**物联网平台3.0方案设计**

# 编写目的

本方案在物联网平台2.0基础上进行优化升级，对平台区和租户区进行分离，增加数据分析产品，增加数据分发规则引擎，增加组态、算法和报表建模工具，构建设备中心。

# 总体架构

****

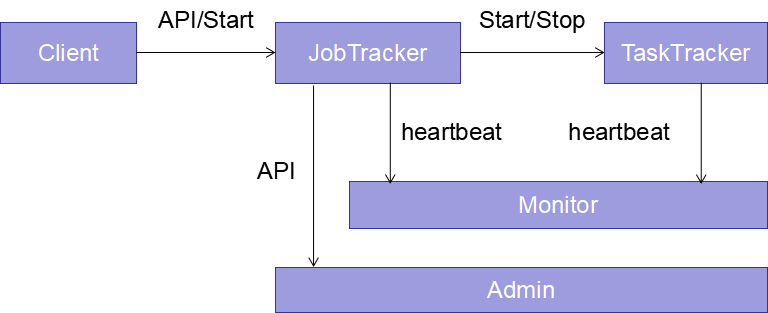
# 数据分发

## 3.1 规则引擎

实现数据过滤、数据分发功能，为了接收不同租户任务调度请求，采用高可用的TaskScheduler来实现。

### 3.1.1 基于高可用的TaskScheduler任务调度的框架

**（1）任务场景**



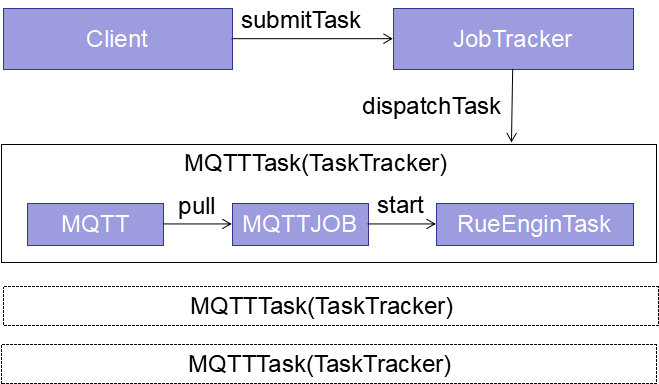
1. 租户选择数据规则（选择数据过滤或分发到目的地）Task1
2. 点击启动，客户端JobClient向任务调度中心JobTracker发送任务调度Task1
3. 调度中心JobTracker接受任务并分配任务给TaskTracker节点，TaskTracker负责执行任务，执行完反馈给JobTacker
4. 对于流式任务（数据流），除非人为向JobTrakcer发送关闭命令停止TaskTracker任务，否则TaskTracker会一直执行

**（2）LTS调度框架**

选择LTS作为规则引擎的数据调度框架，是因为LTS(light-task-scheduler)用于解决分布式任务调度问题，支持实时任务，定时任务和Cron任务，有较好的伸缩性，扩展性，健壮稳定性。

具体网址：https://gitee.com/hugui/light-task-scheduler

### 3.1.2 基于TaskScheduler租户MQTT调度模型



结合实际规则引擎具体任务，任务场景如下：

1. 设备数据通过MQTT协议上传至云平台；
2. 用户在页面选择MQTT发送地址和过滤规则，封装后调用客户端发送请求；
3. 客户端提交任务给JobTracker
4. JobTracker分发任务给其中一个节点的TaskTracker
5. TaskTracker接受任务后，将任务分配给MQTTJob，MQTTJob负责从MQTT的Topic里面拉取数据；
6. 拉取后的数据经过RuleEngineTask规则引擎任务进行处理；

### 3.1.3 基于责任链的RuleEngineTask设计模式

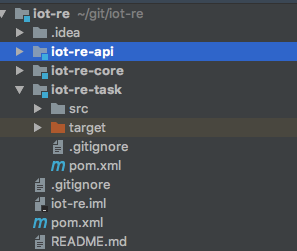
RuleEngineTask是整个规则引擎数据处理的核心类，前面主要负责设备数据如何经过LTB的处理交给不同的节点去处理，最后都是交给每个节点的RuleEngineTask来完成任务，每个节点上的RuleEngineTask都一样。



1. RuleEnineTask是一个标准的责任链模式，同样的数据从一条链上穿过，交给不同的数据过滤器去处理；
2. ShuffleFilter是典型的数据过滤器，对Json格式的数据进行处理，支持类SQL语句，例如：select f1,f2,f3 from t where tmp>50 and kw<100
3. DBFilter

数据库过滤器，将数据分发到不同的数据库，例如：TSDBFilter，分发给时序数据库，MongoFilter，分发给MongoDB数据库。DBFilter不支持用户自定义转发规则，Json数据会按照设备分表，每个Json字段对应数据库的表字段。一个设备对应一张表，一个Json对应一个表子段。

### 项目结构



1. **iot-re-api**

iot-ruleengine项目对外暴露的api，client的所有调用的api操作都在这个工程，相当于LTS的Client。

1. **接口**

启动任务api/1.0/Start

停止任务api/1.0/Stop

重启任务api/1.0/Restart

1. **iot-re-core**

项目核心包，一些SQL过滤器、DB过滤器、工具类会放在这个工程。

**（3）iot-re-task**

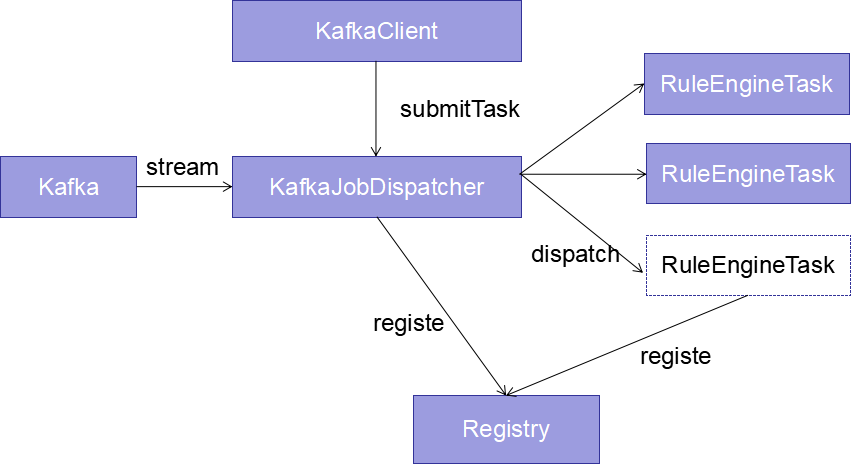
相当于LTS的TaskTracker。

### 启动工程

1. 配置lts数据库和zookeeper地址，详见lts文档
2. 启动Zookeeper
3. 启动Client，iot-re-api，run IOTReApplication
4. 启动JobTracker，进入light-task-scheduler/dist/bin目录，./lts-jobtracker.sh lts start
5. 启动lts-admin，检测lts，dist/bin/lts-admin.sh start
6. 启动TaskTracker，iot-re-task，run IOTReTaskApplication，

### 基于Dubbo的平台Kafka动态数据分发总体架构

如果是从平台的Kafka直接拉取数据，需要从全网设备数据中取筛选租户所需要的数据，数据处理量极大，架构上选择Dubbo分布式调度来处理，将全网数据负载到不同的节点上进行处理。



# 数据存储

**4.1 时序数据库TSDB**

基于容器化实现数据存储、监控、限流等功能。

1. 数据存储格式
2. 数据监控
3. 数据限流
4. 基于Docker的配置

**4.2 航天kafka**

基于容器化实现用户数据订阅功能，通过将MQTT数据发送给Kafka，用户可以自行订阅动态数据。

# 物分析工具

**5.1 Spark引擎**

实现Spark Stream、Spark SQL、Spark、MLib全套解决方案，为用户提供基于Spark的内存分析服务。

实现5个节点以内的Spark容器化引擎。

1. 基于Kubenetes的Spark编排
2. Spark可视化监控

# 建模工具

## 6.1 算法建模

基于美林的算法建模工具解决方案

1. 历史数据分析

基于租户或平台的历史数据分析

1. 平台历史数据获取
2. 平台租户数据获取
3. 动态数据分析
4. 基于租户TSDB的数据实时分析
5. 数据接口格式

## 6.2 报表建模

基于美林的BI报表解决方案

1. 静态数据分析

基于设备中心API的静态数据分析

1. 动态数据分析
2. 基于租户的TSDB的数据分析
3. 数据接口格式

## 6.3 组态建模

基于多比的组态解决方案

1. 设备静态数据

调用设备中心API建立设备模型

1. 设备动态数据
2. 基于租户TSDB或Kafka的动态数据展示
3. 数据接口格式

# 设备中心

所有用户设备在设备中心创建、实例化，设备静态数据都存储在设备中心。

1. 元数据管理

元数据包括型号表、设备表、客户表等。

1. 消息同步
2. API接口

对外暴露的接口说明

（四）第三方接入流程