果使用 Maven，您应该为项目配置以下依赖项：

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-policy-enforcer</artifactId>
    <version>${keycloak.version}</version>
</dependency>
```

当您启用策略强制程序时，发送到应用程序的所有请求都会被截获，并被授予对受保护的资源的访问权限，具体取决于红帽构建 Keycloak 向发出请求的权限。

策略强制通过红帽构建的 Keycloak 管理控制台与应用程序的路径和您为资源服务器创建的资源相关联。???默认情况下，当创建资源服务器时，红帽构建的 Keycloak 会为资源服务器创建一个 默认配置，

以便您可以快速启用策略强制。

9.1. 配置

策略强制器配置使用 JSON 格式，以及大多数不需要设置任何时间，如果您想要根据资源服务器中可用的资源自动解析受保护的路径。

如果要手动定义受保护的资源，您可以使用稍微更详细的格式：

```
{
  "enforcement-mode" : "ENFORCING",
  "paths": [
   {
     "path" : "/users/*",
     "methods" : [
      {
        "method": "GET",
        "scopes" : ["urn:app.com:scopes:view"]
      },
      {
        "method": "POST",
        "scopes" : ["urn:app.com:scopes:create"]
      }
     ]
   }
  ]
}
```

以下是每个配置选项的描述：

- enforcement-mode

  指定如何强制实施策略。

  - ENFORCING

    （默认模式）默认拒绝请求，即使没有策略与给定资源关联。

  - PERMISSIVE

即使没有与给定资源关联的策略，也允许请求。

- DISABLED

完全禁用策略评估并允许访问任何资源。当 强制模式 是 DISABLED 时，应用程序仍然能够通过 Authorization 上下文获取红帽构建的 Keycloak 授予的所有权限

- on-deny-redirect-to

定义在从服务器获取"访问被拒绝"消息时重定向客户端请求的 URL。默认情况下，适配器使用 403 HTTP 状态代码进行响应。

- path-cache

定义策略实施器应如何跟踪应用程序和红帽构建的 Keycloak 中定义的路径之间的关联。为了避免对红帽构建的 Keycloak 服务器进行不必要的请求，需要在路径和受保护的资源之间缓存关联，以避免对红帽构建的 Keycloak 服务器进行不必要的请求。

- Lifespan

定义条目应过期的时间（以毫秒为单位）。如果没有提供，则默认值为 30000。等于 0 的值可以设置为完全禁用缓存。这样设置一个等于 -1 的值来禁用缓存的过期。

- max-entries

定义应保存在缓存中的条目限制。如果没有提供，则默认值为 1000。

- 路径

指定保护的路径。此配置是可选的。如果没有定义，策略强制程序通过在 Keycloak 的红帽构建中获取您定义的资源来发现所有路径，其中这些资源使用代表应用程序中的一些路径的 URIS 定义。

- name

要与给定路径关联的服务器上的资源名称。与 路径 一起使用时，策略强制器会忽略资源的 URIS 属性，并使用您提供的路径。

- path

（必需）相对于应用程序上下文路径的 URI。如果指定了这个选项，策略 enforcer 会使用相同值的 URI 查询服务器以获取资源。目前支持路径匹配的最基本逻辑。有效路径示例包括：

  - 通配符： Attr
  - suffix: Attr.html
  - sub-paths: /pathAttr
  - path 参数： /resource/{id}
  - 完全匹配： /resource
  - Pattern: /{version}/resource, /api/{version}/resource, /api/{version}/resource ldapsearch

- 方法

HTTP 方法（如 GET、POST、PATCH）来保护以及它们如何与服务器中给定资源的范围相关联。

  - 方法

    HTTP 方法的名称。

  - 范围

与方法关联的范围的字符串数组。当您将范围与特定方法关联时，尝试访问受保护的资源（或路径）的客户端必须提供 RPT，授予列表中指定的所有范围的权限。例如，如果您使用范围 create 定义方法 POST，RPT 必须包含在执行向路径执行 POST 时授予 create 范围访问权限的权限。

- scopes-enforcement-mode

对于与方法关联的范围，引用强制模式的字符串。值可以是 ALL 或 ANY。如果 ALL，则必须授予所有定义的范围，才能使用该方法访问资源。如果 ANY，则应至少授予一个范围，才能使用该方法获取资源的访问权限。默认情况下，强制模式设置为 ALL。

- enforcement-mode

指定如何强制实施策略。

- ENFORCING

（默认模式）默认拒绝请求，即使没有与给定资源关联的策略。

- DISABLED

- claim-information-point

定义必须解决并推送到 Keycloak 服务器的红帽构建的一个或多个声明，以便这些声明可供策略使用。如需了解更多详细信息，请参阅声明信息点。

- lazy-load-paths

指定适配器应该如何为与应用程序中的路径关联的资源获取服务器。如果为 true，则策略强制器将相应地获取资源，并请求的路径。当您不希望在部署过程中从服务器获取所有资源（如果您未提供 路径），或者您只定义了一组子路径，并希望按需获取其他路径，则此配置特别有用。

- http-method-as-scope

指定范围应如何映射到 HTTP 方法。如果设置为 true，策略实施器将使用当前请求的 HTTP 方法来检查是否应授予访问权限。启用后，请确保您的 Keycloak 中的资源与代表您要保护的每个 HTTP 方法的范围相关联。

- claim-information-point

定义必须解决并推送到 Keycloak 服务器的红帽构建的一个或多个 全局 声明，以便这些声明可供策略使用。如需了解更多详细信息，请参阅声明信息点。

9.2. 声明信息点

声明信息点(CIP)负责解析声明并将这些声明推送到红帽 Keycloak 服务器，以提供有关策略访问上下文的更多信息。它们可以定义为 policy-enforcer 的配置选项，以便从不同的源解析声明，例如：

- HTTP 请求（参数、标头、正文等）
- 外部 HTTP 服务
- 配置中定义的静态值
- 通过实施 Claim Information Provider SPI 的任何其他源

将声明推送到红帽 Keycloak 服务器构建时，策略不仅可以基于用户是谁，还可以考虑上下文和内容，具体根据谁、原因、何时、位置以及给定交易而考虑。它都关于基于上下文的授权，以及如何使用运行时信息来支持精细的授权决策。

9.2.1. 从 HTTP 请求获取信息

以下是如何从 HTTP 请求中提取声明的几个示例：

keycloak.json

```
{
  "paths": [
```

```
    {
      "path": "/protected/resource",
      "claim-information-point": {
        "claims": {
          "claim-from-request-parameter": "{request.parameter['a']}",
          "claim-from-header": "{request.header['b']}",
          "claim-from-cookie": "{request.cookie['c']}",
          "claim-from-remoteAddr": "{request.remoteAddr}",
          "claim-from-method": "{request.method}",
          "claim-from-uri": "{request.uri}",
          "claim-from-relativePath": "{request.relativePath}",
          "claim-from-secure": "{request.secure}",
          "claim-from-json-body-object": "{request.body['/a/b/c']}",
          "claim-from-json-body-array": "{request.body['/d/1']}",
          "claim-from-body": "{request.body}",
          "claim-from-static-value": "static value",
          "claim-from-multiple-static-value": ["static", "value"],
          "param-replace-multiple-placeholder": "Test {keycloak.access_token['/custom_claim/0']}
and {request.parameter['a']}"
        }
      }
    }
  ]
}
```

9.2.2. 从外部 HTTP 服务获取信息

以下是如何从外部 HTTP 服务中提取声明的几个示例：

keycloak.json

```
{
  "paths": [
    {
      "path": "/protected/resource",
      "claim-information-point": {
        "http": {
          "claims": {
            "claim-a": "/a",
            "claim-d": "/d",
            "claim-d0": "/d/0",
            "claim-d-all": [
              "/d/0",
              "/d/1"
            ]
          },
          "url": "http://mycompany/claim-provider",
```

```
        "method": "POST",
        "headers": {
          "Content-Type": "application/x-www-form-urlencoded",
          "header-b": [
            "header-b-value1",
            "header-b-value2"
          ],
          "Authorization": "Bearer {keycloak.access_token}"
        },
        "parameters": {
          "param-a": [
            "param-a-value1",
            "param-a-value2"
          ],
          "param-subject": "{keycloak.access_token['/sub']}",
          "param-user-name": "{keycloak.access_token['/preferred_username']}",
          "param-other-claims": "{keycloak.access_token['/custom_claim']}"
        }
      }
    }
  ]
}
```

9.2.3. 静态声明

keycloak.json

```
{
  "paths": [
    {
      "path": "/protected/resource",
      "claim-information-point": {
        "claims": {
          "claim-from-static-value": "static value",
          "claim-from-multiple-static-value": ["static", "value"]
        }
      }
    }
  ]
}
```

9.2.4. 声明信息供应商 SPI

当任何内置供应商都不足以解决其要求时，开发人员可以使用 Claim Information Provider SPI 支持不同的声明信息点。

例如，要实施一个新的 CIP 提供程序，您需要在应用的 classpath 中实施 org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory 和 ClaimInformationPointProvider Factory，并提供文件 META-INF/services/org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory。

org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory 示例：

```
public class MyClaimInformationPointProviderFactory implements
ClaimInformationPointProviderFactory<MyClaimInformationPointProvider> {

    @Override
    public String getName() {
        return "my-claims";
    }

    @Override
    public void init(PolicyEnforcer policyEnforcer) {

    }

    @Override
    public MyClaimInformationPointProvider create(Map<String, Object> config) {
        return new MyClaimInformationPointProvider(config);
    }
}
```

每个 CIP 提供程序都必须与名称关联，如 MyClaimInformationPointProviderFactory.getName 方法中定义的。名称将用于从 policy-enforcer 配置中的 claim-information-point 部分映射到实现。

在处理请求时，策略 enforcer 将调用 MyClaimInformationPointProviderFactory.create 方法，以获取 MyClaimInformationPointProvider 的实例。调用时，为此特定 CIP 提供程序定义的任何配置（通过 claim-information-point）作为映射传递。

ClaimInformationPointProvider 示例：

```
public class MyClaimInformationPointProvider implements ClaimInformationPointProvider {

    private final Map<String, Object> config;

    public MyClaimInformationPointProvider(Map<String, Object> config) {
        this.config = config;
```

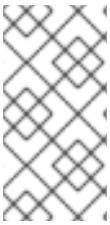
```
    }

    @Override
    public Map<String, List<String>> resolve(HttpFacade httpFacade) {
        Map<String, List<String>> claims = new HashMap<>();

        // put whatever claim you want into the map

        return claims;
    }
}
```

9.3. 获取授权上下文

启用策略强制后，服务器获取的权限可通过 org.keycloak.AuthorizationContext 提供。此类提供了多种方法，您可以用来获取权限，并确定是否授予特定资源或范围的权限。

获取 Servlet 容器中的授权上下文

```
HttpServletRequest request = // obtain javax.servlet.http.HttpServletRequest
AuthorizationContext authzContext = (AuthorizationContext)
request.getAttribute(AuthorizationContext.class.getName());
```



注意

授权上下文可帮助您对服务器做出和返回的决策提供更多控制。例如，您可以使用它来构建隐藏或显示项目的动态菜单，具体取决于与资源或范围关联的权限。

```
if (authzContext.hasResourcePermission("Project Resource")) {
    // user can access the Project Resource
}

if (authzContext.hasResourcePermission("Admin Resource")) {
    // user can access administration resources
}

if (authzContext.hasScopePermission("urn:project.com:project:create")) {
    // user can create new projects
}
```

AuthorizationContext 代表红帽构建的 Keycloak 授权服务的主要功能之一。在上面的示例中，您可以看到受保护的资源不直接与管理它们的策略关联。

使用基于角色的访问控制(RBAC)考虑一些类似的代码：

```
if (User.hasRole('user')) {
    // user can access the Project Resource
}

if (User.hasRole('admin')) {
    // user can access administration resources
}

if (User.hasRole('project-manager')) {
    // user can create new projects
}
```

虽然这两个示例都满足相同的要求，但它们以不同的方式进行配置。在 RBAC 中，角色仅隐式定义其资源的访问权限。使用红帽构建的 Keycloak，您可以创建更易管理的代码，该代码直接专注于资源，无论您使用 RBAC、基于属性的访问控制(ABAC)还是任何其他 BAC 变体。您有给定资源或范围的权限，或者您没有该权限。

现在，假设您的安全要求已更改，并且项目管理器除了项目管理器外，SpmO 也可以创建新项目。

安全要求改变，但红帽构建的 Keycloak 无需更改应用程序代码来满足新的要求。当应用程序基于资源和范围标识符后，您只需要更改与授权服务器中特定资源关联的权限或策略的配置。在本例中，与项目资源关联的权限和策略，或范围 urn:project.com:project:create 将被更改。

9.4. 使用 AUTHORIZATIONCONTEXT 获取授权客户端实例

AuthorizationContext 还可用于获取配置为应用程序的 Authorization Client API 的引用：

```
    ClientAuthorizationContext clientContext =
ClientAuthorizationContext.class.cast(authzContext);
    AuthzClient authzClient = clientContext.getClient();
```

在某些情况下，由策略实施器保护的资源服务器需要访问授权服务器提供的 API。借助 AuthzClient 实例，资源服务器可以与服务器交互，以便以编程方式创建资源或检查特定权限。

9.5. JAVASCRIPT 集成

红帽构建的 Keycloak 服务器附带了一个 JavaScript 库，可用于与策略实施器保护的资源服务器进行交互。这个库基于红帽构建的 Keycloak JavaScript 适配器，可以集成该适配器，以便您的客户端从红帽

构建的 Keycloak 服务器获取权限。

您可以通过在网页中包含以下 脚本 标签，从正在运行的红帽构建的 Keycloak Server 实例获取此库：

```
<script src="http://.../js/keycloak-authz.js"></script>
```

接下来，您可以创建 KeycloakAuthorization 实例，如下所示：

```
const keycloak = ... // obtain a Keycloak instance from keycloak.js library
const authorization = new KeycloakAuthorization(keycloak);
```

keycloak-authz.js 库提供两个主要功能：

- 如果您要访问 UMA 保护的资源服务器，请使用权限票据从服务器获取权限。
- 通过发送应用程序想要访问的资源和范围，从服务器获取权限。

在这两种情况下，该程序库都允许您轻松与资源服务器和红帽构建的 Keycloak 授权服务交互，以获取具有您的客户端权限的令牌，作为 bearer 令牌来访问资源服务器上的受保护的资源。

9.5.1. 处理来自 UMA-Protected 资源服务器的授权响应

如果资源服务器受策略执行者保护，它会根据由 bearer 令牌传输的权限响应客户端请求。通常，当您尝试访问一个没有访问受保护资源权限的 bearer 令牌的资源服务器时，资源服务器会使用 401 状态代码和 WWW-Authenticate 标头来响应。

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="${realm}",
   as_uri="https://${host}:${port}/realms/${realm}",
   ticket="016f84e8-f9b9-11e0-bd6f-0021cc6004de"
```

如需更多信息，请参阅 UMA 授权过程。

您的客户端需要执行的操作是从资源服务器返回的 WWW-Authenticate 标头中提取权限票据，并使用库发送授权请求，如下所示：

-

```
// prepare a authorization request with the permission ticket
const authorizationRequest = {};
authorizationRequest.ticket = ticket;

// send the authorization request, if successful retry the request
Identity.authorization.authorize(authorizationRequest).then(function (rpt) {
    // onGrant
}, function () {
    // onDeny
}, function () {
    // onError
});
```

授权 功能是完全异步的，支持一些回调功能来从服务器接收通知：

- onGrant: 函数的第一个参数。如果授权成功，并且服务器返回具有请求权限的 RPT，则回调会收到 RPT。
- onDeny ：函数的第二个参数。只有服务器拒绝授权请求时调用。
- onError ：函数的第三个参数。只有服务器意外响应时才会调用。

大多数应用应使用 onGrant 回调在 401 响应后重试请求。后续请求应包含 RPT，作为重试的 bearer 令牌。

9.5.2. 获取权利

keycloak-authz.js 库提供了一个 权利 功能，您可以通过提供客户端想要访问的资源和范围，从服务器获取 RPT。

有关如何获取具有用户可访问的所有资源和范围权限的 RPT 的示例

```
authorization.entitlement('my-resource-server-id').then(function (rpt) {
    // onGrant callback function.
    // If authorization was successful you'll receive an RPT
    // with the necessary permissions to access the resource server
});
```

有关如何获取具有特定资源和范围的权限的 RPT 的示例

```
authorization.entitlement('my-resource-server', {
  "permissions": [
    {
      "id" : "Some Resource"
    }
  ]
}).then(function (rpt) {
  // onGrant
});
```

使用 授权 功能时，您必须提供您要访问的资源服务器的 client_id。

授权 功能是完全异步的，支持一些回调功能来从服务器接收通知：

- onGrant: 函数的第一个参数。如果授权成功，并且服务器返回具有请求权限的 RPT，则回调会收到 RPT。
- onDeny ：函数的第二个参数。只有服务器拒绝授权请求时调用。
- onError ：函数的第三个参数。只有服务器意外响应时才会调用。

9.5.3. 授权请求

授权 和授权 功能都接受授权请求对象。此对象可使用以下属性设置：

- 权限

  代表资源和范围的对象数组。例如：

  ```
  const authorizationRequest = {
    "permissions": [
      {
  ```

```
            "id" : "Some Resource",
            "scopes" : ["view", "edit"]
        }
    ]
}
```

- metadata

  一个对象，其属性定义服务器应如何处理授权请求。

  - response_include_resource_name

    如果资源名称应包含在 RPT 的权限中，指示到服务器的布尔值。如果为 false，则仅包含资源标识符。

  - response_permissions_limit

    为 RPT 可以具有的权限数量定义限制的整数 N。与 rpt 参数一同使用时，只有最后 N 个请求的权限才会保存在 RPT 中

- submit_request

  指示服务器是否应该创建权限请求到权限票据引用的资源和范围的布尔值。这个参数仅在与 ticket 参数一起使用时生效，作为 UMA 授权过程的一部分。

9.5.4. 获取 RPT

如果您已使用库提供的任何授权功能获得 RPT，您可以始终从授权对象获取 RPT （假设它已被前面显示的技术之一初始化）：

```
const rpt = authorization.rpt;
```

9.6. 配置 TLS/HTTPS

当服务器使用 HTTPS 时，请确保您的策略强制器配置如下：

```
{
  "truststore": "path_to_your_trust_store",
```

```
  "truststore-password": "trust_store_password"
}
```

以上配置启用了 TLS/HTTPS 到授权客户端，以便使用 HTTPS 方案远程访问红帽 Keycloak 服务器的红帽构建。

注意

强烈建议您在访问红帽构建的 Keycloak 服务器端点时启用 TLS/HTTPS。