

Drools 技术指南

Drools（JBoss Rules）具有一个易于访问企业策略、易于调整以及易于管理的开源业务规则引擎，符合业内标准，速度快、效率高。业务分析师或审核人员可以利用它轻松查看业务规则，从而检验是否已编码的规则执行了所需的业务规则。

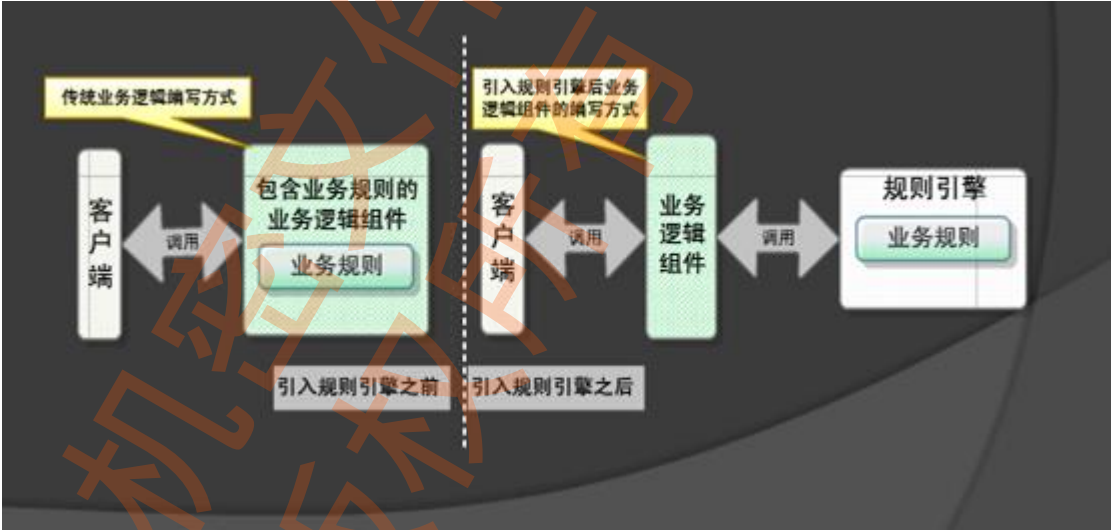
JBoss Rules 的前身是 Codehaus 的一个开源项目叫 **Drools**。最近被纳入 JBoss 门下，更名为 JBoss Rules，成为了 JBoss 应用服务器的规则引擎。

Drools 是为 Java 量身定制的基于 Charles Forgy 的 **RETE 算法**的规则引擎的实现。具有了 OO 接口的 RETE,使得商业规则有了更自然的表达。

Drools 的简要概述

Drools 是一款基于 Java 的开源规则引擎，以将复杂多变的规则从硬编码中解放出来，以规则脚本的形式存放在文件中，使得规则的变更不需要修正代码重启机器就可以立即在线上环境生效

规则引擎由推理引擎发展而来，是一种嵌入在应用程序中的组件，实现了将业务决策从应用程序代码中分离出来，并使用预定义的语义模块编写业务决策。接受数据输入，解释业务规则，并根据业务规则做出业务决策,从而给编程带来了极大的方便。
下图就是引用了规则引擎后的效果：



图(1-1)

Drools 新特性

Drools 推出了一套新的基于 KIE 概念的 API，其目的是将之前版本中对规则引擎繁琐的调用和加载过程加以简化。

Drools6 给我的最大不同就是把 rules 打包成 jar，使用端通过 kie-ci 来动态从 maven repo 中获取指定 rules jar 版本，虽然和 maven 有紧耦合，简化以及清晰了 rules 的使用和动态升级，例如：系统建立 2 个项目：一个 Drools 项目来实现规则，验收规则，生成 jar 包，另外一个就是真正要用规则的项目，直接通过引入不同版本的 jar 包实现规则动态升级。

引入业务规则技术的目的

对系统的使用人员

- 把业务策略（规则）的创建、修改和维护的权利交给业务经理
- 提高业务灵活性
- 加强业务处理的透明度，业务规则可以被管理
- 减少对 IT 人员的依赖程度
- 避免将来升级的风险

对 IT 开发人员

- 简化系统架构，优化应用
- 提高系统的可维护性和维护成本
- 方便系统的整合
- 减少编写“硬代码”业务规则的成本和风险

这里引用了一位 Drools 大咖的博客文章

<http://blog.csdn.net/lifetragedy/article/details/51143914>

如果大咖看到了，请谅解。小编在这里先谢谢您啦 MK 大神！

应用场景

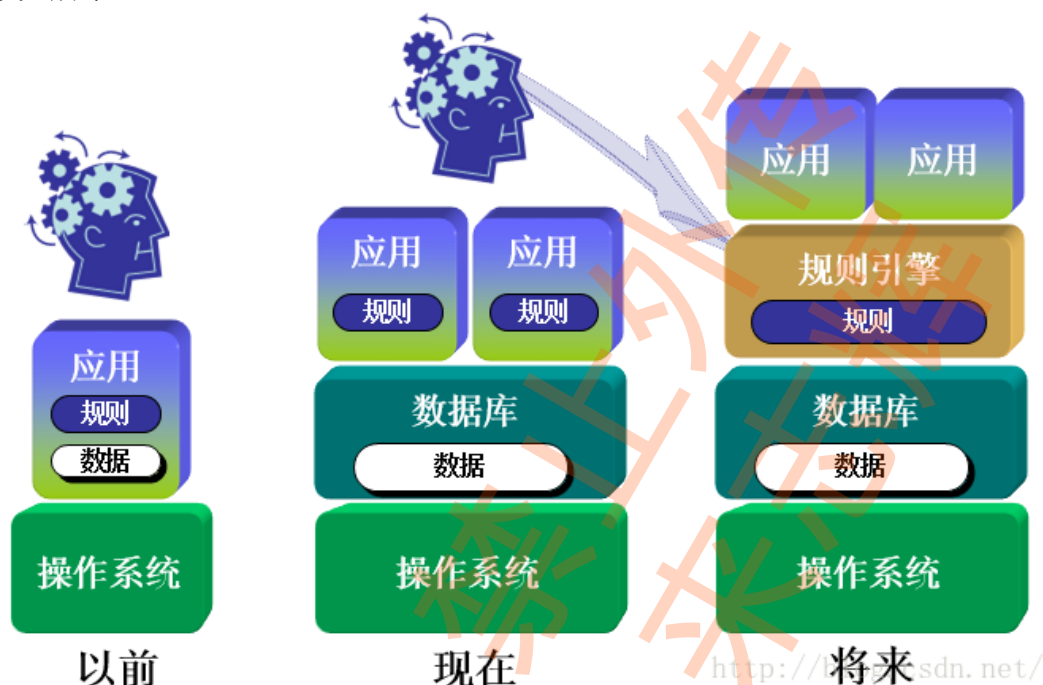
- 为提高效率，管理流程必须自动化，尽管现代商业规则异常复杂。
- 市场要求业务规则经常变化，系统必须依据业务规则的变化快速、低成本的更新。
- 为了快速、低成本的更新，业务人员应能直接管系统中的规则，不需要程序开发人员参与。

作用与优点：

- 将业务规则与业务系统分离，解耦合；
- 实现**自然语言**描述规则逻辑，业务人员易于理解；
- 可视化的规则定制、审批、查询及管理；

- 能有效的提高实现复杂逻辑的代码的可维护性；
- 应付特殊状况，即客户一开始没有提到要将业务逻辑考虑在内；
- 符合组织对敏捷或迭代开发过程的使用；

如下如所示：



(图：1-2)

Drools 的基本工作过程

之前我们一般的做法都是使用一个接口进行业务的工作，首先要传进去参数，其次要获取到接口的实现执行完毕后的结果。其实 Drools 也大相径庭，我们需要传递进去数据，用于规则的检查，调用外部接口，同时还可能需要获取到规则执行完毕后得到的结果。

在 drools 中，这个传递数据进去的对象，术语叫 Fact 对象。Fact 对象是一个普通的 java bean(小编是这样理解的)，规则中可以对当前对象进行任何的读写操作，调用该对象提供的方法，当一个 java bean 插入到 working Memory（内存储存）中，规则使用的是原有对象的引用，规则通过对 fact 对象的读写，实现对应用数据的读写，对于其中的属性，需要提供 getter setter 访问器，规则中，可以动态的往当前 working Memory 中插入删除新的 fact 对象。

注：何为 fact 对象 下面只是小编对 fact 对象的一些理解和认知，有说的不对的地方还请指点

Fact 对象

Fact 是指在 Drools 规则应用当中，将一个普通的 JavaBean 插入到规则的 WorkingMemory 当中后的对象。规则可以对 Fact 对象进行任意的读写操作，当一个 JavaBean 插入到 WorkingMemory 当中变成 Fact 之后，Fact 对象不是对原来的 JavaBean 对象进行 Clon(克隆)，而是原来 **JavaBean 对象的引用**。

规则进行计算的时候需要用到应用系统当中的数据，这些数据设置在 Fact 对象当中，然后将其插入到规则的 WorkingMemory 当中，一个 Fact 对象通常是一个具有 getter 和 setter 方法的 POJO 对象，通过 getter 和 setter 方法可以方便的对 Fact 对象 f 进行操作，所以我们可以简单的把 Fact 对象理解为规则与应用系统数据交互的桥梁或通道。

当 Fact 对象插入到 WorkingMemory 当中后，会与当前 WorkingMemory 当中所有的规则进行匹配，同时返回一个 FactHandler 对象。FactHandler 对象是插入到 WorkingMemory 当中 Fact 对象的引用句柄，通过 FactHandler 对象可以实现对对应的 Fact 对象通过 API 进行删除及修改等操作。

Drools 文件多变的扩展名

说了这么多，这规则引擎是个什么文件呢，也是 *.java *.class *.js 之类的？当然不是，规则引擎可以分为多种方式，最原始也是最基本的是 *.drl 文件，当然也可以是 *.xml 的方式，还可以是 *.xls or *.xlsx 的方式：看着就很灵活是吧。

Drools Hello world

在上面的文章里，我们对 Drools 有一个简单的认识，在这里章节里，小编对详细的对 Drools 语法进行一个说明：

在小编看来 Drools 的基础语法可分为三块内容，包路径、引用、规则体；一个最简单的规则至少要包含“包路径”，“规则体”这两部分。下面我们就写一个 hello world。

hello.drl'

```
package rules.testwrod
    rule "test001"
        when
            eval(true);
        then
            System.out.println("hello world");
        end
    end
```

上面的例子是最简单的一个规则，只要触发该规则时，就会在控制空 输出 hello world，那么这几个都代表的是什么意思呢，说的这三大块内容以是什么呢，小编这里给大家分析一下上面的例子，

首先：规则文件及扩展名，hello.drl 规则名是可以随便起的，不要求像 java 首字母大写，这里小编要提示一下读者，规则名起名最好还是规范，见名知意。

Drl 文件内容：

package 这就是三大块中的包路径，这里的路径是逻辑路径，理论上是可以随便写的，但不能不写。但为了更方便的开发，这里小编提醒大家最好与文件目录同名，像 java 一下以点 (.)的方式隔开

rule 就是三大块中的规则体，以 rule 开头，以 end 结尾，每个规则文件可以包含多个 rule。规则体分为三个部分，LHS RHS 属性 三大部分，下面小编会对这三个部分做一个简单的说明，在语法详情的篇章里，会有说明的。

LHS：条件部分又被称之为 Left Hand Side，简称为 LHS，在一个规则当中 when 与 then

中间的部分就是 LHS 部分。在 LHS 当中，可以包含 0~n 个条件，如果 LHS 部分没空的话，那么引擎会自动添加一个 eval(true)的条件，由于该条件总是返回 true，所以 LHS 为空的规则总是返回 true。

也就是这样的效果

hello.drl'

```
package rules.testwrod
    rule "test001"
        when
            //这里如果为空 则表示 eval(true);
        then
            System.out.println("hello world");
        end
```

RHS：结果部分又被称之为 Right Hand Side，简称为 RHS，在一个规则当中 then 后面部分就是 RHS，只有在 LHS 的所有条件都满足时 RHS 部分才会执行。

RHS 部分是规则真正要做事情的部分，可以将因条件满足而要触发的动作写在该部分当中，在 RHS 当中可以使用 LHS 部分当中定义的绑定变量名、设置的全局变量、或者是直接编写 Java 代码（对于要用到的 Java 类，需要在规则文件当中用 import 将类导入后方能使用，这点和 Java 文件的编写规则相同）。

我们知道，在规则当中 LHS 就是用来放置条件的，所以在 RHS 当中虽然可以直接编写 Java 代码，但不建议在代码当中有条件判断，如果需要条件判断，那么请重新考虑将其放在 LHS 当中，否则就违背了使用规则的初衷。

在 Drools 当中，在 RHS 里面，提供了一些对当前 Working Memory 实现快速操作的宏函数或对象，比如 insert/insertLogical、update/modify 和 retract 就可以实现对当前 Working Memory 中的 Fact 对象进行新增、修改或者是删除；如果您觉得还要使用 Drools 当中提供的其它方法，那么您还可以使用另一外宏对象 drools，通过该对象可以使用更多的操作当前 Working Memory 的方法；同时 Drools 还提供了一个名为 kcontext 的宏对象，使我们可以通过该对象直接访问当前 Working Memory 的 KnowledgeRuntime。

这里我们有提到了 import，这是一个什么概念呢，其实很简单，就是引入所需要的 Java 类或方法。**import**：导入规则文件需要使用到的外部变量，这里的使用方法跟 java 相同，但是不同于 java 的是，这里的 import 导入的不仅仅可以是一个类，也可以是这个类中的某一个可访问的静态方法。小编这里要提醒一下读者，在规则文件里 package 永远是第一行比如：

import com.drools.demo.point.PointDomain; 导入类

import function com.drools.demo.point.PointDomain.getByld; 导入静态方法

例如下面代码：

hello.drl

```
package rules.testwrod
import cn.test.Person;
    rule "test001"
        when
```

```
$p:Person();  
then  
    ntln("hello "+$p.getName());  
end
```

Drools 的 API 调用

在上一章节里，小编简单的讲述了规则文件的编辑语法与规范，读者还没有看过 rule 的执行过程，下面我们就通过例子对 rule 进行一下调用。在 Drools 当中，规则的编译与运行要通过 Drools 提供的各种 API 来实现，这些 API 总体来讲可以分为三类：**规则编译**、**规则收集**和**规则的执行**。

在调用时，我们先要做以下几个操作：

1、Kmodule.xml 的编辑

kmodule.xml 文件放到 **src/main/resources/META-INF/**文件夹下

```
public static final String KMODULE_FILE_NAME = "kmodule.xml";  
public static final String KMODULE_JAR_PATH = "META-INF/" + KMODULE_FILE_NAME;  
public static final String KMODULE_INFO_JAR_PATH = "META-INF/kmodule.info";  
public static final String KMODULE_SRC_PATH = "src/main/resources/" + KMODULE_JAR_PATH;  
public static final String KMODULE_SPRING_JAR_PATH = "META-INF/kmodule-spring.xml";
```

(图 2-1)

代码的实现（具体内容）

```
<?xml version="1.0" encoding="UTF-8"?>  
<kmodule xmlns="http://www.drools.org/xsd/kmodule">  
    <kbase name="kbase1" packages="rules.testwrod">  
        <ksession name="session"/>  
    </kbase>  
</kmodule>
```

2、API 的说明，创建一个 java 文件

Mavne pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
    <modelVersion>4.0.0</modelVersion>  
    <groupId>com.drools.modules.test</groupId>  
    <artifactId>drools-moudles</artifactId>  
    <version>1.0-SNAPSHOT</version>  
    <packaging>jar</packaging>  
  
    <name>drools-moudles</name>  
    <url>http://maven.apache.org</url>
```



```
<properties>

  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  <!-- drools 规则引擎 版本 -->

  <drools.version>7.0.0.Final</drools.version>

  <spring.version>4.2.6.RELEASE</spring.version>

  <log4j2.version>2.5</log4j2.version>

</properties>

<!-- 依赖项定义 -->

<dependencies>

  <!-- start drools -->

  <dependency>

    <groupId>org.drools</groupId>

    <artifactId>drools-core</artifactId>

    <version>${drools.version}</version>

  </dependency>

  <dependency>

    <groupId>org.drools</groupId>

    <artifactId>drools-compiler</artifactId>

    <version>${drools.version}</version>

  </dependency>

  <dependency>

    <groupId>org.drools</groupId>

    <artifactId>drools-decisiontables</artifactId>

    <version>${drools.version}</version>

  </dependency>

  <dependency>

    <groupId>org.drools</groupId>

    <artifactId>drools-workbench-models-guided-template</artifactId>

    <version>${drools.version}</version>

  </dependency>

  <dependency>

    <groupId>org.drools</groupId>

    <artifactId>drools-simulator</artifactId>
```

```
<version>${drools.version}</version>
</dependency>

<dependency>

  <groupId>org. jbpm</groupId>

  <artifactId>jbpm-flow-builder</artifactId>

  <version>${drools.version}</version>

</dependency>

<dependency>

  <groupId>org. kie</groupId>

  <artifactId>kie-spring</artifactId>

  <version>${drools.version}</version>

</dependency>

<dependency>

  <groupId>org. kie</groupId>

  <artifactId>kie-ci</artifactId>

  <version>${drools.version}</version>

</dependency>

<dependency>

  <groupId>org. kie</groupId>

  <artifactId>kie-internal</artifactId>

  <version>${drools.version}</version>

</dependency>

<dependency>

  <groupId>org. kie</groupId>

  <artifactId>kie-api</artifactId>

  <version>${drools.version}</version>

</dependency>

<dependency>

  <groupId>org. drools</groupId>

  <artifactId>drools-workbench-models-guided-dtable</artifactId>

  <version>${drools.version}</version>
```



```

</dependency>

<dependency>

    <groupId>org. drools</groupId>

    <artifactId>drools-templates</artifactId>

    <version>${drools.version}</version>

</dependency>

<!-- end drools -->
</dependencies>

<build>

    <testResources>

        <testResource>

            <directory>

                ${project.basedir}/src/main/resources

            </directory>

        </testResource>

    </testResources>

</build>

<plugins>

<plugin>

    <groupId>org. apache. maven. plugins</groupId>

    <artifactId>maven-compiler-plugin</artifactId>

    <configuration>

        <source>1.7</source>

        <target>1.7</target>

    </configuration>

</plugin>

</plugins>

</build>
</project>

```

JAVA code

```

package com.drools.test;

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

```

```

public class TestWrod{
    public static void main(String[] args) {
        KieServices kss = KieServices.Factory.get();
        KieContainer kc = kss.getKieClasspathContainer();
        KieSession ks =kc.newKieSession("session");
        int count = ks.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        ks.dispose();
    }
}

```

分析 java 代码

从 classpath 中读取 kmodule，创建 KieContainer 容器。

利用 kieContainer 对象创建一个新的 KieSession，创建 session 的时候我们传入了一个 name: **session**”，这个字符串很眼熟吧，这个就是我们定义的 kmodule.xml 文件中定义的 ksession 的 name。

kieContainer 根据 kmodule.xml 定义的 ksession 的名称找到 KieSession 的定义，然后创建一个 KieSession 的实例。

KieSession 就是一个到规则引擎的链接，通过它就可以跟规则引擎通讯，并且发起执行规则的操作。

然后通过 kSession.fireAllRules 方法来通知规则引擎执行规则

ks.dispose();最后将 kiesession 连接关闭

那让我们看一下结果 如图 2-2

```

2017-16-25 03:16:41 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_RULES
2017-16-25 03:16:41 [DEBUG] org.drools...DefaultAgenda - Fire Loop
hello world
2017-16-25 03:16:41 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-16-25 03:16:41 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-16-25 03:16:41 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
总执行了1条规则

```

(图 2-2)

上面只最简单的 hello world 了，是不是很容易就懂了呢，好！那小编再增加一点点难度，我们往规则里插入一个值，来进行一个简单的业务判断。

业务说明：判断人名是张三，年龄 30 岁，就将该人名改为李四

实体 POJO

```

package com.drools.test;

public class Person {
    private String name;
    private int age;
    private String desc;

    public Person(String name, int age) {
        this.name = name;
    }
}

```

```
        this.age = age;
    }
    ... 此处省略 get set 方法，但读者做例子时一定要加上哦
```

规则代码如下：

Person.drl

```
package rules.testwrod

import com.drools.test.Person
rule test001
    when
        $p:Person(name=="张三",age==30);
    then
        $p.setName("李四");
        System.out.println("改完后的名字"+$p.getName());
    end
```

在 API 代码说明

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

public class TestWrod
{
    public static void main(String[] args)
    {
        KieServices kss = KieServices.Factory.get();
        KieContainer kc = kss.getKieClasspathContainer();
        KieSession ks =kc.newKieSession("session");
        Person person=new Person("张三",30);
        FactHandle insert = ks.insert(person);
        int count = ks.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        System.out.println(person.getName());
        ks.dispose();
    }
}
```

那让我们看一下结果 如图 2-3

改完后的名字李四

```
2017-34-25 04:34:01 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-34-25 04:34:01 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-34-25 04:34:01 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
总执行了1条规则
李四
```

(图 2-3)

在控制台上我们可以看到是我们想要的结果。在规则里变了，java 中的 Bean 也发生了变化，这就满足了我们业务上的要求？

小编为什么在最后打了一个问号呢，我们的真的改变了 fact 对象嘛，是我们真正想要的结果嘛，看起来是没问题的，控制台也输出，但如果我稍稍修改一下业务的话，在之前的业务上添加 并将名为李四的的年龄设置为 40，那我规则就应该是这样了
规则代码如下：

Person.drl

```
package rules.testwrod

import com.drools.test.Person
rule test001
    when
        $p:Person(name=="张三",age==30);
    then
        $p.setName("李四");
        System.out.println("改完后的名字"+$p.getName());
    end

rule test002
    when
        $p:Person(name=="李四");
    then
        $p.setAge(40);
        System.out.println("改完后的名字"+$p.getName()+"改完后的年龄"+$p.getAge());
    end
```

java 的代码不变，执行结果，我们发现结果与第一次相同，难道是我们写的代码没有编译？为什么没有生效呢，test001 规则明明已经将 Person 中的 name 属性改为“李四”了那为什么值规则 test002 没有执行呢，这里小编就要郑重的提一句了，这是因为 rete 的算法问题，什么是 rete 算法呢，在后面的章节里小编会做一个详细的说明，好！，那小编先带着读者解决这个问题。其实解决起来很简单，只要在第一个规则里添加之前所说的 update 就可以了。

将 test001 规则中的 then 中 \$p.setName("李四");下方添加 **update(\$p);**再次运行
那让我们看一下结果 如图 2-4

改完后的名字李四

```
2017-15-25 05:15:27 [DEBUG] org.drools...DefaultAgenda - Fire Loop
```

改完后的名字李四改完后的年龄40

```
2017-15-25 05:15:27 [DEBUG] org.drools...DefaultAgenda - Fire Loop
```

```
2017-15-25 05:15:27 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
```

```
2017-15-25 05:15:27 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
```

总执行了2条规则

李四

(图 2-4)

注：小编是这样认为的：其实导致这个原因的是因为 rete 算法的问题，简单说明一下，rete 算法会将规则中的内容先全部加载出来，我们在规则中看似把 Person 的 name 属性改变了，但本质中只是引用发生了改变，fact 对象是没有真正改变的。当 fact 对象发生真正改变时，规则将重新执行，但这样是有风险的，容易产生死循环。解决方案会在 rule 的属性中有说明

Drools7 版本关于 session 的不同方法

小编为什么会要将 session 独立出来一个章节呢，这是有原因的，我们在开发过程会遇到各种情况，这个 session 的状态是很重要的，也会在开发中经常使用。

KieSession 用于与规则引擎进行交互的会话。

会话分为两类：

- 有状态的 KieSession
- 无状态的 StatelessKieSession

KieSession 有状态的

KieSession 会在多次与规则引擎进行交互中，维护会话的状态。

定义 KieSession，在 *kmodule.xml* 文件中定义 type 为 **stateful** 的 session：

```
<ksession name="stateful_session" type="stateful"></ksession>
```

注意：stateful 是 type 属性的默认值。

获取 KieSession 实例

```
KieSession statefulSession = kieContainer.newKieSession("stateful_session");
```

接下来，可以在 KieSession 执行一些操作。最后，如果需要清理 KieSession 维护的状态，调用 dispose() 方法。

StatelessKieSession

与 KieSession 相反，StatelessKieSession 隔离了每次与规则引擎的交互，不会维护会话的状态。

如果将 session 比作编程语言中的函数，StatelessKieSession 就是无副作用的函数。

StatelessKieSession 适用场景：

- 数据校验
- 运算
- 数据过滤
- 消息路由
- 任何能被描述成函数或公式的规则

定义 StatelessKieSession，在 *kmodule.xml* 文件中定义 type 为 **stateless** 的 session：

```
<ksession name="stateless_session" type="stateless"></ksession>
```

如果我们想要用无状态的 kie-session 的话就必须这样定义了。

获取 StatelessKieSession 实例：

```
StatelessKieSession statelessKieSession =  
kieContainer.newStatelessKieSession("stateless_session");  
//通过 KieServices 获取 command 工厂类 KieCommands:
```

```

KieCommands commandFactory = kieServices.getCommands();
//可以使用工程类 KieCommands 调用 newXXXCommand 开头的方法创建 command 实例。
//会话执行 command:
statelessKieSession.execute(command);
//----- 分隔线 -----
[source code] 调用的是该接口
public interface StatelessRuleSession {
    void execute(java.lang.Object o);
    void execute(java.lang.Iterable iterable);
}
StatelessKnowledgeSessionImpl 实现类
    public void execute(Object object) {
        StatefulKnowledgeSession ksession = newWorkingMemory();
        try {
            ksession.insert( object );
            ksession.fireAllRules();
        } finally {
            dispose(ksession);
        }
    }
}

```

从代码中我们可以看出来，也是通过 finally 中的 dispose 方法来删除的，只是与 kieSession 方式不同。

Drools 内部功能详细介绍

规则文件

在 Drools 当中，一个标准的规则文件就是一个以 “.dr1” 结尾的文本文件，由于它是一个标准的文本文件，所以可以通过一些记事本工具对其进行打开、查看和编辑。规则是放在规则文件当中的，一个规则文件可以存放多个规则，除此之外，在规则文件当中还可以存放用户自定义的函数、数据对象及自定义查询等相关在规则当中可能会用到的一些对象。

一个标准的规则文件的结构代码清单：

<code>package package-name</code>	包名，必须的，指限于逻辑上的管理，若自定义查询或者函数属于同一包名，不管物理位置如何，都可以调用
<code>imports</code>	需要导入的类名
<code>globals</code>	全局变量
<code>functions</code>	函数
<code>queries</code>	查询
<code>rules</code>	规则，可以有多个

除 package 之外，其它对象在规则文件中的顺序是任意的，也就是说在规则文件当中必须要有一个 package 声明，同时 package 声明必须要放在规则文件的第一行，规则文件当中的 package 和 Java 语言当中的 package 有相似之处，但不完全相同。在 Java 当中 package 的作用是用来对功能相似或相关的文件放在同一个 package 下进行管理，这种 package 管理既有物理上 Java 文件位置的管理也有逻辑上的文件位置的管理，在 Java 当中这种通过 package 管理文件要求在文件位置在逻辑上与物理上要保持一致；但在 Drools 的规则文件当中 package 对于规则文件中规则的管理只限于逻辑上的管理，而不管其在物理上的位置如何，这点是规则与 Java 文件的 package 的区别。

对于同一 package 下的用户自定义函数、自定义的查询等，不管这些函数与查询是否在同一个规则文件里面，在规则里面是可以直接使用的，这点和 Java 的同一 package 里的 Java 类调用是一样的。

规则语言

<code>rule "name"</code>	
<code>attributes</code>	属性
<code>when</code>	
<code>LHS</code>	条件
<code>then</code>	
<code>RHS</code>	结果
<code>end</code>	

一个规则可以包含三个部分：唯有 attributes 部分是可选的，其他都是必填信息：

定义当前规则执行的一些属性等，比如是否可被重复执行、过期时间、生效时间等条件部分：

即 LHS，定义当前规则的条件，如 `when Message()`；判断当前 `workingMemory` 中是否存在 `Message` 对象。

结果部分：

即 RHS，这里可以写普通 java 代码，即当前规则条件满足后执行的操作，可以直接调用 Fact 对象的方法来操作应用

条件部分：

条件部分又被称之为 Left Hand Side，简称为 LHS，下文当中，如果没有特别指出，那么所说的 LHS 均指规则的条件部分，在一个规则当中 when 与 then 中间的部分就是 LHS 部分。在 LHS 当中，可以包含 $0 \sim n$ 个条件，如果 LHS 部分没空的话，那么引擎会自动添加一个 `eval(true)` 的条件，由于该条件总是返回 `true`，所以 LHS 为空的规则总是返回 `true`。

```
rule "name"
  when
    eval(true)
  then
  end
```

LHS 部分是由一个或多个条件组成，条件又称之为 pattern（匹配模式），多个 pattern 之间用可以使用 `and` 或 `or` 来进行连接，同时还可以使用小括号来确定 pattern 的优先级

[绑定变量名:]Object([field 约束])

对于一个 pattern 来说“绑定变量名”是可选的，如果在当前规则的 LHS 部分的其它的 pattern 要用到这个对象，那么可以通过为该对象设定一个绑定变量名来实现对其引用，对于绑定变量的命名，通常的作法是为其添加一个“\$”符号作为前缀，这样可以很好的与 Fact 的属性区别开来；绑定变量不仅可以用在对象上，也可以用在对象的属性上面，命名方法与对象的命名方法相同；“field 约束”是指当前对象里相关字段的条件限制，

```
rule "rule1"
  when
    $customer:Customer()
  then
    <action>...
  end
```

规则中 LHS 部分单个 pattern（模式）的情形。

规则中“\$customer”是就是一个绑定到 Customer 对象的“绑定变量名”，该规则的 LHS 部分表示，要求 Fact 对象必须是 Customer 类型，该条件满足了那么它的 LHS 会返回 `true`。

下面这种写法就是包含两种 pattern（模式）：

```
rule "rule1"
  when
    $customer:Customer(age>20,gender=='male')
    Order(customer==$customer,price>1000)
  then
    ....
```

```
end
```

简单说明一下上面的代码

第一个: pattern(模式) 有三个约束

- 1、对象 类型必须是 Cutomer;
- 2、Cutomer 的 age 要大于 20
- 3、Cutomer 的 gender 要是 male

第二个:pattern(模式) 有三个约束

- 1、对象类型必须是 Order;
- 2、Order 对应的 Cutomer 必须是前面的那个 Customer
- 3、当前这个 Order 的 price 要大于 1000

这两个 pattern 没有符号连接, 在 Drools 当中在 pattern 中没有连接符号, 那么就用 and 来作为默认连接, 所以在该规则的 LHS 部分 中两个 pattern(模式) 只有都满足了才会返回 true。默认情况下, 每行可以用 “;” 来作为结束符 (和 Java 的结束一样), 当然行尾也可以不加 “;” 结尾。

约束连接

对于对象内部的多个约束的连接, 可以采用 “&&” (and)、 “||” (or) 和 “,” (and) 来实现

这三个连接符号如果没有用小括号来显示的定义优先级的话, 那么它们的执行顺序是:

“&&” (and)、 “||” (or)

表面上看 “,” 与 “&&” 具有相同的含义, 但是有一点需要注意, “,” 与 “&&” 和 “||” 不能混合使用, 也就是说在有 “&&” 或 “||” 出现的 LHS 当中, 是不可以有 “,” 连接符出现的, 反之亦然

Drools 提供了十二中类型比较操作符: 如果进行常量比较, 必须通过 eval(条件) 或者对象引用比较对象属性, 不能单独使用, 这语法与 java 是一样的

>|<, >=|<=, ==|!= 这几个不多说啦

contains | not contains

memberOf | not memberOf

matches | not matches

下面说明一下后面 6 种的含意

contains: 比较操作符 contains 是用来检查一个 Fact 对象的某个字段 (该字段要是是一个 Collection 或是一个 Array 类型的对象) 是否包含一个指定的对象

语法格式:

```
Object( field[Collection/Array] contains value)
```

Contains.drl 写法

```
package rules.testwrod
import com.drools.test.Person
```

```

import com.drools.test.School
rule test001
    when
        $s:School();
        $p:Person(name contains $s.name);
    then
        System.out.println("恭喜你，成功的使用了 contains");
    end

```

JavaAPI 写法

```

package com.drools.test;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.FactHandle;
public class TestWrod{
    public static void main(String[] args){
        KieServices kss = KieServices.Factory.get();
        KieContainer kc = kss.getKieClasspathContainer();
        KieSession ks =kc.newKieSession("session");
        School school=new School();
        school.setCount(50);
        school.setName("一班");
        Person person=new Person("一班",30);
        FactHandle insert = ks.insert(person);
        ks.insert(school);
        int count = ks.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        ks.dispose();
    }
}

```

contains 只能用于对象的某个 Collection/Array 类型的字段与另外一个值进行比较，作为比较的值可以是一个静态的值，也可以是一个变量(绑定变量或者是一个 global 对象)，说的可能有点麻烦，小编在这里给大家再通俗的说一下，其实 contains 就是用来比较属性值是否与被比较值相同，但是这两个属性名是相同的。

not contains: not contains 作用与 contains 作用相反，not contains 是用来判断一个 Fact 对象的某个字段（Collection/Array 类型）是不是包含一个指定的对象，和 contains 比较符相同，它也只能用在对象的 field 当中，举例说明

Contains.drl 写法

```

package rules.testwrod
import com.drools.test.Person
import com.drools.test.School

```

```

rule test001
    when
        $s:School();
        $p:Person(age not contains $s.count);
    then
        System.out.println("恭喜你，成功的使用了 not contains");
    end
end

```

JavaAPI 写法

```

package com.drools.test;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.FactHandle;
public class TestWrod{
    public static void main(String[] args){
        KieServices kss = KieServices.Factory.get();
        KieContainer kc = kss.getKieClasspathContainer();
        KieSession ks =kc.newKieSession("session");
        School school=new School();
        school.setCount(50);
        school.setName("一班");
        Person person=new Person("一班",30);
        FactHandle insert = ks.insert(person);
        ks.insert(school);
        int count = ks.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        ks.dispose();
    }
}

```

结果一定是我们想要的，是因为 Person 中的属性是 age 而 School 的属性是 count，就算值相同，属性名不同 在 not contains 下是成立的。

memberOf: memberOf 是用来判断某个 Fact 对象的某个字段是否在一个集合 (Collection/Array) 当中，用法与 contains 有些类似，但也有不同

memberOf 的语法如下：

```
Object(fieldName memberOf value[Collection/Array])
```

可以看到 memberOf 中集合类型的数据是作为被比较项的，集合类型的数据对象位于 memberOf 操作符后面，同时在使用 memberOf 比较操作符时被比较项一定要是一个变量 (绑定变量或者是一个 global 对象)，而不能是一个静态值。如何给全局变量赋值:ksession.setGlobal("list",list);举例说明

memberOf.drl 文件

```

package rules.testwrod

import com.drools.test.Person

global java.util.List list;
rule test001
    when
        $p:Person(name memberOf list);
    then
        System.out.println("恭喜你，成功的使用了 memberOf");
    end

```

JavaAPI 写法

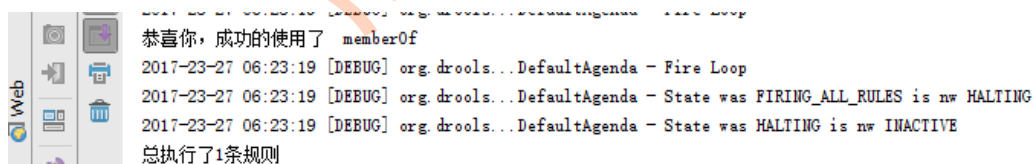
```

package com.drools.test;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.FactHandle;
public class TestWrod{
    public static void main(String[] args){
        KieServices kss = KieServices.Factory.get();
        KieContainer kc = kss.getKieClasspathContainer();
        KieSession ks =kc.newKieSession("session");
        List list=new ArrayList();
        list.add("张三");
        list.add("李四");
        list.add("王五");
        list.add("赵六");
        Person person=new Person("张三",50);
        ks.setGlobal("list",list);
        FactHandle insert = ks.insert(person);
        ks.insert(school);
        int count = ks.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        ks.dispose();
    }
}

```

结果肯定是我们想要的了



not memberOf: 该操作符与 `memberOf` 作用恰恰相反，是用来判断 Fact 对象当中某个字段值是不是中某个集合（Collection/Array）当中。小编这里就不给读者举例说明，有兴趣的读者可以自己尝试一下。

matches: `matches` 是用来对某个 Fact 的字段与标准的 Java 正则表达式进行相似匹配，被比较的字符串可以是一个标准的 Java 正则表达式，有一点小编要提醒读者注意，那就是正则表达式字符串当中不用考虑 “\” 的转义问题。

语法如下：

```
Object(fieldName matches “正则表达式”)
```

举例说明

Matches.drl

```
package rules.testwrod

import com.drools.test.Person

rule test001
    when
        $p:Person(name matches "张.*");
    then
        System.out.println("恭喜你，成功的使用了 matches");
    end
```

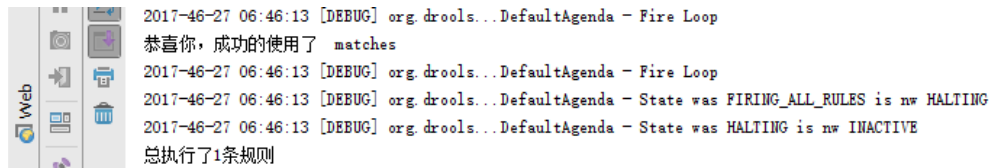
JavaAPI 写法

```
package com.drools.test;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.FactHandle;

public class TestWrod{
    public static void main(String[] args){
        KieServices kss = KieServices.Factory.get();
        KieContainer kc = kss.getKieClasspathContainer();
        KieSession ks =kc.newKieSession("session");
        Person person=new Person("张三",30);
        FactHandle insert = ks.insert(person);
        ks.insert(school);
        int count = ks.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        ks.dispose();
    }
}
```

结果如下：



小编这里补充几点：简单来说就是模糊查询，语法是不是“*.三”这样的语法是错误的。

该规则是用来查找所有 Person 对象的 name 属性是不是以“张”字开头，如果满足这一条件那么就将该 Person 对象的 name 属性打印出来

not matches:与 matches 作用相反，是用来将某个 Fact 的字段与一个 Java 标准正则表达式进行匹配，看是不是能与正则表达式匹配。 小编这里就不多写做例子了，建议读者要自己试一下，体验一下结果

语法扩展部分

访问 List 数据结构

`$customer.accounts[3]`等同于`$customer.getAccounts(3)`

访问 Map 数据结构

`$customerMap["123"]`等同于`$customerMap.get("123")`

Drools 属性说明

要说到 Rule 的属性，那就有必须详细的说明一下了，属性首先是有优先级的，而且每种属性的搭配很重要，如果你不理解这些属性，很可能出各种问题，小编在研究的时候对这些属性做了详细的说明，下面我们就从简单到复杂的一个一个说明

规则的属性共有 12 个，

它们分别是：

activation-group、agenda-group、auto-focus、date-effective、date-expires、dialect、duration、enabled、lock-on-active、no-loop、ruleflow-group、salience
这些属性分别适用于不同的场景，下面我们就来分别介绍 这些属性的含义及用法。

Salience 优先级

作用是用来设置规则执行的**优先级**，salience 属性的值是一个数字，数字越大执行优先级越高，同时它的值可以是一个负数。默认情况下，规则的 salience 默认值为 0，所以如果我们不手动设置规则的 salience 属性，那么如果不设置它的执行顺序则规则是随机的。

看下以下代码：

```
package rules.testwrod
```




```

rule test001
salience 2
    when
    then
        System.out.println("执行 test001");
    end

rule test002
salience 1
    when
    then
        System.out.println("执行 test002");
    end
end

```

执行的结果:



```

执行test001
2017-31-28 10:31:12 [DEBUG] org.drools...DefaultAgenda - Fire Loop
执行test002
2017-31-28 10:31:12 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-31-28 10:31:12 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-31-28 10:31:12 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
总执行了2条规则

```

no-loop 防止死循环

在一个规则当中如果条件满足就对 Working Memory 当中的某个 Fact 对象进行了修改，比如使用 update 将其更新到当前的 Working Memory 当中，这时引擎会再次检查所有的规则是否满足条件，如果满足会再次执行，可能出现死循环的。如何避免这种情况呢，这时可以引入 no-loop 属性来解决这个问题。no-loop 属性的作用是用来控制已经执行过的规则在条件再次满足时是否再次执行，no-loop 属性的值是一个布尔型，默认情况下规则的 no-loop 属性的值为 false，如果 no-loop 属性值为 true，那么就表示该规则只会被引擎检查一次，如果满足条件就执行规则的 RHS 部分

注意：如果引擎内部因为对 Fact 更新引起引擎再次启动检查规则，那么它会忽略掉所有的 no-loop 属性设置为 true 的规则。

看下以下代码：

```

package rules.testwrod

import com.drools.test.Person

rule test001
//no-loop true
    when
        $p:Person(name=="张三");
    then
        $p.setAge(50);
        update($p)
    end
end

```

```
System.out.println("不设置 no-loop 时的效果");
end
```

执行的结果:



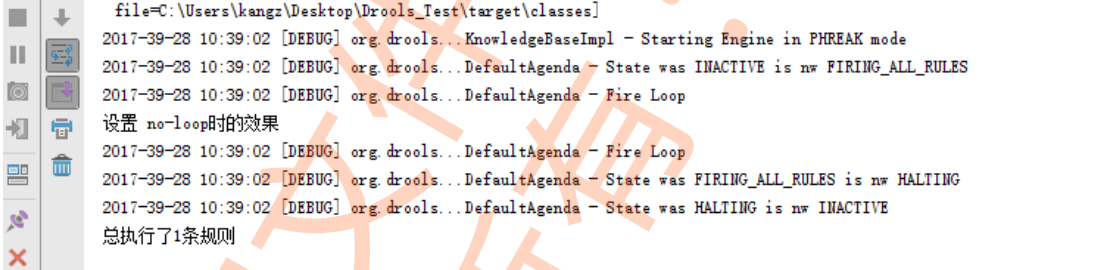
看下以下代码:

```
package rules.testwrod

import com.drools.test.Person

rule test001
no-loop true
    when
        $p:Person(name=="张三");
    then
        $p.setAge(50);
        update($p)
        System.out.println("设置 no-loop 时的效果");
    end
end
```

执行的结果:



以下情况 就算是设置了 no-loop 为 true 也会出现死循环

看下以下代码:

```
package rules.testwrod

import com.drools.test.Person

rule test001
no-loop true
    when
        $p:Person(name=="张三");
    then
        $p.setAge(50);
        update($p)
        System.out.println("设置 no-loop 时的效果");
    end
end
```

```

rule test002
no-loop true
    when
        $p:Person(age==50);
    then
        $p.setName("张三");
        update($p)
        System.out.println("设置 no-loop 时的效果");
    end
end

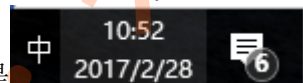
```

date-effective 日期比较小于等于

该属性是用来控制规则只有在到达后才会触发，在规则运行时，引擎会自动拿当前操作系统的时候与 date-effective 设置的时间值进行比对，只有**当前系统时间**>=date-effective 设置的时间值时，规则才会触发执行，否则执行将不执行。在没有设置该属性的情况下，规则随时可以触发，没有这种限制。

date-effective 的值为一个日期型的字符串，默认情况下，date-effective 可接受的日期格式为“dd-**MMM**-yyyy”，例如 2017 年 02 月 28 日在设置为 date-effective 的值时，如果您的操作系统为英文的，那么应该写成“28- February-2017”；如果是中文

操作系统“28-二月-2017”举例说明：注当前日期是



看下以下代码：

```

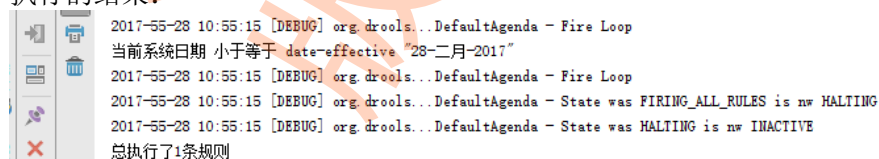
package rules.testwrod

import com.drools.test.Person

rule test001
    date-effective "28-二月-2017"
    when
        eval(true)
    then
        System.out.println("当前系统日期 小于等于 date-effective \"28-二月-2017\"");
    end
end

```

执行的结果：



其实也可以修改成这样

date-effective "50-二月-2017" 注：这样是不报错的，当你去测试时神奇的发现，这样写是会当前日期进行累加。

date-expires 日期比较大于

该属性的作用与 date-effective 属性恰恰相反，当前系统时间 < date-expires 值，date-expires 的作用是用来设置规则的有效期，引擎在执行规则的时候，会检查规则有没有 date-expires 属性，如果有的话，那么会将这个属性的值与当前系统时间进行比对，如果大于系统时间，那么规则就执行，否则就不执行。该属性的值同样也是一个日期类型，默认格式也是 “dd-MMM-yyyy”，具体用法与 date-effective 属性相同。

如何写成 java 传统的日期格式呢 System.setProperty("drools.dateformat", "yyyy-MM-dd HH:mm:ss"); 时分秒可加可不加
但要注意的是，在进行规则引擎格式化日期的时候，最好是将其写在初始化 kie 相关代码之前

```
public void ruleDasTdest() throws Exception{
    System.setProperty("drools.dateformat", "yyyy-MM-dd"); 正确的
    KieServices ks = KieServices.Factory.get();
    KieContainer kc = ks.getKieClasspathContainer();
    KieSession ksession=kc.newKieSession("Rulebasedrl-session");
    System.setProperty("drools.dateformat", "yyyy-MM-dd"); 错误的
    Person p=new Person("张三",10);
}
```

这里要注意一下，如果是整合 spring 时，规则中要使用的话，则先在静态块修改 drools 默认的时间格式，java 的执行方式，先静后动，先父后子

```
static {
    // 修改 drools 默认识别的时间格式
    System.setProperty("drools.dateformat", "yyyy-MM-dd");
}
```

Dialect 方言

该属性用来定义规则当中要使用的语言类型，在 drools5 与 drools6.4 版本中，支持两种类型的语言 mvel 和 java，官方举例也是按这两种类型的语言举例说明的。默认情况下，如果没有手工设置规则的 dialect，那么使用的 java 语言。在特殊情况下会用到。比如 **accumulate** 时

Enabled 是否可用

enabled 属性比较简单，它是用来定义一个规则是否可用的。该属性的值是一个布尔值，默认该属性的值为 true，表示规则是可用的，如果手工为一个规则添加一个 enabled 属性，并且设置其 enabled 属性值为 false，那么引擎就不会执行该规则。

Duration 定时器（经被淘汰）

对于一个规则来说，如果设置了该属性，那么规则将在该属性指定的值之后在另外一个线程里触发。该属性对应的值为一个长整型，单位是毫秒 通过 mian 方法测试

lock-on-active 规则只执行一次

当在规则上使用 ruleflow-group 属性或 agenda-group 属性的时候，将 lock-on-action 属性的值设置为 true，可能避免因某些 Fact 对象被修改而使已经执行过的规则再次被激活执行。可以看出该属性与 no-loop 属性有相似之处，no-loop 属性是为了避免 Fact 修改或调用了 insert、retract、update 之类而导致规则再次激活执行，这里的 lock-on-action 属性也是起这个作用，但 lock-on-active 是 no-loop 的增强版属性，它主要作用在使用 ruleflow-group 属性或 agenda-group 属性的时候。lock-on-active 属性默认值为 false。设置为 true 值后，该规则只被执行一次。

activation-group 分组

该属性的作用是将若干个规则划分成一个组，用一个字符串来给这个组命名，这样在执行的时候，**具有相同 activation-group 属性的规则中只要有一个会被执行**，其它的规则都将不再执行。也就是说，在一组具有相同 activation-group 属性的规则当中，只有一个规则会被执行，其它规则都将不会被执行。当然对于具有相同 activation-group 属性的规则当中究竟哪一个会先执行，则可以用 salience 之类属性来实现。

agenda-group 议程分组

规则的调用与执行是通过 StatelessSession 或 ksession 来实现的，一般的顺序是创建一个 StatelessSession 或 ksession，将各种经过编译的规则 package 添加到 session 当中，接下来将规则当中可能用到的 Global 对象和 Fact 对象插入到 Session 当中，最后调用 fireAllRules 方法来触发、执行规则。在没有调用最后一步 fireAllRules 方法之前，所有的规则及插入的 Fact 对象都存放在一个名叫 Agenda 表的对象当中，这个 Agenda 表中每一个规则及与其匹配相关业务数据叫做 Activation，在调用 fireAllRules 方法后，这些 Activation 会依次执行，这些位于 Agenda 表中的 Activation 的执行顺序在没有设置相关用来控制顺序的属性时（比如 salience 属性），它的执行顺序是随机的，不确定的。

Agenda-group 是用来在 Agenda 的基础之上，对现在的规则进行再次分组，具体的分组方法可以采用为规则添加 agenda-group 属性来实现，agenda-group 属性的值也是一个字符串，通过这个字符串，可以将规则分为若干个 Agenda Group，默认情况下，引擎在调用这些设置了 agenda-group 属性的规则的时候需要显示的指定某个 Agenda Group 得到 Focus（焦点），这样位于该 Agenda Group 当中的规则才会触发执行，否则将不执行。

看下以下代码：

```
AgendaGroup. drl
```

```

package rules.testwrod

import com.drools.test.Person

rule test001
    agenda-group "test001"
    when
        eval(true)
    then
        System.out.println("test001 焦点");
    end
end

```

Java 代码

```

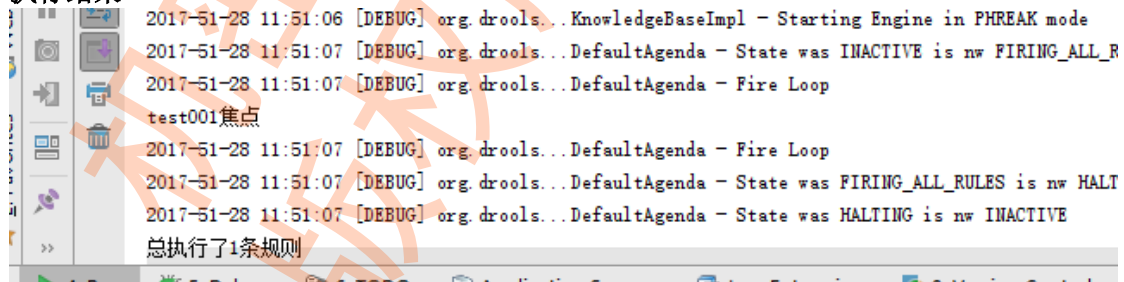
package com.drools.test;

import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.rule.FactHandle;

public class TestWrod{
    public static void main(String[] args){
        KieServices kss = KieServices.Factory.get();
        KieContainer kc = kss.getKieClasspathContainer();
        KieSession ks =kc.newKieSession("session");
        ks.getAgenda().getAgendaGroup("test001").setFocus();
        int count = ks.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        ks.dispose();
    }
}

```

执行结果



```

2017-51-28 11:51:06 [DEBUG] org.drools...KnowledgeBaseImpl - Starting Engine in PHREAK mode
2017-51-28 11:51:07 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_R
2017-51-28 11:51:07 [DEBUG] org.drools...DefaultAgenda - Fire Loop
test001焦点
2017-51-28 11:51:07 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-51-28 11:51:07 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALT
2017-51-28 11:51:07 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
总执行了1条规则

```

实际应用当中 agenda-group 可以和 auto-focus 属性一起使用，这样就不会在代码当中显示的为某个 Agenda Group 设置 setFocus 了。一旦将某个规则的 auto-focus 属性设置为 true，那么即使该规则设置了 agenda-group 属性，我们也不需要在代码当中显示的设置 Agenda Group 的 setFocus 了。

注：当加入 activation-group 时同一组的 Focus 相同时，只会执行其中一个。执行方式从上至下。

同一组的 Focus 不同时，只会执行其中一个。执行方式是最后一个获取焦点的规则

不同组的 Focus 相同时，会执行多个规则，但只要获取 Focus 的才会执行

不同组的 Focus 不同时，会执行多个，但执行方式是从最后一个获取焦点的规则开始

auto-focus 焦点分组

前面我们也提到 auto-focus 属性，它的作用是用来在已设置了 agenda-group 的规则上设置该规则是否可以自动读取 Focus，如果该属性设置为 true，那么在引擎执行时，就不需要显示的为某个 Agenda-Group 设置 Focus，否则需要。但将 auto-focus 设置成 true 再通过 agenda-group 指定 Focus 则 agenda-group 不生效。如果在规则中加入 activation-group 时则会进行分组，但结果会根据 auto-focus 及 when 条件进行判定。

对于规则的执行的控制，还可以使用 Agenda Filter 来实现。在 Drools 当中，提供了一个名为 org.drools.runtime.rule.AgendaFilter 的 Agenda Filter 接口，用户可以实现该接口，通过规则当中的某些属性来控制规则要不要执行。

org.drools.runtime.rule.AgendaFilter 接口只有一个方法需要实现
方法体如下：

```
public boolean accept(Match match);
```

在该方法当中提供了一个 Match 参数，通过该参数我们可以得到当前正在执行的规则对象或其它一些属性，该方法要返回一个布尔值，该布尔值就决定了要不要执行当前这个规则，返回 true 就执行规则，否则就不执行。

AgendaFilter.drl

```
package rules.testwrod

rule test001
  auto-focus true
  when
    eval(true)
  then
    System.out.println("test001 焦点 auto-focus");
end
```

Java 代码

```
package com.drools.test;

import org.kie.api.runtime.rule.AgendaFilter;
import org.kie.api.runtime.rule.Match;

/**
 * Created by kangz on 2017/2/28.
 */
```



```

public class Testaccept implements AgendaFilter {
    private String ruleName;
    public Testaccept(String ruleName) {
        this.ruleName = ruleName;
    }
    @Override
    public boolean accept(Match match) {
        String name=match.getRule().getName();
        if(name.startsWith(ruleName)){
            return true;
        }else{
            return false;
        }
    }
}

```

实现方法 API 操作

```

@Test
public void Test001() {
    KieServices kss = KieServices.Factory.get();
    KieContainer kc = kss.getKieClasspathContainer();
    KieSession ks =kc.newKieSession("session");
    AgendaFilter filter=new Testaccept("test001");
    int count = ks.fireAllRules(filter);
    ks.dispose();
}

```

ruleflow-group 规则流

在使用规则流的时候要用到 ruleflow-group 属性，该属性的值为一个字符串，作用是用来将规则划分为一个个的组，然后在规则流当中通过使用 ruleflow-group 属性的值，从而使用对应的规则，该属性会通过流程的走向确定要执行哪一条规则。在规则流中有具体的说明。

Drools drl 注释的使用

单行注释

单行注释可以采用“#”或者“//”来进行标记，

```
//规则rule1的注释
rule "rule1"
    when
        eval(true) #没有条件判断
    then
        System.out.println("rule1 execute");
    end
```

多行注释

如果要注释的内容较多，可以采用 Drools 当中的多行注释标记来实现。Drools 当中的 多行注释标记与 Java 语法完全一样，以“/*”开始，以“*/”结束。

```
/*
规则rule1的注释
这是一个测试用规则
*/
rule "rule1"
    when
        eval(true) #没有条件判断
    then
        System.out.println("rule1 execute");
    end
```

Drools 函数的使用详述

insert 插入

函数 insert 的作用与我们在 Java 类当中调用 KieSession 对象的 insert 方法的作用相同，都是用来将一个 Fact 对象插入到当前的 Working Memory 当中
基本用法格式如下：

```
insert(new Object());
```

一旦调用 insert 宏函数，那么 Drools 会重新与所有的规则再重新匹配一次，对于没有设置 no-loop 属性为 true 的规则，如果条件满足，不管其之前是否执行过都会再执行一次，这个特性不仅存在于 insert 宏函数上，后面介绍的 update/ modify、retract 宏函数同样具有该特性，所以在某些情况下因考虑不周调用 insert、update/ modify 或 retract/delete 容易发生死循环

Insert.drl

```
package rules.testwrod

import com.drools.test.Person
```

```

rule test002
    when
        $p:Person(name=="张三");
    then
        System.out.println("新添加的 Person"+$p.getName());
    End

rule test001
    when
        eval(true)
    then
        Person person=new Person();
        person.setName("张三");
        person.setAge(20);
        insert(person);
    end

```

JAVA 代码

```

@Test
public void Test001() {
    KieServices kss = KieServices.Factory.get();
    KieContainer kc = kss.getKieClasspathContainer();
    KieSession ks =kc.newKieSession("session");
    int count = ks.fireAllRules();
    ks.dispose();
}

```

执行结果

```

2017-50-28 02:50:47 [DEBUG] org.drools...DefaultAgenda - Fire Loop
新添加的Person张三
2017-50-28 02:50:47 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-50-28 02:50:47 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-50-28 02:50:47 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
总执行了2条规则

```

代码中并在添加 insert，所以在规则中添加也是正确的，如果结果没有出来，就可能是因为执行的顺序有关，可以设置优先级最好加上 no-loop 的属性

insertLogical 插入

insertLogical 作用与 insert 类似，它的作用也是将一个 Fact 对象插入到当前的 Working Memory 当中

update 修改

update 函数意义与其名称一样，用来实现对当前 Working Memory 当中的 Fact 进行更新， update 宏函数的作用与 StatefulSession 对象的 update 方法的作用基本相同，

都是用来告诉当前的 Working Memory 该 Fact 对象已经发生了变化。它的用法有两种形式，一种是直接更新一个 Fact 对象，另一种为通过指定 FactHandle 来更新与指定 FactHandle 对应的 Fact 对象，下面我们就来通过两个实例来说明 update 的这两种用法。

先来看第一种用法，直接更新一个 Fact 对象。

第一种用法的格式如下：

```
update(Object());
```

Person.drl

```
package rules.testwrod

import com.drools.test.Person
rule test001
    when
        $p:Person(name=="张三",age==30);
    then
        $p.setName("李四");
        update($p)
        System.out.println("改完后的名字"+$p.getName());
    end
```

执行后的结果

```
2017-31-28 03:31:12 [DEBUG] org.drools...DefaultAgenda - Fire Loop
修改后的结果为李四
2017-31-28 03:31:12 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-31-28 03:31:12 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-31-28 03:31:12 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
总执行了1条规则
```

还有一种方式，也是特别常用的

```
package rules.testwrod

import com.drools.test.Person
rule test001
    when
        $p:Person(name=="张三",age==30);
    then
        modify(){
            $p.setName("李四");
        }
        System.out.println("改完后的名字"+$p.getName());
    end
```

第二种用法的格式如下：不常用，写的很麻烦

```
update(new FactHandle(),new Object());
```

```

3 import com.drool.test.Person;
4 rule "rule1"
5     salience 2
6     when
7         eval(true);
8     then
9         Person person =new Person();
10        person.setName("张三");
11        person.setAge(1);
12        insert(person);
13        System.out.println("1111111111");
14 end
15
16 rule "rule2"
17     salience 1
18     when
19         $p:Person(name=="张三",age<10);
20     then
21         Person person =new Person();
22         person.setName("张三");
23         person.setAge(1);
24         $p.setAge($p.getAge()+1);
25         update(drools.getWorkingMemory().getFactHandleByIdentity($p),person);
26         System.out.println("rule2----"+$p.getName());
27 end
28

```

这是小编之前做的例子，因为不常用，所以这里只是提一下，读者做为了解即可。

retract 删除功能

和 kession 的 retract/delete 方法一样，宏函数 retract/delete 也是用来将 Working Memory 当中 某个 Fact 对象从 Working Memory 当中删除，下面就通过一个例子来说明 retract 宏函数的用法

Delete.drl

```

package rules.testwrod

import com.drools.test.Person

rule test002
salience 2
    when
        $p:Person(name=="张三",age==50);
    then
        retract($p);
        System.out.println("修改后的结果为"+$p.getName());
    end

rule test001
salience 1
    when
        $p:Person();
    then

```

```
System.out.println("删除后的结果为"+$p.getName());
```

```
end
```

执行后的结果

```
2017-45-28 03:45:29 [DEBUG] org.drools...DefaultAgenda - Fire Loop
修改后的结果为张三
2017-45-28 03:45:29 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-45-28 03:45:29 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-45-28 03:45:29 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-45-28 03:45:29 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
张三
总执行了1条规则
```

我们发现，第二个规则并没有执行，所以删除成功

```
retract($p);可改成 delete($p)
```

drools 宏对象和 java 中的类一样，可以通过 drools 来获取里面的属性方法。

想要在规则文档里获取更多的实现对当前的 Working Memory 控制，那么可以使用 drools 宏对象实现，通过使用 drools 宏对象可以实现在规则文件里直接访问 Working Memory。在前面介绍 update 宏函数的时候我们就使用 drools 宏对象来访问当前的 Working Memory，得到一个指定的 Fact 对象的 FactHandle。同时前面介绍的 insert、insertLogical、update 和 retract 宏函数的功能皆可以通过使用 drools 宏对象来实现。

举例说明：

普通宏函数

```
rule "rule1"
  salience 11
  when
    eval(true);
  then
    Customer cus=new Customer();
    cus.setName("张三");
    insert(cus);
  end
```

drools 宏函数

```
rule "rule1"
  salience 11
  when
    eval(true);
  then
    Customer cus=new Customer();
    cus.setName("张三");
    drools.insert(cus)
  end
```

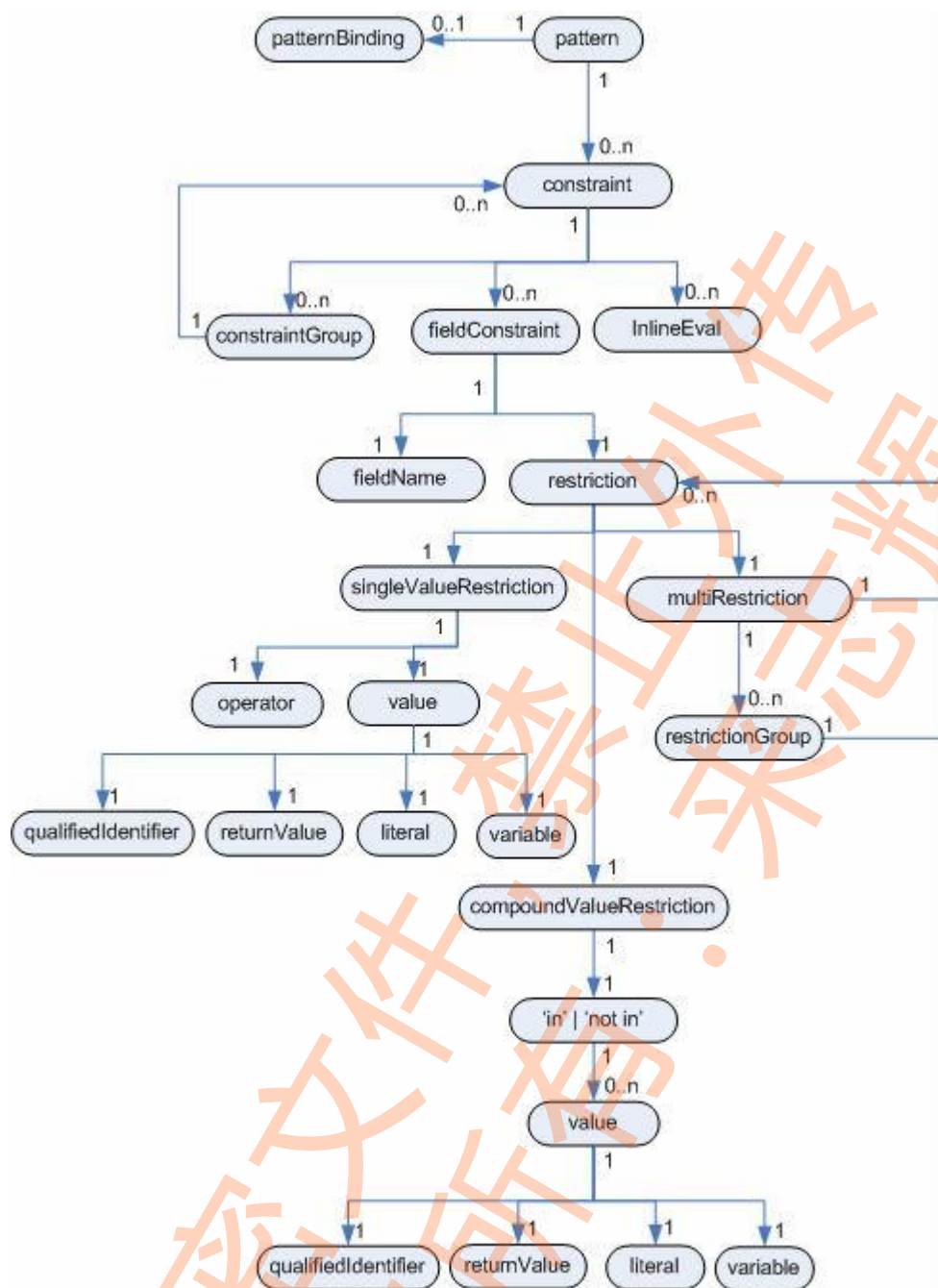
两个规则写法不同，但实现的效果是一样的。

以下是 drools 常用的方法

方法名称	用法格式	含义
getWorkingMemory()	drools.getWorkingMemory()	获取当前的 WorkingMemory 对象
halt()	drools.halt()	在当前规则执行完成后，不再执行其它未执行的规则。
getRule()	drools.getRule()	得到当前的规则对象
insert(new Object)	drools.insert(new Object)	向当前的 WorkingMemory 当中插入指定的对象，功能与宏函数 insert 相同。
update(new Object)	drools.update(new Object)	更新当前的 WorkingMemory 中指定的对象，功能与宏函数 update 相同。
update(FactHandle Object)	drools.update(FactHandle Object)	更新当前的 WorkingMemory 中指定的对象，功能与宏函数 update 相同。
retract(new Object)	drools.retract(new Object)	从当前的 WorkingMemory 中删除指定的对象，功能与宏函数 retract 相同。

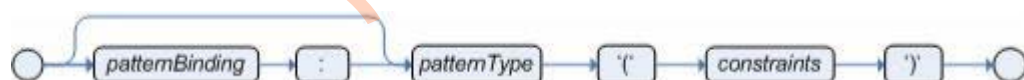
Drools 语法篇之左手边（when）

书写规范，不能使用输入法全角
模式元素是最重要的条件元素。下面的整体关系图表，勾画出了构成模式的约束的各个部分，以及它们如何一起工作；然后用铁路图表和实例更详细地介绍了每一个。



模式整体关系图表

在 ER 图表的顶层，你可以看见该模式由零个或多个约束构成，并且有一个可选择的模式绑定。下面的铁路图显示了它的语法



Pattern 模式

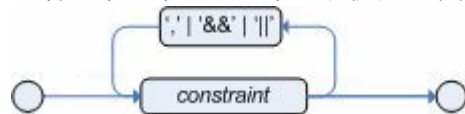
在它的最简单格式中，没有约束，一个模式匹配给定类型的一个事实。在下面的案例中，类型为 Cheese，这意味着模式会匹配在工作内存中的所有 Cheese 对象。

注意类型不必是一些事实对象的实际的类。模式可以引用子类，甚至接口，因此匹配的事实可能来自不同的类。

为了引用匹配的对象，一个模式可以绑定变量，如 \$c 前缀符号 (\$) 是可选的；在复杂的规则中可能非常有用，它有助于区别变量和字段。

```
$c : Cheese()
```

在模式的扩号内部是所有动作发生的地方。约束可以是一个字段约束，内嵌计算 (Inline Eval)，或者一个约束组。约束可以使用：',', '&&' or '||' 符号分隔。



字段约束

字段约束为命名字段指定限制；字段名可以有一个可选的变量绑定。有三种类型的限制：单值限制、复合值限制和多限制。

JavaBeans 作为事实

字段从对象的访问方法派生。如果你的模型对象遵守 Java Bean 模式，那么通过 "getXXX" 或 "setXXX" 方法暴露字段，这些方法都是无参返回东西。在模式内部，可以使用 bean 命名协议访问字段，所以，"getType" 将作为 "type" 被访问。Drools 使用标准的自省 (Introspector) 类做这种映射。

例如，有关我们的 Cheese 类，模式 `Cheese(type == "brie")` 应用了一个 Cheese 实例的 getType() 方法。如果一个字段名没有发现，编译器将采用一个无参数的方法作为名字。因此，由于约束 `Cheese(toString == "cheddar")`，调用了 toString() 方法。在这种情况下，你使用正确的大小写方法全名，但是无括号。请确保你访问的方法没有使用参数，并且在事实访问器 (accessors) 中，它没有改变在某种程序上可能影响到规则的对象状态。记住规则引擎有效地缓存了匹配的结果，使它在调用之间更快。

限定标识符

也可以使用枚举，支持 jdk1.4 和 5 样式的枚举。对于后者，你必须使用一个 JDK 5 环境中执行。

布尔型字面文字限制

```
Cheese( smelly == SomeClass.TRUE )
```

绑定变量限制

变量可以绑定到事实或它们的字段，然后在推论字段约束中使用。一个绑定变量被称为一个声明。有效的运算符由被约束字段的类型确定；可以的地方会尝试强制。绑定变量限制使用 '=' 提供更快的执行，象我们可以使用散列 (hash) 索引提高性能一样。

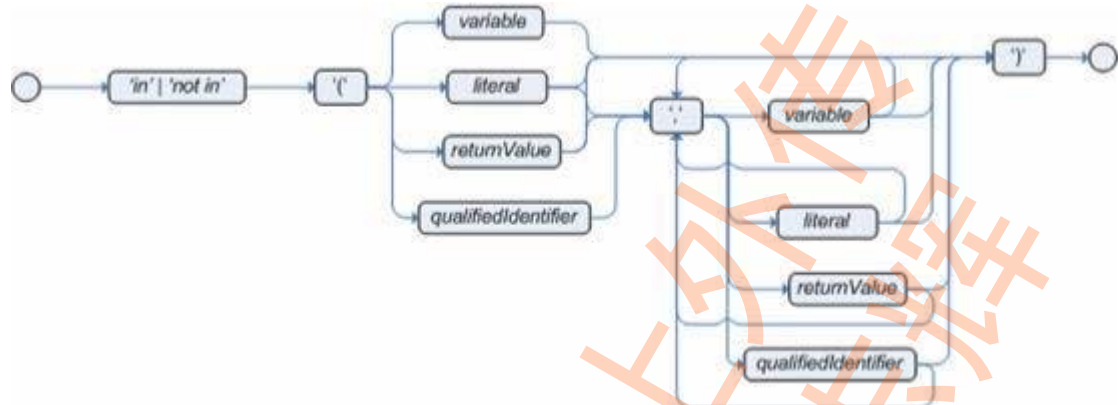
```
Person( likes : favouriteCheese )
```

```
Cheese( type == likes )
```

这里，likes 是绑定到它声明的任何 Person 实例的 favouriteCheese 字段的变量。它被用于约束随后模式中的 Cheese 的类型。可以使用任何有效的 Java 变量名，它可以使用一个前缀 '\$'，你常看见用它来区分字段声明。下面的例子，显示了一个 \$stilton 声明，绑定到第一个匹配的、并且使用了 contains 运算符的模式对象——注意，可选 '\$' 的使用。

复合值限制

复合值限制被用于超过一个可能的值匹配的地方。目前只支持 `in` 和 `no in` 运算符。这个运算符的第二个操作数必须是一个逗号分开的值列表，用括号括起来的。值可以用变量、字面文字、返回值或限定标识符提供的。这两个运算符实际是“语法糖（syntactic sugar）”，内部使用 `!='` 和 `=='` 运算符改写为多限制的一个列表。



使用 `in` 的复合值限制

```
Person( $cheese : favouriteCheese )  
Cheese( type in ( "stilton", "cheddar", $cheese )
```

多限制

简单的多限制使用 `&&`

```
Person( age > 30 && < 40 )/也可 “ , ”替换&&
```

复杂多约束使用分组多限制

```
Person( age ( (> 30 && < 40) || (> 20 && < 25) ) )
```

这里 `and` 或 `or` 就不多解释了。我们说一说 比较重要的一些 `when` 条件约束元素

条件元素 `eval`



条件元素 `eval` 本上是包罗万象的，它允许执行任何语义代码（返回一个 `boolean` 原型）。这个代码可以引用绑定在规则 LHS 中的变量，以及在规则包中的函数。滥用 `eval`，减少了你的规则的声明，可能导致引擎的性能较差。虽然 `eval` 可以用在模式中的任何地方，但是最佳作法是把它作为规则的 LHS 中的最后的条件元素添加。

`Evals` 没有被索引，因此不如字段约束有效果。然而，当函数返回随着时间推移而不允许在字段约束中使用的变化值时，使用 `evals` 比较理想。

条件元素 `not`



条件元素 `not` 是一阶逻辑的不存在 (non-existential) 量词, 用于检查在工作内存中不存在某东西。把“not”看作“一定没有……”的意思。

关键字 `not` 紧跟着用扩号扩起它所应用的条件元素。在单模式的最简情况下 (如下所示), 你可以选

择忽略扩号

`not Bus()`

条件元素 `exists`



条件元素 `exists` 是一阶逻辑的存在 (existential) 量词, 用于检查在工作内存中存在某东西。把“exists”看作“至少有一个……”的意思。

关键字 `exists` 紧跟着用扩号扩起它所应用的条件元素。在单模式的最简情况下 (如下所示), 你可以选择忽略扩号。

`exists Bus()`

`exists Bus(color == "red")`

// 括号是可选的:

`exists (Bus(color == "red", number == 42))`

// “exists ”与嵌套插入 括号是必需的:

`exists (Bus(color == "red") and Bus(color == "blue"))`

我们对上面的例子做一个简单的说明

`exists (Bus(color == "red", number == 42))` 很简单, 就是说 `Bus` 的 `color` 属性为 `red` `number` 属性为 `42` 当这两个条件都满足是, 返回值为 `true`

`exists (Bus(color == "red") and Bus(color == "blue"))` 这个是指 `color` 匹配多个值时, 当然 也要插入两个对象, 这两个对象里 一个存放 `red` 一个存入 `blue`。 `exists` 会去工作内存中查找对应对象的内的属性值, 进行对比。

条件元素 `forall`



在 `Dools` 中的条件元素 `forall` 完全支持一阶逻辑。当匹配第一个模式的所有事实匹配所有剩余模式时, 条件元素 `forall` 求值为 `true`。例如:

```
rule "All English buses are red"
  when
    forall( $bus : Bus( type == 'english')
      Bus( this == $bus, color = 'red' ) )
  then
    # all english buses are red
end
```

在上面的规则中，我们“选择”类型为"english"的所有 Bus 对象。然后，对每个匹配这个模式的事实，我们计算其随后的模式，如果为匹配，forall 条件元素求值为 true。要声明在工作内存中给定类型的所有事实必须匹配一组约束，可以用单个模式简明编写 forall。例如：

```
rule "All Buses are Red"
  when
    forall( Bus( color == 'red' ) )
  then
    # all asserted Bus facts are red
  end
```

另外一个例子，显示了 forall 内部的多个模式：

```
rule "all employees have health and dental care programs"
  when
    forall( $emp : Employee()
            HealthCare( employee == $emp )
            DentalCare( employee == $emp )
          )
  then
    # all employees have health and dental care
  end
```

为了完善的表现，forall 可以被嵌套在其他条件元素内部。例如，forall 可以被用在 not 条件元素内部。

注意，单个模式时扩号是可选的，所以，使用嵌套 forall，必须使用扩号：
forall 与 not 条件元素的组合

```
rule "not all employees have health and dental care"
  when
    not ( forall( $emp : Employee()
                 HealthCare( employee == $emp )
                 DentalCare( employee == $emp )
               )
  then
    # not all employees have health and dental care
  end
```

另外注意，not(forall(p1 p2 p3...))等价于：
not(p1 and not(and p2 p3...))

注意 forall 是一个域分隔符也很重要。因此，它可以使用任何前面绑定的变量，但是在它里面绑定的变量不能在它的外面使用

条件元素 from



条件元素 from 让用户指定任意的资源，用于 LHS 模式的数据匹配。这允许引擎在非工作内存数据的基础上进行推断。数据源可以是一个绑定变量的一个子字段，或者方法调用的结果。它是一个超强结构，允许开箱即可与其他应用程序组件或框架集成使用。常见的集成例

子是使用 hibernate 命名查询随时从数据库索取数据。用于定义数据源的表达式是任何遵守标准 MVEL 语法的表达式。它允许你轻松地使用对象属性导航，执行方法调用，以及访问映射和集合的元素。

下面是一个推断的例子，绑定了另一个模式的子字段：

```
rule "validate zipcode"
  when
    Person( $personAddress : address )
    Address( zipcode == "23920W") from $personAddress
  then
    # zip code is ok
  end
```

利用 Drools 新表现的所有灵活性，你可以用多种方式解决这个问题。下面是相同的，除显示你如何与'from'一起使用一个图形符号之外。

```
rule "validate zipcode"
  when
    $p : Person( )
    $a : Address( zipcode == "23920W") from $p.address
  then
    # zip code is ok
  end
```

前面的例子使用单个模式进行计算。条件元素 from 也支持对象源，其返回一个对象集合在这种情况下，from 将会迭代集合中的所有对象，并分别尝试匹配它们每一个。例如，如果我们希望一条规则，在一个定单中的每个项目实施 10%的折扣，我们可以做：

```
rule "apply 10% discount to all items over US$ 100,00 in an order"
  when
    $order : Order()
    $item : OrderItem( value > 100 ) from $order.items
  then
    # apply discount to $item
  end
```

在上面的例子中，对每个给定的定单，每个项目的值大于 100 时，将使规则引发一次。然而，在使用 from 时，你必须小心，特别是与 lock-on-active 规则属性联合使用时，因为它可能产生不希望的结果。

考虑前面提供的例子，现在可略加修改如下：

```
rule "Assign people in North Carolina (NC) to sales region 1"
  ruleflow-group "test"
  lock-on-active true
  when
    $p : Person( )
    $a : Address( state == "NC") from $p.address
  then
    modify ($p) {} #Assign person to sales region 1 in a modify block
  end
  rule "Apply a discount to people in the city of Raleigh"
  ruleflow-group "test"
```



```

lock-on-active true
when
    $p : Person( )
    $a : Address( city == "Raleigh") from $p.address
then
    modify ($p) {} #Apply discount to person in a modify block
end

```

在上面的例子中，销售 1 区为北卡罗莱纳州（NC），在罗利（Raleigh）城中的人，给予优惠；即你希望两条规则都被激活且被引发。相反，你会发现只有第二条规则被引发了。如果你打开了审计日志，你将会看见，在第二条规则引发时，它禁用了第一条规则。因为当一组事实变化时，lock-on-active 规则属性防止规则创建新的活动，所以第一条规则未能被重新激活。虽然该组事实没有被改变，而实际上每次计算使用的 from 都会产生一个新的事实。

首先，检查你为什么使用上面的模式是重要的。你可能有多条规则横跨不同的规则流（rule-flow）组。当你修改工作内存时，在你的规则流下游的其他规则（在不同规则流组中的）需要被重新计算，modify 的使用至关重要。然而，你不希望在相同规则流组中的其他规则在一个另外的递归中被设置为活动。在这种情况下，no-loop 属性是无效的，因为它只防止规则的自身递归激活。因此，你求助于 lock-on-active。

有几种方法解决这个问题：

- 1、当你可以断言所有事实到工作内存中，或者在你的约束表达式中使用嵌套对象引用时（下面所示），应避免使用 from
- 2、把用于修改块中的变量的赋值语句作为你的条件（LHS）中的最后一条放置
- 3、当你可以显式地管理在同一规则流组内的规则在另外一个中如何设置活动时，应避免使用 lock-on-active

当你可以直接断言所有事实到工作内存时，首选解决方案是尽量减少 from 的使用。在上面的例子中，Person 和 Address 实例都可以被断言到工作内存中。在这种情况下，因为该图相当简单，更容易的解决方案是如下这样修改你的规则

```

rule "Assign people in North Carolina (NC) to sales region 1"
    ruleflow-group "test"
    lock-on-active true
    when
        $p : Person(address.state == "NC" )
    then
        modify ($p) {} #Assign person to sales region 1 in a modify block
    end
end
rule "Apply a discount to people in the city of Raleigh"
    ruleflow-group "test"
    lock-on-active true
    when
        $p : Person(address.city == "Raleigh" )
    then
        modify ($p) {} #Apply discount to person in a modify block
    end
end

```

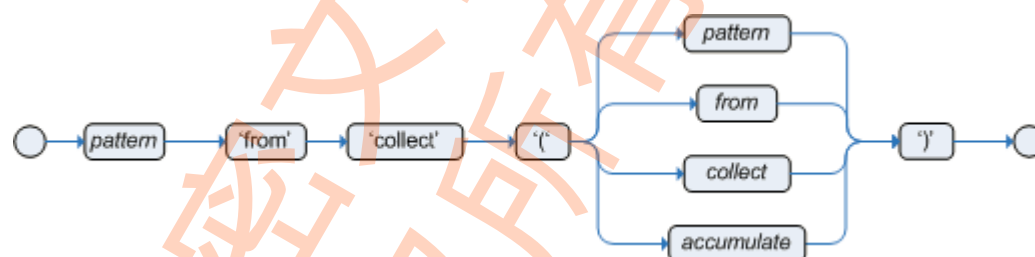
现在，你会发现两条规则如希望的一样被引发。然而，它并不总是可以访问上述的嵌套事实。考虑一个例子，Person 持有一个或多个 Address，并且你希望使用一个存在的量词匹配至

少有一个符合一定条件的地址的人。在这种情况下，你将不得不求助于 from 的使用，在集合上进行推断。有几种 from 的用法完成这种情况，并且不是所有它们与 lock-on-active 一起使用会表现出问题。

例如，下面的 from 用法，使两条规则如希望的一样被引发

```
rule "Assign people in North Carolina (NC) to sales region 1"
  ruleflow-group "test"
  lock-on-active true
  when
    $p : Person($addresses : addresses)
    exists (Address(state == "NC") from $addresses)
  then
    modify ($p) {} #Assign person to sales region 1 in a modify block
  end
end
rule "Apply a discount to people in the city of Raleigh"
  ruleflow-group "test"
  lock-on-active true
  when
    $p : Person($addresses : addresses)
    exists (Address(city == "Raleigh") from $addresses)
  then
    modify ($p) {} #Apply discount to person in a modify block
  end
end
```

条件元素 collect



条件元素 collect 允许规则在来自特定资源或工作内存的一个对象集合上进行推断。在一阶逻辑术语中，这是一个基数量词。小编这里简单的说明一下，在上面的 from 语法说明中我们可以了解到 from 是用来遍历的，值是分开获取的，加上 collect 后就会变成集合，将值获取后直接由集合进行管理。

一个简单的例子：

```
import java.util.ArrayList
rule "Raise priority if system has more than 3 pending alarms"
  when
    $system : System()
    $alarms : ArrayList( size >= 3 )
    from collect( Alarm( system == $system, status == 'pending' ) )
  then
```

```

# Raise priority, because system $system has
# 3 or more alarms pending. The pending alarms
# are $alarms.
end

```

在上面的例子中，该规则将为每个特定系统在工作内存中为查找所有紧急报警，并在 ArrayLists 中聚合它们。如果为一个特定系统找到了 3 个或更多，该规则被引发。

collect 的结果模式可以是任何具体的类，其实现了 java.util.Collection 接口，并且提供了一个无参公共构造函数。这意味着，你可以使用 Java 集合，如 ArrayList，LinkedList，HashSet，等等，或者你自己的类，只要实现了 java.util.Collection 接口，并且提供了一个无参公共构造函数。资源和结果模式都可以象任何其他模式一样被约束。

在 collect 条件元素之前的变量绑定是在资源和结果模式两个作用域中，因此你可以使用它们约束你的资源和结果模式。但是注意，collect 是一个域绑定分隔符，所以任何在它内部的绑定不可用于它的外部。

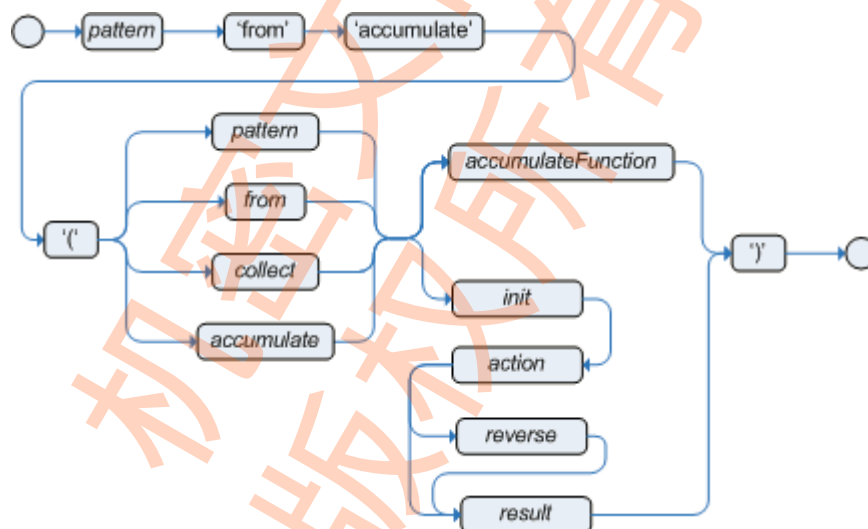
collect 认可嵌套的 from 条件元素。下面的例子是一个有效的 collect 用法：

```

import java.util.LinkedList;
rule "Send a message to all mothers"
when
    $town : Town( name == 'Paris' )
    $mothers : LinkedList()
    from collect( Person( gender == 'F', children > 0 ) from $town.getPeople() )
then
    # send a message to all mothers
end

```

条件元素 accumulate



条件元素 accumulate 是一个更灵活强大的 collect 形式，在这个意义上，它可以被用来做 collect 所做的事，也可以实现 collect 不能做的事。它主要做的事是允许规则迭代整个一个对象的集合，为每个元素定制执行动作，并在结束时返回一个结果对象，accumulate 既支持预定义的累积函数的使用，或也可以使用内联的自定义代码，内联定制代码应该避免，因为它是规则的作者更难维护，并且经常导致代码重复。Accumulate 功能更容易测试和重用。

Accumulate CE (preferred syntax)

accumulate(<source pattern 源模式>; <functions 函数> [;<constraints>])

```
rule "取对象中的最大值, 和最小值"
dialect "mvel"
when
    accumulate(Person($value:age),
                $min:min($value),
                $max:max($value);
                $max<=5
    )
then
    System.out.println($min+">>>>>>>>>>"+$max);
end
```

average	平均值
min	最小值
max	最大值
count	统计
sum	求和
collectList	返回 List
collectSet	返回 HashSet

```
rule "sun 求和后减 1"  
dialect "mvel"  
when  
    //$n:Number() from //可有可无  
    accumulate(Person($value:age),  
        $sum:sum($value-1)  
    )  
then  
    System.out.println("求和减 1>>>>>>>>>>" + $sum);  
end
```

那新建的这个自定义的函数(方法) 我们应该怎么用呢。其实很简单。引用就可以用了，只是在引用时，加一个关键字就 OK

```
import accumulate com.drools.TestAccunmulateneeded getResult
```

getResult 在自定义的类中可以写多个。也都能正常使用，这就说明，可以在同一个类可以写多个函数，即也可以对原来的方法进行修改。

内置的函数(总和、平均值等)是由引擎自动导入。只有用户定义定制的累积函数需要显式导入。

Accumulate with inline custom code

这个语法在刚研究的时候，官方就给你出了一个说明：

使用 inline 定制 code 的 accumulate 不是一个好的做法，原因有以下几点：维护和测试困难规则，使用它们代码重用性不高，好比在 JSP 中写 java 是一个道理，实现自己的 accumulate 时能非常简单明了，易于单元测试和使用，这些是重要的，这种形式的 accumulate 只支持身后兼容

下面是 inline 的语法结构

```
<result pattern>from accumulate(<source pattern>,init(<init code>),action(<action code>),reverse(<reverse code>),result(<result expression>))
```

下面我们对这些语法进行一个说明：

<source pattern>:这个表示源模式。用法：也就是我们常用手 Object(xx:XX 属性) 这个会去匹配每一个源对象。

<init code>:用法说明：init 是做初始化用的，简单的说，在 source pattern 遍历完之后就已经触发，有点像 for 的开头

<action code>:用法说明:action 会执行所以满足条件的源对象进行操作，像是 for 的方法体。在里面可写 java code

<reverse code>: 这是一个可选的被选方言的语义代码块，如果存在，将为不再匹配资源模式的每个资源对象执行。这个代码块的目的是不做在<action code> 块中做的任何计算，所以，当一个资源对象被修改或删除收时，引擎可能做递减计算，极大地提升了这些操作的性能

<result expression>: 返回值，是根据 action 上面两个遍历出来的结果进行一个返回，这个返回值中也可以进行计算。

<result pattern>: 返回值类型，在<result expression>返回值的类型再一次进行匹配，如果匹配不成功则返回 false。

举一下例子先来说明一下 注：<reverse code> 没使用

```
rule "测试 accumulatefrom 用法 1"
dialect "mvel"
when
$total : String() from
    accumulate(Person($value:dous),
        init( Double total = 0.1; Person p = new Person();),
        action( total += $value; ),
        result( p )
    )
then
System.out.println($total+"accumulate from 用法 求和");
end
```

在上面的例子中，我们可以清楚的看到，init 是做了一些初始化的工作，action 做的是迭代并计算的操作，result 是返回结果，这个规则可以返回类型 是在 from 前面定义的类型，上述例子中返回值是 p 它是一个 Person 的类，因为 Person 有 toString()，所以可正常返回，还能返回更多的类型与计算，这个得自己摸索研究了。

使用 Java 作为语义方言的例子,因此,注意使用分号作为初始化语句分隔符是强制性的,行动

和反向代码块。结果是一个表达式,因此,它不承认';'。如果用户使用其他方言,他必须服从,方言的具体语法。

和之前所述的, `reverse code` 是可选的, 但是强烈建议用户官写, 为了受益于改进的性能 `update and delete`

下面我们就谈一谈 例子中没有的 `reverse` 属性吧

首先我们先看一下 java 代码的用法: 不多解释, 很简单的调用。

禁止外传
机密文件，
版权所有

```

@Test
public void 测试 accumulatefrom 用法 reverse() throws Exception {
    Resource dis = ResourceFactory.newClassPathResource("rules/testdrl/accumulatefreverse.drl",
    TestTemplate01.class);
    KieHelper helper = new KieHelper();
    helper.addResource(dis, ResourceType.DRL);
    KieSession ksession = helper.build().newKieSession();
        for(int i=1;i<6;i++){
            Person person=new Person();
            person.setDous(1.0+i);
            person.setAge(i);
            ksession.insert(person);
        }
    int i = ksession.fireAllRules(new RuleNameStartsWithAgendaFilter("测试 accumulatef"));
    System.out.println( "      " + i + "次");
    ksession.dispose();
}

```

DRL 文件的使用：

```

/*rule "测试 accumulatefrom3 用法 reverse2"
dialect "mvel"
when
    $ps:Person(dous>=3)
then
    $ps.setDous(1.2);
    update($ps);
    System.out.println($ps.dous);
end*/
//为什么会将上面这一规则注释，因为根据调用方法，此规则会先执行，会影响到后面的规则

rule "测试 accumulatefrom 用法 reverse"
dialect "mvel"
when
    $total : Double() from
        accumulate(Person(dous>=3,$age:age),
            init(Double totls = 0.0),
            action(totls+=$age;System.out.println(totls+">>>>>>>");),
            reverse( totls-=$age;   System.out.println(totls+"<<<<<<<");),//,
            result( totls )
        )
then
    System.out.println($total+"+++++++");//+$s.count+$s.name);
end

```

```

rule "测试 accumulatefrom3 用法 reverse"
dialect "mvel"
when
$ps:Person(dous>=3)
then
$ps.setDous(1.2);
    update($ps);
    System.out.println($ps.dous);
end

```

简单的说明一下上面的例子：规则先会执行 from 用法 reverse 的这个规则，因为有满足的条件，所以会执行，但执行完成后执行 from3 用法 reverse 这个规则，这个规则用了 update 方法，将值改了，所以会重新

下面就针对上面的例子我们做一个总结：

- 1、accumulate 的使用，有一个很重要的函数 action。这个函数提供了匹配源模式的执行动作
- 2、将 action 看成两个状态，当源对象匹配源模式时，定会触发 action,我将触发过 action 的源对象称为有状态的（等同于标记），反之为无状态的
- 3、当我们传入的源对象在 RHS 中或者是其他规则的 RHS 中发生了改变(update,insert...)，则会触发所有满足条件规则的再次执行。
- 4、当规则再次被执行时，遇到 accumulate 时，则有状态的源对象会先执行 reverse 函数进行"回滚"操作，并将修改后的源对象再次与 accumulate 条件进行比较，如果比较为 true 则该对象(被修改过的)会再次触发 action 函数，如果比较为 false 则不会执行 action 函数
- 5、当规则再次被执行时，遇到 accumulate 时，则无状态的源对象会与 accumulate 条件进行比较，如果比较为 true 则该对象(被修改过的)会第一次触发 action 函数，如果比较为 false 则不会执行 action 函数。

注：在小编的多方面测试 accumulate 中，得出几个结论：

- 1、accumulate 有三种形式，分别是 inline 、\$min:min(XXX),min();
- 2、inline 这种形式虽然官方不推荐，不过不代表不能使用，这种形式
- 3、\$min:min(XXX)这只是其中的一种写法,但要注意的是,如果在函数前面加了\$xxx 变量时，则 accumulate 不是使用 from 方式,至于如何取\$XXX 变量的值,和模式方式是一样。当然根据上面的介绍,我们也可以通过在函数的结束符";"后面加条件约束,引用的方法也是同模式方式一样。

```

rule "取对象中的最大值，和最小值"
dialect "mvel"
when
    accumulate(Person($value:age),
        $min:min($value),
        $max:max($value);
        $max<=5
    )
then
    System.out.println($min+">>>>>>>>>"+$max);

```



```
end
```

4、min()写法,前面没有加任何的变量引用说明,当然也不可能在函数的结束符";"后面再加条件约束,并且,想要使用函数中返回的结果,在这种没有引用变量的前提下,则必须要引用 from 关键字, 但 from 关键字有一个问题,就是返回的结果,必须是一个(表示说源模式结束符后面只能有一个函数),且使用函数时,不能有变量引用,这样才能正常使用 from。

```
rule "min 的使用"
dialect "mvel"
when
    $n:Number()
    accumulate(Person($value:age);
                min($value)
    )
then
    System.out.println("min 的使用");
end
```

5、根据上面的说法 使用 accumulate 语法上如果 source pattern 结束符后面函数的结果有变量, 则不能使用 from,且使用 from 返回结果必须是一个。

上面说的其实都是 accumulate 中自带的一些函数和语法, 那是远远不够的, accumulate 的强大远远在我上面的例子之上, 下面我们就再深入一点: 试一下自定义函数的用法 (是不是很吊的样子.....)

首先, 我们要实现一个接口, 在 6.4 版本中要 AccumulateFunction

注: (官方上给出的接口在 6.4 版本中是不能用的)

实现了接口会重写里面的方法:

```
package com.drools;

import org.kie.api.runtime.rule.AccumulateFunction;

import java.io.*;

/**
 * Created by kangz on 2016/9/6.
 */
public class TestAccumulatedneeded implements AccumulateFunction{

    public static class Factorial implements Externalizable {
        public Factorial(){}

        public double total = 1;
        @Override
        public void writeExternal(ObjectOutput out) throws IOException {
            out.writeDouble(total);
        }
    }
}
```

```

}

@Override
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
    total=in.readDouble();
}
}

@Override
public Class<?> getResultType() {
    return Number.class;
}

@Override
public Serializable createContext() {
    return new Factorial();
}

@Override
public void init(Serializable serializable) throws Exception {
    Factorial factorial= (Factorial) serializable;
    factorial.total=1;
}

@Override
public void accumulate(Serializable serializable, Object o) {
    Factorial factorial= (Factorial) serializable;
    factorial.total *= ((Number)o).doubleValue();
}

@Override
public void reverse(Serializable serializable, Object o) throws Exception {
}

@Override
public Object getResult(Serializable serializable) throws Exception {
    Factorial factorial= (Factorial) serializable;
    Double d =new Double(((Factorial) serializable).total ==1?1:((Factorial) serializable).total);
    return d;
}

@Override
public boolean supportsReverse() {
    return true;
}

```

上面则就是自定义函数的例子。

如果想知道什么时候调用这些方法，则可以通过在方法中打印输出，后续我会将这些方法的使用法及作用做一个详细的说明，现在先卖个关子。

那我们写好上自定义函数了，我们在怎么用才行呢。这个简单，想想之前的 `function` 外部方法的使用，想到这里，你就应该明白了，不过关键字不同。

`import accumulate from itertools` 自定义的名称

通过方面的方法，我们就能使用我们自定义的函数了，但要注意的是，使用 `accumulate` 时，返回值上面有说过，这里不多说了。用法和 `sum`, `min` 等等是一样的，但要注意的是，如果我们自定的名称是 `sum`, `min` 等等，调用用的也是自带的方法，不会对原方法进行重写，那具体的用法是这样的：

整合 Drools6.4.0+Spring4.2

整合 Drools6.4.0+Spring+maven 整合如下配置 POM.xml

```
<!-- drools 规则引擎 版本 -->
<drools.version>6.4.0.Final</drools.version>
<spring.version>4.2.6.RELEASE</spring.version>
<!-- start drools 最少引用-->
<dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-spring</artifactId>
    <version>${drools.version}</version>
</dependency>
<!-- end drools -->
```

```
<!-- TEST begin -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
    <exclusions>
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
```

```

<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
<version>${spring.version}</version>
<exclusions>
  <exclusion>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
  </exclusion>
</exclusions>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-tx</artifactId>
  <version>${spring.version}</version>
</dependency>

```

```

<!-- spring orm -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${spring.version}</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-test</artifactId>
  <version>${spring.version}</version>
</dependency>
<!-- TEST end -->

```

Spring.xml 配置文件

```

<?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:kie="http://drools.org/schema/kie-spring"
  xsi:schemaLocation="
    http://drools.org/schema/kie-spring
    http://drools.org/schema/kie-spring.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">

```

```

<kie:kmodule id="kmodule"> <!-- 起名规范就可以 -->
    <kie:kbase name="kbase" packages="rules.spring"><!-- 起名规范就可以
packages=/指到最后一层文件夹 以 小数点为分隔符，与 java 相似/ -->
        <kie:ksession name="ksession" /> <!-- 起名规则就可以，这个的 name 是要在下面的
测试类中用到的 -->
    </kie:kbase>
</kie:kmodule>
    <!-- 注解 -->
    <bean id="kiePostProcessor"
class="org.kie.spring.annotations.KModuleAnnotationPostProcessor"/>

    <!-- Bean 工厂后置处理程序
<bean id="kiePostProcessor" class="org.kie.spring.KModuleBeanFactoryPostProcessor"/>
-->
</beans>

```

测试类

```

package drools.test;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.kie.api.cdi.KSession;
import org.kie.api.runtime.KieSession;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration( { "classpath:beast.xml" })
public class PersonDrools {
    @KSession("ksession")//注：这里的值与配置文件中的值是一样的
    KieSession ksession;
    @Test
    public void runRules(){
        int count = ksession.fireAllRules();
        System.out.println("总执行了"+count+"条规则");
        ksession.dispose();
    }
}

```

DRL 文件

```

package rules.spring;
rule "rule1" salience 10
when

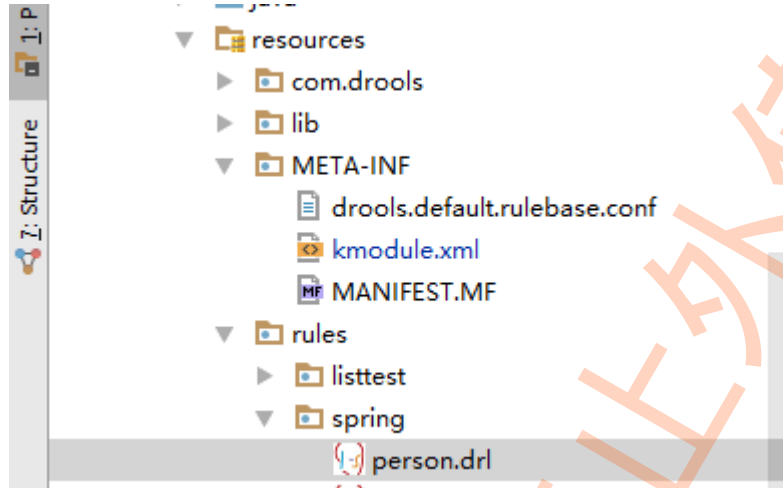
```

```

eval(true)
then
    System.out.println("hello spring drl ");
end

```

文件所在目录



通过 maven 整合 drools+spring 时 要注意：用@Test 运行时，有时 spring 是找到规则文件的，

方法一：可以找到你的项目 target 下 test-classes 下将你的规则文件 全部都放到里面。

方法二：在 pom.xml 配置文件也可以写下面这段代码也可以解决 spring 找不到规则文件的问题

```

<build>
  <testResources>
    <testResource>
      <directory>
        ${project.basedir}/src/main/resources
      </directory>
    </testResource>
  </testResources>
</build>

```

注：如果在 spring.xml 中引用不到 <kie:xxx> 之类的标签可以将 <http://drools.org/schema/kie-spring.xsd> 替换为

<https://raw.githubusercontent.com/droolsjbpm/droolsjbpm-integration/master/kie-spring/src/main/resources/org/kie/spring/kie-spring-6.0.0.xsd>

如果还是有问题，则看一下官方推荐。

与 web 项目整合的说明

Spring+web+drools 整合时，我这里先提一种方案：

```
<beanid="kiePostProcessor"class="org.kie.spring.KModuleAnnotationPostProcessor"/>
```

在 Spring 配置文件中使上面这段代码（其他不变）。

@Autowired

private KieBase kiebase;

然后在实现类里加上这面这段代码，并通过 KieSession kSession=kiebase.newKieSession();创

建会话。就完成了对 Ksession 的实例化了。

这种写法是可以在 **Controller** 里初始化的，小编这里提醒大家，如果不想使用 spring 自带的注入方式，还是按小编上面的例子的话，则 `@KSession("ksession")` 不能在 Controller 层中使用。

领域语言（自然语言）DSL

Drools 自然语言：是业务人员通过 `dslr` 文件编写的规则文件，业务人员可能不懂技术。通过用文字描述实现业务规则。什么时候使用 **DSL** 呢，

DSL 可以作为规则创作（以及创作者）和引擎操作的域对象之间的一个隔离层。**DSL** 也可以担当条件的“模板”或者在你规则中反复使用的动作，也许每次只使用了参数改变。如果你的规则需要被低技术的人们阅读和确认（这里指的是业务人员，不懂技术的），**DSL** 就起了决定性的作用。如果你规则的条件或推论遵守相同的模式，你可以用一个模板表达它们。你希望隐藏你的实现细节，专注业务规则。你需要提供一个根据预定义模板编辑规则的控制方法。**DSL** 在运行时没有影响规则，它们只是一个解析/编译时的功能。但要 **DSL** 文件解析成机器能读懂的程序，必须要在 `dslr` 文件中引用解析业务人员所写的业务规则。

DSL 机制是允许我们定制 **conditional expressions**（条件表达式 **LHS**）和 **consequence actions**（结果值的 也就是 **RHS**），也可以替换全局变量(如果替换，要进一步研究)

那我们从官方的例子做一个分析，例子如下：

`[when]Something is {colour}=Something(colour=="{colour}")`

关于上面的语法，`[when]`指表达式的作用域：即它属于一条规则的 **LHS** 或 **RHS**。在`[作用域]`后面的部分是你使用在该规则中的表达式（通常是一个自然语言表达式，但也不一定是）。`"=`的右边部分是映射到规则语言（当然，它的格式取决于你说的是 **LHS** 还是 **RHS**，例如，如果它是 **LHS**，那么它是正常的 **LHS** 语法，如果它是 **RHS**，那么它是 Java 代码片断）。

它的执行步骤分为三步：

1. 解析器将获得你指定的表达式，并提取出现在输入中的`{colour}`（称为令牌）相匹配的值。
2. 利用映射右手边的相应的`{colour}`（称为令牌），
3. 插入字符串替换任何被整个表达式匹配线的 **DSL** 规则文件。

简单来说就是将 左手边（=号左边）的 `{colour}` 做为模板参数，传值给右手边（=号右边）的`{colour}`的一个值的解析过程，整体的 左手边的自然语言表达式 是由右手边规则语言表达所提供解析和匹配的。小编这里就先给读者介绍一个简单的例子，我们来感受一下 **DSL** 带来的奇妙之旅。首先，我们要先创建一个 **DSL** 文件，这里的 `dslr` 是与我们平时写的 `drl` 文件相似，只是 **LHS** 部分与 **RHS** 部分是通过汉字去写的。

PersonDslr.dslr

```
package rules.testdsl;
import com.drools.api.rule.Person;
expander PersonDSL.dsl
rule "dslTest"
    when
        There is a Person with
            - age is less than or equal to 30
            - name equals "stilton"
    then
        公司给予你"《高级项目经理的职位》"的荣誉称号
```

end

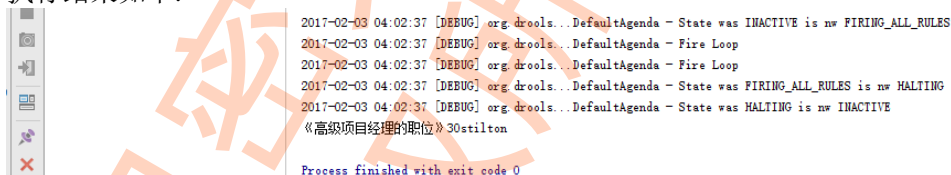
PersonDSL.dsl

```
[when][]is less than or equal to<=  
[when][]is less than=<  
[when][]is greater than or equal to>=  
[when][]is greater than=>  
[when][]is equal to===  
[when][]equals===  
[when][]年龄=age  
[when][]- {field:\w*}={field}  
[when]There is a Person with=$p:Person()  
[then]公司给予你"{post}"的荣誉称号=$p.setPost("{post}");
```

Java 代码

```
@Test  
public void tsetkeyword() throws Exception {  
    Resource dsl = ResourceFactory.newClassPathResource("rules/testdsl/PersonDSL.dsl");  
    Resource dsrl = ResourceFactory.newClassPathResource("rules/testdsl/PersonDsrl.dsrl");  
    KieHelper helper = new KieHelper();  
    helper.addResource(dsl, ResourceType.DSL);  
    helper.addResource(dsrl, ResourceType.DSLR);  
    KieSession session = helper.build().newKieSession();  
    com.drools.api.rule.Person p=new com.drools.api.rule.Person();  
    p.setAge(30);  
    p.setName("stilton");  
    session.insert(p);  
    int i=session.fireAllRules();  
    System.out.println(p.getPost()+p.getAge()+p.getName());  
    session.dispose();  
}
```

执行结果如下:



```
2017-02-03 04:02:37 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_RULES  
2017-02-03 04:02:37 [DEBUG] org.drools...DefaultAgenda - Fire Loop  
2017-02-03 04:02:37 [DEBUG] org.drools...DefaultAgenda - Fire Loop  
2017-02-03 04:02:37 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING  
2017-02-03 04:02:37 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE  
《高级项目经理的职位》30stilton  
Process finished with exit code 0
```

结果与我们想的一样，与小编之前所说的 drl 都是不一样的，这里是我们通过业务人员能看明白的文字进行判断并在 LHS 和 RHS 部分中体现出来的，根据上面我们得出的一小部分结论后，我们再深入一步研究，

注意：表达式（即条件部分，等号左边的）作为正则表达式模式匹配操作对线的 DSL 规则文件，全部或者部分匹配的行。这表示我们可以使用（例如）"?"显示可选的字符，我们就可以克服变化自然语言短语的 DSL。但要注意的是，由于表达式是正则表达式模式，使用时 要通过\"来转意

好！对 DSL 有了一点点认识之后，那我们就说一下通过正则的方式 做一个比较复杂例子再更深一步对领域语言做一个分析

Person.dsrl

```

package rules.testdsl;
import com.drools.api.rule.Person;
import com.drools.api.rule.School;
expander PersonDSL.dsl
rule "rule2"
when
    我们要找一个人
    - 姓名 等于 "张三"
    - 年龄 等于 30
    - 年小 等于 "李四小"
    我们要找一家学校
    - 学名 等于 "新一中"
    - 人数 等于 50
then
    公司给予你"《高级项目经理的职位》"的荣誉称号
    输出这家学校的名字
    1 学名 等于 "新一中"
end

```

PersonDSL.dsl

```

[when][]我们要找一个人=$p:Person()
[when][]我们要找一家学校=$s:School()
[when][]姓名=name
[when][]年龄=age
[when][]学名=name
[when][]年小=aGess
[when][]人数=count
[when][]等于===
[when][]- {field:[\u4e00-\u9fa5_a-zA-Z0-9^"]+} {operator} {value:[\u4e00-\u9fa5_a-zA-Z0-9^"]+}={field}
{operator} {value}
[then]公司给予你"{post}"的荣誉称号=$p.setPost("{post}");
[then]输出这家学校的名字= System.out.println("这家学校的名字是"+$s.getName());
System.out.println(kcontext.getRule().getName());
[then]学名=name66
[then][]1 {field:[\u4e00-\u9fa5_a-zA-Z0-9^"]+} {operator} {value:[\u4e00-\u9fa5_a-zA-Z0-9^"]+}
=System.out.println("{field} +++++"); System.out.println("{operator} +++++"); System.out.println({value});

```

通过上面的例子，我们可能得出一结论，不管是在[when]还是[then]中，学名的定义是不冲突的，我们在 dsl 文件中可以看到学名已经被替换了 name 或 name66 【为了区别 when then】在我们输出的时候进行正则匹配时 其实匹配的是 已经被替换了的 name 或 name66

如果在使用 一些关键符号时 比如说：{} 我们拿官方上的例子说明一下“

if (foo) {doSomething();} 如果我们这样直接写的话 DSL 会认为是 左手边（等号左边）中的定义，这样一来，我们的 if 语法就是出错，这时，我们要拿 \进行转意。 例如：

```
[then]do something= if (foo) \{ doSomething(); \}
```

作用域和关键字

如果你使用 GUI 编辑 DSL，或用文本编辑，你会注意到一个 [作用域] 项目在每个映射行的开始。它指示是否是该语句/文字应用到 LHS，RHS，或者是一个关键字。有效的值有[condition],[consequence] 和[keyword] ([when] 和 [then] 分别与[condition]和[consequence] 相同)。当使用期[keyword]时，它的意思是你可以映射语言的任何关键字，诸如"rule"或"end"为另外东西。通常，它只用于你希望使用非英语规则语言时（最好将它映射成单个文字）。

先定义一个和 dsl 很相似内容的 dslr 文件，简单来说，在规则中用自然语法，肯定是有特殊的语法和文件的，不单单只是一个 dsl 的文件，dslr 文件，就是用来写规则的。这里 dslr 只是文件扩展名

Test.dslr

```
rule "name"
when

then
    hello zhangsan
end
```

Test.dsl

```
[when]    //备注一下：这里因为没有条件部分 所以这里空的
[then]    hello {name:\w*}=System.out.println("{name}");
```

简单的说明一下，`:\w*` 是指 `zhangsan` 是按这个正则所取的内容。也就是说解析器将匹配那个匹配了表达式的语句片断。`:\w*` 不是必须的。如果没有 `:\w*` 这个的话 则表式 `zhangsan` 就是表达式的值。

好，我们再升级一下这个规则 添加一个约束条件
在 LHS 中，为类型增加条件约束通过一个“-”实现。

Test.dslr

```
rule "name"
when
    Person
    - is a zhangsan
then
    hello zhangsan
end
```

Test.dsl

```
[when] Person=Person()
[when] - is a {name:\w*}= name=="{name}"
[then]hello {name:\w*}=System.out.println("{name}");
```

小编这里强调一下“-”的使用，“-”我们可以看作是 `Person` 类型中，属性添加条件约束，就上面的例子来说，`- is a zhangsan` 其实就是我们 dsl 文件中所写的 `Person(name=="zhangsan")`

这里我们再深入一下：写一个规则：

Test.dslr

```
package rules.testdsl;
```

```

import com.drools.api.rule.Person;
import com.drools.api.rule.School;
expander PersonDSL.dsl
rule "rule2"
    when
        我们要找一个人
        - 姓名 等于 "张三"
        - 年龄 等于 30
        - 校名 等于 "新一中"
        - 人数 等于 50
    then
        公司给予你"《高级项目经理的职位》"的荣誉称号
        输出这家学校的名字
    end

```

Test.dsl

```

[when][]我们要找一个人=$p:Person()
[when][]姓名=name
[when][]年龄=age
[when][]校名=$sname:school.name
[when][]人数=school.count
[when][]小于或等于=<
[when][]小于=<
[when][]大于或等于==>
[when][]大于=>
[when][]等于===
[when][]- {conditions}={conditions}
[then]公司给予你"{post}"的荣誉称号=$p.setPost("{post}");
[then]输出这家学校的名字= System.out.println("这家学校的名字是"+$sname);

```

其实很简单，就是 Person 这个类 姓名为张三 年龄等于 30 校名是 新一中 人数是 50 的一条规则。

上面的情况都是 && 的关系（在括号里面），我们再加难度，将规则改为 使用 || 的形式，之前我们经过测试， "-" 的形式 其实就是 该类里面对属性的操作，默认为 "," 也就是 && 的关系，如果我们想要将 "，" 改为 && 所替换的值，那我们就要这样写：

Test.dslr

```

rule "rule2"
    when
        我们要找一个人
        - 姓名 等于 "张三"
        - 年龄 等于 30 并 年小 等于 "李四小"
        我们要找一家学校
        - 学名 等于 "新一中"
        - 人数 等于 50
    then
        公司给予你"《高级项目经理的职位》"的荣誉称号
    end

```

输出这家学校的名字

1 学名 并 "新一中"

end

Test.dsl

```
[when][]我们要找一个人=$p:Person()
[when][]我们要找一家学校=$s:School()
[when][]姓名=name
[when][]年龄=age
[when][]学名=name
[when][]年小=aGess
[when][]人数=count
[when][]小于或等于=<=
[when][]小于=<
[when][]大于或等于==>
[when][]大于=>
[when][]等于===
[when][]或者=or
[when][]并=&&
[when][]或=||
[when][]- {field:[\u4e00-\u9fa5_a-zA-Z0-9^"]+} {operator} {value:[\u4e00-\u9fa5_a-zA-Z0-9^"]+}={field} {operator} {value}
```

上面的例子中，小编在 年龄 30 的后面加上“并”在 dsl 文件中 并=&&， 所以改变了默认的“，”的使用，当然也可以将并改为或但这两种方式， 只能在操作属性时进行使用，且要写在同一行进行操作，如果想要写成 and or 的方式那就看下面的例子吧

Test.dslr

```
[when][]我们要找一个人=$p:Person()
[when][]我们要找一家学校=$s:School()
[when][]姓名=name
[when][]年龄=age
[when][]学名=name
[when][]年小=aGess
[when][]人数=count
[when][]小于或等于=<=
[when][]小于=<
[when][]大于或等于==>
[when][]大于=>
[when][]等于===
[when][]或者=or
[when][]并且=and
[when][]- {field:[\u4e00-\u9fa5_a-zA-Z0-9^"]+} {operator} {value:[\u4e00-\u9fa5_a-zA-Z0-9^"]+}={field} {operator} {value}
[then]公司给予你"{post}"的荣誉称号=$p.setPost("{post}");
[then]输出这家学校的名字= System.out.println("这家学校的名字是"+$s.getName());
System.out.println(kcontext.getRule().getName());
```

Test.dsl


```

rule "rule2"
  when
    我们要找一个人
    - 姓名 等于 "张三"
    - 年龄 等于 30
    - 年小 等于 "李四小"
  或者
    我们要找一家学校
    - 学名 等于 "新一中"
    - 人数 等于 50
  then
    公司给予你"《高级项目经理的职位》"的荣誉称号
    输出这家学校的名字
  end

```

注意，使用 OR 时要注意 存在短路机制，上面的这段代码在 `drl` 文件中是这样体现的

```
$p:Person(name=="张三",age==30,aGess=="李四小") or $s:School(name=="新一中",count==50)
```

经过上面的例子，我们不难发现，在 DSL 中所替换了的值，例如 `年龄=age` 在我们编辑 LHS 或者 RHS 部分时就已经将其替换了，在使用 DSL 时，我们一定注意的一个点就是顺序的问题，在 RSLR 文件中使用括号时，要注意，最好是将括号部分条件 写在同一行。

规则模板

在规则引擎中，Drools 提供了一个规则模板的概念，如何理解这个模板呢，小编这里就给读者详细的说明一下，规则模板，即规则条件比较值是可变的，且可生成多个规则进行规则调用。规则模板在小编看来可以分为两种，第一种为官方上提到的以 `drt` 扩展名+`xls`(源数据)的方式，第二种为 API 模板赋值方式。

小编这里就对这两种方式，一一为读者进行解说

使用规则模板时，我们要添加 jar 包的引用
编辑 `pom.xml` 文件，添加依赖

```

<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-templates</artifactId>
</dependency>

```

第一种：drt+xls (xlsx) 方式

与决策表（不一定需要电子表格）相关的是“规则模板”（`drools-templates` 模块中）。它们使用任何表格式的数据源作为一个规则数据源——填入模板产生多数规则。这可以允

许两个更灵活的电子表格，而且实例在现有的数据库中管辖（代价是预先开发产生规则的模板）。

利用规则模板，数据与规则分离，并且有关规则的数据驱动部分没有限制。所以，你同时可以做你在

规则表中能够做的任何事情，你也可以做到以下几点：

- 存储你的数据在数据库中（或者任何其他格式）。
- 根据数据的值有条件地产生规则。
- 为你的规则的任何部分使用数据（例如，条件运算符，类名，属性名）。
- 在相同的数据上运行不同的模板。

Test.drt

```
template header
age
log

package rules.testdrt;
import com.drools.api.rule.Person;
global java.util.List list;

template "cheesefans"
rule "Cheese fans_{row.rowNumber}"
    when
        Person(age == @{age})
    then
        list.add("@{log}");
    end
end template
```

上面的代码，就是一个规则模板的文件，我们不难发现，与我们之前所接触的 `drl` 规则文件是有所不同的，多了 `template` 相关的关键字，从字面意思看来，这就是一个模板，`@{age}`、`@{log}` 小编这里的定义是占位符，那上在代码除了我们所熟知的 `drl` 方法，其他关键字都代码什么呢，小编这里对每一行做了一个说明，请看下面的代码

```
template header    #注：template header 固定写法，必须在模板的最开始部分
age                #注：age log 表示标题的是依次显示数据的列的列表。在这个案例中，我们命名第一列为
"age"，第三列为"log"
log
                  #注：这里是一空行，表示定义结束

package rules.testdrt;
import com.drools.api.rule.Person;
global java.util.List list;

template "cheesefans"
#注"template" 关键字通知一个规则模板开始。在一个模板文件中可能超过一个模板。模板应该有一个唯一的名字
rule "Cheese fans_{row.rowNumber}"
#注： @{row.rowNumber}， 它给每个数据行一个唯一的数字， 使你能产生唯一的规则名字
    when
```

```
Person(age == @{age})
then
    list.add("@{log}");
end
end template
```

在 xls 中的列数据要与规则模板中用来做操作的类型包括 java 中的类型保持一致。否则会报错，但要注意的一点是 如果用的是 String 类型的 在模板中引用 必须是要加 “”的 @{age} 是数字类型的 所以可以省略不加

Test.xls（数据源）

Case	Persons age	Cheese type	Log
Old guy	42	stilton	Old man stilton
Young guy	21	cheddar	Young man cheddar

根据上面例子+xls 中的数据，其实是可以生成这样一个规则文件：

每一行就是一条规则，从 row 值开始

```
package org.drools.examples.templates;
global java.util.List list;
rule "Cheese fans_1"
when
    Person(age == 42)
then
    list.add("Old man stilton");
end
rule "Cheese fans_2"
when
    Person(age == 21)
then
    list.add("Young man cheddar");
end
```

相信读者看到这里，是不是已经明白了？ 其实就是将 xls 中的数据填写到模板文件中。但在编写 xls 有几个细节要注意一下：下面小编就对 xls 文件进行分析

第一点：xls 中的具体行数据是根据配置文件 row 值来确定的，如果设置为 row="1"则表示从第一行开始取数据

第二点：xls 中的具体列数据是根据配置文件 col 值来确定的，如果设置为 col="1"则表示从第一列开始取数据

具体的 java 调用，其实与执行规则是一样的，只是在配置文件中是不同的，kmodule.xml

```
<kbase name="kbase3" packages="rules.testdrt">
    <ruleTemplate dtable="rules/testdrt/ExampleCheese.xls"
        template="rules/testdrt/Cheese.drt"
        row="2" col="2"/>
    <ksession name="session3"/>
</kbase>
```

ruleTemplate 规则模板，dtable 引用 xls 文件，表示二维表 template 具体的模板文件

row/col 表示行/列，引用 xls 的话 该值是不能省略的。

Test.drl

JAVA 代码

```
@Test
public void ruleDaseTest2() throws Exception {
    ObjectDataCompiler converter = new ObjectDataCompiler();
    //赋值 给模板属性
    Personst p=new Personst();
    p.setSex(false);
    p.setName("张三");
    Collection<Personst> cfl = new ArrayList<Personst>();
    cfl.add(p);//每 add 一次，就代码一条规则
    InputStream dis = ResourceFactory.newClassPathResource("rules/testdrltem/Item2.drl",
TestTemplate01.class).getInputStream();
    String drl = converter.compile(cfl, dis);
    System.out.println(drl);
    KieHelper helper = new KieHelper();
    helper.addContent(drl, ResourceType.DRL);
    KieSession ksession = helper.build().newKieSession();
    Person ps = new Person();
    ps.setAge(30);
    ps.setSex(false);
    ksession.insert(ps);
    int i = ksession.fireAllRules();
    System.out.println(ps.getName() + "          " + i + "次");
    ksession.dispose();
}
```

规则模板中的数据源 JavaBean

```
public class Personst {
    private boolean bSex;
    private int nCount;
    //此次省去 get set 方法，但读者一定要加上哦，

}
```

经输出到控制台的规则模板的内容

[illegible]

这里小编再给模板补上一些说明，模板是可以传一些基本类型，至于集合或对象，则是传的它们的 `toString()` 方法，所以在模板中，取不到相应的集合内容，和地址。要模板属性中，如果是用的 `javaBean` 的话，要注意，其实 `javaBean` 的属性引用，调用的是 `get` 方法，如果用比如：`bName;bSex`；这样的名来命名，则生成的 `getXXX` 方法会是 `getbSet().getbName()`，我们在属性中引用时会引用不到，所以得出的一个结论就是，`javaBean` 属性做模板属性时，要注意，生成的 `get` 方法，`get` 后第一个字母必需大写，还是有一点就是，在模板中，其实属性引用的是无参的有返回值的方法，也可以在模板里直接写 `get` 方法等。

规则模板是不是很大，其实规则模板的强大，在这章节里，小编又一次提到了占位符的概念，那我们是不是可以将其理解成或改造成动态规则呢，是的，规则模板是可以这样做的，小编这里给读者一个提示，像在 API 中给模板的占位符赋值时，我们就能看出来，在规则模板中，我们也可以将条件写在赋值，这样就能实现动态规则了。

Tomcat 配置详细说明 liunx

传统安装方式

1. 下载 linux 版本的 jdk 包，这边载的是 jdk-8u91-linux-x64.tar.gz
2. 在/usr/local 下使用"mkdir java"创建 java 目录，将 jdk 文件放入其中
3. 通过"tar -zxvf ./jdk-8u91-linux-x64.tar.gz -C ."将其进行解压
4. 配置环境变量，运行"sudo vi /etc/profile"，在最后插入要配置的内容：
export JAVA_HOME=/usr/local/java/jdk1.8.0_91/
export PATH=\$JAVA_HOME/bin:\$PATH
export CLASSPATH=.:\$JAVA_HOME/lib/dt.jar:\$JAVA_HOME/lib/tools.jar
按 Esc 键，输入(:wq 保存并退出)
5. 运行"source /etc/profile"，使配置环境生效
6. 运行"java -version"看是否生效，若出现 jdk 版本号，则安装并配置环境变量成功，如下图所示：

```
[root@zy49 /]# java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.91-b14, mixed mode)
```

下载 tomcat, 下载地址: <http://tomcat.apache.org/download-70.cgi>

下载 Workbench 的 war 包, 下载地址: <http://www.drools.org/download/download.html>

Drools and jBPM integration	Drools and jBPM integration with third party project like Spring. Distribution zip contains binaries, examples and sources.	Distribution ZIP
		WildFly 10 WAR
		Tomcat 7+ WAR
		EAP 6.4 x WAR
		EAP 7 x WAR
		WebSphere 8.5.x WAR
		WebLogic 12c WAR
		Guvnor 5.x migration tool ZIP
		KIE Config CLI ZIP
		Example GIT Repositories ZIP
Drools Workbench	Drools Workbench is the web application and repository to govern Drools and jBPM assets. See documentation for details about installation.	
Drools and jBPM tools	Eclipse plugins and support for Drools, jBPM and Guvnor functionality. Distribution zip contains binaries and sources.	Distribution ZIP

1、下载成功后, 将 war 包放在 tomcat - workbench/webapps/下 并改名为 kie-wb.war

光有一个 war 包是肯定不够, 还要导入一些 jar 一几个配置文件,

2、将下列几个 jar 拷贝到 TOMCAT_HOME/lib:

- * btm-2.1.4.jar
- * btm-tomcat55-lifecycle-2.1.4.jar
- * h2-1.3.161.jar
- * jta-1.1.jar
- * slf4j-api-1.7.2.jar
- * slf4j-jdk14-1.7.2.jar
- * log4j-api-2.1.jar
- * log4j-slf4j-impl-2.1.jar

log4j-api-2.1.jar 和 log4j-slf4j-impl-2.1.jar 在官方说明没有提到, 但是不添加的话, 在 tomcat 启动加载的时候会报错!

3、在 TOMCAT_HOME/conf 下, 新建 btm-config.properties(vim 命令新建), 内容如下(每个语句后面不能有空格):

```
bitronix.tm.serverId=tomcat-btm-node0
bitronix.tm.journal.disk.logPart1Filename=${btm.root}/work/btm1.tlog
bitronix.tm.journal.disk.logPart2Filename=${btm.root}/work/btm2.tlog
bitronix.tm.resource.configuration=${btm.root}/conf/resources.properties
```

在 TOMCAT_HOME/conf 下, 新建 resources.properties(vim 命令新建), 内容如下(每个语句后面不能有空格):

```
resource.ds1.className=bitronix.tm.resource.jdbc.lrc.LrcXADataSource
resource.ds1.uniqueName=jdbc/jbpm
resource.ds1.minPoolSize=10
resource.ds1.maxPoolSize=20
resource.ds1.driverProperties.driverClassName=org.h2.Driver
resource.ds1.driverProperties.url=jdbc:h2:file:~/jbpm
resource.ds1.driverProperties.user=sa
resource.ds1.driverProperties.password=
resource.ds1.allowLocalTransactions=true
```

4、在 TOMCAT_HOME/bin 下，新建 setenv.sh(vim 命令新建)，内容如下：

```
CATALINA_OPTS="-Xmx512M -XX:MaxPermSize=512m -
Dbtm.root=$CATALINA_HOME
-Dbitronix.tm.configuration=$CATALINA_HOME/conf/btm-config.properties
-Djbpm.tsr.jndi.lookup=java:comp/env/TransactionSynchronizationRegistry
-Djava.security.auth.login.config=$CATALINA_HOME/webapps/kie-wb/WEB-
INF/classes/login.config
-Dorg.jboss.logging.provider=jdk
-Dorg.jbpm.cdi.bm=java:comp/env/BeanManager
-Dorg.guvnor.m2repo.dir=/root/.m2/repository
-Dorg.kie.demo=false
-Dorg.kie.example=false"
```

5、配置 JEE security

- a) 将"kie-tomcat-integration" JAR 拷贝到 TOMCAT_HOME/lib(org.kie:kie-tomcat-integration)
- b) 将"JACC" JAR 拷贝到 TOMCAT_HOME/lib(javax.security.jacc:artifactId=javax.security.jacc-api in JBoss Maven Repository)
- c) 将"slf4j-api" JAR 拷贝到 TOMCAT_HOME/lib (org.slf4j:artifactId=slf4j-api in JBoss Maven Repository)

6、在 TOMCAT_HOME/conf/server.xml 的 Host 节点最后添加：

```
<Valve className="org.kie.integration.tomcat.JACCValve" />
```

7、编辑 TOMCAT_HOME/conf/tomcat-users.xml，添加'analyst'或者'admin'角色，添加 kie-wb 相应的用户，如下：

```
<role rolename="admin" />
<role rolename="analyst" />
<user username="kie" password="kie" roles="admin" />
<user username="kie-analyst" password="kie-analyst" roles="analyst" />
```

8.启动 tomcat，访问 http://linux 的 ip 地址:8080/kie-wb，可以看到以下界面，说明已搭建成功！



参数 -Dorg.kie.demo=false 的作用是在无互联网环境下去运行 kie-drools 时，如果不加此参数 kie-drools 会在每次运行时去 GIT 试图加载 kie-drools 的 demo，如果你的服务器为虚拟

机或者是无互联网环境时它会因为建立 internet 连接超时而抛出一个疑似 memory leak 的 exception 而导致整个 war 工程加载失败。

-Dorg.kie.demo=false

-Dorg.kie.example=false

Mysql 安装方式

在 Linux 上进行部署的，用的是 mysql 为存储介质。小编先说明一下 如果安装 mysql。

第一步：执行命名 mysql 数据库是否安装

rpm -qa | grep mysql // 这个命令就会查看该操作系统上是否已经安装了 mysql 数据库有的话，我们就通过 **rpm -e** 命令或者 **rpm -e --nodeps** 命令来卸载掉

```
[root@xiaoluo ~]# rpm -e mysql // 普通删除模式
[root@xiaoluo ~]# rpm -e --nodeps mysql // 强力删除模式，如果使用上面命令删除时，提示有依赖的其它文件，则用该命令可以对其进行强力删除
```

第二步：执行命名，查看 yum 上提供的 mysql 下载版本

yum list | grep mysql 命令来查看 yum 上提供的 mysql 数据库可下载的版本下图是出现的版本

```
[root@xiaoluo ~]# yum list | grep mysql
apr-util-mysql.x86_64                1.3.9-3.el6_0.1      base
bacula-director-mysql.x86_64        5.0.0-12.el6         base
bacula-storage-mysql.x86_64        5.0.0-12.el6         base
dovecot-mysql.x86_64                1:2.0.9-5.el6        base
freeradius-mysql.x86_64             2.1.12-4.el6_3       base
libdbi-dbd-mysql.x86_64             0.8.3-5.1.el6        base
mod_auth_mysql.x86_64               1:3.0.0-11.el6_0.1   base
mysql.x86_64                        5.1.67-1.el6_3       updates
mysql-bench.x86_64                  5.1.67-1.el6_3       updates
mysql-connector-java.noarch         1:5.1.17-6.el6        base
mysql-connector-odbc.x86_64         5.1.5r1144-7.el6     base
mysql-devel.i686                    5.1.67-1.el6_3       updates
mysql-devel.x86_64                  5.1.67-1.el6_3       updates
mysql-embedded.i686                 5.1.67-1.el6_3       updates
mysql-embedded.x86_64               5.1.67-1.el6_3       updates
mysql-embedded-devel.i686           5.1.67-1.el6_3       updates
mysql-embedded-devel.x86_64         5.1.67-1.el6_3       updates
mysql-libs.i686                     5.1.67-1.el6_3       updates
mysql-libs.x86_64                   5.1.67-1.el6_3       updates
mysql-server.x86_64                 5.1.67-1.el6_3       updates
mysql-test.x86_64                   5.1.67-1.el6_3       updates
php-mysql.x86_64                    5.3.3-22.el6         base
qt-mysql.i686                       1:4.6.2-26.el6_4     updates
qt-mysql.x86_64                     1:4.6.2-26.el6_4     updates
rsyslog-mysql.x86_64                5.8.10-6.el6         base
```

第三步：执行命名安装 mysql

通过输入 **yum install -y mysql-server mysql mysql-devel**

查看刚安装好的 mysql-server 的版本 **rpm -qi mysql-server**

第四步：执行命名启动服务

通过输入 **service mysqld start** 命令就可以启动我们的 mysql 服务。

第五步：执行命名，进 mysql 进行一些配置

我们在使用 mysql 数据库时，都得首先启动 mysqld 服务，我们可以通过 **chkconfig --list | grep mysqld** 命令来查看 mysql 服务是不是开机自动启动

```
mysqld                                0:关闭    1:关闭    2:关闭    3:关闭    4:关闭    5:关闭    6:关闭
```

当然可以通过 **chkconfig mysqld on** 命令来将其设置成开机启动，这样就不用每次都去手动启动了

mysqld 0:关闭 1:关闭 2:启用 3:启用 4:启用 5:启用 6:关闭

第六步：为 root 帐号设置密码

mysql 数据库安装完以后只会有一个 root 管理员账号，但是此时的 root 账号还并没有为其设置密码，在第一次启动 mysql 服务时，会进行数据库的一些初始化工作，在输出的一大串信息中，我们看到有这样一行信息：

```
/usr/bin/mysqladmin -u root password 'new-password' // 为root账号设置密码
```

通过命名设置密码：

mysqladmin -u root password 'admin'// 通过该命令给 root 账号设置密码为 admin

通过 mysql -u root -p 命令来登录我们的 mysql 数据库了

```
[root@hadoop002 ~]# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

第七步：创建数据库

create database drools; 创建数据库

因为刚安装的数据库会出现问题，未授权。启动时会报错，我们这里通过两个命名执行一下：远程 MYSQL 1103 错误 ERROR 1130: Host *.*.* is not allowed to connect to

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' IDENTIFIED BY 'admin' WITH GRANT OPTION;
```

这里的 admin 写自己 mysql 的登录密码

flush privileges; 这个表示刷新用户权限

安装 Mysql 成功后，小编就带着读者来完成 mysql+workbench 的安装方式，其实读者安装过传统方式，那下面就非常的容易。

1、拷贝 mysql 的驱动 jar 包到 TOMCAT_HOME/lib

2、修改 resources.properties 配置文件

```
resource.ds1.className=bitronix.tm.resource.jdbc.lrc.LrcXADataSource
resource.ds1.uniqueName=jdbc/jbpm
resource.ds1.minPoolSize=10
resource.ds1.maxPoolSize=20
resource.ds1.driverProperties.driverClassName=com.mysql.jdbc.Driver
resource.ds1.driverProperties.url=jdbc:mysql://localhost:3306/drools?useUnicode=true&characterEncoding=UTF-8
resource.ds1.driverProperties.user=root
resource.ds1.driverProperties.password=admin
resource.ds1.allowLocalTransactions=true
```

3、TOMCAT_HOME/conf/目录下修改 context.xml 增加

```
<Resource name="jdbc/jbpm" auth="Container" type="javax.sql.DataSource"
driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/drools?useUnicode=true&characterEncoding=UTF-8"
username="root"
password="admin"
maxActive="20"
maxIdle="1"
maxWait="5000" />
```

```
at java.lang.Thread.run(Thread.java:745)
Jul 06, 2016 7:33:49 AM org.apache.catalina.startup.ContextConfig processContextConfig
SEVERE: Parse error in context.xml for /docs
org.xml.sax.SAXParseException; systemId: file:/usr/local/tomcat7/conf/context.xml; lineNumber: 37; columnNumber: 86; The reference to entity "characterEncoding" must end with the ';' delimiter
at com.sun.org.apache.xerces.internal.util.ErrorHandlerWrapper.createSAXParseException(ErrorHandlerWrapper.java:203)
at com.sun.org.apache.xerces.internal.util.ErrorHandlerWrapper.fatalError(ErrorHandlerWrapper.java:177)
at com.sun.org.apache.xerces.internal.impl.XMLErrorReporter.reportError(XMLErrorReporter.java:400)
at com.sun.org.apache.xerces.internal.impl.XMLErrorReporter.reportError(XMLErrorReporter.java:322)
```

这里要注意：因为在 xml 中 & 是一个特殊字符所以要进行转义 &=&，如果不设置就会报上面图片中的错在启动时

4、修改：kie-drools-wb/WEB-INF/classes/META-INF/persistence.xml

```
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<properties>
<!--
<property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
-->
<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>
<property name="hibernate.max_fetch_depth" value="3"/>
<property name="hibernate.hbm2ddl.auto" value="update"/>
<property name="hibernate.show_sql" value="false"/>
<property name="hibernate.transaction.manager_lookup_class" value="org.hibernate.transaction.JBossTransactionManagerLookup"/>
<!-- BZ 841786: AS7/EAP 6/H1b 4 uses new (sequence) generators which seem to cause problems -->
<property name="hibernate.id.new_generator_mappings" value="false"/>
</properties>
</persistence-unit>
```

5、启动 tomcat

登录：http://IP 地址:8080/kie-drools-wb 如果部署成功 就是出现下面这个页面。

mysql 相关的数据库中会生成 41 张表。

Oracle 安装方式

既然有了 Mysql 安装方式，那么自然就有了 Oracle 安装方式，相信读者也能想到，只要将数据源换成 Oracle 并引用 Oracle 驱动 Jar 包即可，但所生成的表则不是 41 张，而是 38 张表左右。

workbench 与 java 的交互

光说了这么多如何调用 workbench，这 workbench 光在本地服务上创建了规则了，那里的规则又如何使用呢，小编下面就给读者说一说 workbench 与 java 的两种交互方式，配置这两种交互方式得有一些前置工作要做，那小编就先来讲一讲我们要做的前置工作是什么，

第一步：创建组织单元及资料库

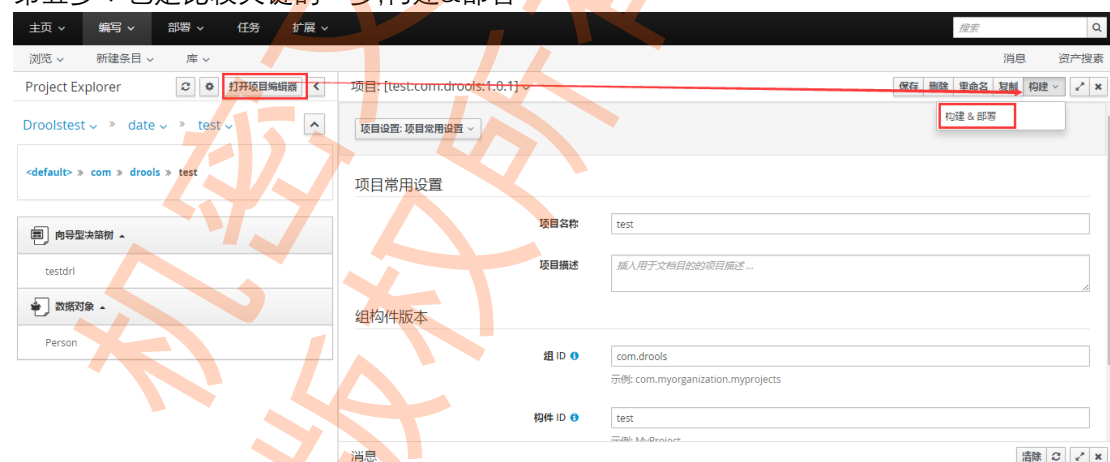
第二步：新建项目，这里要特别注意一下 项目中的构件部分

第三步：新建数据对象及规则，这里小编用的是决策树



第四步：检验规则是否正确，通过没影场景。

第五步：也是比较关键的一步,构建&部署



完成以上前置条件后，下面小编就来讲述一下 workbench 与 java 的两种访问方式，简单方式与自动扫描

简单方式

使用简单试与 workbench 进行交互，首先要创建一个 maven 的项目，引用相关的 jar 包，及一个管理配置 请看下面的 pom.xml 的配置

```
<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>org.kirito</groupId>
    <artifactId>testdrools</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <properties>
        <drools.version>6.4.0.Final</drools.version>
    </properties>
    <repositories>
        <repository>
            <id>guvnor-m2-repo</id>
            <name>Guvnor M2 Repo</name>
            <url>http://localhost:8888/kie-drools-wb/maven2/</url>
        </repository>
    </repositories>
    <dependencies>
        <!-- drools -->
        <dependency>
            <groupId>org.kie</groupId>
            <artifactId>kie-api</artifactId>
            <version>${drools.version}</version>
        </dependency>
        <dependency>
            <groupId>org.drools</groupId>
            <artifactId>drools-core</artifactId>
            <version>${drools.version}</version>
        </dependency>
        <dependency>
            <groupId>org.drools</groupId>
            <artifactId>drools-compiler</artifactId>
            <version>${drools.version}</version>
        </dependency>
        <dependency>
            <groupId>org.drools</groupId>
            <artifactId>drools-decisiontables</artifactId>
            <version>${drools.version}</version>
```

```

</dependency>
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-templates</artifactId>
    <version>${drools.version}</version>
</dependency>
<b><dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-ci</artifactId>
    <version>${drools.version}</version>
</dependency>
<!-- test -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.7</version>
    <scope>test</scope>
</dependency>
</dependencies>
</project>

```

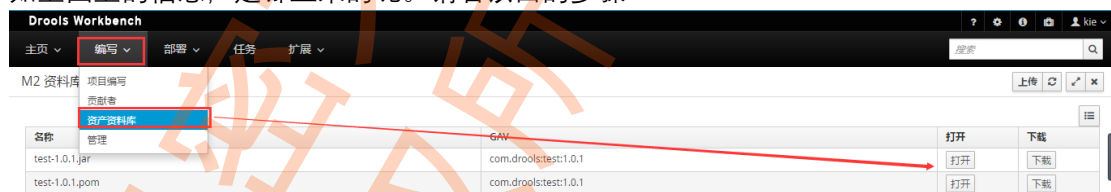
里面最重要的两个部分，小编已经用加粗方式标记出来了

```

<repositories>
  <repository>
    <id>guvnor-m2-repo</id>
    <name>Guvnor M2 Repo</name>
    <url>http://localhost:8888/kie-drools-wb/maven2/</url>
  </repository>
</repositories>

```

如上图上的信息，是哪里来的呢。请看以面的步骤

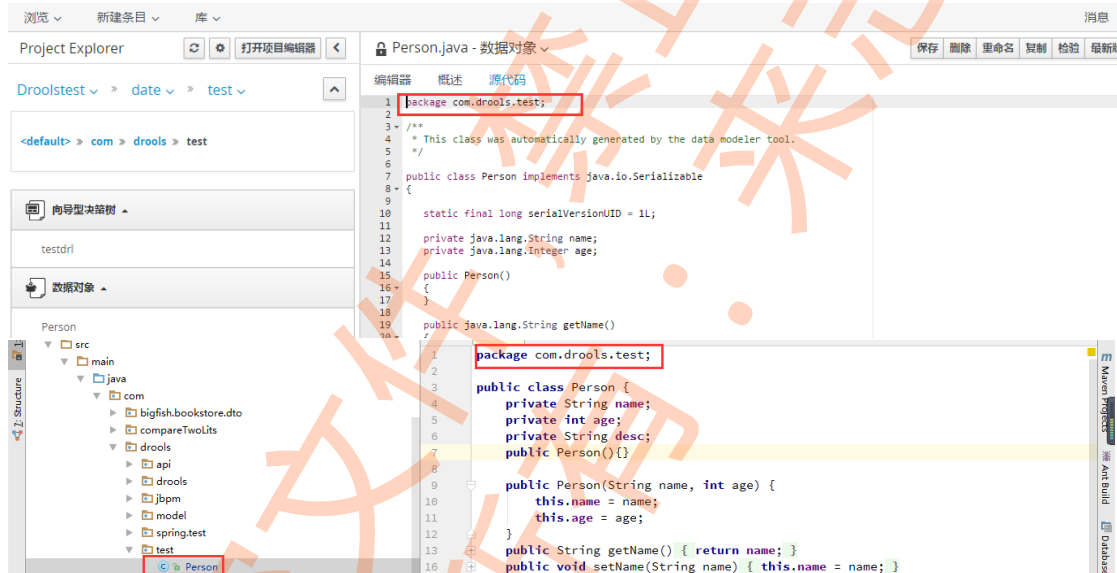




小编在前面的章节中说过。Workbench 是很多功能的结合体，我们这里所构建成功的项目，其实是生成了一个 jar 包，至于与 java 交互，相信读者也已经明白，就是为了要引用这个 jar 包。

第二个加粗部分的用途是：Java 程序通过远程地址读取 kie-drools 仓库内的规则用的就是这个包。

返回到我们的项目编辑中来，本地的项目 package 和事实文件（fact）的名称和路径一定要和远程的 kie—drools-wb 上的项目名称路径保持一致



这是相当重要的部分了，否则规则中 fact 事实是无法插入的
编写远程调用客户端的程序，代码解释

```
package com.drools.test;

import org.drools.compiler.kproject.ReleaseIdImpl;
import org.drools.core.io.impl.UrlResource;
import org.junit.Test;
import org.kie.api.KieServices;
import org.kie.api.builder.KieModule;
import org.kie.api.builder.KieRepository;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

import java.io.IOException;
```

```

import java.io.InputStream;

/**
 * Created by kangz on 2017/3/8.
 */
public class Testworkbench {

    @Test
    public void test() throws IOException {
        String url = "http://localhost:8888/kie-drools-wb/maven2/com/drools/test/1.0.1/test-1.0.1.jar";
        ReleaseIdImpl releaseId = new ReleaseIdImpl("com.drools", "test", "1.0.1");
        KieServices ks = KieServices.Factory.get();
        KieRepository kr = ks.getRepository();
        UrlResource urlResource = (UrlResource) ks.getResources().newUrlResource(url);
        urlResource.setUsername("kie");
        urlResource.setPassword("kie");
        urlResource.setBasicAuthentication("enabled");
        InputStream is = urlResource.getInputStream();
        KieModule kModule = kr.addKieModule(ks.getResources().newInputStreamResource(is));
        KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());
        KieSession kieSession = kContainer.newKieSession();
        Person p=new Person();
        p.setAge(50);
        p.setName("张三");

        kieSession.insert(p);
        int i=kieSession.fireAllRules();
        System.out.print("共执行了"+i+"条规则");
        System.out.print("修改后的结果"+p.getName());

    }

}

```

执行后的结果与我们猜想的是一样的。

```

2017-07-08 02:07:56 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_RULES
2017-07-08 02:07:56 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-07-08 02:07:56 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-07-08 02:07:56 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-07-08 02:07:56 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
共执行了1条规则修改后的结果李四
Process finished with exit code 0

```

小编就给读者分析一下

第一段代码：创建访问，找到要访问的 jar 包

```

String url = "http://localhost:8888/kie-drools-wb/maven2/com/drools/test/1.0.1/test-1.0.1.jar";
ReleaseIdImpl releaseId = new ReleaseIdImpl("com.drools", "test", "1.0.1");
KieServices ks = KieServices.Factory.get();

```



```
KieRepository kr = ks.getRepository();
```

```
public ReleaseIdImpl(String groupId,  
    String artifactId,  
    String version) {  
    this.groupId = groupId;  
    this.artifactId = artifactId;  
    this.version = version;  
}
```

第二段代码：以下代码做的事情就是相当于在 IE 中打开 KIE-DROOLS 的 Web 地址，然后输入用户名、密码并点击登录

```
UrlResource urlResource = (UrlResource) ks.getResources().newUrlResource(url);  
urlResource.setUsername("kie");  
urlResource.setPassword("kie");  
urlResource.setBasicAuthentication("enabled");  
InputStream is = urlResource.getInputStream();
```

第三段代码：访问规则，执行规则。

```
KieModule kModule = kr.addKieModule(ks.getResources().newInputStreamResource(is));  
KieContainer kContainer = ks.newKieContainer(kModule.getReleaseId());  
KieSession kieSession = kContainer.newKieSession();  
Person p=new Person();  
p.setAge(50);  
p.setName("张三");  
  
kieSession.insert(p);  
int i=kieSession.fireAllRules();
```

自动扫描

使用自动扫描功能，其实与简单方式大同小异，只是不用在 pom.xml 中设置而是在 settings.xml 中配置的

```
<repositories>  
  <repository>  
    <id>guvnor-m2-repo</id>  
    <name>Guvnor M2 Repo</name>  
    <url>http://localhost:8888/kie-drools-wb/maven2</url>  
  </repository>  
</repositories>
```

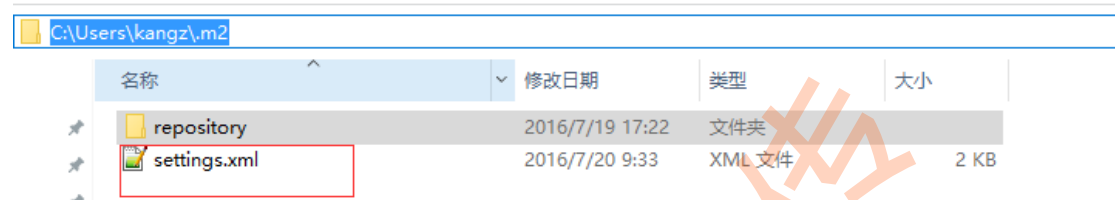
通过 Drools 的 workbench 实现与 java 的自动扫描功能。具体方式有两种，第一种是 ci-api 的形式，第二种是 Spring 整合。

因为 workbench 与 maven 相似，这里的组 ID 构建 ID 版本号就相当重要了，之前讲说通过直接引用 jar 的方式去实现 web 端的一些规则。现在是可以通过 ci-api 和 spring 的方式

进去访问。

做自动扫描前,要有几个前置条件,与简单试的前置条件是一样的,小编这里就不在重复了,直接说关键点了。

第一步:要进 maven 进行配置, 这里需要修改 settings.xml 文件



```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>guvnor-m2-repo</id>
      <username>kie</username> <!-- 这里是登录的时的帐号密码, 如果不设置将没有权限会报错 -->
      <password>kie</password>
      <configuration>
        <wagonProvider>httpclient</wagonProvider>
        <httpConfiguration>
          <all>
            <usePreemptive>true</usePreemptive>
          </all>
        </httpConfiguration>
      </configuration>
    </server>
  </servers>

  <profiles>
    <profile>
      <id>guvnor-m2-repo</id>
      <repositories>
        <repository>
          <id>guvnor-m2-repo</id>
          <name>Guvnor M2 Repo</name>
          <url>http://localhost:8888/kie-drools-wb/maven2/</url>
          <layout>default</layout>
          <releases>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
          </releases>
          <snapshots>
            <enabled>true</enabled>
```

```

        <updatePolicy>always</updatePolicy> <!--更新策略, 常常 -->
    </snapshots>
</repository>
</repositories>
<activation>
    <activeByDefault>true</activeByDefault> <!--这里要设置成 true -->
</activation>
</profile>
</profiles>
<activeProfiles>
    <activeProfile>guvnor-m2-repo</activeProfile> <!-- 这个设置也是必须要有的 -->
</activeProfiles>
</settings>

```

Java API 方式

```

@Test
public void runRules2() {
    KieServices kieServices = KieServices.Factory.get();
    ReleaseId releaseId = kieServices.newReleaseId("com.drools", "test", "1.0.1");
    KieContainer kContainer = kieServices.newKieContainer(releaseId);
    KieScanner kScanner = kieServices.newKieScanner(kContainer);
    // 启动 KieScanner 轮询 Maven 存储库每 10 秒
    kScanner.start(10000L);
    while (true) {
        try {
            KieSession kSession = kContainer.newKieSession();
            Person p = new Person();
            p.setAge(30);
            p.setName("张三");
            kSession.insert(p);
            kSession.fireAllRules();
            System.out.println(p.getName());
            Thread.sleep(1000L);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

```

```

*/
ReleaseId newReleaseId(String groupId, String artifactId, String version);

```

执行结果:

李四

```
2017-54-08 02:54:08 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_RULES
2017-54-08 02:54:08 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-54-08 02:54:08 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-54-08 02:54:08 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-54-08 02:54:08 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
```

分析以上代码：

KieServices kieServices = KieServices.Factory.get(); 获取服务 单例模式

Releaseld releaseld = kieServices.newReleaseld("com.drools", "test", "1.0.1");

后面的参数是不是很眼熟，这里就是在页面中设置的三个参数。这里通过 Releaseld 将这个 jar 获取到

KieScanner kScanner = kieServices.newKieScanner(kContainer);用于配置扫描间隔

kScanner.start(10000L); 单位是毫秒

那我们怎么知道是实现了自动扫描功能呢，很简单，不需要关闭程序，在 workbeanch 中修改规则文件，保存并重新构建。。

看输出的结果是否有变化，如果没有变化，证明配置失败，如果有变化证明配置成功

```
2017-54-08 02:54:59 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-54-08 02:54:59 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
王五
2017-55-08 02:55:00 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_RULES
2017-55-08 02:55:00 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-55-08 02:55:00 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-55-08 02:55:00 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-55-08 02:55:00 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
```

Spring 配置文件方式

要通过 spring 方式就要先建立一个 xml 文件。xml 文件如下内容

```
<?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:kie="http://drools.org/schema/kie-spring"
xsi:schemaLocation="
    http://drools.org/schema/kie-spring
    http://drools.org/schema/kie-spring.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
<kie:import releaseld-ref="springweb" enableScanner="true" scannerInterval="12000"/>
    <kie:releaseld id="springweb" groupId="com.drools" artifactId="test" version="1.0.1"/>
    <bean id="kiePostProcessor" class="org.kie.spring.annotations.KModuleAnnotationPostProcessor"/>
</beans>
```

分析一下配置文件中的内容

<kie:import releaseld-ref="springweb" enableScanner="true" scannerInterval="12000"/> enableScanner 表示是否扫描 scannerInterval 表示扫描间隔

<kie:releaseld id="springweb" groupId="com.drools" artifactId="test" version="1.0.1"/> 这里的配置就多了一个 id 这个 id 是为了配置 kie:import 引用的。

配置完之后写 java 代码：具体代码如下

```
@RunWith(SpringUnit4ClassRunner.class)
```

```
@ContextConfiguration( { "classpath:spring.xml" })
public class PersonDrools {
```

```
    @KSession("defaultKieSession")
    private KieSession kSession;

    @Test
    public void runRules() {
        while (true) {
            try {
                Person p = new Person();
                p.setAge(30);
                p.setName("张三");
                kSession.insert(p);

                int i = kSession.fireAllRules();
                System.out.println(p.getName() + i);
                Thread.sleep(10000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

执行结果

```
2017-10-08 03:10:04 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-10-08 03:10:04 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
1
王五
2017-10-08 03:10:05 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_RULES
2017-10-08 03:10:05 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-10-08 03:10:05 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-10-08 03:10:05 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-10-08 03:10:05 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
1
赵六
2017-11-08 03:11:21 [DEBUG] org.drools...DefaultAgenda - State was INACTIVE is nw FIRING_ALL_RULES
2017-11-08 03:11:21 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-11-08 03:11:21 [DEBUG] org.drools...DefaultAgenda - Fire Loop
2017-11-08 03:11:21 [DEBUG] org.drools...DefaultAgenda - State was FIRING_ALL_RULES is nw HALTING
2017-11-08 03:11:21 [DEBUG] org.drools...DefaultAgenda - State was HALTING is nw INACTIVE
```

使用 Spring 方式有以下几个注意点：

@KSession("defaultKieSession")private KieSession kSession; 这里必须
要这样写 引用的是 private KieSession xxx; 否则不会直接扫描。
@KReleaseId(groupId = "com.drools.web.test", artifactId = "drools_web_class", version = "1.6.8-SNAPSHOT")
这行代码可写可不写，如果不写就会引用 Spring 中配置好的，如果要加此行，最好与配置文件中的引用的要完全一样。否则就会替换掉配置文件中的引用，并且不会进行自动扫描。
如果在配置文件未配置 kie:improt 和 kie:kreleaseId 的话 通过注解也是可以引用到 workbeanch 的 jar 包的，但是不会进行自动扫描的。
从这段话的解释来说。我们可以这样理解

- 1、在不进行自动扫描 我们可以通过三种方式进行操作,第一种在配置文件中 kie:import 参数只写 releaseId-ref 第二种不配置 xml 通过注解的方式引用规则 第三种：通过原生 api，不设置扫描 kScanner.start(10000L);
- 2、设置自动扫描的方式，我们有两种方式，上述已经说过，这里不在说明。

版本控制

下面小编就说一下有关版本的问题

关于版本的问题，其实写法和 maven 是大同小异的，这里我推荐用两种方式

第一可通过 LATEST

第二可通过 SNAPSHOT

这第一种写法是这样的：将设置的 version="" 写成 LATEST 及可，简单的说就是将设置版本号的地方写成 LATEST

例如 `<kie:releaseId id="springweb" groupId="com.drools.web.test" artifactId="drools_web_class" version="LATEST"/>`

这样写的话 在 workbench 中设置版本号是就必须写成 x.y.z 例如：1.6.3 但如果用这个属性的话，下一次升级时，就要将版本号进行一个累加，举例说明，当前版本号是 1.6.8 下一次升级时的值是必须大于 1.6.8 的



The image shows two instances of a version input field. The first instance has the text '版本' (Version) followed by a blue information icon and a text box containing '1.6.8'. The second instance is similar but contains '1.6.9'.

这样才是生效的。

第二种写法就是这样的：将设置的 version="" 写成 1.6.8-SNAPSHOT 及可，这样的写法有一个好处就是可以将当前版本进行覆盖。但有一点的是，必须在设置版本号后面写成如下图这种形式：



The image shows a version input field with the text '版本' (Version) followed by a blue information icon and a text box containing '1.6.9-SNAPSHOT'.

以上两种方式已说明了自动扫描和全新的写法。

使用自动扫描是比较消耗服务器的，但在 6.5 版本以后，就改善了很多，占用 cpu 与内存就没有那么厉害了。读者可以自己尝试一下。

注:自动扫描的工作流程，因为 workbench 构建项目是一个标准的 maven 项目，所以小编认为，要使用这个 jar 包,则

第一步就是要下载这个 jar 包到本地仓库，其实 api 或配置文件是没有功能去下载的，真正去下载是通过 settings.xml 方式（简单方式不同）

第二步将下载后的 jar 包通过 api 或 spring.xml 方式 加载到项目中，并运行 kiesession，上在的例子中小编使用的其实都是没有配置过的默认的 kiesession,当然默认的是有状态的，这个小编在之前就已经说过了，workbench 就是一个 web 版本开发工具。

Wildfly 服务器的配置及 kie-server+workbench

前面的章节中有说过 workbench 或 kie-server 也是可以配置在 wildfly 服务器上的，下面小编就简单的说明一下如何在 wildfly 上配置 workbench+kie-server。

小编也是参考了很多博客和例子才总结出来的，与有雷同，是正常的。wildfly 服务器是

基于 Jboss 的，是一个基于 J2EE 的开放源代码的应用服务器。JBoss 代码遵循 LGPL 许可，可以在任何商业应用中免费使用，而不用支付费用。JBoss 是一个管理 EJB 的容器和服务，支持 EJB 1.1、EJB 2.0 和 EJB3 的规范。但 JBoss 核心服务不包括支持 servlet/JSP 的 WEB 容器，一般与 Tomcat 或 Jetty 绑定使用。

上面是一个简单的应用，具体的详细，请自己找。在这篇文章中，我们具体讲一下 wildfly 的 domain 方式的

1、Wildfly 下载 地址：http://wildfly.org/downloads/ 最好是能将内存设置的大一些

版本众多，小编选用的 8.2 版本，在 workbench6.5 版本中使用的是 10X 版本，

8.2.0.Final	2014-11-20	Java EE7 Full & Web Distribution	LGPL	126 MB	ZIP
				113 MB	TGZ
		Update Existing 8.1.0.Final Install	LGPL	62 MB	ZIP
		Application Server Source Code	LGPL	36 MB	ZIP
				22 MB	TGZ
		Release Notes			Notes

2、下载完成后放到 Linux 上，我存放在 /usr/local 路径下。我的主机 IP 是 192.168.80.10
安装过程：tar -zxvf wildfly-8.2.0.Final.tar.gz 并将 wildfly-8.2.0.Final 改名为 wildfly8

```
[root@localhost local]# pwd
/usr/local
[root@localhost local]# ls
jdk jdk-7u79-linux-x64.tar.gz wildfly8 wildfly-8.2.0.Final.tar.gz
```

配置 wildfly 的用户名及密码

/usr/local/wildfly8/bin 执行 ./add-user.sh

./add-user.sh Enter

What type of user do you wish to add?

- a) Management User (mgmt-users.properties)
- b) Application User (application-users.properties)

(a): a Enter

Enter the details of the new user to add.

Using realm 'ManagementRealm' as discovered from the existing property files.

Username :admin Enter 这个可自定义 但建议写 admin

Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.

- The password should not be one of the following restricted values {root, admin, administrator}
- The password should contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)
- The password should be different from the username

Password : 这里输入密码 Enter

JBAS015267: Password must have at least 1 non-alphanumeric symbol.

Are you sure you want to use the password entered yes/no? yes Enter

Re-enter Password : 这里是重复密码 Enter

What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[]: admin **Enter** 设置用户组

About to add user 'admin' for realm 'ManagementRealm'

Is this correct yes/no? **yes** **Enter**

Added user 'admin' to file '/usr/local/wildfly8/standalone/configuration/mgmt-users.properties'

Added user 'admin' to file '/usr/local/wildfly8/domain/configuration/mgmt-users.properties'

Added user 'admin' with groups admin to file '/usr/local/wildfly8/standalone/configuration/mgmt-groups.properties'

Added user 'admin' with groups admin to file '/usr/local/wildfly8/domain/configuration/mgmt-groups.properties'

Is this new user going to be used for one AS process to connect to another AS process?

e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.

yes/no? **yes** **Enter**

To represent the user add the following to the server-identities definition <secret value="YWRtaW4xMjM=" />
这个值是用来做集群的

新建用户完成之后 开始配置相关文件：我这里要单机配置，所以我们这里只关心一个配置文件

cd /usr/local/wildfly8/domain/configuration 进入到这个目录下

编辑 host.xml

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

<host name="master" xmlns="urn:jboss:domain:2.2">

  <management>
    <security-realms>
      <security-realm name="ManagementRealm">
        <authentication>
          <local default-user="$local" skip-group-loading="true" />
          <properties path="mgmt-users.properties" relative-to="jboss.domain.config.dir"/>
        </authentication>
        <authorization map-groups-to-roles="false">
          <properties path="mgmt-groups.properties" relative-to="jboss.domain.config.dir"/>
        </authorization>
      </security-realm>
      <security-realm name="ApplicationRealm">
        <authentication>
          <local default-user="$local" allowed-users="*" skip-group-loading="true" />
          <properties path="application-users.properties" relative-to="jboss.domain.config.dir" />
        </authentication>
      </security-realm>
    </security-realms>
  </management>
</host>
```

```

        </authentication>
        <authorization>
            <properties path="application-roles.properties" relative-
to="jboss.domain.config.dir"/>
        </authorization>
    </security-realm>
</security-realms>
<audit-log>
    <formatters>
        <json-formatter name="json-formatter"/>
    </formatters>
    <handlers>
        <file-handler name="host-file" formatter="json-formatter" relative-
to="jboss.domain.data.dir" path="audit-log.log"/>
        <file-handler name="server-file" formatter="json-formatter" relative-
to="jboss.server.data.dir" path="audit-log.log"/>
    </handlers>
    <logger log-boot="true" log-read-only="false" enabled="false">
        <handlers>
            <handler name="host-file"/>
        </handlers>
    </logger>
    <server-logger log-boot="true" log-read-only="false" enabled="false">
        <handlers>
            <handler name="server-file"/>
        </handlers>
    </server-logger>
</audit-log>
<management-interfaces>
    <native-interface security-realm="ManagementRealm">
        <socket interface="management" port="${jboss.management.native.port:9999}"/>
    </native-interface>
    <http-interface security-realm="ManagementRealm" http-upgrade-enabled="true">
        <socket interface="management" port="${jboss.management.http.port:9990}"/>
    </http-interface>
</management-interfaces>
</management>

<domain-controller>
    <local/>
    <!-- Alternative remote domain controller configuration with a host and port -->
    <!-- <remote host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9999}"
security-realm="ManagementRealm"/> -->
</domain-controller>

```

```

<interfaces>
  <interface name="management">
    <inet-address value="${jboss.bind.address.management:192.168.80.10}"/>
  </interface>
  <interface name="public">
    <inet-address value="${jboss.bind.address:192.168.80.10}"/>
  </interface>
  <interface name="unsecure">
    <!-- Used for IIOP sockets in the standard configuration.
         To secure JacORB you need to setup SSL -->
    <inet-address value="${jboss.bind.address.unsecure:192.168.80.10}"/>
  </interface>
</interfaces>

<jvms>
  <jvm name="default">
    <heap size="64m" max-size="256m"/>
    <permgen size="256m" max-size="256m"/>
    <jvm-options>
      <option value="-server"/>
    </jvm-options>
  </jvm>
</jvms>

<servers>
  <server name="server-one" group="main-server-group">
  </server>
</servers>
</host>

```

注意，要与你自己的 host.xml 与我写的进行对比，看一下哪里有不同；配置完成后保存

3、配置服务：

```
vim /etc/default/wildfly.conf
```

jdk 的安装路径

```
JAVA_HOME="/usr/local/jdk"
```

JBOSS_HOME 是 Wildfly 的安装根目录 之前为什么要进行改名 就是要用在这里

```
JBOSS_HOME="/usr/local/wildfly8"
```

这里需要改为执行当前登录 Linux 系统的用户名

```
JBOSS_USER=root
```

指定运行模式为 domain

```
JBOSS_MODE=domain
```

指定 domain 的配置文件为 domain.xml，slave 的配置文件为 host.xml

```
JBOSS_DOMAIN_CONFIG=domain.xml
```

JBOSS_HOST_CONFIG=host.xml

4、为系统配置服务

##将.sh 启动命令放到系统目录下

```
cp wildfly8/bin/init.d/wildfly-init-redhat.sh /etc/init.d/wildfly
```

##增加执行权限

```
chmod +x /etc/init.d/wildfly
```

##增加系统服务

```
chkconfig --add wildfly
```

##设置开机启动

```
chkconfig wildfly on
```

##启动 wildfly，记得先启动 master

```
service wildfly start
```

5、下面我们就可以进行验证了

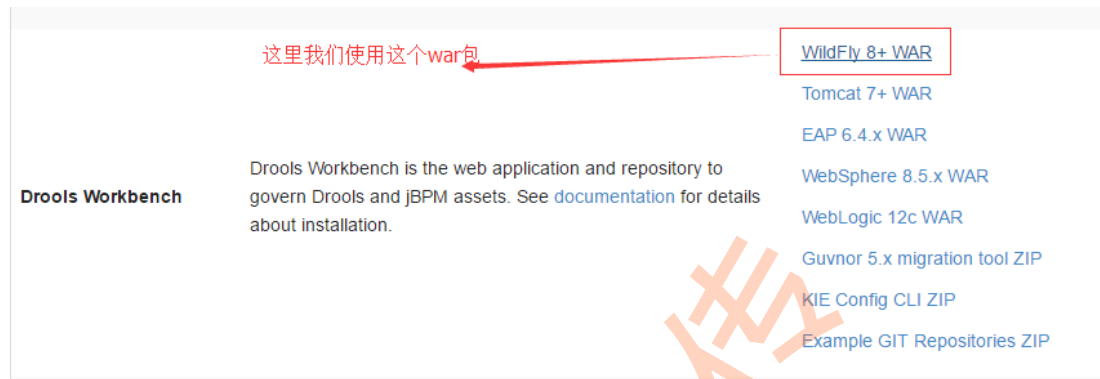
<http://192.168.80.10:9990/console/App.html> 访问地址



上述小编将了如何搭建 wildfly domian 的方式下面小编再讲述一下如何在 wildfly 中使用 kie-server+workbench

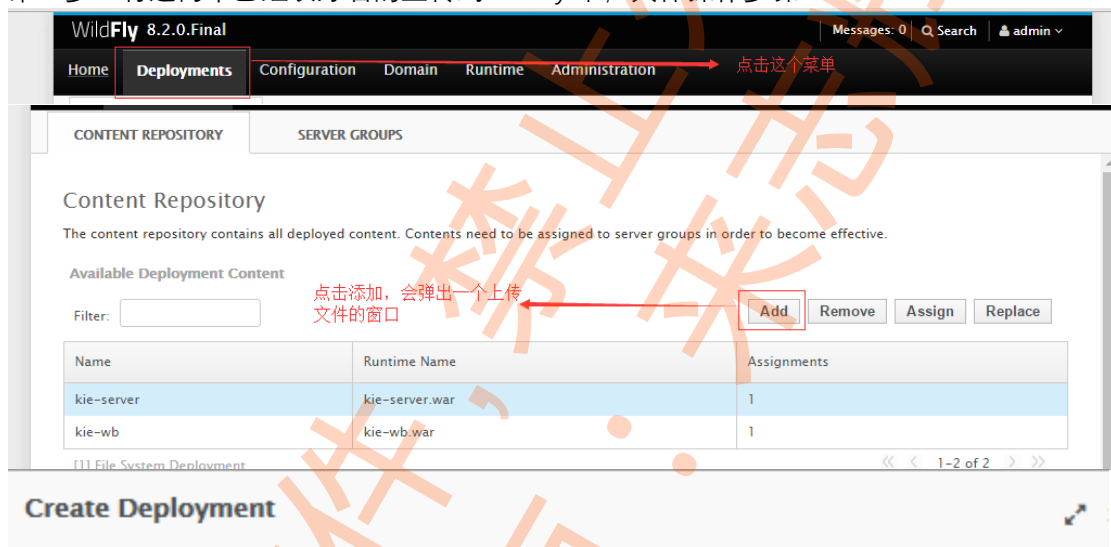
第一步：就是下载 wildfly 所用的两个 war 包，及 kie-server,workbench

<http://www.drools.org/download/download.html> 下载地址。之前有讲过 kie-server 的安装在 tomcat 下。这里发生的变化只是不在用 tomcat workbench war 包



和之前一样 将下载好的 war 包 改名成 将 wildfly8.war 改成 kie-wb.war 将 kie-server-6.4.0.Final-ee7.war 修名为 kie-server.war 只是我们这里不通过解压,而是通过在下载后的路径下进行修改名称

第二步: 将这两个已经改好名的上传到 wildfly 中, 具体操作步骤:



Step 2/2: Verify Deployment Names

[Need Help?](#)

Name:

Runtime Name:

上传文件后点击一下步然后会出现这个页面, 这里要注意的是 Runtime Name 的名最好与上传文件名保持一致, 至于Name值可随意, 但最好走默认的就好

Available Deployment Content

Filter:

全部上传完之后 选择其中一个 点击发布, 选择 main-server-group 就好;

Add Remove Assign Replace

Name	Runtime Name	Assignments
kie-server	kie-server.war	1
kie-wb	kie-wb.war	1

发布成功后这里会变成1

[1] File System Deployment << < 1-2 of 2 > >>

然后我们选择下面这个菜单完成配置文件的设置, 这里的配置与 setenv.sh 很像

System Properties

These properties are available throughout the configuration. The Boot-Time flag specifies if a property should be passed into the JVM start (-Dproperty=value)

Add Remove

java.net.preferIPv4Stack	true 这个是默认的	true
org.kie.demo	false	true
org.kie.example	false	true
org.kie.server.controller	http://192.168.80.10:8080/kie-wb/rest/controll er	true
org.kie.server.controller.pwd	kieserver	true
org.kie.server.controller.user	kieserver	true
org.kie.server.id	wildfly-kieserver	true
org.kie.server.location	http://192.168.80.10:8080/kie-server/services/r est/server	true
org.kie.server.persistence.dialect	org.hibernate.dialect.H2Dialect	true
org.kie.server.persistence.ds	java:jboss/datasources/ExampleDS	true
org.kie.server.persistence.tm	org.hibernate.service.jta.platform.internal.JBossA ppServerJtaPlatform	true

这些配置完成后 要不急, 还要去服务器上创建新的用户。即 workbench 和 kie-server 的用户。

配置过程与上面说的一样。 只是之前是系统用户, 这次我们要创建应用用户, 将之前写的 a 改成 b 用户类型 后面的步骤是一样的。但在创建 kie-server 用户时要注意, 它的分组必须是 kie-server。在 tomcat 里 用户名和密码都必须是默认的, 但在 wildfly 里 是可以改变的, 如果没有设置的话, 就必须是默认用户及密码。

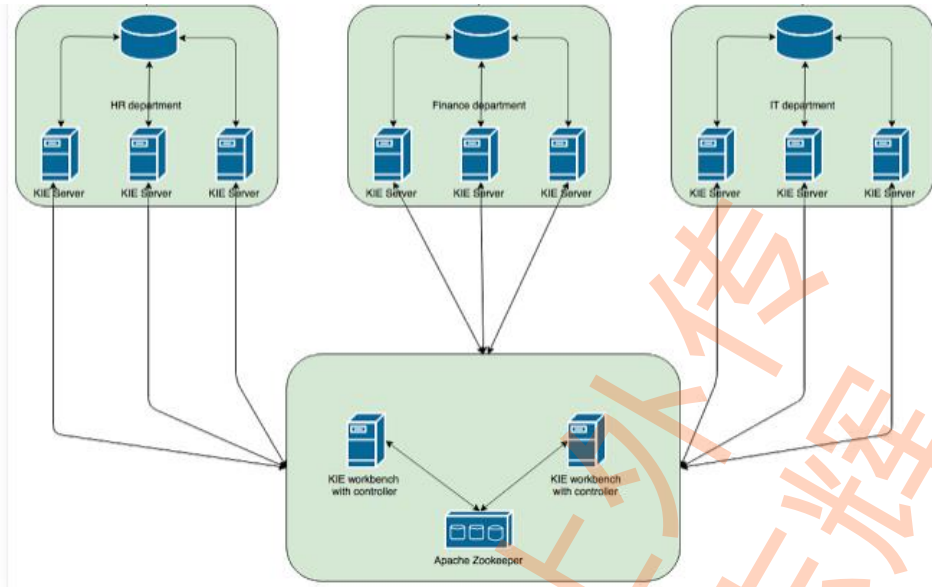
完成所以步骤后, 重启服务, 和之前介绍 workbench 一样 直接打开就可以了, 地址 http://192.168.80.10:8080/kie-wb

页面正常打开就成功了, 如果页面不能正常访问, 则肯定是有步骤出错了, 请重新配置就好。

Wildfly 集群部署

wildfly 服务器的集群的特点, 重点在这个集群方面, 那我们要实现一个怎样的效果呢, 其实很简单, 我们举例说明:

首先我们先看一下 集群的结构图



我们这里设置了 5 台机器，分别为：

10.0.5.213(kie-wb1——workbench)服务器

10.0.5.119(kie-wb2——workbench)服务器

10.0.5.207(kie-server)服务器

10.0.5.208(kie-server)服务器

10.0.5.84 (zookeeper+Helix)服务器

配置起来其实很简单，在上一章节中，我们介绍了 workbench 与 kie-server 分布式的部署，相信大家对此已经有所了解只要将已经配置成功的 kie-wb 与 kie-server 服务器进行克隆就好了，这里就可以省去我们很多的麻烦。

然后我们对 zookeeper+Helix 所在的服务器进行配置：

- 1、安装 zookeeper，安装过程 略 [下载地址](#) 我们这里选用的是 3.4.6 版本
- 2、安装 Helix，安装过程 略 [下载地址](#) 我们这里选用的是 0.6.5 版本
- 3、将 zookeeper-3.4.6/conf/的 zoo_sample.cfg 改名为 zoo.cfg 不用进行其他设置
- 4、启动 zookeeper `./zkServer.sh start` 在 \$ZOOKEEPER_HOME/bin 目录下执行
- 5、配置 Helix 服务器
 - 到指定的目录下
 - `cd $HELIX_HOME/bin`
 - 创建集群：
 - `./helix-admin.sh --zkSvr localhost:2181 --addCluster kie-cluster`
 - zookeeper 服务器使用的 zkSvr 值必须匹配。集群名称(kie-cluster)可以根据需要改变。
 - 将节点添加到集群：
 - `# Node 1 $./helix-admin.sh --zkSvr localhost:2181 --addNode kie-cluster nodeOne:12345 # Node 2 $./helix-admin.sh --zkSvr localhost:2181 --addNode kie-cluster nodeTwo:12346 ...` 这里要注意一下，其实这个添加只要在 zookeeper 所在机器上做就可以了，节点的值是唯一的
 - 添加资料到集群
 - `./helix-admin.sh --zkSvr localhost:2181 --addResource kie-cluster vfs-repo 1 LeaderStandby AUTO_REBALANCE`
 - 平衡集群的初始化：

- o ./helix-admin.sh --zkSvr localhost:2181 --rebalance kie-cluster
vfs-repo 2
- 开始 Helix 控制器管理集群:
 - o ./run-helix-controller.sh --zkSvr localhost:2181 --cluster kie-cluster 2>&1 > /tmp/controller.log &

6、配置 workbench 所在服务器 在机器 1 上配置

```
<server name="server-one" group="main-server-group">
  <!--配置属性 -->
  <system-properties>
    <property name="jboss.node.name" value="nodeOne" boot-time="false"/>
    <property name="org.uberfire.nio.git.dir" value="/tmp/kie/nodeone" boot-time="false"/>
    <property name="org.uberfire.metadata.index.dir" value="/tmp/kie/nodeone" boot-time="false"/>
    <property name="org.uberfire.cluster.id" value="kie-cluster" boot-time="false"/>
    <property name="org.uberfire.cluster.zk" value="10.0.5.84:2181" boot-time="false"/>
    <property name="org.uberfire.cluster.local.id" value="nodeOne_12345" boot-time="false"/>
    <property name="org.uberfire.cluster.vfs.lock" value="vfs-repo" boot-time="false"/>
    <property name="org.kie.demo" value="false" boot-time="false" />
    <property name="org.kie.example" value="false" boot-time="false"/>
    <property name="org.uberfire.nio.git.daemon.enabled" value="true" boot-time="false"/>
    <property name="org.uberfire.nio.git.daemon.host" value="10.0.5.213" boot-time="false" />
    <property name="org.uberfire.nio.git.daemon.port" value="9418" boot-time="false"/>
    <property name="org.kie.server.user" value="kieserver" boot-time="false" />
    <property name="org.kie.server.pwd" value="kieserver" boot-time="false" />
    <property name="org.guvnor.m2repo.dir" value="/root/.m2/repository" boot-time="false" />
  </system-properties>
</server>
```

那读者有没有发现什么呢，其实配置我们就加了 org.uberfire.cluster.xxx 的几行配置，但要注意的，有些地方还是要进行修改的，我这里只是简单的说明一下： 将机器 1 上的配置完全复制到机器 2 上， 并将 org.uberfire.nio.git.daemon.host 的值设置为机器 2 上的地址， org.uberfire.cluster.local.id 的值设置为 nodeTwo_12346，这样 你是没有发现了什么，对，我们将其余的 One or one 全部改为 Two or tow,就 OK 啦。

7、配置 kie-server 所在的服务器,还是对 host.xml 进行配置

```
<server name="server-one" group="main-server-group">
  <system-properties>
    <!--配置本机地址-->
    <property name="org.kie.server.location" value="http://10.0.5.207:8080/kie-server/services/rest/server"
boot-time="false"/>
    <property name="org.kie.server.id" value="wildfly-kieserver1" boot-time="false"/>
    <property name="org.kie.server.controller.user" value="kieserver" boot-time="false"/>
    <property name="org.kie.server.controller.pwd" value="kieserver" boot-time="false"/>
    <!--配置 kie-web 地址 failover 策略-->
```

```

<property name="org.kie.server.controller" value="http://10.0.5.213:8080/kie-
wb/rest/controller,http://10.0.5.119:8080/kie-wb/rest/controller" boot-time="false"/>

<property name="org.kie.server.persistence.dialect" value="org.hibernate.dialect.H2Dialect" boot-
time="false"/>

<property name="org.kie.server.persistence.ds" value="java:jboss/datasources/ExampleDS" boot-
time="false"/>

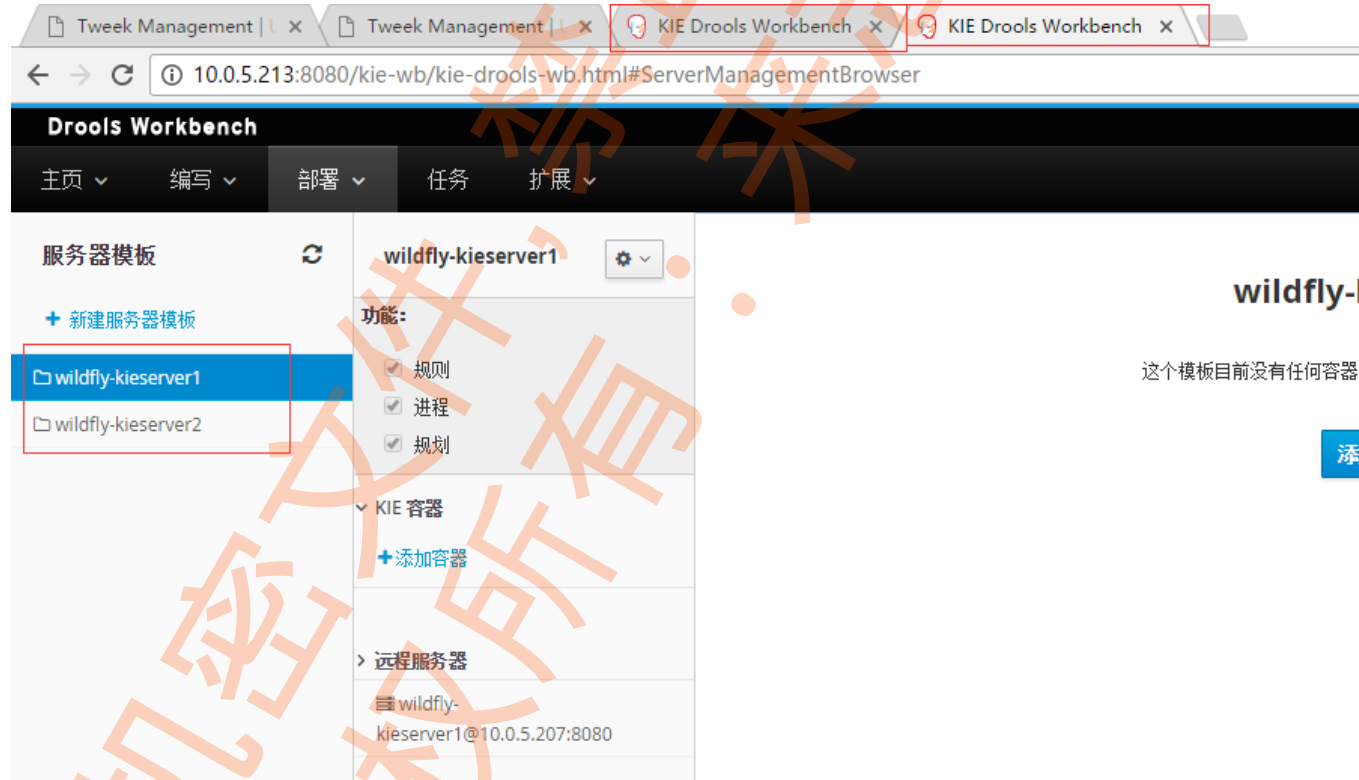
<property name="org.kie.server.persistence.tm"
value="org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform" boot-time="false"/>

<property name="kie.maven.settings.custom" value="/root/.m2/settings.xml" boot-time="false" />

</system-properties>
</server>

```

我们来简单的说明一下 org.kie.server.id 值是可变是，在 kie-server 机器上如果设置 value="xxx1" 则我们也可能在 kie-server 机器 2 上设置 value="xxx2" 这样的设置 其实是将 kie-server 模板进行了分组，并且每个服务过都是独立的， 还有一个区别在于 org.kie.server.controller 的设置，这个值，我们很清楚，这个属性是用来做 kie-wb 请求的，如果这里设置了多个有效的 kie-wbURL，则在页面上的体现是这样的



动态规则的说明及解决方案

动态规则，是做规则引擎最想知道的问题，小编也加过一些群，每一个人都喜欢问这样的问题，动态规则怎么实现，小编在这一章节中，就给读者们介绍一下所谓动态规则是什么意思。

在小编的认知中，动态规则可分为以下几种方案

1. 拼接规则语法，形成完成的规则内容，通过 `string` 方式调用规则
2. 通过模板方式对规则进行修改
3. `workbench` 自动扫描
4. `workbench` 整合 `kie-server`
5. 通过执行规则文件执行规则
6. 动态创建 `kjar`
7. 官方推荐方式

Drools 部分扩展

扩展一：关于 Conditional Element (CE) 有条件元素的

在使用 `fireAllRules` 调用规则时，如果未在方法中指定执行哪些规则名的话，方法 `fireAllRules` 会执行所以满足条件的规则，如果设置了指定规则名的执行方式，那么会执行所指定规则名的规则，但 LHS 部分因 CE 的机制，也会被执行，但未被指定的规则名则不会执行 RHS 部分。

扩展二：上传 jar 包，添加依赖关系或者是用在 kie-

server 容器的新建

一般情况，我们都是从资料库中添加相关的 jar 包，这些 jar 有两种上传的方式：

方法一：如下图所示，我们通过上传文件方式进行操作，该 jar 包可以用在依赖关系也可以用在 `kie-server` 创建容器，可用于发部与部署服务器的功能



方法二：我们可以通过直接将 jar 包放在 workbench 构建生成 jar 包目录的地方，其实功能是与 workbench 是一样的，但强烈建议使用通过 workbench 来进行上传，这样可检测错误

扩展三：如果将 workbench 构建时生成的 jar 直接放到公司所在的私服，其实这个不只是针对 workbench 来说的，更准确的说是关于 maven 的配置，在 pom.xml 中加入

```
<distributionManagement>
<repository>
<id>thirdparty</id>
<name>Nexus Release Repository</name>
<url>http://localhost:8081/nexus/content/repositories/thirdparty/</url>
</repository>
</distributionManagement>
```

地址为公司私服的地址，但要注意的是，要配置权限在 settings.xml 中

码农 1 号博客专栏 <http://blog.csdn.net/column/details/13994.html> 码农 1 号 Drools 视频学习地址 <http://edu.csdn.net/course/detail/5583/103397>

码农 1 号《Drools 技术指南加密版》博客地址 <http://blog.csdn.net/u013115157/article/details/64123846>

特此声明：

小编出新版本的 Drools 技术指南的文档啦，相比之前的免费版添加了如下特点：

1. 内容更加全面，更多细节，更多知识点全部在新版本中有所体现，并且会附

上小编做的例子

2. 目前国内最全的 Drools 中文文档。也是目前国内唯一一本，

3. 凡购买文档者，都将被标记为会员，可享受今后所更新的文档

内容更全更精彩，请抓紧时间，机不可失，可添加小编的微信：

kangzuguan 联系我，购买成功后，拉入微信会员群。

4.Drools 规则引擎 VIP 群，只有购买了技术指南的小伙伴才能加入的群。

5.621361960，593177274 两个 Drools 技术讨论群

经过小编的不懈努力，结合了自己的认知和笔记，用时两周将些文档编辑成功，读者如果心疼小编，就给一些辛苦费啥的，小编后续会继续努力将底层实现和算法的知识也写进来的

文章是不可编辑的，读者要是看到哪里有问题或出错的地方，可随时与小编联系，小编的 QQ 是 448998253 邮箱是 kangzuguan@qq.com

下面是小编的微信转帐二维码，小编再次谢谢读者的支持，小编会更努力的

推荐使用微信支付

¥18.88



Morgan Lai(**辉)



微信支付