Red Hat Decision Manager 7.13

管理红帽决策管理器和 KIE 服务器设置

Last Updated: 2023-10-27

法律通告

摘要

本文档论述了如何修改红帽决策管理器和 KIE 服务器设置以及属性以满足您的业务需求。

目录

前言

作为开发人员或系统管理员，您可以修改红帽决策管理器和 KIE 服务器设置和属性，以满足您的业务需求。您可以修改红帽决策管理器运行时、业务中心接口或 KIE 服务器的行为。

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、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 *CTO Chris Wright* 的信息。

部分 I. 管理和监控 KIE 服务器

作为系统管理员，您可以在生产环境中安装、配置和升级红帽决策管理器，快速轻松地对系统故障进行故障排除，并确保系统运行最佳。

先决条件

- 安装了 Red Hat JBoss Enterprise Application Platform 7.4。如需更多信息，请参阅 *Red Hat JBoss Enterprise Application Platform 7.4 安装指南*。
- 安装了 Red Hat Decision Manager。如需更多信息，请参阅 *规划 Red Hat Decision Manager 安装*。
- Red Hat Decision Manager 正在运行，您可以使用 **admin** 角色登录 Business Central。如需更多信息，请参阅 *规划 Red Hat Decision Manager 安装* 。

第 1 章 RED HAT DECISION MANAGER 组件

该产品由 Business Central 和 KIE 服务器组成。

- Business Central 是您创建和管理业务规则的图形用户界面。您可以在 Red Hat JBoss EAP 实例或 Red Hat OpenShift Container Platform(OpenShift)上安装 Business Central。
  Business Central 也作为独立 JAR 文件提供。您可以使用 Business Central 独立 JAR 文件运行 Business Central，而无需将其部署到应用服务器。

- KIE 服务器是执行规则和其他工件的服务器。它用于实例化和执行规则并解决规划问题。您可以在红帽 JBoss EAP 实例中安装 KIE 服务器（在红帽 JBoss EAP 集群中），在 OpenShift 上、Oracle WebLogic 服务器实例、IBM WebSphere Application Server 实例或作为 Spring Boot 应用程序的一部分。
  您可以将 KIE 服务器配置为在受管或非受管模式下运行。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。KIE 容器是项目的特定版本。如果管理 KIE 服务器，则 Process Automation Manager 控制器管理 KIE 服务器配置，并与 Process Automation Manager 控制器交互，以创建和维护 KIE 容器。

第 2 章 系统与 MAVEN 集成

Red Hat Decision Manager 旨在与红帽 JBoss 中间件 Maven Repository 和 Maven Central 存储库用作依赖性来源。确保两个依赖项都可用于项目构建。

确保您的项目取决于构件的特定版本。**LATEST** 或 **RELEASE** 通常用于指定和管理应用程序中的依赖关系版本。

- **LATEST** 是指构件的最新部署(snapshot)版本。
- **RELEASE** 是指存储库中的最后一个非快照版本。

通过使用 **LATEST** 或 **RELEASE**，当发布第三方库的新版本时，不必更新版本号，但您失去对软件版本影响的构建所带来的控制。

2.1. 对本地项目的预先验证

如果您的环境无法访问互联网，设置一个内部的 Nexus 并使用它而不是 Maven Central 或其他公共存储库。要将 JAR 从红帽决策管理器服务器的远程 Maven 存储库导入到本地 Maven 项目，请打开存储库服务器的预先接受身份验证。您可以通过在 **pom.xml** 文件中为 **guvnor-m2-repo** 配置验证完成此操作，如下所示：

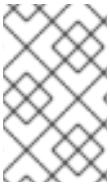
```
<server>
  <id>guvnor-m2-repo</id>
  <username>admin</username>
  <password>admin</password>
  <configuration>
    <wagonProvider>httpclient</wagonProvider>
    <httpConfiguration>
      <all>
        <usePreemptive>true</usePreemptive>
      </all>
    </httpConfiguration>
  </configuration>
</server>
```

另外，您可以使用 Base64 编码凭证设置 Authorization HTTP 标头：

```
<server>
  <id>guvnor-m2-repo</id>
  <configuration>
   <httpHeaders>
     <property>
       <name>Authorization</name>
       <!-- Base64-encoded "admin:admin" -->
       <value>Basic YWRtaW46YWRtaW4=</value>
     </property>
    </httpHeaders>
  </configuration>
</server>
```

2.2. BUSINESS CENTRAL 中的重复的 GAV 检测

在 Business Central 中，会检查所有已重复的 **GroupId**、**ArtifactId** 和 **Version** (GAV)值。如果存在 GAV 重复，则执行的操作将取消。



注意

在 **Development Mode** 中的项目禁用了重复的 GAV 检测。要在 Business Central 中启用重复的 GAV 检测，请转至 Project **Settings → General Settings → Version**，并将 **Development Mode** 选项切换为 **OFF** （如果适用）。

每次执行以下操作时执行重复的 GAV 检测：

- 保存项目的项目定义。
- 保存 **pom.xml** 文件。
- 安装、构建或部署项目。

为重复的 GAV 检查以下 Maven 存储库：

- <repositories& **gt;** 和 **pom.xml** 文件的 & **lt;distributionManagement** > 元素中指定的软件仓库。
- Maven **settings.xml** 配置文件中指定的存储库。

2.3. 在 BUSINESS CENTRAL 中管理重复的 GAV 检测设置

具有管理员角色的 Business Central 用户可以修改为项目的重复 **GroupId**、**ArtifactId** 和 **Version** (GAV) 值检查的软件仓库列表。

注意

在 Development Mode 中的项目禁用了重复的 GAV 检测。要在 Business Central 中启用重复的 GAV 检测，请转至 Project Settings → General Settings → Version，并将 Development Mode 选项切换为 OFF （如果适用）。

流程

1. **在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。**
2. 单击项目 **Settings** 选项卡，然后单击 **Validation** 以打开存储库列表。
3. 选择或清除所有列出的仓库选项，以启用或禁用重复的 **GAV** 检测。

   以后，只针对您为验证启用的存储库，才会报告重复的 GAV。

注意

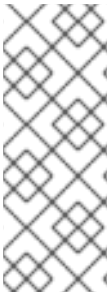
要禁用此功能，将系统启动时 Business Central 的 org.guvnor.project.gav.check.disabled 系统属性设置为 true ：

```
$ ~/EAP_HOME/bin/standalone.sh -c standalone-full.xml
-Dorg.guvnor.project.gav.check.disabled=true
```

第 3 章 将补丁更新和次版本升级到 RED HAT PROCESS AUTOMATION MANAGER

自动更新工具通常随补丁更新和 Red Hat Process Automation Manager 的新次要版本一起提供，以便于更新 Red Hat Process Automation Manager 的某些组件，如 Business Central、KIE Server 和无头进程自动化管理器控制器。其他 Red Hat Process Automation Manager 工件（如决策引擎和独立 Business Central）作为每个次发行版本的新工件发布，您必须重新安装它们以应用更新。

您可以使用相同的自动更新工具，将补丁更新和次版本升级到 Red Hat Process Automation Manager 7.13。Red Hat Process Automation Manager 的补丁更新（如从 7.13 升级到 7.13.2）包括最新的安全更新和程序错误修复。Red Hat Process Automation Manager 的次发行版本（如从版本 7.12.x 升级到 7.13），包括功能增强、安全更新和程序错误修复。



注意

Red Hat Process Automation Manager 更新工具中仅包含对 Red Hat Process Automation Manager 的更新。红帽 JBoss EAP 的更新必须使用红帽 JBoss EAP 补丁发布来应用。有关红帽 JBoss EAP 补丁的更多信息，请参阅 *Red Hat JBoss EAP 补丁和升级指南*。

先决条件

- 您的 Red Hat Process Automation Manager 和 KIE 服务器实例没有运行。在运行 Red Hat Process Automation Manager 或 KIE Server 实例时，不要应用更新。

流程

1. 导航到红帽客户门户网站中的 Software Downloads 页面（需要登录），然后从下拉列表中选择产品和版本。

   如果您要升级到 Red Hat Process Automation Manager 的新次要版本，如从 7.12.x 升级到 7.13，首先将最新的补丁更新应用到您当前的 Red Hat Process Automation Manager 版本，然后按照此流程升级到新的次版本。

2. 单击 Patches，下载 Red Hat Process Automation Manager [VERSION] Update Tool，并将下载的 rhpam-$VERSION-update.zip 文件提取到临时目录中。

   这个版本工具会自动更新 Red Hat Process Automation Manager 的某些组件，如 Business Central、KIE Server 和无头进程自动化管理器控制器。首先使用此次更新工具来应用更新，然后安装与 Red Hat Process Automation Manager 发行版本相关的任何其他更新或新发行工件。

3.

如果要保留更新工具更新的任何文件，进入提取的 rhpam-$VERSION-update 文件夹，打开 blacklist.txt 文件，并将相对路径添加到您不想更新的文件中。

当在 blacklist.txt 文件中列出文件时，更新脚本不会将该文件替换为新版本，而是将该文件保留到位，并在同一位置将新版本添加到 .new 后缀。如果您的块设备不再被分发，更新工具会创建一个带有 .removed 后缀的空标志文件。然后，您可以选择手动保留、合并或删除这些新文件。

在 blacklist.txt 文件中排除的文件示例：

```
WEB-INF/web.xml  // Custom file
styles/base.css  // Obsolete custom file kept for record
```

更新后阻止的文件目录的内容：

```
$ ls WEB-INF
web.xml web.xml.new
```

```
$ ls styles
base.css base.css.removed
```

4.

在命令终端中，导航到提取 rhpam-$VERSION-update.zip 文件的临时目录，并以以下格式运行 apply-updates 脚本：

重要

在应用更新前，请确保您的 Red Hat Process Automation Manager 和 KIE 服务器实例没有运行。在运行 Red Hat Process Automation Manager 或 KIE Server 实例时，不要应用更新。

在 Linux 或基于 Unix 的系统中：

```
$ ./apply-updates.sh $DISTRO_PATH $DISTRO_TYPE
```

在 Windows 中：

```
$ .\apply-updates.bat $DISTRO_PATH $DISTRO_TYPE
```

$DISTRO_PATH 部分是相关分发目录的路径，而 $DISTRO_TYPE 部分是您使用此更新进行更新的发行类型。

Red Hat Process Automation Manager 更新工具支持以下发行版类型：

- **rhpam-business-central-eap7-deployable: Updates Business Central (business-central.war)**
- **rhpam-kie-server-ee8: Updates KIE Server (kie-server.war)**

注意

更新工具将会更新并替换红帽 JBoss EAP EE7 到红帽 JBoss EAP EE8。红帽 JBoss EAP EE7 用于 WebLogic 和 WebSphere，而版本 EE8 则用于红帽 JBoss EAP。确保更新工具不会更新 WebLogic 和 WebSphere 上的 KIE 服务器。

- **rhpam-kie-server-jws: Updates KIE Server on Red Hat JBoss Web Server (kie-server.war)**
- **rhpam-controller-ee7 ：更新无头流程自动化管理器控制器(controller.war)**
- **rhpam-controller-jws ：在 Red Hat JBoss Web Server (controller.war)上更新无头流程自动化管理器控制器**

在 Red Hat JBoss EAP 上为完整的 Red Hat Process Automation Manager 发行版更新到 Business Central 和 KIE 服务器示例：

```
$ ./apply-updates.sh ~EAP_HOME/standalone/deployments/business-central.war
rhpam-business-central-eap7-deployable

$ ./apply-updates.sh ~EAP_HOME/standalone/deployments/kie-server.war rhpam-kie-
server-ee8
```

使用无头进程自动化管理器控制器的示例：

```
$ ./apply-updates.sh ~EAP_HOME/standalone/deployments/controller.war rhpam-controller-ee7
```

更新脚本会在提取的 **rhpam-$VERSION-update** 文件夹中创建一个 备份 文件夹，其中包含指定发行版的副本，然后继续进行更新。

5. 更新工具完成后，返回到红帽客户门户网站的 **Software Downloads** 页面，在其中下载更新工具并安装与您的 **Red Hat Process Automation Manager** 发行版本相关的任何其他更新或新发行工件。

对于 **Red Hat Process Automation Manager** 发行版本中已存在的文件，如决策引擎或其他附加组件的 **.jar** 文件，请将文件的现有版本替换为红帽客户门户网站中的新版本。

6. 如果您使用独立的 **Red Hat Process Automation Manager 7.13.2 Maven Repository** 工件 **(rhpam-7.13.2-maven-repository.zip)**，如在 **air-gap** 环境中，下载 **Red Hat Process Automation Manager 7.13.2 Maven Repository**，并将下载的 **rhpam-7.13.2-maven-repository.zip** 文件提取到现有的 **~/maven-repository** 目录中，以更新相关的内容。

Maven 存储库更新示例：

```
$ unzip -o rhpam-7.13.2-maven-repository.zip 'rhba-7.13.2.GA-maven-repository/maven-repository/*' -d /tmp/rhbaMavenRepoUpdate

$ mv /tmp/rhbaMavenRepoUpdate/rhba-7.13.2.GA-maven-repository/maven-repository/ $REPO_PATH/
```

注意

在完成更新后，您可以删除 **/tmp/rhbaMavenRepoUpdate** 文件夹。

7. 可选： 如果要将 **Red Hat Process Automation Manager** 从使用基于属性的用户存储改为基于文件的用户存储，请完成以下步骤：

a. 进入 **$JBOSS_HOME** 目录并运行以下命令之一：

- 在 **Linux** 或基于 **Unix** 的系统中：

```
$ ./bin/standalone.sh --admin-only -c standalone-full.xml
```

```
$ ./bin/jboss-cli.sh --connect --file=rhpam-$VERSION-update/elytron/add-kie-fs-realm.cli
```

- **在 Windows 中：**

```
$ ./bin/standalone.bat --admin-only -c standalone-full.xml
```

```
$ ./bin/jboss-cli.bat --connect --file=rhpam-$VERSION-update/elytron/add-kie-fs-realm.cli
```

b. **运行以下命令：**

- **在 Linux 或基于 Unix 的系统中：**

```
$ ./bin/elytron-tool.sh filesystem-realm --users-file standalone/configuration/application-users.properties --roles-file standalone/configuration/application-roles.properties --output-location standalone/configuration/kie-fs-realm-users --filesystem-realm-name kie-fs-realm-users
```

- **在 Windows 中：**

```
$ ./bin/elytron-tool.bat filesystem-realm --users-file standalone/configuration/application-users.properties --roles-file standalone/configuration/application-roles.properties --output-location standalone/configuration/kie-fs-realm-users --filesystem-realm-name kie-fs-realm-users
```

c. **进入您提取 rhpam-$VERSION-update.zip 文件的目录，并运行以下命令来应用 kie-fs-realm 补丁：**

- **在 Linux 或基于 Unix 的系统中：**

```
$ ./elytron/kie-fs-realm-patch.sh ~/$JBOSS_HOME/standalone/configuration/kie-fs-realm-users/
```

- **在 Windows 中：**

```
$ ./elytron/kie-fs-realm-patch.bat ~/$JBOSS_HOME/standalone/configuration/kie-fs-realm-users/
```

8. 完成应用所有相关更新后，启动 Red Hat Process Automation Manager 和 KIE Server 并登录到 Business Central。

9. 验证 Business Central 窗口中的所有项目数据都显示和准确，然后在 Business Central 窗口右上角存在且准确，点击您的资料名称并点击 About 来验证更新的产品版本号。

如果您遇到错误或注意到 Business Central 中缺少的数据，您可以恢复 rhpam-$VERSION-update 文件夹中 备份 文件夹中的内容，以恢复更新工具更改。您还可以在红帽客户门户网站中重新安装以前版本的 Red Hat Process Automation Manager 中的相关发行工件。恢复之前的分发后，您可以重新尝试运行更新。

第 4 章 配置并启动 KIE 服务器

您可以通过在启动 KIE 服务器时定义必要的配置来配置 KIE 服务器位置、用户名、密码和其他相关属性。

流程

导航到 Red Hat Decision Manager 7.13 bin 目录，并使用以下属性启动新的 KIE 服务器：根据您的环境调整具体属性。

```
$ ~/EAP_HOME/bin/standalone.sh --server-config=standalone-full.xml (1)
-Dorg.kie.server.id=myserver (2)
-Dorg.kie.server.user=kie_server_username (3)
-Dorg.kie.server.pwd=kie_server_password (4)
-Dorg.kie.server.controller=http://localhost:8080/business-central/rest/controller (5)
-Dorg.kie.server.controller.user=controller_username (6)
-Dorg.kie.server.controller.pwd=controller_password (7)
-Dorg.kie.server.location=http://localhost:8080/kie-server/services/rest/server (8)
```

(1) 使用 standalone-full.xml 服务器配置文件启动命令

(2) 必须与 Business Central 中定义的服务器配置名称匹配的服务器 ID

(3) 要从 Process Automation Manager 控制器与 KIE 服务器连接的用户名

(4) 从 Process Automation Manager 控制器与 KIE 服务器连接的密码

(5) 进程自动化管理器控制器位置，使用 /rest/controller 后缀的 Business Central URL

(6) 连接到 Process Automation Manager 控制器 REST API 的用户名

(7)

8

KIE 服务器位置（本例中的与 Business Central 相同的实例上）

注意

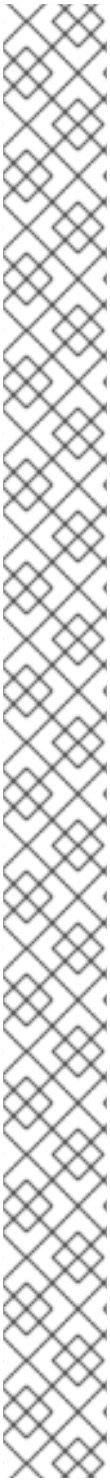
如果在单独的应用程序服务器实例上（红帽 JBoss EAP 或其他）安装 Business Central 和 KIE 服务器，请为 KIE 服务器位置使用单独的端口以避免与 Business Central 产生端口冲突。如果还没有配置单独的 KIE 服务器端口，您可以添加端口偏移并在 KIE 服务器属性中相应地调整 KIE 服务器端口值。

例如：

```
-Djboss.socket.binding.port-offset=150
-Dorg.kie.server.location=http://localhost:8230/kie-server/services/rest/server
```

如果 Business Central 端口是 8080，如本例中所示，则 KIE 服务器端口定义偏移 150 为 8230。

KIE 服务器连接到新的 Business Central，并收集要部署的部署单元（KIE 容器）的列表。



注意

当您在依赖项 JAR 文件中使用类来访问 KIE 服务器客户端的 KIE 服务器时，您可以在 Business Central 中获取 ConversionException 和 ForbiddenClassException。为了避免在 Business Central 中生成这些例外，请执行以下操作之一：

- **如果在客户端上生成异常，请在 kie-server 客户端中添加以下系统属性：**

```
System.setProperty("org.kie.server.xstream.enabled.packages", "org.example.**");
```

- **如果在服务器端生成异常，从 Red Hat Decision Manager 安装目录中打开 standalone-full.xml，请在 <system-properties> 标签下设置以下属性：**

```
<property name="org.kie.server.xstream.enabled.packages" value="org.example.**"/>
```

- **设置以下 JVM 属性：**

```
-Dorg.kie.server.xstream.enabled.packages=org.example.**
```

预期不会使用这些系统属性配置 KJAR 中存在的类。确保在系统属性中只使用已知的类来避免任何漏洞。

org.example 是一个示例软件包，您可以定义要使用的任何软件包。您可以指定用逗号分开的多个软件包，如 org.example1.* * , org.example2.* * , org.example3.* *。

您还可以添加特定的类，例如 org.example1.Mydata1、org.example2.Mydata2。

第 5 章 受管 KIE 服务器

受管实例需要一个可用的 Process Automation Manager 控制器来启动 KIE 服务器。

流程自动化管理器控制器以集中的方式管理 KIE 服务器配置。每个流程自动化管理器控制器可以同时管理多个配置，并可在环境中有多个流程自动化管理器控制器。受管 KIE 服务器可以使用 Process Automation Manager 控制器列表进行配置，但每次只能连接到一个控制器。

重要

所有 Process Automation Manager 控制器都应同步，以确保为服务器提供了相同的配置集合，而不考虑它连接的 Process Automation Manager 控制器。

当使用 Process Automation Manager 控制器列表配置 KIE 服务器时，它将尝试在启动时连接到每个控制器，直到连接成功建立其中之一。如果无法建立连接，服务器将不会启动，即使有可用的本地存储也可用。这样可确保一致性并防止服务器使用冗余配置运行。

注意

要在不连接到 Process Automation Manager 控制器的情况下以独立模式运行 KIE 服务器，请参阅 *第 6 章 非受管 KIE 服务器*。

第 6 章 非受管 KIE 服务器

非受管 KIE 服务器是一个独立实例，因此必须使用 KIE 服务器本身的 REST/JMS API 单独配置。服务器会自动保留该配置到文件中，并用作内部服务器状态（在重新启动时）。

以下操作会更新配置：

- 部署 KIE 容器
- 取消部署 KIE 容器
- 启动 KIE 容器
- 停止 KIE 容器

注意

如果重启 KIE 服务器，它将尝试在关闭前重新建立相同的状态。因此，运行的 KIE 容器（部署单元）将启动，但不会停止它们。

第 7 章 在 KIE 服务器和 BUSINESS CENTRAL 中配置环境模式

您可以将 KIE 服务器设置为在 生产环境 模式下运行，或者在 开发 模式下运行。开发模式提供了灵活的部署策略，可让您更新现有部署单元（KIE 容器），同时维护活跃的进程实例来进行小更改。它还允许您在更新活跃进程实例进行更大更改前重置部署单元状态。生产模式是生产环境的最佳选择，每个部署都会创建一个新的部署单元。

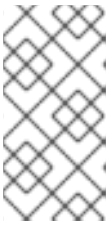
在开发环境中，您可以点击 Deploy in Business Central 将构建的 KJAR 文件部署到 KIE 服务器，而无需停止任何正在运行的实例（如果适用），或者单击 Redeploy 以部署构建的 KJAR 文件并替换所有实例。下次部署或重新部署构建的 KJAR 时，以前的部署单元（KIE 容器）会在同一目标 KIE 服务器中自动更新。

在生产环境中，业务中心的 Redeploy 选项被禁用，您只能单击 Deploy 以将构建的 KJAR 文件部署到 KIE 服务器上的新部署单元（KIE 容器）。

流程

1. 要配置 KIE 服务器环境模式，请将 org.kie.server.mode 系统属性设置为 org.kie.server.mode=development 或 org.kie.server.mode=production。

2. 要在 Business Central 中配置项目部署行为，请转至 Project Settings → General Settings → Version，再切换 Development Mode 选项。



注意

默认情况下，Business Central 中的 KIE 服务器和所有新项目均为开发模式。

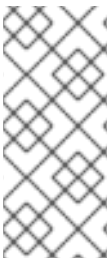
您不能部署打开开发模式的项目，或使用手动将 SNAPSHOT 版本后缀添加到生产模式中的 KIE 服务器。

第 8 章 配置 KIE 服务器以连接至 BUSINESS CENTRAL

警告

本节提供了一个示例设置，可用于测试目的。部分值不适用于生产环境，并被标记为。

如果在 Red Hat Process Automation Manager 环境中没有配置 KIE 服务器，或者在 Red Hat Process Automation Manager 环境中需要额外的 KIE 服务器，您必须配置 KIE 服务器来连接到 Business Central。



注意

如果要在 Red Hat OpenShift Container Platform 上部署 KIE 服务器，*请参阅使用 Operator 在 Red Hat OpenShift Container Platform 4 上部署 Red Hat Decision Manager 环境*。

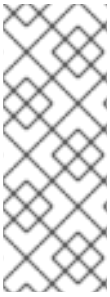
可以管理或非受管 KIE 服务器。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。如果管理 KIE 服务器，则 Process Automation Manager 控制器管理 KIE 服务器配置，并与 Process Automation Manager 控制器交互，以创建和维护 KIE 容器。

注意

如果 KIE 服务器由 Business Central 管理，并且您已从 ZIP 文件安装了红帽决策管理器，请本小节中所述的更改。如果安装了 Business Central，您可以使用无头流程自动化管理器控制器来管理 KIE 服务器，如 第 9 章 *安装并运行无头 Process Automation Manager 控制器* 所述。

先决条件

- Business Central 和 KIE 服务器安装在红帽 JBoss EAP 安装基本目录中(*EAP_HOME*)。



注意

您必须在生产环境中的不同服务器上安装 Business Central 和 KIE 服务器。在本例中，我们仅使用一个名为 controllerUser 的用户，其包含 rest-all 和 kie-server 角色。但是，如果您在同一服务器上安装 KIE 服务器和 Business Central，例如在开发环境中，请更改共享 standalone-full.xml 文件，如本节所述。

- 存在具有以下角色的用户：
  - 在 Business Central 中，拥有 其余角色的用户
  - 在 KIE 服务器上，角色为 kie-server的用户

流程

1. 在 Red Hat Process Automation Manager 安装目录中，进入 standalone-full.xml 文件。例如，如果您为 Red Hat Process Automation Manager 使用 Red Hat JBoss EAP 安装，请转至 $EAP_HOME/standalone/configuration/standalone-full.xml。

2. 打开 standalone-full.xml 文件并在 < system-properties> 标签下设置以下 JVM 属性：

表 8.1. 管理的 KIE 服务器实例的 JVM 属性

属性	值	备注
org.kie.server.id	default-kie-server	KIE 服务器 ID。
org.kie.server.controller	http://localhost:8080/business-central/rest/controller	Business Central 的位置。连接到 Business Central API 的 URL。
org.kie.server.controller.user	controllerUser	用户名（具有 rest-all 角色）可以登录到 Business Central。
org.kie.server.controller.pwd	controllerUser1234;	登录 Business Central 的用户的密码。
org.kie.server.location	http://localhost:8080/kie-server/services/rest/server	KIE 服务器的位置。连接到 KIE 服务器的 API 的 URL。

表 8.2. Business Central 实例的 JVM 属性

属性	值	备注
org.kie.server.user	**controllerUser**	使用角色 **kie-server** 的用户名。
org.kie.server.pwd	**controllerUser1234;**	用户的密码。

以下示例演示了如何配置 KIE 服务器实例：

```
<property name="org.kie.server.id" value="default-kie-server"/>
<property name="org.kie.server.controller" value="http://localhost:8080/business-central/rest/controller"/>
<property name="org.kie.server.controller.user" value="controllerUser"/>
<property name="org.kie.server.controller.pwd" value="controllerUser1234;"/>
<property name="org.kie.server.location" value="http://localhost:8080/kie-server/services/rest/server"/>
```

以下示例演示了如何为 Business Central 实例配置 ：

```
<property name="org.kie.server.user" value="controllerUser"/>
<property name="org.kie.server.pwd" value="controllerUser1234;"/>
```

3. **要验证 KIE 服务器是否已成功启动，请在 KIE 服务器运行时向 http://*SERVER:PORT*/kie-server/services/rest/server/ 发送 GET 请求。有关在 KIE 服务器上运行 Red Hat Process Automation Manager 的更多信息，请参阅 *运行 Red Hat Process Automation Manager*。**

成功验证后，您会收到类似以下示例的 XML 响应：

```
<response type="SUCCESS" msg="Kie Server info">
  <kie-server-info>
    <capabilities>KieServer</capabilities>
    <capabilities>BRM</capabilities>
    <capabilities>BPM</capabilities>
    <capabilities>CaseMgmt</capabilities>
    <capabilities>BPM-UI</capabilities>
    <capabilities>BRP</capabilities>
    <capabilities>DMN</capabilities>
    <capabilities>Swagger</capabilities>
    <location>http://localhost:8230/kie-server/services/rest/server</location>
    <messages>
      <content>Server KieServerInfo{serverId='first-kie-server', version='7.5.1.Final-redhat-1', location='http://localhost:8230/kie-server/services/rest/server', capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger]}started successfully at Mon Feb 05 15:44:35 AEST 2018</content>
      <severity>INFO</severity>
```

```
            <timestamp>2018-02-05T15:44:35.355+10:00</timestamp>
        </messages>
        <name>first-kie-server</name>
        <id>first-kie-server</id>
        <version>7.5.1.Final-redhat-1</version>
    </kie-server-info>
</response>
```

4. 验证成功注册：

    a. 登录到 Business Central。

    b. 点 Menu → Deploy → Execution Servers。

    如果注册成功，您将看到注册的服务器 ID。

第 9 章 安装并运行无头 PROCESS AUTOMATION MANAGER 控制器

您可以将 KIE 服务器配置为在受管或非受管模式下运行。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。如果管理 KIE 服务器，则 Process Automation Manager 控制器管理 KIE 服务器配置，并与 Process Automation Manager 控制器交互，以创建和维护 KIE 容器。

Business Central 有一个嵌入式 Process Automation Manager 控制器。如果您安装 Business Central，请使用 执行服务器 页面来创建和维护 KIE 容器。如果要在没有 Business Central 的情况下自动执行 KIE 服务器管理，您可以使用无头流程 Automation Manager 控制器。

9.1. 使用安装程序使用 PROCESS AUTOMATION MANAGER 控制器配置 KIE 服务器

KIE 服务器可由 Process Automation Manager 控制器管理，也可以是非受管。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。如果管理 KIE 服务器，则 Process Automation Manager 控制器管理 KIE 服务器配置，并与 Process Automation Manager 控制器交互，以创建和维护 KIE 容器。

Process Automation Manager 控制器与 Business Central 集成。如果安装了 Business Central，您可以使用 Business Central 中的 Execution Server 页面与 Process Automation Manager 控制器交互。

您可以使用互动或 CLI 模式的安装程序来安装 Business Central 和 KIE 服务器，然后使用 Process Automation Manager 控制器配置 KIE 服务器。

先决条件

- 提供了具有备份 Red Hat JBoss EAP 7.4 服务器安装的两个计算机。
- 需要足够的用户权限以完成安装。

流程

1. 在第一个计算机上，以交互模式或 CLI 模式运行安装程序。如需更多信息，*请参阅在 Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Decision Manager*。
2. 在 Component Selection 页面中，清除 KIE 服务器 框。

3. 完成 Business Central 安装。

4. 在第二个计算机上，以交互模式或 CLI 模式运行安装程序。

5. 在 组件选择 页面中，清除 Business Central 框。

6. 在 Configure Runtime Environment 页面中，选择 Perform Advanced Configuration。

7. 选择" 自定义 KIE 服务器"属性，然后单击" 下一步"。

8. 输入 Business Central 的控制器 URL，并为 KIE 服务器配置其他属性。控制器 URL 具有以下格式，其中 & lt;HOST:PORT > 是第二个计算机上的 Business Central 地址：

```
<HOST:PORT>/business-central/rest/controller
```

9. 完成安装。

10. 要验证 Process Automation Manager 控制器现在与 Business Central 集成，请转至 Business Central 中的 Execution Servers 页面，并确认您配置的 KIE 服务器出现在 REMOTE SERVERS 下。

9.2. 安装无头流程自动化管理器控制器

您可以安装无头进程自动化管理器控制器，并使用 REST API 或 KIE Server Java Client API 与其交互。

先决条件

- 提供了备份的 Red Hat JBoss EAP 安装版本 7.4。红帽 JBoss EAP 安装的基础目录称为 *EAP_HOME*。
- 需要足够的用户权限以完成安装。

流程

流程

1. 进入红帽客户门户网站中的 Software Downloads 页面（需要登录），然后从下拉列表中选择产品和版本：
   - 产品 ： 流程自动化管理器
   - Version: 7.13.2
2. 下载 Red Hat Process Automation Manager 7.13.2 Add Ons (rhpam-7.13.2-add-ons.zip 文件)。
3. 提取 rhpam-7.13.2-add-ons.zip 文件。rhpam-7.13.2-controller-ee7.zip 文件位于提取的目录中。
4. 将 rhpam-7.13.2-controller-ee7.zip 存档提取到临时目录中。在以下示例中，此目录名为 *TEMP_DIR*。
5. 将 *TEMP_DIR*/rhpam-7.13.2-controller-ee7/controller.war 目录复制到 *EAP_HOME*/standalone/deployments/。

警告

确保您复制的无头流程自动化管理器控制器部署的名称不会与 Red Hat JBoss EAP 实例中的现有部署冲突。

6. 将 *TEMP_DIR*/rhpam-7.13.2-controller-ee7/SecurityPolicy/ 目录的内容复制到 *EAP_HOME*/bin。
7. 当系统提示覆盖文件时，选择是 。
8.

在 *EAP_HOME*/standalone/deployments/ 目录中，创建名为 controller.war.dodeploy 的空文件。此文件可确保服务器启动时自动部署无头 Process Automation Manager 控制器。

9.2.1. 创建无头进程自动化管理器控制器用户

在使用无头 Process Automation Manager 控制器前，您必须创建一个具有 kie-server 角色的用户。

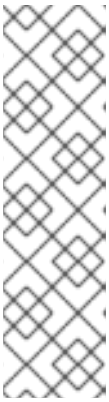
先决条件

- 无头流程自动化管理器控制器安装在 Red Hat JBoss EAP 安装(*EAP_HOME*)的基本目录中。

流程

1. 在终端应用中，导航到 *EAP_HOME*/bin 目录。
2. 输入以下命令，将 &lt ;USERNAME& gt ; 和 <PASSWORD > 替换为您选择的用户名和密码。

```
$ ./bin/jboss-cli.sh --commands="embed-server --std-
out=echo,/subsystem=elytron/filesystem-realm=ApplicationRealm:add-
identity(identity=<USERNAME>),/subsystem=elytron/filesystem-
realm=ApplicationRealm:set-password(identity=<USERNAME>, clear=
{password='<PASSWORD>'}),/subsystem=elytron/filesystem-
realm=ApplicationRealm:add-identity-attribute(identity=<USERNAME>, name=role,
value=['kie-server'])"
```



注意

确保指定的用户名与现有用户、角色或组不同。例如，不要创建用户名为 admin 的用户。

密码必须至少包含八个字符，且必须至少包含一个数字和一个非字母数字字符，但不包括 和 (ampersand)。

3. 记录您的用户名和密码。

9.2.2. 配置 KIE 服务器和无头进程自动化管理器控制器

如果 KIE 服务器将由无头进程自动化管理器控制器管理，您必须在 KIE Server 安装中编辑 standalone-full.xml 文件，以及在无头进程自动化管理器控制器安装中编辑 standalone.xml 文件。

先决条件

- KIE 服务器安装在 *EAP_HOME* 中。
- 无头流程自动化管理器控制器安装在 *EAP_HOME* 中。

注意

您应该在生产环境中的不同服务器上安装 KIE 服务器和无头进程自动化管理器控制器。但是，如果您在同一服务器上安装 KIE 服务器和无头进程自动化管理器控制器，例如在开发环境中，在共享的 standalone-full.xml 文件中进行这些更改。

- 在 KIE 服务器节点上，存在具有 kie-server 角色的用户。
- 在服务器节点上，存在具有 kie-server 角色的用户。

流程

1. 在 *EAP_HOME*/standalone/configuration/standalone-full.xml 文件中，将以下属性添加到 < system-properties> 部分，并将 & lt;USERNAME > 和 <USER_PWD &gt ; 替换为该用户的凭证：

```
<property name="org.kie.server.user" value="<USERNAME>"/>
<property name="org.kie.server.pwd" value="<USER_PWD>"/>
```

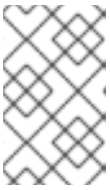
2. 在 KIE Server *EAP_HOME*/standalone/configuration/standalone-full.xml 文件中，将以下属性添加到 < system-properties> 部分：

```
<property name="org.kie.server.controller.user" value="<CONTROLLER_USER>"/>
<property name="org.kie.server.controller.pwd" value="<CONTROLLER_PWD>"/>
<property name="org.kie.server.id" value="<KIE_SERVER_ID>"/>
```

```
  <property name="org.kie.server.location" value="http://<HOST>:<PORT>/kie-
server/services/rest/server"/>
  <property name="org.kie.server.controller" value="<CONTROLLER_URL>"/>
```

3. 在这个文件中，替换以下值：

- 将 & lt;CONTROLLER_USER> 和 <CONTROLLER_PWD& gt; 替换为用户的凭证，并将 kie-server 角色替换为 kie-server 角色。
- 将 <KIE_SERVER_ID > 替换为 KIE 服务器安装的 ID 或名称，如 rhpam-7.13.2-kie-server-1。
- 将 <HOST > 替换为 KIE 服务器主机的 ID 或名称，例如 localhost 或 192.7.8.9。
- 将 <PORT > 替换为 KIE 服务器主机的端口，例如 8080。



注意

org.kie.server.location 属性指定 KIE 服务器的位置。

- 将 <CONTROLLER_URL > 替换为无头进程自动化管理器控制器的 URL。KIE 服务器在启动过程中连接到此 URL。

9.3. 运行无头进程自动化管理器控制器

在 Red Hat JBoss EAP 上安装无头 Process Automation Manager 控制器后，使用此流程运行无头 Process Automation Manager 控制器。

先决条件

- 无头流程自动化管理器控制器在 Red Hat JBoss EAP 安装(*EAP_HOME*)的基域中安装和配置。

流程

1. 在终端应用中，导航到 *EAP_HOME*/bin。

2. 如果您在安装 KIE 服务器的 Red Hat JBoss EAP 实例相同的 Red Hat JBoss EAP 实例上安装了无头 Process Automation Manager 控制器，请输入以下命令之一：

   - 在 Linux 或基于 UNIX 的系统中：

     ```
     $ ./standalone.sh -c standalone-full.xml
     ```

   - 在 Windows 中：

     ```
     standalone.bat -c standalone-full.xml
     ```

3. 如果您在安装 KIE 服务器的 Red Hat JBoss EAP 实例中的独立 Red Hat JBoss EAP 实例上安装了无头 Process Automation Manager 控制器，请使用 standalone.sh 脚本启动无头 Process Automation Manager 控制器：

   注意

   在这种情况下，请确保对 standalone.xml 文件进行所有必要的配置更改。

   - 在 Linux 或基于 UNIX 的系统中：

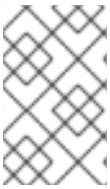
     ```
     $ ./standalone.sh
     ```

   - 在 Windows 中：

     ```
     standalone.bat
     ```

4. 要验证无头流程自动化管理器控制器是否在 Red Hat JBoss EAP 上工作，请输入以下命令，其中 < CONTROLLER> 和 & lt;CONTROLLER_PWD > 是用户名和密码。此命令的输出提供有关 KIE 服务器实例的信息。

   ```
   curl -X GET "http://<HOST>:<PORT>/controller/rest/controller/management/servers" -H
   "accept: application/xml" -u '<CONTROLLER>:<CONTROLLER_PWD>'
   ```



注意

另外，您可以使用 KIE Server Java API 客户端来访问无头进程自动化管理器控制器。

9.4. 使用无头进程自动化管理器控制器集群 KIE 服务器

Process Automation Manager 控制器与 Business Central 集成。但是，如果您没有安装 Business Central，您可以安装无头流程自动化管理器控制器，并使用 REST API 或 KIE Server Java 客户端 API 与其交互。

先决条件

- 提供了备份的 Red Hat JBoss EAP 安装版本 7.4 或更高版本。红帽 JBoss EAP 安装的基础目录称为 *EAP_HOME*。
- 需要足够的用户权限以完成安装。
- Red Hat *JBoss EAP 集群环境中安装和配置 Red Hat Decision Manager 所述*，具有共享文件夹的 NFS 服务器。

流程

1. 进入红帽客户门户网站中的 Software Downloads 页面（需要登录），然后从下拉列表中选择产品和版本：
   - PRODUCT: Process Automation Manager
   - Version: 7.13.2
2. 下载 Red Hat Process Automation Manager 7.13.2 Add Ons (rhpam-7.13.2-add-ons.zip 文件)。
3. 提取 rhpam-7.13.2-add-ons.zip 文件。rhpam-7.13.2-controller-ee7.zip 文件位于提取的目录中。
4.

将 rhpam-7.13.2-controller-ee7.zip 存档提取到临时目录中。在以下示例中，此目录名为 *TEMP_DIR*。

5. 将 *TEMP_DIR*/rhpam-7.13.2-controller-ee7/controller.war 目录复制到 *EAP_HOME*/standalone/deployments/。

警告

确保您复制的无头流程自动化管理器控制器部署的名称不会与 Red Hat JBoss EAP 实例中的现有部署冲突。

6. 将 *TEMP_DIR*/rhpam-7.13.2-controller-ee7/SecurityPolicy/ 目录的内容复制到 *EAP_HOME*/bin。

7. 当系统提示覆盖文件时，请单击 **Yes**。

8. 在 *EAP_HOME*/standalone/deployments/ 目录中，创建名为 controller.war.dodeploy 的空文件。此文件可确保服务器启动时自动部署无头 Process Automation Manager 控制器。

9. 在文本编辑器中打开 *EAP_HOME*/standalone/configuration/standalone.xml 文件。

10. 在 < system-properties> 元素中添加以下属性，并将 &lt ;NFS_STORAGE > 替换为存储模板配置的 NFS 存储的绝对路径：

```
<system-properties>
  <property name="org.kie.server.controller.templatefile.watcher.enabled" value="true"/>
  <property name="org.kie.server.controller.templatefile" value="<NFS_STORAGE>"/>
</system-properties>
```

模板文件包含特定部署场景的默认配置。

如果将 org.kie.server.controller.templatefile.watcher.enabled 属性的值设置为 true，则会启动一个单独的线程来监视模板文件的修改。这些检查的默认间隔为 30000 毫秒，可以由

org.kie.server.controller.templatefile.watcher.interval 系统属性进一步控制。如果此属性的值设为 false，则仅在服务器重启时检测到对模板文件的更改。

11. 要启动无头 Process Automation Manager 控制器，请导航到 *EAP_HOME*/bin 并输入以下命令：

- 在 Linux 或基于 UNIX 的系统中：

```
$ ./standalone.sh
```

- 在 Windows 中：

```
standalone.bat
```

有关在 Red Hat JBoss Enterprise Application Platform 集群环境中运行 Red Hat Decision Manager 的更多信息，请参阅 *在 Red Hat JBoss EAP 集群环境中安装和配置 Red Hat Decision Manager*。

第 10 章 部署描述符

流程和规则存储在基于 Apache Maven 的打包中，它们称为知识存档或 KJAR。规则、流程、资产和其它项目工件是由 Maven 构建和管理 JAR 文件的一部分。保存在名为 kmodule.xml 的 KJAR 中的 META-INF 目录的文件可用于定义 KIE 基础和会话。默认情况下，这个 kmodule.xml 文件为空。

每当 KIE 服务器等运行时组件要处理 KJAR 时，它会查找 kmodule.xml 以构建运行时表示。

部署描述符对 kmodule.xml 文件进行补充，并对部署提供精细的控制。这些描述符的存在是可选的，您的部署可以成功进行。您可以使用这些描述符来设置完全技术属性，包括持久性、审计和运行时策略等 meta 值。

这些描述符允许您在多个级别上配置 KIE 服务器，包括服务器级别默认、每个 KJAR 的不同部署描述符和其他服务器配置。您可以使用描述符对默认的 KIE 服务器配置进行简单自定义，可能每个 KJAR。

您可以在名为 kie-deployment-descriptor.xml 的文件中定义这些描述符，并将此文件放到 META-INF 文件夹中的 kmodule.xml 文件。您可以通过将这个默认位置和文件名指定为系统参数来更改这个默认位置和文件名：

```
-Dorg.kie.deployment.desc.location=file:/path/to/file/company-deployment-descriptor.xml
```

10.1. 部署描述符配置

通过部署描述符，用户可以在多个级别上配置执行服务器：

- *服务器级别*：主要级别，以及应用到服务器上部署的所有 KJAR 的主级别。
- *KJAR 级别*：您可以基于每个 KJAR 配置描述符。
- *部署时间级别*：在部署 KJAR 时应用的描述符。

部署描述符指定的细粒度配置项优先于服务器级别，除了基于集合的配置项目时，会合并它们。层次结构与此类似：*部署时间配置 > KJAR 配置 > 服务器配置*。

注意

部署时间配置适用于通过 REST API 进行的部署。

例如，如果在服务器级别定义的持久性模式（您可以在服务器级别定义的其中之一）是 NONE，但在 KJAR 级别中指定为 JPA，则实际模式将是 KJAR 的 JPA。如果在 KJAR（或者没有部署描述符的情况下，对于持久性模式指定）的持久性模式，它将回退到服务器级配置，在这种情况下为 NONE （如果没有服务器级别部署描述符，则为 JPA ）。

您可以配置什么配置？

高级别的技术配置细节可通过部署描述符来配置。下表列出了它们，以及每个表的可见和默认值。

表 10.1. 部署描述符

Configuration	XML Entry	Permissible Valuesible Values	默认值
运行时数据的持久性单元名称	persistence-unit	任何有效的持久性软件包名称	org.jbpm.domain
审计数据的持久性单元名称	audit-persistence-unit	任何有效的持久性软件包名称	org.jbpm.domain
持久性模式	persistence-mode	JPA, NONE	JPA
Audit 模式	audit-mode	JPA、JMS 或 NONE	JPA
运行时策略	runtime-strategy	SINGLETON、PER_REQUEST 或 PER_PROCESS_INSTANCE	单例
要注册的事件 Listener 列表	event-listeners	有效的监听程序类名称作为 **ObjectModel**	没有默认值
要注册的任务事件 Listener 列表	task-event-listeners	有效的监听程序类名称作为 **ObjectModel**	没有默认值
要注册的 Work Item 处理程序列表	work-item-handlers	有效的 Work Item Handler 类指定为 **NamedObjectHandler**	没有默认值
要注册的全局列表	全局	以 **NamedObjectModel** 形式提供的有效全局变量	没有默认值

Configuration	XML Entry	Permissible Valuesible Values	默认值
要注册的策略（用于可插拔变量持久性）	marshalling-strategies	有效的 **ObjectModel** 类	没有默认值
被授予 KJAR 资源的访问权限所需的角色	required-roles	字符串角色名称	没有默认值
KIE 会话的其他环境条目	environment-entries	有效 **NamedObjectModel**	没有默认值
KIE 会话的其他配置选项	配置	有效 **NamedObjectModel**	没有默认值
在远程服务中使用序列化的类	remoteable-class	有效 **CustomClass**	没有默认值

警告

在生产环境中，不要将 Singleton 运行时策略与 EJB Timer 调度程序（KIE 服务器的默认调度程序）一起使用。这种组合可能会导致 Hibernate 存在负载问题。如果没有特定原因来使用其他策略，则建议按进程实例运行时策略。有关此限制的更多信息，请参阅 Singleton 策略和 EJBTimerScheduler 的 Hibernate 问题。

10.2. 管理部署描述符

在 Menu → Design → Design → PROJECT_NAME → Settings → Deployments 中，可以在 Business Central 中配置部署描述符。

每次创建项目时，都会使用默认值生成 stock kie-deployment-descriptor.xml 文件。

不需要为所有 KJAR 提供完整的部署描述符。可以提供部分部署描述符，并推荐使用。例如，如果您需要使用不同的审计模式，您可以为 KJAR 指定，所有其他属性将定义在服务器级别定义的默认值。

使用 OVERRIDE_ALL 合并模式时，必须指定所有配置项目，因为相关的 KJAR 将始终使用指定的配置，且不会与层次结构中的任何其他部署描述符合并。

10.3. 限制访问运行时引擎

可以在部署描述符中编辑 **required-roles** 配置项。此属性通过确保仅授权属于此属性所定义组的用户，限制对每个KJAR 或每个服务器级别上运行时引擎的访问。

安全角色可用于限制对进程定义的访问，或者在运行时限制访问。

默认行为是根据存储库限制将所需的角色添加到此属性中。若要提供与安全域中定义的实际角色匹配的角色，您可以手动编辑这些属性。

流程

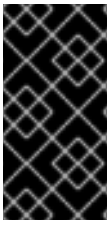
1. 要在 **Business Central** 中打开项目部署描述符配置，打开 **Menu** → **Design** → **Design** → **PROJECT_NAME** → **Settings** → **Deployments**。
2. 从配置设置列表中，单击 **Required Roles**，然后单击 **Add Required Role**。
3. 在 **Add Required Role** 窗口中，键入您要具有访问此部署的权限的角色名称，然后点 **Add**。
4. 要添加更多具有访问部署权限的角色，请重复前面的步骤。
5. 添加完所有所需的角色后，点 **Save**。

第 11 章 RED HAT DECISION MANAGER 中的 PROMETHEUS METRICS 监控

Prometheus 是一个开源系统监控工具包，可与 Red Hat Decision Manager 一同使用来收集和存储与执行业务规则、流程、决策模型和未编译(DMN)模型和其他红帽决策管理器资产相关的指标。您可以通过 REST API 调用 KIE 服务器、通过 Prometheus 表达式浏览器或使用 Grafana 等数据图形工具访问存储的指标。

您可以为内部 KIE 服务器实例、Spring Boot 上的 KIE 服务器或 Red Hat OpenShift Container Platform 上部署 KIE 服务器配置 Prometheus 指标监控。

有关 KIE 服务器与 Prometheus 公开的可用指标列表，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.2 Source Distribution /kie-server-services-prometheus/src/main/7.13.2 -sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus。https://access.redhat.com/jbossnetwork/restricted/listSoftware.html



重要

Red Hat support for Prometheus 仅限于 Red Hat 产品文档中提供的设置和配置建议。

11.1. 为 KIE 服务器配置 PROMETHEUS 指标监控

您可以将 KIE 服务器实例配置为使用 Prometheus 来收集并存储与红帽 Decision Manager 中业务资产活动相关的指标。有关 KIE 服务器与 Prometheus 公开的可用指标列表，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.2 Source Distribution /kie-server-services-prometheus/src/main/7.13.2 -sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus。https://access.redhat.com/jbossnetwork/restricted/listSoftware.html

先决条件

- 已安装 KIE 服务器。
- 您有 kie-server 用户角色访问权限到 KIE 服务器。
- 已安装 Prometheus。有关下载和使用 Prometheus 的详情，请查看 Prometheus 文档页

面。

流程

1. 在您的 KIE 服务器实例中，将 org.kie.prometheus.server.ext.disabled 系统属性设置为 false 以启用 Prometheus 扩展。您可在启动 KIE 服务器时或在 Red Hat Decision Manager 发行版的 standalone.xml 或 standalone-full.xml 文件中定义此属性。

2. 如果您在 Spring Boot 上运行 Red Hat Decision Manager，请在 application.properties 系统属性中配置所需的密钥：

Spring Boot application.properties key for Red Hat Decision Manager 和 Prometheus

```
kieserver.drools.enabled=true
kieserver.dmn.enabled=true
kieserver.prometheus.enabled=true
```

3. 在 Prometheus 发行版本的 prometheus.yaml 文件中，在 scrape_configs 部分添加以下设置，将 Prometheus 配置为从 KIE 服务器中提取指标：

prometheus.yaml 文件中的 scrape 配置

```
scrape_configs:
  - job_name: 'kie-server'
    metrics_path: /SERVER_PATH/services/rest/metrics
    basicAuth:
      username: USER_NAME
      password: PASSWORD
    static_configs:
      - targets: ["HOST:PORT"]
```

Spring Boot 的 prometheus.yaml 文件中的 scrape 配置（如果适用）

```
scrape_configs:
  - job_name: 'kie'
    metrics_path: /rest/metrics
    static_configs:
      - targets: ["HOST:PORT"]
```

根据您的 KIE 服务器位置和设置值。

4. 启动 KIE 服务器实例。

红帽 JBoss EAP 上的红帽决策管理器入门命令示例

```
$ cd ~/EAP_HOME/bin
$ ./standalone.sh --c standalone-full.xml
```

启动配置的 KIE 服务器实例后，Prometheus 开始收集指标，KIE 服务器会将指标发布到 REST API 端点 http://HOST:PORT/SERVER/services/rest/metrics （或在 Spring Boot 上，或在 Spring Boot 上）发布指标到 http://HOST:PORT/rest/metrics。

5. 在 REST 客户端或 curl 实用程序中，使用以下组件发送 REST API 请求，以验证 KIE 服务器是否发布指标：

对于 REST 客户端：

- 身份验证 ：使用 kie-server 角色输入 KIE 服务器用户的用户名和密码。
- HTTP 标头 ：设置以下标头：
  - 接受:application/json

- **HTTP 方法** ：设置为 **GET**。
- **URL** ：输入 **KIE** 服务器 **REST API** 基础 **URL** 和指标端点，如 **http://localhost:8080/kie-server/services/rest/metrics**（或在 **Spring Boot**、**http://localhost:8080/rest/metrics**上）。

对于 **curl** 实用程序：

- **-U** ：使用 **kie-server** 角色输入 **KIE** 服务器用户的用户名和密码。
- **-H** ：设置以下标头：
  - **接受:application/json**
- **-X** ：设置为 **GET**。
- **URL** ：输入 **KIE** 服务器 **REST API** 基础 **URL** 和指标端点，如 **http://localhost:8080/kie-server/services/rest/metrics**（或在 **Spring Boot**、**http://localhost:8080/rest/metrics**上）。

红帽 JBoss EAP 上红帽决策管理器的 curl 命令示例

```
curl -u 'baAdmin:password@1' -X GET "http://localhost:8080/kie-server/services/rest/metrics"
```

Spring Boot 上的 Red Hat Decision Manager 的 curl 命令示例

```
curl -u 'baAdmin:password@1' -X GET "http://localhost:8080/rest/metrics"
```

服务器响应示例

```
# HELP kie_server_container_started_total Kie Server Started Containers
# TYPE kie_server_container_started_total counter
kie_server_container_started_total{container_id="task-assignment-kjar-1.0",} 1.0
# HELP solvers_running Number of solvers currently running
# TYPE solvers_running gauge
solvers_running 0.0
# HELP dmn_evaluate_decision_nanosecond DMN Evaluation Time
# TYPE dmn_evaluate_decision_nanosecond histogram
# HELP solver_duration_seconds Time in seconds it took solver to solve the constraint
problem
# TYPE solver_duration_seconds summary
solver_duration_seconds_count{solver_id="100tasks-5employees.xml",} 1.0
solver_duration_seconds_sum{solver_id="100tasks-5employees.xml",} 179.828255925
solver_duration_seconds_count{solver_id="24tasks-8employees.xml",} 1.0
solver_duration_seconds_sum{solver_id="24tasks-8employees.xml",} 179.995759653
# HELP drl_match_fired_nanosecond Drools Firing Time
# TYPE drl_match_fired_nanosecond histogram
# HELP dmn_evaluate_failed_count DMN Evaluation Failed
# TYPE dmn_evaluate_failed_count counter
# HELP kie_server_start_time Kie Server Start Time
# TYPE kie_server_start_time gauge
kie_server_start_time{name="myapp-kieserver",server_id="myapp-
kieserver",location="http://myapp-kieserver-demo-
monitoring.127.0.0.1.nip.io:80/services/rest/server",version="7.4.0.redhat-20190428",}
1.557221271502E12
# HELP kie_server_container_running_total Kie Server Running Containers
# TYPE kie_server_container_running_total gauge
kie_server_container_running_total{container_id="task-assignment-kjar-1.0",} 1.0
# HELP solver_score_calculation_speed Number of moves per second for a particular solver
solving the constraint problem
# TYPE solver_score_calculation_speed summary
solver_score_calculation_speed_count{solver_id="100tasks-5employees.xml",} 1.0
solver_score_calculation_speed_sum{solver_id="100tasks-5employees.xml",} 6997.0
solver_score_calculation_speed_count{solver_id="24tasks-8employees.xml",} 1.0
solver_score_calculation_speed_sum{solver_id="24tasks-8employees.xml",} 19772.0
```

如果 KIE 服务器中没有指标，请查看并验证本节中描述的 KIE 服务器和 Prometheus 配置。

您还可以在 http://HOST:PORT/graph 中与 Prometheus 表达式浏览器中收集的指标进行交互，或者将 Prometheus 数据源与数据图形工具（如 Grafana）集成：

图 11.1. 带有 KIE 服务器指标的 Prometheus 表达式浏览器

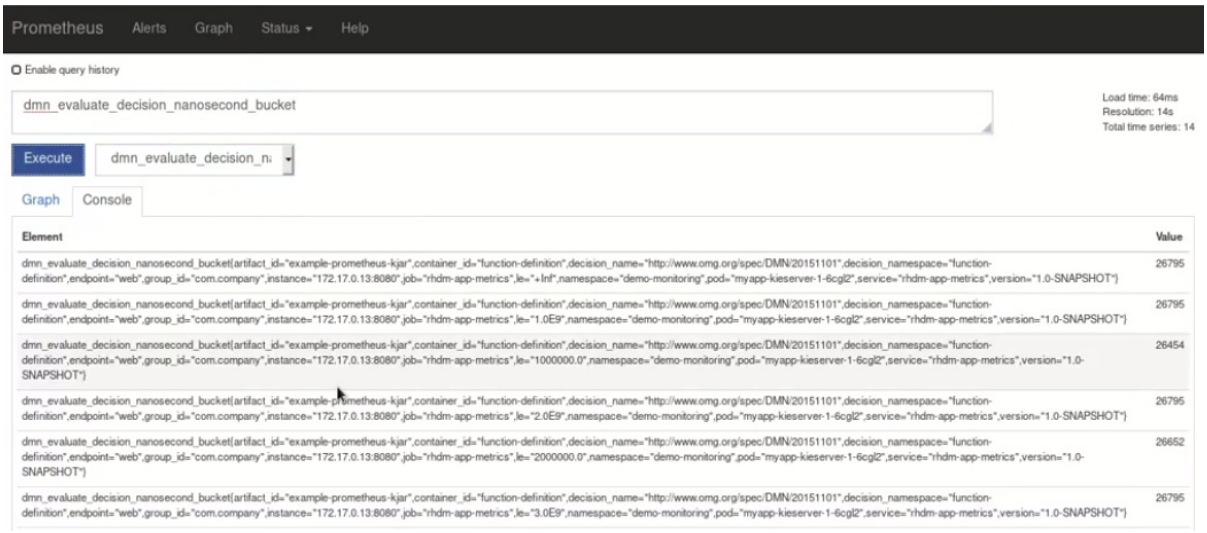


图 11.2. 带有 KIE 服务器目标的 Prometheus 表达式浏览器

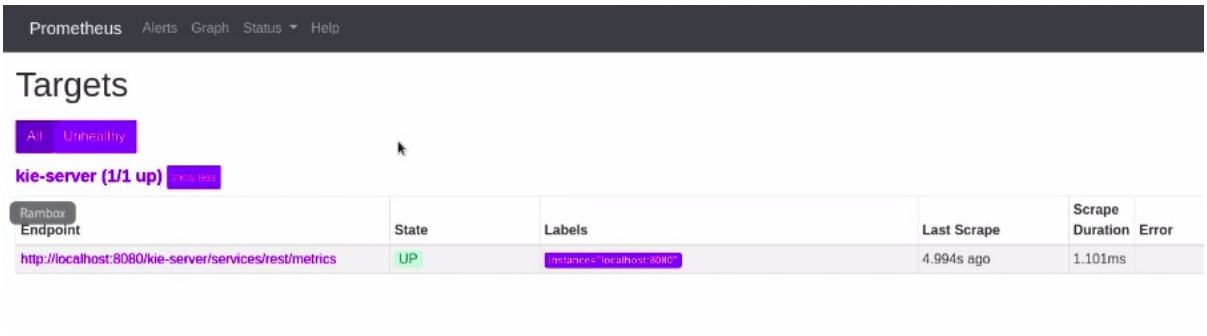
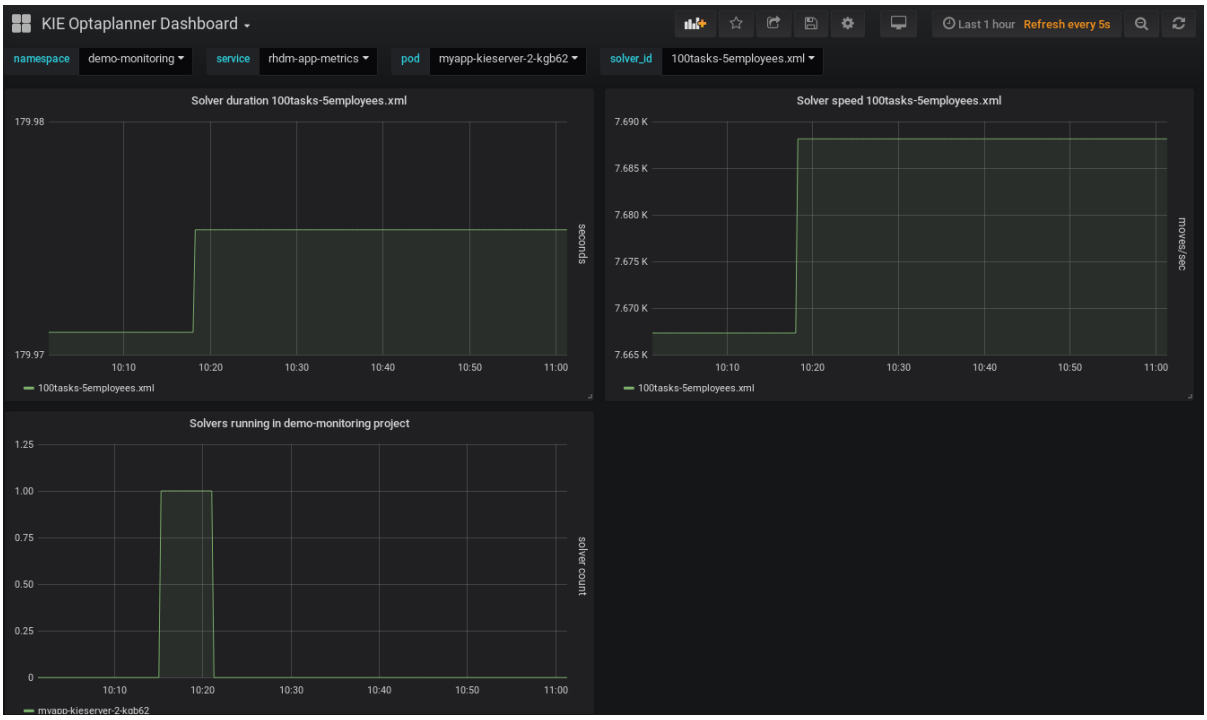


图 11.3. DMN 型号的 KIE 服务器指标的 Grafana 仪表板

图 11.4. 带有临时解决方案的 KIE 服务器指标的 Grafana 仪表板



其他资源

- Prometheus 入门
- Prometheus 的 Grafana 支持
- 在 Grafana 中使用 Prometheus

11.2. 在 RED HAT OPENSHIFT CONTAINER PLATFORM 上为 KIE 服务器配置 PROMETHEUS 指标监控

您可以配置 Red Hat OpenShift Container Platform 上的 KIE 服务器部署，以使用 Prometheus 来收集并存储与 Red Hat Decision Manager 中的业务资产活动相关的指标。有关 KIE 服务器与 Prometheus 公开的可用指标列表，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.2 Source Distribution /kie-server-services-prometheus/src/main/7.13.2 -sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus。https://access.redhat.com/jbossnetwork/restricted/listSoftware.html

先决条件

- KIE 服务器已安装并部署到 Red Hat OpenShift Container Platform 上。如需有关

OpenShift 上的 KIE Server 的更多信息，请参阅 Red Hat Decision Manager 7.13 产品文档中的相关 OpenShift 部署选项。

- 您有 kie-server 用户角色访问权限到 KIE 服务器。
- 已安装 Prometheus Operator。有关下载和使用 Prometheus Operator 的信息，请参阅 GitHub 中的 Prometheus Operator 项目。

流程

1. 在 OpenShift 上的 KIE 服务器部署的 DeploymentConfig 对象中，将 PROMETHEUS_SERVER_EXT_DISABLED 环境变量设置为 false 来启用 Prometheus 扩展。您可以在 OpenShift web 控制台中设置此变量，或者在命令终端中使用 oc 命令：

```
oc set env dc/<dc_name> PROMETHEUS_SERVER_EXT_DISABLED=false -n
<namespace>
```

如果您还没有在 OpenShift 上部署 KIE 服务器，然后在计划用于 OpenShift 部署的 OpenShift 模板中（如 rhpam713-prod-immutable-kieserver.yaml），您可以将 PROMETHEUS_SERVER_EXT_DISABLED 模板参数设置为 false 来启用 Prometheus 扩展。

如果您使用 OpenShift Operator 在 OpenShift 上部署 KIE 服务器，然后在 KIE 服务器配置中，将 PROMETHEUS_SERVER_EXT_DISABLED 环境变量设置为 false 来启用 Prometheus 扩展：

```
apiVersion: app.kiegroup.org/v1
kind: KieApp
metadata:
  name: enable-prometheus
spec:
  environment: rhpam-trial
  objects:
    servers:
    - env:
      - name: PROMETHEUS_SERVER_EXT_DISABLED
        value: "false"
```

2. 创建一个 service-metrics.yaml 文件，以添加将 KIE 服务器的指标公开给 Prometheus 的服务：

```
apiVersion: v1
kind: Service
```

```
metadata:
  annotations:
    description: RHPAM Prometheus metrics exposed
  labels:
    app: myapp-kieserver
    application: myapp-kieserver
    template: myapp-kieserver
    metrics: rhpam
  name: rhpam-app-metrics
spec:
  ports:
    - name: web
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    deploymentConfig: myapp-kieserver
  sessionAffinity: None
  type: ClusterIP
```

3. 在命令终端中，使用 oc 命令将 service-metrics.yaml 文件应用到 OpenShift 部署：

```
oc apply -f service-metrics.yaml
```

4. 创建一个 OpenShift secret，如 metrics-secret，以访问 KIE 服务器上的 Prometheus 指标。secret 必须包含带有 KIE 服务器用户凭证的"username"和"password"元素。如需有关 OpenShift 机密的信息，请参阅《OpenShift *开发人员指南*》中的 机密 章节。

5. 创建一个 service-monitor.yaml 文件来定义 ServiceMonitor 对象。服务监控器可让 Prometheus 连接到 KIE 服务器指标服务。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: rhpam-service-monitor
  labels:
    team: frontend
spec:
  selector:
    matchLabels:
      metrics: rhpam
  endpoints:
    - port: web
      path: /services/rest/metrics
      basicAuth:
        password:
          name: metrics-secret
          key: password
```

```
      username:
        name: metrics-secret
        key: username
```

6. 在命令终端中，使用 oc 命令将 service-monitor.yaml 文件应用到 OpenShift 部署：

```
oc apply -f service-monitor.yaml
```

完成这些配置后，Prometheus 开始收集指标，KIE 服务器会将指标发布到 REST API 端点 http://HOST:PORT/kie-server/services/rest/metrics。

您可以在 http://HOST:PORT/graph 中与 Prometheus 表达式浏览器中收集的指标交互，或者将 Prometheus 数据源与 Grafana 等数据图形工具集成。

Prometheus 表达式浏览器位置 http://HOST:PORT/graph 的主机和端口在安装 Prometheus Operator 时公开 Prometheus Web 控制台的路由中定义。如需有关 OpenShift 路由的信息，请参阅 OpenShift *架构* 文档中的 路由 章节。

图 11.5. 带有 KIE 服务器指标的 Prometheus 表达式浏览器

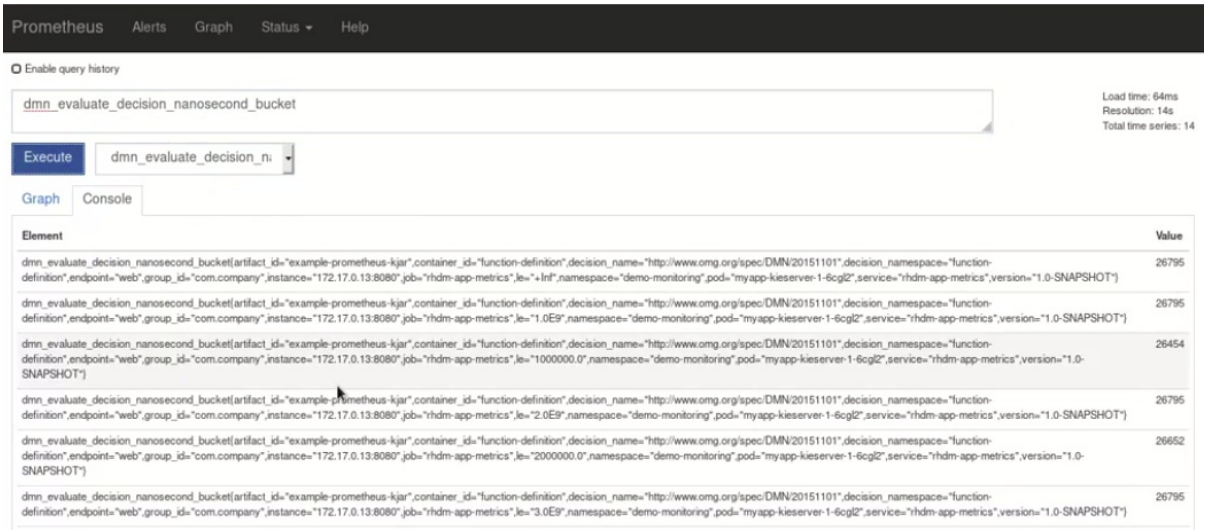


图 11.6. 带有 KIE 服务器目标的 Prometheus 表达式浏览器

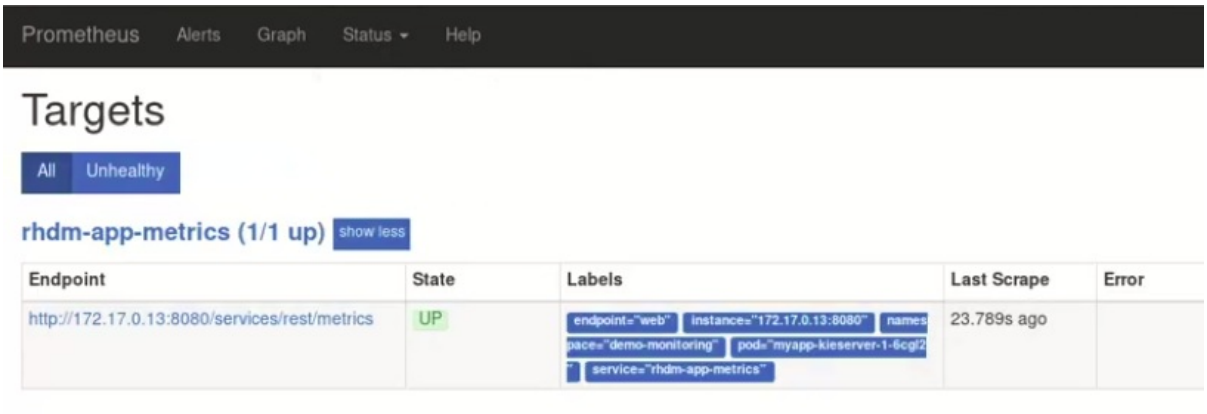
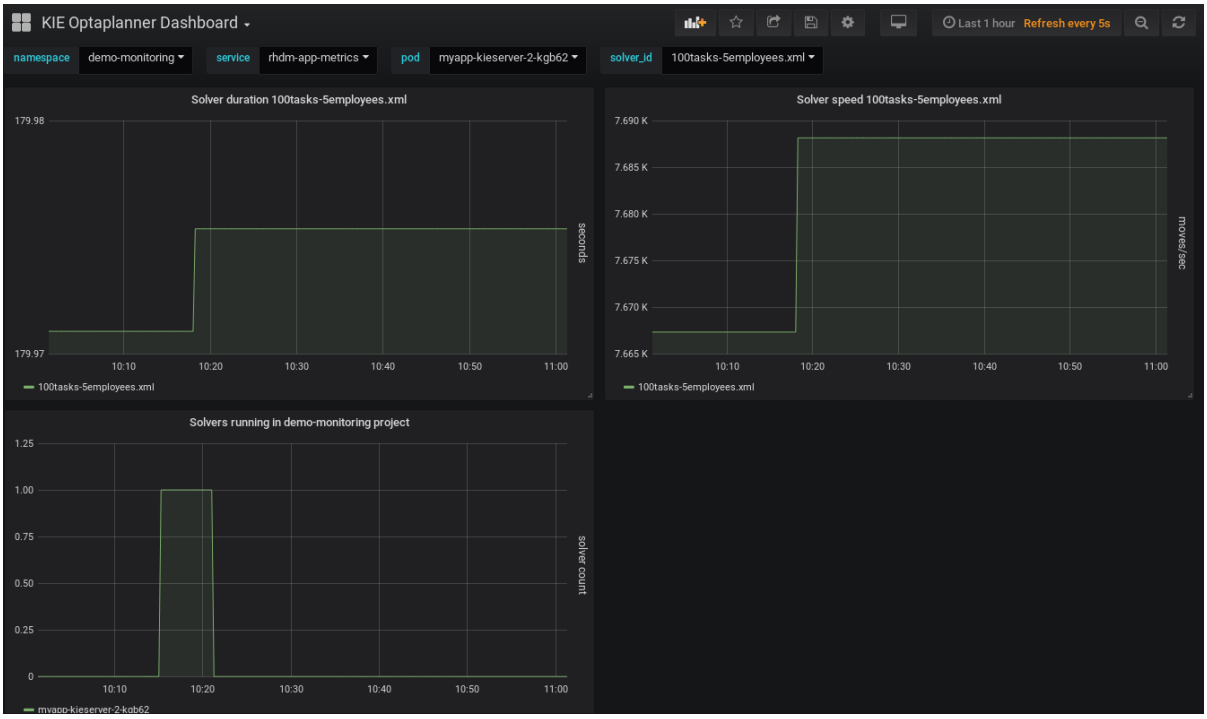


图 11.7. DMN 型号的 KIE 服务器指标的 Grafana 仪表板

图 11.8. 带有临时解决方案的 KIE 服务器指标的 Grafana 仪表板



其他资源

- Prometheus Operator

- Prometheus Operator 入门
- Prometheus RBAC
- Prometheus 的 Grafana 支持
- 在 Grafana 中使用 Prometheus
- Red Hat Decision Manager 7.13 产品文档中的 OpenShift部署选项

11.3. 使用自定义指标在 KIE 服务器中扩展 PROMETHEUS 指标监控

将 KIE 服务器实例配置为使用 Prometheus 指标监控后，您可以将 KIE 服务器中的 Prometheus 功能扩展为使用根据您的业务需求的自定义指标。然后，Prometheus 会收集并存储您的自定义指标以及 KIE 服务器通过 Prometheus 公开的默认指标。

例如，此流程定义由 Prometheus 收集和存储的自定义决策模型和 Notation(DMN)指标。

先决条件

- Prometheus 指标监控是为 KIE 服务器实例配置的。有关内部使用 KIE 服务器 Prometheus 配置的详情，请参考 第 11.1 节 “为 KIE 服务器配置 Prometheus 指标监控”。有关 Red Hat OpenShift Container Platform 上使用 KIE 服务器 Prometheus 配置的详情，请参考 第 11.2 节 “在 Red Hat OpenShift Container Platform 上为 KIE 服务器配置 Prometheus 指标监控”。

流程

1. 创建一个空的 Maven 项目，并在项目的 pom.xml 文件中定义以下打包类型和依赖项：

示例项目中的 pom.xml 文件示例

```
<packaging>jar</packaging>

<properties>
```

```
  <version.org.kie>7.67.0.Final-redhat-00019</version.org.kie>
</properties>

<dependencies>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-common</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-drools</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-prometheus</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-dmn-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-dmn-core</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.jbpm</groupId>
    <artifactId>jbpm-services-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.jbpm</groupId>
    <artifactId>jbpm-executor</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.optaplanner</groupId>
    <artifactId>optaplanner-core</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>io.prometheus</groupId>
```

```
    <artifactId>simpleclient</artifactId>
    <version>0.5.0</version>
  </dependency>
</dependencies>
```

2. 从 org.kie.services.prometheus.PrometheusMetricsProvider 接口实施相关的监听程序，作为定义自定义 Prometheus 指标的自定义监听程序的一部分，如下例所示：

在自定义监听器类中的 DMNRuntimeEventListener 侦听程序实施示例

```
package org.kie.server.ext.prometheus;

import io.prometheus.client.Gauge;
import org.kie.dmn.api.core.ast.DecisionNode;
import org.kie.dmn.api.core.event.AfterEvaluateBKMEvent;
import org.kie.dmn.api.core.event.AfterEvaluateContextEntryEvent;
import org.kie.dmn.api.core.event.AfterEvaluateDecisionEvent;
import org.kie.dmn.api.core.event.AfterEvaluateDecisionServiceEvent;
import org.kie.dmn.api.core.event.AfterEvaluateDecisionTableEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateBKMEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateContextEntryEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateDecisionEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateDecisionServiceEvent;
import org.kie.dmn.api.core.event.BeforeEvaluateDecisionTableEvent;
import org.kie.dmn.api.core.event.DMNRuntimeEventListener;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.services.api.KieContainerInstance;

public class ExampleCustomPrometheusMetricListener implements
DMNRuntimeEventListener {

    private final KieContainerInstance kieContainer;

    private final Gauge randomGauge = Gauge.build()
            .name("random_gauge_nanosecond")
            .help("Random gauge as an example of custom KIE Prometheus metric")
            .labelNames("container_id", "group_id", "artifact_id", "version",
"decision_namespace", "decision_name")
            .register();

    public ExampleCustomPrometheusMetricListener(KieContainerInstance
containerInstance) {
        kieContainer = containerInstance;
    }

    public void beforeEvaluateDecision(BeforeEvaluateDecisionEvent e) {
    }
```

```
    public void afterEvaluateDecision(AfterEvaluateDecisionEvent e) {
        DecisionNode decisionNode = e.getDecision();
        ReleaseId releaseId = kieContainer.getResource().getReleaseId();
        randomGauge.labels(kieContainer.getContainerId(), releaseId.getGroupId(),
                releaseId.getArtifactId(), releaseId.getVersion(),
                decisionNode.getModelName(), decisionNode.getModelNamespace())
            .set((int) (Math.random() * 100));
    }

    public void beforeEvaluateBKM(BeforeEvaluateBKMEvent event) {
    }

    public void afterEvaluateBKM(AfterEvaluateBKMEvent event) {
    }

    public void beforeEvaluateContextEntry(BeforeEvaluateContextEntryEvent event) {
    }

    public void afterEvaluateContextEntry(AfterEvaluateContextEntryEvent event) {
    }

    public void beforeEvaluateDecisionTable(BeforeEvaluateDecisionTableEvent event)
{
    }

    public void afterEvaluateDecisionTable(AfterEvaluateDecisionTableEvent event) {
    }

    public void beforeEvaluateDecisionService(BeforeEvaluateDecisionServiceEvent
event) {
    }

    public void afterEvaluateDecisionService(AfterEvaluateDecisionServiceEvent event)
{
    }
}
```

PrometheusMetricsProvider 接口包含用于收集 Prometheus 指标所需的监听程序。接口由您在项目 pom.xml 文件中声明的 kie-server-services-prometheus 依赖项集成。

在本例中，ExamplePrometheusMetricListener 类实施 DMNRuntimeEventListener 侦听器（从 PrometheusMetricsProvider 接口），并定义要收集和存储的自定义 DMN 指标。

3. 作为自定义指标提供商类的一部分，实施 PrometheusMetricsProvider 接口，将您的自定义监听程序与 PrometheusMetricsProvider 接口相关联，如下例所示：

自定义 metrics 提供者类中的 PrometheusMetricsProvider 接口实施示例

```
package org.kie.server.ext.prometheus;

import org.jbpm.executor.AsynchronousJobListener;
import org.jbpm.services.api.DeploymentEventListener;
import org.kie.api.event.rule.AgendaEventListener;
import org.kie.api.event.rule.DefaultAgendaEventListener;
import org.kie.dmn.api.core.event.DMNRuntimeEventListener;
import org.kie.server.services.api.KieContainerInstance;
import org.kie.server.services.prometheus.PrometheusMetricsProvider;
import org.optaplanner.core.impl.phase.event.PhaseLifecycleListener;
import org.optaplanner.core.impl.phase.event.PhaseLifecycleListenerAdapter;

public class MyPrometheusMetricsProvider implements PrometheusMetricsProvider {

    public DMNRuntimeEventListener
createDMNRuntimeEventListener(KieContainerInstance kContainer) {
        return new ExampleCustomPrometheusMetricListener(kContainer);
    }

    public AgendaEventListener createAgendaEventListener(String kieSessionId,
KieContainerInstance kContainer) {
        return new DefaultAgendaEventListener();
    }

    public PhaseLifecycleListener createPhaseLifecycleListener(String solverId) {
        return new PhaseLifecycleListenerAdapter() {
        };
    }

    public AsynchronousJobListener createAsynchronousJobListener() {
        return null;
    }

    public DeploymentEventListener createDeploymentEventListener() {
        return null;
    }
}
```

在本例中，MyPrometheusMetricsProvider 类实施 PrometheusMetricsProvider 接口，并包含您的自定义 ExampleCustomPrometheusMetricListener 侦听器类。

4. 要使新指标提供程序可供 KIE 服务器发现，请在 文件中创建一个 META-INF/services/org.kie.services.prometheus. PrometheusMetricsProvider 文件，并在文件中

添加完全限定的类名称。在本例中，该文件包含一行 org.kie.server.ext.prometheus.MyPrometheusMetricsProvider。

5. 构建您的项目，并将生成的 JAR 文件复制到项目的 ~/kie-server.war/WEB-INF/lib 目录中。例如，在红帽 JBoss EAP 上，此目录的路径是 *EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib。

   如果要在 Red Hat OpenShift Container Platform 上部署 Red Hat Decision Manager，请创建一个自定义 KIE Server 镜像，并将此 JAR 文件添加到镜像中。有关使用额外 JAR 文件创建自定义 KIE 服务器镜像的更多信息，请参阅使用 *Operator 在 Red Hat OpenShift Container Platform 4 上部署 Red Hat Decision Manager 环境*。

6. 启动 KIE 服务器并将构建的项目部署到正在运行的 KIE 服务器。您可以使用 Business Central 接口或 KIE 服务器 REST API 部署项目（ PUT 请求到 http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}）。

   在正在运行的 KIE 服务器上部署项目后，Prometheus 开始收集指标和 KIE 服务器，将指标发布到 REST API 端点 http://HOST:PORT/SERVER/services/rest/metrics （或者在 Spring Boot 上，或在 Spring Boot 上）发布指标。

第 12 章 配置 OPENSHIFT 连接超时

默认情况下，OpenShift 路由配置为超时超过 30 秒的 HTTP 请求。这可能导致 Business Central 中的会话超时问题，从而导致以下行为：

- "无法完成您的请求。出现以下异常：(TypeError): Cannot read property 'indexOf' of null."
- "无法完成您的请求。出现以下异常：(TypeError): b is null."
- 点击 Business Central 中的项目 或 服务器 链接时会显示一个空白页面。

所有 Business Central 模板都已经包括扩展超时配置。

要在 Business Central OpenShift 路由中配置更长的超时，请在目标路由中添加 haproxy.router.openshift.io/timeout: 60s 注解：

```
- kind: Route
  apiVersion: v1
  id: "$APPLICATION_NAME-rhpamcentr-http"
  metadata:
    name: "$APPLICATION_NAME-rhpamcentr"
    labels:
      application: "$APPLICATION_NAME"
    annotations:
      description: Route for Business Central's http service.
      haproxy.router.openshift.io/timeout: 60s
  spec:
    host: "$BUSINESS_CENTRAL_HOSTNAME_HTTP"
    to:
      name: "$APPLICATION_NAME-rhpamcentr"
```

有关特定于全局路由的超时注解的完整列表，请参阅 OpenShift 文档。

第 13 章 定义 LDAP 登录域

当您将 Red Hat Decision Manager 设置为使用 LDAP 进行身份验证和授权时，定义 LDAP 登录域，因为 Git SSH 身份验证可能会使用另一个安全域。

要定义 LDAP 登录域，请使用 org.uberfire.domain 系统属性。例如，在 Red Hat JBoss Enterprise Application Platform 上，在 standalone.xml 文件中添加此属性，如下所示：

```
<system-properties>
  <!-- other system properties -->
  <property name="org.uberfire.domain" value="LDAPAuth"/>
</system-properties>
```

确保经过身份验证的用户在 LDAP 中关联有适当的角色（管理员、分析师、审阅人员）。

第 14 章 通过 RH-SSO 验证第三方客户端

要使用由 Business Central 或 KIE 服务器（例如 curl、w wget、Web 浏览器或自定义 REST 客户端）提供的不同远程服务，且必须通过 RH-SSO 服务器进行身份验证，并具有有效的令牌来执行请求。要使用远程服务，经过身份验证的用户必须具有以下角色：

- REST - 全部使用 Business Central 远程服务。
- kie-server 使用 KIE 服务器远程服务。

使用 RH-SSO 管理控制台创建这些角色，并将它们分配到使用远程服务的用户。

您的客户端可以使用以下选项之一通过 RH-SSO 验证：

- 基本身份验证（如果客户端支持）
- 基于令牌的身份验证

14.1. 基本身份验证（BASIC AUTHENTICATION）

如果您在 Business Central 和 KIE 服务器的 RH-SSO 客户端适配器配置中启用了基本身份验证，您可以避免令牌授予和刷新调用并调用服务，如下例所示：

- 对于基于 Web 的远程软件仓库端点：

```
curl http://admin:password@localhost:8080/business-central/rest/repositories
```

- 对于 KIE 服务器：

```
curl http://admin:password@localhost:8080/kie-server/services/rest/server/
```

第 15 章 KIE 服务器系统属性

KIE 服务器接受以下系统属性（bootstrap 交换机）来配置服务器的行为：

表 15.1. 禁用 KIE 服务器扩展的系统属性

属性	值	默认	描述
org.drools.server.ext.disabled	**true**,**false**	**false**	如果设为 **true**，则禁用商业规则管理(BRM)支持（例如，规则支持）。
org.optaplanner.server.ext.disabled	**true**,**false**	**false**	如果设置为 **true**，则禁用红帽构建的 OptaPlanner 支持。
org.kie.prometheus.server.ext.disabled	**true**,**false**	**true**	如果设置为 **true**，则禁用 Prometheus 服务器扩展。
org.kie.scenariosimulation.server.ext.disabled	**true**,**false**	**true**	如果设置为 **true**，则禁用测试场景服务器扩展。
org.kie.dmn.server.ext.disabled	**true**,**false**	**false**	如果设置为 **true**，则禁用 KIE Server DMN 支持。
org.kie.swagger.server.ext.disabled	**true**,**false**	**false**	如果设置为 **true**，则禁用 KIE 服务器 swagger 文档支持

注意

下表中列出的一些 Process Automation Manager 控制器属性被标记为必需。在 Business Central 中创建或删除 KIE 服务器容器时，设置这些属性。如果您在没有与 Business Central 交互的情况下单独使用 KIE 服务器，则不需要设置必要的属性。

表 15.2. Process Automation Manager 控制器所需的系统属性

属性	值	默认	描述
org.kie.server.id	字符串	N/A	要分配给服务器的任意 ID。如果在 Business Central 外部配置了无头进程自动化管理器控制器，这是服务器连接到无头进程自动化管理器控制器的 ID，以获取 KIE 容器配置。如果没有提供，则会自动生成 ID。

属性	值	默认	描述
org.kie.server.user	字符串	**kieserver**	在以受管模式运行时，从 Process Automation Manager 控制器与 KIE 服务器连接的用户名需要。在 Business Central 系统属性中设置此属性。在使用 Process Automation Manager 控制器时设置此属性。
org.kie.server.pwd	字符串	**kieserver1!**	在以受管模式运行时，从 Process Automation Manager 控制器连接到 KIE 服务器的密码。在 Business Central 系统属性中设置此属性。在使用 Process Automation Manager 控制器时设置此属性。
org.kie.server.token	字符串	N/A	一个属性，允许您在 Process Automation Manager 控制器和 KIE 服务器间使用基于令牌的身份验证，而不是基本用户名和密码身份验证。Process Automation Manager 控制器将令牌作为请求标头中的参数发送。服务器需要长期提供的访问令牌，因为令牌没有刷新。
org.kie.server.location	URL	N/A	Process Automation Manager 控制器用来在这个服务器上调用的 KIE 服务器实例的 URL，例如 **http://localhost:8230/kie-server/services/rest/server**。使用 Process Automation Manager 控制器时需要设置此属性。
org.kie.server.controller	逗号分隔列表	N/A	Process Automation Manager 控制器 REST 端点的以逗号分隔的 URL 列表，例如 **http://localhost:8080/business-central/rest/controller**。使用 Process Automation Manager 控制器时需要设置此属性。
org.kie.server.controller.user	字符串	**kieserver**	连接到 Process Automation Manager 控制器 REST API 的用户名。使用 Process Automation Manager 控制器时需要设置此属性。
org.kie.server.controller.pwd	字符串	**kieserver1!**	连接到 Process Automation Manager 控制器 REST API 的密码。使用 Process Automation Manager 控制器时需要设置此属性。
org.kie.server.controller.token	字符串	N/A	一个属性，允许您在 KIE 服务器和 Process Automation Manager 控制器之间使用基于令牌的身份验证，而不是基本用户名和密码身份验证。服务器在请求标头中将令牌作为参数发送。服务器需要长期提供的访问令牌，因为令牌没有刷新。

属性	值	默认	描述
org.kie.server.controller.connect	Long	**10000**	服务器启动时，重复尝试将 KIE 服务器连接到 Process Automation Manager 控制器之间的等待时间（毫秒）。

表 15.3. 加载密钥存储的系统属性

属性	值	默认	描述
kie.keystore.keyStoreURL	URL	N/A	URL 用于加载 Java Cryptography Extension KeyStore(JCEKS)。例如，**file:///home/kie/keystores/keystore.jceks**.
kie.keystore.keyStorePwd	字符串	N/A	密码用于 JCEKS。
kie.keystore.key.server.alias	字符串	N/A	存储密码的 REST 服务的密钥名称。
kie.keystore.key.server.pwd	字符串	N/A	REST 服务别名的密码。
kie.keystore.key.ctrl.alias	字符串	N/A	默认 REST Process Automation Manager 控制器的密钥别名。
kie.keystore.key.ctrl.pwd	字符串	N/A	默认 REST Process Automation Manager 控制器的别名密码。

表 15.4. 重试提交事务的系统属性

属性	值	默认	描述
org.kie.optlock.retries	整数	5	此属性描述了进程引擎永久重试事务的次数。
org.kie.optlock.delay	整数	50	第一次重试前的延迟时间（以毫秒为单位）。
org.kie.optlock.delayFactor	整数	4	每次后续重试增加延迟时间的倍数。使用默认值时，进程引擎在第一个重试前等待 50 毫秒，在第二个重试前等待 200 毫秒，即第三个重试前 800 毫秒，以此类推。

表 15.5. 其他系统属性

属性	值	默认	描述
kie.maven.settings.custom	路径	N/A	Maven 配置的自定义 **settings.xml** 文件的位置。
kie.server.jms.queues.response	字符串	队列/**KIE.SERVER.RESPONSE**	JMS 的响应队列 JNDI 名称。
org.drools.server.filter.classes	**true**,**false**	**false**	当设置为 **true** 时，CRI KIE 服务器扩展接受由 **XmlRootElement** 或 **Remotable** 注解标注的自定义类。
org.kie.server.domain	字符串	N/A	用于使用 JMS 验证用户身份的 JAAS **LoginContext** 域。
org.kie.server.repo	路径	**.**	存储 KIE 服务器状态文件的位置。
org.kie.server.sync.deploy	**true**,**false**	**false**	指示 KIE 服务器存放部署的属性，直到 Process Automation Manager 控制器提供容器部署配置。此属性仅影响在受管模式下运行的服务器。可用的选项如下： ACTIVE **false** ：与 Process Automation Manager 控制器的连接是异步的。应用程序启动，连接到 Process Automation Manager 控制器，并成功后部署容器。即使容器可用前，应用程序也会接受请求。ACTIVE **true** ：服务器应用程序的部署会加入 Process Automation Manager 控制器连接线程，并带有主部署并等待其完成。这个选项可能会导致在更多应用程序位于同一服务器上时潜在的死锁。在一个服务器实例中只使用一个应用。
org.kie.server.startup.strategy	**ControllerBasedStartupStrategy**, **LocalContainersStartupStrategy**	**ControllerBasedStartupStrategy**	KIE 服务器的启动策略，用于控制部署的 KIE 容器以及部署它们的顺序。

属性	值	默认	描述
org.kie.server.mgmt.api.disabled	**true**,**false**	**false**	当设置为 **true** 时，禁用 KIE 服务器管理 API。
org.kie.server.xstream.enabled.packages	Java 软件包，如 **org.kie.example**。您还可以指定通配符表达式，如 **org.kie.example.***。	N/A	一个属性，用于指定允许使用 XStream 进行总结的附加软件包。
org.kie.store.services.class	字符串	**org.drools.persistence.jpa.KnowledgeStoreServiceImpl**	实施 **KieStoreServices** 的类的完全限定名称，负责引导 KieSession 实例。
org.kie.server.strict.id.format	**true**,**false**	**false**	使用 JSON 总结时，如果属性设置为 **true**，它始终会以正确的 JSON 格式返回响应。例如，如果原始响应仅包含一个数字，则响应将以 JSON 格式嵌套。例如，**{"value" : 1}**。
org.kie.server.json.customObjectDeserializerCNFEBehavior	**忽略**、**警告**、**异常**	**IGNORE**	在使用 JSON unmarshalling 时，当找不到有效负载中的类时，可以使用此属性更改此行为，如下所示： ● 如果属性设置为 **IGNORE**，则有效负载将转换为 HashMap ● 如果属性设为 **WARN**，则有效负载将转换为 HashMap，并记录警告 ● 如果属性设为 **EXCEPTION**，则 KIE 服务器会抛出异常
org.kie.server.strict.jaxb.format	**true**,**false**	**false**	当此属性的值设置为 **true** 时，KIE 服务器在 REST API 有效负载中验证数据类型。例如，如果数据字段具有数字数据类型并包含数字以外的内容，您将收到错误。

第 16 章 KIE 服务器功能和扩展

KIE 服务器的功能由插件扩展决定，您可以启用、禁用或进一步扩展以满足您的业务需求。KIE 服务器支持以下默认功能和扩展：

表 16.1. KIE 服务器功能和扩展

功能名称	扩展名称	描述
KieServer	**KieServer**	提供 KIE 服务器的核心功能，如在服务器实例上创建和处理 KIE 容器
BRM	**grafana**	提供商业规则管理(BRM)功能，如插入事实和执行业务规则
BRP	**OptaPlanner**	提供业务资源规划(BRP)功能，比如实施解决者
DMN	**DMN**	提供 Decision Model 和 Notation(DMN)功能，如管理 DMN 数据类型和执行 DMN 模型
Swagger	**Swagger**	提供 Swagger web-interface 功能与 KIE 服务器 REST API 交互

要查看正在运行的 KIE 服务器实例的支持的扩展，请将 GET 请求发送到以下 REST API 端点，并查看 XML 或 JSON 服务器响应：

KIE 服务器信息的 GET 请求的基本 URL

```
http://SERVER:PORT/kie-server/services/rest/server
```

使用 KIE 服务器信息的 JSON 响应示例

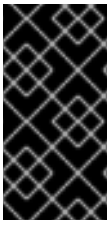
```
{
  "type": "SUCCESS",
  "msg": "Kie Server info",
  "result": {
    "kie-server-info": {
      "id": "test-kie-server",
      "version": "7.67.0.20190818-050814",
      "name": "test-kie-server",
      "location": "http://localhost:8080/kie-server/services/rest/server",
```

```
    "capabilities": [
      "KieServer",
      "BRM",
      "BRP",
      "DMN",
      "Swagger"
    ],
    "messages": [
      {
        "severity": "INFO",
        "timestamp": {
          "java.util.Date": 1566169865791
        },
        "content": [
          "Server KieServerInfo{serverId='test-kie-server', version='7.67.0.20190818-050814',
name='test-kie-server', location='http:/localhost:8080/kie-server/services/rest/server',
capabilities=[KieServer, BRM, BRP, DMN, Swagger]', messages=null',
mode=DEVELOPMENT}started successfully at Sun Aug 18 23:11:05 UTC 2019"
        ]
      }
    ],
    "mode": "DEVELOPMENT"
  }
 }
}
```


要启用或禁用 KIE 服务器扩展，请配置相关的 *.server.ext.disabled KIE 服务器系统属性。例如，要禁用 BRM 功能，请设置系统属性 org.drools.server.ext.disabled=true。有关所有 KIE 服务器系统属性，请参阅 *第 15 章 KIE 服务器系统属性*。

默认情况下，KIE 服务器扩展通过 REST 或 JMS 数据传输提供，并使用预定义的客户端 API。您可以使用其他 REST 端点扩展现有 KIE 服务器功能，将支持的传输方法扩展到 REST 或 JMS 外，或者在 KIE 服务器客户端中扩展功能。

KIE 服务器功能中的这一灵活性使您能够根据业务需求调整 KIE 服务器实例，而不必为默认 KIE 服务器功能适应您的业务需求。



重要

如果您扩展了 KIE 服务器功能，红帽不支持用作自定义实现和扩展一部分的自定义代码。

16.1. 使用自定义 REST API 端点扩展现有 KIE 服务器功能

KIE 服务器 REST API 可让您与红帽决策管理器中的 KIE 容器和商业资产（如业务规则、流程和解决方案）进行交互，而无需使用 Business Central 用户界面。可用的 REST 端点由 KIE 服务器系统属性中启用了的功能决定（例如，/ org.drools.server.ext.disabled=false 表示 ECDSA 功能）。您可以使用自定义 REST API 端点扩展现有 KIE 服务器功能，以进一步调整 KIE 服务器 REST API 以满足您的业务需求。

例如，这个过程使用以下自定义 REST API 端点扩展了 Drools KIE 服务器扩展（用于 BRM 功能）：

自定义 REST API 端点示例

```
/server/containers/instances/{containerId}/ksession/{ksessionId}
```

这个示例自定义端点接受插入到决策引擎工作内存的列表，自动执行所有规则，并从指定的 KIE 容器中的 KIE 会话检索所有对象。

流程

1. 创建一个空的 Maven 项目，并在项目的 pom.xml 文件中定义以下打包类型和依赖项：

示例项目中的 pom.xml 文件示例

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.67.0.Final-redhat-00019</version.org.kie>
</properties>

<dependencies>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-internal</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
```

```
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-common</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-drools</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-rest-common</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
  </dependency>
</dependencies>
```

2. 在项目中的 Java 类中实施 org.kie.server.services.api.KieServerApplicationComponentsService 接口，如下例所示：

KieServerApplicationComponentsService 接口的实现示例

```
public class CusomtDroolsKieServerApplicationComponentsService implements
KieServerApplicationComponentsService { 1

  private static final String OWNER_EXTENSION = "Drools"; 2
```

```
    public Collection<Object> getAppComponents(String extension,
SupportedTransports type, Object... services) { (3)
        // Do not accept calls from extensions other than the owner extension:
        if ( !OWNER_EXTENSION.equals(extension) ) {
            return Collections.emptyList();
        }

        RulesExecutionService rulesExecutionService = null; (4)
        KieServerRegistry context = null;

        for( Object object : services ) {
            if( RulesExecutionService.class.isAssignableFrom(object.getClass()) ) {
                rulesExecutionService = (RulesExecutionService) object;
                continue;
            } else if( KieServerRegistry.class.isAssignableFrom(object.getClass()) ) {
                context = (KieServerRegistry) object;
                continue;
            }
        }

        List<Object> components = new ArrayList<Object>(1);
        if( SupportedTransports.REST.equals(type) ) {
            components.add(new CustomResource(rulesExecutionService, context)); (5)
        }

        return components;
    }

}
```

(1)

向应用程序启动时部署的 KIE 服务器基础架构提供 REST 端点。

(2)

指定您要扩展的扩展，如本例中的 Drools 扩展。

(3)

返回 REST 容器必须部署的所有资源。在 KIE 服务器实例中启用的每个扩展会调用 getAppComponents 方法，因此 if（!OWNER_EXTENSION.equals(extension)） 调用为除指定的 OWNER_EXTENSION 扩展以外的扩展返回空集合。

(4)

列出您要使用的指定扩展中的服务，如本例中的 Drools 扩展中的 RulesExecutionService 和 KieServerRegistry 服务。

5

指定扩展的传输类型（即 REST ）（本例中为REST ），以及返回资源作为 组件 列表一部分的 CustomResource 类。

3. 实施 KIE 服务器可以用来为新的 REST 资源提供额外的功能，如下例所示：

CustomResource 类的实现示例

```
// Custom base endpoint:
@Path("server/containers/instances/{containerId}/ksession")
public class CustomResource {

    private static final Logger logger =
LoggerFactory.getLogger(CustomResource.class);

    private KieCommands commandsFactory =
KieServices.Factory.get().getCommands();

    private RulesExecutionService rulesExecutionService;
    private KieServerRegistry registry;

    public CustomResource() {

    }

    public CustomResource(RulesExecutionService rulesExecutionService,
KieServerRegistry registry) {
        this.rulesExecutionService = rulesExecutionService;
        this.registry = registry;
    }

    // Supported HTTP method, path parameters, and data formats:
    @POST
    @Path("/{ksessionId}")
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Response insertFireReturn(@Context HttpHeaders headers,
            @PathParam("containerId") String id,
            @PathParam("ksessionId") String ksessionId,
            String cmdPayload) {

        Variant v = getVariant(headers);
        String contentType = getContentType(headers);

        // Marshalling behavior and supported actions:
        MarshallingFormat format = MarshallingFormat.fromType(contentType);
        if (format == null) {
            format = MarshallingFormat.valueOf(contentType);
```

```
        }
        try {
            KieContainerInstance kci = registry.getContainer(id);

            Marshaller marshaller = kci.getMarshaller(format);

            List<?> listOfFacts = marshaller.unmarshall(cmdPayload, List.class);

            List<Command<?>> commands = new ArrayList<Command<?>>();
            BatchExecutionCommand executionCommand =
commandsFactory.newBatchExecution(commands, ksessionId);

            for (Object fact : listOfFacts) {
                commands.add(commandsFactory.newInsert(fact, fact.toString()));
            }
            commands.add(commandsFactory.newFireAllRules());
            commands.add(commandsFactory.newGetObjects());

            ExecutionResults results = rulesExecutionService.call(kci,
executionCommand);

            String result = marshaller.marshall(results);


            logger.debug("Returning OK response with content '{}'", result);
            return createResponse(result, v, Response.Status.OK);
        } catch (Exception e) {
            // If marshalling fails, return the `call-container` response to maintain backward
compatibility:
            String response = "Execution failed with error : " + e.getMessage();
            logger.debug("Returning Failure response with content '{}'", response);
            return createResponse(response, v,
Response.Status.INTERNAL_SERVER_ERROR);
        }

    }
}
```

在本例中，自定义端点的 CustomResource 类指定以下数据和行为：

- 使用基本端点 server/containers/instances/{containerId}/ksession
- 使用 POST HTTP 方法

- 预期在 REST 请求中给出了以下数据：
  - containerId 作为路径参数
  - ksessionId 作为路径参数
  - 将事实列为消息有效负载
- 支持所有 KIE 服务器数据格式：
  - XML（JAXB、XStream）
  - JSON
- Unmarshals 在 List<?& gt; 集合中，为列表中的每个项目创建一个 InsertCommand 实例，后跟 FireAllRules 和 GetObject 命令。
- 将所有命令添加到调用决策引擎的 BatchExecutionCommand 实例中。

4. 要使新端点可以被 KIE 服务器发现，请在 文件中创建一个 META-INF/services/org.kie.server.api.KieServerApplicationApplicationComponentsService 文件，并添加 KieApplicationApplicationComponentsService 实施类的完全限定域名。在本例中，该文件包含一行 org.kie.server.ext.drools.rest.CusomtDroolsKieServerComponentsService。

5. 构建您的项目，并将生成的 JAR 文件复制到项目的 ~/kie-server.war/WEB-INF/lib 目录中。例如，在红帽 JBoss EAP 上，此目录的路径是 *EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib。

6. 启动 KIE 服务器并将构建的项目部署到正在运行的 KIE 服务器中。您可以使用 Business Central 接口或 KIE 服务器 REST API 部署项目（ PUT 请求到 http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}）。

在某个正在运行的 KIE 服务器上部署项目后，您可以开始与新的 REST 端点交互。

在本例中，您可以使用以下信息来调用新端点：

- 请求 URL 示例： http://localhost:8080/kie-server/services/rest/server/containers/instances/demo/ksession/defaultKieSession
- HTTP 方法： POST
- HTTP 标头：
  - content-Type: application/json
  - accept: application/json
- 消息有效负载示例：

```
[
  {
    "org.jbpm.test.Person": {
      "name": "john",
      "age": 25
    }
  },
  {
    "org.jbpm.test.Person": {
      "name": "mary",
      "age": 22
    }
  }
]
```

- 服务器响应示例： 200 (success)
- 服务器日志输出示例：

```
13:37:20,347 INFO  [stdout] (default task-24) Hello mary
13:37:20,348 INFO  [stdout] (default task-24) Hello john
```

16.2. 扩展 KIE 服务器以使用自定义数据传输

默认情况下，KIE 服务器扩展通过 REST 或 JMS 数据传输提供。您可以扩展 KIE 服务器以支持自定义数据传输，以根据您的业务需求调整 KIE 服务器传输协议。

例如，这个步骤将自定义数据传输添加到使用 Drools 扩展的 KIE 服务器中，该扩展基于 Apache MINA，它是一个开源 Java 网络应用程序框架。示例自定义 MINA 传输基于字符串的数据，这些数据依赖于现有的 marshalling 操作，且只支持 JSON 格式。

流程

1. 创建一个空的 Maven 项目，并在项目的 pom.xml 文件中定义以下打包类型和依赖项：

示例项目中的 pom.xml 文件示例

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.67.0.Final-redhat-00019</version.org.kie>
</properties>

<dependencies>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-internal</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-common</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
```

```
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-services-drools</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
  </dependency>
  <dependency>
    <groupId>org.apache.mina</groupId>
    <artifactId>mina-core</artifactId>
    <version>2.1.3</version>
  </dependency>
</dependencies>
```

2. 在项目中的 Java 类中实施 org.kie.server.services.api.KieServerExtension 接口，如下例所示：

KieServerExtension 接口的实现示例

```
public class MinaDroolsKieServerExtension implements KieServerExtension {

    private static final Logger logger =
LoggerFactory.getLogger(MinaDroolsKieServerExtension.class);

    public static final String EXTENSION_NAME = "Drools-Mina";

    private static final Boolean disabled =
Boolean.parseBoolean(System.getProperty("org.kie.server.drools-mina.ext.disabled",
"false"));
    private static final String MINA_HOST = System.getProperty("org.kie.server.drools-
mina.ext.port", "localhost");
    private static final int MINA_PORT =
Integer.parseInt(System.getProperty("org.kie.server.drools-mina.ext.port", "9123"));

    // Taken from dependency on the `Drools` extension:
```

```
    private KieContainerCommandService batchCommandService;

    // Specific to MINA:
    private IoAcceptor acceptor;

    public boolean isActive() {
        return disabled == false;
    }

    public void init(KieServerImpl kieServer, KieServerRegistry registry) {

        KieServerExtension droolsExtension = registry.getServerExtension("Drools");
        if (droolsExtension == null) {
            logger.warn("No Drools extension available, quitting...");
            return;
        }

        List<Object> droolsServices = droolsExtension.getServices();
        for( Object object : droolsServices ) {
            // If the given service is null (not configured), continue to the next service:
            if (object == null) {
                continue;
            }
            if( KieContainerCommandService.class.isAssignableFrom(object.getClass()) ) {
                batchCommandService = (KieContainerCommandService) object;
                continue;
            }
        }
        if (batchCommandService != null) {
            acceptor = new NioSocketAcceptor();
            acceptor.getFilterChain().addLast( "codec", new ProtocolCodecFilter( new
TextLineCodecFactory( Charset.forName( "UTF-8" ))));

            acceptor.setHandler( new TextBasedIoHandlerAdapter(batchCommandService)
);
            acceptor.getSessionConfig().setReadBufferSize( 2048 );
            acceptor.getSessionConfig().setIdleTime( IdleStatus.BOTH_IDLE, 10 );
            try {
                acceptor.bind( new InetSocketAddress(MINA_HOST, MINA_PORT) );

                logger.info("{} -- Mina server started at {} and port {}", toString(),
MINA_HOST, MINA_PORT);
            } catch (IOException e) {
                logger.error("Unable to start Mina acceptor due to {}", e.getMessage(), e);
            }

        }
    }

    public void destroy(KieServerImpl kieServer, KieServerRegistry registry) {
        if (acceptor != null) {
            acceptor.dispose();
            acceptor = null;
        }
        logger.info("{} -- Mina server stopped", toString());
    }
```

```
    public void createContainer(String id, KieContainerInstance kieContainerInstance,
Map<String, Object> parameters) {
        // Empty, already handled by the `Drools` extension

    }

    public void disposeContainer(String id, KieContainerInstance kieContainerInstance,
Map<String, Object> parameters) {
      // Empty, already handled by the `Drools` extension

    }

    public List<Object> getAppComponents(SupportedTransports type) {
        // Nothing for supported transports (REST or JMS)
        return Collections.emptyList();
    }

    public <T> T getAppComponents(Class<T> serviceType) {

        return null;
    }

    public String getImplementedCapability() {
        return "BRM-Mina";
    }

    public List<Object> getServices() {
        return Collections.emptyList();
    }

    public String getExtensionName() {
        return EXTENSION_NAME;
    }

    public Integer getStartOrder() {
        return 20;
    }

    @Override
    public String toString() {
        return EXTENSION_NAME + " KIE Server extension";
    }
}
```

KieServerExtension 接口是 KIE 服务器可以用来为新的 MINA 传输提供额外功能的主扩展接口。接口由以下组件组成：

KieServerExtension 接口概述

```
public interface KieServerExtension {

    boolean isActive();

    void init(KieServerImpl kieServer, KieServerRegistry registry);

    void destroy(KieServerImpl kieServer, KieServerRegistry registry);

    void createContainer(String id, KieContainerInstance kieContainerInstance,
Map<String, Object> parameters);

    void disposeContainer(String id, KieContainerInstance kieContainerInstance,
Map<String, Object> parameters);

    List<Object> getAppComponents(SupportedTransports type);

    <T> T getAppComponents(Class<T> serviceType);

    String getImplementedCapability(); (1)

    List<Object> getServices();

    String getExtensionName(); (2)

    Integer getStartOrder(); (3)
}
```

(1) 指定此扩展涵盖的能力。能力在 KIE 服务器中必须是唯一的。

(2) 为扩展名定义人类可读的名称。

(3) 确定何时应启动指定的扩展。对于其他扩展具有依赖项的扩展，此设置不得与父设置冲突。例如，在这个示例中，这个自定义扩展依赖于 Drools 扩展，它的 StartOrder 设置为 0，因此这个自定义附加组件扩展必须大于 0（示例实现中的设置为 20）。

在前面的 MinaDroolsKieServerExtension 示例实现中，init 方法是从 Drools 扩展收集服务的主要元素，用于引导 MINA 服务器。KieServerExtension 接口中的其他方法都可以与标准

实施保持一致，以满足接口要求。

TextBasedIoHandlerAdapter 类是响应传入请求的 MINA 服务器上的处理程序。

3. 为 MINA 服务器实施 text BasedIoHandlerAdapter 处理程序，如下例所示：

TextBasedIoHandlerAdapter 处理程序的实现示例

```
public class TextBasedIoHandlerAdapter extends IoHandlerAdapter {

   private static final Logger logger =
LoggerFactory.getLogger(TextBasedIoHandlerAdapter.class);

   private KieContainerCommandService batchCommandService;

   public TextBasedIoHandlerAdapter(KieContainerCommandService
batchCommandService) {
      this.batchCommandService = batchCommandService;
   }

   @Override
   public void messageReceived( IoSession session, Object message ) throws
Exception {
      String completeMessage = message.toString();
      logger.debug("Received message '{}'", completeMessage);
      if( completeMessage.trim().equalsIgnoreCase("quit") ||
completeMessage.trim().equalsIgnoreCase("exit") ) {
         session.close(false);
         return;
      }

      String[] elements = completeMessage.split("\\|");
      logger.debug("Container id {}", elements[0]);
      try {
         ServiceResponse<String> result =
batchCommandService.callContainer(elements[0], elements[1],
MarshallingFormat.JSON, null);

         if (result.getType().equals(ServiceResponse.ResponseType.SUCCESS)) {
            session.write(result.getResult());
            logger.debug("Successful message written with content '{}'",
result.getResult());
         } else {
            session.write(result.getMsg());
            logger.debug("Failure message written with content '{}'", result.getMsg());
         }
      } catch (Exception e) {
```

```
        }
    }
}
```

在本例中，处理器类接收文本信息，并在 Drools 服务中执行它们。

在使用 TextBasedIoHandlerAdapter 处理程序实现时，请考虑以下处理器要求和行为：

- 您提交到处理程序的任何对象都必须是一行，因为每个传入的传输请求都是一行。
- 您必须在此一行中传递 KIE 容器 ID，以便处理程序需要格式 containerID|payload。
- 您可以使用 marshaller 生成的方式设置响应。响应可以是多行。
- 该处理程序支持 *流模式*，允许您发送命令而不与 KIE 服务器会话断开连接。要在流模式下结束 KIE 服务器会话，请将 exit 或 exit 命令发送至服务器。

4. 要使新数据传输可被 KIE 服务器发现，请在 文件中创建一个 META-INF/services/org.kie.server.api. KieServerExtension 文件，并在该文件中添加完全限定的类名称。在本例中，该文件包含一行 org.kie.server.ext.mina.MinastandaloneKieServerExtension。

5. 构建您的项目，并将生成的 JAR 文件复制到项目的 mina-core-2.0.9.jar 文件（此示例中取决于该扩展）到项目的 ~/kie-server.war/WEB-INF/lib 目录中。例如，在红帽 JBoss EAP 上，此目录的路径是 *EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib。

6. 启动 KIE 服务器并将构建的项目部署到正在运行的 KIE 服务器。您可以使用 Business Central 接口或 KIE 服务器 REST API 部署项目（ PUT 请求到 http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}）。

在正在运行的 KIE 服务器上部署项目后，您可以在 KIE 服务器日志中查看新数据传输的状态，然后开始使用您的新数据传输：

服务器日志中的新数据传输

```
Drools-Mina KIE Server extension -- Mina server started at localhost and port 9123
Drools-Mina KIE Server extension has been successfully registered as server extension
```

在本例中，您可以使用 Telnet 与 KIE 服务器中的新的基于 MINA 的数据传输进行交互：

在命令终端中启动 Telnet 并连接到端口 9123 上的 KIE 服务器

```
telnet 127.0.0.1 9123
```

在命令终端中与 KIE 服务器交互示例

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

# Request body:
demo|{"lookup":"defaultKieSession","commands":[{"insert":{"object":
{"org.jbpm.test.Person":{"name":"john","age":25}}}},{"fire-all-rules":""}]}

# Server response:
{
  "results" : [ {
    "key" : "",
    "value" : 1
  } ],
  "facts" : [ ]
}

demo|{"lookup":"defaultKieSession","commands":[{"insert":{"object":
{"org.jbpm.test.Person":{"name":"mary","age":22}}}},{"fire-all-rules":""}]}
{
  "results" : [ {
    "key" : "",
    "value" : 1
  } ],
  "facts" : [ ]
}
```

```
demo|{"lookup":"defaultKieSession","commands":[{"insert":{"object":
{"org.jbpm.test.Person":{"name":"james","age":25}}}},{"fire-all-rules":""}]}
{
  "results" : [ {
    "key" : "",
    "value" : 1
  } ],
  "facts" : [ ]
}
exit
Connection closed by foreign host.
```

服务器日志输出示例

```
16:33:40,206 INFO  [stdout] (NioProcessor-2) Hello john
16:34:03,877 INFO  [stdout] (NioProcessor-2) Hello mary
16:34:19,800 INFO  [stdout] (NioProcessor-2) Hello james
```

16.3. 使用自定义客户端 API 扩展 KIE 服务器客户端

KIE 服务器使用预定义的客户端 API，您可以与之交互以使用 KIE 服务器服务。您可以使用自定义客户端 API 扩展 KIE 服务器客户端，以满足您的业务需求。

例如，这个步骤将自定义客户端 API 添加到 KIE 服务器，以适应自定义数据传输（之前为本情景）基于 Apache MINA（开源 Java 网络应用程序框架）。

流程

1. 创建一个空的 Maven 项目，并在项目的 pom.xml 文件中定义以下打包类型和依赖项：

示例项目中的 pom.xml 文件示例

```
<packaging>jar</packaging>
```

```
<properties>
  <version.org.kie>7.67.0.Final-redhat-00019</version.org.kie>
 </properties>

 <dependencies>
  <dependency>
    <groupId>org.kie.server</groupId>
    <artifactId>kie-server-api</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
  <dependency>
     <groupId>org.kie.server</groupId>
     <artifactId>kie-server-client</artifactId>
     <version>${version.org.kie}</version>
   </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${version.org.kie}</version>
  </dependency>
</dependencies>
```

2. 在项目中的 Java 类中实施相关的 ServicesClient 接口，如下例所示：

RulesMinaServicesClient 接口示例

```
public interface RulesMinaServicesClient extends RuleServicesClient {

}
```

需要特定的接口，因为您必须基于接口注册客户端实施，而且只能有一个给定接口的实施。

在本例中，基于自定义 MINA 的数据传输使用 Drools 扩展，因此本例 RulesMinaServicesClient 接口扩展了 Drools 扩展中现有的 RuleServicesClient 客户端 API。

3. 实施 KIE 服务器可以用来为新的 MINA 传输提供额外客户端功能的 RulesMinaServicesClient 接口，如下例所示：

RulesMinaServicesClient 接口的实现示例

```
public class RulesMinaServicesClientImpl implements RulesMinaServicesClient {

    private String host;
    private Integer port;

    private Marshaller marshaller;

    public RulesMinaServicesClientImpl(KieServicesConfiguration configuration,
ClassLoader classloader) {
        String[] serverDetails = configuration.getServerUrl().split(":");

        this.host = serverDetails[0];
        this.port = Integer.parseInt(serverDetails[1]);

        this.marshaller =
MarshallerFactory.getMarshaller(configuration.getExtraJaxbClasses(),
MarshallingFormat.JSON, classloader);
    }

    public ServiceResponse<String> executeCommands(String id, String payload) {

        try {
            String response = sendReceive(id, payload);
            if (response.startsWith("{")) {
                return new ServiceResponse<String>(ResponseType.SUCCESS, null,
response);
            } else {
                return new ServiceResponse<String>(ResponseType.FAILURE, response);
            }
        } catch (Exception e) {
            throw new KieServicesException("Unable to send request to KIE Server", e);
        }
    }

    public ServiceResponse<String> executeCommands(String id, Command<?> cmd) {
        try {
            String response = sendReceive(id, marshaller.marshall(cmd));
            if (response.startsWith("{")) {
                return new ServiceResponse<String>(ResponseType.SUCCESS, null,
response);
            } else {
                return new ServiceResponse<String>(ResponseType.FAILURE, response);
            }
        } catch (Exception e) {
            throw new KieServicesException("Unable to send request to KIE Server", e);
        }
    }

    protected String sendReceive(String containerId, String content) throws Exception {

        // Flatten the content to be single line:
```

```
        content = content.replaceAll("\\n", "");

        Socket minaSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;

        StringBuffer data = new StringBuffer();
        try {
            minaSocket = new Socket(host, port);
            out = new PrintWriter(minaSocket.getOutputStream(), true);
            in = new BufferedReader(new
InputStreamReader(minaSocket.getInputStream()));

            // Prepare and send data:
            out.println(containerId + "|" + content);
            // Wait for the first line:
            data.append(in.readLine());
            // Continue as long as data is available:
            while (in.ready()) {
                data.append(in.readLine());
            }

            return data.toString();
        } finally {
            out.close();
            in.close();
            minaSocket.close();
        }
    }
}
```

这个实现示例指定了以下数据和行为：

- 使用基于套接字的通讯进行简单性
- 依赖于 KIE 服务器客户端的默认配置，并使用 ServerUrl 提供 MINA 服务器的主机和端口
- 将 JSON 指定为 marshalling 格式
- 需要接收消息作为以 open bracket {开头的 JSON 对象

- 在等待响应的第一行时，使用直接套接字与阻塞 API 通信，然后读取所有可用的行
- 不要使用 *流模式*，因此在调用命令后断开 KIE 服务器会话

4. 在项目中的 Java 类中实施 org.kie.client.client.helper.KieServicesClientBuilder 接口，如下例所示：

KieServicesClientBuilder 接口的实现示例

```
public class MinaClientBuilderImpl implements KieServicesClientBuilder { (1)

  public String getImplementedCapability() { (2)
    return "BRM-Mina";
  }

  public Map<Class<?>, Object> build(KieServicesConfiguration configuration,
ClassLoader classLoader) { (3)
    Map<Class<?>, Object> services = new HashMap<Class<?>, Object>();

    services.put(RulesMinaServicesClient.class, new
RulesMinaServicesClientImpl(configuration, classLoader));

    return services;
  }

}
```

(1) 可让您向通用 KIE 服务器客户端基础架构提供额外的客户端 API

(2) 定义客户端使用的 KIE 服务器功能(extension)

(3) 提供客户端实施的映射，其中键是接口，值是完全初始化的实现

5. 要使新客户端 API 发现 KIE 服务器客户端，请在 文件中创建一个 META-INF/services/org.kie.client.client.helper. KieServicesClientBuilder 文件，并在文件中添加完全限定的类名称。在本例中，该文件包含一行 org.kie.server.ext.mina.client.MinaClientBuilderImpl。

6. 构建您的项目，并将生成的 JAR 文件复制到项目的 ~/kie-server.war/WEB-INF/lib 目录中。例如，在红帽 JBoss EAP 上，此目录的路径是 *EAP_HOME*/standalone/deployments/kie-server.war/WEB-INF/lib。

7. 启动 KIE 服务器并将构建的项目部署到正在运行的 KIE 服务器中。您可以使用 Business Central 接口或 KIE 服务器 REST API 部署项目（ PUT 请求到 http://SERVER:PORT/kie-server/services/rest/server/containers/{containerId}）。

在某个正在运行的 KIE 服务器上部署项目后，您可以开始与新的 KIE 服务器客户端交互。您可以采用与标准 KIE 服务器客户端相同的方法，创建客户端配置和客户端实例，并通过类型检索服务客户端，以及调用客户端方法。

在本例中，您可以创建一个 RulesMinaServiceClient 客户端实例，并通过 MINA 传输调用 KIE 服务器上的操作：

创建 RulesMinaServiceClient 客户端的实现示例

```
protected RulesMinaServicesClient buildClient() {
    KieServicesConfiguration configuration =
KieServicesFactory.newRestConfiguration("localhost:9123", null, null);
    List<String> capabilities = new ArrayList<String>();
    // Explicitly add capabilities (the MINA client does not respond to `get-server-info`
requests):
    capabilities.add("BRM-Mina");

    configuration.setCapabilities(capabilities);
    configuration.setMarshallingFormat(MarshallingFormat.JSON);

    configuration.addJaxbClasses(extraClasses);

    KieServicesClient kieServicesClient =
KieServicesFactory.newKieServicesClient(configuration);

    RulesMinaServicesClient rulesClient =
kieServicesClient.getServicesClient(RulesMinaServicesClient.class);

    return rulesClient;
}
```

通过 MINA 传输调用 KIE 服务器操作的示例配置

```
RulesMinaServicesClient rulesClient = buildClient();

List<Command<?>> commands = new ArrayList<Command<?>>();
BatchExecutionCommand executionCommand =
commandsFactory.newBatchExecution(commands, "defaultKieSession");

Person person = new Person();
person.setName("mary");
commands.add(commandsFactory.newInsert(person, "person"));
commands.add(commandsFactory.newFireAllRules("fired"));

ServiceResponse<String> response = rulesClient.executeCommands(containerId,
executionCommand);
Assert.assertNotNull(response);

Assert.assertEquals(ResponseType.SUCCESS, response.getType());

String data = response.getResult();

Marshaller marshaller = MarshallerFactory.getMarshaller(extraClasses,
MarshallingFormat.JSON, this.getClass().getClassLoader());

ExecutionResultImpl results = marshaller.unmarshall(data,
ExecutionResultImpl.class);
Assert.assertNotNull(results);

Object personResult = results.getValue("person");
Assert.assertTrue(personResult instanceof Person);

Assert.assertEquals("mary", ((Person) personResult).getName());
Assert.assertEquals("JBoss Community", ((Person) personResult).getAddress());
Assert.assertEquals(true, ((Person) personResult).isRegistered());
```

第 17 章 KIE 服务器的性能调整注意事项

以下关键概念或建议做法可以帮助您优化 KIE 服务器性能。这些概念在本章节中进行了概述，在适用的情况下，会详细介绍相关文档。本节将根据需要在 Red Hat Decision Manager 的新版本时扩展或更改。

确保在开发过程中启用了开发模式

您可以将 Business Central 中的 KIE 服务器或特定项目设置为使用 生产 模式或 开发模式。默认情况下，Business Central 中的 KIE 服务器和所有新项目均为开发模式。这个模式提供有助于开发体验的功能，如灵活的项目部署策略以及优化开发期间 KIE 服务器性能的功能，如禁用的 GAV 检测。使用开发模式，直到红帽决策管理器环境建立并完全为生产模式做好准备。

有关配置环境模式或重复的 GAV 检测的详情，请查看以下资源：

- *第 7 章 在 KIE 服务器和 Business Central 中配置环境模式*
- *打包并部署 Red Hat Decision Manager 项目*

根据您的需要调整 KIE 服务器的功能和扩展

KIE 服务器的功能由插件扩展决定，您可以启用、禁用或进一步扩展以满足您的业务需求。默认情况下，KIE 服务器扩展通过 REST 或 JMS 数据传输提供，并使用预定义的客户端 API。您可以使用其他 REST 端点扩展现有 KIE 服务器功能，将支持的传输方法扩展到 REST 或 JMS 外，或者在 KIE 服务器客户端中扩展功能。

KIE 服务器功能中的这一灵活性使您能够根据业务需求调整 KIE 服务器实例，而不必为默认 KIE 服务器功能适应您的业务需求。

有关启用、禁用或扩展 KIE 服务器功能的详情，请参考 *第 16 章 KIE 服务器功能和扩展*。

第 18 章 其他资源

- *在 Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Decision Manager*
- *规划 Red Hat Decision Manager 安装*
- *在 Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Decision Manager*
- *使用 Operator 在 Red Hat OpenShift Container Platform 4 上部署 Red Hat Decision Manager 环境*
- *使用模板在 Red Hat OpenShift Container Platform 3 上部署 Red Hat Decision Manager 环境*

部分 II. 配置 BUSINESS CENTRAL 设置和属性

作为管理员，您可以在 admin Settings 页面中自定义以下内容：

- 角色 ：设置角色的主页、优先级和权限。
- 组 ：设置组的主页、优先级和权限，以及创建和删除组。
- 用户 ：创建和删除用户，添加或删除用户，以及查看用户权限。
- 工件 ：查看 M2 存储库工件、上传工件、查看和下载 JAR 文件。
- 数据源 ：添加、更新或删除数据源和数据库驱动程序。
- 数据集 ：创建、修改或删除数据集。
- 项目 ：查看和编辑项目首选项，如文件导出属性、空间属性、默认值和高级 GAV 属性。
- 工件存储库 ：管理工件存储库属性。
- 语言 ：设置 Business Central 语言。
- 进程管理 ：设置 Business Central 中的默认分页选项。
- 进程设计器 ：设置图表编辑器属性。
- SSH Keys ：添加或删除 SSH 密钥。

- 自定义任务管理任务 ：启用或禁用默认服务任务，并上传自定义服务任务。
- Dashbuilder Data Transfer ：导入并导出 Dashbuilder 数据作为 Business Central 中的 ZIP 文件。
- Profile ：将 workbench 配置集设置为 Planner 和 Rules 或 Full。
- archetypes : View, add, validate, set as default, and delete the archetypes.在 Business Central 中创建新项目时用作模板。

先决条件

- 安装了 Red Hat JBoss Enterprise Application Platform 7.4.1。如需更多信息，请参阅 *Red Hat JBoss Enterprise Application Platform 7.4 安装指南*。
- Red Hat Decision Manager 已安装并运行。如需更多信息，*请参阅在 Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Decision Manager*。
- 使用 admin 用户角色登录到 Business Central。

第 19 章 用户和组群管理

Business Central 为安全管理定义了三种类型的实体：用户、组和角色。您可以为角色和组分配权限。您可以在 Business Central 中分配以下角色：

- process-admin
- Manager
- admin
- 分析师
- rest-all
- 开发人员
- rest-project
- user

注意

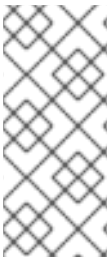
应用程序角色 Registry 中的用户角色具有角色标识符，而用户组则不行。

根据需要，使用 Business Central 创建和管理任意数量的用户和组。必须将一个用户分配到至少一个特定于用户的角色才能登录到 Business Central。用户特权取决于用户所属组和角色的权限。请注意，如果用户分配了多个角色或组，则角色或组优先级将被视为。

19.1. 创建用户

用户特权和设置由分配给用户的角色以及用户所属的组控制。您可以在 Business Central 中创建任意

数量的用户。



注意

不要在进程引擎或 KIE Server 中创建一个名为 unknown 的用户。未知 用户帐户是具有超级用户访问权限的保留系统名称。当用户没有登录时，未知 用户帐户执行与 SLA 违反监听程序相关的任务。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 用户。
2. 单击 New user，输入用户名，然后单击 Next。
3. 若要为用户分配角色，请单击 Roles 选项卡，单击 Add Roles，选择所需角色，然后单击 Add to selected roles。
4. 可选： 要为用户分配组，点 Groups 选项卡，点 Add to groups，选择所需组，然后点 Add to selected groups。
5. 点 Create。
6. 单击 Yes 为用户设置密码，输入所需的密码，然后单击 更改。

注意

用户必须至少有一个角色才能访问 Business Central。

19.2. 编辑用户

您可以使用 Business Central Settings 页面中的 Users 选项更改用户的组群和角色。所有用户权限均基于用户的组和角色权限。您可以从 Permissions 选项卡中查看用户权限。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 用户。

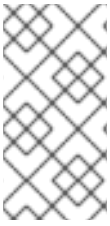
2. 在 All users 列表中点击您要编辑的用户。用户详情显示在右侧窗格中。

3. 点 Edit 执行以下任务：

- 要更改用户组，点 Groups 选项卡，点 Add to groups，选择您希望用户所属的组，点 Add to selected groups，然后点 Save。
- 要更改用户的角色，请单击 Roles 选项卡，单击 Add roles，选择您要分配给用户的角色，单击 Add to selected roles，然后单击 Save。
- 要查看用户权限，请点击 权限选项卡 并展开属性。
- 要更改密码，请单击" 更改密码"，输入新密码，然后单击" 更改 "。
- 若要删除用户，请单击 Delete，然后单击 Yes 以确认删除。

19.3. 创建组

在 Business Central 中，您可以使用组来控制用户集合的权限。您可以根据需要创建多个组，但组必须至少有一个用户。



注意

如果 Business Central 使用红帽单点登录(RH-SSO)，则 Business Central 中的组是从 RH-SSO 中的角色创建的。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 组。

2. 单击 **New group**，输入组名称，然后单击 **Next**。

3. 选择您要添加到此组的用户，然后点 **Add selected users**。

   新创建的组列在 **All groups** 下。

19.4. 编辑组

您可以根据要求编辑组的属性，如主页、优先级和权限。在 **Business Central Settings** 页面中的 **Groups** 选项，您可以修改或删除组。

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **组**。

2. 在 **All groups** 列表中点击您要编辑的组。用户详情显示在右侧窗格中。

3. 从 **Home Page** 列表中选择主页。

4. 从优先级列表中选择 **优先级**。

5. 在 **Permissions** 部分中，展开 **resource** 属性并更改其权限。

   注意

   您可为 **页面**、**编辑器**、**空格和** 项目权限 **添加例外**。

6. 点击 **Save** 以应用更改。

第 20 章 安全管理

安全管理是管理用户、组和权限的过程。您可以从 Business Central Security 管理页面控制对 Business Central 资源的访问和功能。

Business Central 为安全管理定义了三种类型的实体：用户、组和角色。您可以为角色和组分配权限。用户从用户所属的组和角色继承权限。

20.1. 安全管理供应商

在安全管理上下文中，域限制访问不同的应用资源。realm 包含有关用户、组、角色和权限的信息。特定域的 concrete 用户和组管理服务实施称为安全管理提供程序。

如果内置的安全管理提供程序没有满足应用程序安全性域的要求，您可以构建和注册自己的安全管理供应商。

注意

如果没有安装安全管理提供程序，则管理安全域的用户界面将不可用。在安装和配置安全管理提供程序后，用户和组管理功能会在安全管理用户界面中自动启用。

Business Central 包括红帽 JBoss EAP 安全管理提供商，它支持基于 application-users.properties 或 application-roles.properties 属性文件的内容的域类型。

20.1.1. 根据属性文件配置红帽 JBoss EAP 安全管理提供程序

您可以构建和注册您自己的红帽 JBoss EAP 安全管理供应商。要基于属性文件使用红帽 JBoss EAP 安全管理提供程序，请完成以下步骤。

先决条件

- 安装了红帽 JBoss EAP。

流程

1. 要使用红帽 JBoss EAP 实例中的现有用户或角色属性，请在

EAP_HOME/standalone/configuration/application-users.properties 和 EAP_HOME/standalone/application-roles.properties 文件中包含以下系统属性，如下例所示：

```
<property name="org.uberfire.ext.security.management.wildfly.properties.realm"
value="ApplicationRealm"/>
<property name="org.uberfire.ext.security.management.wildfly.properties.users-file-path"
value="/standalone/configuration/application-users.properties"/>
<property name="org.uberfire.ext.security.management.wildfly.properties.groups-file-path"
value="/standalone/configuration/application-roles.properties"/>
```

下表提供了这些属性的描述和默认值：

表 20.1. 基于属性文件的红帽 JBoss EAP 安全管理供应商

属性	描述	默认值
org.uberfire.ext.security.management.wildfly.properties.realm	域的名称。这个属性不是必须的。	**ApplicationRealm**
org.uberfire.ext.security.management.wildfly.properties.users-file-path	用户属性文件的绝对路径。这个属性是必需的。	**./standalone/configuration/application-users.properties**
org.uberfire.ext.security.management.wildfly.properties.groups-file-path	groups 属性文件的绝对文件路径。这个属性是必需的。	**./standalone/configuration/application-roles.properties**

2. 在应用程序的根目录中创建 **security-management.properties** 文件。例如，创建以下文件：

```
src/main/resources/security-management.properties
```

3. 在 **security-management.properties** 文件中输入以下系统属性和安全供应商名称作为值：

```
<property name="org.uberfire.ext.security.management.api.userManagementServices"
value="WildflyUserManagementService"/>
```

20.1.2. 根据属性文件和 CLI 模式配置红帽 JBoss EAP 安全管理供应商

要基于属性文件和 CLI 模式使用红帽 JBoss EAP 安全管理提供程序，请完成以下步骤。

先决条件

- 安装了红帽 JBoss EAP。

流程

1. 要使用红帽 JBoss EAP 实例中的现有用户或角色属性，请在 EAP_HOME/standalone/configuration/application-users.properties 和 EAP_HOME/standalone/application-roles.properties 文件中包含以下系统属性，如下例所示：

```
<property name="org.uberfire.ext.security.management.wildfly.cli.host" value="localhost"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.port" value="9990"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.user" value="
<USERNAME>"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.password" value="
<USER_PWD>"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.realm"
value="ApplicationRealm"/>
```

下表提供了这些属性的描述和默认值：

表 20.2. 基于属性文件和 CLI 模式的红帽 JBoss EAP 安全管理供应商

属性	描述	默认值
org.uberfire.ext.security.management.wildfly.cli.host	原生管理接口主机。	**localhost**
org.uberfire.ext.security.management.wildfly.cli.port	原生管理接口端口。	**9990**
org.uberfire.ext.security.management.wildfly.cli.user	原生管理接口用户名。	不适用
org.uberfire.ext.security.management.wildfly.cli.password	原生管理接口用户的密码。	不适用
org.uberfire.ext.security.management.wildfly.cli.realm	供应用的安全上下文使用的域。	**ApplicationRealm**

2. 在应用程序的根目录中创建 security-management.properties 文件。例如，创建以下文件：

```
src/main/resources/security-management.properties
```

3. 在 security-management.properties 文件中输入以下系统属性和安全供应商名称作为值：

```
<property name="org.uberfire.ext.security.management.api.userManagementServices"
value="WildflyCLIUserManagementService"/>
```

20.2. 权限和设置

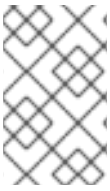
权限是授权用户，用于执行与应用程序内特定资源相关的操作。例如，用户可以具有以下权限：

- 查看页面。
- 保存项目。
- 查看存储库。
- 删除仪表板

您可以授予或拒绝权限，并且权限可以特定于全局或资源。您可以使用权限来保护资源的访问，并自定义应用程序中的功能。

20.2.1. 在 Business Central 中更改组和角色的权限

在 Business Central 中，您无法更改个人用户的权限。但是，您可以更改组和角色的权限。更改的权限适用于具有角色或属于您更改的组的用户。



注意

对角色或组进行的任何更改会影响与该角色或组关联的所有用户。

先决条件

- 使用 admin 用户角色登录到 Business Central。

流程

1. 要访问 **Business Central** 中的 **安全管理** 页面，可选择屏幕右上角的 **Admin** 图标。

2. 单击 **Business Central Settings** 页面中的 **Roles**、**Groups** 或 **Users**。

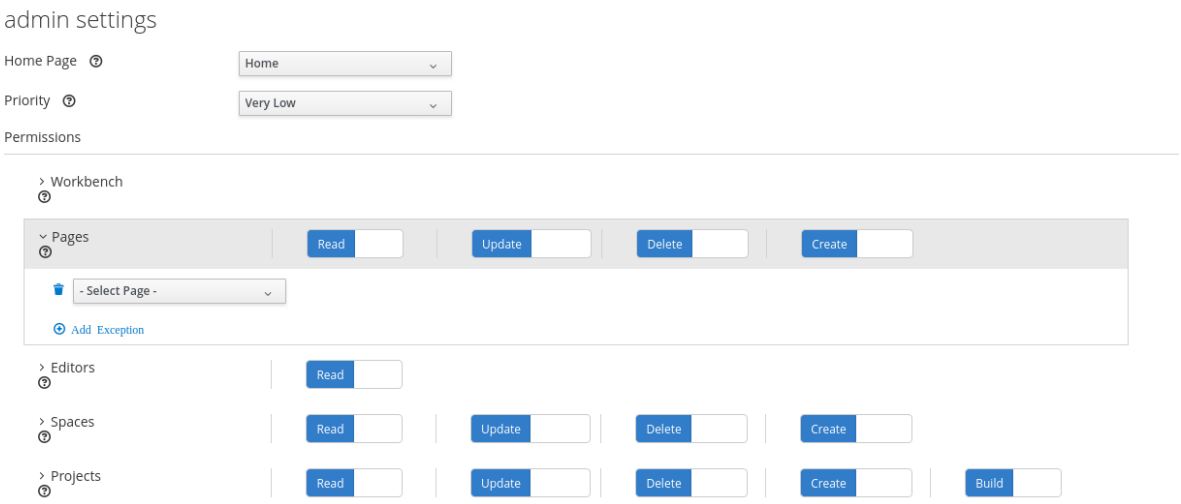
   Security 管理页面 会在您点击的图标的标签页中打开。

3. 在列表中点击您要编辑的角色或组。所有详情都显示在右侧面板中。

4. 在 **Settings** 部分下设置 主页 或 优先级。

5. 在 **Permissions** 部分设置 **Business Central**、页面、编辑器、空格和项目权限。

图 20.1. 设置权限



6. 单击资源类型旁边的箭头，以展开您要更改权限的资源类型。

7. 可选： 要添加资源类型的异常，请单击 **Add Exception**，然后根据需要设置权限。

注意

您不能在 **Business Central** 资源类型中添加例外。

8. 点击 Save。

20.2.2. 更改 Business Central 主页

主页是您登录 Business Central 后出现的页面。默认情况下，主页设置为 Home。您可以为每个角色和组指定不同的主页。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Roles 或 Groups。
2. 选择角色或组。
3. 从 Home Page 列表中选择一个页面。
4. 点击 Save。

注意

角色或组必须具有对页面的读访问权限，然后才能使它成为主页。

20.2.3. 设置优先级

用户可以具有多个角色，并属于多个组。Priority 设置决定了角色或组的优先级顺序。

先决条件

- 使用 admin 用户角色登录到 Business Central。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Roles 或 Groups。

2. 选择**角色**或组。

3. **从优先级菜单中选择优先级，然后单击 Save。**

注意

如果用户具有具有冲突设置的角色或属于某一组，则应用具有最高优先级的角色或组的设置。

第 21 章 工件管理

您可以在 Business Central 中管理 Artifacts 页面中的工件。工件仓库是一个本地 Maven 存储库，每个安装只有一个 Maven 存储库。Business Central 建议使用 Maven 存储库解决方案，如 *Sonatype Nexus™*、*Apache Archiva™* 或 *JFrog Artifactory™*。

Artifacts 页面 列出了 Maven 存储库中的所有工件。您可以将工件上传到 Maven 存储库。

注意

您只能将 JAR、KJAR 和 pom.xml 文件上传到 Artifacts 存储库。

21.1. 查看工件

您可以在 Artifacts 页面中查看 local maven 存储库 的所有内容。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Artifacts。
2. 点 Open 查看工件详情。
3. 单击 Ok 以返回 Artifacts 页面。

21.2. 下载工件

您可以从 Business Central 存储库下载工件并将其保存到项目的本地存储中。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Artifacts。
2. 点 Download。

3. 浏览要保存工件的目录。

4. 点击 **Save**。

21.3. 上传工件

您可以将工件从本地存储上传到 Business Central 中的项目。

流程

1. 在 Business Central 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Artifacts**。

2. 点 **Upload**。

3. 点 **Choose File** 并浏览到您要上传工件的目录。

4. 点 **Upload**。

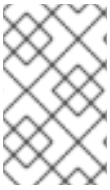
注意

如果您使用非 Maven 工件，首先使用 **mvn deploy** 命令将工件部署到 Maven 存储库，然后刷新 Business Central 中的构件列表。

第 22 章 数据源和数据库驱动程序管理

Business Central 提供数据源管理功能，允许您定义用于访问数据库的数据源。然后，其他 Business Central 组件（如数据集）会使用这些数据源。数据库驱动程序启用数据源和目标数据库之间的通信。

在 Data Source Authoring 页面中，您可以将数据源和数据库驱动程序添加到 Business Central 中。



注意

Business Central 提供可以使用的默认数据源，但无法编辑或删除。

22.1. 添加数据源

您可以从 Data Sources Authoring 页面向 Business Central 添加新数据源。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Data Sources。
2. 在 DataSource Explorer 窗格中，点 Add DataSource。

   New data source 窗口将打开。
3. 在 New data source 窗口中，输入数据源的" 名称 "、"连接 URL、用户 "、"密码" 和"驱动程序"字段。
4. 单击 Test Connection，以验证与数据库的连接。
5. 点 Finish。

22.2. 编辑数据源

您可以编辑数据源的属性，并测试其与 Business Central 中的数据库的连接。

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Data Sources**。
2. 在 **DataSource Explorer** 窗格中，点击您要编辑的数据源。
3. 在 **Data Source Definition** 窗格中，对 **Name**、**Connection URL**、**User**、**Password** 和 **Driver** 字段进行必要的更改。
4. 单击 **Test Connection**，以验证与数据库的连接。
5. 点 **Update**。
6. 点击 **Save**。

22.3. 删除数据源

您可以从 **Business Central** 中的 **DataSource Explorer** 窗格删除现有数据源。

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Data Sources**。
2. 在 **DataSource Explorer** 窗格中，点击您要删除的数据源。
3. 单击 **Remove**。
4. 单击 **Delete** 以确认删除数据源。

22.4. 添加数据库驱动程序

您可以在 Business Central 中添加新数据库驱动程序。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Data Sources。
2. 在 DataSource Explorer 窗格中，点 Add Driver。

   New driver 窗口将打开。
3. 在 New driver 窗口中，输入名称 ,Driver Class Name 、Group Id、Artifact Id 和 Version 字段。
4. 点 Finish。

22.5. 编辑数据库驱动程序

您可以从 Driver Definition 窗格编辑数据库驱动程序的属性。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Data Sources。
2. 在 DataSource Explorer 窗格中，选择您要编辑的驱动程序。
3. 在 Driver Definition 窗格中，对 Name、Driver Class Name 、组 Id、Artifact Id 和 Version 字段进行必要的更改。
4. 点 Update。
5. 点 是。

22.6. 删除数据库驱动程序

您可以从 Business Central 的 Data Source Definition 窗格中删除数据库驱动程序。

流程

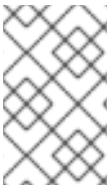
1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Data Sources。
2. 在 DataSource Explorer 窗格中，选择您要删除的驱动程序。
3. 单击 Remove。
4. 点击 Delete。

第 23 章 数据集编写

数据集是相关信息的集合，可以存储在数据库中、Microsoft Excel 文件中的或内存中。数据集定义指示 Business Central 方法访问、读取和解析数据集。Business Central 不会存储数据。它允许您定义数据集的访问权限，无论数据存储位置。

例如，如果数据存储在数据库中，则有效数据集可包含整个数据库，或作为 SQL 查询结果的数据库子集。在这两种情况下，数据都用作报告业务中心组件的输入，然后显示该信息。

要访问数据集，您必须创建并注册数据集定义。数据集定义指定数据集的位置、访问它的选项、对其进行解析以及其中包含的列。



注意

Data Sets 页面仅对具有 admin 角色的用户可见。

23.1. 添加数据集

您可以创建数据集，以从外部数据源获取数据，并将该数据用于报告组件。

流程

1. 在 Business Central 中，前往 Admin → Data Sets。

   这时将打开 Data Sets 页面。

2. 点 New Data Set 并选择以下供应商类型之一：

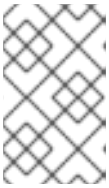
   - bean： 从 Java 类生成数据集
   - CSV： 从远程或本地 CSV 文件中生成数据集
   - SQL： 从 ANSI-SQL 兼容数据库生成数据集

- **Elastic Search：** 从 Elastic Search 节点生成数据集
- **Prometheus：** 使用 Prometheus 查询生成数据集
- **Kafka：** 使用 Kafka 代理、消费者或制作者中的指标生成数据集

注意

您必须为 **Prometheus**、**Kafka** 和 **Execution Server** 选项配置 **KIE** 服务器。

3. 完成 **Data Set Creation Wizard** 并点 **Test**。



注意

配置步骤根据您选择的供应商的不同而有所不同。

4. 点击 **Save**。

23.2. 编辑数据集

您可以编辑现有的数据集，以确保获取到报告组件的数据为最新版本。

流程

1. 在 **Business Central** 中，前往 **Admin** → **Data Sets**。

   这时将打开 **Data Set Explorer** 页面。

2. 在 **Data Set Explorer** 窗格中，搜索您要编辑的数据集合，选择数据集，然后单击 **Edit**。

3. 在 **Data Set Editor** 窗格中，根据需要使用适当的标签页编辑数据。该选项卡会根据您选择的

数据集供应商类型而有所不同。

例如，以下更改可用于编辑 CSV 数据供应商：

- CSV 配置： 使您能够更改数据集定义的名称、源文件、分隔符和其他属性。
- preview ： 允许您预览数据。在 CSV Configuration 选项卡中点 Test 后，系统会执行数据集查找调用，如果数据可用，会出现一个预览。请注意， Preview 选项卡有两个子选项卡：
  - data 列： 允许您指定数据设置定义中的哪些列。
  - filter： 允许您添加新过滤器。
- 高级： 使您能够管理以下配置：
  - 缓存： 请参阅缓存数据 以了解更多信息。
  - 通过 缓存生命周期， 您可以指定数据设置（或数据）被刷新的时间间隔。当 后端数据 改变时， 刷新缓存的数据功能。

4. 进行必要的更改后， 单击 Validate。

5. 点击 Save。

23.3. 数据刷新

通过数据刷新功能， 您可以指定数据设置（或数据）被刷新的时间间隔。您可以在数据集 的高级标签页中访问数据刷新每个 功能。当 后端数据 改变时， 刷新缓存的数据功能。

23.4. 缓存数据

Business Central 提供使用内存数据存储数据收集和执行数据操作的缓存机制。缓存数据减少了网络

流量、远程系统有效负载和处理时间。为避免性能问题，请在 Business Central 中配置缓存设置。

对于生成数据集合的任何数据查找调用，缓存方法决定执行数据查找调用的位置，以及保存生成的数据集的位置。例如，数据查找调用的示例将是本地参数设置为"Urban"的所有影片应用程序。

Business Central 数据集功能提供了两个缓存级别：

- 客户端级别
- 后端级别

您可以在数据集 的高级 标签页上设置客户端缓存和后端缓存设置。

客户端缓存

当缓存打开后，数据集会在查找操作期间在 Web 浏览器中缓存，并进一步查找操作不会对后端执行请求。在 Web 浏览器中处理数据设置操作，如分组、聚合、过滤和排序。仅在数据集大小小时启用客户端缓存，例如：数据的大小小于 10 MB 的数据集。对于大型数据集，会出现浏览器问题，如性能缓慢或间歇的空闲状态。客户端缓存可减少包括对存储系统的请求的请求数量。

后端缓存

启用缓存后，决定引擎缓存数据集。这可减少到远程存储系统的后端请求数量。所有数据收集操作都是使用内存数据在决策引擎中执行的。仅在数据集大小没有频繁更新时启用后端缓存，并可在内存中存储和处理。在对远程存储出现低延迟连接问题的情况下，使用后端缓存也很有用。

注意

在 Data Set Editor 的 Advanced 选项卡中，后端缓存设置并不总是可见，因为 Java 和 CSV 数据供应商依赖于后端缓存（必须位于内存中的数据设置）以使用内存决策引擎来解决任何数据查找操作。

第 24 章 ARCHETYPE 管理

Business Central 提供了一个 archetype 管理功能，可让您列出、添加、验证、默认设置为 archetypes。您可以在 Business Central 的 Archetypes 页面中管理 archetypes。archetypes 是在 Apache Maven 存储库中安装的项目，您可以根据需要设置或创建模板结构。

有关 archetypes 的最新和详细信息，请查看 Archetypes 页面简介。

24.1. 列出 ARCHETYPES

Archetypes 页面列出了 Business Central 中添加的所有架构类型。此列表提供有关 组 ID、Artifact ID 、版本、创建日期、Status 以及 archetype 操作 的详细信息。

先决条件

- 您已创建了 archetype，并在 maven 存储库的 Business Central Settings 中列出它。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Archetypes。

   在 Status 列中，绿色图标指示它是一个有效的 archetype，红色图标表示它是一个无效的架构架构，而 blue 图标则表示对应的 archetype 是新空格的默认 archetype。

24.2. 添加 ARCHETYPE

您可以向 Business Central 添加新的 archetype。

先决条件

- 您已在 Maven 存储库中安装了 archetype。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Archetypes。

2. 单击 **Add Archetype**。

3. 在 **Add Archetype** 面板中，分别在 **Group ID**、**Artifact ID** 和 **Version** 字段中输入 GAV 属性。

4. 点击 **Add**。

Business Central 验证新添加的 **archetype** 并使它可用作空格中的模板。

24.3. 管理 ARCHETYPE 的附加功能

您可以删除、设置默认并验证 **Business Central** 中 **Archetypes** 页面中的 **archetypes**。

先决条件

- 您已创建 **archetype** 并列在 **Maven** 存储库的 **Business Central Settings** 中。

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Archetypes**。

2. 在 **Actions** 列中，点 **archetype** 右侧的

图标。

   - 从下拉菜单中选择 **Delete**，从列表中删除 **archetype**。
   - 从下拉菜单中选择 **Validate** 以验证 **archetype** 是否有效。

注意

启动 **Business Central** 后，会自动验证所有已注册的 **archetypes**。

- 从下拉菜单中选择 **Set as default**，将 archetype 设置为新空格的默认值。

24.4. 使用 ARCHETYPES 创建项目

您可以使用 archetypes 在 Business Central 中创建项目。当您在 Business Central 中创建项目时，它会被添加到连接到红帽决策管理器安装的 Git 存储库中。

先决条件

- 您已创建了 archetype，并在 Maven 存储库中的 Business Central Settings 中列出它。
- 您已在 Business Central 中将 archetype 设置为默认值。

流程

1. 在 Business Central 中，转至 **Menu** → **Design** → **Projects**。
2. 从 archetype 模板选择或创建要向其添加新项目的空间。
3. 单击 **Add Project**。
4. 在 **Name** 和 **Description** 字段中分别键入项目名称和描述。
5. 点 **Configure Advanced Options**。
6. 选择 **基于模板** 复选框。
7. 如果需要，从下拉列表中选择 archetype。

   默认 archetype 已在空间中设置。

8. 点击 **Add**。

项目的 资产 视图基于所选 **archetype** 模板打开。

24.5. 使用 BUSINESS CENTRAL 中的空间设置来管理 ARCHETYPES

将 **archetypes** 添加到 **Business Central** 时，您可以在所有空格中使用它们作为模板。您可以从 **Settings** 选项卡中管理所有 **archetypes**，该选项卡可在空间中可用。此选项卡仅对具有 **admin** 角色的用户可见。

先决条件

- 您已在 **Maven** 存储库中安装了 **archetype**。
- 您已创建了 **archetype**，并在 **Maven** 存储库中的 **Business Central Settings** 中列出它。

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Projects**。
2. 选择或创建您要管理 **archetypes** 的空格。默认空间为 **MySpace**。
3. 单击 **Settings**。
4. 要在空间中包含或排除 **archetypes**，请选择 **Include** 复选框。
5. 在 **Actions** 列中，点 **archetype** 右侧的  图标，然后从下拉菜单中选择 **Set** 作为默认值，将 **archetype** 设置为空间的默认值。
6. 点击 **Save**。

第 25 章 自定义项目首选项

在 Business Central 中，项目是您空间的一部分并存储相关的资产。您可以在一个空间中添加多个项目。

例如，一个机构包括不同的部门，如 HR、Payroll、Engineering 和 R&D。您可以将每个部门映射到 Business Central 中的空间，以及添加相应的项目。

您可以自定义 Business Central 中的项目设置。另外，您可以创建新项目，或者从现有的 Git 存储库克隆项目。

流程

1. 在 Business Central 中，选择右上角的 Admin 图标，再选择 项目。
2. 在 "项目"首选项 "面板中，选择您想要修改的首选项。项目首选项包括：

- 项目导入 ：此首选项由以下属性组成：
  - 选择 允许在集群中导入多个项目，以导入集群中的多个项目。
- 导出文件 ：这种首选项由以下属性组成：

表 25.1. 文件导出属性

字段	描述
PDF 情况介绍	判断 PDF 是否是 portrait 还是 landscape。
PDF 单元	确定 PDF 单元是 *PT*、*MM*、*CN* 或 *IN*。
PDF 页面格式	确定 PDF 页面格式是否为 *A[0-10]*、*B[0-10]* 或 *C[0-10]*。

- 空格 ：此首选项由以下属性组成：

表 25.2. 空格属性

字段	描述
Name	如果没有存在，则自动创建空间的默认名称。
所有者	如果没有存在，则自动创建空间的默认所有者。
组 ID	如果不存在，则自动创建的空间的默认组 ID。
别名（以 sular 形式）	决定空格的自定义别名（单数）。
alias（以复数形式）	决定空格的自定义别名（复数）

- **默认值 ：此首选项由以下属性组成：**

表 25.3. 默认值属性

字段	描述
版本	创建项目时的默认版本号。
描述	项目在创建项目时的默认描述。
分支	使用 Git 存储库时要使用的默认分支。
资产(Aram Per Page)	用于自定义项目中每个页面资产数量。默认值为 **15**。

- **高级 GAV 首选项 ：此首选项由以下属性组成：**

表 25.4. 高级 GAV 首选项属性

字段	描述
禁用 GAV 冲突检查？	决定是否启用或禁用 GAV 冲突检查。禁用此复选框可让项目包含相同的 GAV（组 ID、工件和版本）。
是否允许子 GAV 版本？	决定允许子项目或子项目包含 GAV 版本。

注意

在开发模式中为项目禁用重复的 GAV 检测。要在 Business Central 中为项目启用重复的 GAV 检测，请转至 Project Settings → General Settings → Version，并将 Development Mode 选项切换为 OFF （如果适用）。

3. **点击 Save。**

第 26 章 自定义工件存储库属性

在某些情况下，项目需要解决外部依赖项以构建域模型 JAR 文件。仓库包含所需的工件，并具有以下功能：

- 存储库是 Maven 存储库。
- 所有快照都带有时间戳。
- 资产大多保存在本地硬盘中。

默认情况下，工件存储库位于 $WORKING_DIRECTORY/repositories/kie 中。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Artifact Repository。Artifact Repository 页面将打开。
2. 进行选择并在 Properties 部分输入信息。
3. 点击 Save。

第 27 章 自定义语言设置

您可以在 Business Central Settings 页面中更改语言。Business Central 支持以下语言：

- English
- 西班牙语
- 法语
- 日语

默认语言为英文。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Languages。语言选择器 窗口将打开。
2. 从 Language 列表中选择所需的语言。
3. 点 确定。

第 28 章 自定义进程管理

您可以编辑 Process 管理 页面上的 每页 属性，自定义 Business Central 中的默认分页选项。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Process Administration。

2. 在 Properties 部分中，更新 每个页面属性的默认项，然后单击 Save。

注意

您可以指定要在每个页面上显示的 10、20、50 或 100 个项目。

第 29 章 自定义过程设计器

您可以编辑 Business Central Settings 页面中图表编辑器的属性，自定义 Business Central Central 中的流程设计器。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 Process Designer。
2. 在 Properties 部分中，更新以下任何属性：
   - 选择 Auto hide category panel 复选框，以自动隐藏类别工具栏面板。
   - 在 Drawing 区域宽度 字段中，输入 2800 和 5600 之间的整数值，以设置绘制区域的宽度。
   - 在 Drawing 区域 height 字段中，输入 1400 和 2800 之间的整数值，以设置绘制区域的高度。
   - 如果使用高分辨率显示，请选择 启用 HiDPI 复选框，并看到"模糊"文本和对象。默认禁用这个选项。
3. 点击 Save。

第 30 章 SSH 密钥

Business Central 提供 SSH 密钥存储服务，以启用用户 SSH 身份验证。Business Central 提供可配置的默认 SSH 密钥存储、可扩展的 API（用于自定义实施），并支持多个 SSH 公钥格式。

您可以访问 Business Central Settings 页面中的 SSH Keys 选项，以注册您的 SSH 公钥。

30.1. 默认 SSH 密钥存储

Business Central 中包括的默认 SSH 密钥存储提供了一种基于文件的存储机制，用于存储用户的公钥。默认情况下，B Business Central 使用 *.security 文件夹作为根目录。但是，您还可以通过将 appformer.ssh.keys.storage.folder 系统属性的值设置为指向不同的文件夹来使用自定义存储路径。

SSH 公钥存储在 {securityFolderPath}/pkeys/{userName}/ 文件夹结构中。

每个 SSH 公钥均由以下文件组成，位于 storage 文件夹中：

- {keyID}.pub ：此文件包含 SSH 公钥内容。由于文件名决定了系统上的逻辑密钥 ID，请确保在运行时不会修改文件名。

  例如：

  ```
  ssh-rsa
  AAAAB3NzaC1yc2EAAAADAQABAAACAQDmak4Wu23RZ6XmN94bOsqecZxuTa4RRhhQm
  HmTZjMB7HM57/90u/B/gB/GhsPEu1nAXL0npY56tT/MPQ8vRm2C2W9A7CzN5+z5yyL3W01Y
  Zy3kzslk77CjULjfhrcfQSL3b2sPG5jv5E5/nyC/swSytucwT/PE7aXTS9H6cHIKUdYPzIt94SHoBx
  WRIK7PJi9d+eLB+hmDzvbVa1ezu5a8yu2kcHi6NxxfI5iRj2rsceDTp0imC1jMoC6ZDfBvZSxL9FX
  TMwFdNnmTlJveBtv9nAbnAvIWlilS0VOkdj1s3GxBxeZYAcKbcsK9sJzusptk5dxGsG2Z8vInaglN
  6OaOQ7b7tcomzCYYwviGQ9gRX8sGsVrw39gsDIGYP2tA4bRr7ecHnlNg1b0HCchA5+QCDk
  4Hbz1UrnHmPA2Lg9c3WGm2qedvQdVJXuS3mlwYOqL40aXPs6890PvFJUlpiVSznF50djPnws
  MxJZEf1HdTXgZD1Bh54ogZf7czyUNfkNkE69yJDbTHjpQd0cKUQnu9tVxqmBzhX31yF4VcsMe
  ADcf2Z8wlA3n4LZnC/GwonYlq5+G93zJpFOkPhme8c2XuPuCXF795lsxyJ8SB/AlwPJAhEtm0y
  0s0l1l4eWqxsDxkBOgN+ivU0czrVMssHJEJb4o0FLf7iHhOW56/iMdD9w== userName
  ```

- .{keyID}.pub.meta ：此文件包含 JSON 格式的关键元数据。如果密钥没有元数据，则会动态生成新的元数据文件。

  例如：

```
{
  "name":"Key",
  "creationDate":"Oct 10, 2018 10:10:50 PM",
  "lastTimeUsed":"Oct 11, 2018 12:11:23 PM"
}
```

30.2. 自定义 SSH 密钥存储

您可以根据您的要求扩展和自定义默认的 SSH 密钥存储。使用 appformer.ssh.keystore 系统属性指定要使用的 SSH 服务的 Java 类名称。如果未定义此属性或者它包含不正确的值，则会加载默认的 SSH 密钥存储。

注意

要创建 SSH 密钥存储的自定义实施，您的 Java 类必须实施 uber fire-ssh-api 模块中定义的 org.uberfire.ssh.backend.keystore.SSHKeyStore 类。

30.3. 创建 SSH 密钥

在 Business Central 中添加或注册 SSH 密钥前，您必须在您的系统中生成 SSH 密钥。

流程

1. 在 系统上打开一个命令终端。

2. 运行 ssh-keygen 命令创建 SSH 密钥，如下例所示，其中 < user_login& gt; 是您的用户名：

```
ssh-keygen -t rsa -b 4096 -C "<user_login>"
```

注意

Business Central 密钥存储支持的 SSH 密钥格式是 ssh-rsa、ssh-dss、ecdsa-sha2-nistp256、ecdsa-sha2-nistp384 和 ecdsa-sha2-nistp521。

3. 提示时，按 Enter 并接受默认密钥文件位置，如下例所示，其中 < user_login& gt; 是您的用户名：

```
Enter a file in which to save the key (/home/<user_login>/.ssh/id_rsa): [Press enter]
```

4. 在命令提示符处，输入并确认密码短语：

```
Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]
```

5. 启动 **ssh-agent** ：

```
eval "$(ssh-agent -s)"
Agent pid <any-number-here>
```

6. 将新的 SSH 私钥添加到 **ssh-agent**。如果您使用了不同的密钥名称，请在该代码中替换 **id_rsa** ：

```
ssh-add ~/.ssh/id_rsa
```

30.4. 使用 SSH 密钥存储注册 SSH 公钥

您必须将新创建的 SSH 公钥注册到 Business Central 密钥存储。

流程

1. 在 系统上打开一个命令终端。
2. 如下例所示，其中 **id_rsa** 是您的密钥名称：

```
cat ~/.ssh/id_rsa.pub
```

3. 复制 SSH 公钥的内容。
4. 在 Business Central 中，选择屏幕右上角的 **Admin** 图标，然后选择 **SSH Keys**。
5. 在 **SSH Keys** 页面中，点 **Add SSH Key**。

6. 在 Add SSH Key 窗口中，在 Name 字段中输入名称，并将 SSH 公钥的内容复制到 Key 字段。

注意

Name 和 Key 字段是必须的。

7. 点 Add SSH Key 注册密钥。

30.5. 删除 SSH 密钥

您可以从 SSH Keys 页面从 Business Central 中删除 SSH 密钥。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 SSH Keys。
2. 在 SSH Keys 页面中，点击您要删除的 SSH 密钥的删除图标。
3. 单击 Delete SSH Key 以确认删除。

第 31 章 在 BUSINESS CENTRAL 中管理自定义任务

自定义任务（工作流项目）是可以运行自定义逻辑的任务。您可以在多个业务进程或 Business Central 中的所有项目中自定义和重复使用自定义任务。您还可以在设计器面板中添加自定义元素，包括名称、图标、子类别、输入和输出参数以及文档。Red Hat Decision Manager 在 Business Central 中的自定义任务存储库中提供一组自定义任务。您可以启用或禁用默认自定义任务，并将自定义任务上传到 Business Central 中，以便在相关进程中实施任务。

注意

Red Hat Decision Manager 包括一组有限的支持的自定义任务。不支持没有包括在 Red Hat Decision Manager 中的自定义任务。

流程

1. 在 Business Central 中，点右上角的  并选择 **Custom Tasks Administration**。

   本页列出了自定义任务安装设置，以及在整个 Business Central 中项目中的进程可用的自定义任务。您在项目级别设置中启用的自定义任务在项目级别设置中可用，然后安装要在进程中使用的每个自定义任务。自定义任务安装到项目中的方式由您在此 Custom Tasks Administration 页面上的 Settings 下启用或禁用的全局设置决定。

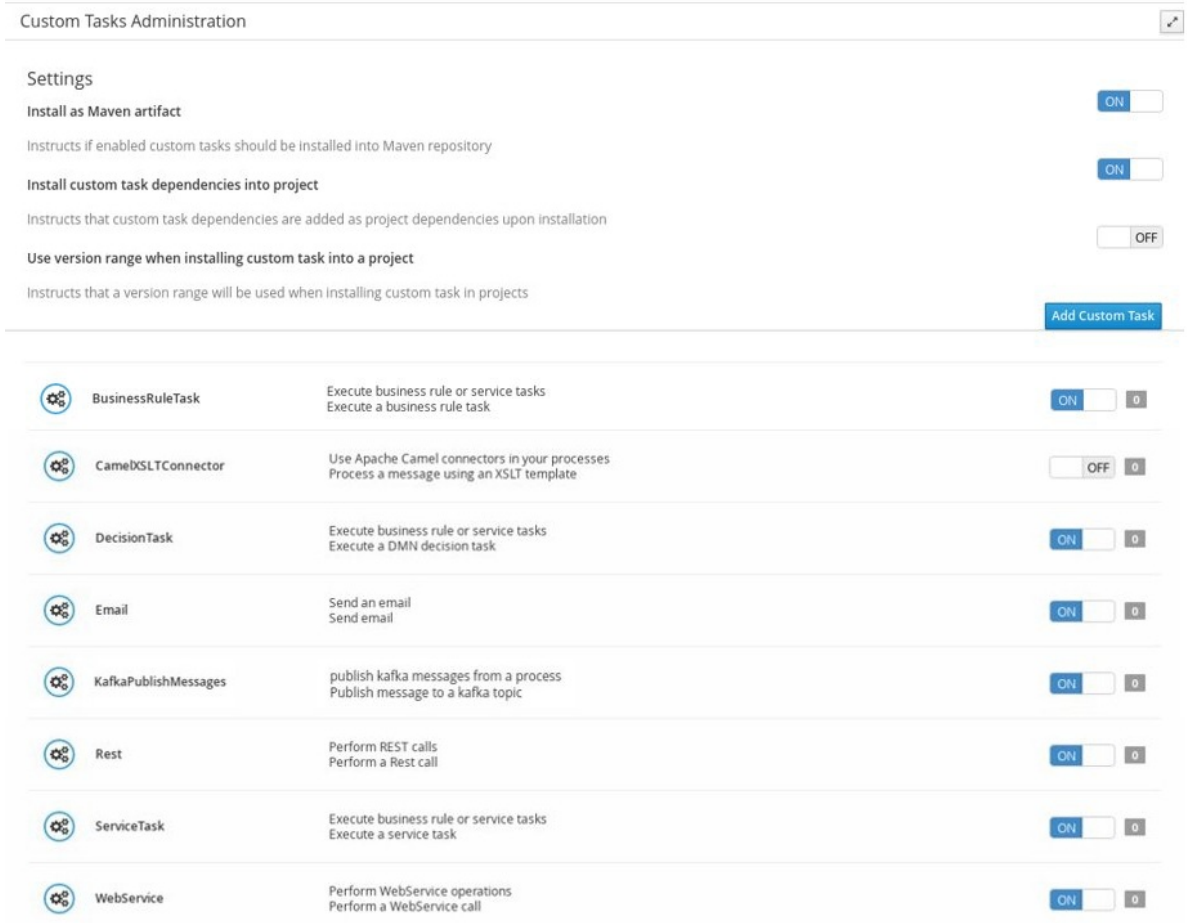
2. 在 Settings 下，启用或禁用每个设置，以确定用户在项目级别安装可用自定义任务时如何实施。

   可用的自定义任务设置如下：

   - 安装为 Maven 工件 ：将自定义任务 JAR 文件上传到使用 Business Central 配置的 Maven 存储库（如果尚不存在）。
   - 将 自定义任务依赖项安装到项目 ：将任何自定义任务依赖项添加到安装任务的项目的 pom.xml 文件中。
   - 将 自定义任务安装到项目时 使用版本范围 ：使用版本范围而不是作为项目依赖项添加的自定义任务的固定版本。示例： [7.16） 而不是 7.16.0.Final

3. 根据需要启用或禁用任何可用的自定义任务（设置为 ON 或 OFF）。您在 Business Central 中所有项目的项目级别设置中显示您启用的自定义任务。

图 31.1. 启用自定义任务和自定义任务设置



4. 要添加自定义任务，请点 Add Custom Task，浏览到相关的 JAR 文件，然后点 Upload 图标。如果类实施 WorkItemHandler，您可以通过将文件单独添加到 Business Central 来将注解替换为 .wid 文件。

5. 可选： 要删除自定义任务，请点击您要删除的自定义任务行的 remove，然后点击 Ok 以确认删除。

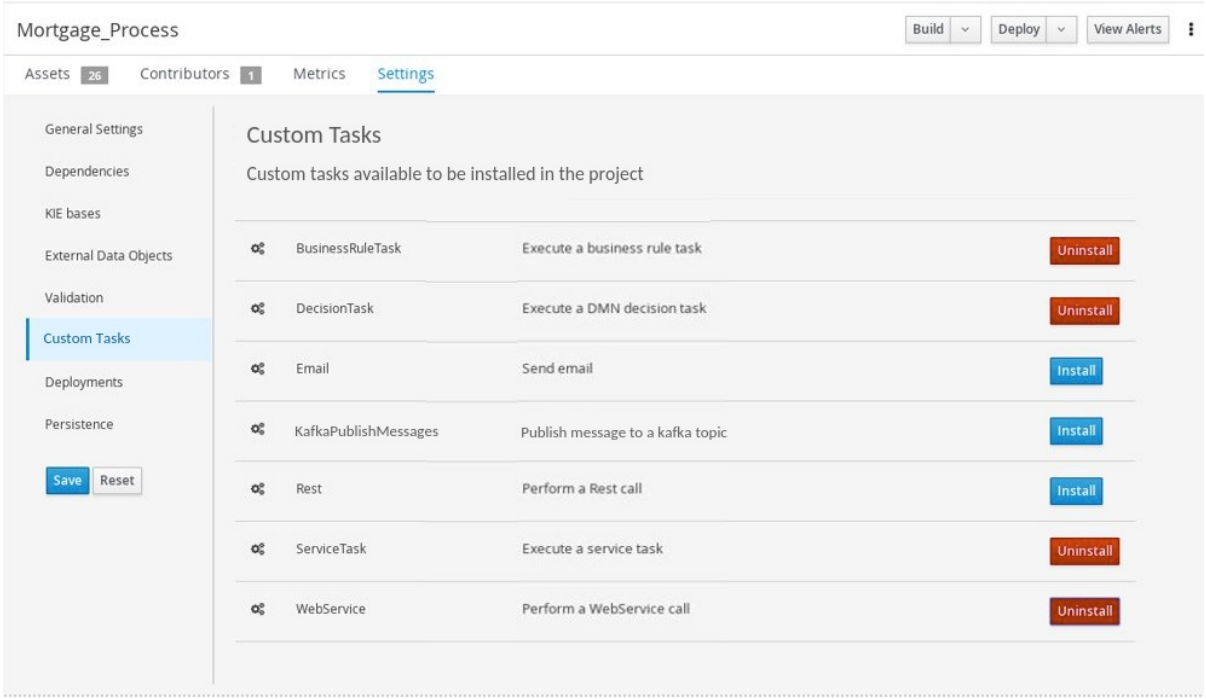
6. 配置所有必要的自定义任务后，进入 Business Central 中的项目，再进入 project Settings → Custom Tasks 页面，以查看您启用的可用自定义任务。

7. 对于每个自定义任务，点 Install 使任务可供该项目中的进程使用，或者点击 Uninstall 从项目中的进程中排除任务。

8. 如果在安装自定义任务时提示您输入其他信息，请输入所需信息，然后再次点 Install。

自定义任务所需的参数取决于任务的类型。例如，规则和决策任务需要工件 GAV 信息（组 ID、工件 ID、版本）、电子邮件任务需要主机和端口访问信息，并且 REST 任务需要 API 凭据。其他自定义任务可能不需要任何其他参数。

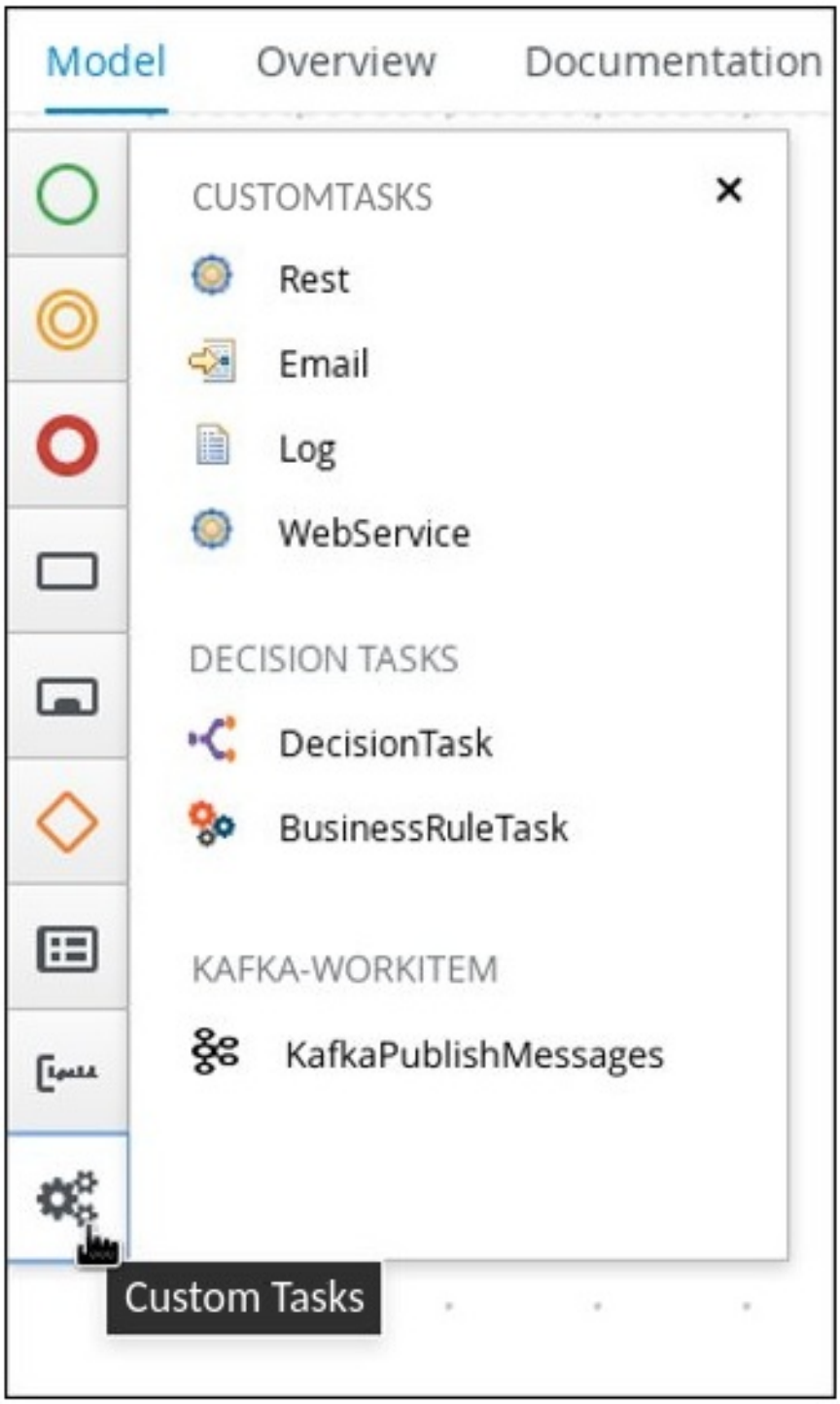
图 31.2. 安装自定义任务以便在进程中使用



9. 点击 Save。

10. 返回到项目页面，在项目中选择或添加批准过程，并在流程设计器 palette 中选择 Custom Tasks 选项来查看您启用和安装的可用自定义任务：

图 31.3. 在进程设计器中访问已安装的自定义任务



第 32 章 LDAP 连接

Business Central 为 LDAP 服务器提供专用的 UserGroupCallback 实施，以便用户任务服务直接从 LDAP 服务检索用户、组和角色的信息。

您可以配置以下 LDAP UserGroupCallback 实现属性：

表 32.1. LDAP UserGroupCallback 属性

属性	描述
ldap.bind.user	用于连接到 LDAP 服务器的用户名。 如果未指定，则此属性为可选，并且 LDAP 服务器接受匿名访问。
ldap.bind.pwd	用于连接到 LDAP 服务器的密码。 如果未指定，则此属性为可选，并且 LDAP 服务器接受匿名访问。
ldap.user.ctx	使用用户信息在 LDAP 中上下文。
ldap.role.ctx	在 LDAP 中具有组和角色的上下文.
ldap.user.roles.ctx	在 LDAP 中具有用户组和角色成员资格信息的上下文。 若未指定，则此属性为可选，并且使用了 **ldap.role.ctx** 属性。
ldap.user.filter	过滤搜索用户信息。 此属性通常包含替换键 {0}，它们被替换为参数。
ldap.role.filter	过滤搜索组和角色信息。 此属性通常包含替换键 {0}，它们被替换为参数。
ldap.user.roles.filter	过滤搜索用户和组角色成员资格信息。 此属性通常包含替换键 {0}，它们被替换为参数。
ldap.user.attr.id	LDAP 中用户 ID 的属性名称。 如果未指定，则此属性为可选，并且改为使用 **uid** 属性。

属性	描述
ldap.roles.attr.id	LDAP 中组和角色 ID 的属性名称。 若未指定，此属性为可选，并且使用了 **cn** 属性。
ldap.user.id.dn	DN 中的用户 ID 指示回调在搜索角色之前查询用户 DN。这是可选的，默认为 **false**。
java.naming.factory.initial	初始上下文工厂类名称；默认情况下为 **com.sun.jndi.ldap.LdapCtxFactory**。
java.naming.security.authentication	如果可能的值不是、 **简单** **和强大的** 验证类型。**默认为 simple。**
java.naming.security.protocol	要使用的安全协议，例如 **ssl**。
java.naming.provider.url	LDAP url（默认为 **ldap://localhost:389**; 如果协议被设置为 *ssl*，则 **ldap://localhost:636**

32.1. LDAP USERGROUPCALLBACK 实现

您可以通过使用以下方法之一配置对应的 LDAP 属性来使用 LDAP UserGroupCallback 实现：

- **以编程方式：使用相应的 LDAPUserGroupCallbackImpl 属性构建属性对象，并使用与它参数相同的属性对象创建 LDAPUserGroupCallbackImpl。**

  例如：

```
import org.kie.api.PropertiesConfiguration;
import org.kie.api.task.UserGroupCallback;
...
Properties properties = new Properties();
properties.setProperty(LDAPUserGroupCallbackImpl.USER_CTX, "ou=People,dc=my-domain,dc=com");
properties.setProperty(LDAPUserGroupCallbackImpl.ROLE_CTX, "ou=Roles,dc=my-domain,dc=com");
properties.setProperty(LDAPUserGroupCallbackImpl.USER_ROLES_CTX, "ou=Roles,dc=my-domain,dc=com");
properties.setProperty(LDAPUserGroupCallbackImpl.USER_FILTER, "(uid={0})");
properties.setProperty(LDAPUserGroupCallbackImpl.ROLE_FILTER, "(cn={0})");
properties.setProperty(LDAPUserGroupCallbackImpl.USER_ROLES_FILTER, "(member={0})");
```

```
UserGroupCallback ldapUserGroupCallback = new
LDAPUserGroupCallbackImpl(properties);

UserGroupCallbackManager.getInstance().setCallback(ldapUserGroupCallback);
```

- **声明性：在应用程序的根目录中创建 jbpm.usergroup.callback.properties 文件，或者将文件位置指定为系统属性。**

  例如：

  -Djbpm.usergroup.callback.properties=FILE_LOCATION_ON_CLASSPATH

  确保在启动用户任务服务器时注册 LDAP 回调。

  例如：

```
#ldap.bind.user=
#ldap.bind.pwd=
ldap.user.ctx=ou\=People,dc\=my-domain,dc\=com
ldap.role.ctx=ou\=Roles,dc\=my-domain,dc\=com
ldap.user.roles.ctx=ou\=Roles,dc\=my-domain,dc\=com
ldap.user.filter=(uid\={0})
ldap.role.filter=(cn\={0})
ldap.user.roles.filter=(member\={0})
#ldap.user.attr.id=
#ldap.roles.attr.id=
```

其他资源

- *角色和用户*
- *Red Hat Single Sign-On 服务器管理指南*
- *定义 LDAP 登录域*
- *LDAP 登录模块*

- *LDAP 扩展登录模块*
- *高级LDAP 登录模块*
- *advancedAdLDAP 登录模块*
- *LDAP 连接选项*
- *LDAPUsers 登录模块*

第 33 章 数据库连接

Business Central 为使用红帽 Decision Manager 的数据库服务器提供专用 用户GroupCallback 实现，以启用用户任务服务。用户任务服务有助于检索直接来自数据库的用户和组（角色）的信息。

您可以配置以下数据库 UserGroupCallback 实现属性：

表 33.1. 数据库用户GroupCallback 属性

属性	描述
db.ds.jndi.name	用于连接的数据源的 JNDI 名称
db.user.query	验证用户是否存在
db.user.roles.query	为给定用户收集组
db.roles.query	验证组是否存在

33.1. 数据库用户GROUPCALLBACK 实现

在数据库 UserGroupCallback 实现中，您必须创建所需的数据库。您可以通过使用以下方法之一配置对应的数据库属性：

- 以编程方式：使用对应的 DBUserGroupCallbackImpl 属性构建属性对象，并使用同样的属性对象创建 DBUserGroupCallbackImpl。

  例如：

```
import static org.jbpm.services.task.identity.DBUserGroupCallbackImpl.DS_JNDI_NAME;
import static
org.jbpm.services.task.identity.DBUserGroupCallbackImpl.PRINCIPAL_QUERY;
import static org.jbpm.services.task.identity.DBUserGroupCallbackImpl.ROLES_QUERY;
import static
org.jbpm.services.task.identity.DBUserGroupCallbackImpl.USER_ROLES_QUERY;
...
props = new Properties();
props.setProperty(DS_JNDI_NAME, "jdbc/jbpm-ds");
props.setProperty(PRINCIPAL_QUERY, "select userId from Users where userId = ?");
props.setProperty(ROLES_QUERY, "select groupId from UserGroups where groupId = ?");
props.setProperty(USER_ROLES_QUERY, "select groupId from UserGroups where userId =
```

```
?");

callback = new DBUserGroupCallbackImpl(props);
```

- **声明性：在应用程序的根目录中创建 jbpm.usergroup.callback.properties 文件，或者将文件位置指定为系统属性。**

  例如：

  -Djbpm.usergroup.callback.properties=FILE_LOCATION_ON_CLASSPATH

  确保在启动用户任务服务器时注册数据库回调。

  例如：

```
System.setProperty("jbpm.usergroup.callback.properties",
"/jbpm.usergroup.callback.db.properties");
callback = new DBUserGroupCallbackImpl(true);
...
db.ds.jndi.name = jdbc/jbpm-ds
db.user.query = select userId from Users where userId = ?
db.roles.query = select groupId from UserGroups where groupId = ?
db.user.roles.query = select groupId from UserGroups where userId = ?
```

其他资源

- *角色和用户*

第 34 章 使用 SETTINGS.XML 文件配置 MAVEN

Java 应用程序开发使用 Apache Maven 构建自动化工具构建和管理软件项目。Maven 使用项目对象模型(POM)配置 XML 文件来定义项目属性和构建过程。

Maven 使用存储库来存储 Java 库、插件和其他构建工件。存储库可以是本地或远程存储库。本地存储库是从本地计算机上缓存的远程仓库下载工件。远程存储库是使用通用协议访问的任何其他存储库，如 http:// （位于 HTTP 服务器上时），或者在位于文件服务器时，使用 file:// 访问。default 存储库是公共远程 Maven 2 Central 存储库。Maven 的配置通过修改 settings.xml 文件来执行。您可以在 M2_HOME/conf/settings.xml 文件中配置全局 Maven 设置，或者在 USER_HOME/.m2/settings.xml 文件中配置用户级设置。

其他资源

- *为 Business Central 和 KIE 服务器配置外部 Maven 存储库*
- *在 Maven 中打包和部署 Red Hat Decision Manager 项目*
- *Red Hat Decision Manager 的 Maven 设置和存储库*
- *系统与 Maven 集成*
- *欢迎使用 Apache Maven*
- *Apache Maven 项目 - 仓库简介*
- *Apache Maven Parent POMs 参考*.

第 35 章 GAV 检查管理

在 Business Central 中，项目由 Group ID、Artifact ID 和版本(GAV)Maven 命名约定来标识。GAV 值区分项目和项目版本，以及识别特定项目的依赖项。

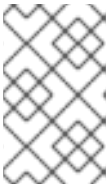
默认情况下，B Business Central 会检测重复的 GAV。具有 *admin* 角色的用户可以禁用此功能。

35.1. 配置 GAV 检查和子 GAV 版本

这个步骤描述了如何在 Business Central 中配置 GAV 检查。

流程

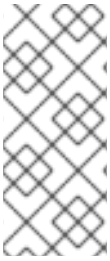
1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 在项目窗口中，单击 Settings 选项卡。
3. 在 General Settings 选项卡中，执行以下任一任务：
   - 要启用其他项目具有相同的 GAV，请选择 Disable GAV conflict 复选框。
   - 要启用子项目具有 GAV 版本，请选择 Allow child GAV 版本 复选框。
4. 点击 Save。



注意

您可以点击 重置 来撤消所有更改。

5. 单击 Save 以确认更改。



注意

在 Development Mode 中的项目禁用了重复的 GAV 检测。要在 Business Central 中启用重复的 GAV 检测，请转至 Project Settings → General Settings → Version，并将 Development Mode 选项切换为 OFF （如果适用）。

35.2. 为所有项目配置 GAV 检查

此流程描述了如何配置 GAV 检查 Business Central 中所有项目。您还可以在系统启动时禁用 GAV 检查。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，然后选择 项目。这时将打开 Projects 窗口。
2. 在 Advanced GAV 首选项 选项卡中，执行以下任一任务：
   - 要启用其他项目具有相同的 GAV，请选择 Disable GAV conflict 复选框。
   - 要启用子项目具有 GAV 版本，请选择 Allow child GAV 版本 复选框。
3. 点击 Save。

注意

您还可以通过将系统启动时 Business Central 的 org.guvnor.project.gav.check.disabled 系统属性设置为 *true* 来禁用重复的 GAV 检测功能：

```
$ ~/EAP_HOME/bin/standalone.sh -c standalone-full.xml
-Dorg.guvnor.project.gav.check.disabled=true
```

第 36 章 在 KIE 服务器和 BUSINESS CENTRAL 中配置环境模式

您可以将 KIE 服务器设置为在 生产环境 模式下运行，或者在 开发 模式下运行。开发模式提供了灵活的部署策略，可让您更新现有部署单元（KIE 容器），同时维护活跃的进程实例来进行小更改。它还允许您在更新活跃进程实例进行更大更改前重置部署单元状态。生产模式是生产环境的最佳选择，每个部署都会创建一个新的部署单元。

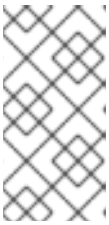
在开发环境中，您可以点击 Deploy in Business Central 将构建的 KJAR 文件部署到 KIE 服务器，而无需停止任何正在运行的实例（如果适用），或者单击 Redeploy 以部署构建的 KJAR 文件并替换所有实例。下次部署或重新部署构建的 KJAR 时，以前的部署单元（KIE 容器）会在同一目标 KIE 服务器中自动更新。

在生产环境中，业务中心的 Redeploy 选项被禁用，您只能单击 Deploy 以将构建的 KJAR 文件部署到 KIE 服务器上的新部署单元（KIE 容器）。

流程

1. 要配置 KIE 服务器环境模式，请将 org.kie.server.mode 系统属性设置为 org.kie.server.mode=development 或 org.kie.server.mode=production。

2. 要在 Business Central 中配置项目部署行为，请转至 Project Settings → General Settings → Version，再切换 Development Mode 选项。



注意

默认情况下，Business Central 中的 KIE 服务器和所有新项目均为开发模式。

您不能部署打开开发模式的项目，或使用手动将 SNAPSHOT 版本后缀添加到生产模式中的 KIE 服务器。

第 37 章 GIT HOOK 和远程 GIT 存储库集成

Git hook 是在 Git 事件之前或 git push 等 Git 事件之前或之后执行的 bash 脚本。在 Business Central 中，您可以使用 Git hook 配置存储库，在每次发生事件时触发指定操作。有关 Git hook 的更多信息，*请参阅自定义 Git Hook*。

您可以使用 post-commit Git hook 将远程 Git 存储库与 Business Central 集成。这可让您在 Business Central 和远程存储库之间进行自动内容复制。例如，您可以实施实时备份策略，其中您对 Business Central 项目的更改会复制到远程 Git 存储库。

注意

Business Central 只支持 post-commit Git hook。

post-commit Git hook 作为同步操作在每个提交后执行。Business Central 会等待 post-commit bash 完成且在存储库中没有其他写入操作。

37.1. 创建 POST-COMMIT GIT HOOK

您可以创建一个 post-commit Git hook bash 脚本文件，用于执行该文件中包含的代码，或者从其他文件（如 Java 程序）执行代码。

流程

1. 创建 post-commit Git hook 文件：

```
$ touch post-commit
```

2. 将 post-commit 文件的权限设置为 755 ：

```
$ chmod 755 post-commit
```

3. 将 #!/bin/bash 以及任何所需代码添加到 post-commit 文件中，例如：

- 将所有更改推送到远程存储库：

```
#!/bin/bash
git push origin +master
```

- 记录信息：

```
#!/bin/bash
echo 'Hello World'
```

- 执行另一个文件的代码：

```
#!/bin/bash
java -jar _EAP_HOME_/bin/.niogit/<SPACE>/<PROJECT_NAME>.git/hooks/git-push.jar
```

注意

要使用执行 Java 代码的 post-commit Git hook，您必须使用以下 Java 库：

- JGit ：用于与内部 Business Central Git 存储库交互。
- 用于 Java 的 GitHub API ：用于与 GitHub 通信。

有关 post-commit Git hook 和 Java 代码示例的更多信息，请参阅 *Business Central post-commit Git Hooks Integration*。

37.2. 导入远程 GIT 存储库

您可以将远程 Git 存储库导入到 Business Central，并配置 post-commit Git hook 以自动将更改推送到该远程存储库。

先决条件

- Red Hat Process Automation Manager 安装在 Red Hat JBoss EAP 7.4 服务器实例中。
- Red Hat Decision Manager 项目存在于外部 Git 存储库中。

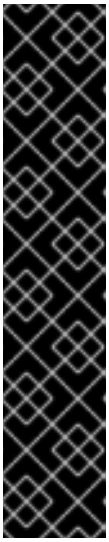
- 读取外部 Git 存储库的访问凭据。

- （对于 Windows）Cygwin 安装在安装过程中添加的 Git 软件包，并将到 Cygwin /bin 文件夹的路径添加到环境变量 PATH 变量中。例如，C:\cygwin64\bin。有关 Cygwin 安装的更多信息，请参阅 安装和更新 Cygwin 软件包。

流程

1. 在 Business Central 中，前往 Menu → Projects。
2. 选择或创建您要导入 Git 项目的空间。
3. 点击屏幕右侧的  并选择 Import Project。
4. 在 Import Project 窗口中，输入 Git 存储库的 URL，如 https://github.com/USERNAME/REPOSITORY_NAME.git，以及 Git 存储库的凭据。
5. 点 Import。

该项目添加到 Business Central Git 存储库，然后在该空间中可用。



重要

使用 HTTPS 或 Git 协议，而不是 SCP 风格的 SSH URL。如果您使用这个 URL，B Business Central 不支持基本的 SSH URL 和错误。

您必须在 Git 供应商中配置您的公共 ssh 密钥。

Git 存储库必须是 KJAR 项目，仅包含与 Red Hat Decision Manager 版本兼容的单个 KJAR。KJAR 内容必须位于存储库的根目录中。

6. 在命令终端中，导航到位于项目的存储库 Git 文件夹中的 hook 文件夹。例如：

```
$ cd _EAP_HOME_/bin/.niogit/<SPACE>/<PROJECT_NAME>.git/hooks
```

7. 创建一个 post-commit 文件，将更改推送到远程 Git 存储库。例如：

```
#!/bin/sh
git push origin +master
```

有关创建 post-commit Git hook 的更多信息，请参阅 第 37.1 节 “创建 post-commit Git hook”。

8. 可选：要检查配置是否成功，请在 Business Central 中创建指导规则：

   a. 在 Business Central 中，转至 Menu → Projects → Add Asset → Guided Rule。

   b. 在 Create new Guided Rule 页面中，输入所需的信息。

   c. 点 确定。

   Business Central 自动将所有更改推送到远程存储库。

其他资源

- *自定义 Git - Git Hook*

37.3. 为现有远程 GIT 项目存储库配置 GIT HOOK

如果您有一个现有的远程 Git 存储库项目，您可以在现有项目的远程 Git 存储库中创建 post-commit Git hook，并将远程 Git 存储库与 Business Central 集成。

先决条件

- Red Hat Process Automation Manager 安装在 Red Hat JBoss EAP 7.4 服务器实例中。

- Red Hat Decision Manager 项目存在于外部 Git 存储库中。
- 读取外部 Git 存储库的访问凭据。
- （对于 Windows 操作系统），Cygwin 安装在安装过程中添加的 Git 软件包，而 Cygwin /bin 文件夹的路径会添加到您的环境变量 PATH 变量中。例如，C:\cygwin64\bin。有关 Cygwin 安装的更多信息，请参阅 安装和更新 Cygwin 软件包。

流程

1. 在命令终端中，导航到位于项目的存储库 Git 文件夹中的 hook 文件夹。例如：

```
$ cd _EAP_HOME_/bin/.niogit/<SPACE>/<PROJECT_NAME>.git/hooks
```

2. 创建一个 post-commit 文件，将更改推送到远程 Git 存储库。例如：

```
#!/bin/sh
git push origin +master
```

有关创建 post-commit Git hook 的更多信息，请参阅 第 37.1 节 “创建 post-commit Git hook”。

3. 可选：要检查配置是否成功，请在 Business Central 中创建指导规则：

   a. 在 Business Central 中，转至 Menu → Projects → Add Asset → Guided Rule。

   b. 在 Create new Guided Rule 页面中，输入所需的信息。

   c. 点 确定。

   Business Central 自动将所有更改推送到远程存储库。

37.4. 将 GIT HOOK 配置为 BUSINESS CENTRAL 的系统属性

如果您没有现有的 Git 存储库项目，或者想将 post-commit Git hook 应用到大量项目存储库，您可以指定包含 org.uberfire.nio.git.hooks 系统属性的 hook 文件的目录。此目录复制到 Git 存储库。

注意

如果您指定 org.uberfire.nio.git.hooks 系统属性，则所有 Business Central 内部存储库和项目仓库都使用 post-commit Git hook。您应该只在脚本中使用完全限定的路径。

先决条件

- Red Hat Process Automation Manager 安装在 Red Hat JBoss EAP 7.4 服务器实例中。
- （对于 Windows 操作系统），Cygwin 安装在安装过程中添加的 Git 软件包，而 Cygwin /bin 文件夹的路径会添加到您的环境变量 PATH 变量中。例如，C:\cygwin64\bin。有关 Cygwin 安装的更多信息，请参阅 安装和更新 Cygwin 软件包。

流程

1. 在本地系统的目录中创建 post-commit Git hook。

   有关创建 post-commit Git hook 的更多信息，请参阅 第 37.1 节 “创建 post-commit Git hook”。

2. 要使用 org.uberfire.nio.git.hooks 系统属性使用 hook 文件来指定目录，请执行以下操作之一：

   - 将 org.uberfire.nio.git.hooks 系统属性添加到 standalone.xml 文件。例如：

     ```
     <system-properties>
       <property name="org.uberfire.nio.git.hooks" value="_EAP_HOME_/hooks">
       </property>
       ...
     </system-properties>
     ```

   - 在执行 Business Central 时，请使用 -Dorg.uberfire.nio.git.hooks 环境变量。例如：

```
$ ./standalone.sh -c standalone-full.xml -
Dorg.uberfire.nio.git.hooks=_EAP_HOME_/hooks
```

3. 启动 **Business Central**。

post-commit Git hook 复制到所有 Business Central 内部存储库和项目存储库。

其他资源

- *自定义 Git - Git Hook*

37.5. 集成远程 GIT 存储库

在以下示例中，您可以使用 **post-commit Git hook** 和 **Java** 代码将 **Business Central** 与远程 **Git** 存储库集成。有关 **Java** 代码示例，请参阅 ***Business Central post-commit Git Hooks Integration***。这个示例提供以下功能：

- 自动生成模板 **.gitremote** 配置文件
- 对所需参数验证 **.gitremote** 配置文件
- **Git** 会忽略 **.gitremote** 文件的 **ignore** 参数中定义的模式
- 用户的信息和通知输出
- 支持 **GitLab** 和 **GitHub** 令牌身份验证
- 支持 **GitLab** 组和子组项目创建
- 支持 **GitHub** 机构存储库创建

先决条件

- Red Hat Process Automation Manager 安装在 Red Hat JBoss EAP 7.4 服务器实例中。
- 已安装 Java Development Kit(JDK)8。
- 已安装 Maven。

流程

1. 在终端窗口中，将 GitHub 存储库克隆到您的系统：

```
$ git clone https://github.com/kiegroup/bc-git-integration-push.git
```

2. 进入克隆的存储库：

```
$ cd bc-git-integration-push
```

3. 执行 Maven 清理安装：

```
$ mvn clean install
```

4. 在 *EAP_HOME* 目录中创建一个 /hooks 文件夹：

```
$ mkdir -p _EAP_HOME_/hooks/
```

5. 将 git-push-2.1-SNAPSHOT.jar 复制到 *EAP_HOME*/hooks/ 文件夹：

```
$ cp bc-git-integration-push/target/git-push-2.1-SNAPSHOT.jar _EAP_HOME_/hooks/
```

6. 可选：要创建模板 .gitremote 配置文件，请运行 git-push-2.1-SNAPSHOT.jar:

```
$ java -jar git-push-2.1-SNAPSHOT.jar
```

模板 .gitremote 配置文件示例

```
#This is an auto generated template empty property file
provider=GIT_HUB
login=
password=
token=
remoteGitUrl=https://api.github.com/
useSSH=false
ignore=.*demo.*, test.*
githubOrg=OrgName
gitlabGroup=Group/subgroup
```

7. **修改 .gitremote 配置文件参数。**

表 37.1. .gitremote 参数示例

参数	描述
provider	Git 提供程序。只有两个值才会被接受： GIT_HUB 和 GIT_LAB。必填
login	Git 供应商的用户名。必填
password	纯文本密码。如果提供了 **令牌**，则不需要此项。
token	生成的令牌，以替换基于 **不安全连接 的用户名和密码** 。注：如果没有设置警告，会显示一个警告，您使用的是不受保护的连接。如果 **提供密码**，则不需要此项。注意：GitLab 只支持令牌身份验证。
remoteGitUrl	用于任何供应商的公共供应商 URL 或本地托管企业。必需。注：公共 GitHub URL 应该是 API URL。例如, api.github.com。
useSSH	布尔值以允许 SSH 协议将更改推送到远程存储库。可选。默认 = false。注： 这个参数使用本地 **~/.ssh/** 目录来获取 SSH 配置。
Ignore	以逗号分隔的正则表达式，忽略与任何这些表达式匹配的项目名称。可选。
githubOrg	如果 GitHub 用作提供程序，则定义存储库组织。可选。
gitlabGroup	如果 GitLab 用作提供程序可选，则定义存储库组和子组。

8. **在 *EAP_HOME*/hooks 中创建 post-commit Git hook 文件：**

```
$ touch post-commit
```

9. 将 post-commit 文件的权限设置为 755 ：

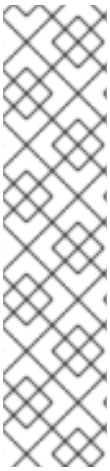
```
$ chmod 755 post-commit
```

10. 将 #!/bin/bash 和 code 添加到 post-commit 文件中，以执行 git-push-2.1-SNAPSHOT.jar ：

```
$ echo "#\!/bin/bash\njava -jar $APP_SERVER_HOME/hooks/git-push-2.1-SNAPSHOT.jar" > hooks/post-commit
```

11. 以 -Dorg.uberfire.nio.git.hooks 环境变量开始 Business Central。例如：

```
$ ./standalone.sh -c standalone-full.xml -Dorg.uberfire.nio.git.hooks=_EAP_HOME_/hooks
```



注意

要使用执行 Java 代码的 post-commit Git hook，您必须使用以下 Java 库：

- JGit ：用于与内部 Business Central Git 存储库交互。
- 用于 Java 的 GitHub API ：用于与 GitHub 通信。

有关 post-commit Git hook 和 Java 代码示例的更多信息，请参阅 *Business Central post-commit Git Hooks Integration*。

37.6. GIT HOOK 退出代码

当 Git hook 退出整数值时，返回决定 Git hook 执行的状态。此整数值称为 Git hook 退出代码。执行状态可以是 success(1)，警告（2 到 30）或错误（31 至 255））。

37.7. 自定义 GIT HOOK 通知

Business Central 提供了一种机制，让用户能够根据 hook 退出代码接收自定义 Git hook 通知。

要启用通知机制，您必须创建一个包含自定义消息的 *.properties 文件，然后将该文件的路径指定为 appformer.git.hooks.bundle 系统属性的值。

流程

1. 创建 *.properties 文件并为每个退出代码添加一行，格式为对应的消息：

   <exit_code>=<display_message>

   是 <exit_code> Git hook 退出代码， & lt;display_message > 是用户显示的自定义消息。

   例如：

```
0=Success! All working as expected.
1=Warning! Please check the logs and advise your admin.
.
.
31=Error! Please advise your admin immediately.
```

注意

不需要在 *.properties 文件中定义所有可能的退出代码。通知只针对在 *.properties 文件中定义的退出代码出现。

重要

通知服务只支持属性文件中设定的 ISO 8859-1 (LATIN 1)字符集。如果要使用扩展字符，请使用其转义的 Unicode 字符序列。

2. 要启用 Git hook 通知，将文件的路径指定为 appformer.git.hooks.bundle 系统属性的值。

   请参阅以下 standalone.xml 文件示例，它带有指向 Messages.properties 文件的设置：

```
<system-properties>
  <property name="appformer.git.hooks.bundle" value="/opt/jboss-as/git-hooks-
messages/Messages.properties">
  </property>
  ...
</system-properties>
```

-

37.7.1. Business Central 中的 Git hook 通知

您可以在 Business Central 中查看 Git hook 通知。有三个 Git hook 退出代码通知类型。

表 37.2. Git hook UI 通知类型

退出代码	自定义消息	UI 通知颜色
0	成功！所有工作均符合预期。	绿色
1 到 30	警告！请检查日志并建议您的管理员。	orange
31 到 255	错误！请立即推荐您的管理员。	红色

重要

UNIX 机器只支持 0（成功）到 255（错误）之间的错误代码，此范围之外的任何退出代码都将最终转换为不同的代码，从而导致显示错误的通知信息。

Windows 机器没有此限制并支持广泛的退出代码。

37.7.2. Git hook 通知国际化支持

您可以将额外属性文件放在与由 appformer.git.hooks.bundle 系统属性指定的原始属性文件相同的路径中，可以国际化通知消息。

不同本地化文件的名称必须是 < filename>_<lang>.properties，其中 <filename > 与原始文件相同。例如，如果系统属性指向 Messages.properties，您可以为英语创建 Messages_en.properties、适用于法语的 Messages_fr.properties，或者用于意大利语的 Messages_it.properties。

通知服务将根据用户的语言选择属性文件，如果没有该语言的可用转换，它将使用原始 Messages.properties 文件中的条目。

第 38 章 针对 BUSINESS CENTRAL 中分支的角色访问控制

Business Central 为用户授予限制特定协作者类型目标分支的访问选项。安全检查使用 Security Management 屏幕和 contributors 源来授予或拒绝空格和项目的权限。例如，如果用户具有更新项目的安全性权限，并根据贡献类型具有该分支的写入权限，则可以创建新的资产。

38.1. 自定义基于角色的访问控制

您可以为 Business Central 中的项目的每个分支自定义 contributors 角色权限。例如，您可以为分配给分支的每个角色设置 Read、Write、Delete 和 Deploy 访问权限。

流程

1. 在 Business Central 中，转至 Menu → Design → Projects。
2. 如果需要，添加一个新的投稿者：
   a. 单击项目名称，然后单击 Contributors 选项卡。
   b. 点 Add Contributor。
   c. 在文本字段中输入用户名。
   d. 从下拉列表中选择 Contributor 角色类型。
   e. 点 确定。
3. 为相关贡献者自定义基于角色的访问控制：
   a. 点 Settings → Branch Management。
   b. 从下拉列表中选择分支名称。

c.

在 Role Access 部分中，选择或取消选择权限复选框，为每个可用角色类型指定基于角色的分支访问权限。

d.

单击 Save，然后再次单击 Save 以确认更改。

第 39 章 查看进程实例日志

您可以从 Logs 选项卡查看实例的所有进程事件。实例日志会列出所有当前和上一个进程状态。Business Central 有两个用于流程实例的日志，业务和技术 日志。

流程

1. 在 Business Central 中，前往 Menu → Manage → Process Instances。
2. 在 Manage Process Instances 页面上，单击您要查看的日志的进程实例。
3. 选择 Logs 选项卡：
   - 单击 Business 查看业务和事件日志。
   - 点击 Technical 查看技术事件日志。
   - 点 As c 或 Desc 更改日志文件的顺序。

第 40 章 BUSINESS CENTRAL 系统属性

本节中列出的 Business Central 系统属性传递到 standalone*.xml 文件。

Git 目录

使用以下属性设置 Business Central Git 目录的位置和名称：

- org.uberfire.nio.git.dir: Business Central Git 目录的位置。
- org.uberfire.nio.git.dirname: Business Central Git 目录的名称。默认值： .niogit.
- org.uberfire.nio.git.ketch: Enables 或 disable Git ketch。
- org.uberfire.nio.git.hooks: Git hook 目录的位置。

git over HTTP

使用以下属性配置通过 HTTP 对 Git 存储库的访问：

- org.uberfire.nio.git.proxy.ssh.over.http: 指定 SSH 是否应使用 HTTP 代理。默认值： false。
- http.proxyHost ：定义 HTTP 代理的主机名。默认值： null。
- http.proxyPort ：定义 HTTP 代理的主机端口（整数值）。默认值： null。
- http.proxyUser ：定义 HTTP 代理的用户名。
- HTTP.proxyPassword ：定义 HTTP 代理的用户密码。
- org.uberfire.nio.git.http.enabled: Enables 或 disable the HTTP 守护进程。默认值： true.

- org.uberfire.nio.git.http.host: 如果启用了 HTTP 守护进程，它将使用此属性作为主机标识符。这是一个信息性属性，用于显示如何通过 HTTP 访问 Git 存储库。HTTP 仍然依赖于 servlet 容器。默认值： localhost.

- org.uberfire.nio.git.http.hostname: 如果 HTTP 守护进程被启用，它将使用此属性作为主机名标识符。这是一个信息性属性，用于显示如何通过 HTTP 访问 Git 存储库。HTTP 仍然依赖于 servlet 容器。默认值： localhost.

- org.uberfire.nio.git.http.port ：如果启用了 HTTP 守护进程，它将使用此属性作为端口号。这是一个信息性属性，用于显示如何通过 HTTP 访问 Git 存储库。HTTP 仍然依赖于 servlet 容器。默认值： 8080.

Git over HTTPS

使用以下属性通过 HTTPS 配置对 Git 存储库的访问：

- org.uberfire.nio.git.proxy.ssh.over.https: 指定 SSH 是否使用了 HTTPS 代理。默认值： false。

- HTTPS.proxyHost ：定义 HTTPS 代理的主机名。默认值： null。

- HTTPS.proxyPort ：定义 HTTPS 代理的主机端口（整数值）。默认值： null。

- HTTPS.proxyUser ：定义 HTTPS 代理的用户名。

- HTTPS.proxyPassword ：定义 HTTPS 代理的用户密码。

- user.dir ：用户目录的路径。

- org.uberfire.nio.git.https.enabled: 启用或禁用 HTTPS 守护进程。默认值： false

- org.uberfire.nio.git.https.host: 如果 HTTPS 守护进程已启用，它将使用此属性作为主

机标识符。这是一个信息性属性，用于显示如何通过 HTTPS 访问 Git 存储库。HTTPS 仍然依赖于 servlet 容器。默认值： localhost.

- org.uberfire.nio.git.https.hostname: 如果 HTTPS 守护进程已启用，它将使用此属性作为主机名标识符。这是一个信息性属性，用于显示如何通过 HTTPS 访问 Git 存储库。HTTPS 仍然依赖于 servlet 容器。默认值： localhost.

- org.uberfire.nio.git.https.port: 如果 HTTPS 守护进程已启用，它将使用此属性作为端口号。这是一个信息性属性，用于显示如何通过 HTTPS 访问 Git 存储库。HTTPS 仍然依赖于 servlet 容器。默认值： 8080.

JGit

- org.uberfire.nio.jgit.cache.instances ：定义 JGit 缓存大小。

- org.uberfire.nio.jgit.cache.overflow.cleanup.size: 定义 JGit 缓存溢出清理大小。

- org.uberfire.nio.jgit.remove.eldest.iterations: Enables 或 disable whether remove eldest JGit 迭代。

- org.uberfire.nio.jgit.cache.evict.threshold.duration: 定义 JGit 驱除阈值持续时间。

- org.uberfire.nio.jgit.cache.evict.threshold.time.unit ：定义 JGit 驱除阈值单元。

Git 守护进程

使用以下属性来启用和配置 Git 守护进程：

- org.uberfire.nio.git.daemon.enabled: Enables 或 disable the Git 守护进程。默认值： true.

- org.uberfire.nio.git.daemon.host: 如果 Git 守护进程已启用，它将使用此属性作为本地主机标识符。默认值： localhost.

- org.uberfire.nio.git.daemon.hostname: 如果 Git 守护进程已启用，它将使用此属性作

为本地主机名标识符。默认值： localhost

- org.uberfire.nio.git.daemon.port: 如果 Git 守护进程已启用，它将使用此属性作为端口号。默认值： 9418。

- org.uberfire.nio.git.http.sslVerify: 启用或禁用 Git 存储库的 SSL 证书检查。默认值： true.

注意

如果已经使用默认或者分配的端口，则会自动选择一个新端口。确保端口可用，并检查日志以了解更多信息。

Git SSH

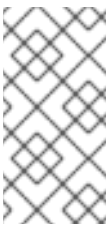
使用以下属性来启用和配置 Git SSH 守护进程：

- org.uberfire.nio.git.ssh.enabled: 启用或禁用 SSH 守护进程。默认值： true.

- org.uberfire.nio.git.ssh.host: 如果启用了 SSH 守护进程，它将使用此属性作为本地主机标识符。默认值： localhost.

- org.uberfire.nio.git.ssh.hostname: 如果启用了 SSH 守护进程，它将使用此属性作为本地主机名标识符。默认值： localhost.

- org.uberfire.nio.git.ssh.port: 如果启用了 SSH 守护进程，它将使用此属性作为端口号。默认值：800 1。



注意

如果已经使用默认或者分配的端口，则会自动选择一个新端口。确保端口可用，并检查日志以了解更多信息。

- org.uberfire.nio.git.ssh.cert.dir:存储本地证书的 .security 目录的位置。默认值：工作目录。

- org.uberfire.nio.git.ssh.idle.timeout:设置 SSH 闲置超时。

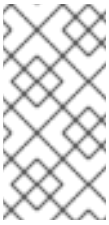
- org.uberfire.nio.git.ssh.passphrase ：在使用 SCP 风格 URL 克隆 git 存储库时用于访问操作系统的公钥存储的密码短语。示例： git@github.com:user/repository.git.

- org.uberfire.nio.git.ssh.algorithm: Algorithm 供 SSH 使用。默认值： RSA.

- org.uberfire.nio.git.gc.limit: Sets the GC 限值。

- org.uberfire.nio.git.ssh.ciphers ：以逗号分隔的密码字符串。可用的密码是 aes128-ctr,aes192-ctr,aes256-ctr,arcfour128,arcfour256,aes192-cbc,aes256-cbc.如果没有使用 属性，则会载入所有可用的密码。

- org.uberfire.nio.git.ssh.macs:以逗号分隔的消息验证代码(MAC)字符串。可用的 MACs 是 hmac-md5,hmac-md5-96,hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512。如果没有使用 属性，则会加载所有可用的 MAC。



注意

如果您计划使用 RSA 或 DSA 以外的任何算法，请确保将应用服务器设置为使用 Bouncy Castle JCE 库。

KIE 服务器节点和流程自动化管理器控制器

使用以下属性配置来自 Process Automation Manager 控制器的 KIE 服务器节点的连接：

- org.kie.server.controller ：URL 用于连接到 Process Automation Manager 控制器。例如，ws://localhost:8080/business-central/websocket/controller。

- org.kie.server.user: 用于从 Process Automation Manager 控制器连接到 KIE 服务器节点的用户名。只有在将此 Business Central 安装用作 Process Automation Manager 控制器时才需要此属性。

-

org.kie.server.pwd: 用于从 Process Automation Manager 控制器连接到 KIE 服务器节点的密码。只有在将此 Business Central 安装用作 Process Automation Manager 控制器时才需要此属性。

Maven 和 miscellaneous

使用以下属性配置 Maven 和其他其它功能：

- **kie.maven.offline.force ：强制 Maven 的行为如离线一样。如果为 true，禁用在线依赖关系解析。默认值： false。**

注意

仅在 Business Central 中使用此属性。如果您与任何其他组件共享一个运行时环境，请隔离配置并将其应用到 Business Central。

- **org.uberfire.gzip.enable: Enables 或在 GzipFilter 压缩过滤器中禁用 Gzip 压缩。默认值： true.**

- **org.kie.workbench.profile: 选择 Business Central 配置集。可能的值有 FULL 或 PLANNER_AND_RULES。一个前缀 FULL_ 设定配置集，并从管理员首选项中隐藏配置集首选项。默认值： FULL**

- **org.appformer.m2repo.url: Business Central 在查找依赖项时使用 Maven 存储库的默认位置。它定向到 Business Central 中的 Maven 存储库，例如 http://localhost:8080/business-central/maven2。在启动 Business Central 前设置此属性。默认值：到内 m2 存储库的文件路径。**

- **appformer.ssh.keystore ：通过指定类名称定义要与 Business Central 搭配使用的自定义 SSH 密钥存储。如果 属性不可用，则使用默认的 SSH 密钥存储。**

- **appformer.ssh.keys.storage.folder ：在使用默认 SSH 密钥存储时，此属性为用户的 SSH 公钥定义存储文件夹。如果属性不可用，则密钥将存储在 Business Central .security 文件夹中。**

- **appformer.experimental.features: 启用实验功能框架。默认值： false。**

- org.kie.demo ：启用来自 GitHub 的演示应用程序的外部克隆。

- org.uberfire.metadata.index.dir ：存储 Lucene .index 目录的位置。默认值：工作目录。

- org.uberfire.ldap.regex.role_mapper: Regex 模式用于将 LDAP 主体名称映射到应用程序角色名称。请注意，在匹配原则值和角色名称时，变量角色必须是模式的一部分，因为应用程序角色名称替换变量角色。

- org.uberfire.sys.repo.monitor.disabled ：禁用配置监控器。除非确定，否则不要禁用。默认值： false。

- org.uberfire.secure.key ：密码加密使用的密码。默认值： org.uberfire.admin。

- org.uberfire.secure.alg: Crypto 算法由密码加密使用。默认值：P BEWithMD5AndDES.

- org.uberfire.domain: uberfire 使用的 Security-domain name。默认值：ApplicationRealm.

- org.guvnor.m2repo.dir: Place 存储 Maven 存储库文件夹。默认值：&lt ;working-directory>/repositories/kie。

- org.guvnor.project.gav.check.disabled: Disables group ID、工件 ID 和版本(GAV)检查。默认值： false。

- org.kie.build.disable-project-explorer ：禁用 Project Explorer 中所选项目的自动构建。默认值： false。

- org.kie.builder.cache.size ：定义项目构建器的缓存大小。默认值 ：20。

- org.kie.library.assets_per_page:您可以在项目屏幕中自定义每个页面的资产数量。默认值 ：15。

- org.kie.verification.disable-dtable-realtime-verification: 禁用决策表的实时验证和验证。默认值： false。

处理自动化管理器控制器

使用以下属性来配置如何连接到 Process Automation Manager 控制器：

- org.kie.workbench.controller: 用于连接到 Process Automation Manager 控制器的 URL，例如： ws://localhost:8080/kie-server-controller/websocket/controller。
- org.kie.workbench.controller.user: Process Automation Manager 控制器用户。默认值： kieserver.
- org.kie.workbench.controller.pwd ： Process Automation Manager 控制器密码。默认值： kieserver1!.
- org.kie.workbench.controller.token ：用于连接 Process Automation Manager 控制器的令牌字符串。

Java Cryptography Extension KeyStore(JCEKS)

使用以下属性配置 JCEKS：

- kie.keystore.keyStoreURL ：用于加载 Java Cryptography Extension KeyStore(JCEKS)的 URL。例如，file:///home/kie/keystores/keystore.jceks。
- kie.keystore.keyStorePwd ：用于 JCEKS 的密码。
- kie.keystore.key.ctrl.alias ：默认 REST Process Automation Manager 控制器的密钥别名。
- kie.keystore.key.ctrl.pwd ：默认 REST Process Automation Manager 控制器的别名密码。

渲染

使用以下属性在 Business Central 和 KIE 服务器呈现的形式间切换：

- org.jbpm.wb.forms.renderer.ext ：切换 Business Central 和 KIE 服务器之间的表单渲染。默认情况下，表单渲染由 Business Central 执行。默认值： false。

- org.jbpm.wb.forms.renderer.name: 可让您在 Business Central 和 KIE 服务器呈现的表单之间进行切换。默认值： workbench.

第 41 章 与 BUSINESS CENTRAL 相关的性能调优注意事项

以下关键概念或建议做法可以帮助您优化 Business Central 配置和红帽决策管理器的性能。这些概念在本章节中进行了概述，在适用的情况下，会详细介绍相关文档。本节将根据需要在 Red Hat Decision Manager 的新版本时扩展或更改。

确保在开发过程中启用了开发模式

您可以将 Business Central 中的 KIE 服务器或特定项目设置为使用 生产 模式或 开发模式。默认情况下，Business Central 中的 KIE 服务器和所有新项目均为开发模式。这个模式提供有助于开发体验的功能，如灵活的项目部署策略以及优化开发期间 KIE 服务器性能的功能，如禁用的 GAV 检测。使用开发模式，直到红帽决策管理器环境建立并完全为生产模式做好准备。

有关配置环境模式或重复的 GAV 检测的详情，请查看以下资源：

- *第 36 章 在 KIE 服务器和 Business Central 中配置环境模式*
- *打包并部署 Red Hat Decision Manager 项目*

禁用复杂指导决策表的验证和验证

Business Central 的决策表验证和验证功能默认启用。这个功能可帮助您验证您的指导的决定表，但使用复杂指导的决策表，此功能可以隐藏决策引擎性能。您可以通过将 org.kie.verification.disable-dtable-realtime-verification 系统属性值设为 true 来禁用此功能。

有关指导决策表验证的更多信息，*请参阅使用指导决策表设计决策服务*。

如果您有很多大型项目，禁用自动构建

在 Business Central 中，当您在 Project Explorer 侧面板中的项目间导航时，所选项目会自动构建，以便 Alerts 窗口被更新来显示项目的构建错误。如果您在进行活跃开发中的很多项目之间有大型项目或频繁切换，则此功能可以隐藏业务中心和决策引擎性能。

要禁用自动项目构建，将 org.kie.build.disable-project-explorer 系统属性设置为 true。

部分 III. 在 BUSINESS CENTRAL 中使用独立视角

作为业务规则开发人员，您可以嵌入来自 Web 应用程序业务中心的独立视角，然后使用它们编辑规则、流程、决策表和其他资产。

先决条件

- Business Central 已部署并在 Web/应用程序服务器中运行。
- 您已登录到 Business Central。

第 42 章 BUSINESS CENTRAL 中的独立视角

业务中心根据资产的格式为编写资产提供专用编辑器。Business Central 具有可让您单独使用这些编辑器的功能。这个功能被称为编辑器的独立视角模式，或者只是 *独立视角*。

作为业务规则开发人员，您可以在 Web 应用程序中嵌入独立视角，然后使用它编辑规则、流程、决策表和其他资产。嵌入了视角后，您可以在不切换到 Business Central 的情况下编辑自己的应用程序中的资产。您可以使用此功能自定义 Web 应用程序。除了独立视角外，您还可以在应用程序中嵌入独立自定义页面（仪表板）。

您可以通过在带有独立和视角参数的浏览器中使用特定的 Web 地址来访问 独立 视角。独立视角的 Web 地址也可以包含其他参数。

第 43 章 使用独立库视角

您可以使用 Business Central 的库视角来选择您要编辑的项目。您还可以对所选项目执行所有创作功能。

独立库透视图可以通过两种方式使用，*使用* header=UberfireBreadcrumbsContainer *参数。区别在于，在库视角的顶部会显示带有 标头 参数的地址的面包。使用这个链接，您可以为 项目 创建额外的空间。*

流程

1. *登录到 Business Central。*
2. *在网页浏览器中输入适当的 Web 地址：*
   a. *在不使用 标头 参数 的情况下访问 独立库视角*

      http://localhost:8080/business-central/kie-wb.jsp?standalone=true&perspective=LibraryPerspective

      在浏览器中打开 standalone 库视角，没有面包对面包的面包。
   b. *使用 标头 参数访问独立库视角*

      http://localhost:8080/business-central/kie-wb.jsp?standalone=true&perspective=LibraryPerspective&header=UberfireBreadcrumbsContainer

      在浏览器中打开带有面包的面包（面包对面包）的单机库视角。

第 44 章 使用独立编辑器视角

您可以使用 Business Central 的独立编辑器视角访问资产的特定编辑器。使用这个视角，您可以打开资产的编辑器，并可以根据需要修改资产。

访问资产的独立编辑器视角的 Web 地址包含 独立 和 路径 参数。path 参数必须包含到资产的完整路径，Web 地址可以以 #StandaloneEditorPerspective 字符串结尾。另外，根据不同的 path 参数，您可以在独立模式下访问特定资产的编辑器。

流程

1. 登录到 Business Central。
2. 在 Web 浏览器中，根据需要输入适当的 Web 地址，例如：
   a. 要编辑进程：

      ```
      http://localhost:8080/business-central/kie-wb.jsp?standalone&path=default://master@MySpace/Shop/src/main/resources/com/purchase.bpmn#StandaloneEditorPerspective
      ```

      Process Designer 在独立模式中打开。

   b. 要编辑表单：

      ```
      http://localhost:8080/business-central/kie-wb.jsp?standalone&path=default://master@MySpace/Mortgage_Process/src/main/resources/ApplicationMortgage.frm#StandaloneEditorPerspective
      ```

      Form Modeler 在独立模式中打开。

第 45 章 使用独立内容管理器视角

通过使用应用程序中的独立内容管理器视角，您可以创建并编辑应用程序的内容及其导航菜单。

流程

1. 登录到 Business Central。
2. 在 Web 浏览器中，在地址栏中输入以下 Web 地址：

   ```
   http://localhost:8080/business-central/kie-wb.jsp?
   standalone=true&perspective=ContentManagerPerspective
   ```

   独立内容管理器透视图在浏览器中打开。

第 46 章 使用独立自定义页面（仪表板）

除了独立视角外，您还可以在应用程序中嵌入自定义页面（也称为仪表板）。要从应用程序访问自定义页面，请提供自定义页面的名称作为 **perspective** 参数的值。请注意，**perspective** 参数区分大小写。

流程

1. 登录到 **Business Central**。
2. 在 **Web** 浏览器中，在地址栏中输入自定义页面的 **Web** 地址，例如：

```
http://localhost:8080/business-central/kie-wb.jsp?
standalone=true&perspective=CustomPageName
```

独立自定义页面会在浏览器中打开。将值 **CustomPageName** 替换为您要在独立模式中使用的自定义页面的名称。

部分 IV. 在 BUSINESS CENTRAL 中创建自定义页面

作为业务分析员或业务规则开发人员，您可以使用 Business Central 中的页面和动态仪表板来显示有关项目的特定信息。仪表板是包含至少一个动态报告组件的页面集合。您可以定义数据集，以向仪表板的报告组件提供。您可以将仪表板导出到 Red Hat JBoss EAP 上的独立 Dashbuilder Runtime dashboard viewer 中，或 Red Hat OpenShift Container Platform 上的 Dashbuilder Standalone dashboard viewer。

先决条件

- 以有权编辑页面的用户身份登录到 Business Central。

第 47 章 BUSINESS CENTRAL 自定义仪表板

仪表板是 Business Central 页面的集合，其中至少包含一个报告组件。仪表板通常包含数据集、导航树和权限。

- 数据集编写：定义用于访问数据的数据集，并通过页面显示数据。如需更多信息，请参阅添加数据集。
- 安全管理 - 在这个阶段，设置角色和组权限，用于定义用户在工作 Business Central 时授予用户的特权。如需更多信息，请参阅 安全管理。

其他资源

要从以前版本的 Business Central 迁移仪表板，请使用 第 50.1 节 “导出仪表板数据” 中描述的 Dashbuilder Data Transfer 功能。

第 48 章 数据集编写

数据集是相关信息的集合，可以存储在数据库中、Microsoft Excel 文件中的或内存中。数据集定义指示 Business Central 方法访问、读取和解析数据集。Business Central 不会存储数据。它允许您定义数据集的访问权限，无论数据存储位置。

例如，如果数据存储在数据库中，则有效数据集可包含整个数据库，或作为 SQL 查询结果的数据库子集。在这两种情况下，数据都用作报告业务中心组件的输入，然后显示该信息。

要访问数据集，您必须创建并注册数据集定义。数据集定义指定数据集的位置、访问它的选项、对其进行解析以及其中包含的列。

注意

Data Sets 页面仅对具有 admin 角色的用户可见。

48.1. 添加数据集

您可以创建数据集，以从外部数据源获取数据，并将该数据用于报告组件。

流程

1. 在 Business Central 中，前往 Admin → Data Sets。

   这时将打开 Data Sets 页面。

2. 点 New Data Set 并选择以下供应商类型之一：

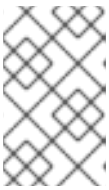
   - bean： 从 Java 类生成数据集
   - CSV： 从远程或本地 CSV 文件中生成数据集
   - SQL： 从 ANSI-SQL 兼容数据库生成数据集

- *Elastic Search：从 Elastic Search 节点生成数据集*
- *Prometheus：使用 Prometheus 查询生成数据集*
- *Kafka：使用 Kafka 代理、消费者或制作者中的指标生成数据集*

注意

您必须为 Prometheus、Kafka 和 Execution Server 选项配置 KIE 服务器。

3. *完成 Data Set Creation Wizard 并点 Test。*



注意

配置步骤根据您选择的供应商的不同而有所不同。

4. *点击 Save。*

48.2. 编辑数据集

您可以编辑现有的数据集，以确保获取到报告组件的数据为最新版本。

流程

1. *在 Business Central 中，前往 Admin → Data Sets。*

   这时将打开 Data Set Explorer 页面。

2. *在 Data Set Explorer 窗格中，搜索您要编辑的数据集合，选择数据集，然后单击 Edit。*

3. *在 Data Set Editor 窗格中，根据需要使用适当的标签页编辑数据。该选项卡会根据您选择的*

数据集供应商类型而有所不同。

例如，以下更改可用于编辑 CSV 数据供应商：

- CSV 配置： 使您能够更改数据集定义的名称、源文件、分隔符和其他属性。
- preview ： 允许您预览数据。在 CSV Configuration 选项卡中点 Test 后，系统会执行数据集查找调用，如果数据可用，会出现一个预览。请注意， Preview 选项卡有两个子选项卡：
  - data 列： 允许您指定数据设置定义中的哪些列。
  - filter： 允许您添加新过滤器。
- 高级： 使您能够管理以下配置：
  - 缓存： 请参阅缓存数据 以了解更多信息。
  - 通过 缓存生命周期， 您可以指定数据设置（或数据）被刷新的时间间隔。当 后端数据 改变时， 刷新缓存的数据功能。

4. 进行必要的更改后， 单击 Validate。
5. 点击 Save。

48.3. 数据刷新

通过数据刷新功能， 您可以指定数据设置（或数据）被刷新的时间间隔。您可以在数据集 的高级标签页中访问数据刷新每个 功能。当 后端数据 改变时， 刷新缓存的数据功能。

48.4. 缓存数据

Business Central 提供使用内存数据存储数据收集和执行数据操作的缓存机制。缓存数据减少了网络

流量、远程系统有效负载和处理时间。为避免性能问题，请在 Business Central 中配置缓存设置。

对于生成数据集合的任何数据查找调用，缓存方法决定执行数据查找调用的位置，以及保存生成的数据集的位置。例如，数据查找调用的示例将是本地参数设置为"Urban"的所有影片应用程序。

Business Central 数据集功能提供了两个缓存级别：

- 客户端级别
- 后端级别

您可以在数据集 的高级 标签页上设置客户端缓存和后端缓存设置。

客户端缓存

当缓存打开后，数据集会在查找操作期间在 Web 浏览器中缓存，并进一步查找操作不会对后端执行请求。在 Web 浏览器中处理数据设置操作，如分组、聚合、过滤和排序。仅在数据集大小小时启用客户端缓存，例如：数据的大小小于 10 MB 的数据集。对于大型数据集，会出现浏览器问题，如性能缓慢或间歇的空闲状态。客户端缓存可减少包括对存储系统的请求的请求数量。

后端缓存

启用缓存后，决定引擎缓存数据集。这可减少到远程存储系统的后端请求数量。所有数据收集操作都是使用内存数据在决策引擎中执行的。仅在数据集大小没有频繁更新时启用后端缓存，并可在内存中存储和处理。在对远程存储出现低延迟连接问题的情况下，使用后端缓存也很有用。

注意

在 Data Set Editor 的 Advanced 选项卡中，后端缓存设置并不总是可见，因为 Java 和 CSV 数据供应商依赖于后端缓存（必须位于内存中的数据设置）以使用内存决策引擎来解决任何数据查找操作。

第 49 章 安全管理

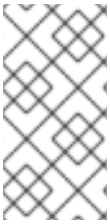
安全管理是管理用户、组和权限的过程。您可以从 Business Central Security 管理页面控制对 Business Central 资源的访问和功能。

Business Central 为安全管理定义了三种类型的实体：用户、组和角色。您可以为角色和组分配权限。用户从用户所属的组和角色继承权限。

49.1. 安全管理供应商

在安全管理上下文中，域限制访问不同的应用资源。realm 包含有关用户、组、角色和权限的信息。特定域的 concrete 用户和组管理服务实施称为安全管理提供程序。

如果内置的安全管理提供程序没有满足应用程序安全性域的要求，您可以构建和注册自己的安全管理供应商。



注意

如果没有安装安全管理提供程序，则管理安全域的用户界面将不可用。在安装和配置安全管理提供程序后，用户和组管理功能会在安全管理用户界面中自动启用。

Business Central 包括红帽 JBoss EAP 安全管理提供商，它支持基于 application-users.properties 或 application-roles.properties 属性文件的内容的域类型。

49.1.1. 根据属性文件配置红帽 JBoss EAP 安全管理提供程序

您可以构建和注册您自己的红帽 JBoss EAP 安全管理供应商。要基于属性文件使用红帽 JBoss EAP 安全管理提供程序，请完成以下步骤。

先决条件

- 安装了红帽 JBoss EAP。

流程

1. 要使用红帽 JBoss EAP 实例中的现有用户或角色属性，请在

EAP_HOME/standalone/configuration/application-users.properties 和 EAP_HOME/standalone/application-roles.properties 文件中包含以下系统属性，如下例所示：

```
<property name="org.uberfire.ext.security.management.wildfly.properties.realm"
value="ApplicationRealm"/>
<property name="org.uberfire.ext.security.management.wildfly.properties.users-file-path"
value="/standalone/configuration/application-users.properties"/>
<property name="org.uberfire.ext.security.management.wildfly.properties.groups-file-path"
value="/standalone/configuration/application-roles.properties"/>
```

下表提供了这些属性的描述和默认值：

表 49.1. 基于属性文件的红帽 JBoss EAP 安全管理供应商

属性	描述	默认值
org.uberfire.ext.security.management.wildfly.properties.realm	域的名称。这个属性不是必须的。	**ApplicationRealm**
org.uberfire.ext.security.management.wildfly.properties.users-file-path	用户属性文件的绝对路径。这个属性是必需的。	**./standalone/configuration/application-users.properties**
org.uberfire.ext.security.management.wildfly.properties.groups-file-path	groups 属性文件的绝对文件路径。这个属性是必需的。	**./standalone/configuration/application-roles.properties**

2. *在应用程序的根目录中创建 security-management.properties 文件。例如，创建以下文件：*

```
src/main/resources/security-management.properties
```

3. *在 security-management.properties 文件中输入以下系统属性和安全供应商名称作为值：*

```
<property name="org.uberfire.ext.security.management.api.userManagementServices"
value="WildflyUserManagementService"/>
```

49.1.2. 根据属性文件和 CLI 模式配置红帽 JBoss EAP 安全管理供应商

要基于属性文件和 CLI 模式使用红帽 JBoss EAP 安全管理提供程序，请完成以下步骤。

先决条件

- *安装了红帽 JBoss EAP。*

流程

1. *要使用红帽 JBoss EAP 实例中的现有用户或角色属性，请在 EAP_HOME/standalone/configuration/application-users.properties 和 EAP_HOME/standalone/application-roles.properties 文件中包含以下系统属性，如下例所示：*

```
<property name="org.uberfire.ext.security.management.wildfly.cli.host" value="localhost"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.port" value="9990"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.user" value="
<USERNAME>"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.password" value="
<USER_PWD>"/>
<property name="org.uberfire.ext.security.management.wildfly.cli.realm"
value="ApplicationRealm"/>
```

下表提供了这些属性的描述和默认值：

表 49.2. 基于属性文件和 CLI 模式的红帽 JBoss EAP 安全管理供应商

属性	描述	默认值
org.uberfire.ext.security.management.wildfly.cli.host	原生管理接口主机。	**localhost**
org.uberfire.ext.security.management.wildfly.cli.port	原生管理接口端口。	**9990**
org.uberfire.ext.security.management.wildfly.cli.user	原生管理接口用户名。	不适用
org.uberfire.ext.security.management.wildfly.cli.password	原生管理接口用户的密码。	不适用
org.uberfire.ext.security.management.wildfly.cli.realm	供应用的安全上下文使用的域。	**ApplicationRealm**

2. *在应用程序的根目录中创建 security-management.properties 文件。例如，创建以下文件：*

```
src/main/resources/security-management.properties
```

3. 在 security-management.properties 文件中输入以下系统属性和安全供应商名称作为值：

```
<property name="org.uberfire.ext.security.management.api.userManagementServices"
value="WildflyCLIUserManagementService"/>
```

49.2. 权限和设置

权限是授权用户，用于执行与应用程序内特定资源相关的操作。例如，用户可以具有以下权限：

- 查看页面。
- 保存项目。
- 查看存储库。
- 删除仪表板

您可以授予或拒绝权限，并且权限可以特定于全局或资源。您可以使用权限来保护资源的访问，并自定义应用程序中的功能。

49.2.1. 在 Business Central 中更改组和角色的权限

在 Business Central 中，您无法更改个人用户的权限。但是，您可以更改组和角色的权限。更改的权限适用于具有角色或属于您更改的组的用户。

注意

对角色或组进行的任何更改会影响与该角色或组关联的所有用户。

先决条件

- 使用 admin 用户角色登录到 Business Central。

流程

1. *要访问 Business Central 中的 安全管理 页面，可选择屏幕右上角的 Admin 图标。*

2. *单击 Business Central Settings 页面中的 Roles、Groups 或 Users。*

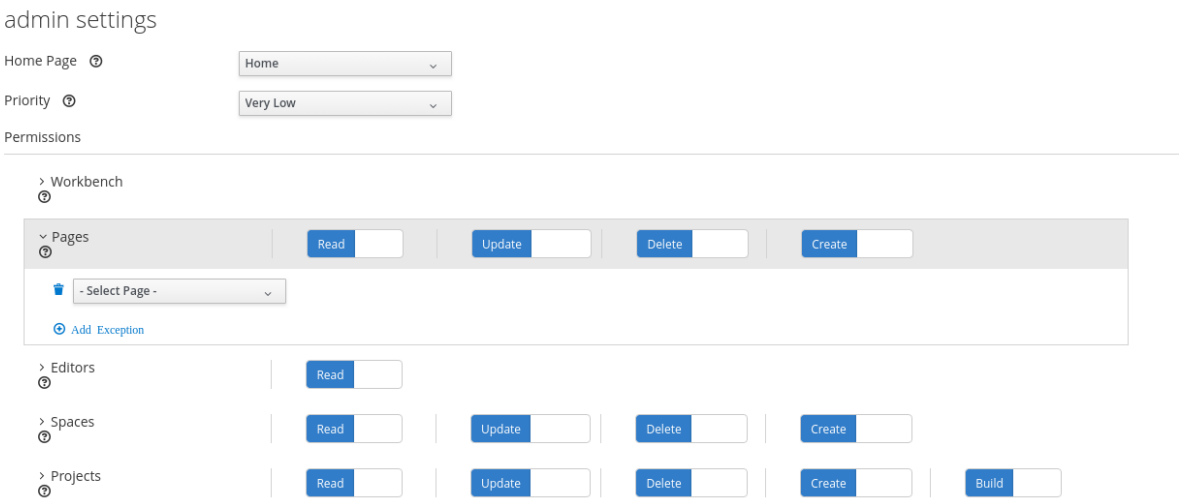
   Security 管理页面 会在您点击的图标的标签页中打开。

3. *在列表中点击您要编辑的角色或组。所有详情都显示在右侧面板中。*

4. *在 Settings 部分下设置 主页 或 优先级。*

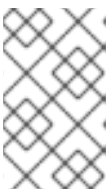
5. *在 Permissions 部分设置 Business Central、页面、编辑器、空格和项目权限。*

图 49.1. 设置权限



6. *单击资源类型旁边的箭头，以展开您要更改权限的资源类型。*

7. *可选：要添加资源类型的异常，请单击 Add Exception，然后根据需要设置权限。*



注意

您不能在 Business Central 资源类型中添加例外。

8. 点击 **Save**。

49.2.2. 更改 Business Central 主页

主页是您登录 Business Central 后出现的页面。默认情况下，主页设置为 **Home**。您可以为每个角色和组指定不同的主页。

流程

1. 在 Business Central 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Roles** 或 **Groups**。
2. 选择角色或组。
3. 从 **Home Page** 列表中选择一个页面。
4. 点击 **Save**。

注意

角色或组必须具有对页面的读访问权限，然后才能使它成为主页。

49.2.3. 设置优先级

用户可以具有多个角色，并属于多个组。**Priority** 设置决定了角色或组的优先级顺序。

先决条件

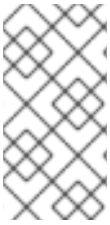
- 使用 **admin** 用户角色登录到 **Business Central**。

流程

1. 在 Business Central 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Roles** 或 **Groups**。

2. 选择角色或组。

3. 从优先级菜单中选择优先级，然后单击 Save。



注意

如果用户具有具有冲突设置的角色或属于某一组，则应用具有最高优先级的角色或组的设置。

第 50 章 导出、导入和部署仪表板

在 Business Central 中创建仪表板后，您可以导出仪表板数据，并将其导入到 Business Central 的另一个实例、Dashbuilder Runtime 或 Dashbuilder Standalone。

注意

这个功能只能由管理员用户访问。

50.1. 导出仪表板数据

您可以将 Business Central 中所有仪表板数据（如来自 Business Central）导出为 ZIP 文件。

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，再选择 Dashbuilder Data Transfer。
2. 在 Dashbuilder Data Transfer 页面上，单击 Export all。

下载包含所有仪表板数据的 export.zip 文件。export.zip 文件结构由数据类型分隔，如下例所示：

```
dashbuilder/datasets/definitions/dataset-example1.dset
dashbuilder/datasets/definitions/dataset-example2.dset
dashbuilder/datasets/readme.md
VERSION
```

50.2. 导入 BUSINESS CENTRAL 仪表板

如果归档的结构与以下示例相同，您可以从 ZIP 文件中导入 Dashbuilder 数据：

```
dashbuilder/datasets/definitions/dataset-example1.dset
dashbuilder/datasets/definitions/dataset-example2.dset
dashbuilder/datasets/readme.md
VERSION
```

流程

1. 在 Business Central 中，选择屏幕右上角的 Admin 图标，再选择 Dashbuilder Data Transfer。

警告

您应该仅导入仪表板数据到红帽决策管理器清洁安装，以避免覆盖现有系统中的数据。

2. 在 Dashbuilder Data Transfer 页面上，单击 Choose File 图标。

3. 导航到您要导入的 ZIP 文件并选择文件。

4. 点 Upload 图标。

5. 点 Import。

附录 A. 版本信息

文档最新更新于 2023 年 2 月 1 日（周三）。

附录 B. 联系信息

Red Hat Decision Manager 文档团队 ：brms-docs@redhat.com