Migration Toolkit for Applications 7.0

规则开发指南

创建自定义规则以增强迁移覆盖。

Last Updated: 2024-05-25

Migration Toolkit for Applications 7.0 规则开发指南

创建自定义规则以增强迁移覆盖。

法律通告

摘要

本指南介绍了如何为应用程序的 Migration Toolkit 创建自定义 XML 规则。

目录

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 简介

1.1. 关于规则开发指南

本指南适用于希望为应用程序(MTA)工具创建自定义基于 YAML 的规则的软件工程师。

详情请参阅 *Migration Toolkit for Applications 简介* 了解概述和 *CLI 指南*。

1.1.1. 本指南使用 <MTA_HOME>

本指南使用 **<MTA_HOME>** 可替换变量来指示您的 MTA 安装的路径。

mta-7.0.3-cli<OS>.zip* 提取一个名为 **mta-cli** 的二进制文件。

在本指南中遇到 **<MTA_HOME>** 时，将其替换为 MTA 安装的实际路径。

1.2. MTA 规则

Migration Toolkit for Applications (MTA)包含基于规则的迁移工具（分析器），可用于分析您计划迁移的应用程序使用的应用程序用户界面(API)、技术和架构。MTA 分析器规则使用以下规则模式：

```
when(condition)
 message(message)
 tag(tags)
```

您可以在内部使用 MTA 规则来执行以下任务：

- 从存档中提取文件.
- 解译文件.
- 扫描和分类文件类型。
- 分析 XML 和其他文件内容。
- 分析应用程序代码。
- 构建报告。

MTA 根据规则执行结果构建数据模型，并将组件数据和关系存储在图形数据库中。然后，可以根据迁移规则和报告目的的要求查询和更新此数据库。

注意

您可以创建自己的自定义分析器规则。您可以使用自定义规则来识别自定义库或可能未被提供的标准迁移规则涵盖的其他组件。

第 2 章 创建 YAML 规则

每个分析器规则是一组用来分析源代码并检测迁移问题的指令。

分析器解析用户提供的规则，将它们应用到应用程序的源代码，并为匹配的规则生成问题。由一个或多个规则的集合组成一个规则集。创建规则集提供了一种方法来组织多个实现常见目标的规则。分析器 CLI 使用规则集作为输入参数。

2.1. YAML 规则结构和语法

MTA 规则使用 YAML 编写。每个规则由元数据、条件和操作组成，指示分析器在给定条件匹配时采取指定操作。

MTA 中的 YAML 规则文件包含一个或多个 YAML 规则。

2.1.1. 规则元数据

规则元数据包含有关规则的常规信息。元数据结构如下：

```
ruleId: "unique_id" (1)
labels: (2)
  # key=value pair
  - "label1=val1"
  # valid label with value omitted
  - "label2"
  # valid label with empty value
  - "label3="
  # subdomain prefixed key
  - "konveyor.io/label1=val1"
effort: 1 (3)
category: mandatory (4)
```

1. 该 ID 在规则所属的规则集中必须是唯一的。
2. 有关标签格式的描述信息，请参见以下。
3. **effort** 是一个整数值，指示解决这个问题所需的工作量程度。
4. **category** 描述了迁移问题的严重性。该值可以是 **强制的**、**可选** 或 **潜在的**。有关这些类别的描述，请参阅 规则类别。

2.1.1.1. 规则标签

标签是为规则或规则集以及依赖项指定的 **key=val** 对。对于依赖项，供应商在检索它们时向依赖项添加标签。规则集上的标签由属于它的所有规则自动继承。

标签格式

标签在 **labels** 字段中指定，作为 **key=val** 格式的字符串列表，如下所示：

```
labels:
- "key1=val1"
- "key2=val2"
```

标签的密钥可以是 subdomain-prefix：

```
labels:
- "konveyor.io/key1=val1"
```

标签值可以为空：

```
labels:
- "konveyor.io/key="
```

标签值可以省略。在这种情况下，它被视为空值：

```
labels:
- "konveyor.io/key"
```

保留标签

分析器定义一些具有特殊含义的标签，如下所示：

- **konveyor.io/source**: 标识规则或规则集应用到的源技术
- **konveyor.io/target**: 标识规则或规则集的目标技术

标签选择**器**

分析器 CLI 将 **--label-selector** 字段用作选项。它是支持逻辑 AND, OR 和 NOT 操作的字符串表达式。您可以使用它根据标签过滤或过滤出规则。

示例：

- 要过滤所有具有键 **konveyor.io/source** 和 value **eap6** 的标签的规则：
  --label-selector="konveyor.io/source=eap6"
- 要过滤所有具有键 **konveyor.io/source** 和 any 值的标签的规则：
  --label-selector="konveyor.io/source"
- 使用 **&** amp;& operator 对多个规则执行逻辑 AND 操作：
  --label-selector="key1=val1 && key2"
- 使用 **||** 运算符对多个规则执行逻辑 OR 操作：
  --label-selector="key1=val1 || key2"
- 要执行 NOT 操作，以过滤掉使用 **!** 运算符设置的 **key1=val1** 标签的规则：
  --label-selector="!key1=val1"
- 使用 AND 对子表达式和控制优先级进行分组：
  --label-selector="(key1=val1 || key2=val2) && !val3"

依赖项标签

分析器引擎向依赖项添加标签。这些标签提供有关依赖项的额外信息，如其编程语言以及依赖项是开源还是内部。

目前，分析器将以下标签添加到依赖项中：

```
labels:
- konveyor.io/dep-source=internal
- konveyor.io/language=java
```

依赖项标签选择**器**

分析器 CLI 接受 **--dep-label-selector** 选项，该选项允许根据标签从依赖项生成的过滤或过滤出事件。

例如，分析器会将 **konveyor.io/dep-source** 标签添加到具有值的依赖项中，指示依赖项是否是已知的开源依赖项。

要排除所有这样的开源依赖项的事件，您可以使用 **--dep-label-selector**，如下所示：

konveyor-analyzer … --dep-label-selector !konveyor.io/dep-source=open-source

分析器中的 Java 提供程序也可以向软件包列表中添加 exclude 标签。要排除所有这些软件包，您可以使用 **--dep-label-selector** 和 **!** operator，如下所示：

konveyor-analyzer … --dep-label-selector !konveyor.io/exclude

2.1.1.2. 规则类别

- **必需**
  - 您必须解决成功迁移的问题，否则生成的应用程序将无法成功构建或运行。此类问题的示例是在目标平台上不支持的专有 API。
- **optional**
  - 如果没有解决这个问题，应用程序应该可以正常工作，但结果可能不是最佳。如果您没有在迁移时进行更改，则需要在迁移完成后尽快按计划设置。此类问题的一个示例是 EJB 2.x 代码没有升级到 EJB 3。
- **潜在**
  - 您需要在迁移过程中检查问题，但没有足够的信息来确定解决问题是否必须成功。当目标平台上没有直接兼容类型时，会迁移第三方专有类型。

2.1.1.3. 规则**操作**

规则可以包含两种类型的操作： *message* 和 *tag*。每个规则都包括其中一个规则或两者。

消息操作

当规则匹配时，message 操作会创建一条消息问题。由提供程序导出的自定义数据也可以在消息中使用。

Message: "helpful message about the issue"

Example:

```
- ruleID: test-rule
  when:
    <CONDITION>
  message: Test rule matched. Please resolve this migration issue.
```

另外，消息可以包含到外部 URL 的超链接，以提供有关此问题或快速修复的相关信息。

```
links:
  - url: "konveyor.io"
    title: "Short title for the link"
```

消息也可以是模板，以包含有关通过规则上的自定义变量进行匹配的信息。

标签**操作**

标签操作指示在找到匹配项时为应用程序生成一个或多个标签。**tag** 字段中的每一字符串可以是以逗号分隔的标签列表。另外，您可以为标签分配类别。

```
tag:
  - "tag1,tag2,tag3"
  - "Category=tag4,tag5"
```

Example

```
- ruleID: test-rule
  when:
    <CONDITION>
  tag:
  - Language=Golang
  - Env=production
  - Source Code
```

标签可以是字符串或 **key=val** 对，其中键被视为 MTA 中的标签类别。任何具有标签操作的规则都被称为本文档中的"标记规则"。

请注意，不会为仅包含标签操作的规则创建问题。

2.1.1.4. 规则**条件**

每个规则都有一个 **when** 块，它指定了一个需要满足 MTA 执行特定操作的条件。

when 块包含一个条件，但该条件可以在它下嵌套多个条件。

```
when:
  <condition>
    <nested-condition>
```

MTA 支持三种类型的条件： **提供程序**、**和 或**。

2.1.1.4.1. **供应商条件**

MTA 支持多语言源代码分析。使用 **供应商** 条件启用源代码中的特定语言。此条件定义特定语言供应商的搜索查询。**供应商** 条件还指定提供程序的"功能"用于分析代码。

供应商 条件格式为 < **provider_name>.<capability&gt**; :

```
when:
  <provider_name>.<capability>
    <input_fields>
```

分析器目前支持以下供应商 **条件** ：

- **builtin**
- **java**
- **Go**

2.1.1.4.1.1. 内置 供应商

builtin 是一种内部提供程序，可以分析引擎生成的各种文件和内部元数据。

此供应商具有以下功能：

- **file**
- **FileContent**
- **xml**
- **json**
- **hasTags**

file

文件 功能使提供程序能够搜索与给定模式匹配的源代码中的文件。

```
when:
  builtin.file:
    pattern: "<regex_to_match_filenames>"
```

FileContent

filecontent 功能可让提供程序搜索与给定模式匹配的内容。

```
when:
  builtin.filecontent:
    filePattern: "<regex_to_match_filenames_to_scope_search>"
    pattern: "<regex_to_match_content_in_the_matching_files>"
```

xml

xml 功能可让供应商在提供的 XML 文件列表上查询 XPath 表达式。该功能采用 2 个输入参数 **xpath** 和 **文件路径**。

```
when:
  builtin.xml:
    xpath: "<xpath_expressions>" (1)
    filepaths: (2)
      - "/src/file1.xml"
      - "/src/file2.xml"
```

1. **XPath** 必须是有效的 XPath 表达式。
2. **filepaths** 是要应用 XPath 查询的文件列表。

json

json 功能可让供应商在提供的 JSON 文件列表中查询 XPath 表达式。目前， **json** 仅使用 XPath 作为输入，并对代码库中的所有 JSON 文件执行搜索。

```
when:
  builtin.json:
    xpath: "<xpath_expressions>" (1)
```

1. **XPath** 必须是有效的 XPath 表达式。

hasTags

hasTags 功能使提供程序能够查询应用标签。它查询内部数据结构，以检查应用是否具有给定的标签。

```
when:
  # when more than one tags are given, a logical AND is implied
  hasTags: (1)
    - "tag1"
    - "tag2"
```

1. 指定了多个标签时，代表逻辑 AND。

2.1.1.4.1.2. Java 供应商

java 提供程序分析 Java 源代码。

此供应商具有以下功能：

- **referenced**
- **依赖项.**

referenced

引用 的功能可让供应商在源代码中查找引用。该功能采用两个输入参数，即 **模式 和位置**。

```
when:
  java.referenced:
    pattern: "<pattern>" (1)
    location: "<location>" (2)
```

1. 要匹配的 RegEx 模式，例如 **org.kubernetes memcached**
2. 指定需要匹配模式的确切位置，如 **IMPORT**

支持的位置如下：

- **CONSTRUCTOR_CALL**
- **TYPE**
- **继承**

- **METHOD_CALL**
- **注解**
- **IMPLEMENTS_TYPE**
- **ENUM_CONSTANT**
- **RETURN_TYPE**
- **IMPORT**
- 变量_DECLARATION

依赖项

依赖项 功能使供应商能够查找给定应用程序的依赖项。MTA 生成应用程序依赖项列表，您可以使用此功能查询列表，并检查应用程序是否在指定范围内的依赖项存在。

```
when:
  java.dependency:
    name: "<dependency_name>" 1
    upperbound: "<version_string>" 2
    lowerbound: "<version_string>" 3
```

1 要搜索的依赖项名称

2 依赖于依赖项版本的上限

3 降低依赖项版本绑定

2.1.1.4.1.3. Go 供应商

Go 供应商分析 Go 源代码。此提供程序的功能 **被引用 和依赖项**。

referenced

引用 的功能可让供应商在源代码中查找引用。

```
when:
  go.referenced: "<regex_to_find_reference>"
```

依赖项

依赖项 功能可让供应商查找应用程序的依赖项。

```
when:
  go.dependency:
    name: "<dependency_name>" 1
    upperbound: "<version_string>" 2
    lowerbound: "<version_string>" 3
```

1 要搜索的依赖项名称

2 依赖于依赖项版本的上限

3 降低依赖项版本绑定

2.1.1.4.2. 自定义变量

供应商条件可以关联自定义变量。您可以使用自定义变量从源代码中匹配行捕获相关信息。这些变量的值与源代码中匹配的数据进行干预。这些值可用于在规则的操作中生成详细的模板消息（请参阅 消息操作）。它们可以添加到 **customVariables** 字段中的规则中：

```
- ruleID: lang-ref-004
  customVariables:
  - pattern: '([A-z]+)\.get\(\)' 1
    name: VariableName 2
   message: "Found generic call - {{ VariableName }}" 3
  when:
    java.referenced:
      location: METHOD_CALL
      pattern: com.example.apps.GenericClass.get
```

1 **Pattern** ：找到匹配项时，源代码行中匹配的 RegEx 模式

2 **名称** ：可以在模板中使用的变量名称

3 Message : 使用自定义变量消息的模板

2.1.1.5. 逻辑条件

分析器提供了两个基本逻辑条件，和 或，它们可让您聚合其他条件的结果并创建更复杂的查询。

2.1.1.5.1. 和 条件

和 条件对一组条件的结果执行逻辑 AND 操作。当其 *所有子* 条件都匹配时，和 条件都匹配。

```
when:
  and:
    - <condition1>
    - <condition2>
```

Example

```
when:
  and:
    - java.dependency:
        name: junit.junit
```

```
      upperbound: 4.12.2
      lowerbound: 4.4.0
  - java.referenced:
      location: IMPORT
      pattern: junit.junit
```

条件也可以嵌套在其他条件内。

Example

```
when:
  and:
  - and:
    - go.referenced: "*CustomResourceDefinition*"
    - java.referenced:
        pattern: "*CustomResourceDefinition*"
  - go.referenced: "*CustomResourceDefinition*"
```

2.1.1.5.2. 或 条件

或 条件对一组条件的结果执行逻辑 OR 操作。当其 *任何子* 条件匹配时，或 条件匹配。

```
when:
  or:
    - <condition1>
    - <condition2>
```

Example

```
when:
  or:
  - java.dependency:
      name: junit.junit
      upperbound: 4.12.2
      lowerbound: 4.4.0
  - java.referenced:
      location: IMPORT
      pattern: junit.junit
```

2.1.2. Rulesets（规则集）

组规则形成规则集。MTA 不需要每个规则文件都属于规则集，但您可以使用规则集对实现通用目标并传递规则到规则集的多个规则进行分组。

您可以通过将一个或多个 YAML 规则放在目录中并创建 ruleset.yaml 文件来创建规则集。当您使用 --rules 选项将此目录作为输入传递给 MTA CLI 时，此目录中的所有规则都将被视为 ruleset.yaml 文件定义的规则集的一部分。

ruleset.yaml 文件存储规则集的元数据。

```
name: "Name of the ruleset" 1
description: "Description of the ruleset"
labels: 2
 - key=val
```

1

名称在提供的规则集中必须是唯一的。

2

ruleset 标签由属于规则集的所有规则继承。

2.2. 创建基本 YAML 规则

这部分论述了如何创建基本 MTA YAML 规则。假设您已安装了 MTA。有关安装说明，请参阅 *MTA CLI 指南*。

2.2.1. 创建基本 YAML 规则模板

基于 MTA YAML 的规则有以下基本结构：

```
when(condition)
 message(message)
 tag(tags)
```

流程

1. 在 /home/<USER& gt;/ 目录中，创建一个文件，其中包含 YAML 规则的基本语法，如下所示：

```
- category: mandatory
  description: |
   <DESCRIPTION TITLE>
```

```
 <DESCRIPTION TEXT>
effort: <EFFORT>
labels:
- konveyor.io/source=<SOURCE_TECH>
- konveyor.io/target=<TARGET_TECH>
links:
- url: <HYPERLINK>
 title: <HYPERLINK_TITLE>
message: <MESSAGE>
tag:
- <TAG1>
- <TAG2>
ruleID: <RULE_ID>
when:
 <CONDITIONS>
```

2.2.2. 创建基本 YAML 规则集模板

如果要对多个类似的规则进行分组，您可以通过将其文件放在目录中，并在目录的根目录中创建 ruleset.yaml 文件，为它们创建一个规则集。当您使用 --rules 选项将此目录作为输入传递给 MTA CLI 时，MTA 会将该目录中的所有文件视为属于 ruleset.yaml 文件中定义的规则集。

流程

1. 如果要使用 --rules 选项传递整个目录，请为 ruleset.yaml 文件创建一个模板：

```
name: <RULESET_NAME> (1)
description: <RULESET_DESCRIPTION>
labels: (2)
 - key=val
```

(1) 名称在提供的规则集中必须是唯一的。

(2) ruleset 标签由属于规则集的所有规则继承。

2.2.3. 创建 YAML 规则

每个规则文件包含一个或多个 YAML 规则。每个规则都包含元数据、条件和操作。

流程

1. 创建 when 条件。

   YAML 规则的 when 条件可以是 provider、和 或 或。

   a. 创建 供应商 条件

      提供程序条件用于定义特定语言提供程序的搜索查询，并调用提供程序的特定功能。

      条件的一般格式为 < provider_name>.<capability>。该条件也具有内部字段来指定搜索的详细信息。创建 供应商 条件及其内部字段的方式取决于您使用的供应商以及您调用的功能。

      下表列出了可用的供应商及其功能。选择适合您要创建的规则用途的供应商及其功能。这个条件部分不包含任何条件的字段。

供应商	功能	描述
java	**referenced**	查找带有可选代码位置的模式引用以了解详细搜索
	依赖项	检查应用程序是否有给定依赖项
builtin	**xml**	使用 XPath 查询搜索 XML 文件
	json	使用 JSONPath 查询搜索 JSON 文件
	FileContent	使用 RegEx 模式搜索常规文件中的内容
	file	查找名称与指定模式匹配的文件
	hasTags	通过标记规则检查是否为应用程序创建标签
Go	**referenced**	查找模式的引用
	依赖项	检查应用程序是否有给定依赖项

      以下示例显示了使用 引用 功能的 java 供应商条件。

      Example

```
when:
  java.referenced:
```

2. **在 供应商 条件中添加合适的字段。**

下表列出了所有可用的供应商、它们的功能及其字段。选择属于您选择的供应商和功能的字段。请注意，一些字段是必需的。

供应商	功能	字段	必需？	描述
java	**referenced**	**pattern**	是	正则表达式模式
		位置	否	源代码位置；请查看以下以了解所有支持的搜索位置列表
	依赖项	**name**	是	依赖项的名称
		nameregex	否	与名称匹配的正则表达式模式
		upperBound	否	匹配大于或等于的版本号
		lowerBound	否	匹配大于或等于的版本号
builtin	**xml**	**XPath**	是	XPath 查询
		命名空间	否	将查询范围到命名空间的映射
		filepaths	否	要缩减搜索的文件可选列表
	json	**XPath**	是	XPath 查询
		filepaths	否	要缩减搜索的文件可选列表
	FileContent	**pattern**	是	在内容中匹配的正则表达式模式
		filePattern	否	仅搜索名称与此模式匹配的文件
	file	**pattern**	是	查找具有与此模式匹配的文件
	hasTags	这是字符串标签的内联列表。有关标签格式的详情，请参阅 *Tag Action*。		
Go	**referenced**	**pattern**	是	正则表达式模式

供应商	功能	字段	必需？	描述
	依赖项	**name**	是	依赖项的名称
		nameregex	否	与名称匹配的正则表达式模式
		upperBound	否	匹配大于或等于的版本号
		lowerBound	否	匹配大于或等于的版本号

以下搜索位置可用于范围 java 搜索：

- **CONSTRUCTOR_CALL**
- **TYPE**
- **继承**
- **METHOD_CALL**
- **注解**
- **IMPLEMENTS_TYPE**
- **ENUM_CONSTANT**
- **RETURN_TYPE**
- **IMPORT**
- **VARIABLE_DECLARATION**

以下示例显示了搜索软件包引用的规则 的时间 条件。

Example

```
when:
  java.referenced:
    location: PACKAGE
    pattern: org.jboss.*
```

3. 创建 AND 或 OR 条件

- 当其 *所有子* 条件都匹配时，和 条件都匹配。创建 和 条件，如下所示：

```
when:
  and:
  - java.dependency:
      name: junit.junit
      upperbound: 4.12.2
      lowerbound: 4.4.0
  - java.referenced:
      location: IMPORT
      pattern: junit.junit
```

- 当其任何子条件都匹配时，或 条件匹配。*创建 或 条件，如下所示：*

```
when:
  or:
  - java.dependency:
      name: junit.junit
      upperbound: 4.12.2
      lowerbound: 4.4.0
  - java.referenced:
      location: IMPORT
      pattern: junit.junit
```

2.2.4. 使用自定义 YAML 规则运行分析

要运行分析，请在 CLI 中使用 --rules 选项。

流程

- 要在一条规则文件 /home/<USER>/rule.yaml 中使用规则，请运行以下命令：

```
mta-cli analyze --input /home/<USER>/data/ --output /home/<USER>/output/ --rules
/home/<USER>/rule.yaml
```

其中：

  - /home/<USER>/data/ - 源代码或二进制的目录
  - /home/<USER>/output/ - 报告的目录(HTML 和 YAML)

- 要使用多个规则文件，您需要将它们放在目录中，并添加 ruleset.yaml 文件。然后，该目录被视为 规则集，您可以将其作为输入传递给 --rules 选项。

请注意，如果您要在 CLI 中使用 --target 或 --source 选项，引擎将只选择与该目标标签匹配的规则。因此，请确保已在规则上添加了目标或源标签。如需了解更多详细信息，请参阅 保留标签。

2.3. 创建第一个 YAML 规则

本节介绍了创建和测试第一个基于 MTA YAML 的规则的过程。这假设您已安装了 MTA。有关安装说明， 请参阅 CLI 指南中的 安装和运行 CLI。

在本例中，您将创建一个规则来发现应用程序定义了包含 < class-loading> 元素的 jboss- web.xml 文件，并提供描述如何迁移代码的文档的链接。

2.3.1. 为规则创建 YAML 文件

为您的第一个规则创建 YAML 文件。

```
$ mkdir /home/<USER>/rule.yaml
```

2.3.2. 创建数据来测试规则

1. 在目录中创建 jboss-web.xml 和 pom.xml 文件：

```
mkdir /home/<USER>/data/
touch /home/<USER>/data/jboss-web.xml
touch /home/<USER>/data/pom.xml
```

2. *在您创建的 **jboss-web.xml** 文件中，粘贴以下内容：*

```
<!DOCTYPE jboss-web PUBLIC "-//JBoss//DTD Web Application 4.2//EN"
"http://www.jboss.org/j2ee/dtd/jboss-web_4_2.dtd">
<jboss-web>
   <class-loading java2ClassLoadingCompliance="false">
      <loader-repository>
         seam.jboss.org:loader=@projectName@
         <loader-repository-config>java2ParentDelegation=false</loader-repository-config>
      </loader-repository>
   </class-loading>
</jboss-web>
```

3. *在您创建的 **pom.xml** 文件中，粘贴以下内容：*

```
<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>test</groupId>
 <artifactId>test</artifactId>
 <version>1.1.0-SNAPSHOT</version>

 <properties>
<maven.compiler.source>1.7</maven.compiler.source>
<maven.compiler.target>1.7</maven.compiler.target>
 </properties>

 <dependencies>
 </dependencies>
</project>
```

2.3.3. 创建规则

*基于 **MTA YAML** 的规则使用以下规则模式：*

```
when(condition)
  perform(action)
```

流程

1. 在您创建的 rule.yaml 文件中，粘贴以下内容：

```
- ruleID: <UNIQUE_RULE_ID> (1)
  description: <DESCRIPTION> (2)
  when:
    <CONDITION(S)> (3)
  message: <MESSAGE> (4)
  labels: <LABELS> (5)
  effort: <EFFORT> (6)
  links:
  - <LINKS> (7)
```

(1) 规则的唯一 ID，例如 jboss5-web-class-loading。

(2) 规则的文本描述。

(3) 完成 when 块指定一个或多个条件：

a. 使用 内置 提供程序的 XML 功能，因为此规则在 XML 文件中检查匹配项。

b. 要在作为 jboss -web 的子类加载 元素上匹配，请使用 XPath 表达式 jboss-web/web-loading 作为 XML 查询。在这种情况下，您需要一个条件：

```
when:
  builtin.xml:
    xpath: jboss-web/class-loading
```

(4) 解释迁移问题的有用消息。当规则匹配时，报告中会生成消息。例如：

```
message: The class-loading element is no longer valid in the jboss-web.xml file.
```

(5) 规则的字符串标签列表。

6

修复此问题的预期故事点数。

7

一个或多个超链接指向有关迁移问题的文档。

```
links:
- url: https://access.redhat.com/documentation/zh-CN/JBoss_Enterprise_Application_Platform/6.4/html-single/Migration_Guide/index.html#Create_or_Modify_Files_That_Control_Class_Loading_in_JBoss_Enterprise_Application_Platform_6
  title: Create or Modify Files That Control Class Loading in JBoss EAP 6
```

这个规则现已完成，类似如下：

```
- ruleID: jboss5-web-class-loading
  description: Find class loading element in JBoss XML file.
  when:
    builtin.xml:
      xpath: jboss-web/class-loading
  message: The class-loading element is no longer valid in the jboss-web.xml file.
  effort: 3
  links:
  - url: https://access.redhat.com/documentation/zh-CN/JBoss_Enterprise_Application_Platform/6.4/html-single/Migration_Guide/index.html#Create_or_Modify_Files_That_Control_Class_Loading_in_JBoss_Enterprise_Application_Platform_6
    title: Create or Modify Files That Control Class Loading in JBoss EAP 6
```

2.3.4. 安装规则

流程

1. 将 CLI 指向您创建的规则文件：

```
–rules /home/<USER>/rules.yaml
```

2.3.5. 测试规则

流程

要测试规则，请在 MTA CLI 中使用 rules 选项将输入指向您创建的测试数据并传递规则：

-

```
mta-cli analyze --input /home/<USER>/data/ --output /home/<USER>/output/ --rules
/home/<USER>/rules.yaml
```

2.3.6. 查看报告

查看报告以确保它提供了预期的结果。

流程

1. 分析完成后，命令会输出 HTML 报告的路径：

```
INFO[0066] Static report created. Access it at this URL:
URL="file:/home/<USER>/output/static-report/index.html"
```

在 web 浏览器中打开 /home/<USER_NAME>/output/static-report/index.html。

2. 导航到左侧菜单中的 问题 选项卡。

3. 验证规则是否已执行：

   a. 在 issues 表中，在搜索栏中输入 JBoss XML。

   b. 验证表中是否存在标题 Find class loading element in JBoss XML 文件中的 问题。

4. 点 jboss-web.xml 链接打开受影响的文件。

第 3 章 测试 XML 规则

创建 XML 规则后，您应该创建一个测试规则以确保它可以正常工作。

3.1. 创建测试规则

使用与创建测试规则的过程类似的进程创建一个 XML 测试，但有以下区别：

- 测试规则应当放在要测试的规则下的 test/ 目录中。
- 测试类等任何数据都应放在 tests/ 目录下的 data/ 目录中。
- 测试规则应使用 .windup.test.xml 扩展。
- 这些规则使用 Test XML 规则结构中定义的结构。

另外，建议您创建一个遵循它测试的规则名称的测试规则。例如，如果创建了一个规则，其文件名为 proprietary-rule.mta.xml，则测试规则应称为 proprietary-rule.windup.test.xml。

3.1.1. 测试 XML 规则结构

所有测试 XML 规则都定义为包含一个或多个 rulesets 的 rulesets 规则集中的一个元素。如需了解更多详细信息，请参阅 MTA XML 规则模式。

规则测试是针对特定迁移区域的一个或多个测试组。这是 <ruletest> 元素的基本结构。

- <ruletest id="<RULE_TOPIC>-test">: 定义它作为唯一 MTA ruletest，并将其指定为唯一的 ruletest id。
  - <testDataPath>：定义用于测试的所有数据的路径，如类或文件。
  - <sourceMode>：指定传递的数据是否只包括源文件。如果存档（如 EAR、WAR 或

JAR）正在使用，则这应设置为 false。默认值为 true。

- <rulePath> : 要测试的规则的路径。这应该以要测试的规则的名称结尾。
- <ruleset>: Rulesets 包括测试的逻辑。它们与 Rulesets 中定义的相同。

3.1.2. XML 规则语法

除了标准 XML 规则语法中的标签外， when 条件通常用于创建测试规则时：

- <not>
- <iterable-filter>
- <classification-exists>
- <hint-exists>

除了标准 perform action 语法中的标签外，以下 when 条件通常用于测试规则中的操作：

- <fail>

3.1.2.1. <not> 语法

概述

<not> 元素是标准逻辑 not 操作符，如果不满足条件，通常用于执行 <fail>。

以下是测试规则的一个示例，如果分析末尾只有特定消息，则会失败。

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
        id="proprietary-servlet-test" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
```

```
 <testDataPath>data/</testDataPath>
 <rulePath>../proprietary-servlet.windup.xml</rulePath>
 <ruleset>
  <rules>
   <rule id="proprietary-servlet-01000-test">
    <when>
     <!--
   The `<not>` will perform a logical _not_ operator on the elements within.
  -->
      <not>
       <!--
     The defined `<iterable-filter>` has a size of `1`. This rule will only match on a single
instance of the defined hint.
    -->
        <iterable-filter size="1">
         <hint-exists message="Replace the proprietary @ProprietaryServlet annotation with the Java
EE 7 standard @WebServlet annotation*" />
        </iterable-filter>
      </not>
     </when>
     <!--
  This `<perform>` element is only executed if the previous `<when>` condition is false.
      This ensures that it only executes if there is not a single instance of the defined hint.
     -->
     <perform>
      <fail message="Hint for @ProprietaryServlet was not found!" />
     </perform>
    </rule>
   </rules>
  </ruleset>
</ruletest>
```

<not> 元素没有唯一属性或子元素。

3.1.2.2. <iterable-filter> 语法

概述

<iterable-filter> 元素统计验证条件的次数。详情请参阅 IterableFilter 类。

下例中查找指定消息的四个实例：

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
        id="proprietary-servlet-test" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
 <testDataPath>data/</testDataPath>
 <rulePath>../proprietary-servlet.mta.xml</rulePath>
 <ruleset>
  <rules>
```

```
    <rule id="proprietary-servlet-03000-test">
      <when>
        <!--
      The `<not>` will perform a logical _not_ operator on the elements within.
    -->
          <not>
      <!--
        The defined `<iterable-filter>` has a size of `4`. This rule will only match on four instances
of the defined hint.
      -->
            <iterable-filter size="4">
              <hint-exists message="Replace the proprietary @ProprietaryInitParam annotation with the
Java EE 7 standard @WebInitParam annotation*" />
            </iterable-filter>
          </not>
        </when>
  <!--
    This `<perform>` element is only executed if the previous `<when>` condition is false.
    In this configuration, it only executes if there are not four instances of the defined hint.
  -->
        <perform>
          <fail message="Hint for @ProprietaryInitParam was not found!" />
        </perform>
      </rule>
    </rules>
  </ruleset>
</ruletest>
```

<iterable-filter> 元素没有唯一的子元素。

<iterable-filter> 元素属性

属性名称	类型	描述
size	整数	验证的次数。

3.1.2.3. <classification-exists> 语法

<classification-exists> 元素决定分析中是否包含特定的分类标题。如需更多信息，请参阅 ClassificationExists 类。

重要

当测试包含特殊字符（如 [或 ' ）的消息时，您必须用反斜杠(\)进行转义来正确匹配。

以下是搜索特定分类标题的示例。

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
        id="proprietary-servlet-test" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
 <testDataPath>data/</testDataPath>
 <rulePath>../weblogic.mta.xml</rulePath>
 <ruleset>
  <rules>
   <rule id="weblogic-01000-test">
    <when>
     <!--
   The `<not>` will perform a logical _not_ operator on the elements within.
  -->
      <not>
   <!--
     The defined `<classification-exists>` is attempting to match on the defined title.
     This classification would have been generated by a matching `<classification
title="WebLogic scheduled job" .../>` rule.
    -->
        <classification-exists classification="WebLogic scheduled job" />
      </not>
    </when>
 <!--
  This `<perform>` element is only executed if the previous `<when>` condition is false.
  In this configuration, it only executes if there is not a matching classification.
 -->
     <perform>
      <fail message="Triggerable not found" />
     </perform>
   </rule>
  </rules>
 </ruleset>
</ruletest>
```

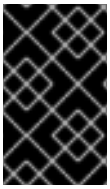
<classification-exists> 没有唯一的子元素。

<has-classification> 元素属性

属性名称	类型	描述
classification	字符串	要搜索的 **<classification> title**。
in	字符串	可选参数，限制对包含所定义文件名的文件的匹配。

3.1.2.4. <hint-exists> 语法

<hint-exists> 元素决定分析中是否包含特定的提示。它搜索已定义消息的任何实例，通常用于搜索 <message> 元素的开头或特定类。详情请参阅 HintExists 类。



重要

当测试包含特殊字符（如 [或 ' ）的消息时，您必须用反斜杠(\)进行转义来正确匹配。

以下是搜索特定提示的示例。

```
<ruletest xmlns="http://windup.jboss.org/schema/jboss-ruleset"
        id="proprietary-servlet-test" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
 <testDataPath>data/</testDataPath>
 <rulePath>../weblogic.windup.xml</rulePath>
 <ruleset>
  <rules>
   <rule id="weblogic-eap7-05000-test">
    <when>
     <!--
   The `<not>` will perform a logical _not_ operator on the elements within.
  -->
      <not>
   <!--
     The defined `<hint-exists>` is attempting to match on the defined message.
     This message would have been generated by a matching `<message>` element on the
`<hint>` condition.
    -->
        <hint-exists message="Replace with the Java EE standard method
.*javax\.transaction\.TransactionManager\.resume\(Transaction tx\).*" />
      </not>
     </when>
 <!--
  This `<perform>` element is only executed if the previous `<when>` condition is false.
  In this configuration, it only executes if there is not a matching hint.
 -->
     <perform>
      <fail message="Note to replace with standard TransactionManager.resume is missing!" />
     </perform>
    </rule>
  </rules>
 </ruleset>
</ruletest>
```

<hint-exists> 元素没有唯一的子元素。

<hint-exists> 元素属性

属性名称	类型	描述
message	字符串	要搜索的 **<hint> message**。

属性名称	类型	描述
in	字符串	可选参数限制与引用所给文件名的 **InLineHintModels** 匹配。

3.1.2.5. <fail> 语法

<fail> 元素将执行报告为失败，并显示相关的消息。通常与 <not> 条件一起使用，仅在不满足条件时才显示消息。

<fail> 元素没有唯一的子元素。

<fail> 元素属性

属性名称	类型	描述
message	字符串	要显示的消息。

3.2. 手动测试 XML 规则

您可以针对应用程序文件运行 XML 规则来测试它：

```
$ <MTA_HOME>/mta-cli [--sourceMode] --input <INPUT_ARCHIVE_OR_FOLDER> --output
<OUTPUT_REPORT_DIRECTORY> --target <TARGET_TECHNOLOGY> --packages
<PACKAGE_1> <PACKAGE_2> <PACKAGE_N>
```

您应看到以下结果：

```
Report created: <OUTPUT_REPORT_DIRECTORY>/index.html
        Access it at this URL: file:///<OUTPUT_REPORT_DIRECTORY>/index.html
```

有关如何运行 MTA 的更多信息，请参阅 Migration Toolkit for Applications CLI 指南。

3.3. 使用 JUNIT 测试规则

创建了测试规则后，它可以作为 JUnit 测试的一部分进行分析，以确认该规则是否符合执行的所有条件。MTA 规则存储库中的 WindupRulesMultipleTests 类旨在同时测试多个规则，并根据任何缺失的要求提供反馈。

先决条件

- *分叉并克隆 MTA XML 规则。此仓库的位置被称为 <RULESETS_REPO>。*
- *创建一个测试文件。*

创建 JUnit 测试配置

以下说明详细介绍了使用 Eclipse 创建 JUnit 测试。当使用其他 IDE 时，建议您参考 IDE 文档了解创建 JUnit 测试的说明。

1. *将 MTA 规则集仓库导入到您的 IDE 中。*
2. *将自定义规则以及相应的测试和数据复制到 </path/to/RULESETS_REPO>/rules-reviewed/<RULE_NAME>/ 中。这应当创建以下目录结构：*

目录结构

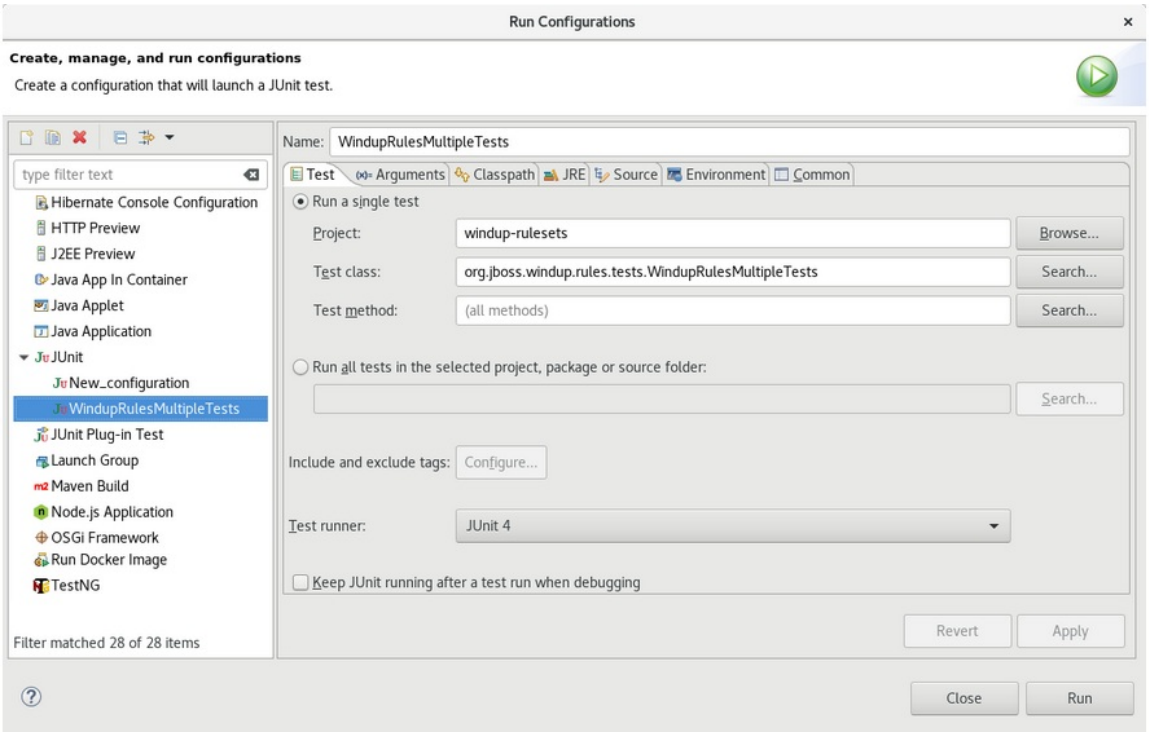
```
├── *rules-reviewed/* _(Root directory of the rules found within the project)_
│   ├── *<RULE_NAME>/* _(Directory to contain the newly developed rule and tests)_
│   │   ├── *<RULE_NAME>.windup.xml* _(Custom rule)_
│   │   ├── *tests/* _(Directory that contains any test rules and data)_
│   │   │   ├── *<RULE_NAME>.windup.test.xml* _(Test rule)_
│   │   │   └── *data/* _(Optional directory to contain test rule data)_
```

3. *从顶部菜单栏中选择 Run。*
4. *从出现的下拉列表中选择 Run Configuration...。*
5. *在左侧的选项中，右键单击 JUnit，再选择 New。*
6. *使用以下命令：*

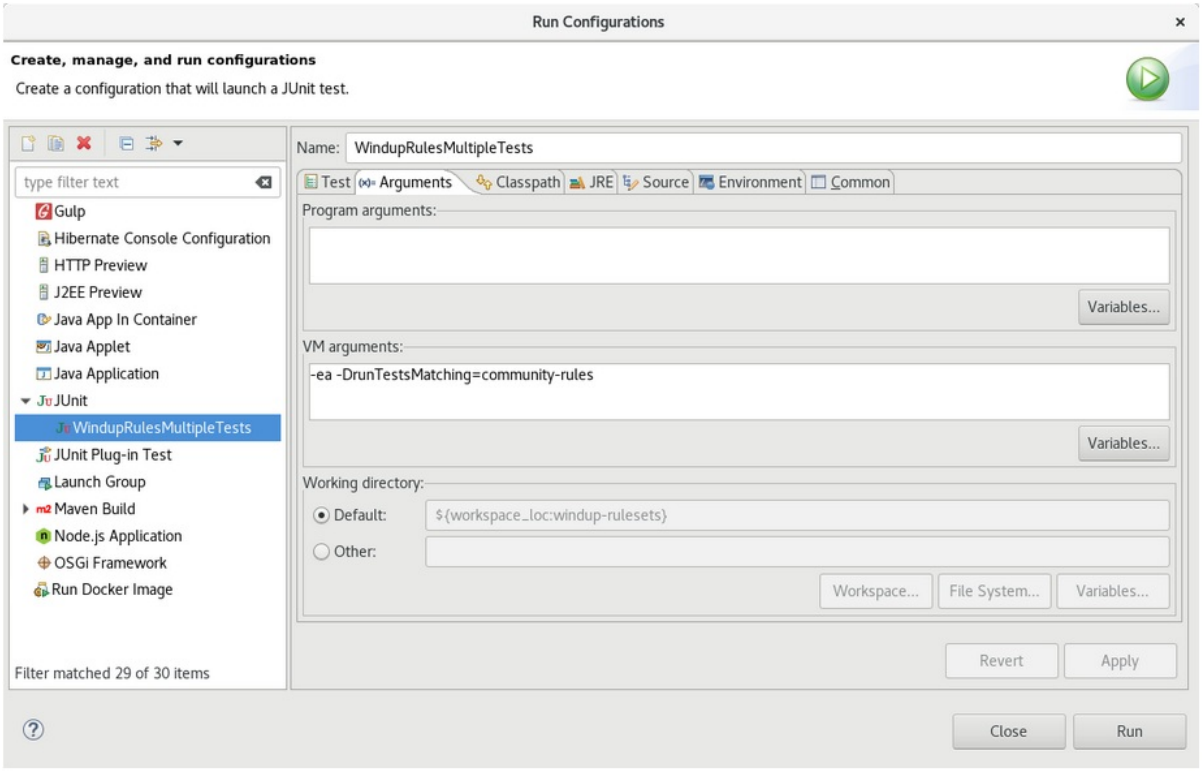
- ***Name ： JUnit 测试的名称，如 WindupRulesMultipleTests。***

- ***Project: 确保将其设置为 windup-rulesets。***

- ***Test class: 将其设置为 org.jboss.windup.rules.tests.WindupRulesMultipleTests。***



7. ***选择 Arguments 选项卡，并添加 -DrunTestsMatching=<RULE_NAME> VM 参数。例如，如果您的规则名称是 community-rules，则您可以添加 -DrunTestsMatching=community-rules，如以下镜像中所示。***



8. *点右下角的 Run 以开始测试。*

执行完成后，结果就可以进行分析。如果所有测试都通过，则正确格式化测试规则。如果所有测试都未通过，建议解决测试失败中出现的每个问题。

3.4. 关于验证报告

验证报告提供有关测试规则和失败的详细信息，并包含以下部分：

- *概述*

本节包含测试运行总数并报告错误和失败的数量。它显示总成功率以及生成报告的时间（以秒为单位）。

- *软件包列表*

本节包含为每个软件包执行的测试数量，并报告错误和失败的数量。它显示成功率以及要分析的每个软件包的时间（以秒为单位）。

此时会显示一个名为 org.jboss.windup.rules.tests 的软件包，除非定义了额外的测试案例。

- 测试问题单

本节描述了测试问题单。每个失败都包含一个 Details 部分，可用于显示断言堆栈 trace，包括可表示错误源的人类可读行。

3.4.1. 创建验证报告

您可以为自定义规则创建验证报告。

先决条件

- 您必须分叉并克隆 MTA XML 规则。
- 您必须具有一个或多个测试 XML 规则才能进行验证。

流程

1. 进入到本地 windup-rulesets 存储库。
2. 为您的自定义规则和测试创建一个目录：windup-rulesets/rules-reviewed/myTests。
3. 将自定义规则和测试复制到 windup-rulesets/rules-reviewed/<myTests> 目录中。
4. 从 windup-rulesets 存储库的根目录中运行以下命令：

```
$ mvn -Dtest=WindupRulesMultipleTests -DrunTestsMatching=<myTests> clean
<myReport>:report 1 2
```

1. 指定包含自定义规则和测试的目录。如果省略 -DrunTestsMatching 参数，验证报告将包含所有测试，并且生成需要更长的时间。
2.

指定您的报告名称。

验证报告在 windup-rulesets/target/site/ 仓库中创建。

3.4.2. 验证报告错误消息

验证报告包含运行规则和测试时遇到的错误。

下表包含错误消息以及如何解决错误。

表 3.1. 验证报告错误消息

错误消息	描述	解决方案
没有与规则匹配的测试文件	当规则文件没有对应的测试文件时，会发生此错误。	为现有规则创建测试文件。
测试规则 Ids <RULE_NAME> 没有找到!	当规则存在但没有对应的 ruletest 时，会抛出此错误。	为现有规则创建测试。
XML 解析在文件 <FILE_NAME> 上失败	XML 文件中的语法无效，无法通过规则验证器成功解析。	更正无效的语法。
未找到来自 **<testDataPath>** 标签的测试文件路径。预期的测试文件的路径为：<RULE_DATA_PATH>	测试规则中 **<testDataPath>** 标签中定义的路径中没有找到任何文件。	创建 **<testDataPath>** 标签中定义的路径，确保所有所需的数据文件都位于此目录中。
未执行带有 id=" <RULE_ID>" 的规则。	在此验证过程中没有执行具有提供的 id 的规则。	确保存在与指定规则中定义的条件匹配的测试数据文件。

第 4 章 覆盖规则

您可以通过 MTA 或自定义规则覆盖分发的核心规则。例如，您可以更改规则的匹配条件、工作或提示文本。这可以通过复制原始规则，将其标记为规则覆盖，并进行必要的调整。

您可以通过创建一个带有空 <rule> 元素的 <rule> 覆盖来禁用规则。

4.1. 覆盖一个规则

您可以覆盖内核或自定义规则。

流程

1. 复制包含您要覆盖的规则的 XML 文件到自定义规则目录。

   自定义规则可以放置在 <MTA_HOME>/rules, ${user.home}/.mta/rules/ 中，或由 --userRulesDirectory 命令行参数指定的目录。

2. 编辑 XML 文件，使其只包含您要覆盖的规则的 <rule> 元素。

   注意

   新规则集中未被新规则集覆盖的规则会正常运行。

3. 确保保留相同的规则和规则集 ID。当您复制原始规则 XML 时，这将确保 ID 相匹配。

4. 确保覆盖规则集中的目标技术与您为运行分析指定的目标之一匹配。

5. 将 <overrideRules>true</overrideRules> 元素添加到 ruleset 元数据。

6. 更新规则定义。

您可以更改规则定义中的任何内容。新规则覆盖其整个原始规则。

以下规则覆盖示例将 weblogic 规则集中的 weblogic-02000 规则 的工作量 从 1 改为 3 ：

规则覆盖定义示例

```
<?xml version="1.0"?>
<ruleset id="weblogic"
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd"> 1
    <metadata>
        ...
        <overrideRules>true</overrideRules> 2
    </metadata>
    <rules>
        <rule id="weblogic-02000" xmlns="http://windup.jboss.org/schema/jboss-ruleset"> 3
            <when>
                <javaclass references="weblogic.utils.StringUtils.{*}"/>
            </when>
            <perform>
                <hint effort="3" category-id="mandatory" title="WebLogic StringUtils Usage"> 4
                    <message>Replace with the StringUtils class from Apache Commons.</message>
                    <link href="https://commons.apache.org/proper/commons-lang/" title="Apache Commons
Lang"/>
                    <tag>weblogic</tag>
                </hint>
            </perform>
        </rule>
    </rules>
</ruleset>
```

1

确保 ruleset id 与原始 ruleset id 匹配。

2

将 <overrideRules>true</overrideRules> 添加到 <metadata> 部分。

3

确保 rule id 匹配原始的 rule id。

4

更新的 effort。

当您运行 MTA 时，该规则会用相同的规则 ID 覆盖原始规则。您可以通过查看 Rule Provider Executions Overview 的内容来验证已使用的新规则。

4.2. 禁用已规则

要禁用规则，请按照以下示例创建带有空 <rule> 元素的规则覆盖定义：

规则覆盖定义示例来禁用一个规则

```
<?xml version="1.0"?>
<ruleset id="weblogic"
    xmlns="http://windup.jboss.org/schema/jboss-ruleset"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://windup.jboss.org/schema/jboss-ruleset
http://windup.jboss.org/schema/jboss-ruleset/windup-jboss-ruleset.xsd">
    <metadata>
        ...
        <overrideRules>true</overrideRules>
    </metadata>
    <rules>
        <rule id="weblogic-02000" xmlns="http://windup.jboss.org/schema/jboss-ruleset">
        1
        </rule>
    </rules>
</ruleset>
```

1

<rule> 原始为空，因此 weblogic ruleset 中的 weblogic-02000 规则被禁用。

第 5 章 使用自定义规则类别

您可以创建自定义规则类别，并为它们分配 MTA 规则。

注意

尽管 MTA 使用旧的 severity 字段处理规则，但您必须更新自定义规则以使用新的 category-id 字段。

5.1. 添加一个自定义类别

您可以将自定义类别添加到规则类别文件中。

流程

1. 编辑位于 <MTA_HOME>/rules/migration-core/core.windup.categories.xml 的类别文件。
2. 添加新的 <category> 元素并填写以下参数：
   - id ：用于引用类别的 MTA 规则的 ID。
   - priority ：相对于其他类别的排序优先级。首先会显示具有最低值的类别。
   - name ：类别的显示名称。
   - description: 类别的描述。

自定义规则类别示例

```
<?xml version="1.0"?>
<categories>
  ...
  <category id="custom-category" priority="20000">
    <name>Custom Category</name>
```

```
        <description>This is a custom category.</description>
    </category>
</categories>
```

此类别可供 MTA 规则引用。

5.2. 为自定义类别分配规则

您可以为新的自定义类别分配一个规则。

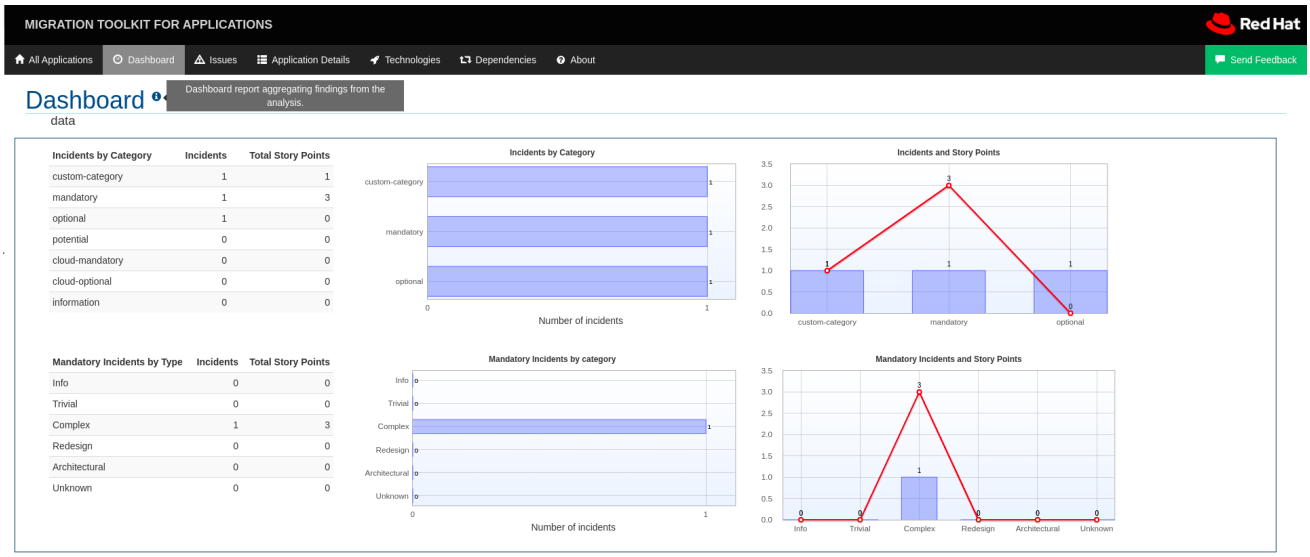
流程

在 MTA 规则中，按如下所示更新 category-id 字段。

```
<rule id="rule-id">
    <when>
        ...
    </when>
    <perform>
        <hint title="Rule Title" effort="1" category-id="custom-category">
            <message>Hint message.</message>
        </hint>
    </perform>
</rule>
```

如果满足此规则条件，则此规则识别的事件会用到您的自定义类别。自定义类别显示在仪表板上，并在问题报告中显示。

*图 **5.1.** 仪表板上的自定义类别*



附录 A. 参考材料

A.1. 关于规则故事点

A.1.1. 什么是故事点？

故事点是敏捷软件开发中常用的抽象指标，用于估算实施功能或更改所需的工作量水平。

应用的 Migration Toolkit for Applications 使用故事点来代表迁移特定应用程序构造所需的工作程度，以及整个应用程序。它不一定转换为“人-小时”，但该值在不同的任务间应保持一致。

A.1.2. 故事点如何在规则中估计

估算一个规则的故事点的工作量水平可能很棘手。以下是估算规则所需工作量时，使用的一般准则 MTA。

努力级别	故事点	描述
信息	0	迁移过程中具有非常低或没有优先级的信息。
微小	1	迁移是一个微小的变化或一个简单的库交换，没有或有最小的 API 更改。
复杂	3	迁移任务所需的更改比较复杂，但有一个已包括在文档中的解决方案。
重新设计	5	迁移任务需要重新设计或完整的库更改，并有显著的 API 更改。
架构重组	7	迁移会需要组件或子系统的完全的架构重组。
Unknown	13	迁移解决方案并非已知的，可能需要进行彻底的重写。

A.1.3. 任务类别

除了工作程度外，您还可以对迁移任务进行分类，以指明任务的严重性。下列类别用于对问题进行分组，以帮助确定迁移的工作量。

Mandatory（必需）

必须成功完成该任务才能成功迁移。如果没有进行任何更改，则生成的应用不会成功构建或运行。例如，替换在目标平台中不支持的专有 API。

选填

如果没有完成迁移任务，应用程序应该可以正常工作，但结果可能不是最佳。如果迁移时没有进行任何更改，建议在迁移完成后尽快按计划设置。

Potential

应在迁移过程中检查该任务，但没有足够的详细信息来确定任务是否成功完成。当没有直接兼容类型时，这将迁移第三方专有类型。

信息

该任务会包括告知您存在某些文件。可能需要将它们检查或修改为现代化工作的一部分，但通常不需要进行更改。

有关分类任务的更多信息，请参阅使用自定义规则类别。

A.2. 其他资源

A.2.1. 查看现有 MTA XML 规则

基于 MTA XML 的规则位于 GitHub 上，其位置为：https://github.com/windup/windup-rulesets/tree/master/rules/rules-reviewed。

您可以在本地计算机上分叉和克隆 MTA XML 规则。

规则按目标平台和功能分组。当您创建新规则时，找到与所需规则类似的规则，并将它用作入门模板。

新规则不断添加，因此最好经常检查更新。

A.2.1.1. 分叉和克隆应用程序 XML 规则的 Migration Toolkit

Migration Toolkit for Applications windup-rulesets 存储库提供如何创建基于 Java 的规则附加组件和 XML 规则的工作示例。您可以使用它们作为创建自己的自定义规则的起点。

必须在您的机器上安装了 git 客户端。

1. 点 Migration Toolkit for Applications Rulesets GitHub 页面上的 Fork 链接，在您自己的 Git 中创建项目。fork 创建的 fork GitHub 存储库 URL 应该类似如下：https://github.com/<YOUR_USER_NAME>/windup-rulesets.git。

2. 将 Migration Toolkit for Applications rulesets 存储库克隆到本地文件系统：

```
$ git clone https://github.com/<YOUR_USER_NAME>/windup-rulesets.git
```

3. 这会在本地文件系统上创建并填充 windup-rulesets 目录。导航到新创建的目录，例如

```
$ cd windup-rulesets/
```

4. 如果要能够检索最新的代码更新，请添加远程 upstream 仓库，以便您可以获取原始分叉的存储库的任何更改。

```
$ git remote add upstream https://github.com/windup/windup-rulesets.git
```

5. 从 upstream 仓库获取最新的文件。

```
$ git fetch upstream
```

A.2.2. 其他资源

- MTA Javadoc: http://windup.github.io/windup/docs/latest/javadoc
- MTA Jira issue tracker: https://issues.redhat.com/projects/MTA/issues
- MTA 邮件列表： windup-eng@redhat.com

更新于 2024-05-24