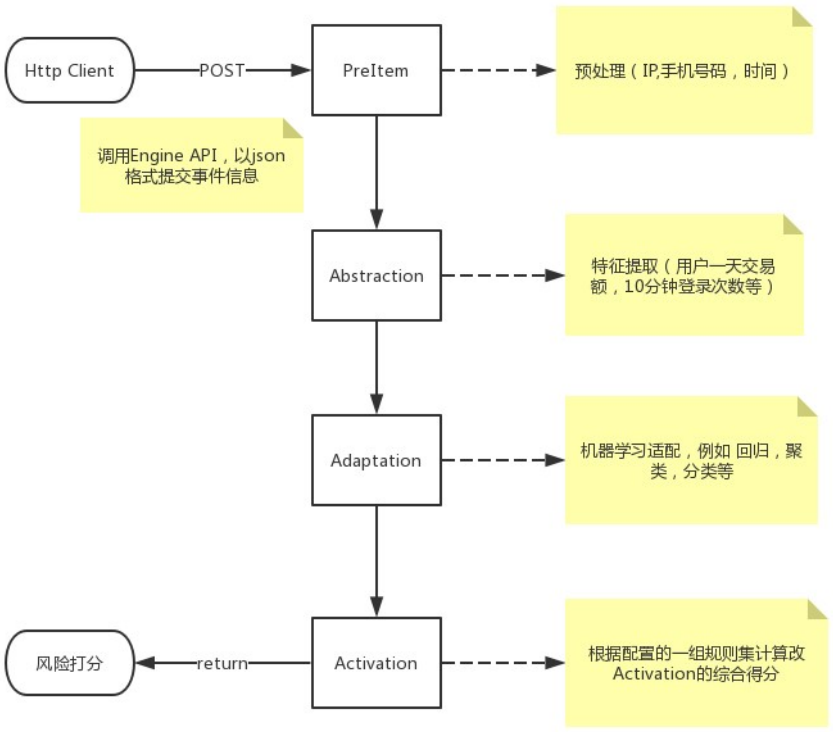
目录

home

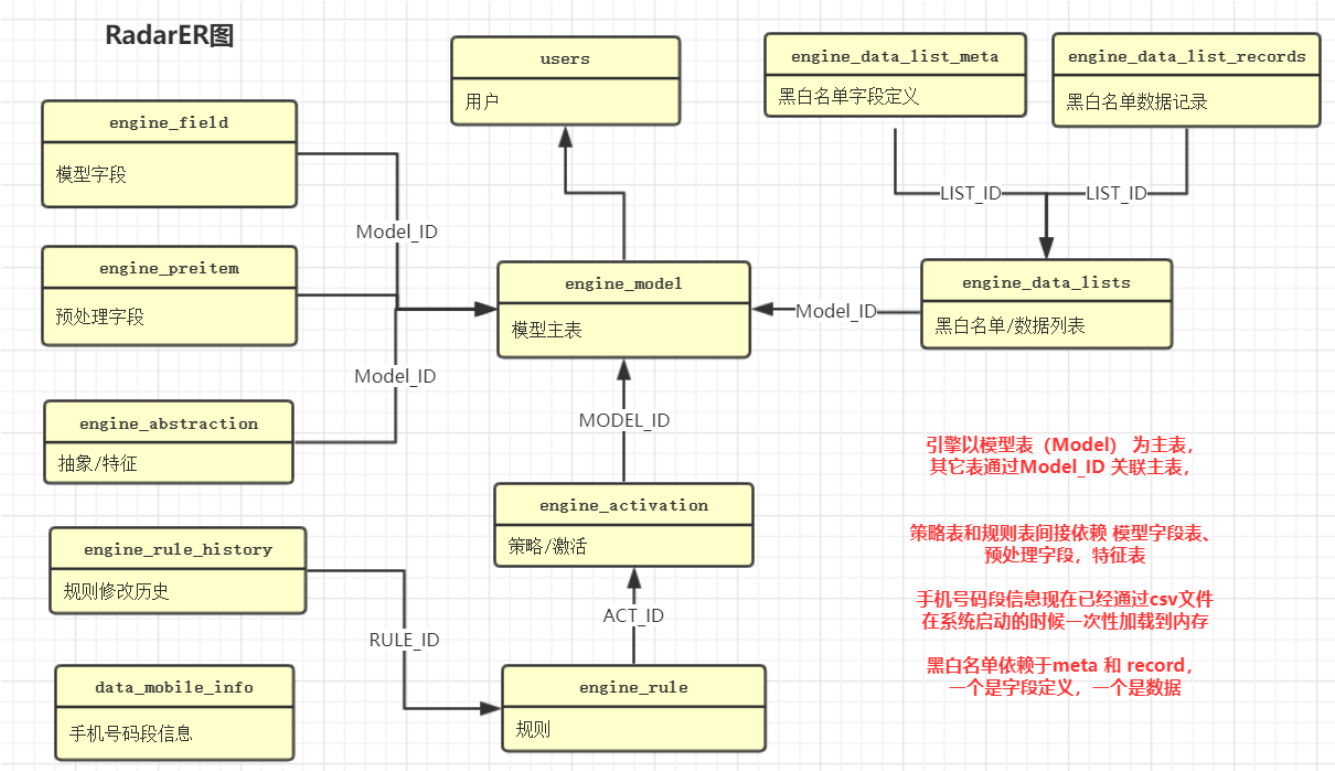
介绍

一款基于java语言，使用Springboot + Mongodb + Groovy + ES 等框架搭建的轻量级实时风控引擎，适用于反欺诈应用场景，开箱即用。

引擎处理流程



数据ER关系图



Demo数据

以ting提现为例：

序号	模型名	实体名	事件ID	唯一标识	事件时间
1	登录行为模板	userId	eventId	查看	loginTime
2	注册行为模板	userId	eventId	查看	registerTime
3	交易行为模板	userId	eventId	查看	eventTime
4	Ting提现	userId	eventId	查看	eventTime

数据样例：

```
modelGuid : DB8A078F-97FE-4A7F-AAC0-5AF1A6C36CE8
reqId: 100005
jsonInfo: {
      "eventId":"100005", "mobile":"18516249909", "userId":"18516249909",
      "eventTime":1576035291628, "userIP":"180.175.166.148",
      "deviceId": "SDKSKDSLD-ASDFA-32348235", "os":"ios","amount":500.0,
      "channel":"alipay"
      }
```

响应样例：

```
{
   "success": true,
   "msg": "",
   "code": "100",
   "data": {},
   "abstractions": { // 特征 指标的计算结果，方便大家调试和核对
      "tran_did_1_day_qty": 2,
      "tran_did_ip_1_day_qty": 0,
      "tran_uid_ip_1_day_qty": 1,
      "tran_uid_1_hour_amt": 1000.0,
```

```
      "tran_ip_1_hour_qty": 2,
      "tran_ip_1_day_amt": 1000.0,
      "tran_ip_10_min_amt": 1000.0,
      "tran_ip_1_hour_amt": 1000.0,
      "tran_uid_1_day_amt": 1000.0,
      "tran_did_10_min_qty": 2,
      "tran_did_1_hour_qty": 2,
      "tran_ip_1_day_qty": 2,
      "tran_ip_10_min_qty": 2,
      "tran_uid_10_min_amt": 1000.0
   },
   "adaptations": null,
   "activations": {
      "transaction_exception": {
         "risk": "reject", // 风险级别：pass 通过， reject 拒绝， review 人工审核
         "score": 120  //风险积分
      }
   },
   "hitsDetail": { // 具体命中的规则项
      "transaction_exception": [
         {
            "key": "381",
            "value": 120.0,
            "desc": "1天内IP交易金额大于1000"
         }
      ]
   },
   "respTimes": { // 各模块耗时情况
      "adaptations": 0,
      "activations": 2,
      "abstractions": 27
   }
}
```


使用手册

https://gitee.com/freshday/radar/wikis/manual?sort_id=1637446

manual

本Wiki配有视频教程: http://www.riskengine.cn

1. 安装（适用于版本v1.0.6）

```
git clone https://gitee.com/freshday/radar.git
mvn clean install
```

2. 配置

2.1 数据库

```
CREATE DATABASE IF NOT EXISTS radar DEFAULT CHARSET utf8mb4;
# 初始化数据库
source radar-init.sql
source radar-1.0.6.sql
source radar-1.0.7.sql
source radar-1.0.8.sql
```

2.2 中间件

项目启动还需要安装 redis 、 mongodb、elasticsearch
windows 安装程序可以通过我的百度网盘下载,
链接：https://pan.baidu.com/s/1C-UdV71tAa6n07ZNo_EvVw 提取码：q3p4

或者通过docker 方式安装

```
// redis
docker pull redis:3.2
docker run --name redis3.2 -p 6379:6379  -d redis:3.2 redis-server --appendonly yes

// mongo
docker pull mongo:4.0.13-xenial
docker run --name mongo4.0 -p 27017:27017 -d mongo:4.0.13-xenial

// es
docker pull elasticsearch:6.8.7
docker run -d --name es6.8  -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node"  --restart=always  elasticsearch:6.8.7
```

V1.0.4后 es 升级到了7.6.X

```
docker run -d --name es7.6  -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node"  --restart=always  elasticsearch:7.6.1
```

2.3 应用配置

两个启动项目的application.yml 都需要进行修改, 除了mysql配置外，在启动前还需要配置如下几个必要的选项：mongodb,redis, 手机号码段文件，ip地址库, 依赖资源见上面的百度网盘

```
mongodb:
  url: mongodb://localhost:27017/radar   //mongodb
mobile:
  info:
    path: D:/soft/moble_info.csv        //手机号码段信息
ip2region:
  db:
    path: D:/soft/ip2region.db          // IP地址库
```

3. 启动

```
# 运行服务端
cd radar-admin
java -jar radar-admin.jar
# 运行引擎端
cd radar-engine
java -jar radar-engine.jar
```

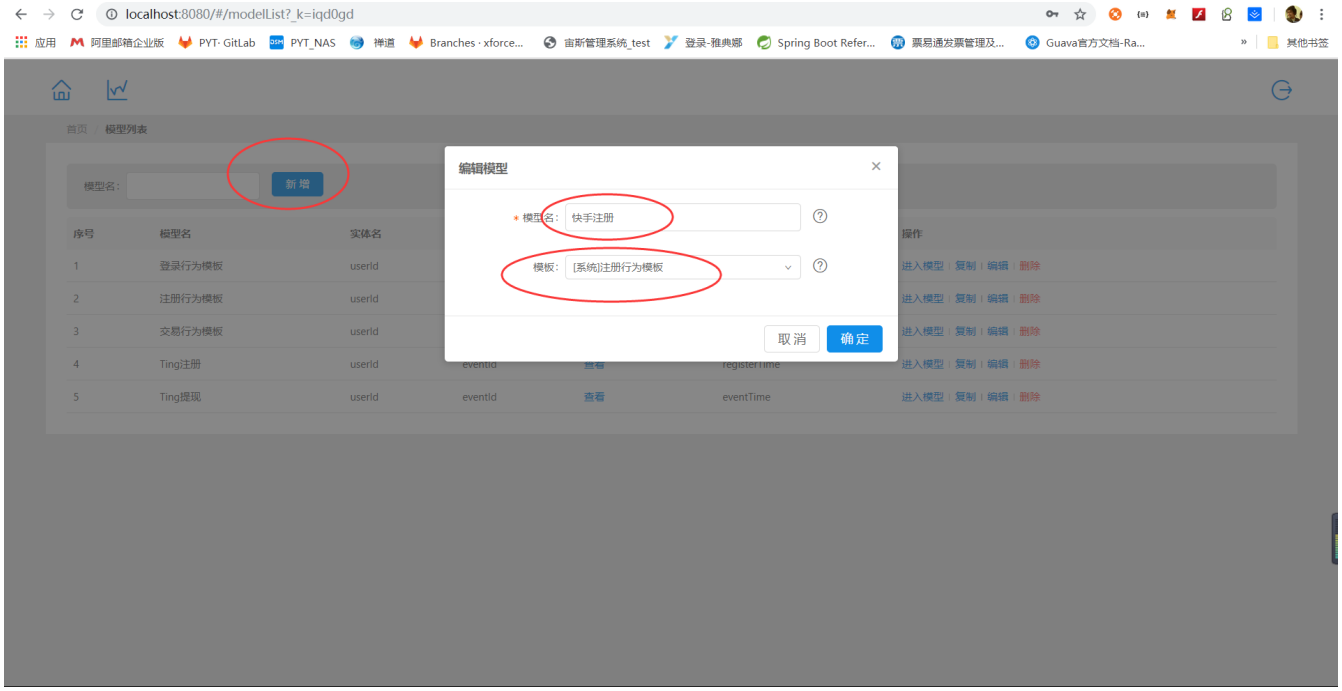
管理端入口：http://localhost:6580
默认用户：admin/123456

引擎端入口：http://localhost:6581

4. 模型配置

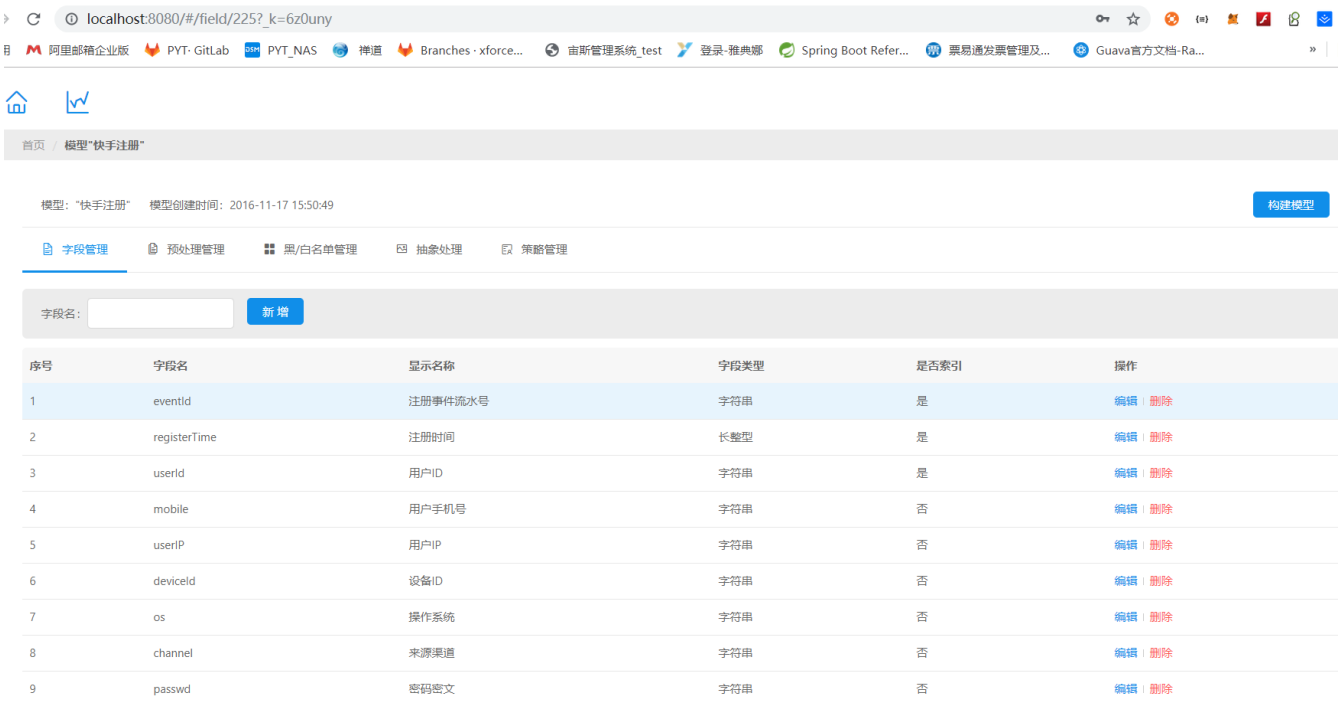
新建模型

初次熟悉系统的时候建议选择使用 模板 创建模型。



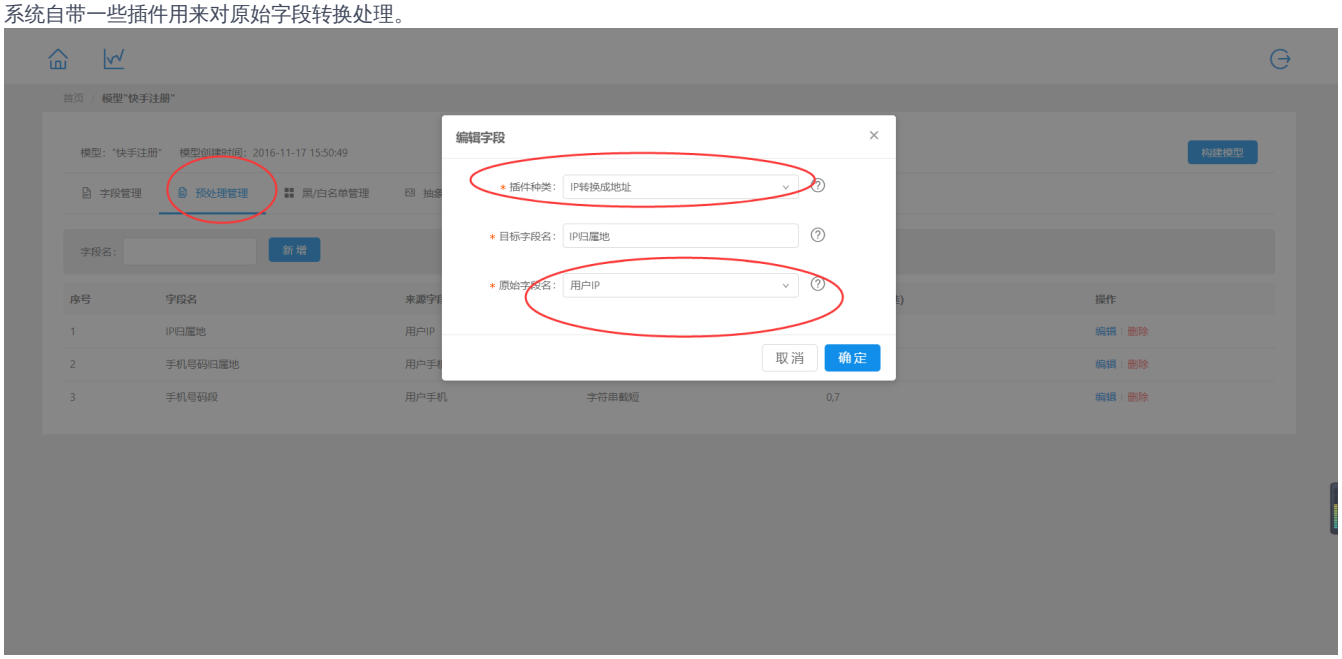
字段管理

查看刚才新建模型的字段，通过修改和增加调整模型。



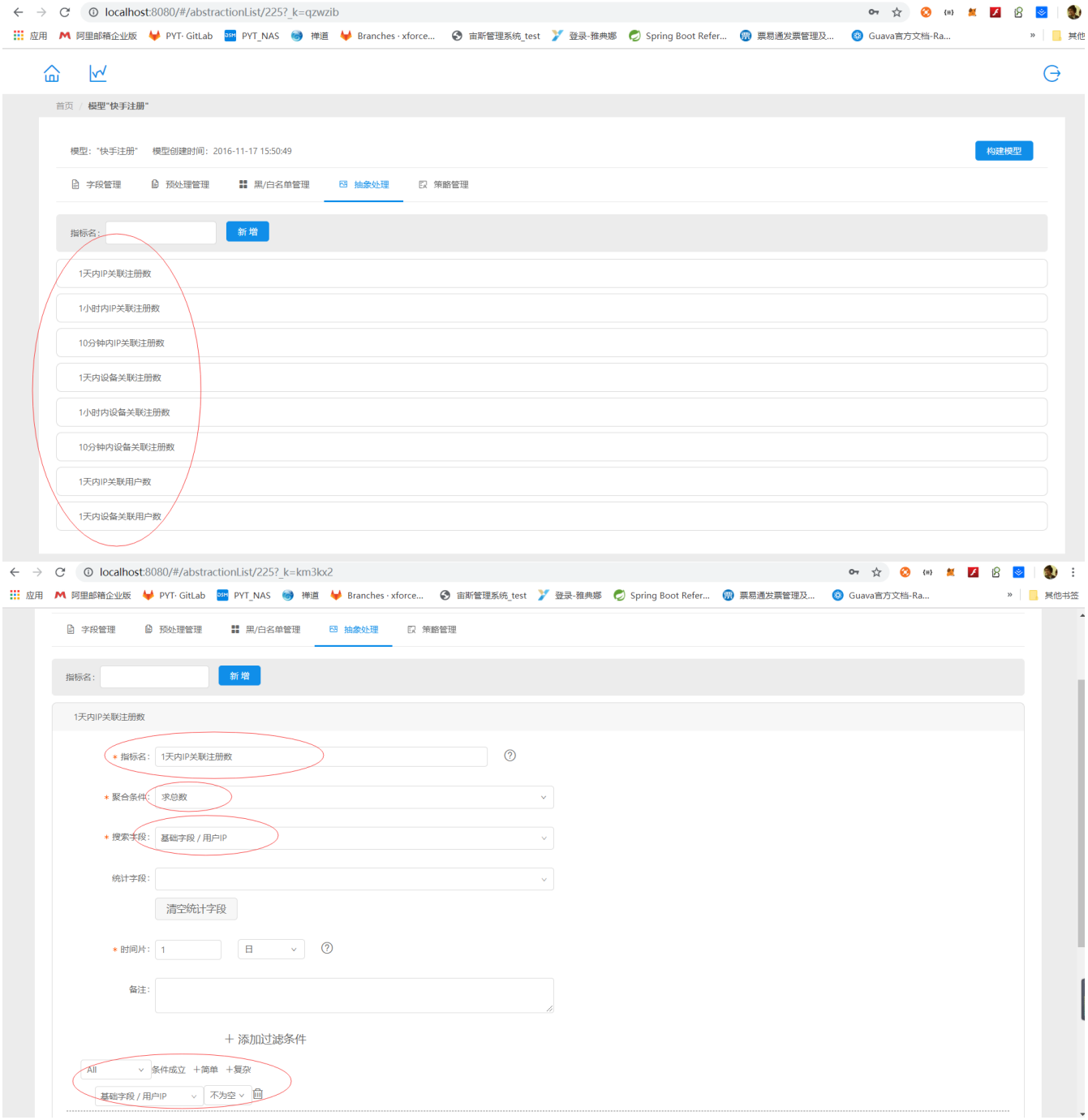
预处理字段

系统自带一些插件用来对原始字段转换处理。



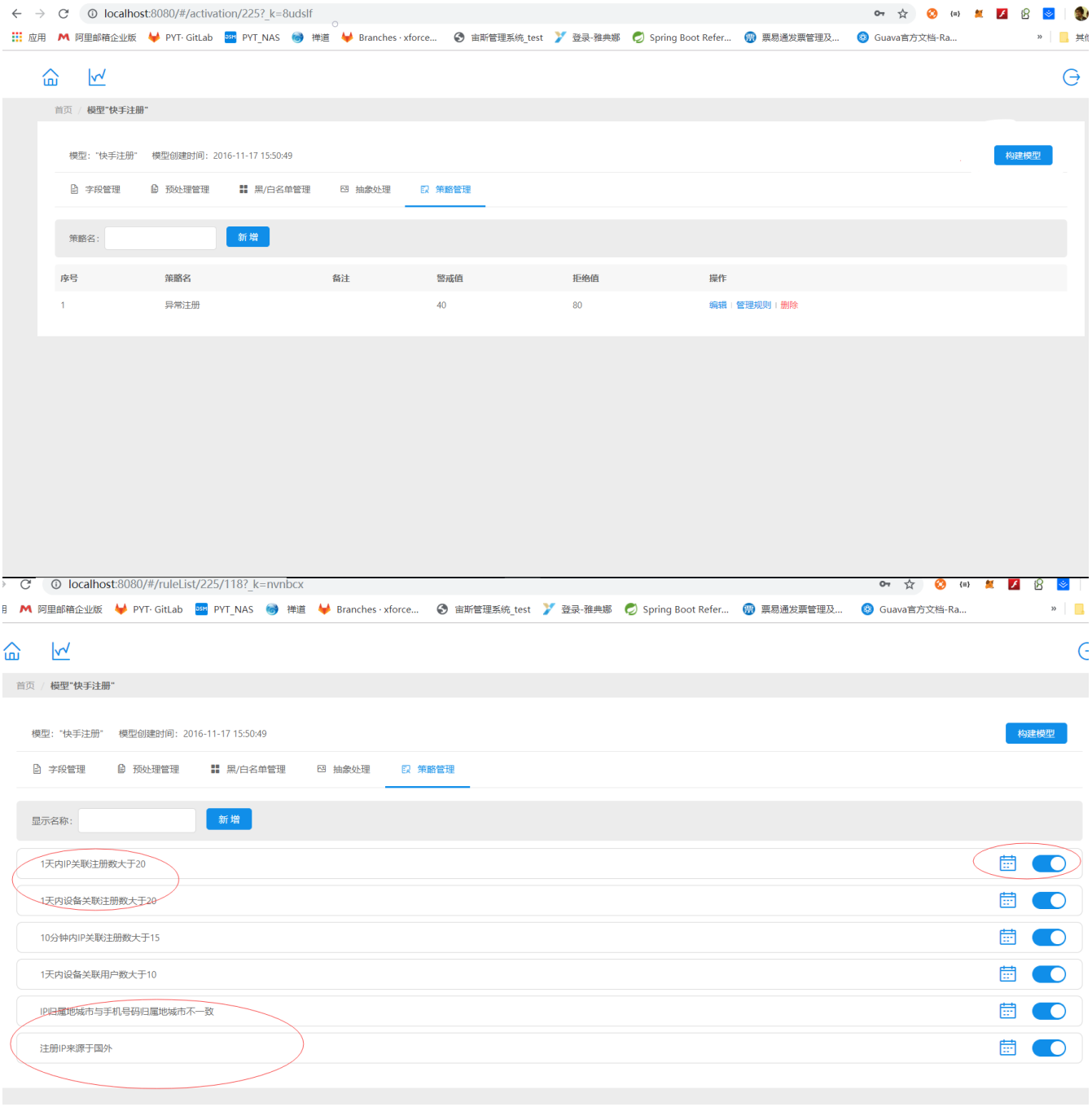
特征处理

定义我们需要关心的指标。



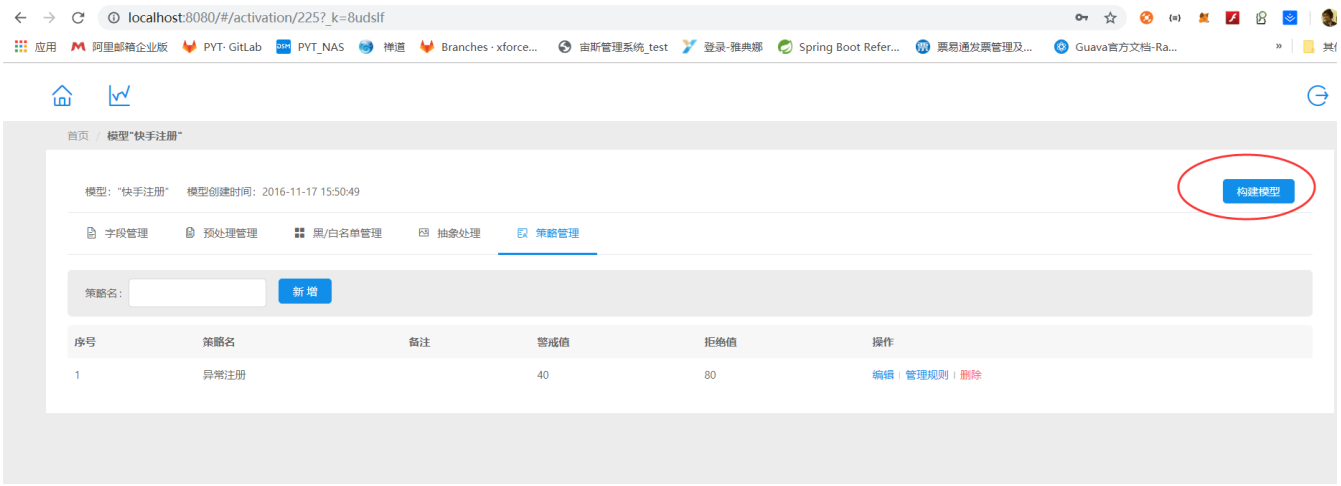
Activation 策略集管理

整个风险的量化过程就在这里，模型的输出点，组合若干条特征，综合计分，通常定义两个分数线，一个是审核线（低分数表示需要人工审核， 一个拒绝线，表示此交易可以直接拒绝）



构建模型

点击构建模型，Mongodb,ES将会为模型创建索引信息



模型激活

最后一定要点击激活按钮，大功告成！

模型建立好了，赶紧测试一下吧！ 测试模型

manual-docker(更新中-暂时不可用)

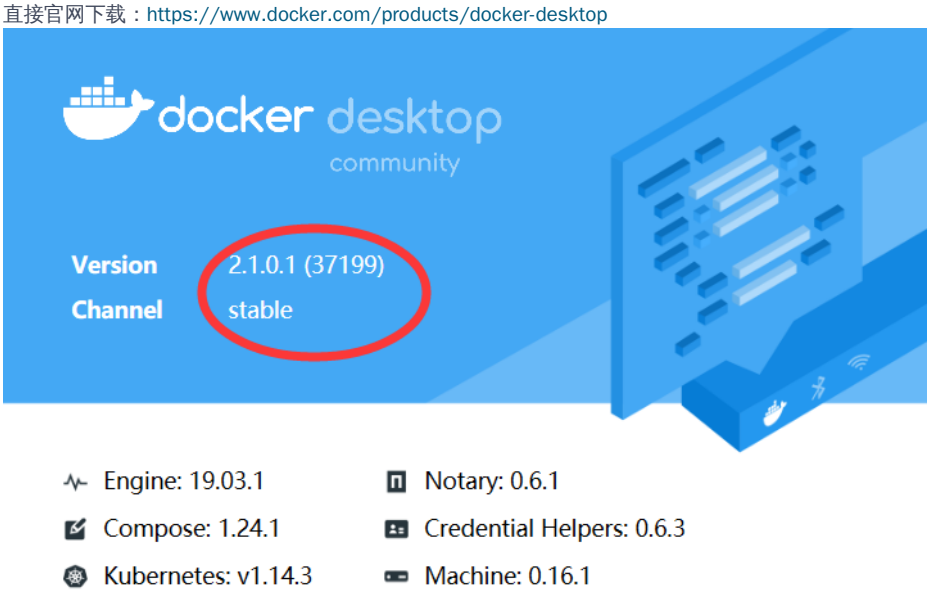
数据库

创建数据库，并导入数据到本地mysql数据库，数据库文件见 resources目录

```
CREATE DATABASE IF NOT EXISTS radar DEFAULT CHARSET utf8 COLLATE utf8_general_ci;
# 初始化数据库
source radar-init.sql
srouce radar-1.0.6.sql
```

安装 docker

直接官网下载：https://www.docker.com/products/docker-desktop



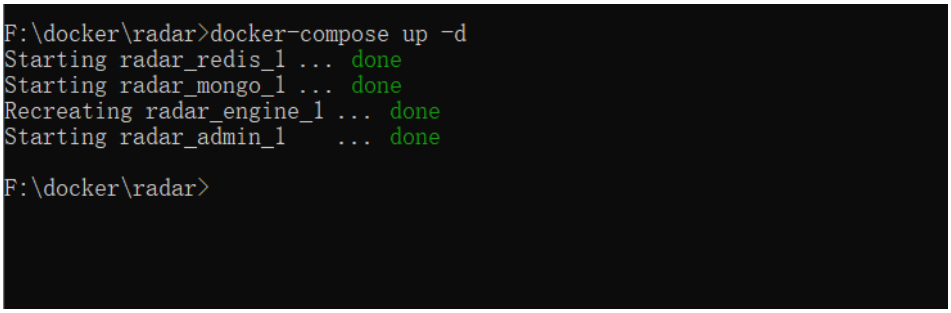
镜像下载

链接：https://pan.baidu.com/s/1C-UdV71tAa6n07ZNo_EvVw 提取码：q3p4 下载 *docker-radar.zip*，然后解压修改application.yml 配置文件，替换mysql 数据库url，用户名，密码等数据库信息。

启动镜像

```
docker-compose up -d
```

如图：



访问入口

管理端：http://localhost:6580/

引擎端：http://localhost:6581/

docker-compose.yml

```
version: '2'
services:
  admin:
    build: ./admin
    image: radar-admin:1.0.3
    ports:
      - "6580:6580"
    links:
      - "redis:redis"
      - "mongo:mongo"
  engine:
    image: radar-engine:1.0.3
    build: ./engine
    ports:
      - "6581:6581"
    links:
      - "redis:redis"
      - "mongo:mongo"
  redis:
    image: redis:3.2
  mongo:
    image: mongo:4.0.13-xenial
```

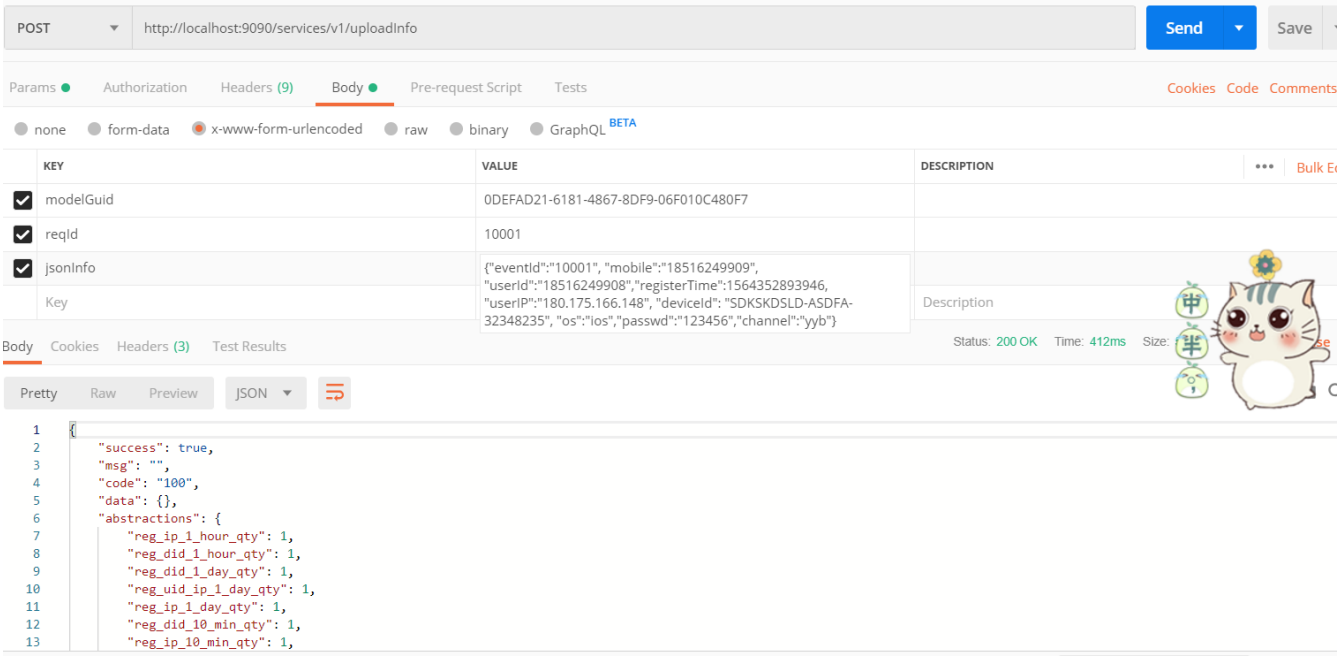
test

风控EngineAPI

项目采用Swagger做API文档管理，进行测试前请确认模型已经进行过 构建模型 模型激活
URL: engine api http://localhost:6581/swagger-ui.html

提交事件获取风险打分，通过POST + JSON 的方式提交数据，注意参数说明
modelGuid：model guid 可以通过管理控制台界面查看
reqId ： 请求流水号
jsonInfo： 事件具体json信息

序号	模型名	实体名	事件ID	唯一标识	事件时间
1	登录行为模板	userId	eventId	查看	loginTime
2	注册行为模板	userId	eventId	查看	registerTime
3	交易行为模板	userId	eventId	查看	eventTime
4	Ting提现	userId	eventId	查看	eventTime



请求数据样例： 建议使用 RequestBody

```
{
"guid" : "DB8A078F-97FE-4A7F-AAC0-5AF1A6C36CE8",
"reqId": 100001,
"jsonInfo": {
```

```
        "eventId":"100001", "mobile":"18516249909", "userId":"18516249909",
        "eventTime":1672924515000, "userIP":"180.175.166.148",
        "deviceId": "SDKSKDSLD-ASDFA-32348235", "os":"ios","amount":509.0,
        "channel":"alipay"
        }
}
```

或者使用请求参数:

```
modelGuid : DB8A078F-97FE-4A7F-AAC0-5AF1A6C36CE8
reqId: 100005
jsonInfo: {
        "eventId":"100005", "mobile":"18516249909", "userId":"18516249909",
        "eventTime":1672924515000, "userIP":"180.175.166.148",
        "deviceId": "SDKSKDSLD-ASDFA-32348235", "os":"ios","amount":500.0,
        "channel":"alipay"
        }
```

响应样例：

```
{
    "success": true,
    "msg": "",
    "code": "100",
    "data": {},
    "abstractions": { // 特征 指标的计算结果，方便大家调试和核对
        "tran_did_1_day_qty": 2,
        "tran_did_ip_1_day_qty": 0,
        "tran_uid_ip_1_day_qty": 1,
        "tran_uid_1_hour_amt": 1000.0,
        "tran_ip_1_hour_qty": 2,
        "tran_ip_1_day_amt": 1000.0,
        "tran_ip_10_min_amt": 1000.0,
        "tran_ip_1_hour_amt": 1000.0,
        "tran_uid_1_day_amt": 1000.0,
        "tran_did_10_min_qty": 2,
        "tran_did_1_hour_qty": 2,
        "tran_ip_1_day_qty": 2,
        "tran_ip_10_min_qty": 2,
        "tran_uid_10_min_amt": 1000.0
    },
    "adaptations": null,
    "activations": {
        "transaction_exception": {
            "risk": "reject", // 风险级别 : pass 通过， reject 拒绝， review 人工审核
            "score": 120  //风险积分
        }
    },
    "hitsDetail": { // 具体命中的规则项
        "transaction_exception": [
            {
                "key": "381",
                "value": 120.0,
                "desc": "1天内IP交易金额大于1000"
            }
        ]
    },
    "respTimes": { // 各模块耗时情况
        "adaptations": 0,
        "activations": 2,
        "abstractions": 27
    }
}
```

faq

问：radar 提供哪些功能？
答：radar 目前主要提供一个风控引擎平台，通过规则引擎统一管理风险，并支持可视化配置和修改，最大程度的做到所见即所得。

问：radar 怎么集成其他平台或者数据能力？
答：radar 通过插件机制，集成其他数据能力，目前系统自带手机号码段和IP转换（ip2region）两个数据处理插件，通过插件的形式获取关联数据，后续前端页面会提供插件管理功能，方便大家落地的到具体的应用场景。

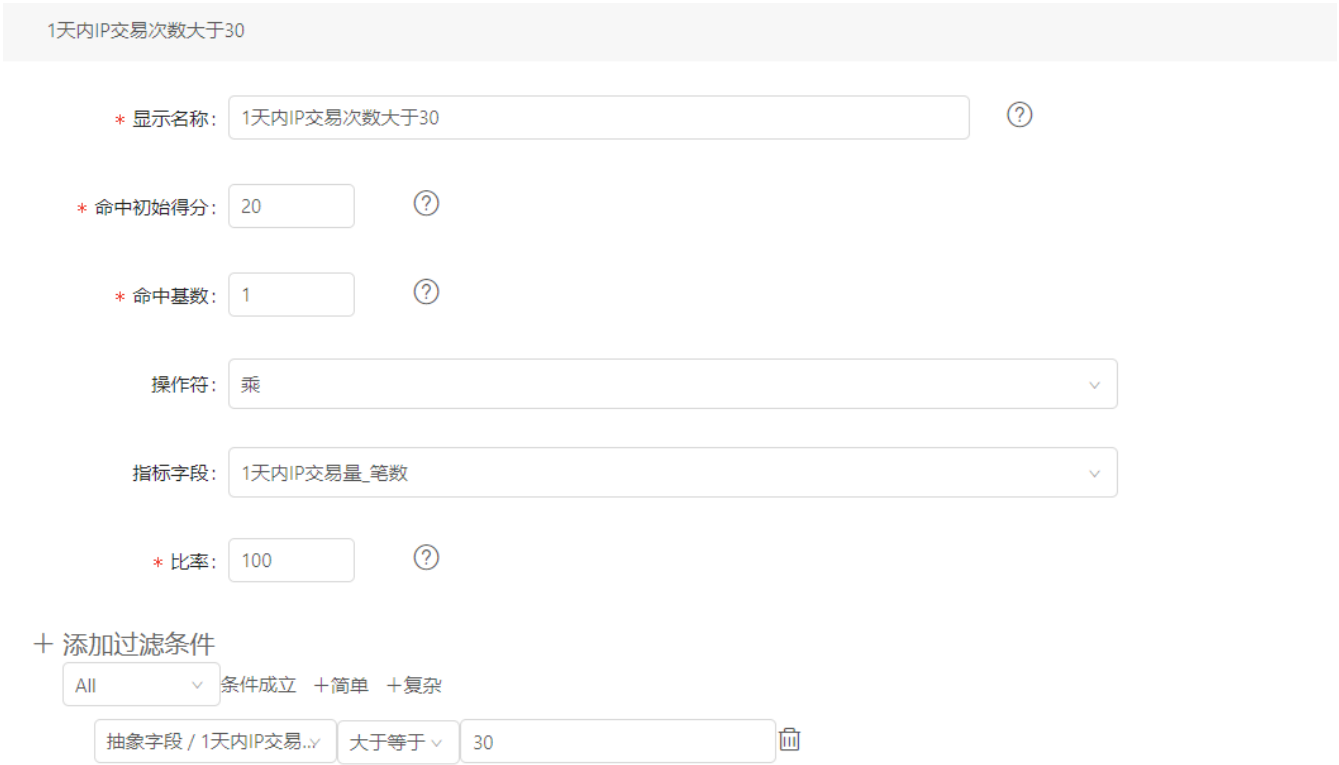
问： 作为一款实时的风控引擎，radar 为什么会采用 Springboot + Mongodb + Groovy 的架构设计？ 为什么没有采用其它更加优秀的 Storm, Flink ，Drools 等框架？
答：作为一款开源版的风控引擎，在设计之初也经过多次调研，开源的目的是设计成通用风控解决方案，普及风控知识以及解决方法，首要的目的是通用，易用，好用，所以在选择技术方案的时候做了一些取舍，不是说 storm, flink 在实时流式处理方面不好，选择 mongodb 主要是考虑在 数据存储方式，数据时间窗口更具有优势，弱schema、json格式文档存储，自动失效，nosql，shading 更加具有通用性。
而 Drools 是大家都较为熟悉的非常优秀的规则引擎框架，之所以选择Groovy 自定义规则引擎，主要是考虑 Drools 在可视化编辑方面，还不够灵活，前端支持难度大，参考了开源社区其他人做的产品，好多还是手写 drools 脚本，而radar 做到了规则定义全中文支持，看不到变量的定义，配置灵活多变，当然groovy 动态脚本也有缺点，动态编译非常消耗性能，高并发可能导致频繁fullGc的问题。

问：目前radar底层使用mongodb，考虑的是长时间窗口（目前推荐3个月），相对于其它实时流式引擎（flink）来说，只能说是准实时，在处理时间窗口较短的场景（秒级，分钟，小时）有明显的弱势，后续是否需要支持flink，以应对实时性非常高的场景？
答：从目前项目架构来说，特征（abstraction）的提取的实现类是基于mongodb来实现的，理论上来说，只需要在基于flink 再实现一遍即可，后续将列入版本计划，同时支持 mongodb 和flink 通过配置选择使用。

问：规则的解析和执行是在什么时候进行的？
答：目前特征（abstraction）的提取和策略集(activation)的执行 都涉及到规则脚本的执行，规则的解析的逻辑在前端页面，具体可以参考 GroovyScriptUtil.java 和 groovyUtil.jsx (react) 这两个类的使用。

问： radar 的评分怎么做的？

答： radar 采用综合累计积分， 简单的来说： f(r) = （ax +n1）+ (by + n2) + (cz + n3)，f(r)为风险总分， xyz 为 特征，， abc 为权重，n 为偏移量。



如图，过滤条件 1天内IP交易次数大于 30，假设现在达到31， 这条策略的计分为：20 + 1 * 31 * 100% = 51
最后简化成数学表达式：

$$f(risk) = \sum_{i=0}^{n} K' x_i + b$$

注：xi 为第 i个特征值

问： 机器学习支持的方式（PMML）？参考链接：https://www.cnblogs.com/pinard/p/9220199.html 答：*现阶段很多机器学习框架都支持java 直接调用，所以直接集成相应的api 就好。*

backlog

1. 关于模型C 依赖于 模型A,B的结果的情况，组合模型怎么处理？
   可以使用 HTTP 插件，在 计算C 模型的时候，加载 A,B模型的结果
2. 规则计分的公式能否是配置多种算法进行选择？

risk-define

风险的定义

翻开手机里面的APP可以看到用户注册，用户登录，提现，充值，兑换，交易等常见的功能，所谓风险就是对这些交易（各种交易行为的统称）非正常程度的评估。

风险控制

通常我们是如何处理风险的了？
例如：注册，用户提交注册请求的时候，我们会检测当前设备或者IP 在当前时间段的注册数，如果达到某一数量，限制注册一段时间。
登录， 很典型的处理方式，累计密码错误次数（通常3次以上），我们会延长其尝试时间。 提现， 限制小额提现的交易次数， 大额交易需要提前预约等。

风险的量化

所谓量化，就是以用户的行为特征为依据，然后通过科学的方式进行计算，得出一个综合的分数，通过这个风险的分数可以直观的判断交易的风险程度。

风险的集中管理

我们知道企业做大后，会有很多产品线，而几乎每一个产品都需要做风险控制，通常我们都是把风险控制的逻辑写在相应的业务功能代码里， 大量重复的风控逻辑代码耦合在我们的业务逻辑里面，随着时间的累积，代码会变得异常复杂，会给后期的维护造成巨大的人力成本和风险。

所以风险的集中管理势在必行，只有通过一个统一的管理平台，使用规则引擎，采用可视化配置的形式，平台化管理不同产品风控策略才是一种更好的方式, 而这正是Radar的初衷。

插件使用

HTTP PLUGIN

用于关联其它系统数据，目前只支持Get请求方式，参数通过占位符形式传递，不支持post+ requstbody。

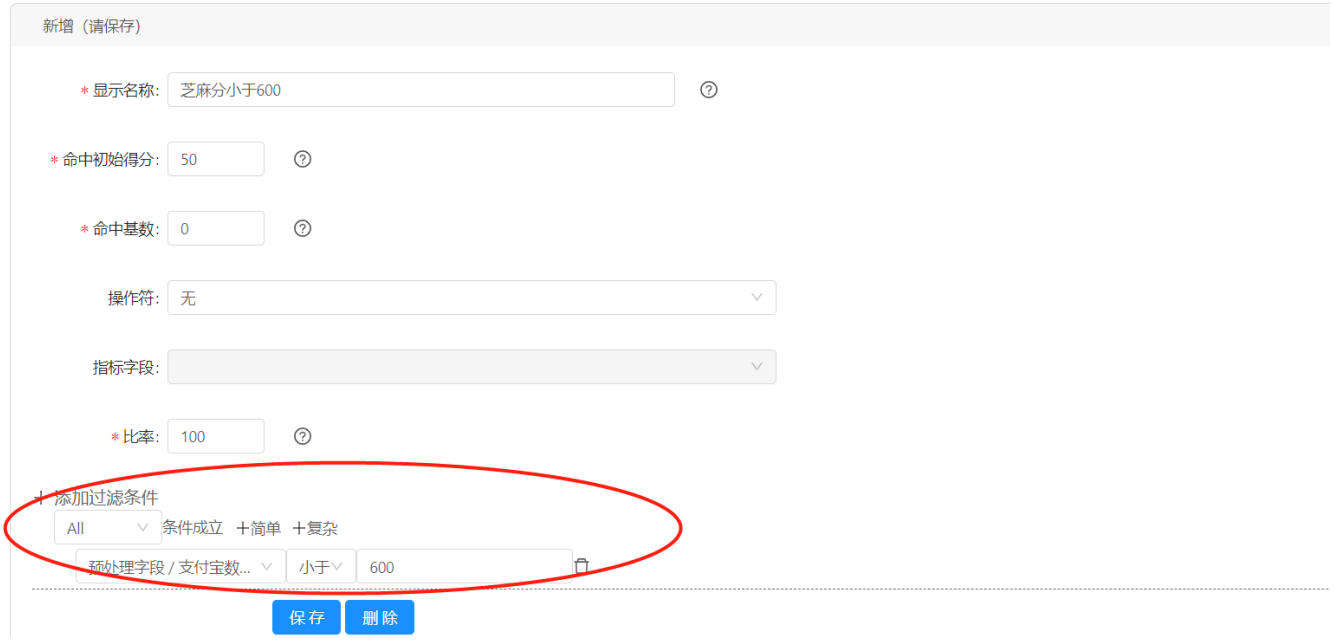
配置插件

选择http util 插件，注意配置响应字段定义，用于engine关联响应字段。
响应字段定义样例:

```
[{
 "column": "aliscore",
 "title": "芝麻分",
 "type": "INTEGER"
}, {
 "column": "huabei",
 "title": "花呗额度",
 "type": "INTEGER"
}]
```

在预处理模块配置

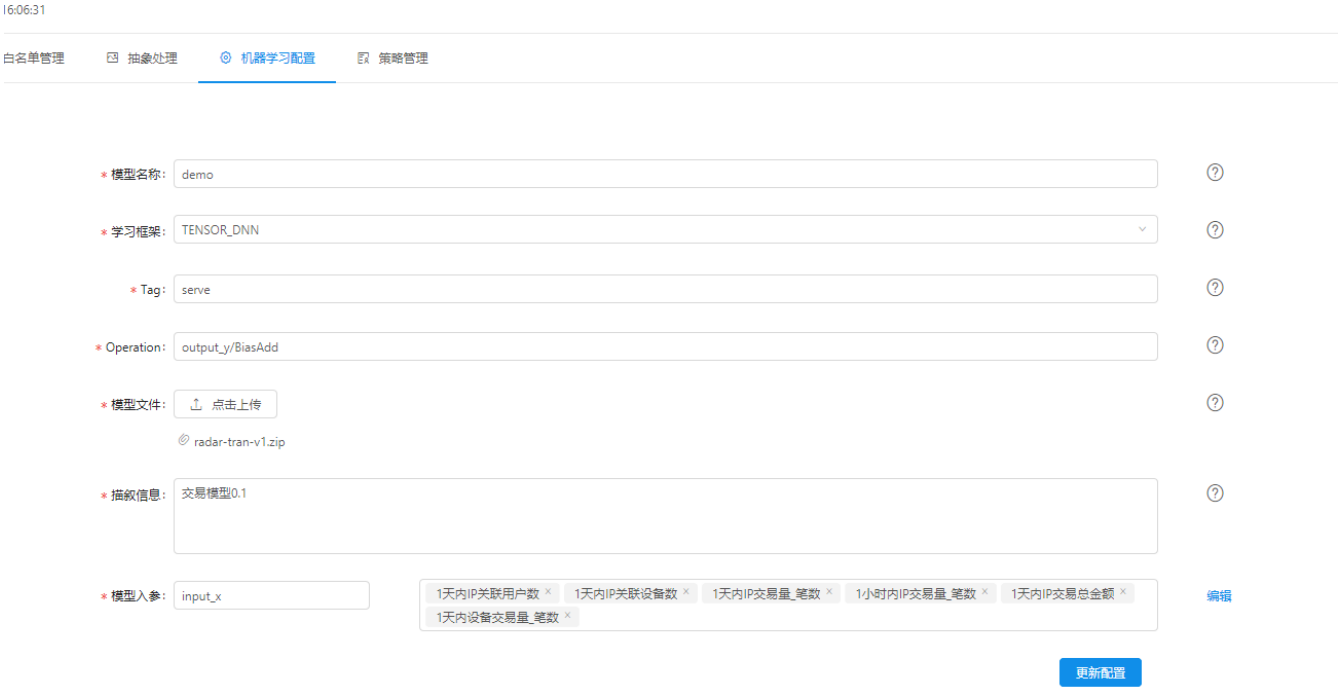
如何在规则中使用


新增（请保存）
* 显示名称: 芝麻分小于600
* 命中初始得分: 50
* 命中基数: 0
操作符: 无
指标字段:
* 比率: 100
添加过滤条件
All
条件成立 +简单 +复杂
预处理字段 / 支付宝数...
小于
600
保存
删除

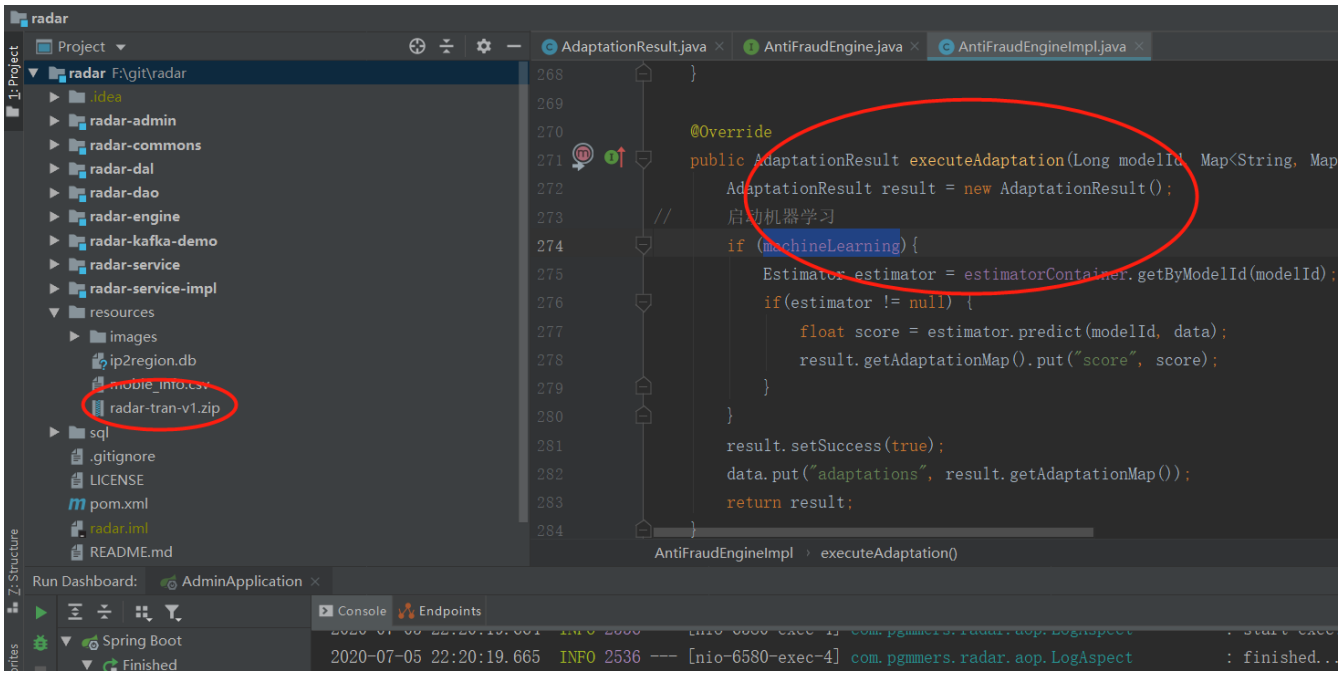
机器学习页面使用介绍

目前仅仅只是集成了tensorflow，预言阶段，因为问的人多，所以把测试demo使用贴出来 ，默认的测试模型见工程 resources目录, 这是一个关于交易性质的事件.

配置



代码和默认模型



规则引擎生成的元数据和Groovy脚本

通过管理平台配置的规则，会生成两种信息，一种用于前端页面显示的元信息，一种是用于groovy 脚本，用于运行时，分别如下：

```
"ruleDefinition": {
 "linking": "All",
 "conditions": [{
  "class": "SMPL",
  "expressions": [{
   "column": "abstractions.tran_ip_1_hour_qty",
   "type": "DOUBLE",
   "class": "ENTATTR"
  }, {
   "type": "DOUBLE",
   "class": "CONST",
   "value": "15"
  }],
  "enabled": true,
  "operator": "Greater_Equal"
 }],
 "class": "PDCT",
 "enabled": true
}
```

```
class ActivationCheckScript {
    public boolean check(def data, def lists) {
        if (data.abstractions.tran_ip_1_hour_qty>=15)
         return true;
         else
            return false;
    }
}
```

捐赠费用使用明细

2020.03.17 购买演示环境服务器1台（华为云） 571元 2020.12.25 给开发者发红包 752元

名词解释

名词解释

Model: 事件模型

用户行为事件， 例如：注册，登录，购买，提现等具体的业务行为。

事件模型三要素

也就是事件行为三要素，风控系统的核心定义：事件流水ID(例如：交易流水号)，实体ID（例如：userId,表示某个人或物），事件发生时间（例如：交易时间），简单来说就是谁（可以是人，也可以是物）什么时候做了什么事。（who, when, what）

PreItem: 预处理

像经纬度，IP，手机号码段等事件属性，可能无法直接计算，通过预处理插件 转换成 其他格式， 例如:经纬度，ip 可以通过对应插件变成位置和地址，手机号码可以通过插件获取其它系统的用户画像信息等。

Abstraction: 特征

特征工程，例如用户小时交易次数，IP 一天交易金额，设备一小时交易次数。。。

Adaptation: 机器学习模型适配器

使用训练好的机器学习模型，进行检测

Activation: 激活器

概念类似于机器学习里面的 (Activation Function)， 一个模型可以定义多个 activation（相当于不同维度的检测报告）,每个activation都可以独立配置规则，单独打分。 例如，用户注册行为， 可以定义：异常注册， 垃圾注册， 可以输出多个activation。

Rule: 规则

在计算 abstraction 和 activation 之前，需要先检查数据是否正常，检查就是按照rule 配置进行检测。

Plugin：插件

为了通用性，项目自身并不提供敏感数据能力，但是通过扩展插件来获得其它系统或者中间数据功能， 目前项目自带手机号码段和IP地址转换两款插件，理论上可以通过扩展插件集成任何其它数据的能力， 例如：如果商户自身带有内部用户画像，可以自定义一个插件用过用户ID来获取用户画像数据。

release note

v1.0.6

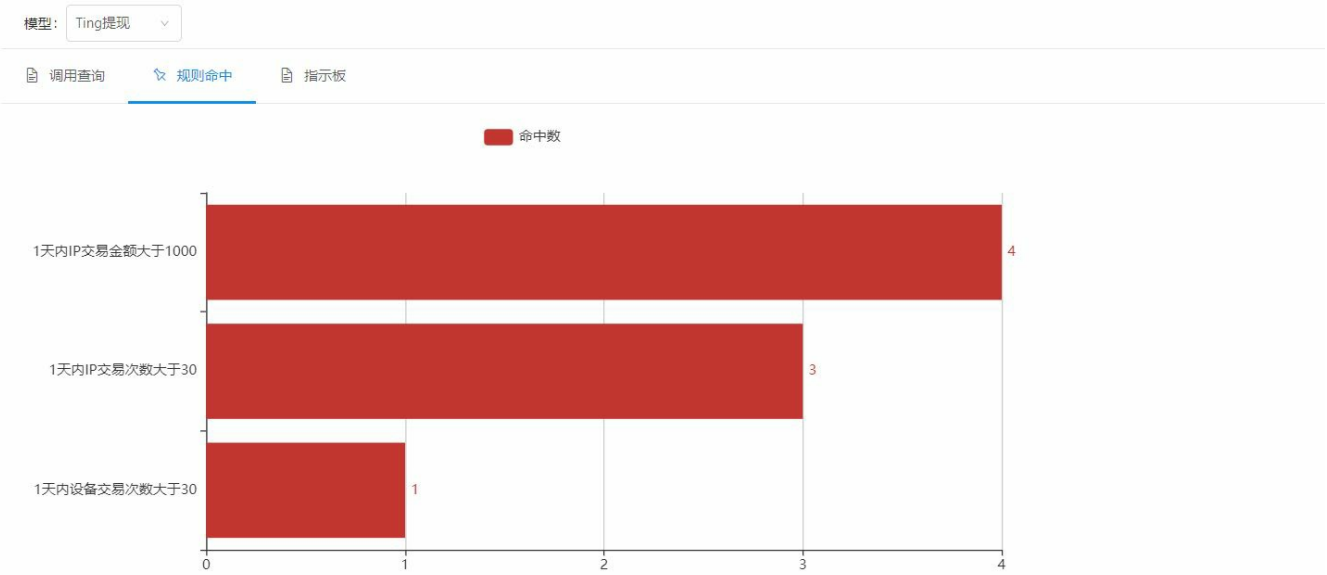
1.经纬度转换成行政区域
2.HTTP插件支持多参数
3.前端支持系统黑白黑白名单
4.规则命中图标支持时间过滤

v1.0.4

1. mongodb的访问方式升级为 SpringData。
2. elasticsearch 升级为7.X 使用 high level api 访问。
3. 集成 mapstruts， 提升对象copy 性能。
4. 配置文件拆解，支持多环境配置
5. kafka集成radar Demo

v1.0.3

docker 镜像的编写， 需要包含 mysql, redis, mongodb, es 等中间件。
data list record 导入功能 前端部分
增加 ES 中间件，提供数据查询，规则命中统计报表等功能
机器学习支持, 主流机器学习框架的API集成，用于已经训练好的模型调用 （TensorFlow）。



字段名	命中数	操作
1天内设备交易次数大于30	1	查看明细
1天内IP交易次数大于30	3	查看明细
1天内IP交易金额大于1000	4	查看明细

v1.0.2

列表数据导入
Rest 工具 插件（通过Http获取其它关联数据）

v1.0.1

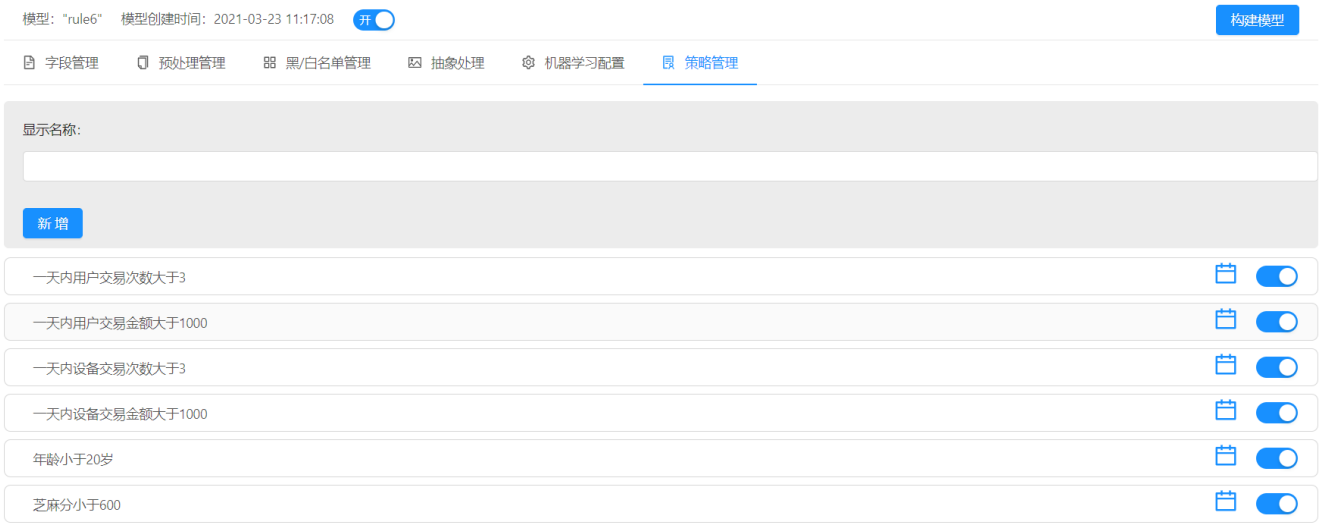
React 版本升级（v15.0.0）
集成 JWT(JSON WEB TOKEN)，前后端分离标准化
日期格式化插件

压力测试报告

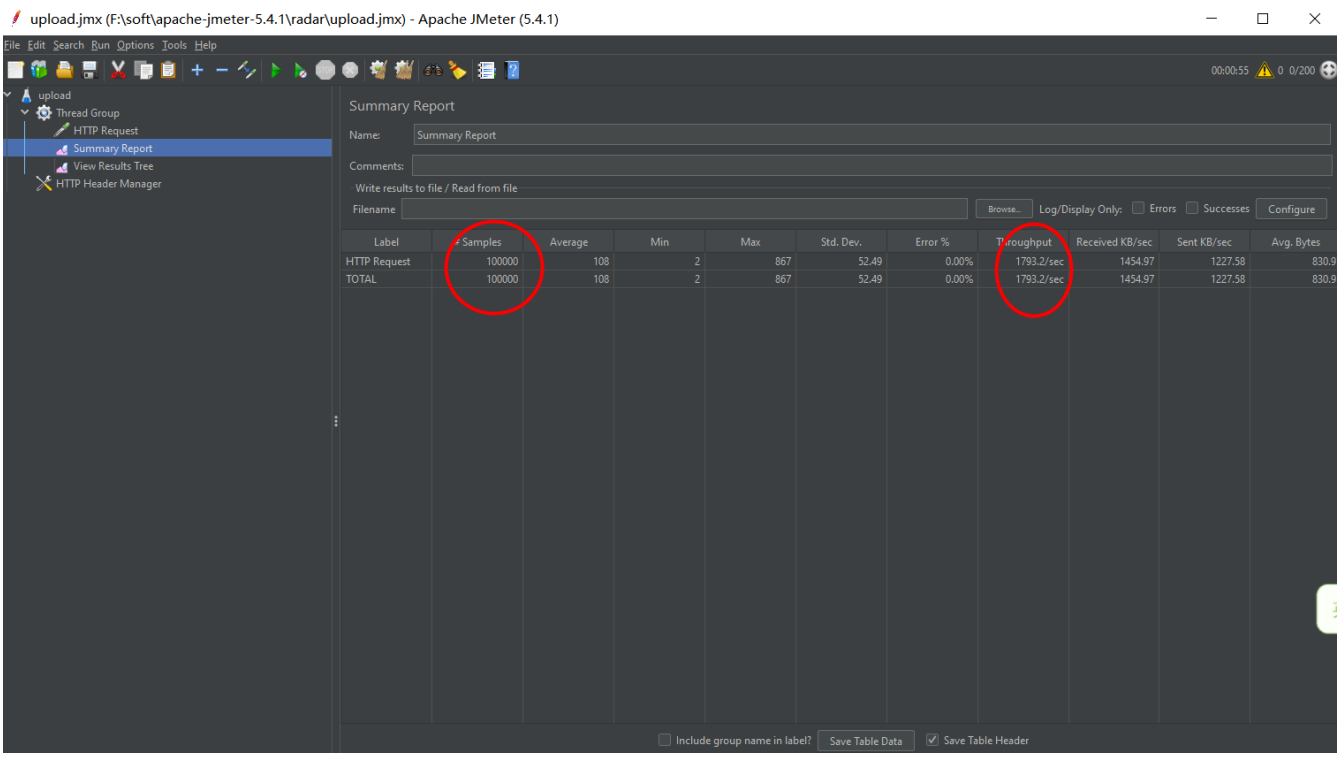
压力测试报告--200并发（JMeter）

本地测试，不使用任何特征提取功能，测试纯规则下的性能情况

规则配置



测试结果



压力测试报告-网友提供

1000并发测试，感谢网友(安达)提供的测试报告，30个docker实例，异步上报，优化完后，5000TPS