Learn Drools: Part II(Cross Product)

In the previous article, we got a basic idea what is Drools and How it works. In this article, we pay attention to the details how Rule works with facts.

When we write Rules Often a question Pops up our mind is

**How can I invoke one rule from another Rule like we call a method?**

The answer is no we can’t call a Rule from another Rule , The reason Drools match the Rules with incoming data/facts and if data satisfies Rule condition it stores the data in an **Agenda, I**t immediately not fire the **then** part of a Rule as, It may be possible Same Data or facts can be matched by different rules so It stores matching facts in a Agenda and after firing all rules then it takes the action on Agenda i.e fires the then parts on facts.

So, as the rules apply on data/facts it is not possible to call a rule from another as we can’t pass any arguments/facts beforehand as facts are coming at runtime.

So, rather than confused the Rules with functional/oops programming consider it with **RDBMS.**

**Where we can compare Drools Rules and Facts with RDBMS component**

|  |  |
| --- | --- |
| **Drools Component** | **RDBMS Component** |
| **Data/Facts/Domain Object data structure** | **Table** |
| **Facts/Domain Objects** | **Rows** |
| **Fact /Domain Object attributes** | **Column** |
| **Domain Objects HAS-A relationship** | **Joining/Cartesian Product** |
| **Drools operators** | **Filter Criterias** |
| **Drools When** | **When in Sql** |
| **Drools Then** | **Trigger** |
| **Drools Rule** | **Sql Query** |

Try to understand the above relationship with a simple example, Suppose we have Two Domain Objects say Department and Employee, and Each Employee is associated with a Department.

Step 1: Creating Data structure for Department and Employee, Nothing but two Simple POJOs which same as Two tables Department and Employee.

Department.java

package com.example.droolsExample.pojo;

public class Department {

String name;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

Employee.java

package com.example.droolsExample.pojo;

public class Employee {

String name;

boolean manager;

String message;

Department dept;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public boolean isManager() {

return manager;

}

public void setManager(boolean manager) {

this.manager = manager;

}

public String getMessage() {

return message;

}

public void setMessage(String message) {

this.message = message;

}

public Department getDept() {

return dept;

}

public void setDept(Department dept) {

this.dept = dept;

}

}

Step 2: Create a simple rule

employee.drl

package com.rules

import com.example.droolsExample.pojo.Employee

import com.example.droolsExample.pojo.Department

rule "print cross product"

when

emp: Employee();

dept: Department();

then

System.out.println("Fire print cross product Rule");

System.out.println(dept.getName() + "::" + emp.getName()+ ": "+emp.getDept().getName());

end

We create a simple rule , it tells that for every instance of Employee and Department in working memory print the department name and employee name and Employee's department name.

It is same as the following sql

***Select \* from Employee emp Department dept where emp.deptId=dept.deptId***

So it will create all cartesian product in Working memory

So If we have two departments and two employees in working memory it creates four combinations and then fires the rules on these combinations.

Step 3: Create Working memory and insert Objects to show the result.

package com.example.droolsExample;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.Reader;

import org.drools.compiler.compiler.DroolsParserException;

import org.drools.compiler.compiler.PackageBuilder;

import org.drools.core.RuleBase;

import org.drools.core.RuleBaseFactory;

import org.drools.core.WorkingMemory;

import com.example.droolsExample.pojo.Department;

import com.example.droolsExample.pojo.Employee;

public class DroolsTest {

public static void main(String[] args) throws DroolsParserException,

IOException {

DroolsTest droolsTest = new DroolsTest();

droolsTest.executeDroolsEmployee();

}

public void executeDroolsEmployee() throws DroolsParserException, IOException {

PackageBuilder packageBuilder = new PackageBuilder();

String ruleFile = "/com/rules/employee.drl";

InputStream resourceAsStream = getClass().getResourceAsStream(ruleFile);

Reader reader = new InputStreamReader(resourceAsStream);

packageBuilder.addPackageFromDrl(reader);

org.drools.core.rule.Package rulesPackage = packageBuilder.getPackage();

RuleBase ruleBase = RuleBaseFactory.newRuleBase();

ruleBase.addPackage(rulesPackage);

WorkingMemory workingMemory = ruleBase.newStatefulSession();

Department dep = new Department();

dep.setName("Civil");

Department dep1 = new Department();

dep1.setName("IT");

Employee emp = new Employee();

emp.setName("Shamik Mitra");

emp.setManager(true);

emp.setDept(dep1);

Employee emp1 = new Employee();

emp1.setName("Samir Mitra");

emp1.setManager(true);

emp1.setDept(dep);

workingMemory.insert(dep);

workingMemory.insert(dep1);

workingMemory.insert(emp);

workingMemory.insert(emp1);

workingMemory.fireAllRules();

}

}

Step 4: Checking the Output

Civil::Shamik Mitra: IT

IT::Shamik Mitra: IT

Civil::Samir Mitra: Civil

IT::Samir Mitra: Civil

Please note that as we have inserted two employees Objects and Two department Objects in working memory so it creates four different combinations.

In the next Article, we will discuss about How to filter this result set using Drools.