规则引擎调研报告

二〇一九年七月

编 写：王翰宇

目 录

[1 定义 1](#_Toc14508379)

[2 工作原理 1](#_Toc14508380)

[3 运行方式 1](#_Toc14508381)

[4 规则的形式 2](#_Toc14508382)

[4.1 DRL文件 2](#_Toc14508383)

[4.2 Decision tables 3](#_Toc14508384)

[4.3 CEP 4](#_Toc14508385)

[5 可解决的需求 5](#_Toc14508386)

[5.1 数据校验 5](#_Toc14508387)

[5.1.1 内容 5](#_Toc14508388)

[5.1.2 返回值 5](#_Toc14508389)

[5.1.3 实现方式 5](#_Toc14508390)

[5.2 数据增删改查 6](#_Toc14508391)

[5.2.1 内容 6](#_Toc14508392)

[5.2.2 实现方式 7](#_Toc14508393)

[5.3 规则管理 7](#_Toc14508394)

[5.3.1 内容 7](#_Toc14508395)

[5.3.2 实现方式 7](#_Toc14508396)

[6 常见应用场景 7](#_Toc14508397)

[7 产品方案 8](#_Toc14508398)

[7.1 Drools Business Central Workbench网页版平台（免费） 8](#_Toc14508399)

[7.2 Red Hat Process Automation Manager（Drools企业付费版） 9](#_Toc14508400)

[7.3 IBM Decision Composer（免费） 10](#_Toc14508401)

[7.4 IBM Operational Decision Manager（付费） 10](#_Toc14508402)

[7.5 旗正规则引擎 11](#_Toc14508403)

[7.6 其他 11](#_Toc14508404)

[8 API 11](#_Toc14508405)

[8.1 KIE API 11](#_Toc14508406)

[8.2 JSR 94 11](#_Toc14508407)

[9 建议 12](#_Toc14508408)

# 定义

规则引擎将业务规则与业务流程分离，可以在独立于代码的条件下定义、测试、运行、维护规则。

# 和工作原理

规则引擎的核心是推理引擎。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)。

图片包含 杯子

描述已自动生成

图1 规则引擎

规则被存储在Production Memory里，用来匹配的fact被存储在Working Memory里。系统可能会出现多个规则对应同一个事实的情况，Agenda会使用能恰当解决冲突的策略来安排这些规则执行的顺序。

# 运行方式

推理引擎存在两者推理方式：正向推理（forward chaining）和逆向推理（backward chaining），**常用的是正向推理，**逆向推理比较少见。

Forward chaining是数据驱动的推理算法；其根据假设，不断地寻找符合假设的事实。Forward chaining先测试规则的条件是否被满足，然后判断是否执行生产规则。简单的正向推理就是基于其他属性的值，来给指定属性赋值；更复杂的正向推理支持一阶的微积分，即可以做定量计算，由RETE算法执行。

当系统可能推测出多个目标结果/属性，而不能提前获知哪个目标一定是最终的结果时，应该使用forward chaining。想在推测过程中获取尽可能多的信息时，也应该使用forward chaining。

Backward chaining是一个执行规则的递归算法，是目标驱动型。Backward chaining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。在backward chaining的过程中，推理引擎会通过操作（结论）来测试规则。一旦规则的结论实现了以后，规则就不会被重新评估了。

# 规则的形式

## DRL文件

1. 介绍

DRL文件，是Drools中的规则文件，后缀名为.drl，是记录规则的**普遍形式**；其编写遵循Drools规则语法。DRL文件的整体结构如下：

package package-name

imports

globals

functions

queries

**rules**

除了package必须是DRL文件的第一个元素，其他元素可选，且顺序不固定。

一个规则在一个包中必须具有独一无二的名字。如果在DRL文件中重复定义两个相同名字的规则，加载时就会报错。如果向包中添加一个名字已经存在的规则，该规则会覆盖之前的同名规则。规则不允许嵌套。

一个简单的规则结构如下所示：

rule "<name>"

<attributes>

when

<condition>

then

<action>

end

attributes展示规则的执行方式，是可选的；

condition是规则的条件部分，遵循一个固定的语法规则；

action通常是一个可以本地执行的代码块。

2. 应用场景

由于DRL文件不允许规则嵌套，所以适合存放逻辑结构不是特别复杂的规则。

## Decision tables

1. 介绍

Drools支持以数据表的形式来管理规则，该形式称为Decision tables。Drools支持引入的文件包括.xls和.csv文件。

决策表代表了一组样板相同，而只是属性的值不同的规则。它的每一行就是一条规则，该行的每一列是条件或者操作。

图片包含 屏幕截图

描述已自动生成

图2 决策表示例

根据表格可支持数据的属性，各产品供应商提供的技术不同。大多数决策表支持系统只能测试决策表里的数值和字符，不过有的厂商也支持测试属性/变量。操作列可以多于1个，也可以在操作列里运行一个函数。

2. 应用场景

适用于：

使用了表格来管理数据和计算的业务；

规则的条件和结论使用了很多相同的属性；

业务人员倾向于在.xls和.csv文件里管理规则；

业务人员管理规则时有修改参数的需求。

不适用于：

只有少量规则的业务；

规则不遵循统一模板的业务。

## CEP

1. 介绍

CEP, Complex Event Processing，意即能处理多个事件，从一系列事件中发现重要事件，发现事件之间的联系和从事件中推断出数据之间的关系。Drools Fusion模块提供了CEP的性能。

规则示例：“当顾客在过去30秒内访问了5个包含图片的网页时”

图片包含 屏幕截图

描述已自动生成

图3 CEP规则示例

2. 应用场景

需要处理大量事件，但只有其中的一小部分事件需要关注；

事件的顺序不可变，事件记录了业务状态的变化；

事件之间存在明显的关系；

业务系统需要处理事件的集合，更关注事件的特征和相互关系。

# 可解决的需求

## 数据校验

### 内容

包括数据的合规性、用户校验、权限管理

例如：

1. 在元数据管理中，可实现检查水泵水轮机转速参数的取值范围。根据取 值范围约束，判断模型是否合规；

2. 在项目数据管理中，可实现对上传字段进行校验，校验通过后方可进 入平台。

### 返回值

可能产生的校验结果：

1. 直接结果，即返回校验正确与否；

2. 数据列表，即返回错误的数据列表；

3. 无返回值，并打印校验通过与否的信息。

### 实现方式

1. 数据校验

1. 示例：

public class Applicant {

private String name;

private int age;

private boolean valid;

// getter and setter methods here

}

根据上面的数据模型，我们设计一个规则：18岁以下的申请人会被拒绝

package com.company.license

rule "Is of valid age"

when

$a : Applicant( age < 18 )

then

$a.setValid( false );

System.out.println("18 and under are rejected.");

end

$a是一个绑定变量，它允许我们引用匹配的对象。

数据规则可以附加在用户身上，描述用户可以看到哪些资源，比如创建变量isAdmin = true.

2. 账户校验

1. 示例：

rule "increase balance for credits"

agenda-group "calculation"

when

ap : AccountPeriod()

acc : Account( $accountNo : accountNo )

CashFlow( type == CREDIT,

accountNo == $accountNo,

date >= ap.start && <= ap.end,

$amount : amount )

then

acc.balance += $amount;

end

## 数据增删改查

### 内容

1. 用户账号、角色、业务参数、工程特征标准模型、业务实体标准模型、 编码字段、标准流程等的新增、查询、修改和删除。

2. 在配额管理中，业务用户可查看申请服务的配额上限和配额已用情况、 并发峰值记录和付费情况记录。

### 实现方式

在项目代码里搜索数据库，将存储数据库内容的对象插入Drools，读取数据。再调用.drl规则文件，利用规则对数据进行过滤分析，然后对过滤后的数据进行操作，根据需要决定是否更新数据库。

## 规则管理

### 内容

1. 开发人员可通过代码对规则进行管理。

2. 业务人员可对规则进行可视化管理。

### 实现方式

1. 把规则存储在数据库里，通过操作数据库来管理规则；

1. 把.drl文件放在数据库中维护，规则变更后，直接修改数据库，然后动态加载 规则到Drools的工作内存，系统无需重启，规则即时生效。

2. 通过KIE Workbench进行可视化管理。

# 常见应用场景

1. 金融类 - 保险（寿险、车险、理赔），风控等

2. 通信运营商 - 定制资费套餐

3. 超市和商城 - 积分计算规则

4. 航空公司 - 不同销售渠道的定价规则

5. 大型公司 - 工资核算

6. 物联网

# 产品方案

## Drools Business Central Workbench网页版平台（免费）

多种搭建方式，还需进一步配置

图片包含 屏幕截图

描述已自动生成

图4 KIE Server搭建场景

图片包含 屏幕截图

描述已自动生成

图5 Drools Business Central Workbench数据对象页面

图片包含 屏幕截图

描述已自动生成

图6 规则的Editor页面

图片包含 屏幕截图

描述已自动生成

图7 规则的源代码页面

## Red Hat Process Automation Manager（Drools企业付费版）

价格有待咨询。

## IBM Decision Composer（免费）

IBM Decision Composer是一个直观的、基于云的决策自动化工具，帮助发现和了解业务规则和Operational Decision Manager的基本内容。Decision Composer是一个无代码环境，业务用户可以在其中进行建模、编写、验证和共享业务规则。然后，开发人员可以在任何企业、云或移动应用程序中立即调用编写的决策。

IBM Decision Composer是免费的，每月能运行1000个决策。

在图形页面中，直角蓝色方框表示规则，圆角绿色方框表示规则相关的变量。

图片包含 屏幕截图

描述已自动生成

图8 IBM Decision Composer规则可视化页面

## IBM Operational Decision Manager（付费）

IBM中国客服提供了一个可参考的价格范围，一个小型保险公司配备的一套规则引擎价格在60~70万元之间，具体价格根据服务器数量而定，第二年的维护费用为该区间数目的20%左右。IBM技术支持会提供对公司IT和非IT人员的培训，之后每年的维护费用会逐渐减少。

IBM客服的建议是公司需要明确对规则引擎的具体需求，并衡量使用规则引擎产品的必要性和性价比。如果规则不复杂，推荐自行编码解决。如需要推进，可联系IBM技术人员就如何在业务中应用规则引擎进行咨询。

## 旗正规则引擎

客服支持不足

## 其他

其他基于Java的开源规则引擎有OpenRules, business-rules, Durable Rules等。

# API

## KIE API

KIE Group是四个项目的集合，其中为人熟悉的是jBPM和Drools。这些项目都有一定的关联关系，并且存在通用的API，涉及的功能包括构建（building）、部署（deploying）和加载（loading）等，这些API以 “KIE”作为前缀来表示他们是通用的API。核心组件包括KieServices, KieContainer, KieSession等。Drools中主要应用的是KIE API。

示例：在规则引擎中创建会话



图9 创建规则会话示例

## JSR 94

JSR 94是由JCP(Java Community Process)组织所制定的java规则引擎API的J8ava请求规范，主要定义了规则引擎在Java运行时的一些API。它的目的是为了促进规则引擎技术在java程序中的发展，增加Java规则引擎厂商之间的交流以及标准化工作，让使用规则引擎的第三方应用操作起来更加方便与规范，也为了简化规则引擎商对外提供的API。

Drools里有两部分用到了JSR 94。一是用于建立和注册RuleExecutionSet对象的API，二是这些RuleExecutionSet的运行。

应用示例：

// RuleServiceProviderImpl is registered to http://drools.org/

// via a static initialization block

Class.forName("org.drools.jsr94.rules.RuleServiceProviderImpl");

// Get the rule service provider from the provider manager.

RuleServiceProvider ruleServiceProvider =

RuleServiceProviderManager.getRuleServiceProvider("http://drools.org/");

# 建议

本报告调研了多种规则引擎产品，由于Drools是公认的、拥有众多用户的规则引擎，所以主要研究了Drools。不过随着调研深入，看到了更多的反馈，比如美团技术团队对于Drools的想法：

优势：

- 功能较为完善，具有如系统监控、操作平台等功能。

劣势：

- 学习曲线陡峭，其引入的DRL语言较复杂，独立的系统很难进行二次 - 开发；

- 以内存实现时间窗功能，无法支持较长跨度的时间窗；

- 无法有效支持定时触达（如用户在浏览发生后30分钟触达支付条件判 断）。

根据以上考量，美团最终选择了轻量的表达式引擎[Aviator](https://link.juejin.im/?target=https%3A%2F%2Fcode.google.com%2Farchive%2Fp%2Faviator%2F)。

同时，IBM客服人员也建议仔细评估使用规则引擎的必要性，先明确具体需求，再决定使用什么产品。

综上，推荐使用Drools，但仍需进一步讨论。