Red Hat Process Automation Manager 7.12

部署和管理红帽流程自动化管理器服务

Last Updated: 2023-04-08

Red Hat Process Automation Manager 7.12 部署和管理红帽流程自动化管理器服务

法律通告

摘要

本文档论述了如何使用 Business Central 界面或使用 KIE API 部署和管理红帽流程自动化管理器项目和资产。

目录

前言

作为业务决策和流程的开发人员，您必须将开发的 Red Hat Process Automation Manager 项目部署到 KIE 服务器，以便开始使用您在 Red Hat Process Automation Manager 中创建的服务。您可以使用 Business Central 接口或使用 KIE API 部署和管理红帽流程自动化管理器项目和资产。

使开源包含更多

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

部分 I. 打包和部署 RED HAT PROCESS AUTOMATION MANAGER 项目

作为业务规则开发人员，您必须将开发的红帽流程自动化管理器项目构建和部署至 KIE 服务器，以便开始使用您在 Red Hat Process Automation Manager 中创建的服务。您可以从 Business Central 开发和部署项目，从独立的 Maven 项目、Java 应用程序或使用多种平台的组合开发和部署。例如，您可以在 Business Central 中开发一个项目并使用 KIE Server REST API 进行部署，或者在配置了 Business Central 的 Maven 中开发项目并使用 Business Central 进行部署。

先决条件

- 要部署的项目已进行开发和测试。对于 Business Central 中的项目，请考虑使用测试场景测试项目中的资产。例如，请参阅使用 *测试场景* *测试决策服务*。

第 1 章 RED HAT PROCESS AUTOMATION MANAGER 项目打包

Red Hat Process Automation Manager 项目包含您在红帽流程自动化管理器中开发的业务资产。Red Hat Process Automation Manager 中的每个项目都打包为知识 JAR (KJAR)文件，文件包括 Maven 项目对象模型文件(**pom.xml**)，其中包含项目的构建、环境和 KIE 模块描述符文件(**kmodule.xml**)，其中包含项目中资产的 KIE 基础和 KIE 会话配置。您可以将打包的 KJAR 文件部署到运行决策服务、处理应用程序和其他可部署资产（统称为 *服务*）的 KIE 服务器。这些服务在运行时通过实例化 KIE 容器 *或部署单元* 使用。项目 KJAR 文件存储在 Maven 存储库中，由三个值识别： **GroupId**、**ArtifactId** 和 **Version** (GAV)。对于可能需要部署的每个新版本，**Version** 值必须是唯一的。要识别工件（包括 KJAR 文件），您需要所有三个 GAV 值。

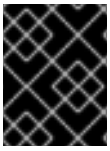
在构建和部署项目时，Business Central 中的项目会自动打包。对于 Business Central 之外的项目，如 Java 应用程序内的独立 Maven 项目或项目，您必须在附加的 **kmodule.xml** 文件中配置 KIE 模块描述符设置，或直接在 Java 应用程序中构建和部署项目。

第 2 章 BUSINESS CENTRAL 中的项目部署

您可以使用 Business Central 开发业务资产和服务，并管理为项目部署配置的 KIE 服务器实例。开发项目后，您可以在 Business Central 中构建项目，并将其自动部署到 KIE Server。要启用自动部署，Business Central 包含内置 Maven 存储库。从 Business Central 中，您可以启动、停止或删除包含您构建和部署的服务及其项目版本的部署单元(KIE 容器)。

您还可以将多个 KIE 服务器连接到同一个 Business Central 实例，并将它们分组到不同的服务器配置(**菜单 → Deploy → Execution Servers**)。属于同一服务器配置的服务器运行相同的服务，但您可以在不同的配置中部署不同的项目或不同版本的项目。

例如，您可以在 **Production** 配置中的 **Test** configuration and production 服务器中测试服务器。随着您在项目中开发业务资产和服务，您可以在 **Test** 服务器配置上部署项目。当项目版本经过充分测试后，您可以将其部署到 **Production** 服务器配置中。在这种情况下，若要保持开发项目，可在项目设置中更改版本。然后，新版本和旧版本被视为内置 Maven 存储库中的不同工件。您可以在 **Test** 服务器配置上部署新版本，并在 **Production** 服务器配置中继续运行旧版本。此部署过程简单，但存在显著限制。值得注意的是，没有足够的访问控制：开发人员可以直接将项目部署到生产环境中。



重要

您不能使用 Business Central 将 KIE 服务器移动到不同的服务器配置中。您必须更改服务器的配置文件，以更改服务器的服务器配置名称。

2.1. 配置 KIE 服务器以连接到 BUSINESS CENTRAL

警告

本节提供了可用于测试目的的示例设置。其中一些值不适用于生产环境，并被标记为此类值。

如果您的红帽流程自动化管理器环境中没有配置 KIE 服务器，或者需要在 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 Process Automation Manager 环境* 以将其配置为连接到 Business Central。

先决条件

- Business Central 和 KIE 服务器安装在红帽 JBoss EAP 安装的基础目录(***EAP_HOME***)中。

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，我们只使用一个名为 **controllerUser** 的用户，其中包含 **rest-all** 和 **kie-server** 角色。但是，如果您在同一服务器上安装 KIE Server 和 Business Central，例如在开发环境中，请在共享 **standalone-full.xml** 文件中进行更改，如本节所述。

- 存在具有以下角色的用户：
  - 在 Business Central 中，角色为 **rest-all**的用户
  - 在 KIE 服务器上，具有角色 **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，请转到 **$EAP_HOME/standalone/configuration/standalone-full.xml**。
2. 打开 **standalone-full.xml** 文件，并在 &lt **;system-properties&** gt; 标签下设置以下 JVM 属性：

表 2.1. 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**	具有角色 **rest-all** 的用户名，可以登录到 Business Central。
org.kie.server.controller.pwd	**controllerUser1234;**	可以登录到 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 Server API 的 URL。

表 2.2. Business Central 实例的 JVM 属性

属性	值	备注
org.kie.server.user	**controllerUser**	带有角色 **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 服务器中运行红帽流程自动化管理器的更多信息，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. 验证注册是否成功：

   a. 登录 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。

2.2. 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** 模式 **或开发** 模式下运行。开发模式提供了一个灵活的部署策略，可让您在维护活跃进程实例以进行小更改的同时更新现有部署单元(KIE 容器)。它还允许您在更新活跃进程实例以进行较大的更改前重置部署单元状态。生产模式是生产环境中的最佳选择，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），或者点击 **Redeploy** 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，Business Central 中的 **Redeploy** 选项被禁用，您只能点 **Deploy** 将构建的 KJAR 文件部署到 KIE 服务器上新的部署单元(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，请转至项目 **Settings** → **General Settings** → **Version** 并切换 **Development Mode** 选项。

注意

默认情况下，KIE 服务器和 Business Central 中的所有新项目都处于开发模式中。

您不能部署打开 **Development 模式** 的项目，或使用手动将 **SNAPSHOT** 版本后缀添加到生产模式的 KIE 服务器中。

2.3. 为 BUSINESS CENTRAL 和 KIE 服务器配置外部 MAVEN 存储库

您可以将 Business Central 和 KIE 服务器配置为使用外部 Maven 存储库，如 Nexus 或 Artifactory，而不是内置的存储库。这可让 Business Central 和 KIE 服务器访问和下载外部 Maven 存储库中维护的工件。

重要

存储库中的工件不会接收自动安全补丁，因为 Maven 需要工件不可变。因此，缺少已知安全漏洞补丁的工件会保留在存储库中，以避免依赖它们的构建。修补工件的版本号会递增。如需更多信息，请参阅 JBoss Enterprise Maven Repository。

注意

有关在 Red Hat OpenShift Container Platform 上为编写环境配置外部 Maven 存储库的详情，请查看以下文档：

- *使用 Operator 在 Red Hat OpenShift Container Platform 4 上部署 Red Hat Process Automation Manager 环境*
- *使用模板在 Red Hat OpenShift Container Platform 3 上部署 Red Hat Process Automation Manager 环境*

先决条件

- 安装了 Business Central 和 KIE 服务器。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。

流程

1. 使用连接和访问外部存储库的详情，创建一个 Maven **settings.xml** 文件。有关 **settings.xml** 文件的详情，请查看 Maven Settings Reference。
2. 将文件保存到已知位置，例如 **/opt/custom-config/settings.xml**。
3. 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，请转到 **$EAP_HOME/standalone/configuration/standalone-full.xml**。

4. 打开 **standalone-full.xml** 并在 & **lt;system-properties** > 标签下，将 **kie.maven.settings.custom** 属性设置为 **settings.xml** 文件的完整路径名称。例如：

```
<property name="kie.maven.settings.custom" value="/opt/custom-config/settings.xml"/>
```

5. 启动或重新启动 Business Central 和 KIE 服务器.

后续步骤

对于您要导出或作为 KJAR 工件到外部 Maven 存储库的每个 Business Central 项目，您必须在 **pom.xml** 文件中添加存储库信息。具体说明请查看 第 2.4 节 “将 Business Central 项目导出到外部 Maven 存储库”。

2.4. 将 BUSINESS CENTRAL 项目导出到外部 MAVEN 存储库

如果您为 Business Central 和 KIE 服务器配置了外部 Maven 存储库，您必须将存储库信息添加到您要导出的每个 Business Central 项目的 **pom.xml** 文件中，或作为 KJAR 工件推送到该外部存储库。然后，您可以根据需要通过存储库来进度项目 KJAR 文件，以使用 Business Central 或 KIE Server REST API 部署 KJAR 文件。

先决条件

- 已将 Business Central 和 KIE Server 配置为使用外部 Maven 存储库。如果您在内部部署了 Business Central，以了解有关配置外部 Maven 存储库的更多信息，请参阅 第 2.3 节 “为 Business Central 和 KIE 服务器配置外部 Maven 存储库”。如果您在 Red Hat OpenShift Container Platform 上部署了编写环境，请参阅以下文档：
  - *使用 Operator 在 Red Hat OpenShift Container Platform 4 上部署 Red Hat Process Automation Manager 环境*
  - *使用模板在 Red Hat OpenShift Container Platform 3 上部署 Red Hat Process Automation Manager 环境*

流程

1. 在 Business Central 中，进入 **Menu → Design → Projects**，单击项目名称，然后选择项目中的任何资产。
2. 在屏幕左侧的 **Project Explorer** 菜单中，点击 **Customize View** gear 图标，然后选择 **Repository View → pom.xml**。
3. 在 **pom.xml** 文件末尾添加以下设置（在 **</ project>** 关闭标签前面）。该值必须与您在 **settings.xml** 文件中定义的设置对应。

```
<distributionManagement>
<repository>
<id>${maven-repo-id}</id>
<url>${maven-repo-url}</url>
<layout>default</layout>
</repository>
</distributionManagement>
```

4. 单击 **Save** 以保存 **pom.xml** 文件更改。

对您要导出或作为 KJAR 工件到外部 Maven 存储库的每个 Business Central 项目重复此步骤。

2.5. 在 BUSINESS CENTRAL 中构建和部署项目

开发项目后，您可以在 Business Central 中构建项目，并将它部署到配置的 KIE 服务器。Business Central 中的项目在您构建和部署项目时自动打包为 KJARs，包含所有必要的组件。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。
2. 在右上角，单击 **Deploy** 以构建项目并将其部署到 KIE Server。要编译项目而不将它部署到 KIE Server，请单击 **Build**。

注意

您还可以选择 **Build & Install** 选项来构建项目，并将 KJAR 文件发布到配置的 Maven 存储库，而无需部署到 KIE 服务器。在开发环境中，您可以点击 **Deploy** 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），或者点击 **Redeploy** 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，**Redeploy** 选项被禁用，您可以点 **Deploy** only 将构建的 KJAR 文件部署到 KIE 服务器上的新部署单元(KIE 容器)。

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**。要在 Business Central 中为对应项目配置部署行为，请转至 Project **Settings → General Settings → Version** 并切换 **Development Mode** 选项。默认情况下，KIE 服务器和 Business Central 中的所有新项目都处于开发模式中。您不能部署打开 **Development 模式** 的项目，或使用手动将 **SNAPSHOT** 版本后缀添加到生产模式的 KIE 服务器中。

如果只有一个 KIE 服务器连接到 Business Central，或者所有连接的 KIE 服务器都在同一服务器配置中，则项目中的服务将在部署单元(KIE 容器)中自动启动。

如果有多个服务器配置，则在 Business Central 中显示部署对话框，提示您指定服务器和部署详情。

3. 如果显示部署对话框，请验证或设置以下值：
   - **部署单元 Id / Deployment Unit Alias**：验证在 KIE 服务器中运行该服务的部署单元(KIE 容器)的名称和别名。您通常不需要更改这些设置。有关 KIE 容器别名的更多信息，请参阅 第 2.6.3 节 “KIE 容器别名”。
   - **服务器配置**：选择用于部署此项目的服务器配置。稍后，您可以将其部署到其他配置的服务器，而无需重新构建项目。
   - **启动部署单元？** 验证是否选择了此框来启动部署单元(KIE 容器)。如果清除此框，该服务将部署到服务器上，但不启动。

要查看项目部署详情，请单击屏幕顶部的部署横幅中的 **View deployment details**，或者在 **Deploy** 下拉菜单中。这个选项将您定向到 **Menu → Deploy → Execution Servers** 页面。

2.6. BUSINESS CENTRAL 中的部署单元

项目中的服务在运行时会在配置的 KIE 服务器上通过实例化的 KIE 容器 *或部署单元* 使用。当您在 Business Central 中构建和部署项目时，会在配置的服务器中自动创建部署单元。您可以根据需要在 Business Central 中启动、停止或删除部署单元。您也可以从以前构建的项目创建额外的部署单元，并在 Business Central 中配置的现有或新的 KIE 服务器上启动它们。

2.6.1. 在 Business Central 中创建部署单元

作为红帽流程自动化管理器配置的一部分，应该已存在一个或多个部署单元，但是如果不是，您可以从之前在 Business Central 中构建的项目中创建部署单元。

先决条件

- 您要为其创建新的部署单元的项目已在 Business Central 中构建。

流程

1. 在 Business Central 中，前往 **Menu → Deploy → Execution servers**。
2. 在 **Server Configurations** 下，选择现有配置或点 **New Server Configuration** 来创建配置。
3. 在 **Deployment Units** 下，点 **Add Deployment Unit**。
4. 如果需要，在 **Alias** 字段中添加一个别名。
5. 在窗口中的表中，选择一个 GAV，再单击 GAV 旁边的 **Select** 来填充部署单元数据字段。
6. 选择 **Start Deployment Unit?**框立即启动该服务，或者清除该框以便稍后启动该服务。
7. 点 **Finish**。
   该服务的新部署单元会被创建并放置在为这个服务器配置配置的 KIE 服务器上。如果您选择了 **Start Deployment Unit?**，服务将启动。

2.6.2. 在 Business Central 中启动、停止和删除部署单元

启动部署单元时，部署单元中的服务可供使用。如果只有一个 KIE 服务器连接到 Business Central，或者所有连接的 KIE 服务器都在同一服务器配置中，则服务会在部署项目时自动在部署单元中自动启动。如果有多个服务器配置可用，则部署时会提示您指定服务器和部署详情并启动部署单元。但是，您可以在任何时候手动启动、停止或删除 Business Central 中的部署单元，以根据需要管理部署的服务。

流程

1. 在 Business Central 中，前往 **Menu → Deploy → Execution servers**。
2. 在 **Server Configuration** 下，选择一个配置。
3. 在 **Deployment Units** 下，选择一个部署单元。
4. 单击右上角的 **Start**、**停止** 或 **Remove**。要删除正在运行的部署单元，请将其停止，然后将其删除。

2.6.3. KIE 容器别名

KIE 容器（部署单元）的别名是 KIE 服务器实例中的一个代理，可帮助处理同一容器部署的不同版本。您可以将单个别名链接到容器的不同版本。当容器升级时，链接的别名会自动指向容器的新版本。有关创建 KIE 容器别名的详情，请参考 第 2.6.1 节 “在 Business Central 中创建部署单元”。

例如，如果每次部署容器的新版本时，客户端应用都会更改，客户端应用可以指向容器别名。部署新容器版本时，相关的别名会更新，所有请求都会被自动路由到新容器，而无需更改客户端应用。

考虑包含单个进程并使用以下属性的示例项目：

- **GroupId**: org.jbpm
- **artifactId :** my-project
- **Version**: 1.0
- **containerID** ：my-project

当您更新、构建和部署上述项目时，相关的项目会使用最新版本在 KIE 服务器上更新，并包含以下属性：

- **GroupId**: org.jbpm
- **artifactId :** my-project
- **Version**: 2.0

如果要部署项目的最新版本，则需要将 **containerID** 更新为 **my-project2**，因为 **my-project** 容器指向旧版本。

注意

每个项目版本包含不同的 **containerID** 名称。相关的客户端应用程序需要了解它们交互的所有项目版本。

容器别名还帮助您管理 KIE 容器。您可以在创建容器时显式设置容器别名，或根据关联的 **ArtifactId** 名称隐式设置容器别名。如果需要，您可以在多个容器中添加单个别名。如果没有指定容器别名，则项目的 **ArtifactId** 会默认设置为容器别名。

当您为包含不同 **GroupId** 和 **ArtifactId** 名称的多个容器设置别名时，您可以在每次与 KIE 服务器交互时使用相同的别名。

您可以在以下用例中使用容器别名：

- 使用最新版本的 **进程在客户端应用程序上启动新的进程实例**
- **与一个进程的现有特定版本进行交互**
- **在一个进程中的现有任务进行交互**
- **与进程定义镜像和表单交互**

例如，在部署项目的 1.0 版本后，您将向以下 KIE Server REST API 端点发送 POST 请求，以便在项目中启动进程：

/http://localhost:8230/kie-server/services/rest/server/containers/my-project/processes/evaluation/instances

发送的请求从 **org.jbpm:my-project:1.0** 启动一个新进程实例，其中 **my-project** 被定义为容器别名。之后，当您部署项目的 2.0 版本并发送同一请求时，新实例从 **org.jbpm:my-project:2.0** 开始。您可以在不添加 **containerID** 名称的情况下部署最新版本的进程。

2.7. 在 BUSINESS CENTRAL 中编辑项目的 GAV 值

GroupId、**ArtifactId** 和 **Version** (GAV)值标识 Maven 存储库中的项目。当 Business Central 和 KIE 服务器位于同一文件系统上并使用相同的 Maven 存储库时，每次构建新版本的项目都会在存储库中自动更新。但是，如果 Business Central 和 KIE 服务器位于单独的文件系统中，并且使用单独的本地 Maven 存储库，您必须针对任何新版本的项目 GAV 值来更新项目 GAV 值，以确保项目被视为与旧版本不同的工件。

注意

仅用于开发目的，您可以在项目 **Settings → General Settings → Version** 中切换 **Development Mode** 选项，以在项目版本中添加 **SNAPSHOT** 后缀。此后缀指示 Maven 根据 Maven 策略获取新快照更新。不要使用 **Development 模式**，或者为生产环境手动添加 **SNAPSHOT** 版本后缀。

您可以在项目 **Settings** 屏幕中设置 GAV 值。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。
2. 单击项目 **设置** 选项卡。
3. 在 **General Settings** 中，**根据需要修改**组 ID、**工件 ID** 或 **Version** 字段。如果您部署了项目并正在开发新版本，通常您需要提高版本号。

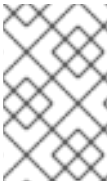
注意

仅用于开发目的，您可以在项目 **Settings → General Settings → Version** 中切换 **Development Mode** 选项，以在项目版本中添加 **SNAPSHOT** 后缀。此后缀指示 Maven 根据 Maven 策略获取新快照更新。不要使用 **Development 模式**，或者为生产环境手动添加 **SNAPSHOT** 版本后缀。

4. 点 **Save** 完成。

2.8. BUSINESS CENTRAL 中重复的 GAV 检测

在 Business Central 中，检查项目中任何重复的 **GroupId**、Art**dfactId** 和 **Version** (GAV)值的所有 Maven 存储库。如果存在 GAV 重复，则执行的操作会被取消。



注意

开发模式 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，请转至项目 **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都会执行重复的 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：

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

2.8.1. 在 Business Central 中管理重复的 GAV 检测设置

具有 **admin** 角色的 Business Central 用户可以修改项目的重复 **GroupId**、Artdfact**Id** 和 **Version** (GAV) 值的软件仓库列表。

注意

开发模式 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，请转至项目 **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**，再单击项目名称。
2. 单击项目 **Settings** 选项卡，然后单击 **Validation** 以打开存储库列表。
3. 选择或清除任何列出的存储库选项，以启用或禁用重复的 GAV 检测。
   以后，只会为验证启用的软件仓库报告重复的 GAV。

注意

要禁用此功能，请在系统启动时将 Business Central 的 **org.guvnor.project.gav.check.disabled** 系统属性设为 **true** ：

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

第 3 章 没有 BUSINESS CENTRAL 的项目部署

作为在 Business Central 界面中开发和部署项目的替代选择，您可以使用独立的 Maven 项目或您自己的 Java 应用程序来开发红帽流程自动化管理器项目，并在 KIE 容器（部署单元）中部署它们到配置的 KIE 服务器。然后，您可以使用 KIE Server REST API 启动、停止或删除包含您构建和部署的服务及其项目的版本的 KIE 容器。通过这一灵活性，您可以继续使用现有应用程序工作流使用红帽流程自动化管理器功能开发业务资产。

在构建和部署项目时，Business Central 中的项目会自动打包。对于 Business Central 之外的项目，如 Java 应用程序内的独立 Maven 项目或项目，您必须在附加的 **kmodule.xml** 文件中配置 KIE 模块描述符设置，或直接在 Java 应用程序中构建和部署项目。

3.1. 配置 KIE 模块描述符文件

KIE 模块是一个 Maven 项目或模块，它带有额外的元数据文件 **META-INF/kmodule.xml**。所有 Red Hat Process Automation Manager 项目都需要一个 **kmodule.xml** 文件才能正确打包和部署。这个 **kmodule.xml** 文件是一个 KIE 模块描述符，用来定义项目中资产的 KIE 基础和 KIE 会话配置。KIE 基础是一个存储库，其中包含 Red Hat Process Automation Manager 中的所有规则、流程和其他业务资产，但不包含任何运行时数据。KIE 会话存储并执行运行时数据，如果您已在 **kmodule.xml** 文件中定义 KIE 会话，则从 KIE 容器创建并直接从 KIE 容器创建。

如果您在 Business Central （如独立的 Maven 项目或 Java 应用程序内创建项目）创建项目，您必须在附加的 **kmodule.xml** 文件中配置 KIE 模块描述符设置，或直接在 Java 应用程序中构建和部署项目。

流程

1. 在项目的 **~/resources/META-INF** 目录中，创建一个至少包含以下内容的 **kmodule.xml** 元数据文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

这个空 **kmodule.xml** 文件足以生成一个默认 KIE 基础，其中包含 **项目资源** 路径下找到的所有文件。默认 KIE 基础也包含一个默认的 KIE 会话，该会话会在构建时在应用程序中创建 KIE 容器时触发。

以下示例是更高级的 **kmodule.xml** 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.drools.org/xsd/kmodule">
  <configuration>
     <property key="drools.evaluator.supersetOf"
value="org.mycompany.SupersetOfEvaluatorDefinition"/>
  </configuration>
  <kbase name="KBase1" default="true" eventProcessingMode="cloud"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">
    <ksession name="KSession1_1" type="stateful" default="true" />
    <ksession name="KSession1_2" type="stateful" default="true" beliefSystem="jtms" />
  </kbase>
  <kbase name="KBase2" default="false" eventProcessingMode="stream"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
```

```
    <fileLogger file="debugInfo" threaded="true" interval="10" />
    <workItemHandlers>
      <workItemHandler name="name" type="new org.domain.WorkItemHandler()" />
    </workItemHandlers>
    <listeners>
      <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener" />
      <agendaEventListener type="org.domain.FirstAgendaListener" />
      <agendaEventListener type="org.domain.SecondAgendaListener" />
      <processEventListener type="org.domain.ProcessListener" />
    </listeners>
  </ksession>
 </kbase>
</kmodule>
```

这个示例定义了两个 KIE 基础。两个 KIE 基础中包含规则资产的 **特定软件包**。当您以这种方式指定软件包时，您必须将规则文件组织到反映指定软件包的文件夹结构中。两个 KIE 会话从 **KBase1** KIE 基础实例化，以及一个来自 **KBase2** 的 KIE 会话。**KBase2** 的 KIE 会话是一个 **无状态** KIE 会话，这意味着之前调用 KIE 会话（上一会话状态）中的数据会在会话调用之间丢弃。该 KIE 会话还指定了一个文件（或控制台）日志记录器、**WorkItemHandler** 和显示三种支持的类型的监听程序： **ruleRuntimeEventListener**、**agendaEventListener** 和 **processEventListener**。& **lt;configuration** > 元素定义可用于进一步自定义 **kmodule.xml** 文件的可选属性。

作为手动将 **kmodule.xml** 文件附加到项目的替代选择，您可以使用 Java 应用程序中的 **KieModuleModel** 实例来以编程方式创建用于定义 KIE 基础和 KIE 会话的 **kmodule.xml** 文件，然后将项目中的所有资源添加到 KIE 虚拟文件系统 **KieFileSystem** 中。

以编程方式创建 kmodule.xml 并将其添加到 KieFileSystem

```
import org.kie.api.KieServices;
import org.kie.api.builder.model.KieModuleModel;
import org.kie.api.builder.model.KieBaseModel;
import org.kie.api.builder.model.KieSessionModel;
import org.kie.api.builder.KieFileSystem;

KieServices kieServices = KieServices.Factory.get();
KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel("KBase1")
  .setDefault(true)
  .setEqualsBehavior(EqualityBehaviorOption.EQUALITY)
  .setEventProcessingMode(EventProcessingOption.STREAM);

KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel("KSession1_1")
  .setDefault(true)
  .setType(KieSessionModel.KieSessionType.STATEFUL)
  .setClockType(ClockTypeOption.get("realtime"));

KieFileSystem kfs = kieServices.newKieFileSystem();
kfs.writeKModuleXML(kieModuleModel.toXML());
```

2. 在手动配置 **kmodule.xml** 文件后，手动或以编程方式在项目中检索 KIE 基础和 KIE 会话，以验证配置：

```
KieServices kieServices = KieServices.Factory.get();
```

```
KieContainer kContainer = kieServices.getKieClasspathContainer();

KieBase kBase1 = kContainer.getKieBase("KBase1");
KieSession kieSession1 = kContainer.newKieSession("KSession1_1"),
   kieSession2 = kContainer.newKieSession("KSession1_2");

KieBase kBase2 = kContainer.getKieBase("KBase2");
StatelessKieSession kieSession3 = kContainer.newStatelessKieSession("KSession2_1");
```

如果 **KieBase** 或 **KieSession** 已在 **kmodule.xml** 文件中配置为 **default="true"**，如前面的 **kmodule.xml** 示例中所示，您可以在不传递任何名称的情况下从 KIE 容器检索它们：

```
KieContainer kContainer = ...

KieBase kBase1 = kContainer.getKieBase();
KieSession kieSession1 = kContainer.newKieSession(),
   kieSession2 = kContainer.newKieSession();

KieBase kBase2 = kContainer.getKieBase();
StatelessKieSession kieSession3 = kContainer.newStatelessKieSession();
```

要增加或减少在决策引擎中缓存的 KIE 模块或工件版本的最大数量，您可以在 Red Hat Process Automation Manager 发行版本中修改以下系统属性的值：

- **kie.repository.project.cache.size** ：在决策引擎中缓存的最大 KIE 模块数。默认值 **：100**
- **kie.repository.project.versions.cache.size**: 在决策引擎中缓存的相同工件的最大版本数。默认值 **：10**

对于 KIE 存储库配置的完整列表，请 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution** ZIP 文件，再导航到 **~/rhpam-7.12.0-sources/src/drools-$VERSION/drools-compiler/src/main/java/org/drools/compiler/compiler/kie/builder/impl/KieRepositorympl.java.java**.

有关 **kmodule.xml** 文件的更多信息，请从 红帽客户门户网站中下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution** ZIP 文件（（如果还没有下载）），并查看 **$FILE_HOME/rhpam-$VERSION-sources/kie-api-parent-$VERSION/kie-api/src/main/resources/org/kie/api/** 中的 **kmodule.xsd** XML schema。

3.1.1. KIE 模块配置属性

您的项目的 KIE 模块描述符文件 (**kmodule.xml**) 中提供了可选的 **<configuration>** 项用于定义属性 **key** 和 **value** 对，可以进一步自定义您的 **kmodule.xml** 文件。

kmodule.xml 文件中的配置属性示例

```
<kmodule>
  ...
  <configuration>
    <property key="drools.dialect.default" value="java"/>
    ...
  </configuration>
  ...
</kmodule>
```

以下是项目的 KIE 模块描述符文件(**kmodule.xml**)中支持的 < **configuration** > 属性键和值：

drools.dialect.default

设置默认 Drools dialect。
支持的值： **java**,**mvel**

```
<property key="drools.dialect.default"
  value="java"/>
```

Drools.accumulate.function.$FUNCTION

将实施积累功能的类链接到指定功能名称，该名称允许您将自定义积累功能添加到决策引擎中。

```
<property key="drools.accumulate.function.hyperMax"
  value="org.drools.custom.HyperMaxAccumulate"/>
```

Drools.evaluator.$EVALUATION

将实施评估器定义的类链接到指定的评估器名称，以便您可以将自定义评估器添加到决策引擎中。评估器与自定义操作器类似。

```
<property key="drools.evaluator.soundslike"
  value="org.drools.core.base.evaluators.SoundslikeEvaluatorsDefinition"/>
```

drools.dump.dir

设置 Red Hat Process Automation Manager **dump/log** 目录的路径。

```
<property key="drools.dump.dir"
  value="$DIR_PATH/dump/log"/>
```

drools.defaultPackageName

为项目中的业务资产设置默认软件包。

```
<property key="drools.defaultPackageName"
  value="org.domain.pkg1"/>
```

drools.parser.processStringEscapes

设置字符串转义函数。如果此属性设为 **false**，则 **\n** 字符将不会解释为换行符。
支持的值： **true** （默认）、**false**

```
<property key="drools.parser.processStringEscapes"
  value="true"/>
```

drools.kbuilder.severity.$DUPLICATE

为构建 KIE 基础时报告的重复规则、进程或功能的实例设置严重性。例如，如果您将 **duplicateRule** 设置为 **ERROR**，则在构建 KIE 基础时为检测到的任何重复规则生成一个错误。
支持的密钥后缀： **duplicateRule**,**duplicateProcess**,**duplicateFunction**

支持的值： **INFO**、**WARNING**、**ERROR**

```
<property key="drools.kbuilder.severity.duplicateRule"
  value="ERROR"/>
```

drools.propertySpecific

设置决策引擎的属性重新活动。
支持的值： **DISABLED**,**ALLOWED**,**ALWAYS**

```
<property key="drools.propertySpecific"
  value="ALLOWED"/>
```

Drools.lang.level

设置 DRL 语言级别。
支持的值： **DRL5**、 **DRL6**、**DRL6_STRICT** （默认）

```
<property key="drools.lang.level"
  value="DRL_STRICT"/>
```

3.1.2. KIE 模块支持的 KIE 基础属性

KIE 基础是一个存储库，您可以在项目的 KIE 模块描述符文件(**kmodule.xml**)中定义，其中包含 Red Hat Process Automation Manager 中的所有规则、流程和其他业务资产。当您在 **kmodule.xml** 文件中定义 KIE 基础时，您可以指定特定的属性和值来进一步自定义 KIE 基本配置。

kmodule.xml 文件中的 KIE 基本配置示例

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" eventProcessingMode="stream" equalsBehavior="equality"
declarativeAgenda="enabled" packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1"
sequential="false">
    ...
  </kbase>
  ...
</kmodule>
```

以下是项目的 KIE 模块描述符文件(**kmodule.xml**)中支持的 **kbase** 属性和值：

表 3.1. KIE 模块支持的 KIE 基础属性

属性	支持的值	描述
name	任何名称	定义从 **KieContainer** 检索 **KieBase** 的名称。*此属性是必需的。*
includes	KIE 模块中其他 KIE 基本对象的逗号分隔列表	定义此 KIE 基础镜像中包含的其他 KIE 基础对象和工件。如果您将其声明为模块的 **pom.xml** 文件中的一个依赖项，则可以在多个 KIE 模块中包含 KIE 基础。

属性	支持的值	描述
软件包	KIE 库中要包含的、以逗号分隔的软件包列表 Default: **all**	定义此 KIE 基础中包含的工件软件包（如规则和流程）。默认情况下，**~/resources** 目录中的所有工件都包含在 KIE 基础中。此属性允许您限制编译的工件数量。仅编译属于此属性中指定的列表的软件包。
default	**true**,**false** 默认：**false**	确定 KIE 基础是否为模块的默认 KIE 基础，以便可以从 KIE 容器创建而无需传递任何名称。每个模块只能有一个默认 KIE 基础。
equalsBehavior	**身份**，**平等** 默认： **identity**	定义当将新事实插入到工作内存时，Red Hat Process Automation Manager 的行为。如果设置为 **identity**，则始终创建新的 **FactHandle**，除非工作内存中已存在的同一对象。如果设置为 **equality**，则仅根据插入的事实的 **equals （）** 方法，才会在新插入的对象不等于现有事实时创建新的 **FactHandle**。当您希望对象根据功能相等而不是显式身份进行评估时，请使用 **平等** 模式。
eventProcessingMode	**云**,**流** 默认： **cloud**	决定如何在 KIE 库中处理事件。如果此属性设置为 **云**，KIE 基础会将事件视为普通事实。如果将此属性设置为 **流**，则允许对事件进行临时原因。
declarativeAgenda	**禁用**，**启用** Default: **disabled**	决定是否启用声明性议程。
sequential	**true**,**false** 默认：**false**	确定是否启用顺序模式。在后续模式中，决策引擎会按照在决策引擎单中列出的顺序评估规则，而不考虑工作内存的变化。如果您使用无状态 KIE 会话，且您不希望执行规则来影响 agenda 中的后续规则，请启用此属性。

3.1.3. KIE 模块支持的 KIE 会话属性

KIE 会话存储并执行运行时数据，如果您已在项目的 KIE 模块描述符文件(**kmodule.xml**)中定义 KIE 会话，则直接从 KIE 容器创建。当您在 **kmodule.xml** 文件中定义 KIE bases 和 KIE 会话时，您可以指定某些属性和值来进一步自定义 KIE 会话配置。

kmodule.xml 文件中的 KIE 会话配置示例

```
<kmodule>
```

```
  ...
  <kbase>
    ...
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
    ...
  </kbase>
  ...
</kmodule>
```

以下是项目的 KIE 模块描述符文件(**kmodule.xml**)中支持的 **ksession** 属性和值：

表 3.2. KIE 模块支持的 KIE 会话属性

属性	支持的值	描述
name	任何名称	定义从 **KieContainer** 检索 **KieSession** 的名称。*此属性是必需的。*
type	**有状态**、**无状态** 默认： **stateful**	决定在 KIE 会话调用之间是否保留(**有状态**)还是丢弃(**无状态**)。将会话设置为 **stateful** 可让您使用工作内存进行迭代工作；将会话设置为 **stateless** 通常用于一次性执行的资产。**无状态** 会话存储一个知识状态，每次执行规则时都会更改新的事实、更新或删除。在 **无状态** 会话中执行没有关于之前操作的信息，如规则执行。
default	**true**,**false** 默认：**false**	确定 KIE 会话是否为模块的默认会话，以便可以从 KIE 容器创建，而无需传递任何名称。每个模块只能有一个默认 KIE 会话。
clockType	实时，伪 默认： **realtime**	决定是否由系统时钟分配事件时间戳，还是由应用程序控制的伪时钟。这个时钟对于在时间规则中的单元测试特别有用。
beliefSystem	**simple**,**jtms**,**defeasible** 默认： **simple**	定义 KIE 会话使用的 belief 系统类型。信道系统从知识(facts)中推断出事实。例如，如果根据稍后从决策引擎中删除的另一个事实插入新事实，则系统也可以确定还应删除新插入的事实。

3.2. 在 MAVEN 中打包和部署 RED HAT PROCESS AUTOMATION MANAGER 项目

如果要将 Business Central 之外的 Maven 项目部署到配置的 KIE 服务器，您可以编辑项目 **pom.xml** 文件，将项目打包为 KJAR 文件，并使用 KIE 基础和 KIE 会话配置添加 **kmodule.xml** 文件。

先决条件

- 您有一个包含 Red Hat Process Automation Manager 商业资产的 Maven 项目。

- KIE Server 已安装，并且配置了 **kie-server** 用户访问权限。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。

流程

1. 在 Maven 项目的 **pom.xml** 文件中，将打包类型设置为 **kjar** 并添加 **kie-maven-plugin** 构建组件：

```
<packaging>kjar</packaging>
...
<build>
  <plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>${rhpam.version}</version>
      <extensions>true</extensions>
    </plugin>
  </plugins>
</build>
```

kjar 打包类型激活 **kie-maven-plugin** 组件以验证和预编译工件资源。& **lt;version** > 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（如 7.59.0.Final-redhat-00006）。这些设置需要正确打包 Maven 项目以进行部署。

注意

考虑将 Red Hat Business Automation Manager (BOM)依赖项添加到项目的 **pom.xml** 文件中，而不是为单独的依赖项指定 Red Hat Process Automation Manager < **version** >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确传输依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

有关 Red Hat Business Automation BOM 的更多信息，请参阅 Red Hat Process Automation Manager 和 Maven 库版本之间的映射是什么？

2. 可选：如果您的项目包含 Decision Model 和 Notation (DMN)资产，请在 **pom.xml** 文件中添加以下依赖项来启用 DMN 可执行模型。DMN 可执行模型启用 DMN 决策表逻辑，以便更有效地评估。

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
```

```
  <scope>provided</scope>
  <version>${rhpam.version}</version>
</dependency>
```

3. 在 Maven 项目的 ~/**resources** 目录中，创建一个 **META-INF/kmodule.xml** 元数据文件，其至少包含以下内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

这个 **kmodule.xml** 文件是一个 KIE 模块描述符，是所有 Red Hat Process Automation Manager 项目所需的。您可以使用 KIE 模块来定义一个或多个 KIE 基础，以及每个 KIE 基础的一个或多个 KIE 会话。

有关 **kmodule.xml** 配置的详情请参考 第 3.1 节 “配置 KIE 模块描述符文件”。

4. 在 Maven 项目中的相关资源中，配置 **.java** 类来创建 KIE 容器，以及一个 KIE 会话来加载 KIE 基础：

```
import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public void testApp() {

  // Load the KIE base:
  KieServices ks = KieServices.Factory.get();
  KieContainer kContainer = ks.getKieClasspathContainer();
  KieSession kSession = kContainer.newKieSession();

}
```

在本例中，KIE 容器读取从 **testApp** 项目的类路径构建的文件。**KieServices** API 可让您访问所有 KIE 构建和运行时配置。

您还可以通过将项目 **ReleaseId** 传递给 **KieServices** API 来创建 KIE 容器。 **ReleaseId** 通过项目 **pom.xml** 文件中的 **GroupId**、**ArterfactId** 和 **Version** (GAV) 值生成。

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public void testApp() {

  // Identify the project in the local repository:
  ReleaseId rid = new ReleaseIdImpl("com.sample", "my-app", "1.0.0");

  // Load the KIE base:
  KieServices ks = KieServices.Factory.get();
  KieContainer kContainer = ks.newKieContainer(rid);
```

```
    KieSession kSession = kContainer.newKieSession();

}
```

5. 在命令终端中，导航到 Maven 项目目录，再运行以下命令来构建项目：

```
mvn clean install
```

对于 DMN 可执行模型，运行以下命令：

```
mvn clean install -DgenerateDMNModel=YES
```

如果构建失败，请解决命令行错误消息中描述的任何问题，然后再次尝试验证文件，直到构建成功为止。

注意

如果 Maven 项目中的规则资产没有从可执行规则模型构建，请验证以下依赖项是否在项目的 **pom.xml** 文件中构建，并重新构建项目：

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

默认情况下，红帽流程自动化管理器中的规则资产需要这个依赖项，以便从可执行规则模型构建。此依赖项作为 Red Hat Process Automation Manager 核心打包的一部分提供，但根据您的 Red Hat Process Automation Manager 升级历史记录，您可能需要手动添加这个依赖项来启用可执行规则模型行为。

有关可执行规则模型的详情请参考 第 3.4 节 “可执行规则模型”。

6. 成功在本地构建并测试项目后，将项目部署到远程 Maven 存储库：

```
mvn deploy
```

3.3. 在 JAVA 应用程序中打包和部署红帽流程自动化管理器项目

如果要从您自己的 Java 应用程序内部部署项目到配置的 KIE 服务器，您可以使用 **KieModuleModel** 实例以编程方式创建一个 **kmodule.xml** 文件来定义 KIE 基础和 KIE 会话，然后将项目中的所有资源添加到 KIE 虚拟文件系统 **KieFileSystem** 中。

先决条件

- 您有一个包含红帽流程自动化管理器商业资产的 Java 应用程序。
- KIE Server 已安装，并且配置了 **kie-server** 用户访问权限。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。

流程

1. 可选： 如果您的项目包含 Decision Model 和 Notation (DMN)资产，请将以下内容添加到 Java 项目的相关类路径中，以启用 DMN 可执行文件模型。DMN 可执行模型启用 DMN 决策表逻辑，以便更有效地评估。

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
  <scope>provided</scope>
  <version>${rhpam.version}</version>
</dependency>
```

& **lt;version** > 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（如 7.59.0.Final-redhat-00006）。

注意

考虑将 Red Hat Business Automation Manager (BOM)依赖项添加到项目的 **pom.xml** 文件中，而不是为单独的依赖项指定 Red Hat Process Automation Manager < **version** >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确传输依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

有关 Red Hat Business Automation BOM 的更多信息，请参阅 Red Hat Process Automation Manager 和 Maven 库版本之间的映射是什么？

2. 使用 **KieServices** API 使用所需的 KIE 基础和 KIE 会话创建一个 **KieModuleModel** 实例。**KieServices** API 可让您访问所有 KIE 构建和运行时配置。 **KieModuleModel** 实例为您的项目生成 **kmodule.xml** 文件。
有关 **kmodule.xml** 配置的详情请参考 第 3.1 节 “配置 KIE 模块描述符文件”。

3. 将您的 **KieModuleModel** 实例转换为 XML，并将 XML 添加到 **KieFileSystem**。

以编程方式创建 kmodule.xml 并将其添加到 KieFileSystem

```
import org.kie.api.KieServices;
import org.kie.api.builder.model.KieModuleModel;
import org.kie.api.builder.model.KieBaseModel;
import org.kie.api.builder.model.KieSessionModel;
import org.kie.api.builder.KieFileSystem;

 KieServices kieServices = KieServices.Factory.get();
 KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

 KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel("KBase1")
   .setDefault(true)
```

```
    .setEqualsBehavior(EqualityBehaviorOption.EQUALITY)
    .setEventProcessingMode(EventProcessingOption.STREAM);

  KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel("KSession1")
    .setDefault(true)
    .setType(KieSessionModel.KieSessionType.STATEFUL)
    .setClockType(ClockTypeOption.get("realtime"));

  KieFileSystem kfs = kieServices.newKieFileSystem();
  kfs.writeKModuleXML(kieModuleModel.toXML());
```

4. 将项目中任何剩余的红帽流程自动化管理器资产添加到 **KieFileSystem** 实例中。工件必须采用 Maven 项目文件结构。

```
import org.kie.api.builder.KieFileSystem;

KieFileSystem kfs = ...
kfs.write("src/main/resources/KBase1/ruleSet1.drl", stringContainingAValidDRL)
  .write("src/main/resources/dtable.xls",
    kieServices.getResources().newInputStreamResource(dtableFileStream));
```

在本例中，项目资产作为 **String** 变量和 **Resource** 实例添加。您还可以使用 **KieResources** factory 创建 **Resource** 实例，也由 **KieServices** 实例提供。**KieResources** 类提供了将 **InputStream**, **URL**, 和 **File** 对象，或一个代表您的文件系统路径的 **String** 转换到一个 **KieFileSystem** 可以管理的 **Resource** 实例的方法。

当将项目工件添加到 **KieFileSystem** 时，您还可以为 **Resource** 对象明确分配一个 **ResourceType** 属性：

```
import org.kie.api.builder.KieFileSystem;

KieFileSystem kfs = ...
kfs.write("src/main/resources/myDrl.txt",
  kieServices.getResources().newInputStreamResource(drlStream)
    .setResourceType(ResourceType.DRL));
```

5. 使用带有 **buildAll （）** 方法的 **KieBuilder** 来构建 **KieFileSystem** 的内容，并创建一个 KIE 容器来部署它：

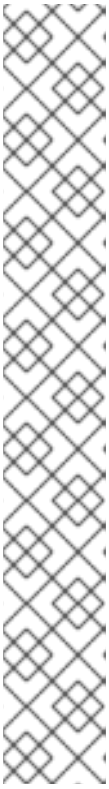
```
import org.kie.api.KieServices;
import org.kie.api.KieServices.Factory;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieBuilder;
import org.kie.api.runtime.KieContainer;

KieServices kieServices = KieServices.Factory.get();
KieFileSystem kfs = ...

KieBuilder kieBuilder = ks.newKieBuilder( kfs );
kieBuilder.buildAll()
assertEquals(0, kieBuilder.getResults().getMessages(Message.Level.ERROR).size());

KieContainer kieContainer = kieServices
  .newKieContainer(kieServices.getRepository().getDefaultReleaseId());
```

构建 **ERROR** 表示项目编译失败，没有生成 **KieModule**，且不会添加到 **KieRepository** 单例。**WARNING** 或 **INFO** 结果表示项目编译成功，以及构建过程的相关信息。



注意

要从可执行规则模型在 Java 应用程序项目中构建规则资产，请验证以下依赖项是否在项目的 **pom.xml** 文件中：

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

红帽流程自动化管理器中的规则资产需要这个依赖项，以便从可执行规则模型构建。此依赖项作为 Red Hat Process Automation Manager 核心打包的一部分提供，但根据您的 Red Hat Process Automation Manager 升级历史记录，您可能需要手动添加这个依赖项来启用可执行规则模型行为。

验证依赖项后，使用以下修改后的 **buildAll （）** 选项启用可执行模型：

```
kieBuilder.buildAll(ExecutableModelProject.class)
```

有关可执行规则模型的详情请参考 第 3.4 节 “可执行规则模型”。

3.4. 可执行规则模型

Red Hat Process Automation Manager 中的规则资产默认使用标准 **kie-maven-plugin** 插件从可执行规则模型构建。可执行规则模型是嵌入式模型，它为构建时执行的规则集提供基于 Java 的表示。在以前的 Red Hat Process Automation Manager 版本中，可执行模型是更有效的标准资产打包的替代方案，并允许 KIE 容器和 KIE 基础更快地创建，特别是在您有大量 DRL (Drools 规则语言)文件和其他 Red Hat Process Automation Manager 资产时。

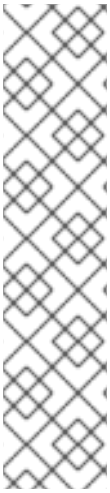
如果您没有使用 **kie-maven-plugin** 插件，或者项目中缺少所需的 **drools-model-compiler** 依赖项，则在没有可执行模型的情况下构建规则资产。因此，要在构建期间生成可执行模型，请确保在项目 **pom.xml** 文件中添加 **kie-maven-plugin** 插件和 **drools-model-compiler** 依赖项。

可执行规则模型为您的项目提供以下特定优点：

- **编译时间：** 使用打包的 Red Hat Process Automation Manager 项目(KJAR)包含 DRL 文件和其他 Red Hat Process Automation Manager 工件列表，该工件将规则基础与一些预生成的类一起定义约束和结果。当 KJAR 从 Maven 存储库下载并安装在 KIE 容器中时，必须解析和编译这些 DRL 文件。这个过程可能会很慢，特别是对于大型规则集。使用可执行模型，您可以在项目 KJAR 中打包实施项目规则基础的可执行模型的 Java 类，并以更快的方式重新创建 KIE 容器及其 KIE 基础。在 Maven 项目中，您可以使用 **kie-maven-plugin** 插件在编译过程中从 DRL 文件自动生成可执行模型源。

- **运行时：** 在可执行模型中，所有限制都定义为 Java lambda 表达式。同样的 lambda 表达式也用于约束评估，因此您不再需要将 **mvel** 表达式用于解释评估，或使用即时(JIT)进程将基于 **mvel**的约束转换为字节码。这会更快、更有效的运行时。

- **开发时间：** 可执行模型可让您使用决策引擎的新功能来开发和试验，而无需直接以 DRL 格式编码元素或修改 DRL 解析器来支持它们。



注意

对于可执行规则模型中的查询定义，您只能使用最多 10 个参数。

对于可执行规则模型中的变量，您只能使用 24 个绑定变量（包括内置的 **drools** 变量）。例如，以下规则结果使用了超过 24 个绑定变量，并创建一个编译错误：

```
...
then
  $input.setNo25Count(functions.sumOf(new Object[]{$no1Count_1, $no2Count_1,
$no3Count_1, ..., $no25Count_1}).intValue());
  $input.getFirings().add("fired");
  update($input);
```

3.4.1. 在 Red Hat Process Automation Manager 项目中修改或禁用可执行规则模型

Red Hat Process Automation Manager 中的规则资产默认使用标准 **kie-maven-plugin** 插件从可执行规则模型构建。可执行模型是之前 Red Hat Process Automation Manager 版本中的标准资产打包的更有效的替代方法。但是，如果需要，您可以修改或禁用可执行规则模型，将 Red Hat Process Automation Manager 项目构建为基于 DRL 的 KJAR，而不是默认的基于模型的 KJAR。

流程

按照通常的方式构建您的 Red Hat Process Automation Manager 项目，但根据项目类型提供备用构建选项：

- 对于 Maven 项目，在命令终端中导航到 Maven 项目目录，再运行以下命令：

  ```
  mvn clean install -DgenerateModel=<VALUE>
  ```

  将 **<VALUE** > 替换为三个值之一：

  - **YES_WITHDRL** ：（默认）生成与原始项目中的 DRL 文件对应的可执行文件模型，并将 DRL 文件添加到生成的 KJAR 中。（KIE 基础是从可执行模型构建的。
  - **YES** ：生成与原始项目中的 DRL 文件对应的可执行模型，并从生成的 KJAR 中排除 DRL 文件。
  - **NO** ：不要生成可执行模型。

  禁用默认可执行模型行为的 build 命令示例：

  ```
  mvn clean install -DgenerateModel=NO
  ```

- 对于以编程方式配置的 Java 应用程序，默认禁用可执行模型。在 KIE 虚拟文件系统 **KieFileSystem** 中添加规则资产，并使用带有以下 **buildAll （）** 方法之一的 **KieBuilder** ：

  - **buildAll （）** （默认）或 **buildAll (DrlProject.class)**: 不生成可执行模型。
  - **BuildAll (ExecutableModelProject.class)** ：生成与原始项目中的 DRL 文件对应的可执行文件模型。

  启用可执行模型行为的代码示例：

  ```
  import org.kie.api.KieServices;
  ```

```
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieBuilder;

 KieServices ks = KieServices.Factory.get();
 KieFileSystem kfs = ks.newKieFileSystem()
 kfs.write("src/main/resources/KBase1/ruleSet1.drl", stringContainingAValidDRL)
 .write("src/main/resources/dtable.xls",
   kieServices.getResources().newInputStreamResource(dtableFileStream));

 KieBuilder kieBuilder = ks.newKieBuilder( kfs );
 // Enable executable model
 kieBuilder.buildAll(ExecutableModelProject.class)
 assertEquals(0, kieBuilder.getResults().getMessages(Message.Level.ERROR).size());
```

3.5. 使用 KIE 扫描程序来监控和更新 KIE 容器

Red Hat Process Automation Manager 中的 KIE 扫描程序会监控您的 Maven 存储库以获取 Red Hat Process Automation Manager 项目的新 **SNAPSHOT** 版本，然后将项目的最新版本部署到指定的 KIE 容器。您可以在开发环境中使用 KIE 扫描程序，在新版本可用时更有效地维护您的 Red Hat Process Automation Manager 项目部署。

重要

对于生产环境，请不要将 KIE 扫描程序与 **SNAPSHOT** 项目版本搭配使用，以避免意外或意外的项目更新。KIE 扫描程序适用于使用 **SNAPSHOT** 项目版本的开发环境。

避免将 KIE 扫描程序与业务流程一起使用。将 KIE 扫描程序与进程搭配使用可能会导致无法预见的更新，然后在更改与正在运行的进程实例不兼容时导致长时间运行的进程出现错误。

先决条件

- **kie-ci.jar** 文件位于 Red Hat Process Automation Manager 项目的类路径上。

流程

1. 在项目中的相关 **.java** 类中，注册并启动 KIE 扫描程序，如下例所示：

   为 KIE 容器注册并启动 KIE 扫描程序

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.builder.KieScanner;

...

KieServices kieServices = KieServices.Factory.get();
ReleaseId releaseId = kieServices
  .newReleaseId("com.sample", "my-app", "1.0-SNAPSHOT");
KieContainer kContainer = kieServices.newKieContainer(releaseId);
KieScanner kScanner = kieServices.newKieScanner(kContainer);
```

```
// Start KIE scanner for polling the Maven repository every 10 seconds (10000 ms)
kScanner.start(10000L);
```

在本例中，KIE 扫描程序被配置为以固定间隔运行。最小 KIE 扫描程序轮询的间隔为 1 毫秒 (ms)，最大轮询间隔是数据类型 **long** 的最大值。轮询间隔 0 或更少会导致 **java.lang.IllegalArgumentException: pollingInterval 必须是正** 错误。您还可以通过调用 **scanNow （）** 方法，将 KIE 扫描程序配置为按需运行。

示例中的项目组 ID、工件 ID 和版本(GAV)设置定义为 **com.sample:my-app:1.0-SNAPSHOT**。项目版本必须包含 **-SNAPSHOT** 后缀，使 KIE 扫描程序能够检索指定工件版本的最新构建。如果您更改了快照项目版本号，如增加到 **1.0.1-SNAPSHOT**，那么您还必须在 KIE 扫描程序配置中的 GAV 定义中更新版本。KIE 扫描程序不会检索具有静态版本的项目的更新，如 **com.sample:my-app:1.0**。

2. 在 Maven 存储库的 **settings.xml** 文件中，将 **updatePolicy** 配置设置为 **always** 以使 KIE 扫描程序正常工作：

```
<profile>
  <id>guvnor-m2-repo</id>
  <repositories>
    <repository>
      <id>guvnor-m2-repo</id>
      <name>BA Repository</name>
      <url>http://localhost:8080/business-central/maven2/</url>
      <layout>default</layout>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
      </snapshots>
    </repository>
  </repositories>
</profile>
```

KIE 扫描程序开始轮询后，如果 KIE 扫描程序检测到指定 KIE 容器中 **SNAPSHOT** 项目的更新版本，KIE 扫描程序会自动下载新项目的逐步构建。此时，从 KIE 容器创建的所有新 **KieBase** 和 **KieSession** 对象都使用新项目版本。

有关使用 KIE Server API 启动或停止 KIE 扫描程序的详情，请参考使用 KIE API *与 Red Hat Process Automation Manager 进行交互*。

3.6. 在 KIE 服务器中启动服务

如果您已经从 Business Central 之外的 Maven 或 Java 项目部署了红帽流程自动化管理器资产，您可以使用 KIE Server REST API 调用来启动 KIE 容器（部署单元）及其中的服务。您可以使用 KIE Server REST API 来启动服务，无论您的部署类型（包括从 Business Central 的部署），但从 Business Central 部署的项目会自动启动，也可以在 Business Central 界面中启动。

先决条件

- KIE Server 已安装，并且配置了 **kie-server** 用户访问权限。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。

流程

在命令终端中，运行以下 API 请求将服务加载到 KIE Server 中的 KIE 容器中，并启动它：

```
$ curl --user "<username>:<password>" -H "Content-Type: application/json" -X PUT -d '{"container-id" : "<containerID>","release-id" : {"group-id" : "<groupID>","artifact-id" : "<artifactID>","version" : "<version>"}}' http://<serverhost>:<serverport>/kie-server/services/rest/server/containers/<containerID>
```

替换以下值：

- **<username >,**<password> : 具有 **kie-server** 角色的用户的用户名和密码。
- **<containerID> ：** KIE 容器（部署单元）的标识符。您可以使用任何随机标识符，但在命令(URL 和数据)中的位置中都必须相同。
- **<groupID>**, **<artifactID>**, **<version>:** 项目的 GAV 值。
- **<server** host> : KIE Server 的主机名，如果您在与 KIE Server 相同的主机上运行命令，则为 **localhost**。
- **<serverport> ：** KIE 服务器的端口号。

例如：

```
curl --user "rhpamAdmin:password@1" -H "Content-Type: application/json" -X PUT -d '{"container-id" : "kie1","release-id" : {"group-id" : "org.kie.server.testing","artifact-id" : "container-crud-tests1","version" : "2.1.0.GA"}}' http://localhost:39043/kie-server/services/rest/server/containers/kie1
```

3.7. 在 KIE 服务器中停止和删除服务

如果您已从 Business Central 之外的 Maven 或 Java 项目启动 Red Hat Process Automation Manager 服务，您可以使用 KIE Server REST API 调用来停止并删除包含该服务的 KIE 容器（部署单元）。您可以使用 KIE Server REST API 来停止服务，无论您的部署类型（包括来自 Business Central 的部署），但也可在 Business Central 界面中停止服务。

先决条件

- KIE Server 已安装，并且配置了 **kie-server** 用户访问权限。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。

流程

在命令终端中，运行以下 API 请求来停止和删除 KIE 容器，并带有 KIE Server 上的服务：

```
$ curl --user "<username>:<password>" -X DELETE http://<serverhost>:<serverport>/kie-server/services/rest/server/containers/<containerID>
```

替换以下值：

- **<username >,**<password> : 具有 **kie-server** 角色的用户的用户名和密码。

- **<containerID> ：** KIE 容器（部署单元）的标识符。您可以使用任何随机标识符，但在命令(URL 和数据)中的位置中都必须相同。
- **<server** host> : KIE Server 的主机名，如果您在与 KIE Server 相同的主机上运行命令，则为 **localhost**。
- **<serverport> ：** KIE 服务器的端口号。

例如：

```
curl --user "rhpamAdmin:password@1" -X DELETE http://localhost:39043/kie-
server/services/rest/server/containers/kie1
```

第 4 章 其他资源

- 使用 *DRL 规则设计决策服务中的*"执行规则"
- *使用 KIE API 与 Red Hat Process Automation Manager 交互*
- *使用 Operator 在 Red Hat OpenShift Container Platform 4 上部署 Red Hat Process Automation Manager 环境*
- *使用模板在 Red Hat OpenShift Container Platform 3 上部署 Red Hat Process Automation Manager 环境*

部分 II. 在 BUSINESS CENTRAL 中管理项目

作为流程管理员，您可以使用红帽流程自动化管理器中的 Business Central 管理一个或多个分支上的新、示例和导入项目。

先决条件

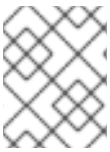
- 安装了 Red Hat JBoss Enterprise Application Platform 7.4。详情请查看 *Red Hat JBoss Enterprise Application Platform 7.4 安装指南*。
- Red Hat Process Automation Manager 使用 KIE Server 安装和配置。如需更多信息，请参阅 *在 Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Process Automation Manager*。
- Red Hat Process Automation Manager 正在运行，您可以使用 **开发人员** 角色登录到 Business Central。如需更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装* 。

第 5 章 RED HAT PROCESS AUTOMATION MANAGER 项目

Red Hat Process Automation Manager 项目包含您在红帽流程自动化管理器中开发的业务资产，并分配给一个空格（如 **MySpace**中的 **MyProject** ）。项目还包含配置文件，如 Maven 项目对象模型文件(**pom.xml**)，其中包含项目中关于构建、环境和 KIE 模块描述符文件(**kmodule.xml**)，其中包含项目中资产的 KIE Base 和 KIE Session 配置。

第 6 章 将业务流程迁移到新的流程设计程序

Business Central 中的旧流程设计程序在红帽流程自动化管理器 7.12.0 中已弃用。它将在以后的 Red Hat Process Automation Manager 发行版本中删除。传统的进程设计程序将不会收到任何新的增强或功能。如果要使用新的流程设计器，开始将流程迁移到新的设计人员。在新的进程设计过程中创建所有新进程。



注意

流程引擎将继续支持使用传统设计器生成的业务流程执行和部署到 KIE 服务器。如果您的旧流程可正常运行，而且您不打算更改，则此时不必再迁移到新的设计人员。

您只能在新的设计程序中迁移包含受支持的业务流程节点的业务流程。在以后的 Red Hat Process Automation Manager 版本中会添加更多节点。

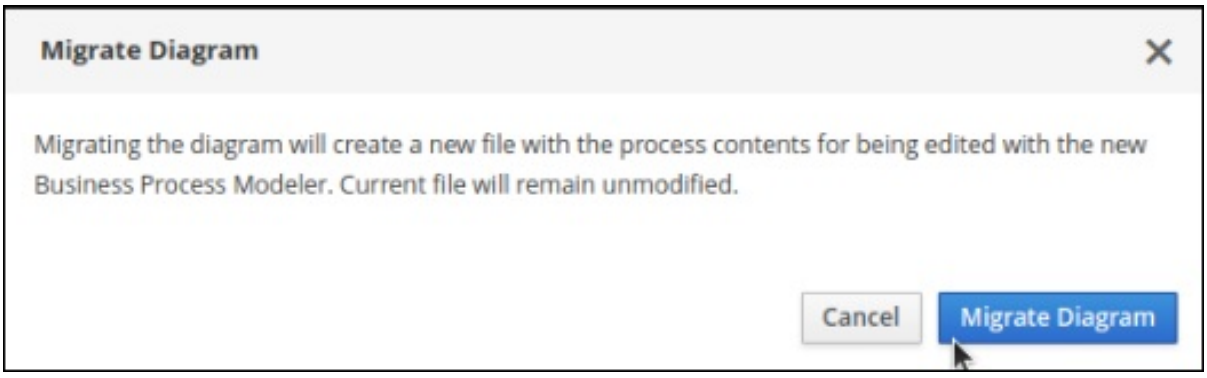
先决条件

- 您有一个现有项目，其中包含使用旧流程设计程序创建的业务流程资产。

流程

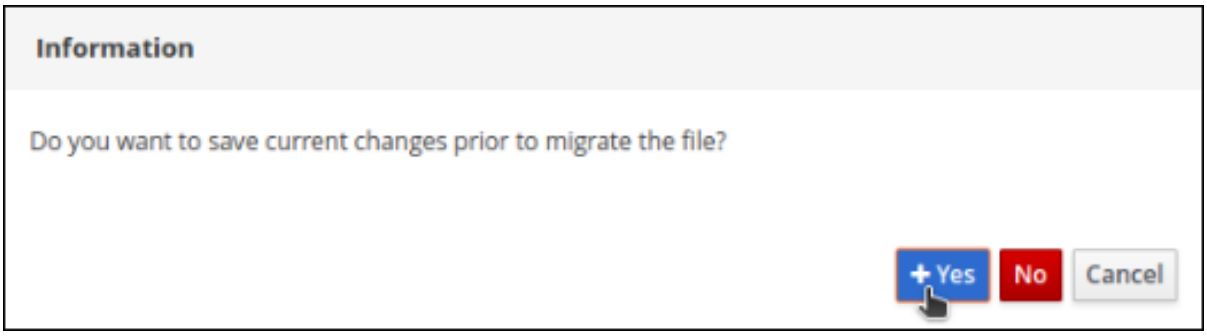
1. 在 Business Central 中，点 **Menu** → **Design** → **Projects**。
2. 点击您要迁移的项目，如 **Mortgage_Process**。
3. 点 **Ok** 以打开项目的资产列表。
4. 单击项目的 **Business Process** asset，在传统流程设计程序中打开它。
5. 点 **Migrate** → **Migrate Diagram**。

   图 6.1. 迁移确认信息

6. 选择 **Yes** 或 **No** 以确认您是否进行了更改。只有在您更改了旧业务流程时，此选项才可用。

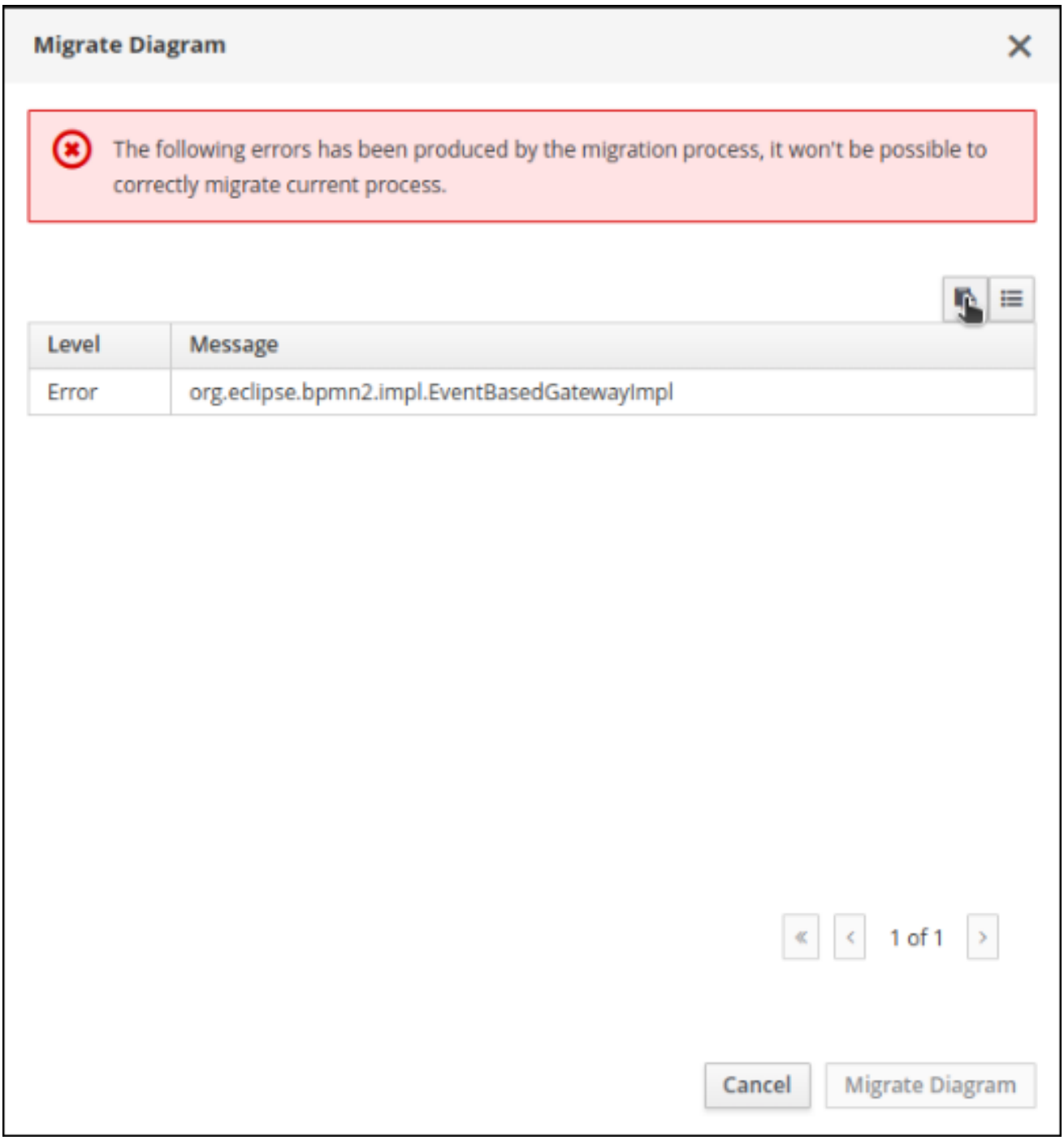
图 6.2. 保存图更改确认



如果迁移成功，业务流程会在新的流程设计程序中打开，业务流程的扩展从 *.bpmn2 改为 *.bpmn。

如果因为不支持的节点类型导致迁移失败，则 Business Central 会显示以下出错信息：

图 6.3. 迁移失败信息



第 7 章 在 BUSINESS CENTRAL 中修改现有的项目

Business Central 包括多个示例项目，可用于熟悉产品及其功能。示例项目已设计并已创建，用于演示各种业务场景。您可以修改示例项目，以满足您的特定业务需求。例如，Red Hat Process Automation Manager 7.12 包含 **Mortgage_Process** 示例项目，它由预定义的数据对象、指导决策表、指导规则、表单和业务流程组成。您可以编辑示例来优化您的迁移过程。

如果没有现有的 Business Central 项目样本与您的要求一致，您可以创建新项目或从 Git 存储库导入。更多信息请参阅 *第 9 章 从 Git 存储库导入项目*。您可以从 Git 导入任何其他项目。例如，在另一个 Business Central 实例中开发的项目。

第 8 章 创建 MORTGAGE-PROCESS 项目

项目是数据对象、业务流程、指导规则、决策表和表单等资产的容器。您要创建的项目与 Business Central 中的现有 **Mortgage_Process** 示例项目类似。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**。
   Red Hat Process Automation Manager 提供一个名为 **MySpace** 的默认空间，如下图所示。您可以使用默认空间来创建和测试示例项目。

   图 8.1. 默认空间

2. 单击 **Add Project**。
3. 在 **Name** 字段中输入 **mortgage-process**。
4. 点击 **Configure Advanced Options** 并使用以下值修改 GAV 字段：
   - 组 ID ：**com.myspace**
   - 工件 ID:**mortgage-process**
   - Version:**1.0.0**
5. 点击 **Add**。

项目的 **Assets** 视图将打开。

8.1. 修改 MORTGAGE_PROCESS 示例项目

Mortgage_Process 示例项目包括预定义的数据对象、指导决策表、指导规则、表单和业务流程。使用示例项目可让您快速使用 Red Hat Process Automation Manager。在实际业务场景中，您可以通过提供特定于业务需求的数据来创建所有资产。

导航到 **Mortgage_Process** 示例项目，以查看预定义的资产。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**。
2. 在屏幕右上角，单击 **Add Project** 旁边的箭头，再选择 **Try Samples**。
3. 选择 **Mortgage_Process**，再单击 **Ok**。项目的 **Assets** 视图将打开。
4. 点击您要修改的资产。可以编辑所有资产来满足您的项目要求。

8.2. 使用 ARCHETYPES 创建项目

archetypes 是在 Apache Maven 存储库中安装的项目，其中包含特定的模板结构。您还可以使用 archetypes 生成项目模板的参数化版本。当您使用 archetype 创建项目时，会将其添加到连接到 Red Hat Process Automation Manager 安装的 Git 存储库中。

先决条件

- 您已创建了 archetype，并将其添加到 Business Central **Settings** 中的 **Archetypes** 页面中。有关创建 archetypes 的详情，请参考创建存档类型指南。
- 您已在 Business Central 的空白中设置默认架构类型。

有关 archetypes 管理的更多信息，*请参阅配置 Business Central 设置和属性*。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**。
2. 从 archetype 模板选择要添加新项目的空格。
3. 单击 **Add Project**。
4. 在 **Name** 和 **Description** 字段中输入项目名称和描述。
5. 单击 **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 from-down 选项。选择了已在空间中设置的默认 archetype。
8. 点击 **Add**。

项目的 Assets 视图会根据所选的 archetype 模板打开。

第 9 章 从 GIT 存储库导入项目

Git 是一个分布式版本控制系统。它将修订版本作为提交对象实施。当您将更改保存到存储库时，会在 Git 存储库中创建新的提交对象。

Business Central 使用 Git 来存储项目数据，包括规则和流程等资产。当您在 Business Central 中创建项目时，它会被添加到连接到 Business Central 的 Git 存储库中。如果您在 Git 存储库中拥有项目，您可以通过 Business Central 空格导入项目的 master 分支，或者将 master 分支和其他特定分支导入到 Business Central Git 存储库中。

先决条件

- Red Hat Process Automation Manager 项目存在于外部 Git 存储库中。
- 您有对该外部 Git 存储库的读取访问权限所需的凭证。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**。
2. 选择或创建您要导入项目的空间。默认空间为 **MySpace**。
3. 在屏幕右上角，单击 **Add Project** 旁边的箭头，然后选择 **Import Project**。
4. 在 **Import Project** 窗口中，输入 Git 存储库的 URL 和凭证，该存储库包含您要导入的项目，然后单击 **Import**。此时会显示 **Import Projects** 页面。
5. 可选： 要导入 master 和特定分支，请执行以下任务：
   a. 在 **Import Projects** 页面中，点分支  图标。
   b. 在 **要导入的分支** 窗口中，从列表中选择 branch。

   注意

   您必须至少选择 master 分支。

   c. 点 **确定**。
6. 在 **Import Projects** 页面上，确保项目被突出显示，然后单击 **Ok**。

第 10 章 重新查看项目版本

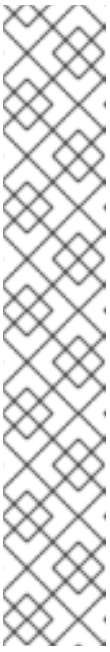
在构建和部署项目的新实例前，您可以修复 Red Hat Process Automation Manager 中的项目的版本号。如果新项目有新版本，您需要恢复新版本，则项目会保留旧版本。

先决条件

- KIE 服务器已部署并连接到 Business Central。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**。
2. 点您要部署的项目，如 **Mortgage_Process**。
3. 单击 **Deploy**。
   - 如果没有带有项目名称的容器，则会自动创建具有默认值的容器。
   - 如果已经部署了项目的旧版本，请转至项目设置并更改项目版本。完成后，保存更改并点 **Deploy**。这将部署同一项目的新版本，并有最新的更改，以及旧版本。



注意

您还可以选择 **Build & Install**选项来构建项目，并将 KJAR 文件发布到配置的 Maven 存储库，而无需部署到 KIE 服务器。在开发环境中，您可以点击 **Deploy** 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），或者点击 **Redeploy** 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，**Redeploy** 选项被禁用，您可以点 **Deploy** only 将构建的 KJAR 文件部署到 KIE 服务器上的新部署单元(KIE 容器)。

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**。要在 Business Central 中为对应项目配置部署行为，请转至 Project **Settings → General Settings → Version** 并切换 **Development Mode** 选项。默认情况下，KIE 服务器和 Business Central 中的所有新项目都处于开发模式中。您不能部署打开 **Development 模式** 的项目，或使用手动将 **SNAPSHOT** 版本后缀添加到生产模式的 KIE 服务器中。

4. 要查看项目部署详情，请单击屏幕顶部的部署横幅中的 **View deployment details**，或者在 **Deploy** 下拉菜单中。这个选项将您定向到 **Menu → Deploy → Execution Servers** 页面。
5. 要验证进程定义，请点击 **Menu → Manage → Process Definitions**，然后点击  。
6. 点击 **Actions** 列中的  并选择 **Start** 来启动进程的新实例。

第 11 章 配置项目设置

从 Red Hat Process Automation Manager 7.12 开始，Business Central 包含新进程设计程序中的其他项目设置类别。

先决条件

- 您已创建了 Business Central 项目。

流程

1. 要访问项目 **设置** 选项卡，请在 Business Central 中转至 **Menu → Design → Projects**。
2. 单击项目名称。
3. 点 **Settings** 查看或修改以下项目设置：
   - **常规设置** - 允许用户设置项目的**名称**、**描述**、组 **ID**、**工件 ID**、**版本** (GAV)和 **开发模式** 属性。它还包括以下选项：
     - **URL** - 使用 将项目克隆项目的只读 URL 指定为 git 存储库。
     - **禁用 GAV 冲突检查** - 确定是否启用或禁用 GAV 冲突检查。禁用此功能可让项目具有相同的 GAV 值。
     - **Allows child GAV 版本** - 允许子项目的 GAV 版本。
   - **依赖项** - 用来手动添加依赖项，方法是输入 **组 ID**、**工件 ID 和版本**，或者从 Business Central 中的存储库项目添加。对于每个依赖项，为 **Package white list**选项设置 **All** 或 **None**。
   - **KIE Bases** - 之前称为" **知识库"的新名称**。您必须指定一个 KIE 基础作为默认值。提供以下详情来添加 Kie 基础：
     - **名称**
     - **包括的 KIE 基础**
     - **软件包**
     - **equal Behavior** - **Identity** 或 **Equality**
     - **事件处理模型** - **流** 或**云**
     - **KIE 会话**
   - **外部** 数据对象 - 数据对象没有在项目或规则作者需要的项目依赖项中显式定义。外部数据对象通常由 Java 运行时提供，如 **java.util.List**。
   - **验证** - 在创建新项目或模块时，用来检查项目 GAV 的唯一 Maven 存储库，或者在安装或部署项目到 Maven 存储库时。
   - **服务任务** - 可向项目添加以下服务任务：
     - **BusinessRuleTask** - 执行业务规则任务
     - **决策任务** - 执行 DMN 决策任务

- **电子邮件** - 发送电子邮件
- **JMSSendTask** - Send JMS Message
- **rest** - 执行 Rest 调用
- **ServiceTask** - 执行服务任务
- **WebService** - 执行 Web 服务调用

- **Deployment** - 部署被分为以下类别：
  - **常规设置** - 运行时**策略**、**持久性单元名称**、**持久性模式**、审计**持久性** 单元名称以及 审计**模式**
  - **Marshalling 策略**
  - **全局**
  - **事件监听程序**
  - **所需的角色**
  - **可远程类**
  - **任务事件监听程序**
  - **配置**
  - **环境条目**
  - **工作项目处理程序**
- **Persistence** - Persistence 被分为以下类别：
  - **持久性单元**
  - **持久性供应商**
  - **数据源**
  - **properties** - 用于设置以下属性的值，以及创建新属性：
    - **hibernate.dialect**
    - **hibernate.max_fetch_depth**
    - **hibernate.hbm2ddl.auto**
    - **hibernate.show_sql**
    - **hibernate.id.new_generator_mappings**
    - **hibernate.transaction.jta.platform**
  - **Project Persistable Data Objects**
- **分支管理** - 根据分支名称和分配的用户角色提供分支角色访问权限。

4. 点击 **Save**。

第 12 章 BUSINESS CENTRAL 中的多个分支

Business Central 中的多个分支支持提供基于现有分支（包括其所有资产）创建新分支的功能。所有新的、导入和示例项目在默认的 **master** 分支中打开。您可以根据需要创建任意数量的分支，并可互换处理多个分支，而不影响 **master** 分支上的原始项目。

Red Hat Process Automation Manager 7.12 包括对持久分支的支持，这意味着 Business Central 会记住使用的最后一个分支，并在重新登录时在该分支中打开。

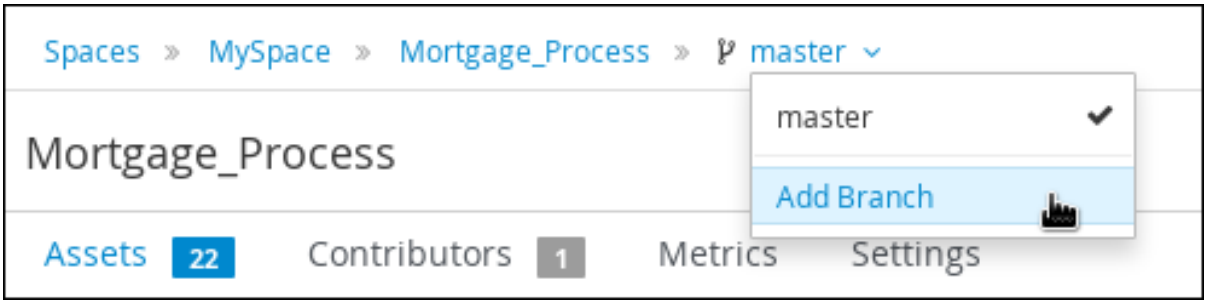
12.1. 创建分支

您可以在 Business Central 中创建新的分支，并将其命名为.最初，您将仅具有默认的 **master** 分支。当您为项目创建新分支时，您要生成所选分支的副本。您可以在新分支上更改项目，而不影响原始 **master** 分支版本。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**。
2. 单击项目以创建新分支，如 **Mortgage_Process** 示例项目。
3. 点 **master → Add Branch**。

图 12.1. 创建新分支菜单



4. 在 **Name** 字段中输入 **testBranch1**，然后从 **Add Branch** 窗口中选择 **master**。其中 **testBranch1** 是您要命名新分支的任何名称。
5. 从 **Add Branch** 窗口中选择将成为新分支的基础的分支。这可以是任何现有分支。
6. 点击 **Add**。

图 12.2. 添加新分支窗口

添加新分支后，您将重定向到它，它将包含在 master 分支中项目中的所有资产。

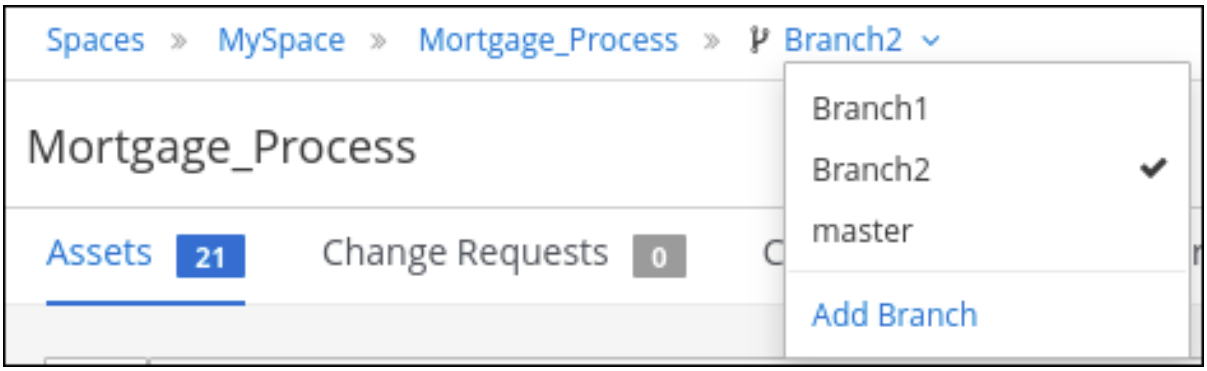
12.2. 选择分支

您可以在分支间切换以对项目资产进行修改并测试修订的功能。

流程

1. 单击当前的分支名称，再从下拉列表中选择所需的项目分支。

图 12.3. 选择分支菜单



选择分支后，您会被重定向到包含项目以及您定义的所有资产的分支。

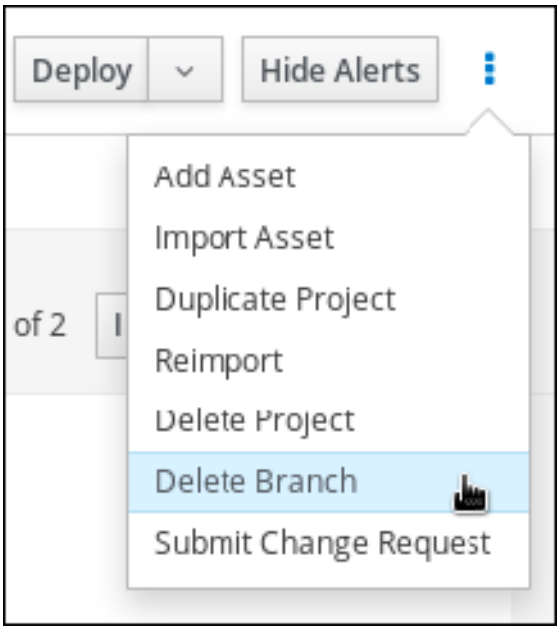
12.3. 删除分支

您可以删除除 master 分支外的任何分支。Business Central 不允许删除 master 分支以避免破坏您的环境。您必须位于 master 以外的任何分支中，才能执行以下步骤。

流程

1. 点击屏幕右上角的  并选择 **Delete Branch**。

图 12.4. 删除分支



2. 在 **Delete Branch** 窗口中，输入您要删除的分支的名称。
3. 点 **Delete Branch**。该分支已被删除，项目分支切换到 master 分支。

12.4. 构建和部署项目

开发项目后，您可以从 Business Central 的指定分支构建项目，并将它部署到配置的 KIE 服务器。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。
2. 在右上角，单击 **Deploy** 以构建项目并将其部署到 KIE Server。

注意

您还可以选择 **Build & Install** 选项来构建项目，并将 KJAR 文件发布到配置的 Maven 存储库，而无需部署到 KIE 服务器。在开发环境中，您可以点击 **Deploy** 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），或者点击 **Redeploy** 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，**Redeploy** 选项被禁用，您可以点 **Deploy** only 将构建的 KJAR 文件部署到 KIE 服务器上的新部署单元(KIE 容器)。

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**。要在 Business Central 中为对应项目配置部署行为，请转至 Project **Settings → General Settings → Version** 并切换 **Development Mode** 选项。默认情况下，KIE 服务器和 Business Central 中的所有新项目都处于开发模式中。您不能部署打开 **Development 模式** 的项目，或使用手动将 **SNAPSHOT** 版本后缀添加到生产模式的 KIE 服务器中。

如果构建失败，请解决屏幕底部的 **Alerts** 面板中描述的任何问题。

要查看项目部署详情，请单击屏幕顶部的部署横幅中的 **View deployment details**，或者在 **Deploy** 下拉菜单中。这个选项将您定向到 **Menu → Deploy → Execution Servers** 页面。

如需有关项目部署选项的更多信息，*请参阅打包和部署红帽流程自动化管理器项目*。

第 13 章 更改 BUSINESS CENTRAL 中的请求

如果您在 Business Central 项目中有多个分支，并且您在要合并到另一个分支的分支中进行更改，您可以创建更改请求。具有查看目标分支（通常为 **master** 分支）的任何用户都可以看到更改请求。

13.1. 创建更改请求

在项目中更改后，您可以在 Business Central 项目中创建更改请求，例如在向资产中添加或删除属性后。

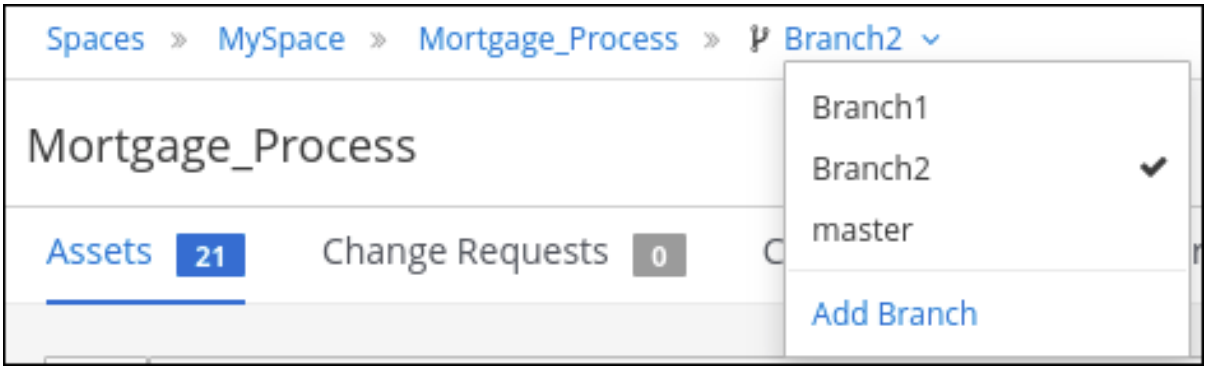
先决条件

- 您有多个 Business Central 项目的分支。
- 您在您要合并到另一个分支的一个分支中进行更改。

流程

1. 在 Business Central 中，进入 **Menu → Design → Projects**，再选择包含您要合并的更改的空间和项目。
2. 在项目页面中，选择包含更改的分支。

   图 13.1. 选择分支菜单

3. 执行以下任务之一以提交更改请求：

   - 点击屏幕右上角的  并选择 **Submit Change Request**。
   - 点 **Change Requests** 选项卡，然后点 **Submit Change Request**。
     此时会出现 **Submit Change Request** 窗口。

4. 输入摘要和描述，选择目标分支，然后单击 **Submit**。target 分支是合并更改的分支。点 **Submit** 后，将显示更改请求窗口。

13.2. 使用更改请求

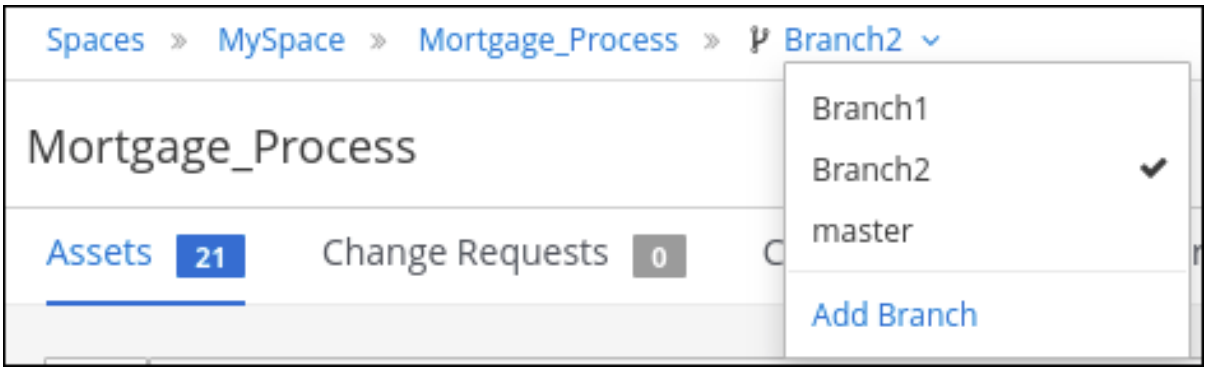
您可以查看您有权访问的任何分支的更改请求。您必须具有管理员权限才能接受更改请求。

先决条件

- 您有多个 Business Central 项目的分支。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，然后选择一个空格和项目。
2. 在项目页面中，验证您是否位于正确的分支中。



3. 点 **Change Requests** 选项卡。此时会出现待处理更改请求列表。
4. 要过滤更改请求，请选择 **Search** 框左面的 **Open**, **Closed**, 或 **All**
5. 要搜索特定的更改请求，请在 **Search** 框中输入 ID 或文本，然后点放大镜图标。
6. 要查看更改请求详情，请单击摘要链接。更改请求窗口有两个标签页：
   a. 参阅 **Overview** 选项卡来获取有关更改请求的一般信息。
   b. 单击 **Changed Files** 选项卡，再展开一个文件来查看所提议的更改。
7. 点击右上角的按钮。
   - 单击 **Squash 和 Merge**，将所有提交压缩到一个提交中，并将提交合并到目标分支。
   - 单击 **Merge**，将更改合并到目标分支中。
   - 单击 **Reject** 以拒绝更改，并使目标分支保持不变。
   - 单击 **Close** 以关闭更改请求，而不拒绝或接受请求。请注意，只有创建提交更改请求的用户才可以关闭它。
   - 单击 **Cancel** 以返回到项目窗口，而不进行任何更改。

部分 III. 在 BUSINESS CENTRAL 中管理资产

作为流程管理员，您可以在红帽流程自动化管理器中使用 Business Central 来管理资产，如规则、业务流程和决策表。

先决条件

- 安装了 Red Hat JBoss Enterprise Application Platform 7.4。详情请查看 *Red Hat JBoss Enterprise Application Platform 7.4 安装指南*。
- Red Hat Process Automation Manager 使用 KIE Server 安装和配置。如需更多信息，请参阅在 *Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Process Automation Manager* 。
- Red Hat Process Automation Manager 正在运行，您可以使用 **开发人员** 角色登录到 Business Central。如需更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装* 。

第 14 章 资产概述

业务规则、流程定义文件以及 Business Central 中创建的其他资产和资源存储在 KIE 服务器访问的 Artifact 存储库(Knowledge Store)中。

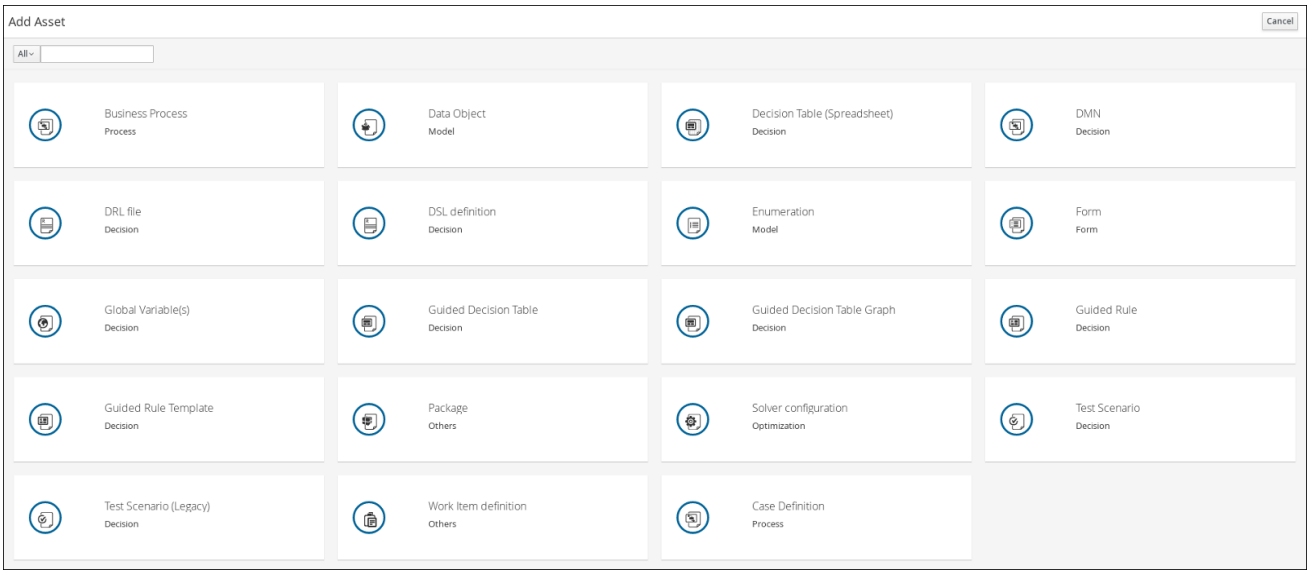
工件存储库是您业务知识的集中存储库。它连接多个 GIT 存储库，以便您可以从一个环境中访问它们，同时在不同位置存储不同类型的知识和工件。GITOPS 是一个分布式版本控制系统，它将修订版本作为提交对象实施。每次您将更改保存到存储库时，这将在 GIT 存储库中创建新提交对象。类似地，用户也可以复制现有的存储库。此复制过程通常称为克隆，生成的存储库可以称为克隆。每个克隆都包含文件集合的完整历史记录，而克隆的存储库具有与原始存储库相同的内容。

Business Central 提供了一个 Web 前端，使您能够查看和更新存储的内容。要访问工件存储库资产，请转至 Business Central 中的 **Menu → Design → Projects**，然后点击项目名称。

第 15 章 资产类型

在 Business Central 存储库中可进行版本控制的任何内容都是资产。项目可以包含规则、软件包、业务流程、决策表、事实模型、域特定语言(DSL)或特定于项目要求的任何其他资产。

下图显示了 Red Hat Process Automation Manager 7.12 中的可用资产。



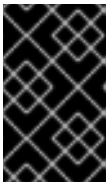
注意

问题单管理(Preview)和问题单定义资产类型仅在 case 项目中可用。

以下小节描述了 Red Hat Process Automation Manager 7.12 中的每个资产类型。

- 业务流程
  业务流程是描述实现商业目标所需步骤的图表。

- 问题单管理(Preview)
  案例管理是业务流程管理(BPM)的扩展，使您能够管理可适应性的业务流程。案例管理为非可重用、无法预计的进程提供了问题解决，而不是针对日常、可预测的任务提供 BPM 效率方法。当进程无法提前预测时，它管理一次性情况。

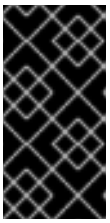
  重要

  业务流程应用程序示例仅包含技术预览。红帽产品服务等级协议(SLA)不支持技术预览功能，且可能并不完善，且不建议在生产环境中使用。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

- 问题单定义
  问题单使用 Business Central 中的 Case 定义流程设计程序进行设计。案例设计是案例管理的基础，并根据情况设定具体目标和任务。可以通过添加动态任务或进程在运行时动态修改案例流。

- 数据对象
  数据对象是您创建的规则资产的构建块。数据对象是项目指定软件包中作为 Java 对象实施的自定义数据类型。例如，您可以创建一个带有数据字段 Name、Address 和 Date of Birth 的 Person 对象，以指定 loan 应用程序规则的个人详情。这些自定义数据类型决定了您的资产和您的决定服务所基于的数据。

- 决策表（电子表格）
  决策表是存储在电子表格中的规则集合，或作为指导决策管理器用户界面存储在红帽决策管理器用户界面中。在外部 XLS 或 XLSX 文件中定义规则后，您可以在 Business Central 中将该文件作为您的项目中的决策表上传。



重要

您通常应该只上传一个路由表，其中包含 Business Central 中每个规则包的必要 **RuleTable** 定义。您可以为单独的软件包上传单独的路由表电子表格，但在同一个软件包中上传多个电子表格可能会导致冲突 **RuleSet** 或 **RuleTable** 属性中的编译错误，因此不建议这样做。

- DMN
  决策模型和注释(DMN)为业务决策设计和决策实施之间的差距提供了标准化的桥梁。您可以使用 Business Central 中的 DMN 设计器设计 DMN 决策要求图(DRD)，并为完整且可正常工作的 DMN 决策模型定义决策逻辑。

- DRL 文件
  规则文件通常是一个扩展名为 .drl 的文件。在 DRL 文件中，您可以有多个规则、查询和功能，以及导入、全局和查询等资源声明。但是，您还可以将规则分散到多个规则文件中（在这种情况下，建议使用扩展 .rule，但不是必需的）- 在文件中分散规则有助于管理大量规则。DRL 文件只是一个文本文件。

- DSL 定义
  域特定语言(DSL)是创建专用于您的问题域的规则语言的方法。一组 DSL 定义包括从 DSL "sentences" 转换为 DRL 构造，允许您使用所有底层规则语言和决策引擎功能。

- Enumeration
  数据枚举是一个可选资产类型，可以配置为为引导设计器提供下拉列表。它们像任何其他资产一样存储和编辑，并应用到它们所属的软件包。

- 格式
  形式用于收集用于处理过程中的用户数据。Business Central 提供自动生成表单的选项，然后可以对其进行编辑以满足特定的业务流程要求。

- 全局变量
  全局变量用于向规则提供应用对象。通常，它们用于提供规则使用的数据或服务，特别是规则使用的应用程序服务，以及从规则返回数据，如规则添加的日志或值，或者用于与应用程序交互的规则，以及执行回调。

- 指导决策表
  决策表是存储在电子表格中的规则集合，或作为指导决策管理器用户界面存储在红帽决策管理器用户界面中。

- 指导决策表图
  指导决策表 Graph 是一个相关指导决策表的集合，显示在单一设计人员中。您可以使用此设计器来更好地视觉化和处理一个位置中各种相关决策表。另外，当一个表中的条件或操作使用与另一个表中条件或操作相同的数据类型时，表将与表图设计器中的一行物理链接。

  例如，如果一个路由表确定了 loan 应用程序率，另一个表使用应用程序率来确定某些其他操作，则两个路由表在指导的路由表图中链接。

- 参考规则
  规则为决策引擎提供要针对的决策引擎的逻辑。规则包括名称、属性、规则左侧的 **when** 语句，以及规则右侧的 **then** 语句。

- 参考规则模板
  指导规则模板为编译到 Drools 规则语言(DRL)的多个规则提供可重复使用的规则结构，并为您的项目组成决策服务的核心。

- 软件包
  所有资产都包含在 Business Central 中的软件包中。软件包是规则的文件夹，也充当"命名空间"。

- solver 配置
  Solver 配置由 Solver designer 创建，可以在部署 KJAR 后在执行 Solver 或 plain Java 代码中运行。您可以在 Business Central 中编辑并创建 Solver 配置。

- 测试场景
  通过 Red Hat Process Automation Manager 测试场景，您可以在将它们部署到生产之前验证规则、模型和事件的功能。测试场景使用数据用于类似您事实或项目模型实例的条件。此数据与给定的一组规则匹配，如果预期结果与实际结果匹配，则测试成功。如果预期的结果与实际结果不匹配，则测试会失败。

- 测试场景(Legacy)
  Red Hat Process Automation Manager 7.12 包括对旧测试场景的支持，因为默认的测试场景资产仍在开发中。

- work Item 定义
  work 项定义定义如何显示自定义任务。例如，任务名称、图标、参数和类似的属性。

第 16 章 创建资产

您可以在 Business Central 项目中创建业务流程、规则、DRL 文件和其他资产。

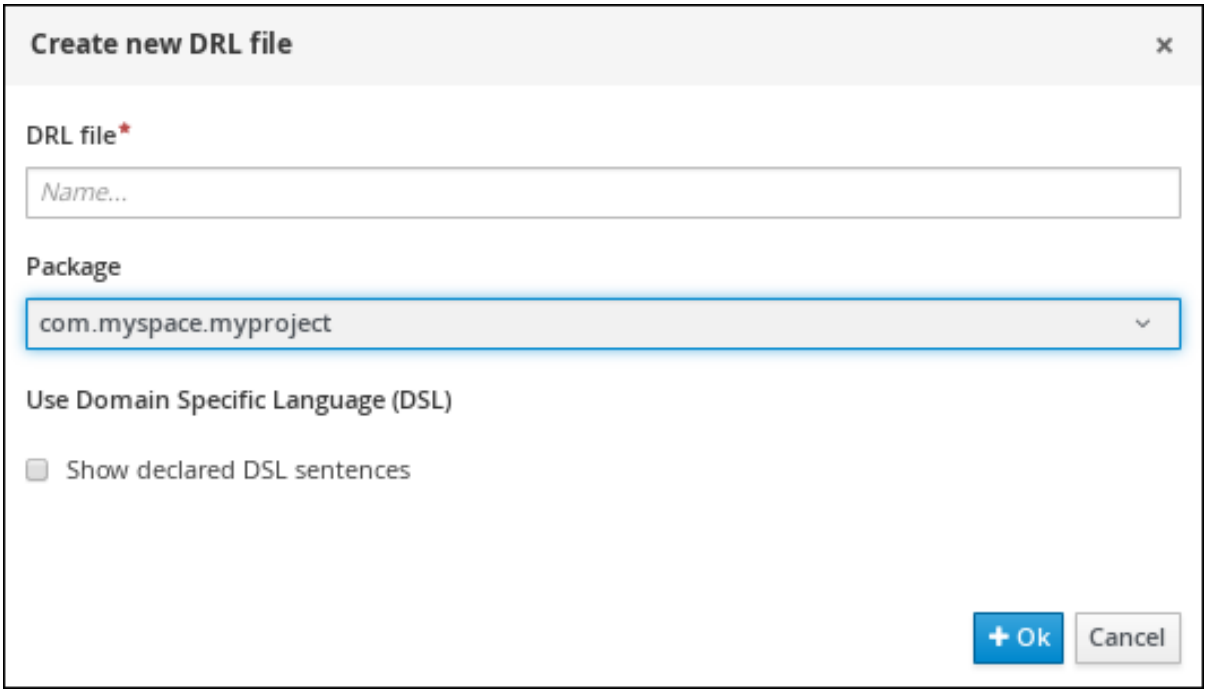
注意

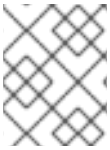
迁移业务流程是一个不可逆的过程。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。例如，**评估**。
2. 点 **Add Asset** 并选择 asset 类型。
3. 在 **Create new *asset_type*** 窗口中，添加所需信息并点 **Ok**。

图 16.1. 定义资产





注意

如果您还没有创建项目，您可以添加项目、使用示例项目或导入现有项目。如需更多信息，请参阅在 *Business Central 中管理项目*。

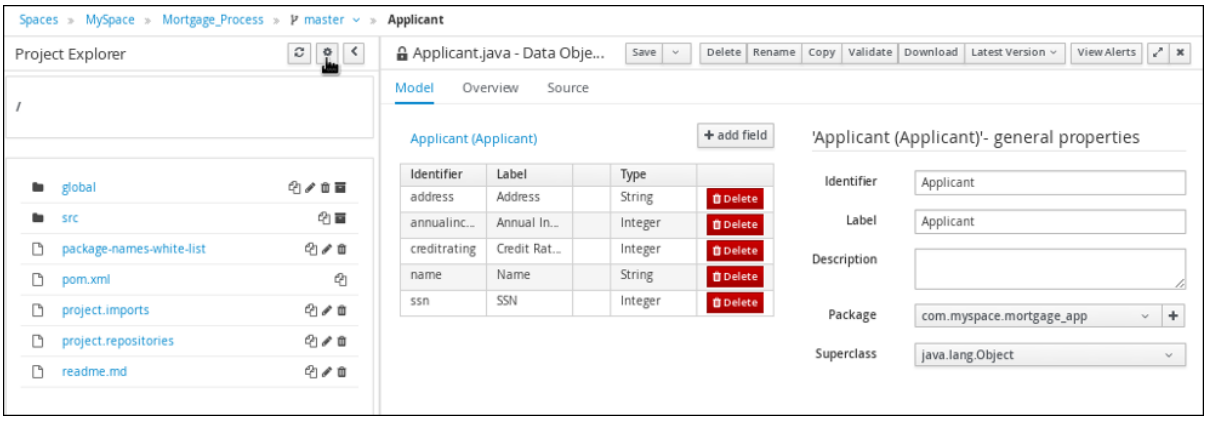
第 17 章 重命名、复制或删除资产

创建并定义了资产后，您可以使用 **Project Explorer** 的 **Repository View** 来根据需要复制、重命名、删除或归档资产。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。
2. 点资产名称，点左上角的  来扩展 **Project Explorer**。
3. 点击 **Project Explorer** 工具栏中的  并选择 **Repository View** 来显示组成资产的文件夹和文件。
4. 根据需要，使用每个列出资产旁的图标来复制、重命名、删除或归档资产。其中一些选项可能并不适用于所有资产。

图 17.1. 复制、重命名、删除或归档资产



5. 使用以下工具栏按钮来复制、重命名或删除资产。

图 17.2. 工具栏选项

第 18 章 管理资产元数据和版本历史记录

Business Central 中的大多数资产都有相关的元数据和版本信息，以帮助您在项目内识别和组织它们。您可以在 Business Central 中管理资产元数据和版本历史记录。

流程

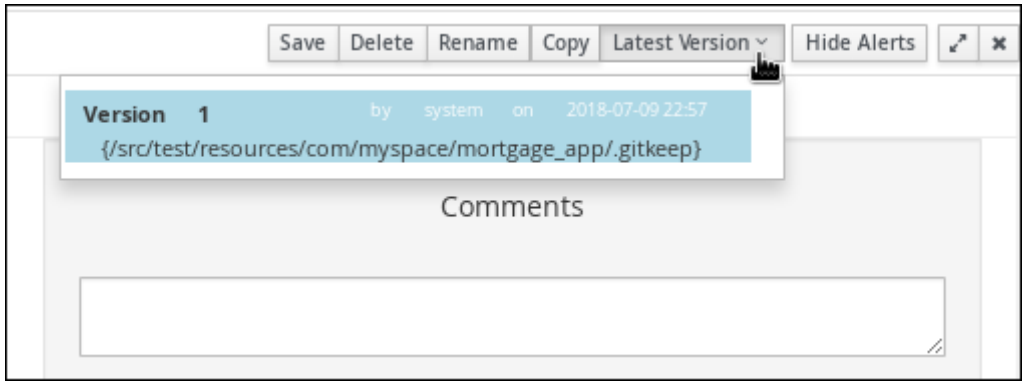
1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。
2. 从列表中选择资产以打开资产设计程序。
3. 在资产设计窗口中，选择 **Overview**。如果资产没有 **Overview** 选项卡，则不会与该资产关联的元数据。

4. 选择 **Version History** 或 **Metadata** 选项卡来编辑和更新版本和元数据详情。

注意

更新资产工作版本的另一种方式是点击资产设计器右上角的 **Latest Version**。

图 18.1. 资产的最新版本



5. 点 **Save** 保存更改。

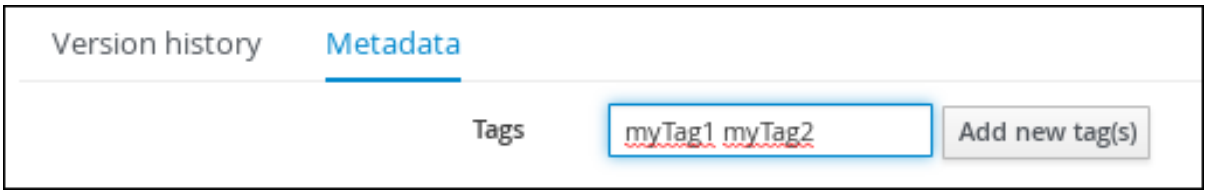
第 19 章 按标签过滤资产

您可以在每个资产的元数据中应用标签，然后按照 Project Explorer 中的标签对资产进行分组。这个功能可帮助您快速搜索特定类别的资产。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。
2. 点资产名称打开资产编辑器。
3. 在资产编辑器窗口中，进入 **Overview → Metadata**。
4. 在 **Tags** 字段中，输入新标签的名称，再单击 **Add new tag (s)**。您可以通过将标签名称与空格分开来为资产分配多个标签。

   图 19.1. 创建标签

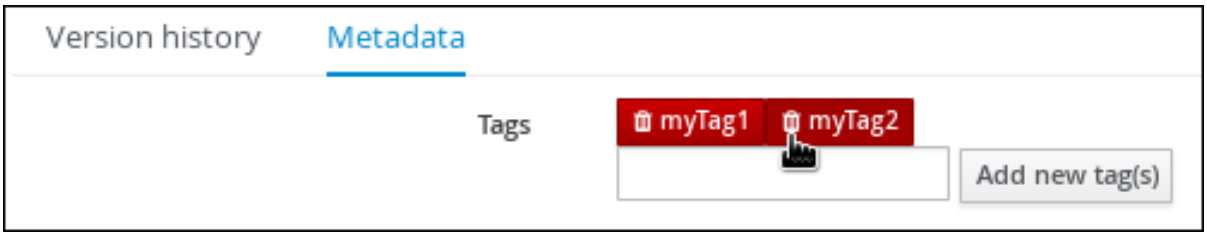
   分配的标签显示为 **Tags** 字段旁边的按钮。

   图 19.2. 元数据视图中的标签

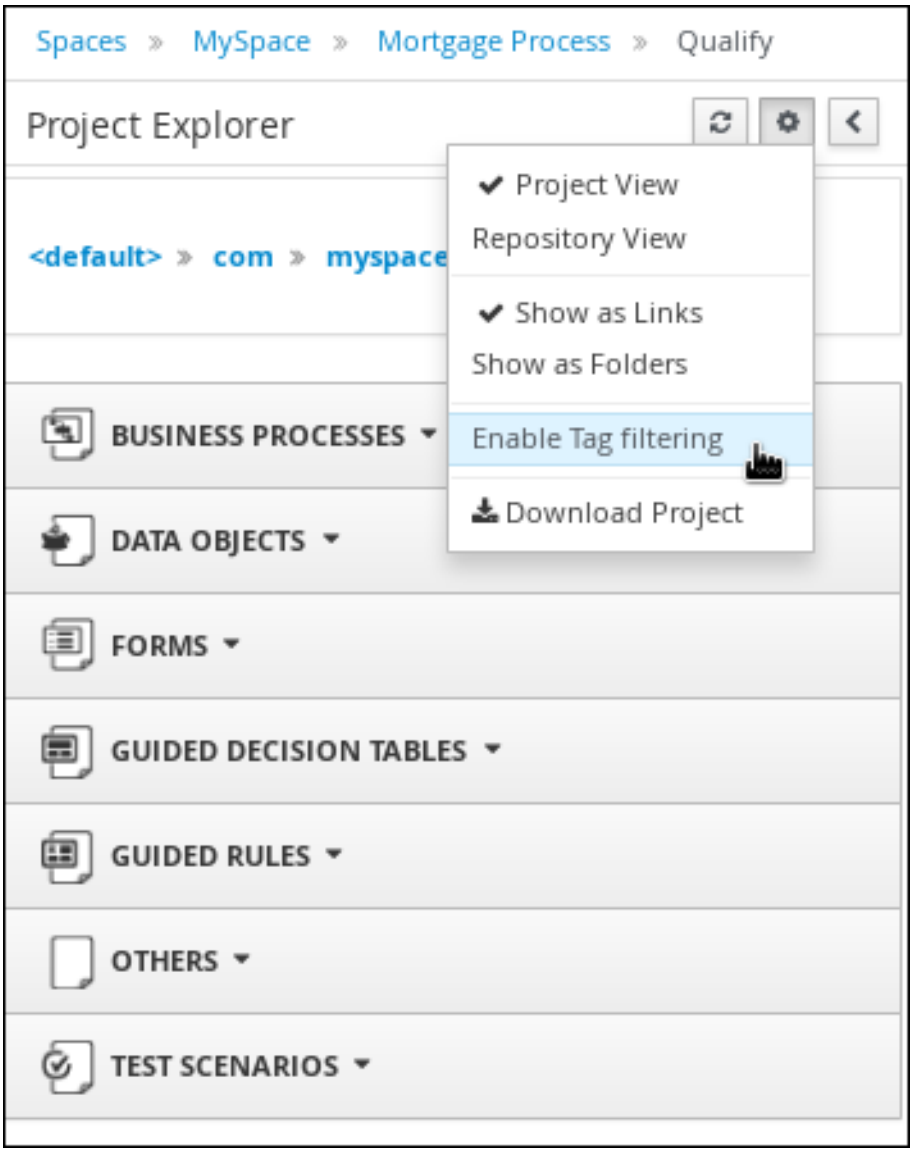
   单击标签按钮上的垃圾箱图标，以删除该标签。

   图 19.3. 删除元数据视图中的标签

5. 点 **Save** 保存您的更改。
6. 点击左上角的  ，展开 **Project Explorer**。
7. 点击 **Project Explorer** 工具栏中的  并选择 **Enable Tag 过滤**。

图 19.4. 启用标签过滤



这会在 Project Explorer 中显示 **Filter by Tag** 下拉菜单。

图 19.5. 按标签过滤

您可以通过此过滤器对资产进行排序，以显示包括所选元数据标签的所有资产和服务任务。

第 20 章 解锁资产

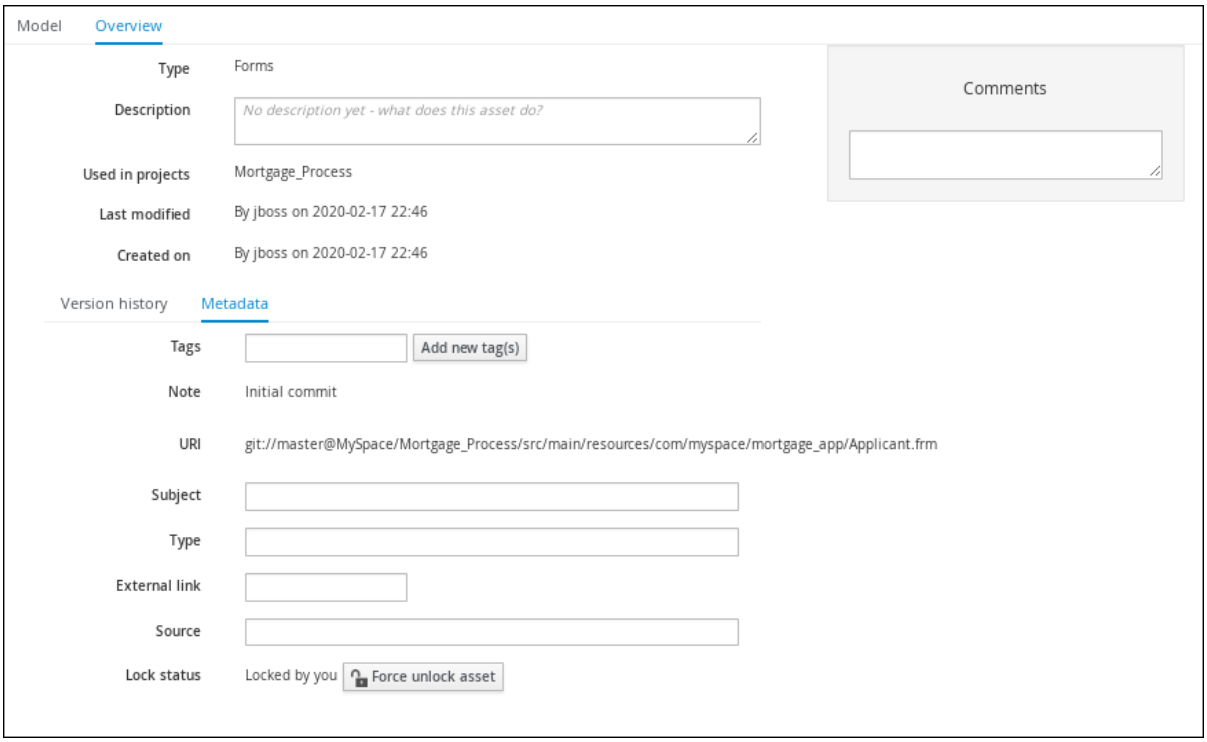
默认情况下，每当您在 Business Central 中打开和修改资产时，该资产将自动锁定给您的独家使用，以避免多用户设置冲突。当您会话结束或保存或关闭资产时，这个锁定会被自动释放。此锁定功能可确保用户不会覆盖彼此的更改。

但是，如果您需要编辑由另一个用户锁定的文件，您可以强制解锁资产。

流程

1. 在 Business Central 中，前往 **Menu → Design → Projects**，再单击项目名称。
2. 从列表中选择资产以打开资产设计程序。
3. 进入 **Overview → Metadata**，并查看 **Lock Status**。

   图 20.1. 解锁元数据视图



   如果资产已被另一个用户编辑，则会在 **Lock status** 字段中显示以下内容：

   被 <user_name> 锁定

4. 点 **Force unlock asset** 解锁。
   此时会显示以下确认弹出信息：

   您确定要释放此资产锁定吗？这可能导致 <user_name> 丢失未保存的更改！

5. 单击 **Yes** 进行确认。
   资产返回到解锁的状态，资产会在资产旁边显示锁定图标选项。

部分 IV. 使用 KIE API 与 RED HAT PROCESS AUTOMATION MANAGER 交互

作为业务规则开发人员或系统管理员，您可以使用 KIE API 与红帽流程自动化管理器中的 KIE 服务器、KIE 容器和业务资产进行交互。您可以使用 KIE Server REST API 和 Java 客户端 API 与 KIE 容器和业务资产（如业务规则、流程和解决程序）交互，使流程自动化管理器控制器 REST API 和 Java 客户端 API 与 KIE 服务器模板和实例交互，以及知识库 REST API 与 Business Central 中的空格和项目交互。

KIE 服务器和流程自动化管理器控制器的 REST API 端点

KIE 服务器以及 Process Automation Manager 控制器的 REST API 端点列表与本文档独立发布，并动态维护，以确保端点选项和数据尽可能最新。使用本文档了解 KIE Server 和 Process Automation Manager 控制器 REST API 的作用以及如何使用它们，并使用单独维护的 REST API 端点获取特定端点详情。

如需 KIE Server REST API 端点和描述的完整列表，请使用以下资源之一：

- 在 jBPM 文档页面的 Execution Server REST API（静态）
- KIE Server REST API 的 Swagger UI 位于 **http://SERVER:PORT/kie-server/docs**（动态，需要运行 KIE Server）

如需流程 Automation Manager 控制器 REST API 端点和描述的完整列表，请使用以下资源之一：

- JBPM 文档页面上的控制器 REST API （静态）
- 位于 **http://SERVER:PORT/CONTROLLER/docs** 的过程 Automation Manager 控制器 REST API 的 Swagger UI （需要运行 Process Automation Manager 控制器）

先决条件

- Red Hat Process Automation Manager 已安装并运行。有关安装和启动选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。
- 您可以使用以下用户角色访问 Red Hat Process Automation Manager：
  - **kie-server** ：用于访问 KIE Server API 功能，并在没有 Business Central 的情况下访问无头流程 Automation Manager 控制器 API 功能（如果适用）
  - **rest-all** ：为内置流程自动化管理器控制器和 Business Central 知识库访问 Business Central API 功能
  - **admin** ：控制对 Red Hat Process Automation Manager 的完全管理访问权限
    虽然每个 KIE API 都不需要这些用户角色，但请考虑获取所有它们以确保您可以在不中断的情况下访问任何 KIE API。有关用户角色的更多信息，请参阅 *规划红帽流程自动化管理器安装*。

第 21 章 KIE 服务器 REST API 用于 KIE 容器和业务资产

Red Hat Process Automation Manager 提供了一个 KIE Server REST API，您可以在不使用 Business Central 用户界面的情况下与 KIE 容器和业务资产（如业务规则、流程和解决器）进行交互。通过此 API 支持，您可以更有效地维护红帽流程自动化管理器资源，并优化您的与红帽流程自动化管理器的集成和开发。

使用 KIE Server REST API，您可以执行以下操作：

- 部署或分离 KIE 容器
- 检索和更新 KIE 容器信息
- 返回 KIE 服务器状态和基本信息
- 检索和更新业务资产信息
- 执行业务资产（如规则和流程）

KIE Server REST API 请求需要以下组件：

身份验证

KIE Server REST API 需要 HTTP 基本身份验证或基于令牌的身份验证，用于用户角色 **kie-server**。要查看为您的 Red Hat Process Automation Manager 分发配置的用户角色，请导航到 **~/$SERVER_HOME/standalone/configuration/application-roles.properties** 和 **~/application-users.properties**。
要添加具有 **kie-server** 角色的用户，请导航到 **~/$SERVER_HOME/bin**，再运行以下命令：

```
$ ./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'])"
```

有关用户角色和 Red Hat Process Automation Manager 安装选项的更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装*。

HTTP 标头

KIE Server REST API 需要以下用于 API 请求的 HTTP 标头：

- **接受**: 请求客户端接受的数据格式：
  - **application/json** (JSON)
  - **application/xml** (XML，用于 JAXB 或 XSTREAM)
- **Content-Type**: **POST** 或 **PUT** API 请求数据的数据格式：
  - **application/json** (JSON)
  - **application/xml** (XML，用于 JAXB 或 XSTREAM)
- **X-KIE-ContentType**: **application/xml** XSTREAM API 请求和响应所需的标头：
  - **XSTREAM**

HTTP 方法

KIE 服务器 REST API 支持以下 API 请求的 HTTP 方法：

- **GET** ：从指定的资源端点检索指定的信息
- **POST** ：更新资源或资源实例
- **PUT** ：更新或创建资源实例
- **DELETE** ：删除资源或资源实例

基本 URL

KIE 服务器 REST API 请求的基本 URL 是 **http://SERVER:PORT/kie-server/services/rest/**，如 **http://localhost:8080/kie-server/services/rest/**。

Endpoints

KIE Server REST API 端点（如指定 KIE 容器的 **/server/containers/{containerId}**）是您附加到 KIE Server REST API 基础 URL 的 URI，以访问红帽流程自动化管理器中的相应资源或资源类型。

/server/containers/{containerId} 端点的请求 URL 示例

http://localhost:8080/kie-server/services/rest/server/containers/MyContainer

请求参数和请求数据

许多 KIE 服务器 REST API 请求需要请求 URL 路径中的特定参数来识别或过滤特定资源并执行特定操作。您可以在端点中添加 URL 参数，格式为 **?<PARAM>=<VALUE>&<PARAM>=<VALUE** >。

带有参数的 GET 请求 URL 示例

http://localhost:8080/kie-server/services/rest/server/containers?groupId=com.redhat&artifactId=Project1&version=1.0&status=STARTED

HTTP **POST** 和 **PUT** 请求可能还需要请求正文或文件，并附带请求。

POST 请求 URL 和 JSON 请求正文数据示例

http://localhost:8080/kie-server/services/rest/server/containers/MyContainer/release-id

```
{
  "release-id": {
    "artifact-id": "Project1",
    "group-id": "com.redhat",
    "version": "1.1"
  }
}
```

21.1. 使用 REST 客户端或 CURL 工具使用 KIE 服务器 REST API 发送请求

KIE 服务器 REST API 允许您在红帽流程自动化管理器中与 KIE 容器和业务资产（如业务规则、流程和解决程序）交互，而无需使用 Business Central 用户界面。您可以使用任何 REST 客户端或 curl 工具发送 KIE 服务器 REST API 请求。

先决条件

- KIE 服务器已安装并运行。
- 您有 **kie-server** 用户角色对 KIE 服务器的访问权限。

流程

1. 找到您要发送请求的相关 API 端点，如 **[GET] /server/containers** 从 KIE Server 检索 KIE 容器。
2. 在 REST 客户端或 curl 工具中，输入到 **/server/containers** 的 **GET** 请求的以下组件。根据您的用例调整任何请求详情。
   对于 REST 客户端：
   - **身份验证** ：使用 **kie-server** 角色输入 KIE Server 用户的用户名和密码。
   - **HTTP Headers** ：设置以下标头：
     - **接受:application/json**
   - **HTTP 方法** ：设置为 **GET**。
   - **URL** ：输入 KIE Server REST API 基础 URL 和端点，如 **http://localhost:8080/kie-server/services/rest/server/containers**。

   对于 curl 工具：
   - **-u** ：使用 **kie-server** 角色输入 KIE Server 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 Server REST API 基础 URL 和端点，如 **http://localhost:8080/kie-server/services/rest/server/containers**。

   ```
   curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
   "http://localhost:8080/kie-server/services/rest/server/containers"
   ```

3. 执行请求并查看 KIE 服务器响应。
   服务器响应示例(JSON)：

   ```
   {
     "type": "SUCCESS",
     "msg": "List of created containers",
     "result": {
       "kie-containers": {
         "kie-container": [
           {
             "container-id": "itorders_1.0.0-SNAPSHOT",
             "release-id": {
               "group-id": "itorders",
               "artifact-id": "itorders",
               "version": "1.0.0-SNAPSHOT"
             },
             "resolved-release-id": {
   ```

```
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [],
          "container-alias": "itorders"
        }
      ]
    }
  }
}
```

4. 在本例中，复制或记下来自其中一个部署的 KIE 容器的项目 **group-id**、**artifact-id 和版本** (GAV) 数据。

5. 在 REST 客户端或 curl 实用程序中，发送带有以下内容的另外一个 API 请求，它向 **/server/containers/{containerId}** 发送一个 **PUT** 请求，以使用复制的项目 GAV 数据部署新的 KIE 容器。根据您的用例调整任何请求详情。
   对于 REST 客户端：

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

     注意

     当您将 **fields=not_null** 添加到 **Content-Type** 时，会从 REST API 响应中排除 null 字段。

   - **HTTP 方法** ：设置为 **PUT**。
   - **URL** ：输入 KIE Server REST API 基础 URL 和端点，如 **http://localhost:8080/kie-server/services/rest/server/containers/MyContainer**。
   - **请求正文** ：添加带有新 KIE 容器的配置项的 JSON 请求正文：

```
{
  "config-items": [
    {
      "itemName": "RuntimeStrategy",
      "itemValue": "SINGLETON",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "MergeMode",
      "itemValue": "MERGE_COLLECTIONS",
```

```
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KBase",
      "itemValue": "",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KSession",
      "itemValue": "",
      "itemType": "java.lang.String"
    }
  ],
  "release-id": {
    "group-id": "itorders",
    "artifact-id": "itorders",
    "version": "1.0.0-SNAPSHOT"
  },
  "scanner": {
    "poll-interval": "5000",
    "status": "STARTED"
  }
}
```

对于 curl 工具：

- **-u** ：使用 **kie-server** 角色输入 KIE Server 用户的用户名和密码。
- **-** h ：设置以下标头：
  - **接受:application/json**
  - **Content-Type:application/json**

注意

当您将 **fields=not_null** 添加到 **Content-Type** 时，会从 REST API 响应中排除 null 字段。

- **-x** ：设置为 **PUT**。
- **URL** ：输入 KIE Server REST API 基础 URL 和端点，如 **http://localhost:8080/kie-server/services/rest/server/containers/MyContainer**。
- **-d** ：添加 JSON 请求正文或文件(**@file.json**)，其中包含新 KIE 容器的配置项：

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type:
application/json" -X PUT "http://localhost:8080/kie-
server/services/rest/server/containers/MyContainer" -d "{ \"config-items\": [ { \"itemName\":
\"RuntimeStrategy\", \"itemValue\": \"SINGLETON\", \"itemType\": \"java.lang.String\" }, {
\"itemName\": \"MergeMode\", \"itemValue\": \"MERGE_COLLECTIONS\", \"itemType\":
\"java.lang.String\" }, { \"itemName\": \"KBase\", \"itemValue\": \"\", \"itemType\":
\"java.lang.String\" }, { \"itemName\": \"KSession\", \"itemValue\": \"\", \"itemType\":
\"java.lang.String\" } ], \"release-id\": { \"group-id\": \"itorders\", \"artifact-id\": \"itorders\",
\"version\": \"1.0.0-SNAPSHOT\" }, \"scanner\": { \"poll-interval\": \"5000\", \"status\":
\"STARTED\" }}"
```

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type:
application/json" -X PUT "http://localhost:8080/kie-
server/services/rest/server/containers/MyContainer" -d @my-container-configs.json
```

6. 执行请求并查看 KIE 服务器响应。
服务器响应示例(JSON)：


```
{
  "type": "SUCCESS",
  "msg": "Container MyContainer successfully deployed with module itorders:itorders:1.0.0-
SNAPSHOT.",
  "result": {
    "kie-container": {
      "container-id": "MyContainer",
      "release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "resolved-release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "status": "STARTED",
      "scanner": {
        "status": "STARTED",
        "poll-interval": 5000
      },
      "config-items": [],
      "messages": [
        {
          "severity": "INFO",
          "timestamp": {
            "java.util.Date": 1540584717937
          },
          "content": [
            "Container MyContainer successfully created with module itorders:itorders:1.0.0-
SNAPSHOT."
          ]
        }
      ],
      "container-alias": null
    }
  }
}
```


如果您遇到请求错误，请检查返回的错误消息并相应地调整您的请求。

处理实例的 REST API 请求

对于将复杂数据对象发送到进程实例端点 **/server/containers/{containerId}/processes/{processId}/instances** 的 REST API 请求，请确保在请求正文中包含完全限定类名称（如 **com.myspace.Person**）或简单类名称（如 **Person**）。请求正文需要类名称映射到 Red Hat Process Automation Manager 中的正确业务对象。如果您从请求中排除类名称，KIE Server 不会将对象卸载到预期的类型。

进程实例的正确请求正文

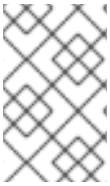
```
{
  "id": 4,
  "lease": {
    "com.myspace.restcall.LeaseModel": {
      "annualRent": 109608,
      "isAutoApproved": false
    }
  }
}
```

进程实例的请求正文不正确

```
{
  "id": 4,
  "lease": {
    "annualRent": 109608,
    "isAutoApproved": false
  }
}
```

21.2. 使用 SWAGGER 接口通过 KIE SERVER REST API 发送请求

KIE 服务器 REST API 支持 Swagger Web 界面，您可以使用它而不是独立 REST 客户端或 curl 实用程序与 KIE 容器和业务资产（如业务规则、流程和解决器）进行交互，而无需使用 Business Central 用户界面。



注意

默认情况下，KIE 服务器的 Swagger Web 界面由 **org.kie.swagger.server.ext.disabled=false** 系统属性启用。要在 KIE 服务器中禁用 Swagger Web 界面，请将这个系统属性设置为 **true**。

先决条件

- KIE 服务器已安装并运行。
- 您有 **kie-server** 用户角色对 KIE 服务器的访问权限。

流程

1. 在 Web 浏览器中，导航到 **http://SERVER:PORT/kie-server/docs**，如 **http://localhost:8080/kie-server/docs**，并使用 KIE Server 用户的用户名和密码登录，使用 **kie-server** 角色。

2. 在 Swagger 页面中，选择要发送请求的相关 API 端点，如 **KIE Server 和 KIE containers→ [GET] /server/containers** 从 KIE Server 检索 KIE 容器。

3. 点 **Try it out**，并提供您要过滤结果的任何可选参数。

4. 在 **Response 内容类型下拉菜单中**，选择服务器响应所需的格式，如用于 JSON 格式的 **application/json**。

5. 点 **Execute** 并查看 KIE Server 响应。
服务器响应示例(JSON)：


```
{
  "type": "SUCCESS",
  "msg": "List of created containers",
  "result": {
    "kie-containers": {
      "kie-container": [
        {
          "container-id": "itorders_1.0.0-SNAPSHOT",
          "release-id": {
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "resolved-release-id": {
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [],
          "container-alias": "itorders"
        }
      ]
    }
  }
}
```


6. 在本例中，复制或记下来自其中一个部署的 KIE 容器的项目 **group-id**、**artifact-id 和版本** (GAV) 数据。

7. 在 Swagger 页面中，导航到 **KIE Server 和 KIE containers→ [PUT] /server/containers/{containerId}** 端点，以发送另一个请求以使用复制的项目 GAV 数据部署新的 KIE 容器。根据您的用例调整任何请求详情。

8. 点击 **Try it out**，并为请求输入以下组件：

   - **containerId** ：输入新 KIE 容器的 ID，如 **MyContainer**。

   - **body** ： **将参数内容类型设置**为 所需的请求正文格式，如用于 JSON 格式的 **application/json**，并使用新 KIE 容器的配置项添加请求正文：

```
{
  "config-items": [
    {
      "itemName": "RuntimeStrategy",
      "itemValue": "SINGLETON",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "MergeMode",
      "itemValue": "MERGE_COLLECTIONS",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KBase",
      "itemValue": "",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KSession",
      "itemValue": "",
      "itemType": "java.lang.String"
    }
  ],
  "release-id": {
    "group-id": "itorders",
    "artifact-id": "itorders",
    "version": "1.0.0-SNAPSHOT"
  },
  "scanner": {
    "poll-interval": "5000",
    "status": "STARTED"
  }
}
```

9. 在 **Response 内容类型下拉菜单中**，选择服务器响应所需的格式，如用于 JSON 格式的 **application/json**。

10. 点 **Execute** 并查看 KIE Server 响应。
服务器响应示例(JSON)：

```
{
  "type": "SUCCESS",
  "msg": "Container MyContainer successfully deployed with module itorders:itorders:1.0.0-SNAPSHOT.",
  "result": {
    "kie-container": {
      "container-id": "MyContainer",
      "release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "resolved-release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
```

```
        "version": "1.0.0-SNAPSHOT"
      },
      "status": "STARTED",
      "scanner": {
        "status": "STARTED",
        "poll-interval": 5000
      },
      "config-items": [],
      "messages": [
        {
          "severity": "INFO",
          "timestamp": {
            "java.util.Date": 1540584717937
          },
          "content": [
            "Container MyContainer successfully created with module itorders:itorders:1.0.0-
SNAPSHOT."
          ]
        }
      ],
      "container-alias": null
    }
  }
}
```


如果您遇到请求错误，请检查返回的错误消息并相应地调整您的请求。

处理实例的 REST API 请求

对于将复杂数据对象发送到进程实例端点 **/server/containers/{containerId}/processes/{processId}/instances** 的 REST API 请求，请确保在请求正文中包含完全限定类名称（如 **com.myspace.Person**）或简单类名称（如 **Person**）。请求正文需要类名称映射到 Red Hat Process Automation Manager 中的正确业务对象。如果您从请求中排除类名称，KIE Server 不会将对象卸载到预期的类型。

进程实例的正确请求正文

```
{
  "id": 4,
  "lease": {
    "com.myspace.restcall.LeaseModel": {
      "annualRent": 109608,
      "isAutoApproved": false
    }
  }
}
```

进程实例的请求正文不正确

```
{
  "id": 4,
  "lease": {
    "annualRent": 109608,
    "isAutoApproved": false
  }
}
```

21.3. 支持的 KIE 服务器 REST API 端点

KIE Server REST API 为 Red Hat Process Automation Manager 中的以下资源类型提供端点：

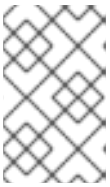
- KIE 服务器和 KIE 容器
- KIE 会话资产（用于运行时命令）
- DMN 资产
- 规划解决者
- Process
- 进程镜像
- 流程和任务表单
- 任务
- 情况
- 文档

- Jobs
- 查询进程、任务和问题单
- 自定义查询

KIE 服务器 REST API 基本 URL 是 **http://SERVER:PORT/kie-server/services/rest/**。所有请求都需要 **kie-server** 用户角色的基于 HTTP 基本身份验证或基于令牌的身份验证。

如需 KIE Server REST API 端点和描述的完整列表，请使用以下资源之一：

- 在 jBPM 文档页面的 Execution Server REST API（静态）
- KIE Server REST API 的 Swagger UI 位于 **http://SERVER:PORT/kie-server/docs**（动态，需要运行 KIE Server）



注意

默认情况下，KIE 服务器的 Swagger Web 界面由 **org.kie.swagger.server.ext.disabled=false** 系统属性启用。要在 KIE 服务器中禁用 Swagger Web 界面，请将这个系统属性设置为 **true**。

为了使 API 可以访问处理镜像，需要在 **$SERVER_HOME/standalone/deployments/business-central.war/org.kie.workbench.KIEWebapp/profiles/jbpm.xml** 中为您的 Red Hat Process Automation Manager 项目配置系统属性 **<storesvgonsave enabled="true"/>**。此属性默认设置为 **true**。如果 API 没有使用进程镜像，在 文件中将其设置为 **true**，重启 KIE Server，修改相关进程并保存它，然后构建和部署项目。此属性允许存储 SVG 镜像，以便可以通过 KIE Server REST API 检索它们。

21.3.1. 自定义查询

您可以使用自定义查询端点在 Red Hat Process Automation Manager 中创建和访问自定义查询。自定义查询可以从 Red Hat Process Automation Manager 数据库请求任何数据。

Red Hat Process Automation Manager 中包含大量自定义查询。您可以使用这些查询来访问进程实例和用户任务的完整列表。

在运行一个自定义查询时，您必须在 **mapper** 参数中提供一个 *query mapper* 的名称。映射器将 SQL 查询结果映射到对象以获取 JSON 响应。您可以实施自己的查询结果映射程序，或使用 Red Hat Process Automation Manager 提供的映射程序。红帽流程自动化管理器中的查询映射器与其他对象关系映射(ORM)供应商类似，如 Hibernate，将表映射到实体。

例如，如果自定义查询返回进程实例数据，您可以使用 **org.jbpm.kie.services.impl.query.mapper.ProcessInstanceQueryMapper** mapper，也注册为 **ProcessInstances**。如果自定义查询返回人工任务数据，您可以使用 **org.jbpm.kie.services.impl.query.mapper.UserTaskInstanceQueryMapper** mapper，也可以注册为 **UserTasks**。您还可以使用提供额外信息的其他映射程序。

有关 Red Hat Process Automation Manager 中包含的查询映射程序列表，请参阅 GitHub 存储库。

21.3.2. 特定 DMN 模型的 REST 端点

Red Hat Process Automation Manager 提供特定于模型的 DMN KIE 服务器端点，可用于与特定的 DMN 模型交互，而无需使用 Business Central 用户界面。

对于 Red Hat Process Automation Manager 中容器中的每个 DMN 模型，以下 KIE Server REST 端点会根据 DMN 模型的内容自动生成：

- **POST /server/containers/{containerId}/dmn/models/{modelname}**: 一个 business-domain 端点用于评估容器中的指定 DMN 模型
- **POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}**: 一个 business-domain 端点，用于在容器中在特定 DMN 模型中评估指定的决策服务组件
- **POST /server/containers/{containerId}/dmn/models/{modelname}/dmnresult** ：用于评估指定 DMN 模型的端点，其中包含自定义正文有效负载并返回 **DMNResult** 响应，包括业务域上下文、帮助程序消息和帮助程序决策指针
- **POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}/dmnresult**: 一个端点，用于在特定 DMN 模型中评估指定的决策服务组件，并返回 **DMNResult** 响应，包括业务域上下文、帮助程序信息以及帮助决策服务中的决策指针
- **GET /server/containers/{containerId}/dmn/models/{modelname}**: 一个端点，用于在没有决策逻辑的情况下返回标准 DMN XML，并包含指定 DMN 模型的输入和输出
- **GET /server/containers/{containerId}/dmn/openapi.json (|.yaml)**: 一个用于检索 Swagger 或 OAS 的端点，用于指定容器中的 DMN 模型

您可以使用这些端点与 DMN 模型或模型中的特定决策服务交互。当您决定使用这些 REST 端点的 business-domain 和 **dmnresult** 变体间，请查看以下注意事项：

- **REST 业务域端点** ：如果客户端应用程序只关注正评估结果，则使用此端点类型，不对解析 **Info** 或 **Warn** 信息感兴趣，并且只需要 HTTP 5xx 响应任何错误。由于类似于 DMN 建模行为的单例服务结果，这种端点对单页应用（如单页应用）也很有用。
- **REST dmnresult 端点** ：如果客户端需要解析 **Info**、**Warn** 或 **Error** 信息，则使用此端点类型。

对于每个端点，使用 REST 客户端或 curl 工具发送带有以下组件的请求：

- **基本 URL:http://*HOST*:*PORT*/kie-server/services/rest/**
- **路径参数** ：
  - **{containerId}**: 容器的字符串标识符，如 **mykjar-project**
  - **{modelName}**: DMN 模型的字符串标识符，如 **流量冲突**
  - **{decisionServiceName}**: DMN DRG 中的决策服务组件的字符串标识符，如 **TrafficViolationDecisionService**
  - **dmnresult** ：字符串标识符，使端点返回一个更加详细的 **DMNResult** 响应，它带有更详细的 **Info**, **Warn**, 和 **Error** 信息。
- HTTP 标头: 只适用对于 **POST** 请求：
  - **接受:application/json**
  - **Content-type:application/json**
- HTTP 方法 ：**GET** 或 **POST**

以下端点中的示例基于 **mykjar-project** 容器，该容器包含 **流量冲突** DMN 模型，其中包含 **流量 ViolationDecisionService** 决策服务组件。

对于所有这些端点，如果出现 DMN 评估 **Error** 消息，则返回 **DMNResult** 响应并带有 HTTP 5xx 错误。如果发生 DMN **Info** 或 **Warn** 消息，则会在 **X-Kogito-decision-messages** 扩展 HTTP 头中返回相关响应，以用于客户端业务逻辑。当需要更完善的客户端业务逻辑时，客户端可以使用端点的 **dmnresult** 变体。

为指定容器中的 DMN 模型检索 Swagger 或 OAS

GET /server/containers/{containerId}/dmn/openapi.json (|.yaml)

REST 端点示例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/openapi.json(|.YAML)

返回没有决策逻辑的 DMN XML

GET /server/containers/{containerId}/dmn/models/{modelname}

REST 端点示例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic 违反

curl 请求示例

```
curl -u wbadmin:wbadmin -X GET "http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic%20Violation" -H "accept: application/xml"
```

响应示例(XML)

```
<?xml version='1.0' encoding='UTF-8'?>
<dmn:definitions xmlns:dmn="http://www.omg.org/spec/DMN/20180521/MODEL/" xmlns="https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF" xmlns:di="http://www.omg.org/spec/DMN/20180521/DI/" xmlns:kie="http://www.drools.org/kie/dmn/1.2" xmlns:feel="http://www.omg.org/spec/DMN/20180521/FEEL/" xmlns:dmndi="http://www.omg.org/spec/DMN/20180521/DMNDI/" xmlns:dc="http://www.omg.org/spec/DMN/20180521/DC/" id="_1C792953-80DB-4B32-99EB-25FBE32BAF9E" name="Traffic Violation" expressionLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/" typeLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/" namespace="https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF">
  <dmn:extensionElements/>
  <dmn:itemDefinition id="_63824D3F-9173-446D-A940-6A7F0FA056BB" name="tDriver" isCollection="false">
    <dmn:itemComponent id="_9DAB5DAA-3B44-4F6D-87F2-95125FB2FEE4" name="Name" isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_856BA8FA-EF7B-4DF9-A1EE-E28263CE9955" name="Age" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
```

```
    </dmn:itemComponent>
    <dmn:itemComponent id="_FDC2CE03-D465-47C2-A311-98944E8CC23F" name="State"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_D6FD34C4-00DC-4C79-B1BF-BBCF6FC9B6D7" name="City"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_7110FE7E-1A38-4C39-B0EB-AEEF06BA37F4" name="Points"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_40731093-0642-4588-9183-1660FC55053B" name="tViolation"
isCollection="false">
    <dmn:itemComponent id="_39E88D9F-AE53-47AD-B3DE-8AB38D4F50B3" name="Code"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_1648EA0A-2463-4B54-A12A-D743A3E3EE7B" name="Date"
isCollection="false">
      <dmn:typeRef>date</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_9F129EAA-4E71-4D99-B6D0-84EEC3AC43CC" name="Type"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
      <dmn:allowedValues kie:constraintType="enumeration" id="_626A8F9C-9DD1-44E0-9568-
0F6F8F8BA228">
        <dmn:text>"speed", "parking", "driving under the influence"</dmn:text>
      </dmn:allowedValues>
    </dmn:itemComponent>
    <dmn:itemComponent id="_DDD10D6E-BD38-4C79-9E2F-8155E3A4B438" name="Speed
Limit" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_229F80E4-2892-494C-B70D-683ABF2345F6" name="Actual
Speed" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_2D4F30EE-21A6-4A78-A524-A5C238D433AE" name="tFine"
isCollection="false">
    <dmn:itemComponent id="_B9F70BC7-1995-4F51-B949-1AB65538B405" name="Amount"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_F49085D6-8F08-4463-9A1A-EF6B57635DBD" name="Points"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:inputData id="_1929CBD5-40E0-442D-B909-49CEDE0101DC" name="Violation">
    <dmn:variable id="_C16CF9B1-5FAB-48A0-95E0-5FCD661E0406" name="Violation"
typeRef="tViolation"/>
  </dmn:inputData>
```

```
  <dmn:decision id="_4055D956-1C47-479C-B3F4-BAEB61F1C929" name="Fine">
    <dmn:variable id="_8C1EAC83-F251-4D94-8A9E-B03ACF6849CD" name="Fine"
typeRef="tFine"/>
    <dmn:informationRequirement id="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
      <dmn:requiredInput href="#_1929CBD5-40E0-442D-B909-49CEDE0101DC"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmn:inputData id="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" name="Driver">
    <dmn:variable id="_A80F16DF-0DB4-43A2-B041-32900B1A3F3D" name="Driver"
typeRef="tDriver"/>
  </dmn:inputData>
  <dmn:decision id="_8A408366-D8E9-4626-ABF3-5F69AA01F880" name="Should the driver be
suspended?">
    <dmn:question>Should the driver be suspended due to points on his license?</dmn:question>
    <dmn:allowedAnswers>"Yes", "No"</dmn:allowedAnswers>
    <dmn:variable id="_40387B66-5D00-48C8-BB90-E83EE3332C72" name="Should the driver be
suspended?" typeRef="string"/>
    <dmn:informationRequirement id="_982211B1-5246-49CD-BE85-3211F71253CF">
      <dmn:requiredInput href="#_1F9350D7-146D-46F1-85D8-15B5B68AF22A"/>
    </dmn:informationRequirement>
    <dmn:informationRequirement id="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
      <dmn:requiredDecision href="#_4055D956-1C47-479C-B3F4-BAEB61F1C929"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmndi:DMNDI>
    <dmndi:DMNDiagram>
      <di:extension/>
      <dmndi:DMNShape id="dmnshape-_1929CBD5-40E0-442D-B909-49CEDE0101DC"
dmnElementRef="_1929CBD5-40E0-442D-B909-49CEDE0101DC" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="708" y="350" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_4055D956-1C47-479C-B3F4-BAEB61F1C929"
dmnElementRef="_4055D956-1C47-479C-B3F4-BAEB61F1C929" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="709" y="210" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_1F9350D7-146D-46F1-85D8-15B5B68AF22A"
dmnElementRef="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="369" y="344" width="100" height="50"/>
        <dmndi:DMNLabel/>
```

```
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_8A408366-D8E9-4626-ABF3-5F69AA01F880"
dmnElementRef="_8A408366-D8E9-4626-ABF3-5F69AA01F880" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="534" y="83" width="133" height="63"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNEdge id="dmnedge-_800A3BBB-90A3-4D9D-BA5E-A311DED0134F"
dmnElementRef="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
        <di:waypoint x="758" y="375"/>
        <di:waypoint x="759" y="235"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_982211B1-5246-49CD-BE85-3211F71253CF"
dmnElementRef="_982211B1-5246-49CD-BE85-3211F71253CF">
        <di:waypoint x="419" y="369"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_AEC4AA5F-50C3-4FED-A0C2-261F90290731"
dmnElementRef="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
        <di:waypoint x="759" y="235"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
    </dmndi:DMNDiagram>
  </dmndi:DMNDI>
```

在指定容器中评估指定的 DMN 模型

POST /server/containers/{containerId}/dmn/models/{modelname}

REST 端点示例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic 违反

curl 请求示例

```
curl -u wbadmin:wbadmin-X POST "http://localhost:8080/kie-
server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation" -H  "accept:
application/json" -H  "Content-Type: application/json" -d "{\"Driver\":{\"Points\":15},\"Violation\":
{\"Date\":\"2021-04-08\",\"Type\":\"speed\",\"Actual Speed\":135,\"Speed Limit\":100}}"
```

使用输入数据的 POST 请求正文示例

```
{
  "Driver": {
    "Points": 15
  },
  "Violation": {
    "Date": "2021-04-08",
    "Type": "speed",
    "Actual Speed": 135,
```

```
    "Speed Limit": 100
  }
}
```

响应示例(JSON)

```
{
  "Violation": {
    "Type": "speed",
    "Speed Limit": 100,
    "Actual Speed": 135,
    "Code": null,
    "Date": "2021-04-08"
  },
  "Driver": {
    "Points": 15,
    "State": null,
    "City": null,
    "Age": null,
    "Name": null
  },
  "Fine": {
    "Points": 7,
    "Amount": 1000
  },
  "Should the driver be suspended?": "Yes"
}
```

在容器中的指定 DMN 模型中评估指定的决策服务

POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}
对于此端点，请求正文必须包含决策服务的所有要求。响应是决策服务生成的 DMN 上下文，包括决策值、原始输入值以及所有其他 parametric DRG 组件（以序列化格式）。例如，业务知识模型以字符串的签名形式提供。

如果决策服务由单输出决策组成，则响应是该特定决策的结果值。在模型本身调用决策服务时，此行为在规格功能的 API 级别提供等效的值。例如，您可以从单页 web 应用程序与 DMN 决策服务交互。

图 21.1. 带有单输出决定 的流量ViolationDecisionService 决策服务示例

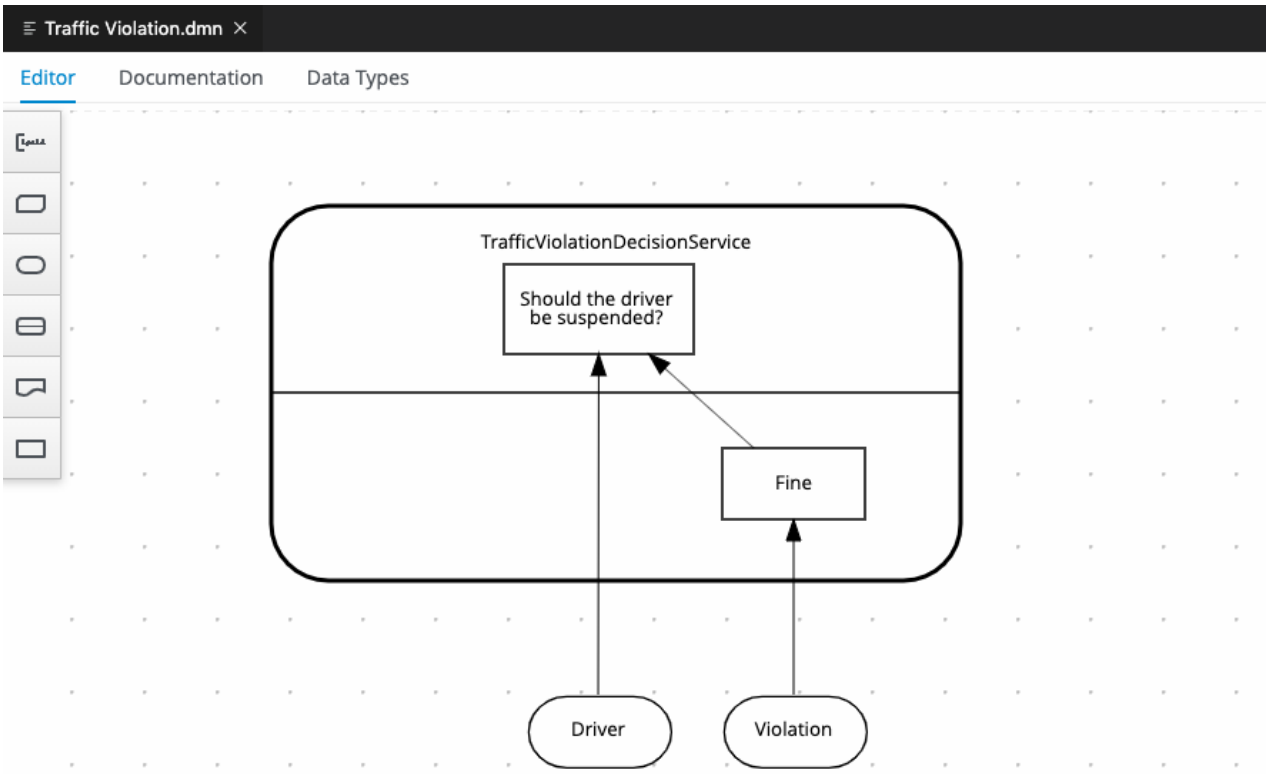
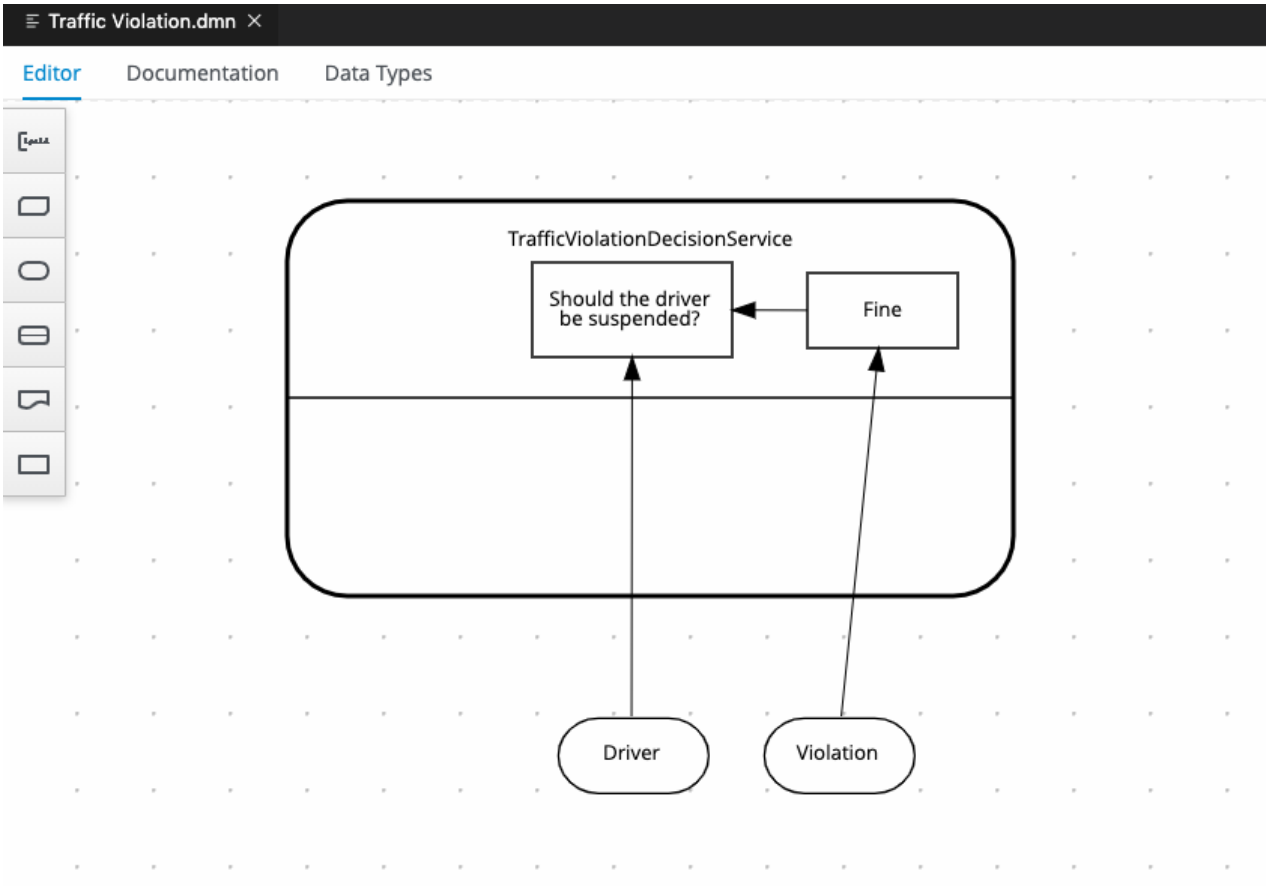


图 21.2. 带有多输出决定 的流量ViolationDecisionService 决策服务示例



REST 端点示例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic违反/TrafficViolationDecisionService

使用输入数据的 POST 请求正文示例

```
{
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 请求示例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

单输出决策(JSON)的响应示例

```
"No"
```

多输出决策(JSON)的响应示例

```
{
  "Violation": {
    "Type": "speed",
    "Speed Limit": 100,
    "Actual Speed": 120
  },
  "Driver": {
    "Points": 2
  },
  "Fine": {
    "Points": 3,
    "Amount": 500
  },
  "Should the driver be suspended?": "No"
}
```

在指定的容器中评估指定的 DMN 模型并返回 DMNResult 响应

POST /server/containers/{containerId}/dmn/models/{modelname}/dmnresult

REST 端点示例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic违反/dmnresult

使用输入数据的 POST 请求正文示例

```
{
```

```
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 请求示例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/dmnresult -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

响应示例(JSON)

```
{
  "namespace": "https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF",
  "modelName": "Traffic Violation",
  "dmnContext": {
    "Violation": {
      "Type": "speed",
      "Speed Limit": 100,
      "Actual Speed": 120,
      "Code": null,
      "Date": null
    },
    "Driver": {
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Fine": {
      "Points": 3,
      "Amount": 500
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
    {
      "decisionId": "_4055D956-1C47-479C-B3F4-BAEB61F1C929",
      "decisionName": "Fine",
      "result": {
        "Points": 3,
        "Amount": 500
      },
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
```

```
    },
    {
      "decisionId": "_8A408366-D8E9-4626-ABF3-5F69AA01F880",
      "decisionName": "Should the driver be suspended?",
      "result": "No",
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    }
  ]
}
```

在指定容器中的 DMN 模型中评估指定的决策服务并返回 DMNResult 响应

POST
/server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}/dmnresult

REST 端点示例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic违反/TrafficViolationDecisionService/dmnresult

使用输入数据的 POST 请求正文示例

```
{
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 请求示例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService/dmnresult -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

响应示例(JSON)

```
{
  "namespace": "https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF",
  "modelName": "Traffic Violation",
  "dmnContext": {
    "Violation": {
      "Type": "speed",
      "Speed Limit": 100,
      "Actual Speed": 120,
      "Code": null,
      "Date": null
    },
```

```
    "Driver": {
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
    {
      "decisionId": "_8A408366-D8E9-4626-ABF3-5F69AA01F880",
      "decisionName": "Should the driver be suspended?",
      "result": "No",
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    }
  ]
}
```

第 22 章 KIE SERVER JAVA 客户端 API 用于 KIE 容器和业务资产

Red Hat Process Automation Manager 提供了一个 KIE Server Java 客户端 API，可让您使用 Java 客户端应用程序的 REST 协议连接到 KIE Server。您可以使用 KIE Server Java 客户端 API 作为 KIE 服务器 REST API 的替代方案，与 Red Hat Process Automation Manager 中的 KIE 容器和商业资产（如业务规则、流程和解决器）进行交互。通过此 API 支持，您可以更有效地维护红帽流程自动化管理器资源，并优化您的与红帽流程自动化管理器的集成和开发。

使用 KIE Server Java 客户端 API，您可以执行 KIE Server REST API 支持的以下操作：

- 部署或分离 KIE 容器
- 检索和更新 KIE 容器信息
- 返回 KIE 服务器状态和基本信息
- 检索和更新业务资产信息
- 执行业务资产（如规则和流程）

KIE Server Java 客户端 API 请求需要以下组件：

身份验证

KIE 服务器 Java 客户端 API 需要对用户角色 **kie-server** 的 HTTP 基本身份验证。要查看为您的 Red Hat Process Automation Manager 分发配置的用户角色，请导航到 **~/$SERVER_HOME/standalone/configuration/application-roles.properties** 和 **~/application-users.properties**。
要添加具有 **kie-server** 角色的用户，请导航到 **~/$SERVER_HOME/bin**，再运行以下命令：

```
$ ./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'])"
```

有关用户角色和 Red Hat Process Automation Manager 安装选项的更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装*。

项目依赖项

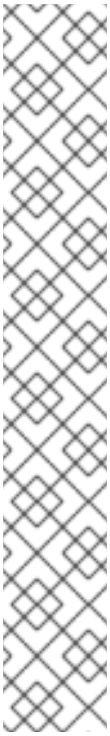
KIE Server Java 客户端 API 需要对 Java 项目的相关类路径以下依赖项：

```
<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- For runtime commands -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <scope>runtime</scope>
  <version>${rhpam.version}</version>
</dependency>
```

```
<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

Red Hat Process Automation Manager 依赖项的 **<version>** 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（如 7.59.0.Final-redhat-00006）。



注意

考虑将 Red Hat Business Automation Manager (BOM)依赖项添加到项目的 **pom.xml** 文件中，而不是为单独的依赖项指定 Red Hat Process Automation Manager < **version** >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确传输依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

有关 Red Hat Business Automation BOM 的更多信息，请参阅 RHPAM 产品和 maven 库版本之间的映射是什么？

客户端请求配置

使用 KIE Server Java 客户端 API 的所有 Java 客户端请求必须至少定义以下服务器通信组件：

- **kie-server** 用户的凭证
- KIE 服务器位置，如 **http://localhost:8080/kie-server/services/rest/server**
- API 请求和响应的 Marshalling 格式(JSON、JAXB 或 XSTREAM)
- **KieServicesConfiguration** 对象和 **KieServicesClient** 对象，它充当使用 Java 客户端 API 启动服务器通信的入口点
- 一个 **KieServicesFactory** 对象，用于定义 REST 协议和用户访问
- 使用的任何其他客户端服务，如 **RuleServicesClient**、**ProcessServicesClient** 或 **QueryServicesClient**

以下是带有以下组件的基本和高级客户端配置示例：

基本客户端配置示例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
```

```
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;

public class MyConfigurationObject {

  private static final String URL = "http://localhost:8080/kie-server/services/rest/server";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

  private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

  private static KieServicesConfiguration conf;
  private static KieServicesClient kieServicesClient;

  public static void initialize() {
    conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);

    //If you use custom classes, such as Obj.class, add them to the configuration.
    Set<Class<?>> extraClassList = new HashSet<Class<?>>();
    extraClassList.add(Obj.class);
    conf.addExtraClasses(extraClassList);

    conf.setMarshallingFormat(FORMAT);
    kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
  }
}
```

带有其他客户端服务的高级客户端配置示例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.CaseServicesClient;
import org.kie.server.client.DMNServicesClient;
import org.kie.server.client.DocumentServicesClient;
import org.kie.server.client.JobServicesClient;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.ProcessServicesClient;
import org.kie.server.client.QueryServicesClient;
import org.kie.server.client.RuleServicesClient;
import org.kie.server.client.SolverServicesClient;
import org.kie.server.client.UIServicesClient;
import org.kie.server.client.UserTaskServicesClient;
import org.kie.server.api.model.instance.ProcessInstance;
import org.kie.server.api.model.KieContainerResource;
import org.kie.server.api.model.ReleaseId;

public class MyAdvancedConfigurationObject {

    // REST API base URL, credentials, and marshalling format
    private static final String URL = "http://localhost:8080/kie-server/services/rest/server";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";;

    private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;
```

```
    private static KieServicesConfiguration conf;

    // KIE client for common operations
    private static KieServicesClient kieServicesClient;

    // Rules client
    private static RuleServicesClient ruleClient;

    // Process automation clients
    private static CaseServicesClient caseClient;
    private static DocumentServicesClient documentClient;
    private static JobServicesClient jobClient;
    private static ProcessServicesClient processClient;
    private static QueryServicesClient queryClient;
    private static UIServicesClient uiClient;
    private static UserTaskServicesClient userTaskClient;

    // DMN client
    private static DMNServicesClient dmnClient;

    // Planning client
    private static SolverServicesClient solverClient;

    public static void main(String[] args) {
        initializeKieServerClient();
        initializeDroolsServiceClients();
        initializeJbpmServiceClients();
        initializeSolverServiceClients();
    }

    public static void initializeKieServerClient() {
        conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);
        conf.setMarshallingFormat(FORMAT);
        kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
    }

    public static void initializeDroolsServiceClients() {
        ruleClient = kieServicesClient.getServicesClient(RuleServicesClient.class);
        dmnClient = kieServicesClient.getServicesClient(DMNServicesClient.class);
    }

    public static void initializeJbpmServiceClients() {
        caseClient = kieServicesClient.getServicesClient(CaseServicesClient.class);
        documentClient = kieServicesClient.getServicesClient(DocumentServicesClient.class);
        jobClient = kieServicesClient.getServicesClient(JobServicesClient.class);
        processClient = kieServicesClient.getServicesClient(ProcessServicesClient.class);
        queryClient = kieServicesClient.getServicesClient(QueryServicesClient.class);
        uiClient = kieServicesClient.getServicesClient(UIServicesClient.class);
        userTaskClient = kieServicesClient.getServicesClient(UserTaskServicesClient.class);
    }

    public static void initializeSolverServiceClients() {
        solverClient = kieServicesClient.getServicesClient(SolverServicesClient.class);
    }
}
```

22.1. 使用 KIE SERVER JAVA 客户端 API 发送请求

KIE 服务器 Java 客户端 API 可让您使用 Java 客户端应用中的 REST 协议连接到 KIE 服务器。您可以使用 KIE Server Java 客户端 API 作为 KIE 服务器 REST API 的替代方案，与 Red Hat Process Automation Manager 中的 KIE 容器和商业资产（如业务规则、流程和解决器）进行交互。

先决条件

- KIE 服务器已安装并运行。
- 您有 **kie-server** 用户角色对 KIE 服务器的访问权限。
- 您有一个带有 Red Hat Process Automation Manager 资源的 Java 项目。

流程

1. 在客户端应用程序中，确保以下依赖项已添加到 Java 项目的相关类路径中：

```
<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- For runtime commands -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <scope>runtime</scope>
  <version>${rhpam.version}</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

2. 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution**，并进入 **~/rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/server/client** 来访问 KIE Server Java 客户端。

3. 在 **~/kie/server/client** 文件夹中，识别您要发送的请求的相关 Java 客户端，如 **KieServicesClient** 访问 KIE 容器和其他资产的客户端服务。

4. 在您的客户端应用中，为 API 请求创建一个 **.java** 类。类必须包含必要的导入、KIE 服务器位置和用户凭据、**KieServicesClient** 对象和要执行的客户端方法，如 **createContainer** 和来自 **KieServicesClient** 客户端的 **disposeContainer**。根据您的用例调整任何配置详情。

   创建和处理容器

```
import org.kie.server.api.marshalling.MarshallingFormat;
```

```
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.api.model.KieContainerResource;
import org.kie.server.api.model.ServiceResponse;

public class MyConfigurationObject {

  private static final String URL = "http://localhost:8080/kie-server/services/rest/server";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

  private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

  private static KieServicesConfiguration conf;
  private static KieServicesClient kieServicesClient;

  public static void initialize() {
    conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);

  public void disposeAndCreateContainer() {
      System.out.println("== Disposing and creating containers ==");

      // Retrieve list of KIE containers
      List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
      if (kieContainers.size() == 0) {
          System.out.println("No containers available...");
          return;
      }

      // Dispose KIE container
      KieContainerResource container = kieContainers.get(0);
      String containerId = container.getContainerId();
      ServiceResponse<Void> responseDispose =
kieServicesClient.disposeContainer(containerId);
      if (responseDispose.getType() == ResponseType.FAILURE) {
          System.out.println("Error disposing " + containerId + ". Message: ");
          System.out.println(responseDispose.getMsg());
          return;
      }
      System.out.println("Success Disposing container " + containerId);
      System.out.println("Trying to recreate the container...");

      // Re-create KIE container
      ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
      if(createResponse.getType() == ResponseType.FAILURE) {
          System.out.println("Error creating " + containerId + ". Message: ");
          System.out.println(responseDispose.getMsg());
          return;
      }
      System.out.println("Container recreated with success!");
      }
  }
}
```

您可以使用 **org.kie.server.api.model.ServiceResponse<T>** 对象来定义服务响应，其中 **T** 代表返回的响应类型。**ServiceResponse** 对象具有以下属性：

- **字符串 *消息*** ：返回响应消息
- **ResponseType *type*** ：返回 **SUCCESS** 或 **FAILURE**
- **T *结果*** ：返回请求的对象

在本例中，当您分离容器时，**ServiceResponse** 会返回 **Void** 响应。在创建容器时，**ServiceResponse** 会返回 **KieContainerResource** 对象。

注意

客户端与集群环境中的特定 KIE 服务器容器之间的对话是通过独特的 **对话ID 来保护**。使用 **X-KIE-ConversationId** REST 标头传输 **对话 ID**。如果更新容器，请取消设置之前的 talk **ID**。使用 **KieServiesClient.completeConversation （）** 来取消设置 Java API 的 talk **ID**。

5. 从项目目录运行配置的 **.java** 类来执行请求，并查看 KIE 服务器响应。
如果您启用了调试日志，KIE 服务器会根据您配置的 marshalling 格式（如 JSON）响应详细的响应。

新 KIE 容器的服务器响应示例(log)：

```
10:23:35.194 [main] INFO  o.k.s.a.m.MarshallerFactory - Marshaller extensions init
10:23:35.396 [main] DEBUG o.k.s.client.balancer.LoadBalancer - Load balancer
RoundRobinBalancerStrategy{availableEndpoints=[http://localhost:8080/kie-
server/services/rest/server]} selected url 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.398 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to send GET
request to 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.440 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to deserialize
content:
 '{
  "type" : "SUCCESS",
  "msg" : "Kie Server info",
  "result" : {
    "kie-server-info" : {
      "id" : "default-kieserver",
      "version" : "7.11.0.Final-redhat-00003",
      "name" : "default-kieserver",
      "location" : "http://localhost:8080/kie-server/services/rest/server",
      "capabilities" : [ "KieServer", "BRM", "BPM", "CaseMgmt", "BPM-UI", "BRP", "DMN",
"Swagger" ],
      "messages" : [ {
        "severity" : "INFO",
        "timestamp" : {
  "java.util.Date" : 1540814906533
},
        "content" : [ "Server KieServerInfo{serverId='default-kieserver', version='7.11.0.Final-
redhat-00003', name='default-kieserver', location='http://localhost:8080/kie-
server/services/rest/server', capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP,
DMN, Swagger], messages=null}started successfully at Mon Oct 29 08:08:26 EDT 2018" ]
      } ]
    }
```

```
  }
}'
 into type: 'class org.kie.server.api.model.ServiceResponse'
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - KieServicesClient
connected to: default-kieserver version 7.11.0.Final-redhat-00003
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Supported capabilities by
the server: [KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger]
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability KieServer
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - No builder found for
'KieServer' capability
10:23:35.654 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BRM
10:23:35.654 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.DroolsServicesClientBuilder@6b927fb' for capability 'BRM'
10:23:35.655 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.RuleServicesClient=org.kie.server.client.impl.RuleServicesClientImpl@4a94
ee4}
10:23:35.655 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BPM
10:23:35.656 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.JBPMServicesClientBuilder@4cc451f2' for capability 'BPM'
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.JobServicesClient=org.kie.server.client.impl.JobServicesClientImpl@1189d
52, interface
org.kie.server.client.admin.ProcessAdminServicesClient=org.kie.server.client.admin.impl.Proces
sAdminServicesClientImpl@36bc55de, interface
org.kie.server.client.DocumentServicesClient=org.kie.server.client.impl.DocumentServicesClient
mpl@564fabc8, interface
org.kie.server.client.admin.UserTaskAdminServicesClient=org.kie.server.client.admin.impl.User
TaskAdminServicesClientImpl@16d04d3d, interface
org.kie.server.client.QueryServicesClient=org.kie.server.client.impl.QueryServicesClientImpl@4
9ec71f8, interface
org.kie.server.client.ProcessServicesClient=org.kie.server.client.impl.ProcessServicesClientImp
@1d2adfbe, interface
org.kie.server.client.UserTaskServicesClient=org.kie.server.client.impl.UserTaskServicesClientI
mpl@36902638}
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability CaseMgmt
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.CaseServicesClientBuilder@223d2c72' for capability 'CaseMgmt'
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.admin.CaseAdminServicesClient=org.kie.server.client.admin.impl.CaseAdm
nServicesClientImpl@2b662a77, interface
org.kie.server.client.CaseServicesClient=org.kie.server.client.impl.CaseServicesClientImpl@7f0
eb4b4}
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BPM-UI
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.JBPMUIServicesClientBuilder@5c33f1a9' for capability 'BPM-UI'
10:23:35.677 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.UIServicesClient=org.kie.server.client.impl.UIServicesClientImpl@223191a
```

```
}
10:23:35.678 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BRP
10:23:35.678 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.OptaplannerServicesClientBuilder@49139829' for capability
'BRP'
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.SolverServicesClient=org.kie.server.client.impl.SolverServicesClientImpl@7
7fbd92c}
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability DMN
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.DMNServicesClientBuilder@67c27493' for capability 'DMN'
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.DMNServicesClient=org.kie.server.client.impl.DMNServicesClientImpl@35e
2d654}
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability Swagger
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - No builder found for
'Swagger' capability
10:23:35.681 [main] DEBUG o.k.s.client.balancer.LoadBalancer - Load balancer
RoundRobinBalancerStrategy{availableEndpoints=[http://localhost:8080/kie-
server/services/rest/server]} selected url 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.701 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to send PUT
request to 'http://localhost:8080/kie-server/services/rest/server/containers/employee-
rostering3' with payload '{
  "container-id" : null,
  "release-id" : {
    "group-id" : "employeerostering",
    "artifact-id" : "employeerostering",
    "version" : "1.0.0-SNAPSHOT"
  },
  "resolved-release-id" : null,
  "status" : null,
  "scanner" : null,
  "config-items" : [ ],
  "messages" : [ ],
  "container-alias" : null
}'
10:23:38.071 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to deserialize
content:
 '{
  "type" : "SUCCESS",
  "msg" : "Container employee-rostering3 successfully deployed with module
employeerostering:employeerostering:1.0.0-SNAPSHOT.",
  "result" : {
    "kie-container" : {
      "container-id" : "employee-rostering3",
      "release-id" : {
        "group-id" : "employeerostering",
        "artifact-id" : "employeerostering",
        "version" : "1.0.0-SNAPSHOT"
      },
      "resolved-release-id" : {
```

```
        "group-id" : "employeerostering",
        "artifact-id" : "employeerostering",
        "version" : "1.0.0-SNAPSHOT"
      },
      "status" : "STARTED",
      "scanner" : {
        "status" : "DISPOSED",
        "poll-interval" : null
      },
      "config-items" : [ ],
      "messages" : [ {
        "severity" : "INFO",
        "timestamp" : {
  "java.util.Date" : 1540909418069
},
        "content" : [ "Container employee-rostering3 successfully created with module
employeerostering:employeerostering:1.0.0-SNAPSHOT." ]
      } ],
      "container-alias" : null
    }
  }
}'
 into type: 'class org.kie.server.api.model.ServiceResponse'
```

如果您遇到请求错误，请检查返回的错误消息并相应地调整 Java 配置。

22.2. 支持的 KIE 服务器 JAVA 客户端

以下是 Red Hat Process Automation Manager 发行版本的 **org.kie.server.client** 软件包中提供的一些 Java 客户端服务。您可以使用这些服务与 KIE Server REST API 中的相关资源交互。

- **KieServicesClient** ：用作其他 KIE 服务器 Java 客户端的入口点，用于与 KIE 容器交互
- **JobServicesClient** ：用于调度、取消、重新队列和获取作业请求
- **RuleServicesClient** ：用于将命令发送到服务器来执行与规则相关的操作，如执行规则或将对象插入到 KIE 会话中
- **SolverServicesClient** ：用于执行所有 Red Hat build of OptaPlanner 操作，如获取解决方案状态和最佳解决方案，或处理解决者
- **ProcessServicesClient** ：用于启动、信号和中止进程或工作项目
- **QueryServicesClient** ：用于查询进程、进程节点和进程变量
- **UserTaskServicesClient** ：用于执行所有 user-task 操作，如启动、声明或取消任务，以及通过指定字段查询任务，如按用户或进程实例 ID
- **UIServicesClient** ：用于获取表单(XML 或 JSON)和进程镜像(SVG)的字符串表示。
- **ProcessAdminServicesClient** ：为处理实例的操作提供接口（可以在 **~/org/kie/server/client/admin**中找到）
- **UserTaskAdminServicesClient** ：为带有用户任务的操作提供一个接口（可以在 **~/org/kie/server/client/admin**中找到）

getServicesClient 方法提供对这些客户端中的任何一个的访问：

```
RuleServicesClient rulesClient = kieServicesClient.getServicesClient(RuleServicesClient.class);
```

对于可用的 KIE Server Java 客户端的完整列表，请 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution**，再导航到 **~/rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/client/client**.

22.3. 使用 KIE SERVER JAVA 客户端 API 的请求示例

以下是 KIE Server Java 客户端 API 请求示例，用于与 KIE 服务器进行基本交互。对于可用的 KIE Server Java 客户端的完整列表，请 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution**，再导航到 **~/rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/client/client**.

列出 KIE 服务器功能

您可以使用 **org.kie.server.api.model.KieServerInfo** 对象来识别服务器功能。**KieServicesClient** 客户端需要服务器功能信息才能正确生成服务客户端。您可以在 **KieServicesConfiguration** 中全局指定功能；否则，它们会自动从 KIE Server 检索。

返回 KIE 服务器功能的请求示例

```
public void listCapabilities() {

  KieServerInfo serverInfo = kieServicesClient.getServerInfo().getResult();
  System.out.print("Server capabilities:");

  for (String capability : serverInfo.getCapabilities()) {
    System.out.print(" " + capability);
  }

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

在 KIE 服务器中列出 KIE 容器

KIE 容器由 **org.kie.server.api.model.KieContainerResource** 对象表示。资源列表由 **org.kie.server.api.model.KieContainerResourceList** 对象表示。

从 KIE 服务器返回 KIE 容器的请求示例

```
public void listContainers() {
    KieContainerResourceList containersList = kieServicesClient.listContainers().getResult();
    List<KieContainerResource> kieContainers = containersList.getContainers();
    System.out.println("Available containers: ");
    for (KieContainerResource container : kieContainers) {
        System.out.println("\t" + container.getContainerId() + " (" + container.getReleaseId() + ")");
    }
}
```

您可以使用 **org.kie.server.api.model.KieContainerResourceFilter** 类的实例过滤 KIE 容器结果，该类传递给 **org.kie.server.client.KieServicesClient.listContainers （）** 方法。

根据发行版本 ID 和状态返回 KIE 容器的请求示例

```
public void listContainersWithFilter() {

    // Filter containers by releaseId "org.example:container:1.0.0.Final" and status FAILED
    KieContainerResourceFilter filter = new KieContainerResourceFilter.Builder()
            .releaseId("org.example", "container", "1.0.0.Final")
            .status(KieContainerStatus.FAILED)
            .build();

    // Using previously created KieServicesClient
    KieContainerResourceList containersList = kieServicesClient.listContainers(filter).getResult();
    List<KieContainerResource> kieContainers = containersList.getContainers();

    System.out.println("Available containers: ");

    for (KieContainerResource container : kieContainers) {
        System.out.println("\t" + container.getContainerId() + " (" + container.getReleaseId() + ")");
    }
}
```

在 KIE 服务器中创建和处理 KIE 容器

您可以使用 **KieServicesClient** 客户端中的 **createContainer** 和 **disposeContainer** 方法来分离并创建 KIE 容器。在本例中，当您分离容器时，**ServiceResponse** 会返回 **Void** 响应。在创建容器时，**ServiceResponse** 会返回 **KieContainerResource** 对象。

取消和重新创建 KIE 容器的请求示例

```
public void disposeAndCreateContainer() {
    System.out.println("== Disposing and creating containers ==");

    // Retrieve list of KIE containers
    List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
    if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
    }

    // Dispose KIE container
    KieContainerResource container = kieContainers.get(0);
    String containerId = container.getContainerId();
    ServiceResponse<Void> responseDispose = kieServicesClient.disposeContainer(containerId);
    if (responseDispose.getType() == ResponseType.FAILURE) {
        System.out.println("Error disposing " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Success Disposing container " + containerId);
    System.out.println("Trying to recreate the container...");

    // Re-create KIE container
    ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
```

```
    if(createResponse.getType() == ResponseType.FAILURE) {
        System.out.println("Error creating " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Container recreated with success!");
}
```

在 KIE 服务器中执行运行时命令

Red Hat Process Automation Manager 支持向 KIE Server 发送与资产相关的操作的运行时命令，如在 KIE 会话中插入或调整对象或触发所有规则。支持的运行时命令的完整列表位于 Red Hat Process Automation Manager 实例的 **org.drools.core.command.runtime** 软件包中。

您可以使用 **org.kie.api.command.KieCommands** 类来插入命令，并使用 **org.kie.api.KieServices.get （）.getCommands （）** 来实例化 **KieCommands** 类。如果要添加多个命令，请使用 **BatchExecutionCommand** 打包程序。

插入对象并触发所有规则的请求示例

```
import org.kie.api.command.Command;
import org.kie.api.command.KieCommands;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.RuleServicesClient;
import org.kie.server.client.KieServicesClient;
import org.kie.api.KieServices;

import java.util.Arrays;

...

public void executeCommands() {

  String containerId = "hello";
  System.out.println("== Sending commands to the server ==");
  RuleServicesClient rulesClient = kieServicesClient.getServicesClient(RuleServicesClient.class);
  KieCommands commandsFactory = KieServices.Factory.get().getCommands();

  Command<?> insert = commandsFactory.newInsert("Some String OBJ");
  Command<?> fireAllRules = commandsFactory.newFireAllRules();
  Command<?> batchCommand = commandsFactory.newBatchExecution(Arrays.asList(insert,
fireAllRules));

  ServiceResponse<String> executeResponse = rulesClient.executeCommands(containerId,
batchCommand);

  if(executeResponse.getType() == ResponseType.SUCCESS) {
    System.out.println("Commands executed with success! Response: ");
    System.out.println(executeResponse.getResult());
  } else {
    System.out.println("Error executing rules. Message: ");
    System.out.println(executeResponse.getMsg());
  }
}
```

注意

客户端与集群环境中的特定 KIE 服务器容器之间的对话是通过独特的 **对话ID 来保护**。使用 **X-KIE-ConversationId** REST 标头传输 **对话 ID**。如果更新容器，请取消设置之前的 talk **ID**。使用 **KieServiesClient.completeConversation （）** 来取消设置 Java API 的 talk **ID**。

列出 KIE 容器中的可用进程

您可以使用 **QueryServicesClient** 客户端列出可用的进程定义。**QueryServicesClient** 方法使用分页，因此除了您所做的查询外，您必须提供当前页面和每个页面的结果数量。在本例中，查询从页 **0** 开始，并列出前 **1000** 结果。

KIE 服务器中列出业务流程的请求示例

```
public void listProcesses() {
    System.out.println("== Listing Business Processes ==");
    QueryServicesClient queryClient =
kieServicesClient.getServicesClient(QueryServicesClient.class);
    List<ProcessDefinition> findProcessesByContainerId =
queryClient.findProcessesByContainerId("rewards", 0, 1000);
    for (ProcessDefinition def : findProcessesByContainerId) {
        System.out.println(def.getName() + " - " + def.getId() + " v" + def.getVersion());
    }
}
```

在 KIE 容器中启动业务流程

您可以使用 **ProcessServicesClient** 客户端来启动业务流程。使用 **addExtraClasses （）** 方法，确保进程需要的任何自定义类都添加到 **KieServicesConfiguration** 对象中。

开始业务流程的请求示例

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.ProcessServicesClient;
...

public static void startProcess() {

  //Client configuration setup
  KieServicesConfiguration config = KieServicesFactory.newRestConfiguration(SERVER_URL,
LOGIN, PASSWORD);

  //Add custom classes, such as Obj.class, to the configuration
```

```
  Set<Class<?>> extraClassList = new HashSet<Class<?>>();
  extraClassList.add(Obj.class);
  config.addExtraClasses(extraClassList);
  config.setMarshallingFormat(MarshallingFormat.JSON);

  // ProcessServicesClient setup
  KieServicesClient client = KieServicesFactory.newKieServicesClient(config);
  ProcessServicesClient processServicesClient =
client.getServicesClient(ProcessServicesClient.class);

  // Create an instance of the custom class
  Obj obj = new Obj();
  obj.setOk("ok");

  Map<String, Object> variables = new HashMap<String, Object>();
  variables.put("test", obj);

  // Start the process with custom class
  processServicesClient.startProcess(CONTAINER, processId, variables);
}
```

运行自定义查询

您可以使用 **QueryServicesClient** 客户端的 **QueryDefinition** 对象在 KIE 服务器中注册和执行自定义查询。

在 KIE Server 中注册和执行自定义查询的请求示例

```
// Client setup
KieServicesConfiguration conf = KieServicesFactory.newRestConfiguration(SERVER_URL,
LOGIN, PASSWORD);
KieServicesClient client = KieServicesFactory.newKieServicesClient(conf);

// Get the QueryServicesClient
QueryServicesClient queryClient = client.getServicesClient(QueryServicesClient.class);

// Build the query
QueryDefinition queryDefinition = QueryDefinition.builder().name(QUERY_NAME)
        .expression("select * from Task t")
        .source("java:jboss/datasources/ExampleDS")
        .target("TASK").build();

// Specify that two queries cannot have the same name
queryClient.unregisterQuery(QUERY_NAME);

// Register the query
queryClient.registerQuery(queryDefinition);

// Execute the query with parameters: query name, mapping type (to map the fields to an object),
page number, page size, and return type
List<TaskInstance> query = queryClient.query(QUERY_NAME,
QueryServicesClient.QUERY_MAP_TASK, 0, 100, TaskInstance.class);

// Read the result
```

```
for (TaskInstance taskInstance : query) {
    System.out.println(taskInstance);
}
```

在本例中，**目标** 指示查询服务应用默认过滤器。或者，您可以手动设置过滤器参数。**Target** 类支持以下值：

```
public enum Target {
    PROCESS,
    TASK,
    BA_TASK,
    PO_TASK,
    JOBS,
    CUSTOM;
}
```

第 23 章 红帽流程自动化管理器中的 KIE 服务器和 KIE 容器命令

Red Hat Process Automation Manager 支持针对与服务器相关的或容器相关操作（如检索服务器信息或删除容器）发送到 KIE Server 的服务器命令。支持的 KIE 服务器配置命令的完整列表位于 Red Hat Process Automation Manager 实例的 **org.kie.server.api.commands** 软件包中。

在 KIE Server REST API 中，您可以使用 **org.kie.server.api.commands** 命令作为 **POST** 请求到 **http://SERVER:PORT/kie-server/services/rest/server/config** 的请求正文。有关使用 KIE 服务器 REST API 的更多信息，请参阅 *第 21 章 KIE 服务器 REST API 用于 KIE 容器和业务资产*。

在 KIE Server Java 客户端 API 中，您可以使用父 **KieServicesClient** Java 客户端中的对应方法作为 Java 应用中的嵌入式 API 请求。所有 KIE 服务器命令都由 Java 客户端 API 中提供的方法执行，因此您不需要在 Java 应用中嵌入实际的 KIE Server 命令。有关使用 KIE Server Java 客户端 API 的详情，请参考 *第 22 章 KIE Server Java 客户端 API 用于 KIE 容器和业务资产*。

23.1. KIE 服务器和 KIE 容器命令示例

以下是您可以在 KIE Server 中使用 KIE Server REST API 或 Java 客户端 API 的 KIE Server 命令示例，用于 KIE 服务器中的与服务器相关的或容器相关操作：

- **GetServerInfoCommand**
- **GetServerStateCommand**
- **CreateContainerCommand**
- **GetContainerInfoCommand**
- **ListContainersCommand**
- **CallContainerCommand**
- **DisposeContainerCommand**
- **GetScannerInfoCommand**
- **UpdateScannerCommand**
- **UpdateReleaseIdCommand**

有关支持的 KIE 服务器配置和管理命令的完整列表，请查看 Red Hat Process Automation Manager 实例中的 **org.kie.server.api.commands** 软件包。

您可以单独运行 KIE Server 命令，也可以作为批处理 REST API 请求或批处理 Java API 请求一起运行：

批处理 REST API 请求以创建、调用和分离 KIE 容器(JSON)

```
{
  "commands": [
    {
      "create-container": {
        "container": {
          "status": "STARTED",
          "container-id": "command-script-container",
          "release-id": {
            "version": "1.0",
```

```
            "group-id": "com.redhat",
            "artifact-id": "Project1"
          }
        }
      }
    },
    {
      "call-container": {
        "payload": "{\n  \"commands\" : [ {\n    \"fire-all-rules\" : {\n      \"max\" : -1,\n      \"out-identifier\" :
null\n    }\n  } ]\n}",
        "container-id": "command-script-container"
      }
    },
    {
      "dispose-container": {
        "container-id": "command-script-container"
      }
    }
  ]
}
```

批处理 Java API 请求以检索、分配和重新创建 KIE 容器

```
public void disposeAndCreateContainer() {
    System.out.println("== Disposing and creating containers ==");

    // Retrieve list of KIE containers
    List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
    if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
    }

    // Dispose KIE container
    KieContainerResource container = kieContainers.get(0);
    String containerId = container.getContainerId();
    ServiceResponse<Void> responseDispose = kieServicesClient.disposeContainer(containerId);
    if (responseDispose.getType() == ResponseType.FAILURE) {
        System.out.println("Error disposing " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Success Disposing container " + containerId);
    System.out.println("Trying to recreate the container...");

    // Re-create KIE container
    ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
    if(createResponse.getType() == ResponseType.FAILURE) {
        System.out.println("Error creating " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Container recreated with success!");
}
```

本节中的每个命令都包括 KIE Server REST API 的 REST 请求正文示例(JSON)，以及来自 KIE Server Java 客户端 API 的 **KieServicesClient** Java 客户端的嵌入式方法示例。

GetServerInfoCommand

返回有关此 KIE 服务器实例的信息。

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "get-server-info" : { }
  } ]
}
```

Java 客户端方法示例

```
KieServerInfo serverInfo = kieServicesClient.getServerInfo();
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Kie Server info",
      "result": {
        "kie-server-info": {
          "id": "default-kieserver",
          "version": "7.11.0.Final-redhat-00001",
          "name": "default-kieserver",
          "location": "http://localhost:8080/kie-server/services/rest/server",
          "capabilities": [
            "KieServer",
            "BRM",
            "BPM",
            "CaseMgmt",
            "BPM-UI",
            "BRP",
            "DMN",
            "Swagger"
          ],
          "messages": [
            {
              "severity": "INFO",
              "timestamp": {
                "java.util.Date": 1538502533321
              },
              "content": [
                "Server KieServerInfo{serverId='default-kieserver', version='7.11.0.Final-redhat-00001',
name='default-kieserver', location='http://localhost:8080/kie-server/services/rest/server',
capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger],
messages=null}started successfully at Tue Oct 02 13:48:53 EDT 2018"
              ]
```

```
          }
        ]
      }
    }
  }
 ]
}
```

GetServerStateCommand

返回有关此 KIE 服务器实例的当前状态和配置的信息。

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "get-server-state" : { }
  } ]
}
```

Java 客户端方法示例

```
KieServerStateInfo serverStateInfo = kieServicesClient.getServerState();
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Successfully loaded server state for server id default-kieserver",
      "result": {
        "kie-server-state-info": {
          "controller": [
            "http://localhost:8080/business-central/rest/controller"
          ],
          "config": {
            "config-items": [
              {
                "itemName": "org.kie.server.location",
                "itemValue": "http://localhost:8080/kie-server/services/rest/server",
                "itemType": "java.lang.String"
              },
              {
                "itemName": "org.kie.server.controller.user",
                "itemValue": "controllerUser",
                "itemType": "java.lang.String"
              },
              {
                "itemName": "org.kie.server.controller",
                "itemValue": "http://localhost:8080/business-central/rest/controller",
                "itemType": "java.lang.String"
              }
            ]
          },
```

```
        "containers": [
         {
           "container-id": "employee-rostering",
           "release-id": {
             "group-id": "employeerostering",
             "artifact-id": "employeerostering",
             "version": "1.0.0-SNAPSHOT"
           },
           "resolved-release-id": null,
           "status": "STARTED",
           "scanner": {
             "status": "STOPPED",
             "poll-interval": null
           },
           "config-items": [
             {
               "itemName": "KBase",
               "itemValue": "",
               "itemType": "BPM"
             },
             {
               "itemName": "KSession",
               "itemValue": "",
               "itemType": "BPM"
             },
             {
               "itemName": "MergeMode",
               "itemValue": "MERGE_COLLECTIONS",
               "itemType": "BPM"
             },
             {
               "itemName": "RuntimeStrategy",
               "itemValue": "SINGLETON",
               "itemType": "BPM"
             }
           ],
           "messages": [],
           "container-alias": "employeerostering"
         }
        ]
       }
     }
   }
  ]
}
```


CreateContainerCommand

在 KIE 服务器中创建 KIE 容器。

表 23.1. 命令属性

名称	描述	要求

名称	描述	要求
container	包含 **container-id**、**release-id** 数据（组 ID、工件 ID、版本）、**状态**，以及新 KIE 容器的任何其他组件	必填

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "create-container" : {
      "container" : {
        "status" : null,
        "messages" : [ ],
        "container-id" : "command-script-container",
        "release-id" : {
          "version" : "1.0",
          "group-id" : "com.redhat",
          "artifact-id" : "Project1"
        },
        "config-items" : [ ]
      }
    }
  } ]
}
```

Java 客户端方法示例

```
ServiceResponse<KieContainerResource> response =
kieServicesClient.createContainer("command-script-container", resource);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully deployed with module
com.redhat:Project1:1.0.",
      "result": {
        "kie-container": {
          "container-id": "command-script-container",
          "release-id": {
            "version" : "1.0",
            "group-id" : "com.redhat",
            "artifact-id" : "Project1"
          },
          "resolved-release-id": {
            "version" : "1.0",
            "group-id" : "com.redhat",
            "artifact-id" : "Project1"
          },
          "status": "STARTED",
```

```
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [],
          "messages": [
            {
              "severity": "INFO",
              "timestamp": {
                "java.util.Date": 1538762455510
              },
              "content": [
                "Container command-script-container successfully created with module
com.redhat:Project1:1.0."
              ]
            }
          ],
          "container-alias": null
        }
      }
    }
  ]
}
```

GetContainerInfoCommand

返回 KIE 服务器中指定的 KIE 容器的信息。

表 23.2. 命令属性

名称	描述	要求
container-id	KIE 容器的 ID	必填

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "get-container-info" : {
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java 客户端方法示例

```
ServiceResponse<KieContainerResource> response =
kieServicesClient.getContainerInfo("command-script-container");
```

服务器响应示例(JSON)

```
{
  "response": [
```

```
    {
      "type": "SUCCESS",
      "msg": "Info for container command-script-container",
      "result": {
        "kie-container": {
          "container-id": "command-script-container",
          "release-id": {
            "group-id": "com.redhat",
            "artifact-id": "Project1",
            "version": "1.0"
          },
          "resolved-release-id": {
            "group-id": "com.redhat",
            "artifact-id": "Project1",
            "version": "1.0"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [

          ],
          "container-alias": null
        }
      }
    }
  ]
}
```


ListContainersCommand

返回此 KIE 服务器实例中创建的 KIE 容器列表。

表 23.3. 命令属性

名称	描述	要求
kie-container-filter	包含 **release-id-filter**、**container-status-filter** 以及您要过滤结果的任何其他 KIE 容器属性的可选映射	选填

REST 请求正文(JSON)示例


```
{
  "commands" : [ {
    "list-containers" : {
      "kie-container-filter" : {
        "release-id-filter" : { },
        "container-status-filter" : {
          "accepted-status" : ["FAILED"]
        }
      }
```

```
    }
  } ]
}
```

Java 客户端方法示例

```
KieContainerResourceFilter filter = new KieContainerResourceFilter.Builder()
        .status(KieContainerStatus.FAILED)
        .build();

KieContainerResourceList containersList = kieServicesClient.listContainers(filter);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "List of created containers",
      "result": {
        "kie-containers": {
          "kie-container": [
            {
              "container-id": "command-script-container",
              "release-id": {
                "group-id": "com.redhat",
                "artifact-id": "Project1",
                "version": "1.0"
              },
              "resolved-release-id": {
                "group-id": "com.redhat",
                "artifact-id": "Project1",
                "version": "1.0"
              },
              "status": "STARTED",
              "scanner": {
                "status": "STARTED",
                "poll-interval": 5000
              },
              "config-items": [
                {
                  "itemName": "RuntimeStrategy",
                  "itemValue": "SINGLETON",
                  "itemType": "java.lang.String"
                },
                {
                  "itemName": "MergeMode",
                  "itemValue": "MERGE_COLLECTIONS",
                  "itemType": "java.lang.String"
                },
                {
                  "itemName": "KBase",
                  "itemValue": "",
                  "itemType": "java.lang.String"
                },
```

```
            {
              "itemName": "KSession",
              "itemValue": "",
              "itemType": "java.lang.String"
            }
          ],
          "messages": [
            {
              "severity": "INFO",
              "timestamp": {
                "java.util.Date": 1538504619749
              },
              "content": [
                "Container command-script-container successfully created with module
com.redhat:Project1:1.0."
              ]
            }
          ],
          "container-alias": null
        }
      ]
    }
  }
 ]
}
```


CallContainerCommand

调用 KIE 容器并执行一个或多个运行时命令。有关 Red Hat Process Automation Manager 运行时命令的详情，请参考 *第 24 章 Red Hat Process Automation Manager 中的运行时命令*。

表 23.4. 命令属性

名称	描述	要求
container-id	要调用的 KIE 容器的 ID	必填
payload	要在 KIE 容器上执行的 **BatchExecutionCommand** 打包程序中的一个或多个命令	必填

REST 请求正文(JSON)示例


```
{
  "commands" : [ {
    "call-container" : {
      "payload" : "{\n  \"lookup\" : \"defaultKieSession\",\n  \"commands\" : [ {\n    \"fire-all-rules\" : {\n
\"max\" : -1,\n      \"out-identifier\" : null\n    }\n  } ]\n}",
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java 客户端方法示例

```
List<Command<?>> commands = new ArrayList<Command<?>>();
      BatchExecutionCommand batchExecution1 =
commandsFactory.newBatchExecution(commands, "defaultKieSession");
      commands.add(commandsFactory.newFireAllRules());

      ServiceResponse<ExecutionResults> response1 =
ruleClient.executeCommandsWithResults("command-script-container", batchExecution1);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": "{\n  \"results\" : [ ],\n  \"facts\" : [ ]\n}"
    }
  ]
}
```

DisposeContainerCommand

在 KIE 服务器中分离指定的 KIE 容器。

表 23.5. 命令属性

名称	描述	要求
container-id	要分离的 KIE 容器的 ID	必填

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "dispose-container" : {
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java 客户端方法示例

```
ServiceResponse<Void> response = kieServicesClient.disposeContainer("command-script-
container");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
```

```
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully disposed.",
      "result": null
    }
  ]
}
```

GetScannerInfoCommand

如果适用，返回有关在指定 KIE 容器中用于自动更新的 KIE 扫描程序的信息。

表 23.6. 命令属性

名称	描述	要求
container-id	使用 KIE 扫描程序的 KIE 容器的 ID	必填

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "get-scanner-info" : {
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java 客户端方法示例

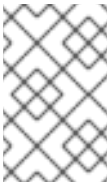
```
ServiceResponse<KieScannerResource> response =
kieServicesClient.getScannerInfo("command-script-container");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Scanner info successfully retrieved",
      "result": {
        "kie-scanner": {
          "status": "DISPOSED",
          "poll-interval": null
        }
      }
    }
  ]
}
```

UpdateScannerCommand

启动或停止 KIE 扫描程序来控制为更新的 KIE 容器部署的轮询。



注意

避免将 KIE 扫描程序与业务流程一起使用。将 KIE 扫描程序与进程搭配使用可能会导致无法预见的更新，然后在更改与正在运行的进程实例不兼容时导致长时间运行的进程出现错误。

表 23.7. 命令属性

名称	描述	要求
container-id	使用 KIE 扫描程序的 KIE 容器的 ID	必填
status	要在 KIE 扫描程序上设置的状态 (**STARTED**、**STOPPED**)	必填
poll-interval	允许以毫秒为单位的轮询持续时间	仅在启动扫描程序时才需要

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "update-scanner" : {
      "scanner" : {
        "status" : "STARTED",
        "poll-interval" : 10000
      },
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java 客户端方法示例

```
KieScannerResource scannerResource = new KieScannerResource();
scannerResource.setPollInterval(10000);
scannerResource.setStatus(KieScannerStatus. STARTED);

ServiceResponse<KieScannerResource> response =
kieServicesClient.updateScanner("command-script-container", scannerResource);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Kie scanner successfully created.",
      "result": {
        "kie-scanner": {
          "status": "STARTED",
          "poll-interval": 10000
```

```
      }
     }
    }
   ]
}
```

UpdateReleaseIdCommand

更新指定 KIE 容器的发行版本 ID 数据（组 ID、工件 ID、版本）。

表 23.8. 命令属性

名称	描述	要求
container-id	要更新的 KIE 容器的 ID	必填
releaseId	更新了要应用到 KIE 容器的 GAV （组 ID、工件 ID、版本）数据	必填

REST 请求正文(JSON)示例

```
{
  "commands" : [ {
    "update-release-id" : {
      "releaseId" : {
        "version" : "1.1",
        "group-id" : "com.redhat",
        "artifact-id" : "Project1"
      },
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java 客户端方法示例

```
ServiceResponse<ReleaseId> response = kieServicesClient.updateReleaseId("command-script-container", "com.redhat:Project1:1.1");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Release id successfully updated",
      "result": {
        "release-id": {
          "group-id": "com.redhat",
          "artifact-id": "Project1",
          "version": "1.1"
        }
      }
```

```
    }
  ]
}
```

第 24 章 RED HAT PROCESS AUTOMATION MANAGER 中的运行时命令

Red Hat Process Automation Manager 支持向 KIE Server 发送与资产相关的操作的运行时命令，如执行所有规则或插入 KIE 会话中的对象。支持的运行时命令的完整列表位于 Red Hat Process Automation Manager 实例的 **org.drools.core.command.runtime** 软件包中。

在 KIE Server REST API 中，您可以使用全局 **org.drools.core.command.runtime** 命令或特定于规则的 **org.drools.core.command.runtime.rule** 命令作为 **POST** 请求的请求正文到 **http://SERVER:PORT/kie-server/services/rest/server/containers/instances/{containerId}**。有关使用 KIE 服务器 REST API 的更多信息，请参阅 *第 21 章 KIE 服务器 REST API 用于 KIE 容器和业务资产*。

在 KIE Server Java 客户端 API 中，您可以将这些命令嵌入到 Java 应用程序以及相关的 Java 客户端中。例如，对于规则相关的命令，您可以使用带有内嵌命令的 **RuleServicesClient** Java 客户端。有关使用 KIE Server Java 客户端 API 的详情，请参考 *第 22 章 KIE Server Java 客户端 API 用于 KIE 容器和业务资产*。

24.1. RED HAT PROCESS AUTOMATION MANAGER 中的运行时命令示例

以下是您可以在 KIE Server REST API 中用于 KIE Server REST API 或 Java 客户端 API 的运行时命令示例，用于 KIE Server 中与资产相关的操作：

- **BatchExecutionCommand**
- **InsertObjectCommand**
- **RetractCommand**
- **ModifyCommand**
- **GetObjectCommand**
- **GetObjectsCommand**
- **InsertElementsCommand**
- **FireAllRulesCommand**
- **StartProcessCommand**
- **SignalEventCommand**
- **CompleteWorkItemCommand**
- **AbortWorkItemCommand**
- **QueryCommand**
- **SetGlobalCommand**
- **GetGlobalCommand**

有关支持的运行时命令的完整列表，请查看 Red Hat Process Automation Manager 实例中的 **org.drools.core.command.runtime** 软件包。

本节中的每个命令都包括 KIE Server REST API 的 REST 请求正文示例(JSON)，以及 KIE Server Java 客户端 API 的嵌入式 Java 命令示例。Java 示例使用对象 **org.drools.compiler.test.Person**，**字段名称**（字符串） **和年龄** (Integer)。

BatchExecutionCommand

包含要一起执行的多个命令。

表 24.1. 命令属性

名称	描述	要求
commands	要执行的命令列表。	必填
lookup	设置要执行的命令的 KIE 会话 ID。对于无状态 KIE 会话，需要此属性。对于有状态 KIE 会话，此属性是可选的，如果没有指定，则使用默认的 KIE 会话。	无状态 KIE 会话需要此项，对于有状态 KIE 会话是可选的

注意

KIE 会话 ID 位于 Red Hat Process Automation Manager 项目的 **kmodule.xml** 文件中。要在 Business Central 中查看或添加 KIE 会话 ID 以用于 **lookup** command 属性，请导航到 Business Central 中的相关项目，再前往项目 **Settings → KIE bases → KIE 会话**。如果没有 KIE 基础，请点击 **Add KIE base→ KIE sessions** 来定义新的 KIE 基础和 KIE 会话。

JSON 请求正文示例

```
{
  "lookup": "ksession1",
  "commands": [ {
     "insert": {
       "object": {
         "org.drools.compiler.test.Person": {
           "name": "john",
           "age": 25
         }
       }
     }
    },
    {
      "fire-all-rules": {
        "max": 10,
        "out-identifier": "firedActivations"
      }
    }
  ]
}
```

Java 命令示例

```
InsertObjectCommand insertCommand = new InsertObjectCommand(new Person("john", 25));
FireAllRulesCommand fireCommand = new FireAllRulesCommand();
```

```
BatchExecutionCommand batch = new
BatchExecutionCommandImpl(Arrays.asList(insertCommand, fireCommand), "ksession1");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": 0,
              "key": "firedActivations"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

InsertObjectCommand

将对象插入到 KIE 会话中。

表 24.2. 命令属性

名称	描述	要求
对象	要插入的对象	必填
out-identifier	从对象插入创建并添加到执行结果中的 **FactHandle** ID	选填
return-object	确定对象是否在执行结果中返回的布尔值（默认为 **true**）	选填
entry-point	插入的入口点	选填

JSON 请求正文示例

```
{
  "commands": [ {
     "insert": {
       "entry-point": "my stream",
       "object": {
         "org.drools.compiler.test.Person": {
           "age": 25,
```

```
            "name": "john"
          }
        },
        "out-identifier": "john",
        "return-object": false
      }
    }
  ]
}
```

Java 命令示例

```
Command insertObjectCommand =
  CommandFactory.newInsert(new Person("john", 25), "john", false, null);

ksession.execute(insertObjectCommand);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": [
            {
              "value": {
                "org.drools.core.common.DefaultFactHandle": {
                  "external-form": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
                }
              },
              "key": "john"
            }
          ]
        }
      }
    }
  ]
}
```

RetractCommand

限制 KIE 会话中的对象.

表 24.3. 命令属性

名称	描述	要求
fact-handle	与要调整的对象关联的 **FactHandle**	必填

JSON 请求正文示例

```
{
  "commands": [ {
     "retract": {
       "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
     }
    }
  ]
}
```

示例 Java 命令：使用 FactHandleFromString

```
RetractCommand retractCommand = new RetractCommand();
retractCommand.setFactHandleFromString("123:234:345:456:567");
```

示例 Java 命令：使用插入的对象中的 FactHandle

```
RetractCommand retractCommand = new RetractCommand(factHandle);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

ModifyCommand

在 KIE 会话中修改之前插入的对象。

表 24.4. 命令属性

名称	描述	要求
fact-handle	与要修改的对象关联的 **FactHandle**	必填
setters	对象修改的设置列表	必填

JSON 请求正文示例

```
{
  "commands": [ {
     "modify": {
       "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap",
       "setters": {
         "accessor": "age",
         "value": 25
       }
     }
   }
  ]
}
```

Java 命令示例

```
ModifyCommand modifyCommand = new ModifyCommand(factHandle);

List<Setter> setters = new ArrayList<Setter>();
setters.add(new SetterImpl("age", "25"));

modifyCommand.setSetters(setters);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

GetObjectCommand

从 KIE 会话检索对象。

表 24.5. 命令属性

名称	描述	要求
fact-handle	与要检索的对象关联的 **FactHandle**	必填
out-identifier	从对象插入创建并添加到执行结果中的 **FactHandle** ID	选填

JSON 请求正文示例

```
{
  "commands": [ {
     "get-object": {
       "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap",
       "out-identifier": "john"
     }
    }
  ]
}
```

Java 命令示例

```
GetObjectCommand getObjectCommand = new GetObjectCommand();
getObjectCommand.setFactHandleFromString("123:234:345:456:567");
getObjectCommand.setOutIdentifier("john");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": null,
              "key": "john"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

GetObjectsCommand

以集合形式从 KIE 会话检索所有对象。

表 24.6. 命令属性

名称	描述	要求
object-filter	从 KIE 会话返回的对象的过滤器	选填
out-identifier	执行结果中使用的标识符	选填

JSON 请求正文示例

```
{
  "commands": [ {
     "get-objects": {
       "out-identifier": "objects"
     }
   }
  ]
}
```

Java 命令示例

```
GetObjectsCommand getObjectsCommand = new GetObjectsCommand();
getObjectsCommand.setOutIdentifier("objects");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": [
                {
                  "org.apache.xerces.dom.ElementNSImpl": "<?xml version=\"1.0\" encoding=\"UTF-
16\"?>\n<object xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=\"person\">
<age>25</age><name>john</name>\n <\/object>"
                },
                {
                  "org.drools.compiler.test.Person": {
                    "name": "john",
                    "age": 25
                  }
                }
              ],
              "key": "objects"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

InsertElementsCommand

将对象列表插入到 KIE 会话中。

表 24.7. 命令属性

名称	描述	要求
对象	要插入到 KIE 会话中的对象列表	必填
out-identifier	从对象插入创建并添加到执行结果中的 **FactHandle** ID	选填
return-object	确定对象是否在执行结果中返回的布尔值。默认值： **true**。	选填
entry-point	插入的入口点	选填

JSON 请求正文示例

```
{
  "commands": [ {
    "insert-elements": {
      "objects": [
        {
          "containedObject": {
            "@class": "org.drools.compiler.test.Person",
            "age": 25,
            "name": "john"
          }
        },
        {
          "containedObject": {
            "@class": "Person",
            "age": 35,
            "name": "sarah"
          }
        }
      ]
    }
  }
]
}
```

Java 命令示例

```
List<Object> objects = new ArrayList<Object>();
objects.add(new Person("john", 25));
objects.add(new Person("sarah", 35));

Command insertElementsCommand = CommandFactory.newInsertElements(objects);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
```

```
    "type": "SUCCESS",
    "msg": "Container command-script-container successfully called.",
    "result": {
      "execution-results": {
        "results": [],
        "facts": [
          {
            "value": {
              "org.drools.core.common.DefaultFactHandle": {
                "external-form": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
              }
            },
            "key": "john"
          },
          {
            "value": {
              "org.drools.core.common.DefaultFactHandle": {
                "external-form": "0:4:436792766:-
2127720266:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
              }
            },
            "key": "sarah"
          }
        ]
      }
    }
  }
 ]
}
```


FireAllRulesCommand

执行 KIE 会话中的所有规则。

表 24.8. 命令属性

名称	描述	要求
max	要执行的规则的最大数量。默认值为 **-1，** 不会对执行施加任何限制。	选填
out-identifier	用于检索执行结果中触发的规则数量的 ID。	选填
agenda-filter	用于规则执行的议程过滤。	选填

JSON 请求正文示例


```
{
  "commands" : [ {
    "fire-all-rules": {
      "max": 10,
      "out-identifier": "firedActivations"
```

```
    }
  } ]
}
```

Java 命令示例

```
FireAllRulesCommand fireAllRulesCommand = new FireAllRulesCommand();
fireAllRulesCommand.setMax(10);
fireAllRulesCommand.setOutIdentifier("firedActivations");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": 0,
              "key": "firedActivations"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

StartProcessCommand

使用进程 ID 启动一个进程。您也可以传递要插入的参数和初始数据。

表 24.9. 命令属性

名称	描述	要求
processId	要启动的进程的 ID	必填
parameters	**Map <String,Object >** 参数在进程启动中传递参数	选填
data	在进程启动前插入 KIE 会话的对象列表	选填

JSON 请求正文示例

```
{
  "commands": [
    {
      "start-process": {
```

```
        "processId": "myProject.myProcess",
        "data": null,
        "parameter": [],
        "out-identifier": null
      }
    }
  ]
}
```

Java 命令示例

```
StartProcessCommand startProcessCommand = new StartProcessCommand();
startProcessCommand.setProcessId("org.drools.task.processOne");
```

服务器响应示例(JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [],
      "facts": []
    }
  }
}
```

SignalEventCommand

向 KIE 会话发送一个信号事件。

表 24.10. 命令属性

名称	描述	要求
event-type	传入事件的类型	必填
process-instance-id	要信号的进程实例的 ID	选填
event	传入事件的数据	选填

JSON 请求正文示例

```
{
  "commands": [
    {
      "signal-event": {
        "process-instance-id": 1001,
        "correlation-key": null,
        "event-type": "start",
        "event": {
          "org.kie.server.testing.Person": {
```

```
            "fullname": "john",
            "age": 25
          }
        }
      }
    }
  ]
}
```

Java 命令示例

```
SignalEventCommand signalEventCommand = new SignalEventCommand();
signalEventCommand.setProcessInstanceId(1001);
signalEventCommand.setEventType("start");
signalEventCommand.setEvent(new Person("john", 25));
```

服务器响应示例(JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [],
      "facts": []
    }
  }
}
```

CompleteWorkItemCommand

在 KIE 会话中完成一个工作项目。

表 24.11. 命令属性

名称	描述	要求
workItemId	要完成的工作项目的 ID	必填
results	工作项目的结果	选填

JSON 请求正文示例

```
{
  "commands": [ {
    "complete-work-item": {
        "id": 1001
    }
  }
]
}
```

Java 命令示例

```
CompleteWorkItemCommand completeWorkItemCommand = new
CompleteWorkItemCommand();
completeWorkItemCommand.setWorkItemId(1001);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

AbortWorkItemCommand

在 KIE 会话中中止工作项，其方式与 **ksession.getWorkItemManager （）.abortWorkItem (workItemId)** 相同。

表 24.12. 命令属性

名称	描述	要求
workItemId	要中止的工作项目的 ID	必填

JSON 请求正文示例

```
{
  "commands": [ {
     "abort-work-item": {
       "id": 1001
     }
    }
  ]
}
```

Java 命令示例

```
AbortWorkItemCommand abortWorkItemCommand = new AbortWorkItemCommand();
abortWorkItemCommand.setWorkItemId(1001);
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

QueryCommand

执行 KIE 基础中定义的查询。

表 24.13. 命令属性

名称	描述	要求
name	查询名称。	必填
out-identifier	查询结果的 ID。查询结果会添加到具有此标识符的执行结果中。	选填
参数	要作为参数传递的对象列表。	选填

JSON 请求正文示例

```
{
  "commands": [
    {
      "query": {
        "name": "persons",
        "arguments": [],
        "out-identifier": "persons"
      }
    }
  ]
}
```

Java 命令示例

```
QueryCommand queryCommand = new QueryCommand();
queryCommand.setName("persons");
queryCommand.setOutIdentifier("persons");
```

服务器响应示例(JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [
        {
          "value": {
            "org.drools.core.runtime.rule.impl.FlatQueryResults": {
              "idFactHandleMaps": {
                "type": "LIST",
                "componentType": null,
                "element": [
                  {
                    "type": "MAP",
                    "componentType": null,
                    "element": [
                      {
                        "value": {
                          "org.drools.core.common.DisconnectedFactHandle": {
                            "id": 1,
                            "identityHashCode": 1809949690,
                            "objectHashCode": 1809949690,
                            "recency": 1,
                            "object": {
                              "org.kie.server.testing.Person": {
                                "fullname": "John Doe",
                                "age": 47
                              }
                            },
                            "entryPointId": "DEFAULT",
                            "traitType": "NON_TRAIT",
                            "external-form":
"0:1:1809949690:1809949690:1:DEFAULT:NON_TRAIT:org.kie.server.testing.Person"
                          }
                        },
                        "key": "$person"
                      }
                    ]
                  }
                ]
              },
              "idResultMaps": {
                "type": "LIST",
                "componentType": null,
                "element": [
                  {
                    "type": "MAP",
                    "componentType": null,
                    "element": [
                      {
                        "value": {
                          "org.kie.server.testing.Person": {
                            "fullname": "John Doe",
                            "age": 47
                          }
```

```
                    },
                    "key": "$person"
                  }
                ]
              }
            ]
          },
          "identifiers": {
            "type": "SET",
            "componentType": null,
            "element": [
              "$person"
            ]
          }
        }
      },
      "key": "persons"
    }
  ],
  "facts": []
 }
}
}
```

SetGlobalCommand

将对象设置为全局状态。

表 24.14. 命令属性

名称	描述	要求
identifier	KIE 基础中定义的全局变量的 ID	必填
对象	要设置为全局变量的对象	选填
out	从执行结果中排除您设置的全局变量的布尔值	选填
out-identifier	全局执行结果的 ID	选填

JSON 请求正文示例

```
{
  "commands": [
    {
      "set-global": {
        "identifier": "helper",
        "object": {
          "org.kie.server.testing.Person": {
            "fullname": "kyle",
            "age": 30
          }
        },
```

```
        "out-identifier": "output"
      }
    }
  ]
}
```

Java 命令示例

```
SetGlobalCommand setGlobalCommand = new SetGlobalCommand();
setGlobalCommand.setIdentifier("helper");
setGlobalCommand.setObject(new Person("kyle", 30));
setGlobalCommand.setOut(true);
setGlobalCommand.setOutIdentifier("output");
```

服务器响应示例(JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [
        {
          "value": {
            "org.kie.server.testing.Person": {
              "fullname": "kyle",
              "age": 30
            }
          },
          "key": "output"
        }
      ],
      "facts": []
    }
  }
}
```

GetGlobalCommand

检索之前定义的全局对象。

表 24.15. 命令属性

名称	描述	要求
identifier	KIE 基础中定义的全局变量的 ID	必填
out-identifier	执行结果中使用的 ID	选填

JSON 请求正文示例

```
{
  "commands": [ {
```

```
      "get-global": {
        "identifier": "helper",
        "out-identifier": "helperOutput"
      }
    }
  ]
}
```

Java 命令示例

```
GetGlobalCommand getGlobalCommand = new GetGlobalCommand();
getGlobalCommand.setIdentifier("helper");
getGlobalCommand.setOutIdentifier("helperOutput");
```

服务器响应示例(JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": null,
              "key": "helperOutput"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

第 25 章 用于 KIE 服务器模板和实例的 AUTOMATION MANAGER 控制器 REST API

Red Hat Process Automation Manager 提供了一个流程自动化管理器控制器 REST API，您可以在不使用 Business Central 用户界面的情况下与 KIE Server 模板（配置）、KIE 服务器实例（远程服务器）以及关联的 KIE 容器（部署单元）进行交互。通过此 API 支持，您可以更有效地维护红帽流程自动化管理器服务器和资源，并优化与红帽流程自动化管理器的集成和开发。

使用 Process Automation Manager 控制器 REST API，您可以执行以下操作：

- 检索有关 KIE 服务器模板、实例和关联的 KIE 容器的信息
- 更新、启动或停止与 KIE 服务器模板和实例关联的 KIE 容器
- 创建、更新或删除 KIE 服务器模板
- 创建、更新或删除 KIE 服务器实例

对 Process Automation Manager 控制器 REST API 的请求需要以下组件：

身份验证

根据控制器类型，流程 Automation Manager 控制器 REST API 需要以下用户角色基于 HTTP 基本身份验证或基于令牌的身份验证：

- 如果您安装了 Business Central 且想要使用内置 Automation Manager 控制器，REST **-all** 用户角色
- 如果您安装无头进程 Automation Manager 控制器与 Business Central 分开，则 **kie-server** 用户角色

要查看为您的 Red Hat Process Automation Manager 分发配置的用户角色，请导航到 **~/$SERVER_HOME/standalone/configuration/application-roles.properties** 和 **~/application-users.properties**。

要添加具有 **kie-server** 角色或 **rest-all** 角色的用户，请导航到 **~/$SERVER_HOME/bin**，并使用指定的角色或角色运行以下命令：

```
$ ./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','rest-all'])"
```

要使用 Process Automation Manager 控制器访问配置 **kie-server** 或 **rest-all** 用户，请导航到 **~/$SERVER_HOME/standalone/configuration/standalone-full.xml**，取消注释 **org.kie.server** 属性（如果适用），并添加控制器用户登录凭证和控制器位置（如果需要）：

```
<property name="org.kie.server.location" value="http://localhost:8080/kie-server/services/rest/server"/>
<property name="org.kie.server.controller" value="http://localhost:8080/business-central/rest/controller"/>
<property name="org.kie.server.controller.user" value="baAdmin"/>
<property name="org.kie.server.controller.pwd" value="password@1"/>
<property name="org.kie.server.id" value="default-kieserver"/>
```

有关用户角色和 Red Hat Process Automation Manager 安装选项的更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装*。

HTTP 标头

Process Automation Manager 控制器 REST API 需要以下 API 请求的 HTTP 标头：

- **接受**: 请求客户端接受的数据格式：
  - **application/json** (JSON)
  - **application/xml** （用于 JAXB 的XML）
- **Content-Type**: **POST** 或 **PUT** API 请求数据的数据格式：
  - **application/json** (JSON)
  - **application/xml** （用于 JAXB 的XML）

HTTP 方法

Process Automation Manager 控制器 REST API 支持以下 API 请求 HTTP 方法：

- **GET** ：从指定的资源端点检索指定的信息
- **POST** ：更新资源或资源实例
- **PUT** ：创建资源或资源实例
- **DELETE** ：删除资源或资源实例

基本 URL

Process Automation Manager 控制器 REST API 请求的基本 URL 是 **http://SERVER:PORT/CONTROLLER/rest/**，例如，如果您使用内置于 Business Central 的 Process Automation Manager，则为 **http://localhost:8080/business-central/rest/**。

Endpoints

进程 Automation Manager 控制器 REST API 端点，如指定 KIE Server 模板的 **/controller/management/servers/{serverTemplateId}**，是您附加到 Process Automation Manager 控制器 REST API 基础 URL 中的 URI，以访问 Red Hat Process Automation Manager 中对应的服务器资源或服务器资源类型。

/controller/management/servers/{serverTemplateId} 端点的请求 URL 示例

http://localhost:8080/business-central/rest/controller/management/servers/default-kieserver

请求参数和请求数据

有些进程 Automation Manager 控制器 REST API 请求需要请求 URL 路径中的特定参数来识别或过滤特定资源并执行特定操作。您可以在端点中添加 URL 参数，格式为 **?<PARAM>=<VALUE>&<PARAM>=<VALUE** >。

带有参数的 DELETE 请求 URL 示例

http://localhost:8080/business-central/rest/controller/server/new-kieserver-instance?location=http://localhost:8080/kie-server/services/rest/server

HTTP **POST** 和 **PUT** 请求可能还需要请求正文或文件，并附带请求。

PUT 请求 URL 和 JSON 请求正文数据示例

http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver

```
{
  "server-id": "new-kieserver",
  "server-name": "new-kieserver",
  "container-specs": [],
  "server-config": {},
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}
```

25.1. 使用 REST 客户端或 CURL 工具使用 PROCESS AUTOMATION MANAGER 控制器 REST API 发送请求

Process Automation Manager 控制器 REST API 可让您在 Red Hat Process Automation Manager 中与 KIE Server 模板（配置）、KIE 服务器实例（远程服务器）以及关联的 KIE 容器（部署单元）进行交互。您可以使用任何 REST 客户端或 curl 工具发送流程 Automation Manager 控制器 REST API 请求。

先决条件

- KIE 服务器已安装并运行。
- Process Automation Manager 控制器或无头进程 Automation Manager 控制器已安装并在运行。
- 如果您安装了 Business Central，或者 Kie **-server** 用户角色访问无头流程自动化管理器控制器，则您拥有对流程自动化管理器控制器的 **rest-all** 用户角色访问权限。

流程

1. 识别您要发送请求的相关 API 端点，如 **[GET] /controller/management/servers**，以便从 Process Automation Manager 控制器检索 KIE Server 模板。
2. 在 REST 客户端或 curl 工具中，输入到 **controller/management/servers** 的 **GET** 请求的以下组件。根据您的用例调整任何请求详情。
   对于 REST 客户端：
   - **身份验证** ：使用 **rest-all** 角色输入流程 Automation Manager 控制器用户的用户名和密码，或使用 **kie-server** 角色输入无头处理 Automation Manager 控制器用户。
   - **HTTP Headers** ：设置以下标头：
     - **接受:application/json**
   - **HTTP 方法** ：设置为 **GET**。
   - **URL** ：输入 Process Automation Manager 控制器 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/controller/management/servers**。

   对于 curl 工具：

- **-** U ：使用 **rest-all** 角色或无头进程 Automation Manager 控制器用户输入进程 Automation Manager 控制器 **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 Process Automation Manager 控制器 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/controller/management/servers**。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
"http://localhost:8080/business-central/rest/controller/management/servers"
```

3. 执行请求并查看流程 Automation Manager 控制器响应。
服务器响应示例(JSON)：


```
{
  "server-template": [
    {
      "server-id": "default-kieserver",
      "server-name": "default-kieserver",
      "container-specs": [
        {
          "container-id": "employeerostering_1.0.0-SNAPSHOT",
          "container-name": "employeerostering",
          "server-template-key": {
            "server-id": "default-kieserver",
            "server-name": "default-kieserver"
          },
          "release-id": {
            "group-id": "employeerostering",
            "artifact-id": "employeerostering",
            "version": "1.0.0-SNAPSHOT"
          },
          "configuration": {
            "RULE": {
              "org.kie.server.controller.api.model.spec.RuleConfig": {
                "pollInterval": null,
                "scannerStatus": "STOPPED"
              }
            },
            "PROCESS": {
              "org.kie.server.controller.api.model.spec.ProcessConfig": {
                "runtimeStrategy": "SINGLETON",
                "kbase": "",
                "ksession": "",
                "mergeMode": "MERGE_COLLECTIONS"
              }
            }
          },
          "status": "STARTED"
        },
        {
          "container-id": "mortgage-process_1.0.0-SNAPSHOT",
```

```
        "container-name": "mortgage-process",
        "server-template-key": {
          "server-id": "default-kieserver",
          "server-name": "default-kieserver"
        },
        "release-id": {
          "group-id": "mortgage-process",
          "artifact-id": "mortgage-process",
          "version": "1.0.0-SNAPSHOT"
        },
        "configuration": {
          "RULE": {
            "org.kie.server.controller.api.model.spec.RuleConfig": {
              "pollInterval": null,
              "scannerStatus": "STOPPED"
            }
          },
          "PROCESS": {
            "org.kie.server.controller.api.model.spec.ProcessConfig": {
              "runtimeStrategy": "PER_PROCESS_INSTANCE",
              "kbase": "",
              "ksession": "",
              "mergeMode": "MERGE_COLLECTIONS"
            }
          }
        },
        "status": "STARTED"
      }
    ],
    "server-config": {},
    "server-instances": [
      {
        "server-instance-id": "default-kieserver-instance@localhost:8080",
        "server-name": "default-kieserver-instance@localhost:8080",
        "server-template-id": "default-kieserver",
        "server-url": "http://localhost:8080/kie-server/services/rest/server"
      }
    ],
    "capabilities": [
      "RULE",
      "PROCESS",
      "PLANNING"
    ]
  }
 ]
}
```


4. 在 REST 客户端或 curl 实用程序中，发送带有以下组件的 API 请求，它向 **/controller/management/servers/{serverTemplateId}** 发送一个 **PUT** 请求，以创建新的 KIE Server 模板。根据您的用例调整任何请求详情。
对于 REST 客户端：

   - **身份验证** ：使用 **rest-all** 角色输入流程 Automation Manager 控制器用户的用户名和密码，或使用 **kie-server** 角色输入无头处理 Automation Manager 控制器用户。

   - **HTTP 标头** ：设置以下标头：

- **接受**:**application/json**
- **Content-Type**:**application/json**

- **HTTP 方法** ：设置为 **PUT**。
- **URL** ：输入 Process Automation Manager 控制器 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver**。
- **请求正文** ：添加带有新 KIE Server 模板配置的 JSON 请求正文：

```
{
  "server-id": "new-kieserver",
  "server-name": "new-kieserver",
  "container-specs": [],
  "server-config": {},
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}
```

对于 curl 工具：

- **-** U ：使用 **rest-all** 角色或无头进程 Automation Manager 控制器用户输入进程 Automation Manager 控制器 **用户的用户名和密码**。
- **-** h ：设置以下标头：
  - **接受**:**application/json**
  - **Content-Type**:**application/json**
- **-x** ：设置为 **PUT**。
- **URL** ：输入 Process Automation Manager 控制器 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver**。
- **-d** ：添加 JSON 请求正文或文件(**@file.json**)，其中包含新的 KIE Server 模板的配置：

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X PUT "http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver" -d "{ \"server-id\": \"new-kieserver\", \"server-name\": \"new-kieserver\", \"container-specs\": [], \"server-config\": {}, \"capabilities\": [ \"RULE\", \"PROCESS\", \"PLANNING\" ]}"
```

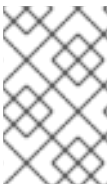
```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X PUT "http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver" -d @my-server-template-configs.json
```

5. 执行请求并确认成功完成流程 Automation Manager 控制器响应。

如果您遇到请求错误，请检查返回的错误消息并相应地调整您的请求。

25.2. 使用 SWAGGER 接口使用 PROCESS AUTOMATION MANAGER 控制器 REST API 发送请求

Process Automation Manager 控制器 REST API 支持一个 Swagger web 界面，您可以使用它而不是独立 REST 客户端或 curl 工具与 Red Hat Process Automation Manager 中的 KIE 服务器模板、实例和相关 KIE 容器交互，而无需使用 Business Central 用户界面。



注意

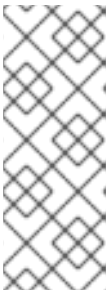
默认情况下，Process Automation Manager 控制器的 Swagger web 界面由 **org.kie.workbench.swagger.disabled=false** 系统属性启用。要禁用 Process Automation Manager 控制器的 Swagger Web 界面，请将这个系统属性设置为 **true**。

先决条件

- Process Automation Manager 控制器已安装并运行。
- 如果您安装了 Business Central，或者 Kie **-server** 用户角色访问无头流程自动化管理器控制器，则您拥有对流程自动化管理器控制器的 **rest-all** 用户角色访问权限。

流程

1. 在 Web 浏览器中，导航到 **http://SERVER:PORT/CONTROLLER/docs**，如 **http://localhost:8080/business-central/docs**，并使用 Process Automation Manager 控制器用户的用户名和密码登录，使用 **rest-all** 角色或无头流程 Automation Manager 控制器用户，使用 **kie-server** 角色登录。



注意

如果您使用内置在 Business Central 中的 Process Automation Manager 控制器，与流程 Automation Manager 控制器关联的 Swagger 页面被识别为 Business Central REST 服务的 " business Central API"。如果您在不使用 Business Central 的情况下使用无头处理 Automation Manager 控制器，与无头进程 Automation Manager 控制器关联的 Swagger 页面被识别为 "Controller API"。在这两种情况下，流程 Automation Manager 控制器 REST API 端点都是相同的。

2. 在 Swagger 页面中，选择要发送请求的相关 API 端点，如 **Controller :: KIE Server 模板和 KIE containers → [GET] /controller/management/servers**，以便从 Process Automation Manager 控制器检索 KIE Server 模板。
3. 点 **Try it out**，并提供您要过滤结果的任何可选参数（如果适用）。
4. 在 **Response 内容类型下拉菜单中**，选择服务器响应所需的格式，如用于 JSON 格式的 **application/json**。
5. 点 **Execute** 并查看 KIE Server 响应。
服务器响应示例(JSON)：

```
{
  "server-template": [
    {
      "server-id": "default-kieserver",
```

```
"server-name": "default-kieserver",
"container-specs": [
 {
  "container-id": "employeerostering_1.0.0-SNAPSHOT",
  "container-name": "employeerostering",
  "server-template-key": {
   "server-id": "default-kieserver",
   "server-name": "default-kieserver"
  },
  "release-id": {
   "group-id": "employeerostering",
   "artifact-id": "employeerostering",
   "version": "1.0.0-SNAPSHOT"
  },
  "configuration": {
   "RULE": {
    "org.kie.server.controller.api.model.spec.RuleConfig": {
     "pollInterval": null,
     "scannerStatus": "STOPPED"
    }
   },
   "PROCESS": {
    "org.kie.server.controller.api.model.spec.ProcessConfig": {
     "runtimeStrategy": "SINGLETON",
     "kbase": "",
     "ksession": "",
     "mergeMode": "MERGE_COLLECTIONS"
    }
   }
  },
  "status": "STARTED"
 },
 {
  "container-id": "mortgage-process_1.0.0-SNAPSHOT",
  "container-name": "mortgage-process",
  "server-template-key": {
   "server-id": "default-kieserver",
   "server-name": "default-kieserver"
  },
  "release-id": {
   "group-id": "mortgage-process",
   "artifact-id": "mortgage-process",
   "version": "1.0.0-SNAPSHOT"
  },
  "configuration": {
   "RULE": {
    "org.kie.server.controller.api.model.spec.RuleConfig": {
     "pollInterval": null,
     "scannerStatus": "STOPPED"
    }
   },
   "PROCESS": {
    "org.kie.server.controller.api.model.spec.ProcessConfig": {
     "runtimeStrategy": "PER_PROCESS_INSTANCE",
     "kbase": "",
     "ksession": "",
```

```
            "mergeMode": "MERGE_COLLECTIONS"
          }
        }
      },
      "status": "STARTED"
    }
  ],
  "server-config": {},
  "server-instances": [
    {
      "server-instance-id": "default-kieserver-instance@localhost:8080",
      "server-name": "default-kieserver-instance@localhost:8080",
      "server-template-id": "default-kieserver",
      "server-url": "http://localhost:8080/kie-server/services/rest/server"
    }
  ],
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
 }
]
}
```


6. 在 Swagger 页面中，导航到 **Controller :: KIE Server 模板和 KIE containers→ [GET] /controller/management/servers/{serverTemplateId}** 端点来发送另一个请求以创建新的 KIE Server 模板。根据您的用例调整任何请求详情。

7. 点击 **Try it out**，并为请求输入以下组件：

   - **serverTemplateId** ：输入新 KIE Server 模板的 ID，如 **new-kieserver**。
   - **body** ： **将参数内容类型设置为** 所需的请求正文格式，如用于 JSON 格式的 **application/json**，并使用新 KIE Server 模板的配置添加请求正文：


```
{
 "server-id": "new-kieserver",
 "server-name": "new-kieserver",
 "container-specs": [],
 "server-config": {},
 "capabilities": [
  "RULE",
  "PROCESS",
  "PLANNING"
 ]
}
```


8. 在 **Response 内容类型下拉菜单中**，选择服务器响应所需的格式，如用于 JSON 格式的 **application/json**。

9. 点 **Execute** 并确认成功处理 Automation Manager 控制器响应。
   如果您遇到请求错误，请检查返回的错误消息并相应地调整您的请求。

25.3. 支持的流程自动化管理器控制器 REST API 端点

Process Automation Manager 控制器 REST API 提供与 KIE Server 模板（配置）、KIE 服务器实例（远程服务器）以及关联的 KIE 容器（部署单元）交互的端点。Process Automation Manager 控制器 REST API 基本 URL 是 **http://SERVER:PORT/CONTROLLER/rest/**。所有请求都需要 HTTP 基本身份验证，如果您安装了 Business Central 并希望使用内置的 Process Automation Manager控制器则为 **rest-all** 用户使用基于令牌的验证；如果安装了独立于 Business Central 的无头 Process Automation Manager 控制器，则为 **kie-server** 使用基于令牌的身份验证。

如需流程 Automation Manager 控制器 REST API 端点和描述的完整列表，请使用以下资源之一：

- JBPM 文档页面上的控制器 REST API （静态）
- 位于 **http://SERVER:PORT/CONTROLLER/docs** 的过程 Automation Manager 控制器 REST API 的 Swagger UI （需要运行 Process Automation Manager 控制器）

注意

默认情况下，Process Automation Manager 控制器的 Swagger web 界面由 **org.kie.workbench.swagger.disabled=false** 系统属性启用。要禁用 Process Automation Manager 控制器的 Swagger Web 界面，请将这个系统属性设置为 **true**。

如果您使用内置在 Business Central 中的 Process Automation Manager 控制器，与流程 Automation Manager 控制器关联的 Swagger 页面被识别为 Business Central REST 服务的 " business Central API"。如果您在不使用 Business Central 的情况下使用无头处理 Automation Manager 控制器，与无头进程 Automation Manager 控制器关联的 Swagger 页面被识别为 "Controller API"。在这两种情况下，流程 Automation Manager 控制器 REST API 端点都是相同的。

第 26 章 为 KIE 服务器模板和实例处理自动化管理器控制器 JAVA 客户端 API

Red Hat Process Automation Manager 提供了一个流程 Automation Manager 控制器 Java 客户端 API，可让您使用 Java 客户端应用程序中的 REST 或 WebSocket 协议连接到流程自动化管理器控制器。您可以使用 Process Automation Manager 控制器 Java 客户端 API 作为 Process Automation Manager 控制器 REST API 的替代选择，来与 KIE Server 模板（配置）、KIE Server 实例（远程服务器）以及 Red Hat Process Automation Manager 中的关联 KIE 容器（部署单元）进行交互，而无需使用 Business Central 用户界面。通过此 API 支持，您可以更有效地维护红帽流程自动化管理器服务器和资源，并优化与红帽流程自动化管理器的集成和开发。

使用 Process Automation Manager 控制器 Java 客户端 API，您还可以执行流程 Automation Manager 控制器 REST API 支持的以下操作：

- 检索有关 KIE 服务器模板、实例和关联的 KIE 容器的信息
- 更新、启动或停止与 KIE 服务器模板和实例关联的 KIE 容器
- 创建、更新或删除 KIE 服务器模板
- 创建、更新或删除 KIE 服务器实例

处理 Automation Manager 控制器 Java 客户端 API 请求需要以下组件：

身份验证

Process Automation Manager 控制器 Java 客户端 API 需要以下用户角色的 HTTP 基本身份验证，具体取决于控制器类型：

- 如果您安装了 Business Central 且想要使用内置 Automation Manager 控制器，REST **-all** 用户角色
- 如果您安装无头进程 Automation Manager 控制器与 Business Central 分开，则 **kie-server** 用户角色

要查看为您的 Red Hat Process Automation Manager 分发配置的用户角色，请导航到 **~/$SERVER_HOME/standalone/configuration/application-roles.properties** 和 **~/application-users.properties**。

要添加具有 **kie-server** 角色或 **rest-all** 角色的用户（假设已经设置了 Keystore），请导航到 **~/$SERVER_HOME/bin**，并使用指定的角色或角色运行以下命令：

```
$ ./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=['rest-all','kie-server'])"
```

如果没有设置 Keystore，则执行以下命令来创建密钥存储：

```
$ keytool -importpassword -keystore
$SERVER_HOME/standalone/configuration/kie_keystore.jceks -keypass
<SECRETKEYPASSWORD> -alias kieserver -storepass <SECRETSTOREPASSWORD> -
storetype JCEKS
```

另外，将以下属性添加到 **~/$SERVER_HOME/standalone/configuration/standalone-full.xml** ：

```
    <property name="kie.keystore.keyStoreURL"
value="file:///data/jboss/rhpam780/standalone/configuration/kie_keystore.jceks"/>
    <property name="kie.keystore.keyStorePwd" value="<SECRETSTOREPASSWORD>"/>
    <property name="kie.keystore.key.server.alias" value="kieserver"/>
    <property name="kie.keystore.key.server.pwd" value="<SECRETKEYPASSWORD>"/>
    <property name="kie.keystore.key.ctrl.alias" value="kieserver"/>
    <property name="kie.keystore.key.ctrl.pwd" value="<SECRETKEYPASSWORD>"/>
```

要使用 Process Automation Manager 控制器访问配置 **kie-server** 或 **rest-all** 用户，请导航到 **~/$SERVER_HOME/standalone/configuration/standalone-full.xml**，取消注释 **org.kie.server** 属性（如果适用），并添加控制器用户登录凭证和控制器位置（如果需要）：

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

有关用户角色和 Red Hat Process Automation Manager 安装选项的更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装*。

项目依赖项

Process Automation Manager 控制器 Java 客户端 API 需要在 Java 项目的相关类路径上以下依赖项：

```
<!-- For remote execution on controller -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-controller-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- For REST client -->
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-client</artifactId>
  <version>${resteasy.version}</version>
</dependency>

<!-- For WebSocket client -->
<dependency>
  <groupId>io.undertow</groupId>
  <artifactId>undertow-websockets-jsr</artifactId>
  <version>${undertow.version}</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

Red Hat Process Automation Manager 依赖项的 **<version>** 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（如 7.59.0.Final-redhat-00006）。

注意

考虑将 Red Hat Business Automation Manager (BOM)依赖项添加到项目的 **pom.xml** 文件中，而不是为单独的依赖项指定 Red Hat Process Automation Manager < **version** >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确传输依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

有关 Red Hat Business Automation BOM 的更多信息，请参阅 RHPAM 产品和 maven 库版本之间的映射是什么？

客户端请求配置

与 Process Automation Manager 控制器 Java 客户端 API 的所有 Java 客户端请求必须至少定义以下控制器通信组件：

- 如果您安装了无头流程 Automation Manager 控制器，则 **rest-all** 用户的凭证（如果安装了 Business Central）或 **kie-server** 用户
- REST 或 WebSocket 协议的进程自动化管理器控制器位置：
  - REST URL 示例： **http://localhost:8080/business-central/rest/controller**
  - WebSocket URL 示例： **ws://localhost:8080/headless-controller/websocket/controller**
- API 请求和响应的 Marshalling 格式(JSON 或 JAXB)
- **KieServerControllerClient** 对象，充当使用 Java 客户端 API 启动服务器通信的入口点
- **KieServerControllerClientFactory** 定义 REST 或 WebSocket 协议和用户访问权限
- 所用的进程 Automation Manager 控制器客户端服务或服务，如 **listServerTemplates**、**getServerTemplate** 或 **getServerInstances**

以下是带有以下组件的 REST 和 WebSocket 客户端配置示例：

使用 REST 的客户端配置示例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;
```

```
public class ListServerTemplatesExample {

    private static final String URL = "http://localhost:8080/business-central/rest/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

    public static void main(String[] args) {
        KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
                                                                                           USER,
                                                                                           PASSWORD);

        final ServerTemplateList serverTemplateList = client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                         serverTemplateList.getServerTemplates().length,
                                         URL));
    }
}
```

使用 WebSocket 的客户端配置示例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class ListServerTemplatesExample {

    private static final String URL = "ws://localhost:8080/my-controller/websocket/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

    public static void main(String[] args) {
        KieServerControllerClient client =
KieServerControllerClientFactory.newWebSocketClient(URL,
                                                                                             USER,
                                                                                             PASSWORD);

        final ServerTemplateList serverTemplateList = client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                         serverTemplateList.getServerTemplates().length,
                                         URL));
    }
}
```

26.1. 使用 PROCESS AUTOMATION MANAGER 控制器 JAVA 客户端 API 发送请求

Process Automation Manager 控制器 Java 客户端 API 可让您使用 Java 客户端应用程序的 REST 或

WebSocket 协议连接到 Process Automation Manager 控制器。您可以使用 Process Automation Manager 控制器 Java 客户端 API 作为 Process Automation Manager 控制器 REST API 的替代选择，来与 KIE Server 模板（配置）、KIE Server 实例（远程服务器）以及 Red Hat Process Automation Manager 中的关联 KIE 容器（部署单元）进行交互，而无需使用 Business Central 用户界面。

先决条件

- KIE 服务器已安装并运行。
- Process Automation Manager 控制器或无头进程 Automation Manager 控制器已安装并在运行。
- 如果您安装了 Business Central，或者 Kie **-server** 用户角色访问无头流程自动化管理器控制器，则您拥有对流程自动化管理器控制器的 **rest-all** 用户角色访问权限。
- 您有一个带有 Red Hat Process Automation Manager 资源的 Java 项目。

流程

1. 在客户端应用程序中，确保以下依赖项已添加到 Java 项目的相关类路径中：

```
<!-- For remote execution on controller -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-controller-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- For REST client -->
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-client</artifactId>
  <version>${resteasy.version}</version>
</dependency>

<!-- For WebSocket client -->
<dependency>
  <groupId>io.undertow</groupId>
  <artifactId>undertow-websockets-jsr</artifactId>
  <version>${undertow.version}</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

2. 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution**，并进入 **~/rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller/kie-server-controller/src/main/java/org/kie/server/controller/client** 来访问流程自动化管理器 Java 客户端。

3. 在 **~/kie/server/controller/client** 文件夹中，识别您要发送的请求的相关 Java 客户端实施，如 **RestKieServerControllerClient** 实施，以访问 KIE Server 模板和 REST 协议中的 KIE 容器的客户端服务。

4. 在您的客户端应用中，为 API 请求创建一个 **.java** 类。类必须包含必要的导入、流程 Automation Manager 控制器位置和用户凭证、**KieServerControllerClient** 对象和客户端方法来执行，如 **createServerTemplate** 和来自 **RestKieServerControllerClient** 实现的 **createContainer**。根据您的用例调整任何配置详情。

创建与 KIE Server 模板和 KIE 容器交互

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.KieContainerStatus;
import org.kie.server.api.model.KieScannerStatus;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.controller.api.model.spec.*;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class RestTemplateContainerExample {

  private static final String URL = "http://localhost:8080/business-central/rest/controller";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

    private static KieServerControllerClient client;

    public static void main(String[] args) {
        KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
                                                                                           USER,
                                                                                           PASSWORD,
                                                                                           MarshallingFormat.JSON);
        // Create server template and KIE container, start and stop KIE container, and delete
server template
        ServerTemplate serverTemplate = createServerTemplate();
        ContainerSpec container = createContainer(serverTemplate);
        client.startContainer(container);
        client.stopContainer(container);
        client.deleteServerTemplate(serverTemplate.getId());
    }

    // Re-create and configure server template
    protected static ServerTemplate createServerTemplate() {
        ServerTemplate serverTemplate = new ServerTemplate();
        serverTemplate.setId("example-client-id");
        serverTemplate.setName("example-client-name");
        serverTemplate.setCapabilities(Arrays.asList(Capability.PROCESS.name(),
                                       Capability.RULE.name(),
                                       Capability.PLANNING.name()));

        client.saveServerTemplate(serverTemplate);
```

```
        return serverTemplate;
    }

    // Re-create and configure KIE containers
    protected static ContainerSpec createContainer(ServerTemplate serverTemplate){
        Map<Capability, ContainerConfig> containerConfigMap = new HashMap();

        ProcessConfig processConfig = new ProcessConfig("PER_PROCESS_INSTANCE",
"kieBase", "kieSession", "MERGE_COLLECTION");
        containerConfigMap.put(Capability.PROCESS, processConfig);

        RuleConfig ruleConfig = new RuleConfig(500l, KieScannerStatus.SCANNING);
        containerConfigMap.put(Capability.RULE, ruleConfig);

        ReleaseId releaseId = new ReleaseId("org.kie.server.testing", "stateless-session-kjar",
"1.0.0-SNAPSHOT");

        ContainerSpec containerSpec = new ContainerSpec("example-container-id", "example-
client-name", serverTemplate, releaseId, KieContainerStatus.STOPPED,
containerConfigMap);
        client.saveContainerSpec(serverTemplate.getId(), containerSpec);

        return containerSpec;
    }
}
```

5. 从项目目录运行配置的 **.java** 类来执行请求，并查看 Process Automation Manager 控制器响应。如果您启用了调试日志，KIE 服务器会根据您配置的 marshalling 格式（如 JSON）响应详细的响应。如果您遇到请求错误，请检查返回的错误消息并相应地调整 Java 配置。

26.2. 支持的流程自动化管理器控制器 JAVA 客户端

以下是 Red Hat Process Automation Manager 发行版本的 **org.kie.server.controller.client** 软件包中的一些 Java 客户端服务。您可以使用这些服务与 Process Automation Manager 控制器中的相关资源交互，类似于流程 Automation Manager 控制器 REST API。

- **KieServerControllerClient** ：用作与进程自动化管理器控制器通信的入口点
- **RestKieServerControllerClient** ：用于与 REST 协议中的 KIE 服务器模板和 KIE 容器交互的实施（在 **~/org/kie/server/controller/client/rest**）
- **WebSocketKieServerControllerClient** ：用于与 WebSocket 协议中的 KIE 服务器模板和 KIE 容器交互的实施（在 **~/org/kie/server/controller/client/websocket**中）

对于可用的流程自动化管理器控制器 Java 客户端的完整列表，请 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution**，再导航到 **~/rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller/src/main/java/org/kie/server/server/server/client/client**.

26.3. 使用 PROCESS AUTOMATION MANAGER 控制器 JAVA 客户端 API 的请求示例

以下是处理 Automation Manager 控制器 Java 客户端 API 请求示例，以便与流程 Automation Manager 控制器进行基本交互。对于可用的流程自动化管理器控制器 Java 客户端的完整列表，请 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution**，再导航到 **~/rhpam-**

7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller/src/main/java/org/kie/server/server/server/client/client.

创建与 KIE 服务器模板和 KIE 容器交互

您可以使用 REST 或 WebSocket Process Automation Manager 控制器客户端中的 **ServerTemplate** 和 **ContainerSpec** 服务来创建、分散和更新 KIE 服务器模板和 KIE 容器，以及启动和停止 KIE 容器，如下例所示。

创建并与 KIE Server 模板和 KIE 容器交互的请求示例

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.KieContainerStatus;
import org.kie.server.api.model.KieScannerStatus;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.controller.api.model.spec.*;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class RestTemplateContainerExample {

  private static final String URL = "http://localhost:8080/business-central/rest/controller";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

    private static KieServerControllerClient client;

    public static void main(String[] args) {
        KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
                                                                USER,
                                                                PASSWORD,
                                                                MarshallingFormat.JSON);
        // Create server template and KIE container, start and stop KIE container, and delete server
template
        ServerTemplate serverTemplate = createServerTemplate();
        ContainerSpec container = createContainer(serverTemplate);
        client.startContainer(container);
        client.stopContainer(container);
        client.deleteServerTemplate(serverTemplate.getId());
    }

    // Re-create and configure server template
    protected static ServerTemplate createServerTemplate() {
        ServerTemplate serverTemplate = new ServerTemplate();
        serverTemplate.setId("example-client-id");
        serverTemplate.setName("example-client-name");
        serverTemplate.setCapabilities(Arrays.asList(Capability.PROCESS.name(),
                                        Capability.RULE.name(),
                                        Capability.PLANNING.name()));

        client.saveServerTemplate(serverTemplate);

        return serverTemplate;
```

```
    }

    // Re-create and configure KIE containers
    protected static ContainerSpec createContainer(ServerTemplate serverTemplate){
        Map<Capability, ContainerConfig> containerConfigMap = new HashMap();

        ProcessConfig processConfig = new ProcessConfig("PER_PROCESS_INSTANCE",
"kieBase", "kieSession", "MERGE_COLLECTION");
        containerConfigMap.put(Capability.PROCESS, processConfig);

        RuleConfig ruleConfig = new RuleConfig(500l, KieScannerStatus.SCANNING);
        containerConfigMap.put(Capability.RULE, ruleConfig);

        ReleaseId releaseId = new ReleaseId("org.kie.server.testing", "stateless-session-kjar",
"1.0.0-SNAPSHOT");

        ContainerSpec containerSpec = new ContainerSpec("example-container-id", "example-client-
name", serverTemplate, releaseId, KieContainerStatus.STOPPED, containerConfigMap);
        client.saveContainerSpec(serverTemplate.getId(), containerSpec);

        return containerSpec;
    }
}
```

列出 KIE 服务器模板并指定连接超时(REST)

当您将 REST 协议用于流程 Automation Manager 控制器 Java 客户端 API 请求时，您可以提供自己的 **javax.ws.rs.core.Configuration** 规范来修改底层 REST 客户端 API，如连接超时。

返回服务器模板并指定连接超时的 REST 请求示例

```
import java.util.concurrent.TimeUnit;
import javax.ws.rs.core.Configuration;
import org.jboss.resteasy.client.jaxrs.ResteasyClientBuilder;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class RESTTimeoutExample {

  private static final String URL = "http://localhost:8080/business-central/rest/controller";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

  public static void main(String[] args) {

    // Specify connection timeout
    final Configuration configuration =
          new ResteasyClientBuilder()
              .establishConnectionTimeout(10,
                                          TimeUnit.SECONDS)
              .socketTimeout(60,
                             TimeUnit.SECONDS)
               .getConfiguration();
```

```
        KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
                                                                                           USER,
                                                                                           PASSWORD,
                                                                                           MarshallingFormat.JSON,
                                                                                           configuration);

        // Retrieve list of server templates
        final ServerTemplateList serverTemplateList = client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                         serverTemplateList.getServerTemplates().length,
                                         URL));
    }
}
```

列出 KIE Server 模板并指定事件通知(WebSocket)

当您将 WebSocket 协议用于进程 Automation Manager 控制器 Java 客户端 API 请求时，您可以根据客户端 API 连接到的特定 Process Automation Manager 控制器中的更改启用事件通知。例如，当 KIE Server 模板或实例在 Process Automation Manager 控制器中连接或更新时，您可以收到通知。

返回服务器模板的 WebSocket 请求示例，并指定事件通知

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.events.*;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;
import org.kie.server.controller.client.event.EventHandler;

public class WebSocketEventsExample {

    private static final String URL = "ws://localhost:8080/my-controller/websocket/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    public static void main(String[] args) {
        KieServerControllerClient client =
KieServerControllerClientFactory.newWebSocketClient(URL,
                                                                                           USER,
                                                                                           PASSWORD,
                                                                                           MarshallingFormat.JSON,
                                                                                           new TestEventHandler());

        // Retrieve list of server templates
        final ServerTemplateList serverTemplateList = client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                         serverTemplateList.getServerTemplates().length,
                                         URL));
        try {
            Thread.sleep(60 * 1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Set up event notifications
```

```
    static class TestEventHandler implements EventHandler {

        @Override
        public void onServerInstanceConnected(ServerInstanceConnected
serverInstanceConnected) {
            System.out.println("serverInstanceConnected = " + serverInstanceConnected);
        }

        @Override
        public void onServerInstanceDeleted(ServerInstanceDeleted serverInstanceDeleted) {
            System.out.println("serverInstanceDeleted = " + serverInstanceDeleted);
        }

        @Override
        public void onServerInstanceDisconnected(ServerInstanceDisconnected
serverInstanceDisconnected) {
            System.out.println("serverInstanceDisconnected = " + serverInstanceDisconnected);
        }

        @Override
        public void onServerTemplateDeleted(ServerTemplateDeleted serverTemplateDeleted) {
            System.out.println("serverTemplateDeleted = " + serverTemplateDeleted);
        }

        @Override
        public void onServerTemplateUpdated(ServerTemplateUpdated serverTemplateUpdated) {
            System.out.println("serverTemplateUpdated = " + serverTemplateUpdated);
        }

        @Override
        public void onServerInstanceUpdated(ServerInstanceUpdated serverInstanceUpdated) {
            System.out.println("serverInstanceUpdated = " + serverInstanceUpdated);
        }

        @Override
        public void onContainerSpecUpdated(ContainerSpecUpdated containerSpecUpdated) {
            System.out.println("onContainerSpecUpdated = " + containerSpecUpdated);
        }
    }
}
```

第 27 章 BUSINESS CENTRAL 进程的 FDO 流程流畅 API

Red Hat Process Automation Manager 提供了一个 ClusterClaim 过程流畅的 API，可让您使用工厂创建业务流程。您还可以手动验证使用流程流畅 API 创建的业务流程。进程流畅 API 在 **org.kie.api.fluent** 软件包中定义。

因此，您可以使用流畅的 API 过程在几行代码中创建业务流程，而不是使用forwarder2 XML 标准。

27.1. 带有 BPMN 进程流畅 API 的请求示例

以下示例包括用于与流程进行基本交互的 grub 处理 API 请求。如需了解更多示例，请 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Source Distribution**，再导航到 **~/rhpam-7.12.0-sources/src/droolsjbpm-knowledge-$VERSION/kie-api/src/main/java/org/kie/api/fluent**。

创建与 Business Central 业务流程交互

以下示例显示了基本业务流程，以及脚本任务、异常处理程序和变量：

创建并与 Business Central 业务流程交互的请求示例

```
Process process =
        // Create process builder
        factory.processBuilder(processId)
            // package and name
            .packageName("org.jbpm")
            .name("My process")
            // start node
            .startNode(1).name("Start").done()
            // Add variable of type string
            .variable(var("pepe", String.class))
            // Add exception handler
            .exceptionHandler(IllegalArgumentException.class, Dialect.JAVA,
"System.out.println(\"Exception\");")
            // script node in Java language that prints "action"
            .actionNode(2).name("Action")
            .action(Dialect.JAVA,
                "System.out.println(\"Action\");").done()
            // end node
            .endNode(3).name("End").done()
            // connections
            .connection(1,
                    2)
            .connection(2,
                    3)
            .build();
```

在本例中，会获取 **ProcessBuilderFactory** 引用，然后使用 **processBuilder (String processId)** 方法创建 **ProcessBuilder** 实例，它将与给定的进程 Id 关联。**ProcessBuilder** 实例允许您使用流畅的 API 构建所创建进程的定义。

业务流程由三个组件组成：

- header ： header 部分包含全局元素，如进程、导入和变量的名称。
  在上例中，标头包含进程的名称和版本，以及软件包名称。

- nodes： nodes 部分包含作为进程一部分的所有不同节点。

在上例中，通过调用 **startNode()**, **actionNode()**, 和 **endNode()** 方法，将节点添加到进程中。这些方法返回特定的 **NodeBuilder**，供您设置该节点的属性。在代码完成配置该特定节点后，**done （）** 方法会返回 **NodeContainerBuilder** 来添加更多节点（如有必要）。

- Connection: connections 部分将节点链接到创建流 chart。
在上例中，添加所有节点后，您必须通过在它们之间创建连接来进行连接。您可以调用 **connection （）** 方法来链接节点。

最后，您可以调用 **build （）** 方法并获取生成的进程定义。**build （）** 方法还会验证进程定义，并在进程定义无效时抛出异常。

27.2. 执行流程的请求示例

创建有效进程定义实例后，您可以使用公共和内部 KIE API 的组合来执行它。要执行进程，请创建一个 **Resource**，它用于创建 **KieBase**。通过使用 **KieBase**，您可以创建一个 **KieSession** 来执行进程。

以下示例使用 **ProcessBuilderFactory.toBytes** 进程创建 **ByteArrayResource** 资源。

执行进程的请求示例

```
// Build resource from Process
KieResources resources = ServiceRegistry.getInstance().get(KieResources.class);
Resource res = resources
                .newByteArrayResource(factory.toBytes(process))
                .setSourcePath("/tmp/processFactory.bpmn2"); // source path or target path must be
set to be added into kbase
// Build kie base from this resource using KIE API
KieServices ks = KieServices.Factory.get();
KieRepository kr = ks.getRepository();
KieFileSystem kfs = ks.newKieFileSystem();
kfs.write(res);
KieBuilder kb = ks.newKieBuilder(kfs);
kb.buildAll(); // kieModule is automatically deployed to KieRepository if successfully built.
KieContainer kContainer = ks.newKieContainer(kr.getDefaultReleaseId());
KieBase kbase = kContainer.getKieBase();
// Create kie session using KieBase
KieSessionConfiguration conf = ...;
Environment env = ....;
KieSession ksession = kbase.newKieSession(conf,env);
// execute process using same process Id that is used to obtain ProcessBuilder instance
ksession.startProcess(processId)
```

第 28 章 知识存储 BUSINESS CENTRAL 空间和项目的 REST API

Red Hat Process Automation Manager 提供了一个 知识库存储 REST API，您可以使用它们与红帽流程自动化管理器中的项目和空格交互，而无需使用 Business Central 用户界面。知识库存储是 Red Hat Process Automation Manager 中资产的工件存储库。通过此 API 支持，您可以促进和自动化 Business Central 项目和空格维护。

使用 知识库 REST API，您可以执行以下操作：

- 检索有关所有项目和空格的信息
- 创建、更新或删除项目和空格
- 构建、部署和测试项目
- 检索有关之前关于存储 REST API 请求或*作业*的信息

知识存储 REST API 请求需要以下组件：

身份验证

知识库存储 REST API 需要对用户角色 **rest-all** 进行 HTTP 基本身份验证或基于令牌的身份验证。要查看为您的 Red Hat Process Automation Manager 分发配置的用户角色，请导航到 **~/$SERVER_HOME/standalone/configuration/application-roles.properties** 和 **~/application-users.properties**。
要添加具有 **rest-all** 角色的用户，请导航到 **~/$SERVER_HOME/bin**，再运行以下命令：

```
$ ./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=['rest-all'])"
```

有关用户角色和 Red Hat Process Automation Manager 安装选项的更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装*。

HTTP 标头

知识库存储 REST API 需要为 API 请求使用以下 HTTP 标头：

- **接受**: 请求客户端接受的数据格式：
  - **application/json** (JSON)
- **Content-Type**: **POST** 或 **PUT** API 请求数据的数据格式：
  - **application/json** (JSON)

HTTP 方法

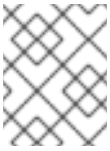
知识库存储 REST API 支持以下 API 请求 HTTP 方法：

- **GET** ：从指定的资源端点检索指定的信息
- **POST** ：创建或更新资源
- **PUT** ：更新资源

- **DELETE** ：删除资源

基本 URL

知识库 REST API 请求的基本 URL 是 **http://SERVER:PORT/business-central/rest/**，如 **http://localhost:8080/business-central/rest/**。



注意

知识库存储的 REST API 基础 URL 以及内置在 Business Central 中的流程自动化管理器控制器是相同的，因为两者都被视为 Business Central REST 服务的一部分。

Endpoints

了解存储 REST API 端点，如指定空间的 **/spaces/{spaceName}**，是您附加到 知识库 REST API 基础 URL 中的 URI，以访问 Red Hat Process Automation Manager 中的相应资源或资源类型。

/spaces/{spaceName} 端点的请求 URL 示例

http://localhost:8080/business-central/rest/spaces/MySpace

请求数据

知识库 REST API 中的 HTTP **POST** 请求可能需要 JSON 请求正文，并附带请求。

POST 请求 URL 和 JSON 请求正文数据示例

http://localhost:8080/business-central/rest/spaces/MySpace/projects

```
{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```

28.1. 使用 REST 客户端或 CURL 工具使用 知识库存储 REST API 发送请求

知识库存储 REST API 可让您与 Red Hat Process Automation Manager 中的项目和空格交互，而无需使用 Business Central 用户界面。您可以使用任何 REST 客户端或 curl 工具发送知识库 REST API 请求。

先决条件

- Business Central 已安装并运行。
- 您有 **rest-all** 用户角色对 Business Central 的访问权限。

流程

1. 识别您要向其发送 请求的相关 API 端点，如 **[GET] /spaces** 以检索 Business Central 中的空格。
2. 在 REST 客户端或 curl 工具中，为对 **/space** 的 **GET** 请求输入以下组件。根据您的用例调整任何请求详情。
   对于 REST 客户端：

- **身份验证** ：使用 **rest-all** 角色输入 Business Central 用户的用户名和密码。
- **HTTP Headers** ：设置以下标头：
  - **接受:application/json**
- **HTTP 方法** ：设置为 **GET**。
- **URL** ：输入知识库 REST API 基本 URL 和端点，如 **http://localhost:8080/business-central/rest/spaces**。

对于 curl 工具：

- **-** U ：输入 Business Central 用户的用户名和密码，以及 **rest-all** 角色。
- **-** h : 设置以下标头：
  - **接受:application/json**
- **-x** ：设置为 **GET**。
- **URL** ：输入知识库 REST API 基本 URL 和端点，如 **http://localhost:8080/business-central/rest/spaces**。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
"http://localhost:8080/business-central/rest/spaces"
```

3. 执行请求并查看 KIE 服务器响应。
服务器响应示例(JSON)：


```
[
 {
  "name": "MySpace",
  "description": null,
  "projects": [
   {
    "name": "Employee_Rostering",
    "spaceName": "MySpace",
    "groupId": "employeerostering",
    "version": "1.0.0-SNAPSHOT",
    "description": "Employee rostering problem optimisation using Planner. Assigns
employees to shifts based on their skill.",
    "publicURIs": [
     {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
     },
     {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
     }
    ]
   },
   {
    "name": "Mortgage_Process",
    "spaceName": "MySpace",
```

```
        "groupId": "mortgage-process",
        "version": "1.0.0-SNAPSHOT",
        "description": "Getting started loan approval process in BPMN2, decision table, business
rules, and forms.",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
          }
        ]
      }
    ],
    "owner": "admin",
    "defaultGroupId": "com.myspace"
  },
  {
    "name": "MySpace2",
    "description": null,
    "projects": [
      {
        "name": "IT_Orders",
        "spaceName": "MySpace",
        "groupId": "itorders",
        "version": "1.0.0-SNAPSHOT",
        "description": "Case Management IT Orders project",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-IT_Orders-1"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-IT_Orders-1"
          }
        ]
      }
    ],
    "owner": "admin",
    "defaultGroupId": "com.myspace"
  }
]
```


4. 在 REST 客户端或 curl 实用程序中，发送另外一个带有以下组件的 API 请求，它向 **/spaces/{spaceName}/projects** 发送一个 **POST** 请求，以便在一个空间内创建一个项目。根据您的用例调整任何请求详情。
   对于 REST 客户端：

   - **身份验证** ：使用 **rest-all** 角色输入 Business Central 用户的用户名和密码。
   - **HTTP Headers** ：设置以下标头：
     - **接受**:**application/json**

- **接受语言**:**en-US**
- **Content-Type**:**application/json**

- **HTTP 方法** ：设置为 **POST**。
- **URL** ：输入知识库 REST API 基本 URL 和端点，如 **http://localhost:8080/business-central/rest/spaces/MySpace/projects**。
- **请求正文** ：添加带有新项目标识数据的 JSON 请求正文：

```
{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```

对于 curl 工具：

- **-** U ：输入 Business Central 用户的用户名和密码，以及 **rest-all** 角色。
- **-** h ：设置以下标头：
  - **接受**:**application/json**
  - **接受语言** ：**en-US** （如果未定义，则 JVM 中的默认区域设置会反映）
  - **Content-Type**:**application/json**
- **-x** ：设置为 **POST**。
- **URL** ：输入知识库 REST API 基本 URL 和端点，如 **http://localhost:8080/business-central/rest/spaces/MySpace/projects**。
- **-d** ：添加一个 JSON 请求正文或文件(**@file.json**)，带有新项目的标识数据：

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Accept-Language: en-US" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/spaces/MySpace/projects" -d "{ \"name\": \"Employee_Rostering\", \"groupId\": \"employeerostering\", \"version\": \"1.0.0-SNAPSHOT\", \"description\": \"Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.\"}"
```

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Accept-Language: en-US" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/spaces/MySpace/projects" -d @my-project.json
```

5. 执行请求并查看 KIE 服务器响应。
服务器响应示例(JSON)：

```
{
  "jobId": "1541017411591-6",
  "status": "APPROVED",
  "spaceName": "MySpace",
```

```
  "projectName": "Employee_Rostering",
  "projectGroupId": "employeerostering",
  "projectVersion": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```


如果您遇到请求错误，请检查返回的错误消息并相应地调整您的请求。

28.2. 支持的知识库文章 REST API 端点

知识库存储 REST API 提供了在红帽流程自动化管理器中管理空格和项目的端点，以及检索有关之前知识存储 REST API 请求或 *作业* 的信息。

28.2.1. space

知识库存储 REST API 支持以下端点来管理 Business Central 中的空格。知识库存储 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要对 **rest-all** 用户角色进行 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /spaces

返回 Business Central 中的所有空格。

服务器响应示例(JSON)


```
[
  {
    "name": "MySpace",
    "description": null,
    "projects": [
      {
        "name": "Employee_Rostering",
        "spaceName": "MySpace",
        "groupId": "employeerostering",
        "version": "1.0.0-SNAPSHOT",
        "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
          }
        ]
      },
      {
        "name": "Mortgage_Process",
        "spaceName": "MySpace",
        "groupId": "mortgage-process",
        "version": "1.0.0-SNAPSHOT",
        "description": "Getting started loan approval process in BPMN2, decision table, business rules, and forms.",
```

```
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
          }
        ]
      }
    ],
    "owner": "admin",
    "defaultGroupId": "com.myspace"
  },
  {
    "name": "MySpace2",
    "description": null,
    "projects": [
      {
        "name": "IT_Orders",
        "spaceName": "MySpace",
        "groupId": "itorders",
        "version": "1.0.0-SNAPSHOT",
        "description": "Case Management IT Orders project",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-IT_Orders-1"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-IT_Orders-1"
          }
        ]
      }
    ],
    "owner": "admin",
    "defaultGroupId": "com.myspace"
  }
]
```


[GET] /spaces/{spaceName}

返回有关指定空间的信息。

表 28.1. 请求参数

名称	描述	类型	要求
spaceName	要检索的空间的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "name": "MySpace",
  "description": null,
  "projects": [
    {
      "name": "Mortgage_Process",
      "spaceName": "MySpace",
      "groupId": "mortgage-process",
      "version": "1.0.0-SNAPSHOT",
      "description": "Getting started loan approval process in BPMN2, decision table, business rules,
and forms.",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
        }
      ]
    },
    {
      "name": "Employee_Rostering",
      "spaceName": "MySpace",
      "groupId": "employeerostering",
      "version": "1.0.0-SNAPSHOT",
      "description": "Employee rostering problem optimisation using Planner. Assigns employees to
shifts based on their skill.",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
        }
      ]
    },
    {
      "name": "Evaluation_Process",
      "spaceName": "MySpace",
      "groupId": "evaluation",
      "version": "1.0.0-SNAPSHOT",
      "description": "Getting started Business Process for evaluating employees",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-Evaluation_Process"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-Evaluation_Process"
        }
      ]
```

```
    },
    {
      "name": "IT_Orders",
      "spaceName": "MySpace",
      "groupId": "itorders",
      "version": "1.0.0-SNAPSHOT",
      "description": "Case Management IT Orders project",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-IT_Orders"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-IT_Orders"
        }
      ]
    }
  ],
  "owner": "admin",
  "defaultGroupId": "com.myspace"
}
```

[POST] /spaces

在 Business Central 中创建一个空格。

表 28.2. 请求参数

名称	描述	类型	要求
正文（body）	**名称**、**描述**、**所有者**、**defaultGroupId** 以及新空间的任何其他组件	请求正文	必填

请求正文示例(JSON)

```
{
  "name": "NewSpace",
  "description": "My new space.",
  "owner": "admin",
  "defaultGroupId": "com.newspace"
}
```

服务器响应示例(JSON)

```
{
  "jobId": "1541016978154-3",
  "status": "APPROVED",
  "spaceName": "NewSpace",
  "owner": "admin",
  "defaultGroupId": "com.newspace",
  "description": "My new space."
}
```

[PUT] /spaces

更新 Business Central 中空间 **的说明、所有者和 默**认组 ID。

请求正文示例(JSON)

```
{
  "name": "MySpace",
  "description": "This is updated description",
  "owner": "admin",
  "defaultGroupId": "com.updatedGroupId"
}
```

服务器响应示例(JSON)

```
{
  "jobId": "1592214574454-1",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "owner": "admin",
  "defaultGroupId": "com.updatedGroupId",
  "description": "This is updated description"
}
```

[DELETE] /spaces/{spaceName}

从 Business Central 中删除指定的空间。

表 28.3. 请求参数

名称	描述	类型	要求
spaceName	要删除的空间的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId": "1541127032997-8",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "owner": "admin",
  "description": "My deleted space.",
  "repositories": null
}
```

28.2.2. 项目

知识库存储 REST API 支持以下端点，以便在 Business Central 中管理、构建和部署项目。知识库存储 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要对 **rest-all** 用户角色进行 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /spaces/{spaceName}/projects

返回指定空间中的项目。

表 28.4. 请求参数

名称	描述	类型	要求
spaceName	正在检索项目的空间的名称	字符串	必填

服务器响应示例(JSON)


```
[
  {
    "name": "Mortgage_Process",
    "spaceName": "MySpace",
    "groupId": "mortgage-process",
    "version": "1.0.0-SNAPSHOT",
    "description": "Getting started loan approval process in BPMN2, decision table, business rules, and forms.",
    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
      },
      {
        "protocol": "ssh",
        "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
      }
    ]
  },
  {
    "name": "Employee_Rostering",
    "spaceName": "MySpace",
    "groupId": "employeerostering",
    "version": "1.0.0-SNAPSHOT",
    "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
      },
      {
        "protocol": "ssh",
        "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
      }
    ]
  },
  {
    "name": "Evaluation_Process",
    "spaceName": "MySpace",
    "groupId": "evaluation",
    "version": "1.0.0-SNAPSHOT",
    "description": "Getting started Business Process for evaluating employees",
```

```
    "publicURIs": [
     {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-Evaluation_Process"
     },
     {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-Evaluation_Process"
     }
    ]
   },
   {
    "name": "IT_Orders",
    "spaceName": "MySpace",
    "groupId": "itorders",
    "version": "1.0.0-SNAPSHOT",
    "description": "Case Management IT Orders project",
    "publicURIs": [
     {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-IT_Orders"
     },
     {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-IT_Orders"
     }
    ]
   }
]
```


[GET] /spaces/{spaceName}/projects/{projectName}

返回指定空间中指定项目的信息。

表 28.5. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	要检索的项目的名称	字符串	必填

服务器响应示例(JSON)


```
{
  "name": "Employee_Rostering",
  "spaceName": "MySpace",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
  "publicURIs": [
```

```
    {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
    },
    {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
    }
  ]
}
```

[POST] /spaces/{spaceName}/projects

在指定空间中创建项目。

表 28.6. 请求参数

名称	描述	类型	要求
spaceName	在其中创建新项目的空间的名称	字符串	必填
正文（body）	新项目的**名称**,**groupId**,**版本**,**description**, 和新项目的任何其他组件	请求正文	必填

请求正文示例(JSON)

```
{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```

服务器响应示例(JSON)

```
{
  "jobId": "1541017411591-6",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "projectName": "Employee_Rostering",
  "projectGroupId": "employeerostering",
  "projectVersion": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```

[DELETE] /spaces/{spaceName}/projects/{projectName}

从指定的空间删除指定项目。

表 28.7. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	要删除的项目名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId": "1541128617727-10",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/git/clone

将项目克隆到指定 Git 地址的指定空间中。

表 28.8. 请求参数

名称	描述	类型	要求
spaceName	要克隆项目的空间的名称	字符串	必填
正文（body）	要克隆 **的项目的名称**、**描述**、**描述** 和 Git 存储库 **userName**、**密码**，以及 **gitURL**。	请求正文	必填

请求正文示例(JSON)

```
{
  "name": "Employee_Rostering",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
  "userName": "baAdmin",
  "password": "password@1",
  "gitURL": "git://localhost:9418/MySpace/example-Employee_Rostering"
}
```

服务器响应示例(JSON)

```
{
  "jobId": "1541129488547-13",
  "status": "APPROVED",
  "cloneProjectRequest": {
    "name": "Employee_Rostering",
    "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
```

```
    "userName": "baAdmin",
    "password": "password@1",
    "gitURL": "git://localhost:9418/MySpace/example-Employee_Rostering"
  },
  "spaceName": "MySpace2"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/compile

在指定空间中编译指定项目（等同于 **mvn compile**）。

表 28.9. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	要编译的项目的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId": "1541128617727-10",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/test

在指定空间中测试指定项目（等同于 **mvn test**）。

表 28.10. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	要测试的项目的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId": "1541132591595-19",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/install

在指定空间中安装指定项目（等同于 **mvn install**）。

表 28.11. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	要安装的项目名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId": "1541132668987-20",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/deploy

在指定空间中部署指定项目（等同于 **mvn deploy**）。

表 28.12. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	要部署的项目的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId": "1541132816435-21",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

28.2.3. 作业(API 请求)

除了返回的请求详情外，知识库 REST API 中的所有 **POST** 和 **DELETE** 请求都会返回一个与每个请求关联的作业 ID。您可以使用作业 ID 来查看请求状态或删除发送的请求。

知识存储 REST API 请求或 *作业* 可具有以下状态：

表 28.13. 作业状态(API 请求状态)

状态	描述
已接受	请求已被接受，正在处理。
BAD_REQUEST	请求包含不正确的内容，且不接受。
RESOURCE_NOT_EXIST	请求的资源(path)不存在。
DUPLICATE_RESOURCE	资源已存在。
SERVER_ERROR	KIE Server 中出现错误。
SUCCESS	请求成功完成。
FAIL	请求失败。
已批准	申请已批准。
已拒绝	请求被拒绝。
GOD	由于以下原因之一，无法找到请求的作业 ID： ● 请求被明确删除。 ● 请求已完成，已从状态缓存中删除。在缓存达到其最大容量后，请求会从状态缓存中删除。 ● 请求永不会存在。

知识库文章 REST API 支持以下端点来检索或删除发送的 API 请求。知识库存储 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要对 **rest-all** 用户角色进行 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /jobs/{jobId}

返回指定作业的状态（之前发送的 API 请求）。

表 28.14. 请求参数

名称	描述	类型	要求
jobId	要检索的作业 ID （例如：**1541010216919-1**）	字符串	必填

服务器响应示例(JSON)

```
{
  "status": "SUCCESS",
  "jobId": "1541010216919-1",
  "result": null,
  "lastModified": 1541010218352,
  "detailedResult": [
    "level:INFO, path:null, text:Build of module 'Mortgage_Process' (requested by system) completed.\n Build: SUCCESSFUL"
  ]
}
```

[DELETE] /jobs/{jobId}

删除指定的作业（之前发送的 API 请求）。如果尚未处理作业，此请求将从作业队列中删除作业。此请求不会取消或停止持续作业。

表 28.15. 请求参数

名称	描述	类型	要求
jobId	要删除的作业的 ID （例如：**1541010216919-1**）	字符串	必填

服务器响应示例(JSON)

```
{
  "status": "GONE",
  "jobId": "1541010216919-1",
  "result": null,
  "lastModified": 1541132054916,
  "detailedResult": [
    "level:INFO, path:null, text:Build of module 'Mortgage_Process' (requested by system) completed.\n Build: SUCCESSFUL"
  ]
}
```

28.2.4. 分支

知识库存储 REST API 支持以下端点来管理 Business Central 中的分支。知识库存储 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要对 **rest-all** 用户角色进行 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /spaces/{spaceName}/projects/{projectName}/branches

返回指定项目和空间中的所有分支。

表 28.16. 请求参数

名称	描述	类型	要求
spaceName	正在检索项目的空间的名称	字符串	必填

名称	描述	类型	要求
projectName	正在检索分支的项目的名称	字符串	必填

服务器响应示例(JSON)

```
[
  {
    "name":"master"
  }
]
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches

在指定项目和空间中添加指定的分支。

表 28.17. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	需要在其中创建新分支的项目的名称	字符串	必填
正文（body）	项目的 **newBranchName** 和 **baseBranchName**	请求正文	必填

请求正文示例(JSON)

```
{
  "newBranchName":  "branch01",
  "baseBranchName": "master"
}
```

服务器响应示例(JSON)

```
{
    "jobId":          "1576175811141-3",
    "status":         "APPROVED",
    "spaceName":      "Space123",
    "projectName":    "ProjABC",
    "newBranchName":  "b1",
    "baseBranchName": "master",
    "userIdentifier": "bc"
}
```

[DELETE] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}

删除指定项目和空间中的指定分支。

表 28.18. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	分支所在的项目的名称	字符串	必填
branchName	要删除的分支的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId":         "1576175811421-5",
  "status":        "APPROVED",
  "spaceName":     "Space123",
  "projectName":   "ProjABC",
  "branchName":    "b1",
  "userIdentifier": "bc"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/compile

在指定项目和空间中编译指定的分支。如果没有指定 **branchName**，则请求将应用到 master 分支。

表 28.19. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	分支所在的项目的名称	字符串	必填
branchName	要编译的分支的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "jobId":      "1576175811233-4",
  "status":     "APPROVED",
  "spaceName":  "Space123",
```

```
    "projectName": "ProjABC",
    "branchName":  "b1",
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/install

在指定的项目和空间中安装指定的分支。如果没有指定 **branchName**，则请求将应用到 master 分支。

表 28.20. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	分支所在的项目的名称	字符串	必填
branchName	要安装的分支的名称	字符串	必填

服务器响应示例(JSON)

```
{
    "jobId":       "1576175811233-4",
    "status":      "APPROVED",
    "spaceName":   "Space123",
    "projectName": "ProjABC",
    "branchName":  "b1",
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/test

在指定的项目和空间中测试指定的分支。如果没有指定 **branchName**，则请求将应用到 master 分支。

表 28.21. 请求参数

名称	描述	类型	要求
spaceName	项目所在空间的名称	字符串	必填
projectName	分支所在的项目的名称	字符串	必填
branchName	要测试的分支的名称	字符串	必填

服务器响应示例(JSON)

-

```
{
   "jobId":       "1576175811233-4",
   "status":      "APPROVED",
   "spaceName":   "Space123",
   "projectName": "ProjABC",
   "branchName":  "b1",
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/deploy

在指定的项目和空间中部署指定的分支。如果没有指定 **branchName**，则请求将应用到 master 分支。

表 28.22. 请求参数

名称	描述	类型	要求
spaceName	项目所在的空间的名称	字符串	必填
projectName	分支所在的项目的名称	字符串	必填
branchName	要部署的分支的名称	字符串	必填

服务器响应示例(JSON)

```
{
   "jobId":       "1576175811233-4",
   "status":      "APPROVED",
   "spaceName":   "Space123",
   "projectName": "ProjABC",
   "branchName":  "b1",
}
```

第 29 章 BUSINESS CENTRAL 组、角色和用户的安全管理 REST API

Red Hat Process Automation Manager 提供了一个安全管理 REST API，您可以在不使用 Business Central 用户界面的情况下管理红帽流程自动化管理器中的组、角色和用户。通过此 API 支持，您可以促进和自动化对 Business Central 组、角色、用户和授予的权限管理。

使用安全管理 REST API，您可以执行以下操作：

- 检索有关所有组、角色、用户及其授予权限的信息
- 创建、更新或删除组和用户
- 更新组、角色和用户的授予的权限
- 检索有关分配给用户的组和角色的信息

安全管理 REST API 请求需要以下组件：

身份验证

安全管理 REST API 需要 HTTP 基本身份验证或基于令牌的身份验证，用于用户角色 **admin**。要查看为您的 Red Hat Process Automation Manager 分发配置的用户角色，请导航到 **~/$SERVER_HOME/standalone/configuration/application-roles.properties** 和 **~/application-users.properties**。
要添加具有 **admin** 角色的用户，请导航到 **~/$SERVER_HOME/bin** 并运行以下命令：

```
$ ./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=['admin'])"
```

有关用户角色和 Red Hat Process Automation Manager 安装选项的更多信息，请参阅 *规划 Red Hat Process Automation Manager 安装*。

HTTP 标头

安全管理 REST API 需要以下 HTTP 标头用于 API 请求：

- **接受**: 请求客户端接受的数据格式：
  - **application/json** (JSON)
- **Content-Type**: **POST** 或 **PUT** API 请求数据的数据格式：
  - **application/json** (JSON)

HTTP 方法

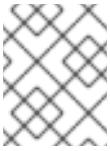
安全管理 REST API 支持以下 API 请求 HTTP 方法：

- **GET** ：从指定的资源端点检索指定的信息
- **POST** ：创建或更新资源
- **PUT** ：更新资源

- **DELETE** ：删除资源

基本 URL

安全管理 REST API 请求的基本 URL 是 **http://SERVER:PORT/business-central/rest/**，如 **http://localhost:8080/business-central/rest/**。



注意

内置于 Business Central 中的安全管理、知识存储和流程自动化管理器控制器的 REST API 基础 URL 相同，因为它们都被视为 Business Central REST 服务的一部分。

Endpoints

安全管理 REST API 端点（如指定用户的 **/users/{userName}** ）是您附加到安全管理 REST API 基础 URL 的 URI，以访问 Red Hat Process Automation Manager 中的对应资源或资源类型。

/users/{userName} 端点的请求 URL 示例

http://localhost:8080/business-central/rest/users/newUser

请求数据

安全管理 REST API 中的 HTTP **POST** 请求可能需要 JSON 请求正文，并附带请求。

POST 请求 URL 和 JSON 请求正文数据示例

http://localhost:8080/business-central/rest/users/newUser/groups

```
[
  "newGroup"
]
```

29.1. 使用 REST 客户端或 CURL 工具使用安全管理 REST API 发送请求

安全管理 REST API 可让您管理红帽流程自动化管理器中的组、角色和用户，而无需使用 Business Central 用户界面。您可以使用任何 REST 客户端或 curl 工具发送安全管理 REST API 请求。

先决条件

- Business Central 已安装并运行。
- 您有 **管理** 用户角色对 Business Central 的访问权限。

流程

1. 识别您要向其发送 请求的相关 API 端点，如 **[GET] /groups** 来检索 Business Central 中的组。
2. 在 REST 客户端或 curl 实用程序中，输入到 **/groups** 的 **GET** 请求的以下组件。根据您的用例调整任何请求详情。
   对于 REST 客户端：
   - **身份验证** ：使用 **admin** 角色输入 Business Central 用户的用户名和密码。
   - **HTTP Headers** ：设置以下标头：

- **接受**:**application/json**

- **HTTP 方法** ：设置为 **GET**。
- **URL** ：输入安全管理 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/groups**。

对于 curl 工具：

- **-** U ：使用 **admin** 角色输入 Business Central 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** ：输入安全管理 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/groups**。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
"http://localhost:8080/business-central/rest/groups"
```

3. 执行请求并查看 KIE 服务器响应。
服务器响应示例(JSON)：

```
[
  {
 "group1"
 },
 {
  "group2"
 }
]
```

4. 在 REST 客户端或 curl 实用程序中，发送另外一个带有以下组件的 API 请求，它向 **/users/{userName}/groups** 发送一个 **POST** 请求，以便更新为一个用户分配的组。根据您的用例调整任何请求详情。
对于 REST 客户端：

- **身份验证** ：使用 **admin** 角色输入 Business Central 用户的用户名和密码。
- **HTTP Headers** ：设置以下标头：
  - **接受**:**application/json**
  - **Content-Type**:**application/json**
- **HTTP 方法** ：设置为 **POST**。
- **URL** ：输入安全管理 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/users/newUser/groups**。
- **请求正文** ：添加带有新组的标识数据的 JSON 请求正文：

```
[
  "newGroup"
]
```

对于 curl 工具：

- **-U** ：使用 **admin** 角色输入 Business Central 用户的用户名和密码。
- **-h** ：设置以下标头：
  - **接受:application/json**
  - **Content-Type:application/json**
- **-x** ：设置为 **POST**。
- **URL** ：输入安全管理 REST API 基础 URL 和端点，如 **http://localhost:8080/business-central/rest/users/newUser/groups**。
- **-d** ：添加一个 JSON 请求正文或文件(**@file.json**)，带有新组的标识数据：

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/users/newUser/groups" -d "["newGroup"]"
```

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/users/newUser/groups" -d @user-groups.json
```

5. 执行请求并查看 KIE 服务器响应。
   服务器响应示例(JSON)：

```
{
  "status": "OK",
  "message": "Groups [newGroup] are assigned successfully to user wbadmin"
}
```

如果您遇到请求错误，请检查返回的错误消息并相应地调整您的请求。

29.2. 支持的安全管理 REST API 端点

安全管理 REST API 提供端点，用于管理 Business Central 中的组、角色、用户和权限。它包括管理员也可以使用 Business Central 中的 **安全管理** 页面执行的安全性和权限管理任务。

29.2.1. 组

安全管理 REST API 支持以下端点来管理 Business Central 中的组。安全管理 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要 **admin** 用户角色的 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /groups

返回 Business Central 中的所有组。

服务器响应示例(JSON)

```
[
  {
  "group1"
 },
 {
  "group2"
 }
]
```

[POST] /groups

在 Business Central 中创建组。组必须至少分配了一个用户。

表 29.1. 请求参数

名称	描述	类型	要求
正文（body）	分配给新组的组和用户的名称	请求正文	必填

请求正文示例(JSON)

```
{
  "name": "groupName",
  "users": [
    "userNames"
  ]
}
```

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "Group newGroup is created successfully."
}
```

[DELETE] /groups/{groupName}

从 Business Central 中删除指定的组。

表 29.2. 请求参数

名称	描述	类型	要求
groupName	要删除的组名称	字符串	必填

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "Group newGroup is deleted successfully."
}
```

```
}
```

29.2.2. 角色

安全管理 REST API 支持以下在 Business Central 中管理角色的端点。安全管理 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要 **admin** 用户角色的 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /roles

返回 Business Central 中的所有角色。

服务器响应示例(JSON)

```
[
  {
    "name": "process-admin"
  },
  {
    "name": "manager"
  },
  {
    "name": "admin"
  }
]
```

29.2.3. 用户

安全管理 REST API 支持以下端点来管理 Business Central 中的用户。安全管理 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要 **admin** 用户角色的 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /users

返回 Business Central 中的所有用户。

服务器响应示例(JSON)

```
[
    "newUser",
    "user1",
    "user2",
]
```

[GET] /users/{userName}/groups

返回分配给指定用户的所有组。

表 29.3. 请求参数

名称	描述	类型	要求
userName	检索分配的组的用户的用户名	字符串	必填

服务器响应示例(JSON)

```
[
  {
  "group1"
 },
 {
  "group2"
 }
]
```

[GET] /users/{userName}/roles

返回分配给指定用户的所有角色。

表 29.4. 请求参数

名称	描述	类型	要求
userName	检索分配角色的用户的名称	字符串	必填

服务器响应示例(JSON)

```
[
  {
    "name": "process-admin"
  },
  {
    "name": "manager"
  },
  {
    "name": "admin"
  }
]
```

[POST] /users

创建带有指定角色和组的指定用户。

请求正文示例(JSON)

```
{
  "name": "newUser",
  "roles": [
    "admin",
    "developer"
  ],
  "groups": [
    "group1",
    "group2"
  ]
}
```

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "User newUser is created successfully."
}
```

[Post] /users/{userName}/changePassword

更改指定用户的密码。

表 29.5. 请求参数

名称	描述	类型	要求
userName	更改密码的用户名称	字符串	必填

request 命令示例

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/users/newUser/changePassword" -d newpassword
```

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "Password for newUser has been updated successfully."
}
```

[DELETE] /users/{userName}

从 Business Central 删除指定用户。

表 29.6. 请求参数

名称	描述	类型	要求
userName	要删除的用户的名称	字符串	必填

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "User newUser is deleted successfully."
}
```

[POST] /users/{userName}/groups

覆盖分配给具有新组的指定用户的现有组。

表 29.7. 请求参数

名称	描述	类型	要求
userName	正在更新组的用户名称	字符串	必填

请求正文示例(JSON)

```
[
  "newGroup"
]
```

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "Groups [newGroup] are assigned successfully to user wbadmin"
}
```

[POST] /users/{userName}/roles

覆盖分配给具有新角色的指定用户的现有角色。

表 29.8. 请求参数

名称	描述	类型	要求
userName	正在更新角色的用户的名称	字符串	必填

请求正文示例(JSON)

```
[
  "admin"
]
```

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "Roles [admin] are assigned successfully to user wbadmin"
}
```

29.2.4. 权限

安全管理 REST API 支持以下端点，以管理授予 Business Central 中的组、角色和用户的权限。安全管理 REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。所有请求都需要 **admin** 用户角色的 HTTP 基本身份验证或基于令牌的身份验证。

[GET] /groups/{groupName}/permissions

返回授予指定组的所有权限。

表 29.9. 请求参数

名称	描述	类型	要求
groupName	用于检索权限的组名称	字符串	必填

服务器响应示例(JSON)


```
{
 "homePage": "HomePerspective",
 "priority": -10,
 "project": {
  "read": {
   "access": false,
   "exceptions": []
  },

 },
 "spaces": {
  "read": {
   "access": true,
   "exceptions": [
    "MySpace"
   ]
  },
 },
 "editor": {
  "read": {
   "access": false,
   "exceptions": [
    "GuidedDecisionTreeEditorPresenter"
   ]
  },
  "create": null,
  "update": null,
  "delete": null,
  "build": null
 },
 "pages": {
  "read": {
   "access": true,
   "exceptions": []
  },
  "build": null
 },
 "workbench": {
  "editDataObject": false,
  "plannerAvailable": false,
  "editGlobalPreferences": false,
  "editProfilePreferences": false,
  "accessDataTransfer": false,
  "jarDownload": true,
```

```
  "editGuidedDecisionTableColumns": true
 }
}
```

[GET] /roles/{roleName}/permissions

返回授予指定角色的所有权限。

表 29.10. 请求参数

名称	描述	类型	要求
roleName	用于检索权限的角色名称	字符串	必填

服务器响应示例(JSON)

```
{
 "homePage": "HomePerspective",
 "priority": -10,
 "project": {
  "read": {
   "access": false,
   "exceptions": []
  },

 },
 "spaces": {
  "read": {
   "access": true,
   "exceptions": [
    "MySpace"
   ]
  },
 },
 "editor": {
  "read": {
   "access": false,
   "exceptions": [
    "GuidedDecisionTreeEditorPresenter"
   ]
  },
  "create": null,
  "update": null,
  "delete": null,
  "build": null
 },
 "pages": {
  "read": {
   "access": true,
   "exceptions": []
  },
  "build": null
 },
 "workbench": {
```

```
  "editDataObject": false,
  "plannerAvailable": false,
  "editGlobalPreferences": false,
  "editProfilePreferences": false,
  "accessDataTransfer": false,
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
 }
}
```

[GET] /users/{userName}/permissions

返回授予指定用户的所有权限。

表 29.11. 请求参数

名称	描述	类型	要求
userName	检索权限的用户名称	字符串	必填

服务器响应示例(JSON)

```
{
 "homePage": null,
 "priority": null,
 "project": {
  "read": {
   "access": false,
   "exceptions": []
  },

 },
 "spaces": {
  "read": {
   "access": true,
   "exceptions": [
    "MySpace"
   ]
  },
 },
 "editor": {
  "read": {
   "access": false,
   "exceptions": [
    "GuidedDecisionTreeEditorPresenter"
   ]
  },
  "create": null,
  "update": null,
  "delete": null,
  "build": null
 },
 "pages": {
  "read": {
```

```
    "access": true,
    "exceptions": []
   },
   "build": null
  },
  "workbench": {
   "editDataObject": false,
   "plannerAvailable": false,
   "editGlobalPreferences": false,
   "editProfilePreferences": false,
   "accessDataTransfer": false,
   "jarDownload": true,
   "editGuidedDecisionTableColumns": true
  }
 }
```

[post] /groups/{groupName}/permissions

更新指定组的权限。

表 29.12. 请求参数

名称	描述	类型	要求
groupName	正在更新权限的组名称	字符串	必填

请求正文示例(JSON)

```
{
  "homepage": "HomePerspective",
  "priority": 10,
  "pages": {
    "create": true,
    "read": false,
    "delete": false,
    "update": false,
    "exceptions": [
      {
        "name": "HomePerspective",
        "permissions": {
          "read": true
        }
      }
    ]
  },
  "project": {
    "create": true,
    "read": true,
    "delete": false,
    "update": false,
    "Build": false
  },
  "spaces": {
```

```
    "create": true,
    "read": true,
    "delete": false,
    "update": false
  },
  "editor": {
    "read": true
  },
  "workbench": {
    "editDataObject": true,
    "plannerAvailable": true,
    "editGlobalPreferences": true,
    "editProfilePreferences": true,
    "accessDataTransfer": true,
    "jarDownload": true,
    "editGuidedDecisionTableColumns": true
  }
}
```

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "Group newGroup permissions are updated successfully."
}
```

[post] /roles/{roleName}/permissions

更新指定角色的权限。

表 29.13. 请求参数

名称	描述	类型	要求
roleName	正在更新权限的角色名称	字符串	必填

请求正文示例(JSON)

```
{
 "homepage": "HomePerspective",
 "priority": 10,
 "pages": {
  "create": true,
  "read": false,
  "delete": false,
  "update": false,
  "exceptions": [{
   "name": "HomePerspective",
   "permissions": {
    "read": true
   }
  }]
 },
 "project": {
```

```
  "create": true,
  "read": true,
  "delete": false,
  "update": false,
  "Build": false
 },
 "spaces": {
  "create": true,
  "read": true,
  "delete": false,
  "update": false
 },
 "editor": {
  "read": true
 },
 "workbench": {
  "editDataObject": true,
  "plannerAvailable": true,
  "editGlobalPreferences": true,
  "editProfilePreferences": true,
  "accessDataTransfer": true,
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
 }
}
```

服务器响应示例(JSON)

```
{
  "status": "OK",
  "message": "Role newRole permissions are updated successfully."
}
```

29.2.4.1. Business Central 中支持的权限

以下是 Red Hat Process Automation Manager 中的可用权限。管理员使用这些权限来允许在 Business Central 中对组、角色或用户进行特定操作。

优先级

priority 是一个整数，用于定义分配多个角色或组的用户的优先级。新组的默认值为 **-100**。在 Business Central 中，您可以将整数值设置为优先级，该优先级使用以下规则解析：

表 29.14. 优先级值表

整数值	优先级
少于 -5	非常低
-5 到 0	低
等于 0	NORMAL

整数值	优先级
0 到 5 之间	HIGH
大于 5	非常高

主页

Home Page 表示用户的默认登录页面。

Workbench

Workbench 由以下定义的权限组成：

```
{
  "editDataObject": true,
  "plannerAvailable": true,
  "editGlobalPreferences": true,
  "editProfilePreferences": true,
  "accessDataTransfer": true,
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
}
```

pages, Editor, Spaces, and Projects

以下是基于资源类型的权限可能的值：

- **PAGES: read,create,update,delete**
- **EDITOR: read**
- **SPACES: read,create,update,delete**
- **PROJECT: read,create,update,delete,build**

您可以使用以下代码在 **Pages**、**Editor**、**Spaces** 和 **Projects** 权限中添加例外：

```
{
  "pages": {
    "read": false,
    "exceptions": [
      {
        "resourceName": "ProcessInstances",
        "permissions": {
          "read": false
        }
      },
      {
        "resourceName": "ProcessDefinitions",
        "permissions": {
          "read": false
        }
      }
```

```
    ]
  }
}
```

name 属性是您添加为例外的资源的标识符。使用以下 REST API 端点来获取可能标识符列表。REST API 基本 URL 是 **http://SERVER:PORT/business-central/rest/**。

- **[GET] /perspectives**: 返回 Business Central 中所有页面的视角名称
- **[GET] /editors**: 返回 Business Central 中的所有编辑器
- **[GET] /spaces**: 返回 Business Central 中的所有空格
- **[GET] /spaces/{spaceName}/projects**: 返回指定空间中的项目

页的服务器响应示例(JSON)

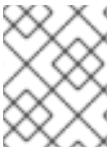
```
"pages": {
 "create": true,
 "read": false,
 "exceptions": [
   {
  "name": "HomePerspective",
  "permissions": {
   "read": true
  }
 }
]
}
```

第 30 章 KIE 会话和任务服务的 EJB API

Red Hat Process Automation Manager 提供了一个 Enterprise 3.0.0 (EJB) API，可用于嵌入式用例从应用程序远程访问 **KieSession** 和 **TaskService** 对象。EJB API 允许在红帽流程自动化管理器和远程客户应用程序中关闭流程引擎之间的事务集成。

虽然 KIE 服务器不支持 EJB，但您可以使用 EJB 作为与远程 REST 或 JMS 操作与 KIE 服务器类似的进程引擎的远程协议。

EJB 接口的实现是单一框架独立且容器无关的 API，您可以和特定于框架的代码一起使用。EJB 服务通过 Red Hat Process Automation Manager 中的 **org.jbpm.services.api** 和 **org.jbpm.services.ejb** 软件包公开。该实施不支持 **RuleService** 类，但 **ProcessService** 类会公开一个 **执行** 方法，供您使用各种规则相关的命令，如 **InsertCommand** 和 **FireAllRulesCommand**。



注意

Red Hat Process Automation Manager 中的 **org.jbpm.services.cdi** 软件包也支持上下文和依赖注入(CDI)。但是，为了避免 EJB 集成冲突，请不要一起使用 EJB 和 CDI。

30.1. 支持的 EJB 服务

对于 Red Hat Process Automation Manager 中的可用 Enterprise paxos (EJB)服务的完整列表，请 从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.12.0 Maven Repository** 再导航到 **~/jboss-rhba-7.12.0.GA-maven-repository/maven-repository/maven-repository/jbpm/jbpm-services-ejb-***。

为 jBPM 服务提供 EJB 接口的工件位于以下软件包中：

- **org.jbpm.services.ejb.api**: 包含 EJB 接口的 jBPM 服务 API 扩展
- **org.jbpm.services.ejb.impl**: 在核心服务实施之上包含 EJB 包装程序
- **org.jbpm.services.ejb.client** ：包含仅在 Red Hat JBoss EAP 上支持的 EJB 远程客户端实施。

org.jbpm.services.ejb.api 软件包包含以下可与远程 EJB 客户端一起使用的服务接口：

- **DefinitionServiceEJBRemote** ：使用这个界面收集有关进程(ID、名称和版本)、进程变量（名称和类型）、定义可重复使用的子进程、特定于域的服务、用户任务和用户任务输入和输出的信息。
- **DeploymentServiceEJBRemote** ：使用此接口启动部署和未部署。接口包括方法 **deploy**, **undeploy**, **getRuntimeManager**, **getDeployedUnits**, **isDeployed**, **activate**, **deactivate**, 和 **getDeployedUnit**。使用 **DeploymentUnit** 实例调用 **deploy** 方法，通过构建 **RuntimeManager** 实例将单元部署到运行时引擎中。部署成功后，会创建一个 **DeployedUnit** 实例并缓存来进一步使用。（要使用这些方法，您必须在 Maven 存储库中安装项目的工件。）
- **ProcessServiceEJBRemote** ：使用此界面控制一个或多个进程和工作项目的生命周期。
- **RuntimeDataServiceEJBRemote** ：使用此接口检索与运行时相关的数据，如进程实例、进程定义、节点实例信息和变量信息。接口包括多个便捷的方法，用于根据所有者、状态和时间收集任务信息。
- **UserTaskServiceEJBRemote** ：使用此接口来控制用户任务的生命周期。这个界面包括多个与用户任务交互的方便方法，如 **激活**、启动、**停止**，**以及执行**。
- **QueryServiceEJBRemote** ：使用此接口进行高级查询。

- **ProcessInstanceMigrationServiceEJBRemote** ：在部署进程定义的新版本时，使用此接口迁移进程实例。

如果您在相同的 KIE 服务器实例上运行 EJB 应用程序和 Business Central，则您可以通过设置 **org.jbpm.deploy.sync.int** 系统属性，按照特定间隔同步 EJB 和 Business Central 之间的信息。在服务完成同步后，您可以使用 REST 操作访问更新的信息。

注意

红帽流程自动化管理器中的 EJB 服务适用于嵌入式用例。如果您在同一 KIE 服务器实例上运行 EJB 应用程序和 Business Central，还必须在 EJB 应用程序的类路径中添加 **kie-services** 软件包。

30.2. 部署 EJB 服务 WAR 文件

您可以使用 Enterprise fsid (EJB)接口创建和部署您要用作 Red Hat Process Automation Manager 分发一部分的 EJB 服务 WAR 文件。

流程

1. 使用启动 Java 类注册人工任务回调，如下例所示：

```
@Singleton
@Startup
public class StartupBean {

  @PostConstruct
  public void init()
  { System.setProperty("org.jbpm.ht.callback", "jaas"); }

}
```

2. 构建您的 EJB 项目，以根据您的项目配置生成 WAR 文件。

3. 在运行 Red Hat Process Automation Manager 的 Red Hat JBoss EAP 实例中部署生成的文件。避免将 **Singleton** 策略用于运行时会话。**Singleton** 策略可能会导致应用程序多次从底层文件系统中加载相同的 **ksession** 实例，并导致安全锁定异常。

   如果要在 Red Hat JBoss EAP 实例上部署 EJB WAR 文件，与运行 Red Hat Process Automation Manager 的单独部署，请配置您的应用程序或应用服务器来调用远程 EJB 并传播安全上下文。

   如果您使用 Hibernate 为红帽流程自动化管理器创建数据库架构，请更新 Business Central 中的 **persistence.xml** 文件，并设置 **hibernate.hbm2ddl.auto** 属性的值 **来更新**，而不是 **创建**。

4. 通过创建基本 Web 应用程序并注入 EJB 服务来在本地测试部署，如下例所示：

```
@EJB(lookup = "ejb:/sample-war-ejb-
app/ProcessServiceEJBImpl!org.jbpm.services.ejb.api.ProcessServiceEJBRemote")
private ProcessServiceEJBRemote processService;

@EJB(lookup = "ejb:/sample-war-ejb-
app/UserTaskServiceEJBImpl!org.jbpm.services.ejb.api.UserTaskServiceEJBRemote")
private UserTaskServiceEJBRemote userTaskService;

@EJB(lookup = "ejb:/sample-war-ejb-
```

```
app/RuntimeDataServiceEJBImpl!org.jbpm.services.ejb.api.RuntimeDataServiceEJBRemote")

private RuntimeDataServiceEJBRemote runtimeDataService;
```

有关使用红帽 JBoss EAP 开发和部署 EJB 应用程序的更多信息，请参阅 *开发 EJB 应用*。

第 31 章 其他资源

- *管理和监控 KIE 服务器*
- *打包和部署 Red Hat Process Automation Manager 项目*

附录 A. 版本控制信息

文档最新更新于 2023 年 2 月 1 日（周三）。

附录 B. 联系信息

Red Hat Process Automation Manager 文档团队： brms-docs@redhat.com