介绍和普通用户指南

by The JBoss Drools team

第 1 章　欢迎

我总是和终端业务用户争论不休，理解规则和流程、最新规则和事件处理的区别。对此，在他们的意识中有这样的问题，并且他们希望使用某些软件模拟它。使用两个供应商提供的传统方式，强迫业务用户与一个刚刚获得方式的面向流程或面向规则的方法一起工作， 他们将使用该工具模拟这点，往往整体是极为混乱的。

PegaSystems 和 Microsoft 做了大量的工作，显示两者可以被结合，并且可以使用一个行为的模拟方法。这让业务用户更自然地应用提供给他们的各种方法工作，不必使用获得方式的工具。从面向流程到面向规则或中间的灰色地带——任何当时的相应问题被模拟。

Drools 5.0 更进了一步，不仅增加了基于工作流与 Drools 流的 BPMN2，而且增加了事件处理与 Drools Fusion（熔合），为软件开发者创建了一个更全面的方法。此处的术语“全面的”被用来强调整体和它局部间的相互依赖性的重要性。

Drools 5.0 现在被划分为 5 个模块，每个都自带手册——Guvnor (BRMS/BPMS), Expert (Rules), Fusion (CEP), Flow (Process/Workflow) 和 Planner。Guvnor 是我们的基于网页的管理系统，在规则世界中传统提及的，如一个 BRMS（商业规则管理系统）。我们决定抛弃一个担当管理工作的 BRMS 术语，因为它不是规则的细节。Expert 是传统的规则引擎。Fusion 是事件处理边，它担当数据/传感器的熔合术语。Flow 是我们的工作流模块，由 Kris Verlaenen 领导，已经做了一些了不起的工作；他正把流与 jBPM 5 合成一体。第 5 个模块是 Planner，由 Geoffrey De Smet 撰写，解决分配和调度类型的问题。虽然还处在开发的初期阶段，但已显示了许多期望。我们希望为 2011 版增加 Semantics（语义），基于描述逻辑，并让它担当下一代 Drools 引擎的一部分。

我一直工作在规则领域，现在大约 7 年了。我终于感觉到我紧握东西，想法开始成形，并且真正的创新开始出现。对我来说，感觉就象我确实知道我们现在将做什么，与过去相比，有许多疯狂的猜想和探索。我与 Edson Tirelli 和 Davide Sottara 一起努力工作在 Drools

Expert 设计文档上。我邀请你阅读文档和参与 http://community.jboss.org/wiki/DroolsLanguageEnhancements。该文档带有东西给下一级，推动 Drools 成为一个混合的引擎，不仅是一个有能力生产规则的系统，而且混合了逻辑编程（prolog）与功能编程，以及带有其他想法的更有表现力和现代感语言的描述逻辑。

我希望你能感觉到我们的团队和我在 Drools 上工作的热情，希望在你的冒险期间能感染你。

Mark Proctor (Drools 创建者和领导)

第 2 章　Drools 发行说明

2.1　在 Drools 5.1.0 中，值得注意的新东西。

2.1.1　Drools API

如在 Drools 5.0 中一样，仍然可以使用配置配置一个 KnowledgeBase（知识库），通过一个 xml 改变集合（change set），而不是编程方式。然而，现在的 change-set 命名空间被版本化了。这意味着，在 Drools 5.1.0 中，1.0.0 xsd 应该被引用。

例子 2.1 下面是一个简单的版本 1.0.0 改变集合

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
xs:schemaLocation='http://drools.org/drools-5.0/change-set change-set-5.0.xsd
http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/drools-api/src/main/resources/change-set-1.0.0.xsd' >
  <add>
      <resource source='classpath:org/domain/someRules.drl' type='DRL' />
```

```
      <resource source='classpath:org/domain/aFlow.drf' type='DRF' />
  </add>
</change-set>
```

2.1.2 核心

2.1.2.1 JMX 监控

JMX 监控被添加来支持知识库监控。这尤其重要，象一个常需要事件处理的长期运行流程。初始化整合 JOPR 也被添加。JMX 可以被启用，使用属性设置知识库：

```
drools.mbeans = <enabled|disabled>
```

或者这个选项在运行时启用：

```
kbaseConf.setOption( MBeansOption.ENABLED )
```

2.1.2.2 Spring

Drools 现在有扩展的 Spring 支持，XSD 可以在 drools-spring jar 中发现。命名空间为"http://drools.org/schema/drools-spring"。

例子 2.2 知识库构建器例子

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
```

```
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:drools="http://drools.org/schema/drools-spring"
       xmlns:camel="http://camel.apache.org/schema/spring"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
                           http://drools.org/schema/drools-spring http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/drools-container/drools-spring/src/main/resources/org/drools/container/spring/drools-spring-1.0.0.xsd
                           http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd">
  <drools:resource id="resource1" type="DRL" source="classpath:org/drools/container/spring/testSpring.drl"/>

  <drools:kbase id="kbase1">
    <drools:resources>
      <drools:resource type="DRL" source="classpath:org/drools/container/spring/testSpring.drl"/>
      <drools:resource ref="resource1"/>
      <drools:resource source="classpath:org/drools/container/spring/IntegrationExampleTest.xls"   type="DTABLE">

        <drools:decisiontable-conf input-type="XLS" worksheet-name="Tables_2" />
      </drools:resource>
    </drools:resources>

    <drools:configuration>
      <drools:mbeans enabled="true" />
      <drools:event-processing-mode mode="STREAM" />
```

```
    </drools:configuration>
  </drools:kbase>
</beans>
```

知识库接受下面的配置：″advanced-process-rule-integration, multithread, mbeans, event-processing-mode, accumulate-functions, evaluators and assert-behavior″。

根据 kbase 参考可以创建 ksessions

例子 2.3 知识会话

```
<drools:ksession id="ksession1" type="stateless" name="stateless1" kbase="kbase1" />
<drools:ksession id="ksession2" type="stateful" kbase="kbase1" />
```

象知识库的知识会话可以接受一些配置，包括"work-item-handlers, "keep-references", "clock-type", "jpa-persistence"。

例子 2.4 知识会话的配置

```
<drools:ksession id="ksession1" type="stateful"  kbase="kbase1" >
  <drools:configuration>
    <drools:work-item-handlers>
      <drools:work-item-handler name="handlername" ref="handlerid" />
    </drools:work-item-handlers>
    <drools:keep-reference enabled="true" />
    <drools:clock-type type="REALTIME"  />
  </drools:configuration>
```

```
</drools:ksession>
```

StatefulKnowledgeSessions（有状态知识会话）可以被配置为 JPA 持久化

例子 2.5 JPA 配置用于 StatefulKnowledgeSessions

```
<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.h2.Driver" />
  <property name="url" value="jdbc:h2:tcp://localhost/DroolsFlow" />
  <property name="username" value="sa" />
  <property name="password" value="" />
</bean>

<bean id="myEmf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFa
ctoryBean">
  <property name="dataSource" ref="ds" />
  <property name="persistenceUnitName" value="org.drools.persistence.jpa.local" />
</bean>

<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="myEmf" />
</bean>

<drools:ksession id="jpaSingleSessionCommandService" type="stateful" kbase="kbase1
">
  <drools:configuration>
```

```
    <drools:jpa-persistence>
      <drools:transaction-manager ref="txManager" />
      <drools:entity-manager-factory ref="myEmf" />
      <drools:variable-persisters>
        <drools:persister for-class="javax.persistence.Entity" implementation="org.drools.persistence.processinstance.persisters.JPAVariablePersister"/>
        <drools:persister for-class="java.lang.String" implementation="org.drools.container.spring.beans.persistence.StringVariablePersister"/>
        <drools:persister for-class="java.io.Serializable" implementation="org.drools.persistence.processinstance.persisters.SerializableVariablePersister"/>
      </drools:variable-persisters>
    </drools:jpa-persistence>
  </drools:configuration>
</drools:ksession>
```

知识库会话可以支持启动批处理脚本，前面的版本使用"script"元素名，它将被更新为"batch"。下面的命令被支持："insert-object", "set-global", "fire-all-rules", "fire-until-halt", "start-process", "signal-event"。可以使用匿名 Bean 或命名"ref" 属性。

例子 2.6 启动批处理命令

```
<drools:ksession id="jpaSingleSessionCommandService" type="stateful" kbase="kbase1
">
  <drools:script>
    <drools:insert-object ref="person1" />
    <drools:start-process process-id="proc name">
      <drools:parameter identifier="varName" ref="varRef" />
    </drools:start-process>
```

```
    <drools:fire-all-rules />
  </drools:script>
</drools:ksession>
```

在 Spring 中支持执行节点(ExecutionNodes)，它们提供了一个注册的 ksessions。它可以与 Camel 一起提供 ksessions 路由。

2.1.2.3 Camel

Spring 可以与 Camel 组合，提供声明式规则服务。一个 Camel 策略从 Drools 被添加，Drools 提供魔法注入类加载器（ClassLoader），类加载器被 ksession 用于任何数据的格式化，它还增强了 JAXB 和 XStream 数据的格式化。假如在 Jaxb 情况下，它添加另外的 Drools 路径信息，利用 XStream 注册 Drools 相关联的转换器，并取别名。

你可以使用不同的地址，创建你需要的尽可能多的端点。需要 CommandMessagBodyReader 分配有效负荷给 Camel 处理。

例子 2.8 Rest 端点配置

```
<cxf:rsServer id="rsServer"
              address="/kservice/rest"
              serviceClass="org.drools.jax.rs.CommandExecutorImpl">
  <cxf:providers>
    <bean class="org.drools.jax.rs.CommandMessageBodyReader"/>
  </cxf:providers>
</cxf:rsServer>
```

然后，Camel 路由可以连接到 CXF 端点，允许你控制如数据的格式化和根据 Drools ksessions 执行等事情的有效负荷。Drools 策略增加某些智能给路由。如果使用 JAXB 或 XStream，它将注入定制路径和转换器，它也可以在服务器边设置类加载器，基于目标 ksessions 的。在客户边，它自动解包 Response（响应）对象。

这个例子使用了一个增强的 XStream 的数据格式解组有效负荷，并且根据 ksession1 实例执行它。"node"在这儿引用了 ExecutionContext（执行上下文），它是一个注册的 ksessions 上下文。

例子 2.9 Camel 路由

```
<bean id="droolsPolicy" class="org.drools.camel.component.DroolsPolicy" />

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="cxfrs://bean://rsServer"/>
    <policy ref="droolsPolicy">
      <unmarshal ref="xstream" />
      <to uri="drools://node/ksession1" />
      <marshal ref="xstream" />
    </policy>
  </route>
</camelContext>
```

Drools 端点 "drools:node/ksession1"，由执行节点名字、分隔符和可选的知识会话名字构成。如果知识会话没有指定路由，会查找在输入的有效负荷的实例中的"lookup"属性，或者在头部属性"DroolsLookup" 中去查找。

2.1.2.4 Drools 服务器

Spring, Camel 和 CXF 可以被组合，用于声明式服务，drools-server 是一个.war，组合了它们与某些开箱即用的例子 xml，就像一类模板。如果你正在 jboss 容器中使用 war,，那么你需要添加这个组件，http://camel.apache.org/camel-jboss.html。该 war 包含了一个 test.jsp，显示一个象你开始的例子一样的回显。这个例子只是执行一个简单的"echo" 类型的应用程序。它发送一个消息到规则服务器，规则服务器预先增加单词"echo"到开头，并发送它回去。默认时，消息是"Hello World"，使用 url 参数 msg —— test.jsp?msg="My Custom Message"，可以传递不同的消息。

2.1.2.5 知识代理增量变化支持

新版的知识库代理在它配置中支持 newInstance = false（增量变化集合构建）。

当设置这个属性为 false，知识库代理使用存在的知识库参考应用增量变化。现在知识库代理可以用增量的方式处理监控的资源更改。更改的定义被编译，并和原始版本比较。根据定义的类型，以不同的方式表现：

- Rules: 用于规则，代理搜索在它的属性中的更改，左手边和右手边。
- Queries: 查询总在 kbase 上被替代，无论它们被更改还是没有被更改。
- 其他定义: 所有其他定义总是在 kbase 中被替代（只要它们被更改）。我们希望在将来的版本中，为更改定义的探测增加更好的支持。

当前的实现只支持规则、查询和函数定义的探测。类型声明不能被探测。

2.1.2.6 会话检查与报告框架

一个用于运行时会话检查与报告的基于框架的新的 API 被引入，从而更好地在调试或应用程序成型时采集数据。这个检查框架将成为加工功能的基础，用来帮助为每个会话的内容提

供更详细的信息。这个 api 是实验性的，并不是用在 drools-api 中，但可自由的使用，并帮助我们改进它。

要检查一个会话，可以使用以下 api 调用之一：

例子 2.10 创建一个会话检查器

```
StatefulKnowledgeSession ksession = ...

// ... insert facts, fire rules, etc
SessionInspector inspector = new SessionInspector( ksession );
StatefulKnowledgeSessionInfo info = inspector.getSessionInfo();
```

StatefulKnowledgeSessionInfo 实例将包含许多在会话分析期间采集的相关数据。提供了一个简单的例子报告模板，并且可以利用下面的 api 调用产生：

例子 2.11 产生一个报告

```
String report = SessionReporter.generateReport( "simple", info, null );
```

2.1.3 Expert

2.1.3.1 差异更新

传统 Rete 算法用 retract + assert（撤消+断言）做一个更新，对一个给定的事实，这导致所有的局部匹配被撤销，然而，在 assert（断言）期间，其中一些会被再次重建；因为它们在更新前为 true，并且在更新后仍然为 true。这导致许多不需要的对象被摧毁，并且这些行为增加了垃圾收集器的负荷。现在的更新是一个单程，并在适当的位置检查局部匹配，避免不必要的局部匹配被摧毁。经历事件和实事维护的一个正规化过程也不再需要；正规化过程

是我们准备着眼于激活 retract（撤消）的地方，被插入的激活断定真正被添加的东西，由真正插入的东西确定“差异”。

2.1.3.2 Channels（通道）

退出点已经用更合适的名字 Channels（通道）替代，我们觉得它更合适，因为多个规则引擎可以使用它们，并且不是切入点的精确对立面。切入点明显与进入 Rete 网络中的一个分区有关联。

2.1.3.3 现场查询

Doorls 一直有一个查询支持，但结果是以一个可迭代的集合返回；这使监测随着时间的变化很困难。

我们现在用现场查询实现它，它与一个侦听器相连，而不是返回一个可迭代的集合。这些现实查询持续打开创建的一个视图，并且为这个视图的内容发布变化事件。所以，你现在可以执行你的查询，利用参数，并在结果视图中侦听变化。

例子 2.12 实现 ViewChangedEventListener

```
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 LiveQuery

LiveQuery query = ksession.openLiveQuery( "cheeses",     new Object[] { "cheddar", "stilton" },  listener );
...

...
query.dispose() // make sure you call dispose when you want the query to close
```

一个 Doorls 博客文章包含了一个用于现场查询的 Glazed 列表集成的例子

http://blog.athico.com/2010/07/glazed-lists-examples-for-drools-live.html（不可用）

[http://planet.jboss.org/view/post.seam;jsessionid=8F25E6F156E3093608A38BDCCE17369C?post=glazed_lists_examples_for_drools_live_querries]

2.1.3.4 定时器和日历

规则现在支持基于 interval（间隔）和 cron 定时器，替代了过时的 duration（期限） 属性。

例子 2.13 定时器属性使用的例子

```
timer ( int: <initial delay> <repeat interval>? )

timer ( int: 30s )

timer ( int: 30s 5m )

timer ( cron: <cron expression> )

timer ( cron:* 0/15 * * * ? )
```

Interval "int:" 定时器遵守 JDK 语义，初始化延迟，紧跟一个可选的重复间隔。Cron "cron:" 定时器遵守标准的 cron 表达式：

例子 2.14 一个 Cron 例子

```
rule "Send SMS 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
```

当规则被引发时，现在可以控制日历。日历 api 是模拟 Quartzhttp://www.quartz-scheduler.org/ ：

例子 2.15 调节一个 Quartz 日历

```
Calendar weekDayCal = QuartzHelper.quartzCalendarAdapter(org.quartz.Calendar quartzCal)
```

日历使用 StatefulKnowledgeSession（有状态知识会话）注册：

例子 2.16 注册一个日历

```
ksession.getCalendars().set( "week day", weekDayCal );
```

它们可以同时用于普通规则和包含定时器的规则中。 规则的 calendar 属性可以有一个或多个逗点日历名字：

例子 2.17 一起使用日历和定时器

```
rule "weekdays are high priority"
  calendars "weekday"
  timer (int:0 1h)
when
   Alarm()
then
   send( "priority high - we have an alarm );
end
```

```
rule "weekend are low priority"
  calendars "weekend"
  timer (int:0 4h)
when
   Alarm()
then
   send( "priority low - we have an alarm );
end
```

2.1.3.5 决策表（Excel）

2.1.3.5.1 简单模板，用于单元格内部可变长度逗点分隔的列表

在一个单元格中可以有一个逗点分隔的列表，并且用一个 forall 模板可以渲染它们。

例子 2.18 DTable forall 语法

```
forall(<separator>?){<codesnippt>}
```

例子 2.19 DTable forall 例子

```
forall(,) {propertyName == $}
forall(&&) {propertyName == $}
forall(||) {propertyName == $}
```

forall(||) {propertyNameA == $} && forall(||){propertyNameB == $}

等等

2.1.4 Flow

2.1.4.1 BPMN2

正如我们早期已宣称的一样，Dools 团队已决定支持使用 XML 指定业务流程的即将到来的 BPMN 2.0 规范。这个里程碑包含了 BPMN2 解析器的重大扩展，使用 Drools Flow 来支持 BPMN2 的更多功能。更具体的有：

- 更广泛的事件支持：更多的事件类型组合（开始、 中间、结束）和事件触发（例如，包括错误、升级、定时器、条件、信号事件），以及已包括（中断和非中断）边界事件。
- 子流程参数。
- 发散的综合性出入口
- 等等

BPMN2 已被集成在完整的 Dools 工具链中，支持业务流程的整个生命周期。它包括：

- 能够与我们的 Eclipse 工具一起使用 BPMN2 流程。
- Guvnor 作为流程库。
- 基于网页的管理，使用 BPM 控制台。
- 审计和调试。
- 特殊域流程。
- 等等

因此，Drools Flow 不仅是一流的开源流程引擎，天然地支持这样一个重要的 BPMN2 结构集合，而且我们的面向知识的方法，也允许你容易组合你的 BPMN2 流程与业务规则，以及复杂的事件处理，完全使用相同的 APIs 和工具。

2.1.4.2 基于网页的管理控制台

现在也可以通过一个网页控制台管理 Drools Flow 流程。它包括的功能有：管理你的流程实例（开始/停止/检查），检查你的（人的）任务列表，执行那些任务，产生报表。

这个控制台实际上是 Heiko Braun 的工作（卓越的），他创建了一个通用的 BMP 控制台，可以被用来支持多种流程语言。我们实现了必要的组件，允许这个控制台能与 Drools Flow 引擎通信。

2.1.4.3 可插式可变持久化

Drools Flow 可以持久化运行流程的运行时间状态到一个数据库（所以它们完全不需要在内存，并且在故障情况时可以被存储）。我们的默认持久化机制是用一个二进制对象（利用相关的元数据）存储关联于一个流程实例的所有运行时间信息。关联这个流程实例（又名流程实例变量）的数据也作为这个二进制对象的一部分被存储。然而这可以产生问题（1）在数据没有被序列化时，（2）当用流程实例状态的部分持久化，对象太大时，或者（3）在其他地方已经被持久化时。为此，我们实现了可插式可变持久化，用户可以定义多少变量的值被存储。例如，它允许你存储个别变量的值，并且支持 JPA 实体被分别存储和引用（避免状态重复）。

2.1.4.4 改进的流程实例移植

随着时间的推移，流程可以进化。每当一个流程更新，确定已经运行的流程实例会发生什么是重要的。我们改进了我们的支持，用于移植运行的流程实例到流程定义的一个新版本。更多信息请看 Drools Flow 文档。

2.1.4.5 安装脚本

Drools 构建输出了一个安装器，它简化了 Guvnor 和 gwt-控制台的 Eclipse 插件的安装。它创建了和复制了需要的 jars 和 war，并且布置它们到 JBoss AS。它也包含了一个简单的求值流程例子，你可以用它测试你的安装。更多信息，下载 Drools 安装器，请看其内部的自述文件 readme。

2.1.5 Guvnor

外观已被清洁，例如少了弹出窗口。意味着用断言和有关行为发生的信息更改之后的节省，以及如果出错时，更好的错误报告。

2.1.5.3 讨论

注释是在下面的“文档”部分（当然是可选的）（并且有一个 Atom 供应给它们）

图 2.1 实时讨论

一个"backchannel（反向通道）"类型连接，保持了对浏览器的开放，允许消息推回——这意味着（在启用时）信息实时显示（和其他便利的事情一样，如果某东西添加到一个列表——列表就会被更新）

2.1.5.4 收件箱

收件箱功能提供了这样的能力，跟踪你打开或编辑的东西——这显示在主导航器中的"Inbox"项目下。

图 2.2 收件箱分类

●·最近打开的

点击 recently opened 条目，会打开“最近”打开过的所有项目的一个列表（它跟踪最近你看过的数百个项目）。

●·最近编辑的

所有你保存过的项目，或者注释，将马上显示在这里。

●·收到的更改

它跟踪“其他人”对“你”的“最近编辑”列表中的项目所做的更改。当你打开这些项目，那么它们会从这个列表中删除（但仍然保留在你最近编辑过的列表中）

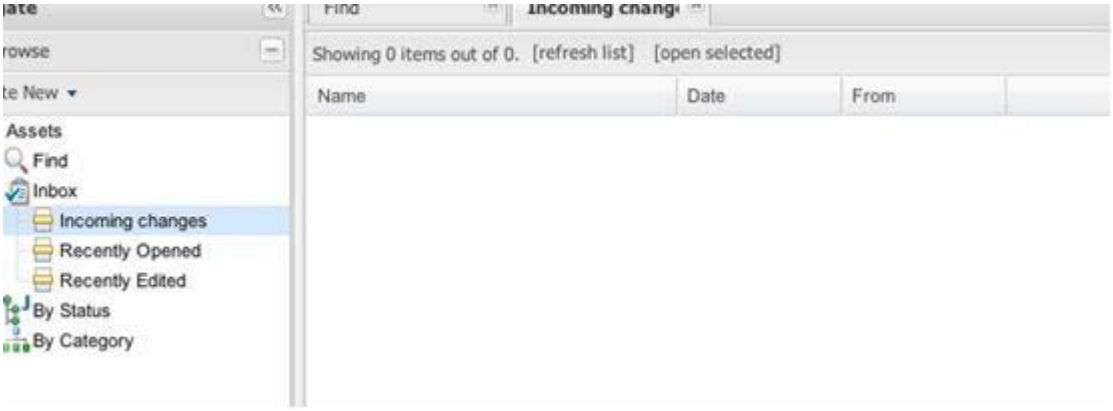


图 2.3 收件箱项目列表。

2.1.5.5 批量导入器

Guvnor 导入器是一个 maven 构建工具，递规你的规则目录结构，并且构建一个导入文件，通过 import/export 管理功能可以手动导入它到 Drools-Guvnor 网页界面。

2.1.5.6 DroolsDoc

PDF 文档包含了有关包和每个 DRL 资产（asset）的信息。DroolsDoc 作为知识包可以从 "Information and important URLs"下的包视图（package view）下载。

2.1.5.7 更新为 GWT2.0

GWT 被更新，使 Drools Guvnor 更快。

2.1.5.8 安装选择器

安装选择器允许用户选择什么样的资产构建，依据：

- 状态（例如，开发，质量保证等）
- 分类
- 元数据

2.1.5.9 单资产验证

只要是你正在运行的资产（规则流、规则、决策表），就可以验证。验证发现诸如在一个规则中的冲突限制或在决策表中的多余行等问题。

2.1.5.10 全局区

储存在全局区的资产可以被所有包共享。

2.1.5.11 快照之间的差异检查

列出两个快照之间的变化。

2.1.5.12 在一个标签中查看多个资产

可以在一个视图中打开多个资产。所有的资产可以作为一个组被保存和编辑。

2.1.5.13 From/Collect/Accumulate 支持

指导编辑器已基本支持 From, Accumulate 和 Collect 模式。你可以添加它们中的任何一个结构作为正规模式。新的表达式构建器组件被创建，增加了对嵌套方法调用的一个变量支持。在规则的 WHEN 部分顶部使用了“加”按钮，或者使用呈现在每个模式中的新的“在后面增加”按钮，将打开弹出，添加新的条件元素到你的规则中。在可能元素的列表中的，你会发现三个新的条目：”From”, “From Accumulate”和 “From Collect”。

当你添加一个新的”From”元素，你会在指导编辑器看到下面这样图象的东西。左边模式的”From”条件元素是一个正规模式。你可以在这里添加你希望的任何类型的条件元素。右边部分的”From”模式是一个表达式构建器。

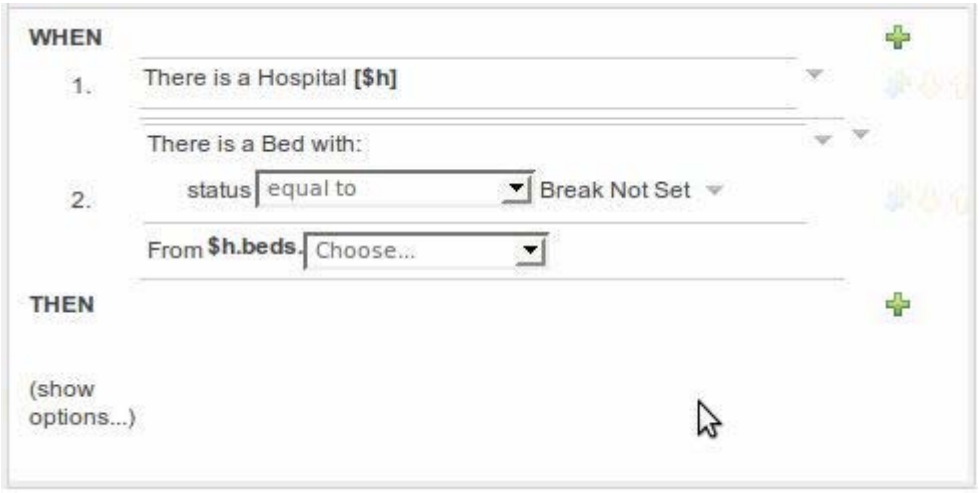


图 2.4 来自一个 CE 构建器

当你在左边模式中使用“From Collect”时，你可以选择“java.util.Collection”, “java.util.List” 或“java.util.Set” 事实类型。这个事实类型会被自动包含在事实类型列表中。

右边模式的收集条件元素可以是下面模式之一：

- Fact Type Pattern（事实类型模式）
- Free Form Expression（自由表单表达式）
- From Pattern（From 模式）
- From Collect Pattern（From Collect 模式）

■·From Accumulate Pattern（From Accumulate 模式）

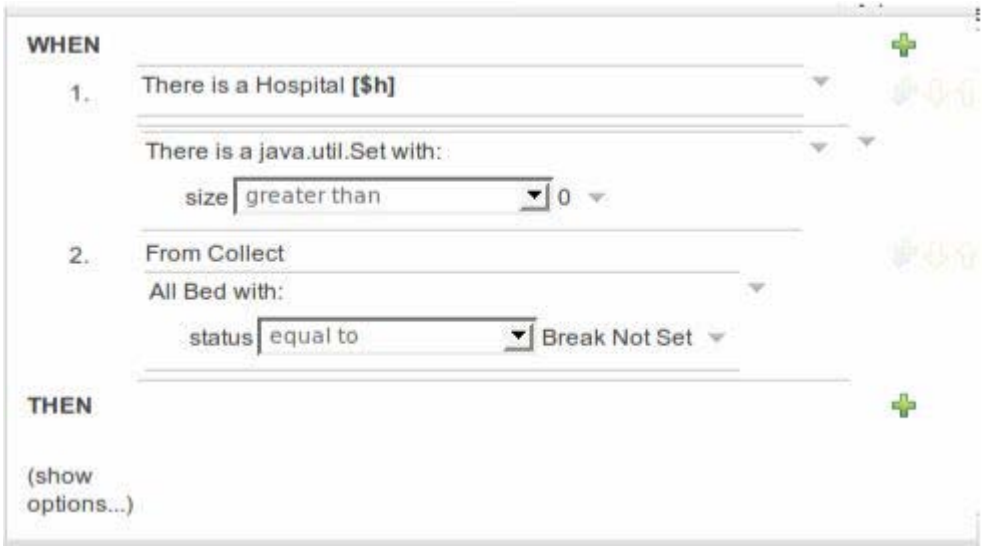


图 2.5 来自 Collect CE 构建器

当使用“From Accumulate”时，左边模式可以是任何事实类型模式。这个条件元素的右边部分被划分成两个：

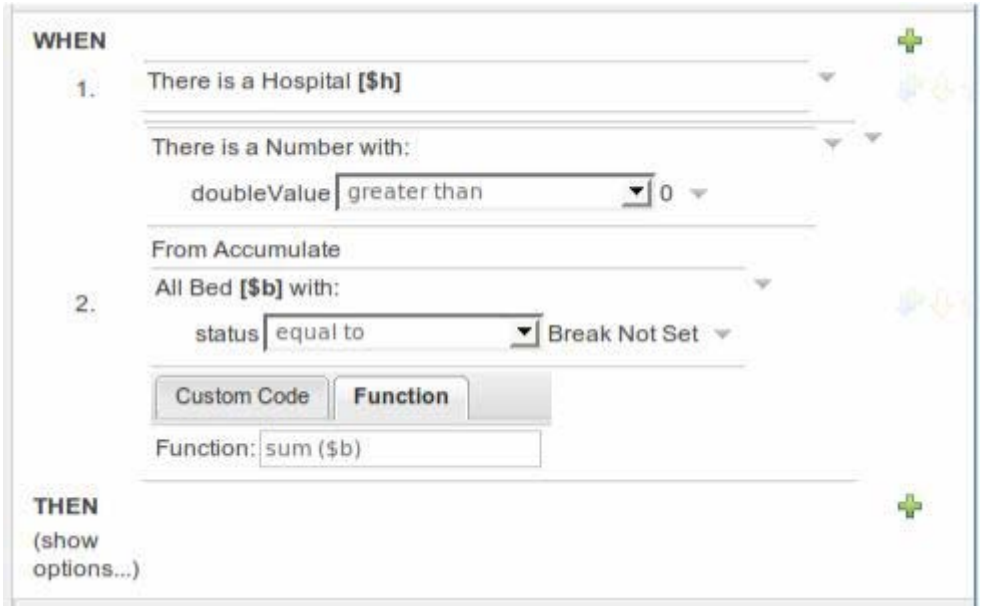


图 2.6 来自累积 CE 构建器

左边模式可以是任何事实类型模式。这个条件元素的右边部分被划分成两个：

源模式：（在载图中的，Bed $n）可以是任何事实类型、From、Collect 或者 Accumulate 模式

累积函数：在这里你可发现一个标签式面板，你可以输入一个累积函数（在截图中的 sum()），或者你可以使用“Custom Code”创建一个在线定制函数。

2.1.5.14 规则模板

规则模板可以被指导编辑器用于编辑复杂的规则，然后通过一个电子表格的表格数据隐喻轻松地编写它。替代了一个字段的值，仅是用名为"Template Key"标识它，并且该 key 可以作为网格中的一个列。每行可以应用规则模板来产生一个规则。

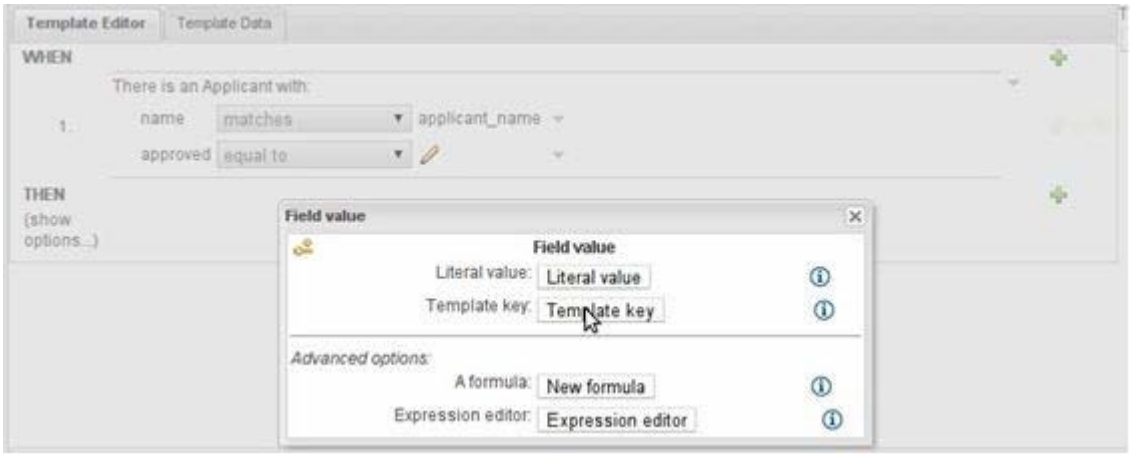


图 2.7 在'WHEN'部分增加了一个 Template Key。

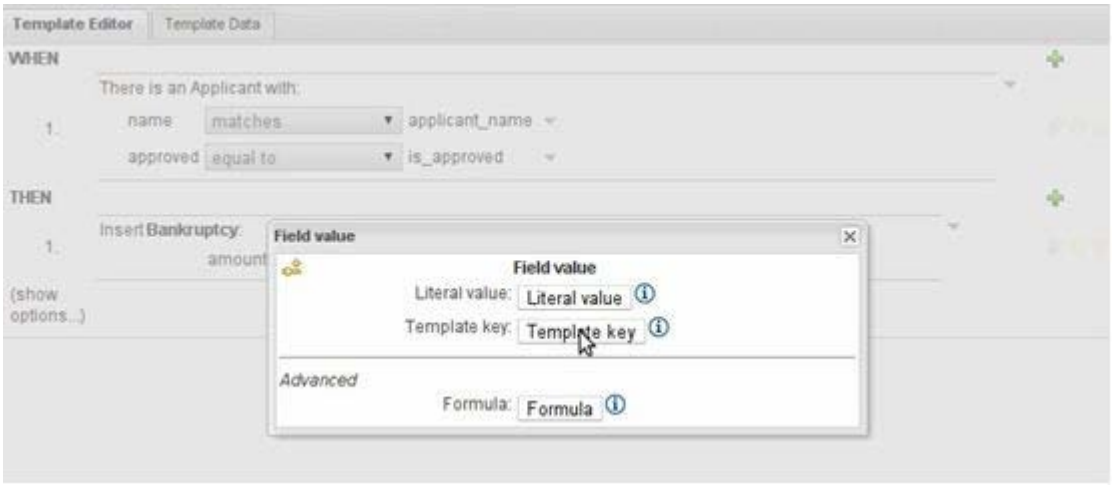


图 2.8 "THEN"部分增加了一个 Template Key。

图 2.9 根据这些 Template Key 填充行。

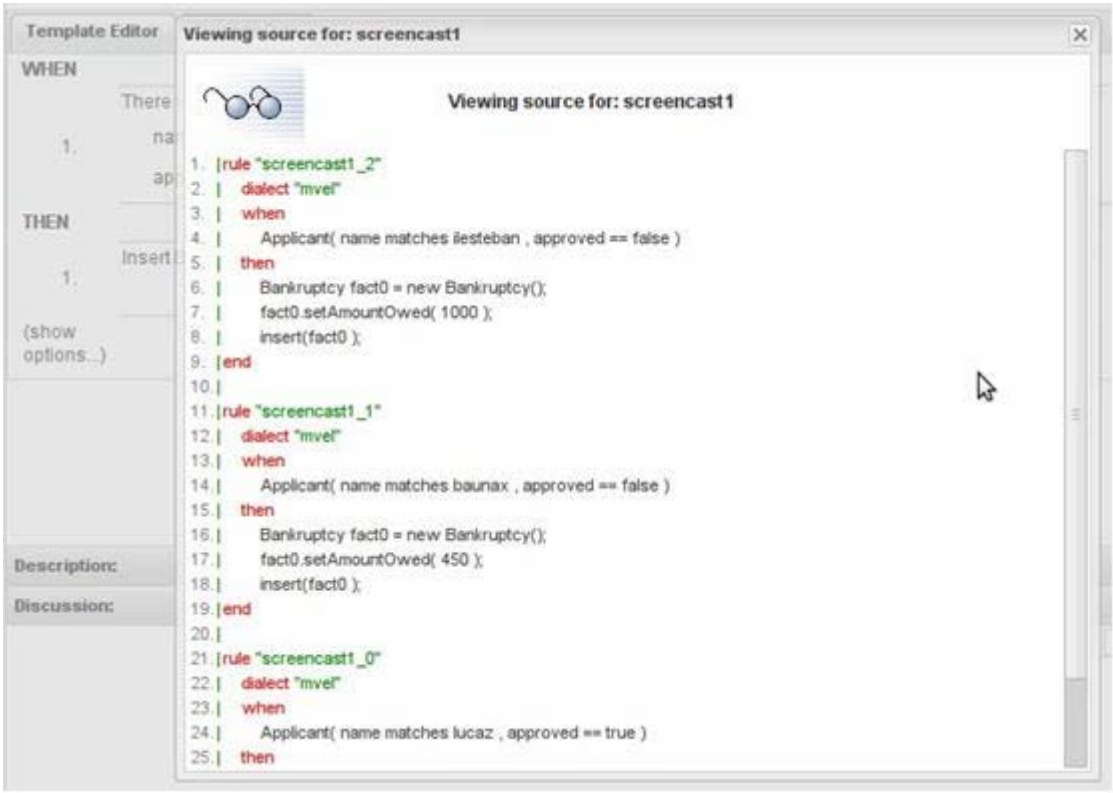


图 2.10 每行产生一个规则

2.1.5.15　工作集

在建模规则时，用户暴露所有的事实类型，它可能是整个的很小部分。工作集允许相关的事实类型分组在一起，并在编写规则时，提供一个更易于管理的选择事实类型的视图。

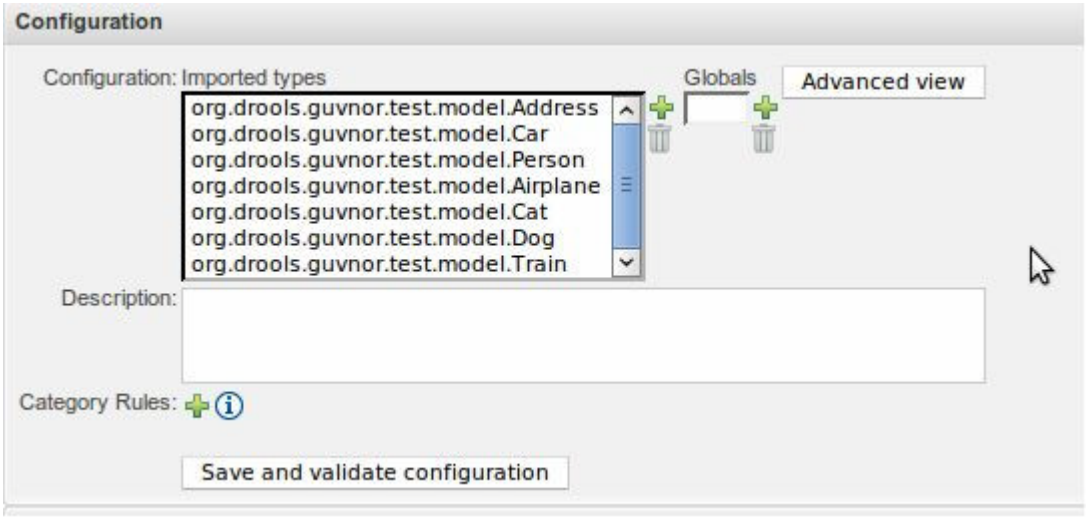


图 2.11 显示导入的类型。

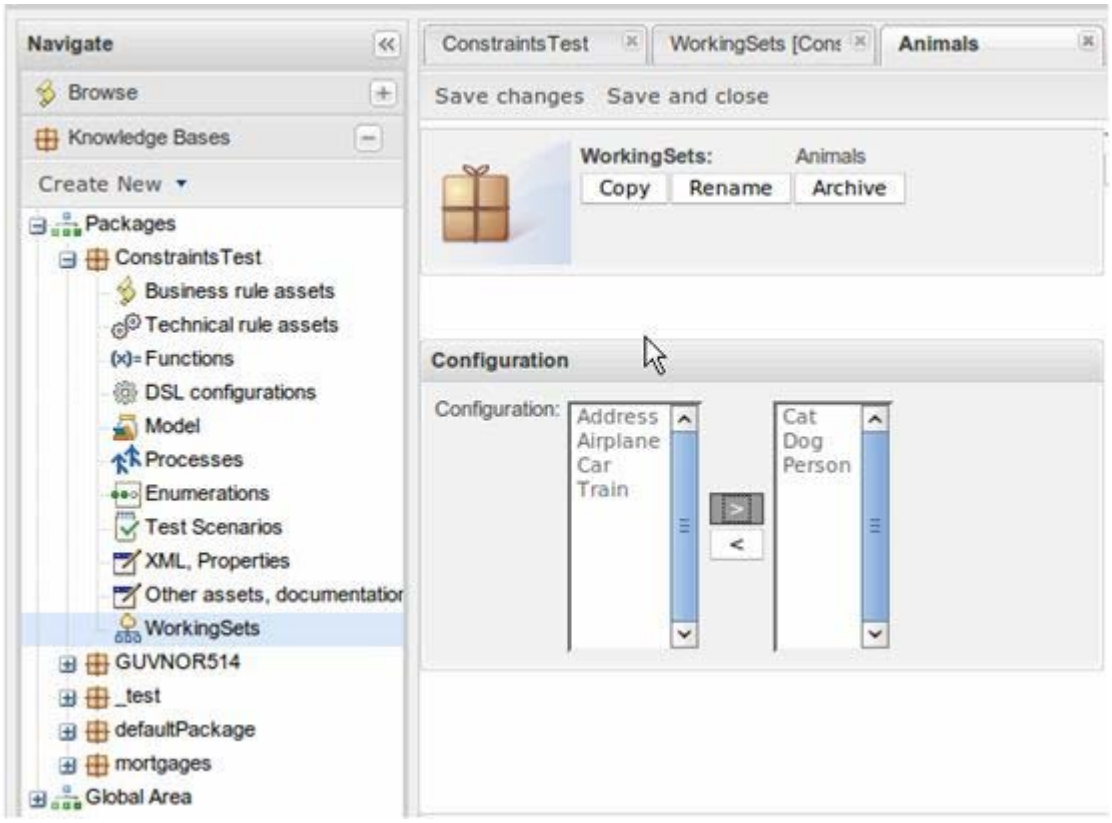


图 2.12 创建一个动物工作集。

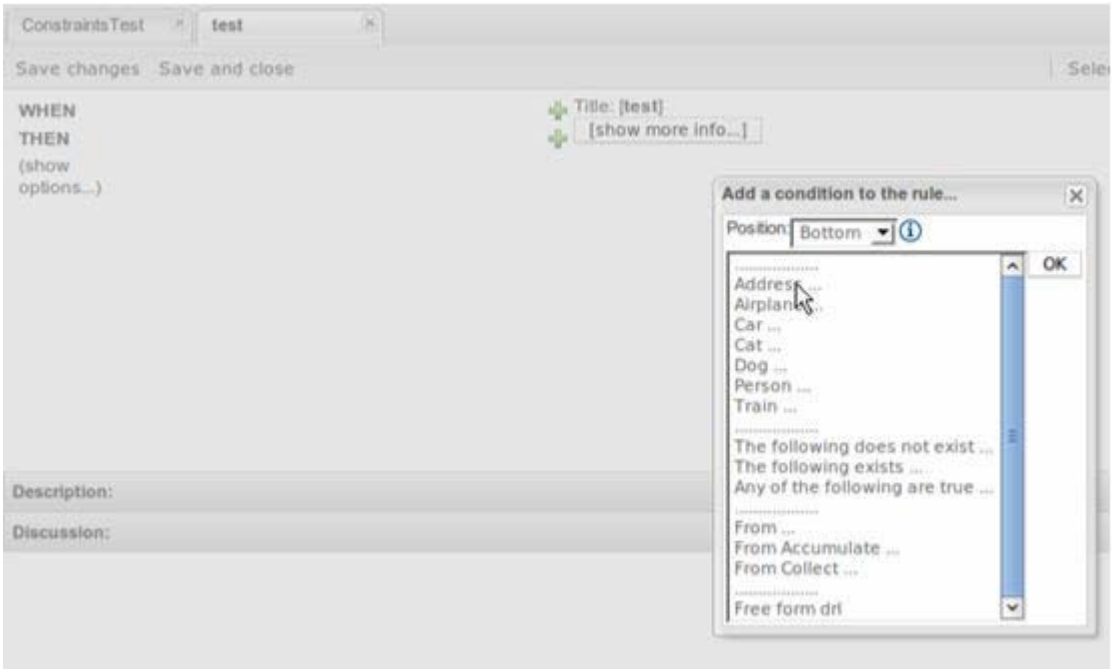


图 2.13 没有使用工作集，所有类型是可见的。

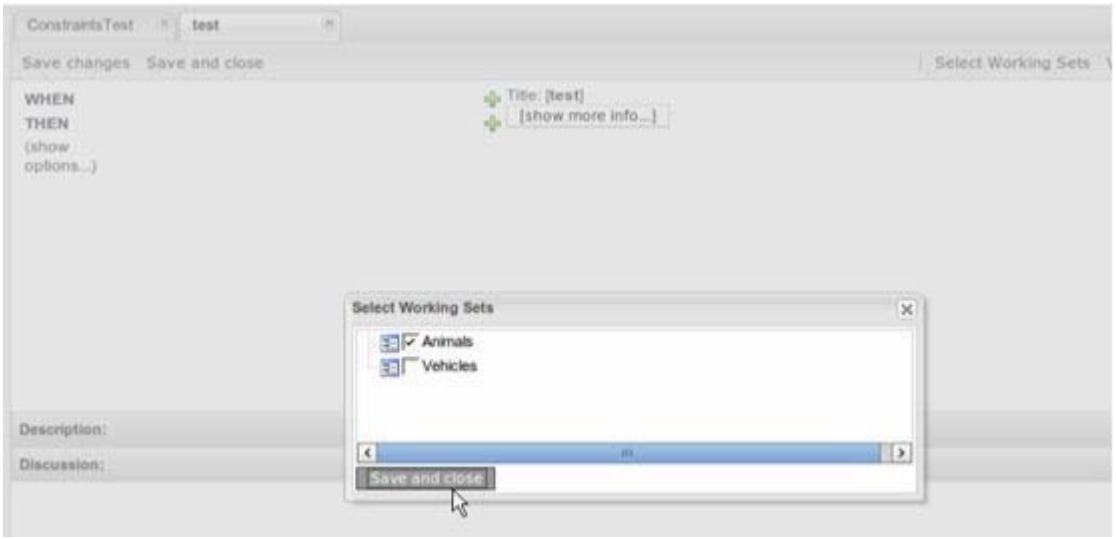


图 2.14 选择动物工作集。

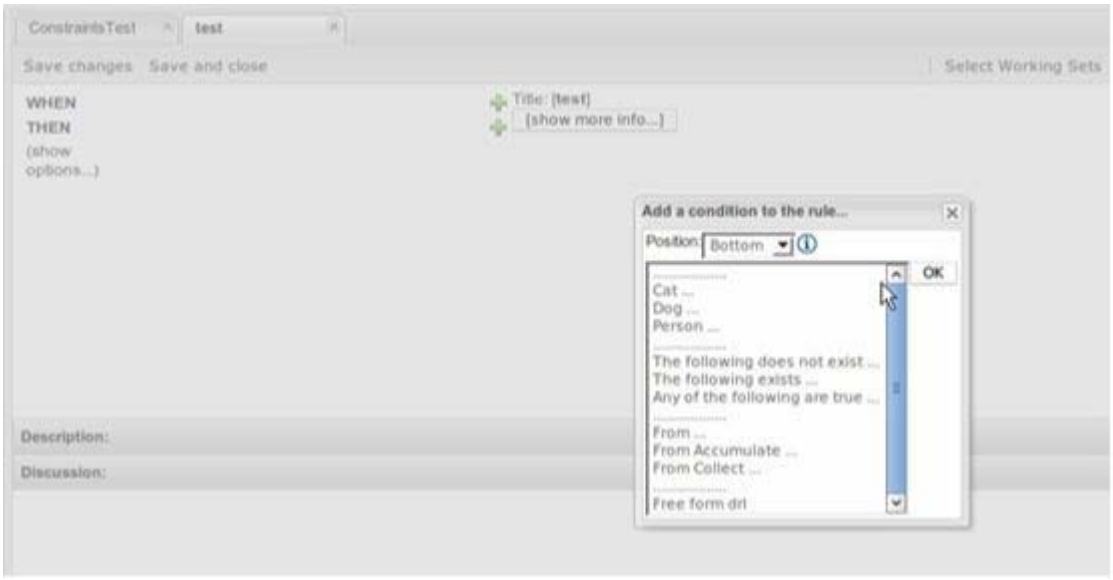


图 2.15 可用的类型减少了。

2.1.5.16 事实约束

工作集可以与事实约束相结合，提供额外的设计时间验证。例如，如果你依据某个人的年龄编写一个规则，在设计时，我们可以分辨有效的范围，并且使用它来约束作者。事实约束是工作集的一部分，并且在编辑一个规则时，你必然选择工作集约束，那个你希望用于作为验证处理部分的约束。

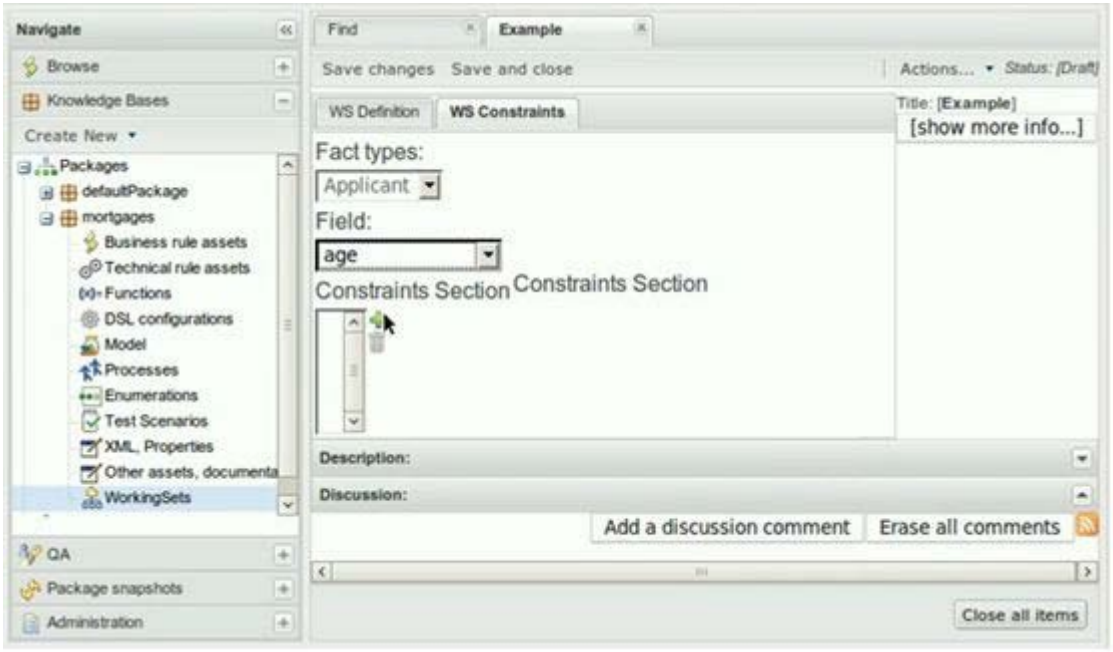


图 2.16 选择字段。

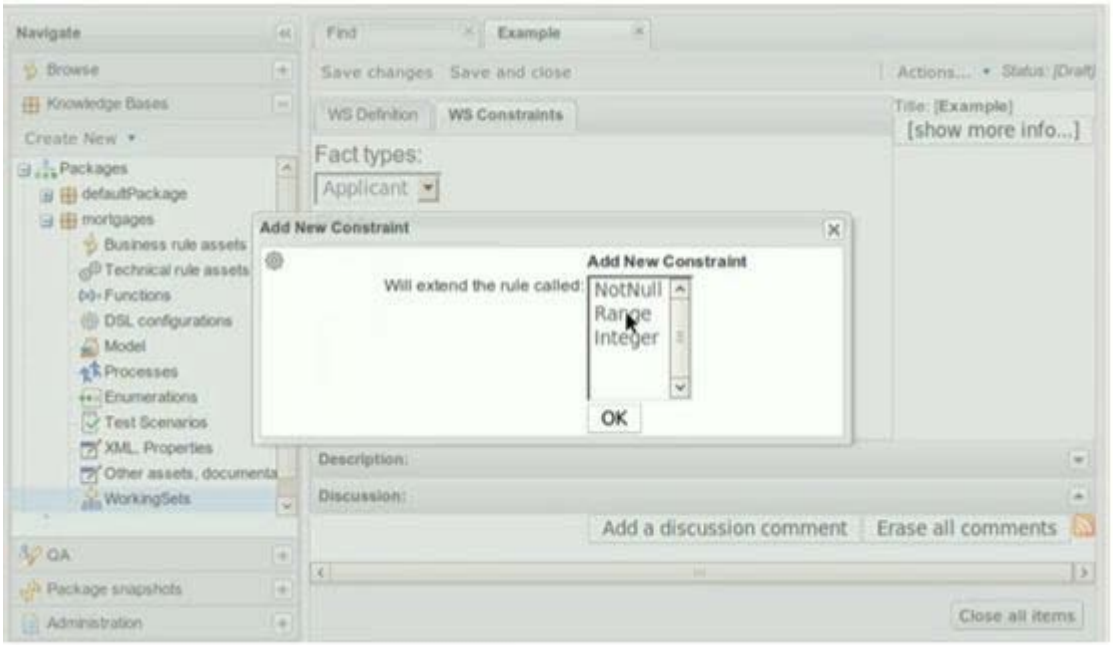


图 2.17 选择字段约束类型.

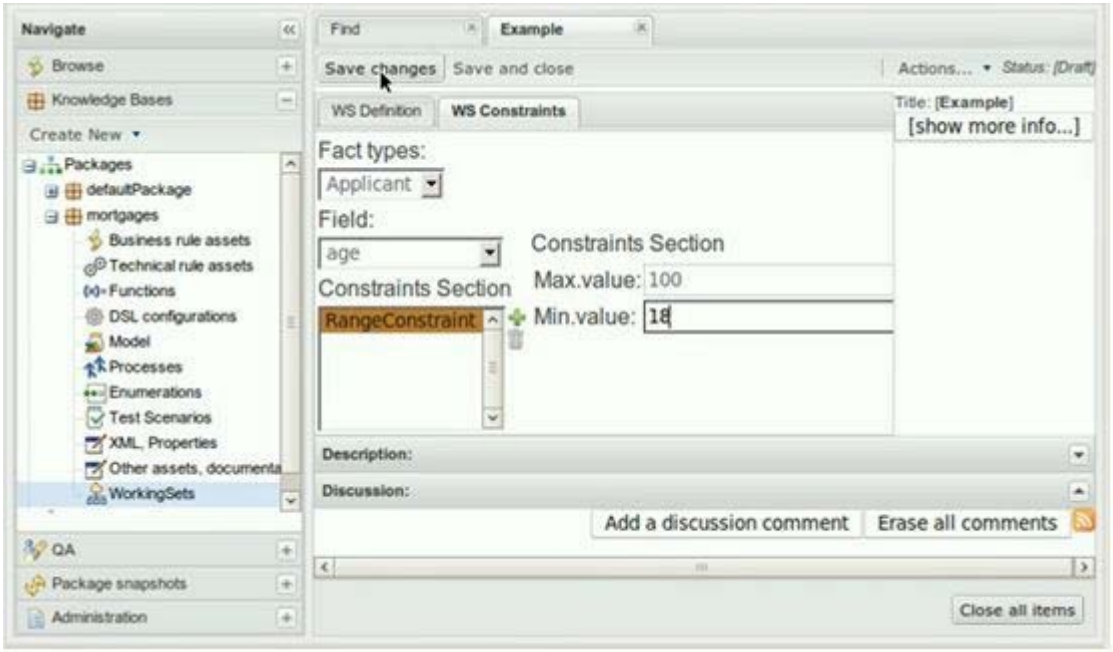


图 2.18 一个范围选择的例子。

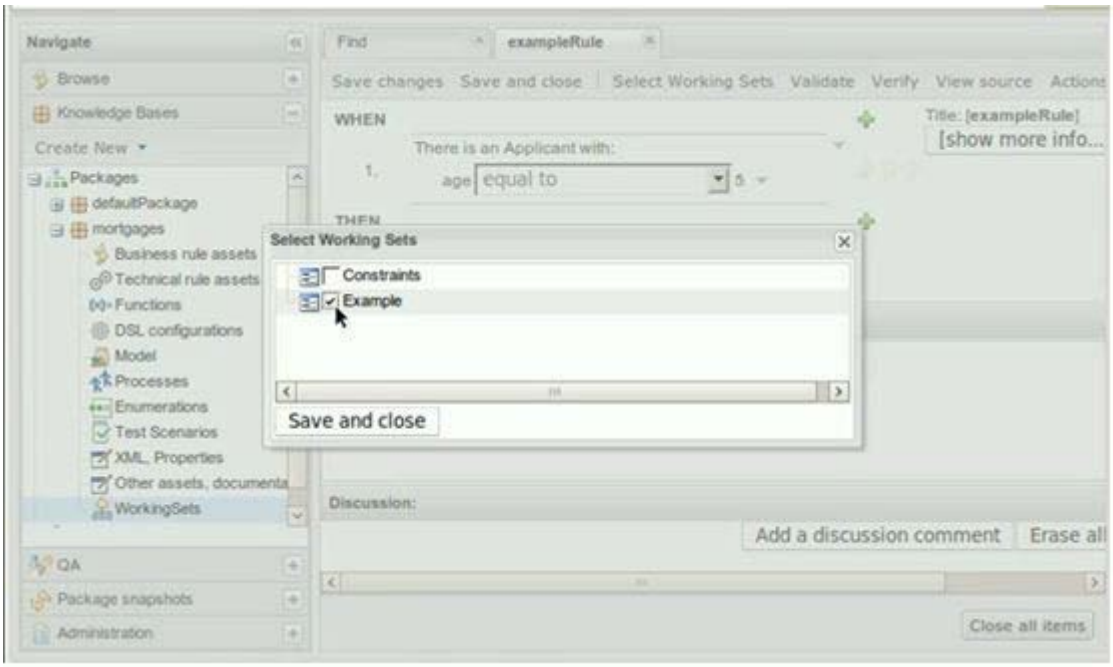


图 2.19 选择工作集，用于验证年龄字段。

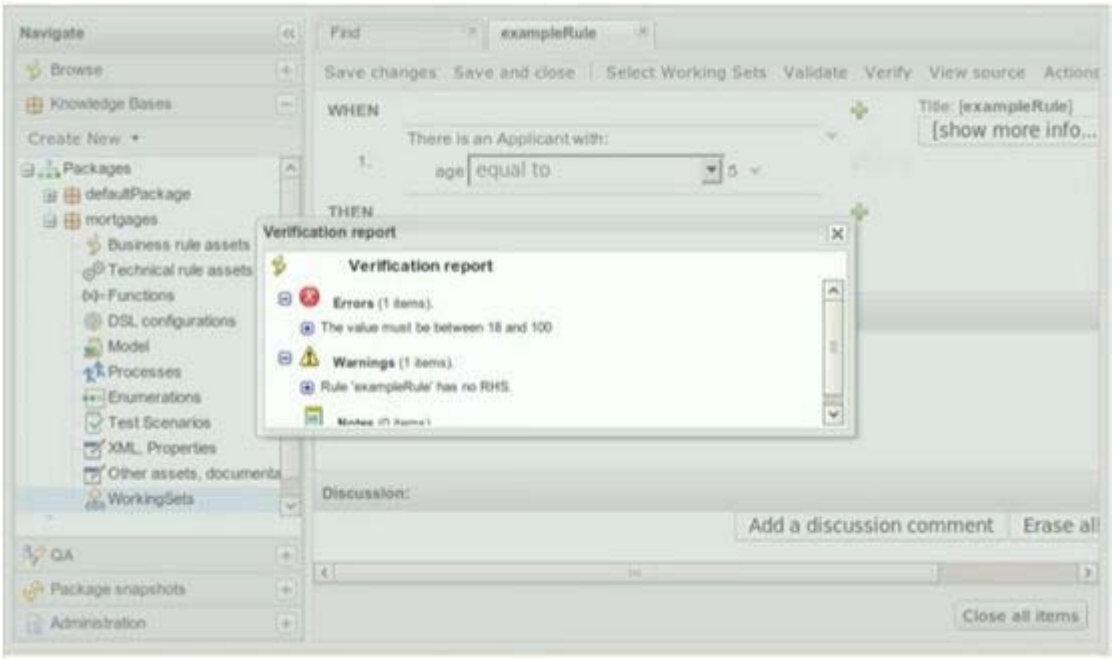


图 2.20　5 岁是无效的。

2.1.5.17　指导规则编辑器

编辑规则变得更明确。编辑器是更小“盒子”，且用一个普通文本编写更多规则。增加了"contains"关键字，现在在绑定变量给限制是更容易了。

2.1.5.17.1　模式排序

指导编辑器在 lhs 和 rhs 部分支持模式重排序，以及位置插入，新的模式可以以任何顺序被插入（并不总是在最后）。

图 2.21　上下移动元素。

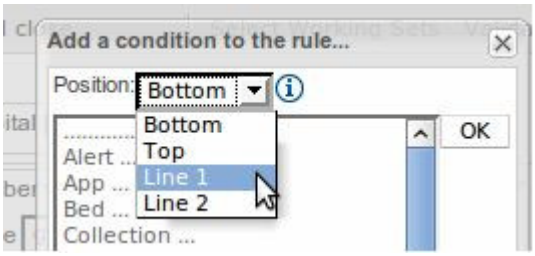


图 2.22 在任何位置插入元素。

2.1.5.18 决策表（Guvnor）

增加了关键字"in"。

可以移动列，并且新行的位置可以自由选择。

2.1.6 Eclipse

2.1.6.3 在大纲视图中分组规则

现在你可以为议程组使用排序或分组。

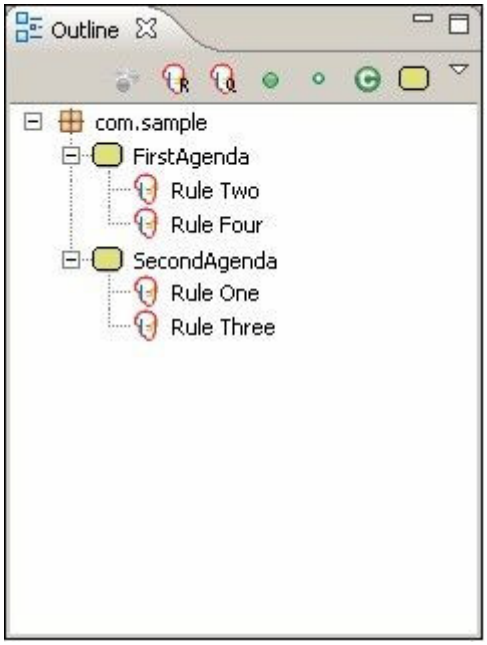


图 2.23 分组议程组

图 2.24 排序议程组

2.1.6.4 支持在审计视图中拖/放文件。

现在可以拖放日志文件到审计视图中。

2.1.7 已知的问题

2.1.7.3 多线程模式

使用试验的多线程执行模式存在一个已知的问题，如下面 JIRA 中描述的一样：

2.2 在 Drools 5.0.0 中值得关注的新东西

2.2.1 Drools API

Drools 现在有完善的 api/实现分离物，不再面向规则。当我们开始支持其他逻辑形式时，如工作流和事件处理时，这是一个重要的策略。最重要的改变是我们面向知识，而不是面向规则。drools-api 模块提供接口和工厂，并且与以前相比，我们已竭力提供更好的 javadocs，带有一些代码片段。drools-api 还有助于清楚地显示有意作为一个用户的 api 是什么，只是一个引擎的 api 是什么，而 Drools 核心和 Drools 编译器没有充分明确这点。你将使用的最普通的接口有：

- org.drools.builder.KnowledgeBuilder
- org.drools.KnowledgeBase
- org.drools.agent.KnowledgeAgent
- org.drools.runtime.StatefulKnowledgeSession
- org.drools.runtime.StatelessKnowledgeSession

工厂类，带有静态方法，提供上面接口的实例。一个可插式提供者方法被用来允许提供者实现在运行时连接工厂。你将最常使用的工厂有：

- org.drools.builder.KnowledgeBuilderFactory
- org.drools.io.ResourceFactory
- org.drools.KnowledgeBaseFactory
- org.drools.agent.KnowledgeAgentFactory

例子 2.20 加载一个规则资源的一个典型例子

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ), ResourceType.DRL );
if ( kbuilder.hasErrors() ) {
  System.err.println( builder.getErrors().toString() );
}

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addKnowledgePackages( builder.getKnowledgePackages() );

StatefulKnowledgeSession ksession = knowledgeBase.newStatefulKnowledgeSession();
ksession.insert( new Fibonacci( 10 ) );
ksession.fireAllRules();

ksession.dispose();
```

一个典型的加载一个流程资源的例子。注意 ResourceType 使用相应的资源类型被改变。

例子 2.21 一个典型的加载一个流程资源的例子。注意 ResourceType 使用相应的资源类型被改变。

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
        ResourceType.DRF );
if ( kbuilder.hasErrors() ) {
  System.err.println( builder.getErrors().toString() );
}
```

```
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addKnowledgePackages( builder.getKnowledgePackages() );

StatefulKnowledgeSession ksession = knowledgeBase.newStatefulKnowledgeSession();
ksession.startProcess( "Buy Order Process" );

ksession.dispose();
```

'kbuilder'，'kbase'，'ksession'是经常使用的变量标识符,用 knowledge 的 k 开头。

例子 2.22 我们已统一了决策树如何加载，现在不再需要利用电子表格编译器预先生成一致的 DRL。

```
DecisionTableConfiguration dtconf =
KnowledgeBuilderFactory.newDecisionTableConfiguration();
dtconf.setInputType( DecisionTableInputType.XLS );
dtconf.setWorksheetName( "Tables_2" );
kbuilder.add( ResourceFactory.newUrlResource( "file://IntegrationExampleTest.xls" ),
        ResourceType.DTABLE,
        dtconf );
```

使用配置也可以配置一个 KnowledgeBase，通过一个 xml 更改集，而不是编程方式。

例子 2.23 一个简单的更改集例子

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
xs:schemaLocation='http://drools.org/drools-5.0/change-set change-set-5.0.xsd' >

  <add>
     <resource source='classpath:org/domain/someRules.drl' type='DRL' />
     <resource source='classpath:org/domain/aFlow.drf' type='DRF' />
  </add>

</change-set>
```

例子 2.24 可以象其他资源类型一样添加

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
        ResourceType.ChangeSet );
```

KnowledgeAgent 与 RuleAgent 相比较，其他较大的变化是轮询扫描现在是一个服务了。进一步说，是在代理通知与资源监视之间有一个抽象，允许使用其他轮询机制。

例子 2.25 这些服务当前不再默认启动，要启动它们要如下这样：

```
ResourceFactory.getResourceChangeNotifierService().start();
ResourceFactory.getResourceChangeScannerService().start();
```

增加了两个接口，ResourceChangeNotifier 和 ResourceChangeMonitor。KnowledgeAgent 使用 ResourceChangeNotifie 实现订购资源更改通知。通过添加 ResourceChangeMonitor 通知 ResourceChangeNotifie 资源改变了。我们现在只提供一个开箱既用的的监视器，ResourceChangeScannerService，它轮询资源的变化。然而，存在有 api 用于用户添加自己的监视器，所以可以使用一个如 jms 这样的基于推的监视器。

ResourceFactory.getResourceChangeNotifierService().addResourceChangeMonitor(my JmsMonitor);

2.2.2 Drools Guvnor

■·新外观的网页工具

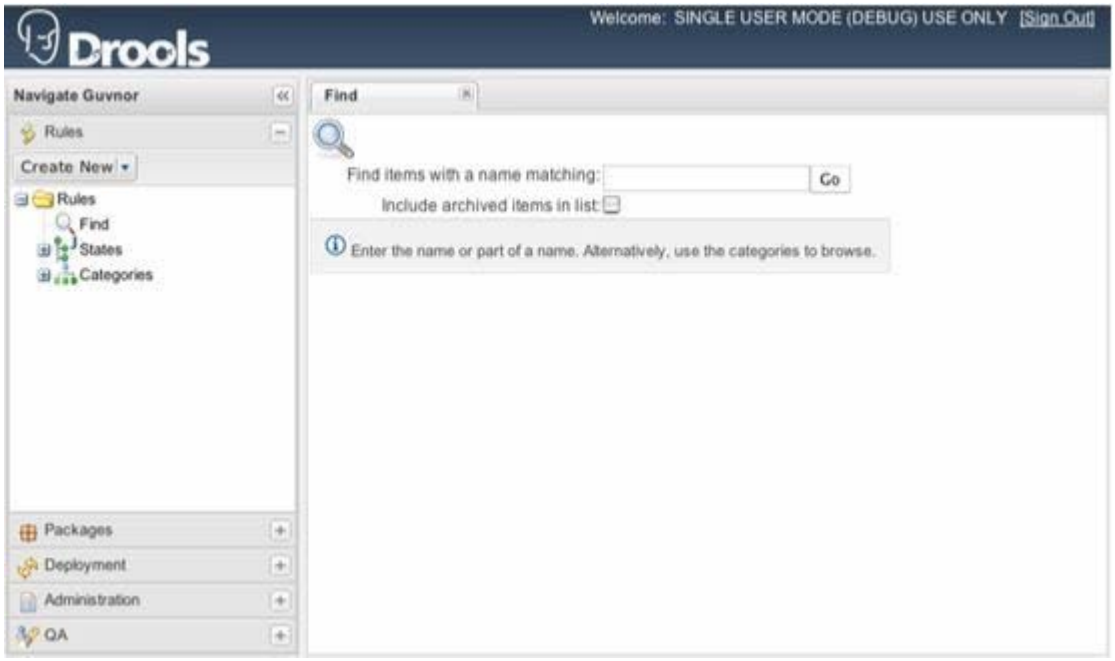


图 2.25 新外观

■·网页基于决策表编辑器

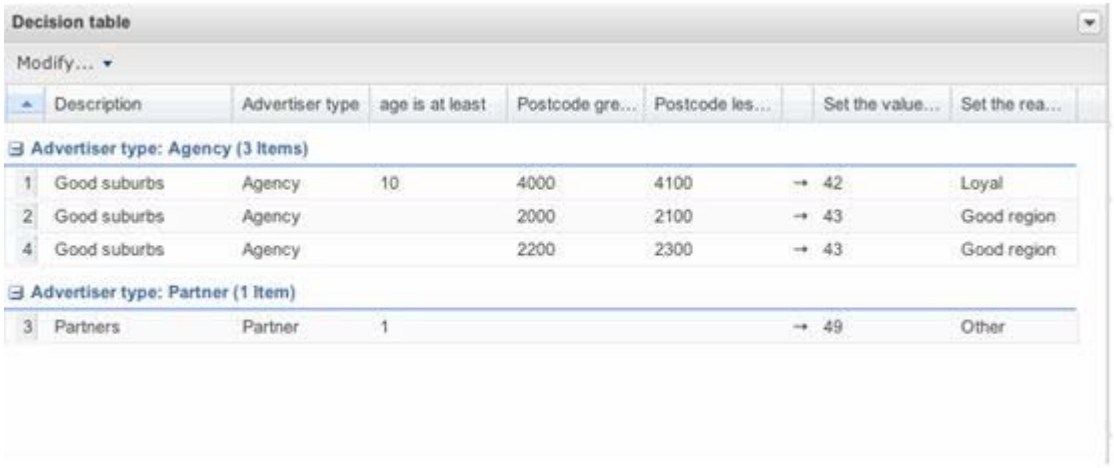


图 2.26 网页基于决策表编辑器

■·集成的场景测试

图 2.27 运行所有场景

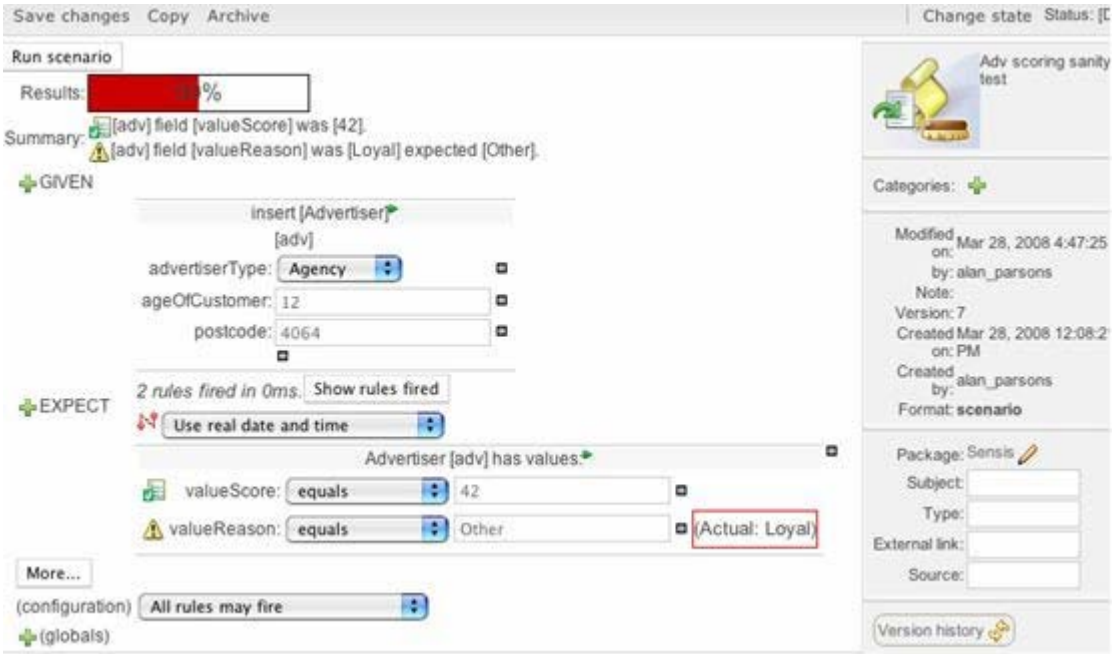


图 2.28 运行单个场景

■·WebDAV 文件基于接口库

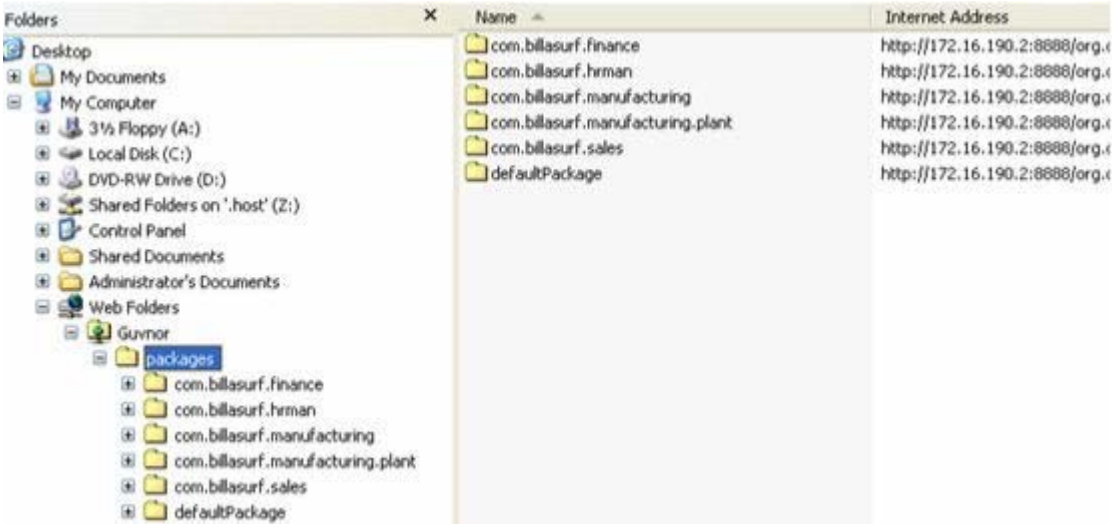


图 2.29 WebDAV

■·声明式模拟类型（不使用 pojos 的类型）

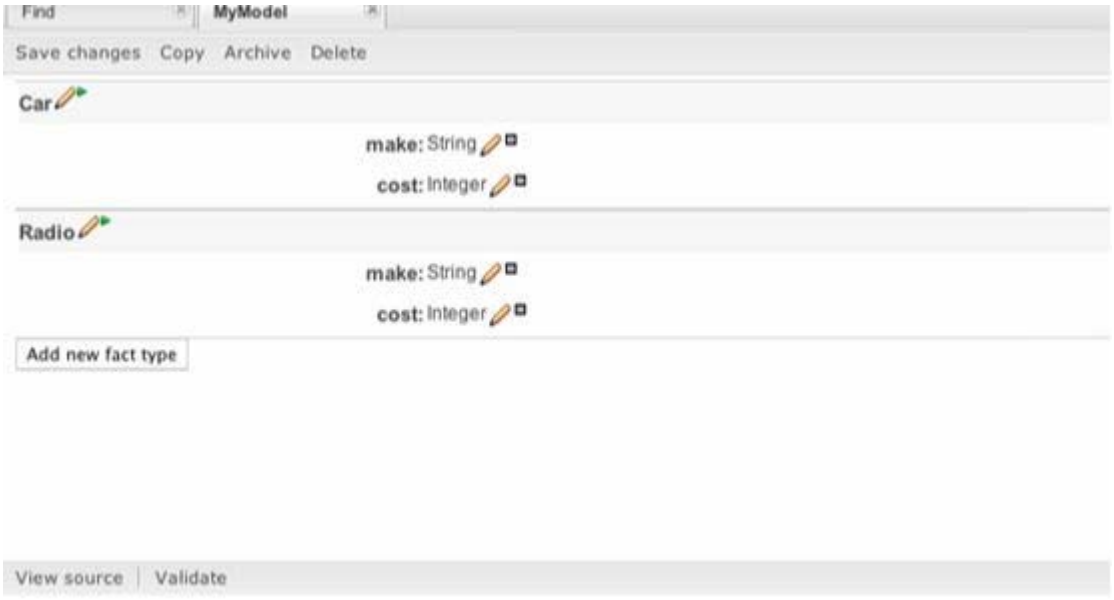


图 2.30 声明式模拟

它与"declare"语句一起工作——你现在可以在 drl 本身内声明类型。然后你可以装配它们，不必使用 pojo（如果你喜欢）。然后这些类型可用于规则库中。

- ◆·细粒度安全（锁定访问每包或每分类应用程序）。只有分类权限的用户具有有限的 UI 能力（针对企业用户）
- ◆·执行服务器——执行通过 XML 或 JSON 访问规则。
- ◆·分类规则允许你为一个分类设置“父规则”。出现在给定分类的任何规则会“继承”指定的规则——例如，继承条件/ LHS。分类的基本规则可以在包配置标签上设置。RHS 不被继承，只有 LHS 被继承。
- ◆·场景运行器检测无限循环。
- ◆·现在可以设置在指南编辑器中的 DSL 句子，用于显示作为一个下接菜单的枚举、作为一个数据拾取器的数据、作为一个单选框的布尔值，并且使用正则表达式来校验用户输入（在 Guvnor 中的 DSL 小饰件）。
- ◆·函数可以使用文本编辑器进行编辑。
- ◆·可以添加对象到全局集合。
- ◆·翻译为英语，西班牙语，汉语和日语。

2.2.3 Drools Expert

2.2.3.1 非对称 Rete 算法实现

不再需要影子代理。影子代理保护引擎免受有关实事的信息变化的影响，如果发生在引擎控件的外部，它可能不会被修改或撤消。

2.2.3.2 包构建器现在可以构建多个命名空间

你不再需要对一个包命名空间构建一个 PackageBuilder 。只要保持为所有命名空间增加你的 DRLs，并且 getPackages()为每个使用的命名空间返加一个包数组。

例子 2.26 获得多个包

```
Package[] packages = pkgBuilder.getPackages();
```

2.2.3.3 规则库连接包构建器

现在可以连接一个 RuleBase 到一个 PackageBuilder，这意味着规则被构建，并且同时被添加到规则库。PackageBuilder 使用现行的 RuleBase 的 Package 实例作为它的资源，取消了发生在现有方法中的 Package 创造和融合。

例子 2.27 连接规则库（RuleBase）到包构建器（PackageBuilder）

```
RuleBase ruleBase = RuleBaseFactory.newRuleBase();
PackageBuilder pkgBuilder = new PackageBuilder( ruleBase, null );
```

2.2.3.4 有状态会话的二进制编码

有状态会话现在可以保存，并可在以后的日期中恢复。预加载数据会话现在可以被创建。对用户对象的持久化可使用可插式策略，例如，hibernate 或特征（identity）映射。

2.2.3.5 类型声明

Drools 现在支持一种新的基础结构，称为类型声明。这个结构达到两个目的：能够声明实事元数据，以及能够为规则引擎动态地产生局部的新的事实类型。Guvnor 模拟工具在底层使用了它。下面是该结构的一个例子：

例子 2.28 声明 StockTick

```
declare StockTick
```

```
  @role( event )
  @timestamp( timestampAttr )

  companySymbol : String
  stockPrice : double
  timestampAttr : long
end
```

2.2.3.6 声明实事元数据

要声明和关联事实的元数据，只需要对你想声明的每个元数据 ID 使用@符号。例子：

例子 2.29 声明元数据

```
declare StockTick
  @role( event )
end
```

2.2.3.7 触发 Bean 产生

要激活动态 bean 产生，仅为你的类型声明添加字段和类型。

例子 2.30 声明 Person

```
declare Person
  name : String
```

```
  age : int
end
```

2.2.3.8 DSL 的改进

一系列 DSL 的改进被实现，包括一个完善的新解析器，并且能够为匹配的变量声明匹配的掩码（mask）。例如，它可限定一个电话号码字段为 2 位数的国家代码+ 3 位区号+ 8 位数字电话号码，所有连接以“ - ”（破折号）连接，通过象这样声明 DSL 映射：电话号码为{number:\d{2}-\d{3}-\d{8}}，所有有效的 Java 正则表达式可以用于变量的掩码中。

2.2.3.9 fireUntilHalt()

Drools 现在支持 fireUntilHalt()功能，它以一种被动模式启动引擎，在那儿规则会被连续引发，直到调用了 halt()。这尤其对 CEP（complex event processing）场景有用，CEP 场景需要俗称的“活动查询”。

2.2.3.10 规则库分区和多线程传播

Drools ReteOO 算法现在支持一个选项，用于以多线程模式启动规则库，在那儿 Drools ReteOO 网络被划分为多个部分，然后规则被多个线程并发计算。对通常有几个独立规则并发运行的 CEP，它也有一个要求，接近实时性能/吞吐量的要求，并且一个计算不能干扰其他的计算。

2.2.3.11 XSD 模式支持

Drools 现在支持 XSD 模式。记住虽然 XSD 模式以用于 Drools 类加载器的本地 POJO 类生成。在包构建器中存在有一个帮助类用于模式的产生。一旦数据模式被生成，你通常使用 JAXB 数据加载器插入数据。

2.2.3.12 数据加载器

Drools 现在支持两种数据加载器，Smooks 和 JAXB。Smooks 是一个用于 ETL 的开源数据转换工具，JAXB 是一个标准的 Sun 数据映射工具。单元测试显示 Smooks 和 JAXB 均可在这里找到。

2.2.3.13 类型安全配置

在 Drools 中，除了能够通过配置文件配置选项外，也可用系统属性配置，通过 API 的 setProperty（）方法设置属性，Drools-API 现在支持类型安全配置。我们不希望为每个可能的配置方法增加特殊的方法，有两个原因：它污染了 API，并且每次都有一个新选项增加到 Drools，API 将不得不改变。而这种方式，我们遵循模块化，类基于配置，在此处为每个可能的配置，一个新的选项类增加到了 API，除了灵活之外，也维持了 API 的稳定。所以，现在为了设置配置选项，你只需要使用枚举或者提供每个选项的工厂。例如，如果你希望为断言行为"equality" 配置知识库，并且自动从模式匹配中删除特征（identities），你只需要使用下面的枚举：

例子 2.31 配置

```
KnowledgeBaseConfiguration config =
KnowledgeBaseFactory.newKnowledgeBaseConfiguration();
config.setOption( AssertBehaviorOption.EQUALITY );
config.setOption( RemoveIdentitiesOption.YES );
```

对于选项，我们不需要预定义约束，或者可以假设多个值，提供一个工厂方法。例如，配置 alpha 值为 5，只使用 get()工厂方法：

例子 2.32 配置 alpha 值

```
config.setOption( AlphaThresholdOption.get(5) );
```

正如你所见，为每个不同的可能配置，使用了相同的 setOption()方法，然而它们仍然是类型安全的。

2.2.3.14 新的累积函数：collectSet 和 collectList

有时候，有必要收集来自实事属性的值的集合或列表，而不是实事本身。在这种情况下，不可能使用 collect CE。所以对这种情况，现在 Drools 有两个新的累积函数：collectSet 用于收集值的集合（即，无重复的值），collectList 用于收集值的列表（即，允许重复的值）：

例子 2.33 新的累积函数

```
# collect the set of unique names in the working memory
$names : Set() from accumulate( Person( $n : name, $s : surname ),
                    collectSet( $n + " " + $s ) )

# collect the list of alarm codes from the alarms in the working memory
$codes : List() from accumulate( Alarm( $c : code, $s : severity ),
                     collectList( $c + $s ) )
```

2.2.3.15 用于类型声明的新元数据：@propertyChangeSupport

事实实现了象定义在 Javabean(tm)规范中的属性改变的支持。现在可以注释，让引擎注册自身来侦听事实属性的变化。在 Drools 4 API 的 insert（）方法中使用的布尔参数已过时，并且不存在于 drools-aip 模块中了。

例子 2.34 @propertyChangeSupport

```
declare Person
    @propertyChangeSupport
end
```

2.2.3.16 批处理器

批处理器允许一个知识会话使用命令脚本，此外，无论是 StatelessKnowledgeSession 还是 StatefulKnowledgeSession 实现都可以使用 CommandFactory 创建这个接口命令，且使用"execute" 方法执行，如下所示：

例子 2.35 使用 CommandFactory

```
ksession.execute( CommandFactory.newInsert( person ) );
```

尽管这样，你通常会希望执行一个批处理命令，通过组合命令 BatchExecution 可以完成它。BatchExecutionResults 现在用来处理结果，某些命令可以使用"out"标识符，用它来添加结果到 BatchExecutionResult。现在可以轻松地执行查询，并把结果添加到 BatchExecutionResult。这个结果进一步被限定到这个执行调用，并且通过 BatchExecutionResults 返回。

例子 2.36 使用 BatchExecutionResult

```
List<Command> cmds = new ArrayList<Command>();
cmds.add( CommandFactory.newSetGlobal( "list1", new ArrayList(), true ) );
cmds.add( CommandFactory.newInsert( new Person( "jon", 102 ), "person" ) );
cmds.add( CommandFactory.newQuery( "Get People" "getPeople" );
```

```
BatchExecutionResults results =
ksession.execute( CommandFactory.newBatchExecution( cmds ) );

results.getValue( "list1" ); // returns the ArrayList

results.getValue( "person" ); // returns the inserted fact Person

results.getValue( "Get People" );// returns the query as a QueryResults instance.

End
```

CommandFactory 详细描述支持的命令，它们所有都可以使用 XStream 和 BatchExecutionHelper 进行编码。可以使用管道组合它们来自动化会话脚本。

例子 2.37 使用 PipelineFactory

```
Action executeResultHandler = PipelineFactory.newExecuteResultHandler();

Action assignResult = PipelineFactory.newAssignObjectAsResult();

assignResult.setReceiver( executeResultHandler );

Transformer outTransformer =
PipelineFactory.newXStreamToXmlTransformer( BatchExecutionHelper.newXStreamMar
shaller() );

outTransformer.setReceiver( assignResult );

KnowledgeRuntimeCommand batchExecution = PipelineFactory.newBatchExecutor();

batchExecution.setReceiver( outTransformer );

Transformer inTransformer =
PipelineFactory.newXStreamFromXmlTransformer( BatchExecutionHelper.newXStream
Marshaller() );

inTransformer.setReceiver( batchExecution );

Pipeline pipeline = PipelineFactory.newStatelessKnowledgeSessionPipeline( ksession );

pipeline.setReceiver( inTransformer );
```

对一个规则集使用上面所述的，会更新一个 Cheese 事实的价格，下面给定的 xml 会插入一个使用了输出标识符（out-identifier）的 Cheese 实例。

例子 2.38 更新 Cheese 实事

```
<batch-execution>
<insert out-identifier='outStilton'>
  <org.drools.Cheese>
    <type>stilton</type>
    <price>25</price>
    <oldPrice>0</oldPrice>
  </org.drools.Cheese>
</insert>
</batch-execution>
```

然后我们会得到 BatchExecutionResults：

例子 2.39 更新 Cheese 实事

```
<batch-execution-results>
 <result identifier='outStilton'>
   <org.drools.Cheese>
     <type>stilton</type>
     <oldPrice>0</oldPrice>
     <price>30</price>
   </org.drools.Cheese>
 </result>
</batch-execution-results>
```

2.2.3.17 编码

MarshallerFactory 被用来编码和解码 StatefulKnowledgeSessions。最简单的，它可以象下面这样使用：

例子 2.40 使用 MarshallerFactory

```
// ksession is the StatefulKnowledgeSession
// kbase is the KnowledgeBase
ByteArrayOutputStream baos = new ByteArrayOutputStream();
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase );
marshaller.marshall( baos, ksession );
baos.close();
```

然而，在处理引用的用户数据时，你需要更有弹性地使用编码。要达成它，我们有一个 ObjectMarshallingStrategy 接口。我们提供了两个实现，但是用户可以自己实现它。提供的两个是 IdentityMarshallingStrategy 和 SerializeMarshallingStrategy。默认为 SerializeMarshallingStrategy，如上例所示，它只在一个用户实例上调用 Serializable 或 Externalizable 方法。而 IdentityMarshallingStrategy 为每个用户对象创建了一个整数 id，并存储它们在一个映射中，该 id 被写入到该流中。在解码时，它只简单地查看 IdentityMarshallingStrategy 映射，取回该实例。这意味着，如果你使用 IdentityMarshallingStrategy，它对编码实例的生命周期是有状态的，并且会创建 ids，保持它企图编码的所有对象的引用。

例子 2.41 使用 IdentityMarshallingStrategy 编码

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase, new
ObjectMarshallingStrategy[] { MarshallerFactory.newIdentityMarshallingStrategy() } );
marshaller.marshall( baos, ksession );
baos.close();
```

为增加弹性，我们不能想当然地认为单策略更合适，所以我们增加了一个 ObjectMarshallingStrategyAcceptor 接口，每个 ObjectMarshallingStrategy 都有。编码器有一个策略链，并且当它企图读或写一个用户对象时，它遍历策略，询问它们是否承担负责编码用户对象。提供了一个实现为 ClassFilterAcceptor。它允许使用字符和通匹符匹配类名。默认为"*.*"，所以在上面使用的 IdentityMarshallingStrategy，它有一个默认的"*.*"接收器。然而，比方说，我们希望序列化所有类，一个给定的包除外，这种情况，我们会使用身份查询，如下所示：

例子 2.42 使用身份查询

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();

ObjectMarshallingStrategyAcceptor identityAceceptor =
MarshallerFactory.newClassFilterAcceptor( new String[] { "org.domain.pkg1.*" } );

ObjectMarshallingStrategy identityStratetgy =
MarshallerFactory.newIdentityMarshallingStrategy( identityAceceptor );

Marshaller marshaller = MarshallerFactory.newMarshaller( kbase, new
ObjectMarshallingStrategy[] { identityStratetgy,
MarshallerFactory.newSerializeMarshallingStrategy() } );

marshaller.marshall( baos, ksession );

baos.close();
```

2.2.3.18 知识代理

KnowlegeAgent 由 KnowlegeAgentFactory 创建。KnowlegeAgent 提供自动加载、缓存和重新加载资源，并且根据一个属性文件配置它。当 KnowlegeBase 使用的资源更改时，KnowlegeAgent 可以更新或重构 KnowlegeBase。通过给定工厂的配置确定 KnowlegeAgent 的策略，但通常使用基于标准轮询的拉策略。我们希望增加基于推的更新，并在将来的版本中重构它。下面的例子，构建了一个代理，它根据在路径字符串中指定的文件构建了一个新的 KnowledgeBase。它会每 30 秒拉这些文件，而不是更新存在

的一个，因为 "newInstance" 设置为了"true" （然而，目前只支持"true" 值，并且很难编码到引擎中）。

例子 2.43 构建一个代理

```
// Set the interval on the ResourceChangeScannerService if you are to use it and default of 60s is not desirable.
ResourceChangeScannerConfiguration sconf =
ResourceFactory.getResourceChangeScannerService().newResourceChangeScannerConfiguration();
sconf.setProperty( "drools.resource.scanner.interval",
                   "30" ); // set the disk scanning interval to 30s, default is 60s
ResourceFactory.getResourceChangeScannerService().configure( sconf );
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
KnowledgeAgentConfiguration aconf = KnowledgeAgentFactory.newKnowledgeAgentConfiguration();
aconf.setProperty( "drools.agent.scanDirectories",
                   "true" ); // we want to scan directories, not just files, turning this on turns on file
scanning
aconf.setProperty( "drools.agent.newInstance",
                   "true" ); // resource changes results in a new instance of the KnowledgeBase being built,
                             // this cannot currently be set to false for incremental building

KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "test agent", // the name of the agent
                                                                 kbase, // the KnowledgeBase to use, the
Agent will also monitor any exist knowledge definitions
                                                                 aconf );
kagent.applyChangeSet( ResourceFactory.newUrlResource( url ) ); // resource to the change-set xml for the
resources to add
```

KnowledgeAgents 可以使用一个空的 KnowledgeBase 或植入的一个。如果提供了一个植入的 KnowledgeBase，KnowledgeAgent 会遍历 KnowledgeBase，并订阅它发现的资源。虽然 KnowledgeBuilder 可以构建在一个目录中发现的所有资源，但信息会被 KnowledgeBuilder 丢弃，因些那些目录不会继续被扫描。只有作为 applyChangeSet(Resource)方法的一部分指定的目录才会被监控。

2.2.4 Drools Flow

Drools 4.0 有简单的 "RuleFlow"用于组织规则 。Drools 5.0 引入了一个强大（可扩大）的工作流引擎。它允许用户使用规则和流程（流程和规则的强力交互是可能的）指定他们的业务逻辑，并提供了一个统一的环境。

2.2.4.1 在一个特殊断点的流程实例视图

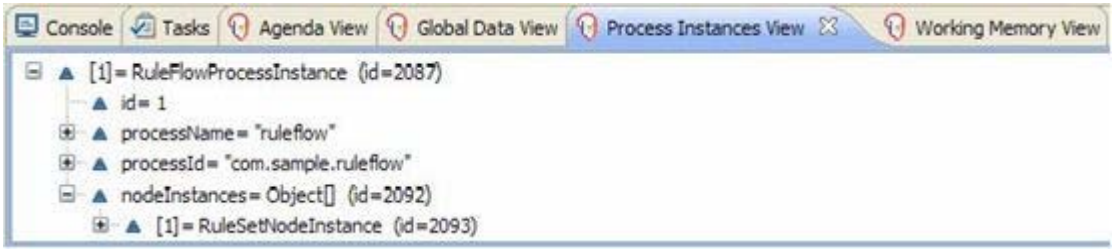


图 2.31 规则流属性

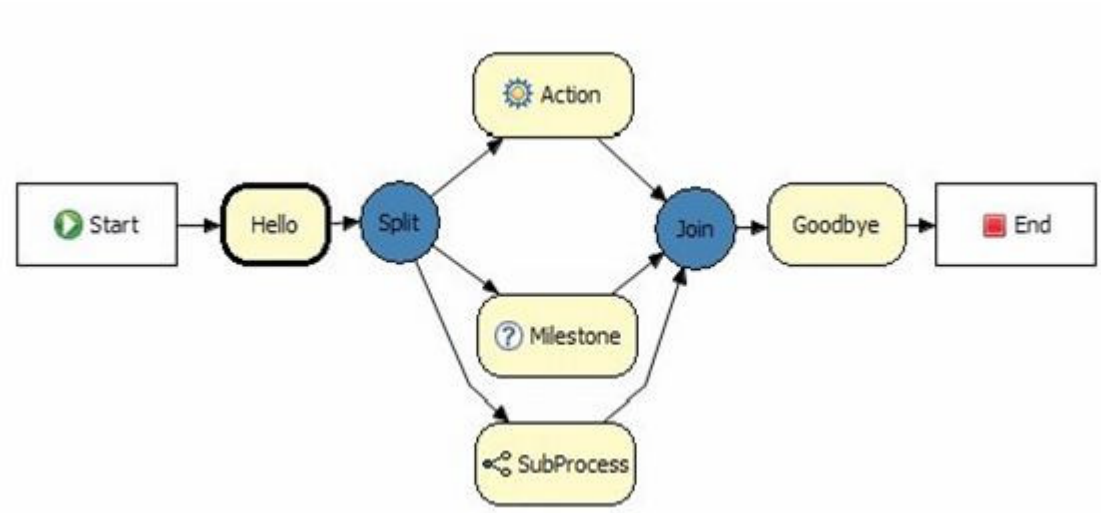


图 2.32 在一个工作流中的一个特殊断点中的当前活动节点

2.2.4.2 新节点

计时器：

可以增加一个计时器节点，它导致执行的节点等待一个特定时期。目前只使用 JDK 默认的初始延迟和重复延迟，更复杂的计时器可以会用于将来的里程碑中。

人类任务：

流程可以包括需要人类参与者来执行的任务。人类任务包括诸如任务名字、优先权、描述、参与者 id 等等参数。流程引擎可以使用我们的可插式工作项目（见后）方便地与存在的人类任务组件整合（例如，一个 WS-HumanTask 实现）。泳道和分配 （Swimlanes and assignment ）规则也被支持。

在调色板的截图中显示了两个新组件，并且工作流本身显示了使用的人类任务。也显示了两个"work items"，在下一节会被解释。

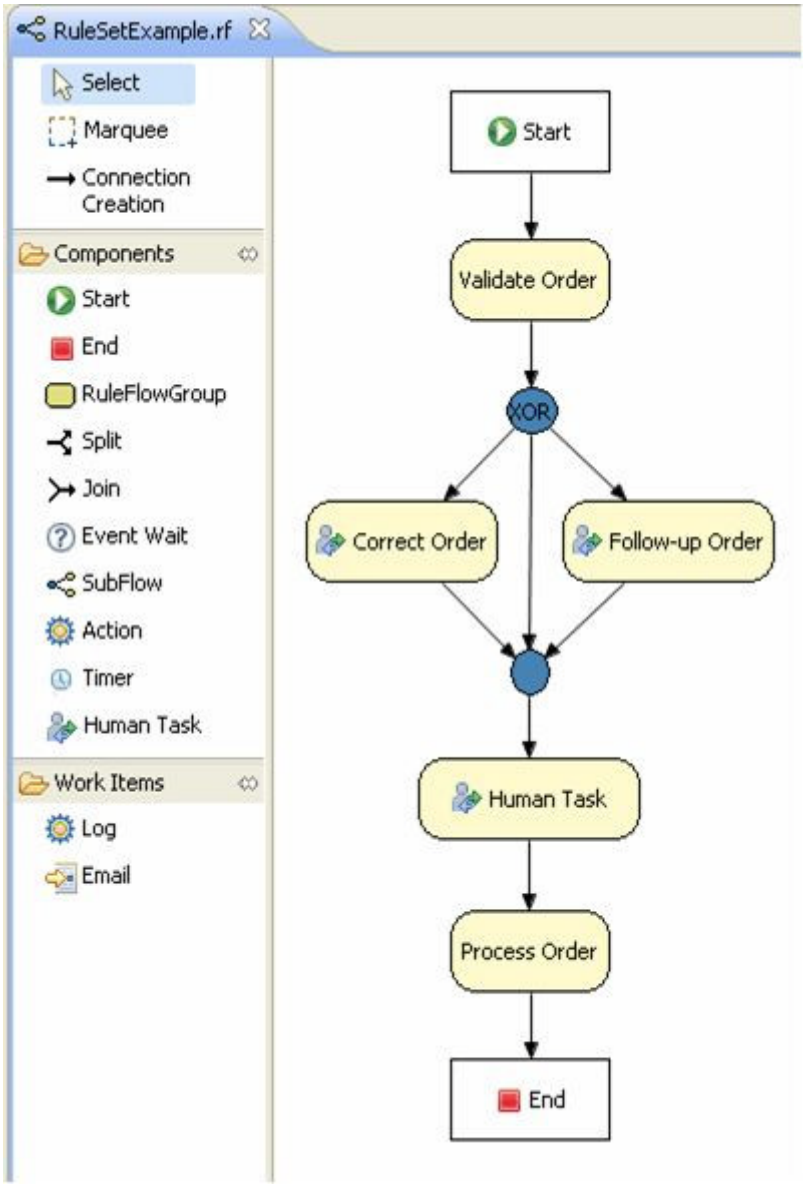


图 2.33 人类任务

2.2.4.3 域特殊工作项目

域特殊工作项目是用户创建的有助于定制任务执行的可插式节点。它们提供了一个 api 用于在调色板中指定一个新的图标，以及用于该任务属性的 gui 编辑器。如果没有提供 gui 编辑器，那么则默认为基于键值对表单的一个文本。然后 api 允许执行那些指定的工作项目的行为。默认提供了 Email 和 Log 工作项目。有关如何实现它们，在 Drools flow 中已被更新。

下图显示了用于一个工作流中的三个不同的工作项目，"Blood Pressure", "BP Medication", "Notify GP"：

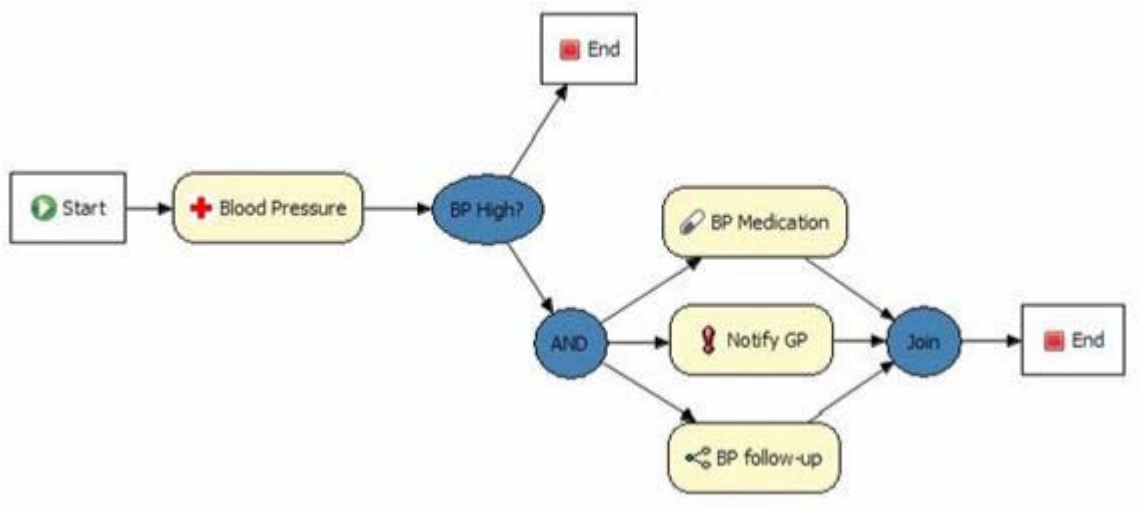


图 2.34 工作项目

还有一个新的"Notification"工作项目

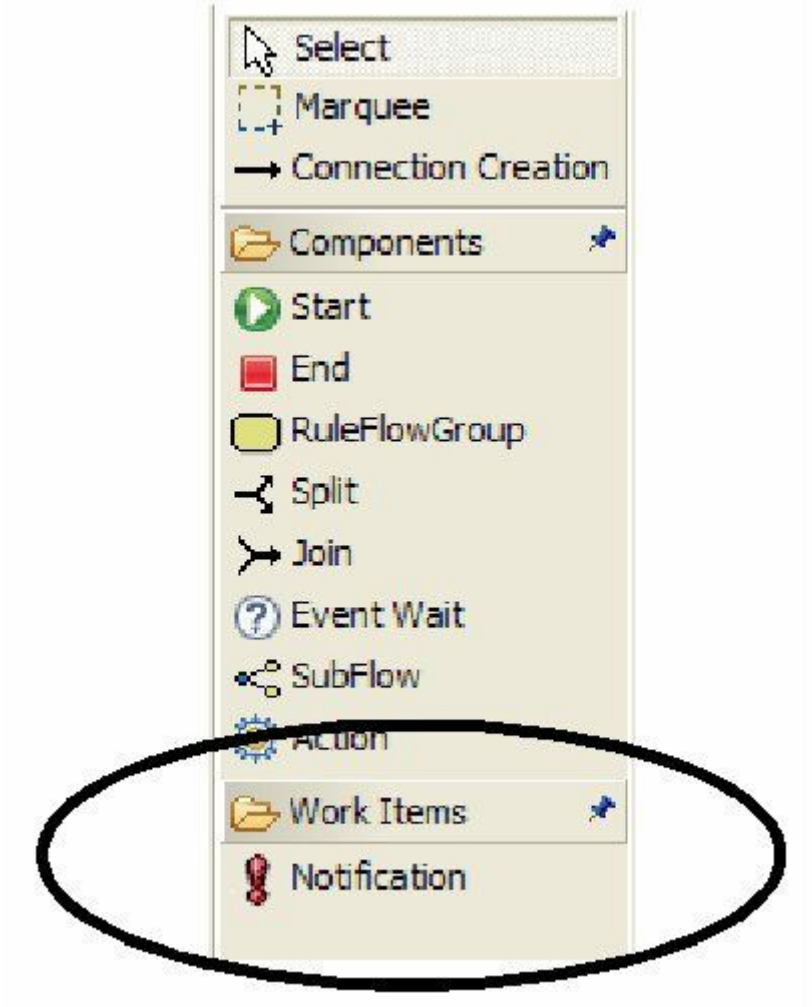


图 2.35 Notification

2.2.4.4 可扩展流程定义语言（ePDL）

Drools 4.0 使用 XStream 存储它的内容，这是不容易人为编写的。Drools 5.0 引入了 ePDL，它是一个用于我们流程语言的特殊 XML，它也允许域特殊扩展，在博客帖子“Drools 的可扩展流程定义语言（ePDL）和语义模块框架（SMF）”中已被详细谈及。如下所示，XML 语言的一个例子，使用了 DSL 扩展。

```
<process  name="process name" id="process name" package-name="org.domain"
 xmlns="http://drools.org/drools-4.0/process"
 xmlns:mydsl="http://domain/org/mydsl"
 xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
 xs:schemaLocation="http://drools.org/drools-4.0/process drools-processes-4.0.xsd" >
```

```
 <nodes>
   <start id="0" />

  <action id="1" dialect="java">
       list.add( "action node was here" );
   </action>

    <mydsl:logger id="2" type="warn">
         This is my message
     <mydsl:logger>

  <end id="3" />
 </nodes>

<connections>
   <connection from="0 to="1" />
   <connection from="1" to="2" />
   <connection from="2" to="3" />
 </connections>

</process>
```

2.2.4.5 可插式节点

用于该框架的基本节点完全是可插式的，使它易于扩展和实现其他执行模式。我们已经有一个 OSWorkflow 的局部实现，并正和 Deigo 一起完成它，为 OSWorkflow 用户提供一个移值路径。其他的增强，包括异常域、在各种节点上包括进入和退出（on-entry and on-exit）行为的能力、集成了我们的用于长期运行流程的状态持久化的二进制持久化机制等等。了解更多内容请看 Drools flow 文档。

2.2.4.6 人类任务

在流程的上下文中人类任务的管理是十分重要的。因为我们允许用户插入他们喜欢的任何任务组件，所以我们开发了一个人类任务管理组件，用于支持基于 WS - HumanTask 规范的人类任务的整个生命周期。

2.2.4.7 Drools flow 语言的新功能

- 事件节点允许一个流程响应一个外部事件。
- 异常处理器和异常处理器域用于处理可能被抛出的异常。
- ForEach 节点允许你多次实例化你的流程段，为一个集合中的每个元素。
- 数据类型的支持已得到扩展。
- 普通的节点类型集成了定时器。

因此，新的节点类型和属性已被增加到了 Eclipse 中的 Drools Flow 编辑器。在集成测试中，你可以发现这些新功能的例子（例如 ProcessExceptionHandlerTest，ProcessTimerTest 等）。

2.2.4.8 工作项目

我们的可插式工作项目方法，允许你以一个声明的方式，把域特殊（domain-specific ）工作插入到你的流程中。我们计划构建一个普通工作项目的库，并且已经提供了用于发送电子邮件、查找文件、归档、执行系统命令、日志和人类任务的实现。

2.2.4.9 JPA

改进了对持久化（JPA）和事务（JTA）的支持。

例子 2..45 如何使用持久化和事件与流程结合的例子

```
// create a new JPA-based session and specify the JPA entity manager factory
Environment env = KnowledgeBaseFactory.newEnvironment();
env.set( EnvironmentName.ENTITY_MANAGER_FACTORY, Persistence.createEntityManagerFactory( "emf-name" ) );
env.set( EnvironmentName.TRANSACTION_MANAGER, TransactionManagerServices.getTransactionManager() );

StatefulKnowledgeSession ksession = JPAKnowledgeService.newStatefulKnowledgeSession( kbase, null, env );
// KnowledgeSessionConfiguration may be null, and a default will be used
int sessionId = ksession.getId();

// if no transaction boundary is specified, the method invocation is executed in a new transaction
automatically
ProcessInstance processInstance = ksession.startProcess( "org.drools.test.TestProcess" );

// the users can also specify the transaction boundary themselves
UserTransaction ut = (UserTransaction) new InitialContext().lookup( "java:comp/UserTransaction" );
ut.begin();
ksession.insert( new Person( "John Doe" ) );
ksession.startProcess( "org.drools.test.TestProcess" );
ksession.fireAllRules();
ut.commit();
```

2.2.4.10 变量注入

支持在代码约束和动作中，在 MVEL 和 Java 中，直接访问流程变量，所以，在你的流程中有一个变量名为"person"，你现在可以如下这样描述约束：

例子 2.46 变量注入例子

* [Java code constraint] return person.getAge() > 20;
* [MVEL action] System.out.println(person.name);

2.2.4.11 杂项改进

■·流程实例现在可以通过标记事件节点属性"external" 为 ture,侦听外部事件。外部事件使用 session.signalEvent(type, eventData)单独发送给引擎。在你的流程中如何使

用事件，详情请看 Drools Flow：
https://hudson.jboss.org/hudson/job/drools/lastSuccessfulBuild/artifact/trunk/target/docs/drools-flow/html/ch03.html#d0e917

■·流程实例对多线程是安全的（因为多线程被阻止在相同的流程实例上工作）。

■·流程持久化/事务的支持已被进一步改进。详情请查看 drools-process/drools-process-enterprise 项目

■·人类任务组件已被扩展来支持在任务执行期间的输入/输出/异常的各种数据。

例子 2.47 因此，任务客户端的生命周期的方法已经扩展为允许内容的数据

```
taskClient.addTask(task, contentData, responseHandler)
taskClient.complete(taskId, userId, outputData,responseHandler)
taskFail.complete(taskId, userId, outputData,responseHandler)

long contentId = task.getTaskData().getDocumentContentId();
taskClient.getContent(contentId, responseHandler);
ContentData content = responseHandler.getContent();
```

■·现在在编辑期间，可以移植老的 Drools4 RuleFlows (使用 xstream 格式的) 到 Drools5 流程 (使用可读式的 xml)。当以下系统属性设置：drools.ruleflow.port = ture 时，在添加 RuleFlow 到 KnowledgeBase 时，移植会自动被执行。

■·增加了一个新类型的结合点（join）,它将等待直到它的 n 中的 m 个连接已被完成。这个 n 既可以是在流程中的硬编码，也可以是基于该过程中的一个变量的值。

■·已做了改进，使持久化配置更容易。持久化方法是基于一个命令服务，确保了客户端的调用在一个事务内部被执行，并且在命令成功执行之后状态被存储在数据库中。虽然在 M4 版中已经可以直接使用该命令，我们还是扩展了它，目的在于除了可以简单地使用配置文件配置持久化外，人们可以使用标准的 StatefulKnowledgeSession 接口。详情请查看 Drools Flow 文档中有关持久化的部分。

2.2.5 Drools Fusion

通过支持大量的 CEP（complex event processing）功能，以及支持事件在规则引擎中作为一流公民，Drools 5.0 带给了规则世界超强的事件处理。

2.2.5.1 事件语义

事件是（从规则引擎的视角来看）特殊类型的事实，具有很多的特色：

- 它们是不变的。
- 它们有强壮的时间相关关系。
- 它们必须有明确的生命周期窗口。
- 在它们的生命周期窗口过期后，它们必须被透明地回收垃圾。
- 它们必须是时间限制的。
- 它们必须被包括在滑动窗口中用于推理。

2.2.5.2 事件声明

通过简单地为实事声明元数据，任何事实可以假定一个事件角色，和它相应的事件语义。

例子 2.48 存在的和生成的 beans 都支持事件语义：

```
# existing bean assuming an event role
import org.drools.test.StockTick
declare StockTick
  @role( event )
end

# generated bean assuming an event role
declare Alarm
```

```
  @role( event )
  type : String
  timestamp : long
end
```

2.2.5.3 入口点的流侦听器

一个新的关键字"from entry-point"已被增添，允许在一个规则中的一个模式侦听一个流，这避免了必须把对象插入到在工作内存中的开销，依据所有规则它可能是完全合理的。

$st : StockTick(company == "ACME", price > 10) from entry-point "stock stream"

例子 2.49 插入一个实事进入一个入口点。

```
WorkingMemoryEntryPoint entry = wm.getWorkingMemoryEntryPoint( "stock stream" );
entry.insert( ticker );
```

StreamTest 为此显示了一个单元。

2.2.5.4 事件相关性和新的运算符

事件相关性和基于时间的约束的支持是事件处理需要的，并且被 Drools 5.0 完全支持。新的，开箱即用的，时间约束运算符可以在这些测试案例规则：test_CEP_TimeRelationalOperators.drl 中见到。

如你在上面的测试中所见， Drools 支持两个事件：原始事件，它是指没有期限的及时事件；组合事件，它是带有不同的开始和结束时间戳的事件。

运算符的完整列表如下：

- coincides
- before
- after
- meets
- metby
- overlaps
- overlappedby
- during
- includes
- starts
- startedby
- finishes
- finishedby

2.2.5.5 滑动时间窗

Drools 5.0 增加了支持，用于整个事件滑动窗口的推理。例如，

StockTick(symbol == "RHAT") over window:time(60)

上面的例子只会模式匹配发生在最后 60 秒的 RHAT 股票记号，抛弃任何大于它的事件。

2.2.5.6 会话时钟

启用全事件处理能力需要能够配置一个会话时钟，并与其交互。Drools 增加了时间推理和会话时钟配置的支持，允许它不仅运行实时事件处理，而且也可通过重演一个场景模拟 what-if 场景和 post-processing 场景审核。

例子 2.50 时钟被指定为 SessionConfiguration 的一部分，在会话创建期间的一个可选指定

新类

```
SessionConfiguration conf = new SessionConfiguration();
```

```
conf.setClockType( ClockType.PSEUDO_CLOCK );
StatefulSession session = ruleBase.newStatefulSession( conf );
```

2.2.5.7 事件垃圾收集

因为事件通常有强壮的时间关系，当事件能够匹配时，推断一个逻辑时间窗口是可能的。引擎使用它的计算能力，当事件不再能够匹配任何规则时，会自动收回那个事件。

2.2.5.8 时间单元支持

Drools 采用了一个简单的语法用于时间单元，基于 ISO 8601 期限语法。它允许用户轻松地增加时间约束到编写时间在知名单元中的规则。例如：

```
SomeEvent( this after[1m,1h30m] $anotherEvent )
```

如果 SomeEvent 发生在 $anotherEvent 之后的 1 分钟到 1 小时 30 分钟之间，上面的模式会匹配。

2.2.5.9 支持事件到期策略

增加了定义每类型到限策略的能力。在下面例子中，在它们进入系统后，StockTick 事件会到期 10 分钟：

```
declare StockTick @role( event ) @expires( 10m ) end
```

2.2.5.10 支持对任意日期的时间操作

例子 2.51 增加了点及时（point-in-tim）运算符（前，后，和相等），用于任意日期字段。

```
rule "Allow access"
when
WorkHours( $s : start, $e : end )
LogIn( time after $s, time before $e )
then
// allow access
end
```

2.2.6 Eclipse IDE

■·支持多种运行时间：现在 IDE 支持多种运行时间。一个 Drools 运行时间就是在你的文件系统中的一个 jars 的集合，代表一个特定的 Drools 项目 jars 版本。为创建一个运行时间，你必须把 IDE 指向你选择的版本，或者在你的文件系统中，根据包含在 Drools Eclipse 插件中的 jars 简单地创建一个新的运行时间。如下所示，你可以通过打开 Eclipse preferences 并选择 Drools -> Installed Drools Runtimes 配置 Drools 运行时间。

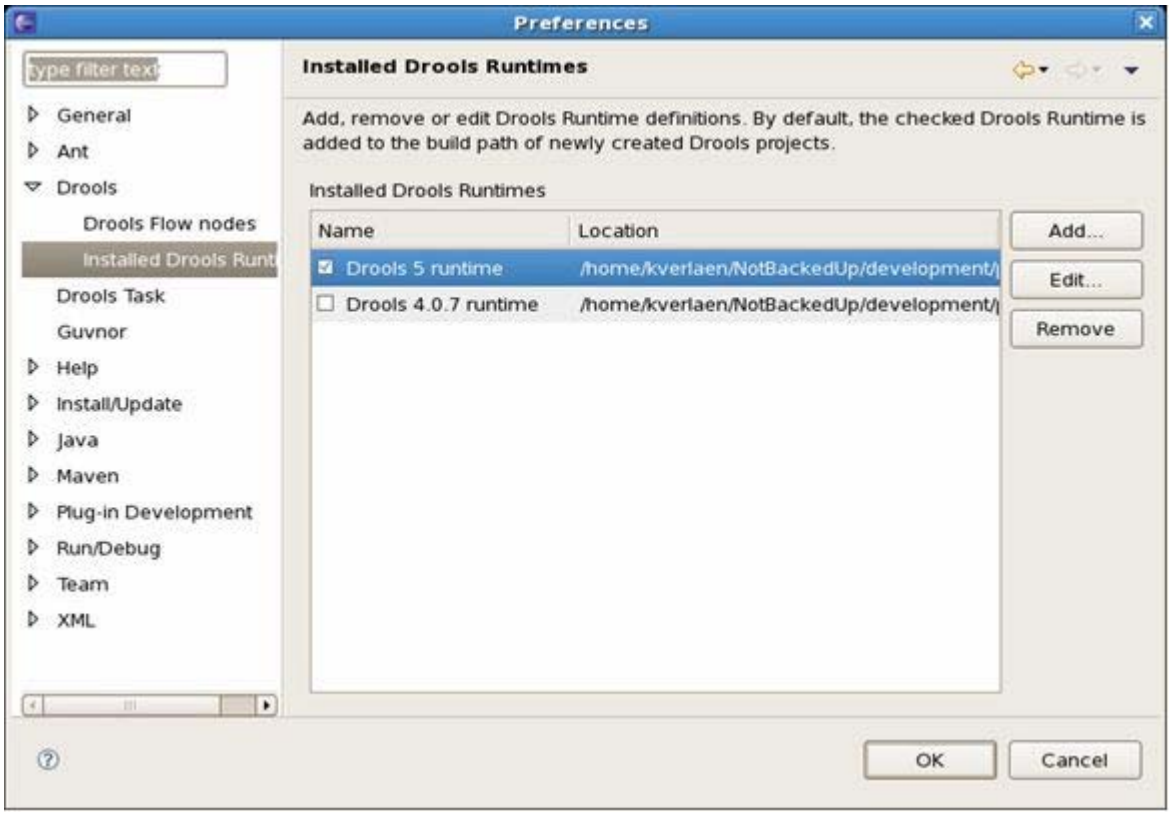


图 2.36 运行时间。

■·使用 mvel 方言的规则的调试已经被修补。

■·Drools 流编辑器

◆·流程皮肤允许你定义不同的规则流可视化节点。我们现在支持两种皮肤：一个是以前存在的默认皮肤；一个是 BPMN 皮肤，使用类似 BPMN 的可视化节点表示：http://blog.athico.com/2008/10/drools-flow-and-bpmn.html

◆·(X)OR 分割现在以连接标签显示约束的名字

◆·定制工作项目编辑器已经被改变，现在能正确地发信号通知流程。

2.3 Drools 4.0 中的新东西

Drools 4.0 在 Drools 3.0.x 系列基础上有重大更新。被开发的一个全新的功能集，特别集中在语言表达能力、引擎性能、工具的可用性上。最令人感兴趣的更改清单如下所示。

2.3.1 语言表达能力的改进

■·新的条件元素：from, collect, accumulate, forall

■·新字段的约束运算符：not matches, not contains, in, not in, memberOf, not memberOf

■·新的隐式自引用字段：this

■·完全支持条件元素嵌套、一阶逻辑的完备性。

■·支持多限制和约束连接&&和||。

■·删除了以前的语言限制，如字符转义和关键字冲突。

■·支持可插式方言，并且完全支持 MVEL 脚本语言（http://mvel.codehaus.org/）

■·完全重写了 DSL，允许充分地本地化。

■·事实属性 auto-vivification，用于返回的值限制和内联求值约束。

■·支持嵌套访问器、属性导航、以及简单集合、数组和映射语法。

■·改进了 XML 规则的支持。

2.3.2 核心引擎性能的改进

■·本地支持基本类型，避免了不断自动装箱。

■·支持可选的透明影子事实。

■·用于复杂规则的 Rete 网络性能的改进。

■·支持规则流。

■·支持有状态和无状态的工作内存（规则引擎会话）。

■·支持异步工作内存动作。

■·规则引擎代理，用于热布置和 BRMS 整合。

■·动态特点，用于规则冲突解决方案。

■·支持参数查询。

■·支持挂起命令。

■·支持顺序执行模式。

■·支持可插式全局变量解析器。

2.3.3 IDE 的改进

■·支持规则断点调试。

■·可见即可视支持规则流。

■·用于规则创作的新的指南编辑器。

■·升级到支持所有新引擎功能。

2.3.4 业务规则管理系统——BRMS

■·新的 BRMS 工具。

■·带有优雅的 WEB 2.0 ajax 功能的友好用户网页界面。

■·包的配置。

■·规则创作，使用指南编辑器（下拉菜单）和文本编辑易于编辑规则。

■·包的编译和部署。

■·使用规则代理轻松部署。

■·易于组织分类和搜索资源。

■·启用版本化，你可以轻松地使用前面保存的替换你的资源。

■·JCR（Java Content Repository）兼容规则资源库。

2.3.5 杂项改进

■·精简依赖关系，减少了内存占用。

2.4 从 Drools 3.0.x 升级到 Drools 4.0.x 的技巧

如前所述，Drools 4.0 在 Drools 3.0.x 系列基础上有重大更新。不幸的是，为了实现这个版本既定目标，引入了某些向后兼容性的问题，如在邮件列表和博客中讨论的一样。

本手册这部分是一个进行中的工作，并将简单地描述如何从 Drools 3.0.x 升级到 Drools 4.0.x。

2.4.1 API 的变化

对普通用户而言，有几个 API 变化是可见的，并且需要被修正。

2.4.1.1 工作内存的创建

Drools 3.0.x 只有一个工作内存类型，其工作如一个有状态工作内存。Drools 4.0.x 引入了一个单独的 APIs 用于有状态和无状态工作内存，其现在被称为规则会话。在 Drools 3.0.x 中，如下所示创建一个工作内存：

例子 2.52 在 Drools 3.0.x 中，工作内存的创建。

```
WorkingMemory wm = rulebase.newWorkingMemory();
```

在 Drools 4.0.x 中必须改为：

例子 2.53 在 Drools 4.0.x 中，工作内存的创建。

```
StatefulSession wm = rulebase.newStatefulSession();
```

StatefulSession 对象具有 Drools 3.0.x 的工作内存一样的行为（继承了 WorkingMemory 内存接口），所以使用这个修正应该没有什么问题。

2.4.1.2 工作内存动作

Drools 4.0.x 现在支持可插式方言，并且内置了对 Java 和 MVEL 脚本的支持。为了避免关键字冲突，工作内存动作被重新命名，如下所示：

表 2.1 工作内存动作等价的 API 方法

Drools 3.0.x	Drools 4.0.x
WorkingMemory.assertObject()	WorkingMemory.insert()
WorkingMemory.assertLogicalObject()	WorkingMemory.insertLogical()

WorkingMemory.modifyObject()	WorkingMemory.update()

2.4.2 规则语言的变化

DRL 规则语言也有一些向后兼容的变化，在下面详述。

2.4.2.1 工作内存动作

在规则结果中的工作内存动作也用相似于 API 中所做变化的方法被改变。汇总的变化如下表所示：

表 2.2 工作内存动作等价的 DRL 命令

Drools 3.0.x	Drools 4.0.x
assert()	insert()
assertLogical()	insertLogical()
modify()	update()

2.4.2.2 基本支持和拆箱

Drools 3.0.x 没有对基本类型的本地支持，因此，它自动装箱所有基本类型在它各自的包裹器类中。这种方式，任何装箱变量的绑定使用需要一个手动拆箱。

Drools 4.0.x 已完全支持基本类型，再也不用包裹值。所以，所有原先的 DRL 拆包方法调用必须被删除。

例子 2.54 Drools 3.0.x 手动拆包

```
rule "Primitive int manual unbox"
when
    $c : Cheese( $price : price )
then
    $c.setPrice( $price.intValue() * 2 )
end
```

上面的规则在 Drools 4.0.x 中则是：

例子 2.55 Drools 4.0.x 的基本支持

```
rule "Primitive support"
when
    $c : Cheese( $price : price )
then
    $c.setPrice( $price * 2 )
end
```

2.4.3 Drools 更新工具

Drools 更新工具是一个简单的程序，用于升级 DRL 文件，从 Drools 3.0.x 到 Drools 4.0.x。

在这点上，主要目标是是升级内存动作的调用，从 Drools 3.0.x 到 Drools 4.0.x，但是希望它在未来几周成长中覆盖其他情况。注意它没有在规则文件中做一个哑文本搜索和替换是重要的，但是它的确解析了规则文件，并且尝试确保它不做任何不希望的事情。因此所以，它是一个用于更新大集合的规则文件的安全工具。

Drools 更新工具作为一个 maven 项目可以在下面的资源库找到：http://anonsvn.labs.jboss.com/labs/jbossrules/trunk/experimental/drools-update/ 。你只需要找到它，使用项目的 pom.xml 文件执行 maven clean install 动作即可。然后，使用下面命令，解决你可能需要付出代价的所有类路径的依赖项：

```
java -cp $CLASSPATH org.drools.tools.update.UpdateTool -f <filemask> [-d <basedir>] [-s <sufix>]
```

象下面这样，程序的参数是非常容易理解的。

- -h ——帮助，显示一个非常简单的帮助使用列表。
- -d ——你的源库目录
- -f —— 文件被更新的模式。格式与 ANT 中使用的相同：* = 单个文件、目录，** =任何子目录级。例子：src/main/resources/**/*.drl = 匹配在/src/main/resources 中的任何子目录内的所有 DRL 文件。
- -s ——后缀，被增加到所有更新文件的后缀。

2.4.4 在 Drools 4.0 中的 DSL 语法

要重点注意的是，DSL 模板引擎已被完全重写，提高了灵活性。DSL 语法的一个新功能之一是支持正则表达式。这种方式，你现在可以使用 regexp 重写你的映射，而得到更多的灵活性，如在 DSL 章节解释的一样。尽管，现在你必然使用 regexp 含义的转义字符。例如，如果前面你有一个匹配，如下：

例子 2.56 Drools 3.0.x 的映射

```
[when][]- the {attr} is in [ {values} ]={attr} in ( {values} )
```

现在，你需要转义'['和']'字符，因为它们在 regexp 有特别的含义。所以，在 Drools 4.0.x 的映射将是：

例子 2.57 使用了转义实际行动的 Drools 4.0.x 的映射

```
[when][]- the {attr} is in \[ {values} \]={attr} in ( {values} )
```

2.4.5　4.0.2 版的规则流更新

4.0.2 版的规则流功能已被更新，现在你的所有规则流必须声明一个包名字。

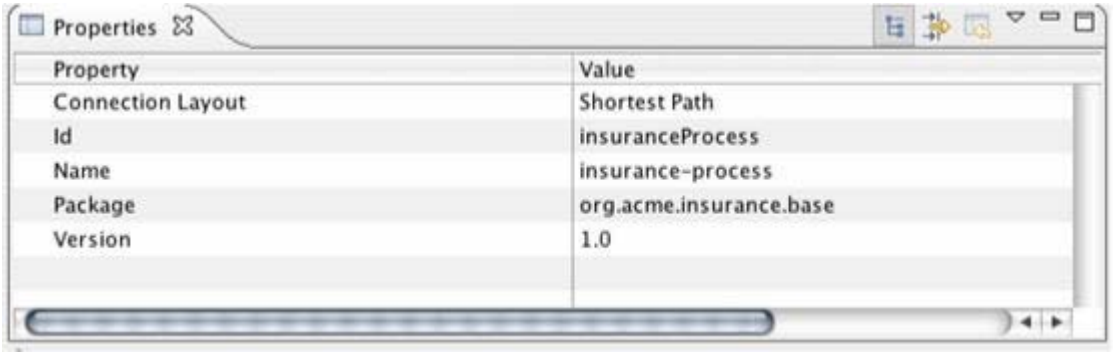


图 2.37 规则流属性

第 3 章　安装和设置（核心和 IDE）

3.1　安装和使用

Drools 提供了一个基于 Eclipse 的 IDE（可选的），但是在它的核心只需要 Java 1.5（Java SE）。

一个简单的入门方法是下载和安装 Eclipse 插件——这也需要安装 Eclipse GEF 框架（见后，如果你以前没有安装它）。它会提供给你继续下去需要的所有依赖项：你可以简单地创建一个新的规则项目，并且所有事件会帮你做好。有关它的详情，请看“规则工作台和 IDE”章节。安装 Eclipse 插件通常与你解压一个文件到你 Eclipse 插件目录一样简单。

Eclipse 插件的使用并不是必需的。规则文件只是一个文本输入（或视情况而定的电子表格），而 IDE（也称为规则工作台）仅仅是为了方便。人们用很多方法集成规则引擎，并没有“一刀切”。

另外，你可以下载二进制发行版，并在你的项目类路径中包含相关的 jars。

3.1.1 依赖项和 jars

Drools 被分成了几个模块，某些在规则部署/编译期间需要，某些在运行时需要。在很多情况下，人们只是想在运行时包含所有的依赖项，这是好事。这让你有更多的灵活性。然而，有些人可能希望让他们的“运行时间”精简到最低限度，因为他们会以二进制形式部署规则——这也是可能的。核心运行时间引擎可以是十分简洁的，整个 2 个 jar 文件只需要几百 KB。

下面是构成 JBoss 规则的重要库的一个描述：

- drools-api.jar —— 提供接口和工厂。Drools-api 还有助于清楚地显示作为一个用户 api 的目的是什么，以及只作为一个引擎 api 的目的是什么。
- drools-core.jar —— 这是核心引擎，运行时间组件。包含了 RETE 引擎和 LEAPS 引擎。如果你是预编译的规则，则它只是运行时间依赖项（部署通过 Package 或 RuleBase 对象）。
- drools-compiler.jar ——它包含编译器/构建器组件，用于获取规则资源，以及构建可执行规则库。它通常是你的应用程序的一个运行时间依赖项，但是如果你是预编译了你的规则，则不需要它。这取决于 drools-core。
- drools-jsr94.jar —— 它是 JSR-94 兼容实现，它实际上是在 drools-compiler 组件上的一个分层。 注意，由于 JSR-94 规范的本性，不是所有功能都轻易地通过个接口暴露。在某些情况下，直接到 Drools API 会是更容易的，但是在某些环境下，要求 JSR-94 。
- drools-decisiontables.jar ——这是决策表“编译”组件，其使用了 drools-compiler 组件。它支持 excel 和 CSV 输入格式。

还有不少上面组件需要的其他依赖项，它们大多数用于 drools-compiler, drools-jsr94 或 drools-decisiontables 模块。某些关键的要注意，"POI" 提供电子表格解析能力，而 "antlr"提供规则语言自身的解析。

注意：如果你在 j2ee 或 servlet 容器中使用 Drools，你使用"JDT"会遭遇到类路径问题，那么你可以切换为 janino 编译器。设置系统属性"drools.compiler"：例如，
-Ddrools.compiler=JANINO

对于在一个发行版中的有关依赖项的最新日期信息，请参考 README_DEPENDENCIES.txt 文件，它可以在下载包的 lib 目录中或在项目的根目录中找到。

3.1.2 Runtime

如果你以二进制格式（以 KnowledgePackage 对象或 KnowledgeBase 对象等等）部署规则，需要提及"runtime"。这是一个可选的功能，让你保持你的运行时间很轻巧。你可以使用 drools-compiler 生产“在流程之外（"out of process"）”的规则包，然后部署它到一个运行时间系统。这个运行时间系统只需要用于执行的 drools-core.jar 和 drools-api 。这是一个可选的部署模式，而大部分人不需要这么多地“裁减”他们的应用程序，然而它是特定环境下的一个理想选项。

3.1.3 安装 IDE（规则工作台）

规则工作台(用于 Eclipse)要求 Eclipse 3.4 以上版本，以及 Eclipse GEF 3.4 以上版本。你可以通过下载插件或使用升级站点安装它。

另外的选择是使用期 JBoss IDE，它带有所有需要的插件预装包，以及一个独立于规则的其他工具的选择。你可以选择只安装来自 JBoss IDE 附带“包”的规则。

3.1.3.1 安装 GEF（一个需要的依赖项）

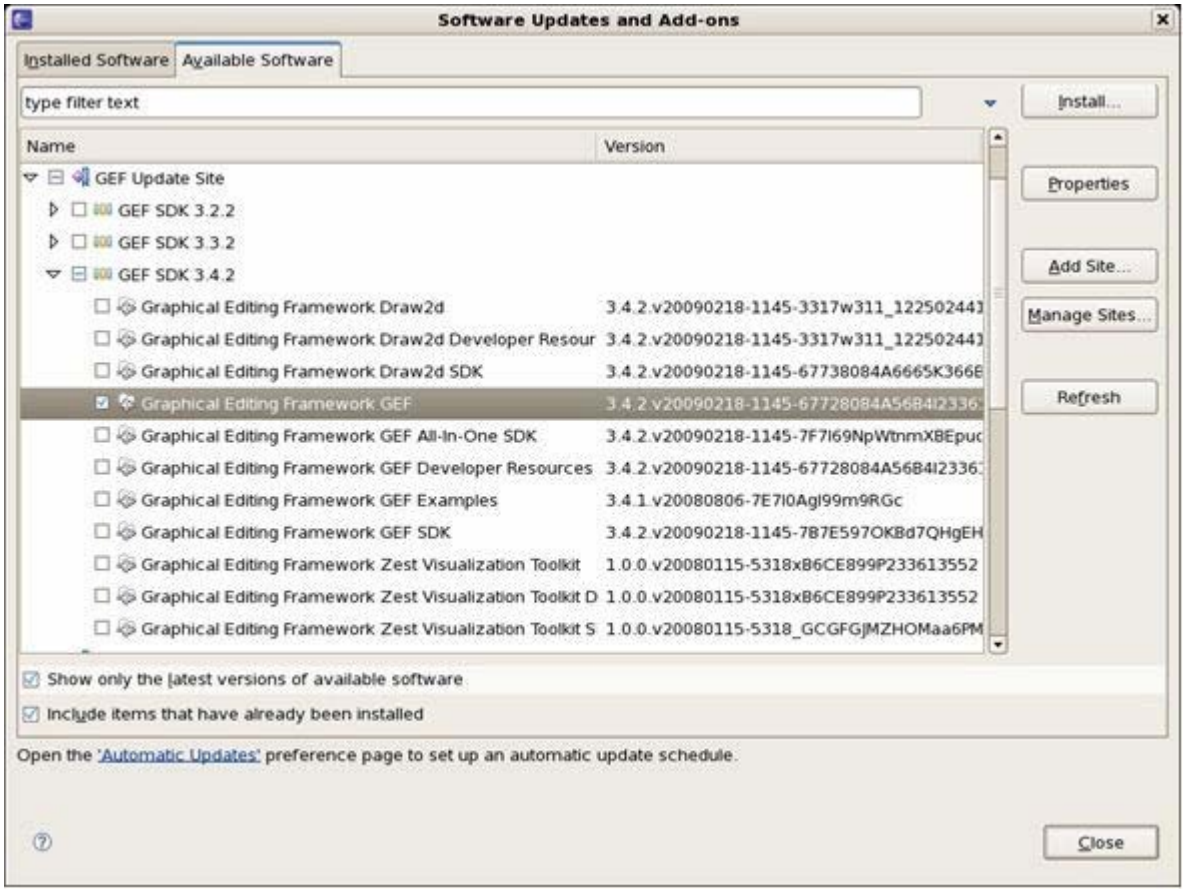
GEF（Graphical Editing Framework）是 Eclipse 图形编辑框架，它用于插件中的图形浏览组件。

如果你没有安装 GEF，你可以使用内置的升级机制安装它（或从 Eclipse.org 网站下载 GEF，不推荐）。JBoss IDE 已有 GEF，如果象大部分其他 Eclipse“发布版”一样做，那么对一些人来说这一步是多余的。

从帮助菜单打开 Help->Software updates...->Available Software->Add Site，定位：

```
http://download.eclipse.org/tools/gef/updates/releases/
```

接下选 GEF 插件：



按 next，同意安装插件（可能需要重启 Eclipse）。一旦完成，则可以继续安装规则插件。

3.1.3.2 从 ZIP 文件安装 GEF

要安装 ZIP 文件，下载并解包文件。在 zip 内你会看见一个插件目录，和插件 jar 本身。你放置插件 jar 到 Eclipse 应用程序 plug-in 目录，然后重启 Eclipse.

3.1.3.3 从 ZIP 文件安装 Drools 插件

从下面的链接下载 Drools Eclipse IDE 插件。在你的主 Eclipse 文件夹解包下载的文件(不要只拷贝文件在那儿，提取它，最终让 feature 和 plugin 的 jar 更新在 Eclipse 的 eatures 和 plugin 目录中），并启 Eclipse。

http://www.jboss.org/drools/downloads.html

要检查安装是否成功，尝试打开 Drools perspective：在你的 Eclipse 窗口的右上角点'Open Perspective'按钮，选'Other...'，然后挑选 Drools perspective。如果你没有找到 Drools perspective，则安装可以没有成功。检查你是否正确地执行了每一步：你有正确 Eclipse (3.4.x)版本吗？你安装了 Eclipse GEF 吗（检查是否 org.eclipse.gef_3.4.*.jar 存在于你的 eclipse 根目录文件夹的 plugins 目录中）？你正确地提取了 Drools Eclipse 插件吗(是否 org.drools.eclipse_*.jar 存在于你的 eclipse 根目录文件夹的 plugins 目录中)？如果没有找到问题，试着联系我们（例如，在 IRC 或用户邮件列表上），详情请看我们的主页：

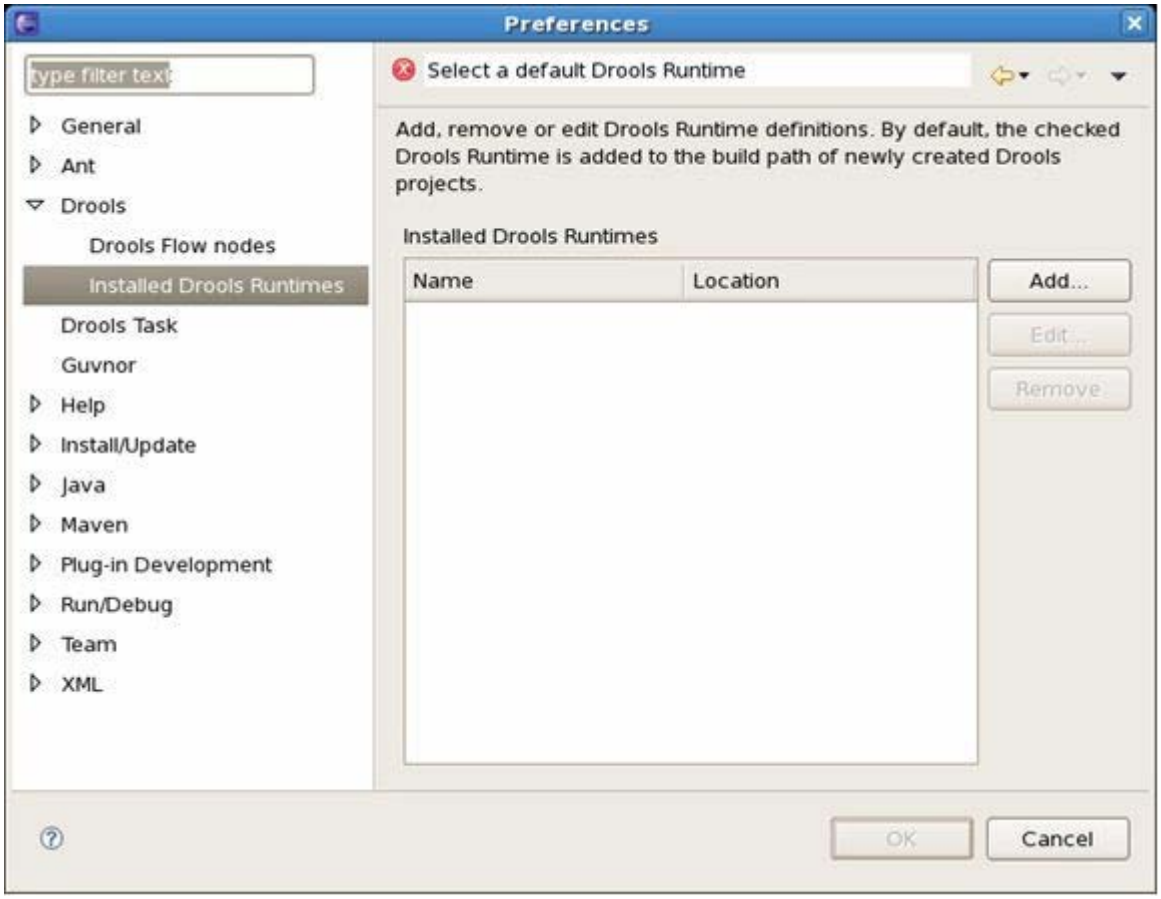
http://www.jboss.org/drools/

3.1.3.4 Drools Runtimes

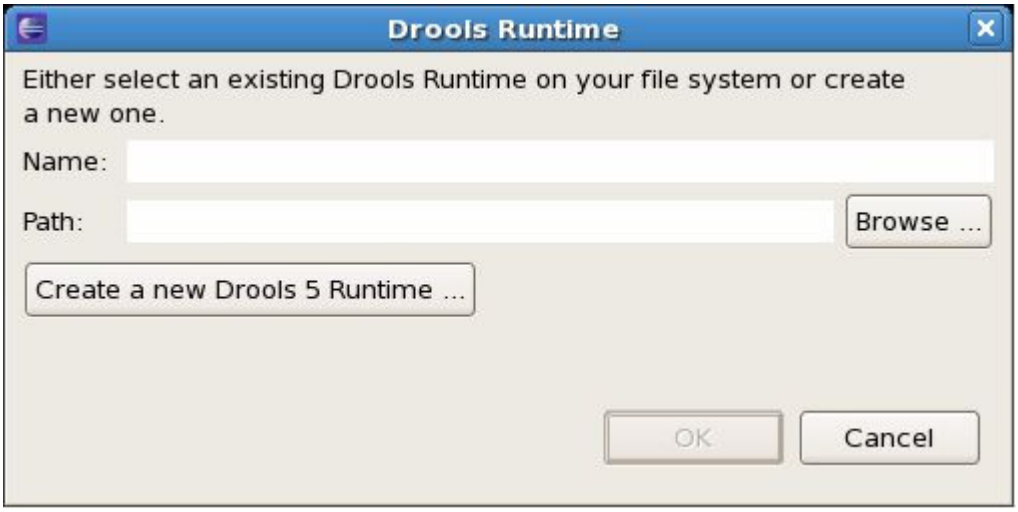
Drools 运行时间是在你系统上的一个 jars 集合，它代表 Drools 项目 jars 的一个特定版本。要创建一个运行时间，你必然指定 IDE 到你选择的版本。如果你希望根据包括插件本身的最新的 Drools 项目 jars 创建一个新的运行时间，你也可以轻松做到这点。你需要为你的 Eclipse 工作空间指定一个默认的 Drools 运行时间，但是每个单独的项目可以覆盖默认的，并为项目特定选择一个合适的运行时间。

3.1.3.4.1 定义一个 Drools 运行时间

你需要使用 Eclipse preferences 视图定义一个或多个运行时间。要打开你的参数选，在菜单窗口中选择 Preferences 菜单项。一个新的参数选择对话窗口会显示所有你的参数选择。在对话窗口听左边，在 Drools 分类下面，选择"Installed Drools runtimes"。那么则在右边的面板显示定义的 Drools 运行时间。如果你没有定义任何运行时间，看起来如下图所示：

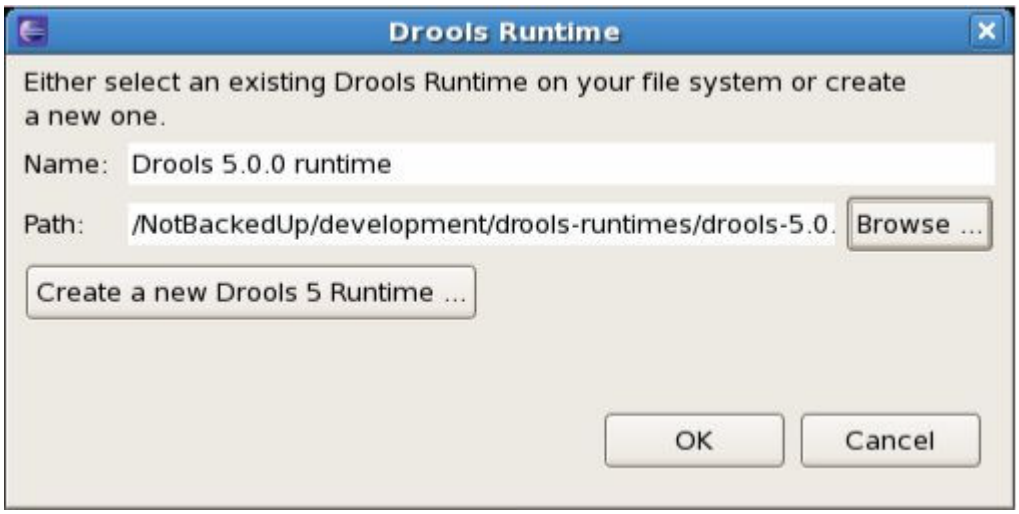


要定义一个新 Drools 运行时间，点击 add 按钮。会弹出如下对话框：需要你的运行时间名字，以及可以在你的文件系统中查找到它的的位置。



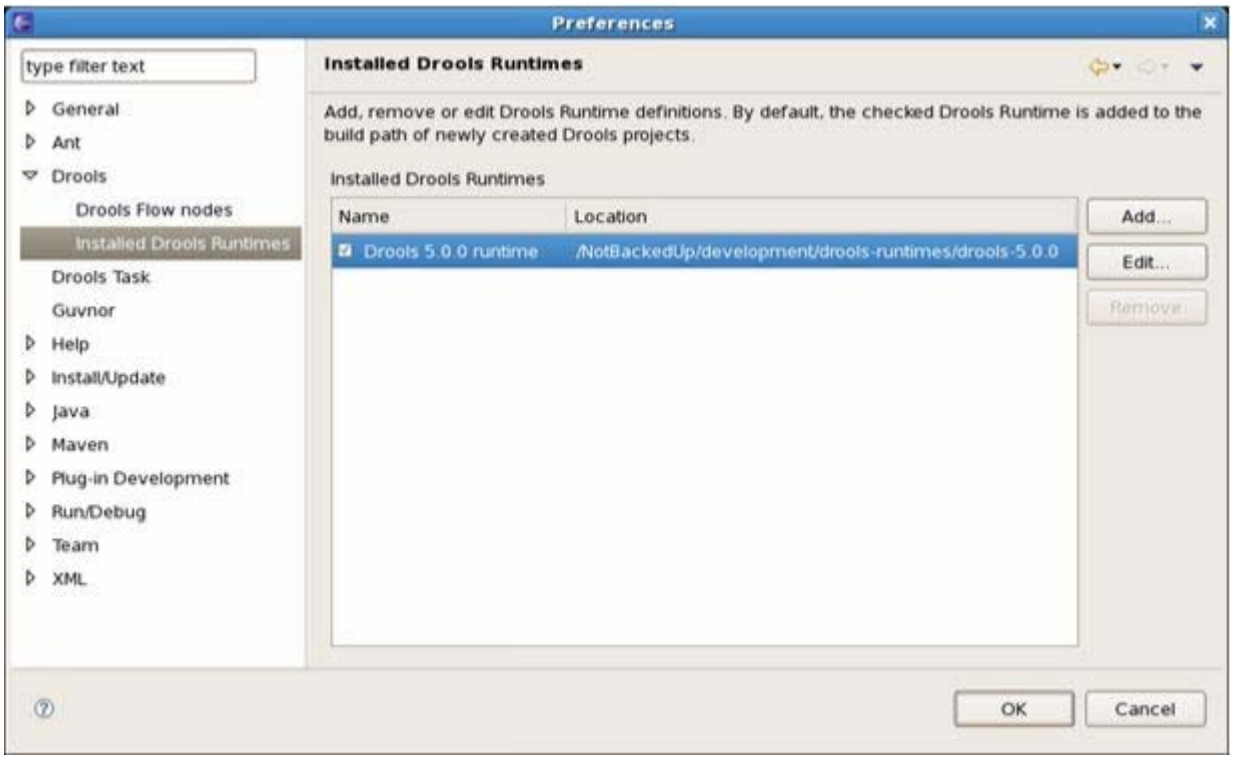
通常你有两个选择：

1. 如果你只是想使用包含在 Drools Eclipse 插件中的默认 jars，你可以点击 "Create a new Drools 5 runtime ..." 按钮，自动创建一个新的 Drools 运行时间。一个文件浏览器会出现，询问你，在你和文件系统中，选择存放你希望创建的运行时间的文件夹。然后，该插件会自动拷贝所有的依赖项到这个特定的文件夹。在选择文件夹后，对话框如会下图所示：

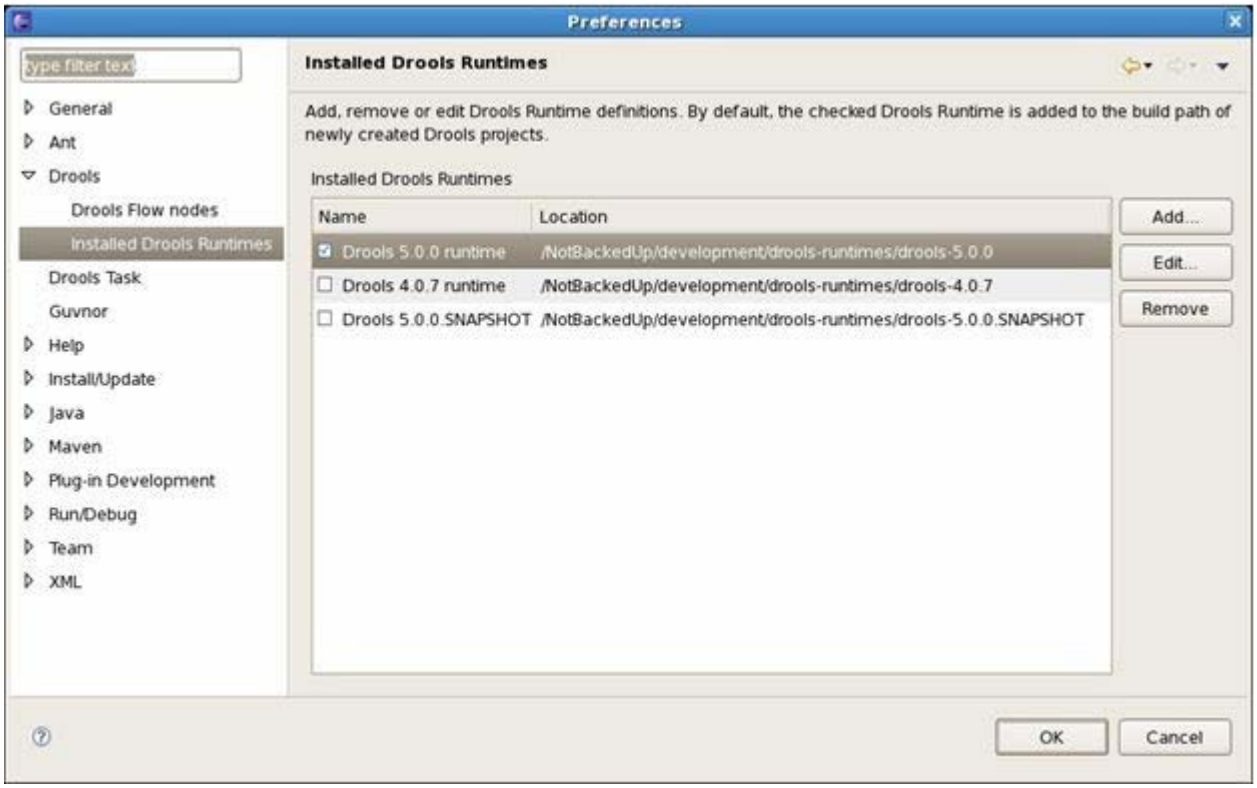


2. 如何你想使用 Drools 项目的一个特定版本，你应该在你的文件系统上创建一个文件夹，其包含所有必需的 Drools 库和依赖项。而不是使用上解释的方式创建一个新的 Drools 运行时间，给你的运行时间一个名字，并选择包含所有需要 jars 的那个文件夹的位置。

点击 ok 按钮后，运行时间会显示在 installed Drools runtimes 的表中，如下所示。点击新创建的运行时间前面的复选框，设置它为默认 Drools 运行时间。默认 Drools 运行时间，会被用来作为所有你没有指定项目运行时间的 Drools 项目的运行时间。



你需要多少运行时间，你就可以增加多少运行时间。例如，下面的截图，显示了一个配置，已定义了三个运行时间：Drools 4.0.7 runtime, Drools 5.0.0 runtime 和 Drools 5.0.0.SNAPSHOT runtime。Drools 5.0.0 runtime 被设置为默认运行时间。

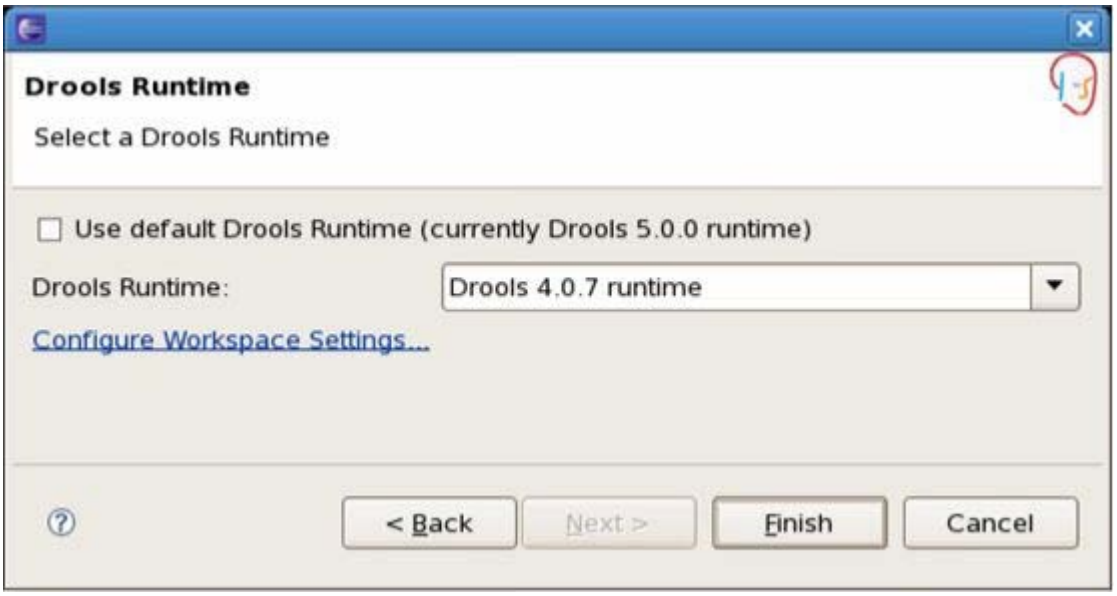


注意：如果你改变了默认运行时间，并且你希望确保所有项目使用更新为相应类路径的默认运行时间，你需要重启 Eclipse。

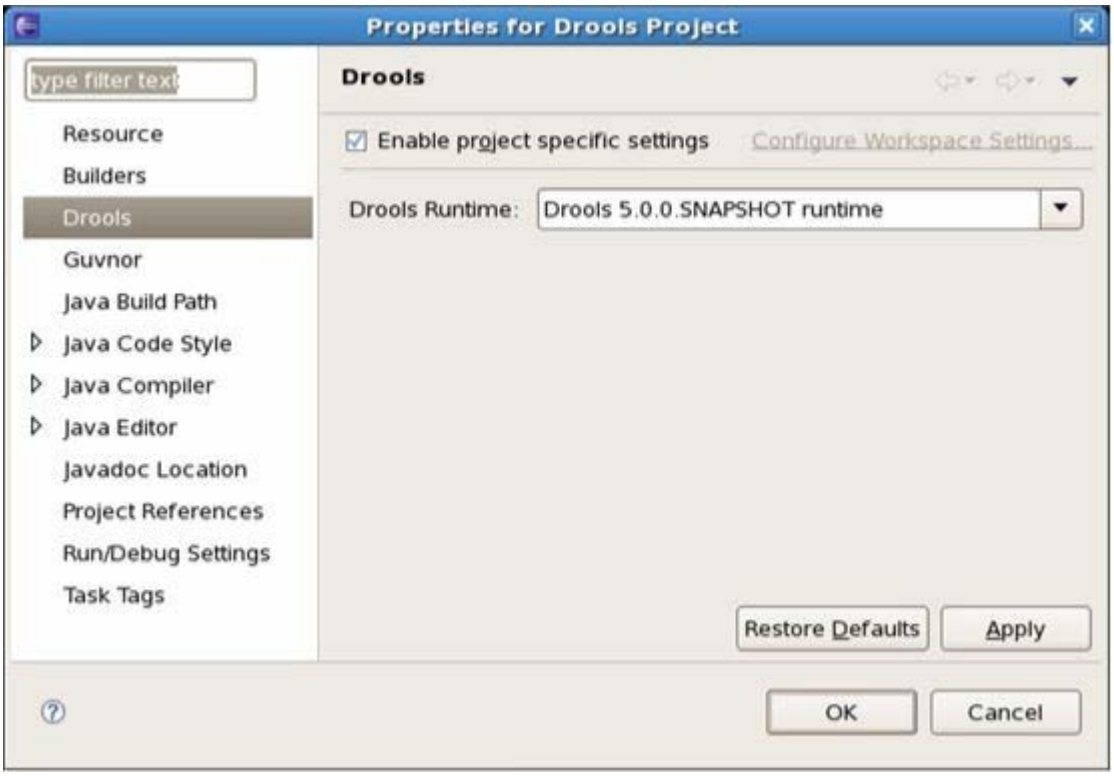
3.1.3.4.2 为你的 Drools 项目选一个运行时间

只要你创建一个 Drools 项目（使用 New Drools Project 向导，或者通过使用"Convert to Drools Project"（当你在 Drools 透视图中右击一个存在的 Java 项目，会显示它）转换一个存在的 Java 项目），插件会自动添加所有需要的 jars 到你的项目的类路径。

当创建一个新的 Drools 项目，插件会自动为那个项目使用默认 Drools 运行时间，除非你指定了一个特定项目 Drools 运行时间。你可以在 New Drools Project 向导的最后一步去做，如下所示，取消"Use default Drools runtime"复选框的选择，在下拉列表框中选择合适的运行时间。如果你点击了"Configure workspace settings ..."链接，显示当前安装 Drools 运行时间的工作空间参数设置会被打开，所以你可以在那儿添加新的运行时间。



你可以在任何时候改变 Drools 项目的运行时间，通过打开项目属性（右击项目，选择 Properties），并且选择 Drools 分类，如下所示。选择"Enable project specific settings" 复选框，并从下拉列表框选择合适的运行时间。如果你点击了"Configure workspace settings ..."链接，显示当前安装 Drools 运行时间的工作空间参数设置会被打开，所以你可以在那儿添加新的运行时间。如果你取消了"Enable project specific settings" 复选框的选择，它会使用你在全局参数设置中定义的默认运行时间。



3.2 根据源代码安装

因为 Drools 是一个开源项目，根据源代码构建的指令是手册的一部分！根据源代码构建意味着你可以领先使用最新的功能。虽然 Drools 的各方面相当复杂，但是许多用户找到了成为贡献者的道路。

Drools 使用 JDK1.5 或更高的版本，你还需要安装以下工具。最低要求提供的版本号：

■·Eclipse 3.4

http://www.eclipse.org/

■·Subversion Client 1.4

http://subversion.tigris.org

http://tortoisesvn.tigris.org - recommended win32 client

■·Maven 2.0.9

http://maven.apache.org/

■·Ant 1.7.0

http://ant.apache.org

确保可执行的 ant, maven 和 java 是在你的类路径中。例子给出的说明是用于 win32 系统：

Path=D:\java\j2sdk1.5.0_8\bin;D:\java\apache-ant-1.7\bin;D:\java\maven-2.0.9\bin

下面的环境变量也需要设置。例子给出的说明是用于 win32 系统：

JAVA_HOME=D:\java\j2sdk1.5.0_8
ANT_HOME=D:\java\apache-ant-1.7
MAVEN_HOME=D:\java\maven-2.0.9

以前的版本习惯使用基于 ANT 构建的机制，但是现在强制使用 MAVEN，尽管 ANT 被用于 MAVEN 内部的文档构建建议。

3.3 源代码的检查输出

来自两个 Subversion 库的 Drools 是可用的：

■·匿名 SVN

http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/

■·开发者放心的 SVN

https://svn.jboss.org/repos/labs/labs/jbossrules/trunk/

检查输出 Drools 源代码，只需执行以下命令：

fmeyer:~/jboss $ svn checkout http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/ trunk

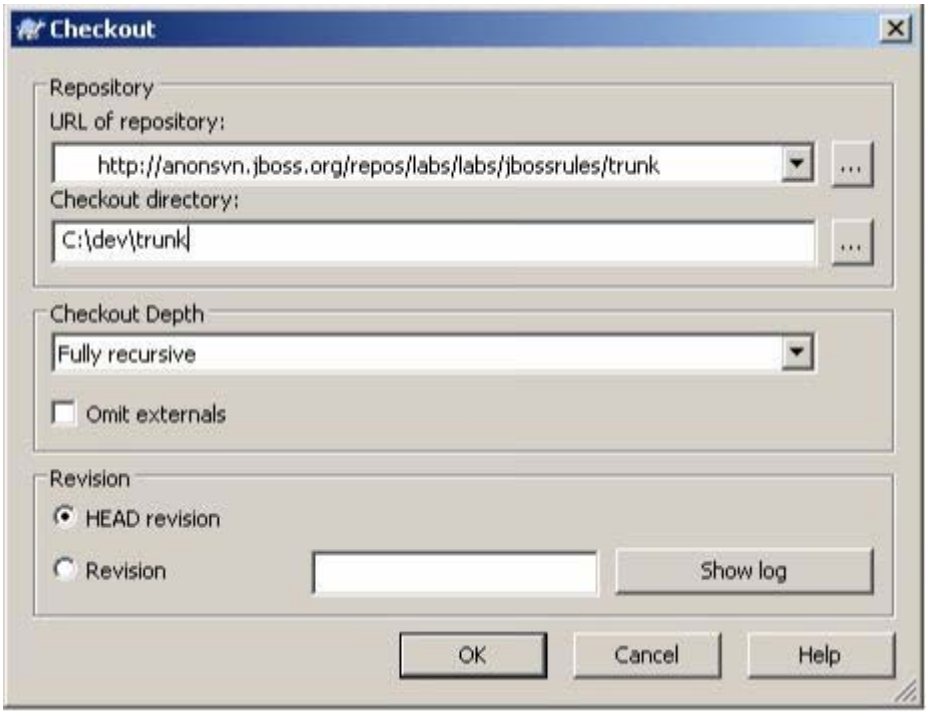
并等待完成文件下载：

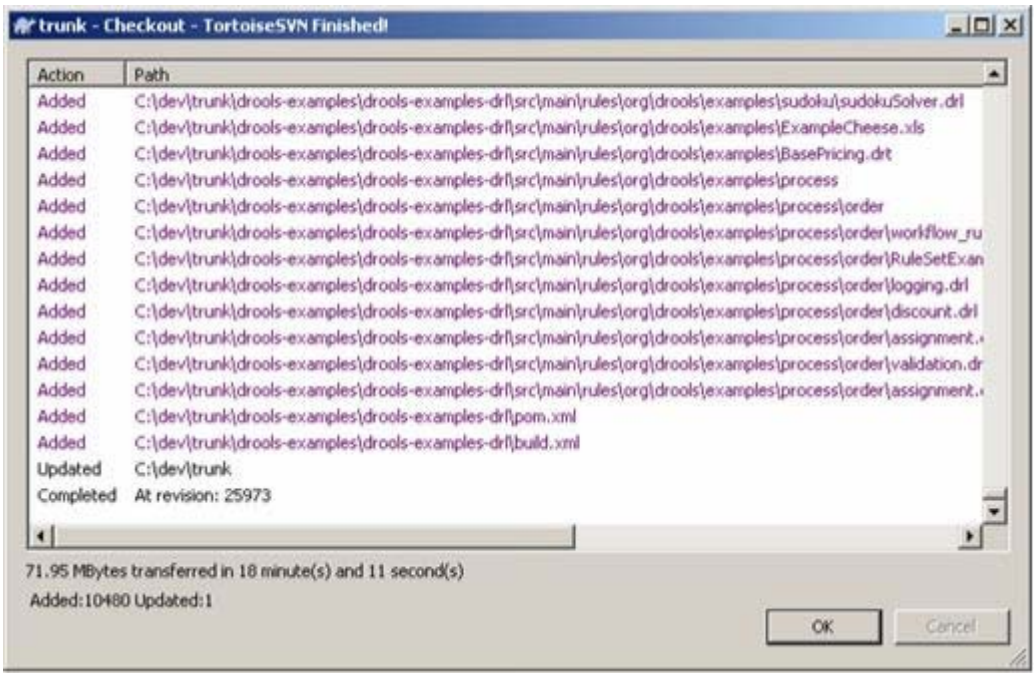
```
A    trunk/drools-repository
A    trunk/drools-repository/.classpath
A    trunk/drools-repository/.project
A    trunk/drools-repository/doc
A    trunk/drools-repository/doc/repository_layout.jpeg
A    trunk/drools-repository/doc/high_level_design.jpeg
A    trunk/drools-repository/doc/javadoc
A    trunk/drools-repository/doc/javadoc/serialized-form.html
A    trunk/drools-repository/doc/javadoc/index-all.html
A    trunk/drools-repository/doc/javadoc/stylesheet.css
A    trunk/drools-repository/doc/javadoc/allclasses-frame.html
A    trunk/drools-repository/doc/javadoc/package-list
A    trunk/drools-repository/doc/javadoc/overview-tree.html
A    trunk/drools-repository/doc/javadoc/org
A    trunk/drools-repository/doc/javadoc/org/drools
A    trunk/drools-repository/doc/javadoc/org/drools/repository
A    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use
A    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use/RuleSet.html
A    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use/RulesRepositoryException.html
A    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use/RulesRepository.html
A    trunk/drools-repository/doc/javadoc/org/drools/repository/RuleSet.html

....

snip

....

A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltz
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltz/waltz.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/manners
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/manners/manners.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltzdb
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltzdb/waltzdb.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples
```

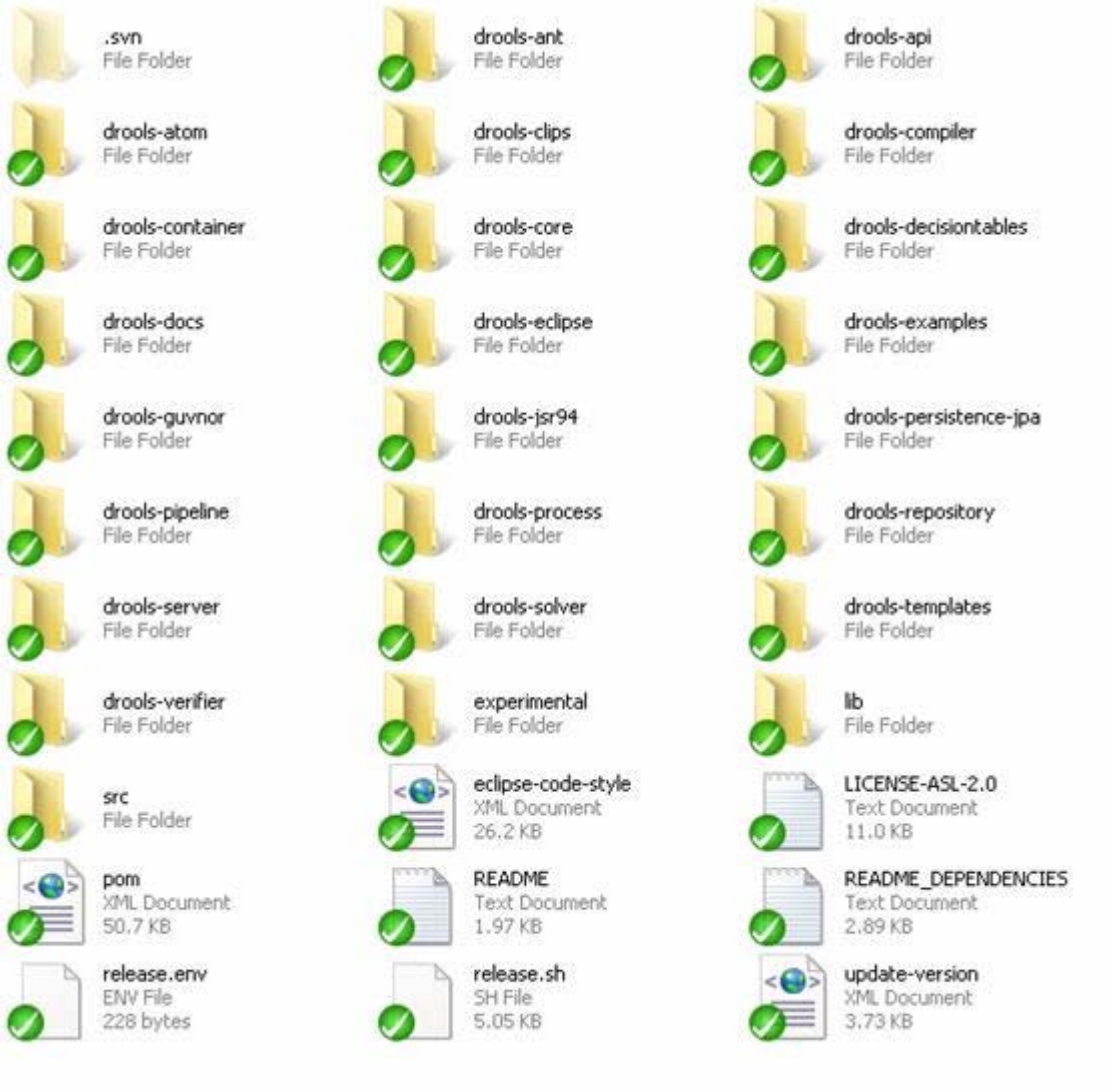
```
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/TroubleTicketWithDSL.
drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/TroubleTicket.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/calculate.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/generation.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/calculate.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/registerNeighb
or.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/killAll.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/registerNeighb
or.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/conway-agendag
roup.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/killAll.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/conway-ruleflo
w.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/generation.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/ticketing.dsl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/StateExampleUsingSali
ence.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/golf.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/LogicalAssertionsNotP
ingPong.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/StateExampleDynamicRu
le.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/sudoku
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/sudoku/sudoku.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/HelloWorld.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/ExamplePolicyPricing.
xls
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/HonestPolitician.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/Fibonacci.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/StateExampleUsingAgen
dGroup.drl
A    trunk/drools-examples/drools-examples-drl/pom.xml
A    trunk/drools-examples/drools-examples-drl/build.xml
 U   trunk
Checked out revision 13656.
```

虽然，我们强力推荐使用命令行工具操作版本库，你也可以使用 Eclipse 集成的 SVN 客户端或 TortoiseSVN。

设置 TortoiseSVN 用于从 subversion 库的检查输出，点击 ok。一旦检查输出完成，你会看到如下文件夹。







3.4 构建

3.4.1 构建源代码

现在我们已有源代码了，接下来是构建和安装源代码。从 Drools 3.1 版开始使用 Maven 2 构建系统。有两个可用于启用关联模块"documentation"和"Eclipse"的配置文件;它们让开发者更快的构建核心模块。Eclipse 配置文件会下载 Eclipse 到 drools-Eclipse 文件夹内，它超过了 100MB 的下载（取决于你的操作系统），然而它只需要下载一次。如果你愿意，你可以移动 Eclipse 下载到其他位置，并使用 -DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror 指定它。下面构建所有 jars,documentationt 和 Eclipse zip，为避免下载 Eclipse，使用了一个指定的本地文件夹:

```
mvn -Declipse -Ddocumentation clean install
-DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror
```

你也可以产生分布式构建，把一切都放入 zips 内，如下所示：

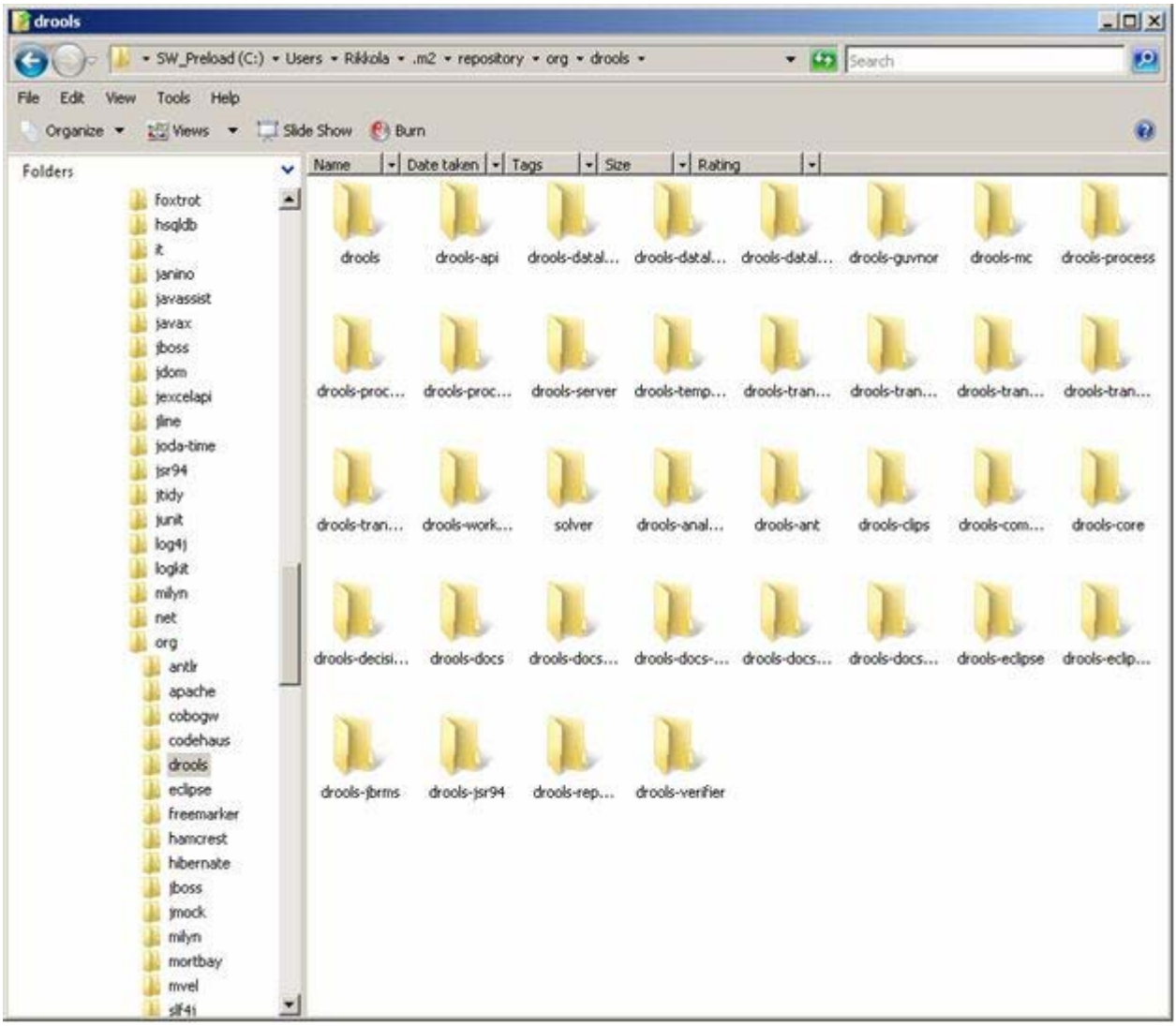
```
mvn -Declipse -Ddocumentation clean install
-DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror
mvn -Ddocumentation -Declipse -DskipTests package javadoc:javadoc assembly:assembly
-DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror
```

注意，必须首先做安装，因为 javadoc:javadoc 不会工作，除非 jars 是在本地 Maven 库中，而且在第二个运行时测试会被跳过。assembly:assembly 会失败，除非为 Maven 增加内存，在 windows 中下面的命令是有效的：set MAVEN_OPTS=-Xmx512m

输入 mvn clean 命令，清除旧的东西，然后测试和构建源代码，并且会报告任何错误。

产生的结果 jars 从项目的顶层被放入/target 目录中。

当 Maven 构建每个模块时，它会自动安装产生的结果 jars 在本地的 Maven 2 库中。



3.4.2 构建手册

手册的构建现在被集成到了 Maven 构建过程中，并且通过使用配置文件（-Ddocumentation）开关或改变到主目录中进行构建。

Drools 对这个手册使用 Docbook。Maven 被用来构建文档，并且这个构建产生三种不同的格式，全部共享相同的图标目录。

- html_single ——整个手册使用一个单独的 html 文档。
- html ——手册被分成多个文档，并放在一个框架中。左边框提供导航。
- Eclipse ——文档适合被包含在一个 Eclipse 的插件中。

在 drools-docs 目录中，可以通过调用 mvn package，根据项目的 pom.xml 文件生成手册，或者在你构建源代码时，增加-Ddocumentation 开关。生成的手册在每个 drools-docs 子目录的 target/ 目录中。在 Drools 项目的根目录运行 mvn -Ddocumentation package assembly:assembly，生成并复制文档到一个 zip 文件，该 zip 文件位于根文件夹的 target/ 目录。

```
[trikkola@trikkola trunk]$ mvn -Ddocumentation clean package assembly:assembly
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   Drools
[INFO]   Drools :: API
[INFO]   Drools :: Core
[INFO]   Drools :: Compiler
[INFO]   Drools :: Templates
[INFO]   Drools :: Decision Tables
[INFO]   Drools :: JSR-94 API Module
[INFO]   Drools :: Pipeline :: Transformer :: Smooks
[INFO]   Drools :: Pipeline :: Transformer :: JAXB
[INFO]   Drools :: Pipeline :: Transformer :: XStream
[INFO]   Drools :: Pipeline :: Transformer :: JXLS
[INFO]   Drools :: Pipeline :: Messenger :: JMS
[INFO]   Drools :: Pipeline
[INFO]   Drools :: Process :: WorkItems
[INFO]   Drools :: Process :: Task
[INFO]   Drools :: Process :: BAM
[INFO]   Drools :: Process
[INFO]   Drools :: Persistence :: JPA
[INFO]   Drools :: Server
[INFO]   Drools :: Verifier
[INFO]   Drools :: Ant Task
[INFO]   Drools :: Repository
[INFO]   Drools :: Guvnor
[INFO]   Drools :: Microcontainer
[INFO]   Drools :: Clips
[INFO]   Drools :: Planner parent
[INFO]   Drools :: Planner core
[INFO]   Drools :: Planner examples
[INFO] Searching repository for plugin with prefix: 'assembly'.
WAGON_VERSION: 1.0-beta-2
[INFO] ------------------------------------------------------------------------
[INFO] Building Drools
[INFO]    task-segment: [clean, package]
```

```
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean]
[INFO] [site:attach-descriptor]
[INFO] Preparing source:jar
[WARNING] Removing: jar from forked lifecycle, to prevent recursive invocation.
[INFO] No goals needed for project - skipping
[INFO] [source:jar {execution: default}]
[INFO] Preparing source:test-jar
[WARNING] Removing: jar from forked lifecycle, to prevent recursive invocation.
[WARNING] Removing: test-jar from forked lifecycle, to prevent recursive invocation.
[INFO] No goals needed for project - skipping
[INFO] [source:test-jar {execution: default}]
[INFO] ------------------------------------------------------------------------
[INFO] Building Drools :: API
[INFO]    task-segment: [clean, package]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean]
[INFO] Deleting directory /home/trikkola/jboss-drools/trunk/drools-api/target

...snip ...

[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] Drools ..................................... SUCCESS [59.889s]
[INFO] Drools :: API .............................. SUCCESS [4.832s]
[INFO] Drools :: Core ............................. SUCCESS [11.027s]
[INFO] Drools :: Compiler ......................... SUCCESS [10.400s]
[INFO] Drools :: Templates ........................ SUCCESS [1.018s]
[INFO] Drools :: Decision Tables .................. SUCCESS [1.179s]
[INFO] Drools :: JSR-94 API Module ................ SUCCESS [1.001s]
[INFO] Drools :: Pipeline :: Transformer :: Smooks ... SUCCESS [0.651s]
[INFO] Drools :: Pipeline :: Transformer :: JAXB ..... SUCCESS [0.711s]
[INFO] Drools :: Pipeline :: Transformer :: XStream ... SUCCESS [0.465s]
[INFO] Drools :: Pipeline :: Transformer :: JXLS ..... SUCCESS [0.481s]
[INFO] Drools :: Pipeline :: Messenger :: JMS ......... SUCCESS [0.879s]
[INFO] Drools :: Pipeline ......................... SUCCESS [0.006s]
[INFO] Drools :: Process :: WorkItems ............. SUCCESS [1.526s]
[INFO] Drools :: Process :: Task .................. SUCCESS [3.104s]
[INFO] Drools :: Process :: BAM ................... SUCCESS [0.580s]
[INFO] Drools :: Process .......................... SUCCESS [0.005s]
[INFO] Drools :: Persistence :: JPA ............... SUCCESS [0.958s]
[INFO] Drools :: Server ........................... SUCCESS [2.216s]
```

```
[INFO] Drools :: Verifier ................................. SUCCESS [1.836s]
[INFO] Drools :: Ant Task ................................. SUCCESS [0.722s]
[INFO] Drools :: Repository ............................... SUCCESS [3.925s]
[INFO] Drools :: Guvnor ................................... SUCCESS [19.850s]
[INFO] Drools :: Microcontainer ........................... SUCCESS [0.676s]
[INFO] Drools :: Clips .................................... SUCCESS [1.464s]
[INFO] Drools :: Planner parent ........................... SUCCESS [0.527s]
[INFO] Drools :: Planner core ............................. SUCCESS [2.209s]
[INFO] Drools :: Planner examples ......................... SUCCESS [4.689s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 minutes 24 seconds
[INFO] Finished at: Tue Apr 07 15:11:14 EEST 2009
[INFO] Final Memory: 48M/178M
[INFO] ------------------------------------------------------------------------>
```

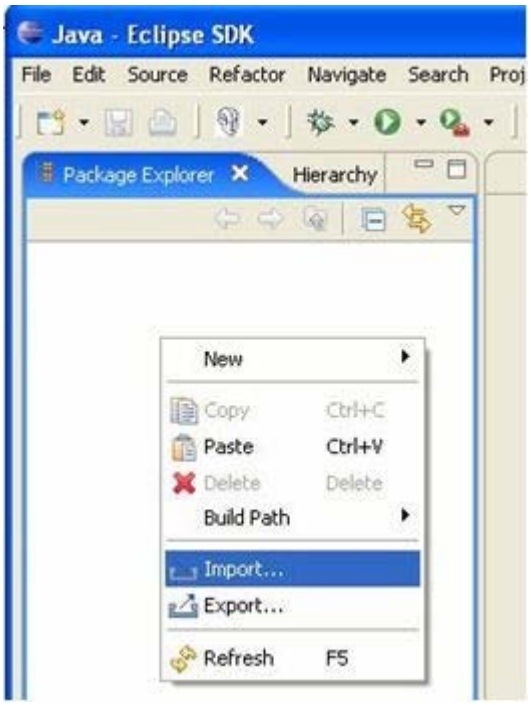
生成的的手册可以在 target/drools-docs-$VERSION.jar 文件中找到，一个具有所有格式的压缩存档。

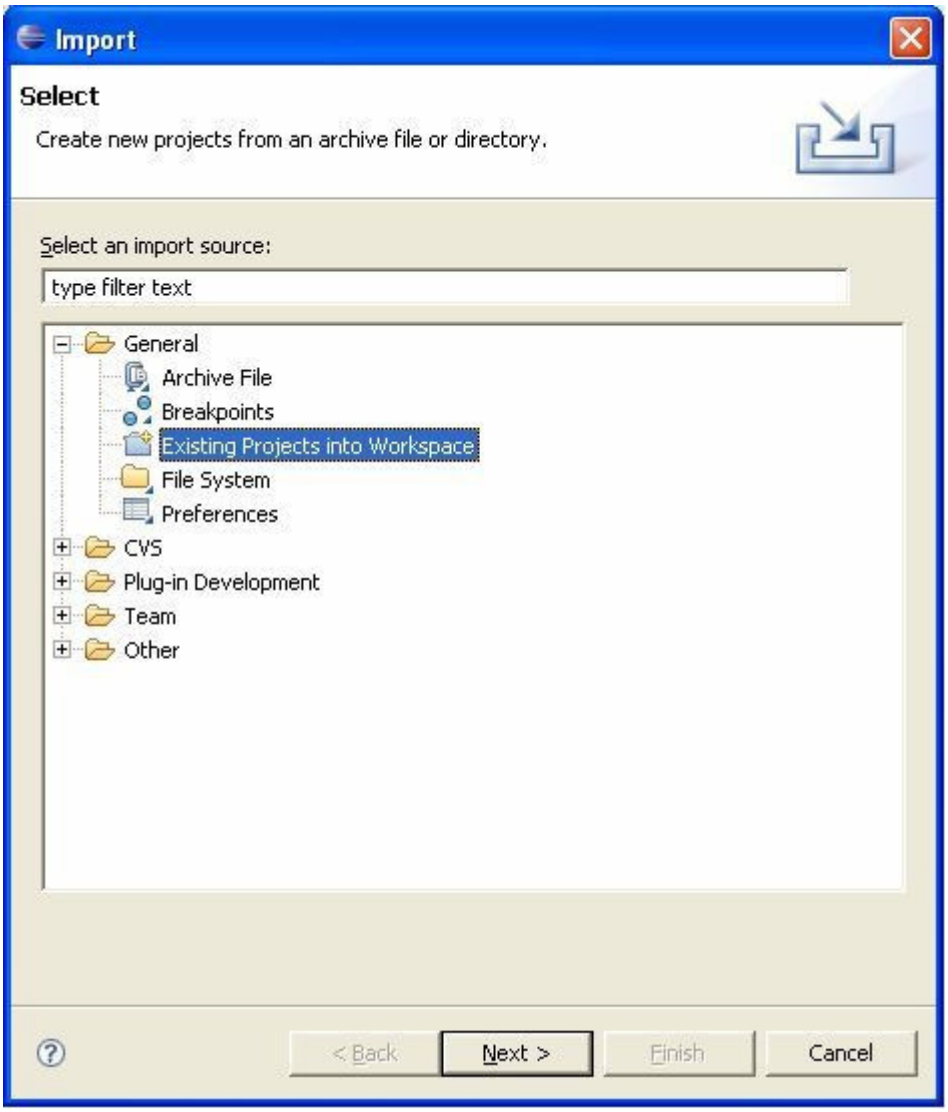
3.5 Eclipse

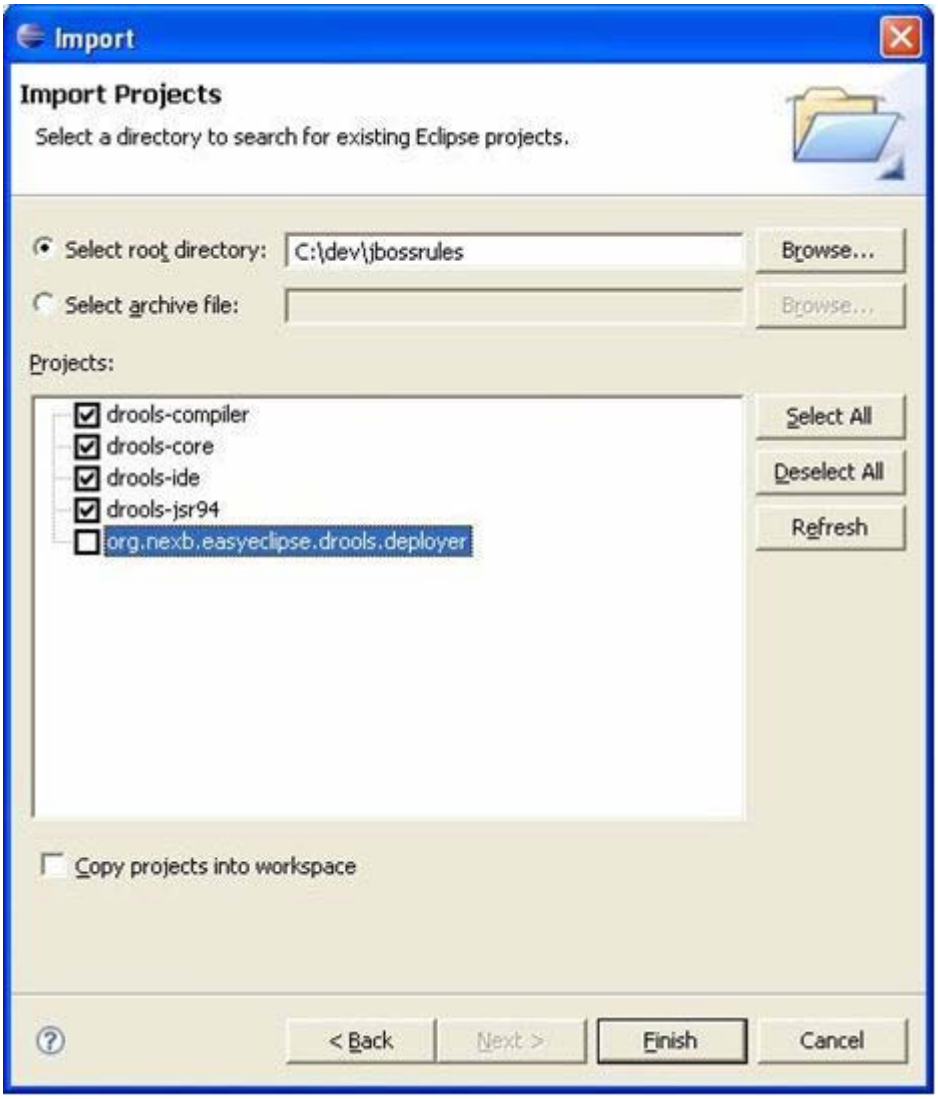
3.5.1 导入 Eclipse 项目

随着生成的 Eclipse 项目文件，现在可以导入到 Eclipse 中。当启动 Eclipse 时，在 subversion 的检查输出根目录中打开工作空间。

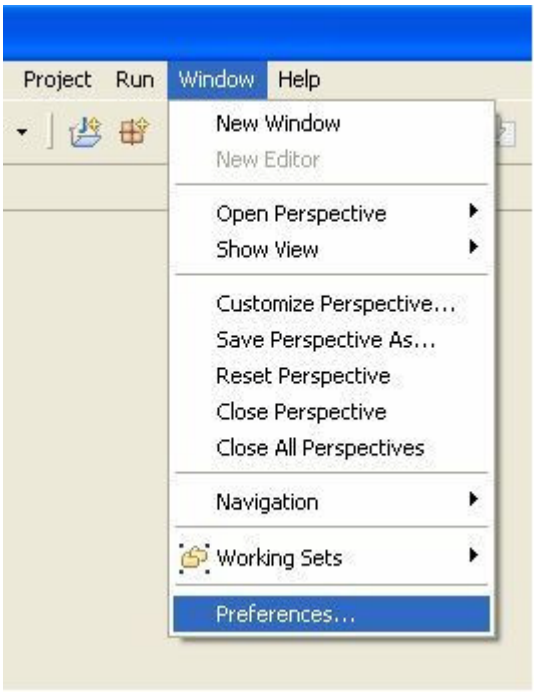

Workspace Launcher
Select a workspace
Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.
Workspace:
c:\dev\jbossrules
Browse...
Use this as the default and do not ask again
OK
Cancel

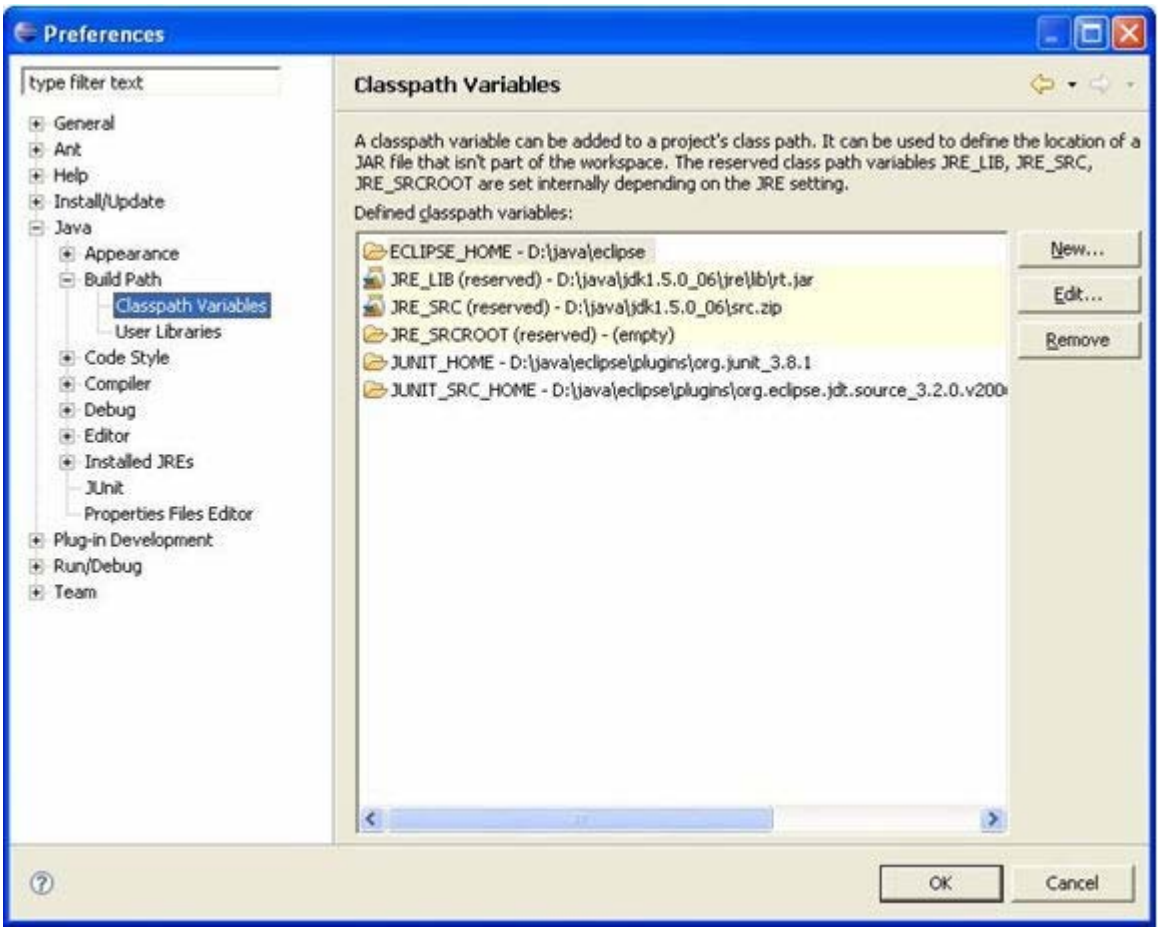

Java - Eclipse SDK
File Edit Source Refactor Navigate Search Proj
Package Explorer
Hierarchy
New
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Build Path
Import...
Export...
Refresh F5


Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
CVS
Plug-in Development
Team
Other
< Back
Next >
Finish
Cancel

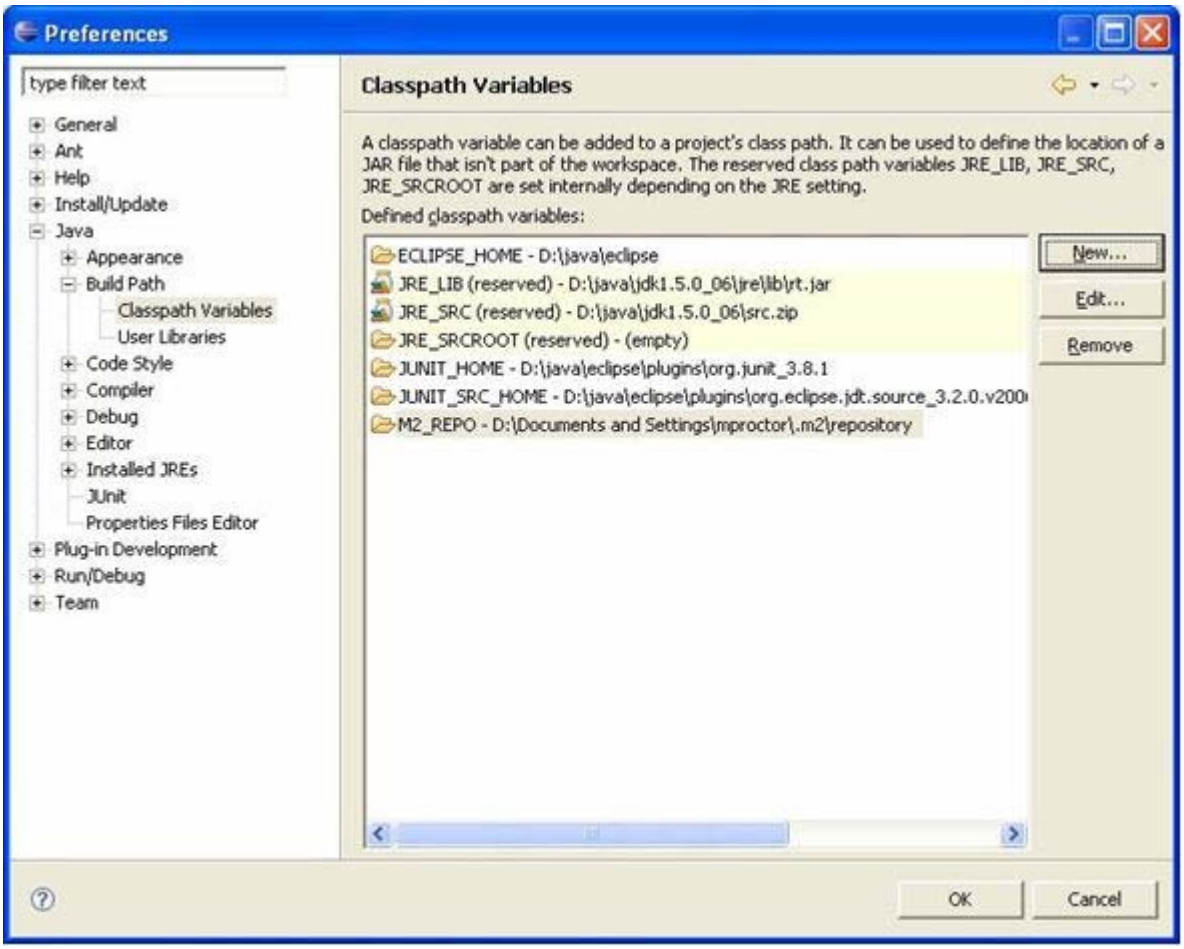


在调用 mvn install 时，所有项目的依赖项被下载，并且被增加到本地 Maven 库中。Eclipse 不能找到这些依赖项，除非你告诉它库在什么地方。要做到这点，设置一个 M2_REPO 类路径变量。


Project
Run
Window
Help
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective
Close Perspective
Close All Perspectives
Navigation
Working Sets
Preferences...


Preferences
type filter text
General
Ant
Help
Install/Update
Java
Appearance
Build Path
Classpath Variables
User Libraries
Code Style
Compiler
Debug
Editor
Installed JREs
JUnit
Properties Files Editor
Plug-in Development
Run/Debug
Team
Classpath Variables
A classpath variable can be added to a project's class path. It can be used to define the location of a JAR file that isn't part of the workspace. The reserved class path variables JRE_LIB, JRE_SRC, JRE_SRCROOT are set internally depending on the JRE setting.
Defined classpath variables:
ECLIPSE_HOME - D:\java\eclipse
JRE_LIB (reserved) - D:\java\jdk1.5.0_06\jre\lib\rt.jar
JRE_SRC (reserved) - D:\java\jdk1.5.0_06\src.zip
JRE_SRCROOT (reserved) - (empty)
JUNIT_HOME - D:\java\eclipse\plugins\org.junit_3.8.1
JUNIT_SRC_HOME - D:\java\eclipse\plugins\org.eclipse.jdt.source_3.2.0.v200
New...
Edit...
Remove
OK
Cancel


New Variable Entry
Name:
M2_REPO
Path:
):/Documents and Settings/mproctor/.m2/repository
File...
Folder...
OK
Cancel


Preferences
type filter text
General
Ant
Help
Install/Update
Java
Appearance
Build Path
Classpath Variables
User Libraries
Code Style
Compiler
Debug
Editor
Installed JREs
JUnit
Properties Files Editor
Plug-in Development
Run/Debug
Team
Classpath Variables
A classpath variable can be added to a project's class path. It can be used to define the location of a JAR file that isn't part of the workspace. The reserved class path variables JRE_LIB, JRE_SRC, JRE_SRCROOT are set internally depending on the JRE setting.
Defined classpath variables:
ECLIPSE_HOME - D:\java\eclipse
JRE_LIB (reserved) - D:\java\jdk1.5.0_06\jre\lib\rt.jar
JRE_SRC (reserved) - D:\java\jdk1.5.0_06\src.zip
JRE_SRCROOT (reserved) - (empty)
JUNIT_HOME - D:\java\eclipse\plugins\org.junit_3.8.1
JUNIT_SRC_HOME - D:\java\eclipse\plugins\org.eclipse.jdt.source_3.2.0.v200
M2_REPO - D:\Documents and Settings\mproctor\.m2\repository
New...
Edit...
Remove
OK
Cancel