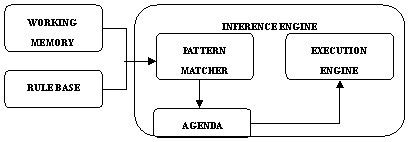
# 规则引擎

## 什么是规则引擎

规则引擎是一种嵌套在应用程序中的组件，它实现了将业务规则从应用程序代码中分离出来。规则引擎使用特定的语法编写业务规则，规则引擎可以接受数据输入、解释业务规则、并根据业务规则做出相应的决策。

Java规则引擎起源于基于规则的专家系统，而基于规则的专家系统又是专家系统的其中一个分支。专家系统属于人工智能的范畴，它模仿人类的推理方式，使用试探性的方法进行推理，并使用人类能理解的术语解释和证明它的推理结论。为了更深入地了解Java规则引擎，下面简要地介绍基于规则的专家系统。RBES包括三部分：Rule Base（knowledge base）、Working Memory（fact base）和Inference Engine。它们的结构如下系统所示：



如图所示，推理引擎包括三部分：模式匹配器（Pattern Matcher）、议程（Agenda）和执行引擎（Execution Engine）。推理引擎通过决定哪些规则满足事实或目标，并授予规则优先级，满足事实或目标的规则被加入议程。模式匹配器决定选择执行哪个规则，何时执行规则；议程管理模式匹配器挑选出来的规则的执行次序；执行引擎负责执行规则和其他动作。 和人类的思维相对应，推理引擎存在两者推理方式：演绎法（Forward-Chaining）和归纳法（Backward-Chaining）。演绎法从一个初始的事实出发，不断地应用规则得出结论（或执行指定的动作）。而归纳法则是根据假设，不断地寻找符合假设的事实。Rete算法是目前效率最高的一个Forward-Chaining推理算法，许多Java规则引擎都是基于Rete算法来进行推理计算的。 推理引擎的推理步骤如下：

(1)将初始数据（fact）输入Working Memory。

(2)使用Pattern Matcher比较规则库（rule base）中的规则（rule）和数据（fact）。

(3)如果执行规则存在冲突（conflict），即同时激活了多个规则，将冲突的规则放入冲突集合。

(4)解决冲突，将激活的规则按顺序放入Agenda。

(5)使用执行引擎执行Agenda中的规则。

上述即是规则引擎的原始架构，Java规则引擎就是从这一原始架构演变而来的。

## 引入规则引擎后带来的好处

1. 实现业务逻辑与业务规则的分离，实现业务规则的集中管理

2. 可以动态修改业务规则，从而快速响应需求变更

3. 使业务分析人员也可以参与编辑、维护系统的业务规则

4. 使用规则引擎提供的规则编辑工具，使复杂的业务规则实现变得的简单

# Drools规则引擎

Drools是Jboss公司旗下一款开源的规则引擎，它基于Rete算法的开源业务规则引擎；提供了强大的Eclipse Plugin开发支持；通过使用其中的DSL(Domain Specific Language)，可以实现用自然语言方式来描述业务规则，使得业务分析人员也可以看懂业务规则代码。

与Drools功能类似的同类开源产品主要有：OpenRules、OpenLexicon等，商业产品功能比较强。

Drools官网：<http://www.jboss.org/drools>

Drools and jBPM consist out of several projects:（Drools软件包提供的几个部分的功能）

[[http://jboss.org/file-access/default/members/drools/freezone/guvnor_icon_32x.png](http://www.jboss.org/drools/drools-guvnor.html)Drools Guvnor (Business Rules Manager)](http://www.jboss.org/drools/drools-guvnor.html) （规则集管理器）

[[http://jboss.org/file-access/default/members/drools/freezone/expert_icon_32x.png](http://www.jboss.org/drools/drools-expert.html)Drools Expert (rule engine)](http://www.jboss.org/drools/drools-expert.html)                      （规则引擎）

[[http://jboss.org/file-access/default/members/drools/freezone/flow_icon_32x.png](http://www.jboss.org/drools/drools-flow.html)jBPM 5 (process/workflow)](http://www.jboss.org/drools/drools-flow.html)                       （工作流）  
[[http://jboss.org/file-access/default/members/drools/freezone/fusion_icon_32x.png](http://www.jboss.org/drools/drools-fusion.html)Drools Fusion (event processing/temporal reasoning)](http://www.jboss.org/drools/drools-fusion.html)   （规则集引擎搜索与规划）

[[https://www.jboss.org/dms/drools/images/planner_icon_32px.png](http://www.jboss.org/drools/drools-planner.html)Drools Planner (automated planning)](http://www.jboss.org/drools/drools-planner.html)        （决策表）

## Drools API介绍

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。完成这些工作的API 主要有KnowledgeBuilder、KnowledgeBase、StatefulKnowledgeSession、StatelessKnowledgeSession、等，它们起到了对规则文件进行收集、编译、查错、插入fact、设置global、执行规则或规则流等作用。

### KnowledgeBuilder

KnowledgeBuilder 的作用就是用来在业务代码当中收集已经编写好的规则， 然后对这些规则文件进行编译， 最终产生一批编译好的规则包（KnowledgePackage）给其它的应用程序使用。KnowledgeBuilder 在编译规则的时候可以通过其提供的hasErrors()方法得到编译规则过程中发现规则是否有错误，如果有的话通过其提的getErrors()方法将错误打印出来，以帮助我们找到规则当中的错误信息。创建 KnowledgeBuilder 对象使用的是KnowledgeBuilderFactory 的newKnowledgeBuilder方法。如下:

**public** **class** KnowledgeBuilderTest {

**public** **static** **void** main(String[] args) {

KnowledgeBuilder builder = KnowledgeBuilderFactory

.*newKnowledgeBuilder*();

builder.add(**new** ClassPathResource("test.drl",

KnowledgeBuilderTest.**class**), ResourceType.*DRL*);

//打印编译错误日志

**if** (builder.hasErrors()) {

System.*out*.println("规则中存在错误，错误消息如下：");

KnowledgeBuilderErrors kbuidlerErrors = builder.getErrors();

**for** (Iterator iter = kbuidlerErrors.iterator(); iter.hasNext();) {

System.*out*.println(iter.next());

}

}

// 产生规则包的集合

Collection<KnowledgePackage> knowledgePackage = builder

.getKnowledgePackages();

}

}

### KnowledgeBase

KnowledgeBase 是Drools 提供的用来收集应用当中知识（knowledge）定义的知识库对象，在一个KnowledgeBase 当中可以包含普通的规则（rule）、规则流(rule flow)、函数定义(function)、用户自定义对象（type model）等。

创建一个KnowledgeBase 要通过KnowledgeBaseFactory 对象提供的newKnowledgeBase（）方法来实现，这其中创建的时候还可以为其指定一个KnowledgeBaseConfiguration 对象，KnowledgeBaseConfiguration 对象是一个用来存放规则引擎运行时相关环境参数定义的配置对象。

代码见KnowledgeBuilderTest.java

### StatefulKnowledgeSession

在Drools5 当中提供了两个对象与规则引擎进行交互：StatefulKnowledgeSession

和StatelessKnowledgeSession。

StatefulKnowledgeSession 对象是一种最常用的与规则引擎进行交互的方式，它可以与规

则引擎建立一个持续的交互通道，在推理计算的过程当中可能会多次触发同一数据集。在代码当中，最后使用完StatefulKnowledgeSession 对象之后，一定要调用其dispose()方法释放资源。

StatefulKnowledgeSession 可以接受外部插入（insert）的业务数据——也叫fact，一个fact 对象通常是一个普通的Java 的POJO，一般它们会有若干个属性，每一个属性都会对应getter 和setter 方法，用来对外提供数据的设置与访问。一般来说，在Drools 规则引擎当中，fact 所承担的作用就是将规则当中要用到的业务数据从应用当中传入进来，对于规则当中产生的数据及状态的变化通常不用fact 传出。如果在规则当中需要有数据传出，那么可以通过在StatefulKnowledgeSession 当中设置global 对象来实现，一个global 对象也是一个普通的Java 对象，在向StatefulKnowledgeSession 当中设置global 对象时不用insert 方法而用setGlobal 方法实现。

创建一个StatefulKnowledgeSession 要通过KnowledgeBase 对象来实现。

代码见KnowledgeBaseTest.java

### StatelessKnowledgeSession

StatelessKnowledgeSession 的作用与StatefulKnowledgeSession 相仿，它们都是用来接收业务数据、执行规则的。事实上，StatelessKnowledgeSession 对StatefulKnowledgeSession 做了包装，使得在使用StatelessKnowledgeSession 对象时不需要再调用dispose()方法释放内存资源了。

因为StatelessKnowledgeSession 本身所具有的一些特性，决定了它的使用有一定的局限性。在使用StatelessKnowledgeSession 时不能进行重复插入fact 的操作、也不能重复的调用fireAllRules()方法来执行所有的规则，对应这些要完成的工作在StatelessKnowledgeSession当中只有execute(…)方法，通过这个方法可以实现插入所有的fact 并且可以同时执行所有的规则或规则流，事实上也就是在执行execute(…)方法的时候就在StatelessKnowledgeSession内部执行了insert()方法、fireAllRules()方法和dispose()方法。

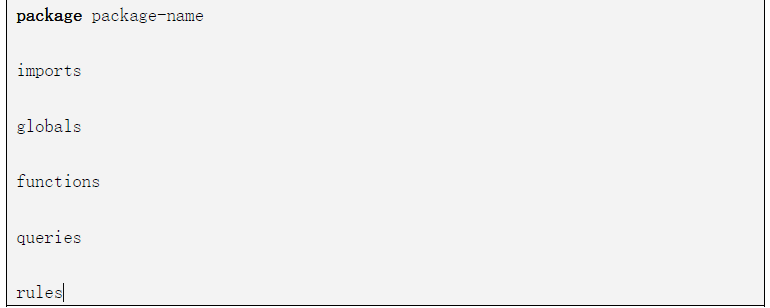
## Fact 对象

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 中变成Fact 之后，Fact 对象不是对原来的JavaBean 对象进行Clon，而是原来JavaBean 对象的引用。规则在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，这样在规则当中就可以通过对Fact 对象数据的读写，从而实现对应用数据的读写操作。一个Fact 对象通常是一个具有getter 和setter 方法的POJO 对象，通过这些getter 和setter 方法可以方便的实现对Fact 对象的读写操作，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。

## 规则

### 规则文件

在 Drools 当中，一个标准的规则文件就是一个以“.drl”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。



对于一个规则文件而言，首先声明package 是必须的，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，也有不同之处。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

一个规则文件可以存放若干个规则，每一个规则通过规则名称来进行标识。格式如下：



一个规则通常包括三个部分：属性部分（attribute）、条件部分（LHS）和结果部分（RHS）。

#### 条件部分

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 部分为空的话，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。

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。

一个pattern 的语法如：[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；

单个pattern 的情形：

rule "rule1"

when

$customer:Customer()

then

<action>…

End

规则中“$customer”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

多个pattern 的情形：

rule "rule1"

when

$customer:Customer(age>20,gender==’male’)

Order(customer==$customer,price>1000)

then

<action>…

End

规则就包含两个pattern，第一个pattern 有三个约束，分别是：对象类型必须是Cutomer；同时Cutomer 的age 要大于20 且gender 要是male；第二个pattern 也有三个约束，分别是：对象类型必须是Order，同时Order 对应的Cutomer 必须是前面的那个Customer 且当前这个Order 的price 要大于1000。在这两个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 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。

##### 约束链接

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，上述规则的LHS 部分的两个pattern 就里对象内部约束就采用“,”来实现，“&&”（and）、“||”(or)和“,”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)和“,”。

rule "rule1"

when

$customer:Customer(age>20||gender==’male’ && city==’sh’)

then

<action>…

End

规则的LHS 部分只有一个pattern，在这个pattern 中有四个约束，首先必须是一个Customer 对象，然后要么该对象gender=’male’且city=’sh’，要么age>20，在Customer 对象的字段约束当中，age>20 和gender=’male’且city=’sh’这两个有一个满足就可以了，这是因为“&&”连接符的优先级要高于“||”，gender==’male’ && city==’sh’用一小括号括起来，这样看起来会更加直观。

##### 比较操作符

在前面规则例子当中，我们已经接触到了诸如“>”、“= =”之类的比较操作符，在Drools5当中共提供了十二种类型的比较操作符，分别是：>、>=、<、<=、= =、!=、contains、not contains、memberof、not memberof、matches、not matches；

###### Contains

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

**rule** "rule1"

**when**

$order:Order();

$customer:Customer(age >20, orders **contains** $order);

**then**

System.out.println($customer.getName());

**end**

###### 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])

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.

**global** String[] orderNames;

**rule** "rule1"

**when**

$order:Order(name **memberOf** orderNames);

**then**

System.out.println($order.getName());

**end**

###### 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，但有一点需要注意，那就是正则表达式字符串当中不用考虑“\”的转义问题。matches 使用语法如下：Object(fieldName matches “正则表达式”)

**import** java.util.List;

**rule** "rule1"

**when**

$customer:Customer(name **matches** "李.\*");

**then**

System.out.println($customer.getName());

**end**

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

#### 结果部分

条件部分又被称之为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 和retract 就可以实现对当前Working Memory中的Fact 对象进行新增、删除或者是修改；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WorkingMemory 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。

###### Insert

函数insert 的作用与我们在Java 类当中调用StatefulKnowledgeSession 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。它的基本用法如: 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、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 或retract 容易发生死循环.

###### update

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。

第一种用法，直接更新一个Fact 对象,格式如update(new Object());

第二种用法的格式如update(new FactHandle(),new Object());

###### retract

和StatefulSession 的retract 方法一样，宏函数retract 也是用来将Working Memory 当中某个Fact 对象从Working Memory 当中删除，格式如retract (new Object())

###### drools

drools 宏对象能实现对当前的Working Memory 控制，通过使用drools 宏对象可以实现在规则文件里直接访问WorkingMemory。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WorkingMemory，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。

###### modify

modify 是一个表达式块，它可以快速实现对Fact 对象多个属性进行修改，修改完成后会自动更新到当前的Working Memory 当中。它的基本语法格式如：

modify(fact-expression){

<修改Fact 属性的表达式>[,<修改Fact 属性的表达式>\*]

}

#### 属性部分

规则属性是用来控制规则执行的重要工具，在前面举出的关于规则的例子当中，已经接触了如控制规则执行优先级的salience，和是否允许规则执行一次的no-loop 等。在目前的Drools5 当中，规则的属性共有13 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、when，这些属性分别适用于不同的场景

http://liureying.blog.163.com/blog/static/615135201111141012748/?COLLCC=4182910789&

#### 函数

函数是定义在规则文件当中一段码块，作用是将在规则文件当中若干个规则都会用到的业务操作封装起来，实现业务代码的复用，减少规则编写的工作量。函数的编写位置可以是规则文件当中package 声明后的任何地方，Drools 当中函数声明的语法格式如：

**function void/Object** functionName(Type arg...) {

/\*函数体的业务代码\*/

}

#### 查询

查询是Drools 当中提供的一种根据条件在当前的WorkingMemory 当中查找Fact 的方法。查询是定义在规则文件当中，和函数一样，查询的定义可以是package 语句下的任意位置，在Drools 当中查询可分为两种：一种是不需要外部传入参数；一种是需要外部传入参数。

在 Drools 当中查询以query 关键字开始，以end 关键字结束，在package 当中一个查询要有唯一的名称，查询的内容就是查询的条件部分，条件部分内容的写法与规则的LHS 部分写法完全相同。查询的格式如下：

query "query name"

#conditions

End

query "query name" (Object obj,...)

#conditions

End

#### 对象定义

在 Drools 当中，可以定义两种类型的对象：一种是普通的类型Java Fact 的对象；另一种是用来描述Fact 对象或其属性的元数据对象。

http://wenku.baidu.com/link?url=kx2\_RKAzzVgOzMC3S96HUiL849\_UTs\_DhPa9qfxhKnbOy1hLLb8c6xeW844jENB3rICd6v3VewZCYfS8p-qQcW6n-rzpek2zruBqKTcV3ry