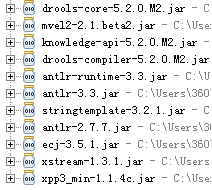
# drools 学习笔记

2017年09月26日 10:28:04 [sanshou](https://me.csdn.net/sanshou" \t "https://blog.csdn.net/sanshou/article/details/_blank) 阅读数 362

****1.Drools简单例子****

首先是搭建一个可供进行Drools开发的框架。Jboss官方推荐使用Drools Eclipse IDE进行开发，但是我发现其插件的安装相当繁琐，对其他的组件依赖严重，而且里面新手能用到的东东不多，下面就偷懒来搭建一个demo。

此demo基于Maven3进行开发，没有用过Maven的同学还是希望先看看Maven的东西。当然，不用maven也是没有问题的，你需要用到的包如下图：



当然最好还是用maven。首先使用ecplise新建一个maven的工程：TestDrools，在其中Pom.xml中添加如下依赖：

<dependencies>

        <dependency>

            <groupId>org.drools</groupId>

            <artifactId>drools-core</artifactId>

            <version>5.2.0.M2</version>

        </dependency>

        <dependency>

            <groupId>org.drools</groupId>

            <artifactId>drools-compiler</artifactId>

            <version>5.2.0.M2</version>

        </dependency>

        <dependency>

            <groupId>com.thoughtworks.xstream</groupId>

            <artifactId>xstream</artifactId>

            <version>1.3.1</version>

        </dependency>

</dependencies>

我们假定如下情景：网站伴随业务产生而进行的积分发放操作。比如支付宝信用卡还款奖励积分等。

我们定义一下发放规则：

积分的发放参考因素有：交易笔数、交易金额数目、信用卡还款次数、生日特别优惠等。

定义规则：

// 过生日，则加10分，并且将当月交易比数翻倍后再计算积分

// 2011-01-08 - 2011-08-08每月信用卡还款3次以上，每满3笔赠送30分

// 当月购物总金额100以上，每100元赠送10分

// 当月购物次数5次以上，每五次赠送50分

// 特别的，如果全部满足了要求，则额外奖励100分

// 发生退货，扣减10分

// 退货金额大于100，扣减100分

首先我们进入的Drools规则的编制阶段。这里采用drl文件定义规则，我们分别建立两个drl文件。

addpoint.drl：

package com.drools.demo.point

import com.jd.drools.test.PointDomain;

rule birthdayPoint

// 过生日，则加10分，并且将当月交易比数翻倍后再计算积分

salience 100

lock-on-active true

when

$pointDomain : PointDomain(birthDay == true)

then

$pointDomain.setPoint($pointDomain.getPoint()+10);

$pointDomain.setBuyNums($pointDomain.getBuyNums()\*2);

$pointDomain.setBuyMoney($pointDomain.getBuyMoney()\*2);

$pointDomain.setBillThisMonth($pointDomain.getBillThisMonth()\*2);

$pointDomain.recordPointLog($pointDomain.getUserName(),"birthdayPoint");

end

rule billThisMonthPoint

// 2011-01-08 - 2011-08-08每月信用卡还款3次以上，每满3笔赠送30分

salience 99

lock-on-active true

date-effective "2011-01-08 23:59:59"

date-expires "2011-08-08 23:59:59"

when

$pointDomain : PointDomain(billThisMonth >= 3)

then

$pointDomain.setPoint($pointDomain.getPoint()+$pointDomain.getBillThisMonth()/3\*30);

$pointDomain.recordPointLog($pointDomain.getUserName(),"billThisMonthPoint");

end

rule buyMoneyPoint

// 当月购物总金额100以上，每100元赠送10分

salience 98

lock-on-active true

when

$pointDomain : PointDomain(buyMoney >= 100)

then

$pointDomain.setPoint($pointDomain.getPoint()+ (int)$pointDomain.getBuyMoney()/100 \* 10);

$pointDomain.recordPointLog($pointDomain.getUserName(),"buyMoneyPoint");

end

rule buyNumsPoint

// 当月购物次数5次以上，每五次赠送50分

salience 97

lock-on-active true

when

$pointDomain : PointDomain(buyNums >= 5)

then

$pointDomain.setPoint($pointDomain.getPoint()+$pointDomain.getBuyNums()/5 \* 50);

$pointDomain.recordPointLog($pointDomain.getUserName(),"buyNumsPoint");

end

rule allFitPoint

// 特别的，如果全部满足了要求，则额外奖励100分

salience 96

lock-on-active true

when

$pointDomain:PointDomain(buyNums >= 5 && billThisMonth >= 3 && buyMoney >= 100)

then

$pointDomain.setPoint($pointDomain.getPoint()+ 100);

$pointDomain.recordPointLog($pointDomain.getUserName(),"allFitPoint");

End

subpoint.drl:

package com.drools.demo.point

import com.jd.drools.test.PointDomain;

rule subBackNumsPoint

// 发生退货，扣减10分

salience 10

lock-on-active true

when

$pointDomain : PointDomain(backNums >= 1)

then

$pointDomain.setPoint($pointDomain.getPoint()-10);

$pointDomain.recordPointLog($pointDomain.getUserName(),"subBackNumsPoint");

end

rule subBackMondyPoint

// 退货金额大于100，扣减100分

salience 9

lock-on-active true

when

$pointDomain : PointDomain(backMondy >= 100)

then

$pointDomain.setPoint($pointDomain.getPoint()-10);

$pointDomain.recordPointLog($pointDomain.getUserName(),"subBackMondyPoint");

End

这样我们就把开头所述的规则浓缩到这两个文件当中，Drools中可以使用PackageBuilder类来编译这两个文件。（具体用法在下面有体现）

接下来进入Drools的运行阶段。首先需要说明Drools中一个比较重要的概念：fact对象。

在Drools 当中是通过向WorkingMemory中插入Fact对象的方式来实现规则引擎与业务数据的交互，对于Fact对象就是普通的具有若干个属性及其对应的getter与setter方法的JavaBean对象。Drools除了可以接受用户在外部向WorkingMemory当中插入现成的Fact对象，还允许用户在规则文件当中定义一个新的Fact 对象, 在规则文件当中定义Fact 对象要以declare 关键字开头，以end 关键字结尾，中间部分就是该Fact 对象的属性名及其类型等信息的声明。

我们定义此例中的fact对象：PointDomain.java

/\*\*

 \* 积分计算对象

 \* ****@author**** quzishen

 \*/

****public**** ****class**** PointDomain {

// 用户名

****private**** String userName;

****public**** String getUserName() {

****return**** userName;

}

****public**** ****void**** setUserName(String userName) {

****this****.userName = userName;

}

****public**** ****boolean**** isBirthDay() {

****return**** birthDay;

}

****public**** ****void**** setBirthDay(****boolean**** birthDay) {

****this****.birthDay = birthDay;

}

****public**** ****long**** getPoint() {

****return**** point;

}

****public**** ****void**** setPoint(****long**** point) {

****this****.point = point;

}

****public**** ****int**** getBuyNums() {

****return**** buyNums;

}

****public**** ****void**** setBuyNums(****int**** buyNums) {

****this****.buyNums = buyNums;

}

****public**** ****int**** getBackNums() {

****return**** backNums;

}

****public**** ****void**** setBackNums(****int**** backNums) {

****this****.backNums = backNums;

}

****public**** ****double**** getBuyMoney() {

****return**** buyMoney;

}

****public**** ****void**** setBuyMoney(****double**** buyMoney) {

****this****.buyMoney = buyMoney;

}

****public**** ****double**** getBackMondy() {

****return**** backMondy;

}

****public**** ****void**** setBackMondy(****double**** backMondy) {

****this****.backMondy = backMondy;

}

****public**** ****int**** getBillThisMonth() {

****return**** billThisMonth;

}

****public**** ****void**** setBillThisMonth(****int**** billThisMonth) {

****this****.billThisMonth = billThisMonth;

}

// 是否当日生日

****private**** ****boolean**** birthDay;

// 增加积分数目

****private**** ****long**** point;

// 当月购物次数

****private**** ****int**** buyNums;

// 当月退货次数

****private**** ****int**** backNums;

// 当月购物总金额

****private**** ****double**** buyMoney;

// 当月退货总金额

****private**** ****double**** backMondy;

// 当月信用卡还款次数

****private**** ****int**** billThisMonth;

/\*\*

 \* 记录积分发送流水，防止重复发放

 \* ****@param**** userName 用户名

 \* ****@param**** type 积分发放类型

 \*/

****public**** ****void**** recordPointLog(String userName, String type){

System.*out*.println("增加对"+userName+"的类型为"+type+"的积分操作记录.");

}

}

规则有了，交互的对象也有了，我们需要实现一个workingMemory来装载这些对象进行运算。在Drools5 当中提供了两个对象与规则引擎进行交互：StatefulKnowledgeSession

和StatelessKnowledgeSession。本例中使用了StatefulKnowledgeSession进行交互。

前面说过一个RuleBase可以同时初始化多个Working Memory，而RuleBase是通过Drools中的

RuleBaseFactory产生的。我们先定义一个工厂类用于获取单例的RuleBase。

RuleBaseFacatory.java

/\*\*

 \* RuleBaseFacatory 单实例RuleBase生成工具

 \* ****@author**** quzishen

 \*/

****public**** ****class**** RuleBaseFacatory {

****private**** ****static**** RuleBase *ruleBase*;

****public**** ****static**** RuleBase getRuleBase(){

****return**** ****null**** != *ruleBase* ? *ruleBase* : RuleBaseFactory.*newRuleBase*();

}

}

接下来定义一个定义积分规则接口，里面包含了初始化RuleBase、workingMemory以及执行规则的方法。

PointRuleEngine.java

/\*\*

 \* 规则接口

 \* ****@author**** quzishen

 \*/

****public**** ****interface**** PointRuleEngine {

/\*\*

 \* 初始化规则引擎

 \*/

****public**** ****void**** initEngine();

/\*\*

 \* 刷新规则引擎中的规则

 \*/

****public**** ****void**** refreshEnginRule();

/\*\*

 \* 执行规则引擎

 \* ****@param**** pointDomain 积分Fact

 \*/

****public**** ****void**** executeRuleEngine(****final**** PointDomain pointDomain);

}

定义它的实现类，并封装main方法用于测试规则是否有效。

PointRuleEngineImpl.java

****import**** java.io.BufferedReader;

****import**** java.io.File;

****import**** java.io.FileNotFoundException;

****import**** java.io.FileReader;

****import**** java.io.IOException;

****import**** java.io.InputStream;

****import**** java.io.InputStreamReader;

****import**** java.io.Reader;

****import**** java.util.ArrayList;

****import**** java.util.List;

****import**** org.drools.RuleBase;

****import**** org.drools.StatefulSession;

****import**** org.drools.compiler.DroolsParserException;

****import**** org.drools.compiler.PackageBuilder;

****import**** org.drools.spi.Activation;

****import**** com.jd.drools.test.PointDomain;

/\*\*

 \* 规则接口实现类

 \* ****@author**** quzishen

 \*/

****public**** ****class**** PointRuleEngineImpl ****implements**** PointRuleEngine {

****private**** RuleBase ruleBase;

/\* (non-Javadoc)

 \* @see com.drools.demo.point.PointRuleEngine#initEngine()

 \*/

****public**** ****void**** initEngine() {

// 设置时间格式

System.*setProperty*("drools.dateformat", "yyyy-MM-dd HH:mm:ss");

ruleBase = RuleBaseFacatory.*getRuleBase*();

****try**** {

PackageBuilder backageBuilder = getPackageBuilderFromDrlFile();

ruleBase.addPackages(backageBuilder.getPackages());

} ****catch**** (DroolsParserException e) {

e.printStackTrace();

} ****catch**** (IOException e) {

e.printStackTrace();

} ****catch**** (Exception e) {

e.printStackTrace();

}

}

/\* (non-Javadoc)

 \* @see com.drools.demo.point.PointRuleEngine#refreshEnginRule()

 \*/

****public**** ****void**** refreshEnginRule() {

ruleBase = RuleBaseFacatory.*getRuleBase*();

org.drools.rule.Package[] packages = ruleBase.getPackages();

****for****(org.drools.rule.Package pg : packages) {

ruleBase.removePackage(pg.getName());

}

initEngine();

}

/\* (non-Javadoc)

 \* @see com.drools.demo.point.PointRuleEngine#executeRuleEngine(com.drools.demo.point.PointDomain)

 \*/

****public**** ****void**** executeRuleEngine(****final**** PointDomain pointDomain) {

****if****(****null**** == ruleBase.getPackages() || 0 == ruleBase.getPackages().length) {

****return****;

}

StatefulSession statefulSession = ruleBase.newStatefulSession();

statefulSession.insert(pointDomain);

// fire

statefulSession.fireAllRules(****new**** org.drools.spi.AgendaFilter() {

****public**** ****boolean**** accept(Activation activation) {

****return**** !activation.getRule().getName().contains("\_test");

}

});

statefulSession.dispose();

}

/\*\*

 \* 从Drl规则文件中读取规则

 \* ****@return****

 \* ****@throws**** Exception

 \*/

****private**** PackageBuilder getPackageBuilderFromDrlFile() ****throws**** Exception {

// 获取测试脚本文件

List<String> drlFilePath = getTestDrlFile();

// 装载测试脚本文件

List<Reader> readers = readRuleFromDrlFile(drlFilePath);

PackageBuilder backageBuilder = ****new**** PackageBuilder();

****for**** (Reader r : readers) {

backageBuilder.addPackageFromDrl(r);

}

// 检查脚本是否有问题

****if****(backageBuilder.hasErrors()) {

****throw**** ****new**** Exception(backageBuilder.getErrors().toString());

}

****return**** backageBuilder;

}

/\*\*

 \* ****@param**** drlFilePath 脚本文件路径

 \* ****@return****

 \* ****@throws**** FileNotFoundException

 \*/

****private**** List<Reader> readRuleFromDrlFile(List<String> drlFilePath) ****throws**** FileNotFoundException {

****if**** (****null**** == drlFilePath || 0 == drlFilePath.size()) {

****return**** ****null****;

}

List<Reader> readers = ****new**** ArrayList<Reader>();

****for**** (String ruleFilePath : drlFilePath) {

readers.add(****new**** FileReader(****new**** File(ruleFilePath)));

}

****return**** readers;

}

/\*\*

 \* 获取测试规则文件

 \*

 \* ****@return****

 \*/

****private**** List<String> getTestDrlFile() {

List<String> drlFilePath = ****new**** ArrayList<String>();

drlFilePath

.add("D:\\myworkspace\\TestDrools\\target\\classes\\addpoint.drl");

drlFilePath

.add("D:\\myworkspace\\TestDrools\\target\\classes\\subpoint.drl");

****return**** drlFilePath;

}

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

PointRuleEngine pointRuleEngine = ****new**** PointRuleEngineImpl();

****while****(****true****){

InputStream is = System.*in*;

BufferedReader br = ****new**** BufferedReader(****new**** InputStreamReader(is));

String input = br.readLine();

System.*out*.println("请输入命令：");

****if****(****null**** != input && "s".equals(input)){

System.*out*.println("初始化规则引擎...");

pointRuleEngine.initEngine();

System.*out*.println("初始化规则引擎结束.");

}****else**** ****if****("e".equals(input)){

****final**** PointDomain pointDomain = ****new**** PointDomain();

System.*out*.println("初始化规则引擎...");

pointRuleEngine.initEngine();

System.*out*.println("初始化规则引擎结束.");

pointDomain.setUserName("hello kity");

pointDomain.setBackMondy(100d);

pointDomain.setBuyMoney(500d);

pointDomain.setBackNums(1);

pointDomain.setBuyNums(5);

pointDomain.setBillThisMonth(5);

pointDomain.setBirthDay(****true****);

pointDomain.setPoint(0l);

pointRuleEngine.executeRuleEngine(pointDomain);

System.*out*.println("执行完毕BillThisMonth："+pointDomain.getBillThisMonth());

System.*out*.println("执行完毕BuyMoney："+pointDomain.getBuyMoney());

System.*out*.println("执行完毕BuyNums："+pointDomain.getBuyNums());

System.*out*.println("执行完毕规则引擎决定发送积分："+pointDomain.getPoint());

} ****else**** ****if****("r".equals(input)){

System.*out*.println("刷新规则文件...");

pointRuleEngine.refreshEnginRule();

System.*out*.println("刷新规则文件结束.");

}

}

}

}

执行main方法，输入'e'，得到：

初始化规则引擎...

初始化规则引擎结束.

增加对hello kity的类型为birthdayPoint的积分操作记录.

增加对hello kity的类型为buyMoneyPoint的积分操作记录.

增加对hello kity的类型为buyNumsPoint的积分操作记录.

增加对hello kity的类型为allFitPoint的积分操作记录.

增加对hello kity的类型为subBackNumsPoint的积分操作记录.

增加对hello kity的类型为subBackMondyPoint的积分操作记录.

执行完毕BillThisMonth：10

执行完毕BuyMoney：1000.0

执行完毕BuyNums：10

执行完毕规则引擎决定发送积分：290

****2.Droolsv API解释****

Drools API可以分为三类：规则编译、规则收集和规则的执行

API:

1. KnowledgeBuilder规则编译：规则文件进行编译， 最终产生一批编译好的规则包(KnowledgePackage)供其它的应用程序使用

2. KnowledgeBase：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

3. StatefulKnowledgeSession：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。在用户的代码当中，最后使用完StatefulKnowledgeSession 对象之后，一定要调用其dispose()方法以释放相关内存资源。有状态的

4. StatelessKnowledgeSession：使用StatelessKnowledgeSession 对象时不需要再调用dispose()方法释放内存资源不能进行重复插入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

5. 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Clone，而是原来JavaBean 对象的引用

6.

****7.Drools规则****

7.1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，标准的规则文件格式：

package package-name //包名是必须的，并放在第一行，包名对于规则文件中规则的管理只限于逻辑上的

imports

globals

functions

queries

rules

7.2规则语言

一个标准规则的结构

rule "name" //规则名称

attributes //属性部分

when

LHS //left hand sid条件部分

then

RHS //right hand sid结果部分

End

7.2.1条件部分

条件部分又被称之为Left Hand Side，简称为LHS，条件又称之为pattern（匹配模式）: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，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，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。

1. 比较操作符：共计12种：

>、>=、<、<=、= =、!=、

contains、not contains、memberof、not memberof、matches、not matches

1) 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

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)

示例：

****package****test

****rule****"rule1"

****when****

$order:Order();

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

****then****

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

****end****

2) Not Contains：与contains作用相反

3) Member Of ：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，member of 前边是某个数据对象且一定要是一个变量(绑定变量或者是一个global 对象)，后边是数据对象集合：

示例：

****package****test

****global****String[] orderNames;

****rule****"rule1"

****when****

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

****then****

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

****End****

4) Not member of：与member of作用相反

5) 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

示例：

****package****test

****import****java.util.List;

****rule****"rule1"

****when****

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

****then****

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

****end****

6) not matches:与matches相反

结果部分：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, salience该属性的作用是通过一个数字来确认规则执行的优先级，数字越大，执行越靠前。

函数介绍：

ü Insert：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。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

ü 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WorkingMemroy 当中

ü update:用来实现对当前Working Memory 当中的Fact 进行更新。如果希望规则只执行一次，那么可以通过设置规则的no-loop属性为true 来实现

示例：

****package****test

****import****java.util.List;

****query****"query fact count"

Customer();

****end****

****rule****"rule1"

****salience****2

****when****

****eval****(****true****);

****then****

Customer cus=****new****Customer();

cus.setName("张三");

cus.setAge(1);

****insert****(cus);

****end****

****rule****"rule2"

****salience****1

****when****

$customer:Customer(name=="张三",age<10);

****then****

$customer.setAge($customer.getAge()+1);

****update****($customer);

System.out.println("----------"+$customer.getName());

****End****

****示例说明：****

调用update 宏函数更新Customer 对象后Working Memory 当中还只存在一个Customer 对象

ü retract：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

ü drools:宏对象可以实现在规则文件里直接访问Working Memory

常用方法说明：

|  |  |
| --- | --- |
| 方法名称 | 含义说明 |
| getWorkingMemory() | 获取当前的WorkingMemory 对象 |
| halt() | 在当前规则执行完成后，不再执行  其它未执行的规则。 |
| getRule() | 得到当前的规则对象 |
| 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) | 更新当前的WorkingMemory 中指定  的对象，功能与宏函数update 相同 |
| 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) | 从当前的WorkingMemory 中删除指  定的对象，功能与宏函数retract 相  同。 |
| kcontext | 作用主要是用来得到当前的  KnowledgeRuntime 对象，KnowledgeRuntime 对象可以实现与引擎的各种交互 |

ü 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

7.2.2属性部分

规则的属性共有13 个分别是：activation-group、agenda-group、auto-focus、date-effective、date-expires、dialect、duration、enabled、lock-on-active、no-loop、ruleflow-group、salience、when

1. 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，那么它的执行顺序是随机的。

2. 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

3. date-effective：在规则运行时，引擎会自动拿当前操作系统的时间与date-effective设置的时间值进行比对，只有当系统时间>=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。日期格式：dd-MM-yyyy

4. date-expires该属性的作用与date-effective属性恰恰相反，如果大于系统时间，那么规则就执行，否则就不执行。日期格式：dd-MM-yyyy

5. enabled: true执行该规则，false不执行该规则

6. dialect：该属性用来定义规则当中要使用的语言类型：mvel 和java，如果没有手工设置规则的dialect，默认使用的java 语言

7. duration: 该属性对应的值为一个长整型，单位是毫秒。如果设置了该属性，那么规则将在该属性值之后时间，在另外一个线程里触发

8. lock-on-active：该属性为boolean，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

9. activation-group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。

10. agenda-group: agenda-group规则的调用与执行是通过StatelessSession 或StatefulSession 来实现的，一般的顺序是创建一个StatelessSession 或StatefulSession，将各种经过编译的规则的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

11. auto-focus: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独取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。

12. ruleflow-group: 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

7.2.3注释

1. 单行注释：采用“#”或者“//”来进行标记

2. 多行注释：以“/\*”开始，以“\*/”结束

7.3函数

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

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

/\*函数体的业务代码\*/

}

函数以function标记开头，可以有或无返回类型，然后定义方法名和参数，语法基本同java一致，不同规则文件的函数相互之间是不可见的。

示例：

****package****test

****import****java.util.List;

****import****java.util.ArrayList;

/\*

一个测试函数

用来向Customer对象当中添加指定数量的Order对象的函数

\*/

****function void****setOrder(Customer customer,****int****orderSize) {

List ls=****new****ArrayList();

****for****(****int****i=0;i<orderSize;i++){

Order order=****new****Order();

ls.add(order);

}

customer.setOrders(ls);

}

/\*

测试规则

\*/

****rule****"rule1"

****when****

$customer :Customer();

****then****

setOrder($customer,5);

System.out.println("rule 1 customer has order

size:"+$customer.getOrders().size());

****end****

/\*

测试规则

\*/

****rule****"rule2"

****when****

$customer :Customer();

****then****

setOrder($customer,10);

System.out.println("rule 2 customer has order

size:"+$customer.getOrders().size());

****end****

7.4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，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

7.4.1无参数查询

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

示例：

****query****"testQuery"

customer:Customer(age>30,orders.size >10)

****end****

查询的调用是由StatefulSession完成的，通过调用StatefulSession对象的getQueryResults(String queryName)方法实现对查询的调用，该方法的调用会返回一个QueryResults对象，QueryResults是一个类似于Collection接口的集合对象，在它当中存放在若干个QueryResultsRow对象，通过QueryResultsRow可以得到对应的Fact对象，从而实现根据条件对当前WorkingMemory当中Fact 对象的查询

7.4.2参数查询

和函数一样，查询也可以接收外部传入参数

代码示例：

****query****"testQuery"(****int****$age,String $gender)

customer:Customer(age>$age,gender==$gender)

****end****

7.5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。

7.5.1 java Fact 对象

在Drools 当中是通过向WorkingMemory中插入Fact对象的方式来实现规则引擎与业务数据的交互，对于Fact对象就是普通的具有若干个属性及其对应的getter与setter方法的JavaBean对象。Drools除了可以接受用户在外部向WorkingMemory当中插入现成的Fact对象，还允许用户在规则文件当中定义一个新的Fact 对象, 在规则文件当中定义Fact 对象要以declare 关键字开头，以end 关键字结尾，中间部分就是该Fact 对象的属性名及其类型等信息的声明。

示例：

****declare****Address

city : String

addressName : String

****end****

7.5.2元数据定义

为Fact对象的属性或者是规则来定义元数据，元数据定义采用的是“@”符号开头，后面是元数据的属性名（属性名可以是任意的），然后是括号，括号当中是该元数据属性对应的具体值

示例：

@author(jacob)