Red Hat Process Automation Manager 7.13

在 Red Hat Process Automation Manager 中开发决策服务

Last Updated: 2024-03-19

Red Hat Process Automation Manager 7.13 在 Red Hat Process Automation Manager 中开发决策服务

法律通告

摘要

本文档论述了如何使用决策模型和 Notation(DMN)模型、EVolumeDB 规则语言(DRL)文件、指导决策表和其他决策资产开发决策服务。

目录

前言

作为业务决策的开发人员，您可以使用 Red Hat Process Automation Manager 来使用 Decision Model 和 Notation(DMN)模型、Eward 语言语言(DRL)规则、指导决策表和其他规则授权资产来开发决策服务。

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. 使用 DMN 模型设计决策服务

作为业务分析员或业务规则开发人员，您可以使用 Decision Model 和 Notation(DMN)以图形化方式建模决策服务。DMN 决策模型的决策要求由决定在一个或多个决策要求图(DRD)中描述的决定。DRD 可以代表 DMN 模型的一部分或所有 DRG。DRD 从开始到尾追踪业务决策，每个决策节点都使用在 DMN 框式表达式（如决策表）中定义的逻辑。

Red Hat Process Automation Manager 为 DMN 1.1、1.2、1.3 和 1.4 模型提供运行时支持，符合级别 3。您可以直接在 Business Central 中设计 DMN 模型，或使用 VS Code 中的 Red Hat Process Automation Manager DMN 模型程序，或将现有 DMN 模型导入到用于部署和执行的 Red Hat Process Automation Manager 项目中。您导入到 Business Central 的任何 DMN 1.1 和 1.3 模型（不包含 DMN 1.3 功能），在 DMN 设计程序中打开，保存将转换为 DMN 1.2 模型。

有关 DMN 的更多信息，请参阅对象管理组(OMG) Decision Model 和 Notation 规格 。

有关带有示例 DMN 决策服务的逐步教程，*请参阅开始使用决策服务*。

第 1 章 红帽流程自动化管理器中的决策资产

Red Hat Process Automation Manager 支持一些可用于为决策服务定义业务决策的资产。每个决策资产都有不同的优势，您可以根据目标和需求，首选使用一个或多个资产的组合。

下表重点介绍红帽流程自动化管理器项目支持的主要决策资产，以帮助您决定或确认在决策服务中定义决策的最佳方法。

表 1.1. Red Hat Process Automation Manager 支持的决策资产

asset	亮点	编写工具	Documentation
决策模型和符号(DMN)型号	• 决策模型基于由对象管理组(OMG)定义的符号标准。 • 使用代表部分或所有决策要求图(DRG)的图形化决策要求图(DRG)跟踪业务决策流程 • 使用 XML 模式，允许在 DMN 兼容平台间共享 DMN 模型 • 支持 Friendly Enough Expression Language(FEEL)，在 DMN 决策表中定义决策逻辑，以及其他 DMN 框的表达式 • 可以与业务流程模型和符号(BPMN)流程模型有效集成 • 是创建全面、清晰和稳定的决策流程的最佳选择	Business Central 或其他与 DMN 兼容的编辑器	*使用 DMN 模型设计决策服务*
指导的决定表	• 是您在 Business Central 中基于 UI 的表格设计者创建的规则表 • 是电子表格决策表的向导替代选择 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值 • 支持点击策略、实时验证和其他资产不支持的其他功能 • 是以受控表格格式创建规则的最佳选择，可最小化编译错误	Business Central	*使用指导决策表设计决策服务*

asset	亮点	编写工具	Documentation
电子表格决策表	• 是 XLS 或 XLSX 表格决策表，您可以上传到 Business Central • 支持创建规则模板的模板键和值 • 是创建已在 Business Central 之外管理的决策表中规则的最佳选择 • 在上传后，对规则进行编译具有严格的语法要求	电子表格编辑器	*使用电子表格决策表设计决策服务*
指导规则	• 是您在 Business Central 中基于 UI 的规则设计者创建的单独规则 • 为可接受的输入提供字段和选项 • 是以受控格式创建单一规则的最佳选择，以最小化编译错误	Business Central	*使用指导规则设计决策服务*
指导规则模板	• 是您在 Business Central 中基于 UI 的模板设计者创建的可重复使用的规则结构 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值（与此资产的目的不相同） • 是创建具有相同规则结构但具有不同定义字段值的很多规则的最佳	Business Central	*使用指导规则模板设计决策服务*
DRL 规则	• 是直接在 **.drl** 文本文件中定义的单独规则 • 提供定义规则和其他规则行为的最大的灵活性 • 可以在特定的独立环境中创建，并与 Red Hat Process Automation Manager 集成 • 是创建需要高级 DRL 选项的规则的最佳选择 • 对规则进行正确编译的规则具有严格的语法要求	业务中心或集成开发环境(IDE)	*使用 DRL 规则设计决策服务*

asset	亮点	编写工具	Documentation
预测模型标记语言(PMML)型号	• 是基于数据 Mining Group(DMG)定义的标记标准预测数据分析模型。 • 使用允许 PMML 模式在 PMML 兼容的平台之间共享的 XML 模式 • 支持 Regression、scorecard、Tree、Mining 和其他模型类型 • 可以包含在独立 Red Hat Process Automation Manager 项目中，或者导入到 Business Central 中的项目 • 在 Red Hat Process Automation Manager 中，将预测数据合并为决策服务的最佳选择	PMML 或 XML 编辑器	*使用 PMML 模型设计决策服务*

在定义决策时，您还可以考虑将红帽构建的 Kogito 构建用于云原生决策服务。有关使用红帽构建的 Kogito 微服务的更多信息，请参阅 Red Hat *Process Automation Manager 中的 Red Hat build of Kogito* 。

第 2 章 RED HAT PROCESS AUTOMATION MANAGER 192.168.1.0/24 和 DMN 模型器

Red Hat Process Automation Manager 提供以下扩展或应用程序，您可以使用图形模型设计流程模型和决策模型以及使用图形模型(DMN)决策模型的决策模型。

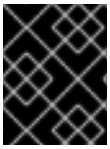
- **Business Central** ：允许您在相关的嵌入式设计器中查看并设计 过程、DMN 模型和测试场景文件。
  要使用 Business Central，您可以设置一个包含 Business Central 的开发环境，以设计新规则和流程，以及 KIE 服务器来执行和测试创建的规则和流程。

- **Red Hat Process Automation Manager VS Code 扩展** ：允许您在 Visual Studio Code (VS Code)中查看并设计 the 型号、DMN 模型和测试场景文件。VS Code 扩展需要 VS Code 1.46.0 或更高版本。
  要安装 Red Hat Process Automation Manager VS Code 扩展，请在 VS Code 中选择 **Extensions** 菜单选项，并搜索并安装 **Red Hat Business Automation Bundle**扩展。

- **Standalone the and DMN 编辑器** ：可让您查看并设计嵌入在 web 应用程序中的 Ice 和 DMN 模型。要下载必要的文件，您可以使用 NPM registry 中的 NPM 工件，或直接从 https://<YOUR_PAGE>/ **dmn/index.js 以及 https://<YOUR_PAGE>/dmn / index.js** 的 DMN 独立编辑器库下载 JavaScript 文件。

2.1. 安装 RED HAT PROCESS AUTOMATION MANAGER VS CODE 扩展捆绑包

Red Hat Process Automation Manager 提供了一个 **Red Hat Business Automation Bundle**VS Code 扩展，可让您直接在 VS Code 中设计决策模型和表示法(DMN)决策模型、业务流程模型和表示法(DSLN) 2.0 业务，以及测试场景。VS Code 是开发新商业应用程序的首选集成开发环境(IDE)。只有在需要时，Red Hat Process Automation Manager 还会为 DMN 或 Thorntail 支持提供单独的 **DMN Editor 和 Tailoring Editor** VS Code 扩展。



重要

VS Code 中的编辑器与 Business Central 中的编辑器部分兼容，VS Code 不支持几个 Business Central 功能。

先决条件

- 已安装 VS Code 的最新稳定版本。

流程

1. 在 VS Code IDE 中，选择 **Extensions** 菜单选项，并搜索 **Red Hat Business Automation Bundle** for DMN、Tist 和 test 场景文件支持。
   对于 DMN 或 the 文件支持，您也可以搜索单独的 **DMN Editor** 或 Tailoring **Editor** 扩展。

2. 当 **Red Hat Business Automation Bundle**扩展出现在 VS Code 中时，选择它并单击 **Install**。

3. 对于最佳 VS Code 编辑器行为，请在完成扩展安装后，重新加载或关闭和重新启动 VS Code 实例。

安装 VS Code 扩展捆绑包后，您在 VS Code 中打开或创建的 **.dmn**、**.bpmn** 或 **.bpmn2** 文件将自动显示为图形模型。此外，您打开或创建的 **.scesim** 文件将自动显示为表格测试场景模型，用于测试您的决策功能。

如果 DMN、Trant 或 test 场景模型仅打开 DMN、Tist 或 test 场景文件的 XML 源，并显示错误消息，检查报告的错误和模型文件，以确保正确定义了所有元素。

注意

对于新的 DMN 或 IaaS 模型，您还可以在 Web 浏览器中输入 **dmn.new** 或 **bpmn.new**，以便在在线模型中设计 DMN 或 192.168.1.0/24 模型。完成创建模型后，您可以在在线模型页面中单击 **Download**，将 DMN 或 the 文件导入到 VS Code 中的 Red Hat Process Automation Manager 项目中。

2.2. 配置 RED HAT PROCESS AUTOMATION MANAGER 独立编辑器

Red Hat Process Automation Manager 提供独立编辑器，这些编辑器在自包含的库中分发，为每个编辑器提供一个 all-in-one JavaScript 文件。JavaScript 文件使用全面的 API 来设置和控制编辑器。

您可以使用以下方法安装独立编辑器：

- 手动下载每个 JavaScript 文件
- 使用 NPM 软件包

流程

1. 使用以下方法之一安装独立编辑器：
   手动下载每个 JavaScript 文件 ：对于此方法，请按照以下步骤执行：

   a. 下载 JavaScript 文件。

   b. 将下载的 Javascript 文件添加到托管应用程序中。

   c. 将以下 &lt **;script&** gt; 标签添加到 HTML 页面中：

   DMN 编辑器的 HTML 页面的脚本标签

   ```
   <script src="https://<YOUR_PAGE>/dmn/index.js"></script>
   ```

   您的 HTML 页面的脚本标签

   ```
   <script src="https://<YOUR_PAGE>/bpmn/index.js"></script>
   ```

   使用 NPM 软件包 ：对于这个方法，请按照以下步骤执行：

   a. 在您的 **package.json** 文件中添加 NPM 软件包：

   添加 NPM 软件包

   ```
   npm install @kie-tools/kie-editors-standalone
   ```

   b. 将每个编辑器库导入到您的 **TypeScript** 文件中：

   导入每个编辑器

```
import * as DmnEditor from "@kie-tools/kie-editors-standalone/dist/dmn"
import * as BpmnEditor from "@kie-tools/kie-editors-standalone/dist/bpmn"
```

2. 安装独立编辑器后，使用提供的编辑器 API 打开所需的编辑器，如下例所示来打开 DMN 编辑器。每个编辑器的 API 相同。

打开 DMN 独立编辑器

```
const editor = DmnEditor.open({
  container: document.getElementById("dmn-editor-container"),
  initialContent: Promise.resolve(""),
  readOnly: false,
  origin: "",
  resources: new Map([
    [
      "MyIncludedModel.dmn",
      {
        contentType: "text",
        content: Promise.resolve("")
      }
    ]
  ])
});
```

在编辑器 API 中使用以下参数：

表 2.1. 参数示例

参数	描述
container	附加编辑器的 HTML 元素。
initialContent	承诺 DMN 模型内容。这个参数可以为空，如下例所示： • **Promise.resolve("")** • **Promise.resolve("<DIAGRAM_CONTENT_DIRECTLY_HERE>")** • **fetch("MyDmnModel.dmn").then(content ⇒ content.text())**
ReadOnly （可选）	允许您在编辑器中允许更改。设置为 **false** （默认）以允许编辑器中的只读模式的内容编辑和 **true**。
原始 （可选）	存储库的来源。默认值为 **window.location.origin**。
资源 （可选）	编辑器的资源映射。例如，此参数用于为 DMN 编辑器提供包含的型号，或为 the 编辑器提供工作项目定义。映射中的每个条目都包含一个资源名称，以及一个由 **content-type** (**text** 或 **binary**) 和 **content** (与 **initialContent** 参数类似) 组成的对象。

返回的对象包含操作编辑器所需的方法。

表 2.2. 返回的对象方法

方法	描述
getContent(): Promise<string>	返回包含编辑器内容的承诺。
setContent(path: string, content: string): void	设置编辑器的内容。
getPreview(): Promise<string>	返回包含当前图的 SVG 字符串的承诺。
subscribeToContentChanges（callback: (isDirty: boolean) void void: (isDirty: boolean) void void	设置在编辑器中内容更改时调用的回调，并返回相同的回调来取消订阅。
unsubscribeToContentChanges (callback: (isDirty: boolean) rhcs void): void	当内容在编辑器中更改时，取消订阅传递的回调。
markAsSaved(): void	重置编辑器状态，表示保存编辑器中的内容。另外，它还激活与内容更改相关的订阅回调。
undo(): void	撤销编辑器中的最后更改。另外，它还激活与内容更改相关的订阅回调。
redo(): void	在编辑器中更改最后的未撤消更改。另外，它还激活与内容更改相关的订阅回调。
close(): void	关闭编辑器。
getElementPosition (selector: string): Promise<Rect>	提供当元素位于 canvas 或视频组件内时扩展标准查询选择器的替代选择。**选择器** 参数必须遵循 **<PROVIDER>::: <SELECT>** 格式，如 **Canvas:::MySquare** 或 **Video:::PresenterHand**。此方法返回一个代表元素位置的 **Rect**。
envelopeApi: MessageBusClientApi<KogitoEditorEnvelopeApi>	这是一个高级编辑器 API。有关高级编辑器 API 的更多信息，请参阅 MessageBusClientApi 和 KogitoEditorEnvelopeApi。

第 3 章 使用 MAVEN 创建并执行 DMN 和 SVIRT 模型

您可以使用 Maven 架构类型，使用 Red Hat Process Automation Manager VS Code 扩展而不是 Business Central 在 VS Code 中开发 DMN 和 Tailoring 模型。然后，您可以根据需要将 archetypes 与 Red Hat Process Automation Manager 决策和处理服务集成。此开发 DMN 和 the 型号的方法有助于使用 Red Hat Process Automation Manager VS Code 扩展构建新的商业应用程序。

流程

1. 在命令终端中，导航到您要存储新的 Red Hat Process Automation Manager 项目的本地文件夹。

2. 输入以下命令使用 Maven archtype 在定义的文件夹中生成项目：

   使用 Maven 架构类型生成项目

   ```
   mvn archetype:generate \
       -DarchetypeGroupId=org.kie \
       -DarchetypeArtifactId=kie-kjar-archetype \
       -DarchetypeVersion=7.67.0.Final-redhat-00024
   ```

   此命令生成一个具有所需依赖项的 Maven 项目，并生成所需的目录和文件来构建您的业务应用程序。您可以在开发项目时使用 Git 版本控制系统（推荐）。

   如果要在同一目录中生成多个项目，请在上一个命令中添加 **-DgroupId=<groupid> -DartifactId=<artifactId>** 来指定生成的业务应用程序的 **artifactId** 和 **groupId**。

3. 在 VS Code IDE 中，单击 **File**，选择 **Open Folder**，再导航到使用上一命令生成的文件夹。

4. 在创建第一个资产前，为您的新应用程序设置软件包，如 **org.kie.businessapp**，并在以下路径中创建对应的目录：

   - **PROJECT_HOME/src/main/java**
   - **PROJECT_HOME/src/main/resources**
   - **PROJECT_HOME/src/test/resources**

   例如，您可以为 **org.kie.businessapp** 软件包创建 **PROJECT_HOME/src/main/java/org/kie/businessapp**。

5. 使用 VS Code 为您的新应用程序创建资产。您可以使用以下方法创建 Red Hat Process Automation Manager VS Code 扩展支持的资产：

   - 要创建业务进程，请在 **PROJECT_HOME/src/main/resources/org/kie/businessapp** 目录中创建一个包含 **.bpmn** 或 **.bpmn2** 的新文件，如 **Process.bpmn**。
   - 要创建 DMN 模型，请在 **PROJECT_HOME/src/main/resources/org/kie/businessapp** 目录中创建一个带有 **.dmn** 的新文件，如 **AgeDecision.dmn**。
   - 要创建测试场景模拟模型，请在 **PROJECT_HOME/src/test/resources/org/kie/businessapp** 目录中创建一个 **.scesim** 的新文件，如 **TestAgeScenario.scesim**。

6. 在 Maven 架构类型中创建资产后，在命令行中进入项目的根目录(tains **pom.xml**)，并运行以下命令来构建项目的了解 JAR (KJAR)：

```
mvn clean install
```

如果构建失败，请解决命令行错误消息中描述的任何问题，并尝试验证项目，直到构建成功为止。但是，如果构建成功，您可以在 **PROJECT_HOME/target** 目录中找到新应用程序的工件。

注意

使用 **mvn clean install** 命令通常会在开发期间的每一主要更改后验证您的项目。

您可以使用 REST API 在正在运行的 KIE 服务器上部署商业应用程序生成的知识 JAR (KJAR)。有关使用 REST API 的更多信息，*请参阅使用 KIE API 与 Red Hat Process Automation Manager 交互* 。

第 4 章 决策模型和符号(DMN)

决策模型和符号(DMN)是对象管理组(OMG)制定的标准，用于描述和建模操作决策。DMN 定义了一个 XML 模式，使 DMN 模型可以在 DMN 兼容平台和机构中共享，以便业务分析者和业务规则开发人员能够合作设计和实施 DMN 决策服务。DMN 标准与 相似，可与业务流程建模和符号(BPMN)标准一起使用，以设计和建模业务流程。

有关 DMN 后台和应用的详情，请查看 OMG Decision Model 和 Notation 规格 。

4.1. DMN 一致性级别

DMN 规格在软件实施中定义了三个增量级别。在一个级别声明合规性的产品还必须符合前面的级别。例如，合规级别 3 的实施还必须在级别 1 和 2 中包含支持的组件。有关每个合规级别的正式定义，请参阅 OMG Decision Model 和 Notation 规格 。

以下列表总结了三个 DMN 合规级别：

一致级别 1

DMN 一致性级别 1 实现支持决策要求图(DRDs)、决策逻辑和决策表，但决策模型不可执行。任何语言都可用于定义表达式，包括自然、非结构化语言。

一致级别 2

DMN 一致性级别 2 的实现包括合规级别 1 中的要求，并支持 Simplified Friendly Enough Expression Language(S-FEEL)表达式和完全可执行决策模型。

合规级别 3

DMN 一致性级别 3 的实现包括一致性级别 1 和 2 中的要求，并支持 Friendly Enough Expression Language(FEEL)表达式、完整的开箱即用表达式以及完全可执行决策模型。

Red Hat Process Automation Manager 为 DMN 1.1、1.2、1.3 和 1.4 模型提供运行时支持，符合级别 3。您可以直接在 Business Central 中设计 DMN 模型，或使用 VS Code 中的 Red Hat Process Automation Manager DMN 模型程序，或将现有 DMN 模型导入到用于部署和执行的 Red Hat Process Automation Manager 项目中。您导入到 Business Central 的任何 DMN 1.1 和 1.3 模型（不包含 DMN 1.3 功能），在 DMN 设计程序中打开，保存将转换为 DMN 1.2 模型。

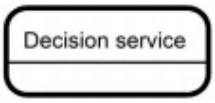
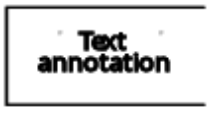
4.2. DMN 决策要求图(DRD)组件

决策要求图(DRD)是 DMN 模型的可视化表示。DRD 可以代表 DMN 模型的部分或所有决定要求图(DRG)。DDRD 使用决策节点、商业知识模型、业务知识来源、输入数据和决策服务来追踪业务决策。

下表总结了 DRD 中的组件：

表 4.1. DRD 组件

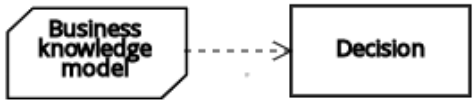
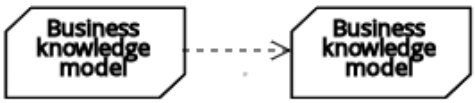
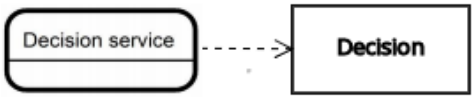
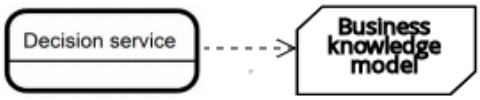
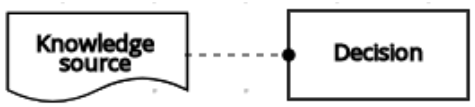
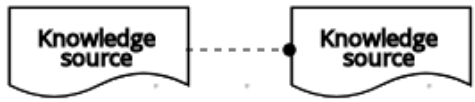
组件		描述	表示法
元素	决策	一个或多个输入元素的节点根据定义的决策逻辑来确定输出。	Decision

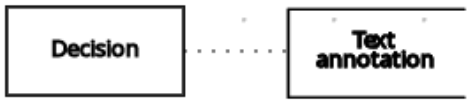
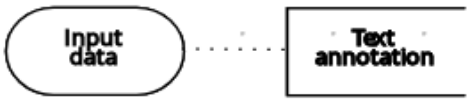
组件	描述		表示法
	商业知识模型	通过一个或多个决策元素重复使用功能。具有相同逻辑的决策但依赖于不同的子项数据或子子数据，使用商业知识模型来确定要遵循哪些流程。	
	知识源	监管决策或商业知识模型的外部机构、记录、提交或策略。知识源是指参考实际因素而不是可执行的业务规则。	
	输入数据	决策节点或业务知识模型中使用的信息。输入数据通常包括与业务相关的业务级别概念或对象，如在客制策略中使用的 loan Applicationlicant 数据。	
	决策服务	包含一组可用于调用的服务已发布的一组可重用决策的顶级决策。决策服务可从外部应用程序或 BPMN 业务流程中调用。	
要求连接器	信息要求	从输入数据节点或决定节点与需要信息的另一个决定节点的连接。	
	知识要求	从业务知识模型与决策节点连接，或与调用决策逻辑的其他业务知识模型连接。	
	颁发机构要求	从输入数据节点或决定节点到依赖知识源或从知识源到决策节点、业务知识模型或其他知识源的连接。	
工件	文本注解	请说明与输入数据节点、决策节点、业务知识模型或知识来源相关的说明。	
	关联	从输入数据节点、决策节点、商业知识模型或知识源到文本注解的连接。	

下表总结了 DRD 元素间的允许连接器：

表 4.2. DRD 连接器规则

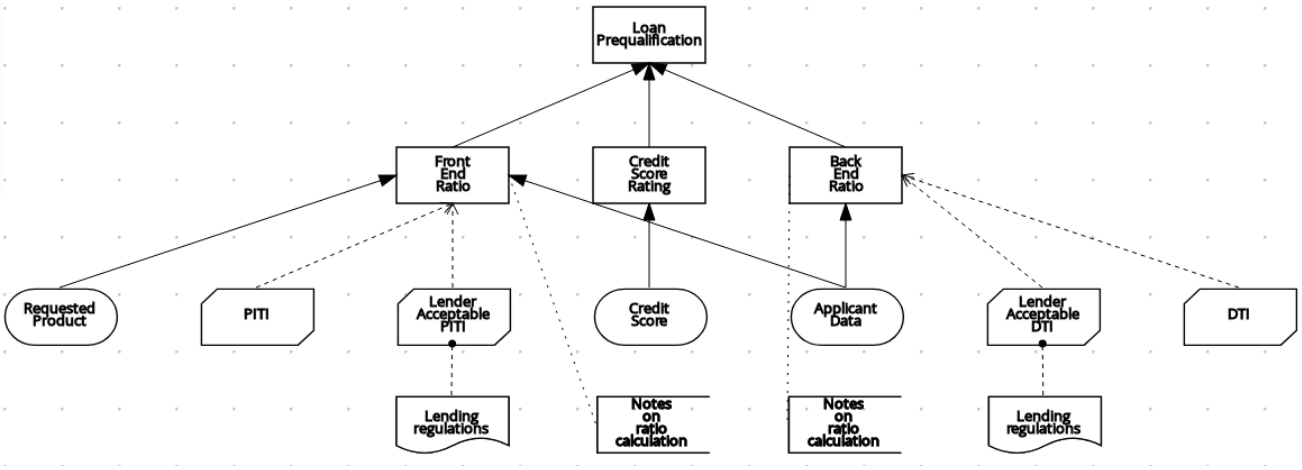
从开始	连接到	连接类型	示例

从开始	连接到	连接类型	示例
决策	决策	信息要求	
商业知识模型	决策	知识要求	
	商业知识模型		
决策服务	决策	知识要求	
	商业知识模型		
输入数据	决策	信息要求	
	知识源	颁发机构要求	
知识源	决策	颁发机构要求	
	商业知识模型		
	知识源		

从开始	连接到	连接类型	示例
决策	文本注解	关联	
商业知识模型			
知识源			
输入数据			

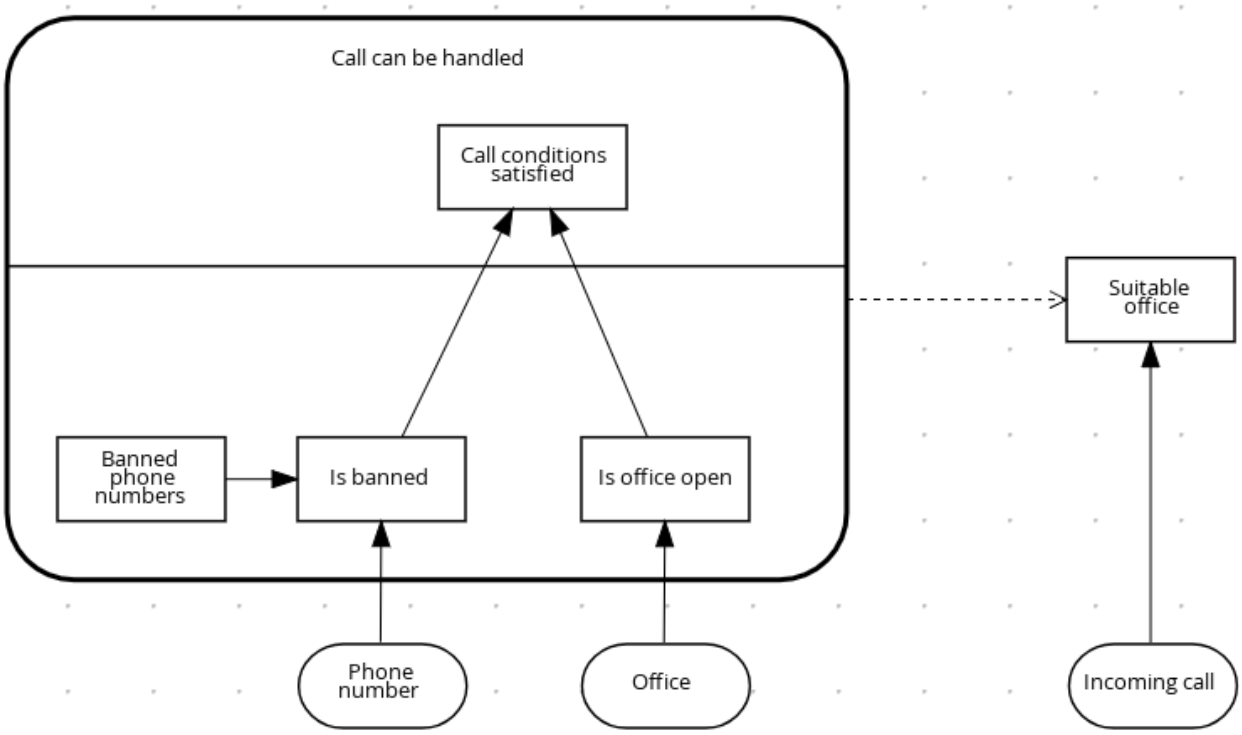
以下示例 DRD 演示了其中的一些 DMN 组件：

图 4.1. DRD 示例： Loan prequalification



以下示例 DRD 演示了作为可重用决策服务一部分的 DMN 组件：

图 4.2. DRD 示例：作为决策服务的电话调用处理



在 DMN 决策服务节点中，底部片段中的决策节点纳入了来自决策服务之外的输入数据，以达到决策服务节点的主要网段中最终决策。然后，在 DMN 模型的后续决策或商业知识要求中实施来自决策的顶级决策。您可以在其他 DMN 模式下重复使用 DMN 决策服务，以应用具有不同输入数据和不同传出连接的相同决策逻辑。

4.3. FEEL 中的规则表达式

友好的表达式语言(FEEL)是对象管理组(OMG)DMN 规范定义的表达式语言。FEEL 表达式在 DMN 模型中定义决策的逻辑。FEEL 旨在通过为决策模型构造分配语义来促进决策建模和执行。FEEL 表达式在决策要求图(DRDs)中占用表单元单元格，用于决策节点和商业知识模型。

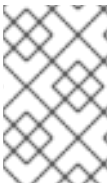
有关 DMN 中的 FEEL 的更多信息，请参阅 OMG Decision Model 和 Notation 规格 。

4.3.1. FEEL 中的数据类型

友好的表达式语言(FEEL)支持以下数据类型：

- 数字
- 字符串
- 布尔值
- 日期
- Time
- 日期和时间
- 天数和时间持续时间
- 多年和月的时间

- Functions
- 上下文
- 范围（或间隔）
- 列表



注意

DMN 规格目前不提供将变量声明为 **功能、上下文、范围或列表的** 显式方法，但 Red Hat Process Automation Manager 会扩展 DMN 内置类型来支持这些类型的变量。

以下列表描述了每种数据类型：

数字

FEEL 中的数字基于 IEEE 754-2008 年 12 月 128 格式，有 34 个数字的精度。在内部，数字在 Java 中表示为 MathContext DECIMAL128。https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.htmlFEEL 只支持一个数字数据类型，因此同一类型用于表示整数和浮点号。

FEEL 编号使用点(.)作为十进制分隔符。FEEL 不支持 -INF、+INF 或 NaN。FEEL 使用 null 来代表无效的数字。

Red Hat Process Automation Manager 扩展 DMN 规格并支持额外的数字表示法：

- **Scientific： 您可以使用带有后缀 e<exp> 或 E<exp> 的科学标记。例如，1.2e3 与编写表达式 1.2*10**3 相同，但是一个字面上的值，而不是表达式。**
- **十六进制 ： 您可以使用带有前缀 0x 的十六进制数字。例如： 0xff 与十进制数字 255 相同。支持大写字母和小写字母。例如： 0XFF 与 0xff 相同。**
- **类型后缀： 您可以使用类型后缀 f、F、d、D、l 和 L。这些后缀会被忽略。**

字符串

FEEL 中的字符串是任意字符序列，用双引号括起。

示例

```
"John Doe"
```

布尔值

FEEL 使用三值布尔逻辑，因此布尔值逻辑表达式可以有 true、false 或 null 的值。

日期

FEEL 不支持日期字面量，但您可以使用内置的 date（） 函数来构造日期值。FEEL 中的日期字符串遵循 XML 架构部分 2: Datatypes 文档中定义的格式。格式为 "YYYY-MM-DD"，其中 YYYY 是包含四位数字的年份，MM 是带有两个数字的月份数，DD 是当天的数目。

例如：

```
date( "2017-06-23" )
```

日期对象的时间等于 "00:00:00"，即午夜。日期被视为本地，不带时区。

Time

FEEL 不支持时间字面量，但您可以使用内置的 time（） 函数来构造时间值。FEEL 中的时间字符串遵循 XML 架构部分 2: Datatypes 文档中定义的格式。格式为 " hh:mm:ss[.uu][(+-)h:mm]"，h 是一天的小时数（从 00 到 23），mm 是小时内分钟，ss 是分钟的几秒。（可选）字符串可在第二之内定义毫秒(uuu)的数量，并包含正数(+)或从 UTC 时间起的负值(-)误差来定义其时区。您可以使用字母 z 来表示 UTC 时间，这与偏移 -00:00 相同。如果没有定义偏移，时间将被视为 local。

示例：

```
time( "04:25:12" )
time( "14:10:00+02:00" )
time( "22:35:40.345-05:00" )
time( "15:00:30z" )
```

定义偏移或时区的时间值不能与未定义偏移或时区的本地时间进行比较。

日期和时间

FEEL 不支持日期和时间字面量，但您可以使用内置 的日期和时间（date 和 time（） 函数）来构造日期和时间值。FEEL 中的日期和时间字符串遵循 XML 架构部分 2: Datatypes 文档中定义的格式。格式为 "< date>T<time>"，其中 <date > 和 <time > 遵循预定义的 XML 模式格式，由 T 添加。

示例：

```
date and time( "2017-10-22T23:59:00" )
date and time( "2017-06-13T14:10:00+02:00" )
date and time( "2017-02-05T22:35:40.345-05:00" )
date and time( "2017-06-13T15:00:30z" )
```

定义偏移或时区的日期和时间值不能与未定义偏移或时区的本地日期和时间值进行比较。

重要

如果您的 DMN 规格的实现不支持 XML 架构中的空格，请使用关键字 dateTime 作为 日期和时间的同步。

天数和时间持续时间

FEEL 不支持天和持续时间字面量，但您可以使用内置的 duration（） 函数来构造天数和持续时间值。FEEL 中的天数和持续时间字符串遵循 XML 架构第 2 部分定义 的格式，但仅限于天、小时和秒。不支持数月和年时间。

示例：

```
duration( "P1DT23H12M30S" )
duration( "P23D" )
duration( "PT12H" )
duration( "PT35M" )
```

重要

如果您的 DMN 规格的实现不支持 XML 架构中的空格，请使用关键字 dayTimeDuration 作为 天数 和持续时间的同步。

多年和月的时间

FEEL 不支持多年和月的字面量，但您可以使用内置的 duration（） 函数来构造天数和持续时间值。FEEL 中的年和月字符串遵循 XML 架构第 2 部分定义的格式：数据类型 文档，但仅限于数年和月。不支持天、小时、分钟或秒。

示例：

```
duration( "P3Y5M" )
duration( "P2Y" )
duration( "P10M" )
duration( "P25M" )
```

重要

如果您的 DMN 规格的实现在 XML 模式中不支持空格，使用关键字 yearMonthDuration 作为年份 和月的时间。

Functions

FEEL 具有 功能 文字（或匿名功能），可用于创建功能。DMN 规范目前不提供将变量声明为 功能 的一种明确方法，但 Red Hat Process Automation Manager 会将 DMN 内置类型扩展为支持功能变量。

例如：

```
function(a, b) a + b
```

在这个示例中，FEEL 表达式会创建一个功能，添加参数 a 和 b，并返回结果。

上下文

FEEL 具有 上下文 文字，可用于创建上下文。FEEL 中的 上下文是 键值对列表，类似于 Java 等语言映射。DMN 规格目前不提供将变量声明为 上下文 的方法，但 Red Hat Process Automation Manager 会将 DMN 内置类型扩展为支持上下文变量。

例如：

```
{ x : 5, y : 3 }
```

在本例中，表达式会创建一个包含两个条目（ x 和 y ）的上下文，代表 chart 中的协调。

在 DMN 1.2 中，创建上下文的另一个方法是创建包含键列表作为属性的项定义，然后将变量声明为具有项目定义类型。

Red Hat Process Automation Manager DMN API 支持 DMN ItemDefinition structural 类型，其以两种方式表示：

- 用户定义的 Java 类型：Must 是定义属性的有效 JavaBeans 对象，为 DMN ItemDefinition 中的每个组件的 getters。如果需要，您也可以将 @FEELProperty 注释用于代表一个组件名称（这会导致无效的 Java 标识符）的 getter。

- java.util.Map 接口：映射需要定义适当的条目，其键对应于 DMN ItemDefinition 中的组件名称。

范围（或间隔）

FEEL 有 范围 文字，可用于创建范围或间隔。FEEL 中 的范围是 一个定义较低和上限的值，可在其中打开或关闭它们。DMN 规格目前不提供将变量声明为 范围 的方法，但 Red Hat Process Automation Manager 会将 DMN 内置类型扩展为支持范围变量。

范围语法以以下格式定义：

```
range          := interval_start endpoint '..' endpoint interval_end
interval_start := open_start | closed_start
open_start     := '(' | ']'
closed_start   := '['
interval_end   := open_end | closed_end
open_end       := ')' | '['
closed_end     := ']'
endpoint       := expression
```

端点的表达式必须返回可比较的值，并且较低绑定端点必须小于上限的端点。

例如，以下字面表达式定义了 1 到 10 之间的间隔，包括边界（两个端点上的关闭间隔）：

```
[ 1 .. 10 ]
```

以下字面表达式定义了 1 小时和 12 小时之间的间隔，其中包括下限（关闭间隔），但排除了上限（打开的时间间隔）：

```
[ duration("PT1H") .. duration("PT12H") )
```

您可以使用决策表中的范围来测试值范围，或者在简单字面表达式中使用范围。例如，如果变量

x 的值介于 0 到 100 之间，则以下字面表达式返回 true ：

```
x in [ 1 .. 100 ]
```

列表

FEEL 具有 列表 文字，可用于创建项目列表。FEEL 中的 列表 由以方括号括起的值列表来表示。DMN 规格目前不提供将变量声明为 列表 （）的显式方式，但 Red Hat Process Automation Manager 会将 DMN 内置类型扩展为支持列表变量。

例如：

```
[ 2, 3, 4, 5 ]
```

FEEL 中的所有列表都包含同一类型的元素，且不可变。列表中的元素可以通过索引来访问，其中第一个元素为 1。负索引可以访问从列表末尾开始的元素，以便 -1 是最后一个元素。

例如，以下表达式返回列表 x 的第二个元素：

```
x[2]
```

以下表达式返回列表 x 的 second-to-last 元素：

```
x[-2]
```

列表中的元素也可以按函数 数计算，后者将元素列表用作参数。

例如，以下表达式返回 4 ：

```
count([ 2, 3, 4, 5 ])
```

4.3.2. FEEL 中的内置功能

为促进与其他平台和系统间的互操作性，Friendly Enoendly Enough Expression Language(FEEL) 包括了一个内置功能库。内置的 FEEL 功能在 Drools Decision Model 和 Notation(DMN)引擎中实施，以便您可以使用 DMN 决策服务中的功能。

以下小节描述了每个内置 FEEL 功能，它的格式是 *NAME(PARAMETERS)*。有关 DMN 中的 FEEL 功能的更多信息，请参阅 OMG Decision Model 和 Notation 规格。

4.3.2.1. 转换功能

以下功能支持在不同类型值之间进行转换。其中一些功能使用特定的字符串格式，如下例所示：

- 日期字符串 ：遵循 XML 架构部分 2: 数据类型 文档中定义的格式，如 2020-06-01
- 时间字符串 ：遵循以下格式之一：
  - XML 架构第 2 部分定义的格式：数据类型 文档，如 23:59:00z
  - 由 ISO 8601 定义的本地时间格式，后跟 @ 和 IANA Timezone，如 00:01:00@Etc/UTC
- 日期字符串 ：遵循 日期 字符串的格式，后跟 T 和 时间字符串，如 2012-12-25T11:00:00Z
- duration string ：遵循 XQuery 1.0 和 XPath 2.0 Data Model 中定义 的天数和时间 持续时间，如 P1Y2M

日期（从 中）- *使用日期*

从 转换为 日期 值。

表 4.3. 参数

参数	类型	格式
from	字符串	日期字符串

示例

```
date( "2012-12-25" ) - date( "2012-12-24" ) = duration( "P1D" )
```

日期（从 中）- *使用日期和时间*

从 转换为 日期 值，并将时间组件设置为 null。

表 4.4. 参数

参数	类型
from	日期和时间

示例

```
date(date and time( "2012-12-25T11:00:00Z" )) = date( "2012-12-25" )
```

日期（年、月、日）

生成指定年份、月份和日期值 的日期。

表 4.5. 参数

参数	类型
year	number
month	number
day	number

示例

```
date( 2012, 12, 25 ) = date( "2012-12-25" )
```

日期和时间（日期、时间 ）

从指定 日期 生成日期和时间，并忽略任何时间和指定时间。

表 4.6. 参数

参数	类型
date	**日期和时间**
time	**time**

示例

```
date and time ( "2012-12-24T23:59:00" ) = date and time(date( "2012-12-24" ), time( "23:59:00" ))
```

日期和时间（从 ）

从指定字符串生成 日期和时间。

表 4.7. 参数

参数	类型	格式
from	**字符串**	**日期字符串**

示例

```
date and time( "2012-12-24T23:59:00" ) + duration( "PT1M" ) = date and time( "2012-12-25T00:00:00" )
```

time(from)

从指定字符串生成一个 时间。

表 4.8. 参数

参数	类型	格式
from	字符串	时间字符串

示例

```
time( "23:59:00z" ) + duration( "PT2M" ) = time( "00:01:00@Etc/UTC" )
```

time(from)

从指定参数生成一个 时间，并忽略任何日期组件。

表 4.9. 参数

参数	类型
from	时间 或 日期和时间

示例

```
time(date and time( "2012-12-25T11:00:00Z" )) = time( "11:00:00Z" )
```

时间（小时、分钟、第二个、偏移？）

从指定小时、分钟和第二个组件值生成时间。

表 4.10. 参数

参数	类型
hour	number
minute	number
second	number
offset （可选）	天和时间持续时间 或 null

示例

```
time( "23:59:00z" ) = time(23, 59, 0, duration( "PT0H" ))
```

（来自、分组分隔符、十进制分隔符）

使用指定的分隔符 从 转换为 数字。

表 4.11. 参数

参数	类型
from	代表有效数字的字符串
分组分隔符	空格（）、comma（、）、period(.)或 null
十进制分隔符	与 分组分隔符 相同，但值无法匹配

示例

```
number( "1 000,0", " ", "," ) = number( "1,000.0", ",", "." )
```

string(from)

提供指定参数的字符串表示。

表 4.12. 参数

参数	类型
from	非 null 值

例子

```
string( 1.1 ) = "1.1"
string( null ) = null
```

duration(from)

从 天和持续时间 值转换为 数年和月内的时间 值。

表 4.13. 参数

参数	类型	格式
from	**字符串**	**duration string**

例子

```
date and time( "2012-12-24T23:59:00" ) - date and time( "2012-12-22T03:45:00" ) = duration(
"P2DT20H14M" )
duration( "P2Y2M" ) = duration( "P26M" )
```

多年和月度（从 到 ）

计算两个指定参数之间的 年和月时间。

表 4.14. 参数

参数	类型
from	日期和时间
至	日期和时间

示例

```
years and months duration( date( "2011-12-22" ), date( "2013-08-24" ) ) = duration( "P1Y8M" )
```

4.3.2.2. 布尔值功能

以下功能支持布尔值操作。

not(negand)

执行 negand 运算对象的逻辑命名。

表 4.15. 参数

参数	类型
negand	布尔值

例子

-

```
not( true ) = false
not( null ) = null
```

4.3.2.3. 字符串功能

以下功能支持字符串操作。

注意

在 FEEL 中，Unicode 字符根据其代码点进行计算。

substring（字符串、开始位置、长度？）

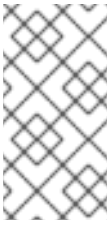
返回指定长度开始位置的子字符串。第一个字符位于位置值 1。

表 4.16. 参数

参数	类型
字符串	字符串
开始位置	number
长度 （可选）	number

例子

```
substring( "testing",3 ) = "sting"
substring( "testing",3,3 ) = "sti"
substring( "testing", -2, 1 ) = "n"
substring( "\U01F40Eab", 2 ) = "ab"
```



注意

在 FEEL 中，字符串字面上的 "\U01F40Eab" 是 sHistoryLimitab 字符串（horse 符号后跟 和 b）。

string length(string)

计算指定字符串的长度。

表 4.17. 参数

参数	类型
字符串	字符串

例子

```
string length( "tes" ) = 3
string length( "\U01F40Eab" ) = 3
```

大写（字符串 ）

生成指定字符串的大写版本。

表 4.18. 参数

参数	类型
字符串	字符串

示例

```
upper case( "aBc4" ) = "ABC4"
```

小写（字符串 ）

生成指定字符串的小写版本。

表 4.19. 参数

参数	类型
字符串	字符串

示例

```
lower case( "aBc4" ) = "abc4"
```

子字符串前面（字符串，匹配 ）

计算匹配项前的子字符串。

表 4.20. 参数

参数	类型
字符串	字符串
匹配	字符串

例子

```
substring before( "testing", "ing" ) = "test"
substring before( "testing", "xyz" ) = ""
```

子字符串之后（字符串，匹配 ）

计算匹配项后的子字符串。

表 4.21. 参数

参数	类型
字符串	字符串
匹配	字符串

例子

```
substring after( "testing", "test" ) = "ing"
substring after( "", "a" ) = ""
```

replace（输入、模式、替换、标志？）

计算正则表达式替换。

表 4.22. 参数

参数	类型
输入	字符串
pattern	字符串
替换	字符串
标记 （可选）	字符串

注意

此函数使用 XQuery 1.0 和 XPath 2.0 Functions 和 Operator 中定义的正则表达式参数。

示例

> replace("abcd", "(ab)|(a)", "[1=$1][2=$2]") = "[1=ab][2=]cd"

contains（字符串，匹配）

如果字符串包含匹配项，则返回 true。

表 4.23. 参数

参数	类型
字符串	字符串
匹配	字符串

示例

> contains("testing", "to") = false

以 （字符串）开头，匹配 .

如果字符串以匹配项开头，则返回 true

表 4.24. 参数

参数	类型
字符串	字符串
匹配	字符串

示例

```
starts with( "testing", "te" ) = true
```

结束为 （字符串，匹配 ）

如果字符串以匹配项结尾，则返回 true。

表 4.25. 参数

参数	类型
字符串	字符串
匹配	字符串

示例

```
ends with( "testing", "g" ) = true
```

匹配（输入、模式、标志？）

如果输入与正则表达式匹配，则返回 true。

表 4.26. 参数

参数	类型
输入	字符串
pattern	字符串
标记 （可选）	字符串

注意

此函数使用 XQuery 1.0 和 XPath 2.0 Functions 和 Operator 中定义的正则表达式参数。

示例

```
matches( "teeesting", "^te*sting" ) = true
```

split(string, delimiter)

返回原始字符串的列表，并将其分成分隔符正则表达式模式。

表 4.27. 参数

参数	类型
字符串	字符串
delimiter	正则表达式模式匹配 字符串

注意

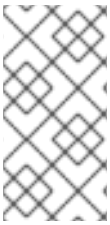
此函数使用 XQuery 1.0 和 XPath 2.0 Functions 和 Operator 中定义的正则表达式参数。

例子

```
split( "John Doe", "\\s" ) = ["John", "Doe"]
split( "a;b;c;;", ";" ) = ["a","b","c","","",""]
```

4.3.2.4. 列出功能

以下功能支持列表操作。



注意

在 FEEL 中，列表中的第一个元素的索引是 1。列表中最后一个元素的索引可以识别为 -1。

列表包含（list, 元素）

如果列表包含元素，则返回 true。

表 4.28. 参数

参数	类型
list	list
元素	任何类型，包括 null

示例

```
list contains( [1,2,3], 2 ) = true
```

count(list)

计算列表中的元素。

表 4.29. 参数

参数	类型
list	list

例子

```
count( [1,2,3] ) = 3
count( [] ) = 0
count( [1,[2,3]] ) = 2
```

min(list)

返回列表中的最低可组合元素。

表 4.30. 参数

参数	类型
list	**list**

其他签名

```
min( e1, e2, ..., eN )
```

例子

```
min( [1,2,3] ) = 1
min( 1 ) = 1
min( [1] ) = 1
```

max(list)

返回列表中的最大可组合元素。

表 4.31. 参数

参数	类型
list	**list**

其他签名

```
max( e1, e2, ..., eN )
```

例子

```
max( 1,2,3 ) = 3
max( [] ) = null
```

sum(list)

返回列表中数字的总和。

表 4.32. 参数

参数	类型
list	**数字** 元素 **列表**

其他签名

```
sum( n1, n2, ..., nN )
```

例子

```
sum( [1,2,3] ) = 6
sum( 1,2,3 ) = 6
sum( 1 ) = 1
sum( [] ) = null
```

平均（列表 ）

计算列表中元素的平均平均值（通常平均值）。

表 4.33. 参数

参数	类型
list	**数字** 元素 **列表**

其他签名

```
mean( n1, n2, ..., nN )
```

例子

```
mean( [1,2,3] ) = 2
mean( 1,2,3 ) = 2
mean( 1 ) = 1
mean( [] ) = null
```

all(list)

如果列表中的所有元素都为 **true**，则返回 **true**。

表 4.34. 参数

表 4.34. 参数

参数	类型
list	**布尔值 元素 列表**

其他签名

```
all( b1, b2, ..., bN )
```

例子

```
all( [false,null,true] ) = false
all( true ) = true
all( [true] ) = true
all( [] ) = true
all( 0 ) = null
```

any(list)

如果列表中任何元素为 true，则返回 true。

表 4.35. 参数

参数	类型
list	**布尔值 元素 列表**

其他签名

```
any( b1, b2, ..., bN )
```

例子

```
any( [false,null,true] ) = true
any( false ) = false
any( [] ) = false
any( 0 ) = null
```

子列表（列表、开始位置、长度？）

返回起始位置的子列表，限制为长度元素。

表 4.36. 参数

参数	类型
list	list
开始位置	number
长度 （可选）	number

示例

```
sublist( [4,5,6], 1, 2 ) = [4,5]
```

append(list, item)

创建附加到项目或项目的列表。

表 4.37. 参数

参数	类型
list	list

参数	类型
项	任何类型

示例

```
append( [1], 2, 3 ) = [1,2,3]
```

concatenate(list)

创建作为串联列表的结果的列表。

表 4.38. 参数

参数	类型
list	**list**

示例

```
concatenate( [1,2],[3] ) = [1,2,3]
```

插入 before(list, position, newItem)

创建一个带有 **newItem** 在指定位置插入的列表。

表 4.39. 参数

参数	类型
list	**list**
位置	**number**

参数	类型
newItem	任何类型

示例

```
insert before( [1,3],1,2 ) = [2,1,3]
```

删除（列表，位置 ）

创建一个包含指定位置排除的元素的列表。

表 4.40. 参数

参数	类型
list	**list**
位置	**number**

示例

```
remove( [1,2,3], 2 ) = [1,3]
```

反向（列出 ）

返回反向列表。

表 4.41. 参数

参数	类型
list	**list**

示例

```
reverse( [1,2,3] ) = [3,2,1]
```

（列表的索引，匹配 ）

返回与 元素匹配的索引。

参数

- 类型列表
- 与任何类型匹配

表 4.42. 参数

参数	类型
list	**list**
匹配	任何类型

示例

```
index of( [1,2,3,2],2 ) = [2,4]
```

union(list)

返回多个列表中所有元素的列表，并排除重复项。

表 4.43. 参数

参数	类型
list	list

示例

> union([1,2],[2,3]) = [1,2,3]

不同的值（列出）

返回单个列表中的元素列表，并排除重复项。

表 4.44. 参数

参数	类型
list	list

示例

> distinct values([1,2,3,2,1]) = [1,2,3]

flatten(list)

返回扁平化列表。

表 4.45. 参数

参数	类型
list	list

示例

```
flatten( [[1,2],[[3]], 4] ) = [1,2,3,4]
```

product(list)

返回列表中数字的产品。

表 4.46. 参数

参数	类型
list	**数字** 元素 **列表**

其他签名

```
product( n1, n2, ..., nN )
```

例子

```
product( [2, 3, 4] ) = 24
product( 2, 3, 4 ) = 24
```

median(list)

返回列表中数字的介质。如果元素的数量是奇数，则结果是中间元素。如果元素数量甚至是，则结果是两个中间元素的平均。

表 4.47. 参数

参数	类型
list	**数字** 元素 **列表**

其他签名

```
median( n1, n2, ..., nN )
```

例子

```
median( 8, 2, 5, 3, 4 ) = 4
median( [6, 1, 2, 3] ) = 2.5
median( [ ] ) = null
```

stddev(list)

返回列表中数字的标准偏差。

表 4.48. 参数

参数	类型
list	**数字** 元素 **列表**

其他签名

```
stddev( n1, n2, ..., nN )
```

例子

```
stddev( 2, 4, 7, 5 ) = 2.081665999466132735282297706979931
stddev( [47] ) = null
stddev( 47 ) = null
stddev( [ ] ) = null
```

mode(list)

返回列表中数字的模式。如果返回多个元素，则数字以升序排序。

表 4.49. 参数

参数	类型
list	**数字** 元素 **列表**

其他签名

```
mode( n1, n2, ..., nN )
```

例子

```
mode( 6, 3, 9, 6, 6 ) = [6]
mode( [6, 1, 9, 6, 1] ) = [1, 6]
mode( [ ] ) = [ ]
```

4.3.2.4.1. 循环语句

loop 语句可以转换列表或验证某些元素是否满足特定条件：

for in (list)

迭代列表的元素。

表 4.50. 参数

参数	类型
list	**任何** 元素 **列表**

例子

```
for i in [1, 2, 3] return i * i = [1, 4, 9]
for i in [1,2,3], j in [1,2,3] return i*j = [1, 2, 3, 2, 4, 6, 3, 6, 9]
```

有些 in (list)满足（条件）

如果列表中的任何元素满足条件，则返回单个布尔值(true 或 false)。

表 4.51. 参数

参数	类型
list	**任何** 元素 **列表**
条件	布尔值表达式被评估为 true 或 false

例子

```
some i in [1, 2, 3] satisfies i > 3 = true
some i in [1, 2, 3] satisfies i > 4 = false
```

每个 in (list)满足（条件）

如果列表中的每个元素满足条件，则返回单个布尔值(true 或 false)。

表 4.52. 参数

参数	类型
list	任何 元素 列表
条件	布尔值表达式被评估为 true 或 false

例子

```
every i in [1, 2, 3] satisfies i > 1 = false
every i in [1, 2, 3] satisfies i > 0 = true
```

4.3.2.5. 数字功能

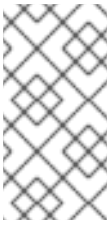
以下功能支持数字操作。

decimal(n, scale)

返回具有指定缩放的数字。

表 4.53. 参数

参数	类型
n	number
scale	范围 [leveloffset6111.6176]范围中的 数字



注意

这个功能被实施为与 FEEL:number 定义一致，用于将十进制数字舍入到最接近的十进制数字。

例子

```
decimal( 1/3, 2 ) = .33
decimal( 1.5, 0 ) = 2
decimal( 2.5, 0 ) = 2
decimal( 1.035, 2 ) = 1.04
decimal( 1.045, 2 ) = 1.04
decimal( 1.055, 2 ) = 1.06
decimal( 1.065, 2 ) = 1.06
```

floor(n)

返回小于或等于指定数字的最大整数。

表 4.54. 参数

参数	类型
n	number

例子

```
floor( 1.5 ) = 1
floor( -1.5 ) = -2
```

ceiling(n)

返回大于或等于指定数字的最小整数。

表 4.55. 参数

参数	类型
n	number

例子

```
ceiling( 1.5 ) = 2
ceiling( -1.5 ) = -1
```

abs(n)

返回绝对值。

表 4.56. 参数

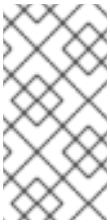
参数	类型
n	、 天和时间持续时间、年和月的时间

例子

```
abs( 10 ) = 10
abs( -10 ) = 10
abs( @"PT5H" ) = @"PT5H"
abs( @"-PT5H" ) = @"PT5H"
```

modulo(dividend, divisor)

返回除形器的划分的剩余部分。如果划分或 divisor 是负的，则结果是与 divisor 相同的符号。



注意

此函数也表示为 modul o(dividend, divisor)= 划分 - divisor*floor(dividend/divisor)。

表 4.57. 参数

参数	类型
划分	number
divisor	number

例子

```
modulo( 12, 5 ) = 2
modulo( -12,5 )= 3
modulo( 12,-5 )= -3
modulo( -12,-5 )= -2
modulo( 10.1, 4.5 )= 1.1
modulo( -10.1, 4.5 )= 3.4
modulo( 10.1, -4.5 )= -3.4
modulo( -10.1, -4.5 )= -1.1
```

sqrt(number)

返回指定编号的方括号。

表 4.58. 参数

参数	类型
n	number

示例

```
sqrt( 16 ) = 4
```

log(number)

返回指定数量的日志变化。

表 4.59. 参数

表 4.59. 参数

参数	类型
n	number

示例

```
decimal( log( 10 ), 2 ) = 2.30
```

exp(number)

返回 Euler 的数字 提升至指定数目的电源。

表 4.60. 参数

参数	类型
n	number

示例

```
decimal( exp( 5 ), 2 ) = 148.41
```

odd(number)

如果指定的数字为奇数，则返回 true。

表 4.61. 参数

参数	类型
n	number

例子

```
odd( 5 ) = true
odd( 2 ) = false
```

even(number)

如果指定号甚至是 true，则返回 true。

表 4.62. 参数

参数	类型
n	number

例子

```
even( 5 ) = false
even ( 2 ) = true
```

4.3.2.6. 日期和时间功能

以下功能支持日期和时间操作。

is(value1, value2)

如果两个值都为 FEEL 语义域中的相同元素，则返回 true。

表 4.63. 参数

参数	类型
value1	任何类型
value2	任何类型

例子

```
is( date( "2012-12-25" ), time( "23:00:50" ) ) = false
is( date( "2012-12-25" ), date( "2012-12-25" ) ) = true
is( time( "23:00:50z" ), time( "23:00:50" ) ) = false
```

4.3.2.7. 范围功能

以下功能支持时序订购操作，以建立单个计算值和此类值范围之间的关系。这些功能与 Health Level Seven(HL7)国际 质量语言(CQL)1.4 语法中 的组件类似。

before( )

当一个元素 A 之前，并且满足评估到 true 的相关要求时，返回 true。

签名

a. before(point1 point2)

b. 在点范围 之前。）

c. before（范围点）

d. before(range1,range2)

评估对 true的要求

a. point1 < point2

b. 点 < range.start or（point = range.start and not(range.start)）

c.

range.end < point or（range.end = point, not(range.end)）

d.

range1.end < range2.start or or（not(range1.end included)or not(range2.start included)和 range1.end = range2.start）

例子

```
before( 1, 10 ) = true
before( 10, 1 ) = false
before( 1, [1..10] ) = false
before( 1, (1..10] ) = true
before( 1, [5..10] ) = true
before( [1..10], 10 ) = false
before( [1..10), 10 ) = true
before( [1..10], 15 ) = true
before( [1..10], [15..20] ) = true
before( [1..10], [10..20] ) = false
before( [1..10), [10..20] ) = true
before( [1..10], (10..20] ) = true
```

after( )

当元素 A 在元素 B 之后并且满足评估为 true 的相关要求时，返回 true。

签名

a.

之后（点1点2）

b.

后（点范围 ）

c.

在完成范围后，请点 .

d.

after(range1 range2)

评估对 true的要求

a.
point1 > point2

b.
point > range.end or（point = range.end, not(range.end)）

c.
range.start > point or（range.start = point, not(range.start included)）

d.
range1.start > range2.end or（（not(range1.start included)or not(range2.end)和 range1.start = range2.end）

例子

```
after( 10, 5 ) = true
after( 5, 10 ) = false
after( 12, [1..10] ) = true
after( 10, [1..10) ) = true
after( 10, [1..10] ) = false
after( [11..20], 12 ) = false
after( [11..20], 10 ) = true
after( (11..20], 11 ) = true
after( [11..20], 11 ) = false
after( [11..20], [1..10] ) = true
after( [1..10], [11..20] ) = false
after( [11..20], [1..11) ) = true
after( (11..20], [1..11] ) = true
```

meets( )

当元素 A 满足一个元素 B 以及满足评估为 true 的相关要求时，返回 true。

签名

a.
meets(range1, range2)

评估对 true的要求

a.
range1.end included 和 range2.start 包括和范围1.end = range2.start

例子

```
meets( [1..5], [5..10] ) = true
meets( [1..5), [5..10] ) = false
meets( [1..5], (5..10] ) = false
meets( [1..5], [6..10] ) = false
```

met by( )

当一个元素 A 满足元素 B 以及满足评估到 true 的相关要求时，返回 true。

签名

a.
因（范围1、范围2）满足.

评估对 true的要求

a.
range1.start included 和 range2.end 包括和 range1.start = range2.end

例子

```
met by( [5..10], [1..5] ) = true
met by( [5..10], [1..5) ) = false
met by( (5..10], [1..5] ) = false
met by( [6..10], [1..5] ) = false
```

overlaps( )

当元素 A 重叠一个元素 B 以及满足评估到 true 的相关要求时，返回 true。

签名

签名

a.

overlaps(range1, range2)

评估对 true的要求

a.

（范围1.end > range2.start 或（range1.end = range2.start and(range1.end included)）和 （范围1.start < range2.end or(range1.start =2.end, and range1.start = range1.start included and range2.end included)

例子

```
overlaps( [1..5], [3..8] ) = true
overlaps( [3..8], [1..5] ) = true
overlaps( [1..8], [3..5] ) = true
overlaps( [3..5], [1..8] ) = true
overlaps( [1..5], [6..8] ) = false
overlaps( [6..8], [1..5] ) = false
overlaps( [1..5], [5..8] ) = true
overlaps( [1..5], (5..8] ) = false
overlaps( [1..5), [5..8] ) = false
overlaps( [1..5), (5..8] ) = false
overlaps( [5..8], [1..5] ) = true
overlaps( (5..8], [1..5] ) = false
overlaps( [5..8], [1..5) ) = false
overlaps( (5..8], [1..5) ) = false
```

overlaps before( )

当元素 A 在元素 B 之前以及满足评估为 true 的相关要求时，返回 true。

签名

a.

overlaps before(range1 range2)

评估对 true的要求

a.

（ range1.start < range2.start or(range1.start = range2.start and range1.start included)and range2.start included）和 （范围1.end > range2.start or(range1.end = range2.start and range1.end included)和 range2.start included）和 （1.end <

range2.end or(range1.end = range2.end = range2.end = range2.end)或 2.start included）和 .

例子

```
overlaps before( [1..5], [3..8] ) = true
overlaps before( [1..5], [6..8] ) = false
overlaps before( [1..5], [5..8] ) = true
overlaps before( [1..5], (5..8] ) = false
overlaps before( [1..5), [5..8] ) = false
overlaps before( [1..5), (1..5] ) = true
overlaps before( [1..5], (1..5] ) = true
overlaps before( [1..5), [1..5] ) = false
overlaps before( [1..5], [1..5] ) = false
```

overlaps after( )

当元素 A 在元素 B 之后以及满足评估为 true 的相关要求时，返回 true。

签名

a.

overlaps after(range1 range2)

评估对 true的要求

a.

（范围2.start 或范围2.start = range1.start 和 range2.start（包括范围2.start included）和 （范围1.start included）和 （范围2.end > range1.start or(range2.end = range1.start and range2.end included)和范围1.start included）和 （范围2.end < range1.end or（range2.end = 1 和(range2.end = 1end)或包括范围）

例子

```
overlaps after( [3..8], [1..5] )= true
overlaps after( [6..8], [1..5] )= false
overlaps after( [5..8], [1..5] )= true
overlaps after( (5..8], [1..5] )= false
overlaps after( [5..8], [1..5) )= false
overlaps after( (1..5], [1..5) )= true
overlaps after( (1..5], [1..5] )= true
```

```
overlaps after( [1..5], [1..5) )= false
overlaps after( [1..5], [1..5] )= false
overlaps after( (1..5), [1..5] )= false
overlaps after( (1..5], [1..6] )= false
overlaps after( (1..5], (1..5] )= false
overlaps after( (1..5], [2..5] )= false
```

finishes( )

当一个元素 A 完成一个元素 B 以及满足评估为 true 的相关要求时，返回 true。

签名

a. 完成（点，范围 ）

b. 完成(range1, range2)

评估对 true的要求

a. range.end included 和 range.end = point

b. range1.end included = range2.end = range1.end = range2.end, and（range1.start > range2.start or(range1.start = range2.start,)或 range2.start included）或 range2.start included）

例子

```
finishes( 10, [1..10] ) = true
finishes( 10, [1..10) ) = false
finishes( [5..10], [1..10] ) = true
finishes( [5..10), [1..10] ) = false
finishes( [5..10), [1..10) ) = true
finishes( [1..10], [1..10] ) = true
finishes( (1..10], [1..10] ) = true
```

完成于。）

当一个元素 A 完成一个元素 B 以及满足评估为 true 的相关要求时，返回 true。

签名

a. 由 （范围结束，请点 ）

b. 完成按（范围1范围2）

评估对 true的要求

a. range.end included 和 range.end = point

b. range1.end included = range2.end = range1.end = range2.end,（range1.start < range2.start or（range1.start = range2.start and（range1.start included or not(range2.start)）

例子

```
finished by( [1..10], 10 ) = true
finished by( [1..10), 10 ) = false
finished by( [1..10], [5..10] ) = true
finished by( [1..10], [5..10) ) = false
finished by( [1..10), [5..10) ) = true
finished by( [1..10], [1..10] ) = true
finished by( [1..10], (1..10] ) = true
```

includes( )

当一个元素 A 包含一个元素 B 以及满足评估为 true 的相关要求时，返回 true。

签名

a. includes(range, point)

b.

includes(range1, range2)

评估对 true的要求

a.

（range.start < point 和 range.end > point）或(range.start = point and range.start included)或(range.end = point.end included)

b.

（范围1.start < range2.start 或（range1.start = range2.start and(range1.start included)）和 （范围1.end > range2.end or(range1.end = range2.end = range2.end = range2.end, orrange1.end included)）

例子

```
includes( [1..10], 5 ) = true
includes( [1..10], 12 ) = false
includes( [1..10], 1 ) = true
includes( [1..10], 10 ) = true
includes( (1..10], 1 ) = false
includes( [1..10), 10 ) = false
includes( [1..10], [4..6] ) = true
includes( [1..10], [1..5] ) = true
includes( (1..10], (1..5] ) = true
includes( [1..10], (1..10) ) = true
includes( [1..10), [5..10) ) = true
includes( [1..10], [1..10) ) = true
includes( [1..10], (1..10] ) = true
includes( [1..10], [1..10] ) = true
```

during( )

当元素 A 在元素 B 期间并且满足评估为 true 的相关要求时，返回 true。

签名

a.

在（点中，范围）

b.

during(range1 range2)

评估对 true的要求

a.
（range.start < point 和 range.end > point）或(range.start = point and range.start included)或(range.end = point.end included)

b.
（range2.end > range1.start or（range2.start = range1.start and(range2.start included)）和（包括range1.start included）和 （范围2.end > range1.end or（range2.end = range1.end = range1.end and(range2.end included)））

例子

```
during( 5, [1..10] ) = true
during( 12, [1..10] ) = false
during( 1, [1..10] ) = true
during( 10, [1..10] ) = true
during( 1, (1..10] ) = false
during( 10, [1..10) ) = false
during( [4..6], [1..10] ) = true
during( [1..5], [1..10] ) = true
during( (1..5], (1..10] ) = true
during( (1..10), [1..10] ) = true
during( [5..10), [1..10) ) = true
during( [1..10), [1..10] ) = true
during( (1..10], [1..10] ) = true
during( [1..10], [1..10] ) = true
```

starts( )

当一个元素 A 启动一个元素 B 以及满足评估为 true 的相关要求时，返回 true。

签名

a.
开始（点，范围 ）

b.
starts(range1, range2)

评估对 true的要求

a.
range.start = 点和范围.start 包括

b.
range1.start = range2.start 和 range1.start included = range2.start includes and（range1.end < range2.end or（range1.end = range2.end, not(range1.end included) 或 range2.end included）

例子

```
starts( 1, [1..10] ) = true
starts( 1, (1..10] ) = false
starts( 2, [1..10] ) = false
starts( [1..5], [1..10] ) = true
starts( (1..5], (1..10] ) = true
starts( (1..5], [1..10] ) = false
starts( [1..5], (1..10] ) = false
starts( [1..10], [1..10] ) = true
starts( [1..10), [1..10] ) = true
starts( (1..10), (1..10) ) = true
```

通过 启动（）

当一个元素 A 由元素 B 启动，同时满足评估为 true 的相关要求时，返回 true。

签名

a.
由范围启动（范围，点 ）

b.
由范围1、范围2启动.

评估对 true的要求

a.
range.start = 点和范围.start 包括

b.
range1.start = range2.start 和 range1.start included = range2.start includes and（range2.end < range1.end = range1.end = range1.end, not(range2.end included)或 range1.end included）

例子

```
started by( [1..10], 1 ) = true
started by( (1..10], 1 ) = false
started by( [1..10], 2 ) = false
started by( [1..10], [1..5] ) = true
started by( (1..10], (1..5] ) = true
started by( [1..10], (1..5] ) = false
started by( (1..10], [1..5] ) = false
started by( [1..10], [1..10] ) = true
started by( [1..10], [1..10) ) = true
started by( (1..10), (1..10) ) = true
```

coincides( )

当一个元素与元素 B 在一起以及满足评估为 true 的相关要求时，返回 true。

签名

a.
coincides(1, point2)

b.
coincides(range1, range2)

评估对 true的要求

a.
point1 = point2

b.
range1.start = range2.start 和 range1.start included = range2.start includes and range1.end = range2.end = included = range2.end = included

例子

```
coincides( 5, 5 ) = true
coincides( 3, 4 ) = false
coincides( [1..5], [1..5] ) = true
```

```
coincides( (1..5), [1..5] ) = false
coincides( [1..5], [2..6] ) = false
```

4.3.2.8. 临时功能

以下功能支持一般的临时操作。

年天（日期 ）

返回本年度当天的 Gregorian 编号。

表 4.64. 参数

参数	类型
date	日期和时间

示例

```
day of year( date(2019, 9, 17) ) = 260
```

星期几（日期 ）

返回星期几：" Monday"、"Tuesday"
、"Wednesday"、"Thursday"、"Friday"、"Saturday" 或 "Sunday"。

表 4.65. 参数

参数	类型
date	日期和时间

示例

```
day of week( date(2019, 9, 17) ) = "Tuesday"
```

年月（日期 ）

返回一年的 Gregorian： "January", "February", "March", "April", "May", "June", "July", "August", "September", "October", "vember" , "vember", "December".

表 4.66. 参数

参数	类型
date	日期和时间

示例

```
month of year( date(2019, 9, 17) ) = "September"
```

年月（日期 ）

返回 ISO 8601 所定义年的 Gregorian 周。

表 4.67. 参数

参数	类型
date	日期和时间

例子

```
week of year( date(2019, 9, 17) ) = 38
week of year( date(2003, 12, 29) ) = 1
week of year( date(2004, 1, 4) ) = 1
week of year( date(2005, 1, 1) ) = 53
week of year( date(2005, 1, 3) ) = 1
week of year( date(2005, 1, 9) ) = 1
```

4.3.2.9. 排序功能

以下功能支持排序操作。

sort（列表，在前面）

返回同一元素的列表，但根据排序函数排序。

表 4.68. 参数

参数	类型
list	list
前	function

示例

```
sort( list: [3,1,4,5,2], precedes: function(x,y) x < y ) = [1,2,3,4,5]
```

4.3.2.10. 上下文功能

以下功能支持上下文操作。

get value(m, key)

从指定条目键的上下文返回值。

表 4.69. 参数

参数	类型
m	context

参数	类型
key	字符串

例子

```
get value( {key1 : "value1"}, "key1" ) = "value1"
get value( {key1 : "value1"}, "unexistent-key" ) = null
```

get entries(m)

为指定上下文返回键值对列表。

表 4.70. 参数

参数	类型
m	context

示例

```
get entries( {key1 : "value1", key2 : "value2"} ) = [ { key : "key1", value : "value1" }, {key :
"key2", value : "value2"} ]
```

4.3.3. FEEL 中的变量和功能名称

与许多传统的表达式语言不同，Friendly Enough Expression Language(FEEL)支持空格和几个特殊字符，作为变量和功能名称的一部分。FEEL 名称必须以 字母 ？或 _ 元素开头。还允许使用 unicode 字母字符。变量名称不能以语言关键字开头，如 和、true 或 每个。变量名称中的剩余字符可以是起始字符的任意字符，以及 数字、空格和特殊字符，如 +、--、/、、*、' 和 。

例如，以下名称是所有有效的 FEEL 名称：

- age
- 十三日期
- 动态 234 预检查过程

FEEL 中的变量和功能名称有几个限制：

不确定性

使用空格、关键字和其他特殊字符，因为名称的一部分可使 FEEL 模糊。在表达式的上下文中解决了混淆，与从左到右的顺序匹配名称。解析器解析变量名称，作为范围中匹配最长的名称。如果需要，您可以使用 （） 来混淆名称。

名称中的空格

DMN 规格限制在 FEEL 名称中使用空格。根据 DMN 规格，名称可以包含多个空格，但不能包含两个连续的空格。

为了便于使用语言并避免因为空格导致的常见错误，Red Hat Process Automation Manager 会删除连续使用空间的限制。Red Hat Process Automation Manager 支持使用任意数量的连续空格的变量名称，但会将它们规范化到一个空间中。例如，在 Red Hat Process Automation Manager 中，变量引用带有空格和第一个空格 的名字。

Red Hat Process Automation Manager 还规范化了使用其它空格，如网页、标签页和换行符中的不可破的空白。从 Red Hat Process Automation Manager FEEL 引擎视角，所有这些字符在处理前被规范化为一个空白。

中的关键字

中的 关键字是语言中唯一不能用作变量名称一部分的关键字。虽然规格允许在变量名称中间使用关键字，但变量名称 中的 用法与 的 getmmar 定义（每个 和 某些 表达式结构）冲突。

4.4. 框表达式中的 DMN 决策逻辑

DMN 中的方框表达式是您用来在决策要求图(DRD)中定义决策节点和商业知识模型的基本逻辑。一些

方框的表达式可包含其他框的表达式，但顶级的表达式与单个 DRD 工件的决策逻辑对应。DRD 代表 DMN 决策模型的流，但已框的表达式定义各个节点的实际决策逻辑。DRD 和 boxed 表达式组成一个完整的、功能 DMN 决策模型。

以下是 DMN 框表达式的类型：

- 决策表
- 字面表达式
- 上下文
- 关系
- Functions
- 调用
- 列表

注意

Red Hat Process Automation Manager 不提供 Business Central 中的方框表达式，但支持 FEEL 列表 数据类型，您可以在开箱即用的字面表达式中使用。有关 Red Hat Process Automation Manager 中列出 数据类型和其他 FEEL 数据类型的更多信息，请参阅 第 4.3.1 节 “FEEL 中的数据类型”。

您在框表达式中使用的所有 Friendly Enough Expression Language(FEEL)表达式必须符合 OMG Decision Model 和 Notation 规格 中的 FEEL 语法要求。

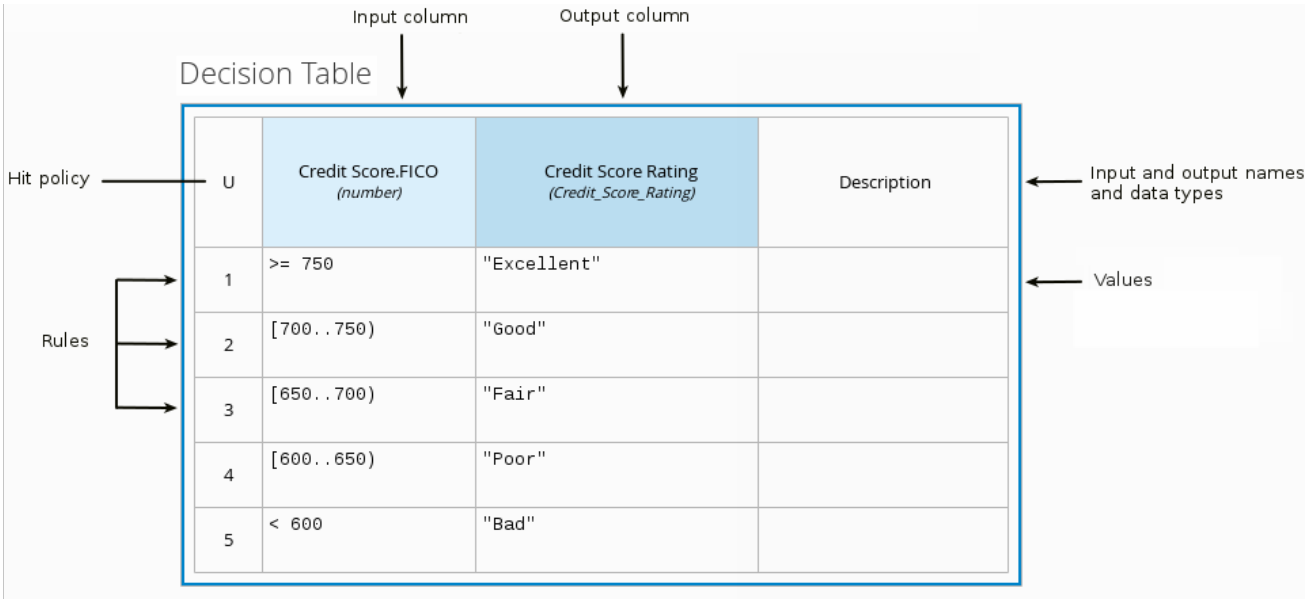
4.4.1. DMN 决策表

DMN 中的决策表是按表格格式显示一个或多个业务规则的可视化表示。您可以使用决策表为在决策模

型给定点上应用这些规则的决定节点定义规则。每个规则均由表中的一行组成，包括针对该特定行定义条件（输入）和结果（输出）的列。每行的定义足够精确，可以利用条件的值生成结果。输入和输出值可以是 FEEL 表达式或定义的数据类型值。

例如，以下决策表根据有定义的 loan applicant 分数分数来决定分数等级：

图 4.3. 得分评级的决策表



根据申请款款资格和投标类型，以下决策表确定了申请策略：

图 4.4. 用于 lending 策略的决定表

Strategy *(Decision Table)*

U	Eligibility *(string)*	BureauCallType *(string)*	Strategy *(tStrategy)*	Description
1	"INELIGIBLE"	-	"DECLINE"	Disregard BureauCallType when ineligible.
2	"ELIGIBLE"	"FULL","MINI"	"BUREAU"	
3	"ELIGIBLE"	"NONE"	"THROUGH"	

下表决定 loans 特定资格的资格，作为 loan prequalification 决策模型中的排除决定节点：

图 4.5. loan prequalification 的决策表

Loan Pre-Qualification *(Decision Table)*

F	Credit Score Rating *(Credit_Score_Rating)*	Back End Ratio *(Back_End_Ratio)*	Front End Ratio *(Front_End_Ratio)*	Loan Pre-Qualification *(Loan_Qualification)*		Description
				Qualification *(string)*	Reason *(string)*	
1	"Poor", "Bad"	-	-	"Not Qualified"	"Credit Score too low."	
2	-	"Insufficient"	"Sufficient"	"Not Qualified"	"Debt to income ratio is too high."	
3	-	"Sufficient"	"Insufficient"	"Not Qualified"	"Mortgage payment to income ratio is too high."	
4	-	"Insufficient"	"Insufficient"	"Not Qualified"	"Debt to income ratio is too high AND mortgage payment to income ratio is too high."	
5	"Fair", "Good", "Excellent"	"Sufficient"	"Sufficient"	"Qualified"	"The borrower has been successfully prequalified for the requested loan."	

决策表是建模规则和决策逻辑的流行方法，在许多方法（如 DMN）和实施框架中使用。

重要

Red Hat Process Automation Manager 支持 DMN 决策表和 Drools-native 决策表，但它们是具有不同语法要求的不同资产类型，且不可交换。有关 Red Hat Process Automation Manager 中的 dros 原生决策表的更多信息，*请参阅使用电子表格决策表设计决策服务*。

4.4.1.1. DMN 决策表中的点击策略

按策略决定了在决定表中的多个规则与提供的输入值匹配时如何到达结果。例如，如果决策表中的一个规则向军事人员应用销售折扣，并且另一条规则向学生应用折扣，那么当客户既是学生，在军事区中，决策表点击政策必须指明是否应用一个折扣或其他（唯一 ）或购买（1 个有效）或折扣（选择Sum）。您可以在决策表的左下角指定点击策略的单一字符（U、F、C+）。

DMN 支持以下决策表点击策略：

- 唯一(U)：仅发送一个要匹配的规则。任何重叠都会引发错误。
- any(A)：一个 要匹配的多个规则，但它们必须具有相同的输出。如果多个匹配规则没有相同的输出，则会出现一个错误。
- 优先级(P)：通过不同的输出来传输要匹配的多个规则。输出值列表中第一个输出会被选择。

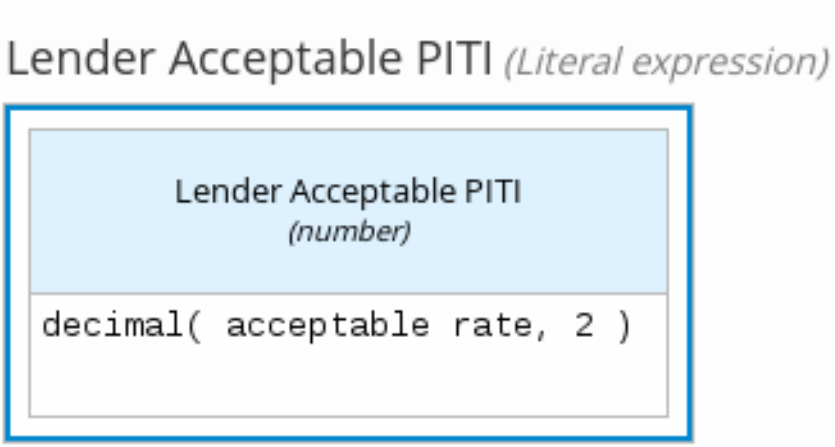
- First(F)：按规则顺序使用第一个匹配项。
- 收集（C+、C>、C<、C#）：基于聚合功能，来自多个规则的输出。
  - 收集(C)：任意列表中的聚合值。
  - 收集 Sum(C+)：输出所有收集的值的总和。值必须是数字。
  - 收集 Min(C<)：输出匹配项中的最小值。生成的值必须比较，如数字、日期或文本（循环顺序）。
  - collect Max(C>)：输出匹配项中的最大值。生成的值必须比较，如数字、日期或文本（循环顺序）。
  - collect Count(C#)：输出匹配规则的数量。

4.4.2. 已安装的字面表达式

DMN 中的方框字面表达式是表格单元格中的文字，通常是带有标记的列和分配的数据类型。您可以使用封闭的字面表达式，在决定特定节点的 FEEL 中直接在 FEEL 中定义简单或复杂节点逻辑或决策数据。literal FEEL 表达式必须符合 OMG Decision Model 和 Notation 规格中的 FEEL 语法要求。

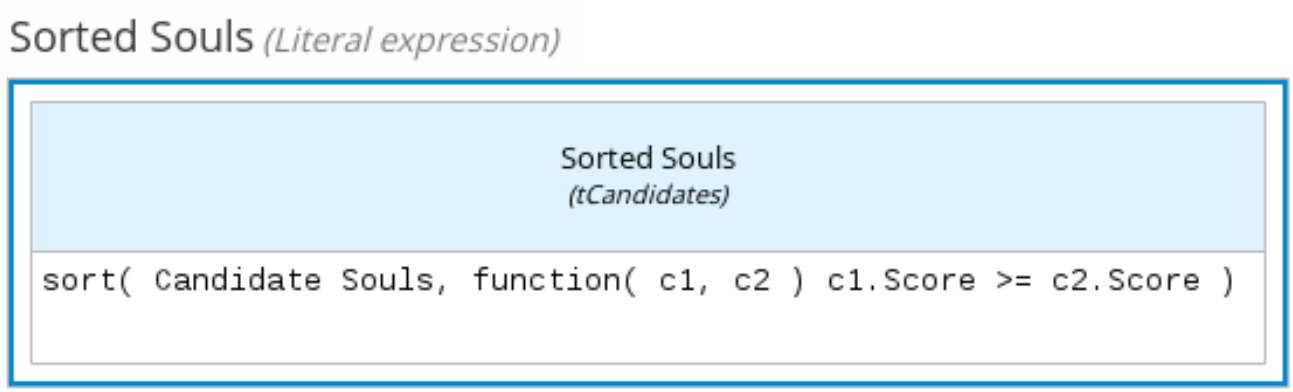
例如，以下方框的字面表达式定义了最低可接受 PITI 计算（principal、interest、gincipes 和 保险），其中 可接受的率 是一个在 DMN 模型中定义的变量：

图 4.6. 开箱即用的字面表达式，最小 PITI 值



以下固定字面表达式根据具体标准（如年龄、位置和兴趣）在在线保护应用程序中对潜在候选（相对应）进行排序：

图 4.7. 用于匹配的在线 dating 候选的字面表达式



4.4.3. Boxed 上下文表达式

DMN 中的方框上下文表达式是一组变量名称和值，其值具有结果值。每个 name-value 对都是上下文条目。您可以使用上下文表达式来代表决策逻辑中的数据定义，并在 DMN 决策模型中为所需决策元素设置一个值。框上下文表达式中的值可以是数据类型值或 FEEL 表达式，或者可以包含任何类型的嵌套子表达式，如决策表、字面表达式或其他上下文表达式。

例如，以下方框上下文表达式定义了根据定义的数据类型（TPassengerTable、tFlightNumberList）在flight-rebooking 决策模式中排序延迟乘客的因素：

图 4.8. Boxed 上下文表达式，用于 flight 乘客等待列表

Prioritized Waiting List *(Context)*

#	Prioritized Waiting List *(tPassengerTable)*	
1	Cancelled Flights *(tFlightNumberList)*	`Flight List[ Status = "cancelled" ].Flight Number`
2	Waiting List *(tPassengerTable)*	`Passenger List[ list contains( Cancelled Flights, Flight Number ) ]`
	<result>	`sort( Waiting List, Passenger Priority )`

以下方框上下文表达式定义了可确定是否可以基于主体、兴趣、税务和保险(PITI)作为前端比例计算的因素：

图 4.9. 用于前端客户端 PITI 比率的方框上下文表达式

Front End Ratio *(Context)*

#	Front End Ratio *(Front_End_Ratio)*		
1	Client PITI *(number)*		
		#	PITI
		1	pmt *(number)*	`(Requested Product.Amount*((Requested Product.Rate/100)/12))/ (1-(1/(1+(Requested Product.Rate/100)/12)**-Requested Product.Term))`
		2	tax *(number)*	`Applicant Data.Monthly.Tax`
		3	insurance *(number)*	`Applicant Data.Monthly.Insurance`
		4	income *(number)*	`Applicant Data.Monthly.Income`
	<result>	`if Client PITI <= Lender Acceptable PITI() then "Sufficient" else "Insufficient"`	

4.4.4. 框关系表达式

DMN 中的方框关系表达式是传统的数据表，其中包含给定实体（以行的形式列出）。在特定节点决定相关实体的决策中，您可以使用框关系表来定义相关实体的决策数据。框关系表达式与上下文表达式相似，因为它们设置变量名称和值，但关系表达式不包含结果值，并且根据每个列中定义的变量列出所有变量值。

例如，以下开箱即用关系表达式提供有关员工的恶意决策中的员工信息：

图 4.10. 与员工信息建立框关系表达式

Employee Information *(Relation)*

#	Name *(string)*	Dept *(string)*	Salary *(number)*
1	"John"	"Sales"	100000
2	"Mary"	"Finances"	120000

4.4.5. Boxed 功能表达式

DMN 中的 box 功能表达式是一个参数化的表达式，包含字面的 FEEL 表达式、外部 JAVA 或 PMML 函数的嵌套上下文表达式，或任何类型的嵌套式表达式。默认情况下，所有业务知识模型都定义为开箱即用的功能表达式。您可以使用 boxed 功能表达式为您的决策逻辑上调用功能，并定义所有业务知识模型。

例如，以下方框的函数表达式决定在flight-rebooking decision model 中的flight 动态容量：

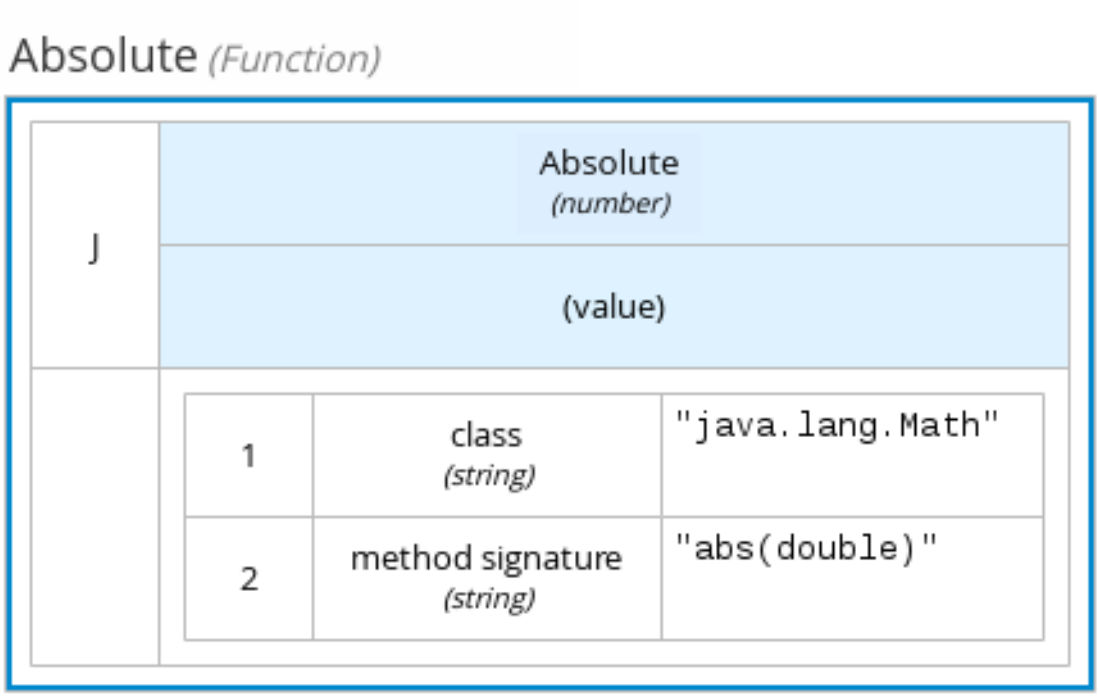
图 4.11. 用于动态容量的 box 功能表达式

Flight Capacity *(Function)*

F	Flight Capacity *(boolean)*
	(flight, rebooked list)
	`flight.Capacity > count( rebooked list[ Flight Number = flight.Flight Number ] )`

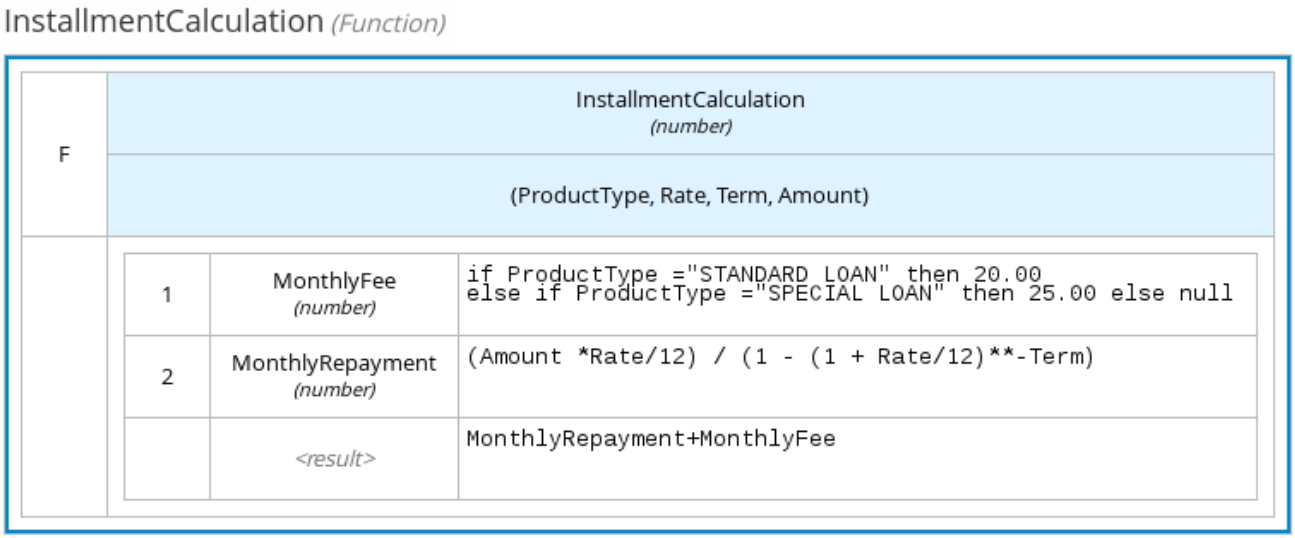
以下开箱即用的功能表达式包含基本的 Java 功能，作为在决策模型计算中确定绝对值的上下文表达式：

图 4.12. 开箱即用的功能表达式，用于绝对值



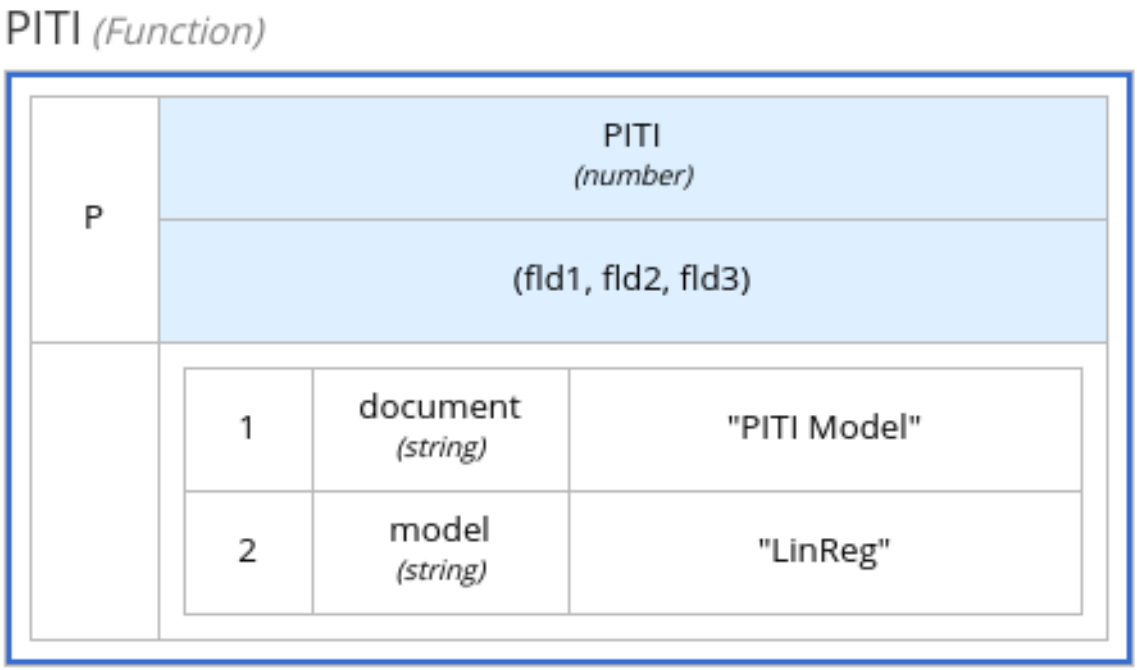
以下方框式函数表达式确定一个月的分流安装，作为商业知识模型确定在方便的决策中，该函数值定义为嵌套上下文表达式：

图 4.13. 在商业知识模式下安装计算的已选式功能表达式



以下开箱即用的功能表达式使用 DMN 文件中附带的 PMML 模型来定义最低可接受的 PITI 计算（费用、兴趣、税务和保险）：

图 4.14. 具有业务知识模型中包含的 PMML 模型的 box 功能表达式



4.4.6. Boxed invocation 表达式

DMN 中的开箱即用调用表达式是调用业务知识模型的框式表达式。框调用表达式包含要调用的业务知识模型的名称以及参数绑定列表。每个绑定都由一行中的两个方框表达式表示： 左侧的方框包含参数的名称，右侧框中包含了用于评估调用的商业知识模型的绑定表达式。您可以在特定的决策节点上调用该决策节点，这是决策模型中定义的业务知识模型。

例如，以下复选框调用调用表达式调用 Reassign Next Passenger Business knowledge model，作为在flight-rebooking 决策模型中包括的决策节点：

图 4.15. 开箱即用的调用表达式来重新分配flight 乘客

Rebooked Passengers *(Invocation)*

#	Rebooked Passengers *(tPassengerTable)*	
	Reassign Next Passenger	
1	Waiting List *(tPassengerTable)*	`Prioritized Waiting List`
2	Reassigned Passengers List *(tPassengerTable)*	`[]`
3	Flights *(tFlightTable)*	`Flight List`

以下复选框调用调用表达式调用 InstallmentCalculation Business knowledge model，以在做出经济决定前计算每月安装量：

图 4.16. 所需每月安装的调用表达式

RequiredMonthlyInstallment *(Invocation)*

#	RequiredMonthlyInstallment *(number)*	
	InstallmentCalculation	
1	ProductType *(string)*	`RequestedProduct.ProductType`
2	Rate *(number)*	`RequestedProduct.Rate`
3	Term *(string)*	`RequestedProduct.Term`
4	Amount *(number)*	`RequestedProduct.Amount`

4.4.7. Boxed list 表达式

DMN 中的方框列表表达式代表项目的 FEEL 列表。您可以使用已框的列表来定义决定特定节点的相关

项目列表。您还可以将字面的 FEEL 表达式用于列出单元中的项目，以创建更复杂的列表。

例如，以下复选框列表表达式标识 loan Application decisions 服务中的已获信分数机构：

图 4.17. 批准分数机构的已选信列表表达式

Approved credit score agencies *(List)*

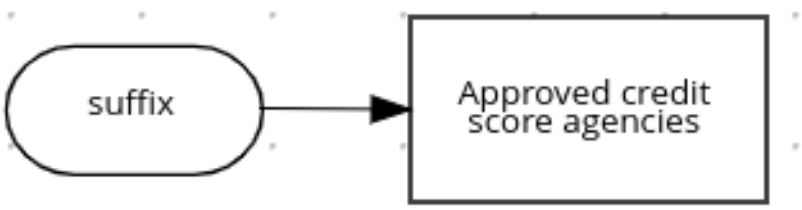
1	"Acme Agency, Inc."
2	"Top Scores, Inc."
3	"Global Scoring, Inc."

以下复选框列表表达式还标识出类机构，但使用 FEEL 逻辑根据 DMN 输入节点定义机构状态（Inc.、Spulce、SAS、GA）：

图 4.18. 使用 FEEL 逻辑的已核算列表表达式（用于批准的信贷分数）状态

Approved credit score agencies *(List)*

1	"Acme Agency" + suffix
2	"Top Scores" + suffix
3	"Global Scoring" + suffix



4.5. DMN 模型示例

以下是一个实际的 DMN 模型示例，它演示了如何使用决策建模，根据输入数据、情况和公司指南来获得决策。在这种情况下，一个从 San Diego 到 New York 的航班被取消，需要受影响的航空公司为其不便见的乘客找到备选安排。

首先，桥接将收集必要的信息，以确定从差到其目的地的最佳方法：

输入数据

- flights 列表
- 乘客列表

决策

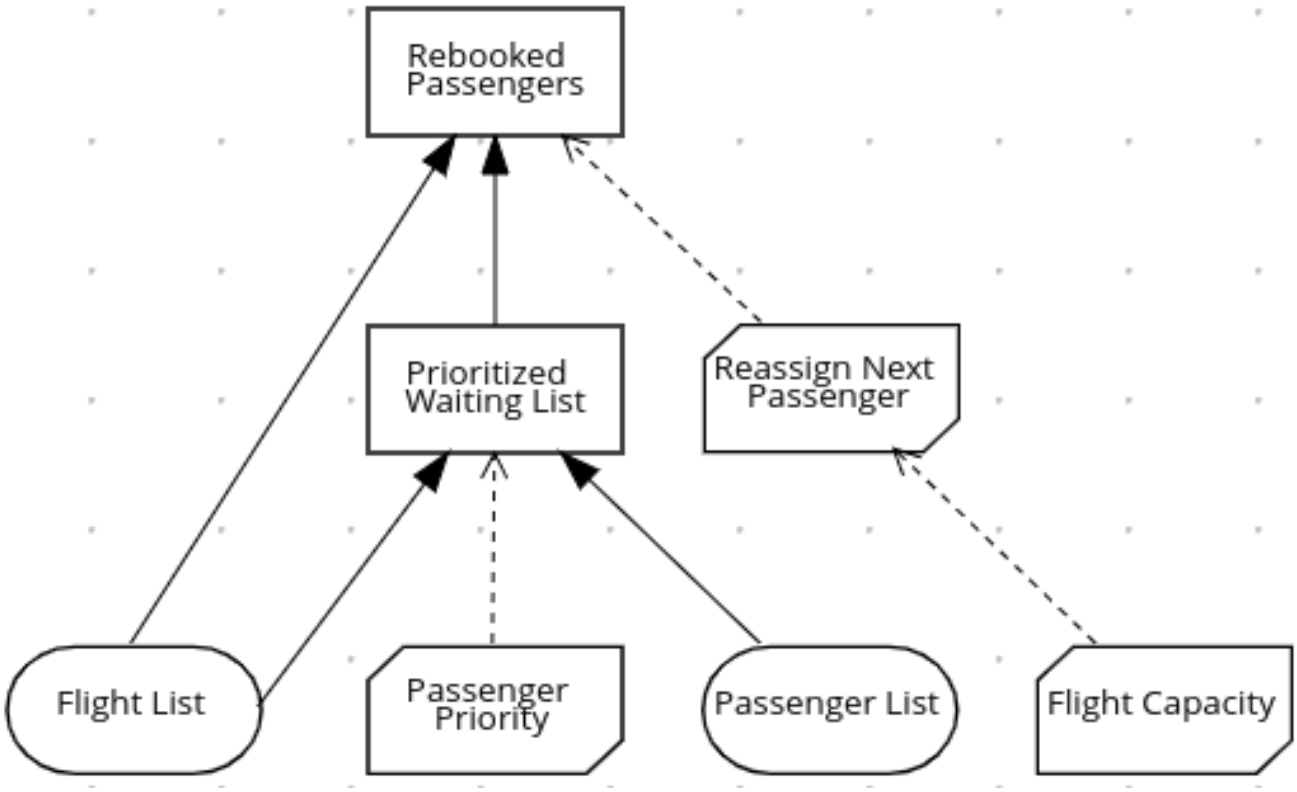
- 排列新班上的乘客的乘客
- 确定将提供哪些乘客

商业知识模型

- 决定乘客优先级的公司进程
- 任何有可用空间的机
- 用于确定如何分配不便传递传递者的公司规则

然后，在以下决策要求图(DRD)以确定最佳重新预订解决方案时，使用 DMN 标准对决策流程建模：

图 4.19. 用于飞行的 DRD



与流图类似，DRD 使用形成表示进程中的不同元素。OVAL 包含两个必要输入数据，rerectangles 包含模型中的决策点，通过调控基（业务知识模型）包含可重复调用的可重复使用的逻辑。

每个元素的 DRD draws 逻辑来自 box 表达式，使用 FEEL 表达式或数据类型值提供变量定义。

某些已选箱的表达式是基本的，例如建立优先级等待列表的以下决策：

图 4.20. 框式上下文表达式示例显示优先级等待列表

Prioritized Waiting List *(Context)*

#	Prioritized Waiting List *(tPassengerTable)*	
1	Cancelled Flights *(tFlightNumberList)*	`Flight List[ Status = "cancelled" ].Flight Number`
2	Waiting List *(tPassengerTable)*	`Passenger List[ list contains( Cancelled Flights, Flight Number ) ]`
	<result>	`sort( Waiting List, Passenger Priority )`

某些方框的表达式比更详细的和计算更为复杂，例如以下商业知识模型用于重新分配下一个延迟乘客：

图 4.21. 用于乘客重新分配的 boxed 功能表达式

Reassign Next Passenger *(Function)*

F	Reassign Next Passenger *(tPassengerTable)*		
	(Waiting List, Reassigned Passengers List, Flights)		
	1	Next Passenger *(tPassenger)*	`Waiting List[1]`
	2	Original Flight *(tFlight)*	`Flights[ Flight Number = Next Passenger.Flight Number ][1]`

	3	Best Alternate Flight *(tFlight)*	`Flights[ From = Original Flight.From and`
			`To = Original Flight.To and`
			`Departure > Original Flight.Departure and`
			`Status = "scheduled" and`
			`Flight Capacity( item, Reassigned Passengers List ) ][1]`

	4	Reassigned Passenger *(tPassenger)*	1 Name *(string)* `Next Passenger.Name` 2 Status *(string)* `Next Passenger.Status` 3 Miles *(number)* `Next Passenger.Miles` 4 Flight Number *(string)* `Best Alternate Flight.Flight Number` *<result>* *Select expression*
	5	Remaining Waiting List *(tPassengerTable)*	`remove( Waiting List, 1 )`
	6	Updated Reassigned Passengers List *(tPassengerTable)*	`append( Reassigned Passengers List, Reassigned Passenger )`

		<result>	`if count( Remaining Waiting List ) > 0`
			`then`
			`Reassign Next Passenger( Remaining Waiting List,`
			`Updated Reassigned Passengers List,`
			`Flights )`
			`else`
			`Updated Reassigned Passengers List`

以下是这个决策模型的 DMN 源文件：

```
<dmn:definitions xmlns="https://www.drools.org/kie-dmn/Flight-rebooking"
xmlns:dmn="http://www.omg.org/spec/DMN/20151101/dmn.xsd"
xmlns:feel="http://www.omg.org/spec/FEEL/20140401" id="_0019_flight_rebooking" name="0019-
flight-rebooking" namespace="https://www.drools.org/kie-dmn/Flight-rebooking">
 <dmn:itemDefinition id="_tFlight" name="tFlight">
  <dmn:itemComponent id="_tFlight_Flight" name="Flight Number">
   <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tFlight_From" name="From">
   <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tFlight_To" name="To">
   <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tFlight_Dep" name="Departure">
   <dmn:typeRef>feel:dateTime</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tFlight_Arr" name="Arrival">
   <dmn:typeRef>feel:dateTime</dmn:typeRef>
```

```
  </dmn:itemComponent>
  <dmn:itemComponent id="_tFlight_Capacity" name="Capacity">
    <dmn:typeRef>feel:number</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tFlight_Status" name="Status">
    <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemComponent>
</dmn:itemDefinition>
<dmn:itemDefinition id="_tFlightTable" isCollection="true" name="tFlightTable">
  <dmn:typeRef>tFlight</dmn:typeRef>
</dmn:itemDefinition>
<dmn:itemDefinition id="_tPassenger" name="tPassenger">
  <dmn:itemComponent id="_tPassenger_Name" name="Name">
    <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tPassenger_Status" name="Status">
    <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tPassenger_Miles" name="Miles">
    <dmn:typeRef>feel:number</dmn:typeRef>
  </dmn:itemComponent>
  <dmn:itemComponent id="_tPassenger_Flight" name="Flight Number">
    <dmn:typeRef>feel:string</dmn:typeRef>
  </dmn:itemComponent>
</dmn:itemDefinition>
<dmn:itemDefinition id="_tPassengerTable" isCollection="true" name="tPassengerTable">
  <dmn:typeRef>tPassenger</dmn:typeRef>
</dmn:itemDefinition>
<dmn:itemDefinition id="_tFlightNumberList" isCollection="true" name="tFlightNumberList">
  <dmn:typeRef>feel:string</dmn:typeRef>
</dmn:itemDefinition>
<dmn:inputData id="i_Flight_List" name="Flight List">
  <dmn:variable name="Flight List" typeRef="tFlightTable"/>
</dmn:inputData>
<dmn:inputData id="i_Passenger_List" name="Passenger List">
  <dmn:variable name="Passenger List" typeRef="tPassengerTable"/>
</dmn:inputData>
<dmn:decision name="Prioritized Waiting List" id="d_PrioritizedWaitingList">
  <dmn:variable name="Prioritized Waiting List" typeRef="tPassengerTable"/>
  <dmn:informationRequirement>
    <dmn:requiredInput href="#i_Passenger_List"/>
  </dmn:informationRequirement>
  <dmn:informationRequirement>
    <dmn:requiredInput href="#i_Flight_List"/>
  </dmn:informationRequirement>
  <dmn:knowledgeRequirement>
    <dmn:requiredKnowledge href="#b_PassengerPriority"/>
  </dmn:knowledgeRequirement>
  <dmn:context>
    <dmn:contextEntry>
      <dmn:variable name="Cancelled Flights" typeRef="tFlightNumberList"/>
      <dmn:literalExpression>
        <dmn:text>Flight List[ Status = "cancelled" ].Flight Number</dmn:text>
      </dmn:literalExpression>
    </dmn:contextEntry>
    <dmn:contextEntry>
```

```
      <dmn:variable name="Waiting List" typeRef="tPassengerTable"/>
      <dmn:literalExpression>
        <dmn:text>Passenger List[ list contains( Cancelled Flights, Flight Number ) ]</dmn:text>
      </dmn:literalExpression>
    </dmn:contextEntry>
    <dmn:contextEntry>
      <dmn:literalExpression>
        <dmn:text>sort( Waiting List, passenger priority )</dmn:text>
      </dmn:literalExpression>
    </dmn:contextEntry>
  </dmn:context>
</dmn:decision>
<dmn:decision name="Rebooked Passengers" id="d_RebookedPassengers">
  <dmn:variable name="Rebooked Passengers" typeRef="tPassengerTable"/>
  <dmn:informationRequirement>
    <dmn:requiredDecision href="#d_PrioritizedWaitingList"/>
  </dmn:informationRequirement>
  <dmn:informationRequirement>
    <dmn:requiredInput href="#i_Flight_List"/>
  </dmn:informationRequirement>
  <dmn:knowledgeRequirement>
    <dmn:requiredKnowledge href="#b_ReassignNextPassenger"/>
  </dmn:knowledgeRequirement>
  <dmn:invocation>
    <dmn:literalExpression>
      <dmn:text>reassign next passenger</dmn:text>
    </dmn:literalExpression>
    <dmn:binding>
      <dmn:parameter name="Waiting List"/>
      <dmn:literalExpression>
        <dmn:text>Prioritized Waiting List</dmn:text>
      </dmn:literalExpression>
    </dmn:binding>
    <dmn:binding>
      <dmn:parameter name="Reassigned Passengers List"/>
      <dmn:literalExpression>
        <dmn:text>[]</dmn:text>
      </dmn:literalExpression>
    </dmn:binding>
    <dmn:binding>
      <dmn:parameter name="Flights"/>
      <dmn:literalExpression>
        <dmn:text>Flight List</dmn:text>
      </dmn:literalExpression>
    </dmn:binding>
  </dmn:invocation>
</dmn:decision>
<dmn:businessKnowledgeModel id="b_PassengerPriority" name="passenger priority">
  <dmn:encapsulatedLogic>
    <dmn:formalParameter name="Passenger1" typeRef="tPassenger"/>
    <dmn:formalParameter name="Passenger2" typeRef="tPassenger"/>
    <dmn:decisionTable hitPolicy="UNIQUE">
      <dmn:input id="b_Passenger_Priority_dt_i_P1_Status" label="Passenger1.Status">
        <dmn:inputExpression typeRef="feel:string">
          <dmn:text>Passenger1.Status</dmn:text>
        </dmn:inputExpression>
```

```
    <dmn:inputValues>
      <dmn:text>"gold", "silver", "bronze"</dmn:text>
    </dmn:inputValues>
  </dmn:input>
  <dmn:input id="b_Passenger_Priority_dt_i_P2_Status" label="Passenger2.Status">
    <dmn:inputExpression typeRef="feel:string">
      <dmn:text>Passenger2.Status</dmn:text>
    </dmn:inputExpression>
    <dmn:inputValues>
      <dmn:text>"gold", "silver", "bronze"</dmn:text>
    </dmn:inputValues>
  </dmn:input>
  <dmn:input id="b_Passenger_Priority_dt_i_P1_Miles" label="Passenger1.Miles">
    <dmn:inputExpression typeRef="feel:string">
      <dmn:text>Passenger1.Miles</dmn:text>
    </dmn:inputExpression>
  </dmn:input>
  <dmn:output id="b_Status_Priority_dt_o" label="Passenger1 has priority">
    <dmn:outputValues>
      <dmn:text>true, false</dmn:text>
    </dmn:outputValues>
    <dmn:defaultOutputEntry>
      <dmn:text>false</dmn:text>
    </dmn:defaultOutputEntry>
  </dmn:output>
  <dmn:rule id="b_Passenger_Priority_dt_r1">
    <dmn:inputEntry id="b_Passenger_Priority_dt_r1_i1">
      <dmn:text>"gold"</dmn:text>
    </dmn:inputEntry>
    <dmn:inputEntry id="b_Passenger_Priority_dt_r1_i2">
      <dmn:text>"gold"</dmn:text>
    </dmn:inputEntry>
    <dmn:inputEntry id="b_Passenger_Priority_dt_r1_i3">
      <dmn:text>>= Passenger2.Miles</dmn:text>
    </dmn:inputEntry>
    <dmn:outputEntry id="b_Passenger_Priority_dt_r1_o1">
      <dmn:text>true</dmn:text>
    </dmn:outputEntry>
  </dmn:rule>
  <dmn:rule id="b_Passenger_Priority_dt_r2">
    <dmn:inputEntry id="b_Passenger_Priority_dt_r2_i1">
      <dmn:text>"gold"</dmn:text>
    </dmn:inputEntry>
    <dmn:inputEntry id="b_Passenger_Priority_dt_r2_i2">
      <dmn:text>"silver","bronze"</dmn:text>
    </dmn:inputEntry>
    <dmn:inputEntry id="b_Passenger_Priority_dt_r2_i3">
      <dmn:text>-</dmn:text>
    </dmn:inputEntry>
    <dmn:outputEntry id="b_Passenger_Priority_dt_r2_o1">
      <dmn:text>true</dmn:text>
    </dmn:outputEntry>
  </dmn:rule>
  <dmn:rule id="b_Passenger_Priority_dt_r3">
    <dmn:inputEntry id="b_Passenger_Priority_dt_r3_i1">
      <dmn:text>"silver"</dmn:text>
```

```
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r3_i2">
          <dmn:text>"silver"</dmn:text>
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r3_i3">
          <dmn:text>>= Passenger2.Miles</dmn:text>
        </dmn:inputEntry>
        <dmn:outputEntry id="b_Passenger_Priority_dt_r3_o1">
          <dmn:text>true</dmn:text>
        </dmn:outputEntry>
      </dmn:rule>
      <dmn:rule id="b_Passenger_Priority_dt_r4">
        <dmn:inputEntry id="b_Passenger_Priority_dt_r4_i1">
          <dmn:text>"silver"</dmn:text>
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r4_i2">
          <dmn:text>"bronze"</dmn:text>
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r4_i3">
          <dmn:text>-</dmn:text>
        </dmn:inputEntry>
        <dmn:outputEntry id="b_Passenger_Priority_dt_r4_o1">
          <dmn:text>true</dmn:text>
        </dmn:outputEntry>
      </dmn:rule>
      <dmn:rule id="b_Passenger_Priority_dt_r5">
        <dmn:inputEntry id="b_Passenger_Priority_dt_r5_i1">
          <dmn:text>"bronze"</dmn:text>
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r5_i2">
          <dmn:text>"bronze"</dmn:text>
        </dmn:inputEntry>
        <dmn:inputEntry id="b_Passenger_Priority_dt_r5_i3">
          <dmn:text>>= Passenger2.Miles</dmn:text>
        </dmn:inputEntry>
        <dmn:outputEntry id="b_Passenger_Priority_dt_r5_o1">
          <dmn:text>true</dmn:text>
        </dmn:outputEntry>
      </dmn:rule>
    </dmn:decisionTable>
  </dmn:encapsulatedLogic>
  <dmn:variable name="passenger priority" typeRef="feel:boolean"/>
</dmn:businessKnowledgeModel>
<dmn:businessKnowledgeModel id="b_ReassignNextPassenger" name="reassign next passenger">
  <dmn:encapsulatedLogic>
    <dmn:formalParameter name="Waiting List" typeRef="tPassengerTable"/>
    <dmn:formalParameter name="Reassigned Passengers List" typeRef="tPassengerTable"/>
    <dmn:formalParameter name="Flights" typeRef="tFlightTable"/>
    <dmn:context>
      <dmn:contextEntry>
        <dmn:variable name="Next Passenger" typeRef="tPassenger"/>
        <dmn:literalExpression>
          <dmn:text>Waiting List[1]</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
```

```
        <dmn:variable name="Original Flight" typeRef="tFlight"/>
        <dmn:literalExpression>
          <dmn:text>Flights[ Flight Number = Next Passenger.Flight Number ][1]</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:variable name="Best Alternate Flight" typeRef="tFlight"/>
        <dmn:literalExpression>
          <dmn:text>Flights[ From = Original Flight.From and To = Original Flight.To and
Departure > Original Flight.Departure and Status = "scheduled" and has capacity( item,
Reassigned Passengers List ) ][1]</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:variable name="Reassigned Passenger" typeRef="tPassenger"/>
        <dmn:context>
          <dmn:contextEntry>
            <dmn:variable name="Name" typeRef="feel:string"/>
            <dmn:literalExpression>
              <dmn:text>Next Passenger.Name</dmn:text>
            </dmn:literalExpression>
          </dmn:contextEntry>
          <dmn:contextEntry>
            <dmn:variable name="Status" typeRef="feel:string"/>
            <dmn:literalExpression>
              <dmn:text>Next Passenger.Status</dmn:text>
            </dmn:literalExpression>
          </dmn:contextEntry>
          <dmn:contextEntry>
            <dmn:variable name="Miles" typeRef="feel:number"/>
            <dmn:literalExpression>
              <dmn:text>Next Passenger.Miles</dmn:text>
            </dmn:literalExpression>
          </dmn:contextEntry>
          <dmn:contextEntry>
            <dmn:variable name="Flight Number" typeRef="feel:string"/>
            <dmn:literalExpression>
              <dmn:text>Best Alternate Flight.Flight Number</dmn:text>
            </dmn:literalExpression>
          </dmn:contextEntry>
        </dmn:context>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:variable name="Remaining Waiting List" typeRef="tPassengerTable"/>
        <dmn:literalExpression>
          <dmn:text>remove( Waiting List, 1 )</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:variable name="Updated Reassigned Passengers List" typeRef="tPassengerTable"/>
        <dmn:literalExpression>
          <dmn:text>append( Reassigned Passengers List, Reassigned Passenger )</dmn:text>
        </dmn:literalExpression>
      </dmn:contextEntry>
      <dmn:contextEntry>
        <dmn:literalExpression>
```

```
        <dmn:text>if count( Remaining Waiting List ) > 0 then reassign next passenger(
Remaining Waiting List, Updated Reassigned Passengers List, Flights ) else Updated
Reassigned Passengers List</dmn:text>
        </dmn:literalExpression>
       </dmn:contextEntry>
      </dmn:context>
    </dmn:encapsulatedLogic>
    <dmn:variable name="reassign next passenger" typeRef="tPassengerTable"/>
    <dmn:knowledgeRequirement>
      <dmn:requiredKnowledge href="#b_HasCapacity"/>
    </dmn:knowledgeRequirement>
  </dmn:businessKnowledgeModel>
  <dmn:businessKnowledgeModel id="b_HasCapacity" name="has capacity">
    <dmn:encapsulatedLogic>
      <dmn:formalParameter name="flight" typeRef="tFlight"/>
      <dmn:formalParameter name="rebooked list" typeRef="tPassengerTable"/>
      <dmn:literalExpression>
        <dmn:text>flight.Capacity > count( rebooked list[ Flight Number = flight.Flight Number ]
)</dmn:text>
      </dmn:literalExpression>
    </dmn:encapsulatedLogic>
    <dmn:variable name="has capacity" typeRef="feel:boolean"/>
  </dmn:businessKnowledgeModel>
</dmn:definitions>
```

第 5 章 RED HAT PROCESS AUTOMATION MANAGER 中的 DMN 支持

Red Hat Process Automation Manager 为 DMN 1.1、1.2、1.3 和 1.4 模型提供运行时支持，符合级别 3。您可以通过几种方法将 DMN 模型与您的 Red Hat Process Automation Manager 决策服务集成：

- 使用 DMN 设计器直接在 Business Central 中设计您的 DMN 模型。
- 在 Business Central 中导入 DMN 文件(Menu → Design → Projects → Import Asset)。您导入到 Business Central 的任何 DMN 1.1 和 1.3 模型（不包含 DMN 1.3 功能），在 DMN 设计程序中打开，保存将转换为 DMN 1.2 模型。
- 在没有 Business Central 的情况下，将 DMN 文件作为项目知识 JAR(KJAR)文件的一部分打包。

下表总结了 Red Hat Process Automation Manager 中每个 DMN 版本的设计和运行时支持：

表 5.1. Red Hat Process Automation Manager 中的 DMN 支持

DMN 版本	DMN 引擎支持	DMN 模型器支持	
	æ‰§è¡Œ	Open	ä¿å˜
DMN 1.1	✓	✓	✗
DMN 1.2	✓	✓	✓
DMN 1.3	✓	✓	✗
DMN 1.4	✓	✗	✗

除了所有 DMN 一致性级别 3 要求外，红帽流程自动化管理器还包括对 FEEL 和 DMN 模型组件的增强和修复，以优化使用红帽流程自动化管理器实施 DMN 决策服务的经验。从平台的角度来看，DMN 模型与 Red Hat Process Automation Manager 中的任何其他业务资产一样，如 DRL 文件或电子表格决策

表，您可以在 Red Hat Process Automation Manager 项目中包含并部署到 KIE 服务器，以便启动您的 DMN 决策服务。

有关使用 Red Hat Process Automation Manager 项目打包和部署方法包含外部 DMN 文件的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

您可以使用红帽构建的 Kogito 微服务设计一个新的 DMN 决策服务，作为 DMN 决策服务的云原生功能的替代选择。您可以将 DMN 服务迁移到红帽构建的 Kogito 微服务。有关迁移到红帽构建的 Kogito 微服务的更多信息，请参阅 *迁移到红帽构建的 Kogito 微服务*。

5.1. RED HAT PROCESS AUTOMATION MANAGER 中的可配置 DMN 属性

Red Hat Process Automation Manager 提供了以下 DMN 属性，您可以在 KIE 服务器或客户端应用程序上执行 DMN 模型时配置。当您在 KIE 服务器上部署项目时，您可以使用 Red Hat Process Automation Manager 项目中的 kmodule.xml 文件来配置其中的一些属性。

org.kie.dmn.strictConformance

启用后，此属性会默认禁用除 DMN 标准外提供的扩展或配置集，如一些帮助程序功能或 DMN 1.2 的增强功能，反向移植到 DMN 1.1。您可以使用此属性配置决策引擎，使其只支持纯 DMN 功能，比如在运行 DMN 技术兼容性套件 (TCK)时。

默认值： **false**

```
-Dorg.kie.dmn.strictConformance=true
```

org.kie.dmn.runtime.typecheck

启用后，此属性启用验证符合在 DMN 模型中声明类型的实际值，作为 DRD 元素的输入或输出。您可以使用此属性验证是否提供给 DMN 模型或由 DMN 模型生成的数据是否与模型中指定的对象兼容。

默认值： **false**

```
-Dorg.kie.dmn.runtime.typecheck=true
```

org.kie.dmn.decisionservice.coercesingleton

默认情况下，此属性生成决策服务的结果，定义单一输出决策是输出决策值的单个值。禁用后，此属性使决策服务的结果是定义单个输出决策 的结果是具有该决策的单个条目的上下文。您可以使用此属性来根据项目要求调整决策服务输出。

默认值为： **true**

```
-Dorg.kie.dmn.decisionservice.coercesingleton=false
```

org.kie.dmn.profiles.$PROFILE_NAME

当使用 Java 完全限定名称评估时，此属性会在启动时将 DMN 配置集加载到决策引擎中。您可以使用此属性实现预定义的 DMN 配置集，其支持的功能与 DMN 标准不同。例如，如果您使用 Signavio DMN modeller 创建 DMN 模型，请使用此属性在 DMN 决策服务中实施来自 Signavio DMN 配置集的功能。

```
-Dorg.kie.dmn.profiles.signavio=org.kie.dmn.signavio.KieDMNSignavioProfile
```

org.kie.dmn.runtime.listeners.$LISTENER_NAME

当使用 Java 完全限定名称评估时，此属性会在启动时加载并注册 DMN Runtime Listener。您可以使用此属性注册 DMN 侦听器，以便在 DMN 模型评估过程中获得多个事件的通知。

要在 KIE 服务器上部署项目时配置此属性，请在项目的 **kmodule.xml** 文件中修改此属性。当监听器特定于您的项目并且必须在 KIE 服务器中将配置应用到您部署的项目时，此方法非常有用。

```
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
  <configuration>
    <property key="org.kie.dmn.runtime.listeners.mylistener" value="org.acme.MyDMNListener"/>
  </configuration>
</kmodule>
```

要为 **Red Hat Process Automation Manager** 环境全局配置此属性，请使用命令终端或任何其他全局应用程序配置机制修改此属性。当决策引擎作为 Java 应用程序的一部分嵌入时，此方法非常有用。

```
-Dorg.kie.dmn.runtime.listeners.mylistener=org.acme.MyDMNListener
```

org.kie.dmn.compiler.execmodel

启用后，此属性可让 DMN 决策表逻辑在运行时编译成可执行规则模型。您可以使用此属性来更有效地评估 DMN 决策表逻辑。当可执行模型编译最初没有在项目编译期间执行时，此属性很有用。启用此属性可能会导致决策引擎在第一次评估过程中添加编译时间，但后续编译效率更高。

默认值： **false**

```
-Dorg.kie.dmn.compiler.execmodel=true
```

5.2. RED HAT PROCESS AUTOMATION MANAGER 中的可配置 DMN 验证

默认情况下，Red Hat Process Automation Manager 项目的 pom.xml 文件中的 kie-maven-plugin 组件使用以下 < validateDMN> 配置来执行 DMN 模型资产的预编译验证，并执行 DMN 决策表静态分析：

- VALIDATE_SCHEMA: DMN 模型文件根据 DMN 规格 XSD 模式进行验证，以确保文件有效 XML 并符合规格。
- VALIDATE_MODEL ：为 DMN 模型执行预编译分析，以确保基本语义与 DMN 规范一致。
- ANALYZE_DECISION_TABLE: 为差距或重叠静态分析 DMN 决策表，并确保决策表的语义遵循最佳实践。

您可以修改默认的 DMN 验证和 DMN 决策表分析行为，以便在项目构建期间只执行指定的验证，也可以完全禁用此默认行为，如下例所示：

DMN 验证和决策表分析的默认配置

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>

<validateDMN>VALIDATE_SCHEMA,VALIDATE_MODEL,ANALYZE_DECISION_TABLE</validate
DMN>
  </configuration>
</plugin>
```

配置仅执行 DMN 决策表静态分析

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
```

```
  <configuration>
    <validateDMN>ANALYZE_DECISION_TABLE</validateDMN>
  </configuration>
</plugin>
```

配置仅执行 XSD 模式验证

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <validateDMN>VALIDATE_SCHEMA</validateDMN>
  </configuration>
</plugin>
```

配置只执行 DMN 模型验证

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <validateDMN>VALIDATE_MODEL</validateDMN>
  </configuration>
</plugin>
```

配置来禁用所有 DMN 验证

```
<plugin>
  <groupId>org.kie</groupId>
  <artifactId>kie-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
```

```
    <validateDMN>disable</validateDMN>
  </configuration>
</plugin>
```

注意

如果您输入了未识别的 &lt ;validateDMN& gt; 配置标记，则所有预编译验证都会被禁用，并且 Maven 插件会发出相关的日志消息。

第 6 章 在 BUSINESS CENTRAL 中创建和编辑 DMN 型号

您可以使用 Business Central 中的 DMN 设计器设计 DMN 决策要求图(DRD)并为完整功能 DMN 决策模型定义决策逻辑。Red Hat Process Automation Manager 为 DMN 1.2 模型提供符合级别 3 的设计支持，包括 FEEL 和 DMN 模型组件的增强和修复，以优化使用 Red Hat Process Automation Manager 实施 DMN 决策服务的体验。Red Hat Process Automation Manager 还提供对 DMN 1.1、1.2、1.3 和 1.4 模型的运行时支持，但任何 DMN 1.1 和 1.3 模型（不包含您导入到 Business Central 中的 DMN 1.3 功能），在 DMN 设计程序中打开，并保存将转换为 DMN 1.2 模型。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。

2. 在 Business Central 项目中创建或导入 DMN 文件。

   要创建 DMN 文件，点 Add Asset → DMN，输入一个参考性 DMN 模型名称，选择适当的 软件包，然后点击 Ok。

   要导入现有的 DMN 文件，请单击 Import Asset，输入 DMN 模型名称，选择适当的 软件包，选择 DMN 文件，然后单击确定。

   新的 DMN 文件现在在 Project Explorer 的 DMN 面板中列出，并显示 DMN 决策要求图(DRD)canvas。

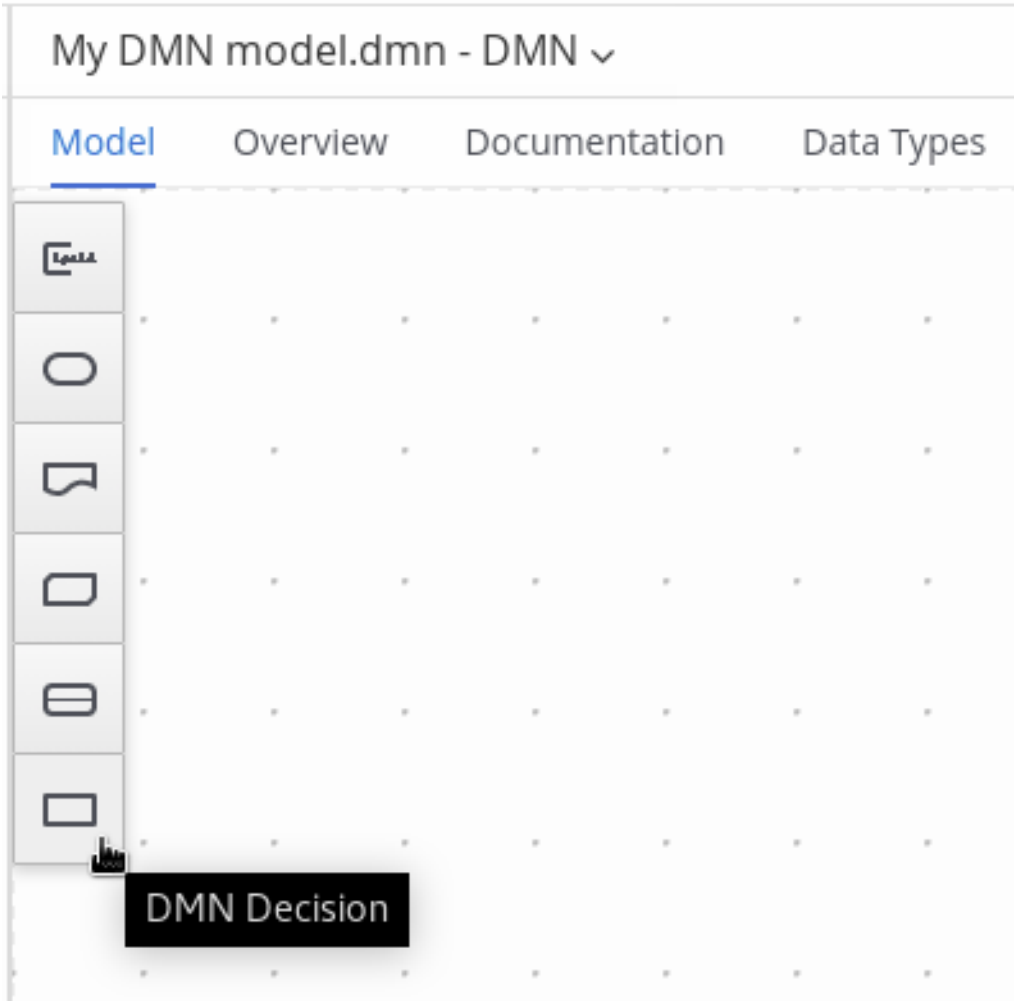
   注意

   如果您导入了不包含布局信息的 DMN 文件，则在 DMN 设计器中自动格式化导入的决策要求图(DRD)。点 DMN 设计器中的 Save 保存 DRD 布局。

   如果没有自动格式化导入的 DRD，您可以在 DMN 设计器的右上角选择 Perform Automatic 布局 图标来格式化 DRD。

3. 在新的或导入的 DMN 决策要求图(DRD)中开始从左侧工具栏中点并拖动其中一个 DMN 节点：

图 6.1. 添加 DRD 组件



可用的 DRD 组件如下：

- 决策 ：将此节点用于 DMN 决策，其中一个或多个输入元素根据定义的决策逻辑确定输出。
- 商业知识模型 ：此节点可通过一个或多个决策元素来重复利用功能。具有相同逻辑的决策但依赖于不同的子项数据或子子数据，使用商业知识模型来确定要遵循哪些流程。
- 知识来源 ：将此节点用于对遵循决策或商业知识模型的外部授权、文档、提交或策略。知识源是指参考实际因素而不是可执行的业务规则。
- 输入数据 ：使用此节点获取决策节点或业务知识模型中使用的信息。输入数据通常包括与业务相关的业务级别概念或对象，如在客制策略中使用的 loan Applicationlicant 数据。
- 文本注解 ：使用此节点来解释与输入数据节点、决策节点、业务知识模型或知识来源相

关的备注。

- 决策服务 ：此节点包含一组可重用决策，实施为要调用的决策服务。决策服务可用于其他 DMN 模型，并可从外部应用程序或 BPMN 业务流程调用。

4. 在 DMN 设计器 canvas 中，双击新的 DRD 节点进入信息性节点名称。

5. 如果节点是决策或商业知识模型，请选择节点以显示节点选项，并点击 Edit 图标打开 DMN 框的表达式设计程序，以定义节点的决策逻辑：

图 6.2. 打开新决策节点复选框表达式

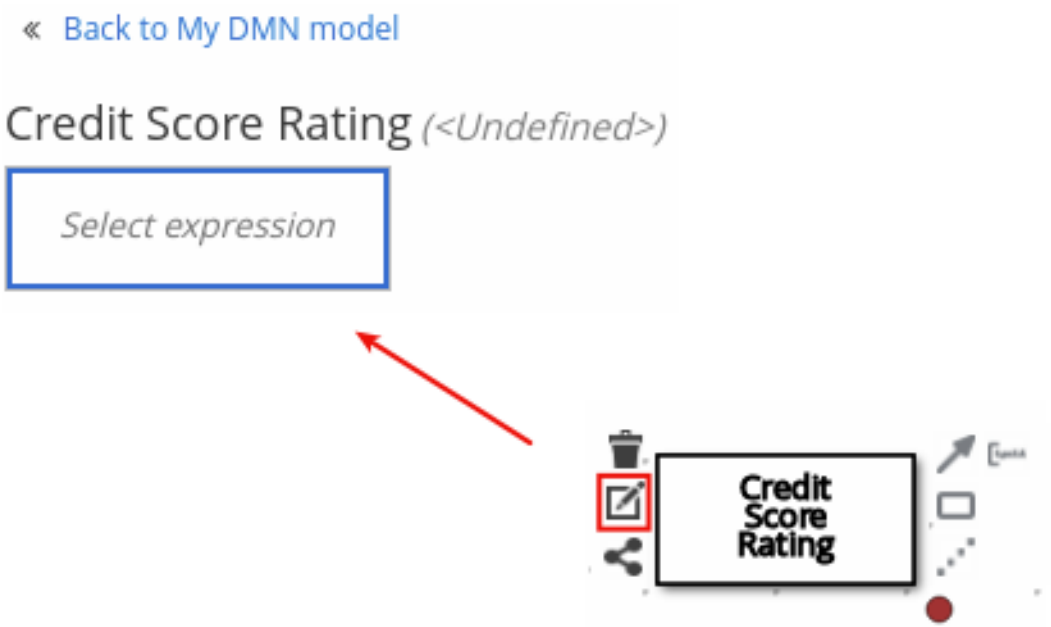
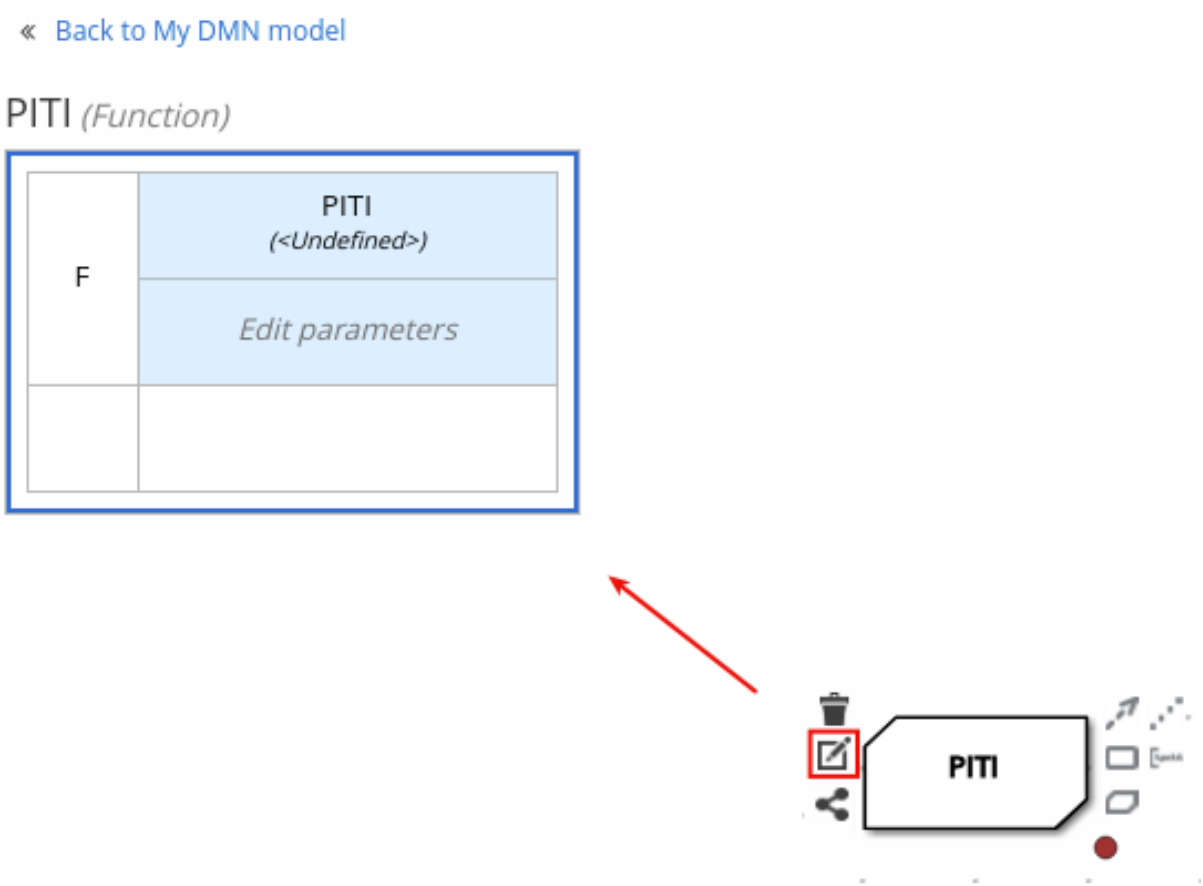


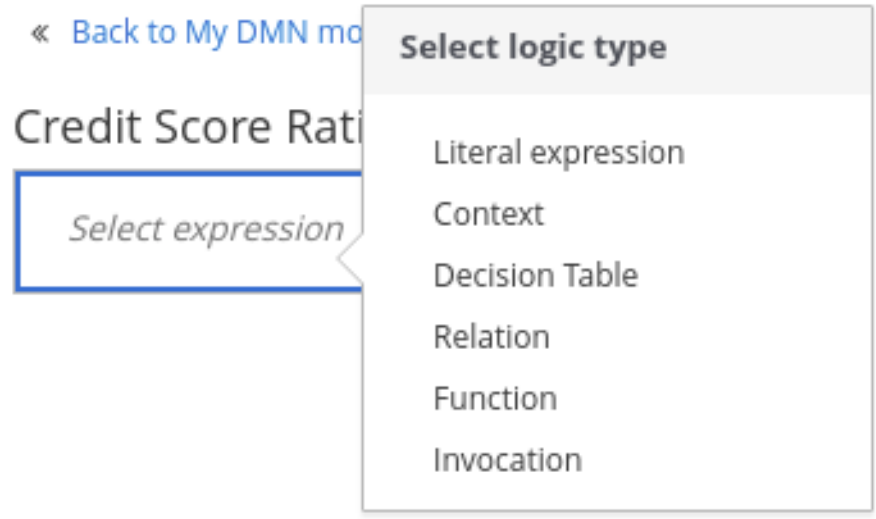
图 6.3. 打开新的商业知识模型已选中表达式



默认情况下，所有商业知识模型都被定义为带有字面意义的 FEEL 表达式、外部 JAVA 或 PMML 函数的嵌套上下文表达式，或任何类型的嵌套式表达式。

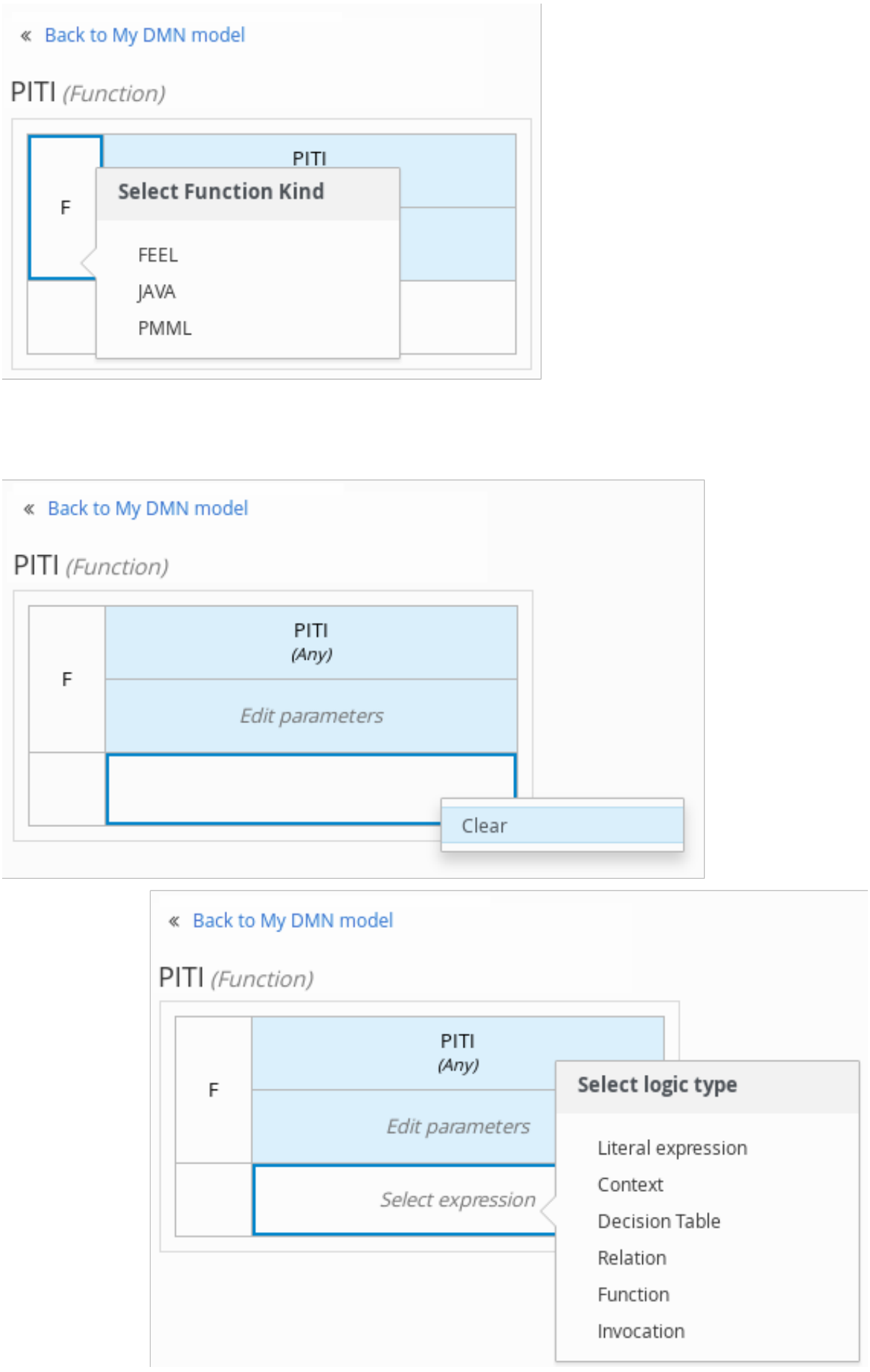
对于决策节点，您可以点击未定义表来选择要使用的已框表达式类型，如方框式的字面表达式、方框式上下文表达式、决策表或其他 DMN 框的表达式。

图 6.4. 为决策节点选择逻辑类型



对于业务知识模型，您可以单击左上角的功能单元以选择功能类型，或者右键单击功能价值单元，选择 Clear，然后选择另一个类型的选框表达式。

图 6.5. 为业务知识模型选择功能或其他逻辑类型



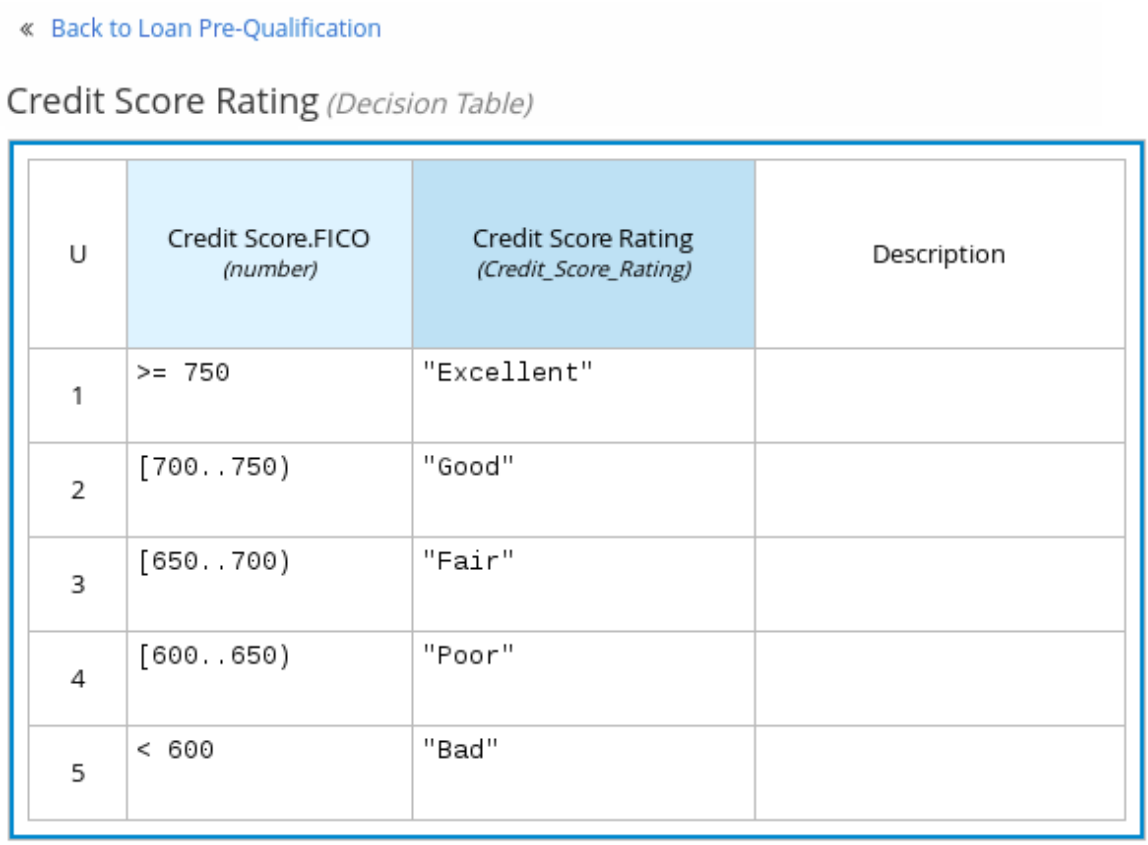
6.

在选中的表达式设计器中，用于决策节点（任何表达式类型）或业务知识模型（功能表达式），单击适用的表单元格来定义表名称、变量数据类型、变量名称和值、功能参数和绑定，或者 FEEL 表达式包含在决策逻辑中。

您可以右键单击适用于适用操作的其他操作的单元，如插入或删除表行和列或清除表内容。

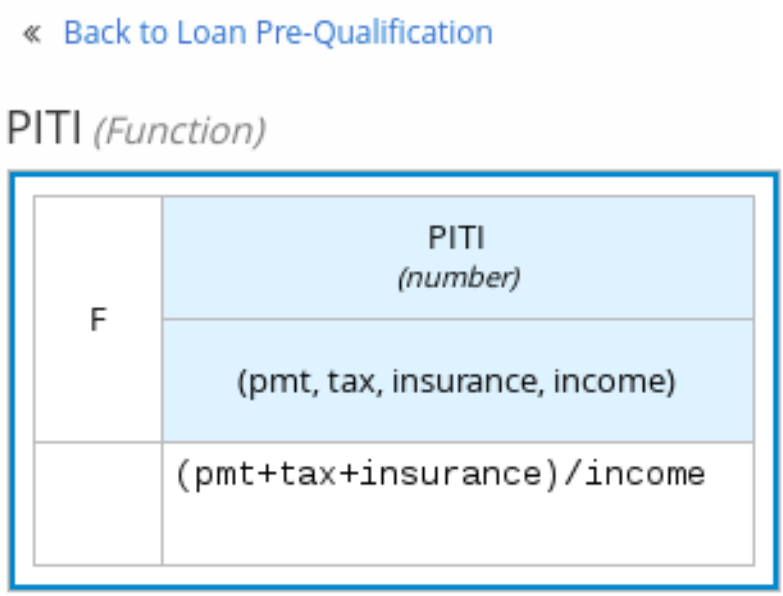
以下是一个决策节点的示例决策表，它根据 loan applicant 的贡献分数确定分数等级：

图 6.6. 得分评级的决策节点决策表



以下是业务知识模型的框框功能表达式，它根据主体、兴趣、税务和保险(PITI)计算分流支付：

图 6.7. PITI 计算业务知识模型功能



7. 为所选节点定义决策逻辑后，点 Back to "<MODEL_NAME>" 以返回到 DRD 视图。

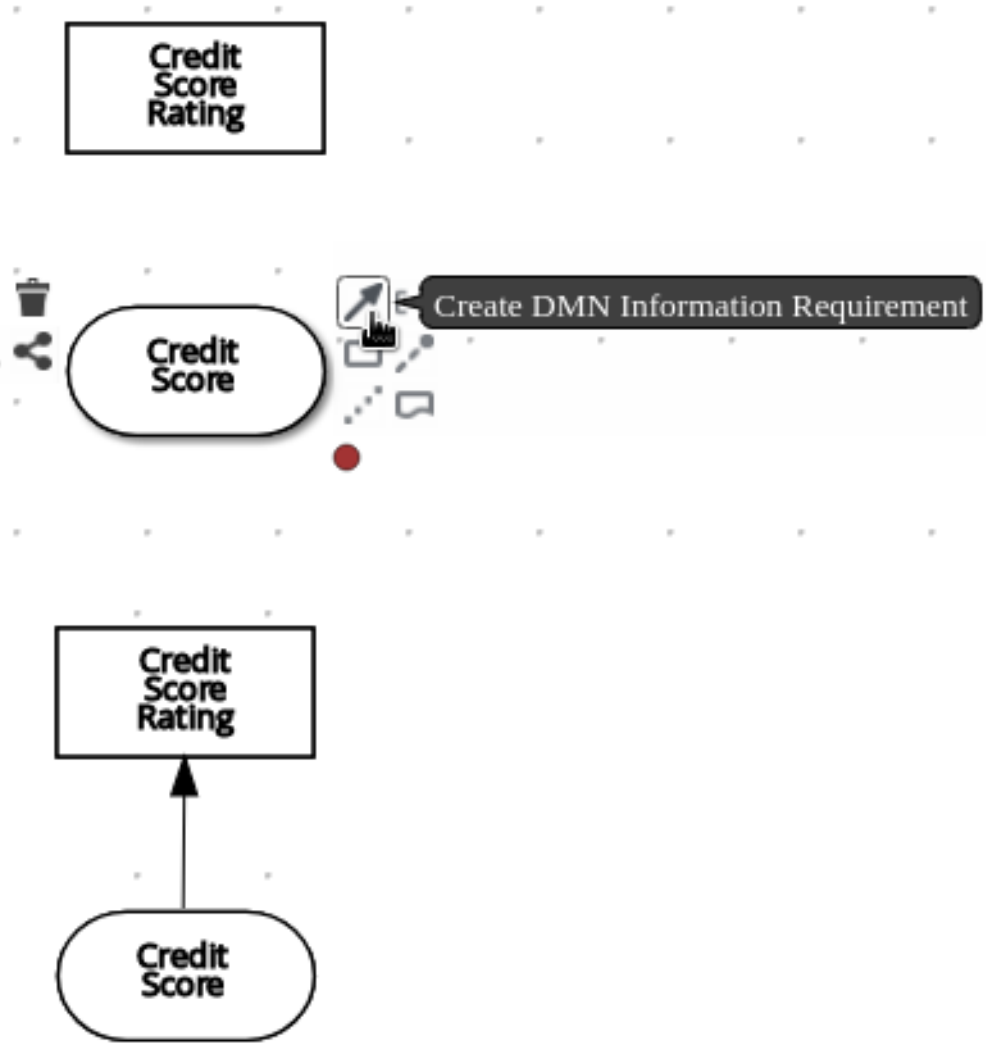
8. 对于所选的 DRD 节点，使用可用的连接选项创建并连接到 DRD 中的下一个节点，或者点击新节点并从左工具栏中分离新节点。

节点类型决定支持哪些连接选项。例如，Input 数据节点 可以使用适用的连接类型连接到决策节点、知识源或文本注解，而 知识源 节点则可连接到任何 DRD 元素。决策 节点只能连接到其他决策或文本注解。

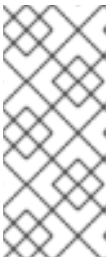
根据节点类型，有以下连接类型可用：

- 信息要求 ：将此从输入数据节点或决策节点的连接与需要信息的另一种决策节点一起使用。
- 知识要求 ：将此从业务知识模式与决策节点的连接与称为决策逻辑的其他业务知识模型进行联系。
- 颁发机构要求 ：将此从输入数据节点或决策节点与依赖知识来源的连接，或者从知识源到决策节点、业务知识模型或其他知识来源。
- 关联 ：使用此连接从输入数据节点、决策节点、业务知识模型或知识源到文本注解。

图 6.8. 将分数得分输入连接到分数等级决定



9. 继续添加和定义决策模型中剩余的 DRD 组件。定期在 DMN 设计器中点 Save 来保存您的工作。



注意

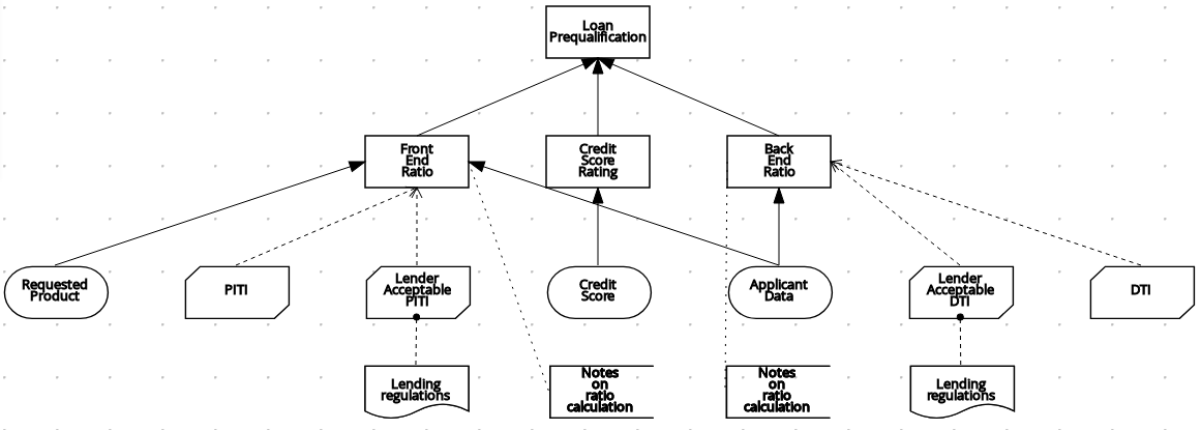
当您定期保存 DRD 时，DMN 设计程序会对 DMN 模型执行静态验证，并可能会生成错误消息，直到模型被完全定义为止。在完全定义 DMN 模型后，如果任何错误保留，请相应地对指定问题进行故障排除。

10. 添加并定义 DRD 的所有组件后，点 Save 保存并验证完成的 DRD。

要调整 DRD 布局，您可以在 DMN 设计器右上角选择 Perform 自动布局 图标。

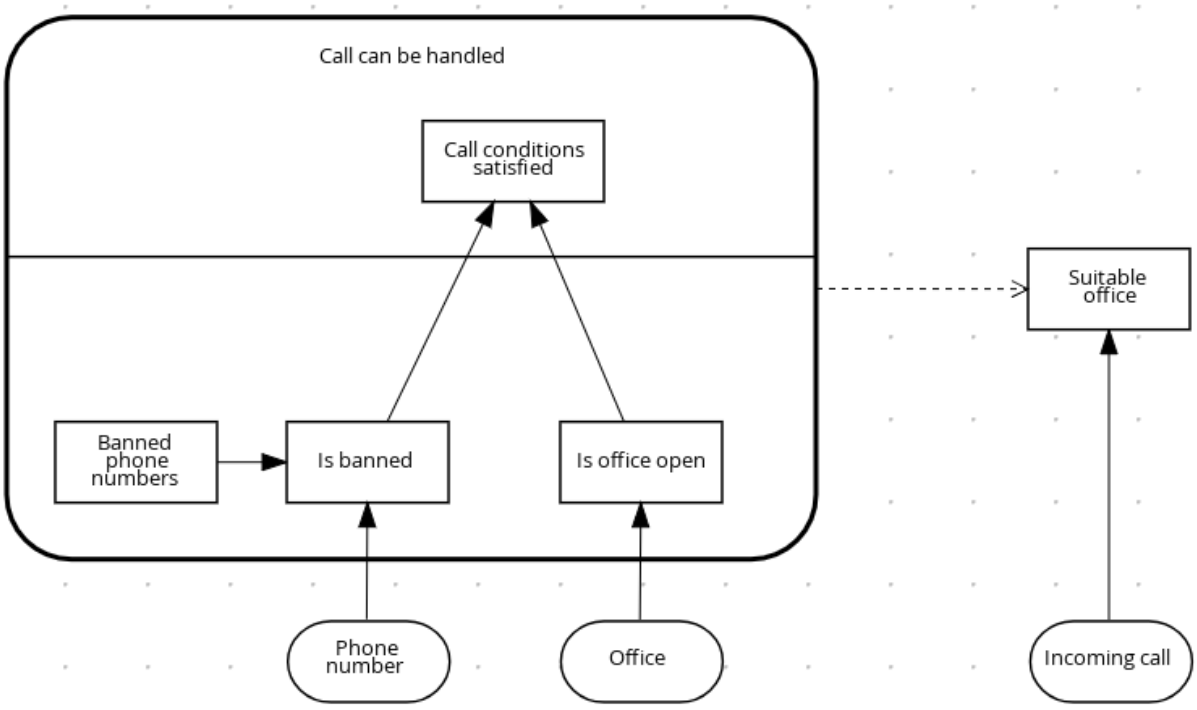
以下是 loan prequalification 决策模型的 DRD 示例：

图 6.9. 为 loan prequalification 完成 DRD



以下是使用可重复使用的决策服务处理决策模型的 DRD 示例：

图 6.10. 使用决策服务进行电话接手处理完成的 DRD



在 DMN 决策服务节点中，底部片段中的决策节点纳入了来自决策服务之外的输入数据，以达到决策服务节点的主要网段中最终决策。然后，在 DMN 模型的后续决策或商业知识要求中实施来自决策的顶级决策。您可以在其他 DMN 模式下重复使用 DMN 决策服务，以应用具有不同输入数据和不同传出连接的相同决策逻辑。

6.1. 在 BUSINESS CENTRAL 的框中定义 DMN 决策逻辑

DMN 中的方框表达式是您用来在决策要求图(DRD)中定义决策节点和商业知识模型的基本逻辑。一些

方框的表达式可包含其他框的表达式，但顶级的表达式与单个 DRD 工件的决策逻辑对应。DRD 代表 DMN 决策模型的流，但已框的表达式定义各个节点的实际决策逻辑。DRD 和 boxed 表达式组成一个完整的、功能 DMN 决策模型。

您可以使用 Business Central 中的 DMN 设计器，使用内置的表达式为 DRD 组件定义决策逻辑。

先决条件

- 在 Business Central 中创建或导入 DMN 文件。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects，点项目名称并选择要修改的 DMN 文件。

2. 在 DMN 设计器 canvas 中，选择您要定义的决定节点或商业知识模型节点，并点击 Edit 图标打开 DMN 框的表达式设计器：

图 6.11. 打开新决策节点复选框表达式

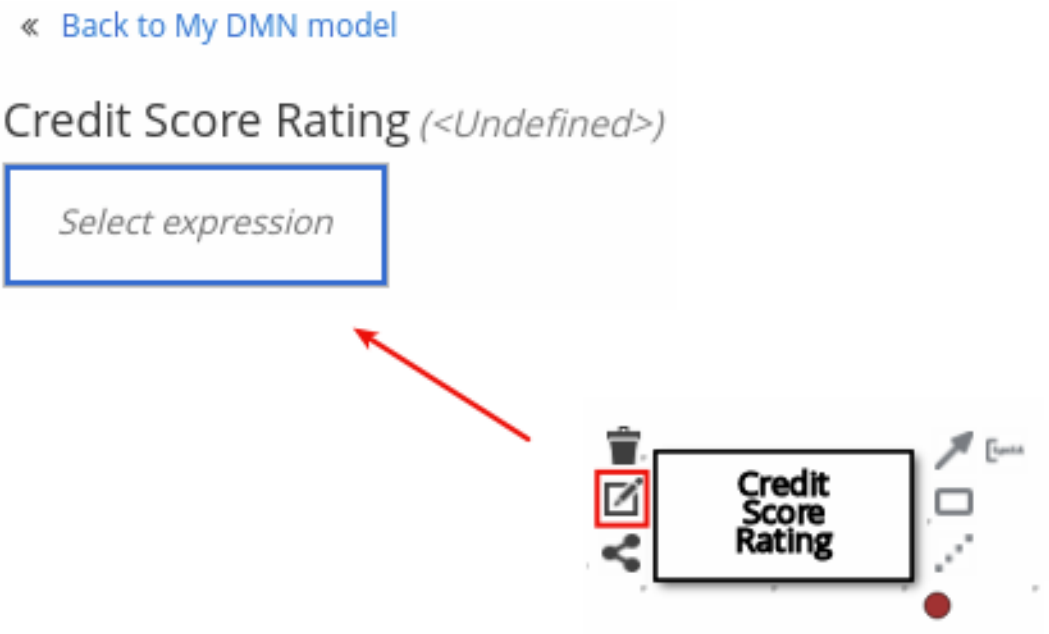
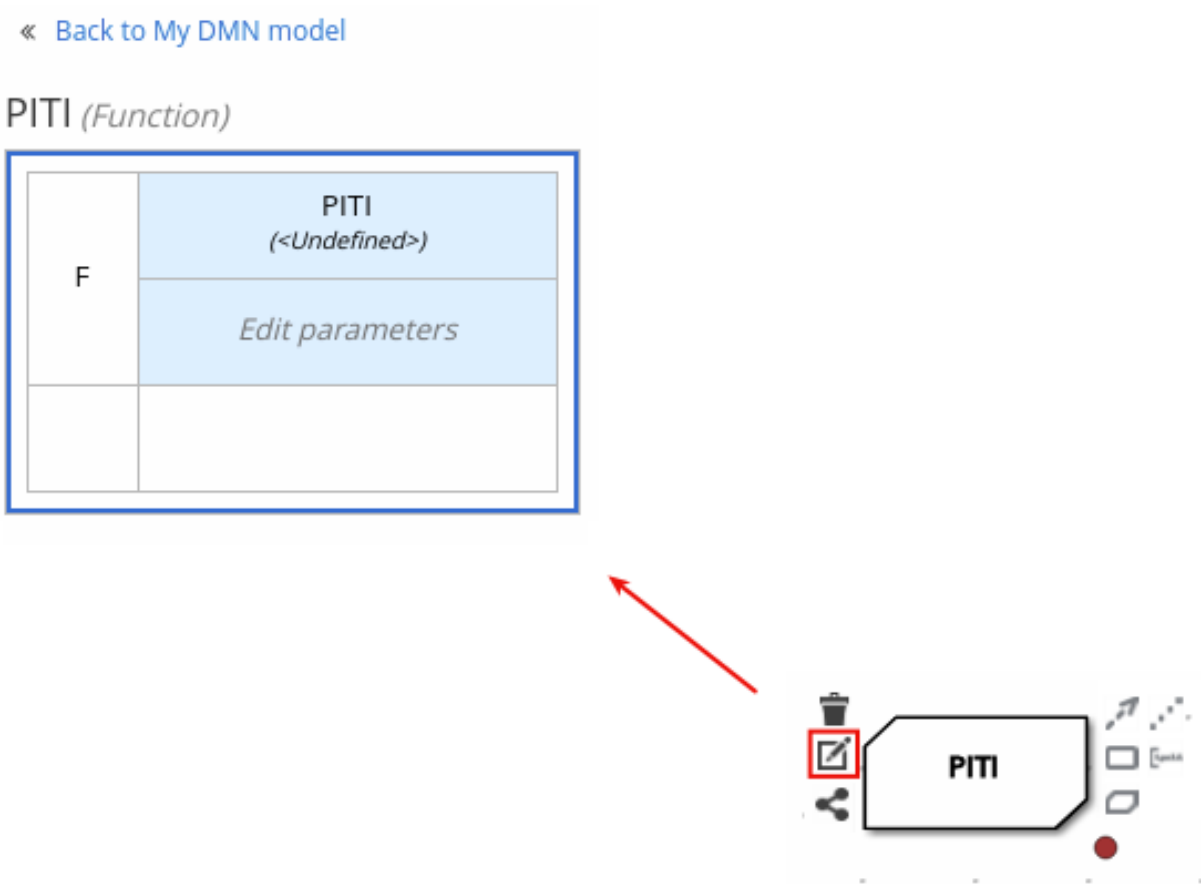


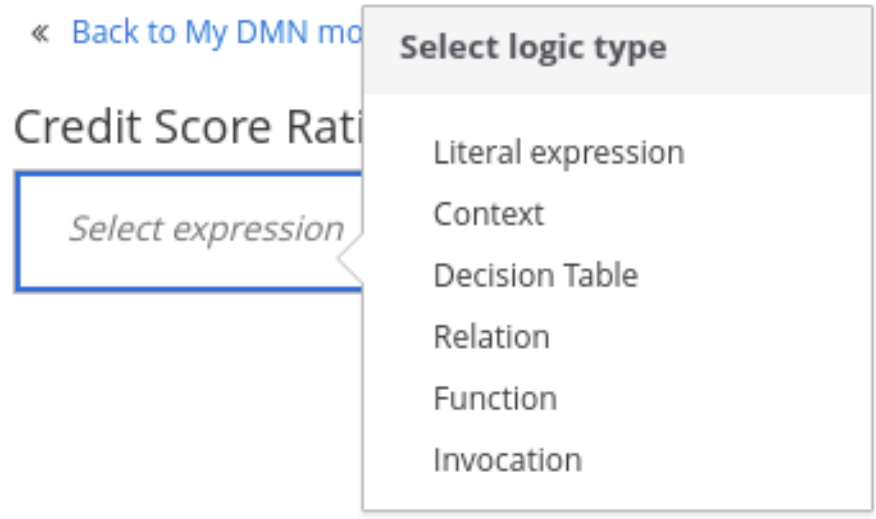
图 6.12. 打开新的商业知识模型已选中表达式



默认情况下，所有商业知识模型都被定义为带有字面意义的 FEEL 表达式、外部 JAVA 或 PMML 函数的嵌套上下文表达式，或任何类型的嵌套式表达式。

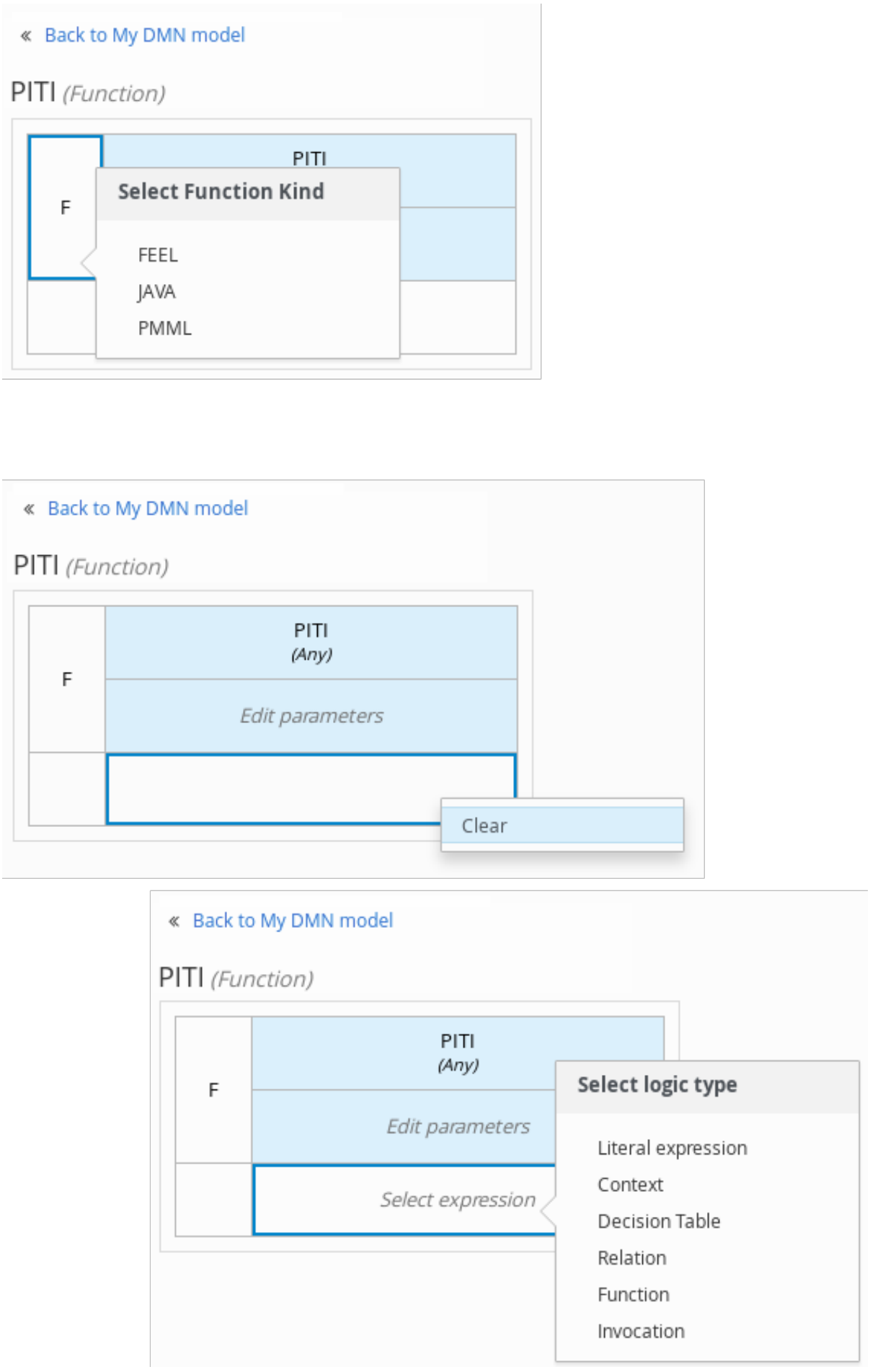
对于决策节点，您可以点击未定义表来选择要使用的已框表达式类型，如方框式的字面表达式、方框式上下文表达式、决策表或其他 DMN 框的表达式。

图 6.13. 为决策节点选择逻辑类型



对于业务知识模型节点，您单击左上角的功能单元以选择功能类型，或者右键单击功能价值单元，选择 Clear，然后选择另一个类型的方框。

图 6.14. 为业务知识模型选择功能或其他逻辑类型



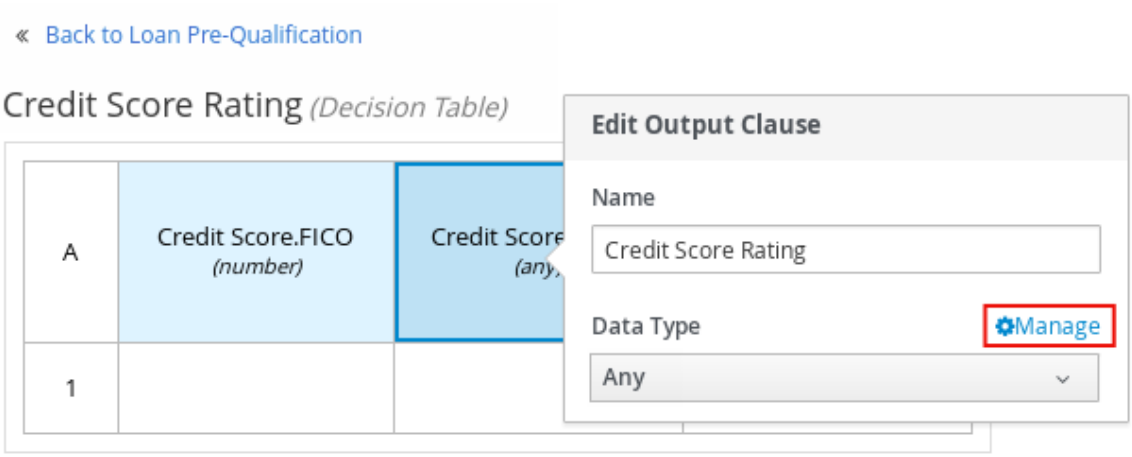
3. 在本例中，使用决策节点并选择 **Decision Table** 作为已选的表达式类型。

DMN 中的图标表是按表格格式显示一个或多个规则的可视化表示。每个规则均由表中的一行组成，包括针对该特定行定义条件（输入）和结果（输出）的列。

4. 单击 **input** 列标题，以定义输入条件的名称和数据类型。例如，将输入列命名为 **Inventory Score.FICO**，数字 数据类型。此列指定数字得分值或贷款的范围。

5. 单击 **output** 列标题，以定义输出值的名称和数据类型。例如，将输出列命名为 **Score Rating and the Data Type** 选项，单击 **Manage** 来转至 **Data Types** 页面，您可以在其中创建一个分数评级为约束。

图 6.15. 管理列标题值的数据类型

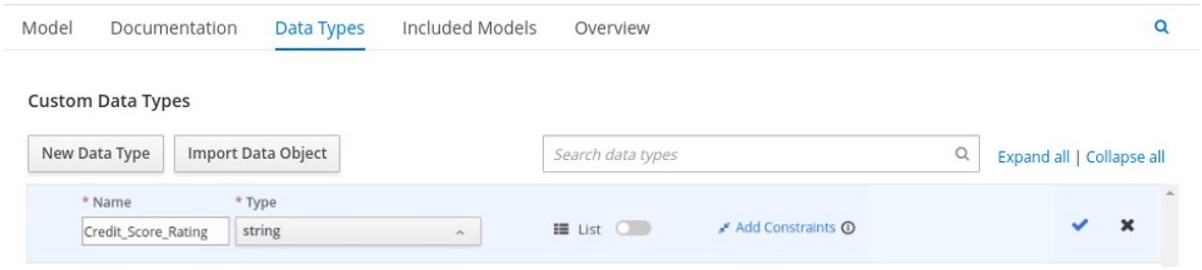


6. 在 **Data Types** 页面上，点 **New Data Type** 添加新数据类型或点击 **Import Data Object** 从您要用作 **DMN** 数据类型的项目中导入现有数据对象。

如果您从项目中导入数据对象作为 **DMN** 数据类型，然后更新该对象，您必须重新导入数据对象作为 **DMN** 数据类型，以在 **DMN** 模型中应用更改。

在本例中，单击 **New Data Type** 并创建 **credit _Score_R ating** 数据类型 作为字符串 ：

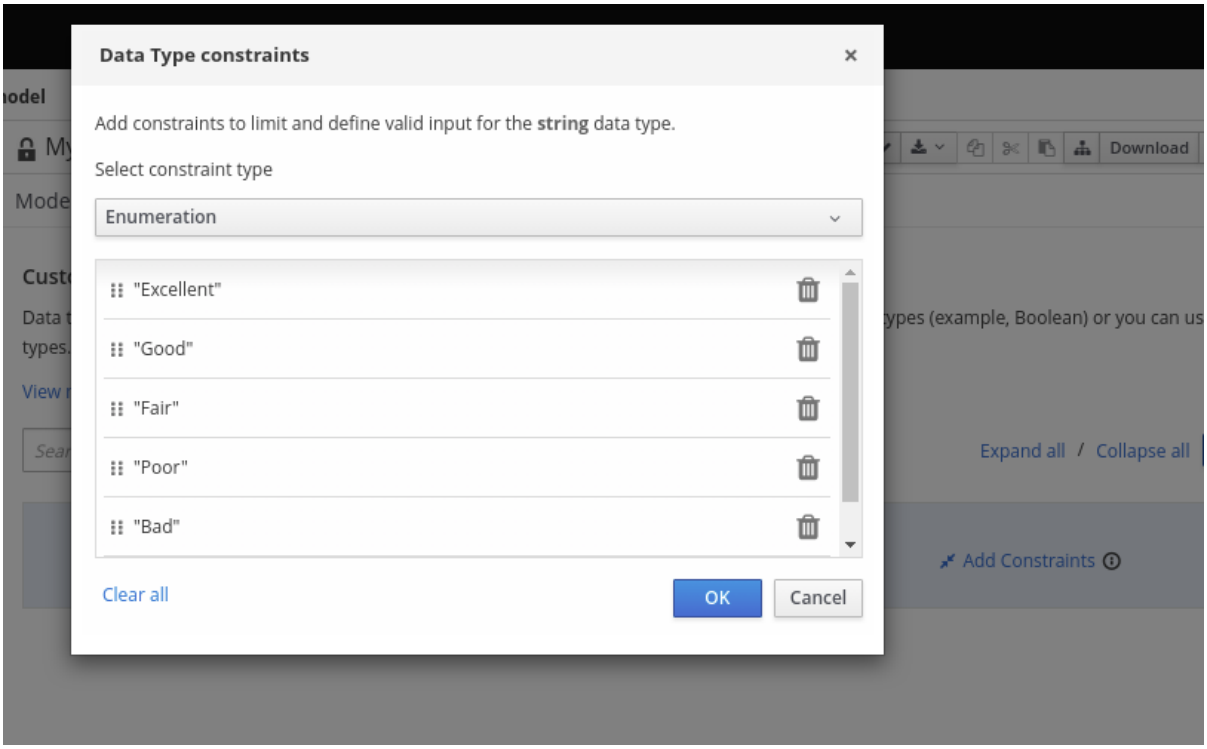
图 6.16. 添加新数据类型



7. 点 Add Constraints，从下拉列表中选择 Enumeration，并添加以下限制：

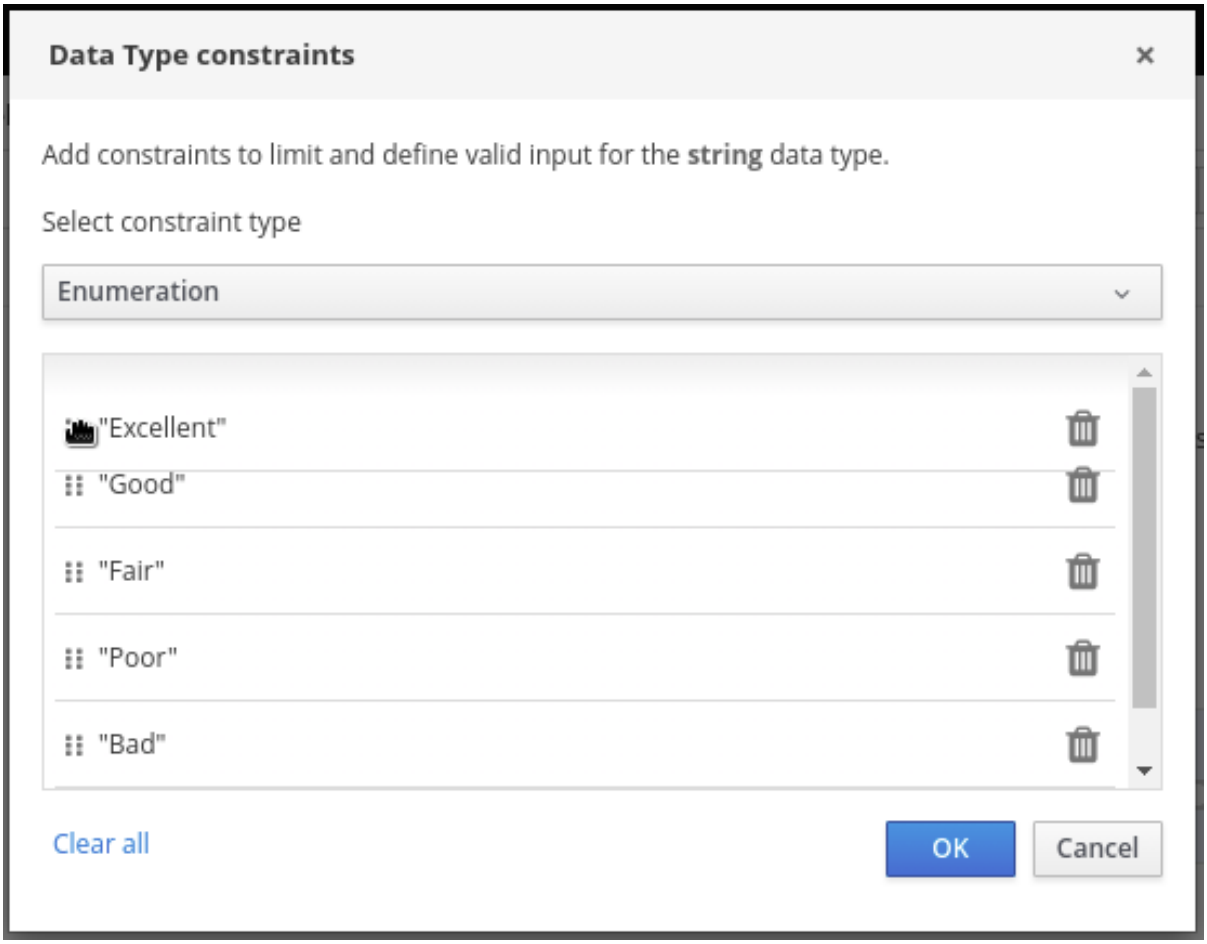
- "Excellent"
- "Good"
- "Fair"
- "Poor"
- "bad"

图 6.17. 在新数据类型中添加限制



要更改数据类型约束顺序，您可以点击约束行的左侧，并根据需要拖动行：

图 6.18. 拖动约束以更改约束顺序



有关指定数据类型的约束类型和语法要求的详情，请查看 Decision Model 和 Notation 规格。

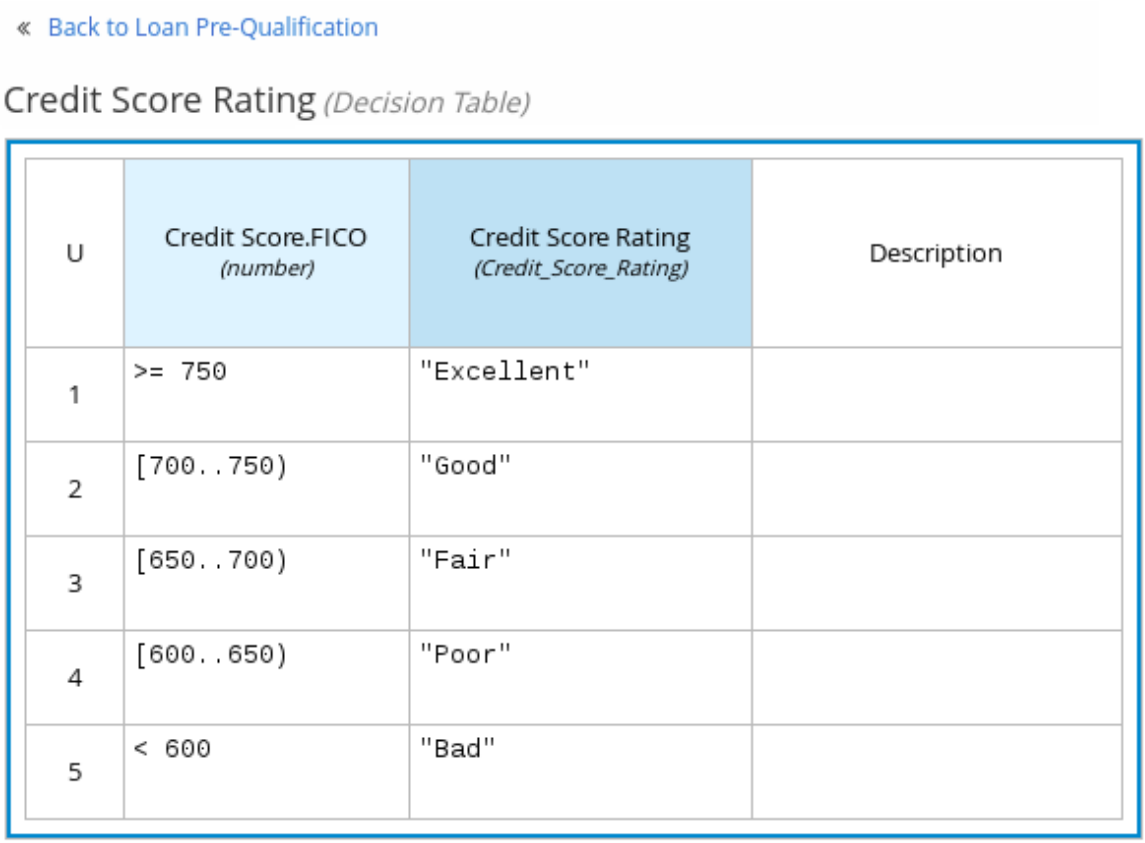
8. 点 OK 保存约束，然后点数据类型右侧的检查标记保存数据类型。

9. 返回到 credit Score Rating 决策表，单击 credit Score Rating 列标题，并将数据类型设置为这个新的自定义数据类型。

10. 使用 credit Score.FICO 输入列来定义贡献分数或值范围，并使用 credit Score Rating 列指定在 credit _Score_Rating 数据类型中定义的相应等级 之一。

右键点击任意值单元插入或删除行(rules)或列(clauses)。

图 6.19. 得分评级的决策节点决策表



« Back to Loan Pre-Qualification

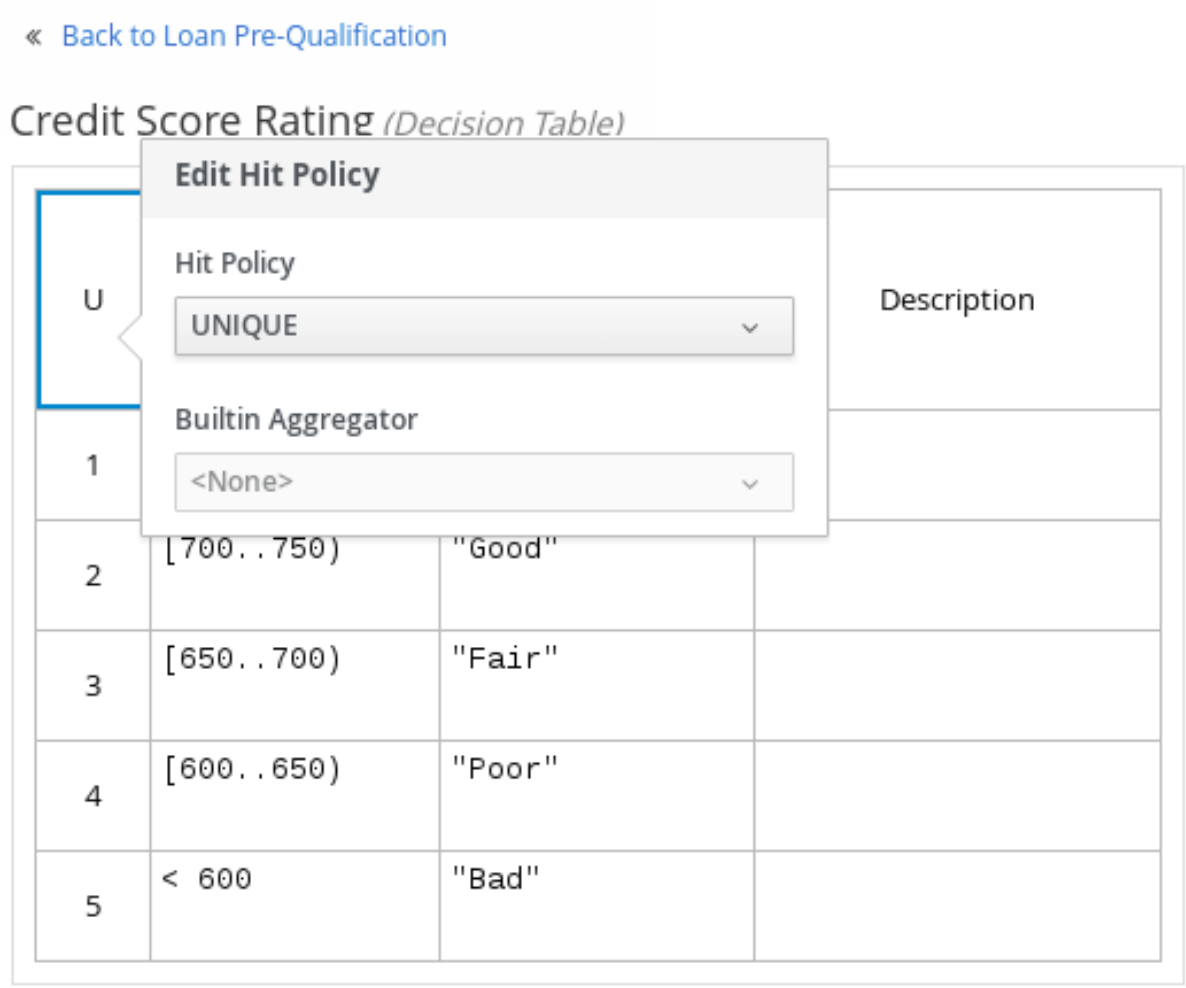
Credit Score Rating *(Decision Table)*

U	Credit Score.FICO *(number)*	Credit Score Rating *(Credit_Score_Rating)*	Description
1	>= 750	"Excellent"	
2	[700..750)	"Good"	
3	[650..700)	"Fair"	
4	[600..650)	"Poor"	
5	< 600	"Bad"	

11. **在定义所有规则后，单击决策表的左上角，以定义规则 Hit Policy 和 Builtin Aggregator（仅限 COLLECT hit 策略）。**

hit policy 决定在决定表中的多个规则与提供的输入值匹配时如何到达结果。内置的聚合器决定如何使用 COLLECT hit 策略来聚合规则值。

图 6.20. 定义决策表点击策略



以下示例是更加复杂的决策表，该表格决定 loans 金级资格，作为同一 loan prequalification 决策模型中包括的决定节点：

图 6.21. loan prequalification 的决策表

Loan Pre-Qualification *(Decision Table)*

F	Credit Score Rating *(Credit_Score_Rating)*	Back End Ratio *(Back_End_Ratio)*	Front End Ratio *(Front_End_Ratio)*	Loan Pre-Qualification *(Loan_Qualification)*		Description
				Qualification *(string)*	Reason *(string)*	
1	"Poor", "Bad"	-	-	"Not Qualified"	"Credit Score too low."	
2	-	"Insufficient"	"Sufficient"	"Not Qualified"	"Debt to income ratio is too high."	
3	-	"Sufficient"	"Insufficient"	"Not Qualified"	"Mortgage payment to income ratio is too high."	
4	-	"Insufficient"	"Insufficient"	"Not Qualified"	"Debt to income ratio is too high AND mortgage payment to income ratio is too high."	
5	"Fair", "Good", "Excellent"	"Sufficient"	"Sufficient"	"Qualified"	"The borrower has been successfully prequalified for the requested loan."	

对于决策表以外的新式表达式类型，您需要遵循这些指南来导航框的表达式表并为决策逻辑定义变量和参数，但根据已框的表达式类型的要求。一些方框式表达式（如方框的字面表达式）可以是单列表，而

其他已框的表达式（如功能、上下文和调用表达式）可以是带有其他类型的嵌套式表达式的多列表。

例如，以下方框上下文表达式定义了参数，它们根据主体、兴趣、税收和保险(PITI)，它确定 loan applicant 是否可以满足最低制裁支付费用：

图 6.22. 用于前端客户端 PITI 比率的方框上下文表达式

Front End Ratio *(Context)*

#	Front End Ratio *(Front_End_Ratio)*			
1	Client PITI *(number)*	#	PITI	
		1	pmt *(number)*	(Requested Product.Amount*((Requested Product.Rate/100)/12))/ (1-(1/(1+(Requested Product.Rate/100)/12)**-Requested Product.Term))
		2	tax *(number)*	Applicant Data.Monthly.Tax
		3	insurance *(number)*	Applicant Data.Monthly.Insurance
		4	income *(number)*	Applicant Data.Monthly.Income
	<result>	if Client PITI <= Lender Acceptable PITI() then "Sufficient" else "Insufficient"		

以下方框式函数表达式确定一个月的分流安装，作为商业知识模型确定在方便的决策中，该函数值定义为嵌套上下文表达式：

图 6.23. 在商业知识模式下安装计算的已选式功能表达式

InstallmentCalculation *(Function)*

F	InstallmentCalculation *(number)*		
	(ProductType, Rate, Term, Amount)		
	1	MonthlyFee *(number)*	if ProductType ="STANDARD LOAN" then 20.00 else if ProductType ="SPECIAL LOAN" then 25.00 else null
	2	MonthlyRepayment *(number)*	(Amount *Rate/12) / (1 - (1 + Rate/12)**-Term)
		<result>	MonthlyRepayment+MonthlyFee

有关每个已选表达式类型的信息和示例，请参阅 第 4.4 节 “框表达式中的 DMN 决策逻辑”。

6.2. 在 BUSINESS CENTRAL 中为 DMN 框表达式创建自定义数据类型

在 Business Central 中的 DMN 框表达式中，数据类型决定在相关表格、列内或已框表达式中使用的数据的结构。您可以使用默认 DMN 数据类型（如 String、Number，布尔值）或创建自定义数据类型来指定要为框式表达式值实施的其他字段和约束。

为已框表达式创建的自定义数据类型可以是简单或结构化的：

- 简单 数据类型仅具有名称和类型分配。示例： Age（编号）
- 结构化 数据类型包含多个与父数据类型关联的字段。示例：单个类型 Person，包含 字段名称（字符串 ）、Age（编号 ）、电子邮件（字符串）。

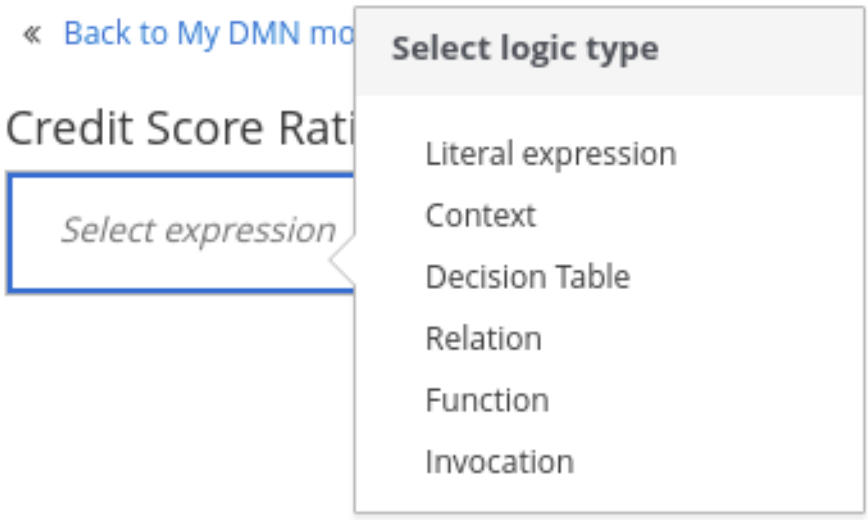
先决条件

- 在 Business Central 中创建或导入 DMN 文件。

流程

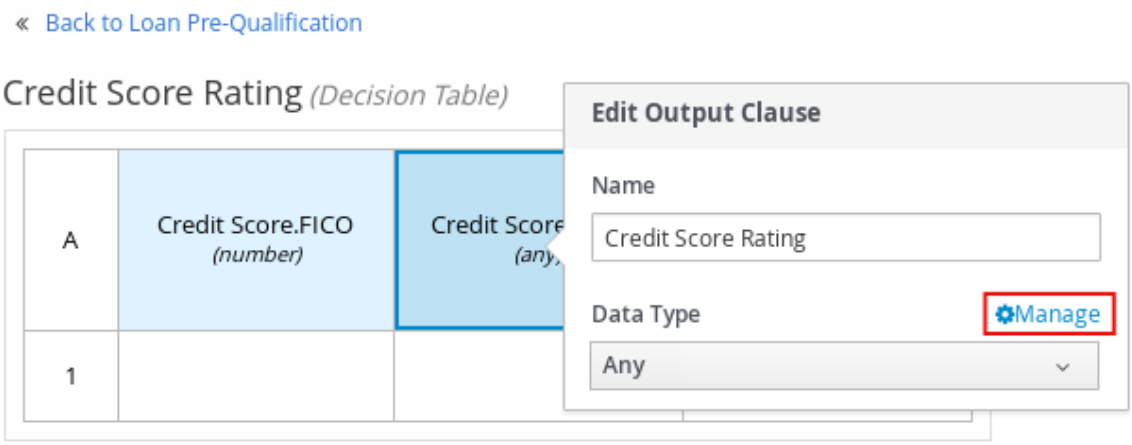
1. 在 Business Central 中，前往 Menu → Design → Projects，点项目名称并选择要修改的 DMN 文件。
2. 在 DMN 设计器 Canvas 中，选择您要定义数据类型的决策节点或商业知识模型，并点击 Edit 图标打开 DMN 框式表达式设计程序。
3. 如果已选中的表达式适用于尚未定义的决定节点，请单击 undefined 表来选择要使用的已选框表达式类型，如方框式的字面表达式、关闭上下文表达式、决策表或其他 DMN 框式表达式。

图 6.24. 为决策节点选择逻辑类型



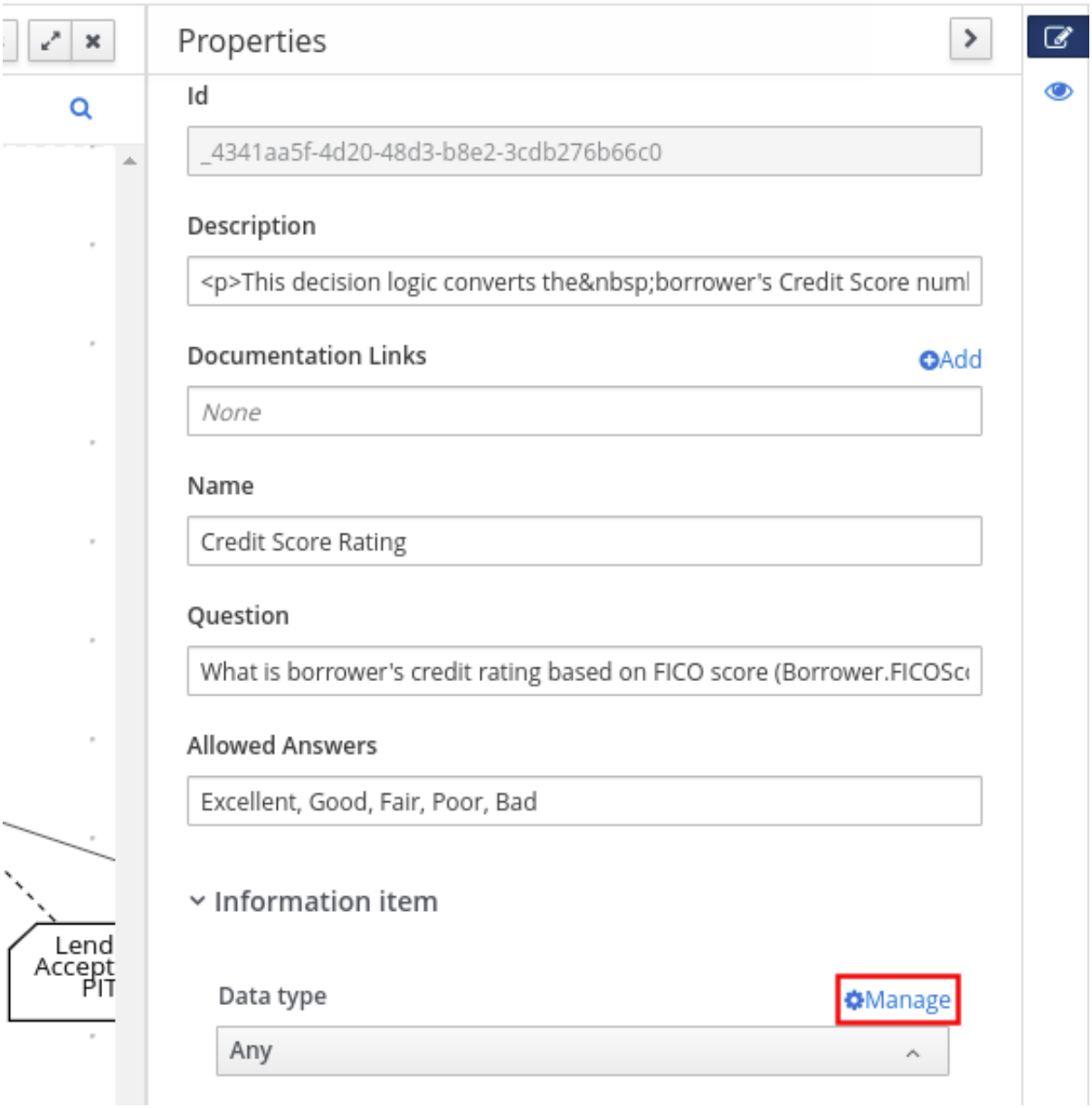
4. 点表标头、列标题或参数字段的单元（取决于您要定义数据类型的表达式类型），然后单击 **Manage** 以进入可以创建自定义数据类型的 **Data Types** 页面。

图 6.25. 管理列标题值的数据类型



您还可以通过选择 **DMN** 设计器右上角的 **Properties** 图标，为特定决策节点或业务知识模型节点设置和管理自定义数据类型：

图 6.26. 在决策需求图中管理数据类型(DRD)属性



您在 box 表达式中为指定单元定义的数据类型决定了您在该关联表、列或已框表达式中使用的数据的结构。

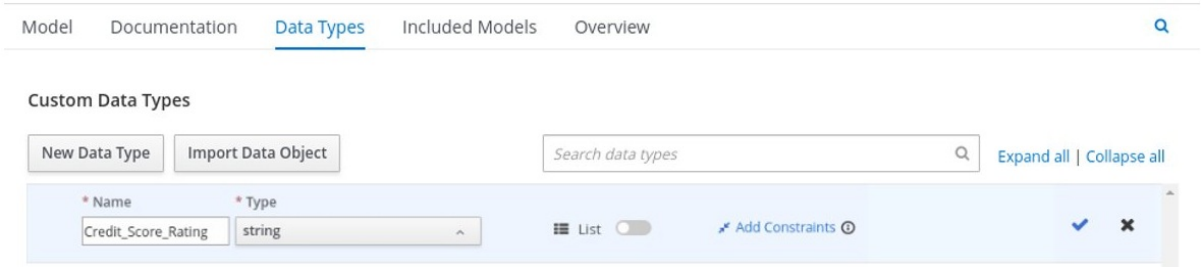
在这个示例中，DMN 决策表 的贡献列 Score Rating 定义了一组自定义分数等级，以以适用度的贡献分数为基础。

5. 在 Data Types 页面上，点 New Data Type 添加新数据类型或点击 Import Data Object 从您要用作 DMN 数据类型的项目中导入现有数据对象。

如果您从项目中导入数据对象作为 DMN 数据类型，然后更新该对象，您必须重新导入数据对象作为 DMN 数据类型，以在 DMN 模型中应用更改。

在本例中，单击 **New Data Type** 并创建 **credit _Score_R ating** 数据类型 作为字符串 ：

图 6.27. 添加新数据类型

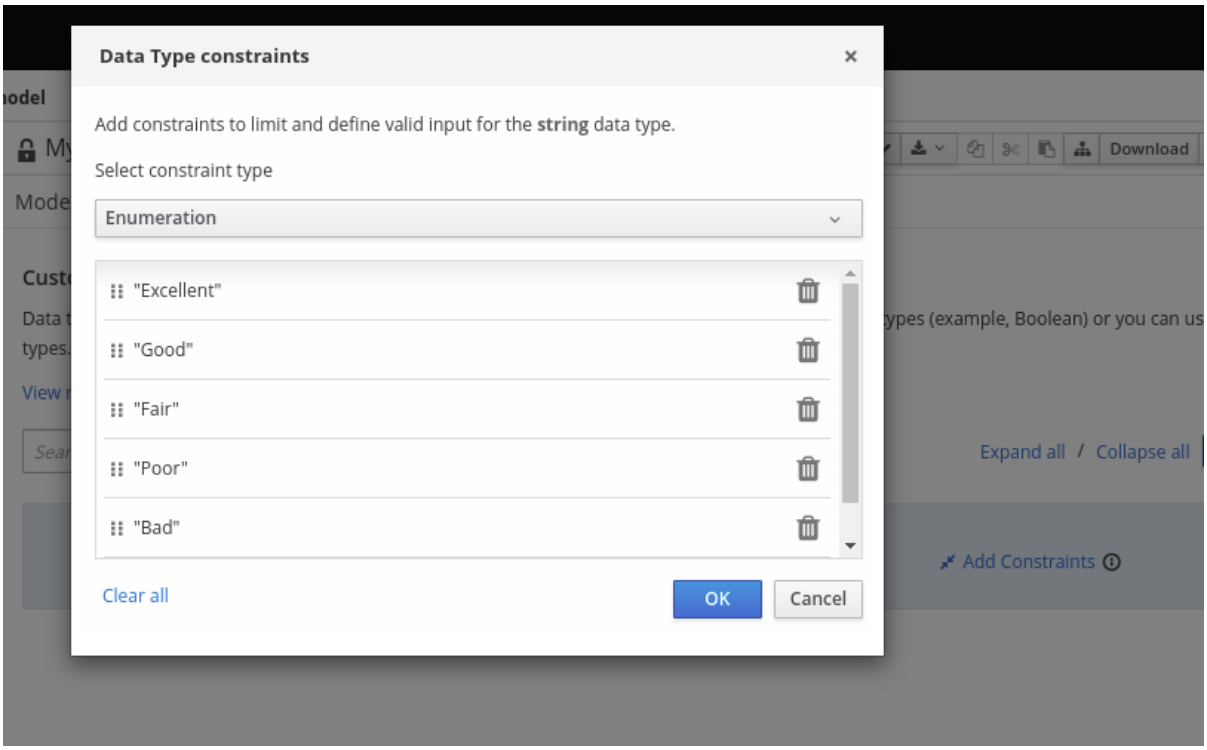


如果数据类型需要项目列表，请启用 **List** 设置。

6. 点 **Add Constraints**，从下拉列表中选择 **Enumeration**，并添加以下限制：

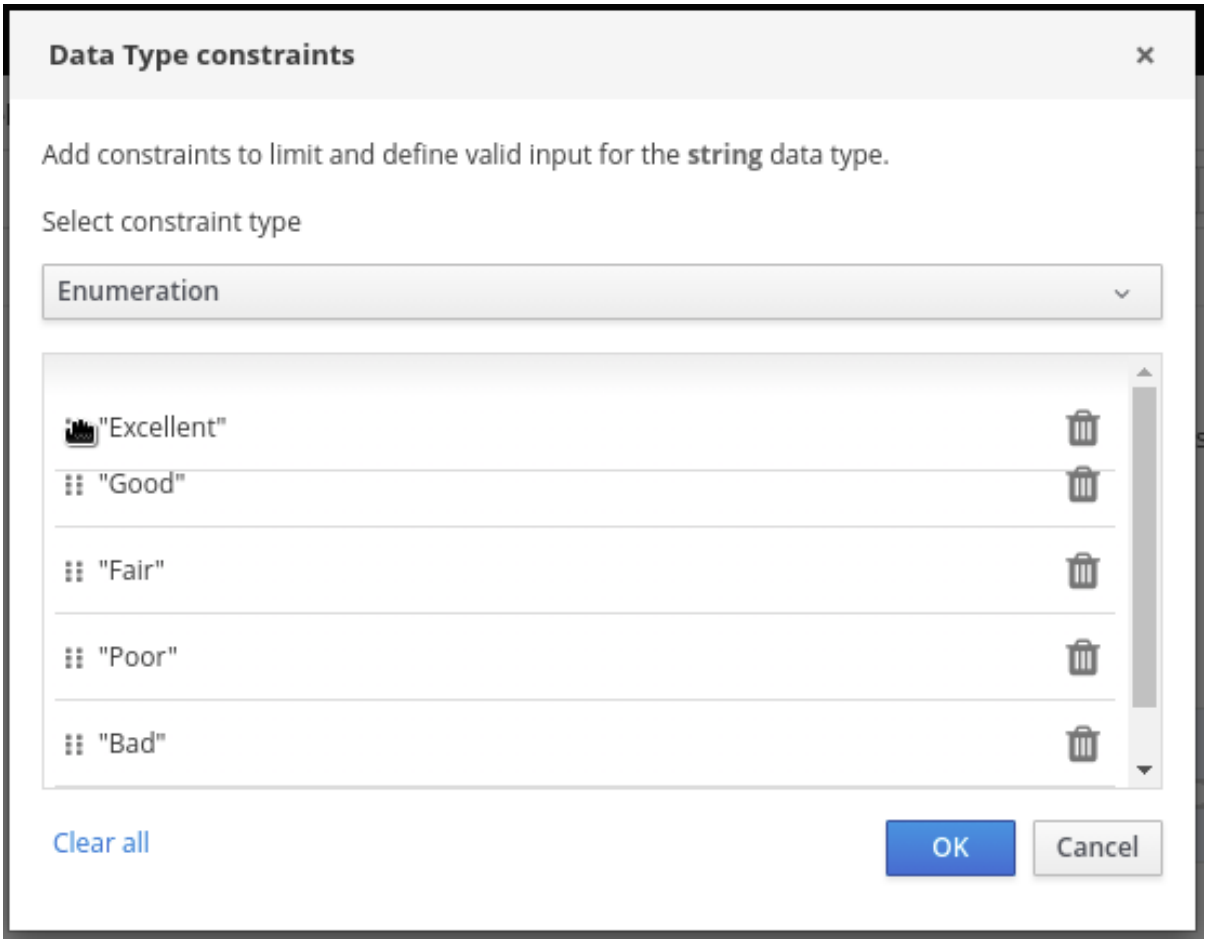
- "Excellent"
- "Good"
- "Fair"
- "Poor"
- "bad"

图 6.28. 在新数据类型中添加限制



要更改数据类型约束顺序，您可以点击约束行的左侧，并根据需要拖动行：

图 6.29. 拖动约束以更改约束顺序



有关指定数据类型的约束类型和语法要求的详情，请查看 Decision Model 和 Notation 规格。

7. 点 OK 保存约束，然后点数据类型右侧的检查标记保存数据类型。

8. 返回 贡献度级别 决策表，点 credit Score Rating 列标题，将数据类型设置为此新的自定义数据类型，并使用您指定的评级限制定义该列的规则值。

图 6.30. 得分评级的决策表

« Back to Loan Pre-Qualification

Credit Score Rating *(Decision Table)*

U	Credit Score.FICO *(number)*	Credit Score Rating *(Credit_Score_Rating)*	Description
1	>= 750	"Excellent"	
2	[700..750)	"Good"	
3	[650..700)	"Fair"	
4	[600..650)	"Poor"	
5	< 600	"Bad"	

在这种情况的 DMN 决策模型中，贡献性 决策流至以下需要自定义数据类型 的 Loan Pre qualification 决策：

图 6.31. loan prequalification 的决策表

Loan Pre-Qualification *(Decision Table)*

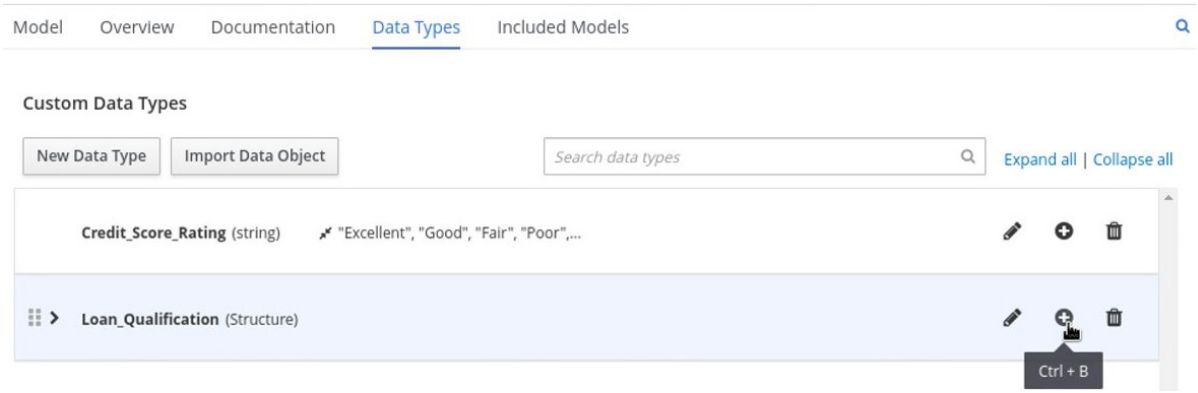
F	Credit Score Rating *(<Undefined>)*	Back End Ratio *(<Undefined>)*	Front End Ratio *(<Undefined>)*	Loan Pre-Qualification *(<Undefined>)*		Description
				Qualification *(string)*	Reason *(string)*	
1						

9. 继续本例，返回 Data Types 窗口，单击 New Data Type，然后创建一个 Loan_Qualification 数据类型作为没有限制的 Structure。

保存新的结构化数据类型时，会显示第一个子字段，以便您开始在此父数据类型中定义嵌套数据字段。您可以在框中的表达式中使用这些子字段与父结构化数据类型相关联，如嵌套列标题，或在上下文或功能表达式中嵌套表参数。

对于其他子字段，请选择 Loan_Qualification 数据类型旁边的添加图标：

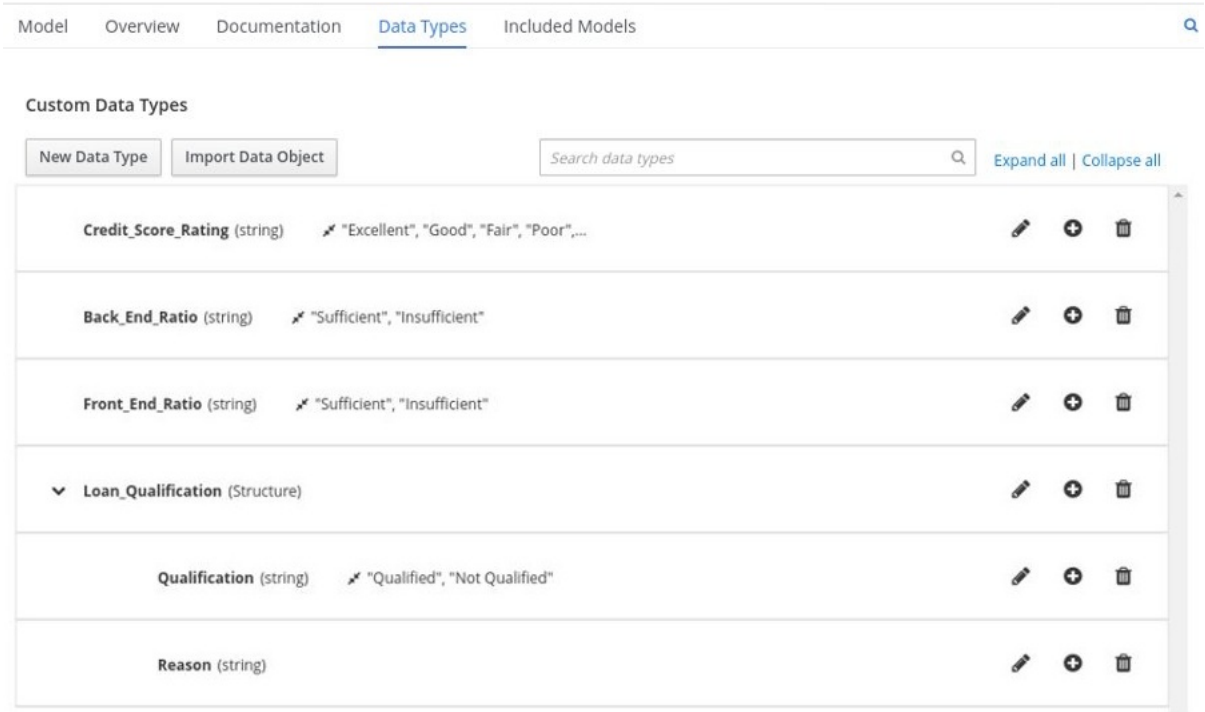
图 6.32. 使用嵌套字段添加新的结构化数据类型



10. 在本例中，在结构化 Loan_Qualification 数据类型下，添加一个带有 "Qualified" 和" Not Qualified" enumeration 约束的 Qualification 字段，以及没有限制的 Reason 字段。另外，添加简单的 Back_End_Ratio 和 Front_End_Ratio 数据类型（包括 "Sufficient" e Insufficient" enumeration 约束）。

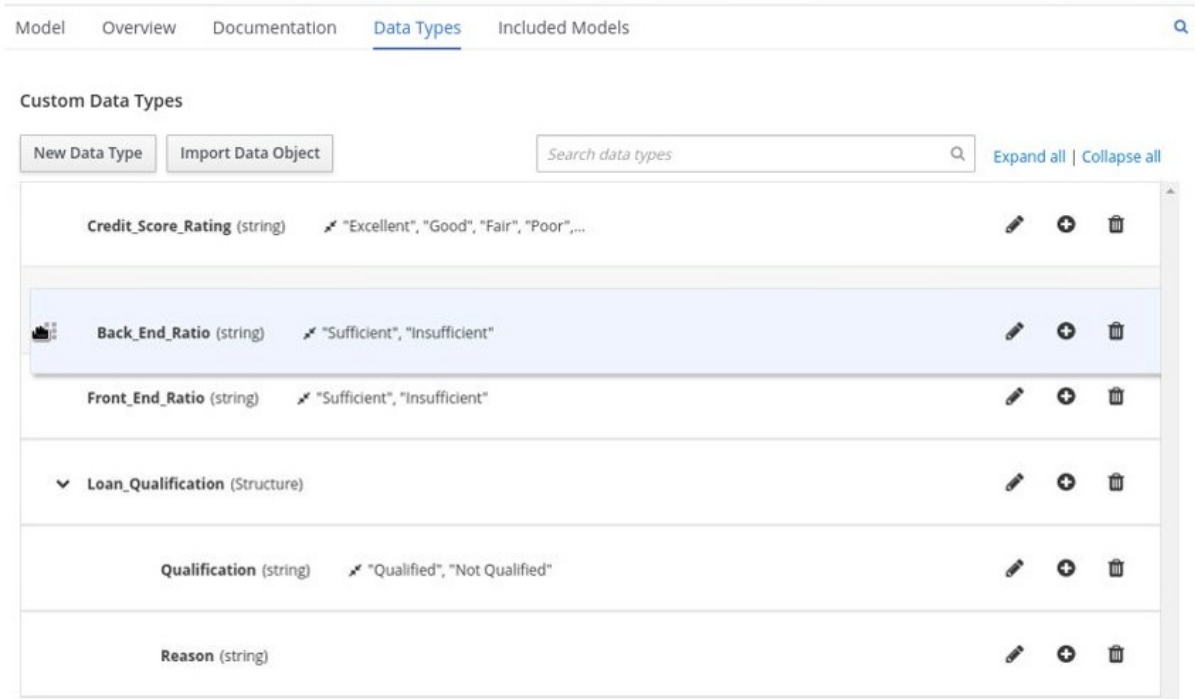
点击您创建的每种数据类型右侧的检查标记保存更改。

图 6.33. 使用限制添加嵌套数据类型



要更改数据类型的顺序或嵌套，您可以点击数据类型行的左侧，并根据需要拖动行：

图 6.34. 拖动数据类型以更改数据类型顺序或嵌套



11. 返回各个列的决策表，单击列标题单元格，将数据类型设置为新的对应的自定义数据类型，然后根据需要定义列所需的规则值，并提供您指定的限制（如果适用）。

图 6.35. loan prequalification 的决策表

Loan Pre-Qualification *(Decision Table)*

F	Credit Score Rating *(Credit_Score_Rating)*	Back End Ratio *(Back_End_Ratio)*	Front End Ratio *(Front_End_Ratio)*	Loan Pre-Qualification *(Loan_Qualification)*		Description
				Qualification *(string)*	Reason *(string)*	
1	"Poor", "Bad"	-	-	"Not Qualified"	"Credit Score too low."	
2	-	"Insufficient"	"Sufficient"	"Not Qualified"	"Debt to income ratio is too high."	
3	-	"Sufficient"	"Insufficient"	"Not Qualified"	"Mortgage payment to income ratio is too high."	
4	-	"Insufficient"	"Insufficient"	"Not Qualified"	"Debt to income ratio is too high AND mortgage payment to income ratio is too high."	
5	"Fair", "Good", "Excellent"	"Sufficient"	"Sufficient"	"Qualified"	"The borrower has been successfully prequalified for the requested loan."	

对于决策表以外的选择表达式类型，您需要遵循这些指南来导航框的表达式表并根据需要定义自定义数据类型。

例如，以下方框的函数表达式使用自定义 tCandidate 和 tProfile 结构化数据类型来关联数据进行在线持久性兼容性：

图 6.36. 用于在线持久性兼容性的 boxed 功能表达式

Evaluate Match *(Function)*

F	Evaluate Match *(tCandidate)*	
	(Lonely Soul, Candidate)	
1	Profile1 *(tProfile)*	Lonely Soul
2	Profile2 *(tProfile)*	Candidate
3	Is Match *(boolean)*	Is Soul a Match(Lonely Soul, Candidate) and Is Soul a Match(Candidate, Lonely Soul)
4	Score *(number)*	Number of Matching Interests(Lonely Soul, Candidate) - absolute(Lonely Soul.Age - Candidate.Age)
	<result>	*Select expression*

图 6.37. 用于在线持久性兼容性的自定义数据类型定义

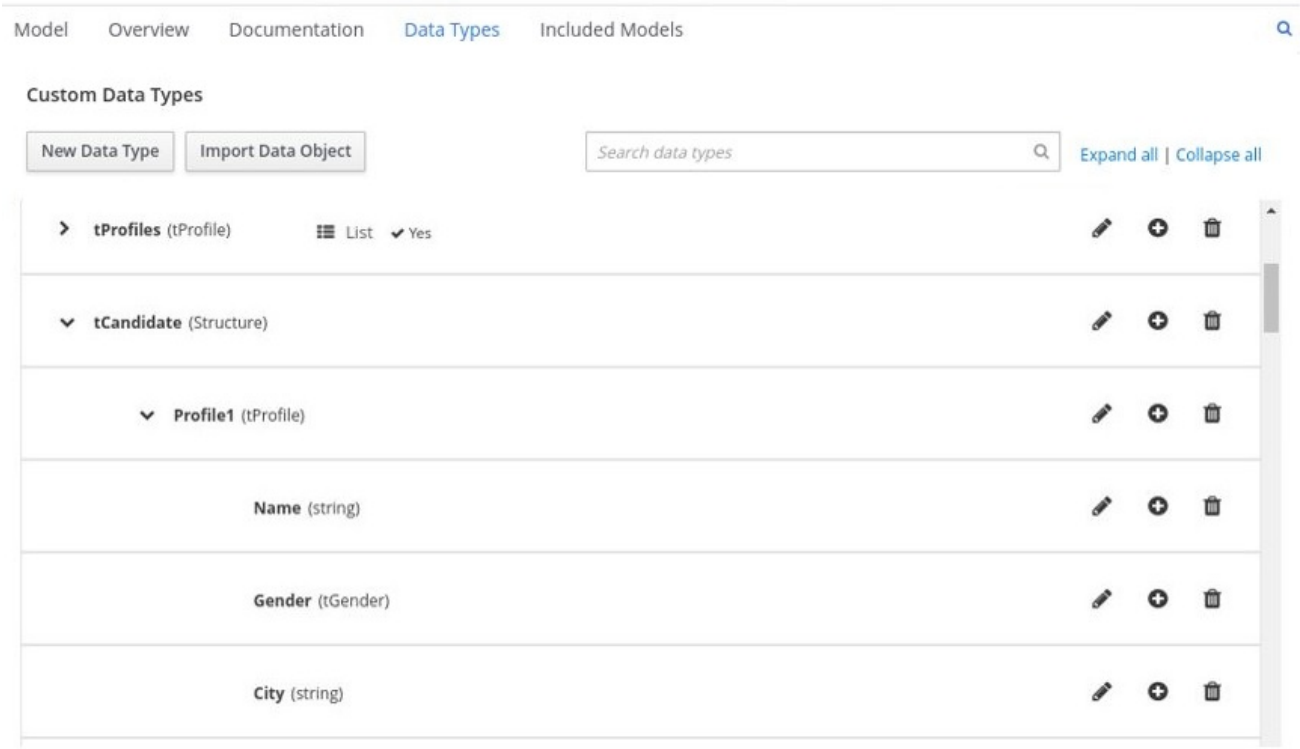
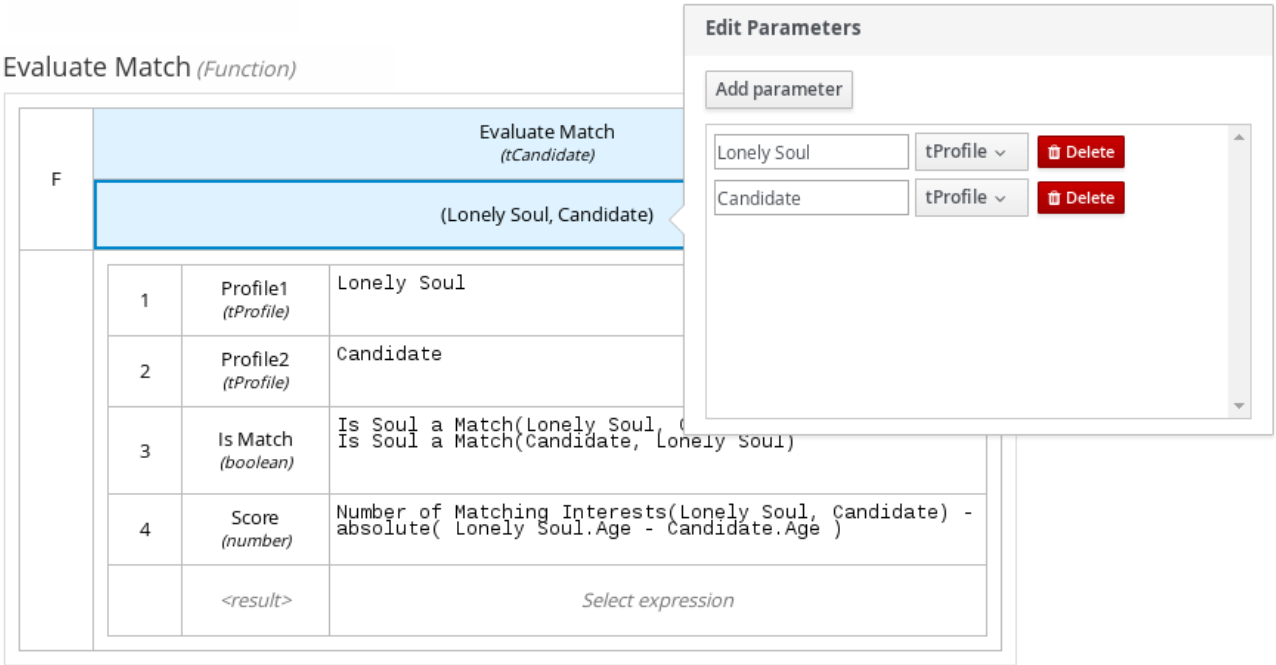


图 6.38. 具有用于在线持久性兼容性的自定义数据类型的参数定义



6.3. BUSINESS CENTRAL 中的 DMN 文件中包括模型

在 Business Central 中的 DMN 设计器中，您可以使用包含的 Models 选项卡在指定的 DMN 文件中包含其他 DMN 模型和预测模型标记语言(PMML)模型。当您在另一个 DMN 文件中包括 DMN 模型时，您可以在相同的决策要求图(DRD)中使用这两种模型的所有节点和逻辑。当您在 DMN 文件中包含 PMML 模型时，您可以将 PMML 模型作为作为 DMN 决策节点或商业知识节点的 box 功能表达式调用。

您不能在 Business Central 中包含来自其他项目的 DMN 或 PMML 型号。

6.3.1. 在 Business Central 的 DMN 文件中包括其他 DMN 模型

在 Business Central 中，您可以在指定的 DMN 文件中包含项目中的其他 DMN 模型。当您在另一个 DMN 文件中包括 DMN 模型时，您可以在相同的决策要求图(DRD)中使用这两个模型的所有节点和逻辑，但您无法从包含的模型中编辑节点。要从包含模型编辑节点，您必须直接为包含的模型更新源文件。如果您为包含的 DMN 模型更新源文件，打开包含 DMN 模型的 DMN 文件（或关闭一个重新打开）以验证更改。

您不能在 Business Central 中包含其他项目的 DMN 模型。

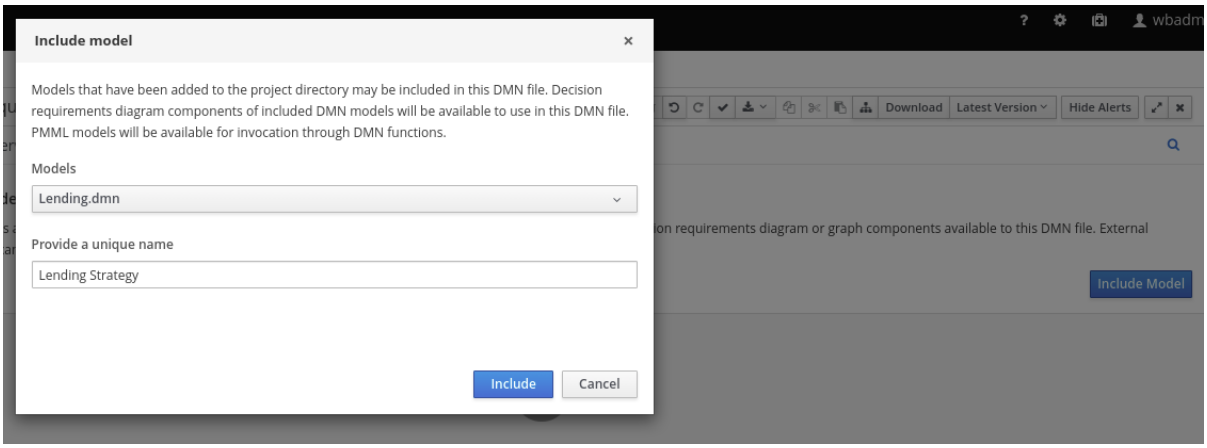
先决条件

- DMN 模型会在 Business Central 中创建或导入（作为 .dmn 文件）作为您要在其中包含型号的 DMN 文件。

流程

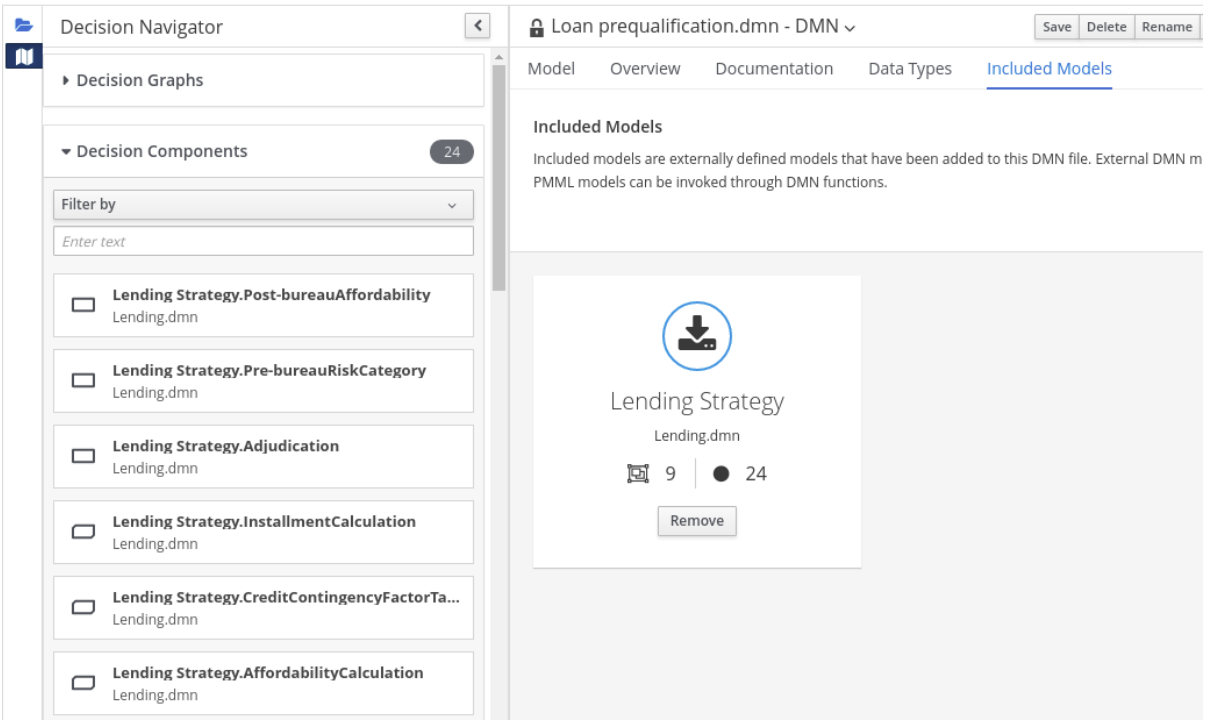
1. 在 Business Central 中，前往 Menu → Design → Projects，点项目名称并选择要修改的 DMN 文件。
2. 在 DMN 设计器中，点 包含的 Models 选项卡。
3. 点 Include Model，在 Models 列表中选择 DMN 模型，输入所含模型的唯一名称，然后点 Include:

图 6.39. 包括 DMN 模型



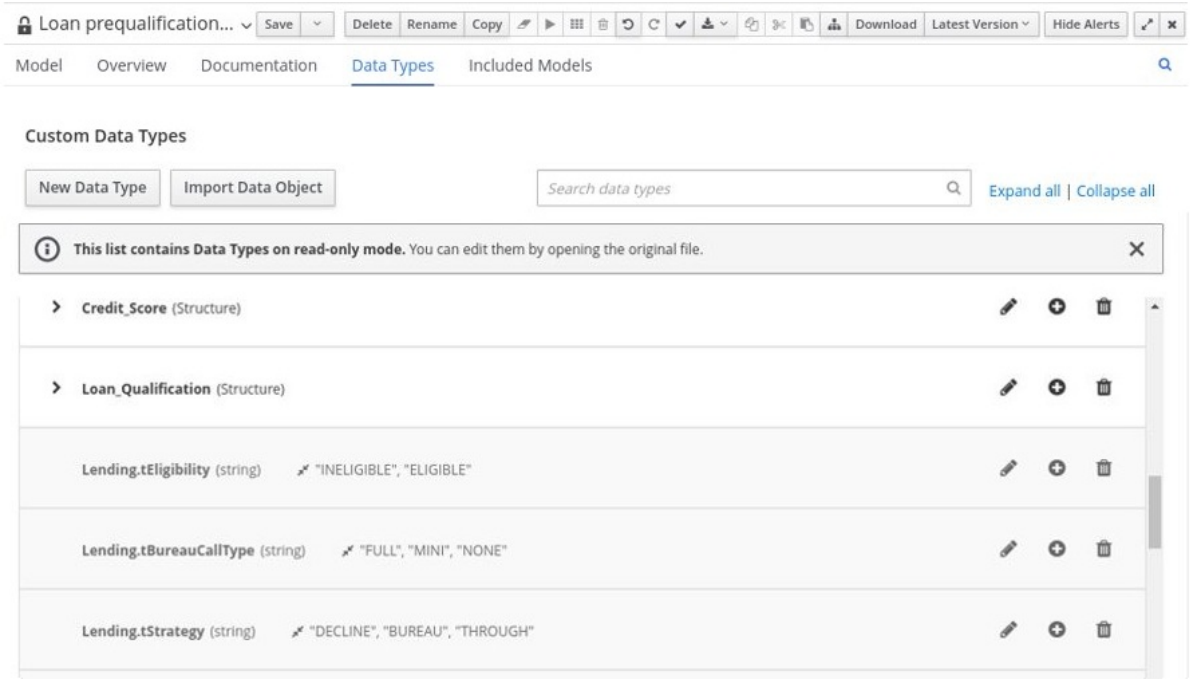
DMN 模型添加到这个 DMN 文件，在 Decision Navigator 视图中的 Decision 组件 下列出所有 DRD 节点：

图 6.40. 带有来自包含 DMN 模型中的决定组件的 DMN 文件



包含模型中的所有数据类型也列在 DMN 文件的 Data Types 标签页中的只读模式下：

图 6.41. 具有所含 DMN 模型中数据类型的 DMN 文件

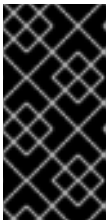


4. 在 DMN 设计器的 Model 选项卡中，点击并将包含的 DRD 组件拖到 canvas 中以开始在 DRD 中实现：

图 6.42. 从包括 DMN 模型中添加 DRD 组件

要编辑 DRD 节点或包含模型的数据类型，您必须直接为包含的模型更新源文件。如果您为包含的 DMN 模型更新源文件，打开包含 DMN 模型的 DMN 文件（或关闭一个重新打开）以验证更改。

要编辑包含的模型名称或从 DMN 文件中删除包含的模型，请使用 DMN 设计器中包括的 Models 选项卡。



重要

当您删除包含的模型时，来自当前在 DRD 中使用的模型中的任何节点也会被删除。

6.3.2. 在 Business Central 中的 DMN 文件中包括 PMML 型号

在 Business Central 中，您可以在指定 DMN 文件中的项目中包含预测模型标记语言(PMML)模型。当您在 DMN 文件中包含 PMML 模型时，您可以将 PMML 模型作为作为 DMN 决策节点或商业知识节点的 box 功能表达式调用。如果您更新包含 PMML 模型的源文件，则必须在 DMN 文件中删除并重新包含 PMML 模型，以应用源更改。

您不能在 Business Central 中包含其他项目的 PMML 型号。

先决条件

- **PMML 模式在 Business Central 所在的同一个项目中导入（作为 .pmml 文件），作为您要在其中包含该模型的 DMN 文件。**

流程

1. **在您的 DMN 项目中，将以下内容添加到项目 pom.xml 文件中，以启用 PMML 评估：**

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
  <scope>provided</scope>
</dependency>

<!-- Alternative dependencies for JPMML Evaluator, override `kie-pmml` dependency -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-jpmml</artifactId>
  <version>${rhpam.version}</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jpmml</groupId>
  <artifactId>pmml-evaluator</artifactId>
  <version>1.5.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.jpmml</groupId>
  <artifactId>pmml-evaluator-extension</artifactId>
  <version>1.5.1</version>
  <scope>provided</scope>
</dependency>
```

要访问 Business Central 中的项目 pom.xml 文件，您可以选择项目中任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

如果要使用带有 PMML(JPMML)的 Java Evaluator API 的完整 PMML 规格实现，请使用 DMN 项目中的替代 JPMML 依赖项集合。如果 JPMML 依赖项和标准 kie-pmml 依赖项都存在，则禁用 kie-pmml 依赖项。有关 JPMML 许可条款的详情，请参考 Openscoring.io。

重要

传统的 kie-pmml 依赖项在 Red Hat Process Automation Manager 7.10.0 中已弃用，并将在以后的 Red Hat Process Automation Manager 发行版本中被 kie-pmml-trusty 依赖项替代。

注意

考虑将 Red Hat Business Automation Manager (BOM)依赖项添加到项目 pom.xml 文件的 dependencies Management 部分，而不是为单个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

2. 如果您在 DMN 项目中添加了 JPMML 依赖项以使用 JPMML Evaluator，请下载以下 JAR 文件并将它们添加到 ~/kie-server.war/WEB-INF/lib 和 ~/business-central.war/WEB-INF/lib 目录：

- Red Hat Process Automation Manager 7.13.5 Maven Repository distribution 中的 kie-dmn-jpmml JAR 文件(rhpam-7.13.5-maven-repository/maven-repository/org/kie/kie-dmn-jpmml/7.67.0.Final-redhat-00024/kie-dmn-jpmml-7.67.0.Final-redhat-00024.jar)
  https://access.redhat.com/jbossnetwork/restricted/listSoftware.html?downloadType=distributions&product=rhpam&version=7.13.5

- JPMML Evaluator 1.5.1 JAR 文件来自在线 Maven 存储库

- JPMML Evaluator Extensions 1.5.1 JAR 文件来自在线 Maven 存储库

工件需要在 KIE 服务器和 Business Central 中启用 JPMML 评估。

重要

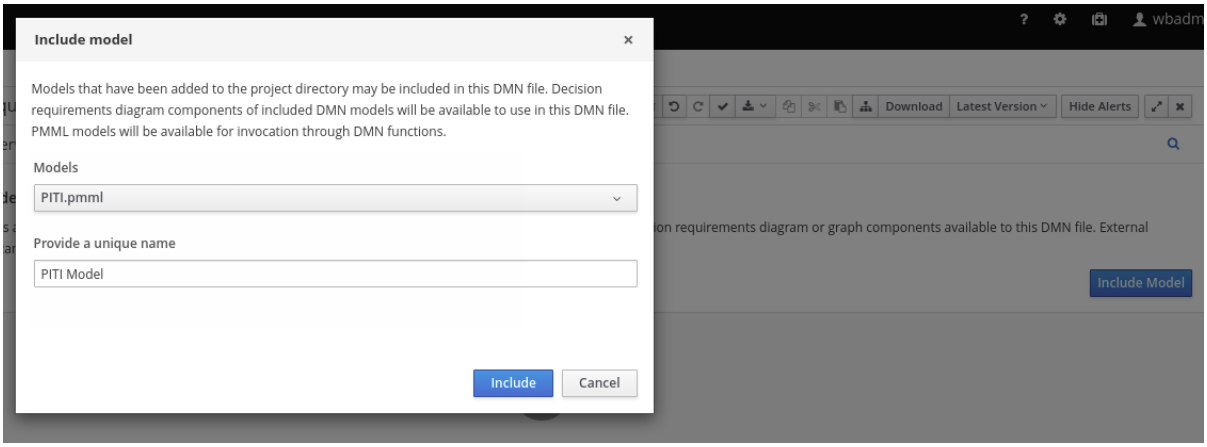
红帽支持与 PMML(JPMML)的 Java Evaluator API 集成，以便在 Red Hat Process Automation Manager 中进行 PMML 执行。但是，红帽不支持 JPMML 库。如果您在 Red Hat Process Automation Manager 发行版本中包括 JPMML 库，请参阅 JPMML 的 Openscoring.io 许可条款。

3. 在 Business Central 中，前往 Menu → Design → Projects，点项目名称并选择要修改的 DMN 文件。

4. 在 DMN 设计器中，点 包含的 Models 选项卡。

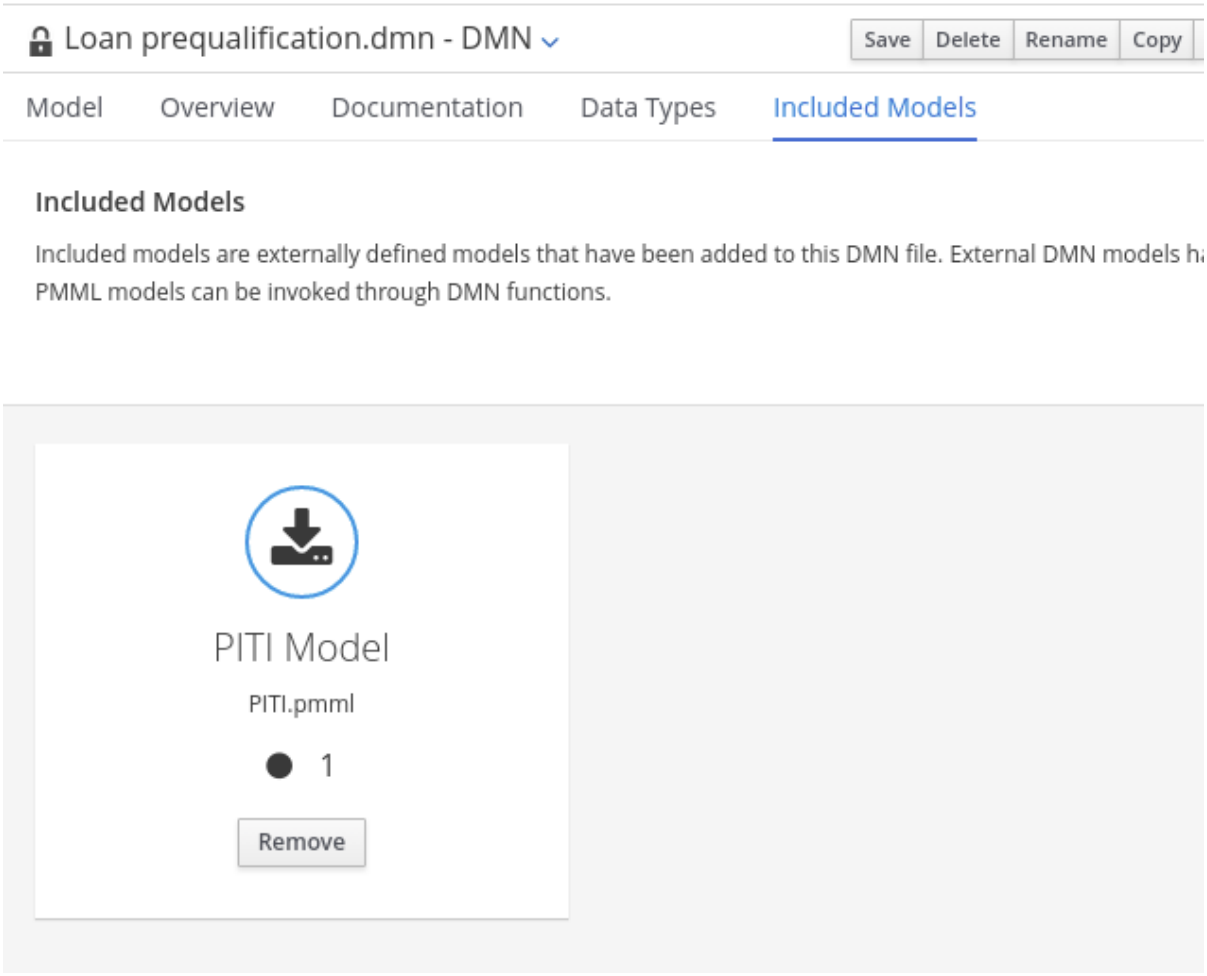
5. 点 Include Model，在 Models 列表中选择 PMML model，输入所含模型的唯一名称，然后点 Include:

图 6.43. 包括 PMML 模型



PMML 模型添加到这个 DMN 文件中：

图 6.44. 包含 PMML 模型的 DMN 文件



6. **在 DMN 设计器的 Model 选项卡中，选择或创建您要调用 PMML 模型的决策节点或业务知识模型节点，并点击 Edit 图标打开 DMN 框的表达式设计器：**

图 6.45. 打开新决策节点复选框表达式

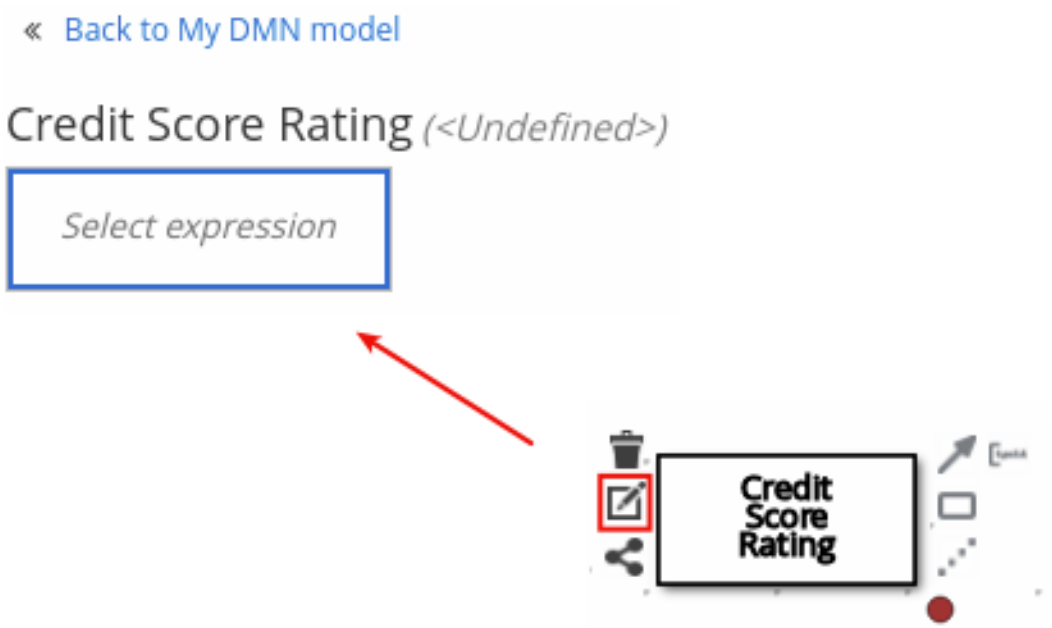
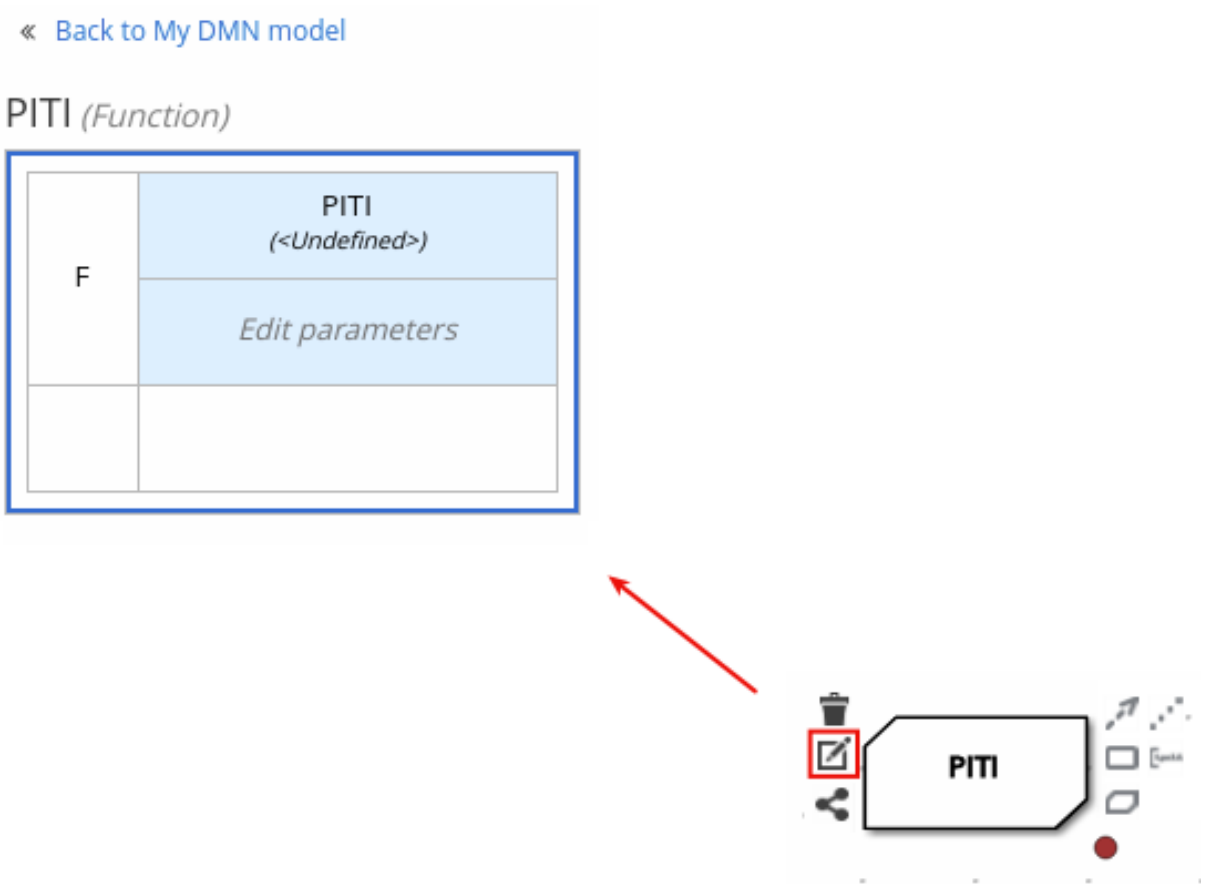


图 6.46. 打开新的商业知识模型已选中表达式



7. **将表达式类型设置为 Function （业务知识模型节点的默认），单击左上角的功能单元，然后选择 PMML。**

8. **在表中的文档和 型号 行中，双击未定义单元格，以指定其包含的 PMML 文档以及该文件中的相关 PMML 模型：**

图 6.47. 在 DMN 商业知识模型中添加 PMML 模型

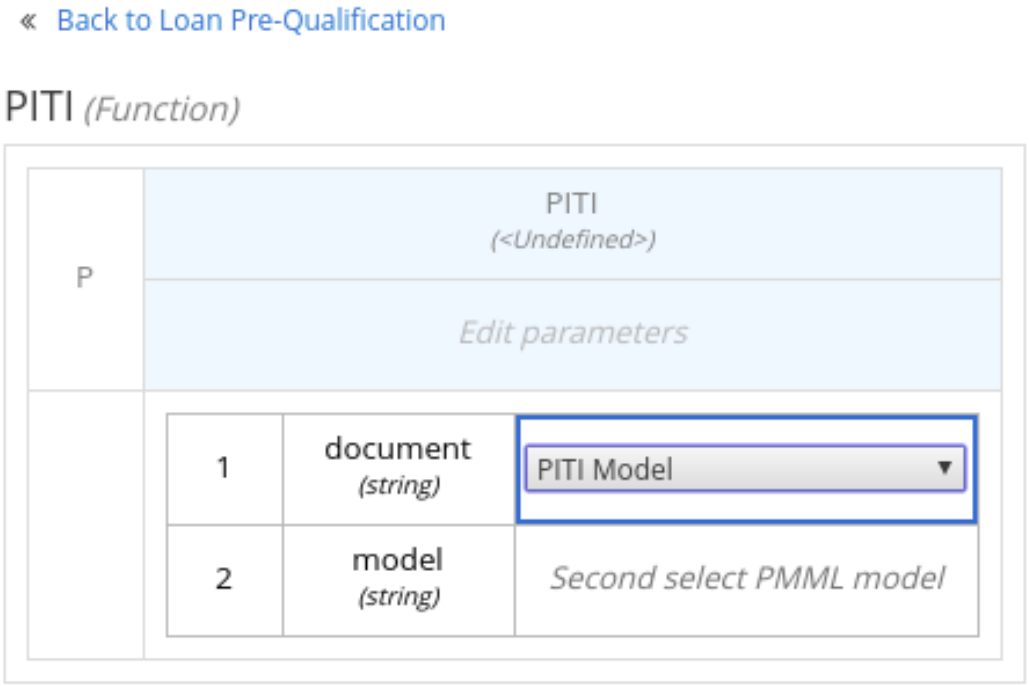
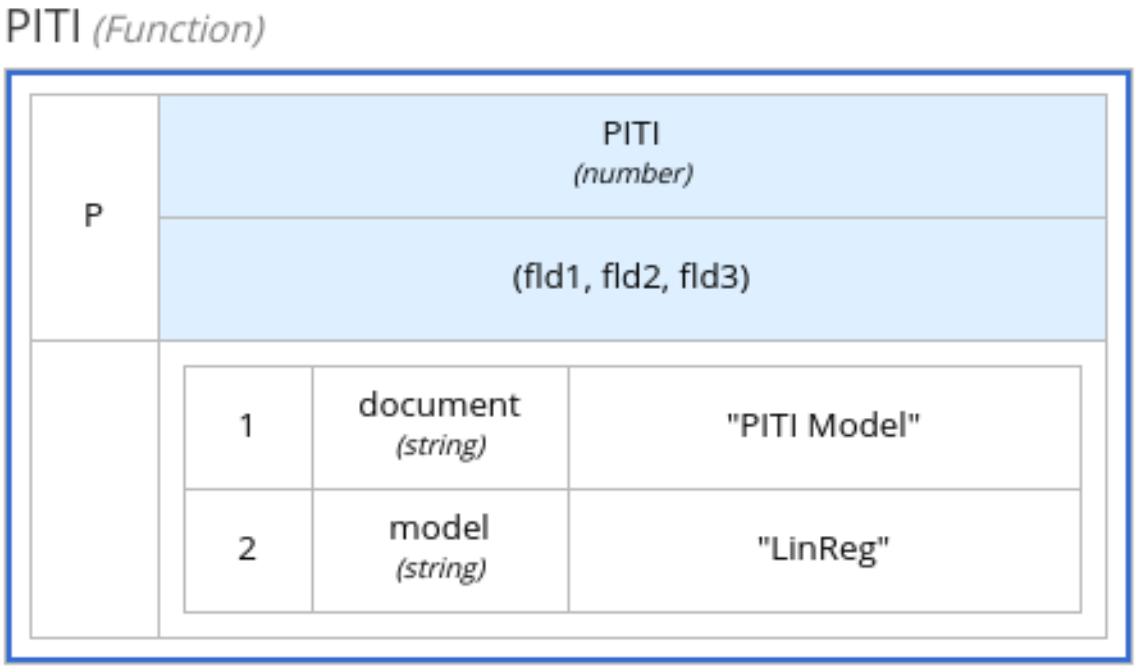


图 6.48. DMN 商业知识模型中的 PMML 定义示例



如果您更新包含 PMML 模型的源文件，则必须在 DMN 文件中删除并重新包含 PMML 模型，以应用源更改。

要编辑包含的模型名称或从 DMN 文件中删除包含的模型，请使用 DMN 设计器中包括的 Models 选项卡。

6.4. 在 BUSINESS CENTRAL 中创建带有多个图表的 DMN 模型

对于复杂的 DMN 模型，您可以使用 Business Central 中的 DMN 设计器设计多个 DMN 决策要求图(DRD)，这些图代表 DMN 决策模型的整体决策要求图(DRG)。在简单情况下，您可以使用单个 DRD 代表决策模型的所有总体 DRG，但在复杂情况下，单个 DRD 可能会变得非常困难。因此，为更好地组织具有许多决策要求的 DMN 决策模型，您可以将模型划分为较小的嵌套式 DRD，构成整个 DRG 的更大集中 DRD 表示。

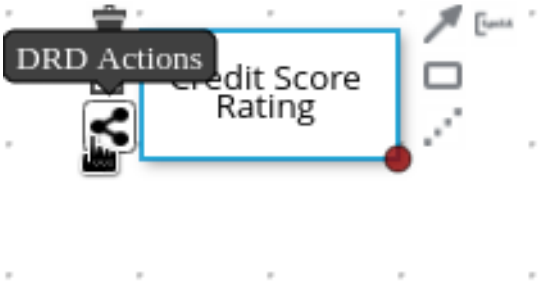
先决条件

- 您了解如何在 Business Central 中设计 DRD。有关创建 DRD 的详情，请参考 *第 6 章 在 Business Central 中创建和编辑 DMN 型号*。

流程

1. 在 Business Central 中，导航到您的 DMN 项目并在项目中创建或导入 DMN 文件。
2. 打开新的或导入的 DMN 文件，以在 DMN 设计器中查看 DRD，并开始使用左侧工具栏中的 DMN 节点设计或修改 DRD。
3. 对于您要在一个独立的嵌套 DRD 中定义的 DMN 节点，选择该节点，点 DRD Actions 图标，然后从可用选项中选择。

图 6.49. DRD 操作图标



可用的选项如下：

- 创建 ：使用这个选项创建嵌套的 DRD，您可以在其中单独定义所选节点的 DMN 组件和图表。

- 将 添加到: 如果您已经创建了嵌套的 DRD，则使用这个选项将所选节点添加到现有 DRD。

- 删除: 如果您选择的节点已在嵌套的 DRD 中，使用这个选项从那个嵌套的 DRD 中删除节点。

在 DMN 决策模型中创建嵌套的 DRD 后，新的 DRD 在单独的 DRD 菜单中选择，可用的 DRD 和组件将在 Decision Navigator 左侧菜单中列出。您可以使用 Decision Navigator 菜单来重命名或删除嵌套的 DRD。

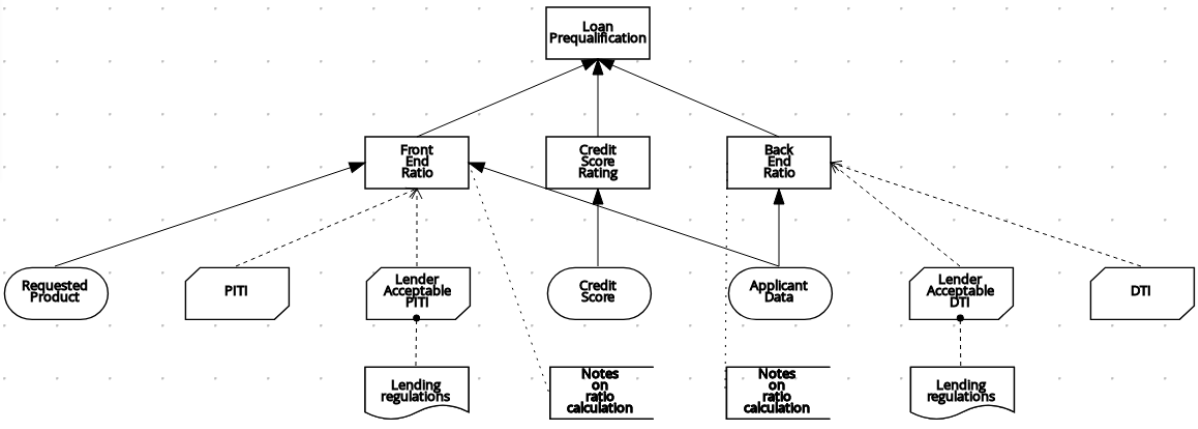
图 6.50. 在 Decision Navigator 菜单中重命名新的嵌套 DRD

4. 在新的嵌套的 DRD 中，像平常一样，为 DMN 模型中所有必要组件设计流和逻辑。

5. 继续为您的决策模型添加并定义任何其他嵌套的 DRD，并保存完整的 DMN 文件。

例如，以下 loan prequalification 决策模型的 DRD 包含了适用于任何嵌套的 DRD 的模型的所有 DMN 组件。这个示例依赖 DRD 来执行所有组件和逻辑，从而产生一个大型而复杂的图表。

图 6.51. loan prequalification single DRD



另外，按照此流程中的步骤，您可以将此示例 DRD 划分为多个嵌套的 DRD 来更好地组织决策要求，如下例所示：

图 6.52. loan prequalification 多嵌套 DRD

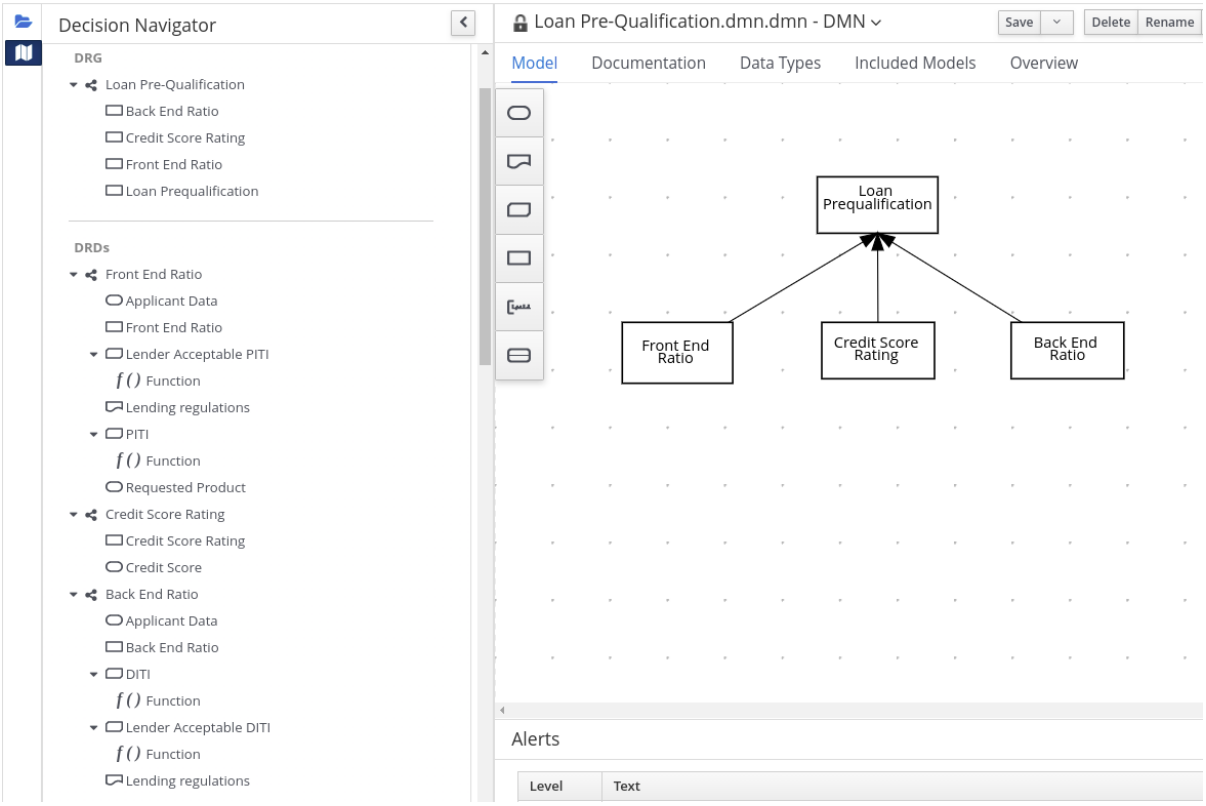


图 6.53. 前端比例 DRD 概述

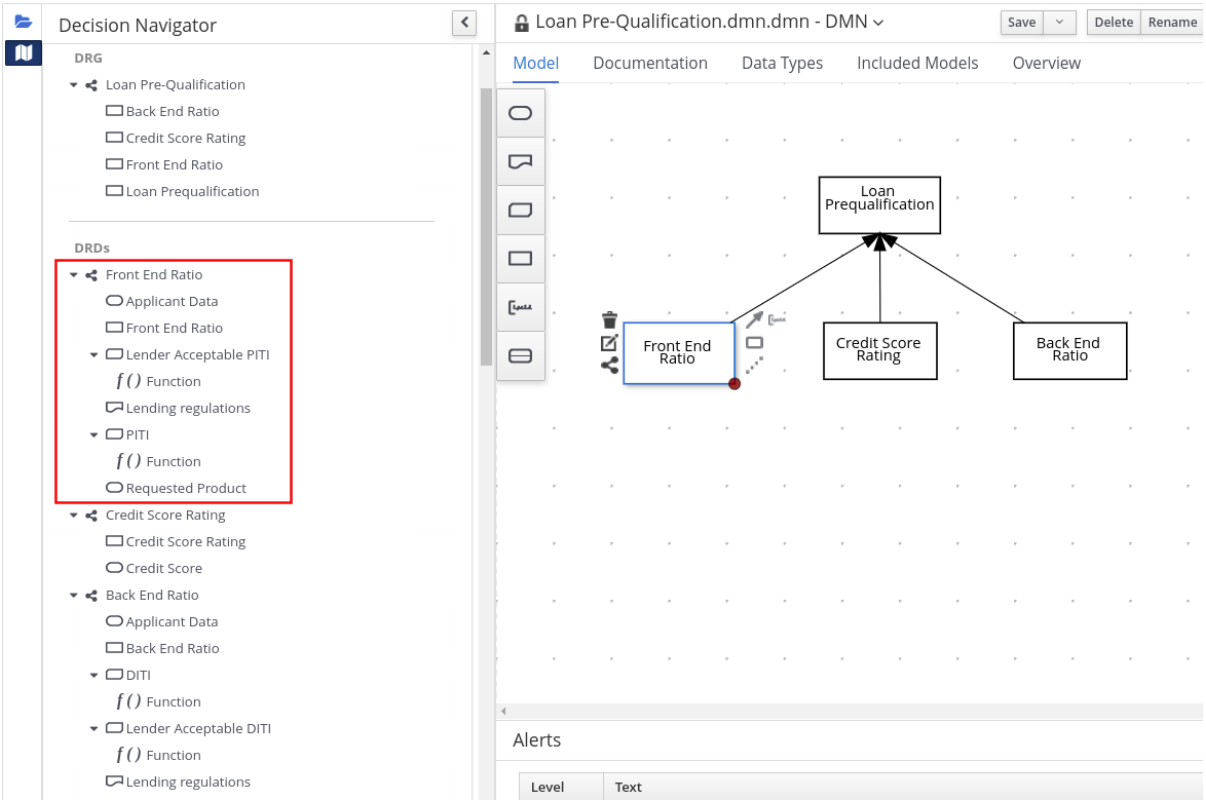


图 6.54. DRD 用于前端比例

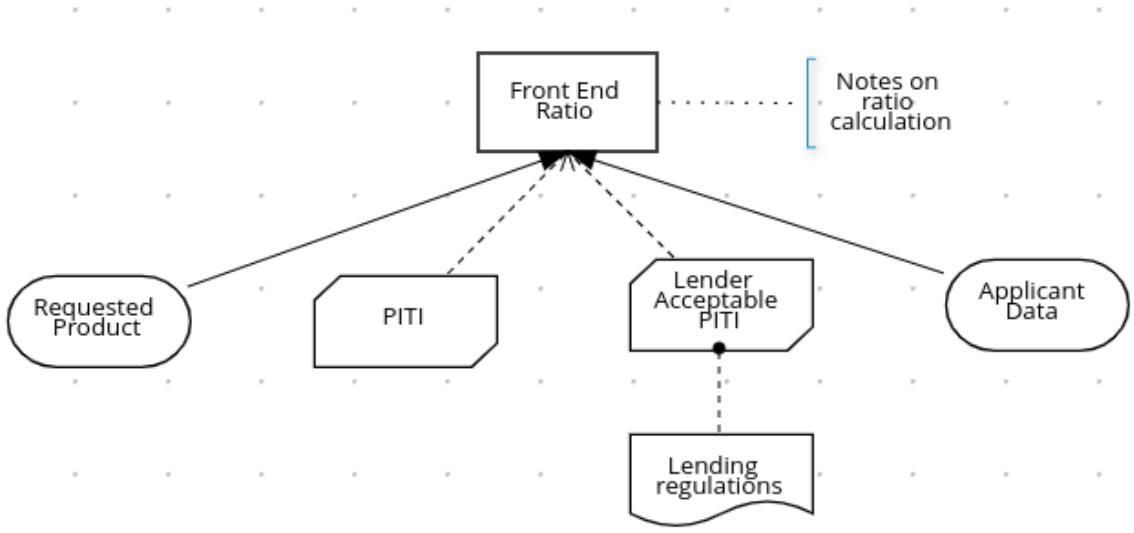


图 6.55. 贡献评级 DRD 概述

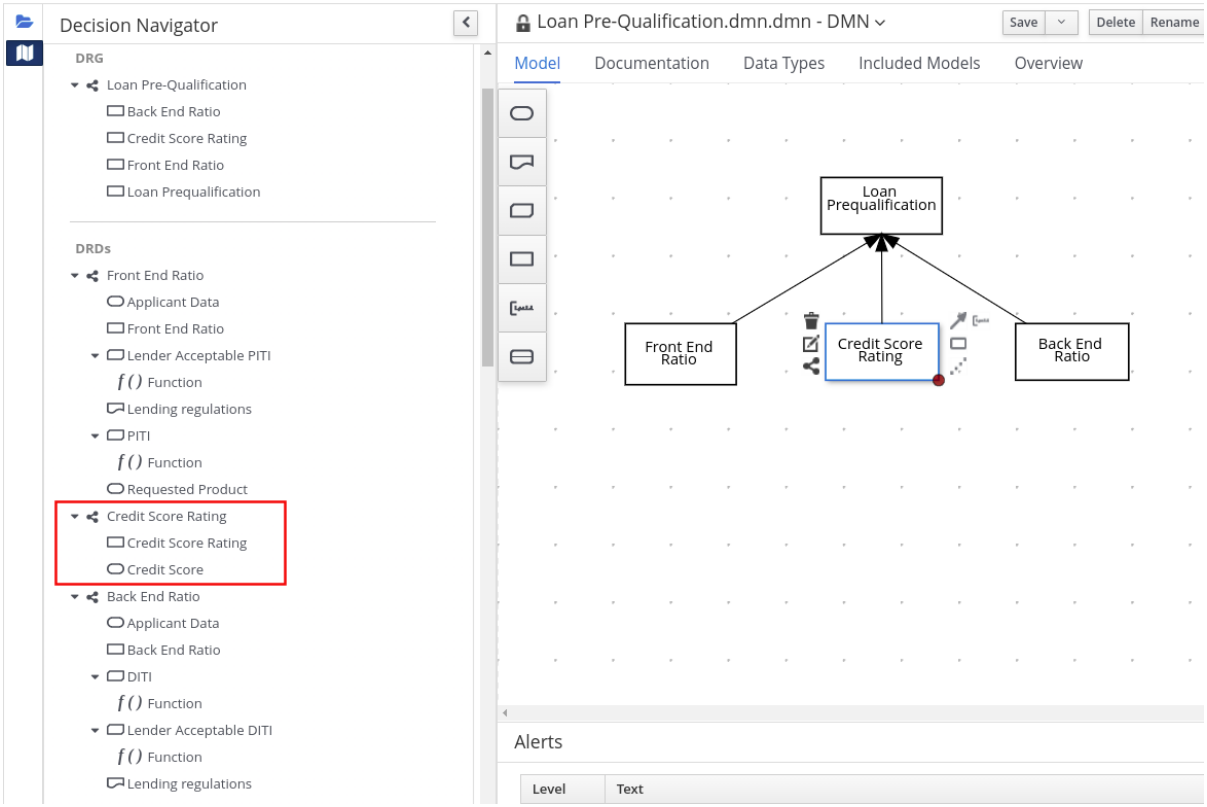


图 6.56. 得分评级的 DRD

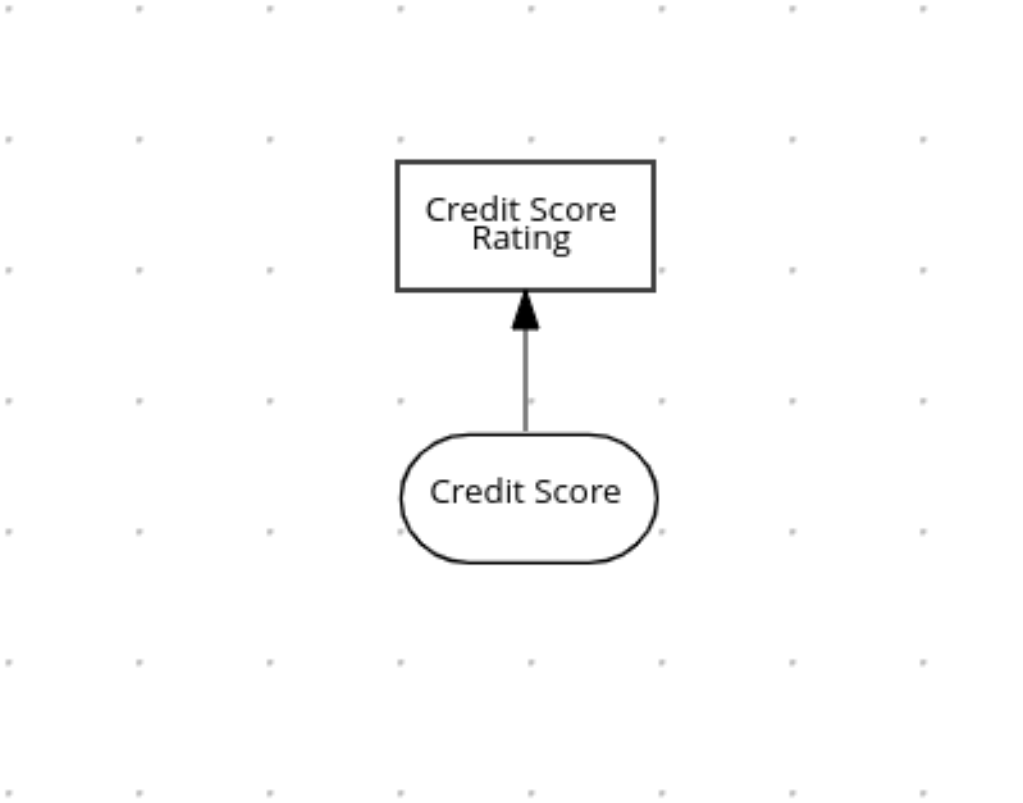


图 6.57. 后端比例 DRD 概述

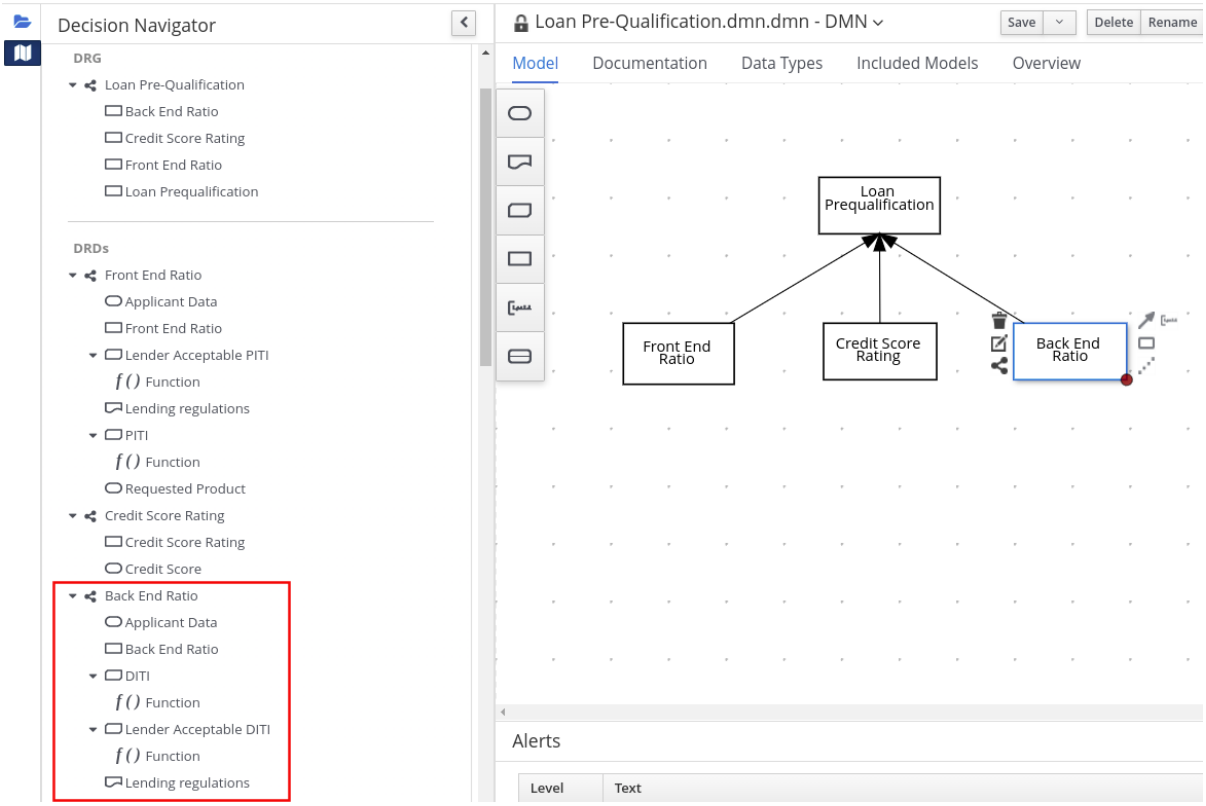
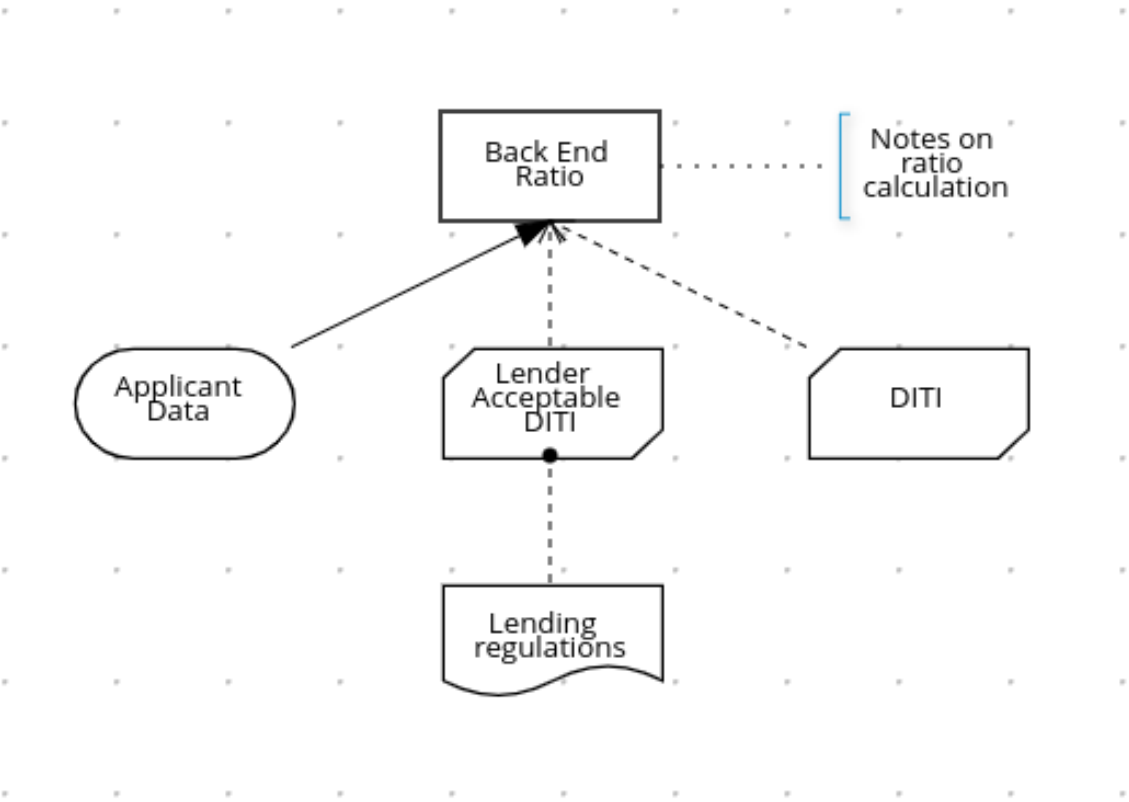


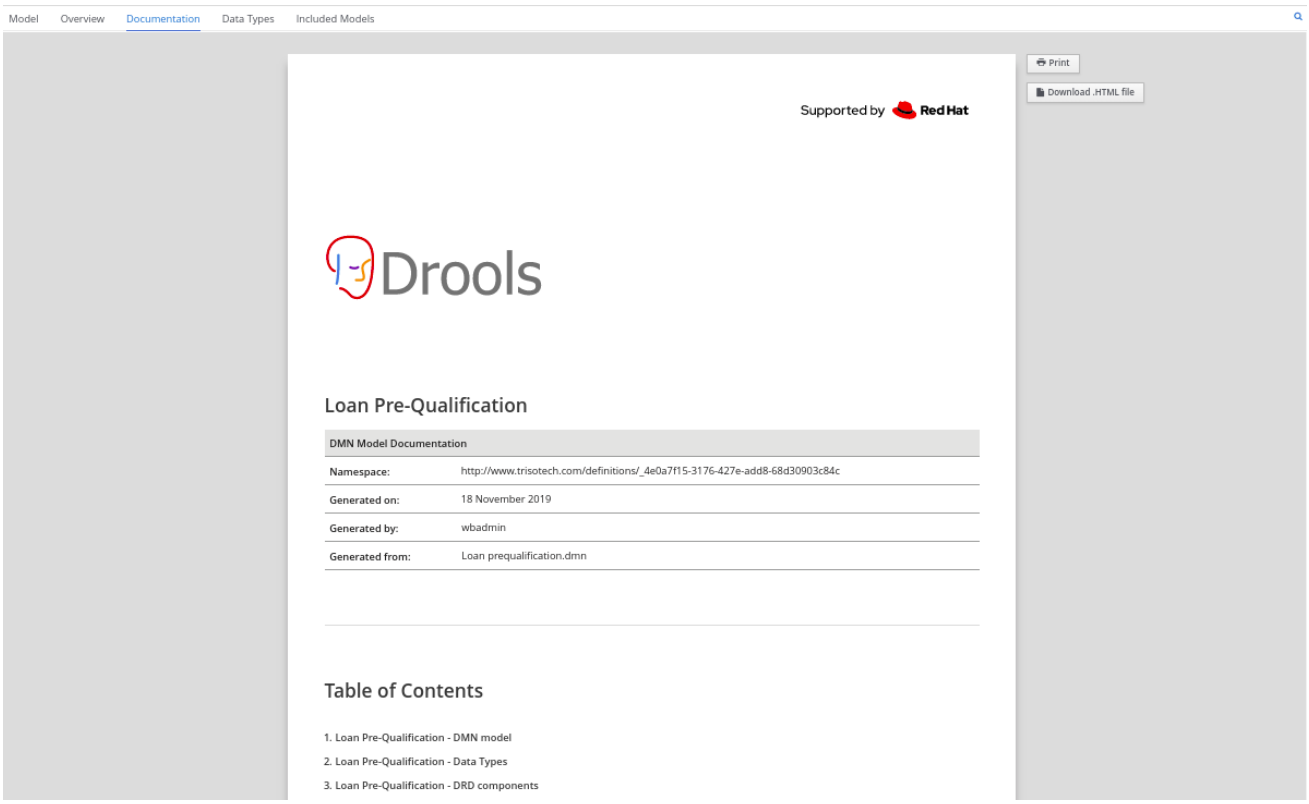
图 6.58. DRD 作为后端比例



6.5. BUSINESS CENTRAL 中的 DMN 模型文档

在 Business Central 中的 DMN 设计者中，您可以使用 Documentation 标签生成您的 DMN 模型报告，您可以打印或下载为 HTML 文件以供离线使用。DMN 模型报告包含您的 DMN 模型模型中的所有决策要求图(DRDs)、数据类型和已框表达式。您可以使用此报告来共享 DMN 模型详情，或作为内部报告工作流的一部分。

图 6.59. DMN 模型报告示例



6.6. BUSINESS CENTRAL 中的 DMN 设计器导航和属性

Business Central 中的 DMN 设计器提供以下额外功能，以帮助您浏览决策需求图(DRD)的组件和属性。

DMN 文件和图视图

在 DMN 设计器的左上角，选择 Project Explorer 视图以便在所有 DMN 和其他文件之间导航，或者选择 Decision Navigator 视图来在所选 DRD 的决策组件、图形和框表达式之间进行导航：

图 6.60. project Explorer 视图

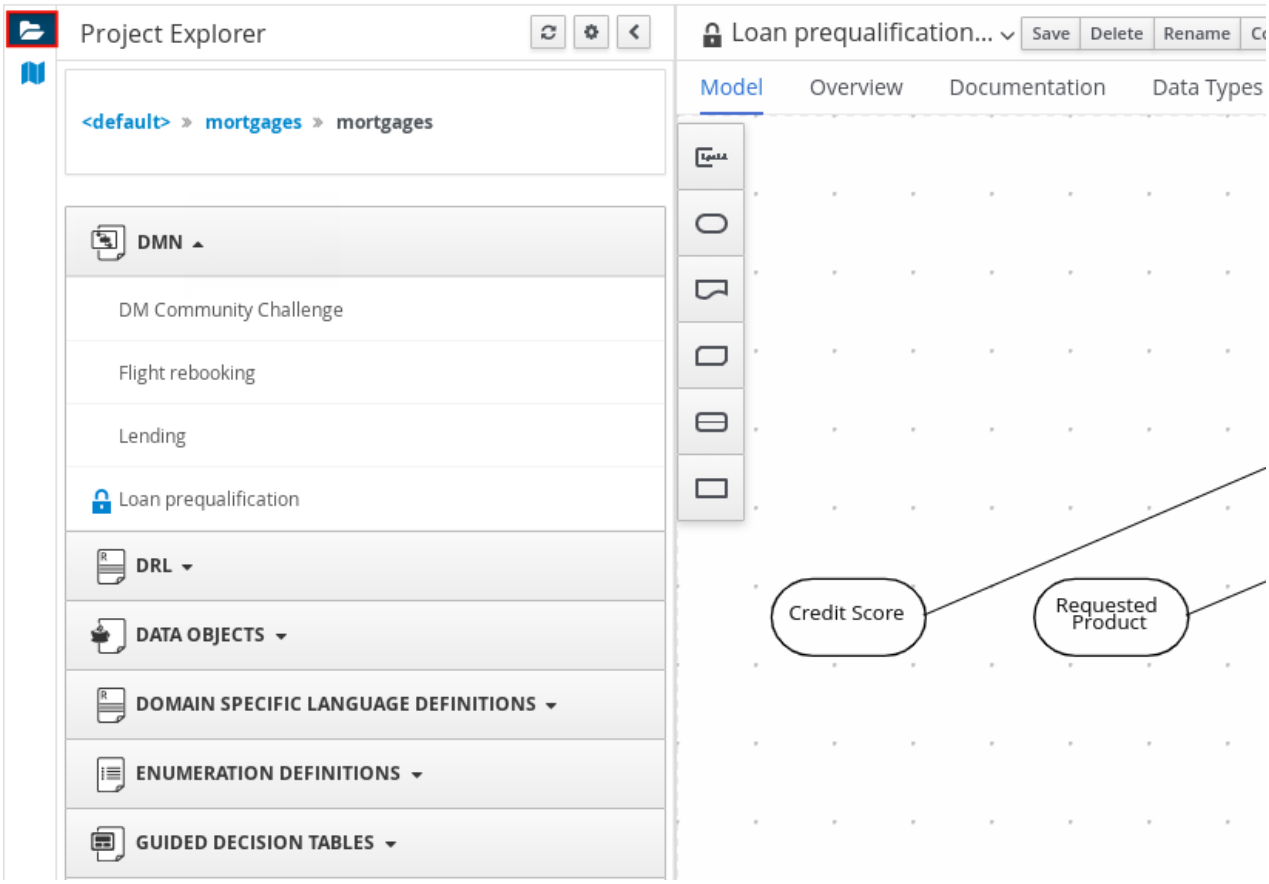
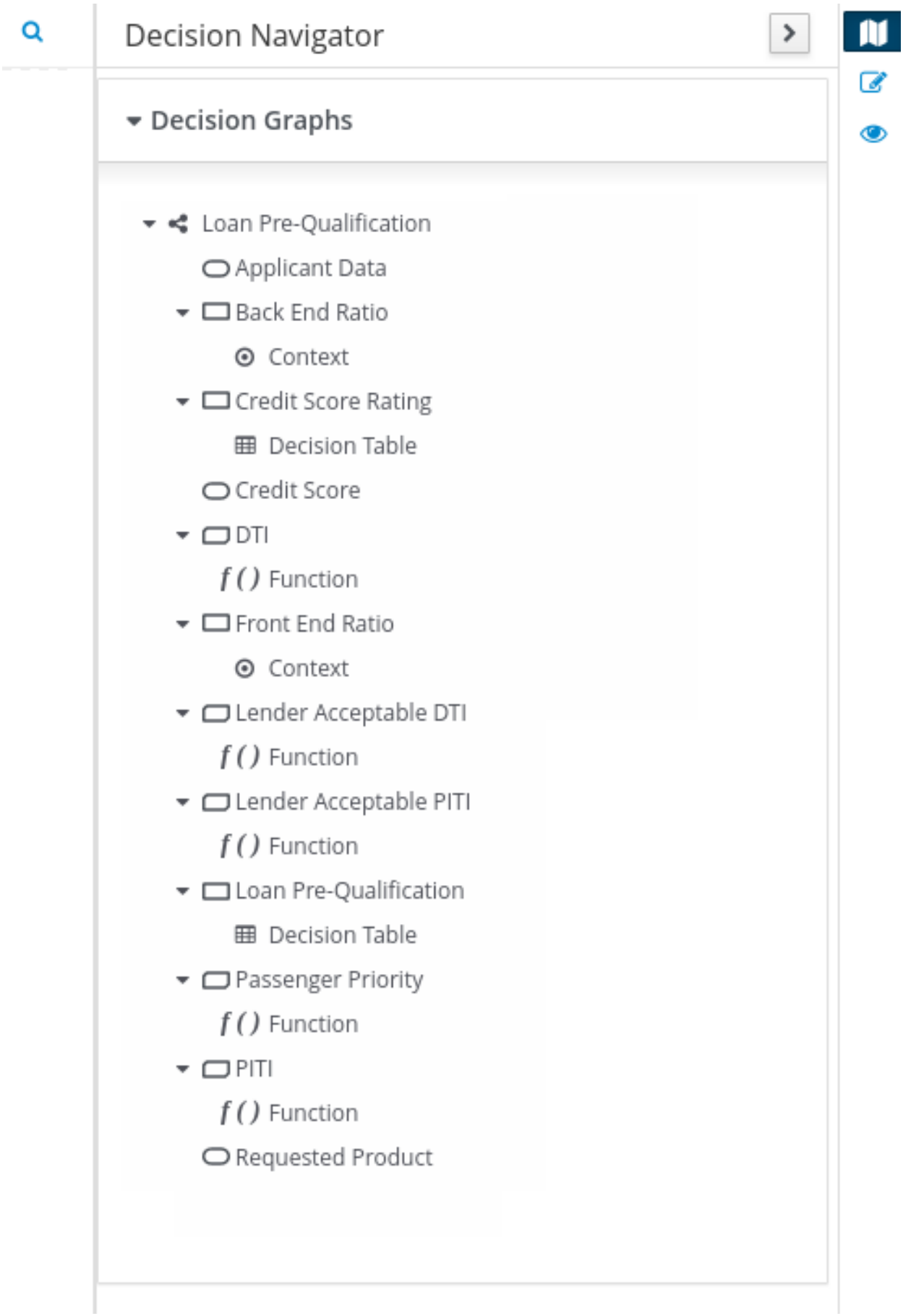
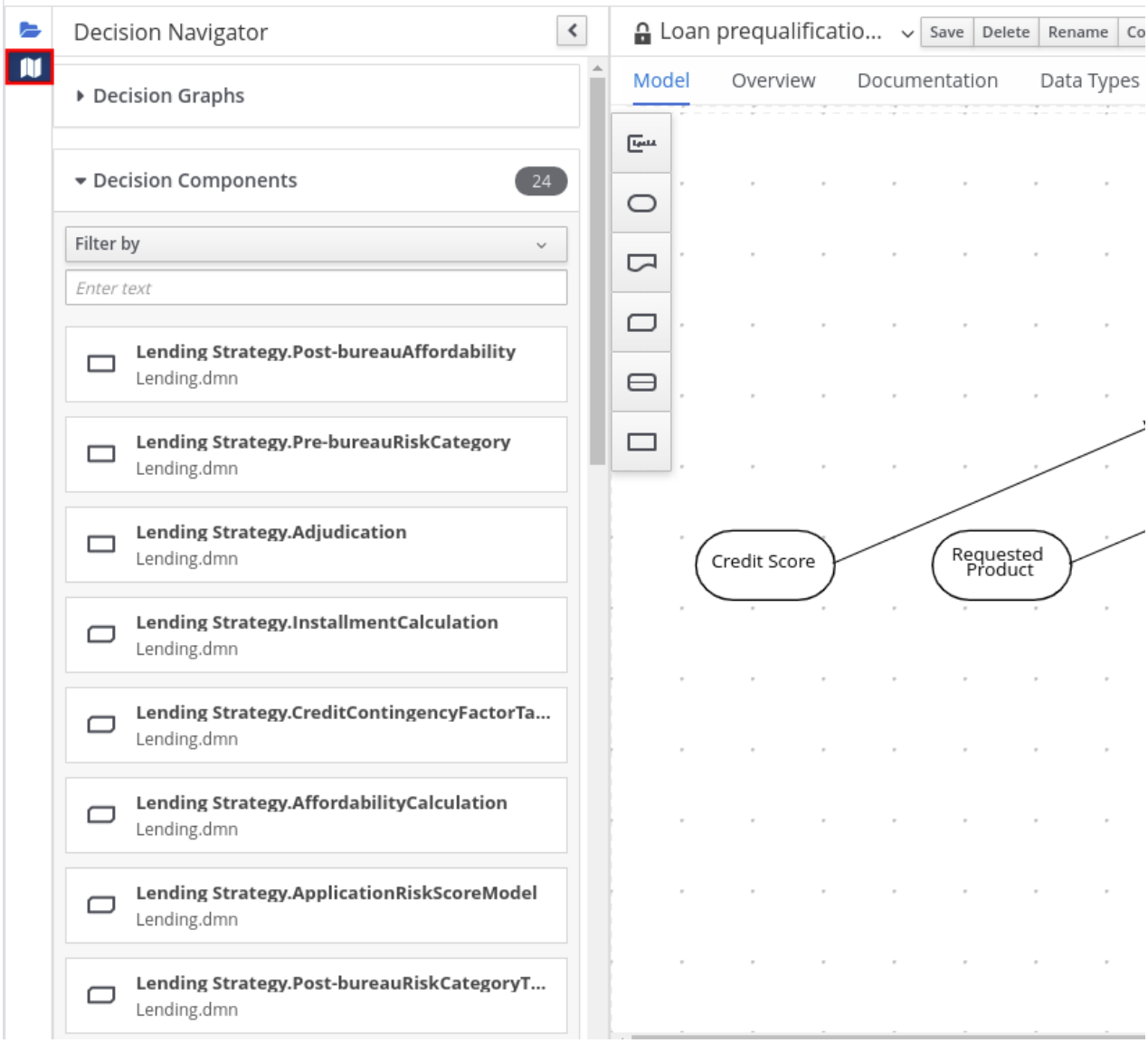


图 6.61. decision Navigator 视图



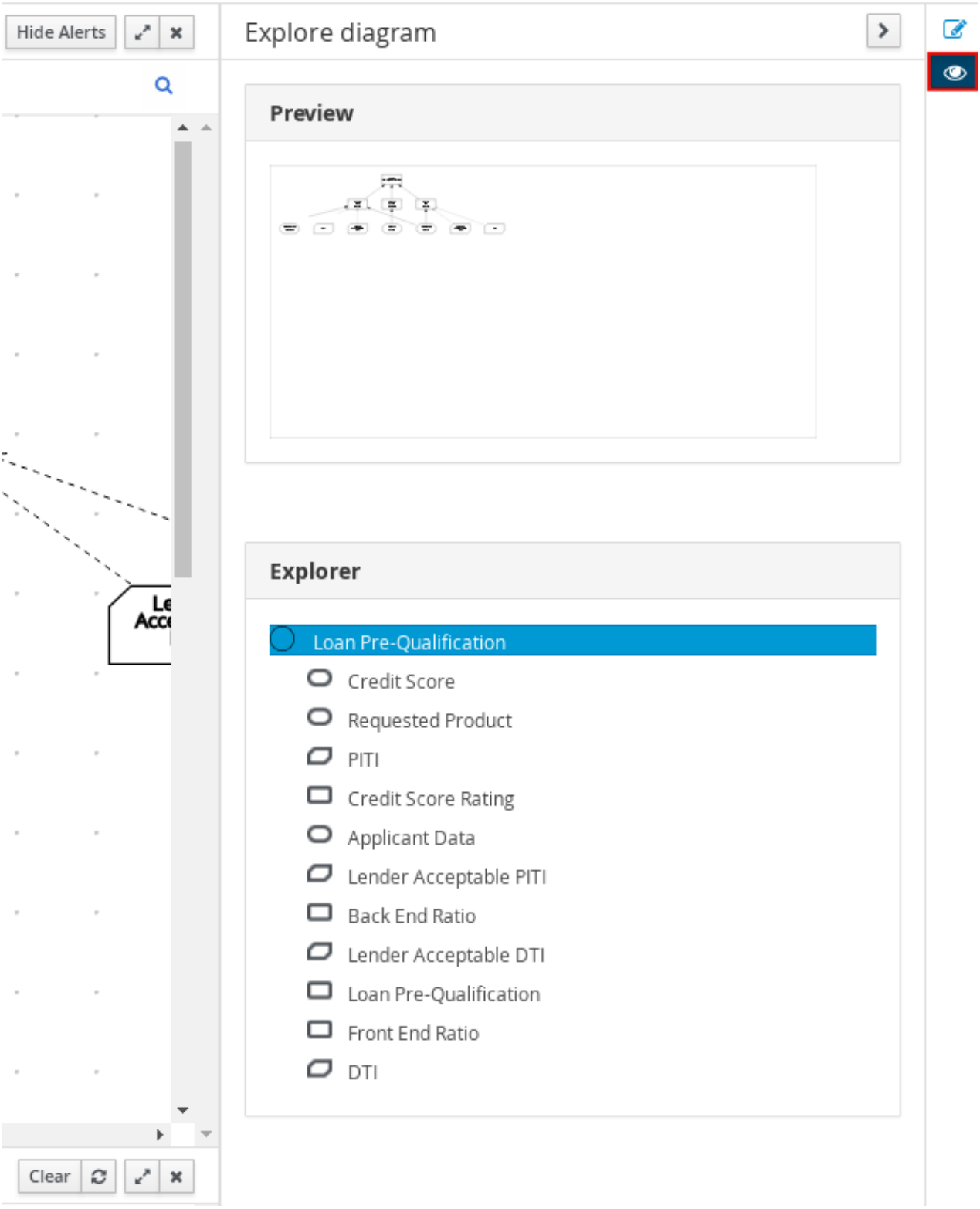


注意

DMN 文件中包括的任何 DMN 模型的 DRD 组件也列在 DMN 文件的 Decision 组件 面板中。

在 DMN 设计器右上角，选择 Explore 图表 图标来查看所选 DRD 的升级预览，并在所选 DRD 的节点之间进行导航：

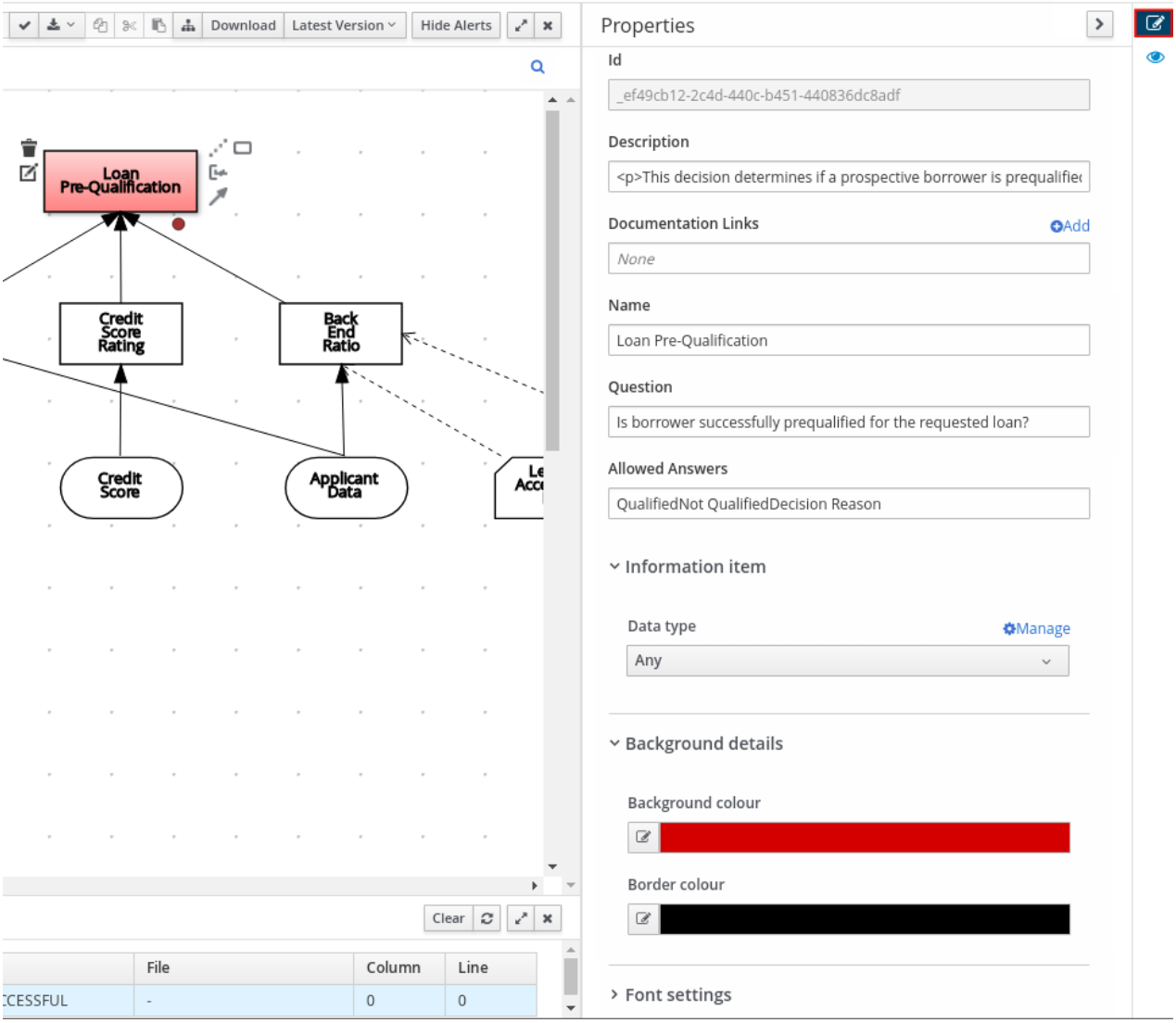
图 6.62. 浏览图表视图



DRD 属性和设计

在 DMN 设计器右上角，选择 Properties 图标来修改标识信息、数据类型和显示所选 DRD、DRD 节点或框表达式单元：

图 6.63. DRD 节点属性

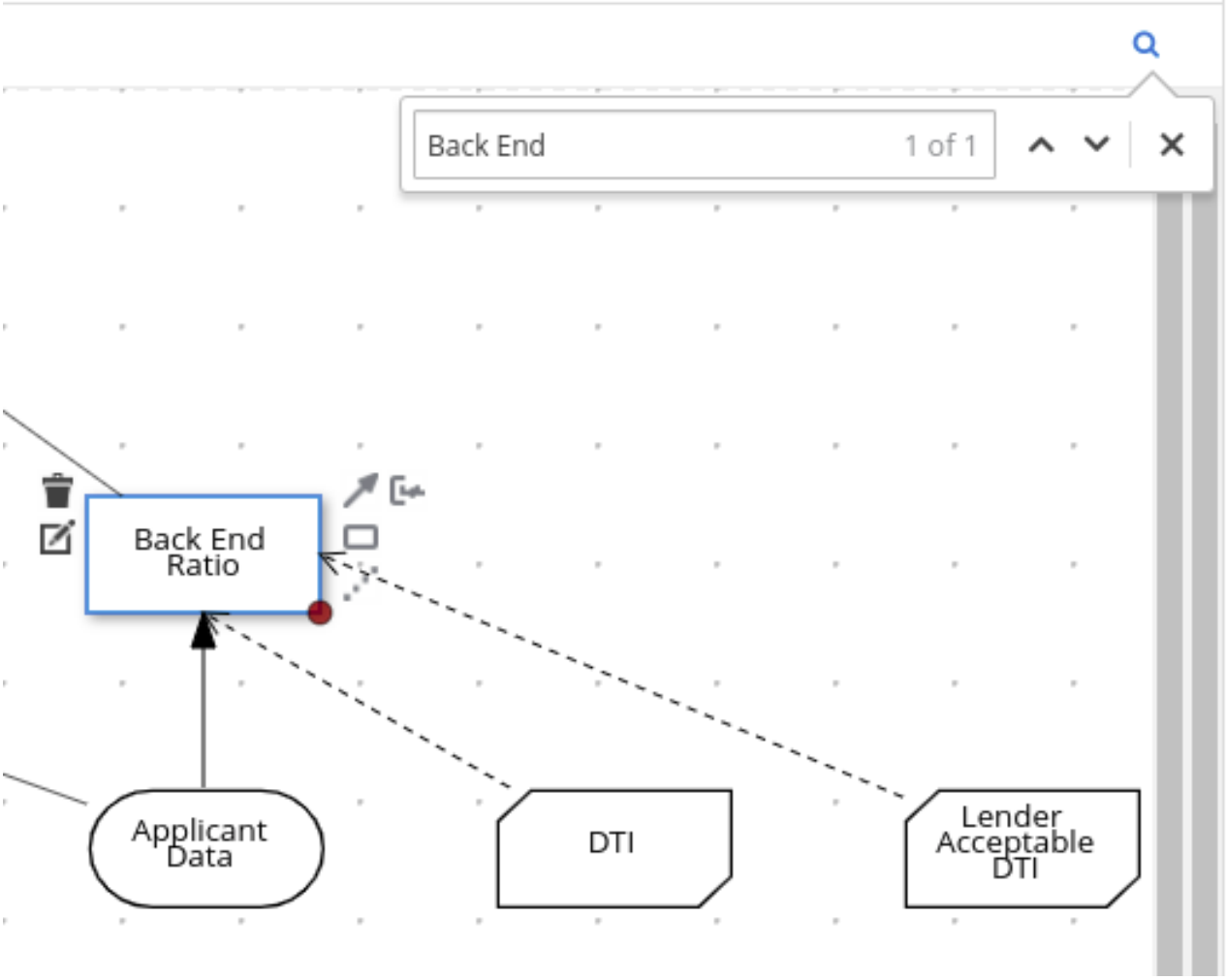


要查看整个 DRD 的属性，请点击 DRD 可清空后台而不是特定节点。

DRD 搜索

在 DMN 设计器右上角，使用搜索栏搜索出现在 DRD 的文本。搜索功能对具有多个节点的复杂 DRD 特别有用：

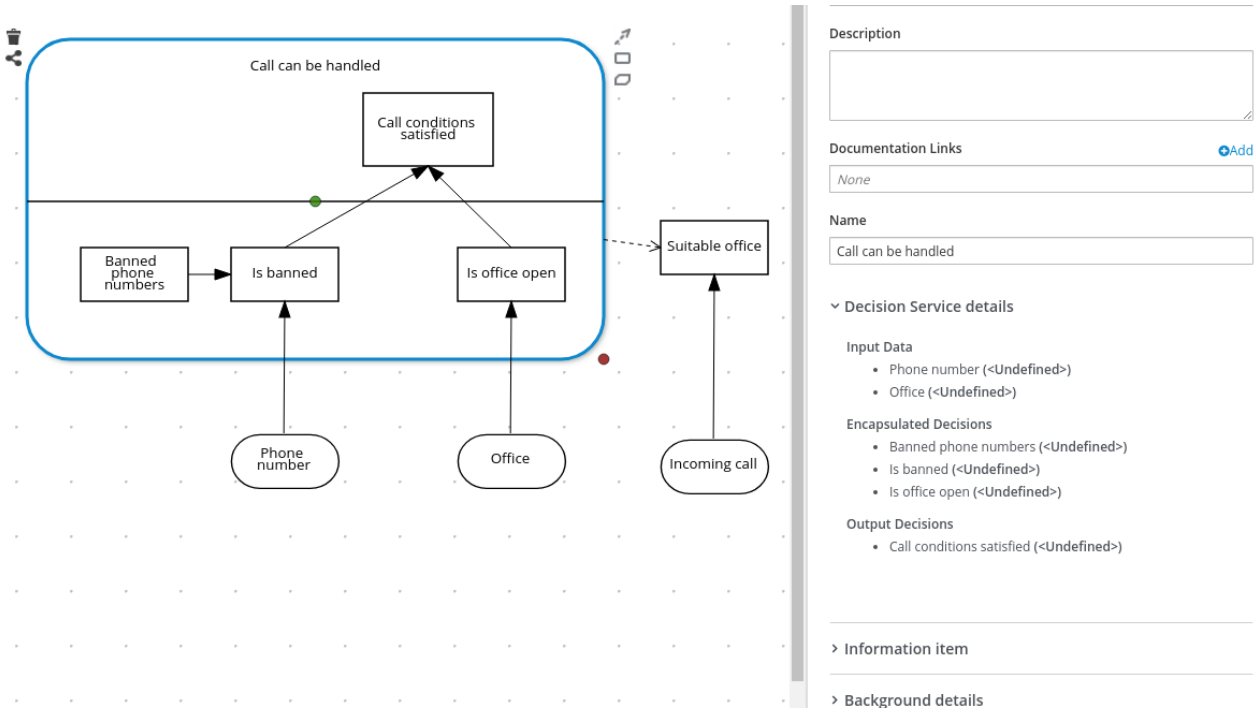
图 6.64. DRD 搜索



DMN 决策服务详情

在 DMN 设计器中选择一个决策服务节点，以在 Properties 面板中查看其他属性，包括 Input Data、Encapsulated Decisions 和 Output Decisions。

图 6.65. 决策服务详情


Call can be handled
Call conditions satisfied
Banned phone numbers
Is banned
Is office open
Suitable office
Phone number
Office
Incoming call
Description
Documentation Links
Add
None
Name
Call can be handled
Decision Service details
Input Data
Phone number (<Undefined>)
Office (<Undefined>)
Encapsulated Decisions
Banned phone numbers (<Undefined>)
Is banned (<Undefined>)
Is office open (<Undefined>)
Output Decisions
Call conditions satisfied (<Undefined>)
Information item
Background details

第 7 章 DMN 模型执行

您可以使用 Business Central 在 Red Hat Process Automation Manager 项目中创建或导入 DMN 文件，或者在没有 Business Central 的情况下将 DMN 文件打包为项目知识 JAR(KJAR)文件的一部分。在 Red Hat Process Automation Manager 项目中实现 DMN 文件后，您可以通过部署包含它的 KIE 服务器以获取 KIE 服务器来执行 DMN 决策服务，或通过直接操作 KIE 容器作为调用应用程序的依赖项来执行 DMN 决策服务。还提供了创建和部署 DMN 知识软件包的其它选项，大多数选项与所有知识资产（如 DRL 文件或进程定义）类似。

有关使用项目打包和部署方法包含外部 DMN 资产的详情，请参考 *打包和部署 Red Hat Process Automation Manager 项目*。

7.1. 直接在 JAVA 应用程序中嵌入 DMN 调用

当知识资产直接嵌入到调用程序中，或者被实际拉取为 KJAR 的 Maven 依赖关系时，KIE 容器是本地的。如果代码版本和 DMN 定义版本之间有紧密集成，您通常会直接将知识资产嵌入到项目中。在有意更新并重新部署应用程序后，对决策的任何更改都会生效。这种方法的一个优点是，正确的操作不依赖于任何外部依赖项来运行，这可能受锁定的环境限制。

使用 Maven 依赖项可进行进一步的灵活性，因为决策的特定版本可以动态更改（例如，使用系统属性），它可以定期扫描更新并自动更新。这会对服务的部署时间进行外部依赖，但在本地执行决策，从而减少在运行期间对外部服务的依赖。

先决条件

- 您已将 DMN 项目构建为 KJAR 工件并将其部署到 Maven 存储库中，或者已将 DMN 资产作为项目 classpath 的一部分：

```
mvn clean install
```

有关项目打包和部署以及可执行模型的更多信息，请参阅打包和部署 *Red Hat Process Automation Manager 项目*。

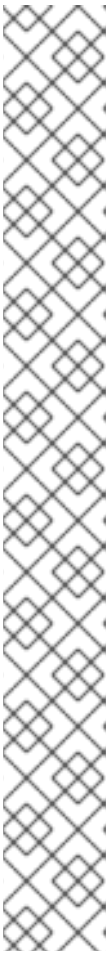
流程

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

```
<!-- Required for the DMN runtime API -->
<dependency>
  <groupId>org.kie</groupId>
```

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

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

& lt;version > 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（例如 7.67.0.Final-redhat-00024）。



注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

2. 从 类路径 或 ReleaseId 创建 KIE 容器：

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

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

备选选项：

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

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

3. 使用模型 命名空间和 modelName 从 KIE 容器获取 DMNRuntime 并对 DMN 模型的引用：

```
DMNRuntime dmnRuntime =
KieRuntimeFactory.of(kieContainer.getKieBase()).get(DMNRuntime.class);

String namespace = "http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a";
String modelName = "dmn-movieticket-ageclassification";

DMNModel dmnModel = dmnRuntime.getModel(namespace, modelName);
```

4. 为所需模型执行决策服务：

```
DMNContext dmnContext = dmnRuntime.newContext(); (1)

for (Integer age : Arrays.asList(1,12,13,64,65,66)) {
   dmnContext.set("Age", age); (2)
   DMNResult dmnResult =
      dmnRuntime.evaluateAll(dmnModel, dmnContext); (3)

   for (DMNDecisionResult dr : dmnResult.getDecisionResults()) { (4)
      log.info("Age: " + age + ", " +
            "Decision: '" + dr.getDecisionName() + "', " +
            "Result: " + dr.getResult());
  }
}
```

(1) 实例化新的 DMN 上下文，作为模型评估的输入。请注意，本例多次通过 Age 类决策进行循环。

(2) 为输入 DMN 上下文分配输入变量。

(3) 评估 DMN 模型中定义的所有 DMN 决策。

(4)

这个示例输出如下：

```
Age 1 Decision 'AgeClassification' : Child
Age 12 Decision 'AgeClassification' : Child
Age 13 Decision 'AgeClassification' : Adult
Age 64 Decision 'AgeClassification' : Adult
Age 65 Decision 'AgeClassification' : Senior
Age 66 Decision 'AgeClassification' : Senior
```

如果之前未编译 DMN 模型作为可执行模型以便更有效的执行，您可以在执行 DMN 模型时启用以下属性：

```
-Dorg.kie.dmn.compiler.execmodel=true
```

7.2. 使用 KIE 服务器 JAVA 客户端 API 执行 DMN 服务

KIE 服务器 Java 客户端 API 提供了通过 REST 或 KIE 服务器的 JMS 接口调用远程 DMN 服务的轻量级方法。这个方法减少了与 KIE 基本交互所需的运行时依赖项数量。从决策定义中分离调用代码，通过使它们能够按照适当的步调进行迭代，从而提高了灵活性。

有关 KIE Server Java 客户端 API 的更多信息，*请参阅使用 KIE API 与 Red Hat Process Automation Manager 交互*。

先决条件

- KIE 服务器是安装和配置的，包括具有 kie-server 角色的用户的已知用户名和凭证。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。
- 您已将 DMN 项目构建为 KJAR 工件，并将其部署到 KIE 服务器中：

```
mvn clean install
```

有关项目打包和部署以及可执行模型的更多信息，请参阅打包和部署 *Red Hat Process Automation Manager 项目*。

- 您有包含 DMN 模型的 KIE 容器的 ID。如果存在多个模型，还必须知道相关模型的型号命名

空间和型号名称。

流程

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

```
<!-- Required for the KIE Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

& lt;version > 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（例如 7.67.0.Final-redhat-00024）。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

2. 使用适当的连接信息实例化 KieServicesClient 实例。

例如：

```
KieServicesConfiguration conf =
    KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD); 1

conf.setMarshallingFormat(MarshallingFormat.JSON); 2

KieServicesClient kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
```

1

连接信息：

- URL 示例 ：http://localhost:8080/kie-server/services/rest/server
- 该凭据应当引用具有 kie-server 角色的用户。

2

Marshalling 格式是一个 org.kie.server.api.marshalling.MarshallingFormat 的实例。它控制消息是 JSON 还是 XML。Marshalling 格式的选项为 JSON、JAXB 或 XSTREAM。

3. 通过调用 KIE 服务器 Java 客户端上的方法 getServicesClient（） 从连接到相关 KIE 服务器 Java 客户端的 KIE 服务器 Java 客户端获取 DMNServicesClient Client ：

```
DMNServicesClient dmnClient =
kieServicesClient.getServicesClient(DMNServicesClient.class );
```

dmnClient 现在可在 KIE 服务器上执行决策服务。

4. 为所需模型执行决策服务。

例如：

```
for (Integer age : Arrays.asList(1,12,13,64,65,66)) {
    DMNContext dmnContext = dmnClient.newContext(); 1
    dmnContext.set("Age", age); 2
    ServiceResponse<DMNResult> serverResp = 3
        dmnClient.evaluateAll($kieContainerId,
                              $modelNamespace,
                              $modelName,
                              dmnContext);
```

```
    DMNResult dmnResult = serverResp.getResult(); 4
    for (DMNDecisionResult dr : dmnResult.getDecisionResults()) {
        log.info("Age: " + age + ", " +
                "Decision: '" + dr.getDecisionName() + "', " +
                "Result: " + dr.getResult());
    }
}
```

1

实例化新的 DMN 上下文，作为模型评估的输入。请注意，本例多次通过 Age 类决策进行循环。

2

为输入 DMN 上下文分配输入变量。

3

评估 DMN 模型中定义的所有 DMN 决策：

- $kieContainerId 是部署了包含 DMN 模型的 KJAR 的容器的 ID
- $modelNamespace 是型号的命名空间。
- $modelName 是模型的名称。

4

DMN Result 对象可从服务器响应中获得。

此时，dmnResult 包含所评估的 DMN 模型中的所有决定结果。

您还可以使用 DMNServicesClient 的替代方法，在模型中仅执行特定的 DMN 决策。

注意

如果 KIE 容器仅包含一个 DMN 模型，您可以省略 $modelNamespace 和 $modelName，因为 KIE Server API 会默认选择它。

7.3. 使用 KIE 服务器 REST API 执行 DMN 服务

与 KIE 服务器的 REST 端点直接交互，提供调用代码和决策逻辑定义之间的最分离。调用代码完全没有直接依赖项，您可以在一个完全不同的开发平台（如 Node.js 或 .NET ）中实施。本节中的示例演示 Nix 风格的 curl 命令，但提供相关信息来适应任何 REST 客户端。

当您使用 KIE 服务器的 REST 端点时，最佳实践是定义一个域对象 POJO Java 类，使用标准 KIE 服务器 marshalling 注解标注。例如，以下代码使用正确注解的域对象 Person 类：

POJO Java 类示例

```
@javax.xml.bind.annotation.XmlAccessorType(javax.xml.bind.annotation.XmlAccessType.FIELD)
public class Person implements java.io.Serializable {

  static final long serialVersionUID = 1L;

  private java.lang.String id;
  private java.lang.String name;

@javax.xml.bind.annotation.adapters.XmlJavaTypeAdapter(org.kie.internal.jaxb.LocalDateXmlAdapter.class)
  private java.time.LocalDate dojoining;

  public Person() {
  }

  public java.lang.String getId() {
    return this.id;
  }

  public void setId(java.lang.String id) {
    this.id = id;
  }

  public java.lang.String getName() {
    return this.name;
  }

  public void setName(java.lang.String name) {
    this.name = name;
  }
```

```
    public java.time.LocalDate getDojoining() {
        return this.dojoining;
    }

    public void setDojoining(java.time.LocalDate dojoining) {
        this.dojoining = dojoining;
    }

    public Person(java.lang.String id, java.lang.String name,
            java.time.LocalDate dojoining) {
        this.id = id;
        this.name = name;
        this.dojoining = dojoining;
    }

}
```

有关 KIE Server REST API 的更多信息，*请参阅使用 KIE API 与 Red Hat Process Automation Manager 交互*。

先决条件

- KIE 服务器是安装和配置的，包括具有 kie-server 角色的用户的已知用户名和凭证。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。
- 您已将 DMN 项目构建为 KJAR 工件，并将其部署到 KIE 服务器中：

```
mvn clean install
```

有关项目打包和部署以及可执行模型的更多信息，请参阅打包和部署 *Red Hat Process Automation Manager 项目*。

- 您有包含 DMN 模型的 KIE 容器的 ID。如果存在多个模型，还必须知道相关模型的型号命名空间和型号名称。

流程

1. 确定用于访问 KIE 服务器 REST API 端点的基础 URL。这需要了解以下值（使用默认本地部署值作为示例）：

- 主机（本地主机）
- 端口(8080)
- 根上下文(kie-server)
- 基本 REST 路径(services/rest/)

本地部署中的基本 URL 示例：

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

2. 确定用户身份验证要求。

当在 KIE 服务器配置中直接定义用户时，使用 HTTP 基本身份验证并需要用户名和密码。成功请求需要该用户具有 kie-server 角色。

以下示例演示了如何在 curl 请求中添加凭证：

```
curl -u username:password <request>
```

如果使用 Red Hat Single Sign-On 配置 KIE 服务器，则请求必须包含 bearer 令牌：

```
curl -H "Authorization: bearer $TOKEN" <request>
```

3. 指定请求和响应的格式。REST API 端点同时使用 JSON 和 XML 格式，并使用请求标头来设置：

JSON

```
curl -H "accept: application/json" -H "content-type: application/json"
```

XML

```
curl -H "accept: application/xml" -H "content-type: application/xml"
```

4. **可选：查询容器以获取部署的决策模型列表：**

   [GET] server/containers/{containerId}/dmn

   curl 请求示例：

```
curl -u krisv:krisv -H "accept: application/xml" -X GET "http://localhost:8080/kie-server/services/rest/server/containers/MovieDMNContainer/dmn"
```

   XML 输出示例：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="OK models successfully retrieved from container 'MovieDMNContainer'">
   <dmn-model-info-list>
      <model>
         <model-namespace>http://www.redhat.com/_c7328033-c355-43cd-b616-0aceef80e52a</model-namespace>
         <model-name>dmn-movieticket-ageclassification</model-name>
         <model-id>_99</model-id>
         <decisions>
            <dmn-decision-info>
               <decision-id>_3</decision-id>
               <decision-name>AgeClassification</decision-name>
            </dmn-decision-info>
         </decisions>
      </model>
   </dmn-model-info-list>
</response>
```

   JSON 输出示例：

```
{
  "type" : "SUCCESS",
  "msg" : "OK models successfully retrieved from container 'MovieDMNContainer'",
  "result" : {
    "dmn-model-info-list" : {
      "models" : [ {
        "model-namespace" : "http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a",
        "model-name" : "dmn-movieticket-ageclassification",
        "model-id" : "_99",
        "decisions" : [ {
          "decision-id" : "_3",
          "decision-name" : "AgeClassification"
        } ]
      } ]
    }
  }
}
```

5. 执行模型：

   [POST] server/containers/{containerId}/dmn

   curl 请求示例：

```
curl -u krisv:krisv -H "accept: application/json" -H "content-type: application/json" -X POST
"http://localhost:8080/kie-server/services/rest/server/containers/MovieDMNContainer/dmn" -d
"{ \"model-namespace\" : \"http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a\", \"model-name\" : \"dmn-movieticket-ageclassification\", \"decision-name\" : [
], \"decision-id\" : [ ], \"dmn-context\" : {\"Age\" : 66}}"
```

   JSON 请求示例：

```
{
  "model-namespace" : "http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a",
  "model-name" : "dmn-movieticket-ageclassification",
  "decision-name" : [ ],
  "decision-id" : [ ],
  "dmn-context" : {"Age" : 66}
}
```

   XML 请求示例（JAXB 格式）：

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<dmn-evaluation-context>
  <model-namespace>http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a</model-namespace>
  <model-name>dmn-movieticket-ageclassification</model-name>
  <dmn-context xsi:type="jaxbListWrapper" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
    <type>MAP</type>
    <element xsi:type="jaxbStringObjectPair" key="Age">
      <value xsi:type="xs:int" xmlns:xs="http://www.w3.org/2001/XMLSchema">66</value>
    </element>
  </dmn-context>
</dmn-evaluation-context>
```

注意

无论请求格式如何，请求都需要以下元素：

- **型号命名空间**
- **型号名称**
- **包含输入值的上下文对象**

JSON 响应示例：

```
{
 "type" : "SUCCESS",
 "msg" : "OK from container 'MovieDMNContainer'",
 "result" : {
  "dmn-evaluation-result" : {
   "messages" : [ ],
   "model-namespace" : "http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a",
   "model-name" : "dmn-movieticket-ageclassification",
   "decision-name" : [ ],
   "dmn-context" : {
    "Age" : 66,
    "AgeClassification" : "Senior"
   },
   "decision-results" : {
    "_3" : {
     "messages" : [ ],
     "decision-id" : "_3",
     "decision-name" : "AgeClassification",
     "result" : "Senior",
```

```
        "status" : "SUCCEEDED"
      }
    }
  }
 }
}
```

XML（JAXB 格式）响应示例：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="OK from container 'MovieDMNContainer'">
    <dmn-evaluation-result>
        <model-namespace>http://www.redhat.com/_c7328033-c355-43cd-b616-
0aceef80e52a</model-namespace>
        <model-name>dmn-movieticket-ageclassification</model-name>
        <dmn-context xsi:type="jaxbListWrapper"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <type>MAP</type>
            <element xsi:type="jaxbStringObjectPair" key="Age">
                <value xsi:type="xs:int"
xmlns:xs="http://www.w3.org/2001/XMLSchema">66</value>
            </element>
            <element xsi:type="jaxbStringObjectPair" key="AgeClassification">
                <value xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema">Senior</value>
            </element>
        </dmn-context>
        <messages/>
        <decisionResults>
            <entry>
                <key>_3</key>
                <value>
                    <decision-id>_3</decision-id>
                    <decision-name>AgeClassification</decision-name>
                    <result xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Senior</result>
                    <messages/>
                    <status>SUCCEEDED</status>
                </value>
            </entry>
        </decisionResults>
    </dmn-evaluation-result>
</response>
```

7.4. 特定 DMN 模型的 REST 端点

Red Hat Process Automation Manager 提供特定于模型的 DMN KIE Server 端点，您可以在不使用 Business Central 用户界面的情况下与特定的 DMN 模型交互。

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

- POST /server/containers/{containerId}/dmn/models/{modelname}: 一个用于评估容器中的指定 DMN 模型的业务域端点
- POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}: 一个用于评估容器中可用特定 DMN 模型中指定的决策服务组件的业务域端点
- POST /server/containers/{containerId}/dmn/models/{modelname}/dmnresult: 一个端点来评估包含自定义正文有效负载并返回 DMNResult 响应，包括业务域上下文、帮助程序消息和帮助程序决策指针
- POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}/dmnresult: 一个端点来评估特定 DMN 模型中的指定决策服务组件，并返回 DMNResult 响应，包括业务域上下文、帮助程序消息和帮助决策方向服务
- GET /server/containers/{containerId}/dmn/models/{modelname}: 一个端点，用于在没有决策逻辑的情况下返回标准 DMN XML，并包含指定 DMN 模型的输入和输出
- GET /server/containers/{containerId}/dmn/openapi.json (|.yaml): 一个端点，用于在指定容器中为 DMN 模型检索 Swagger 或 OAS

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

- REST Business-domain 端点 ：如果客户端应用程序只关注正评估结果，请使用此端点类型，不负责解析 Info 或 Warn 消息，且只需要 HTTP 5xx 响应来进行任何错误。由于单一的决策服务结果与 DMN 模型行为类似，这种类型的端点对单页应用程序的客户端也很有用。
- REST dmnresult 端点 ：如果客户端需要解析 Info、Warn 或 Error 消息，请使用此端点类型。

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

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

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

对于所有这些端点，如果出现 DMN 评估 Error 消息，则返回 DMNResult 响应并带有 HTTP 5xx 错误。如果发生 DMN Info 或 Warn 消息，则会在 X-Kogito-decision-messages 扩展 HTTP 标头中返回相关响应以及业务域 REST 正文，以用于客户端侧的业务逻辑。当需要更多优化的客户端业务逻辑的要求时，客户端可以使用端点的 dmnresult 变体。

在指定容器中检索 DMN 模型的 Swagger 或 OAS

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

REST 端点示例

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

返回没有决策逻辑的 DMN XML

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

REST 端点示例

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

curl 请求示例

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

响应示例(XML)

```
<?xml version='1.0' encoding='UTF-8'?>
<dmn:definitions xmlns:dmn="http://www.omg.org/spec/DMN/20180521/MODEL/" xmlns="https://kiegroup.org/dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF" xmlns:di="http://www.omg.org/spec/DMN/20180521/DI/" xmlns:kie="http://www.drools.org/kie/dmn/1.2" xmlns:feel="http://www.omg.org/spec/DMN/20180521/FEEL/" xmlns:dmndi="http://www.omg.org/spec/DMN/20180521/DMNDI/" xmlns:dc="http://www.omg.org/spec/DMN/20180521/DC/" id="_1C792953-80DB-4B32-99EB-25FBE32BAF9E" name="Traffic Violation" expressionLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/" typeLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/" namespace="https://kiegroup.org/dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF">
  <dmn:extensionElements/>
  <dmn:itemDefinition id="_63824D3F-9173-446D-A940-6A7F0FA056BB" name="tDriver" isCollection="false">
```

```
    <dmn:itemComponent id="_9DAB5DAA-3B44-4F6D-87F2-95125FB2FEE4" name="Name"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_856BA8FA-EF7B-4DF9-A1EE-E28263CE9955" name="Age"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_FDC2CE03-D465-47C2-A311-98944E8CC23F" name="State"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_D6FD34C4-00DC-4C79-B1BF-BBCF6FC9B6D7" name="City"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_7110FE7E-1A38-4C39-B0EB-AEEF06BA37F4" name="Points"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_40731093-0642-4588-9183-1660FC55053B" name="tViolation"
isCollection="false">
    <dmn:itemComponent id="_39E88D9F-AE53-47AD-B3DE-8AB38D4F50B3" name="Code"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_1648EA0A-2463-4B54-A12A-D743A3E3EE7B" name="Date"
isCollection="false">
      <dmn:typeRef>date</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_9F129EAA-4E71-4D99-B6D0-84EEC3AC43CC" name="Type"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
      <dmn:allowedValues kie:constraintType="enumeration" id="_626A8F9C-9DD1-44E0-9568-
0F6F8F8BA228">
        <dmn:text>"speed", "parking", "driving under the influence"</dmn:text>
      </dmn:allowedValues>
    </dmn:itemComponent>
    <dmn:itemComponent id="_DDD10D6E-BD38-4C79-9E2F-8155E3A4B438" name="Speed
Limit" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_229F80E4-2892-494C-B70D-683ABF2345F6" name="Actual
Speed" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_2D4F30EE-21A6-4A78-A524-A5C238D433AE" name="tFine"
isCollection="false">
    <dmn:itemComponent id="_B9F70BC7-1995-4F51-B949-1AB65538B405" name="Amount"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_F49085D6-8F08-4463-9A1A-EF6B57635DBD" name="Points"
isCollection="false">
```

```
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:inputData id="_1929CBD5-40E0-442D-B909-49CEDE0101DC" name="Violation">
    <dmn:variable id="_C16CF9B1-5FAB-48A0-95E0-5FCD661E0406" name="Violation"
typeRef="tViolation"/>
  </dmn:inputData>
  <dmn:decision id="_4055D956-1C47-479C-B3F4-BAEB61F1C929" name="Fine">
    <dmn:variable id="_8C1EAC83-F251-4D94-8A9E-B03ACF6849CD" name="Fine"
typeRef="tFine"/>
    <dmn:informationRequirement id="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
      <dmn:requiredInput href="#_1929CBD5-40E0-442D-B909-49CEDE0101DC"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmn:inputData id="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" name="Driver">
    <dmn:variable id="_A80F16DF-0DB4-43A2-B041-32900B1A3F3D" name="Driver"
typeRef="tDriver"/>
  </dmn:inputData>
  <dmn:decision id="_8A408366-D8E9-4626-ABF3-5F69AA01F880" name="Should the driver be
suspended?">
    <dmn:question>Should the driver be suspended due to points on his
license?</dmn:question>
    <dmn:allowedAnswers>"Yes", "No"</dmn:allowedAnswers>
    <dmn:variable id="_40387B66-5D00-48C8-BB90-E83EE3332C72" name="Should the driver be
suspended?" typeRef="string"/>
    <dmn:informationRequirement id="_982211B1-5246-49CD-BE85-3211F71253CF">
      <dmn:requiredInput href="#_1F9350D7-146D-46F1-85D8-15B5B68AF22A"/>
    </dmn:informationRequirement>
    <dmn:informationRequirement id="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
      <dmn:requiredDecision href="#_4055D956-1C47-479C-B3F4-BAEB61F1C929"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmndi:DMNDI>
    <dmndi:DMNDiagram>
      <di:extension/>
      <dmndi:DMNShape id="dmnshape-_1929CBD5-40E0-442D-B909-49CEDE0101DC"
dmnElementRef="_1929CBD5-40E0-442D-B909-49CEDE0101DC" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="708" y="350" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_4055D956-1C47-479C-B3F4-BAEB61F1C929"
dmnElementRef="_4055D956-1C47-479C-B3F4-BAEB61F1C929" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="709" y="210" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_1F9350D7-146D-46F1-85D8-15B5B68AF22A"
```

```
dmnElementRef="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="369" y="344" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_8A408366-D8E9-4626-ABF3-5F69AA01F880"
dmnElementRef="_8A408366-D8E9-4626-ABF3-5F69AA01F880" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="534" y="83" width="133" height="63"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNEdge id="dmnedge-_800A3BBB-90A3-4D9D-BA5E-A311DED0134F"
dmnElementRef="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
        <di:waypoint x="758" y="375"/>
        <di:waypoint x="759" y="235"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_982211B1-5246-49CD-BE85-3211F71253CF"
dmnElementRef="_982211B1-5246-49CD-BE85-3211F71253CF">
        <di:waypoint x="419" y="369"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_AEC4AA5F-50C3-4FED-A0C2-261F90290731"
dmnElementRef="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
        <di:waypoint x="759" y="235"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
    </dmndi:DMNDiagram>
  </dmndi:DMNDI>
```

评估指定容器中的指定 DMN 模型

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

REST 端点示例

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

curl 请求示例

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

带有输入数据的 POST 请求正文示例

```
{
  "Driver": {
    "Points": 15
  },
  "Violation": {
    "Date": "2021-04-08",
    "Type": "speed",
    "Actual Speed": 135,
    "Speed Limit": 100
  }
}
```

响应示例(JSON)

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

```
  },
  "Should the driver be suspended?": "Yes"
}
```

评估容器中指定的 DMN 模型中指定的决策服务

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

对于此端点，请求正文必须包含决策服务的所有要求。响应是决策服务生成的 DMN 上下文，包括决策值、原始输入值以及所有其他 parametric DRG 组件（序列化形式）。例如，业务知识模型以字符串序列化形式提供。

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

图 7.1. 带有单输出决策的 TrafficViolationDecisionService 决策服务示例

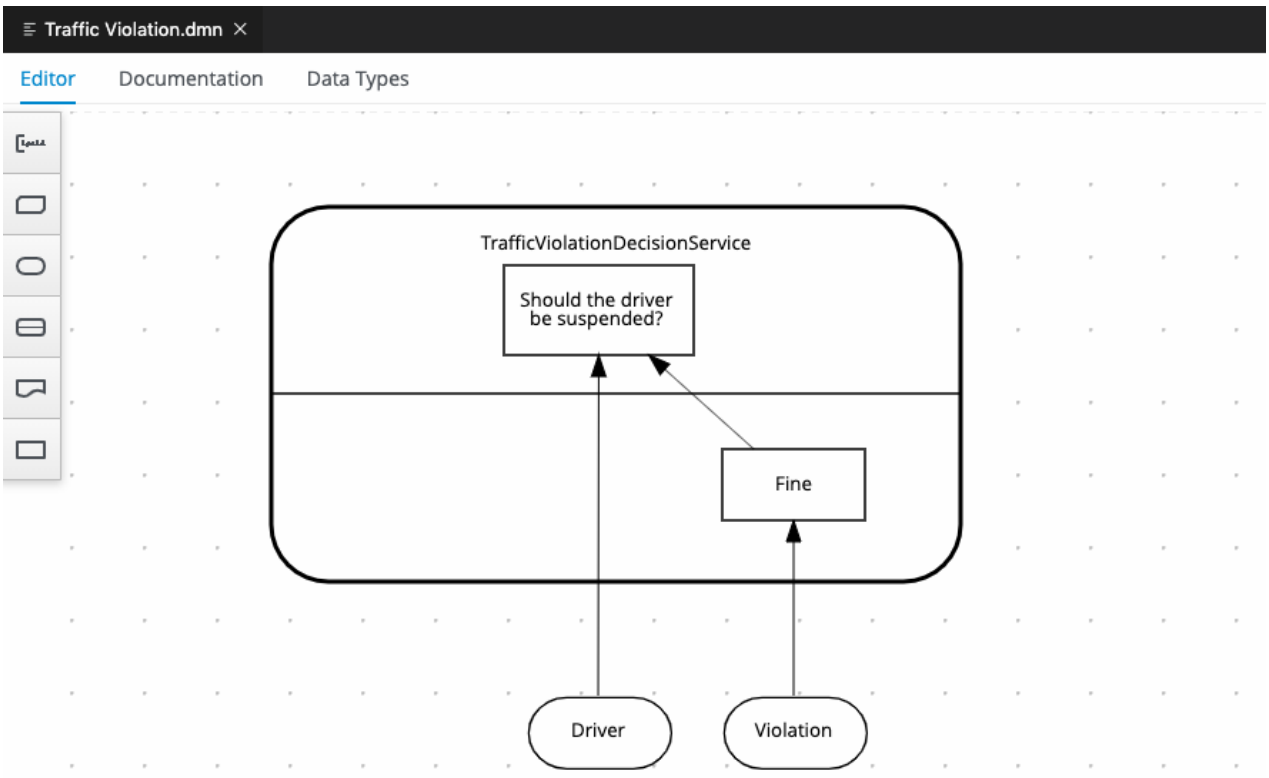
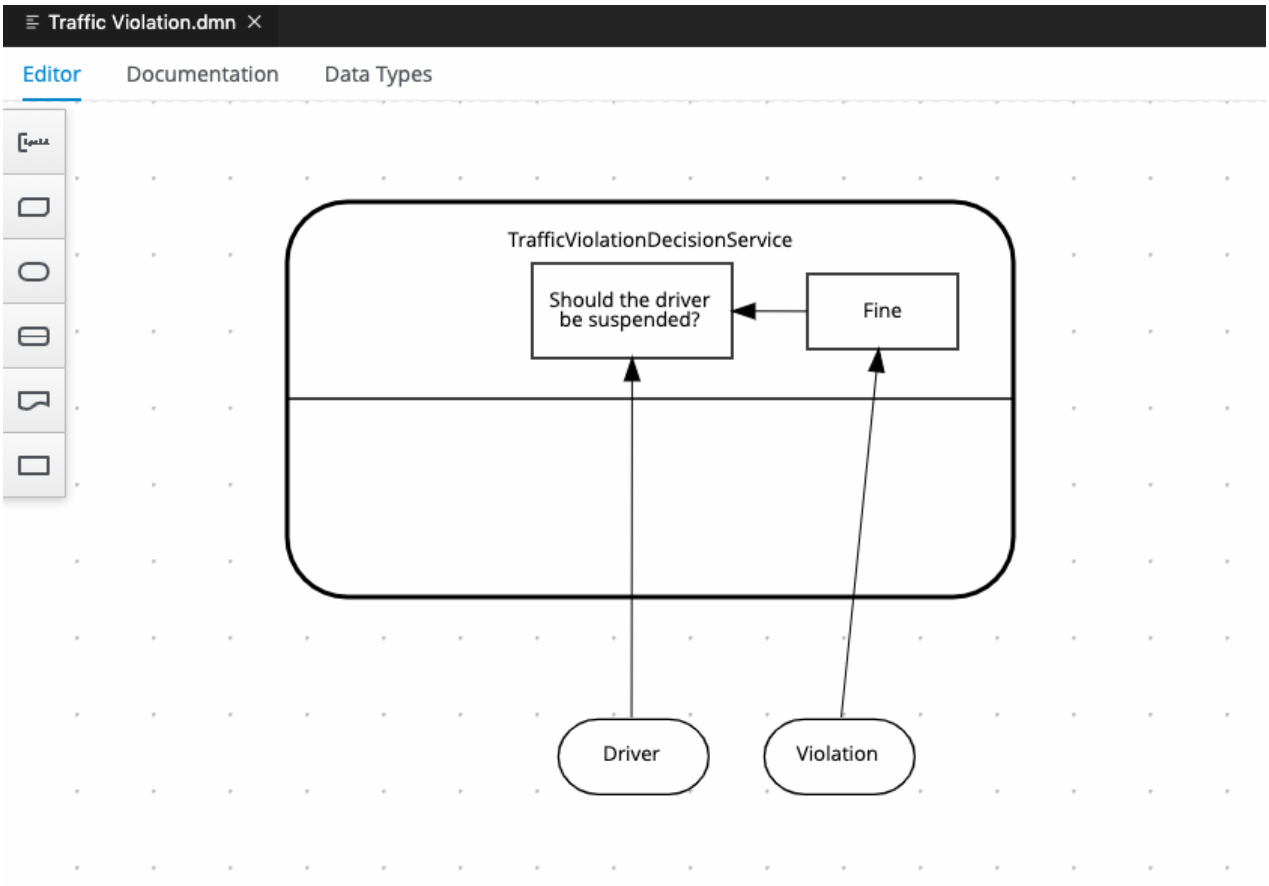


图 7.2. 带有多输出决定的 TrafficViolationDecisionService 决策服务示例



REST 端点示例

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

带有输入数据的 POST 请求正文示例

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

curl 请求示例

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

单输出决策(JSON)的响应示例

```
"No"
```

多输出决策(JSON)的响应示例

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

评估指定容器中的指定 DMN 模型，并返回 DMNResult 响应

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

REST 端点示例

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

带有输入数据的 POST 请求正文示例

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

curl 请求示例

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

响应示例(JSON)

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

```
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Fine": {
      "Points": 3,
      "Amount": 500
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
    {
      "decisionId": "_4055D956-1C47-479C-B3F4-BAEB61F1C929",
      "decisionName": "Fine",
      "result": {
        "Points": 3,
        "Amount": 500
      },
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    },
    {
      "decisionId": "_8A408366-D8E9-4626-ABF3-5F69AA01F880",
      "decisionName": "Should the driver be suspended?",
      "result": "No",
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    }
  ]
}
```


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

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

REST 端点示例

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

带有输入数据的 POST 请求正文示例

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

curl 请求示例

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

响应示例(JSON)

```
{
  "namespace": "https://kiegroup.org/dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF",
  "modelName": "Traffic Violation",
  "dmnContext": {
    "Violation": {
      "Type": "speed",
      "Speed Limit": 100,
      "Actual Speed": 120,
      "Code": null,
      "Date": null
    },
    "Driver": {
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
```

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

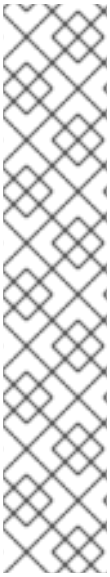
第 8 章 其他资源

- **决策模型和符号规格**
- **DMN 技术兼容性工具包**
- ***打包和部署 Red Hat Process Automation Manager 项目***
- ***使用 KIE API 与 Red Hat Process Automation Manager 交互***

部分 II. 使用 PMML 型号设计决策服务

作为业务规则开发人员，您可以使用预测模型标记语言(PMML)来定义统计或数据分割模型，它们可与 Red Hat Process Automation Manager 中的决策服务集成。Red Hat Process Automation Manager 包括对 Regression、scorecard、Tree 和 Mining model 的消费者一致性支持。Red Hat Process Automation Manager 不包括内置的 PMML 模型编辑器，但您可以使用 XML 或 PMML 特定的授权工具来创建 PMML 模型，然后将其与红帽流程自动化管理器项目集成。

有关 PMML 的更多信息，请参阅 DMG PMML 规格。



注意

您还可以使用决策模型和 Notation(DMN)模型设计您的决策服务，并将 PMML 模型作为 DMN 服务的一部分包括在内。有关 Red Hat Process Automation Manager 7.13 中的 DMN 支持的详情，请查看以下资源：

- *开始使用决策服务*（逐步教程，带有 DMN 决策服务示例）
- *使用 DMN 模型设计决策服务* (Red Hat Process Automation Manager 中的 DMN 支持和功能视图)

第 9 章 红帽流程自动化管理器中的决策资产

Red Hat Process Automation Manager 支持一些可用于为决策服务定义业务决策的资产。每个决策资产都有不同的优势，您可以根据目标和需求，首选使用一个或多个资产的组合。

下表重点介绍红帽流程自动化管理器项目支持的主要决策资产，以帮助您决定或确认在决策服务中定义决策的最佳方法。

表 9.1. Red Hat Process Automation Manager 支持的决策资产

asset	亮点	编写工具	Documentation
决策模型和符号(DMN)型号	• 决策模型基于由对象管理组(OMG)定义的符号标准。 • 使用代表部分或所有决策要求图(DRG)的图形化决策要求图(DRG)跟踪业务决策流程 • 使用 XML 模式，允许在 DMN 兼容平台间共享 DMN 模型 • 支持 Friendly Enough Expression Language(FEEL)，在 DMN 决策表中定义决策逻辑，以及其他 DMN 框的表达式 • 可以与业务流程模型和符号(BPMN)流程模型有效集成 • 是创建全面、清晰和稳定的决策流程的最佳选择	Business Central 或其他与 DMN 兼容的编辑器	*使用 DMN 模型设计决策服务*
指导的决定表	• 是您在 Business Central 中基于 UI 的表格设计者创建的规则表 • 是电子表格决策表的向导替代选择 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值 • 支持点击策略、实时验证和其他资产不支持的其他功能 • 是以受控表格格式创建规则的最佳选择，可最小化编译错误	Business Central	*使用指导决策表设计决策服务*

asset	亮点	编写工具	Documentation
电子表格决策表	• 是 XLS 或 XLSX 表格决策表，您可以上传到 Business Central • 支持创建规则模板的模板键和值 • 是创建已在 Business Central 之外管理的决策表中规则的最佳选择 • 在上传后，对规则进行编译具有严格的语法要求	电子表格编辑器	*使用电子表格决策表设计决策服务*
指导规则	• 是您在 Business Central 中基于 UI 的规则设计者创建的单独规则 • 为可接受的输入提供字段和选项 • 是以受控格式创建单一规则的最佳选择，以最小化编译错误	Business Central	*使用指导规则设计决策服务*
指导规则模板	• 是您在 Business Central 中基于 UI 的模板设计者创建的可重复使用的规则结构 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值（与此资产的目的不相同） • 是创建具有相同规则结构但具有不同定义字段值的很多规则的最佳	Business Central	*使用指导规则模板设计决策服务*
DRL 规则	• 是直接在 **.drl** 文本文件中定义的单独规则 • 提供定义规则和其他规则行为的最大的灵活性 • 可以在特定的独立环境中创建，并与 Red Hat Process Automation Manager 集成 • 是创建需要高级 DRL 选项的规则的最佳选择 • 对规则进行正确编译的规则具有严格的语法要求	业务中心或集成开发环境(IDE)	*使用 DRL 规则设计决策服务*

asset	亮点	编写工具	Documentation
预测模型标记语言(PMML)型号	• 是基于数据 Mining Group(DMG)定义的标记标准预测数据分析模型。 • 使用允许 PMML 模式在 PMML 兼容的平台之间共享的 XML 模式 • 支持 Regression、scorecard、Tree、Mining 和其他模型类型 • 可以包含在独立 Red Hat Process Automation Manager 项目中，或者导入到 Business Central 中的项目 • 在 Red Hat Process Automation Manager 中，将预测数据合并为决策服务的最佳选择	PMML 或 XML 编辑器	*使用 PMML 模型设计决策服务*

在定义决策时，您还可以考虑将红帽构建的 Kogito 构建用于云原生决策服务。有关使用红帽构建的 Kogito 微服务的更多信息，请参阅 Red Hat *Process Automation Manager 中的 Red Hat build of Kogito*。

第 10 章 预测模型标记语言(PMML)

预测模型标记语言(PMML)是一个基于 XML 的标准，由 Data Mining Group(DMG)设定，用于定义统计数据和数据分割模型。PMML 模式可以在 PMML 兼容平台和跨组织之间共享，以便业务分析和开发人员在设计、分析和实施 PMML 资产和服务方面具有统一。

有关 PMML 的背景和应用程序的更多信息，请参阅 DMG PMML 规格。

10.1. PMML 一致性级别

PMML 规格在软件实施中定义制作者和消费者一致性级别，以确保 PMML 模型创建并可靠地集成。有关每个符合等级的正式定义，请查看 DMG PMML 一致性 页面。

以下列表总结了 PMML 合规级别：

生产者一致性

如果工具或应用程序生成有效 PMML 文档，则至少为一种模型生成有效的 PMML 文档。满足 PMML producer 一致性要求可确保模型定义文档正确，并定义与模型规格中定义的语义条件一致的模型实例。

消费者规范

如果应用程序接受至少一种模型的有效 PMML 文档，则应用程序符合使用者。满足消费者一致性要求，确保根据制作者一致性创建的 PMML 模型可以集成并用作定义。例如，如果应用程序消费者符合 Regression 模型类型，则有效的 PMML 文档定义了由不同符合生产者生成的模型。

Red Hat Process Automation Manager 包括对以下 PMML 模型类型的消费者一致支持：

- 回归模型
- Scorecard 模型
- 树形模型

- 最小模型 （使用子类型 模型Chain，选择All，然后选择First）
- 集群模型

有关所有 PMML 模型类型的列表，包括 Red Hat Process Automation Manager 中不支持的项，请参阅 DMG PMML 规格。

第 11 章 PMML 模型示例

PMML 定义了一个 XML 模式，它允许在不同 PMML 兼容平台之间使用 PMML 模型。PMML 规格允许多个软件平台与同一文件配合使用，用于编写、测试和生产执行，假定满足生产者和消费者一致性。

以下是 PMML Regression、Scorecard、Tree、Min Mining 和 clusters 模型的示例。这些示例演示了您可以在 Red Hat Process Automation Manager 中与决策服务集成支持的模型。

有关 PMML 示例，请参阅 DMG PMML Sample Files 页面。

PMML Regression 模型示例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="5">
    <DataField dataType="double" name="fld1" optype="continuous"/>
    <DataField dataType="double" name="fld2" optype="continuous"/>
    <DataField dataType="string" name="fld3" optype="categorical">
      <Value value="x"/>
      <Value value="y"/>
    </DataField>
    <DataField dataType="double" name="fld4" optype="continuous"/>
    <DataField dataType="double" name="fld5" optype="continuous"/>
  </DataDictionary>
  <RegressionModel algorithmName="linearRegression" functionName="regression" modelName="LinReg" normalizationMethod="logit" targetFieldName="fld4">
    <MiningSchema>
      <MiningField name="fld1"/>
      <MiningField name="fld2"/>
      <MiningField name="fld3"/>
      <MiningField name="fld4" usageType="predicted"/>
      <MiningField name="fld5" usageType="target"/>
    </MiningSchema>
    <RegressionTable intercept="0.5">
      <NumericPredictor coefficient="5" exponent="2" name="fld1"/>
      <NumericPredictor coefficient="2" exponent="1" name="fld2"/>
      <CategoricalPredictor coefficient="-3" name="fld3" value="x"/>
      <CategoricalPredictor coefficient="3" name="fld3" value="y"/>
      <PredictorTerm coefficient="0.4">
        <FieldRef field="fld1"/>
        <FieldRef field="fld2"/>
      </PredictorTerm>
    </RegressionTable>
  </RegressionModel>
</PMML>
```

-

PMML Scorecard 模型示例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.dmg.org/PMML-4_2">
  <Header copyright="JBoss"/>
  <DataDictionary numberOfFields="4">
    <DataField name="param1" optype="continuous" dataType="double"/>
    <DataField name="param2" optype="continuous" dataType="double"/>
    <DataField name="overallScore" optype="continuous" dataType="double" />
    <DataField name="finalscore" optype="continuous" dataType="double" />
  </DataDictionary>
  <Scorecard modelName="ScorecardCompoundPredicate" useReasonCodes="true" isScorable="true" functionName="regression"    baselineScore="15" initialScore="0.8" reasonCodeAlgorithm="pointsAbove">
    <MiningSchema>
      <MiningField name="param1" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="param2" usageType="active" invalidValueTreatment="asMissing">
      </MiningField>
      <MiningField name="overallScore" usageType="target"/>
      <MiningField name="finalscore" usageType="predicted"/>
    </MiningSchema>
    <Characteristics>
      <Characteristic name="ch1" baselineScore="50" reasonCode="reasonCh1">
        <Attribute partialScore="20">
          <SimplePredicate field="param1" operator="lessThan" value="20"/>
        </Attribute>
        <Attribute partialScore="100">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="lessOrEqual" value="25"/>
          </CompoundPredicate>
        </Attribute>
        <Attribute partialScore="200">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="param1" operator="greaterOrEqual" value="20"/>
            <SimplePredicate field="param2" operator="greaterThan" value="25"/>
          </CompoundPredicate>
        </Attribute>
      </Characteristic>
      <Characteristic name="ch2" reasonCode="reasonCh2">
        <Attribute partialScore="10">
          <CompoundPredicate booleanOperator="or">
            <SimplePredicate field="param2" operator="lessOrEqual" value="-5"/>
            <SimplePredicate field="param2" operator="greaterOrEqual" value="50"/>
          </CompoundPredicate>
        </Attribute>
```

```
      <Attribute partialScore="20">
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="param2" operator="greaterThan" value="-5"/>
          <SimplePredicate field="param2" operator="lessThan" value="50"/>
        </CompoundPredicate>
      </Attribute>
    </Characteristic>
  </Characteristics>
 </Scorecard>
</PMML>
```

PMML Tree 模型示例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-
1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.dmg.org/PMML-4_2">
 <Header copyright="JBOSS"/>
 <DataDictionary numberOfFields="5">
  <DataField dataType="double" name="fld1" optype="continuous"/>
  <DataField dataType="double" name="fld2" optype="continuous"/>
  <DataField dataType="string" name="fld3" optype="categorical">
   <Value value="true"/>
   <Value value="false"/>
  </DataField>
  <DataField dataType="string" name="fld4" optype="categorical">
   <Value value="optA"/>
   <Value value="optB"/>
   <Value value="optC"/>
  </DataField>
  <DataField dataType="string" name="fld5" optype="categorical">
   <Value value="tgtX"/>
   <Value value="tgtY"/>
   <Value value="tgtZ"/>
  </DataField>
 </DataDictionary>
 <TreeModel functionName="classification" modelName="TreeTest">
  <MiningSchema>
   <MiningField name="fld1"/>
   <MiningField name="fld2"/>
   <MiningField name="fld3"/>
   <MiningField name="fld4"/>
   <MiningField name="fld5" usageType="predicted"/>
  </MiningSchema>
  <Node score="tgtX">
   <True/>
   <Node score="tgtX">
    <SimplePredicate field="fld4" operator="equal" value="optA"/>
    <Node score="tgtX">
     <CompoundPredicate booleanOperator="surrogate">
```

```
          <SimplePredicate field="fld1" operator="lessThan" value="30.0"/>
          <SimplePredicate field="fld2" operator="greaterThan" value="20.0"/>
        </CompoundPredicate>
        <Node score="tgtX">
          <SimplePredicate field="fld2" operator="lessThan" value="40.0"/>
        </Node>
        <Node score="tgtZ">
          <SimplePredicate field="fld2" operator="greaterOrEqual" value="10.0"/>
        </Node>
      </Node>
      <Node score="tgtZ">
        <CompoundPredicate booleanOperator="or">
          <SimplePredicate field="fld1" operator="greaterOrEqual" value="60.0"/>
          <SimplePredicate field="fld1" operator="lessOrEqual" value="70.0"/>
        </CompoundPredicate>
        <Node score="tgtZ">
          <SimpleSetPredicate booleanOperator="isNotIn" field="fld4">
            <Array type="string">optA optB</Array>
          </SimpleSetPredicate>
        </Node>
      </Node>
    </Node>
    <Node score="tgtY">
      <CompoundPredicate booleanOperator="or">
        <SimplePredicate field="fld4" operator="equal" value="optA"/>
        <SimplePredicate field="fld4" operator="equal" value="optC"/>
      </CompoundPredicate>
      <Node score="tgtY">
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="fld1" operator="greaterThan" value="10.0"/>
          <SimplePredicate field="fld1" operator="lessThan" value="50.0"/>
          <SimplePredicate field="fld4" operator="equal" value="optA"/>
          <SimplePredicate field="fld2" operator="lessThan" value="100.0"/>
          <SimplePredicate field="fld3" operator="equal" value="false"/>
        </CompoundPredicate>
      </Node>
      <Node score="tgtZ">
        <CompoundPredicate booleanOperator="and">
          <SimplePredicate field="fld4" operator="equal" value="optC"/>
          <SimplePredicate field="fld2" operator="lessThan" value="30.0"/>
        </CompoundPredicate>
      </Node>
    </Node>
  </Node>
 </TreeModel>
</PMML>
```

PMML Mining model(modelChain)示例

```
<PMML version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.dmg.org/PMML-4_2">
  <Header>
    <Application name="Drools-PMML" version="7.0.0-SNAPSHOT" />
  </Header>
  <DataDictionary numberOfFields="7">
    <DataField name="age" optype="continuous" dataType="double" />
    <DataField name="occupation" optype="categorical" dataType="string">
      <Value value="SKYDIVER" />
      <Value value="ASTRONAUT" />
      <Value value="PROGRAMMER" />
      <Value value="TEACHER" />
      <Value value="INSTRUCTOR" />
    </DataField>
    <DataField name="residenceState" optype="categorical" dataType="string">
      <Value value="AP" />
      <Value value="KN" />
      <Value value="TN" />
    </DataField>
    <DataField name="validLicense" optype="categorical" dataType="boolean" />
    <DataField name="overallScore" optype="continuous" dataType="double" />
    <DataField name="grade" optype="categorical" dataType="string">
      <Value value="A" />
      <Value value="B" />
      <Value value="C" />
      <Value value="D" />
      <Value value="F" />
    </DataField>
    <DataField name="qualificationLevel" optype="categorical" dataType="string">
      <Value value="Unqualified" />
      <Value value="Barely" />
      <Value value="Well" />
      <Value value="Over" />
    </DataField>
  </DataDictionary>
  <MiningModel modelName="SampleModelChainMine" functionName="classification">
    <MiningSchema>
      <MiningField name="age" />
      <MiningField name="occupation" />
      <MiningField name="residenceState" />
      <MiningField name="validLicense" />
      <MiningField name="overallScore" />
      <MiningField name="qualificationLevel" usageType="target"/>
    </MiningSchema>
    <Segmentation multipleModelMethod="modelChain">
      <Segment id="1">
        <True />
        <Scorecard modelName="Sample Score 1" useReasonCodes="true" isScorable="true" functionName="regression"              baselineScore="0.0" initialScore="0.345">
          <MiningSchema>
            <MiningField name="age" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="occupation" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="residenceState" usageType="active" invalidValueTreatment="asMissing" />
            <MiningField name="validLicense" usageType="active" invalidValueTreatment="asMissing" />
```

```
        <MiningField name="overallScore" usageType="predicted" />
      </MiningSchema>
      <Output>
        <OutputField name="calculatedScore" displayName="Final Score" dataType="double"
feature="predictedValue"                targetField="overallScore" />
      </Output>
      <Characteristics>
        <Characteristic name="AgeScore" baselineScore="0.0" reasonCode="ABZ">
          <Extension name="cellRef" value="$B$8" />
          <Attribute partialScore="10.0">
            <Extension name="cellRef" value="$C$10" />
            <SimplePredicate field="age" operator="lessOrEqual" value="5" />
          </Attribute>
          <Attribute partialScore="30.0" reasonCode="CX1">
            <Extension name="cellRef" value="$C$11" />
            <CompoundPredicate booleanOperator="and">
              <SimplePredicate field="age" operator="greaterOrEqual" value="5" />
              <SimplePredicate field="age" operator="lessThan" value="12" />
            </CompoundPredicate>
          </Attribute>
          <Attribute partialScore="40.0" reasonCode="CX2">
            <Extension name="cellRef" value="$C$12" />
            <CompoundPredicate booleanOperator="and">
              <SimplePredicate field="age" operator="greaterOrEqual" value="13" />
              <SimplePredicate field="age" operator="lessThan" value="44" />
            </CompoundPredicate>
          </Attribute>
          <Attribute partialScore="25.0">
            <Extension name="cellRef" value="$C$13" />
            <SimplePredicate field="age" operator="greaterOrEqual" value="45" />
          </Attribute>
        </Characteristic>
        <Characteristic name="OccupationScore" baselineScore="0.0">
          <Extension name="cellRef" value="$B$16" />
          <Attribute partialScore="-10.0" reasonCode="CX2">
            <Extension name="description" value="skydiving is a risky occupation" />
            <Extension name="cellRef" value="$C$18" />
            <SimpleSetPredicate field="occupation" booleanOperator="isIn">
              <Array n="2" type="string">SKYDIVER ASTRONAUT</Array>
            </SimpleSetPredicate>
          </Attribute>
          <Attribute partialScore="10.0">
            <Extension name="cellRef" value="$C$19" />
            <SimpleSetPredicate field="occupation" booleanOperator="isIn">
              <Array n="2" type="string">TEACHER INSTRUCTOR</Array>
            </SimpleSetPredicate>
          </Attribute>
          <Attribute partialScore="5.0">
            <Extension name="cellRef" value="$C$20" />
            <SimplePredicate field="occupation" operator="equal" value="PROGRAMMER" />
          </Attribute>
        </Characteristic>
        <Characteristic name="ResidenceStateScore" baselineScore="0.0" reasonCode="RES">
          <Extension name="cellRef" value="$B$22" />
          <Attribute partialScore="-10.0">
            <Extension name="cellRef" value="$C$24" />
```

```
            <SimplePredicate field="residenceState" operator="equal" value="AP" />
          </Attribute>
          <Attribute partialScore="10.0">
            <Extension name="cellRef" value="$C$25" />
            <SimplePredicate field="residenceState" operator="equal" value="KN" />
          </Attribute>
          <Attribute partialScore="5.0">
            <Extension name="cellRef" value="$C$26" />
            <SimplePredicate field="residenceState" operator="equal" value="TN" />
          </Attribute>
        </Characteristic>
        <Characteristic name="ValidLicenseScore" baselineScore="0.0">
          <Extension name="cellRef" value="$B$28" />
          <Attribute partialScore="1.0" reasonCode="LX00">
            <Extension name="cellRef" value="$C$30" />
            <SimplePredicate field="validLicense" operator="equal" value="true" />
          </Attribute>
          <Attribute partialScore="-1.0" reasonCode="LX00">
            <Extension name="cellRef" value="$C$31" />
            <SimplePredicate field="validLicense" operator="equal" value="false" />
          </Attribute>
        </Characteristic>
      </Characteristics>
    </Scorecard>
  </Segment>
  <Segment id="2">
    <True />
    <TreeModel modelName="SampleTree" functionName="classification"
missingValueStrategy="lastPrediction" noTrueChildStrategy="returnLastPrediction">
      <MiningSchema>
        <MiningField name="age" usageType="active" />
        <MiningField name="validLicense" usageType="active" />
        <MiningField name="calculatedScore" usageType="active" />
        <MiningField name="qualificationLevel" usageType="predicted" />
      </MiningSchema>
      <Output>
        <OutputField name="qualification" displayName="Qualification Level" dataType="string"
feature="predictedValue"                    targetField="qualificationLevel" />
      </Output>
      <Node score="Well" id="1">
        <True/>
        <Node score="Barely" id="2">
          <CompoundPredicate booleanOperator="and">
            <SimplePredicate field="age" operator="greaterOrEqual" value="16" />
            <SimplePredicate field="validLicense" operator="equal" value="true" />
          </CompoundPredicate>
          <Node score="Barely" id="3">
            <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="50.0" />
          </Node>
          <Node score="Well" id="4">
            <CompoundPredicate booleanOperator="and">
              <SimplePredicate field="calculatedScore" operator="greaterThan" value="50.0" />
              <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="60.0" />
            </CompoundPredicate>
          </Node>
          <Node score="Over" id="5">
```

```
            <SimplePredicate field="calculatedScore" operator="greaterThan" value="60.0" />
           </Node>
          </Node>
          <Node score="Unqualified" id="6">
           <CompoundPredicate booleanOperator="surrogate">
            <SimplePredicate field="age" operator="lessThan" value="16" />
            <SimplePredicate field="calculatedScore" operator="lessOrEqual" value="40.0" />
            <True />
           </CompoundPredicate>
          </Node>
         </Node>
        </TreeModel>
      </Segment>
    </Segmentation>
  </MiningModel>
</PMML>
```

PMML 集群模型示例

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML version="4.1" xmlns="http://www.dmg.org/PMML-4_1">
  <Header>
    <Application name="KNIME" version="2.8.0"/>
  </Header>
  <DataDictionary numberOfFields="5">
    <DataField name="sepal_length" optype="continuous" dataType="double">
      <Interval closure="closedClosed" leftMargin="4.3" rightMargin="7.9"/>
    </DataField>
    <DataField name="sepal_width" optype="continuous" dataType="double">
      <Interval closure="closedClosed" leftMargin="2.0" rightMargin="4.4"/>
    </DataField>
    <DataField name="petal_length" optype="continuous" dataType="double">
      <Interval closure="closedClosed" leftMargin="1.0" rightMargin="6.9"/>
    </DataField>
    <DataField name="petal_width" optype="continuous" dataType="double">
      <Interval closure="closedClosed" leftMargin="0.1" rightMargin="2.5"/>
    </DataField>
    <DataField name="class" optype="categorical" dataType="string"/>
  </DataDictionary>
  <ClusteringModel modelName="SingleIrisKMeansClustering" functionName="clustering"
modelClass="centerBased" numberOfClusters="4">
    <MiningSchema>
      <MiningField name="sepal_length" invalidValueTreatment="asIs"/>
      <MiningField name="sepal_width" invalidValueTreatment="asIs"/>
      <MiningField name="petal_length" invalidValueTreatment="asIs"/>
      <MiningField name="petal_width" invalidValueTreatment="asIs"/>
      <MiningField name="class" usageType="predicted"/>
    </MiningSchema>
    <ComparisonMeasure kind="distance">
```

```
      <squaredEuclidean/>
    </ComparisonMeasure>
    <ClusteringField field="sepal_length" compareFunction="absDiff"/>
    <ClusteringField field="sepal_width" compareFunction="absDiff"/>
    <ClusteringField field="petal_length" compareFunction="absDiff"/>
    <ClusteringField field="petal_width" compareFunction="absDiff"/>
    <Cluster name="virginica" size="32">
      <Array n="4" type="real">6.9125000000000005 3.099999999999999 5.846874999999999
2.1312499999999996</Array>
    </Cluster>
    <Cluster name="versicolor" size="41">
      <Array n="4" type="real">6.23658536585366 2.8585365853658535 4.807317073170731
1.6219512195121943</Array>
    </Cluster>
    <Cluster name="setosa" size="50">
      <Array n="4" type="real">5.005999999999999 3.4180000000000006 1.464
0.2439999999999999</Array>
    </Cluster>
    <Cluster name="unknown" size="27">
      <Array n="4" type="real">5.529629629629629 2.6222222222222222 3.940740740740741
1.2185185185185188</Array>
    </Cluster>
  </ClusteringModel>
</PMML>
```

第 12 章 RED HAT PROCESS AUTOMATION MANAGER 中的 PMML 支持

Red Hat Process Automation Manager 包括对以下 PMML 模型类型的消费者一致支持：

- 回归模型
- Scorecard 模型
- 树形模型
- 最小模型 （使用子类型 模型Chain，选择All，然后选择First）
- 集群模型

有关所有 PMML 模型类型的列表，包括 Red Hat Process Automation Manager 中不支持的项，请参阅 DMG PMML 规格。

Red Hat Process Automation Manager 提供了两种实现，包括 PMML 传统和 PMML 信任。

重要

PMML 传统实现在 Red Hat Process Automation Manager 7.10.0 中已弃用，并将在以后的 Red Hat Process Automation Manager 发行版本中被 PMML 信任实现替代。

Red Hat Process Automation Manager 不包括内置的 PMML 模型编辑器，但您可以使用 XML 或 PMML 特定的授权工具创建 PMML 模型，然后在 Red Hat Process Automation Manager 中的决策服务中集成 PMML 模型。您可以在 Business Central(Menu → Design → Projects → Import Asset)中导入 PMML 文件，或将 PMML 文件打包为项目知识 JAR(KJAR)文件的一部分，而无需 Business Central。

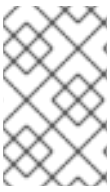
有关使用项目打包和部署方法包括 PMML 文件的更多信息，请参阅打包和部署 *Red Hat Process Automation Manager 项目*。

您可以将 PMML 服务迁移到红帽构建的 Kogito 微服务。有关迁移到红帽构建的 Kogito 微服务的更多信息，请参阅 *迁移到红帽构建的 Kogito 微服务*。

12.1. RED HAT PROCESS AUTOMATION MANAGER 中的 PMML 信任支持和命名约定

当您将 PMML 文件添加到 Red Hat Process Automation Manager 中的项目时，会生成多个资产。tree 和 scorecard 模型转换为规则，回归和最小模型被转换为 Java 类。PMML 模型的每种类型可生成不同的资产集合，但所有 PMML 模型类型至少会产生以下一组资产：

- 派生自 PMML 文件名的根软件包
- 在 root 软件包中，用于实例化模型的 Java 工厂类
- 特定于名称从模型名称派生的模型的子软件包
- 对于规则模型，用于实例化 规则 网络的两个规则映射类
- 对于最小模型，子模型软件包和类嵌套在父模型中



注意

目前，每个 PMML 文件只允许一个模型。另外，不支持扩展。

以下是生成的 PMML 软件包和类的命名约定：

- root 软件包名称是原始 PMML 文件的名称（小写），且没有空格，例如 sampleregression。
- 生成的 工厂 Java 类的名称是 PMML 文件名，其格式为 fileName+" Factory " 和第一个大写字母，例如 SampleRegressionFactory。
- 模型的子软件包名称是原始模型的名称，没有空格，例如：compound

nestedpredicatescorecard。

- 生成的数据类型的名称由模型类型决定：
  - 规则模型：生成顶层 PMMLRuleMappersImpl，包括对子软件包中嵌套的 PMMLRuleMapperImpl 类的引用。
  - 最小模型：
    - 创建的分段子软件包的名称是原始模型的名称，没有空格，并且 分段 以 modelName+" segmentation " 添加到其中，例如 混合分段。
    - 在 分段 子软件包中，会创建一个 分段 Java 类，其中包含对嵌套模型的引用。创建 的分段 Java 类的名称是添加 Segmentation 的模型名称，格式为 modelName+Segmentation，例如 MixedMiningSegmentation。
    - 对于每个片段，会创建一个特定的子软件包。段特定子软件包的名称是小写的原始模型名称，从 0 开始，以 modelName+ segment +integer 格式添加。例如，混合 segment0，混合于segment1。

PMML 信任实现的已知限制

以下列表显示了没有为 PMML 信任实现的元素：

- 目标 元素没有实现
- 未实施 extension 元素
- 未实现的 MiningSchema 或 MiningField 元素包括：
  - 重要
  - outliers

- lowValue
- highValue
- invalidValueTreatment
- invalidValueReplacement

- 输出不实现的 OutputField 元素包括：
  - 决策
  - 值
  - 规则功能
  - algorithm
  - isMultiValued
  - segmentId
  - isFinalResult

- 不支持的 TransformationDictionary 或 LocalTransformation 表达式包括：
  - NormContinuous

- NormDiscrete
- MapValues
- TextIndex
- 聚合
- lag

- ModelStats、ModelExplanation 和 ModelExplanation 元素不会在所有模型中实施，包括回归、树、scorecard 和 mining
- 验证 元素不在树、scorecard 和 mining 模型中实施
- VariableWeight 元素在 mining 模型中没有实施
- 未实现的树模型元素，包括：
  - IsMissing 或 IsNotMissing
  - Surrogate in CompoundPredicate
  - missingValuePenalty
  - splitCharacteristic
  - isScorable

12.2. RED HAT PROCESS AUTOMATION MANAGER 中的 PMML 传统支持和命名约定

当您将 PMML 文件添加到 Red Hat Process Automation Manager 中的项目时，会生成多个资产。PMML 模型的每种类型可生成不同的资产集合，但所有 PMML 模型类型至少会产生以下一组资产：

- 包含与 PMML 模型关联的所有规则的 DRL 文件
- 至少两个 Java 类：
  - 用作模型类型的默认对象类型的数据卷
  - RuleUnit 类用于管理数据源和规则执行

如果 PMML 文件使用 MiningModel 作为 root 模型，则生成每个文件的多个实例。

以下是生成的 PMML 传统软件包、类和规则的命名约定：

- 如果在 PMML 型号文件中未给出软件包名称，则默认软件包名称 org.kie.pmml.pmml_4_2 为生成的规则的模型名称作为前缀，格式为 "org.kie.pmml_4_2"+modelName。
- 生成的 RuleUnit Java 类的软件包名称与生成规则的软件包名称相同。
- 生成的 RuleUnit Java 类的名称是以 modelName+" RuleUnit " 格式添加到其中的 RuleUnit 的模型名称。
- 每个 PMML 模型至少生成一个数据类。这些类的软件包名称是 org.kie.pmml.pmml_4_2.model。
- 生成数据类的名称由模型类型决定，前缀为模型名称：
  - 回归模型：一个名为 modelName+"RegressionData"的数据类型。

- Scorecard 模型：一个名为 modelName+ 的数据卷"ScoreCardData"
- 树结构模型：两个数据类型，第一个命名的 modelName+"TreeNode" 和第二个名为 modelName+"TreeToken"
- Mining model: 一个名为 modelName+ 的数据类型"MiningModelData"

注意

最小模型还会生成每个片段中的所有规则和类。

12.2.1. Red Hat Process Automation Manager 中的 PMML 扩展

PMML 传统规格支持 扩展 PMML 模型内容的扩展元素。您可以在几乎每个 PMML 模型定义级别使用扩展，作为模型主要元素中的第一个和最后一个子项，以获得最大灵活性。有关 PMML 扩展的更多信息，请参阅 DMG PMML 扩展机制。

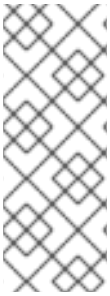
要优化 PMML 集成，Red Hat Process Automation Manager 支持以下额外 PMML 扩展：

- modelPackage ：为生成的规则和 Java 类设计软件包名称。将此扩展包含在 PMML 模型文件的 Header 部分中。
- adapter ：指定用于包含规则输入和输出数据的构造类型（bean 或 trait）。在 PMML 模型文件的 MiningSchema 或 Output 部分（或两者）中插入此扩展。
- externalClass ：与 适配器 扩展结合使用，在定义 MiningField 或 OutputField 中使用。此扩展包含一个类，其中包含名称与 MiningField 或 OutputField 元素的名称匹配的属性。

第 13 章 PMML 模型执行

您可以使用 Business Central(Menu → Design → Projects → Import Asset)将 PMML 文件导入到 Red Hat Process Automation Manager 项目中，或者在没有 Business Central 的情况下将 PMML 文件打包为您的项目知识 JAR(KJAR)文件的一部分。在 Red Hat Process Automation Manager 项目中实现 PMML 文件后，您可以通过直接嵌入 Java 应用程序中的 PMML 调用，或将 ApplyPmmlModelCommand 命令发送到配置的 KIE 服务器来执行 PMML 决策服务。

有关使用项目打包和部署方法包括 PMML 资产的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。



注意

您还可以将 PMML 模型作为 Business Central 中决策模型和 Notation(DMN)服务的一部分。当您在 DMN 文件中包含 PMML 模型时，您可以将 PMML 模型作为作为 DMN 决策节点或商业知识节点的 box 功能表达式调用。有关在 DMN 服务中包含 PMML 模型的更多信息，*请参阅使用 DMN 模型设计决策服务*。

13.1. 直接在 JAVA 应用程序中嵌入 PMML 信任调用

当知识资产直接嵌入到调用程序中，或者被实际拉取为 KJAR 的 Maven 依赖关系时，KIE 容器是本地的。如果代码版本和 PMML 定义版本之间有紧密关系，则您直接将知识资产嵌入到项目中。在有意更新并重新部署应用程序后，对决策的任何更改都会生效。这种方法的一个优点是，正确的操作不依赖于任何外部依赖项来运行，这可能受锁定的环境限制。

先决条件

- 已创建包含要执行的 PMML 模型的 KJAR。有关项目打包的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

流程

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

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml-dependencies</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
```

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

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

& lt;version > 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（例如 7.67.0.Final-redhat-00024）。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

2. 从 类路径 或 ReleaseId 创建 KIE 容器：

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

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

备选选项：

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

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

3. 创建用于执行模型的 PMMLRuntime 实例：

```
PMMLRuntime pmmlRuntime =
KieRuntimeFactory.of(kieContainer.getKieBase()).get(PMMLRuntime.class);
```

4. 创建一个 PMMLRequestData 类实例，将 PMML 模型应用到数据集：

```
PMMLRequestData pmmlRequestData = new PMMLRequestData({correlation_id},
{model_name});
pmmlRequestData.addRequestParam({parameter_name}, {parameter_value})
...
```

5. 创建包含输入数据的 PMMLContext 类的实例：

```
PMMLContext pmmlContext = new PMMLContextImpl(pmmlRequestData);
```

6. 在使用您创建的所需的 PMML 类实例执行 PMML 模型时，检索 PMML4Result ：

```
PMML4Result pmml4Result = pmmlRuntime.evaluate({model_name}, pmmlContext);
```

13.2. 直接嵌入 JAVA 应用程序中的 PMML 传统调用

当知识资产直接嵌入到调用程序中，或者被实际拉取为 KJAR 的 Maven 依赖关系时，KIE 容器是本地的。如果代码版本和 PMML 定义版本之间有紧密关系，则您直接将知识资产嵌入到项目中。在有意更新并重新部署应用程序后，对决策的任何更改都会生效。这种方法的一个优点是，正确的操作不依赖于任何外部依赖项来运行，这可能受锁定的环境限制。

使用 Maven 依赖项可进行进一步的灵活性，因为决策的特定版本可以动态更改（例如，使用系统属性），它可以定期扫描更新并自动更新。这会对服务的部署时间进行外部依赖，但在本地执行决策，从而减少在运行期间对外部服务的依赖。

先决条件

- 已创建包含要执行的 PMML 模型的 KJAR。有关项目打包的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

流程

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

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

& lt;version > 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（例如 7.67.0.Final-redhat-00024）。

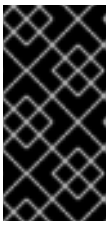
注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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



重要

要使用传统实施，请确保将 kie-pmml-ement ation 系统属性设置为 legacy。

2. 从 类路径 或 ReleaseId 创建 KIE 容器：

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

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

备选选项：

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

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

3. 创建 PMMLRequestData 类的实例，它将您的 PMML 模型应用于一组数据：

```
public class PMMLRequestData {
   private String correlationId; (1)
   private String modelName; (2)
   private String source; (3)
   private List<ParameterInfo<?>> requestParams; (4)
   ...
}
```

(1) 标识与特定请求或结果关联的数据

(2) 应该应用到请求数据的模型名称

(3) 由内部生成的 PMMLRequestData 对象用来识别生成请求的片段

(4) 发送输入数据点的默认机制

4. 创建 PMML4Result 类的实例，其中包含将 PMML 的规则应用到输入数据的输出信息：

```
public class PMML4Result {
   private String correlationId;
   private String segmentationId; (1)
   private String segmentId; (2)
   private int segmentIndex; (3)
   private String resultCode; (4)
   private Map<String, Object> resultVariables; (5)
   ...
}
```

(1) 当模型类型为 MiningModel 时使用。segmentationId 用于区分多个分段。

(2) 与 segmentationId 结合使用来识别生成结果的片段。

(3)

用于维护网段的顺序。

4

用于确定模型是否已成功应用，其中 OK 指示成功。

5

包含结果变量的名称及其关联的值。

除了普通的 getter 方法外，M PMML4Result 类还支持以下方法直接检索结果变量的值：

```
public <T> Optional<T> getResultValue(String objName, String objField, Class<T>
clazz, Object...params)

public Object getResultValue(String objName, String objField, Object...params)
```

5. 创建 ParameterInfo 类的实例，作为作为 PMMLRequestData 类一部分的基本数据类型对象的打包程序：

```
public class ParameterInfo<T> { 1
   private String correlationId;
   private String name; 2
   private String capitalizedName;
   private Class<T> type; 3
   private T value; 4
   ...
}
```

1

参数化类来处理许多不同的类型

2

预期作为型号输入的变量名称

3

是变量的实际类型的类

4

6. 根据您创建的所需的 PMML 类实例执行 PMML 模型：

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
   RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
   PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
   PMML4Result resultHolder = new PMML4Result(correlationId);
   variables.entrySet().forEach( es -> {
      request.addRequestParam(es.getKey(), es.getValue());
   });

   DataSource<PMMLRequestData> requestData =
executor.newDataSource("request");
   DataSource<PMML4Result> resultData = executor.newDataSource("results");
   DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

   requestData.insert(request);
   resultData.insert(resultHolder);

   List<String> possiblePackageNames =
calculatePossiblePackageNames(modelName,
                                                 modelPkgName);
   Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                             (InternalKnowledgeBase)kbase,
                                             possiblePackageNames);

   if (ruleUnitClass != null) {
      executor.run(ruleUnitClass);
      if ( "OK".equals(resultHolder.getResultCode()) ) {
        // extract result variables here
      }
   }
}

protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
   RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
   Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
   RuleImpl ruleImpl = null;
   for (String pkgName: possiblePackages) {
    if (pkgs.containsKey(pkgName)) {
       InternalKnowledgePackage pkg = pkgs.get(pkgName);
       ruleImpl = pkg.getRule(startingRule);
       if (ruleImpl != null) {
          RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
          if (descr != null) {
             return descr.getRuleUnitClass();
          }
```

```
        }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
    String basePkgName =
PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
}
```

规则由 RuleUnitExecutor 类执行。RuleUnitExecutor 类将创建 KIE 会话并将所需的 DataSource 对象添加到这些会话，然后基于 RuleUnit 来执行规则，后者作为参数传递到 run（） 方法。calculatePossiblePackageNames 和 getStartingRuleUnit 方法决定了传递给 run（） 方法的 RuleUnit 类的完全限定名称。

为方便您的 PMML 模型执行，您还可以使用 Red Hat Process Automation Manager 支持的 PMML4ExecutionHelper 类。有关 PMML 帮助程序类的更多信息，请参阅 第 13.2.1 节 “PMML 执行帮助程序类”。

13.2.1. PMML 执行帮助程序类

Red Hat Process Automation Manager 提供了一个 PMML4ExecutionHelper 类，它可帮助创建 PMML 模型执行所需的 PMMLRequestData 类，并帮助使用 RuleUnitExecutor 类执行规则。

以下是在没有时间和 PMML4ExecutionHelper 类的情况下执行的 PMML 模型示例，作为比较：

在不使用 PMML4ExecutionHelper的情况下执行 PMML 模型示例

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String correlationId,
                         String modelPkgName) {
    RuleUnitExecutor executor = RuleUnitExecutor.create().bind(kbase);
```

```
    PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
    PMML4Result resultHolder = new PMML4Result(correlationId);
    variables.entrySet().forEach( es -> {
        request.addRequestParam(es.getKey(), es.getValue());
    });

    DataSource<PMMLRequestData> requestData = executor.newDataSource("request");
    DataSource<PMML4Result> resultData = executor.newDataSource("results");
    DataSource<PMMLData> internalData = executor.newDataSource("pmmlData");

    requestData.insert(request);
    resultData.insert(resultHolder);

    List<String> possiblePackageNames = calculatePossiblePackageNames(modelName,
                                                    modelPkgName);
    Class<? extends RuleUnit> ruleUnitClass = getStartingRuleUnit("RuleUnitIndicator",
                                                (InternalKnowledgeBase)kbase,
                                                possiblePackageNames);

    if (ruleUnitClass != null) {
        executor.run(ruleUnitClass);
        if ( "OK".equals(resultHolder.getResultCode()) ) {
          // extract result variables here
        }
    }
}

protected Class<? extends RuleUnit> getStartingRuleUnit(String startingRule,
InternalKnowledgeBase ikb, List<String> possiblePackages) {
    RuleUnitRegistry unitRegistry = ikb.getRuleUnitRegistry();
    Map<String,InternalKnowledgePackage> pkgs = ikb.getPackagesMap();
    RuleImpl ruleImpl = null;
    for (String pkgName: possiblePackages) {
      if (pkgs.containsKey(pkgName)) {
          InternalKnowledgePackage pkg = pkgs.get(pkgName);
          ruleImpl = pkg.getRule(startingRule);
          if (ruleImpl != null) {
              RuleUnitDescr descr = unitRegistry.getRuleUnitFor(ruleImpl).orElse(null);
              if (descr != null) {
                  return descr.getRuleUnitClass();
              }
          }
      }
    }
    return null;
}

protected List<String> calculatePossiblePackageNames(String modelId,
String...knownPackageNames) {
    List<String> packageNames = new ArrayList<>();
    String javaModelId = modelId.replaceAll("\\s","");
    if (knownPackageNames != null && knownPackageNames.length > 0) {
        for (String knownPkgName: knownPackageNames) {
            packageNames.add(knownPkgName + "." + javaModelId);
        }
    }
```

```
    String basePkgName = PMML4UnitImpl.DEFAULT_ROOT_PACKAGE+"."+javaModelId;
    packageNames.add(basePkgName);
    return packageNames;
  }
```

使用 PMML4ExecutionHelper进行 PMML 模型执行示例

```
public void executeModel(KieBase kbase,
                         Map<String,Object> variables,
                         String modelName,
                         String modelPkgName,
                         String correlationId) {
  PMML4ExecutionHelper helper =
PMML4ExecutionHelperFactory.getExecutionHelper(modelName, kbase);
  helper.addPossiblePackageName(modelPkgName);

  PMMLRequestData request = new PMMLRequestData(correlationId, modelName);
  variables.entrySet().forEach(entry -> {
    request.addRequestParam(entry.getKey(), entry.getValue);
  });

  PMML4Result resultHolder = helper.submitRequest(request);
  if ("OK".equals(resultHolder.getResultCode)) {
    // extract result variables here
  }
}
```

当您使用 PMML4ExecutionHelper 时，您不需要像典型的 PMML 模型执行一样指定可能的软件包名称或 RuleUnit 类。

要构建 PMML4ExecutionHelper 类，请使用 PMML4ExecutionHelper class 确定如何检索 PMML4ExecutionHelper 的实例。

以下是构建 PMML4ExecutionHelper 类类方法的 PMML4ExecutionHelper 类方法：

pmML4ExecutionHelperFactory 方法，用于 KIE 基础中的 PMML 资产

当 PMML 资产已编译并从现有 KIE 基础中使用这些方法：

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase
kbase)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, KieBase
kbase, boolean includeMiningDataSources)
```

pmML4ExecutionHelperFactory 方法用于项目类路径上的 PMML 资产

当 PMML 资产位于项目类路径上时，请使用以下方法。classPath 参数是 PMML 文件的项目类路径位置：

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName,  String
classPath, KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName,String
classPath, KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

PMML4ExecutionHelperFactory 方法用于字节阵列中的 PMML 资产

当 PMML 资产采用字节阵列的形式时，请使用以下方法：

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[]
content, KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, byte[]
content, KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

资源中的 PMML4ExecutionHelper factory 方法，用于资源中的 PMML 资产

当 PMML 资产采用 org.kie.api.io.Resource 对象的形式时，请使用以下方法：

```
public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource
resource, KieBaseConfiguration kieBaseConf)

public static PMML4ExecutionHelper getExecutionHelper(String modelName, Resource
resource, KieBaseConfiguration kieBaseConf, boolean includeMiningDataSources)
```

注意

classpath、byte 数组和资源 PMML4ExecutionHelperFactory 方法为生成的规则和 Java 类创建 KIE 容器。容器用作 RuleUnitExecutor 使用的 KIE 基础的源。容器没有保留。在 KIE 基本中已有 PMML4ExecutionHelperFactory 方法的 PMML4ExecutionHelperFactory 方法不会以这种方式创建 KIE 容器。

13.3. 使用 KIE 服务器执行 PMML 模式

您可以通过将 ApplyPmmlModelCommand 命令发送到配置的 KIE 服务器来执行部署到 KIE 服务器的 PMML 型号。当您使用此命令时，PMMLRequestData 对象发送到 KIE 服务器，并将一个 PMML4Result 结果对象作为回复接收。您可以从配置的 Java 类或从 REST 客户端直接向 KIE 服务器 REST API 发送 PMML 请求到 KIE 服务器。

先决条件

- KIE 服务器是安装和配置的，包括具有 kie-server 角色的用户的已知用户名和凭证。有关安装选项，请参阅 *规划 Red Hat Process Automation Manager 安装*。
- KIE 容器以包括 PMML 模型的 KJAR 的形式部署。有关项目打包的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。
- 您有包含 PMML 模型的容器 ID。

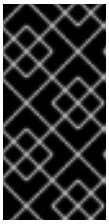
流程

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

传统实现示例

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required for the KIE Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required if not using classpath KIE container -->
```

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```



重要

要使用传统实施，请确保将 **kie-pmml-ement ation** 系统属性设置为 **legacy**。

信任实施示例

```
<!-- Required for the PMML compiler -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>kie-pmml-dependencies</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required for the KIE public API -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-api</artifactId>
  <version>${rhpam.version}</version>
</dependencies>

<!-- Required for the KIE Server Java client API -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- Required if not using classpath KIE container -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

& lt;version > 是项目中当前使用的 Red Hat Process Automation Manager 的 Maven 工件版本（例如 7.67.0.Final-redhat-00024）。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

2. 从 类路径 或 ReleaseId 创建 KIE 容器：

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

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "my-kjar", "1.0.0" );
KieContainer kieContainer = kieServices.newKieContainer( releaseId );
```

备选选项：

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

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

3. 创建一个类来向 KIE 服务器发送请求并接收响应：

```
public class ApplyScorecardModel {
  private static final ReleaseId releaseId =
```

```
        new ReleaseId("org.acme","my-kjar","1.0.0");
  private static final String containerId = "SampleModelContainer";
  private static KieCommands commandFactory;
  private static ClassLoader kjarClassLoader; (1)
  private RuleServicesClient serviceClient; (2)

  // Attributes specific to your class instance
  private String rankedFirstCode;
  private Double score;

  // Initialization of non-final static attributes
  static {
    commandFactory = KieServices.Factory.get().getCommands();

    // Specifications for kjarClassLoader, if used
    KieMavenRepository kmp = KieMavenRepository.getMavenRepository();
    File artifactFile = kmp.resolveArtifact(releaseId).getFile();
    if (artifactFile != null) {
      URL urls[] = new URL[1];
      try {
        urls[0] = artifactFile.toURI().toURL();
        classLoader = new
KieURLClassLoader(urls,PMML4Result.class.getClassLoader());
      } catch (MalformedURLException e) {
        logger.error("Error getting classLoader for "+containerId);
        logger.error(e.getMessage());
      }
    } else {
      logger.warn("Did not find the artifact file for "+releaseId.toString());
    }
  }

  public ApplyScorecardModel(KieServicesConfiguration kieConfig) {
    KieServicesClient clientFactory =
KieServicesFactory.newKieServicesClient(kieConfig);
    serviceClient = clientFactory.getServicesClient(RuleServicesClient.class);
  }
  ...
  // Getters and setters
  ...

  // Method for executing the PMML model on KIE Server
  public void applyModel(String occupation, int age) {
    PMMLRequestData input = new PMMLRequestData("1234","SampleModelName");
(3)
    input.addRequestParam(new
ParameterInfo("1234","occupation",String.class,occupation));
    input.addRequestParam(new ParameterInfo("1234","age",Integer.class,age));

    CommandFactoryServiceImpl cf = (CommandFactoryServiceImpl)commandFactory;
    ApplyPmmlModelCommand command = (ApplyPmmlModelCommand)
cf.newApplyPmmlModel(request); (4)

    ServiceResponse<ExecutionResults> results =
      ruleClient.executeCommandsWithResults(CONTAINER_ID, command); (5)
```

```
    if (results != null) {  6
      PMML4Result resultHolder = (PMML4Result)results.getResult().getValue("results");
      if (resultHolder != null && "OK".equals(resultHolder.getResultCode())) {
        this.score = resultHolder.getResultValue("ScoreCard","score",Double.class).get();
        Map<String,Object> rankingMap =
            (Map<String,Object>)resultHolder.getResultValue("ScoreCard","ranking");
        if (rankingMap != null && !rankingMap.isEmpty()) {
          this.rankedFirstCode = rankingMap.keySet().iterator().next();
        }
      }
    }
  }
}
```

1. 如果您没有在客户端项目依赖项中包含 KJAR，请定义类加载程序

2. 标识配置设置中定义的服务客户端，包括 KIE 服务器 REST API 访问凭证

3. 初始化 PMMLRequestData 对象

4. 创建 ApplyPmmlModelCommand的实例

5. 使用服务客户端发送命令

6. 检索执行的 PMML 模型的结果

4. 执行类实例，以将 PMML 调用请求发送到 KIE 服务器。

或者，也可以使用 JMS 和 REST 接口将 ApplyPmmlModelCommand 命令发送到 KIE 服务器。对于 REST 请求，您可以使用 ApplyPmmlModelCommand 命令作为 JSON、JAXB 或 XStream 请求格式的 POST 请求。

POST 端点示例

```
http://localhost:8080/kie-
server/services/rest/server/containers/instances/SampleModelContainer
```

JSON 请求正文示例

```
{
  "commands": [ {
    "apply-pmml-model-command": {
      "outIdentifier": null,
      "packageName": null,
      "hasMining": false,
      "requestData": {
        "correlationId": "123",
        "modelName": "SimpleScorecard",
        "source": null,
        "requestParams": [
          {
            "correlationId": "123",
            "name": "param1",
            "type": "java.lang.Double",
            "value": "10.0"
          },
          {
            "correlationId": "123",
            "name": "param2",
            "type": "java.lang.Double",
            "value": "15.0"
          }
        ]
      }
    }
  }
  ]
}
```

使用端点和正文的 curl 请求示例

```
curl -X POST "http://localhost:8080/kie-
server/services/rest/server/containers/instances/SampleModelContainer" -H "accept:
```

```
application/json" -H "content-type: application/json" -d "{ \"commands\": [ { \"apply-pmml-model-command\": { \"outIdentifier\": null, \"packageName\": null, \"hasMining\": false, \"requestData\": { \"correlationId\": \"123\", \"modelName\": \"SimpleScorecard\", \"source\": null, \"requestParams\": [ { \"correlationId\": \"123\", \"name\": \"param1\", \"type\": \"java.lang.Double\", \"value\": \"10.0\" }, { \"correlationId\": \"123\", \"name\": \"param2\", \"type\": \"java.lang.Double\", \"value\": \"15.0\" } ] } } } ]}"
```

JSON 响应示例

```
{
  "results" : [ {
    "value" : {"org.kie.api.pmml.DoubleFieldOutput":{
  "value" : 40.8,
  "correlationId" : "123",
  "segmentationId" : null,
  "segmentId" : null,
  "name" : "OverallScore",
  "displayValue" : "OverallScore",
  "weight" : 1.0
}},
    "key" : "OverallScore"
  }, {
    "value" : {"org.kie.api.pmml.PMML4Result":{
  "resultVariables" : {
    "OverallScore" : {
      "value" : 40.8,
      "correlationId" : "123",
      "segmentationId" : null,
      "segmentId" : null,
      "name" : "OverallScore",
      "displayValue" : "OverallScore",
      "weight" : 1.0
    },
    "ScoreCard" : {
      "modelName" : "SimpleScorecard",
      "score" : 40.8,
      "holder" : {
        "modelName" : "SimpleScorecard",
        "correlationId" : "123",
        "voverallScore" : null,
        "moverallScore" : true,
        "vparam1" : 10.0,
        "mparam1" : false,
        "vparam2" : 15.0,
        "mparam2" : false
      },
      "enableRC" : true,
      "pointsBelow" : true,
      "ranking" : {
```

```
        "reasonCh1" : 5.0,
        "reasonCh2" : -6.0
      }
    }
  },
  "correlationId" : "123",
  "segmentationId" : null,
  "segmentId" : null,
  "segmentIndex" : 0,
  "resultCode" : "OK",
  "resultObjectName" : null
}},
    "key" : "results"
  } ],
  "facts" : [ ]
}
```

第 14 章 其他资源

- PMML 规格
- *打包和部署 Red Hat Process Automation Manager 项目*
- *使用 KIE API 与 Red Hat Process Automation Manager 交互*

部分 III. 使用 DRL 规则设计决策服务

作为业务规则开发人员，您可以使用 Business Central 中的 DRL(Drools Rule Language)设计器来定义业务规则。DRL 规则以自由格式 .drl 文本文件直接定义，而不是像 Business Central 中的其他类型的规则资产一样以指导或 tabular 格式定义。这些 DRL 文件组成了项目的决策服务的核心。

注意

您还可以使用决策模型和 Notation(DMN)模型设计您的决策服务，而不是基于规则或基于表的资产。有关 Red Hat Process Automation Manager 7.13 中的 DMN 支持的详情，请查看以下资源：

- *开始使用决策服务* （逐步教程，带有 DMN 决策服务示例）
- *使用 DMN 模型设计决策服务* (Red Hat Process Automation Manager 中的 DMN 支持和功能视图)

先决条件

- Business Central 中创建了 DRL 规则的空间和项目。每个资产都与分配给一个空间的项目相关联。详情请参阅 *开始使用决策服务。*

第 15 章 红帽流程自动化管理器中的决策资产

Red Hat Process Automation Manager 支持一些可用于为决策服务定义业务决策的资产。每个决策资产都有不同的优势，您可以根据目标和需求，首选使用一个或多个资产的组合。

下表重点介绍红帽流程自动化管理器项目支持的主要决策资产，以帮助您决定或确认在决策服务中定义决策的最佳方法。

表 15.1. Red Hat Process Automation Manager 支持的决策资产

asset	亮点	编写工具	Documentation
决策模型和符号 (DMN)型号	• 决策模型基于由对象管理组 (OMG)定义的符号标准。 • 使用代表部分或所有决策要求图 (DRG)的图形化决策要求图 (DRG)跟踪业务决策流程 • 使用 XML 模式，允许在 DMN 兼容平台间共享 DMN 模型 • 支持 Friendly Enough Expression Language(FEEL)，在 DMN 决策表中定义决策逻辑，以及其他 DMN 框的表达式 • 可以与业务流程模型和符号 (BPMN)流程模型有效集成 • 是创建全面、清晰和稳定的决策流程的最佳选择	Business Central 或其他与 DMN 兼容的编辑器	*使用 DMN 模型设计决策服务*
指导的决定表	• 是您在 Business Central 中基于 UI 的表格设计者创建的规则表 • 是电子表格决策表的向导替代选择 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值 • 支持点击策略、实时验证和其他资产不支持的其他功能 • 是以受控表格格式创建规则的最佳选择，可最小化编译错误	Business Central	*使用指导决策表设计决策服务*

asset	亮点	编写工具	Documentation
电子表格决策表	• 是 XLS 或 XLSX 表格决策表，您可以上传到 Business Central • 支持创建规则模板的模板键和值 • 是创建已在 Business Central 之外管理的决策表中规则的最佳选择 • 在上传后，对规则进行编译具有严格的语法要求	电子表格编辑器	*使用电子表格决策表设计决策服务*
指导规则	• 是您在 Business Central 中基于 UI 的规则设计者创建的单独规则 • 为可接受的输入提供字段和选项 • 是以受控格式创建单一规则的最佳选择，以最小化编译错误	Business Central	*使用指导规则设计决策服务*
指导规则模板	• 是您在 Business Central 中基于 UI 的模板设计者创建的可重复使用的规则结构 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值（与此资产的目的不相同） • 是创建具有相同规则结构但具有不同定义字段值的很多规则的最佳	Business Central	*使用指导规则模板设计决策服务*
DRL 规则	• 是直接在 **.drl** 文本文件中定义的单独规则 • 提供定义规则和其他规则行为的最大的灵活性 • 可以在特定的独立环境中创建，并与 Red Hat Process Automation Manager 集成 • 是创建需要高级 DRL 选项的规则的最佳选择 • 对规则进行正确编译的规则具有严格的语法要求	业务中心或集成开发环境(IDE)	*使用 DRL 规则设计决策服务*

asset	亮点	编写工具	Documentation
预测模型标记语言(PMML)型号	• 是基于数据 Mining Group(DMG)定义的标记标准预测数据分析模型。 • 使用允许 PMML 模式在 PMML 兼容的平台之间共享的 XML 模式 • 支持 Regression、scorecard、Tree、Mining 和其他模型类型 • 可以包含在独立 Red Hat Process Automation Manager 项目中，或者导入到 Business Central 中的项目 • 在 Red Hat Process Automation Manager 中，将预测数据合并为决策服务的最佳选择	PMML 或 XML 编辑器	*使用 PMML 模型设计决策服务*

在定义决策时，您还可以考虑将红帽构建的 Kogito 构建用于云原生决策服务。有关使用红帽构建的 Kogito 微服务的更多信息，请参阅 Red Hat *Process Automation Manager 中的 Red Hat build of Kogito*。

第 16 章 DRL（DROOLS 规则语言）规则

DRL(Drools Rule)规则是您在 .drl 文本文件中直接定义的业务规则。这些 DRL 文件是 Business Central 中所有其他规则资产渲染的源。您可以在 Business Central 界面中创建和管理 DRL 文件，或使用 Red Hat CodeReady Studio 或其他集成开发环境(IDE)在外部创建它们。DRL 文件可以包含一个或多个规则，它们至少定义规则条件（在时）和操作（然后再）。Business Central 中的 DRL 设计器为 Java、DRL 和 XML 提供语法高亮显示。

DRL 文件由以下组件组成：

DRL 文件中的组件

```
package

import

function  // Optional

query  // Optional

declare   // Optional

global   // Optional

rule "rule name"
    // Attributes
    when
        // Conditions
    then
        // Actions
end

rule "rule2 name"

...
```

以下示例 DRL 规则决定了 loan 应用程序决策服务中的年龄限制：

loan Application age 限制的规则示例

```
rule "Underage"
  salience 15
  agenda-group "applicationGroup"
  when
    $application : LoanApplication()
    Applicant( age < 21 )
  then
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

DRL 文件包含单个或多个规则、查询和函数，并可以定义由规则和查询分配和使用的属性等资源声明。DRL 软件包必须在 DRL 文件的顶部列出，规则通常最后列出。所有其他 DRL 组件可遵循任何顺序。

每个规则必须在规则软件包中具有唯一的名称。如果您在软件包中的任何 DRL 文件中使用相同的规则名称多次，则规则无法编译。始终使用双引号括起规则名称（规则"rule name"），以防止可能出现的编译错误，特别是在规则名称中使用空格。

与 DRL 规则相关的所有数据对象都必须位于与 Business Central 中的 DRL 文件相同的项目软件包中。默认导入同一软件包中的资产。其他软件包中的现有资产可以通过 DRL 规则导入。

16.1. DRL 中的软件包

软件包是 Red Hat Process Automation Manager 中相关资产的文件夹，如数据对象、DRL 文件、决策表和其他资产类型。软件包还充当每个规则组的唯一命名空间。单个规则基础可以包含多个软件包。您通常将软件包的所有规则与软件包声明保存在与软件包声明相同的文件中，以便自包含软件包。但是，您可以从规则中使用的其他软件包导入对象。

以下示例是 DRL 文件在 rtgage 应用程序决策服务中的软件包名称和命名空间：

DRL 文件中的软件包定义示例

```
package org.mortgages;
```

16.2. 在 DRL 中导入声明

与在 Java 中导入语句类似，DRL 文件中的导入会识别您要在规则中使用的任何对象的完全限定路径和类型名称。您使用 packageName.objectName 格式指定软件包和数据对象，并在单独的行中指定多个导入。决策引擎自动从 Java 软件包中导入类，其名称与 DRL 软件包相同，并从 java.lang 包中导入。

以下示例是 rtgage 应用程序决策服务中 loan Application 对象的导入声明：

DRL 文件中的导入声明示例

```
import org.mortgages.LoanApplication;
```

16.3. DRL 中的功能

DRL 文件中的功能将语义代码放在规则源文件中，而不是在 Java 类中。如果规则中的某个部分被重复使用，并且每个规则的参数不同，则功能特别有用。在 DRL 文件中的规则上方，您可以声明函数或导入帮助程序类的静态方法作为功能，然后在规则的操作(then)部分中按名称使用函数(then)。

以下示例演示了在 DRL 文件中声明或导入的功能：

带有规则（选项 1）的功能声明示例

```
function String hello(String applicantName) {
    return "Hello " + applicantName + "!";
}

rule "Using a function"
  when
    // Empty
  then
    System.out.println( hello( "James" ) );
end
```

使用规则导入功能示例（选项 2）

```
import function my.package.applicant.hello;

rule "Using a function"
  when
    // Empty
  then
    System.out.println( hello( "James" ) );
end
```

16.4. DRL 中的查询

DRL 文件中的查询搜索决策引擎的工作内存，以查找与 DRL 文件中的规则相关的事实。您可以在 DRL 文件中添加查询定义，然后获取应用程序代码的匹配结果。查询搜索一组定义的条件，且不需要在或 随后 指定规格时使用。查询名称对于 KIE 基础是全局的，因此，在项目中的所有规则查询中必须是唯一的。要返回查询的结果，您可以使用 ksession.get QueryResults ("name") 来构建 QueryResults 定义，其中 "name" 是查询名称。这会返回一个查询结果列表，允许您检索与查询匹配的对象。您可以在 DRL 文件中定义规则之上的查询和查询结果参数。

以下示例是 DRL 文件中的 DRL 文件中的查询定义，其中包含了相关的应用程序代码：

DRL 文件中的查询定义示例

```
query "people under the age of 21"
    $person : Person( age < 21 )
end
```

获取查询结果的应用程序代码示例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
System.out.println( "we have " + results.size() + " people under the age  of 21" );
```

您还可以使用标准 for 循环迭代返回的 QueryResults。每个元素都是一个 QueryResultsRow，可用于访问 tuple 中的每个列。

获取和迭代查询结果的应用程序代码示例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
System.out.println( "we have " + results.size() + " people under the age of 21" );

System.out.println( "These people are under the age of 21:" );

for ( QueryResultsRow row : results ) {
    Person person = ( Person ) row.get( "person" );
    System.out.println( person.getName() + "\n" );
}
```

16.5. DRL 中的类型声明和元数据

DRL 文件中的声明定义了新的事实类型或元数据，以便供 DRL 文件中的规则使用的事实类型：

- 新事实类型： Red Hat Process Automation Manager 的 java.lang 软件包中的默认事实类型是 对象，但您可以根据需要声明 DRL 文件中的其他类型的类型。在 DRL 文件中声明事实类型可让您直接在决策引擎中定义新的事实模型，而无需在 Java 等低级别语言中创建模型。您也可以在已经构建域模型时声明新类型，并且您希望将此模型与主要在原因过程中使用的其他实体补充。

- 事实类型的元数据： 您可以将格式 @key(value) 中的元数据与新的或现有事实相关联。元数据可以是不由事实属性表示的任何数据类型，在该事实类型的所有实例之间是一致的。可在运行时查询元数据，供决策引擎在原因过程中使用。

16.5.1. 在 DRL 中没有元数据的类型声明

新事实的声明不需要任何元数据，但必须包含属性或字段列表。如果类型声明不包含标识属性，则决策引擎会在类路径中搜索现有的事实类，并在缺少类时引发错误。

以下示例是 DRL 文件中没有元数据的新事实类型 Person 声明：

带有规则的新事实类型的声明示例

```
declare Person
  name : String
  dateOfBirth : java.util.Date
  address : Address
end

rule "Using a declared type"
  when
    $p : Person( name == "James" )
  then   // Insert Mark, who is a customer of James.
    Person mark = new Person();
    mark.setName( "Mark" );
    insert( mark );
end
```

在本例中，新事实类型 Person 有三个属性，即dateOfBirth，地址为。 每个属性都有一个类型，可以是任何有效的 Java 类型，包括您创建另一个类或之前声明的事实类型。dateOfBirth 属性类型 java.util.Date、Java API 和 address 属性具有之前定义的事实类型 Address。

为了避免在每次声明时写入类的完全限定名称，您可以将完整类名称定义为 导入 子项的一部分：

导入中具有完全限定类名称的类型声明示例

```
import java.util.Date

declare Person
    name : String
    dateOfBirth : Date
    address : Address
end
```

当您声明新的事实类型时，决策引擎在编译时生成代表事实类型的 Java 类。生成的 Java 类是类型定义的一对一 JavaBeans 映射。

例如，以下 Java 类由示例 Person 类型声明生成：

为 Person fact 类型声明生成的 Java 类

```
public class Person implements Serializable {
    private String name;
    private java.util.Date dateOfBirth;
    private Address address;

    // Empty constructor
    public Person() {...}

    // Constructor with all fields
    public Person( String name, Date dateOfBirth, Address address ) {...}

    // If keys are defined, constructor with keys
    public Person( ...keys... ) {...}

    // Getters and setters
    // `equals` and `hashCode`
    // `toString`
}
```

然后，您可以使用规则中生成的类与其他事实一样，如上例中的 Person 类型声明所示：

使用声明的 Person 事实类型的规则示例

```
rule "Using a declared type"
  when
    $p : Person( name == "James" )
  then   // Insert Mark, who is a customer of James.
    Person mark = new Person();
    mark.setName( "Mark" );
    insert( mark );
end
```

16.5.2. DRL 中的枚举类型声明

DRL 支持以声明 enum <factType> 的格式 声明 enum <factType > 的声明，后跟以分号结尾的值列表。然后，您可以使用 DRL 文件中的规则中的枚举列表。

例如，以下枚举类型声明为员工调度规则定义了一周的天数：

具有调度规则的枚举类型声明示例

```
declare enum DaysOfWeek

SUN("Sunday"),MON("Monday"),TUE("Tuesday"),WED("Wednesday"),THU("Thursday"),FRI("Friday"
),SAT("Saturday");

   fullName : String
end

rule "Using a declared Enum"
when
   $emp : Employee( dayOff == DaysOfWeek.MONDAY )
then
   ...
end
```

16.5.3. DRL 中的扩展类型声明

DRL 支持格式类型声明继承，声明 <factType1> 扩展了 <factType2>。要根据 DRL 中声明的子类型扩展 Java 中声明的类型，您可以在没有任何字段的声明中重复父类型。

例如，以下类型声明从顶级 Person 类型扩展 Student 类型，以及 Student 子类型 中的 LongTerm Stude 类型：

扩展类型声明示例

```
import org.people.Person

declare Person end

declare Student extends Person
    school : String
end

declare LongTermStudent extends Student
    years : int
    course : String
end
```

16.5.4. 带有 DRL 中元数据的类型声明

您可以将格式 @key(value)（值是可选的）中的元数据与事实类型或事实属性相关联。元数据可以是不由事实属性表示的任何数据类型，在该事实类型的所有实例之间是一致的。可在运行时查询元数据，供决策引擎在原因过程中使用。在事实类型属性前声明的任何元数据都会被分配给 fact 类型，而您在属性后声明的元数据被分配到该特定属性。

在以下示例中，为 Person 事实类型声明两个元数据属性 @author 和 @dateOfCreation，为 name 属性声明两个元数据项目 @key 和 @maxLength。@key metadata 属性没有所需的值，因此省略括号和值。

事实类型和属性的元数据声明示例

```
import java.util.Date

declare Person
    @author( Bob )
    @dateOfCreation( 01-Feb-2009 )

    name : String @key @maxLength( 30 )
    dateOfBirth : Date
    address : Address
end
```

对于现有类型的元数据属性声明，您可以识别完全限定的类名称，作为所有声明条款的 导入 子项的一部分，或作为个别 声明声明 的一部分：

导入类型的元数据声明示例

```
import org.drools.examples.Person

declare Person
    @author( Bob )
    @dateOfCreation( 01-Feb-2009 )
end
```

声明类型的元数据声明示例

```
declare org.drools.examples.Person
    @author( Bob )
    @dateOfCreation( 01-Feb-2009 )
end
```

16.5.5. DRL 中事实类型和属性声明的元数据标签

虽然您可以在 DRL 声明中定义自定义元数据属性，但决策引擎也支持以下预定义元数据标签用于事实类型或事实类型属性声明。

注意

本节中的示例参考 VoiceCall 类假设示例应用程序域模型包括以下类详情：

示例电信域模型中的 VoiceCall 事实类

```
public class VoiceCall {
  private String  originNumber;
  private String  destinationNumber;
  private Date    callDateTime;
  private long    callDuration;  // in milliseconds

  // Constructors, getters, and setters
}
```

@role

此标签决定，给定事实类型是作为常规事实处理，还是在复杂事件处理期间在决策引擎中的事件进行处理。

default 参数： fact

支持的参数： 事实、事件

```
@role( fact | event )
```

示例： Declare VoiceCall 作为事件类型

```
declare VoiceCall
  @role( event )
end
```

@timestamp

此标签会自动分配给决策引擎中的每个事件。默认情况下，会话时钟提供的时间，并在事件插入决策引擎的工作内存中时分配给事件。您可以指定自定义时间戳属性，而不是会话时钟添加的默认时间戳。

default 参数：决策引擎会话时钟添加的时间

支持的参数： Session clock time 或 custom time stamp 属性

```
@timestamp( <attributeName> )
```

示例： Declare VoiceCall timestamp 属性

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
end
```

@duration

此标签决定决策引擎中事件的持续时间。事件可以是基于间隔的事件或时间点事件。基于间隔的事件的时间，并在决策引擎的工作内存中保留一段时间，直到它们的持续时间可用为止。点内事件没有持续时间，基本上是基于 interval 的事件，且持续为零。默认情况下，决策引擎中的每个事件都会有零持续时间。您可以指定自定义持续时间属性，而不是默认值。

默认参数：Null（零）

支持的参数： 自定义持续时间属性

```
@duration( <attributeName> )
```

示例： Declare VoiceCall duration 属性

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
```

```
  @duration( callDuration )
end
```

@expires

此标签决定事件在决策引擎工作内存过期前的时间持续时间。默认情况下，事件过期，事件不再匹配并激活任何当前规则。您可以定义事件应过期的时间。此标签定义还覆盖从 KIE 基础中的时序限制和滑动窗口计算的隐式到期偏移。只有在决策引擎以流模式运行时，该标签才可用。

默认参数：Null（事件后的事件过期，不再匹配并激活规则）

支持的参数：自定义 timeOffset 属性，格式为 [#d][#h][#m][#s][[ms]]

```
@expires( <timeOffset> )
```

示例：对 VoiceCall 事件进行禁止过期偏移

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
  @duration( callDuration )
  @expires( 1h35m )
end
```

@typesafe

此选项卡确定给定事实类型是使用 还是没有类型安全性的编译。默认情况下，所有类型声明都是使用启用类型 security 进行的编译。您可以覆盖此行为来类型非安全评估，其中所有限制都是以 MVEL 约束并执行动态生成的。这在处理没有通用或混合类型集合的集合时很有用。

默认参数： true

支持的参数： true、false

```
@typesafe( <boolean> )
```

–

示例： Declare VoiceCall 用于 type-unsafe 评估

```
declare VoiceCall
  @role( fact )
  @typesafe( false )
end
```

@serialVersionUID

此标记在事实声明中为 serializable 类定义标识 serialVersionUID 值。如果 serializable 类没有明确声明 serialVersionUID，则序列化运行时间会根据类的不同方面计算该类的默认 serialVersionUID 值，如 Java Object Serialization 规格 中所述。但是，为了实现最佳反序列化结果并更好地与序列化 KIE 会话的兼容性，请在相关类或您的 DRL 声明中根据需要设置 serialVersionUID。

默认参数： Null

支持的参数： 自定义 serialVersionUID 整数

```
@serialVersionUID( <integer> )
```

示例：对 VoiceCall 类的 Declare serialVersionUID

```
declare VoiceCall
  @serialVersionUID( 42 )
end
```

@key

此标签启用事实类型属性，用作事实类型的键标识符。然后，生成的类可以实施 equals（） 和 hashCode（） 方法，以确定类型的两个实例是否等于。决策引擎还可使用所有关键属性作为参数生成构造器。

默认参数： None

支持的参数： 无

```
<attributeDefinition> @key
```

示例： Declare Person类型属性作为键

```
declare Person
    firstName : String @key
    lastName : String @key
    age : int
end
```

在本例中，决策引擎检查 firstName 和 lastName 属性，以确定 Person 的两个实例是否相互相等，但它没有检查 age 属性。决策引擎还会隐式生成三个构造器：一个没有参数，一个含有 @key 字段，另一个含有所有字段：

键声明中的构造器示例

```
Person() // Empty constructor

Person( String firstName, String lastName )

Person( String firstName, String lastName, int age )
```

然后，您可以基于密钥构造器创建类型为 的实例，如下例所示：

使用密钥构造器的实例示例

```
Person person = new Person( "John", "Doe" );
```

@position

此标记决定了位置参数中声明事实类型属性或字段的位置，覆盖属性的默认声明顺序。您可以使用此标签来修改模式中的位置限制，同时保持类型声明和位置参数的一致格式。此标签仅适用于 classpath 上类的字段。如果一个类中的一些字段使用此标签，则没有此标签的属性将按照声明的顺序定位。支持类继承，但不支持方法接口。

默认参数： None

支持的参数： 任何整数

```
<attributeDefinition> @position ( <integer> )
```

示例：拒绝事实类型和覆盖声明的顺序

```
declare Person
    firstName : String @position( 1 )
    lastName : String @position( 0 )
    age : int @position( 2 )
    occupation: String
end
```

在本例中，属性按以下顺序在位置参数中排列优先级：

1. **lastName**
2. **firstName**
3. **age**

4.
occupation

在位置参数中，您不需要指定字段名称，因为位置映射到已知命名字段。例如，参数 **Person(lastName == "Doe")** 与 **Person("Doe";)** 相同，其中 **lastName** 字段在 **DRL** 声明中具有最高位置注解。分号 **;** 表示在它是一个位置参数之前的所有内容。您可以使用分号将它们分开，在模式中混合位置和命名参数。位置参数中的任何变量都未绑定到映射到该位置的字段。

以下示例模式演示了构建位置和命名参数的不同方法。该模式有两个限制和绑定，分号区分指定参数部分的 **positional** 部分。在位置参数中支持使用字面意义的变量和字面量，但不支持单独变量。

带有位置和指定参数的模式示例

```
Person( "Doe", "John", $a; )

Person( "Doe", "John"; $a : age )

Person( "Doe"; firstName == "John", $a : age )

Person( lastName == "Doe"; firstName == "John", $a : age )
```

位置参数可以归类为 *输入参数* 或 *输出参数*。输入参数包含之前声明的绑定并使用未验证受到该绑定的限制。当绑定尚不存在时，输出参数会生成声明并将其绑定到 **positional** 参数代表的字段。

在扩展类型声明中，在定义 **@position** 注释时要谨慎，因为属性位置在子类型中继承。这种继承可能会导致混合属性顺序在在某些情况下可能会引起混淆。两个字段可以具有相同的 **@position** 值，并且不需要声明连续的值。如果重复位置，则使用继承的方式解决冲突，即父类型中的位置值具有优先权，然后使用第一个到最后一个声明中的声明顺序。

例如，以下扩展类型声明导致混合位置优先级：

带有混合位置注解的扩展事实类型示例

```
declare Person
    firstName : String @position( 1 )
    lastName : String @position( 0 )
```

```
    age : int @position( 2 )
    occupation: String
end

declare Student extends Person
    degree : String @position( 1 )
    school : String @position( 0 )
    graduationDate : Date
end
```

在本例中，属性按以下顺序在位置参数中排列优先级：

1. **lastname** （在父类型中的位置 0）
2. **院校** （子类型中的位置 0）
3. **FirstName** （在父类型中的位置 1）
4. **程度** （子类型中的位置 1）
5. **age** （在父类型中的位置 2）
6. **occupation** （没有位置注解的前字段）
7. **graduationDate** （无位置注解的第二个字段）

16.5.6. 为事实类型更改设置和监听程序

默认情况下，决策引擎不会在每次触发规则时重新评估所有事实类型的事实模式，而是仅响应给定模式内受限制或绑定的属性。例如，如果规则调用 **modify（）** 作为规则操作的一部分，但该操作不会在 KIE 基础中生成新数据，则决策引擎不会自动重新评估所有事实模式，因为没有修改数据。这个属性 **reactivity** 行为可防止 KIE 库中不需要的递归，并导致更有效的规则评估。这个行为还意味着您不需要总是使用 **no-loop rule** 属性来避免无限重复。

您可以使用以下 KnowledgeBuilderConfiguration 选项修改或禁用此属性重新活动行为，然后根据需要使用 Java 类或 DRL 文件中的属性更改设置来微调属性重新活动：

- ALWAYS:（默认）所有类型都是属性 reactive，但您可以使用 @classReactive 属性-change 设置禁用特定类型的属性重新活动。
- ALLOWED ：无类型是属性 reactive，但您可以使用 @propertyReactive 属性-change 设置为特定类型启用属性重新活动。
- DISABLED: No type 是属性 reactive。所有属性更改监听程序都会被忽略。

在 KnowledgeBuilderConfiguration 中属性重新活动设置示例

```
KnowledgeBuilderConfiguration config =
KnowledgeBuilderFactory.newKnowledgeBuilderConfiguration();
config.setOption(PropertySpecificOption.ALLOWED);
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder(config);
```

或者，您可以在 Red Hat Process Automation Manager 发行版的 standalone.xml 文件中更新 drools.propertySpecific 系统属性：

在系统属性中设置的属性重新活动示例

```
<system-properties>
  ...
  <property name="drools.propertySpecific" value="ALLOWED"/>
  ...
</system-properties>
```

决策引擎支持以下属性更改设置和监听程序，用于事实类或声明的 DRL 事实类型：

@classReactive

如果在决策引擎中将属性重新活动设置为 **ALWAYS** （所有类型都是 **reactive**），则此标签会禁用特定 **Java** 类的默认属性重新活动行为或声明的 **DRL** 事实类型。如果您希望决策引擎在每次触发规则时重新评估指定事实类型的所有事实模式，则可使用此标签，而不是仅响应给定模式内受约束或绑定的属性。

示例： 禁用 DRL 类型声明中的默认属性重新活动

```
declare Person
  @classReactive
    firstName : String
    lastName : String
end
```

示例：禁用 Java 类中的默认属性重新活动

```
@classReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@propertyReactive

如果在决策引擎中将属性重新活动设置为 **ALLOWED** （没有属性重新活跃），则此标签为特定 **Java** 类启用属性重新活动，或者为声明的 **DRL** 事实类型启用属性重新活动。如果您希望决策引擎仅响应指定事实类型内受限制或绑定的属性，则可使用此标签，而不是在每次触发规则时重新评估所有事实模式。

示例：在 DRL 类型声明中启用属性重新活动（在全局禁用重新活动时）

```
declare Person
  @propertyReactive
    firstName : String
```

```
    lastName : String
end
```

示例：在 Java 类中启用属性重新活动（当全局禁用重新活动时）

```
@propertyReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@watch

此标签为您在 DRL 规则中以实际模式指定的附加属性启用属性重新活动。只有在决策引擎中将属性重新活动设置为 ALWAYS 时，或将属性重新活动设置为 ALLOWED 且相关事实类型使用 @propertyReactive 标签时，才支持该标签。您可以在 DRL 规则中使用此标签在事实属性重新活动逻辑中添加或排除特定属性。

默认参数： None

支持的参数：Property name, * （全部）、! （不是）、!* （无属性）

```
<factPattern> @watch ( <property> )
```

示例：在事实模式下启用或禁用属性重新活动

```
// Listens for changes in both `firstName` (inferred) and `lastName`:
Person(firstName == $expectedFirstName) @watch( lastName )

// Listens for changes in all properties of the `Person` fact:
Person(firstName == $expectedFirstName) @watch( * )

// Listens for changes in `lastName` and explicitly excludes changes in `firstName`:
Person(firstName == $expectedFirstName) @watch( lastName, !firstName )

// Listens for changes in all properties of the `Person` fact except `age`:
Person(firstName == $expectedFirstName) @watch( *, !age )
```

```
// Excludes changes in all properties of the `Person` fact (equivalent to using `@classReactivity`
tag):
Person(firstName == $expectedFirstName) @watch( !* )
```

如果您在使用 @classReactive 标签的事实类型中使用 @watch 标签（禁用属性重新活动），或在决策引擎中将属性重新活动设置为 ALLOWED，则决策引擎将 @watch 标签用于属性。如果您在监听器注解中重复属性（如 @watch(firstName, ! firstName) ）时，也会发生编译错误。

@propertyChangeSupport

对于实现对 JavaBeans 规格中定义的属性更改的事实，此标签使决策引擎能够监控事实属性中的更改。

示例： Declare 属性更改在 JavaBeans 对象中支持

```
declare Person
  @propertyChangeSupport
end
```

16.5.7. 访问应用程序代码中声明的 DRL 类型

DRL 中声明的类型通常在 DRL 文件中使用，而 Java 模型通常在规则和应用程序间共享模型时使用。因为在 KIE 基本编译时生成声明的类型，所以应用程序在应用程序运行时无法访问它们。在某些情况下，应用程序需要从声明的类型直接访问和处理事实，特别是当应用程序换行决策引擎时，为规则管理提供了更高级别的领域、特定于域的用户界面。

要直接从应用程序代码中处理声明的类型，您可以在 Red Hat Process Automation Manager 中使用 org.drools.definition.type.FactType API。通过此 API，您可以在声明的事实类型中实例化、读取和写入字段。

以下示例代码直接从应用程序修改 Person 事实类型：

通过 FactType API 处理声明的事实类型的应用程序代码示例

```
import java.util.Date;

import org.kie.api.definition.type.FactType;
import org.kie.api.KieBase;
import org.kie.api.runtime.KieSession;

...

// Get a reference to a KIE base with the declared type:
KieBase kbase = ...

// Get the declared fact type:
FactType personType = kbase.getFactType("org.drools.examples", "Person");

// Create instances:
Object bob = personType.newInstance();

// Set attribute values:
personType.set(bob, "name", "Bob" );
personType.set(bob, "dateOfBirth", new Date());
personType.set(bob, "address", new Address("King's Road","London","404"));

// Insert the fact into a KIE session:
KieSession ksession = ...
ksession.insert(bob);
ksession.fireAllRules();

// Read attributes:
String name = (String) personType.get(bob, "name");
Date date = (Date) personType.get(bob, "dateOfBirth");
```

API 还包括其他有用的方法，例如一次性设置所有属性、从 map 集合中读取值，或一次性读取所有属性到 映射 集合中。

虽然 API 行为与 Java 反射相似，但 API 不使用反射，它依赖于通过生成的字节码实施的更高性能访问器。

16.6. DRL 中的全局变量

DRL 文件中的全局变量通常会为规则提供数据或服务，如规则后果中使用的应用程序服务，并返回规则中的数据，如规则中添加的日志或值等。您可以通过 KIE 会话配置或 REST 操作在决策引擎的工作内

存中设置全局值，声明 DRL 文件中的规则以外的全局变量，然后在规则的操作（然后）部分使用。对于多个全局变量，请在 DRL 文件中使用单独的行。

以下示例演示了决策引擎的全局变量列表配置以及 DRL 文件中的对应全局变量定义：

决策引擎的全局列表配置示例

```
List<String> list = new ArrayList<>();
KieSession kieSession = kiebase.newKieSession();
kieSession.setGlobal( "myGlobalList", list );
```

包含规则的全局变量定义示例

```
global java.util.List myGlobalList;

rule "Using a global"
  when
    // Empty
  then
    myGlobalList.add( "My global list" );
end
```

警告

不要使用全局变量在规则中建立条件，除非全局变量具有恒定的不可变值。全局变量不会插入到决策引擎的工作内存中，因此决策引擎无法跟踪变量的值更改。

不要使用全局变量在规则间共享数据。规则始终原因并响应正常工作的内存状态。因此，如果要将数据从规则传递给规则，请向决策引擎的工作内存中断言数据。

全局变量的用例可能是电子邮件服务的实例。在调用决策引擎的集成代码中，您可以获取您的 emailService 对象，然后在决策引擎的工作内存中设置它。在 DRL 文件中，您声明有一个全局类型 emailService，并为它指定名称 "email"，然后在规则结果中，您可以使用电子邮件等操作，如 email.sendSMS(number)。

如果您在多个软件包中声明具有相同标识符的全局变量，则必须使用相同类型设置所有软件包，以便它们都引用同一全局值。

16.7. DRL 中的规则属性

规则属性是您可以添加到业务规则中修改规则行为的额外规格。在 DRL 文件中，通常以以下格式在规则条件和操作之上定义规则属性：

```
rule "rule_name"
    // Attribute
    // Attribute
    when
        // Conditions
    then
        // Actions
end
```

下表列出了您可以分配给规则的属性的名称和支持值：

表 16.1. 规则属性

属性	值
salience	定义规则优先级的整数。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。 示例：**salience 10**
enabled	布尔值。选择了 选项后，会启用该规则。如果没有选择 选项，该规则将被禁用。 示例：**enabled true**
date-effective	包含日期和时间定义的字符串。只有在当前日期和时间后面是 **date-effective** 属性后才能激活该规则。 示例：**日期为 "4-Sep-2018"**

属性	值
date-expires	包含日期和时间定义的字符串。如果当前的日期和时间位于 **date-expires** 属性后，则无法激活该规则。 示例： **date-expires "4-Oct-2018"**
no-loop	布尔值。选择 选项时，如果规则触发之前满足条件，则无法重新激活该规则（循环）。如果没有选择条件，可以在这些情形中循环该规则。 示例： **no-loop true**
日程表（日程）	为您指定要为其分配该规则的日程表组的字符串。日程表组允许您对规则组进行更多执行控制。只有已获取焦点的管理者组中的规则才能够被激活。 示例： **日程-组 "GroupName"**
activation-group	您要为其分配该规则的激活（或 XOR）组的字符串。在激活组中，只能激活一条规则。第一条规则取消激活组中所有规则的待处理激活。 示例： **activation-group "GroupName"**
duration	如果规则条件仍满足，则用于定义在激活规则的时间持续时间（以毫秒为单位）的长整数值。 示例： **duration 10000**
timer	用于标识 **int** (interval)或 **cron** 计时器定义的字符串，用于调度规则。 示例： **timer(cron:* 0/15 * * ?)** （每 15 分钟）
日历	用于调度规则的 Quartz 日历定义。 示例： **calendars "* * 0-7,18-23 ?* *"** （排除非工作时间）
auto-focus	布尔值，仅适用于 schedule groups 中的规则。选择 选项时，下一次激活规则时，会自动把焦点分配给分配给该规则的日程表组。 示例： **auto-focus true**
lock-on-active	布尔值，仅适用于规则流组或日程组中的规则。选择了 选项时，规则的 ruleflow 组下次变为活跃时间，或者规则的日程表组接收焦点，无法再次激活该规则，直到 ruleflow 组不再活跃，否则将失去焦点。这是 **no-loop** 属性的一个更强大的版本，因为无论更新的来源，都丢弃匹配规则的激活（而不只是规则本身）。此属性非常适合计算规则，其中有多个规则修改事实，而您不想再次匹配和触发任何规则。 示例： **lock-on-active true**
ruleflow-group	标识规则流组的字符串。在规则流组中，只有在相关规则流激活组时，规则才能触发。 示例： **ruleflow-group "GroupName"**

属性	值
dialect	标识 **JAVA** 或 **MVEL** 的字符串，用作规则中代码表达式的语言。默认情况下，该规则使用在软件包级别上指定的断言。这里指定的任意分区会覆盖规则的软件包选择设置。 示例：**第一个"JAVA"**  **注意** 当您在没有可执行模型的情况下使用 Red Hat Process Automation Manager 时，**去选"JAVA"** 规则会导致只支持 Java 5 语法。有关可执行模型的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

16.7.1. DRL 中的计时器和日历规则属性

计时器和日历是 DRL 规则属性，可让您将调度和时间限制应用到 DRL 规则。根据用例，这些属性需要额外的配置。

DRL 规则中的 timer 属性是识别 int (interval)或 cron 计时器定义的字符串，用于调度规则并支持以下格式：

计时器属性格式

```
timer ( int: <initial delay> <repeat interval> )

timer ( cron: <cron expression> )
```

间隔计时器属性示例

```
// Run after a 30-second delay
timer ( int: 30s )

// Run every 5 minutes after a 30-second delay each time
timer ( int: 30s 5m )
```

cron 计时器属性示例

```
// Run every 15 minutes
timer ( cron:* 0/15 * * * ? )
```

间隔定时计时器遵循 **java.util.Timer** 对象的语义，其初始延迟和可选的重复间隔。**cron** 计时器遵循标准 **Unix cron** 表达式。

以下示例 **DRL** 规则每 **15** 分钟使用 **cron** 计时器发送 **SMS** 文本消息：

带有 cron 计时器的 DRL 规则示例

```
rule "Send SMS message every 15 minutes"
  timer ( cron:* 0/15 * * * ? )
  when
    $a : Alarm( on == true )
  then
    channels[ "sms" ].insert( new Sms( $a.mobileNumber, "The alarm is still on." );
end
```

通常，一个由计时器控制的规则在触发规则并且根据计时器设置重复执行规则时，会变为活跃的规则。当规则条件不再与传入的事实匹配时，执行将停止。但是，决策引擎通过定时器处理规则的方式取决于决策引擎是 *主动模式* 还是采用 *被动模式*。

默认情况下，决策引擎以 *被动模式* 运行，并根据定义的计时器设置（用户或应用程序明确调用 **fireAllRules（）** ）来评估规则。相反，如果用户或应用程序调用 **fireUntilHalt（）**，则决策引擎以 *主动模式* 启动，并持续评估规则，直到用户或应用明确调用 **halt（）**。

当决策引擎处于主动模式时，即使在从调用返回至 **fireUntilHalt（）** 后，也会执行规则后果，而且决策引擎会保持对工作内存做出的任何更改。*例如，删除触发计时器规则执行这一事实，会导致重复执行终*

止，并插入事实，以便某些规则匹配会导致规则执行。但是，决策引擎不会持续激活，但只有在执行规则后才活跃。因此，在下一次执行计时器控制的规则前，决策引擎不会响应异步事实插入。禁止 KIE 会话终止所有计时器活动。

当决策引擎采用被动模式时，只有在再次调用 fireAllRules（） 时，才会评估定时规则规则。但是，您可以通过使用 TimedRuleExecutionOption 选项配置 KIE 会话，以被动模式更改默认的 timer-execution 行为，如下例所示：

KIE 会话配置，以被动模式自动执行规则

```
KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();
ksconf.setOption( TimedRuleExecutionOption.YES );
KSession ksession = kbase.newKieSession(ksconf, null);
```

您还可以在 TimedRuleExecutionOption 选项上设置 FILTERED 规格，该规格允许您定义回调来过滤这些规则，如下例所示：

KIE 会话配置，过滤自动执行哪些时间规则

```
KieSessionConfiguration ksconf = KieServices.Factory.get().newKieSessionConfiguration();
conf.setOption( new TimedRuleExecutionOption.FILTERED(new TimedRuleExecutionFilter() {
    public boolean accept(Rule[] rules) {
        return rules[0].getName().equals("MyRule");
    }
}) );
```

对于间隔计时器，您还可以使用带有 expr 而不是 int 的表达式计时器，将延迟和间隔定义为表达式，而不是固定值。

以下示例 DRL 文件声明了一个事实类型，其延迟和周期后带有表达式计时器，并在后续规则中使用的周期：

带有表达式计时器的规则示例

```
declare Bean
  delay   : String = "30s"
  period  : long = 60000
end

rule "Expression timer"
  timer ( expr: $d, $p )
  when
    Bean( $d : delay, $p : period )
  then
    // Actions
end
```

本例中的表达式（如 $d 和 $p ）可以使用规则的模式匹配中定义的任何变量。变量可以是可以解析为持续时间的任何 String 值，也可以是内部转换为 长 值（以毫秒为单位）转换的任何数值。

间隔和表达式计时器可使用以下可选参数：

- 开始和结束 ： 代表 日期或 一个长值的字符串。该值也可以是以 新 日期 （(Number)n）格式转换为 Java 日期的数字。
- repeat-limit ：定义计时器允许的最大重复数的整数。如果同时设置了 end 和 repeat-limit 参数，则计时器在到达两个的第一个时停止。

可选的 start、end 和 repeat-limit 参数的 timer 属性示例

```
timer (int: 30s 1h; start=3-JAN-2020, end=4-JAN-2020, repeat-limit=50)
```

在这个示例中，在 2020 年 1 月 3 日开始（从 2020 年 1 月 3 日开始，或当周期重复 50 次时）为每小时调度规则。

如果系统暂停（例如，会话被序列化后，然后是反序列化），则规则被调度为从缺失的激活中恢复，无论暂停期间丢失了多少激活，然后被调度该规则以与计时器设置继续同步。

DRL 规则中的 calendar 属性是用于调度规则的 Quartz 日历定义，它支持以下格式：

calendar 属性格式

```
calendars "<definition or registered name>"
```

日历属性示例

```
// Exclude non-business hours
calendars "* * 0-7,18-23 ? * *"

// Weekdays only, as registered in the KIE session
calendars "weekday"
```

您可以基于 Quartz 日历 API 调节 Quartz 日历，然后在 KIE 会话中注册日历，如下例所示：

改编 Quartz Calendar

```
Calendar weekDayCal = QuartzHelper.quartzCalendarAdapter(org.quartz.Calendar quartzCal)
```

在 KIE 会话中注册日历

```
ksession.getCalendars().set( "weekday", weekDayCal );
```

您可以将日历用于标准规则，以及使用计时器的规则。calendar 属性可以包含一个或多个以 字符串文字形式编写的日历名称。

以下示例规则使用日历和计时器来调度规则：

带有日历和计时器的规则示例

```
rule "Weekdays are high priority"
  calendars "weekday"
  timer ( int:0 1h )
  when
    Alarm()
  then
    send( "priority high - we have an alarm" );
end

rule "Weekends are low priority"
  calendars "weekend"
  timer ( int:0 4h )
  when
    Alarm()
  then
    send( "priority low - we have an alarm" );
end
```

16.8. DRL 中的规则条件(WHEN)

当 DRL 规则的一部分（也称为 Left Hand Side(LHS)）包含要执行操作的条件。条件由一系列指定模式和 约束 组成，具有可选的绑定和支持的规则条件元素（关键字），具体取决于软件包中的可用数据对象。例如，如果银行需要 loan applicants 已有 21 余年，那么 "Underage" 规则的 when 条件将是适用的 （年龄 < 21 ）。

注意

如果 DRL 通常是特定时间点的确定性执行流的一部分，则 DRL 而不是使用。相反，如果 指示条件评估不受特定评估序列的限制，而是随时持续发生。每当满足条件时，都会执行相关的操作。

如果 when 部分为空，则条件被视为 true，在决策引擎中第一次执行 fireAllRules（） 调用。如果要使用规则设置决策引擎状态，这很有用。

以下示例规则使用空条件在每次执行规则时插入事实：

没有条件的规则示例

```
rule "Always insert applicant"
  when
    // Empty
  then   // Actions to be executed once
    insert( new Applicant() );
end

// The rule is internally rewritten in the following way:

rule "Always insert applicant"
  when
    eval( true )
  then
    insert( new Applicant() );
end
```

如果规则条件使用多个模式，且没有定义的关键字（如 和、或 ），则默认是 和 ：

没有关键字组合的规则示例

```
rule "Underage"
  when
    application : LoanApplication()
    Applicant( age < 21 )
  then
```

```
    // Actions
end

// The rule is internally rewritten in the following way:

rule "Underage"
  when
    application : LoanApplication()
    and Applicant( age < 21 )
  then
    // Actions
end
```

16.8.1. 模式和限制

DRL 规则状况中的 模式是 与决策引擎匹配的片段。模式可能会与插入到决策引擎工作内存的每个事实匹配。模式也可以包含 约束，以进一步定义要匹配的事实。

在最简单的形式中，没有限制，模式匹配给定类型的事实。在以下示例中，type 是 Person，因此模式将与决策引擎的工作内存中的所有 Person 对象匹配：

单一事实类型的模式示例

```
Person()
```

类型不需要是某些事实对象的实际类。模式可以参考超类甚至接口，可能会与许多不同类匹配的事实。例如，以下模式匹配决策引擎工作内存中的所有对象：

所有对象的模式示例

```
Object() // Matches all objects in the working memory
```

括起约束的特征的括号，如人员年龄上的以下约束：

带有约束的模式示例

```
Person( age == 50 )
```

约束(constraint)是一个返回 true 或 false 的表达式。DRL 中的模式限制基本上是带有一些增强功能的 Java 表达式，如属性访问，以及一些差别，如 equivalent （） 和 !equals（） 语义 == 和 != （而不是相同 语义）。

任何 JavaBeans 属性都可以直接从模式约束进行访问。bean 属性通过标准 JavaBeans getter 在内部公开，该参数不使用任何参数，并返回一些参数。例如，age 属性在 DRL 中以 年龄 形式编写，而不是 getter getAge（） ：

使用 JavaBeans 属性的 DRL 约束语法

```
Person( age == 50 )

// This is the same as the following getter format:

Person( getAge() == 50 )
```

Red Hat Process Automation Manager 使用标准的 JDK Introspector 类来实现这个映射，因此它遵循标准 JavaBeans 规格。要获得最佳决策引擎性能，请使用属性访问格式，如 age，而不是显式使用 getters，如 getAge（）。

警告

不要使用属性访问器以可能影响规则的方式更改对象状态，因为决策引擎会缓存调用之间的匹配结果，以获得更高的效率。

例如，不要使用以下方法使用属性 accessors：

```
public int getAge() {
    age++; // Do not do this.
    return age;
}
```

```
public int getAge() {
    Date now = DateUtil.now(); // Do not do this.
    return DateUtil.differenceInYears(now, birthday);
}
```

在第二个示例中插入一条事实，它会将当前日期嵌套到工作内存中，并根据需要更新 fireAllRules（） 之间的事实。

但是，如果无法找到属性的 getter，则编译器将使用属性名称作为回退方法名称，不带参数：

如果没有找到对象，则回退方法

```
Person( age == 50 )

// If `Person.getAge()` does not exist, the compiler uses the following syntax:

Person( age() == 50 )
```

您还可以在模式中嵌套访问属性，如下例所示。嵌套属性由决策引擎索引。

带有嵌套属性访问的模式示例

```
Person( address.houseNumber == 50 )

// This is the same as the following format:

Person( getAddress().getHouseNumber() == 50 )
```

警告

在有状态 KIE 会话中，请谨慎使用嵌套访问器，因为决策引擎的工作内存不知道任何嵌套值，且不会检测到它们何时更改。当其任何父引用都插入到工作内存中时，或者要修改嵌套值，请将所有 outer 事实标记为更新。在上例中，当 houseNumber 属性更改时，具有该地址的任何 Person 都必须标记为更新。

您可以使用返回布尔值的任何 Java 表达式作为模式括号内的约束。Java 表达式可以与其他表达式增强一起混合，如属性访问：

使用属性访问和 Java 表达式具有约束的模式示例

```
Person( age == 50 )
```

您可以使用括号（如任意逻辑或数学表达式中）更改评估优先级：

约束的评估顺序示例

```
Person( age > 100 && ( age % 10 == 0 ) )
```

您还可以在约束中重复使用 Java 方法，如下例所示：

使用可重复使用的 Java 方法的限制示例

```
Person( Math.round( weight / ( height * height ) ) < 25.0 )
```

警告

不要使用约束来改变对象的状态，这可能会影响规则，因为决策引擎缓存调用之间的匹配结果，以获得更高的效率。在规则条件中执行的任何方法都必须是只读方法。另外，事实的状态不应在规则调用之间更改，除非这些事实在每次更改的工作内存中标记为更新。

例如，不要使用以下方法使用模式约束：

```
Person( incrementAndGetAge() == 10 ) // Do not do this.
```

```
Person( System.currentTimeMillis() % 1000 == 0 ) // Do not do this.
```

标准 Java 运算符优先级适用于约束 DRL 中的 operator，DRL 运算符则遵循标准的 Java 语义，但 == 和 != 运算符除外。

== 运算符使用 null-safe equals（） 语义，而不是常见的 相同的 语义。例如，pattern Person(firstName == "John") 与 java.util.Objects.equals（person.getFirstName（）, "John"） 类似，因为 "John" 不是 null，模式也类似于 "John".equals（person.getFirstName（））。

!= 运算符使用 null-safe !equals（） 语义，而不是通常相同的 语义。例如，Person(firstName != "John") 类似于 !java.util.Objects.equals（person.getFirstName（）, "John"）。

如果项和约束的值不同类型，则决策引擎使用类型协调来解决冲突并减少编译错误。例如，如果 "ten" 在数字 evaluator 中作为字符串提供，则会发生编译错误，而 "10" 则与数字 10 合并。在 coercion 中，字段类型始终优先于值类型：

带有 coerced 值的约束示例

```
Person( age == "10" ) // "10" is coerced to 10
```

对于约束组，您可以使用 分隔符 使用隐式 和连接 语义：

具有多个限制的模式示例

```
// Person is at least 50 years old and weighs at least 80 kilograms:
Person( age > 50, weight > 80 )

// Person is at least 50 years old, weighs at least 80 kilograms, and is taller than 2 meters:
Person( age > 50, weight > 80, height > 2 )
```

注意

虽然 && 和， 但 operator 具有相同的语义，但可使用不同的优先级解决它们。&& operator 前面是 || 运算符，并且 && 和 || 运算符都位于 , operator 前。使用顶级约束中的逗号操作人员获得最佳决策引擎性能和人类可读性。

您不能将逗号运算符嵌入到复合约束表达式中，例如在括号中：

复合约束表达式中滥用的逗号示例

```
// Do not use the following format:
Person( ( age > 50, weight > 80 ) || height > 2 )

// Use the following format instead:
Person( ( age > 50 && weight > 80 ) || height > 2 )
```

16.8.2. 模式和限制中的绑定变量

您可以将变量绑定到模式和限制，以引用规则其他部分中的对象。绑定变量可帮助您以更加高效的方式在数据模型中注解事实，更加有效或更一致地定义规则。要在规则中更轻松地区分变量和字段，请将标准格式 $variable 用于变量，特别是在复杂的规则中。这个约定很有用，但在 DRL 中不需要。

例如，以下 DRL 规则将变量 $p 用于具有 Person 事实的模式：

带有绑定变量的模式

```
rule "simple rule"
  when
    $p : Person()
  then
    System.out.println( "Person " + $p );
end
```

同样，您也可以将变量绑定到模式约束中的属性，如下例所示：

```
// Two persons of the same age:
Person( $firstAge : age ) // Binding
Person( age == $firstAge ) // Constraint expression
```

注意

约束绑定仅考虑遵循它的第一个原子表达式。在以下示例中，模式仅将人的年龄绑定到变量 **$a** ：

```
Person( $a : age * 2 < 100 )
```

对于更清晰且效率的规则定义，请分隔约束绑定和约束表达式。虽然支持混合绑定和表达式，但这可能会复杂模式并影响评估效率。

```
// Do not use the following format:
Person( $a : age * 2 < 100 )

// Use the following format instead:
Person( age * 2 < 100, $a : age )
```

在上例中，如果要绑定到人员年龄的 **$a** 变量，则必须将其嵌套到括号中，如下例所示：

```
Person( $a : (age * 2) )
```

决策引擎不支持绑定到同一声明，但不支持在多个属性间对参数进行解释。虽然位置参数始终以未验证处理，但存在命名参数的符号 **:=**。

以下示例模式在两个 **Person** 事实中统一 **age** 属性：

带有未验证的模式示例

```
Person( $age := age )
Person( $age := age )
```

未验证为第一个发生次数声明绑定，限制在出现的顺序时与 **bound** 字段的值相同。

16.8.3. 嵌套限制和内联广播

在某些情况下，您可能需要访问嵌套对象的多个属性，如下例所示：

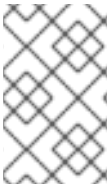
访问多个属性的模式示例

```
Person( name == "mark", address.city == "london", address.country == "uk" )
```

您可以将这些属性访问器分组为使用语法 .(<constraints>) 的嵌套对象，以获得更易读的规则，如下例所示：

带有分组限制的模式示例

```
Person( name == "mark", address.( city == "london", country == "uk") )
```



注意

period 前缀。 将嵌套的对象约束与方法调用区分。

当您以模式中使用嵌套对象时，您可以使用语法 < type>#<subtype > 来广播到子类型，并使父类型可用的 getters 提供给子类型。您可以使用对象名称或完全限定类名称，您可以转换到一个或多个子类型，如下例所示：

内联转换为子类型的模式示例

```
// Inline casting with subtype name:
Person( name == "mark", address#LongAddress.country == "uk" )

// Inline casting with fully qualified class name:
Person( name == "mark", address#org.domain.LongAddress.country == "uk" )
```

```
// Multiple inline casts:
Person( name == "mark", address#LongAddress.country#DetailedCountry.population > 10000000 )
```

这些示例模式将地址转换为 LongAddress，以及上一次示例中的 DetailedCountry，使父地址可供子类型使用。

您可以使用 instanceof operator 在随后使用那个字段的模式来推断指定类型的结果，如下例所示：

```
Person( name == "mark", address instanceof LongAddress, address.country == "uk" )
```

如果无法进行内联广播（例如，如果 instanceof 返回假），则评估将被视为 false。

16.8.4. 限制中的日期

默认情况下，决策引擎支持日期格式 dd-mmm-yyyyyyyyyyy。您可以通过为系统属性 drools.dateformat="dd-mmm-yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyymm".您还可以通过使用 drools.defaultlanguage 和 drools.defaultcountry 系统属性更改语言区域来自定义日期格式（例如，手册的区域设置为 drools.defaultlanguage=th 和 drools.defaultry=TH）。

带有日期字面限制的模式示例

```
Person( bornBefore < "27-Oct-2009" )
```

16.8.5. DRL 模式限制中支持的 operator

DRL 为模式限制中的 Operator 支持标准 Java 语义，但有一些例外，也有一些在 DRL 中唯一的额外运算符。以下列表总结了 DRL 约束中与标准 Java 语义或在 DRL 限制中唯一处理的情况。

.(), #

使用 .（） operator 将属性访问器分组为嵌套对象，并使用 # operator 广播到嵌套对象中的子

类型。广播到子类型，从父类型提供给子类型。您可以使用对象名称或完全限定的类名称，您可以转换到一个或多个子类型。

带有嵌套对象的模式示例

```
// Ungrouped property accessors:
Person( name == "mark", address.city == "london", address.country == "uk" )

// Grouped property accessors:
Person( name == "mark", address.( city == "london", country == "uk") )
```

注意

period 前缀。 将嵌套的对象约束与方法调用区分。

内联转换为子类型的模式示例

```
// Inline casting with subtype name:
Person( name == "mark", address#LongAddress.country == "uk" )

// Inline casting with fully qualified class name:
Person( name == "mark", address#org.domain.LongAddress.country == "uk" )

// Multiple inline casts:
Person( name == "mark", address#LongAddress.country#DetailedCountry.population > 10000000
)
```

!.

使用此运算符解引用 null-safe 的属性。!. 运算符左侧的值必须不为空（假定为 != null）才能获得正面的模式匹配的结果。

带有 null-safe dereferencing 的约束示例

```
Person( $streetName : address!.street )

// This is internally rewritten in the following way:

Person( address != null, $streetName : address.street )
```

[]

使用此运算符按键通过索引或映射值访问 List 值。

带有 List 和 Map 访问的限制示例

```
// The following format is the same as `childList(0).getAge() == 18`:
Person(childList[0].age == 18)

// The following format is the same as `credentialMap.get("jdoe").isValid()`:
Person(credentialMap["jdoe"].valid)
```

<, <=, >, >=

将这些运算符用于具有自然排序的属性。例如，对于 Date 字段，& lt; 运算符表示 在 之前，对于 String 字段，运算符表示 先按字母顺序排列。这些属性仅适用于可比较的属性。

使用 before operator 的约束示例

```
Person( birthDate < $otherBirthDate )

Person( firstName < $otherFirstName )
```

==, !=

将这些运算符使用 等于（） 和 !equals（） 方法。

null-safe equality 的约束示例

```
Person( firstName == "John" )

// This is similar to the following formats:

java.util.Objects.equals(person.getFirstName(), "John")
"John".equals(person.getFirstName())
```

null-safe 没有相等的约束示例

```
Person( firstName != "John" )

// This is similar to the following format:

!java.util.Objects.equals(person.getFirstName(), "John")
```

&&, ||

使用这些运算符创建一个缩写组合关系条件，该条件在字段上增加多个限制。您可以使用括号 () 对限制进行分组，以创建递归语法模式。

带有缩写关系的限制示例

```
// Simple abbreviated combined relation condition using a single `&&`:
Person(age > 30 && < 40)

// Complex abbreviated combined relation using groupings:
Person(age ((> 30 && < 40) || (> 20 && < 25)))

// Mixing abbreviated combined relation with constraint connectives:
Person(age > 30 && < 40 || location == "london")
```

匹配，不匹配

使用这些运算符表示字段匹配或与指定的 Java 正则表达式不匹配。通常，正则表达式是一个字符串文字，但解析为有效正则表达式的变量也受到支持。这些运算符仅应用到 String 属性。如果您使用 与 null 值匹配，则得到的评估始终为 false。如果您使用 与 null 值不匹配，则得到的评估始终为 true。与在 Java 中一样，编写为 字符串 文字的正则表达式必须使用双反斜杠 \\ 来转义。

匹配或不匹配正则表达式的约束示例

```
Person( country matches "(USA)?\\S*UK" )

Person( country not matches "(USA)?\\S*UK" )
```

包含，但不包含

使用这些运算符验证是否为 Array 或 a Collection 包含或不包含指定的值的字段。这些运算符适用于 Array 或 Collection 属性，但也可以使用这些运算符来取代 String.contains（） 和 !String.tains（） 约束检查。

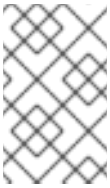
带有 的限制示例， 但不包含集合 (Collection)

```
// Collection with a specified field:
FamilyTree( countries contains "UK" )

FamilyTree( countries not contains "UK" )


// Collection with a variable:
FamilyTree( countries contains $var )

FamilyTree( countries not contains $var )
```

带有 的约束示例， 不包含 用于 String 字面上的示例

```
// Sting literal with a specified field:
Person( fullName contains "Jr" )
```

```
Person( fullName not contains "Jr" )

// String literal with a variable:
Person( fullName contains $var )

Person( fullName not contains $var )
```



注意

为实现**向后兼容，excludes 操作器支持了一个 不包含 的同**义词。

memberOf，不是 memberOf

使用这些运**算符**验证**某个字段是否是数**组**的成**员，还**是定**义为变**量的 Collection。Array 或 Collection 必**须**是一个**变**量。**

带有 memberOf 且不是 memberOf 的限制示例

```
FamilyTree( person memberOf $europeanDescendants )

FamilyTree( person not memberOf $europeanDescendants )
```

听起来

使用此运**算符**验证**某个**词语**是否几乎是相同的声音，使用英**语**探**测**器作**为给**定**值（与 匹配 Operator 相似）。此 Operator 使用 Soundex 算法。**

具有声音 一样**的**约**束示例**

```
// Match firstName "Jon" or "John":
Person( firstName soundslike "John" )
```

str

使用此运算符验证某个字段是否以 String 开头或以指定的值结束。您还可以使用此运算符验证 String 的长度。

使用预先限制示例

```
// Verify what the String starts with:
Message( routingValue str[startsWith] "R1" )

// Verify what the String ends with:
Message( routingValue str[endsWith] "R2" )

// Verify the length of the String:
Message( routingValue str[length] 17 )
```

在 中，而不是在 中

使用这些运算符在约束（复合值限制）中指定多个可能的值。复合值限制的功能只在 in 和 not Operator 中被支持。这些运算符的第二运算对象必须是以括号括起的逗号分隔的值列表。您可以将值作为变量、文字、返回值或合格标识符提供。这些运算符在内部使用 operators == 或 != 重写为多个限制的列表。

使用 中的 和 not in 的限制示例

```
Person( $color : favoriteColor )
Color( type in ( "red", "blue", $color ) )

Person( $color : favoriteColor )
Color( type notin ( "red", "blue", $color ) )
```

16.8.6. DRL 模式限制中的 Operator 优先级

DRL 支持适用于适用约束运算符的标准 Java 操作优先级，但有一些例外，也有一些在 DRL 中唯一的额外运算符。下表列出了适用的 DRL 运算符优先级，从高到低优先级：

表 16.2. DRL 模式限制中的 Operator 优先级

Operator 类型	Operator	备注
嵌套或 null-safe 属性访问	**.**, **.()**, **!.**	不是标准 Java 语义
列出 或 **映射** 访问	**[]**	不是标准 Java 语义
约束绑定	**:**	不是标准 Java 语义
Multiplicative	*, /**%**	
additive	**+**, **-**	
改变	**>>**, **>>>**, **<<**	
关系	**<**, **<=**, **>** , **&** gt;= , **instanceof**	
等于	**== !=**	使用等 （） 和 **!equals**（） 语义，它 **并不相同，而不是相同的** 语义
Non-short-circuiting **and**	**&**	
非短路 **或**	**^**	
Non-short-circuiting includes **OR**	|	
逻辑 **AND**	**&&**	
逻辑 **OR**	||	
Ternary	**? :**	
逗号分隔 **AND**	**,**	不是标准 Java 语义

16.8.7. DRL（关键字）中支持的规则条件元素

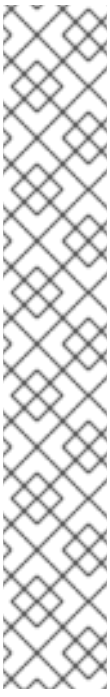
DRL 支持以下规则条件元素（关键字），供您用于 DRL 规则条件中定义的模式：

和

使用此选项将条件组件分组到逻辑中。支持在fix 和 前缀中。您可以使用括号 () 显式分组模式。默认情况下，所有列出的模式都与 和 结合使用，并且 未指定任何组合。

带有 和的模式示例

```
//Infix `and`:
Color( colorType : type ) and Person( favoriteColor == colorType )

//Infix `and` with grouping:
(Color( colorType : type ) and (Person( favoriteColor == colorType ) or Person( favoriteColor ==
colorType ))

// Prefix `and`:
(and Color( colorType : type ) Person( favoriteColor == colorType ))

// Default implicit `and`:
Color( colorType : type )
Person( favoriteColor == colorType )
```



注意

不要将前导声明绑定与 and 关键字一起使用（例如，您可以使用 或 ）。声明一次只能引用单一事实，如果您对 和 使用声明绑定，那么当 和 满足时，它将匹配事实并产生错误。

和使用示例

```
// Causes compile error:
$person : (Person( name == "Romeo" ) and Person( name == "Juliet"))
```

或

使用此选项将条件组件分组到逻辑不符中。支持在修复和 前缀或 支持中。您可以使用括号 () 显式分组模式。您也可以通过 或 使用模式绑定，但每个模式必须单独绑定。

带有 或的模式示例

```
//Infix `or`:
Color( colorType : type ) or Person( favoriteColor == colorType )

//Infix `or` with grouping:
(Color( colorType : type ) or (Person( favoriteColor == colorType ) and Person( favoriteColor ==
colorType ))

// Prefix `or`:
(or Color( colorType : type ) Person( favoriteColor == colorType ))
```

带有 或 and pattern binding 的特征示例

```
pensioner : (Person( sex == "f", age > 60 ) or Person( sex == "m", age > 65 ))

(or pensioner : Person( sex == "f", age > 60 )
    pensioner : Person( sex == "m", age > 65 ))
```

决策引擎不会直接解释 or 元素，而是使用逻辑转换来使用或作为多个子规则重写规则。此过程最终会生成具有单个 或 根节点的规则，以及每个 condition 元素的一个子规则。每个子规则都已激活并执行（如任何普通规则），在子规则之间没有特殊行为或交互。

因此，可以考虑 或 condition 元素生成两个或更多类似规则的快捷方式，当两个或更多人对不浏览的术语为 true 时，可以创建多个激活。

exists

使用此选项指定必须存在的事实和限制。这个选项仅在第一个匹配项中触发，而不是后续匹配项。如果您将这个元素与多个模式搭配使用，请将模式与括号 （） 括起。

存在带有 的模式示例

```
exists Person( firstName == "John")
```

```
exists (Person( firstName == "John", age == 42 ))

exists (Person( firstName == "John" ) and
        Person( lastName == "Doe" ))
```

not

使用此选项指定必须不存在的事实和限制。如果您将这个元素与多个模式搭配使用，请将模式与括号 () 括起。

没有的模式示例

```
not Person( firstName == "John")

not (Person( firstName == "John", age == 42 ))

not (Person( firstName == "John" ) and
     Person( lastName == "Doe" ))
```

全部

使用此选项验证匹配第一个模式的所有事实都与所有剩余的模式匹配。当 满足总 结构时，该规则将评估为 **true**。此元素是一个范围分隔符，因此可以使用任何之前绑定的变量，但不能在它外使用。

带有 **forall**的规则示例

```
rule "All full-time employees have red ID badges"
  when
    forall( $emp : Employee( type == "fulltime" )
                Employee( this == $emp, badgeColor = "red" ) )
  then
    // True, all full-time employees have red ID badges.
end
```

在本例中，该规则选择类型为 "fulltime" 的所有 Employee 对象。对于匹配此模式的每个事实，该规则将评估其遵循的模式（带有颜色），以及规则将评估为 true。

要在决策引擎的工作内存中给定类型的所有事实都必须与一组限制匹配，您可以借助单一模式进行简单操作。

带有 forall 和 single 模式的规则示例

```
rule "All full-time employees have red ID badges"
  when
    forall( Employee( badgeColor = "red" ) )
  then
    // True, all full-time employees have red ID badges.
end
```

您可以将 forall 结构与多个模式搭配使用，或者将其嵌套到其他状况元素中，如 not 元素结构内。

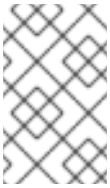
具有 forall 和多个模式的规则示例

```
rule "All employees have health and dental care programs"
  when
    forall( $emp : Employee()
            HealthCare( employee == $emp )
            DentalCare( employee == $emp )
          )
  then
    // True, all employees have health and dental care.
end
```

带有 forall 而不是的规则示例

```
rule "Not all employees have health and dental care"
```

```
  when
    not ( forall( $emp : Employee()
                  HealthCare( employee == $emp )
                  DentalCare( employee == $emp ) )
        )
  then
    // True, not all employees have health and dental care.
end
```



注意

forall(p1 p2 p3 ...) 等同于 not（p1, not（和 p3 ...））。

from

使用它来指定模式的数据源。这可让决策引擎与不在工作内存中的数据进行原因。数据源可以是绑定的变量或方法调用的结果上的子字段。用于定义对象源的表达式是符合常规 MVEL 语法的任何表达式。因此，通过 from 元素，您可以轻松地使用对象属性导航、执行方法调用和访问映射和集合元素。

带有 from 和 pattern binding 的规则示例

```
rule "Validate zipcode"
  when
    Person( $personAddress : address )
    Address( zipcode == "23920W" ) from $personAddress
  then
    // Zip code is okay.
end
```

带有 的规则示例以及图形表示法

```
rule "Validate zipcode"
  when
    $p : Person()
    $a : Address( zipcode == "23920W" ) from $p.address
```

```
  then
    // Zip code is okay.
end
```

*带有 **from** 的规则示例来迭代所有对象*

```
rule "Apply 10% discount to all items over US$ 100 in an order"
  when
    $order : Order()
    $item  : OrderItem( value > 100 ) from $order.items
  then
    // Apply discount to `$item`.
end
```

注意

对于大型对象集合，而不是添加带有决策引擎必须迭代的大型图的对象，直接将集合添加到 KIE 会话，然后在状况中加入集合，如下例所示：

```
when
  $order : Order()
  OrderItem( value > 100, order == $order )
```

来自 lock-on-active 规则属性的规则示例

```
rule "Assign people in North Carolina (NC) to sales region 1"
  ruleflow-group "test"
  lock-on-active true
  when
    $p : Person()
    $a : Address( state == "NC" ) from $p.address
  then
    modify ($p) {} // Assign the person to sales region 1.
end

rule "Apply a discount to people in the city of Raleigh"
  ruleflow-group "test"
  lock-on-active true
```

```
  when
    $p : Person()
    $a : Address( city == "Raleigh" ) from $p.address
  then
    modify ($p) {} // Apply discount to the person.
end
```

重要

使用 lock-on-active 规则属性中的 可能会导致规则没有被执行。您可以使用以下方法之一解决这个问题：

- 当您可以将所有事实插入到决策引擎的工作内存或者使用约束表达式中的嵌套对象引用时，请避免使用 from 元素。
- 将 modify（） 块中使用的变量设置为您的规则条件中的最后一个句子。
- 当您可以明确管理同一 ruleflow 组中的规则放入激活时，避免使用 lock-on-active 规则属性。

包含 from 子句的模式不能跟以括号开头的其他模式。这个限制的原因是 DRL 解析器 从 表达式中读取为 "l（String（）或 Number （））， 它无法将这个表达式与函数调用区分开。这一点最简单的方法是将 from 子句嵌套在括号中，如下例所示：

来自 的规则示例不正确且正确

```
// Do not use `from` in this way:
rule R
  when
    $l : List()
    String() from $l
    (String() or Number())
  then
    // Actions
end

// Use `from` in this way instead:
```

```
rule R
  when
    $l : List()
    (String() from $l)
    (String() or Number())
  then
    // Actions
end
```

entry-point

使用它来定义与模式的数据源对应的入口点或 事件源。此元素通常用于 from condition 元素。您可以为事件声明一个入口点，以便决策引擎仅从该入口点使用数据来评估规则。您可以在 DRL 规则或您的 Java 应用程序中引用一个入口点，以隐式声明该入口点。

*带有 **entry-point的规则示例***

```
rule "Authorize withdrawal"
  when
    WithdrawRequest( $ai : accountId, $am : amount ) from entry-point "ATM Stream"
    CheckingAccount( accountId == $ai, balance > $am )
  then
    // Authorize withdrawal.
end
```

*带有 **EntryPoint** 对象的 **Java** 应用代码示例并插入事实*

```
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.EntryPoint;

// Create your KIE base and KIE session as usual:
KieSession session = ...

// Create a reference to the entry point:
EntryPoint atmStream = session.getEntryPoint("ATM Stream");

// Start inserting your facts into the entry point:
atmStream.insert(aWithdrawRequest);
```

collect

使用它来定义规则作为条件的一部分使用的对象集合。该规则从指定的源或决策引擎工作内存获取集合。collect 元素的结果模式可以是实施 java.util.Collection 接口的任何 concrete 类，并提供默认的 no-arg 公共构造器。您可以使用 List、LinkedList 和 HashSet 等 Java 集合，或者您自己的类。如果变量在条件中的 collect 元素之前绑定，您可以使用变量来约束您的源和结果模式。但是，在 collect 元素内进行的任何绑定都无法供其外使用。

带有 collect的规则示例

```
import java.util.List

rule "Raise priority when system has more than three pending alarms"
  when
    $system : System()
    $alarms : List( size >= 3 )
              from collect( Alarm( system == $system, status == 'pending' ) )
  then
    // Raise priority because `$system` has three or more `$alarms` pending.
end
```

在本例中，该规则评估每个给定系统决策引擎工作内存中的所有待处理警报，并在列表中对它们进行分组。如果为给定系统找到三个或更多警报，则会执行该规则。

您还可以使用带有与元素嵌套的 收集 元素，如下例所示：

带有 收集和 嵌套的规则示例

```
import java.util.LinkedList;

rule "Send a message to all parents"
  when
    $town : Town( name == 'Paris' )
    $mothers : LinkedList()
               from collect( Person( children > 0 )
                             from $town.getPeople()
                           )
```

```
  then
    // Send a message to all parents.
end
```

累计型

使用此选项迭代对象集合，为每个元素执行自定义操作，并返回一个或多个结果对象（如果约束评估为 true）。此元素是收集条件元素的一种更加灵活、强大的形式。您可以使用 累积 条件或根据需要实施自定义功能中的预定义功能。您还可以在规则条件中使用 缩写 cc。

使用以下命令在规则中定义 累积 条件：

累积的首选格式

```
accumulate( <source pattern>; <functions> [;<constraints>] )
```

注意

虽然决策引擎支持替代格式用于向后兼容元素，但这种格式是规则和应用程序中优化性能的首选。

决策引擎支持以下预定义的 累积 功能：这些功能接受任何表达式作为输入。

- average
- 分钟
- max
- 数量

- ***sum***
- ***collectList***
- ***collectSet***

在以下示例中，min、max 和 average 是计算每个传感器所有读取的最小、最大值和平均温度值的累积函数：

带有 累积 值的规则示例

```
rule "Raise alarm"
  when
    $s : Sensor()
    accumulate( Reading( sensor == $s, $temp : temperature );
                $min : min( $temp ),
                $max : max( $temp ),
                $avg : average( $temp );
                $min < 20, $avg > 70 )
  then
    // Raise the alarm.
end
```

以下示例规则使用 平均 功能来 连续计算所有项目的平均利润：

以累计方式计算平均利润的规则示例

```
rule "Average profit"
  when
    $order : Order()
    accumulate( OrderItem( order == $order, $cost : cost, $price : price );
                $avgProfit : average( 1 - $cost / $price ) )
  then
    // Average profit for `$order` is `$avgProfit`.
end
```

要在 累积 条件中使用自定义特定域的功能，请创建一个实施 org.kie.api.runtime.rule.AccumulateFunction 接口的 Java 类。例如，以下 Java 类定义了 AverageData 功能的自定义实现：

具有 平均 功能的定制实施的 Java 类示例

```
// An implementation of an accumulator capable of calculating average values

public class AverageAccumulateFunction implements
org.kie.api.runtime.rule.AccumulateFunction<AverageAccumulateFunction.AverageData> {

    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {

    }

    public void writeExternal(ObjectOutput out) throws IOException {

    }

    public static class AverageData implements Externalizable {
        public int    count = 0;
        public double total = 0;

        public AverageData() {}

        public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException {
            count   = in.readInt();
            total   = in.readDouble();
        }

        public void writeExternal(ObjectOutput out) throws IOException {
            out.writeInt(count);
            out.writeDouble(total);
        }

    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#createContext()
     */
    public AverageData createContext() {
        return new AverageData();
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#init(java.io.Serializable)
```

```
     */
    public void init(AverageData context) {
        context.count = 0;
        context.total = 0;
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#accumulate(java.io.Serializable,
java.lang.Object)
     */
    public void accumulate(AverageData context,
                           Object value) {
        context.count++;
        context.total += ((Number) value).doubleValue();
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#reverse(java.io.Serializable,
java.lang.Object)
     */
    public void reverse(AverageData context, Object value) {
        context.count--;
        context.total -= ((Number) value).doubleValue();
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#getResult(java.io.Serializable)
     */
    public Object getResult(AverageData context) {
        return new Double( context.count == 0 ? 0 : context.total / context.count );
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#supportsReverse()
     */
    public boolean supportsReverse() {
        return true;
    }

    /* (non-Javadoc)
     * @see org.kie.api.runtime.rule.AccumulateFunction#getResultType()
     */
    public Class< ? > getResultType() {
        return Number.class;
    }

}
```

要在 DRL 规则中使用自定义功能，请使用导入 accumulate 语句导入函数：

导入自定义功能的格式

```
import accumulate <class_name> <function_name>
```

带有导入的 average 功能的规则示例

```
import accumulate AverageAccumulateFunction.AverageData average

rule "Average profit"
  when
    $order : Order()
    accumulate( OrderItem( order == $order, $cost : cost, $price : price );
                $avgProfit : average( 1 - $cost / $price ) )
  then
    // Average profit for `$order` is `$avgProfit`.
end
```

16.8.8. OOPath 语法，在 DRL 规则条件中对象图形

OOPath 是 XPath 的一个面向对象的语法扩展，专为浏览 DRL 规则条件限制中的对象图形。OOPath 使用 XPath 的紧凑标记来导航相关的元素，同时处理集合和过滤限制，对于对象图形来说，OOPath 特别有用。

当事实字段是一个集合时，您可以使用 from condition 元素（关键字）来绑定和原因，而不是该集合中的所有项目。如果您需要浏览规则条件约束中的对象图，从 condition 元素广泛使用会导致详细和重复的语法，如下例所示：

使用 from 浏览对象图的规则示例

```
rule "Find all grades for Big Data exam"
  when
    $student: Student( $plan: plan )
    $exam: Exam( course == "Big Data" ) from $plan.exams
    $grade: Grade() from $exam.grades
```

```
  then
    // Actions
end
```

在本例中，域模型包含带有 计划 研究的 Student 对象。计划 可以有零个或多个 考试 实例，一个 考试 可以有零个或多个预期实例。只有图形的 root 对象，本例中的 学员 需要位于决策引擎的工作内存中以便此规则设置才能正常工作。

作为使用语句中广泛使用的效率，您可以使用缩写的 OOPath 语法，如下例所示：

使用 OOPath 语法浏览对象图形的规则示例

```
rule "Find all grades for Big Data exam"
  when
    Student( $grade: /plan/exams[course == "Big Data"]/grades )
  then
    // Actions
end
```

正式，OOPath 表达式的核心抓取以扩展 Backus-Naur 表单(EBNF)表示法定义：

OOPath 表达式的 EBNF 表示法

```
OOPExpr = [ID ( ":" | ":=" )] ( "/" | "?/" ) OOPSegment { ( "/" | "?/" | "." ) OOPSegment } ;
OOPSegment = ID ["#" ID] ["[" ( Number | Constraints ) "]"]
```

在实践中，OOPath 表达式具有以下特性：

- 以正斜杠 / 开头，或以问号和正斜杠 ?/ 开头（如果它是非主动 OOPath 表达式（在本节稍

后进行评分）。

- 可以使用 period . operator 解引用对象的单一属性。

- 可以使用正斜杠 / 运算符解引用对象的多个属性。如果返回集合，则表达式会迭代集合中的值。

- 可以过滤掉不满足一个或多个限制的对象。约束被编写为方括号之间的 predicate 表达式，如下例所示：

  作为 predicate 表达式的限制

  ```
  Student( $grade: /plan/exams[ course == "Big Data" ]/grades )
  ```

- 可以分解一个遍历对象到通用集合中声明的类子类。后续限制也可以安全地访问该子类中声明的属性，如下例所示。不是此内联广播中指定的类实例的对象会自动过滤。

  使用 downcast 对象的限制

  ```
  Student( $grade: /plan/exams#AdvancedExam[ course == "Big Data", level > 3 ]/grades )
  ```

- 可以重新引用在当前迭代图前遍历图形的对象。例如，以下 OOPath 表达式仅匹配所传递考试的平均等级：

  使用 backreferenced 对象的限制

  ```
  Student( $grade: /plan/exams/grades[ result > ../averageResult ] )
  ```

- 也可以是另一个 OOPath 表达式，如下例所示：

递归约束表达式

```
Student( $exam: /plan/exams[ /grades[ result > 20 ] ] )
```

- 可以通过方括号 [] 间的索引访问对象，如下例所示。要遵循 Java 惯例，OOPath 索引基于 0，而 XPath 索引es 则基于 1。

通过索引对对象的访问限制

```
Student( $grade: /plan/exams[0]/grades )
```

OOPath 表达式可以是 reactive 或 non-reactive。在评估 OOPath 表达式的过程中，决策引擎不会响应涉及深度嵌套对象的更新。

要使这些对象被动更改，修改对象以扩展类 org.drools.core.phreak.ReactiveObject。在修改对象以扩展 ReactiveObject 类后，域对象调用继承的方法 通知修改，以在更新其中一个字段时通知决策引擎，如下例所示：

通知决策引擎（即考试已移至其他课程）的对象方法示例

```
public void setCourse(String course) {
    this.course = course;
    notifyModification(this);
    }
```

使用下列对应的 OOPath 表达式时，当考试移动到不同的课程时，重新执行该规则以及与该规则匹配的等级列表：

来自 "Big Data" 规则的 OOPath 表达式示例

```
Student( $grade: /plan/exams[ course == "Big Data" ]/grades )
```

您还可以使用 ?/ 分隔符而不是 / 分隔符仅在 OOPath 表达式的一个子端口中禁用 reinteractive，如下例所示：

部分非活跃的 OOPath 表达式示例

```
Student( $grade: /plan/exams[ course == "Big Data" ]?/grades )
```

在此例中，决策引擎对考试所做的更改做出反应，或者如果计划中添加了考试，但不将新评测添加到现有考试中。

如果 OOPath 部分不是活跃的，OOPath 表达式的所有剩余部分也将变为 non-reactive。例如，以下 OOPath 表达式完全非活跃：

完全非活跃的 OOPath 表达式示例

```
Student( $grade: ?/plan/exams[ course == "Big Data" ]/grades )
```

因此，您不能在同一个 OOPath 表达式中多次使用 ?/ 分隔符。例如，以下表达式会导致编译错误：

带有重复非活动标记的 OOPath 表达式示例

```
Student( $grade: /plan?/exams[ course == "Big Data" ]?/grades )
```

启用 OOPath 表达式重新活动的另一个替代方法是使用 Red Hat Process Automation Manager 中的 List 和 Set 接口的专用实现。这些实现是 ReactiveList 和 ReactiveSet 类。另外还可使用 ReactiveCollection 类。该实施还通过 Iterator 和 ListIterator 类提供对执行可变操作的被动支持。

以下示例类使用这些类来配置 OOPath 表达式重新活动：

配置 OOPath 表达式重新活动的 Java 类示例

```
public class School extends AbstractReactiveObject {
   private String name;
   private final List<Child> children = new ReactiveList<Child>(); 1

   public void setName(String name) {
      this.name = name;
      notifyModification(); 2
   }

   public void addChild(Child child) {
      children.add(child); 3
      // No need to call `notifyModification()` here
   }
}
```

1

对标准 Java List 实例使用 ReactiveList 实例来被动支持。

2

3

children 字段是一个 ReactiveList 实例，因此不需要 notifyModification（） 方法调用。通知会自动处理，与通过 children 字段执行的所有其他变异操作一样。

16.9. DRL(THEN)中的规则操作

规则的 then 部分（也称为规则的 右 Hand Side(RHS) ）包含满足规则条件部分时要执行的操作。操作由一个或多个 方法 组成，它们根据规则条件和软件包中的可用数据对象执行结果。例如，如果某个公司需要 Lan applicant 的时间超过 21 年，则规则状况 Applicant (age < 21)和 loan applicant 超过 21 年，则 "Underage" 规则的操作将被设置为 setApproved (false)，因为 applicant 结束了 loan。

规则操作的主要用途是在决策引擎工作内存中插入、删除或修改数据。有效的规则操作是 small、声明和可读。如果您需要在规则操作中使用需要或有条件的代码，请将该规则分成多个较小的子声明规则。

loan Application age 限制的规则示例

```
rule "Underage"
  when
    application : LoanApplication()
    Applicant( age < 21 )
  then
    application.setApproved( false );
    application.setExplanation( "Underage" );
end
```

16.9.1. DRL 中支持的规则操作方法

DRL 支持在 DRL 规则操作中使用的以下规则操作方法。您可以使用这些方法修改决策引擎的工作内存，而无需首先引用正常工作的内存实例。这些方法充当 Red Hat Process Automation Manager 发行版本中 RuleContext 类提供的方法快捷方式。

对于所有规则操作方法，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.5 Source Distribution ZIP 文件，并导航到 ~/rhpam-7.13.5-sources/src/kie-api-parent-$VERSION/kie-api/src/main/java/org/kie/api/runtime/rule/RuleContext.java.
https://access.redhat.com/jbossnetwork/restricted/listSoftware.html

set

使用此选项设置字段的值。

```
set<field> ( <value> )
```

*设置 **loan application** 批准的值的规则操作示例*

```
$application.setApproved ( false );
$application.setExplanation( "has been bankrupt" );
```

修改

使用此选项指定要修改的字段，并通知更改的决策引擎。此方法提供了一种结构化方法来事实更新。它将 更新操作与 设置者调用合并，以更改对象字段。

```
modify ( <fact-expression> ) {
   <expression>,
   <expression>,
   ...
}
```

修改 loan Application 数量和批准的规则操作示例

```
modify( LoanApplication ) {
        setAmount( 100 ),
        setApproved ( true )
}
```

update

使用此选项指定字段和要更新的完整相关事实，并通知更改的决策引擎。事实更改后，您必须在更改可能受更新的值影响的其他事实之前调用 更新。要避免添加的步骤，请改为使用 修改 方法。

```
update ( <object, <handle> )  // Informs the decision engine that an object has changed

update ( <object> )  // Causes `KieSession` to search for a fact handle of the object
```

更新 loan Application 数量和批准的规则操作示例

```
LoanApplication.setAmount( 100 );
update( LoanApplication );
```

注意

如果您提供属性更改监听程序，则在对象更改时不需要调用这个方法。有关属性更改监听程序的更多信息，请参阅 **Red Hat Process Automation Manager 中的 决策引擎**。

insert

使用它来将 新事实 插入到决策引擎的工作内存中，并根据需要定义结果字段和值。

```
insert( new <object> );
```

插入新 loan applicant 对象的规则操作示例

```
insert( new Applicant() );
```

insertLogical

使用它来在决策引擎中以逻辑方式插入新的事实。决策引擎负责对插入和检索事实的逻辑决策。在常规或声明插入后，必须明确调整事实。逻辑插入后，当插入事实的条件不再为 **true** 时，插入的事实会自动清空。

```
insertLogical( new <object> );
```

用于逻辑地插入新 loan applicant 对象的规则操作示例

```
insertLogical( new Applicant() );
```

delete

使用它来从决策引擎中删除对象。DRL 中还支持关键字 retract，并执行同样的操作，但 删除 通常优先于 DRL 代码，以便通过关键字 插入。

```
delete( <object> );
```

删除 loan applicant 对象的规则操作示例

```
delete( Applicant );
```

16.9.2. drools 变量的其他规则操作方法

除了标准规则操作方法外，决策引擎还支持方法和预定义的 drools 变量，您可以在规则操作中使用。

您可以使用 drools 变量从 Red Hat Process Automation Manager 发行版中的 org.kie.api.runtime.rule.RuleContext 类调用方法，这也是标准规则操作方法所基于的类。对于所有 drools 规则操作选项，请从红帽客户门户网站下载 Red Hat Process Automation Manager 7.13.5 Source Distribution ZIP 文件，并导航到 ~/rhpam-7.13.5-sources/src/kie-api-parent-$VERSION/kie-api/src/main/java/org/kie/api/runtime/rule/RuleContext.java。https://access.redhat.com/jbossnetwork/restricted/listSoftware.html

drools 变量包含有关触发规则的信息以及激活触发规则的事实集合的方法：

- drools.getRule （）.getName （） ：返回当前触发的规则的名称。
- drools.getMatch （） ：返回激活当前触发规则的 匹配。它包含记录和调试目的的信息，

例如 drools.getMatch （）.getObjects （） 返回对象列表，允许规则按照正确的元组顺序触发。

在 drools 变量中，您还可以获得对 KieRuntime 的引用，提供与正在运行的会话交互的有用方法，例如：

- drools.getKieRuntime （）.halt （）：如果之前称为 fireUntilHalt （） 的用户或应用程序执行，则终止规则执行。当用户或应用程序调用 fireUntilHalt （） 方法时，决策引擎以 active 模式启动，并评估规则，直到用户或应用程序显式调用 halt （） 方法。否则，决策引擎默认以被动模式 运行，仅当用户或应用程序显式调用 fireAllRules （） 方法时评估规则。

- drools.getKieRuntime （）.getAgenda （） ：返回对 KIE 会话 Agenda 的引用，并提供对规则激活组、规则站组和 ruleflow 组的访问。

访问日程表组"CleanUp"的示例，并设置重点

```
drools.getKieRuntime().getAgenda().getAgendaGroup( "CleanUp" ).setFocus();
```

+ 这个示例将重点设置为规则所属的指定电缆组。

- drools.getKieRuntime （）.setGlobal （）, ~.getGlobal （）, ~.getGlobals （） ：设置或检索全局变量。

- drools.getKieRuntime （）.get Environment （） ：返回运行时环境，类似于您的操作系统环境。

- drools.getKieRuntime （）.getQueryResults (<string> query) ： 运行查询并返回结果。

16.9.3. 具有条件和命名后果的高级规则操作

通常，有效的规则操作是小的、声明性且可读。然而，在有些情况下，每个规则具有单一后果的限制可能有一定难度，从而导致详细和重复的规则语法，如下例所示：

具有详细和重复语法的规则示例

```
rule "Give 10% discount to customers older than 60"
  when
    $customer : Customer( age > 60 )
  then
    modify($customer) { setDiscount( 0.1 ) };
end

rule "Give free parking to customers older than 60"
  when
    $customer : Customer( age > 60 )
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
end
```

重复部分解决方案是使第二规则扩展第一条规则，如以下修改的示例所示：

有扩展条件的部分增强示例规则

```
rule "Give 10% discount to customers older than 60"
  when
    $customer : Customer( age > 60 )
  then
    modify($customer) { setDiscount( 0.1 ) };
end

rule "Give free parking to customers older than 60"
    extends "Give 10% discount to customers older than 60"
  when
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
end
```

作为更有效的替代方法，您可以使用修改后的条件和标记的相应规则操作将这两个规则整合到一条规则中，如下例所示：

带有条件和名为 **results** 的整合示例规则

```
rule "Give 10% discount and free parking to customers older than 60"
  when
    $customer : Customer( age > 60 )
    do[giveDiscount]
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
  then[giveDiscount]
    modify($customer) { setDiscount( 0.1 ) };
end
```

这个示例规则使用两个操作：通常的默认操作以及名为 **giveDiscount** 的另一个操作。当一个超过 **60** 年的客户在 **KIE** 基础中发现时，**giveDiscount** 操作是激活的，无论客户是否拥有回车。

您可以使用额外条件（如以下示例中的 **if** 语句）配置命名结果的激活。**if** 语句中的条件始终评估在之前紧随它的模式。

带有额外条件的合并示例规则

```
rule "Give free parking to customers older than 60 and 10% discount to golden ones among them"
  when
    $customer : Customer( age > 60 )
    if ( type == "Golden" ) do[giveDiscount]
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
  then[giveDiscount]
    modify($customer) { setDiscount( 0.1 ) };
end
```

您还可以使用嵌套（如果构建）等嵌套评估不同的规则条件，如以下更复杂的示例所示：

具有更复杂的条件合并的示例规则

```
rule "Give free parking and 10% discount to over 60 Golden customer and 5% to Silver ones"
  when
    $customer : Customer( age > 60 )
    if ( type == "Golden" ) do[giveDiscount10]
    else if ( type == "Silver" ) break[giveDiscount5]
    $car : Car( owner == $customer )
  then
    modify($car) { setFreeParking( true ) };
  then[giveDiscount10]
    modify($customer) { setDiscount( 0.1 ) };
  then[giveDiscount5]
    modify($customer) { setDiscount( 0.05 ) };
end
```

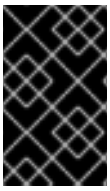
这个示例规则为 **10% 的折扣，免费向高金客户打电话，但只购买 5% 的折扣，而无需免费向 Silver 客户打电话。该规则激活了名为 giveDiscount5 的后果，其关键字为 break 而不是 do。关键字在决策引擎日程表中产生后果，使规则条件的其余部分能够继续评估，同时 中断 会阻止任何进一步的状况评估。如果一个命名后果与具有 do 的任何条件不匹配，但没有通过 中断 激活，则该规则无法编译，因为不会到达规则的条件部分。**

16.10. DRL 文件中的注释

DRL 支持以双正斜杠 // 和带有正斜杠和星号 /* ...*/ 括起的单行注释。您可以使用 DRL 注释来注解 DRL 文件中的规则或任何相关组件。在处理 DRL 文件时，决策引擎会忽略 DRL 注释。

包含注释的规则示例

```
rule "Underage"
  // This is a single-line comment.
  when
    $application : LoanApplication()  // This is an in-line comment.
    Applicant( age < 21 )
  then
    /* This is a multi-line comment
    in the rule actions. */
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```



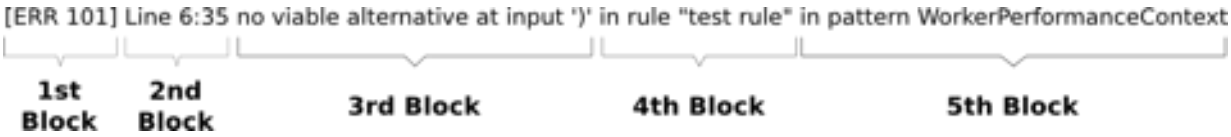
重要

DRL 注释不支持 hash 符号 #。

16.11. DRL 故障排除的错误消息

Red Hat Process Automation Manager 为 DRL 错误提供了标准化的信息，以帮助您进行故障排除并解决 DRL 文件中的问题。错误消息使用以下格式：

图 **16.1. DRL 文件问题的错误消息格式**



- **1st Block：** 错误代码
- **2nd Block：** 发生错误的 DRL 源中的行和列
- **第 3 个块：** 问题的描述
- **4th Block:** 组件在 DRL 源中（rule、function、query）发生错误
- **5th Block：** 发生错误的 DRL 源中的 Pattern（如果适用）

Red Hat Process Automation Manager 支持以下标准化的错误消息：

101：没有可行的替代方案

表示解析器到达决定点，但无法识别替代方案。

带有错误拼写的规则示例

```
1: rule "simple rule"
2:   when
3:     exists Person()
4:     exits Student()  // Must be `exists`
5:   then
6: end
```

*错误**消息***

```
[ERR 101] Line 4:4 no viable alternative at input 'exits' in rule "simple rule"
```

没有**规则**名称的**规则**示例

```
1: package org.drools.examples;
2: rule    // Must be `rule "rule name"` (or `rule rule_name` if no spacing)
3:   when
4:     Object()
5:   then
6:     System.out.println("A RHS");
7: end
```

*错误**消息***

```
[ERR 101] Line 3:2 no viable alternative at input 'when'
```

在这个示例中，解析程序在预期规则名称时遇到关键字，因此当 作为不正确的预期令牌时，它标志是正确的令牌。

带有不正确的语法的规则示例

```
1: rule "simple rule"
2:   when
3:     Student( name == "Andy )  // Must be `"Andy"`
4:   then
5: end
```

错误消息

```
[ERR 101] Line 0:-1 no viable alternative at input '<eof>' in rule "simple rule" in pattern Student
```

注意

行和列值为 **0:-1** 表示解析器到达源文件的结尾(<eof>)，但遇到不完整的结构，通常是因为缺少引号 "..."、postrophes '...' 或括号 (...)。

102：不匹配的输入

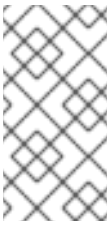
表示解析器预期在当前输入位置缺少特定符号。

带有不完整规则语句的规则示例

```
1: rule simple_rule
2:   when
3:     $p : Person(
       // Must be a complete rule statement
```

*错误**消息***

```
[ERR 102] Line 0:-1 mismatched input '<eof>' expecting ')' in rule "simple rule" in pattern Person
```



注意

行和列**值为 **0:-1 表示解析器到达**源文件的**结尾(<eof>)，但遇到不完整的**结构，通常是因为**缺少引号 "..."、postrophes '...' 或括号 (...)。

*带**有不正确的**语法的**规则**示例***

```
1: package org.drools.examples;
2:
3: rule "Wrong syntax"
4:   when
5:     not( Car( ( type == "tesla", price == 10000 ) || ( type == "kia", price == 1000 ) ) from $carList )
       // Must use `&&` operators instead of commas `,`
6:   then
7:     System.out.println("OK");
8: end
```

*错误**信息***

```
[ERR 102] Line 5:36 mismatched input ',' expecting ')' in rule "Wrong syntax" in pattern Car
[ERR 101] Line 5:57 no viable alternative at input 'type' in rule "Wrong syntax"
[ERR 102] Line 5:106 mismatched input ')' expecting 'then' in rule "Wrong syntax"
```

***在这个示例中，sntactic** 问题会导**致多个**错误**消息相互相关。用 & amp;& 运算符替换逗号 的**单个解决方案解决了所有**错误。 **如果您遇到多个**错误，则**一次解决一个**错误会导**致上一个**错误。*

103：失败的 predicate

表示验证语义 predicate 被评估为 false。这些语义 predicates 通常用于识别 DRL 文件中的组件关键字，如 声明、规则、存在、不存在 等等。

带有无效关键字的规则示例

```
1: package nesting;
2:
3: import org.drools.compiler.Person
4: import org.drools.compiler.Address
5:
6: Some text  // Must be a valid DRL keyword
7:
8: rule "test something"
9:   when
10:     $p: Person( name=="Michael" )
11:   then
12:     $p.name = "other";
13:     System.out.println(p.name);
14: end
```

错误消息

```
[ERR 103] Line 6:0 rule 'rule_key' failed predicate:
{(validateIdentifierKey(DroolsSoftKeywords.RULE))}? in rule
```

有些文本行 无效，因为它没有以 开头或者不是 DRL 关键字结构的一部分，因此解析器无法验证 DRL 的其余部分。

注意

此错误与 102 类似，但通常涉及 DRL 关键字。

104：不允许使用分号结束

表示规则条件中的 eval（） 子句使用分号；不能使用一个分号。

带有 eval（） 和 trailing分号的规则示例

```
1: rule "simple rule"
2:   when
3:     eval( abc(); )  // Must not use semicolon `;`
4:   then
5: end
```

错误消息

```
[ERR 104] Line 3:4 trailing semi-colon not allowed in rule "simple rule"
```

105：与任何内容不匹配

表示，解析程序在 grammar 中到达子规则，该规则必须至少匹配一次替代，但 sub-rule 不匹配任何内容。解析器已输入了一个没有方法的分支。

在空条件中带有无效文本的规则示例

```
1: rule "empty condition"
2:   when
3:     None  // Must remove `None` if condition is empty
4:   then
5:      insert( new Person() );
6: end
```

错误消息

> *[ERR 105] Line 2:2 required (...)+ loop did not match anything at input 'WHEN' in rule "empty condition"*

在本例中，条件旨在为空，但使用单词 None。通过删除 None （不是有效的 DRL 关键字、数据类型或模式结构）解决这个错误。

注意

如果您遇到您无法解决的其他 DRL 错误消息，请联系您的红帽大客户经理。

第 17 章 数据对象

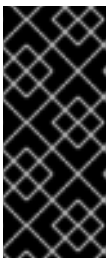
数据对象是您创建的规则资产的构建块。数据对象是在项目指定软件包中作为 Java 对象实施的自定义数据类型。例如，您可以创建一个带有数据字段、address 和 DateOfBirth 的 Person 对象，以指定 loan Application 规则的个人详情。 这些自定义数据类型决定了您的资产和您的决策服务所基于的数据。

17.1. 创建数据对象

以下流程是创建数据对象的一般概述。它并不适用于特定的业务资产。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Data Object。
3. 输入唯一数据对象名称，然后选择您希望数据对象可用于其他规则资产的软件包。 同一名称的数据对象不能位于同一软件包中。在指定的 DRL 文件中，您可以从任何软件包中导入数据对象。



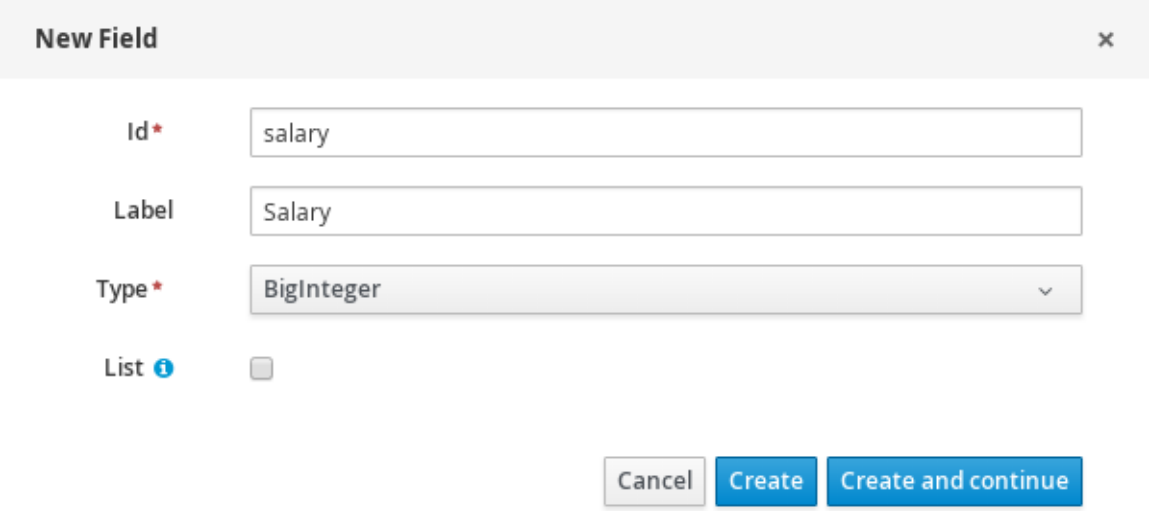
从其他软件包导入数据对象

您可以将另一软件包中的现有数据对象直接导入到资产设计人员，如指导规则或指导决策表设计人员。在项目中选择相关规则资产以及资产设计器，请转至 Data Objects → New item 以选择要导入的对象。

4. 要使数据对象持久化，请选择"永久 "复选框。Persistable 数据对象可以根据 JPA 规格存储在数据库中。默认的 JPA 为 Hibernate。
5. 点 确定。
6. 在数据对象设计器中，点 add 字段在 对象中添加包含属性 Id、标签和 类型的字段。所需的属性标有星号(*)。
   - id： 输入字段的唯一 ID。

- **label:** （可选）输入字段的标签。
- **Type：** 输入字段的数据类型。
- **列表 ：** （可选）选择此复选框，以启用字段保存指定类型的多个项目。

图 17.1. 在数据对象中添加数据字段



7. 点 Create 添加新字段，或者点击 Create 并继续添加新字段并继续添加其他字段。

注意

要编辑字段，请选择字段行，并使用屏幕右侧的 常规属性。

第 18 章 在 BUSINESS CENTRAL 中创建 DRL 规则

您可以在 Business Central 中为您的项目创建和管理 DRL 规则。在每个 DRL 规则文件中，您可以根据您在软件包中创建的或导入的数据对象定义规则条件、操作和其他与该规则相关的组件。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → DRL 文件。
3. 输入信息式 DRL 文件名 并选择相应的 软件包。您指定的软件包必须是分配或将被分配所需数据对象的同一软件包。

   如果项目中已定义了任何域特定语言(DSL)资产，您也可以选择 Show 声明 DSL 句子。这些 DSL 资产将作为在 DRL 设计器中定义的条件和操作可以使用的对象。
4. 点 Ok 创建规则资产。

   新的 DRL 文件现已列在 Project Explorer 的 DRL 面板中，如果您选择了 Show 声明 DSL 句子选项，则在 DSL R 面板中列出。分配此 DRL 文件的软件包列在文件的顶部。
5. 在 DRL 设计器左侧面板中的 fact 类型 列表中，确认您的规则需要的所有数据对象和数据对象字段（每个）。如果没有，您可以使用 DRL 文件中的 import 语句从其他软件包中导入相关数据对象，或者在您的软件包中创建 数据对象。
6. 在所有数据对象都就位后，返回到 DRL 设计器的 Model 选项卡，并使用以下任何组件定义 DRL 文件：

   DRL 文件中的组件

```
package

import

function  // Optional
```

```
query  // Optional

declare   // Optional

global   // Optional

rule "rule name"
    // Attributes
    when
        // Conditions
    then
        // Actions
end

rule "rule2 name"

...
```

- 软件包:(automatic)会在创建 DRL 文件并选择软件包时为您定义。

- 导入 ：使用它来识别来自此软件包或要在 DRL 文件中要使用的其他软件包的数据对象。以 packageName.objectName 格式指定软件包和数据对象，并在单独的行中指定多个导入。

  导入数据对象

  ```
  import org.mortgages.LoanApplication;
  ```

- 功能 ：（可选）使用它在 DRL 文件中包含规则使用的功能。DRL 文件中的功能将语义代码放在规则源文件中，而不是在 Java 类中。如果规则中的某个部分被重复使用，并且每个规则的参数不同，则功能特别有用。在 DRL 文件中的规则上方，您可以声明函数或导入帮助程序类的静态方法作为功能，然后在规则的操作(then)部分中按名称使用函数(then)。

  使用规则（选项 1）声明并使用函数.

  ```
  function String hello(String applicantName) {
  ```

```
    return "Hello " + applicantName + "!";
}

rule "Using a function"
  when
    // Empty
  then
    System.out.println( hello( "James" ) );
end
```

通过规则导入并使用函数（选项 2）

```
import function my.package.applicant.hello;

rule "Using a function"
  when
    // Empty
  then
    System.out.println( hello( "James" ) );
end
```

- **query:（可选）使用此命令搜索与 DRL 文件中规则相关的事实。您可以在 DRL 文件中添加查询定义，然后获取应用程序代码的匹配结果。查询搜索一组定义的条件，且不需要在或 随后 指定规格时使用。查询名称对于 KIE 基础是全局的，因此，在项目中的所有规则查询中必须是唯一的。要返回查询的结果，请使用 ksession.get QueryResults ("name") 构造传统的 QueryResults 定义，其中 "name" 是查询名称。这会返回一个查询结果列表，允许您检索与查询匹配的对象。在 DRL 文件中定义规则之上的查询和查询结果参数。**

DRL 文件中的查询定义示例

```
query "people under the age of 21"
    $person : Person( age < 21 )
end
```

获取查询结果的应用程序代码示例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
System.out.println( "we have " + results.size() + " people under the age  of 21" );
```

- 声明：（可选）使用它声明由 DRL 文件中的规则使用的新事实类型。Red Hat Process Automation Manager 的 java.lang 包中的默认事实类型是 对象，但您可以根据需要声明 DRL 文件中的其他类型。在 DRL 文件中声明事实类型可让您直接在决策引擎中定义新的事实模型，而无需在 Java 等低级别语言中创建模型。

  声明并使用新的事实类型

```
declare Person
  name : String
  dateOfBirth : java.util.Date
  address : Address
end

rule "Using a declared type"
  when
    $p : Person( name == "James" )
  then   // Insert Mark, who is a customer of James.
    Person mark = new Person();
    mark.setName( "Mark" );
    insert( mark );
end
```

- global：（可选）使用它在 DRL 文件中包括规则要使用的全局变量。全局变量通常提供规则的数据或服务，如规则结果中使用的应用程序服务，并从规则返回数据，如规则中添加的日志或值。通过 KIE 会话配置或 REST 操作在决策引擎的工作内存中设置全局值，声明 DRL 文件中的规则以外的全局变量，然后在规则的操作（然后）部分使用。对于多个全局变量，请在 DRL 文件中使用单独的行。

  为决策引擎设置全局列表配置

```
List<String> list = new ArrayList<>();
KieSession kieSession = kiebase.newKieSession();
kieSession.setGlobal( "myGlobalList", list );
```

在规则中定义全局列表

```
global java.util.List myGlobalList;

rule "Using a global"
  when
    // Empty
  then
    myGlobalList.add( "My global list" );
end
```

警告

不要使用全局变量在规则中建立条件，除非全局变量具有恒定的不可变值。全局变量不会插入到决策引擎的工作内存中，因此决策引擎无法跟踪变量的值更改。

不要使用全局变量在规则间共享数据。规则始终原因并响应正常工作的内存状态。因此，如果要将数据从规则传递给规则，请向决策引擎的工作内存中断言数据。

- **规则：使用这个方法在 DRL 文件中定义每个规则。规则包含格式规则 "name" 的规则，后跟定义规则行为的可选属性（如 salience 或 no-loop），后面是 when 和 then 定义。每个规则必须在规则软件包中具有唯一的名称。规则的 when 部分包含必须符合执行操作的条件。例如，如果银行需要 loan applicants 已有 21 余年，则 "Underage" 规则的 when 条件将是适用的 (age < 21)。规则的 then 部分包含满足规则条件部分时要执行的操作。例如，当 loan applicant 旧有 21 年时，然后 操作将被 设置(false)，因为申请者处于年龄之内。**

loan Application age 限值的规则

```
rule "Underage"
  salience 15
  when
    $application : LoanApplication()
    Applicant( age < 21 )
  then
    $application.setApproved( false );
    $application.setExplanation( "Underage" );
end
```

每个 DRL 文件必须至少指定软件包、导入、导入和规则组件。所有其他组件都是可选的。

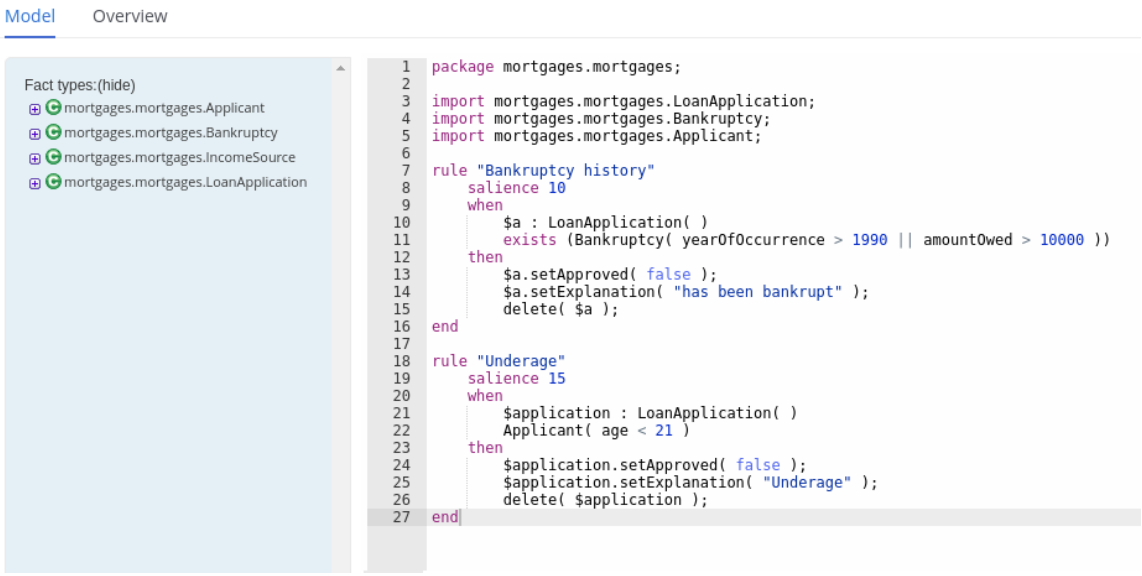
以下是 loan 应用程序决策服务中的 DRL 文件示例：

loan application 的 DRL 文件示例

```
package org.mortgages;

import org.mortgages.LoanApplication;
import org.mortgages.Bankruptcy;
import org.mortgages.Applicant;

rule "Bankruptcy history"
 salience 10
 when
  $a : LoanApplication()
  exists (Bankruptcy( yearOfOccurrence > 1990 || amountOwed > 10000 ))
 then
  $a.setApproved( false );
  $a.setExplanation( "has been bankrupt" );
  delete( $a );
end

rule "Underage"
 salience 15
 when
  $application : LoanApplication()
  Applicant( age < 21 )
 then
  $application.setApproved( false );
```

```
    $application.setExplanation( "Underage" );
    delete( $application );
end
```

图 18.1. Business Central 中 loan application 的 DRL 文件示例



7. 在定义了规则的所有组件后，在 DRL 设计器右上角点击 Validate 来验证 DRL 文件。如果文件验证失败，解决错误消息中描述的任何问题，查看 DRL 文件中的所有语法和组件，并尝试验证文件，直到文件通过为止。

8. 点 DRL 设计器中的 Save 保存您的更改。

18.1. 在 DRL 规则中添加 WHEN 条件

规则的 when 部分包含必须符合执行操作的条件。例如，如果银行需要 loan applicants 已有 21 余年，那么 "Underage" 规则的 when 条件将是适用的 （年龄 < 21 ）。条件由一系列指定模式和约束组成，具有可选的绑定和其他支持的 DRL 元素，它们基于软件包中的可用数据对象。

先决条件

- 软件包在 DRL 文件的顶部定义。在创建该文件时，应该已为您完成这一操作。
- 规则中使用的数据对象 导入 列表在 DRL 文件的 package 行中定义。数据对象可从这个软件包，或者从 Business Central 中的其他软件包中获取。

- 规则名称在软件包下面以 "name" 的格式定义，导入，以及应用到整个 DRL 文件的其他行。同一软件包中不能多次使用相同的规则名称。可选的规则属性（如 sali ence 或 no-loop）在 when 部分前定义规则名称下。

流程

1. 在 DRL 设计程序中，在规则中输入以开始添加条件语句。when 部分包含零个或更多事实模式，它们为规则定义条件。

   如果 when 部分为空，则条件被视为 true，在决策引擎中第一次执行 fireAllRules（） 调用。如果要使用规则设置决策引擎状态，这很有用。

   没有条件的规则示例

```
rule "Always insert applicant"
  when
    // Empty
  then   // Actions to be executed once
    insert( new Applicant() );
end

// The rule is internally rewritten in the following way:

rule "Always insert applicant"
  when
    eval( true )
  then
    insert( new Applicant() );
end
```

2. 输入第一个条件的模式，带有可选的限制、绑定和其他支持的 DRL 元素。基本模式格式为 **<patternBinding> : <patternType>(<constraints>)**。模式基于软件包中的可用数据对象，并定义要在 then 部分中触发操作的条件。

   - 简单模式：没有约束的简单模式与给定类型的事实匹配。例如，以下条件仅存在 applicant。

```
when
  Applicant()
```

- 具有限制的模式：具有约束的模式与给定类型的事实匹配，以及以括号括起的额外限制。例如，以下条件是 applicant 为 21 的年龄。

```
when
  Applicant( age < 21 )
```

- 带有绑定模式：模式的绑定是规则的其他组件可以使用的简写引用来引用定义的模式。例如，以下绑定到 LoanApplication 的相关操作适用于age applicants。

```
when
  $a : LoanApplication()
  Applicant( age < 21 )
then
  $a.setApproved( false );
  $a.setExplanation( "Underage" )
```

3. 继续定义适用于此规则的所有条件模式。以下是定义 DRL 条件的一些关键字选项：

- 和 ：使用这个条件组件将条件组件分组到逻辑中。支持在fix 和 前缀中。默认情况下，所有列出的模式都与 和 结合使用，并且 未指定任何组合。

```
// All of the following examples are interpreted the same way:
$a : LoanApplication() and Applicant( age < 21 )

$a : LoanApplication()
and Applicant( age < 21 )

$a : LoanApplication()
Applicant( age < 21 )

(and $a : LoanApplication() Applicant( age < 21 ))
```

- 或 ：使用这个条件组件将条件组件分组到逻辑不满中。支持在修复和 前缀或 支持中。

```
// All of the following examples are interpreted the same way:
Bankruptcy( amountOwed == 100000 ) or IncomeSource( amount == 20000 )

Bankruptcy( amountOwed == 100000 )
or IncomeSource( amount == 20000 )

(or Bankruptcy( amountOwed == 100000 ) IncomeSource( amount == 20000 ))
```

- 存在 ：使用它来指定必须存在的事实和约束。这个选项仅在第一个匹配项中触发，而不是后续匹配项。如果您将这个元素与多个模式搭配使用，请将模式与括号 （） 括起。

```
exists ( Bankruptcy( yearOfOccurrence > 1990 || amountOwed > 10000 ) )
```

- **Not ：使用它来指定必须不存在的事实和约束。**

```
not ( Applicant( age < 21 ) )
```

- **all ：使用这个方法验证所有与第一个模式匹配的事实是否与所有剩余的模式匹配。当满足总 结构时，该规则将评估为 true。**

```
forall( $app : Applicant( age < 21 )
        Applicant( this == $app, status = 'underage' ) )
```

- **从 ：使用这个方法为模式指定数据源。**

```
Applicant( ApplicantAddress : address )
Address( zipcode == "23920W" ) from ApplicantAddress
```

- **入口点 ：使用此定义与模式匹配的 Entry Point。通常与 的 一同使用。**

```
Applicant() from entry-point "LoanApplication"
```

- **collect ：使用它来定义规则可用作条件的一部分的对象集合。在示例中，每个给定影片的决定引擎中所有待处理的应用程序都分组到 列表中。如果找到三个或更多待处理的应用程序，则会执行该规则。**

```
$m : Mortgage()
$a : List( size >= 3 )
   from collect( LoanApplication( Mortgage == $m, status == 'pending' ) )
```

- **accumulate ：使用它迭代对象集合，为每个元素执行自定义操作，并返回一个或多个结果对象（如果限制评估为 true）。这个选项是更灵活且强大的 收集 形式。使用 accumulate (<source pattern>; <functions> [;<constraints>])。在示例中， min、 max 和 average 是积累的功能， 它计算每个传感器的所有读取的最小、最大值和平均温度值。其他支持的功能包括 count、 sum、 lari ance、 StandardDev iation、 collectList 和 collectSet。**

```
$s : Sensor()
accumulate( Reading( sensor == $s, $temp : temperature );
        $min : min( $temp ),
```

```
$max : max( $temp ),
$avg : average( $temp );
$min < 20, $avg > 70 )
```

注意

有关 DRL 规则状况的更多信息，请参阅 第 16.8 节 “DRL 中的规则条件 (WHEN)”。

4. 在定义了规则的所有条件组件后，在 DRL 设计器右上角点击 Validate 来验证 DRL 文件。如果文件验证失败，解决错误消息中描述的任何问题，查看 DRL 文件中的所有语法和组件，并尝试验证文件，直到文件通过为止。

5. 点 DRL 设计器中的 Save 保存您的更改。

18.2. 在 DRL 规则中添加存储操作

规则的 then 部分包含满足规则条件部分时要执行的操作。例如，当 loan ApplicationIicant 超过 21 年时，将设置 "Underage" 规则的操作，因为申请者不足，因为申请者正处于年龄之内。操作由一个或多个方法组成，它们根据规则条件和软件包中的可用数据对象执行结果。规则操作的主要用途是在决策引擎工作内存中插入、删除或修改数据。

先决条件

- 软件包在 DRL 文件的顶部定义。在创建该文件时，应该已为您完成这一操作。
- 规则中使用的数据对象 导入 列表在 DRL 文件的 package 行中定义。数据对象可从这个软件包，或者从 Business Central 中的其他软件包中获取。
- 规则 名称在 软件包 下面以 "name" 的格式定义，导入，以及应用到整个 DRL 文件的其他行。同一软件包中不能多次使用相同的规则名称。可选的规则属性（如 sali ence 或 no-loop）在 when 部分前定义规则名称下。

流程

1. 在 DRL 设计器中， 在规则的 when 部分后输入，以开始添加操作语句。

2.

根据规则的条件，基于事实模式输入要执行的一个或多个操作。

以下是定义 DRL 操作的一些关键字选项：

- 设置 ：使用此项设置字段的值。

```
$application.setApproved ( false );
$application.setExplanation( "has been bankrupt" );
```

- 修改 ：使用此命令指定要修改的字段，并通知更改的决策引擎。此方法提供了一种结构化方法来事实更新。它将 更新操作与 设置者调用合并，以更改对象字段。

```
modify( LoanApplication ) {
        setAmount( 100 ),
        setApproved ( true )
}
```

- 更新 ：使用它来指定字段以及要更新整个相关事实，并通知更改的决策引擎。事实更改后，您必须在更改可能受更新的值影响的其他事实之前调用 更新。要避免添加的步骤，请改为使用 修改 方法。

```
LoanApplication.setAmount( 100 );
update( LoanApplication );
```

- 插入 ：使用这个方法在决策引擎中插入 新事实。

```
insert( new Applicant() );
```

- 插入逻辑 ：使用这个方法在决策引擎中以逻辑方式插入新事实。决策引擎负责对插入和检索事实的逻辑决策。在常规或声明插入后，必须明确调整事实。逻辑插入后，当插入事实的条件不再为 **true** 时，插入的事实会自动清空。

```
insertLogical( new Applicant() );
```

- 删除 ：使用它来从决策引擎删除对象。DRL 中还支持关键字 **retract**，并执行同样的操作，但 删除 通常优先于 DRL 代码，以便通过关键字 插入。

```
delete( Applicant );
```

注意

有关 DRL 规则操作的详情请参考 第 16.9 节 “DRL(THEN)中的规则操作”。

3. 在定义了规则的所有操作组件后，在 DRL 设计器的右上角点击 Validate 来验证 DRL 文件。如果文件验证失败，解决错误消息中描述的任何问题，查看 DRL 文件中的所有语法和组件，并尝试验证文件，直到文件通过为止。

4. 点 DRL 设计器中的 Save 保存您的更改。

第 19 章 执行规则

在确定了示例规则或在 Business Central 中创建自己的规则后，您可以在本地或 KIE 服务器上构建并部署相关的项目并在 KIE 服务器上执行规则，以测试规则。

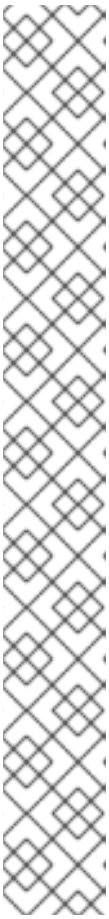
先决条件

- 业务中心和 KIE 服务器已安装并运行。有关安装选项，请参阅 规划 Red Hat Process Automation Manager 安装。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 在项目 资产 页面右上角，单击 Deploy 以构建该项目，并将它部署到 KIE 服务器。如果构建失败，请解决屏幕底部的 Alerts 面板中描述的问题。

   有关项目部署选项的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。



注意

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

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

默认情况下，Red Hat Process Automation Manager 中的规则资产需要这个依赖项，从可执行规则模型构建。这个依赖关系包含在 Red Hat Process Automation Manager 核心打包中，但取决于 Red Hat Process Automation Manager 升级历史记录，您可能需要手动添加这个依赖关系来启用可执行规则模型行为。

有关可执行规则模型的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。

3. 在 Business Central 之外创建 Maven 或 Java 项目（如果尚未创建），您可将其用于在本地执行规则，或用作在 KIE 服务器上执行规则的客户端应用程序。该项目必须包含 pom.xml 文件以及执行项目资源的任何其他必要组件。

   有关 test 项目示例，请参阅 "创建和执行 DRL 规则的方法 "。

4. 打开 test 项目或客户端应用程序的 pom.xml 文件，并添加以下依赖项（如果还没有添加）：

   - kie-ci ：启用客户端应用程序以使用 ReleaseId在本地加载 Business Central 项目数据
   - kie-server-client ：使您的客户端应用程序能够与 KIE 服务器上的资产远程交互
   - slf4j:（可选）使您的客户端应用程序能够使用 Simple Logging Facade 用于 Java(SLF4J)，以在与 KIE 服务器交互后返回调试信息

   客户端应用程序 pom.xml 文件中的 Red Hat Process Automation Manager 7.13 的依赖项示例：

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

   对于这些工件的可用版本，请在 Nexus Repository Manager 在线搜索组 ID 和工件 ID。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

5. 确保在客户端应用 pom.xml 文件中定义了包含模型类的工件依赖项，它们与部署的项目的 pom.xml 文件中完全相同。如果模型类的依赖关系因客户端应用和项目之间有所不同，则可能会出现执行错误。

要访问 Business Central 中的项目 pom.xml 文件，请在项目左侧选择任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

例如，以下 Person 类依赖项同时出现在客户端和部署的项目 pom.xml 文件中：

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. 如果您将 slf4j 依赖项添加到用于调试日志的客户端应用程序 pom.xml 文件，请在相关类路径（例如，在 Maven 中的 src/main/resources/META-INF）上创建一个 simplelogger.properties 文件（例如，在 Maven 中的 src/main/resources/META-INF）：

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

-

7.

在您的客户端应用中，创建一个包含必要导入和 main（） 方法的 .java 主类，以加载 KIE 基础、插入事实和执行规则。

例如，项目中的 Person 对象包含 getter 和 setter 方法，用于设置并检索一个人的名字、姓氏、每小时速率和分流。项目中的以下 Wage 规则计算 wage 和 hourly 速率值，并根据结果显示消息：

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

要在 KIE 服务器之外测试此规则（如果需要），请配置 .java 类以导入 KIE 服务、KIE 容器和 KIE 会话，然后使用 main（） 方法针对定义的事实模型触发所有规则：

本地执行规则

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

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

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

      // Set up the fact model:
      Person p = new Person();
```

```
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert the person into the session:
        kSession.insert(p);

        // Fire all rules:
        kSession.fireAllRules();
        kSession.dispose();
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
  }
```

要在 KIE 服务器上测试此规则，请配置 .java 类，其导入和规则执行信息与本地示例类似，另外还指定 KIE 服务配置和 KIE 服务客户端详情：

在 KIE 服务器上执行规则

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {
```

```
  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert Person into the session:
      KieCommands kieCommands = KieServices.Factory.get().getCommands();
      List<Command> commandList = new ArrayList<Command>();
      commandList.add(kieCommands.newInsert(p, "personReturnId"));

      // Fire all rules:
      commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
      BatchExecutionCommand batch =
kieCommands.newBatchExecution(commandList, sessionName);

      // Use rule services client to send request:
      RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
      ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
      System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

8.
从项目目录中运行配置的 .java 类。您可以在开发平台（如 Red Hat CodeReady Studio）或命令行中运行该文件。

Maven 执行示例（在项目目录中）：

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

Java 执行示例（在项目目录中）

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

9.
在命令行中检查规则执行状态并在服务器日志中。如果有任何规则没有按预期执行，请检查项目中配置的规则，以及验证提供的数据的主类配置。

第 20 章 其他创建和执行 DRL 规则的方法

作为在 Business Central 界面中创建和管理 DRL 规则的替代方法，您可以使用 Red Hat CodeReady Studio 或者另一个集成的开发环境(IDE)在外部创建 DRL 规则文件作为 Maven 或 Java 项目的一部分。然后，这些独立项目可集成在 Business Central 的现有 Red Hat Process Automation Manager 项目中的内容 JAR(KJAR)依赖关系。独立项目中的 DRL 文件必须至少包含必要的软件包规格、导入 列表 和规则 定义。其他 DRL 组件（如全局变量和功能）都是可选的。与 DRL 规则相关的所有数据对象必须包含在独立 DRL 项目或部署中。

您还可以在 Maven 或 Java 项目中使用可执行规则模型，以提供构建时要执行的规则集的基于 Java 的表示。可执行模型是 Red Hat Process Automation Manager 中标准资产打包的更有效替代，并让 KIE 容器和 KIE 基础能够更快创建，特别是在您拥有大量 DRL（Drools 规则语言）文件和其他 Red Hat Process Automation Manager 资产列表。

20.1. 使用 JAVA 创建并执行 DRL 规则

您可以使用 Java 对象创建 DRL 文件与规则，并将对象与红帽流程自动化管理器决策服务集成。如果您已为您的决策服务使用外部 Java 对象，并且希望继续相同的工作流，则创建 DRL 规则的方法很有用。如果您还没有使用这个方法，则建议红帽流程自动化管理器的 Business Central 界面创建 DRL 文件和其他规则资产。

流程

1. 创建运行规则或规则的 Java 对象。

   在本例中，Person.java 文件是在 my-project 目录中创建的。Person 类包含 getter 和 setter 方法，用于设置和检索第一个名称、姓氏、每小时速率和个人的 wage：

```
public class Person {
  private String firstName;
  private String lastName;
  private Integer hourlyRate;
  private Integer wage;

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }
```

```
    public void setLastName(String lastName) {
      this.lastName = lastName;
    }

    public Integer getHourlyRate() {
      return hourlyRate;
    }

    public void setHourlyRate(Integer hourlyRate) {
      this.hourlyRate = hourlyRate;
    }

    public Integer getWage(){
      return wage;
    }

    public void setWage(Integer wage){
      this.wage = wage;
    }
  }
```

2. 在 my-project 目录下，以 .drl 格式创建一个规则文件。DRL 文件必须至少包含软件包规格（如果适用），由规则或规则使用的数据对象导入列表，以及条件和动作时使用的一个或多个规则。

以下 Wage.drl 文件包含一个 Wage 规则，它计算 wage 和 hourly 速率值，并显示基于结果的消息：

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

3. 创建主类，并将它保存到与您创建的 Java 对象相同的目录中。主类将加载 KIE 基础并执行规则。

4. 在主类中，添加所需的 导入 语句来导入 KIE 服务、KIE 容器和 KIE 会话。然后，载入 KIE 基础、插入事实，并从将事实模型传递给规则的 main（） 方法执行规则。

在本例中，my-project 中创建了一个 RulesTest.java 文件，其中包含所需的导入和 main（） 方法：

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

public class RulesTest {
  public static final void main(String[] args) {
    try {
      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.getKieClasspathContainer();
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

5. 从红帽客户门户下载 Red Hat Process Automation Manager 7.13.5 Source Distribution ZIP 文件，并将它提取到 my-project/pam-engine-jars/ 下。

6. 在 my-project/META-INF 目录中，创建包含以下内容的 kmodule.xml 元数据文件：

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

这个 kmodule.xml 文件是一个 KIE 模块描述符，它选择到 KIE 基础并配置会话。此文件可让您定义和配置一个或多个 KIE 基础，并在特定 KIE 基础中包含 特定软件包 的 DRL 文件。您还

可以从每个 KIE 基础中创建一个或多个 KIE 会话。

以下示例显示了更高级的 kmodule.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.drools.org/xsd/kmodule">
  <kbase name="KBase1" default="true" eventProcessingMode="cloud"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">
    <ksession name="KSession1_1" type="stateful" default="true" />
    <ksession name="KSession1_2" type="stateful" default="true" beliefSystem="jtms" />
  </kbase>
  <kbase name="KBase2" default="false" eventProcessingMode="stream"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
      <fileLogger file="debugInfo" threaded="true" interval="10" />
      <workItemHandlers>
        <workItemHandler name="name" type="new org.domain.WorkItemHandler()" />
      </workItemHandlers>
      <listeners>
        <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener" />
        <agendaEventListener type="org.domain.FirstAgendaListener" />
        <agendaEventListener type="org.domain.SecondAgendaListener" />
        <processEventListener type="org.domain.ProcessListener" />
      </listeners>
    </ksession>
  </kbase>
</kmodule>
```

这个示例定义了两个 KIE 基础。两个 KIE 会话是由 KBase1 KIE 基础实例化，以及来自 KBase2 的 KIE 会话。KBase2 中的 KIE 会话是一个 无状态 KIE 会话，这意味着之前调用 KIE 会话（之前会话状态）中的数据会在会话调用之间丢弃。规则 资产的软件包包括在 KIE 基础中。当您以这种方式指定软件包时，您必须将 DRL 文件组织到反映指定软件包的文件夹结构中。

7. 在您的 Java 对象中创建并保存所有 DRL 资产后，导航到命令行中的 my-project 目录，再运行以下命令来构建 Java 文件。将 RulesTest.java 替换为您的 Java 主类的名称。

```
javac -classpath "./pam-engine-jars/*:." RulesTest.java
```

如果构建失败，请解决命令行错误消息中描述的任何问题，并尝试验证 Java 对象，直到对象通过为止。

8. 在您的 Java 文件构建成功后，运行以下命令在本地执行规则。将 RulesTest 替换为您的 Java 主类前缀。

```
java -classpath "./pam-engine-jars/*:." RulesTest
```

9. 检查规则，以确保它们正确执行，并解决 Java 文件中需要的任何更改。

要将新规则资产与红帽流程自动化管理器中的现有项目集成，您可以将新的 Java 项目作为知识 JAR(KJAR)编译，并将它添加为 Business Central 中项目的 pom.xml 文件中的依赖关系。要访问 Business Central 中的项目 pom.xml 文件，您可以选择项目中任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

20.2. 使用 MAVEN 创建和执行 DRL 规则

您可以使用 Maven archetypes 创建 DRL 文件与规则，并将 archetypes 与 Red Hat Process Automation Manager 决策服务集成。如果您已将外部 Maven archetypes 用于决策服务，并且希望继续相同的工作流，则创建 DRL 规则的方法很有用。如果您还没有使用这个方法，则建议红帽流程自动化管理器的 Business Central 界面创建 DRL 文件和其他规则资产。

流程

1. 进入要创建 Maven archetype 并运行以下命令的目录：

```
mvn archetype:generate -DgroupId=com.sample.app -DartifactId=my-app -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

这会创建一个带有以下结构的目录 **my-app** ：

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- sample
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- sample
                    `-- app
                        `-- AppTest.java
```

my-app 目录包含以下关键组件：

- 用于存储应用程序源的 src/main 目录
- 用于存储测试源的 src/test 目录
- 带有项目配置的 pom.xml 文件

2. 创建规则或规则在 Maven archetype 中操作的 Java 对象。

在本例中，Person.java 文件是在 my-app/src/main/java/com/sample/app 目录中创建的 Person.java 文件。Person 类包含 getter 和 setter 方法，用于设置和检索第一个名称、姓氏、每小时速率和个人的 wage：

```
package com.sample.app;

public class Person {

  private String firstName;
  private String lastName;
  private Integer hourlyRate;
  private Integer wage;

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  public Integer getHourlyRate() {
    return hourlyRate;
  }

  public void setHourlyRate(Integer hourlyRate) {
    this.hourlyRate = hourlyRate;
  }

  public Integer getWage(){
    return wage;
```

```
  }

  public void setWage(Integer wage){
    this.wage = wage;
  }
}
```

3. 以 my-app/src/main/resources/com/sample/app 格式创建一个规则文件。 DRL 文件必须至少包含软件包规格、规则或规则要使用的数据对象导入列表，以及带有条件和动作的一个或多个规则。

以下 Wage.drl 文件包含导入 Person 类的 Wage 规则，计算出 wage 和 hourly 速率值，并根据结果显示一条消息：

```
package com.sample.app;

import com.sample.app.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

4. 在 my-app/src/main/resources/META-INF 目录中，创建包含以下内容的 kmodule.xml 元数据文件：

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

这个 kmodule.xml 文件是一个 KIE 模块描述符，它选择到 KIE 基础并配置会话。此文件可让您定义和配置一个或多个 KIE 基础，并在特定 KIE 基础中包含 特定软件包 的 DRL 文件。您还可以从每个 KIE 基础中创建一个或多个 KIE 会话。

以下示例显示了更高级的 kmodule.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.drools.org/xsd/kmodule">
  <kbase name="KBase1" default="true" eventProcessingMode="cloud"
```

```
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">
    <ksession name="KSession1_1" type="stateful" default="true" />
    <ksession name="KSession1_2" type="stateful" default="true" beliefSystem="jtms" />
  </kbase>
  <kbase name="KBase2" default="false" eventProcessingMode="stream"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
      <fileLogger file="debugInfo" threaded="true" interval="10" />
      <workItemHandlers>
        <workItemHandler name="name" type="new org.domain.WorkItemHandler()" />
      </workItemHandlers>
      <listeners>
        <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener" />
        <agendaEventListener type="org.domain.FirstAgendaListener" />
        <agendaEventListener type="org.domain.SecondAgendaListener" />
        <processEventListener type="org.domain.ProcessListener" />
      </listeners>
    </ksession>
  </kbase>
</kmodule>
```

这个示例定义了两个 KIE 基础。两个 KIE 会话是由 KBase1 KIE 基础实例化，以及来自 KBase2 的 KIE 会话。KBase2 中的 KIE 会话是一个 无状态 KIE 会话，这意味着之前调用 KIE 会话（之前会话状态）中的数据会在会话调用之间丢弃。规则 资产的软件包包括在 KIE 基础中。当您以这种方式指定软件包时，您必须将 DRL 文件组织到反映指定软件包的文件夹结构中。

5. *在 my-app/pom.xml 配置文件中，指定应用程序所需的库。提供 Red Hat Process Automation Manager 依赖项，以及应用程序的 组ID、工件 ID 和版本 (GAV)。*

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.sample.app</groupId>
<artifactId>my-app</artifactId>
<version>1.0.0</version>
<repositories>
  <repository>
    <id>jboss-ga-repository</id>
    <url>http://maven.repository.redhat.com/ga/</url>
  </repository>
</repositories>

<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
```

```
    <dependency>
      <groupId>org.drools</groupId>
      <artifactId>drools-compiler</artifactId>
      <version>VERSION</version>
    </dependency>
    <dependency>
      <groupId>org.drools</groupId>
      <artifactId>drools-mvel</artifactId>
      <version>VERSION</version>
    </dependency>
    <dependency>
      <groupId>org.kie</groupId>
      <artifactId>kie-api</artifactId>
      <version>VERSION</version>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  </project>
```

有关 Red Hat Process Automation Manager 中的 Maven 依赖项和 BOM（主要的资料）的详情，请参考 Red Hat Process Automation Manager 和 Maven 库版本之间的映射？

6. 使用 my-app/src/test/java/com/sample/app/AppTest.java 中的 testApp 方法来测试规则。AppTest.java 文件默认由 Maven 创建。

7. 在 AppTest.java 文件中，添加所需的 导入 语句来导入 KIE 服务、KIE 容器和 KIE 会话。然后，载入 KIE 基础、插入事实，并从将事实模型传递给规则的 testApp（） 方法执行规则。

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

public void testApp() {

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

  // Set up the fact model:
  Person p = new Person();
  p.setWage(12);
  p.setFirstName("Tom");
  p.setLastName("Summers");
  p.setHourlyRate(10);
```

```
    // Insert the person into the session:
    kSession.insert(p);

    // Fire all rules:
    kSession.fireAllRules();
    kSession.dispose();
  }
```

8. 在 Maven archetype 中创建并保存所有 DRL 资产后，导航到命令行中的 my-app 目录，再运行以下命令来构建您的文件：

```
mvn clean install
```

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

9. 文件构建成功后，请运行以下命令在本地执行规则。将 com.sample.app 替换为您的软件包名称。

```
mvn exec:java -Dexec.mainClass="com.sample.app.App"
```

10. 检查规则以确保它们正确执行，并处理文件中所需的所有更改。

要将新规则资产与红帽流程自动化管理器中的现有项目集成，您可以将新的 Maven 项目作为知识 JAR(KJAR)编译，并将它添加为 Business Central 中项目的 pom.xml 文件中的依赖关系。要访问 Business Central 中的项目 pom.xml 文件，您可以选择项目中任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

第 21 章 RED HAT PROCESS AUTOMATION MANAGER 中用于 IDE 的示例

Red Hat Process Automation Manager 提供了以 Java 类分发的示例决策，您可导入到您的集成开发环境(IDE)中。您可以使用这些示例来更好地了解决策引擎功能，或者将其用作您在 Red Hat Process Automation Manager 项目中定义的决策的参考。

以下示例决策集是 Red Hat Process Automation Manager 中提供的一些示例：

- hello World 示例 ：demonstrates 基本规则执行和使用调试输出
- State 示例 ：通过规则先和日程组展示转发链和冲突解决
- Fibonacci 示例 ：通过规则 salience 递归和冲突解决
- 银行示例 ：demonstrates 模式匹配、基本排序和计算
- pet Store 示例 ：演示规则映射组、全局变量、回调和 GUI 集成
- Sudoku 示例 ：演示复杂模式匹配、问题解决、回调和 GUI 集成
- Doom 示例内部 ：演示后链和递归

注意

有关红帽构建的 OptaPlanner 提供的优化示例，请参阅开始使用 Red Hat build of OptaPlanner。

21.1. 在 IDE 中导入和执行 RED HAT PROCESS AUTOMATION MANAGER 示例决策

您可以将 Red Hat Process Automation Manager 示例决策导入到您的集成开发环境(IDE)中，再执行它们来探索规则和代码功能。您可以使用这些示例来更好地了解决策引擎功能，或者将其用作您在 Red Hat Process Automation Manager 项目中定义的决策的参考。

先决条件

- 已安装 Java 8 或更高版本。
- 已安装 Maven 3.5.x 或更高版本。
- 安装了 IDE，如 Red Hat CodeReady Studio。

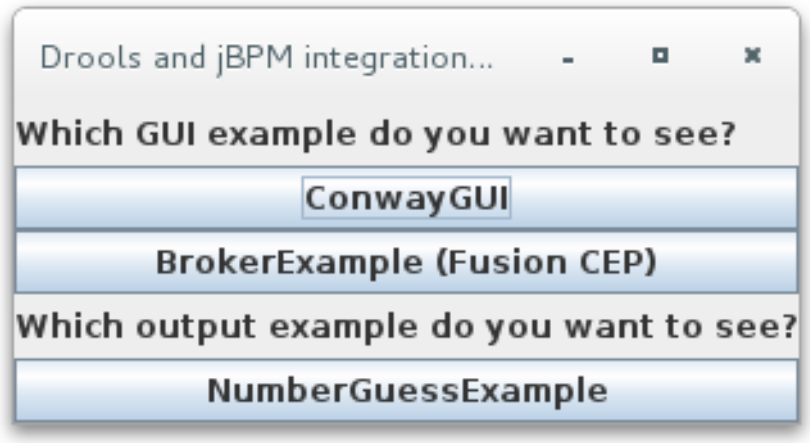
流程

1. 将 Red Hat Process Automation Manager 7.13.5 源分发从红帽客户门户网站下载到临时目录，如 /rhpam-7.13.5-sources。
2. 打开 IDE，然后选择 File → Import → Maven → Existing Maven Projects，或用于导入 Maven 项目的对等选项。
3. 单击 Browse，导航到 ~/rhpam-7.13.5-sources/src/drools-$VERSION/drools-examples（或，对于 Life 示例的 Conway's Game、~/rhpam-7.13.5-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples），再导入该项目。
4. 导航到您要运行的示例软件包，并使用 主 方法查找 Java 类。
5. 右键点击 Java 类并选择 Run As → Java Application 以运行示例。

要通过基本用户界面运行所有示例，请在 org.drools.examples 主类中运行 DroolsExamplesApp.java 类（或者，在 Conway 的 Game of Life 中）运行 DroolsJbpmIntegrationExamplesApp.java 类。

图 21.1. drools-examples(DroolsExamplesApp.java)中的所有示例接口

图 21.2. droolsjbpm-integration-examples(DroolsJbpmIntegrationExamplesApp.java)中的所有示例接口



21.2. HELLO WORLD 示例决策（基本规则和调试）

Hello World 示例决策集演示了如何将对象插入到决策引擎工作内存中，如何使用规则匹配对象，以及如何配置日志记录以跟踪决策引擎的内部活动。

以下是 Hello World 示例的概述：

- 名称 ：helloworld
- 主要课程:org.drools.examples.helloworld.HelloWorldExample （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.helloworld.HelloWorld.drl (src/main/resources)
- 目标 ：演示基本规则执行和使用调试输出

在 Hello World 示例中，会生成一个 KIE 会话，以启用规则执行。所有规则都需要一个 KIE 会话来执行。

用于规则执行的 KIE 会话

```
KieServices ks = KieServices.Factory.get(); (1)
KieContainer kc = ks.getKieClasspathContainer(); (2)
KieSession ksession = kc.newKieSession("HelloWorldKS"); (3)
```

(1) 获取 KieServices 工厂。这是应用程序用来与决策引擎交互的主要界面。

(2) 从项目类路径创建 KieContainer。这会检测 /META-INF/kmodule.xml 文件，该文件使用 KieModule 配置并实例化 KieContainer。

(3) 根据 /META-INF/kmodule.xml 文件中定义的 "HelloWorldKS" KIE 会话配置创建一个 KieSession。

注意

有关 Red Hat Process Automation Manager 项目打包的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

Red Hat Process Automation Manager 有一个公开内部引擎活动的事件模型。两个默认调试监听器： DebugAgendaEventListener 和 DebugRuleRuntimeEventListener，将调试事件信息打印到 System.err 输出。KieRuntimeLogger 提供执行审核，您可以在图形查看器中查看结果。

调试监听器和审计记录

```
// Set up listeners.
ksession.addEventListener( new DebugAgendaEventListener() );
ksession.addEventListener( new DebugRuleRuntimeEventListener() );

// Set up a file-based audit logger.
KieRuntimeLogger logger = KieServices.get().getLoggers().newFileLogger( ksession,
```

```
"./target/helloworld" );

// Set up a ThreadedFileLogger so that the audit view reflects events while debugging.
KieRuntimeLogger logger = ks.getLoggers().newThreadedFileLogger( ksession,
"./target/helloworld", 1000 );
```

日志记录器是在 Agenda 和 RuleRuntime 监听程序基础上构建的专用实现。在决策引擎执行完成后，会调用 logger.close（）。

这个示例创建了一个带有消息 "Hello World" 的 Message 对象，将 status HELLO 插入到 KieSession 中，使用 fireAllRules（） 执行规则。

数据插入和执行

```
// Insert facts into the KIE session.
final Message message = new Message();
message.setMessage( "Hello World" );
message.setStatus( Message.HELLO );
ksession.insert( message );

// Fire the rules.
ksession.fireAllRules();
```

规则执行使用数据模型将数据作为输入和输出到 KieSession。本例中的数据模型有两个字段：消息，它是一个 String，其状态为 HELLO 或 GOODBYE。

数据模型类

```
public static class Message {
    public static final int HELLO   = 0;
    public static final int GOODBYE = 1;

    private String         message;
    private int            status;
    ...
}
```

这两个规则位于 src/main/resources/org/drools/helloworld/helloworld/HelloWorld.drl 文件中。

"Hello World" 规则的 when 条件指出，针对插入到 KIE 会话的每个 Message 对象激活规则，其具有 status Message.HELLO。另外，会创建两个变量绑定：变量 消息 绑定到 message 属性，变量 m 绑定到匹配的 Message 对象本身。

规则的 then 操作指定将绑定变量 消息 的内容打印到 System.out，然后更改绑定到 m 的 Message 对象 的消息 和状态属性的值。该规则使用 modify 语句应用一个语句中的分配块，并通知块末尾的更改的决策引擎。

"hello World" 规则

```
rule "Hello World"
  when
    m : Message( status == Message.HELLO, message : message )
  then
    System.out.println( message );
    modify ( m ) { message = "Goodbye cruel world",
                   status = Message.GOODBYE };
end
```

"Good Bye" 规则与 "Hello World" 规则类似，但它会与具有状态 Message.GOODBYE 的 Message 对象匹配。

"好"规则

```
rule "Good Bye"
  when
    Message( status == Message.GOODBYE, message : message )
  then
    System.out.println( message );
end
```

要执行示例，请在 IDE 中将 org.drools.examples.helloworld.HelloWorldExample 类作为 Java 应用程序运行。该规则会写入到 System.out，debug 侦听器写入 System.err，审计日志记录器 在目标/helloworld.log 中创建日志文件。

IDE 控制台中的 system.out 输出

```
Hello World
Goodbye cruel world
```

IDE 控制台中的 system.err 输出

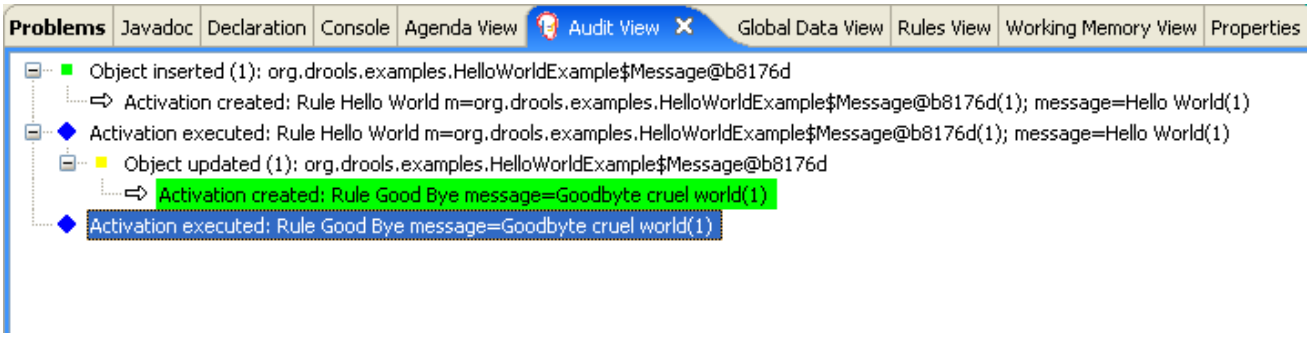
```
==>[ActivationCreated(0): rule=Hello World;
        tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectInserted: handle=
[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
        object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[BeforeActivationFired: rule=Hello World;
        tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
==>[ActivationCreated(4): rule=Good Bye;
        tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectUpdated: handle=
[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
        old_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96;
        new_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[AfterActivationFired(0): rule=Hello World]
[BeforeActivationFired: rule=Good Bye;
        tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[AfterActivationFired(4): rule=Good Bye]
```

为了更好地了解本例中的执行流，您可以将审计日志文件从 target/helloworld.log 加载到 IDE 调试视图或 Audit View （例如，如果可用）。

在本例中，audit 视图显示 对象已插入，这将为 "Hello World" 规则创建一个激活。然后执行激活，它

会更新 Message 对象并导致 "Good Bye" 规则激活。最后，执行 "Good Bye" 规则。当您在 Audit View 中选择一个事件时，origin 事件（此示例中是 "创建的" 事件）将以绿色突出显示。

图 21.3. hello World 示例 Audit 视图



21.3. 状态决策示例（转发链和冲突解析）

State 示例决策集演示了决策引擎如何使用正向链以及正在正常工作内存中的事实更改来解决按顺序规则的执行冲突。该示例着重介绍通过相同值或通过在规则中定义的日程表组解决冲突。

以下是状态示例概述：

- 名称 ：状态
- 主类 ：org.drools.examples.state.StateExampleUsingSalience,org.drools.examples.state.StateExampleUsingAgendaGroup （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.state.*.drl （在 src/main/resources）
- 目标 ：通过规则认证和日程组演示转发链和冲突解决情况

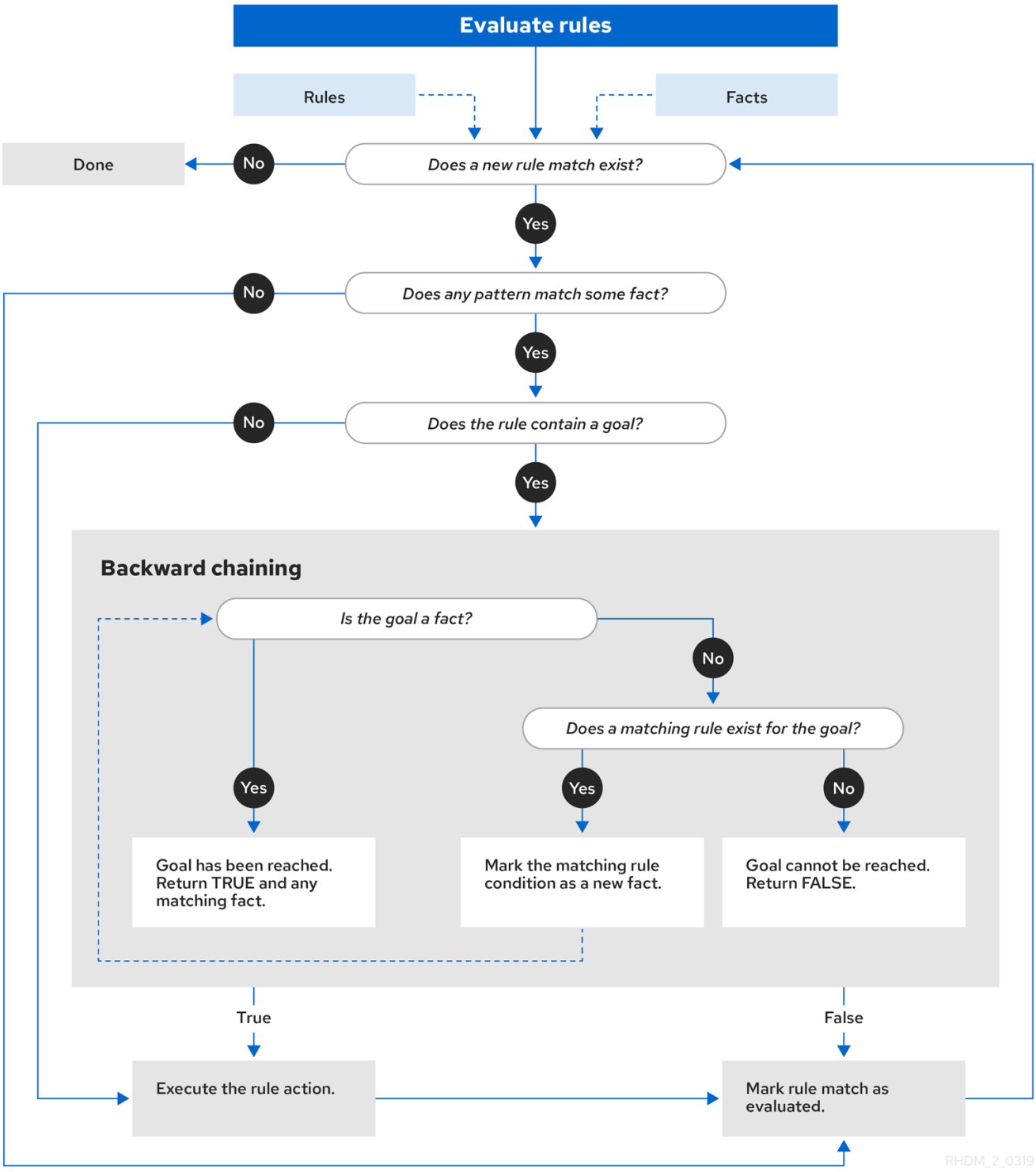
forward-chaining 规则系统是一个由数据驱动的系统，它从决策引擎的工作内存从事实开始，并对事实做出响应。当对象插入到工作内存时，因为更改是由日程表计划执行而变为 true 的任何规则条件。

相反，反向链接规则系统是一个由目标驱动的系统，从结论开始，决定引擎尝试满足，通常使用递归。如果系统无法达到结论或目标，它会搜索部分当前目标的子项。系统会继续这个过程，直到初始的结论是满足或者所有子语满意。

Red Hat Process Automation Manager 中的决策引擎使用正向和向后链来评估规则。

下图显示了如何使用转发链在逻辑流中的反向链接片段评估规则：

图 21.4. 使用转发和向后链的规则评估逻辑



在 State 示例中，每个 State 类都有一个名称及其当前状态的字段（请参阅类 org.drools.examples.state.State）。以下状态是每个对象的两个可能状态：

- NOTRUN

- 完成

State class

```
public class State {
   public static final int NOTRUN   = 0;
   public static final int FINISHED = 1;

   private final PropertyChangeSupport changes =
      new PropertyChangeSupport( this );

   private String name;
   private int    state;

   ... setters and getters go here...
}
```

State 示例包含两个版本的同一示例，用于解决规则执行冲突：

- 一个 StateExampleUsingSalience 版本，它通过使用规则 salience 解决了冲突
- 一个 StateExampleUsingAgendaGroups 版本，它通过使用规则日程表组解决冲突

两个状态示例都涉及四个状态对象：A、B、C 和 D。最初，其状态设定为 NOTRUN，这是示例使用的构造器的默认值。

使用 salience 的州示例

State 示例的 StateExampleUsingSalience 版本使用规则中的 salience 值来解决规则执行冲突。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。

示例将每个 State 实例插入到 KIE 会话中，然后调用 fireAllRules（）。

Salience 状态执行示例

```
final State a = new State( "A" );
final State b = new State( "B" );
final State c = new State( "C" );
final State d = new State( "D" );

ksession.insert( a );
ksession.insert( b );
ksession.insert( c );
ksession.insert( d );

ksession.fireAllRules();

// Dispose KIE session if stateful (not required if stateless).
ksession.dispose();
```

要执行该示例，请运行 org.drools.examples.state.StateExampleUsingSalience 类作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的 Salience State 示例输出

```
A finished
B finished
C finished
D finished
```

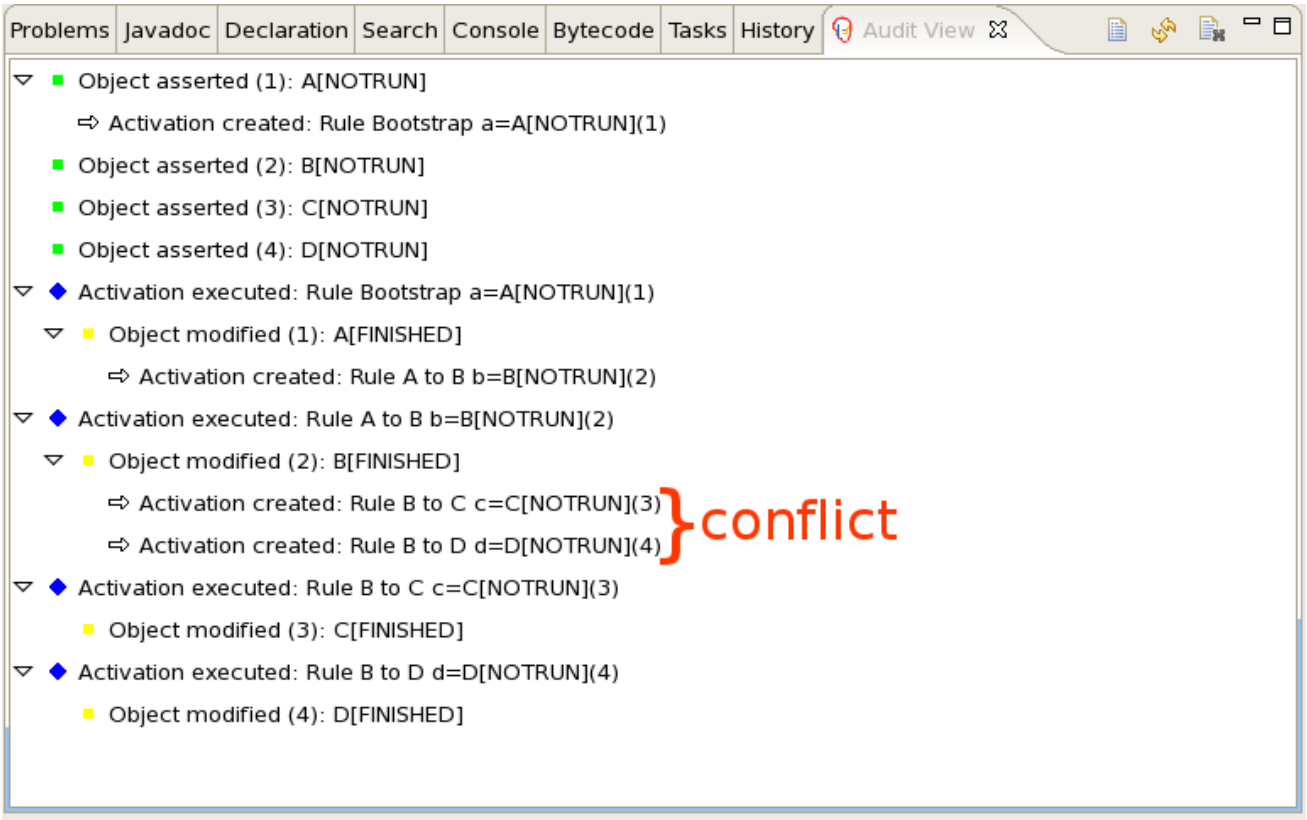
有四个规则：

首先，"Bootstrap" 规则触发，将 A 设置为状态为 FINISHED，然后使 B 将其状态更改为 FINISHED。对象 C 和 D 都依赖于 B，从而导致了值解析的冲突。

为了更好地了解本例中的执行流，您可以将审计日志文件从 target/state.log 加载到 IDE 调试视图或 Audit View （例如，如果可用）。

在本例中， *审计视图* 显**示状态中**对**象 A 的断言不会激活 "Bootstrap"** *规则*，**而其他**对**象的断言没有立即生效。**

图 ***21.5. Salience State 示例 Audit*** *视图*



salience State 示例中的*规则****"引导"示例***

```
rule "Bootstrap"
  when
    a : State(name == "A", state == State.NOTRUN )
  then
    System.out.println(a.getName() + " finished" );
    a.setState( State.FINISHED );
end
```

"Bootstrap" *规则***的执行会将 A 的状**态**更改**为 **FINISHED，后者可激活**规则 **"A 到 B "。**

salience State 示例中的*规则****"A 到 B"***

```
rule "A to B"
  when
    State(name == "A", state == State.FINISHED )
    b : State(name == "B", state == State.NOTRUN )
  then
    System.out.println(b.getName() + " finished" );
    b.setState( State.FINISHED );
end
```

规则 **"A 到 B" 的执行将 B 的状态更改为 FINISHED，后者可激活 "B 到 C" 和 "B 到 D"，将其激活置于决策引擎日程表上。**

salience State 示例中的规则"B 至 C"和"B to D"

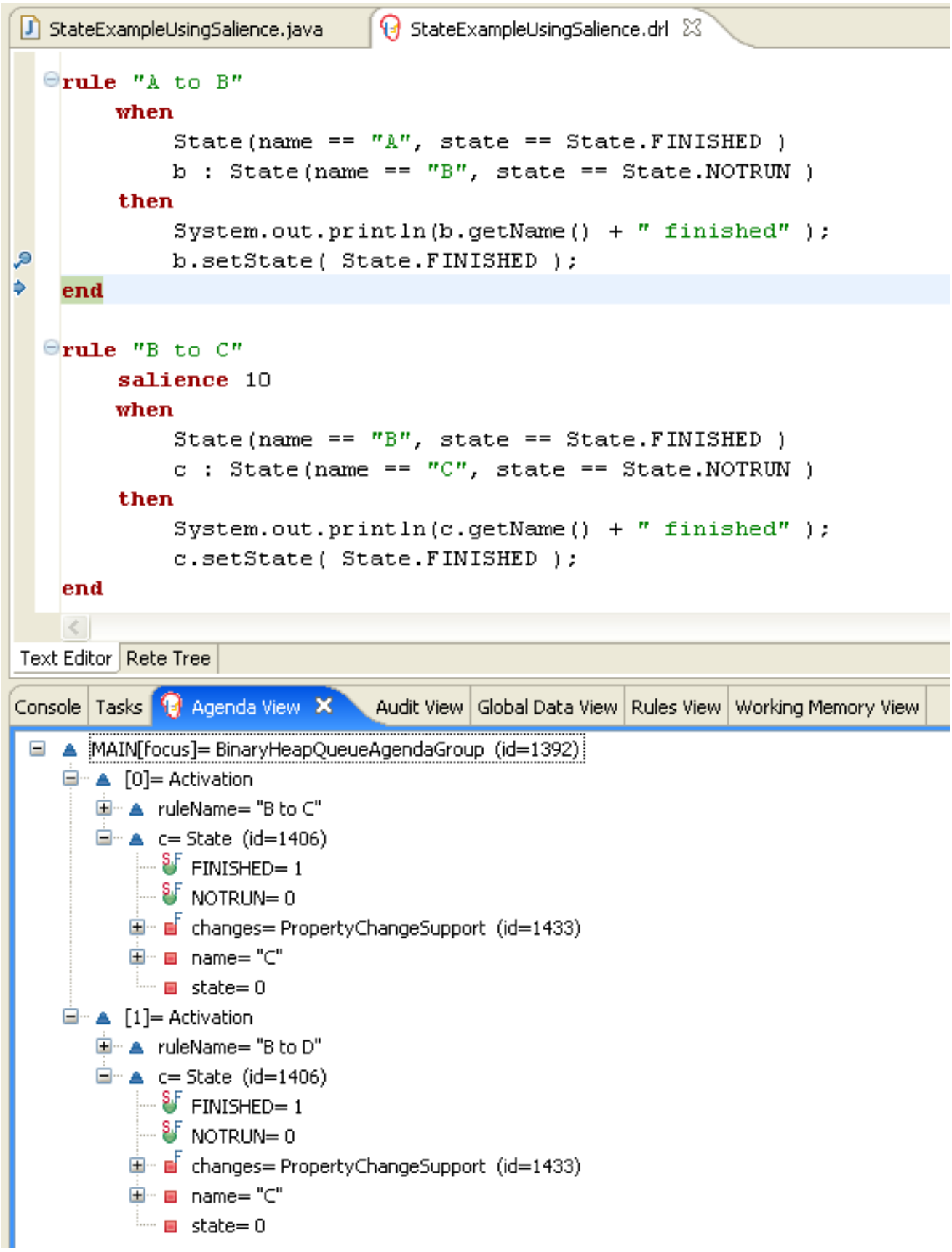
```
rule "B to C"
    salience 10
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
end

rule "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

此时，规则可能会触发，因此规则存在冲突。冲突解决策略使决策引擎日程表决定要触发的规则。规则 "B to C" 的值较高（10 与默认值 0不同），因此它首先触发对象 C，将对象 C 改为 state FINISHED。

IDE 中的 Audit 视图显示对规则 "A 到 B" 中的 State 对象的修改，这会导致两个激活冲突。

您还可以使用 IDE 中的 Agenda View 来调查决策引擎日程表的状态。在本例中，Agen da View 显示规则 "A 到 B" 中的断点，以及带有两个冲突的规则的日程表状态。规则 "B to D" 最后触发，将对象 D 修改为状态为 FINISHED。

图 21.6. Salience State example Agenda View



使用日程组进行状态示例

State 示例的 State 示例中的 StateExampleUsingAgendaGroups 版本使用 Rules 中的 table 组来解决规则执行冲突。日程表组使您可以对决策引擎日程表进行分区，以便对规则组提供更多执行控制。默认情况下，所有规则均位于 MAIN 日程小组。您可以使用 schedule -group 属性来指定该规则的不同日程表组。

最初，工作内存专注于 MAIN 的日程安排。仅当该组收到相关事项时，即可参与日程表组中的规则。您可以使用方法 setFocus（） 或 rule 属性 auto-focus 来设置焦点。auto-focus 属性允许当规则匹配和激活时，自动为课程安排人员自动给定规则。

在本例中，auto-focus 属性可让规则 "B to C" 在 "B to D" 前触发。

会议小组示例中的规则"B to C"

```
rule "B to C"
    agenda-group "B to C"
    auto-focus true
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "B to D" ).setFocus();
end
```

在 Registration group "B to C" 中的规则 "B to C" 调用 setFocus（），使其活动规则可以触发，然后启用规则 "B to D "。

会议小组示例中的规则"B to D"

```
rule "B to D"
    agenda-group "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
```

```
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

要执行该示例，请运行 org.drools.examples.state.StateExampleUsingAgendaGroups 类作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出（与状态示例的 salience 版本相同）：

IDE 控制台中查看组状态示例输出

```
A finished
B finished
C finished
D finished
```

State 示例中的动态事实

此 State 示例中的另一个值得注意的概念是根据实施 PropertyChangeListener 对象的对象，使用动态事实。要让决策引擎查看和响应事实属性更改，应用程序必须通知发生更改的决策引擎。您可以使用 modify 语句在规则中明确配置此通信，或者通过指定事实实施 PropertyChangeSupport 接口（如 format）规范定义的 PropertyChangeSupport 接口来显式配置此通信。

本例演示了如何使用 PropertyChangeSupport 接口以避免规则中明确 修改 语句的需求。要使用此接口，请确保您的事实实施 PropertyChangeSupport，这与类 org.drools.example.State 实施的方式相同，然后在 DRL 规则文件中使用以下代码，将决策引擎配置为侦听这些事实上的属性更改：

声明动态事实

```
declare type State
  @propertyChangeSupport
end
```

使用 PropertyChangeListener 对象时，每个 setter 必须实施通知的额外代码。例如，以下 state 的设置者位于类 org.drools.examples 中：

带有 PropertyChangeSupport 的 setter 示例

```
public void setState(final int newState) {
    int oldState = this.state;
    this.state = newState;
    this.changes.firePropertyChange( "state",
                                     oldState,
                                     newState );
}
```

21.4. FIBONACCI 示例决策（接收和冲突解析）

Fibonacci 示例决策集演示了决策引擎如何使用递归来解析序列中的规则执行冲突。这个示例侧重于通过在规则中定义的同等值来解决冲突。

以下是 Fibonacci 示例概述：

- 名称：fibonacci
- 主类：org.drools.examples.fibonacci.FibonacciExample （在 src/main/java中）
- 模块：drools-examples
- 键入：Java 应用程序
- 规则文件：org.drools.examples.fibonacci.Fibonacci.drl (src/main/resources)

- 目标 ：通过规则策略检查和冲突解决

Fibonacci Numbers 形成一个以 0 和 1 开头的序列。下一个 Fibonacci 编号通过添加前两个 Fibonacci number 中获取： 0, 1, 1, 2, 3, 5, 8, 8, 13, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946 等。

Fibonacci 示例使用单一事实类 Fibonacci 和以下两个字段：

- 序列
- value

sequence 字段显示对象在 Fibonacci 数字序列中的位置。value 字段显示该序列位置的 Fibonacci 对象的值，其中 -1 表示仍然需要计算的值。

Fibonacci 类

```
public static class Fibonacci {
    private int  sequence;
    private long value;

    public Fibonacci( final int sequence ) {
        this.sequence = sequence;
        this.value = -1;
    }

    ... setters and getters go here...
}
```

要执行该示例，请运行 org.drools.examples.fibonacci.FibonacciExample 类，作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的 Fibonacci 示例输出

```
recurse for 50
recurse for 49
recurse for 48
recurse for 47
...
recurse for 5
recurse for 4
recurse for 3
recurse for 2
1 == 1
2 == 1
3 == 2
4 == 3
5 == 5
6 == 8
...
47 == 2971215073
48 == 4807526976
49 == 7778742049
50 == 12586269025
```

要在 Java 中实现此行为，示例将一个 Fibonacci 对象插入一个 Fibonacci 对象，其序列字段为 50。然后，示例使用递归规则插入其他 49 Fibonacci 对象。

本示例使用 MVEL dialect 修改 关键字来启用块 setter 操作并通知决策引擎更改，而不是实现 PropertyChangeSupport 接口使用动态事实。

Fibonacci 示例执行

```
ksession.insert( new Fibonacci( 50 ) );
ksession.fireAllRules();
```

本例使用以下三个规则：

- *"recurse"*
- *"bootstrap"*
- *"计算"*

规则 **"Recurse" 匹配每个断言的 Fibonacci 对象，其值为 -1，创建并断出新的 Fibonacci 对象，其序列比当前匹配对象小。每次添加 Fibonacci 对象时，只要存在等于 1 的 sequence 字段，则规则重新匹配并再次触发。如果没有 条件元素，当您在内存中拥有 50 Fibonacci 对象后，不使用条件元素停止匹配规则。该规则也有 salience 值，因为您需要在执行 "Bootstrap" 规则前断断所有 50 Fibonacci 对象。**

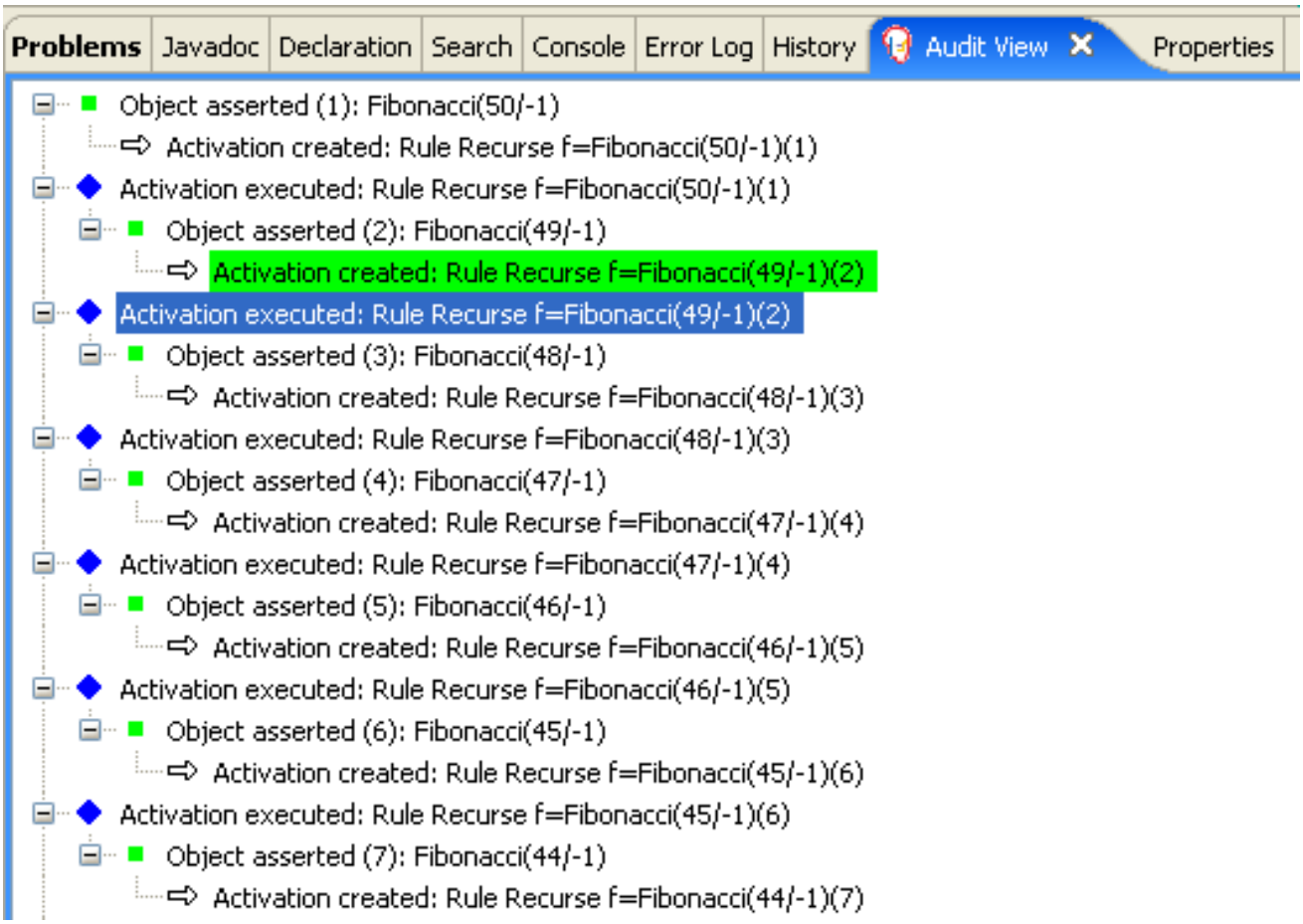
规则 **"Recurse"**

```
rule "Recurse"
    salience 10
  when
    f : Fibonacci ( value == -1 )
    not ( Fibonacci ( sequence == 1 ) )
  then
    insert( new Fibonacci( f.sequence - 1 ) );
    System.out.println( "recurse for " + f.sequence );
end
```

为**了更好地理解本例的执行流，您可以将审计日志文件从 target/fibonacci.log 加载到 IDE 调试视图或 Audit View （例如，如果可用，位于 Window → Show View in some IDE）。**

在本例中，审计 视图显示 Fibonacci 对象的原始断言，其 序列字段为 50，它通过 Java 代码完成。在那里，审计 视图显示规则的连续递归，其中每个断言的 Fibonacci 对象会导致 "Recurse" 规则变为激活并再次触发。

图 21.7. Audit 视图中的规则"重复"



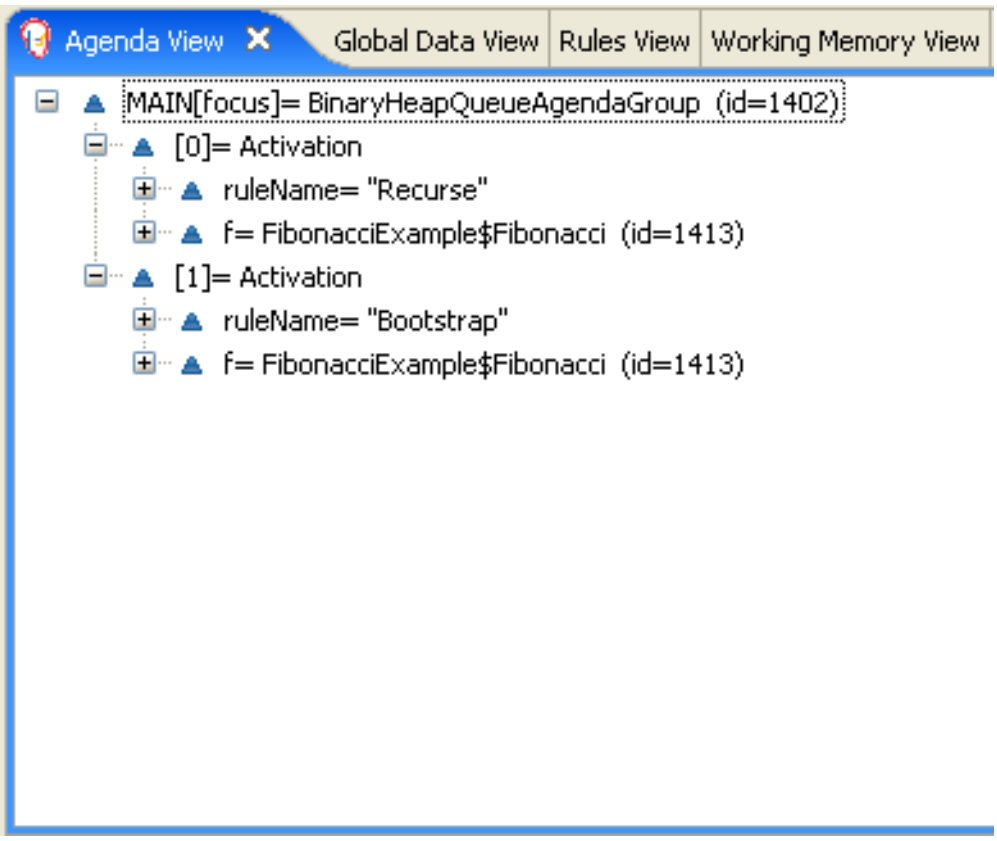
如果 Fibonacci 带有 序列字段的 Fib onacci 对象被断言，"Bootstrap" 规则会与 "Recurse" 规则一起匹配和激活。请注意，对字段 序列 的多个限制，使用 1 或 2 测试是否相等：

规则 "Bootstrap"

```
rule "Bootstrap"
  when
    f : Fibonacci( sequence == 1 || == 2, value == -1 ) // multi-restriction
  then
    modify ( f ){ value = 1 };
    System.out.println( f.sequence + " == " + f.value );
end
```

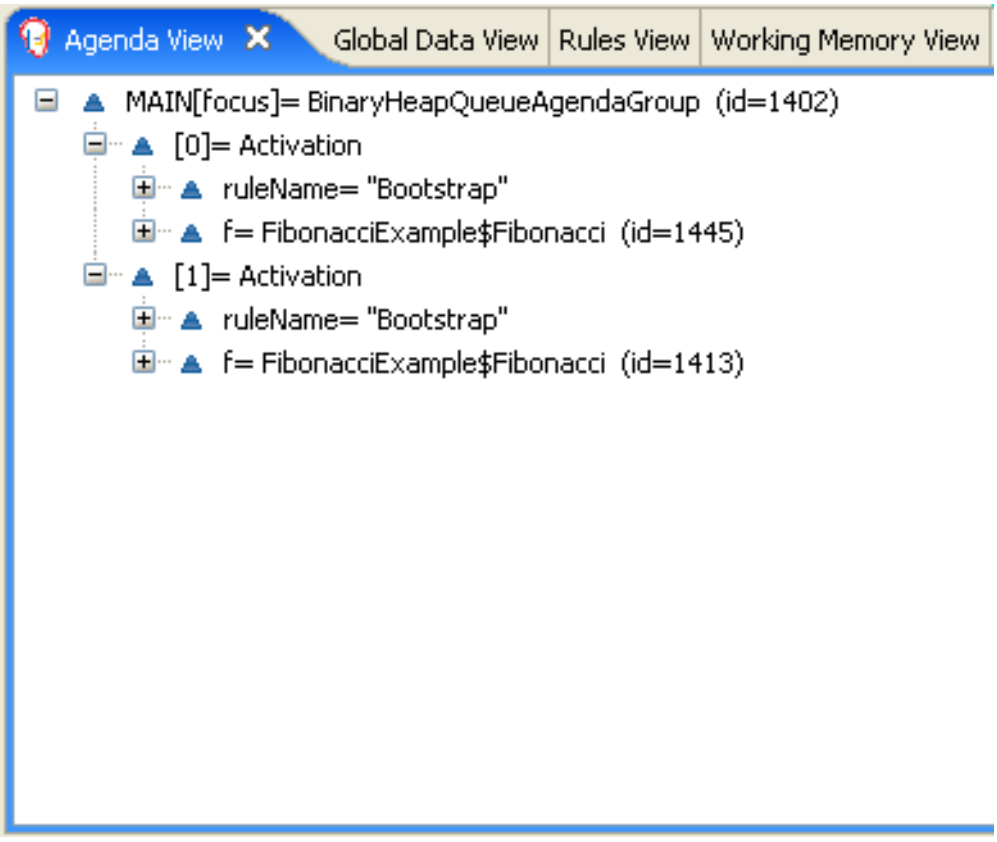
您还可以使用 IDE 中的 Agenda View 来调查决策引擎日程表的状态。"Bootstrap" 规则尚未触发，因为 "Recurse" 规则具有更高的 salience 值。

图 21.8. Agenda View 1 中的规则"Recurse"和"Bootstrap"



当断言了 序列 为 1 的 Fibonacci 对象时，会再次匹配 "Bootstrap" 规则，从而导致此规则有两个激活。"Recurse" 规则不匹配并激活，因为 not 条件元素在存在 序列 的 Fibonacci 对象时立即停止匹配规则。

图 21.9. Agenda View 2 中的规则"Recurse"和"Bootstrap"



"Bootstrap" 规则使用 序列 1 和 2 的值来设置对象。现在，有两个 Fibonacci 对象的值不等于 -1，"Calculate" 规则可以匹配。

此时，工作内存中有近 50 Fibonacci 对象。您需要依次选择适当的三角来计算其每个值。如果您在没有字段限制的规则中使用三个 Fibonacci 模式来限制可能的跨产品，则结果将是 50x49x48 个可能的组合，从而导致大约 125,000 个可能的规则触发，其中大多数不正确。

"Calculate" 规则使用字段限制来以正确顺序评估三个 Fibonacci 模式。这种技术称为与 跨产品匹配 的跨产品。

第一个模式找到任何值为 != -1 的 Fibonacci 对象，并且绑定模式和字段。第二个 Fibonacci 对象执行相同的操作，但添加了额外的字段约束，以确保其序列大于绑定到 f1 的 Fibonacci 对象。当这个规则首次触发时，您知道只有序列 1 和 2 的值为 1，并且两个限制可确保 f1 参考序列 1 和 f2 参考序列 2。

最终模式找到一个值等于 -1 且序列大于 f2 的 Fibonacci 对象。

此时，可以从可用的跨产品正确选择三个 Fibonacci 对象，您可以计算绑定到 f3 的第三个 Fibonacci 对象的值。

规则"计算"

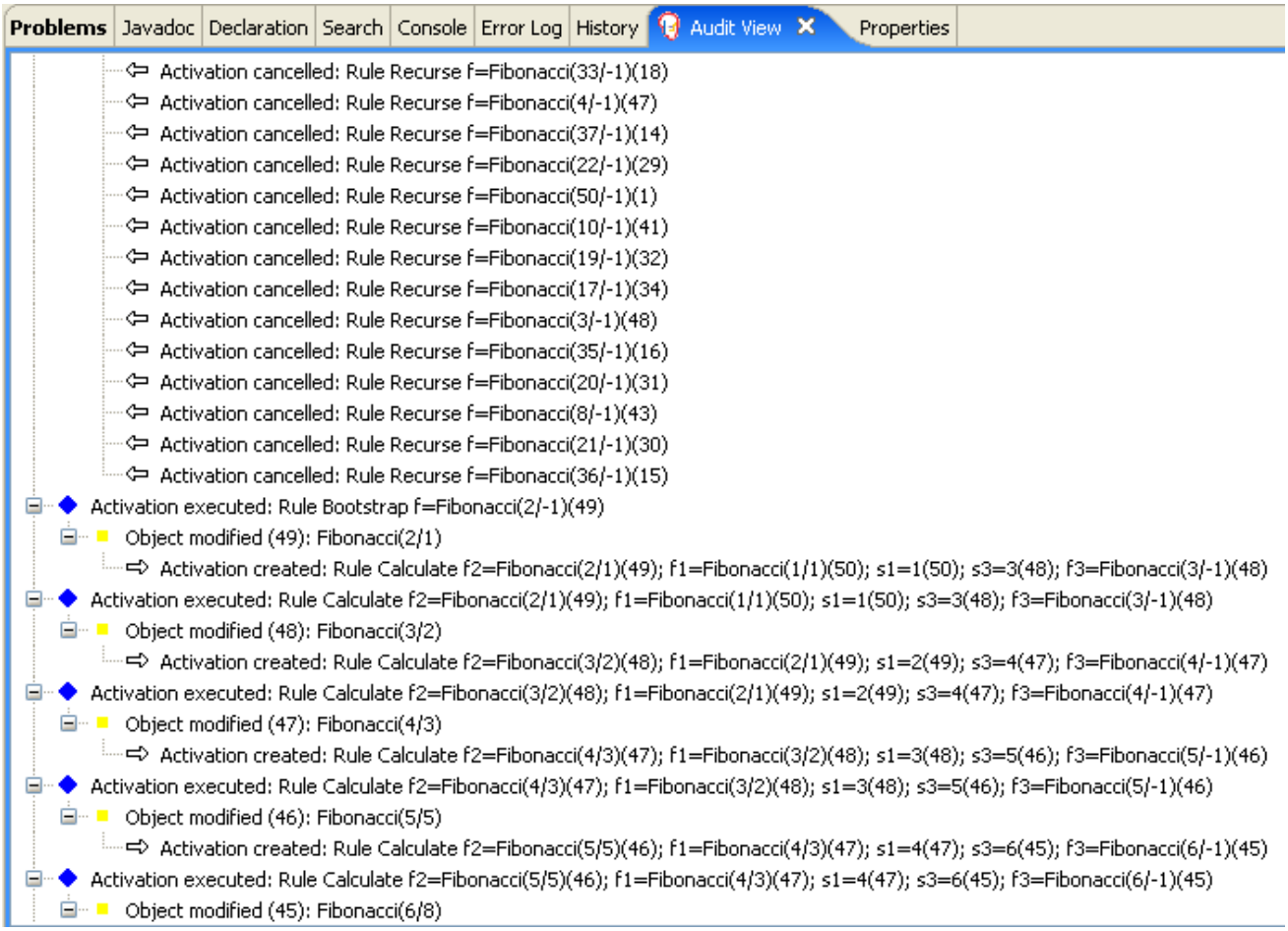
```
rule "Calculate"
  when
    // Bind f1 and s1.
    f1 : Fibonacci( s1 : sequence, value != -1 )
    // Bind f2 and v2, refer to bound variable s1.
    f2 : Fibonacci( sequence == (s1 + 1), v2 : value != -1 )
    // Bind f3 and s3, alternative reference of f2.sequence.
    f3 : Fibonacci( s3 : sequence == (f2.sequence + 1 ), value == -1 )
  then
    // Note the various referencing techniques.
    modify ( f3 ) { value = f1.value + v2 };
    System.out.println( s3 + " == " + f3.value );
end
```

modify 语句更新绑定到 f3 的 Fibonacci 对象的值。这意味着，您现在有一个没有等于 -1 的新

Fibonacci 对象，它允许 "Calculate" 规则重新匹配并计算下一个 Fibonacci 号。

IDE 的 debug 视图或 Audit View 显示触发最后 "Bootstrap" 规则如何修改 Fibonacci 对象，启用 "Calculate" 规则以匹配，然后修改另一个 Fibonacci 对象，以便重新匹配 "Calculate" 规则。此过程将继续，直到为所有 Fibonacci 对象设置了值。

图 21.10. Audit 视图中的规则



21.5. 定价示例决策（决策表）

定价示例决策集演示了如何使用电子表格决策表来以表格形式计算保险政策的零售成本，而不是直接在 DRL 文件中计算。

以下是定价示例概述：

- *名称：decisiontable*
- *主类：org.drools.examples.decisiontable.PricingRuleDTExample（在 src/main/java 中）*

- 模块：drools-examples
- 键入：Java 应用程序
- 规则文件：org.drools.examples.decisiontable.ExamplePolicyPricing.xls (src/main/resources)
- 目标：演示电子表格决策表的使用来定义规则

电子表格决策表是 XLS 或 XLSX 电子表格，其中包含以表格格式定义的业务规则。您可以包括带有独立红帽流程自动化管理器项目的电子表格决策表，或者在 Business Central 中将其上传到项目。决策表中的每一行都是一个规则，每个列都是条件、操作或其他规则属性。在创建并上传您的决定表至 Red Hat Process Automation Manager 项目中后，您定义的规则会像所有其他规则资产一样编译到 Drools 规则语言(DRL)规则中。

定价示例的目的是提供一组业务规则来计算基础价格和适用于适用于特定类型的保险政策的 car 驱动程序折扣。驱动程序的年龄和历史以及策略类型，所有为计算基本高级的贡献，其他规则计算驱动程序可能有资格获得的潜在折扣。

要执行示例，请运行 org.drools.examples.decisiontable.PricingRuleDTExample 类，作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

```
Cheapest possible
BASE PRICE IS: 120
DISCOUNT IS: 20
```

执行示例的代码遵循典型的执行模式：加载规则，并插入事实，并且创建一个无状态的 KIE 会话。本例中的区别在于，规则在 ExamplePolicyPricing.xls 文件中定义，而不是 DRL 文件或其他来源。使用模板和 DRL 规则，电子表格文件被加载到决策引擎中。

电子表格决策表设置

ExamplePolicyPricing.xls spreadsheet 在第一个标签页中包含两个 decision 表：

- ***基本定价**规则*
- ***促销折扣**规则*

***当**电子表格示例演示**时，您只能使用**电子表格中的第一个**标签页**来**创建决策表，但多个表可以在一个**标签页**内。决策表不一定遵循自**顶逻辑，但更是捕**获规则**生成的数据的一种方式。**规则**的**评估不一定按**给定**顺序使用，因**为该**决策引擎的所有普通原理仍然适用。因此，您可以在**电子表格的同一个**标签页**中有多个路由表。*

***决策表通**过**相**应**的**规则**模板文件 BasePricing.drt 和 promotionPricing .drt 执行。这些模板文件通**过其模板参数来引用决策表，并直接引用 decision 表中条件和操作的各种**标头**。*

BasePricing.drt** 规则**模板文件

```
template header
age[]
profile
priorClaims
policyType
base
reason

package org.drools.examples.decisiontable;

template "Pricing bracket"
age
policyType
base

rule "Pricing bracket_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}
        , priorClaims == "@{priorClaims}"
        , locationRiskProfile == "@{profile}"
    )
    policy: Policy(type == "@{policyType}")
  then
    policy.setBasePrice(@{base});
    System.out.println("@{reason}");
end
end template
```

促销.drt 规则模板文件

```
template header
age[]
priorClaims
policyType
discount

package org.drools.examples.decisiontable;

template "discounts"
age
priorClaims
policyType
discount

rule "Discounts_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}, priorClaims == "@{priorClaims}")
    policy: Policy(type == "@{policyType}")
  then
    policy.applyDiscount(@{discount});
end
end template
```

规则是通过 KIE Session DTableWithTemplateKB 的 kmodule.xml 参考执行，它特别提到了 ExamplePolicyPricing.xls spreadsheet，并且需要成功执行规则。这个执行方法可让您将规则作为独立单元（如本例中）执行，或者将规则包括在打包的 JAR(KJAR)文件中，以便电子表格与要执行的规则一起打包。

要成功执行规则和电子表格，需要 kmodule.xml 文件的以下部分：

```
  <kbase name="DecisionTableKB" packages="org.drools.examples.decisiontable">
    <ksession name="DecisionTableKS" type="stateless"/>
  </kbase>

  <kbase name="DTableWithTemplateKB" packages="org.drools.examples.decisiontable-template">
    <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
                  template="org/drools/examples/decisiontable-template/BasePricing.drt"
                  row="3" col="3"/>
    <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
                  template="org/drools/examples/decisiontable-template/PromotionalPricing.drt"
```

```
                row="18" col="3"/>
        <ksession name="DTableWithTemplateKS"/>
    </kbase>
```

作为使用规则模板文件执行决策表的替代方法，您可以使用 DecisionTableConfiguration 对象，并将输入电子表格指定为输入类型，如 DecisionTableInputType.xls ：

```
DecisionTableConfiguration dtableconfiguration =
    KnowledgeBuilderFactory.newDecisionTableConfiguration();
        dtableconfiguration.setInputType( DecisionTableInputType.XLS );

        KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

        Resource xlsRes = ResourceFactory.newClassPathResource(
"ExamplePolicyPricing.xls",
                                                getClass() );
        kbuilder.add( xlsRes,
                ResourceType.DTABLE,
                dtableconfiguration );
```

定价示例使用两种事实类型：

- 驱动
- 策略.

这个示例在相应的 Java 类 Driver.java 和 Policy.java 中同时设置了默认值。Driver 旧是 30 年，没有以前的声明，目前存在 LOW 的风险配置集。为应用该驱动程序的 Policy 是 COMPREHENSIVE。

在任何决策表中，每行都被视为不同的规则，每个列都是条件或一个操作。每行都在决定表中评估，除非在执行时清除表格。

决策表电子表格（XLS 或 XLSX）需要两个定义规则数据的关键区域：

- RuleSet 区域
- 规则 区域

电子表格的 RuleSet 区域定义了您要对同一软件包中的所有规则（不仅仅是电子表格）全局应用的元素，如规则集名称或通用规则属性。RuleTable 区域定义了实际规则（箭头）以及条件、操作和其他规则属性（列），这些属性构成指定规则集中的规则表。表格电子表格可以包含多个可 规则 的区域，但只能包含一个 RuleSet 区域。

图 21.11. 决策表配置

	C	D	E	F	G	H
	RuleSet	org.drools.examples.decisiontable				
	Notes	This decision table is for working out some basic prices and pretending actuaries don't exist				
	RuleTable Pricing bracket					
	CONDITION	CONDITION	CONDITION	CONDITION	ACTION	ACTION
	Driver			policy: Policy		
	age >= $1, age <= $2	locationRiskProfile	priorClaims	type	policy.setBasePrice($param);	System.out.println("$param");
	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason

RuleTable 区域还定义了规则属性应用到的对象，在本例中为 Driver 和 Policy，后面接对象的限制。例如，定义 Age Bracket 列的 Driver 对象约束为 age >= $1, age <= $2，其中以逗号分隔的范围在表列中值中定义，如 18,24。

基本定价规则

定价示例中的基本价格规则 表评估驱动程序的年龄、风险配置集、声明次数和策略类型，并根据这些条件生成策略的基本价格。

图 21.12. 基本价格计算

	B	C	D	E	F	G	H
9	Base pricing rules	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason
10	Young safe package	18, 24	LOW	1	COMPREHENSIVE	450	
11			MED		FIRE_THEFT	200	Priors not relevant
12			MED	0	COMPREHENSIVE	300	
13			LOW		FIRE_THEFT	150	
14			LOW	0	COMPREHENSIVE	150	Safe driver discount
15	Young risk	18,24	MED	1	COMPREHENSIVE	700	
16		18,24	HIGH	0	COMPREHENSIVE	700	Location risk
17		18,24	HIGH		FIRE_THEFT	550	Location risk
18	Mature drivers	25,30		0	COMPREHENSIVE	120	Cheapest possible
19		25,30		1	COMPREHENSIVE	300	
20		25,30		2	COMPREHENSIVE	590	
21		25,35		3	THIRD_PARTY	800	High risk

Driver 属性在下表中定义：

- age Bracket ： age bracket 为条件 年龄 >=$1, age <=$2 的定义，它定义了驱动程序年龄的条件界限。此条件列突出显示了 $1 和 $2 的使用，在电子表格中用逗号分隔。您可以将这些值写为 18、24 或 18，24 或 18，这两种格式都可以执行业务规则。

- 位置风险配置集: risk 配置集是一个字符串，该示例程序始终以 LOW 形式传递，但可以改为反映 MED 或 HIGH 。

- 之前声明 的数量：声明的数量定义为一个整数，条件列必须完全等于触发操作。该值不是范围，仅完全匹配。

决策表的 Policy 可在条件和规则操作和规则操作中使用，并在下表列中定义属性：

- 策略类型适用于 ：策略类型是以字符串形式传递的条件： COMPREHENSIVE、FIRE_THEFT 或 THIRD_PARTY。

- Base $ AUD: basePrice 作为 ACTION，它通过约束 策略.setBasePrice($param)设置价格；基于与这个值对应的电子表格单元。当您为这个决定表执行对应的 DRL 规则时，规则的 然后 部分针对与事实匹配的 true 条件执行这个 action 语句，并将基本价格设置为对应的值。

- Record Reason ：当规则成功执行时，此操作会在 System.out 控制台上生成一个输出信息，它反映了哪些规则触发。之后会在应用程序中捕获并打印。

这个示例也使用左侧的第一列来分类规则。此列仅用于注释，对规则执行没有影响。

促销折扣规则

定价 示例中的促销折扣规则 表格评估了驱动程序的年龄、之前声明和策略类型，以便根据保险政策的价格生成可能的折扣。

图 21.13. 折扣计算

29	Promotional discount rules	Age Bracket	Number of prior claims	Policy type applying for	Discount %
30	Rewards for safe drivers	18,24	0	COMPREHENSIVE	1
31		18,24	0	FIRE_THEFT	2
32		25,30	1	COMPREHENSIVE	5
33		25,30	2	COMPREHENSIVE	1
34		25,30	0	COMPREHENSIVE	20

此决定表包含驱动程序可能有资格享受的折扣条件。与基本价格计算类似，此表评估了驱动程序之前声明的期限、驱动程序之前的声明数量，以及用于确定要应用的 Discount % 速率的策略 类型。例如，如果驱动程序是过去 30 年，没有之前的声明，并且会申请 COMPREHENSIVE 策略，则驱动程序将享有 20% 的折扣。

21.6. PET STORE 示例决策（示例组、全局变量、回调和 GUI 集成）

Pet Store 示例决策集演示了如何在规则中使用平板组和全局变量，以及如何将 Red Hat Process Automation Manager 规则与图形用户界面(GUI)集成，在这种情况下，基于 Swing 的桌面应用程序。这个示例还演示了如何使用回调与正在运行的决策引擎交互，以在运行时根据工作内存中的更改更新 GUI。

以下是 Pet Store 示例的概述：

- 名称 ：petstore
- 主类 ：org.drools.examples.petstore.PetStoreExample （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.petstore.PetStore.drl (src/main/resources)
- 目标 ：演示规则索引组、全局变量、回调和 GUI 集成

在 Pet Store 示例中，示例 PetStoreExample.java 类定义了以下主体类（除了多个类来处理 Swing 事件外）：

- Petstore 包含 main（） 方法。
- PetStoreUI 负责创建和显示基于 Swing 的 GUI。此类包含多个较小的类，主要用于响应各种 GUI 事件，比如鼠标点击的用户。
- TableModel 包含表数据。这个类本质上是扩展 Swing 类 AbstractTableModel 的 JavaBean。
- CheckoutCallback 允许 GUI 与规则交互。
- Ordershow 保留您要购买的项目。
- 购买 存储订购详情以及您要购买的产品。
- 产品是 JavaBean，其中包含可供购买的产品及其价格的详细信息。

本例中的大部分 Java 代码是基于普通 JavaBean 或 Swing。有关 Swing 组件的更多信息，请参阅有关 使用 JFC/Swing 创建 GUI 的 Java 教程。

Pet Store 示例中的规则执行行为

与其他示例决定设置被断言并立即触发，Pet Store 示例不会执行规则，直到根据用户交互收集更多事实。这个示例通过构造器创建的 PetStoreUI 对象来执行规则，接受 Vector 对象 库存 来收集该产品。然后，示例使用 Checkout Callback 类的实例，其中包含之前载入的规则基础。

pet Store KIE 容器和事实执行设置

```
// KieServices is the factory for all KIE services.
KieServices ks = KieServices.Factory.get();

// Create a KIE container on the class path.
KieContainer kc = ks.getKieClasspathContainer();
```

```
// Create the stock.
Vector<Product> stock = new Vector<Product>();
stock.add( new Product( "Gold Fish", 5 ) );
stock.add( new Product( "Fish Tank", 25 ) );
stock.add( new Product( "Fish Food", 2 ) );

// A callback is responsible for populating the working memory and for firing all rules.
PetStoreUI ui = new PetStoreUI( stock,
                                new CheckoutCallback( kc ) );
ui.createAndShowGUI();
```

触发规则的 Java 代码位于 CheckoutCallBack.checkout（） 方法中。当用户在 UI 中点击 Checkout 时触发 此方法。

来自 CheckoutCallBack.checkout（）的规则执行

```
public String checkout(JFrame frame, List<Product> items) {
    Order order = new Order();

    // Iterate through list and add to cart.
    for ( Product p: items ) {
        order.addItem( new Purchase( order, p ) );
    }

    // Add the JFrame to the ApplicationData to allow for user interaction.

    // From the KIE container, a KIE session is created based on
    // its definition and configuration in the META-INF/kmodule.xml file.
    KieSession ksession = kcontainer.newKieSession("PetStoreKS");

    ksession.setGlobal( "frame", frame );
    ksession.setGlobal( "textArea", this.output );

    ksession.insert( new Product( "Gold Fish", 5 ) );
    ksession.insert( new Product( "Fish Tank", 25 ) );
    ksession.insert( new Product( "Fish Food", 2 ) );

    ksession.insert( new Product( "Fish Food Sample", 0 ) );

    ksession.insert( order );

    // Execute rules.
    ksession.fireAllRules();

    // Return the state of the cart
    return order.toString();
}
```

示例代码将两个元素传递给 CheckoutCallBack.checkout（） 方法。一个元素是处理 JFrame Swing 组件，位于 GUI 的底部。第二个元素是顺序项目的列表，它来自 GUI 右上角的表格区域的信息。

for 循环将来自于 GUI 的订购项列表转换为 Order JavaBean，同时包含在文件 PetStoreExample.java 中。

在这种情况下，规则会在无状态 KIE 会话中触发，因为所有数据都存储在 Swing 组件中，并在用户点击 UI 中的 Checkout 之前执行。每次用户点击 Checkout 时，列表的内容都会从 Swing TableModel 移到 KIE 会话工作内存中，然后使用 ksession.fireAllRules（） 方法执行。

在此代码中，对 KieSession 有 9 个调用。其中之一从 KieContainer 创建新的 KieSession （在这个 KieContainer 中通过的示例从 main（） 方法中的 CheckoutCallBack 类传递）。接下来的两个调用通过规则中存放全局变量的两个对象： Swing 文本区域以及用于编写消息的 Swing 帧。并将有关产品的信息插入 KieSession 以及顺序列表中的更多信息。最终的调用是标准 fireAllRules（）。

pet Store 规则文件导入、全局变量和 Java 功能

PetStore.drl 文件包含标准软件包和导入语句，以使规则可以使用各种 Java 类。规则文件还包括用于在规则（定义为 帧 和 textArea ）中使用的 全局变量。全局变量包含对之前由称为 setGlobal（） 方法的 Java 代码传递的 Swing 组件 JFrame 和 JTextArea 组件的引用。与规则中的标准变量不同，在规则触发后，全局变量会保留其在 KIE 会话生命周期中的值。这意味着这些全局变量的内容可以对所有后续规则进行评估。

PetStore.drl 软件包、导入和全局变量

```
package org.drools.examples;

import org.kie.api.runtime.KieRuntime;
import org.drools.examples.petstore.PetStoreExample.Order;
import org.drools.examples.petstore.PetStoreExample.Purchase;
import org.drools.examples.petstore.PetStoreExample.Product;
import java.util.ArrayList;
import javax.swing.JOptionPane;

import javax.swing.JFrame;

global JFrame frame
global javax.swing.JTextArea textArea
```

PetStore.drl 文件还包含两个使用中的规则：

PetStore.drl Java 功能

```
function void doCheckout(JFrame frame, KieRuntime krt) {
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                             "Would you like to checkout?",
                                             "",
                                             JOptionPane.YES_NO_OPTION,
                                             JOptionPane.QUESTION_MESSAGE,
                                             null,
                                             options,
                                             options[0]);

       if (n == 0) {
           krt.getAgenda().getAgendaGroup( "checkout" ).setFocus();
       }
}

function boolean requireTank(JFrame frame, KieRuntime krt, Order order, Product fishTank,
int total) {
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                             "Would you like to buy a tank for your " + total + " fish?",
                                             "Purchase Suggestion",
                                             JOptionPane.YES_NO_OPTION,
                                             JOptionPane.QUESTION_MESSAGE,
                                             null,
                                             options,
                                             options[0]);

       System.out.print( "SUGGESTION: Would you like to buy a tank for your "
                         + total + " fish? - " );

       if (n == 0) {
            Purchase purchase = new Purchase( order, fishTank );
            krt.insert( purchase );
            order.addItem( purchase );
            System.out.println( "Yes" );
       } else {
           System.out.println( "No" );
```

```
    }
    return true;
}
```

这两个功能执行以下操作：

- doCheckout（） 显示一个对话框，它要求用户是否被委派或需要签出。如果用户确实如此，则重点设置为 结账 员组，使该组中的规则启用（可能）触发。
- requireTank（） 会显示一个对话框，该对话框要求用户如果她或想要购买财务语。如果用户确实有，则会在工作内存中的订购列表中添加一个新芬兰的 tank 产品。

注意

在本例中，所有规则和功能都位于同一个规则文件中，以提高效率。在生产环境中，您通常将不同文件中的规则和功能分开，或构建静态 Java 方法并使用导入功能导入文件，如导入功能 my.package.name.hello。

pet Store 规则与日程组

Pet Store 示例中的大多数规则使用 table 组来控制规则执行。日程表组允许您对决策引擎日程表进行分区，以便对规则组提供更多执行控制。默认情况下，所有规则均位于 MAIN 日程小组。您可以使用 schedule -group 属性来指定该规则的不同日程表组。

最初，工作内存专注于 MAIN 的日程安排。仅当该组收到相关事项时，即可参与日程表组中的规则。您可以使用方法 setFocus（） 或 rule 属性 auto-focus 来设置焦点。auto-focus 属性允许当规则匹配和激活时，自动为课程安排人员自动给定规则。

Pet Store 示例对规则使用以下日程组：

- "init"
- "评估"

- "显示项目"
- "checkout"

例如，示例规则 "Explode Cart" 使用 "init" 资格将 cart 项目触发并插入到 KIE 会话工作内存中：

规则"Explode Cart"

```
// Insert each item in the shopping cart into the working memory.
rule "Explode Cart"
    agenda-group "init"
    auto-focus true
    salience 10
  when
    $order : Order( grossTotal == -1 )
    $item : Purchase() from $order.items
  then
    insert( $item );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "show items" ).setFocus();
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "evaluate" ).setFocus();
end
```

该规则与所有尚未计算的订单匹配。每个购买项目的执行循环按该顺序排列。

该规则使用与其 schedule 组相关的以下功能：

- 日程表组"init" 定义日程表组的名称。在这种情况下，组中只有一个规则。但是，Java 代码和规则都无法专注于此组，因此它也取决于其触发的机会的 auto-focus 属性。
- auto-focus true 可确保此规则，而作为 schedule group 中的唯一规则，但从 Java 代码调用 fireAllRules（） 时有机会触发。
- kcontext....setFocus（） 将焦点设置为 "show items" 和 "evaluate" 日程表组，支持他们触发的规则。在实践中，您要循环顺序的所有项目，将它们插入到内存中，然后在各个插入后触

发其他规则。

"显示项目" 日程表组仅包含一条规则，"项目方式"对于当前 KIE 会话工作内存顺序的每个购买，规则会根据规则文件中定义的文本区域将详情记录到 GUI 底部的文本区域。

规则"显示方式"

```
rule "Show Items"
    agenda-group "show items"
  when
    $order : Order()
    $p : Purchase( order == $order )
  then
   textArea.append( $p.product + "\n");
end
```

"评估" 日程表组还从 "Explode Cart" 规则获得。此日程表组包含两个规则 ："Free Fish Food Sample" 和 "Suggest Tank"，按该顺序执行。

规则"Free Fish Food Sample"

```
// Free fish food sample when users buy a goldfish if they did not already buy
// fish food and do not already have a fish food sample.
rule "Free Fish Food Sample"
    agenda-group "evaluate" (1)
  when
    $order : Order()
    not ( $p : Product( name == "Fish Food") && Purchase( product == $p ) ) (2)
    not ( $p : Product( name == "Fish Food Sample") && Purchase( product == $p ) ) (3)
    exists ( $p : Product( name == "Gold Fish") && Purchase( product == $p ) ) (4)
    $fishFoodSample : Product( name == "Fish Food Sample" );
  then
    System.out.println( "Adding free Fish Food Sample to cart" );
    purchase = new Purchase($order, $fishFoodSample);
    insert( purchase );
    $order.addItem( purchase );
end
```

只有在以下条件都满足时，才会触发 "Free Fish Food Sample" 规则：

1. 日程组 "评估" 在规则执行中评估。
2. 用户还没有 fish food。
3. 用户还没有一个自由的页式示例。
4. 用户按数字顺序使用 goldfish。

如果顺序事实满足所有这些要求，则创建一个新产品(Fish Food Sample)并添加到工作内存中的顺序中。

规则"Suggest Tank"

```
// Suggest a fish tank if users buy more than five goldfish and
// do not already have a tank.
rule "Suggest Tank"
    agenda-group "evaluate"
  when
    $order : Order()
    not ( $p : Product( name == "Fish Tank") && Purchase( product == $p ) ) 1
    ArrayList( $total : size > 5 ) from collect( Purchase( product.name == "Gold Fish" ) ) 2
    $fishTank : Product( name == "Fish Tank" )
  then
    requireTank(frame, kcontext.getKieRuntime(), $order, $fishTank, $total);
end
```

只有在以下条件满足时才触发 "Suggest Tank" 规则：

1

用户没有按顺序的ish tank。

2

用户按以下顺序排列，有五种种。

当规则触发时，它会调用规则 文件中定义的 requireTank（） 函数。此函数会显示一个对话框，询问用户是否是她或他想要购买玻里程表。如果用户确实有，则会在工作内存中的订购列表中添加一个新芬兰的 tank 产品。当规则调用 requireTank（） 函数时，该规则会传递帧的全局变量，以便该函数能够处理 Swing GUI。

Pet Store 示例中的 "do checkout" 规则没有日程表组，没有条件，因此该规则始终执行并被视为默认 MAIN 更新组的一部分。

规则 "do checkout"

```
rule "do checkout"
  when
  then
    doCheckout(frame, kcontext.getKieRuntime());
end
```

当规则触发时，它会调用规则 文件中定义的 doCheckout（） 函数。此函数显示一个对话框，询问用户是否她或他想要签出。如果用户确实如此，则重点设置为 结账 员组，使该组中的规则启用（可能）触发。当规则调用 doCheckout（） 函数时，该规则会传递 帧 全局变量，以便该函数能够处理 Swing GUI。

注意

本例还演示了故障排除技术（如果结果未按预期一样执行）：您可以从规则的 when 语句中删除条件，并在 then 语句中测试操作以验证操作是否正确执行。

"检查" 日程表组包含三个规则，用于处理订单并申请任何折扣："总额"、"应用 5% 的折扣" 和 "应用 10% 的折扣 "。

规则"总额"、"Apply 5% 的折扣"和"Apply 10% 的折扣"

```
rule "Gross Total"
    agenda-group "checkout"
  when
    $order : Order( grossTotal == -1)
    Number( total : doubleValue ) from accumulate( Purchase( $price : product.price ),
                                                   sum( $price ) )
  then
    modify( $order ) { grossTotal = total }
    textArea.append( "\ngross total=" + total + "\n" );
end

rule "Apply 5% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 10 && < 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.95;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end

rule "Apply 10% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.90;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end
```

如果用户还没有计算总额的单位，Grossing Total accate the product 总计（将总值放在 KIE 会话中），并使用 textArea global 变量通过 Swing JTextArea 显示它。

如果总额介于 10 到 20 左右，则 "Apply 5% discount" 规则计算总折扣，将其添加到 KIE 会话中，并将其显示在文本区域中。

如果总总额不低于 20，"Apply 10% discount" 规则计算总额，将其添加到 KIE 会话中，并在文本区域中显示它。

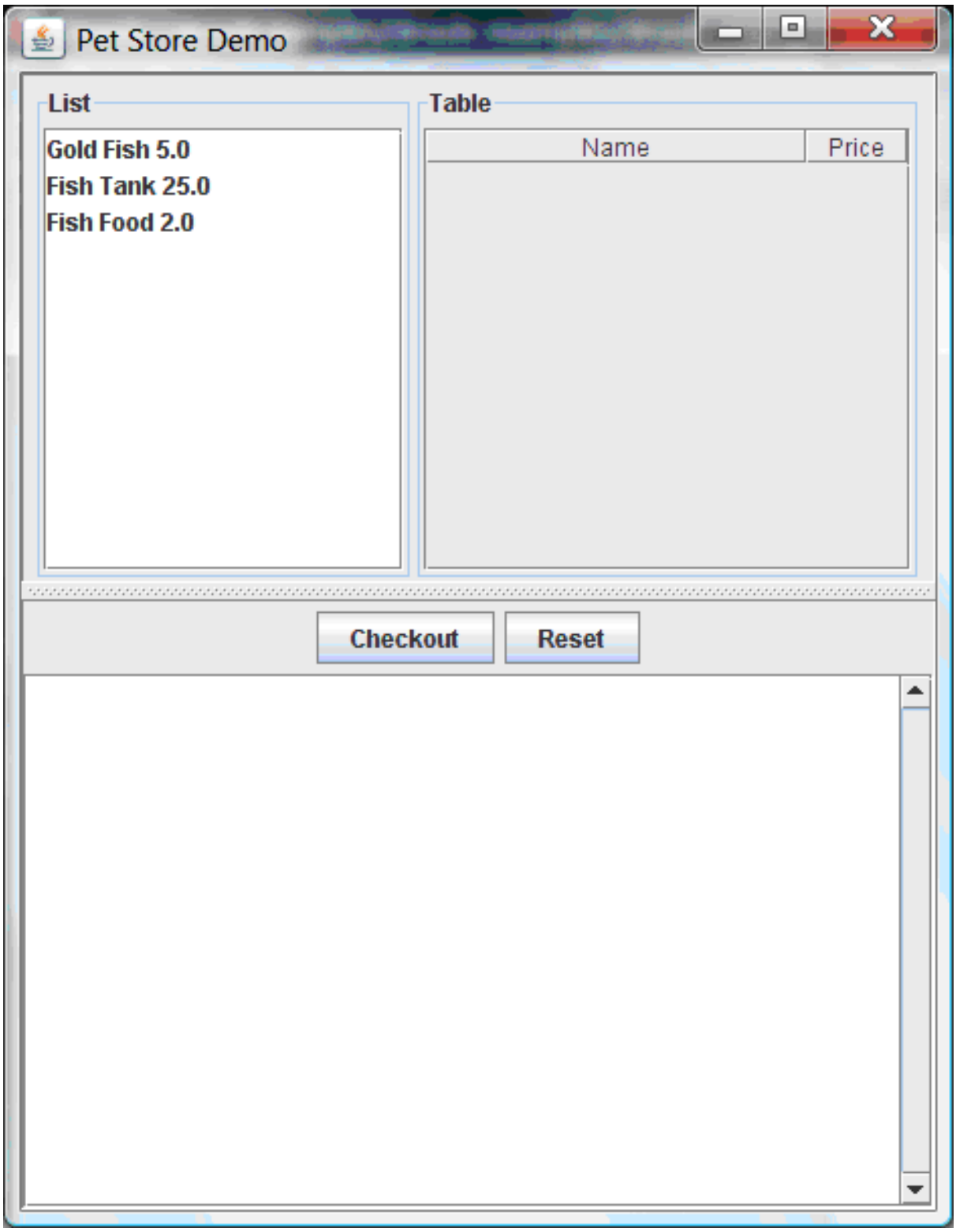
pet Store 的执行示例

与其他 Red Hat Process Automation Manager 决策示例类似，您可以通过运行

org.drools.examples.petstore.PetStoreExle 类作为 IDE 中的 Java 应用程序来执行 Pet Store 示例。

当您执行 Pet Store 示例时，会显示 Pet Store Demo GUI 窗口。此窗口显示可用产品列表（左下）、选定产品的空列表（右、Checkout 和 Reset 按钮(middle)）和一个空系统消息区域(bottom)。

图 21.14. 启动后 pet Store 示例 GUI



本例中发生以下事件来建立此执行行为：

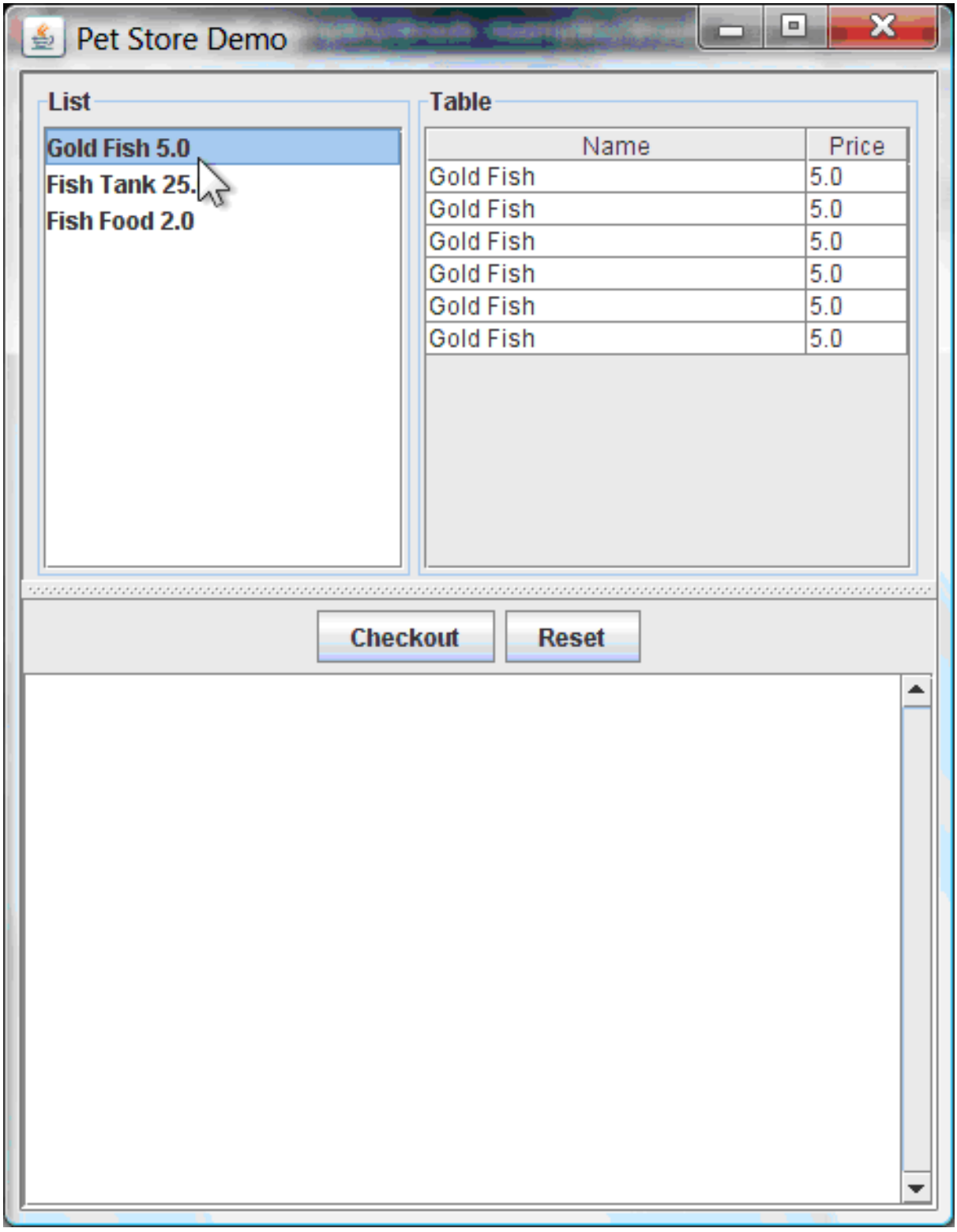
1. main（） 方法已运行并加载规则基础，但尚未触发规则。目前，这是与运行的规则连接中的唯一代码。

2. 新的 PetStoreUI 对象已创建，并为规则基础赋予句柄，供以后使用。

3. 各种 Swing 组件已执行其功能，并显示初始 UI 屏幕并等待用户输入。

您可以从列表中点各种产品以浏览 UI 设置：

图 21.15. 探索 Pet Store 示例 GUI



还没有触发规则代码。UI 使用 Swing 代码来检测用户鼠标点击并将所选产品添加到用于 UI 右上角显示的 TableModel 对象中。本例演示了 Model-View-Controller 设计模式。

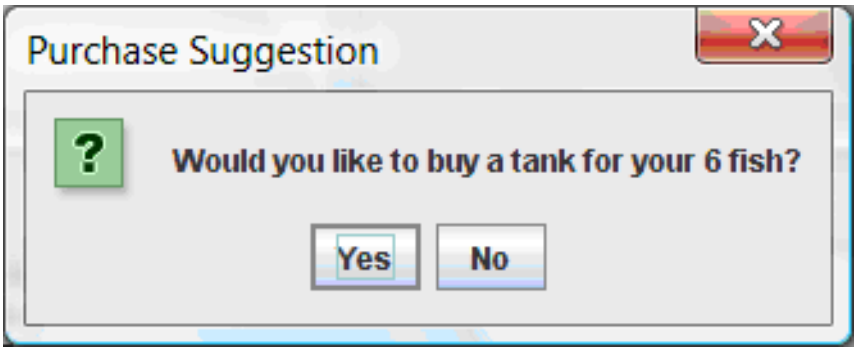
当您点 Checkout 时，规则会按以下方式触发：

1. *CheckOutCallBack.checkout（） 方法由 Swing 类调用（事件），等待用户点击 Checkout。这会将 TableModel 对象（UI 右上角）中的数据插入到 KIE 会话工作内存中。然*

后，该方法会触发规则。

2. "Explode Cart" 规则是第一个触发，auto-focus 属性设为 true。通过 cart 中的所有产品的规则循环，确保产品处于工作内存中，然后提供 "显示项目"和" 评估" 日程表组来触发的选项。这些组中的规则将 cart 的内容添加到文本区域(bottom)中，评估您是否有资格获得免费的放大分，并确定是否希望购买欺诈。

图 21.16. 芬兰 tank 资格



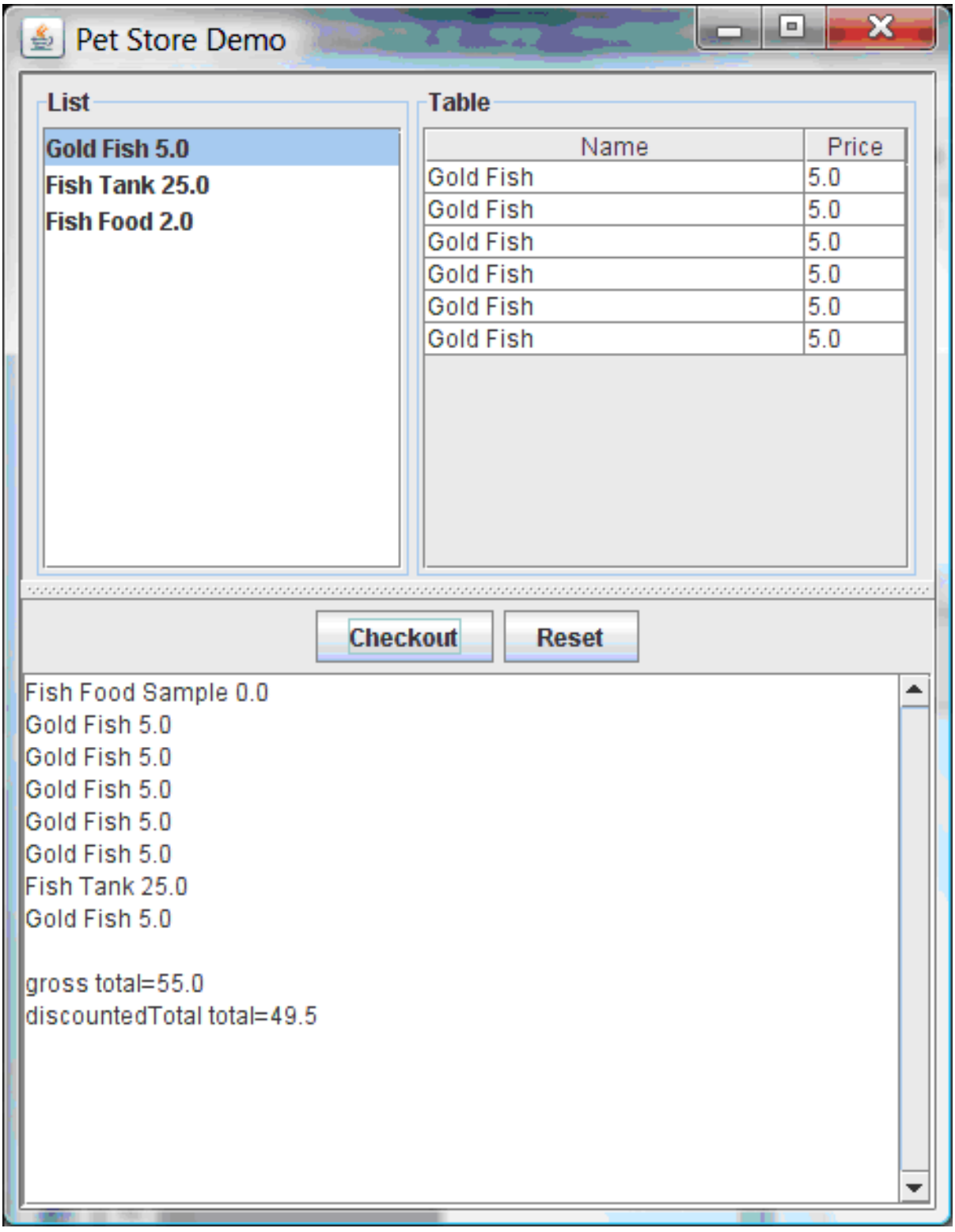
3. "do checkout" 规则是下一个需要触发的，因为其他任何日程组当前没有关注，因为它是默认的 MAIN 日程小组的一部分。这个规则始终调用 doCheckout（） 函数，它会询问您是否要签出。

4. doCheckout（） 函数将焦点设置为 "checkout" Table 组，该组中的规则给出了要触发的选项。

5. "检查" 日程表组中的规则显示车队的内容，并应用相应的折扣。

6. 然后，等待用户输入可选择更多产品（并导致规则再次触发），或关闭 UI。

*图 **21.17. 所有规则触发后的 pet Store 示例 GUI***



您可以添加更多 System.out 调用来在 IDE 控制台中演示此事件流：

IDE 控制台中的 system.out 输出

```
Adding free Fish Food Sample to cart
SUGGESTION: Would you like to buy a tank for your 6 fish? - Yes
```

21.7. HONEST POLITICIAN 示例决策（维护与健保）

Honest Politician 示例决策集展示了在规则中采用逻辑插入和使用 salience 的真实概念。

以下是 Honest Politician 示例概述：

- Name : honest Hician
- 主类:org.drools.examples.honest Hian.HonestPoliticianExample （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.honest Hian.HonestPolitician.drl (src/main/resources)
- 目标 ：根据事实的逻辑插入以及规则中的使用方法论证了真实维护的概念

Honest Politician 示例的基本内部是对象只能存在，而声明是 true。规则结果可以逻辑地插入带有 insertLogical（） 方法的对象。这意味着，在 KIE 会话工作内存中，只要逻辑插入的规则保持在 KIE 会话工作内存中。当规则不再为 true 时，对象会被自动重试。

在本例中，规则执行会导致一组自治亚更改，以防出现损坏企业的结果。评估完各个文所评估时，这些属性从 honesty 属性设置为 true，但有一个规则触发，使自治亚体不再是 honest。当它们的状态从 honest 改为 dishonest 时，它们会从工作内存中删除。该规则可通知决策引擎如何优先考虑为它们定义的任何规则，否则使用默认 salience 值 0。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。

Politician 和 Hope 类

示例中的示例类 Politician 为 honest Leaician 进行了配置。Politician 类由 String 项 名称和 一个布尔值项目组成：

Politician 类

```
public class Politician {
    private String name;
    private boolean honest;
    ...
}
```

Hope 类确定是否存在 Hope 对象。这个类没有有意义的成员，但只要 society 希望就出现在正常工作的内存中。

希望课程

```
public class Hope {

    public Hope() {

    }
  }
```

shardician honesty 的规则定义

在 Honest Politician 示例中，当工作内存中至少存在一个 honest shardician 时，"We have an honest Politician" 规则逻辑地插入一个新的 Hope 对象。当所有协调员都变得不开时，Hope 对象会被自动重新处理。此规则具有值为 10 的 salience 属性，以确保它在任何其他规则之前触发，因为 在该阶段 "Hope is Dead" 规则为 true。

规则"我们有一个最哈佛尼亚"

```
rule "We have an honest Politician"
    salience 10
```

```
  when
    exists( Politician( honest == true ) )
  then
    insertLogical( new Hope() );
end
```

当 Hope 对象存在时，"Hope Lives" 规则与 and fires 匹配和触发。这个规则也具有 10 相似的值，以便它的优先级高于 Honest" 规则。

规则**"停止实时"**

```
rule "Hope Lives"
    salience 10
  when
    exists( Hope() )
  then
    System.out.println("Hurrah!!! Democracy Lives");
end
```

最初，存在四个最哈尔地，因此此规则有 4 个激活，它们发生冲突。每个规则依次触发，破坏每个工作员，从而使它们不再频繁。当所有四个协调器都已损坏时，没有 Hoticians 含有属性 honest == true。规则 "We have an hest Politician" 不再是 true，它的逻辑插入的对象（由于 是新 Hope（）的最后执行）会自动重新处理。

规则**"利用 Honest"**

```
rule "Corrupt the Honest"
  when
    politician : Politician( honest == true )
    exists( Hope() )
  then
    System.out.println( "I'm an evil corporation and I have corrupted " + politician.getName() );
    modify ( politician ) { honest = false };
end
```

当 Hope 对象自动经由真相维护系统时，没有应用到 Hope 的条件元素不再为 true，"Hope is Dead" 规则匹配并触发。

规则"Hope is Dead"

```
rule "Hope is Dead"
  when
    not( Hope() )
  then
    System.out.println( "We are all Doomed!!! Democracy is Dead" );
end
```

执行和审核跟踪示例

在 HonestPoliticianExample.java 类中，四个协调员并将 Honest state 设置为 true 以针对定义的业务规则插入评估：

HonestPoliticianExample.java 类执行

```
public static void execute( KieContainer kc ) {
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        ksession.fireAllRules();

        ksession.dispose();
    }
```

要执行这个示例，请运行 org.drools.examples.honest Hician.HonestPoliticianExample 类，作为

IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的执行输出

```
Hurrah!!! Democracy Lives
I'm an evil corporation and I have corrupted President of Umpa Lumpa
I'm an evil corporation and I have corrupted Prime Minster of Cheeseland
I'm an evil corporation and I have corrupted Tsar of Pringapopaloo
I'm an evil corporation and I have corrupted Omnipotence Om
We are all Doomed!!! Democracy is Dead
```

输出显示，虽然至少有一个最哈尔、演示文稿。但是，由于每个人都受到一些公司损坏，所有自治亚人都变得不满，因此演示不容忽视。

为了更好地理解本例的执行流，您可以修改 HonestPoliticianExample.java 类，使其包含 DebugRuleRuntimeEventListener 侦听器以及审计日志记录器来查看执行详情：

HonestPoliticianExample.java 类，带有审计日志记录器

```
package org.drools.examples.honestpolitician;

import org.kie.api.KieServices;
import org.kie.api.event.rule.DebugAgendaEventListener; 1
import org.kie.api.event.rule.DebugRuleRuntimeEventListener;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public class HonestPoliticianExample {

  /**
   * @param args
   */
  public static void main(final String[] args) {
   KieServices ks = KieServices.Factory.get(); 2
   //ks = KieServices.Factory.get();
     KieContainer kc = KieServices.Factory.get().getKieClasspathContainer();
     System.out.println(kc.verify().getMessages().toString());
```

```
        //execute( kc );
        execute( ks, kc); 3
    }

    public static void execute( KieServices ks, KieContainer kc ) { 4
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        // The application can also setup listeners 5
        ksession.addEventListener( new DebugAgendaEventListener() );
        ksession.addEventListener( new DebugRuleRuntimeEventListener() );

        // Set up a file-based audit logger.
        ks.getLoggers().newFileLogger( ksession, "./target/honestpolitician" ); 6

        ksession.fireAllRules();

        ksession.dispose();
    }

}
```

1

向导入处理 DebugAgendaEventListener 和 DebugRuleRuntimeEventListener的软件包添加

2

创建一个 KieServices Factory 和 ks 元素来生成日志，因为此审计日志无法在 KieContainer 级别提供

3

修改 执行 方法以使用 KieServices 和 KieContainer

4

除了 KieContainer外，修改 执行 方法以传递 KieServices

5

创建监听程序

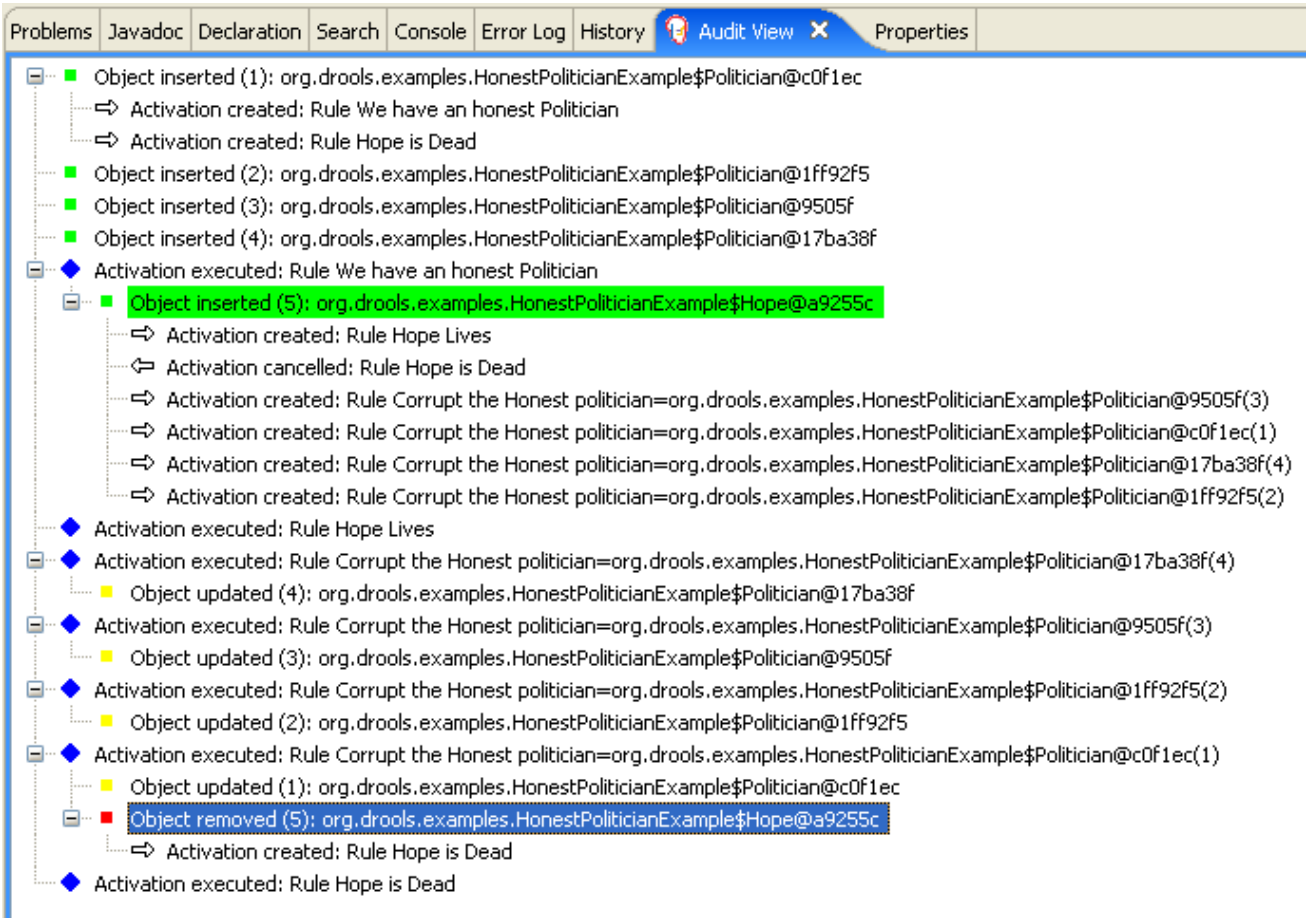
6

在执行规则后构建可传递给 debug 视图或 Audit View 或 IDE 的日志

当使用这个修改的日志功能运行 Honest Politician 时，您可以将审计日志文件从 target/honest Hician.log 加载到 IDE 调试视图或 Audit View （例如，在一些 IDE 中的 Window → Show View 中）。

在本例中，审计视图显示执行流、插入和重新操作，如示例类和规则中定义：

图 21.18. Honest Politician 示例审计视图



当插入第一个协调程序时，会发生两个激活。规则 "We have honest Politician" 只激活第一个插入了 Handomician 的时间，因为它使用 exists 条件元素，即在插入至少一个自治器时匹配。规则 "Hope is Dead" 在此阶段也被激活，因为 Hope 对象尚未插入。规则 "我们首先触发了 honest Politician"，因为它具有高于规则 "Hope is Dead" 的值，并插入 Hope 对象（以绿色高亮显示）。Hope 对象插入对象可激活规则 "Hope Lives"，并停用规则 "Hope is Dead "。insertion 还激活每个插入了 honest APPician 的规则 "Corrupt the Honest"。执行规则 "Hope Lives" 并打印 "Hurrah!!Democracy Lives"。

接下来，对于每个人来说，规则 "Corrupt the Honest" 触发，打印 "I't an evil 企业和 I have corrupted X"，其中 X 是 Hotician 的名称，并将 otician honesty 值修改为 false。当最后的 honest 此目录被破坏时，Hope 将自动由真相维护系统（以蓝色突出显示）进行重新处理。绿色突出显示的区域显示了当前所选蓝色突出显示区域的来源。更改了 Hope 事实后，规则 "Hope is dead" 触发后，打印 "We 都是 Doomed!!Democracy 是 Dead"。

21.8. SUDOKU 示例决策（COMPLEX PATTERN MATCHING、回调和 GUI 集成）

Sudoku 示例决策基于流行数字 puzzle Sudoku 的示例决策，演示了如何在 Red Hat Process Automation Manager 中使用规则，以根据各种限制在大型潜在解决方案空间中找到解决方案。这个示例还演示了如何将 Red Hat Process Automation Manager 规则集成到图形用户界面(GUI)中，在这种情况下，基于 Swing 的桌面应用程序，以及如何使用回调与正在运行的决策引擎进行交互，以在运行时根据工作内存更改更新 GUI。

以下是 Sudoku 示例的概述：

- 名称 ：sudoku
- 主类 ：org.drools.examples.sudoku.SudokuExample （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.sudoku.*.drl （在 src/main/resources）
- 目标 ：演示复杂的模式匹配、问题解决、回调和 GUI 集成

Sudoku 是基于逻辑的数字放置 puzzle。目标是填充 9x9 网格，以便每个列、每个列、每个行以及九个 3x3 区域均包含来自 1 到 9 次的数字。puzzle 设定者提供部分完成的网格，而 puzzle solver 的任务是利用这些限制完成网格。

解决问题的一般策略是，当您插入新数字时，它必须在其特定 3x3 区域、行和列内唯一。这个 Sudoku 示例决策设置使用红帽流程自动化管理器规则来解决 Sudoku puzzles 的问题，并试图解决包含

无效条目的缺陷的 puzzles。

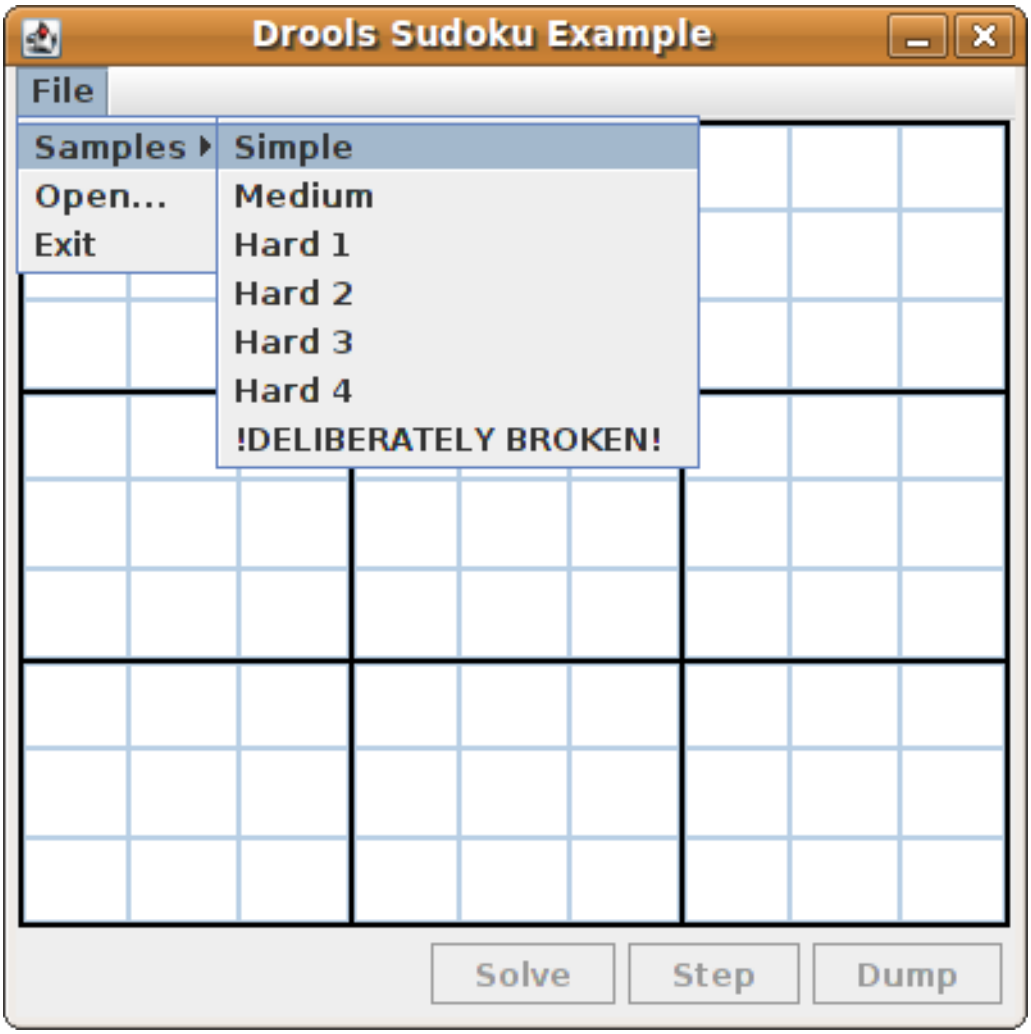
Sudoku 示例执行和交互

与其他红帽流程自动化管理器决策示例类似，您可以通过运行 org.drools.examples.sudoku.SudokuExample 类作为 IDE 中的 Java 应用程序来执行 SudokuExample 类。

当您执行 Sudoku 示例时，会出现 Drools Sudoku Example GUI 窗口。此窗口包含空网格，但该程序随附了存储于内部的各种网格，您可以加载和解决。

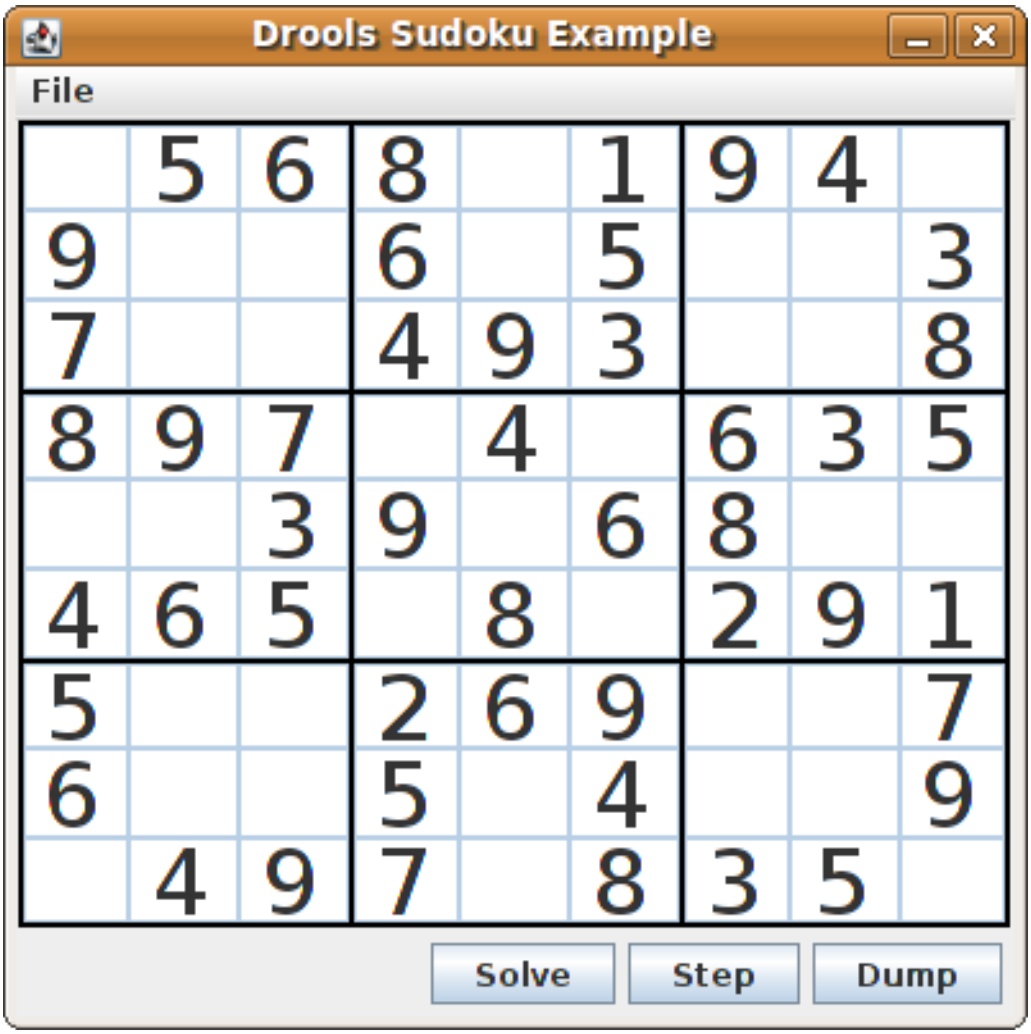
点 File → Samples → Simple 加载其中一个示例。请注意，在加载网格前，所有按钮都被禁用。

图 21.19. 启动时的 Sudoku 示例 GUI



加载 简单示例 时，网格会根据点的初始状态进行填写。

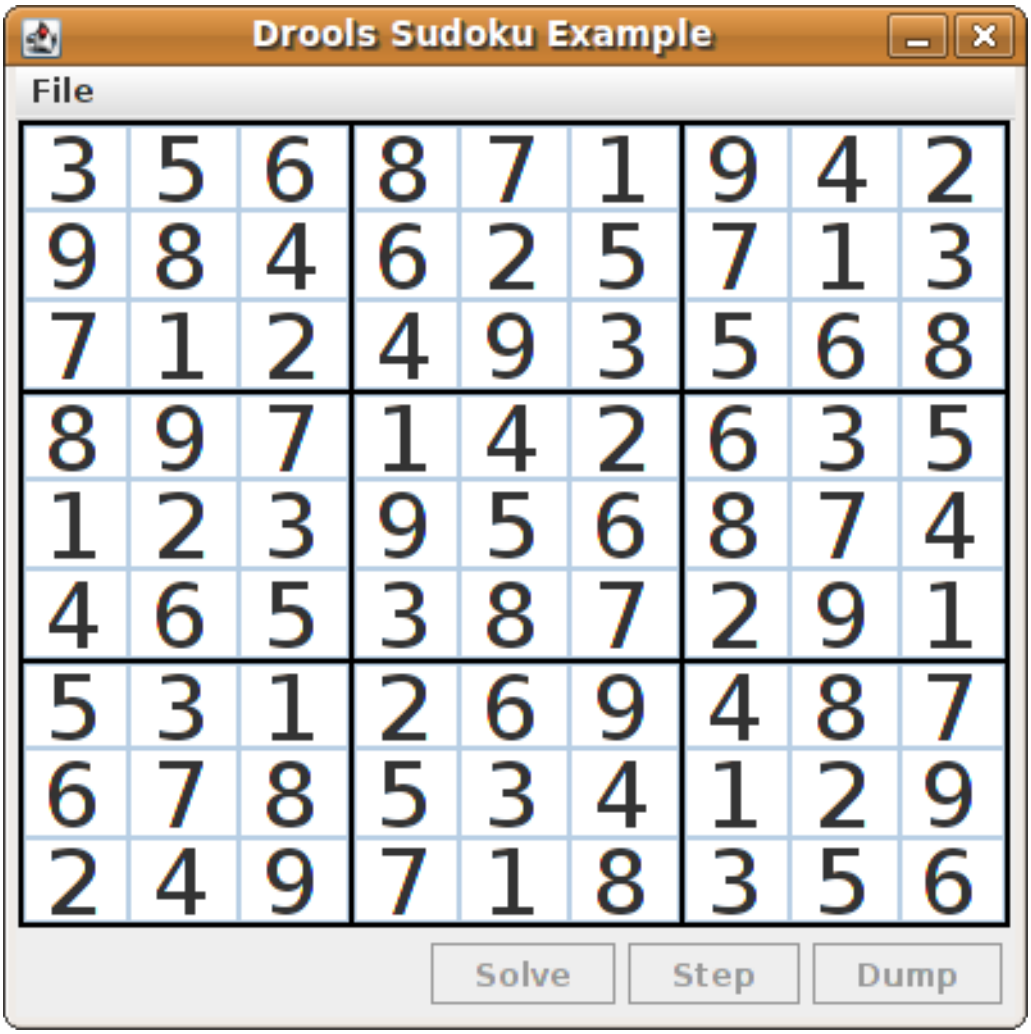
图 21.20. 加载简单示例后的 Sudoku 示例 GUI



从以下选项中选择：

- 点 Solve 触发 Sudoku 示例中定义的规则，该示例中填充剩余的值，并且使按钮再次不活跃。

图 21.21. 已解决简单示例



- 点 Step 查看规则集找到的下一个数字。IDE 中的控制台窗口显示从中解决问题的规则的详细信息。

IDE 控制台中的步骤执行输出

```
single 8 at [0,1]
column elimination due to [1,2]: remove 9 from [4,2]
hidden single 9 at [1,2]
row elimination due to [2,8]: remove 7 from [2,4]
remove 6 from [3,8] due to naked pair at [3,2] and [3,7]
hidden pair in row at [4,6] and [4,4]
```

- 点击 Dump 查看网格的状态，其中单元显示了既定的值或剩余的可能性。

在 IDE 控制台中转储执行输出

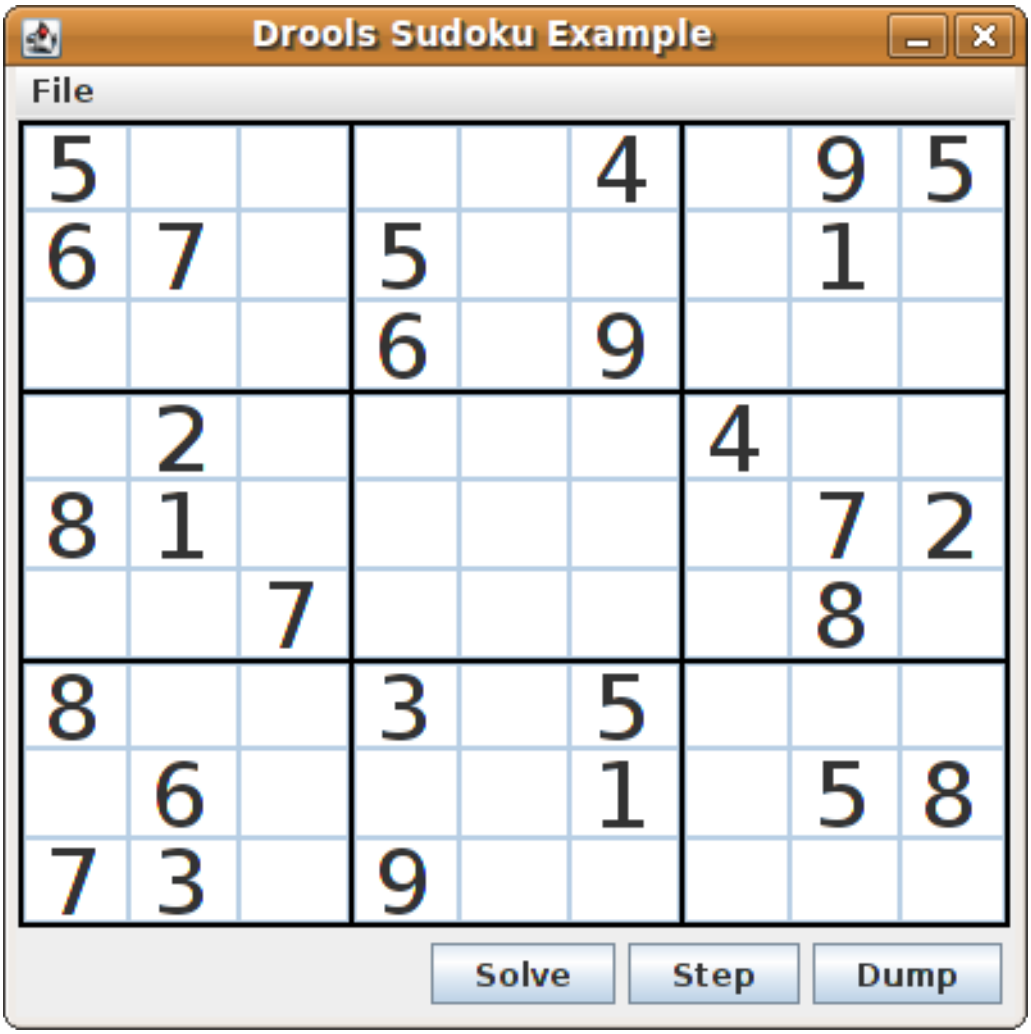
在 IDE 控制台中打印执行输出

```
      Col: 0     Col: 1     Col: 2     Col: 3     Col: 4     Col: 5     Col: 6     Col: 7     Col: 8
Row 0:  123456789  --- 5 ---  --- 6 ---  --- 8 ---  123456789  --- 1 ---  --- 9 ---  --- 4 ---
123456789
Row 1:  --- 9 ---  123456789  123456789  --- 6 ---  123456789  --- 5 ---  123456789
123456789  --- 3 ---
Row 2:  --- 7 ---  123456789  123456789  --- 4 ---  --- 9 ---  --- 3 ---  123456789  123456789
--- 8 ---
Row 3:  --- 8 ---  --- 9 ---  --- 7 ---  123456789  --- 4 ---  123456789  --- 6 ---  --- 3 ---  --- 5 ---
Row 4:  123456789  123456789  --- 3 ---  --- 9 ---  123456789  --- 6 ---  --- 8 ---  123456789
123456789
Row 5:  --- 4 ---  --- 6 ---  --- 5 ---  123456789  --- 8 ---  123456789  --- 2 ---  --- 9 ---  --- 1 ---
Row 6:  --- 5 ---  123456789  123456789  --- 2 ---  --- 6 ---  --- 9 ---  123456789  123456789
--- 7 ---
Row 7:  --- 6 ---  123456789  123456789  --- 5 ---  123456789  --- 4 ---  123456789
123456789  --- 9 ---
Row 8:  123456789  --- 4 ---  --- 9 ---  --- 7 ---  123456789  --- 8 ---  --- 3 ---  --- 5 ---
123456789
```

Sudoku 示例包含一个有意破坏示例文件，示例中定义的规则可以解析。

点 File → Samples → !DELIBERATELY BROKEN! 加载损坏的示例。网格从一些问题开始，例如，该值 5 显示在第一行中不允许的两次。

图 **21.22. broken Sudoku 示例初始状**态



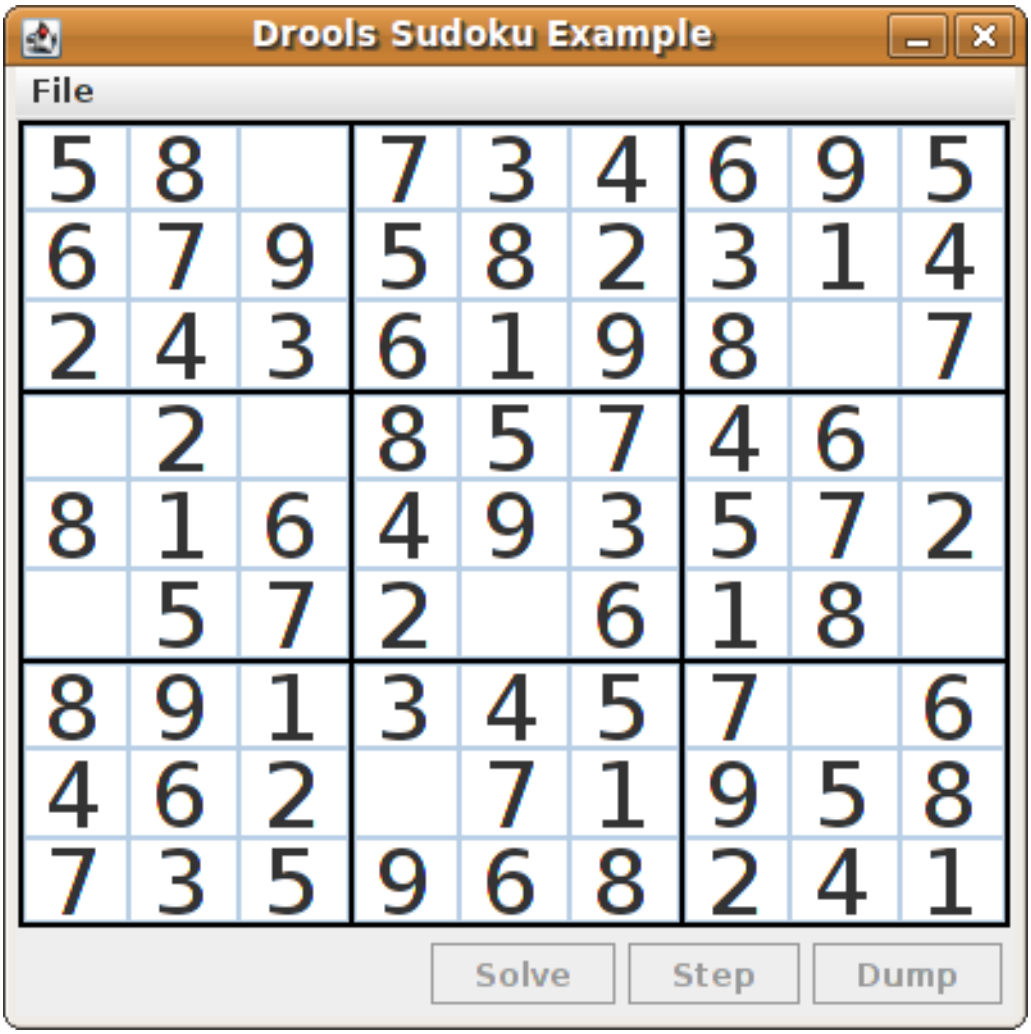
单击 **Solve，将解决**规则应**用到此无效网格。Sudoku 示例中的关**联**解决**问题检测样**本中的**问题，**并尽可能**尝试**解决**这**个**问题。这**个**过**程没有完成，并使一些**单**元留空。**

IDE 控制台窗口中会显示 **solve** 规则**活**动：

检测**到无法正常工作的**问题

```
cell [0,8]: 5 has a duplicate in row 0
cell [0,0]: 5 has a duplicate in row 0
cell [6,0]: 8 has a duplicate in col 0
cell [4,0]: 8 has a duplicate in col 0
Validation complete.
```

图 21.23. 有问题的解决方案示例尝试



名为 Hard 的 Sudoku 示例文件更为复杂，解决规则可能无法解决它们。IDE 控制台窗口中会显示不成功的解决方案：

未解析的硬示例

```
Validation complete.
...
Sorry - can't solve this grid.
```

根据仍是单元的候选值集，用于解决中断示例实施标准解析技术的规则。例如，如果一个集合包含单个值，则这是单元的值。对于九个单元里有一个值出现的一个值，规则插入了某些特定单元的"使用解决方案值"类型 设置 的事实。这一事实导致从该单元所属的任何组中所有其他单元格中消除这个值，且值将被重新处理。

示例中的其他规则会减少某些单元的可分值。规则 "naked pair", "hidden pair in row", "hidden pair in column", 和 "hidden pair in square" 消除了可能性，但没有建立解决方案。规则 "位于 行中的 X-wing in rows", "'X-wing in columns'", "interection removal line", and "intersection removal 列" 来执行更复杂的清除。

Sudoku 示例类

org.drools.examples.sudoku.swing 包含了为 Sudoku puzzles 实施框架的以下核心组件：

- SudokuGridModel 类定义了一个接口，它将一个接口存储一个 Sudoku puzzle，作为 Cell 对象的 9x9 网格。
- SudokuGridView 类是一个 Swing 组件，它可以视觉化 SudokuGridModel 类的任何实现。
- SudokuGridEvent 和 SudokuGridListener 类在模型和视图之间沟通状态变化。当一个单元值被解析或更改时，将触发事件。
- SudokuGridSamples 类为演示目的提供了部分填充的 Sudoku 模糊。

注意

这个软件包对 Red Hat Process Automation Manager 库没有依赖软件包。

package org.drools.examples.sudoku 包含以下用于实施元素 Cell 对象及其各种聚合的类集：

- CellFile 类，其子类型 CellRow、CellCol 和 CellSqr 是 CellGroup 类。
- Cell 和 CellGroup 子类的 SetOfNine 中的一个属性 自由 提供带有 type Set<Integer> 的属性。对于 Cell 类，该集合代表个人候选集。对于 CellGroup 类，该集合就是其所有单元集的 union（仍然需要分配的数字集）。

在 Sudoku 示例中，81 Cell 和 27 CellGroup 对象和由 Cell 属性 cellRow、cellCol 和 cellSqr 提供的链接，以及 CellGroup 属性 单元单元 （ Cell II 对象列表）。使用这些组件，您可以编写用于检测特定情况的规则，允许为某个候选项集中的单元或消除值分配值。

- Setting class 用于触发值分配后的操作。在检测到新情况的所有规则中使用了 设置 事实，以避免重新操作中间状态。

- Stepping 类用于低优先级规则，当 "Step" 没有定期终止时，会停止紧急情况。这个行为表示这个程序无法解决。

- 主类 org.drools.examples.sudoku.SudokuExample 实施 Java 应用程序，并合并所有这些组件。

Sudoku 验证规则(validate.drl)

Sudoku 示例中的 validate.drl 文件包含在单元组中检测重复数字的验证规则。它们组合成 "验证" 认证组，用户可在用户加载点后明确激活规则。

三个规则 "重复为 cell ..." 所有功能的 when 条件如下：

- 规则中的第一个条件会找到一个带有分配值的单元。

- 规则中的第二个条件会拉取单元所属的三个单元组。

- 最终条件找到一个单元（除前一个）的单元（除前一个），其值与第一单元相同，以及同一行、列或方括号，具体取决于规则。

规则 "duicate in cell ..."

```
rule "duplicate in cell row"
  when
    $c: Cell( $v: value != null )
    $cr: CellRow( cells contains $c )
    exists Cell( this != $c, value == $v, cellRow == $cr )
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in row " + $cr.getNumber() );
end

rule "duplicate in cell col"
  when
    $c: Cell( $v: value != null )
    $cc: CellCol( cells contains $c )
    exists Cell( this != $c, value == $v, cellCol == $cc )
```

```
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in col " + $cc.getNumber() );
end

rule "duplicate in cell sqr"
  when
    $c: Cell( $v: value != null )
    $cs: CellSqr( cells contains $c )
    exists Cell( this != $c, value == $v, cellSqr == $cs )
  then
    System.out.println( "cell " + $c.toString() + " has duplicate in its square of nine." );
end
```

规则 **"terminate group" 是最后触**发**的最后一个。**这**个**规则**会**显**示信息并停止序列。**

规则**"确定**组**"**

```
rule "terminate group"
    salience -100
  when
  then
    System.out.println( "Validation complete." );
    drools.halt();
end
```

Sudoku 解决规则**(sudoku.drl)**

Sudoku 示例中的 sudoku.drl 文件包含三种类**型的**规则**：一个**组处**理一个到**单**元的分配，**另**一个**组检测**到可行的分配，第三个**组**会从 candidate 集合中消除**值**。**

规则 **"set a value", "iminate a value from Cell " 和 "retract settings" 取决于 Setting** 对**象的存在。第一个**规则处**理分配**给单**元的分配，操作会从**单**元 的空**闲 组**中**删**除**值。这**个**组**也会降低一个**计**数器，在零**时**将控制**权**返回到名**为 **fireUntilHalt（） 的 Java** 应**用程序。**

规则 **"imininate a value from Cell" 的目的是**减**少与新分配的**单**元相关的所有**单**元的列表。最后，当**进**行所有的精**简时，规则 **"retract** 设**置" 会回收触**发 设**置 事**实**。**

rules "set a value", "iminate a value from a Cell" and "retract set"

```
// A Setting object is inserted to define the value of a Cell.
// Rule for updating the cell and all cell groups that contain it
rule "set a value"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // A matching Cell, with no value set
    $c: Cell( rowNo == $rn, colNo == $cn, value == null,
              $cr: cellRow, $cc: cellCol, $cs: cellSqr )

    // Count down
    $ctr: Counter( $count: count )
  then
    // Modify the Cell by setting its value.
    modify( $c ){ setValue( $v ) }
    // System.out.println( "set cell " + $c.toString() );
    modify( $cr ){ blockValue( $v ) }
    modify( $cc ){ blockValue( $v ) }
    modify( $cs ){ blockValue( $v ) }
    modify( $ctr ){ setCount( $count - 1 ) }
end

// Rule for removing a value from all cells that are siblings
// in one of the three cell groups
rule "eliminate a value from Cell"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // The matching Cell, with the value already set
    Cell( rowNo == $rn, colNo == $cn, value == $v, $exCells: exCells )

    // For all Cells that are associated with the updated cell
    $c: Cell( free contains $v ) from $exCells
  then
    // System.out.println( "clear " + $v + " from cell " + $c.posAsString()  );
    // Modify a related Cell by blocking the assigned value.
    modify( $c ){ blockValue( $v ) }
end

// Rule for eliminating the Setting fact
rule "retract setting"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // The matching Cell, with the value already set
    $c: Cell( rowNo == $rn, colNo == $cn, value == $v )

    // This is the negation of the last pattern in the previous rule.
    // Now the Setting fact can be safely retracted.
    not( $x: Cell( free contains $v )
```

```
        and
        Cell( this == $c, exCells contains $x ) )
  then
    // System.out.println( "done setting cell " + $c.toString() );
    // Discard the Setter fact.
    delete( $s );
    // Sudoku.sudoku.consistencyCheck();
end
```

两种解决规则可检测某个情况，在可以分配多个单元时。对于一个包含单个数字的候选集，规则 "single" 触发。当 单个候选者没有单元格时，规则"隐藏 单"触发，但存在包含候选的单元时，这不包括在单元所属的三个组里的其它单元格中。这两个规则都创建并插入 设置 事实。

规则"单一"和"隐藏单一"

```
// Detect a set of candidate values with cardinality 1 for some Cell.
// This is the value to be set.
rule "single"
  when
    // Currently no setting underway
    not Setting()

    // One element in the "free" set
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount == 1 )
  then
    Integer i = $c.getFreeValue();
    if (explain) System.out.println( "single " + i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, i ) );
end

// Detect a set of candidate values with a value that is the only one
// in one of its groups. This is the value to be set.
rule "hidden single"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Some integer
    $i: Integer()

    // The "free" set contains this number
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount > 1, free contains $i )

    // A cell group contains this cell $c.
    $cg: CellGroup( cells contains $c )
```

```
    // No other cell from that group contains $i.
    not ( Cell( this != $c, free contains $i ) from $cg.getCells() )
  then
    if (explain) System.out.println( "hidden single " + $i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, $i ) );
end
```

来自最大的组的规则（单独或在两组或三组）实施多种解决技术，用于手动解决 Sudoku 议会。

规则 **"naked pair" 在组的两个单元中检测相同的候选大小集合。这两个值可以从该组的所有其他候选组中删除。**

规则**"naked pair"**

```
// A "naked pair" is two cells in some cell group with their sets of
// permissible values being equal with cardinality 2. These two values
// can be removed from all other candidate lists in the group.
rule "naked pair"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // One cell with two candidates
    $c1: Cell( freeCount == 2, $f1: free, $r1: cellRow, $rn1: rowNo, $cn1: colNo, $b1: cellSqr )

    // The containing cell group
    $cg: CellGroup( freeCount > 2, cells contains $c1 )

    // Another cell with two candidates, not the one we already have
    $c2: Cell( this != $c1, free == $f1 /*** , rowNo >= $rn1, colNo >= $cn1 ***/ ) from $cg.cells

    // Get one of the "naked pair".
    Integer( $v: intValue ) from $c1.getFree()

    // Get some other cell with a candidate equal to one from the pair.
    $c3: Cell( this != $c1 && != $c2, freeCount > 1, free contains $v ) from $cg.cells
  then
    if (explain) System.out.println( "remove " + $v + " from " + $c3.posAsString() + " due to naked pair
at " + $c1.posAsString() + " and " + $c2.posAsString() );
    // Remove the value.
    modify( $c3 ){ blockValue( $v ) }
end
```

*...**"中三个**规则**" 函数与**规则 **"naked pair"** 类**似。**这**些**规则检测**一个**组**只**两**个**单**元格中**两**个数字的子集，且在**组**的任何其他**单**元格中都没有**发**生任何**值。这**意味着，其他所有候**选**者都可以从**两**个**单**元格中去除以**隐**藏的**隐**藏**对。*

*规则**"hidden** 对**在 ..."***

```
// If two cells within the same cell group contain candidate sets with more than
// two values, with two values being in both of them but in none of the other
// cells, then we have a "hidden pair". We can remove all other candidates from
// these two cells.
rule "hidden pair in row"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Establish a pair of Integer facts.
    $i1: Integer()
    $i2: Integer( this > $i1 )

    // Look for a Cell with these two among its candidates. (The upper bound on
    // the number of candidates avoids a lot of useless work during startup.)
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellRow: cellRow )

    // Get another one from the same row, with the same pair among its candidates.
    $c2: Cell( this != $c1, cellRow == $cellRow, freeCount > 2, free contains $i1 && contains $i2 )

    // Ascertain that no other cell in the group has one of these two values.
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellRow.getCells() )
  then
    if( explain) System.out.println( "hidden pair in row at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    // Set the candidate lists of these two Cells to the "hidden pair".
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellCol: cellCol )
```

```
    $c2: Cell( this != $c1, cellCol == $cellCol, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellCol.getCells() )
  then
    if (explain) System.out.println( "hidden pair in column at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in square"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
            $cellSqr: cellSqr )
    $c2: Cell( this != $c1, cellSqr == $cellSqr, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellSqr.getCells() )
  then
    if (explain) System.out.println( "hidden pair in square " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end
```

*两条规则处**理行和列中的 "Xwing "。当**值的每**两个可能**单元（或列）中都只包括**两个可能的**单元格**时，这些候**选人也存在于同一列（或行），那么可以**删除列内（或行）的所有其他候**选者。当遵循**这些**规则**中的模式序列**时，请注意如何以方便的**词语**表达**的条件，如 相同的 或 仅导**致 具有合适的限制的模式，或以 not 为**前缀。*

*规则 **"X-wings in …"***

```
rule "X-wings in rows"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
            $ra: cellRow, $rano: rowNo,        $c1: cellCol,       $c1no: colNo )
    $cb1: Cell( freeCount > 1, free contains $i,
            $rb: cellRow, $rbno: rowNo > $rano,      cellCol == $c1 )
    not( Cell( this != $ca1 && != $cb1, free contains $i ) from $c1.getCells() )

    $ca2: Cell( freeCount > 1, free contains $i,
```

```
            cellRow == $ra, $c2: cellCol,       $c2no: colNo > $c1no )
    $cb2: Cell( freeCount > 1, free contains $i,
            cellRow == $rb,      cellCol == $c2 )
    not( Cell( this != $ca2 && != $cb2, free contains $i ) from $c2.getCells() )

    $cx: Cell( rowNo == $rano || == $rbno, colNo != $c1no && != $c2no,
            freeCount > 1, free contains $i )
  then
    if (explain) {
       System.out.println( "X-wing with " + $i + " in rows " +
          $ca1.posAsString() + " - " + $cb1.posAsString() +
          $ca2.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "X-wings in columns"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
            $c1: cellCol, $c1no: colNo,         $ra: cellRow,        $rano: rowNo )
    $ca2: Cell( freeCount > 1, free contains $i,
            $c2: cellCol, $c2no: colNo > $c1no,      cellRow == $ra )
    not( Cell( this != $ca1 && != $ca2, free contains $i ) from $ra.getCells() )

    $cb1: Cell( freeCount > 1, free contains $i,
            cellCol == $c1, $rb: cellRow,  $rbno: rowNo > $rano )
    $cb2: Cell( freeCount > 1, free contains $i,
            cellCol == $c2,      cellRow == $rb )
    not( Cell( this != $cb1 && != $cb2, free contains $i ) from $rb.getCells() )

    $cx: Cell( colNo == $c1no || == $c2no, rowNo != $rano && != $rbno,
            freeCount > 1, free contains $i )
  then
    if (explain) {
       System.out.println( "X-wing with " + $i + " in columns " +
          $ca1.posAsString() + " - " + $ca2.posAsString() +
          $cb1.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString()  );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

这两个规则 ***"intersection removal ..." 基于一个方方或单一列内出现某种数量的受限位置。这意味着这个数字必须位于一行或列的两个或者三单元格之一，并可从组所有其他单元的候选单元中删除。这个模式决定了限制发生的情况，然后针对方括号外的每个单元格以及同一单元文件内触发。***

rules "interection removal ..."

```
rule "intersection removal column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
    $c: Cell( free contains $i, $cs: cellSqr, $cc: cellCol )
    // Does not occur in another cell of the same square and a different column
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellCol != $cc )

    // A cell exists in the same column and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellCol == $cc, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "column elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "intersection removal row"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
    $c: Cell( free contains $i, $cs: cellSqr, $cr: cellRow )
    // Does not occur in another cell of the same square and a different row.
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellRow != $cr )

    // A cell exists in the same row and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellRow == $cr, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "row elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

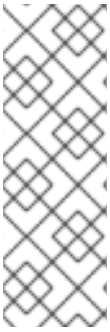
这些规则已足够多，但并非所有 Sudoku 模糊。为了解决非常困难的网格，规则集需要更复杂的规则。（通常，一些不清点可以通过试用和错误解决。）

21.9. CONWAYS OF LIFE EXAMPLE DECISIONS（RULEFLOW 组和 GUI 集成）

Conwayss Game of Life example 决策集，按照 John Conway 的 advertise cellular automaton，演示如何使用 ruleflow 组来控制规则执行。这个示例还演示了如何将 Red Hat Process Automation Manager 规则与图形用户界面(GUI)集成，在这种情况下，基于 Swing 的实施是生命周期。

以下是生命周期游戏(Conway)示例概述：

- 名称 ：conway
- 主要类别 ：org.drools.examples.conway.ConwayRuleFlowGroupRun,org.drools.examples.conway.Conway.Conway.ConwayAgendaGroupRun (in src/main/java)
- 模块:droolsjbpm-integration-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.conway.*.drl （在 src/main/resources）
- 目标 ：演示 ruleflow 组和 GUI 集成



注意

生命周期示例的 Conway's Game 与 Red Hat Process Automation Manager 中的大多数其他示例决策集分开，并位于来自红帽客户门户网站的 ~/rhpam-7.13.5-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples，它包括了来自红帽 客户门户网站的 Red Hat Process Automation Manager 7.13.5 -sources/src/droolsjbpm-integration-examples。

在 Conway 的游戏中，用户通过创建初始配置或具有定义的属性的高级模式来与游戏进行交互，然后观察初始状态的演变方式。游戏旨在展示人们的开发，生成生成。根据所有单元的同步评估，来自前一个结果的生成结果。

以下基本规则管理下一代内容：

- 如果实时单元单元少于两个实时邻居，它会断出一个词子。
- 如果实时单元单元有超过 3 个实时邻居，它将从过度浏览。
- 如果一个死的单元只有 3 个实时邻居，它会进入生命期。

任何不符合这些条件的单元都会被保留为生成下一个。

Conways of Life 示例游戏使用带有 ruleflow-group 属性的 Red Hat Process Automation Manager 规则来定义在游戏中实施的模式。示例还包括一个决策集版本，通过日程安排组实现相同的行为。日程表组使您可以对决策引擎日程表进行分区，以便对规则组提供执行控制。默认情况下，所有规则均位于 MAIN 日程小组。您可以使用 schedule -group 属性来指定该规则的不同日程表组。

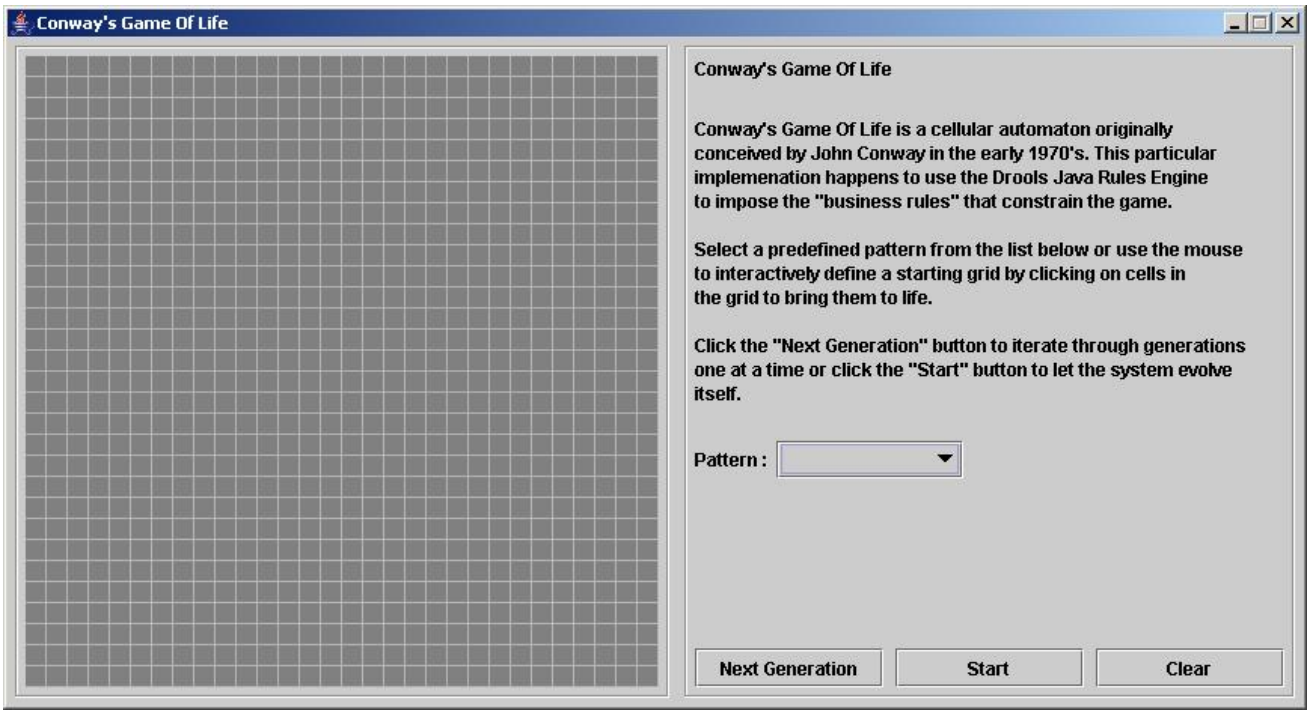
此概述不使用日程表组探索 Conway 示例的版本。有关日程表组的更多信息，请参阅 Red Hat Process Automation Manager 示例决策集，其具体处理日程表组。

沟通执行和互动示例

与其他 Red Hat Process Automation Manager 决策示例类似，您可以通过运行 org.drools.examples.conway.ConwayRuleFlowGroupRun 类作为 IDE 中的 Java 应用程序来执行 Conway 规则流示例。

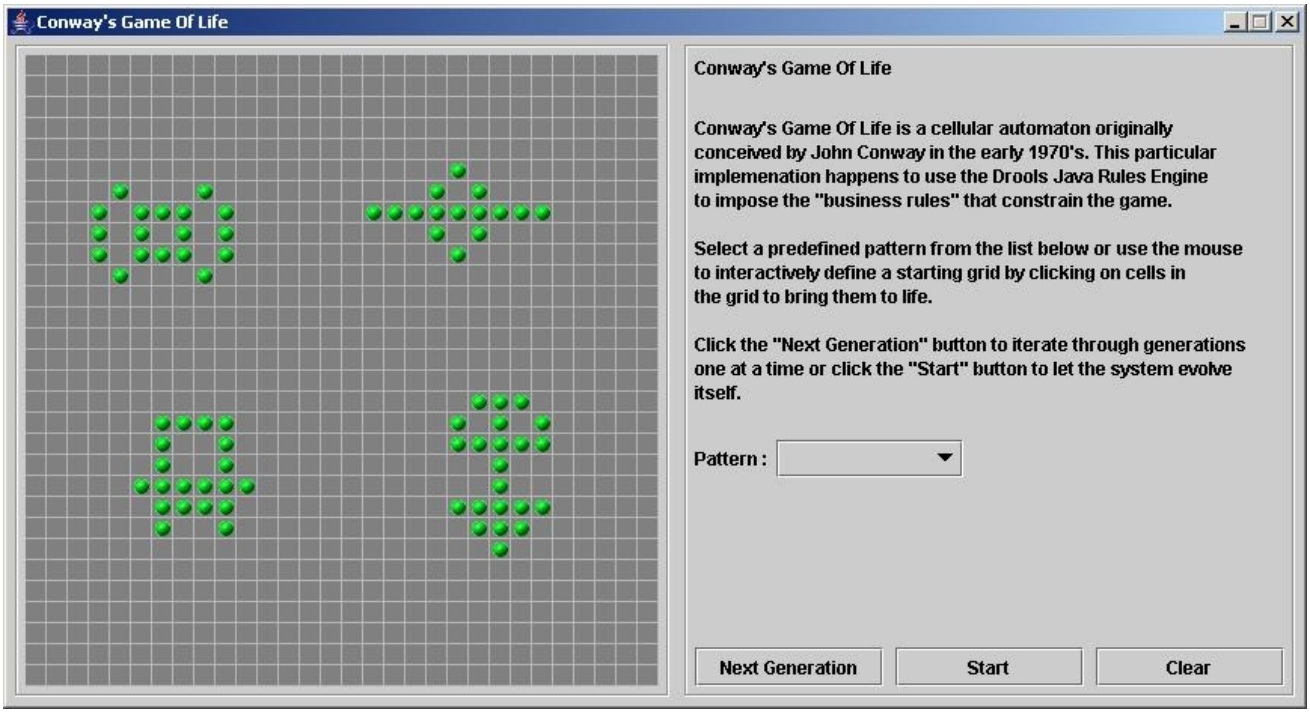
当您执行 Conway 示例时，会出现 Life ways Game of Life GUI 窗口的 Game。此窗口包含空网格，或"arena"，其中将进行模拟。网格最初为空，因为还没有在系统中提供实时单元。

图 21.24. 启动后继续 GUI 示例



从 Pattern 下拉菜单中选择预定义的模式，再单击 Next Generation 以单击每个填充生成。每个单元都可以处于活动状态或死的，其中 live cells 包含一个绿色的 ball。随着人们从初始模式演进，根据游戏的规则，与邻座单元相关的单元格或相对于邻居单元的划分。

图 21.25. 在 Conway 示例中生成演进



邻居不仅包括左、右、顶部和下部的单元格，这些单元格是连接的单位，因此每个单元总数为 8 个邻居。例外是基格（仅有三个邻居），以及这四个边的单元格，分别是五个邻居。

您可以通过单击单元格来手动干预来创建或终止单元。

要从初始模式自动通过 evolution 运行，请单击 Start。

使用 ruleflow 组验证规则示例

ConwayRuleFlowGroupRun 示例中的规则使用 ruleflow 组来控制规则执行。ruleflow group 是 rule flow-group rule 属性关联的一组规则。这些规则只能在激活组时触发。只有当 ruleflow 图的协作达到代表组的节点时，组本身才会变为活跃状态。

Conway 示例对规则使用以下 ruleflow 组：

- "注册邻居"
- "评估"
- "calculate"
- "重置计算"
- "birth"
- "kill"
- "全部技能"

所有 Cell 对象都插入到 KIE 会话中，ruleflow 组中允许 "register ..." 规则由 ruleflow 进程执行。这一组四个规则可在某些单元及其 northeastern、northern、northwestern 和 western 邻居之间创建邻居关系。

这种关系是双向的，可以处理其他四个方向。Border 单元不需要任何特殊处理。这些单元不会与邻居单元（没有任何）搭配使用。

通过为所有规则触发了所有激活的时间，所有单元格都与其所有邻居单元相关。

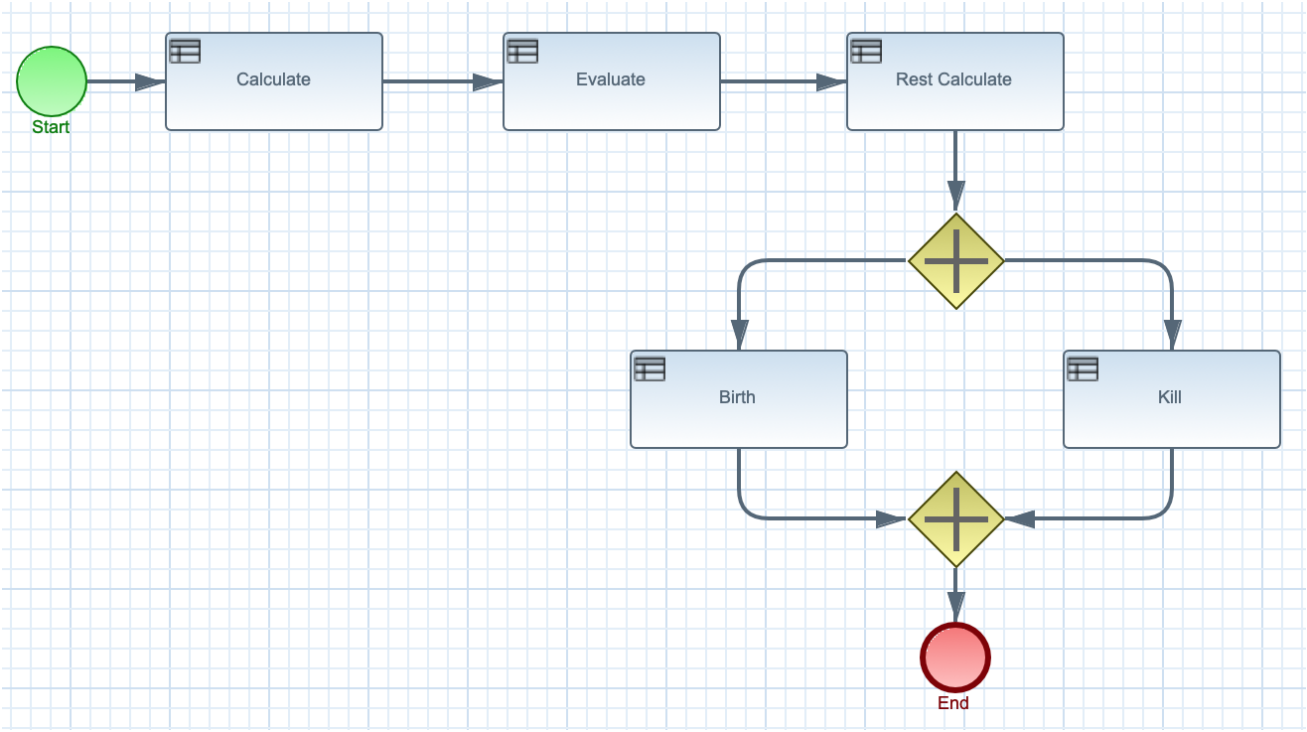
规则 **"register ..."**

```
rule "register north east"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northEast : Cell( row  == ($row - 1), col == ( $col + 1 ) )
  then
    insert( new Neighbor( $cell, $northEast ) );
    insert( new Neighbor( $northEast, $cell ) );
end

rule "register north"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $north : Cell( row  == ($row - 1), col == $col )
  then
    insert( new Neighbor( $cell, $north ) );
    insert( new Neighbor( $north, $cell ) );
end

rule "register north west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northWest : Cell( row  == ($row - 1), col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $northWest ) );
    insert( new Neighbor( $northWest, $cell ) );
end

rule "register west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $west : Cell( row  == $row, col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $west ) );
    insert( new Neighbor( $west, $cell ) );
end
```

插入所有单元格后，一些 Java 代码会将模式应用到网格，将某些单元设置为 **Live**。然后，当用户点击 **Start** 或 **Next Generation** 时，该示例执行 **Generation ruleflow**。此 ruleflow 管理每个生成周期中的

所有单元更改。

图 21.26. 生成规则流



ruleflow 进程进入 "evaluate" ruleflow 组，以及组中的任何活动规则可以触发。这个组中的规则 "Kill the ..." 和 "Give Birth" 将规则规则应用到 birth 或 kill 单元。这个示例使用 phase 属性来根据特定规则组驱动 Cell 对象的原因。通常，该阶段与 ruleflow 进程定义中的 ruleflow 组关联。

请注意，该示例不会更改此时任何 Cell 对象的状态，因为它必须在应用这些更改前完成完整的评估。示例将单元设置为 Phase.KILL 或 Phase.BIRTH，后者稍后用于控制应用到 Cell 对象的操作。

规则"Kill the ..." 和 "Give Birth"

```
rule "Kill The Lonely"
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has fewer than 2 live neighbors.
    theCell: Cell( liveNeighbors < 2, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
        setPhase( Phase.KILL );
    }
end

rule "Kill The Overcrowded"
```

```
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has more than 3 live neighbors.
    theCell: Cell( liveNeighbors > 3, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
       setPhase( Phase.KILL );
    }
end

rule "Give Birth"
    ruleflow-group "evaluate"
    no-loop
  when
    // A dead cell has 3 live neighbors.
    theCell: Cell( liveNeighbors == 3, cellState == CellState.DEAD,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
       theCell.setPhase( Phase.BIRTH );
    }
end
```

评估了网格中的所有 Cell 对象后，该示例使用 "reset calculate" 规则清除 "calculate" ruleflow 组中的任何激活。然后，如果 ruleflow 激活了 ruleflow，示例将启用规则 "kill" 和 "birth" 来触发的规则。这些规则应用状态更改。

规则"重置计算"、 "kill"和"birth"

```
rule "reset calculate"
    ruleflow-group "reset calculate"
  when
  then
    WorkingMemory wm = drools.getWorkingMemory();
    wm.clearRuleFlowGroup( "calculate" );
end

rule "kill"
    ruleflow-group "kill"
    no-loop
  when
    theCell: Cell( phase == Phase.KILL )
  then
    modify( theCell ){
       setCellState( CellState.DEAD ),
```

```
        setPhase( Phase.DONE );
    }
end

rule "birth"
    ruleflow-group "birth"
    no-loop
  when
    theCell: Cell( phase == Phase.BIRTH )
  then
    modify( theCell ){
        setCellState( CellState.LIVE ),
        setPhase( Phase.DONE );
    }
end
```

在这个阶段，几个 Cell 对象已被修改，其状态会更改为 LIVE 或 DEAD。当单元变为 live 或 dead 时，示例使用规则 "Calculate ..." 中的邻居关系来迭代所有周围的单元格，从而增加或减少 liveNeighbor 计数。任何改变计数的单元也会设置为 EVALUATE 阶段，以确保它在 ruleflow 过程的评估阶段包含在原因中。

在为所有单元确定和设置实时数后，ruleflow 进程结束。如果用户最初点击 Start，则决策引擎在该处重启规则流。如果用户最初单击 下一步生成，用户可以请求另一个生成。

规则**"Calculate ..."**

```
rule "Calculate Live"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.LIVE )
    Neighbor( cell == theCell, $neighbor : neighbor )
  then
    modify( $neighbor ){
        setLiveNeighbors( $neighbor.getLiveNeighbors() + 1 ),
        setPhase( Phase.EVALUATE );
    }
end

rule "Calculate Dead"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.DEAD )
    Neighbor( cell == theCell, $neighbor : neighbor )
```

```
  then
    modify( $neighbor ){
       setLiveNeighbors( $neighbor.getLiveNeighbors() - 1 ),
       setPhase( Phase.EVALUATE );
    }
end
```

21.10. DOOM 示例决策内部（反向连锁和递归）

Doom 示例决策集的 House 展示了决策引擎如何使用向串联和递归来达到等级系统中定义的目标或子项。

以下是 Doom 示例 House 的概述：

- 名称 ：反向链接
- 主类 ：org.drools.examples.backwardchaining.HouseOfDoomMain （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.backwardchaining.BC-Example.drl (src/main/resources)
- 目标 ：演示后向链和递归

反向链规则系统是一个目标驱动的系统，从结论开始，决定引擎尝试满足（通常使用递归）。如果系统无法达到结论或目标，它会搜索部分当前目标的子项。系统会继续这个过程，直到初始的结论是满足或者所有子语满意。

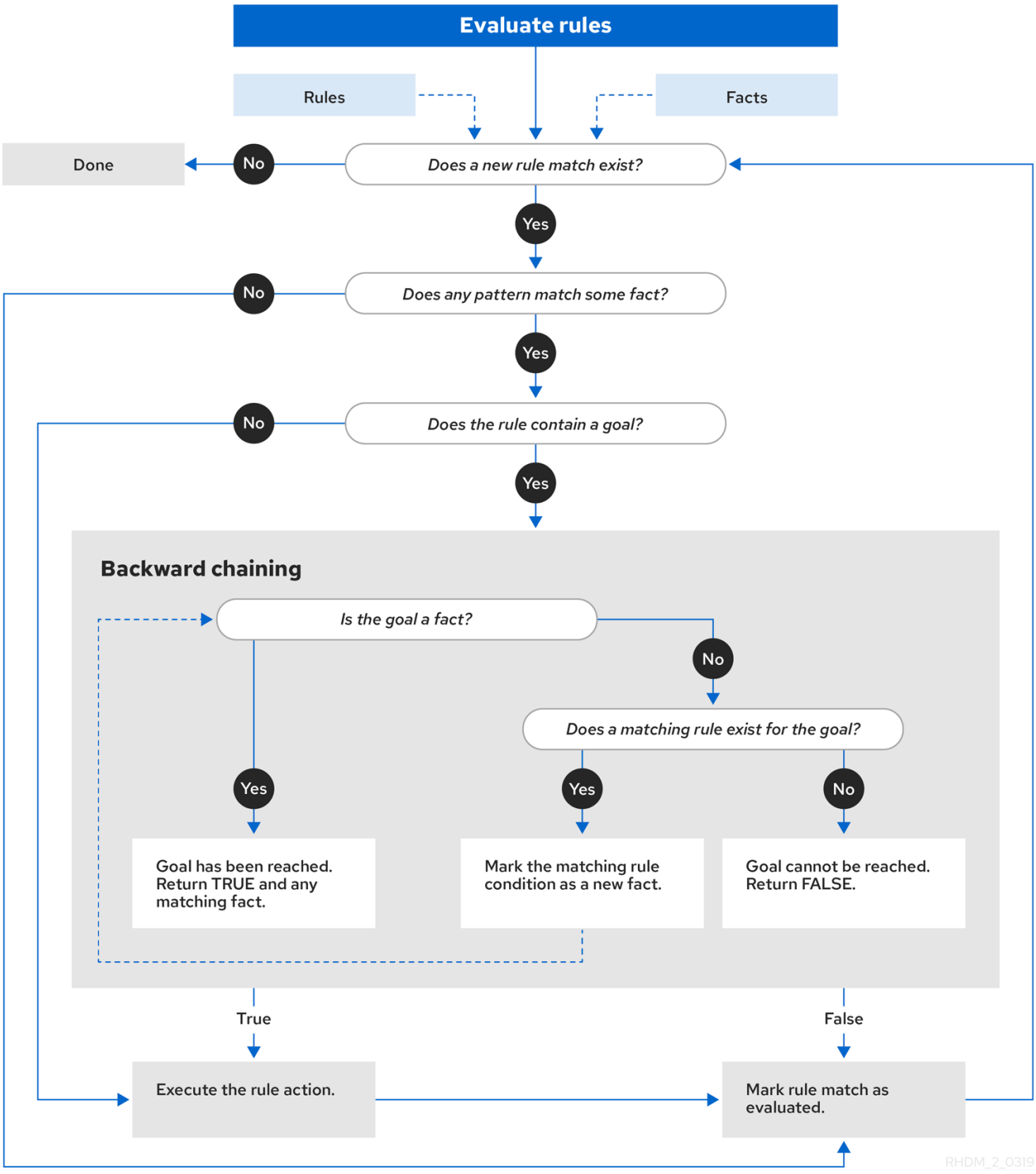
与之相反，正向链规则系统是一个数据驱动的系统，从决策引擎的工作内存中以事实开头，并对该事

实的更改做出响应。当对象插入到工作内存时，因为更改是由日程表计划执行而变为 true 的任何规则条件。

Red Hat Process Automation Manager 中的决策引擎使用正向和向后链来评估规则。

下图显示了如何使用转发链在逻辑流中的反向链接片段评估规则：

图 21.27. 使用转发和向后链的规则评估逻辑



例如，House of Doom 示例使用包含各种查询类型的规则来查找房间和项目的位置。示例类 Location.java 包含示例中使用的 项目 和位置 元素。示例类 HouseOfDoomMain.java 在其所在位置插入项目或房间，并执行规则。

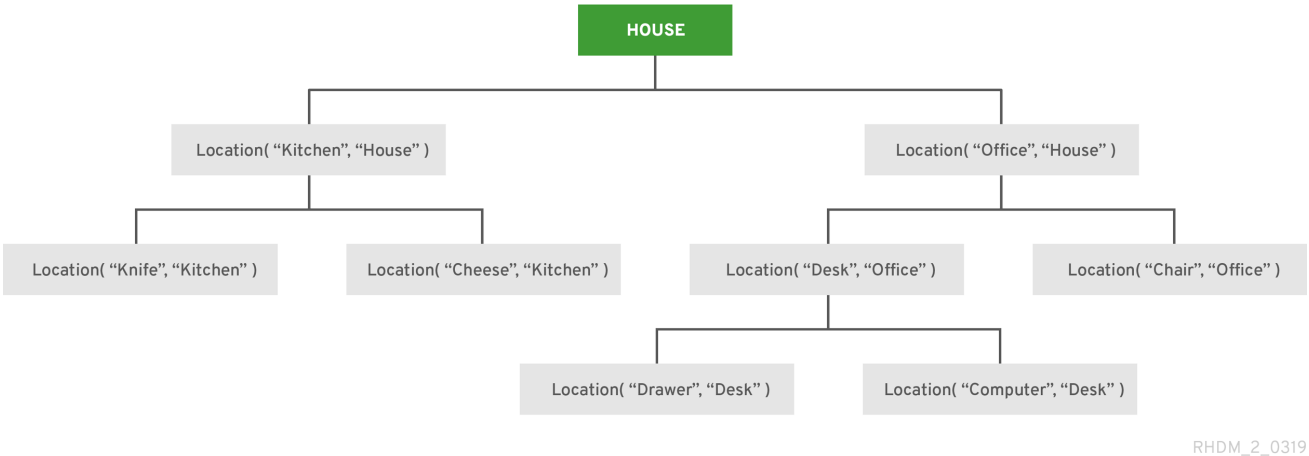
HouseOfDoomMain.java 类中的项目和位置

```
ksession.insert( new Location("Office", "House") );
ksession.insert( new Location("Kitchen", "House") );
ksession.insert( new Location("Knife", "Kitchen") );
ksession.insert( new Location("Cheese", "Kitchen") );
ksession.insert( new Location("Desk", "Office") );
ksession.insert( new Location("Chair", "Office") );
ksession.insert( new Location("Computer", "Desk") );
ksession.insert( new Location("Drawer", "Desk") );
```

示例规则依赖于向后链和递归来确定内部结构中所有项目和房间的位置。

下图展示了 Doom 的 House 的结构以及其中的项目和房间：

图 21.28. Doom 结构内部



要执行示例，请运行 org.drools.examples.backwardchaining.HouseOfDoomMain 类，在 IDE 中作为 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的执行输出

```
go1
Office is in the House
---
go2
Drawer is in the House
---
go3
---
Key is in the Office
---
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
---
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

示例中的所有规则均已触发，以检测内部所有项目的位置，并在输出中打印各个项目的位置。

递归查询和相关规则

递归查询会重复搜索数据结构的层次结构，以获得元素之间的关系。

在 Doom 示例的 House of Doom 示例中， BC-Example.drl 文件包含一个 isContainedIn 查询，示例中大多数规则的查询用于递归评估插入到决策引擎中的数据结构：

BC-Example.drl 中的递归查询

```
query isContainedIn( String x, String y )
  Location( x, y; )
  or
  ( Location( z, y; ) and isContainedIn( x, z; ) )
end
```

规则 "go" 打印插入到系统中的每个字符串，以确定如何实施项目，规则 "go1" 调用查询 为 ContainedIn ：

规则 "go" 和 "go1"

```
rule "go" salience 10
  when
    $s : String()
  then
    System.out.println( $s );
end

rule "go1"
  when
    String( this == "go1" )
    isContainedIn("Office", "House"; )
  then
    System.out.println( "Office is in the House" );
end
```

这个示例将 "go1" 字符串插入到决策引擎中，并激活 "go1" 规则来检测该项目 办事处 位于位置

House 中：

插入字符串和触发规则

```
ksession.insert( "go1" );
ksession.fireAllRules();
```

IDE 控制台中的规则**"go1"**输**出**

```
go1
Office is in the House
```

临时**防止**规则

传输**冲突是父元素中包含的元素之**间**的关系，**该**元素在分**级结构**中高于多个**级别**。**

规则 **"go2"** 标识 **Drawer 和 House 的**传输**冲突：D rawer 在 House** 处**位于**办**事**处**的 Desk。**

```
rule "go2"
  when
    String( this == "go2" )
    isContainedIn("Drawer", "House"; )
  then
    System.out.println( "Drawer is in the House" );
end
```

这**个示例将 "go2" 字符串插入到决策引擎中，并激活 "go2"** 规则，**以**检测项**目 Drawer 最**终**在位置 House 中：**

插入字符串和触发规则

```
ksession.insert( "go2" );
ksession.fireAllRules();
```

-

IDE 控制台中的规则"go2"输出

```
go2
Drawer is in the House
```

决策引擎根据以下逻辑决定这一结果：

1. 查询会以递归方式搜索内部的几个级别，以检测 Drawer 和 House 之间传输冲突。
2. 查询该选项使用 (z, y;) 而不是使用 Location(x, y;)，因为 Drawer 不在 House 中。
3. z 参数目前未绑定，这意味着它没有值并返回参数中的所有内容。
4. y 参数目前绑定到 House，因此 z 返回 office 和 Kitchen。
5. 该查询从 办公室 收集信息并递归检查该办事处是否位于 办事处。对于这些参数，调用查询行 isContainedIn(x, z;)。
6. 办事处 没有直接存在 Drawer 实例，因此无法找到任何匹配项。
7. 通过 z unbound，查询会返回 办事处 中的数据，并确定 z == Desk。

   ```
   isContainedIn(x==drawer, z==desk)
   ```

8. isContainedIn 查询会以递归方式搜索三次，而且在第三个时间，查询检测到 Desk 中的 Drawer 实例。

```
Location(x==drawer, y==desk)
```

9. 在第一个位置上的此匹配项后，查询会以递归方式搜索结构，以确定 Drawer 位于 Desk 中，Desk 位于 办事处，且该 办事处 位于 House 中。因此，Drawer 位于 House 中，该规则会满足。

被动查询规则

被动查询会搜索数据结构的层次结构，以获得元素之间的关系，并在修改结构中的元素时动态更新。

规则 "go3" 函数作为被动查询，通过传输冲突检测到在 办公室中 是否出现新的项目 密钥 ： 办公室的 Drawer 中的密钥。

规则"go3"

```
rule "go3"
  when
    String( this == "go3" )
    isContainedIn("Key", "Office"; )
  then
    System.out.println( "Key is in the Office" );
end
```

这个示例将 "go3" 字符串插入到决策引擎中，并激活 "go3" 规则。最初，此规则不满意，因为内部结构中没有项目 密钥，因此该规则不会产生任何输出。

插入字符串和触发规则

```
ksession.insert( "go3" );
ksession.fireAllRules();
```

IDE 控制台中的规则"go3"输出（不满意）

```
go3
```

然后，示例在位置 Drawer 中插入一个新项目 密钥，它位于 办事处。这个更改满足在 "go3" 规则中传输冲突，输出会被相应地填充。

插入新项目位置和触发规则

```
ksession.insert( new Location("Key", "Drawer") );
ksession.fireAllRules();
```

IDE 控制台中的规则"go3"输出(satisfied)

```
Key is in the Office
```

此更改也会在查询后续递归搜索中包含的结构中添加另一个级别。

在规则中带有 unbound 参数的查询

带有一个或多个未绑定参数的查询会返回查询定义的（绑定）参数内所有未定义（绑定）项。如果查询中的所有参数都未绑定，则查询会返回查询范围内的所有项目。

规则 "go4" 使用 unbound 参数项搜索绑定参数 办事处 的所有项目，而不是使用 bound 参数搜索 office 中的特定项目：

规则"go4"

```
rule "go4"
  when
```

```
    String( this == "go4" )
    isContainedIn(thing, "Office"; )
  then
    System.out.println( thing + "is in the Office" );
end
```

这个示例将 "go4" 字符串插入到决策引擎中，并激活 "go4" 规则，以返回办公室中的所有项目：

插入字符串和触发规则

```
ksession.insert( "go4" );
ksession.fireAllRules();
```

IDE 控制台中的规则"go4"输出

```
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
```

规则 "go5" 使用 unbound 参数项 和位置 来搜索整个 House 数据结构中的所有项目及其位置：

规则**"go5"**

```
rule "go5"
  when
    String( this == "go5" )
    isContainedIn(thing, location; )
```

```
  then
    System.out.println(thing + " is in " + location );
end
```

这个示例将 "go5" 字符串插入到决策引擎中，并激活 "go5" 规则，以返回 House 数据结构中的所有项目及其位置：

插入字符串和触发规则

```
ksession.insert( "go5" );
ksession.fireAllRules();
```

IDE 控制台中的规则"go5"输出

```
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

第 22 章 与 DRL 相关的性能调优注意事项

以下关键概念或建议做法可帮助您优化 DRL 规则和决策引擎性能。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。

定义从左到右侧的模式约束的属性和值

在 DRL 模式限制中，确保事实属性名称位于运算符的左侧，并且值（续或变量）位于右侧。属性名称必须始终是索引中的键，而不是值。例如，写入 Person(firstName == "John") 而不是 Person(John" == firstName)。定义约束属性和值从右到左可隐藏决策引擎性能。

有关 DRL 模式和约束的详情，请参考 第 16.8 节 “DRL 中的规则条件(WHEN)”。

在可能的情况下，使用等于运算符以外的其他 Operator 类型

虽然决策引擎支持许多可用于定义业务逻辑的 DRL 操作器类型，但同样性运算符 == 被决策引擎最常评估。无论何时，都可使用此运算符而非其他 Operator 类型。例如，其模式 Person(firstName == "John") 的评估比 Person(firstName != "OtherName") 更高效。在某些情况下，只使用相等的运算符可能并不现实，因此请考虑您的所有业务逻辑需要以及 DRL 操作符的选项。

首先列出限制性最严格的规则条件

对于具有多个条件的规则，请列出大多数限制中的条件，以便决策引擎能够在不满足更严格的条件时避免评估整个条件集合。

例如，以下条件是行行书规则的一部分，适用于预订机班和热线的差旅者。在此情景中，客户很少会用航班来享受此折扣的热线，因此热线条件很少满足且很少执行该规则。因此，第一个条件排序更为有效，因为它可防止决策引擎频繁评估动态条件，并在不满足 hotel 条件时不必要的地评估。

首选条件顺序： hotel 和 flight

```
when
  $h:hotel() // Rarely booked
  $f:flight()
```

效率低：flight 和 hotel

```
when
  $f:flight()
  $h:hotel() // Rarely booked
```

有关 DRL 模式和约束的详情，请参考 第 16.8 节 “DRL 中的规则条件(WHEN)”。

避免过度使用子句来迭代大型对象集合

避免使用 DRL 规则中的 from condition 元素来迭代大量对象集合，如下例所示：

带有 from clause 的条件示例

```
when
  $c: Company()
  $e : Employee ( salary > 100000.00) from $c.employees
```

在这种情况下，当规则条件被评估并减少规则评估时，决策引擎会迭代大型图形。

另外，除了添加带有大量图表的对象，而是直接将集合添加到 KIE 会话，然后在状况中加入集合，如下例所示：

没有 from clause 的条件示例

```
when
  $c: Company();
  Employee (salary > 100000.00, company == $c)
```

在这个示例中，决策引擎只迭代列表一次，并且可以更有效地评估规则。

有关 from element 或其他 DRL 条件元素的更多信息，请参阅 第 16.8.7 节 “DRL（关键字）中支持的规则条件元素”。

在用于调试日志的规则中使用决策引擎事件监听程序而不是 system.out.println 语句

您可以在用于调试日志和控制台输出的规则操作中使用 system.out.println 语句，但对许多规则执行这一操作可能会妨碍规则评估。作为更有效的替代方案，请尽可能使用内置的决策引擎事件监听程序。如果这些监听器无法满足您的要求，请使用决策引擎支持的系统日志实用程序，如 Logback、Apache Commons Logging 或 Apache Log4j。

有关支持的决策引擎事件监听程序和日志记录工具的更多信息，请参阅 Red Hat Process Automation Manager 中的决策引擎。

使用 drools-metric 模块识别规则中的障碍

您可以使用 drools-metric 模块来标识较慢的规则，特别是处理许多规则时。drools-metric 模块还可协助分析决策引擎性能。请注意，drools-metric 模块不用于生产环境。但是，您可以在测试环境中执行分析。

要使用 drools-metric 分析决策引擎性能，首先将 drools-metric 添加到项目依赖项中：

drools-metric的项目依赖项示例

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-metric</artifactId>
</dependency>
```

如果要使用 drools-metric 启用 trace 日志记录，请为 org.drools.metric.util.MetricLogUtils 配置日志记录器，如下例所示：

logback.xml 配置文件示例

```
<configuration>
  <logger name="org.drools.metric.util.MetricLogUtils" level="trace"/>
  ...
<configuration>
```

–

或者，您可以使用 drools-metric 使用 Micrometer 来公开数据。要公开数据，请启用您选择的 Micrometer registry，如下例所示：

Micrometer 的项目依赖项示例

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-jmx</artifactId> <!-- Discover more registries at micrometer.io. -->
</dependency>
```

Micrometer 的 Java 代码示例

```
Metrics.addRegitry(new JmxMeterRegistry(s -> null, Clock.SYSTEM));
```

无论您是使用日志记录还是 Micrometer，您都需要通过将系统属性 drools.metric.logger.enabled 设置为 true 来启用 MetricLogUtils。另外，您可以通过设置 drools.metric.logger.threshold 系统属性来更改指标报告的微秒阈值。

注意

仅报告超过阈值的节点执行。默认值为 500。

将 drools-metric 配置为使用日志记录后，规则执行会生成日志，如下例所示：

规则执行输出示例

```
TRACE [JoinNode(6) - [ClassObjectType class=com.sample.Order]], evalCount:1000,
elapsedMicro:5962
TRACE [JoinNode(7) - [ClassObjectType class=com.sample.Order]], evalCount:100000,
elapsedMicro:95553
TRACE [ AccumulateNode(8) ], evalCount:4999500, elapsedMicro:2172836
TRACE [EvalConditionNode(9)]:
cond=com.sample.Rule_Collect_expensive_orders_combination930932360Eval1Invoker@ee2a692
], evalCount:49500, elapsedMicro:18787
```

这个示例包括以下关键参数：

- evalCount 是节点执行期间插入的事实的约束评估数量。当 evalCount 与 Micrometer 搭配使用时，包含数据的计数器名为 org.drools.metric.evaluation.count。
- elapsedMicro 是节点在微秒中所经过的时间。当 elapsedMicro 与 Micrometer 一起使用时，查找名为 org.drools.metric.elapsed.time 的计时器。

如果您发现了一个未完成的 evalCount 或 elapsedMicro 日志，请将节点名称与 ReteDumper.dumpAssociatedRulesRete（） 输出相关联，以识别与节点关联的规则。

ReteDumper 用法示例

```
ReteDumper.dumpAssociatedRulesRete(kbase);
```

ReteDumper 输出示例

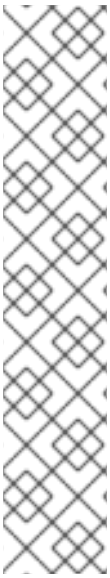
```
[ AccumulateNode(8) ] : [Collect expensive orders combination]
...
```

第 23 章 后续步骤

- 使用测试场景测试决策服务
- 打包和部署 Red Hat Process Automation Manager 项目

部分 IV. 使用指导的决策表设计决策服务

作为业务分析员或业务规则开发人员，您可以使用指导的决策表以向导的表格格式定义业务规则。这些规则编译到 Drools Rule Language(DRL)中，并形成您项目的决策服务的核心。



注意

您还可以使用决策模型和 Notation(DMN)模型设计您的决策服务，而不是基于规则或基于表的资产。有关 Red Hat Process Automation Manager 7.13 中的 DMN 支持的详情，请查看以下资源：

- 开始使用决策服务 （逐步教程，带有 DMN 决策服务示例）
- 使用 DMN 模型设计决策服务 (Red Hat Process Automation Manager 中的 DMN 支持和功能视图)

先决条件

- Business Central 中创建了指导决策表的空间和项目。每个资产都与分配给一个空间的项目相关联。详情请参阅 开始使用决策服务。

第 24 章 红帽流程自动化管理器中的决策资产

Red Hat Process Automation Manager 支持一些可用于为决策服务定义业务决策的资产。每个决策资产都有不同的优势，您可以根据目标和需求，首选使用一个或多个资产的组合。

下表重点介绍红帽流程自动化管理器项目支持的主要决策资产，以帮助您决定或确认在决策服务中定义决策的最佳方法。

表 24.1. Red Hat Process Automation Manager 支持的决策资产

asset	亮点	编写工具	Documentation
决策模型和符号 (DMN)型号	• 决策模型基于由对象管理组 (OMG)定义的符号标准。 • 使用代表部分或所有决策要求图 (DRG)的图形化决策要求图 (DRG)跟踪业务决策流程 • 使用 XML 模式，允许在 DMN 兼容平台间共享 DMN 模型 • 支持 Friendly Enough Expression Language(FEEL)，在 DMN 决策表中定义决策逻辑，以及其他 DMN 框的表达式 • 可以与业务流程模型和符号 (BPMN)流程模型有效集成 • 是创建全面、清晰和稳定的决策流程的最佳选择	Business Central 或其他与 DMN 兼容的编辑器	*使用 DMN 模型设计决策服务*
指导的决定表	• 是您在 Business Central 中基于 UI 的表格设计者创建的规则表 • 是电子表格决策表的向导替代选择 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值 • 支持点击策略、实时验证和其他资产不支持的其他功能 • 是以受控表格格式创建规则的最佳选择，可最小化编译错误	Business Central	*使用指导决策表设计决策服务*

asset	亮点	编写工具	Documentation
电子表格决策表	• 是 XLS 或 XLSX 表格决策表，您可以上传到 Business Central • 支持创建规则模板的模板键和值 • 是创建已在 Business Central 之外管理的决策表中规则的最佳选择 • 在上传后，对规则进行编译具有严格的语法要求	电子表格编辑器	*使用电子表格决策表设计决策服务*
指导规则	• 是您在 Business Central 中基于 UI 的规则设计者创建的单独规则 • 为可接受的输入提供字段和选项 • 是以受控格式创建单一规则的最佳选择，以最小化编译错误	Business Central	*使用指导规则设计决策服务*
指导规则模板	• 是您在 Business Central 中基于 UI 的模板设计者创建的可重复使用的规则结构 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值（与此资产的目的不相同） • 是创建具有相同规则结构但具有不同定义字段值的很多规则的最佳	Business Central	*使用指导规则模板设计决策服务*
DRL 规则	• 是直接在 **.drl** 文本文件中定义的单独规则 • 提供定义规则和其他规则行为的最大的灵活性 • 可以在特定的独立环境中创建，并与 Red Hat Process Automation Manager 集成 • 是创建需要高级 DRL 选项的规则的最佳选择 • 对规则进行正确编译的规则具有严格的语法要求	业务中心或集成开发环境(IDE)	*使用 DRL 规则设计决策服务*

asset	亮点	编写工具	Documentation
预测模型标记语言(PMML)型号	• 是基于数据 Mining Group(DMG)定义的标记标准预测数据分析模型。 • 使用允许 PMML 模式在 PMML 兼容的平台之间共享的 XML 模式 • 支持 Regression、scorecard、Tree、Mining 和其他模型类型 • 可以包含在独立 Red Hat Process Automation Manager 项目中，或者导入到 Business Central 中的项目 • 在 Red Hat Process Automation Manager 中，将预测数据合并为决策服务的最佳选择	PMML 或 XML 编辑器	*使用 PMML 模型设计决策服务*

在定义决策时，您还可以考虑将红帽构建的 Kogito 构建用于云原生决策服务。有关使用红帽构建的 Kogito 微服务的更多信息，请参阅 Red Hat Process Automation Manager 中的 Red Hat build of Kogito。

第 25 章 指导的决定表

指导的决策表是表格决策表的向导替代选择，以表格格式定义业务规则。借助指导的决策表，您由 Business Central 中的基于 UI 的向导领导，可帮助您根据项目中的指定数据对象定义规则属性、元数据、条件和操作。创建引导式练习后，您定义的规则会和所有其他规则资产一起编译为 Drools 规则语言 (DRL)规则。

与指导决策表相同的项目软件包中，所有相关的数据对象都必须位于与指导的决策表相同的数据对象。默认导入同一软件包中的资产。创建必要的数据对象和指导决策表后，您可以使用《指导决策表设计器中的数据 对象 "选项卡，验证所有所需的数据对象还是通过添加新项目来导入其他现有数据对象。

第 26 章 数据对象

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

26.1. 创建数据对象

以下流程是创建数据对象的一般概述。它并不适用于特定的业务资产。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Data Object。
3. 输入唯一数据对象名称，然后选择您希望数据对象可用于其他规则资产的软件包。 同一名称的数据对象不能位于同一软件包中。在指定的 DRL 文件中，您可以从任何软件包中导入数据对象。

从其他软件包导入数据对象

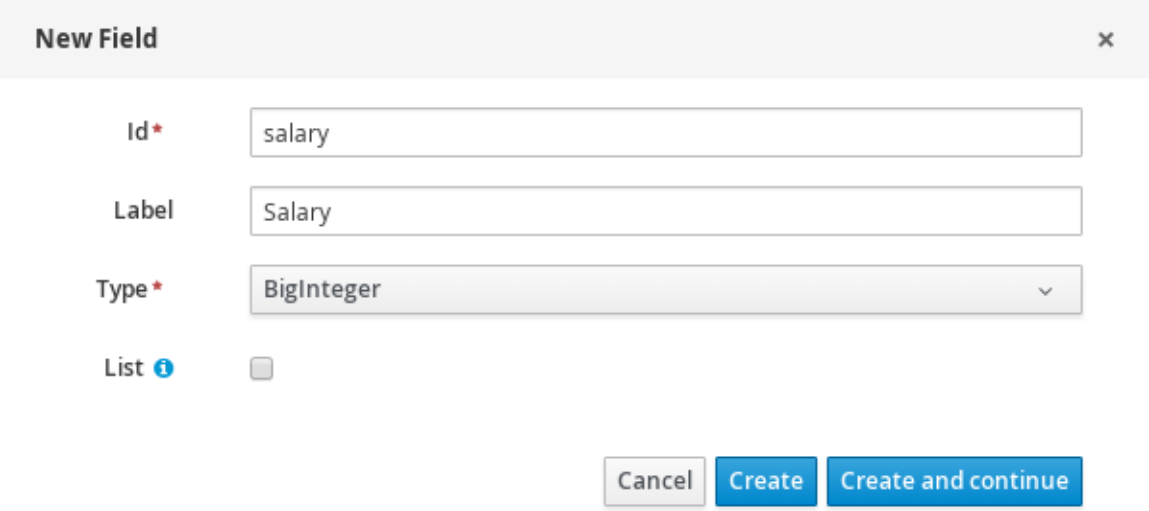
您可以将另一软件包中的现有数据对象直接导入到资产设计人员，如指导规则或指导决策表设计人员。在项目中选择相关规则资产以及资产设计器，请转至 Data Objects → New item 以选择要导入的对象。

4. 要使数据对象持久化，请选择"永久 "复选框。Persistable 数据对象可以根据 JPA 规格存储在数据库中。默认的 JPA 为 Hibernate。
5. 点 确定。
6. 在数据对象设计器中，点 add 字段在 对象中添加包含属性 Id、标签和 类型的字段。所需的属性标有星号(*)。

- id： 输入字段的唯一 ID。

- **label:** （可选）输入字段的标签。
- **Type：** 输入字段的数据类型。
- **列表 ：** （可选）选择此复选框，以启用字段保存指定类型的多个项目。

图 26.1. 在数据对象中添加数据字段



7. 点 Create 添加新字段，或者点击 Create 并继续添加新字段并继续添加其他字段。

注意

要编辑字段，请选择字段行，并使用屏幕右侧的 常规属性。

第 27 章 创建指导的决策表

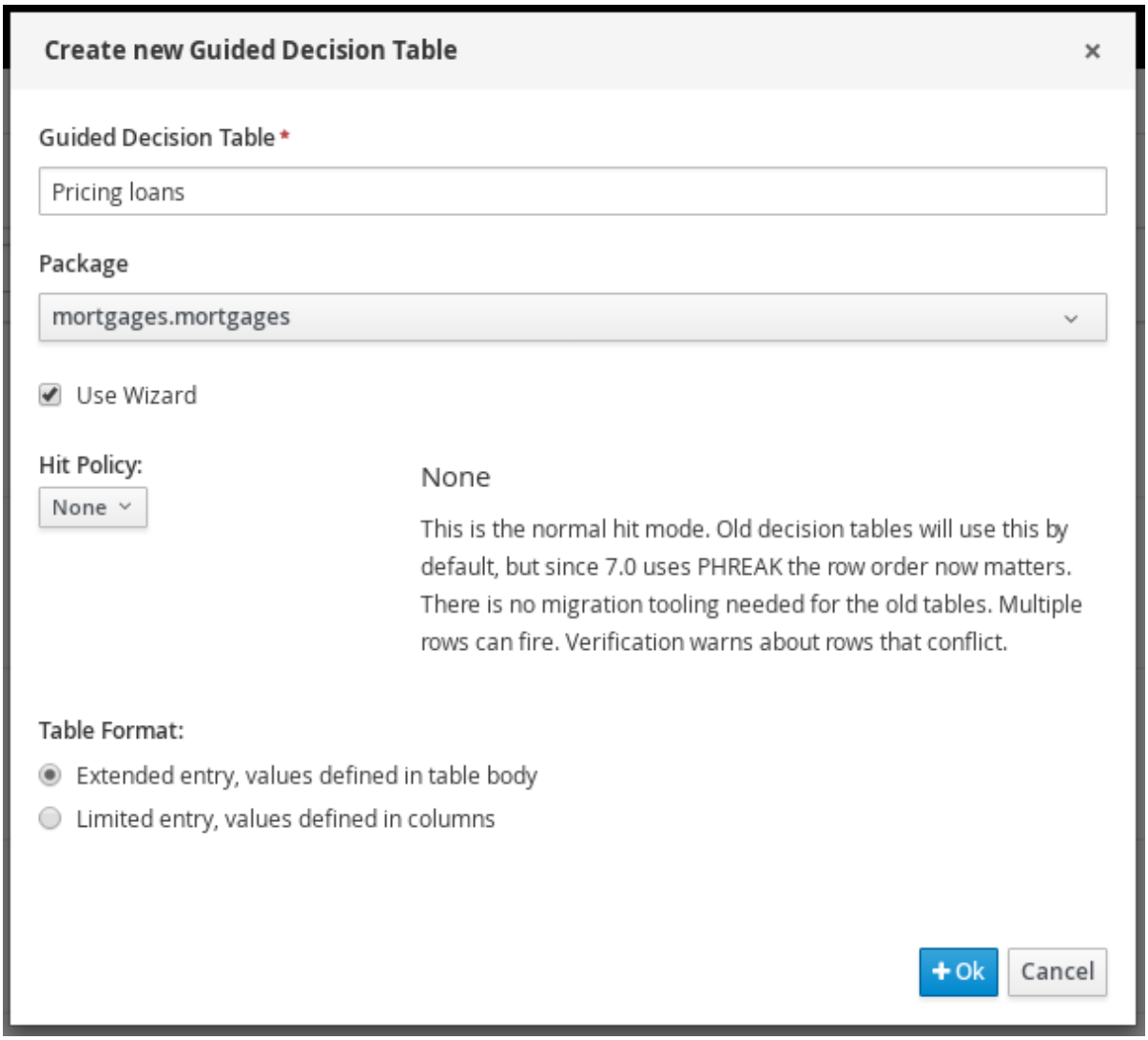
您可以使用指导的决策表，以可添加至您的业务规则项目中的表格格式定义规则属性、元数据、条件和操作。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Guided Decision Table。
3. 输入信息 式指南决策表 名称并选择相应的 软件包。您指定的软件包必须是分配或将被分配所需数据对象的同一软件包。
4. 选择 Use Wizard 完成向导中的表设置，或者保留此选项未选择来完成创建表，并在的指导决策表设计器中指定剩余的配置。
5. 选择您希望表中规则行的点击策略以符合。详情请查看 第 28 章 指导决策表的点击策略。
6. 指定您是否希望 扩展条目 或 有限条目 表。详情请查看 第 28.1.1 节 “指导决策表的类型”。
7. 单击确定以完成 设置。如果您选择了 Use Wizard，则会显示 Guided Decision Table 向导。如果您没有选择 Use Wizard 选项，则不会显示此提示，而是直接进入表设计程序。

   例如，以下向导设置适用于 loan Application decision 服务中的指导决策表：

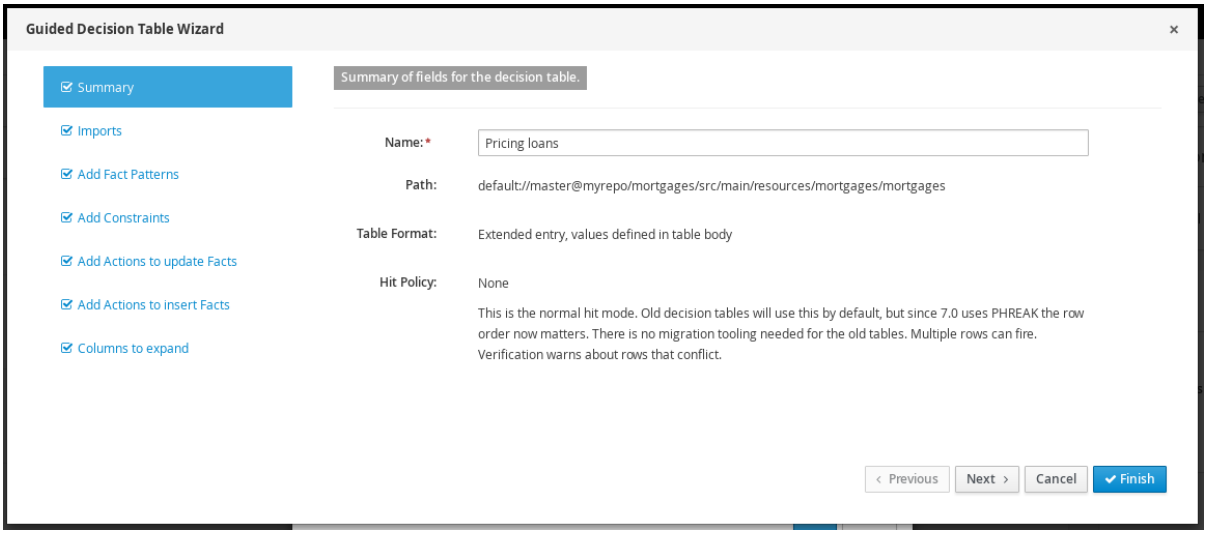
图 27.1. 创建指导决策表



8.

如果您使用向导，请添加任何可用的导入、事实模式、约束和操作，并选择表列是否应该扩展。点 Finish 关闭向导并查看表设计器。

图 27.2. Guided Decision Table 向导



在指导的决策表设计器中，您可以添加或编辑列和行，并做其他最终调整。

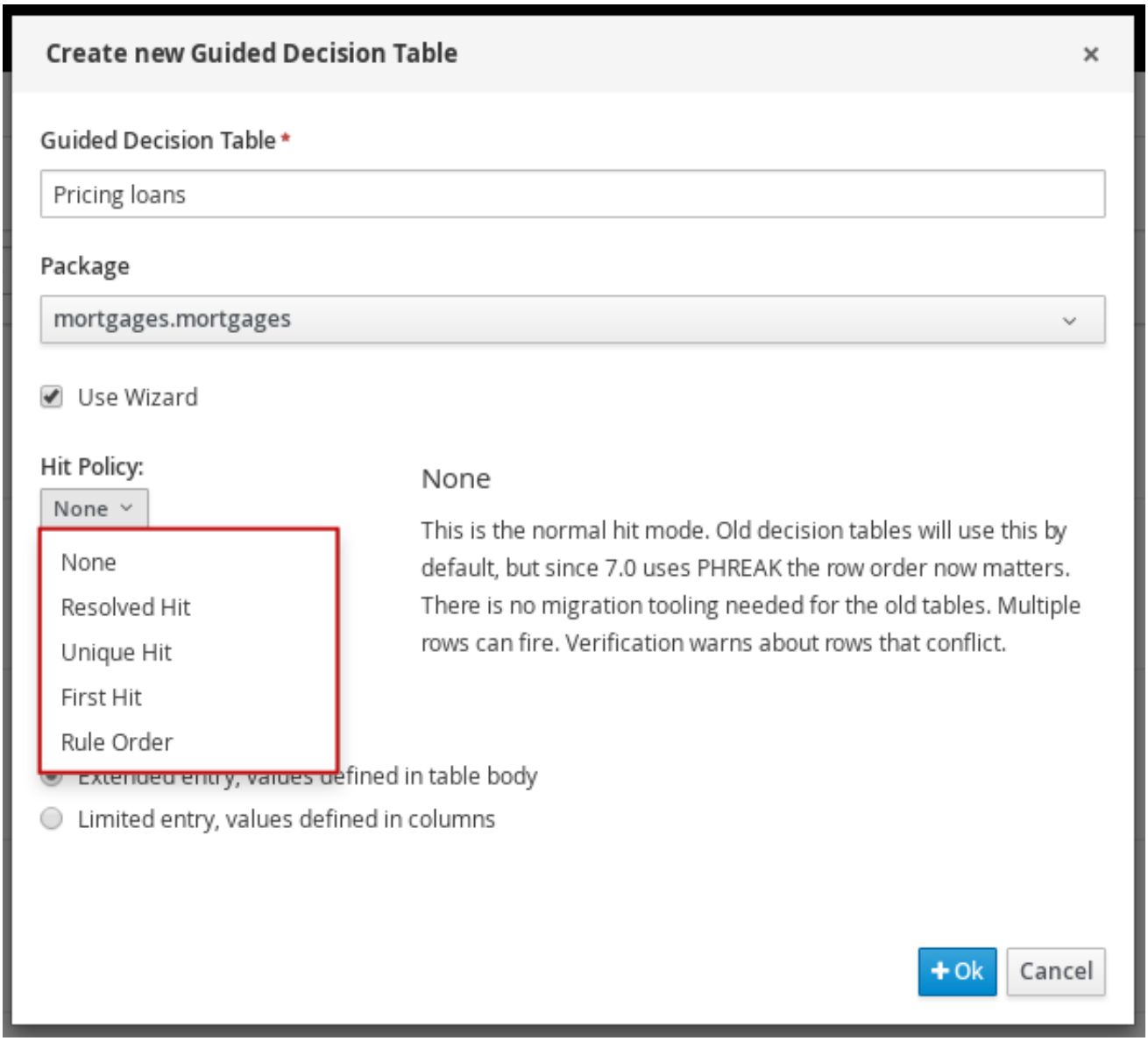
第 28 章 指导决策表的点击策略

按策略决定了对指导决策表中的规则(row)的顺序、按指定优先级、根据指定的优先级还是其他选项。

可用的点击策略如下：

- none：（默认点击策略）可以执行多行，验证警告有关冲突的行。已上传的任何决策表（使用非指南的表格表格）都将采用这种点击策略。
- 解析 Hit：根据指定优先级，每次只能执行一行，无论列表顺序如何（例如，您可以给出行 10 的优先级超过行 5）。这意味着您可以保持您想要的视觉可读性的行的顺序，但指定优先级例外。
- 唯一的 Hit：每次只能执行一行，每行必须是唯一的，且未满足条件重叠。如果执行多个行，则验证会在开发时会发出警告。
- First Hit：按 表中列出的顺序（从上到下），一次只能执行一行。
- 规则顺序：可以执行多个行，验证不能报告行之间的冲突，因为它们应该发生。

图 28.1. 可用点击策略



28.1. 按策略示例：针对电影问题单的折扣的决策表

以下是一个示例决策表，这些表格根据客户的年龄、学位或军事状态提供折扣。

表 28.1. 针对movie ticket 的可用折扣的决定表示例

行号	折扣类型	折扣
1	高级公民(age 60+)	10%
2	student	10%
3	军事	10%

在这个示例中，最后应用的总折扣会根据为表指定的点击策略而有所不同：

- none/Rule Order： 如果采用 None 和 Rule Order hit 策略，所有适用规则均将被纳入。在这种情况下，允许为每个客户提供折扣。

  示例：一个由学生和军队老师组成的高级员工将享受全部 3 个折扣，总计 30%。

  主要区别：对于 None，则会为应用的多个行创建警告。使用 Rule Order 时，不会创建这些警告。

- 第一个 Hit/Resolved Hit： 在第一个 Hit 和 Resolved Hit 策略中，只能应用其中一个折扣。

  对于第一个 Hit，会应用列表中第一个满足的折扣，并会忽略其他项。

  示例：一个高级公民，也是学生，军队老员将只获得 10% 的高级员工折扣，因为这在表的最前面列出。

  对于解决问题的 Hit，需要修改的表。在表中为您分配一个优先级例外的折扣（不论列出的顺序）将首先应用。要分配这个例外，请包括一个新列，指定其他一个折扣（行）的优先级。

  示例：如果军事折扣比年龄或学生折扣更优先考虑，尽管有所列出的订单，那么高级消费者也是学生，军队退还只能享受 10% 的军用折扣，无论年龄或学生状态如何。

  请考虑以下修改的决定表，它适用于 解决 Hit 策略：

表 28.2. 符合 解决 Hit 策略的修改的决策表

行号	折扣类型	具有优先级 over Row	折扣
1	高级公民(age 60+)		10%
2	student		10%
3	军事	1	10%

在这个修改表中，军事折扣基本上是新行 1，因此每本期和学员折扣都享有优先权，并在以后添加的其它折扣。您不需要指定优先级超过 "1 和 2" 行，只有 行 "1"。这会使行点击顺序更改为 3 → 1 → 2 → … 等，如表增长。

注意

如果您实际将军用折扣移至第 1 行，则行顺序将同样改变，而是向该表应用第一个 Hit 策略。但是，如果您想要以某种方式列出的规则并有所不同，例如在字母顺序表中应用的规则，则 Resolved Hit 策略会很有用。

主要区别：在第一个 Hit 的情况下，规则按列出的顺序严格应用。使用 Resolved Hit 时，除非指定了优先级例外，否则将按照列出的顺序应用规则。

- unique Hit：需要修改的表。使用 唯一 Hit 策略时，必须以一种方式创建行，以一次无法满足多个规则。但是，您仍然可以通过行键指定是否应用一条规则或多个规则。这样，您可以使用唯一的 Hit 策略来更精细，防止出现重叠警告。

请考虑以下修改的决定表，它符合 唯一的 Hit 策略：

表 28.3. 修订版决定表，该表格符合 唯一的 Hit 策略

行号	高级公司(age 65+)	是学员	为 Military	折扣
1	是	否	否	10%
2	否	是	否	10%
3	否	否	是	10%
4	是	是	否	20%
5	是	否	是	20%
6	否	是	是	20%
7	是	是	是	30%

在这个修改表中，每行都是唯一的，且没有重叠，任何单个折扣或任何折扣组合都符合。

28.1.1. 指导决策表的类型

Red Hat Process Automation Manager 中支持两种类型的决策表： Extended entry 和 Limited 条目表。

- *扩展条目： Extend Entry 决策表是列定义指定 Pattern、 Field 和 Operator 但没有值的。值或状态自身在决策表的正文中持有。*

Pricing loans

#	Description	application : LoanApplication				ome : IncomeSou	application		
		amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

- *有限条目： 除了 Pattern、 Field 和 Operator 外， 列定义还可指定值的一个有限条目。表状态（在表格正文中保存）是布尔值布尔值（标记为复选框）的布尔值无效， 表示应应用该列或匹配。负值（清除的复选框）意味着该列不适用。*

Credit rating

#	Description	CR = AA	CR = OK	CR = Sub prime	Approve	Decline
1		☑	☐	☐	☑	☐
2		☐	☑	☐	☑	☐
3		☐	☐	☑	☐	☑

第 29 章 在指导的表中添加列

在创建了指导的决策表后，您可以在指导的决策表设计器中定义和添加各种列。

先决条件

- 所有将用于列参数的数据对象（如 Facts 和 Fields）都已在找到了指导决策表的同一软件包中创建，或者已从另一个软件包导入到《指导决策表设计器的" 新建 "项目中。

有关这些列参数的描述，请查看 第 30 章 指导决策表中的列类型 中每种列类型的"必需列参数"片段。

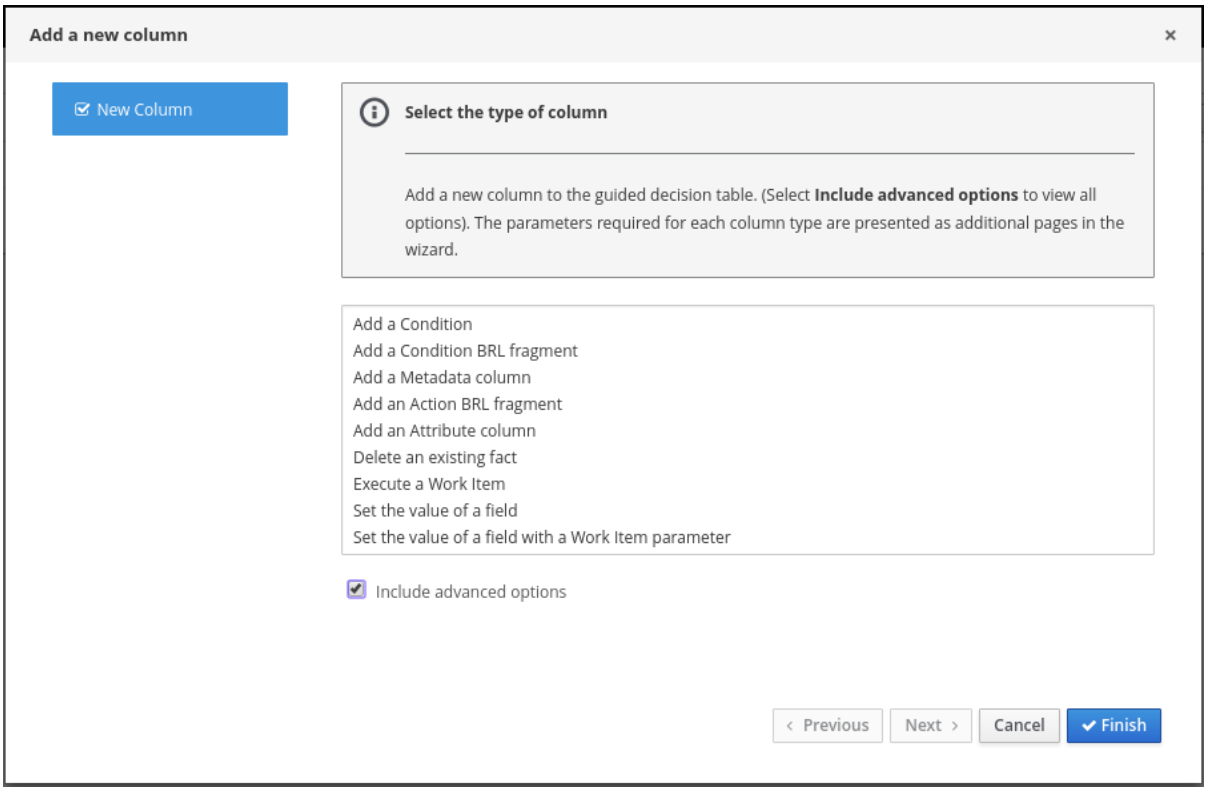
有关创建数据对象的详情，请参考 第 26.1 节 “创建数据对象”。

流程

1. 在指南的决策表设计器中，点击 Columns → Insert Column。

2. 点 Include advanced options 查看完整的列选项列表。

图 29.1. 添加列



3. 选择您要添加的列类型，单击 **Next**，然后按照向导中的步骤指定添加列所需的数据。

   有关设置的每个列类型和所需参数的描述，请参阅 第 30 章 指导决策表中的列类型。

4. 单击 **Finish** 以添加配置的列。

添加所有列后，您可以开始向列添加规则的行以完成决策表。详情请查看 第 34 章 在指导的表中添加行并定义规则。

以下是金级应用程序决策服务的示例决策表：

图 29.2. 完整的表格示例

Pricing loans									
#	Description	application : LoanApplication				ome : IncomeSou	application		
		amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

第 30 章 指导决策表中的列类型

为指导决策表添加一个新的列向导提供了以下列选项：（选择"包含高级选项"来查看所有选项。）

- 添加一个条件
- 添加条件 BRL 片段
- 添加元数据列
- 添加操作 BRL 片段
- 添加属性列
- 删除现有事实
- 执行 Work Item
- 设置字段的值
- 使用 Work Item 结果设置字段的值

以下几节中描述了这些列类型和每个 Add a new 列向导所需的参数。

重要信息：列参数所需的数据对象

本节中描述的部分列参数（如 Fact Patterns 和 Fields）提供仅由已经在找到指导决策表的同一软件包中定义的数据对象组成的下拉选项。软件包的可用数据对象列在 Project Explorer 的 Data Objects 面板中，并在引导决策表设计器的数据对象选项卡中列出。您可以根据需要在软件包中创建额外的数据对象，或者从指示决策表设计器的数据对象 → New item 中的另一个软件包导入它们。有关创建数据对象的详情，请参考 第 26.1 节 “创建数据对象”。

30.1. "添加条件"

条件代表规则左侧("WHEN")部分定义的 fact 模式。使用这个列选项，您可以定义一个或多个条件列，用于检查带有特定字段值的数据对象是否存在或不存在，并且影响规则的操作（"THEN"部分。您可以在 condition 表中为事实定义绑定，或者选择之前定义的绑定。您还可以选择对模式进行导航。

规则条件示例

```
when
  $i : IncomeSource( type == "Asset" ) // Binds the IncomeSource object to the $i variable
then
  ...
end
```

```
when
  not IncomeSource( type == "Asset" ) // Negates matching pattern
then
  ...
end
```

指定绑定后，您可以定义字段限制。如果使用相同的事实模式绑定定义两个或多个列，则字段约束会变得相同模式的复合字段限制。如果您为单个模型类定义了多个绑定，每个绑定都会在规则中成为单独的模型类("WHEN")侧。

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

- Pattern： 从您在表中已使用的事实模式列表中选择或创建新的事实模式。事实模式是软件包中可用数据对象的组合（请参阅所需数据 对象的信息 ）和您指定的模型类绑定。（示例： LoanApplication [application] 或 IncomeSource [income]，其中 bracketed 部分是绑定到给定事实类型的绑定。）
- 入口点： 定义事实模式的入口点（若适用）。入口点是聚合或流，如果指定，事实将进入决策引擎。（示例： 应用流、贡献度检查流）
- 计算类型： 选择以下计算类型之一：

- 字面值：单元格中的值将使用 operator 与字段进行比较。
- formula：将评估单元格中的表达式，然后将与 字段进行比较。
- predicate：不需要字段，表达式将评估为 true 或 false。

- 字段：从之前指定的事实模式中选择一个字段。字段选项在项目的 Data Objects 面板中定义。（示例：LoanApplication 事实类型中 的数量 或 长度 字段）
- binding （可选）：根据需要为之前选择的字段定义绑定。（例如：对于模式 LoanApplication [application] [application]、字段 数量 和 operator 等于，如果将绑定设置为 $amount，则结束条件将是 应用程序 : LoanAppplication($amount : amount == [value]).
- operator ：选择要应用到前面指定的事实模式和字段的 Operator。
- 值列表（可选）：输入值选项列表（以逗号和空格分开），以限制规则("WHEN")部分的表格输入数据。当定义此值列表时，该列的表单元格中将作为下拉列表提供值，用户只能从中选择一个选项。（示例：周一、周三、周五 以只指定以下三个选项）
- 默认值（可选）：选择之前定义的值选项之一作为新行自动出现在单元格中的默认值。如果没有指定默认值，则表单元格默认为空白。您还可以从项目 Explorer 的枚举 定义面板中列出的任何之前配置的数据枚举 的值中选择默认值。（您可以在 Menu → Design → Projects → [select project] → Add Asset → Enumeration 中创建 enumerations。）
- 标题（描述）：为列添加标头文本。
- 隐藏列：选择此项以隐藏列，或清除此列以显示列。

30.1.1. 在 condition 列单元格中插入 任何其他值

对于引导式练习中的简单条件列，如果设置了以下参数，您可以在 列中对表单元应用任何其他值：

- 条件列的计算类型已设置为 Literal 值。
- Operator 已设置为等同性 == 或不等质量 !=。

任何其他值可让为尚未在表中的规则中明确定义的任何其他字段值定义规则。在 DRL 源中，任何其他不在中。

用于任何其他条件的规则示例

```
when
  IncomeSource( type not in ("Asset", "Job") )
  ...
then
  ...
end
```

流程

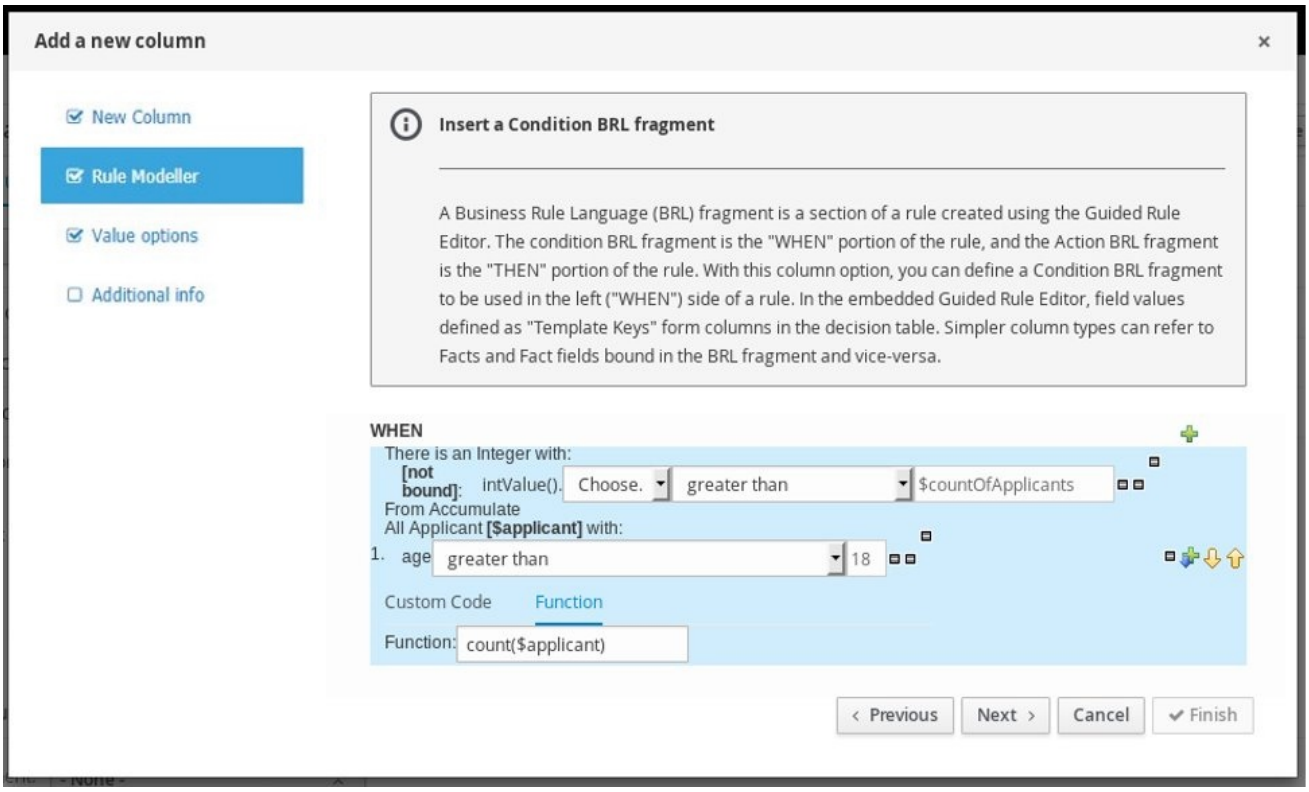
1. 选择使用 == 或 != 运算符的 condition 列的单元。
2. 在表设计器右上角，点击 Edit → Insert "any other" value。

30.2. "添加条件 BRL 片段"

业务规则语言(BRL)片段是使用指导规则设计器创建的规则的一个部分。条件 BRL 片段是规则的"WHEN"部分，操作 BRL 片段是规则的"THEN"部分。使用这个列选项，您可以定义一个规则左侧("WHEN")侧使用的条件 BRL 片段。更简单的栏类型可以参考 BRL 片段中绑定的事实和事实字段，反之亦然。

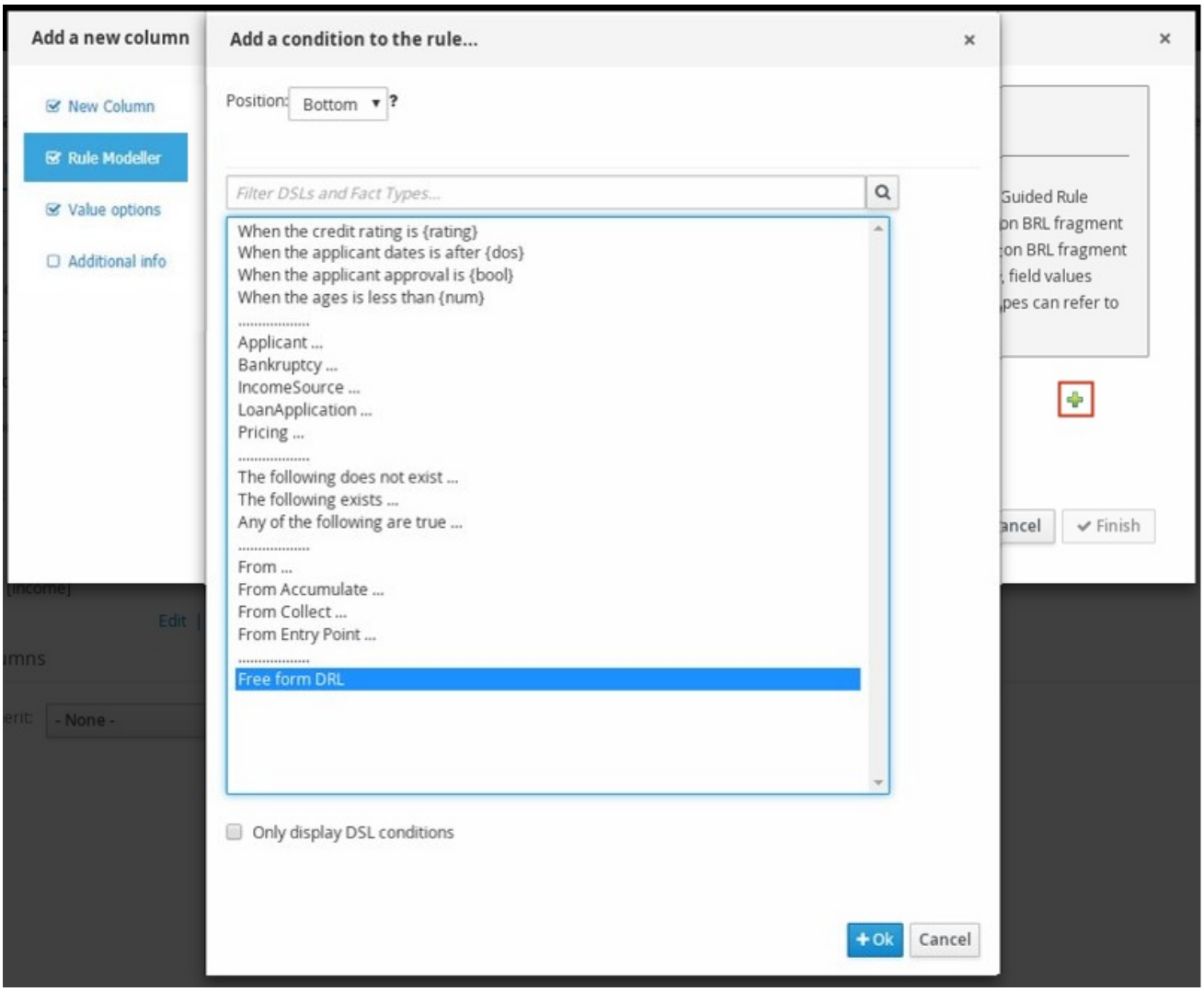
以下示例是 loan 应用程序的一个状况 BRL 片段：

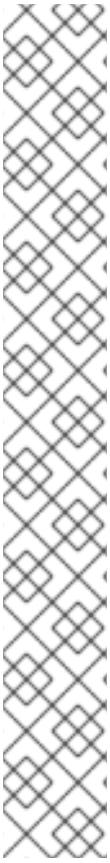
图 **30.1. 使用嵌入的指南**规则设计**器添加 BRL 片段**



您还可以从状况选项**列表中**选择 **Free form DRL，以定义条件 BRL 片段而无需嵌入的指南**规则设计**器。**

图 30.2. 使用免费形式 DRL 添加条件 BRL 片段





模板键

当您为条件 BRL 片段添加字段时，其中一个值是 Template 键 （而不是 Literal 或 Formula）。模板键是占位符变量，在生成指导的决策表时，会使用指定的值进行更改，并为每个指定模板键值的表中形成单独的列。您可以在 Value options 页面中指定 Template 键的默认值。在决策表中 Literal 和 Formula 值是静态的，但可以根据需要修改 Template 键值。

在嵌入式指南规则设计器中，您可以通过选择 Template key 字段选项并以 $key 格式在编辑器中输入值，将模板键值添加到字段中。例如，$age 在决策表中创建一个 $age 列。

在自由形式的 DRL 中，您可以使用 @{key} 格式为事实添加模板键值。例如，Person(age > @{age}) 在决策表中创建一个 $age 列。

使用模板键添加的新列的数据类型是 String。

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

- rule Modeller： 定义规则的 BRL 片段（"WHEN"部分）。
- 标题（描述）： 为列添加标头文本。
- 隐藏列： 选择此项以隐藏列，或清除此列以显示列。

30.3. "添加元数据列"

使用这个列选项，您可以在 decision 表中将 metadata 元素定义为列。每个列代表 DRL 规则中的普通元数据注解。默认情况下，metadata 列是隐藏的。要显示列，请在指导决策表设计器中点击 Edit Columns，再清除 Hide 列 复选框。

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

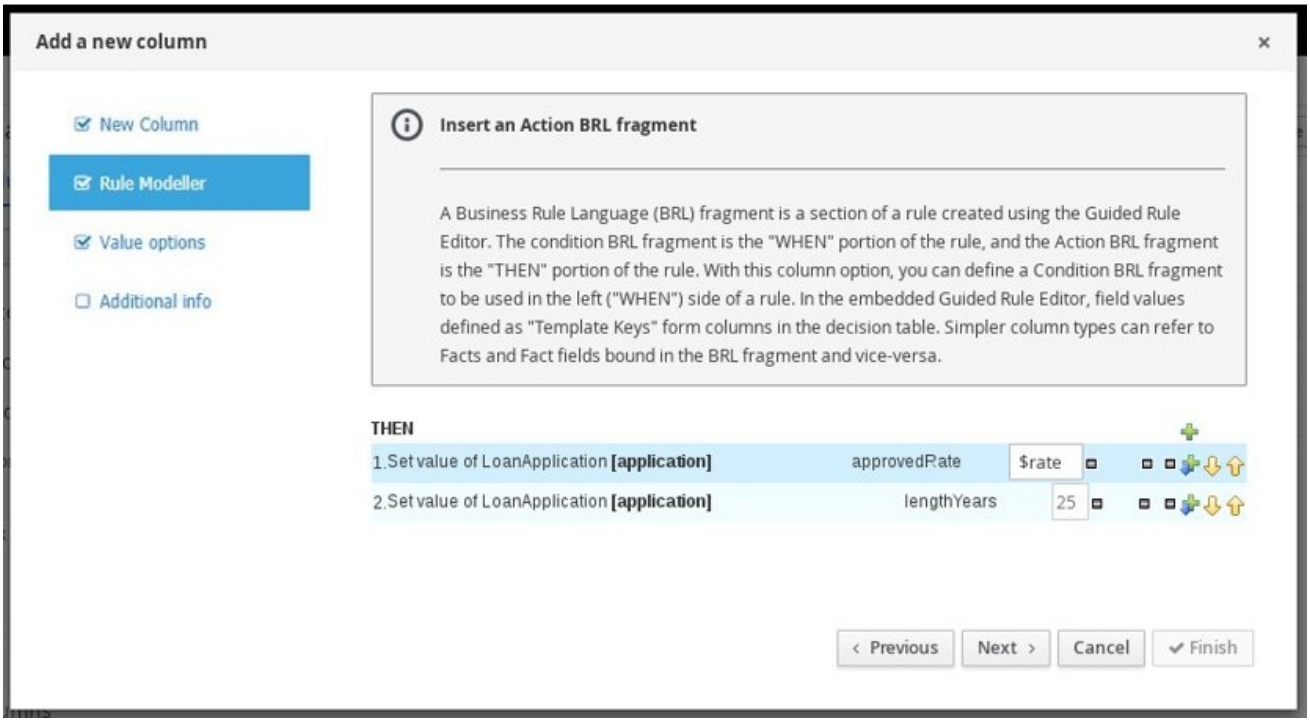
- **metadata ：以 Java 变量形式输入元数据项目的名称（即，它不能以数字或包含空格或特殊字符开头）。**

30.4. "添加操作 BRL 片段"

业务规则语言(BRL)片段是使用指导规则设计器创建的规则的一个部分。条件 BRL 片段 是规则的"WHEN"部分，操作 BRL 片段是规则的"THEN"部分。通过此列选项，您可以定义规则右侧("THEN")侧要使用的操作 BRL 片段。更简单的栏类型可以参考 BRL 片段中绑定的事实和事实字段，反之亦然。

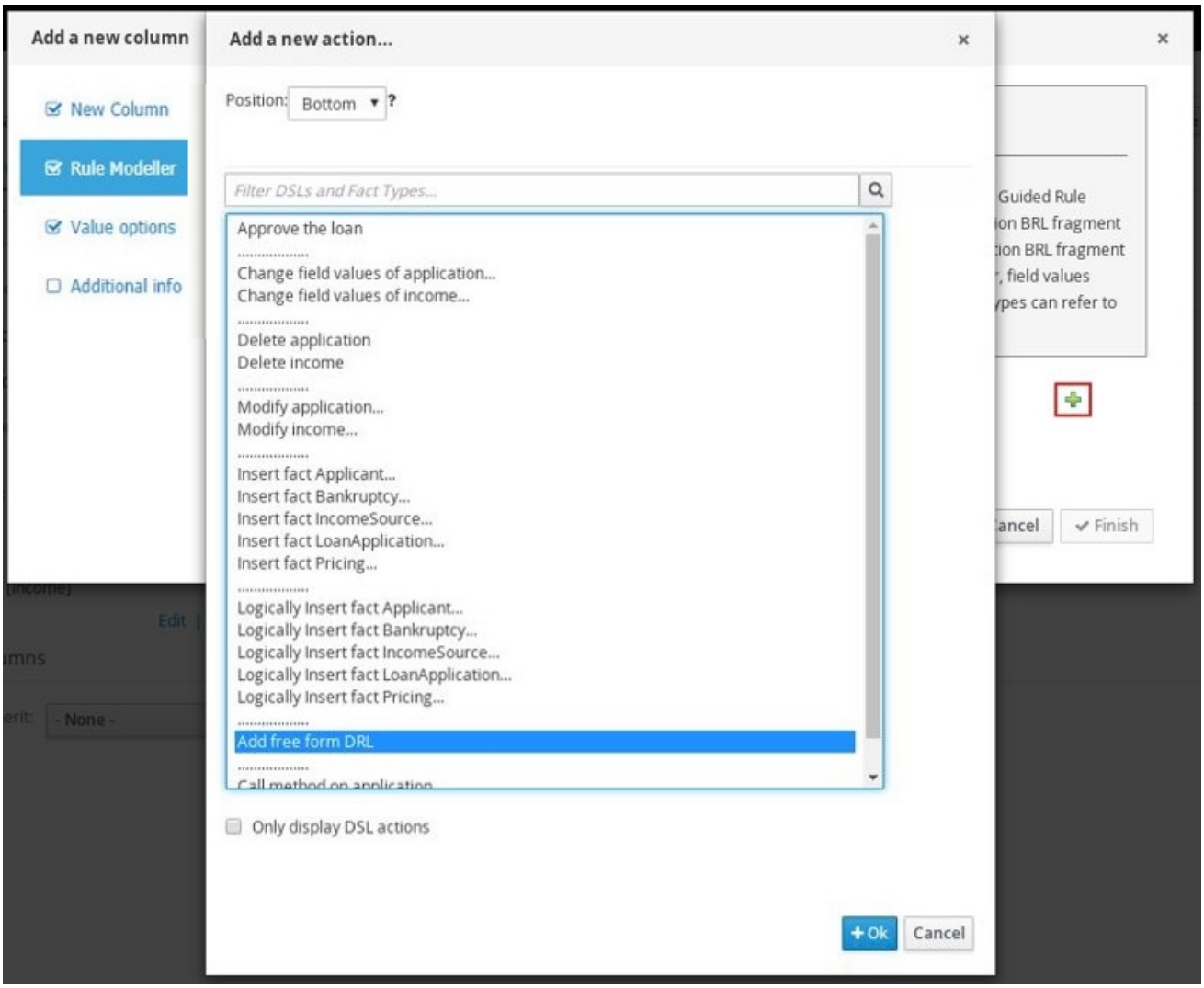
以下示例是 loan 应用程序的操作 BRL 片段：

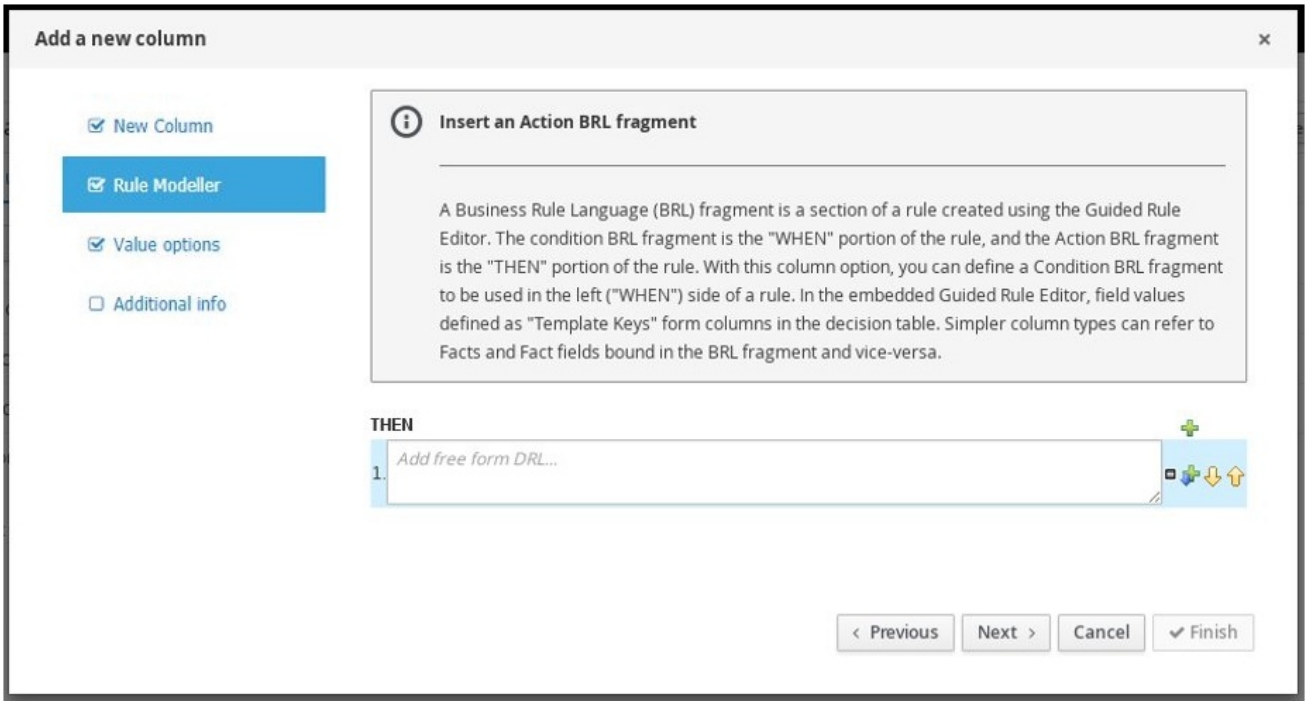
图 30.3. 使用嵌入式指导规则设计器添加操作 BRL 片段

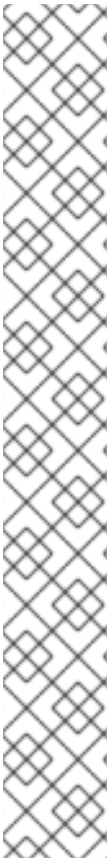


您还可以从操作选项列表中选择 Add free form DRL，以定义操作 BRL 片段，而无需嵌入的指南规则设计器。

图 30.4. 使用免费形式 DRL 添加操作 BRL 片段







模板键

当您为操作 BRL 片段添加字段时，其中一个值是 Template 键 （而不是 Literal 或 Formula）。模板键是占位符变量，在生成指导的决策表时，会使用指定的值进行更改，并为每个指定模板键值的表中形成单独的列。您可以在 Value options 页面中指定 Template 键的默认值。在决策表中 Literal 和 Formula 值是静态的，但可以根据需要修改 Template 键值。

在嵌入式指南规则设计器中，您可以通过选择 Template key 字段选项并以 $key 格式在编辑器中输入值，将模板键值添加到字段中。例如，$age 在决策表中创建一个 $age 列。

在自由形式的 DRL 中，您可以使用 @{key} 格式为事实添加模板键值。例如， Person(age > @{age}) 在决策表中创建一个 $age 列。

使用模板键添加的新列的数据类型是 String。

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

- 规则模型程序： 定义规则的 BRL 片段（"关键EN"部分）。
- 标题（描述）： 为列添加标头文本。
- 隐藏列： 选择此项以隐藏列，或清除此列以显示列。

30.5. "添加属性列"

使用这个列选项，您可以添加代表任何 DRL 规则属性的一个或多个属性列，如 Saliance、Enabled、Date-Effective 等。

例如，以下指导的决策表使用 salience 属性来指定规则优先级，以及为评估启用或禁用规则。首先评估具有较高 sali ence 值的规则，只有在选择了复选框时，才会评估带有 enabled 属性的规则。

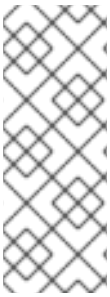
图 30.5. 带有 salience 和 已启用 属性的规则示例来定义评估行为

Pricing loans							
#	Description	salience	enabled	application : LoanApplication			
				amount min	amount max	period	deposit max
1		100	☑	131000	200000	30	20000
2			☑	10000	100000	20	2000
3			☐	100001	130000	20	3000

带有规则属性的规则源示例

```
rule "Row 1 Pricing loans"
  salience 100
  enabled true
  when
    ...
  then
    ...
end
...
rule "Row 3 Pricing loans"
  enabled false
  when
    ...
  then
    ...
end
```

有关每个属性的描述，请从向导的列表中选择属性。



按策略和属性

请注意，根据您为决策表定义的点击策略，一些属性可能会被禁用，因为它们由 hit 策略在内部使用。例如，如果您为这个表分配了 Resolved Hit 策略，以便根据表中指定的优先级顺序应用行（规则），那么 Salience 属性将已过时。其原因是，Salience 属性根据定义的健全值升级规则优先级，该值将被表中的 Resolved Hit 策略覆盖。

required Column Parameter

Add a new 列 向导需要以下参数来设置此列类型：

- attribute： 选择要应用到该列的属性。

30.6. "删除现有事实"

通过此列选项，您可以实施一个操作来删除之前作为表中添加的事实模式。创建此列时，该列的表单元格中以下拉列表提供事实类型，用户可从中选择一个选项。

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

- 标题（描述）： 为列添加标头文本。
- 隐藏列： 选择此项以隐藏列，或清除此列以显示列。

30.7. "执行 WORK ITEM"

使用这个列选项，您可以根据 Business Central 中的预定义工作项目定义来执行工作项目处理程序。（您可以在 Menu → Design → Projects → [select project] → Add Asset → Work Item 定义中 创建工作项。）

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

- work Item： 从预定义的工作项目列表中选择。
- 标题（描述）： 为列添加标头文本。
- 隐藏列： 选择此项以隐藏列，或清除此列以显示列。

30.8. "设置字段的值"

使用这个列选项，您可以实施一个操作，在之前绑定的事实上为规则的"THEN"部分设置字段值。您可

以选择通知修改值的决策引擎，这样可导致重新激活其他规则。

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

- Pattern： 从您的表中已使用的事实模式列表或状况 BRL 片段选择或创建新的事实模式。事实模式是软件包中可用数据对象的组合（请参阅所需数据 对象的信息 ）和您指定的模型类绑定。（示例： LoanApplication [application] 或 IncomeSource [income]，其中 bracketed 部分是绑定到给定事实类型的绑定。）

- 字段： 从之前指定的事实模式中选择一个字段。字段选项在项目的 Data Objects 面板中定义。（示例： LoanApplication 事实类型中 的数量 或 长度 字段）

- 值列表（可选）： 输入值选项列表（以逗号和空格分开），以限制规则中操作("THEN")部分的操作输入数据。当定义此值列表时，该列的表单元格中将作为下拉列表提供值，用户只能从中选择一个选项。（示例： 接受、拒绝、待处理）

- 默认值（可选）： 选择之前定义的值选项之一作为新行自动出现在单元格中的默认值。如果没有指定默认值，则表单元格默认为空白。您还可以从项目 Explorer 的枚举 定义面板中列出的任何之前配置的数据枚举 的值中选择默认值。（您可以在 Menu → Design → Projects → [select project] → Add Asset → Enumeration 中创建 enumerations。）

- 标题（描述）： 为列添加标头文本。

- 隐藏列： 选择此项以隐藏列，或清除此列以显示列。

- 逻辑插入： 当所选的事实模式当前没有在表格的另一个列中使用时显示此选项（请参阅下一个字段描述）。选择它以逻辑方式将事实模式插入到决策引擎中，或者清除它以定期插入它。决策引擎负责对插入和检索事实的逻辑决策。在常规或声明插入后，必须明确指定事实。逻辑插入后，当在第一个位置断言相关条件时，会自动重新处理事实，不再是 true。

- 更新引擎及更改： 此选项将显示在指导决策表的另一个列中。选择此项以使用修改后的字段值更新决策引擎，或明确此选项，不更新决策引擎。

30.9. "设置值为 WORK ITEM 结果的项的值"

使用这个列选项，您可以实施一个操作，将之前定义的 fact 字段的值设置为规则的"THEN"部分的工作项目处理程序的结果。工作项目必须定义与绑定事实上字段相同的数据类型结果参数，以便将该字段设置为 return 参数。（您可以在 Menu → Design → Projects → [select project] → Add Asset → Work Item 定义中 创建工作项。）

创建此列的表中必须已创建了 Execute a Work Item 列。

必需的列参数

Add a new 列 向导需要以下参数来设置此列类型：

- Pattern： 从您的表中已使用的事实模式列表中选择或创建新的事实模式。事实模式是软件包中可用数据对象的组合（请参阅所需数据 对象的信息 ）和您指定的模型类绑定。（示例： LoanApplication [application] 或 IncomeSource [income]，其中 bracketed 部分是绑定到给定事实类型的绑定。）
- 字段： 从之前指定的事实模式中选择一个字段。字段选项在项目的 Data Objects 面板中定义。（示例： LoanApplication 事实类型中 的数量 或 长度 字段）
- work Item： 从预定义的工作项目列表中选择。（工作项目必须定义与绑定事实上字段相同的数据类型结果参数，以便将该字段设置为 return 参数。）
- 标题（描述）： 为列添加标头文本。
- 隐藏列： 选择此项以隐藏列，或清除此列以显示列。
- 逻辑插入： 当所选的事实模式当前没有在表格的另一个列中使用时显示此选项（请参阅下一个字段描述）。选择它以逻辑方式将事实模式插入到决策引擎中，或者清除它以定期插入它。决策引擎负责对插入和检索事实的逻辑决策。在常规或声明插入后，必须明确指定事实。逻辑插入后，当在第一个位置断言相关条件时，会自动重新处理事实，不再是 true。
- 更新引擎及更改： 此选项将显示在指导决策表的另一个列中。选择此项以使用修改后的字段值更新决策引擎，或明确此选项，不更新决策引擎。

第 31 章 在指导的决策表中查看规则名称列

如果需要，您可以在指导决策表中查看 Rule Name 列。

流程

1. 在指导的决策表设计人员中，单击 Columns。
2. 选择 Show rule name 列 复选框。
3. 点 Finish 保存。

默认规则名称格式为 Row(row_number)(table_name)。如果没有指定规则名称，则 Source 包含默认值。在"指导的决策"表中，您可以在 Rule Name 列中添加规则名称并覆盖默认值。

第 32 章 在指导决策表中对列值进行排序

您可以在指导决策表中对值进行排序。

先决条件

- 您在指导决策表中创建了所需的列。

流程

1. 双击您要按升序排序的列标头。
2. 要将同一列的值按降序排列，请双击列标头。

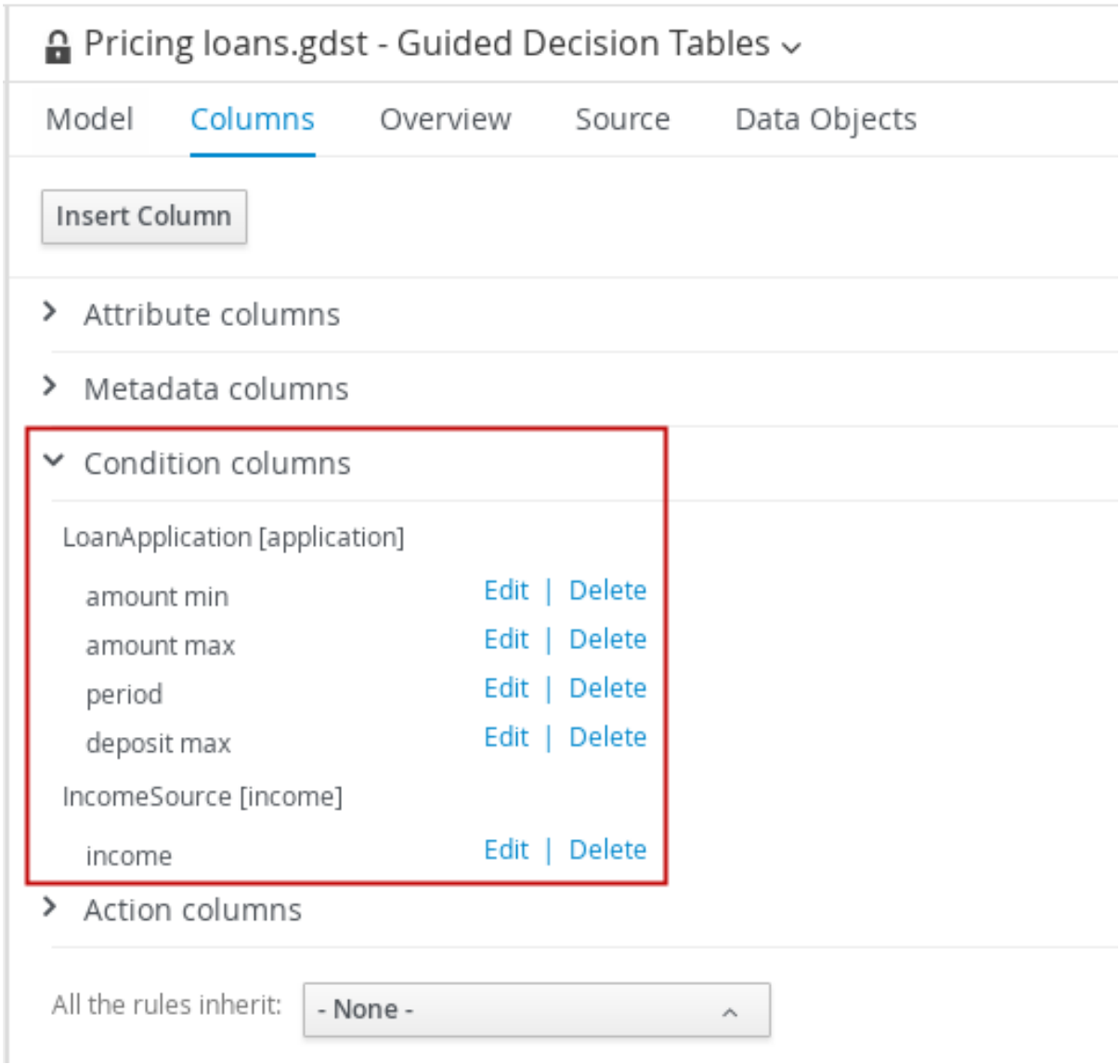
第 33 章 编辑或删除引导式练习中的列

您可以在指导的决策表设计器的任意时间编辑或删除您创建的列。

流程

1. 在指导的决策表设计人员中，单击 **Columns**。
2. 展开适当的部分，再单击列名称旁边的 **Edit** 或 **Delete**。

图 33.1. 编辑或删除列



注意

如果现有操作列使用与 condition 列相同的模式匹配参数，则无法删除条件列。

3. ***在任何列更改后，单击向导中的 Finish 以保存。***

第 34 章 在指导的表中添加行并定义规则

在指导决策表中创建了列后，您可以在指导的决策表设计人员中添加行并定义规则。

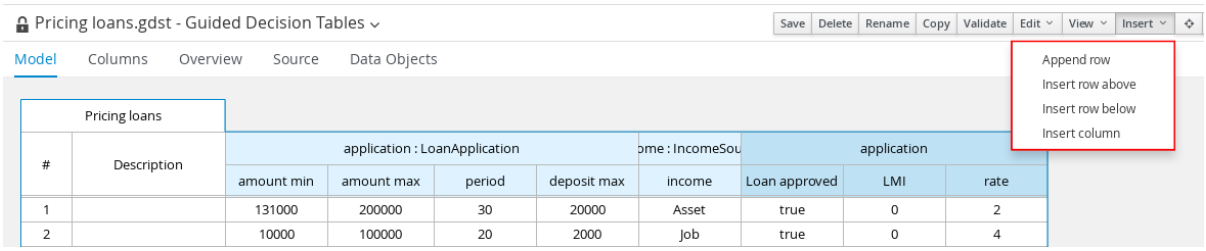
先决条件

- 按照 第 29 章 在指导的表中添加列 所述，添加了指导决策表的列。

流程

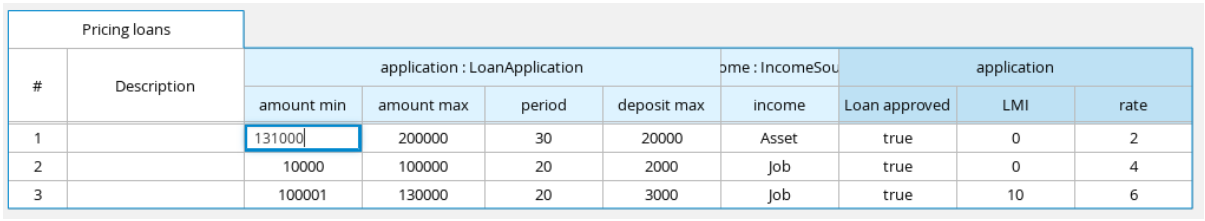
1. 在指南的决定表设计器中，点击 **Insert → Append 行 或 Insert 行 之一。（您也可以单击 Insert 列 以打开列向导并定义新列。）**

图 34.1. 添加行

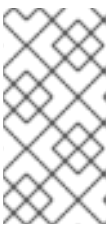


2. 双击每个单元并输入数据。对于带有指定的值的单元，请从单元下拉列表中选择。

图 34.2. 在每个单元中输入输入数据



3. 在指导决策表中定义所有数据行后，单击指导决策表设计器右上角的 **Validate** 以验证表。如果表验证失败，解决错误消息中描述的任何问题，查看表中的所有组件，然后重试验证表直到表通过为止。



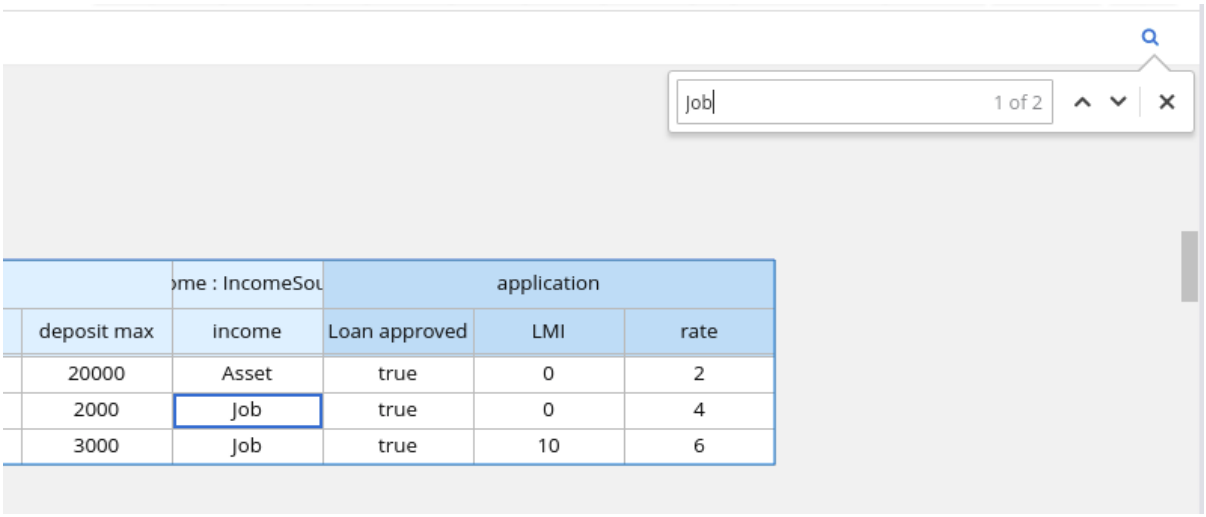
注意

虽然指导性决策表具有实时验证和验证，但您仍应手动验证完成的决策表以确保最佳结果。

4. 点表设计器中的 Save 保存您的更改。

在定义了指南的决定表内容后，在《指导决策表设计器右上角）上，如果需要搜索指南中显示的文本，则可以使用搜索栏。搜索功能在带有许多值的复杂指导决策表中特别有用：

图 34.3. 搜索指导决策表内容



第 35 章 在规则资产中定义下拉列表的枚举数

Business Central 中的枚举定义决定了指导规则、指导规则模板和指导决策表中的条件或操作字段的可能值。enumeration 定义包含一个 fact.field 映射到支持值列表，该列表显示为规则资产相关字段中的下拉列表。当用户选择基于与枚举定义相同的事实和字段的字段时，则会显示定义值的下拉列表。

您可以在 Business Central 中，或者在 Red Hat Process Automation Manager 项目的 DRL 源中定义枚举器。

流程

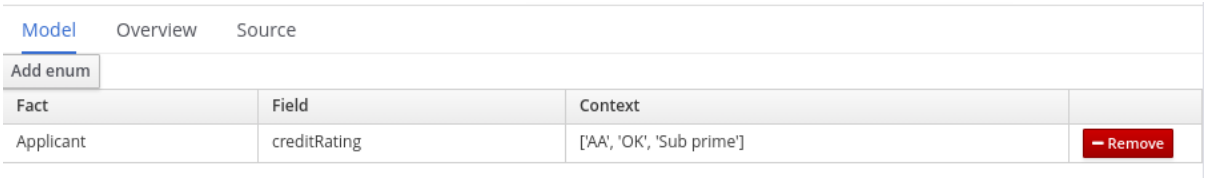
1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Enumeration。
3. 输入参考 名称 并选择相应的 软件包。您指定的软件包必须是分配或分配所需数据对象和相关规则资产的同一软件包。
4. 单击 Ok 以创建枚举。

   新的 enumeration 现已列在 Project Explorer 的 Enumeration Definitions 面板中。
5. 在 enumerations designer 的 Model 选项卡中，点 Add enum，并为枚举定义以下值：

   - 事实 ：指定您项目同一软件包中的现有数据对象，您要将此枚举关联。在 Project Explorer 中打开 Data Objects 面板，以查看可用的数据对象，或根据需要创建相关的数据对象作为新资产。
   - 字段 ：指定一个现有字段标识符，作为您为 事实 选择的 data 对象的一部分定义。在 Project Explorer 中打开 Data Objects 面板，以选择相关的数据对象并查看可用 标识符选项列表。如果需要，您可以为 data 对象创建相关标识符。
   - context ：指定一个值列表，格式为 ['string1','string2','string3'] 或 [integer1,integer2,integer3]，您想要映射到 Fact 和 Field 定义。这些值将显示为规则资产中相关字段的下拉列表。

例如，以下枚举值定义了 loan 应用决策服务中 applicant 贡献等级的下拉菜单：

图 35.1. Business Central 中申请信贷评级的枚举示例



DRL 源中应用贡献度评级的枚举示例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```

在本例中，对于任何指导规则、指导规则模板或指导决策表，这些表格位于项目的同一软件包中，并使用 Applicant 数据对象和 creditRating 字段，配置的值作为下拉列表提供：

图 35.2. 指导规则或指导规则模板中的 enumeration 下拉列表示例

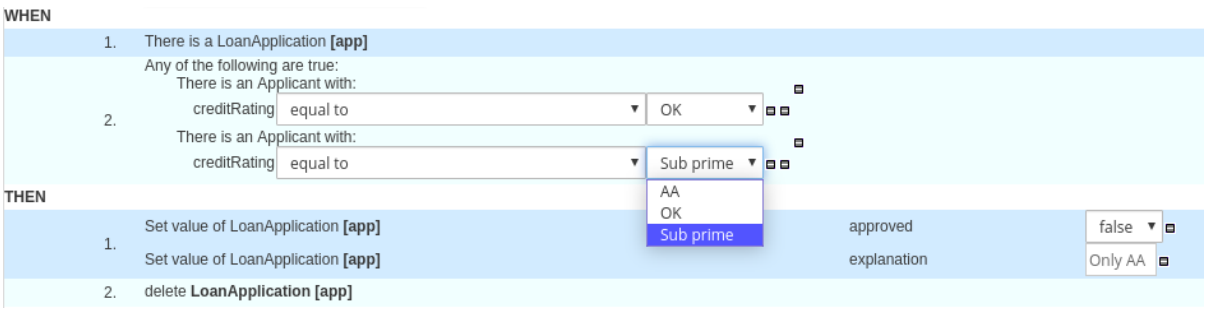
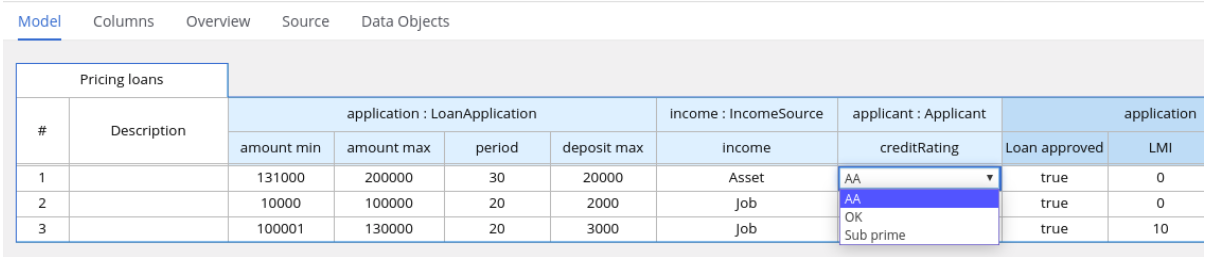


图 35.3. 指导决策表中的枚举下拉列表示例



35.1. 规则资产的高级枚举选项

对于红帽流程自动化管理器项目中的枚举定义的高级用例，请考虑以下扩展选项来定义枚举：

Business Central 中的 DRL 值和值之间的映射

如果您希望 enumeration 值显示在 Business Central 界面中的不同于 DRL 源中显示的不同或更全面的值，使用格式 "fact.field" : ['sourceValue1=UIValue1','sourceValue2=UIValue2','sourceValue2=UIValue2', ...] 用于您的 enumeration 值。

例如，在 loan status 的以下枚举定义中，选项 A 或 D 会在 DRL 文件中使用，但选项已批准或拒绝在 Business Central 中：

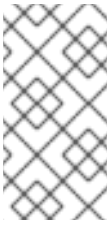
```
'Loan.status' : ['A=Approved','D=Declined']
```

枚举值依赖项

如果您希望在一个下拉列表中选择的值，以决定后续下拉列表中的可用选项，请使用 'fact.fieldB[fieldA=value1]' : ['value2', 'value3', 'value3', ...] 用于您的 enumeration 定义。

例如，在保险政策的以下枚举定义中，policyType 字段接受 Home 或 Car 的值。用户选择的策略类型决定了可用的策略 覆盖 字段选项：

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```



注意

在规则条件和操作中不应用枚举依赖关系。例如，在这个保险策略用例中，规则条件中的所选策略不会决定规则操作中的可用覆盖范围选项（如果适用）。

枚举中的外部数据源

如果要从外部数据源检索枚举值列表，而不是直接在枚举定义中定义值，请在项目的类路径上添加一个 helper 类，该类返回 java.util.List 字符串列表。在枚举定义中，而不是指定值列表，标识您配置为从外部检索值的帮助程序类。

例如，在 loan applicant 区域的以下枚举定义中，而不是以 "Applicant.region" 格式明确定义：['country1', 'country2', ...], enumeration 使用一个帮助类返回外部定义的值列表：

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

在本例中，DataHelper 类包含一个 getListOfRegions（） 方法，它返回字符串列表。在规则资产中相关字段的下拉列表中载入 enumerations。

您还可以将依赖的 enumeration 定义从 helper 类中动态加载，方法是将依赖字段标识为引号内帮助程序类的调用范围：

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}")'
```

如果要完全从外部数据源（如相关数据库）加载所有枚举的数据，则可实施返回 Map<String、List<String> 映射的 Java 类。映射的键是 fact.field 映射，值是值的 java.util.List<String> 列表。

例如，以下 Java 类为相关的枚举定义了 loan applicant 区域：

```
public class SampleDataSource {

  public Map<String, List<String>> loadData() {
    Map data = new HashMap();

    List d = new ArrayList();
    d.add("AU");
    d.add("DE");
    d.add("ES");
    d.add("UK");
    d.add("US");
    ...
    data.put("Applicant.region", d);

    return data;
  }

}
```

以下枚举定义与本例 Java 类相关。枚举不包含对事实或字段名称的引用，因为它们是在 Java 类中定义：

```
=(new SampleDataSource()).loadData()
```

= 运算符使 Business Central 能够加载 helper 类中的所有枚举数据。当请求枚举定义以便在编辑器中使用时，静态评估帮助程序方法。

注意

目前，Business Central 不支持在没有事实和字段定义的情况下定义枚举。要以这种方式为关联的 Java 类定义枚举值，请在 Red Hat Process Automation Manager 项目中使用 DRL 源。

第 36 章 实时验证和指导决策表验证

Business Central 为指导决策表提供实时验证和验证功能，以确保您的表格可以自由完成并出错。每个单元更改后都会验证引导的决策表。如果检测到逻辑中的问题，会出现一个错误通知并描述问题。

36.1. 指导决策表中的问题类型

验证和验证功能检测到以下类型的问题：

冗余

当决策表中的两个行执行同一组事实时，冗余就会发生。例如，在两天检查客户端的生日，并提供生日折扣可享受八五折优惠。

Subsumption

Subsumption 类似于冗余，并在两个规则执行同样结果时发生，但一个在另一事实子集中执行。例如，请考虑以下规则：

- 当 Person age > 10 then increase Counter
- 当 Person age > 20 然后增加 Counter

在这种情况下，如果某个人旧有 15 年，则只有一个规则触发，如果一个人是 20 年，则这两个规则都触发。这样的情况在运行时作为冗余造成类似问题。

Conflicts

当两个类似条件有不同的后果时，会出现冲突情况。在下表中的两个行（规则）或两个单元格之间可能会出现冲突。

以下示例演示了在决定表中两个行之间的冲突：

- 当 Deposit > 20000 then Approve Loan
- 当 Deposit > 20000 then Refuse Loan

在这种情况下，无法知道 loan款将获得批准或非批准。

以下示例演示了在决定表中的两个单元间的冲突：

- 当 Age > 25
- Age < 25 年时

含有冲突单元的行从不执行。

唯一唯一策略 划分

当将 unique Hit 策略应用到决策表中时，一次只能执行一行，并且每行必须是唯一的，且未满足条件重叠。如果执行多个行，则验证报告将标识 broken hit 策略。例如，请在表中考虑以下条件，以确定价格折扣：

- 当为 Student 用户 = true 时
- Is Military = true

如果客户既是学员还是军事区，则适用这两个条件并打破 唯一的 Hit 策略。因此，在这种表中创建的行必须采用一种方式创建，不允许多个规则同时触发。有关点击策略的详情，请参考 第 28 章 指导决策表的点击策略。

deficiy

认为与冲突类似，因此决定表中的规则逻辑不完整。例如，请考虑以下规则：

- 当 Age > 20 然后 Approve Loan
- 当 Deposit < 20000 then Refuse Loan

对于超过 20 年以上的人，这两个规则可能会引起混淆，并且存款不足 20000。您可以添加更多

约束以避免冲突。

缺少 Columns

当删除的列会导致不完整或不正确的逻辑时，规则无法正确触发。这会检测到，以便您可以处理缺少的列，或者调整逻辑使其不依赖于意外删除的条件或操作。

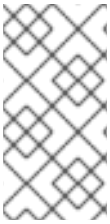
不完整的范围

如果表包含与可能字段值的限制，但没有定义所有可能的值，则字段值的范围不完整。验证报告标识所提供的任何不完整范围。例如，如果您的表检查某个应用程序是否已批准，验证报告会提醒您确保同时处理没有批准应用程序的情况。

36.2. 通知类型

验证和验证功能使用三种类型的通知：

- 错误：一个严重的问题，可能会导致指导决策表无法按照运行时设计工作。例如，冲突报告为错误。
- 警告：一个严重的问题，可能不会阻止指导决策表正常工作，但需要注意。例如，psumptions 报告为警告。
- 信息 ：可能不会阻止指导决策表正常工作但需要注意的中等问题。例如，缺少列报告为信息。



注意

Business Central 验证和验证不会妨碍您保存不正确的更改。这个功能只会在编辑时报告问题，您仍然可以继续忽略这些问题并保存您的更改。

36.3. 禁用指导决策表的验证和验证

Business Central 的决策表验证和验证功能默认启用。这个功能可帮助您验证您的指导的决定表，但使用复杂指导的决策表，此功能可以隐藏决策引擎性能。您可以通过在 Red Hat Process Automation

Manager 发行版中将 org.kie.verification.disable-dtable-realtime-verification 系统属性值设为 true 来禁用此功能。

流程

1. *导航到 ~/standalone-full.xml 并添加以下系统属性：*

```
<property name="org.kie.verification.disable-dtable-realtime-verification" value="true"/>
```

例如，在红帽 JBoss EAP 上，您可以在 $EAP_HOME/standalone/configuration/standalone-full.xml 中添加这个系统属性。

第 37 章 将指南的 DECISIONS 表转换为电子表格决策表

在 Business Central 中定义了指导决策表后，您可以将指导决策表转换为 XLS 表格决策表文件，用于离线参考和文件共享。指导的决策表必须是扩展条目指导决策表，才能转换。转换工具不支持有限的条目指导表。

有关电子表格决策表的更多信息，请参阅使用电子表格决策表设计决策服务。

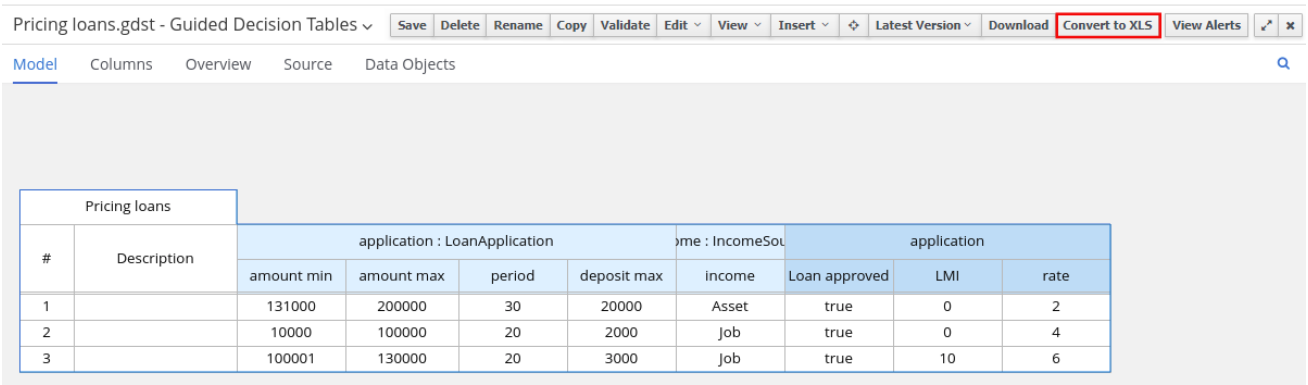
警告

指导的决策表和电子表格表格是支持不同功能的表格格式。在将两个决策表格式（例如，例如）之间不同的支持功能都会被修改或丢失，当您将一个决策表格式转换为另一个字段时，都会修改或丢失。

流程

在 Business Central 中，进入您想要在决策表设计器右上角转换的指南决策表资产，点 Convert to XLS ：

图 37.1. 转换上传的决定表



转换后，您可以在项目中作为电子表格决策资产提供转换，您可以下载这些表格以便离线参考。

第 38 章 执行规则

在确定了示例规则或在 Business Central 中创建自己的规则后，您可以在本地或 KIE 服务器上构建并部署相关的项目并在 KIE 服务器上执行规则，以测试规则。

先决条件

- 业务中心和 KIE 服务器已安装并运行。有关安装选项，请参阅 规划 Red Hat Process Automation Manager 安装。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 在项目 资产 页面右上角，单击 Deploy 以构建该项目，并将它部署到 KIE 服务器。如果构建失败，请解决屏幕底部的 Alerts 面板中描述的问题。

有关项目部署选项的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。

注意

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

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

默认情况下，Red Hat Process Automation Manager 中的规则资产需要这个依赖项，从可执行规则模型构建。这个依赖关系包含在 Red Hat Process Automation Manager 核心打包中，但取决于 Red Hat Process Automation Manager 升级历史记录，您可能需要手动添加这个依赖关系来启用可执行规则模型行为。

有关可执行规则模型的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。

3. 在 Business Central 之外创建 Maven 或 Java 项目（如果尚未创建），您可将其用于在本地执行规则，或用作在 KIE 服务器上执行规则的客户端应用程序。该项目必须包含 pom.xml 文件以及执行项目资源的任何其他必要组件。

   有关 test 项目示例，请参阅 "创建和执行 DRL 规则的方法 "。

4. 打开 test 项目或客户端应用程序的 pom.xml 文件，并添加以下依赖项（如果还没有添加）：

   - kie-ci ：启用客户端应用程序以使用 ReleaseId在本地加载 Business Central 项目数据
   - kie-server-client ：使您的客户端应用程序能够与 KIE 服务器上的资产远程交互
   - slf4j:（可选）使您的客户端应用程序能够使用 Simple Logging Facade 用于 Java(SLF4J)，以在与 KIE 服务器交互后返回调试信息

   客户端应用程序 pom.xml 文件中的 Red Hat Process Automation Manager 7.13 的依赖项示例：

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

   对于这些工件的可用版本，请在 Nexus Repository Manager 在线搜索组 ID 和工件 ID。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

5. 确保在客户端应用 pom.xml 文件中定义了包含模型类的工件依赖项，它们与部署的项目的 pom.xml 文件中完全相同。如果模型类的依赖关系因客户端应用和项目之间有所不同，则可能会出现执行错误。

要访问 Business Central 中的项目 pom.xml 文件，请在项目左侧选择任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

例如，以下 Person 类依赖项同时出现在客户端和部署的项目 pom.xml 文件中：

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. 如果您将 slf4j 依赖项添加到用于调试日志的客户端应用程序 pom.xml 文件，请在相关类路径（例如，在 Maven 中的 src/main/resources/META-INF）上创建一个 simplelogger.properties 文件（例如，在 Maven 中的 src/main/resources/META-INF）：

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

-

7. 在您的客户端应用中，创建一个包含必要导入和 main（） 方法的 .java 主类，以加载 KIE 基础、插入事实和执行规则。

例如，项目中的 Person 对象包含 getter 和 setter 方法，用于设置并检索一个人的名字、姓氏、每小时速率和分流。项目中的以下 Wage 规则计算 wage 和 hourly 速率值，并根据结果显示消息：

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

要在 KIE 服务器之外测试此规则（如果需要），请配置 .java 类以导入 KIE 服务、KIE 容器和 KIE 会话，然后使用 main（） 方法针对定义的事实模型触发所有规则：

本地执行规则

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

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

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

      // Set up the fact model:
      Person p = new Person();
```

```
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert the person into the session:
        kSession.insert(p);

        // Fire all rules:
        kSession.fireAllRules();
        kSession.dispose();
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
  }
```

要在 KIE 服务器上测试此规则，请配置 .java 类，其导入和规则执行信息与本地示例类似，另外还指定 KIE 服务配置和 KIE 服务客户端详情：

在 KIE 服务器上执行规则

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {
```

```
  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert Person into the session:
      KieCommands kieCommands = KieServices.Factory.get().getCommands();
      List<Command> commandList = new ArrayList<Command>();
      commandList.add(kieCommands.newInsert(p, "personReturnId"));

      // Fire all rules:
      commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
      BatchExecutionCommand batch =
kieCommands.newBatchExecution(commandList, sessionName);

      // Use rule services client to send request:
      RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
      ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
      System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

8. 从项目目录中运行配置的 .java 类。您可以在开发平台（如 Red Hat CodeReady Studio）或命令行中运行该文件。

Maven 执行示例（在项目目录中）：

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

Java 执行示例（在项目目录中）

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

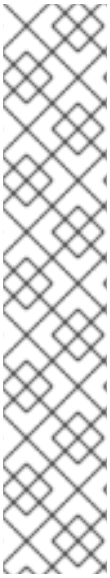
9. 在命令行中检查规则执行状态并在服务器日志中。如果有任何规则没有按预期执行，请检查项目中配置的规则，以及验证提供的数据的主类配置。

第 39 章 后续步骤

- 使用测试场景测试决策服务
- 打包和部署 Red Hat Process Automation Manager 项目

部分 V. 使用电子表格决策表设计决策服务

作为业务分析员或业务规则开发人员，您可以在电子表格表格的表格格式中定义业务规则，然后将电子表格上传到 Business Central 中的项目。这些规则编译到 Drools Rule Language(DRL)中，并形成您项目的决策服务的核心。



注意

您还可以使用决策模型和 Notation(DMN)模型设计您的决策服务，而不是基于规则或基于表的资产。有关 Red Hat Process Automation Manager 7.13 中的 DMN 支持的详情，请查看以下资源：

- 开始使用决策服务 （逐步教程，带有 DMN 决策服务示例）
- 使用 DMN 模型设计决策服务 (Red Hat Process Automation Manager 中的 DMN 支持和功能视图)

先决条件

- Business Central 中创建了决策表的空间和项目。每个资产都与分配给一个空间的项目相关联。详情请参阅 开始使用决策服务。

第 40 章 红帽流程自动化管理器中的决策资产

Red Hat Process Automation Manager 支持一些可用于为决策服务定义业务决策的资产。每个决策资产都有不同的优势，您可以根据目标和需求，首选使用一个或多个资产的组合。

下表重点介绍红帽流程自动化管理器项目支持的主要决策资产，以帮助您决定或确认在决策服务中定义决策的最佳方法。

表 40.1. Red Hat Process Automation Manager 支持的决策资产

asset	亮点	编写工具	Documentation
决策模型和符号 (DMN)型号	• 决策模型基于由对象管理组 (OMG)定义的符号标准。 • 使用代表部分或所有决策要求图 (DRG)的图形化决策要求图 (DRG)跟踪业务决策流程 • 使用 XML 模式，允许在 DMN 兼容平台间共享 DMN 模型 • 支持 Friendly Enough Expression Language(FEEL)，在 DMN 决策表中定义决策逻辑，以及其他 DMN 框的表达式 • 可以与业务流程模型和符号 (BPMN)流程模型有效集成 • 是创建全面、清晰和稳定的决策流程的最佳选择	Business Central 或其他与 DMN 兼容的编辑器	*使用 DMN 模型设计决策服务*
指导的决定表	• 是您在 Business Central 中基于 UI 的表格设计者创建的规则表 • 是电子表格决策表的向导替代选择 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值 • 支持点击策略、实时验证和其他资产不支持的其他功能 • 是以受控表格格式创建规则的最佳选择，可最小化编译错误	Business Central	*使用指导决策表设计决策服务*

asset	亮点	编写工具	Documentation
电子表格决策表	• 是 XLS 或 XLSX 表格决策表，您可以上传到 Business Central • 支持创建规则模板的模板键和值 • 是创建已在 Business Central 之外管理的决策表中规则的最佳选择 • 在上传后，对规则进行编译具有严格的语法要求	电子表格编辑器	*使用电子表格决策表设计决策服务*
指导规则	• 是您在 Business Central 中基于 UI 的规则设计者创建的单独规则 • 为可接受的输入提供字段和选项 • 是以受控格式创建单一规则的最佳选择，以最小化编译错误	Business Central	*使用指导规则设计决策服务*
指导规则模板	• 是您在 Business Central 中基于 UI 的模板设计者创建的可重复使用的规则结构 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值（与此资产的目的不相同） • 是创建具有相同规则结构但具有不同定义字段值的很多规则的最佳	Business Central	*使用指导规则模板设计决策服务*
DRL 规则	• 是直接在 **.drl** 文本文件中定义的单独规则 • 提供定义规则和其他规则行为的最大的灵活性 • 可以在特定的独立环境中创建，并与 Red Hat Process Automation Manager 集成 • 是创建需要高级 DRL 选项的规则的最佳选择 • 对规则进行正确编译的规则具有严格的语法要求	业务中心或集成开发环境(IDE)	*使用 DRL 规则设计决策服务*

asset	亮点	编写工具	Documentation
预测模型标记语言 (PMML)型号	• 是基于数据 Mining Group(DMG) 定义的标记标准预测数据分析模型。 • 使用允许 PMML 模式在 PMML 兼容的平台之间共享的 XML 模式 • 支持 Regression、scorecard、Tree、Mining 和其他模型类型 • 可以包含在独立 Red Hat Process Automation Manager 项目中，或者导入到 Business Central 中的项目 • 在 Red Hat Process Automation Manager 中，将预测数据合并为决策服务的最佳选择	PMML 或 XML 编辑器	*使用 PMML 模型设计决策服务*

在定义决策时，您还可以考虑将红帽构建的 Kogito 构建用于云原生决策服务。有关使用红帽构建的 Kogito 微服务的更多信息，请参阅 Red Hat Process Automation Manager 中的 Red Hat build of Kogito。

第 41 章 电子表格决策表

电子表格决策表是 XLS 或 XLSX 电子表格，其中包含以表格格式定义的业务规则。您可以包括带有独立红帽流程自动化管理器项目的电子表格决策表，或者在 Business Central 中将其上传到项目。决策表中的每一行都是一个规则，每个列都是条件、操作或其他规则属性。创建并上传电子表格表格后，您定义的规则会与所有其他规则资产一样编译为 Drools Rule Language(DRL)规则。

与电子表格表格表格表格表格表格表格表格表格相同的项目软件包中，所有与电子表格相关的数据对象都必须位于。默认导入同一软件包中的资产。可使用决策表导入其他软件包中的现有资产。

第 42 章 数据对象

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

42.1. 创建数据对象

以下流程是创建数据对象的一般概述。它并不适用于特定的业务资产。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Data Object。
3. 输入唯一数据对象名称，然后选择您希望数据对象可用于其他规则资产的软件包。 同一名称的数据对象不能位于同一软件包中。在指定的 DRL 文件中，您可以从任何软件包中导入数据对象。

从其他软件包导入数据对象

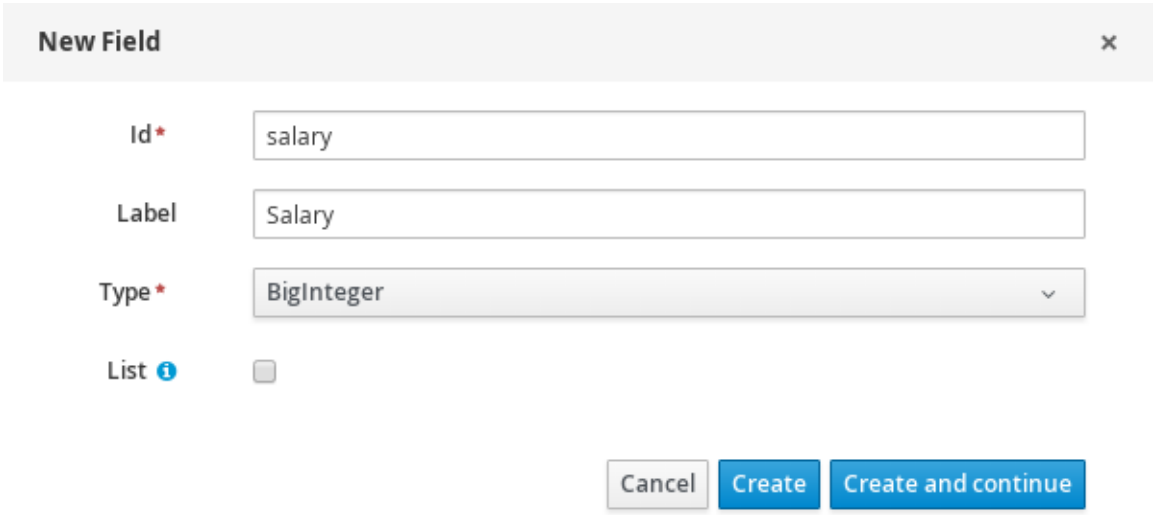
您可以将另一软件包中的现有数据对象直接导入到资产设计人员，如指导规则或指导决策表设计人员。在项目中选择相关规则资产以及资产设计器，请转至 Data Objects → New item 以选择要导入的对象。

4. 要使数据对象持久化，请选择"永久 "复选框。Persistable 数据对象可以根据 JPA 规格存储在数据库中。默认的 JPA 为 Hibernate。
5. 点 确定。
6. 在数据对象设计器中，点 add 字段在 对象中添加包含属性 Id、标签和 类型的字段。所需的属性标有星号(*)。

- id： 输入字段的唯一 ID。

- **label:（可选）输入字段的标签。**
- **Type：输入字段的数据类型。**
- **列表：（可选）选择此复选框，以启用字段保存指定类型的多个项目。**

图 42.1. 在数据对象中添加数据字段



7. **点 Create 添加新字段，或者点击 Create 并继续添加新字段并继续添加其他字段。**

注意

要编辑字段，请选择字段行，并使用屏幕右侧的 常规属性。

第 43 章 决策表用例

在线购物网站列出了已订购项目的发运费用。该站点在以下情况下提供免费发运：

- 排序的项目数量为 4 个或更多，结账总数为 $300 或更多。
- 选择标准发运（自购买之日起 4 或 5 个工作日）。

以下是在以下情况下的发运率：

表 43.1. 对于小于 $300 的订单

项数	交付日	美国运运费用、N = 数量项
3 个或更少	下一天 第 2 天 Standard（标准）	35 15 10
4 个或更多	下一天 第 2 天 Standard（标准）	N*7.50 N*3.50 N*2.50

表 43.2. 对于超过 $300 的订单

项数	交付日	美国运运费用、N = 数量项
3 个或更少	下一天 第 2 天 Standard（标准）	25 10 N*1.50
4 个或更多	下一天 第 2 天 Standard（标准）	N*5 N*2 FREE

以下示例电子表格决策表中显示了这些条件和速率：

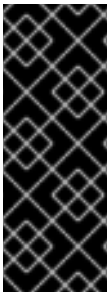
*图 **43.1.** 发运费用的表格*

	A	B	C	D	E	F
1		RuleSet	Charge Calculator			
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge			
3		Variables	Integer totalCount			
4		Sequential	TRUE			
5		SequentialMaxPriority	10			
6						
7		RuleTable Basic				
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION	
9		$order : Order				
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));	
11		Min items	Max items	Delivered in days	Pay charge in US dollars	
12	expensive	0	3	1	35	
13		0	3	2	15	
14		0	3		10	
15		4		1	$order.getItemsCount() * 7.5	
16		4		2	$order.getItemsCount() * 3.5	
17	cheap	4			$order.getItemsCount() * 2.5	
18						
19		RuleTable Expensive				
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
21		$order : Order				
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1	insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than	Pay charge in US dollars
24	expensive	0	3	1	300	25
25		0	3	2	300	10
26		0	3		300	$order.getItemsCount() * 1.5
27		4		1	300	$order.getItemsCount() * 5
28		4		2	300	$order.getItemsCount() * 2
29	cheap	4			300	0
30						

*为了在 **Business Central** 中上传表格，该表必须符合 **XLS** 或 **XLSX** 电子表格中的特定结构和语法要求，如下例所示。更多信息请参阅 **第 44 章 定义电子表格决策表**。*

第 44 章 定义电子表格决策表

电子表格决策表（XLS 或 XLSX）需要两个关键区域来定义规则数据：RuleSet 区域和一个 RuleTable 区域。电子表格的 RuleSet 区域定义了您要对同一软件包中的所有规则（不仅仅是电子表格）全局应用的元素，如规则集名称或通用规则属性。RuleTable 区域定义了实际规则（箭头）以及条件、操作和其他规则属性（列），这些属性构成指定规则集中的规则表。表格的表格可以包含多个可 规则的区域，但只能包含一个 RuleSet 区域。



重要

通常，根据 Business Central 中的规则软件包，您通常只 上传一个 决策表，其中包含所有必要规则定义定义。您可以为单独的软件包上传单独的决策表电子表格，但在同一个软件包中上传多个电子表格可能会导致与冲突的 RuleSet 或 RuleTable 属性中相应的错误，因此不建议这样做。

在您定义决策表时，请参考以下示例电子表格：

图 44.1. 发运费用的电子表格决策表示例

	A	B	C	D	E	F
1		RuleSet	Charge Calculator			
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge			
3		Variables	Integer totalCount			
4		Sequential	TRUE			
5		SequentialMaxPriority	10			
6						
7		RuleTable Basic				
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION	
9		$order : Order				
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));	
11		Min items	Max items	Delivered in days	Pay charge in US dollars	
12	expensive	0	3	1	35	
13		0	3	2	15	
14		0	3		10	
15		4		1	$order.getItemsCount() * 7.5	
16		4		2	$order.getItemsCount() * 3.5	
17	cheap	4			$order.getItemsCount() * 2.5	
18						
19		RuleTable Expensive				
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
21		$order : Order				
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1	insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than	Pay charge in US dollars
24	expensive	0	3	1	300	25
25		0	3	2	300	10
26		0	3		300	$order.getItemsCount() * 1.5
27		4		1	300	$order.getItemsCount() * 5
28		4		2	300	$order.getItemsCount() * 2
29	cheap	4			300	0
30						

流程

1.

在新的 XLS 或 XLSX 电子表格中，转至第二栏或第三列，并标记单元 RuleSet （例如 1）。为左侧的列或列保留描述性元数据（可选）。

2. 在右侧的单元里，为 RuleSet 输入一个名称。此命名规则集将包含 规则 软件包中定义的所有规则。

3. 在 RuleSet 单元下，定义您要对软件包中的所有规则表进行全局应用的任何规则属性（每个单元一个）。指定右侧的单元格中的属性值。例如，您可以输入 Import 标签并在右侧的单元格中，为您要导入的其它软件包（格式为 package.name.object.name）指定相关的数据对象。有关支持的单元标签和值，请参阅 第 44.1 节 “规则集定义”。

4. 在 RuleSet 区域中，在 RuleSet 单元下，跳过一行并标记一个新的单元 规则 （示例中是 7），然后在相同的单元中输入表名称。名称用作来自此规则表的所有规则的初始部分，并附加行号。您可以通过插入 NAME 属性列来覆盖此自动命名。

5. 使用下面的四个行根据需要定义以下元素（如例中的 8-11 ）：

- 规则属性： Conditions、actions 或其他属性。有关支持的单元标签和值，请参阅 第 44.2 节 “RuleTable 定义”。
- 对象类型： 规则属性应用到的数据对象。如果同一对象类型适用于多个列，请将对象单元格合并到多个列之间的一个单元格中（如示例决策表中所示），而不是在多个单元中重复对象类型。当对象类型合并时，合并范围下的所有列都将合并到单个模式中用于匹配单个事实的一组约束。在单独的列中重复对象时，各个列可以创建不同的模式，可能会匹配不同的或相同的事实。
- 约束： 对象类型的限制。
- 列标签： （可选）列的任何描述性标签（可选）作为视觉 aid。如果未使用，留空。

注意

作为填充对象类型和约束单元格的替代选择，您可以将对象类型单元格或单元格留空，然后在对应的约束单元或单元格中输入完整的表达式。例如，如果对象类型和 itemsCount > $1 作为约束（不同的单元格），您可以保留对象类型单元格，然后在约束单元中输入 Order (itemsCount > $1)，然后为其他约束单元执行相同的操作。

6. 定义了所有必要的规则属性（列），根据需要输入每个列的值，按行行（例如，第 12-17 一行）生成规则（例如，第 12-17）。没有数据的单元将被忽略（比如当条件或操作不适用时）。

如果您需要在此决策表格中添加更多规则表，在上表中的最后一条规则后跳过一行，在与前面的 RuleTable 和 RuleSet cells 相同，再按照本部分中相同的步骤创建新表（示例为 19-29）。

7. 保存 XLS 或 XLSX 电子表格来完成。

注意

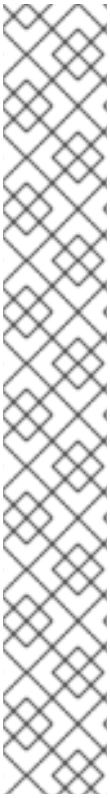
默认情况下，在上传 Business Central 中电子表格的电子表格时，只有电子表格的首个工作表才会作为决策表进行处理。每个 RuleSet 名称都与 RuleTable 名称组合，必须在同一软件包中的所有决策表文件中是唯一的。

如果要处理多个工作表表，请创建一个 .properties 文件，其名称与电子表格图书器相同。.properties 文件必须包含带有以逗号分隔的值(CSV)的属性表，例如：

```
sheets=Sheet1,Sheet2
```

在上传 Business Central 中的决策表后，规则会呈现如下 DRL 规则，如以下示例表格：

```
//row 12
rule "Basic_12"
salience 10
  when
    $order : Order( itemsCount > 0, itemsCount <= 3, deliverInDays == 1 )
  then
    insert( new Charge( 35 ) );
end
```



启用在单元值中使用的空格

默认情况下，在决策引擎处理决策表单元之前或之后的任何空格之前或之后的任何空格会被移除。要保留您在单元格之前或之后使用的空白空间，请将 drools 设为 Red Hat Process Automation Manager 发行中的 false。

例如，如果您在红帽 JBoss EAP 中使用 Red Hat Process Automation Manager，请在 $EAP_HOME/standalone/configuration/standalone-full.xml 文件中添加以下系统属性：

```
<property name="drools.trimCellsInDTable" value="false"/>
```

如果您使用嵌入在 Java 应用程序中的决策引擎，使用以下命令添加系统属性：

```
java -jar yourApplication.jar -Ddrools.trimCellsInDTable=false
```

44.1. 规则集定义

决策表的 RuleSet 区域中的条目定义了您要应用到软件包中所有规则的 DRL 结构和规则属性（不仅仅在电子表格中）。条目必须位于垂直的单元对序列中，其中第一个单元包含一个标签以及右侧包含值的单元格。表格电子表格只能有一个 RuleSet 区域。

下表列出了 RuleSet 定义支持的标签和值：

表 44.1. 支持的 RuleSet 定义

标签	值	使用
ruleset	生成的 DRL 文件的软件包名称。可选，默认为 **rule_table**。	必须是第一个条目。
sequential	**true** 或 **false**。如果为 **true**，则使用 salience 来确保规则从上下触发。	可选，最多一次。如果省略，则不会强制触发顺序。
SequentialMaxPriority	整数数字值	可选，最多一次。在连续模式中，此选项用于设置 salience 的开始值。如果省略，默认值为 65535。

标签	值	使用
SequentialMinPriority	整数数字值	可选，最多一次。在连续模式中，使用这个选项检查是否没有违反此最小值。如果省略，则默认值为 0。
EscapeQuotes	**true** 或 **false**。如果为 **true**，则引号会被转义，以便它们在 DRL 中以文字形式出现。	可选，最多一次。如果省略，则转义引号。
IgnoreNumericFormat	**true** 或 **false**。如果为 **true**，则忽略数字值的格式，如百分比和货币。	可选，最多一次。如果省略，DRL 会采用格式化的值。
Import	从另一个软件包中导入的、以逗号分隔的 Java 类列表。	可选，可以重复使用。
变量	DRL 全局声明（类型后跟变量名称）。必须使用逗号分隔多个全局定义。	可选，可以重复使用。
Functions	根据 DRL 语法，一个或多个功能定义。	可选，可以重复使用。
查询	根据 DRL 语法，一个或多个查询定义。	可选，可以重复使用。
声明	根据 DRL 语法，一个或多个声明类型。	可选，可以重复使用。
å•ä½	此决策表中生成的规则单元。	可选，最多一次。如果省略，则规则不属于任何单元。
dialect	**Java** 或 **mvel**.决策表操作中使用的 dialect。	可选，最多一次。如果省略，则使用 **java**。

警告

在某些情况下，Microsoft Office、LibreOffice 和 OpenOffice 可能会以不同的方式编码双引号，从而导致编译错误。例如，"A" 将失败，但 "A" 将传递。

44.2. RULETABLE 定义

决策表的 " 规则"区域中的条目定义了该规则表中的条件、操作和其他规则属性。表格的表格可以包含多个可 规则 的区域。

下表列出了规则定义支持的标签（列标题）和值。对于列标题，您可以使用给定标签或以表中列出的字母开头的任何自定义标签。

表 44.2. 支持的规则 定义

标签	或以 开头的自定义标签	值	使用
NAME	N	提供从该行生成的规则的名称。默认值由 **RuleTable** 标签和行号中的内容构建。	最多一列。
描述	I	在生成的规则中生成注释。	最多一列。
条件	C	代码片段和交集值，用于在状况中的模式内构建约束。	每个规则表至少一个。
操作	A	代码片段和交集值，用于构造规则导致操作的操作。	每个规则表至少一个。
元数据	@	用来构建规则元数据条目的代码片段和交集值。	可选，任意数量的列。

以下小节提供有关条件、操作和元数据列如何使用单元数据的更多详情：

Conditions

对于列头 **CONDITION**，连续的行中的单元会产生一个条件元素：

- 第一单元： COND ITION 的第一个单元格中的文本将开发成规则条件的模式，并使用下一行中的代码片段作为约束。如果单元与一个或多个邻居单元合并，则会形成含有多个限制的单一模式。所有限制都合并到一个圆括号列表中，并附加到这个单元文本中。

  如果这个单元单元为空，位于下面的单元格中的代码片段必须自行生成有效的条件元素。例如，如果对象类型和 itemsCount > $1 作为约束（不同的单元格），您可以保留对象类型单元格，然后在约束 单元中输入 Order (itemsCount > $1)，然后对任何其他约束单元执行相同的操作。

  要包括没有限制的模式，您可以在另一模式文本前编写模式，带有或不含空对括号。您也可以将 from 子句附加到模式。

  如果模式以 eval 结尾，代码片段会生成布尔值表达式，以便在 eval 之后将包含到一组

括号中。

您可以终止带有 @watch 注释的模式，该注释用于自定义该模式要重新激活的属性。

- 第二单元格：在 CONDITION 下第二个单元格中的文本作为在第一单元中引用的约束进行处理。这个单元中的代码片段可以通过将值与列中的单元格隔离来修改。如果要创建一个限制，使用 == 与下方单元格中的值进行比较，那么该字段选择器已经足够了。如果您只使用字段选择器，但您想要使用条件而无需附加任何 == 比较，则必须使用符号？终止条件。任何其它比较运算符都必须作为代码片段中最后一个项目指定，且附加了下面的单元中的值。对于所有其他约束形式，您必须标记位置，使其包含带有符号 $param 的单元格的内容。如果您使用符号 $1、$2 等，以及以下单元中的以逗号分隔的值列表，则可以多个插入。但是，不要用逗号分开 $1、$2 等，或者以逗号分隔，否则表将无法处理。

  要根据模式扩展文本 ($delimiter){ $snippet }，请为以下每个单元里的逗号分隔列表的值重复一次（以逗号分隔列表的值），插入符号 $ 符号，并使用所给 $delimiter 来加入这些扩展。请注意，forall 结构可由其他文本包围起来。

  如果第一个单元包含对象，则完成的代码片段会添加到该单元中的条件元素中。会自动提供一对括号，并在合并的单元中的模式中添加多个限制时使用逗号分隔逗号。如果第一个单元格为空，此单元格中的代码片段必须自行生成有效的条件元素。例如，如果对象类型和 itemsCount > $1 作为约束（不同的单元格），您可以保留对象类型单元格，然后在约束 单元中输入 Order (itemsCount > $1)，然后对任何其他约束单元执行相同的操作。

- 第三单元：在 CONDITION 下的第三单元格中文本是您为列定义的描述性标签，它是一个视觉帮助。

- 第四单元：在第四行中，非空条目为干预提供数据。空白单元会省略此规则的条件或约束。

Actions

对于列标题 ACTION，连续行中的单元会导致 action 语句：

- 第一单元：在 ACTION 下第一个单元格中的文本是可选的。如果存在，该文本将解释为对象引用。

- 第二单元格：在 ACTION 以下的第二个单元格中文本是通过将值与一列中的单元关联值来修改的代码片段。对于单边插入，标记 的位置，用符号 $param 标记包含单元内容的位

置。如果您使用符号 $1、$2 等符号，以及下面单元格中的以逗号分隔的值列表，则可以进行多个插入。但是，不要用逗号分隔 $1、$2 等，否则表将无法处理。

不含任何标记符号的文本可以在不进行干预的情况下执行方法调用。在这种情况下，在单元格下面的行中使用任何非空条目，使其包含声明。支持 forall 结构。

如果第一个单元包含对象，那么单元文本（按句点跟随），第二个单元中的文本会被停止运行，从而导致一个方法调用一个操作声明。如果第一个单元单元为空，此单元中的代码片段必须自行生成有效的 action 元素。

- 第三单元：以下第三单元格中的文本是您为列定义的描述性标签，它是一个视觉帮助。
- 第四单元：在第四行中，非空条目为干预提供数据。空白单元会省略此规则的条件或约束。

元数据

对于列头 METADATA，连续行中的单元格会为生成的规则生成元数据注解：

- 第一单元：METADATA 的第一个单元格中的文本将被忽略。
- 第二单元格：METADATA 下第二个单元里的文本使用规则行中的单元格中的值进行干预。元数据标志字符 @ 是前缀，因此您不需要将该字符包含在这个单元的文本中。
- 第三单元：在 METADATA 下的第三单元格中的文本是您为列定义的描述性标签，作为可视化工具。
- 第四单元：在第四行中，非空条目为干预提供数据。空白单元格会导致此规则出现对元数据注解的省略。

44.3. RULESET 或 RULETABLE 定义的其他规则属性

RuleSet 和 RuleTable 区域也支持其他规则属性的标签和值，如 PRIORITY 或 NO-LOOP。在 RuleSet 区域中指定的规则属性将影响同一软件包中的所有规则资产（而不仅仅是电子表格）。在 规则区域中指定的规则 属性将仅影响该规则表中的规则。您只能在 RuleSet 区域中使用每条规则属性一次，并

在 规则 区域中使用一次。如果在电子表格内的 RuleSet 和 RuleTable 区域中使用相同的属性，则 RuleTable 使用 priority，并且 RuleSet 区域中的 属性会被覆盖。

下表列出了支持的标签（列标题）以及额外 RuleSet 或 RuleTable 定义的值。对于列标题，您可以使用给定标签或以表中列出的字母开头的任何自定义标签。

表 44.3. RuleSet 或 RuleTable 定义的其他规则属性

标签	或以 开头的自定义标签	值
PRIORITY	P	定义规则的 **salience** 值的整数。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。由 **Sequential** 标志覆盖。 示例： **PRIORITY 10**
日期有效	V	包含日期和时间定义的字符串。仅当当前的日期和时间在 **DATE-EFFECTIVE** 属性后才会激活该规则。 示例： **DATE-EFFECTIVE "4-Sep-2018"**
DATE-EXPIRES	Z	包含日期和时间定义的字符串。如果当前的日期和时间位于 **DATE-EXPIRES** 属性后，则无法激活该规则。 示例： **DATE-EXPIRES "4-Oct-2018"**
NO-LOOP	U	布尔值。当这个选项被设置为 **true** 时，如果规则被重新触发一个之前满足的条件，则无法重新激活（循环）规则。 示例： **NO-LOOP true**
AGENDA-GROUP	G	为您指定要为其分配该规则的日程表组的字符串。日程表组允许您对规则组进行更多执行控制。只有已获取焦点的管理者组中的规则才能够被激活。 示例： **AGENDA-GROUP "GroupName"**
ACTIVATION-GROUP	X	您要为其分配该规则的激活（或 XOR）组的字符串。在激活组中，只能激活一条规则。第一条规则取消激活组中所有规则的待处理激活。 示例： **ACTIVATION-GROUP "GroupName"**
DURATION	D	如果规则条件仍满足，则用于定义在激活规则的时间持续时间（以毫秒为单位）的长整数值。 示例： **DURATION 10000**

标签	或以 开头的自定义标签	值
TIMER	T	用于标识 **int** (interval)或 **cron** 计时器定义的字符串，用于调度规则。 示例： **TIMER "*/5 * * * *"** （每 5 分钟）
日历	E	用于调度规则的 Quartz 日历定义。 示例： **CALENDAR "* * 0-7,18-23 ?* *"** （排除非工作时间）
AUTO-FOCUS	F	布尔值，仅适用于 schedule groups 中的规则。当此选项设置为 **true** 时，下一次激活规则时，会自动把主要提供给分配给该规则的 table 组。 示例： **AUTO-FOCUS true**
LOCK-ON-ACTIVE	L	布尔值，仅适用于规则流组或日程组中的规则。当此选项设置为 **true** 时，规则的 ruleflow 组下次变为活跃时，规则的管理者组会集中接受，否则无法再次激活该规则，直到 ruleflow 组不再活跃，或 table 组失去焦点。这是 **no-loop** 属性的一个更强大的版本，因为无论更新的来源，都丢弃匹配规则的激活（而不只是规则本身）。此属性非常适合计算规则，其中有多个规则修改事实，而您不想再次匹配和触发任何规则。 示例： **LOCK-ON-ACTIVE true**
RULEFLOW-GROUP	R	标识规则流组的字符串。在规则流组中，只有在相关规则流激活组时，规则才能触发。 示例： **RULEFLOW-GROUP "GroupName"**

图 44.2. 使用属性列的决策表电子表格示例

	A	B	C	D	E	F	G	H
1		RuleSet	Charge Calculator					
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge					
3		Variables	Integer totalCount					
4		Sequential	TRUE					
5		SequentialMaxPriority	10					
6								
7		RuleTable Basic						
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION			
9		$order : Order						
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));			
11		Min items	Max items	Delivered in days	Pay charge in US dollars			
12	expensive	0	3	1	35			
13		0	3	2	15			
14		0	3		10			
15		4		1	$order.getItemsCount() * 7.5			
16		4		2	$order.getItemsCount() * 3.5			
17	cheap	4			$order.getItemsCount() * 2.5			
18								
19		RuleTable Expensive						
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	RULEFLOW-GROUP	NOLOOP	ACTION
21		$order : Order						
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1			insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than			Pay charge in US dollars
24	expensive	0	3	1	300		TRUE	25
25		0	3	2	300		TRUE	10
26		0	3		300		TRUE	$order.getItemsCount() * 1.5
27		4		1	300	discount assessment		$order.getItemsCount() * 5
28		4		2	300	discount assessment		$order.getItemsCount() * 2
29	cheap	4			300	discount assessment		0
30								

第 45 章 将电子表格决策表上传到 BUSINESS CENTRAL

在决策表的外部 XLS 或 XLSX 表格中定义规则后，您可以将电子表格文件上传到 Business Central 中的项目。

重要

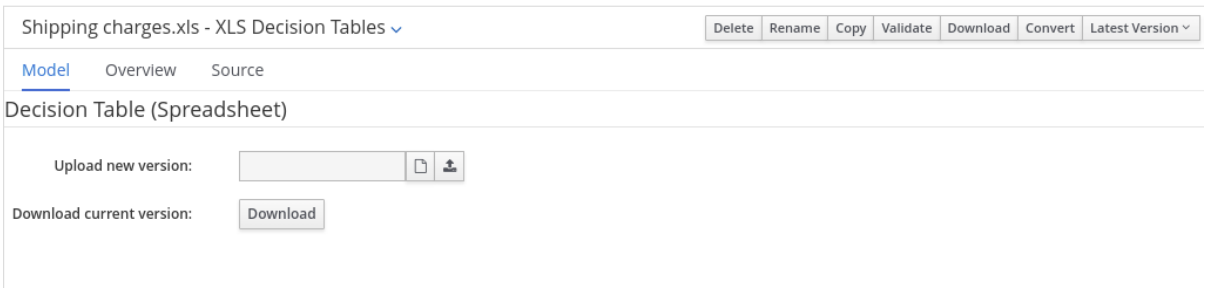
通常，根据 Business Central 中的规则软件包，您通常只上传一个决策表，其中包含所有必要规则定义定义。您可以为单独的软件包上传单独的决策表电子表格，但在同一个软件包中上传多个电子表格可能会导致与冲突的 RuleSet 或 RuleTable 属性中相应的错误，因此不建议这样做。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Decision Table(Spreadsheet)
3. 输入信息性决策表名称并选择相应的软件包。
4. 点 Choose File 图标，然后选择电子表格。点 确定 进行上传。
5. 在 decision table designer 中，单击右上角工具栏中的 Validate 来验证表。如果表验证失败，打开 XLS 或 XLSX 文件并解决任何语法错误。有关语法帮助，请参阅 第 44 章 定义电子表格决策表。

您可以上传决定表的新版本，或者下载当前版本：

图 45.1. 上传的决定表选项



第 46 章 将上传的电子表格决策表转换为 BUSINESS CENTRAL 中的指导决策表

将 XLS 或 XLSX 电子表格决策表文件上传到 Business Central 中的项目后，您可以将决策表转换为直接在 Business Central 中修改的表格。

有关指导决策表的更多信息，请参阅使用指导决策表设计决策服务。

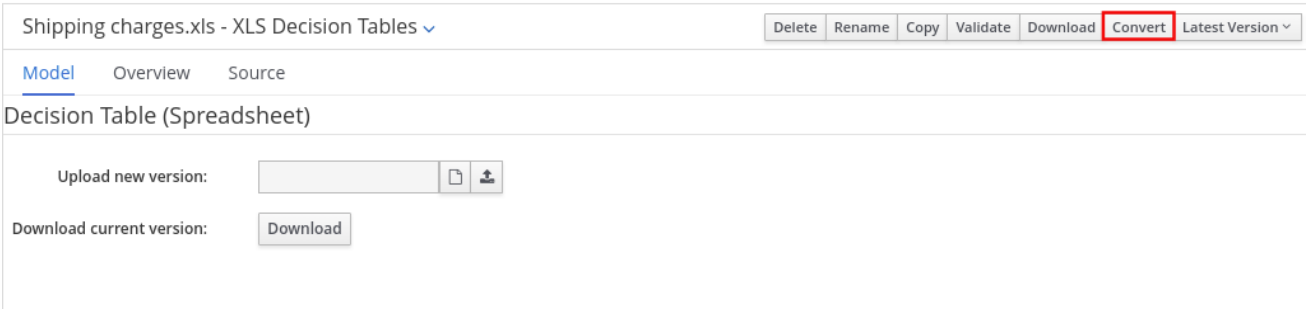
警告

指导的决策表和电子表格表格是支持不同功能的决策表格式。在将两个决策表格式转换为另一表格式时，任何不同支持的功能都会修改或丢失。

流程

1. 在 Business Central 中，进入您希望在决策表设计器右上角转换和上传的决策表资产，点 Convert：

图 46.1. 转换上传的决定表



2. 转换后，您可以在项目中直接修改 Business Central 的项目中提供转换的决策表资产。

第 47 章 执行规则

在确定了示例规则或在 Business Central 中创建自己的规则后，您可以在本地或 KIE 服务器上构建并部署相关的项目并在 KIE 服务器上执行规则，以测试规则。

先决条件

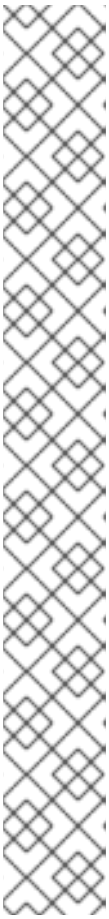
- 业务中心和 KIE 服务器已安装并运行。有关安装选项，请参阅 规划 Red Hat Process Automation Manager 安装。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。

2. 在项目 资产 页面右上角，单击 Deploy 以构建该项目，并将它部署到 KIE 服务器。如果构建失败，请解决屏幕底部的 Alerts 面板中描述的问题。

   有关项目部署选项的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。



注意

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

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

默认情况下，Red Hat Process Automation Manager 中的规则资产需要这个依赖项，从可执行规则模型构建。这个依赖关系包含在 Red Hat Process Automation Manager 核心打包中，但取决于 Red Hat Process Automation Manager 升级历史记录，您可能需要手动添加这个依赖关系来启用可执行规则模型行为。

有关可执行规则模型的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。

3. 在 Business Central 之外创建 Maven 或 Java 项目（如果尚未创建），您可将其用于在本地执行规则，或用作在 KIE 服务器上执行规则的客户端应用程序。该项目必须包含 pom.xml 文件以及执行项目资源的任何其他必要组件。

   有关 test 项目示例，请参阅 "创建和执行 DRL 规则的方法 "。

4. 打开 test 项目或客户端应用程序的 pom.xml 文件，并添加以下依赖项（如果还没有添加）：

   - kie-ci ：启用客户端应用程序以使用 ReleaseId在本地加载 Business Central 项目数据
   - kie-server-client ：使您的客户端应用程序能够与 KIE 服务器上的资产远程交互
   - slf4j:（可选）使您的客户端应用程序能够使用 Simple Logging Facade 用于 Java(SLF4J)，以在与 KIE 服务器交互后返回调试信息

   客户端应用程序 pom.xml 文件中的 Red Hat Process Automation Manager 7.13 的依赖项示例：

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

   对于这些工件的可用版本，请在 Nexus Repository Manager 在线搜索组 ID 和工件 ID。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

5. 确保在客户端应用 pom.xml 文件中定义了包含模型类的工件依赖项，它们与部署的项目的 pom.xml 文件中完全相同。如果模型类的依赖关系因客户端应用和项目之间有所不同，则可能会出现执行错误。

要访问 Business Central 中的项目 pom.xml 文件，请在项目左侧选择任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

例如，以下 Person 类依赖项同时出现在客户端和部署的项目 pom.xml 文件中：

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. 如果您将 slf4j 依赖项添加到用于调试日志的客户端应用程序 pom.xml 文件，请在相关类路径（例如，在 Maven 中的 src/main/resources/META-INF）上创建一个 simplelogger.properties 文件（例如，在 Maven 中的 src/main/resources/META-INF）：

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

-

7.

在您的客户端应用中，创建一个包含必要导入和 main（） 方法的 .java 主类，以加载 KIE 基础、插入事实和执行规则。

例如，项目中的 Person 对象包含 getter 和 setter 方法，用于设置并检索一个人的名字、姓氏、每小时速率和分流。项目中的以下 Wage 规则计算 wage 和 hourly 速率值，并根据结果显示消息：

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

要在 KIE 服务器之外测试此规则（如果需要），请配置 .java 类以导入 KIE 服务、KIE 容器和 KIE 会话，然后使用 main（） 方法针对定义的事实模型触发所有规则：

本地执行规则

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

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

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

      // Set up the fact model:
      Person p = new Person();
```

```
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert the person into the session:
        kSession.insert(p);

        // Fire all rules:
        kSession.fireAllRules();
        kSession.dispose();
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
  }
```

要在 KIE 服务器上测试此规则，请配置 .java 类，其导入和规则执行信息与本地示例类似，另外还指定 KIE 服务配置和 KIE 服务客户端详情：

在 KIE 服务器上执行规则

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {
```

```
  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert Person into the session:
      KieCommands kieCommands = KieServices.Factory.get().getCommands();
      List<Command> commandList = new ArrayList<Command>();
      commandList.add(kieCommands.newInsert(p, "personReturnId"));

      // Fire all rules:
      commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
      BatchExecutionCommand batch =
kieCommands.newBatchExecution(commandList, sessionName);

      // Use rule services client to send request:
      RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
      ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
      System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

8. 从项目目录中运行配置的 .java 类。您可以在开发平台（如 Red Hat CodeReady Studio）或命令行中运行该文件。

Maven 执行示例（在项目目录中）：

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

Java 执行示例（在项目目录中）

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

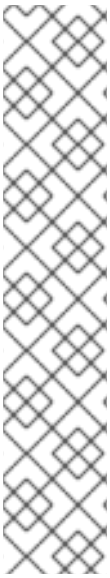
9. 在命令行中检查规则执行状态并在服务器日志中。如果有任何规则没有按预期执行，请检查项目中配置的规则，以及验证提供的数据的主类配置。

第 48 章 后续步骤

- 使用测试场景测试决策服务
- 打包和部署 Red Hat Process Automation Manager 项目

部分 VI. 使用指导规则设计决策服务

作为业务分析员或业务规则开发人员，您可以使用 Business Central 中的指导规则设计人员定义业务规则。这些指导规则编译为 Drools 规则语言(DRL)，并形成您的项目决策服务的核心。



注意

您还可以使用决策模型和 Notation(DMN)模型设计您的决策服务，而不是基于规则或基于表的资产。有关 Red Hat Process Automation Manager 7.13 中的 DMN 支持的详情，请查看以下资源：

- 开始使用决策服务 （逐步教程，带有 DMN 决策服务示例）
- 使用 DMN 模型设计决策服务 (Red Hat Process Automation Manager 中的 DMN 支持和功能视图)

先决条件

- Business Central 中创建了指导规则的空间和项目。每个资产都与分配给一个空间的项目相关联。详情请参阅 开始使用决策服务。

第 49 章 红帽流程自动化管理器中的决策资产

Red Hat Process Automation Manager 支持一些可用于为决策服务定义业务决策的资产。每个决策资产都有不同的优势，您可以根据目标和需求，首选使用一个或多个资产的组合。

下表重点介绍红帽流程自动化管理器项目支持的主要决策资产，以帮助您决定或确认在决策服务中定义决策的最佳方法。

表 49.1. Red Hat Process Automation Manager 支持的决策资产

asset	亮点	编写工具	Documentation
决策模型和符号 (DMN)型号	• 决策模型基于由对象管理组 (OMG)定义的符号标准。 • 使用代表部分或所有决策要求图 (DRG)的图形化决策要求图 (DRG)跟踪业务决策流程 • 使用 XML 模式，允许在 DMN 兼容平台间共享 DMN 模型 • 支持 Friendly Enough Expression Language(FEEL)，在 DMN 决策表中定义决策逻辑，以及其他 DMN 框的表达式 • 可以与业务流程模型和符号 (BPMN)流程模型有效集成 • 是创建全面、清晰和稳定的决策流程的最佳选择	Business Central 或其他与 DMN 兼容的编辑器	*使用 DMN 模型设计决策服务*
指导的决定表	• 是您在 Business Central 中基于 UI 的表格设计者创建的规则表 • 是电子表格决策表的向导替代选择 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值 • 支持点击策略、实时验证和其他资产不支持的其他功能 • 是以受控表格格式创建规则的最佳选择，可最小化编译错误	Business Central	*使用指导决策表设计决策服务*

asset	亮点	编写工具	Documentation
电子表格决策表	• 是 XLS 或 XLSX 表格决策表，您可以上传到 Business Central • 支持创建规则模板的模板键和值 • 是创建已在 Business Central 之外管理的决策表中规则的最佳选择 • 在上传后，对规则进行编译具有严格的语法要求	电子表格编辑器	*使用电子表格决策表设计决策服务*
指导规则	• 是您在 Business Central 中基于 UI 的规则设计者创建的单独规则 • 为可接受的输入提供字段和选项 • 是以受控格式创建单一规则的最佳选择，以最小化编译错误	Business Central	*使用指导规则设计决策服务*
指导规则模板	• 是您在 Business Central 中基于 UI 的模板设计者创建的可重复使用的规则结构 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值（与此资产的目的不相同） • 是创建具有相同规则结构但具有不同定义字段值的很多规则的最佳	Business Central	*使用指导规则模板设计决策服务*
DRL 规则	• 是直接在 **.drl** 文本文件中定义的单独规则 • 提供定义规则和其他规则行为的最大的灵活性 • 可以在特定的独立环境中创建，并与 Red Hat Process Automation Manager 集成 • 是创建需要高级 DRL 选项的规则的最佳选择 • 对规则进行正确编译的规则具有严格的语法要求	业务中心或集成开发环境(IDE)	*使用 DRL 规则设计决策服务*

asset	亮点	编写工具	Documentation
预测模型标记语言 (PMML)型号	• 是基于数据 Mining Group(DMG) 定义的标记标准预测数据分析模型。 • 使用允许 PMML 模式在 PMML 兼容的平台之间共享的 XML 模式 • 支持 Regression、scorecard、Tree、Mining 和其他模型类型 • 可以包含在独立 Red Hat Process Automation Manager 项目中，或者导入到 Business Central 中的项目 • 在 Red Hat Process Automation Manager 中，将预测数据合并为决策服务的最佳选择	PMML 或 XML 编辑器	*使用 PMML 模型设计决策服务*

在定义决策时，您还可以考虑将红帽构建的 Kogito 构建用于云原生决策服务。有关使用红帽构建的 Kogito 微服务的更多信息，请参阅 Red Hat Process Automation Manager 中的 Red Hat build of Kogito。

第 50 章 指导规则

指导规则是您通过规则创建在业务中心基于 UI 的指导规则设计者中创建的业务规则。指导规则设计器根据所定义规则的数据对象，提供可接受输入的字段和选项。您定义的指南规则与所有其他规则资产一样编译为 Drools Rule Language(DRL)规则。

与指导规则相关的所有数据对象都必须位于与指导规则相同的项目软件包中。默认导入同一软件包中的资产。创建必要的数据对象和指导规则后，您可以使用指南规则设计器的 Data Objects 选项卡来验证所有所需数据对象是否已列出或导入其他现有数据对象，方法是添加新项目。

第 51 章 数据对象

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

51.1. 创建数据对象

以下流程是创建数据对象的一般概述。它并不适用于特定的业务资产。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Data Object。
3. 输入唯一数据对象名称，然后选择您希望数据对象可用于其他规则资产的软件包。 同一名称的数据对象不能位于同一软件包中。在指定的 DRL 文件中，您可以从任何软件包中导入数据对象。

从其他软件包导入数据对象

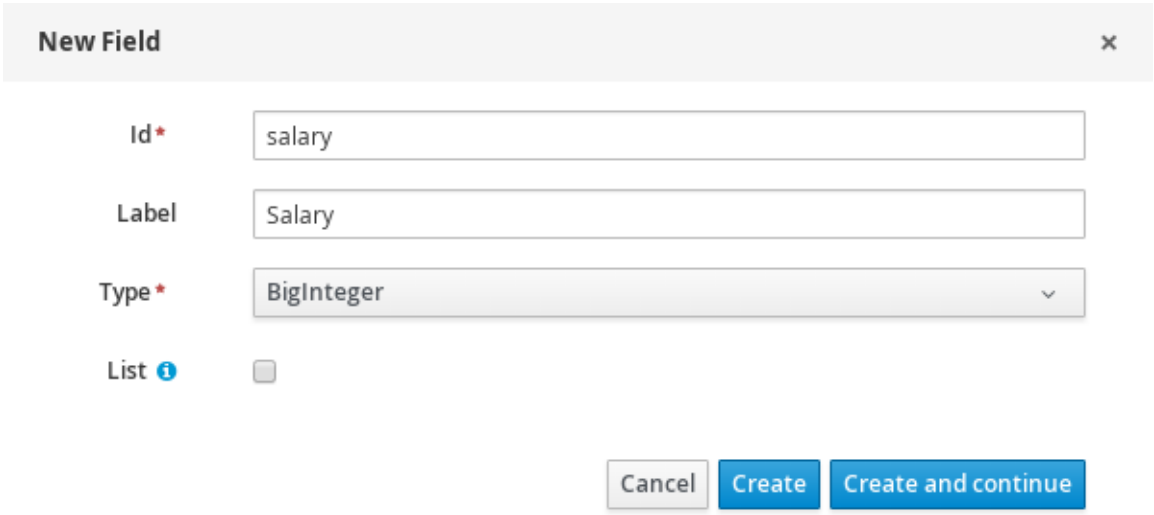
您可以将另一软件包中的现有数据对象直接导入到资产设计人员，如指导规则或指导决策表设计人员。在项目中选择相关规则资产以及资产设计器，请转至 Data Objects → New item 以选择要导入的对象。

4. 要使数据对象持久化，请选择"永久 "复选框。Persistable 数据对象可以根据 JPA 规格存储在数据库中。默认的 JPA 为 Hibernate。
5. 点 确定。
6. 在数据对象设计器中，点 add 字段在 对象中添加包含属性 Id、标签和 类型的字段。所需的属性标有星号(*)。

- id： 输入字段的唯一 ID。

- **label:** （可选）输入字段的标签。
- **Type：** 输入字段的数据类型。
- **列表 ：** （可选）选择此复选框，以启用字段保存指定类型的多个项目。

图 51.1. 在数据对象中添加数据字段



7. 点 Create 添加新字段，或者点击 Create 并继续添加新字段并继续添加其他字段。

注意

要编辑字段，请选择字段行，并使用屏幕右侧的 常规属性。

第 52 章 创建指导规则

借助指导规则，您可以根据与规则关联的数据对象，以结构化格式定义业务规则。您可以单独为项目创建并定义指导规则。

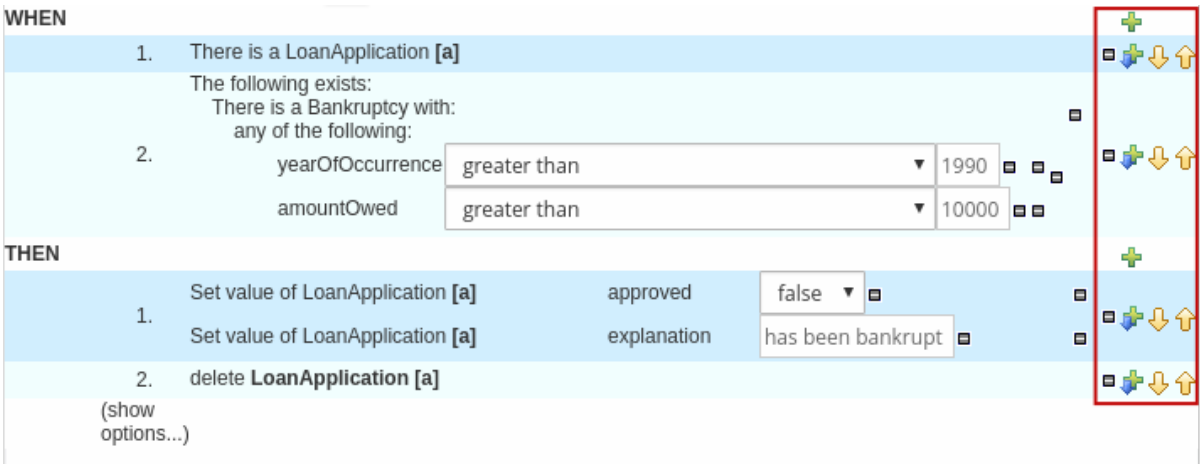
流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Guided Rule。
3. 输入信息 引导规则 名称并选择相应的 软件包。您指定的软件包必须是分配或将被分配所需数据对象的同一软件包。

   如果项目中已定义了任何域特定语言(DSL)资产，您也可以选择 Show 声明 DSL 句子。这些 DSL 资产将变成可用于您在指导规则设计人员中定义的条件和操作对象。
4. 点 Ok 创建规则资产。

   新的指导规则现在列在 Project Explorer 的引导 规则 面板中，如果您选择了 Show 声明 DSL 句子选项，则在引导规则（带有 DSL ） 面板中列出。
5. 点 Data Objects 选项卡，并确认列出了您的规则所需的所有数据对象。如果没有，请单击 New item 以从其他软件包导入数据对象，或者在您的软件包 中创建数据对象。
6. 在所有数据对象都就位后，根据可用的数据对象，返回到指导规则设计器的 Model 选项卡，并使用窗口右侧的按钮来添加和定义 WHEN (action)部分。

图 52.1. 指导规则设计器



规则的 WHEN 部分包含执行某个操作必须满足的条件。例如，如果银行需要 loan applicants 已有 21 余年，那么 Underage 规则的 WHEN 条件 比 | 21 少。

规则的ExpansionN 部分包含满足规则条件部分时要执行的操作。例如，当 loan applicant 低于 21 年时，cu N 操作将 批准 设置为 false，因为申请者处于年龄之内。

您还可以为更复杂的规则指定例外，例如，如果银行在涉及保守者时可以批准出不便的申请者。在这种情况下，您可以创建或导入 guarantor 数据对象，然后将该字段添加到指导规则中。

7. 在定义了规则的所有组件后，在指导规则设计器的右上角点击 Validate 来验证指导的规则。如果规则验证失败，解决错误消息中描述的任何问题，查看规则中的所有组件，然后重试验证规则直到规则通过为止。

8. 在指导规则设计器中点击 Save 保存您的更改。

52.1. 在指导规则中添加 WHEN 条件

规则的 WHEN 部分包含执行某个操作必须满足的条件。例如，如果银行需要 loan applicants 已有 21 余年，那么 Underage 规则的 WHEN 条件 比 | 21 少。您可以设置简单或复杂的条件，以确定您的规则是如何应用的。

先决条件

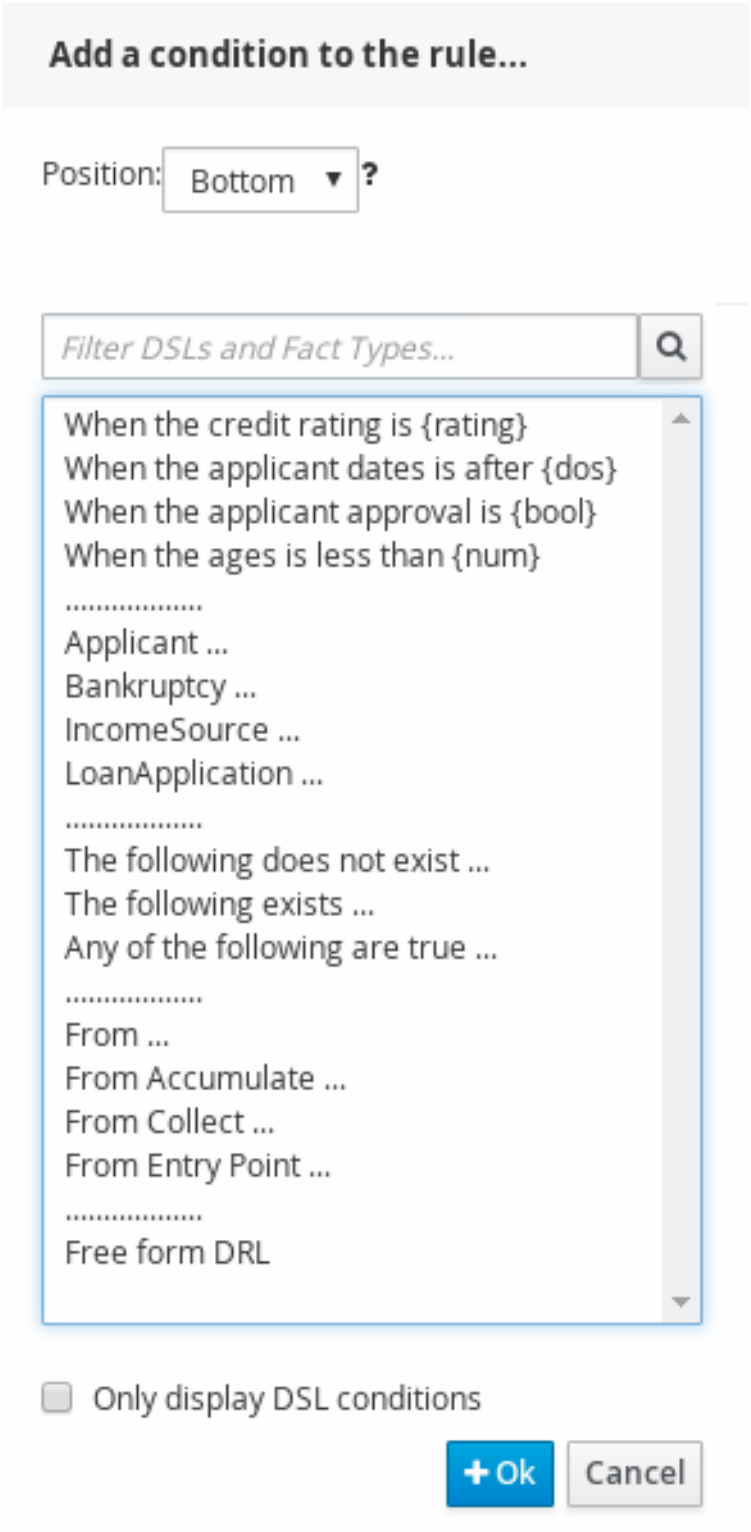
- 您的规则所需的所有数据对象已创建或导入，并列在指导规则设计器 的数据对象 选项卡中。

流程

1. 在指导规则设计程序中，点击 WHEN 部分右侧的加号图标  。

   将打开可用条件元素 的规则窗口中的 Add a condition。

图 52.2. 为规则添加一个条件



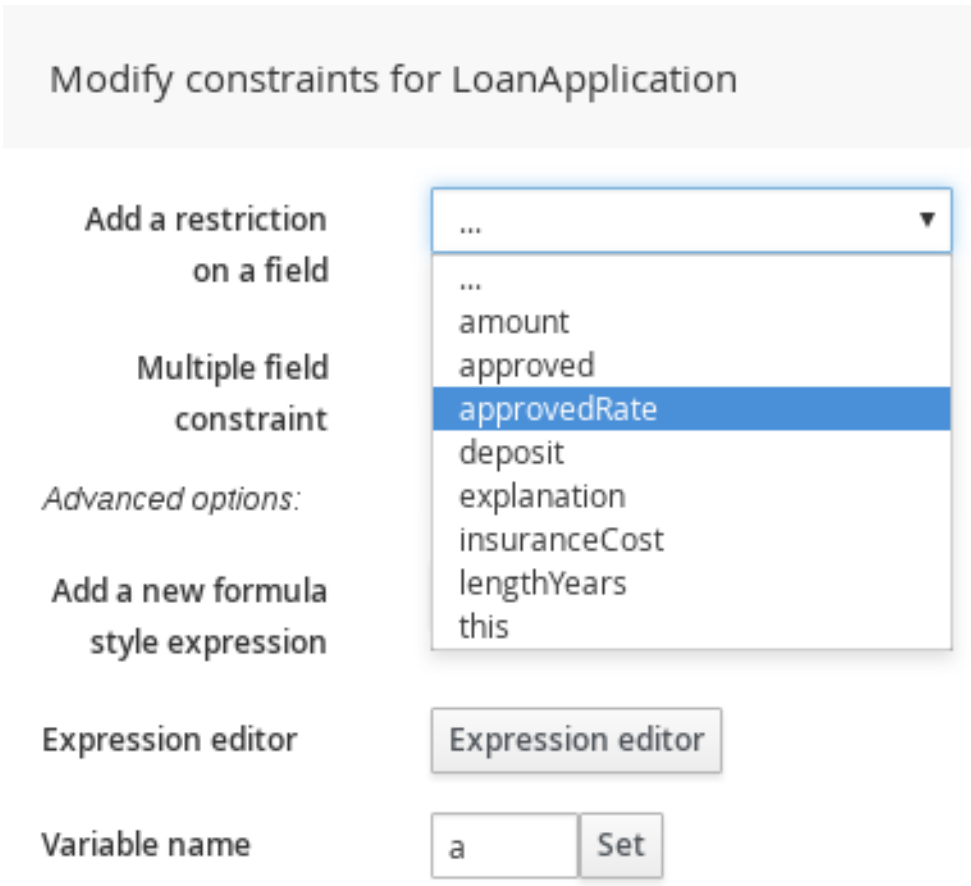
列表中包含指导规则设计器的 Data Objects 选项卡中的数据对象、为软件包定义的 DSL 对象（如果您在创建本指南规则时选择了 Show 声明 DSL 句），以及以下标准选项：

- 以下内容不存在：使用它指定必须不存在的事实和约束。
- 存在：使用它来指定必须存在的事实和限制。这个选项仅在第一个匹配项中触发，而不是后续匹配项。
- 以下任意一个为 true：使用此列出必须满足的任何事实或约束。
- from：使用它来为规则定义 From 条件元素。
- 选修日期：使用此命令 定义规则的 仲裁条件元素。
- from Collect：使用这个定义规则的 Collect 条件元素。
- From Entry Point：使用这个项来定义模式的 Entry Point。
- 自由形式 DRL：使用这个项插入自由格式的 DRL 字段，您可以在其中自由定义条件元素，而无需指导规则设计人员。

2. 选择一个 condition 元素（如 LoanApplication）并单击确定。

3. 单击指导规则设计器中的 condition 元素，并使用 LoanApplication 窗口中的 Modify 约束来添加字段、应用多个字段限制、添加新的公式风格表达式、应用表达式编辑器或设置变量名称。

图 52.3. 修改条件



注意

变量名称可让您识别指导规则中的其他结构中的事实或字段。例如，您可以将 LoanApplication 的变量设置为 a，然后引用一个单独的 Bankruptcy 约束，指定银行所基于的应用程序。

```
a : LoanApplication()
Bankruptcy( application == a ).
```

选择约束后，窗口会自动关闭。

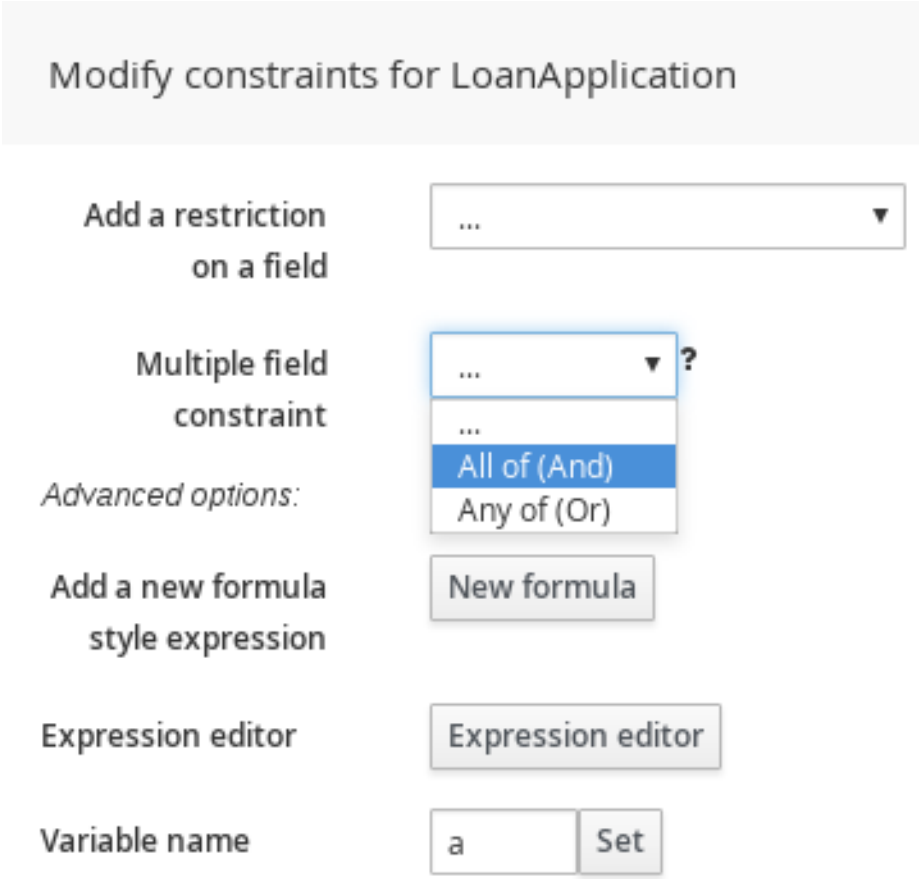
4. 从添加的限值旁边的下拉菜单中选择限制（例如，大于）的 operator。

5. 点编辑图标(

)定义字段值。字段值可以是字面值、一个公式或完整的 MVEL 表达式。

6. 要应用多个字段约束，点 LoanApplication 窗口的 Modify 约束，从多个字段 约束下拉菜单中选择 All of(And) 或 any (Or)。

图 52.4. 添加多个字段限制



7. 单击指导规则设计人员中的约束，进一步定义字段值。

8. 在定义了规则的所有条件组件后，在指导规则设计器的右上角点击 Validate 来验证指导的规则条件。如果规则验证失败，解决错误消息中描述的任何问题，查看规则中的所有组件，然后重试验证规则直到规则通过为止。

9. 在指导规则设计器中点击 Save 保存您的更改。

52.2. 在指导规则中添加功能

规则的 THEN 部分包含满足规则 WHEN 条件时要执行的操作。例如，当 loan applicant 低于 21 年时，cu N 操作可能会将 批准 设置为 false，因为申请者处于年龄之下。您可以设置简单或复杂的操作来确定如何应用规则。

先决条件

- ***您的规则所需的所有数据对象已创建或导入，并列在指导规则设计器 的数据对象 选项卡中。***

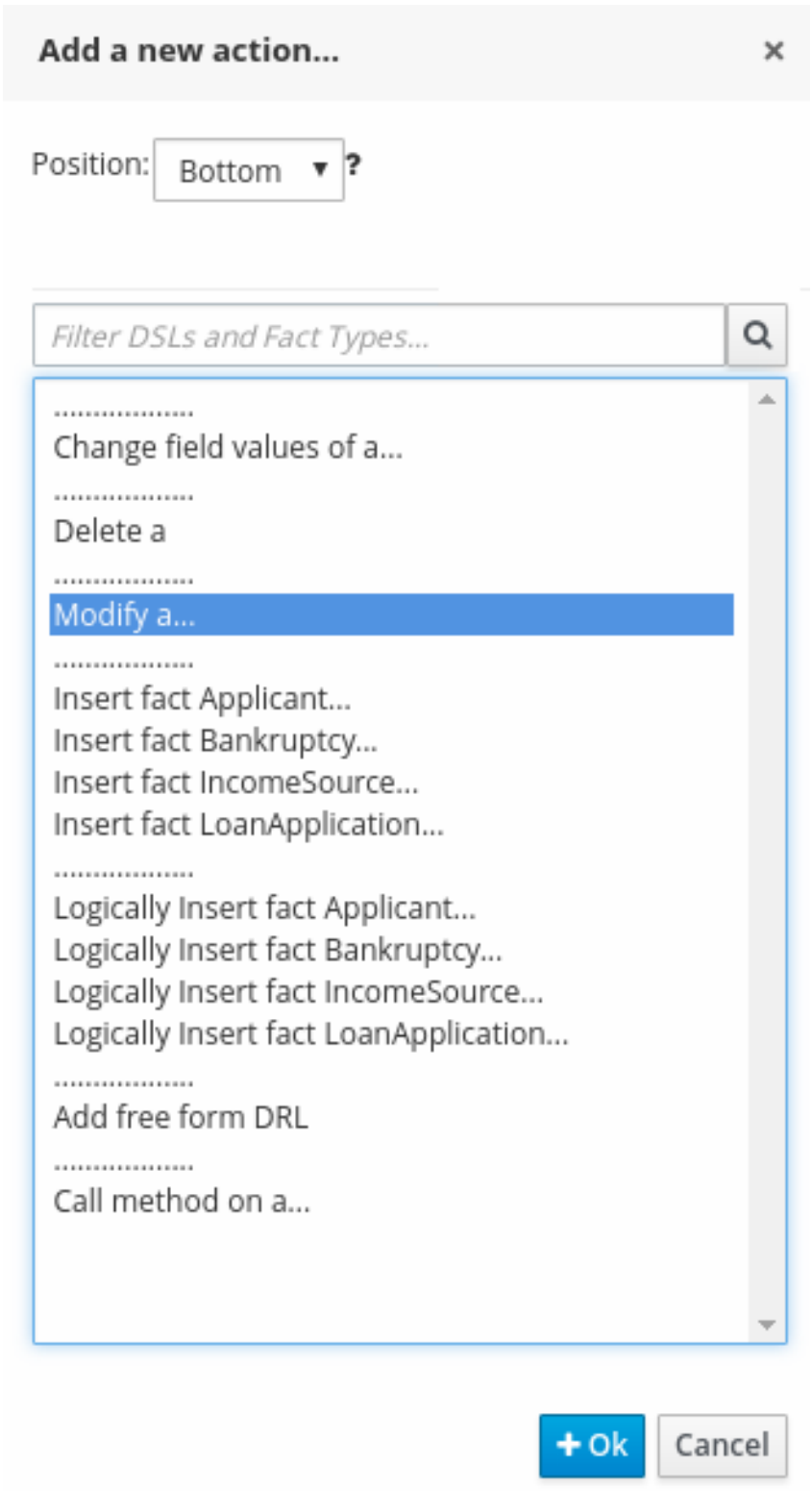
流程

1. ***在指导规则设计器中，点击 the the rightN 部分右侧的加号图标***

。

   此时会打开 Add a new action 窗口，其中带有可用的操作元素。

图 **52.5.** 为规则**添加新操作**



列表中包含基于指导规则设计**器的数据**对**象以及**为软**件包定**义**的 DSL** 对**象 （如果您在**创**建此指**导规则时显**示所 声明的 DSL** 项**）的数据**对**象的插入和修改**选项**：**

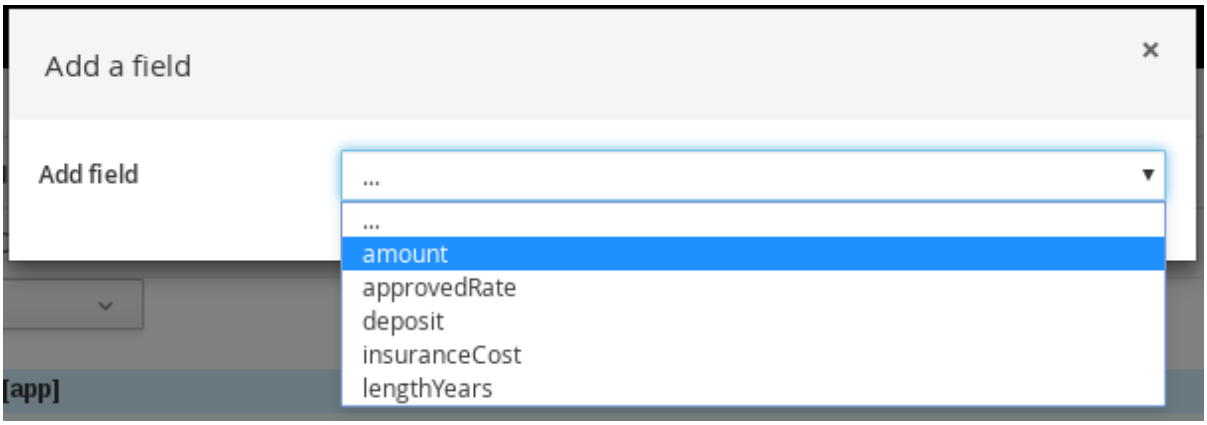
- **更改字段**值**： 使用此命令在事**实**（如 LoanApplication）上**设**置字段**值**，而不通知更改的决策引擎。**

- 删除：使用这个事实删除。
- 修改：使用这个方法指定要修改的字段，并通知更改的决策引擎。
- 插入事实：使用它插入事实，并为事实定义生成的字段和值。
- 逻辑插入事实：使用此事实将事实逻辑插入到决策引擎中，并为事实定义结果字段和值。决策引擎负责对插入和检索事实的逻辑决策。在常规或声明插入后，必须明确指定事实。逻辑插入后，当最初断言的事实不再为 true 时，会自动重新处理事实。
- 添加自由表格 DRL：使用此项来插入自由定义条件元素的免费 DRL 字段，而无需指导规则设计程序。
- 调用方法：使用此方法从另一事实调用方法。

2. 选择一个 action 元素（如 Modify）并点击 Ok。

3. 单击指导规则设计器中的 action 元素，并使用 Add a field 窗口来选择字段。

图 52.6. 添加一个字段



选择了字段后，窗口会自动关闭。

4. 点编辑图标(


)定义字段值。字段值可以是字面值或一个公式。

5. 在定义了规则的所有操作组件后，在指导规则设计器的右上角点击 Validate 来验证指导的规则操作。如果规则验证失败，解决错误消息中描述的任何问题，查看规则中的所有组件，然后重试验证规则直到规则通过为止。

6. 在指导规则设计器中点击 Save 保存您的更改。

52.3. 在规则资产中定义下拉列表的枚举数

Business Central 中的枚举定义决定了指导规则、指导规则模板和指导决策表中的条件或操作字段的可能值。enumeration 定义包含一个 fact.field 映射到支持值列表，该列表显示为规则资产相关字段中的下拉列表。当用户选择基于与枚举定义相同的事实和字段的字段时，则会显示定义值的下拉列表。

您可以在 Business Central 中，或者在 Red Hat Process Automation Manager 项目的 DRL 源中定义枚举器。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。

2. 点 Add Asset → Enumeration。

3. 输入参考 名称 并选择相应的 软件包。您指定的软件包必须是分配或分配所需数据对象和相关规则资产的同一软件包。

4. 单击 Ok 以创建枚举。

   新的 enumeration 现已列在 Project Explorer 的 Enumeration Definitions 面板中。

5. 在 enumerations designer 的 Model 选项卡中，点 Add enum，并为枚举定义以下值：

   - 事实 ：指定您项目同一软件包中的现有数据对象，您要将此枚举关联。在 Project Explorer 中打开 Data Objects 面板，以查看可用的数据对象，或根据需要创建相关的数据

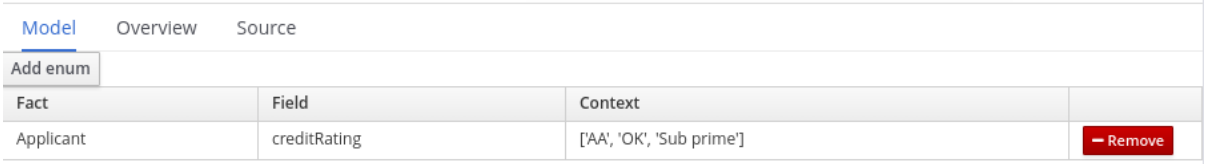
对象作为新资产。

- **字段**：指定一个现有字段标识符，作为您为事实选择的 data 对象的一部分定义。在 Project Explorer 中打开 Data Objects 面板，以选择相关的数据对象并查看可用标识符选项列表。如果需要，您可以为 data 对象创建相关标识符。

- **context**：指定一个值列表，格式为 ['string1','string2','string3'] 或 [integer1,integer2,integer3]，您想要映射到 Fact 和 Field 定义。这些值将显示为规则资产中相关字段的下拉列表。

例如，以下枚举值定义了 loan 应用决策服务中 applicant 贡献等级的下拉菜单：

图 52.7. Business Central 中申请信贷评级的枚举示例



DRL 源中应用贡献度评级的枚举示例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```

在本例中，对于任何指导规则、指导规则模板或指导决策表，这些表格位于项目的同一软件包中，并使用 Applicant 数据对象和 creditRating 字段，配置的值作为下拉列表提供：

图 52.8. 指导规则或指导规则模板中的 enumeration 下拉列表示例

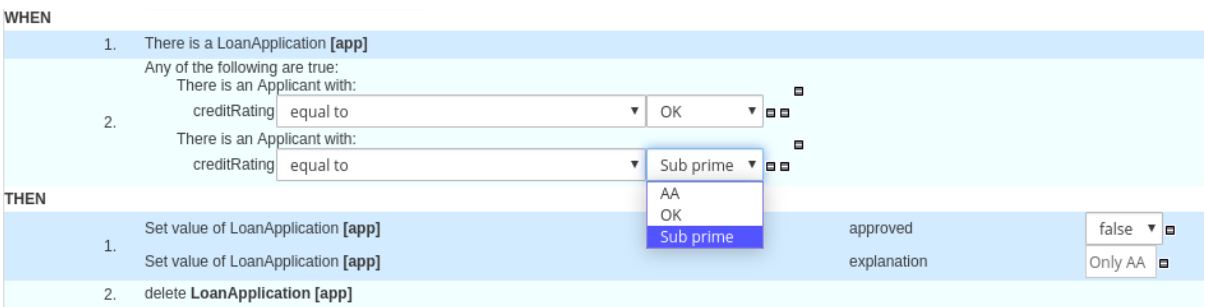
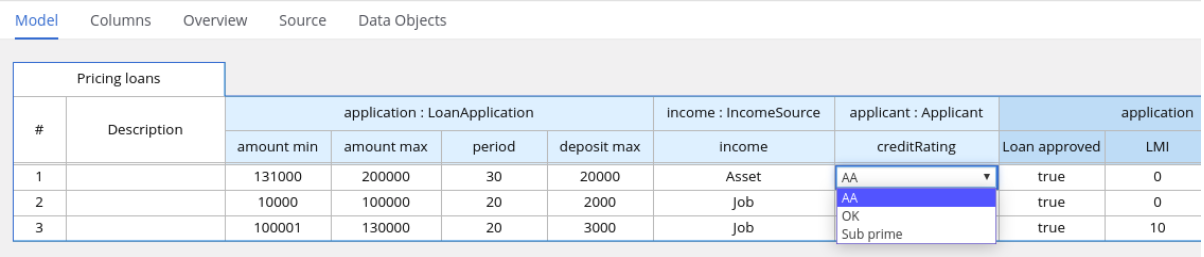


图 52.9. 指导决策表中的枚举下拉列表示例



52.3.1. 规则资产的高级枚举选项

对于红帽流程自动化管理器项目中的枚举定义的高级用例，请考虑以下扩展选项来定义枚举：

Business Central 中的 DRL 值和值之间的映射

如果您希望 enumeration 值显示在 Business Central 界面中的不同于 DRL 源中显示的不同或更全面的值，使用格式 "fact.field" : ['sourceValue1=UIValue1','sourceValue2=UIValue2','sourceValue2=UIValue2', ...] 用于您的 enumeration 值。

例如，在 loan status 的以下枚举定义中，选项 A 或 D 会在 DRL 文件中使用，但选项已批准或拒绝在 Business Central 中：

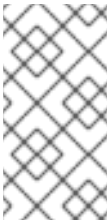
```
'Loan.status' : ['A=Approved','D=Declined']
```

枚举值依赖项

如果您希望在一个下拉列表中选择的值，以决定后续下拉列表中的可用选项，请使用 'fact.fieldB[fieldA=value1]' : ['value2', 'value3', 'value3', ...] 用于您的 enumeration 定义。

例如，在保险政策的以下枚举定义中，policyType 字段接受 Home 或 Car 的值。用户选择的策略类型决定了可用的策略 覆盖 字段选项：

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```



注意

在规则条件和操作中不应用枚举依赖关系。例如，在这个保险策略用例中，规则条件中的所选策略不会决定规则操作中的可用覆盖范围选项（如果适用）。

枚举中的外部数据源

如果要从外部数据源检索枚举值列表，而不是直接在枚举定义中定义值，请在项目的类路径上添加一个 helper 类，该类返回 java.util.List 字符串列表。在枚举定义中，而不是指定值列表，标识您配置为从外部检索值的帮助程序类。

例如，在 loan applicant 区域的以下枚举定义中，而不是以 "Applicant.region" 格式明确定义：['country1', 'country2', ...], enumeration 使用一个帮助类返回外部定义的值列表：

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

在本例中，DataHelper 类包含一个 getListOfRegions（） 方法，它返回字符串列表。在规则资产中相关字段的下拉列表中载入 enumerations。

您还可以将依赖的 enumeration 定义从 helper 类中动态加载，方法是将依赖字段标识为引号内帮助程序类的调用范围：

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}")'
```

如果要完全从外部数据源（如相关数据库）加载所有枚举的数据，则可实施返回 Map<String、List<String> 映射的 Java 类。映射的键是 fact.field 映射，值是值的 java.util.List<String> 列表。

例如，以下 Java 类为相关的枚举定义了 loan applicant 区域：

```
public class SampleDataSource {

  public Map<String, List<String>> loadData() {
    Map data = new HashMap();

    List d = new ArrayList();
    d.add("AU");
    d.add("DE");
    d.add("ES");
    d.add("UK");
    d.add("US");
    ...
    data.put("Applicant.region", d);

    return data;
  }

}
```

以下枚举定义与本例 Java 类相关。枚举不包含对事实或字段名称的引用，因为它们是在 Java 类中定义：

```
=(new SampleDataSource()).loadData()
```

= 运算符使 Business Central 能够加载 helper 类中的所有枚举数据。当请求枚举定义以便在编辑器中使用时，静态评估帮助程序方法。

注意

目前，Business Central 不支持在没有事实和字段定义的情况下定义枚举。要以这种方式为关联的 Java 类定义枚举值，请在 Red Hat Process Automation Manager 项目中使用 DRL 源。

52.4. 添加其他规则选项

您还可以使用规则设计器在规则中添加元数据，定义其他规则属性（比如 salience 和 no-loop），也可冻结规则的修改来限制条件或操作。

流程

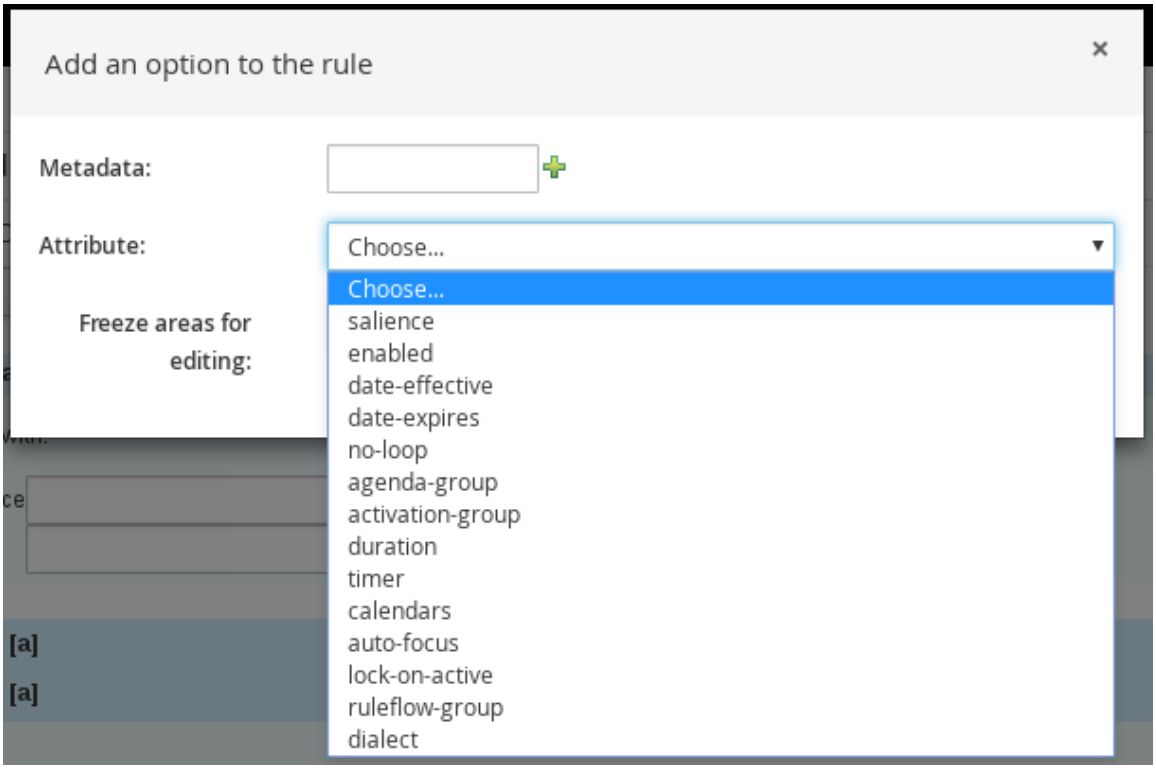
1. 在规则设计器中，点 (show options...) in the THEN 部分。
2. 点击窗口右侧的加号图标

来添加选项。
3. 选择要添加到规则中的选项：
   - Metadata ： 输入元数据标签并点击加号图标(

)。然后，在规则设计器提供的字段中输入所需的数据。
   - attribute： 从规则属性列表中选择。然后，在规则设计器中显示的字段或选项中定义值。

- 冻结区域进行编辑： 选择 Conditions 或 Actions，以限制在规则设计器中修改的区域。

图 52.10. 规则选项



4. 单击规则设计器中的 Save，以保存您的工作。

52.4.1. 规则属性

规则属性是您可以添加到业务规则中修改规则行为的额外规格。

下表列出了您可以分配给规则的属性的名称和支持值：

表 52.1. 规则属性

属性	值
salience	定义规则优先级的整数。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。 示例： **salience 10**

属性	值
enabled	布尔值。选择了 选项后，会启用该规则。如果没有选择 选项，该规则将被禁用。 示例： **enabled true**
date-effective	包含日期和时间定义的字符串。只有在当前日期和时间后面是 **date-effective** 属性后才能激活该规则。 示例： **日期为 "4-Sep-2018"**
date-expires	包含日期和时间定义的字符串。如果当前的日期和时间位于 **date-expires** 属性后，则无法激活该规则。 示例： **date-expires "4-Oct-2018"**
no-loop	布尔值。选择 选项时，如果规则触发之前满足条件，则无法重新激活该规则（循环）。如果没有选择条件，可以在这些情形中循环该规则。 示例： **no-loop true**
日程表（日程）	为您指定要为其分配该规则的日程表组的字符串。日程表组允许您对规则组进行更多执行控制。只有已获取焦点的管理者组中的规则才能够被激活。 示例： **日程-组 "GroupName"**
activation-group	您要为其分配该规则的激活（或 XOR）组的字符串。在激活组中，只能激活一条规则。第一条规则取消激活组中所有规则的待处理激活。 示例： **activation-group "GroupName"**
duration	如果规则条件仍满足，则用于定义在激活规则的时间持续时间（以毫秒为单位）的长整数值。 示例： **duration 10000**
timer	用于标识 **int** (interval)或 **cron** 计时器定义的字符串，用于调度规则。 示例： **timer(cron:* 0/15 * * ?)** （每 15 分钟）
日历	用于调度规则的 Quartz 日历定义。 示例： **calendars "* * 0-7,18-23 ?* *"** （排除非工作时间）
auto-focus	布尔值，仅适用于 schedule groups 中的规则。选择 选项时，下一次激活规则时，会自动把焦点分配给分配给该规则的日程表组。 示例： **auto-focus true**

属性	值
lock-on-active	布尔值，仅适用于规则流组或日程组中的规则。选择了 选项时，规则的 ruleflow 组下次变为活跃时间，或者规则的日程表组接收焦点，无法再次激活该规则，直到 ruleflow 组不再活跃，否则将失去焦点。这是 **no-loop** 属性的一个更强大的版本，因为无论更新的来源，都丢弃匹配规则的激活（而不只是规则本身）。此属性非常适合计算规则，其中有多个规则修改事实，而您不想再次匹配和触发任何规则。 示例： **lock-on-active true**
ruleflow-group	标识规则流组的字符串。在规则流组中，只有在相关规则流激活组时，规则才能触发。 示例： **ruleflow-group "GroupName"**
dialect	标识 **JAVA** 或 **MVEL** 的字符串，用作规则中代码表达式的语言。默认情况下，该规则使用在软件包级别上指定的断言。这里指定的任意分区会覆盖规则的软件包选择设置。 示例： **第一个"JAVA"**  **注意** 当您在没有可执行模型的情况下使用 Red Hat Process Automation Manager 时，**去选"JAVA"** 规则会导致只支持 Java 5 语法。有关可执行模型的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

第 53 章 执行规则

在确定了示例规则或在 Business Central 中创建自己的规则后，您可以在本地或 KIE 服务器上构建并部署相关的项目并在 KIE 服务器上执行规则，以测试规则。

先决条件

- 业务中心和 KIE 服务器已安装并运行。有关安装选项，请参阅 规划 Red Hat Process Automation Manager 安装。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 在项目 资产 页面右上角，单击 Deploy 以构建该项目，并将它部署到 KIE 服务器。如果构建失败，请解决屏幕底部的 Alerts 面板中描述的问题。

有关项目部署选项的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。

注意

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

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

默认情况下，Red Hat Process Automation Manager 中的规则资产需要这个依赖项，从可执行规则模型构建。这个依赖关系包含在 Red Hat Process Automation Manager 核心打包中，但取决于 Red Hat Process Automation Manager 升级历史记录，您可能需要手动添加这个依赖关系来启用可执行规则模型行为。

有关可执行规则模型的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。

3. 在 Business Central 之外创建 Maven 或 Java 项目（如果尚未创建），您可将其用于在本地执行规则，或用作在 KIE 服务器上执行规则的客户端应用程序。该项目必须包含 pom.xml 文件以及执行项目资源的任何其他必要组件。

   有关 test 项目示例，请参阅 "创建和执行 DRL 规则的方法 "。

4. 打开 test 项目或客户端应用程序的 pom.xml 文件，并添加以下依赖项（如果还没有添加）：

   - kie-ci ：启用客户端应用程序以使用 ReleaseId在本地加载 Business Central 项目数据
   - kie-server-client ：使您的客户端应用程序能够与 KIE 服务器上的资产远程交互
   - slf4j:（可选）使您的客户端应用程序能够使用 Simple Logging Facade 用于 Java(SLF4J)，以在与 KIE 服务器交互后返回调试信息

   客户端应用程序 pom.xml 文件中的 Red Hat Process Automation Manager 7.13 的依赖项示例：

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

   对于这些工件的可用版本，请在 Nexus Repository Manager 在线搜索组 ID 和工件 ID。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

5. 确保在客户端应用 pom.xml 文件中定义了包含模型类的工件依赖项，它们与部署的项目的 pom.xml 文件中完全相同。如果模型类的依赖关系因客户端应用和项目之间有所不同，则可能会出现执行错误。

要访问 Business Central 中的项目 pom.xml 文件，请在项目左侧选择任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

例如，以下 Person 类依赖项同时出现在客户端和部署的项目 pom.xml 文件中：

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. 如果您将 slf4j 依赖项添加到用于调试日志的客户端应用程序 pom.xml 文件，请在相关类路径（例如，在 Maven 中的 src/main/resources/META-INF）上创建一个 simplelogger.properties 文件（例如，在 Maven 中的 src/main/resources/META-INF）：

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

-

7. 在您的客户端应用中，创建一个包含必要导入和 main（） 方法的 .java 主类，以加载 KIE 基础、插入事实和执行规则。

例如，项目中的 Person 对象包含 getter 和 setter 方法，用于设置并检索一个人的名字、姓氏、每小时速率和分流。项目中的以下 Wage 规则计算 wage 和 hourly 速率值，并根据结果显示消息：

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

要在 KIE 服务器之外测试此规则（如果需要），请配置 .java 类以导入 KIE 服务、KIE 容器和 KIE 会话，然后使用 main（） 方法针对定义的事实模型触发所有规则：

本地执行规则

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

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

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

      // Set up the fact model:
      Person p = new Person();
```

```
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert the person into the session:
        kSession.insert(p);

        // Fire all rules:
        kSession.fireAllRules();
        kSession.dispose();
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
  }
```

要在 KIE 服务器上测试此规则，请配置 .java 类，其导入和规则执行信息与本地示例类似，另外还指定 KIE 服务配置和 KIE 服务客户端详情：

在 KIE 服务器上执行规则

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {
```

```
  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert Person into the session:
      KieCommands kieCommands = KieServices.Factory.get().getCommands();
      List<Command> commandList = new ArrayList<Command>();
      commandList.add(kieCommands.newInsert(p, "personReturnId"));

      // Fire all rules:
      commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
      BatchExecutionCommand batch =
kieCommands.newBatchExecution(commandList, sessionName);

      // Use rule services client to send request:
      RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
      ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
      System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

8. 从项目目录中运行配置的 .java 类。您可以在开发平台（如 Red Hat CodeReady Studio）或命令行中运行该文件。

Maven 执行示例（在项目目录中）：

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

Java 执行示例（在项目目录中）

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

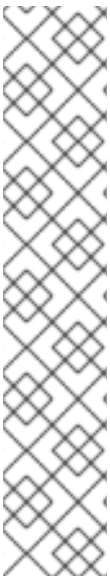
9. 在命令行中检查规则执行状态并在服务器日志中。如果有任何规则没有按预期执行，请检查项目中配置的规则，以及验证提供的数据的主类配置。

第 54 章 后续步骤

- 使用测试场景测试决策服务
- 打包和部署 Red Hat Process Automation Manager 项目

部分 VII. 使用指导规则模板设计决策服务

作为业务分析员或业务规则开发人员，您可以使用 Business Central 中的指导规则模板设计人员定义规则模板。这些指导规则模板为编译为 Drools Rule Language(DRL)的多个规则提供了一个可重复使用的规则结构，并为您的项目形成决策服务的核心。



注意

您还可以使用决策模型和 Notation(DMN)模型设计您的决策服务，而不是基于规则或基于表的资产。有关 Red Hat Process Automation Manager 7.13 中的 DMN 支持的详情，请查看以下资源：

- 开始使用决策服务 （逐步教程，带有 DMN 决策服务示例）
- 使用 DMN 模型设计决策服务 (Red Hat Process Automation Manager 中的 DMN 支持和功能视图)

先决条件

- Business Central 中创建了指导规则模板的空间和项目。每个资产都与分配给一个空间的项目相关联。详情请参阅 开始使用决策服务。

第 55 章 红帽流程自动化管理器中的决策资产

Red Hat Process Automation Manager 支持一些可用于为决策服务定义业务决策的资产。每个决策资产都有不同的优势，您可以根据目标和需求，首选使用一个或多个资产的组合。

下表重点介绍红帽流程自动化管理器项目支持的主要决策资产，以帮助您决定或确认在决策服务中定义决策的最佳方法。

表 55.1. Red Hat Process Automation Manager 支持的决策资产

asset	亮点	编写工具	Documentation
决策模型和符号 (DMN)型号	• 决策模型基于由对象管理组 (OMG)定义的符号标准。 • 使用代表部分或所有决策要求图 (DRG)的图形化决策要求图 (DRG)跟踪业务决策流程 • 使用 XML 模式，允许在 DMN 兼容平台间共享 DMN 模型 • 支持 Friendly Enough Expression Language(FEEL)，在 DMN 决策表中定义决策逻辑，以及其他 DMN 框的表达式 • 可以与业务流程模型和符号 (BPMN)流程模型有效集成 • 是创建全面、清晰和稳定的决策流程的最佳选择	Business Central 或其他与 DMN 兼容的编辑器	*使用 DMN 模型设计决策服务*
指导的决定表	• 是您在 Business Central 中基于 UI 的表格设计者创建的规则表 • 是电子表格决策表的向导替代选择 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值 • 支持点击策略、实时验证和其他资产不支持的其他功能 • 是以受控表格格式创建规则的最佳选择，可最小化编译错误	Business Central	*使用指导决策表设计决策服务*

asset	亮点	编写工具	Documentation
电子表格决策表	• 是 XLS 或 XLSX 表格决策表，您可以上传到 Business Central • 支持创建规则模板的模板键和值 • 是创建已在 Business Central 之外管理的决策表中规则的最佳选择 • 在上传后，对规则进行编译具有严格的语法要求	电子表格编辑器	*使用电子表格决策表设计决策服务*
指导规则	• 是您在 Business Central 中基于 UI 的规则设计者创建的单独规则 • 为可接受的输入提供字段和选项 • 是以受控格式创建单一规则的最佳选择，以最小化编译错误	Business Central	*使用指导规则设计决策服务*
指导规则模板	• 是您在 Business Central 中基于 UI 的模板设计者创建的可重复使用的规则结构 • 为可接受的输入提供字段和选项 • 支持创建规则模板的模板键和值（与此资产的目的不相同） • 是创建具有相同规则结构但具有不同定义字段值的很多规则的最佳	Business Central	*使用指导规则模板设计决策服务*
DRL 规则	• 是直接在 **.drl** 文本文件中定义的单独规则 • 提供定义规则和其他规则行为的最大的灵活性 • 可以在特定的独立环境中创建，并与 Red Hat Process Automation Manager 集成 • 是创建需要高级 DRL 选项的规则的最佳选择 • 对规则进行正确编译的规则具有严格的语法要求	业务中心或集成开发环境(IDE)	*使用 DRL 规则设计决策服务*

asset	亮点	编写工具	Documentation
预测模型标记语言 (PMML)型号	• 是基于数据 Mining Group(DMG) 定义的标记标准预测数据分析模型。 • 使用允许 PMML 模式在 PMML 兼容的平台之间共享的 XML 模式 • 支持 Regression、scorecard、Tree、Mining 和其他模型类型 • 可以包含在独立 Red Hat Process Automation Manager 项目中，或者导入到 Business Central 中的项目 • 在 Red Hat Process Automation Manager 中，将预测数据合并为决策服务的最佳选择	PMML 或 XML 编辑器	*使用 PMML 模型设计决策服务*

在定义决策时，您还可以考虑将红帽构建的 Kogito 构建用于云原生决策服务。有关使用红帽构建的 Kogito 微服务的更多信息，请参阅 Red Hat Process Automation Manager 中的 Red Hat build of Kogito。

第 56 章 指导规则模板

指导规则模板是具有占位符值（模板键）的业务规则结构，它们与独立数据表中定义的实际值进行修改。该模板的对应数据表中定义的值每行都会产生规则。如果许多规则具有相同的条件、操作和其他属性，但事实或约束的值有所不同，则指导规则模板是理想的选择。在这种情况下，您可以创建一个适用于每个规则的规则，而不是创建很多相似的指南规则规则模板，而是在数据表中定义不同的值。

指导规则模板设计程序根据定义的规则模板的数据对象以及添加模板键值的对应数据表提供可接受的模板输入字段和选项。创建指南规则模板并在相应的数据表中添加值后，您定义的规则与所有其他规则资产一样编译为 Drools 规则语言(DRL)规则。

与指导规则模板相关的所有数据对象都必须位于与指导规则模板相同的项目软件包中。默认导入同一软件包中的资产。创建必要的数据对象和指导规则模板后，您可以使用指南规则模板设计器 的数据对象 选项卡来验证所有所需的数据对象是否被列出或导入其他现有数据对象，方法是添加新项目。

第 57 章 数据对象

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

57.1. 创建数据对象

以下流程是创建数据对象的一般概述。它并不适用于特定的业务资产。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Data Object。
3. 输入唯一数据对象名称，然后选择您希望数据对象可用于其他规则资产的软件包。 同一名称的数据对象不能位于同一软件包中。在指定的 DRL 文件中，您可以从任何软件包中导入数据对象。

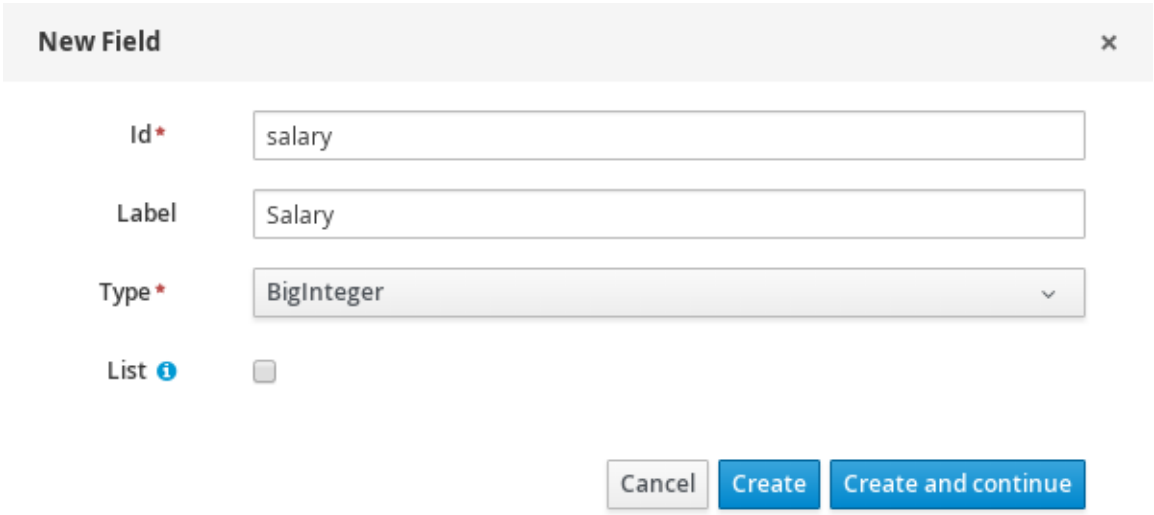
从其他软件包导入数据对象

您可以将另一软件包中的现有数据对象直接导入到资产设计人员，如指导规则或指导决策表设计人员。在项目中选择相关规则资产以及资产设计器，请转至 Data Objects → New item 以选择要导入的对象。

4. 要使数据对象持久化，请选择"永久 "复选框。Persistable 数据对象可以根据 JPA 规格存储在数据库中。默认的 JPA 为 Hibernate。
5. 点 确定。
6. 在数据对象设计器中，点 add 字段在 对象中添加包含属性 Id、标签和 类型的字段。所需的属性标有星号(*)。
   - id： 输入字段的唯一 ID。

- **label:** （可选）输入字段的标签。
- **Type：** 输入字段的数据类型。
- **列表 ：** （可选）选择此复选框，以启用字段保存指定类型的多个项目。

图 57.1. 在数据对象中添加数据字段



7. 点 Create 添加新字段，或者点击 Create 并继续添加新字段并继续添加其他字段。

注意

要编辑字段，请选择字段行，并使用屏幕右侧的 常规属性。

第 58 章 创建指导规则模板

您可以使用指导规则模板来定义与数据表中定义的实际值对应的占位符值（模板键）的规则结构。指导规则模板是一种有效替代方法，可单独定义许多使用相同结构的指南规则集。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。

2. 点 Add Asset → Guided Rule Template。

3. 输入 参考规则模板名称 并选择相应的 软件包。您指定的软件包必须是分配或将被分配所需数据对象的同一软件包。

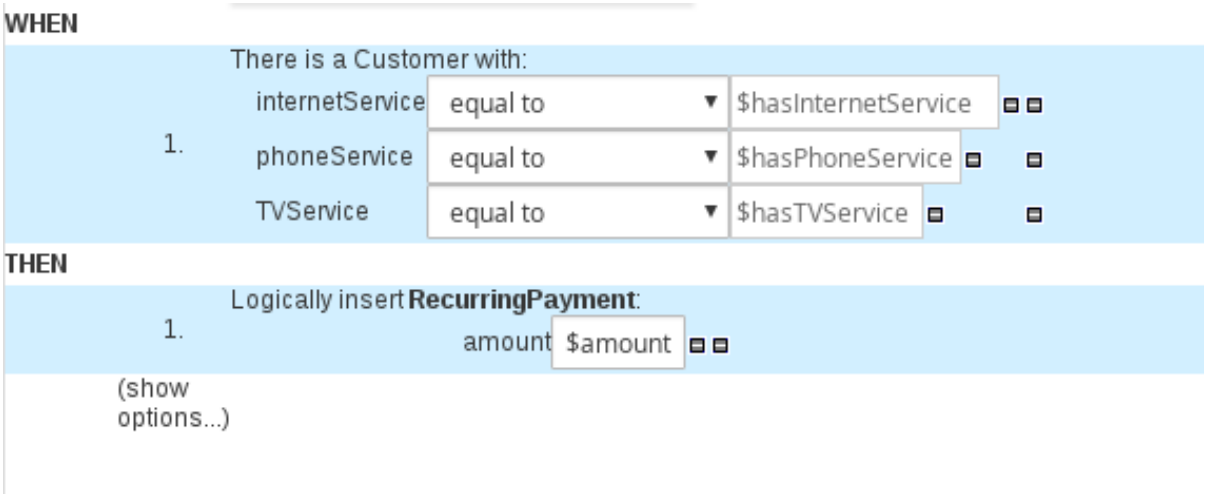
4. 单击 Ok 以创建规则模板。

   新的指导规则模板现在列在 Project Explorer 的 《规则模板 "面板中。

5. 点 Data Objects 选项卡，并确认列出了您的规则所需的所有数据对象。如果没有，请单击 New item 以从其他软件包导入数据对象，或者在您的软件包 中创建数据对象。

6. 在所有数据对象都就位后，返回 Model 选项卡，并使用窗口右侧的按钮来添加和定义规则模板的 WHEN （条件）和(action)部分，具体取决于可用数据对象。对于不同的字段值，以规则设计器格式使用 $key 格式的模板键，或者以自由形式为 DRL（如果使用）格式使用 @{key}。

图 58.1. 指导规则模板示例



模板键的备注

模板键是指导规则模板中的基本模板。模板键是模板中的启用字段值与您在对应数据表中定义的实际值进行更改，以便从同一模板生成不同的规则。对于基于该模板的所有规则结构的一部分的值结构，您可以使用其他值（如 Literal 或 Formula ）。但是，如果规则中的任何不同值，请使用带有指定键的 Template 键字段类型。在指导规则模板中没有模板键，相应的数据表不会在模板设计器中生成，模板基本上作为单独的指导规则。

规则模板的 WHEN 部分是必须满足以下条件才能执行操作的条件。例如，如果通信公司根据自己订阅的服务（互联网、电话和电视）为客户收费，那么 WHEN 条件之一是 internetService | 等于 | $haInternetService。模板键 $hasInternetService 使用模板 data 表中定义的实际布尔值值（true 或 false）进行交集。

该规则模板的一部分是满足规则条件部分时要执行的操作。例如，如果客户只为互联网服务订阅，则模板键 $amount 对 RecurringPayment 的 RecurringPayment 的操作会将实际每月数量设置为数据表中为互联网服务收费定义的整数值。

7. 在定义了规则的所有组件后，在指导规则模板设计器中点击 Save 以保存您的工作。

58.1. 在指导规则模板中添加 WHEN 条件

规则的 WHEN 部分包含执行某个操作必须满足的条件。例如，如果通信公司根据自己订阅的服务（互联网、电话和电视）为客户收费，那么 WHEN 条件之一是 internetService | 等于 | $haInternetService。模板键 $hasInternetService 使用模板 data 表中定义的实际布尔值值（true 或 false）进行交集。

先决条件

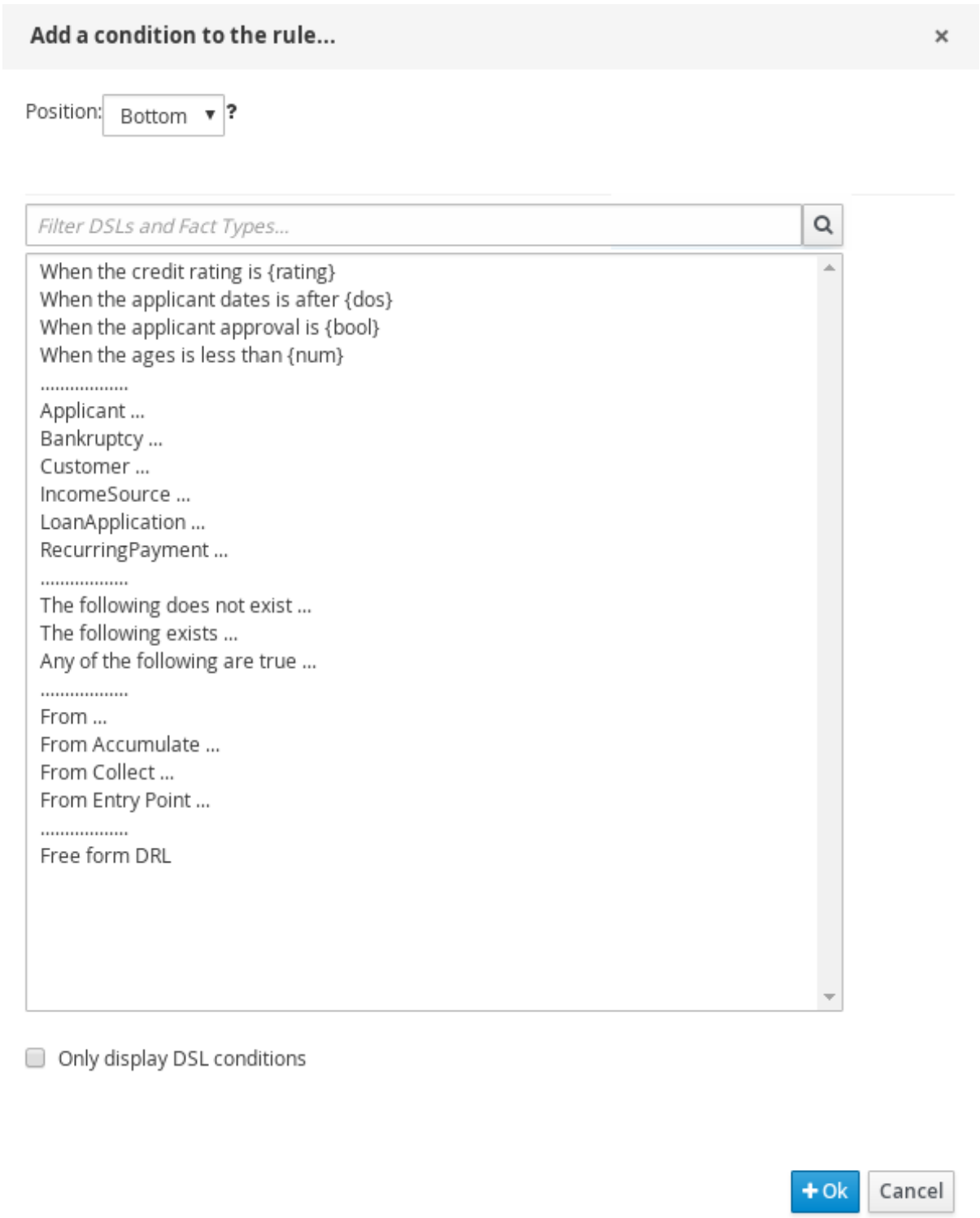
- 您的规则所需的所有数据对象已创建或导入，并列在指导规则模板设计器 的数据对象 选项卡中。

流程

1. 在指导规则模板设计器中，点击 WHEN 部分右侧的加号图标  。

   将打开可用条件元素 的规则窗口中的 Add a condition。

图 58.2. 为规则添加一个条件



列表中包含指导规则模板设计器的 Data Objects 选项卡中的数据对象、为软件包定义的任何 DSL 对象，以及以下标准选项：

- 以下内容不存在：使用它指定必须不存在的事实和约束。
- 存在：使用它来指定必须存在的事实和限制。这个选项仅在第一个匹配项中触发，而不

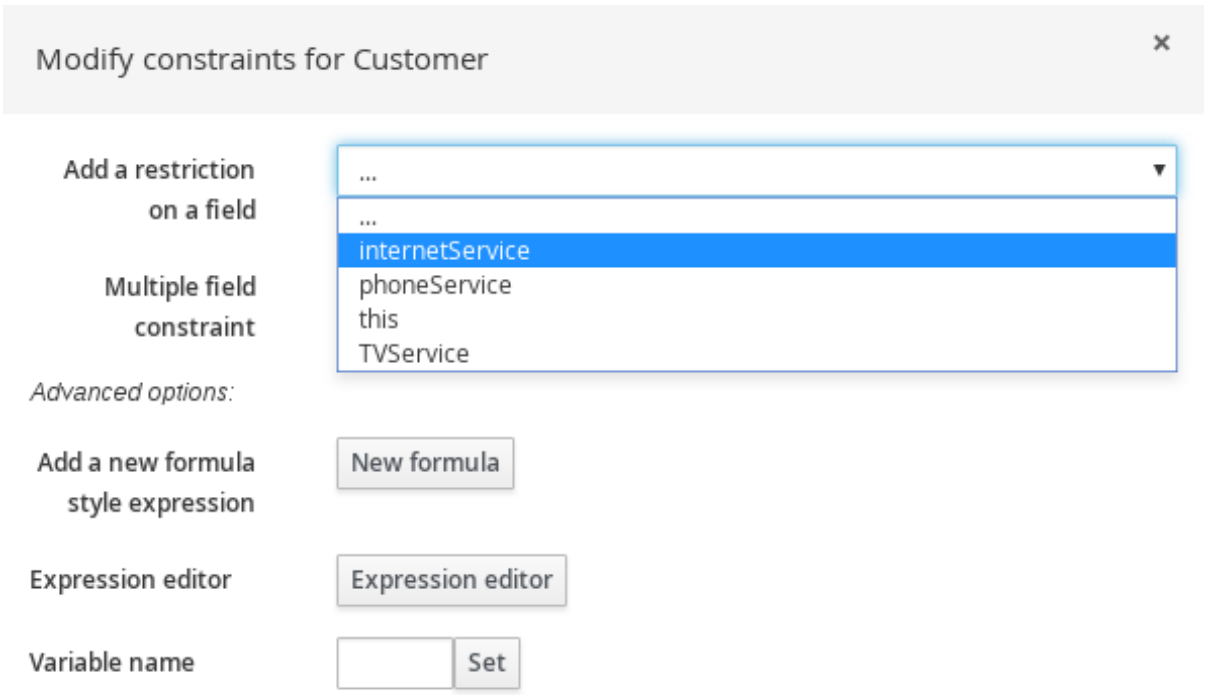
是后续匹配项。

- 以下任意一个为 true： 使用此列出必须满足的任何事实或约束。
- from： 使用它来为规则定义 From 条件元素。
- 选修日期： 使用此命令 定义规则的 仲裁条件元素。
- from Collect： 使用这个定义规则的 Collect 条件元素。
- From Entry Point： 使用这个项来定义模式的 Entry Point。
- 自由形式 DRL： 使用这个项插入自由格式的 DRL 字段，您可以在其中自由定义条件元素，而无需指导规则设计人员。对于自由形式 DRL 的模板键，请使用 @{key} 格式。

2. 选择一个 condition 元素（例如，客户），然后单击确定。

3. 单击提示规则模板设计器中的 condition 元素，并使用 Customer 窗口中的 Modify 约束 添加字段、应用多个字段约束、添加新的公式风格表达式、应用表达式编辑器或设置变量名称。

图 58.3. 修改条件



注意

变量名称可让您识别指导规则中的其他结构中的事实或字段。例如，您可以将客户的变量设置为 c，然后在单独的 Applicant 约束下引用 c，指定 客户 是 Applicant。

```
c : Customer()
Applicant( this == c )
```

选择约束后，窗口会自动关闭。

4. 从添加限制旁边的下拉菜单中选择限制（例如，等于）的 operator。

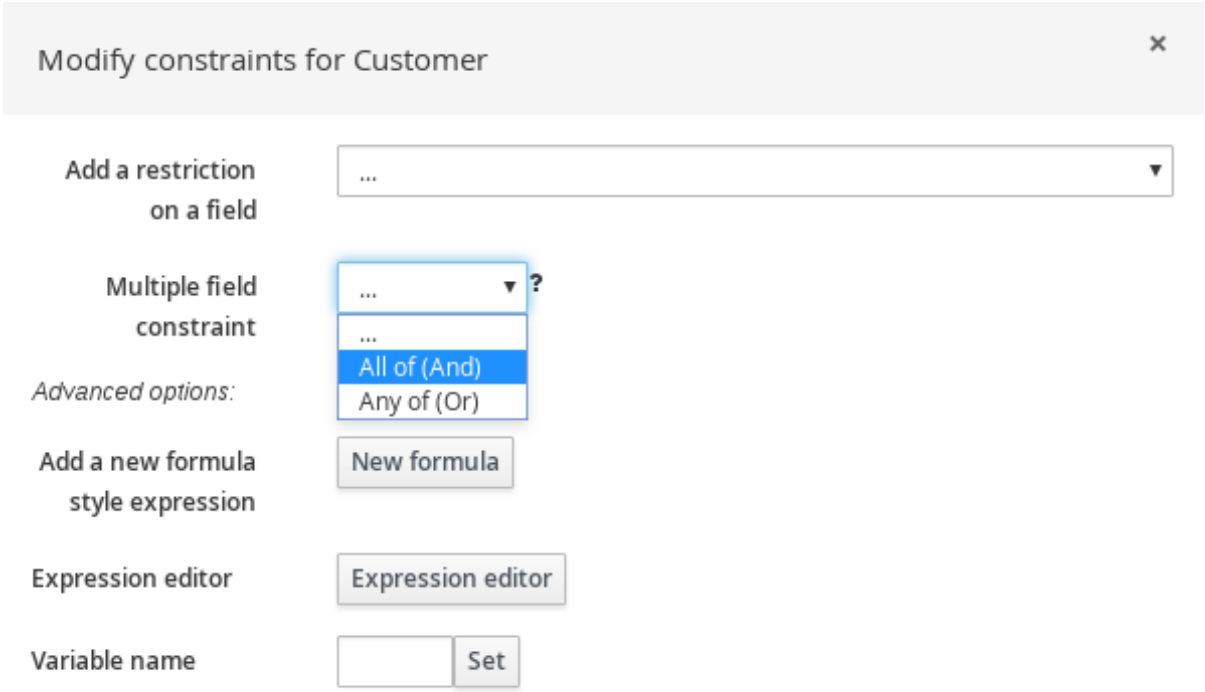
5. 点编辑图标(

)定义字段值。

6. 如果此值因基于此模板的规则的不同而不同，请选择 Template 键，并以 $key 格式添加模板键。这允许字段值使用您在对应数据表中定义的实际值来从同一模板生成不同的规则。对于与规则不同且不是规则模板一部分的字段值，您可以使用任何其他值类型。

7. *要应用多个字段约束，点 客户窗口的修改限制，从 多字段约束 下拉菜单中选择 All of(And) 或任意 (Or)。*

图 58.4. 添加多个字段限制



8. *单击指导规则模板设计器中的约束，进一步定义字段值。*

9. *在定义所有条件元素后，在指导规则模板设计器中点击 Save 以保存您的工作。*

58.2. 在指导规则模板中添加特征操作

该规则模板的一部分是满足规则条件部分时要执行的操作。例如，如果客户只为互联网服务订阅，则模板键 $amount 对 RecurringPayment 的 RecurringPayment 的操作会将实际每月数量设置为数据表中为互联网服务收费定义的整数值。

先决条件

- *您的规则所需的所有数据对象已创建或导入，并列在指导规则模板设计器 的数据对象 选项卡中。*

流程

1. *在指导规则模板设计器中，点击 the the rightN 部分右侧的加号图标(*

)。

此时会打开 **Add a new action** 窗口，其中带有可用的操作元素。

图 **58.5.** 为规则**添加新操作**

该**列表包含基于指**导规则**模板**设计**器的数据**对**象以及**为软**件包定**义**的任何 DSL** 对**象 的数据**对**象的插入和修改**选项**：**

- **插入事实：使用它插入事实，并为事实定义生成的字段和值。**

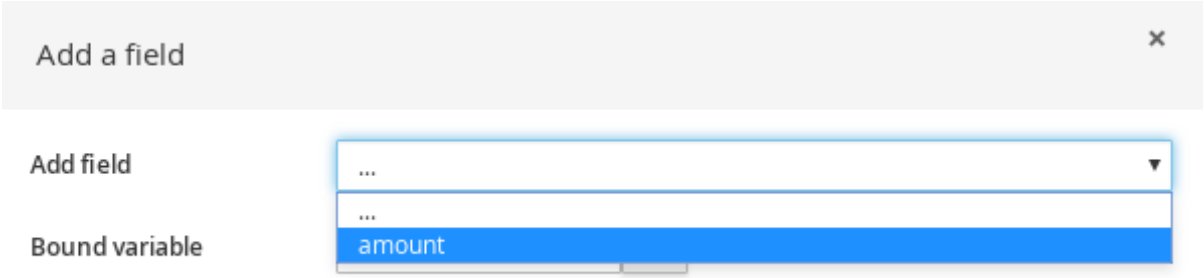
- 逻辑**插入事实：使用此事实将事实逻辑插入到决策引擎中，并为事实定义结果字段和值。决策引擎负责对插入和检索事实的逻辑决策。在常规或声明插入后，必须明确指定事实。逻辑插入后，当最初断言的事实不再为 true 时，会自动重新处理事实。**

- **添加自由表格 DRL：使用此项来插入自由定义条件元素的免费 DRL 字段，而无需指导规则设计程序。对于自由形式 DRL 的模板键，请使用 @{key} 格式。**

2. 选择**一个 action 元素（例如，logically Insert fact RecurringPayment）并**单击 **Ok。**

3. 单击**指导规则模板设计器中的 action 元素，并使用 Add a field 窗口来选择字段。**

图 **58.6. 添加一个字段**



选择**了字段后，窗口会自动关闭。**

4. **点**编辑图标**(**

)定义字段值。

5. **如果此值因基于此模板的规则的不同而不同，**请选择 **Template** 键，**并以 $key 格式添加模板键。这允许字段值使用您在对应数据表中定义的实际值来从同一模板生成不同的规则。对于与规则不同且不是规则模板一部分的字段值，您可以使用任何其他值类型。**

6. **在定义所有操作元素后，在指导规则模板设计器中点击 Save 以保存您的工作。**

58.3. 在规则资产中定义下拉列表的枚举数

Business Central 中的枚举定义决定了指导规则、指导规则模板和指导决策表中的条件或操作字段的可能值。enumeration 定义包含一个 fact.field 映射到支持值列表，该列表显示为规则资产相关字段中的下拉列表。当用户选择基于与枚举定义相同的事实和字段的字段时，则会显示定义值的下拉列表。

您可以在 Business Central 中，或者在 Red Hat Process Automation Manager 项目的 DRL 源中定义枚举器。

流程

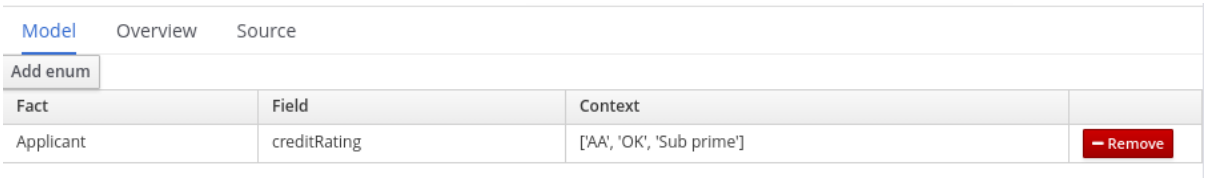
1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Enumeration。
3. 输入参考 名称 并选择相应的 软件包。您指定的软件包必须是分配或分配所需数据对象和相关规则资产的同一软件包。
4. 单击 Ok 以创建枚举。

   新的 enumeration 现已列在 Project Explorer 的 Enumeration Definitions 面板中。
5. 在 enumerations designer 的 Model 选项卡中，点 Add enum，并为枚举定义以下值：

   - 事实 ：指定您项目同一软件包中的现有数据对象，您要将此枚举关联。在 Project Explorer 中打开 Data Objects 面板，以查看可用的数据对象，或根据需要创建相关的数据对象作为新资产。
   - 字段 ：指定一个现有字段标识符，作为您为 事实 选择的 data 对象的一部分定义。在 Project Explorer 中打开 Data Objects 面板，以选择相关的数据对象并查看可用 标识符选项列表。如果需要，您可以为 data 对象创建相关标识符。
   - context ：指定一个值列表，格式为 ['string1','string2','string3'] 或 [integer1,integer2,integer3]，您想要映射到 Fact 和 Field 定义。这些值将显示为规则资产中相关字段的下拉列表。

例如，以下枚举值定义了 loan 应用决策服务中 applicant 贡献等级的下拉菜单：

图 58.7. Business Central 中申请信贷评级的枚举示例



DRL 源中应用贡献度评级的枚举示例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```

在本例中，对于任何指导规则、指导规则模板或指导决策表，这些表格位于项目的同一软件包中，并使用 Applicant 数据对象和 creditRating 字段，配置的值作为下拉列表提供：

图 58.8. 指导规则或指导规则模板中的 enumeration 下拉列表示例

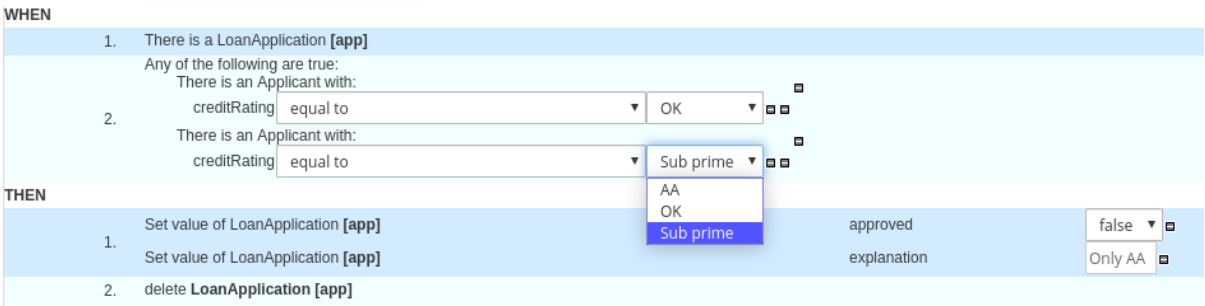
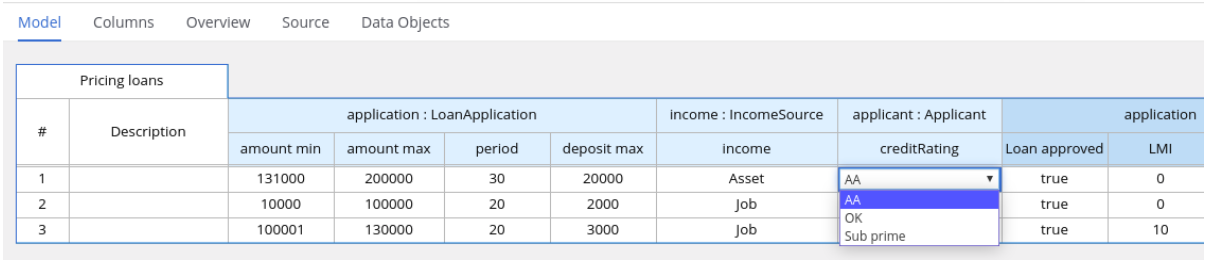


图 58.9. 指导决策表中的枚举下拉列表示例



58.3.1. 规则资产的高级枚举选项

对于红帽流程自动化管理器项目中的枚举定义的高级用例，请考虑以下扩展选项来定义枚举：

Business Central 中的 DRL 值和值之间的映射

如果您希望 enumeration 值显示在 Business Central 界面中的不同于 DRL 源中显示的不同或更全面的值，使用格式 "fact.field" : ['sourceValue1=UIValue1','sourceValue2=UIValue2','sourceValue2=UIValue2', ...] 用于您的 enumeration 值。

例如，在 loan status 的以下枚举定义中，选项 A 或 D 会在 DRL 文件中使用，但选项已批准或拒绝在 Business Central 中：

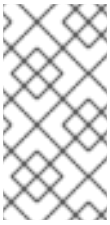
```
'Loan.status' : ['A=Approved','D=Declined']
```

枚举值依赖项

如果您希望在一个下拉列表中选择的值，以决定后续下拉列表中的可用选项，请使用 'fact.fieldB[fieldA=value1]' : ['value2', 'value3', 'value3', ...] 用于您的 enumeration 定义。

例如，在保险政策的以下枚举定义中，policyType 字段接受 Home 或 Car 的值。用户选择的策略类型决定了可用的策略 覆盖 字段选项：

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```



注意

在规则条件和操作中不应用枚举依赖关系。例如，在这个保险策略用例中，规则条件中的所选策略不会决定规则操作中的可用覆盖范围选项（如果适用）。

枚举中的外部数据源

如果要从外部数据源检索枚举值列表，而不是直接在枚举定义中定义值，请在项目的类路径上添加一个 helper 类，该类返回 java.util.List 字符串列表。在枚举定义中，而不是指定值列表，标识您配置为从外部检索值的帮助程序类。

例如，在 loan applicant 区域的以下枚举定义中，而不是以 "Applicant.region" 格式明确定义：['country1', 'country2', ...], enumeration 使用一个帮助类返回外部定义的值列表：

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

在本例中，DataHelper 类包含一个 getListOfRegions（） 方法，它返回字符串列表。在规则资产中相关字段的下拉列表中载入 enumerations。

您还可以将依赖的 enumeration 定义从 helper 类中动态加载，方法是将依赖字段标识为引号内帮助程序类的调用范围：

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}")'
```

如果要完全从外部数据源（如相关数据库）加载所有枚举的数据，则可实施返回 Map<String、List<String> 映射的 Java 类。映射的键是 fact.field 映射，值是值的 java.util.List<String> 列表。

例如，以下 Java 类为相关的枚举定义了 loan applicant 区域：

```
public class SampleDataSource {

  public Map<String, List<String>> loadData() {
    Map data = new HashMap();

    List d = new ArrayList();
    d.add("AU");
    d.add("DE");
    d.add("ES");
    d.add("UK");
    d.add("US");
    ...
    data.put("Applicant.region", d);

    return data;
  }

}
```

以下枚举定义与本例 Java 类相关。枚举不包含对事实或字段名称的引用，因为它们是在 Java 类中定义：

```
=(new SampleDataSource()).loadData()
```

= 运算符使 Business Central 能够加载 helper 类中的所有枚举数据。当请求枚举定义以便在编辑器中使用时，静态评估帮助程序方法。

注意

目前，Business Central 不支持在没有事实和字段定义的情况下定义枚举。要以这种方式为关联的 Java 类定义枚举值，请在 Red Hat Process Automation Manager 项目中使用 DRL 源。

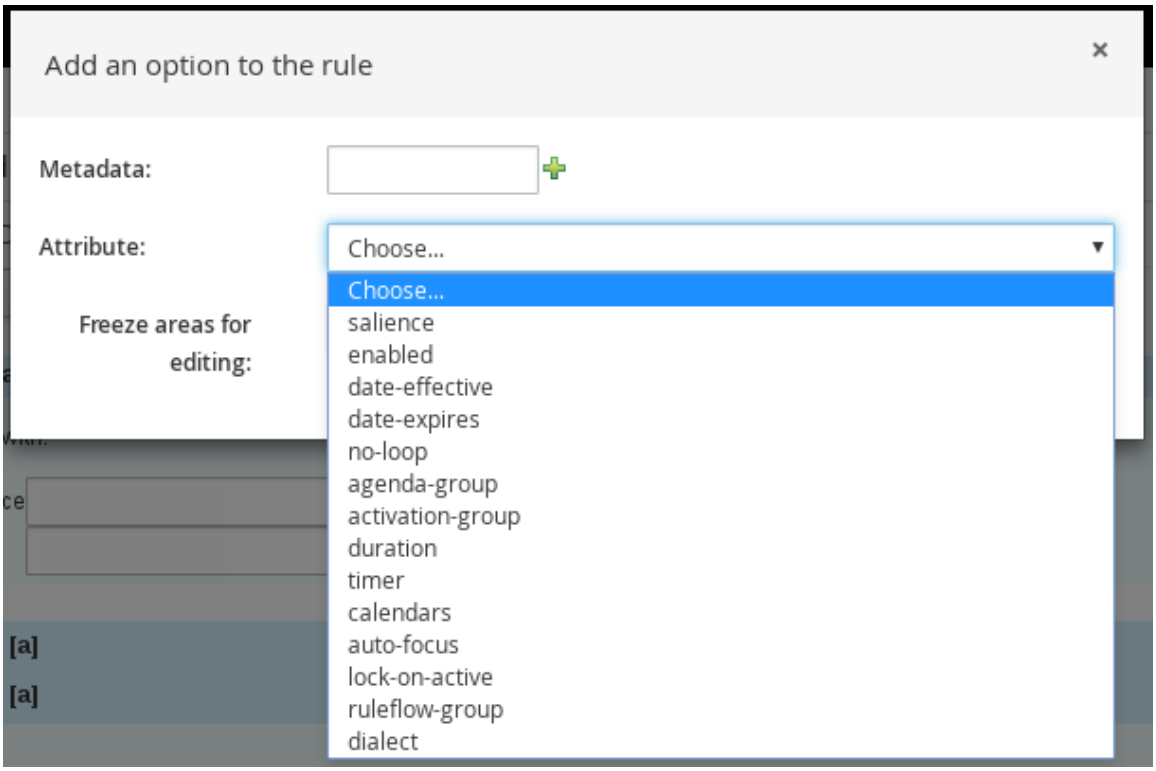
58.4. 添加其他规则选项

您还可以使用规则设计器在规则中添加元数据，定义其他规则属性（比如 salience 和 no-loop），也可冻结规则的修改来限制条件或操作。

流程

1. 在规则设计器中，点 (show options...) in the THEN 部分。
2. 点击窗口右侧的加号图标  来添加选项。
3. 选择要添加到规则中的选项：
   - Metadata ： 输入元数据标签并点击加号图标()。然后，在规则设计器提供的字段中输入所需的数据。
   - attribute： 从规则属性列表中选择。然后，在规则设计器中显示的字段或选项中定义值。
   - 冻结区域进行编辑： 选择 Conditions 或 Actions，以限制在规则设计器中修改的区域。

图 **58.10.** 规则选项



4. 单击规则设计器中的 **Save**，以保存您的工作。

58.4.1. 规则属性

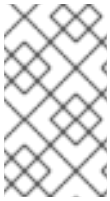
规则属性是您可以添加到业务规则中修改规则行为的额外规格。

下表列出了您可以分配给规则的属性的名称和支持值：

表 58.1. 规则属性

属性	值
salience	定义规则优先级的整数。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。 示例：**salience 10**
enabled	布尔值。选择了 选项后，会启用该规则。如果没有选择 选项，该规则将被禁用。 示例：**enabled true**

属性	值
date-effective	包含日期和时间定义的字符串。只有在当前日期和时间后面是 **date-effective** 属性后才能激活该规则。 示例： **日期为 "4-Sep-2018"**
date-expires	包含日期和时间定义的字符串。如果当前的日期和时间位于 **date-expires** 属性后，则无法激活该规则。 示例： **date-expires "4-Oct-2018"**
no-loop	布尔值。选择 选项时，如果规则触发之前满足条件，则无法重新激活该规则（循环）。如果没有选择条件，可以在这些情形中循环该规则。 示例： **no-loop true**
日程表（日程）	为您指定要为其分配该规则的日程表组的字符串。日程表组允许您对规则组进行更多执行控制。只有已获取焦点的管理者组中的规则才能够被激活。 示例： **日程-组 "GroupName"**
activation-group	您要为其分配该规则的激活（或 XOR）组的字符串。在激活组中，只能激活一条规则。第一条规则取消激活组中所有规则的待处理激活。 示例： **activation-group "GroupName"**
duration	如果规则条件仍满足，则用于定义在激活规则的时间持续时间（以毫秒为单位）的长整数值。 示例： **duration 10000**
timer	用于标识 **int** (interval)或 **cron** 计时器定义的字符串，用于调度规则。 示例： **timer(cron:* 0/15 * * ?)** （每 15 分钟）
日历	用于调度规则的 Quartz 日历定义。 示例： **calendars "* * 0-7,18-23 ?* *"** （排除非工作时间）
auto-focus	布尔值，仅适用于 schedule groups 中的规则。选择 选项时，下一次激活规则时，会自动把焦点分配给分配给该规则的日程表组。 示例： **auto-focus true**
lock-on-active	布尔值，仅适用于规则流组或日程组中的规则。选择了 选项时，规则的 ruleflow 组下次变为活跃时间，或者规则的日程表组接收焦点，无法再次激活该规则，直到 ruleflow 组不再活跃，否则将失去焦点。这是 **no-loop** 属性的一个更强大的版本，因为无论更新的来源，都丢弃匹配规则的激活（而不只是规则本身）。此属性非常适合计算规则，其中有多个规则修改事实，而您不想再次匹配和触发任何规则。 示例： **lock-on-active true**

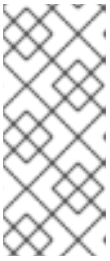
属性	值
ruleflow-group	标识规则流组的字符串。在规则流组中，只有在相关规则流激活组时，规则才能触发。 示例： **ruleflow-group "GroupName"**
dialect	标识 **JAVA** 或 **MVEL** 的字符串，用作规则中代码表达式的语言。默认情况下，该规则使用在软件包级别上指定的断言。这里指定的任意分区会覆盖规则的软件包选择设置。 示例： **第一个"JAVA"**  **注意** 当您在没有可执行模型的情况下使用 Red Hat Process Automation Manager 时，**去选"JAVA"** 规则会导致只支持 Java 5 语法。有关可执行模型的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

第 59 章 定义指导规则模板的数据表

创建指导规则模板并添加字段值的模板键后，指导规则模板设计器的数据表中会显示数据表。数据表中的每个列都对应于您在指导规则模板中添加的模板键。使用此表格按行为每个模板键行定义值。您在该模板的 data 表中定义的值每行都会生成规则。

流程

1. 在指导规则模板设计器中，单击 **Data** 选项卡以查看数据表。数据表中的每个列都对应于您在指导规则模板中添加的模板键。



注意

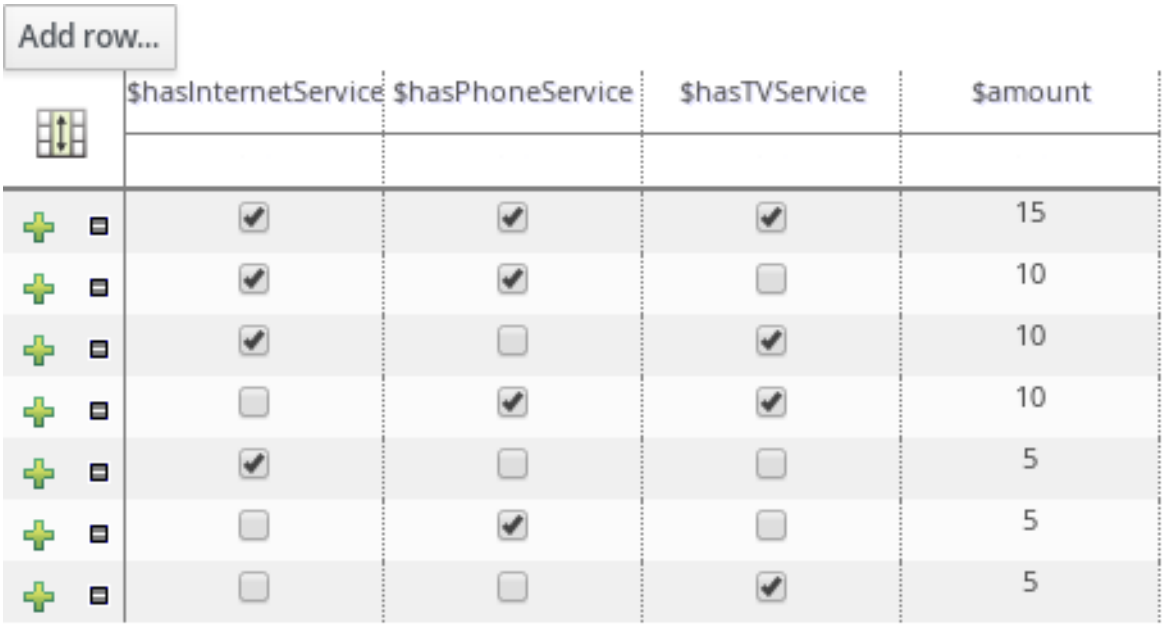
如果您没有在规则模板中添加任何模板键，则不会显示此数据表，且模板无法作为 genuine 模板，但基本上作为单独的指导规则。因此，在创建指导规则模板的过程中，模板键是基本的。

2. 单击 **Add row**，再定义每个模板键列的 **data** 值来生成该规则（行）。

3. 继续添加行并定义所生成的每个规则的数据值。您可以点每个新 行的 **Add** 行，或者点击加号图标

或减号图标来添加或删除行。

图 **59.1.** 指导规则模板的数据表示例



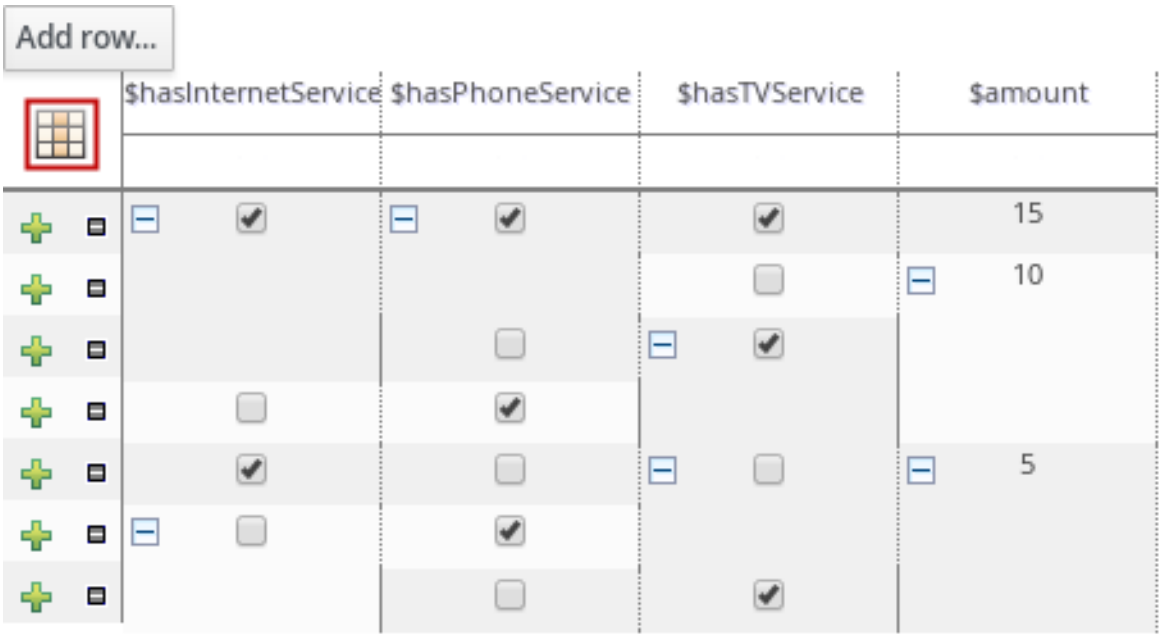
要查看 DRL 代码，请点击指导规则模板设计器中的 Source 选项卡。

例如：

```
rule "PaymentRules_6"
 when
  Customer( internetService == false ,
   phoneService == false ,
   TVService == true )
 then
  RecurringPayment fact0 = new RecurringPayment();
  fact0.setAmount( 5 );
  insertLogical( fact0 );
end

rule "PaymentRules_5"
 when
  Customer( internetService == false ,
   phoneService == true ,
   TVService == false )
 then
  RecurringPayment fact0 = new RecurringPayment();
  fact0.setAmount( 5 );
  insertLogical( fact0 );
end
...
//Other rules omitted for brevity.
```

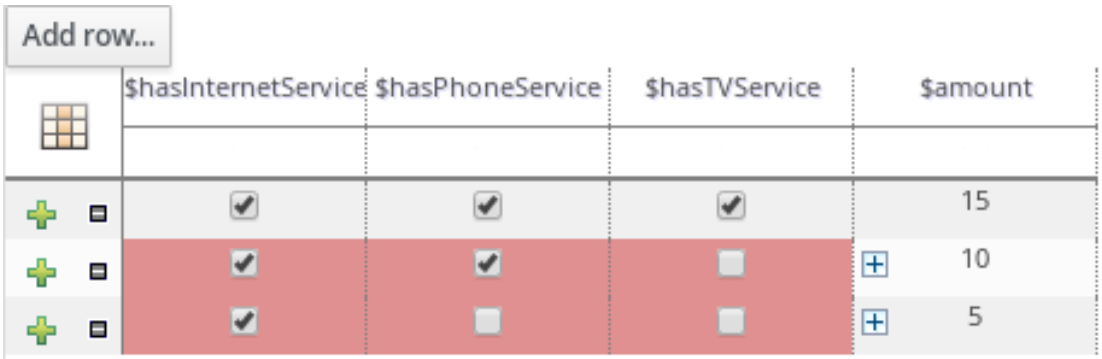
4. 作为视觉辅助，请单击数据表左上角的网格图标来切换单元合并（如果需要）。同一列中的单元格将合并到一个单元格中。

*图 **59.2. 数据表中合并**单元*



然后，您可以使用每个新合并**单元左上角的展开/折**叠图标 **[+/-] 折**叠与合并**单元**对应**的行，或者重新关**闭**的行。

*图 **59.3. 折**叠**同步**单元*



5.

在**为**所有**规则**定**义**模板密**钥**数据并**调**整表**显**示后，**单击**指南**规则**模板**设计**程序右上角的 Validate 以**验证**指**导规则**模板。如果**规则**模板**验证**失**败，**解决**错误**消息中描述的任何**问题，查**看**规则**模板和数据表中定**义**的所有**组**件，然后重**试验证规则**模板直到**规则**模板通**过为**止。

6.

在指**导规则**模板**设计**器中点**击 Save 保存您的更改。

第 60 章 执行规则

在确定了示例规则或在 Business Central 中创建自己的规则后，您可以在本地或 KIE 服务器上构建并部署相关的项目并在 KIE 服务器上执行规则，以测试规则。

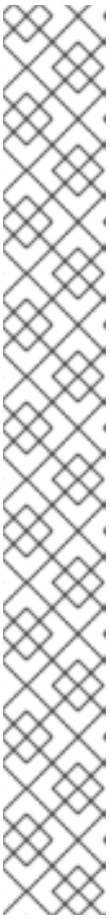
先决条件

- 业务中心和 KIE 服务器已安装并运行。有关安装选项，请参阅 规划 Red Hat Process Automation Manager 安装。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 在项目 资产 页面右上角，单击 Deploy 以构建该项目，并将它部署到 KIE 服务器。如果构建失败，请解决屏幕底部的 Alerts 面板中描述的问题。

有关项目部署选项的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。



注意

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

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

默认情况下，Red Hat Process Automation Manager 中的规则资产需要这个依赖项，从可执行规则模型构建。这个依赖关系包含在 Red Hat Process Automation Manager 核心打包中，但取决于 Red Hat Process Automation Manager 升级历史记录，您可能需要手动添加这个依赖关系来启用可执行规则模型行为。

有关可执行规则模型的更多信息，请参阅 打包和部署 Red Hat Process Automation Manager 项目。

3. 在 Business Central 之外创建 Maven 或 Java 项目（如果尚未创建），您可将其用于在本地执行规则，或用作在 KIE 服务器上执行规则的客户端应用程序。该项目必须包含 pom.xml 文件以及执行项目资源的任何其他必要组件。

   有关 test 项目示例，请参阅 "创建和执行 DRL 规则的方法 "。

4. 打开 test 项目或客户端应用程序的 pom.xml 文件，并添加以下依赖项（如果还没有添加）：

   - kie-ci ：启用客户端应用程序以使用 ReleaseId在本地加载 Business Central 项目数据
   - kie-server-client ：使您的客户端应用程序能够与 KIE 服务器上的资产远程交互
   - slf4j:（可选）使您的客户端应用程序能够使用 Simple Logging Facade 用于 Java(SLF4J)，以在与 KIE 服务器交互后返回调试信息

   客户端应用程序 pom.xml 文件中的 Red Hat Process Automation Manager 7.13 的依赖项示例：

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.67.0.Final-redhat-00024</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

   对于这些工件的可用版本，请在 Nexus Repository Manager 在线搜索组 ID 和工件 ID。

注意

考虑将 Red Hat Business Automation 材料清单(BOM)依赖项添加到项目 pom.xml 文件，而不是为每个依赖项指定 Red Hat Process Automation Manager < version >。Red Hat Business Automation BOM 适用于 Red Hat Decision Manager 和 Red Hat Process Automation Manager。添加 BOM 文件时，项目中包含来自提供的 Maven 存储库的正确依赖项版本。

BOM 依赖项示例：

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.13.5.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

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

5. 确保在客户端应用 pom.xml 文件中定义了包含模型类的工件依赖项，它们与部署的项目的 pom.xml 文件中完全相同。如果模型类的依赖关系因客户端应用和项目之间有所不同，则可能会出现执行错误。

要访问 Business Central 中的项目 pom.xml 文件，请在项目左侧选择任何现有资产，然后在屏幕左侧的 Project Explorer 菜单中，单击 Customize View gear 图标并选择 Repository View → pom.xml。

例如，以下 Person 类依赖项同时出现在客户端和部署的项目 pom.xml 文件中：

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. 如果您将 slf4j 依赖项添加到用于调试日志的客户端应用程序 pom.xml 文件，请在相关类路径（例如，在 Maven 中的 src/main/resources/META-INF）上创建一个 simplelogger.properties 文件（例如，在 Maven 中的 src/main/resources/META-INF）：

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

-

7. 在您的客户端应用中，创建一个包含必要导入和 main（） 方法的 .java 主类，以加载 KIE 基础、插入事实和执行规则。

例如，项目中的 Person 对象包含 getter 和 setter 方法，用于设置并检索一个人的名字、姓氏、每小时速率和分流。项目中的以下 Wage 规则计算 wage 和 hourly 速率值，并根据结果显示消息：

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

要在 KIE 服务器之外测试此规则（如果需要），请配置 .java 类以导入 KIE 服务、KIE 容器和 KIE 会话，然后使用 main（） 方法针对定义的事实模型触发所有规则：

本地执行规则

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

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

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

      // Set up the fact model:
      Person p = new Person();
```

```
        p.setWage(12);
        p.setFirstName("Tom");
        p.setLastName("Summers");
        p.setHourlyRate(10);

        // Insert the person into the session:
        kSession.insert(p);

        // Fire all rules:
        kSession.fireAllRules();
        kSession.dispose();
      }

      catch (Throwable t) {
        t.printStackTrace();
      }
    }
  }
```

要在 KIE 服务器上测试此规则，请配置 .java 类，其导入和规则执行信息与本地示例类似，另外还指定 KIE 服务配置和 KIE 服务客户端详情：

在 KIE 服务器上执行规则

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {
```

```
  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
                                        username,
                                        password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert Person into the session:
      KieCommands kieCommands = KieServices.Factory.get().getCommands();
      List<Command> commandList = new ArrayList<Command>();
      commandList.add(kieCommands.newInsert(p, "personReturnId"));

      // Fire all rules:
      commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
      BatchExecutionCommand batch =
kieCommands.newBatchExecution(commandList, sessionName);

      // Use rule services client to send request:
      RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
      ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
      System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

8. 从项目目录中运行配置的 .java 类。您可以在开发平台（如 Red Hat CodeReady Studio）或命令行中运行该文件。

Maven 执行示例（在项目目录中）：

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

Java 执行示例（在项目目录中）

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

9. 在命令行中检查规则执行状态并在服务器日志中。如果有任何规则没有按预期执行，请检查项目中配置的规则，以及验证提供的数据的主类配置。

第 61 章 后续步骤

- 使用测试场景测试决策服务
- 打包和部署 Red Hat Process Automation Manager 项目

部分 VIII. 使用测试场景测试决策服务

作为业务分析员或业务规则开发人员，您可以在部署项目之前使用 Business Central 中的测试决策服务。您可以测试基于 DMN 和规则的决策服务，以确保它们正常运行，并如预期一样。另外，您可以在项目开发过程中随时测试决策服务。

先决条件

- Business Central 中创建了决策服务的空间和项目。详情请参阅 开始使用决策服务。
- 业务规则及其相关数据对象已针对基于规则的决策服务进行了定义。详情请参阅使用 指导决策表设计决策服务。
- 为基于 DMN 的决策服务定义了 DMN 决策逻辑及其关联的自定义数据类型。详情请参阅使用 DMN 模型设计决策服务。

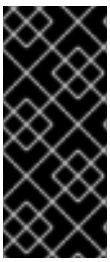
注意

定义的业务规则不是测试场景的技术前提条件，因为情景可以测试构成业务规则的定义数据。但是，首先创建规则会有所帮助，以便您也可以在测试场景中测试整个规则，以便在这种情况下更密切地与预期决策服务匹配。对于基于 DMN 的测试场景，确保为决策服务定义了 DMN 决策逻辑及其关联的自定义数据类型。

第 62 章 测试场景

通过测试 Red Hat Process Automation Manager 中的场景，您可以在将其部署到生产环境前验证业务规则和业务规则数据的功能（适用于基于 DMN 的测试场景）。通过测试场景，您可以使用项目中的数据根据一个或多个定义的业务规则来设置给定条件和预期结果。当您运行该场景时，会比较规则实例的预期结果和实际结果。如果预期的结果与实际结果匹配，则测试会成功。如果预期结果与实际结果不匹配，则测试会失败。

Red Hat Process Automation Manager 包括新的 测试场景 设计人员和之前的 测试场景(Legacy) 设计人员。默认设计器是新的测试场景设计器，它支持测试规则和 DMN 模型，并提供测试场景的增强整体用户体验。如果需要，您可以继续使用旧的测试场景程序，该设计只支持基于规则的测试场景。



重要

旧的测试场景设计程序从 Red Hat Process Automation Manager 版本 7.3.0 中弃用。在以后的 Red Hat Process Automation Manager 发行版本中会删除它。改为使用新的测试场景设计程序。

您可以使用多种方法运行定义的测试场景，例如，您可以在项目级别或特定测试场景资产中运行可用的测试场景。测试场景是独立的，不会影响或修改其他测试场景。您可以在 Business Central 项目开发过程中随时运行测试场景。您不必编译或部署决定服务来运行测试场景。

您可以将数据对象从不同软件包导入到与测试场景相同的项目软件包。默认导入同一软件包中的资产。创建必要的数据对象和测试场景后，您可以使用测试场景设计器的 Data Objects 选项卡来验证所有必需的数据对象是否已通过 添加新项 来导入其他现有数据对象。

重要

在测试场景文档中，测试场景 和测试 场景设计人员 的所有引用都适用于新版本，除非明确声明为旧版本。

第 63 章 数据对象

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

63.1. 创建数据对象

以下流程是创建数据对象的一般概述。它并不适用于特定的业务资产。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Data Object。
3. 输入唯一数据对象名称，然后选择您希望数据对象可用于其他规则资产的软件包。 同一名称的数据对象不能位于同一软件包中。在指定的 DRL 文件中，您可以从任何软件包中导入数据对象。

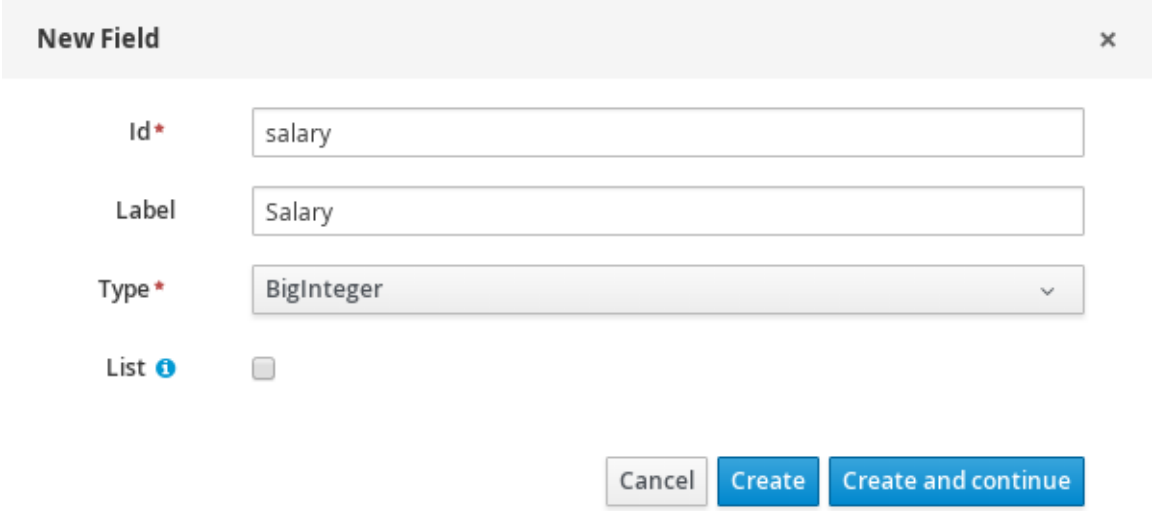
从其他软件包导入数据对象

您可以将另一软件包中的现有数据对象直接导入到资产设计人员，如指导规则或指导决策表设计人员。在项目中选择相关规则资产以及资产设计器，请转至 Data Objects → New item 以选择要导入的对象。

4. 要使数据对象持久化，请选择"永久 "复选框。Persistable 数据对象可以根据 JPA 规格存储在数据库中。默认的 JPA 为 Hibernate。
5. 点 确定。
6. 在数据对象设计器中，点 add 字段在 对象中添加包含属性 Id、标签和 类型的字段。所需的属性标有星号(*)。
   - id： 输入字段的唯一 ID。

- **label:** （可选）输入字段的标签。
- **Type：** 输入字段的数据类型。
- **列表 ：** （可选）选择此复选框，以启用字段保存指定类型的多个项目。

图 63.1. 在数据对象中添加数据字段



7. 点 Create 添加新字段，或者点击 Create 并继续添加新字段并继续添加其他字段。

注意

要编辑字段，请选择字段行，并使用屏幕右侧的 常规属性。

第 64 章 BUSINESS CENTRAL 中的测试场景设计器

测试场景设计人员提供了一个表格布局，可帮助您定义场景模板和所有关联的测试案例。设计器布局由一个表组成，其中包含一个标题和各个行。标头包含三个部分、GIVEN 和 EXPECT 行、一个实例一行，以及含有对应字段的行。标头也称为测试场景模板，单独的行称为测试场景定义。

测试场景模板或标头分为两个部分：

- GIVEN 数据对象及其字段 - 代表输入信息
- EXPECT 数据对象及其字段 - 代表对象及其字段，它们根据给定信息检查准确的值，这也构成了预期的结果。

测试场景定义代表模板的独立测试情况。

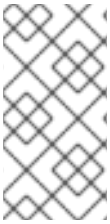
您可以从设计人员左侧面板中访问 Project Explorer，而从右面板中，您可以访问 Settings、测试工具、Scenario Cheatsheet、Test Report 和 coverage Report 选项卡。您可以访问 Settings 选项卡，以查看并编辑基于规则和基于 DMN 的测试场景的全局设置。您可以使用 Test Tools 来配置数据对象映射。场景 C heatsheet 选项卡包含备注和备忘单，您可将其用作参考。Test Report 选项卡中显示测试和场景状态概述。要查看测试覆盖统计，您可以使用测试场景设计器右侧的 覆盖范围报告 选项卡。

64.1. 导入数据对象

测试场景设计器会加载所有位于与测试场景相同的软件包中的数据对象。您可以从设计器中的 Data Objects 选项卡中查看所有数据对象。载入的数据对象也会显示在 Test Tools 面板中。

您需要关闭并重新打开设计者，以防数据对象更改（例如，在创建新数据对象或删除现有数据时）。从列表中选择一个数据对象来显示其字段和字段类型。

如果您想要使用位于与测试场景不同的软件包中的数据对象，您需要首先导入数据对象。按照以下步骤，为基于规则的测试场景导入数据对象。



注意

在创建基于 DMN 的测试场景时，您无法导入任何数据对象。基于 DMN 的测试场景不使用项目中任何数据对象，而是使用 DMN 文件中定义的自定义数据类型。

流程

1. 进入测试场景设计器中的 Project Explorer 面板。
2. 在 Test Scenario 中选择一个测试场景。
3. 选择 Data Objects 选项卡，再单击 New Item。
4. 在 Add import 窗口中，从下拉列表中选择 data 对象。
5. 单击确定，然后单击保存。
6. 关闭并重新打开测试场景设计器，以从数据对象列表中查看新数据对象。

64.2. 导入测试场景

您可以使用项目视图的 Asset 选项卡中的 Import Asset 按钮导入现有的测试场景。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 从项目的 Asset 选项卡中，单击 Import Asset。
3. 在 Create new Import Asset 窗口中，
   - 输入导入资产的名称。

- 从 Package 下拉列表中选择软件包。
- 在 Please select a file to upload 中，单击 Choose File... to browse to test scenario file。

4. 选择 文件，然后单击 Open。
5. 点 Ok，测试场景在测试场景器中打开。

64.3. 保存测试场景

在创建测试场景模板或定义测试场景时，您可以随时保存测试场景。

流程

1. 在测试场景设计器工具栏中，单击 Save。
2. 在 Confirm Save 窗口上，
   a. 如果您想添加关于测试场景的评论，请单击 一个注释。
   b. 再次点击 Save。

显示在屏幕上成功保存了测试场景的消息。

64.4. 复制测试场景

您可以使用右上角的 Copy 按钮将现有测试场景复制到同一软件包或某个其他软件包。

流程

1. 在测试场景设计器工具栏中，单击 Copy。

2. 在 Make a Copy 窗口中，

   a. 在 New Name 字段中输入名称。

   b. 选择您要将测试场景复制到的软件包。

   c. 可选：要添加注释，点 add a comment。

   d. 点 Make a Copy。

在屏幕上显示复制测试方案的消息。

64.5. 下载测试场景

您可以将测试场景的副本下载到您的本地机器中，以备将来参考或作为备份。

流程

在右下角的测试场景设计器工具栏中点击 Download 图标。

.scesim 文件已下载到您的本地机器。

64.6. 在测试场景的版本间切换

Business Central 可让您在不同版本的测试场景间进行切换。每次保存该方案时，该方案的新版本都会列在 Latest Versions 下。要使用这个功能，您必须至少保存测试场景文件。

流程

1. 从右上角的测试场景设计器工具栏，单击 Latest Version。文件的所有版本都列在 Latest Version 下（如果存在）。

2. 点您要处理的版本。

   所选测试场景的版本在测试场景设计器中打开。

3. 在设计器工具栏中点 Restore。

4. 在 Confirm Restore 中，

   a. 要添加注释，请单击" 添加注释 "。

   b. 点击 Restore 确认。

在屏幕上显示所选版本已被成功重新加载的消息。

64.7. 查看或隐藏警报面板

Alerts 面板显示在测试场景设计人员或项目视图的底部。它包含执行测试失败时构建信息和错误消息。

流程

在右上角的设计器工具栏中，单击 Hide Alerts/View Alerts 以启用或禁用报告面板。

64.8. 上下文菜单选项

测试场景设计器提供上下文菜单选项，可让您对表执行基本操作，如添加、删除和复制行和列。要使用上下文菜单，您需要右键单击一个表元素。菜单选项根据您选择的表格元素的不同而有所不同。

表 64.1. 上下文菜单选项

表元素	cell label	可用上下文菜单选项
标头	#, scenario description	插入行

表元素	cell label	可用上下文菜单选项
	给定，预期	插入最左边列、Insert rightmost 列、Insert 行（下面）
	实例 1, 实例 2, 属性 1, 属性 2	插入列左、Insert 列右、删除列、Duplicate Instance、Insert 行。
行	所有行号的单元格，测试场景描述或测试场景定义	插入行（在下面插入行）、Duplicate row、Delete row、Run scenario

表 64.2. 表交互描述

表互动	描述
插入最左边的列	根据用户选择，插入一个新的左边列（在表的 GIVEN 或 EXPECT 部分中）。
插入右列	根据用户选择，插入一个新的右边列（在表的 GIVEN 或 EXPECT 部分中）。
插入列左侧的	在选定列的左侧插入新列。新列与所选列（基于用户选择的 GIVEN 或 EXPECT 部分）相同。
插入列右	在所选列的右侧插入新列。新列与所选列（基于用户选择的 GIVEN 或 EXPECT 部分）相同。
删除列	删除所选列。
插入行	在所选行上方插入新行。
插入行	在所选行的下面插入新行。如果从标头单元格调用，请使用 index 1 插入新行。
重复的行	复制所选行。
重复的实例	复制所选实例。
删除行	删除所选行。
运行场景	运行单个测试场景。

Insert 列或 Insert 列左 上下文菜单选项的行为不同。

- ***如果所选列没有定义类型，则会添加一个没有类型的新列。***

- 如果选中的列定义了类型，则创建了新的空列或带有父实例类型的列。
  - 如果从实例标头执行操作，则创建一个新列，且没有类型。
  - 如果从属性标头执行操作，则创建一个含有父实例类型的新列。

64.9. 测试场景的全局设置

您可以使用测试场景设计器右侧的全局 Settings 标签页来设置和修改资产的额外属性。

64.9.1. 为基于规则的测试场景配置全局设置

按照以下步骤查看并编辑基于规则的测试场景的全局设置。

流程

1. 点测试场景设计器右侧的 Settings 标签页来显示属性。
2. 在 Settings 面板中配置以下属性：
   - 名称 ：您可以使用设计器右上角的 Rename 选项更改现有测试场景的名称。
   - 类型 ：此属性指定它是基于规则的测试场景，它是只读的。
   - 无状态会话 ：选择或清除此复选框，以指定 KieSession 是否无状态。

     注意

     如果当前的 KieSession 无状态，且未选中复选框，则测试将失败。
   - KieSession:（可选）输入测试场景的 KieSession。

- *RuleFlowGroup/AgendaGroup:（可选）输入 RuleFlowGroup 或 AgendaGroup 作为测试场景。*

3. *可选：要在测试后从项目级别跳过整个模拟，请选中 复选框。*

4. *点击 Save。*

64.9.2. 为基于 DMN 的测试场景配置全局设置

按照以下步骤查看并编辑基于 DMN 的测试场景的全局设置。

流程

1. *点测试场景设计器右侧的 Settings 标签页来显示属性。*

2. *在 Settings 面板中配置以下属性：*

   - *名称 ：您可以使用设计器右上角的 Rename 选项更改现有测试场景的名称。*
   - *类型 ：此属性指定它是基于 DMN 的测试场景，它是只读的。*
   - *DMN 模型 ：（可选）为测试场景输入 DMN 模型。*
   - *DMN 名称 ：这是 DMN 模型的名称，它是只读的。*
   - *DMN 命名空间 ：这是 DMN 模型的默认命名空间，它是只读的。*

3. *可选：要在测试后从项目级别跳过整个模拟，请选中 复选框。*

4. *点击 Save。*

第 65 章 测试场景模板

在指定测试场景定义前，您需要创建一个测试场景模板。测试场景表的标头定义每个场景的模板。您需要为 GIVEN 和 EXPECT 部分设置实例和属性标头的类型。实例标头映射到特定数据对象（事实），而属性标头映射到对应数据对象的特定字段。

使用测试场景设计器，您可以为基于规则和基于 DMN 的测试场景创建测试场景模板。

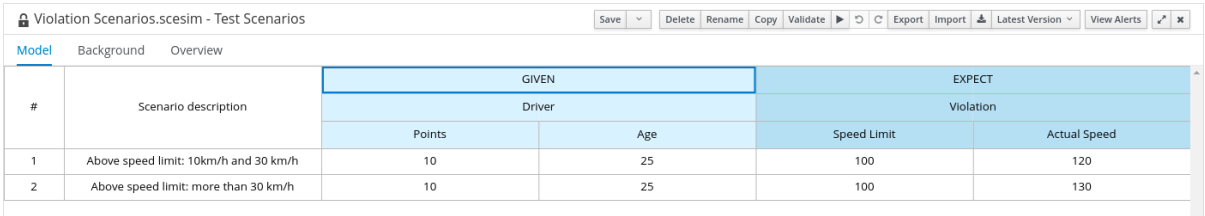
65.1. 为基于规则的测试场景创建测试场景模板

按照以下步骤为您的基于规则的测试场景创建测试场景模板，以验证您的规则和数据。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects，再点击您要为其创建测试场景的项目。
2. 点 Add Asset → Test Scenario。
3. 输入 测试 Scenario 名称并选择相应的 软件包。您选择的软件包必须包含所有所需的数据对象和规则资产已被分配或将被分配。
4. 选择 RULE 作为 Source 类型。
5. 点 Ok 在测试场景设计器中创建并打开测试场景。
6. 将 GIVEN 列标头映射到数据对象：

图 65.1. 测试场景 GIVEN 标头单元



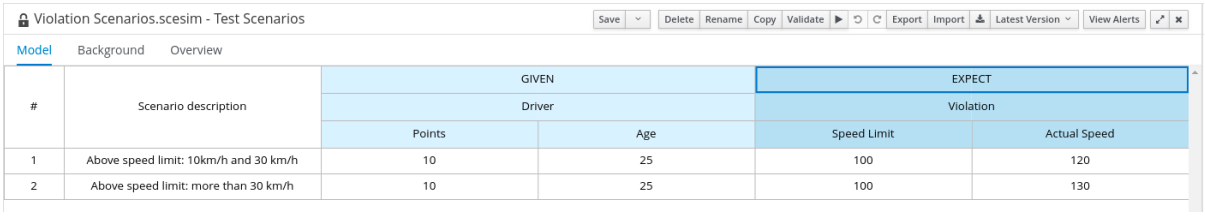
a. ***在 GIVEN 部分中**选择**一个**实**例**标头单**元。***

b. ***从 Test Tools** 选项**卡中**选择**数据**对**象。***

c. ***点 Insert Data Object。***

7. ***将 EXPECT 列**标题**映射到数据**对**象：***

*图 **65.2.** 测试场**景 EXPECT** 标头单**元***



#	Scenario description	GIVEN		EXPECT	
		Driver		Violation	
		Points	Age	Speed Limit	Actual Speed
1	Above speed limit: 10km/h and 30 km/h	10	25	100	120
2	Above speed limit: more than 30 km/h	10	25	100	130

a. ***在 EXPECT 部分**选择**一个**实**例**标头单**元。***

b. ***从 Test Tools** 选项**卡中**选择**数据**对**象。***

c. ***点 Insert Data Object。***

8. ***将数据**对**象字段映射到属性**单**元：***

a. *选择**一个**实**例**标头单**元或属性**标头单**元。***

b. ***从 Test Tools** 选项**卡中**选择 **data** 对**象字段。***

c. ***点 Insert Data Object。***

9. ***要插入数据**对**象的更多属性，**请**右**键单击**属性**标题，**并根据需要**选择 **Insert 列右 或 Insert 列。***

10. 在测试场景执行过程中为属性单元定义 java 方法：

    a. 选择一个实例标头单元或属性标头单元。

    b. 从 Test Tools 选项卡中选择 data 对象字段。

    c. 点 Insert Data Object。

    d. 使用带有前缀 # 的 MVEL 表达式来定义测试场景执行的 java 方法。

    e. 要插入数据对象的更多属性，请右键单击属性标题单元格，根据需要选择 Insert 列 或 Insert 列。

11. 根据需要，使用上下文菜单添加或删除列和行。

有关基于规则的场景中表达式语法的详情，请参考 第 70.1 节 “基于规则的测试场景中的表达式语法”。

65.2. 在基于规则的测试场景中使用别名

在测试场景设计器中，当您映射带有数据对象的标头单元时，数据对象会从 Test Tools 选项卡中删除。您可以使用别名将数据对象重新映射到另一个标头单元。通过别名，您可以在测试场景中指定相同数据对象的多个实例。您还可以创建属性别名来直接在表中重命名使用的属性。

流程

1. 在 Business Central 中的测试场景设计器中，双击标题单元并手动更改名称。确保唯一命名别名。

现在，实例会出现在 Test Tools 选项卡中的数据对象列表中。

第 66 章 测试基于 DMN 的测试场景的模板

Business Central 会自动为每个基于 DMN 的测试场景资产生成模板，其中包含相关 DMN 模型的所有指定输入和决策。对于 DMN 模型中的每个输入节点，会添加一个 GIVEN 列，而每个决策节点都由 EXPECT 列表示。您可以根据需要修改默认模板。另外，要仅测试整个 DMN 模型的特定部分，可以删除生成的列，以及将决策节点从 EXPECT 移到 GIVEN 部分。

66.1. 为基于 DMN 的测试场景创建测试场景模板

按照以下步骤验证您的 DMN 模型的基于 DMN 的测试场景模板。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects，再点击您要为其创建测试场景的项目。
2. 点 Add Asset → Test Scenario。
3. 输入 测试 Scenario 名称并选择相应的 软件包。
4. 选择 DMN 作为 Source 类型。
5. 使用 Choose DMN asset 选项选择现有的 DMN 资产。
6. 点 Ok 在测试场景设计器中创建并打开测试场景。

   模板会自动生成，您可以根据自己的需要修改它。
7. 在测试场景执行过程中为属性单元定义 java 方法：
   a. 单击实例标题单元或属性标题单元。

b. 从 Test Tools 选项卡中选择 data 对象字段。

c. 点 Insert Data Object。

d. 使用表达式来定义用于测试场景的 java 方法。

e. 要为数据对象添加更多属性，请右键单击属性标题单元格，并根据需要选择 Insert 列或 Insert 列。

8. 根据需要，使用上下文菜单添加或删除列和行。

有关基于 DMN 的场景中表达式语法的详情，请参考 第 70.2 节 “基于 DMN 的测试场景中的表达式语法”。

第 67 章 定义测试场景

创建测试场景模板后，您必须在以后定义测试场景。测试场景表的行定义了单个测试场景。测试场景具有唯一的索引号、描述、输入值（提供值）和一组输出值（预期值）。

先决条件

- 为所选的测试场景创建了测试场景模板。

流程

1. 在测试场景设计器中打开测试场景。
2. 输入测试场景的描述，并在行的每个单元处填写所需的值。
3. 根据需要，使用上下文菜单添加或删除行。

   双击单元以启动内联编辑。要从测试评估中跳过特定的单元格，请将其留空。

定义测试场景后，您可以下次运行测试。

第 68 章 测试场景中的后台实例

在测试场景设计器中，您可以使用 背景 标签页为基于规则和基于 DMN 的测试场景添加和设置后台数据。您可以根据可用的数据对象添加和定义常见用于整个测试场景模拟的 GIVEN 数据。背景 标签能够在每个测试场景中添加和共享数据。使用 Background 选项卡 添加 的数据无法被 Model 标签页数据覆盖。

例如，如果测试场景示例在所有测试场景中需要同一 Age 的值，您可以在 Background 页面定义 Age 值，并从测试场景表模板中排除该列。在这种情况下，所有测试场景的 Age 设置为 25。

图 68.1. 带有 Age 的重复值的测试场景示例

Model | Background | Overview | Data Objects

#	Scenario description	GIVEN			EXPECT	
		Driver			Violation	
		Age	Points	Name	Speed Limit	Actual Speed
1	Insert value	25	Insert value	Insert value	Insert value	Insert value
2	Insert value	25	Insert value	Insert value	Insert value	Insert value

图 68.2. Age 的重复值后台定义示例

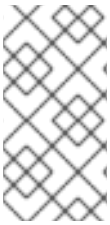
Model | Background | Overview | Data Objects

GIVEN
Driver
Age
25

图 68.3. 带有排除的 Age 列的修改后的测试场景模板

Model | Background | Overview | Data Objects

#	Scenario description	GIVEN		EXPECT	
		Driver		Violation	
		Points	Name	Speed Limit	Actual Speed
1	Insert value	Insert value	Insert value	Insert value	Insert value
2	Insert value	Insert value	Insert value	Insert value	Insert value



注意

在后台选项卡中定义的 GIVEN 数据只能在相同 *.scesim 文件的测试场景之间共享，且不会在不同的测试场景中共享。

68.1. 在基于规则的测试场景中添加后台数据

按照以下步骤在基于规则的测试场景中添加和设置后台数据。

先决条件

- 为所选的测试场景创建基于规则的测试场景模板。有关创建基于规则的测试场景的详情请参考 第 65.1 节 “为基于规则的测试场景创建测试场景模板”。

- 定义了单个测试场景。有关定义测试场景的详情请参考 第 67 章 定义测试场景。

流程

1. 在测试场景设计器中打开基于规则的测试场景。

2. 点测试场景设计器的背景信息选项卡。

3. 在 GIVEN 部分中选择一个实例标头单元，以添加后台数据对象字段。

4. 在 Test Tools 面板中选择 data 对象。

5. 点 Insert Data Object。

6. 选择属性标头单元来添加后台数据对象字段。

7. 在 Test Tools 面板中选择 data 对象。

8. 点 Insert Data Object。

9. 要为数据对象添加更多属性，请右键单击属性标题单元格，并根据需要选择 Insert 列 或 Insert 列。

10. 根据需要，使用上下文菜单添加或删除列和行。

11. 运行定义的测试场景。

68.2. 在基于 DMN 的测试场景中添加后台数据

按照以下步骤在基于 DMN 的测试场景中添加和设置后台数据。

先决条件

- 为所选测试场景创建基于 DMN 的测试场景模板。有关创建基于 DMN 的测试场景的详情请参考 第 66.1 节 “为基于 DMN 的测试场景创建测试场景模板”。
- 定义了单个测试场景。有关定义测试场景的详情请参考 第 67 章 定义测试场景。

流程

1. 在测试场景设计器中打开基于 DMN 的测试场景。
2. 点测试场景设计器的背景信息选项卡。
3. 在 GIVEN 部分中选择一个实例标头单元，以添加后台数据对象字段。
4. 在 Test Tools 面板中选择 data 对象。
5. 点 Insert Data Object。
6. 选择属性标头单元来添加后台数据对象字段。
7. 在 Test Tools 面板中选择 data 对象。
8. 点 Insert Data Object。
9. 要为数据对象添加更多属性，请右键单击属性标题单元格，并根据需要选择 Insert 列 或 Insert 列。

10. 根据需要，使用上下文菜单添加或删除列和行。

11. 运行定义的测试场景。

第 69 章 在测试场景中使用列表和映射集合

测试场景设计器支持基于 DMN 以及基于规则的测试场景的列表和映射集合。您可以创建并定义集合，如列表或映射 作为 GIVEN 和 EXPECT 列中特定单元的值。

对于基于规则的测试场景，设计程序支持以下集合：

- java.util.Collection
- java.util.List
- java.util.ArrayList
- java.util.LinkedList
- java.util.Map
- java.util.HashMap
- java.util.LinkedHashMap
- java.util.TreeMap

注意

测试场景设计器不支持 java.util.Set 集合。对于映射条目，您必须将条目键设置为 String 数据类型。

要传递基于规则的集合编辑器的 EXPECT 列中的参数，请使用 actualValue 关键字，然后在基于 DMN 的测试场景中使用 ? 关键字。

流程

1. ***首先设置列类型（使用类型为列表或映射的字段）。***

2. ***双击列中的某一单元格以输入一个值。***

3. ***在集合编辑器中为数据对象创建列表值：***

   a. *选择* ***Create List。***

   b. *单击* ***Add new item。***

   c. *输入所需值并点击检查图标*
   
   来保存您添加的每个集合项目。

   d. ***点击 Save。***

   e. ***要编辑集合中的项目，请点击集合弹出窗口中的铅笔图标。***

   f. ***点 Save Changes。***

   g. ***要从集合中删除项目，请点击集合弹出窗口中的 bin 图标。***

4. ***在集合编辑器弹出窗口中定义数据对象的列表值：***

   a. *选择* ***Define List。***

   b. ***使用 MVEL 或 FEEL 表达式在文本字段中定义列表值。***

基于规则的测试场景使用 MVEL 表达式语言，基于 DMN 的测试场景使用 FEEL 表达式语言。

c. 点击 Save。

5. 在集合编辑器弹出窗口中为数据对象创建映射值：

a. 选择 Create Map。

b. 单击 Add new item。

c. 输入所需值并点击检查图标

来保存您添加的每个集合项目。

d. 点击 Save。

e. 要编辑集合中的项目，请点击集合弹出窗口中的铅笔图标。

f. 点 Save Changes。

g. 要从集合中删除项目，请点击集合弹出窗口中的 bin 图标。

6. 在集合编辑器中为数据对象定义映射值：

a. 选择 Define Map。

b. 使用 MVEL 或 FEEL 表达式在文本字段中定义映射值。

基于规则的测试场景使用 MVEL 表达式语言，基于 DMN 的测试场景使用 FEEL 表达式语言。

c.

点击 Save。

注意

要定义基于 DMN 的测试场景的映射值，您可以添加事实并使用 FEEL 表达式，而不使用集合编辑器。

7.

点 Remove 删除整个集合。

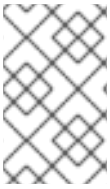
第 70 章 测试场景中的表达式语法

测试场景设计器支持基于规则和基于 DMN 的测试场景的不同表达式语言。基于规则的测试场景支持 MVFLEX 表达式语言(MVEL)和基于 DMN 的测试场景，支持 Friendly Enough Expression Language(FEEL)。

70.1. 基于规则的测试场景中的表达式语法

基于规则的测试场景支持以下内置数据类型：

- 字符串
- 布尔值
- 整数
- Long
- 双
- 浮点值
- 字符
- 字节
- 短
- LocalDate



注意

对于任何其他数据类型，请使用带有前缀 # 的 MVEL 表达式。

按照测试场景设计器中的 BigDecimal 示例以使用 # 前缀来设置 java 表达式：

- 输入 # java.math.BigDecimal.valueOf(10) 作为 GIVEN 列值。
- Enter # actualValue.intValue() == 10 for the EXPECT column value.

您可以引用 java 表达式中的 EXPECT 列的实际值来执行条件。

测试场景设计器支持以下基于规则的测试场景定义表达式：

表 70.1. 表达式语法的描述

Operator	描述
=	指定与值相等。这适用于所有列，是 GIVEN 列支持的唯一 Operator。
=, =!, <>	指定值的质量。此运算符可以与其他运算符合并。
<, >, <=, >=	指定比较：小于、大于、小于或等于，且大于.
#	此运算符用于将 java 表达式值设置为属性标题单元格，可以作为 java 方法执行。
[value1, value2, value3]	指定值列表。如果一个或多个值有效，则场景定义将评估为 true。
expression1; expression2; expression3	指定表达式列表。如果所有表达式都有效，则场景定义将评估为 true。

注意

在评估基于规则的测试场景时，会从评估中跳过一个空单元格。要定义空字符串，请使用 =、[] 或 ; 以及定义 null 值，请使用 null。

表 70.2. 表达式示例

表达式	描述
-1	实际值等于 -1。
< 0	实际值小于 0。
! > 0	实际值不大于 0。
[-1, 0, 1]	实际值等于 -1 或 0 或 1。
<> [1, -1]	实际值不等于 1 或 -1。
!100; 0	实际值不等于 100，但等于 0。
!= < 0; <> > 1	实际值不超过 0 个或大于 1。
<> <= 0; >= 1	实际值不小于或等于 0，但大于或等于 1。

您可以在基于规则的测试场景设计器右侧的 Scenario Cheatsheet 选项卡中引用支持的命令和语法。

70.2. 基于 DMN 的测试场景中的表达式语法

测试场景器中基于 DMN 的测试场景支持以下数据类型：

表 70.3. 基于 DMN 的场景支持的数据类型

支持的数据类型	描述
数字和字符串	字符串必须用引号分隔，例如：" **John Doe**"、**"Brno"** 或 **""**。
布尔值	**true**、**假** 和 **null**。
日期和时间	例如，**date("2019-05-13")** 或 **时间("14:10:00+02:00")**.
功能	支持内置数功能，如 **avg**、**max**。
contexts	例如，**{x : 5, y : 3}**。
范围和列表	例如： **[1 .10]** 或 **[2、 3、 4、 5]**。

注意

在评估基于 DMN 的测试场景时，会从评估中跳过一个空单元格。要在基于 DMN 的测试场景中定义空字符串，请使用 " 和 定义 null 值，请使用 null。

您可以参考基于 DMN 测试场景设计器右侧的 Scenario Cheatsheet 选项卡中的支持的命令和语法。

第 71 章 运行测试场景

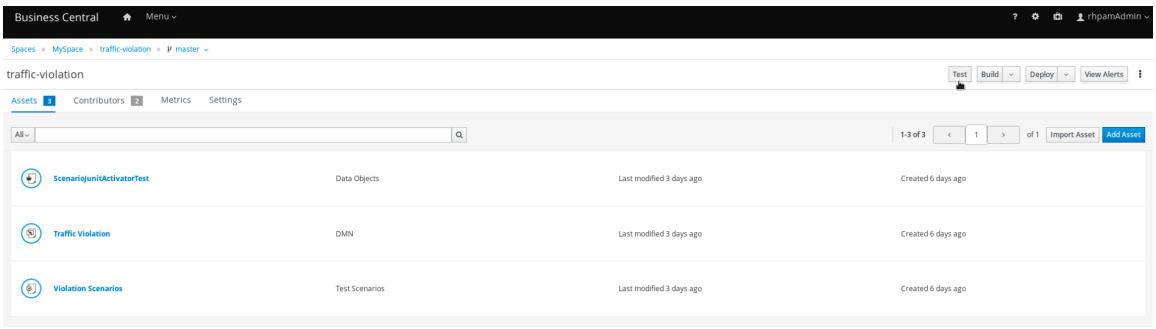
创建测试场景模板并定义测试场景后，您可以运行测试来验证您的业务规则和数据。

流程

1. 要运行定义的测试场景，请执行以下任一任务：

   - 要在项目页面右上角的多个资产中执行所有可用的测试场景，请单击 **Test**。

   图 71.1. 从项目视图中运行所有测试场景

   - 要执行 .scesim 文件中定义的所有可用测试场景，在 Test Scenario Designer 的顶部，点 Run Test  图标。

   - 要运行在单个 .scesim 文件中定义的单个测试场景，请右键单击您要运行的测试场景，然后选择 **Run scenario**。

2. **Test Report** 面板显示测试和场景状态概述。

   测试执行后，如果测试场景中输入的值与预期值不匹配，则将突出显示对应的单元。

3. 如果测试失败，您可以执行以下任务来排除故障：

   - 要在弹出窗口中查看错误消息，请将鼠标光标悬停在突出显示的单元上。

- 要打开设计人员底部的 Alerts 面板或用于错误消息的项目视图，请单击 View Alerts。

- 进行必要的更改，然后再次运行测试，直到场景通过为止。

第 72 章 本地运行测试场景

在 Red Hat Process Automation Manager 中，您可以直接在 Business Central 中或本地使用命令行运行测试场景。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 在 Project 的主页上，选择 Settings 选项卡。
3. 选择 git URL 并点 Clipboard  复制 git url。
4. 打开一个命令终端，再导航到要克隆 git 项目的目录。
5. 运行以下命令：

```
git clone your_git_project_url
```

将 your_git_project_url 替换为相关的数据，如 git://localhost:9418/MySpace/ProjectTestScenarios。

6. 成功克隆项目后，导航到 git 项目目录并执行以下命令：

```
mvn clean test
```

您的项目的构建信息和测试结果（如测试运行的数量以及测试运行是成功）是否显示在命令终端中。如果出现故障，在 Business Central 中进行必要的更改，拉取更改并再次运行命令。

第 73 章 导出和导入测试方案电子表格

这些部分介绍了如何在测试场景设计器中导出和导入测试方案电子表格。您可以使用 Microsoft Excel 或 LibreOffice Calc 等软件分析和管理测试场景电子表格。测试场景设计器支持 .CSV 文件格式。有关 Comma-Separated Values(CSV)格式的 RFC 规格的更多信息，请参阅 Comma -Separated Values(CSV)文件的通用格式和 MIME 类型。

73.1. 导出测试方案电子表格

按照以下步骤，使用 Test Scenario 设计器导出测试场景电子表格。

流程

1. 在右上角的 Test Scenario Designer 工具栏中，点 Export 按钮。

2. 在本地文件目录中选择一个目的地，再确认保存 .CSV 文件。

.CSV 文件将导出到您的本地机器。

73.2. 导入测试场景电子表格

按照以下步骤，使用 Test Scenario 设计器导入测试场景电子表格。

流程

1. 在右上角的 Test Scenario Designer 工具栏中点 Import 按钮。

2. 在 Select file to Import prompt 中，点 Choose File... 并选择您想从本地文件目录中导入的 .CSV 文件。

3. 点 Import。

.CSV 文件导入到 Test Scenario designer。

警告

您不能修改所选 .CSV 文件中的标头。否则，电子表格可能无法成功导入。

第 74 章 测试场景的覆盖报告

测试场景设计人员提供了一个明确的、一致的方式，用于在测试场景设计人员右侧的中使用 覆盖范围统计来显示测试覆盖范围统计。您还可以下载覆盖范围报告来查看和分析测试覆盖统计。下载的测试场景覆盖报告支持 .CSV 文件格式。有关 Comma-Separated Values(CSV)格式的 RFC 规格的更多信息，请参阅 Comma -Separated Values(CSV)文件的通用格式和 MIME 类型。

您可以查看基于规则和基于 DMN 的测试场景的覆盖报告。

74.1. 为基于规则的测试场景生成覆盖报告

在基于规则的测试方案中，覆盖范围报告 标签页包含有关以下内容的详细信息：

- 可用规则数
- 触发的规则数
- 触发规则的百分比
- 执行规则的百分比以 pie chart 表示
- 每个规则执行的次数
- 为每个定义的测试场景执行的规则

按照流程为基于规则的测试场景生成覆盖报告：

先决条件

- 为所选的测试场景创建基于规则的测试场景模板。有关创建基于规则的测试场景的详情请参考 第 65.1 节 “为基于规则的测试场景创建测试场景模板”。

- 定义了单个测试场景。有关定义测试场景的详情请参考 第 67 章 定义测试场景。

注意

要为基于规则的测试场景生成覆盖范围报告，您必须至少创建一个规则。

流程

1. 在测试场景设计器中打开基于规则的测试场景。
2. 运行定义的测试场景。
3. 点击测试场景设计器右侧的 覆盖范围报告 来显示测试覆盖统计。
4. 可选：要下载测试场景覆盖报告，请点击 **Download report**。

74.2. 为基于 DMN 的测试场景生成覆盖报告

在基于 DMN 的测试场景中，覆盖范围报告 标签包含有关以下内容的详细信息：

- 可用决策数
- 已执行决策的数量
- 已执行决策的百分比
- 作为 **pie chart** 的执行决策的百分比
- 每次决策执行的次数

- 为每个定义的测试场景执行决策

按照以下步骤为基于 DMN 的测试场景生成覆盖报告：

先决条件

- 为所选测试场景创建基于 DMN 的测试场景模板。有关创建基于 DMN 的测试场景的详情请参考 第 66.1 节 “为基于 DMN 的测试场景创建测试场景模板”。

- 定义了单个测试场景。有关定义测试场景的详情请参考 第 67 章 定义测试场景。

流程

1. 在测试场景设计器中打开基于 DMN 的测试场景。

2. 运行定义的测试场景。

3. 点击测试场景设计器右侧的 覆盖范围报告 来显示测试覆盖统计。

4. 可选：要下载测试场景覆盖报告，请点击 Download report。

第 75 章 使用 KIE 服务器 REST API 执行测试场景

与 KIE 服务器的 REST 端点直接交互，提供调用代码和决策逻辑定义之间的最分离。您可以使用 KIE 服务器 REST API 在外部执行测试场景。它针对已部署的项目执行测试场景。

注意

默认情况下禁用此功能，使用 org.kie.scenariosimulation.server.ext.disabled 系统属性启用它。

有关 KIE Server REST API 的更多信息，请参阅使用 KIE API 与 Red Hat Process Automation Manager 交互。

先决条件

- KIE 服务器是安装和配置的，包括具有 kie-server 角色的用户的已知用户名和凭证。有关安装选项，请参阅 规划 Red Hat Process Automation Manager 安装。
- 您已将项目构建为 KJAR 工件并将其部署到 KIE 服务器。
- 您有 KIE 容器的 ID。

流程

1. 确定用于访问 KIE 服务器 REST API 端点的基础 URL。这需要了解以下值（使用默认本地部署值作为示例）：
   - 主机（本地主机）
   - 端口(8080)
   - 根上下文(kie-server)

- 基本 REST 路径(services/rest/)

流量违反项目的本地部署中的基本 URL 示例：

http://localhost:8080/kie-server/services/rest/server/containers/traffic_1.0.0-SNAPSHOT

2. 确定用户身份验证要求。

当在 KIE 服务器配置中直接定义用户时，使用 HTTP 基本身份验证并需要用户名和密码。成功请求需要该用户具有 kie-server 角色。

以下示例演示了如何在 curl 请求中添加凭证：

```
curl -u username:password <request>
```

如果使用 Red Hat Single Sign-On 配置 KIE 服务器，则请求必须包含 bearer 令牌：

```
curl -H "Authorization: bearer $TOKEN" <request>
```

3. 指定请求和响应的格式。REST API 端点使用 XML 格式，并使用请求标头来设置：

XML

```
curl -H "accept: application/xml" -H "content-type: application/xml"
```

4. 执行测试场景：

[POST] server/containers/{containerId}/scesim

***curl** 请求示例：*

```
curl -X POST "http://localhost:8080/kie-server/services/rest/server/containers/traffic_1.0.0-SNAPSHOT/scesim"\ -u 'wbadmin:wbadmin;' \ -H "accept: application/xml" -H "content-type: application/xml"\ -d @Violation.scesim
```

***XML** 请求示例：*

```
<ScenarioSimulationModel version="1.8">
 <simulation>
  <scesimModelDescriptor>
   <factMappings>
    <FactMapping>
     <expressionElements/>
     <expressionIdentifier>
      <name>Index</name>
      <type>OTHER</type>
     </expressionIdentifier>
     <factIdentifier>
      <name>#</name>
      <className>java.lang.Integer</className>
     </factIdentifier>
     <className>java.lang.Integer</className>
     <factAlias>#</factAlias>
     <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
     <columnWidth>70.0</columnWidth>
    </FactMapping>
    <FactMapping>
     <expressionElements/>
     <expressionIdentifier>
      <name>Description</name>
      <type>OTHER</type>
     </expressionIdentifier>
     <factIdentifier>
      <name>Scenario description</name>
      <className>java.lang.String</className>
     </factIdentifier>
     <className>java.lang.String</className>
     <factAlias>Scenario description</factAlias>
     <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
     <columnWidth>300.0</columnWidth>
    </FactMapping>
    <FactMapping>
     <expressionElements>
      <ExpressionElement>
       <step>Driver</step>
      </ExpressionElement>
      <ExpressionElement>
       <step>Points</step>
      </ExpressionElement>
     </expressionElements>
     <expressionIdentifier>
```

```
    <name>0|1</name>
    <type>GIVEN</type>
   </expressionIdentifier>
   <factIdentifier>
    <name>Driver</name>
    <className>Driver</className>
   </factIdentifier>
   <className>number</className>
   <factAlias>Driver</factAlias>
   <expressionAlias>Points</expressionAlias>
   <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
   <columnWidth>114.0</columnWidth>
  </FactMapping>
  <FactMapping>
   <expressionElements>
    <ExpressionElement>
     <step>Violation</step>
    </ExpressionElement>
    <ExpressionElement>
     <step>Type</step>
    </ExpressionElement>
   </expressionElements>
   <expressionIdentifier>
    <name>0|6</name>
    <type>GIVEN</type>
   </expressionIdentifier>
   <factIdentifier>
    <name>Violation</name>
    <className>Violation</className>
   </factIdentifier>
   <className>Type</className>
   <factAlias>Violation</factAlias>
   <expressionAlias>Type</expressionAlias>
   <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
   <columnWidth>114.0</columnWidth>
  </FactMapping>
  <FactMapping>
   <expressionElements>
    <ExpressionElement>
     <step>Violation</step>
    </ExpressionElement>
    <ExpressionElement>
     <step>Speed Limit</step>
    </ExpressionElement>
   </expressionElements>
   <expressionIdentifier>
    <name>0|7</name>
    <type>GIVEN</type>
   </expressionIdentifier>
   <factIdentifier>
    <name>Violation</name>
    <className>Violation</className>
   </factIdentifier>
   <className>number</className>
   <factAlias>Violation</factAlias>
   <expressionAlias>Speed Limit</expressionAlias>
```

```
  <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
  <columnWidth>114.0</columnWidth>
 </FactMapping>
 <FactMapping>
  <expressionElements>
   <ExpressionElement>
    <step>Violation</step>
   </ExpressionElement>
   <ExpressionElement>
    <step>Actual Speed</step>
   </ExpressionElement>
  </expressionElements>
  <expressionIdentifier>
   <name>0|8</name>
   <type>GIVEN</type>
  </expressionIdentifier>
  <factIdentifier>
   <name>Violation</name>
   <className>Violation</className>
  </factIdentifier>
  <className>number</className>
  <factAlias>Violation</factAlias>
  <expressionAlias>Actual Speed</expressionAlias>
  <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
  <columnWidth>114.0</columnWidth>
 </FactMapping>
 <FactMapping>
  <expressionElements>
   <ExpressionElement>
    <step>Fine</step>
   </ExpressionElement>
   <ExpressionElement>
    <step>Points</step>
   </ExpressionElement>
  </expressionElements>
  <expressionIdentifier>
   <name>0|11</name>
   <type>EXPECT</type>
  </expressionIdentifier>
  <factIdentifier>
   <name>Fine</name>
   <className>Fine</className>
  </factIdentifier>
  <className>number</className>
  <factAlias>Fine</factAlias>
  <expressionAlias>Points</expressionAlias>
  <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
  <columnWidth>114.0</columnWidth>
 </FactMapping>
 <FactMapping>
  <expressionElements>
   <ExpressionElement>
    <step>Fine</step>
   </ExpressionElement>
   <ExpressionElement>
    <step>Amount</step>
```

```
        </ExpressionElement>
      </expressionElements>
      <expressionIdentifier>
        <name>0|12</name>
        <type>EXPECT</type>
      </expressionIdentifier>
      <factIdentifier>
        <name>Fine</name>
        <className>Fine</className>
      </factIdentifier>
      <className>number</className>
      <factAlias>Fine</factAlias>
      <expressionAlias>Amount</expressionAlias>
      <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
      <columnWidth>114.0</columnWidth>
    </FactMapping>
    <FactMapping>
      <expressionElements>
        <ExpressionElement>
          <step>Should the driver be suspended?</step>
        </ExpressionElement>
      </expressionElements>
      <expressionIdentifier>
        <name>0|13</name>
        <type>EXPECT</type>
      </expressionIdentifier>
      <factIdentifier>
        <name>Should the driver be suspended?</name>
        <className>Should the driver be suspended?</className>
      </factIdentifier>
      <className>string</className>
      <factAlias>Should the driver be suspended?</factAlias>
      <expressionAlias>value</expressionAlias>
      <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
      <columnWidth>114.0</columnWidth>
    </FactMapping>
  </factMappings>
</scesimModelDescriptor>
<scesimData>
  <Scenario>
    <factMappingValues>
      <FactMappingValue>
        <factIdentifier>
          <name>Scenario description</name>
          <className>java.lang.String</className>
        </factIdentifier>
        <expressionIdentifier>
          <name>Description</name>
          <type>OTHER</type>
        </expressionIdentifier>
        <rawValue class="string">Above speed limit: 10km/h and 30 km/h</rawValue>
      </FactMappingValue>
      <FactMappingValue>
        <factIdentifier>
          <name>Driver</name>
          <className>Driver</className>
```

```
    </factIdentifier>
    <expressionIdentifier>
      <name>0|1</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">10</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|6</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">&amp;quot;speed&amp;quot;</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|7</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">100</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|8</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">120</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Fine</name>
      <className>Fine</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|11</name>
      <type>EXPECT</type>
    </expressionIdentifier>
    <rawValue class="string">3</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Fine</name>
      <className>Fine</className>
    </factIdentifier>
```

```
          <expressionIdentifier>
            <name>0|12</name>
            <type>EXPECT</type>
          </expressionIdentifier>
          <rawValue class="string">500</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>Should the driver be suspended?</name>
            <className>Should the driver be suspended?</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>0|13</name>
            <type>EXPECT</type>
          </expressionIdentifier>
          <rawValue class="string">&amp;quot;No&amp;quot;</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>#</name>
            <className>java.lang.Integer</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>Index</name>
            <type>OTHER</type>
          </expressionIdentifier>
          <rawValue class="string">1</rawValue>
        </FactMappingValue>
      </factMappingValues>
    </Scenario>
  </scesimData>
</simulation>
<background>
  <scesimModelDescriptor>
    <factMappings>
      <FactMapping>
        <expressionElements/>
        <expressionIdentifier>
          <name>1|1</name>
          <type>GIVEN</type>
        </expressionIdentifier>
        <factIdentifier>
          <name>Empty</name>
          <className>java.lang.Void</className>
        </factIdentifier>
        <className>java.lang.Void</className>
        <factAlias>Instance 1</factAlias>
        <expressionAlias>PROPERTY 1</expressionAlias>
        <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
        <columnWidth>114.0</columnWidth>
      </FactMapping>
    </factMappings>
  </scesimModelDescriptor>
  <scesimData>
    <BackgroundData>
      <factMappingValues>
```

```
        <FactMappingValue>
          <factIdentifier>
            <name>Empty</name>
            <className>java.lang.Void</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>1|1</name>
            <type>GIVEN</type>
          </expressionIdentifier>
        </FactMappingValue>
      </factMappingValues>
    </BackgroundData>
  </scesimData>
 </background>
 <settings>
  <dmnFilePath>src/main/resources/org/kie/example/traffic/traffic_violation/Traffic
Violation.dmn</dmnFilePath>
  <type>DMN</type>
  <fileName></fileName>
  <dmnNamespace>https://kiegroup.org/dmn/_A4BCA8B8-CF08-433F-93B2-
A2598F19ECFF</dmnNamespace>
  <dmnName>Traffic Violation</dmnName>
  <skipFromBuild>false</skipFromBuild>
  <stateless>false</stateless>
 </settings>
 <imports>
  <imports/>
 </imports>
</ScenarioSimulationModel>
```

XML 响应**示例：**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Test Scenario successfully executed">
    <scenario-simulation-result>
        <run-count>5</run-count>
        <ignore-count>0</ignore-count>
        <run-time>31</run-time>
    </scenario-simulation-result>
</response>
```

第 76 章 使用示例 MORTGAGES 项目创建测试场景

本章介绍了使用测试场景设计器从 Business Central 提供的示例 Mortgages 项目中创建和执行测试场景。本章中的测试场景示例基于 Mortgages 项目中的 定价 loans 指导决策表。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects，然后点击 Mortgages。
2. 如果项目未列在 Projects 下，请从 MySpace 中点击 Try Samples → Mortgages → OK。

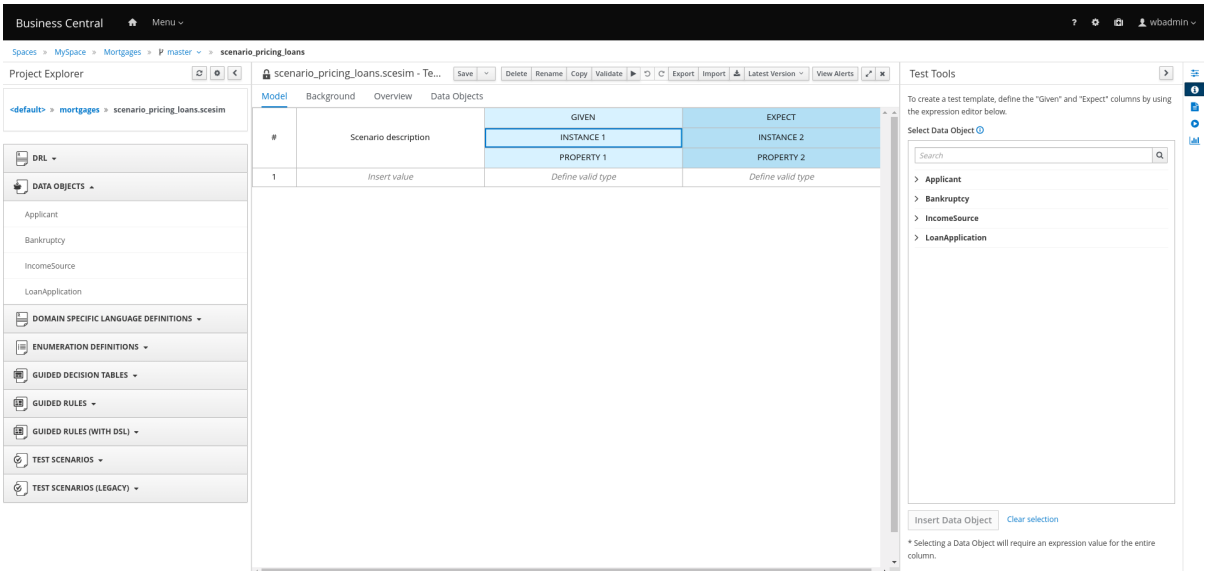
   此时会出现 Assets 窗口。
3. 点 Add Asset → Test Scenario。
4. 输入 scenario_pricing_loans 作为 Test Scenario 名称，然后从软件包下拉列表中选择默认的 mortgages.mortgages 软件包。

   您选择的软件包必须包含所有必要规则资产。
5. 选择 RULE 作为 Source 类型。
6. 点击 Ok 在测试场景设计器中创建并打开测试场景。
7. 展开 Project Explorer 并验证以下内容：

   - Application li cant ,Bankruptcy,IncomeSource 和 LoanApplication data 对象存在。
   - 存在定价 金指导决策表。
   - 验证 Test Scenario 下列出了新的测试场景

8. 在验证了所有内容是否就位后，返回测试场景设计器的 Model 选项卡，并根据可用的数据对象定义 GIVEN 和 EXPECT 数据。

图 76.1. 空白测试场景设计器



9. 定义 GIVEN 列详情：

a. 在 GIVEN 列标题下，单击名为 INSTANCE 1 的单元。

b. 在 Test Tools 面板中，选择 LoanApplication data 对象。

c. 点 Insert Data Object。

10. 要为数据对象创建属性，请右键单击属性标题单元格，根据需要选择 Insert 列 或 Insert 列。在本例中，您需要在 GIVEN 列下创建两个属性单元格。

11. 选择第一个属性标头单元：

a. 在 Test Tools 面板中，选择并展开 LoanApplication data 对象。

b. 点 数量。

c.

单击 **Insert Data Object，将 data** 对象字段映射到属性标题单元。

12.

选择**第二个属性**标头单元：

a.

在 Test Tools 面板中，选择**并展开 LoanApplication data** 对象。

b.

单击 **stored。**

c.

点 Insert Data Object。

13.

选择**第三个属性**标头单元：

a.

在 Test Tools 面板中，选择**并展开 LoanApplication data** 对象。

b.

点 lengthYears

c.

点 Insert Data Object。

14.

右键单击 **LoanApplication** 标头单元格，**并**选择 **Insert 列右。**为**右**创建一个新的 **GIVEN 列。**

15.

选择**新的**标头单元：

a.

在 Test Tools 面板中，选择 **IncomeSource data** 对象。

b.

点 Insert Data Object 将 data 对象映射到标头单元。

16.

选择 **IncomeSource 下的属性**标头单元：

a.

在 Test Tools 面板中，选择并展开 IncomeSource data 对象。

b.

点 type。

c.

单击 Insert Data Object，将 data 对象字段映射到属性标题单元。

现在，您已定义了所有 GIVEN 列单元。

17.

接下来，定义 EXPECT 列详情：

a.

在 EXPECT 列标题下，单击名为 INSTANCE 2 的单元。

b.

从 Test Tools 面板中，选择 LoanApplication data 对象。

c.

点 Insert Data Object。

18.

要为数据对象创建属性，请右键单击属性标题单元格，根据需要选择 Insert 列 或 Insert 列。在 EXPECT 列下创建两个属性单元格。

19.

选择第一个属性标头单元：

a.

在 Test Tools 面板中，选择并展开 LoanApplication data 对象。

b.

单击 已批准。

c.

单击 Insert Data Object，将 data 对象字段映射到属性标题单元。

20.

选择第二个属性标头单元：

a. 在 Test Tools 面板中，选择并展开 LoanApplication data 对象。

b. 点 insuranceCost。

c. 单击 Insert Data Object，将 data 对象字段映射到属性标题单元。

21. 选择第三个属性标头单元：

a. 在 Test Tools 面板中，选择并展开 LoanApplication data 对象。

b. 单击 已批准的。

c. 单击 Insert Data Object，将 data 对象字段映射到属性标题单元。

22. 要定义测试场景，请在第一行中输入以下数据：

- 输入 Row 1 测试场景 作为 Scenario Description、150000 作为 数量，19 0000 作为 存款 值，30 作为 lengthYears，以及 Asset 作为 GIVEN 列值 的类型。
- 输入" 批准 "，0 作为 保险Cost，2 作为 EXPECT 列值的批准。

23. 然后在第二行中输入以下数据：

- 输入 Row 2 测试场景 作为 Scenario Description、100002 作为数量，2999 作为 存款 量，20 作为 lengthYears，以及 Job 作为 GIVEN 列值 的类型。
- 输入" 批准 "，10 作为 保险Cost 和 6，作为 EXPECT 列值的批准。

24. 定义了所有 GIVEN、EXPECT 和其他用于情境数据后，点击测试场景设计器中的 Save 来保存您的工作。

25. 单击右上角的 **Run Test**，以运行 **.scesim** 文件。

测试结果会显示在 **Test Report** 面板中。单击 **View Alerts** 以显示 **Alerts** 部分中的消息。如果测试失败，引用窗口底部的 **Alerts** 部分中的消息，查看并更正该场景中的所有组件，然后重试验证该场景直到情况通过为止。

26. 在进行所有必要的更改后，点击测试场景设计器中的 **Save** 以保存您的作业。

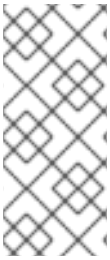
第 77 章 业务中心测试场景（传统）

Red Hat Process Automation Manager 目前支持新的 测试场景 设计人员和之前的 测试场景 (Legacy) 设计人员。默认设计器是新的测试场景设计器，它支持测试规则和 DMN 模型，并提供测试场景增强的整体用户体验。如果需要，您可以继续使用旧的测试场景程序，该设计只支持基于规则的测试场景。

77.1. 创建并运行测试场景（传统）

您可以在 Business Central 中创建测试场景，在部署前测试商业规则数据的功能。基本测试场景必须至少包含以下数据：

- 相关数据对象
- GIVEN facts
- EXPECT 结果



注意

传统的测试场景设计器支持 LocalDate java 内置数据类型。您可以使用 dd-mmm-yyy date 格式中的 LocalDate java 内置数据类型。例如，您可以使用 17-Oct-2020 日期格式设定此设置。

通过这些数据，测试场景可以根据定义的事实验证该规则实例的预期和实际结果。您还可以将 CALL METHOD 和任何可用的 全局 添加到测试场景，但这些场景设置是可选的。

流程

1. 在 Business Central 中，前往 Menu → Design → Projects 并点项目名称。
2. 点 Add Asset → Test Scenarios(Legacy)。
3. 输入信息 性 Test Scenario 名称并选择相应的 软件包。您指定的软件包必须是分配或分配必要规则资产的同一软件包。您可以将任何软件包中的数据对象导入到资产的设计程序中。

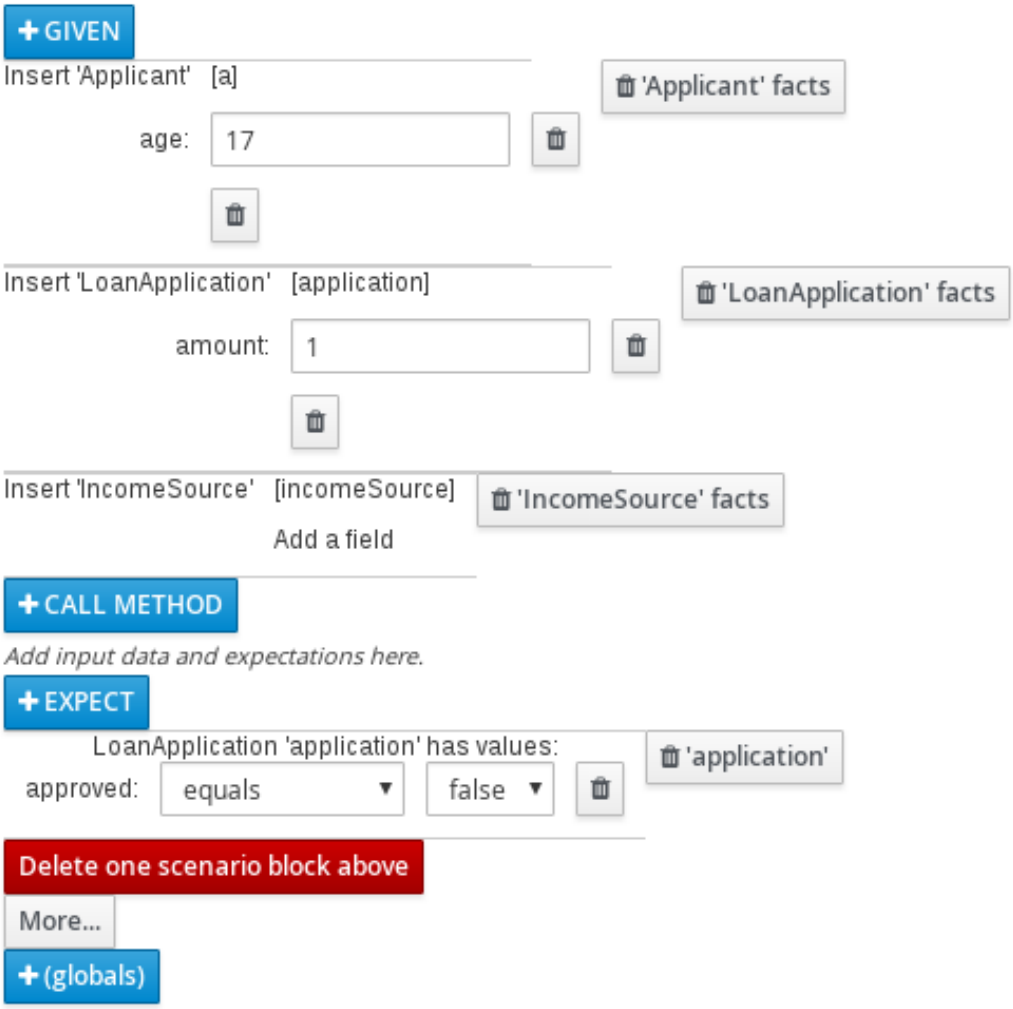
4. 点 Ok 创建测试场景。

新的测试场景现在列在 Project Explorer 的 Test Scenarios 面板中，

5. 点 Data Objects 选项卡，验证是否列出了您要测试的规则所需的所有数据对象。如果没有，请单击 New item 以从其他软件包导入所需的数据对象，或者在您的软件包中创建 数据对象。

6. 在所有数据对象都就位后，返回测试场景设计器的 Model 选项卡，并根据可用数据对象定义场景的 GIVEN 和 EXPECT 数据。

图 77.1. 测试场景设计器



GIVEN 部分定义测试的输入事实。例如，如果项目的下限规则拒绝了 21 日期为 applicants 的 applicants，那么测试情景中的 GIVEN 事实可能是设置为小于 21 的 年龄。

EXPECT 部分根据 GIVEN 输入事实定义预期的结果。也就是说，GIVEN 输入事实 EXPECT 其他事实是有效的或整个要激活的规则。例如，在情景中，在 21 年龄下给定应用程序的事实将 EXPECT 结果为 LoanApplication，并带有 approved set 为 false （因为相关不足导致的不足），也可以是整体激活 Underage 规则。

7. 可选：在测试场景中添加 CALL METHOD 和任何 全局 ：

   - CALL METHOD： 使用这个方法在发起规则执行时从另一事实调用方法。点 CALL METHOD，选择一个事实，然后点击  来选择要调用的方法。您可以从 Java 库或通过为项目导入的 JAR 调用任何 Java 类方法（如 ArrayList 中的方法）。

   - globals： 使用它来在测试场景中验证的项目中添加您要验证的任何全局变量。单击 globals 以选择要验证的变量，然后在测试场景设计器中点击全局名称并定义要应用到全局变量的字段值。如果没有可用的全局变量，则必须将其创建为 Business Central 中的新资产。全局变量命名对象，它们对决策引擎可见，但与对象与事实上的对象不同。全局对象中的更改不会触发规则重新评估。

8. 点测试场景设计器底部的更多内容，根据需要将其他数据块添加到相同的场景文件中。

9. 定义了所有 GIVEN、EXPECT 和其他用于情境数据后，点击测试场景设计器中的 Save 来保存您的工作。

10. 点击右上角的 Run scenario 以运行此 .scenario 文件，或者点击 Run all scenarios 运行项目软件包中的所有已保存的 .scenario 文件（如果有多项）。虽然 Run scenario 选项不需要保存单独的 .scenario 文件，但 运行所有场景 选项都需要保存所有 .scenario 文件。

    如果测试失败，解决窗口底部的 Alerts 消息中描述的任何问题，查看该方案中的所有组件，然后再次尝试验证该场景直到情况通过为止。

11. 在所有更改后，单击测试场景设计器中的 Save，以保存您的作业。

77.1.1. 在测试场景中添加 GIVEN 事实（传统）

GIVEN 部分定义测试的输入事实。例如，如果项目的下限规则拒绝了 21 日期为 applicants 的 applicants，那么测试情景中的 GIVEN 事实可能是设置为小于 21 的 年龄。

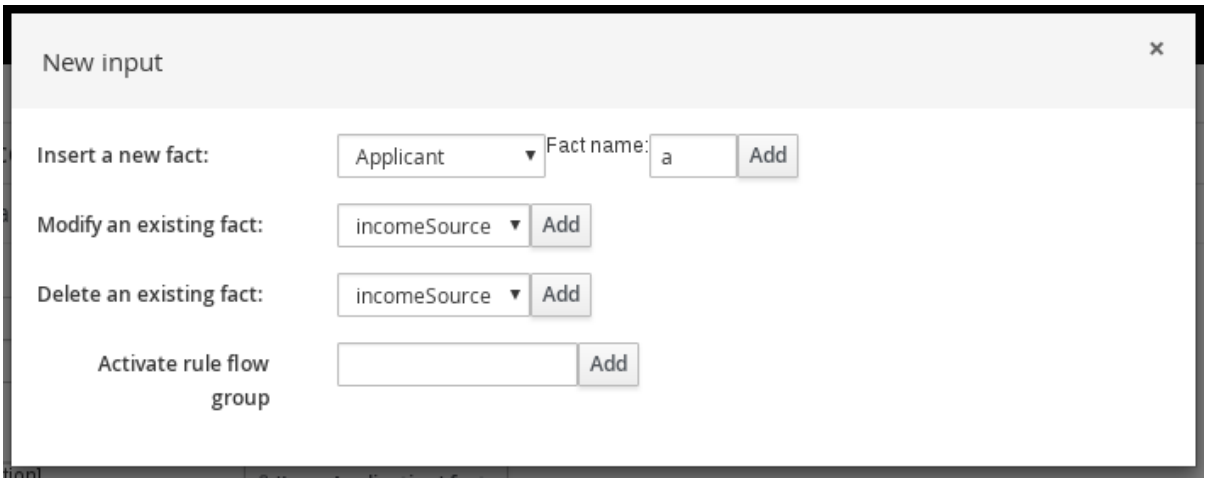
先决条件

- 您的测试场景需要的所有数据对象已创建或导入，并列在 Test Scenarios(Legacy) 设计人员 的数据对象 选项卡中。

流程

1. 在 Test Scenarios(Legacy) designer 中，单击 GIVEN 以通过可用事实打开 New input 窗口。

图 77.2. 在测试场景中添加 GIVEN 输入



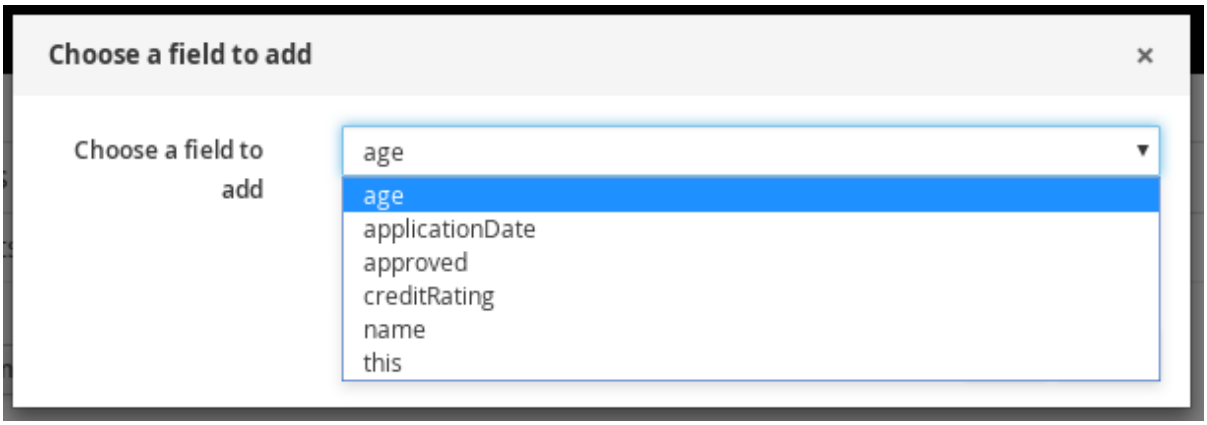
该列表包括以下选项，具体根据测试场景设计器 的数据对象 选项卡中的数据对象：

- 插入新事实： 使用这个事实并修改其字段值。输入事实的变量 作为事实 名称。
- 修改现有事实： （仅在添加另一事实后应用）。 使用此选项指定之前插入的事实，以便在执行情景期间在决策引擎中修改。
- 删除现有事实： （仅在添加另一事实后应用）。 使用此选项指定之前插入的事实，以便在执行场景之间的决策引擎中删除。
- 激活规则流组： 使用这个指定要激活的规则流组，以便可以测试该组中的所有规则。

2. 为所需输入选项选择事实，然后单击添加。例如，将 Insert a new facts: to Applicant 设置为 Applica nt，并为 事实名称 输入或任何其他变量。

3. 单击测试场景设计器中的事实，再选择要修改的字段。

图 77.3. 修改 fact 字段



4. 点击编辑图标(

)并从以下字段值中选择：

- 字面值： 创建一个打开字段，在其中输入一个特定字面值。
- bound variable： 将字段的值设置为绑定到所选变量的事实。字段类型必须与 bound 变量类型匹配。
- 创建新事实： 使您能够创建新事实并将其分配为父事实的字段值。然后，您可以单击测试场景设计中的子事实，并同样地分配字段值或嵌套其他事实。

5. 继续为场景添加任何其他 GIVEN 输入数据，并在测试场景器中点击 Save 来保存您的工作。

77.1.2. 添加 EXPECT 结果会导致测试场景（传统）

EXPECT 部分根据 GIVEN 输入事实定义预期的结果。也就是说，GIVEN 输入事实 EXPECT 其他指定的事实是有效的或整个要激活的规则。例如，在情景中，在 21 年龄下给定应用程序的事实将 EXPECT 结果为 LoanApplication，并带有 approved set 为 false （因为相关不足导致的不足），也可以是整体激活 Underage 规则。

先决条件

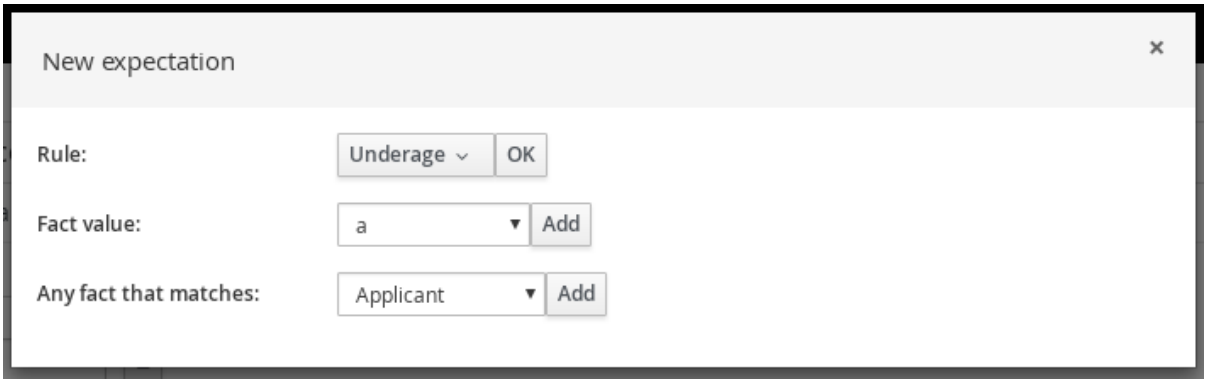
-

您的测试场景需要的所有数据对象已创建或导入，并列在 Test Scenarios(Legacy) 设计人员 的数据对象 选项卡中。

流程

1. 在 Test Scenarios(Legacy) designer 中，单击 EXPECT 以打开 New expectation 窗口及可用事实。

图 77.4. 在测试场景中添加 EXPECT 结果



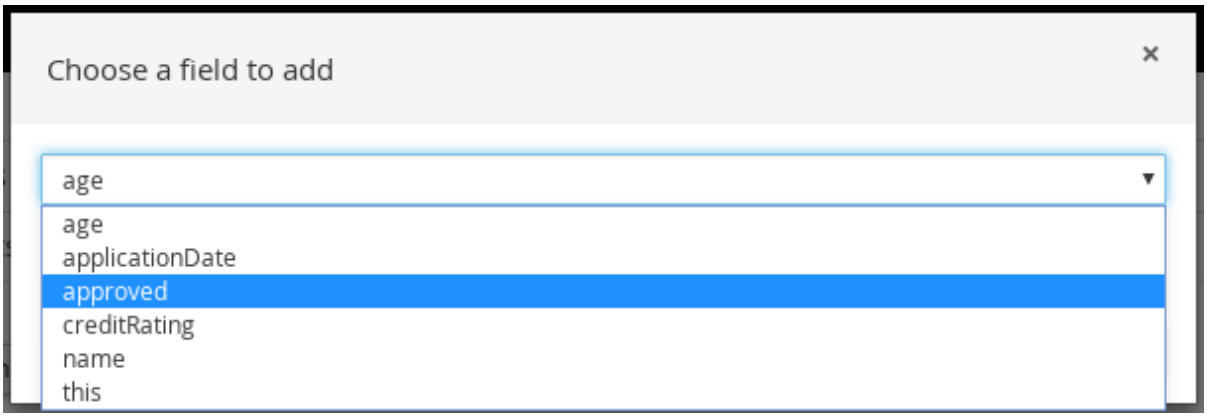
这个列表包括以下选项，具体取决于 GIVEN 部分中的数据，以及测试场景设计器的 Data Objects 选项卡中可用的数据对象：

- rule ：使用这个方法，在项目中指定预期作为 GIVEN 输入时应激活的特定规则。键入应激活的规则的名称，或者从规则列表中选择，然后在测试场景设计器中指定应激活规则的次数。
- 事实值：使用它来选择事实，并为它定义值，以作为 GIVEN 部分中定义的事实。事实由之前为 GIVEN 输入定义的事实 名称 列出。
- 匹配的任何事实：使用此项来验证由于 GIVEN 输入而至少存在具有指定值的事实。

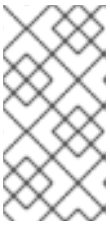
2. 为所需的预期（如 事实值：应用程序）选择事实，然后单击添加 或 确定。

3. 单击测试场景设计器中的事实，再选择要添加的字段并进行修改。

图 77.5. 修改 fact 字段



4. 将字段值设置为当 GIVEN 输入后（如 已批准 | 等于 | false）后，应该有效。



注意

在旧的测试场景设计器中，您可以在 EXPECT 字段中使用 ["value1", "value2"] 字符串格式来验证字符串列表。

5. 继续为场景添加任何其他 EXPECT 输入数据，并在测试场景器中点击 Save 来保存您的工作。

6. 在定义并保存了所有 GIVEN、EXPECT 和其他情境数据后，点击右上角的 Run scenario 以运行此 .scenario 文件，或者点击 Run all scenarios 在项目软件包中运行所有已保存的 .scenario 文件（如果存在多个）。虽然 Run scenario 选项不需要保存单独的 .scenario 文件，但 运行所有场景 选项都需要保存所有 .scenario 文件。

如果测试失败，解决窗口底部的 Alerts 消息中描述的任何问题，查看该方案中的所有组件，然后再次尝试验证该场景直到情况通过为止。

7. 在所有更改后，单击测试场景设计器中的 Save，以保存您的作业。

第 78 章 传统和新测试场景设计器的功能比较

Red Hat Process Automation Manager 支持新的测试场景设计程序和之前的测试场景(Legacy)设计人员。

默认设计器是新的测试场景设计器，它支持测试规则和 DMN 模型，并提供测试场景的增强整体用户体验。您可以继续使用旧的测试场景设计程序，该设计只支持基于规则的测试场景。

重要

新的测试场景设计器具有改进的布局和功能集，并持续开发。但是，旧的测试场景设计程序已在 Red Hat Process Automation Manager 7.3.0 中弃用，并将在以后的 Red Hat Process Automation Manager 发行版本中删除。

下表重点介绍了 Red Hat Process Automation Manager 支持传统和新的测试场景设计程序的主要功能，以帮助您在项目中决定合适的测试场景设计程序。

- + 表示该功能存在于测试场景设计器中。
- - 表示测试场景设计中不存在该功能。

表 78.1. 传统和新测试场景设计器的主要特性

功能和亮点	新的设计器	旧设计器	Documentation
创建并运行测试场景 ● 您可以在 Business Central 中创建测试场景，在部署前测试商业规则数据的功能。 ● 基本测试场景必须至少有相关的数据对象 GIVEN 事实和 EXPECT 结果。 ● 您可以运行测试以验证您的业务规则和数据。	+	+	● 有关创建规则和基于 DMN 的测试场景的详情请参考 第 65 章 *测试场景模板*。 ● 有关运行测试场景的详情请参考 第 71 章 *运行测试场景*。 ● 有关创建和运行测试场景（传统）的详情，请参考 第 77.1 节 “创建并运行测试场景（传统）”。

功能和亮点	新的设计器	旧设计器	Documentation
在测试场景中添加 GIVEN 事实 • 您可以插入并验证测试的 GIVEN 事实。	+	+	• 有关在新的测试场景设计器中添加 GIVEN 事实的更多信息，请参阅 第 65 章 *测试场景模板*。 • 有关在测试场景（传统）中添加 GIVEN 事实的更多信息，请参阅 第 77.1.1 节 “在测试场景中添加 GIVEN 事实（传统）”。
添加 EXPECT 会导致测试情况 • EXPECT 部分根据 GIVEN 输入事实定义预期的结果。 • 它代表根据提供的信息检查的确切值的对象及其字段。	+	+	• 有关添加 EXPECT 生成新测试场景程序的详情，请参考 第 65 章 *测试场景模板*。 • 有关添加 EXPECT 结果在测试场景中（传统）的详情，请参考 第 77.1.2 节 “添加 EXPECT 结果会导致测试场景（传统）”。
KIE 会话 • 您可以在测试场景级别设置中设置 KIE 会话。	+	+	不适用
测试场景级别的 KIE 基础 • 您可以在测试场景级别设置中设置 KIE 基础。	-	+	不适用
KIE 基础项目级别 • 您可以在项目级别设置上设置 KIE 基础。	+	+	不适用
模拟日期和时间 • 您可以为旧的测试场景设计程序设置模拟日期和时间。	-	+	不适用

功能和亮点	新的设计器	旧设计器	Documentation
规则流组 ● 您可以指定要激活的规则流组来测试该组中的所有规则。	+	+	● 有关在新测试场景中设置规则流组的更多信息，请参阅 第 64.9.1 节 "为基于规则的测试场景配置全局设置"。 ● 有关在测试场景（传统）中设置规则流组的更多信息，请参阅 第 77.1.1 节 "在测试场景中添加 GIVEN 事实（传统）"。
全局变量 ● 全局变量命名对象，它们对决策引擎可见，但与对象与事实上的对象不同。 ● 为新的测试场景设置全局变量已弃用。 ● 如果要针对不同的场景重复使用数据集，您可以使用 **后台** 实例。	-	+	● 有关新测试场景中 **后台** 实例的详情，请参考 第 68 章 *测试场景中的后台实例*。 ● 有关测试场景（传统）中的全局变量的更多信息，请参阅 第 77.1 节 "创建并运行测试场景（传统）"。
调用方法 ● 您可以在启动规则执行时从另一事实调用方法。 ● 您可以从 Java 库或从导入项目的 JAR 调用任何 Java 类方法。	+	+	● 有关在新测试场景中调用方法的更多信息，请参阅 第 70 章 *测试场景中的表达式语法*。 ● 有关在测试场景中调用方法（传统）的详情，请参考 第 77.1 节 "创建并运行测试场景（传统）"。
修改现有的事实 ● 您可以在执行场景之间的决策引擎中修改前面插入的事实。	-	+	有关在测试场景中修改现有事实（传统）的更多信息，请参阅 第 77.1.1 节 "在测试场景中添加 GIVEN 事实（传统）"。
绑定的变量 ● 您可以将字段的值设置为绑定到所选变量的事实。 ● 在新的测试场景设计器中，您无法在测试场景网格中定义变量，并在 **GIVEN** 或 **EXPECTED** 单元中重复使用它。	-	+	有关如何在测试场景（传统）中设置绑定变量的更多信息，请参阅 第 77.1.1 节 "在测试场景中添加 GIVEN 事实（传统）"。

第 79 章 后续步骤

打包和部署 Red Hat Process Automation Manager 项目

部分 IX. RED HAT PROCESS AUTOMATION MANAGER 中的决策引擎

作为业务规则开发人员，您对红帽流程自动化管理器中的决策引擎的了解可以帮助您设计更有效的业务资产和更具扩展性的决策管理架构。决策引擎是红帽流程自动化管理器组件，用于存储、流程和评估数据，以执行业务规则并达到您定义的决策。本文档描述了在 Red Hat Process Automation Manager 中创建业务规则系统和决策服务时需要考虑的决策引擎的基本概念和功能。

第 80 章 RED HAT PROCESS AUTOMATION MANAGER 中的决策引擎

决策引擎是红帽流程自动化管理器中的引擎。决策引擎存储、流程和评估数据，以执行您定义的业务规则或决策模型。决策引擎的基本功能是将传入数据或 事实 匹配到规则条件，并确定是否以及执行规则。

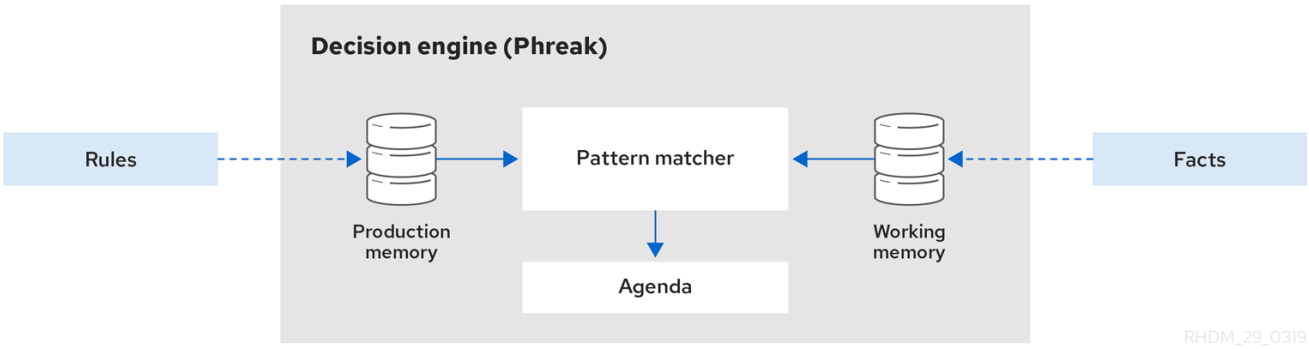
决策引擎使用以下基本组件进行操作：

- 规则： 您定义的业务规则或 DMN 决策。所有规则必须至少包含触发规则的条件和规则指示的操作。
- 事实： 在决策引擎中输入或更改引擎与执行相关规则的规则条件匹配的数据。
- 生产内存： 规则存储在决策引擎中的位置。
- 工作内存： 事实存储在决策引擎中的位置。
- 日程表： 注册和排序激活规则的位置（如果适用）准备执行。

当业务用户或自动化系统在 Red Hat Process Automation Manager 中添加或更新规则时，该信息将以一个或多个事实的形式插入决策引擎的工作内存中。决策引擎将这些事实与存储在生产内存中的规则条件匹配，以确定符合条件的规则执行。（这一与规则匹配的事实的进程通常被称为 模式匹配。） 满足规则条件时，决策引擎会在日程上激活和注册规则，其中决策引擎将优先排序或冲突规则来准备执行。

下图演示了决策引擎的这些基本组件：

图 80.1. 基本决策引擎组件概述



有关决策引擎中的规则和事实行为的详情和示例，请参阅 第 82 章 在决策引擎中影响和真相维护。

这些核心概念可帮助您更好地了解决策引擎的其他高级组件、流程和子进程，并在 Red Hat Process Automation Manager 中设计更有效的业务资产。

第 81 章 KIE 会话

在 Red Hat Process Automation Manager 中，KIE 会话存储并执行运行时数据。如果您已在 KIE 模块描述符文件(kmodule.xml)中定义 KIE 会话，则由 KIE 的基础创建，或直接从 KIE 容器创建。

kmodule.xml 文件中的 KIE 会话配置示例

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

KIE 基础是您在项目的 KIE 模块描述符文件(kmodule.xml)中定义的仓库，包含 Red Hat Process Automation Manager 中的所有规则、流程和其他业务资产，但不包含任何运行时数据。

kmodule.xml 文件中的 KIE 基本配置示例

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

KIE 会话可以是无状态或有状态。在无状态 KIE 会话中，之前调用 KIE 会话（之前会话状态）中的数据会在会话调用之间丢弃。在有状态 KIE 会话中，数据会被保留。您使用的 KIE 会话的类型取决于您的项目要求以及希望数据来自不同资产调用的数据。

81.1. 无状态 KIE 会话

无状态 KIE 会话是一种会话，不使用推断时间对事实进行迭代更改。在无状态 KIE 会话中，来自之前调用 KIE 会话（之前会话状态）中的数据会在会话调用之间丢弃，而在有状态 KIE 会话中，数据会被保留。无状态 KIE 会话的行为和函数类似，因为生成的结果由 KIE 基本的内容以及要在特定时间点上传递给 KIE 会话的数据决定。KIE 会话没有之前传递给 KIE 会话的任何数据的内存。

无状态 KIE 会话通常用于以下用例：

- 验证，例如验证某个人是否有资格获得分流
- 计算（如计算抵抵达高级）
- 路由和过滤，如对传入电子邮件进行排序，或将传入电子邮件发送到目标

例如，请考虑以下驱动程序的许可证数据模型和示例 DRL 规则：

驱动程序许可证应用程序的数据模型

```
public class Applicant {
  private String name;
  private int age;
  private boolean valid;
  // Getter and setter methods
}
```

驱动程序许可证应用程序的 DRL 规则示例

```
package com.company.license

rule "Is of valid age"
when
  $a : Applicant(age < 18)
```

```
then
  $a.setValid(false);
end
```

有效年龄法定性 不长于 18 年。当将 Applicant 对象插入到决策引擎中时，决策引擎将评估每个规则的约束并搜索匹配项。"objectType" 约束总是表示，在评估任何显式字段限制后。变量 $a 是引用规则后果中匹配对象的绑定变量。

注意

美元符号($)是可选的，有助于区分变量名称和字段名称。

在本例中，Red Hat Process Automation Manager 项目的 ~/resources 文件夹中的规则和所有其他文件都使用以下代码构建：

创建 KIE 容器

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

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

此代码编译类路径上找到的所有规则文件，并在 KieContainer 中添加了此编译结果（即 KieModule 对象）。

最后，State lessKieSession 对象由 KieContainer 进行实例化，并针对指定数据执行：

实例化无状态 KIE 会话并输入数据

```
StatelessKieSession kSession = kContainer.newStatelessKieSession();

Applicant applicant = new Applicant("Mr John Smith", 16);
```

```
assertTrue(applicant.isValid());

ksession.execute(applicant);

assertFalse(applicant.isValid());
```

在无状态 KIE 会话配置中，execute（） 调用充当实例化 KieSession 对象的组合方法，添加所有用户数据并执行用户命令、调用 fireAllRules（），然后调用 dispose（）。因此，通过无状态 KIE 会话，您不需要在会话调用后调用 fireAllRules（） 或在使用有状态 KIE 会话时调用 dispose（）。

在这种情况下，指定的申请时间为 18，因此应用程序会下降。

有关更复杂的用例，请参见以下示例。这个示例使用无状态 KIE 会话，并根据对象列表（如集合）执行规则。

为驱动程序许可证应用程序扩展数据模型

```
public class Applicant {
  private String name;
  private int age;
  // Getter and setter methods
}

public class Application {
  private Date dateApplied;
  private boolean valid;
  // Getter and setter methods
}
```

为驱动程序许可证应用程序扩展 DRL 规则集

```
package com.company.license

rule "Is of valid age"
when
```

```
  Applicant(age < 18)
  $a : Application()
then
  $a.setValid(false);
end

rule "Application was made this year"
when
  $a : Application(dateApplied > "01-jan-2009")
then
  $a.setValid(false);
end
```

在无状态 KIE 会话中使用可迭代执行扩展 Java 源

```
StatelessKieSession ksession = kbase.newStatelessKnowledgeSession();
Applicant applicant = new Applicant("Mr John Smith", 16);
Application application = new Application();

assertTrue(application.isValid());
ksession.execute(Arrays.asList(new Object[] { application, applicant })); (1)
assertFalse(application.isValid());

ksession.execute
  (CommandFactory.newInsertIterable(new Object[] { application, applicant })); (2)

List<Command> cmds = new ArrayList<Command>(); (3)
cmds.add(CommandFactory.newInsert(new Person("Mr John Smith"), "mrSmith"));
cmds.add(CommandFactory.newInsert(new Person("Mr John Doe"), "mrDoe"));

BatchExecutionResults results =
ksession.execute(CommandFactory.newBatchExecution(cmds));
assertEquals(new Person("Mr John Smith"), results.getValue("mrSmith"));
```

(1) 针对 Arrays.asList（） 方法生成的可迭代对象集合执行规则的方法。每个集合元素都会在执行任何匹配规则之前插入。execute（Object 对象） 和 execute(Iterable objects) 方法是来自 BatchExecutor 接口的 execute（Command 命令） 方法。

(2) 使用 CommandFactory 接口执行可迭代对象集合。

3

用于使用许多不同的命令或结果输出标识符的 BatchExecutor 和 Command onnectionFactoryy 配置。CommandFactory 接口支持在 BatchExecutor 中使用的其他命令，如 StartProcess、Query、Query 和 SetGlobal。

81.1.1. 无状态 KIE 会话中全局变量

StatelessKieSession 对象支持全局变量（全局），您可以将它们配置为以会话范围的全局、委派全局或执行范围全局解析。

- session -scoped globals：对于会话范围的全局全局，您可以使用方法 get Globals （）返回可访问 KIE 会话全局的全局实例。这些全局用于所有执行调用。请谨慎使用 mutable 全局，因为可以在不同的线程中同时执行执行调用。

  会话范围全局

```
import org.kie.api.runtime.StatelessKieSession;

StatelessKieSession ksession = kbase.newStatelessKieSession();

// Set a global `myGlobal` that can be used in the rules.
ksession.setGlobal("myGlobal", "I am a global");

// Execute while resolving the `myGlobal` identifier.
ksession.execute(collection);
```

- delegate globals：委派 全局值，您可以将值分配给全局（使用 setGlobal(String, Object)），以便该值存储在将标识符映射到值的内部集合中。此内部集合中的标识符具有优先权，超过任何提供的委派。如果在此内部集合中找不到标识符，则使用委派全局（若有）。

- 执行范围的全局全局：对于执行范围的全局，您可以使用 Command 对象来设置传递给命令执行程序接口以进行特定全局解析的全局。

CommandExecutor 接口还允许您使用全局标识符、插入的事实和查询结果来导出数据：

全局、插入的事实和查询结果的标识符

```
import org.kie.api.runtime.ExecutionResults;

// Set up a list of commands.
List cmds = new ArrayList();
cmds.add(CommandFactory.newSetGlobal("list1", new ArrayList(), true));
cmds.add(CommandFactory.newInsert(new Person("jon", 102), "person"));
cmds.add(CommandFactory.newQuery("Get People" "getPeople"));

// Execute the list.
ExecutionResults results = ksession.execute(CommandFactory.newBatchExecution(cmds));

// Retrieve the `ArrayList`.
results.getValue("list1");
// Retrieve the inserted `Person` fact.
results.getValue("person");
// Retrieve the query as a `QueryResults` instance.
results.getValue("Get People");
```

81.2. 有状态 KIE 会话

有状态 KIE 会话是一种会话，用于随着时间的推移对事实进行迭代更改。在有状态 KIE 会话中，来自之前调用 KIE 会话（之前会话状态）的数据会在会话调用之间保留，而在无状态 KIE 会话中，数据会被丢弃。

警告

确保在运行有状态 KIE 会话后调用 dispose（） 方法，以便在会话调用之间不会发生内存泄漏。

有状态 KIE 会话通常用于以下用例：

- 监控 （如监控库存市场并自动化购买流程）

- 诊断，如运行故障查找进程或医疗诊断流程
- 物流，如 parcel 跟踪和交付配置
- 确保合规性，如验证市场交易的合法性

例如，请考虑以下触发警报数据模型和示例 DRL 规则：

sprinklers 和触发警报的数据模型

```
public class Room {
  private String name;
  // Getter and setter methods
}

public class Sprinkler {
  private Room room;
  private boolean on;
  // Getter and setter methods
}

public class Fire {
  private Room room;
  // Getter and setter methods
}

public class Alarm { }
```

激活 sprinklers 和 alarm 设置的 DRL 规则示例

```
rule "When there is a fire turn on the sprinkler"
when
  Fire($room : room)
  $sprinkler : Sprinkler(room == $room, on == false)
then
  modify($sprinkler) { setOn(true) };
  System.out.println("Turn on the sprinkler for room "+$room.getName());
end
```

```
rule "Raise the alarm when we have one or more fires"
when
    exists Fire()
then
    insert( new Alarm() );
    System.out.println( "Raise the alarm" );
end

rule "Cancel the alarm when all the fires have gone"
when
    not Fire()
    $alarm : Alarm()
then
    delete( $alarm );
    System.out.println( "Cancel the alarm" );
end


rule "Status output when things are ok"
when
    not Alarm()
    not Sprinkler( on == true )
then
    System.out.println( "Everything is ok" );
end
```

对于 当一个触发时，触发了 sprinkler 规则，在发生触发时，会为使用该房间创建 Fire 类的实例并插入到 KIE 会话中。该规则为 Fire 实例中匹配特定 空间 添加约束，以便只检查该空间的 sprinkler。执行此规则时，sprinkler 激活。其他示例规则决定了警报何时激活或相应地取消激活。

无状态 KIE 会话依赖于标准 Java 语法来修改字段，有状态 KIE 会话依赖于规则中的修改语句来通知更改的决策引擎。然后，决策引擎会超出变化的原因，并评估对后续规则执行的影响。这个过程是决策引擎使用差异和 真相维护 的一部分，在有状态 KIE 会话中至关重要。

在本例中，Red Hat Process Automation Manager 项目的 ~/resources 文件夹中的示例规则和所有其他文件都使用以下代码构建：

创建 KIE 容器

```
KieServices kieServices = KieServices.Factory.get();
KieContainer kContainer = kieServices.getKieClasspathContainer();
```

此代码编译类路径上找到的所有规则文件，并在 KieContainer 中添加了此编译结果（即 KieModule 对象）。

最后，KieSession 对象从 KieContainer 进行实例化，并针对指定数据执行：

实例化有状态 KIE 会话并输入数据

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

String[] names = new String[]{"kitchen", "bedroom", "office", "livingroom"};
Map<String,Room> name2room = new HashMap<String,Room>();
for( String name: names ){
    Room room = new Room( name );
    name2room.put( name, room );
    ksession.insert( room );
    Sprinkler sprinkler = new Sprinkler( room );
    ksession.insert( sprinkler );
}

ksession.fireAllRules();
```

控制台输出

```
> Everything is ok
```

添加数据后，决策引擎会完成所有模式匹配，但没有执行规则，因此会显示配置的验证消息。在新的数据触发规则条件时，决策引擎会执行规则来激活警报，然后取消已激活的警报：

输入新数据来触发规则

-

```
Fire kitchenFire = new Fire( name2room.get( "kitchen" ) );
Fire officeFire = new Fire( name2room.get( "office" ) );

FactHandle kitchenFireHandle = ksession.insert( kitchenFire );
FactHandle officeFireHandle = ksession.insert( officeFire );

ksession.fireAllRules();
```

控制台输出

```
> Raise the alarm
> Turn on the sprinkler for room kitchen
> Turn on the sprinkler for room office
```

```
ksession.delete( kitchenFireHandle );
ksession.delete( officeFireHandle );

ksession.fireAllRules();
```

控制台输出

```
> Cancel the alarm
> Turn off the sprinkler for room office
> Turn off the sprinkler for room kitchen
> Everything is ok
```

在本例中，为返回的 factHandle 对象保留一个引用。事实句柄是插入的实例的内部引擎引用，让实例被调整或修改。

如示例所示，来自之前有状态 KIE 会话（激活的警报）的数据和结果会影响随后的会话（alarm 取消）的调用。

81.3. KIE 会话池

在有大量 KIE 运行时数据和高系统活动的用例中，可能会创建 KIE 会话并经常被处理。KIE 会话的高可用性并非始终消耗大量时间，但当过渡数以百万计的时间时，流程可能会成为瓶颈并需要大量清理工作。

对于这些高容量情况，您可以使用 KIE 会话池，而不是很多单独的 KIE 会话。要使用 KIE 会话池，您可以从 KIE 容器获取 KIE 会话池，定义池中初始 KIE 会话数量，并尽可能从那个池中创建 KIE 会话：

KIE 会话池示例

```
// Obtain a KIE session pool from the KIE container
KieContainerSessionsPool pool = kContainer.newKieSessionsPool(10);

// Create KIE sessions from the KIE session pool
KieSession kSession = pool.newKieSession();
```

在这个示例中，KIE 会话池从 10 个 KIE 会话开始，但您可以指定所需的 KIE 会话数。此整数值是最初在池中创建的 KIE 会话的数量。如果运行的应用程序需要，池中 KIE 会话的数量可以动态增长超过该值。

在定义了 KIE 会话池后，在下次使用 KIE 会话池后，您下次使用 KIE 会话并在其上调用 dispose() 时，会重置 KIE 会话并推送回池，而不是销毁。

KIE 会话池通常适用于有状态 KIE 会话，但 KIE 会话池还可影响您使用多个 execute() 调用的无状态 KIE 会话。当您直接从 KIE 容器创建无状态 KIE 会话时，KIE 会话将继续为每个 execute() 调用内部创建新的 KIE 会话。相反，当您从 KIE 会话池中创建无状态 KIE 会话时，KIE 会话仅使用该池提供的特定 KIE 会话。

当使用 KIE 会话池完成时，您可以调用 shutdown() 方法以避免内存泄漏。或者，您也可以在 KIE 容器上调用 dispose() 来关闭从 KIE 容器创建的所有池。

第 82 章 在决策引擎中影响和真相维护

决策引擎的基本功能是将数据与业务规则匹配，并确定是否执行规则。为确保相关的数据应用于适当的规则，决策引擎会基于现有知识进行评分，并根据推断的信息执行操作。

例如，以下 DRL 规则决定了 adults 的年龄要求，如在总线密码短语策略中：

定义年龄要求的规则

```
rule "Infer Adult"
when
  $p : Person(age >= 18)
then
  insert(new IsAdult($p))
end
```

根据此规则，决策引擎是个人是dult 还是一个子级，它会执行指定的操作（ 然后是 结果）。每个已有 18 周年或更早的人都会在工作内存中插入了 IsAdult 实例。然后可以在任何规则中调用这种年龄和总线传递关系，比如以下规则片段：

```
$p : Person()
IsAdult(person == $p)
```

在很多情况下，规则系统中的新数据是其他规则执行的结果，这种新数据可能会影响其他规则的执行。如果决策引擎因为执行规则而断言数据，则决策引擎利用真相维护来证明断言并在将信息应用到其他规则时执行真实性。真相维护还有助于识别不一致和处理预测。例如，如果执行两条规则并导致分类操作，则决策引擎根据之前计算的结论选择操作。

决策引擎使用声明或逻辑插入来插入事实：

- 声明插入： 使用 insert（） 定义的 .在声明了插入后，通常会明确指定事实。（一般 使用时插入 的术语称为" 插入"。 ）
- 逻辑插入： Defined with insertLogical（）.逻辑插入后，当插入事实的条件不再为 true 时，插入的事实会自动清空。当没有条件支持逻辑插入时，会重试事实。逻辑插入的事实被视为

由 决策 引擎规定。

例如，以下 DRL 规则使用指定的事实插入来确定发出子总线通过或 adult 总线传递的年龄要求：

提供总线通行证的规则，声明插入

```
rule "Issue Child Bus Pass"
when
  $p : Person(age < 18)
then
  insert(new ChildBusPass($p));
end

rule "Issue Adult Bus Pass"
when
  $p : Person(age >= 18)
then
  insert(new AdultBusPass($p));
end
```

这些规则在决策引擎中无法轻松维护，因为总线死机增长，从子到双lt 总线传递。或者，通过逻辑事实插入，这些规则也可以分隔为总线取用器年龄和规则的规则。事实的逻辑插入取决于 when 子句的真相。

以下 DRL 规则使用逻辑插入来确定子和临时的年龄要求：

子代和同线要求、逻辑插入

```
rule "Infer Child"
when
  $p : Person(age < 18)
then
  insertLogical(new IsChild($p))
end

rule "Infer Adult"
when
  $p : Person(age >= 18)
```

```
then
  insertLogical(new IsAdult($p))
end
```

重要

对于逻辑插入，您的事实对象必须根据 Java 标准覆盖 java.lang.Object 对象中的等和散列 方法。如果两个对象 等于 不同的方法是否相互返回 true，如果它们 的散列 代码方法返回相同的值，则有两个对象相等。如需更多信息，请参阅您的 Java 版本的 Java API 文档。

当规则中的条件为 false 时，该事实会被自动重试。这个行为在本示例中非常有用，因为这两个规则是相互排斥的。在本例中，如果人比 18 年年长，则规则逻辑插入 IsChild 事实。个人已有 18 年或更早时间之后，IsChild 事实会自动调整，并插入 IsAdult 事实。

然后，以下 DRL 规则决定是否发出子总线通过或逻辑方式插入 ChildBusPass 和 AdultBusPass 事实。这个规则配置是可行的，因为决策引擎中的相依性维护系统支持连锁逻辑插入集。

用于发布总线传递的规则，逻辑插入

```
rule "Issue Child Bus Pass"
when
  $p : Person()
    IsChild(person == $p)
then
  insertLogical(new ChildBusPass($p));
end

rule "Issue Adult Bus Pass"
when
  $p : Person()
    IsAdult(person =$p)
then
  insertLogical(new AdultBusPass($p));
end
```

当某个人名超过 18 年时，IsChild 事实和个人的 子进程BusPass 事实会被重新处理。在这组条件下，

您可以关联另一条规则，指出某人在 18 年后必须返回该子通过。当决策引擎自动对 ChildBusPass 对象进行响应时，将执行以下规则来向该用户发送请求：

用于通知总线新传递拥有者的规则

```
rule "Return ChildBusPass Request"
when
  $p : Person()
    not(ChildBusPass(person == $p))
then
  requestChildBusPass($p);
end
```

以下流程图演示了声明和逻辑插入的生命周期：

*图 **82.1.** 规定的插入*

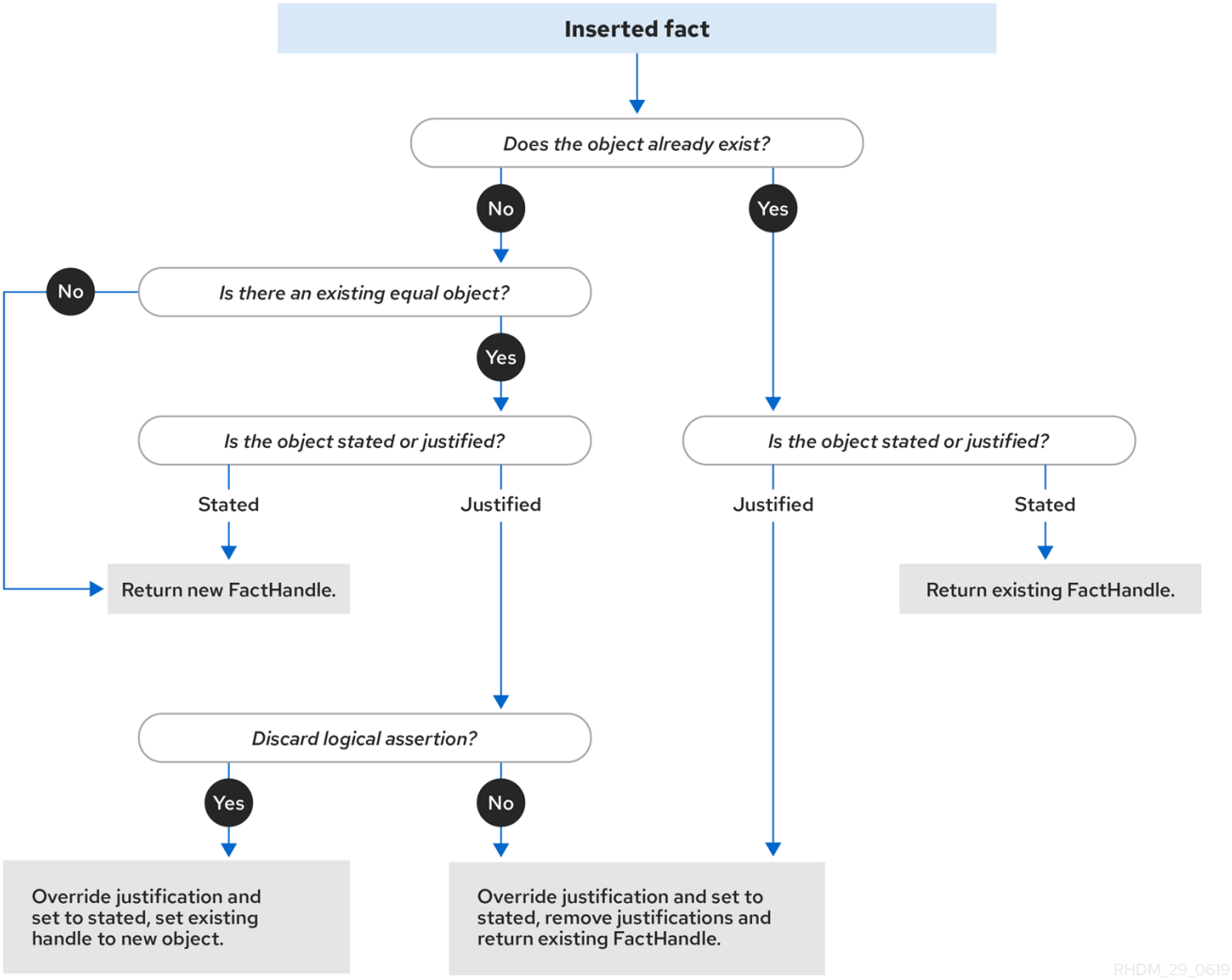
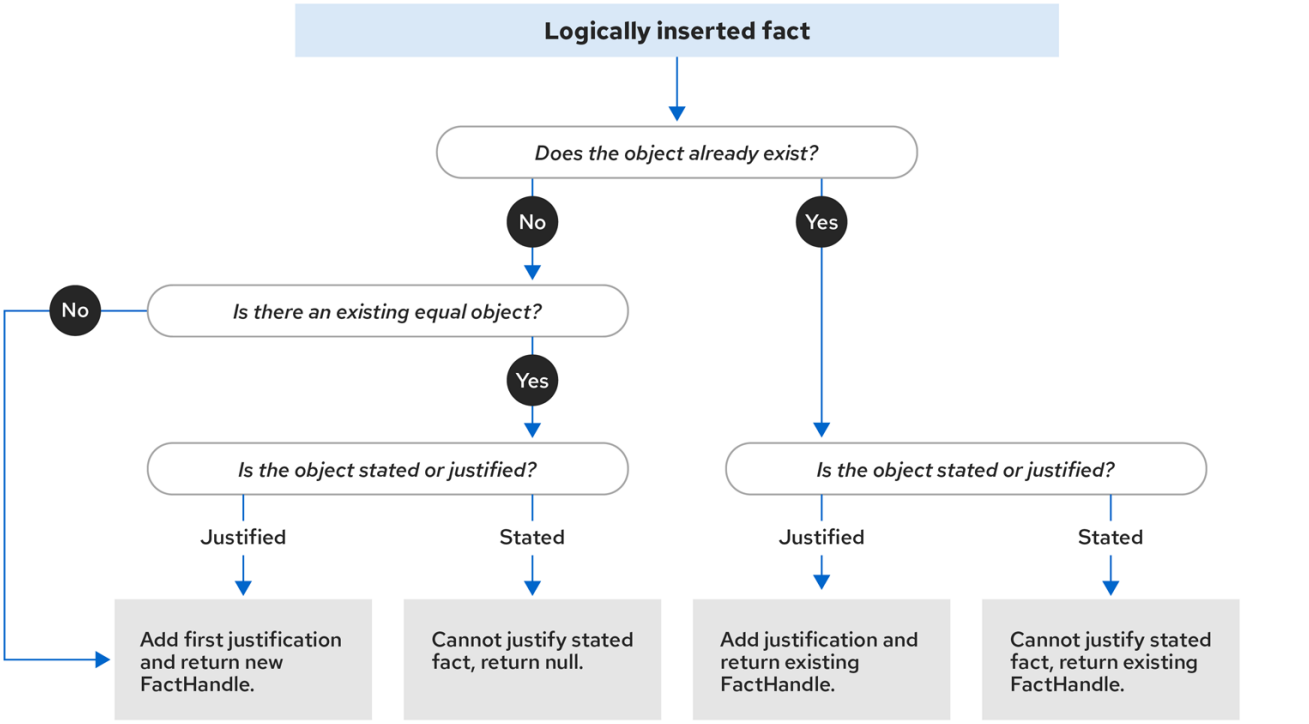

Inserted fact
Does the object already exist?
No
Yes
No
Is there an existing equal object?
Yes
Is the object stated or justified?
Is the object stated or justified?
Stated
Justified
Justified
Stated
Return new FactHandle.
Return existing FactHandle.
Discard logical assertion?
Yes
No
Override justification and set to stated, set existing handle to new object.
Override justification and set to stated, remove justifications and return existing FactHandle.
RHDM_29_0619

图 82.2. 逻辑插入



当决策引擎在规则执行过程中插入对象时，决策引擎通过执行规则来统一对象。对于每个逻辑插入，只能有一个相等的对象，每个后续相同的逻辑插入会增加该逻辑插入的论证计数器。当规则条件变为 untrue 时，将删除拒绝。当存在其他合理性时，会自动对逻辑对象进行重新处理。

82.1. 在决策引擎中事实相等模式

决策引擎支持以下事实制度模式，决定决策引擎如何存储并比较插入的事实：

- 身份 ：（默认）决策引擎使用 IdentityHashMap 存储所有插入的事实。对于每个新事实插入，决策引擎会返回一个新的 factHandle 对象。如果再次插入了事实，则决策引擎会返回原始事实事实，忽略同一事实的重复插入。在这个模式中，只有在决策引擎是具有相同身份的同一个对象时，两个事实才是相同的。

- 等 性：决策引擎使用 HashMap 来存储所有插入的事实。根据插入的事实不等于现有事实，决策引擎才会返回新的事实(fact)对象。 在这个模式中，如果决策引擎的组成方式，两个事实都是一样的，无论身份，都一样。当您希望基于功能相等而非显式身份评估对象时，请使用此模式。

举一个事实相同的模式，请考虑以下示例事实：

事实示例

```
Person p1 = new Person("John", 45);
Person p2 = new Person("John", 45);
```

在 身份 模式中，事实 p1 和 p2 是 Person 类的不同实例，并且被视为单独的对象，因为它们具有单独的身份。在等模式下，事实 p1 和 p2 被视为同一对象，因为它们以相同的方式组成。行为的区别在于，您可以如何与事实句柄进行交互。

例如，假设您将事实 p1 和 p2 插入到决策引擎中，稍后您要检索 p1 的事实句柄。在 身份 模式中，您必须指定 p1 才能返回该确切对象的事实句柄，而以等性模式，您可以指定 p1、 p2 或 新的 Person("John" 45) 来返回事实句柄。

在 身份 模式下插入事实并返回事实(fact)的代码示例

```
ksession.insert(p1);

ksession.getFactHandle(p1);
```

插入事实的代码示例，并以 相等 模式返回事实句柄

```
ksession.insert(p1);

ksession.getFactHandle(p1);

// Alternate option:
ksession.getFactHandle(new Person("John", 45));
```

要设置事实相等模式，请使用以下选项之一：

- 将系统属性 drools.equalityBehavior 设置为 identity （默认）或 equality。

- 以编程方式创建 KIE 基础，设置相等模式：

```
KieServices ks = KieServices.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(EqualityBehaviorOption.EQUALITY);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

- 在 KIE 模块描述符文件(kmodule.xml)中为特定 Red Hat Process Automation Manager 项目设置相等模式：

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

第 83 章 在决策引擎中执行控制

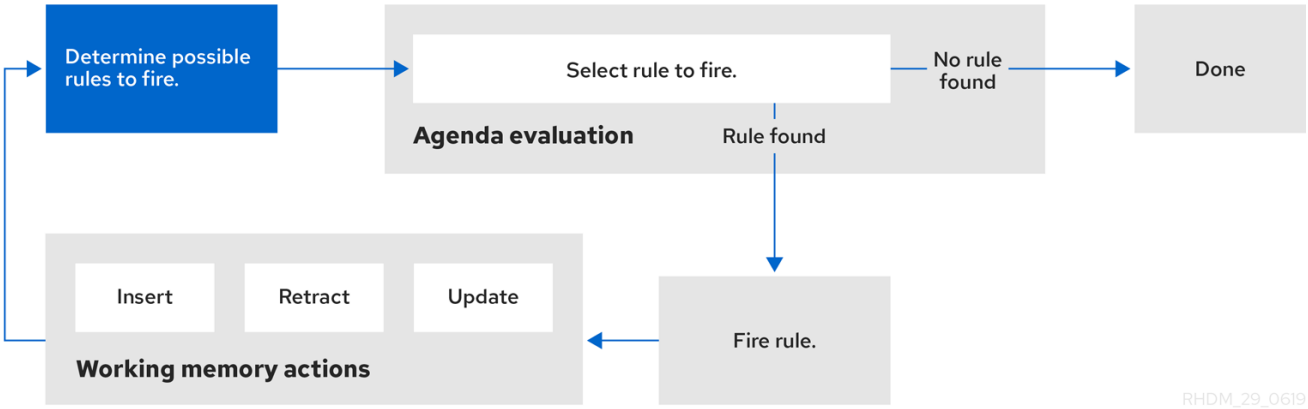
当新规则数据进入决策引擎的工作内存时，规则可能完全匹配且符合执行条件。单一工作内存操作可产生多个符合条件的规则执行。当某个规则完全匹配时，决定引擎创建激活实例，引用规则和匹配事实，并将激活添加到决策引擎日程表中。日程表使用冲突解决策略控制这些规则激活的执行顺序。

在 Java 应用程序第一个调用 fireAllRules（） 后，决策引擎循环在两个阶段重复执行：

- 日程评估.在此阶段，决策引擎选择可以执行的所有规则。如果没有可执行规则，则执行周期将结束。如果找到可执行规则，则决策引擎在时间表中注册激活，然后进入正常工作的内存操作阶段来执行规则导致操作。

- 操作内存操作。在此阶段，决策引擎针对之前在日程中注册的所有已激活规则执行规则后果操作（每个规则的一部分）。完成所有结果操作后，或者主 Java 应用程序进程调用 fireAllRules（） 后，决策引擎将返回到日程评估阶段，以重新评估规则。

图 83.1. 决定引擎中的双阶段执行过程



当日程上存在多个规则时，一个规则的执行可能会导致从日程表中删除另一条规则。要避免这种情况，您可以定义在决策引擎中如何以及规则执行的时间。定义规则执行顺序的一些常见方法是使用规则 salience、电缆组或激活组。

83.1. 规则的隔离

每个规则都有一个整数 sali ence 属性，它决定了执行的顺序。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。规则的默认 salience 值为零，但salience 可以是负数或正数。

例如，以下 DRL 规则按照所示顺序列在决策引擎堆栈中：

```
rule "RuleA"
salience 95
when
    $fact : MyFact( field1 == true )
then
    System.out.println("Rule2 : " + $fact);
    update($fact);
end

rule "RuleB"
salience 100
when
   $fact : MyFact( field1 == false )
then
   System.out.println("Rule1 : " + $fact);
   $fact.setField1(true);
   update($fact);
end
```

RuleB 规则是第二种规则，但它的值高于 **RuleA** 规则，因此首先执行。

83.2. 规则的日程组

日程安排小组是通过相同的日程安排规则属性，共同 开展的一系列 规则。安排按决策引擎日程表分组分组规则。在任何时候，只有一个组 专注于 执行某个规则，然后再在其他 schedule groups 中的规则之前。您可以利用 setFocus（） 邀请您获取此表格小组的关注。您还可以使用 auto-focus 属性定义规则，以便下一次激活规则时，会自动将重点分配给分配给该规则的整个日程表组。

每次在 Java 应用程序中执行 setFocus（） 调用时，决策引擎会将指定的日程表组添加到规则堆栈的顶部。默认日程小组 "MAIN" 包含不属于指定日程表组的所有规则，并且首先在堆栈中执行，除非另一个组有关注。

例如，以下 DRL 规则属于指定的索引组，它们按以下所示的顺序列在决策引擎堆栈中：

银行应用程序的 DRL 规则示例

```
rule "Increase balance for credits"
  agenda-group "calculation"
when
  ap : AccountPeriod()
  acc : Account( $accountNo : accountNo )
  CashFlow( type == CREDIT,
            accountNo == $accountNo,
            date >= ap.start && <= ap.end,
```

```
        $amount : amount )
then
  acc.balance  += $amount;
end
```

```
rule "Print balance for AccountPeriod"
  agenda-group "report"
when
  ap : AccountPeriod()
  acc : Account()
then
  System.out.println( acc.accountNo +
                      " : " + acc.balance );
end
```

在本例中，必须首先执行 "报告" 日程中的规则，并且 "计算" 日程表组中的规则必须始终执行第二个。然后可以执行其他日程组中的任何剩余规则。因此，在执行其他规则之前， "报告 "和"计算" 组必须接收执行该顺序的焦点：

设定日程组执行顺序的关注

```
Agenda agenda = ksession.getAgenda();
agenda.getAgendaGroup( "report" ).setFocus();
agenda.getAgendaGroup( "calculation" ).setFocus();
ksession.fireAllRules();
```

您还可以使用 clear（） 方法，在每一机会执行前取消属于给定日程表组生成的所有激活：

取消所有其他规则激活

```
ksession.getAgenda().getAgendaGroup( "Group A" ).clear();
```

83.3. 规则的激活组

分组是一组由同一 激活分组规则属性绑定到一起 的一组规则。在该组中，只能执行一条规则。在满足该组中的规则后，来自该激活组中的所有其他待处理规则执行都将从日程表中删除。

例如，以下 DRL 规则属于指定的激活组，按照所示的顺序列在决策引擎堆栈中：

银行 DRL 规则示例

```
rule "Print balance for AccountPeriod1"
  activation-group "report"
when
  ap : AccountPeriod1()
  acc : Account()
then
  System.out.println( acc.accountNo +
                      " : " + acc.balance );
end
```

```
rule "Print balance for AccountPeriod2"
  activation-group "report"
when
  ap : AccountPeriod2()
  acc : Account()
then
  System.out.println( acc.accountNo +
                      " : " + acc.balance );
end
```

在本例中，如果执行 "报告" 激活组中的第一条规则，则组中的第二条规则以及日程上的所有其他可执行规则均从日程表中删除。

83.4. 在决策引擎中规则执行模式和线程安全

决策引擎支持以下规则执行模式，它们决定了决策引擎如何执行规则：

- 被动模式 ：（默认）决策引擎在用户或应用程序明确调用 fireAllRules（） 时评估规则。决策引擎中的被动模式最适合需要直接控制规则评估和执行，或用于使用决策引擎中伪时钟实施的复杂事件处理(CEP)应用程序。

使用被动模式的决策引擎的 CEP 应用代码示例

```
KieSessionConfiguration config =
KieServices.Factory.get().newKieSessionConfiguration();
config.setOption( ClockTypeOption.get("pseudo") );
KieSession session = kbase.newKieSession( conf, null );
SessionPseudoClock clock = session.getSessionClock();

session.insert( tick1 );
session.fireAllRules();

clock.advanceTime(1, TimeUnit.SECONDS);
session.insert( tick2 );
session.fireAllRules();

clock.advanceTime(1, TimeUnit.SECONDS);
session.insert( tick3 );
session.fireAllRules();

session.dispose();
```

- Active 模式 ：如果用户或应用程序调用 fireUntilHalt（），则决策引擎以主动模式启动，并持续评估规则，直到用户或应用明确调用 halt（）。决策引擎中的主动模式最适合那些将规则评估和执行权限委托给决策引擎，或用于使用决策引擎中实时时钟实施的复杂事件处理(CEP)应用程序的应用程序。主动模式也是使用活跃查询的 CEP 应用程序的最佳选择。

带有主动模式下决策引擎的 CEP 应用代码示例

```
KieSessionConfiguration config =
KieServices.Factory.get().newKieSessionConfiguration();
config.setOption( ClockTypeOption.get("realtime") );
KieSession session = kbase.newKieSession( conf, null );

new Thread( new Runnable() {
  @Override
  public void run() {
     session.fireUntilHalt();
  }
} ).start();

session.insert( tick1 );

... Thread.sleep( 1000L ); ...
```

```
session.insert( tick2 );

... Thread.sleep( 1000L ); ...

session.insert( tick3 );

session.halt();
session.dispose();
```

此示例调用来自专用执行线程的 fireUntilHalt（），以防止当决策引擎继续评估规则时，当前线程被无限期阻止。专用线程还允许您在应用程序代码的后续阶段调用 halt（）。

虽然您应该避免使用 fireAllRules（） 和 fireUntilHalt（） 调用，特别是从不同的线程中，决策引擎可以安全地使用 thread-safety 逻辑和内部状态机处理这种情况。如果 fireAllRules（） 调用正在进行中，并且调用 fireUntilHalt（），则决策引擎将继续以被动模式运行，直到 fireAllRules（） 操作完成，然后以主动模式开始响应 fireUntilHalt（） 调用。但是，如果决策引擎在活跃的模式下运行，且调用 fireAllRules（） 调用，并且您调用 fireAllRules（） 调用将被忽略，则决策引擎将继续以活跃模式运行，直到调用 halt（）。

对于以活跃模式添加的线程安全，决策引擎支持 提交（） 方法，您可以在线程安全、atomic 操作中对 KIE 会话执行操作：

带有 submit（） 方法的示例应用程序代码，可在主动模式下执行 atomic 操作

```
KieSession session = ...;

new Thread( new Runnable() {
  @Override
  public void run() {
     session.fireUntilHalt();
  }
} ).start();

final FactHandle fh = session.insert( fact_a );

... Thread.sleep( 1000L ); ...

session.submit( new KieSession.AtomicAction() {
  @Override
  public void execute( KieSession kieSession ) {
    fact_a.setField("value");
    kieSession.update( fh, fact_a );
    kieSession.insert( fact_1 );
```

```
    kieSession.insert( fact_2 );
    kieSession.insert( fact_3 );
  }
} );

... Thread.sleep( 1000L ); ...

session.insert( fact_z );

session.halt();
session.dispose();
```

从客户端角度而言，线程安全和原子操作也很有帮助。例如，您可能需要在给定时间插入多个事实，但需要将决策引擎视为原子操作，并等待所有插入操作再次评估规则。

83.5. 在决策引擎中事实传播模式

决策引擎支持以下事实传播模式，决定决策引擎如何通过引擎网络插入事实，以准备规则执行：

- lazy:(Default)事实在规则执行时传播到批处理集合，而不是实时由用户或应用插入事实。因此，通过决策引擎传播事实的顺序可能与直接插入事实的顺序不同。
- 即时 ：事实会立即按照用户或应用程序插入的顺序立即传播。
- eager: 事实在批处理集合中传播（在批处理集合中），但在规则执行前。决策引擎将这种传播行为用于具有 no-loop 或 lock-on-active 属性的规则。

默认情况下，决策引擎中的 Phreak 规则算法使用 lazy fact propagation 来改进规则评估整体。然而，在一些情况下，这种 lazy Propagation 行为可能会改变某些可能需要立即或立即传播的某些规则执行的结果。

例如，以下规则使用带有 ? 前缀的指定查询以拉取或被动方式调用查询：

带有被动查询的规则示例

```
query Q (Integer i)
    String( this == i.toString() )
end

rule "Rule"
  when
    $i : Integer()
    ?Q( $i; )
  then
    System.out.println( $i );
end
```

在本例中，只有在满足查询在 Integer 前插入的 String 的字符串时才执行该规则，例如以下示例命令中：

应该触发规则执行的命令示例

```
KieSession ksession = ...
ksession.insert("1");
ksession.insert(1);
ksession.fireAllRules();
```

但是，由于 Phreak 中默认的 lazy Propagation 行为，决策引擎不会检测到本例中两个事实的插入序列，因此无论 String 和 Integer insertion 顺序如何，都执行此规则。在本例中，预期规则评估需要立即传播。

要改变决策引擎传播模式来实现预期的规则评估，您可以将 @Propagation(<type>) 标签添加到您的规则，并将 < type > 设置为 LAZY、IMMEDIATE 或 EAGER。

在同一示例规则中，即时传播注解允许仅在满足查询条件的 String 在 Integer 前插入，以如预期一样评估规则：

带有被动查询和指定传播模式的规则示例

```
query Q (Integer i)
    String( this == i.toString() )
end

rule "Rule" @Propagation(IMMEDIATE)
  when
    $i : Integer()
    ?Q( $i; )
  then
    System.out.println( $i );
end
```

83.6. 日程评估过滤器

决策引擎支持过滤器界面中的 AgendaFilter 对象，您可以在完成评估期间允许或拒绝对指定规则进行评估。作为 fireAllRules（） 调用的一部分，您可以指定一个日程表过滤器。

以下示例代码只允许以字符串 "Test" 结尾的规则，以评估和执行。所有其他规则均从决策引擎日程表中过滤。

日程过滤定义示例

```
ksession.fireAllRules( new RuleNameEndsWithAgendaFilter( "Test" ) );
```

第 84 章 决定引擎中的 PHREAK 规则算法

Red Hat Process Automation Manager 中的决策引擎使用 Phreak 算法来进行规则评估。Phreak 从 Rete 算法发展，包括面向对象系统以前版本的 Red Hat Process Automation Manager 中引入的增强 Rete algorithm ReteOO。总体而言，Phreak 比 Rete 和 ReteOO 更容易进行扩展，并在大型系统中速度快。

当 Rete 被视为"不良规则评估）和面向数据的情况下，Phreak 被视为 lazy（推迟的规则评估）和目标。Rete 算法在插入、更新和删除操作过程中执行许多操作，以便查找所有规则的部分匹配。规则匹配期间的 Rete 算法数量需要很多时间，在最终执行规则前需要很多时间，特别是在大型系统中。使用 Phreak 时，这个部分规则匹配会有意延迟，以便更有效地处理大量数据。

Phreak 算法为之前的 Rete 算法添加了以下一组改进：

- 上下文内存的三个层：节点、网段和规则内存类型
- 基于规则、基于网段和节点的连接
- lazy(delayed)规则评估
- 基于堆栈的评估会暂停和恢复
- 隔离规则评估
- set-oriented propagations

84.1. PHREAK 中的规则评估

当决策引擎启动时，所有规则均被视为与可触发规则的模式匹配数据。在这个阶段，决策引擎中的 Phreak 算法不会评估规则。insert、update 和 delete 操作会被放入排队，Phreak 使用了高流（根据规则最有可能执行）来计算并选择下一个评估规则。当为规则填充了所有必要的输入值时，该规则被视为与相关模式匹配数据相关联。然后，Phreak 会创建代表此规则的目标，并将目标置于优先级队列中，根据规则排序。只有评估了创建目标的规则，且其他潜在的规则评估会延迟。评估单个规则时，仍会通过分段过程来实现节点共享。

与面向 tuple 的 Rete 不同，Phreak propagation 是以集合为导向。对于正在评估的规则，决策引擎会访问第一个节点，并处理所有已排队插入、更新和删除操作。结果添加到集合中，集合会传播到子节点。在子节点中，处理所有已排队的插入、更新和删除操作，将结果添加到同一集合中。然后，集合会传播到下一个子节点，并重复相同的进程，直到它到达终端节点。此周期会创建一个批处理进程效果，可为某些规则结构提供性能优势。

规则的链接和取消链接通过基于网络分段的分层位掩码系统进行。在构建规则网络时，会为由同一组规则共享的规则网络节点创建网段。规则由片段的路径组成。如果某个规则不与任何其他规则共享任何节点，则会成为单个网段。

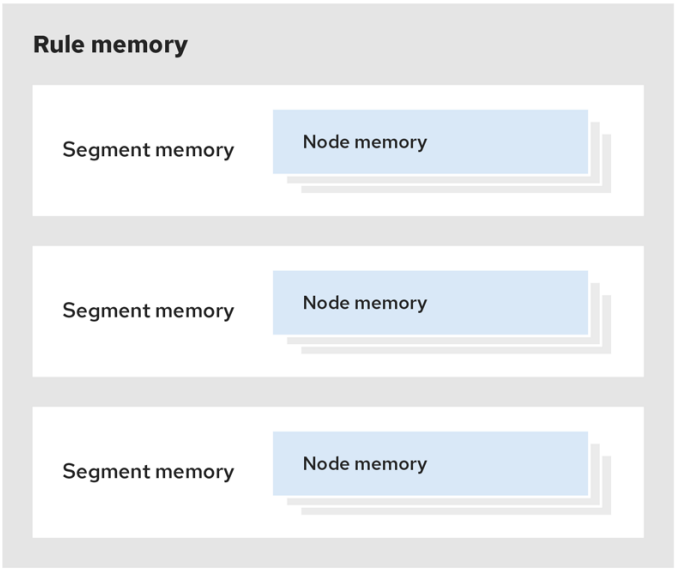
为网段中的每个节点分配一个位掩码偏移。根据这些要求，为规则路径中的每个片段分配另一个位掩码：

- 如果至少有一个输入节点存在，节点位将设置为 on 状态。
- 如果网段中的每个节点都被设置为 on 状态，则片段位也被设置为 on 状态。
- 如果有任何节点位设置为 off 状态，则该片段也会设置为 off 状态。
- 如果规则路径中的每个部分设置为 on 状态，则会被视为链接，并创建一个目标来调度评估规则。

相同的位掩码技术用于跟踪修改的节点、网段和规则。通过此跟踪功能，已经连接了从评估中未调度的规则（如果自其创建的评估目标）被修改过。因此，无法评估部分匹配规则。

在 Phreak 中可以进行规则评估的过程，因为与 Rete 的单个内存单位不同，Phreak 有三个带有节点、网段和规则内存类型的上下文内存层。这种分层可在评估规则的过程中更多上下文理解。

图 84.1. Phreak 三层内存系统

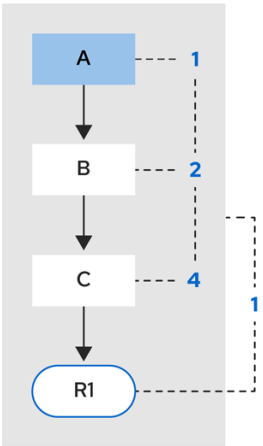


RHDM_29_0319

以下示例演示了如何在 Phreak 中对规则进行组织和评估。

示例 1： 具有三种模式的单一规则(R1)：A、B 和 C。规则形成单一片段，其位为 1、2 和 4。单个片段具有位偏移 1。

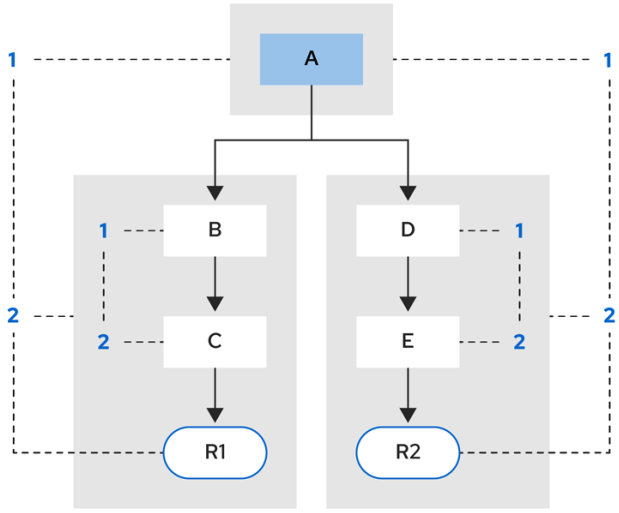
图 84.2. 示例 1：单一规则



RHDM_29_0319

示例 2： 添加了规则 R2 并共享模式 A。

图 84.3. 示例 2：使用模式共享的两个规则



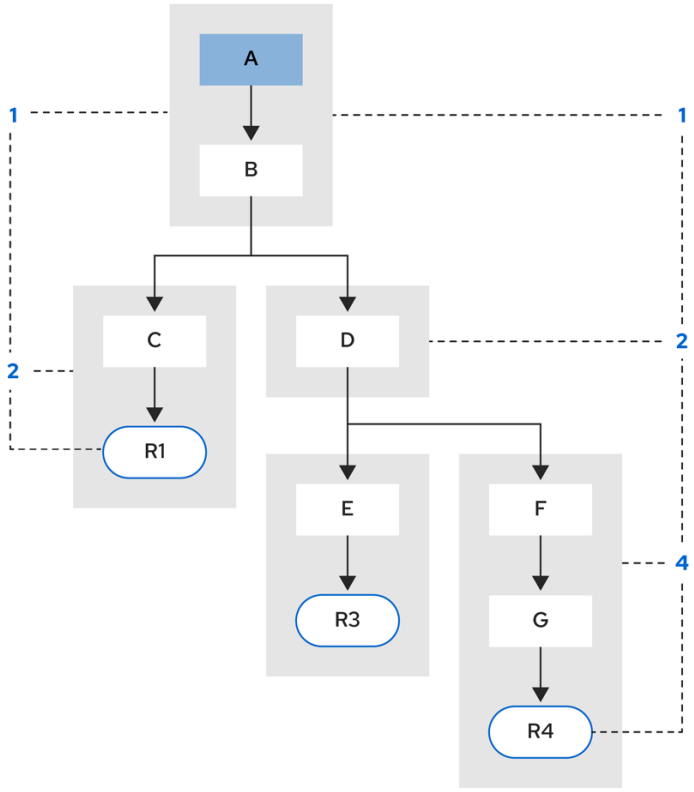
RHDM_29_0319

Pattern A 放置在自己的网段中，为每个规则产生两个片段。这两个片段组成了对应规则的路径。第一个部分由两个路径共享。当特征 A 被链接时，该片段将会被链接。然后，该片段会迭代该片段共享的每个路径，在上将 位 1 设置为。如果模式 B 和 C 稍后打开，则路径 R1 的第二个片段将链接，这会导致为 R1 打开位 2。当为 R1 打开位 1 并位 2 被打开后，该规则现已被链接，并创建了一个目标来计划以后评估和执行规则。

评估规则时，该片段支持共享匹配的结果。每个片段都有一个临时内存，用于队列所有插入、更新和删除该片段。评估 R1 时，规则进程模式 A，这会导致一组元组。该算法检测分段分割，为每个插入、更新和删除创建对等的元组，并在集合中添加它们到 R2 暂存内存。然后，这些元组将与任何现有的已暂存元组合并，并在最终评估 R2 时执行。

示例 3：添加了规则 R3 和 R4，并共享模式 A 和 B。

图 84.4. 示例 3：使用模式共享的三个规则

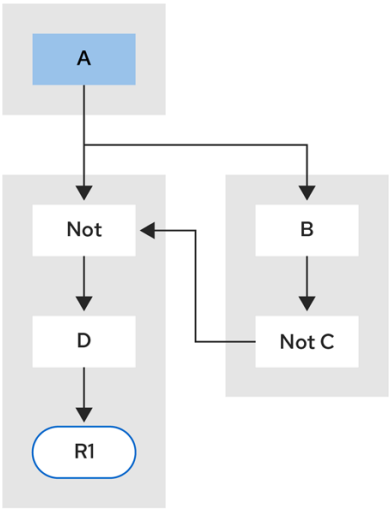


RHDM_29_0319

规则 R3 和 R4 有三个网段，R1 有两个网段。模式 A 和 B 由 R1、R3 和 R4 共享，而模式 D 则由 R3 和 R4 共享。

示例 4： 一个具有子网且没有模式共享的规则(R1)。

图 84.5. 示例 4：带有子网且没有模式共享的规则



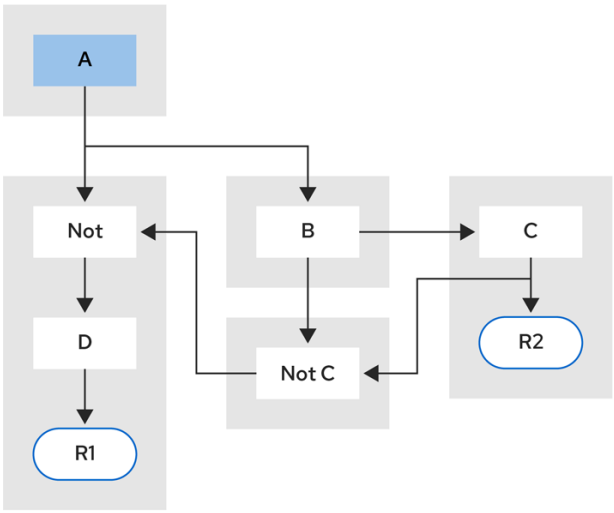
RHDM_29_0319

当 非 Exists 或 Accumulate 节点包含多个元素时，会形成子网。在本例中，元素 B not(C) 组成了子网。元素 not(C) 是一个不需要子网且在 Not 节点内合并的一个元素。该子网使用了专用网段。规则 R1

仍具有两个网段的路径，子网形成另一个内部路径。子网链接后，也会链接到外部网段。

示例 5：包含由规则 R2 共享的子网的组 R1。

图 84.6. 示例 5：两个规则，分别具有子网和模式共享



规则中的子网节点可以通过另一个没有子网的规则共享。这种共享会导致子网片段分成两个片段。

受限制的 Not nodes 和 Accumulate 节点永远不会取消链接一个网段，而且始终被视为启用了其位。

Phreak 评估算法基于堆栈，而非基于方法的。当 StackEntry 用于代表当前正在评估的节点时，可以随时暂停规则评估并恢复。

当规则评估到达子网时，会为外部路径网段和子网片段创建一个 StackEntry 对象。子网片段首先评估，当集合达到子网路径的末尾时，该片段将合并到该网段馈送节点的 staging 列表中。之前的 StackEntry 对象随后恢复，现在可以处理子网的结果。此过程具有添加的好处，特别是对于聚合的节点，对于聚合节点之前，所有工作都将以批处理形式完成。

相同的堆栈系统用于高效的向后兼容性。当规则评估到达查询节点时，评估将暂停，并将查询添加到堆栈中。然后，查询会被评估来生成结果集，该设置保存在 resumed StackEntry 对象的内存位置中，以提取并传播到子节点。如果查询本身被称为其他查询，该过程会重复，而当前查询已暂停，并为当前查询节点设置新的评估。

84.1.1. 带有转发和向后链的规则评估

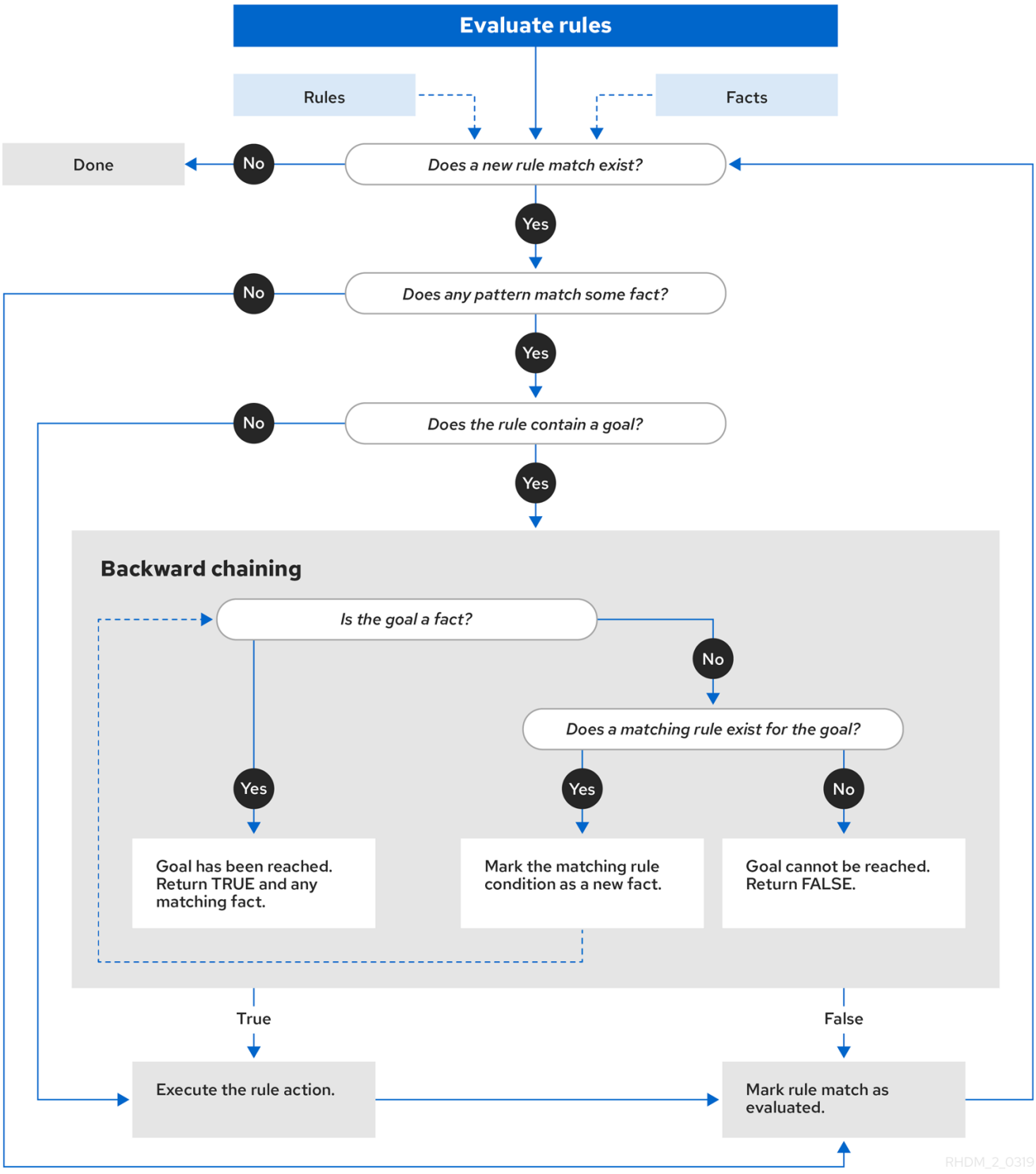
Red Hat Process Automation Manager 中的决策引擎是一个混合原因系统，它使用转发链和后向链

来评估规则。forward-chaining 规则系统是一个由数据驱动的系统，它从决策引擎的工作内存从事实开始，并对事实做出响应。当对象插入到工作内存时，因为更改是由日程表计划执行而变为 true 的任何规则条件。

相反，反向链接规则系统是一个由目标驱动的系统，从结论开始，决定引擎尝试满足，通常使用递归。如果系统无法达到结论或目标，它会搜索部分当前目标的子项。系统会继续这个过程，直到初始的结论是满足或者所有子语满意。

下图显示了如何使用转发链在逻辑流中的反向链接片段评估规则：

图 84.7. 使用转发和向后链的规则评估逻辑



84.2. 规则基础配置

Red Hat Process Automation Manager 包含一个 RuleBaseConfiguration.java 对象，您可以在决策引擎中配置异常处理器设置、多线程执行和连续模式。

对于规则基础配置选项，请从红帽客户门户网站下载 **Red Hat Process Automation Manager 7.13.5 Source Distribution ZIP 文件，并导航到 ~/rhpam-7.13.5-sources/src/drools-$VERSION/drools-**

core/src/main/java/org/drools/core/RuleBaseConfiguration.java.
https://access.redhat.com/jbossnetwork/restricted/listSoftware.html

以下规则基础配置选项可用于决策引擎：

drools.consequenceExceptionHandler

配置后，此系统属性定义管理规则后果所引发异常的类。您可以使用此属性为决策引擎中的规则评估指定自定义异常处理程序。

默认值： org.drools.core.runtime.rule.impl.DefaultConsequenceExceptionHandler

您可以使用以下选项之一指定自定义异常处理程序：

- 在系统属性中指定异常处理程序：

```
drools.consequenceExceptionHandler=org.drools.core.runtime.rule.impl.MyCustomConsequ
enceExceptionHandler
```

- 在以编程方式创建 KIE 基本时指定异常处理程序：

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(ConsequenceExceptionHandlerOption.get(MyCustomCons
equenceExceptionHandler.class));
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

drools.multithreadEvaluation

启用后，该系统属性可让决策引擎通过将 Phreak 规则网络划分为独立的分区，从而并行评估规则。您可以使用此属性提高特定规则基础的规则评估速度。

默认值： false

您可以使用以下选项之一启用多线程评估：

- 启用多线程评估系统属性：
  -

```
drools.multithreadEvaluation=true
```

- 以编程方式创建 KIE 基础时启用多线程评估：

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(MultithreadEvaluationOption.YES);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

警告

并行决策引擎目前不支持使用查询、等同于或日程表组的规则。如果 KIE 库中存在这些规则元素，编译器会发出一个警告，并自动切回到单线程评估。然而，在某些情况下，决定引擎可能无法检测不支持的规则元素和规则。例如，决定引擎可能无法检测规则何时依赖 DRL 文件中的规则排序授予隐式 salience 属性，从而导致因为不支持的 salience 属性导致评估不正确。

drools.sequential

启用后，这个系统属性可在决策引擎中启用连续模式。在连续模式中，决策引擎按照决策引擎日程表中列出的顺序来评估规则，而不考虑工作内存的变化。这意味着，决策引擎会忽略规则中的所有插入、修改 或更新 语句，并在单个序列中执行规则。因此，规则执行可能会以连续模式更快，但重要的更新可能不会应用到您的规则。如果您使用无状态 KIE 会话，且您不希望执行规则在日程中影响后续规则，则可以使用此属性。顺序模式只适用于无状态 KIE 会话。

默认值： **false**

您可以使用以下选项之一启用连续模式：

- 启用后续模式系统属性：

```
drools.sequential=true
```

- 以编程方式创建 KIE 基本时启用顺序模式：

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
```

```
kieBaseConf.setOption(SequentialOption.YES);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

- 在特定 Red Hat Process Automation Manager 项目的 KIE 模块描述符文件中启用连续模式(kmodule.xml)：

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" sequential="true" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

84.3. PHREAK 中的顺序模式

顺序模式是决策引擎支持的高级规则基础配置，由 Phreak 支持，它可让决策引擎按决策引擎日中列出的顺序来评估规则，而无需考虑工作内存的变化。在连续模式中，决策引擎会忽略规则中的任意 插入、修改 或更新 语句，并在单个序列中执行规则。因此，规则执行可能会以连续模式更快，但重要的更新可能不会应用到您的规则。

顺序模式仅适用于无状态 KIE 会话，因为有状态 KIE 会话固地使用之前调用的 KIE 会话中的数据。如果您使用无状态 KIE 会话，并且希望执行规则以在日程表中影响后续规则，则不要启用连续模式。在决策引擎中默认禁用顺序模式。

要启用连续模式，请使用以下选项之一：

- 将系统属性 drools.sequential 设置为 true。
- 以编程方式创建 KIE 基本时启用顺序模式：

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(SequentialOption.YES);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

- 在特定 Red Hat Process Automation Manager 项目的 KIE 模块描述符文件中启用连续模式(kmodule.xml)：

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" sequential="true" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

要配置连续**模式以使用**动态**索引，**请**使用以下**选项**之一：**

- **将系**统**属性 drools.sequential.agenda** 设**置**为 动态**。**
- **以**编**程方式**创**建 KIE 基**础时设**定**连续**日程表：**

```
KieServices ks = KieServices.Factory.get();
KieBaseConfiguration kieBaseConf = ks.newKieBaseConfiguration();
kieBaseConf.setOption(SequentialAgendaOption.DYNAMIC);
KieBase kieBase = kieContainer.newKieBase(kieBaseConf);
```

当您启**用**连续**模式**时，**决定引擎使用以下方法**评**估**规则：**

1. **规**则**按照 salience 和在**规**则集中的位置排序。**
2. **每个可能**规**则匹配的元素会被**创**建。元素位置表示**执**行**顺**序。**
3. **节**点**内存被禁用，但 right-input** 对**象内存除外。**
4. **left-input 适配器**节**点**传**播断开**连**接的，在 Command** 对**象中引用**带**有**该节**点的**对**象。Command** 对**象添加到工作内存中的列表中，以便稍后**执**行。**
5. **所有**对**象都会被断言，然后**检查 **并**执**行命令**对**象列表。**
6. **执**行**列表的所有匹配都将添加到元素中，具体取决于**规**则的序列号。**
7.

包含匹配项的元素按顺序执行。如果您设置了最多规则执行数，则决策引擎不会激活执行日程表中的相应数量。

在连续模式中，LeftInputAdapterNode 节点创建一个 Command 对象，并将其添加到决策引擎的工作内存中。此命令对象包含对 LeftInputAdapterNode 节点和传播对象的引用。这些引用在插入时停止任何左侧输入传播，因此正确的输入传播不需要尝试加入左侧输入。引用还避免了对 left-input 内存的需求。

所有节点都关闭其内存，包括 left-input tuple 内存，但不包括正确的输入对象内存。完成所有断言，并填充了所有对象的正确输入内存后，决策引擎会迭代 LeftInputAdatperNode Command 对象列表。对象传播网络，尝试加入正确的输入对象，但它们不会在左边输入中保留。

安排优先队列的日程安排方式由每条规则的一个元素替代。RuleTerminalNode 节点的序列号表示放置匹配的元素。在 所有命令 对象完成后，会检查元素并执行现有的匹配项。为提高性能，则会保留元素中第一个和最后填充的单元。

创建该网络时，每个 RuleTerminalNode 节点都会根据其 salience 编号及其添加到网络中的顺序接收序列号。

right-input 节点 Imories 通常是用于删除快速对象的哈希映射。因为不支持删除对象，所以当对象的值没有索引时，Phreak 使用对象列表。对于大量对象，索引的散列映射可以提高性能。如果对象只有一个实例，Phreak 使用对象列表而不是索引。

第 85 章 复杂的事件处理(CEP)

在 Red Hat Process Automation Manager 中，事件是应用程序域中大量更改状态的记录。根据域建模，状态更改可能由一个事件、多个原子事件或关联事件的层次结构表示。从复杂的事件处理(CEP)的角度来看，事件是某一时发生在特定时间点的事实或对象，而商业规则是如何响应该事实或对象的数据的定义。例如，在库存代理应用程序中，安全价格的变化、从销售者变为买方更改，或者帐户持有者的平衡的变化被视为事件，因为更改在给定时间处于应用程序域的状态发生。

红帽流程自动化管理器中的决策引擎使用复杂的事件处理(CEP)在事件集合内检测和处理多个事件，从而发现事件之间存在的关系，以及从事件及其关系中推断新数据的关系。

CEP 用例与业务规则用例共享多个要求和目标。

从业务角度来看，基于事件所触发的情况发生，商业规则定义通常根据事件触发的情况进行定义。在以下示例中，事件形成了业务规则的基础：

- 在算法交易应用程序中，如果安全性价格已超过公开价格，则规则将执行一个操作。价格增长由库存交易应用程序上的事件表示。
- 在监控应用程序中，如果服务器机房中的温度增加 X 程度，规则将执行操作。读取的传感器由事件表示。

从技术角度来说，商业规则评估和 CEP 有以下关键相似之处：

- 业务规则评估和 CEP 都需要与企业基础架构和应用程序的无缝集成。这在生命周期管理、审计和安全性方面尤其重要。
- 业务规则评估和 CEP 都有功能要求，如模式匹配和非功能性要求，如响应时间限制和查询规则说明。

CEP 情境有以下关键特征：

- 情景通常处理大量事件，但只会处理少量事件。

- 事件通常是不可变的，代表状态更改的记录。
- 针对事件运行的规则和查询，必须响应检测到的事件模式。
- 相关事件通常具有强大的时序关系。
- 单个事件没有优先级。CEP 系统优先排序相关事件的模式以及它们之间的关系。
- 事件通常需要组成并聚合。

鉴于这些常见的 CEP 方案特征，红帽流程自动化管理器中的 CEP 系统支持以下功能和功能来优化事件处理：

- 具有正确语义的事件处理
- 事件检测、关联、聚合和组成
- 事件源处理
- 临时限制来模拟事件之间的时序关系
- 大量事件的滑动窗口
- 会话范围内的统一时钟
- CEP 用例所需的事件卷
- 被动规则

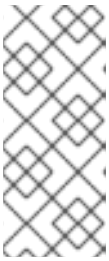
- 在决策引擎(pipeline)中事件输入的适配器

85.1. 复杂事件处理中的事件

在 Red Hat Process Automation Manager 中，事件是应用程序域中大量更改状态的记录。根据域建模，状态更改可能由一个事件、多个原子事件或关联事件的层次结构表示。从复杂的事件处理(CEP)的角度来看，事件是某一时发生在特定时间点的事实或对象，而商业规则是如何响应该事实或对象的数据的定义。例如，在库存代理应用程序中，安全价格的变化、从销售者变为买方更改，或者帐户持有者的平衡的变化被视为事件，因为更改在给定时间处于应用程序域的状态发生。

事件有以下关键特征：

- 是不可变的： 事件是过去发生的、无法更改的改变记录。



注意

决策引擎不会在代表事件的 Java 对象上实施不可变性。这个行为使事件数据增强成为可能。您的应用程序应该能够填充未填充的事件属性，这些属性供决策引擎用于增强事件与推断的数据。但是，您不应该更改已经填充的事件属性。

- 具有强时序限制： 涉及事件的规则通常需要指示在不同时间点上发生的多个事件。
- 具有受管生命周期：由于事件不可变且具有临时的限制，所以它们通常只与指定周期相关联。这意味着决策引擎可以自动管理事件的生命周期。
- 可以使用滑动窗口： 定义一个带有事件的滑块窗口或长度。滑动时间窗是可以处理事件的指定时间段。滑动长度窗口是可以处理的指定数量事件。

85.2. 将事实声明为事件

您可以将事实声明为 Java 类或 DRL 规则文件中的事件，以便决策引擎在复杂事件处理期间将事实作为事件处理。您可以将事实声明为基于间隔的事件或时间点事件。基于间隔的事件的时间，并在决策引擎的工作内存中保留一段时间，直到它们的持续时间可用为止。点内事件没有持续时间，基本上是基于 interval 的事件，且持续为零。

流程

对于 Java 类或 DRL 规则文件中的相关事实类型，请输入 @role（事件） 元数据标签和参数。@role 元数据标签接受以下两个值：

- 事实:(Default)Declas 类型作为常规事实
- 事件: 拒绝类型作为事件

例如，以下片段声明 stock broker 应用中的 StockPoint 事实类型必须作为事件进行处理：

将事实类型声明为事件

```
import some.package.StockPoint

declare StockPoint
  @role( event )
end
```

如果 StockPoint 是 DRL 规则文件中声明的事实类型，而不是在预先存在的类中声明事件，您可以在应用程序代码中声明事件：

声明事实类型 in-line，并将它分配到事件角色

```
declare StockPoint
  @role( event )

  datetime : java.util.Date
  symbol : String
  price : double
end
```

85.3. 事件的元数据标签

决策引擎将以下元数据标签用于插入到决策引擎工作内存中的事件。您可以根据需要更改 Java 类或 DRL 规则文件中的默认元数据标签值。

注意

本节中的示例参考 VoiceCall 类假设示例应用程序域模型包括以下类详情：

示例电信域模型中的 VoiceCall 事实类

```
public class VoiceCall {
  private String  originNumber;
  private String  destinationNumber;
  private Date    callDateTime;
  private long    callDuration;  // in milliseconds

  // Constructors, getters, and setters
}
```

@role

此标签决定，给定事实类型是作为常规事实处理，还是在复杂事件处理期间在决策引擎中的事件进行处理。

default 参数： fact

支持的参数： 事实、事件

```
@role( fact | event )
```

示例： Declare VoiceCall 作为事件类型

```
declare VoiceCall
  @role( event )
end
```

@timestamp

此标签会自动分配给决策引擎中的每个事件。默认情况下，会话时钟提供的时间，并在事件插入决策引擎的工作内存中时分配给事件。您可以指定自定义时间戳属性，而不是会话时钟添加的默认时间戳。

default 参数：决策引擎会话时钟添加的时间

支持的参数： Session clock time 或 custom time stamp 属性

```
@timestamp( <attributeName> )
```

示例： Declare VoiceCall timestamp 属性

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
end
```

@duration

此标签决定决策引擎中事件的持续时间。事件可以是基于间隔的事件或时间点事件。基于间隔的事件的时间，并在决策引擎的工作内存中保留一段时间，直到它们的持续时间可用为止。点内事件没有持续时间，基本上是基于 interval 的事件，且持续为零。默认情况下，决策引擎中的每个事件都会有零持续时间。您可以指定自定义持续时间属性，而不是默认值。

默认参数：Null（零）

支持的参数： 自定义持续时间属性

```
@duration( <attributeName> )
```

示例： Declare VoiceCall duration 属性

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
  @duration( callDuration )
end
```

@expires

此标签决定事件在决策引擎工作内存过期前的时间持续时间。默认情况下，事件过期，事件不再匹配并激活任何当前规则。您可以定义事件应过期的时间。此标签定义还覆盖从 KIE 基础中的时序限制和滑动窗口计算的隐式到期偏移。只有在决策引擎以流模式运行时，该标签才可用。

默认参数：Null（事件后的事件过期，不再匹配并激活规则）

支持的参数：自定义 timeOffset 属性，格式为 [#d][#h][#m][#s][[ms]]

```
@expires( <timeOffset> )
```

示例：对 VoiceCall 事件进行禁止过期偏移

```
declare VoiceCall
  @role( event )
  @timestamp( callDateTime )
  @duration( callDuration )
  @expires( 1h35m )
end
```

85.4. 决策引擎中的事件处理模式

决策引擎以云模式或流模式运行。在云模式中，决策引擎将事实处理为事实，无时序限制、独立于时间。在流模式中，决策引擎实时或近乎实时限制将事实作为事件处理为事件。流模式使用同步在 Red Hat Process Automation Manager 中进行事件处理。

云模式

云模式是决策引擎的默认操作模式。在云模式中，决策引擎将事件视为无顺序的云。事件仍然存在时间戳，但以云模式运行的决策引擎无法从时间戳中提取，因为云模式会忽略目前的时间。这个模式使用规则约束来查找匹配的元组来激活和执行规则。

云模式不会对事实施加任何额外要求。但是，由于此模式中的决策引擎没有概念，因此无法使用时序功能，如滑动窗口或自动生命周期管理。在云模式中，不再需要时，必须明确指定事件。

在云模式中不会实施以下要求：

- *没有时钟同步，因为决策引擎没有时间*
- *没有事件排序，因为决策引擎将事件处理为无顺序的云，因此决策引擎与规则匹配。*

您可以通过在相关配置文件中设置系统属性或使用 Java 客户端 API 来指定云模式：

使用系统属性设置云模式

```
drools.eventProcessingMode=cloud
```

使用 Java 客户端 API 设置云模式

```
import org.kie.api.conf.EventProcessingOption;
import org.kie.api.KieBaseConfiguration;
import org.kie.api.KieServices.Factory;

KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();

config.setOption(EventProcessingOption.CLOUD);
```

您还可以使用特定 Red Hat Process Automation Manager 项目的 KIE 模块描述符文件 (kmodule.xml)中的 eventProcessingMode="<mode>" KIE base 属性来指定云模式：

使用项目 kmodule.xml 文件设置云模式

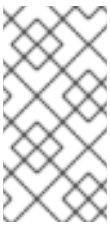
```
<kmodule>
  ...
  <kbase name="KBase2" default="false" eventProcessingMode="cloud"
packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1">
    ...
  </kbase>
  ...
</kmodule>
```

流模式

流模式使决策引擎能够按照时间处理事件，并在插入到决策引擎中实时处理。在流模式中，决策引擎同步事件流（因此，在不同流中的事件可以按照时间或长度的处理）实施分片，并启用自动生命周期管理。

以下要求适用于流模式：

- 每个流中的事件必须按时间排序。
- 必须存在会话时钟才能同步事件流。



注意

您的应用程序不需要强制在流间排序事件，而是使用没有同步的事件流可能会导致意外的结果。

您可以通过在相关配置文件中设置系统属性或使用 Java 客户端 API 来指定流模式：

使用系统属性设置流模式

```
drools.eventProcessingMode=stream
```

使用 Java 客户端 API 设置流模式

```
import org.kie.api.conf.EventProcessingOption;
import org.kie.api.KieBaseConfiguration;
import org.kie.api.KieServices.Factory;

KieBaseConfiguration config = KieServices.Factory.get().newKieBaseConfiguration();

config.setOption(EventProcessingOption.STREAM);
```

您还可以使用特定 Red Hat Process Automation Manager 项目的 KIE 模块描述符文件 (kmodule.xml)中的 eventProcessingMode="<mode>" KIE base 属性来指定流模式：

使用项目 kmodule.xml 文件设置流模式

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

85.4.1. 决策引擎流模式中的负模式

负模式是不满足的条件的模式。例如，如果检测到一个触发器且未激活 sprinkler，则以下 DRL 规则激活触发警报：

使用负模式触发警报规则

```
rule "Sound the alarm"
when
  $f : FireDetected()
  not(SprinklerActivated())
then
  // Sound the alarm.
end
```

在云模式中，决策引擎会预先识别所有事实（常规事实和事件），并立即评估负模式。在流模式中，决策引擎可以支持事实上的临时限制，以便在激活规则前等待设定时间。

流模式中的相同示例规则照常激活 fire 警报，但应用 10 秒的延迟。

触发警报规则，具有负模式和时间延迟（仅限流模式）

```
rule "Sound the alarm"
when
  $f : FireDetected()
  not(SprinklerActivated(this after[0s,10s] $f))
then
  // Sound the alarm.
end
```

以下修改的 fire 警报规则要求每 10 秒发生一 个 Heartbeat 事件。如果没有发生预期的事件，则会执行该规则。此规则在第一个模式和负模式中使用相同的对象类型。负模式具有临时限制，在执行前等待 0 到 10 秒，并排除绑定到 $h 的 Heartbeat 事件，以便可以执行该规则。要执行规则，必须明确排除绑定事件 $h，因为临时约束 [0s, ...] 不严格排除该事件被重新匹配。

fire 警报规则排除了负模式（仅流模式）中的绑定事件。

```
rule "Sound the alarm"
when
  $h: Heartbeat() from entry-point "MonitoringStream"
```

```
  not(Heartbeat(this != $h, this after[0s,10s] $h) from entry-point "MonitoringStream")
then
  // Sound the alarm.
end
```

85.5. 为事实类型更改设置和监听程序

默认情况下，决策引擎不会在每次触发规则时重新评估所有事实类型的事实模式，而是仅响应给定模式内受限制或绑定的属性。例如，如果规则调用 modify（） 作为规则操作的一部分，但该操作不会在 KIE 基础中生成新数据，则决策引擎不会自动重新评估所有事实模式，因为没有修改数据。这个属性 reactivity 行为可防止 KIE 库中不需要的递归，并导致更有效的规则评估。这个行为还意味着您不需要总是使用 no-loop rule 属性来避免无限重复。

您可以使用以下 KnowledgeBuilderConfiguration 选项修改或禁用此属性重新活动行为，然后根据需要使用 Java 类或 DRL 文件中的属性更改设置来微调属性重新活动：

- ALWAYS:（默认）所有类型都是属性 reactive，但您可以使用 @classReactive 属性-change 设置禁用特定类型的属性重新活动。
- ALLOWED ：无类型是属性 reactive，但您可以使用 @propertyReactive 属性-change 设置为特定类型启用属性重新活动。
- DISABLED: No type 是属性 reactive。所有属性更改监听程序都会被忽略。

在 KnowledgeBuilderConfiguration 中属性重新活动设置示例

```
KnowledgeBuilderConfiguration config =
KnowledgeBuilderFactory.newKnowledgeBuilderConfiguration();
config.setOption(PropertySpecificOption.ALLOWED);
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder(config);
```

或者，您可以在 Red Hat Process Automation Manager 发行版的 standalone.xml 文件中更新 drools.propertySpecific 系统属性：

在系统属性中设置的属性重新活动示例

```
<system-properties>
 ...
 <property name="drools.propertySpecific" value="ALLOWED"/>
 ...
</system-properties>
```

决策引擎支持以下属性更改设置和监听程序，用于事实类或声明的 DRL 事实类型：

@classReactive

如果在决策引擎中将属性重新活动设置为 ALWAYS （所有类型都是 reactive），则此标签会禁用特定 Java 类的默认属性重新活动行为或声明的 DRL 事实类型。如果您希望决策引擎在每次触发规则时重新评估指定事实类型的所有事实模式，则可使用此标签，而不是仅响应给定模式内受约束或绑定的属性。

示例： 禁用 DRL 类型声明中的默认属性重新活动

```
declare Person
  @classReactive
    firstName : String
    lastName : String
end
```

示例：禁用 Java 类中的默认属性重新活动

```
@classReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@propertyReactive

如果在决策引擎中将属性重新活动设置为 **ALLOWED**（没有属性重新活跃），则此标签为特定 **Java** 类启用属性重新活动，或者为声明的 **DRL** 事实类型启用属性重新活动。如果您希望决策引擎仅响应指定事实类型内受限制或绑定的属性，则可使用此标签，而不是在每次触发规则时重新评估所有事实模式。

示例：在 DRL 类型声明中启用属性重新活动（在全局禁用重新活动时）

```
declare Person
  @propertyReactive
    firstName : String
    lastName : String
end
```

示例：在 Java 类中启用属性重新活动（当全局禁用重新活动时）

```
@propertyReactive
public static class Person {
    private String firstName;
    private String lastName;
}
```

@watch

此标签为您在 **DRL** 规则中以实际模式指定的附加属性启用属性重新活动。只有在决策引擎中将属性重新活动设置为 **ALWAYS** 时，或将属性重新活动设置为 **ALLOWED** 且相关事实类型使用 **@propertyReactive** 标签时，才支持该标签。您可以在 **DRL** 规则中使用此标签在事实属性重新活动逻辑中添加或排除特定属性。

默认参数： None

支持的参数：Property name, *（全部）、!（不是）、!*（无属性）

```
<factPattern> @watch ( <property> )
```

示例：在事实模式下启用或禁用属性重新活动

```
// Listens for changes in both `firstName` (inferred) and `lastName`:
Person(firstName == $expectedFirstName) @watch( lastName )

// Listens for changes in all properties of the `Person` fact:
Person(firstName == $expectedFirstName) @watch( * )

// Listens for changes in `lastName` and explicitly excludes changes in `firstName`:
Person(firstName == $expectedFirstName) @watch( lastName, !firstName )

// Listens for changes in all properties of the `Person` fact except `age`:
Person(firstName == $expectedFirstName) @watch( *, !age )

// Excludes changes in all properties of the `Person` fact (equivalent to using `@classReactivity`
tag):
Person(firstName == $expectedFirstName) @watch( !* )
```

如果您在使用 @classReactive 标签的事实类型中使用 @watch 标签（禁用属性重新活动），或在决策引擎中将属性重新活动设置为 ALLOWED，则决策引擎将 @watch 标签用于属性。如果您在监听器注解中重复属性（如 @watch(firstName, ! firstName)）时，也会发生编译错误。

@propertyChangeSupport

对于实现对 **JavaBeans** 规格中定义的属性更改的事实，此标签使决策引擎能够监控事实属性中的更改。

示例： Declare 属性更改在 JavaBeans 对象中支持

```
declare Person
    @propertyChangeSupport
end
```

85.6. 事件时序算子

在流模式中，决策引擎支持以下时序运算符，这些事件插入到决策引擎的工作内存中。您可以使用这

些运算符来定义您在 Java 类或 DRL 规则文件中声明的事件行为的时序原因。当决策引擎以云模式运行时，不支持临时运算符。

- *之后*
- *before*
- *coincides*
- *期间*
- *includes*
- *完成*
- *完成于*
- *meets*
- *达到的*
- *重叠*
- *重叠，*
- *Starting*
- *启动者*

之后

此 operator 指定在关联事件后当前事件是否发生。此运算符也可以定义在相关事件后面可以达到相关事件的时间，或者当前事件可在关联事件之后达到限额的时间范围。

例如，如果 $eventA 在 3 分钟和 30 秒之间启动，则以下模式匹配，在 $eventB 完成后为 4 分钟。如果 $eventA 早于 3 分钟，且 30 秒在 $eventB 完成后或超过 4 分钟后启动，则模式不会被匹配。

```
$eventA : EventA(this after[3m30s, 4m] $eventB)
```

您还可以以以下方式表达此 operator：

```
3m30s <= $eventA.startTimestamp - $eventB.endTimeStamp <= 4m
```

after operator 支持最多两个参数值：

- 如果定义了两个值，则间隔在第一个值（示例中为 3 分钟和 30 秒）开始，并以第二个值结束（示例中为 4 分钟）。
- 如果只定义一个值，则间隔在提供的值启动，并无限期地运行，且不会出现结束时间。
- 如果没有定义值，则间隔以 1 毫秒开始，且无限期地运行，且没有结束时间。

after operator 还支持负时间范围：

```
$eventA : EventA(this after[-3m30s, -2m] $eventB)
```

如果第一个值大于第二个值，则决策引擎会自动逆向。例如，决策引擎以相同的方式解释以下两种模式：

```
$eventA : EventA(this after[-3m30s, -2m] $eventB)
$eventA : EventA(this after[-2m, -3m30s] $eventB)
```

before

此运算符指定当前事件是否在关联事件之前发生。此运算符也可以定义在关联事件之前当前事件的时长，或当前事件在关联事件之前可以达到无限的时间范围。

例如，如果 $eventA 在 3 分钟到 30 秒到 4 分钟之间完成，则以下模式匹配，在 $eventB 开始前 4 分钟。如果 $eventA 在前 3 分钟和 30 秒前为 $eventB 开始前，或超过 4 分钟，则在 $eventB 开始前，该模式不会被匹配。

```
$eventA : EventA(this before[3m30s, 4m] $eventB)
```

您还可以以以下方式表达此 operator：

```
3m30s <= $eventB.startTimestamp - $eventA.endTimeStamp <= 4m
```

before operator 支持最多两个参数值：

- 如果定义了两个值，则间隔在第一个值（示例中为 3 分钟和 30 秒）开始，并以第二个值结束（示例中为 4 分钟）。
- 如果只定义一个值，则间隔在提供的值启动，并无限期地运行，且不会出现结束时间。
- 如果没有定义值，则间隔以 1 毫秒开始，且无限期地运行，且没有结束时间。

before operator 还支持负时间范围：

```
$eventA : EventA(this before[-3m30s, -2m] $eventB)
```

如果第一个值大于第二个值，则决策引擎会自动逆向。例如，决策引擎以相同的方式解释以下两种模式：

```
$eventA : EventA(this before[-3m30s, -2m] $eventB)
$eventA : EventA(this before[-2m, -3m30s] $eventB)
```

coincides

此 operator 指定两个事件是否同时发生，且是相同的开始和结束时间。

例如，如果开始和结束时间戳为 $eventA 和 $eventB，则以下模式匹配：

```
$eventA : EventA(this coincides $eventB)
```

coincides operator 支持事件开始和结束时间之间的两个参数值（如果它们不相同）：

- **如果只给出一个参数，则使用参数 来设置这两个事件的开始和结束时间的阈值。**
- **如果给出两个参数，则第一个参数用作开始时间的阈值，第二个参数被用作结束时间的阈值。**

以下模式使用 start 和结束时间阈值：

```
$eventA : EventA(this coincides[15s, 10s] $eventB)
```

如果满足以下条件，则特征匹配：

```
abs($eventA.startTimestamp - $eventB.startTimestamp) <= 15s
&&
abs($eventA.endTimestamp - $eventB.endTimestamp) <= 10s
```

警告

决策引擎不支持 coincides operator 的负间隔。如果您使用负间隔，则决策引擎会生成错误。

期间

此 operator 指定当前事件是否在关联事件启动和结束的时间范围内发生。当前事件必须在关联事件开始后启动，且必须在关联事件结束前结束。（通过 coincides 运算符，开始和结束时间相同或几乎相同。）

例如，如果 $eventA 在 $eventB 开始并且在 $eventB 结束前结束，则以下模式匹配：

```
$eventA : EventA(this during $eventB)
```

您还可以以以下方式表达此 operator：

```
$eventB.startTimestamp < $eventA.startTimestamp <= $eventA.endTimestamp <
$eventB.endTimestamp
```

在 Operator 中，支持一个、两个或四个可选参数：

- 如果定义一个值，这个值是两个事件的开始时间和两个事件结束时间之间的最大距离。
- 如果定义了两个值，则这些值是一个阈值，当前事件开始时间和结束时间必须与相关事件启动和结束时间相关。

  例如，如果值为 5 和 10s，当前事件必须在关联事件开始后 5 到 10 秒之间启动，且必须在关联事件结束前 5 到 10 秒之间结束。
- 如果定义了四个值，则第一个值和第二个值是事件开始时间之间的最小和最大距离，第三值和第四个值则是两个事件结束时间之间的最小和最大距离。

includes

此运算符指定关联事件是否在发生当前事件时的时间段内。相关事件必须在当前事件开始后启动，且必须在当前事件结束前结束。（此运算符的行为是 Operator 行为的反向情况。）

例如，如果 $eventB 在 $eventA 启动后启动，且在 $eventA 结束前结束，则以下模式匹配：

```
$eventA : EventA(this includes $eventB)
```

您还可以以以下方式表达此 operator：

```
$eventA.startTimestamp < $eventB.startTimestamp <= $eventB.endTimestamp <
$eventA.endTimestamp
```

includes operator 支持一个、两个或四个可选参数：

- 如果定义一个值，这个值是两个事件的开始时间和两个事件结束时间之间的最大距离。

- 如果定义了两个值，则这些值是一个阈值，其中关联了事件开始时间和结束时间必须与当前事件开始和结束时间相关。

  例如，如果值为 5 和 10s，相关事件必须在当前事件开始后 5 到 10 秒之间启动，且必须在当前事件结束前 5 到 10 秒之间结束。

- 如果定义了四个值，则第一个值和第二个值是事件开始时间之间的最小和最大距离，第三值和第四个值则是两个事件结束时间之间的最小和最大距离。

完成

此运算符指定当前事件在关联事件后是否启动，但这两个事件同时结束。

例如，如果 $eventA 在 $eventB 启动后启动，则以下模式匹配，并同时在 $eventB 结束后结束：

```
$eventA : EventA(this finishes $eventB)
```

您还可以以以下方式表达此 operator：

```
$eventB.startTimestamp < $eventA.startTimestamp
&&
$eventA.endTimestamp == $eventB.endTimestamp
```

完成 Operator 支持一个可选参数，用于设置两个事件结束时间之间的最长时间：

```
$eventA : EventA(this finishes[5s] $eventB)
```

如果满足这些条件，这个特征匹配：

```
$eventB.startTimestamp < $eventA.startTimestamp
&&
abs($eventA.endTimestamp - $eventB.endTimestamp) <= 5s
```

警告

决策引擎不支持 完成 Operator 的负间隔。如果您使用负间隔，则决策引擎会生成错误。

完成于

此运算符指定关联事件是否在当前事件后启动，但两个事件同时结束。（此运算符的行为与 完成 Operator 的行为相反。）

例如，如果 $eventB 在 $eventA 启动后启动，则以下模式匹配，并同时在 $eventA 结束后结束：

```
$eventA : EventA(this finishedby $eventB)
```

您还可以以以下方式表达此 operator：

```
$eventA.startTimestamp < $eventB.startTimestamp
&&
$eventA.endTimestamp == $eventB.endTimestamp
```

Operator 完成 支持一个可选参数，用于设置两个事件结束时间之间的最长时间：

```
$eventA : EventA(this finishedby[5s] $eventB)
```

如果满足这些条件，这个特征匹配：

```
$eventA.startTimestamp < $eventB.startTimestamp
&&
abs($eventA.endTimestamp - $eventB.endTimestamp) <= 5s
```

警告

决策引擎不支持 operator 完成 的负间隔。如果您使用负间隔，则决策引擎会生成错误。

meets

此 operator 指定关联事件启动时当前事件是否同时结束。

例如，如果 $eventA 在 $eventB 启动时，以下模式匹配：

```
$eventA : EventA(this meets $eventB)
```

您还可以以以下方式表达此 operator：

```
abs($eventB.startTimestamp - $eventA.endTimestamp) == 0
```

meets operator 支持一个可选参数，用于设置当前事件的结束时间和关联事件的开始时间之间的最长时间：

```
$eventA : EventA(this meets[5s] $eventB)
```

如果满足这些条件，这个特征匹配：

```
abs($eventB.startTimestamp - $eventA.endTimestamp) <= 5s
```

警告

决策引擎不支持 meet operator 的负间隔。如果您使用负间隔，则决策引擎会生成错误。

达到的

此运算符指定关联事件在当前事件启动时是否同时结束。（此运算符的行为与 meets 运算符 的行为 相反。）

例如，如果 $eventB 在 $eventA 启动时同时结束，则以下模式匹配：

```
$eventA : EventA(this metby $eventB)
```

您还可以以以下方式表达此 operator：

```
abs($eventA.startTimestamp - $eventB.endTimestamp) == 0
```

Operator 达到 的可选参数支持一个可选参数，用于设置关联事件结束时间和当前事件的开始时间之间的最大距离：

```
$eventA : EventA(this metby[5s] $eventB)
```

如果满足这些条件，这个特征匹配：

```
abs($eventA.startTimestamp - $eventB.endTimestamp) <= 5s
```

警告

决策引擎不支持 operator 达到 的负间隔。如果您使用负间隔，则决策引擎会生成错误。

重叠

此 operator 指定在关联事件启动前当前事件是否启动，它会在关联事件发生的时间范围内结束。当前事件必须在相关事件的开始和结束时间之间结束。

例如，如果 $eventA 在 $eventB 开始前，如果启动了 $eventA，则匹配，然后在 $eventB 结束 $eventB 时结束：

```
$eventA : EventA(this overlaps $eventB)
```

重叠 Operator 支持最多两个参数：

- 如果定义了参数，则该值是关联事件的开始时间和当前事件的结束时间之间的最大距离。
- 如果定义了两个参数，值为关联事件开始时间和当前事件结束时间之间的最小距离（第一个值）和最大距离（秒数）。

重叠，

此运算符指定关联事件是否在当前事件启动前启动，并在当前事件发生的时间范围内结束。相关事件必须在当前事件的开始和结束时间之间结束。（此运算符的行为是 重叠 运算符的行为。）

例如，如果在 $eventA 启动时启动 $eventB，则以下模式匹配，然后在 $eventA 结束前结束：

```
$eventA : EventA(this overlappedby $eventB)
```

Operator 重叠 支持最多两个参数：

- 如果定义一个参数，则该值是当前事件开始时间和关联事件的结束时间之间的最大距离。
- 如果定义了两个参数，值为当前事件开始时间和关联事件结束时间之间的最小距离（第一个值）和最大距离（秒数）。

Starting

此运算符指定两个事件是否同时启动，但当前事件在关联事件结束前结束。

例如，如果 $eventA 和 $eventB 的同时启动，以下模式匹配，$eventA 结束 $eventA 结束：

```
$eventA : EventA(this starts $eventB)
```

您还可以以以下方式表达此 operator：

```
$eventA.startTimestamp == $eventB.startTimestamp
&&
$eventA.endTimestamp < $eventB.endTimestamp
```

start operator 支持一个可选参数，用于设置两个事件启动时间之间的最大距离：

```
$eventA : EventA(this starts[5s] $eventB)
```

如果满足这些条件，这个特征匹配：

```
abs($eventA.startTimestamp - $eventB.startTimestamp) <= 5s
&&
$eventA.endTimestamp < $eventB.endTimestamp
```

警告

决策引擎不支持 启动 operator 的负间隔。如果您使用负间隔，则决策引擎会生成错误。

启动者

此运算符指定两个事件是否同时启动，但关联事件在当前事件结束前结束。（此 Operator 的行为与 启动 Operator 行为相反。）

例如，如果 $eventA 和 $eventB 同时启动，则以下模式匹配，$eventB 在 $eventA 结束前终止：

```
$eventA : EventA(this startedby $eventB)
```

您还可以以以下方式表达此 operator：

```
$eventA.startTimestamp == $eventB.startTimestamp
&&
$eventA.endTimestamp > $eventB.endTimestamp
```

-

Operator 启动的 一个可选参数支持在两个事件的开始时间之间设置最大距离：

```
$eventA : EventA( this starts[5s] $eventB)
```

如果满足这些条件，这个特征匹配：

```
abs( $eventA.startTimestamp - $eventB.startTimestamp ) <= 5s
&&
$eventA.endTimestamp > $eventB.endTimestamp
```

警告

决策引擎不支持 operator 启动 的负间隔。如果您使用负间隔，则决策引擎会生成错误。

85.7. 决定引擎中的会话时钟实现

在复杂的事件处理过程中，决策引擎中的事件可能会具有临时限制，因此需要会话时钟可提供当前时间。例如，如果某个规则需要确定给定库存在最后 60 分钟的平均价格，则决策引擎必须能够将库存价格事件时间戳与会话时钟中的当前时间进行比较。

决策引擎支持实时时钟和伪时钟。根据具体情况，您可以使用一个或多个时钟类型：

- 规则测试： 测试需要一个受控的环境，当测试包含具有临时限制的规则时，您必须能够控制输入规则和事实以及时间流。
- 定期执行： 决策引擎实时响应事件，因此需要实时时钟。
- 特殊环境： 特定环境可能具有特定时间控制要求。例如，集群环境可能需要时钟同步或 Java Enterprise Edition(JEE)环境，可能需要由应用服务器提供的时钟。

- 规则重播或模拟：若要重播或模拟，应用必须能够控制时间流。

当您决定在决策引擎中使用实时时钟还是伪时钟时，请考虑您的环境要求。

实时时钟

实时时钟是决策引擎中的默认时钟实施，使用系统时钟确定时间戳的当前时间。要将决策引擎配置为使用实时时钟，请将 KIE 会话配置参数设置为 realtime ：

在 KIE 会话中配置实时时钟

```
import org.kie.api.KieServices.Factory;
import org.kie.api.runtime.conf.ClockTypeOption;
import org.kie.api.runtime.KieSessionConfiguration;

KieSessionConfiguration config =
KieServices.Factory.get().newKieSessionConfiguration();

config.setOption(ClockTypeOption.get("realtime"));
```

伪时钟

决策引擎中的伪时钟实施有助于测试临时规则，并由应用程序控制。要将决策引擎配置为使用伪时钟，请将 KIE 会话配置参数设置为 伪 ：

在 KIE 会话中配置伪时钟

```
import org.kie.api.runtime.conf.ClockTypeOption;
import org.kie.api.runtime.KieSessionConfiguration;
import org.kie.api.KieServices.Factory;

KieSessionConfiguration config =
KieServices.Factory.get().newKieSessionConfiguration();

config.setOption(ClockTypeOption.get("pseudo"));
```

您还可以使用额外的配置和事实处理程序来控制伪时钟：

在 KIE 会话中控制伪时钟行为

```
import java.util.concurrent.TimeUnit;

import org.kie.api.runtime.KieSessionConfiguration;
import org.kie.api.KieServices.Factory;
import org.kie.api.runtime.KieSession;
import org.drools.core.time.SessionPseudoClock;
import org.kie.api.runtime.rule.FactHandle;
import org.kie.api.runtime.conf.ClockTypeOption;

KieSessionConfiguration conf = KieServices.Factory.get().newKieSessionConfiguration();

conf.setOption( ClockTypeOption.get("pseudo"));
KieSession session = kbase.newKieSession(conf, null);

SessionPseudoClock clock = session.getSessionClock();

// While inserting facts, advance the clock as necessary.
FactHandle handle1 = session.insert(tick1);
clock.advanceTime(10, TimeUnit.SECONDS);

FactHandle handle2 = session.insert(tick2);
clock.advanceTime(30, TimeUnit.SECONDS);

FactHandle handle3 = session.insert(tick3);
```

85.8. 事件流和入口点

决策引擎以事件流的形式处理大量事件。在 DRL 规则声明中，流也称为 入口点。当您在 DRL 规则或 Java 应用程序中声明一个入口点时，决策引擎在编译时标识并创建正确的内部结构，以仅评估该规则。

个入口点或流中的事实(fact)也可以加入决策引擎工作内存中的任何其他入口点的事实。事实始终与输入决策引擎的入口点保持关联。同一类型的事实可以通过几个入口点输入决策引擎，但通过入口点 A 进入决策引擎的事实永远不会与入口点 B 的模式匹配。

事件源有以下特征：

- 流中的事件按照时间戳排序。时间戳可能有不同的语义，但这些语义始终在内部排序。
- 活动流通常具有大量事件。
- 流中的 Atomic 事件通常不会单独使用，仅在流中集中使用。
- 事件流可以是同构，并且包含单一类型的事件，也可以是异构类型，并且包含不同类型的事件。

85.8.1. 为规则数据声明入口点

您可以为事件声明入口点（事件流），以便决策引擎仅使用该入口点中的数据来评估规则。您可以在 DRL 规则或您的 Java 应用程序中引用一个入口点，以隐式声明该入口点。

流程

使用以下方法之一来声明入口点：

- 在 DRL 规则文件中，为插入的事实从入口点 "<name>" 指定：

使用 "ATM Stream" 入口点授权撤回规则

```
rule "Authorize withdrawal"
when
  WithdrawRequest($ai : accountId, $am : amount) from entry-point "ATM Stream"
  CheckingAccount(accountId == $ai, balance > $am)
then
  // Authorize withdrawal.
end
```

使用 "Branch Stream" 入口点应用费用规则

```
rule "Apply fee on withdraws on branches"
when
  WithdrawRequest($ai : accountId, processed == true) from entry-point "Branch Stream"
  CheckingAccount(accountId == $ai)
then
  // Apply a $2 fee on the account.
end
```

银行应用中的 DRL 规则示例插入事件 WithdrawalRequest，其事实 检查帐户，但来自不同入口点。在运行时，决策引擎仅使用 "ATM Stream" 入口点中的数据来评估 授权 撤回规则，并使用仅来自 "Branch Stream" 入口点中的数据评估 应用费用 规则。插入到 "ATM Stream" 的任何事件都无法匹配 "应用费用" 规则的模式，并且插入到 "Branch Stream" 的任何事件都无法匹配 "Authorizeraw" 的模式。

- 在 Java 应用代码中，使用 getEntryPoint（） 方法来指定和获取 EntryPoint 对象，并相应地将事实插入到该入口点中：

带有 EntryPoint 对象的 Java 应用代码并插入事实

```
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.EntryPoint;

// Create your KIE base and KIE session as usual.
KieSession session = ...

// Create a reference to the entry point.
EntryPoint atmStream = session.getEntryPoint("ATM Stream");

// Start inserting your facts into the entry point.
atmStream.insert(aWithdrawRequest);
```

然后，任何从 入口点"ATM Stream"指定的 DRL 规则仅根据此入口点中的数据进行评估。

85.9. 滑动时间窗或长度

在流模式中，决策引擎可以从指定滑动时间或长度窗口处理事件。滑动时间窗是可以处理事件的指定时间段。滑动长度窗口是可以处理的指定数量事件。当您在 DRL 规则或 Java 应用程序中声明一个滑动

窗口时，在决策引擎编译时，标识并创建正确的内部结构，以仅使用滑动窗口来评估该规则。

例如，以下 DRL 规则片断指示决策引擎只处理过去 2 分钟（指定时间窗）或只处理最后 10 个库存点（光纤通道窗口）中的库存点：

来自最后 2 分钟的进程库存点（指定时间窗）

```
StockPoint() over window:time(2m)
```

处理最后 10 个库存点（计算长度窗口）

```
StockPoint() over window:length(10)
```

85.9.1. 为规则数据声明滑动窗口

您可以为事件声明一个时间窗口（时间流）或长度（发生次数），以便决策引擎仅使用该窗口中的数据来评估规则。

流程

在 DRL 规则文件中，为插入的事实指定 window:<time_or_length>(<value>)。

例如，下面两个 DRL 规则会基于平均温度激活触发警报。但是，第一条规则使用一个滑动时间窗来计算最近 10 分钟的平均时间，第二个规则使用了滑动长度窗口来计算过去一百个温度读数的平均值。

平均温度超过滑动时间窗

```
rule "Sound the alarm if temperature rises above threshold"
when
  TemperatureThreshold($max : max)
```

```
  Number(doubleValue > $max) from accumulate(
    SensorReading($temp : temperature) over window:time(10m),
    average($temp))
then
  // Sound the alarm.
end
```

平均温度超过滑动长度窗口

```
rule "Sound the alarm if temperature rises above threshold"
when
  TemperatureThreshold($max : max)
  Number(doubleValue > $max) from accumulate(
    SensorReading($temp : temperature) over window:length(100),
    average($temp))
then
  // Sound the alarm.
end
```

决策引擎丢弃任何超过 10 分钟以上的 SensorReading 事件，或者并非最后一百阅读的一部分，继续重新计算平均值，或实时读取"隐藏"转发。

决策引擎不会自动从 KIE 会话中删除过时的事件，因为其他规则没有滑动窗口声明可能依赖于这些事件。决策引擎将事件存储在 KIE 会话中，直到事件被显式规则声明过期，或者在基于 KIE 基础中的数据的决策引擎内隐式原因。

85.10. 事件的内存管理

在流模式中，决策引擎使用自动内存管理来维护存储在 KIE 会话中的事件。决策引擎可以从 KIE 会话处理任何事件，这些事件不再与任何规则匹配，因为它们是临时限制，并释放被重新处理的事件所保留的任何资源。

决策引擎使用显式或推断的过期日期来改变过时的事件：

- 显式到期： 决定引擎删除在声明 @expires 标签的规则中明确设置为过期的事件：

带有显式过期的 DRL 规则片断

```
declare StockPoint
  @expires( 30m )
end
```

这个示例规则会将任何 stockPoint 事件设置为在 30 分钟后过期，并在 KIE 会话中删除（如果没有其他规则使用事件）。

- Inferred 过期： 决定引擎通过分析规则中的临时限制来隐式计算给定事件的过期偏移：

具有临时限制的 DRL 规则

```
rule "Correlate orders"
when
  $bo : BuyOrder($id : id)
  $ae : AckOrder(id == $id, this after[0,10s] $bo)
then
  // Perform an action.
end
```

在本示例规则中，决策引擎自动计算发生 BuyOrder 事件时，决策引擎需要存储最多 10 秒的事件，并等待匹配的 AckOrder 事件。10 秒后，决策引擎将到期，并将事件从 KIE 会话中删除。AckOrder 事件只能与现有的 BuyOrder 事件匹配，因此决定引擎在没有匹配项并立即删除事件时推断出到期。

决策引擎会分析整个 KIE 基础，以查找每个事件类型的偏移，并确保没有其他规则使用待处理的事件。每当带有显式过期值的隐式到期值时，决策引擎会使用两个的时间更短的时间范围来存储事件的时间更长。

第 86 章 决策引擎查询和实时查询

您可以将查询与决策引擎配合使用，以检索基于规则中事实模式的事实集。模式可能也使用可选参数。

要将查询与决策引擎一起使用，您可以在 DRL 文件中添加查询定义，然后获取应用程序代码的匹配结果。虽然查询迭代结果集合，但您可以使用绑定到查询的任何标识符来访问对应的 fact 或 fact 字段，方法是调用带有绑定变量名称的 get（） 方法作为其参数。如果绑定引用了事实对象，您可以通过调用 getFactHandle（） 和变量名称作为 参数来检索事实句柄。

DRL 文件中的查询定义示例

```
query "people under the age of 21"
    $person : Person( age < 21 )
end
```

获取和迭代查询结果的应用程序代码示例

```
QueryResults results = ksession.getQueryResults( "people under the age of 21" );
System.out.println( "we have " + results.size() + " people under the age of 21" );

System.out.println( "These people are under the age of 21:" );

for ( QueryResultsRow row : results ) {
    Person person = ( Person ) row.get( "person" );
    System.out.println( person.getName() + "\n" );
}
```

监控时间发生变化时，通过迭代返回的集合调用查询和处理结果会比较困难。为了减少对持续查询的难度，Red Hat Process Automation Manager 提供了 实时查询，它使用附加的监听程序更改事件，而不是返回可迭代结果集。通过创建视图并为此视图的内容发布更改事件，以保持实时查询保持打开状态。

要激活实时查询，请使用参数启动查询，并在生成的视图中监控更改。您可以使用 dispose（） 方法终止查询并停止此重试场景。

DRL 文件中的查询定义示例

```
query colors(String $color1, String $color2)
    TShirt(mainColor = $color1, secondColor = $color2, $price: manufactureCost)
end
```

带有事件监听程序和实时查询的应用程序代码示例

```
final List updated = new ArrayList();
final List removed = new ArrayList();
final List added = new ArrayList();

ViewChangedEventListener listener = new ViewChangedEventListener() {
 public void rowUpdated(Row row) {
  updated.add( row.get( "$price" ) );
 }

 public void rowRemoved(Row row) {
  removed.add( row.get( "$price" ) );
 }

 public void rowAdded(Row row) {
  added.add( row.get( "$price" ) );
 }
};

// Open the live query:
LiveQuery query = ksession.openLiveQuery( "colors",
                                  new Object[] { "red", "blue" },
                                  listener );
...
...

// Terminate the live query:
query.dispose()
```

第 87 章 决策引擎事件监听程序和调试日志记录

在执行操作时，决策引擎会生成事件，如事实插入和规则执行。如果您注册事件监听程序，则决策引擎会在执行活动时调用每个监听程序。

事件监听程序具有与不同类型的活动对应的方法。决策引擎将事件对象传递给每个方法；此对象包含有关特定活动的信息。

您的代码可以实施自定义事件监听程序，您还可以添加和删除注册的事件监听程序。这样，您的代码可以收到决策引擎活动的通知，您可以将日志记录和审核工作与应用程序的核心分开。

决策引擎使用以下方法支持以下事件监听程序：

参与事件监听程序

```
public interface AgendaEventListener
    extends
    EventListener {
    void matchCreated(MatchCreatedEvent event);
    void matchCancelled(MatchCancelledEvent event);
    void beforeMatchFired(BeforeMatchFiredEvent event);
    void afterMatchFired(AfterMatchFiredEvent event);
    void agendaGroupPopped(AgendaGroupPoppedEvent event);
    void agendaGroupPushed(AgendaGroupPushedEvent event);
    void beforeRuleFlowGroupActivated(RuleFlowGroupActivatedEvent event);
    void afterRuleFlowGroupActivated(RuleFlowGroupActivatedEvent event);
    void beforeRuleFlowGroupDeactivated(RuleFlowGroupDeactivatedEvent event);
    void afterRuleFlowGroupDeactivated(RuleFlowGroupDeactivatedEvent event);
}
```

规则运行时事件监听程序

```
public interface RuleRuntimeEventListener extends EventListener {
    void objectInserted(ObjectInsertedEvent event);
    void objectUpdated(ObjectUpdatedEvent event);
    void objectDeleted(ObjectDeletedEvent event);
}
```

有关事件类的定义，请参阅 GitHub 存储库。

Red Hat Process Automation Manager 包括以下监听程序的默认实现：DefaultAgendaEventListener 和 DefaultRuleRuntimeEventListener。您可以扩展这些实施来监控特定的事件。

例如，以下代码扩展了 DefaultAgendaEventListener 以监控 AfterMatchFiredEvent 事件，并将这个监听程序附加到 KIE 会话。执行规则时代码打印模式匹配（填充）：

在日程表中监控和打印 AfterMatchFiredEvent 事件的代码示例

```
ksession.addEventListener( new DefaultAgendaEventListener() {
  public void afterMatchFired(AfterMatchFiredEvent event) {
    super.afterMatchFired( event );
    System.out.println( event );
  }
});
```

Red Hat Process Automation Manager 还包括以下决策引擎和规则运行时事件监听程序：

- DebugAgendaEventListener
- DebugRuleRuntimeEventListener

这些事件监听器实施相同的受支持的 event-listener 方法，并默认包括一个 debug print 语句。您可以为特定支持的事件添加额外的监控代码。

例如，以下代码使用 DebugRuleRuntimeEventListener 事件监听程序来监控和打印所有工作内存（规则运行时）事件：

监控和打印所有工作内存事件的代码示例

```
ksession.addEventListener( new DebugRuleRuntimeEventListener() );
```

87.1. 事件监听程序开发实践

决策引擎在规则处理过程中调用事件监听程序。调用会阻止决策引擎的执行。因此，事件监听程序可能会影响决策引擎的性能。

要确保中断最小，请遵循以下指南：

- 任何操作都必须尽可能短。
- 侦听器类不能处于状态。决策引擎可以随时销毁并重新创建监听程序类。
- 不要使用依赖于不同事件监听程序执行顺序的逻辑。
- 不要包括与监听器内决策引擎之外的不同实体交互。例如，请勿包含用于事件通知的 REST 调用。一个例外是日志信息的输出，但日志记录监听程序必须尽可能简单。
- 您可以使用监听程序来修改决策引擎的状态，例如更改变量的值。

第 88 章 在决策引擎中配置日志记录工具

决策引擎将 Java 日志 API SLF4J 用于系统日志。您可以将以下日志记录工具与决策引擎一起使用以调查决策引擎活动，如进行故障排除或数据收集：

- logback
- Apache Commons Logging
- Apache Log4j
- java.util.logging package

流程

对于您要使用的日志记录实用程序，请将相关依赖项添加到 Maven 项目，或者在 Red Hat Process Automation Manager 发行版的 org.drools 软件包中保存相关的 XML 配置文件：

Logback 的 Maven 依赖项示例

```
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

org.drools 软件包中的 logback.xml 配置文件示例

```
<configuration>
  <logger name="org.drools" level="debug"/>
  ...
<configuration>
```

org.drools 软件包中的 log4j.xml 配置文件示例

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">
  <category name="org.drools">
    <priority value="debug" />
  </category>
  ...
</log4j:configuration>
```

注意

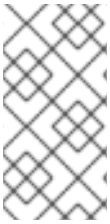
如果您是为方便的小型环境进行开发，请使用 slf4j-nop 或 slf4j-simple logger。

第 89 章 RED HAT PROCESS AUTOMATION MANAGER 中用于 IDE 的示例

Red Hat Process Automation Manager 提供了以 Java 类分发的示例决策，您可导入到您的集成开发环境(IDE)中。您可以使用这些示例来更好地了解决策引擎功能，或者将其用作您在 Red Hat Process Automation Manager 项目中定义的决策的参考。

以下示例决策集是 Red Hat Process Automation Manager 中提供的一些示例：

- hello World 示例 ：demonstrates 基本规则执行和使用调试输出
- State 示例 ：通过规则先和日程组展示转发链和冲突解决
- Fibonacci 示例 ：通过规则 salience 递归和冲突解决
- 银行示例 ：demonstrates 模式匹配、基本排序和计算
- pet Store 示例 ：演示规则映射组、全局变量、回调和 GUI 集成
- Sudoku 示例 ：演示复杂模式匹配、问题解决、回调和 GUI 集成
- Doom 示例内部 ：演示后链和递归



注意

有关红帽构建的 OptaPlanner 提供的优化示例，请参阅开始使用 Red Hat build of OptaPlanner。

89.1. 在 IDE 中导入和执行 RED HAT PROCESS AUTOMATION MANAGER 示例决策

您可以将 Red Hat Process Automation Manager 示例决策导入到您的集成开发环境(IDE)中，再执行它们来探索规则和代码功能。您可以使用这些示例来更好地了解决策引擎功能，或者将其用作您在 Red Hat Process Automation Manager 项目中定义的决策的参考。

先决条件

- 已安装 Java 8 或更高版本。
- 已安装 Maven 3.5.x 或更高版本。
- 安装了 IDE，如 Red Hat CodeReady Studio。

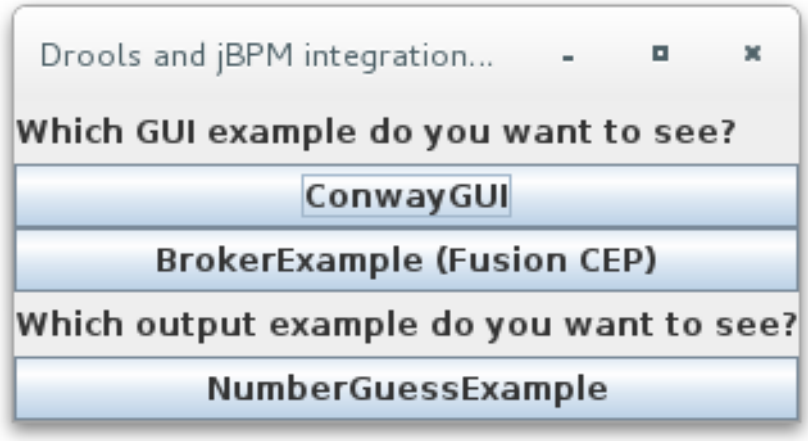
流程

1. 将 Red Hat Process Automation Manager 7.13.5 源分发从红帽客户门户网站下载到临时目录，如 /rhpam-7.13.5-sources。
2. 打开 IDE，然后选择 File → Import → Maven → Existing Maven Projects，或用于导入 Maven 项目的对等选项。
3. 单击 Browse，导航到 ~/rhpam-7.13.5-sources/src/drools-$VERSION/drools-examples（或，对于 Life 示例的 Conway's Game、~/rhpam-7.13.5-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples），再导入该项目。
4. 导航到您要运行的示例软件包，并使用 主 方法查找 Java 类。
5. 右键点击 Java 类并选择 Run As → Java Application 以运行示例。

要通过基本用户界面运行所有示例，请在 org.drools.examples 主类中运行 DroolsExamplesApp.java 类（或者，在 Conway 的 Game of Life 中）运行 DroolsJbpmIntegrationExamplesApp.java 类。

图 89.1. drools-examples(DroolsExamplesApp.java)中的所有示例接口

图 89.2. droolsjbpm-integration-examples(DroolsJbpmIntegrationExamplesApp.java)中的所有示例接口



89.2. HELLO WORLD 示例决策（基本规则和调试）

Hello World 示例决策集演示了如何将对象插入到决策引擎工作内存中，如何使用规则匹配对象，以及如何配置日志记录以跟踪决策引擎的内部活动。

以下是 Hello World 示例的概述：

- **名称 ：helloworld**
- **主要课程:org.drools.examples.helloworld.HelloWorldExample （在 src/main/java中）**
- **模块 ：drools-examples**
- **键入: Java 应用程序**
- **规则文件 ：org.drools.examples.helloworld.HelloWorld.drl (src/main/resources)**
- **目标 ：演示基本规则执行和使用调试输出**

在 Hello World 示例中，会生成一个 KIE 会话，以启用规则执行。所有规则都需要一个 KIE 会话来执行。

用于规则执行的 KIE 会话

```
KieServices ks = KieServices.Factory.get(); (1)
KieContainer kc = ks.getKieClasspathContainer(); (2)
KieSession ksession = kc.newKieSession("HelloWorldKS"); (3)
```

(1) 获取 KieServices 工厂。这是应用程序用来与决策引擎交互的主要界面。

(2) 从项目类路径创建 KieContainer。这会检测 /META-INF/kmodule.xml 文件，该文件使用 KieModule 配置并实例化 KieContainer。

(3) 根据 /META-INF/kmodule.xml 文件中定义的 "HelloWorldKS" KIE 会话配置创建一个 KieSession。

注意

有关 Red Hat Process Automation Manager 项目打包的更多信息，请参阅 *打包和部署 Red Hat Process Automation Manager 项目*。

Red Hat Process Automation Manager 有一个公开内部引擎活动的事件模型。两个默认调试监听器： DebugAgendaEventListener 和 DebugRuleRuntimeEventListener，将调试事件信息打印到 System.err 输出。KieRuntimeLogger 提供执行审核，您可以在图形查看器中查看结果。

调试监听器和审计记录

```
// Set up listeners.
ksession.addEventListener( new DebugAgendaEventListener() );
ksession.addEventListener( new DebugRuleRuntimeEventListener() );

// Set up a file-based audit logger.
KieRuntimeLogger logger = KieServices.get().getLoggers().newFileLogger( ksession,
```

```
"./target/helloworld" );

// Set up a ThreadedFileLogger so that the audit view reflects events while debugging.
KieRuntimeLogger logger = ks.getLoggers().newThreadedFileLogger( ksession,
"./target/helloworld", 1000 );
```

日志记录器是在 Agenda 和 RuleRuntime 监听程序基础上构建的专用实现。在决策引擎执行完成后，会调用 logger.close（）。

这个示例创建了一个带有消息 "Hello World" 的 Message 对象，将 status HELLO 插入到 KieSession 中，使用 fireAllRules（） 执行规则。

数据插入和执行

```
// Insert facts into the KIE session.
final Message message = new Message();
message.setMessage( "Hello World" );
message.setStatus( Message.HELLO );
ksession.insert( message );

// Fire the rules.
ksession.fireAllRules();
```

规则执行使用数据模型将数据作为输入和输出到 KieSession。本例中的数据模型有两个字段：消息，它是一个 String，其状态为 HELLO 或 GOODBYE。

数据模型类

```
public static class Message {
    public static final int HELLO   = 0;
    public static final int GOODBYE = 1;

    private String         message;
    private int            status;
    ...
}
```

这两个规则位于 src/main/resources/org/drools/helloworld/helloworld/HelloWorld.drl 文件中。

"Hello World" 规则的 when 条件指出，针对插入到 KIE 会话的每个 Message 对象激活规则，其具有 status Message.HELLO。另外，会创建两个变量绑定：变量 消息 绑定到 message 属性，变量 m 绑定到匹配的 Message 对象本身。

规则的 then 操作指定将绑定变量 消息 的内容打印到 System.out，然后更改绑定到 m 的 Message 对象 的消息 和状态属性的值。该规则使用 modify 语句应用一个语句中的分配块，并通知块末尾的更改的决策引擎。

"hello World" 规则

```
rule "Hello World"
  when
    m : Message( status == Message.HELLO, message : message )
  then
    System.out.println( message );
    modify ( m ) { message = "Goodbye cruel world",
                   status = Message.GOODBYE };
end
```

"Good Bye" 规则与 "Hello World" 规则类似，但它会与具有状态 Message.GOODBYE 的 Message 对象匹配。

"好"规则

```
rule "Good Bye"
  when
    Message( status == Message.GOODBYE, message : message )
  then
    System.out.println( message );
end
```

要执行示例，请在 IDE 中将 org.drools.examples.helloworld.HelloWorldExample 类作为 Java 应用程序运行。该规则会写入到 System.out，debug 侦听器写入 System.err，审计日志记录器 在目标/helloworld.log 中创建日志文件。

IDE 控制台中的 system.out 输出

```
Hello World
Goodbye cruel world
```

IDE 控制台中的 system.err 输出

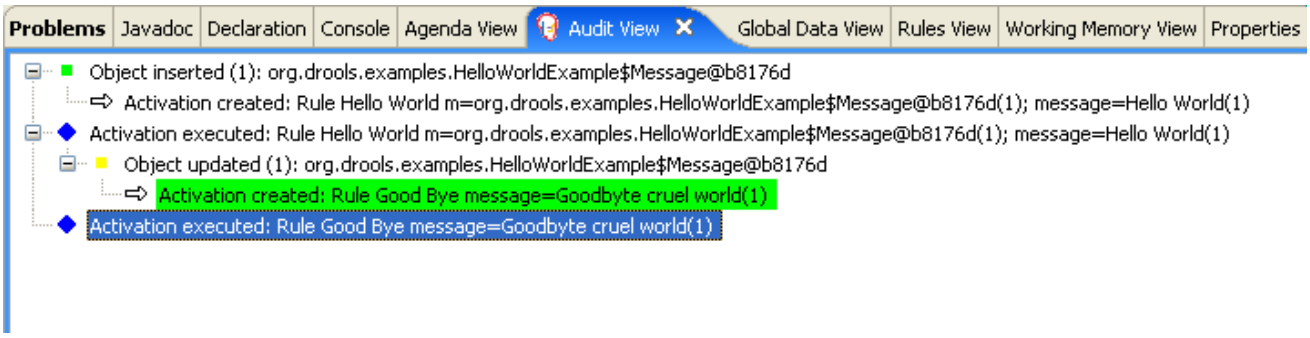
```
==>[ActivationCreated(0): rule=Hello World;
                tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectInserted: handle=
[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
              object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[BeforeActivationFired: rule=Hello World;
                tuple=[fid:1:1:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
==>[ActivationCreated(4): rule=Good Bye;
                tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[ObjectUpdated: handle=
[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96];
             old_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96;
             new_object=org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]
[AfterActivationFired(0): rule=Hello World]
[BeforeActivationFired: rule=Good Bye;
                tuple=[fid:1:2:org.drools.examples.helloworld.HelloWorldExample$Message@17cec96]]
[AfterActivationFired(4): rule=Good Bye]
```

为了更好地了解本例中的执行流，您可以将审计日志文件从 target/helloworld.log 加载到 IDE 调试视图或 Audit View （例如，如果可用）。

在本例中，audit 视图显示 对象已插入，这将为 "Hello World" 规则创建一个激活。然后执行激活，它

会更新 Message 对象并导致 "Good Bye" 规则激活。最后，执行 "Good Bye" 规则。当您在 Audit View 中选择一个事件时，origin 事件（此示例中是 "创建的" 事件）将以绿色突出显示。

图 89.3. hello World 示例 Audit 视图



89.3. 状态决策示例（转发链和冲突解析）

State 示例决策集演示了决策引擎如何使用正向链以及正在正常工作内存中的事实更改来解决按顺序规则的执行冲突。该示例着重介绍通过相同值或通过在规则中定义的日程表组解决冲突。

以下是状态示例概述：

- 名称 ：状态
- 主类 ：org.drools.examples.state.StateExampleUsingSalience,org.drools.examples.state.StateExampleUsingAgendaGroup （在 src/main/java中）
- 模块 ：drools-examples
- 键入：Java 应用程序
- 规则文件 ：org.drools.examples.state.*.drl （在 src/main/resources）
- 目标 ：通过规则认证和日程组演示转发链和冲突解决情况

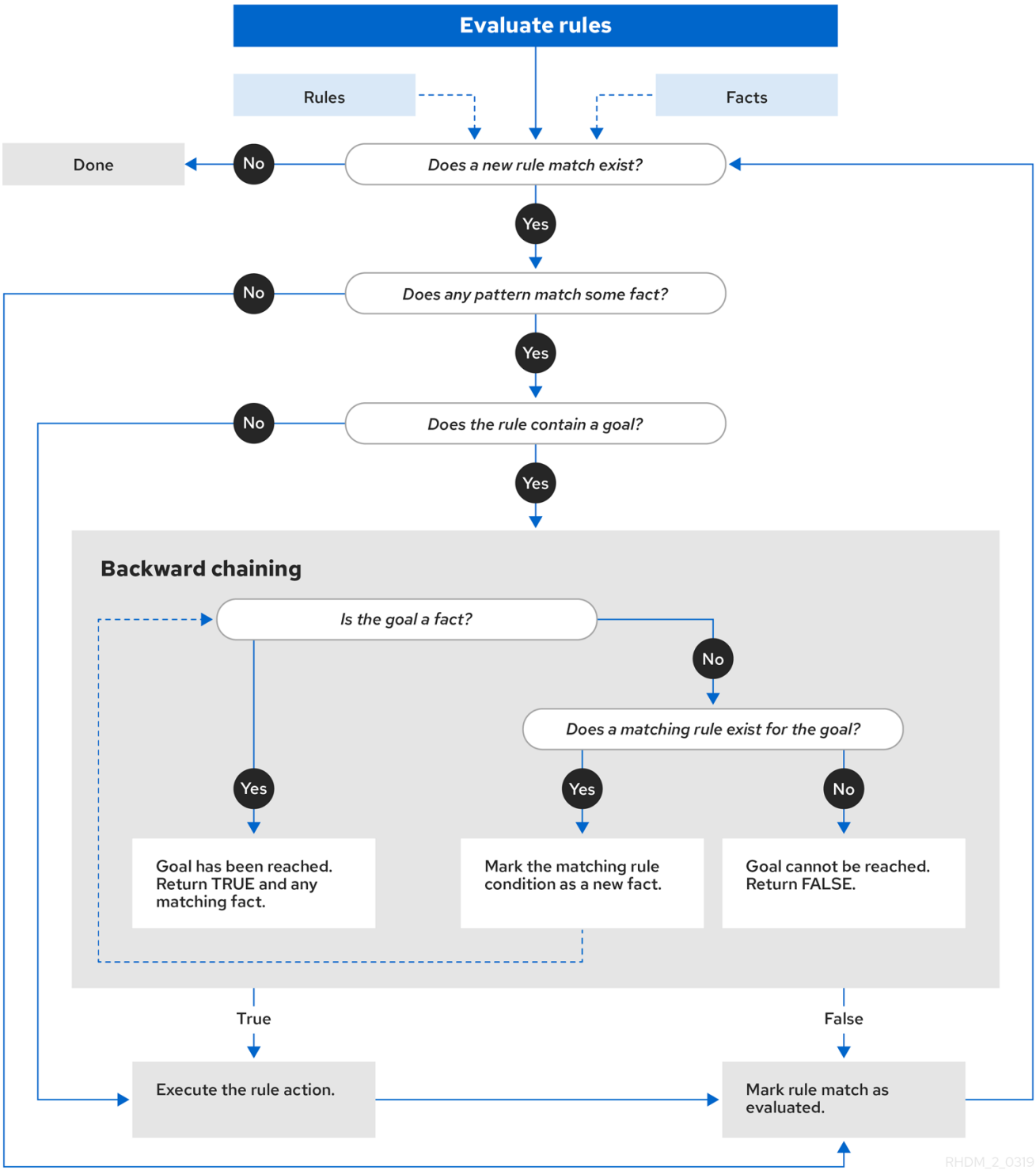
forward-chaining 规则系统是一个由数据驱动的系统，它从决策引擎的工作内存从事实开始，并对事实做出响应。当对象插入到工作内存时，因为更改是由日程表计划执行而变为 true 的任何规则条件。

相反，反向链接规则系统是一个由目标驱动的系统，从结论开始，决定引擎尝试满足，通常使用递归。如果系统无法达到结论或目标，它会搜索部分当前目标的子项。系统会继续这个过程，直到初始的结论是满足或者所有子语满意。

Red Hat Process Automation Manager 中的决策引擎使用正向和向后链来评估规则。

下图显示了如何使用转发链在逻辑流中的反向链接片段评估规则：

图 89.4. 使用转发和向后链的规则评估逻辑



在 State 示例中，每个 State 类都有一个名称及其当前状态的字段（请参阅类 org.drools.examples.state.State）。以下状态是每个对象的两个可能状态：

- NOTRUN

- 完成

State class

```
public class State {
   public static final int NOTRUN   = 0;
   public static final int FINISHED = 1;

   private final PropertyChangeSupport changes =
      new PropertyChangeSupport( this );

   private String name;
   private int    state;

   ... setters and getters go here...
}
```

State 示例包含两个版本的同一示例，用于解决规则执行冲突：

- 一个 StateExampleUsingSalience 版本，它通过使用规则 salience 解决了冲突
- 一个 StateExampleUsingAgendaGroups 版本，它通过使用规则日程表组解决冲突

两个状态示例都涉及四个状态对象： A、B、C 和 D。最初，其状态设定为 NOTRUN，这是示例使用的构造器的默认值。

使用 salience 的州示例

State 示例的 StateExampleUsingSalience 版本使用规则中的 salience 值来解决规则执行冲突。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。

示例将每个 State 实例插入到 KIE 会话中，然后调用 fireAllRules（）。

Salience 状态执行示例

```
final State a = new State( "A" );
final State b = new State( "B" );
final State c = new State( "C" );
final State d = new State( "D" );

ksession.insert( a );
ksession.insert( b );
ksession.insert( c );
ksession.insert( d );

ksession.fireAllRules();

// Dispose KIE session if stateful (not required if stateless).
ksession.dispose();
```

要执行该示例，请运行 org.drools.examples.state.StateExampleUsingSalience 类作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的 Salience State 示例输出

```
A finished
B finished
C finished
D finished
```

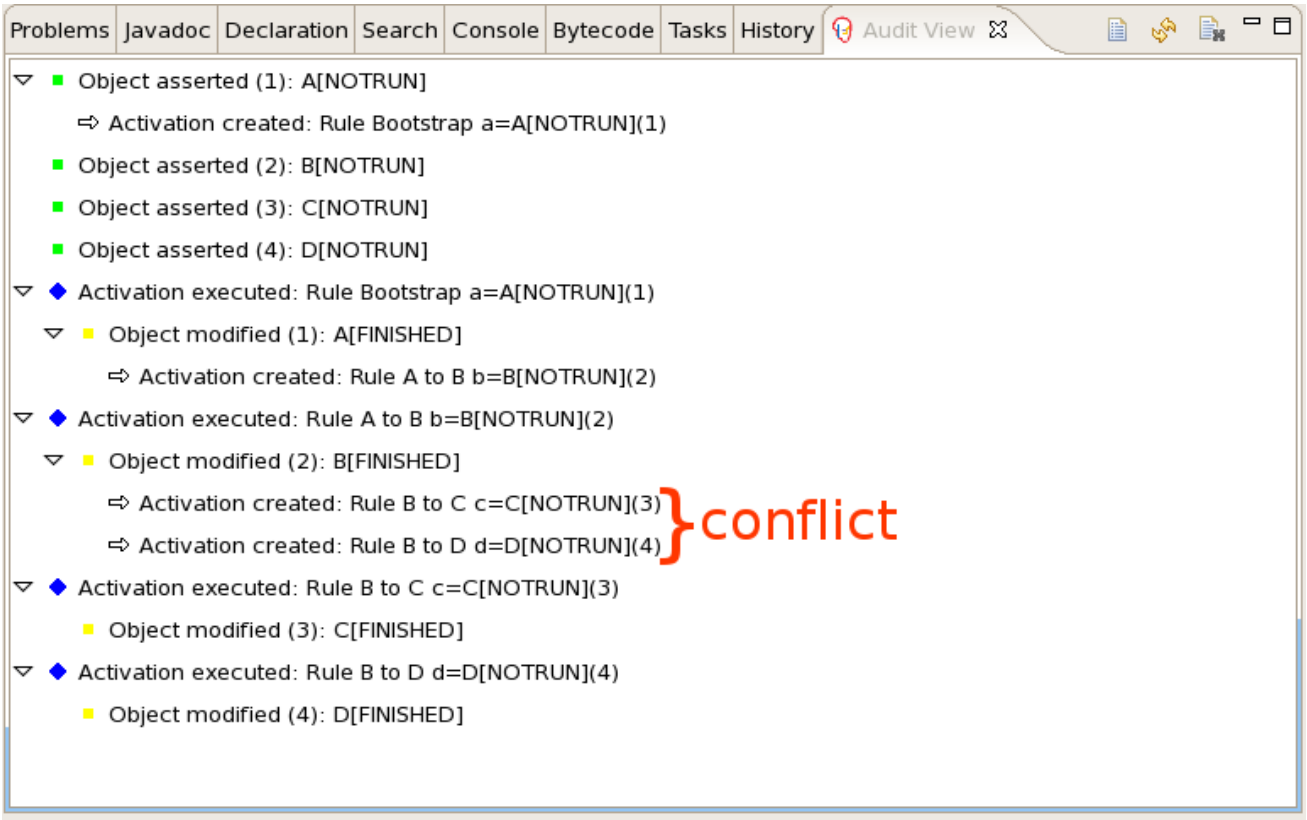
有四个规则：

首先，"Bootstrap" 规则触发，将 A 设置为状态为 FINISHED，然后使 B 将其状态更改为 FINISHED。对象 C 和 D 都依赖于 B，从而导致了值解析的冲突。

为了更好地了解本例中的执行流，您可以将审计日志文件从 target/state.log 加载到 IDE 调试视图或 Audit View （例如，如果可用）。

在本例中，审计视图显示状态中对象 A 的断言不会激活 "Bootstrap" 规则，而其他对象的断言没有立即生效。

图 89.5. Salience State 示例 Audit 视图



salience State 示例中的规则"引导"示例

```
rule "Bootstrap"
  when
    a : State(name == "A", state == State.NOTRUN )
  then
    System.out.println(a.getName() + " finished" );
    a.setState( State.FINISHED );
end
```

"Bootstrap" 规则的执行会将 A 的状态更改为 FINISHED，后者可激活规则 "A 到 B "。

salience State 示例中的规则"A 到 B"

```
rule "A to B"
  when
    State(name == "A", state == State.FINISHED )
    b : State(name == "B", state == State.NOTRUN )
  then
    System.out.println(b.getName() + " finished" );
    b.setState( State.FINISHED );
end
```

规则 **"A 到 B" 的执行将 B 的状态更改为 FINISHED，后者可激活 "B 到 C" 和 "B 到 D"，将其激活置于决策引擎日程表上。**

salience State 示例中的规则"B 至 C"和"B to D"

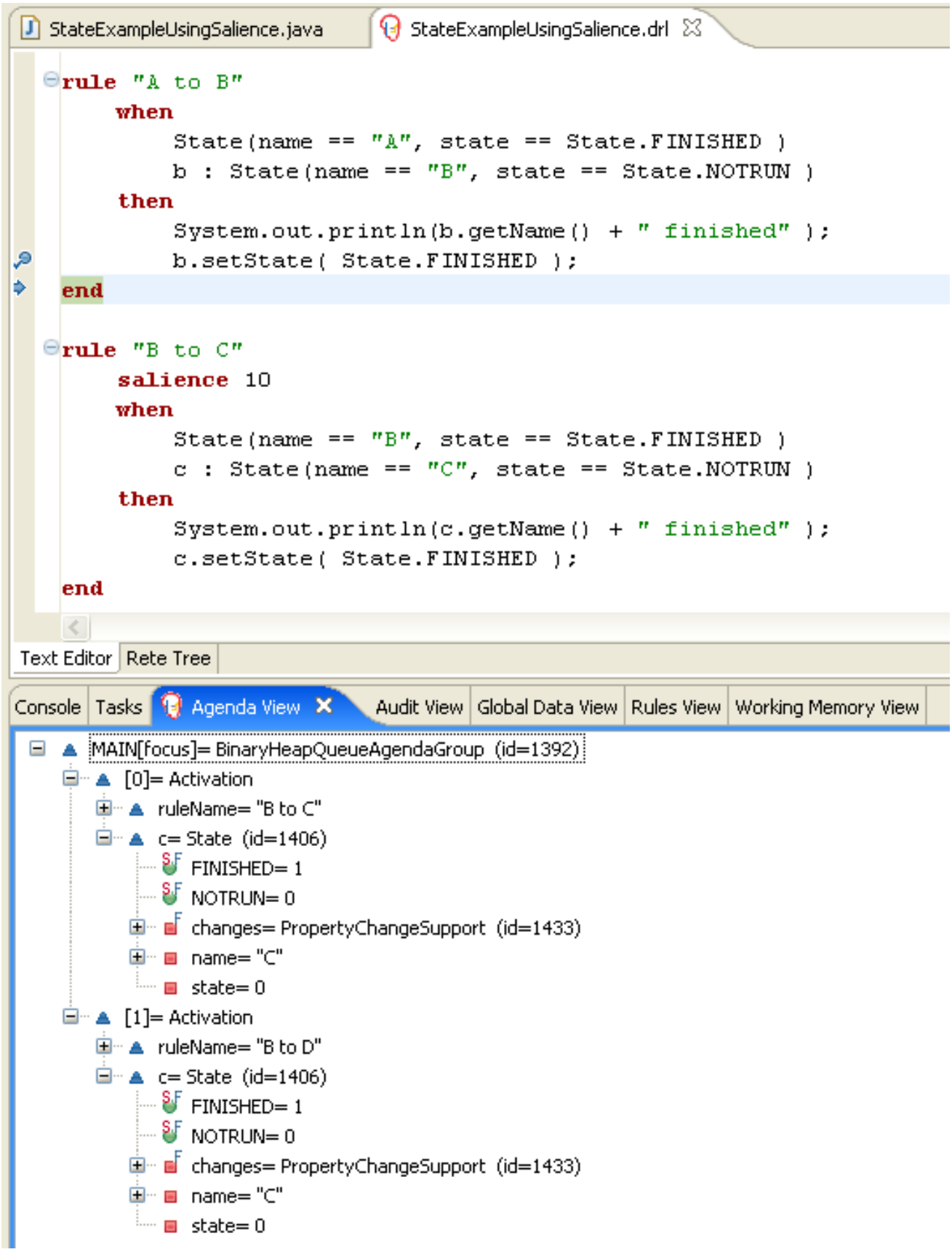
```
rule "B to C"
    salience 10
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
end

rule "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

此时，规则可能会触发，因此规则存在冲突。冲突解决策略使决策引擎日程表决定要触发的规则。规则 "B to C" 的值较高（10 与默认值 0不同），因此它首先触发对象 C，将对象 C 改为 state FINISHED。

IDE 中的 Audit 视图显示对规则 "A 到 B" 中的 State 对象的修改，这会导致两个激活冲突。

您还可以使用 IDE 中的 Agenda View 来调查决策引擎日程表的状态。在本例中，Agen da View 显示规则 "A 到 B" 中的断点，以及带有两个冲突的规则的日程表状态。规则 "B to D" 最后触发，将对象 D 修改为状态为 FINISHED。

图 89.6. Salience State example Agenda View



使用日程组进行状态示例

State 示例的 State 示例中的 StateExampleUsingAgendaGroups 版本使用 Rules 中的 table 组来解决规则执行冲突。日程表组使您可以对决策引擎日程表进行分区，以便对规则组提供更多执行控制。默认情况下，所有规则均位于 MAIN 日程小组。您可以使用 schedule -group 属性来指定该规则的不同日程表组。

最初，工作内存专注于 MAIN 的日程安排。仅当该组收到相关事项时，即可参与日程表组中的规则。您可以使用方法 setFocus（） 或 rule 属性 auto-focus 来设置焦点。auto-focus 属性允许当规则匹配和激活时，自动为课程安排人员自动给定规则。

在本例中，auto-focus 属性可让规则 "B to C" 在 "B to D" 前触发。

会议小组示例中的规则"B to C"

```
rule "B to C"
    agenda-group "B to C"
    auto-focus true
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished" );
    c.setState( State.FINISHED );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "B to D" ).setFocus();
end
```

在 Registration group "B to C" 中的规则 "B to C" 调用 setFocus（），使其活动规则可以触发，然后启用规则 "B to D "。

会议小组示例中的规则"B to D"

```
rule "B to D"
    agenda-group "B to D"
  when
    State(name == "B", state == State.FINISHED )
    d : State(name == "D", state == State.NOTRUN )
  then
```

```
    System.out.println(d.getName() + " finished" );
    d.setState( State.FINISHED );
end
```

要执行该示例，请运行 org.drools.examples.state.StateExampleUsingAgendaGroups 类作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出（与状态示例的 salience 版本相同）：

IDE 控制台中查看组状态示例输出

```
A finished
B finished
C finished
D finished
```

State 示例中的动态事实

此 State 示例中的另一个值得注意的概念是根据实施 PropertyChangeListener 对象的对象，使用动态事实。要让决策引擎查看和响应事实属性更改，应用程序必须通知发生更改的决策引擎。您可以使用 modify 语句在规则中明确配置此通信，或者通过指定事实实施 PropertyChangeSupport 接口（如 format）规范定义的 PropertyChangeSupport 接口来显式配置此通信。

本例演示了如何使用 PropertyChangeSupport 接口以避免规则中明确 修改 语句的需求。要使用此接口，请确保您的事实实施 PropertyChangeSupport，这与类 org.drools.example.State 实施的方式相同，然后在 DRL 规则文件中使用以下代码，将决策引擎配置为侦听这些事实上的属性更改：

声明动态事实

```
declare type State
  @propertyChangeSupport
end
```

使用 PropertyChangeListener 对象时，每个 setter 必须实施通知的额外代码。例如，以下 state 的设置者位于类 org.drools.examples 中：

带有 PropertyChangeSupport 的 setter 示例

```
public void setState(final int newState) {
    int oldState = this.state;
    this.state = newState;
    this.changes.firePropertyChange( "state",
                                     oldState,
                                     newState );
}
```

89.4. FIBONACCI 示例决策（接收和冲突解析）

Fibonacci 示例决策集演示了决策引擎如何使用递归来解析序列中的规则执行冲突。这个示例侧重于通过在规则中定义的同等值来解决冲突。

以下是 Fibonacci 示例概述：

- 名称：fibonacci
- 主类：org.drools.examples.fibonacci.FibonacciExample （在 src/main/java中）
- 模块：drools-examples
- 键入：Java 应用程序
- 规则文件：org.drools.examples.fibonacci.Fibonacci.drl (src/main/resources)

- 目标：通过规则策略检查和冲突解决

Fibonacci Numbers 形成一个以 0 和 1 开头的序列。下一个 Fibonacci 编号通过添加前两个 Fibonacci number 中获取：0, 1, 1, 2, 3, 5, 8, 8, 13, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946 等。

Fibonacci 示例使用单一事实类 Fibonacci 和以下两个字段：

- 序列
- value

sequence 字段显示对象在 Fibonacci 数字序列中的位置。value 字段显示该序列位置的 Fibonacci 对象的值，其中 -1 表示仍然需要计算的值。

Fibonacci 类

```
public static class Fibonacci {
    private int  sequence;
    private long value;

    public Fibonacci( final int sequence ) {
        this.sequence = sequence;
        this.value = -1;
    }

    ... setters and getters go here...
}
```

要执行该示例，请运行 org.drools.examples.fibonacci.FibonacciExample 类，作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的 Fibonacci 示例输出

```
recurse for 50
recurse for 49
recurse for 48
recurse for 47
...
recurse for 5
recurse for 4
recurse for 3
recurse for 2
1 == 1
2 == 1
3 == 2
4 == 3
5 == 5
6 == 8
...
47 == 2971215073
48 == 4807526976
49 == 7778742049
50 == 12586269025
```

要在 Java 中实现此行为，示例将一个 Fibonacci 对象插入一个 Fibonacci 对象，其序列字段为 50。然后，示例使用递归规则插入其他 49 Fibonacci 对象。

本示例使用 MVEL dialect 修改 关键字来启用块 setter 操作并通知决策引擎更改，而不是实现 PropertyChangeSupport 接口使用动态事实。

Fibonacci 示例执行

```
ksession.insert( new Fibonacci( 50 ) );
ksession.fireAllRules();
```

本例使用以下三个规则：

- "recurse"
- "bootstrap"
- "计算"

规则 "Recurse" 匹配每个断言的 Fibonacci 对象，其值为 -1，创建并断出新的 Fibonacci 对象，其序列比当前匹配对象小。每次添加 Fibonacci 对象时，只要存在等于 1 的 sequence 字段，则规则重新匹配并再次触发。如果没有 条件元素，当您在内存中拥有 50 Fibonacci 对象后，不使用条件元素停止匹配规则。该规则也有 salience 值，因为您需要在执行 "Bootstrap" 规则前断断所有 50 Fibonacci 对象。

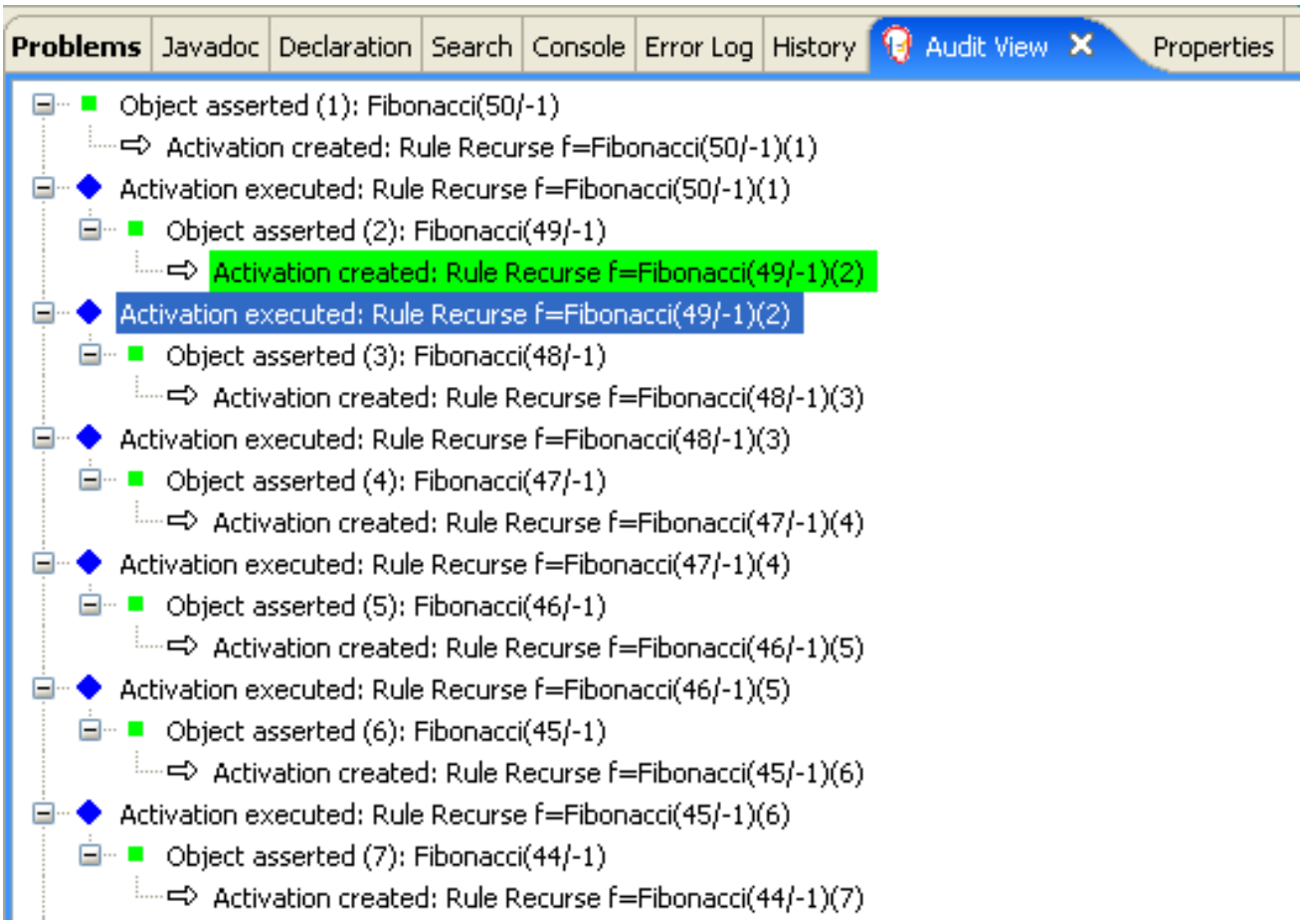
规则 "Recurse"

```
rule "Recurse"
    salience 10
  when
    f : Fibonacci ( value == -1 )
    not ( Fibonacci ( sequence == 1 ) )
  then
    insert( new Fibonacci( f.sequence - 1 ) );
    System.out.println( "recurse for " + f.sequence );
end
```

为了更好地理解本例的执行流，您可以将审计日志文件从 target/fibonacci.log 加载到 IDE 调试视图或 Audit View （例如，如果可用，位于 Window → Show View in some IDE）。

在本例中，审计 视图显示 Fibonacci 对象的原始断言，其 序列字段为 50，它通过 Java 代码完成。在那里，审计 视图显示规则的连续递归，其中每个断言的 Fibonacci 对象会导致 "Recurse" 规则变为激活并再次触发。

图 **89.7. Audit** 视图**中的**规则**"重复"**



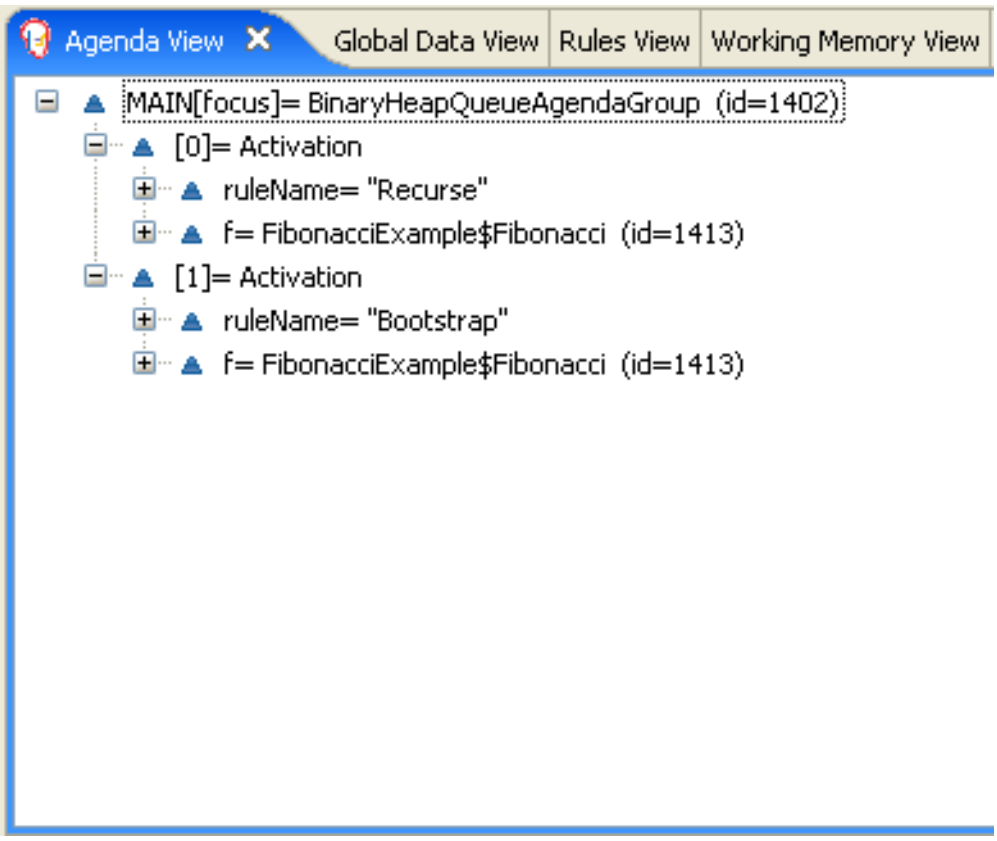
如果 Fibonacci** 带有 **序列字段的 Fib onacci** 对**象被断言，"Bootstrap"** 规则**会与 "Recurse"** 规则**一起匹配和激活。**请**注意，**对**字段 序列 的多个限制，使用 1 或 2** 测试**是否相等：

规则 ***"Bootstrap"***

```
rule "Bootstrap"
  when
    f : Fibonacci( sequence == 1 || == 2, value == -1 ) // multi-restriction
  then
    modify ( f ){ value = 1 };
    System.out.println( f.sequence + " == " + f.value );
end
```

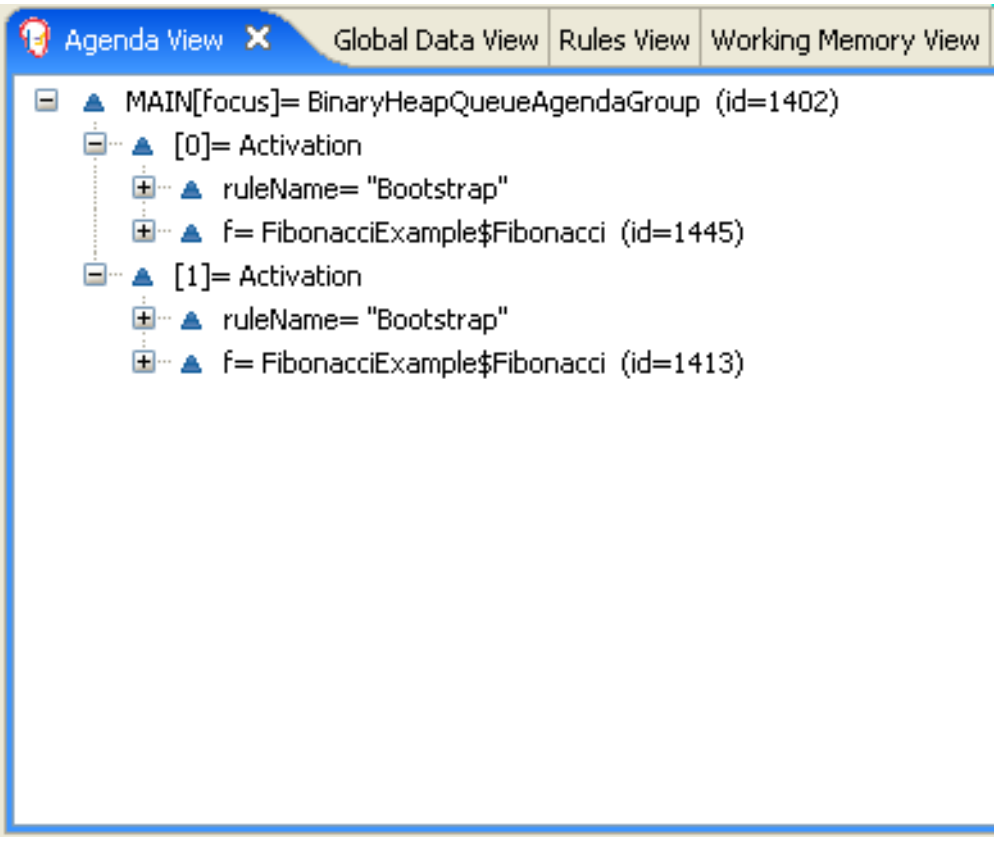
***您**还**可以使用 IDE 中的 Agenda View 来**调查**决策引擎日程表的状**态。**"Bootstrap"** 规则**尚未触**发，**因**为 **"Recurse"** 规则**具有更高的 salience** 值。*

*图 **89.8. Agenda View 1 中的规则"Recurse"和"Bootstrap"***



当断言了 序列 为 1 的 Fibonacci 对象时，会再次匹配 "Bootstrap" 规则，从而导致此规则有两个激活。"Recurse" 规则不匹配并激活，因为 not 条件元素在存在 序列 的 Fibonacci 对象时立即停止匹配规则。

*图 **89.9. Agenda View 2 中的规则"Recurse"和"Bootstrap"***



"Bootstrap" 规则使用 序列 1 和 2 的值来设置对象。现在，有两个 Fibonacci 对象的值不等于 -1，"Calculate" 规则可以匹配。

此时，工作内存中有近 50 Fibonacci 对象。您需要依次选择适当的三角来计算其每个值。如果您在没有字段限制的规则中使用三个 Fibonacci 模式来限制可能的跨产品，则结果将是 50x49x48 个可能的组合，从而导致大约 125,000 个可能的规则触发，其中大多数不正确。

"Calculate" 规则使用字段限制来以正确顺序评估三个 Fibonacci 模式。这种技术称为与 跨产品匹配的跨产品。

第一个模式找到任何值为 != -1 的 Fibonacci 对象，并且绑定模式和字段。第二个 Fibonacci 对象执行相同的操作，但添加了额外的字段约束，以确保其序列大于绑定到 f1 的 Fibonacci 对象。当这个规则首次触发时，您知道只有序列 1 和 2 的值为 1，并且两个限制可确保 f1 参考序列 1 和 f2 参考序列 2。

最终模式找到一个值等于 -1 且序列大于 f2 的 Fibonacci 对象。

此时，可以从可用的跨产品正确选择三个 Fibonacci 对象，您可以计算绑定到 f3 的第三个 Fibonacci 对象的值。

规则"计算"

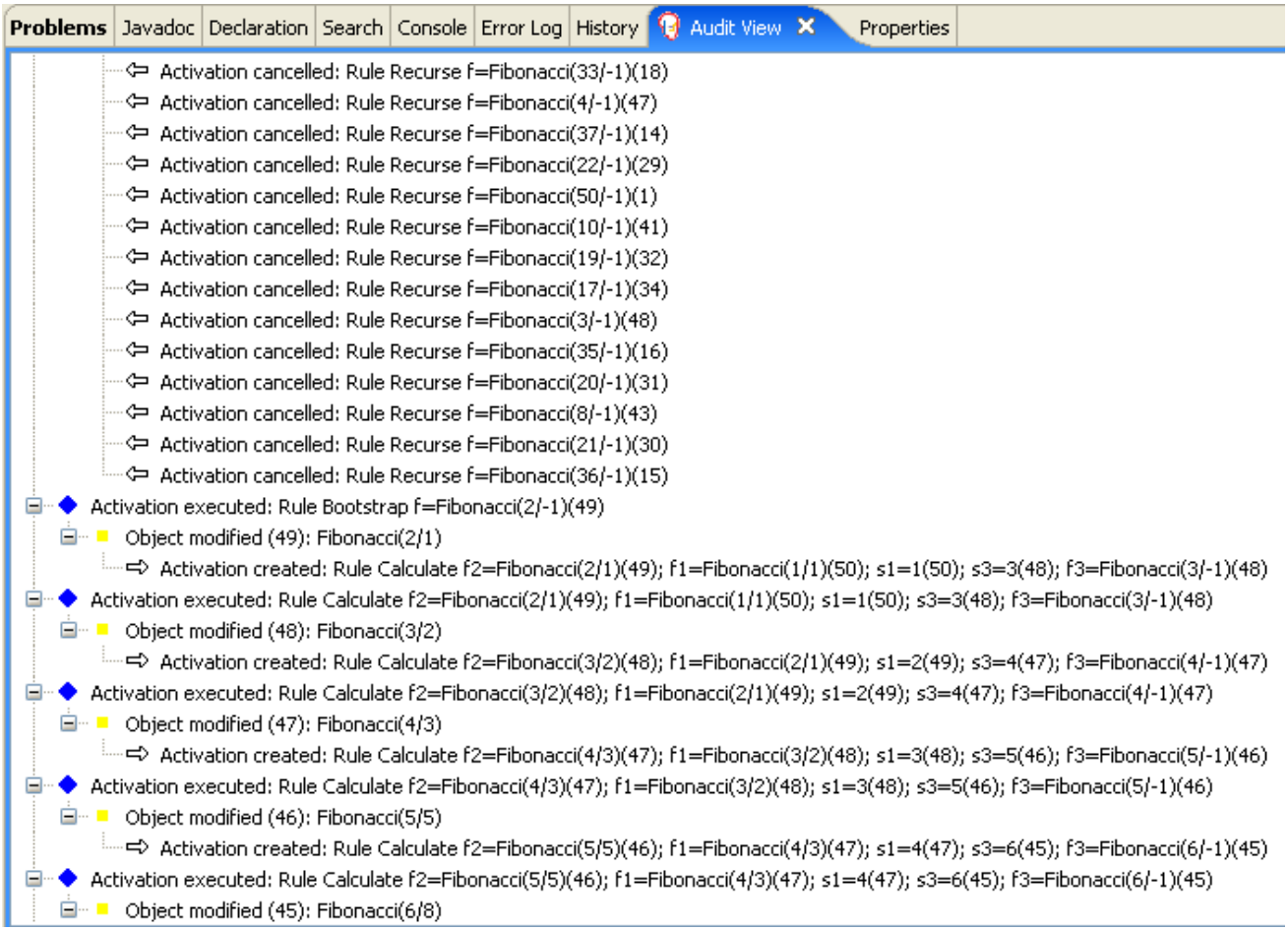
```
rule "Calculate"
  when
    // Bind f1 and s1.
    f1 : Fibonacci( s1 : sequence, value != -1 )
    // Bind f2 and v2, refer to bound variable s1.
    f2 : Fibonacci( sequence == (s1 + 1), v2 : value != -1 )
    // Bind f3 and s3, alternative reference of f2.sequence.
    f3 : Fibonacci( s3 : sequence == (f2.sequence + 1 ), value == -1 )
  then
    // Note the various referencing techniques.
    modify ( f3 ) { value = f1.value + v2 };
    System.out.println( s3 + " == " + f3.value );
end
```

modify 语句更新绑定到 f3 的 Fibonacci 对象的值。这意味着，您现在有一个没有等于 -1 的新

Fibonacci 对象，它允许 "Calculate" 规则重新匹配并计算下一个 Fibonacci 号。

IDE 的 debug 视图或 Audit View 显示触发最后 "Bootstrap" 规则如何修改 Fibonacci 对象，启用 "Calculate" 规则以匹配，然后修改另一个 Fibonacci 对象，以便重新匹配 "Calculate" 规则。此过程将继续，直到为所有 Fibonacci 对象设置了值。

图 89.10. Audit 视图中的规则



89.5. 定价示例决策（决策表）

定价示例决策集演示了如何使用电子表格决策表来以表格形式计算保险政策的零售成本，而不是直接在 DRL 文件中计算。

以下是定价示例概述：

- *名称：decisiontable*
- *主类：org.drools.examples.decisiontable.PricingRuleDTExample（在 src/main/java 中）*

- 模块：drools-examples
- 键入：Java 应用程序
- 规则文件：org.drools.examples.decisiontable.ExamplePolicyPricing.xls (src/main/resources)
- 目标：演示电子表格决策表的使用来定义规则

电子表格决策表是 XLS 或 XLSX 电子表格，其中包含以表格格式定义的业务规则。您可以包括带有独立红帽流程自动化管理器项目的电子表格决策表，或者在 Business Central 中将其上传到项目。决策表中的每一行都是一个规则，每个列都是条件、操作或其他规则属性。在创建并上传您的决定表至 Red Hat Process Automation Manager 项目中后，您定义的规则会像所有其他规则资产一样编译到 Drools 规则语言(DRL)规则中。

定价示例的目的是提供一组业务规则来计算基础价格和适用于适用于特定类型的保险政策的 car 驱动程序折扣。驱动程序的年龄和历史以及策略类型，所有为计算基本高级的贡献，其他规则计算驱动程序可能有资格获得的潜在折扣。

要执行示例，请运行 org.drools.examples.decisiontable.PricingRuleDTExample 类，作为 IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

```
Cheapest possible
BASE PRICE IS: 120
DISCOUNT IS: 20
```

执行示例的代码遵循典型的执行模式：加载规则，并插入事实，并且创建一个无状态的 KIE 会话。本例中的区别在于，规则在 ExamplePolicyPricing.xls 文件中定义，而不是 DRL 文件或其他来源。使用模板和 DRL 规则，电子表格文件被加载到决策引擎中。

电子表格决策表设置

ExamplePolicyPricing.xls spreadsheet 在第一个标签页中包含两个 decision 表：

- ***基本定价**规则*
- ***促销折扣**规则*

***当**电子表格示例演示**时，您只能使用**电子表格中的第一个**标签页**来**创建决策表，但多个表可以在一个**标签页**内。决策表不一定遵循自**顶逻辑，但更是捕**获规则**生成的数据的一种方式。**规则**的**评估不一定按**给定**顺序使用，因**为该**决策引擎的所有普通原理仍然适用。因此，您可以在**电子表格的同一个**标签页**中有多个路由表。*

***决策表通**过**相**应**的**规则**模板文件 BasePricing.drt 和 promotionPricing .drt 执行。这些模板文件通**过其模板参数来引用决策表，并直接引用 decision 表中条件和操作的各种**标头**。*

BasePricing.drt** 规则**模板文件

```
template header
age[]
profile
priorClaims
policyType
base
reason

package org.drools.examples.decisiontable;

template "Pricing bracket"
age
policyType
base

rule "Pricing bracket_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}
        , priorClaims == "@{priorClaims}"
        , locationRiskProfile == "@{profile}"
    )
    policy: Policy(type == "@{policyType}")
  then
    policy.setBasePrice(@{base});
    System.out.println("@{reason}");
end
end template
```

促销.drt 规则模板文件

```
template header
age[]
priorClaims
policyType
discount

package org.drools.examples.decisiontable;

template "discounts"
age
priorClaims
policyType
discount

rule "Discounts_@{row.rowNumber}"
  when
    Driver(age >= @{age0}, age <= @{age1}, priorClaims == "@{priorClaims}")
    policy: Policy(type == "@{policyType}")
  then
    policy.applyDiscount(@{discount});
end
end template
```

规则是通过 **KIE Session DTableWithTemplateKB 的 kmodule.xml 参考执行，它特别提到了 ExamplePolicyPricing.xls spreadsheet，并且需要成功执行规则。这个执行方法可让您将规则作为独立单元（如本例中）执行，或者将规则包括在打包的 JAR(KJAR)文件中，以便电子表格与要执行的规则一起打包。**

要成功执行规则和电子表格，需要 kmodule.xml 文件的以下部分：

```
  <kbase name="DecisionTableKB" packages="org.drools.examples.decisiontable">
    <ksession name="DecisionTableKS" type="stateless"/>
  </kbase>

  <kbase name="DTableWithTemplateKB" packages="org.drools.examples.decisiontable-template">
    <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
                  template="org/drools/examples/decisiontable-template/BasePricing.drt"
                  row="3" col="3"/>
    <ruleTemplate dtable="org/drools/examples/decisiontable-
template/ExamplePolicyPricingTemplateData.xls"
                  template="org/drools/examples/decisiontable-template/PromotionalPricing.drt"
```

```
                row="18" col="3"/>
    <ksession name="DTableWithTemplateKS"/>
  </kbase>
```

作为使用规则模板文件执行决策表的替代方法，您可以使用 DecisionTableConfiguration 对象，并将输入电子表格指定为输入类型，如 DecisionTableInputType.xls ：

```
DecisionTableConfiguration dtableconfiguration =
   KnowledgeBuilderFactory.newDecisionTableConfiguration();
      dtableconfiguration.setInputType( DecisionTableInputType.XLS );

      KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();

      Resource xlsRes = ResourceFactory.newClassPathResource(
"ExamplePolicyPricing.xls",
                                          getClass() );
      kbuilder.add( xlsRes,
               ResourceType.DTABLE,
               dtableconfiguration );
```

定价示例使用两种事实类型：

- 驱动
- 策略.

这个示例在相应的 Java 类 Driver.java 和 Policy.java 中同时设置了默认值。Driver 旧是 30 年，没有以前的声明，目前存在 LOW 的风险配置集。为应用该驱动程序的 Policy 是 COMPREHENSIVE。

在任何决策表中，每行都被视为不同的规则，每个列都是条件或一个操作。每行都在决定表中评估，除非在执行时清除表格。

决策表电子表格（XLS 或 XLSX）需要两个定义规则数据的关键区域：

- RuleSet 区域
- 规则 区域

电子表格的 RuleSet 区域定义了您要对同一软件包中的所有规则（不仅仅是电子表格）全局应用的元素，如规则集名称或通用规则属性。RuleTable 区域定义了实际规则（箭头）以及条件、操作和其他规则属性（列），这些属性构成指定规则集中的规则表。表格电子表格可以包含多个可 规则 的区域，但只能包含一个 RuleSet 区域。

图 89.11. 决策表配置

	C	D	E	F	G	H
	RuleSet	org.drools.examples.decisiontable				
	Notes	This decision table is for working out some basic prices and pretending actuaries don't exist				
	RuleTable Pricing bracket					
	CONDITION	CONDITION	CONDITION	CONDITION	ACTION	ACTION
	Driver			policy: Policy		
	age >= $1, age <= $2	locationRiskProfile	priorClaims	type	policy.setBasePrice($param);	System.out.println("$param");
	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason

RuleTable 区域还定义了规则属性应用到的对象，在本例中为 Driver 和 Policy，后面接对象的限制。例如，定义 Age Bracket 列的 Driver 对象约束为 age >= $1, age <= $2，其中以逗号分隔的范围在表列中值中定义，如 18,24。

基本定价规则

定价示例中的基本价格规则 表评估驱动程序的年龄、风险配置集、声明次数和策略类型，并根据这些条件生成策略的基本价格。

图 89.12. 基本价格计算

	B	C	D	E	F	G	H
9	Base pricing rules	Age Bracket	Location risk profile	Number of prior claims	Policy type applying for	Base $ AUD	Record Reason
10	Young safe package	18, 24	LOW	1	COMPREHENSIVE	450	
11			MED		FIRE_THEFT	200	Priors not relevant
12			MED	0	COMPREHENSIVE	300	
13			LOW		FIRE_THEFT	150	
14			LOW	0	COMPREHENSIVE	150	Safe driver discount
15	Young risk	18,24	MED	1	COMPREHENSIVE	700	
16		18,24	HIGH	0	COMPREHENSIVE	700	Location risk
17		18,24	HIGH		FIRE_THEFT	550	Location risk
18	Mature drivers	25,30		0	COMPREHENSIVE	120	Cheapest possible
19		25,30		1	COMPREHENSIVE	300	
20		25,30		2	COMPREHENSIVE	590	
21		25,35		3	THIRD_PARTY	800	High risk

Driver 属性在下表中定义：

- age Bracket ： age bracket 为条件 年龄 >=$1, age <=$2 的定义，它定义了驱动程序年龄的条件界限。此条件列突出显示了 $1 和 $2 的使用，在电子表格中用逗号分隔。您可以将这些值写为 18、24 或 18，24 或 18，这两种格式都可以执行业务规则。

- 位置风险配置集: risk 配置集是一个字符串，该示例程序始终以 LOW 形式传递，但可以改为反映 MED 或 HIGH 。

- 之前声明 的数量：声明的数量定义为一个整数，条件列必须完全等于触发操作。该值不是范围，仅完全匹配。

决策表的 Policy 可在条件和规则操作和规则操作中使用，并在下表列中定义属性：

- 策略类型适用于 ：策略类型是以字符串形式传递的条件：COMPREHENSIVE、FIRE_THEFT 或 THIRD_PARTY。

- Base $ AUD: basePrice 作为 ACTION，它通过约束 策略.setBasePrice($param)设置价格；基于与这个值对应的电子表格单元。当您为这个决定表执行对应的 DRL 规则时，规则的 然后 部分针对与事实匹配的 true 条件执行这个 action 语句，并将基本价格设置为对应的值。

- Record Reason ：当规则成功执行时，此操作会在 System.out 控制台上生成一个输出信息，它反映了哪些规则触发。之后会在应用程序中捕获并打印。

这个示例也使用左侧的第一列来分类规则。此列仅用于注释，对规则执行没有影响。

促销折扣规则

定价 示例中的促销折扣规则 表格评估了驱动程序的年龄、之前声明和策略类型，以便根据保险政策的价格生成可能的折扣。

图 89.13. 折扣计算

29	Promotional discount rules	Age Bracket	Number of prior claims	Policy type applying for	Discount %
30	Rewards for safe drivers	18,24	0	COMPREHENSIVE	1
31		18,24	0	FIRE_THEFT	2
32		25,30	1	COMPREHENSIVE	5
33		25,30	2	COMPREHENSIVE	1
34		25,30	0	COMPREHENSIVE	20

此决定表包含驱动程序可能有资格享受的折扣条件。与基本价格计算类似，此表评估了驱动程序之前声明的期限、驱动程序之前的声明数量，以及用于确定要应用的 Discount % 速率的策略 类型。例如，如果驱动程序是过去 30 年，没有之前的声明，并且会申请 COMPREHENSIVE 策略，则驱动程序将享有 20% 的折扣。

89.6. PET STORE 示例决策（示例组、全局变量、回调和 GUI 集成）

Pet Store 示例决策集演示了如何在规则中使用平板组和全局变量，以及如何将 Red Hat Process Automation Manager 规则与图形用户界面(GUI)集成，在这种情况下，基于 Swing 的桌面应用程序。这个示例还演示了如何使用回调与正在运行的决策引擎交互，以在运行时根据工作内存中的更改更新 GUI。

以下是 Pet Store 示例的概述：

- 名称 ：petstore
- 主类 ：org.drools.examples.petstore.PetStoreExample （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.petstore.PetStore.drl (src/main/resources)
- 目标 ：演示规则索引组、全局变量、回调和 GUI 集成

在 Pet Store 示例中，示例 PetStoreExample.java 类定义了以下主体类（除了多个类来处理 Swing 事件外）：

- Petstore 包含 main（） 方法。
- PetStoreUI 负责创建和显示基于 Swing 的 GUI。此类包含多个较小的类，主要用于响应各种 GUI 事件，比如鼠标点击的用户。
- TableModel 包含表数据。这个类本质上是扩展 Swing 类 AbstractTableModel 的 JavaBean。
- CheckoutCallback 允许 GUI 与规则交互。
- Ordershow 保留您要购买的项目。
- 购买 存储订购详情以及您要购买的产品。
- 产品是 JavaBean，其中包含可供购买的产品及其价格的详细信息。

本例中的大部分 Java 代码是基于普通 JavaBean 或 Swing。有关 Swing 组件的更多信息，请参阅有关 使用 JFC/Swing 创建 GUI 的 Java 教程。

Pet Store 示例中的规则执行行为

与其他示例决定设置被断言并立即触发，Pet Store 示例不会执行规则，直到根据用户交互收集更多事实。这个示例通过构造器创建的 PetStoreUI 对象来执行规则，接受 Vector 对象 库存 来收集该产品。然后，示例使用 Checkout Callback 类的实例，其中包含之前载入的规则基础。

pet Store KIE 容器和事实执行设置

```
// KieServices is the factory for all KIE services.
KieServices ks = KieServices.Factory.get();

// Create a KIE container on the class path.
KieContainer kc = ks.getKieClasspathContainer();
```

```
// Create the stock.
Vector<Product> stock = new Vector<Product>();
stock.add( new Product( "Gold Fish", 5 ) );
stock.add( new Product( "Fish Tank", 25 ) );
stock.add( new Product( "Fish Food", 2 ) );

// A callback is responsible for populating the working memory and for firing all rules.
PetStoreUI ui = new PetStoreUI( stock,
                                new CheckoutCallback( kc ) );
ui.createAndShowGUI();
```

触发规则的 Java 代码位于 CheckoutCallBack.checkout（） 方法中。当用户在 UI 中点击 Checkout 时触发 此方法。

来自 CheckoutCallBack.checkout（）的规则执行

```
public String checkout(JFrame frame, List<Product> items) {
    Order order = new Order();

    // Iterate through list and add to cart.
    for ( Product p: items ) {
        order.addItem( new Purchase( order, p ) );
    }

    // Add the JFrame to the ApplicationData to allow for user interaction.

    // From the KIE container, a KIE session is created based on
    // its definition and configuration in the META-INF/kmodule.xml file.
    KieSession ksession = kcontainer.newKieSession("PetStoreKS");

    ksession.setGlobal( "frame", frame );
    ksession.setGlobal( "textArea", this.output );

    ksession.insert( new Product( "Gold Fish", 5 ) );
    ksession.insert( new Product( "Fish Tank", 25 ) );
    ksession.insert( new Product( "Fish Food", 2 ) );

    ksession.insert( new Product( "Fish Food Sample", 0 ) );

    ksession.insert( order );

    // Execute rules.
    ksession.fireAllRules();

    // Return the state of the cart
    return order.toString();
}
```

示例代码将两个元素传递给 CheckoutCallBack.checkout（） 方法。一个元素是处理 JFrame Swing 组件，位于 GUI 的底部。第二个元素是顺序项目的列表，它来自 GUI 右上角的表格区域的信息。

for 循环将来自于 GUI 的订购项列表转换为 Order JavaBean，同时包含在文件 PetStoreExample.java 中。

在这种情况下，规则会在无状态 KIE 会话中触发，因为所有数据都存储在 Swing 组件中，并在用户点击 UI 中的 Checkout 之前执行。每次用户点击 Checkout 时，列表的内容都会从 Swing TableModel 移到 KIE 会话工作内存中，然后使用 ksession.fireAllRules（） 方法执行。

在此代码中，对 KieSession 有 9 个调用。其中之一从 KieContainer 创建新的 KieSession （在这个 KieContainer 中通过的示例从 main（） 方法中的 CheckoutCallBack 类传递）。接下来的两个调用通过规则中存放全局变量的两个对象： Swing 文本区域以及用于编写消息的 Swing 帧。并将有关产品的信息插入 KieSession 以及顺序列表中的更多信息。最终的调用是标准 fireAllRules（）。

pet Store 规则文件导入、全局变量和 Java 功能

PetStore.drl 文件包含标准软件包和导入语句，以使规则可以使用各种 Java 类。规则文件还包括用于在规则（定义为 帧 和 textArea ）中使用的 全局变量。全局变量包含对之前由称为 setGlobal（） 方法的 Java 代码传递的 Swing 组件 JFrame 和 JTextArea 组件的引用。与规则中的标准变量不同，在规则触发后，全局变量会保留其在 KIE 会话生命周期中的值。这意味着这些全局变量的内容可以对所有后续规则进行评估。

PetStore.drl 软件包、导入和全局变量

```
package org.drools.examples;

import org.kie.api.runtime.KieRuntime;
import org.drools.examples.petstore.PetStoreExample.Order;
import org.drools.examples.petstore.PetStoreExample.Purchase;
import org.drools.examples.petstore.PetStoreExample.Product;
import java.util.ArrayList;
import javax.swing.JOptionPane;

import javax.swing.JFrame;

global JFrame frame
global javax.swing.JTextArea textArea
```

PetStore.drl 文件还包含两个使用中的规则：

PetStore.drl Java 功能

```
function void doCheckout(JFrame frame, KieRuntime krt) {
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                             "Would you like to checkout?",
                                             "",
                                             JOptionPane.YES_NO_OPTION,
                                             JOptionPane.QUESTION_MESSAGE,
                                             null,
                                             options,
                                             options[0]);

       if (n == 0) {
           krt.getAgenda().getAgendaGroup( "checkout" ).setFocus();
       }
}

function boolean requireTank(JFrame frame, KieRuntime krt, Order order, Product fishTank,
int total) {
        Object[] options = {"Yes",
                            "No"};

        int n = JOptionPane.showOptionDialog(frame,
                                             "Would you like to buy a tank for your " + total + " fish?",
                                             "Purchase Suggestion",
                                             JOptionPane.YES_NO_OPTION,
                                             JOptionPane.QUESTION_MESSAGE,
                                             null,
                                             options,
                                             options[0]);

       System.out.print( "SUGGESTION: Would you like to buy a tank for your "
                         + total + " fish? - " );

       if (n == 0) {
            Purchase purchase = new Purchase( order, fishTank );
            krt.insert( purchase );
            order.addItem( purchase );
            System.out.println( "Yes" );
       } else {
           System.out.println( "No" );
```

```
    }
    return true;
}
```

这两个功能执行以下操作：

- doCheckout（） 显示一个对话框，它要求用户是否被委派或需要签出。如果用户确实如此，则重点设置为 结账 员组，使该组中的规则启用（可能）触发。
- requireTank（） 会显示一个对话框，该对话框要求用户如果她或想要购买财务语。如果用户确实有，则会在工作内存中的订购列表中添加一个新芬兰的 tank 产品。

注意

在本例中，所有规则和功能都位于同一个规则文件中，以提高效率。在生产环境中，您通常将不同文件中的规则和功能分开，或构建静态 Java 方法并使用导入功能导入文件，如导入功能 my.package.name.hello。

pet Store 规则与日程组

Pet Store 示例中的大多数规则使用 table 组来控制规则执行。日程表组允许您对决策引擎日程表进行分区，以便对规则组提供更多执行控制。默认情况下，所有规则均位于 MAIN 日程小组。您可以使用 schedule -group 属性来指定该规则的不同日程表组。

最初，工作内存专注于 MAIN 的日程安排。仅当该组收到相关事项时，即可参与日程表组中的规则。您可以使用方法 setFocus（） 或 rule 属性 auto-focus 来设置焦点。auto-focus 属性允许当规则匹配和激活时，自动为课程安排人员自动给定规则。

Pet Store 示例对规则使用以下日程组：

- "init"
- "评估"

- "显示项目"
- "checkout"

例如，示例规则 "Explode Cart" 使用 "init" 资格将 cart 项目触发并插入到 KIE 会话工作内存中：

规则"Explode Cart"

```
// Insert each item in the shopping cart into the working memory.
rule "Explode Cart"
    agenda-group "init"
    auto-focus true
    salience 10
  when
    $order : Order( grossTotal == -1 )
    $item : Purchase() from $order.items
  then
    insert( $item );
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "show items" ).setFocus();
    kcontext.getKnowledgeRuntime().getAgenda().getAgendaGroup( "evaluate" ).setFocus();
end
```

该规则与所有尚未计算的订单匹配。每个购买项目的执行循环按该顺序排列。

该规则使用与其 schedule 组相关的以下功能：

- 日程表组"init" 定义日程表组的名称。在这种情况下，组中只有一个规则。但是，Java 代码和规则都无法专注于此组，因此它也取决于其触发的机会的 auto-focus 属性。
- auto-focus true 可确保此规则，而作为 schedule group 中的唯一规则，但从 Java 代码调用 fireAllRules（） 时有机会触发。
- kcontext....setFocus（） 将焦点设置为 "show items" 和 "evaluate" 日程表组，支持他们触发的规则。在实践中，您要循环顺序的所有项目，将它们插入到内存中，然后在各个插入后触

发其他规则。

"显示项目" 日程表组仅包含一条规则，"项目方式"对于当前 KIE 会话工作内存顺序的每个购买，规则会根据规则文件中定义的文本区域将详情记录到 GUI 底部的文本区域。

规则"显示方式"

```
rule "Show Items"
    agenda-group "show items"
  when
    $order : Order()
    $p : Purchase( order == $order )
  then
    textArea.append( $p.product + "\n");
end
```

"评估" 日程表组还从 "Explode Cart" 规则获得。此日程表组包含两个规则 ："Free Fish Food Sample" 和 "Suggest Tank"，按该顺序执行。

规则"Free Fish Food Sample"

```
// Free fish food sample when users buy a goldfish if they did not already buy
// fish food and do not already have a fish food sample.
rule "Free Fish Food Sample"
    agenda-group "evaluate" (1)
  when
    $order : Order()
    not ( $p : Product( name == "Fish Food") && Purchase( product == $p ) ) (2)
    not ( $p : Product( name == "Fish Food Sample") && Purchase( product == $p ) ) (3)
    exists ( $p : Product( name == "Gold Fish") && Purchase( product == $p ) ) (4)
    $fishFoodSample : Product( name == "Fish Food Sample" );
  then
    System.out.println( "Adding free Fish Food Sample to cart" );
    purchase = new Purchase($order, $fishFoodSample);
    insert( purchase );
    $order.addItem( purchase );
end
```

只有在以下条件都满足时，才会触发 "Free Fish Food Sample" 规则：

1. 日程组 "评估" 在规则执行中评估。
2. 用户还没有 fish food。
3. 用户还没有一个自由的页式示例。
4. 用户按数字顺序使用 goldfish。

如果顺序事实满足所有这些要求，则创建一个新产品(Fish Food Sample)并添加到工作内存中的顺序中。

规则"Suggest Tank"

```
// Suggest a fish tank if users buy more than five goldfish and
// do not already have a tank.
rule "Suggest Tank"
    agenda-group "evaluate"
  when
    $order : Order()
    not ( $p : Product( name == "Fish Tank") && Purchase( product == $p ) ) 1
    ArrayList( $total : size > 5 ) from collect( Purchase( product.name == "Gold Fish" ) ) 2
    $fishTank : Product( name == "Fish Tank" )
  then
    requireTank(frame, kcontext.getKieRuntime(), $order, $fishTank, $total);
end
```

只有在以下条件满足时才触发 "Suggest Tank" 规则：

1

用户没有按顺序的ish tank。

2

用户按以下顺序排列，有五种种。

当规则触发时，它会调用规则 文件中定义的 requireTank（） 函数。此函数会显示一个对话框，询问用户是否是她或他想要购买玻里程表。如果用户确实有，则会在工作内存中的订购列表中添加一个新芬兰的 tank 产品。当规则调用 requireTank（） 函数时，该规则会传递帧的全局变量，以便该函数能够处理 Swing GUI。

Pet Store 示例中的 "do checkout" 规则没有日程表组，没有条件，因此该规则始终执行并被视为默认 MAIN 更新组的一部分。

规则 "do checkout"

```
rule "do checkout"
  when
  then
    doCheckout(frame, kcontext.getKieRuntime());
end
```

当规则触发时，它会调用规则 文件中定义的 doCheckout（） 函数。此函数显示一个对话框，询问用户是否她或他想要签出。如果用户确实如此，则重点设置为 结账 员组，使该组中的规则启用（可能）触发。当规则调用 doCheckout（） 函数时，该规则会传递 帧 全局变量，以便该函数能够处理 Swing GUI。

注意

本例还演示了故障排除技术（如果结果未按预期一样执行）：您可以从规则的 when 语句中删除条件，并在 then 语句中测试操作以验证操作是否正确执行。

"检查" 日程表组包含三个规则，用于处理订单并申请任何折扣："总额"、"应用 5% 的折扣" 和 "应用 10% 的折扣 "。

规则"总额"、"Apply 5% 的折扣"和"Apply 10% 的折扣"

```
rule "Gross Total"
    agenda-group "checkout"
  when
    $order : Order( grossTotal == -1)
    Number( total : doubleValue ) from accumulate( Purchase( $price : product.price ),
                                                   sum( $price ) )
  then
    modify( $order ) { grossTotal = total }
    textArea.append( "\ngross total=" + total + "\n" );
end

rule "Apply 5% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 10 && < 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.95;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end

rule "Apply 10% Discount"
    agenda-group "checkout"
  when
    $order : Order( grossTotal >= 20 )
  then
    $order.discountedTotal = $order.grossTotal * 0.90;
    textArea.append( "discountedTotal total=" + $order.discountedTotal + "\n" );
end
```

如果用户还没有计算总额的单位，Grossing Total accate the product 总计（将总值放在 KIE 会话中），并使用 textArea global 变量通过 Swing JTextArea 显示它。

如果总额介于 10 到 20 左右，则 "Apply 5% discount" 规则计算总折扣，将其添加到 KIE 会话中，并将其显示在文本区域中。

如果总总额不低于 20，"Apply 10% discount" 规则计算总额，将其添加到 KIE 会话中，并在文本区域中显示它。

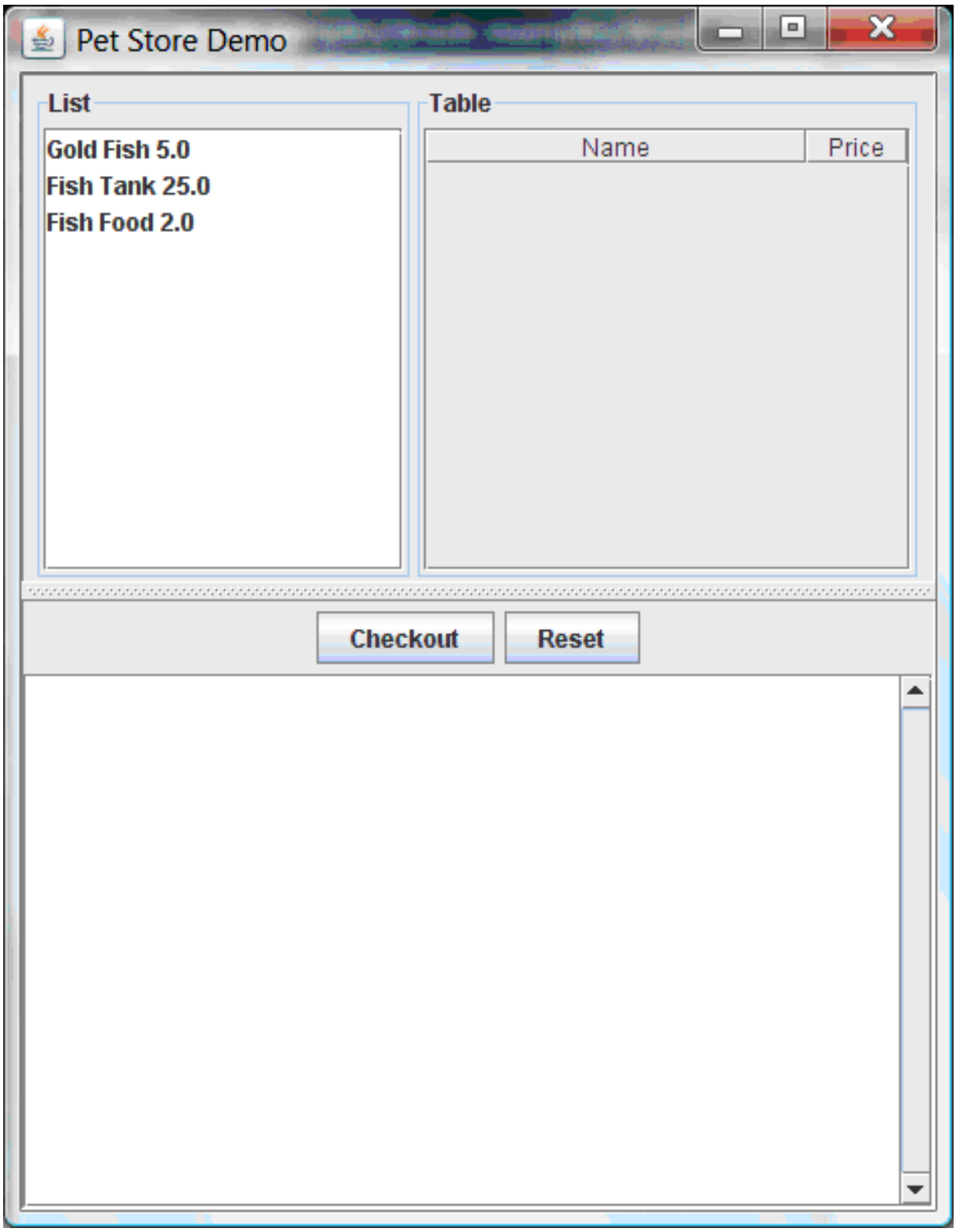
pet Store 的执行示例

与其他 Red Hat Process Automation Manager 决策示例类似，您可以通过运行

org.drools.examples.petstore.PetStoreExle 类作为 IDE 中的 Java 应用程序来执行 Pet Store 示例。

当您执行 Pet Store 示例时，会显示 Pet Store Demo GUI 窗口。此窗口显示可用产品列表（左下）、选定产品的空列表（右、Checkout 和 Reset 按钮(middle)）和一个空系统消息区域(bottom)。

图 89.14. 启动后 pet Store 示例 GUI



本例中发生以下事件来建立此执行行为：

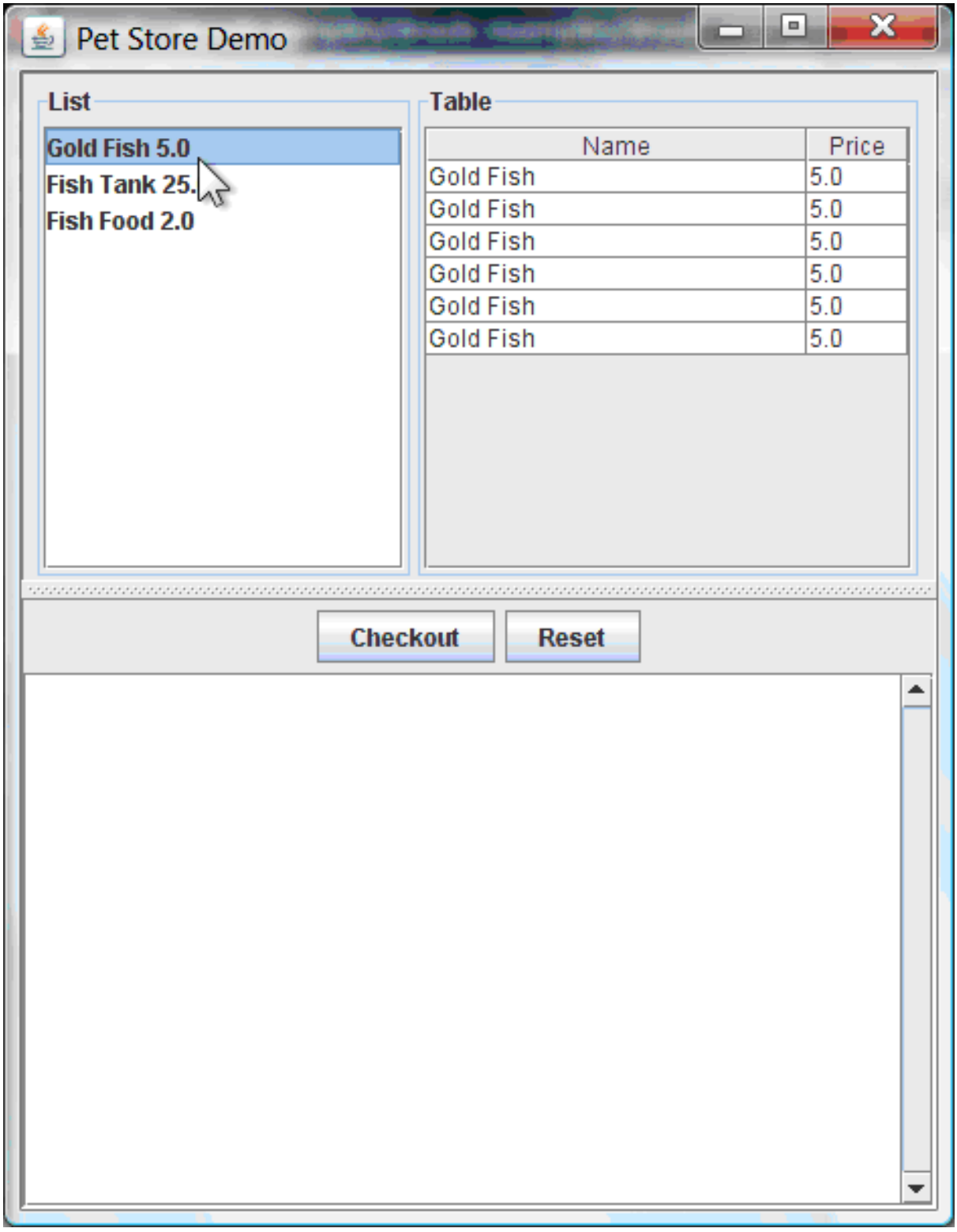
1. main（） 方法已运行并加载规则基础，但尚未触发规则。目前，这是与运行的规则连接中的唯一代码。

2. 新的 PetStoreUI 对象已创建，并为规则基础赋予句柄，供以后使用。

3. 各种 Swing 组件已执行其功能，并显示初始 UI 屏幕并等待用户输入。

您可以从列表中点各种产品以浏览 UI 设置：

图 89.15. 探索 Pet Store 示例 GUI



还没有触发规则代码。UI 使用 Swing 代码来检测用户鼠标点击并将所选产品添加到用于 UI 右上角显示的 TableModel 对象中。本例演示了 Model-View-Controller 设计模式。

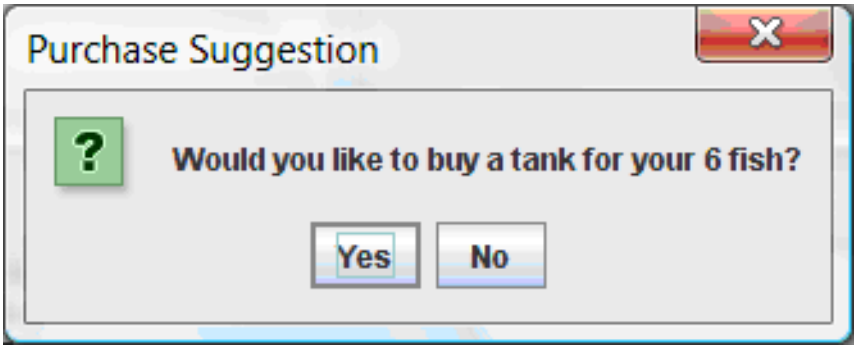
当您点 Checkout 时，规则会按以下方式触发：

1. *CheckOutCallBack.checkout（）方法由 Swing 类调用（事件），等待用户点击 Checkout。这会将 TableModel 对象（UI 右上角）中的数据插入到 KIE 会话工作内存中。然*

后，该方法会触发规则。

2. "Explode Cart" 规则是第一个触发，auto-focus 属性设为 true。通过 cart 中的所有产品的规则循环，确保产品处于工作内存中，然后提供 "显示项目"和" 评估" 日程表组来触发的选项。这些组中的规则将 cart 的内容添加到文本区域(bottom)中，评估您是否有资格获得免费的放大分，并确定是否希望购买欺诈。

图 89.16. 芬兰 tank 资格



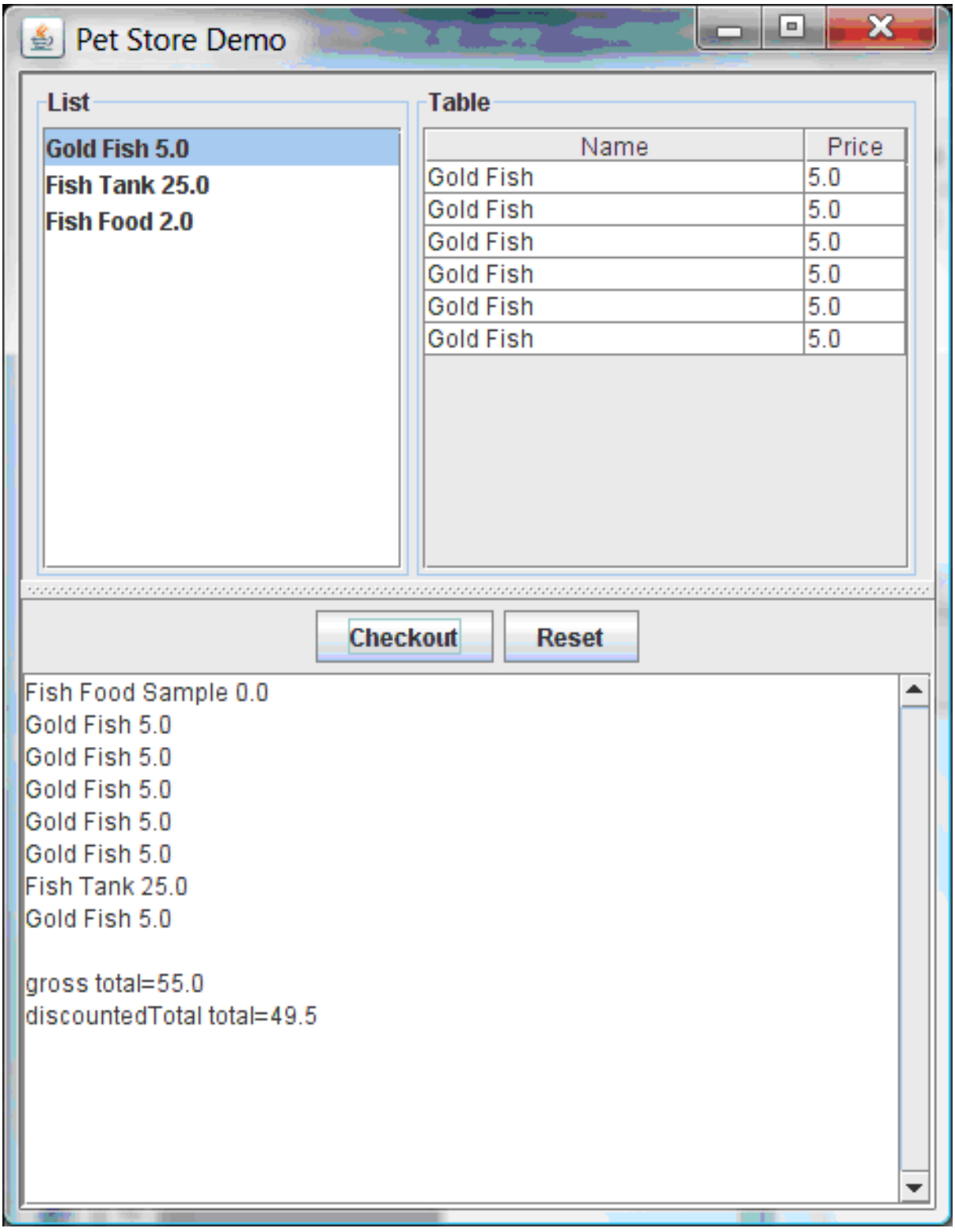
3. "do checkout" 规则是下一个需要触发的，因为其他任何日程组当前没有关注，因为它是默认的 MAIN 日程小组的一部分。这个规则始终调用 doCheckout（） 函数，它会询问您是否要签出。

4. doCheckout（） 函数将焦点设置为 "checkout" Table 组，该组中的规则给出了要触发的选项。

5. "检查" 日程表组中的规则显示车队的内容，并应用相应的折扣。

6. 然后，等待用户输入可选择更多产品（并导致规则再次触发），或关闭 UI。

*图 **89.17. 所有规则触发后的 pet Store 示例 GUI***



您可以添加更多 System.out 调用来在 IDE 控制台中演示此事件流：

IDE 控制台中的 system.out 输出

```
Adding free Fish Food Sample to cart
SUGGESTION: Would you like to buy a tank for your 6 fish? - Yes
```

89.7. HONEST POLITICIAN 示例决策（维护与健保）

Honest Politician 示例决策集展示了在规则中采用逻辑插入和使用 salience 的真实概念。

以下是 Honest Politician 示例概述：

- Name : honest Hician
- 主类:org.drools.examples.honest Hian.HonestPoliticianExample （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.honest Hian.HonestPolitician.drl (src/main/resources)
- 目标 ：根据事实的逻辑插入以及规则中的使用方法论证了真实维护的概念

Honest Politician 示例的基本内部是对象只能存在，而声明是 true。规则结果可以逻辑地插入带有 insertLogical（） 方法的对象。这意味着，在 KIE 会话工作内存中，只要逻辑插入的规则保持在 KIE 会话工作内存中。当规则不再为 true 时，对象会被自动重试。

在本例中，规则执行会导致一组自治亚更改，以防出现损坏企业的结果。评估完各个文所评估时，这些属性从 honesty 属性设置为 true，但有一个规则触发，使自治亚体不再是 honest。当它们的状态从 honest 改为 dishonest 时，它们会从工作内存中删除。该规则可通知决策引擎如何优先考虑为它们定义的任何规则，否则使用默认 salience 值 0。在激活队列中排序时，具有较高优先级的规则会被赋予更高的优先级。

Politician 和 Hope 类

示例中的示例类 Politician 为 honest Leaician 进行了配置。Politician 类由 String 项 名称和 一个布尔值项目组成：

Politician 类

```
public class Politician {
    private String name;
    private boolean honest;
    ...
}
```

Hope 类确定是否存在 Hope 对象。这个类没有有意义的成员，但只要 society 希望就出现在正常工作的内存中。

希望课程

```
public class Hope {

    public Hope() {

    }
  }
```

shardician honesty 的规则定义

在 Honest Politician 示例中，当工作内存中至少存在一个 honest shardician 时，"We have an honest Politician" 规则逻辑地插入一个新的 Hope 对象。当所有协调员都变得不开时，Hope 对象会被自动重新处理。此规则具有值为 10 的 salience 属性，以确保它在任何其他规则之前触发，因为 在该阶段 "Hope is Dead" 规则为 true。

规则"我们有一个最哈佛尼亚"

```
rule "We have an honest Politician"
    salience 10
```

```
  when
    exists( Politician( honest == true ) )
  then
    insertLogical( new Hope() );
end
```

当 Hope 对象存在时，"Hope Lives" 规则与 and fires 匹配和触发。这个规则也具有 10 相似的值，以便它的优先级高于 Honest" 规则。

规则"停止实时"

```
rule "Hope Lives"
    salience 10
  when
    exists( Hope() )
  then
    System.out.println("Hurrah!!! Democracy Lives");
end
```

最初，存在四个最哈尔地，因此此规则有 4 个激活，它们发生冲突。每个规则依次触发，破坏每个工作员，从而使它们不再频繁。当所有四个协调器都已损坏时，没有 Hoticians 含有属性 honest == true。规则 "We have an hest Politician" 不再是 true，它的逻辑插入的对象（由于 是新 Hope（）的最后执行）会自动重新处理。

规则"利用 Honest"

```
rule "Corrupt the Honest"
  when
    politician : Politician( honest == true )
    exists( Hope() )
  then
    System.out.println( "I'm an evil corporation and I have corrupted " + politician.getName() );
    modify ( politician ) { honest = false };
end
```

当 Hope 对象自动经由真相维护系统时，没有 应用到 Hope 的条件元素不再为 true， "Hope is Dead" 规则匹配并触发。

规则"Hope is Dead"

```
rule "Hope is Dead"
  when
    not( Hope() )
  then
    System.out.println( "We are all Doomed!!! Democracy is Dead" );
end
```

执行和审核跟踪示例

在 HonestPoliticianExample.java 类中，四个协调员并将 Honest state 设置为 true 以针对定义的业务规则插入评估：

HonestPoliticianExample.java 类执行

```
public static void execute( KieContainer kc ) {
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        ksession.fireAllRules();

        ksession.dispose();
    }
```

要执行这个示例，请运行 org.drools.examples.honest Hician.HonestPoliticianExample 类，作为

IDE 中的 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的执行输出

```
Hurrah!!! Democracy Lives
I'm an evil corporation and I have corrupted President of Umpa Lumpa
I'm an evil corporation and I have corrupted Prime Minster of Cheeseland
I'm an evil corporation and I have corrupted Tsar of Pringapopaloo
I'm an evil corporation and I have corrupted Omnipotence Om
We are all Doomed!!! Democracy is Dead
```

输出显示，虽然至少有一个最哈尔、演示文稿。但是，由于每个人都受到一些公司损坏，所有自治亚人都变得不满，因此演示不容忽视。

为了更好地理解本例的执行流，您可以修改 HonestPoliticianExample.java 类，使其包含 DebugRuleRuntimeEventListener 侦听器以及审计日志记录器来查看执行详情：

HonestPoliticianExample.java 类，带有审计日志记录器

```
package org.drools.examples.honestpolitician;

import org.kie.api.KieServices;
import org.kie.api.event.rule.DebugAgendaEventListener; 1
import org.kie.api.event.rule.DebugRuleRuntimeEventListener;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public class HonestPoliticianExample {

  /**
   * @param args
   */
  public static void main(final String[] args) {
   KieServices ks = KieServices.Factory.get(); 2
   //ks = KieServices.Factory.get();
     KieContainer kc = KieServices.Factory.get().getKieClasspathContainer();
     System.out.println(kc.verify().getMessages().toString());
```

```
        //execute( kc );
        execute( ks, kc); 3
    }

    public static void execute( KieServices ks, KieContainer kc ) { 4
        KieSession ksession = kc.newKieSession("HonestPoliticianKS");

        final Politician p1 = new Politician( "President of Umpa Lumpa", true );
        final Politician p2 = new Politician( "Prime Minster of Cheeseland", true );
        final Politician p3 = new Politician( "Tsar of Pringapopaloo", true );
        final Politician p4 = new Politician( "Omnipotence Om", true );

        ksession.insert( p1 );
        ksession.insert( p2 );
        ksession.insert( p3 );
        ksession.insert( p4 );

        // The application can also setup listeners 5
        ksession.addEventListener( new DebugAgendaEventListener() );
        ksession.addEventListener( new DebugRuleRuntimeEventListener() );

        // Set up a file-based audit logger.
        ks.getLoggers().newFileLogger( ksession, "./target/honestpolitician" ); 6

        ksession.fireAllRules();

        ksession.dispose();
    }

}
```

1

向导入处理 DebugAgendaEventListener 和 DebugRuleRuntimeEventListener的软件包添加

2

创建一个 KieServices Factory 和 ks 元素来生成日志，因为此审计日志无法在 KieContainer 级别提供

3

修改 执行 方法以使用 KieServices 和 KieContainer

4

除了 KieContainer外，修改 执行 方法以传递 KieServices

5

创建监听程序

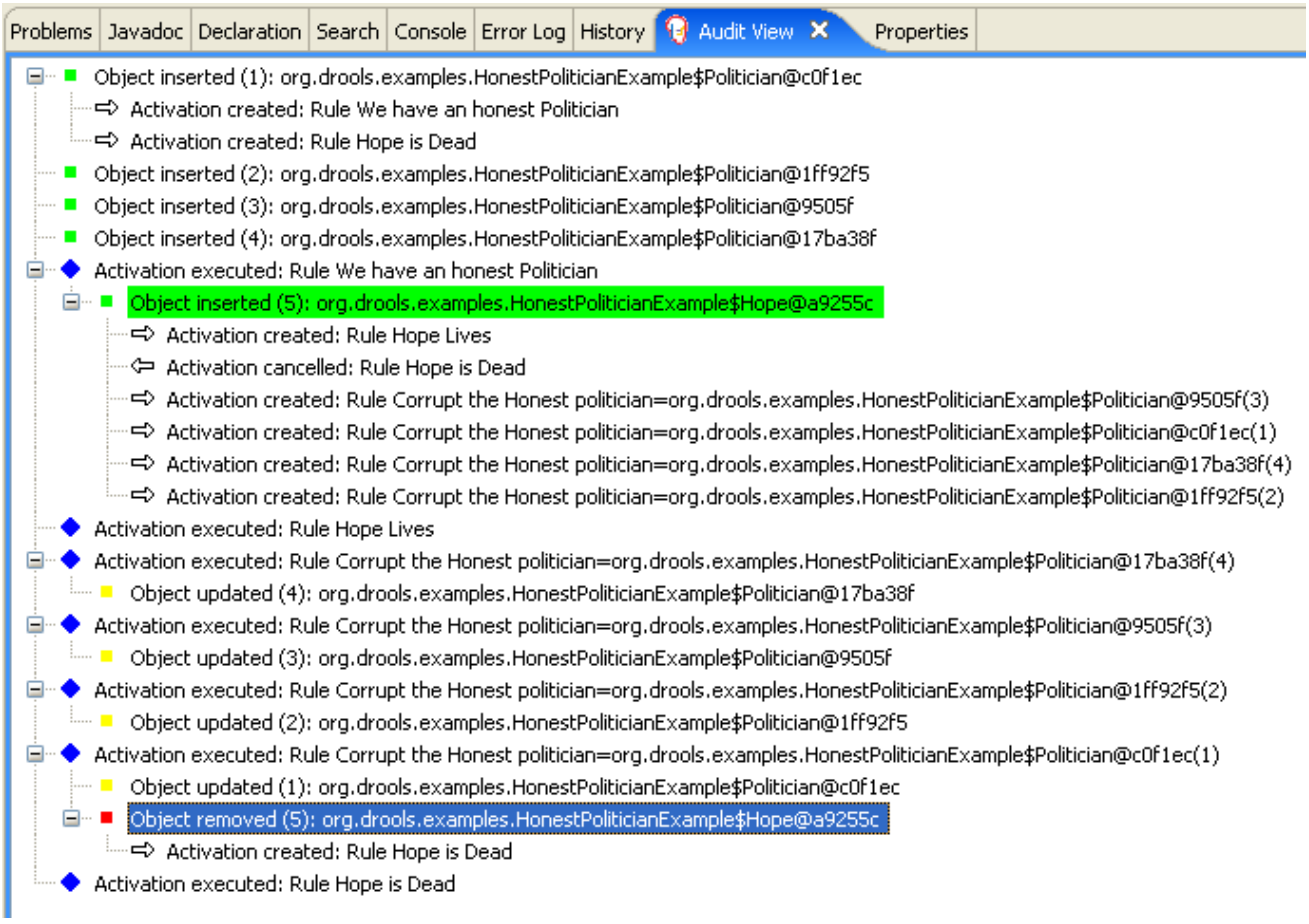
6

在执行规则后构建可传递给 debug 视图或 Audit View 或 IDE 的日志

当使用这个修改的日志功能运行 Honest Politician 时，您可以将审计日志文件从 target/honest Hician.log 加载到 IDE 调试视图或 Audit View （例如，在一些 IDE 中的 Window → Show View 中）。

在本例中，审计视图显示执行流、插入和重新操作，如示例类和规则中定义：

图 89.18. Honest Politician 示例审计视图



当插入第一个协调程序时，会发生两个激活。规则 "We have honest Politician" 只激活第一个插入了 Handomician 的时间，因为它使用 exists 条件元素，即在插入至少一个自治器时匹配。规则 "Hope is Dead" 在此阶段也被激活，因为 Hope 对象尚未插入。规则 "我们首先触发了 honest Politician"，因为它具有高于规则 "Hope is Dead" 的值，并插入 Hope 对象（以绿色高亮显示）。Hope 对象插入对象可激活规则 "Hope Lives"，并停用规则 "Hope is Dead "。insertion 还激活每个插入了 honest APPician 的规则 "Corrupt the Honest"。执行规则 "Hope Lives" 并打印 "Hurrah!!Democracy Lives"。

接下来，对于每个人来说，规则 "Corrupt the Honest" 触发，打印 "I't an evil 企业和 I have corrupted X"，其中 X 是 Hotician 的名称，并将 otician honesty 值修改为 false。当最后的 honest 此目录被破坏时，Hope 将自动由真相维护系统（以蓝色突出显示）进行重新处理。绿色突出显示的区域显示了当前所选蓝色突出显示区域的来源。更改了 Hope 事实后，规则 "Hope is dead" 触发后，打印 "We 都是 Doomed!!Democracy 是 Dead"。

89.8. SUDOKU 示例决策（COMPLEX PATTERN MATCHING、回调和 GUI 集成）

Sudoku 示例决策基于流行数字 puzzle Sudoku 的示例决策，演示了如何在 Red Hat Process Automation Manager 中使用规则，以根据各种限制在大型潜在解决方案空间中找到解决方案。这个示例还演示了如何将 Red Hat Process Automation Manager 规则集成到图形用户界面(GUI)中，在这种情况下，基于 Swing 的桌面应用程序，以及如何使用回调与正在运行的决策引擎进行交互，以在运行时根据工作内存更改更新 GUI。

以下是 Sudoku 示例的概述：

- 名称 ：sudoku
- 主类 ：org.drools.examples.sudoku.SudokuExample （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.sudoku.*.drl （在 src/main/resources）
- 目标 ：演示复杂的模式匹配、问题解决、回调和 GUI 集成

Sudoku 是基于逻辑的数字放置 puzzle。目标是填充 9x9 网格，以便每个列、每个列、每个行以及九个 3x3 区域均包含来自 1 到 9 次的数字。puzzle 设定者提供部分完成的网格，而 puzzle solver 的任务是利用这些限制完成网格。

解决问题的一般策略是，当您插入新数字时，它必须在其特定 3x3 区域、行和列内唯一。这个 Sudoku 示例决策设置使用红帽流程自动化管理器规则来解决 Sudoku puzzles 的问题，并试图解决包含

无效条目的缺陷的 puzzles。

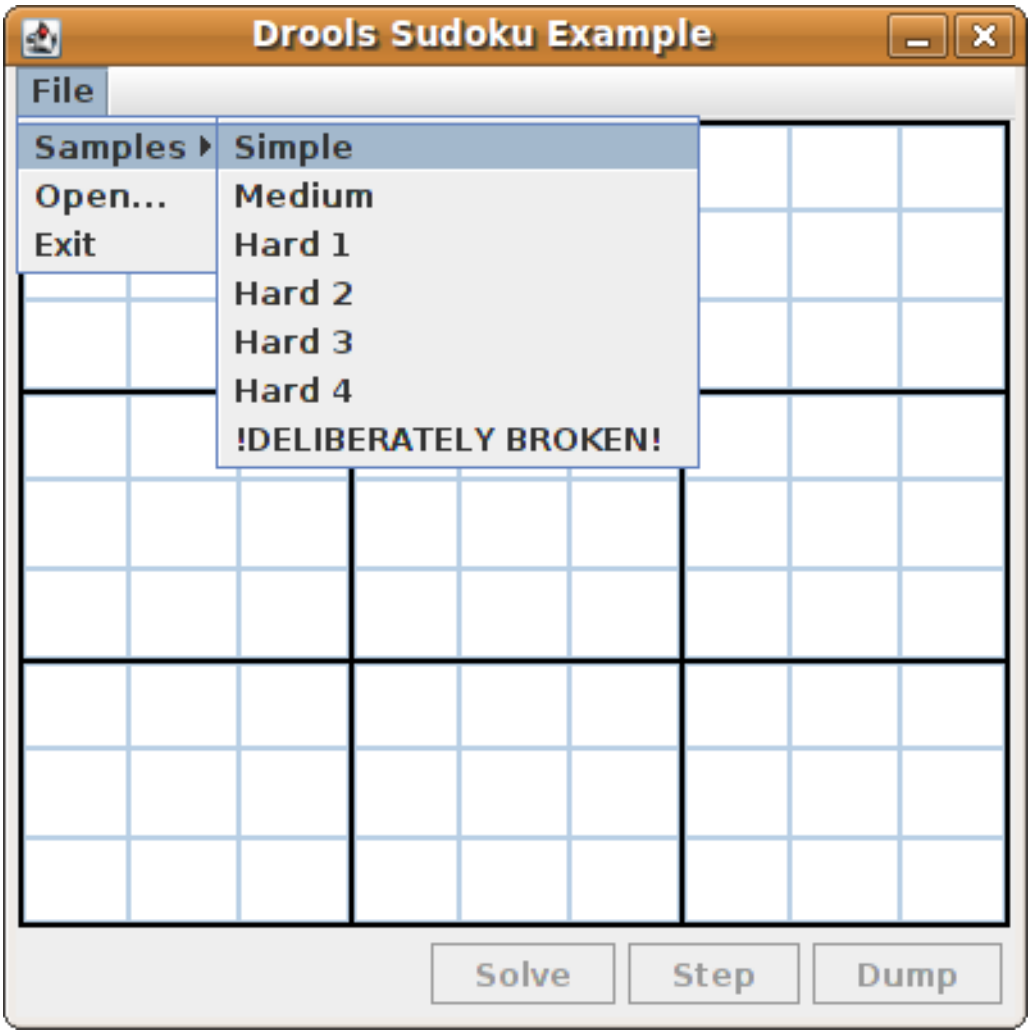
Sudoku 示例执行和交互

与其他红帽流程自动化管理器决策示例类似，您可以通过运行 org.drools.examples.sudoku.SudokuExample 类作为 IDE 中的 Java 应用程序来执行 SudokuExample 类。

当您执行 Sudoku 示例时，会出现 Drools Sudoku Example GUI 窗口。此窗口包含空网格，但该程序随附了存储于内部的各种网格，您可以加载和解决。

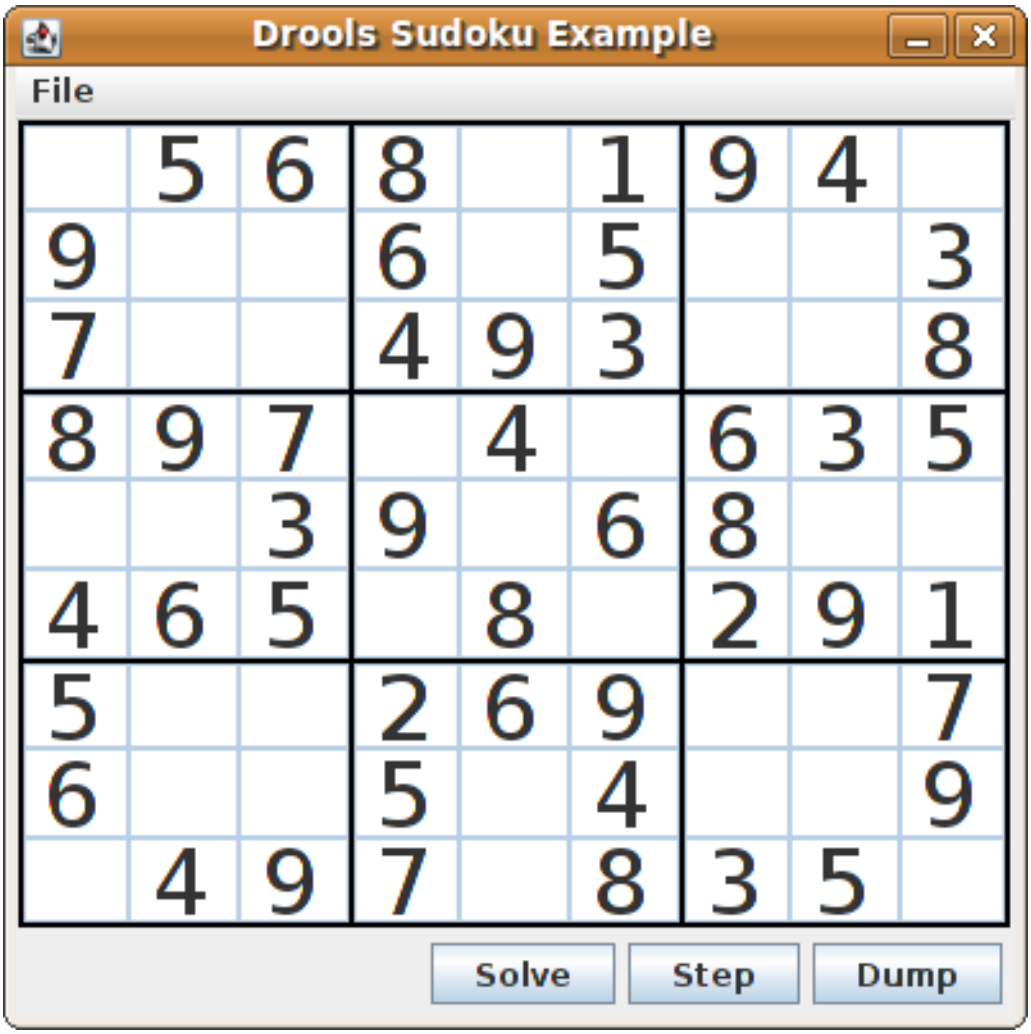
点 File → Samples → Simple 加载其中一个示例。请注意，在加载网格前，所有按钮都被禁用。

图 89.19. 启动时的 Sudoku 示例 GUI



加载 简单示例 时，网格会根据点的初始状态进行填写。

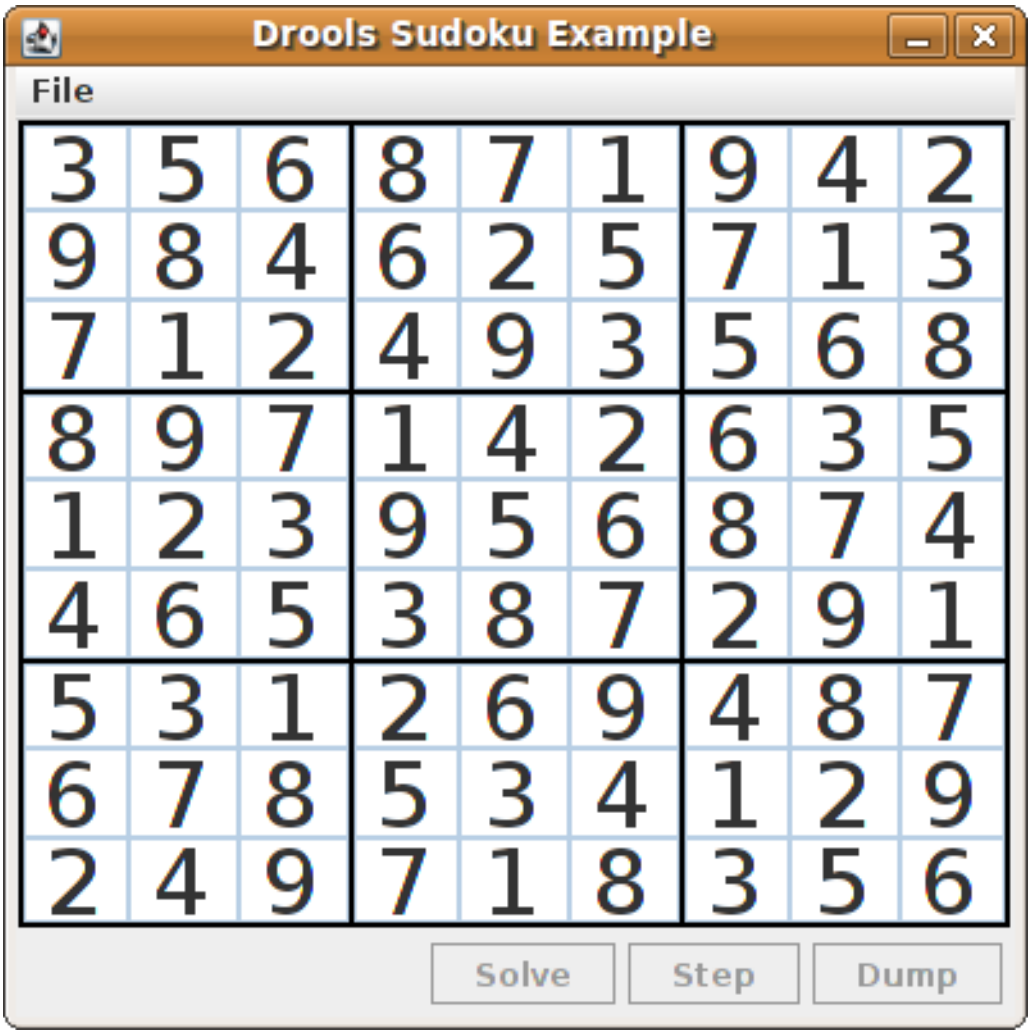
图 89.20. 加载简单示例后的 Sudoku 示例 GUI



从以下选项中选择：

- 点 Solve 触发 Sudoku 示例中定义的规则，该示例中填充剩余的值，并且使按钮再次不活跃。

图 89.21. 已解决简单示例



- 点 Step 查看规则集找到的下一个数字。IDE 中的控制台窗口显示从中解决问题的规则的详细信息。

IDE 控制台中的步骤执行输出

```
single 8 at [0,1]
column elimination due to [1,2]: remove 9 from [4,2]
hidden single 9 at [1,2]
row elimination due to [2,8]: remove 7 from [2,4]
remove 6 from [3,8] due to naked pair at [3,2] and [3,7]
hidden pair in row at [4,6] and [4,4]
```

- 点击 Dump 查看网格的状态，其中单元显示了既定的值或剩余的可能性。

在 IDE 控制台中转储执行输出

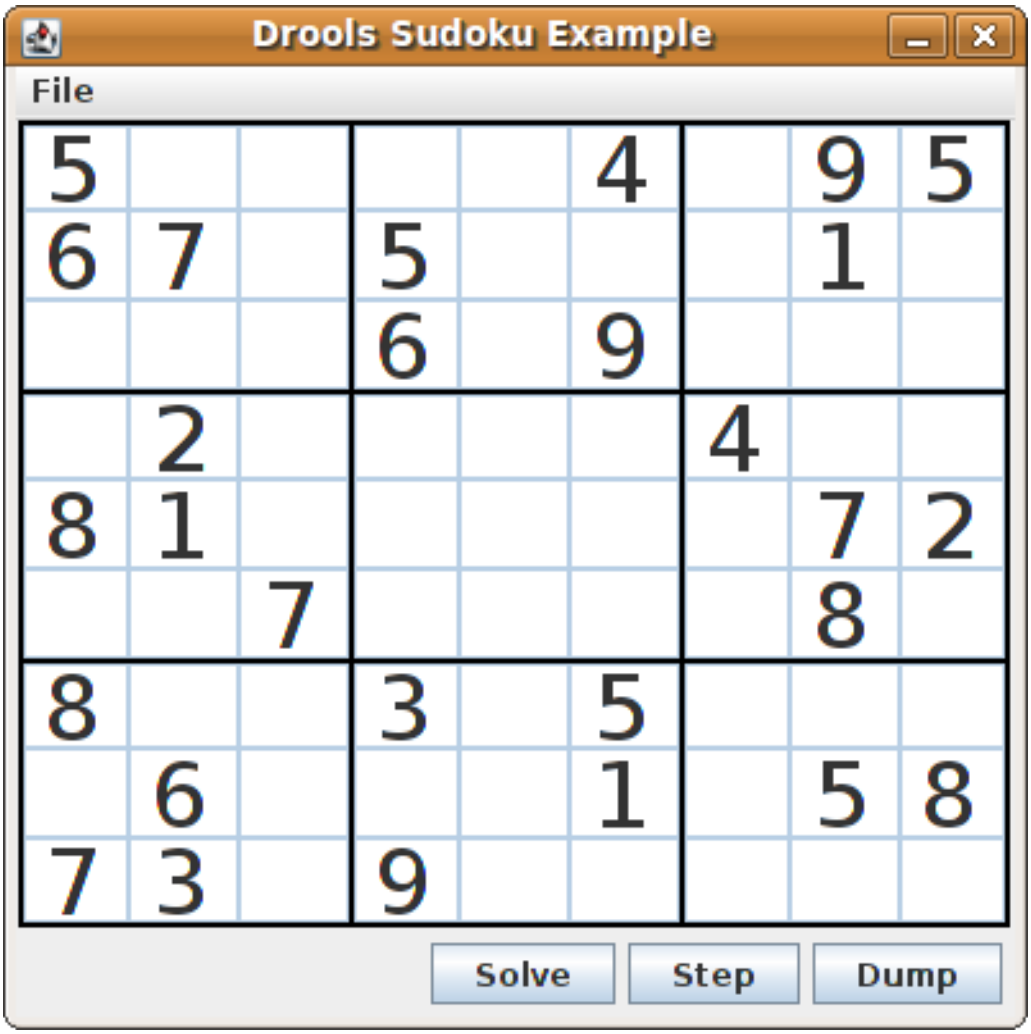
在 IDE 控制台中打印执行输出

```
      Col: 0     Col: 1     Col: 2     Col: 3     Col: 4     Col: 5     Col: 6     Col: 7     Col: 8
Row 0:  123456789  --- 5 ---  --- 6 ---  --- 8 ---  123456789  --- 1 ---  --- 9 ---  --- 4 ---
123456789
Row 1:  --- 9 ---  123456789  123456789  --- 6 ---  123456789  --- 5 ---  123456789
123456789  --- 3 ---
Row 2:  --- 7 ---  123456789  123456789  --- 4 ---  --- 9 ---  --- 3 ---  123456789  123456789
--- 8 ---
Row 3:  --- 8 ---  --- 9 ---  --- 7 ---  123456789  --- 4 ---  123456789  --- 6 ---  --- 3 ---  --- 5 ---
Row 4:  123456789  123456789  --- 3 ---  --- 9 ---  123456789  --- 6 ---  --- 8 ---  123456789
123456789
Row 5:  --- 4 ---  --- 6 ---  --- 5 ---  123456789  --- 8 ---  123456789  --- 2 ---  --- 9 ---  --- 1 ---
Row 6:  --- 5 ---  123456789  123456789  --- 2 ---  --- 6 ---  --- 9 ---  123456789  123456789
--- 7 ---
Row 7:  --- 6 ---  123456789  123456789  --- 5 ---  123456789  --- 4 ---  123456789
123456789  --- 9 ---
Row 8:  123456789  --- 4 ---  --- 9 ---  --- 7 ---  123456789  --- 8 ---  --- 3 ---  --- 5 ---
123456789
```

Sudoku 示例包含一个有意破坏示例文件，示例中定义的规则可以解析。

点 File → Samples → !DELIBERATELY BROKEN! 加载损坏的示例。网格从一些问题开始，例如，该值 5 显示在第一行中不允许的两次。

图 **89.22. broken Sudoku 示例初始状**态



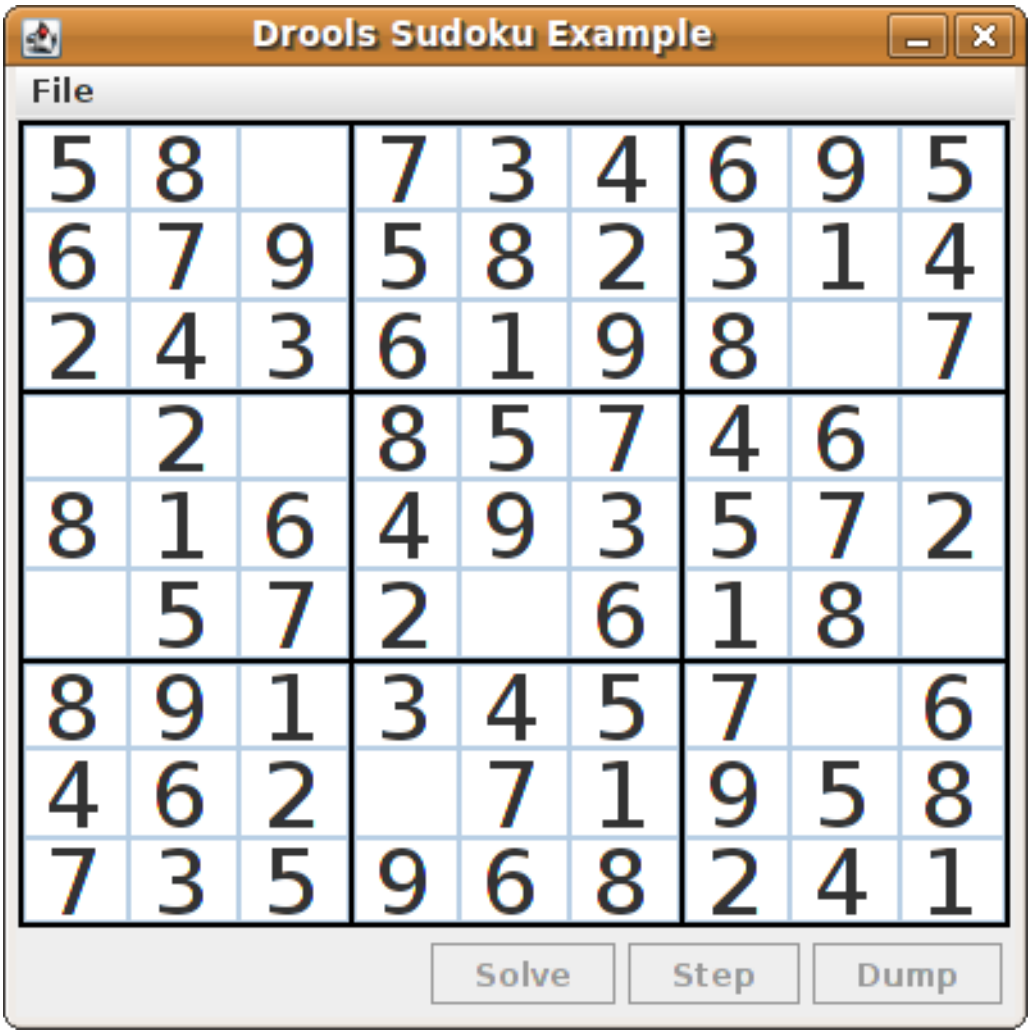
单击 **Solve，将解决**规则应**用到此无效网格。Sudoku 示例中的关**联**解决**问题检测样**本中的**问题，**并尽可能**尝试**解决**这**个**问题。这**个**过**程没有完成，并使一些**单**元留空。**

IDE 控制台窗口中会显示 **solve** 规则**活**动：

检测**到无法正常工作的**问题

```
cell [0,8]: 5 has a duplicate in row 0
cell [0,0]: 5 has a duplicate in row 0
cell [6,0]: 8 has a duplicate in col 0
cell [4,0]: 8 has a duplicate in col 0
Validation complete.
```

图 89.23. 有问题的解决方案示例尝试



名为 Hard 的 Sudoku 示例文件更为复杂，解决规则可能无法解决它们。IDE 控制台窗口中会显示不成功的解决方案：

未解析的硬示例

```
Validation complete.
...
Sorry - can't solve this grid.
```

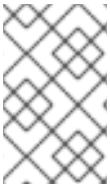
根据仍是单元的候选值集，用于解决中断示例实施标准解析技术的规则。例如，如果一个集合包含单个值，则这是单元的值。对于九个单元里有一个值出现的一个值，规则插入了某些特定单元的"使用解决方案值"类型 设置 的事实。这一事实导致从该单元所属的任何组中所有其他单元格中消除这个值，且值将被重新处理。

示例中的其他规则会减少某些单元的可分值。规则 "naked pair", "hidden pair in row", "hidden pair in column", 和 "hidden pair in square" 消除了可能性，但没有建立解决方案。规则 "位于 行中的 X-wing in rows", "'X-wing in columns'", "interection removal line", and "intersection removal 列" 来执行更复杂的清除。

Sudoku 示例类

org.drools.examples.sudoku.swing 包含了为 Sudoku puzzles 实施框架的以下核心组件：

- SudokuGridModel 类定义了一个接口，它将一个接口存储一个 Sudoku puzzle，作为 Cell 对象的 9x9 网格。
- SudokuGridView 类是一个 Swing 组件，它可以视觉化 SudokuGridModel 类的任何实现。
- SudokuGridEvent 和 SudokuGridListener 类在模型和视图之间沟通状态变化。当一个单元值被解析或更改时，将触发事件。
- SudokuGridSamples 类为演示目的提供了部分填充的 Sudoku 模糊。



注意

这个软件包对 Red Hat Process Automation Manager 库没有依赖软件包。

package org.drools.examples.sudoku 包含以下用于实施元素 Cell 对象及其各种聚合的类集：

- CellFile 类，其子类型 CellRow、CellCol 和 CellSqr 是 CellGroup 类。
- Cell 和 CellGroup 子类的 SetOfNine 中的一个属性 自由 提供带有 type Set<Integer> 的属性。对于 Cell 类，该集合代表个人候选集。对于 CellGroup 类，该集合就是其所有单元集的 union（仍然需要分配的数字集）。

在 Sudoku 示例中，81 Cell 和 27 CellGroup 对象和由 Cell 属性 cellRow、cellCol 和 cellSqr 提供的链接，以及 CellGroup 属性 单元单元 （ Cell II 对象列表）。使用这些组件，您可以编写用于检测特定情况的规则，允许为某个候选项集中的单元或消除值分配值。

- Setting class 用于触发值分配后的操作。在检测到新情况的所有规则中使用了 设置 事实，以避免重新操作中间状态。

- Stepping 类用于低优先级规则，当 "Step" 没有定期终止时，会停止紧急情况。这个行为表示这个程序无法解决。

- 主类 org.drools.examples.sudoku.SudokuExample 实施 Java 应用程序，并合并所有这些组件。

Sudoku 验证规则(validate.drl)

Sudoku 示例中的 validate.drl 文件包含在单元组中检测重复数字的验证规则。它们组合成 "验证" 认证组，用户可在用户加载点后明确激活规则。

三个规则 "重复为 cell ..." 所有功能的 when 条件如下：

- 规则中的第一个条件会找到一个带有分配值的单元。

- 规则中的第二个条件会拉取单元所属的三个单元组。

- 最终条件找到一个单元（除前一个）的单元（除前一个），其值与第一单元相同，以及同一行、列或方括号，具体取决于规则。

规则 "duicate in cell ..."

```
rule "duplicate in cell row"
  when
    $c: Cell( $v: value != null )
    $cr: CellRow( cells contains $c )
    exists Cell( this != $c, value == $v, cellRow == $cr )
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in row " + $cr.getNumber() );
end

rule "duplicate in cell col"
  when
    $c: Cell( $v: value != null )
    $cc: CellCol( cells contains $c )
    exists Cell( this != $c, value == $v, cellCol == $cc )
```

```
  then
    System.out.println( "cell " + $c.toString() + " has a duplicate in col " + $cc.getNumber() );
end

rule "duplicate in cell sqr"
  when
    $c: Cell( $v: value != null )
    $cs: CellSqr( cells contains $c )
    exists Cell( this != $c, value == $v, cellSqr == $cs )
  then
    System.out.println( "cell " + $c.toString() + " has duplicate in its square of nine." );
end
```

规则 **"terminate group" 是最后触**发**的最后一个。**这**个**规则**会**显**示信息并停止序列。**

规则**"确定**组**"**

```
rule "terminate group"
    salience -100
  when
  then
    System.out.println( "Validation complete." );
    drools.halt();
end
```

Sudoku 解决规则**(sudoku.drl)**

Sudoku 示例中的 sudoku.drl 文件包含三种类**型的**规则**：一个**组处**理一个到**单**元的分配，**另**一个**组检测**到可行的分配，第三个**组**会从 candidate 集合中消除**值**。**

规则 **"set a value", "iminate a value from Cell " 和 "retract settings" 取决于 Setting** 对**象的存在。第一个**规则处**理分配**给单**元的分配，操作会从**单**元 的空**闲 组**中**删**除**值。这**个**组**也会降低一个**计**数器，在零**时**将控制**权**返回到名**为 **fireUntilHalt（） 的 Java** 应**用程序。**

规则 **"imininate a value from Cell" 的目的是**减**少与新分配的**单**元相关的所有**单**元的列表。最后，当**进**行所有的精**简时，规则 **"retract** 设**置" 会回收触**发 设**置 事**实**。**

rules "set a value", "iminate a value from a Cell" and "retract set"

```
// A Setting object is inserted to define the value of a Cell.
// Rule for updating the cell and all cell groups that contain it
rule "set a value"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // A matching Cell, with no value set
    $c: Cell( rowNo == $rn, colNo == $cn, value == null,
              $cr: cellRow, $cc: cellCol, $cs: cellSqr )

    // Count down
    $ctr: Counter( $count: count )
  then
    // Modify the Cell by setting its value.
    modify( $c ){ setValue( $v ) }
    // System.out.println( "set cell " + $c.toString() );
    modify( $cr ){ blockValue( $v ) }
    modify( $cc ){ blockValue( $v ) }
    modify( $cs ){ blockValue( $v ) }
    modify( $ctr ){ setCount( $count - 1 ) }
end

// Rule for removing a value from all cells that are siblings
// in one of the three cell groups
rule "eliminate a value from Cell"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // The matching Cell, with the value already set
    Cell( rowNo == $rn, colNo == $cn, value == $v, $exCells: exCells )

    // For all Cells that are associated with the updated cell
    $c: Cell( free contains $v ) from $exCells
  then
    // System.out.println( "clear " + $v + " from cell " + $c.posAsString()  );
    // Modify a related Cell by blocking the assigned value.
    modify( $c ){ blockValue( $v ) }
end

// Rule for eliminating the Setting fact
rule "retract setting"
  when
    // A Setting with row and column number, and a value
    $s: Setting( $rn: rowNo, $cn: colNo, $v: value )

    // The matching Cell, with the value already set
    $c: Cell( rowNo == $rn, colNo == $cn, value == $v )

    // This is the negation of the last pattern in the previous rule.
    // Now the Setting fact can be safely retracted.
    not( $x: Cell( free contains $v )
```

```
        and
        Cell( this == $c, exCells contains $x ) )
  then
    // System.out.println( "done setting cell " + $c.toString() );
    // Discard the Setter fact.
    delete( $s );
    // Sudoku.sudoku.consistencyCheck();
end
```

两种解决规则可检测某个情况，在可以分配多个单元时。对于一个包含单个数字的候选集，规则 "single" 触发。当 单个候选者没有单元格时，规则"隐藏 单"触发，但存在包含候选的单元时，这不包括在单元所属的三个组里的其它单元格中。这两个规则都创建并插入 设置 事实。

规则"单一"和"隐藏单一"

```
// Detect a set of candidate values with cardinality 1 for some Cell.
// This is the value to be set.
rule "single"
  when
    // Currently no setting underway
    not Setting()

    // One element in the "free" set
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount == 1 )
  then
    Integer i = $c.getFreeValue();
    if (explain) System.out.println( "single " + i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, i ) );
end

// Detect a set of candidate values with a value that is the only one
// in one of its groups. This is the value to be set.
rule "hidden single"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Some integer
    $i: Integer()

    // The "free" set contains this number
    $c: Cell( $rn: rowNo, $cn: colNo, freeCount > 1, free contains $i )

    // A cell group contains this cell $c.
    $cg: CellGroup( cells contains $c )
```

```
    // No other cell from that group contains $i.
    not ( Cell( this != $c, free contains $i ) from $cg.getCells() )
  then
    if (explain) System.out.println( "hidden single " + $i + " at " + $c.posAsString() );
    // Insert another Setter fact.
    insert( new Setting( $rn, $cn, $i ) );
end
```

来自最大的组**的**规则（单**独或在**两组**或三**组）实**施多种解决技**术，用于手**动**解决 Sudoku** 议**会。**

规则 **"naked pair" 在**组**的**两**个**单**元中**检测**相同的候**选**大小集合。**这两**个**值**可以从**该组**的所有其他候**选组**中**删**除。**

规则**"naked pair"**

```
// A "naked pair" is two cells in some cell group with their sets of
// permissible values being equal with cardinality 2. These two values
// can be removed from all other candidate lists in the group.
rule "naked pair"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // One cell with two candidates
    $c1: Cell( freeCount == 2, $f1: free, $r1: cellRow, $rn1: rowNo, $cn1: colNo, $b1: cellSqr )

    // The containing cell group
    $cg: CellGroup( freeCount > 2, cells contains $c1 )

    // Another cell with two candidates, not the one we already have
    $c2: Cell( this != $c1, free == $f1 /*** , rowNo >= $rn1, colNo >= $cn1 ***/ ) from $cg.cells

    // Get one of the "naked pair".
    Integer( $v: intValue ) from $c1.getFree()

    // Get some other cell with a candidate equal to one from the pair.
    $c3: Cell( this != $c1 && != $c2, freeCount > 1, free contains $v ) from $cg.cells
  then
    if (explain) System.out.println( "remove " + $v + " from " + $c3.posAsString() + " due to naked pair
at " + $c1.posAsString() + " and " + $c2.posAsString() );
    // Remove the value.
    modify( $c3 ){ blockValue( $v ) }
end
```

...”中三个规则” 函数与规则 “naked pair” 类似。这些规则检测一个组只两个单元格中两个数字的子集，且在组的任何其他单元格中都没有发生任何值。这意味着，其他所有候选者都可以从两个单元格中去除以隐藏的隐藏对。

规则“hidden 对在 ...”

```
// If two cells within the same cell group contain candidate sets with more than
// two values, with two values being in both of them but in none of the other
// cells, then we have a "hidden pair". We can remove all other candidates from
// these two cells.
rule "hidden pair in row"
  when
    // Currently no setting underway
    not Setting()
    not Cell( freeCount == 1 )

    // Establish a pair of Integer facts.
    $i1: Integer()
    $i2: Integer( this > $i1 )

    // Look for a Cell with these two among its candidates. (The upper bound on
    // the number of candidates avoids a lot of useless work during startup.)
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellRow: cellRow )

    // Get another one from the same row, with the same pair among its candidates.
    $c2: Cell( this != $c1, cellRow == $cellRow, freeCount > 2, free contains $i1 && contains $i2 )

    // Ascertain that no other cell in the group has one of these two values.
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellRow.getCells() )
  then
    if( explain) System.out.println( "hidden pair in row at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    // Set the candidate lists of these two Cells to the "hidden pair".
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
$cellCol: cellCol )
```

```
    $c2: Cell( this != $c1, cellCol == $cellCol, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellCol.getCells() )
  then
    if (explain) System.out.println( "hidden pair in column at " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end

rule "hidden pair in square"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i1: Integer()
    $i2: Integer( this > $i1 )
    $c1: Cell( $rn1: rowNo, $cn1: colNo, freeCount > 2 && < 9, free contains $i1 && contains $i2,
            $cellSqr: cellSqr )
    $c2: Cell( this != $c1, cellSqr == $cellSqr, freeCount > 2, free contains $i1 && contains $i2 )
    not( Cell( this != $c1 && != $c2, free contains $i1 || contains $i2 ) from $cellSqr.getCells() )
  then
    if (explain) System.out.println( "hidden pair in square " + $c1.posAsString() + " and " +
$c2.posAsString() );
    modify( $c1 ){ blockExcept( $i1, $i2 ) }
    modify( $c2 ){ blockExcept( $i1, $i2 ) }
end
```

两条规则处理行和列中的 "Xwing "。当值的每两个可能单元（或列）中都只包括两个可能的单元格时，这些候选人也存在于同一列（或行），那么可以删除列内（或行）的所有其他候选者。当遵循这些规则中的模式序列时，请注意如何以方便的词语表达的条件，如 相同的 或 仅导致 具有合适的限制的模式，或以 not 为前缀。

规则 ***"X-wings in …"***

```
rule "X-wings in rows"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
              $ra: cellRow, $rano: rowNo,         $c1: cellCol,        $c1no: colNo )
    $cb1: Cell( freeCount > 1, free contains $i,
              $rb: cellRow, $rbno: rowNo > $rano,      cellCol == $c1 )
    not( Cell( this != $ca1 && != $cb1, free contains $i ) from $c1.getCells() )

    $ca2: Cell( freeCount > 1, free contains $i,
```

```
            cellRow == $ra, $c2: cellCol,       $c2no: colNo > $c1no )
    $cb2: Cell( freeCount > 1, free contains $i,
            cellRow == $rb,      cellCol == $c2 )
    not( Cell( this != $ca2 && != $cb2, free contains $i ) from $c2.getCells() )

    $cx: Cell( rowNo == $rano || == $rbno, colNo != $c1no && != $c2no,
           freeCount > 1, free contains $i )
  then
    if (explain) {
       System.out.println( "X-wing with " + $i + " in rows " +
          $ca1.posAsString() + " - " + $cb1.posAsString() +
          $ca2.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "X-wings in columns"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    $ca1: Cell( freeCount > 1, free contains $i,
            $c1: cellCol, $c1no: colNo,         $ra: cellRow,        $rano: rowNo )
    $ca2: Cell( freeCount > 1, free contains $i,
            $c2: cellCol, $c2no: colNo > $c1no,      cellRow == $ra )
    not( Cell( this != $ca1 && != $ca2, free contains $i ) from $ra.getCells() )

    $cb1: Cell( freeCount > 1, free contains $i,
            cellCol == $c1, $rb: cellRow,  $rbno: rowNo > $rano )
    $cb2: Cell( freeCount > 1, free contains $i,
            cellCol == $c2,      cellRow == $rb )
    not( Cell( this != $cb1 && != $cb2, free contains $i ) from $rb.getCells() )

    $cx: Cell( colNo == $c1no || == $c2no, rowNo != $rano && != $rbno,
           freeCount > 1, free contains $i )
  then
    if (explain) {
       System.out.println( "X-wing with " + $i + " in columns " +
          $ca1.posAsString() + " - " + $ca2.posAsString() +
          $cb1.posAsString() + " - " + $cb2.posAsString() + ", remove from " + $cx.posAsString()  );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

这两个规则 ***"intersection removal ..." 基于一个方方或单一列内出现某种数量的受限位置。这意味着这个数字必须位于一行或列的两个或者三单元格之一，并可从组所有其他单元的候选单元中删除。这个模式决定了限制发生的情况，然后针对方括号外的每个单元格以及同一单元文件内触发。***

rules "interection removal ..."

```
rule "intersection removal column"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
    $c: Cell( free contains $i, $cs: cellSqr, $cc: cellCol )
    // Does not occur in another cell of the same square and a different column
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellCol != $cc )

    // A cell exists in the same column and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellCol == $cc, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "column elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end

rule "intersection removal row"
  when
    not Setting()
    not Cell( freeCount == 1 )

    $i: Integer()
    // Occurs in a Cell
    $c: Cell( free contains $i, $cs: cellSqr, $cr: cellRow )
    // Does not occur in another cell of the same square and a different row.
    not Cell( this != $c, free contains $i, cellSqr == $cs, cellRow != $cr )

    // A cell exists in the same row and another square containing this value.
    $cx: Cell( freeCount > 1, free contains $i, cellRow == $cr, cellSqr != $cs )
  then
    // Remove the value from that other cell.
    if (explain) {
        System.out.println( "row elimination due to " + $c.posAsString() +
                            ": remove " + $i + " from " + $cx.posAsString() );
    }
    modify( $cx ){ blockValue( $i ) }
end
```

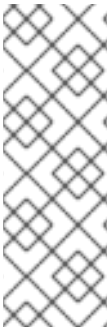
这些规则已足够多，但并非所有 Sudoku 模糊。为了解决非常困难的网格，规则集需要更复杂的规则。（通常，一些不清点可以通过试用和错误解决。）

89.9. CONWAYS OF LIFE EXAMPLE DECISIONS（RULEFLOW 组和 GUI 集成）

Conwayss Game of Life example 决策集，按照 John Conway 的 advertise cellular automaton，演示如何使用 ruleflow 组来控制规则执行。这个示例还演示了如何将 Red Hat Process Automation Manager 规则与图形用户界面(GUI)集成，在这种情况下，基于 Swing 的实施是生命周期。

以下是生命周期游戏(Conway)示例概述：

- 名称 ：conway
- 主要类别 ：org.drools.examples.conway.ConwayRuleFlowGroupRun,org.drools.examples.conway.Conway.Conway.ConwayAgendaGroupRun (in src/main/java)
- 模块:droolsjbpm-integration-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.conway.*.drl （在 src/main/resources）
- 目标 ：演示 ruleflow 组和 GUI 集成



注意

生命周期示例的 Conway's Game 与 Red Hat Process Automation Manager 中的大多数其他示例决策集分开，并位于来自红帽客户门户网站的 ~/rhpam-7.13.5-sources/src/droolsjbpm-integration-$VERSION/droolsjbpm-integration-examples，它包括了来自红帽 客户门户网站的 Red Hat Process Automation Manager 7.13.5 -sources/src/droolsjbpm-integration-examples。

在 Conway 的游戏中，用户通过创建初始配置或具有定义的属性的高级模式来与游戏进行交互，然后观察初始状态的演变方式。游戏旨在展示人们的开发，生成生成。根据所有单元的同步评估，来自前一个结果的生成结果。

以下基本规则管理下一代内容：

- 如果实时单元单元少于两个实时邻居，它会断出一个词子。
- 如果实时单元单元有超过 3 个实时邻居，它将从过度浏览。
- 如果一个死的单元只有 3 个实时邻居，它会进入生命期。

任何不符合这些条件的单元都会被保留为生成下一个。

Conways of Life 示例游戏使用带有 ruleflow-group 属性的 Red Hat Process Automation Manager 规则来定义在游戏中实施的模式。示例还包括一个决策集版本，通过日程安排组实现相同的行为。日程表组使您可以对决策引擎日程表进行分区，以便对规则组提供执行控制。默认情况下，所有规则均位于 MAIN 日程小组。您可以使用 schedule -group 属性来指定该规则的不同日程表组。

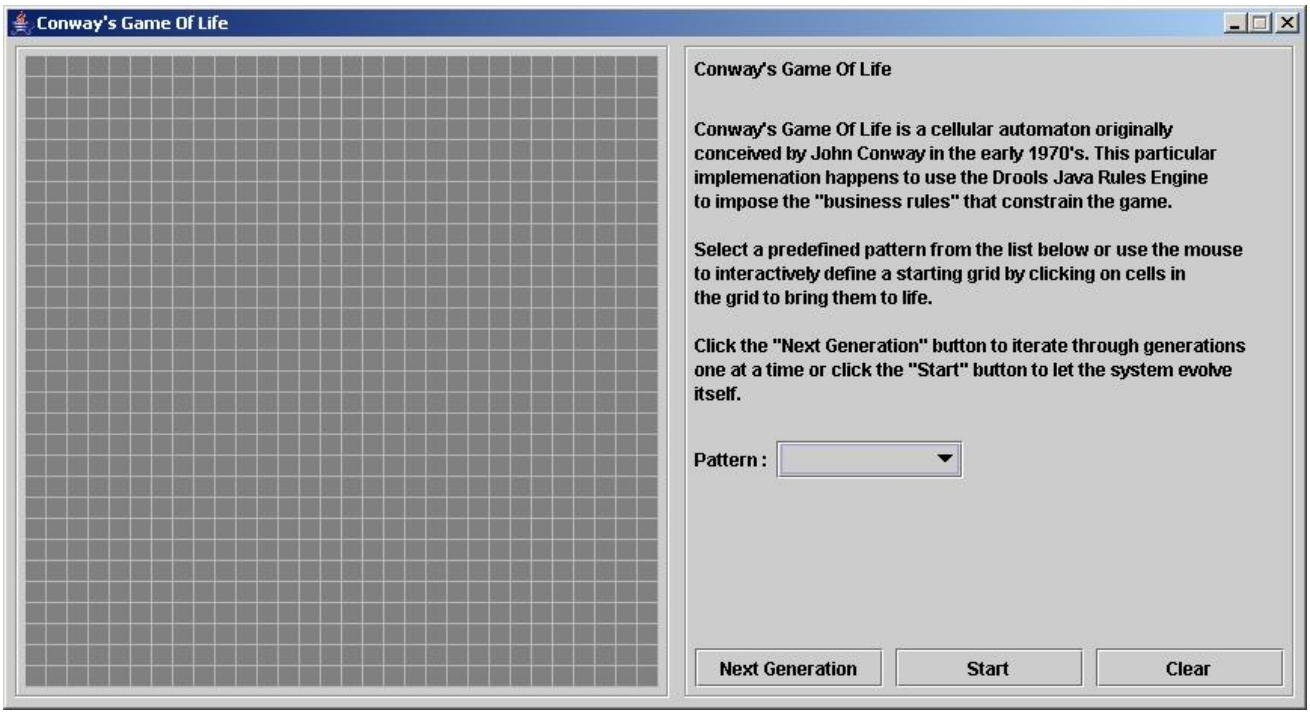
此概述不使用日程表组探索 Conway 示例的版本。有关日程表组的更多信息，请参阅 Red Hat Process Automation Manager 示例决策集，其具体处理日程表组。

沟通执行和互动示例

与其他 Red Hat Process Automation Manager 决策示例类似，您可以通过运行 org.drools.examples.conway.ConwayRuleFlowGroupRun 类作为 IDE 中的 Java 应用程序来执行 Conway 规则流示例。

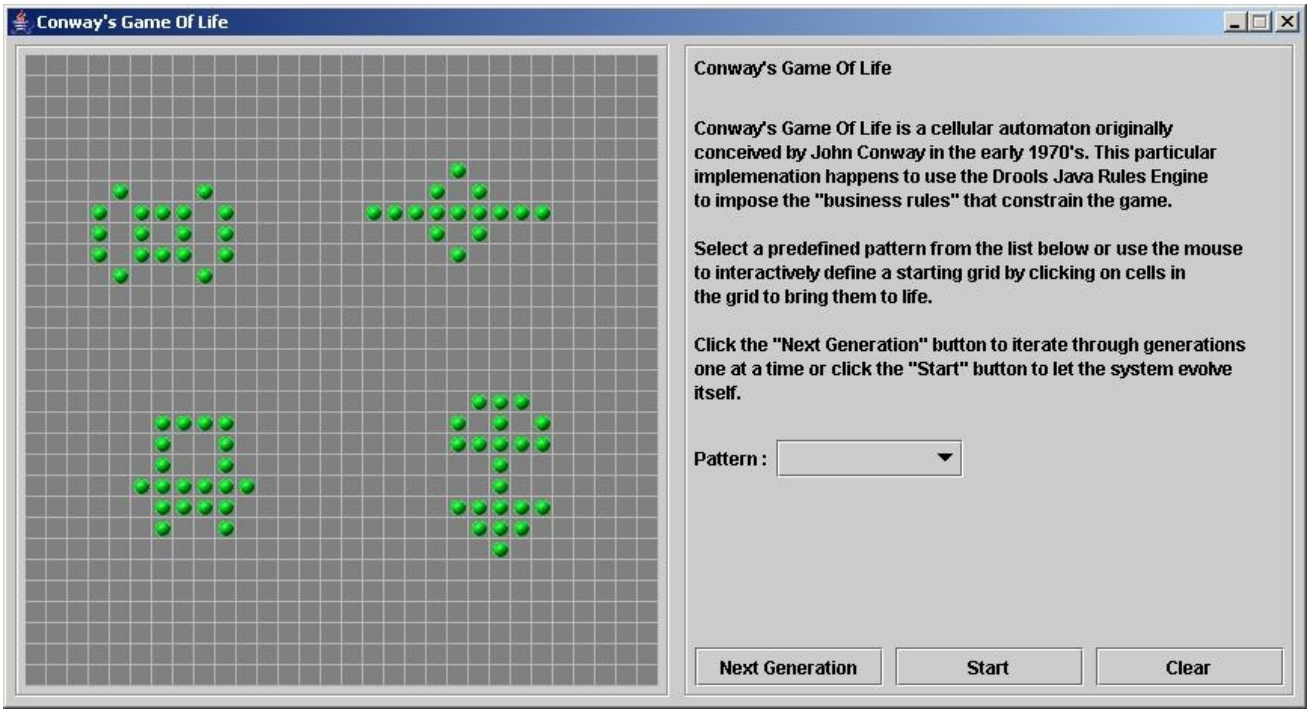
当您执行 Conway 示例时，会出现 Life ways Game of Life GUI 窗口的 Game。此窗口包含空网格，或"arena"，其中将进行模拟。网格最初为空，因为还没有在系统中提供实时单元。

图 89.24. 启动后继续 GUI 示例



从 Pattern 下拉菜单中选择预定义的模式，再单击 Next Generation 以单击每个填充生成。每个单元都可以处于活动状态或死的，其中 live cells 包含一个绿色的 ball。随着人们从初始模式演进，根据游戏的规则，与邻座单元相关的单元格或相对于邻居单元的划分。

图 89.25. 在 Conway 示例中生成演进



邻居不仅包括左、右、顶部和下部的单元格，这些单元格是连接的单位，因此每个单元总数为 8 个邻居。例外是基格（仅有三个邻居），以及这四个边的单元格，分别是五个邻居。

您可以通过单击单元格来手动干预来创建或终止单元。

要从初始模式自动通过 evolution 运行，请单击 Start。

使用 ruleflow 组验证规则示例

ConwayRuleFlowGroupRun 示例中的规则使用 ruleflow 组来控制规则执行。ruleflow group 是 rule flow-group rule 属性关联的一组规则。这些规则只能在激活组时触发。只有当 ruleflow 图的协作达到代表组的节点时，组本身才会变为活跃状态。

Conway 示例对规则使用以下 ruleflow 组：

- "注册邻居"
- "评估"
- "calculate"
- "重置计算"
- "birth"
- "kill"
- "全部技能"

所有 Cell 对象都插入到 KIE 会话中，ruleflow 组中允许 "register ..." 规则由 ruleflow 进程执行。这一组四个规则可在某些单元及其 northeastern、northern、northwestern 和 western 邻居之间创建邻居关系。

这种关系是双向的，可以处理其他四个方向。Border 单元不需要任何特殊处理。这些单元不会与邻居单元（没有任何）搭配使用。

通过为所有规则触发了所有激活的时间，所有单元格都与其所有邻居单元相关。

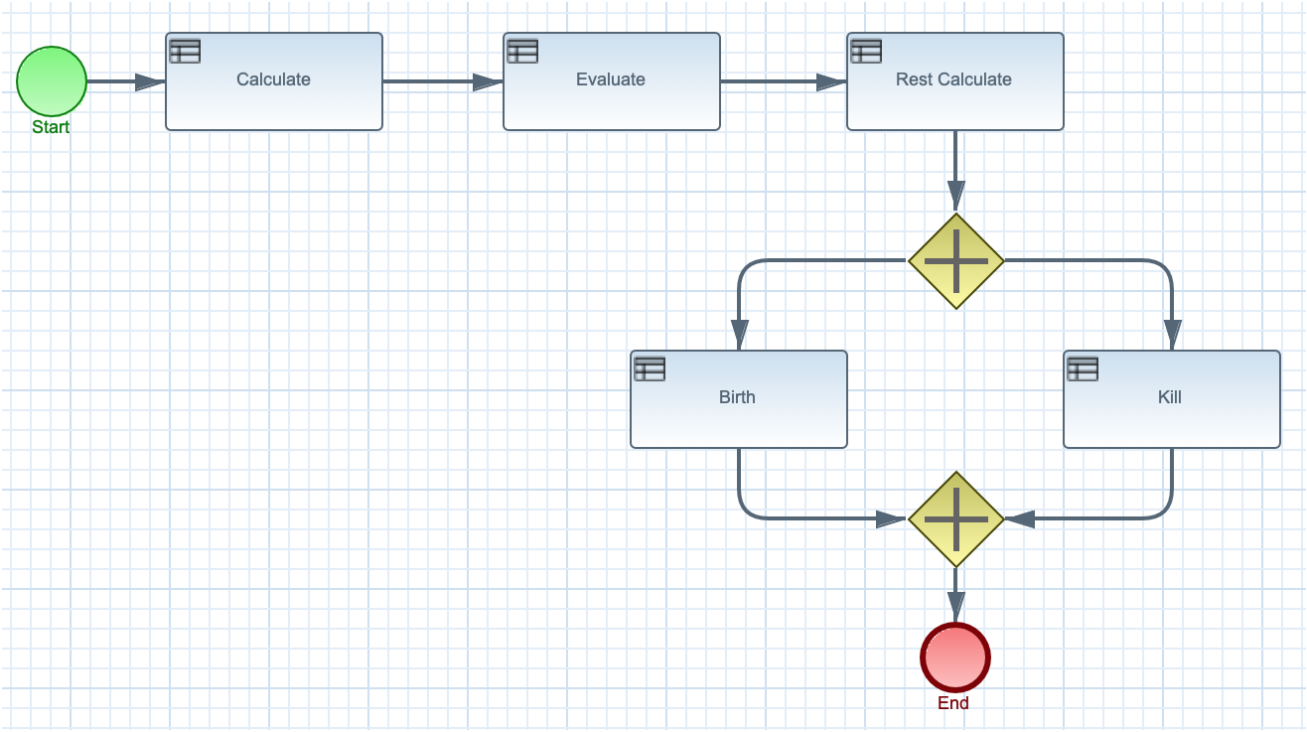
规则 **"register ..."**

```
rule "register north east"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northEast : Cell( row  == ($row - 1), col == ( $col + 1 ) )
  then
    insert( new Neighbor( $cell, $northEast ) );
    insert( new Neighbor( $northEast, $cell ) );
end

rule "register north"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $north : Cell( row  == ($row - 1), col == $col )
  then
    insert( new Neighbor( $cell, $north ) );
    insert( new Neighbor( $north, $cell ) );
end

rule "register north west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $northWest : Cell( row  == ($row - 1), col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $northWest ) );
    insert( new Neighbor( $northWest, $cell ) );
end

rule "register west"
    ruleflow-group "register neighbor"
  when
    $cell: Cell( $row : row, $col : col )
    $west : Cell( row  == $row, col == ( $col - 1 ) )
  then
    insert( new Neighbor( $cell, $west ) );
    insert( new Neighbor( $west, $cell ) );
end
```

插入所有单元格后，一些 Java 代码会将模式应用到网格，将某些单元设置为 **Live**。然后，当用户点击 **Start** 或 **Next Generation** 时，该示例执行 **Generation ruleflow**。此 ruleflow 管理每个生成周期中的

所有单元更改。

图 89.26. 生成规则流



ruleflow 进程进入 "evaluate" ruleflow 组，以及组中的任何活动规则可以触发。这个组中的规则 "Kill the ..." 和 "Give Birth" 将规则规则应用到 birth 或 kill 单元。这个示例使用 phase 属性来根据特定规则组驱动 Cell 对象的原因。通常，该阶段与 ruleflow 进程定义中的 ruleflow 组关联。

请注意，该示例不会更改此时任何 Cell 对象的状态，因为它必须在应用这些更改前完成完整的评估。示例将单元设置为 Phase.KILL 或 Phase.BIRTH，后者稍后用于控制应用到 Cell 对象的操作。

规则"Kill the ..." 和 "Give Birth"

```
rule "Kill The Lonely"
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has fewer than 2 live neighbors.
    theCell: Cell( liveNeighbors < 2, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
        setPhase( Phase.KILL );
    }
end

rule "Kill The Overcrowded"
```

```
    ruleflow-group "evaluate"
    no-loop
  when
    // A live cell has more than 3 live neighbors.
    theCell: Cell( liveNeighbors > 3, cellState == CellState.LIVE,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
       setPhase( Phase.KILL );
    }
end

rule "Give Birth"
    ruleflow-group "evaluate"
    no-loop
  when
    // A dead cell has 3 live neighbors.
    theCell: Cell( liveNeighbors == 3, cellState == CellState.DEAD,
                   phase == Phase.EVALUATE )
  then
    modify( theCell ){
       theCell.setPhase( Phase.BIRTH );
    }
end
```

评估了网格中的所有 Cell 对象后，该示例使用 "reset calculate" 规则清除 "calculate" ruleflow 组中的任何激活。然后，如果 ruleflow 激活了 ruleflow，示例将启用规则 "kill" 和 "birth" 来触发的规则。这些规则应用状态更改。

规则"重置计算"、"kill"和"birth"

```
rule "reset calculate"
    ruleflow-group "reset calculate"
  when
  then
    WorkingMemory wm = drools.getWorkingMemory();
    wm.clearRuleFlowGroup( "calculate" );
end

rule "kill"
    ruleflow-group "kill"
    no-loop
  when
    theCell: Cell( phase == Phase.KILL )
  then
    modify( theCell ){
       setCellState( CellState.DEAD ),
```

```
        setPhase( Phase.DONE );
    }
end

rule "birth"
    ruleflow-group "birth"
    no-loop
  when
    theCell: Cell( phase == Phase.BIRTH )
  then
    modify( theCell ){
        setCellState( CellState.LIVE ),
        setPhase( Phase.DONE );
    }
end
```

在这个阶段，几个 Cell 对象已被修改，其状态会更改为 LIVE 或 DEAD。当单元变为 live 或 dead 时，示例使用规则 "Calculate ..." 中的邻居关系来迭代所有周围的单元格，从而增加或减少 liveNeighbor 计数。任何改变计数的单元也会设置为 EVALUATE 阶段，以确保它在 ruleflow 过程的评估阶段包含在原因中。

在为所有单元确定和设置实时数后，ruleflow 进程结束。如果用户最初点击 Start，则决策引擎在该处重启规则流。如果用户最初单击 下一步生成，用户可以请求另一个生成。

规则*"Calculate ..."*

```
rule "Calculate Live"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.LIVE )
    Neighbor( cell == theCell, $neighbor : neighbor )
  then
    modify( $neighbor ){
        setLiveNeighbors( $neighbor.getLiveNeighbors() + 1 ),
        setPhase( Phase.EVALUATE );
    }
end

rule "Calculate Dead"
    ruleflow-group "calculate"
    lock-on-active
  when
    theCell: Cell( cellState == CellState.DEAD )
    Neighbor( cell == theCell, $neighbor : neighbor )
```

```
  then
    modify( $neighbor ){
       setLiveNeighbors( $neighbor.getLiveNeighbors() - 1 ),
       setPhase( Phase.EVALUATE );
    }
end
```

89.10. DOOM 示例决策内部（反向连锁和递归）

Doom 示例决策集的 House 展示了决策引擎如何使用向串联和递归来达到等级系统中定义的目标或子项。

以下是 Doom 示例 House 的概述：

- 名称 ：反向链接
- 主类 ：org.drools.examples.backwardchaining.HouseOfDoomMain （在 src/main/java中）
- 模块 ：drools-examples
- 键入: Java 应用程序
- 规则文件 ：org.drools.examples.backwardchaining.BC-Example.drl (src/main/resources)
- 目标 ：演示后向链和递归

反向链规则系统是一个目标驱动的系统，从结论开始，决定引擎尝试满足（通常使用递归）。如果系统无法达到结论或目标，它会搜索部分当前目标的子项。系统会继续这个过程，直到初始的结论是满足或者所有子语满意。

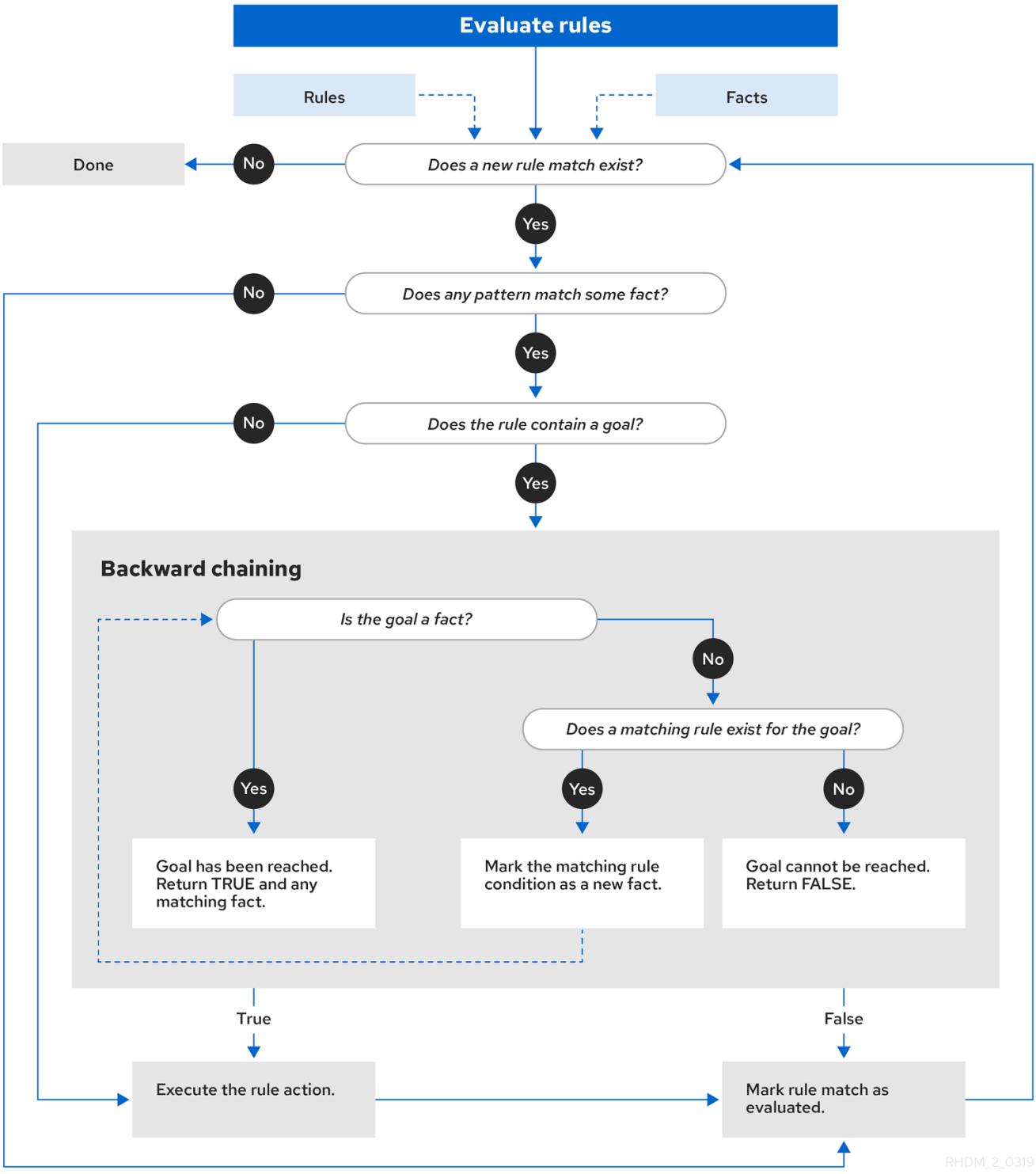
与之相反，正向链规则系统是一个数据驱动的系统，从决策引擎的工作内存中以事实开头，并对该事

实的更改做出响应。当对象插入到工作内存时，因为更改是由日程表计划执行而变为 true 的任何规则条件。

Red Hat Process Automation Manager 中的决策引擎使用正向和向后链来评估规则。

下图显示了如何使用转发链在逻辑流中的反向链接片段评估规则：

图 89.27. 使用转发和向后链的规则评估逻辑



例如，House of Doom 示例使用包含各种查询类型的规则来查找房间和项目的位置。示例类 Location.java 包含示例中使用的 项目 和位置 元素。示例类 HouseOfDoomMain.java 在其所在位置插入项目或房间，并执行规则。

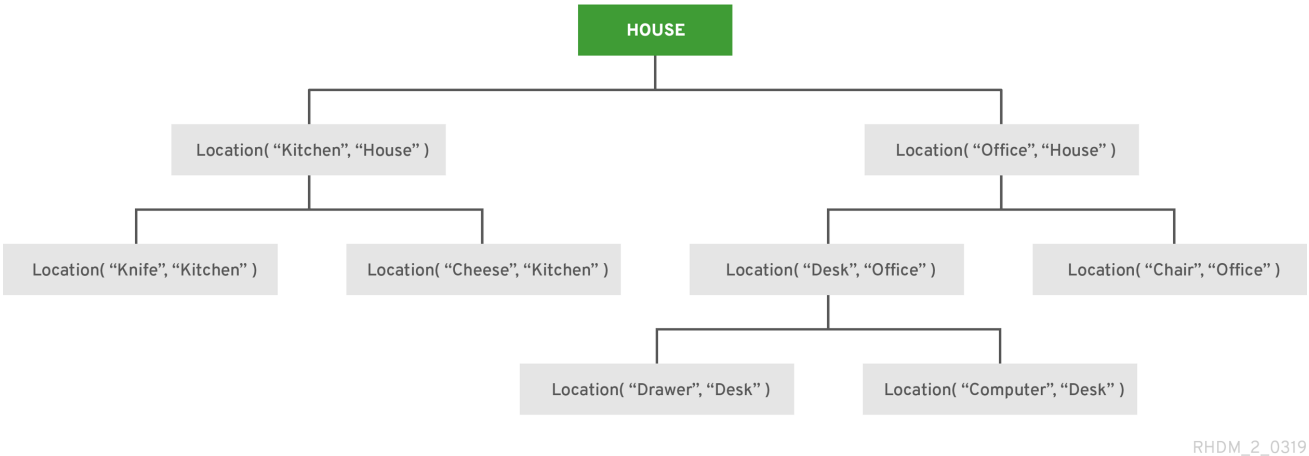
HouseOfDoomMain.java 类中的项目和位置

```
ksession.insert( new Location("Office", "House") );
ksession.insert( new Location("Kitchen", "House") );
ksession.insert( new Location("Knife", "Kitchen") );
ksession.insert( new Location("Cheese", "Kitchen") );
ksession.insert( new Location("Desk", "Office") );
ksession.insert( new Location("Chair", "Office") );
ksession.insert( new Location("Computer", "Desk") );
ksession.insert( new Location("Drawer", "Desk") );
```

示例规则依赖于向后链和递归来确定内部结构中所有项目和房间的位置。

下图展示了 Doom 的 House 的结构以及其中的项目和房间：

图 89.28. Doom 结构内部



要执行示例，请运行 org.drools.examples.backwardchaining.HouseOfDoomMain 类，在 IDE 中作为 Java 应用程序。

执行后，会在 IDE 控制台窗口中显示以下输出：

IDE 控制台中的执行输出

```
go1
Office is in the House
---
go2
Drawer is in the House
---
go3
---
Key is in the Office
---
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
---
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

示例中的所有规则均已触发，以检测内部所有项目的位置，并在输出中打印各个项目的位置。

递归查询和相关规则

递归查询会重复搜索数据结构的层次结构，以获得元素之间的关系。

在 Doom 示例的 House of Doom 示例中， BC-Example.drl 文件包含一个 isContainedIn 查询，示例中大多数规则的查询用于递归评估插入到决策引擎中的数据结构：

BC-Example.drl 中的递归查询

```
query isContainedIn( String x, String y )
  Location( x, y; )
  or
  ( Location( z, y; ) and isContainedIn( x, z; ) )
end
```

规则 **"go" 打印插入到系统中的每个字符串，以确定如何实施项目，规则 "go1" 调用查询 为 ContainedIn ：**

规则 **"go" 和 "go1"**

```
rule "go" salience 10
  when
    $s : String()
  then
    System.out.println( $s );
end

rule "go1"
  when
    String( this == "go1" )
    isContainedIn("Office", "House"; )
  then
    System.out.println( "Office is in the House" );
end
```

这个示例将 "go1" 字符串插入到决策引擎中，并激活 "go1" 规则来检测该项目 办事处 位于位置

House 中：

插入字符串和触发规则

```
ksession.insert( "go1" );
ksession.fireAllRules();
```

IDE 控制台中的规则**"go1"**输**出**

```
go1
Office is in the House
```

临时**防止**规则

传输**冲突是父元素中包含的元素之**间**的关系，**该**元素在分**级结构**中高于多个**级别**。**

规则 **"go2"** 标识 **Drawer 和 House 的**传输**冲突：D rawer 在 House** 处**位于**办**事**处**的 Desk。**

```
rule "go2"
  when
    String( this == "go2" )
    isContainedIn("Drawer", "House"; )
  then
    System.out.println( "Drawer is in the House" );
end
```

这**个示例将 "go2" 字符串插入到决策引擎中，并激活 "go2"** 规则，**以**检测项**目 Drawer 最**终**在位置 House 中：**

插入字符串和触发规则

```
ksession.insert( "go2" );
ksession.fireAllRules();
```

-

IDE 控制台中的规则"go2"输出

```
go2
Drawer is in the House
```

决策引擎根据以下逻辑决定这一结果：

1. 查询会以递归方式搜索内部的几个级别，以检测 Drawer 和 House 之间传输冲突。
2. 查询该选项使用 (z, y;) 而不是使用 Location(x, y;)，因为 Drawer 不在 House 中。
3. z 参数目前未绑定，这意味着它没有值并返回参数中的所有内容。
4. y 参数目前绑定到 House，因此 z 返回 office 和 Kitchen。
5. 该查询从 办公室 收集信息并递归检查该办事处是否位于 办事处。对于这些参数，调用查询行 isContainedIn(x, z;)。
6. 办事处 没有直接存在 Drawer 实例，因此无法找到任何匹配项。
7. 通过 z unbound，查询会返回 办事处 中的数据，并确定 z == Desk。

   ```
   isContainedIn(x==drawer, z==desk)
   ```

8. isContainedIn 查询会以递归方式搜索三次，而且在第三个时间，查询检测到 Desk 中的 Drawer 实例。

```
Location(x==drawer, y==desk)
```

9. 在第一个位置上的此匹配项后，查询会以递归方式搜索结构，以确定 Drawer 位于 Desk 中，Desk 位于 办事处，且该 办事处 位于 House 中。因此，Drawer 位于 House 中，该规则会满足。

被动查询规则

被动查询会搜索数据结构的层次结构，以获得元素之间的关系，并在修改结构中的元素时动态更新。

规则 "go3" 函数作为被动查询，通过传输冲突检测到在 办公室中 是否出现新的项目 密钥 ： 办公室的 Drawer 中的密钥。

规则"go3"

```
rule "go3"
  when
    String( this == "go3" )
    isContainedIn("Key", "Office"; )
  then
    System.out.println( "Key is in the Office" );
end
```

这个示例将 "go3" 字符串插入到决策引擎中，并激活 "go3" 规则。最初，此规则不满意，因为内部结构中没有项目 密钥，因此该规则不会产生任何输出。

插入字符串和触发规则

```
ksession.insert( "go3" );
ksession.fireAllRules();
```

IDE 控制台中的规则"go3"输出（不满意）

```
go3
```

然后，示例在位置 Drawer 中插入一个新项目 密钥，它位于 办事处。这个更改满足在 "go3" 规则中传输冲突，输出会被相应地填充。

插入新项目位置和触发规则

```
ksession.insert( new Location("Key", "Drawer") );
ksession.fireAllRules();
```

IDE 控制台中的规则"go3"输出(satisfied)

```
Key is in the Office
```

此更改也会在查询后续递归搜索中包含的结构中添加另一个级别。

在规则中带有 unbound 参数的查询

带有一个或多个未绑定参数的查询会返回查询定义的（绑定）参数内所有未定义（绑定）项。如果查询中的所有参数都未绑定，则查询会返回查询范围内的所有项目。

规则 "go4" 使用 unbound 参数项搜索绑定参数 办事处 的所有项目，而不是使用 bound 参数搜索 office 中的特定项目：

规则"go4"

```
rule "go4"
  when
```

```
    String( this == "go4" )
    isContainedIn(thing, "Office"; )
  then
    System.out.println( thing + "is in the Office" );
end
```

这个示例将 "go4" 字符串插入到决策引擎中，并激活 "go4" 规则，以返回办公室中的所有项目：

插入字符串和触发规则

```
ksession.insert( "go4" );
ksession.fireAllRules();
```

IDE 控制台中的规则"go4"输出

```
go4
Chair is in the Office
Desk is in the Office
Key is in the Office
Computer is in the Office
Drawer is in the Office
```

规则 "go5" 使用 unbound 参数项 和位置 来搜索整个 House 数据结构中的所有项目及其位置：

规则"go5"

```
rule "go5"
  when
    String( this == "go5" )
    isContainedIn(thing, location; )
```

```
  then
    System.out.println(thing + " is in " + location );
end
```

这个示例将 "go5" 字符串插入到决策引擎中，并激活 "go5" 规则，以返回 House 数据结构中的所有项目及其位置：

插入字符串和触发规则

```
ksession.insert( "go5" );
ksession.fireAllRules();
```

IDE 控制台中的规则"go5"输出

```
go5
Chair is in Office
Desk is in Office
Drawer is in Desk
Key is in Drawer
Kitchen is in House
Cheese is in Kitchen
Knife is in Kitchen
Computer is in Desk
Office is in House
Key is in Office
Drawer is in House
Computer is in House
Key is in House
Desk is in House
Chair is in House
Knife is in House
Cheese is in House
Computer is in Office
Drawer is in Office
Key is in Desk
```

第 90 章 与决策引擎相关的性能调优注意事项

下列关键概念或建议做法可帮助您优化决策引擎性能。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。

对不需要重要决策引擎更新的无状态 KIE 会话使用顺序模式

顺序模式是决策引擎的高级规则基础配置，允许决策引擎以决策引擎日程表中列出的顺序来评估规则，而无需考虑工作内存的变化。因此，规则执行可能会以连续模式更快，但重要的更新可能不会应用到您的规则。顺序模式只适用于无状态 KIE 会话。

要启用顺序模式，请将系统属性 drools.sequential 设置为 true。

有关启用顺序模式或其他选项的详情，请参考 第 84.3 节 “Phreak 中的顺序模式”。

使用带有事件监听程序的简单操作

限制事件监听程序数及其执行的操作类型。将事件监听程序用于简单操作，如调试日志记录和设置属性。在监听器中，复杂的操作（如网络调用）可能会破坏规则执行。在使用 KIE 会话后，删除附加的事件监听程序以便清理会话，如下例所示：

使用后删除的事件监听程序示例

```
Listener listener = ...;
StatelessKnowledgeSession ksession = createSession();
try {
   ksession.insert(fact);
   ksession.fireAllRules();
   ...
} finally {
   if (session != null) {
      ksession.detachListener(listener);
      ksession.dispose();
   }
}
```

有关内置事件监听器以及在决策引擎中调试日志记录的详情，请参考 第 87 章 决策引擎事件监听程序和调试日志记录。

为可执行模型构建配置 LambdaIntrospector 缓存大小

您可以配置 LambdaIntrospector.methodFingerprintsMap 缓存的大小，该缓存在可执行模型构建中使用。缓存的默认大小为 32。当您为缓存大小配置较小的值时，它会减少内存用量。例如，您可以将系统属性 drools.lambda.introspector.cache.size 配置为 0，以达到最小内存使用量。请注意，较小的缓存大小也会降低构建性能。

对可执行模型使用 lambda externalization

启用 lambda 外部化，以优化运行时的内存消耗。它重写在可执行模型中生成和使用的 lambdas。这可让您多次使用相同的模式和相同的约束。当 rete 或 phreak 实例化时，可执行模型变成垃圾收集。

要为可执行模型启用 lambda 外部化，请包括以下属性：

```
-Ddrools.externaliseCanonicalModelLambda=true
```

配置 alpha 节点范围索引阈值

alpha 节点范围索引用于评估规则约束。您可以使用 drools.alphaNodeRangeIndexThreshold 系统属性配置 alpha 节点范围索引的阈值。阈值的默认值为 9，这表示当前面节点包含超过 9 个 alpha 节点时，启用 alpha 节点范围索引。例如，当您有 9 个类似于 Person (age > 10), Person (age > 20), …, Person (age > 90) 的规则时，您可以有类似的 9 个 alpha 节点。

阈值的默认值取决于相关的优点和开销。但是，如果您为阈值配置了较小的值，可以根据您的规则提高性能。例如，您可以将 drools.alphaNodeRangeIndexThreshold 值配置为 6，当您之前节点有超过六个 alpha 节点时，启用 alpha 节点范围索引。您可以根据规则的性能测试结果为阈值设置适当的值。

启用加入节点范围索引

加入的节点范围索引功能仅在需要加入大量事实时才提高性能，例如： 25639）16 组合。当应用程序插入大量事实时，您可以启用加入节点范围索引并评估性能。默认情况下，加入节点范围索引被禁用。

kmodule.xml 文件示例

```
<kbase name="KBase1" betaRangeIndex="enabled">
```

System property for BetaRangeIndexOption

drools.betaNodeRangeIndexEnabled=true

第 91 章 其他资源

- 为 Red Hat Process Automation Manager 设计决策管理架构
- 决策服务入门
- 使用 DRL 规则设计决策服务
- 打包和部署 Red Hat Process Automation Manager 项目

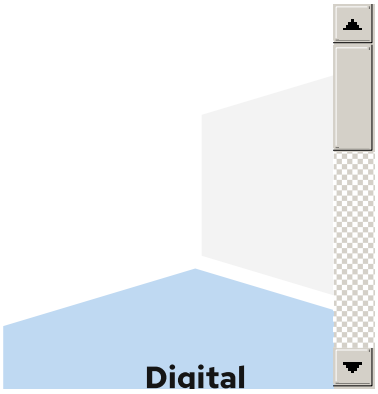
部分 X. 将机器与 RED HAT PROCESS AUTOMATION MANAGER 集成

作为业务分析员或商业规则开发人员，您可以使用 PMML 文件及决策模型及符号(DMN)模型将机器学习与红帽流程自动化管理器集成。

第 92 章 PRMATIC AI

当您考虑智能(AI)时，机器学习和大数据可能会牢记。但是机器学习只是图像的一部分。artificial 智能包括以下技术：

- 机器人：技术、科学和工程集成，以生成能够执行由人类执行的物理任务的计算机
- 机器学习：一组算法的功能可在不明确编程的情况下了解或改进这些算法
- 自然语言处理：需要学习这些过程的一个子集
- 数学优化：使用条件和约束来查找解决问题的最佳解决方案
- 数字决策：使用定义的标准、条件和一系列机器和人工任务来制定决策



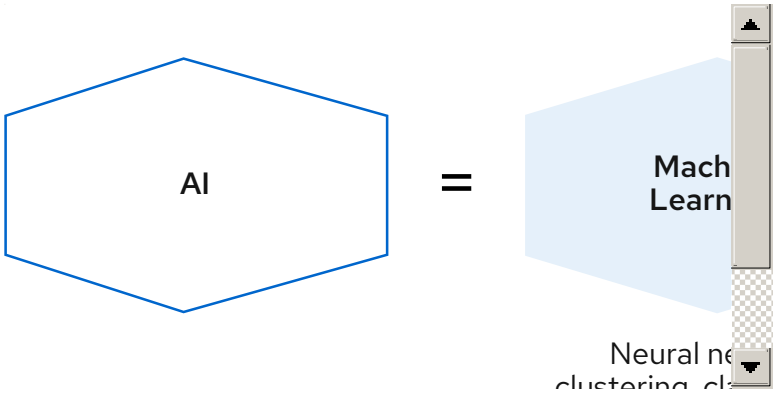
虽然在科学欺诈中中填写了所谓的一般情报(AGI)，但其性能比人更好地表现出，无法区别于他们，并在无需人为干预或控制控制的情况下进行学习和发展，Mon AGI 为 10 年。同时，我们目前也有相当多的 AI，这比我们更有用。实用的 AI 是结合的 AI 技术的集合，为预测客户行为、提供自动客户服务以及帮助客户做出采购决策等解决方案提供解决方案。

领先的行业分析报告显示，之前的组织已经使用 AI 技术，因为他们投资了 AI 潜力，而不是如今 AI 交付的实际情况。AI 项目不是工作效率，因此 AI 项目的投资速度和 AI 项目预算的降低。AI 造成这种不动，通常被称为 AI 风向。AI 早已经历了 AI winters 和 AI springs 的几周期，现在在 AI spring 中被决定。各组织发现了 AI 可以交付的实际情况。不知情地意味着实现实践和现实。AI 有一种实用方法，它考虑了目前可用的 AI 技术，结合了有用的技术，并在为实际问题创建解决方案时添加人为干预。

prmatic AI 解决方案示例

pragmatic AI 中的一个应用程序正在客户支持中。客户会提交报告问题的支持票据，例如登录错误。机器学习算法适用于 ticket，以根据关键字或自然语言处理(NLP)与现有解决方案匹配票据内容。关键词可能会出现在很多解决方案中，有些相关，某些不相关。您可以使用数字决策来确定提供给客户的解决方案。然而，有时该算法提出的任何解决方案都适合向客户提出。这可能是因为所有解决方案都有较低的信心分数，或者多个解决方案都有较高的信心分数。如果无法找到适当的解决方案，则数字决策涉及人为支持团队。要根据可用性和专业知识查找最佳支持人员，数学优化通过考虑员工的声音限制来选择最佳支持票务。

如本例所示，您可以组合机器学习，将信息从数据分析和数字决定中提取到人工知识和体验。然后，您可以应用数学优化来安排人为协助。这是适用于其他情况的模式，例如，信用卡争取和信用卡欺诈检测。



这些技术使用四个行业标准：

- *问题单管理模型和符号(CMMN)*

  CMMN 用于对包含各种可能按不可预测的顺序执行的各种活动进行建模。CMMN 模型是事件中心的。CMMN 通过支持较少结构化工作任务和由人工驱动的任务，克服了使用 BPMN2 可建模的限制。通过结合使用 BPMN 和 CMMN，您可以创建更强大的模型。

- *业务流程模型和符号(BPMN2)*

  BPMN2 规范是一个对象管理组(OMG)规范，用于定义图形表示业务流程的标准，定义元素的执行语义，并提供 XML 格式的进程定义。BPMN2 可以模拟计算机和人工任务。

- *决策模型和符号(DMN)*

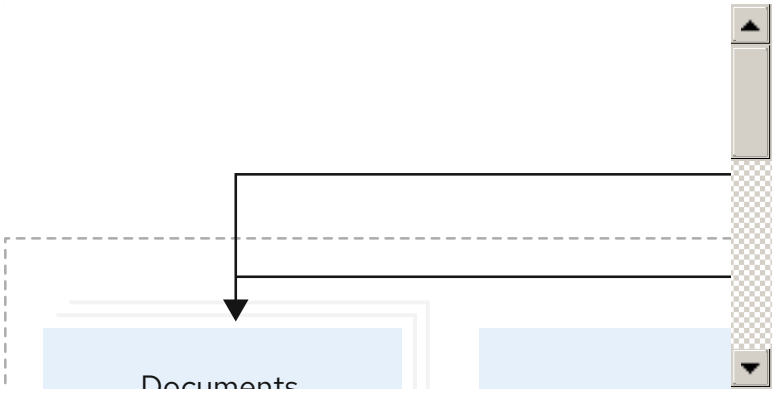
  决策模型和符号(DMN)是 OMG 制定的标准，用于描述和建模操作决策。DMN 定义了一个 XML 模式，使 DMN 模型可以在 DMN 兼容平台和机构中共享，以便业务分析者和业务规则开发

人员能够合作设计和实施 DMN 决策服务。DMN 标准与 相似，可与业务流程建模和符号(BPMN)标准一起使用，以设计和建模业务流程。

- 预测模型标记语言(PMML)

PMML 是用来代表预测模型的语言，使用统计技术来发现或了解大量数据中的数学模型。预测模型使用他们学习的模式来预测新数据中存在模式。使用 PMML 时，您可以在应用程序间共享预测模型。此数据作为 PMML 文件导出，该文件可通过 DMN 模型消耗。随着机器学习框架继续对模型进行培训，可将更新的数据保存到现有的 PMML 文件中。这意味着，您可以使用由任何应用程序创建的预测模型，将模型保存为 PMML 文件。因此，DMN 和 PMML 易于集成。

全部放在一起



本图显示了预测决策自动化的工作方式。

1. 业务数据进入系统，例如来自贷款应用程序的数据。
2. 与预测模型集成的决策模型决定是是否批准贷款，还是需要其他任务。
3. 例如，一个商业行动结果（例如，拒绝信信或贷款优惠）将发送给客户。

下一部分演示了预测决策自动化管理器如何与 Red Hat Process Automation Manager 配合使用。

第 93 章 信用卡欺诈用例

财务行业使用 pragmatic AI 在多个领域进行决策。个领域是信用卡的收费。当客户在信用卡中识别错误或未识别的收费时，客户可能会争取费用。在某些情况下，在某些情况下，需要人为信用卡欺诈的问题，但大多数报告的信用卡欺诈都是完全或部分被攻击的 AI 解析。

该使用案例例子涉及到了 Fortress Bank、银行客户 Joe 和业务流程管理(BPM)开发人员 Michelle。首先，我们将了解银行最初使用红帽流程自动化管理器数字决策使用 AI，随后我们将了解通过机器学习创建的可预测模型标记语言(PMML)模式如何增强决策模型。

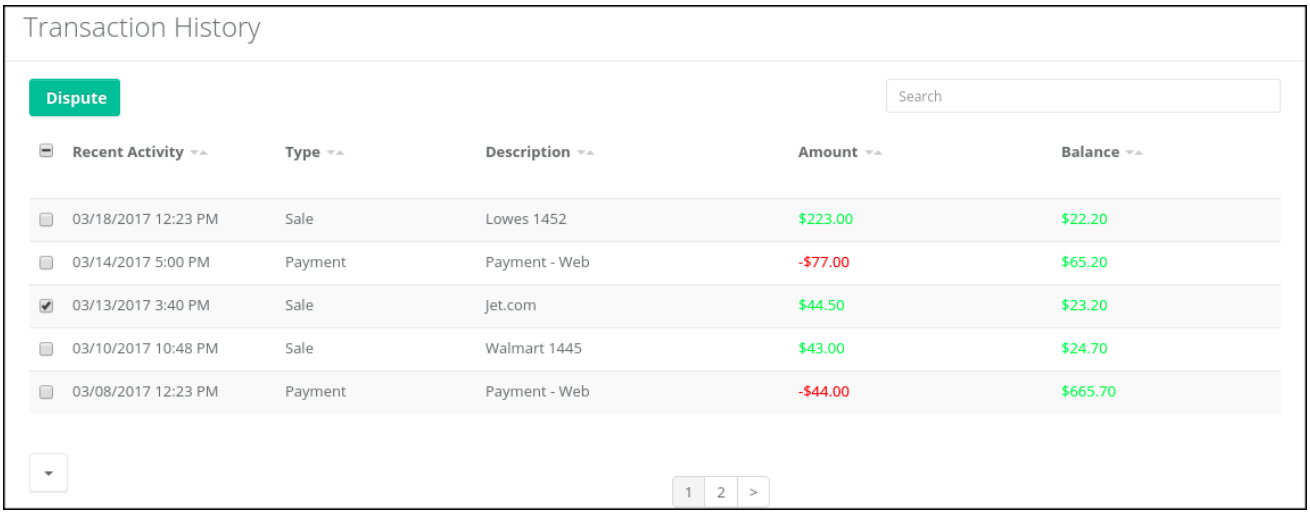
Tensorflow™ 和 R™ 等机器学习模型会产生预测模型。您可以在开放式标准（如 PMML）中保存这些预测模型，以便您可以在 Red Hat Process Automation Manager 或者支持 PMML 标准的其他产品中使用模型。

Red Hat Process Automation Manager 和 Red Hat OpenShift Container Platform

Fortress Bank 在红帽 OpenShift Container Platform 上使用红帽流程自动化管理器来开发和运行 Fortress Bank 决策服务。Red Hat OpenShift Container Platform 是用于开发和运行容器化应用程序的云平台。它旨在使支持它们的应用程序和数据中心从几个机器和应用程序扩展到为数百万客户端服务的数千台机器。Red Hat Process Automation Manager 是用于创建云原生业务自动化应用程序和微服务的红帽中间件平台。它使企业业务和 IT 用户能够记录、模拟、管理、自动化和监控业务流程和决策。Business Central 是 Red Hat Process Automation Manager 仪表板。

使用 Red Hat Process Automation Manager 进行数十个数字决策

joe 是 Fortress Bank 客户。每个月，他登录 Fortress Bank 网站，以便在向其付费之前查看所有费用。这个月，Joe 认为他识别的事务不正确，但数量不正确。供应商支付了 $44.50，而不是 $4.50。joe 选择包含不正确的项的行，并点击 Dispute。



此操作启动一系列有关争议的问题：

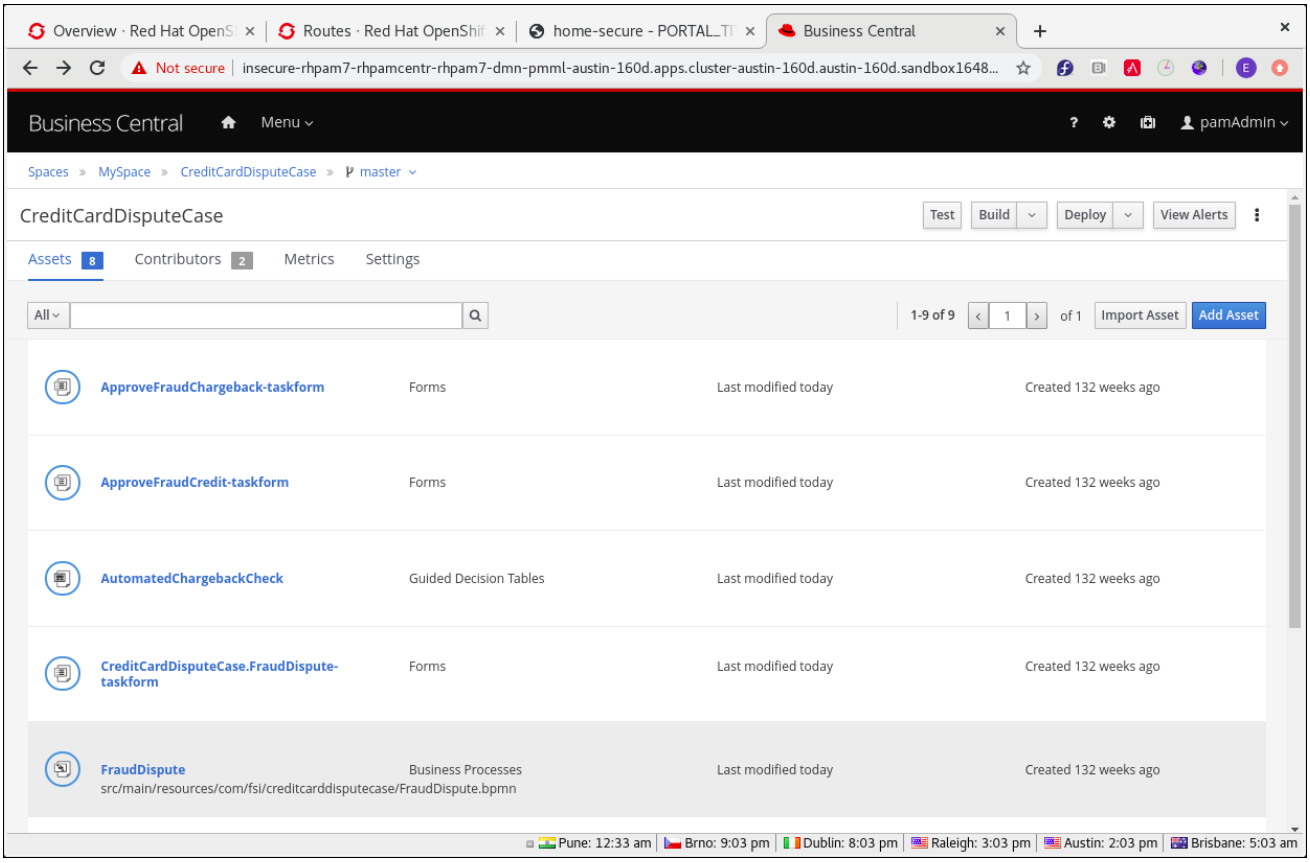
1. 您为什么会接受这些交易？
2. 您的卡是否全时过？
3. 您是否有任何其他方面告诉我们这个争议？

在 Joe 回答问题后，网站为 Joe 提供了事件编号 000004。

现在，Fortress 银行必须决定是否在未经人工调查，还是手动调查申索。手动调查需要更多资源，从而自动处理银行在人力资源方面的成本较低。但是，如果银行自动接受所有争议的金额，由于用于欺诈的申索，成本最终将给银行更大。无论是否要调查对象，还是需要做出决定。

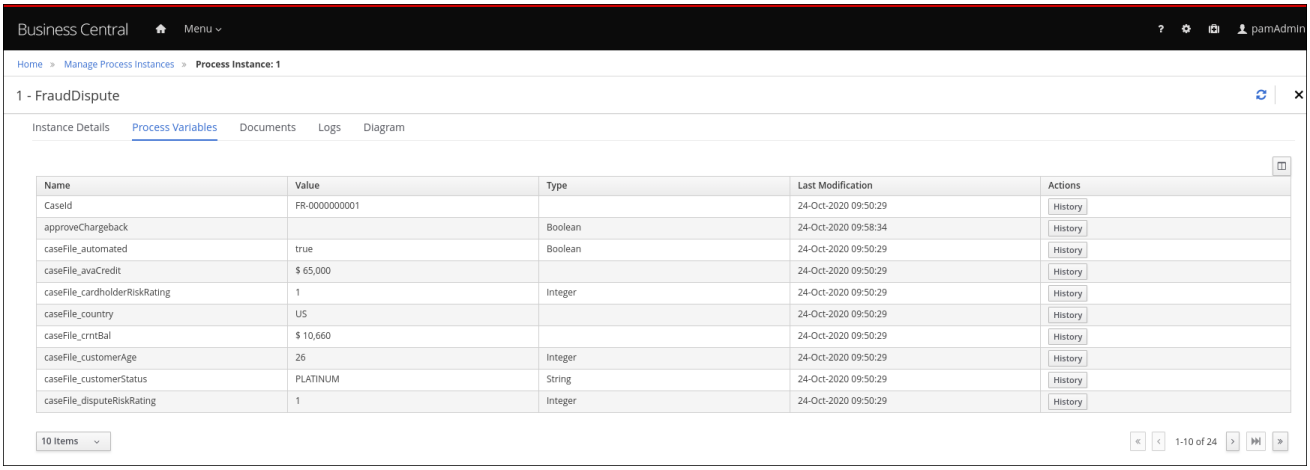
信用卡 Dispute 项目

为了帮助这一决策，Fortress 银行使用 Business Central 创建具有商业中心的 VarnishCardDisputeCase 项目，其中包含模型争议过程的 fraudDispute 业务流程。



进程变量

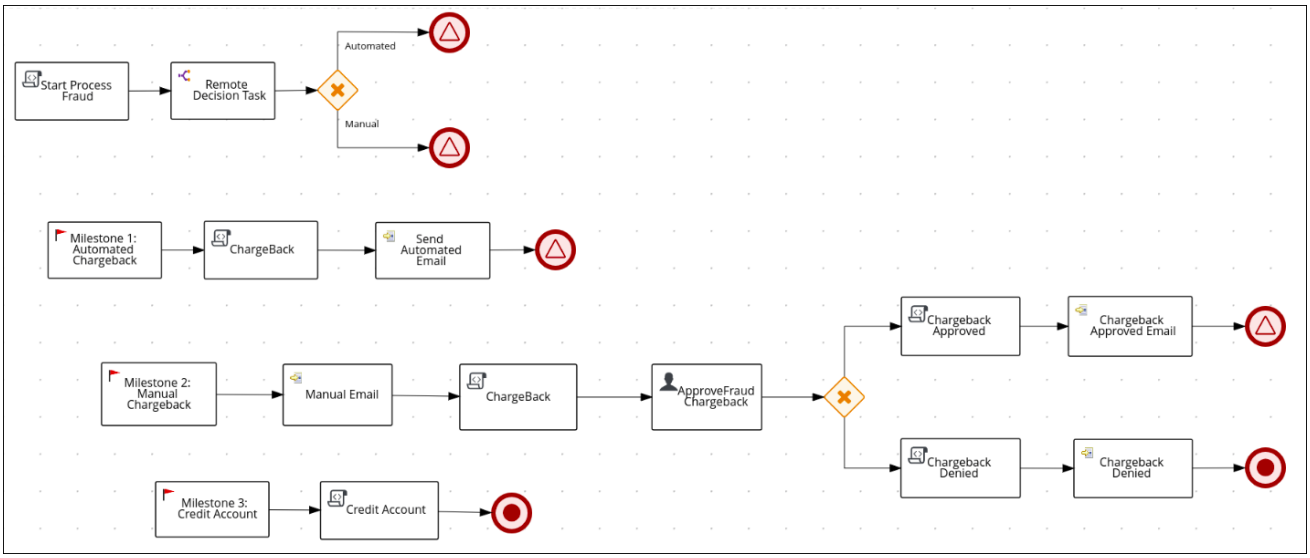
当 Joe 报告了争议时，使用问题单 ID FR-00000004 创建 fraudDispute 进程的实例。Process Variables 选项卡包含多个特定于 Joe 帐户的变量和值，包括 CaseID、caseFile_cardholderRiskRiskRiskRating （信用卡拥有者风险评级）和 caseFile_disputeRiskRating （这种披露的风险评级）：



*该表还具有带有值 **true 的 casefile_automated** 变量。这代表争议满足要自动处理的条件。*

进程图

Diagram 选项卡包含 BPMN 图，显示银行在决定自动或手动处理的决策路径时：

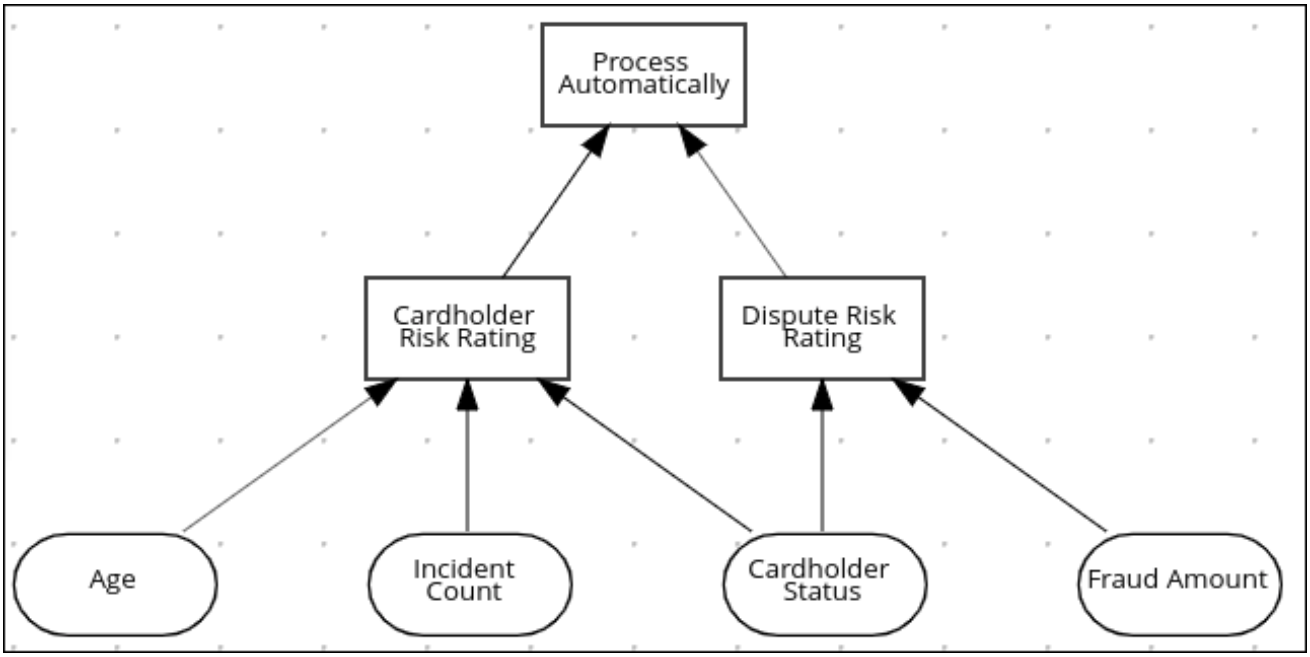


Decision Task 任务包含规则，它会根据 caseFile_cardholderRiskRating 和 caseFile_disputeRiskating 变量以及 caseFile_disputeRisk ating 变量自动收费。如果 Joe 的争论与自动批准条件匹配，则M Milestone 1 跟着批准，争取的金额会被退还给客户。这种子流程非常精益且高效。如果 Joe 的争议需要手动评估，Me Milestone 2 的子进程将启动，这需要某些人人参与，并需要更多的资源来处理。

在这种情况下，决策任务任务 决定处理 Joe 的吞吐量，因此它遵循 Milestone 1: Automatic the。

DMN 模型

以下简化的 DMN 模型显示作为 fraudDispute Decision 任务 一部分的决策过程：



输入变量为 **Age （通配符年龄）、Incident Count （此卡拥有者的前声数）、卡拥有者状态（silver、gold、platinum）和 Fraud Amount。**

Red Hat Process Automation Manager 在决策表中使用输入变量来支持数字决策。决策表由人类业务分析创建。分析师创建案例商业要求分析文档或电子表格，供用户进行审查和审批。然后，项目设计人员使用 Business Central DMN 编辑器将分析文档中的数据传输到 DMN 模型。Fortress Bank Cost Card Dispute process 有两个决定表，即 卡拥有者风险 等级表和 Dispute Risk Rating 表。Cardholder Risk Rating 表包含三个输入变量： Incident Count、cardholder Status 和 Age。Dispute Risk Rating 表包含 卡拥有者状态 输入变量。表根据 cardholder 状态和吞吐量的数量计算 dispute 的风险。

Cardholder Risk Rating *(Decision Table)*

C+	Incident Count *(number)*	Cardholder Status *(string)*	Age *(number)*	Cardholder Risk Rating *(number)*
1	> 3	"PLATINUM"	-	1
2	> 2	"GOLD"	-	1
3	> 2	"SILVER"	-	2
4	> 2	"STANDARD"	-	3
5	-	"SILVER"	< 25	1
6	-	"STANDARD"	< 25	2
7	-	"STANDARD"	>= 25	1
8	-	-	-	0

Dispute Risk Rating *(Decision Table)*

U	Cardholder Status *(string)*	Fraud Amount *(number)*	Dispute Risk Rating *(number)*	Description
1	"STANDARD"	< 25	1	
2	"SILVER"	< 50	1	
3	"GOLD"	< 75	1	
4	"PLATINUM"	< 100	1	
5	"STANDARD"	[25..150)	3	
6	"SILVER"	[50..150)	2	
7	"GOLD"	[75..150)	2	
8	"PLATINUM"	[100..150)	2	
9	"STANDARD"	[150..200)	4	
10	"SILVER"	[150..200)	3	
11	"GOLD"	[150..200)	2	
12	-	>= 200	5	

- ***卡持有者**风险**等**级*

  ***joe 是 25 倍的 Silver 卡**拥**有者。他**曾有**两**个争**议，**因此**风险**等**级为 **2。如果 Joe 之前没有争**议，*则风险评级为 **0。**

- ***争**议风险**等**级*

  由于 Joe 是 Silver 卡**拥**有者，因此Joe 在 Dispute Risk Rating 表上有一个 1** 评级为 **1。如果数量**达**到 $140，Joe 将存在**风险评级为 **2。

以下公式是作**为 **DMN 模型中** 进**程自**动 **最**终**决策的一部分**实现**的，使用**两**个决策表中的分数来确定是否自**动**收取争**议**的金**额**(Milestone1)，**还**是需要更多的**调查**(1)。

> *(Cardholder Risk Rating + Dispute Risk Rating) < 5*

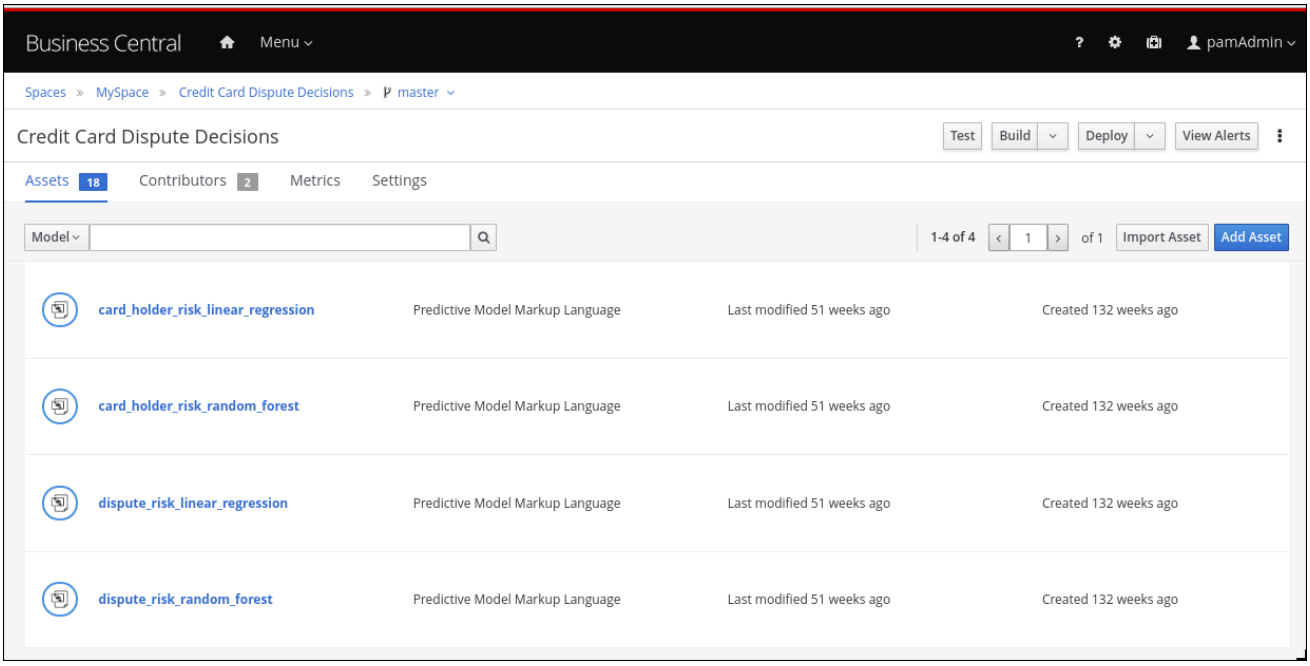
如果 Joe 的**总**体**风险**分数小于 5，**则**会自**动**收**费**其数量(Milestone 1)。如果**总**分数**为 **5 或更高，**则**其争**议**会手**动**处**理**(Milestone 2)。

***在 Red Hat Process Automation Manager 中添加机器学**习*

由于 Fortress Bank** 对**客**户**有**历**史数据，包括以前的交易和争**议历**史，因此**该银**行可通**过**机器学**习**来

创建可在 DMN 模型决策任务中使用的预测模型。与业务分析师创建的决策表相比，这会产生更加准确的评估风险。

fortress Bank 具有两组 PMML 文件，其中包含可更准确评估风险预测的模型。一个集合基于线性回归算法，另一个基于随机的林算法。



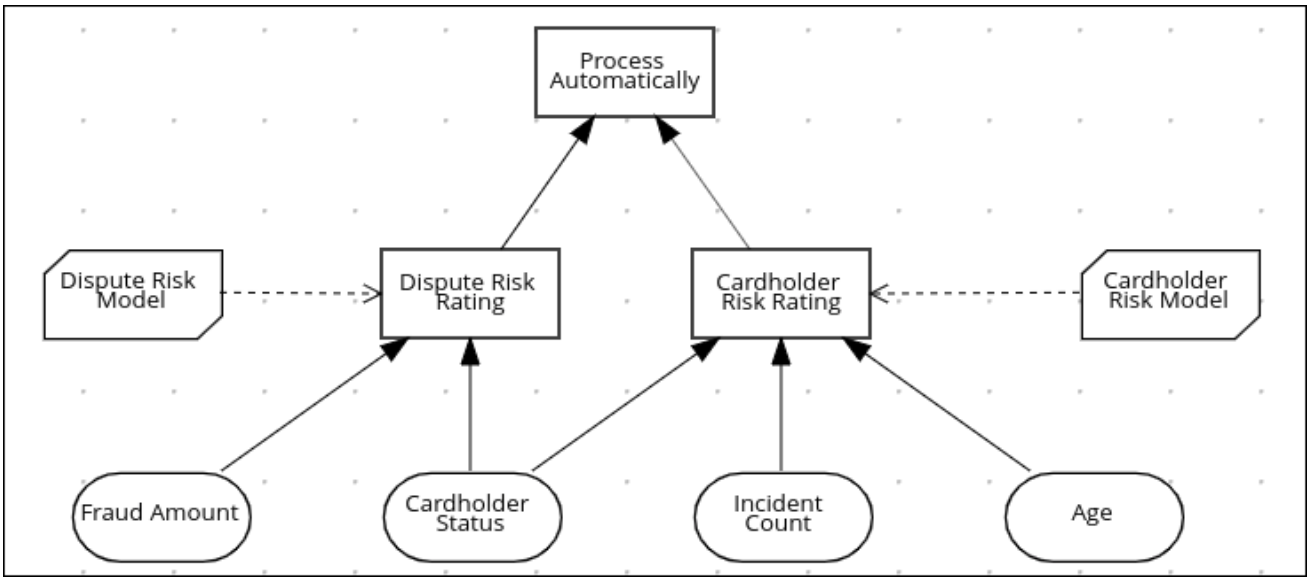
线性回归是统计数据和机器学习中最广泛使用的算法之一。它使用一个线性 equation 来组合一组数字输入和输出值。随机林将许多决策树用作创建预测模型的输入。

添加 PMML 文件

Michelle 将 dispute_risk_linear_regression PMML 文件导入到 her 项目中。她将 卡拥有者风险模型 识别节点添加到 DMN 模型中，并将 dispute_risk_linear_regression PMML 文件与节点相关联。Red Hat Process Automation Manager 分析 PMML 文件，并将输入参数添加到节点。Michelle 将 卡拥有者风险模型 节点与 Dispute Risk Rating 关联。

然后，Michelle 将 credit_card_holder_risk_linear_regression PMML 模型添加到项目中，创建 Dispute Risk Model mode DMN 文件，创建并关联 credit_card_holder_linear_regression PMML 文件。Red Hat Process Automation Manager 分析 PMML 文件，并将输入参数添加到节点。

下图为 Michelle 的已完成 DMN 模型，它可将分析决策表替换为 PMML 文件中的预测模型：



Michelle 现在返回到 fraudDispute BPMN 模型，并使用添加的 PMML 文件更新模型。然后 she 重新部署项目。

增加分数准确度

在这个新场景中，Michelle 重新部署了 Fortress Bank 项目，使用 PMML 模式，我们可以发现 Joe 日志到他的 Fortress Bank 帐户并报告与错误相同的事务时。在 Business Central 中，Michelle 进入 Process Instances 窗口，她看 Joe 的新争议实例。在 Process Variables 选项卡中，Michelle 审查了卡HolderRiskRating 的值以及 争议的RiskRating。它们已改变，因为模型现在使用 PMML 文件。这通过使用基于历史数据的机器学习模型提供更加精确的风险预测。同时，银行策略仍由 DMN 决策模型实施：风险预测低于指定阈值，允许此争议自动处理。

Business Central Menu

Home » Manage Process Instances » **Process Instance: 4**

4 - FraudDispute

Instance Details | Process Variables | Documents | Logs | Diagram

Name	Value	Type
CaseId	FR-0000000004	
approveChargeback		Boolean
caseFile_automated	true	Boolean
caseFile_avaCredit	$ 65,000	
caseFile_cardholderRiskRating	1.753980106953382	Integer
caseFile_country	US	
caseFile_crntBal	$ 10,660	
caseFile_customerAge	26	Integer
caseFile_customerStatus	PLATINUM	String
caseFile_disputeRiskRating	0.37582391437511786	Integer

*监**控***

最后，Fortress Bank 使用 Prometheus 收集有关信用卡争端和 Grafana 的指**标，**以**实时**可**视**化**这些指**标。**monitor 的上半部分**显**示**业务**指**标**关**键**性能指**标**(KPI)，下面部分**显**示了操作指**标** KPI。

93.1. 使用带有 DMN 模型的 PMML 模型来解决信用卡事务争端

这个示例演示了如何使用 Red Hat Process Automation Manager 创建使用 PMML 模型解决信用卡事务争端的 DMN 模型。当客户争取信用卡交易时，系统会决定是否自动处理事务。

先决条件

- Red Hat Process Automation Manager 可用，以下 JAR 文件已添加到 ~/kie-server.war/WEB-INF/lib 和 ~/business-central.war/WEB-INF/lib 目录中：
  - kie-dmn-jpmml-7.67.0.Final-redhat-00024.jar

    此文件位于红帽客户门户网站的软件 下载页面中 的 Red Hat Decision Manager 7.13 Maven 存储库分发中（需要登录）。此文件的组 ID、工件 ID 和版本(GAV)标识符是 org.kie:kie-dmn-jpmml:7.67.0.Final-redhat-00024。如需更多信息，请参阅 Business Central 的 DMN 文件中的"Including PMML 模型"部分 使用 DMN 模型设计决策服务。
  - JPMML Evaluator 1.5.1 JAR 文件
  - JPMML Evaluator Extensions 1.5.1 JAR 文件

    在 KIE 服务器和 Business Central 中启用 JPMML 评估需要这些文件。

重要

红帽支持与 PMML(JPMML)的 Java Evaluator API 集成，以便在 Red Hat Process Automation Manager 中进行 PMML 执行。但是，红帽不支持 JPMML 库。如果您在 Red Hat Process Automation Manager 发行版本中包括 JPMML 库，请参阅 JPMML 的 Openscoring.io 许可条款。

流程

1. 创建 dtree_risk_predictor.pmml 文件，其中包含 第 93.2 节 “信用卡事务事务练习 PMML 文件” 中 XML 示例的内容。
2. 在 Business Central 中，创建 telemetry 卡 Dispute 项目：

a.

导航到 **Menu** → **Design** → **Projects**。

b.

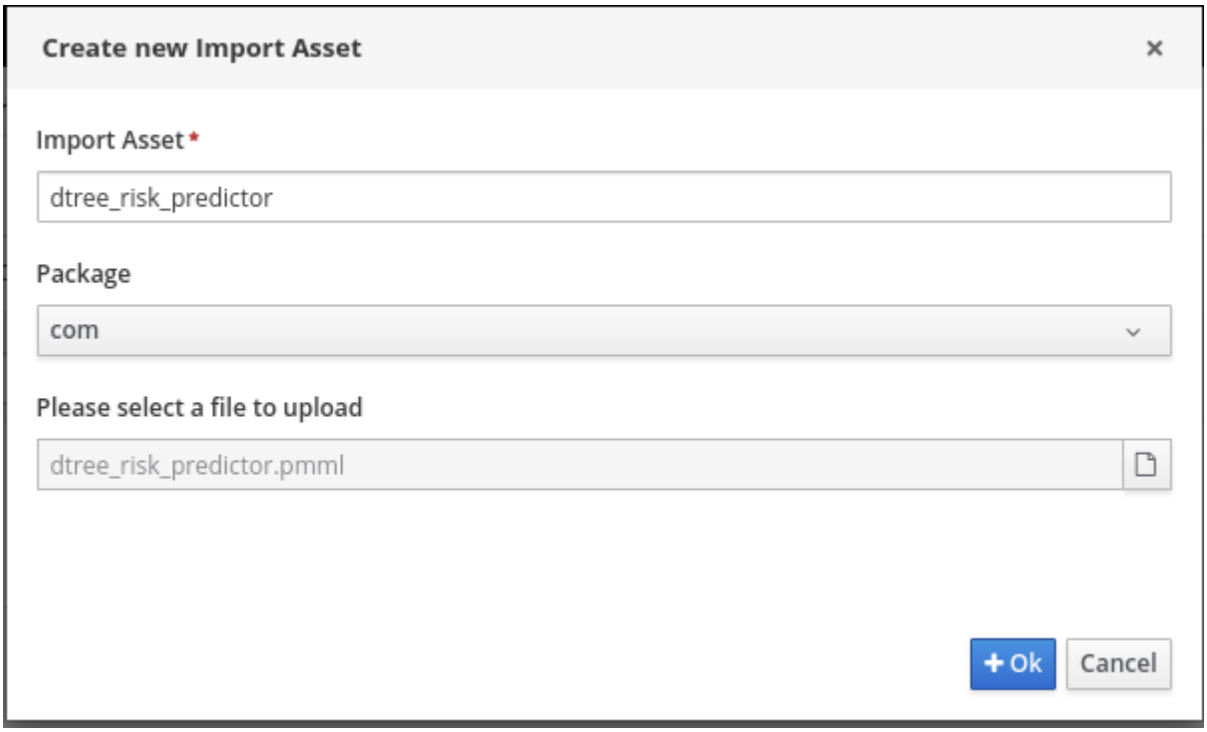
单击 **Add Project**。

c.

在 **Name** 框中，输入 **credit Card Dispute** 并点 **Add**。

3.

在转换卡 **Dispute** 项目的 **assets** 窗口中，将 **dtree_risk_predictor.pmml** 文件导入 **com** 软件包：



a.

单击 **Import Asset**。

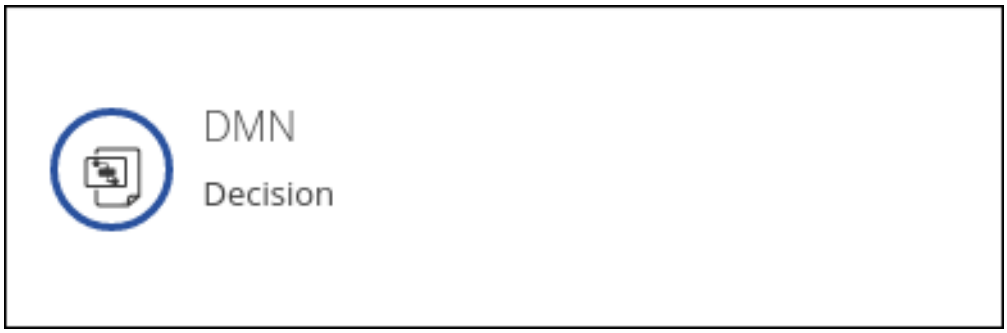
b.

在 **Create new Import Asset** 对话框中，在 **Name** 框中输入 **dtree_risk_predictor**，从 **Package** 菜单中选择 **com**，选择 **dtree_risk_predictor.pmml** 文件，然后单击 **OK**。

dtree_risk_predictor.pmml 文件的内容会出现在 **Overview** 窗口中。

4.

在 **com** 软件包中创建 **Dispute Transaction Check DMN** 模型：



a. *要返回项目窗口，请在面包导航栏尾部点击 credit Card Dispute。*

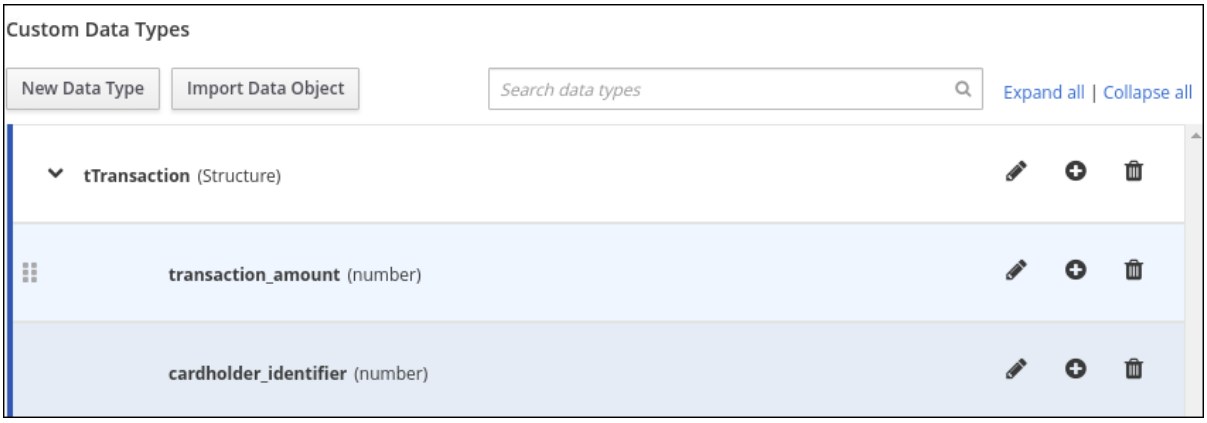
b. *点 Add Asset。*

c. *在资产库中点 DMN。*

d. *在 Create new DMN 对话框中，在 Name 框中输入 Dispute Transaction Check，从 Package 菜单中选择 com，然后单击 OK。*

DMN 编辑器会打开，并附带 Dispute Transaction Check DMN 模型。

5. *创建 tTransaction 自定义数据类型：*



a. *点 Data Types 选项卡。*

b.

点 Add a custom Data Type。

c.

在 Name 框中输入 tTransaction。

d.

从 Type 菜单中选择 Structure。

e.

要创建数据类型，请单击检查标记。

tTransaction 自定义数据类型显示一个变量行。

f.

在变量行的 Name 字段中输入 transaction_amount，从 Type 菜单中选择 Number，然后单击复选标记。

g.

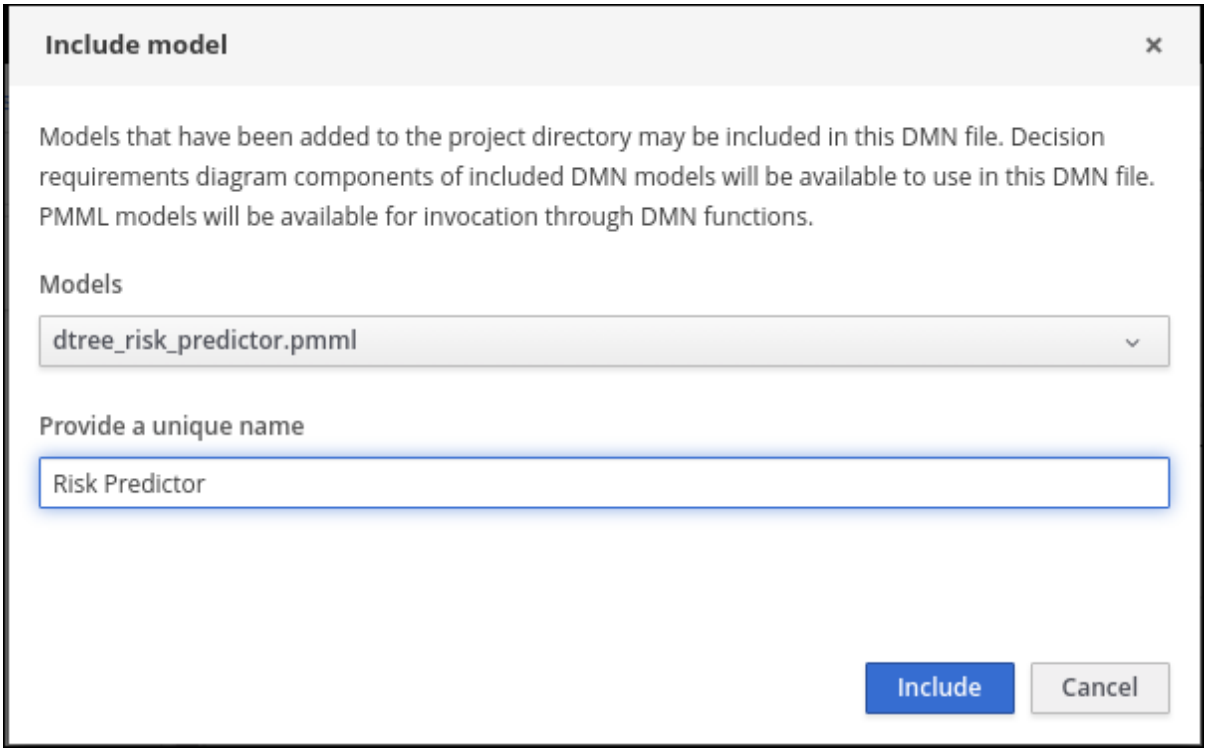
要添加新变量行，请单击 transaction_amount 行上的加号符号。此时会出现一个新行。

h.

在 Name 字段中输入 cardholder_identifier，从 Type 菜单中选择 Number，然后点检查标记。

6.

添加 风险 Predictor dtree_risk_predictor.pmml 模型：

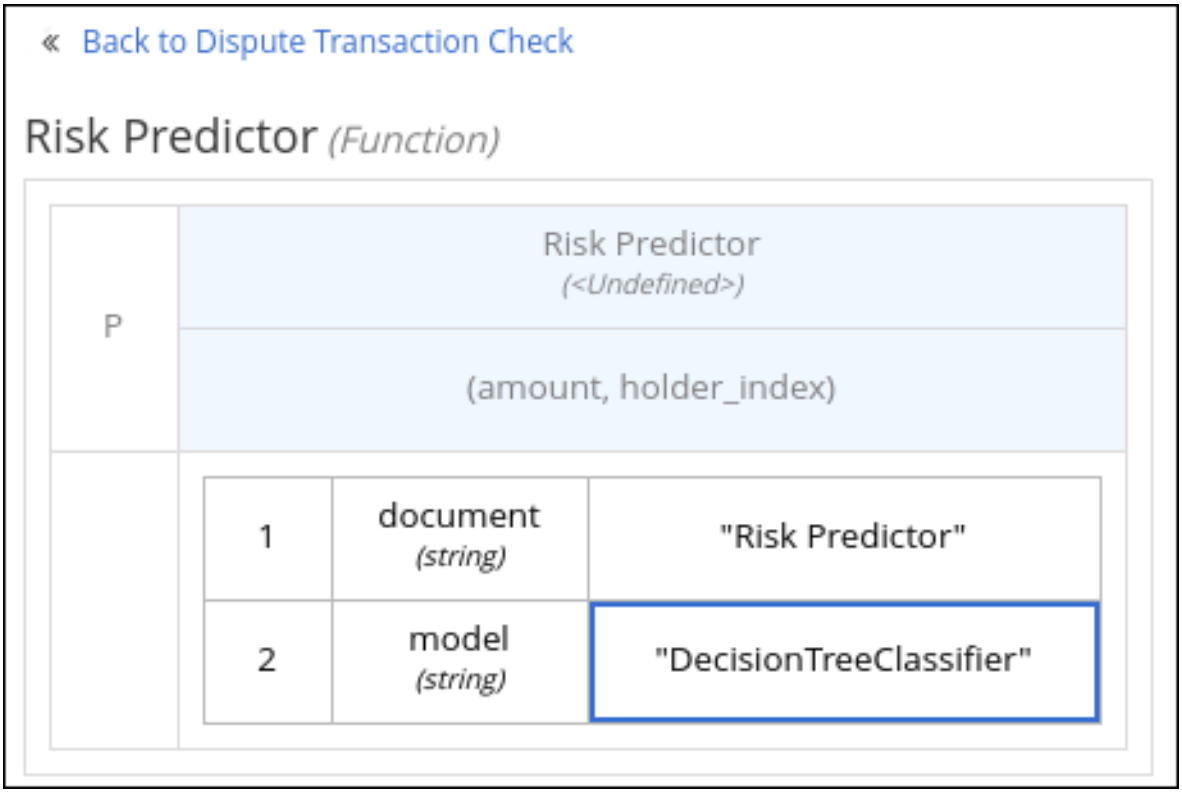


a. ***在 DMN 编辑器的 已包含模型 窗口中，点 Include Model。***

b. ***在 Include Model 对话框中，从 Models 菜单中选择 dtree_risk_predictor.pmml。***

c. ***在 Provide a unique name box 中输入 Risk Predictor，然后单击 OK。***

7. ***使用 风险预测器和 DecisionTreeClassifier 模型创建 风险预测 商业知识模型(BKM)节点：***



a.

在 DMN 编辑器的模型窗口中，将 BKM 节点拖到 DMN 编辑器面板。

b.

重命名节点 风险预测。

c.

点节点左侧的 trash can 图标下的编辑图标。

d.

在 Risk Predictor 框中点 F，然后从 Select Function Kind 菜单中选择 PMML。F 对 P 的更改。

e.

双击 First select PMML 文档 框，然后选择 Risk Predictor。

f.

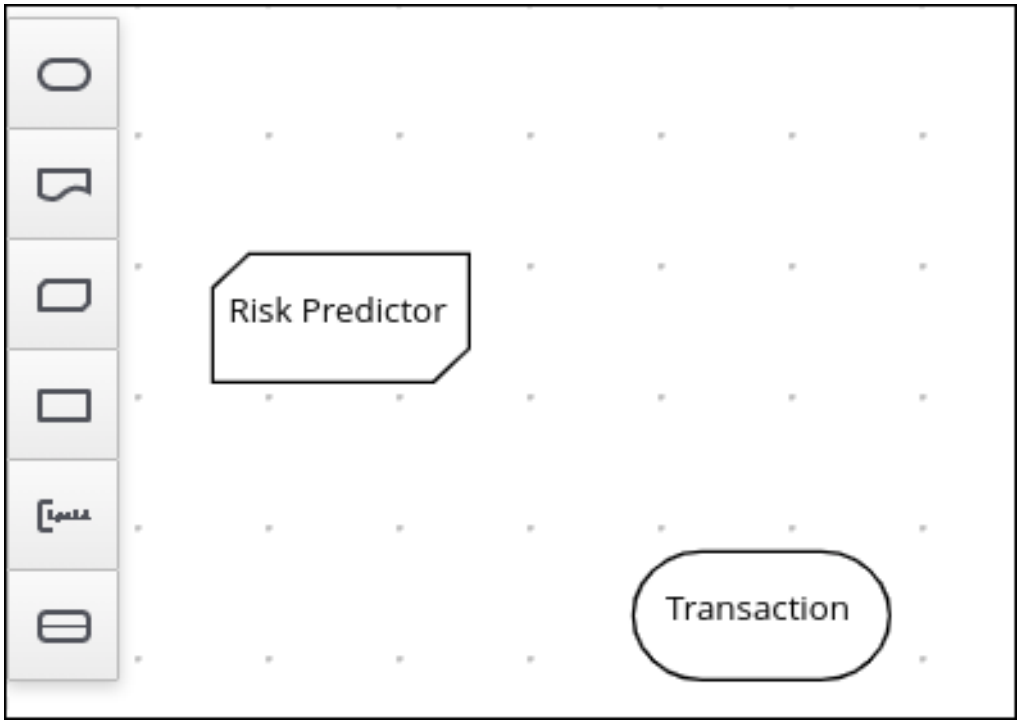
双击 Second select PMML model 框，然后选择 DecisionTreeClassifier。

g.

要返回 DMN 编辑器面板，请点击 Back to Dispute Transaction Check。

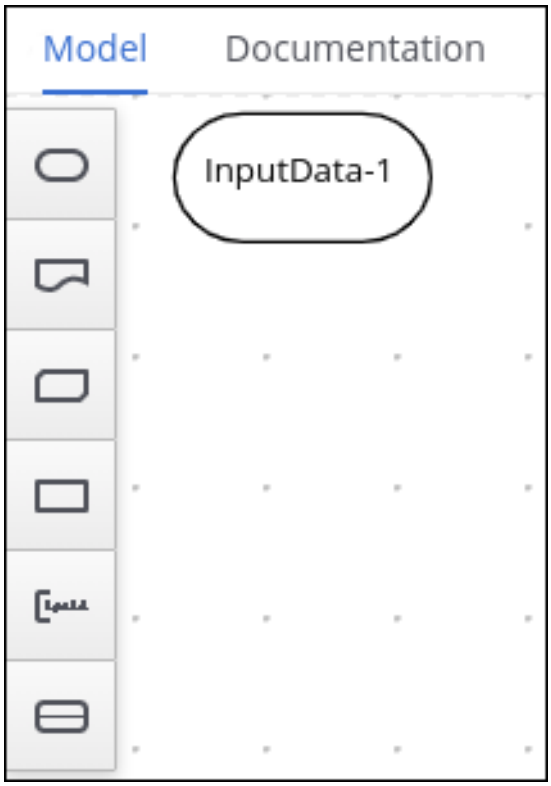
8.

使用数据类型 t Transaction 创建事务输入数据节点：



a.

在 DMN 编辑器的 Model 窗口中，将输入数据节点拖到 DMN 编辑器面板。



b.

重命名节点 事务。

c.

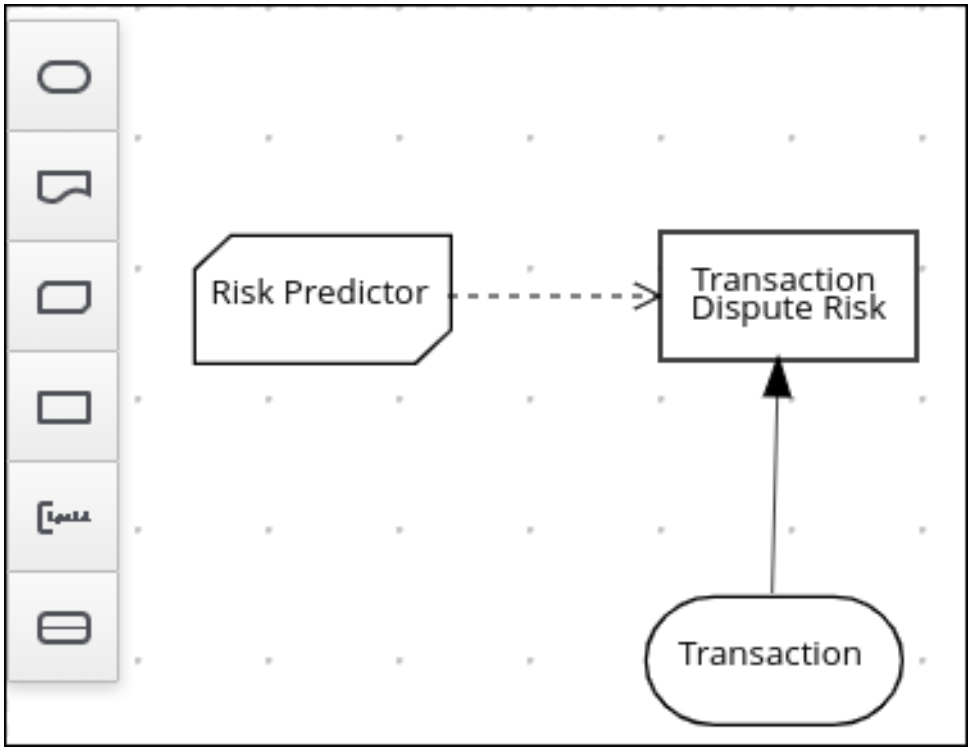
选择节点，然后单击窗口右上角的属性铅笔图标。

d.

在 Properties 面板中，选择 Information Item → Data type → tTransaction，然后关闭面板。

9.

创建 事务风险 决策节点，并为该功能添加 事务 节点以进行数据输入 和风险预测 节点：



a.

在 DMN 编辑器的模型窗口中，将决策数据节点拖到 DMN 编辑器面板。

b. *重命名节点 事务 **Dispute Risk**。*

c. *选择 风险预测 节**点，并将箭**头**从**节**点右上角拖到 Transaction Dispute risk** 节**点。***

d. *选择 **事**务 节**点并将箭**头**从**节**点右**边**拖到 事**务 **Dispute risk** 节**点。***

10. ***在 事**务 **Dispute Risk node 中，** 创**建** 风险预测**器** 调**用功能：***

#	Transaction Dispute Risk *(<Undefined>)*	
	Risk Predictor	
1	amount *(number)*	Transaction.transaction_amount
2	holder_index *(number)*	Transaction.cardholder_identifier

a. 选择事务 Dispute Risk 节点，再单击节点左侧的编辑图标。

b. 点 Select expression，再从菜单中选择 Invocation。

c. 在 Enter 功能框中输入风险预测器。

d. 点 P1。

e. 在 Edit Parameter 对话框中，在 Name 框中输入 数量，从 Data Type 菜单中选择数字，然后按 Enter 键。

f. 单击 Select expression，再从菜单中选择 Literal 表达式。

g. 在数量旁边的复选框，输入 Transaction.transaction_ amount 。

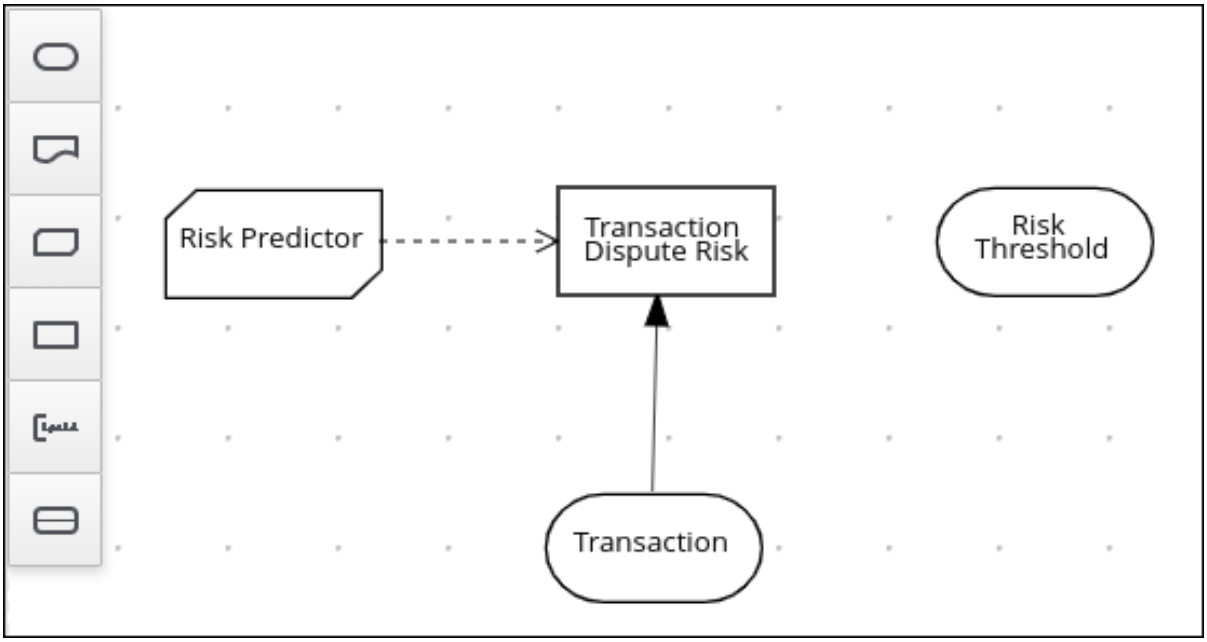
h. 右键单击 1，然后在下面选择" 插入"。这会打开 Edit Parameter 对话框。

i. 在 Name 框中输入 holder_index，从 Data Type 菜单中选择 number，然后按 Enter 键。

j. 单击行 2 上的 Select 表达式，再从菜单中选择 Literal 表达式。

k. 在 数量旁边的框中输入 Transaction.cardholder_identifier。

11. 使用数据类型 号 创建 Risk Threshold 输入数据节点：



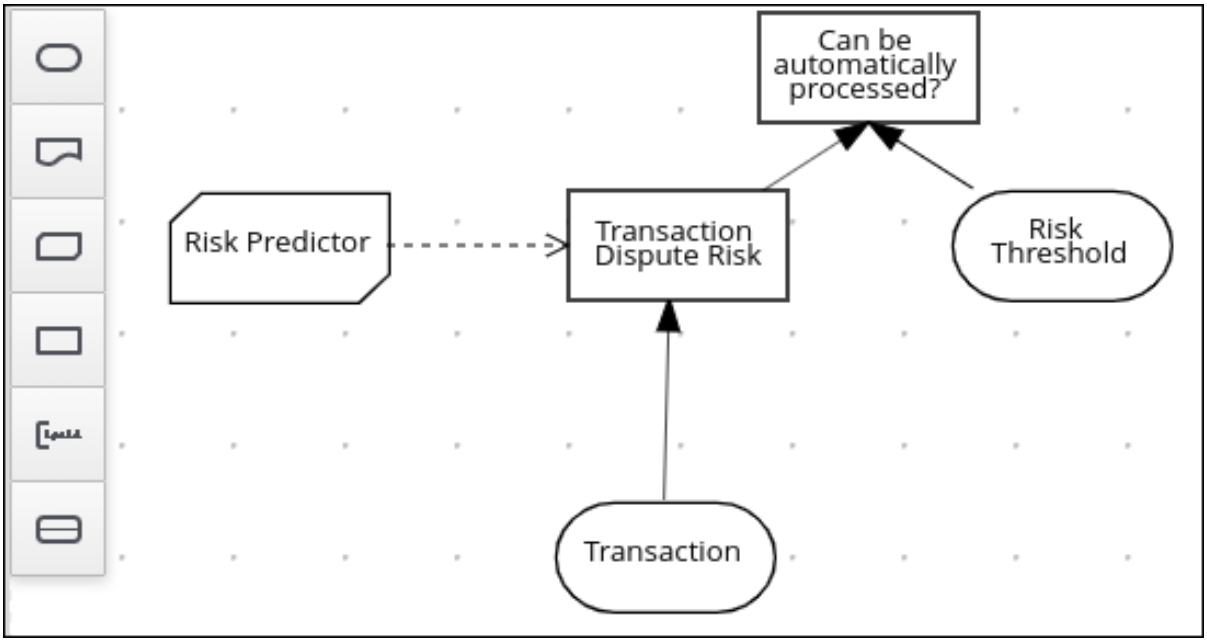
a. ***在 DMN 编辑器的 Model 窗口中，将输入数据节点拖到 DMN 编辑器面板。***

b. ***重命名节点 风险阈值。***

c. *选择节点，然后单击窗口右上角的属性铅笔图标。*

d. ***在 Properties 面板中，选择 Information Item → Data type → number，然后关闭面板。***

12. *创建可自动处理？决定节点作为输入事务 Dispute 风险和风险阈值 节点：*



a. 将决策节点拖到 DMN 编辑器面板并将其重命名为 可自动处理？

b. 选择该节点，然后单击节点左上角的编辑图标。

c. 单击 Select expression，然后从菜单选择 Literal 表达式。

d. 在方框中输入 Transaction Dispute Risk.predicted_dispute_ risk < Risk Threshold。

e. 选择 事务 Dispute Risk 节点，将节点左上角的箭头拖到 可以自动处理？ 节点。

f. 选择 Risk Threshold 节点，将箭头从节点左下角拖到 可以自动处理？ 节点。

13. 保存模型并构建项目：

a. 在 DMN 编辑器中，点 Save。

b. 如有必要，更正出现的任何错误。

c.

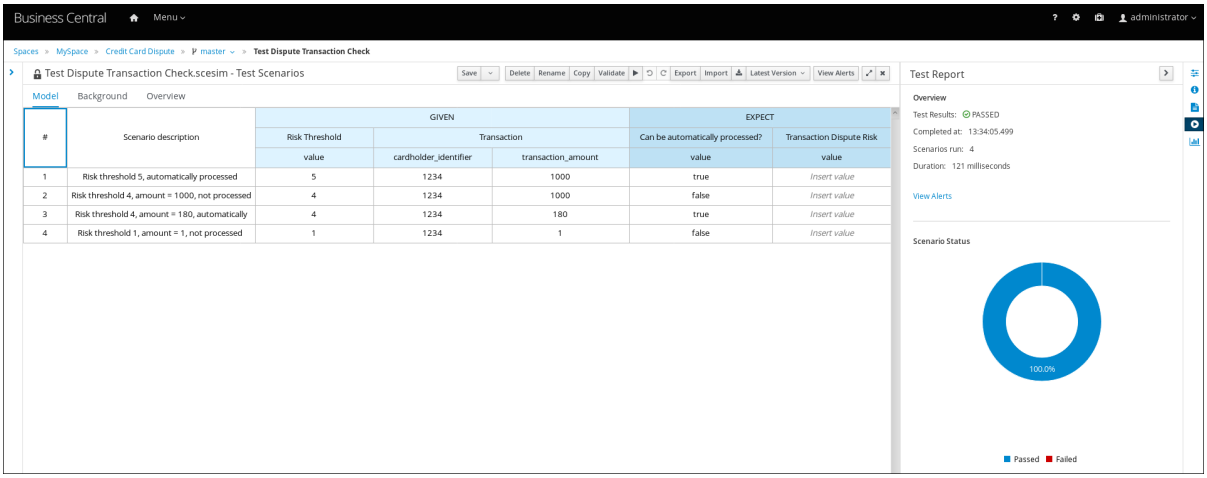
要返回项目窗口，请在面包导航栏尾部点击 credit Card Dispute。

d.

单击 Build。该项目应该能成功构建。

14.

添加并运行测试场景：



a.

点 Add Asset。

b.

选择 Test Scenario。

c.

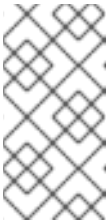
在 Create new Test Scenario 对话框中，输入名称 Test Dispute Transaction Check，从 Package 菜单中选择 com，然后选择 DMN。

d.

从选择 DMN 资产菜单中选择 Dispute Transaction Check.dmn 并点 OK。测试模板构建。

e.

输入以下值并点击 Save ：



注意

不要在 Transaction Dispute Risk 列中添加一个值。这个值由测试场景决定。

表 93.1. 测试场景参数

描述	risk Threshold	cardholder_identifier	transaction_amount	是否可以自动处理？
风险阈值 5，自动处理	5	1234	1000	true
风险阈值 4、数量 = 1000，没有处理	4	1234	1000	false
风险阈值 4、数量 = 180，自动处理	4	1234	180	true
风险阈值 1、数量 = 1，未处理	1	1234	1	false

f. 要运行测试，请单击 **Validate** 右侧的 **Play** 按钮。结果会显示在屏幕右侧的 **Test Report** 面板中。

93.2. 信用卡事务事务练习 PMML 文件

使用以下 XML 内容在 **第 93.1 节 “使用带有 DMN 模型的 PMML 模型来解决信用卡事务争端” 练习**中创建 **dtree_risk_predictor.pmml** 文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML xmlns="http://www.dmg.org/PMML-4_2" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="4.2" xsi:schemaLocation="http://www.dmg.org/PMML-4_2 http://www.dmg.org/v4-2-1/pmml-4-2.xsd">
  <Header copyright="Copyright (c) 2018 Software AG" description="Default Description">
    <Application name="Nyoka" version="4.3.0" />
    <Timestamp>2020-10-09 14:27:26.622723</Timestamp>
  </Header>
  <DataDictionary numberOfFields="3">
    <DataField name="amount" optype="continuous" dataType="double" />
    <DataField name="holder_index" optype="continuous" dataType="double" />
    <DataField name="dispute_risk" optype="categorical" dataType="integer">
      <Value value="1" />
      <Value value="2" />
      <Value value="3" />
      <Value value="4" />
      <Value value="5" />
    </DataField>
  </DataDictionary>
  <TreeModel modelName="DecisionTreeClassifier" functionName="classification" missingValuePenalty="1.0">
```

```
    <MiningSchema>
      <MiningField name="amount" usageType="active" optype="continuous" />
      <MiningField name="holder_index" usageType="active" optype="continuous" />
      <MiningField name="dispute_risk" usageType="target" optype="categorical" />
    </MiningSchema>
    <Output>
      <OutputField name="probability_1" optype="continuous" dataType="double" feature="probability"
value="1" />
      <OutputField name="probability_2" optype="continuous" dataType="double" feature="probability"
value="2" />
      <OutputField name="probability_3" optype="continuous" dataType="double" feature="probability"
value="3" />
      <OutputField name="probability_4" optype="continuous" dataType="double" feature="probability"
value="4" />
      <OutputField name="probability_5" optype="continuous" dataType="double" feature="probability"
value="5" />
      <OutputField name="predicted_dispute_risk" optype="categorical" dataType="integer"
feature="predictedValue" />
    </Output>
    <Node id="0" recordCount="600.0">
      <True />
      <Node id="1" recordCount="200.0">
        <SimplePredicate field="amount" operator="lessOrEqual" value="99.94000244140625" />
        <Node id="2" score="2" recordCount="55.0">
          <SimplePredicate field="holder_index" operator="lessOrEqual" value="0.5" />
          <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
          <ScoreDistribution value="2" recordCount="55.0" confidence="1.0" />
          <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
          <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
          <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
        </Node>
        <Node id="3" score="1" recordCount="145.0">
          <SimplePredicate field="holder_index" operator="greaterThan" value="0.5" />
          <ScoreDistribution value="1" recordCount="145.0" confidence="1.0" />
          <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
          <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
          <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
          <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
        </Node>
      </Node>
      <Node id="4" recordCount="400.0">
        <SimplePredicate field="amount" operator="greaterThan" value="99.94000244140625" />
        <Node id="5" recordCount="105.0">
          <SimplePredicate field="holder_index" operator="lessOrEqual" value="0.5" />
          <Node id="6" score="3" recordCount="54.0">
            <SimplePredicate field="amount" operator="lessOrEqual" value="150.4550018310547" />
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="3" recordCount="54.0" confidence="1.0" />
            <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
          </Node>
          <Node id="7" recordCount="51.0">
            <SimplePredicate field="amount" operator="greaterThan" value="150.4550018310547" />
            <Node id="8" recordCount="40.0">
              <SimplePredicate field="amount" operator="lessOrEqual" value="200.00499725341797" />
```

```
          <Node id="9" recordCount="36.0">
            <SimplePredicate field="amount" operator="lessOrEqual" value="195.4949951171875" />
            <Node id="10" recordCount="2.0">
              <SimplePredicate field="amount" operator="lessOrEqual" value="152.2050018310547" />
              <Node id="11" score="4" recordCount="1.0">
                <SimplePredicate field="amount" operator="lessOrEqual" value="151.31500244140625"
/>
                <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="4" recordCount="1.0" confidence="1.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
              </Node>
              <Node id="12" score="3" recordCount="1.0">
                <SimplePredicate field="amount" operator="greaterThan" value="151.31500244140625"
/>
                <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
                <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
              </Node>
            </Node>
            <Node id="13" recordCount="34.0">
              <SimplePredicate field="amount" operator="greaterThan" value="152.2050018310547" />
              <Node id="14" recordCount="20.0">
                <SimplePredicate field="amount" operator="lessOrEqual" value="176.5050048828125"
/>
                <Node id="15" recordCount="19.0">
                  <SimplePredicate field="amount" operator="lessOrEqual" value="176.06500244140625"
/>
                  <Node id="16" score="4" recordCount="9.0">
                    <SimplePredicate field="amount" operator="lessOrEqual" value="166.6449966430664"
/>
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="4" recordCount="9.0" confidence="1.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                  </Node>
                  <Node id="17" recordCount="10.0">
                    <SimplePredicate field="amount" operator="greaterThan" value="166.6449966430664"
/>
                    <Node id="18" score="3" recordCount="1.0">
                      <SimplePredicate field="amount" operator="lessOrEqual"
value="167.97999572753906" />
                      <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
                      <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                    </Node>
                    <Node id="19" score="4" recordCount="9.0">
                      <SimplePredicate field="amount" operator="greaterThan"
value="167.97999572753906" />
                      <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
```

```
                <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="4" recordCount="9.0" confidence="1.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
               </Node>
              </Node>
             </Node>
             <Node id="20" score="3" recordCount="1.0">
              <SimplePredicate field="amount" operator="greaterThan" value="176.06500244140625"
/>
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
              <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
             </Node>
            </Node>
            <Node id="21" score="4" recordCount="14.0">
             <SimplePredicate field="amount" operator="greaterThan" value="176.5050048828125"
/>
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="4" recordCount="14.0" confidence="1.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
             </Node>
            </Node>
           </Node>
           <Node id="22" recordCount="4.0">
            <SimplePredicate field="amount" operator="greaterThan" value="195.4949951171875" />
            <Node id="23" score="3" recordCount="1.0">
             <SimplePredicate field="amount" operator="lessOrEqual" value="195.76499938964844"
/>
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
              <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
            <Node id="24" recordCount="3.0">
             <SimplePredicate field="amount" operator="greaterThan" value="195.76499938964844"
/>
             <Node id="25" score="4" recordCount="1.0">
              <SimplePredicate field="amount" operator="lessOrEqual" value="196.74500274658203"
/>
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="4" recordCount="1.0" confidence="1.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
             </Node>
             <Node id="26" recordCount="2.0">
              <SimplePredicate field="amount" operator="greaterThan" value="196.74500274658203"
/>
              <Node id="27" score="3" recordCount="1.0">
               <SimplePredicate field="amount" operator="lessOrEqual" value="197.5800018310547"
```

```
/>
                <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="3" recordCount="1.0" confidence="1.0" />
                <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
               </Node>
               <Node id="28" score="4" recordCount="1.0">
                <SimplePredicate field="amount" operator="greaterThan" value="197.5800018310547"
/>
                <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                <ScoreDistribution value="4" recordCount="1.0" confidence="1.0" />
                <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
               </Node>
              </Node>
             </Node>
            </Node>
           </Node>
           <Node id="29" score="5" recordCount="11.0">
            <SimplePredicate field="amount" operator="greaterThan" value="200.00499725341797" />
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="5" recordCount="11.0" confidence="1.0" />
           </Node>
          </Node>
         </Node>
         <Node id="30" recordCount="295.0">
          <SimplePredicate field="holder_index" operator="greaterThan" value="0.5" />
          <Node id="31" score="2" recordCount="170.0">
           <SimplePredicate field="amount" operator="lessOrEqual" value="150.93499755859375" />
           <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
           <ScoreDistribution value="2" recordCount="170.0" confidence="1.0" />
           <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
           <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
           <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
          </Node>
          <Node id="32" recordCount="125.0">
           <SimplePredicate field="amount" operator="greaterThan" value="150.93499755859375" />
           <Node id="33" recordCount="80.0">
            <SimplePredicate field="holder_index" operator="lessOrEqual" value="2.5" />
            <Node id="34" recordCount="66.0">
             <SimplePredicate field="amount" operator="lessOrEqual" value="199.13500213623047" />
             <Node id="35" score="3" recordCount="10.0">
              <SimplePredicate field="amount" operator="lessOrEqual" value="155.56999969482422"
/>
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="3" recordCount="10.0" confidence="1.0" />
              <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
             </Node>
             <Node id="36" recordCount="56.0">
```

```
            <SimplePredicate field="amount" operator="greaterThan" value="155.56999969482422"
/>
            <Node id="37" score="2" recordCount="1.0">
              <SimplePredicate field="amount" operator="lessOrEqual" value="155.9000015258789"
/>
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
              <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
            <Node id="38" recordCount="55.0">
              <SimplePredicate field="amount" operator="greaterThan" value="155.9000015258789"
/>
              <Node id="39" recordCount="31.0">
                <SimplePredicate field="amount" operator="lessOrEqual" value="176.3699951171875"
/>
                <Node id="40" recordCount="30.0">
                  <SimplePredicate field="amount" operator="lessOrEqual"
value="175.72000122070312" />
                  <Node id="41" recordCount="19.0">
                    <SimplePredicate field="amount" operator="lessOrEqual"
value="168.06999969482422" />
                    <Node id="42" recordCount="6.0">
                      <SimplePredicate field="amount" operator="lessOrEqual" value="158.125" />
                      <Node id="43" score="3" recordCount="5.0">
                        <SimplePredicate field="amount" operator="lessOrEqual"
value="157.85499572753906" />
                        <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="3" recordCount="5.0" confidence="1.0" />
                        <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                      </Node>
                      <Node id="44" score="2" recordCount="1.0">
                        <SimplePredicate field="amount" operator="greaterThan"
value="157.85499572753906" />
                        <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                        <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                        <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                      </Node>
                    </Node>
                    <Node id="45" score="3" recordCount="13.0">
                      <SimplePredicate field="amount" operator="greaterThan" value="158.125" />
                      <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="3" recordCount="13.0" confidence="1.0" />
                      <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                    </Node>
                  </Node>
                  <Node id="46" recordCount="11.0">
                    <SimplePredicate field="amount" operator="greaterThan"
value="168.06999969482422" />
```

```
                <Node id="47" score="2" recordCount="1.0">
                  <SimplePredicate field="amount" operator="lessOrEqual"
value="168.69499969482422" />
                  <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                  <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                  <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                  <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                  <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                </Node>
                <Node id="48" recordCount="10.0">
                  <SimplePredicate field="amount" operator="greaterThan"
value="168.69499969482422" />
                  <Node id="49" recordCount="4.0">
                    <SimplePredicate field="holder_index" operator="lessOrEqual" value="1.5" />
                    <Node id="50" score="2" recordCount="1.0">
                      <SimplePredicate field="amount" operator="lessOrEqual"
value="172.0250015258789" />
                      <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                      <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                    </Node>
                    <Node id="51" score="3" recordCount="3.0">
                      <SimplePredicate field="amount" operator="greaterThan"
value="172.0250015258789" />
                      <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="3" recordCount="3.0" confidence="1.0" />
                      <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                      <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                    </Node>
                  </Node>
                  <Node id="52" score="3" recordCount="6.0">
                    <SimplePredicate field="holder_index" operator="greaterThan" value="1.5" />
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="3" recordCount="6.0" confidence="1.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                  </Node>
                </Node>
              </Node>
            </Node>
            <Node id="53" score="2" recordCount="1.0">
              <SimplePredicate field="amount" operator="greaterThan"
value="175.72000122070312" />
              <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
              <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
              <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
            </Node>
          </Node>
          <Node id="54" recordCount="24.0">
            <SimplePredicate field="amount" operator="greaterThan" value="176.3699951171875"
```

```
/>
                <Node id="55" score="3" recordCount="16.0">
                  <SimplePredicate field="amount" operator="lessOrEqual" value="192.0999984741211"
/>
                  <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                  <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                  <ScoreDistribution value="3" recordCount="16.0" confidence="1.0" />
                  <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                  <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                </Node>
                <Node id="56" recordCount="8.0">
                  <SimplePredicate field="amount" operator="greaterThan" value="192.0999984741211"
/>
                  <Node id="57" score="2" recordCount="1.0">
                    <SimplePredicate field="amount" operator="lessOrEqual"
value="192.75499725341797" />
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="1.0" confidence="1.0" />
                    <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                  </Node>
                  <Node id="58" score="3" recordCount="7.0">
                    <SimplePredicate field="amount" operator="greaterThan"
value="192.75499725341797" />
                    <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="3" recordCount="7.0" confidence="1.0" />
                    <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
                    <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
                  </Node>
                </Node>
              </Node>
            </Node>
          </Node>
        </Node>
        <Node id="59" recordCount="14.0">
          <SimplePredicate field="amount" operator="greaterThan" value="199.13500213623047" />
          <Node id="60" score="5" recordCount="10.0">
            <SimplePredicate field="holder_index" operator="lessOrEqual" value="1.5" />
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="5" recordCount="10.0" confidence="1.0" />
          </Node>
          <Node id="61" score="4" recordCount="4.0">
            <SimplePredicate field="holder_index" operator="greaterThan" value="1.5" />
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="4" recordCount="4.0" confidence="1.0" />
            <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
          </Node>
        </Node>
      </Node>
```

```
        <Node id="62" recordCount="45.0">
          <SimplePredicate field="holder_index" operator="greaterThan" value="2.5" />
          <Node id="63" score="2" recordCount="37.0">
            <SimplePredicate field="amount" operator="lessOrEqual" value="199.13999938964844" />
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="37.0" confidence="1.0" />
            <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="4" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
          </Node>
          <Node id="64" score="4" recordCount="8.0">
            <SimplePredicate field="amount" operator="greaterThan" value="199.13999938964844" />
            <ScoreDistribution value="1" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="2" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="3" recordCount="0.0" confidence="0.0" />
            <ScoreDistribution value="4" recordCount="8.0" confidence="1.0" />
            <ScoreDistribution value="5" recordCount="0.0" confidence="0.0" />
          </Node>
        </Node>
      </Node>
    </Node>
  </Node>
 </TreeModel>
</PMML>
```

第 94 章 其他资源

- 问题单管理入门
- 决策服务入门
- 使用 DMN 模型设计决策服务
- 使用 Red Hat Process Automation Manager 开发 Solvers
- 预测 2019 年：预期 Aragmatic Vision of AI

附录 A. 版本信息

文档最新更新于 2024 年 3 月 14 日星期四。

附录 B. 联系信息

Red Hat Process Automation Manager 文档团队：brms-docs@redhat.com