1.Drools 规则引擎基本介绍

1.1Drools 基本介绍

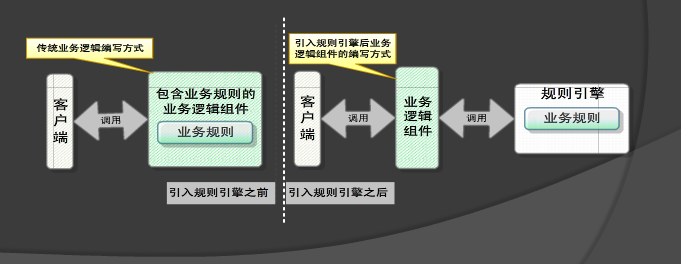
Drools是用Java语言编写的开放源码的规则引擎。

Drools使用RETE算法对规则进行求值，在Drools6.0（当前最新版本）中还引进了PHREAK算法，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，从而便于学习和理解。并且，还可以将 Java 代码直接嵌入到规则文件中，这令 Drools 的学习更加吸引人。

1.2那什么是规则引擎呢？

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。

1.3规则引擎的应用



1.4Drools新特性

Drools推出了一套新的基于KIE概念的API，其目的是将之前版本中对规则引擎繁琐的调用和加载过程加以简化。

Drools6给我的最大不同就是把rules打包成jar，使用端通过kie-ci来动态从maven repo中获取指定rules jar版本，虽然和maven有紧耦合，简化以及清晰了rules的使用和动态升级：系统建立2个项目：一个Drools项目来实现规则，验收规则，生成jar包，另外一个就是真正要用规则的项目，直接通过引入不同版本的jar包实习规则动态升级。

2.Drools 适用

2.1应用场景：

为提高效率，管理流程必须自动化，即使现代商业规则异常复杂。

市场要求业务规则经常变化，系统必须依据业务规则的变化快速、低成本的更新。

为了快速、低成本的更新，业务人员应能直接管系统中的规则，不需要程序开发人员参与。

2.2作用与优点：

将业务规则与业务系统分离，解耦合；

实现自然语言描述规则逻辑，业务人员易于理解；

可视化的规则定制、审批、查询及管理；

能有效的提高实现复杂逻辑的代码的可维护性；

应付特殊状况，即客户一开始没有提到要将业务逻辑考虑在内；

符合组织对敏捷或迭代开发过程的使用；

2.3 适用的系统

企业应用的ERP、CRM以及电子商务的销售系统及营销系统等

3.Drools 实例介绍

3.1Drools编译与运行

在Drools当中，规则的编译与运行要通过Drools提供的各种API来实现，这些API总体来讲可以分为三类：规则编译、规则收集和规则的执行。在Drools当中，规则的编译与运行要通过Drools提供的各种API来实现，这些API总体来讲可以分为三类：规则编译、规则收集和规则的执行。

3.2Kmodule.xml

kmodule.xml文件存放在src/main/resources/META-INF/文件夹下。

|  |  |
| --- | --- |
| 代码 | |
|  | <?xml version="1.0" encoding="UTF-8"?>  <kmodule xmlns="<http://jboss.org/kie/6.0.0/kmodule>">      <**kbase name="rules" packages="rules">**  **<ksessio**n name="ksession-rules"/>      </kbase>      <kbase name="dtables" packages="dtables">          <ksession name="ksession-dtables"/>      </kbase>  </kmodule> |

这个kmodule.xml的文件的定义非常简单，其实也很容易理解：

一个kmodule里面包含了两个kbase，这个也是我们这个例子里面的两个用例，分别对应drl规则文件的例子，一个是对应Excel表格的规则例子。

每一个kbase都有一个name，可以取任意字符串，但是不能重名。

然后都有一个packages，可以看到packages里面的字符串其实就是src/main/resources下面的文件夹的名称，或者叫包名，规则引擎会根据这里定义的包来查找规则定义文件。可以同时定义多个包，以逗号分隔开来就行。

每一个kbase下面可以包含多个ksession，当然本例中都自定义了一个。

每一个ksession都有一个name，名字也可以是任意字符串，但是也不能重复。

kbase和ksession里面的name属性是全局不能重复的。

kbase和ksession中其实还有很多其它的属性，但是在这里不是很重要，就先不提了，后面我们会一一讲解的。

这样一个kmodule.xml文件就建立好了。

在运行时，KieContainer会根据\*Model对象来创建KieModule、KieBase、KieSession对象。其中KieModule和KieBase只会创建一次，而KieSession则有可能创建多次，因为KieSession的创建成本很低，同时KieSession包含了运行时的数据，所以可以销毁、创建若干次。

3.3Drools运行测试

KieServices ks = KieServices.Factory.get();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession-rules");

// go !

Message message = new Message();

message.setMessage("Hello World");

message.setStatus(Message.HELLO);

kSession.insert(message);

kSession.fireAllRules();

kie框架先初始化一个kieservice对象,

从classpath中读取kmodule，创建KieContainder容器，

利用kieContainer对象创建一个新的KieSession，创建session的时候我们传入了一个name：“ksession-rules”，这个字符串很眼熟吧，这个就是我们定义的kmodule.xml文件中定义的ksession的name。kieContainer根据kmodule.xml定义的ksession的名称找到KieSession的定义，然后创建一个KieSession的实例。

KieSession就是一个到规则引擎的链接，通过它就可以跟规则引擎通讯，并且发起执行规则的操作。

通过kSession.insert方法来将事实（Fact）插入到引擎中，也就是Working Memory中。

然后通过kSession.fireAllRules方法来通知规则引擎执行规则。

3.4Drools语法

**package** com.yeepay.ytf.server.user.service

//对一个规则文件而言，package是必须定义的，必须放在规则文件第一行。

//特别的是，package的名字是随意的，不必必须对应物理路径，跟java的package的概念不同，这里只是逻辑上的一种区分。

//同样的package下定义的function和query等可以直接使用。

**import** com.yeepay.ytf.server.user.bean.User;

**import** com.yeepay.ytf.server.user.bean.Response;

**import** java.io.Console;

**import** java.util.Scanner;

**import** com.yeepay.ytf.server.user.service.impl.UserServiceImpl

//导入规则文件需要使用到的外部变量，这里的使用方法跟java相同，但是不同于java的是，这里的import导入的不仅仅可以是一个类，也可以是这个类中的某一个可访问的静态方法。

**rule** "userLogin"

//定义一个规则。rule "ruleName"。一个规则可以包含三个部分：

//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。

//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。

//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。

**salience** 100

// 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

// date-expires：设置规则的过期时间，默认的时间格式：“日-月-年” date-expires "2011-01-31 23:59:59"

// date-effective：设置规则的生效时间，时间格式同上

// 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

**when**

$user:User((phone==**null**||phone=="")&&

(passwd==**null**||passwd=="") )

// $user:User((phone==null||phone==""), (passwd==null||passwd=="") )

// 类型比较操作符 > >= < <= == != contains / not contains / memberOf / not memberOf /matches/ not matches

**then**

Response.setCode("00");

Response.setMesg("请输入用户名,密码");

// 可以调用Fact的方法，比如 $message.execute();操作数据库等等一切操作

// 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。

**End**

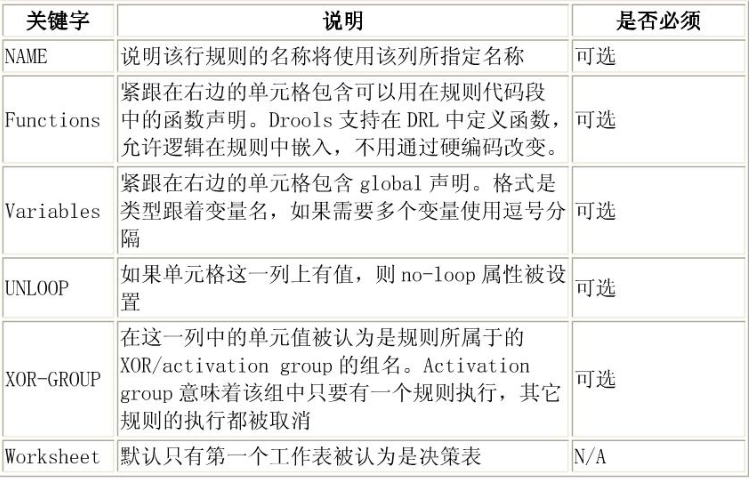
3.5Drools 决策表（Excel）

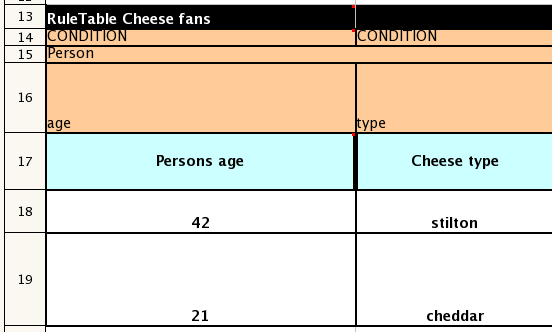
3.5.1 决策表的适用

* **何时使用决策表** **——** 如果规则能够被表达为模板+数据的格式，那你应该考虑使用决策表。决策表中的每一行就是对应模板的一行数据，将产生一个规则。
* **运行决策表** ——Drools引擎所部署的系统中执行决策表。首先，决策表转换成的Drools规则语言（DRL），然后执行规则引擎需求。这意味着它是可能的业务规则的变更，无需停止和启动，或重新部署任何软件。

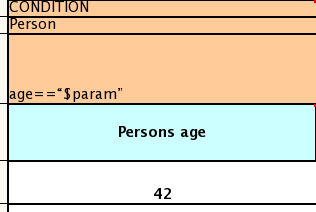
3.5.2决策表中的关键字

Warning

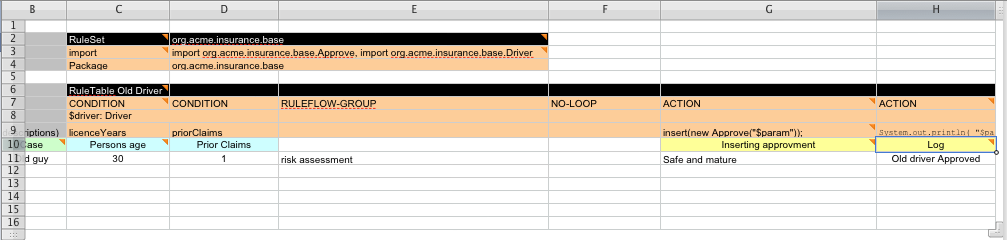




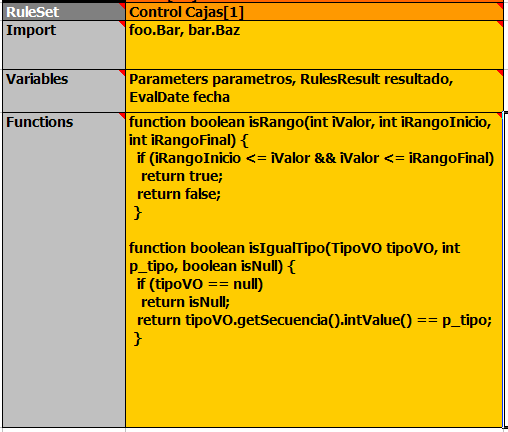
In the following example the marker symbol $param is used.



 Person(age == "42")).



Import, Variables and Functions的示例.



4 drools规则jar更新

4.1方法1

KieServices kieServices = KieServices.Factory.get();

ReleaseId releaseId = kieServices.newReleaseId( "org.acme", "myartifact", "1.0-SNAPSHOT" );

KieContainer kContainer = kieServices.newKieContainer( releaseId );

KieScanner kScanner = kieServices.newKieScanner( kContainer );

// Start the KieScanner polling the Maven repository every 10 seconds

kScanner.start( 10000L );

kScanner，遍历本地或远程maven 规则jar是否更新。

4.2方法2

**public** **void** initService(String groupId, String artifactId, String version)  **throws** NotExistsException {

    ks = KieServices.Factory.get();

**try**{

        kContainer = ks.newKieContainer(ks.newReleaseId(groupId, artifactId, version));

    } **catch** (Exception e){

        StringBuilder sb = **new** StringBuilder("fail to get maven rules kmodule from groupId: [");

        sb.append(groupId).append("] artifactId: [").append(artifactId).append("] version: [").append(version).append("]");

**throw** **new** NotExistsException(sb.toString());

    }

**if**(**null** != kContainer)

        kSession = kContainer.newStatelessKieSession("ksession-rules");

}

@Override

**public** **void** refreshService(String groupId, String artifactId, String version) **throws** NotExistsException {

**try**{

        kContainer.updateToVersion(ks.newReleaseId(groupId, artifactId, version));

    } **catch** (Exception e){

        StringBuilder sb = **new** StringBuilder("fail to get maven rules kmodule from groupId: [");

        sb.append(groupId).append("] artifactId: [").append(artifactId).append("] version: [").append(version).append("]");

**throw** **new** NotExistsException(sb.toString());

    }

**if**(**null** != kContainer)

        kSession = kContainer.newStatelessKieSession("ksession-rules");

}

@Override

**public** **void** fireService(Message info) {

    kSession.execute(info);

}

5 Drools规则 kmodul.xml 更新

**public** **void** initStateFulService(){

kieServices = KieServices.Factory.*get*();

resources = kieServices.getResources();

KieModuleModel kieModuleModel = kieServices.newKieModuleModel();//1

KieBaseModel baseModel = kieModuleModel.newKieBaseModel(

"FileSystemKBase").addPackage("rules");//2

baseModel.newKieSessionModel("FileSystemKSession");//3

fileSystem = kieServices.newKieFileSystem();

String xml = kieModuleModel.toXML();

System.*out*.println(xml);

fileSystem.writeKModuleXML(xml);//5

String fileBasePath = Thread.*currentThread*().getContextClassLoader().getResource("").getPath();

System.*out*.println(fileBasePath);

fileBasePath = fileBasePath.substring(0,fileBasePath.length());

List<String> fileList = GetAndReadAllFile.*getFileList*(**new** File(fileBasePath+"\\com\\yeepay\\ytf\\rules"),"drl");

**for** (String sfile : fileList) {

fileSystem.write("src/main/resources/rules/Rule.drl",

resources.newFileSystemResource(**new** File(sfile)));//6

}

KieBuilder kb = kieServices.newKieBuilder(fileSystem);

kb.buildAll();//7

**if** (kb.getResults().hasMessages(Level.*ERROR*)) {

**throw** **new** RuntimeException("Build Errors:\n"

+ kb.getResults().toString());

}

kContainer = kieServices.newKieContainer(kieServices

.getRepository().getDefaultReleaseId());

}

**public** **void** refreshStateFulService(String path){

fileSystem.write("src/main/resources/rules/UploadRule.drl",resources.newFileSystemResource(path));//6

KieBuilder kb = kieServices.newKieBuilder(fileSystem);

kb.buildAll();//7

**if** (kb.getResults().hasMessages(Level.*ERROR*)) {

**throw** **new** RuntimeException("Build Errors:\n"

+ kb.getResults().toString());

}

kContainer = kieServices.newKieContainer(kieServices

.getRepository().getDefaultReleaseId());

// kSession = kContainer.newKieSession("FileSystemKSession");

}

6.Kie pom.xml依赖

pom中加入：

<dependency>

<groupId>org.jboss.dashboard-builder</groupId>

<artifactId>dashboard-builder-bom</artifactId>

<version>6.1.0.Final</version>

</dependency>