RE 文档

2018-02-07

百度云

cloud.baidu.com

目录

第1章　产品描述

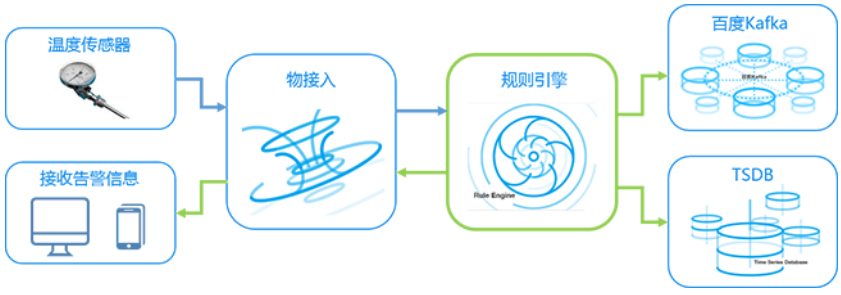
概述

规则引擎并不是一个全新的概念，在传统软件业中已经有相关的产品。在传统商业管理软件中，由于市场要求业务规则变化频繁，IT系统必须依据业务规则的变化而快速、低成本的更新。为了达到该目的，要求业务人员能够直接管理IT系统中的规则而不需要开发人员的参与，这就是规则引擎曾经在传统软件中的功能。

类似地，在物联网中，由于数据量巨大，业务规则可能多种多样，也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，如监控厂区的温度湿度监控点，每十分钟都会有温度和湿度数据传往云端；对于这些数据，我们往往希望它们发挥不同的作用，例如以下应用场景：

- 实时告警异常的数据；
- 两个小时内的温度最大最小和均值等；
- 将全部的数据做冷备份以便查询；
- 对去除异常数据之后的正常数据做数据分析和预测。

规则引擎就是通过灵活的设定规则，将设备传上云端的数据，送往不同的数据目的地（如时序数据库TSDB、Kafka、对象存储BOS等）以达到不同的业务目标。



产品功能

- 构建应用程序

  收集和处理设备生成的数据并根据预定规则执行操作。

- 并行操作

  同一个规则可以应用至一台或多台设备，并且可以并行执行一个或多个操作。

- 多方转发

  将收集到的消息灵活转发到时序数据库TSDB、百度Kafka和对象存储中存储。

- SQL规范

  可以在控制台使用类似SQL的语句编写规则，规则设置简单，符合开发习惯，并且给开发者尽可能大的灵活性。

- 规则验证

  支持对已经创建的规则进行验证，检查数据筛选结果是否符合预期。

产品优势

- 多种服务支持

  支持将消息转发到时序数据库、Kafka、对象存储，无缝对接MapReduce、机器学习、云推送等各种服务。

- 灵活易用

  规则设置灵活简单，一键启停规则，支持各种事件和动作。

- 多场景支持

  支持设备与云端之间的转发，设备与设备之间的转发，设备与第三方服务的转发。

- 强大的API

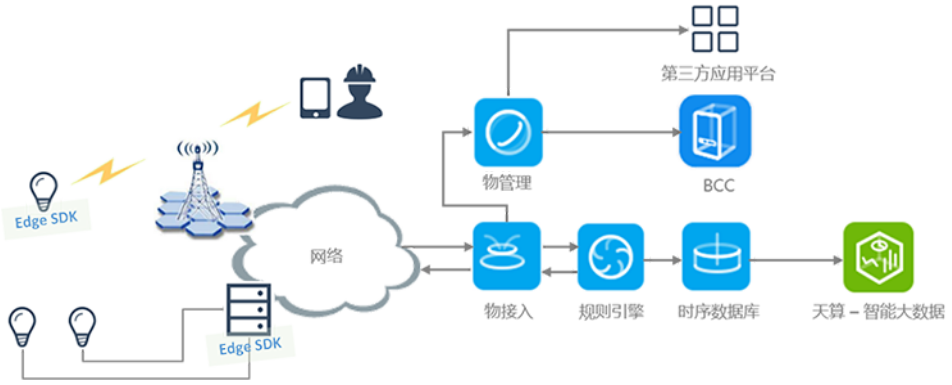
  提供开放标准的API，可通过调用API实现控制台操作，方便第三方应用快速集成云端服务。

应用场景

城市路灯管理属于一个典型的物联网场景，该场景可以复制到很多其它的领域。

传统情况下，路灯管理基本上靠人力，例如：巡查、维护，甚至是靠人工控制路灯开关；而城市路灯数量巨大，分布复杂，对于管理部门来说可能已经投入了大量的人力成本，但仍然会出现故障发现和抢修不及时的情况。同时在传统情况下，由于缺少路灯的运行环境（例如：电压、温度），使用寿命等数据，很难有针对性改进使用方法，进而延长使用寿命，也很难指定准确的维护策略和备件策略。

如下图所示为百度云天工智能物联网平台智慧路灯解决方案架构。



路灯接入可以采用多种方式，例如：3G/4G，电力载波，LPWAN等，无论采用哪种方式，用户只需在接入设备中预装Edge SDK，都能轻松打通路灯和百度云之间的双向安全通道，实现将路灯接入百度云的物接入服务。路灯可以实时将设备状态、电压电流、环境温度、地理坐标等信息发送至物接入服务。同时在工程师的智能终端上预装APP对接百度云，可实时上报工程师的地理位置坐标。

物接入接收到路灯上送的消息后，可将消息分别转发至物管理和规则引擎服务。用户可在规则引擎上制定策略，实现以下操作：

- 当路灯下线或电压电流超过阈值时，检索距离现场最近的维护工程师，并自动发送告警、路灯坐标等信息给指定的维护工程师。工程师的智能终端可对接百度地图服务，自动在地图上显示待维修路灯的位置。
- 将电压电流、环境温度等信息转发至时序数据库，并对接天算大数据平台，用于后续的数据挖掘。

第2章　产品定价

规则引擎暂无收费计划。

第3章　操作指南

3.1　操作准备

3.1.1　注册账号

> 注意：
> 推荐用户使用Firefox或Chrome浏览器执行控制台操作。

在操作专属服务器前，应先完成以下任务：

- 完成百度云账号注册。

3.1.2　操作流程介绍

1. 创建数据源：规则引擎需要通过物接入获取设备端发往云端的消息，因此在配置规则引擎之前，需先配置物接入IoT Hub打通云端和设备端之间的双向通道。物接入主要用于在智能设备与云端之间建立安全的双向连接，实现从设备到云端以及从云端到设备的消息传输。
2. 创建目的地：规则引擎可根据预先设置的规则将消息转发至百度云的其它服务，例如时序数据库、百度Kafka或物接入主题。在配置规则引擎前，建议您先创建好需要用到的其它服务。

    - 百度Kafka

      百度Kafka是基于Apache Kafka的分布式、高可扩展、高通量、多分区和多副本的消息托管服务。百度Kafka封装了Kafka集群细节，并以托管服务形式提供，您可以直接使用百度Kafka创建主题来集成大规模分布式应用，而无需考虑集群运维，仅按照使用量支付处理数据的费用。
    - 时序数据库TSDB

      TSDB是用于管理时间序列数据的专业化数据库。区别于传统的关系型数据库，TSDB针对时间序列数据的存储、查询和展现进行了专门的优化，从而获得极高的数据压缩能力、极优的查询性能，特别适用于物联网应用场景。

- 物接入主题

  物接入主题是物接入服务与前端设备进行消息传播的媒介。规则引擎支持将在一个主题中收到的信息，转发至另一个主题中。如将主题temperature订阅到的消息，筛选出大于阈值的消息，转发至主题alarm中，再用其他客户端订阅alarm客户端主题即可。

3. 创建规则：设置筛选规则、消息来源主题、触发条件和数据目的地。
4. 验证规则：构建Json格式的Payload，对已经创建的规则的筛选结果进行验证。

3.2 创建规则

规则引擎需要通过物接入获取设备端发往云端的消息，因此在配置规则引擎之前，需先配置物接入打通云端和设备端之间的双向通道。有关物接入的配置操作，请参看物接入IoT Hub。

规则引擎从物接入获取到的消息的payload应当是Json格式的，因此在创建规则时，需要使用Json表达式引用。在执行以下操作之前，用户应对Json表达式引用方法有所了解。有关Json表达式的介绍请参看附录-Json表达式

1. 登录百度云官网，点击右上角的“管理控制台”，快速进入控制台界面。
2. 选择“产品服务>规则引擎>规则列表”，进入“规则列表”页面。

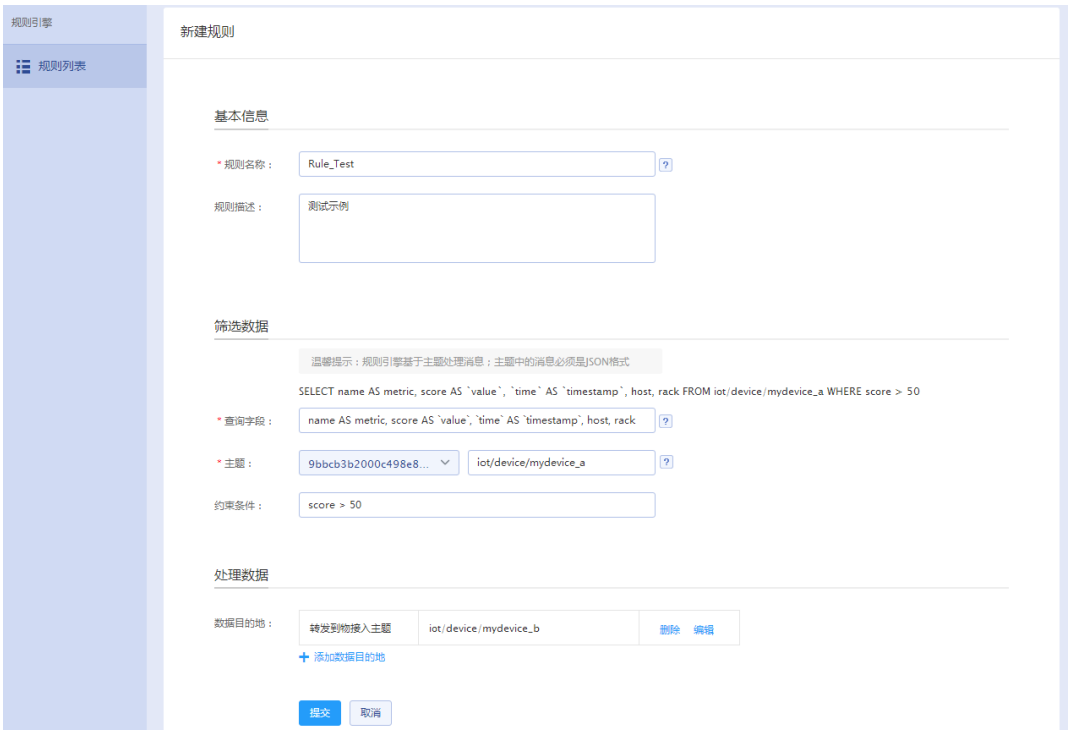
3. 点击“新建规则”，进入规则创建页面，填写以下配置信息：

   - 查询字段：从消息中筛选出来用于后续操作（如：转发至TSDB或百度Kafka）的数据。如果填写“*”，则将原样转发payload中的所有数据至目的地。

     如果将消息转发至TSDB，需要预先对消息格式进行调整，具体操作请查看将消息转发至TSDB。
   - 主题：消息来源的物接入主题。
   - 约束条件：后续操作的触发条件。缺省情况下，转发每一条消息至目的地。

   其中，查询字段和约束条件中支持的操作符如下：

运算符	语义	示例
JSON扩展表达式	JSON的属性访问	info.contact.tel，关于Json表达式的更多内容，请参看附录
=	相等比较	age = 9
<>	不等	name <> ‘abc’
AND	逻辑与	age = 9 AND name <> ‘abc’
OR	逻辑或	age = 9 OR name <> ‘abc’
()	括号	age =9 AND (name <> ‘abc’ OR score > 60.0)
+,-,*,/	数值的加减乘除	2+3, age - 1, score * 100.0, age / 3
<, <=, >, >=	比较操作符	age < 10, age <= 9, score > 60.0, score >= 60.0
in	集合操作	WHERE name in (‘jack’, ‘jim’)
SQL函数	标准的SQL函数	如：ceil, floor等，常见SQL函数，请参看附录
CURRENT_TIMESTAMP	返回当前时间戳（毫秒），即UTC时间	SELECT CURRENT_TIMESTAMP AS TS
LOCALTIMESTAMP()	返回本地时间戳（毫秒），即UTC+8	SELECT LOCALTIMESTAMP AS localTS
NEWID()	返回一个guid（global unique identifier，全球唯一标识），规则引擎会为每一条消息新增一个guid	SELECT NEWID() AS id from t
clientid()	返回消息的发送者id	SELECT clientid() FROM…
clientip()	返回消息发送者的IP	SELECT clientip() FROM…
topic()	返回消息所在的主题	SELECT topic() FROM…
qos()	返回消息发送所用的QoS	SELECT qos() FROM…



4. 点击“添加数据目的地”，新增一个数据目的地，此处可添加多个。

   规则引擎支持将数据转发至“时序数据库TSDB”、“百度Kafka”或“其它物接入主题”。

   将数据转发至“其它物接入主题”是，仅支持将数据转发至同一个endpoint（即消息来源的物接入主题所在的endpoint）下的其它主题，不支持转发至其它endpoint。

   如果将数据转发至TSDB，需要对数据格式进行调整，具体操作请参看将消息转发至TSDB。

5. 返回“规则列表”页面，查看已经创建的规则。

3.3 验证规则

用户可以自己构建Json格式的Payload，对已经创建的规则的筛选结果进行验证。在执行以下操作前，用户应先新建规则。

1. 选择“产品服务>规则引擎>规则列表”，进入“规则列表”页面，查看已经创建的规则。

2. 找到指定的规则，点击规则名称，进入规则详情页面；点击“规则验证”，验证已经创建的规则。

3. 在“输入数据”中构建Json格式的原始数据，选择数据目的地并点击“验证”，此时系统将根据已经创建的规则自动输出筛选后的数据。

3.4 将消息转发至TSDB

3.4.1 通过简单规则调整数据格式

> 注意
> 以下方法仅针对value存放在消息中的情况，如果value存放在数组中，请参看通过“\TSDB\META”调整数据格式。

TSDB数据库要求数据必须包含metric、value和timestamp三个字段，以及额外一个或多个数据作为tag。如果原始消息中不包含这些字段，在将数据转发至TSDB前，需要对数据格式进行调整，如下所示：

原始数据

> 注意
> 原始数据中的“score”: 51为整型，数据没有放在引号“”中。如果原始数据为“score”: “51”，此时该数据为字符串类型。字符串类型数据写入TSDB后将无法在查看面板中生成图表。可以通过在规则中使用CAST函数对数据类型进行修改，具体内容请参看为什么查询面板中生成图表仍为空？。

```
{
    ”time”:1465376157007,
    ”name”:”cpu_idle”,
    ”score”:51,
    ”host”:”server1”,
    ”rack”:”rack1”,
    ”other”:”something”
}
```

规则引擎设置

- 查询字段：name AS metric, score AS \value, \time\ AS \timestamp, host, rack
  由于value为SQL的关键字，使用时需要加反单引号（重音符）。
- 主题：消息来源的物接入主题。
- 约束条件：score > 50

输出数据

```
{
    ”metric”:”cpu_idle”,
    ”_value”:51,
    ”_timestamp”:1465376157007,
    ”host”:”server1”,
    ”rack”:”rack1”
}
```

3.4.2 通过“\TSDB\META”调整数据格式

> 注意
> 以下方法仅针对value存放在数组中的情况，如果value存放在消息中，请参看通过简单规则调整数据格式。

TSDB提供了一个数据描述的格式“\TSDB\META”，方便用户对消息格式进行转化。“\TSDB\META”描述了用户的数据在哪里(json path)，时间戳是哪个字段、时间戳的格式，metric，value等。

“\TSDB\META”仅应用于value在数组中的场景。

消息格式

```
”_TSDB_META”:{必选字段，指定数组位于消息哪个字段
    //
    ”data_array”:”path.to.data.array”,必选字段，指定数组中那个字段是
    //字段value
    ”value_field”:”json.path.to.value.field.in.data.array.object”,可选字段，指定数组中的哪个字
        段是

    //timestamp
    ”point_time”:”json.path.to.timestamp.within.data.array.object”,可选字段

    //，指定消息中的哪个字段是timestamp并且将此时间戳应用到每个数据点上。,说明：如果
    //和都没有指定，则用系统默认时间。如果两者都指定了，则优先使
        用point_timeglobal_timepoint_time
    ”global_time”:”json.path.to.timestamp.in.the.outer.object”,可选字段，指定时间戳的格
        式，例如

    //,”yyyy−MM−dd hh:mm:ssZ。如果时间戳是整数的秒或者毫秒，则可以省略。”
    ”time_format”:”yyyy−MM−dd hh:mm:ssZ”,可选字段，指定数组中的哪个字段是

    //metric
    ”point_metric”:”json.path.to.metric.field.within.data.array.object”,可选参数，指定消息中
        的哪个字段是

    //metric. 说明:和必须有一个，优先使用point_metricglobal_metricpoint_metric
    ”global_metric”:”json.path.to.metric.in.the.outter.object”,可选字段，指定数组中的哪些字
        段是

    //tag
    ”point_tags”:{
        ”tag1”:”json.path.to.tag.field.within.data.array.object”,
        ”tag2”:”json.path.to.tag.field.within.data.array.object”,
        ”tag3”:”json.path.to.tag.field.within.data.array.object”,
       ...
    },可选字段，指定消息中的哪些字段是

    //tag说明：如果
    //和都存在，则系统会把两者都使用起来，global_tagspoint_tags
    ”global_tags”:{
        ”tag1”:”json.path.to.tag.in.the.outter.object”,
        ”tag2”:”json.path.to.tag.in.the.outter.object”
        ...
  }
}
```

举例

以物解析的数据格式为例，物解析的消息如下：

```
{
```

```
    ”bdModbusVer”:1,
    ”gatewayid”:”db829fea-0706-4048-9ea2-9373e441a3a5”,
    ”trantable”:”8c389fad-7ccd-481b-8333-11c98cea071a”,
    ”modbus”:{
      ”request”:{
        ”functioncode”:3,
        ”slaveid”:1,
        ”startAddr”:0,
        ”length”:3
      },
     ”response”:”070200000000”,
     ”parsedResponse”:[
    {
        ”desc”:”bit8AddBit8”,
        ”type”:”INT”,
        ”unit”:””,
        ”value”:”9”,
        ”errno”:0
    }
    ],
    ”error”:null
  },
  ”timestamp”:”2016-10-21 07:50:01-0700”
}
```

对应的\TSDB\META如下：

```
”_TSDB_META”:{
    ”data_array”:”modbus.parsedResponse”,指定//所在的数组value
    ”value_field”:”value”,                                    指定数组中的//字段value

    ”global_time”:”timestamp”,                          由于时间戳存在于消息中，因此使用//；如果时间戳存在于数组中，则使用global_timepoint_time

    ”time_format”:”yyyy-MM-dd hh:mm:ssZ”,
    ”point_metric”:”desc”,                                    由于度量值存在于数组中，因此使用//；如果度量值存在于消息中，则使用point_metricglobal_metric可选字段，指定消息中的哪些字段是

    //tag
    ”global_tags”:{
      ”tag1”:”modbus.request.functioncode”,
      ”tag2”:”gatewayid”
    }
}
```

对应的规则引擎语句

```
SELECT *,
    ’modbus.parsedResponse’ AS _TSDB_META.data_array,
    ’value’ AS _TSDB_META.value_field,
```

```
    ’timestamp’ AS _TSDB_META.global_time,
    ’yyyy−MM−dd hh:mm:ssZ’ AS _TSDB_META.time_format,
    ’desc’ AS _TSDB_META.point_metric,
    ’modbus.request.functioncode’ AS _TSDB_META.global_tags.tag1,
    ’gatewayid’ AS _TSDB_META.global_tags.tag2
FROM topic
```

3.4.3 通过规则引擎写对象下面的所有属性

> 注意
> 以下方法针对的情况为，一个对象下面包含多个点位，其中Key作为TSDB的metric，Value作为TSDB的value的情况，其中Timestamp，tags共享

消息格式

```
{
    ”device”:”20170001”,
    ”ts”:1498790080,
    ”data”:{
        ”temperature”:36,
        ”speed”:98,
        ”voltage”:219.3
            }
}
```

规则引擎设置

- 查询字段：*，‘data’ AS TSDBMETA_v2.metric_nodes.node1，‘ts’ AS TSDBMETA_v2.ts，‘device’ AS TSDBMETA_v2.tags.tag1

输出数据

```
{
    ”device”:”20170001”,
    ”ts”:1498790080,
    ”data”:{
        ”temperature”:36,
        ”speed”:98,
        ”voltage”:219.3
    },
    ”_TSDB_META_v2”:{
        ”metric_nodes”:{
            ”node1”:”data”
        },
        ”ts”:”ts”,
        ”tags”:{
            ”tag1”:”device”
```

```
        }
    }
}
```

3.4.4 写TSDB多个域

> 注意
> 此方法用于数据写入TSDB的一个metric的多个域的情形，同一个点位包含多个值，metric，timestamp，tags共享

消息格式

```
{
    ”ts”:1498097858,
    ”name”:”location”,
    ”model”:”mx2017”,
    ”data”:{
        ”lng”:121.34,
        ”lat”:30.12
    }
}
```

规则引擎设置

- 查询字段：*，‘ts’ AS TSDBMETA_v3.ts，‘name’ AS TSDBMETA_v3.metric，‘model’ AS TSDBMETA_v3.tags.tag1，‘data.lng’ AS TSDBMETA_v3.fields.field1，‘data.lat’ AS TSDBMETA_v3.fields.field2

输出数据

```
{
    ”ts”:1498097858,
    ”name”:”location”,
    ”model”:”mx2017”,
    ”data”:{
        ”lng”:121.34,
        ”lat”:30.12
    },
    ”_TSDB_META_v3”:{
        ”ts”:”ts”,
        ”metric”:”name”,
        ”tags”:{
            ”tag1”:”model”
        },
        ”fields”:{
            ”field1”:”data.lng”,
            ”field2”:”data.lat”
```

```
        }
    }
}
```

3.4.5 多种方法组合示例

以上4种方法可以独立工作，互不影响

输入数据

```
{
    ”device”:”20170001”,
    ”ts”:1498790080,
    ”data”:{
        ”temperature”:36,
        ”speed”:98,
        ”voltage”:219.3
    },
    ”name”:”rps”,
    ”data2”:[
        {
            ”time”:1498790081,
            ”v”:7200
        }
    ]
}
```

规则引擎设置

- 查询字段：*，‘data’ AS TSDBMETA_v2.metric_nodes.node1，‘ts’ AS TSDBMETA_v2.ts，‘device’ AS TSDBMETA_v2.tags.tag1，‘data2’ AS TSDBMETA.data_array，‘v’ AS TSDBMETA.value_field，‘name’ AS TSDBMETA.global_metric，‘time’ AS TSDBMETA.point_time，‘device’ AS TSDBMETA.global_tags.tag1

输入数据

```
{…

    ”_TSDB_META_v2”:{
        ”metric_nodes”:{
            ”node1”:”data”
        },
        ”ts”:”ts”,
        ”tags”:{
            ”tag1”:”device”
        }
    },
```

```
    ”_TSDB_META”:{
        ”data_array”:”data2”,
        ”value_field”:”v”,
        ”global_metric”:”name”,
        ”point_time”:”time”,
        ”global_tags”:{
            ”tag1”:”device”
        }
    }
}
```

3.4.6 延伸资料参考

规则引擎写TSDB支持多域(field)

规则引擎写TSDB语法：写整个JSON对象

物解析服务产出消息增加metrics对象

3.5 通过AS输出嵌套的JSON结构

以下通过一个示例介绍如何输出嵌套的JSON结构。

原始数据

```
{
    ”name”:”china”
}
```

规则引擎设置

- 查询字段：name AS info.name
- 主题：消息来源的物接入主题。
- 约束条件：无

输出数据

```
{
    ”info”:{
        ”name”:”china”
    }
}
```

3.6 将消息动态转发至MQTT主题

规则引擎可以从接收到的消息中自动提取MQTT主题，并将消息转发至该主题，具体操作如下所示：

原始数据

```
{
    ”msg”:”hello”,
    ”info”:{
        ”name”:”chiller−2016”
    }
}
```

规则引擎设置

- 查询字段：select msg from t
- 主题： ‘chillers/’ || info.name
  “||” 是SQL 92标准中连接字符串的操作
- 约束条件：无

输出数据

规则引擎将以下数据转发至“chillers/chiller-2016”这个主题中：

```
{
    ”msg”:”hello”
}
```

3.7 支持函数计算

规则引擎增加了一类新的目的地：函数计算(CFC)。规则引擎将MQTT消息，作为函数的输入，调用函数并等待函数的返回，将函数返回转入另外一个MQTT主题。用户可以对消息进行自定义处理，或者把消息转发到自己的服务器，等等。

函数计算(CFC)是百度提供的一个类似lambda的平台，无需维护任何服务器就可以运行自己的代码，平台可以来维护服务器，并且自动扩容。用户只需要按执行多少付费，没有进行调用，则不会产生费用。

举个例子，假如你的设备往云端上传的消息格式如下：

```
{

    ”ts”:1513654436,

    ”data”:[1,2,3]
```

```
}
```

并且希望把消息中的data数组写入时序数据库(TSDB)。因为规则引擎不支持data数组这种没有key的形式，因此无法直接通过规则引擎写入TSDB。因此我们需要函数计算来将数据先进行变化，将data数组转化成另外一种格式，然后再通过规则引擎写TSDB:

新建一个函数计算(CFC)

记函数名为array_convert，代码如下：

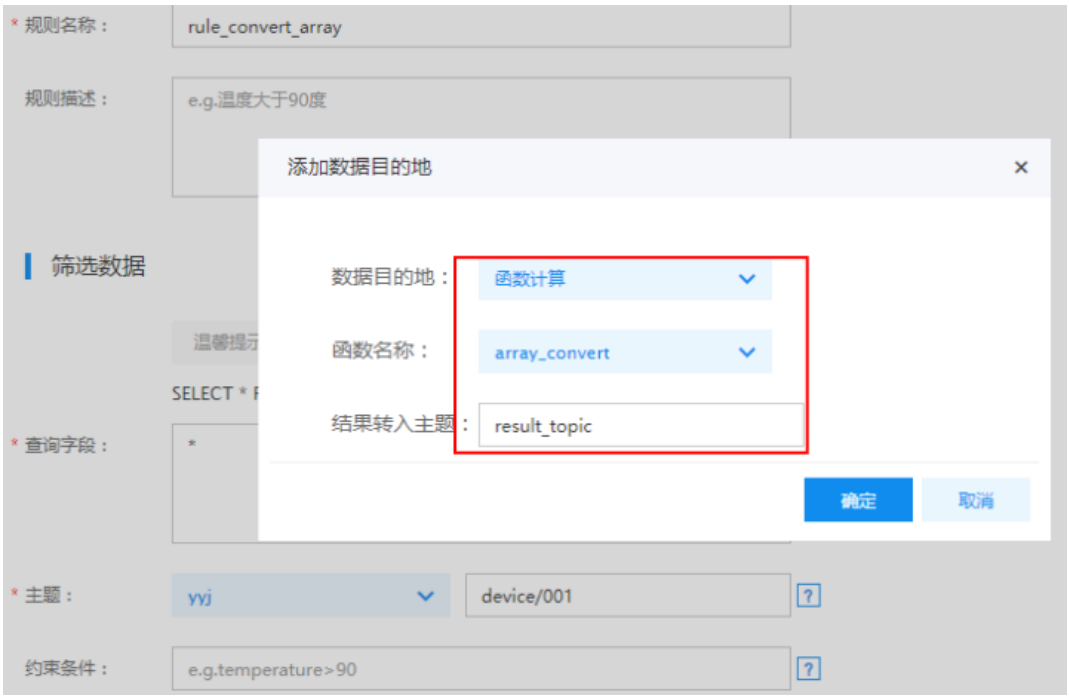
```
exports.handler = (event, context, callback) => {
        var data_array = [];
        event.data.forEach(function(e) {
                var obj = {};
                obj.name = e;
                data_array.push(obj);
});
        event.data_array = data_array;
        callback(null, event);
};
```

函数计算地址：https://console.bce.baidu.com/cfc/#/cfc/functions

新建规则引擎

查询字段：*

点击添加数据目的地，选择函数计算，函数名选择array_convert，结果转入主题填例如:result_topic。



这样，你的设备上报的数据经过规则引擎和函数计算的处理，处理后的结果转入result_topic，转化后的消息格式如下：

```
{
    ”ts”:1513654436,
    ”data”:[1, 2,  3],
    ”data_array”:[
        {
            ”name”:1
        },
        {
            ”name”:3
        },
        {
            ”name”:3
        }
    ]
```

```
}
```

之后就可以通过规则引擎，将主题result_topic中的消息的data_array数组，按照写TSDB的第二种方式把数据写入TSDB了。规则引擎写TSDB的4种方式：http://iot-modbus.gz.bcebos.com/tut/ruleengine2tsdb_tutorial.mp4

需要说明的是：

A) 函数计算期望的输入是JSON格式的，因此你的原始消息一定是JSON的；

B) 结果转入主题可以不填，那样的话，函数计算返回的内容会被丢弃；

C) 函数计算目前还处于白名单测试阶段，需要申请才能开通，请参考文档 https://cloud.baidu.com/doc/CFC/GettingStarted.html#.9A.CE.FA.E5.FB.B0.B5.ED.90.C3.9A.75.57.24.89.BC；

D) 函数计算目前只支持北京区域

3.8 附录

3.8.1 Json表达式

Json简单说就是JavaScript中的对象和数组，所以这两种结构就是对象和数组两种结构，通过这两种结构可以表示各种复杂的结构。

对象：对象在js中表示为“{}”括起来的内容，数据结构为{key：value,key：value,…}的键值对的结构，在面向对象的语言中，key为对象的属性，value为对应的属性值，所以很容易理解，取值方法为对象.key 获取属性值，这个属性值的类型可以是 数字、字符串、数组、对象几种。

3.8.2 常用SQL函数

- ceil(value)：返回不小于value的最小整数值。
- floor(value)：与 ceil()相反，产生小于或等于指定值value的最小整数。
- like：用于在 WHERE 子句中搜索列中的指定模式。举例如下：

```
”LastName”:”Bush”
”FirstName”:”George”
”Address”:”Fifth Avenue”
”City”:”New York”
}
```

从“Persons”表中选取居住在以“N”开始的城市里的人：

“%” 可用于定义通配符（模式中缺少的字母）。

- cast：用于将某种数据类型的表达式显式转换为另一种数据类型。

  语法：CAST (expression AS data_type)

  参数说明：

  - expression：任何有效的SQL Server表达式。
  - AS：用于分隔两个参数，在AS之前的是要处理的数据，在AS之后是要转换的数据类型。
  - data\type：目标系统所提供的数据类型，包括bigint和sql\variant，不能使用用户定义的数据类型。

- mod(value1, value2)：返回一个value1除以value2的余数。
- abs(value)：返回指定值的绝对值。
- power(value1, value2)：返回value1的value2次方根。
- exp(value)：返回一个数字e的value次方根。

第4章　API参考

4.1　介绍

4.1.1　调用方式

概述　规则引擎API的设计采用了Restful风格，每个API功能（也可以称之为资源）都使用URI（Universal Resource Identifier）来唯一确定。对资源的请求方式是通过向资源对应的URI发送标准的 HTTP 请求，比如 GET、PUT、POST等，同时，请求需要遵守签名算法，并包含约定的请求参数

通用约定

- 所有编码都采用UTF-8
- 日期格式采用yyyy-MM-dd方式，如2015-08-10
- 时间格式采用UTC格式：yyyy-MM-ddTHH:mm:ssZ, 如2015-08-20T01:24:32Z
- Content-type为application/json; charset=UTF-8
  - object类型的key必须使用双引号（“）括起来
  - object类型的key必须使用lowerCamelCase表示

头域（Header）	是否必须	说明
Authorization	必须	包含Access Key与请求签名
Host	必须	包含API的域名
x-bce-date	必须	表示时间的字符串，符合时间格式要求
Content-Type	可选	application/ json; charset=utf-8

公共请求头

头域（Header）	说明
Content-Type	只支持JSON格式，application/ json; charset=utf-8
x-bce-request-id	规则引擎后端生成，并自动设置到响应头域中

公共响应头

响应状态码　返回的响应状态码遵循RFC 2616 section 6.1.1

- 1xx: Informational - Request received, continuing process.
- 2xx: Success - The action was successfully received, understood, and accepted.
- 3xx: Redirection - Further action must be taken in order to complete the request.
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled.
- 5xx: Server Error - The server failed to fulfill an apparently valid request.

请求消息体格式（HTTP Request Body）　规则引擎服务要求使用JSON格式的结构体来描述一个请求的具体内容。

示例

以下是一个标准的写入data point时的请求消息体格式：

```
{
    ”datapoints”:[{
        ”metric”:”cpu_idle”,
        ”tags”:{
            ”host”:”server1”,
            ”rack”:”rack1”
        },
        ”timestamp”:1465376157007,
        ”value”:51
    }]
}
```

请求返回格式（HTTP Response）　规则引擎服务均采用JSON格式的消息体作为响应返回的格式。

示例

以下是一个标准的获取metric列表的请求返回：

```
{
    ”metrics”:[
        ”cpu_idle”,
```

```
        ”mem_used”
    ]
}
```

通用错误返回格式 当调用接口出错时，将返回通用的错误格式。Http的返回状态码为4xx或5xx，返回的消息体将包括全局唯一的请求、错误代码以及错误信息。调用方可根据错误码以及错误信息定位问题，当无法定位到错误原因时，可以发工单联系百度技术人员，并提供requestid以便于快速地帮助您解决问题。

消息体定义

参数名	类型	说明
requestId	String	请求的唯一标识
code	String	错误类型代码
message	String	错误的信息说明

错误返回示例

”requestId”:”47e0ef1a-9bf2-11e1-9279-0100e8cf109a”,

”code”:”NoSuchKey”,

”message”:”The resource you requested does not exist”

```
”requestId”:
    ”47e0ef1a-9bf2-11e1-9279-0100e8cf109a”,”code”:”NoSuchKey”,”message”:”The
    resource you requested does not exist”
```

签名认证 规则引擎 API会对每个访问的请求进行身份认证，以保障用户的安全。安全认证采用Access Key与请求签名机制。Access Key由Access Key ID和Secret Access Key组成，均为字符串，由百度云官方颁发给用户。其中Access Key ID用于标识用户身份，Access Key Secret 是用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密。

对于每个HTTP请求，用户需要使用下文所描述的方式生成一个签名字符串，并将认证字符串放在HTTP请求的Authorization头域里。

签名字符串格式

bce-auth-v{version}/{accessKeyId}/{timestamp}/{expireTime}/{signedHeaders}/{signature}

其中：

- version是正整数，目前取值为1。
- timestamp是生成签名时的时间。时间格式符合通用约定。
- expireTime表示签名有效期限，单位为秒，从timestamp所指定的时间开始计算。

- signedHeaders是签名算法中涉及到的头域列表。头域名字之间用分号（;）分隔，如host;x-bce-date。列表按照字典序排列。当signedHeaders为空时表示取默认值。
- signature是256位签名的十六进制表示，由64个小写字母组成，生成方式由如下签名生成算法给出。

签名生成算法　有关签名生成算法的具体介绍，请参看鉴权认证机制。

多区域选择　Region代表着一个独立的地域，是百度云中的重要概念，请参考区域选择说明。百度云中的服务除了极少数如账号服务全局有效之外，绝大部分服务都是区域间隔离的。每个区域的服务独立部署互不影响。服务间共享数据需要通过显式拷贝完成。在API中引用区域必须使用其ID。目前规则引擎服务的区域是“华南-广州”，目前支持http和https调用。

区域	ID	域名	协议
华南-广州	gz	re.iot.gz.baidubce.com	http和https

4.2 接口说明

4.2.1 创建规则

方法	API	说明
POST	/v1/rules	创建规则

请求参数

参数名	说明	示例
endpoint	需要规则引擎处理的mqtt主题对应的服务实例	myendpoint
from	需要规则引擎处理的mqtt主题	topic/sensor01
description	规则描述	过滤传感器温度过高消息的规则
destinations	处理后的消息写往的目的地数组（TSDB, KAFKA,另一个MQTT主题）	见下表

参数名	说明	示例
select	输出哪些字段，SQL语法	*, name, data.temperature name| 规则名称| sensor_warn where| 消息过滤条件，SQL语法| data.temperature>90

其中，destinations数组中，每个元素包含如下2个字段：

参数名	说名	示例
kind	目的地类型，可能取值:MQTT, KAFKA, TSDB, BOS	MQTT
value	对于MQTT: value是目的地MQTT主题对于KAFKA: value是目的地KAFKA主题对于TSDB: value是目的地TSDB数据库的URL对于BOS: value是目的地BOS的bucket，如bos:// mybucket对于MQTT_DYNAMIC: value是一个SQL SELECT子句，用于从原始消息里面选择字段作为目的地主题，如: dest.topic	

返回参数

参数名	说明	示例
endpoint	等同传入参数endpoint	endpoint1
state	规则的状态（ENABLED, DISABLED)	ENABLED
from	等同传入参数from	mytopic1
description	等同传入参数description	desc1
accountUuid	当前账号的account uuid	124343-adef-132232
destinations	类似传入参数destinations，同时加上id, ruleid信息	

参数名	说明	示例
select	等同传入参数select	name, * name|等同传入参数name|rulename where|等同传入参数where| abc > 8 AND name like ‘%yyj%’ uuid|系统为规则分配的唯一id|764736473643

请求示例

```
{
    ”endpoint”:”endpoint1”,
    ”from”:”mytopic1”,
    ”description”:”desc1”,
    ”destinations”:[
        {
            ”value”:”dest_topic1”,
            ”kind”:”MQTT”
        },
        {
            ”value”:”kafka_topic1”,
            ”kind”:”KAFKA”
        },
        {
            ”value”:”TSDB_SERVER_URL”,
            ”kind”:”TSDB”
        }
    ],
    ”select”:”name, *”,
    ”name”:”rulename”,
    ”where”:”abc > 8 AND name like ’%yyj%’”
}
```

响应示例

```
{
    ”endpoint”:”endpoint1”,
    ”state”:”ENABLED”,
    ”from”:”mytopic1”,
    ”description”:”desc1”,
    ”accountUuid”:”baidu”,
    ”destinations”:[
        {
            ”uuid”:”6653da99-bf9a-4e35-ba4f-997e000a699f”,
            ”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
            ”value”:”dest_topic1”,
            ”kind”:”MQTT”
        },
```

```
        {
            ”uuid”:”e03264b7-4e94-43d1-b5cd-fb0e51f08ebe”,
            ”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
            ”value”:”kafka_topic1”,
            ”kind”:”KAFKA”
        }
    ],
    ”select”:”name,*”,
    ”name”:”testname3”,
    ”where”:”abc>8 AND name like’%yyj%’”,
    ”uuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”
}
```

4.2.2 列显规则

方法	API	说明
GET	/v1/rules?pageNo=1&pageSize=20	列显规则

响应示例

```
{
”totalCount”:1,
”result”:[
{
”endpoint”:”endpoint1”,
”state”:”ENABLED”,
”from”:”mytopic1”,
”description”:”desc1”,
”destinations”:[
{
”uuid”:”6653da99-bf9a-4e35-ba4f-997e000a699f”,
”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
”value”:”dest_topic1”,
”kind”:”MQTT”
},
{
”uuid”:”e03264b7-4e94-43d1-b5cd-fb0e51f08ebe”,
”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
”value”:”kafka_topic1”,
”kind”:”KAFKA”
}
],
”select”:”name,*”,
”name”:”testname3”,
”createTime”:”2016-08-11T06:39:49Z”,
”where”:”abc>8 AND name like’%yyj%’”,
```

```
" updateTime" : " 2016-08-11T06:39:49Z" ,
" uuid" : " 218cf057-ec2d-415c-8952-cd5cfb641bde"
}
],
" order" : " desc" ,
" orderBy" : " createtime" ,
" pageSize" : 2,
" pageNo" : 1
}
```

4.2.3 为规则添加目的地

方法	API	说明
POST	/v1/destinations	为规则添加目的地

请求参数

参数名	说名	示例
ruleUuid	需要添加目的地的规则id	34882348382
kind	目的地类型，可能取值:MQTT, KAFKA, TSDB	MQTT
value	对于MQTT: value是目的地MQTT主题对于KAFKA: value是目的地KAFKA主题对于TSDB: value是目的地TSDB数据库的URL对于BOS: value是目的地BOS的bucket，如bos:// mybucket对于MQTT_DYNAMIC: value是一个SQL SELECT子句，用于从原始消息里面选择字段作为目的地主题，如: dest.topic	topic/sensor_warn

请求示例

```
{
" ruleUuid" : " 9dd573fe-9246-4eaa-bc2f-2ef88615521d" ,
" value" : " dest_topic1_no3" ,
" kind" : " MQTT"
}
```

4.2.4 从规则中删除目的地

方法	API	说明
DELETE	/v1/destinations/{desinnation_id}	{desinnation_id}为目的地id

请求示例

4.2.5 读取规则详情

方法	API	说明
GET	/v1/rules/{rule_id}	{rule_id}为规则id

请求示例

响应示例


```
{
 ”endpoint”:”endpoint1”,
 ”state”:”ENABLED”,
 ”from”:”mytopic1”,
 ”description”:”desc1”,
 ”destinations”:[
 {
 ”uuid”:”6653da99-bf9a-4e35-ba4f-997e000a699f”,
 ”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
 ”value”:”dest_topic1”,
 ”kind”:”MQTT”
 },
 {
 ”uuid”:”e03264b7-4e94-43d1-b5cd-fb0e51f08ebe”,
 ”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
 ”value”:”kafka_topic1”,
 ”kind”:”KAFKA”
 }
 ],
 ”select”:”name, *”,
 ”name”:”testname3”,
 ”createTime”:”2016-08-11T06:39:49Z”,
 ”where”:”abc > 8 AND name like ’%yyj%’”,
 ”updateTime”:”2016-08-11T06:39:49Z”,
 ”uuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”
}
```

4.2.6 修改规则

方法	API	说明
PUT	/v1/rules/{rule_id}	{rule_id}为拟修改的规则的id

请求参数

参数名	说明	示例
from	需要规则引擎处理的mqtt主题	topic/sensor01
description	规则描述	过滤传感器温度过高消息的规则
select	输出哪些字段，SQL语法	*, name, data.temperature
where	消息过滤条件，SQL语法	data.temperature>90

请求示例

```
{
”from”:”mytopic1_update”,
”description”:”desc1”,
”select”:”name, *, abc”,
”where”:”abc > 8 AND name like ’%yyj% or 1 < 2’”
}
```

响应示例

```
{
”endpoint”:”endpoint1”,
”state”:”ENABLED”,
”from”:”mytopic1_update”,
”description”:”desc1”,
”destinations”:[
{
”uuid”:”6653da99-bf9a-4e35-ba4f-997e000a699f”,
”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
”value”:”dest_topic1”,
”kind”:”MQTT”
},
{
”uuid”:”e03264b7-4e94-43d1-b5cd-fb0e51f08ebe”,
”ruleUuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”,
”value”:”kafka_topic1”,
”kind”:”KAFKA”
```

```
}
],
”select”:”name,*,abc”,
”name”:”testname3”,
”createTime”:”2016-08-11T06:39:49Z”,
”where”:”abc>8 AND name like’%yyj% or 1<2’”,
”updateTime”:”2016-08-11T06:44:13Z”,
”uuid”:”218cf057-ec2d-415c-8952-cd5cfb641bde”
}
```

4.2.7 批量删除规则

方法	API	说明
POST	/v1/rules/batch/delete	批量删除规则

请求参数

参数名|说明|示例 rules|需要删除的规则id数组|“43257469-643e-4a1c-81df-8cca28d96dbc”,“2f74da12-a846-4b70-bf3a-7a22e0c56bc0”

请求示例

```
{
”rules”:[
”43257469-643e-4a1c-81df-8cca28d96dbc”,
”2f74da12-a846-4b70-bf3a-7a22e0c56bc0”
]
}
```

4.2.8 禁用一条规则

方法	API	说明
PUT	/v1/rules/{ruleid}/disable	{ruleid}为需要禁用的规则id

请求示例

4.2.9 启用一条规则

方法	API	说明
PUT	/v1/rules/{ruleid}/enable	{ruleid}为需要启用的规则id

请求示例

第5章　Java SDK文档

5.1　概述

本文档主要介绍规则引擎Java SDK的安装和使用。在使用本文档前，您需要先了解规则引擎的一些基本知识。若您还不了解规则引擎，可以参考产品描述和操作指南。

5.2　安装SDK工具包

运行环境

Java SDK工具包可在jdk1.6、jdk1.7、jdk1.8环境下运行。

方式一：使用Maven安装

在Maven的pom.xml文件中添加bce-java-sdk的依赖：

```
<dependency>
    <groupId>com.baidubce</groupId>
    <artifactId>bce-java-sdk</artifactId>
    <version>{version}</version>
</dependency>
```

其中，{version}为版本号，可以在SDK下载页面找到。

方式二：直接使用JAR包安装

1. 在官方网站下载Java SDK压缩工具包。
2. 将下载的bce-java-sdk-version.zip解压后，复制到工程文件夹中。
3. 在Eclipse右键“工程 -> Properties -> Java Build Path -> Add JARs”。
4. 添加SDK工具包lib/bce-java-sdk-version.jar和第三方依赖工具包third-party/*.jar。

   其中，version为版本号。

SDK目录结构

```
├─
        auth                                    //签名相关类BCE├──
```

```
        http                                    //的通信相关类BCEHttp├──
        internal                                //内部类SDK├──
        model                                   //公用类BCEmodel├──
        services│└──
                ruleengine 规则引擎的服务相关类                    //│
                   ├── model 规则引擎内部                        //，如
                     或modelRequestResponse│
                   └── RuleEngineClient.class 规则引擎客户端入口类      //├──
        util                                    //公用工具类BCE├──
        BceClientConfiguration.class 对          //的的配置BCEHttpClient├──
        BceClientException.class                //客户端的异常类BCE├──
        BceServiceException.class 与             //服务端交互后的异常类BCE├──
        ErrorCode.class                         //通用的错误码BCE└──
        Region.class                            //提供服务的区域BCE
```

5.3 创建RuleEngineClient

用户可以参考如下代码新建一个RuleEngineClient：

```
String accessKey = ”your-access-key-id”;
String secretKey = ”your-secret-access-key”;

RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);
```

5.4 创建/删除/修改规则

5.4.1 创建规则

请参考以下代码创建规则：

```
// 准备相关的参数
String name = ”myRuleName”;                                          // 规则名
    称
String description = ”my rule description”;                        // 规则描述
String mqttEndpoint = ”myendpoint”;                                  // 消息来源的物接
    入地址endpoint

// 选择哪些字段，可以是，字段名，嵌套字段名*
String select = ”name, temperature, info.location”;

// 消息来源于哪个mqtt topic
String fromMqttTopic = ”topic/mysensor01”;
String whereCondition = ”temperature > 90”;                          // 约束条件

// 消息转发至哪个mqtt 或数据库topicTSDB
```

```
String mqttDestinationTopic = ”topic/mysensor01/warn”;
String tsdbDestinationUrl = ”<put your tsdb url here>”;

// 准备，并且填好数据CreateRuleRequest
CreateRuleRequest request = new CreateRuleRequest();
request.setName(name);
request.setDescription(description);
request.setEndpoint(mqttEndpoint);
request.setSelect(select);
request.setFrom(fromMqttTopic);
request.setWhere(whereCondition);

// 准备好规则引擎的目的地，一个，一个MQTTTSDB
List<Destination> destinations = new ArrayList<Destination>();
Destination mqttDestination = new Destination();
mqttDestination.setKind(DestinationKind.MQTT);
mqttDestination.setValue(mqttDestinationTopic);
destinations.add(mqttDestination);

Destination tsdbDestination = new Destination();
tsdbDestination.setKind(DestinationKind.TSDB);
tsdbDestination.setValue(tsdbDestinationUrl);
destinations.add(tsdbDestination);

request.setDestinations(destinations);

// 调用创建规则接口，返回创建好的规则
Rule rule = client.createRule(request);

// 打印新规则的id
System.out.println新规则的(”id=” + rule.getUuid());

// 打印目的地(destination)的id
for (Destination d: rule.getDestinations()) {
        System.out.println(d.getUuid());
}
```

5.4.2 删除规则

请参考以下代码删除规则：

```
// 创建client
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

// 要删除的规则的id
String ruleId = ”<place your rule id here>”;
DeleteRulesRequest request = new DeleteRulesRequest();
```

```
// 删除是个批量操作，可以一次删除多个规则，所以是一个列表ruleIds
List<String> ruleIds = new ArrayList<String>();
ruleIds.add(ruleId);
request.setRules(ruleIds);

client.deleteRule(request);
```

5.4.3 修改规则

修改规则信息 请参考以下代码修改规则：

```
// 创建client
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

UpdateRuleRequest request = new UpdateRuleRequest();
request.setUuid("rule id that need to be updated"); // 被修改的规则ID
request.setDescription("new description"); // 修改规则描述
request.setFrom("new mqtt topic"); // 修改消息来源topic
request.setSelect("*"); // 修改消息选取规则
request.setWhere("temperature > 95"); // 修改约束条件

client.updateRule(request);
```

为规则增加一个目的地 请参考以下代码为规则增加一个目的地：

```
// 创建client
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

Destination destination = new Destination();

// 指定目的地加到哪个中去rule
destination.setRuleUuid("the rule id here");

// 目的地为百度托管目的地KAFKA
destination.setKind(DestinationKind.KAFKA);
String kafkaTopic = "your baidu kafka topic";
destination.setValue("kafkaTopic");
client.createDestination(destination);
```

从规则删除一个目的地 请参考以下代码为规则删除一个目的地：

```
// 创建client
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

// 需要删除的额目的地ID
```

```
String destinationId = ”your destination id”;
client.deleteDestination(destinationId);
```

5.5 查看已经创建的规则

5.5.1 获取所有规则名称和ID

请参考以下代码获取所有规则：

```
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

int pageNo = 1;
ListRuleResponse response = null;
do {
        ListRuleRequest request = new ListRuleRequest();
        request.setPageNo(pageNo++);
        response = client. listRules (request);

        for (Rule rule : response.getResult()) {
                System.out.println(String.format(”name=%s, id=%s”, rule.getName(), rule.
                    getUuid()));
        }
} while (response != null && response.getTotalCount() > response.getPageNo() * response.
    getPageSize());
```

5.5.2 获取一条规则的详情

请参考以下代码获取一条规则的详情：

```
// 创建client
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

// 规则id
String ruleId = ”your rule id here”;
Rule rule = client.getRule(ruleId);

System.out.println(rule.getName());
```

5.6 禁用/启用一条规则

5.6.1 禁用规则

请参考以下代码禁用一条规则：

```
// 创建client
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

// 需要禁用的规则id
String ruleId = ”put your rule id here”;
client.disableRule(ruleId);
```

5.6.2 启用规则

请参考以下代码启用一条规则：

```
// 创建client
RuleEngineClient client = new RuleEngineClient(accessKey, secretKey);

// 需要启用的规则id
String ruleId = ”put your rule id here”;
client.enableRule(ruleId);
```

5.7 版本说明

5.7.1 v0.10.12

首次发布。

第6章　常见问题

6.1　规则引擎常见问题

6.1.1　规则引擎能处理什么类型的消息格式？

规则引擎基于物接入主题处理消息，物接入主题中的消息必须是JSON格式。

如果规则引擎中采用select * 将消息原样转发至另一个物接入主题或者存储服务，则不受JSON格式的限制。

6.1.2　创建规则时，能不能转发至另一个物接入实例的主题？

规则引擎暂时只支持同一个实例下的消息转发。

6.1.3　创建规则时，能不能从多个主题中筛选数据？

支持通配符#，从多个主题中筛选数据，暂时不支持手动输入两个或多个主题。

6.1.4　规则引擎目前支持哪些数据目的地？

规则引擎目前支持转发至物接入主题、存储到时序数据库、存储到百度Kafka。

6.1.5　如果选择转发至物接入主题，所产生的消息量收费吗？

如果转发至物接入主题，遵循物接入的计费规则，每个月前一百万条免费，查看物接入定价。

6.1.6　规则引擎处理的数据有没有格式要求？

通过规则引擎转发至物接入主题、存储到百度Kafka的数据没有格式要求；存储到时序数据库的数据有格式要求，请查看将消息转发至TSDB。

6.1.7 能不能在一个规则中存储至时序数据库的多个metric？

可以在同一个规则中配置，将一条信息同时转发到时序数据库的多个metric。