# 告警动态策略配置模块

### 一：需求

在告警系统中，不同的情况需要区分不同的告警，针对性的做出响应，也就是针对告警的响应策略。

而策略在不同情况下会有变动，相对来说是比较频繁的，所以需要一个模块（或系统）针对这种情况方便动态的更换策略

模块需求：在告警系统中可以实现动态的告警策略修改

### 二：用户

目标用户是业务人员，默认为不会编程语言，专业领域知识丰富

业务人员在此模块（系统）上的操作是对策略进行维护

模块（系统）增加约束：告警策略的定制必须是易于业务人员理解的

### 三：方案

针对上面的需求，有两种实现方式：

1，使用规则引擎 有很多开源或商业规则引擎可供选择

2，使用脚本，JVM上可用的脚本很多，groovy python javascript都有jvm版本

可维护性

规则引擎：中，相对于if-else的代码，将业务逻辑从代码中剥离出来，实现可配置的策略，更改策略不在需要开发人员参与。但是规则引擎本身比较复杂，对于开发人员来说需要更多的学习和对引擎的理解。

脚本：高，和规则引擎一样，同样将业务逻辑从代码中剥离出来，实现可配置的策略。但是脚本本身并不复杂，但是需要开发人员学习脚本语言

可靠性

规则引擎：高，成熟的架构，已经在很多项目中使用，完善的社区支持

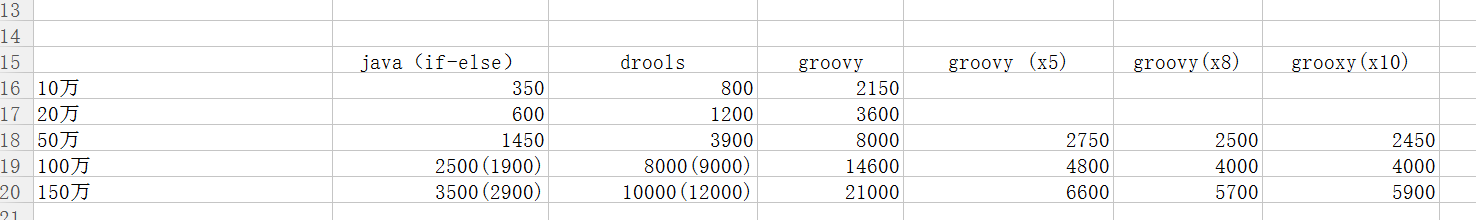
脚本：依赖于项目本身

性能对比

环境



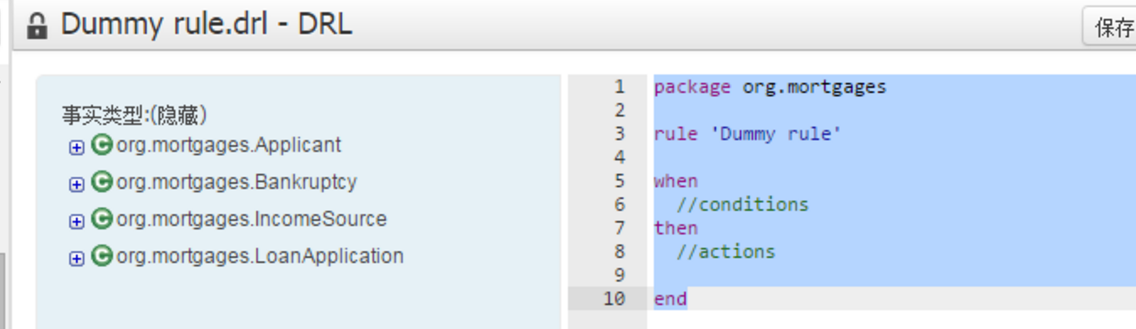
JIRA要求的demo，alarmData处理。



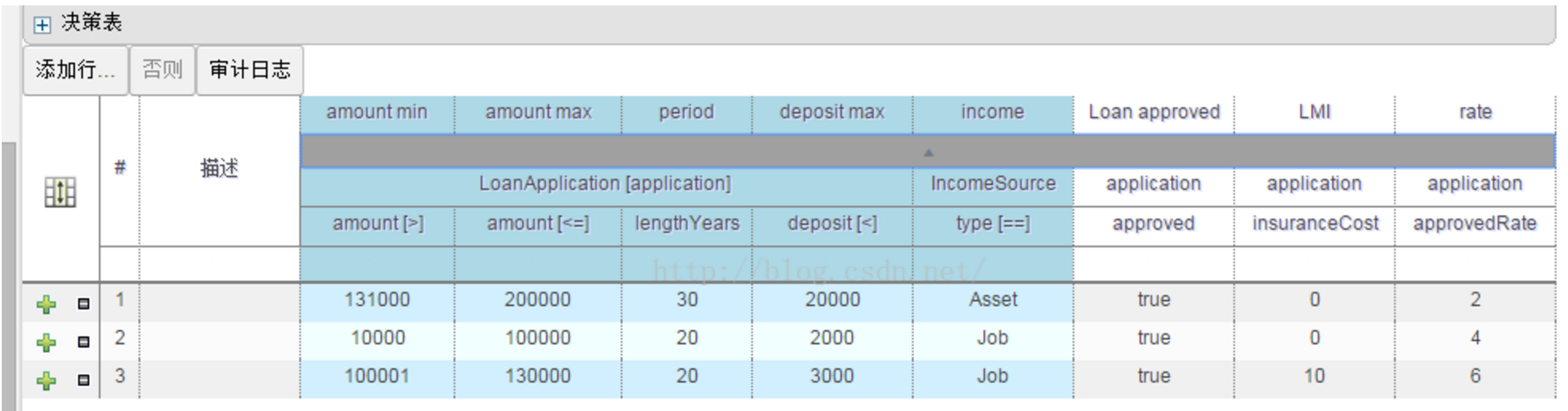
规则引擎

优点1：成熟的全套的解决方案，开源规则引擎drools有配套的drools workbench可以提供多种策略配置方式

drl



决策表



配置向导



也可以自己实现DSL的配置向导，比如下面的规则



和上面的相比，这种方式由业务人员提出语法，和开发人员沟通协商后，得到最终的结果，学习成本更低且更符合业务本身的描述

优点2：对于drools这样广泛使用的规则引擎，无论中文还是英文社区都有较多的资料，遇到问题可以方便的咨询社区

优点3：规则匹配算法，机制。规则引擎内置了一系列的功能，方便了开发人员

缺点1：drl本身对业务人员并不友好，仍然需要二次开发

缺点2：性能，动态配置规则有一定的性能损耗

缺点3：针对性能问题，drools的优化方式需要对drools本身有很深的了解

脚本：

优点1：DSL的定义比规则引擎更灵活

优点2：性能稍好，优化方式上可以使用java常用优化方式，如对多线程的优化

优点3：轻量级，简单，容易理解

缺点1：需要根据自身业务设计DSL（相比于开源引擎的社区），需要对脚本语言有较深的理解

缺点2：功能缺乏，需要自己实现

缺点3：需要自己实现规则匹配（根据业务决定）