Red Hat build of Apache Camel 4.4

HawtIO Diagnostic Console Guide

使用红帽构建的 HawtIO 管理应用程序

Last Updated: 2024-06-12

Red Hat build of Apache Camel 4.4 HawtIO Diagnostic Console Guide

使用红帽构建的 HawtIO 管理应用程序

法律通告

摘要

当您部署支持 HawtIO 的应用程序时，您可以使用 HawtIO 监控并与集成交互。

目录

前言

HawtIO 提供用于查看和管理启用了红帽 HawtIO 的应用程序的企业监控工具。它是一种基于 Web 的控制台，可从浏览器访问以监控和管理已启用 HawtIO 的容器。HawtIO 基于开源 HawtIO 软件 (https://hawt.io/)。HawtIO 诊断控制台指南描述了如何使用 HawtIO 管理应用程序。

本指南的受众是 Apache Camel eco-system 开发人员和管理员。本指南假定对 Apache Camel 和您组织的处理要求有一定的了解。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

第 1 章 HAWTIO 概述

HawtIO 是红帽构建的 Apache Camel 和红帽构建的 AMQ 的诊断控制台。它是使用现代 Web 技术（如 React 和 PatternFly ）构建的可插拔 Web 诊断控制台。HawtIO 提供了一个中央接口，用于检查和管理一个或多个启用了 HawtIO 的容器的详细信息。当您将 HawtIO 独立安装或使用 OpenShift 中的 HawtIO 时，可以使用 HawtIO。您可以在 HawtIO 中查看和管理的集成取决于正在运行的插件。您可以监控 HawtIO 和系统资源，执行更新，以及启动或停止服务。

可插拔架构基于 Webpack Module Federation，具有高度可扩展的；您可以使用插件动态扩展 HawtIO，或者在 JVM 中自动发现插件。HawtIO 已内置 插件 可供您的 JVM 应用开箱即用。插件包括 Apache Camel、连接、JMX、日志、运行时、Qartz 和 Spring Boot。HawtIO 主要设计为用于 Camel Quarkus 和 Camel Spring Boot。它还是用于管理微服务应用的工具。HawtIO 是云原生的；它已准备好接管云！您可以使用 HawtIO Operator 将它部署到 Kubernetes 和 OpenShift。

HawtIO 的优点如下：

- 通过 JMX 的运行时管理 JVM，特别是 Camel 应用程序和 AMQ 代理（带有特殊视图）
- Camel 路由的视觉化和调试/追踪
- 简单管理和监控应用程序指标

第 2 章 INSTALLING HAWTIO

使用 HawtIO 控制台有几个选项：

- 从 CLI 运行 HawtIO 独立（分离模式）(JBang)
- 运行嵌入在 Quarkus 应用程序中的 HawtIO
- 运行嵌入在 Spring Boot 应用程序中的 HawtIO

2.1. 将红帽软件仓库添加到 MAVEN

要访问位于 Red Hat Maven 存储库中的工件，您需要将这些存储库添加到 Maven 的 **settings.xml** 文件中。Maven 在用户主目录的 **.m2** 目录中查找 **settings.xml** 文件。如果没有用户指定的 **settings.xml** 文件，Maven 将使用 **M2_HOME/conf/ settings.xml** 中的系统级 settings.xml 文件。

先决条件：

您知道要在其中添加红帽软件仓库的 **settings.xml** 文件的位置。

流程

1. 在 **settings.xml** 文件中，为红帽 **软件仓库** 添加软件仓库元素，如下例所示：

```
<?xml version="1.0"?>
<settings>

 <profiles>
  <profile>
   <id>extra-repos</id>
   <activation>
    <activeByDefault>true</activeByDefault>
   </activation>
   <repositories>
    <repository>
     <id>redhat-ga-repository</id>
     <url>https://maven.repository.redhat.com/ga</url>
     <releases>
      <enabled>true</enabled>
     </releases>
     <snapshots>
      <enabled>false</enabled>
     </snapshots>
    </repository>
    <repository>
     <id>redhat-ea-repository</id>
     <url>https://maven.repository.redhat.com/earlyaccess/all</url>
     <releases>
      <enabled>true</enabled>
     </releases>
     <snapshots>
      <enabled>false</enabled>
     </snapshots>
    </repository>
   </repositories>
   <pluginRepositories>
```

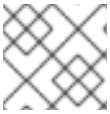
```
      <pluginRepository>
        <id>redhat-ga-repository</id>
        <url>https://maven.repository.redhat.com/ga</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
      <pluginRepository>
        <id>redhat-ea-repository</id>
        <url>https://maven.repository.redhat.com/earlyaccess/all</url>
        <releases>
          <enabled>true</enabled>
        </releases>
        <snapshots>
          <enabled>false</enabled>
        </snapshots>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>

<activeProfiles>
  <activeProfile>extra-repos</activeProfile>
</activeProfiles>

</settings>
```

2.2. 通过 CLI 运行(JBANG)

您可以使用 JBang 从 CLI 安装并运行 HawtIO。



注意

如果您还没有 JBang，首先安装它 ：https://www.jbang.dev/download/

流程

1. 使用 **jbang** 命令在您的机器上安装最新的 HawtIO：

```
$ jbang app install -Dhawtio.jbang.version=4.0.0.redhat-00072 hawtio@hawtio/hawtio
```

注意

这个安装方法仅适用于 *jbang>=0.115.0*。

2. 它将安装 HawtIO 命令。使用以下命令启动 HawtIO 实例：

```
$ hawtio
```

3. 该命令将自动打开位于 http://localhost:8080/hawtio/ 的控制台。要更改端口号，请运行以下命令：

```
$ hawtio --port 8090
```

4. 如需有关 CLI 配置选项的更多信息，请运行以下代码：

```
$ hawtio --help
Usage: hawtio [-hjoV] [-c=<contextPath>] [-d=<plugins>] [-e=<extraClassPath>]
              [-H=<host>] [-k=<keyStore>] [-l=<warLocation>] [-p=<port>]
              [-s=<keyStorePass>] [-w=<war>]
Run Hawtio
  -c, --context-path=<contextPath>
                        Context path.
  -d, --plugins-dir=<plugins>
                        Directory to search for .war files to install as 3rd
                          party plugins.
  -e, --extra-class-path=<extraClassPath>
                        Extra class path.
  -h, --help            Print usage help and exit.
  -H, --host=<host>     Hostname to listen to.
  -j, --join            Join server thread.
  -k, --key-store=<keyStore>
                        JKS keyStore with the keys for https.
  -l, --war-location=<warLocation>
                        Directory to search for .war files.
  -o, --open-url        Open the web console automatic in the web browser.
  -p, --port=<port>     Port number.
  -s, --key-store-pass=<keyStorePass>
                        Password for the JKS keyStore with the keys for https.
  -V, --version         Print Hawtio version
  -w, --war=<war>       War file or directory of the hawtio web application.
```

2.3. 运行 QUARKUS 应用程序

您可以在一个步骤中将 HawtIO 附加到 Quarkus 应用程序。

流程

1. 为 **pom.xml** 中的依赖项添加 **io.hawt:hawtio-quarkus** 和支持 Camel Quarkus 扩展：

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>io.hawt</groupId>
      <artifactId>hawtio-bom</artifactId>
      <version>4.0.0.redhat-00072</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
  <!-- ... other BOMs or dependencies ... -->
</dependencyManagement>
```

```
<dependencies>
 <dependency>
  <groupId>io.hawt</groupId>
  <artifactId>hawtio-quarkus</artifactId>
 </dependency>

  <!-- Mandatory for enabling Camel management via JMX / Hawtio -->
 <dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-management-starter</artifactId>
 </dependency>

 <!-- (Optional) Required for Hawtio Camel route diagram tab -->
 <dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-jaxb</artifactId>
 </dependency>

 <!-- ... other dependencies ... -->
</dependencies>
```

2. 在开发模式中使用 Quarkus 应用程序运行 HawtIO，如下所示：

```
mvn compile quarkus:dev
```

3. 打开 http://localhost:8080/hawtio/ 以查看 HawtIO 控制台。

2.4. 运行 SPRING BOOT 应用程序

您可以通过两个步骤将 HawtIO 附加到 Spring Boot 应用程序。

流程

1. 将 **io.hawt:hawtio-springboot** 和支持 Camel Spring Boot 启动程序添加到 **pom.xml** 中的依赖项：

```
<dependencyManagement>
 <dependencies>
  <dependency>
   <groupId>io.hawt</groupId>
   <artifactId>hawtio-bom</artifactId>
   <version>4.0.0.redhat-00072</version>
   <type>pom</type>
   <scope>import</scope>
  </dependency>
  <!-- ... other BOMs or dependencies ... -->
 </dependencies>
</dependencyManagement>

<dependencies>
 <dependency>
  <groupId>io.hawt</groupId>
  <artifactId>hawtio-springboot</artifactId>
 </dependency>
```

```
 <!-- Mandatory for enabling Camel management via JMX / Hawtio -->
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-management-starter</artifactId>
</dependency>

<!-- (Optional) Required for Hawtio Camel route diagram tab -->
<dependency>
  <groupId>org.apache.camel.springboot</groupId>
  <artifactId>camel-spring-boot-xml-starter</artifactId>
</dependency>

 <!-- ... other dependencies ... -->
</dependencies>
```

2. 通过在 **application.properties** 中添加以下行来启用 HawtIO 和 Jolokia 端点：

```
spring.jmx.enabled = true
management.endpoints.web.exposure.include = hawtio,jolokia
```

3. 在开发模式下使用 Spring Boot 应用程序运行 HawtIO，如下所示：

```
mvn spring-boot:run
```

4. 打开 http://localhost:8080/actuator/hawtio 以查看 HawtIO 控制台。

2.4.1. 配置 HawtIO 路径

如果您不希望具有 HawtIO 端点的 **/actuator** 基本路径，您也可以执行以下操作：

1. 使用 **management.endpoints.web.base-path** 属性自定义 Spring Boot 管理基础路径：

```
management.endpoints.web.base-path = /
```

2. 您还可以通过设置 **management.endpoints.web.path-mapping.hawtio** 属性来自定义 HawtIO 端点的路径：

```
management.endpoints.web.path-mapping.hawtio = hawtio/console
```

3. **Example:**

   a. 有一个正常工作的 Spring Boot 示例，它演示了如何监控一个 Web 应用程序，它公开 Apache Camel 路由、指标等信息，其中包含 HawtIO Spring Boot 示例。

   b. 用于实时值和图表的良好 MBean 是 **java.lang/OperatingSystem**。尝试查看 Camel 路由。请注意，当您更改树中的选择时，根据内容动态更改可用选项卡列表。

第 3 章 配置 HAWTIO

HawtIO 及其插件可以通过系统属性配置其行为。

3.1. 配置属性

下表列出了 HawtIO 核心系统和各种插件的配置属性。

System	default	描述
hawtio.disableProxy	false	将此属性设置为 true 时，ProxyServlet (/hawtio/proxyAttr)可以被禁用。这使得 Connect 插件不可用，这意味着 HawtIO 不再连接到远程 JVM，但有时用户可能希望这样做，因为没有使用 Connect 插件时，用户可能希望这样做。
hawtio.localAddressProbing	true	是否启用代理允许列表的本地地址探测。将此属性设置为 false 以禁用它。
hawtio.proxyAllowlist	localhost, 127.0.0.1	Connect 插件可以通过 ProxyServlet 连接到的目标主机的逗号分隔列表。出于安全原因，未在此允许列表中列出的所有主机都被拒绝进行连接。这个选项可以设置为 * 以允许所有主机。使用 "r:" 为列表添加前缀可以定义正则表达式（例如：localhost,r:myserver[0-9]+.mydomain.com）
hawtio.redirect.scheme		需要身份验证时，方案是将 URL 重定向到登录页面。
hawtio.sessionTimeout		servlet 容器将在客户端访问之间保持打开的最大时间间隔（以秒为单位）。如果没有配置这个选项，则 HawtIO 将使用 servlet 容器的默认会话超时。

3.1.1. Quarkus

对于 Quarkus，所有这些属性都可以在 **application.properties** 或 **application.yaml** 中配置，且带有 **quarkus.hawtio** 前缀。

例如：

```
quarkus.hawtio.disableProxy = true
```

3.1.2. Spring Boot

对于 Spring Boot，所有这些属性都可以在 **application.properties** 或 **application.yaml** 中进行配置，如下所示。

例如：

```
hawtio.disableProxy = true
```

3.2. 通过系统属性配置 JOLOKIA

Jolokia 代理使用 **io.hawt.web.JolokiaConfiguredAgentServlet** 自动部署，它扩展了 Jolokia 原生 **org.jolokia.http.AgentServlet** 类，在 **hawtio-war/WEB-INF/web.xml** 中定义。如果要使用 Jolokia 文档中定义的配置参数自定义 Jolokia Servlet，您可以将它们作为前缀 **jolokia** 传递为系统属性。

例如：

```
jolokia.policyLocation = file:///opt/hawtio/my-jolokia-access.xml
```

3.2.1. RBAC Restrictor

对于一些支持 HawtIO RBAC （基于角色的访问控制）的运行时，HerwtIO 提供了一个自定义 Jolokia Restrictor 实现，它根据 ACL （访问控制列表）策略提供额外的保护功能。

警告

您不能在 Quarkus 和 Spring Boot 中使用 HawtIO RBAC。在这些运行时上启用 RBAC 限制仅会带来额外的负载，而无需任何好处。

要激活 HawtIO RBAC Restrictor，请通过 System 属性配置 Jolokia 参数 **restrictorClass** 以使用 **io.hawt.web.RBACRestrictor**，如下所示：

```
jolokia.restrictorClass = io.hawt.system.RBACRestrictor
```

第 4 章 HAWTIO 的安全性和身份验证

HawtIO 根据其运行的 runtime/containers，启用开箱即用的身份验证。要将 HawtIO 与应用程序一起使用，需要为运行时设置身份验证或禁用 HawtIO 身份验证。

4.1. 配置属性

下表列出了 HawtIO 核心系统的安全相关配置属性。

Name	默认	描述
hawtio.authenticationContainerDiscoveryClasses	**io.hawt.web.tomcat.TomcatAuthenticationContainerDiscovery**	以逗号分开的已用 AuthenticationContainerDiscovery 实现列表。默认情况下，只有 TomcatAuthenticationContainerDiscovery，用于验证来自 tomcat-users.xml 文件的 Tomcat 用户。如果要从配置的 JAAS 登录模块验证 Tomcat 上的用户，或者可以自由地添加更多您自己的类，可以将其删除。
hawtio.authenticationContainerTomcatDigestAlgorithm	NONE	使用 Tomcat tomcat-users.xml 文件时，密码可以散列而不是纯文本。使用它来指定摘要算法；有效值为 NONE、MD5、SHA-256、SHA-256、SHA-384 和 SHA-512。
hawtio.authenticationEnabled	true	是否启用安全性。
hawtio.keycloakClientConfig	**classpath:keycloak.json**	用于前端的 Keycloak 配置文件。如果启用了 Keycloak 集成，则是必须的。
hawtio.keycloakEnabled	false	是否启用或禁用 Keycloak 集成。
hawtio.noCredentials401	false	在启用身份验证时是否返回 HTTP 状态 401，但没有提供凭证。返回 401 将导致浏览器弹出窗口提示凭据。默认情况下，这个选项为 false，返回 HTTP 状态 403。
hawtio.realm	hawtio	用于登录的安全域。
hawtio.rolePrincipalClasses		完全限定的主体类名称。逗号可以分隔多个类。
hawtio.roles	admin, manager, viewer	登录控制台需要用户角色。逗号可以分隔多个允许的角色。设置为 * 或空值，在 HawtIO 验证用户时禁用角色检查。
hawtio.tomcatUserFileLocation	**conf/tomcat-users.xml**	指定 tomcat-users.xml 文件的替代位置，如 /production/userlocation/。

4.2. QUARKUS

HawtIO 使用 Quarkus 和 Keycloak 提供的验证机制进行保护。

如果要为 Quarkus 禁用 HawtIO 身份验证，请在 **application.properties** 中添加以下配置：

```
quarkus.hawtio.authenticationEnabled = false
```

4.2.1. Quarkus 身份验证机制

HawtIO 只是 Quarkus 中的一个 Web 应用程序，因此各种机制 Quarkus 提供用来验证 Web 应用程序的方式与验证 HawtIO 一样。

在这里，我们将如何通过 HawtIO 使用 基于属性的身份验证 来实现演示目的。

重要

不建议在生产环境中使用基于属性的身份验证。这种机制仅用于开发和测试目的。

1. 要将基于属性的身份验证与 HawtIO 搭配使用，请将以下依赖项添加到 **pom.xml** 中：

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-elytron-security-properties-file</artifactId>
</dependency>
```

2. 然后您可以在 **application.properties** 中定义用户来启用身份验证。例如，*使用密码* **s3cr3t!** 和 *role* **admin** 定义 *用户* **hawtio** 类似如下：

```
quarkus.security.users.embedded.enabled = true
quarkus.security.users.embedded.plain-text = true
quarkus.security.users.embedded.users.hawtio = s3cr3t!
quarkus.security.users.embedded.roles.hawtio = admin
```

Example:

有关基于属性的身份验证的工作示例，请参阅 Quarkus 示例。

4.2.2. 使用 Keycloak 的 Quarkus

请参阅 Keycloak 集成 - Quarkus。

4.3. SPRING BOOT

除了标准的 JAAS 身份验证外，Spring Boot 上的 HawtIO 还可以通过 Spring Security 或 Keycloak 进行保护。如果要为 Spring Boot 禁用 HawtIO 身份验证，请在 **application.properties** 中添加以下配置：

```
hawtio.authenticationEnabled = false
```

4.3.1. Spring Security

将 Spring Security 与 HawtIO 搭配使用：

1. 将 **org.springframework.boot:spring-boot-starter-security** 添加到 **pom.xml** 中的依赖项：

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

2. **src/main/resources/application.properties** 中的 Spring Security 配置应该类似如下：

```
spring.security.user.name = hawtio
spring.security.user.password = s3cr3t!
spring.security.user.roles = admin,viewer
```

3. 必须定义安全配置类来设置如何使用 Spring Security 保护应用程序：

```
@EnableWebSecurity
public class SecurityConfig
{
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
   {
      http.authorizeRequests().anyRequest().authenticated()
         .and()
         .formLogin()
         .and()
         .httpBasic()
         .and()
         .csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse());
      return http.build();
   }
}
```

Example:

有关工作 示例，请参阅 springboot-security 示例。

4.3.1.1. 使用 Spring Security 连接到远程应用程序

如果您试图连接到启用了 Spring Security 的远程 Spring Boot 应用程序，请确保 Spring Security 配置允许从 HawtIO 控制台访问。最有可能，默认的 CSRF 保护会禁止远程访问 Jolokia 端点，因此会在 HawtIO 控制台中导致身份验证失败。

警告

请注意，它会将您的应用程序暴露于 CSRF 攻击的风险。

1. 最简单的解决方案是在远程应用中禁用 Jolokia 端点的 CSRF 保护，如下所示：

```
import org.springframework.boot.actuate.autoconfigure.jolokia.JolokiaEndpoint;
import org.springframework.boot.actuate.autoconfigure.security.servlet.EndpointRequest;
```

```
@EnableWebSecurity
public class SecurityConfig
{

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
    {
        ...
        // Disable CSRF protection for the Jolokia endpoint
        http.csrf().ignoringRequestMatchers(EndpointRequest.to(JolokiaEndpoint.class));
        return http.build();
    }

}
```

2. 要保护 Jolokia 端点，即使没有 Spring Security 的 CSRF 保护，您需要在 **src/main/resources/** 下提供一个 **jolokia-access.xml** 文件，类似以下(snippet)，以便只有可信节点可以访问它：

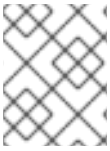
```
<restrict>
  ...
  <cors>
    <allow-origin>http*://localhost:*</allow-origin>
    <allow-origin>http*://127.0.0.1:*</allow-origin>
    <allow-origin>http*://*.example.com</allow-origin>
    <allow-origin>http*://*.example.com:*</allow-origin>

    <strict-checking />
  </cors>
</restrict>
```

4.3.2. 带有 Keycloak 的 Spring Boot

请参阅 Keycloak 集成 - Spring Boot。

第 5 章 在 OPENSHIFT 4 中设置 HAWTIO



注意

虽然 HawtIO 在线应该能够发现 Fuse 7 应用，但包含的 Camel 插件仅支持 Camel 4.x 模型。很可能无法使用 HawtIO 4 管理 Fuse 7 Camel 路由。

在 OpenShift 4.x 上，设置 HawtIO 涉及安装和部署它。此安装的首选机制是使用 OperatorHub 提供的 HawtIO Operator （请参阅第 5.1 节，"使用 OperatorHub 在 OpenShift 4.x 上安装和部署 HawtIO"）。另外，您可以为 HawtIO 自定义基于角色的访问控制(RBAC)，如 OpenShift 4.x 上的 HawtIO 第 2.3 节 "Role-based Access control for HawtIO" 所述。

5.1. 使用 OPERATORHUB 在 OPENSHIFT 4 上安装和部署 HAWTIO

HawtIO Operator 在 OpenShift OperatorHub 中提供，用于安装 HawtIO。要部署 HawtIO，您必须部署已安装 Operator 的实例以及 HawtIO 自定义资源(CR)。

安装和部署 HawtIO：

1. 以具有集群管理员访问权限的用户身份登录 Web 浏览器中的 OpenShift 控制台。

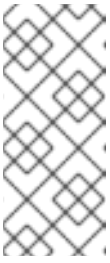
2. **点 Operators，然后点 OperatorHub。**

3. **在搜索字段窗口中，键入 HawtIO 来过滤 Operator 列表。点 HawtIO Operator。**

4. **在 HawtIO Operator 安装窗口中，点 Install。Create Operator Subscription 表单将打开：**

   a. **对于 更新频道，请选择 stable-v1。**

   b. **对于 Installation Mode，接受默认值（集群中的特定命名空间）。**

   注意

   此模式决定了 Operator 将监控 HawtIO CR 的命名空间。这与命名空间 HawtIO 在被完全部署时监控的命名空间不同。后者可以通过 HawtIO CR 配置。

   c. **对于 Installed Namespace，选择要在其中安装 HawtIO Operator 的命名空间。**

d. 对于 **Update Approval**，选择 **Automatic** 或 **Manual** 来配置 OpenShift 如何处理对 HawtIO Operator 的更新。

i. 如果选择了 **Automatic updates** 选项，并且有新版本的 HawtIO Operator 可用，OpenShift Operator Lifecycle Manager (OLM)将自动升级正在运行的 HawtIO 实例，而无需人为干预；

ii. 如果选择了 **Manual** 更新选项，且有新版本的 Operator 可用，OLM 只会创建一个更新请求。然后，集群管理员必须手动批准更新请求，才能将 HawtIO Operator 更新至新版本。

5. 点 **Install** 和 OpenShift 将 HawtIO Operator 安装到当前命名空间中。

6. 要验证安装，点 ***Operators***，然后点 ***Installed Operators***。HawtIO 应显示在操作器列表中。

7. 使用 OpenShift Web 控制台部署 HawtIO：

a. 在 **Installed Operators** 列表中，在 **Name** 列下点 **HawtIO Operator**。

b. 在 **Provided APIs** 下的 **Operator Details** 页面中，点 **Create HawtIO**。

c. 接受配置默认值或选择性地编辑它们。

i. 对于 **Replicas**，若要提高 HawtIO 性能（例如在高可用性环境中），可以增加分配给 HawtIO 的 pod 数量；

ii. 对于 **RBAC** （基于角色的访问控制），只有在 **Config Map** 字段中指定一个值，如果要自定义默认的 RBAC 行为，并且是否安装了 HawtIO Operator 的命名空间中已存在 ConfigMap 文件

iii. 对于 **Nginx**，请参阅 **HawtIO Operator 安装的性能调优**

iv. 对于 **Type**，指定：

A. **集群** ：用于为任何支持 **HawtIO** 的应用程序监控 **OpenShift** 集群上的所有命名空间；

B. ***命名空间*** **：用于仅监控在同一命名空间中部署的启用了 HawtIO 的应用程序。**

d. ***点 Create。HawtIO Operator Details 页面将打开并显示部署的状态。***

8. ***打开 HawtIO ：***

a. ***对于 命名空间 部署：在 OpenShift Web 控制台中，打开安装 HawtIO 操作器的项目，然后选择 Overview。在 Project Overview 页面中，向下滚动到 Launcher 部分，再单击 HawtIO 链接。***

b. ***对于 集群部署，在 OpenShift Web 控制台的标题栏中，单击网格图标。在弹出菜单中，单击 Red Hat Applications，单击 HawtIO URL 链接。***

c. ***登录 HawtIO。浏览器中打开了 Authorize Access 页面，其中列出了所需权限。***

d. ***点击 Allow selected permissions。HawtIO 在浏览器中打开，并显示有权访问的任何 HawtIO 的应用程序 pod。***

9. ***点 Connect 查看被监控的应用程序。此时将打开一个新浏览器窗口，显示 HawtIO 中的应用程序。***

5.2. OPENSHIFT 4 上 HAWTIO 的基于角色的访问控制

HawtIO 提供基于角色的访问控制(RBAC)，它根据 OpenShift 提供的用户授权推断访问。在 HawtIO 中，RBAC 决定用户在 pod 上执行 MBean 操作。

如需有关 OpenShift 授权的信息，请参阅 OpenShift 文档中的使用 RBAC 定义和应用权限 部分。

当使用 Operator 在 OpenShift 上安装 HawtIO 时，默认启用基于角色的访问控制。HawtIO RBAC 利用 OpenShift 中的 pod 资源的操作动词访问，以确定用户在 HawtIO 中对 pod 的 MBean 操作的访问。默认情况下，对于 HawtIO，有两个用户角色：

1. admin ：如果用户可以在 OpenShift 中更新 pod，则用户会被限制掉 HawtIO 的 admin 角色。用户可以在 HawtIO 中为 pod 执行写 MBean 操作。

2. Viewer: 如果用户可以在 OpenShift 中获取 pod，则用户会被限制掉 HawtIO 的 viewer 角色。用户可以在 HawtIO 中为 pod 执行只读 MBean 操作。

5.2.1. 确定 OpenShift 4 上 HawtIO 的访问角色

HawtIO 基于角色的访问控制是从用户的 pod 的 OpenShift 权限中推断出来的。要确定向特定用户授予 HawtIO 访问角色，请获取授予 Pod 用户的 OpenShift 权限。

先决条件：

- 用户名
- pod 的名称

流程 ：

1. 要确定用户是否为 pod 具有 HawtIO admin 角色，请运行以下命令来查看用户是否可以更新 OpenShift 上的 pod：

```
oc auth can-i update pods/<pod> --as <user>
```

2. 如果响应是 yes，则用户具有 Pod 的 admin 角色。用户可以对 pod 执行 HawtIO 中的写入操作。

3. 要确定用户是否为 pod 具有 HawtIO viewer 角色，请运行以下命令来查看用户是否可以在 OpenShift 上获取 pod：

```
oc auth can-i get pods/<pod> --as <user>
```

4. 如果响应是 yes，则用户具有 pod 的 viewer 角色。用户可以在 HawtIO 中为 pod 执行只读操作。根据上下文，HawtIO 会阻止具有 viewer 角色的用户执行写入 MBean 操作，方法是禁用一个选项，或者在 用户尝试写入 MBean 操作时显示这个用户消息允许的操作。

5. 如果没有响应，用户不会绑定到任何 HawtIO 角色，用户无法在 HawtIO 中查看 pod。

5.2.2. 在 OpenShift 4 中自定义对 HawtIO 的基于角色的访问权限

如果您使用 OperatorHub 安装 HawtIO，则默认启用基于角色的访问控制(RBAC)。要自定义 HawtIO RBAC 行为，在部署 HawtIO 之前，必须提供 ConfigMap 资源（定义自定义 RBAC 行为）。此 ConfigMap 的名称应在 HawtIO 自定义资源(CR)的 rbac 配置部分中输入。

自定义 ConfigMap 资源必须添加到安装了 HawtIO Operator 的同一命名空间中。

先决条件 ：

- HawtIO Operator 已从 OperatorHub 安装。

流程 ：

自定义 HawtIO RBAC 角色：

1. 创建 RBAC ConfigMap：

   a. 确保当前的 OpenShift 项目是您要安装 HawtIO 的项目。例如，要在 hawtio-test 项目中安装 HawtIO，请运行以下命令：

   ```
   oc project hawtio-test
   ```

   b. 从模板创建 HawtIO RBAC ConfigMap 文件，并运行这个命令：

```
oc process -f https://raw.githubusercontent.com/hawtio/hawtio-online/2.x/docker/ACL.yaml -p APP_NAME=custom-hawtio | oc create -f -
```

c. 使用以下命令编辑新的自定义 ConfigMap：

```
oc edit ConfigMap custom-hawtio-rbac
```

d. 通过保存编辑，ConfigMap 资源将更新

2. 按照上述所述，创建一个新的 HawtIO CR，并通过在属性 configMap 下添加新 ConfigMap 的名称来编辑 rbac 部分。

3. 点 Create。Operator 应该部署一个新版本的 HawtIO，以使用自定义 ConfigMap

5.3. 从 FUSE 控制台迁移

HawtIO 自定义资源定义(CRD)的版本已从 v1alpha1 升级到 v1。这意味着在安装 HawtIO 操作器后，所有现有 Fuse-Console 自定义资源(CR)将升级到这个新版本。CRD 的当前模式属性保持不变。

CRD 版本属性保留在 CRD 中，但 HawtIO 操作器不再用于安装 HawtIO；它保留下来，因此 Fuse-Console 操作器仍能够正确安装 Fuse-Console。

HawtIO 和 Fuse-Console 应作为独立和独立的应用程序执行。

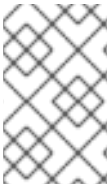
5.4. 在 OPENSHIFT 4 上升级 HAWTIO

Red Hat OpenShift 4.x 处理对 Operator 的更新，包括 HawtIO operator。如需更多信息，请参阅 Operator OpenShift 文档。另外，Operator 更新会根据应用程序的配置方式触发应用程序升级。

5.5. 在 OPENSHIFT 4 中调整 HAWTIO 的性能

默认情况下，HerwtIO 使用以下 Nginx 设置：

- *clientBodyBufferSize: 256k*
- *proxyBuffers: 16 128k*
- *subrequestOutputBufferSize: 10m*



注意

有关这些设置的描述，请参阅 Nginx 文档。

要调整 HawtIO 的性能，您可以设置任何 clientBodyBufferSize、proxyBuffers 和 subrequestOutputBufferSize 环境变量。例如，如果您使用 HawtIO 监控多个 pod 和路由（例如，共 100 个路由），您可以通过将 HawtIO 的 subrequestOutputBufferSize 环境变量设置为 100 m 来解决加载超时问题。

5.5.1. 对 HawtIO Operator 安装的性能调整

在 Openshift 4.x 上，您可以在部署 HawtIO 之前或之后设置 Nginx 性能调优环境变量。如果您随后这样做，OpenShift 会重新部署 HawtIO。

先决条件 ：

- *您必须具有 集群管理员对 OpenShift 集群的访问权限。*

流程 ：

您可以在部署 HawtIO 之前或之后设置环境变量。

1. *在部署 HawtIO 前设置环境变量 ：*
   a. *在 OpenShift Web 控制台中，安装有 HawtIO Operator 的项目，选择 Operators> Installed Operators> HawtIO Operator。*

b. *单击 **HawtIO** 选项卡，然后单击 **Create HawtIO**。*

c. *在 **Create HawtIO** 页面中，在 **Form** 视图中，向下滚动到 **Config> Nginx** 部分。*

d. *展开 **Nginx** 部分，然后设置环境变量。例如：*

i. ***clientBodyBufferSize: 256k***

ii. ***proxyBuffers: 16 128k***

iii. ***subrequestOutputBufferSize: 100m***

e. *点 **Create deploy HawtIO**。*

f. *部署完成后，打开 **Deployments> HawtIO-console** 页面，然后单击 **Environment** 以验证环境变量是否在列表中。*

2. *在部署 **HawtIO** 后设置环境变量 ：*

a. *在 **OpenShift Web** 控制台中，打开部署 **HawtIO** 的项目。*

b. *选择 **Operators> Installed Operators> HawtIO Operator**。*

c. *单击 **HawtIO** 选项卡，然后单击 **HawtIO**。*

d. *选择 **Actions> Edit HawtIO**。*

e. *在 **Editor** 窗口中，滚动到 **spec** 部分。*

f.

在 spec 部分，添加新的 nginx 部分并指定一个或多个环境变量，例如：

```
apiVersion: hawt.io/v1
kind: Hawtio
metadata:
  name: hawtio-console
spec:
  type: Namespace
  nginx:
    clientBodyBufferSize: 256k
    proxyBuffers: 16 128k
    subrequestOutputBufferSize: 100m
```

g.

点击 Save。OpenShift 重新部署 HawtIO。

h.

重新部署完成后，打开 Workloads> Deployments> HawtIO-console 页面，然后点 Environment 来查看列表中的环境变量。

5.5.2. 在 HawtIO 中查看应用程序的性能调整

增强的 HawtIO 的性能调节功能允许查看具有大量 MBeans 的应用程序。要使用此功能，请执行以下步骤。

先决条件 ：

- *您必须具有 集群管理员对 OpenShift 集群的访问权限。*

流程 ：

增加应用程序的内存限值。

1.

在部署 HawtIO 后增加内存限值 ：

a.

在 OpenShift Web 控制台中，打开部署 HawtIO 的项目。

.

b. *选择 **Operators> Installed Operators> HawtIO Operator。***

c. *单击 **HawtIO** 选项卡，然后单击 **HawtIO。***

d. *选择 **Actions> Edit HawtIO。***

e. ***在 Editor 窗口中，向下滚动到 spec.resources 部分。***

f. ***将 请求和限值 的值更新为首选数量***

g. ***点 Save***

h. ***HawtIO** 应该使用新的资源规格重新部署。*

第 6 章 为 HAWTIO 在线设置应用程序

本节演示了如何在 OpenShift 上部署 Camel Quarkus 应用程序，并使用 Camel Quarkus 使它启用了 HawtIO。在 OpenShift 上部署后，它将由 HawtIO 在线发现。

1. 此项目 使用 Quarkus 容器镜像 和 Kubernetes 扩展来构建容器镜像，并将其部署到 Kubernetes/OpenShift 集群(pom.xml)。

2. 启用了 HawtIO 的配置条款中最重要的部分在 < properties> 部分中定义。要使它启用了 HawtIO，必须将 Jolokia 代理附加到配置了 HTTPS 和 SSL client-authentication 的应用。客户端主体应与 HawtIO 在线实例提供的匹配（默认为 hawtio-online.hawtio.svc）。

```
<properties>
  <jolokia.protocol>https</jolokia.protocol>
  <jolokia.host>*</jolokia.host>
  <jolokia.port>8778</jolokia.port>
  <jolokia.useSslClientAuthentication>true</jolokia.useSslClientAuthentication>
  <jolokia.caCert>/var/run/secrets/kubernetes.io/serviceaccount/service-
ca.crt</jolokia.caCert>
  <jolokia.clientPrincipal.1>cn=hawtio-online.hawtio.svc</jolokia.clientPrincipal.1>
  <jolokia.extendedClientCheck>true</jolokia.extendedClientCheck>
  <jolokia.discoveryEnabled>false</jolokia.discoveryEnabled>
</properties>
```

3. 在本地运行应用程序：

   a. 使用以下命令在 development 模式下运行：

   ```
   mvn compile quarkus:dev
   ```

   b. 或者构建项目并执行可运行的 JAR：

   ```
   mvn package && java -jar target/quarkus-app/quarkus-run.jar
   ```

4. 在本地使用 Jolokia 代理运行：

   a. 您可以在本地使用 Jolokia JVM 代理运行这个示例，如下所示：

```
java -javaagent:target/quarkus-app/lib/main/org.jolokia.jolokia-agent-jvm-2.0.1-javaagent.jar -jar target/quarkus-app/quarkus-run.jar
```

5. 将其部署到 OpenShift:

a. 要将其部署到集群中，首先更改 pom.xml 中的容器镜像参数，以适合开发环境。（默认镜像名称为 quay.io/hawtio/hawtio-online-example-camel-quarkus:latest，它应该被推送到 Quay.io 上的 hawtio 组织。）

```
<!--
  Container registry and group should be changed to those which your application uses.
-->
<quarkus.container-image.registry>quay.io</quarkus.container-image.registry>
<quarkus.container-image.group>hawtio</quarkus.container-image.group>
<quarkus.container-image.tag>latest</quarkus.container-image.tag>
```

b. 然后，使用选项 -Dquarkus.container-image.push=true 来构建项目，以将构建镜像推送到首选容器 registry：

```
mvn install -Dquarkus.container-image.push=true
```

c. 部署的资源文件也会在 target/kubernetes/kubernetes.yml 上生成。使用 kubectl 或 oc 命令，使用 resources 文件部署应用程序：

```
kubectl apply -f target/kubernetes/kubernetes.yml
```

d. 部署成功后，pod 已启动，可以在集群中看到应用程序日志。

第 7 章 查看容器和应用程序

当您登录到 OpenShift 的 HawtIO 时，A HawtIO 主页会显示可用的容器。

流程：

1. 若要管理（创建、编辑或删除）容器，请使用 OpenShift 控制台。

2. 要查看 OpenShift 集群上启用了 HawtIO 的应用程序和 AMQ Broker （如果适用），请点击 Online 选项卡

第 8 章 查看并管理 APACHE CAMEL 应用程序

在 HawtIO 的 Camel 选项卡中，您可以查看和管理 Apache Camel 上下文、路由和依赖项。

您可以查看以下详情：

1. 所有正在运行的 Camel 上下文列表
2. 每个 Camel 上下文（如 Camel 版本号和运行时静态）的详细信息
3. 每个 Camel 应用程序及其运行时统计中的所有路由列表
4. 正在运行的路由的图形表示以及实时指标

您还可以通过以下方法与 Camel 应用程序交互：

1. 启动和暂停上下文
2. 管理所有 Camel 应用程序及其路由的生命周期，以便您可以重启、停止、暂停、恢复等。
3. 运行路由的实时追踪和调试
4. 浏览消息并将其发送到 Camel 端点

注意

只有在连接到使用一个或多个 Camel 路由的容器时，Camel 选项卡才可用。

8.1. 启动、暂停或删除上下文

1. 在 Camel 选项卡的树视图中，单击 Camel Contexts。

2. 选中列表中一个或多个上下文旁边的框。

3. 点 Start 或 Suspend。

4. 删除上下文：

   a. 停止上下文。

   b. 单击 ellipse 图标，然后从下拉菜单中选择 Delete。

注意

当您删除上下文时，您可以从部署的应用程序中删除它。

8.2. 查看 CAMEL 应用程序详情

1. 在 Camel 选项卡的树视图中，点 Camel 应用程序。

2. 要查看应用属性和值的列表，请单击 Attributes。

3. 要查看应用程序属性的图形表示，请点 Chart，然后点 Edit 以选择要在图表中看到的属性。

4. 要查看动态和被阻止交换，请点击 Exchanges。

5. 要查看应用端点，请单击 Endpoints。您可以根据 URL、Route ID 和 方向 来过滤列表。

6. 要查看、启用和禁用与 Camel 内置类型转换机制相关的统计信息，用于将消息正文和消息标头转换为不同的类型，请单击 Type Converters。

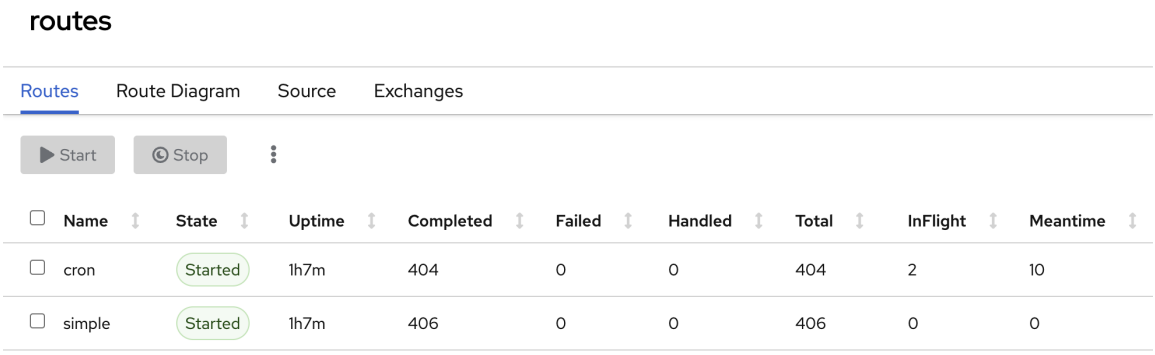
7. 要查看和执行 JMX 操作，如从 XML 添加或更新路由或查找 classpath 中的所有 Camel 组件，请点 Operations。

8.3. 查看 CAMEL 路由列表并与之交互

1. 查看路由列表 ：

   a. 点 Camel 选项卡。

   b. 在树视图中，点应用程序的路由文件夹：

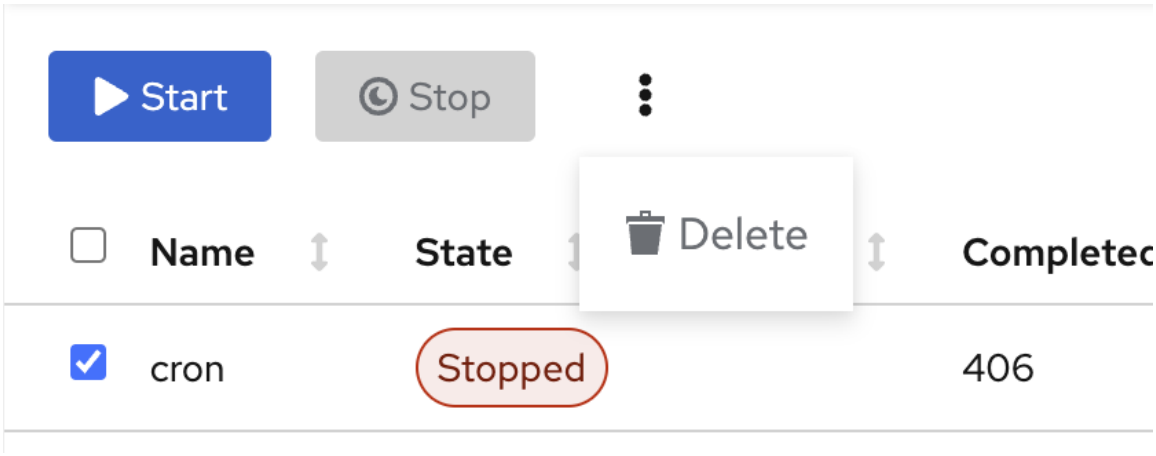


2. 启动、停止或删除一个或多个路由 ：

   a. 选中列表中一个或多个路由旁边的框。

   b. 点 Start 或 Stop。

   c. 要删除路由，您必须首先停止它。然后，单击 ellipse 图标，然后从下拉菜单中选择 删除。



注意

- *删除路由时，您可以将其从部署的应用程序中删除。*
- *您还可以在树视图中选择特定的路由，然后单击右上角的菜单来启动、停止或删除它。*

3. *要查看路由的图形图，请点 Route 图表。*

4. *要查看动态和被阻止交换，请点击 Exchanges。*

5. *要查看端点，请点 Endpoints。您可以根据 URL、Route ID 和方向过滤列表。*

6. *点 Type Converters 查看、启用和禁用与 Camel 内置类型转换机制相关的统计信息，该机制用于将消息正文和消息标头转换为不同的类型。*

7. *与特定路由交互 ：*

   a. *在 Camel 选项卡的树视图中，选择一个路由。要查看路由属性和值的列表，请单击 Attributes。*

   b. *要查看路由属性的图形表示，请点 Chart。您可以点 Edit 来选择要在图表中看到的属性。*

c.
要查看动态和被阻止交换，请点击 **Exchanges**。

d.
点 **Operations** 查看并在路由上执行 **JMX** 操作，如将路由转储为 **XML** 或获取路由的 **Camel ID** 值。

8.
通过路由跟踪消息 ：

a.
在 **Camel** 选项卡的树视图中，选择一个路由。

b.
选择 **Trace**，然后点 **Start tracing**。

9.
将消息发送到路由 ：

a.
在 **Camel** 选项卡的树视图中，打开上下文的端点文件夹，然后选择端点。

b.
单击 **Send** 子选项卡。

c.
以 **JSON** 或 **XML** 格式配置消息。

d.
单击 **Send**。

e.
返回到路由的 **Trace** 选项卡，以查看消息通过路由流。

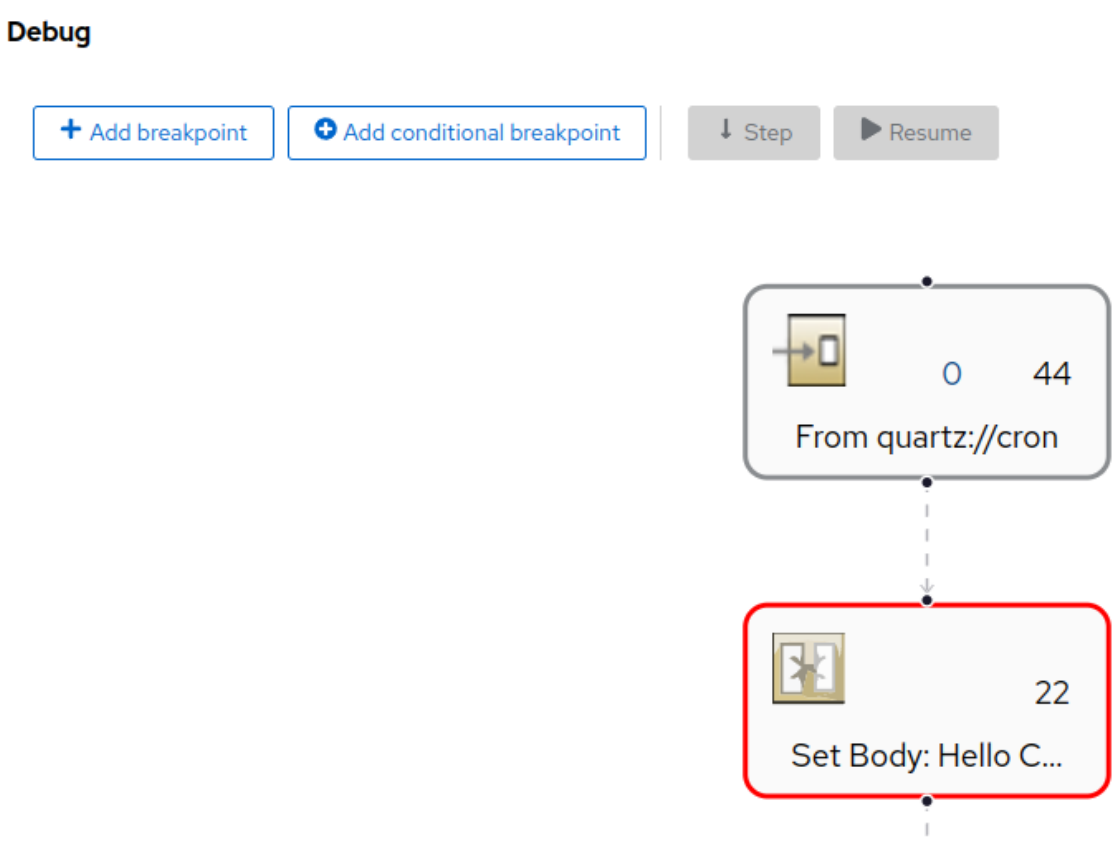
8.4. 调试路由

1.
在 **Camel** 选项卡的树视图中，选择一个路由。

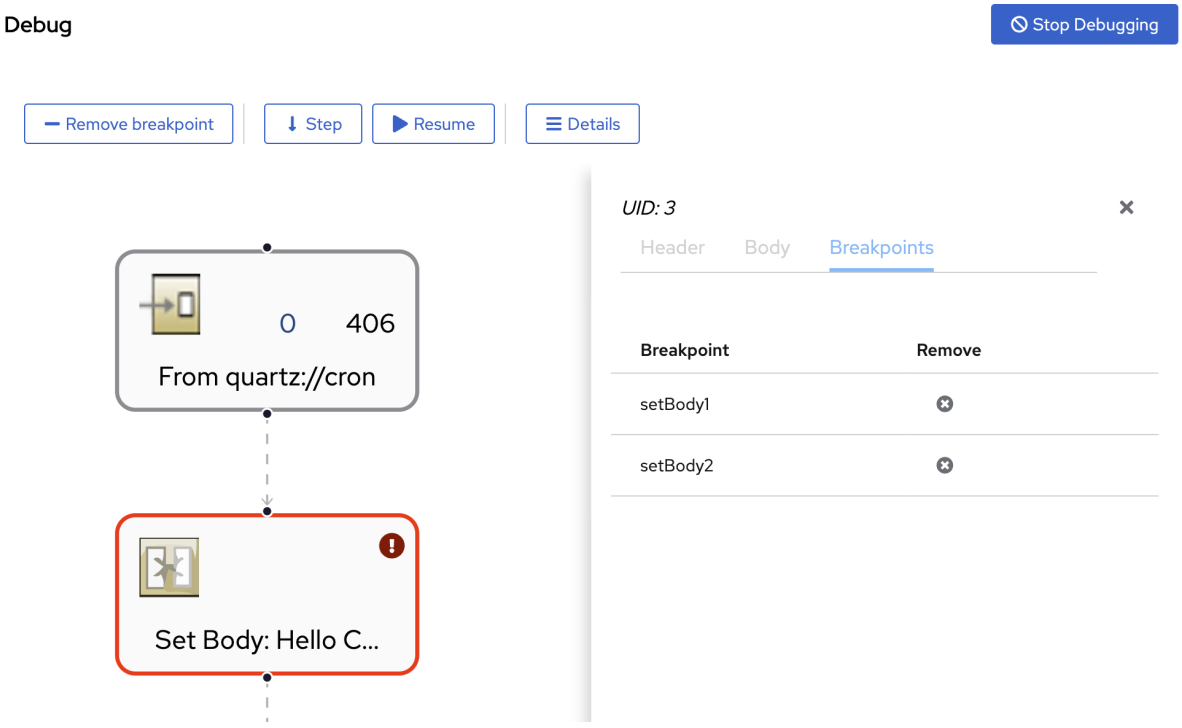
2.
选择 **Debug**，然后单击 **Start debugging**。

3.
要添加断点，请在图中选择一个节点，然后单击 **Add breakpoint**。节点中会出现一个红色的

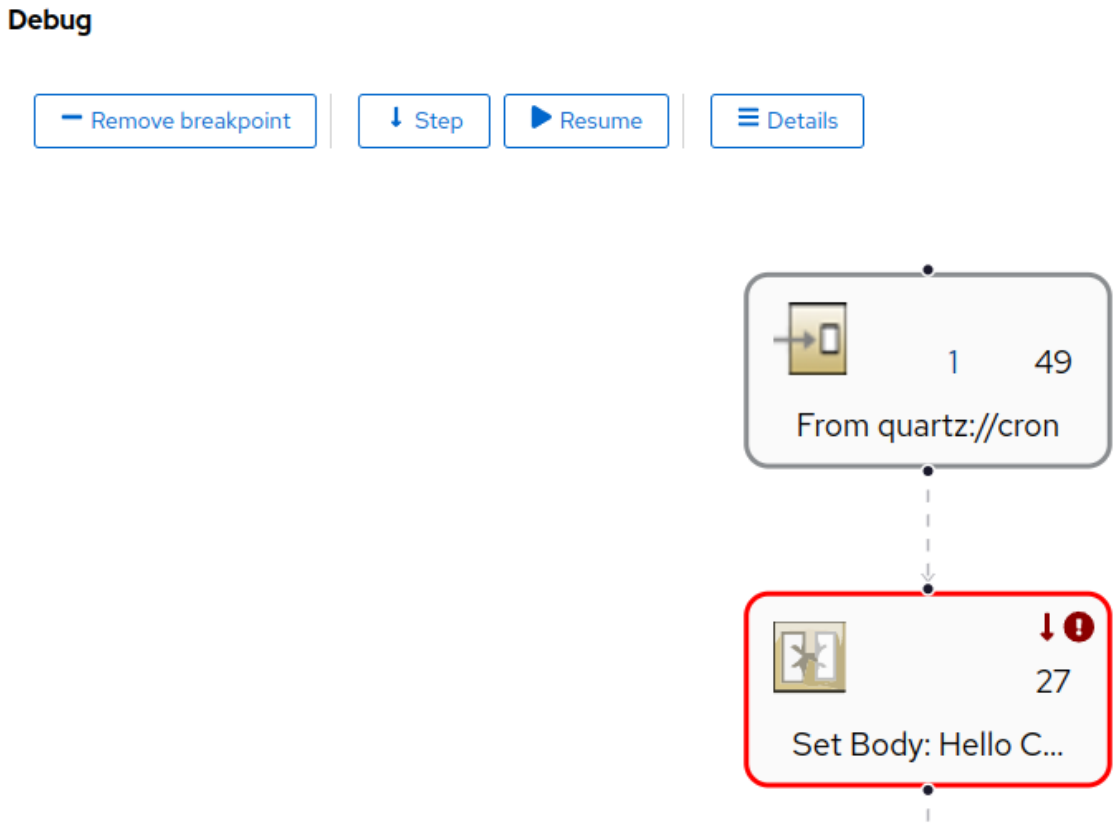
点：



4. *节点添加到断点列表中：*



5. *点下箭头进入下一个节点，或 Resume 按钮恢复运行路由。*



6. *单击 暂停 按钮，以暂停路由的所有线程。*

7. *完成后点 Stop debug。所有断点都被清除。*

第 9 章 查看并管理 JMX 域和 MBEANS

Java 管理扩展(JMX)是一种 Java 技术，允许您在运行时动态管理资源（服务、设备和应用程序）。资源由名为 MBeans （用于 Managed Bean）的对象表示。您可以在资源创建、实施或安装后立即管理和监控资源。

借助 HawtIO 上的 JMX 插件，您可以查看和管理 JMX 域和 MBeans。您可以查看 MBean 属性、运行命令和创建显示 MBeans 统计信息的图表。

JMX 选项卡提供活动 JMX 域和以文件夹组织的 MBeans 的树状视图。您可以在 MBeans 上查看详情和执行命令。

流程 ：

1. 查看并编辑 MBean 属性 ：
   a. 在树视图中，选择一个 MBean。
   b. 单击 Attributes 选项卡。
   c. 点一个属性查看其详情。
2. 执行操作 ：
   a. 在树视图中，选择一个 MBean。
   b. 单击 Operations 选项卡，展开列出的操作之一。
   c. 单击 Execute 以运行该操作。
3. 查看图表 ：

a.

在树视图中，选择一个项目。

b.

点 Chart 选项卡。

第 10 章 查看并管理 QUARTZ SCHEDULES

quartz 是一个功能丰富的开源作业调度库，您可以在大多数 Java 应用程序内集成。您可以使用 Quartz 为执行作业创建简单或复杂的计划。

作业被定义为一个标准 Java 组件，您可以几乎执行该组件来对其进行编程。

如果您的 Camel 路由部署了 camel-quartz 组件，则 HawtIO 显示 Quartz 选项卡。请注意，您还可以通过 JMX 树视图访问 Quartz mbeans。

流程 ：

1. 在 HawtIO 中，单击 Quartz 选项卡。Quartz 页面包含 Quartz 调度程序和调度程序、Triggers 和 Jobs 选项卡的树视图。

2. 要暂停或启动调度程序，请单击 调度程序选项卡上的按钮。

3. 点 Triggers 选项卡查看决定作业何时运行的触发器。例如，触发器可以指定在一天（到 millisecond）、在指定天数或指定次数或特定时间重复启动作业。

    a. 要过滤触发器列表，请从下拉列表中选择 State、Group、Name 或 Type。然后，您可以通过选择或键入 fill-on 字段来进一步过滤列表。

    b. 要暂停、恢复、更新或手动触发触发器，请单击 Action 列中的选项。

4. 点 Jobs 选项卡查看正在运行的作业列表。您可以根据表中的列对列表进行排序：Group、Name、Durable、Recover、Job ClassName 和 Description。

第 11 章 查看线程

您可以查看和监控线程状态。

流程 ：

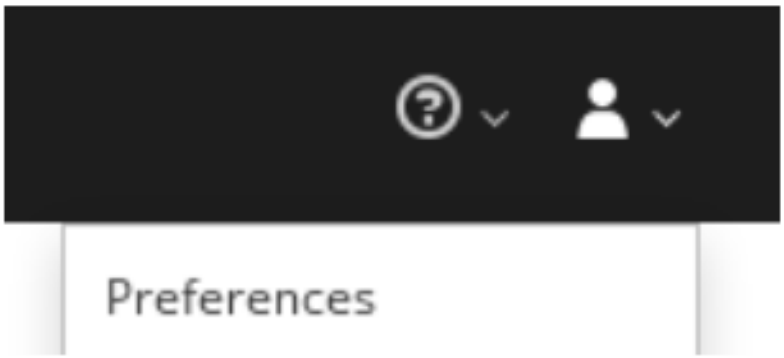
1. 单击 Runtime 选项卡，然后单击 Threads 子选项卡。
2. Threads 页面列出了每个线程的活跃线程和堆栈追踪详情。默认情况下，线程列表以降序 ID 顺序显示所有线程。
3. 要通过增加 ID 进行排序，请点 ID 列标签。
4. （可选）根据线程状态（如 Blocked）或线程名称过滤列表。
5. 要深入查看特定线程的详细信息，如锁定类名称和该线程的完整堆栈追踪，请在 Actions 列中点击 More。

第 12 章 确保在 HAWTIO 中显示正确的数据

如果 HawtIO 中的队列和连接显示缺少队列、缺少连接或显示不一致的图标，请调整 Jolokia 集合大小参数，用于指定响应中 Jolokia marshals 中的最大元素数。

流程：

1. 在 HawtIO 右上角，单击用户图标，然后单击 Preferences。



2. 增加 Maximum collection size 选项的值（默认值为 50,000）。

3. 单击 Close。

第 13 章 OPENID CONNECT 集成

HawtIO 已经支持 Keycloak 作为 OpenID 提供程序。但是 Keycloak 已宣布 HawtIO 使用的配置方法已弃用。因为 OpenID Connect Core 1.0 是分布式身份验证（基于 OAuth 2）的广泛规格和标准方法，因此 HawtIO 4 现在支持通用 OpenID 身份验证。

13.1. 构建块和术语

要了解 HawtIO 如何使用 OpenID Connect 和 OAuth2，需要记住一些基本概念。根据 OpenID Connect （基于 OAuth2构建）分布式身份验证中涉及 3 个主要方：

1. 资源服务器 ：

   托管受保护的资源的服务器组件，其中根据访问令牌限制或授予访问权限。通常，此服务器通过 REST API 访问，而不自行提供用户界面。

2. 客户端 ：

   应用程序（通常用于用户界面）代表用户访问资源服务器（被视为资源所有者）。要访问资源服务器，客户端必须首先获取访问令牌。

   在 OpenID Connect 规格中，客户端被命名为 dependent party (RP)。

3. 授权服务器 ：

   协调客户端和资源服务器之间通信的服务器。客户端要求授权服务器验证用户（资源所有者），如果身份验证成功，则会为客户端发出访问令牌来访问资源服务器。

   在 OpenID Connect 规格中，授权服务器名为 OpenID Provider (OP)。

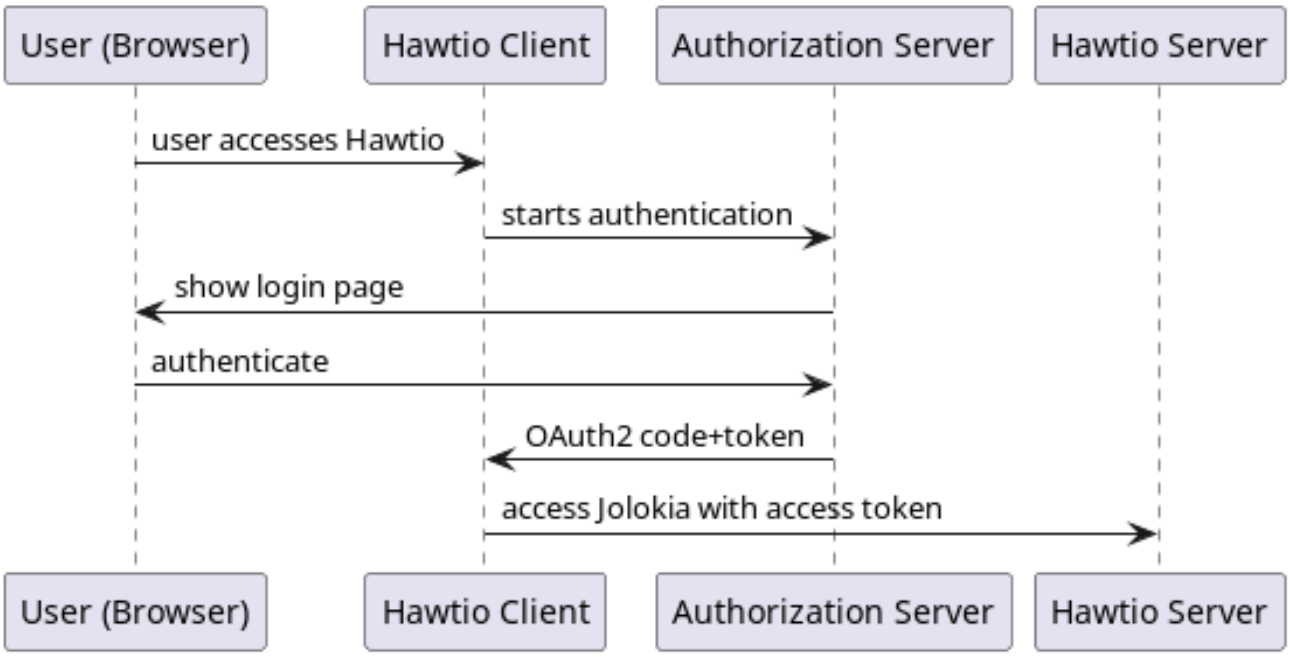
OAuth2 和 OpenID Connect 的主要目标，它允许应用访问 API，而无需使用用户凭据和切换到令牌交换。务必要知道 HawtIO 如何映射到上述角色：

- HawtIO 客户端应用程序是一个 OAuth2 客户端。用户与 Hawtio Web 应用程序交互，后者与运行 Jolokia 代理的 Hawtlo Server (backend)通信。在访问 Jolokia 代理前，HerwtIO 需要

OpenID Connect 访问令牌。为此，HerwtIO 客户端通过将用户重定向到授权服务器来发起 OpenID Connect 身份验证流程。

- *HawtIO 服务器应用是一个 JakartaEE 应用，公开 Jolokia Agent API，它根据访问令牌的内容授权用户操作。使用 OAuth2 术语时，HerwtIO 服务器是一个资源服务器。*

以下 UML 图表显示了巨大的图片。



最重要的方面是：HawtIO 客户端永远不会处理用户凭证。用户通过授权服务器和 HawtIO 客户端仅获取稍后用于访问 Hawtio 服务器的访问令牌（及其 Jolokia API）。

13.2. HAWTIO 中的通用 OPENID CONNECT 身份验证

HawtIO 4 可以与现有 OpenID Connect 供应商（如 Keycloak、Microsoft Entra ID、Auth0、...）一起使用，并使用这些库完整填充任务：

- *Apache HTTP 客户端 4 来实现从 Hawtio 服务器到 OpenID Connect 提供程序的 HTTP 通信（例如，检索有关令牌签名验证的公钥的信息）。*
- *Nimbus JOSE + JWT 库，用于操作和验证 OpenID Connect / OAuth2 访问令牌。*

这些库包含在 HawtIO Server WAR 中，这意味着不需要安装/部署任何其他库（就像 Keycloak 特定的

配置一样）。要使用外部 OpenID Connect 供应商配置 HawtIO，我们需要提供一个配置文件并将 HawtIO 指向其位置。

指定 OIDC (OpenID Connect)配置位置的系统属性是 -Dhawtio.oidcConfig，但如果未指定，则会检查默认位置。默认值为：

- *对于 Karaf 运行时，${karaf.base}/etc/hawtio-oidc.properties*
- *对于 Jetty 运行时，${jetty.home}/etc/hawtio-oidc.properties*
- *对于 Tomcat 运行时，${catalina.home}/conf/hawtio-oidc.properties*
- *对于 JBoss/EAP/Wildfly 运行时，${jboss.server.config.dir}/hawtio-oidc.properties*
- *对于 Apache Artemis 运行时，${artemis.instance.etc}/hawtio-oidc.properties*
- *回退到 classpath:hawtio-oidc.properties （用于嵌入式 Hawtio 用法）*

与 Keycloak 特定配置不同，只有一个带有用来配置 OpenID Connect 配置的所有方面的文件。

以下是模板：

```
# OpenID Connect configuration requred at client side

# URL of OpenID Connect Provider - the URL after which ".well-known/openid-configuration"
can be appended for
# discovery purposes
provider = http://localhost:18080/realms/hawtio-demo
# OpenID client identifier
client_id = hawtio-client
# response mode according to https://openid.net/specs/oauth-v2-multiple-response-types-
1_0.html
response_mode = fragment
# scope to request when performing OpenID authentication. MUST include "openid" and
required permissions
scope = openid email profile
# redirect URI after OpenID authentication - must also be configured at provider side
```

```
redirect_uri = http://localhost:8080/hawtio
# challenge method according to https://datatracker.ietf.org/doc/html/rfc7636
code_challenge_method = S256
# prompt hint according to https://openid.net/specs/openid-connect-core-
1_0.html#AuthRequest
prompt = login

# additional configuration for the server side

# if true, .well-known/openid-configuration will be fetched at server side. This is required
# for proper JWT access token validation
oidc.cacheConfig = true

# time in minutes to cache public keys from jwks_uri
jwks.cacheTime = 60

# a path for an array of roles found in JWT payload. Property placeholders can be used for
parameterized parts
# of the path (like for Keycloak) - but only for properties from this particular file
# example for properly configured Entra ID token
#oidc.rolesPath = roles
# example for Keycloak with use-resource-role-mappings=true
#oidc.rolesPath = resource_access.${client_id}.roles
# example for Keycloak with use-resource-role-mappings=false
oidc.rolesPath = realm_access.roles

# properties for role mapping. Each property with "roleMapping." prefix is used to map an
original role
# from JWT token (found at ${oidc.rolesPath}) to a role used by the application
roleMapping.admin = admin
roleMapping.user = user
roleMapping.viewer = viewer
roleMapping.manager = manager

# timeout for connection establishment (milliseconds)
http.connectionTimeout = 5000
# timeout for reading from established connection (milliseconds)
http.readTimeout = 10000
# HTTP proxy to use when connecting to OpenID Connect provider
#http.proxyURL = http://127.0.0.1:3128

# TLS configuration (system properties can be used, e.g., "${catalina.home}/conf/hawtio.jks")

#ssl.protocol = TLSv1.3
#ssl.truststore = src/test/resources/hawtio.jks
#ssl.truststorePassword = hawtio
#ssl.keystore = src/test/resources/hawtio.jks
#ssl.keystorePassword = hawtio
#ssl.keyAlias = openid connect test provider
#ssl.keyPassword = hawtio
```

此文件配置 HawtIO+OpenID Connect 的几个方面：

- OAuth2 - 配置授权服务器、客户端 ID 和几个 OpenID Connect 相关选项的位置

- JWKS - 从 jwks_uri 获取的公钥的缓存时间，这是公开授权服务器使用的公钥的端点。

- JWT 令牌配置 - 有关声明(JSON Web Token 中的字段)的信息，其中包含与经过身份验证的用户关联的角色。我们还允许将授权服务器中定义的角色映射到应用程序使用的角色(HawtIO 服务器和 Jolokia)。

- HTTP 配置 - 服务器端 HTTP 客户端用来连接到授权服务器（获取 OpenID Connect 元数据和公开公钥）。

这个示例配置可以调整为特定的需求，但在与容器化 Keycloak 一起使用时也可以按原样工作。（请参阅以下）。

13.3. JAAS 角色类配置

OpenID Connect 在 HawtIO 服务器端通过 JAAS 使用。当 HawtIO 客户端获取访问令牌时，会使用 HTTP Authorization: Bearer <access_ token > 标头的每个 Jolokia 请求发送。JWT 令牌中包含的每个角色都是（可能在映射后）作为 JAAS 主题 的角色主体 包含在内。默认情况下（如果没有明确配置），角色主体的类为 io.hawt.web.auth.oidc.RolePrincipal。

但是，可以配置另一个类（要求是 - 它必须包含单一 String 参数构造器）才能用作主体角色类。例如，当与 Apache Artemis 一起使用时，该角色应为 org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal。

有一个指定角色类的系统属性：

```
-Dhawtio.rolePrincipalClasses=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal
```

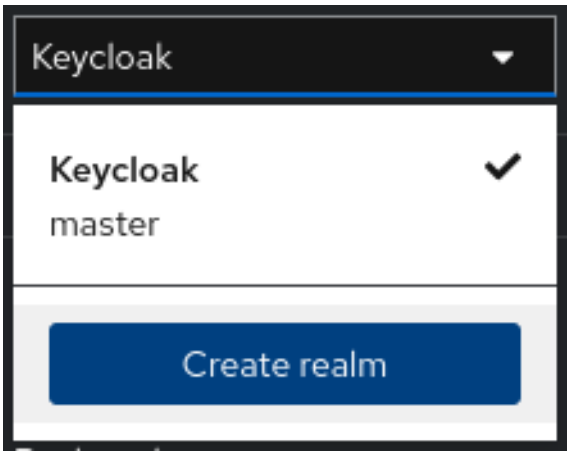
13.4. 在 KEYCLOAK 中使用 HAWTIO 和 OPENID CONNECT 身份验证

运行 Keycloak 实例的最简单方法是使用容器：

```
podman run -d --name keycloak \
```

```
-p 18080:8080 \
-e KEYCLOAK_ADMIN=admin \
-e KEYCLOAK_ADMIN_PASSWORD=admin \
quay.io/keycloak/keycloak:latest start-dev
```

启动后，浏览 http://localhost:18080/admin/master/console/ 并创建新域：

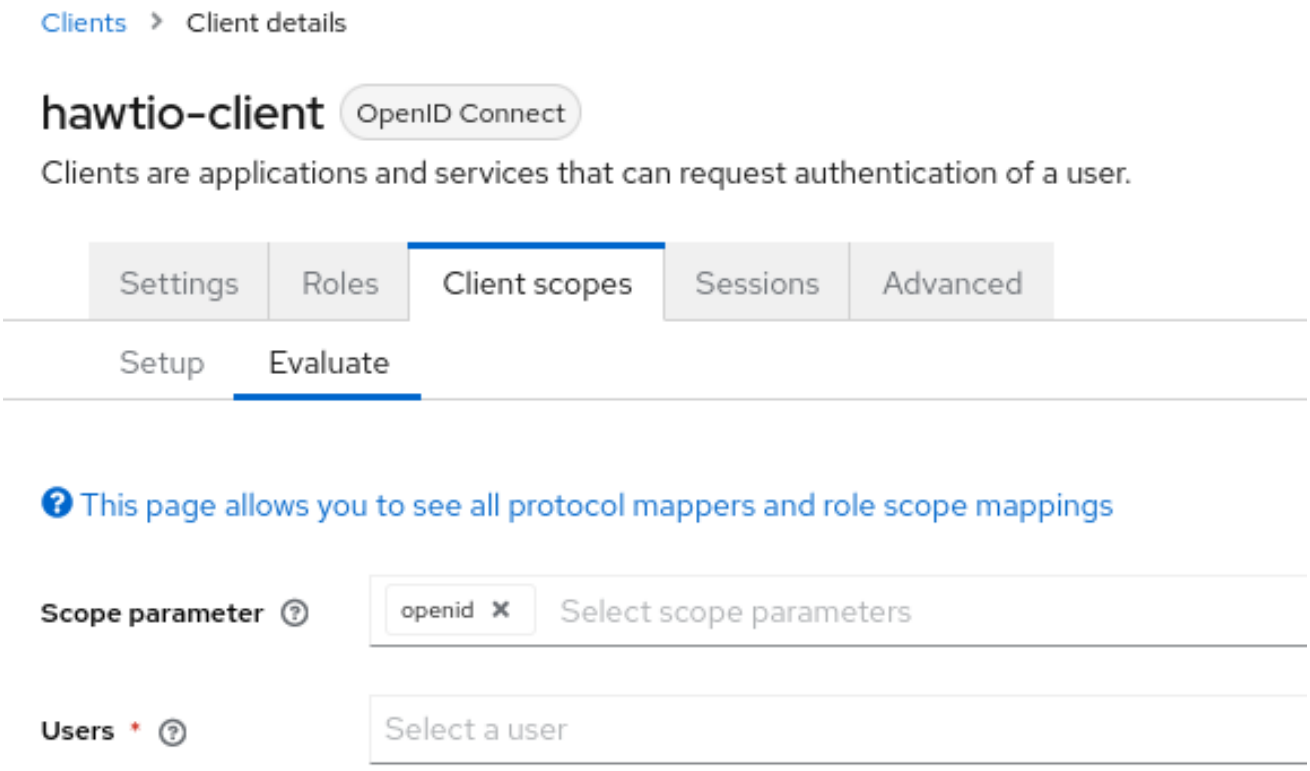


在域创建屏幕中，上传 hawtio-demo-realm.json，它定义了带有预先配置的 hawtio-client 客户端和 3 用户的新的 hawtio-demo 域：

1. 带有角色 管理器、管理员、查看器 和 用户的admin /admin
2. 带有角色查看器和 用户的 viewer /viewer
3. Jdoe/jdoe 带有 用户角色

13.4.1. 检查 JWT 令牌问题

要检查已授予访问令牌的内容，我们可以使用 Keycloak 接口。导航到"客户端"，选择"hawtio-client"，并使用"客户端范围"选项卡和"Evaluate"子选项卡：



然后，在"Users"字段中，如 "admin"，然后单击"Generated access token"。然后，我们可以检查示例令牌：

```
{
  "exp": 1709552728,
  "iat": 1709552428,
  "jti": "0f33971f-c4f7-4a5c-a240-c18ba3f97aa1",
  "iss": "http://localhost:18080/realms/hawtio-demo",
  "aud": "account",
  "sub": "84d156fa-e4cc-4785-91c1-4e0bda4b8ed9",
  "typ": "Bearer",
  "azp": "hawtio-client",
  "session_state": "181a30ac-fce1-4f4f-aaee-110304ccb0e6",
  "acr": "1",
  "allowed-origins":
  [
    "http://0.0.0.0:8181",
    "http://localhost:8080",
    "http://localhost:8181",
    "http://0.0.0.0:10001",
    "http://0.0.0.0:8080",
    "http://localhost:10001",
    "http://localhost:10000",
    "http://0.0.0.0:10000"
  ],
  "realm_access":
  {
    "roles":
    [
      "viewer",
```

```
      "manager",
      "admin",
      "user"
    ]
  },
  "resource_access":
  {
    "account":
    {
      "roles":
      [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "openid profile email",
  "sid": "181a30ac-fce1-4f4f-aaee-110304ccb0e6",
  "email_verified": false,
  "name": "Admin Hawtio",
  "preferred_username": "admin",
  "given_name": "Admin",
  "family_name": "Hawtio",
  "email": "admin@hawt.io"
}
```

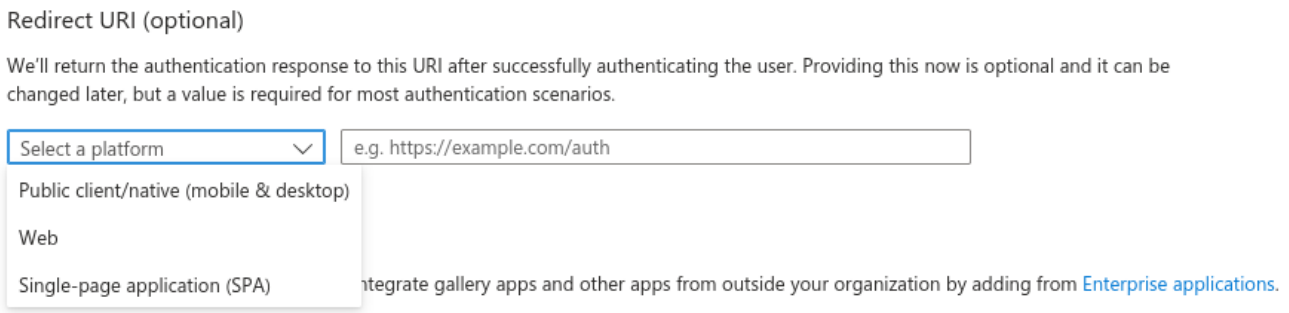
了解 JWT 访问令牌的结构，我们可以检查角色路径是否正确：

```
# example for Keycloak with use-resource-role-mappings=false
oidc.rolesPath = realm_access.roles
```

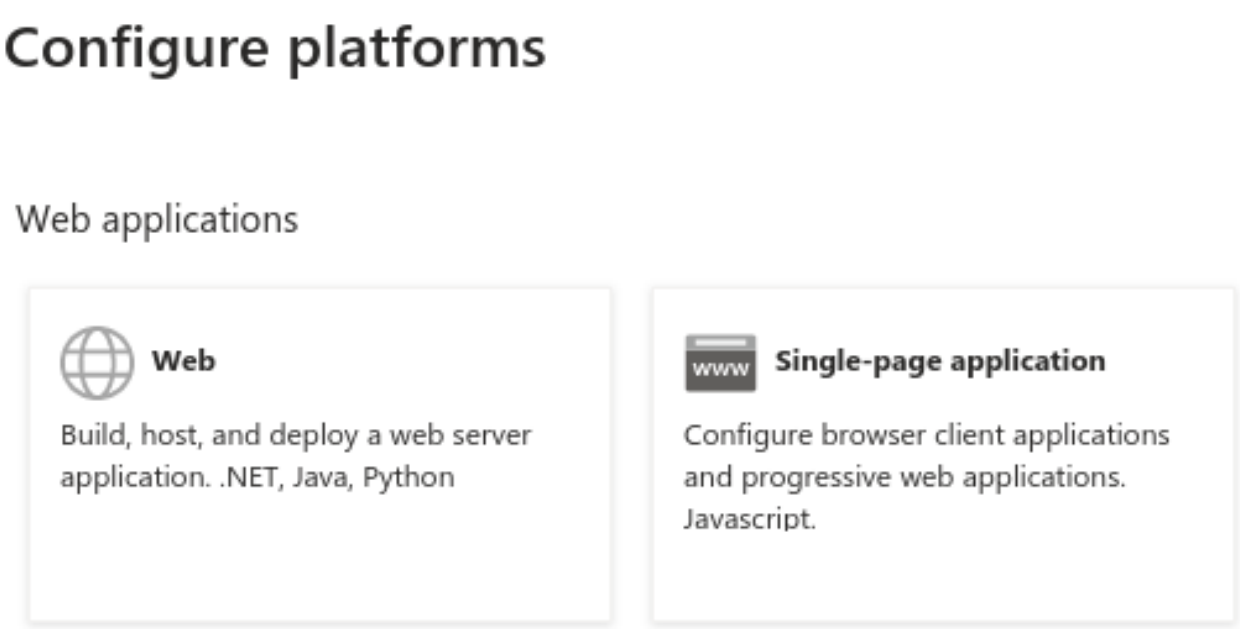
13.5. 在 MICROSOFT ENTRA ID 中使用 HAWTIO 和 OPENID CONNECT 身份验证

Hawtio 4 已使用 Microsoft Entra ID 进行了测试。在理论上，应该使用任何 OpenID Connect 提供程序的所有操作都可以访问相关的 OpenID 提供程序元数据，在实践中，我们需要一些特定于提供程序的配置。

客户端使用 "应用注册"刀片在 Entra ID 中注册。在注册应用程序时，最重要的决定是重定向 URI 的平台类型：



可以选择 2 个选项（我们不考虑"公共客户端/原生（移动和桌面）平台）。稍后配置 Redirect URI 时会显示此 UI：



虽然在第一眼中并不明显选择什么，但状态就足够了：

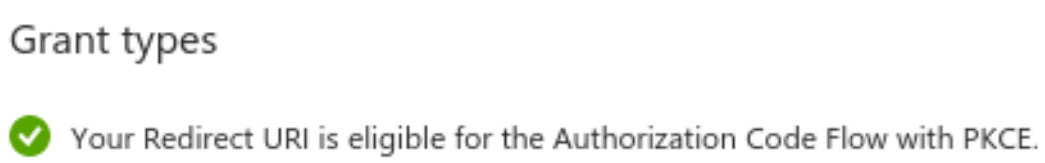
1. ***Web 平台 ：***

   这种类型的客户端适合服务器端应用程序和 API。

2. ***SPA 平台 ：***

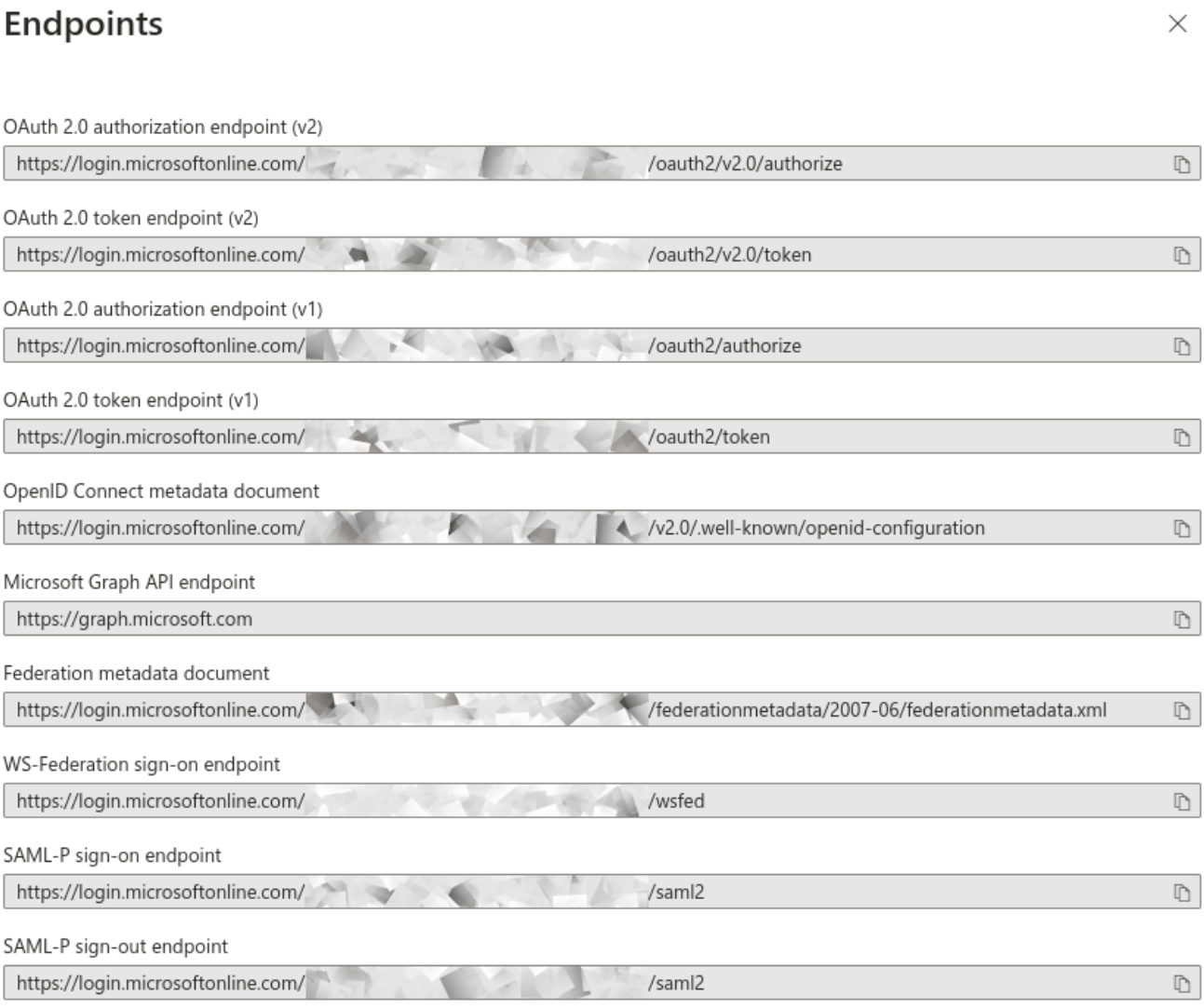
   ***SPA** 应用程序在浏览器中运行，它自然使用"Authorization Code Flow"，因此调用的公共客户端。其原因是，在浏览器应用中存储凭据和 secret 没有良好的方法。*

*选择 **SPA** 平台可在 **Entra ID UI** 中提供此标记：*



13.5.1. 在 Entra ID 中使用单个 SPA 客户端

在 Entra ID 中配置 SPA 客户端后，我们可以已在 hawtio-oidc.properties 中设置相关选项。在 Entra ID 中的"应用注册"刀片式上，我们可以单击"Endpoints"选项卡并展示：



租户 ID 是特定于所使用的 Entra ID / Azure 租户的 UUID。以下是 HawtIO 配置，其中 provider 是您的租户的基本 URL，client_id 是 App Registration 页面的"应用程序（客户端）ID"。

```
# OpenID Connect configuration requred at client side

# URL of OpenID Connect Provider - the URL after which ".well-known/openid-configuration"
can be appended for
# discovery purposes
provider = https://login.microsoftonline.com/00000000-1111-2222-3333-444444444444/v2.0
```

```
# OpenID client identifier
client_id = 55555555-6666-7777-8888-999999999999
# response mode according to https://openid.net/specs/oauth-v2-multiple-response-types-1_0.html
response_mode = fragment
# scope to request when performing OpenID authentication. MUST include "openid" and required permissions
scope = openid email profile
# redirect URI after OpenID authentication - must also be configured at provider side
redirect_uri = http://localhost:8080/hawtio
# challenge method according to https://datatracker.ietf.org/doc/html/rfc7636
code_challenge_method = S256
# prompt hint according to https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest
prompt = login
```

这种配置的问题（其中 openid email profile 发送为 scope 参数），即假定的范围在事实 email openid profile User 中。读取和授予的访问令牌是 （仅显示相关的 JWT 声明）：

```
{
  "aud": "00000003-0000-0000-c000-000000000000",
  "iss": "https://sts.windows.net/8fd8ed3d-c739-410f-83ab-ac2228fa6bbf/",
...
  "app_displayname": "hawtio",
...
  "scp": "email openid profile User.Read",
...
}
```

aud (audience)声明是 00000003-0000-0000-c000-000000000000，它是 … Microsoft Graph API 的 OAuth2 客户端 ID。

不仅此类访问令牌不应由 HawtIO 服务器（使用 Jolokia 代理）使用，使用与 Microsoft Graph API 关联的密钥创建签名。

为了正确配置 Entra ID 并确保生成的访问令牌可由 HawtIO 服务器使用，我们需要 两个应用程序注册 - 用于 HawtIO 客户端和 HawtIO 服务器。请参见以下子章节。

13.5.2. 在 Entra ID 中使用 SPA 和 Web 客户端

建议您在 Entra ID 中设置两个应用程序注册：

- 用于 Hawtio 客户端应用程序的 SPA 客户端 - 这是配置启用了 PKCE 的 OAuth2 公共客户端 的方法。

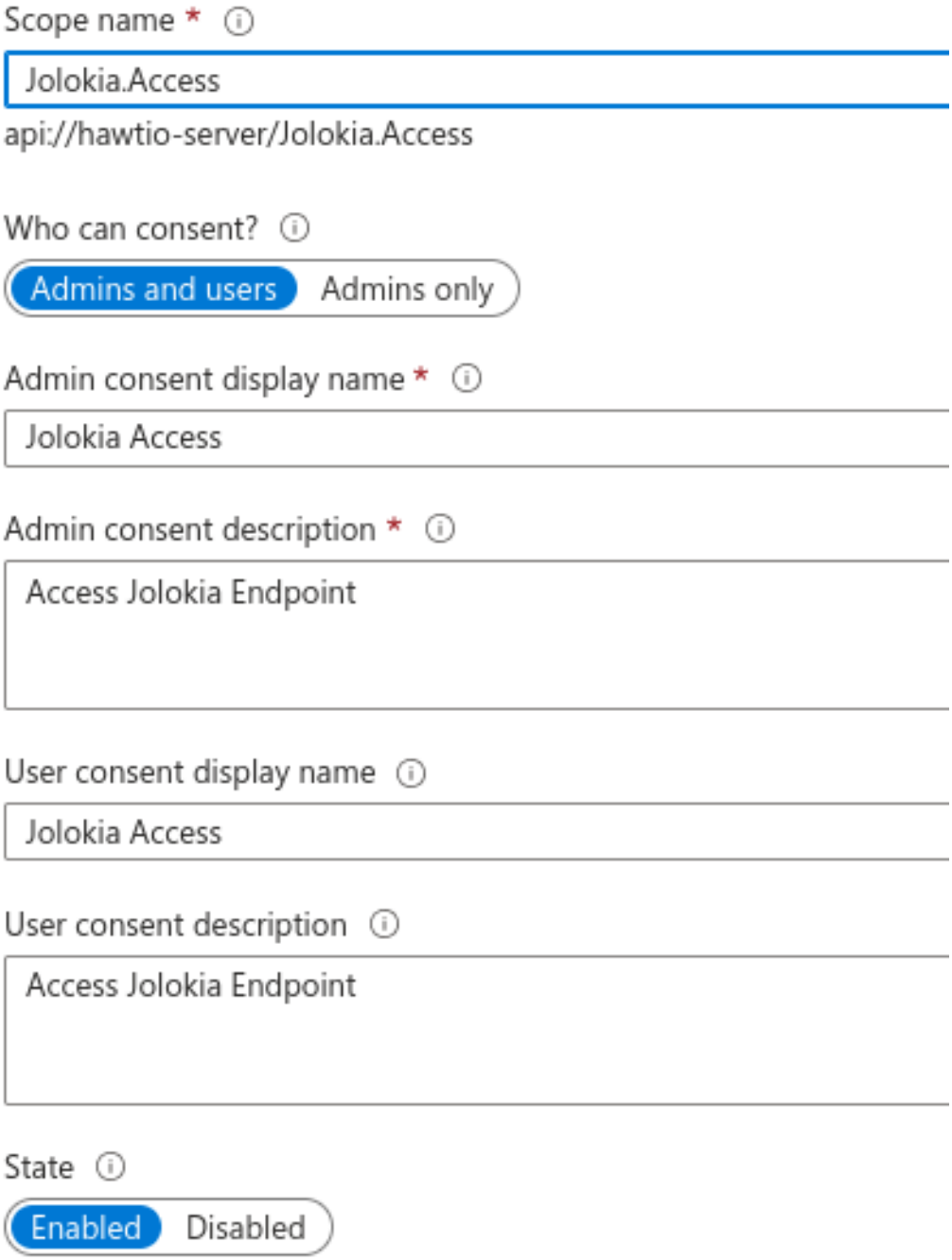
- *用于 Hawtio 服务器应用程序的 Web (API)客户端（实际上，其 Jolokia API）- 是 Entra ID，它公开一个代表的 API，它代表范围（例如） api://hawtio-server/Jolokia.Access，然后在上述 Hawtio 客户端应用程序配置为允许的 API。*

最后，当启动 Authorization Code 流 时，scope 参数中请求的范围是 HawtIO 服务器应用程序（如 api://hawtio-server/Jolokia.Access）定义的范围。

让我们总结 Entra ID 所需的配置。

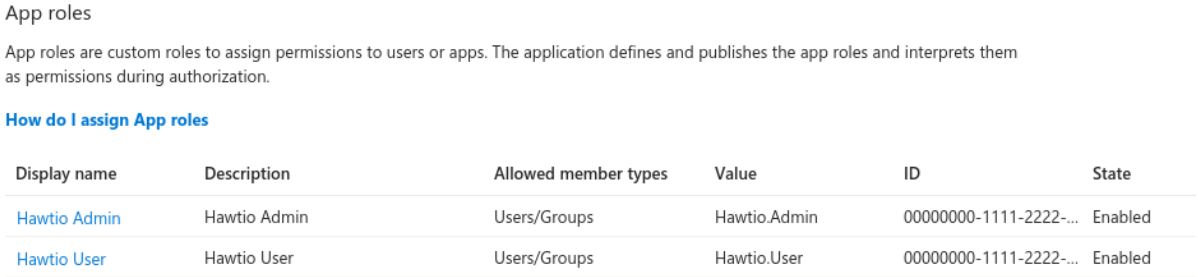
1. *使用"Web"重定向 URI 创建 hawtio-server 应用注册。*

2. *在 "Expose an API" 部分中，添加一个范围，代表可以从 Hawtio 客户端请求的访问范围：*

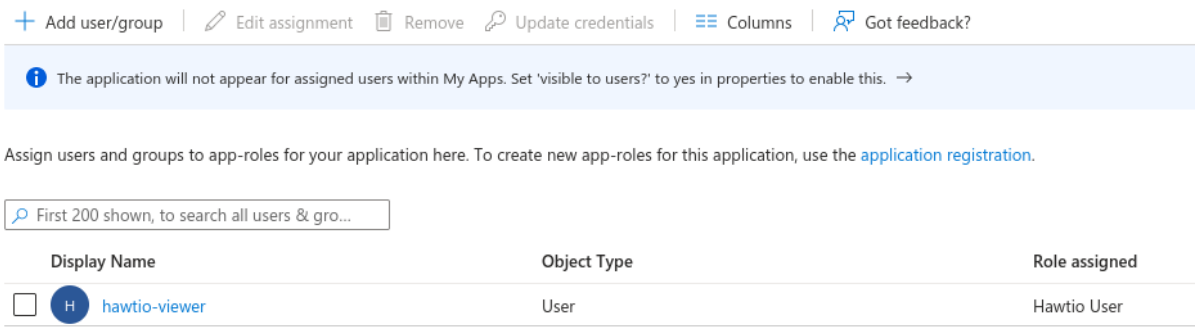


这将创建一个参考的 api://hawtio-server/Jolokia.Access 范围。

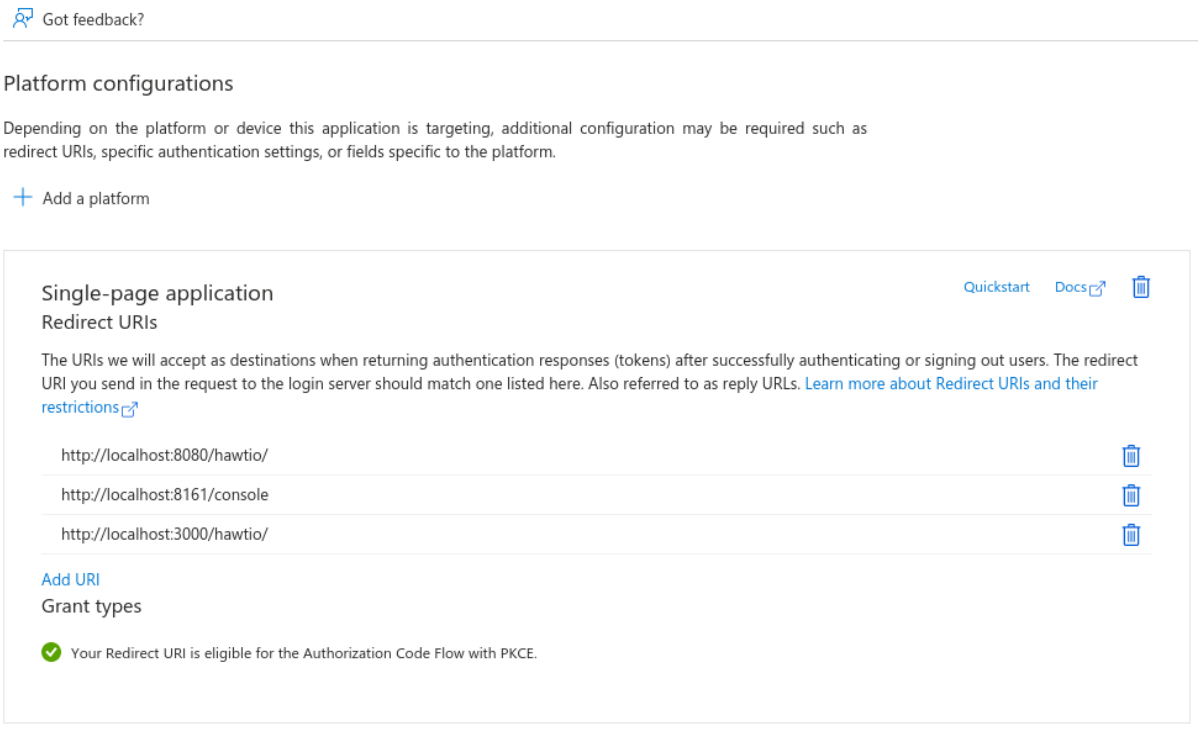
3. *在 hawtio-server 的"App roles"部分中定义您要分配给此客户端范围内的用户的任何角色，例如：*



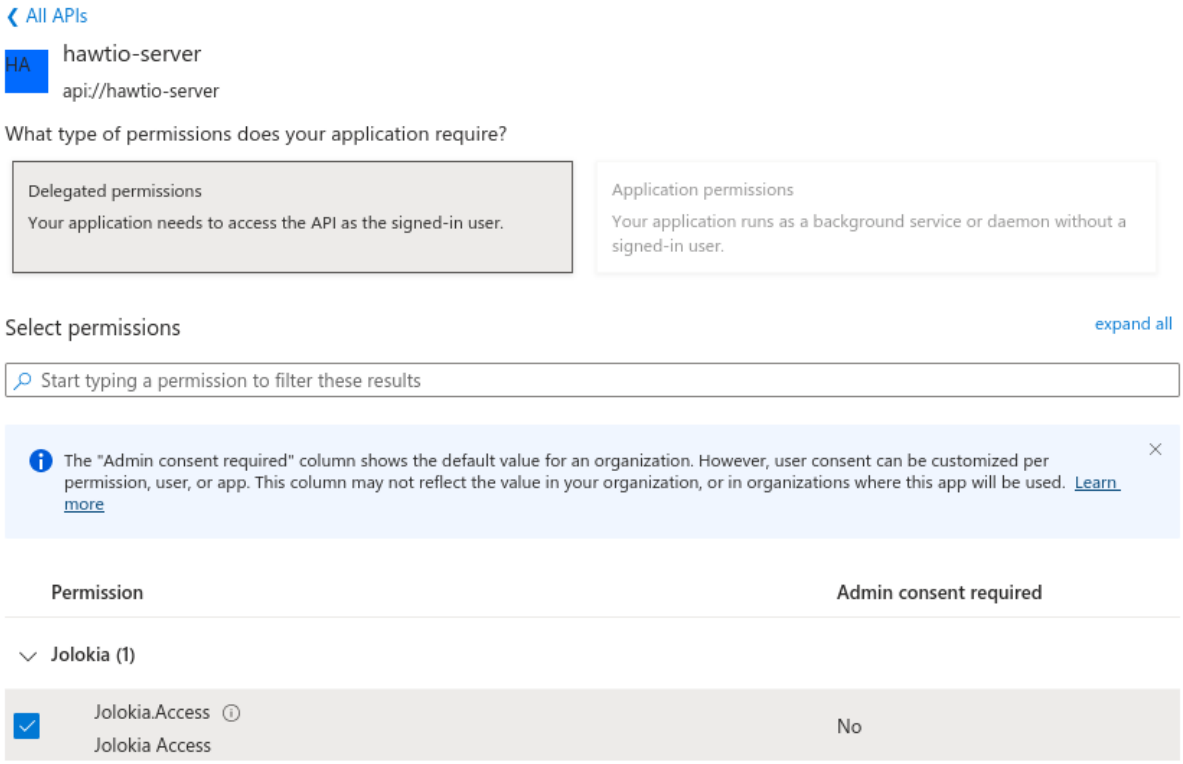
4. ***在"企业级应用程序"刀片中，适用于 hawtio-server 进入"用户和组"选项卡，并添加 user-role 分配。例如：***



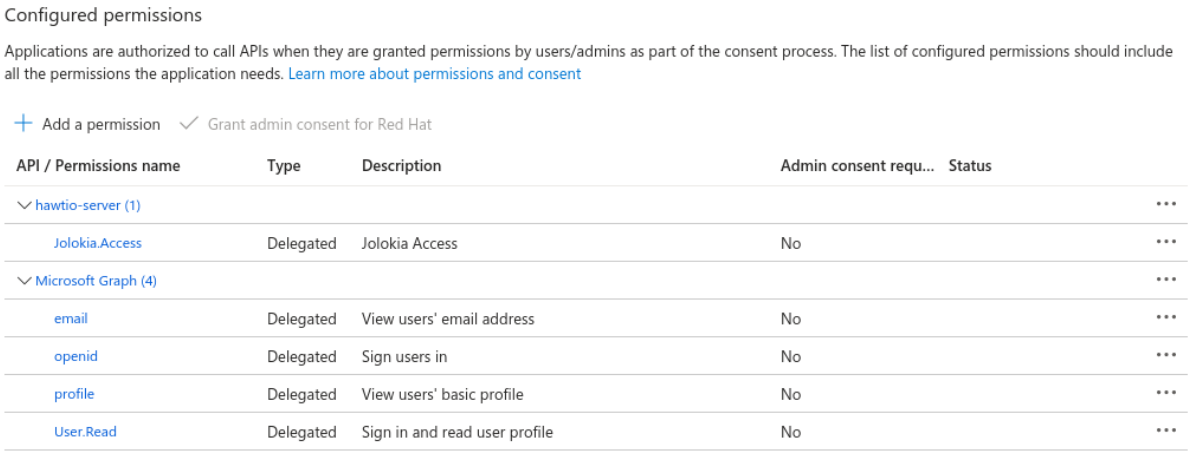
5. ***使用"SPA"重定向 URI 创建 hawtio-client 应用程序注册。***



6. ***在 hawtio-client 应用程序注册的 "API Permissions" 部分中，为 hawtio-server 公开 API 添加 委托权限 ：***



7. *这应该配置一组类似以下的委托权限：*



注意

阅读 Microsoft Entra ID 文档中的 委派权限的更多信息。

8. *企业级应用程序刀片中的 hawtio-client 不需要 User-Role 映射。*

9. *配置上述后，我们可以在 HawtIO 配置中正确设置 scope 参数：*

这将创建一个参考的 api://hawtio-server/Jolokia.Access 范围。

10. 在 hawtio-server 的"App roles"部分中定义您要分配给此客户端范围内的用户的任何角色，例如：

11. 在"企业级应用程序"刀片中，适用于 hawtio-server 进入"用户和组"选项卡，并添加 user-role 分配。例如：

12. 使用 "SPA" Redirect URI 创建 hawtio-client 应用程序注册

13. 在 hawtio-client 应用程序注册的 "API Permissions" 部分中，为 hawtio-server 公开 API 添加委托权限：

这应该配置一组类似以下的委托权限：

注意

阅读 Microsoft Entra ID 文档中的 委派权限的更多信息。

14. 企业级应用程序刀片中的 hawtio-client 不需要 User-Role 映射。

配置上述后，我们可以在 Hawtio 配置中正确设置 scope 参数：

```
# scope to request when performing OpenID authentication. MUST include "openid" and required permissions
scope = openid email profile api://hawtio-server/Jolokia.Access
```

13.5.3. 访问令牌配置

最后，但重要的配置项目是 Token Configuration。对于 hawtio-server 应用程序注册，这是代表 Hawtio 服务器（以及消耗了访问令牌的组件）的应用程序，以确保将组声明添加到访问令牌中。

以下是最小配置：

Optional claims

Optional claims are used to configure additional information which is returned in one or more tokens. Learn more

+ Add optional claim + Add groups claim

Claim	Description	Token type	Optional settings
aud	Used to perform audience validation; emits the client ID of the resource (API) in GUID format	Access	Yes
groups	Optional formatting for group claims	ID, Access, SAML	Yes

组 声明需要包括 安全组和 目录角色和 组，需要按名称表示，而不是 UUID：

Edit groups claim

Adding the groups claim applies to Access, ID, and SAML token types. Learn more

Select group types to include in Access, ID, and SAML tokens.

- [x] Security groups
- [x] Directory roles
- [ ] All groups (includes 3 group types: security groups, directory roles, and distribution lists)
- [ ] Groups assigned to the application (recommended for large enterprise companies to avoid exceeding the limit on the number of groups a token can emit)

Customize token properties by type

ID

Access

- ( ) Group ID
- (•) sAMAccountName
- ( ) NetBIOSDomain\sAMAccountName
- ( ) DNSDomain\sAMAccountName
- ( ) On Premises Group Security Identifier
- [ ] Emit groups as role claims

SAML

为便于参考，以下相关 JSON 片段为 hawtio-server 应用程序注册清单：

```
"optionalClaims":
{
  "idToken":
  [
    {
      "name": "groups",
      "source": null,
      "essential": false,
      "additionalProperties": []
    }
  ],
  "accessToken":
  [
    {
      "name": "groups",
      "source": null,
      "essential": false,
      "additionalProperties":
      [
        "sam_account_name"
      ]
    },
...
```

现在，允许的访问令牌不再特定于 Microsft Graph API audience。它适用于 hawtio-server - aud claim 是 hawtio-server 应用注册的 UUID，appid claim 是 hawtio-client 应用程序注册的 UUID：

```
{
  "aud": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
  "iss": "https://sts.windows.net/.../",
  "iat": 1709626257,
  "nbf": 1709626257,
  "exp": 1709630939,
...
  "appid": "55555555-6666-7777-8888-999999999999",
...
  "groups":
  [
    ...
  ],
...
  "name": "hawtio-viewer",
...
  "roles":
  [
    "Hawtio.User"
  ],
  "scp": "Jolokia.Access",
```

然后转换的角色（可能带有映射）位于 角色 声明中，这反映在配置中：

```
# a path for an array of roles found in JWT payload. Property placeholders can be used for
parameterized parts
# of the path (like for Keycloak) - but only for properties from this particular file
# example for properly configured Entra ID token
#oidc.rolesPath = roles
...
# properties for role mapping. Each property with "roleMapping." prefix is used to map an
original role
# from JWT token (found at ${oidc.rolesPath}) to a role used by the application
roleMapping.Hawtio.User = user
...
```