快速入门规则引擎

规则引擎简介

背景

研究目的：网页活动项目中规则多样，规则使用的都是硬编码的方式，新增或修改规则比较麻烦，且逻辑不够清晰，扩展性不好。由此引入对规则的思考，查阅过业界的做法，了解到规则引擎这一利器，遂调研并学习记录一波。

概述

> 规则引擎起源于基于规则的专家系统，其提供一种可选的计算模型。与通常的命令式模型（由带有条件和循环的命令依次组成）不同，规则引擎基于生产规则系统。这是一组生产规则，每条规则都有一个条件（condition）和一个动作（action）可以将其看作是一组if-then语句。其通过输入一些基础事件，以推演或者归纳等方式，得到最终的执行结果。
>
> 比较学术的说法是：规则引擎由推理引擎发展而来，是一种嵌入在应用程序中的组件，实现了将业务决策从应用程序代码中分离出来，并使用预定义的语义模块编写业务决策。接受数据输入，解释业务规则，并根据业务规则做出业务决策。

组成：

一般分为三个部分：

* Fact： 用来传递数据，获取数据
* LHS(Left Hand Side)：可以理解为规则执行需要满足的条件。
* RHS(Right Hand Sike)：可以理解为规则执行后的返回对象。

优势

* 声明式编程
* 逻辑与数据分离
* 速度及可测量性：Rete算法、Leaps算法等对系统数据匹配对象非常有效率
* 易懂的规则：可以用接近自然语言的方式来编写规则，利于非技术人员编写规则
* 规则变更快，适应庞大的逻辑和复杂的流程分支，减少if-else

应用场景

* 日常活动配置，活动营销
* 风控模型配置

常见的规则引擎
---

* Drools
* Easy Rule
* RuleBook
* 表达式引擎：

+ MVEL
+ FEL
+ simpleEL
+ groovy
+ Aviator

Drools 介绍

> Drools 是一个基于Charles Forgy’s的RETE算法的`[注释1]`，易于访问企业策略、易于调整以及易于管理、基于java语言开发的开源业务规则引擎，符合业内标准，速度快、效率高，能够将Java代码直接嵌入到规则文件中。
>
>
> 官方地址：[www.drools.org/](http://cxyroad.com/”https://www.drools.org/”)

特点

1、简化系统架构，优化应用

2、提高系统的可维护性和维护成本

3、方便系统的整合

4、减少编写“硬代码”业务规则的成本和风险

5、社区活跃

5、学习成本高

基本概念

* Fact：对象之间及对象属性之间的关系
* rule：规则
* module：if语句的条件

规则文件示例

```
package act.lottery
import ink.wbc.drools.entity.User

rule ”lottery”
        no-loop true
        when
                $user:User(money > 100)
        then
                System.out.println(”满足抽奖条件”);
                $user.setAllowLotteryFlag(1);
end
```

* package：逻辑路径
* import：导入的java包
* rule：规则名称，唯一
* no-loop：是否多次循环执行，默认false，true则只执行一次
* when：条件语句
* then：条件成立，执行逻辑
* end：结束规则

使用示例

一、添加依赖

```
<!-- https://mvnrepository.com/artifact/org.drools/drools-compiler -->
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>7.10.0.Final</version>
</dependency>
```

二、创建kmodule.xml配置文件，指明规则文件的目录

```
注意：文件固定放在resource/META-INF下，名称固定为kmodule.xml
<?xml version="1.0" encoding="UTF-8" ?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
    <!--
        name:指定kbase的名称，可以任意，但是需要唯一
        packages:指定规则文件的目录，需要根据实际情况填写，否则无法加载到规则文件，名字任意
        default:指定当前kbase是否为默认
    -->
    <kbase name="droolsDemo" packages="rules" default="true">
        <!--
            ksession：会话对象
            name:指定ksession名称，可以任意，但是需要唯一
            default:指定当前session是否为默认
        -->
        <ksession name="ksession-rule" default="true"/>
```

```
    </kbase>
</kmodule>
```

三、创建实体文件

```
public class User {

    private Integer money;
    private Integer allowLotteryFlag;

    // ... 省略
}
```

四、创建drl文件，表示规则文件

```
package act.lottery
import ink.wbc.drools.entity.User

rule "lottery"
        no-loop true
        when
                $user:User(money > 100)
        then
                System.out.println("满足抽奖条件");
                $user.setAllowLotteryFlag(1);
end
```

五、测试

```
public class DroolsMain {

    public static void main(String[] args) {
        KieServices kieServices = KieServices.Factory.get();
```

```
        //获取KieContainer对象
        KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
        //从Kie容器对象中获取会话对象，用于和规则引擎交互
        KieSession kieSession = kieClasspathContainer.newKieSession();

        User user = new User();
        user.setMoney(200);

        //将数据提供给规则引擎
        kieSession.insert(user);
        //激活规则引擎，由Drools框架自动进行规则匹配，如果规则匹配成功则
执行规则
        kieSession.fireAllRules();
        kieSession.dispose();
        System.out.println("是否允许抽奖：" +
user.getAllowLotteryFlag().equals(1));
    }
}

```

Aviator

> 一个高性能、轻量级的java语言实现的表达式求值引擎，主要用于各种表达式的动态求值

官方地址：

特点

* 支持大部分运算操作符，包括算术操作符、关系运算符、逻辑操作符、正则匹配操作符(=~)、三元表达式?:，并且支持操作符的优先级和括号强制优先级。
* 支持函数调用和自定义函数。
* 支持正则表达式匹配
* 自动类型转换
* 支持传入变量，支持类似a.b.c的嵌套变量访问。
* 性能优秀。
* Aviator的限制，没有if else、do while等语句，没有赋值语句，仅支持逻辑表达式、算术表达式、三元表达式和正则匹配，没有位运算符。

示例

下载依赖

```
        <dependency>
            <groupId>com.googlecode.aviator</groupId>
            <artifactId>aviator</artifactId>
            <version>5.2.7</version>
        </dependency>
Map<String, Object> map = new HashMap<String, Object>(8);
map.put(”money”, ”200”);
System.out.println(AviatorEvaluator.execute(”’充值金额: ’+ money”, map));
```

自定义函数

```
    static class AddFunction extends AbstractFunction {
        @Override
        public AviatorObject call(Map<String, Object> env, AviatorObject
arg1, AviatorObject arg2) {
            double num1 = FunctionUtils.getNumberValue(arg1,
env).doubleValue();
            double num2 = FunctionUtils.getNumberValue(arg2,
env).doubleValue();
            return new AviatorDouble(num1 + num2);
        }

        @Override
        public String getName() {
            return ”add”;
        }
    }

    public static void main(String[] args) {
        // 注册自定义函数
        AviatorEvaluator.addFunction(new AddFunction());
        System.out.println(AviatorEvaluator.execute(”add(100, 2003)”));
    }
```

编译表达式

```
    /**
     * 自定义表达式   *   * @param param   * @return
     */
    public static Object customizeExpression(Map<String, Object> param)
{
        String expression = "a+(b*c)";
        // 编译表达式
        Expression compiledExp = AviatorEvaluator.compile(expression);
        return compiledExp.execute(param);
    }

    public static void main(String[] args) {
        Map<String, Object> param = new HashMap<>(3);
        param.put("a", 100);
        param.put("b", 200);
        param.put("c", 300);
        System.out.println(customizeExpression(param));
    }
```

扩展阅读 [美团酒旅实时数据规则引擎应用实践](http://cxyroad.com/"https://tech.meituan.com/2018/04/19/hb-rt-operation.html")

MVEL 表达式解析器

> MVEL是基于java的表达式语言，但它是动态类型的，它是一种可嵌入的表达式语言和为Java平台提供运行时的语言。
>
>
> 官方文档：[mvel.documentnode.com/](http://cxyroad.com/"http://mvel.documentnode.com/")

特点

* 灵活，性能高，无类型限制

* 资料少，更新维护少

Easy Rule

> 一个简单而强大的java规则引擎，集成mvel表达式和SpEL表达式
>
>
> * 官方仓库地址：[github.com/j-easy/easy...](http://cxyroad.com/”https://github.com/j-easy/easy-rules”)

特点：

* 轻量级框架和易于学习的API
* 基于POJO的开发
* 通过高效的抽象来定义业务规则，使用简单
* 支持创建复合规则
* 支持使用表达式语言（如MVEL`[注释二]`和SpEL`[注释三]`）定义规则
* 规则很复杂，规则编排不好

规则的定义

* Name：一个命名空间下的唯一的规则名称
* Description：规则的简要描述
* Priority：相对于其他规则的优先级，默认为Integer.MAX，值越小，优先级越高
* Facts：用来传递数据，类似于Map
* Conditions：为了应用规则而必须满足的一组条件
* Actions：当条件满足时执行的一组动作

对应的注解：

@Condition、@Fact、@Action

规则的组成

* 简单规则

* 复合规则：CompositeRule

+ UnitRuleGroup：要么应用所有规则，要么不应用任何规则（AND逻辑）
+ ActivationRuleGroup：执行第一个条件成立的规则对应的action（XOR逻辑）
+ ConditionalRuleGroup：如果最高优先级的规则条件成立结果为true，则触发其余规则往下执行![image.png](https://p3-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/a5439ef6d2084f34b0316cfe2ae660fe~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=816&h=443&s=13163&e=png&b=232323)

组合规则示例：

```
    public class LotteryCompositeRule extends UnitRuleGroup {
        public LotteryCompositeRule(Object... rules) {
            for (Object rule : rules) {
                addRule(rule);
            }
        }

        @Override
        public int getPriority() {
            return 0;
        }

        @Rule(name = "vipRule", description = "抽奖规则")
        public static class VipRule {
            @Condition
            public boolean isVip(@Fact("user") User user) {
                return user.getVipFlag() == 1;
            }

            @Action
            public void print(@Fact("user") User user) {
                System.out.println("当前玩家为VIP用户");
            }

            @Priority
            public int getPriority() {
                return 1;
            }
        }
```

```
    @Rule(name = "MoneyRule", description = "抽奖规则")
    public static class MoneyRule {
        @Condition
        public boolean isAllowLottery(@Fact("user") User user) {
            return user.getMoney() >= 100;
        }

        @Action
        public void print(@Fact("user") User user) {
            System.out.println("当前玩家充值为:" + user.getMoney());
            user.setAllowLotteryFlag(1);
        }

        @Priority
        public int getPriority() {
            return 2;
        }
    }
}
```

创建规则的方式

* fulent api：链式编程

```
    public static Rule fluentRule() {
        return new RuleBuilder().name("链式抽奖规则").description("fluent
lottery rule").priority(2).when(f -> {
            User user = f.get("user");
            return user.getMoney() >= 100;
        }).then(f -> {
            System.out.println("满足抽奖条件");
            User user = f.get("user");
            user.setAllowLotteryFlag(1);
        }).build();
    }
```

* 注解

```
```

```
    @Rule(name = "lotteryRule", description = "抽奖规则")
    public class LotteryRule {
        @Condition
        public boolean isAllowLottery(@Fact("user") User user) {
            return user.getMoney() > 100;
        }

        @Action
        public void allowLottery(@Fact("user") User user) {
            user.setAllowLotteryFlag(1);
        }

        @Priority
        public int getPriority() {
            return 1;
        }
    }
```
* mvel表达式

```
    public static Rule mvelRule() {
        return new MVELRule().name("mvel lottery
rule").description("mvel表达式创建规则
").priority(3).when("user.getMoney() >=100")
                .then("user.setAllowLotteryFlag(1)");
    }
```
* spel表达式

```
    public static Rule spElRule() {
        return new SpELRule().name("SpEL lottery
rule").description("SpEl表达式创建规则
").priority(4).when("#{['user'].money > 2}")
                .then("#{['user'].setAllowLotteryFlag(1)}");
    }
```
* yaml形式

yaml文件

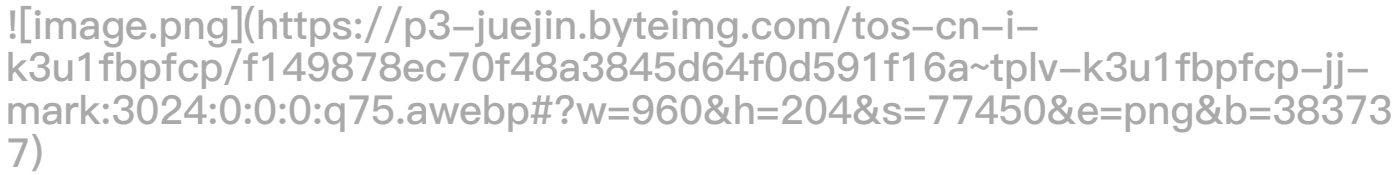
```
    name:”yaml rule”description:”yaml创建规则
”priority:5condition:”user.getMoney() >= 100”actions:-
”user.setAllowLotteryFlag(1);”

    public static Rule yamlRule() {
        SpELRuleFactory spELRuleFactory = new SpELRuleFactory(new
YamlRuleDefinitionReader());
        File ruleFile = new File(”src/main/resources/rules/lottery.yml”);
        try {
            return spELRuleFactory.createRule(new FileReader(ruleFile));
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
```

条件

接口为`Condition`，定义如下：

```
public interface Condition {    /**     * Evaluate the condition according
to the known facts.     *     * @param facts known when evaluating the
rule.     *     * @return true if the rule should be triggered, false
otherwise     */    boolean evaluate(Facts facts);  // ...}
```

evaluate方法用来判断规则是否被触发，Condition有几个实现类:



* SpELCondition：利用 SPEL表达式引擎解析规则

* MVELCondition：利用 MVEL表达式引擎解析规则
* JexlCondition：利用 JEXL表达式引擎解析规则

规则引擎

RulesEngine接口的两种实现

* DefaultRulesEngine：根据规则的自然顺序（默认为优先级）应用规则。
* InferenceRulesEngine：执行所有rules，直到rules为空。

引擎参数：

* skipOnFirstAppliedRule：告诉引擎当第一个规则被触发时跳过后面的规则。
* skipOnFirstFailedRule：告诉引擎当第一个规则失败时跳过后面的规则。
* skipOnFirstNonTriggeredRule：告诉引擎当一个规则没有被触发，那么跳过后面的规则。
* priorityThreshold：设置优先级阈值，优先级低于它的将会被跳过执行。

这些参数在RulesEngineParameters中设置，然后将其赋值给RulesEngine。例如：

```
RulesEngineParameters parameters = new RulesEngineParameters().skipOnFirstAppliedRule(true);DefaultRulesEngine rulesEngine = new DefaultRulesEngine(parameters);
```

创建规则引擎

带参数的方式：

```
RulesEngineParameters parameters = new RulesEngineParameters()DefaultRulesEngine rulesEngine = new DefaultRulesEngine(parameters);
```

```

**规则引擎执行**

```
rulesEngine.fire(rules, facts);
```

### 监听器

规则监听器RuleListener，定义：

```
  public interface RuleListener {
        // 在条件评估前调用
        default boolean beforeEvaluate(Rule rule, Facts facts) {
            return true;
        }

        // 在条件评估执行后调用
        default void afterEvaluate(Rule rule, Facts facts, boolean
evaluationResult) {
        }

        // 在条件评估中发生异常时调用
        default void onEvaluationError(Rule rule, Facts facts, Exception
exception) {
        }

        // 条件执行前调用
        default void beforeExecute(Rule rule, Facts facts) {
        }

        // 条件执行成功后调用
        default void onSuccess(Rule rule, Facts facts) {
        }

        // 条件执行失败后调用
        default void onFailure(Rule rule, Facts facts, Exception exception) {
        }
```
```

```

**注册监听器**

使用规则引擎的registerRuleListener方法即可，参数需要实现RuleListener接口，例如：

```
    public class LotteryListener implements RuleListener {
        /**
         * * @param rule   * @param facts   * @param evaluationResult
         */
        @Override
        public void afterEvaluate(org.jeasy.rules.api.Rule rule, Facts facts,
boolean evaluationResult) {
            if (!evaluationResult) {
                System.out.println("不满足抽奖条件");
            }
        }
    }
// 注册监听器
   rulesEngine.registerRuleListener(new LotteryListener());
```

### 示例

一、下载依赖

```
<dependency>  <groupId>org.jeasy</groupId>  <artifactId>easy-rules-
core</artifactId>  <version>4.1.0</version></dependency>  <!-- 可选
，支持mvel表达式 --><dependency>    <groupId>org.jeasy</groupId>
 <artifactId>easy-rules-mvel</artifactId>
 <version>4.1.0</version></dependency>
```

二、编写规则
```

```
    @Rule(name = "lotteryRule", description = "抽奖规则")
    public class LotteryRule {
        @Condition
        public boolean isAllowLottery(@Fact("user") User user) {
            return user.getMoney() >= 100;
        }

        @Action
        public void allowLottery(@Fact("user") User user) {
            user.setAllowLotteryFlag(1);
        }

        @Priority
        public int getPriority() {
            return 1;
        }
    }
```

三、运行

```
    public static void main(String[] args) {
        User user = new User();
        user.setMoney(200);
        Facts facts = new Facts();
        facts.put("user", user);
        DefaultRulesEngine rulesEngine = new DefaultRulesEngine();
        // 注册监听器
        rulesEngine.registerRuleListener(new LotteryListener());
        // 注册规则
        Rules rules = new Rules();
        rules.register(new LotteryRule());
        rulesEngine.fire(rules, facts);
        System.out.println("是否能够抽奖:" +
user.getAllowLotteryFlag().equals(1));
    }
```

注释
--

* 一、RETE算法

+ Rete 匹配算法是一种进行大量模式集合和大量对象集合间比较的高效方法，通过网络筛选的方法找出所有匹配各个模式的对象和规则。
+ 不足：以空间换时间，可能会耗尽系统资源
+ [houbb.github.io/2020/05/26/...](http://cxyroad.com/”https://houbb.github.io/2020/05/26/rule-engine-03-rete”)
* 二、mvel表达式文档：[mvel.documentnode.com/](http://cxyroad.com/”http://mvel.documentnode.com/”)
* 三、SPEL表达式介绍：[itmyhome.com/spring/expr...](http://cxyroad.com/”http://itmyhome.com/spring/expressions.html”)

参考地址
----

* [blog.csdn.net/justry_deng...](http://cxyroad.com/”https://blog.csdn.net/justry_deng/article/details/118096941”)
* [www.cnblogs.com/lyd44711373...](http://cxyroad.com/”https://www.cnblogs.com/lyd447113735/p/14814892.html”)
* [my.oschina.net/woniuyi/blo...](http://cxyroad.com/”https://my.oschina.net/woniuyi/blog/3119383”)
* [blog.csdn.net/wjc133/arti...](http://cxyroad.com/”https://blog.csdn.net/wjc133/article/details/111954160?utm_medium=distribute.pc_relevant.none-task-blog-2~default~OPENSEARCH~default-8.no_search_link&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2~default~OPENSEARCH~default-8.no_search_link”)

其他
--

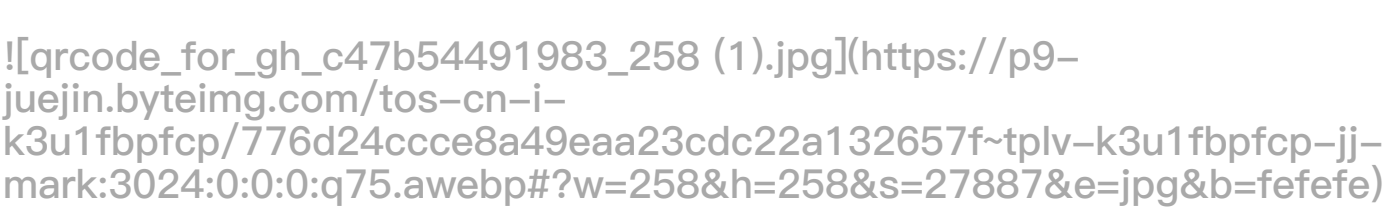
* 可视化规则引擎：[github.com/youseries/u...](http://cxyroad.com/”https://github.com/youseries/urule”)
* 风控引擎Radar：[gitee.com/freshday/ra...](http://cxyroad.com/”https://gitee.com/freshday/radar”)

例子地址
----

* [gitee.com/wubc/rule-e...](http://cxyroad.com/

”https://gitee.com/wubc/rule-engine-demo”)

欢迎：吴编程



原文链接: https://juejin.cn/post/7382931501610025000