



Red Hat Process Automation Manager 7.13

管理 Red Hat Process Automation Manager 和
KIE 服务器设置

Red Hat Process Automation Manager 7.13 管理 Red Hat Process Automation Manager 和 KIE 服务器设置

法律通告

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

本文档论述了如何修改 Red Hat Process Automation Manager 和 KIE 服务器设置以及属性以满足您的业务需求。

目录

前言	6
使开源包含更多	7
部分 I. 管理和监控 KIE 服务器	8
第 1 章 RED HAT PROCESS AUTOMATION MANAGER 组件	9
第 2 章 系统与 MAVEN 集成	10
2.1. 对本地项目的预先验证	10
2.2. BUSINESS CENTRAL 中的重复的 GAV 检测	10
2.3. 在 BUSINESS CENTRAL 中管理重复的 GAV 检测设置	11
第 3 章 将补丁更新和次要发行升级应用到 RED HAT PROCESS AUTOMATION MANAGER	13
第 4 章 配置并启动 KIE 服务器	19
第 5 章 为 KIE 服务器配置 JDBC 数据源	22
第 6 章 受管 KIE 服务器	26
第 7 章 非受管 KIE 服务器	27
第 8 章 在 KIE 服务器和 BUSINESS CENTRAL 中配置环境模式	28
第 9 章 配置 KIE 服务器以连接至 BUSINESS CENTRAL	29
第 10 章 安装并运行无头进程自动化管理器控制器	33
10.1. 使用安装程序通过流程自动化管理器控制器配置 KIE 服务器	33
10.2. 安装无头进程自动化管理器控制器	34
10.3. 运行无头进程自动化管理器控制器	38
10.4. 使用无头进程自动化管理器控制器集群 KIE 服务器	40
第 11 章 为 TLS 支持配置智能路由器	43
第 12 章 在 KIE 服务器中激活或取消激活 KIE 容器	45
第 13 章 部署描述符	46
13.1. 部署描述符配置	46
13.2. 管理部署描述符	48
13.3. 限制访问运行时引擎	49
第 14 章 从 BUSINESS CENTRAL 访问运行时数据	50
第 15 章 RED HAT PROCESS AUTOMATION MANAGER 中的 PROMETHEUS 指标监控	51
15.1. 为 KIE 服务器配置 PROMETHEUS 指标监控	51
15.2. 在 RED HAT OPENSIFT CONTAINER PLATFORM 上为 KIE 服务器配置 PROMETHEUS 指标监控	60
15.3. 使用自定义指标在 KIE 服务器中扩展 PROMETHEUS 指标监控	65
第 16 章 配置 OPENSIFT 连接超时	71
第 17 章 持久性	72
17.1. 配置 KIE 服务器持久性	72
17.2. 配置安全点	74
17.3. 会话持久性实体	76
17.4. 处理实例持久性实体	76
17.5. 工作项目持久性实体	77

17.6. 关联关键实体	77
17.7. 上下文映射实体	78
17.8. 重要的锁定支持	79
17.9. 在 RED HAT PROCESS AUTOMATION MANAGER 中以独立数据库模式持久保留进程变量	81
第 18 章 定义 LDAP 登录域	87
第 19 章 通过 RH-SSO 验证第三方客户端	88
19.1. 基本身份验证 (BASIC AUTHENTICATION)	88
第 20 章 KIE 服务器系统属性	89
第 21 章 KIE 服务器功能和扩展	97
21.1. 使用自定义 REST API 端点扩展现有 KIE 服务器功能	99
21.2. 扩展 KIE 服务器以使用自定义数据传输	106
21.3. 使用自定义客户端 API 扩展 KIE 服务器客户端	114
第 22 章 KIE 服务器的性能调整注意事项	122
第 23 章 其他资源	123
部分 II. 配置 BUSINESS CENTRAL 设置和属性	124
第 24 章 用户和组群管理	126
24.1. 创建用户	126
24.2. 编辑用户	127
24.3. 创建组	128
24.4. 编辑组	129
第 25 章 安全管理	130
25.1. 安全管理供应商	130
25.2. 权限和设置	133
第 26 章 工件管理	137
26.1. 查看工件	137
26.2. 下载工件	137
26.3. 上传工件	138
第 27 章 数据源和数据库驱动程序管理	139
27.1. 添加数据源	139
27.2. 编辑数据源	139
27.3. 删除数据源	140
27.4. 添加数据库驱动程序	140
27.5. 编辑数据库驱动程序	141
27.6. 删除数据库驱动程序	142
第 28 章 数据集编写	143
28.1. 添加数据集	143
28.2. 编辑数据集	144
28.3. 数据刷新	145
28.4. 缓存数据	145
第 29 章 ARCHETYPE 管理	147
29.1. 列出 ARCHETYPES	147
29.2. 添加 ARCHETYPE	147
29.3. 管理 ARCHETYPE 的附加功能	148
29.4. 使用 ARCHETYPES 创建项目	149

29.5. 使用 BUSINESS CENTRAL 中的空间设置来管理 ARCHETYPES	150
第 30 章 自定义项目首选项	151
第 31 章 自定义工件存储库属性	154
第 32 章 自定义语言设置	155
第 33 章 自定义进程管理	156
第 34 章 自定义过程设计器	157
第 35 章 SSH 密钥	158
35.1. 默认 SSH 密钥存储	158
35.2. 自定义 SSH 密钥存储	159
35.3. 创建 SSH 密钥	159
35.4. 使用 SSH 密钥存储注册 SSH 公钥	160
35.5. 删除 SSH 密钥	161
第 36 章 在 BUSINESS CENTRAL 中管理自定义任务	162
第 37 章 LDAP 连接	166
37.1. LDAP USERGROUPCALLBACK 实现	167
第 38 章 数据库连接	170
38.1. 数据库用户 GROUPCALLBACK 实现	170
第 39 章 使用 SETTINGS.XML 文件配置 MAVEN	172
其他资源	172
第 40 章 GAV 检查管理	173
40.1. 配置 GAV 检查和子 GAV 版本	173
40.2. 为所有项目配置 GAV 检查	174
第 41 章 在 KIE 服务器和 BUSINESS CENTRAL 中配置环境模式	175
第 42 章 GIT HOOK 和远程 GIT 存储库集成	176
42.1. 创建 POST-COMMIT GIT HOOK	176
42.2. 导入远程 GIT 存储库	177
42.3. 为现有远程 GIT 项目存储库配置 GIT HOOK	179
42.4. 将 GIT HOOK 配置为 BUSINESS CENTRAL 的系统属性	180
42.5. 集成远程 GIT 存储库	182
42.6. GIT HOOK 退出代码	185
42.7. 自定义 GIT HOOK 通知	185
第 43 章 针对 BUSINESS CENTRAL 中分支的角色访问控制	188
43.1. 自定义基于角色的访问控制	188
第 44 章 查看进程实例日志	190
第 45 章 BUSINESS CENTRAL 系统属性	191
第 46 章 与 BUSINESS CENTRAL 相关的性能调优注意事项	200
部分 III. 在 BUSINESS CENTRAL 中使用独立视角	201
第 47 章 BUSINESS CENTRAL 中的独立视角	202
第 48 章 使用独立库视角	203

第 49 章 使用独立编辑器视角	204
第 50 章 使用独立内容管理器视角	205
第 51 章 使用独立自定义页面（仪表板）	206
部分 IV. 在 BUSINESS CENTRAL 中创建自定义页面	207
第 52 章 BUSINESS CENTRAL 自定义仪表板	208
第 53 章 DASHBUILDER RUNTIME 和 DASHBUILDER STANDALONE	209
53.1. 在 RED HAT JBOSS EAP 上安装 DASHBUILDER 运行时	209
53.2. 在 RED HAT OPENSIFT CONTAINER PLATFORM 上部署 DASHBUILDER STANDALONE	212
第 54 章 数据集编写	218
54.1. 添加数据集	218
54.2. 编辑数据集	219
54.3. 数据刷新	220
54.4. 缓存数据	220
54.5. KIE 服务器数据集，使用 DASHBUILDER RUNTIME 和 DASHBUILDER STANDALONE	221
第 55 章 页面编写	224
55.1. 创建页面	224
55.2. 保存、删除、重命名或复制页面	225
55.3. 导航树	225
55.4. 组件	230
55.5. HEATMAP 组件	236
55.6. 外部组件	244
第 56 章 安全管理	248
56.1. 安全管理供应商	248
56.2. 权限和设置	251
第 57 章 导出、导入和部署仪表板	255
57.1. 导出 BUSINESS CENTRAL 仪表板数据	255
57.2. 导入 BUSINESS CENTRAL 仪表板数据	256
57.3. 在 DASHBUILDER RUNTIME 上部署 BUSINESS CENTRAL 的仪表板	257
附录 A. 版本信息	259
附录 B. 联系信息	260

前言

作为开发人员或系统管理员，您可以修改 Red Hat Process Automation Manager 和 KIE 服务器设置和属性来满足您的业务需求。您可以修改 Red Hat Process Automation Manager 运行时、业务中心接口或 KIE 服务器的行为。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 [CTO Chris Wright](#) 的信息。

部分 I. 管理和监控 KIE 服务器

作为系统管理员，您可以为生产环境安装、配置和升级红帽流程自动化管理器，快速轻松地对系统故障进行故障排除，并确保系统运行最佳。

先决条件

- Red Hat JBoss Enterprise Application Platform 7.4 已安装。如需更多信息，请参阅 [Red Hat JBoss Enterprise Application Platform 7.4 安装指南](#)。
- 安装了 Red Hat Process Automation Manager。如需更多信息，请参阅 [规划 Red Hat Process Automation Manager 安装](#)。
- Red Hat Process Automation Manager 正在运行，您可以使用 **admin** 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](#)。



注意

有关管理和监控在 KIE 服务器上运行的业务流程的详情，请参考 [Business Central 中管理和监控业务流程](#)。

第 1 章 RED HAT PROCESS AUTOMATION MANAGER 组件

该产品由 Business Central 和 KIE 服务器组成。

- Business Central 是您创建和管理业务规则的图形用户界面。您可以在 Red Hat JBoss EAP 实例或 Red Hat OpenShift Container Platform(OpenShift)上安装 Business Central。Business Central 也作为独立 JAR 文件提供。您可以使用 Business Central 独立 JAR 文件运行 Business Central，而无需将其部署到应用服务器。
- KIE 服务器是执行规则和其他工件的服务器。它用于实例化和执行规则并解决规划问题。您可以在红帽 JBoss EAP 实例中安装 KIE 服务器（在红帽 JBoss EAP 集群中），在 OpenShift 上、Oracle WebLogic 服务器实例、IBM WebSphere Application Server 实例或作为 Spring Boot 应用程序的一部分。
您可以将 KIE 服务器配置为在受管或非受管模式下运行。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。KIE 容器是项目的特定版本。如果管理 KIE 服务器，则“流程自动化管理器”控制器管理 KIE 服务器配置，并与流程自动化管理器控制器交互以创建和维护 KIE 容器。

第 2 章 系统与 MAVEN 集成

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

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

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

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

2.1. 对本地项目的预先验证

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

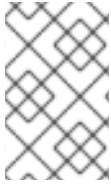
```
<server>
  <id>guvnor-m2-repo</id>
  <username>admin</username>
  <password>admin</password>
  <configuration>
    <wagonProvider>httpClient</wagonProvider>
    <httpConfiguration>
      <all>
        <usePreemptive>true</usePreemptive>
      </all>
    </httpConfiguration>
  </configuration>
</server>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

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

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

2.2. BUSINESS CENTRAL 中的重复的 GAV 检测

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



注意

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

每次执行以下操作时执行重复的 GAV 检测：

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

为重复的 GAV 检查以下 Maven 存储库：

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

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

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



注意

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

流程

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

以后，只针对您为验证启用的存储库，才会报告重复的 GAV。



注意

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

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


第 3 章 将补丁更新和次要发行升级应用到 RED HAT PROCESS AUTOMATION MANAGER

Red Hat Process Automation Manager 的补丁更新和新的次要版本中通常会提供自动更新，以帮助更新 Red Hat Process Automation Manager 的某些组件，如 Business Central、KIE 服务器和无头进程自动化管理器控制器。其他 Red Hat Process Automation Manager 工件（如决策引擎和独立 Business Central）都会作为新的工件发布，且您必须重新安装它们以应用更新。

您可以使用相同的自动更新工具将补丁更新和次要发行升级应用到 Red Hat Process Automation Manager 7.13。Red Hat Process Automation Manager 的补丁更新（如从版本 7.13 更新至 7.13.4）包括最新的安全更新和程序错误修复。Red Hat Process Automation Manager 的次发行版本升级，如从版本 7.12.x 升级到 7.13，包括功能增强、安全更新和程序错误修复。



注意

只有 Red Hat Process Automation Manager 的更新包含在 Red Hat Process Automation Manager 更新工具中。红帽 JBoss EAP 的更新必须使用红帽 JBoss EAP 补丁发布来应用。有关红帽 JBoss EAP 修补的更多信息，请参阅红帽 [JBoss EAP 修补和升级指南](#)。

先决条件

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

流程

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

如果您要升级到 Red Hat Process Automation Manager 的新次版本，如从版本 7.12.x 升级到 7.13，请首先将最新的补丁更新应用到 Red Hat Process Automation Manager 的当前版本，然后再次按照以下步骤升级到新的次版本。

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

这个版本工具会自动更新 Red Hat Process Automation Manager 的某些组件，如 Business Central、KIE 服务器和无头流程自动化管理器控制器。使用这个更新工具，首先应用更新，然后安装与 Red Hat Process Automation Manager 发行版本相关的其他更新或新的发行工件。

3.

如果要保留由更新工具更新的任何文件，请导航到提取的 `rhpm-$VERSION-update` 文件夹，打开 `blacklist.txt` 文件，并将相对路径添加到您不想要更新的文件。

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

在 `blacklist.txt` 文件中排除的文件示例：

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

更新后阻止的文件目录的内容：

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

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

4.

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



重要

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

在 Linux 或基于 Unix 的系统中：

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

在 Windows 中：

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

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

Red Hat Process Automation Manager 更新工具支持以下分布类型：

- **RHPAM-business-central-eap7-deployable: 更新 Business Central(business-central.war)**

- **RHPAM-kie-server-ee8 : 更新 KIE 服务器(kie-server.war)**



注意

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

- **RHPAM-kie-server-jws: 更新 KIE Server on Red Hat JBoss Web Server(kie-server.war)**

- **RHPAM-controller-ee7 : 更新无头进程自动化管理器控制器(controller.war)**

- **RHPAM-controller-jws: 更新 Red Hat JBoss Web Server(controller.war)上的无头进程自动化管理器控制器**

适用于 Red Hat JBoss EAP 上完整红帽流程自动化管理器分发的 Business Central 和 KIE 服务器示例：

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

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

使用，到无头进程 Automation Manager 控制器示例：

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

更新脚本会在解压的 `rhpam-$VERSION-update` 文件夹中创建一个 备份 文件夹，其中包含指定分发的副本，然后继续更新。

5.

更新工具完成后，返回到红帽客户门户网站的 **Software Downloads** 页面，其中包括您下载更新工具，并安装与 **Red Hat Process Automation Manager** 发行版本相关的其他更新或新的发行工件。

对于已在红帽流程自动化管理器分发中已存在的文件，如决策引擎或其他附加组件的 `.jar` 文件，将该文件的现有版本替换为红帽客户门户网站中的新版本。

6.

如果您使用独立 **Red Hat Process Automation Manager 7.13.4 Maven 存储库 工件** (`rhpam-7.13.4-maven-repository.zip`)，如 `air-gap` 环境中，下载 **Red Hat Process Automation Manager 7.13.4 Maven 存储库**，并将下载的 `rhpam-7.13.4-maven-repository.zip` 文件提取到现有的 `~/maven-repository` 目录，以更新相关内容。

Maven 存储库更新示例：

```
$ unzip -o rhpam-7.13.4-maven-repository.zip 'rhba-7.13.4.GA-maven-repository/maven-
repository/*' -d /tmp/rhbaMavenRepoUpdate

$ mv /tmp/rhbaMavenRepoUpdate/rhba-7.13.4.GA-maven-repository/maven-repository/
$REPO_PATH/
```



注意

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

7.

可选：如果您要使用基于属性的用户存储更改 **Red Hat Process Automation Manager**，请完成以下步骤：

a.

进入 `$JBOSS_HOME` 目录并运行以下命令之一：

•

在 **Linux** 或基于 **Unix** 的系统中：

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

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

-

在 Windows 中：

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

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

b.

运行以下命令：

-

在 Linux 或基于 Unix 的系统中：

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

-

在 Windows 中：

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

c.

导航到您提取 `rhpam-$VERSION-update.zip` 文件的目录，并运行以下命令来应用 `kie-fs-realm` 补丁之一：

-

在 Linux 或基于 Unix 的系统中：

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

-

在 Windows 中：

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

-
- 8. 应用完所有相关更新后，启动 **Red Hat Process Automation Manager** 和 **KIE 服务器**，并登录到 **Business Central**。
- 9. 验证 **Business Central** 窗口中的所有项目数据都显示和准确，然后在 **Business Central** 窗口右上角存在且准确，点击您的资料名称并点击 **About** 来验证更新的产品版本号。

如果您在 **Business Central** 中遇到错误或发现任何缺少的数据，您可以在 `rhpm-$VERSION-update` 文件夹中 恢复备份 文件夹中的内容来恢复更新工具更改。您还可以在红帽客户门户网站中重新安装之前版本的 **Red Hat Process Automation Manager** 中的相关发行工件。恢复之前的分发后，您可以重新尝试运行更新。

第 4 章 配置并启动 KIE 服务器

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

流程

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

```
$ ~/EAP_HOME/bin/standalone.sh --server-config=standalone-full.xml 1
-Dorg.kie.server.id=myserver 2
-Dorg.kie.server.user=kie_server_username 3
-Dorg.kie.server.pwd=kie_server_password 4
-Dorg.kie.server.controller=http://localhost:8080/business-central/rest/controller 5
-Dorg.kie.server.controller.user=controller_username 6
-Dorg.kie.server.controller.pwd=controller_password 7
-Dorg.kie.server.location=http://localhost:8080/kie-server/services/rest/server 8
-Dorg.kie.server.persistence.dialect=org.hibernate.dialect.PostgreSQLDialect 9
-Dorg.kie.server.persistence.ds=java:jboss/datasources/psjbpmDS 10
```

1

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

2

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

3

从流程自动化管理器控制器中与 KIE 服务器连接的用户名

4

从流程自动化管理器控制器中与 KIE 服务器连接的密码

5

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

6

用于连接到 **Process Automation Manager** 控制器 REST API 的用户名

7

连接到流程自动化管理器控制器 REST API 的密码

8

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

9

使用 Hibernate 电源

10

用于您之前 Red Hat JBoss BPM Suite 数据库的数据源的 JNDI 名称



注意

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

例如：

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

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

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

注意

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

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

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

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

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

- 设置以下 JVM 属性：

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

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

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

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

第 5 章 为 KIE 服务器配置 JDBC 数据源

数据源是一个对象，它允许 Java 数据库连接(JDBC)客户端（如应用服务器）与数据库建立连接。应用程序在 Java 命名和目录接口(JNDI)树或本地应用程序上下文中查找数据源，并请求数据库连接来检索数据。您必须为 KIE 服务器配置数据源，以确保在服务器和指定数据库之间进行正确的数据交换。

通常，使用 Red Hat Process Automation Manager 的解决方案在单个事务中管理多个资源。用于异步作业、事件和计时器的 JMS，例如：在确保数据原子性和一致性结果时，Red Hat Process Automation Manager 需要数据源中的 XA 驱动程序。如果监听器内存在针对不同架构的事务代码，或者从 jBPM 引擎提供的 hook 派生出来，则还需要一个 XA 驱动程序。

不要使用非 XA 数据源，除非您正面没有参与单一事务的多个资源。



注意

对于生产环境，请指定实际的数据源。不要在生产环境中使用示例数据源。

先决条件

- 您希望用于创建数据库连接的 JDBC 供应商在您要在其上部署 KIE 服务器的所有服务器上配置，如[红帽 JBoss 企业应用服务器配置指南](#)的"Creating Datasources"和"JDBC 驱动程序"部分中所述。
- Red Hat Process Automation Manager 7.13.4 Add Ons (rhpam-7.13.4-add-ons.zip)文件在红帽客户门户网站中的 [Software Downloads](#) 页面中下载。

流程

1. 完成以下步骤以准备数据库：
 - a. 在临时目录中提取 rhpam-7.13.4-add-ons.zip，如 TEMP_DIR。
 - b. Extract TEMP_DIR/rhpam-7.13.4-migration-tool.zip.
 - c. 将您的当前目录更改为 TEMP_DIR/rhpam-7.13.4-migration-tool/ddl-scripts 目录。此目录包含多种数据库类型的 DDL 脚本。

- d. 将您的数据库类型的 DDL 脚本导入到您要使用的数据库。

以下示例在 PostgreSQL 中创建 jBPM 数据库结构

```
psql jbpm < /ddl-scripts/postgresql/postgresql-jbpm-schema.sql
```



注意

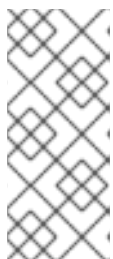
如果您与 Spring Boot 结合使用 PostgreSQL 或 Oracle，则必须导入对应的 Spring Boot DDL 脚本，如 `/ddl-scripts/oracle/oracle-springboot-jbpm-schema.sql` 或 `/ddl-scripts/postgresql/postgresql-springboot-jbpm-schema.sql`。



注意

PostgreSQL DDL 脚本创建 PostgreSQL 模式，其自动递增带有 @LOB 的实体属性的整数值(OID)列。要使用其他二进制列，如 BYTEA 而不是 OID，您必须使用 `postgresql-bytea-jbpm-schema.sql` 脚本创建 PostgreSQL 模式，并设置 Red Hat Process Automation Manager `org.kie.persistence.postgresql.useText=true` 和 `org.kie.persistence.postgresql.useBytea=true` 标志。在创建基于 BYTEA 的 schema 时，不要使用 `postgresql-jbpm-lo-trigger-clob.sql` 脚本。Red Hat Process Automation Manager 不提供从基于 OID 到基于 BYTEA 的 schema 的迁移工具。

2. 在文本编辑器中打开 `EAP_HOME/standalone/configuration/standalone-full.xml`，并找到 `<system-properties>` 标签。
3. 在 `<system-properties>` 标签中添加以下属性，其中 `<DATASOURCE>` 是数据源的 JNDI 名称，`<HIBERNATE_DIALECT>` 是您的数据库休眠状态。



注意

`org.kie.server.persistence.ds` 属性的默认值是 `java:jboss/datasources/ExampleDS`。`org.kie.server.persistence.dialect` 属性的默认值是 `org.hibernate.dialect.H2Dialect`。

```
<property name="org.kie.server.persistence.ds" value="<DATASOURCE>"/>
<property name="org.kie.server.persistence.dialect" value="<HIBERNATE_DIALECT>"/>
```

以下示例演示了如何为 **PostgreSQL hibernate dialect** 配置数据源：

```
<system-properties>
  <property name="org.kie.server.repo" value="{jboss.server.data.dir}"/>
  <property name="org.kie.example" value="true"/>
  <property name="org.jbpm.designer.perspective" value="full"/>
  <property name="designerdataobjects" value="false"/>
  <property name="org.kie.server.user" value="rhpamUser"/>
  <property name="org.kie.server.pwd" value="rhpam123!"/>
  <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="kieserver"/>
  <property name="org.kie.server.controller.pwd" value="kieserver1!"/>
  <property name="org.kie.server.id" value="local-server-123"/>

  <!-- Data source properties. -->
  <property name="org.kie.server.persistence.ds"
value="java:jboss/datasources/KieServerDS"/>
  <property name="org.kie.server.persistence.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
</system-properties>
```

支持以下分区：

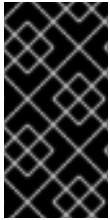
- **DB2: org.hibernate.dialect.DB2Dialect**
- **MSSQL: org.hibernate.SQLServer 2012Dialect**
- **MySQL: org.hibernate.dialect.MySQL5InnoDBDialect**
- **MariaDB : org.hibernate.dialect.MySQL5InnoDBDialect**
- **Oracle : org.hibernate.dialect.Oracle10gDialect**
- **PostgreSQL : org.hibernate.dialect.PostgreSQL82Dialect**

- PostgreSQL 加上 : `org.hibernate.dialect.PostgresPlusPlusDialect`
- Sybase: `org.hibernate.dialect.SybaseASE157Dialect`

第 6 章 受管 KIE 服务器

受管实例需要一个可用的流程自动化管理器控制器来启动 KIE 服务器。

进程自动化管理器控制器以集中的方式管理 KIE 服务器配置。每个流程自动化管理器控制器可以同时管理多个配置，并且环境中可以有多个进程自动化管理器控制器。受管 KIE 服务器可使用进程自动化管理器控制器列表进行配置，但一次只能连接到一个。



重要

所有进程自动化管理器控制器都应同步，以确保为服务器提供相同的配置集合，无论其连接到哪个进程自动化管理器控制器如何。

当使用进程自动化管理器控制器列表配置 KIE 服务器时，它将尝试在启动时尝试连接到每个服务器，直到连接成功建立其中一个。如果无法建立连接，服务器将不会启动，即使有可用的本地存储也可用。这样可确保一致性并防止服务器使用冗余配置运行。



注意

要在不连接到进程自动化管理器控制器的情况下以独立模式运行 KIE 服务器，请参阅 [第 7 章 非受管 KIE 服务器](#)。

第 7 章 非受管 KIE 服务器

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

以下操作会更新配置：

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



注意

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

第 8 章 在 KIE 服务器和 BUSINESS CENTRAL 中配置环境模式

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

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

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

流程

1. 要配置 KIE 服务器环境模式，请将 `org.kie.server.mode` 系统属性设置为 `org.kie.server.mode=development` 或 `org.kie.server.mode=production`。
2. 要在 Business Central 中配置项目部署行为，请转至 **Project Settings** → **General Settings** → **Version**，再切换 **Development Mode** 选项。



注意

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

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

第 9 章 配置 KIE 服务器以连接至 BUSINESS CENTRAL

**警告**

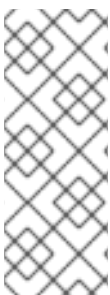
本节提供了一个示例设置，可用于测试目的。部分值不适用于生产环境，并被标记为。

如果您的红帽流程自动化管理器环境中没有配置 KIE 服务器，或者需要红帽流程自动化管理器环境中的其他 KIE 服务器，您必须配置 KIE 服务器以连接到 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。有关将其配置为连接到 Business Central 的说明。

可以管理或非受管 KIE 服务器。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。如果管理 KIE 服务器，则“流程自动化管理器”控制器管理 KIE 服务器配置，并与流程自动化管理器控制器交互以创建和维护 KIE 容器。

**注意**

如果 KIE 服务器由 Business Central 管理，并且您已从 ZIP 文件安装了 Red Hat Process Automation Manager，请对此小节中所述的更改。如果您已安装 Business Central，可以使用无头流程自动化管理器控制器来管理 KIE 服务器，如 [第 10 章 安装并运行无头进程自动化管理器控制器](#) 所述。

先决条件

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



注意

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

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

流程

1. 在 Red Hat Process Automation Manager 安装目录中，导航至 **standalone-full.xml** 文件。例如，如果您使用红帽 **JBoss EAP** 安装红帽流程自动化管理器，请转至 `$EAP_HOME/standalone/configuration/standalone-full.xml`。
2. 打开 **standalone-full.xml** 文件并在 `< system-properties>` 标签下设置以下 **JVM 属性**：

表 9.1. KIE 服务器实例的 JVM 属性

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

表 9.2. Business Central 实例的 JVM 属性

属性	值	备注
<code>org.kie.server.user</code>	<code>controllerUser</code>	使用角色 kie-server 的用户名。
<code>org.kie.server.pwd</code>	<code>controllerUser1234;</code>	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 服务器是否成功启动，请在 KIE 服务器运行时向 `http://SERVER:PORT/kie-server/services/rest/server/` 发送 GET 请求。有关在 KIE 服务器上运行 Red Hat Process Automation Manager 的更多信息，请参阅 [运行 Red Hat Process Automation Manager](#)。

身份验证成功后，您会收到类似以下示例的 XML 响应：

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

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

4.

验证成功注册：

a.

登录到 Business Central。

b.

点 Menu → Deploy → Execution Servers。

如果注册成功，您将看到注册的服务器 ID。

第 10 章 安装并运行无头进程自动化管理器控制器

您可以将 KIE 服务器配置为在受管或非受管模式下运行。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。如果管理 KIE 服务器，则“流程自动化管理器”控制器管理 KIE 服务器配置，并与流程自动化管理器控制器交互以创建和维护 KIE 容器。

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

10.1. 使用安装程序通过流程自动化管理器控制器配置 KIE 服务器

KIE 服务器可由流程自动化管理器控制器进行管理，也可以管理它。如果 KIE 服务器为非受管，您必须手动创建和维护 KIE 容器（部署单元）。如果管理 KIE 服务器，则“流程自动化管理器”控制器管理 KIE 服务器配置，并与流程自动化管理器控制器交互以创建和维护 KIE 容器。

流程自动化管理器控制器与 Business Central 集成。如果您安装 Business Central，您可以使用 Business Central 中的 执行服务器 页面与流程自动化管理器控制器交互。

您可以使用互动或 CLI 模式的安装程序来安装 Business Central 和 KIE 服务器，然后使用流程自动化管理器控制器配置 KIE 服务器。

先决条件

- 有两台已备份的 Red Hat JBoss EAP 7.4 服务器安装的计算机可用。
- 需要足够的用户权限以完成安装。

流程

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

3. 完成 **Business Central** 安装。
4. 在第二个计算机上，以交互模式或 CLI 模式运行安装程序。
5. 在 组件选择 页面中，清除 **Business Central** 框。
6. 在 **Configure Runtime Environment** 页面中，选择 **Perform Advanced Configuration**。
7. 选择"自定义 KIE 服务器"属性，然后单击"下一步"。
8. 输入 **Business Central** 的控制器 URL，并为 KIE 服务器配置其他属性。控制器 URL 具有以下格式，其中 `<HOST:PORT>` 是第二个计算机上的 **Business Central** 地址：

```
<HOST:PORT>/business-central/rest/controller
```
9. 完成安装。
10. 要验证流程自动化管理器控制器现在是否与 **Business Central** 集成，请转至 **Business Central** 中的 执行服务器 页面，并确认您配置的 KIE 服务器是否出现在 **REMOTE SERVERS** 下。

10.2. 安装无头进程自动化管理器控制器

您可以安装无头进程自动化管理器控制器，并使用 REST API 或 KIE 服务器 Java 客户端 API 与它交互。

先决条件

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

步骤

1. 进入红帽客户门户网站中的 [Software Downloads](#) 页面（需要登录），然后从下拉列表中选择产品和版本：
 - 产品：流程自动化管理器
 - 版本：7.13.4
2. 下载 Red Hat Process Automation Manager 7.13.4 Add Ons (`rhpm-7.13.4-add-ons.zip` 文件)。
3. 提取 `rhpm-7.13.4-add-ons.zip` 文件。`rhpm-7.13.4-controller-ee7.zip` 文件位于提取的目录中。
4. 将 `rhpm-7.13.4-controller-ee7.zip` 存档提取到临时目录。在以下示例中，此目录名为 `TEMP_DIR`。
5. 将 `TEMP_DIR/rhpm-7.13.4-controller-ee7/controller.war` 目录复制到 `EAP_HOME/standalone/deployments/`。



警告

确保您复制的无头进程自动化管理器控制器部署的名称与您在 Red Hat JBoss EAP 实例中的现有部署不冲突。

6. 将 `TEMP_DIR/rhpm-7.13.4-controller-ee7/SecurityPolicy/` 目录的内容复制到 `EAP_HOME/bin`。
7. 当系统提示覆盖文件时，选择是。
- 8.

在 `EAP_HOME/standalone/deployments/` 目录中，创建名为 `controller.war.dodeploy` 的空文件。此文件确保服务器启动时自动部署无头进程自动化管理器控制器。

10.2.1. 创建无头进程自动化管理器控制器用户

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

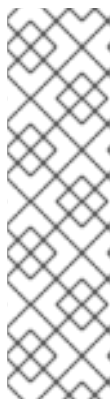
先决条件

- 无头进程自动化管理器控制器安装在 Red Hat JBoss EAP 安装(`EAP_HOME`)的 Base 目录中。

流程

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

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



注意

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

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

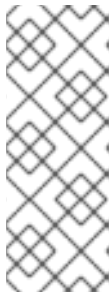
3. 记录您的用户名和密码。

10.2.2. 配置 KIE 服务器和无头进程自动化管理器控制器

如果 KIE 服务器由无头进程自动化管理器控制器管理，您必须编辑 KIE 服务器安装中的 `standalone-full.xml` 文件，并在无头进程自动化管理器安装中编辑 `standalone.xml` 文件。

先决条件

- KIE 服务器安装在 `EAP_HOME` 中。
- 无头进程自动化管理器控制器安装在 `EAP_HOME` 中。



注意

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

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

流程

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

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

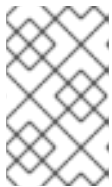
2. 在 KIE Server `EAP_HOME/standalone/configuration/standalone-full.xml` 文件中，将以下属性添加到 `< system-properties >` 部分：

```
<property name="org.kie.server.controller.user" value="<CONTROLLER_USER>"/>
<property name="org.kie.server.controller.pwd" value="<CONTROLLER_PWD>"/>
<property name="org.kie.server.id" value="<KIE_SERVER_ID>"/>
<property name="org.kie.server.location" value="http://<HOST>:<PORT>/kie-server/services/rest/server"/>
<property name="org.kie.server.controller" value="<CONTROLLER_URL>"/>
```

3.

在这个文件中，替换以下值：

- 将 `<CONTROLLER_USER>` 和 `<CONTROLLER_PWD>` 替换为用户的凭证，并将 `kie-server` 角色替换为 `kie-server` 角色。
- 将 `<KIE_SERVER_ID>` 替换为 KIE 服务器安装的 ID 或名称，例如 `rhcam-7.13.4-kie-server-1`。
- 将 `<HOST>` 替换为 KIE 服务器主机的 ID 或名称，例如 `localhost` 或 `192.7.8.9`。
- 将 `<PORT>` 替换为 KIE 服务器主机的端口，例如 `8080`。



注意

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

- 将 `<CONTROLLER_URL>` 替换为无头进程 Automation Manager 控制器的 URL。KIE 服务器在启动过程中连接到此 URL。

10.3. 运行无头进程自动化管理器控制器

在 Red Hat JBoss EAP 上安装了无头进程自动化管理器控制器后，使用此流程运行无头进程自动化管理器控制器。

先决条件

- 无头进程自动化管理器控制器会在 Red Hat JBoss EAP 安装(`EAP_HOME`)的 Base 目录中安装和配置。

流程

1. 在终端应用中，导航到 `EAP_HOME/bin`。
2. 如果您在与安装 KIE 服务器的 Red Hat JBoss EAP 实例相同的 Red Hat JBoss EAP 实例

中安装无头进程自动化管理器控制器，请输入以下命令：

- 在 Linux 或基于 UNIX 的系统中：

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

- 在 Windows 中：

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

3.

如果您在安装了 KIE 服务器的红帽 JBoss EAP 实例中安装了无头流程自动化管理器控制器，请使用 `standalone.sh` 脚本启动无头进程自动化管理器控制器：



注意

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

- 在 Linux 或基于 UNIX 的系统中：

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

- 在 Windows 中：

```
standalone.bat
```

4.

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

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



注意

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

10.4. 使用无头进程自动化管理器控制器集群 KIE 服务器

流程自动化管理器控制器与 **Business Central** 集成。但是，如果您不安装 **Business Central**，则可安装无头流程自动化管理器控制器，并使用 REST API 或 KIE 服务器 Java 客户端 API 与它交互。

先决条件

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

流程

1. 进入红帽客户门户网站中的 **Software Downloads** 页面（需要登录），然后从下拉列表中选择产品和版本：
 - **PRODUCT**：流程自动化管理器
 - **版本**：7.13.4
2. 下载 Red Hat Process Automation Manager 7.13.4 Add Ons (**rhpm-7.13.4-add-ons.zip** 文件)。
3. 提取 **rhpm-7.13.4-add-ons.zip** 文件。**rhpm-7.13.4-controller-ee7.zip** 文件位于提取的目录中。
4. 将 **rhpm-7.13.4-controller-ee7.zip** 存档提取到临时目录。在以下示例中，此目录名为 **TEMP_DIR**。
5. 将 **TEMP_DIR/rhpm-7.13.4-controller-ee7/controller.war** 目录复制到 **EAP_HOME/standalone/deployments/**。

**警告**

确保您复制的无头进程自动化管理器控制器部署的名称与您在 Red Hat JBoss EAP 实例中的现有部署不冲突。

6. 将 `TEMP_DIR/rhpam-7.13.4-controller-ee7/SecurityPolicy/` 目录的内容复制到 `EAP_HOME/bin`。
7. 当系统提示覆盖文件时，请单击 **Yes**。
8. 在 `EAP_HOME/standalone/deployments/` 目录中，创建名为 `controller.war.dodeploy` 的空文件。此文件确保服务器启动时自动部署无头进程自动化管理器控制器。
9. 在文本编辑器中打开 `EAP_HOME/standalone/configuration/standalone.xml` 文件。
10. 在 `<system-properties>` 元素中添加以下属性，并将 `< ;NFS_STORAGE >` 替换为存储模板配置的 NFS 存储的绝对路径：

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

模板文件包含特定部署场景的默认配置。

如果将 `org.kie.server.controller.templatefile.watcher.enabled` 属性的值设置为 `true`，则会启动一个单独的线程来监视模板文件的修改。这些检查的默认间隔为 30000 毫秒，可以由 `org.kie.server.controller.templatefile.watcher.interval` 系统属性进一步控制。如果此属性的值设为 `false`，则仅在服务器重启时检测到对模板文件的更改。

11. 要启动无头进程自动化管理器控制器，请导航到 `EAP_HOME/bin`，然后输入以下命令：

- 在 Linux 或基于 UNIX 的系统中：

```
┆ $ ./standalone.sh
```

- 在 Windows 中：

```
┆ standalone.bat
```

有关在 Red Hat JBoss Enterprise Application Platform 集群环境中运行 Red Hat Process Automation Manager 的更多信息，[请参阅在 Red Hat JBoss EAP 集群环境中安装和配置 Red Hat Process Automation Manager。](#)

第 11 章 为 TLS 支持配置智能路由器

您可以为传输层安全(TLS)支持配置智能路由器（KIE 服务器路由器）来允许 HTTPS 流量。另外，您可以禁用未安全 HTTP 连接至智能路由器。

先决条件

- KIE 服务器安装在红帽 JBoss EAP 7.4 集群的每个节点中。
- 已安装并配置了智能路由器。如需更多信息，请参阅在 [Red Hat JBoss EAP 集群环境中安装和配置 Red Hat Process Automation Manager](#)。

流程

要启动智能路由器，请使用以下方法之一：

- 要启动带有 TLS 支持和启用 HTTPS 的智能路由器以及允许 HTTP 连接，请输入以下命令：

```
java -Dorg.kie.server.router.tls.keystore = <KEYSTORE_PATH>  
-Dorg.kie.server.router.tls.keystore.password = <KEYSTORE_PASSWORD>  
-Dorg.kie.server.router.tls.keystore.keyalias = <KEYSTORE_ALIAS>  
-Dorg.kie.server.router.tls.port = <HTTPS_PORT>  
-jar rhpam-7.13.4-smart-router.jar
```

在本例中，替换以下变量：

- <KEYSTORE_PATH > : 将要存储密钥存储的路径。
- <KEYSTORE_PASSWORD > : 密钥存储密码。
- <KEYSTORE_ALIAS > : 用于存储证书的别名。
- <HTTPS_PORT > : HTTPS 端口。默认 HTTPS 端口为 9443。

- 要启动带有 TLS 支持和启用了 HTTPS 的智能卡并禁用 HTTP 连接，请输入以下命令：

```
java -Dorg.kie.server.router.tls.keystore = <KEYSTORE_PATH>  
-Dorg.kie.server.router.tls.keystore.password = <KEYSTORE_PASSWORD>  
-Dorg.kie.server.router.tls.keystore.keyalias = <KEYSTORE_ALIAS>  
-Dorg.kie.server.router.tls.port = <HTTPS_PORT>  
-Dorg.kie.server.router.port=0  
-jar rhpam-7.13.4-smart-router.jar
```

当 `org.kie.server.router.port` 系统属性设置为 0 时，HTTP 侦听器不会被注册。如果配置了 TLS 且 HTTP 侦听器没有注册，则智能路由器仅侦听 HTTPS 端口。



注意

如果没有配置 TLS，并且您可以通过将 `org.kie.server.router.port` 设置为 0 来禁用 HTTP，则会出现错误并智能路由器停止。

第 12 章 在 KIE 服务器中激活或取消激活 KIE 容器

现在，您可以通过取消激活但同时继续处理现有进程实例和任务来停止从给定容器创建新进程实例。如果取消激活是临时的，您可以稍后再次激活容器。KIE 容器的激活或停用不需要重启 KIE 服务器。

先决条件

- 在 **Business Central** 中创建和配置了 KIE 容器。

流程

1. 登录到 **Business Central**。
2. 在主菜单中，点击 **Menu** → **Deploy** → **Execution Servers**。
3. 在 "服务器配置" 窗格中，在页面的左侧选择您的服务器。
4. 在 **Deployment units** 窗格中，选择您要激活或取消激活的部署单元。
5. 在部署单元窗格的右上角点击 **激活或取消激活**。

在被取消激活后，您无法从 KIE 容器创建进程实例。

第 13 章 部署描述符

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

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

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

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

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

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

13.1. 部署描述符配置

通过部署描述符，用户可以在多个级别上配置执行服务器：

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

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



注意

部署时间配置适用于通过 REST API 进行的部署。

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

您可以配置什么配置？

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

表 13.1. 部署描述符

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

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



警告

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

13.2. 管理部署描述符

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

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

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

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

13.3. 限制访问运行时引擎

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

安全角色可用于限制对进程定义的访问，或者在运行时限制访问。

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

流程

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

第 14 章 从 BUSINESS CENTRAL 访问运行时数据

Business Central 中的以下页面允许您查看 KIE 服务器运行时数据：

- 流程报告
- 任务报告
- 进程定义
- 处理实例
- 执行错误
- Jobs
- 任务

这些页面使用当前登录用户的凭证来加载 KIE 服务器中的数据。因此，若要查看 Business Central 中的运行时数据，请确保满足以下条件：

- 运行 Business Central 应用程序的 KIE 容器（部署单元）中存在该用户。除 kie-server 角色外，此用户还必须已分配了 admin、anas 或 developer 角色，并具有对运行时数据的完整访问权限。manager 和 process_admin 角色还支持访问 Business Central 中的运行时数据页面。
- 用户存在于运行 KIE 服务器的 KIE 容器（部署单元）中，并分配了 kie-server 角色。
- 建立 Business Central 和 KIE 服务器之间的通信。也就是说，KIE 服务器在“流程自动化管理器”控制器中注册，它是 Business Central 的一部分。

第 15 章 RED HAT PROCESS AUTOMATION MANAGER 中的 PROMETHEUS 指标监控

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

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

对于 KIE 服务器通过 Prometheus 公开的可用指标列表，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.4 源分发，并导航到 `~/rhpam-7.13.4-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus`。
<https://access.redhat.com/jbossnetwork/restricted/listSoftware.html>



重要

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

15.1. 为 KIE 服务器配置 PROMETHEUS 指标监控

您可以将 KIE 服务器实例配置为使用 Prometheus 来收集并存储与 Red Hat Process Automation Manager 中的业务资产活动相关的指标。对于 KIE 服务器通过 Prometheus 公开的可用指标列表，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.4 源分发，并导航到 `~/rhpam-7.13.4-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services/kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus`。
<https://access.redhat.com/jbossnetwork/restricted/listSoftware.html>

先决条件

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

流程

1. 在您的 KIE 服务器实例中，将 `org.kie.prometheus.server.ext.disabled` 系统属性设置为 `false` 以启用 Prometheus 扩展。您可在启动 KIE 服务器时，或者在 Red Hat Process Automation Manager 分发的 `standalone.xml` 或 `standalone-full.xml` 文件中定义此属性。
2. 如果您在 Spring Boot 上运行 Red Hat Process Automation Manager，在 `application.properties` 系统属性中配置所需的密钥：

Red Hat Process Automation Manager 和 Prometheus 的 Spring Boot `application.properties` 密钥

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

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

`prometheus.yml` 文件中的 `scrape` 配置

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

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


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

根据您的 KIE 服务器位置和设置值。

4.

启动 KIE 服务器实例。

Red Hat JBoss EAP 上 Red Hat Process Automation Manager 的开始命令示例

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

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

5.

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

对于 REST 客户端：

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

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

对于 curl 实用程序 :

- -U : 使用 `kie-server` 角色输入 KIE 服务器用户的用户名和密码。
- -H : 设置以下标头 :

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

Red Hat JBoss EAP 上 Red Hat Process Automation Manager 的 curl 命令示例

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

Spring Boot 上的 Red Hat Process Automation Manager 的 curl 命令示例

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

服务器响应示例

```

# HELP kie_server_container_started_total Kie Server Started Containers
# TYPE kie_server_container_started_total counter
kie_server_container_started_total{container_id="task-assignment-kjar-1.0",} 1.0
# HELP solvers_running Number of solvers currently running
# TYPE solvers_running gauge
solvers_running 0.0
# HELP dmn_evaluate_decision_nanosecond DMN Evaluation Time
# TYPE dmn_evaluate_decision_nanosecond histogram
# HELP solver_duration_seconds Time in seconds it took solver to solve the constraint
problem
# TYPE solver_duration_seconds summary
solver_duration_seconds_count{solver_id="100tasks-5employees.xml",} 1.0
solver_duration_seconds_sum{solver_id="100tasks-5employees.xml",} 179.828255925
solver_duration_seconds_count{solver_id="24tasks-8employees.xml",} 1.0
solver_duration_seconds_sum{solver_id="24tasks-8employees.xml",} 179.995759653
# HELP drl_match_fired_nanosecond Drools Firing Time
# TYPE drl_match_fired_nanosecond histogram
# HELP dmn_evaluate_failed_count DMN Evaluation Failed
# TYPE dmn_evaluate_failed_count counter
# HELP kie_server_start_time Kie Server Start Time
# TYPE kie_server_start_time gauge
kie_server_start_time{name="myapp-kieserver",server_id="myapp-
kieserver",location="http://myapp-kieserver-demo-
monitoring.127.0.0.1.nip.io:80/services/rest/server",version="7.4.0.redhat-20190428",}
1.557221271502E12
# HELP kie_server_container_running_total Kie Server Running Containers
# TYPE kie_server_container_running_total gauge
kie_server_container_running_total{container_id="task-assignment-kjar-1.0",} 1.0
# HELP solver_score_calculation_speed Number of moves per second for a particular solver
solving the constraint problem
# TYPE solver_score_calculation_speed summary
solver_score_calculation_speed_count{solver_id="100tasks-5employees.xml",} 1.0
solver_score_calculation_speed_sum{solver_id="100tasks-5employees.xml",} 6997.0
solver_score_calculation_speed_count{solver_id="24tasks-8employees.xml",} 1.0
solver_score_calculation_speed_sum{solver_id="24tasks-8employees.xml",} 19772.0
# HELP kie_server_case_started_total Kie Server Started Cases
# TYPE kie_server_case_started_total counter
kie_server_case_started_total{case_definition_id="itorders.orderhardware",} 1.0
# HELP kie_server_case_running_total Kie Server Running Cases
# TYPE kie_server_case_running_total gauge
kie_server_case_running_total{case_definition_id="itorders.orderhardware",} 2.0
# HELP kie_server_data_set_registered_total Kie Server Data Set Registered
# TYPE kie_server_data_set_registered_total gauge
kie_server_data_set_registered_total{name="jbpmProcessInstanceLogs::CUSTOM",uuid="jbpmProcessInstanceLogs",} 1.0
kie_server_data_set_registered_total{name="jbpmRequestList::CUSTOM",uuid="jbpmRequest
List",} 1.0

```

```

kie_server_data_set_registered_total{name="tasksMonitoring::CUSTOM",uid="tasksMonitoring"},} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasks::CUSTOM",uid="jbpmHumanTasks"},} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasksWithUser::FILTERED_PO_TASK",uid="jbpmHumanTasksWithUser"},} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasksWithVariables::CUSTOM",uid="jbpmHumanTasksWithVariables"},} 1.0
kie_server_data_set_registered_total{name="jbpmProcessInstancesWithVariables::CUSTOM",uid="jbpmProcessInstancesWithVariables"},} 1.0
kie_server_data_set_registered_total{name="jbpmProcessInstances::CUSTOM",uid="jbpmProcessInstances"},} 1.0
kie_server_data_set_registered_total{name="jbpmExecutionErrorList::CUSTOM",uid="jbpmExecutionErrorList"},} 1.0
kie_server_data_set_registered_total{name="processesMonitoring::CUSTOM",uid="processesMonitoring"},} 1.0
kie_server_data_set_registered_total{name="jbpmHumanTasksWithAdmin::FILTERED_BA_TASK",uid="jbpmHumanTasksWithAdmin"},} 1.0
# HELP kie_server_execution_error_total Kie Server Execution Errors
# TYPE kie_server_execution_error_total counter
# HELP kie_server_task_completed_total Kie Server Completed Tasks
# TYPE kie_server_task_completed_total counter
# HELP kie_server_container_running_total Kie Server Running Containers
# TYPE kie_server_container_running_total gauge
kie_server_container_running_total{container_id="itorders_1.0.0-SNAPSHOT"},} 1.0
# HELP kie_server_job_cancelled_total Kie Server Cancelled Jobs
# TYPE kie_server_job_cancelled_total counter
# HELP kie_server_process_instance_started_total Kie Server Started Process Instances
# TYPE kie_server_process_instance_started_total counter
kie_server_process_instance_started_total{container_id="itorders_1.0.0-SNAPSHOT",process_id="itorders.orderhardware"},} 1.0
# HELP solver_duration_seconds Time in seconds it took solver to solve the constraint problem
# TYPE solver_duration_seconds summary
# HELP kie_server_task_skipped_total Kie Server Skipped Tasks
# TYPE kie_server_task_skipped_total counter
# HELP kie_server_data_set_execution_time_seconds Kie Server Data Set Execution Time
# TYPE kie_server_data_set_execution_time_seconds summary
kie_server_data_set_execution_time_seconds_count{uid="jbpmProcessInstances"},} 8.0
kie_server_data_set_execution_time_seconds_sum{uid="jbpmProcessInstances"},} 0.056000000000000001
# HELP kie_server_job_scheduled_total Kie Server Started Jobs
# TYPE kie_server_job_scheduled_total counter
# HELP kie_server_data_set_execution_total Kie Server Data Set Execution
# TYPE kie_server_data_set_execution_total counter
kie_server_data_set_execution_total{uid="jbpmProcessInstances"},} 8.0
# HELP kie_server_process_instance_completed_total Kie Server Completed Process Instances
# TYPE kie_server_process_instance_completed_total counter
# HELP kie_server_job_running_total Kie Server Running Jobs
# TYPE kie_server_job_running_total gauge
# HELP kie_server_task_failed_total Kie Server Failed Tasks
# TYPE kie_server_task_failed_total counter
# HELP kie_server_task_exited_total Kie Server Exited Tasks
# TYPE kie_server_task_exited_total counter
# HELP dmn_evaluate_decision_nanosecond DMN Evaluation Time

```

```
# TYPE dmn_evaluate_decision_nanosecond histogram
# HELP kie_server_data_set_lookups_total Kie Server Data Set Running Lookups
# TYPE kie_server_data_set_lookups_total gauge
kie_server_data_set_lookups_total{uuid="jbpmProcessInstances",} 0.0
# HELP kie_server_process_instance_duration_seconds Kie Server Process Instances
Duration
# TYPE kie_server_process_instance_duration_seconds summary
# HELP kie_server_case_duration_seconds Kie Server Case Duration
# TYPE kie_server_case_duration_seconds summary
# HELP dmn_evaluate_failed_count DMN Evaluation Failed
# TYPE dmn_evaluate_failed_count counter
# HELP kie_server_task_added_total Kie Server Added Tasks
# TYPE kie_server_task_added_total counter
kie_server_task_added_total{deployment_id="itorders_1.0.0-
SNAPSHOT",process_id="itorders.orderhardware",task_name="Prepare hardware spec",}
1.0
# HELP drl_match_fired_nanosecond Drools Firing Time
# TYPE drl_match_fired_nanosecond histogram
# HELP kie_server_container_started_total Kie Server Started Containers
# TYPE kie_server_container_started_total counter
kie_server_container_started_total{container_id="itorders_1.0.0-SNAPSHOT",} 1.0
# HELP kie_server_process_instance_sla_violated_total Kie Server Process Instances SLA
Violated
# TYPE kie_server_process_instance_sla_violated_total counter
# HELP kie_server_task_duration_seconds Kie Server Task Duration
# TYPE kie_server_task_duration_seconds summary
# HELP kie_server_job_executed_total Kie Server Executed Jobs
# TYPE kie_server_job_executed_total counter
# HELP kie_server_deployments_active_total Kie Server Active Deployments
# TYPE kie_server_deployments_active_total gauge
kie_server_deployments_active_total{deployment_id="itorders_1.0.0-SNAPSHOT",} 1.0
# HELP kie_server_process_instance_running_total Kie Server Running Process Instances
# TYPE kie_server_process_instance_running_total gauge
kie_server_process_instance_running_total{container_id="itorders_1.0.0-
SNAPSHOT",process_id="itorders.orderhardware",} 2.0
# HELP solvers_running Number of solvers currently running
# TYPE solvers_running gauge
solvers_running 0.0
# HELP kie_server_work_item_duration_seconds Kie Server Work Items Duration
# TYPE kie_server_work_item_duration_seconds summary
# HELP kie_server_job_duration_seconds Kie Server Job Duration
# TYPE kie_server_job_duration_seconds summary
# HELP solver_score_calculation_speed Number of moves per second for a particular solver
solving the constraint problem
# TYPE solver_score_calculation_speed summary
# HELP kie_server_start_time Kie Server Start Time
# TYPE kie_server_start_time gauge
kie_server_start_time{name="sample-server",server_id="sample-
server",location="http://localhost:8080/kie-server/services/rest/server",version="7.68.0-
SNAPSHOT",} 1.557285486469E12
```

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

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

图 15.1. 带有 KIE 服务器指标的 Prometheus 表达式浏览器

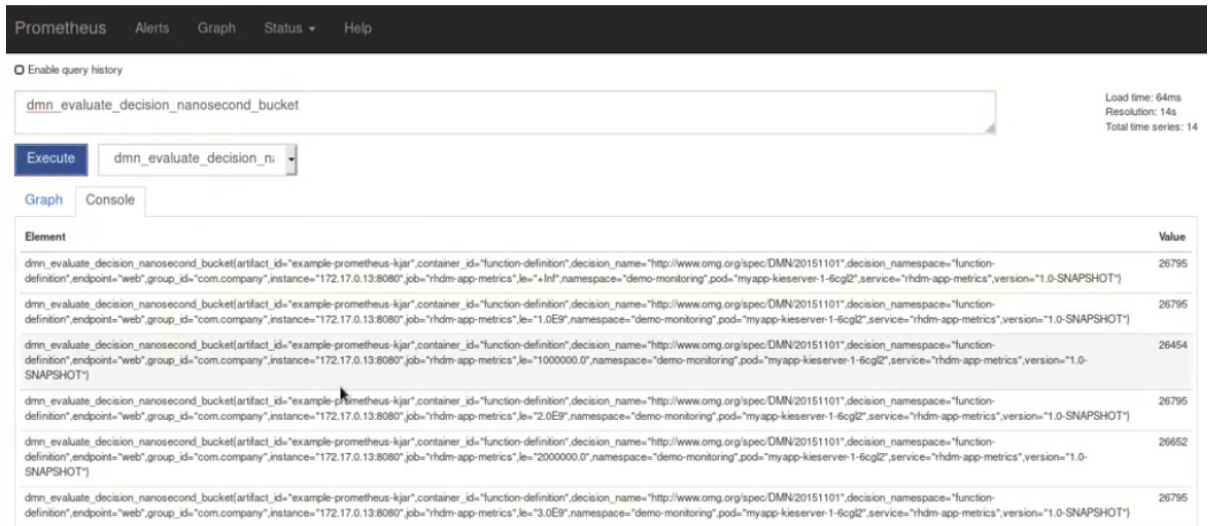


图 15.2. 带有 KIE 服务器目标的 Prometheus 表达式浏览器

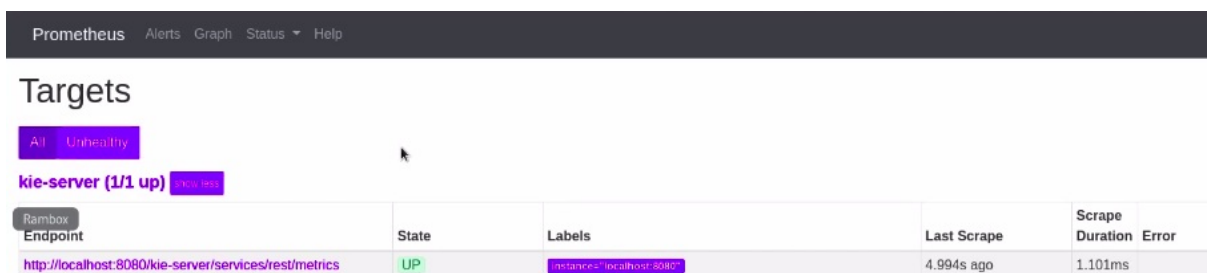


图 15.3. DMN 型号的 KIE 服务器指标的 Grafana 仪表盘



图 15.4. 带有临时解决方案的 KIE 服务器指标的 Grafana 仪表盘

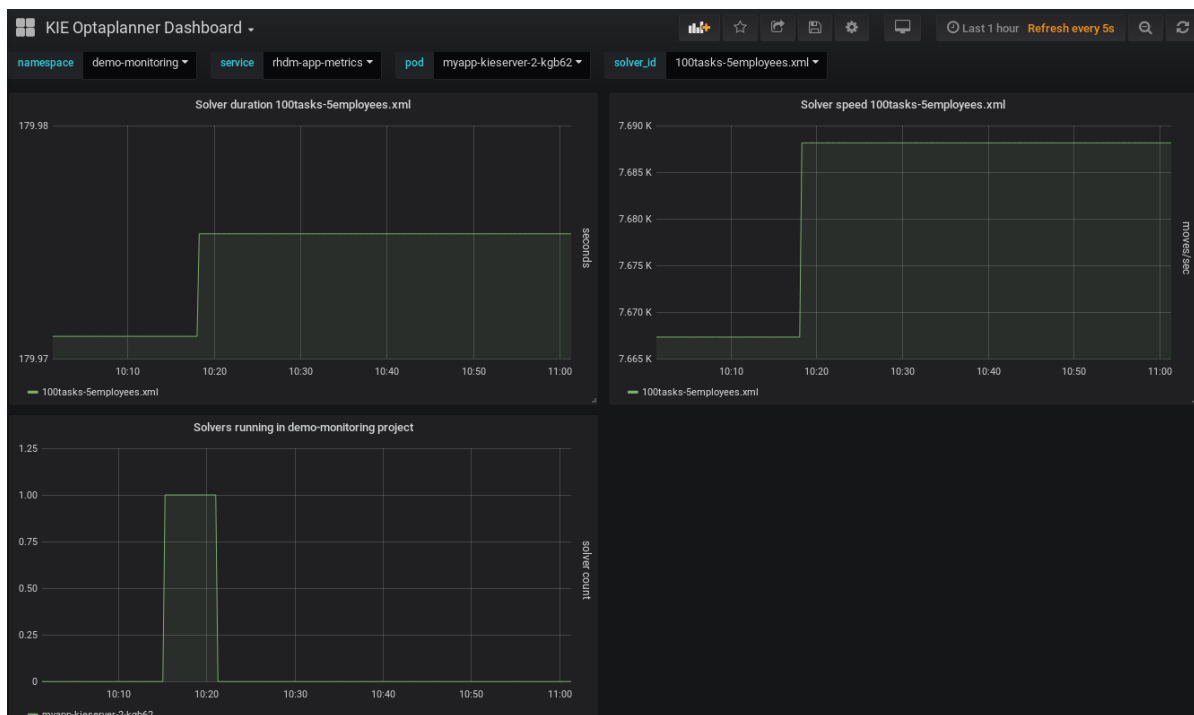
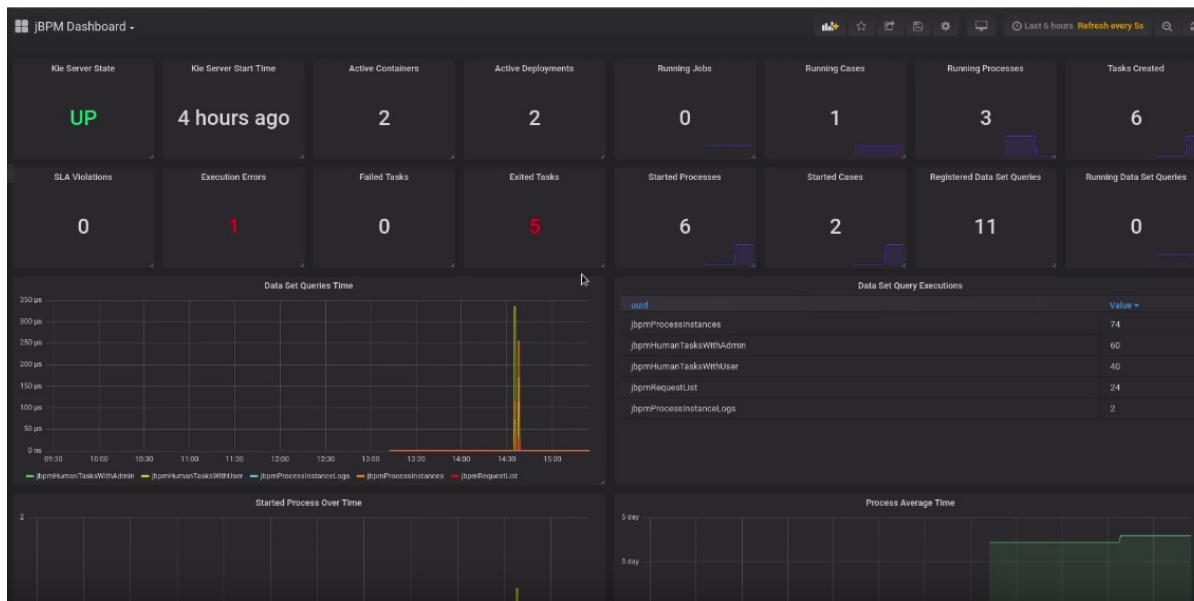


图 15.5. 带有进程、用例和任务的 KIE 服务器指标的 Grafana 仪表盘



其他资源

- [Prometheus 入门](#)
- [Prometheus 的 Grafana 支持](#)
- [在 Grafana 中使用 Prometheus](#)

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

您可以在 Red Hat OpenShift Container Platform 上配置 KIE 服务器部署，以使用 Prometheus 来收集并存储与 Red Hat Process Automation Manager 中的业务资产活动相关的指标。对于 KIE 服务器通过 Prometheus 公开的可用指标列表，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.4 源分发，并导航到 `~/rhpam-7.13.4-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-services/kie-server-services-kie-server-services-prometheus/src/main/java/org/kie/server/services/prometheus`。
<https://access.redhat.com/jbossnetwork/restricted/listSoftware.html>

先决条件

- KIE 服务器已安装并部署到 Red Hat OpenShift Container Platform 上。如需有关 OpenShift 上的 KIE 服务器的更多信息，请参阅 [Red Hat Process Automation Manager 7.13 产品文档中的相关 OpenShift 部署选项](#)。
- 您有 `kie-server` 用户角色访问权限到 KIE 服务器。
- 已安装 Prometheus Operator。有关下载和使用 Prometheus Operator 的信息，请参阅 GitHub 中的 [Prometheus Operator](#) 项目。

流程

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

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

如果您尚未在 OpenShift 上部署您的 KIE 服务器，那么在计划用于 OpenShift 部署的 OpenShift 模板中（例如：`rhpam713-prod-immutable-kieserver.yaml`），您可以将 `PROMETHEUS_SERVER_EXT_DISABLED` 模板参数设置为 `false` 来启用 Prometheus 扩展。

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

```
apiVersion: app.kiegroup.org/v1
kind: KieApp
```



```

metadata:
  name: enable-prometheus
spec:
  environment: rhpam-trial
  objects:
    servers:
      - env:
          - name: PROMETHEUS_SERVER_EXT_DISABLED
            value: "false"

```

2.

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

```

apiVersion: v1
kind: Service
metadata:
  annotations:
    description: RHPAM Prometheus metrics exposed
  labels:
    app: myapp-kieserver
    application: myapp-kieserver
    template: myapp-kieserver
    metrics: rhpam
  name: rhpam-app-metrics
spec:
  ports:
    - name: web
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    deploymentConfig: myapp-kieserver
  sessionAffinity: None
  type: ClusterIP

```

3.

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

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

4.

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

5.

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

```
apiVersion: monitoring.coreos.com/v1
```

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

6.

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

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

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

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

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

图 15.6. 带有 KIE 服务器指标的 Prometheus 表达式浏览器

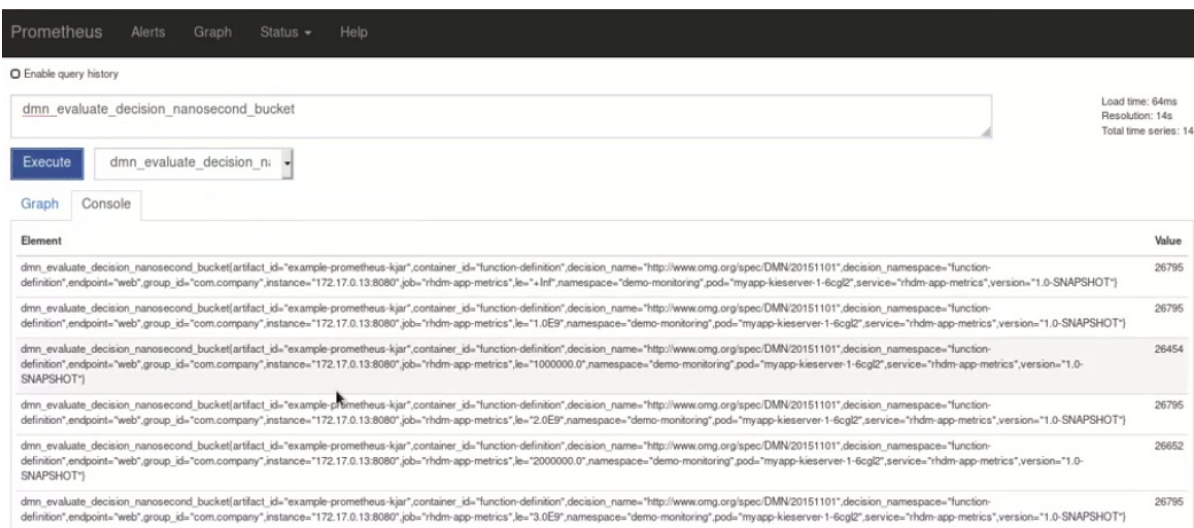


图 15.7. 带有 KIE 服务器目标的 Prometheus 表达式浏览器

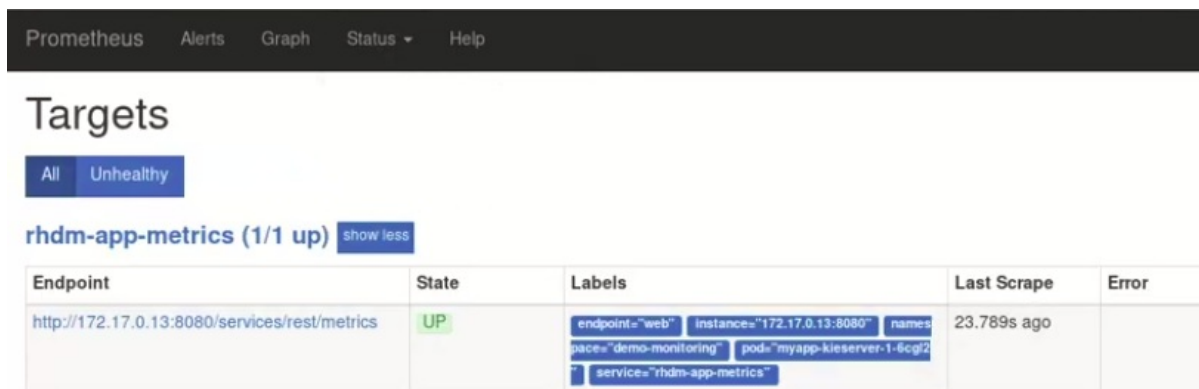


图 15.8. DMN 型号的 KIE 服务器指标的 Grafana 仪表盘



图 15.9. 带有临时解决方案的 KIE 服务器指标的 Grafana 仪表盘

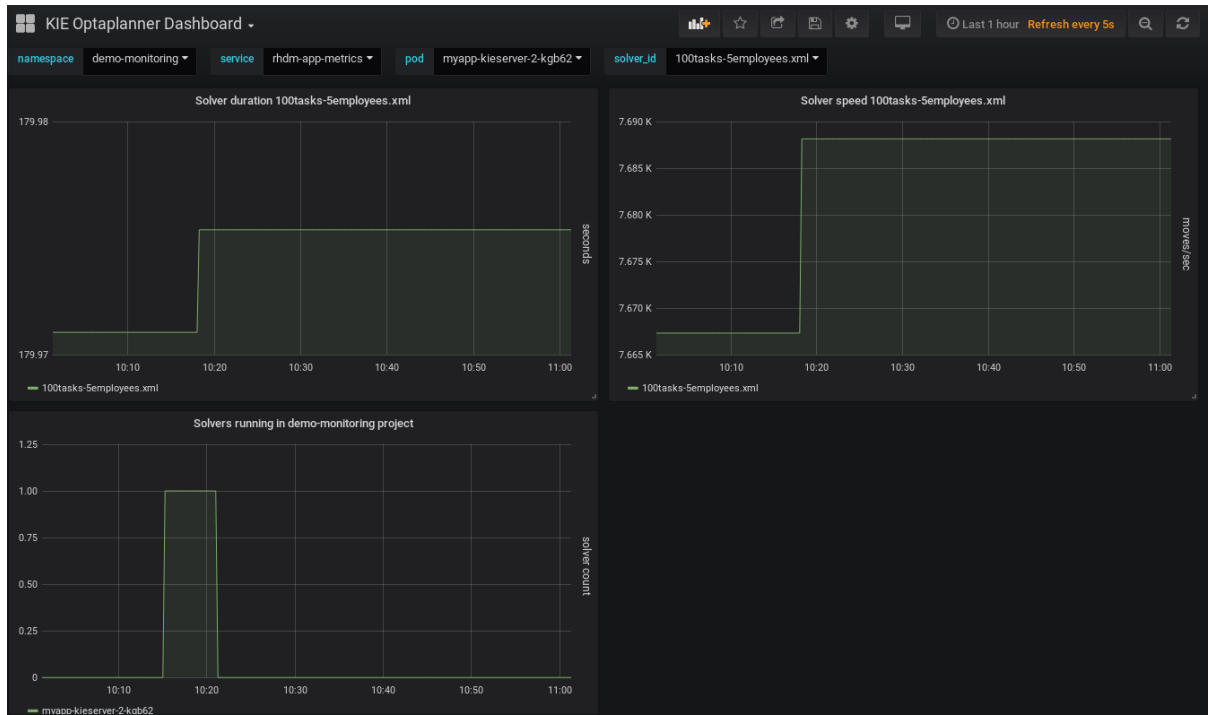
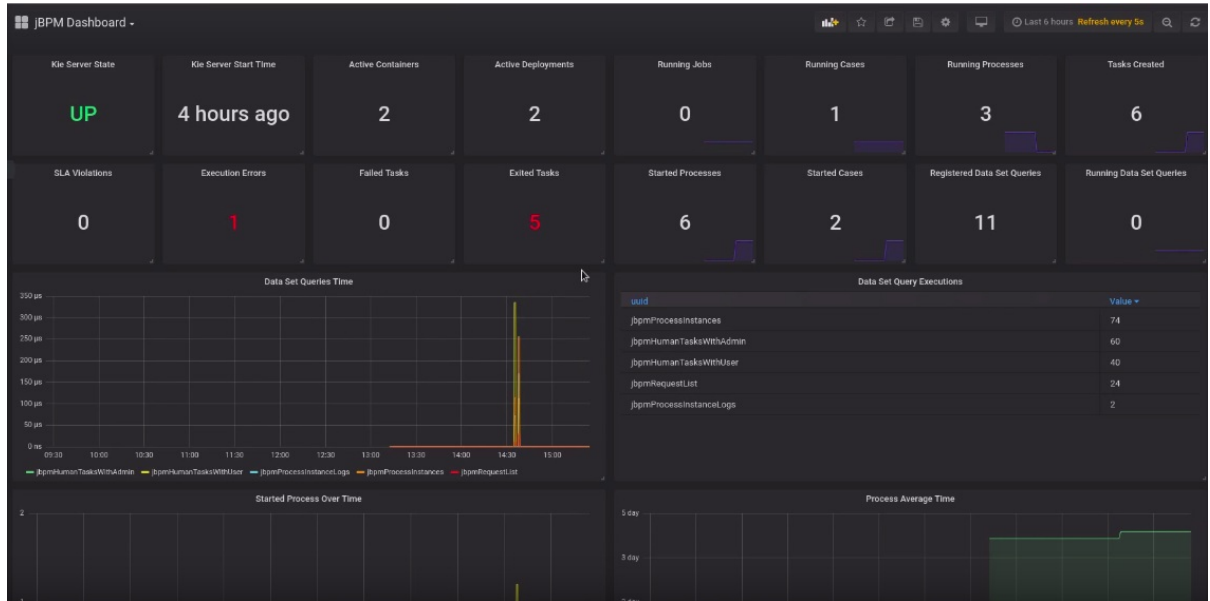


图 15.10. 带有进程、用例和任务的 KIE 服务器指标的 Grafana 仪表盘



其他资源

- [Prometheus Operator](#)
- [Prometheus Operator 入门](#)
- [Prometheus RBAC](#)

- [Prometheus 的 Grafana 支持](#)
- [在 Grafana 中使用 Prometheus](#)
- [Red Hat Process Automation Manager 7.13 产品文档中的 OpenShift部署选项](#)

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

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

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

先决条件

- Prometheus 指标监控是为 KIE 服务器实例配置的。有关内部使用 KIE 服务器 Prometheus 配置的详情，请参考第 15.1 节“[为 KIE 服务器配置 Prometheus 指标监控](#)”。有关 Red Hat OpenShift Container Platform 上使用 KIE 服务器 Prometheus 配置的详情，请参考第 15.2 节“[在 Red Hat OpenShift Container Platform 上为 KIE 服务器配置 Prometheus 指标监控](#)”。

流程

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

示例项目中的 pom.xml 文件示例

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.67.0.Final-redhat-00024</version.org.kie>
</properties>

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

```
</dependency>
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-api</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-services-common</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-services-drools</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-services-prometheus</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-api</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.jbpm</groupId>
  <artifactId>jbpm-services-api</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.jbpm</groupId>
  <artifactId>jbpm-executor</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>org.optaplanner</groupId>
  <artifactId>optaplanner-core</artifactId>
  <version>${version.org.kie}</version>
</dependency>
<dependency>
  <groupId>io.prometheus</groupId>
  <artifactId>simpleclient</artifactId>
  <version>0.5.0</version>
</dependency>
</dependencies>
```

2.

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

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

```

package org.kie.server.ext.prometheus;

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

public class ExampleCustomPrometheusMetricListener implements
DMNRuntimeEventListener {

    private final KieContainerInstance kieContainer;

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

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

    public void beforeEvaluateDecision(BeforeEvaluateDecisionEvent e) {
    }

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

```

```

public void beforeEvaluateBKM(BeforeEvaluateBKMEvent event) {
}

public void afterEvaluateBKM(AfterEvaluateBKMEvent event) {
}

public void beforeEvaluateContextEntry(BeforeEvaluateContextEntryEvent event) {
}

public void afterEvaluateContextEntry(AfterEvaluateContextEntryEvent event) {
}

public void beforeEvaluateDecisionTable(BeforeEvaluateDecisionTableEvent event)
{
}

public void afterEvaluateDecisionTable(AfterEvaluateDecisionTableEvent event) {
}

public void beforeEvaluateDecisionService(BeforeEvaluateDecisionServiceEvent
event) {
}

public void afterEvaluateDecisionService(AfterEvaluateDecisionServiceEvent event)
{
}
}

```

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

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

3.

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

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

```

package org.kie.server.ext.prometheus;

import org.jbpm.executor.AsynchronousJobListener;

```



```

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

public class MyPrometheusMetricsProvider implements PrometheusMetricsProvider {

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

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

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

    public AsynchronousJobListener createAsynchronousJobListener() {
        return null;
    }

    public DeploymentEventListener createDeploymentEventListener() {
        return null;
    }
}

```

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

4. 要使新指标提供程序可供 KIE 服务器发现，请在文件中创建一个 `META-INF/services/org.kie.services.prometheus.PrometheusMetricsProvider` 文件，并在文件中添加完全限定的类名称。在本例中，该文件包含一行 `org.kie.server.ext.prometheus.MyPrometheusMetricsProvider`。
5. 构建您的项目，并将生成的 JAR 文件复制到项目的 `~/kie-server.war/WEB-INF/lib` 目录中。例如，在红帽 JBoss EAP 上，此目录的路径是 `EAP_HOME/standalone/deployments/kie-server.war/WEB-INF/lib`。

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

6.

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

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

第 16 章 配置 OPENSIFT 连接超时

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

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

所有 Business Central 模板都已经包括扩展超时配置。

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

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

有关全局路由特定超时注解的完整列表，请参阅 [OpenShift 文档](#)。

第 17 章 持久性

二进制持久性或编组，将进程实例的状态转换为二进制数据集。二进制持久性是用于永久存储和检索信息的机制。相同的机制也应用于会话状态和工作项目状态。

启用进程实例的持久性时：

- **Red Hat Process Automation Manager 将进程实例信息转换为二进制数据。出于性能的原因，使用自定义序列化而不是 Java 序列化。**
- **二进制数据与其他进程实例元数据一起存储，如进程实例 ID、进程 ID 以及进程启动日期。**

会话也可以存储其他形式的状态，如计时器作业状态或业务规则评估所需的数据。会话状态作为二进制数据集单独保存，以及会话和元数据的 ID。您可以通过重新加载具有给定 ID 的会话来恢复会话状态。使用 `ksession.getId()` 获取会话 ID。

在配置了持久性时，Red Hat Process Automation Manager 将保留以下内容：

- **会话状态**：这包括会话 ID、上次修改日期、业务规则需要评估的会话数据、计时器作业状态。
- **进程实例状态**：这包括进程实例 ID、进程 ID、上次修改日期、上次读取访问日期、进程实例启动日期、运行时数据（包括正在执行的节点、变量值和其他进程实例数据）以及事件类型。
- **工作项目 运行时状态**：包括工作项目 ID、创建日期、名称、进程实例 ID 以及工作项目状态本身。

根据持久性数据，您可以在故障时恢复所有正在运行的进程实例的执行状态，或者从内存中临时删除正在运行的实例，并在以后恢复它们。

17.1. 配置 KIE 服务器持久性

您可以通过传递 `Hibernate` 或 `JPA` 参数作为系统属性来配置 KIE 服务器持久性。

KIE 服务器可以识别带有以下前缀的系统属性，您可以使用带有这些前缀的每个 Hibernate 或 JPA 参数：

- `javax.persistence`
- `hibernate`

流程

1. 要配置 KIE 服务器持久性，请完成以下任一任务：

如果要使用红帽 JBoss EAP 配置文件配置 KIE 服务器持久性，请完成以下任务：

- i. 在 Red Hat Process Automation Manager 安装目录中，导航至 `standalone-full.xml` 文件。例如，如果您使用红帽 JBoss EAP 安装红帽流程自动化管理器，请转至 `$EAP_HOME/standalone/configuration/standalone-full.xml` 文件。
- ii. 打开 `standalone-full.xml` 文件并在 `< system-properties >` 标签下，将 Hibernate 或 JPA 参数设置为系统属性。

使用 Hibernate 参数配置 KIE 服务器持久性示例

```
<system-properties>
...
  <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
...
</system-properties>
```

使用 JPA 参数配置 KIE 服务器持久性示例

```
<system-properties>
...
  <property name="javax.persistence.jdbc.url"
```

```
value="jdbc:mysql://mysql.db.server:3306/my_database?  
useSSL=false&serverTimezone=UTC"/>  
...  
<system-properties>
```

如果要使用命令行配置 KIE 服务器持久性，请完成以下任务：

i.

使用 **-Dkey=value** 直接从命令行中传递参数，如下所示：

使用 **Hibernate** 参数配置 KIE 服务器持久性示例：

```
$EAP_HOME/bin/standalone.sh -Dhibernate.hbm2ddl.auto=create-drop
```

使用 **JPA** 参数配置 KIE 服务器持久性示例：

```
$EAP_HOME/bin/standalone.sh -  
Djavax.persistence.jdbc.url=jdbc:mysql://mysql.db.server:3306/my_database?  
useSSL=false&serverTimezone=UTC
```

17.2. 配置安全点

要允许持久性，请将 **jbpm-persistence** JAR 文件添加到应用的类路径中，并将进程引擎配置为使用持久性。当进程引擎达到安全点时，流程引擎会自动将运行时状态存储在存储中。

安全点是进程实例已暂停的位置。当进程实例调用达到进程引擎中的安全点时，进程引擎将对进程实例的任何更改存储为进程运行时数据的快照。但是，当完成进程实例时，进程运行时数据的持久快照将被自动删除。

BPMN2 安全点节点确保进程引擎在执行停止和提交事务时保存进程定义的状态。以下 **BPMN2 节点**被视为安全点：

- 所有中间 **CATCH 事件**
 - **timer Intermediate 事件**
 - **错误 Intermediate 事件**
 - **条件 Intermediate 事件**
 - **compensation Intermediate 事件**
 - **信号 Intermediate 事件**
 - **升级 Intermediate 事件**
 - **message Intermediate 事件**
- **用户任务**
- **自定义（由用户定义）服务任务，不完成处理程序中的任务**

如果发生失败且需要从存储恢复进程引擎运行时，进程实例会自动恢复，并恢复其执行，因此不需要手动重新加载并触发进程实例。

考虑将运行时持久性数据视为进程引擎的内部。您不应该访问持久的运行时数据或直接修改，因为这可能会意外副作用。

有关当前执行状态的更多信息，请参阅历史记录日志。仅在绝对必要时查询数据库以便运行时数据。

17.3. 会话持久性实体

会话被保留为 **SessionInfo** 实体。这些保留了运行时 KIE 会话的状态，并存储以下数据：

表 17.1. SessionInfo

字段	描述	nullable
id	主密钥。	不是 NULL
lastModificationDate	实体保存到数据库的最后一次时间。	
rulesByteArray	会话状态。	不是 NULL
startDate	会话启动时间。	
OPTLOCK	带有锁定值的 version 字段。	

17.4. 处理实例持久性实体

进程实例将保留为 **ProcessInstanceInfo** 实体，它会在运行时保留进程实例状态并存储以下数据：

表 17.2. ProcessInstanceInfo

字段	描述	nullable
instanceId	主密钥。	不是 NULL
lastModificationDate	实体保存到数据库的最后一次时间。	
lastReadDate	实体从数据库检索的时间。	
processId	进程的 ID。	
processInstanceByteArray	以二进制数据集形式的进程实例的状态。	不是 NULL
startDate	进程的开始时间。	
state	代表进程实例状态的整数。	不是 NULL
OPTLOCK	带有锁定值的 version 字段。	

ProcessInstanceInfo 具有 1:N 与 **EventTypes** 实体的关系。

EventTypes 实体包含以下数据：

表 17.3. EventTypes

字段	描述	nullable
instanceId	对此列上的 ProcessInstanceInfo 主键和外键约束的引用。	不是 NULL
元素	此过程中的已完成事件。	

17.5. 工作项目持久性实体

工作项目保留为 **WorkItemInfo** 实体，该实体在运行时保留特定工作项目实例的状态并存储了以下数据：

表 17.4. WorkItemInfo

字段	描述	nullable
workItemId	主密钥。	不是 NULL
name	工作项目的名称。	
processInstanceId	进程的（主要密钥）ID。此字段没有外键限制。	不是 NULL
state	工作项目的状态。	不是 NULL
OPTLOCK	带有锁定值的 version 字段。	
workitembytearray	工作项目状态作为二进制数据集。	不是 NULL

17.6. 关联关键实体

CorrelationKeyInfo 实体包含有关分配给给定进程实例的关联密钥的信息。这个表是可选的。仅在需要关联功能时使用。

表 17.5. CorrelationKeyInfo

字段	描述	nullable
keyId	主密钥。	不是 NULL
name	分配的关联密钥名称。	
processInstanceId	分配给关联密钥的进程实例的 ID。	不是 NULL
OPTLOCK	带有锁定值的 version 字段。	

CorrelationPropertyInfo 实体包含有关分配给进程实例的关联密钥的关联属性的信息。

表 17.6. CorrelationPropertyInfo

字段	描述	nullable
attributeId	主密钥。	不是 NULL
name	属性的名称。	
value	属性的值。	不是 NULL
OPTLOCK	带有锁定值的 version 字段。	
correlationKey_keyId	映射到关联密钥的外键。	不是 NULL

17.7. 上下文映射实体

ContextMappingInfo 实体包含有关映射到 **KieSession** 的上下文信息的信息。这是 **RuntimeManager** 的内部部分，在使用 **RuntimeManager** 时可被视为可选。

表 17.7. ContextMappingInfo

字段	描述	nullable
mappingId	主密钥。	不是 NULL
CONTEXT_ID	上下文标识符。	不是 NULL
KSESSION_ID	KieSession 标识符。	不是 NULL
OPTLOCK	带有锁定值的 version 字段。	

字段	描述	nullable
OWNER_ID	包含给定映射关联的部署单元的标识符	

17.8. 重要的锁定支持

进程持久性的默认锁定机制 *是理想的选择*。通过多线程高并发到同一进程实例，此锁定策略可能会导致性能不良。

17.8.1. 通过代码配置重要的锁定支持

您可以为每个进程设置锁定机制，并允许它成为重要的。您还可以在每个 **KIE Session level** 或 **Runtime Manager** 级别进行修改，而不仅仅在进程级别。

要设置进程使用 **pessimistic** 锁定，请在运行时环境中使用以下配置：

```
import org.kie.api.runtime.Environment;
import org.kie.api.runtime.EnvironmentName;
import org.kie.api.runtime.manager.RuntimeManager;
import org.kie.api.runtime.manager.RuntimeManagerFactory;

...

env.set(EnvironmentName.USE_PESSIMISTIC_LOCKING, true); ❶

RuntimeManager manager =
RuntimeManagerFactory.Factory.get().newPerRequestRuntimeManager(environment); ❷
```

❶

env 是 **org.kie.api.runtime.Environment** 的实例。

❷

使用此环境创建 **Runtime Manager**。

17.8.2. 在 **Business Central** 中配置伪装锁定支持

业务中心支持对流程的明确锁定。要设置流程在 **Business Central** 中使用 **pessimistic** 锁定，请使用以下步骤：

先决条件

- 在 **Business Central** 中有足够的用户权限。
- 您已创建了 **Business Central** 项目。

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Projects**。
2. 选择您的项目。
3. 单击 **Settings** 选项卡，以访问项目设置。
4. 进入 **Deployments** → **Environment** 条目。
5. 点 **Add Environment Entry**。
6. 要添加环境条目，请输入以下值：
 - 名称：输入您的环境名称。
 - 值：输入您的环境值。
 - 解析器类型：根据需要选择 **MVEL**、**Reflection** 或 **Spring resolver** 类型。
 - 参数：添加参数。
7. 单击 **Save**，然后再次单击 **Save** 以确认更改。

17.9. 在 RED HAT PROCESS AUTOMATION MANAGER 中以独立数据库模式持久保留进程变量

当您创建要在您定义的进程中使用的进程变量时，红帽流程自动化管理器将这些进程变量存储在默认数据库架构中。您可以在单独的数据库架构中保留进程变量，以便在维护和实施流程数据方面具有更大的灵活性。

例如，在您的独立数据库架构中保留进程变量可以帮助您执行以下任务：

- 以人类可读格式维护进程变量
- 将变量提供给 **Red Hat Process Automation Manager** 之外的服务
- 清除 **Red Hat Process Automation Manager** 中默认数据库表的日志，而不丢失进程变量数据



注意

此流程只适用于处理变量。此过程不适用于问题单变量。

先决条件

- 您已在 **Red Hat Process Automation Manager** 中定义了您要在其中实施变量的进程。
- 如果要将变量保存在 **Red Hat Process Automation Manager** 之外的数据库架构中，您可以创建一个数据源和您要使用的独立数据库 **schema**。有关创建数据源的详情，请参考 [配置 Business Central 设置和属性](#)。

流程

1. 在用作进程变量的数据对象文件中，添加以下元素来配置变量持久性：

为变量持久性配置 **Person.java** 对象示例

`@javax.persistence.Entity` **1**

```
@javax.persistence.Table(name = "Person") 2  
public class Person extends org.drools.persistence.jpa.marshaller.VariableEntity 3  
implements java.io.Serializable { 4  
  
    static final long serialVersionUID = 1L;  
  
    @javax.persistence.GeneratedValue(strategy =  
javax.persistence.GenerationType.AUTO, generator = "PERSON_ID_GENERATOR")  
    @javax.persistence.Id 5  
    @javax.persistence.SequenceGenerator(name = "PERSON_ID_GENERATOR",  
sequenceName = "PERSON_ID_SEQ")  
    private java.lang.Long id;  
  
    private java.lang.String name;  
  
    private java.lang.Integer age;  
  
    public Person() {  
    }  
  
    public java.lang.Long getId() {  
    return this.id;  
    }  
  
    public void setId(java.lang.Long id) {  
    this.id = id;  
    }  
  
    public java.lang.String getName() {  
    return this.name;  
    }  
  
    public void setName(java.lang.String name) {  
    this.name = name;  
    }  
  
    public java.lang.Integer getAge() {  
    return this.age;  
    }  
  
    public void setAge(java.lang.Integer age) {  
    this.age = age;  
    }  
  
    public Person(java.lang.Long id, java.lang.String name,  
    java.lang.Integer age) {  
    this.id = id;  
    this.name = name;  
    this.age = age;  
    }  
  
    }
```

1

将数据对象配置为持久性实体。

2

定义用于 **data** 对象的数据库表名称。

3

创建单独的 **MappedVariable** 映射表，用于维护此数据对象和相关进程实例之间的关系。如果您不需要此关系，则不需要扩展 **VariableEntity** 类。如果没有此扩展，数据对象仍会保留，但没有包含额外的数据。

4

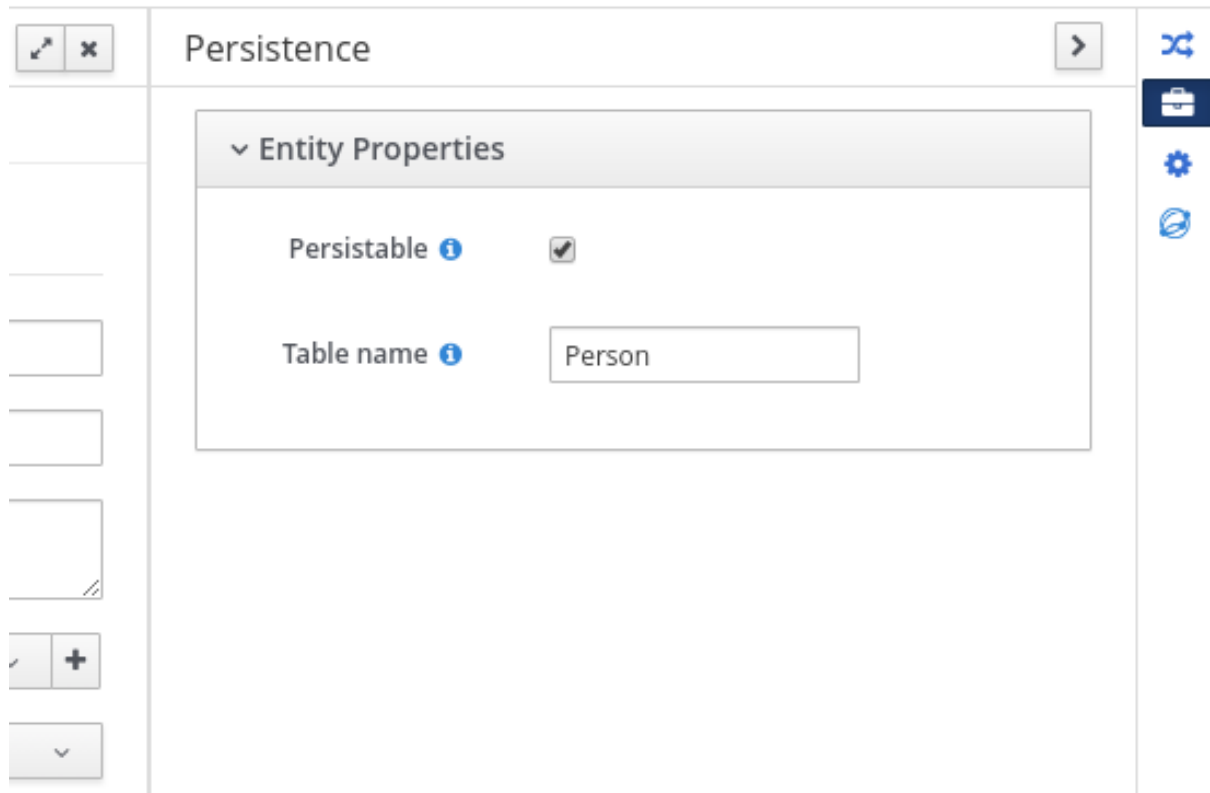
将数据对象配置为 **serializable** 对象。

5

为对象设置持久性 **ID**。

要使用 **Business Central** 使数据对象持久，请导航到项目中的 **data** 对象文件，单击窗口右上角的 **Persistence** 图标，并配置持久性行为：

图 17.1. Business Central 中的持久性配置



2.

在项目的 **pom.xml** 文件中，添加以下用于持久性支持的依赖关系。这个依赖关系包含您在 **data** 对象中配置的 **VariableEntity** 类。

持久性的项目依赖性

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-persistence-jpa</artifactId>
  <version>${rhpm.version}</version>
  <scope>provided</scope>
</dependency>
```

3.

在项目的 **~/META-INF/kie-deployment-descriptor.xml** 文件中，配置 **JPA marshalling** 策略以及要与 **marshaller** 搭配使用的持久性单元。对于定义为实体的对象，需要 **JPA marshalling** 策略和持久性单元。

JPA marshaller 和 **persistence** 单元在 **kie-deployment-descriptor.xml** 文件中配置


```

<marshalling-strategy>
  <resolver>mvel</resolver>
  <identifier>new
org.drools.persistence.jpa.marshaller.JPAPlaceholderResolverStrategy("myPersistenceUnit", classLoader)</identifier>
  <parameters/>
</marshalling-strategy>

```

4.

在项目的 `~/META-INF` 目录中，创建一个 `persistence.xml` 文件，用于指定您要保留的进程变量：

带有数据源配置的 `persistence.xml` 文件示例

```

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:orm="http://java.sun.com/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd
  http://java.sun.com/xml/ns/persistence/orm
  http://java.sun.com/xml/ns/persistence/orm_2_0.xsd">
  <persistence-unit name="myPersistenceUnit" transaction-type="JTA">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source> 1
    <class>org.space.example.Person</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect"/>
      <property name="hibernate.max_fetch_depth" value="3"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.id.new_generator_mappings" value="false"/>
      <property name="hibernate.transaction.jta.platform"
value="org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform"/>
    </properties>
  </persistence-unit>
</persistence>

```

1

设置进程变量保留的数据源

要使用 **Business Central** 配置 **marshalling** 策略、持久性单元和数据源，请导航至项目 **Settings** → **Deployments** → **Marshalling Strategy** 和 **Project Settings** → **Persistence** :

图 17.2. JPA marshaller 配置 Business Central

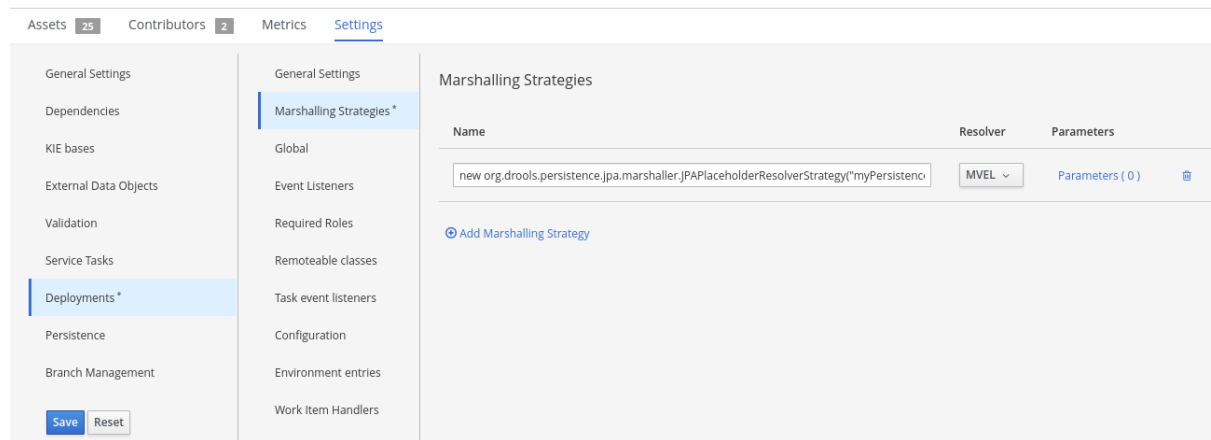
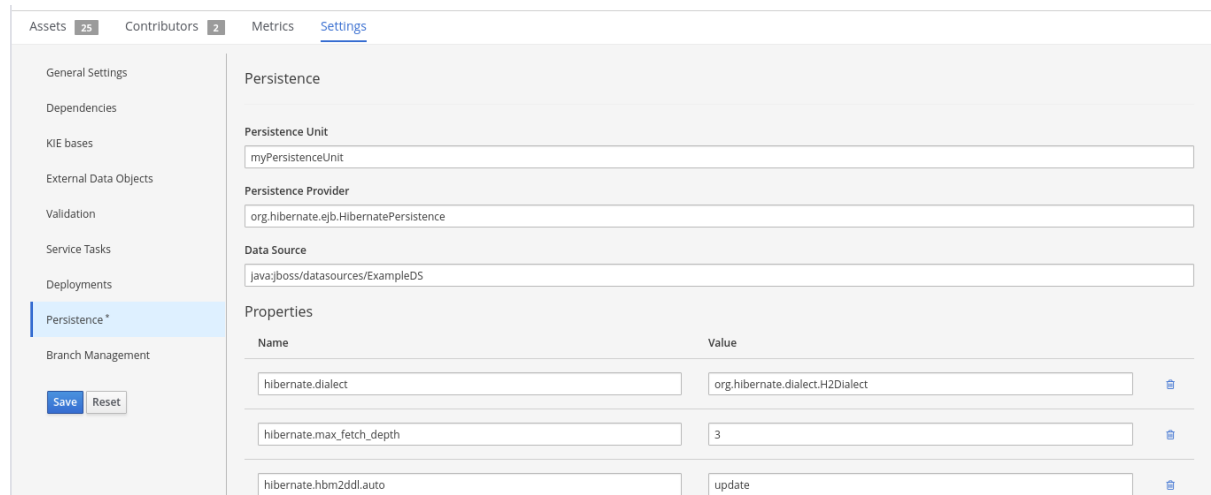


图 17.3. Business Central 中的持久性单元和数据源配置



第 18 章 定义 LDAP 登录域

当您设置 **Red Hat Process Automation Manager** 以使用 **LDAP** 进行身份验证和授权时，定义 **LDAP** 登录域，因为 **Git SSH** 身份验证可能会使用另一个安全域。

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

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

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

第 19 章 通过 RH-SSO 验证第三方客户端

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

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

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

您的客户端可以使用以下选项之一通过 **RH-SSO** 验证：

- 基本身份验证（如果客户端支持）
- 基于令牌的身份验证

19.1. 基本身份验证 (BASIC AUTHENTICATION)

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

- 对于基于 **Web** 的远程软件仓库端点：

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

- 对于 **KIE 服务器**：

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

第 20 章 KIE 服务器系统属性

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

表 20.1. 禁用 KIE 服务器扩展的系统属性

属性	值	默认	描述
<code>org.drools.server.ext.disabled</code>	true,false	false	如果设为 true ，则禁用商业规则管理(BRM)支持（例如，规则支持）。
<code>org.jbpm.server.ext.disabled</code>	true,false	false	如果设置为 true ，则禁用 Red Hat Process Automation Manager 支持（例如，进程支持）。
<code>org.jbpm.ui.server.ext.disabled</code>	true,false	false	如果设置为 true ，则禁用 Red Hat Process Automation Manager UI 扩展。
<code>org.jbpm.case.server.ext.disabled</code>	true,false	false	如果设置为 true ，则禁用 Red Hat Process Automation Manager 问题单管理扩展。
<code>org.optaplanner.server.ext.disabled</code>	true,false	false	如果设置为 true ，则禁用红帽构建的 OptaPlanner 支持。
<code>org.kie.prometheus.server.ext.disabled</code>	true,false	true	如果设置为 true ，则禁用 Prometheus 服务器扩展。
<code>org.kie.scenariosimulation.server.ext.disabled</code>	true,false	true	如果设置为 true ，则禁用测试场景服务器扩展。
<code>org.kie.dmn.server.ext.disabled</code>	true,false	false	如果设置为 true ，则禁用 KIE Server DMN 支持。
<code>org.kie.swagger.server.ext.disabled</code>	true,false	false	如果设置为 true ，则禁用 KIE 服务器 swagger 文档支持



注意

下表列出了一些进程自动化管理器控制器属性已标记为必需。在 **Business Central** 中创建或删除 KIE 服务器容器时，设置这些属性。如果您在没有与 **Business Central** 交互的情况下单独使用 KIE 服务器，则不需要设置必要的属性。

表 20.2. 进程自动化管理器控制器所需的系统属性

属性	值	默认	描述
org.kie.server.id	字符串	N/A	要分配给服务器的任意 ID。如果在 Business Central 之外配置了无头进程自动化管理器控制器，则这是服务器连接到无头进程自动化管理器控制器来获取 KIE 容器配置的 ID。如果没有提供，则会自动生成 ID。
org.kie.server.user	字符串	kieserver	在托管模式下运行时，用于从 Process Automation Manager 控制器连接 KIE 服务器的用户名。在 Business Central 系统属性中设置此属性。在使用流程自动化管理器控制器时设置此属性。
org.kie.server.pwd	字符串	kieserver1!	用于从 Process Automation Manager 控制器与 KIE 服务器连接的密码，需要在受管模式下运行时。在 Business Central 系统属性中设置此属性。在使用流程自动化管理器控制器时设置此属性。
org.kie.server.token	字符串	N/A	属性，允许您在 Process Automation Manager 控制器和 KIE 服务器（而不是基本用户名和密码身份验证）之间使用基于令牌的身份验证。Process Automation Manager 控制器会在请求标头中发送令牌作为参数。服务器需要长期提供的访问令牌，因为令牌没有刷新。
org.kie.server.location	URL	N/A	Process Automation Manager 控制器用来调用这个服务器的 KIE 服务器实例的 URL，例如 http://localhost:8230/kie-server/services/rest/server 。在使用进程自动化管理器控制器时，需要设置此属性。
org.kie.server.controller	逗号分隔列表	N/A	进程自动化管理器控制器 REST 端点的以逗号分隔的 URL 列表，例如 http://localhost:8080/business-central/rest/controller 。在使用进程自动化管理器控制器时，需要设置此属性。
org.kie.server.controller.user	字符串	kieserver	用于连接 Process Automation Manager 控制器 REST API 的用户名。在使用进程自动化管理器控制器时，需要设置此属性。
org.kie.server.controller.pwd	字符串	kieserver1!	连接到 Process Automation Manager 控制器 REST API 的密码。在使用进程自动化管理器控制器时，需要设置此属性。

属性	值	默认	描述
org.kie.server.controller.token	字符串	N/A	属性允许您在 KIE 服务器和流程自动化管理器控制器之间使用基于令牌的身份验证，而不是基本用户名和密码身份验证。服务器在请求标头中将令牌作为参数发送。服务器需要长期提供的访问令牌，因为令牌没有刷新。
org.kie.server.controller.connect	Long	10000	在服务器启动时，重复尝试将 KIE 服务器连接到 Process Automation Manager 控制器之间的等待时间（毫秒）。

表 20.3. 持久性系统属性

属性	值	默认	描述
org.kie.server.persistence.ds	字符串	N/A	数据源 JNDI 名称。在启用 BPM 支持时设置此属性。
org.kie.server.persistence.tm	字符串	N/A	Hibernate 属性的事务管理器平台。在启用 BPM 支持时设置此属性。
org.kie.server.persistence.dialect	字符串	N/A	使用 Hibernate 电源。在启用 BPM 支持时设置此属性。
org.kie.server.persistence.schema	字符串	N/A	要使用的数据库架构。

表 20.4. *executor* 系统属性

属性	值	默认	描述
org.kie.executor.interval	整数	0	Red Hat Process Automation Manager executor 完成了一个作业的时间，它在 org.kie.executor.timeunit 属性中指定的一个时间窗内启动一个新作业。
org.kie.executor.timeunit	java.util.concurrent.TimeUnit constant	秒	指定 org.kie.executor.interval 属性的时间范围。
org.kie.executor.pool.size	整数	1	Red Hat Process Automation Manager 执行程序使用的线程数量。
org.kie.executor.retry.count	整数	3	在失败的作业上重试 Red Hat Process Automation Manager 执行程序尝试的数量。

属性	值	默认	描述
<code>org.kie.executor.jms.queue</code>	字符串	队列/KIE.SERVER.EXECUTOR	KIE 服务器的作业执行器 JMS 队列。
<code>org.kie.executor.jms.jobHeader</code>	<code>true,false</code>	<code>false</code>	如果设置为 <code>true</code> ，则 JMS 标头中的请求标识符包含为 <code>jobId</code> 属性。
<code>org.kie.executor.disabled</code>	<code>true,false</code>	<code>false</code>	如果设置为 <code>true</code> ，则禁用 KIE 服务器执行器。

表 20.5. 人工任务系统属性

属性	值	默认	描述
<code>org.jbpm.ht.callback</code>	mvel ldap db jaas props custom	jaas	指定要使用的用户组回调实现的属性： <ul style="list-style-type: none"> • MVEL : Default ; 主要用于测试。 • LDAP : LDAP; 需要 <code>jbpm.usergroup.callback.properties</code> 文件中的其他配置。 • db: Database ; 需要 <code>jbpm.usergroup.callback.properties</code> 文件中的其他配置。 • JAAS : Jackstack ; 委派至容器，以获取有关用户数据的信息。 • props : 简单属性文件 ; 需要额外文件来保留所有信息 (用户和组)。 • Custom : 自定义实施 ; 指定 <code>org.jbpm.ht.custom.callback</code> 属性中类的完全限定名称。
<code>org.jbpm.ht.custom.callback</code>	完全限定名称	N/A	如果 <code>org.jbpm.ht.callback</code> 属性设置为 <code>custom</code> ，则 <code>UserGroupCallback</code> 接口的自定义实现。

属性	值	默认	描述
org.jbpm.task.cleanup.enabled	true,false	true	启用任务清理作业监听程序，在进程实例完成后删除任务。
org.jbpm.task.bam.enabled	true,false	true	启用任务 BAM 模块来存储任务相关信息。
org.jbpm.ht.admin.user	字符串	Administrator	可以访问 KIE 服务器中的所有任务的用户。
org.jbpm.ht.admin.group	字符串	管理员	用户必须属于的组才能查看 KIE 服务器中的所有任务。

表 20.6. 加载密钥存储的系统属性

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

表 20.7. 重试提交事务的系统属性

属性	值	默认	描述
org.kie.optlock.retries	整数	5	此属性描述了进程引擎永久重试事务的次数。
org.kie.optlock.delay	整数	50	第一次重试前的延迟时间（以毫秒为单位）。

属性	值	默认	描述
org.kie.optlock.delayFactor	整数	4	每次后续重试增加延迟时间的倍数。使用默认值时，进程引擎在第一个重试前等待 50 毫秒，在第二个重试前等待 200 毫秒，即第三个重试前 800 毫秒，以此类推。

表 20.8. 其他系统属性

属性	值	默认	描述
kie.maven.settings.custom	路径	N/A	Maven 配置的自定义 settings.xml 文件的位置。
kie.server.jms.queues.response	字符串	队列/ KIE.SERVER.RESPONSE	JMS 的响应队列 JNDI 名称。
org.drools.server.filter.classes	true,false	false	当设置为 true 时，CRI KIE 服务器扩展接受由 XmlRootElement 或 Remotable 注解标注的自定义类。
org.kie.server.bypass.auth.user	true,false	false	通过这个属性，您可以绕过经过身份验证的用户以获取与任务相关的操作，如查询。
org.jbpm.rule.task.firelimit	整数	10000	此属性指定执行规则的最大数量，以避免规则在无限循环中运行，并使服务器完全无响应。
org.jbpm.ejb.timer.local.cache	true,false	true	此属性关闭 EJB 计时器本地缓存。
org.kie.server.domain	字符串	N/A	用于使用 JMS 验证用户身份的 JAAS LoginContext 域。
org.kie.server.repo	路径	.	存储 KIE 服务器状态文件的位置。

属性	值	默认	描述
<code>org.kie.server.sync.deploy</code>	<code>true,false</code>	<code>false</code>	<p>指定 KIE 服务器保存部署的属性，直到 Process Automation Manager 控制器提供容器部署配置为止。此属性仅影响在受管模式下运行的服务器。可用的选项如下：</p> <p>* false：与进程自动化管理器控制器的连接是异步的。应用程序启动，连接到流程自动化管理器控制器，并成功部署容器。即使容器可用前，应用程序也会接受请求。* true：服务器应用程序的部署加入与主部署相关的流程自动化管理器控制器连接线程，并等待其完成。这个选项可能会导致在更多应用程序位于同一服务器上时潜在的死锁。在一个服务器实例中只使用一个应用。</p>
<code>org.kie.server.startup.strategy</code>	<code>ControllerBasedStartupStrategy, LocalContainersStartupStrategy</code>	<code>ControllerBasedStartupStrategy</code>	KIE 服务器的启动策略，用于控制部署的 KIE 容器以及部署它们的顺序。
<code>org.kie.server.mgmt.api.disabled</code>	<code>true,false</code>	<code>false</code>	当设置为 true 时，禁用 KIE 服务器管理 API。
<code>org.kie.server.xstream.enabled.packages</code>	Java 软件包，如 <code>org.kie.example</code> 。您还可以指定通配符表达式，如 <code>org.kie.example.*</code> 。	N/A	一个属性，用于指定允许使用 XStream 进行总结的附加软件包。
<code>org.kie.store.services.classes</code>	字符串	<code>org.drools.persistence.jpa.KnowledgeStoreServiceImpl</code>	实施 <code>KieStoreServices</code> 的类的完全限定名称，负责引导 <code>KieSession</code> 实例。
<code>org.kie.server.strict.id.format</code>	<code>true,false</code>	<code>false</code>	使用 JSON 总结时，如果属性设置为 true ，它始终会以正确的 JSON 格式返回响应。例如，如果原始响应仅包含一个数字，则响应将以 JSON 格式嵌套。例如， <code>{"value": 1}</code> 。

属性	值	默认	描述
org.kie.server.json.customObjectDeserializerCNFEBehavior	忽略、警告，但除外	IGNORE	<p>虽然使用 JSON unmarshalling 时，如果没有找到有效负载中的类，但可使用此属性来更改此行为，如下所示：</p> <ul style="list-style-type: none"> ● 如果属性设置为 IGNORE，则有效负载将转换为 HashMap ● 如果属性设置为 WARN，则有效负载将转换为 HashMap 并记录警告 ● 如果属性设为 EXCEPTION，则 KIE 服务器会抛出异常
org.kie.server.strict.jaxb.format	true,false	false	<p>当此属性的值设置为 true 时，KIE 服务器在 REST API 有效负载中验证数据类型。例如，如果数据字段具有数字数据类型并包含数字以外的内容，您将收到错误。</p>

第 21 章 KIE 服务器功能和扩展

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

表 21.1. KIE 服务器功能和扩展

功能名称	扩展名称	描述
KieServer	KieServer	提供 KIE 服务器的核心功能，如在服务器实例上创建和处理 KIE 容器
BRM	grafana	提供商业规则管理(BRM)功能，如插入事实和执行业务规则
BPM	jBPM	提供业务流程管理(BPM)功能，如管理用户任务和执行业务流程
BPM-UI	jBPM-UI	提供与业务流程相关的附加用户界面功能，如在进程图中呈现 XML 表单和 SVG 镜像
CaseMgmt	case-Mgmt	为业务流程提供案例管理功能，比如管理问题单定义和里程碑
BRP	OptaPlanner	提供业务资源规划(BRP)功能，比如实施解决者
DMN	DMN	提供 Decision Model 和 Notation(DMN)功能，如管理 DMN 数据类型和执行业务模型
Swagger	Swagger	提供 Swagger web-interface 功能与 KIE 服务器 REST API 交互

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

KIE 服务器信息的 **GET** 请求的基本 URL

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

使用 **KIE 服务器**信息的 **JSON** 响应示例

```
{
```

```

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

```

要启用或禁用 KIE 服务器扩展，请配置相关的 `*.server.ext.disabled` KIE 服务器系统属性。例如，要禁用 BRM 功能，请设置系统属性 `org.drools.server.ext.disabled=true`。有关所有 KIE 服务器系统属性，请参阅 [第 20 章 KIE 服务器系统属性](#)。

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

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

器功能适应您的业务需求。



重要

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

21.1. 使用自定义 REST API 端点扩展现有 KIE 服务器功能

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

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

自定义 REST API 端点示例

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

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

流程

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

示例项目中的 **pom.xml** 文件示例

```
<packaging>jar</packaging>
<properties>
  <version.org.kie>7.67.0.Final-redhat-00024</version.org.kie>
```

```

</properties>

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

```

2.

在项目中的 **Java** 类中实施

`org.kie.server.services.api.KieServerApplicationComponentsService` 接口，如下例所示：

`KieServerApplicationComponentsService` 接口的实现示例

```

public class CusomtDroolsKieServerApplicationComponentsService implements
KieServerApplicationComponentsService { ❶

    private static final String OWNER_EXTENSION = "Drools"; ❷

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

        RulesExecutionService rulesExecutionService = null; ❹
        KieServerRegistry context = null;

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

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

        return components;
    }
}

```

❶

向应用程序启动时部署的 KIE 服务器基础架构提供 REST 端点。

❷

指定您要扩展的扩展，如本例中的 **Drools** 扩展。

3

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

4

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

5

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

3.

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

CustomResource 类的实现示例

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

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

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

    private RulesExecutionService rulesExecutionService;
    private KieServerRegistry registry;

    public CustomResource() {

    }

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

    // Supported HTTP method, path parameters, and data formats:
```

```

@POST
@Path("/{ksessionId}")
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public Response insertFireReturn(@Context HttpHeaders headers,
    @PathParam("containerId") String id,
    @PathParam("ksessionId") String ksessionId,
    String cmdPayload) {

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

    // Marshalling behavior and supported actions:
    MarshallingFormat format = MarshallingFormat.fromType(contentType);
    if (format == null) {
        format = MarshallingFormat.valueOf(contentType);
    }
    try {
        KieContainerInstance kci = registry.getContainer(id);

        Marshaller marshaller = kci.getMarshaller(format);

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

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

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

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

        String result = marshaller.marshall(results);

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

```

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

- 使用基本端点 **server/containers/instances/{containerId}/ksession**
 - 使用 **POST HTTP** 方法
 - 预期在 **REST** 请求中给出了以下数据：
 - **containerId** 作为路径参数
 - **ksessionId** 作为路径参数
 - 将事实列为消息有效负载
 - 支持所有 **KIE** 服务器数据格式：
 - **XML (JAXB, XStream)**
 - **JSON**
 - **Unmarshals** 在 **List<?>** 集合中，为列表中的每个项目创建一个 **InsertCommand** 实例，后跟 **FireAllRules** 和 **GetObject** 命令。
 - 将所有命令添加到调用决策引擎的 **BatchExecutionCommand** 实例中。
4. 要使新端点可以被 **KIE** 服务器发现，请在文件中创建一个 **META-INF/services/org.kie.server.api.KieServerApplicationApplicationComponentsService** 文件，并添加 **KieApplicationApplicationComponentsService** 实施类的完全限定域名。在本例

中，该文件包含一行

org.kie.server.ext.drools.rest.CusomtDroolsKieServerComponentsService.

5.

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

6.

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

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

在本例中，您可以使用以下信息来调用新端点：

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

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

```

    "name": "mary",
    "age": 22
  }
}
]

```

- 服务器响应示例：**200 (success)**

- 服务器日志输出示例：

```

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

```

21.2. 扩展 KIE 服务器以使用自定义数据传输

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

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

流程

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

示例项目中的 **pom.xml** 文件示例

```

<packaging>jar</packaging>

<properties>
  <version.org.kie>7.67.0.Final-redhat-00024</version.org.kie>
</properties>

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

```

```

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

```

2.

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

KieServerExtension 接口的实现示例

```

public class MinaDroolsKieServerExtension implements KieServerExtension {

```

```

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

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

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

    // Taken from dependency on the `Drools` extension:
    private KieContainerCommandService batchCommandService;

    // Specific to MINA:
    private IoAcceptor acceptor;

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

    public void init(KieServerImpl kieServer, KieServerRegistry registry) {

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

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

            acceptor.setHandler( new TextBasedIoHandlerAdapter(batchCommandService)
);

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

                logger.info("{} -- Mina server started at {} and port {}", toString(),
MINA_HOST, MINA_PORT);
            }
        }
    }

```



```

    } catch (IOException e) {
        logger.error("Unable to start Mina acceptor due to {}", e.getMessage(), e);
    }
}

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

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

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

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

public <T> T getAppComponents(Class<T> serviceType) {
    return null;
}

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

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

public String getExtensionName() {
    return EXTENSION_NAME;
}

public Integer getStartOrder() {
    return 20;
}

@Override
public String toString() {

```

```

    return EXTENSION_NAME + " KIE Server extension";
  }
}

```

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

KieServerExtension 接口概述

```

public interface KieServerExtension {

    boolean isActive();

    void init(KieServerImpl kieServer, KieServerRegistry registry);

    void destroy(KieServerImpl kieServer, KieServerRegistry registry);

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

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

    List<Object> getAppComponents(SupportedTransports type);

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

    String getImplementedCapability(); 1

    List<Object> getServices();

    String getExtensionName(); 2

    Integer getStartOrder(); 3
}

```

1

指定此扩展涵盖的能力。能力在 **KIE** 服务器中必须是唯一的。

2

为扩展名定义人类可读的名称。

3

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

在前面的 **MinaDroolsKieServerExtension** 示例实现中，**init** 方法是从 **Drools** 扩展收集服务的主要元素，用于引导 **MINA** 服务器。**KieServerExtension** 接口中的其他方法都可以与标准实施保持一致，以满足接口要求。

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

3.

为 **MINA** 服务器实施 **TextBasedIoHandlerAdapter** 处理程序，如下例所示：

TextBasedIoHandlerAdapter 处理程序的实现示例

```
public class TextBasedIoHandlerAdapter extends IoHandlerAdapter {

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

    private KieContainerCommandService batchCommandService;

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

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

        String[] elements = completeMessage.split("\\|");
        logger.debug("Container id {}", elements[0]);
        try {
```

```
        ServiceResponse<String> result =
batchCommandService.callContainer(elements[0], elements[1],
MarshallingFormat.JSON, null);

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

    }
}
}
```

在本例中，处理器类接收文本信息，并在 **Drools** 服务中执行它们。

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

- 您提交到处理程序的任何对象都必须是一行，因为每个传入的传输请求都是一行。
 - 您必须在此一行中传递 **KIE 容器 ID**，以便处理程序需要格式 **containerID|payload**。
 - 您可以使用 **marshaller** 生成的方式设置响应。响应可以是多行。
 - 该处理程序支持 **流模式**，允许您发送命令而不与 **KIE 服务器** 会话断开连接。要在流模式下结束 **KIE 服务器** 会话，请将 **exit** 或 **exit** 命令发送至服务器。
4. 要使新数据传输可被 **KIE 服务器** 发现，请在文件中创建一个 **META-INF/services/org.kie.server.api.KieServerExtension** 文件，并在该文件中添加完全限定的类名称。在本例中，该文件包含一行 **org.kie.server.ext.mina.MinastandaloneKieServerExtension**。
 5. 构建您的项目，并将生成的 **JAR** 文件复制到项目的 **mina-core-2.0.9.jar** 文件（此示例中取决于该扩展）到项目的 **~/kie-server.war/WEB-INF/lib** 目录中。例如，在红帽 **JBoss EAP** 上，

此目录的路径是 `EAP_HOME/standalone/deployments/kie-server.war/WEB-INF/lib`。

6.

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

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

服务器日志中的新数据传输

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

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

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

```
telnet 127.0.0.1 9123
```

在命令终端中与 **KIE** 服务器交互示例

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

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

# Server response:
{
  "results" : [ {
```

```

    "key" : "",
    "value" : 1
  }],
  "facts" : []
}

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

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

```

服务器日志输出示例

```

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

```

21.3. 使用自定义客户端 API 扩展 KIE 服务器客户端

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

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

流程

1.

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

示例项目中的 **pom.xml** 文件示例

```
<packaging>jar</packaging>

<properties>
  <version.org.kie>7.67.0.Final-redhat-00024</version.org.kie>
</properties>

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

2.

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

RulesMinaServicesClient 接口示例

```
public interface RulesMinaServicesClient extends RuleServicesClient {
}
```

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

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

3.

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

RulesMinaServicesClient 接口的实现示例

```
public class RulesMinaServicesClientImpl implements RulesMinaServicesClient {

    private String host;
    private Integer port;

    private Marshaller marshaller;

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

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

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

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

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



```

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

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

    // Flatten the content to be single line:
    content = content.replaceAll("\n", "");

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

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

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

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

```

这个实现示例指定了以下数据和行为：

- 使用基于套接字的通讯进行简单性
 - 依赖于 **KIE** 服务器客户端的默认配置，并使用 **ServerUrl** 提供 **MINA** 服务器的主机和端口
 - 将 **JSON** 指定为 **marshalling** 格式
 - 需要接收消息作为以 **open bracket {** 开头的 **JSON** 对象
 - 在等待响应的第一行时，使用直接套接字与阻塞 **API** 通信，然后读取所有可用的行
 - 不要使用 流模式，因此在调用命令后断开 **KIE** 服务器会话
4. 在项目中的 **Java** 类中实施 **org.kie.client.client.helper.KieServicesClientBuilder** 接口，如下例所示：

KieServicesClientBuilder 接口的实现示例

```

public class MinaClientBuilderImpl implements KieServicesClientBuilder { 1

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

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

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

        return services;
    }
}

```

1

可让您向通用 **KIE 服务器客户端基础架构** 提供额外的客户端 **API**

2

定义客户端使用的 **KIE 服务器功能(extension)**

3

提供客户端实施的映射，其中键是接口，值是完全初始化的实现

5.

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

6.

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

7.

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

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

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

创建 **RulesMinaServiceClient** 客户端的实现示例

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

```

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

configuration.addJaxbClasses(extraClasses);

KieServicesClient kieServicesClient =
KieServicesFactory.newKieServicesClient(configuration);

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

return rulesClient;
}

```

通过 MINA 传输调用 KIE 服务器操作的示例配置

```

RulesMinaServicesClient rulesClient = buildClient();

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

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

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

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

String data = response.getResult();

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

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

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

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

```


第 22 章 KIE 服务器的性能调整注意事项

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

确保在开发过程中启用了开发模式

您可以将 **Business Central** 中的 KIE 服务器或特定项目设置为使用 生产 模式或 开发模式。默认情况下，**Business Central** 中的 KIE 服务器和所有新项目均为开发模式。这个模式提供有助于开发体验的功能，如灵活的项目部署策略以及优化开发期间 KIE 服务器性能的功能，如禁用的 **GAV** 检测。使用开发模式，直到 **Red Hat Process Automation Manager** 环境创建并完全可用于生产环境模式。

有关配置环境模式或重复的 **GAV** 检测的详情，请查看以下资源：

- [第 8 章 在 KIE 服务器和 Business Central 中配置环境模式](#)
- [打包和部署 Red Hat Process Automation Manager 项目](#)

根据您的需要调整 KIE 服务器的功能和扩展

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

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

有关启用、禁用或扩展 **KIE** 服务器功能的详情，请参考 [第 21 章 KIE 服务器功能和扩展](#)。

第 23 章 其他资源

- [在 Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Process Automation Manager](#)
- [规划 Red Hat Process Automation Manager 安装](#)
- [在 Red Hat JBoss EAP 7.4 上安装和配置 Red Hat Process Automation Manager](#)
- [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. 配置 **BUSINESS CENTRAL** 设置和属性

作为管理员，您可以在 **admin Settings** 页面中自定义以下内容：

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

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

先决条件

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

第 24 章 用户和组管理

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

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



注意

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

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

24.1. 创建用户

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

数量的用户。



注意

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

流程

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



注意

用户必须至少有一个角色才能访问 **Business Central**。

24.2. 编辑用户

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

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 用户。
2. 在 **All users** 列表中点击您要编辑的用户。用户详情显示在右侧窗格中。
3. 点 **Edit** 执行以下任务：
 - 要更改用户组，点 **Groups** 选项卡，点 **Add to groups**，选择您希望用户所属的组，点 **Add to selected groups**，然后点 **Save**。
 - 要更改用户的角色，请单击 **Roles** 选项卡，单击 **Add roles**，选择您要分配给用户的角色，单击 **Add to selected roles**，然后单击 **Save**。
 - 要查看用户权限，请点击 权限选项卡 并展开属性。
 - 要更改密码，请单击"更改密码"，输入新密码，然后单击"更改"。
 - 若要删除用户，请单击 **Delete**，然后单击 **Yes** 以确认删除。

24.3. 创建组

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



注意

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

流程

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

2. 单击 **New group**，输入组名称，然后单击 **Next**。
3. 选择您要添加到此组的用户，然后点 **Add selected users**。

新创建的组列在 **All groups** 下。

24.4. 编辑组

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

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **组**。
2. 在 **All groups** 列表中点击您要编辑的组。用户详情显示在右侧窗格中。
3. 从 **Home Page** 列表中选择主页。
4. 从优先级列表中选择 **优先级**。
5. 在 **Permissions** 部分中，展开 **resource** 属性并更改其权限。



注意

您可为 **页面、编辑器、空格和 项目权限** 添加例外。

6. 单击 **Save** 以应用更改。

第 25 章 安全管理

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

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

25.1. 安全管理供应商

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

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



注意

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

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

25.1.1. 根据属性文件配置红帽 **JBoss EAP** 安全管理提供程序

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

先决条件

- 安装了红帽 **JBoss EAP**。

流程

1. 要使用红帽 **JBoss EAP** 实例中的现有用户或角色属性，请在

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

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

下表提供了这些属性的描述和默认值：

表 25.1. 基于属性文件的红帽 JBoss EAP 安全管理供应商

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

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

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

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

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

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

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

先决条件

- 安装了红帽 **JBoss EAP**。

流程

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

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

下表提供了这些属性的描述和默认值：

表 25.2. 基于属性文件和 **CLI** 模式的红帽 **JBoss EAP** 安全管理供应商

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

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


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

3.

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

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

25.2. 权限和设置

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

- 查看页面。
- 保存项目。
- 查看存储库。
- 删除仪表板

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

25.2.1. 在 **Business Central** 中更改组和角色的权限

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



注意

对角色或组进行的任何更改会影响与该角色或组关联的所有用户。

先决条件

- 使用 **admin** 用户角色登录到 **Business Central**。

流程

1. 要访问 **Business Central** 中的 安全管理 页面，可选择屏幕右上角的 **Admin** 图标。
2. 单击 **Business Central Settings** 页面中的 **Roles**、**Groups** 或 **Users**。
Security 管理页面 会在您点击的图标的标签页中打开。
3. 在列表中点击您要编辑的角色或组。所有详情都显示在右侧面板中。
4. 在 **Settings** 部分下设置 主页 或 优先级。
5. 在 **Permissions** 部分设置 **Business Central**、页面、编辑器、空格和项目权限。

图 25.1. 设置权限

admin settings

Home Page ⓘ

Priority ⓘ

Permissions

> Workbench ⓘ

Pages ⓘ

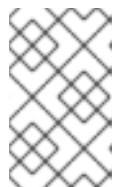
[Add Exception](#)

Editors ⓘ

Spaces ⓘ

Projects ⓘ

6. 单击资源类型旁边的箭头，以展开您要更改权限的资源类型。
7. 可选：要添加资源类型的异常，请单击 **Add Exception**，然后根据需要设置权限。



注意

您不能在 **Business Central** 资源类型中添加例外。

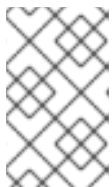
8. 单击 **Save**。

25.2.2. 更改 **Business Central** 主页

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

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Roles** 或 **Groups**。
2. 选择角色或组。
3. 从 **Home Page** 列表中选择一页。
4. 单击 **Save**。



注意

角色或组必须具有对页面的读访问权限，然后才能使它成为主页。

25.2.3. 设置优先级

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

先决条件

- 使用 **admin** 用户角色登录到 **Business Central**。

流程

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

2. 选择角色或组。
3. 从优先级菜单中选择优先级，然后单击 **Save**。



注意

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

第 26 章 工件管理

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

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



注意

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

26.1. 查看工件

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

流程

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

26.2. 下载工件

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

流程

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

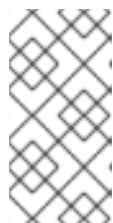
3. 浏览要保存工件的目录。
4. 点击 **Save**。

26.3. 上传工件

您可以将工件从本地存储上传到 **Business Central** 中的项目。

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，再选择 **Artifacts**。
2. 点 **Upload**。
3. 点 **Choose File** 并浏览到您要上传工件的目录。
4. 点 **Upload**。



注意

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

第 27 章 数据源和数据库驱动程序管理

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

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



注意

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

27.1. 添加数据源

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

流程

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

New data source 窗口将打开。

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

27.2. 编辑数据源

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

流程

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

27.3. 删除数据源

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

流程

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

27.4. 添加数据库驱动程序

您可以在 **Business Central** 中添加新数据库驱动程序。

流程

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

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

27.5. 编辑数据库驱动程序

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

流程

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

27.6. 删除数据库驱动程序

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

流程

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

第 28 章 数据集编写

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

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

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



注意

Data Sets 页面仅对具有 **admin** 角色的用户可见。

28.1. 添加数据集

您可以创建数据集，以从外部数据源获取数据，并将该数据用于报告组件。

流程

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

这时将打开 **Data Sets** 页面。

2. 点 **New Data Set** 并选择以下供应商类型之一：

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

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



注意

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

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



注意

配置步骤根据您选择的供应商的不同而有所不同。

4. 点击 **Save**。

28.2. 编辑数据集

您可以编辑现有的数据集，以确保获取到报告组件的数据为最新版本。

流程

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

这时将打开 **Data Set Explorer** 页面。
2. 在 **Data Set Explorer** 窗格中，搜索您要编辑的数据集合，选择数据集，然后单击 **Edit**。
3. 在 **Data Set Editor** 窗格中，根据需要使用适当的标签页编辑数据。该选项卡会根据您选择的

数据集供应商类型而有所不同。

例如，以下更改可用于编辑 **CSV** 数据供应商：

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

28.3. 数据刷新

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

28.4. 缓存数据

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

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

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

Business Central 数据集功能提供了两个缓存级别：

- 客户端级别
- 后端级别

您可以在数据集的高级 标签页上设置客户端缓存和后端缓存设置。

客户端缓存

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

后端缓存

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



注意

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

第 29 章 ARCHETYPE 管理

Business Central 提供了一个 **archetype** 管理功能，可让您列出、添加、验证、默认设置为 **archetypes**。您可以在 **Business Central** 的 **Archetypes** 页面中管理 **archetypes**。**archetypes** 在 **Apache Maven** 软件仓库中安装的项目，您可以在需要时设置或创建模板结构。

有关 **archetypes** 的最新和详细信息，请查看 [Archetypes 页面简介](#)。

29.1. 列出 ARCHETYPES

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

先决条件

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

流程

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

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

29.2. 添加 ARCHETYPE

您可以向 **Business Central** 添加新的 **archetype**。

先决条件

- 您已在 **Maven** 存储库中安装了 **archetype**。

流程

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

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

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


29.3. 管理 ARCHETYPE 的附加功能

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

先决条件

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

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Archetypes**。
2. 在 **Actions** 列中，点 **archetype** 右侧的  图标。
 - 从下拉菜单中选择 **Delete**，从列表中删除 **archetype**。
 - 从下拉菜单中选择 **Validate** 以验证 **archetype** 是否有效。



注意

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

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

29.4. 使用 ARCHETYPES 创建项目

您可以使用 **archetypes** 在 **Business Central** 中创建项目。当您在 **Business Central** 中创建项目时，它会被添加到连接到 **Red Hat Process Automation Manager** 安装的 **Git** 存储库中。

先决条件

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

流程

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

在命名空间中已经设置了默认的 **archetype**。

8. 单击 **Add**。

项目的 **资产** 视图基于所选 **archetype** 模板打开。

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

当您将 **archetypes** 添加到 **Business Central** 中时，您可以将它们用作所有空格中的模板。您可以在 **Settings** 选项卡中管理所有 **archetypes**，它位于空格中。此选项卡仅对具有 **admin** 角色的用户可见。

先决条件

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

流程

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

第 30 章 自定义项目首选项

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

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

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

流程

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

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

表 30.1. 文件导出属性

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

- **空格**：此首选项由以下属性组成：

表 30.2. 空格属性

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

默认值：此首选项由以下属性组成：

表 30.3. 默认值属性

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

高级 GAV 首选项：此首选项由以下属性组成：

表 30.4. 高级 GAV 首选项属性

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



注意

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

3.

点击 **Save**。

第 31 章 自定义工件存储库属性

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

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

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

流程

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

第 32 章 自定义语言设置

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

- **English**
- 西班牙语
- 法语
- 日语

默认语言为英文。

流程

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

第 33 章 自定义进程管理

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

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，然后选择 **Process Administration**。
2. 在 **Properties** 部分中，更新 每个页面属性的默认项，然后单击 **Save**。



注意

您可以指定要在每个页面上显示的 **10**、**20**、**50** 或 **100** 个项目。

第 34 章 自定义过程设计器

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

流程

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

第 35 章 SSH 密钥

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

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

35.1. 默认 SSH 密钥存储

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

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

每个 **SSH 公钥**均由以下文件组成，位于 **storage** 文件夹中：

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

例如：

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQADmak4Wu23RZ6XmN94bOsqecZxuTa4RRhhQm
HmTZjMB7HM57/90u/B/gB/GhsPEu1nAXL0npY56tT/MPQ8vRm2C2W9A7CzN5+z5yyL3W01Y
Zy3kzslk77CjULjfhrcfQSL3b2sPG5jv5E5/nyC/swSytucwT/PE7aXTS9H6cHIKUdYPzlt94SHoBx
WRIK7PJi9d+eLB+hmDzvbVa1ezu5a8yu2kcHi6Nxxfl5iRj2rsceDTp0imC1jMoC6ZDfBvZSxL9F>
TMwFdNnmTIJveBtv9nAbnAviWliiS0VOkdj1s3GxBxeZYAcKbcsK9sJzusptk5dxGsG2Z8vInaglN
6OaOQ7b7tcomzCYYwviGQ9gRX8sGsVrw39gsDIGYP2tA4bRr7ecHnlNg1b0HCchA5+QCDk
4Hbz1UrnHmPA2Lg9c3WGm2qedvQdVJXuS3mlwYOqL40aXPs6890PvFJUipiVSznF50djPnws
MxJZEf1HdTXgZD1Bh54ogZf7czyUNfkNkE69yJDbTHjpQd0cKUQnu9tVxqmBzhX31yF4VcsMe
ADcf2Z8wIA3n4LZnC/GwonYlq5+G93zJpFOkPhme8c2XuPuCXF795lsxyJ8SB/AlwPJAHEtm0y
0s0l1I4eWqxsDxkBOgN+ivU0cZrVMssHJEJb4o0FLf7iHhOW56/iMdD9w== userName
```

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

例如：

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

35.2. 自定义 SSH 密钥存储

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



注意

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

35.3. 创建 SSH 密钥

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

流程

1. 在系统上打开一个命令终端。
2. 运行 `ssh-keygen` 命令创建 SSH 密钥，如下例所示，其中 `< user_login >` 是您的用户名：

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



注意

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

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

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

4. 在命令提示符处，输入并确认密码短语：

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

5. 启动 **ssh-agent**：

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

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

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

35.4. 使用 SSH 密钥存储注册 SSH 公钥

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

流程

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

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

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

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



注意

Name 和 **Key** 字段是必须的。

7. 点 **Add SSH Key** 注册密钥。

35.5. 删除 SSH 密钥

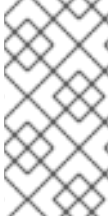
您可以从 **SSH Keys** 页面从 **Business Central** 中删除 **SSH** 密钥。

流程

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

第 36 章 在 **BUSINESS CENTRAL** 中管理自定义任务

自定义任务（工作项目）是可以运行自定义逻辑的任务。您可以在多个业务流程或 **Business Central** 的所有项目之间自定义和重新利用自定义任务。您还可以在设计器面板中添加自定义元素，包括名称、图标、子类别、输入和输出参数以及文档。**Red Hat Process Automation Manager** 在 **Business Central** 中的自定义任务存储库中提供一组自定义任务。您可以启用或禁用默认自定义任务，并将自定义任务上传到 **Business Central** 中以实施相关流程中的任务。



注意

Red Hat Process Automation Manager 包括一组有限的支持的自定义任务。不支持 **Red Hat Process Automation Manager** 中未包含的自定义任务。

流程

1.

在 **Business Central** 中，点右上角的



并选择 **Custom Tasks Administration**。

本页列出了自定义任务安装设置，以及用于整个 **Business Central** 项目中流程的自定义任务。在此页面中启用的自定义任务将在项目级别设置中找到，您可以在其中安装要在进程中使用的每个自定义任务。在项目安装自定义任务的方式是由您在此自定义任务管理页面上的 **Settings** 下启用或禁用的全局设置来确定。

2.

在 **Settings** 下，启用或禁用每个设置，以确定用户在项目级别安装自定义任务时如何实施可用的自定义任务。

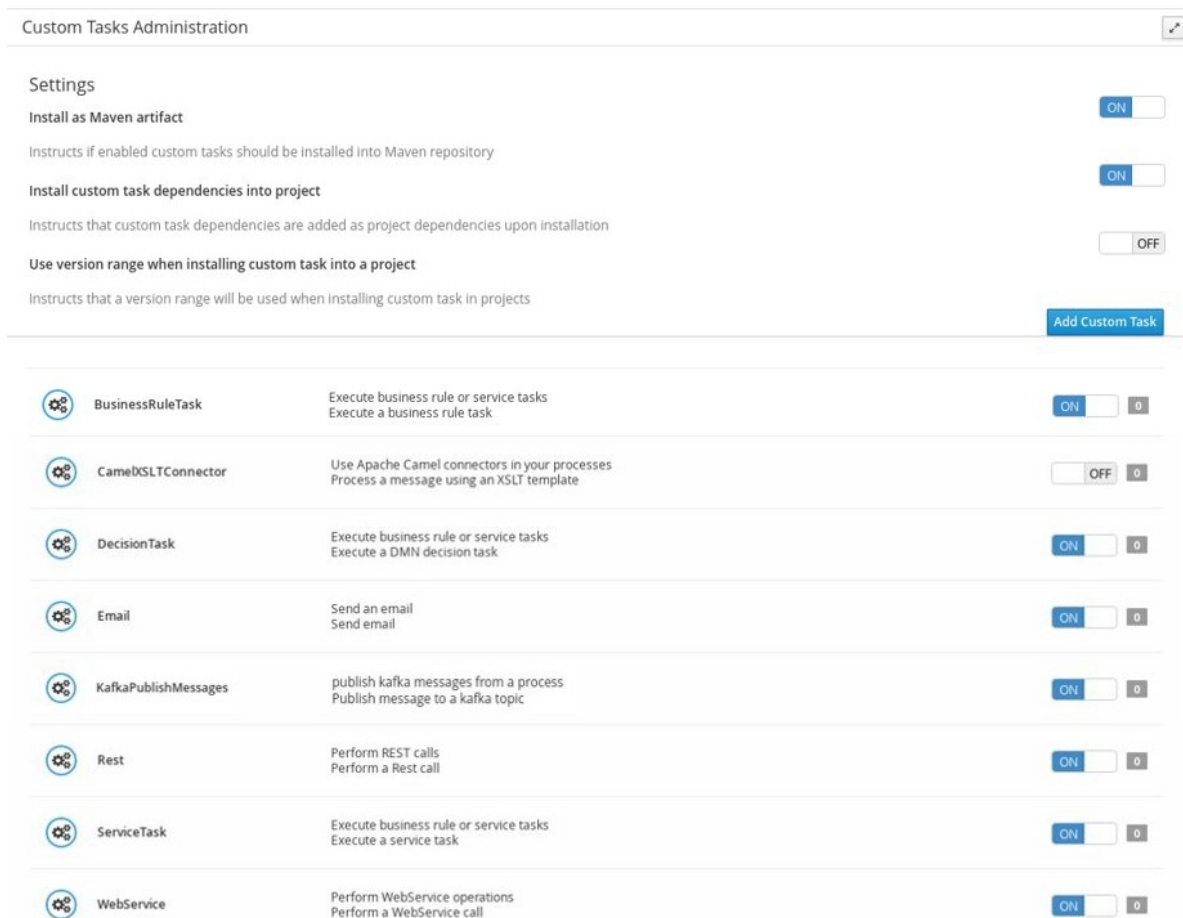
可用的自定义任务设置如下：

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

3.

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

图 36.1. 启用自定义任务和自定义任务设置



4.

要添加自定义任务，请点击 **Add Custom Task**，浏览到相关的 **JAR** 文件，然后单击 **Upload** 图标。如果类实施了一个 **WorkItemHandler**，您可以通过将该文件单独添加到 **Business Central** 中，用 **.wid** 文件替换注解。

5.

可选：要删除自定义任务，请点击您要删除的自定义任务行的 **remove** 并点 **Ok** 确认删除。

6.

在配置所有必需的自定义任务后，进入 **Business Central** 中的项目，进入 **Project Settings** → **Custom Tasks** 页面，以查看您启用的可用自定义任务。

7.

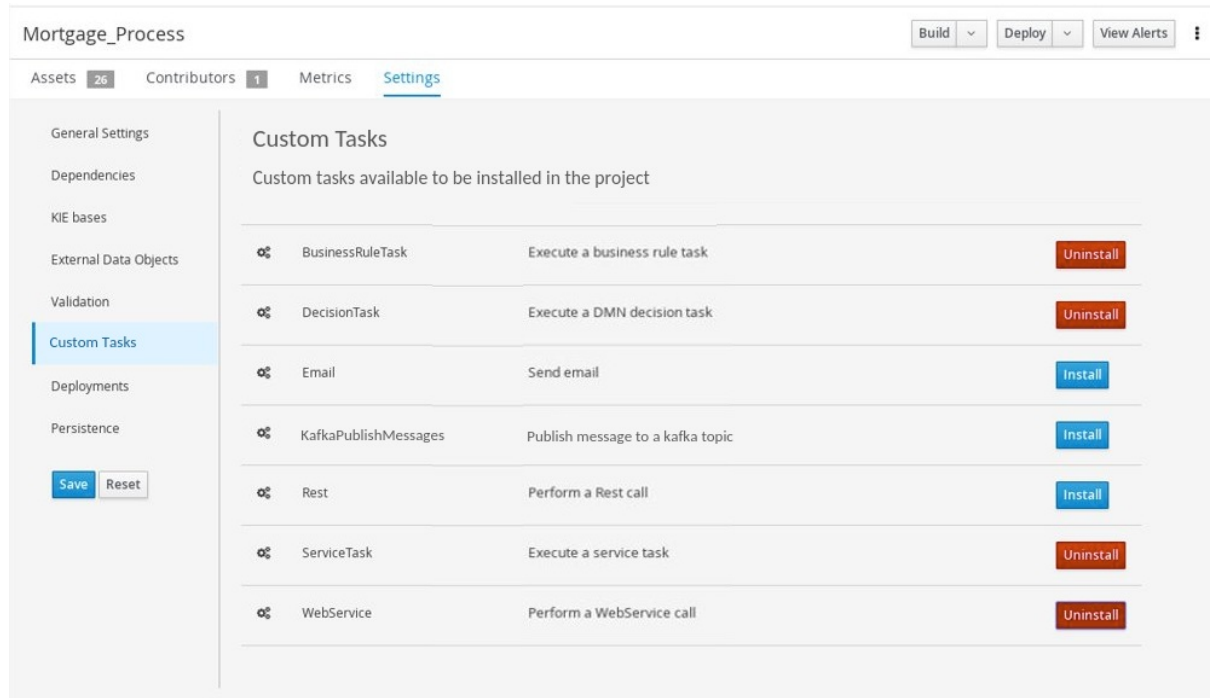
对于每个自定义任务，点 **Install** 使该项目中进程可用的任务，或者点击 **Uninstall** 以从项目中的进程中排除任务。

8.

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

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

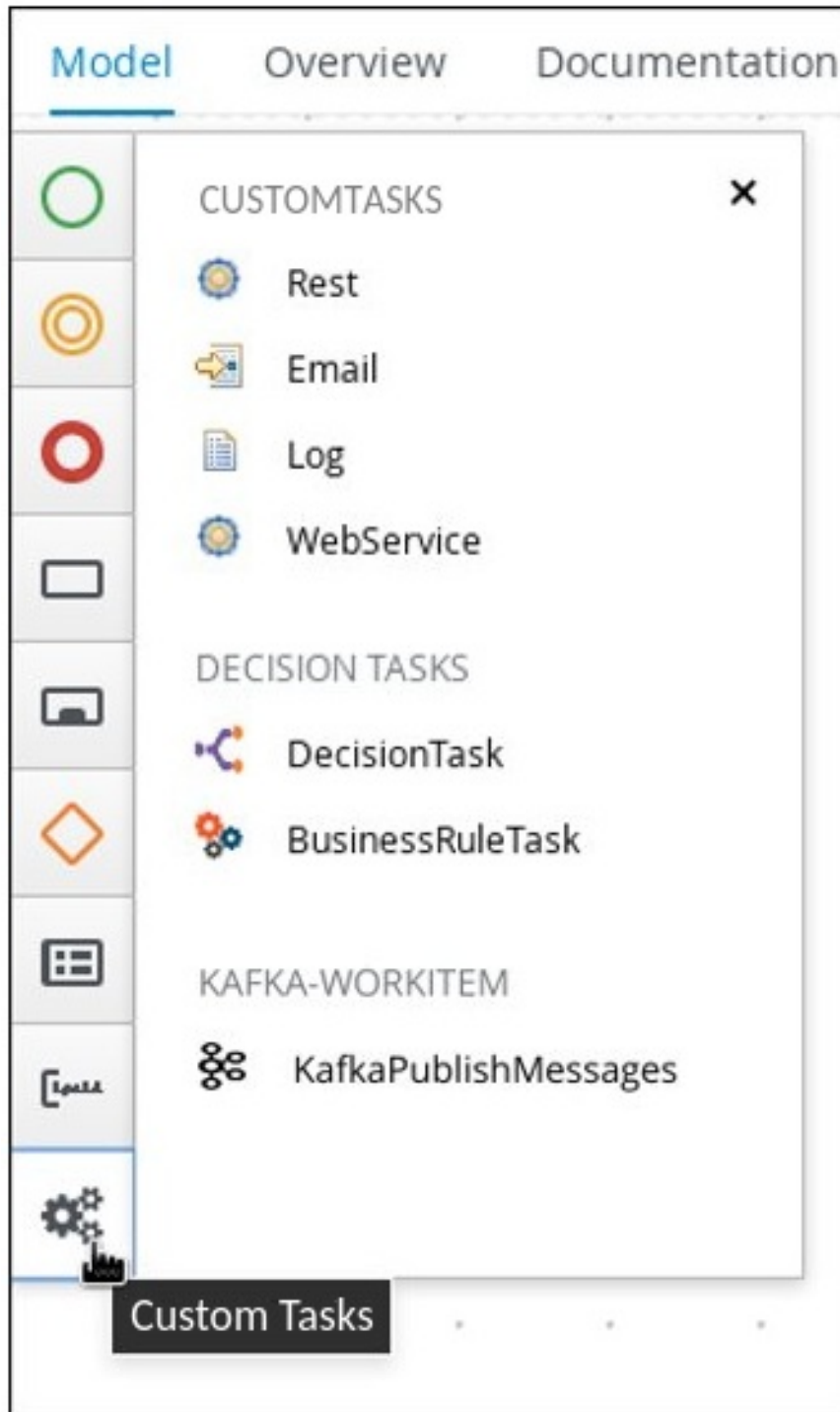
图 36.2. 安装要在进程中使用的自定义任务



9. 点击 **Save**。

10. 返回到项目页面，选择或在项目中添加业务流程，然后在流程设计器面板中选择 **Custom Tasks** 选项以查看您启用并安装的可用自定义任务：

图 36.3. 访问在进程设计器中安装的自定义任务



第 37 章 LDAP 连接

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

您可以配置以下 LDAP **UserGroupCallback** 实现属性：

表 37.1. LDAP **UserGroupCallback** 属性

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

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

37.1. LDAP USERGROUPCALLBACK 实现

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

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

例如：

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

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

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

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

例如：

```
-Djbpm.usergroup.callback.properties=FILE_LOCATION_ON_CLASSPATH
```

确保在启动用户任务服务器时注册 **LDAP** 回调。

例如：

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

其他资源

- [角色和用户](#)
- [红帽单点登录服务器管理指南](#)
- [定义 LDAP 登录域](#)
- [LDAP 登录模块](#)

- [LDAPExtended 登录模块](#)
- [AdvancedLDAP 登录模块](#)
- [AdvancedAdLDAP 登录模块](#)
- [LDAP 连接选项](#)
- [LDAPUsers 登录模块](#)

第 38 章 数据库连接

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

您可以配置以下数据库 **UserGroupCallback** 实现属性：

表 38.1. 数据库用户 **GroupCallback** 属性

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

38.1. 数据库用户 **GROUPCALLBACK** 实现

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

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

例如：

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

```
?");
```

```
callback = new DBUserGroupCallbackImpl(props);
```

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

例如：

```
-Djbpm.usergroup.callback.properties=FILE_LOCATION_ON_CLASSPATH
```

确保在启动用户任务服务器时注册数据库回调。

例如：

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

其他资源

- [角色和用户](#)

第 39 章 使用 **SETTINGS.XML** 文件配置 **MAVEN**

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

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

其他资源

- [为 **Business Central** 和 **KIE** 服务器配置外部 **Maven** 存储库](#)
- [在 **Maven** 中打包和部署 **Red Hat Process Automation Manager** 项目](#)
- [Red Hat Process Automation Manager 的 **Maven** 设置和软件仓库](#)
- [系统与 **Maven** 集成](#)
- [欢迎使用 **Apache Maven**](#)
- [Apache Maven 项目 - 仓库简介](#)
- [Apache Maven Parent POMs 参考.](#)

第 40 章 GAV 检查管理

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

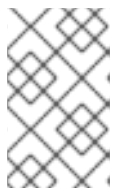
默认情况下，**B Business Central** 会检测重复的 **GAV**。具有 **admin** 角色的用户可以禁用此功能。

40.1. 配置 GAV 检查和子 GAV 版本

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

流程

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



注意

您可以单击 **重置** 来撤消所有更改。

5. 单击 **Save** 以确认更改。



注意

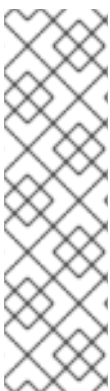
在 **Development Mode** 中的项目禁用了重复的 **GAV** 检测。要在 **Business Central** 中启用重复的 **GAV** 检测，请转至 **Project Settings** → **General Settings** → **Version**，并将 **Development Mode** 选项切换为 **OFF**（如果适用）。

40.2. 为所有项目配置 GAV 检查

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

流程

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



注意

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

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

第 41 章 在 KIE 服务器和 BUSINESS CENTRAL 中配置环境模式

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

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

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

流程

1. 要配置 **KIE 服务器** 环境模式，请将 **org.kie.server.mode** 系统属性设置为 **org.kie.server.mode=development** 或 **org.kie.server.mode=production**。
2. 要在 **Business Central** 中配置项目部署行为，请转至 **Project Settings** → **General Settings** → **Version**，再切换 **Development Mode** 选项。



注意

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

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

第 42 章 GIT HOOK 和远程 GIT 存储库集成

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

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



注意

Business Central 只支持 **post-commit Git hook**。

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

42.1. 创建 POST-COMMIT GIT HOOK

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

流程

1.

创建 **post-commit Git hook** 文件：

```
$ touch post-commit
```

2.

将 **post-commit** 文件的权限设置为 **755**：

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

3.

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

-

将所有更改推送到远程存储库：

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

- 记录信息：

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

- 执行另一个文件的代码：

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



注意

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

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

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

42.2. 导入远程 GIT 存储库


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

先决条件

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

- 读取外部 **Git** 存储库的访问凭据。
- (对于 **Windows**) **Cygwin** 安装在安装过程中添加的 **Git** 软件包，并将到 **Cygwin /bin** 文件夹的路径添加到环境变量 **PATH** 变量中。例如，**C:\cygwin64\bin**。有关 **Cygwin** 安装的更多信息，请参阅 [安装和更新 Cygwin 软件包](#)。

流程

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

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

重要

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

您必须在 **Git** 供应商中配置您的公共 **ssh** 密钥。

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

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

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

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

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

有关创建 **post-commit Git hook** 的更多信息，请参阅 [第 42.1 节“创建 post-commit Git hook”](#)。

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

- a. 在 **Business Central** 中，转至 **Menu** → **Projects** → **Add Asset** → **Guided Rule**。
- b. 在 **Create new Guided Rule** 页面中，输入所需的信息。
- c. 点 **确定**。

Business Central 自动将所有更改推送到远程存储库。

其他资源

- [自定义 Git - Git Hook](#)

42.3. 为现有远程 GIT 项目存储库配置 GIT HOOK

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

先决条件

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

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

流程

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

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

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

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

有关创建 **post-commit Git hook** 的更多信息，请参阅 [第 42.1 节“创建 post-commit Git hook”](#)。

3. 可选：要检查配置是否成功，请在 **Business Central** 中创建指导规则：
 - a. 在 **Business Central** 中，转至 **Menu** → **Projects** → **Add Asset** → **Guided Rule**。
 - b. 在 **Create new Guided Rule** 页面中，输入所需的信息。
 - c. 点 **确定**。

Business Central 自动将所有更改推送到远程存储库。

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

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



注意

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

先决条件

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

流程

1. 在本地系统的目录中创建 **post-commit Git hook**。

有关创建 **post-commit Git hook** 的更多信息，请参阅 [第 42.1 节“创建 post-commit Git hook”](#)。

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

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

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

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

```

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

```

3.

启动 **Business Central**。

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

其他资源

- [自定义 Git - Git Hook](#)

42.5. 集成远程 GIT 存储库

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

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

先决条件

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

流程

1. 在终端窗口中，将 **GitHub** 存储库克隆到您的系统：

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

2. 进入克隆的存储库：

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

3. 执行 **Maven** 清理安装：

```
$ mvn clean install
```

4. 在 **EAP_HOME** 目录中创建一个 **/hooks** 文件夹：

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

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

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

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

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

模板 **.gitremote** 配置文件示例

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

7.

修改 `.gitremote` 配置文件参数。

表 42.1. `.gitremote` 参数示例

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

8.

在 `EAP_HOME/hooks` 中创建 `post-commit Git hook` 文件：

```
$ touch post-commit
```

9. 将 **post-commit** 文件的权限设置为 **755** :

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

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

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

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

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

注意

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

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

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

42.6. GIT HOOK 退出代码

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

42.7. 自定义 GIT HOOK 通知

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

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

流程

1.

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

```
<exit_code>=<display_message>
```

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

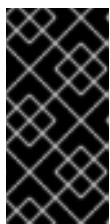
例如：

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



注意

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



重要

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

2.

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

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

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

42.7.1. Business Central 中的 Git hook 通知

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

表 42.2. Git hook UI 通知类型

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



重要

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

Windows 机器没有此限制并支持广泛的退出代码。

42.7.2. Git hook 通知国际化支持

您可以将额外属性文件放在与由 **appformer.git.hooks.bundle** 系统属性指定的原始属性文件相同的路径中，可以国际化通知消息。

不同本地化文件的名称必须是 **<filename>_<lang>.properties**，其中 **<filename>** 与原始文件相同。例如，如果系统属性指向 **Messages.properties**，您可以为英语创建 **Messages_en.properties**、适用于法语的 **Messages_fr.properties**，或者用于意大利语的 **Messages_it.properties**。

通知服务将根据用户的语言选择属性文件，如果没有该语言的可用转换，它将使用原始 **Messages.properties** 文件中的条目。

第 43 章 针对 **BUSINESS CENTRAL** 中分支的角色访问控制

Business Central 为用户授予限制特定协作者类型目标分支的访问选项。安全检查使用 **Security Management** 屏幕和 **contributors** 源来授予或拒绝空格和项目的权限。例如，如果用户具有更新项目的安全性权限，并根据贡献类型具有该分支的写入权限，则可以创建新的资产。

43.1. 自定义基于角色的访问控制

您可以为 **Business Central** 中的项目的每个分支自定义 **contributors** 角色权限。例如，您可以为分配给分支的每个角色设置 **Read**、**Write**、**Delete** 和 **Deploy** 访问权限。

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Projects**。
2. 如果需要，添加一个新的投稿者：
 - a. 单击项目名称，然后单击 **Contributors** 选项卡。
 - b. 点 **Add Contributor**。
 - c. 在文本字段中输入用户名。
 - d. 从下拉列表中选择 **Contributor** 角色类型。
 - e. 点 **确定**。
3. 为相关贡献者自定义基于角色的访问控制：
 - a. 点 **Settings** → **Branch Management**。
 - b. 从下拉列表中选择分支名称。

- c. 在 **Role Access** 部分中，选择或取消选择权限复选框，为每个可用角色类型指定基于角色的分支访问权限。

- d. 单击 **Save**，然后再次单击 **Save** 以确认更改。

第 44 章 查看进程实例日志

您可以从 **Logs** 选项卡查看实例的所有进程事件。实例日志会列出所有当前和上一个进程状态。**Business Central** 有两个用于流程实例的日志，**业务**和**技术** 日志。

流程

1. 在 **Business Central** 中，前往 **Menu** → **Manage** → **Process Instances**。
2. 在 **Manage Process Instances** 页面上，单击您要查看的日志的进程实例。
3. 选择 **Logs** 选项卡：
 - 单击 **Business** 查看业务和事件日志。
 - 单击 **Technical** 查看技术事件日志。
 - 点 **As c** 或 **Desc** 更改日志文件的顺序。

第 45 章 BUSINESS CENTRAL 系统属性

本节中列出的 **Business Central** 系统属性传递到 **standalone*.xml** 文件。

Git 目录

使用以下属性设置 **Business Central Git** 目录的位置和名称：

- **org.uberfire.nio.git.dir:** **Business Central Git** 目录的位置。
- **org.uberfire.nio.git.dirname:** **Business Central Git** 目录的名称。默认值：**.niogit**。
- **org.uberfire.nio.git.ketch:** **Enables** 或 **disable Git ketch**。
- **org.uberfire.nio.git.hooks:** **Git hook** 目录的位置。

git over HTTP

使用以下属性配置通过 **HTTP** 对 **Git** 存储库的访问：

- **org.uberfire.nio.git.proxy.ssh.over.http:** 指定 **SSH** 是否应使用 **HTTP** 代理。默认值：**false**。
- **http.proxyHost** : 定义 **HTTP** 代理的主机名。默认值：**null**。
- **http.proxyPort** : 定义 **HTTP** 代理的主机端口（整数值）。默认值：**null**。
- **http.proxyUser** : 定义 **HTTP** 代理的用户名。
- **HTTP.proxyPassword** : 定义 **HTTP** 代理的用户密码。
- **org.uberfire.nio.git.http.enabled:** **Enables** 或 **disable the HTTP** 守护进程。默认值：**true**。

- **org.uberfire.nio.git.http.host:** 如果启用了 **HTTP** 守护进程，它将使用此属性作为主机标识符。这是一个信息性属性，用于显示如何通过 **HTTP** 访问 **Git** 存储库。**HTTP** 仍然依赖于 **servlet** 容器。默认值：**localhost**。
- **org.uberfire.nio.git.http.hostname:** 如果 **HTTP** 守护进程被启用，它将使用此属性作为主机名标识符。这是一个信息性属性，用于显示如何通过 **HTTP** 访问 **Git** 存储库。**HTTP** 仍然依赖于 **servlet** 容器。默认值：**localhost**。
- **org.uberfire.nio.git.http.port** : 如果启用了 **HTTP** 守护进程，它将使用此属性作为端口号。这是一个信息性属性，用于显示如何通过 **HTTP** 访问 **Git** 存储库。**HTTP** 仍然依赖于 **servlet** 容器。默认值：**8080**。

Git over HTTPS

使用以下属性通过 **HTTPS** 配置对 **Git** 存储库的访问：

- **org.uberfire.nio.git.proxy.ssh.over.https:** 指定 **SSH** 是否使用了 **HTTPS** 代理。默认值：**false**。
- **HTTPS.proxyHost** : 定义 **HTTPS** 代理的主机名。默认值：**null**。
- **HTTPS.proxyPort** : 定义 **HTTPS** 代理的主机端口（整数值）。默认值：**null**。
- **HTTPS.proxyUser** : 定义 **HTTPS** 代理的用户名。
- **HTTPS.proxyPassword** : 定义 **HTTPS** 代理的用户密码。
- **user.dir** : 用户目录的路径。
- **org.uberfire.nio.git.https.enabled:** 启用或禁用 **HTTPS** 守护进程。默认值：**false**
- **org.uberfire.nio.git.https.host:** 如果 **HTTPS** 守护进程已启用，它将使用此属性作为主

机标识符。这是一个信息性属性，用于显示如何通过 **HTTPS** 访问 **Git** 存储库。**HTTPS** 仍然依赖于 **servlet** 容器。默认值：**localhost**。

- **org.uberfire.nio.git.https.hostname**: 如果 **HTTPS** 守护进程已启用，它将使用此属性作为主机名标识符。这是一个信息性属性，用于显示如何通过 **HTTPS** 访问 **Git** 存储库。**HTTPS** 仍然依赖于 **servlet** 容器。默认值：**localhost**。
- **org.uberfire.nio.git.https.port**: 如果 **HTTPS** 守护进程已启用，它将使用此属性作为端口号。这是一个信息性属性，用于显示如何通过 **HTTPS** 访问 **Git** 存储库。**HTTPS** 仍然依赖于 **servlet** 容器。默认值：**8080**。

JGit

- **org.uberfire.nio.jgit.cache.instances** : 定义 **JGit** 缓存大小。
- **org.uberfire.nio.jgit.cache.overflow.cleanup.size**: 定义 **JGit** 缓存溢出清理大小。
- **org.uberfire.nio.jgit.remove.eldest.iterations**: *Enables* 或 *disable whether remove eldest JGit* 迭代。
- **org.uberfire.nio.jgit.cache.evict.threshold.duration**: 定义 **JGit** 驱除阈值持续时间。
- **org.uberfire.nio.jgit.cache.evict.threshold.time.unit** : 定义 **JGit** 驱除阈值单元。

Git 守护进程

使用以下属性来启用和配置 **Git** 守护进程：

- **org.uberfire.nio.git.daemon.enabled**: *Enables* 或 *disable the Git* 守护进程。默认值：**true**。
- **org.uberfire.nio.git.daemon.host**: 如果 **Git** 守护进程已启用，它将使用此属性作为本地主机标识符。默认值：**localhost**。
- **org.uberfire.nio.git.daemon.hostname**: 如果 **Git** 守护进程已启用，它将使用此属性作

为本地主机名标识符。默认值：`localhost`

- **`org.uberfire.nio.git.daemon.port`:** 如果 **Git** 守护进程已启用，它将使用此属性作为端口号。默认值：`9418`。
- **`org.uberfire.nio.git.http.sslVerify`:** 启用或禁用 **Git** 存储库的 **SSL** 证书检查。默认值：`true`。



注意

如果已经使用默认或者分配的端口，则会自动选择一个新端口。确保端口可用，并检查日志以了解更多信息。

Git SSH

使用以下属性来启用和配置 **Git SSH** 守护进程：

- **`org.uberfire.nio.git.ssh.enabled`:** 启用或禁用 **SSH** 守护进程。默认值：`true`。
- **`org.uberfire.nio.git.ssh.host`:** 如果启用了 **SSH** 守护进程，它将使用此属性作为本地主机标识符。默认值：`localhost`。
- **`org.uberfire.nio.git.ssh.hostname`:** 如果启用了 **SSH** 守护进程，它将使用此属性作为本地主机名标识符。默认值：`localhost`。
- **`org.uberfire.nio.git.ssh.port`:** 如果启用了 **SSH** 守护进程，它将使用此属性作为端口号。默认值：`8001`。



注意

如果已经使用默认或者分配的端口，则会自动选择一个新端口。确保端口可用，并检查日志以了解更多信息。

- **`org.uberfire.nio.git.ssh.cert.dir`:** 存储本地证书的 `.security` 目录的位置。默认值：工作目录。

- **org.uberfire.nio.git.ssh.idle.timeout**:设置 SSH 闲置超时。
- **org.uberfire.nio.git.ssh.passphrase** : 在使用 SCP 风格 URL 克隆 git 存储库时用于访问操作系统的公钥存储的密码短语。示例：`git@github.com:user/repository.git`。
- **org.uberfire.nio.git.ssh.algorithm**: Algorithm 供 SSH 使用。默认值：`RSA`。
- **org.uberfire.nio.git.gc.limit**: Sets the GC 限值。
- **org.uberfire.nio.git.ssh.ciphers** : 以逗号分隔的密码字符串。可用的密码是 `aes128-ctr,aes192-ctr,aes256-ctr,arcfour128,arcfour256,aes192-cbc,aes256-cbc`。如果没有使用属性，则会载入所有可用的密码。
- **org.uberfire.nio.git.ssh.macs**:以逗号分隔的消息验证代码(MAC)字符串。可用的 MACs 是 `hmac-md5,hmac-md5-96,hmac-sha1,hmac-sha1-96,hmac-sha2-256,hmac-sha2-512`。如果没有使用属性，则会加载所有可用的 MAC。



注意

如果您计划使用 `RSA` 或 `DSA` 以外的任何算法，请确保将应用服务器设置为使用 `Bouncy Castle JCE` 库。

KIE 服务器节点和流程自动化管理器控制器

使用以下属性，从流程自动化管理器控制器配置与 KIE 服务器节点的连接：

- **org.kie.server.controller**: URL 用于连接到 `Process Automation Manager` 控制器。例如，`ws://localhost:8080/business-central/websocket/controller`。
- **org.kie.server.user**:用于从流程自动化管理器控制器连接到 KIE 服务器节点的用户名。只有在将这个 `Business Central` 安装用作流程自动化管理器控制器时才需要此属性。
- **org.kie.server.pwd** : 用于从流程自动化管理器控制器连接到 KIE 服务器节点的

Password。只有在将这个 **Business Central** 安装用作流程自动化管理器控制器时才需要此属性。

Maven 和 miscellaneous

使用以下属性配置 **Maven** 和其他其它功能：

- **kie.maven.offline.force** : 强制 **Maven** 的行为如离线一样。如果为 **true**，禁用在线依赖关系解析。默认值：**false**。



注意

仅在 **Business Central** 中使用此属性。如果您与任何其他组件共享一个运行时环境，请隔离配置并将其应用到 **Business Central**。

- **org.uberfire.gzip.enable**: **Enables** 或在 **GzipFilter** 压缩过滤器中禁用 **Gzip** 压缩。默认值：**true**。
- **org.kie.workbench.profile**: 选择 **Business Central** 配置集。可能的值有 **FULL** 或 **PLANNER_AND_RULES**。一个前缀 **FULL_** 设定配置集，并从管理员首选项中隐藏配置集首选项。默认值：**FULL**
- **org.appformer.m2repo.url**: **Business Central** 在查找依赖项时使用 **Maven** 存储库的默认位置。它定向到 **Business Central** 中的 **Maven** 存储库，例如 **http://localhost:8080/business-central/maven2**。在启动 **Business Central** 前设置此属性。默认值：到内 **m2** 存储库的文件路径。
- **appformer.ssh.keystore** : 通过指定类名称定义要与 **Business Central** 搭配使用的自定义 **SSH** 密钥存储。如果属性不可用，则使用默认的 **SSH** 密钥存储。
- **appformer.ssh.keys.storage.folder** : 在使用默认 **SSH** 密钥存储时，此属性为用户的 **SSH** 公钥定义存储文件夹。如果属性不可用，则密钥将存储在 **Business Central .security** 文件夹中。
- **appformer.experimental.features**: 启用实验功能框架。默认值：**false**。

- **org.kie.demo** : 启用来自 **GitHub** 的演示应用程序的外部克隆。
- **org.uberfire.metadata.index.dir** : 存储 **Lucene .index** 目录的位置。默认值 : 工作目录。
- **org.uberfire.ldap.regex.role_mapper**: **Regex** 模式用于将 **LDAP** 主体名称映射到应用程序角色名称。请注意, 在匹配原则值和角色名称时, 变量角色必须是模式的一部分, 因为应用程序角色名称替换变量角色。
- **org.uberfire.sys.repo.monitor.disabled** : 禁用配置监控器。除非确定, 否则不要禁用。默认值 : **false**。
- **org.uberfire.secure.key** : 密码加密使用的密码。默认值 : **org.uberfire.admin**。
- **org.uberfire.secure.alg**: **Crypto** 算法由密码加密使用。默认值 : **PBEWithMD5AndDES**。
- **org.uberfire.domain**: **uberfire** 使用的 **Security-domain name**。默认值 : **ApplicationRealm**。
- **org.guvnor.m2repo.dir**: **Place** 存储 **Maven** 存储库文件夹。默认值 : **< ;working-directory>/repositories/kie**。
- **org.guvnor.project.gav.check.disabled**: **Disables group ID、工件 ID 和版本(GAV)检查**。默认值 : **false**。
- **org.kie.build.disable-project-explorer** : 禁用 **Project Explorer** 中所选项目的自动构建。默认值 : **false**。
- **org.kie.builder.cache.size** : 定义项目构建器的缓存大小。默认值 : **20**。
- **org.kie.library.assets_per_page**:您可以在项目屏幕中自定义每个页面的资产数量。默认值 : **15**。

- **org.kie.verification.disable-dtable-realtime-verification:** 禁用决策表的实时验证和验证。默认值：**false**。

进程自动化管理器控制器

使用以下属性配置如何连接到流程自动化管理器控制器：

- **org.kie.workbench.controller:** 用于连接 **Process Automation Manager** 控制器的 URL，例如 **ws://localhost:8080/kie-server-controller/websocket/controller**。
- **org.kie.workbench.controller.user:** **Process Automation Manager** 控制器用户。默认值：**kieserver**。
- **org.kie.workbench.controller.pwd:** **Process Automation Manager** 控制器密码。默认值：**kieserver1!**。
- **org.kie.workbench.controller.token:** 用于连接到 **Process Automation Manager** 控制器的令牌字符串。

Java Cryptography Extension KeyStore(JCEKS)

使用以下属性配置 JCEKS：

- **kie.keystore.keyStoreURL**：用于加载 **Java Cryptography Extension KeyStore(JCEKS)** 的 URL。例如，**file:///home/kie/keystores/keystore.jceks**。
- **kie.keystore.keyStorePwd**：用于 **JCEKS** 的密码。
- **kie.keystore.key.ctrl.alias:** 默认 **REST Process Automation Manager** 控制器的密钥别名。
- **kie.keystore.key.ctrl.pwd:** 默认 **REST Process Automation Manager** 控制器的别名。

渲染

使用以下属性在 **Business Central** 和 **KIE** 服务器呈现的形式间切换：

- **org.jbpm.wb.forms.renderer.ext** : 切换 **Business Central** 和 **KIE** 服务器之间的表单渲染。默认情况下，表单渲染由 **Business Central** 执行。默认值：**false**。
- **org.jbpm.wb.forms.renderer.name**: 让您在 **Business Central** 和 **KIE** 服务器呈现的表单之间进行切换。默认值：**workbench**。

第 46 章 与 **BUSINESS CENTRAL** 相关的性能调优注意事项

以下关键概念或建议做法可以帮助您优化 **Business Central** 配置和 **Red Hat Process Automation Manager** 性能。这些概念在本章节中进行了概述，在适用的情况下，会详细介绍相关文档。本节将在 **Red Hat Process Automation Manager** 的新版本时扩展或更改。

确保在开发过程中启用了开发模式

您可以将 **Business Central** 中的 **KIE** 服务器或特定项目设置为使用 **生产模式** 或 **开发模式**。默认情况下，**Business Central** 中的 **KIE** 服务器和所有新项目均为开发模式。这个模式提供有助于开发体验的功能，如灵活的项目部署策略以及优化开发期间 **KIE** 服务器性能的功能，如禁用的 **GAV** 检测。使用开发模式，直到 **Red Hat Process Automation Manager** 环境创建并完全可用于生产环境模式。

有关配置环境模式或重复的 **GAV** 检测的详情，请查看以下资源：

- [第 41 章 在 **KIE** 服务器和 **Business Central** 中配置环境模式](#)
- [打包和部署 **Red Hat Process Automation Manager** 项目](#)

禁用复杂指导决策表的验证和验证

Business Central 的决策表验证和验证功能默认启用。这个功能可帮助您验证您的指导的决策表，但使用复杂指导的决策表，此功能可以隐藏决策引擎性能。您可以通过将 **org.kie.verification.disable-dtable-realtime-verification** 系统属性值设为 **true** 来禁用此功能。

如需有关指导决策表验证的更多信息，请参阅 [使用指导决策表设计决策服务](#)。

如果您有很多大型项目，禁用自动构建

在 **Business Central** 中，当您在 **Project Explorer** 侧面板中的项目间导航时，所选项目会自动构建，以便 **Alerts** 窗口被更新来显示项目的构建错误。如果您在进行活跃开发中的很多项目之间有大型项目或频繁切换，则此功能可以隐藏业务中心和决策引擎性能。

要禁用自动项目构建，将 **org.kie.build.disable-project-explorer** 系统属性设置为 **true**。

部分 III. 在 **BUSINESS CENTRAL** 中使用独立视角

作为业务规则开发人员，您可以嵌入来自 **Web** 应用程序业务中心的独立视角，然后使用它们编辑规则、流程、决策表和其他资产。

先决条件

- **Business Central** 已部署并在 **Web**/应用程序服务器中运行。
- 您已登录到 **Business Central**。

第 47 章 BUSINESS CENTRAL 中的独立视角

业务中心根据资产的格式为编写资产提供专用编辑器。**Business Central** 具有可让您单独使用这些编辑器的功能。这个功能被称为编辑器的独立视角模式，或者只是独立视角。

作为业务规则开发人员，您可以在 **Web** 应用程序中嵌入独立视角，然后使用它编辑规则、流程、决策表和其他资产。嵌入了视角后，您可以在不切换到 **Business Central** 的情况下编辑自己的应用程序中的资产。您可以使用此功能自定义 **Web** 应用程序。除了独立视角外，您还可以在应用程序中嵌入独立自定义页面（仪表板）。

您可以通过在带有独立和视角参数的浏览器中使用特定的 **Web** 地址来访问独立视角。独立视角的 **Web** 地址也可以包含其他参数。

第 48 章 使用独立库视角

您可以使用 **Business Central** 的库视角来选择您要编辑的项目。您还可以对所选项目执行所有创作功能。

独立库透视图可以通过两种方式使用，使用 `header=UberfireBreadcrumbsContainer` 参数。区别在于，在库视角的顶部会显示带有 `标头` 参数的地址的面包。使用这个链接，您可以为项目创建额外的空间。

流程

1. 登录到 **Business Central**。
2. 在网页浏览器中输入适当的 **Web** 地址：
 - a. 在不使用 `标头` 参数的情况下访问独立库视角

`http://localhost:8080/business-central/kie-wb.jsp?standalone=true&perspective=LibraryPerspective`

在浏览器中打开 **standalone** 库视角，没有面包对面包的面包。

- b. 使用 `标头` 参数访问独立库视角

`http://localhost:8080/business-central/kie-wb.jsp?standalone=true&perspective=LibraryPerspective&header=UberfireBreadcrumbsContainer`

在浏览器中打开带有面包的面包（面包对面包）的单机库视角。

第 49 章 使用独立编辑器视角

您可以使用 **Business Central** 的独立编辑器视角访问资产的特定编辑器。使用这个视角，您可以打开资产的编辑器，并可以根据需要修改资产。

访问资产的独立编辑器视角的 **Web** 地址包含 **独立** 和 **路径** 参数。**path** 参数必须包含到资产的完整路径，**Web** 地址可以以 **#StandaloneEditorPerspective** 字符串结尾。另外，根据不同的 **path** 参数，您可以在独立模式下访问特定资产的编辑器。

流程

1. 登录到 **Business Central**。
2. 在 **Web** 浏览器中，根据需要输入适当的 **Web** 地址，例如：

- a. 要编辑进程：

```
http://localhost:8080/business-central/kie-wb.jsp?
standalone&path=default://master@MySpace/Shop/src/main/resources/com/purchase
.bpmn#StandaloneEditorPerspective
```

Process Designer 在独立模式中打开。

- b. 要编辑表单：

```
http://localhost:8080/business-central/kie-wb.jsp?
standalone&path=default://master@MySpace/Mortgage_Process/src/main/resources/
ApplicationMortgage.frm#StandaloneEditorPerspective
```

Form Modeler 在独立模式中打开。

第 50 章 使用独立内容管理器视角

通过使用应用程序中的独立内容管理器视角，您可以创建并编辑应用程序的内容及其导航菜单。

流程

1. 登录到 **Business Central**。
2. 在 **Web** 浏览器中，在地址栏中输入以下 **Web** 地址：

**`http://localhost:8080/business-central/kie-wb.jsp?
standalone=true&perspective=ContentManagerPerspective`**

独立内容管理器透视图在浏览器中打开。

第 51 章 使用独立自定义页面（仪表板）

除了独立视角外，您还可以在应用程序中嵌入自定义页面（也称为仪表板）。要从应用程序访问自定义页面，请提供自定义页面的名称作为 **perspective** 参数的值。请注意，**perspective** 参数区分大小写。

流程

1. 登录到 **Business Central**。
2. 在 **Web** 浏览器中，在地址栏中输入自定义页面的 **Web** 地址，例如：

**`http://localhost:8080/business-central/kie-wb.jsp?
standalone=true&perspective=CustomPageName`**

独立自定义页面会在浏览器中打开。将值 **CustomPageName** 替换为您要在独立模式中使用的自定义页面的名称。

部分 IV. 在 **BUSINESS CENTRAL** 中创建自定义页面

作为业务分析员或业务规则开发人员，您可以使用 **Business Central** 中的页面和动态仪表板来显示有关项目的特定信息。仪表板是包含至少一个动态报告组件的页面集合。您可以定义数据集，以向仪表板的报告组件提供。您可以将仪表板导出到 **Red Hat JBoss EAP** 上的独立 **Dashbuilder Runtime dashboard viewer** 中，或 **Red Hat OpenShift Container Platform** 上的 **Dashbuilder Standalone dashboard viewer**。

先决条件

- 以有权编辑页面的用户身份登录到 **Business Central**。

第 52 章 BUSINESS CENTRAL 自定义仪表板

仪表板是 **Business Central** 页面的集合，其中至少包含一个报告组件。仪表板通常包含数据集、导航树和权限。

创建自定义仪表板有四个阶段：

- **数据集编写**：定义用于访问数据的数据集，并通过页面显示数据。如需更多信息，[请参阅添加数据集](#)。
- **页面编写**：创建仪表板页面。如需更多信息，[请参阅创建页面](#)。
- **发布** - 在此阶段，当您创建自定义导航树或修改现有默认导航树（工作台树）时，页面之间的导航会被定义。如需更多信息，[请参阅创建导航树](#) 或 [编辑导航树](#)。
- **安全管理** - 在这个阶段，设置角色和组权限，用于定义用户在工作 **Business Central** 时授予用户的特权。如需更多信息，[请参阅安全管理](#)。

其他资源

要从以前版本的 **Business Central** 迁移仪表板，请使用 [第 57.1 节“导出 Business Central 仪表板数据”](#) 中描述的 **Dashbuilder Data Transfer** 功能。

第 53 章 DASHBUILDER RUNTIME 和 DASHBUILDER STANDALONE

Dashbuilder Runtime 和 **Dashbuilder standalone** 是可用于查看从 **Business Central** 中创建的仪表板和从 **Business Central** 中导出的仪表板。这对于查看没有 **Business Central** 的环境中的业务指标很有用。**Dashbuilder Runtime** 可用于在红帽 **JBoss EAP** 上安装。您可以在 **Red Hat OpenShift Container Platform** 上部署 **Dashbuilder Standalone**。

在 **Dashbuilder Runtime** 和 **Dashbuilder Standalone** 中的仪表板页面间导航，与在创建了仪表板的 **Business Central** 实例中进行导航。如果页面属于组，则该组将导入到 **Dashbuilder Runtime** 或 **Dashbuilder Standalone**，以及页面。如果将页面导入为 **Dashbuilder Runtime** 或 **Dashbuilder Standalone Standalone**，但没有用于导航，则页面将添加到 **Runtime Dashboards** 菜单组中。如果没有导出导航，则所有页面都会添加到 **Runtime Dashboards** 菜单组中。

53.1. 在 RED HAT JBOSS EAP 上安装 DASHBUILDER 运行时

要安装 **Dashbuilder Runtime**，请下载 **Dashbuilder Runtime WAR**，再创建一个具有 **admin** 角色的用户。

先决条件

- 您有红帽 **JBoss EAP** 安装。
- 您已在 **Business Central** 中创建并导出仪表板。有关导出 **Dashbuilder** 数据的更多信息，请参阅 [配置 Business Central 设置和属性](#) 指南中的“导出和导入 **Dashbuilder** 数据”部分。

流程

1. 进入红帽客户门户网站中的 [Software Downloads](#) 页面（需要登录），然后从下拉列表中选择产品和版本：
 - 产品：流程自动化管理器
 - 版本：7.13.4
2. 下载 **Red Hat Process Automation Manager 7.13.4 Add Ons (rhpam-7.13.4-add-ons.zip)** 并提取 **ZIP** 文件。

3. 导航到包含提取文件的目录，并提取 `rhpmam-7.13.4-dashbuilder-runtime.zip` 文件。
4. 将您提取的 `dashbuilder-runtime.zip` 文件的内容复制到 `<EAP_HOME>/standalone/deployments` 文件夹，其中 `<EAP_HOME>` 是包含 Red Hat JBoss EAP 安装的 Red Hat JBoss EAP 主目录。
5. 在 Red Hat JBoss EAP 主目录中，输入以下命令来创建具有 `admin` 角色的用户，并指定密码。在以下示例中，将 `<USERNAME>` 和 `<PASSWORD>` 替换为您选择的用户名和密码。

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

6. 在终端应用中，导航到 `EAP_HOME/bin`。
7. 输入以下命令启动 Red Hat JBoss EAP :

- 在 Linux 或基于 UNIX 的系统中 :

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

- 在 Windows 中 :

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

8. 在 Web 浏览器中，打开 URL `http://localhost:8080`。
9. 使用您为 Dashbuilder 运行时创建的用户凭据进行登录。
10. 被建议后，上传您从 Business Central 导出的仪表板。Dashbuilder Runtime 使用该仪表板，直到重启为止。

53.1.1. Dashbuilder Runtime 系统属性

您可以使用系统属性自定义 **Dashbuilder** 运行时。

仪表板路径

仪表板上传后，它存储在文件系统中。存储它的路径由系统属性 **dashbuilder.import.base.dir** 控制。默认为 **/tmp/dashbuilder**。

system 属性是任何仪表板模型的根路径。例如，如果此路径上有多个文件，则可通过访问 **Dashbuilder Runtime** 并传递查询参数导入文件，并使用本路径的文件名传递查询参数导入。例如，如果您要加载 **sales_dashboard**，执行 **runtime_host?import=Sales_dashboard** 和 **Dashbuilder Runtime** 将尝试加载文件 **/tmp/dashbuilder/sales_dashboard.zip**。

静态仪表板

如果您希望运行时实例加载特定的仪表板，您可以更改系统属性 **dashbuilder.runtime.import**。将属性设置为本地文件路径将导致在运行时启动时加载特定的仪表板。

控制上传大小

应用程序服务器默认控制 **POST** 请求大小。您可以使用系统属性 **dashbuilder.runtime.upload.size** 控制上传仪表板的允许大小。该大小应当位于 **KB** 中，默认情况下，这个值为 **96kb**，这表示，如果某人尝试上传大于 **96kb** 的文件，则会显示错误，并且控制面板将不会被安装。

Dashbuilder 运行时的默认页面

Dashbuilder Runtime 中导入的仪表板包含一个默认页面。以下列表提供了 **Dashbuilder Runtime default** 页面更新的摘要：

- 当导入的仪表板仅包含一个页面时，它将被用作默认页面。
- 如果页面命名为 **index**，则它将用作默认页面。
- 在其他情况下，使用 **Dashbuilder Runtime** 的通用主页。

加载外部仪表板

通过 **Dashbuilder Runtime** 可以访问位于可访问的 **URL** 的仪表板。您可以通过使用 **import query** 参数传递 **URL** 来访问 **URL**，如 **runtime_host?**

`import=http://filesHost/sales_dashboard.zip`。



注意

出于安全考虑，默认禁用这个选项。您可以通过将系统属性 `dashbuilder.runtime.allowExternal` 设置为 `true` 来启用它。

53.2. 在 RED HAT OPENSIFT CONTAINER PLATFORM 上部署 DASHBUILDER STANDALONE

您可以使用 **Dashbuilder Standalone** 查看在 **OpenShift** 中创建并从 **Business Central** 导出的 **OpenShift** 中的仪表板。这对于查看没有 **Business Central** 的环境中的业务指标很有用。使用 **Dashbuilder Standalone operator** 在 **Red Hat OpenShift Container Platform** 上部署 **Dashbuilder Standalone** 与其他服务分开部署。

先决条件

- **Dashbuilder Standalone** 位于 **OpenShift** 注册表中。
- 您已准备好了 **OpenShift** 环境，如使用 **Operator** 在 **Red Hat OpenShift Container Platform 4** 上部署 **Red Hat Process Automation Manager** 环境所述。
- 您已在 **Business Central** 中创建并导出仪表板。

流程

1. 在 **Operator** 安装 页面中，在 **Application name** 字段中输入应用程序的名称。
2. 在 **Environment** 字段中输入您的环境名称，如 `rhpam-standalone-dashbuilder`。
3. 点击 **Next**。
4. 可选：在 **Security** 页面中，配置 **LDAP** 或 **Red Hat Single Sign-On**。

5. 在 **Components** 页面中，从 **Components** 列表中选择 **Dashbuilder**。

6. 要添加 **KIE** 服务器数据集，请完成以下任务：



注意

您可以通过重复此步骤来添加额外的 **KIE** 服务器数据集。

a. 单击 **Add new KIE Server DataSets**。

b. 在 **DataSet name** 字段中，输入 **kieserver-1**。

c. 在 **Kie Server Location** 字段中，输入 **KIE** 服务器的位置，例如 <https://my-kie-server:80/services/rest/server>。

d. 要设置凭证，请完成以下任务之一：

- 如果您没有设置令牌，在 **Username** 和 **Password** 字段中输入您的用户名和密码。将 **Token** 字段留空。
- 如果您有令牌，在 **Token** 字段中输入您的令牌。将 **Username** 和 **Password** 字段留空。

自定义资源示例：

```
apiVersion: app.kiegroup.org/v2
kind: KieApp
metadata:
  name: standalone-dashbuilder
spec:
  environment: rhpam-standalone-dashbuilder
  objects:
    dashbuilder:
      config:
        kieServerDataSets:
          - name: kieserver-1
            location: 'https://my-kie-server:80/services/rest/server'
```

```

user: kieserverAdmin
password: kieserverAdminPwd
replaceQuery: true

```

7.

要添加 **KIE** 服务器模板，请完成以下任务：



注意

您可以通过重复此步骤来添加额外的 **KIE** 服务器模板。

a.

单击 **Add new KIE Server Templates**。

b.

在 **Template name** 字段中输入模板的名称，如 **kieserver-template**。

c.

在 **KIE 服务器位置** 字段中，输入 **KIE** 服务器的位置，例如 <https://my-other-kie-server:80/services/rest/server>。

d.

要设置凭证，请完成以下任务之一：



如果您没有设置令牌，在 **Username** 和 **Password** 字段中输入您的用户名和密码。将 **Token** 字段留空。



如果您有令牌，在 **Token** 字段中输入您的令牌。将 **Username** 和 **Password** 字段留空。

```

apiVersion: app.kiegroup.org/v2
kind: KieApp
metadata:
  name: standalone-dashbuilder
spec:
  environment: rhpam-standalone-dashbuilder
  objects:
    dashbuilder:
      config:
        kieServerDataSets:
          - name: kieserver-1
            location: 'https://my-kie-server:80/services/rest/server'
            user: kieserverAdmin
            password: kieserverAdminPwd

```

```

    replaceQuery: true
  kieServerTemplates:
  - name: kieserver-template
    location: 'https://my-another-kie-server:80/services/rest/server'
    user: user
    password: pwd
    replaceQuery: true

```

8.

可选：要为外部路由设置自定义主机名，在自定义主机名中输入要在 **Dashbuilder** 外部 **Route** 字段使用的域，如下例所示：

```
`dashbuilder.example.com`
```



注意

自定义主机名必须有效且可解析。

要更改自定义主机名，您可以修改 **routeHostname** 属性。

9.

可选：要启用和设置 **Edge** 终止路由，请完成以下步骤：

a.

在 **Change route termination** 下，选择 **Enable Edge termination**。

b.

可选：在 **Key** 字段中输入私钥。

c.

可选：在 **Certificate** 字段中输入证书。

d.

可选：在 **CaCertificate** 字段中输入 **CaCertificate**。

53.2.1. Dashbuilder 独立环境变量

当您在 **operator** 中使用 **Dashbuilder Container Image** 时，您可以使用环境变量或通过自定义资源来配置 **Dashbuilder**。

表 53.1. 自定义资源参数

参数	等效的环境变量	描述	示例值
allowExternalFileRegister	DASHBUILDER_ALLOW_EXTERNAL_FILE_REGISTER	允许下载外部（远程）文件。默认值为 false。	False
componentEnable	DASHBUILDER_COMPONENT_ENABLE	启用外部组件。	True
componentPartition	DASHBUILDER_COMPONENT_PARTITION	根据 Runtime Model ID 启用组件分区。默认值为 true。	True
configMapProps	DASHBUILDER_CONFIG_MAP_PROPS	允许将 属性文件用于 Dashbuilder 配置。将附加唯一属性；如果设置了多个属性，则使用属性文件中的一个属性。	True
dataSetPartition	DASHBUILDER_DATASET_PARTITION	根据 Runtime Model ID 启用 Dataset ID 分区。默认值为 true。	True
enableBusinessCentral	–	通过自动配置 Business Central 和 Dashbuilder 实现了与 Business Central 的集成。仅在 operator 上提供。	True
enableKieServer	–	通过自动配置 KIE 服务器和 Dashbuilder 启用与 KIE 服务器集成。仅在 operator 上提供。	True
externalCompDir	DASHBUILDER_EXTERNAL_COMP_DIR	设置存储 dashboard ZIP 文件的基础目录。如果启用了 PersistentConfigs ，并且 ExternalCompDir 被设置为现有路径，则使用 /opt/kie/dashbuilder/components 目录。	–
importFileLocation	DASHBUILDER_IMPORT_FILE_LOCATION	设置静态仪表板以自动运行。如果设置了此属性，则不允许导入。	–
importsBaseDir	DASHBUILDER_IMPORTS_BASE_DIR	设置存储 dashboard ZIP 文件的基础目录。如果启用了 PersistentConfigs ，且 ImportsBaseDir 被设置为现有路径，则使用 /opt/kie/dashbuilder/imports 目录。如果 ImportFileLocation 设为 ImportsBaseDir ，则忽略 ImportsBaseDir 。	–

参数	等效的环境变量	描述	示例值
kieServerDataSets	KIESERVER_DATASETS	定义 KIE 服务器数据集访问配置。	–
kieServerTemplates	KIESERVER_SERVER_TEMPLATES	定义 KIE 服务器模板访问配置。	–
modelFileRemoval	DASHBUILDER_MODEL_FILE_REMOVAL	启用从文件系统自动删除模型文件。默认值为 false。	False
modelUpdate	DASHBUILDER_MODEL_UPDATE	允许 Runtime 检查文件系统中的最后更新模型，以更新内容。默认值为 true。	True
persistentConfigs	–	将 Dashbuilder 设置为不是临时的。如果 ImportFileLocation 设置了 PersistentConfigs ，则忽略。默认值为 true。仅在 operator 上可用。	True
runtimeMultipleImport	DASHBUILDER_RUNTIME_MULTIPLE_IMPORT	允许运行时允许导入（多租户）。默认值为 false。	False
uploadSize	DASHBUILDER_UPLOAD_SIZE	设置仪表板上传的大小限制（在 kb 中）。默认值为 10485760 kb。	10485760
env	–	代表容器中存在的环境变量。	–

您可以使用 **operator** 使用 **env** 属性来设置环境变量。以下示例将 **DASHBUILDER_UPLOAD_SIZE** 属性的值设置为 **1000**。

```

apiVersion: app.kiegroup.org/v2
kind: KieApp
metadata:
  name: standalone-dashbuilder
spec:
  environment: rhpam-standalone-dashbuilder
  objects:
    dashbuilder:
      env:
        - name: DASHBUILDER_UPLOAD_SIZE
          value: '1000'

```

第 54 章 数据集编写

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

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

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



注意

Data Sets 页面仅对具有 **admin** 角色的用户可见。

54.1. 添加数据集

您可以创建数据集，以从外部数据源获取数据，并将该数据用于报告组件。

流程

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

这时将打开 **Data Sets** 页面。

2. 点 **New Data Set** 并选择以下供应商类型之一：

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

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



注意

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

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



注意

配置步骤根据您选择的供应商的不同而有所不同。

4. 点击 **Save**。

54.2. 编辑数据集

您可以编辑现有的数据集，以确保获取到报告组件的数据为最新版本。

流程

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

这时将打开 **Data Set Explorer** 页面。
2. 在 **Data Set Explorer** 窗格中，搜索您要编辑的数据集合，选择数据集，然后单击 **Edit**。
3. 在 **Data Set Editor** 窗格中，根据需要使用适当的标签页编辑数据。该选项卡会根据您选择的

数据集供应商类型而有所不同。

例如，以下更改可用于编辑 **CSV** 数据供应商：

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

54.3. 数据刷新

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

54.4. 缓存数据

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

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

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

Business Central 数据集功能提供了两个缓存级别：

- 客户端级别
- 后端级别

您可以在数据集的高级 标签页上设置客户端缓存和后端缓存设置。

客户端缓存

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

后端缓存

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



注意

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

54.5. KIE 服务器数据集，使用 **DASHBUILDER RUNTIME** 和 **DASHBUILDER STANDALONE**

数据集是相关信息的集合。如果您有一个包含导入的数据集的数据集的 **KIE** 服务器，您可以使用 **Dashbuilder Runtime** 或 **Dashbuilder Standalone** 和 **KIE** 服务器 **REST API** 在导入的数据集上运行查询。

因为 KIE 服务器使用 **Business Central** 作为控制器，所以在 **Business Central** 中创建 KIE 服务器容器。数据集也在 **Business Central** 中创建。KIE 服务器配置是一个模板，您可以在创建数据集或安装容器时引用该模板。

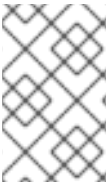
其他服务（如 **Dashbuilder Runtime** 和 **Dashbuilder Standalone**）使用 KIE 服务器 REST API 检索 KIE 服务器信息。**Dashbuilder Runtime** 和 **Dashbuilder Standalone** 访问 KIE 服务器 REST API，以从数据集运行查询。

当在 **Business Central** 中创建 KIE 服务器数据集时，会提供服务器模板信息，供 **Dashbuilder Runtime** 和 **Dashbuilder Standalone** 用于查找 KIE 服务器信息。例如：

```
dashbuilder.kieserver.serverTemplate.{SERVER_TEMPLATE}.location={LOCATION}
dashbuilder.kieserver.serverTemplate.{SERVER_TEMPLATE}.user={USER}
dashbuilder.kieserver.serverTemplate.{SERVER_TEMPLATE}.password={PASSWORD}
dashbuilder.kieserver.serverTemplate.{SERVER_TEMPLATE}.token={TOKEN}
```

您还可以为每个数据集设置 KIE 服务器。例如：

```
dashbuilder.kieserver.dataset.{DATA_SET_NAME}.location={LOCATION}
dashbuilder.kieserver.dataset.{DATA_SET_NAME}.user={USER}
dashbuilder.kieserver.dataset.{DATA_SET_NAME}.password={PASSWORD}
dashbuilder.kieserver.dataset.{DATA_SET_NAME}.token={TOKEN}
```



注意

如果提供凭证，则不会使用令牌身份验证。

您可能需要针对另一个 KIE 服务器安装运行仪表板。当在开发环境中的 KIE 服务器上创建数据集时，数据集查询会在开发 KIE 服务器上创建，如 **DEV**。如果将仪表板导出到生产环境，例如 **PROD**，具有不同 KIE 服务器，则您在 **DEV** 中创建的查询将无法使用，因此引发错误。在这种情况下，可以使用替换查询功能（通过服务器模板或数据集）从数据设置为另一个 KIE 服务器的端口查询：

- 服务器模板示例：

```
dashbuilder.kieserver.serverTemplate.{SERVER_TEMPLATE}.replace_query=true
```

- 数据集示例：

```
dashbuilder.kieserver.dataset.{DATA_SET_NAME}.replace_query=true
```

`replace_query=true` 属性 只需要设置一次，以便 **Dashbuilder Runtime** 或 **Dashbuilder Standalone** 创建查询。创建查询后，您可以删除此系统属性。

其他资源

- [使用 KIE API 与 Red Hat Process Automation Manager 交互](#)

第 55 章 页面编写

页面是以下组件的集合：

- 核心组件
- 导航组件
- 报告组件
- **Heatmaps**

另外，页面可以没有或者任何数量的组件。**Page Editor** 工具用于编辑页面。

页包括 **Fluid** 风格或 页面 样式。**Fluid** 风格是一个带有垂直滚动条的类网页，当页面超过可用高度时。**Page** 风格是一个网页，它始终适用于窗口高度。

55.1. 创建页面

您可以使用页面视角创建一个由不同类型的组件组成的页面。创建页面及其定义的所有组件后，根据需要 使用 页面编辑器 保存、删除、重命名或复制页面。

以下流程描述了如何创建页面并在其中添加所需的组件：

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。
2. 在 页面 面板中，单击新建。或者，单击 **Page Editor** 窗格上的 **New Page**。
3. 在 **New Page** 对话框中，在 **Name** 字段中输入值，再选择所需的风格。

4. 点 **确定**。新的页面会在 **页面 编辑器** 中打开。
5. 在 **组件 窗格** 中，展开组件，并将所需的组件类型拖到编辑器中。
6. 将组件放置到页面中后，从 **Properties** 窗格中编辑其属性。
7. 单击 **Save**，然后再次单击 **Save**。

55.2. 保存、删除、重命名或复制页面

创建并定义了页面后，根据需要使用 **页面编辑器** 保存、删除、重命名或复制页面。

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。
2. 在页面面板中 选择页面。页面会在 **页面 编辑器** 中打开。
3. 执行所需操作，然后从 **页面编辑器** 的右上角选择 **Save**、**Delete**、**Rename** 或 **Copy**。

图 55.1. 保存、删除、重命名或复制页面



55.3. 导航树

Workbench 导航树包含 **Business Central** 的主 菜单中显示的条目。对此导航树结构的更改将反映在 **Home** 页面的主菜单中。例如，可以使用此类机制发布新的页面。

另外，也可以创建额外的导航树。这些自定义树可用于在页面中配置导航组件。您不能从导航面板中删除 **Workbench** 树，但您可以编辑 **Workbench** 树层次结构以符合您的要求。此功能帮助用户使用新的页面自定义 **Business Central** 的主菜单。



注意

默认情况下，在导航面板中显示的 **Workbench** 树是 **Business Central** 的 **Main** 菜单。

55.3.1. 创建导航树

您可以根据需要创建多个自定义导航树。自定义导航树在一个方面与默认 **Workbench** 导航树不同。可以删除自定义导航树，但无法从 **Business Central** 中删除默认树。它们可以包含 **Workbench** 树中的默认组和条目，以及用户创建的组和树。

先决条件

- 有足够的用户权限来创建导航树。

流程

1. 登录到 **Business Central** 并前往 **Menu** → **Design** → **Pages**。
2. 选择 导航面板，然后单击 **New**。
3. 输入新导航树的名称并单击复选标记图标，或者按 **Enter** 键。
4. 单击 **Save**。

55.3.2. 编辑导航树

您可以使用 **Pages** 视角中的导航面板编辑自定义导航树。您可以通过添加组、划分和页条目、删除树特定条目和重新排序、重命名或删除它们来进一步自定义树。

先决条件

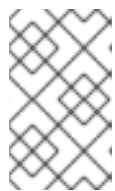
- 有足够的用户权限来编辑导航树。

55.3.3. 将组、划分和页面条目添加到导航树

您可以将组、划分和页面条目添加到导航树。

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。
2. 点击导航面板并选择您想要添加条目的导航树。
3. 单击树的 **gear** 图标，然后选择 **New Group**、**New Divider** 或 **New Page**。
4. 输入新组或页面的名称，点勾号图标或按 **Enter**。



注意

划分程序条目没有 **name** 属性。

5. 点击 **Save**。

55.3.4. 重新排序导航树

在导航面板中，您可以对导航树及其条目进行重新排序。



注意

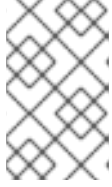
树条目的重新排序选项将根据它们在树形层次结构中的位置而有所不同。

重新排序导航树

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。

2. 单击 导航面板并选择您要重新排序的导航树。
3. 单击 树的 **gear** 图标，然后根据需要上移或下移。



注意

导航树的第一个和最后一个条目只有两个重新排序选项。

4. 单击 **Save**。

重新排序导航树的条目

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。
2. 单击 导航面板，然后展开导航树。
3. 点您要重新排序的条目的 **gear** 图标，然后根据需要上移或下移。
4. 可选：单击 **Goto Page** 来查看所选页面。
5. 单击 **Save**。

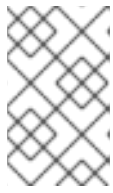
55.3.5. 重命名导航树

您可以重命名除 **Workbench** 树外的所有导航树。

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。

2. 点击导航面板并选择您要重命名的自定义导航树。



注意

要重命名树的条目，请展开树并选择您想要重命名的条目。

3. 单击树或树条目的编辑图标。

4. 输入树的新名称，然后单击复选标记图标。



注意

您不能重命名划分条目。

5. 点击 **Save**。

55.3.6. 删除导航树

您可以从页面透视图的导航面板中删除除 **Workbench** 树外的任何 导航 树。

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。
2. 在导航面板中，选择您要删除的导航树，然后点删除图标。
3. 点击 **Save**。

55.3.7. 删除导航树的条目

您可以删除导航树的条目。

流程

1. 在 **Business Central** 中, 转至 **Menu** → **Design** → **Pages**。
2. 点击 导航面板。
3. 展开包含您要删除的条目的树。
4. 点 条目的删除图标。
5. 点击 **Save**。

55.4. 组件

页面包含不同类型的组件。您可以使用组件面板在页面中使用以下组件类型：

- 核心组件
- 导航组件
- 报告组件
- **Heatmap** 组件

55.4.1. 核心组件

您可以使用核心组件 指定自定义 **HTML** 信息或显示现有的页面。核心组件有三种类型：



注意

核心组件不是必须的。

表 55.1. 核心组件子类型

核心组件子类型	描述
HTML	此组件会打开 HTML 编辑器小部件，用于使用文本、镜像、表、链接和颜色创建 HTML 页面。如果需要，您还可以自定义 HTML 页面。
页面	此组件允许您将之前创建的 HTML 页面添加到新仪表板中。您可以使用此组件嵌套您在仪表板中创建的自定义页面。
徽标	此组件允许您在页面中添加镜像。要在页面上添加镜像，您可以提供一个镜像的 URL，并根据要求设置镜像的宽度和高度。默认镜像是 Dashbuilder 的徽标。Logo URL 字段是必需的。

55.4.2. 导航组件

导航组件用于在页面之间导航。导航组件有六种类型的导航组件。

表 55.2. 导航组件子类型

导航组件子类型	描述
目标 Div	此组件显示标签列表、菜单栏中和导航树的条目。另外，它会跟踪上次点击的项目。
菜单栏	此组件以菜单栏中的格式显示导航树的条目。对于 Business Central 支持的级别数量没有限制。
标题的 Navigator	此组件以标题的形式显示导航组。组显示为文件夹，但选择单个条目和内容时会显示内容。
树(tree)	此组件显示垂直树结构表单中的条目。
Carousel	此组件以 carousel 或滑块显示形式显示所选页面。
标签列表	此组件将所选菜单页面显示为组件顶部的标签页。



注意

非目标流组件（如 **carousel** 或 **标题导航程序**）不需要 **Target Div** 设置。

55.4.3. 报告组件

Reporting 组件用于显示数据收集的数据，格式为图形、表、映射。报告组件有十大类型。有关数据集的更多信息，请参阅 [数据集编写](#) 部分。

您可以使用 **New Displayer** 小部件配置报告组件，其中包含以下标签页：

- **键入**：允许您选择如何以图形方式显示自定义数据。
- **Data**：使您能够从 **Settings** 菜单中提供的 **Data Sets** 部分中从您创建的自定义数据集列表中选择数据集。
- **显示**：使您能够选择和自定义如何通过添加标题、更改颜色、大小自定义内容显示方式。



注意

对于 **Time Series Chart** 组件，**New Displayer widget** 包含 **Data**、**Display** 和 **Component Editor** 选项卡。

表 55.3. 报告组件子类型

报告组件子类型	描述
bar	此组件用于以条图表格式显示数据集的数据。
pie	此组件用于显示数据收集的数据，格式为 pie Chart。在饼图中，每个片段的rc 长度与代表的数量成比例。
行	此组件用于将数据收集数据显示为由两个 axes 上直接线片段连接的一系列数据点。
区域	此组件包含一行 chart 和条图表，用于显示数据收集的数据。
bubble	此组件在双维图表中显示多个圆圈（兆字节）。它是 scatter 图表的规范化，它用 bubbles 替换点。
meter	此组件用于以计量格式显示数据集的数据。
map	此组件可让您在上下文（通常是地理）中定位数据，使用不同的层。数据值以映射上的标记的形式显示。数据值可以充当协调或地址。

报告组件子类型	描述
指标	此组件用于以指标格式显示数据集的数据。您可以使用 Preview 、 HTML 或 Javascript 选项卡来编辑数据。
表	此组件用于以表格格式显示数据集的数据。如果需要，您可以隐藏或显示列。
Filter	此组件允许您过滤数据收集的数据。
时间序列数据库图	此组件用于以时间序列格式显示数据集的数据。

55.4.4. 组件属性

页面中使用的组件关联有不同的属性。**Properties** 面板允许您编辑以下属性来自定义组件：

- **面板 属性**：用于自定义组件面板属性，如 **Width**、**Height** 和 **Background Color**。
- **边缘属性**：用于自定义组件边缘属性，如 **top**、**Bottom**、**Left** 和 **right**。
- **padding 属性**：用于自定义组件 **padding** 属性，如 **Top**、**Bottom**、**Left** 和 **right**。

55.4.5. 将组件放在页面编辑器上以创建页面

要创建页面，您必须将组件拖到 **Pages** 视角的 **Editor** 可以撤离。将所有必需的组件放在页面后，点 **Save**。

55.4.6. 使用 **Preview** 选项卡预览页面

在创建或编辑页面时，点 **Page Editor** 上的 **Preview** 选项卡来预览页面，然后再保存该页面。

55.4.7. 在页面中添加时间序列 **chart** 组件

您可以使用 **Time Series Chart** 组件来代表任何时间序列数据。您可以创建自己的仪表板，以连接到您的时间序列数据集。

您可以导出时间序列组件到 **Dashbuilder Runtime**，并从 **KIE 服务器**或任何 **Prometheus** 数据集检索信息。您还可以使用时间序列 **chart** 组件创建、编辑和构建仪表盘。

先决条件

- **KIE 服务器已部署并连接到 *Business Central*。**

流程

1. 使用以下命令，创建一个新的 **KIE 服务器数据集**：
 - i. 在 **Business Central** 中，前往 **Admin → Data Sets**。

这时将打开 **Data Set Explorer** 页面。
 - ii. 点 **New Data Set** 并选择任何供应商类型，根据您的要求。

这时将打开 **Data Set Creation Wizard** 页面。
 - iii. 在所选提供程序类型的 **Data Set Creation Wizard** 中输入所需的详情，点 **Test**。
 - iv. 点击 **Save**。
2. 在 **Business Central** 中，转至 **Menu → Design → Pages**。
3. 在 **页面** 面板中，单击新建。
4. 单击 **确定**。
5. 在 **New Page** 对话框中，在 **Name** 字段中输入名称并选择所需风格。

新的页面会在 **页面编辑器** 中打开。

6. 在组件面板中，展开 **Reporting** 组件，并将 **Time Series Chart** 拖到页面编辑器中。
7. 在 **Displayer** 编辑器向导中，点 **Data** 选项卡并选择您创建的数据集。
8. 在 **Data** 选项卡中，根据要求选择 **Columns** 字段中的值。
9. 单击 **Display** 选项卡，再根据需要编辑 **Chart**、**Margins**、**Filter**、**刷新** 和 **Columns** 的值。
10. 点 **Component Editor** 选项卡，将以下组件属性更新到 **Component Properties** 字段中：

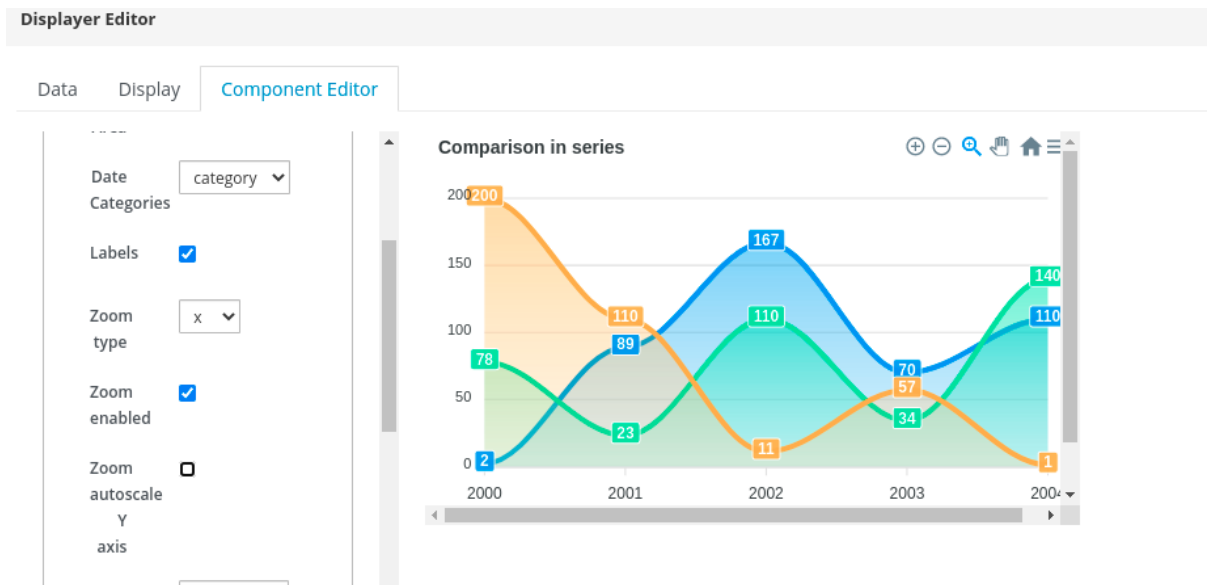
表 55.4. 时间序列组件属性

组件属性	描述
Transpose dataset	定义提供的数据集将时间序列用作单独的列或行。
显示区域	选择将类型设置为行 chart 或一个区域图表的复选框。
日期类别	从下拉列表中选择 类别、日期时间 或数字选项。
标签	选择在数据点上启用或禁用数据标签的复选框。
zoom 类型	从下拉列表中选择 x、y 或 xy 选项。
启用 zoom	选择在 xis chart 中启用缩放的复选框。默认情况下选中该复选框。
zoom autoscale Y axis	选择根据可见区域重新缩放高和低的复选框。
自动选择工具栏	从下拉列表中选择 zoom、os 或 pan 选项。
标题文本	编辑时间序列 chart 组件的标题。
标题对齐	从下拉列表中选择 左侧、中心 或 右 选项，以更改标题对齐。

组件属性	描述
显示工具栏	它是在图表右上角启用或禁用工具栏的复选框。默认情况下选中此复选框。如果启用了这个组件属性，您可以使用 zoom in、zoom out、osoom 和 panning 功能。
Chart 名称	根据您的要求设置 chart 名称。默认情况下，图表名称设置为 Newchart 。

11. 可选：点击图表右上角的 **sandwich** 菜单图标，以下载 **CSV**、**PNG** 或 **SVG** 格式的数据集。
12. 点击 **确定**。

图 55.2. 时间序列组件示例



55.5. HEATMAP 组件

在 **Business Central** 中，您可以在页面中添加热图组件。**Heatmap** 组件用于显示进程图表中的 **heat** 信息。进程图节点上的颜色与分配给每个节点的值相关，并根据分配的值，颜色根据进程图表而有所不同。如果分配的值最大，则 **heat intensifies** 和分配了最小值时，不会在进程图表中显示 **heat**。

您可以将 **heatmap** 组件导出到 **Dashbuilder Runtime** 或 **Dashbuilder Standalone** 并从 **KIE 服务器** 数据集检索 **heat** 信息。您还可以使用 **heatmap** 组件创建、编辑和构建仪表板。

55.5.1. 为进程创建 heatmap 组件

您可以在 **Business Central** 中为特定进程创建 **heatmap** 组件。

先决条件

- **KIE 服务器已部署并连接到 Business Central。**
- 您已在 **Business Central** 中创建一个具有至少一个业务流程资产的项目。
- 已在 **Business Central** 中部署了带有进程定义的项目。
- 创建示例进程实例。

流程

1. 使用以下命令，创建一个新的 **KIE 服务器数据集**：
 - i. 在 **Business Central** 中，前往 **Admin → Data Sets**。

这时将打开 **Data Set Explorer** 页面。
 - ii. 点 **New Data Set** 并选择 **Execution Server provider type**。

这时将打开 **Data Set Creation Wizard** 页面。
 - iii. 输入 **dataset** 的名称。
 - iv. 选择服务器配置。如果部署了项目，则服务器配置可用。
 - v. 从列表中选择 **CUSTOM** 查询目标。

vi.

在 **Query** 字段中输入以下自定义 **SQL** 查询。

```

select
  pil.externalId,
  pil.processId,
  nil.nodeid,
  nil.nodeType,
  nil.nodeName,
  count(nil.nodeid) as total_hits
from
  NodeInstanceLog nil
inner join
  ProcessInstanceLog pil on pil.processInstanceId = nil.processInstanceId
where
  nil.type = 1
group by
  pil.externalId,
  nil.nodeid,
  nil.nodeName

```



注意

如果需要，您可以根据您的数据库修改 **SQL** 查询。

vii.

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

viii.

点击 **Save**。

2.

在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。

3.

在 **页面** 面板中，单击新建。

4.

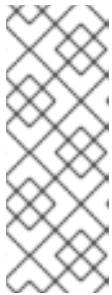
在 **New Page** 对话框中，在 **Name** 字段中输入值，再选择所需的风格。

5.

点击 **确定**。

新页面在 **页面 Editor** 中打开，您可以看到 **Heatmaps** 组件在 **Components** 面板中可用。

6. 在组件面板中，展开 **Heatmaps** 组件，并将 **Process Heatmap** 组件类型拖到 页面编辑器 中。
7. 在 **Displayer** 编辑器 向导中，点 **Data** 选项卡并选择新创建的 **KIE Server dataset**。
8. 在 **Data** 选项卡中，从 **Columns** 字段中选择 **NODEID** 和 **TOTAL_HITS**。
9. 点 **Component Editor** 选项卡，在组件 属性 字段中输入强制字段的值，包括 **Server Template**、容器 **ID** 和流程定义 **ID**。

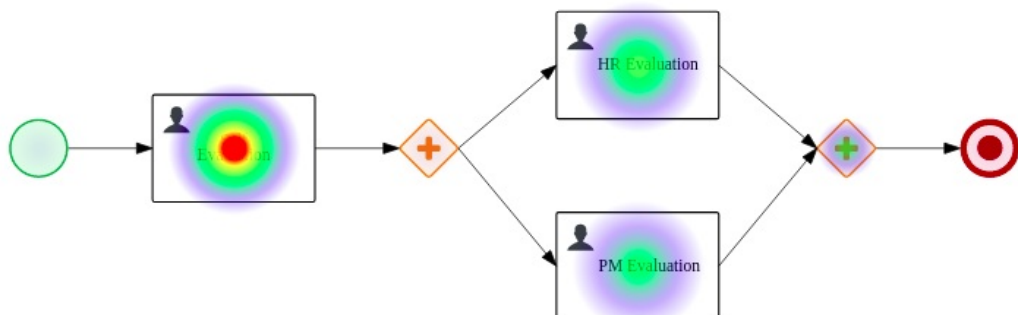


注意

要访问 **Server Template** 的值，请转至 **Deploy** → **Execution Servers** → **Server configuration**。对于容器 **ID** 的值，进入 **Manage** → **Process Instances**，您可以点您要使用的进程实例，在 **Deployment** 中，**Deployment** 对应于容器 **ID**，定义 **ID** 是进程 定义 **ID**。

10. 单击 **Display** 选项卡，再根据需要编辑 **Chart**、**Margins**、**Filter**、**Refresh** 和 **Columns** 的值。
11. 点击 确定。

图 55.3. 进程 heatmap 组件示例



您可以查看进程图表的 **heat** 信息。

55.5.2. 为多个进程创建 heatmap 组件

您可以为 **Business Central** 中的多个进程创建 heatmap 组件。

先决条件

- **KIE 服务器已部署并连接到 Business Central。**
- 您已在 **Business Central** 中创建多个项目，其中至少包含一个业务流程资产。
- 已在 **Business Central** 中部署了带有进程定义的项目。
- 创建示例进程实例。

流程

1. 使用以下步骤创建新的 **KIE 服务器数据集**：
 - i. 在 **Business Central** 中，前往 **Admin** → **Data Sets**。

这时将打开 **Data Set Explorer** 页面。
 - ii. 点 **New Data Set** 并选择 **Execution Server provider type**。

这时将打开 **Data Set Creation Wizard** 页面。
 - iii. 输入 **dataset** 的名称。
 - iv. 选择服务器配置。如果部署了项目，则服务器配置可用。
 - v. 从列表中选择 **CUSTOM** 查询目标。

vi.

在 **Query** 字段中输入以下自定义 **SQL** 查询。

```

select
  pil.externalId,
  pil.processId,
  nil.nodeid,
  nil.nodeType,
  nil.nodeName,
  count(nil.nodeid) as total_hits
from
  NodeInstanceLog nil
inner join
  ProcessInstanceLog pil on pil.processInstanceId = nil.processInstanceId
where
  nil.type = 1
group by
  pil.externalId,
  nil.nodeid,
  nil.nodeName

```



注意

如果需要，您可以根据您的数据库修改 **SQL** 查询。

vii.

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

viii.

点击 **Save**。

2.

在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。

3.

在 **页面** 面板中，单击 **新建**。

4.

在 **New Page** 对话框中，在 **Name** 字段中输入值，再选择所需的风格。

5.

点击 **确定**。

新页面在 **页面 Editor** 中打开，您可以看到 **Heatmaps** 组件在 **Components** 面板中可用。

6. 在组件面板中，展开 **Heatmaps** 组件，并将 **All Process Heatmaps** 组件类型拖到 **canvas** 中。
7. 在 **Displayer** 编辑器向导中，点 **Data** 选项卡并选择新创建的 **KIE Server dataset**。
8. 在 **Data** 选项卡中，从 **Columns** 字段中选择 **EXTERNAL ID**、**PROCESSID**、**NODEID** 和 **TOTAL_HITS**。
9. 在 **Process Selector** 框中，根据需要选择 **Container** 和 **Process** 值。
10. 点 **Component Editor** 选项卡，然后在 **Server Template (mandatory)** 字段中输入值。

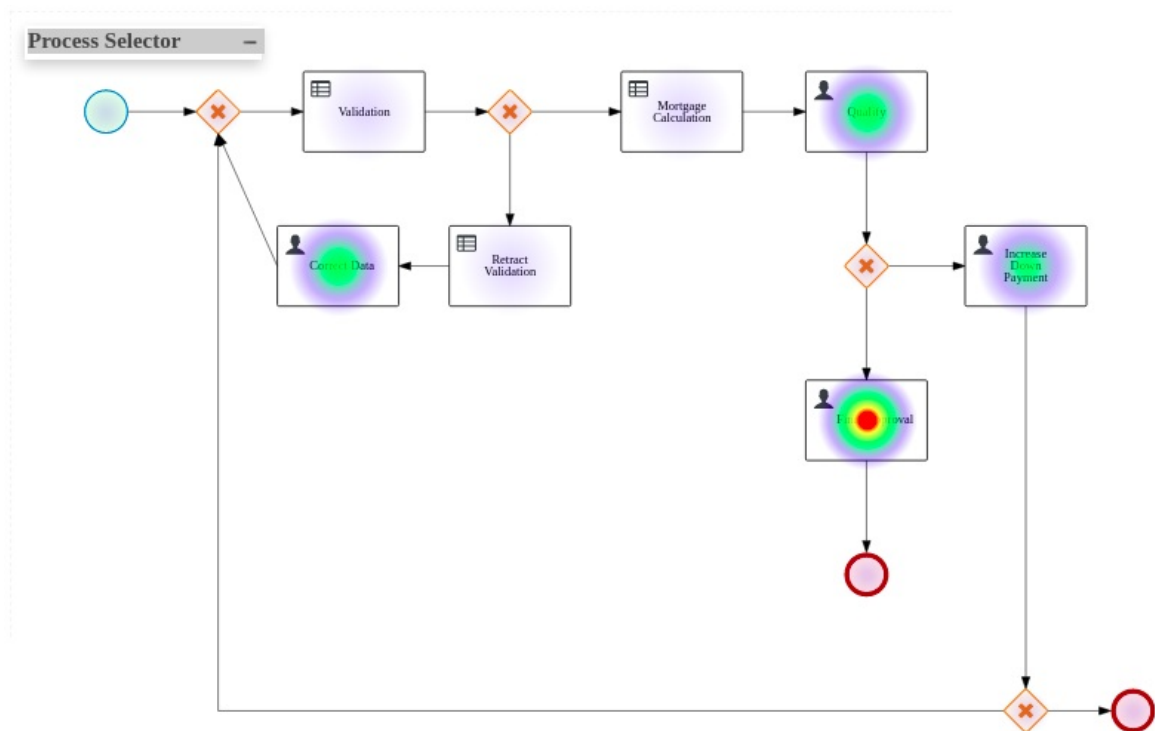


注意

要访问 **Server Template** 的值，请转至 **Deploy** → **Execution Servers** → **Server configuration**。

11. 单击 **Display** 选项卡，再根据需要编辑 **Chart**、**Margins**、**Filter**、**Refresh** 和 **Columns** 的值。
12. 单击 **+OK**。

图 55.4. 多个进程 heatmap 组件示例



您可以查看进程图表的 **heat** 信息。

55.5.3. 执行 heatmap 组件

您可以在 **Business Central** 之外执行内部 **heatmap** 组件，仅用于测试目的。相同的 **API** 用于创建外部组件，您可以使用它来构建自己的组件。有关外部组件的更多信息，请参阅第 55.6 节“外部组件”

要运行特定的 **heatmap** 组件，请完成以下步骤：

先决条件

- 您在系统中安装了 **npm**。有关安装 **npm** 的更多信息，请参阅下载并安装 [Node.js](#) 和 [npm](#)。
- 您已在系统中安装了 **Yarn**。有关安装 **Yarn** 的更多信息，请参阅 [Yarn 安装](#)。
- 您已克隆了 [Appformer 存储库](#)，以运行 **Business Central** 之外的组件。

流程

1. 进入 `appformer/dashbuilder/dashbuilder-shared/dashbuilder-js` 目录。

2. 在 `dashbuilder-js` 目录中打开终端并输入以下命令：

```
yarn run init && yarn run build:fast
```

您可以在 `dashbuilder-js/packages` 目录中看到以下组件：

- `processes-heatmaps-component`
- `process-heatmap-component`
- 徽标-`component`
- `heatmap-component`

3. 进入 `dashbuilder-js/packages` 目录，打开所需的 `heatmap` 组件，然后在终端中输入以下命令：

```
yarn run start
```

4. 要访问组件，在网页浏览器中输入 <http://localhost:9001/>。

所选组件在 `web` 浏览器中显示。

55.6. 外部组件

在 **Business Central** 中，您可以在页面中添加外部组件。组件默认是禁用的。要启用外部组件，将 `dashbuilder.components.enable` 系统属性的值更改为 `true`。

使用 `dashbuilder.components.dir` 系统属性设置和配置外部组件位置。此系统属性的默认值为 `/tmp/dashbuilder/components`。您必须在组件目录中设置组件，并带有一个父目录，用作组件 ID。例

如，如果组件 ID 是 `mycomp`，并且组件目录是 `/tmp/dashbuilder/components`，则组件基础目录为 `/tmp/dashbuilder/components/mycomp`。

Business Central 检查组件目录中的 `manifest.json` 文件。`manifest.json` 必须至少包含一个名称文本参数。

表 55.5. `manifest.json` file descriptions

参数	描述
<code>name</code>	组件部分中显示的 组件名称。
图标	组件图标显示在 组件 部分。
<code>noData</code>	表示组件不需要数据收集的标记。
<code>parameters</code>	参数列表使用 <code>ComponentParameter</code> 类型。支持的参数类型包括 名称、类型、类别、 <code>defaultValue</code> 、标签、强制和 <code>comboValues</code> 。

`manifest.json` 文件示例

```
{
  "name": "Heat Map Experiment",
  "icon": "fa fa-bell-o",
  "parameters": [
    {
      "name": "svg",
      "type": "text",
      "defaultValue": "",
      "label": "SVG XML",
      "category": "SVG Content"
      "mandatory": true
    },
    {
      "name": "svgUrl",
      "type": "text",
      "defaultValue": "",
      "label": "SVG URL",
      "category": "SVG URL"
      "mandatory": true
    }
  ],
  {
    "name": "ksProcessId",
    "type": "text",
    "defaultValue": "",
```

```

    "label": "Process ID",
    "category": "Kie Server"
    "mandatory": true
  }
]
}

```

55.6.1. 创建外部组件

以下流程描述了如何在页面中创建和添加外部组件：

流程

1.

在组件目录中设置含有父目录的组件。

例如，如果组件 ID 是 **mycomp**，并且组件目录是 **/tmp/dashbuilder/components**，则组件基础目录为 **/tmp/dashbuilder/components/mycomp**。

2.

在组件目录中创建 **manifest.json** 文件。

3.

使用 **HTML** 内容创建 **index.html** 文件。

4.

在终端应用中，导航到 **EAP_HOME/bin**。

5.

要启用外部组件，将 **dashbuilder.components.enable** 系统属性的值设置为 **true**：

```

$ ~/EAP_HOME/bin/standalone.sh -c standalone-full.xml
-Ddashbuilder.components.dir={component directory base path} -
Ddashbuilder.components.enable=true

```

6.

启动 **Business Central**，前往 **Menu** → **Design** → **Pages**。

外部组件在组件窗格下提供。

7. 在组件窗格中，展开 **External** 组件，并将所需的组件类型拖到 **canvas** 中。

8. 点击 **Save**。

第 56 章 安全管理

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

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

56.1. 安全管理供应商

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

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



注意

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

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

56.1.1. 根据属性文件配置红帽 **JBoss EAP** 安全管理提供程序

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

先决条件

- 安装了红帽 **JBoss EAP**。

流程

1. 要使用红帽 **JBoss EAP** 实例中的现有用户或角色属性，请在

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

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

下表提供了这些属性的描述和默认值：

表 56.1. 基于属性文件的红帽 JBoss EAP 安全管理供应商

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

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

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

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

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

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

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

先决条件

- 安装了红帽 **JBoss EAP**。

流程

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

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

下表提供了这些属性的描述和默认值：

表 56.2. 基于属性文件和 **CLI** 模式的红帽 **JBoss EAP** 安全管理供应商

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

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

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

3.

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

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

56.2. 权限和设置

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

- 查看页面。
- 保存项目。
- 查看存储库。
- 删除仪表板

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

56.2.1. 在 **Business Central** 中更改组和角色的权限

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



注意

对角色或组进行的任何更改会影响与该角色或组关联的所有用户。

先决条件

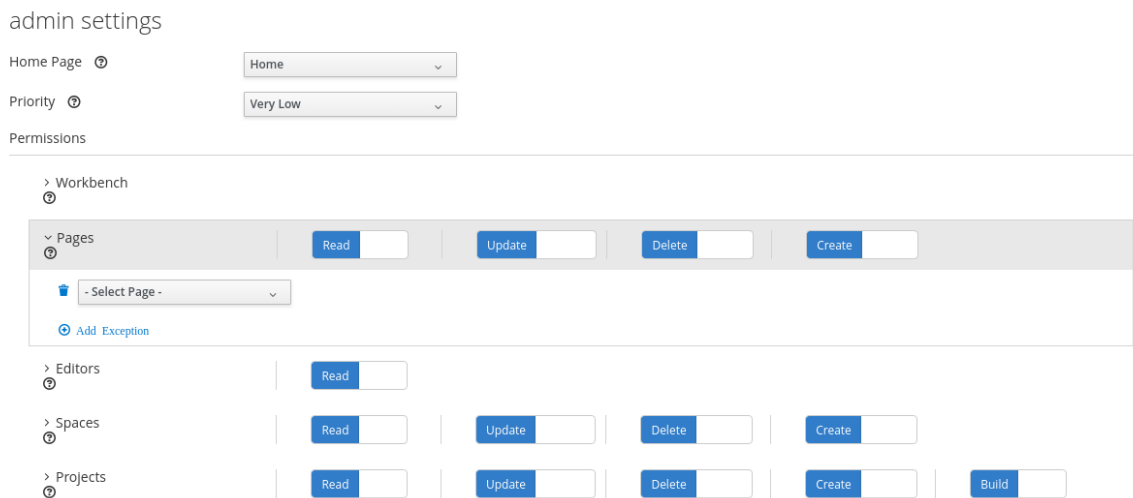
- 使用 **admin** 用户角色登录到 **Business Central**。

流程

1. 要访问 **Business Central** 中的 安全管理 页面，可选择屏幕右上角的 **Admin** 图标。
2. 单击 **Business Central Settings** 页面中的 **Roles**、**Groups** 或 **Users**。

Security 管理页面 会在您点击的图标的标签页中打开。
3. 在列表中点击您要编辑的角色或组。所有详情都显示在右侧面板中。
4. 在 **Settings** 部分下设置 主页 或 优先级。
5. 在 **Permissions** 部分设置 **Business Central**、页面、编辑器、空格和项目权限。

图 56.1. 设置权限



6. 单击资源类型旁边的箭头，以展开您要更改权限的资源类型。
7. 可选：要添加资源类型的异常，请单击 **Add Exception**，然后根据需要设置权限。



注意

您不能在 **Business Central** 资源类型中添加例外。

8. 单击 **Save**。

56.2.2. 更改 **Business Central** 主页

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

流程

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



注意

角色或组必须具有对页面的读访问权限，然后才能使它成为主页。

56.2.3. 设置优先级

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

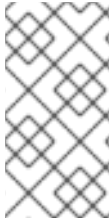
先决条件

- 使用 **admin** 用户角色登录到 **Business Central**。

流程

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

2. 选择角色或组。
3. 从优先级菜单中选择优先级，然后单击 **Save**。



注意

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

第 57 章 导出、导入和部署仪表板

在 **Business Central** 中创建仪表板后，您可以导出仪表板数据，并将其导入到 **Business Central** 的另一个实例、**Dashbuilder Runtime** 或 **Dashbuilder Standalone**。



注意

这个功能只能由管理员用户访问。

57.1. 导出 **BUSINESS CENTRAL** 仪表板数据

您可以将数据收集、页面和导航等仪表板数据导出为 **ZIP** 文件。

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，再选择 **Dashbuilder Data Transfer**。

2. 要导出仪表板数据，请完成以下任一任务：

如果要将所有仪表板数据导出为 **ZIP** 文件，请完成以下任务：

- i. 在 **Dashbuilder Data Transfer** 页面上，单击 **Export all**。

下载包含所有仪表板数据的 **export.zip** 文件。**export.zip** 文件结构由数据类型分隔，如下例所示：

```
dashbuilder/datasets/definitions/dataset-example1.dset
dashbuilder/datasets/definitions/dataset-example2.dset
dashbuilder/datasets/readme.md
dashbuilder/perspectives/page1/perspective_layout
dashbuilder/perspectives/page1/perspective_layout.plugin
dashbuilder/perspectives/page2/perspective_layout
dashbuilder/perspectives/page2/perspective_layout.plugin
dashbuilder/perspectives/readme.md
dashbuilder/navigation/navigation/navtree.json
dashbuilder/navigation/readme.md
VERSION
```

如果要导出创建的自定义用户，并作为 **ZIP** 文件提供仪表板数据，请完成以下任务：

- i. 在 **Dashbuilder Data Transfer** 页面上，单击 **Custom export**。
- ii. 选择您在 **Export Wizard** 面板中的 **ZIP** 文件中包含的数据集和页面，然后单击 **Next**。

Export Wizard 面板验证所选数据集和页面。数据收集和页面的摘要包括在面板中。



注意

导航总是包含在导出的 **ZIP** 文件中。

- iii. 如果您的导出就绪，点 **Download**。

下载包含自定义仪表板数据的 **export.zip** 文件。



注意

您必须在 **Export Wizard** 面板中选择关联的页面和数据集。如果您没有同时选择数据集和页面，则会生成一个错误，且无法下载 **export.zip** 文件。因此，您必须至少选择一个页面。

- iv. 点 **Finish**。

57.2. 导入 **BUSINESS CENTRAL** 仪表板数据

如果归档的结构的结构，您可以使用 **ZIP** 文件将您在 **Business Central** 中创建的仪表板数据导入到另一个 **Business Central** 实例，如下例所示：

```
dashbuilder/datasets/definitions/dataset-example1.dset
dashbuilder/datasets/definitions/dataset-example2.dset
dashbuilder/datasets/readme.md
dashbuilder/perspectives/page1/perspective_layout
dashbuilder/perspectives/page1/perspective_layout.plugin
dashbuilder/perspectives/page2/perspective_layout
dashbuilder/perspectives/page2/perspective_layout.plugin
```

dashbuilder/perspectives/readme.md
dashbuilder/navigation/navigation/navtree.json
dashbuilder/navigation/readme.md
VERSION

流程

1. 在 **Business Central** 中，选择屏幕右上角的 **Admin** 图标，再选择 **Dashbuilder Data Transfer**。



警告

您必须导入仪表板数据到 **Red Hat Process Automation Manager** 的干净安装，以避免覆盖现有系统中的数据。

2. 在 **Dashbuilder Data Transfer** 页面上，单击 **Choose File** 图标。
3. 导航到您要导入的 **ZIP** 文件并选择文件。
4. 点 **Upload** 图标。
5. 点 **Import**。

57.3. 在 **DASHBUILDER RUNTIME** 上部署 **BUSINESS CENTRAL** 的仪表板

您可以在 **Dashbuilder Runtime** 上自动部署 **Business Central** 中的仪表板。**Business Central** 使用逐步导出功能链接到 **Dashbuilder** 运行时。

先决条件

- 在您的系统上配置了 **Dashbuilder** 运行时。

- 在 `standalone.xml` 文件中，您已将 `dashbuilder.runtime.multi` 系统属性设置为 `true`。
- 您已将 `dashbuilder.runtime.location` 系统属性的值设置为 **Dashbuilder Runtime URL**，如下例所示：

```
<property name="dashbuilder.runtime.location" value="http://localhost:8080"
```

- 您已将 `dashbuilder.export.dir` 系统属性设置为共享目录，其中的 **Dashbuilder Runtime** 读取其模型，如下例所示：

```
<property name="dashbuilder.export.dir" value="/tmp/dashbuilder/models/"
```

流程

1. 在 **Business Central** 中，转至 **Menu** → **Design** → **Pages**。
2. 在 **Components** 面板中，根据需要拖放并编辑所需的组件类型到 **canvas**，点 **Save to finish**。
3. 选择屏幕右上角的 **Admin** 图标，再选择 **Dashbuilder Data Transfer**。
4. 在 **Dashbuilder Data Transfer** 页面上，单击 **Custom export**。
5. 在 **Export Wizard** 面板中选择要在 **ZIP** 文件中包含的页面，然后点 **Next**。
6. 选择 **Export Wizard** 并点 **Open**。

您可以看到 **Dashbuilder Runtime** 主页。如果您没有登录，您会被重定向到登录页面。

7. 进入 **Dashboards** → **Runtime Dashboards**，您可以看到页面。

所选数据导出，**Dashbuilder Runtime** 会在打开时更新模型内容。

附录 A. 版本信息

文档最后于 **2023 年 9 月 5 日** 星期二更新。

附录 B. 联系信息

Red Hat Process Automation Manager 文档团队 : brms-docs@redhat.com