**寄云Rule EngineV1.0项目**

**系统概要设计方案**

**20 17 年 5 月 15 日**

**修订信息**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本编号** | **时间** | **修改者** | **描述** | **文件名称** |
| **0.01** | **2017-5-15** | **王宇** | **创建文档** |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |
| --- |
| 1 引言（概述-放入引言里） |
|  |
| 2总体设计 |
| 3模块功能描述 |
| 4接口设计 |
| 5数据结构设计 |
| 6运行设计 |
| 7安全设计 |
| 8系统开发环境 |

# 1引言

规则引擎帮助用户灵活地处理设备消息，用户可通过规则引擎设定消息处理规则，对指定的消息采取相应的措施来对设备进行监控和处理，如发送警告信息；也可以将设备消息转发， 或者过滤到其他部件。

传统的商业规则引擎概念和实现并不能满足工业物联网规则引擎需求。首先，商业规则引擎的条件判断基于决策支持系统，太繁复。其次， 商业规则引擎的动作设定多为商业事件， 如增加积分等， 也不适用于工业物联网平台。

在工业物联网中，由于数据量巨大，业务规则可能多种多样，需要将规则的设置变得简单和友好以适应业务规则的多样和变化。大量的数据往往对应着不同的应用分析场景，如监控温度， 压力等传感器数据是否超标， 或者监控某个时间范围内的变化范围是否满足某个公式条件等。

典型的应用场景包括：

1. 实时告警异常的数据；
2. 计算一天内的温度最大最小和均值等；
3. 将满足某条件的的数据转存以便查询；
4. 过滤异常数据； 等。

规则引擎可以在服务器端运行， 也可以在网关， 或者数据接收端运行以达到不同的业务目标。在网关或者数据接收端运行的规则引擎可以视为边缘计算的部分实现。

本文给出了寄云规则引擎的概要设计思想， 本文的阅读者是产品和研发人员。

# 2背景

## 2.1 基本概念

规则引擎是一种根据规则中包含的指定条件，判断其能否匹配运行时刻的实时条件来执行规则中所规定的动作的引擎。与规则引擎相关的有三个概念

1)规则  
　　规则包含一个条件过滤器和多个动作。一个条件过滤器可以包含多个过滤条件。条件过滤器是多个布尔表达式的组合，其组合结果仍然是一个布尔类型的。在程序运行时，动作将会在条件过滤器值为真的情况下执行。

2）动作

动作是一段可执性的代码。

1. 消息队列

消息队列是规则应用的主体， 消息队列中包含的消息具有各种属性。 这些属性出现在规则过滤条件中， 用于判断某条消息是否满足条件。

## 2.2 竞品分析

规则引擎起源于基于规则的专家系统，而基于规则的专家系统又是专家系统的其中一个分支。 这种规则引擎称为商业规则引擎（Business Rule Engine）. 商业规则引擎采用基于决策支持的算法， 如forward chain， RETE等。 比较著名的开源商业规则引擎（<http://www.jboss.org/drools/>

）是Java语言开发，基于Charles Forgy的RETE算法的开源的业务规则引擎。商业的BOSS， ERP系统中一般都有这个模块。

商业规则引擎概念和实现并不能满足工业物联网规则引擎需求。首先，商业规则引擎的条件判断基于决策支持系统，太繁复。其次， 商业规则引擎的动作设定多为商业事件， 如增加积分等， 也不适用于工业物联网平台。

在工业物联网中，由于数据量巨大，业务规则可能多种多样，需要将规则的设置变得简单和友好以适应业务规则的多样和变化。大量的数据往往对应着不同的应用分析场景，如监控温度， 压力等传感器数据是否超标， 或者监控某个时间范围内的变化范围是否满足某个公式条件等。

Rule engine 作为IOT套件中的一环， 越来越多的在IOT系统中出现。 AWS, azure, 百度云都有rule engine.

* Baidu Rule Engine

<https://cloud.baidu.com/doc/RE/ProductDescription.html>

Msg: iot hub

Action: kafka, TSDB, MQTT topic

Rule : SQL expression

简单匹配

* Amazon Rule

<http://docs.aws.amazon.com/iot/latest/developerguide/iot-rules.html>

消息: MQTT Topic

Action: filter, send msg, save to, generate alarm.

Rule: SQL

http://docs.aws.amazon.com/iot/latest/developerguide/iot-create-rule.html

* Thingsboard

Rule:　SQL statement

简单过滤

Action： 过滤数据到某目的地。

# 3总体设计

## 3.1需求规定

### 3.1.1 功能性需求

## 规则定义， 定义规则条件部分，和动作部分。 包括修改规则。

## 规则注册， 将定义好的规则注册到规则引擎中。

## 规则启动和停止， 启动已注册的规则， 或者停止已启动的规则。

## 规则查询， 查询规则状态。

### 3.1.2 非功能性需求

## Restful API

## 平台一体化用户和角色管理

## GUI 界面

## 客户端安装运行版本， 服务器端安装运行版本

## 3.2运行环境

## 本系统可以在公有云上部署， 也可以在私有云上部署。

本系统是纯软件系统， 可以单机方式运行于客户端操作系统之上。

## 3.3 架构

UI

逻辑模块：

Rule

Rule register

Apply

Rule Storage

实时数据

实时数据

MQ

Rule Engine

实现架构：

UI

Rule

Rule register

Trigger

Postgres

实时数据

实时数据

Kafak

Kafka Streaming

实时数据

Spark Streaming

## 3.4 规则描述

 2.4.1 规则BNFC

Rule :=  if Conditions then Actions

Conditions := Conditions and Condition |

            Conditions or Condition |

Condition

Condition := Key Operator Value

  Key := Key name string | Function（Key name string）

Function（Key name string） : = Window Function

WindowFunction := (day|hour| minute|second) (key, (min|max|mean|var))

 Operator:= Basic Operator |

          Advanced Operator

 Basic  Operator:=   > | < | = | >= | <= |<>

 Advanced Operator := ℗ |SelfDef

 Value: = Static value | Key |Function| Expression(key) | Model

Actions:= Alarm | Filter | SelfDef

 2.4.1 规则约束

1. ℗只能跟Model名
2. Expression（key） 中只能有1个Key， 其他参数都是常量
3. SelfDef action是可执性的代码段（java， Scala， python）

## 3.5 消息队列

MQTT的Topic 与Kafka 的Topic有一一对应关系， 消息队列对应Kafka的Topic。

# 4功能模块描述

## 4.1 规则定义

根据规则描述定义规则。

## 4.2 规则注册

## 规则分为两大部分， 条件和行为。 条件分三部分：key，operator， 和Value。

## 需要注册的内容包括：

## Key

|  |  |  |
| --- | --- | --- |
| Attribute name | Description | Value Example |
| Name |  | Temperature |
| Topic | 所在消息的Topic | Igwid/sensorid |
| Type | 对应Value的类型 | double |

## Operator

|  |  |  |
| --- | --- | --- |
| Attribute name | Description | Value Example |
| Name |  | = |
| ary | 参数个数 | 2 |
| Type | 对应Value的类型 | Any |
| class | 所属类别 | basic |

## Function

|  |  |  |
| --- | --- | --- |
| Attribute name | Description | Value Example |
| Name |  | = |
| ary | 参数个数 | 1 |
| Type | result的类型 | Any |
|  |  |  |

1. Expression

|  |  |  |
| --- | --- | --- |
| Attribute name | Description | Value Example |
| Name |  | = |
| ary | 参数个数 | 2 |
| Type | 对应Value的类型 | Any |
| class | 所属类别 | basic |

目前不支持Expression

1. Model

|  |  |  |
| --- | --- | --- |
| Attribute name | Description | Value Example |
| Name |  | Kmeans |
| module | 执行的模块 | spark |
| Result Type | 对应Value的类型 | Boolean |
| language |  | Java |
|  |  |  |

1. Action

|  |  |  |
| --- | --- | --- |
| Attribute name | Description | Value Example |
| Name |  | filter |
| module | 执行的模块 | spark |
| Result Type | 对应Value的类型 | Boolean |
| language |  | Java |

1. Rule

|  |  |  |
| --- | --- | --- |
| Attribute name | Description | Value Example |
| Name |  | RuleTAlarm |
| condition | 条件 | Temperature >500 and Temperature <100 |
| action | 行为 | TAlarm |
| topics | 数据流的topic | t1, t2 |

1~6的注册都是Rule注册的基础。 当注册一个规则时， 系统会验证条件表达式和行为的合法性。 根据已注册的内容进行验证。

注册成功， 生成一个Kafka的Consumer。 这个Consumer消费在已注册的Rule中发现的Keys 所在的Topics。 根据operator 选择执行的集群， 如果是基础operator， 则为Kafka streaming； 如果是高级operator， 则为spark streaming。

## 4.3 规则应用和停止

注册成功的规则， 可以应用。 选择需要应用的规则， 启动注册时定义好的consumer任务， 开始消费Topic的数据并按照规则生成行为。

## 4.4 规则查询

## 4.4 系统管理模块

包括 用户管理， 角色管理， 系统资源管理等。

用户管理能够添加， 修改和删除用户。 修改用户密码。

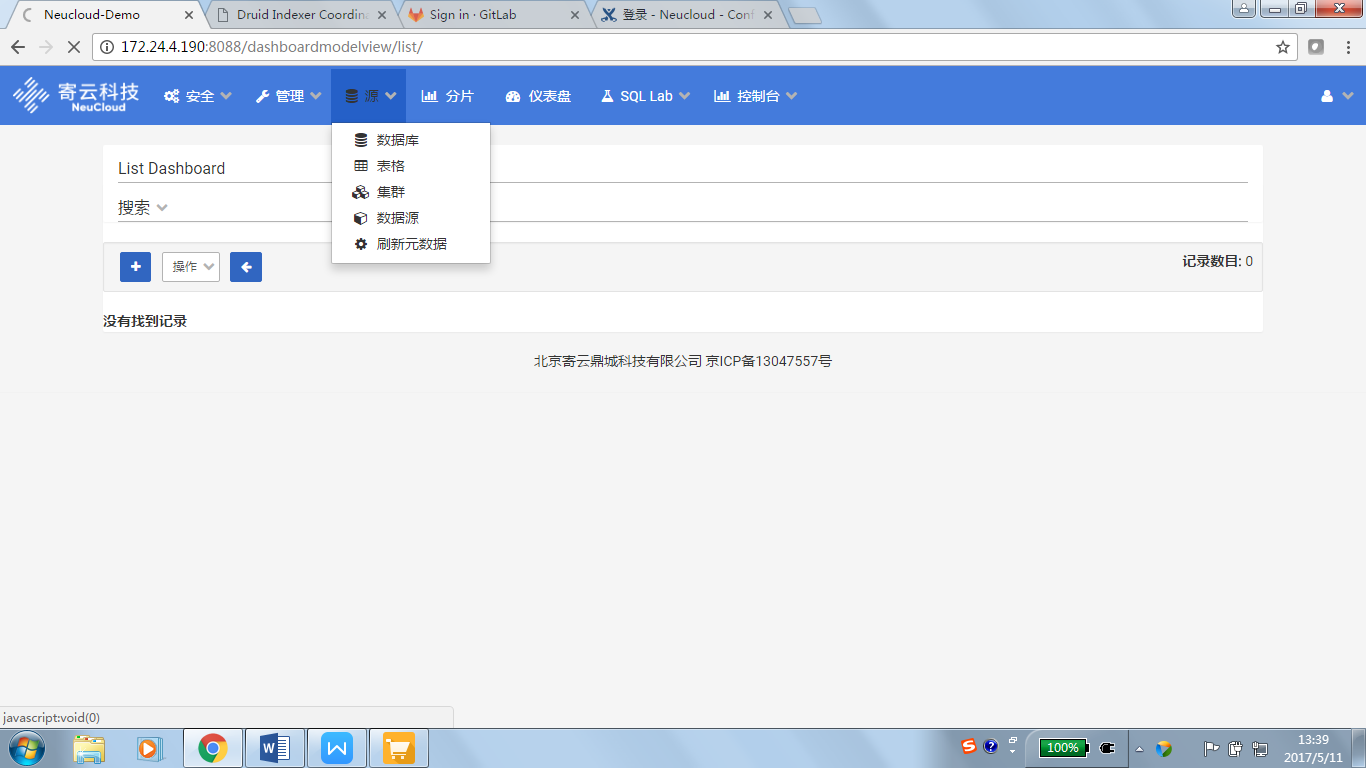
角色管理能够添加， 修改和删除角色。 给用户分配角色。 给角色分配权限。

## 

## 5接口设计

## 5.1用户接口

## 用户通过GUI界面管理， 查询规则引擎系统。



## 5.2外部接口

## 规则引擎Restful API

1. 规则定义
2. 规则注册
3. 规则应用
4. 规则查询

## 

## 5.3内部接口

1.. MQTT 与Kafka接口

MQTT的Topic与Kafka的topic有一一对应关系。 作为数据的输入端。

2. Kafka 与Spark streaming 接口

Model matching。

3. meta-data DB 与Kafka 接口

# 6数据结构设计

规则元数据存储：

Condition：

Function：

Operator：

Action：

# 7运行设计

## 7.2运行依赖

运行本时序数据库系统， 须安装如下组件

1. Kafka 0.9.x
2. Spark 0.9
3. Postgres or Mysql

# 8安全设计

1. 用户管理， 角色管理， 保证访问控制。
2. 数据压缩保证数据安全性。

# 9 系统开发环境

1. Java， Python， Scala 语言混合开发。
2. 前端采用 Node.js 框架

# 10 open issue

1. Model matching 规则实现

超过两个key 的复杂计算？

1. Spark Model 实体化
2. 批量？
3. IGW Topic
4. Kafka Management

1. Online model testing？
2. Action 的同步？