

规则引擎 Drools

1. 问题引出

现有一个在线申请信用卡的业务场景，用户需要录入个人信息，如下图所示：

XX银行信用卡申请表

基本信息

姓名：

性别：

年龄：

学历：

电话：

月收入：

邮寄地址：

其他信息

是否有房：

☐

是否有车：

☐

现持有信用卡数量：

提交申请

通过上图可以看到，用户录入的个人信息包括姓名、性别、年龄、学历、电话、所在公司、职位、月收入、是否有房、是否有车、是否有信用卡等。录入完成后点击申请按钮提交即可。

用户提交申请后，需要在系统的服务端进行**用户信息合法性检查**(是否有资格申请信用卡)，只有通过合法性检查的用户才可以成功申请到信用卡(注意：不同用户有可能申请到的信用卡额度不同)。

检查用户信息合法性的规则如下：

| 规则编号 | 名称 | 描述 |
|------|---------------|---|
| 1 | 检查学历与薪水1 | 如果申请人既没房也没车，同时学历为大专以下，并且月薪少于5000，那么不通过 |
| 2 | 检查学历与薪水2 | 如果申请人既没房也没车，同时学历为大专或本科，并且月薪少于3000，那么不通过 |
| 3 | 检查学历与薪水3 | 如果申请人既没房也没车，同时学历为本科以上，并且月薪少于2000，同时之前没有信用卡的，那么不通过 |
| 4 | 检查申请人已有的信用卡数量 | 如果申请人现有的信用卡数量大于10，那么不通过 |

用户信息合法性检查通过后，还需要根据如下**信用卡发放规则**确定用户所办信用卡的额度：

| 规则编号 | 名称 | 描述 |
|------|-----|---|
| 1 | 规则1 | 如果申请人有房有车，或者月收入在20000以上，那么发放的信用卡额度为15000 |
| 2 | 规则2 | 如果申请人没房没车，但月收入在10000~20000之间，那么发放的信用卡额度为6000 |
| 3 | 规则3 | 如果申请人没房没车，月收入在10000以下，那么发放的信用卡额度为3000 |
| 4 | 规则4 | 如果申请人有房没车或者没房但有车，月收入在10000以下，那么发放的信用卡额度为5000 |
| 5 | 规则5 | 如果申请人有房没车或者是没房但有车，月收入在10000~20000之间，那么发放的信用卡额度为8000 |

思考：如何实现上面的业务逻辑呢？

我们最容易想到的就是使用分支判断(if else)来实现，例如通过如下代码来检查用户信息合法性：

```
//此处为伪代码

//检查用户信息合法性，返回true表示检查通过，返回false表示检查不通过
public boolean checkUser(User user){
    //如果申请人既没房也没车，同时学历为大专以下，并且月薪少于5000，那么不通过
```

```

if(user.getHouse() == null
    && user.getcar() == null
    && user.getEducation().equals("大专以下")
    && user.getSalary < 5000){
    return false;
}
//如果申请人既没房也没车，同时学历为大专或本科，并且月薪少于3000，那么不通过
else if(user.getHouse() == null
    && user.getcar() == null
    && user.getEducation().equals("大专或本科")
    && user.getSalary < 3000){
    return false;
}
//如果申请人既没房也没车，同时学历为本科以上，并且月薪少于2000，同时之前没有
信用卡的，那么不通过
else if(user.getHouse() == null
    && user.getcar() == null
    && user.getEducation().equals("本科以上")
    && user.getSalary < 2000
    && user.getHasCreditCard() == false){
    return false;
}
//如果申请人现有的信用卡数量大于10，那么不通过
else if(user.getCreditCardCount() > 10){
    return false;
}
return true;
}

```

如果用户信息合法性检查通过后，还需要通过如下代码确定用户所办信用卡的额度：

```

//此处为伪代码

//根据用户输入信息确定信用卡额度
public Integer determineCreditCardLimit(User user){
    //如果申请人有房有车，或者月收入在20000以上，那么发放的信用卡额度为15000
    if((user.getHouse() != null && user.getcar() != null)
        || user.getSalary() > 20000){
        return 15000;
    }
    //如果申请人没房没车，并且月收入在10000到20000之间，那么发放的信用卡额度为
    6000
    else if(user.getHouse() == null
        && user.getcar() == null
        && user.getSalary() > 10000
        && user.getSalary() < 20000){
        return 6000;
    }
    //如果申请人没房没车，并且月收入在10000以下，那么发放的信用卡额度为3000
    else if(user.getHouse() == null
        && user.getcar() == null

```

```
        && user.getSalary() < 10000){
            return 3000;
        }
        //如果申请人有房没车或者没房但有车，并且月收入在10000以下，那么发放的信用卡
        额度为5000
        else if((((user.getHouse() != null && user.getcar() == null) ||
        (user.getHouse() == null && user.getcar() != null))
            && user.getSalary() < 10000){
                return 5000;
            }
            //如果申请人有房没车或者没房但有车，并且月收入在10000到20000之间，那么发放
            的信用卡额度为8000
            else if((((user.getHouse() != null && user.getcar() == null) ||
            (user.getHouse() == null && user.getcar() != null))
                && (user.getSalary() > 10000 && user.getSalary() < 20000)){
                    return 8000;
                }
            }
        }
```

通过上面的伪代码我们可以看到，我们的业务规则是通过Java代码的方式实现的。这种实现方式存在如下问题：

- 1、硬编码实现业务规则难以维护
- 2、硬编码实现业务规则难以应对变化
- 3、业务规则发生变化需要修改代码，重启服务后才能生效

那么面对上面的业务场景，还有什么好的实现方式吗？

答案是**规则引擎**。

2. 规则引擎概述

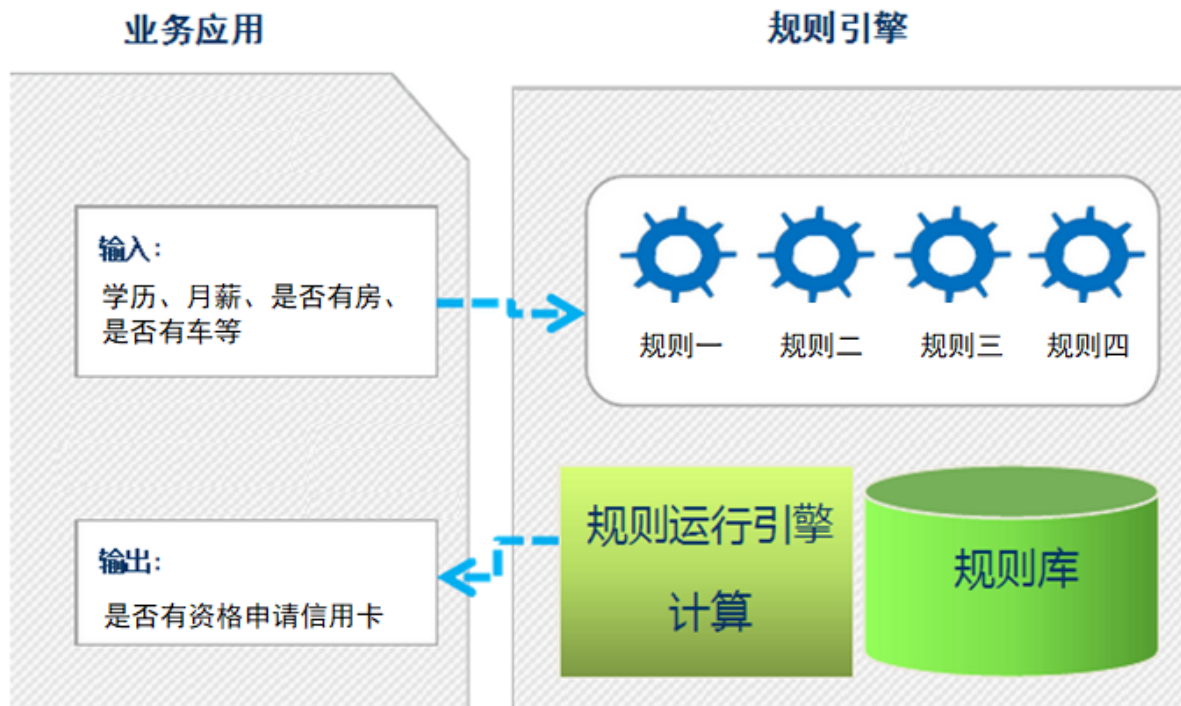
2.1 什么是规则引擎

规则引擎，全称为**业务规则管理系统**，英文名为BRMS(即Business Rule Management System)。规则引擎的主要思想是将应用程序中的业务决策部分分离出来，并使用预定义的语义模块编写业务决策（业务规则），由用户或开发者在需要时进行配置、管理。

需要注意的是规则引擎并不是一个具体的技术框架，而是指的一类系统，即业务规则管理系统。目前市面上具体的规则引擎产品有：drools、VisualRules、iLog等。

规则引擎实现了将业务决策从应用程序代码中分离出来，接收数据输入，解释业务规则，并根据业务规则做出业务决策。规则引擎其实就是一个输入输出平台。

上面的申请信用卡业务场景使用规则引擎后效果如下：



系统中引入规则引擎后，业务规则不再以程序代码的形式驻留在系统中，取而代之的是处理规则的规则引擎，业务规则存储在规则库中，完全独立于程序。业务人员可以像管理数据一样对业务规则进行管理，比如查询、添加、更新、统计、提交业务规则等。业务规则被加载到规则引擎中供应用系统调用。

2.2 使用规则引擎的优势

使用规则引擎的优势如下：

- 1、业务规则与系统代码分离，实现业务规则的集中管理
- 2、在不重启服务的情况下可随时对业务规则进行扩展和维护
- 3、可以动态修改业务规则，从而快速响应需求变更
- 4、规则引擎是相对独立的，只关心业务规则，使得业务分析人员也可以参与编辑、维护系统的业务规则
- 5、减少了硬编码业务规则的成本和风险
- 6、使用规则引擎提供的规则编辑工具，使复杂的业务规则实现变得简单

2.3 规则引擎应用场景

对于一些存在比较复杂的业务规则并且业务规则会频繁变动的系统比较适合使用规则引擎，如下：

- 1、风险控制系统----风险贷款、风险评估
- 2、反欺诈项目----银行贷款、征信验证
- 3、决策平台系统----财务计算
- 4、促销平台系统----满减、打折、加价购

2.4 Drools介绍

drools是一款由JBoss组织提供的基于Java语言开发的开源规则引擎，可以将复杂且多变的业务规则从硬编码中解放出来，以规则脚本的形式存放在文件或特定的存储介质中(例如存放在数据库中)，使得业务规则的变更不需要修改项目代码、重启服务器就可以在线上环境立即生效。

drools官网地址：<https://drools.org/>

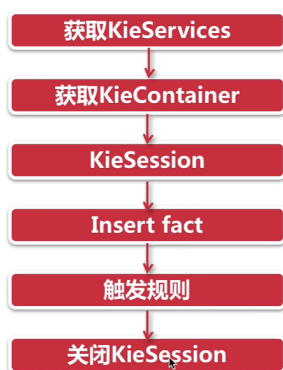
drools源码下载地址：<https://github.com/kiegroup/drools>

在项目中使用drools时，即可以单独使用也可以整合spring使用。如果单独使用只需要导入如下maven坐标即可：

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <version>7.6.0.Final</version>
</dependency>
```

如果我们使用IDEA开发drools应用，IDEA中已经集成了drools插件。如果使用eclipse开发drools应用还需要单独安装drools插件。

drools API开发步骤如下：



3. Drools入门案例

本小节通过一个Drools入门案例来让大家初步了解Drools的使用方式、对Drools有一个整体概念。

3.1 业务场景说明

业务场景：消费者在图书商城购买图书，下单后需要在支付页面显示订单优惠后的价格。具体优惠规则如下：

| 规则编号 | 规则名称 | 描述 |
|------|------|-----------------------|
| 1 | 规则一 | 所购图书总价在100元以下的没有优惠 |
| 2 | 规则二 | 所购图书总价在100到200元的优惠20元 |
| 3 | 规则三 | 所购图书总价在200到300元的优惠50元 |
| 4 | 规则四 | 所购图书总价在300元以上的优惠100元 |

现在需要根据上面的规则计算优惠后的价格。

3.2 开发实现

第一步：创建maven工程drools_quickstart并导入drools相关maven坐标

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <version>7.10.0.Final</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
```

第二步：根据drools要求创建resources/META-INF/kmodule.xml配置文件

```
<?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
    default:指定当前kbase是否为默认
  -->
  <kbase name="myKbase1" packages="rules" default="true">
    <!--
      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>
```

注意：上面配置文件的名称和位置都是固定写法，不能更改

第三步：创建实体类Order

```
package com.itheima.drools.entity;

/**
 * 订单
 */
public class Order {
  private Double originalPrice;//订单原始价格，即优惠前价格
  private Double realPrice;//订单真实价格，即优惠后价格

  public String toString() {
    return "Order{" +
```

```

        "originalPrice=" + originalPrice +
        ", realPrice=" + realPrice +
        '}';
    }

    public Double getOriginalPrice() {
        return originalPrice;
    }

    public void setOriginalPrice(Double originalPrice) {
        this.originalPrice = originalPrice;
    }

    public Double getRealPrice() {
        return realPrice;
    }

    public void setRealPrice(Double realPrice) {
        this.realPrice = realPrice;
    }
}

```

第四步：创建规则文件resources/rules/bookDiscount.drl

```

//图书优惠规则
package book.discount
import com.itheima.drools.entity.Order

//规则一：所购图书总价在100元以下的没有优惠
rule "book_discount_1"
    when
        $order:Order(originalPrice < 100)
    then
        $order.setRealPrice($order.getOriginalPrice());
        System.out.println("成功匹配到规则一：所购图书总价在100元以下的没有优惠");
    end

//规则二：所购图书总价在100到200元的优惠20元
rule "book_discount_2"
    when
        $order:Order(originalPrice < 200 && originalPrice >= 100)
    then
        $order.setRealPrice($order.getOriginalPrice() - 20);
        System.out.println("成功匹配到规则二：所购图书总价在100到200元的优惠20元");
    end

//规则三：所购图书总价在200到300元的优惠50元
rule "book_discount_3"
    when

```



```

        $order:Order(originalPrice <= 300 && originalPrice >= 200)
    then
        $order.setRealPrice($order.getOriginalPrice() - 50);
        System.out.println("成功匹配到规则三：所购图书总价在200到300元的优惠
50元");
    end

//规则四：所购图书总价在300元以上的优惠100元
rule "book_discount_4"
    when
        $order:Order(originalPrice >= 300)
    then
        $order.setRealPrice($order.getOriginalPrice() - 100);
        System.out.println("成功匹配到规则四：所购图书总价在300元以上的优惠100
元");
    end
end

```

第五步：编写单元测试

```

@Test
public void test1(){
    Kieservices kieservices = Kieservices.Factory.get();
    KieContainer kieClasspathContainer =
kieservices.getKieClasspathContainer();
    //会话对象，用于和规则引擎交互
    KieSession kieSession = kieClasspathContainer.newKieSession();

    //构造订单对象，设置原始价格，由规则引擎根据优惠规则计算优惠后的价格
    Order order = new Order();
    order.setOriginalPrice(2100);

    //将数据提供给规则引擎，规则引擎会根据提供的数据进行规则匹配
    kieSession.insert(order);

    //激活规则引擎，如果规则匹配成功则执行规则
    kieSession.fireAllRules();
    //关闭会话
    kieSession.dispose();

    System.out.println("优惠前原始价格：" + order.getOriginalPrice() +
        "，优惠后价格：" + order.getRealPrice());
}

```

通过上面的入门案例我们可以发现，使用drools规则引擎主要工作就是编写规则文件，在规则文件中定义跟业务相关的业务规则，例如本案例定义的就是图书优惠规则。规则定义好后就需要调用drools提供的API将数据提供给规则引擎进行规则模式匹配，规则引擎会执行匹配成功的规则并将计算的结果返回给我们。

可能大家会有疑问，就是我们虽然没有在代码中编写规则的判断逻辑，但是我们还是在规则文件中编写了业务规则，这跟在代码中编写规则有什么本质的区别呢？

我们前面其实已经提到，使用规则引擎时业务规则可以做到动态管理。业务人员可以像管理数据一样对业务规则进行管理，比如查询、添加、更新、统计、提交业务规则等。这样就可以做到在不重启服务的情况下调整业务规则。

3.3 小结

3.3.1 规则引擎构成

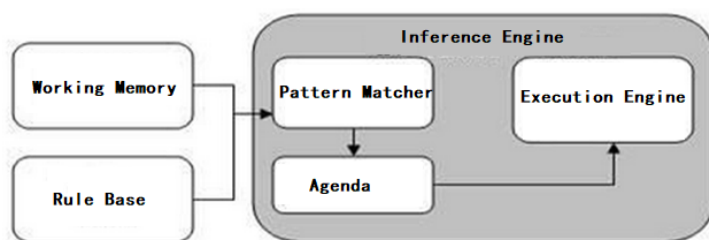
drools规则引擎由以下三部分构成：

- Working Memory（工作内存）
- Rule Base（规则库）
- Inference Engine（推理引擎）

其中Inference Engine（推理引擎）又包括：

- Pattern Matcher（匹配器）
- Agenda(议程)
- Execution Engine（执行引擎）

如下图所示：



3.3.2 相关概念说明

Working Memory：工作内存，drools规则引擎会从Working Memory中获取数据并和规则文件中定义的规则进行模式匹配，所以我们开发的应用程序只需要将我们的数据插入到Working Memory中即可，例如本案例中我们调用`kieSession.insert(order)`就是将`order`对象插入到了工作内存中。

Fact：事实，是指在drools 规则应用当中，将一个普通的JavaBean插入到Working Memory后的对象就是Fact对象，例如本案例中的`Order`对象就属于Fact对象。Fact对象是我们的应用和规则引擎进行数据交互的桥梁或通道。

Rule Base：规则库，我们在规则文件中定义的规则都会被加载到规则库中。

Pattern Matcher：匹配器，将Rule Base中的所有规则与Working Memory中的Fact对象进行模式匹配，匹配成功的规则将被激活并放入Agenda中。

Agenda：议程，用于存放通过匹配器进行模式匹配后被激活的规则。

Execution Engine：执行引擎，执行Agenda中被激活的规则。

3.3.3 规则引擎执行过程

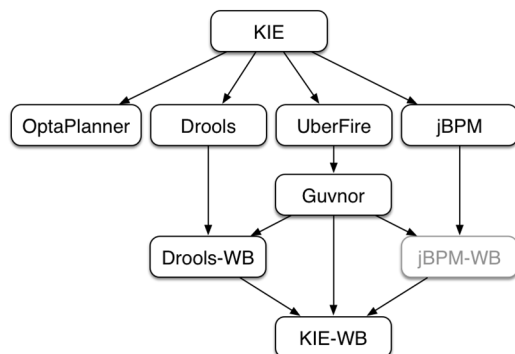


3.3.4 KIE介绍

我们在操作Drools时经常使用的API以及它们之间的关系如下图：



通过上面的核心API可以发现，大部分类名都是以Kie开头。Kie全称为Knowledge Is Everything，即"知识就是一切"的缩写，是jboss一系列项目的总称。如下图所示，Kie的主要模块有OptaPlanner、Drools、UberFire、jBPM。



通过上图可以看到，Drools是整个KIE项目中的一个组件，Drools中还包括一个Drools-WB的模块，它是一个可视化的规则编辑器。

4. Drools基础语法

4.1 规则文件构成

在使用Drools时非常重要的一个工作就是编写规则文件，通常规则文件的后缀为.drl。

drl是Drools Rule Language的缩写。在规则文件中编写具体的规则内容。

一套完整的规则文件内容构成如下：

| 关键字 | 描述 |
|----------|----------------------------------|
| package | 包名，只限于逻辑上的管理，同一个包名下的查询或者函数可以直接调用 |
| import | 用于导入类或者静态方法 |
| global | 全局变量 |
| function | 自定义函数 |
| query | 查询 |
| rule end | 规则体 |

Drools支持的规则文件，除了drl形式，还有Excel文件类型的。

4.2 规则体语法结构

规则体是规则文件内容中的重要组成部分，是进行业务规则判断、处理业务结果的部分。

规则体语法结构如下：

```
rule "ruleName"  
    attributes  
    when  
        LHS  
    then  
        RHS  
end
```

rule：关键字，表示规则开始，参数为规则的唯一名称。

attributes：规则属性，是rule与when之间的参数，为可选项。

when：关键字，后面跟规则的条件部分。

LHS(Left Hand Side)：是规则的条件部分的通用名称。它由零个或多个条件元素组成。如果LHS为空，则它将被视为始终为true的条件元素。

then：关键字，后面跟规则的结果部分。

RHS(Right Hand Side)：是规则的后果或行动部分的通用名称。

end：关键字，表示一个规则结束。

4.3 注释

在drl形式的规则文件中使用注释和Java类中使用注释一致，分为单行注释和多行注释。

单行注释用“//”进行标记，多行注释以“/”开始，以“/”结束。如下示例：

```
//规则 rule1 的注释，这是一个单行注释  
rule "rule1"  
    when  
    then
```

```

        System.out.println("rule1触发");
    end

    /*
    规则 rule2 的注释，
    这是一个多行注释
    */
    rule "rule2"
        when
        then
            System.out.println("rule2触发");
        end
    end

```

4.4 Pattern模式匹配

前面我们已经知道了Drools中的匹配器可以将Rule Base中的所有规则与Working Memory中的Fact对象进行模式匹配，那么我们就需要在规则体的LHS部分定义规则并进行模式匹配。LHS部分由一个或者多个条件组成，条件又称为pattern。

pattern的语法结构为：绑定变量名:Object(Field约束)

其中绑定变量名可以省略，通常绑定变量名的命名一般建议以\$开始。如果定义了绑定变量名，就可以在规则体的RHS部分使用此绑定变量名来操作相应的Fact对象。Field约束部分是需要返回true或者false的0个或多个表达式。

例如我们的入门案例中：

```

//规则二：所购图书总价在100到200元的优惠20元
rule "book_discount_2"
    when
        //Order为类型约束，originalPrice为属性约束
        $order:Order(originalPrice < 200 && originalPrice >= 100)
    then
        $order.setRealPrice($order.getOriginalPrice() - 20);
        System.out.println("成功匹配到规则二：所购图书总价在100到200元的优惠
20元");
    end

```

通过上面的例子我们可以知道，匹配的条件为：

- 1、工作内存中必须存在Order这种类型的Fact对象-----类型约束
- 2、Fact对象的originalPrice属性值必须小于200-----属性约束
- 3、Fact对象的originalPrice属性值必须大于等于100-----属性约束

以上条件必须同时满足当前规则才有可能被激活。

绑定变量既可以用在对象上，也可以用在对象的属性上。例如上面的例子可以改为：

```
//规则二：所购图书总价在100到200元的优惠20元
rule "book_discount_2"
    when
        $order:Order($op:originalPrice < 200 && originalPrice >= 100)
    then
        System.out.println("$op=" + $op);
        $order.setRealPrice($order.getOriginalPrice() - 20);
        System.out.println("成功匹配到规则二：所购图书总价在100到200元的优惠
20元");
    end
```

LHS部分还可以定义多个pattern，多个pattern之间可以使用and或者or进行连接，也可以不写，默认连接为and。

```
//规则二：所购图书总价在100到200元的优惠20元
rule "book_discount_2"
    when
        $order:Order($op:originalPrice < 200 && originalPrice >= 100) and
        $customer:Customer(age > 20 && gender=='male')
    then
        System.out.println("$op=" + $op);
        $order.setRealPrice($order.getOriginalPrice() - 20);
        System.out.println("成功匹配到规则二：所购图书总价在100到200元的优惠
20元");
    end
```

4.5 比较操作符

Drools提供的比较操作符，如下表：

| 符号 | 说明 |
|-----------------|--------------------------------------|
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |
| == | 等于 |
| != | 不等于 |
| contains | 检查一个Fact对象的某个属性值是否包含一个指定的对象值 |
| not contains | 检查一个Fact对象的某个属性值是否不包含一个指定的对象值 |
| memberOf | 判断一个Fact对象的某个属性是否在一个或多个集合中 |
| not memberOf | 判断一个Fact对象的某个属性是否不在一个或多个集合中 |
| matches | 判断一个Fact对象的属性是否与提供的标准的Java正则表达式进行匹配 |
| not matches | 判断一个Fact对象的属性是否不与提供的标准的Java正则表达式进行匹配 |

前6个比较操作符和Java中的完全相同，下面我们重点学习后6个比较操作符。

4.5.1 语法

- **contains | not contains语法结构**

Object(Field[Collection/Array] contains value)

Object(Field[Collection/Array] not contains value)

- **memberOf | not memberOf语法结构**

Object(field memberOf value[Collection/Array])

Object(field not memberOf value[Collection/Array])

- **matches | not matches语法结构**

Object(field matches "正则表达式")

Object(field not matches "正则表达式")

4.5.2 操作步骤

第一步：创建实体类，用于测试比较操作符

```
package com.itheima.drools.entity;
import java.util.List;
```

```
/**
 * 实体类
```

```

    * 用于测试比较操作符
    */
public class ComparisonOperatorEntity {
    private String names;
    private List<String> list;

    public String getNames() {
        return names;
    }

    public void setNames(String names) {
        this.names = names;
    }

    public List<String> getList() {
        return list;
    }

    public void setList(List<String> list) {
        this.list = list;
    }
}

```

第二步：在/resources/rules下创建规则文件comparisonOperator.drl

```

package comparisonOperator
import com.itheima.drools.entity.ComparisonOperatorEntity
/*
    当前规则文件用于测试Drools提供的比较操作符
    */

//测试比较操作符contains
rule "rule_comparison_contains"
    when
        ComparisonOperatorEntity(names contains "张三")
        ComparisonOperatorEntity(list contains names)
    then
        System.out.println("规则rule_comparison_contains触发");
    end

//测试比较操作符not contains
rule "rule_comparison_notContains"
    when
        ComparisonOperatorEntity(names not contains "张三")
        ComparisonOperatorEntity(list not contains names)
    then
        System.out.println("规则rule_comparison_notContains触发");
    end

//测试比较操作符memberOf
rule "rule_comparison_memberOf"

```



```

    when
        ComparisonOperatorEntity(names memberOf list)
    then
        System.out.println("规则rule_comparison_memberOf触发");
end

//测试比较操作符not memberOf
rule "rule_comparison_notMemberOf"
    when
        ComparisonOperatorEntity(names not memberOf list)
    then
        System.out.println("规则rule_comparison_notMemberOf触发");
end

//测试比较操作符matches
rule "rule_comparison_matches"
    when
        ComparisonOperatorEntity(names matches "张.*")
    then
        System.out.println("规则rule_comparison_matches触发");
end

//测试比较操作符not matches
rule "rule_comparison_notMatches"
    when
        ComparisonOperatorEntity(names not matches "张.*")
    then
        System.out.println("规则rule_comparison_notMatches触发");
end

```

第三步：编写单元测试

```

//测试比较操作符
@Test
public void test3(){
    KieServices kieServices = KieServices.Factory.get();
    KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
    KieSession kieSession = kieClasspathContainer.newKieSession();

    ComparisonOperatorEntity comparisonOperatorEntity = new
ComparisonOperatorEntity();
    comparisonOperatorEntity.setNames("张三");
    List<String> list = new ArrayList<String>();
    list.add("张三");
    list.add("李四");
    comparisonOperatorEntity.setList(list);

    //将数据提供给规则引擎，规则引擎会根据提供的数据进行规则匹配，如果规则匹配成功则执行规则
    kieSession.insert(comparisonOperatorEntity);
}

```

```
kiesession.fireAllRules();
kiesession.dispose();
}
```

4.6 执行指定规则

通过前面的案例可以看到，我们在调用规则代码时，满足条件的规则都会被执行。那么如果我们只想执行其中的某个规则如何实现呢？

Drools给我们提供的方式是通过规则过滤器来实现执行指定规则。对于规则文件不用做任何修改，只需要修改Java代码即可，如下：

```
KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kiesession = kieClasspathContainer.newKieSession();

ComparisonOperatorEntity comparisonOperatorEntity = new
ComparisonOperatorEntity();
comparisonOperatorEntity.setNames("张三");
List<String> list = new ArrayList<String>();
list.add("张三");
list.add("李四");
comparisonOperatorEntity.setList(list);
kiesession.insert(comparisonOperatorEntity);

//通过规则过滤器实现只执行指定规则
kiesession.fireAllRules(new
RuleNameEqualsAgendaFilter("rule_comparison_memberOf"));

kiesession.dispose();
```

4.7 关键字

Drools的关键字分为：硬关键字(Hard keywords)和软关键字(Soft keywords)。

硬关键字是我们在规则文件中定义包名或者规则名时明确不能使用的，否则程序会报错。软关键字虽然可以使用，但是不建议使用。

硬关键字包括：true false null

软关键字包括：lock-on-active date-effective date-expires no-loop auto-focus activation-group agenda-group ruleflow-group entry-point duration package import dialect salience enabled attributes rule extend when then template query declare function global eval not in or and exists forall accumulate collect from action reverse result end over init

4.8 Drools内置方法

规则文件的 `RHS` 部分的主要作用是通过插入，删除或修改工作内存中的Fact数据，来达到控制规则引擎执行的目的。Drools提供了一些方法可以用来操作工作内存中的数据，操作完成后规则引擎会重新进行相关规则的匹配，原来没有匹配成功的规则在我们修改数据完成后有可能就会匹配成功了。

创建如下实体类：

```
package com.itheima.drools.entity;

import java.util.List;

/**
 * 学生
 */
public class Student {
    private int id;
    private String name;
    private int age;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

4.8.1 update方法

update方法的作用是更新工作内存中的数据，并让相关的规则重新匹配。

第一步：编写规则文件/resources/rules/student.drl，文件内容如下

```
package student
import com.itheima.drools.entity.Student
```

```

/*
 当前规则文件用于测试Drools提供的内置方法
*/

rule "rule_student_age小于10岁"
  when
    $s:Student(age < 10)
  then
    $s.setAge(15);
    update($s); //更新数据，导致相关的规则会重新匹配
    System.out.println("规则rule_student_age小于10岁触发");
  end

rule "rule_student_age小于20岁同时大于10岁"
  when
    $s:Student(age < 20 && age > 10)
  then
    $s.setAge(25);
    update($s); //更新数据，导致相关的规则会重新匹配
    System.out.println("规则rule_student_age小于20岁同时大于10岁触发");
  end

rule "rule_student_age大于20岁"
  when
    $s:Student(age > 20)
  then
    System.out.println("规则rule_student_age大于20岁触发");
  end
end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

Student student = new Student();
student.setAge(5);

//将数据提供给规则引擎，规则引擎会根据提供的数据进行规则匹配，如果规则匹配成功则
//执行规则
kieSession.insert(student);

kieSession.fireAllRules();
kieSession.dispose();

```

通过控制台的输出可以看到规则文件中定义的三个规则都触发了。

在更新数据时需要注意防止发生死循环。

4.8.2 insert方法

insert方法的作用是向工作内存中插入数据，并让相关的规则重新匹配。

第一步：修改student.drl文件内容如下

```
package student
import com.itheima.drools.entity.Student

/*
  当前规则文件用于测试Drools提供的内置方法
*/

rule "rule_student_age等于10岁"
  when
    $s:Student(age == 10)
  then
    Student student = new Student();
    student.setAge(5);
    insert(student); //插入数据，导致相关的规则会重新匹配
    System.out.println("规则rule_student_age等于10岁触发");
  end

rule "rule_student_age小于10岁"
  when
    $s:Student(age < 10)
  then
    $s.setAge(15);
    update($s);
    System.out.println("规则rule_student_age小于10岁触发");
  end

rule "rule_student_age小于20岁同时大于10岁"
  when
    $s:Student(age < 20 && age > 10)
  then
    $s.setAge(25);
    update($s);
    System.out.println("规则rule_student_age小于20岁同时大于10岁触发");
  end

rule "rule_student_age大于20岁"
  when
    $s:Student(age > 20)
  then
    System.out.println("规则rule_student_age大于20岁触发");
  end
```

第二步：编写单元测试

```

kieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

Student student = new Student();
student.setAge(10);

//将数据提供给规则引擎，规则引擎会根据提供的数据进行规则匹配，如果规则匹配成功则
//执行规则
kieSession.insert(student);

kieSession.fireAllRules();
kieSession.dispose();

```

通过控制台输出可以发现，四个规则都触发了，这是因为首先进行规则匹配时只有第一个规则可以匹配成功，但是在第一个规则中向工作内存中插入了一个数据导致重新进行规则匹配，此时第二个规则可以匹配成功。在第二个规则中进行了数据修改导致第三个规则也可以匹配成功，以此类推最终四个规则都匹配成功并执行了。

4.8.3 retract方法

retract方法的作用是删除工作内存中的数据，并让相关的规则重新匹配。

第一步：修改student.drl文件内容如下

```

package student
import com.itheima.drools.entity.Student

/*
  当前规则文件用于测试Drools提供的内置方法
*/

rule "rule_student_age等于10岁时删除数据"
/*
  salience: 设置当前规则的执行优先级，数值越大越优先执行，默认值为0。
  因为当前规则的匹配条件和下面规则的匹配条件相同，为了保证先执行当前规则，需要
  设置优先级
  */
  salience 100
  when
    $s:Student(age == 10)
  then
    retract($s); //retract方法的作用是删除工作内存中的数据，并让相关的规则
    重新匹配。
    System.out.println("规则rule_student_age等于10岁时删除数据触发");
  end

rule "rule_student_age等于10岁"
  when
    $s:Student(age == 10)

```

```

        then
            Student student = new Student();
            student.setAge(5);
            insert(student);
            System.out.println("规则 rule_student_age 等于 10 岁 触发");
        end

rule "rule_student_age 小于 10 岁"
    when
        $s:Student(age < 10)
    then
        $s.setAge(15);
        update($s);
        System.out.println("规则 rule_student_age 小于 10 岁 触发");
    end

rule "rule_student_age 小于 20 岁 同时 大于 10 岁"
    when
        $s:Student(age < 20 && age > 10)
    then
        $s.setAge(25);
        update($s);
        System.out.println("规则 rule_student_age 小于 20 岁 同时 大于 10 岁 触发");
    end

rule "rule_student_age 大于 20 岁"
    when
        $s:Student(age > 20)
    then
        System.out.println("规则 rule_student_age 大于 20 岁 触发");
    end
end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
    kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

Student student = new Student();
student.setAge(10);

//将数据提供给规则引擎，规则引擎会根据提供的数据进行规则匹配，如果规则匹配成功则
//执行规则
kieSession.insert(student);

kieSession.fireAllRules();
kieSession.dispose();

```

通过控制台输出可以发现，只有第一个规则触发了，因为在第一个规则中将工作内存中的数据删除了导致第二个规则并没有匹配成功。

5. 规则属性

前面我们已经知道了规则体的构成如下：

```
rule "ruleName"  
  attributes  
  when  
    LHS  
  then  
    RHS  
end
```

本章节就是针对规则体的**attributes**属性部分进行讲解。Drools中提供的属性如下表(部分属性)：

| 属性名 | 说明 |
|------------------|-----------------------------|
| salience | 指定规则执行优先级 |
| dialect | 指定规则使用的语言类型，取值为java和mvel |
| enabled | 指定规则是否启用 |
| date-effective | 指定规则生效时间 |
| date-expires | 指定规则失效时间 |
| activation-group | 激活分组，具有相同分组名称的规则只能有一个规则触发 |
| agenda-group | 议程分组，只有获取焦点的组中的规则才有可能触发 |
| timer | 定时器，指定规则触发的时间 |
| auto-focus | 自动获取焦点，一般结合agenda-group一起使用 |
| no-loop | 防止死循环 |

5.1 enabled属性

enabled属性对应的取值为true和false，默认值为true。

用于指定当前规则是否启用，如果设置的值为false则当前规则无论是否匹配成功都不会触发。


```
rule "rule_comparison_notMemberOf"
    //指定当前规则不可用，当前规则无论是否匹配成功都不会执行
    enabled false
    when
        ComparisonOperatorEntity(names not memberOf list)
    then
        System.out.println("规则 rule_comparison_notMemberOf触发");
    end
```

5.2 dialect属性

dialect属性用于指定当前规则使用的语言类型，取值为java和mvel，默认值为java。

注：mvel是一种基于java语法的表达式语言。

mvel像正则表达式一样，有直接支持集合、数组和字符串匹配的操作符。

mvel还提供了用来配置和构造字符串的模板语言。

mvel表达式内容包括属性表达式，布尔表达式，方法调用，变量赋值，函数定义等。

5.3 salience属性

salience属性用于指定规则的执行优先级，取值类型为Integer。数值越大越优先执行。每个规则都有一个默认的执行顺序，如果不设置salience属性，规则体的执行顺序为由上到下。

可以通过创建规则文件salience.drl来测试salience属性，内容如下：

```
package test.salience

rule "rule_1"
    when
        eval(true)
    then
        System.out.println("规则 rule_1触发");
    end

rule "rule_2"
    when
        eval(true)
    then
        System.out.println("规则 rule_2触发");
    end

rule "rule_3"
    when
        eval(true)
    then
        System.out.println("规则 rule_3触发");
    end
```

通过控制台可以看到，由于以上三个规则没有设置salience属性，所以执行的顺序是按照规则文件中规则的顺序由上到下执行的。接下来我们修改一下文件内容：

```
package testsalience

rule "rule_1"
  salience 9
  when
    eval(true)
  then
    System.out.println("规则 rule_1触发");
end

rule "rule_2"
  salience 10
  when
    eval(true)
  then
    System.out.println("规则 rule_2触发");
end

rule "rule_3"
  salience 8
  when
    eval(true)
  then
    System.out.println("规则 rule_3触发");
end
```

通过控制台可以看到，规则文件执行的顺序是按照我们设置的salience值由大到小顺序执行的。

建议在编写规则时使用salience属性明确指定执行优先级。

5.4 no-loop属性

no-loop属性用于防止死循环，当规则通过update之类的函数修改了Fact对象时，可能使当前规则再次被激活从而导致死循环。取值类型为Boolean，默认值为false。测试步骤如下：

第一步：编写规则文件/resource/rules/noloop.drl

```

package testnoloop
import com.itheima.drools.entity.Student
/*
    此规则文件用于测试no-loop属性
*/
rule "rule_noloop"
    when
        // no-loop true
        $student:Student(age == 25)
    then
        update($student); //注意此处执行update会导致当前规则重新被激活
        System.out.println("规则rule_noloop触发");
    end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

Student student = new Student();
student.setAge(25);

//将数据提供给规则引擎，规则引擎会根据提供的数据进行规则匹配，如果规则匹配成功则
//执行规则
kieSession.insert(student);

kieSession.fireAllRules();
kieSession.dispose();

```

通过控制台可以看到，由于我们没有设置no-loop属性的值，所以发生了死循环。接下来设置no-loop的值为true再次测试则不会发生死循环。

5.5 activation-group属性

activation-group属性是指激活分组，取值为String类型。具有相同分组名称的规则只能有一个规则被触发。

第一步：编写规则文件/resources/rules/activationgroup.drl

```

package testactivationgroup
/*
    此规则文件用于测试activation-group属性
*/

rule "rule_activationgroup_1"
    activation-group "mygroup"
    when
    then
        System.out.println("规则rule_activationgroup_1触发");
    end

```

```

end

rule "rule_activationgroup_2"
    activation-group "mygroup"
    when
    then
        System.out.println("规则 rule_activationgroup_2触发");
    end
end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
    kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();
kieSession.fireAllRules();
kieSession.dispose();

```

通过控制台可以发现，上面的两个规则因为属于同一个分组，所以只有一个触发了。同一个分组中的多个规则如果都能够匹配成功，具体哪一个最终能够被触发可以通过salience属性确定。

5.6 agenda-group属性

agenda-group属性为议程分组，属于另一种可控的规则执行方式。用户可以通过设置agenda-group来控制规则的执行，只有获取焦点的组中的规则才会被触发。

第一步：创建规则文件/resources/rules/agendagroup.drl

```

package testagendagroup
/*
    此规则文件用于测试agenda-group属性
*/
rule "rule_agendagroup_1"
    agenda-group "myagendagroup_1"
    when
    then
        System.out.println("规则 rule_agendagroup_1触发");
    end

rule "rule_agendagroup_2"
    agenda-group "myagendagroup_1"
    when
    then
        System.out.println("规则 rule_agendagroup_2触发");
    end

//=====
rule "rule_agendagroup_3"
    agenda-group "myagendagroup_2"
    when
    then
        System.out.println("规则 rule_agendagroup_3触发");
    end

```

```

end

rule "rule_agendagroup_4"
    agenda-group "myagendagroup_2"
    when
    then
        System.out.println("规则 rule_agendagroup_4触发");
    end
end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
    kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

//设置焦点，对应agenda-group分组中的规则才可能被触发
kieSession.getAgenda().getAgendaGroup("myagendagroup_1").setFocus();

kieSession.fireAllRules();
kieSession.dispose();

```

通过控制台可以看到，只有获取焦点的分组中的规则才会触发。与activation-group不同的是，activation-group定义的分组中只能有一个规则可以被触发，而agenda-group分组中的多个规则都可以被触发。

5.7 auto-focus属性

auto-focus属性为自动获取焦点，取值类型为Boolean，默认值为false。一般结合agenda-group属性使用，当一个议程分组未获取焦点时，可以设置auto-focus属性来控制。

第一步：修改/resources/rules/agendagroup.drl文件内容如下

```

package testagendagroup

rule "rule_agendagroup_1"
    agenda-group "myagendagroup_1"
    when
    then
        System.out.println("规则 rule_agendagroup_1触发");
    end

rule "rule_agendagroup_2"
    agenda-group "myagendagroup_1"
    when
    then
        System.out.println("规则 rule_agendagroup_2触发");
    end

//=====
rule "rule_agendagroup_3"
    agenda-group "myagendagroup_2"

```

```

    auto-focus true //自动获取焦点
    when
    then
        System.out.println("规则 rule_agendagroup_3触发");
end

rule "rule_agendagroup_4"
    agenda-group "myagendagroup_2"
    auto-focus true //自动获取焦点
    when
    then
        System.out.println("规则 rule_agendagroup_4触发");
end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();
kieSession.fireAllRules();
kieSession.dispose();

```

通过控制台可以看到，设置auto-focus属性为true的规则都触发了。

5.8 timer属性

timer属性可以通过定时器的方式指定规则执行的时间，使用方式有两种：

方式一： timer (int: ?)

此种方式遵循java.util.Timer对象的使用方式，第一个参数表示几秒后执行，第二个参数表示每隔几秒执行一次，第二个参数为可选。

方式二： timer(cron:)

此种方式使用标准的unix cron表达式的使用方式来定义规则执行的时间。

第一步：创建规则文件/resources/rules/timer.drl

```

package testtimer
import java.text.SimpleDateFormat
import java.util.Date
/*
    此规则文件用于测试timer属性
*/

rule "rule_timer_1"
    timer (5s 2s) //含义：5秒后触发，然后每隔2秒触发一次
    when
    then
        System.out.println("规则 rule_timer_1触发，触发时间为： " +

```

```

        new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date()));
end

rule "rule_timer_2"
    timer (cron:0/1 * * * * ?) //含义：每隔1秒触发一次
    when
    then
        System.out.println("规则rule_timer_2触发，触发时间为：" +
            new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(new Date()));
end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
final KieSession kieSession = kieClasspathContainer.newKieSession();

new Thread(new Runnable() {
    public void run() {
        //启动规则引擎进行规则匹配，直到调用halt方法才结束规则引擎
        kieSession.fireUntilHalt();
    }
}).start();

Thread.sleep(10000);
//结束规则引擎
kieSession.halt();
kieSession.dispose();

```

注意：单元测试的代码和以前的有所不同，因为我们规则文件中使用到了timer进行定时执行，需要程序能够持续一段时间才能够看到定时器触发的效果。

5.9 date-effective属性

date-effective属性用于指定规则的生效时间，即只有当前系统时间大于等于设置的时间或者日期规则才有可能触发。默认日期格式为：dd-MMM-yyyy。用户也可以自定义日期格式。

第一步：编写规则文件/resources/rules/dateeffective.drl

```

package testdateeffective
/*
    此规则文件用于测试date-effective属性
*/
rule "rule_dateeffective_1"
    date-effective "2020-10-01 10:00"
    when
    then
        System.out.println("规则rule_dateeffective_1触发");
    end

```

第二步：编写单元测试

```

//设置日期格式
System.setProperty("drools.dateformat","yyyy-MM-dd HH:mm");
KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
    kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();
kieSession.fireAllRules();
kieSession.dispose();

```

注意：上面的代码需要设置日期格式，否则我们在规则文件中写的日期格式和默认日期格式不匹配程序会报错。

5.10 date-expires属性

date-expires属性用于指定规则的失效时间，即只有当前系统时间小于设置的时间或者日期规则才有可能触发。默认日期格式为：dd-MMM-yyyy。用户也可以自定义日期格式。

第一步：编写规则文件/resource/rules/dateexpires.drl

```

package testdateexpires
/*
    此规则文件用于测试date-expires属性
*/
rule "rule_dateexpires_1"
    date-expires "2019-10-01 10:00"
    when
    then
        System.out.println("规则rule_dateexpires_1触发");
    end

```

第二步：编写单元测试


```
//设置日期格式
System.setProperty("drools.dateformat", "yyyy-MM-dd HH:mm");
KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();
kieSession.fireAllRules();
kieSession.dispose();
```

注意：上面的代码需要设置日期格式，否则我们在规则文件中写的日期格式和默认的时间格式不匹配程序会报错。

6. Drools高级语法

前面章节我们已经知道了一套完整的规则文件内容构成如下：

| 关键字 | 描述 |
|----------|----------------------------------|
| package | 包名，只限于逻辑上的管理，同一个包名下的查询或者函数可以直接调用 |
| import | 用于导入类或者静态方法 |
| global | 全局变量 |
| function | 自定义函数 |
| query | 查询 |
| rule end | 规则体 |

本章节我们就来学习其中的几个关键字。

6.1 global全局变量

global关键字用于在规则文件中定义全局变量，它可以让应用程序的对象在规则文件中能够被访问。可以用来为规则文件提供数据或服务。

语法结构为：**global 对象类型 对象名称**

在使用global定义的全局变量时有两点需要注意：

- 1、如果对象类型为包装类型时，在一个规则中改变了global的值，那么只针对当前规则有效，对其他规则中的global不会有影响。可以理解为它是当前规则代码中的global副本，规则内部修改不会影响全局的使用。
- 2、如果对象类型为集合类型或JavaBean时，在一个规则中改变了global的值，对java代码和所有规则都有效。

下面我们通过代码进行验证：

第一步：创建UserService类

```
package com.itheima.drools.service;

public class UserService {
    public void save(){
        System.out.println("UserService.save()...");
    }
}
```

第二步：编写规则文件/resources/rules/global.drl

```
package testglobal
/*
    此规则文件用于测试global全局变量
*/

global java.lang.Integer count //定义一个包装类型的全局变量
global com.itheima.drools.service.UserService userService //定义一个
JavaBean类型的全局变量
global java.util.List gList //定义一个集合类型的全局变量

rule "rule_global_1"
    when
    then
        count += 10; //全局变量计算，只对当前规则有效，其他规则不受影响
        userService.save(); //调用全局变量的方法
        gList.add("itcast"); //向集合类型的全局变量中添加元素，Java代码和所有规
则都受影响
        gList.add("itheima");
        System.out.println("count=" + count);
        System.out.println("gList.size=" + gList.size());
    end

rule "rule_global_2"
    when
    then
        userService.save();
        System.out.println("count=" + count);
        System.out.println("gList.size=" + gList.size());
    end
```

第三步：编写单元测试

```
KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

//设置全局变量，名称和类型必须和规则文件中定义的全局变量名称对应
kieSession.setGlobal("userService", new UserService());
kieSession.setGlobal("count", 5);
List list = new ArrayList(); //size为0
```

```
kieSession.setGlobal("gLst",list);

kieSession.fireAllRules();
kieSession.dispose();

//因为在规则中为全局变量添加了两个元素，所以现在的size为2
System.out.println(list.size());
```

6.2 query查询

query查询提供了一种**查询working memory中符合约束条件的Fact对象**的简单方法。它仅包含规则文件中的LHS部分，不用指定“when”和“then”部分并且以end结束。具体语法结构如下：

```
query 查询的名称(可选参数)
    LHS
end
```

具体操作步骤：

第一步：编写规则文件/resources/rules/query.drl

```
package testquery
import com.itheima.drools.entity.Student
/*
    此规则文件用于测试query查询
*/

//不带参数的查询
//当前query用于查询working Memory中age>10的Student对象
query "query_1"
    $student:Student(age > 10)
end

//带有参数的查询
//当前query用于查询working Memory中age>10同时name需要和传递的参数name相同的Student对象
query "query_2"(String sname)
    $student:Student(age > 20 && name == sname)
end
```

第二步：编写单元测试

```
KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

Student student1 = new Student();
student1.setName("张三");
student1.setAge(12);
```

```

Student student2 = new Student();
student2.setName("李四");
student2.setAge(8);

Student student3 = new Student();
student3.setName("王五");
student3.setAge(22);

//将对象插入working Memory中
kieSession.insert(student1);
kieSession.insert(student2);
kieSession.insert(student3);

//调用规则文件中的查询
QueryResults results1 = kieSession.getQueryResults("query_1");
int size = results1.size();
System.out.println("size=" + size);
for (QueryResultsRow row : results1) {
    Student student = (Student) row.get("$student");
    System.out.println(student);
}

//调用规则文件中的查询
QueryResults results2 = kieSession.getQueryResults("query_2", "王五");
size = results2.size();
System.out.println("size=" + size);
for (QueryResultsRow row : results2) {
    Student student = (Student) row.get("$student");
    System.out.println(student);
}
//kieSession.fireAllRules();
kieSession.dispose();

```

6.3 function函数

function关键字用于在规则文件中定义函数，就相当于java类中的方法一样。可以在规则体中调用定义的函数。使用函数的好处是可以将业务逻辑集中放置在一个地方，根据需要可以对函数进行修改。

函数定义的语法结构如下：

```

function 返回值类型 函数名(可选参数){
    //逻辑代码
}

```

具体操作步骤：

第一步：编写规则文件/resources/rules/function.drl

```

package testfunction
import com.itheima.drools.entity.Student

```

```

/*
    此规则文件用于测试function函数
*/

//定义一个函数
function String sayHello(String name){
    return "hello " + name;
}

rule "rule_function_1"
    when
        $student:Student(name != null)
    then
        //调用上面定义的函数
        String ret = sayHello($student.getName());
        System.out.println(ret);
    end

```

第二步：编写单元测试

```

KieServices kieServices = KieServices.Factory.get();
KieContainer kieClasspathContainer =
kieServices.getKieClasspathContainer();
KieSession kieSession = kieClasspathContainer.newKieSession();

Student student = new Student();
student.setName("小明");

kieSession.insert(student);

kieSession.fireAllRules();
kieSession.dispose();

```

6.4 LHS加强

前面我们已经知道了在规则体中的LHS部分是介于when和then之间的部分，主要用于模式匹配，只有匹配结果为true时，才会触发RHS部分的执行。本章节我们会针对LHS部分学习几个新的用法。

6.4.1 复合值限制in/not in

复合值限制是指超过一种匹配值的限制条件，类似于SQL语句中的in关键字。Drools规则体中的LHS部分可以使用in或者not in进行复合值的匹配。具体语法结构如下：

Object(field in (比较值1,比较值2...))

举例：

```

$s:Student(name in ("张三","李四","王五"))
$s:Student(name not in ("张三","李四","王五"))

```

6.4.2 条件元素eval

eval用于规则体的LHS部分，并返回一个Boolean类型的值。语法结构如下：

eval(表达式)

举例：

```
eval(true)
eval(false)
eval(1 == 1)
```

6.4.3 条件元素not

not用于判断Working Memory中是否存在某个Fact对象，如果不存在则返回true，如果存在则返回false。语法结构如下：

not Object(可选属性约束)

举例：

```
not Student()
not Student(age < 10)
```

6.4.4 条件元素exists

exists的作用与not相反，用于判断Working Memory中是否存在某个Fact对象，如果存在则返回true，不存在则返回false。语法结构如下：

exists Object(可选属性约束)

举例：

```
exists Student()
exists Student(age < 10 && name != null)
```

可能有人会有疑问，我们前面在LHS部分进行条件编写时并没有使用exists也可以达到判断Working Memory中是否存在某个符合条件的Fact元素的目的，那么我们使用exists还有什么意义？

两者的区别：当向Working Memory中加入多个满足条件的Fact对象时，使用了exists的规则执行一次，不使用exists的规则会执行多次。

例如：

规则文件(只有规则体)：

```

rule "使用exists的规则"
when
    exists Student()
then
    System.out.println("规则：使用exists的规则触发");
end

rule "没有使用exists的规则"
when
    Student()
then
    System.out.println("规则：没有使用exists的规则触发");
end

```

Java代码：

```

kieSession.insert(new Student());
kieSession.insert(new Student());
kieSession.fireAllRules();

```

上面第一个规则只会执行一次，因为Working Memory中存在两个满足条件的Fact对象，第二个规则会执行两次。

6.4.5 规则继承

规则之间可以使用extends关键字进行规则条件部分的继承，类似于java类之间的继承。

例如：

```

rule "rule_1"
when
    Student(age > 10)
then
    System.out.println("规则：rule_1触发");
end

rule "rule_2" extends "rule_1" //继承上面的规则
when
    /*
    此处的条件虽然只写了一个，但是从上面的规则继承了一个条件，
    所以当前规则存在两个条件，即Student(age < 20)和Student(age > 10)
    */
    Student(age < 20)
then
    System.out.println("规则：rule_2触发");
end

```

6.5 RHS加强

RHS部分是规则体的重要组成部分，当LHS部分的条件匹配成功后，对应的RHS部分就会触发执行。一般在RHS部分中需要进行业务处理。

在RHS部分Drools为我们提供了一个内置对象，名称就是drools。本小节我们来介绍几个drools对象提供的方法。

6.5.1 halt

halt方法的作用是立即终止后面所有规则的执行。

```
package testhalt
rule "rule_halt_1"
    when
    then
        System.out.println("规则：rule_halt_1触发");
        drools.halt();//立即终止后面所有规则执行
    end

//当前规则并不会触发，因为上面的规则调用了halt方法导致后面所有规则都不会执行
rule "rule_halt_2"
    when
    then
        System.out.println("规则：rule_halt_2触发");
    end
```

6.5.2 getWorkingMemory

getWorkingMemory方法的作用是返回工作内存对象。

```
package testgetWorkingMemory
rule "rule_getWorkingMemory"
    when
    then
        System.out.println(drools.getWorkingMemory());
    end
```

6.5.3 getRule

getRule方法的作用是返回规则对象。

```
package testgetRule
rule "rule_getRule"
    when
    then
        System.out.println(drools.getRule());
    end
```

6.6 规则文件编码规范

我们在进行drl类型的规则文件编写时尽量遵循如下规范：

- 所有的规则文件(.drl)应统一放在一个规定的文件夹中，如：/rules文件夹
- 书写的每个规则应尽量加上注释。注释要清晰明了，言简意赅

- 同一类型的对象尽量放在一个规则文件中，如所有Student类型的对象尽量放在一个规则文件中
- 规则结果部分(RHS)尽量不要有条件语句，如if(...)，尽量不要有复杂的逻辑和深层次的嵌套语句
- 每个规则最好都加上salience属性，明确执行顺序
- Drools默认dialect为"java"，尽量避免使用dialect "mvel"

7. Spring整合Drools

7.1 Spring简单整合Drools

在项目中使用Drools时往往会跟Spring整合来使用。具体整合步骤如下：

第一步：创建maven工程drools_spring并配置pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.itheima</groupId>
    <artifactId>drools_spring</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <drools.version>7.10.0.Final</drools.version>
        <spring.version>5.0.5.RELEASE</spring.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.drools</groupId>
            <artifactId>drools-compiler</artifactId>
            <version>${drools.version}</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
        </dependency>
        <dependency>
            <groupId>org.kie</groupId>
            <artifactId>kie-spring</artifactId>
            <version>${drools.version}</version>
            <!--注意：此处必须排除传递过来的依赖，否则会跟我们自己导入的Spring
jar包产生冲突-->
            <exclusions>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-tx</artifactId>
```

```

        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${spring.version}</version>
</dependency>
</dependencies>
</project>

```

第二步：创建规则目录/resources/rules，中rules目录中创建规则文件helloworld.drl

```

package helloworld

rule "rule_helloworld"
when
    eval(true)
then
    System.out.println("规则： rule_helloworld触发...");
end

```

第三步：创建Spring配置文件/resources/spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:kie="http://drools.org/schema/kie-spring"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://drools.org/schema/kie-spring
                           http://drools.org/schema/kie-spring.xsd">
    <kie:kmodule id="kmodule">
        <kie:kbase name="kbase" packages="rules">
            <kie:ksession name="ksession"></kie:ksession>
        </kie:kbase>
    </kie:kmodule>
    <bean
class="org.kie.spring.annotations.KModuleAnnotationPostProcessor"></bean>
</beans>
```

第四步：编写单元测试类

```
package com.itheima.test;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.kie.api.KieBase;
import org.kie.api.cdi.KBase;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = "classpath:spring.xml")
public class DroolsSpringTest {
    @KBase("kbase")
    private KieBase kieBase; //注入KieBase对象
    @Test
    public void test1(){
        KieSession kieSession = kieBase.newKieSession();
        kieSession.fireAllRules();
        kieSession.dispose();
    }
}
```

7.2 Spring整合Drools+web

本小节我们来进行Drools和Spring Web的整合。具体操作步骤如下：

第一步：创建maven的war工程drools_springweb并在pom.xml文件中导入相关maven坐标

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.itheima</groupId>
<artifactId>drools_springweb</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>war</packaging>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <drools.version>7.10.0.Final</drools.version>
  <spring.version>5.0.5.RELEASE</spring.version>
</properties>
<dependencies>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>${drools.version}</version>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-spring</artifactId>
    <version>${drools.version}</version>
    <!--注意：此处必须排除传递过来的依赖，否则会跟我们自己导入的Spring jar包
产生冲突-->
    <exclusions>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-beans</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
      </exclusion>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>

```

```

        <artifactId>spring-context</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context-support</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-test</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-web</artifactId>
        <version>${spring.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>${spring.version}</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <configuration>
                <!-- 指定端口 -->
                <port>80</port>
                <!-- 请求路径 -->
                <path>/</path>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

第二步：配置web.xml

```

<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd" >
<web-app>

```

```

<display-name>Archetype Created Web Application</display-name>
<servlet>
  <servlet-name>springmvc</servlet-name>
  <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <!-- 指定加载的配置文件 ， 通过参数contextConfigLocation加载 -->
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:springmvc.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>springmvc</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
</web-app>

```

第三步：创建/resources/springmvc.xml文件

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:kie="http://drools.org/schema/kie-spring"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://drools.org/schema/kie-spring
    http://drools.org/schema/kie-spring.xsd
    http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">

  <kie:kmodule id="kmodule">
    <kie:kbase name="kbase" packages="rules">
      <kie:ksession name="ksession"></kie:ksession>
    </kie:kbase>
  </kie:kmodule>

  <bean
class="org.kie.spring.annotations.KModuleAnnotationPostProcessor"/>

  <!--spring批量扫描-->
  <context:component-scan base-package="com.itheima" />
  <context:annotation-config/>
  <!--springMVC注解驱动-->
  <mvc:annotation-driven/>
</beans>

```

第四步：创建规则文件/resources/rules/helloworld.drl

```
package helloworld

rule "rule_helloworld"
    when
        eval(true)
    then
        System.out.println("规则： rule_helloworld触发...");
    end
```

第五步：创建RuleService

```
package com.itheima.service;

import org.kie.api.KieBase;
import org.kie.api.cdi.KBase;
import org.kie.api.runtime.KieSession;
import org.springframework.stereotype.Service;

@Service
public class RuleService {
    @KBase("kbase")
    private KieBase kieBase;
    public void rule(){
        KieSession kieSession = kieBase.newKieSession();
        kieSession.fireAllRules();
        kieSession.dispose();
    }
}
```

第六步：创建HelloController

```
package com.itheima.controller;

import com.itheima.service.RuleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/hello")
public class HelloController {
    @Autowired
    private RuleService ruleService;
    @RequestMapping("/rule")
    public String rule(){
        ruleService.rule();
        return "OK";
    }
}
```

7.3 Spring Boot整合Drools

目前在企业开发中Spring Boot已经成为主流，本小节我们来进行Spring Boot整合Drools。具体操作步骤：

第一步：创建maven工程drools_springboot并配置pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starters</artifactId>
        <version>2.0.6.RELEASE</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.itheima</groupId>
    <artifactId>drools_springboot</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-aop</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
        </dependency>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
            <version>2.6</version>
        </dependency>
        <!--drools规则引擎-->
        <dependency>
            <groupId>org.drools</groupId>
            <artifactId>drools-core</artifactId>
            <version>7.6.0.Final</version>
        </dependency>
        <dependency>
            <groupId>org.drools</groupId>
            <artifactId>drools-compiler</artifactId>
            <version>7.6.0.Final</version>
        </dependency>
        <dependency>
```



```

        <groupId>org.drools</groupId>
        <artifactId>drools-templates</artifactId>
        <version>7.6.0.Final</version>
    </dependency>
    <dependency>
        <groupId>org.kie</groupId>
        <artifactId>kie-api</artifactId>
        <version>7.6.0.Final</version>
    </dependency>
    <dependency>
        <groupId>org.kie</groupId>
        <artifactId>kie-spring</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-tx</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-beans</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-core</artifactId>
            </exclusion>
            <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-context</artifactId>
            </exclusion>
        </exclusions>
        <version>7.6.0.Final</version>
    </dependency>
</dependencies>
<build>
    <finalName>${project.artifactId}</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.xml</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.xml</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
    </resources>
    <plugins>

```

```

        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

第二步：创建/resources/application.yml文件

```

server:
  port: 8080
spring:
  application:
    name: drools_springboot

```

第三步：创建规则文件/resources/rules/helloworld.drl

```

package helloworld
rule "rule_helloworld"
when
    eval(true)
then
    System.out.println("规则： rule_helloworld触发...");
end

```

第四步：编写配置类DroolsConfig

```

package com.itheima.drools.config;
import org.kie.api.KieBase;
import org.kie.api.KieServices;
import org.kie.api.builder.KieBuilder;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieRepository;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.internal.io.ResourceFactory;
import org.kie.spring.KModuleBeanFactoryPostProcessor;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.core.io.support.PathMatchingResourcePatternResolver;
import org.springframework.core.io.support.ResourcePatternResolver;
import org.springframework.core.io.Resource;

```

```

import java.io.IOException;
/**
 * 规则引擎配置类
 */
@Configuration
public class DroolsConfig {
    //指定规则文件存放的目录
    private static final String RULES_PATH = "rules/";
    private final KieServices kieServices = KieServices.Factory.get();
    @Bean
    @ConditionalOnMissingBean
    public KieFileSystem kieFileSystem() throws IOException {
        KieFileSystem kieFileSystem = kieServices.newKieFileSystem();
        ResourcePatternResolver resourcePatternResolver =
            new PathMatchingResourcePatternResolver();
        Resource[] files =
            resourcePatternResolver.getResources("classpath*:" +
RULES_PATH + "/*.*d");
        String path = null;
        for (Resource file : files) {
            path = RULES_PATH + file.getFilename();
            kieFileSystem.write(ResourceFactory.newClassPathResource(path,
"UTF-8"));
        }
        return kieFileSystem;
    }
    @Bean
    @ConditionalOnMissingBean
    public KieContainer kieContainer() throws IOException {
        KieRepository kieRepository = kieServices.getRepository();
        kieRepository.addKieModule(kieRepository.getDefaultReleaseId());
        KieBuilder kieBuilder =
kieServices.newKieBuilder(kieFileSystem());
        kieBuilder.buildAll();
        return
kieServices.newKieContainer(kieRepository.getDefaultReleaseId());
    }
    @Bean
    @ConditionalOnMissingBean
    public KieBase kieBase() throws IOException {
        return kieContainer().getKieBase();
    }
    @Bean
    @ConditionalOnMissingBean
    public KModuleBeanFactoryPostProcessor kiePostProcessor() {
        return new KModuleBeanFactoryPostProcessor();
    }
}

```

第五步：创建RuleService类

```

package com.itheima.drools.service;

import org.kie.api.KieBase;
import org.kie.api.runtime.KieSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class RuleService {
    @Autowired
    private KieBase kieBase;
    public void rule(){
        KieSession kieSession = kieBase.newKieSession();
        kieSession.fireAllRules();
        kieSession.dispose();
    }
}

```

第六步：创建HelloController类

```

package com.itheima.drools.controller;

import com.itheima.drools.service.RuleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/hello")
public class HelloController {
    @Autowired
    private RuleService ruleService;
    @RequestMapping("/rule")
    public String rule(){
        ruleService.rule();
        return "OK";
    }
}

```

第七步：创建启动类DroolsApplication

```
package com.itheima.drools;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DroolsApplication {
    public static void main(String[] args) {
        SpringApplication.run(DroolsApplication.class, args);
    }
}
```

第八步：启动服务，访问<http://localhost:8080/hello/rule>

8. WorkBench

8.1 WorkBench简介

WorkBench是KIE组件中的元素，也称为KIE-WB，是Drools-WB与JBPM-WB的结合体。它是一个可视化的规则编辑器。WorkBench其实就是一个war包，安装到tomcat中就可以运行。使用WorkBench可以在浏览器中创建数据对象、创建规则文件、创建测试场景并将规则部署到maven仓库供其他应用使用。

下载地址：<https://download.jboss.org/drools/release/7.6.0.Final/kie-drools-wb-7.6.0.Final-tomcat8.war>

注意：下载的war包需要安装到tomcat8中。

8.2 安装方式

软件安装时经常会涉及到软件版本兼容性的问题，所以需要明确各个软件的使用版本。

本课程使用的软件环境如下：

- 操作系统：Windows 10 64位
- JDK版本：1.8
- maven版本：3.5.4
- Tomcat版本：8.5

具体安装步骤：

第一步：配置Tomcat的环境变量CATALINA_HOME，对应的值为Tomcat安装目录

第二步：在Tomcat的bin目录下创建setenv.bat文件，内容如下：

```
CATALINA_OPTS="-Xmx512M \  
-Djava.security.auth.login.config=$CATALINA_HOME/webapps/kie-drools-  
wb/WEB-INF/classes/login.config \  
-Dorg.jboss.logging.provider=jdk"
```

第三步：将下载的WorkBench的war包改名为kie-drools-wb.war并复制到Tomcat的webapps目录下

第四步：修改Tomcat下conf/tomcat-users.xml文件

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users xmlns="http://tomcat.apache.org/xml"
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:schemaLocation="http://tomcat.apache.org/xml tomcat-
users.xsd"
               version="1.0">
  <!--定义admin角色-->
  <role rolename="admin"/>
  <!--定义一个用户，用户名为kie，密码为kie，对应的角色为admin角色-->
  <user username="kie" password="kie" roles="admin"/>
</tomcat-users>
```

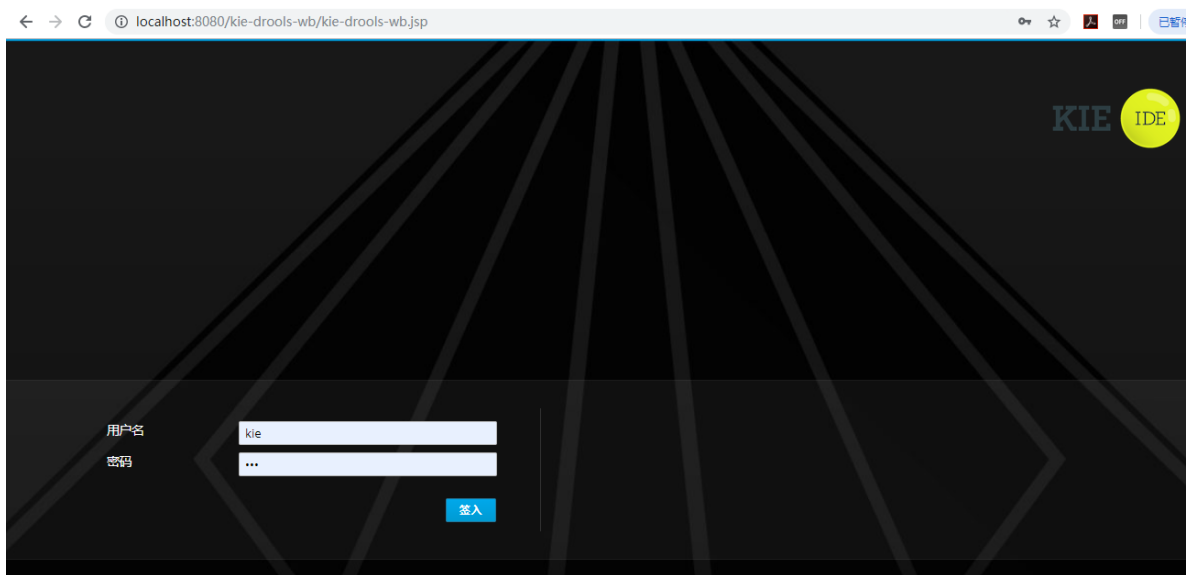
第五步：下载以下三个jar包并复制到Tomcat的lib目录下

```
kie-tomcat-integration-7.10.0.Final.jar
javax.security.jacc-api-1.5.jar
slf4j-api-1.7.25.jar
```

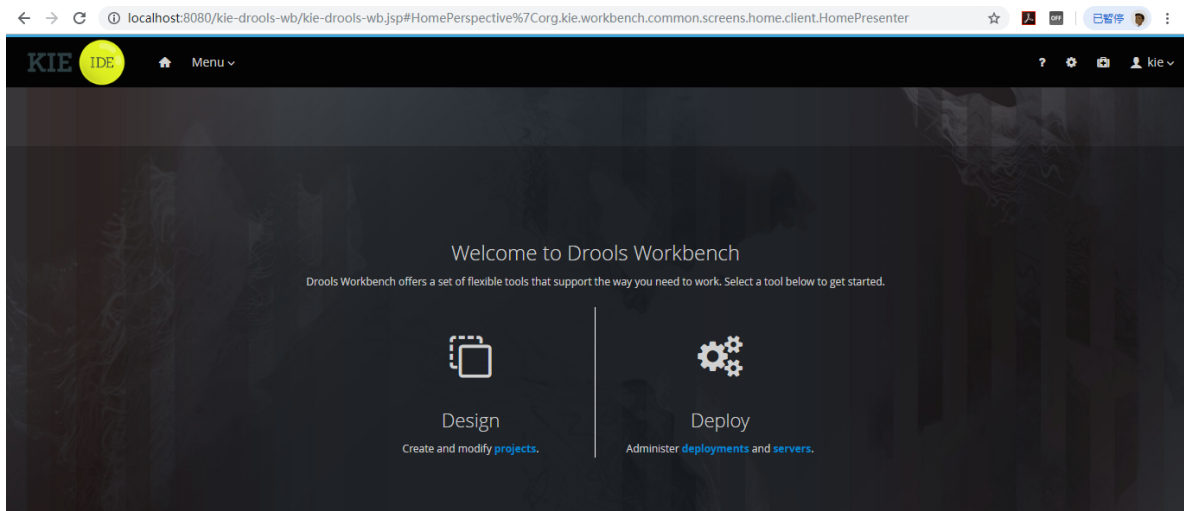
第六步：修改Tomcat的conf/server.xml文件，添加Valve标签，内容为：

```
<valve className="org.kie.integration.tomcat.JACCValve"/>
```

第七步：启动Tomcat并访问<http://localhost:8080/kie-drools-wb>，可以看到WorkBench的登录页面。使用前面在tomcat-users.xml文件中定义的用户进行登录即可



登录成功后进入系统首页：



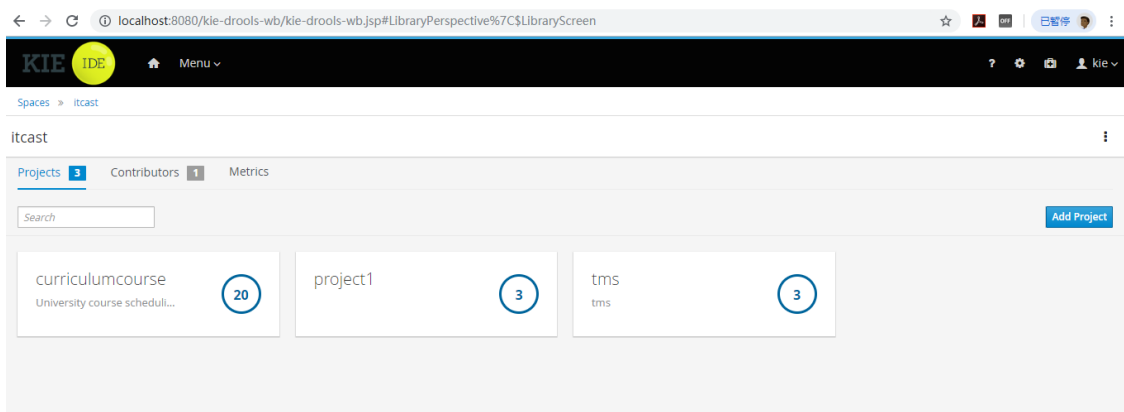
8.3 使用方式

8.3.1 创建空间、项目

WorkBench中存在空间和项目的概念。我们在使用WorkBench时首先需要创建空间（Space），在空间中创建项目，在项目中创建数据对象、规则文件等。

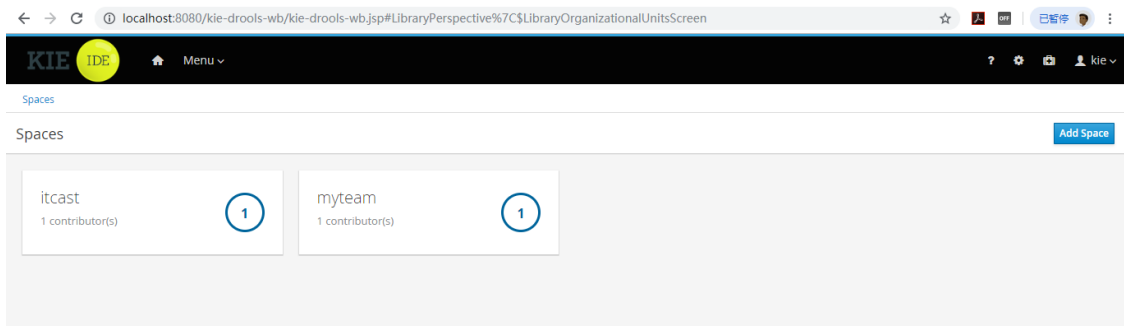
- 创建空间

第一步：登录WorkBench后进行系统首页，点击首页中的Design区域进入项目列表页面：

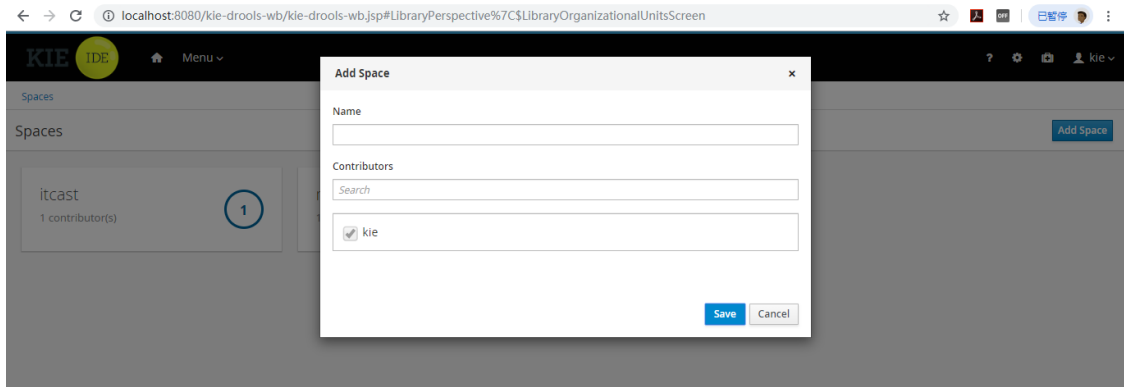


如果是第一次登录还没有创建项目则无法看到项目

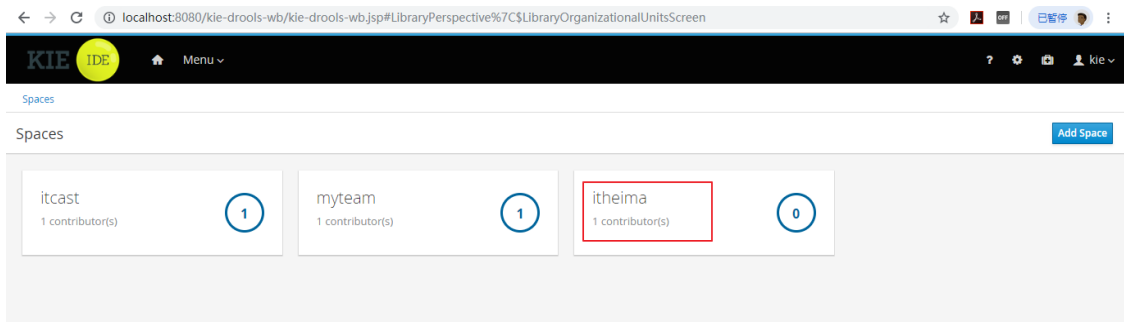
第二步：点击左上角Spaces导航链接进入空间列表页面



第三步：点击右上角Add Space按钮弹出创建添加空间窗口



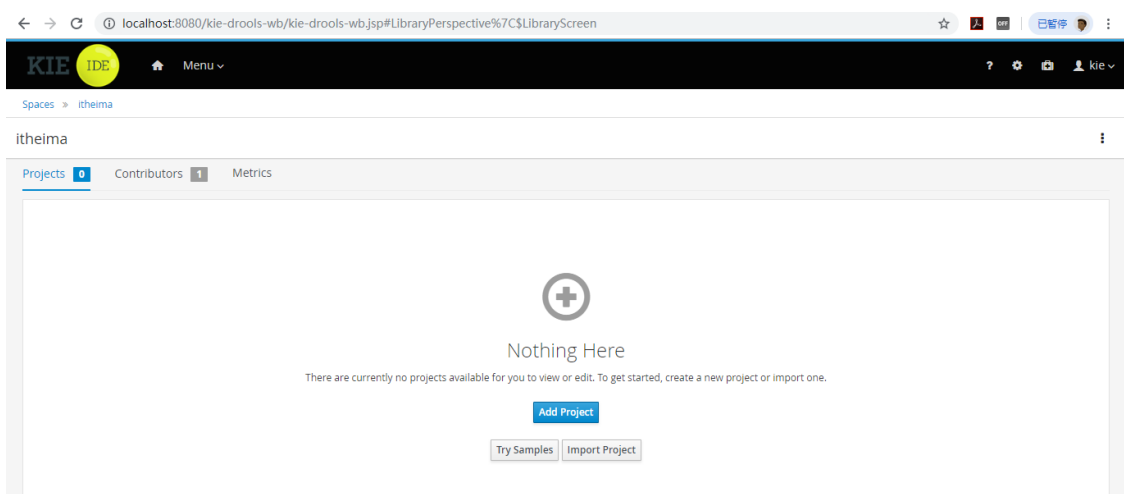
录入空间名称，点击Save按钮则完成空间的创建，如下图：



- 创建项目

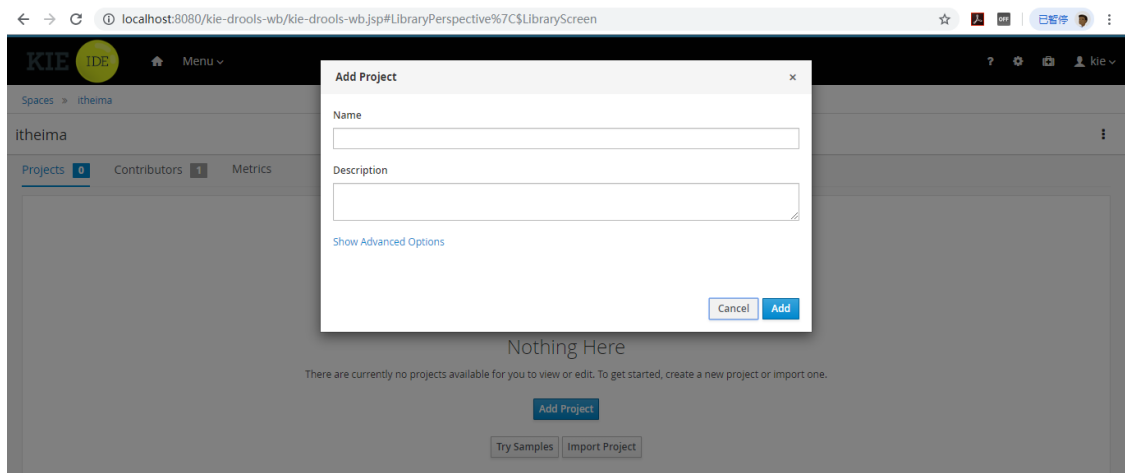
前面已经提到，我们在WorkBench中需要先创建空间，在空间中才能创建项目。上面我们已经创建了一个空间itheima，现在需要在此空间中创建项目。

第一步：点击itheima空间，进入此空间

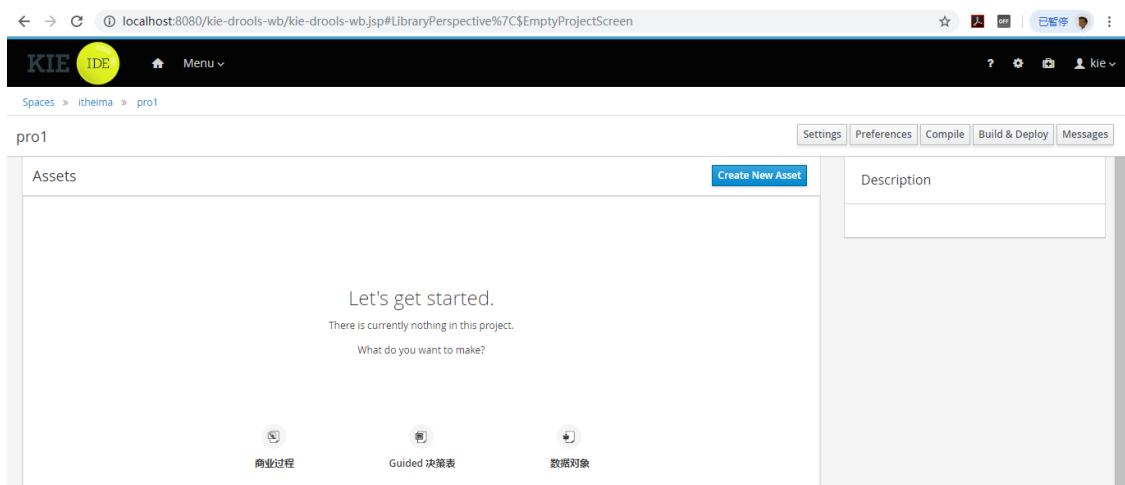


可以看到当前空间中还没有项目

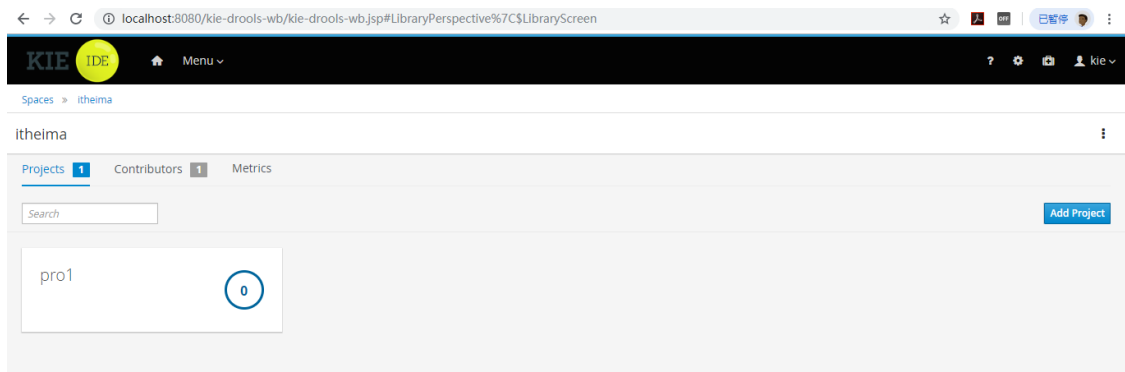
第二步：点击Add Project按钮弹出添加项目窗口



第三步：在添加项目窗口中录入项目名称（例如项目名称为pro1），点击Add按钮完成操作



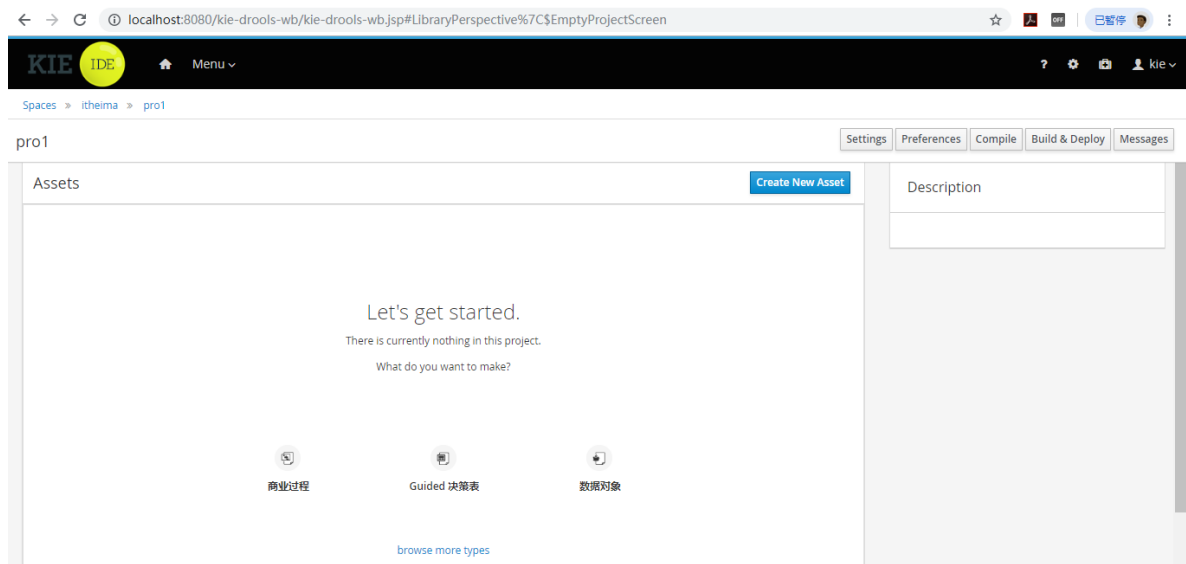
可以看到在完成项目创建后，系统直接跳转到了项目页面。要查看当前itheima空间中的所有项目，可以点击左上角itheima链接：



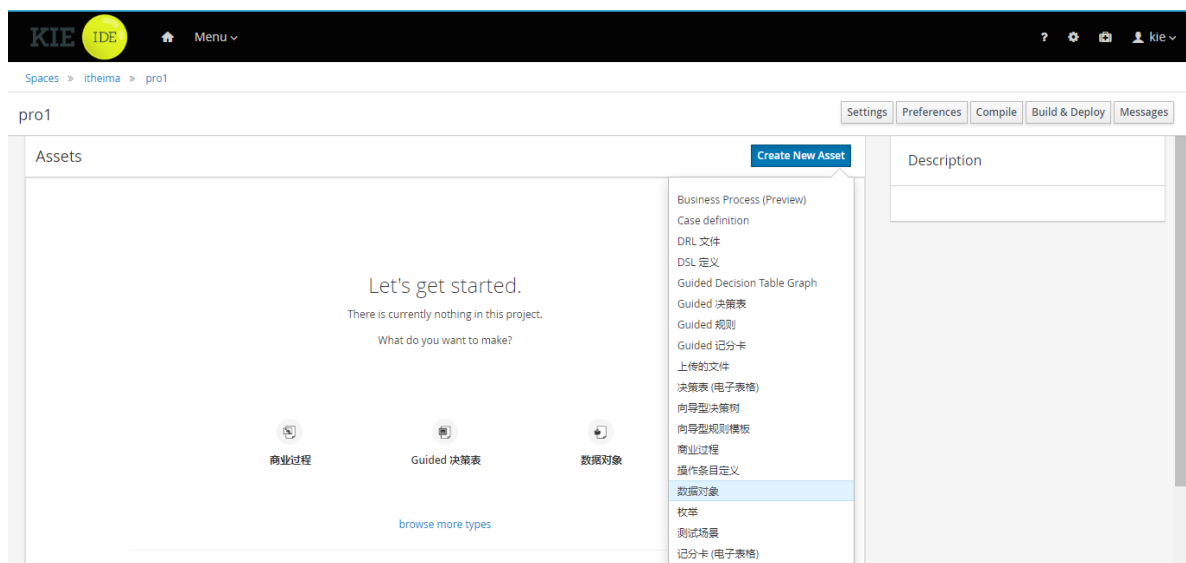
8.3.2 创建数据对象

数据对象其实就是JavaBean，一般都是在drl规则文件中使用进行规则匹配。

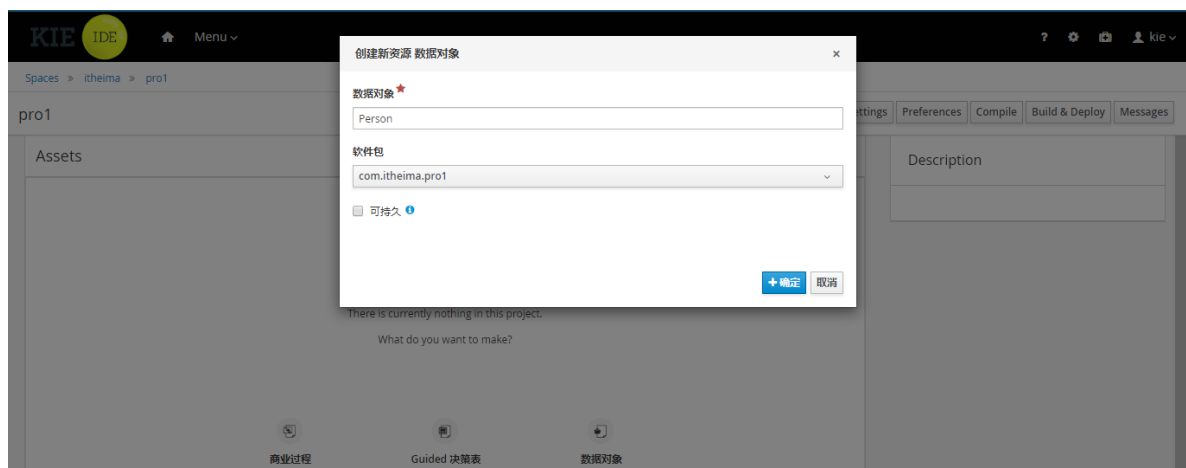
第一步：在itheima空间中点击pro1项目，进入此项目页面



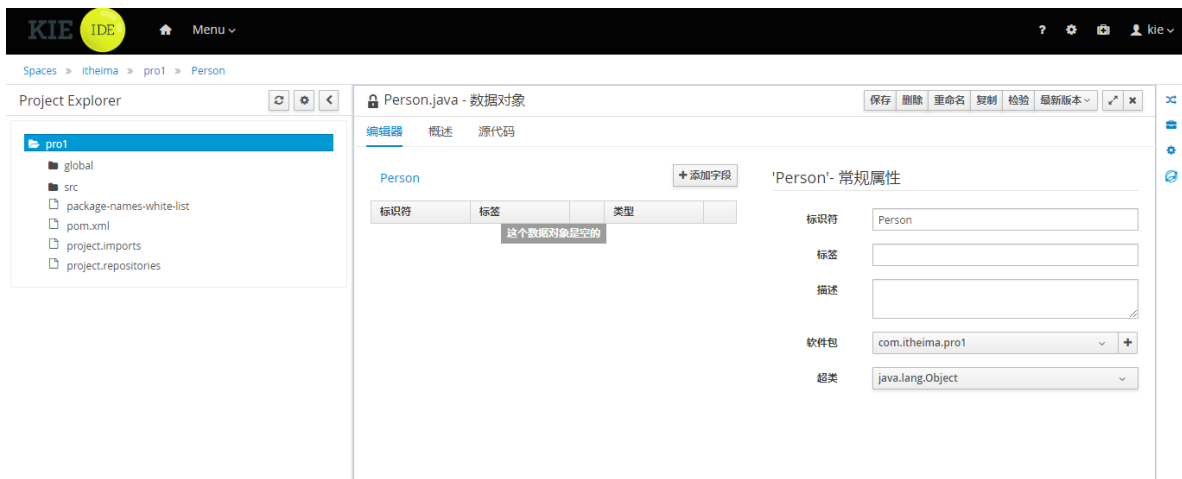
第二步：点击Create New Asset按钮选择“数据对象”



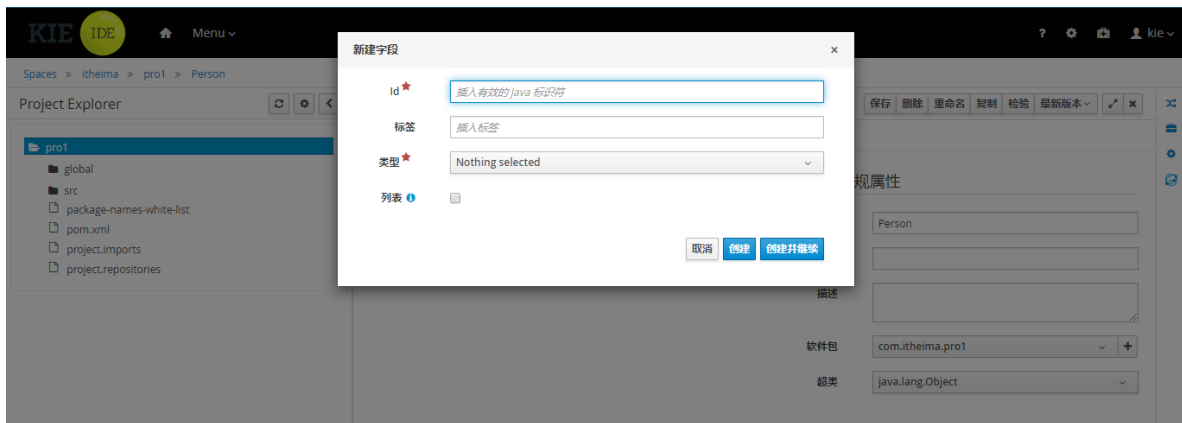
第三步：在弹出的创建数据对象窗口中输入数据对象的名称，点击确定按钮完成操作



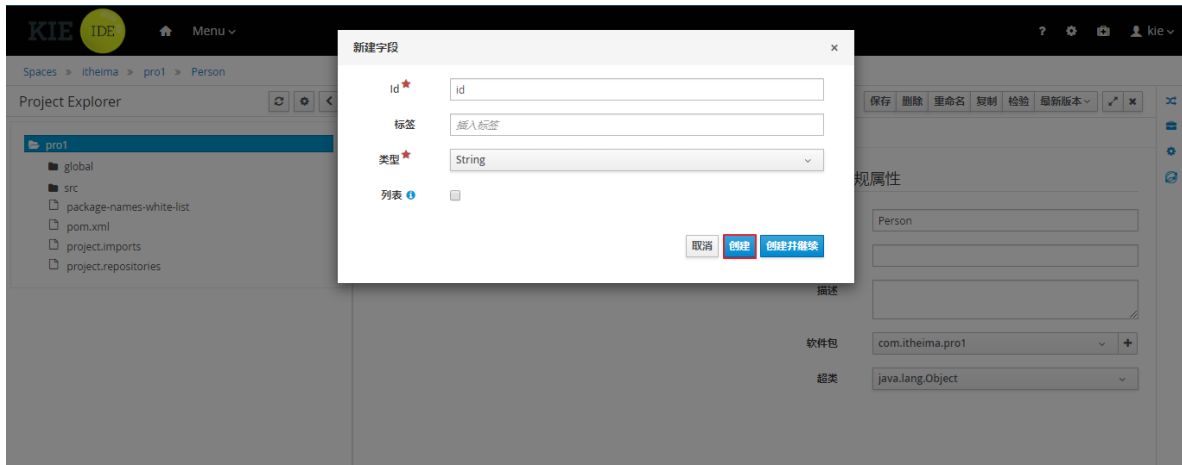
操作完成后可以看到如下：



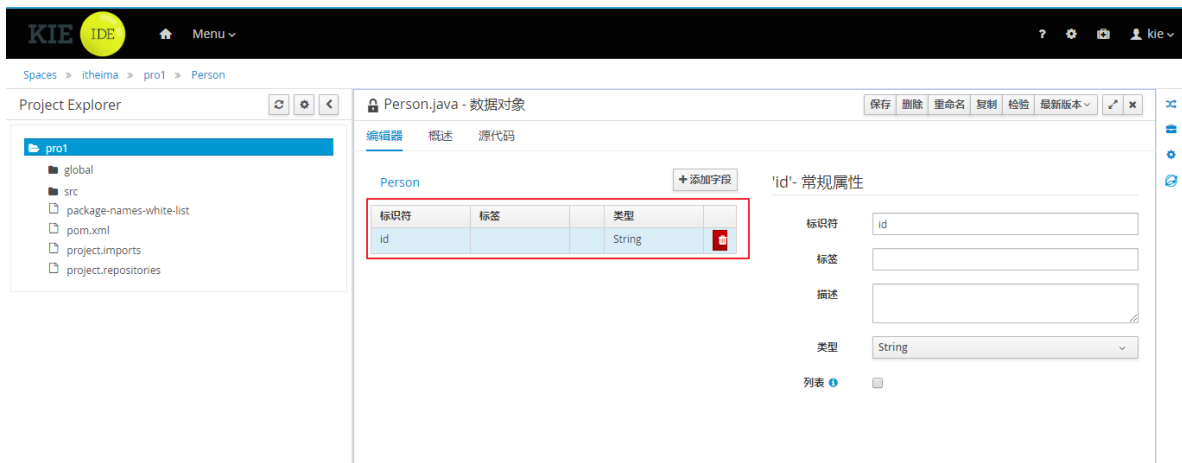
第四步：点击“添加字段”按钮弹出新建字段窗口



第五步：在新建字段窗口中录入字段Id（其实就是属性名），选择类型，点击创建按钮完成操作



完成操作后可以看到刚才创建的字段：



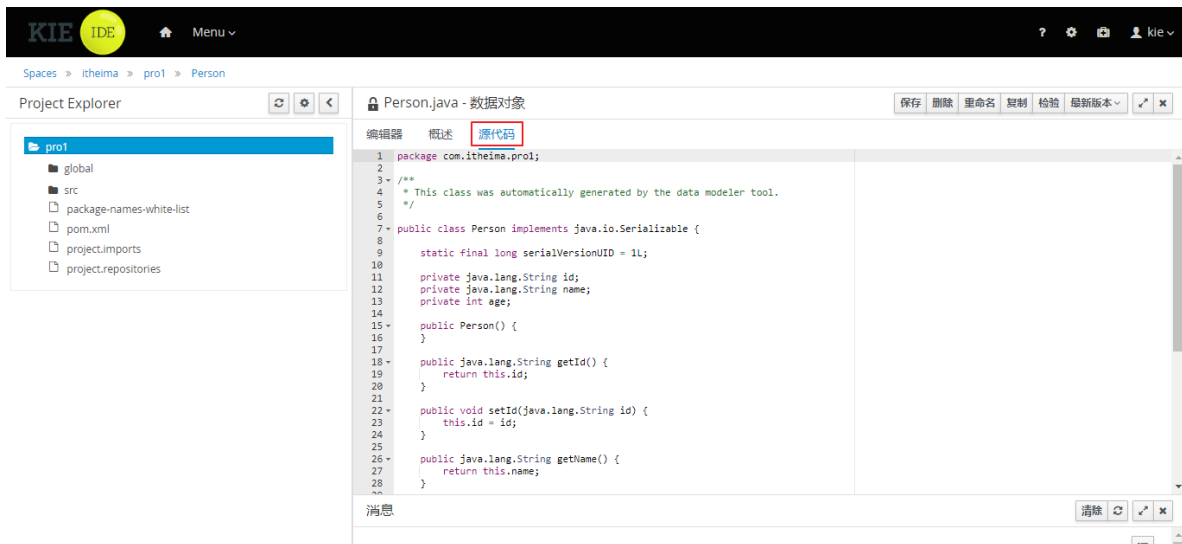
可以点击添加字段按钮继续创建其他字段：



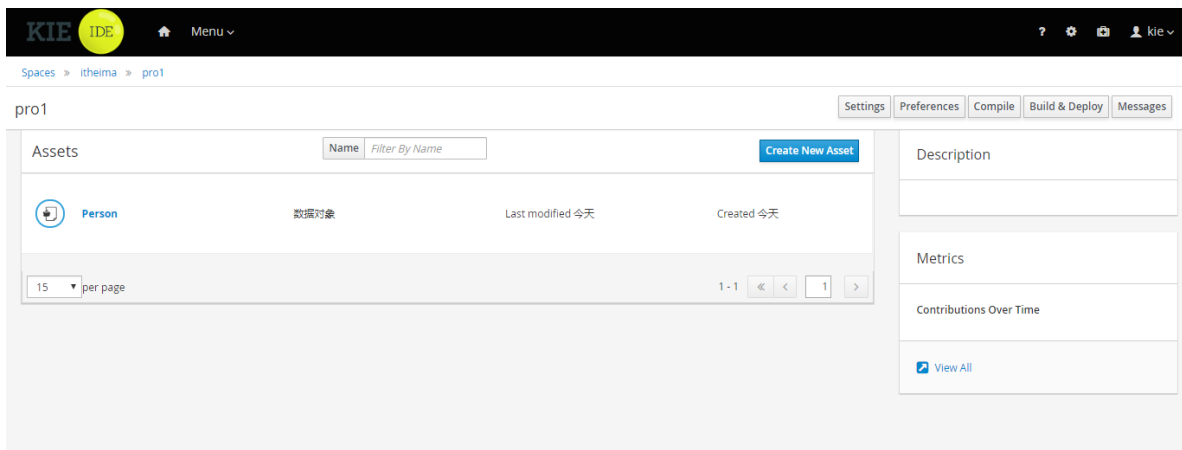
注意添加完字段后需要点击右上角保存按钮完成保存操作：



点击源代码按钮可以查看刚才创建的Person对象源码：

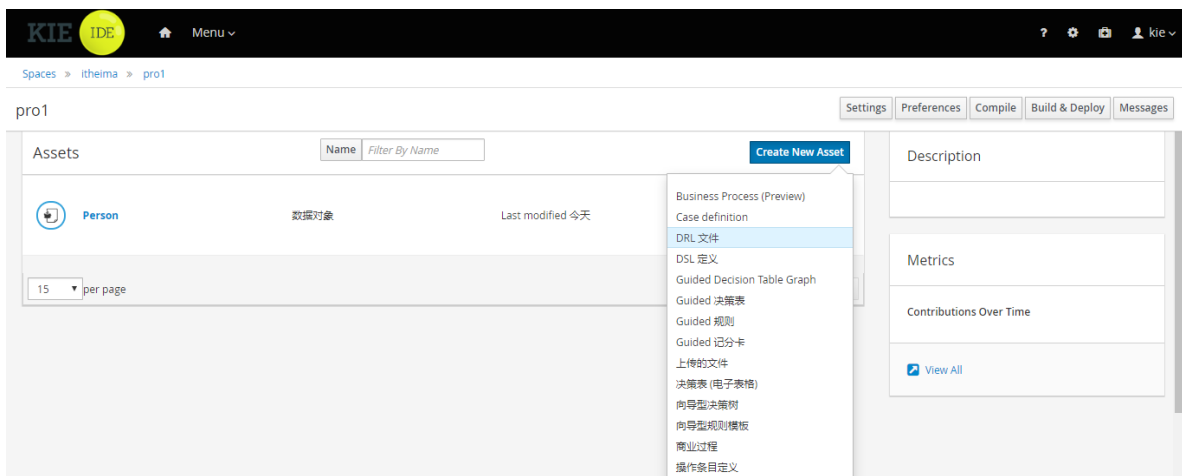


点击左上角pro1项目链接，可以看到当前pro1项目中已经创建的各种类型的对象：

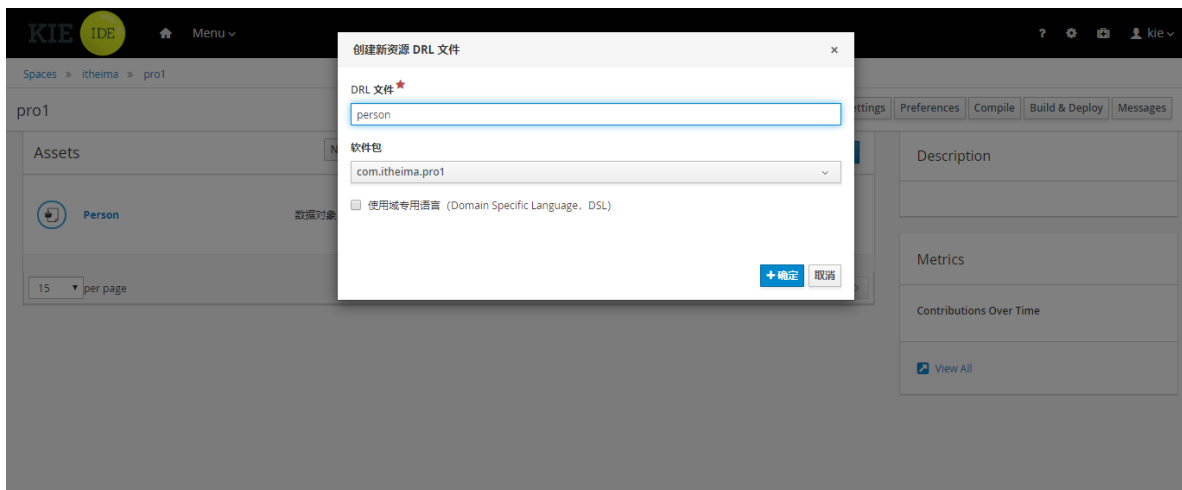


8.3.3 创建DRL规则文件

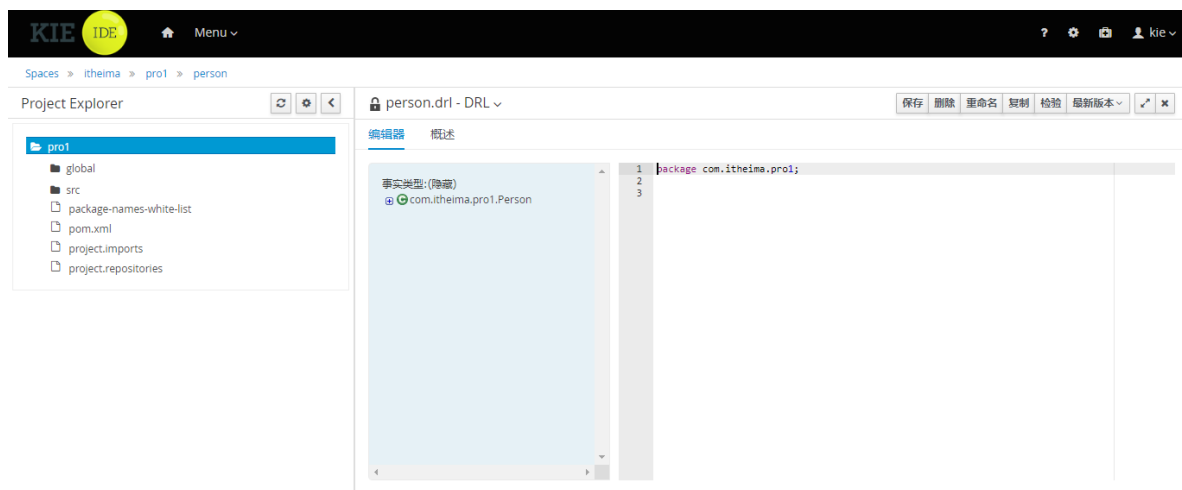
第一步：在pro1项目页面点击右上角Create New Asset按钮，选择“DRL文件”，弹出创建DRL文件窗口



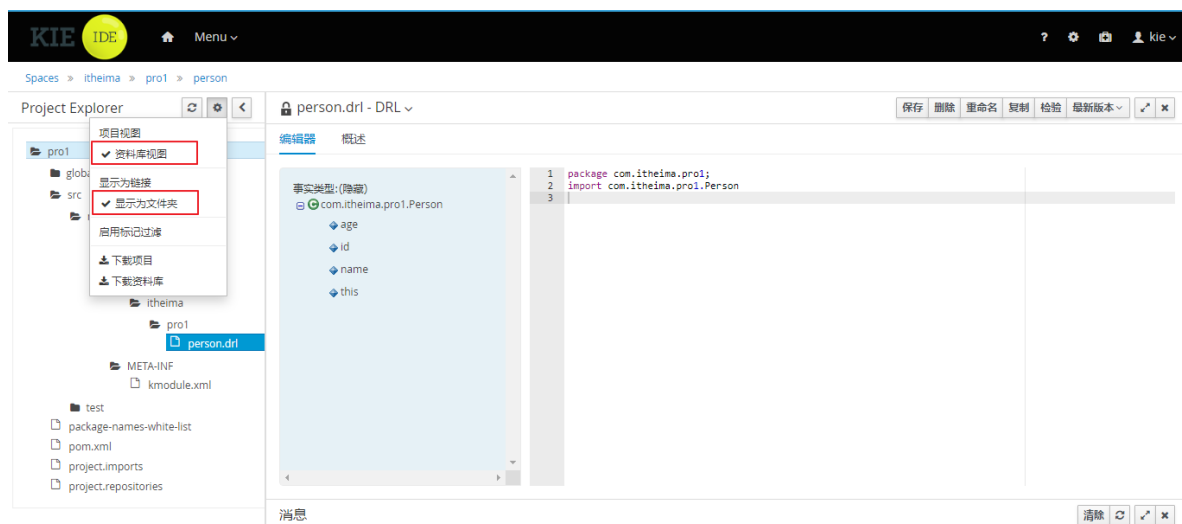
第二步：在添加DRL文件窗口录入DRL文件名称，点击确定按钮完成操作



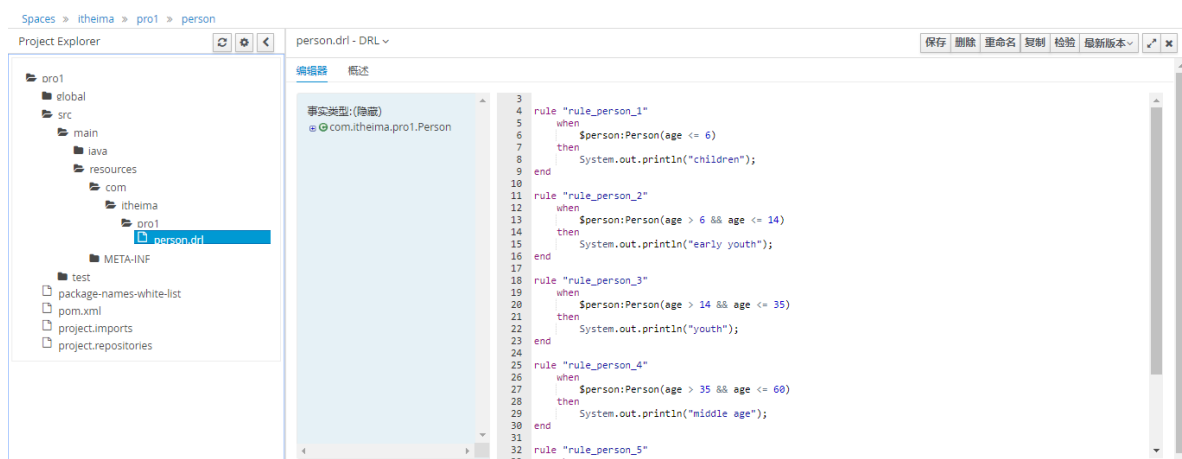
第三步：上面点击确定按钮完成创建DRL文件后，页面会跳转到编辑DRL文件页面



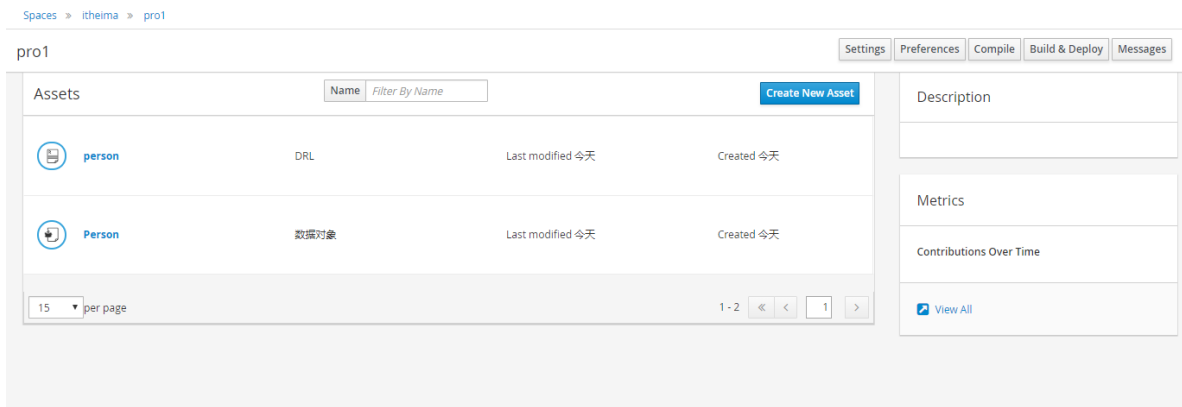
可以看到DRL规则文件页面分为两个部分：左侧为项目浏览视图、右侧为编辑区域，需要注意的是左侧默认展示的不是项目浏览视图，需要点击上面设置按钮，选择“资料库视图”和“显示为文件夹”，如下图所示：



第四步：在编辑DRL文件页面右侧区域进行DRL文件的编写，点击右上角保存按钮完成保存操作，点击检验按钮进行规则文件语法检查



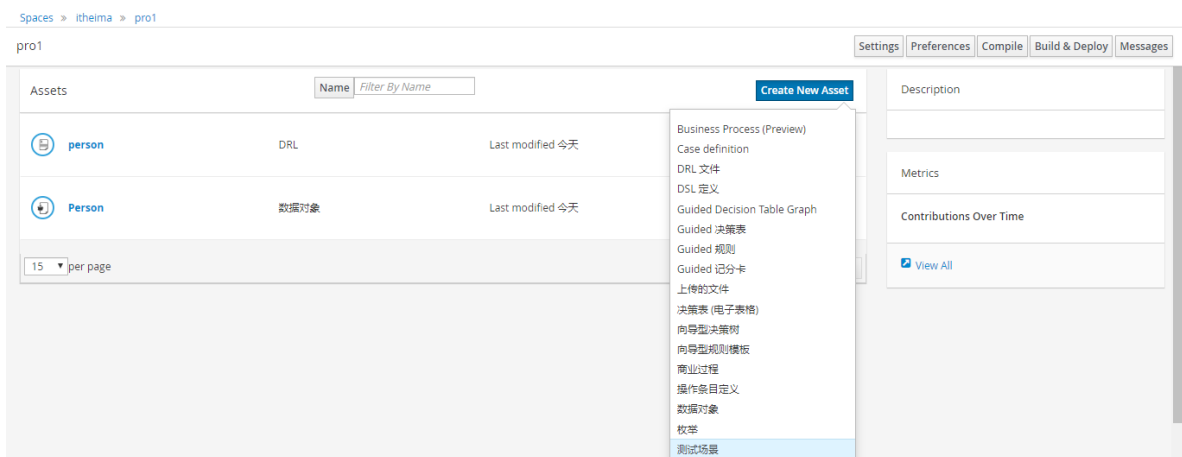
点击左上角pro1项目回到项目页面，可以看到此项目下已经存在两个对象，即person.drl规则文件和Person类：



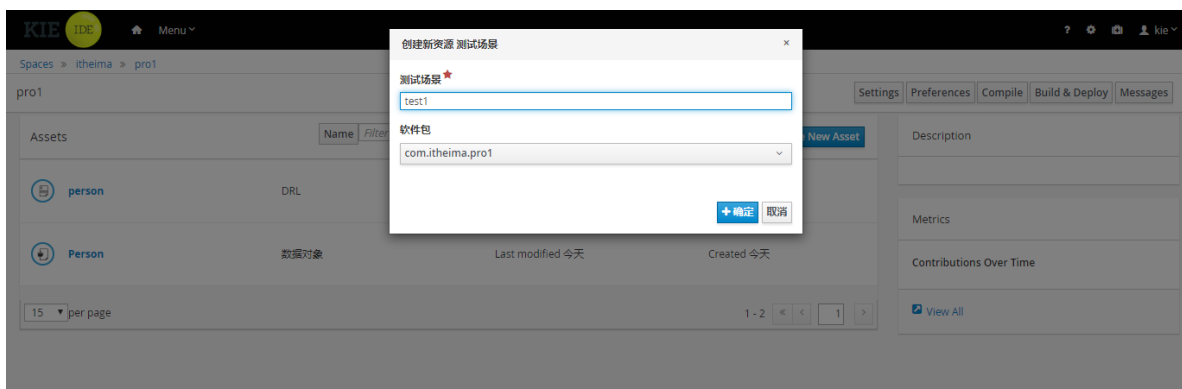
8.3.4 创建测试场景

前面我们已经创建了Person数据对象和person规则文件，现在我们需要测试一下规则文件中的规则，可以通过创建测试场景来进行测试。

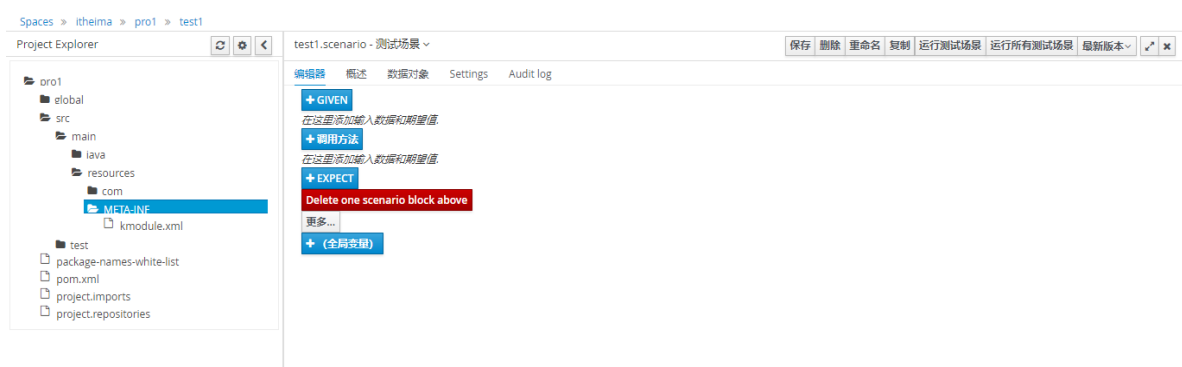
第一步：在项目页面点击Create New Asset按钮选择“测试场景”，弹出创建测试场景窗口



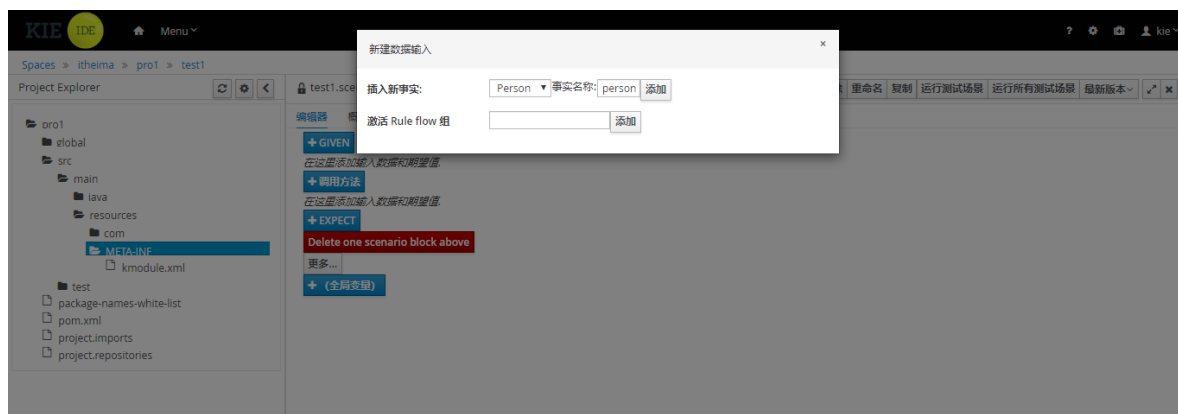
第二步：在弹出的创建测试场景窗口中录入测试场景的名称，点击确定完成操作



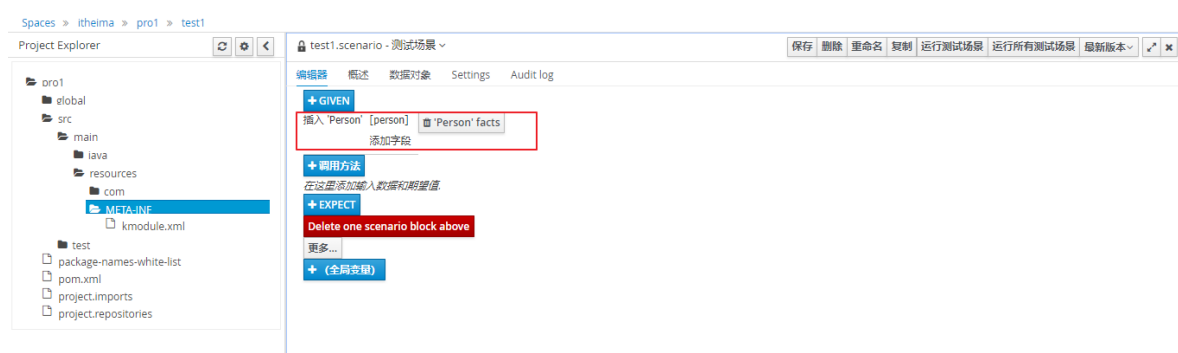
完成测试场景的创建后，页面会跳转到测试场景编辑页面，如下图：



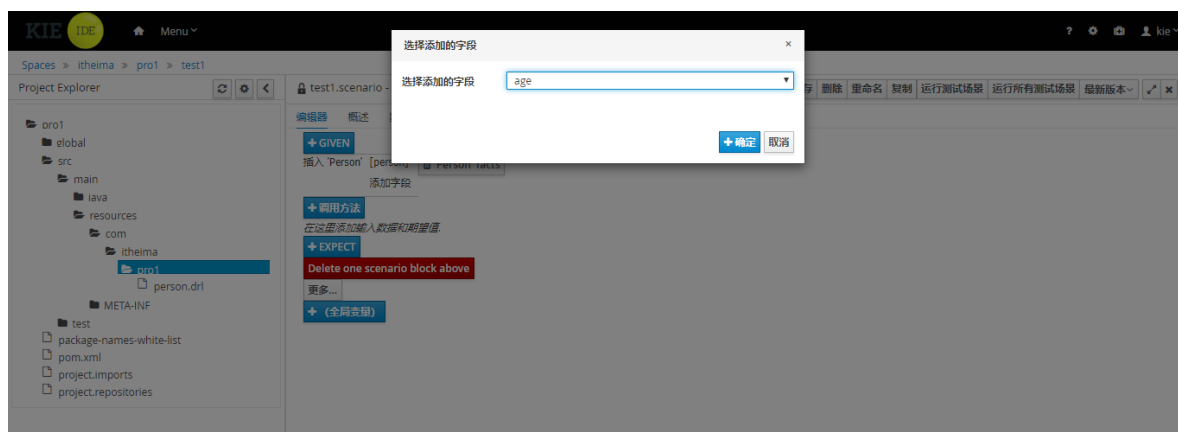
第三步：因为我们编写的规则文件中需要从工作内存中获取Person对象进行规则匹配，所以在测试场景中需要准备Person对象给工作内存，点击“GIVEN”按钮弹出新建数据录入窗口，选择Person类，输入框中输入事实名称（名称任意），如下图



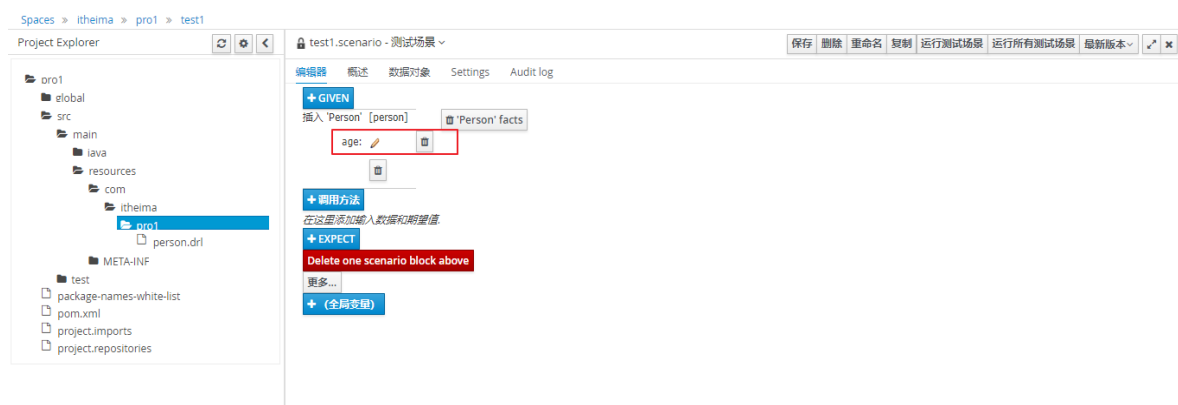
第四步：录入事实名称后点击后面的添加按钮，可以看到Person对象已经添加成功



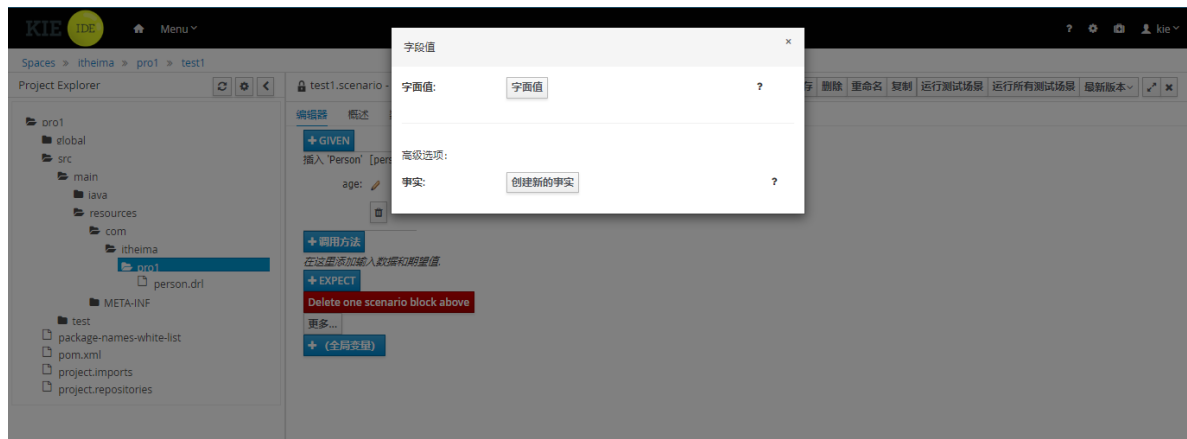
第五步：我们给工作内存提供的Person对象还需要设置age属性的值，点击“添加字段”按钮弹出窗口，选择age属性



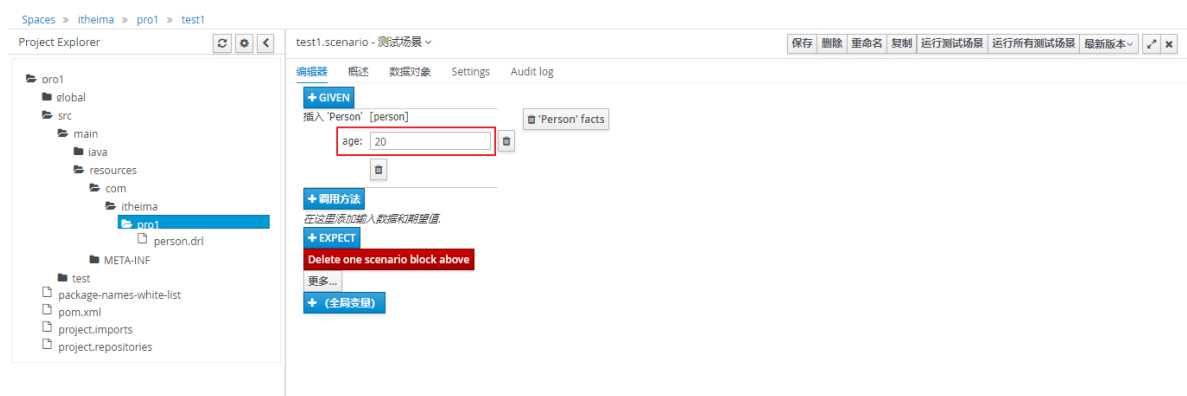
点击确定按钮后可以看到字段已经添加成功：



第六步：点击age属性后面的编辑按钮，弹出字段值窗口

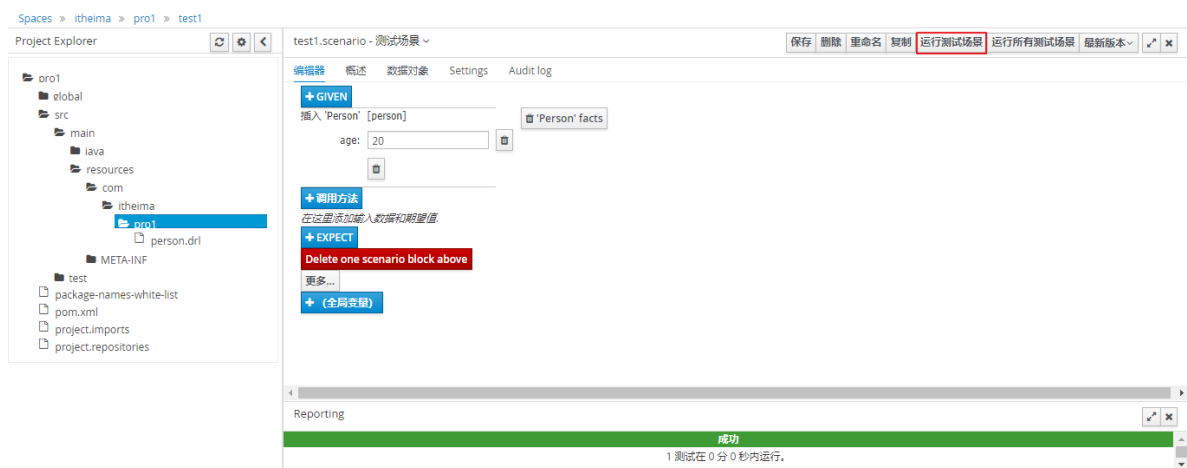


第七步：在弹出的窗口中点击字面值按钮，重新回到测试场景页面，可以看到age后面出现输入框，可以为age属性设置值



设置好age属性的值后点击保存按钮保存测试场景

第八步：点击右上角“运行测试场景”按钮进行测试



测试成功后可以查看WorkBench部署的Tomcat控制台：

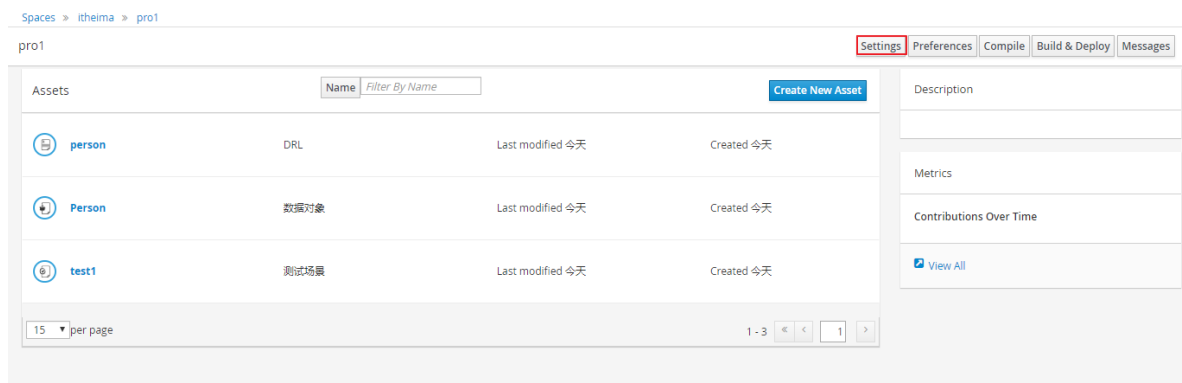
```

Tomcat
13-Jan-2020 10:15:33.215 信息 [localhost-startStop-1] org.apache.catalina.core.StandardContext.checkUnusualURLPattern Su
spicious url pattern: "*.cache.html" in context [/kie-drools-wb] - see sections 12.1 and 12.2 of the Servlet specificati
on
13-Jan-2020 10:15:33.215 信息 [localhost-startStop-1] org.apache.catalina.core.StandardContext.checkUnusualURLPattern Su
spicious url pattern: "*.nocache.js" in context [/kie-drools-wb] - see sections 12.1 and 12.2 of the Servlet specificati
on
13-Jan-2020 10:15:33.700 信息 [localhost-startStop-1] org.apache.jasper.servlet.TldScanner.scanJars At least one JAR was
scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs that were scan
ned but no TLDs were found in them. Skipping unneeded JARs during scanning can improve startup time and JSP compilation
time.
ERROR StatusLogger Log4j2 could not find a logging implementation. Please add log4j-core to the classpath. Using SimpleL
ogger to log to the console...
13-Jan-2020 10:15:33.950 信息 [localhost-startStop-1] org.apache.deltaspike.core.util.ProjectStageProducer.initProjectSt
age Computed the following DeltaSpike ProjectStage: Production
13-Jan-2020 10:15:58.323 信息 [localhost-startStop-1] org.apache.catalina.startup.HostConfig.deployWAR Deployment of web
application archive [D:\work\libs\apache-tomcat-8.5.24\webapps\kie-drools-wb.war] has finished in [38,796] ms
13-Jan-2020 10:15:58.339 信息 [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
13-Jan-2020 10:15:58.354 信息 [main] org.apache.catalina.startup.Catalina.start Server startup in 38891 ms
youth

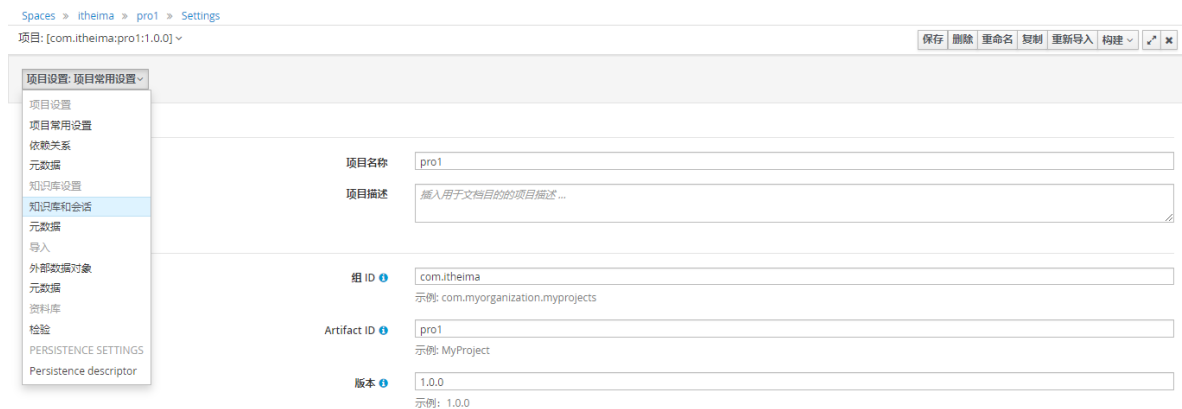
```

8.3.5 设置KieBase和KieSession

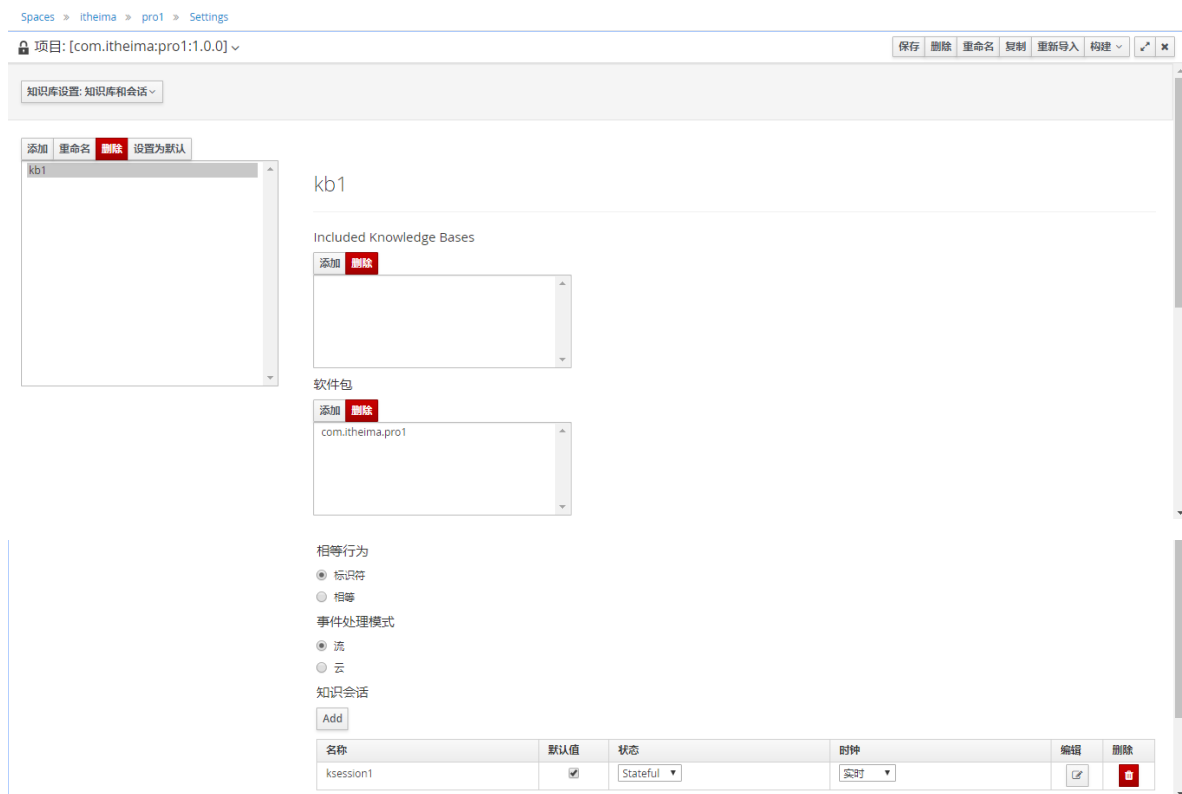
第一步：在pro1项目页面点击右上角Settings按钮进入设置页面



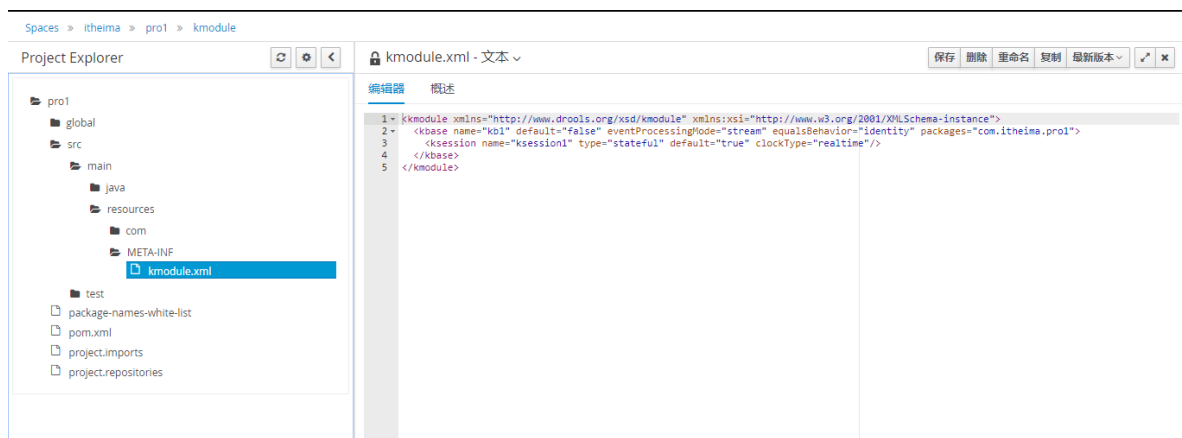
第二步：在设置页面选择“知识库和会话”选项



第三步：在弹出的知识库和会话页面点击“添加”按钮进行设置

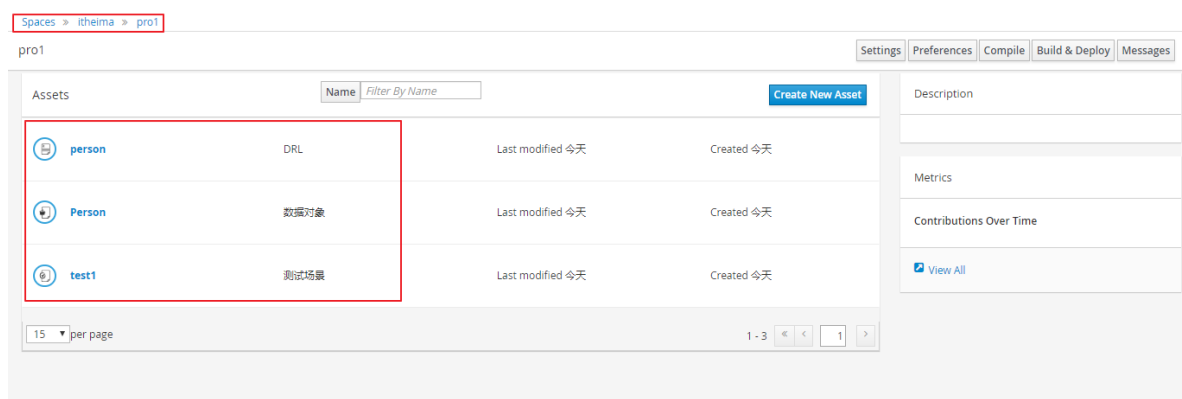


第四步：设置完成后点击右上角保存按钮完成设置操作，可以通过左侧浏览视图点击 kmodule.xml，查看文件内容






8.3.6 编译、构建、部署

前面我们已经在WorkBench中创建了一个空间itheima，并且在此空间中创建了一个项目 pro1，在此项目中创建了数据文件、规则文件和测试场景，如下图：



点击右上角“Compile”按钮可以对项目进行编译，点击“Build&Deploy”按钮进行构建和部署。

部署成功后可以在本地maven仓库中看到当前项目已经被打成jar包：

| > zhaoqx > .m2 > repository > com > itheima > pro1 > 1.0.0 | | |
|--|-----------------|--------------|
| 名称 | 修改日期 | 类型 |
|  _remote.repositories | 2020/1/13 16:06 | REPOSITORY |
|  pro1-1.0.0.jar | 2020/1/13 16:06 | Executable J |
|  pro1-1.0.0.pom | 2020/1/13 16:06 | POM 文件 |

将上面的jar包进行解压，可以看到我们创建的数据对象Person和规则文件person以及kmodule.xml都已经打到jar包中了。

8.3.7 在项目中使用部署的规则

前面我们已经在WorkBench中创建了pro1项目，并且在pro1项目中创建了数据文件、规则文件等。最后我们将此项目打成jar包部署到了maven仓库中。本小节就需要在外部项目中使用我们定义的规则。

第一步：在IDEA中创建一个maven项目并在pom.xml文件中导入相关坐标

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <version>7.10.0.Final</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
```

第二步：在项目中创建一个数据对象Person，需要和WorkBench中创建的Person包名、类名完全相同，属性也需要对应

```
package com.itheima.pro1;

public class Person implements java.io.Serializable {

    static final long serialVersionUID = 1L;

    private java.lang.String id;
    private java.lang.String name;
    private int age;

    public Person() {
    }

    public java.lang.String getId() {
        return this.id;
    }
}
```

```

    public void setId(java.lang.String id) {
        this.id = id;
    }

    public java.lang.String getName() {
        return this.name;
    }

    public void setName(java.lang.String name) {
        this.name = name;
    }

    public int getAge() {
        return this.age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Person(java.lang.String id, java.lang.String name, int age) {
        this.id = id;
        this.name = name;
        this.age = age;
    }
}

```

第三步：编写单元测试，远程加载maven仓库中的jar包最终完成规则调用

```

@Test
public void test1() throws Exception{
    //通过此URL可以访问到maven仓库中的jar包
    //URL地址构成：http://ip地址:Tomcat端口号/WorkBench工程名/maven2/坐标/版本号/xxx.jar
    String url =
        "http://localhost:8080/kie-drools-
wb/maven2/com/itheima/pro1/1.0.0/pro1-1.0.0.jar";

    Kieservices kieservices = Kieservices.Factory.get();

    //通过Resource资源对象加载jar包
    UrlResource resource = (UrlResource)
    kieservices.getResources().newUrlResource(url);
    //通过workbench提供的服务来访问maven仓库中的jar包资源，需要先进行workbench
    的认证
    resource.setUsername("kie");
    resource.setPassword("kie");
    resource.setBasicAuthentication("enabled");

    //将资源转换为输入流，通过此输入流可以读取jar包数据
    InputStream inputStream = resource.getInputStream();
}

```

```

//创建仓库对象，仓库对象中保存Drools的规则信息
KieRepository repository = kieServices.getRepository();

//通过输入流读取maven仓库中的jar包数据，包装成KieModule模块添加到仓库中
KieModule kieModule =
    repository.

addKieModule(kieServices.getResources().newInputStreamResource(inputStrea
m));

//基于KieModule模块创建容器对象，从容器中可以获取session会话
KieContainer kieContainer =
kieServices.newKieContainer(kieModule.getReleaseId());
KieSession session = kieContainer.newKieSession();

Person person = new Person();
person.setAge(10);
session.insert(person);

session.fireAllRules();
session.dispose();
}

```

执行单元测试可以发现控制台已经输出了相关内容。通过WorkBench修改规则输出内容并发布，再次执行单元测试可以发现控制台输出的内容也发生了变化。

通过上面的案例可以发现，我们在IEDA中开发的项目中并没有编写规则文件，规则文件是我们通过WorkBench开发并安装部署到maven仓库中，我们自己开发的项目只需要远程加载maven仓库中的jar包就可以完成规则的调用。这种开发方式的好处是我们的应用可以和业务规则完全分离，同时通过WorkBench修改规则后我们的应用不需要任何修改就可以加载到最新的规则从而实现规则的动态变更。

9. Drools实战

9.1 个人所得税计算器

本小节我们需要通过Drools规则引擎来根据规则计算个人所得税，最终页面效果如下：

个人所得税计算器（2011版）

| | |
|------------------------------------|------------------------------------|
| 税前月收入 | <input type="text" value="15000"/> |
| <input type="button" value="计 算"/> | |
| 应纳税所得额 | <input type="text" value="11500"/> |
| 税率 | <input type="text" value="0.25"/> |
| 速算扣除数 | <input type="text" value="1005"/> |
| 扣税额 | <input type="text" value="1870"/> |
| 税后工资 | <input type="text" value="13130"/> |

9.1.1 名词解释

税前月收入：即税前工资，指交纳个人所得税之前的总工资

应纳税所得额：指按照税法规定确定纳税人在一定期间所获得的所有应税收入减除在该纳税期间依法允许减除的各种支出后的余额

税率：是对征税对象的征收比例或征收额度

速算扣除数：指为解决超额累进税率分级计算税额的复杂技术问题，而预先计算出的一个数据，可以简化计算过程

扣税额：是指实际缴纳的税额

税后工资：是指扣完税后实际到手的工资收入

9.1.2 计算规则

要实现个人所得税计算器，需要了解如下计算规则：

| 规则编号 | 名称 | 描述 |
|------|---------------------------|--------------------------------------|
| 1 | 计算应纳税所得额 | 应纳税所得额为税前工资减去3500 |
| 2 | 设置税率，应纳税所得额 ≤1500 | 税率为0.03，速算扣除数为0 |
| 3 | 设置税率，应纳税所得额在1500至4500之间 | 税率为0.1，速算扣除数为105 |
| 4 | 设置税率，应纳税所得额在4500至9000之间 | 税率为0.2，速算扣除数为555 |
| 5 | 设置税率，应纳税所得额在9000至35000之间 | 税率为0.25，速算扣除数为1005 |
| 6 | 设置税率，应纳税所得额在35000至55000之间 | 税率为0.3，速算扣除数为2755 |
| 7 | 设置税率，应纳税所得额在55000至80000之间 | 税率为0.35，速算扣除数为5505 |
| 8 | 设置税率，应纳税所得额在80000以上 | 税率为0.45，速算扣除数为13505 |
| 9 | 计算税后工资 | 扣税额=应纳税所得额*税率-速算扣除数 税后工资=税前工资-扣税额 |

9.1.3 实现步骤

本实战案例我们基于Spring Boot整合Drools的方式来实现。

第一步：创建maven工程calculation并配置pom.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starters</artifactId>
        <version>2.0.6.RELEASE</version>
    </parent>
    <groupId>cn.itcast</groupId>
    <artifactId>calculation</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
```



```

</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
</dependency>
<dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
</dependency>
<!--drools规则引擎-->
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>7.6.0.Final</version>
</dependency>
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>7.6.0.Final</version>
</dependency>
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-templates</artifactId>
    <version>7.6.0.Final</version>
</dependency>
<dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>7.6.0.Final</version>
</dependency>
<dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-spring</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>
    </exclusions>

```

```

        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
    </exclusion>
</exclusions>
    <version>7.6.0.Final</version>
</dependency>
</dependencies>
<build>
    <finalName>${project.artifactId}</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.xml</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.*</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
    </resources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

第二步：创建/resources/application.yml文件

```

server:
  port: 8080
spring:
  application:
    name: calculation

```

第三步：编写配置类DroolsConfig

```

package com.itheima.drools.config;
import org.kie.api.KieBase;

```

```

import org.kie.api.KieServices;
import org.kie.api.builder.KieBuilder;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieRepository;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.internal.io.ResourceFactory;
import org.kie.spring.KModuleBeanFactoryPostProcessor;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.core.io.support.PathMatchingResourcePatternResolver;
import org.springframework.core.io.support.ResourcePatternResolver;
import org.springframework.core.io.Resource;
import java.io.IOException;
/**
 * 规则引擎配置类
 */
@Configuration
public class DroolsConfig {
    //指定规则文件存放的目录
    private static final String RULES_PATH = "rules/";
    private final KieServices kieServices = KieServices.Factory.get();
    @Bean
    @ConditionalOnMissingBean
    public KieFileSystem kieFileSystem() throws IOException {
        System.setProperty("drools.dateformat", "yyyy-MM-dd");
        KieFileSystem kieFileSystem = kieServices.newKieFileSystem();
        ResourcePatternResolver resourcePatternResolver =
            new PathMatchingResourcePatternResolver();
        Resource[] files =
            resourcePatternResolver.getResources("classpath*:" +
RULES_PATH + "/*.xml");
        String path = null;
        for (Resource file : files) {
            path = RULES_PATH + file.getFilename();
            kieFileSystem.write(ResourceFactory.newClassPathResource(path,
"UTF-8"));
        }
        return kieFileSystem;
    }
    @Bean
    @ConditionalOnMissingBean
    public KieContainer kieContainer() throws IOException {
        KieRepository kieRepository = kieServices.getRepository();
        kieRepository.addKieModule(kieRepository::getDefaultReleaseId);
        KieBuilder kieBuilder =
kieServices.newKieBuilder(kieFileSystem());
        kieBuilder.buildAll();
    }
}

```

```

        return
    }
    kieServices.newKieContainer(kieRepository.getDefaultReleaseId());
}
@Bean
@ConditionalOnMissingBean
public KieBase kieBase() throws IOException {
    return kieContainer().getKieBase();
}
@Bean
@ConditionalOnMissingBean
public KModuleBeanFactoryPostProcessor kiePostProcessor() {
    return new KModuleBeanFactoryPostProcessor();
}
}

```

第四步：编写实体类Calculation

```

package com.itheima.drools.entity;

public class Calculation {
    private double wage;//税前工资
    private double wagemore;//应纳税所得额
    private double cess;//税率
    private double preminus;//速算扣除数
    private double wageminus;//扣税额
    private double actualwage;//税后工资

    public double getWage() {
        return wage;
    }

    public void setWage(double wage) {
        this.wage = wage;
    }

    public double getActualwage() {
        return actualwage;
    }

    public void setActualwage(double actualwage) {
        this.actualwage = actualwage;
    }

    public double getWagemore() {
        return wagemore;
    }

    public void setWagemore(double wagemore) {
        this.wagemore = wagemore;
    }
}

```

```

    public double getCess() {
        return cess;
    }

    public void setCess(double cess) {
        this.cess = cess;
    }

    public double getPreminus() {
        return preminus;
    }

    public void setPreminus(double preminus) {
        this.preminus = preminus;
    }

    public double getWageminus() {
        return wageminus;
    }

    public void setWageminus(double wageminus) {
        this.wageminus = wageminus;
    }

    @Override
    public String toString() {
        return "Calculation{" +
            "wage=" + wage +
            ", actualwage=" + actualwage +
            ", wagemore=" + wagemore +
            ", cess=" + cess +
            ", preminus=" + preminus +
            ", wageminus=" + wageminus +
            '}';
    }
}

```

第五步：在resources/rules下创建规则文件calculation.drl文件

```

package calculation
import com.itheima.drools.entity.Calculation

rule "个人所得税：计算应纳税所得额"
    enabled true
    salience 3
    no-loop true
    date-effective "2011-09-01" //生效日期
    when
        $cal : Calculation(wage>0)
    then
        $cal.setWagemore($cal.getWage()-3500);

```

```

        update($cal);
    end

    rule "个人所得税：设置税率-->应纳税所得额<=1500"
        salience 2
        no-loop true
        activation-group "SETCess_Group"
        when
            $cal : Calculation(wagemore <= 1500)
        then
            $cal.setCess(0.03);
            $cal.setPreminus(0);
            update($cal);
        end

    rule "个人所得税：设置税率-->应纳税所得额在1500至4500之间"
        salience 2
        no-loop true
        activation-group "SETCess_Group"
        when
            $cal : Calculation(wagemore > 1500 && wagemore <= 4500)
        then
            $cal.setCess(0.1);
            $cal.setPreminus(105);
            update($cal);
        end

    rule "个人所得税：设置税率-->应纳税所得额在4500至9000之间"
        salience 2
        no-loop true
        activation-group "SETCess_Group"
        when
            $cal : Calculation(wagemore > 4500 && wagemore <= 9000)
        then
            $cal.setCess(0.2);
            $cal.setPreminus(555);
            update($cal);
        end

    rule "个人所得税：设置税率-->应纳税所得额在9000至35000之间"
        salience 2
        no-loop true
        activation-group "SETCess_Group"
        when
            $cal : Calculation(wagemore > 9000 && wagemore <= 35000)
        then
            $cal.setCess(0.25);
            $cal.setPreminus(1005);
            update($cal);
        end

    rule "个人所得税：设置税率-->应纳税所得额在35000至55000之间"

```

```

    salience 2
    no-loop true
    activation-group "SETCess_Group"
    when
        $cal : Calculation(wagemore > 35000 && wagemore <= 55000)
    then
        $cal.setCess(0.3);
        $cal.setPreminus(2755);
        update($cal);
    end

rule "个人所得税：设置税率-->应纳税所得额在 55000至80000之间"
    salience 2
    no-loop true
    activation-group "SETCess_Group"
    when
        $cal : Calculation(wagemore > 55000 && wagemore <= 80000)
    then
        $cal.setCess(0.35);
        $cal.setPreminus(5505);
        update($cal);
    end

rule "个人所得税：设置税率-->应纳税所得额在 80000以上"
    salience 2
    no-loop true
    activation-group "SETCess_Group"
    when
        $cal : Calculation(wagemore > 80000)
    then
        $cal.setCess(0.45);
        $cal.setPreminus(13505);
        update($cal);
    end

rule "个人所得税：计算税后工资"
    salience 1
    when
        $cal : Calculation(wage > 0 && wagemore > 0 && wagemore > 0 &&
cess > 0)
    then

        $cal.setWageminus($cal.getWagemore()*$cal.getCess()-$cal.getPreminus());
        $cal.setActualwage($cal.getWage()-$cal.getWageminus());

        System.out.println("-----税前工资: "+$cal.getWage());
        System.out.println("-----应纳税所得额: "+$cal.getWagemore());
        System.out.println("-----税率: " + $cal.getCess());
        System.out.println("-----速算扣除数: " + $cal.getPreminus());
        System.out.println("-----扣税额: " + $cal.getWageminus());
        System.out.println("-----税后工资: " + $cal.getActualwage());
    end

```

第六步：创建RuleService

```
package com.itheima.drools.service;

import com.itheima.drools.entity.Calculation;
import org.kie.api.KieBase;
import org.kie.api.runtime.KieSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

/**
 * 调用规则引擎，执行规则
 */
@Service
public class RuleService {
    @Autowired
    private KieBase kieBase;

    //个人所得税计算
    public Calculation calculate(Calculation calculation){
        KieSession kieSession = kieBase.newKieSession();
        kieSession.insert(calculation);
        kieSession.fireAllRules();
        kieSession.dispose();
        return calculation;
    }
}
```

第七步：创建RuleController

```
package com.itheima.drools.controller;

import com.itheima.drools.entity.Calculation;
import com.itheima.drools.service.RuleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/rule")
public class RuleController {
    @Autowired
    private RuleService ruleService;

    @RequestMapping("/calculate")
    public Calculation calculate(double wage){
        Calculation calculation = new Calculation();
        calculation.setWage(wage);
        calculation = ruleService.calculate(calculation);
        System.out.println(calculation);
    }
}
```



```
        return calculation;
    }
}
```

第八步：创建启动类DroolsApplication

```
package com.itheima.drools;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DroolsApplication {
    public static void main(String[] args) {
        SpringApplication.run(DroolsApplication.class);
    }
}
```

第九步：导入静态资源文件到resources/static目录下

9.2 信用卡申请

本小节我们需要通过Drools规则引擎来根据规则进行申请人的合法性检查，检查通过后再根据规则确定信用卡额度，最终页面效果如下：

XX银行信用卡申请

| | | | |
|------|------------------------------------|----------|-----------------------------------|
| 姓名 | <input type="text"/> | 性别 | <input type="text"/> |
| 年龄 | <input type="text"/> | 手机号 | <input type="text"/> |
| 住址 | <input type="text"/> | 学历 | <input type="text" value="专科以下"/> |
| 月收入 | <input type="text" value="10000"/> | 现持有信用卡数量 | <input type="text" value="1"/> |
| 是否有房 | <input type="text" value="是"/> | 是否有车 | <input type="text" value="否"/> |

申请

恭喜你信用卡申请成功，信用卡额度为：8000

9.2.1 计算规则

合法性检查规则如下：

| 规则编号 | 名称 | 描述 |
|------|---------------|---|
| 1 | 检查学历与薪水1 | 如果申请人既没房也没车，同时学历为大专以下，并且月薪少于5000，那么不通过 |
| 2 | 检查学历与薪水2 | 如果申请人既没房也没车，同时学历为大专或本科，并且月薪少于3000，那么不通过 |
| 3 | 检查学历与薪水3 | 如果申请人既没房也没车，同时学历为本科以上，并且月薪少于2000，同时之前没有信用卡的，那么不通过 |
| 4 | 检查申请人已有的信用卡数量 | 如果申请人现有的信用卡数量大于10，那么不通过 |

信用卡额度确定规则：

| 规则编号 | 名称 | 描述 |
|------|-----|---|
| 1 | 规则1 | 如果申请人有房有车，或者月收入在20000以上，那么发放的信用卡额度为15000 |
| 2 | 规则2 | 如果申请人没房没车，但月收入在10000~20000之间，那么发放的信用卡额度为6000 |
| 3 | 规则3 | 如果申请人没房没车，月收入在10000以下，那么发放的信用卡额度为3000 |
| 4 | 规则4 | 如果申请人有房没车或者没房但有车，月收入在10000以下，那么发放的信用卡额度为5000 |
| 5 | 规则5 | 如果申请人有房没车或者是没房但有车，月收入在10000~20000之间，那么发放的信用卡额度为8000 |

9.2.2 实现步骤

第一步：创建maven工程creditCardApply并配置pom.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starters</artifactId>
  <version>2.0.6.RELEASE</version>
</parent>
<groupId>com.itheima</groupId>
<artifactId>creditCardApply</artifactId>
<version>1.0-SNAPSHOT</version>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
  <dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
  </dependency>
  <!--drools规则引擎-->
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-templates</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-spring</artifactId>
    <exclusions>
```

```

        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </exclusion>
    </exclusions>
    <version>7.6.0.Final</version>
</dependency>
</dependencies>
<build>
    <finalName>${project.artifactId}</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.xml</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.*</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
    </resources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

第二步：创建/resources/application.yml文件

```
server:
  port: 8080
spring:
  application:
    name: creditCardApply
```

第三步：编写配置类DroolsConfig

```
package com.itheima.drools.config;
import org.kie.api.KieBase;
import org.kie.api.KieServices;
import org.kie.api.builder.KieBuilder;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieRepository;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.internal.io.ResourceFactory;
import org.kie.spring.KModuleBeanFactoryPostProcessor;
import
org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.core.io.support.PathMatchingResourcePatternResolver;
import org.springframework.core.io.support.ResourcePatternResolver;
import org.springframework.core.io.Resource;
import java.io.IOException;
/**
 * 规则引擎配置类
 */
@Configuration
public class DroolsConfig {
    //指定规则文件存放的目录
    private static final String RULES_PATH = "rules/";
    private final KieServices kieServices = KieServices.Factory.get();
    @Bean
    @ConditionalOnMissingBean
    public KieFileSystem kieFileSystem() throws IOException {
        KieFileSystem kieFileSystem = kieServices.newKieFileSystem();
        ResourcePatternResolver resourcePatternResolver =
            new PathMatchingResourcePatternResolver();
        Resource[] files =
            resourcePatternResolver.getResources("classpath*:" +
RULES_PATH + "/*.");
        String path = null;
        for (Resource file : files) {
            path = RULES_PATH + file.getFilename();
            kieFileSystem.write(ResourceFactory.newClassPathResource(path,
"UTF-8"));
        }
    }
}
```

```

    }
    return kieFileSystem;
}
@Bean
@ConditionalOnMissingBean
public KieContainer kieContainer() throws IOException {
    KieRepository kieRepository = kieServices.getRepository();
    kieRepository.addKieModule(kieRepository::getDefaultReleaseId);
    KieBuilder kieBuilder =
kieServices.newKieBuilder(kieFileSystem());
    kieBuilder.buildAll();
    return
kieServices.newKieContainer(kieRepository.getDefaultReleaseId());
}
@Bean
@ConditionalOnMissingBean
public KieBase kieBase() throws IOException {
    return kieContainer().getKieBase();
}
@Bean
@ConditionalOnMissingBean
public KModuleBeanFactoryPostProcessor kiePostProcessor() {
    return new KModuleBeanFactoryPostProcessor();
}
}

```

第四步：编写实体类CreditCardApplyInfo

```

package com.itheima.drools.entity;
/**
 * 信用卡申请信息
 */
public class CreditCardApplyInfo {
    public static final String EDUCATION_1 = "专科以下";
    public static final String EDUCATION_2 = "专科";
    public static final String EDUCATION_3 = "本科";
    public static final String EDUCATION_4 = "本科以上";

    private String name;
    private String sex;
    private int age;
    private String education;
    private String telephone;
    private double monthlyIncome = 0; //月收入
    private String address;

    private boolean hasHouse = false; //是否有房
    private boolean hasCar = false; //是否有车
    private int hasCreditCardCount = 0; //现持有信用卡数量

    private boolean checkResult = true; //审核是否通过
}

```

```
private double quota = 0;//额度

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getSex() {
    return sex;
}

public void setSex(String sex) {
    this.sex = sex;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}

public String getEducation() {
    return education;
}

public void setEducation(String education) {
    this.education = education;
}

public String getTelephone() {
    return telephone;
}

public void setTelephone(String telephone) {
    this.telephone = telephone;
}

public double getMonthlyIncome() {
    return monthlyIncome;
}

public void setMonthlyIncome(double monthlyIncome) {
    this.monthlyIncome = monthlyIncome;
}

public String getAddress() {
    return address;
}
```

```
}

public void setAddress(String address) {
    this.address = address;
}

public boolean isHasHouse() {
    return hasHouse;
}

public void setHasHouse(boolean hasHouse) {
    this.hasHouse = hasHouse;
}

public boolean isHasCar() {
    return hasCar;
}

public void setHasCar(boolean hasCar) {
    this.hasCar = hasCar;
}

public int getHasCreditCardCount() {
    return hasCreditCardCount;
}

public void setHasCreditCardCount(int hasCreditCardCount) {
    this.hasCreditCardCount = hasCreditCardCount;
}

public boolean isCheckResult() {
    return checkResult;
}

public void setCheckResult(boolean checkResult) {
    this.checkResult = checkResult;
}

public double getQuota() {
    return quota;
}

public void setQuota(double quota) {
    this.quota = quota;
}

public String toString() {
    if(checkResult){
        return "审核通过，信用卡额度为：" + quota;
    }else {
        return "审核不通过";
    }
}
```


第五步：在resources/rules下创建规则文件creditCardApply.drl文件

[illegible]

```

        $c.setCheckResult(false);
        drools.halt();
    end
    rule "如果申请人现有的信用卡数量大于10，那么不通过"
        salience 10
        no-loop true
        when
            $c:CreditCardApplyInfo(hasCreditCardCount > 10)
        then
            $c.setCheckResult(false);
            drools.halt();
        end
    //-----
    --
    //确定额度
    rule "如果申请人有房有车，或者月收入在20000以上，那么发放的信用卡额度为15000"
        salience 1
        no-loop true
        activation-group "quota_group"
        when
            $c:CreditCardApplyInfo(checkResult == true &&
                                    ((hasHouse == true && hasCar == true) ||
                                     (monthlyIncome > 20000)))
        then
            $c.setQuota(15000);
        end
    rule "如果申请人没房没车，但月收入在10000~20000之间，那么发放的信用卡额度为6000"
        salience 1
        no-loop true
        activation-group "quota_group"
        when
            $c:CreditCardApplyInfo(checkResult == true &&
                                    hasHouse == false &&
                                    hasCar == false &&
                                    monthlyIncome >= 10000 &&
                                    monthlyIncome <= 20000)
        then
            $c.setQuota(6000);
        end
    rule "如果申请人没房没车，月收入在10000以下，那么发放的信用卡额度为3000"
        salience 1
        no-loop true
        activation-group "quota_group"
        when
            $c:CreditCardApplyInfo(checkResult == true &&
                                    hasHouse == false &&
                                    hasCar == false &&
                                    monthlyIncome < 10000)
        then
            $c.setQuota(3000);
        end
end

```

```

rule "如果申请人有房没车或者没房但有车，月收入在10000以下，那么发放的信用卡额度为5000"
    salience 1
    no-loop true
    activation-group "quota_group"
    when
        $c:CreditCardApplyInfo(checkResult == true &&
                                ((hasHouse == true && hasCar == false) ||
                                 (hasHouse == false && hasCar == true)) &&
                                monthlyIncome < 10000)
    then
        $c.setQuota(5000);
    end
rule "如果申请人有房没车或者是没房但有车，月收入在10000~20000之间，那么发放的信用卡额度为8000"
    salience 1
    no-loop true
    activation-group "quota_group"
    when
        $c:CreditCardApplyInfo(checkResult == true &&
                                ((hasHouse == true && hasCar == false) ||
                                 (hasHouse == false && hasCar == true)) &&
                                monthlyIncome >= 10000 &&
                                monthlyIncome <= 20000)
    then
        $c.setQuota(8000);
    end
end

```

第六步：创建RuleService

```

package com.itheima.drools.service;

import com.itheima.drools.entity.CreditCardApplyInfo;
import org.kie.api.KieBase;
import org.kie.api.runtime.KieSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class RuleService {
    @Autowired
    private KieBase kieBase;

    //调用Drools规则引擎实现信用卡申请
    public CreditCardApplyInfo creditCardApply(CreditCardApplyInfo creditCardApplyInfo){
        KieSession session = kieBase.newKieSession();
        session.insert(creditCardApplyInfo);
        session.fireAllRules();
        session.dispose();
        return creditCardApplyInfo;
    }
}

```

```
}  
}
```

第七步：创建RuleController

```
package com.itheima.drools.controller;  
  
import com.itheima.drools.entity.CreditCardApplyInfo;  
import com.itheima.drools.service.RuleService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/rule")  
public class RuleController {  
    @Autowired  
    private RuleService ruleService;  
  
    @RequestMapping("/creditCardApply")  
    public CreditCardApplyInfo creditCardApply(@RequestBody  
        CreditCardApplyInfo creditCardApplyInfo){  
        creditCardApplyInfo =  
ruleService.creditCardApply(creditCardApplyInfo);  
        return creditCardApplyInfo;  
    }  
}
```

第八步：创建启动类DroolsApplication

```
package com.itheima.drools;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class DroolsApplication {  
    public static void main(String[] args) {  
        SpringApplication.run(DroolsApplication.class);  
    }  
}
```

第九步：导入静态资源文件到resources/static目录下

9.3 保险产品准入规则

9.3.1 决策表

前面的课程中我们编写的规则文件都是drl形式的文件，Drools除了支持drl形式的文件外还支持xls格式的文件（即Excel文件）。这种xls格式的文件通常称为决策表（decision table）。

决策表（decision table）是一个“精确而紧凑的”表示条件逻辑的方式，非常适合商业级别的规则。决策表与现有的drl文件可以无缝替换。Drools提供了相应的API可以将xls文件编译为drl格式的字符串。

一个决策表的例子如下：

| | | | | | | |
|------|-----------------------------|---|--------------|--------------|-----------------------|-----------------------|
| | RuleSet | rules | | | | |
| | Variables | java.util.List listRules | | | | |
| | Sequential | true | | | | |
| | Import | com.thema.drools.entity.PersonInfoEntity, java.util.List | | | | |
| | RuleTable personCheck | | | | | |
| | CONDITION | CONDITION | CONDITION | AGENDA-GROUP | ACTION | ACTION |
| | \$person : PersonInfoEntity | | | | | |
| | sex != \$1 | age < \$1 age > \$2 | salary < \$1 | | listRules.add("\$1"); | listRules.add("\$1"); |
| | | | | | | |
| 性别规则 | “男” | | | sign | 性别不对 | |
| 年龄规则 | | 22,25 | | sign | | 年龄不合适 |
| 月薪规则 | | | 10000 | sign | | 工资太低了 |

决策表语法：

| 关键字 | 说明 | 是否必须 |
|--------------|---|--|
| RuleSet | 相当于drl文件中的package | 必须，只能有一个。 如果没有设置RuleSet对应的值则使用默认值rule_table |
| Sequential | 取值为Boolean类型。true表示规则按照表格自上到下的顺序执行，false表示乱序 | 可选 |
| Import | 相当于drl文件中的import，如果引入多个类则类之间用逗号分隔 | 可选 |
| Variables | 相当于drl文件中的global，用于定义全局变量，如果有多个全局变量则中间用逗号分隔 | 可选 |
| RuleTable | 它指示了后面将会有一批rule，RuleTable的名称将会作为以后生成rule的前缀 | 必须 |
| CONDITION | 规则条件关键字，相当于drl文件中的when。下面两行则表示 LHS 部分，第三行则为注释行，不计为规则部分，从第四行开始，每一行表示一条规则 | 每个规则表至少有一个 |
| ACTION | 规则结果关键字，相当于drl文件中的then | 每个规则表至少有一个 |
| NO-LOOP | 相当于drl文件中的no-loop | 可选 |
| AGENDA-GROUP | 相当于drl文件中的agenda-group | 可选 |

在决策表中还经常使用到占位符，语法为\$后面加数字，用于替换每条规则中设置的具体值。

上面的决策表例子转换为drl格式的规则文件内容如下：

```
package rules;

import com.itheima.drools.entity.PersonInfoEntity;
import java.util.List;
global java.util.List listRules;

rule "personCheck_10"
    salience 65535
    agenda-group "sign"
    when
        $person : PersonInfoEntity(sex != "男")
    then
        listRules.add("性别不对");
    end

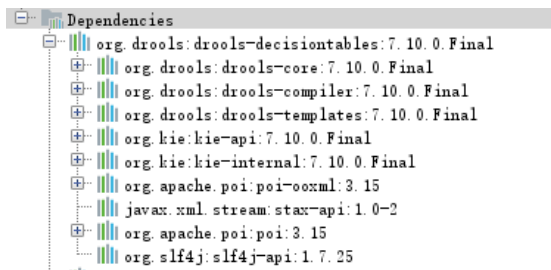
rule "personCheck_11"
    salience 65534
    agenda-group "sign"
    when
        $person : PersonInfoEntity(age < 22 || age > 25)
    then
        listRules.add("年龄不合适");
    end

rule "personCheck_12"
    salience 65533
    agenda-group "sign"
    when
        $person : PersonInfoEntity(salary < 10000)
    then
        listRules.add("工资太低了");
    end
```

要进行决策表相关操作，需要导入如下maven坐标：

```
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-decisiontables</artifactId>
    <version>7.10.0.Final</version>
</dependency>
```

通过下图可以发现，由于maven的依赖传递特性在导入drools-decisiontables坐标后，drools-core和drools-compiler等坐标也被传递了过来



Drools提供的将xls文件编译为drl格式字符串的API如下：

```
String realPath = "C:\\testRule.xls";//指定决策表xls文件的磁盘路径
File file = new File(realPath);
InputStream is = new FileInputStream(file);
SpreadsheetCompiler compiler = new SpreadsheetCompiler();
String drl = compiler.compile(is, InputType.XLS);
```

Drools还提供了基于drl格式字符串创建KieSession的API：

```
KieHelper kieHelper = new KieHelper();
kieHelper.addContent(drl, ResourceType.DRL);
KieSession session = kieHelper.build().newKieSession();
```

基于决策表的入门案例：

第一步：创建maven工程drools_decisiontable_demo并配置pom.xml文件

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-decisiontables</artifactId>
  <version>7.10.0.Final</version>
</dependency>
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
```

第二步：创建实体类PersonInfoEntity

```
package com.itheima.drools.entity;

public class PersonInfoEntity {
    private String sex;
    private int age;
    private double salary;

    public String getSex() {
```

```

        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}

```

第三步：创建xls规则文件（可以直接使用资料中提供的testRule.xls文件）

第四步：创建单元测试

```

@Test
public void test1() throws Exception{
    String realPath = "d:\\testRule.xls";//指定决策表xls文件的磁盘路径
    File file = new File(realPath);
    InputStream is = new FileInputStream(file);
    SpreadsheetCompiler compiler = new SpreadsheetCompiler();
    String drl = compiler.compile(is, InputType.XLS);

    System.out.println(drl);
    KieHelper kieHelper = new KieHelper();
    kieHelper.addContent(drl, ResourceType.DRL);
    KieSession session = kieHelper.build().newKieSession();

    PersonInfoEntity personInfoEntity = new PersonInfoEntity();
    personInfoEntity.setSex("男");
    personInfoEntity.setAge(35);
    personInfoEntity.setSalary(1000);

    List<String> list = new ArrayList<String>();
    session.setGlobal("listRules",list);

    session.insert(personInfoEntity);

    session.getAgenda().getAgendaGroup("sign").setFocus();
}

```



```
session.fireAllRules();

for (String s : list) {
    System.out.println(s);
}
session.dispose();
}
```

9.3.2 规则介绍

各保险公司针对人身、财产推出了不同的保险产品，作为商业保险公司，筛选出符合公司利益最大化的客户是非常重要的，即各保险产品的准入人群是不同的，也就是说保险公司会针对不同的人群特征，制定不同的产品缴费和赔付规则。

我们来看一下某保险产品准入规则的简化版，当不满足以下规则时，系统模块需要返回准入失败标识和失败原因

规则1: 保险公司是: PICC
规则2: 销售区域是: 北京、天津
规则3: 投保人年龄: 0 ~ 17岁
规则4: 保险期间是: 20年、25年、30年
规则5: 缴费方式是: 趸交（一次性交清）或年交
规则6: 保险期与交费期规则一: 保险期间为20年期交费期间最长10年交且不能选择[趸交]
规则7: 保险期与交费期规则二: 保险期间为25年期交费期间最长15年交且不能选择[趸交]
规则8: 保险期与交费期规则三: 保险期间为30年期交费期间最长20年交且不能选择[趸交]
规则9: 被保险人要求: （投保年龄+保险期间）不得大于40周岁
规则10: 保险金额规则: 投保时约定，最低为5万元，超过部分必须为1000元的整数倍
规则11: 出单基本保额限额规则: 线上出单基本保额限额62.5万元，超62.5万元需配合契约转线下出单

在本案例中规则文件是一个Excel文件，业务人员可以直接更改这个文件中指标的值，系统不需要做任何变更。

9.3.3 实现步骤

本案例还是基于Spring Boot整合Drools的架构来实现。

第一步: 创建maven工程insuranceInfoCheck并配置pom.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starters</artifactId>
```

```
<version>2.0.6.RELEASE</version>
</parent>
<groupId>com.itheima</groupId>
<artifactId>insuranceInfoCheck</artifactId>
<version>1.0-SNAPSHOT</version>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-aop</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
  </dependency>
  <dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.6</version>
  </dependency>
  <!--drools规则引擎-->
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-compiler</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-templates</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-api</artifactId>
    <version>7.6.0.Final</version>
  </dependency>
  <dependency>
    <groupId>org.kie</groupId>
    <artifactId>kie-spring</artifactId>
    <exclusions>
      <exclusion>
        <groupId>org.springframework</groupId>
        <artifactId>spring-tx</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
```

```

        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
        </exclusion>
    </exclusions>
    <version>7.6.0.Final</version>
</dependency>
</dependencies>
<build>
    <finalName>${project.artifactId}</finalName>
    <resources>
        <resource>
            <directory>src/main/java</directory>
            <includes>
                <include>**/*.xml</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
        <resource>
            <directory>src/main/resources</directory>
            <includes>
                <include>**/*.*</include>
            </includes>
            <filtering>>false</filtering>
        </resource>
    </resources>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.3.2</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>

```

第二步：创建/resources/application.yml文件

```
server:
  port: 8080
spring:
  application:
    name: insuranceInfoCheck
```

第三步：创建实体类InsuranceInfo

```
package com.itheima.drools.entity;

/**
 * 保险信息
 */
public class InsuranceInfo {
    private String param1;//保险公司
    private String param2;//方案代码
    private String param3;//渠道号
    private String param4;//销售区域
    private String param5;//投保年龄
    private String param6;//保险期间
    private String param7;//缴费期间
    private String param8;//缴费方式
    private String param9;//保障类型
    private String param10;//等待期
    private String param11;//犹豫期
    private String param12;//职业类型
    private String param13;//保额限制
    private String param14;//免赔额
    private String param15;//主险保额
    private String param16;//主险保费
    private String param17;//附加险保额
    private String param18;//附加险保费
    private String param19;//与投保人关系
    private String param20;//与被保人关系
    private String param21;//性别
    private String param22;//证件
    private String param23;//保费
    private String param24;//保额

    //getter setter省略
}
```

第四步：创建决策表文件（也可以使用实战资料中提供的insuranceInfoCheck.xls文件）

| | RuleTable insurance-rules | | | | | | |
|--------|----------------------------------|-----------------------------|--|---|---|-----------------------------|-----|
| | CONDITION | CONDITION | CONDITION | CONDITION | CONDITION | CONDITION | |
| | \$insuranceRules : InsuranceInfo | | | | | | |
| | param1 != \$1 | param4 != \$1,param4 != \$2 | Integer.parseInt(param5) < \$1 Integer.parseInt(param5) > \$2 | param6 != \$1,param6 != \$2,param6 != \$3 | param7 != \$1,param7 != \$2,param7 != \$3,param7 != \$4,param7 != \$5,param7 != \$6 | param8 != \$1,param8 != \$2 | |
| 规则 | 保险公司Code | 销售区域 | 投保年龄 | 保险期间 | 交费期间 | 交费方式：1:趸交 2:年交 | 保险期 |
| 校验保险公司 | "picc" | | | | | | |
| 校验销售区域 | | "北京","天津" | | | | | |
| 校验投保年龄 | | | 0, 17 | | | | |
| 校验保险期间 | | | | 20, 25, 30 | | | |
| 校验交费期间 | | | | | 1, 3, 5, 10, 15, 20 | | |
| 校验交费方式 | | | | | | 1, 2 | |

第五步：封装工具类KieSessionUtils

```
package com.itheima.drools.utils;

import com.itheima.drools.entity.InsuranceInfo;
import com.itheima.drools.entity.PersonInfoEntity;
import org.drools.decisiontable.InputType;
import org.drools.decisiontable.SpreadsheetCompiler;
import org.kie.api.builder.Message;
import org.kie.api.builder.Results;
import org.kie.api.io.ResourceType;
import org.kie.api.runtime.KieSession;
import org.kie.internal.utils.KieHelper;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.List;

public class KieSessionUtils {
    private KieSessionUtils() {

    }
    // 把xls文件解析为String
    public static String getDRL (String realPath) throws
FileNotFoundException {
        File file = new File(realPath); // 例如: C:\\abc.xls
        InputStream is = new FileInputStream(file);
        SpreadsheetCompiler compiler = new SpreadsheetCompiler();
        String drl = compiler.compile(is, InputType.XLS);
        System.out.println(drl);
        return drl;
    }

    // drl为含有内容的字符串
    public static KieSession createKieSessionFromDRL(String drl) throws
Exception{
        KieHelper kieHelper = new KieHelper();
        kieHelper.addContent(drl, ResourceType.DRL);
    }
}
```

```

        Results results = kieHelper.verify();
        if (results.hasMessages(Message.Level.WARNING,
Message.Level.ERROR)) {
            List<Message> messages =
results.getMessages(Message.Level.WARNING, Message.Level.ERROR);
            for (Message message : messages) {
                System.out.println("Error: "+message.getText());
            }
            // throw new IllegalStateException("Compilation errors were
found. Check the logs.");
        }
        return kieHelper.build().newKieSession();
    }

    // realPath为Excel文件绝对路径
    public static KieSession getKieSessionFromXLS(String realPath) throws
Exception {
        return createKieSessionFromDRL(getDRL(realPath));
    }
}

```

第六步：创建RuleService类

```

package com.itheima.drools.service;

import com.itheima.drools.entity.InsuranceInfo;
import com.itheima.drools.utils.KieSessionUtils;
import org.kie.api.runtime.KieSession;
import org.springframework.stereotype.Service;
import java.util.ArrayList;
import java.util.List;

@Service
public class RuleService {
    public List<String> insuranceInfoCheck(InsuranceInfo insuranceInfo)
throws Exception{
        KieSession session =
KieSessionUtils.getKieSessionFromXLS("D:\\rules.xls");
        session.getAgenda().getAgendaGroup("sign").setFocus();

        session.insert(insuranceInfo);

        List<String> listRules = new ArrayList<>();
        session.setGlobal("listRules", listRules);

        session.fireAllRules();

        return listRules;
    }
}

```

第七步：创建RuleController类

```
package com.itheima.drools.controller;

import com.itheima.drools.entity.InsuranceInfo;
import com.itheima.drools.service.RuleService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/rule")
public class RuleController {
    @Autowired
    private RuleService ruleService;

    @RequestMapping("/insuranceInfoCheck")
    public Map insuranceInfoCheck(){
        Map map = new HashMap();

        //模拟数据，实际应为页面传递过来
        InsuranceInfo insuranceInfo = new InsuranceInfo();
        insuranceInfo.setParam1("picc");
        insuranceInfo.setParam4("上海");
        insuranceInfo.setParam5("101");
        insuranceInfo.setParam6("12");
        insuranceInfo.setParam7("222");
        insuranceInfo.setParam8("1");
        insuranceInfo.setParam13("3");

        try {
            List<String> list =
ruleService.insuranceInfoCheck(insuranceInfo);
            if(list != null && list.size() > 0){
                map.put("checkResult", false);
                map.put("msg", "准入失败");
                map.put("detail", list);
            }else{
                map.put("checkResult", true);
                map.put("msg", "准入成功");
            }
            return map;
        } catch (Exception e) {
            e.printStackTrace();
            map.put("checkResult", false);
            map.put("msg", "未知错误");
            return map;
        }
    }
}
```

```
}
```

第八步：创建启动类DroolsApplication

```
package com.itheima.drools;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DroolsApplication {
    public static void main(String[] args) {
        SpringApplication.run(DroolsApplication.class);
    }
}
```