****



**研 究 生 毕 业 论 文**

**（申请工程硕士学位）**

|  |  |
| --- | --- |
| **论文题目** | 饿了么体验保障平台的  设计与实现 |
| **作者姓名** | 金思晔 |
| **学科、专业名称** | 工程硕士(软件工程方向) |
| **研究方向** | 软件工程 |
| **指导教师** | 王金庆 高级工程师 |

**2019 年 4 月 日**

**学 号： MF1832073**

**论文答辩日期： 2020 年 月 日**

**指 导 教 师： （签字）**

**饿了么体验保障平台的**

**设计与实现**

|  |  |
| --- | --- |
| **作 者:** | **金思晔** |
| 指导教师: | **王金庆　高级工程师** |

|  |
| --- |
| **南京大学研究生毕业论文** |
| **(申请工程硕士学位)** |

|  |
| --- |
| **南京大学软件学院** |
| **2019年4月** |

**The Design and Implementation of**

**Experience Guarantee Platform for ELEME**

**Jin, Si Ye**

**Submitted in partial fulfillment of the requirements for the degree of Master of Engineering**

Supervised by

Senior Engineer **Wang, Jinqing**

Software Institute

**NANJING UNIVERSITY**

Nanjing, China

April, 2019

# 摘 要

# Abstract

# 图目录

# 表目录

目录

[摘 要 V](#_Toc25947911)

[Abstract VI](#_Toc25947912)

[第一章 引言 8](#_Toc25947913)

[1.1 项目背景 8](#_Toc25947914)

[1.2 国内外发展现状 9](#_Toc25947915)

[1.3 本文主要工作 9](#_Toc25947916)

[1.4 本文的组织结构 9](#_Toc25947917)

[第二章 技术综述 9](#_Toc25947918)

[2.1 SpringBoot 9](#_Toc25947919)

[2.2 MaxQ 9](#_Toc25947920)

[2.3 Huskar 10](#_Toc25947921)

[2.4 WorkFlow 10](#_Toc25947922)

[2.5 Redis 10](#_Toc25947923)

[2.6 本章小结 10](#_Toc25947924)

[第三章 系统分析与设计 11](#_Toc25947925)

[3.1 系统总体规划 11](#_Toc25947926)

[3.2 系统需求分析 11](#_Toc25947927)

[3.2.1 规则管理模块 11](#_Toc25947928)

[3.2.2 审核流模块 12](#_Toc25947929)

[3.2.3 业务处理模块 13](#_Toc25947930)

[3.3 非功能性需求 15](#_Toc25947931)

[3.5 数据库设计 15](#_Toc25947932)

[3.6 本章小结 15](#_Toc25947933)

[第四章 系统实现 15](#_Toc25947934)

[4.1 规则配置模块 15](#_Toc25947935)

[4.2 审核流模块 16](#_Toc25947936)

[4.3 业务处理模块 17](#_Toc25947937)

[4.5 系统测试 17](#_Toc25947938)

[4.5.1 测试环境 17](#_Toc25947939)

[4.5.2 测试设计 17](#_Toc25947940)

[4.5.3 测试结果 17](#_Toc25947941)

[第五章 总结与展望 17](#_Toc25947942)

[5.1 总结 17](#_Toc25947943)

[5.2 工作展望 17](#_Toc25947944)

[参 考 文 献 18](#_Toc25947945)

[致 谢 18](#_Toc25947946)

# 第一章 引言

## 1.1 项目背景

论文选题源自于本人实习时参与的“饿了么热斧平台”。热斧平台负责三部分功能：超时规则与处理策略的制定、判定订单是否超时以及超时之后的处理。

在电商平台中，超时场景是指订单或者运单未在指定时间内达到某种状态。对超时场景的处理是服务类平台不可或缺的能力，饿了么、美团、淘宝等都有对超时场景的判定和相应的补偿操作。对于不同的超时场景，有不同的处理策略，例如淘宝用户下单后三十分钟未支付，那么系统会取消这笔订单，并释放之前锁定的库存；饿了么用户点外卖后，骑手没有在规定时间内将外卖送到指定地点，那么用户就会收到系统赔付的优惠券等。

饿了么超时场景的判定分散在各个服务当中，例如逆向系统、赔付系统、用户触达系统等。每次对超时规则的修改，产品经理都需要各个系统依次去同步，额外的增加了产品经理的工作量。目前的规则是硬编码在代码中，如果需要修改规则，需要产品提紧急需求，开发人员出技术方案，各方评审方案，开发人员开发，测试人员测试，然后在发布窗口发布。对于开发人员来说，增加了高优先级的任务，需要优先完成紧急需求，增加了开发人员的工作量；对于业务方来说，需求得不到快速响应，延误了策略的实施。由于上述的这些问题，运营人员无法做到对规则的细粒度把控：例如，根据天气配置规则，根据城市配置规则，根据订单来源配置规则等，每次规则变动的需求都会造成牵一发而动全身的影响。

为了解决上述问题，饿了么需要将分散在各个系统中的超时场景判定逻辑抽离出来，集中在一起进行判定，同时需要有一个配置平台，能够对超时规则以及处理逻辑进行配置，以达到快速响应业务需求的目的。为此，本人参与设计并实现了“饿了么热斧平台”，以满足上述要求。

## 1.2 国内外发展现状

当系统越来越庞大，业务逻辑越来越复杂，开发的速度无法适应业务变更速度的时候，将需要经常需要变更的业务逻辑可配置化是一种趋势，常见于营销平台，风控平台，银行系统等。

陌陌风控平台是由陌陌公司开发的一款规则可配置化的风控，风控平台需要处理的风险包括：活动作弊、薅羊毛行为、外挂、刷量等。当应用出现BUG或者业务逻辑漏洞时候，风控平台需要及时的进行响应，将存在的风险降到最低，这就要求风控平台能够快速修改风控规则。陌陌风控平台分为陌陌风控平台规则细分有以下四种：名单型策略，布尔型策略，时段频控型策略，限用户数型策略。名单型策略用于管理用户的黑白名单；布尔型策略用于判断当前用户行为是否符合规则描述；时段频控型策略用户限制单位时间内指定事件出现的次数；限用户数型用于限制用户数量。用户可以在平台上分别进行设置，最终再按照指定顺序进行编排。风控规则的可配置化保证了拦截风险的性。

旗正规则引擎是由杭州旗正信息技术股份有限公司独立开发的一款规则引擎，是一个可视化定制业务逻辑的商业规则管理系统。旗正规则引擎课可以在程序外部对软件项目中涉及到的业务逻辑进行单独管理，并且提供多种语言的API接口供外部程序调用。该系统没有业务场景限制，而是作为一个规则解析平台，为上游系统提供规则解析服务。

Drools是一款使用JAVA语言开发的开放源码规则引擎，使用Rete算法对所编写的规则求值。用户可以通过使用xml文件编写规则，可以使用Drools的.drl文件进行规则编写，也可以直接只用JAVA代码硬编码在业务中。Drools由四个部分组成：Drools Guvnor--业务规则管理系统、Drools Expert—规则引擎、Drools Flow—工作流引擎、Drools Fusion—事件处理。Drools更加适用于需要匹配属性较多的情况。

## 1.3 本文主要工作

本文的目标是做一个能够快速响应业务方需求、对规则粒度自由把控、判定是否命中规则的系统。通过研究当前已有的规则引擎，阅读github上优秀的开源代码，结合生产过程中遇到的问题，提出相应的解决方案。

综上所诉，本文的主要工作有：

1. 分析了当前系统的现状和存在问题。
2. 详细分析了系统的各类需求，并确定系统的总体架构和数据库解构。
3. 根据需求分析和总体设计，完成了模块的划分，并实现各个模块。

## 1.4 本文的组织结构

# 第二章 技术综述

## 2.1 SpringBoot

SpringBoot是由Pivotal团队在2013年开始研发、2014年4月发布第一个版本的全新开源的轻量级框架。它不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用和整个搭建和开发的过程。SpringBoot框架中有两个非常重要的策略：开箱即用和约定优于配置。开箱即用，是指在开发过程中，通过在Maven项目的pom文件中添加相关依赖包，然后使用对应的注解来代替繁琐的XML配置文件以管理对象的生命周期，这个特点使得开发人员摆脱了复杂的配置工作以及依赖管理工作，更加专注于业务逻辑。约定优于配置，是一种由SpringBoot本身来配置目标结构，由开发者在结构中添加信息的软件设计范式。这一特点虽降低了部分灵活性，增加了BUG定位的复杂性，但减少了开发人员需要做出决定的数量，同时减少了大量XML配置，并且可以将代码编译、测试、打包等工作自动化。

## 2.2 RabbitMq

RabbitMq是基于AMQP协议开发的一款开源的消息中间件，AMQP协议又称为高级消息队列协议，是应用层协议的一个开放标准。AMQP的主要特征是面向消息、队列、路由、可靠性、安全。RabbitMq的服务端使用Erlang语言编写，客户端支持Python、Ruby、.Net、Java等语言。

RabbitMq包含以下基本组件：

1. Producer：消息生产者，负责将消息发送到消息队列中
2. Consumer：消息消费者，负责从消息队列中接受消息
3. Message：服务器和应用程序之间传递是数据，由properties和Body组成
4. Exchange：交换机，接受消息，并根据消息将其发送到对应的绑定消息队列上
5. Queue：又称为Message Queue，保存Exchange发过来的消息，并将其发送给消费者。

消息生产者连接到RabbitMQ服务器上，指定对应的Exchange名称，之后消息生产者向exchange中发送消息，exchange接收到消息后，根据routing key路由到跟当前交换机绑定的相匹配的队列里面，消费者监听接收到消息之后开始业务处理，然后发送一个ack确认告知消息已经被消费，RabbitMQ服务器收到ack之后将对应的消息从消息队列里面是删掉。

## 2.3 WorkFlow

工作流本质是一个状态机，由多个状态组成。状态机可以从一个状态扭转到另一个状态。状态机从一个起始状态开始执行，到一个中介状态结束，状态直接使用JSON传递数据。状态非为多种不同的类型，不同的类型的状态有不同的功能，类型节点如下：

1. Success：工作流的出口，表示当前工作流执行成功
2. Fail：工作流的出口，表示当前工作流执行失败
3. Pass：过度状态，可以对上个状态的输出做某些处理，并将处理结果流转到下一个状态
4. Task：工作流的远程调用状态，访问一个远程接口，获取结果
5. Wait：等待状态，是当前工作流等待一段时间或等待到某个固定的时间点
6. Choice：判断状态，提供逻辑判断操作

工作流系统主要有三个部分组成：控制台、触发器和工作流引擎。控制台用于管理工作流的元数据、触发器以及一些配置信息，同时也是http流量的入口；触发器用于定时触发和监听队列触发工作流的执行；工作流引擎是整个工作流执行的核心，根据预定义的工作流，使用数据驱动状态的流转。

## 2.5 Redis

Redis是一个高效的key-value存储系统，与传统关系型数据库不同的是，它以键值对的方式存储数据，支持存储的value类型相对较多，包括String类型，list类型，set类型、zset类型和hash类型，这些数据类型都支持push/pop、add/remove及取交集、并集和差集等丰富的操作，而这些操作都是原子性的。为了保证Redis的高效，Redis中的数据都是存放在内存中，同时，Redis通过nio的方式实现了单线程处理请求，因此Redis有如下的应用场景：

* 缓存：缓存是Redis最重要的一个应用场景，合理的利用缓存，不仅能够提升网站访问的速度，还能大大降低数据库的压力。
* 排行榜：Redis提供的zset数据类型能够实现各种复杂的排行榜应用
* 分布式锁：分布式技术是当前互联网常用的技术，分布式技术面临的问题是对同一个资源的并发访问，当并发量不大的时候，可以通过数据库的悲观锁、乐观锁来实现，但是在并发量高的情况下，利用数据库锁来控制资源的并发访问是不理想的，因此需要使用Redis来实现一个分布式锁，进行更高效的访问。
* 分布式会话：在集群模式下，session的管理独立于应用容器，一般会搭建以Redis为中心的session服务。

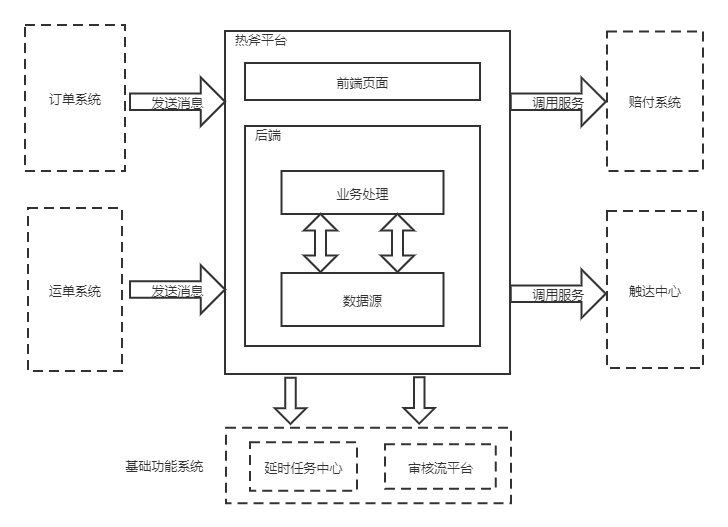
## 2.6 本章小结

##

# 第三章 系统分析与设计

## 3.1 系统边界

热斧平台是公司交易部门当中的一部分，热斧平台将原有分散在各个系统中的规则判断逻辑抽离出来集中在一起处理，因此，热斧平台与订单系统、运单系统、赔付系统、触达系统、审核流系统、延时任务系统等都会有交互。业务描述如下：用户可以在热斧平台上进行规则的编辑，完成编辑之后，平台会将编辑结果提交审核，审核通过的规则会被纳入规则库中。平台依赖于订单和运单的状态变更，每次状态的变更，平台都会和规则库中的规则进行匹配，并根据匹配结果做相应的处理。



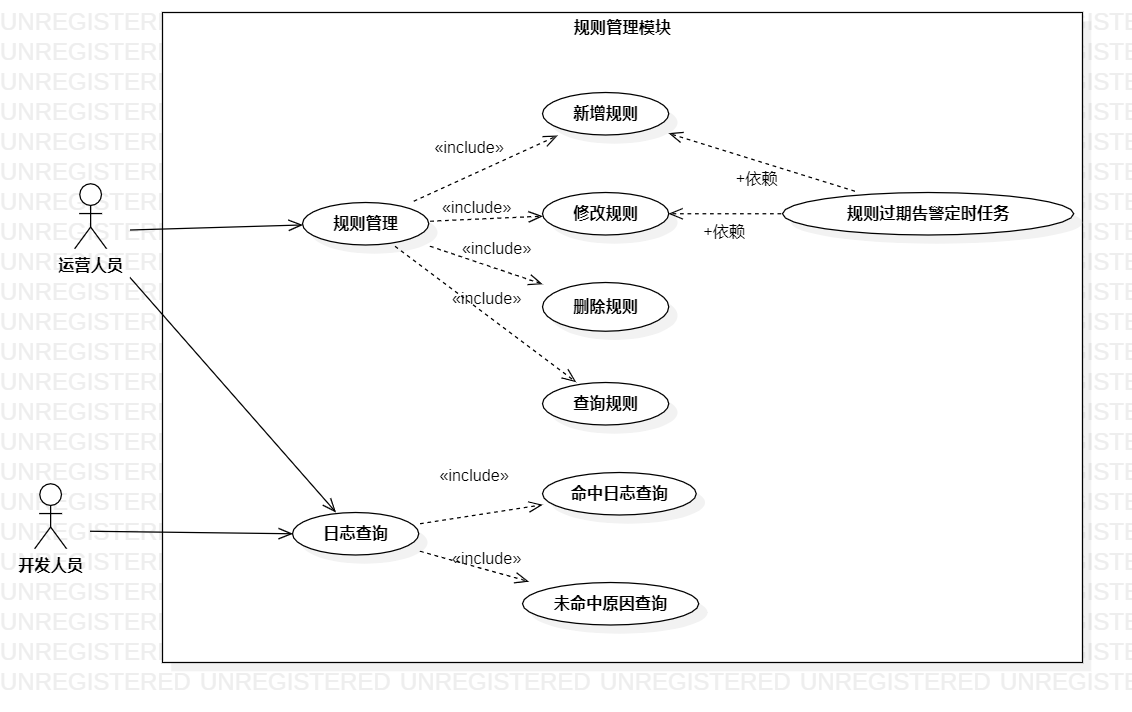
图xxx为系统边界图，其中订单系统和运单系统是热斧平台的上游，订单和运单状态的改变在这两个系统中进行，热斧平台作为中间平台，需要监听订单和运单状态发生变化时的发送的消息，并根据消息进行响应的处理；审核流系统和延时任务系统为基础功能系统，用户可以在前端页面上进行超时规则以及处理策略的配置，为了减少因员工失误而导致的配置错误，配置完成的规则需要进行审核，因此需要使用审核平台的基础能力，规则在判定的时候会存在延时判定的情况，因此需要使用延时任务平台的基础能力；赔付中心与触达中心是热斧平台的下游，也是原有规则判定与策略执行的平台，命中规则之后策略执行由热斧平台控制，因此赔付中心与触达中心需要提供基础能力。

## 3.2 系统需求分析

基于3.1章节中系统的总体规划，结合热斧系统的实际使用场景，系统需要满足以下的需求：

1. 提供规则管理能力，运营人员在前端页面上能够对规则进行新增，修改，删除，查询等操作，并且能够看到规则命中和规则未命中的情况。
2. 提供规则预处理能力，将前端配置好的规则转换成系统能够执行的规则。
3. 提供规则审核流管理的能力，运营人员能够管理规则，包括查找，终止审核中的规则，系统需要在当前审核节点通过的时候，进行审核人的扭转，并且保证消息丢失的情况下能够继续审核。
4. 提供业务处理能力，系统需要监听订单、运单消息，消费这些消息的同时需要与规则库中的规则进行匹配，并且对匹配后的结果进行下一步的处理。

### 3.2.1 规则管理功能



规则管理用例图如图所示，规则管理模块包含了规则管理和日志查询两个模块。

规则管理子模块中，运营人员可以对规则进行操作，以快速响应业务需求，包括对规则的新增、删除、修改、查询。在新增和修改规则的时候，会创建告警定时任务，当规则有效期快过期的时候，告警任务就会发送邮件提醒用户规则快过期，让运营人员提前做处理。

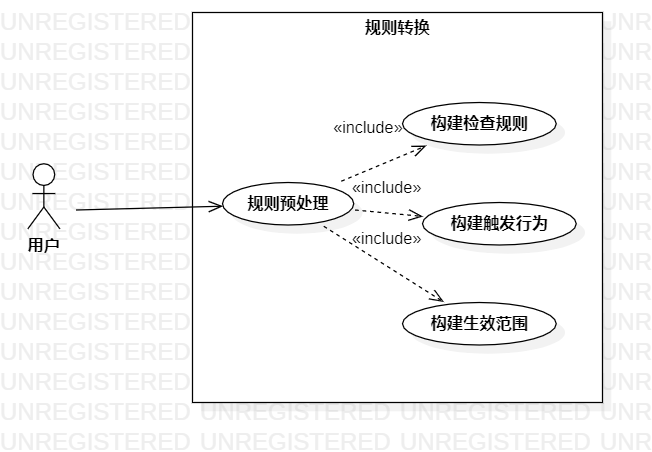
日志查询子模块中，运营人员根据命中日志和未命中原因可以优化当前的规则；开发人员可以查询未命中原因进行故障的排查。

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC1 | 名称 | 操作规则 |
| 描述 | 运用在配置页面上进行规则配置和查看 | | |
| 参与者 | 运营 | | |
| 前置条件 | 用户已经经过身份验证 | | |
| 后置条件 | 触发创建告警任务、触发审核流 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户点击新增规则。 2. 用户进行规则的配置。 3. 用户点击提交审核 4. 系统保存规则草稿，并触发审核流，触发创建告警任务。 | | |
| 拓展流程 | 无 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC2 | 名称 | 创建超时告警任务 |
| 描述 | 创建规则超时告警任务，定时发送告警邮件 | | |
| 参与者 | 系统 | | |
| 前置条件 | 用户修改/创建规则 | | |
| 后置条件 | 无 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户修改/创建规则。 2. 规则传送到后端 3. 后端生成邮件 4. 延时任务中心创建任务。 | | |
| 拓展流程 | 无 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC3 | 名称 | 查询日志 |
| 描述 | 运营和开发人员通过平台查看规则命中日志 | | |
| 参与者 | 运营人员、开发人员 | | |
| 前置条件 | 用户通过权限校验 | | |
| 后置条件 | 无 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户点击日志选项卡 2. 选择日志筛选条件 3. 点击查询按钮 | | |
| 拓展流程 | 无 | | |

### 规则预处理功能



规则按照功能可以分为三大类：超时判定规则、规则生效范围、和规则触发行为。用户配置完成的规则是以对象的方式传到后端系统，后端系统需要把存储在对象中的规则转换成能够执行的规则-以树的方式进行组织，树的每个节点代表着一个操作，其叶子节点是操作需要的参数：

如上图所示，AND节点代表一个操作，其叶子节点是AND节点的两个参数，

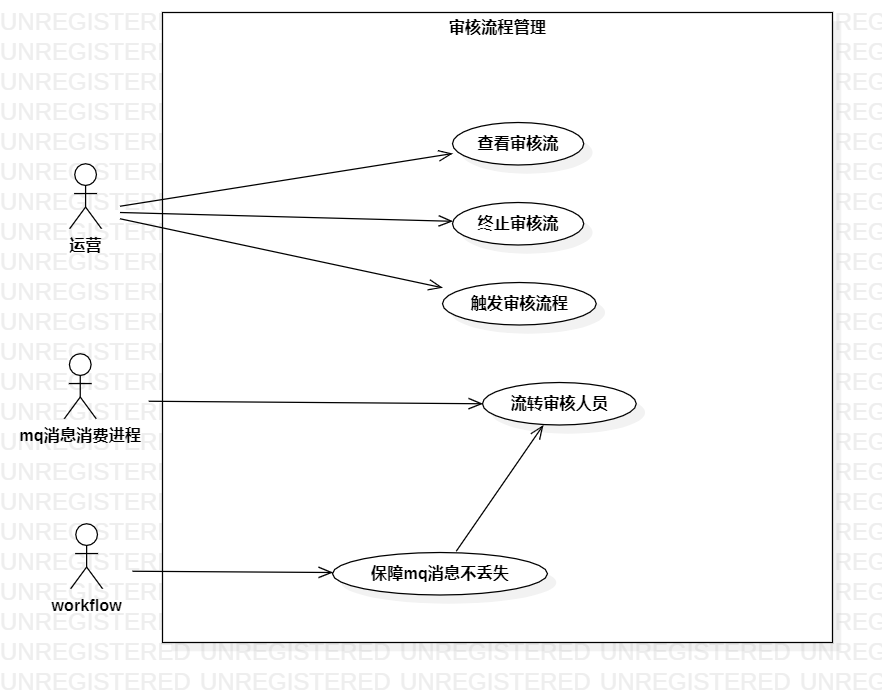
这个操作的含义就是输出AND节点两个值的逻辑与值。作为参数的节点可以是一个常量，也可以是另外的一个操作。因此将存储在对象中的规则按照这样的方式组织，并对该树进行后续遍历，这样就可以达到执行规则的效果。

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC4 | 名称 | 构建检查规则 |
| 描述 | 根据用户配置的检查规则构建能够真实执行的规则 | | |
| 参与者 | 热斧系统 | | |
| 前置条件 | 用户创建/修改规则 | | |
| 后置条件 |  | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户提交规则创建/修改 2. 后端根据检查规则构建规则执行树 | | |
| 拓展流程 | 无 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC5 | 名称 | 构建触发行为 |
| 描述 | 运营和开发人员通过平台查看规则命中日志 | | |
| 参与者 | 热斧系统 | | |
| 前置条件 | 用户创建/修改规则 | | |
| 后置条件 | 无 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户提交规则创建/修改 2. 后端根据触发行为构建动作执行树 | | |
| 拓展流程 | 无 | | |

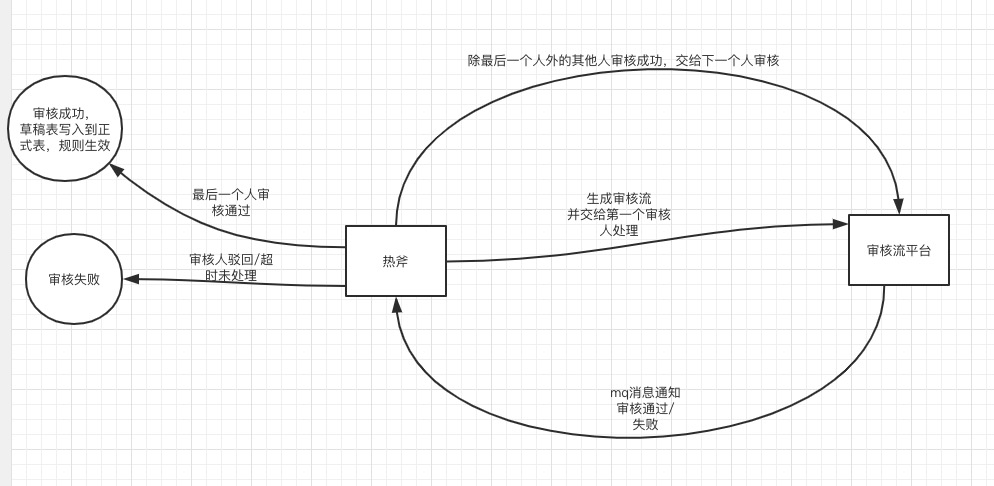
|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC6 | 名称 | 构建生效范围 |
| 描述 | 运营和开发人员通过平台查看规则命中日志 | | |
| 参与者 | 热斧系统 | | |
| 前置条件 | 用户创建/修改规则 | | |
| 后置条件 | 无 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户提交规则创建/修改 2. 后端根据生效范围构建生效范围执行树 | | |
| 拓展流程 | 无 | | |

### 审核流程管理功能



上图是审核流程管理的用例图。运营人员在创建、修改规则之后，就会触发审核流程，系统会在审核平台上创建一个审核流程。运营人员可以根据指定的条件查询审核流当前的状态，根据需求的变化，终止某一条规则的审核。

系统借助于审核平台完成规则的审核，因此系统需要监听审核平台的消息队列，审核流的某个节点审核通过之后，审核平台会向消息队列中发送审核通过的消息，系统接受到消息后，会将审核节点流转到下一个节点，直到审核通过或者审核被拒。



上图为审核流转图

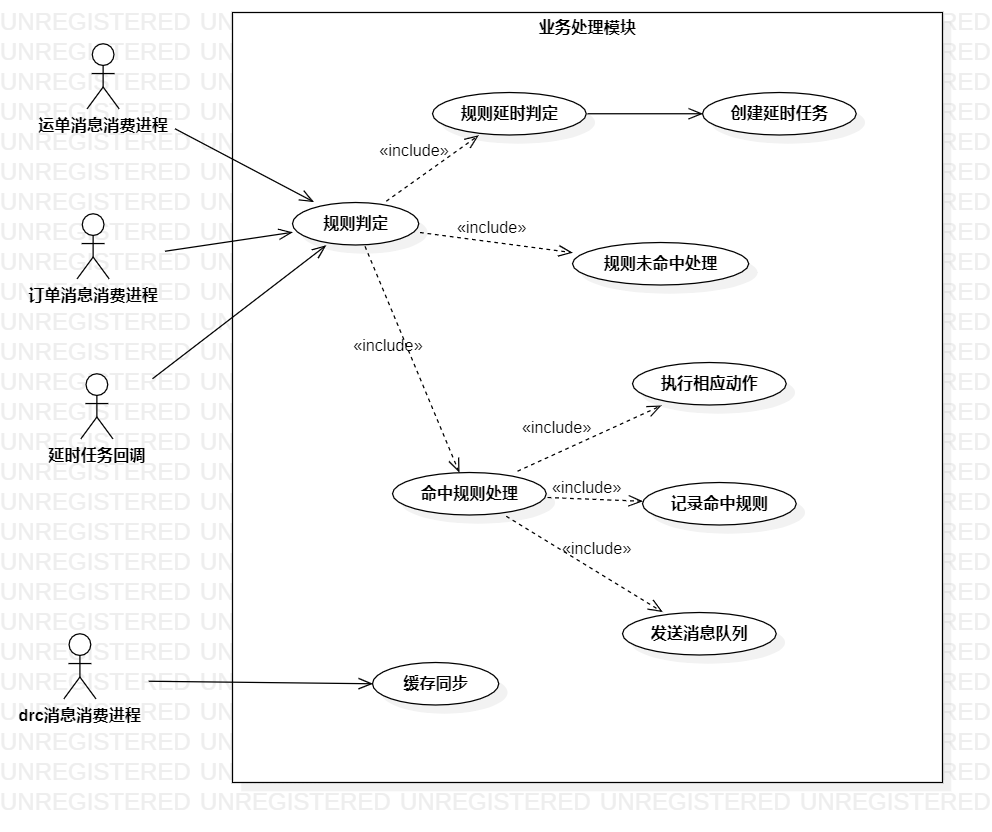
审核流程的流转监听消息队列来实现，消息队列的消息存在丢失的情况，如果审核人员审核通过了，但是通过的消息却丢失了，该审核流将无法流转，最终会停止在某个审核节点上。因此需要在workflow上配置任务，定时检测消息是否丢失。

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC7 | 名称 | 操作审核流 |
| 描述 | 对审核流进行操作，包括查看、终止审核流 | | |
| 参与者 | 运营 | | |
| 前置条件 | 运营配置好规则，并提交审核 | | |
| 后置条件 |  | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户切换至审核流选项卡 2. 选择审核流筛选规则 3. 点击查询 4. 点击终止审核六终止规则的审核 | | |
| 拓展流程 |  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC8 | 名称 | 扭转审核流 |
| 描述 | 运营修改规则 | | |
| 参与者 | 热斧系统 | | |
| 前置条件 | 用户创建审核流 | | |
| 后置条件 | 无 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 监听审核流消息队列 2. 消息到达时解析审核流消息 3. 根据审核流消息终止、成功、扭转审核流 | | |
| 拓展流程 | 无 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC9 | 名称 | 保证审核流 |
| 描述 | 保证消息丢失之后审核流程能够扭转下去 | | |
| 参与者 | 热斧系统、workflow系统 | | |
| 前置条件 | 审核流消息丢失 | | |
| 后置条件 |  | | |
| 优先级 | 中 | | |
| 正常流程 | 1. workflow中配置定时任务和回调接口 2. 比对审核流平台和热斧平台审核记录 3. 将热斧平台审核记录更新为审核流平台数据 | | |
| 拓展流程 | 无 | | |

### 业务处理功能



上图为业务处理功能的用例图，当订单、运单状态改变的时候，就会往消息队列中发送状态改变消息，系统监听到消息后，会根据已有的规则进行判定，判定结果分为三种：命中规则、未命中规则和延迟判定。命中规则后需要执行后续的Action，对命中规则进行记录，同时发送命中消息到消息队列；未命中谷规则需要记录未命中的原因；延时处理需要在延时中心创建延时任务，等到达指定时候之后再去执行。

drc消息是异地多活数据同步消息。为了防止一个机房的机子全部宕机，系统一般会在不同的机房里部署，以两个机房为例，用户一次只会影响一个机房的数据，当用户影响a机房的数据的时候，a机房就会给b机房的消息队列发送一条数据同步消息，b机房接收到消息后，就会将b机房的数据就会和a机房的数据进行同步，保证最终一致性，b机房同理。机房之间发送的同步消息就是drc消息。为了避免频繁的访问数据库，数据库中的规则需要缓存到redis中，当数据库中的数据发生变化的时候，redis中的数据也需要同步，监听drc消息为了同步机房的redis消息。

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC10 | 名称 | 修改规则 |
| 描述 | 运营修改规则 | | |
| 参与者 | 运营 | | |
| 前置条件 | 用户已经经过身份验证 | | |
| 后置条件 | 触发创建告警任务、触发审核流 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户点击修改规则。 2. 用户进行规则的配置。 3. 用户点击提交审核 4. 系统保存规则草稿，并触发审核流，触发创建告警任务。 | | |
| 拓展流程 | 无 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC11 | 名称 | 修改规则 |
| 描述 | 运营修改规则 | | |
| 参与者 | 运营 | | |
| 前置条件 | 用户已经经过身份验证 | | |
| 后置条件 | 触发创建告警任务、触发审核流 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户点击修改规则。 2. 用户进行规则的配置。 3. 用户点击提交审核 4. 系统保存规则草稿，并触发审核流，触发创建告警任务。 | | |
| 拓展流程 | 无 | | |

|  |  |  |  |
| --- | --- | --- | --- |
| ID | UC12 | 名称 | 修改规则 |
| 描述 | 运营修改规则 | | |
| 参与者 | 运营 | | |
| 前置条件 | 用户已经经过身份验证 | | |
| 后置条件 | 触发创建告警任务、触发审核流 | | |
| 优先级 | 中 | | |
| 正常流程 | 1. 用户点击修改规则。 2. 用户进行规则的配置。 3. 用户点击提交审核 4. 系统保存规则草稿，并触发审核流，触发创建告警任务。 | | |
| 拓展流程 | 无 | | |

## 3.3 规则设计

运营人员在进行规则配置的时候并非是随意的配置，而是需要在指定的规则集合下进行配置。

规则主要分为三大部分，超时判定规则、规则生效范围和规则触发动作。

在规则触发条件中，规则的定义如下：

|  |  |
| --- | --- |
| 规则名 | 可选值 |
| 订单类型在指定范围 | 及时单 |
| 预定单 |
| 全部 |
| 主场景在指定范围 | 提交订单 |
| 实际用户T |
| 预计时长 |
| 用户付款 |
| 商家接单 |
| 商家呼单 |
| 骑手接单 |
| 骑手取餐 |
| 骑手送达 |
| 业务阶段（开始节点/结束节点）在指定范围 | 提交订单 |
| 实际用户T |
| 预计时长 |
| 用户付款 |
| 商家接单 |
| 商家呼单 |
| 骑手接单 |
| 骑手取餐 |
| 骑手送达 |
| 未发生事件在指定范围 | 商家接单 |
| 商家呼单 |
| 骑手接单 |
| 骑手取餐 |
| 骑手送达 |
| 时间比较符号 | + |
| - |
| > |
| < |
| = |

表xxx为触发规则，在上述规则中，每一个规则对应着一个最小执行单元，超时判定规则由这些最小执行单元组成，每条判定规则都需要判断订单的类型，订单当前所处的业务阶段，在开始阶段之后或之后的指定时间内，是否发生指定事件。规则的形式相对固定，因此可以通过树的结构将上述最小执行单元进行组织。而执行单元内部是可以变动的，其变动范围如上表中所示，用户可以在前端页面上进行配置，以达到动态修改规则的目的。

在规则生效范围中，规则定义如下

|  |  |
| --- | --- |
| 规则名 | 可选值 |
| 城市区域 | 代理商城市（全国2844个三级行政区） |
| 直营城市（（全国2844个三级行政区）） |
| 订单来源 | PC端web |
| 手机端web |
| Iphone App |
| Android App |
| openapi\_platform |
| 微信小程序 |
| 微信KA |
| 物流类型 | 蜂鸟专送 |
| 众包 |
| 择时达专送 |
| 新零售专送 |
| 蜂鸟快送 |
| 新零售 |
| E配送 |
| 蜂鸟混合送 |
| 蜂鸟质选 |
| 全城送 |
| 蜂鸟独家 |
| 业务类型 | 餐饮 |
| 新零售 |
| 全选 |

表xxx为规则生效范围，每一个规则对应着一个最小执行单元，与触发规则不同的是，生效范围当中的规则非必选，生效范围可以由上述规则中的0个到全部组成。与触发规则相同的是，规则也是以树的结构组织。

规则触发动作

|  |  |  |
| --- | --- | --- |
| 规则名 | 可选值 | |
| 用户触达 | 触达类型 | 短信 |
| 消息推送 |
| 接受者 | 用户 |
| 客户 |
| 短信模板 | |
| 备注说明 | |
| 生效范围 | |
| 用户取消 | 是否支持 | 支持 |
| 不支持 |
| 生效范围 | |
| 用户赔付 | 赔付类型 | 红包 |
| 优惠券 |
| 积分 |
| 触发时间点 | 订单取消或完成时 |
| 场景触发时 |
| 赔付规则名称 | |
| 渠道号 | |
| 生效范围 | |
| 商户索赔 | 是否支持 | 支持 |
| 不支持 |
| 生效范围 | |

函数设计

|  |  |  |
| --- | --- | --- |
| 函数名 | 函数介绍 | 参数介绍 |
| AND | 逻辑与 | 两个参数，返回两个参数的与值 |
| OR | 逻辑或 | 两个参数，返回两个参数的或值 |
| ENV | 从上下文中获取指定对象，并从对象中获取指定值。 | 两个参数，对象的key和值的key。 |
| IN | 数值在指定集合中 | N个参数，第一个参数为key，后续参数为指定结合 |
| LT | 小于 | 两个参数， |
| LE | 小于等于 | 两个参数 |
| GT | 大于 | 两个参数 |
| GE | 大于等于 | 两个参数 |
| ADD | 加 | 两个参数 |
| SUBTRACT | 减 | 两个参数 |
| MULTIPLICATION | 乘 | 两个参数 |
| TRIGGER\_AT | 触发时间 | 两个参数 |
| ORDER\_AND\_DELIVERY\_STATUS | 订单和运单状态 | 两个参数 |
| DELIVERY\_STATUS\_DOUBLE\_CHECK | 运单装填双重检测 | List集合 |

订单消息

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| Type | String | 消息代表的含义 |
| Description | String | 具体说明 |
| Paload.order\_id | Long | 订单id |
| Paload.role | Integer | 消息发起角色 |
| Paload.restaurant\_id | Integer | 餐馆id |
| Paload.user\_id | Integer | 下单用户id |
| Paload.come\_from | Integer | 订单来源 |
| Paload.is\_book | Integer | 预订单 |
| Paload.sent\_at\_millsec | Long | 发送时间戳 |
| Paload.order\_mode | Integer | 订单模型 |

运单消息

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| Type | String | 消息代表的含义 |
| Description | String | 具体说明 |
| Payload.carrier\_driver\_id | String | 骑手id |
| Payload.carrier\_id | String | 配送商id |
| Payload.deliverer\_name | String | 骑手名称 |
| Payload.deliverer\_phone | String | 骑手电话 |
| Payload.order\_id | Long | 订单id |
| Payload.send\_at | Long | 发送时间 |
| Payload.state | String | 运单主状态 |
| Payload.subState | String | 运单子状态 |
| Payload.tracking\_id | Long | 运单id |
| Payload.update\_time | Long | 更新时间 |

超时消息

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| Type | Integer | 消息代表的含义 |
| Event\_name | String | 事件名称 |
| Payload.order\_id | Long | 订单id |
| Payload.timeout | Long | 超时时间 |
| Payload.start\_time | Long | 开始时间 |

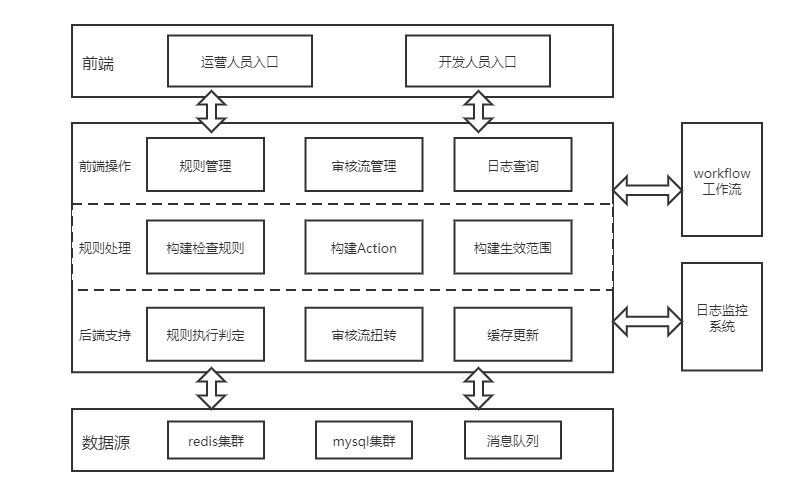
审批流消息

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| Apply\_instance\_id | Long | 审批实例id |
| Previous\_state | String | 上一个状态 |
| Current\_state | String | 当前状态 |
| Aaudit\_log\_id | Long | 审核操作ID |
| Event\_type | EventType | 事件类型 |
| Parameters | String | 参数 |
| Operator | Operator | 操作人 |

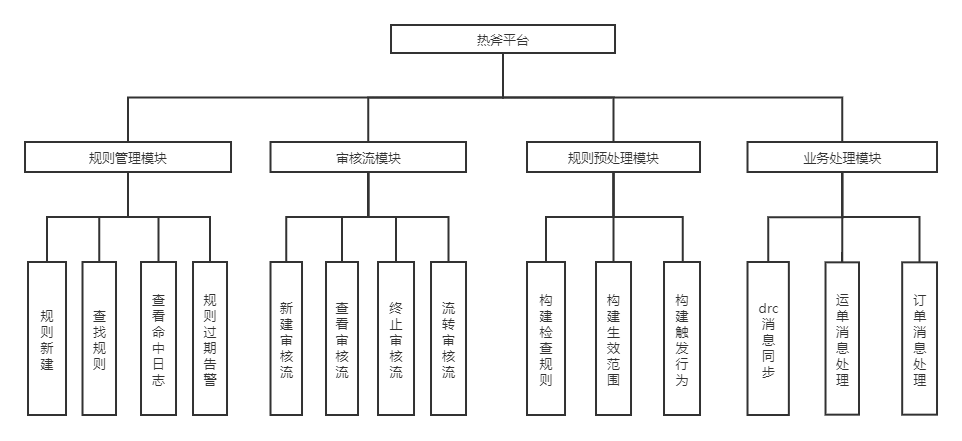
## 3.4 系统总体设计

### 3.4.1 工作原理

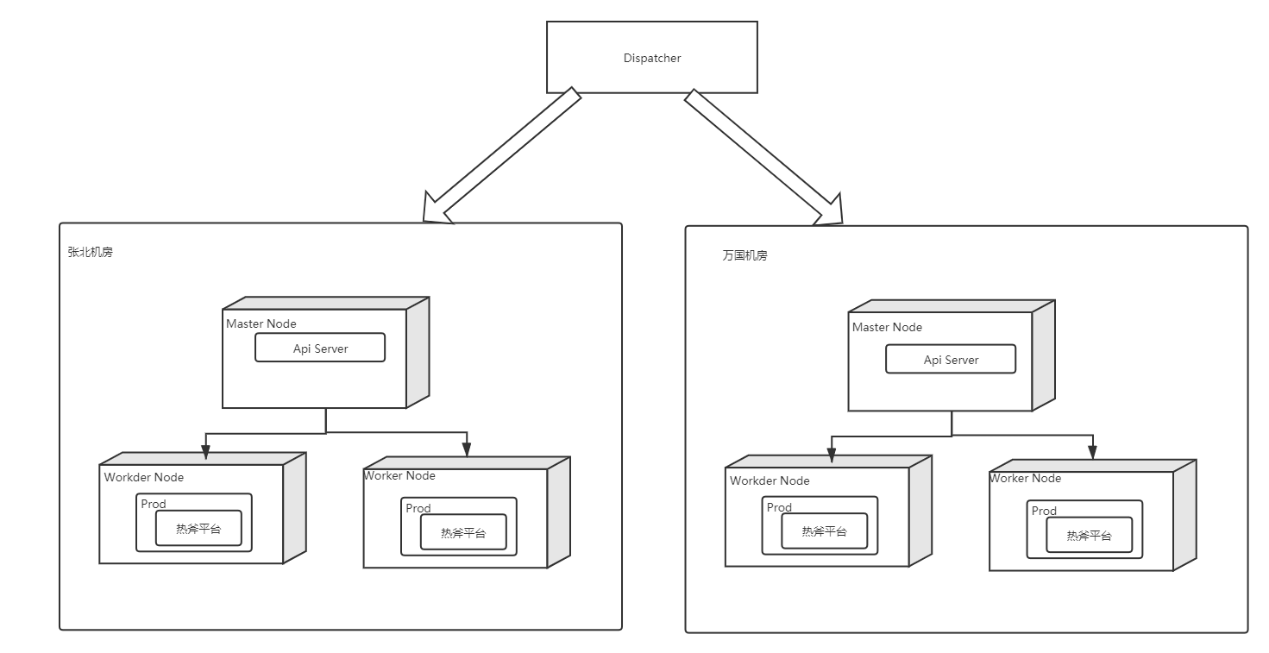
### 3.4.2 架构设计



### 3.4.3 功能组成设计



### 3.4.4 部署设计



## 3.5 数据库设计

规则表：

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| id | Long | 规则id |
| type | int | 场景类型 |
| description | int | 场景码 |
| Trade\_node | Varchar | 主场景 |
| Start\_trade\_node | Varchar | 业务开始节点 |
| End\_trade\_node | Varchar | 业务结束节点 |
| Trigger\_event | Varchar | 触发事件 |
| Status | Int | 规则状态 |
| Start\_time | Long | 开始时间 |
| End\_time | Long | 结束时间 |
| Check\_rules | Text | 校验规则 |
| Actions | Text | 触发行为 |

规则草稿表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| id | Long | 规则id |
| type | int | 场景类型 |
| description | int | 场景码 |
| Trade\_node | Varchar | 主场景 |
| Start\_trade\_node | Varchar | 业务开始节点 |
| End\_trade\_node | Varchar | 业务结束节点 |
| Trigger\_event | Varchar | 触发事件 |
| Status | Int | 规则状态 |
| Start\_time | Long | 开始时间 |
| End\_time | Long | 结束时间 |
| Check\_rules | Text | 校验规则 |
| Actions | Text | 触发行为 |

订单命中规则表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| Id | Long | Id |
| Order\_id | Long | 订单id |
| Scene\_id | Long | 场景id |
| Scene\_code | Int | 场景code |
| Occure\_time | Long | 出现时间 |
| Action\_result | Text | 场景触发的action结果 |
| timeout | Long | 超时时间 |
| Delay | Long | 延迟回调时间 |

场景规则动作表

|  |  |  |
| --- | --- | --- |
| 字段名 | 类型 | 描述 |
| id | Long | Id |
| Record\_id | Long | 触发记录id |
| Status | Long | 场景动作状态 |
| Params | Text | 场景动作参数 |
| Results | Text | 场景动作结果 |
| Creator | Varchar | 创建者 |
| updator | Varchar | 更新者 |
| Type | Int | 动作类型 |
| Order\_id | Long | 订单号 |
| trigger\_event | Varchar | 触发事件 |
| Rule\_id | Long | 规则id |
| Name | Varchar | 动作名称 |

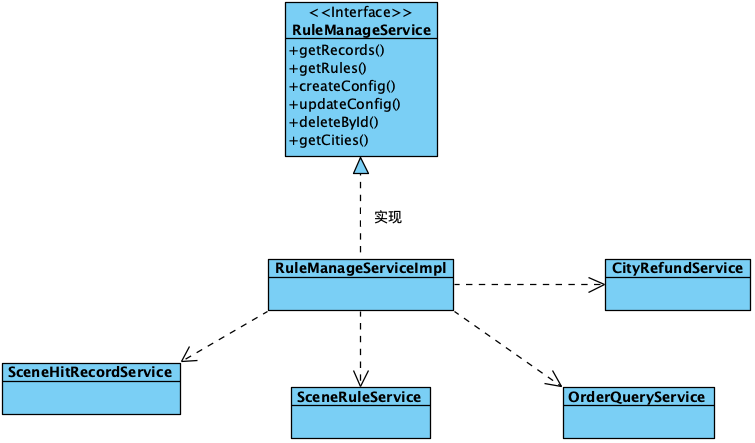
## 3.6 本章小结

##

# 第四章 系统实现

## 4.1 规则管理模块

规则管理模块的功能主要有对已有规则的新增、删除、修改、查询等，同时，为了方便运营人员进行命中与未命中情况的统计，方便开发人员排查线上问题，对命中规则和未命中规则的订单做了相应的统计。

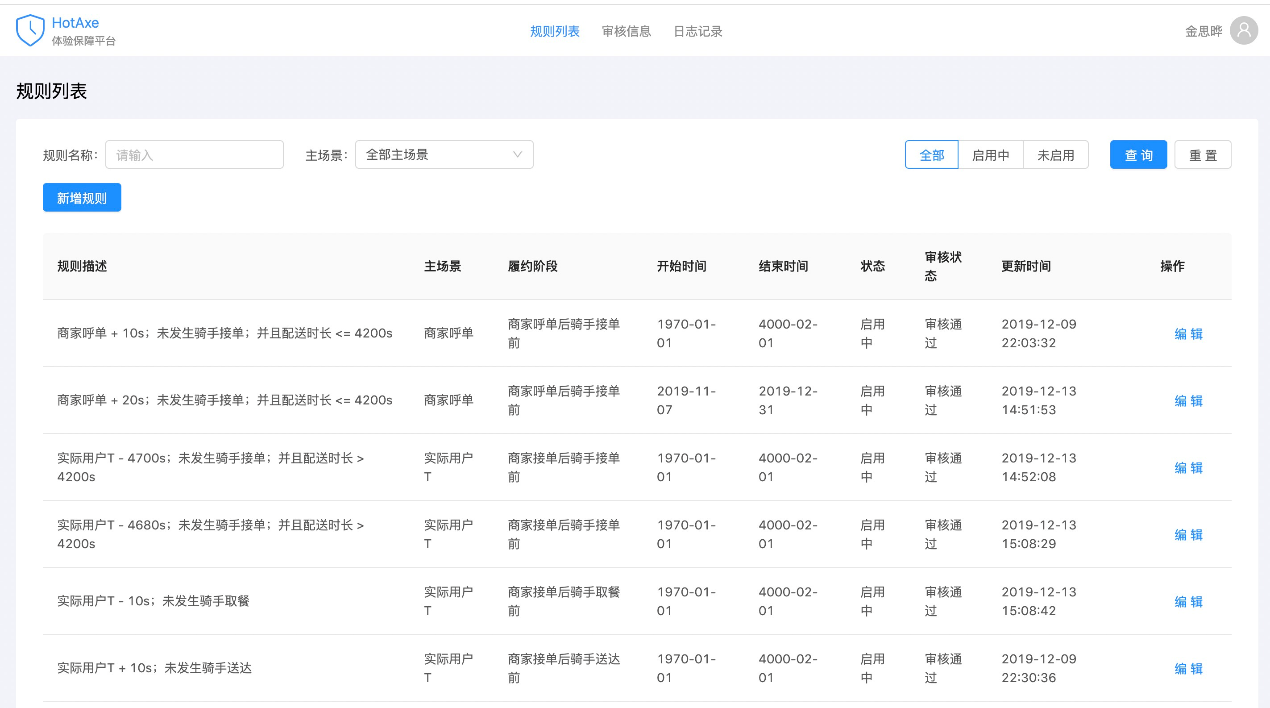


图xxx为规则管理的设计类图，RuleManageServiceImpl负责规则的增删改查以及命中情况的查询。为了实现规则的操纵，RuleMangeServiceImpl类主要通过调用SecneRuleService实现对象当中的方法；为了实现规则命中日志的查询，RuleMangeServiceImpl通过调用SceneHitRecordService实现类当中的方法。除此之外，前端展示的数据，例如节点的可选类型，全国城市信息等通过调用CityRefundService实现类的方法来实现。

规则管理模块的主要功能是对规则的创建、修改、删除、查询和对命中规则日志的查询，以新增规则为例，其实现代码图如下所示。方法createConfig需要传入一个RuleContent对象，该对象中保存用户创建的规则，方法拿到规则后，会对规则进行校验，以确保规则符合要求。校验通过之后调用SceneRuleService的create方法，并根据返回参数判断是否创建成功。而在SceneRuleService的create方法中，需要做如下几件事情：将规则转换成后端能够执行的执行树，创建规则审核流以及在数据库中保存规则的草稿。



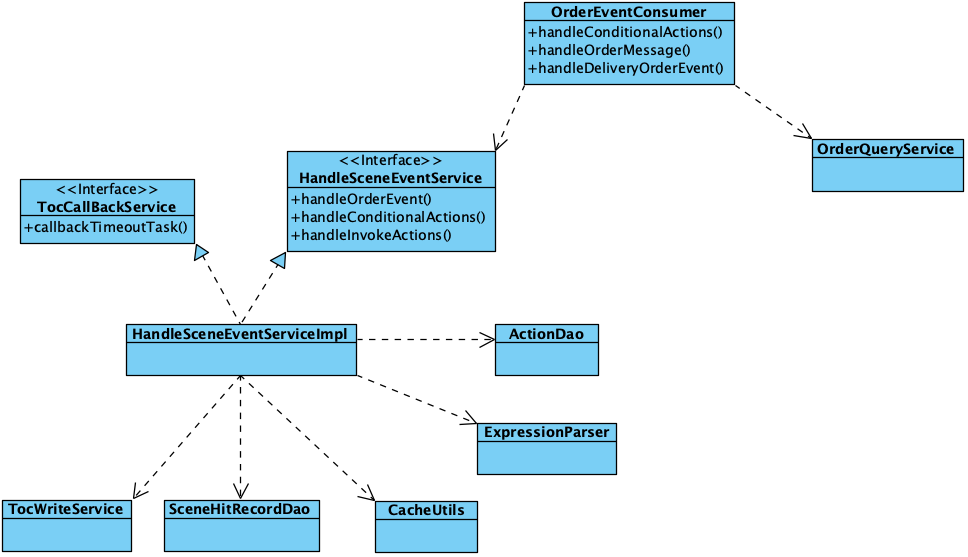
上图为命中规则记录查询的代码，在进行查询之前，需要对查询条件进行校验，包括查询条件不能为空，查询的订单id不能为空，查询时候的偏移量不能超过指定值等。校验完成之后再调用SceneHitRecordService的getHitRecord方法。





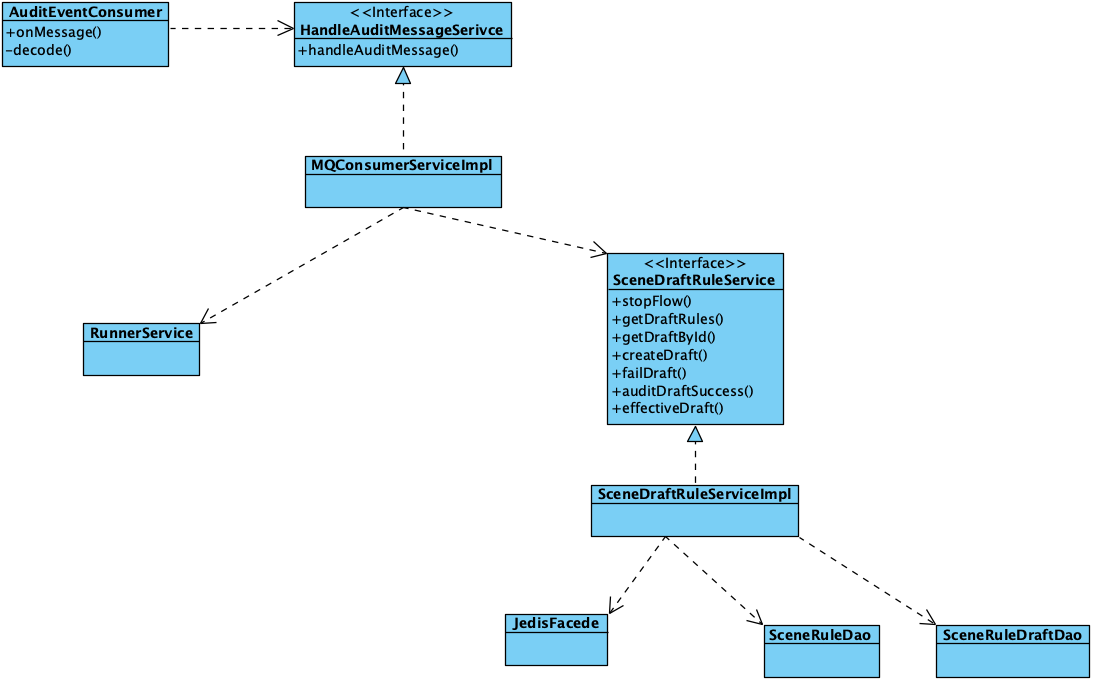


## 4.2 规则预处理模块



## 审核流模块

审核流模块是对新增或修改后规则合理性的一种保障，防止出现因用户操作失误而导致配置出错误的规则，从而直接导致公司的资损。审核流模块的审批是借用的公司已有的审核流平台，在开发过程中，审核流模块需要接入审核流平台，两者之间的交互通过消息队列的方式进行，用户新增或者修改规则之后，就会在审核流平台创建一个审核流，审核人员会在审核流上进行审批，审批的结果会发送到消息队列中，系统审核流模块会监听消息，并根据消息的内容做相应的处理。除此之外，模块还需要保证消息的可靠性， 为此在任务调度平台上创建调度任务，调用审核流模块提供的接口，来保证消息的不丢失。



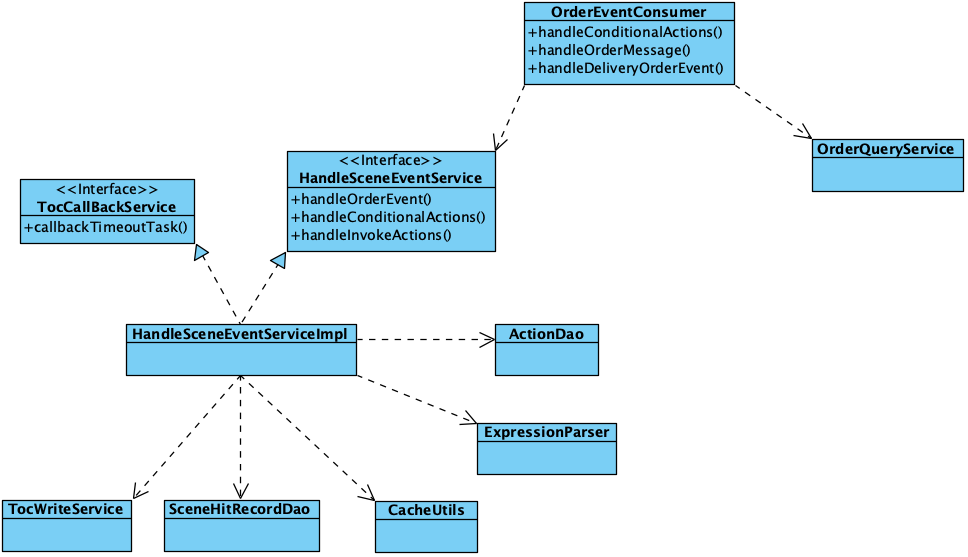
图xxx为审核流的设计类图，AuditEventConsumer是一个消息消费者类，消费从消息队列发送过来，消息监听线程会调用AuditEventConsumer的onMessage方法。对于接受到的消息，onMessage方法内会调用decode()方法消息进行解码。消息的处理接口是HandleAuditMessageService的handleAuditMessage()方法。HandleAuditMessageService的实现类中引用了两个对象：RunnerService和SceneDraftRuleService。RunnerService负责进行流程的流转，当获取到消息队列中的消息时，消息会携带当前是第几个节点，审批的情况，根据这两个信息可以推断出下一个需要审批的人，此时可以调用RunnerService的transfer方法，将审核节点流转到下一个节点。SceneDraftRuleService接口主要是对规则草稿进行操作。操作包括创建规则草稿，更新规则审核信息，获取规则草稿等操作。当审批通过的时候，需要调用effectiveDraft方法，该方法会将规则插入到规则数据库中，并且将缓存中的数据失效。



图xxx为审核流流转代码，



## 4.4 业务处理模块



业务处理模块负责判断当前订单、运单是否发生超时情况，并根据已有规则进行相应的处理。图xxx为业务处理模块流程图

## 4.5 系统测试

系统的测试有两个目标，一方面需要测试系统是否满足需求，另一方面需要找出系统中存在的问题，本章将从测试环境，测试设计以及测试结果三方面进行阐述。

### 4.5.1 测试环境

测试环境基于内部私有云，表xxx为系统测试环境硬件配置，集群使用了2台服务器，服务器的硬件配置为每台8核CPU、32G内存、128G固态硬盘

|  |  |
| --- | --- |
| 硬件项 | 备注 |
| CPU | 8核 |
| 内存 | 32G |
| 硬盘 | 1T |
| 数量 | 两台 |

表xxx为系统测试环境软件配置，每台服务器均安装kubernetes用于集群的管理，使用Docker运行服务应用，数据库使用Mysql。

|  |  |
| --- | --- |
| 软件名称 |  |
| 操作系统 | Centos7.2 |
| 集群管理 | Kubernetes |
| 容器 | Docker |
| 数据库 | Mysql |

### 4.5.2 测试设计

根据第三章的需求分析，本节将对系统的主要功能进行测试，包括规则管理功能、规则预处理功能、审核流功能以及业务处理功能 。通过对上述主要功能的测试，确保系统的功能能够正确正常的运行。

表xxx为规则管理功能的测试用例

|  |  |
| --- | --- |
| 测试ID | TC1 |
| 测试名称 | 规则管理测试 |
| 测试描述 | 测试规则管理功能，包括规则的增删改查，规则命中日志的查看 |
| 测试步骤 | 1. 用户进入规则管理页面，查看已有规则 2. 点击新增按钮，新增规则，点击提交审核按钮 3. 点击修改按钮，对已有规则进行修改，再点击提交审核按钮 4. 点击删除按钮，删除已有规则 5. 点击查看规则命中日志按钮 6. 点击查看未命中原因按钮 |
| 预期结果 | 1. 显示已有规则 2. 新建规则成功，已发送审核 3. 修改规则成功，已发送审核 4. 删除规则成功 5. 显示命中规则的订单 6. 显示订单未命中规则的原因 |

表xxx为审核流功能的测试用例。

|  |  |
| --- | --- |
| 测试ID | TC2 |
| 测试名称 | 审核流功能测试 |
| 测试描述 | 测试审核流的触发、审核流的触发、审核流节点的流转、审核流的终止，以及审核流 |
| 测试步骤 | 1. 用户修改规则 2. 用户新增规则 3. 用户点击进入审核流界面 4. 用户点击终止审核流 5. 审批人员审批审核流 |
| 预期结果 | 1. 触发创建审核流 2. 触发创建审核流 3. 查看已经存在的审核流 4. 审核流被终止 5. 审核通过/审核未通过/流转到下个节点 |

表xxx为规则预处理功能的测试用例

|  |  |
| --- | --- |
| 测试ID | TC3 |
| 测试名称 | 规则预处理测试 |
| 测试描述 | 测试前端规则能够转换成后端执行需要的模式，并且能够正确的执行 |
| 测试步骤 | 1. 用户新建规则，并点击提交 2. 用户修改规则，并点击提交 |
| 预期结果 | 1. 规则转换成功 2. 规则转换成功 |

表xxx为业务处理功能的测试用例

|  |  |
| --- | --- |
| 测试ID | TC4 |
| 测试名称 | 业务功能测试 |
| 测试描述 | 测试业务处理功能是否达到要求 |
| 测试步骤 | 1. 用户下单 |
| 预期结果 | 1. 命中规则 2. 未命中规则 |

### 4.5.3 测试结果

##

# 第五章 总结与展望

## 5.1 总结

##

## 5.2 工作展望

##

# 参 考 文 献

##

# 致 谢

##