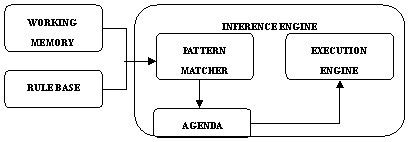
规则引擎

业务流程执行语言(BPEL)

中央规则信息库

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



使用规则引擎场景特点？

单条规则相对简单

规则数量相对庞大

规则之间会有冲突

一条规则本身会触发另一条规则

规则有可能会产生变动（很多时候可能唯一需要变动的就是规则）

工作流引擎现在演化为 业务过程管理，最基本的概念还是业务过程的流转，

业务规则侧重业务逻辑的控制，比如零售行业的价格规则，它会针对 客户 商品/品类 地理位置等设计一个非常复杂的价格管理体系；与业务规则非常紧密的是事件引擎，比如政府部门的应急处理系统

规则引擎使用了匹配规则的方式来进行，因此在应用这些规则引擎时。首先需要将我们具体应用中的业务逻辑做抽象，抽象成一条条规则之后，再打包成一个规则包。一个规则包相当于一个智能块。当数据传递给这个智能块后，系统会以匹配的方式应用满足条件的逻辑处理。

Drools

https://www.drools.org/#

应用程序中业务逻辑最常见的方法是编写 Java 代码来实现需求文档的规则和逻辑

规则引擎试图降低应用程序业务逻辑的开发和维护的困难。可以将**规则引擎看作实现复杂业务逻辑的框架**。大多数规则引擎允许您使用声明性编程来表达对于某些给定信息或知识有效的结果

Drools 是用 Java 语言编写的开放源码规则引擎, Drools允许使用声明方式表达业务逻辑。可以使用非 XML 的本地语言编写规则

何时使用规则引擎？

并非所有应用程序都应使用规则引擎。如果业务逻辑代码包括很多 if-else 语句，则应考虑使用一个规则引擎。**维护复杂的 Boolean 逻辑**可能是非常困难的任务，而规则引擎可以帮助您组织该逻辑。当您可以使用声明方法而非命令编程语言表达逻辑时，变化引入错误的可能性会大大降低。

对客户的了解也是该决策的一个因素。在开发周期期间甚至部署之后添加和更改业务逻辑需求的倾向。

Drools is a Business Rules Management System (BRMS) solution

Drools Workbench (web UI for authoring and management) + Drools Expert (business rules engine)

<http://blog.csdn.net/quzishen/article/details/6163012/>

接口：传参数，接口实现执行，获取结果

drools:传递数据，规则检查和执行，获取结果

在drools,传递数据称为fact对象（java bean),当一个java bean插入到workingMemory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workingMemory中插入删除新的fact对象。

规则文件: .drl

rule "name"

//属性部分：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。

no-loop true

//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。

when

$message:Message(status == 0)

//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。

then

System.out.println("fit");

$message.setStatus(1);

update($message);

end

属性：

no-loop : 定义当前的规则是否不允许多次循环执行，默认是false

lock-on-active true：通过这个标签，可以控制当前的规则只会被执行一次，因为一个规则的重复执行不一定是本身触发的，也可能是其他规则触发的，所以这个是no-loop的加强版

date-expires：设置规则的过期时间，默认的时间格式：“日-月-年”

date-effective：设置规则的生效时间

duration：规则定时，duration 3000 3秒后执行规则

salience：优先级，数值越大越先执行，这个可以控制规则的执行顺序。

条件：

当前规则只有在条件都匹配的时候才会执行

Drools提供了十二中类型比较操作符：

> >= < <= == != contains / not contains / memberOf / not memberOf /matches/ not matches

执行：

insert：往当前workingMemory中插入一个新的Fact对象，会触发规则的再次执行，除非使用no-loop限定；

update：更新

modify：修改，与update语法不同，结果都是更新操作

retract：删除

除了调用Drools提供的api和Fact对象的方法，也可以调用规则文件中定义的方法和类

function void console {

System.out.println();

StringUtils.getId();// 调用外部静态方法，StringUtils必须使用import导入，getId()必须是静态方法

}

declare Address

@author(quzishen) // 元数据，仅用于描述信息

@createTime(2011-1-24)

city : String @maxLengh(100)

postno : int

end

简单地说：

Drl中可以通过Import的方式引入Model类，也可以调用Java的各种函数，也可以自己定义Class and Function

Drl

package package-name

imports

globals

functions

queries

rules

Rules就是规则的部分，结构如下：

rule "name"

attributes

when

LHS(The Rule Language)

then

RHS(Java, Pthyon, Groovy)

End

每一条规则都有名称

每一条规则都有属性如lock-on-active等

When里写的是LHS( left hand side),等同于if里的条件判断，左手边是Drools自己的语法规则

Then里写的是RHS(right hand side),等同于if里的执行语句