# 第八讲
# ThingsBoard（V）
# Lecture 8
# ThingsBoard (V)

明玉瑞 Yurui Ming

yrming@gmail.com

# 声明
# Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

# 远程过程调用（RPC）
# Remote Procedure Call (RPC)

- 在分布式计算中，远程过程调用（RPC）是指计算机程序导致的过程（或子例程）在不同的地址空间（通常在共享网络上的另一台计算机上）中执行的机制。程序员使用与本地调用类似的方式，不需要显式编码远程交互的详细信息。RPC通常通过请求-响应消息传递系统实现，可看作客户端—服务器交互的一种形式（调用方是客户端，执行者是服务器）。RPC模型意味着一定程度的位置透明性，即无论调用过程是本地还是远程，但远程调用通常比本地调用慢几个数量级且可靠性低。

  In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction, aka, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. This is a form of client–server interaction (caller is client, executor is server), typically implemented via a request–response message-passing system. The RPC model implies a level of location transparency, namely that calling procedures are largely the same whether they are local or remote, but usually, remote calls are usually orders of magnitude slower and less reliable than local calls.

# 使用RPC
# Using RPC

- ThingsBoard允许用户在服务器端应用程序和设备以远程过程调用（RPC）方式交互，向/从设备发送命令并接收命令执行的结果。在本讲中，我们将了解如下方面：RPC 类型；基本的 RPC 用例；RPC 客户端和服务器端 API；RPC 小部件。

  ThingsBoard allows users to send Remote Procedure Calls (RPC) from server-side applications to devices and vice versa, to send commands to/from devices and receive results of commands execution. In this lecture, we will focus on the following topics: RPC types; Basic RPC use-cases; RPC client-side and server-side APIs; RPC widgets.

- ThingsBoard RPC功能根据远程过程执行的发起方可以分为两种类型：设备发起的RPC和服务器发起的RPC。为了使用更通俗的名称，我们将设备发起的RPC调用命名为客户端RPC，将服务器发起的RPC命名为服务器端RPC。

  ThingsBoard RPC feature may be divided into two types based on the originator of the remote procedure execution: device-originated and server–originated RPC. In order to use more familiar names, we will name device–originated RPC calls as a client-side RPC and server-originated RPC as server-side RPC.

# 客户端RPC
# Client-side RPC

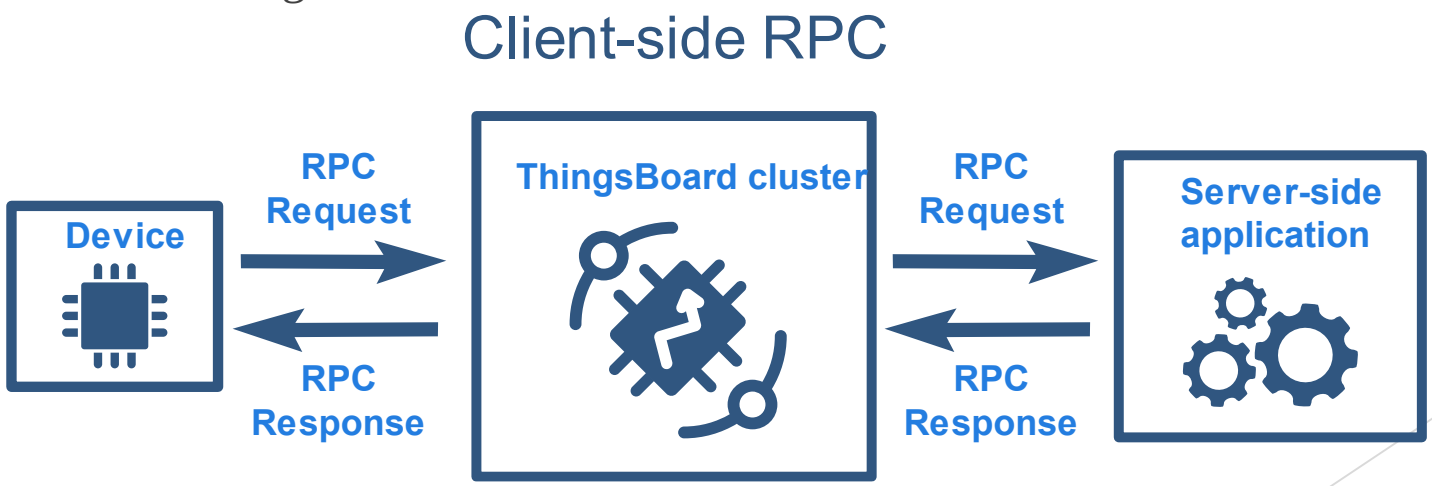
- 客户端 RPC 功能允许用户将请求从设备发送到平台并将响应返回给设备，下面为客户端RPC 调用的典型用例：
  - 灌溉系统通过该平台从在线服务中获取天气预报；
  - 没有系统时钟的受限设备从平台请求当前时间戳；
  - 门禁读卡器向第三方安全系统发送请求，以决定开门和登录访问。

- Client-side RPC feature allows users to send the request from the device to the platform and get the response back to the device. Some typical use cases of the client-side RPC calls are as follows:
  - Irrigation system gets the weather forecast from the online service through the platform.
  - Constrained device without system clock requests the current timestamp from the platform.
  - Access Control card reader sends the request to third-party security system to make a decision to open the door and log access.

# 客户端RPC
# Client-side RPC

- 在底层，平台接收设备发送的一条消息后，交由规则引擎处理。规则引擎可以使用设备属性、遥测或存储在平台中的任何其他数据来进行计算。如果需要，规则引擎也可以调用外部系统。处理消息后，平台将结果发送回设备。如下图所示：

  Under the hood, device sends a message to the platform, which is processed by the Rule Engine. The Rule Engine may apply some calculations using device attributes, telemetry or any other data stored in the platform. Rule Engine may also invoke external system if needed. Once the message is processed, the result is sent back to the device. See the diagram below:



# 客户端RPC
# Client-side RPC

- 客户端RPC请求由两个必选的字段组成：
  - method — 区分RPC调用的方法名称。例如，“getCurrentTime”或“getWeatherForecast”。 该键值对的值是一个字符串。
  - params — 用于处理请求的附加参数。 该值是一个 JSON。如果不需要参数，请保留空 JSON“{}”。
- The client-side RPC request consists of two fields, both of them are mandatory:
  - method - name of the method to distinct the RPC calls. For example, “getCurrentTime” or “getWeatherForecast”. The value of the parameter is a string.
  - params - additional parameters used for processing of the request. The value is a JSON. Leave empty JSON “{}” if no parameters needed.

RPC 请求示例：
Example of the RPC request:

```
{
 "method":"getCurrentTime",
 "params": {}
}
```

RPC 响应可以是任何数字、字符串或 JSON。 例如：
The RPC response may be any number, string or JSON. For example:

```
{
 1631881236974
}
```

# 客户端RPC
# Client-side RPC

- ThingsBoard提供了相应的API来从设备发送RPC命令。API特定于每个支持的网络协议，列举如下。注意LwM2M和SNMP协议还不支持客户端RPC。
  - MQTT客户端RPC API参考
  - CoAP客户端RPC API参考
  - HTTP客户端RPC API参考。

- ThingsBoard provides an API to send RPC commands from the device. The API is specific for each supported network protocol. Note LwM2M and SNMP protocols do not support the client-side RPC yet.
  - MQTT client-side RPC API reference
  - CoAP client-side RPC API reference
  - HTTP client-side RPC API reference

# 客户端RPC
# Client-side RPC

- 平台会将客户端RPC命令被转换为具有“TO_SERVER_RPC_REQUEST”消息类型的规则引擎消息。该消息包含唯一的基于UUID的标识符，该标识符存储在“requestId”元数据字段中。用户可以将规则链设计为使用转换、扩充或任何其他规则节点类型来处理传入消息。一旦传入消息转换为响应消息，就应该使用RPC Call Reply节点向设备发送回复。
- The client-side RPC command is transformed to the Rule Engine message with the “TO_SERVER_RPC_REQUEST” message type. The message contains unique UUID based identifier that is stored in the “requestId” metadata field. You may design your Rule Chain to process the incoming message using transformation, enrichment or any other rule node type. Once the incoming message is transformed to the response message, one should use RPC Call Reply node to send reply to the device.

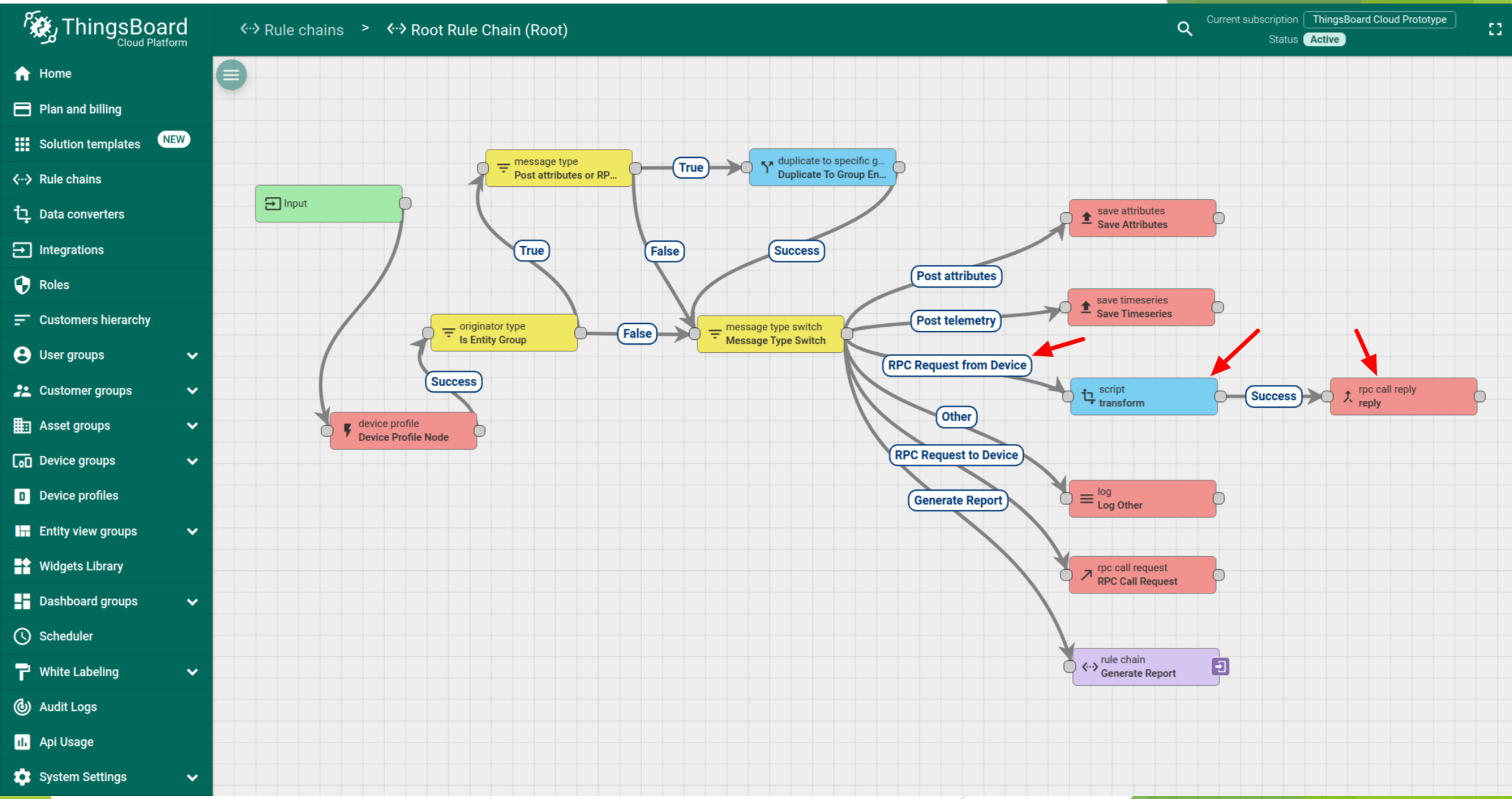
# 客户端RPC
# Client-side RPC

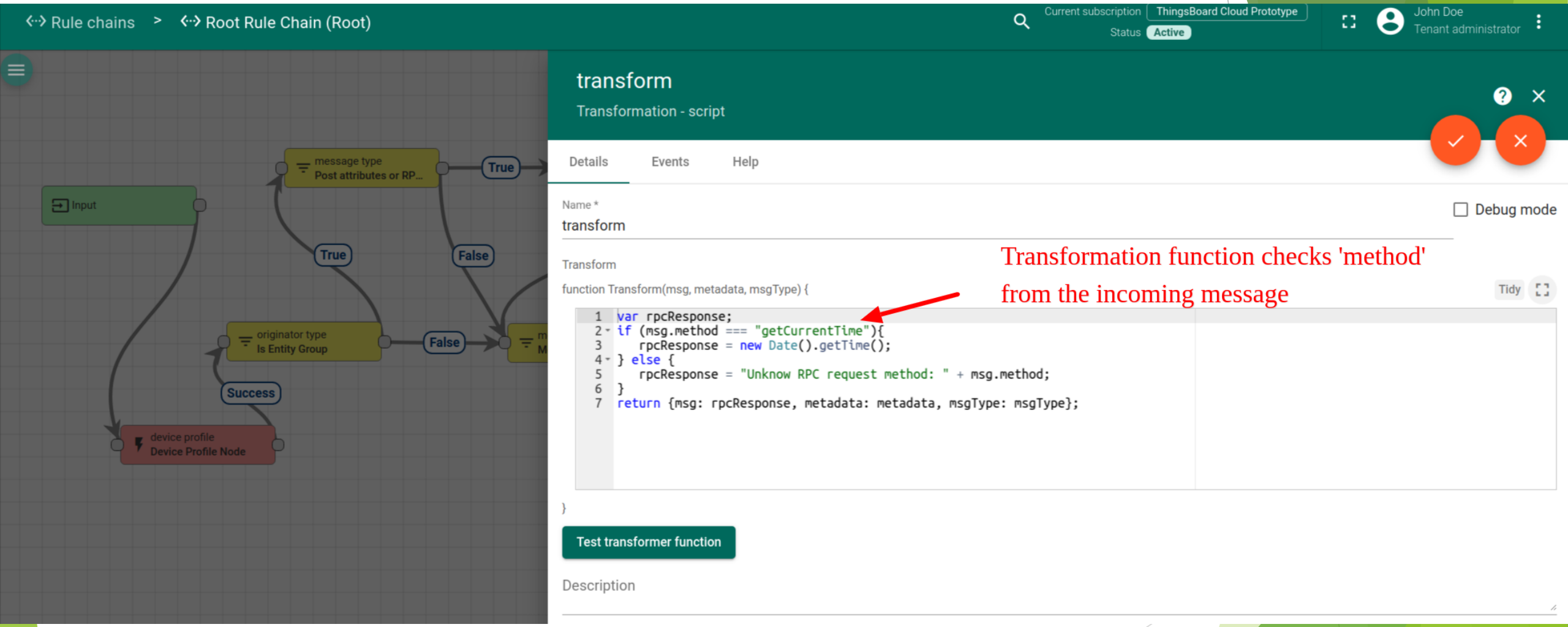
- 下例展示了修改根规则链来处理“getCurrentTime”客户端RPC并以毫秒为单位回复当前时间。下面JS代码的“脚本”转换节点满足此需求：
- The following example demonstrate modifying of root Rule Chain to process “getCurrentTime” client-side RPC and reply with the current time in milliseconds. The “Script” transformation node with the following JS code fulfill this:

```
var rpcResponse;

if (msg.method === "getCurrentTime"){
  rpcResponse = new Data().getTime();
} else {
  rpcResponse = "Unknown RPC request method: "
+ msg.method;
}

return {msg: rpcResponse, metadata: metadata,
msgType: msgType};
```

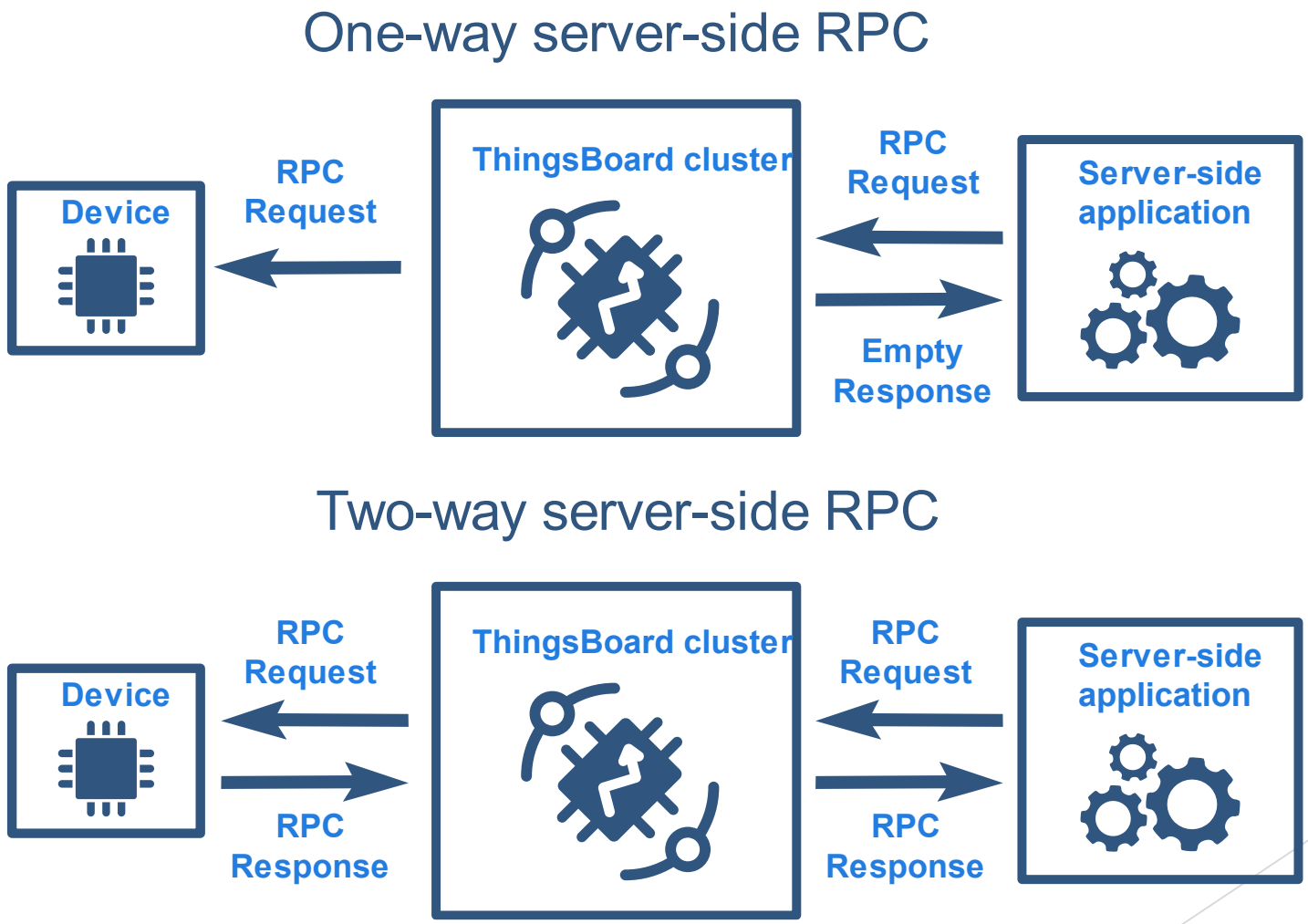

ThingsBoard
Cloud Platform
Rule chains
>
Root Rule Chain (Root)
Current subscription
ThingsBoard Cloud Prototype
Status
Active
Home
Plan and billing
Solution templates
NEW
Rule chains
Data converters
Integrations
Roles
Customers hierarchy
User groups
Customer groups
Asset groups
Device groups
Device profiles
Entity view groups
Widgets Library
Dashboard groups
Scheduler
White Labeling
Audit Logs
Api Usage
System Settings
Input
message type
Post attributes or RP...
True
duplicate to specific g...
Duplicate To Group En...
True
False
Success
originator type
Is Entity Group
False
message type switch
Message Type Switch
Success
device profile
Device Profile Node
Post attributes
Post telemetry
RPC Request from Device
Other
RPC Request to Device
Generate Report
save attributes
Save Attributes
save timeseries
Save Timeseries
script
transform
Success
rpc call reply
reply
log
Log Other
rpc call request
RPC Call Request
rule chain
Generate Report


Rule chains
Root Rule Chain (Root)
Current subscription
ThingsBoard Cloud Prototype
Status
Active
John Doe
Tenant administrator
Input
message type
Post attributes or RP...
True
False
originator type
Is Entity Group
Success
device profile
Device Profile Node
transform
Transformation - script
Details
Events
Help
Name *
transform
Debug mode
Transform
function Transform(msg, metadata, msgType) {
Tidy
1 var rpcResponse;
2 if (msg.method === "getCurrentTime"){
3 rpcResponse = new Date().getTime();
4 } else {
5 rpcResponse = "Unknow RPC request method: " + msg.method;
6 }
7 return {msg: rpcResponse, metadata: metadata, msgType: msgType};
}
Test transformer function
Description
Transformation function checks 'method' from the incoming message

# 服务器端RPC
# Server-side RPC

- 服务器端RPC功能允许用户将请求从平台发送到设备，并可选择将响应返回给平台。服务器端RPC调用的典型用例是各种远程控制：重启、打开/关闭引擎、更改 gpio/执行器的状态、更改配置参数等。服务器端RPC分为单向和双向：
  - 单向 RPC 请求不期望设备提供任何回复。
  - 双向 RPC 请求期望在可配置的超时时间内收到来自设备的响应。。
- Server-side RPC feature allows you to send the request from the platform to the device and optionally get the response back to the platform. The typical use cases of the server-side RPC calls is all sorts of remote control: reboot, turn the engine on/off, change state of the gpio/actuators, change configuration parameters, etc. Server-side RPC is divided into one-way and two-way:
  - One-way RPC request does not expect device to provide any reply.
  - Two-way RPC request expects to receive a response from the device within configurable timeout.

# 服务器端RPC
# Server-side RPC



# 服务器端RPC
# Server-side RPC

- 服务器端RPC请求的主体由多个字段组成：
  - method - 强制性的，用于区分RPC调用的方法名称。例如，“getCurrentTime”或“getWeatherForecast”。该键值对的值是一个字符串。
  - params - 强制性的，用于处理请求的参数。该值是一个JSON。如果不需要参数，请保留空JSON“{}”。
  - timeout - 可选，以毫秒为单位的处理超时值。默认值为10000（10 秒）。最小值为5000（5 秒）。
  - expirationTime - 可选，纪元时间的值（以毫秒为单位，UTC 时区）。如果存在则覆盖超时。
  - persistent - 可选，参见 [persistent]与[lightweight] RPC。默认值为“假”。
  - retries - 可选，定义在网络和/或设备端出现故障时将重新发送持久性RPC的次数。
  - additionalInfo - 可选，定义将添加到[persistent RPC events]的持久RPC的元数据。

# 服务器端RPC
# Server-side RPC

- The body of server-side RPC request consists of multiple fields:
  - method - mandatory, name of the method to distinct the RPC calls. For example, “getCurrentTime” or “getWeatherForecast”. The value of the parameter is a string.
  - params - mandatory, parameters used for processing of the request. The value is a JSON. Leave empty JSON “{}” if no parameters needed.
  - timeout - optional, value of the processing timeout in milliseconds. The default value is 10000 (10 seconds). The minimum value is 5000 (5 seconds).
  - expirationTime - optional, value of the epoch time (in milliseconds, UTC timezone). Overrides timeout if present.
  - persistent - optional, see [persistent] vs [lightweight] RPC. The default value is “false”.
  - retries - optional, defines how many times persistent RPC will be re-sent in case of failures on the network and/or device side.
  - additionalInfo - optional, defines metadata for the persistent RPC that will be added to the [persistent RPC events].

# 服务器端RPC
# Server-side RPC

- 在版本3.3之前，ThingsBoard仅支持轻量级RPC。该类型RPC调用是短暂的，通常在30秒内，这也是平台对任何REST API调用的默认超时。从3.3版本开始，ThingsBoard提供了对持久性RPC调用的支持。持久性RPC具有可配置的生命周期并存储在数据库中。

  Before version 3.3, ThingsBoard supported lightweight RPC only. The lightweight RPC calls are short-lived, typically within 30 seconds which is the default timeout of any REST API call to the platform. Since version 3.3, ThingsBoard provides support of persistent RPC calls. Persistent RPC has a configurable lifetime and is stored in the database.

RPC 请求示例：
Example of the RPC request:

```
{
  "method": "setGPIO",
  "params": {
    "pin": 4,
    "value": 1
  },
  "timeout": 30000
}
```

RPC 响应可以是任何数字、字符串或 JSON。例如：
The RPC response may be any number, string or JSON. For example:

```
{
  "pin": 4,
  "value": 1,
  "changed": true
}
```

# 服务器端RPC
# Server-side RPC

- 服务器端RPC通常使用REST API或仪表板小部件发送。事实上，仪表板小部件使用相同的REST API。一旦平台收到RPC，它就会验证有效负载并运行权限检查。然后，服务器端RPC命令被转换为规则引擎消息。规则引擎可以用额外的参数丰富命令，并最终将命令交付给设备。

  The server-side RPC are typically sent using REST API or dashboard widgets. In fact, dashboard widgets use the same REST API. Once platform received the RPC, it validates the payload and runs permission checks. Then, server-side RPC command is transformed to the Rule Engine message. The Rule Engine may enrich the command with additional parameters and finally issues delivery of the command to the device.

- 用于向设备发送RPC命令的控件小部件有“RPC Button”、“Round Switch”、“Switch Control”和“Knob Control”。 这些小部件的高级设置允许您定义RPC方法名称和参数。用户还可以开发自定义小部件并使用控件api发送RPC命令。

  The Control Widgets used to send RPC commands to the device are “RPC Button”, “Round Switch”, “Switch Control” and “Knob Control”. The advanced settings of those widgets allow you to define RPC method name and params. You may also develop custom widgets and use control api to send RPC commands.

# 服务器端RPC
# Server-side RPC

RPC命令的消息类型为：
Message type of REST API is:
RPC_CALL_FROM_SERVER_TO_DEVICE

- 从小部件或REST API发送的所有服务器端RPC命令最终都会转换为特定消息类型的规则引擎消息。该消息包含基于唯一UUID 的标识符，存储在“requestUUID”元数据字段中。 用户可以将规则链设计为使用转换、扩充或任何其他规则节点类型来处理传入消息，只要保证最后使用RPC Call Request节点将消息发送到设备。

  All server-side RPC commands that are sent from the widgets or REST API are eventually transformed to the rule engine message with specific message type. The message contains unique UUID based identifier that is stored in the “requestUUID” metadata field. The users may design custom Rule Chain to process the incoming message using transformation, enrichment or any other rule node type, as long as using RPC Call Request node to send the message to the device.

- ThingsBoard提供了一个方便的API来接收和处理设备上的服务器端RPC命令。此API特定于受支持的网络协议，目前包括HTTP，CoAP，MQTT。

  ThingsBoard provides a convenient API to receive and process server-side RPC commands on the device. This API is specific for each supported network protocol, currently supporting HTTP, CoAP, MQTT.

# 服务器端RPC
# Server-side RPC

- ThingsBoard 跟踪持久性 RPC 的状态，目前有7种可用状态：
  - QUEUED - RPC已创建并保存到数据库；尚未尝试将RPC发送到设备；当设备联机或已经联机时，ThingsBoard将尝试立即发送RPC；默认情况下，平台将尝试一次发送所有待处理的RPC调用。在极少数情况下，受限设备和队列中有多个消息，这可能会导致网络或设备过载。为避免过载，用户可以使用“ACTORS_RPC_SEQUENTIAL”配置参数启用 RPC 调用的顺序传递。
  - SENT - ThingsBoard尝试将RPC发送到设备。
  - DELIVERED - 设备确认RPC已交付；这是单向RPC处理的最后一步；
  - SUCCESSFUL - ThingsBoard收到双向RPC的回复；
  - TIMEOUT - ThingsBoard传输层（MQTT/CoAP/LwM2M 等）检测到RPC传输超时；使用相应配置参数之一控制超时：MQTT_TIMEOUT（默认 10 秒）、COAP_TIMEOUT（默认 10 秒）、LWM2M_TIMEOUT（默认120秒）。默认情况下，平台不会重试RPC的传递，并且状态将更改为失败。用户可以在RPC主体中配置重试次数。最大重试次数由“ACTORS_RPC_MAX_RETRIES”配置参数控制（默认为 5）。
  - EXPIRED - RPC未交付或平台未在配置的到期时间内收到来自设备的回复；
  - FAILED - 在可配置的重试次数期间未能传递RPC，或者设备固件不支持此类命令。

# 服务器端RPC
# Server-side RPC

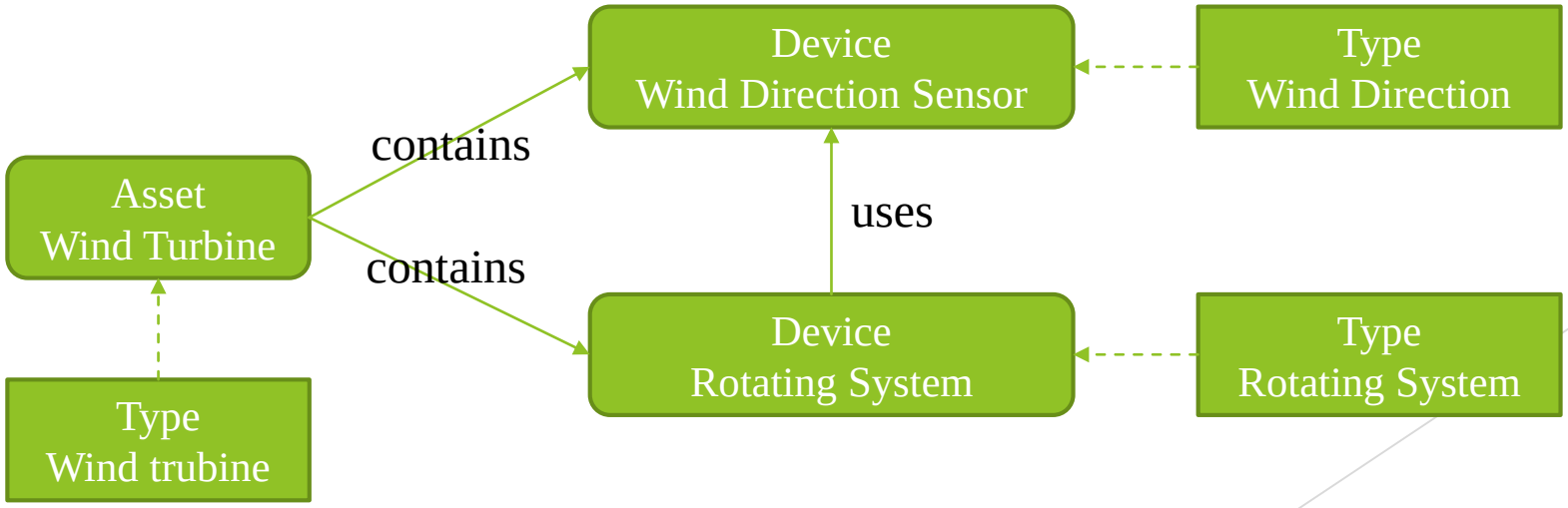
- ThingsBoard tracks state of the persistent RPC. There are 7 available states:
  - QUEUED - RPC was created and saved to the database; No attempt to send the RPC to device yet; ThingsBoard will attempt to send the RPC immediately when device becomes online or if it is already online; The platform will attempt to send all pending RPC calls at once by default. In rare cased of constrained devices and multiple messages in the queue this may lead to overload of the network or device. To avoid the overload, you may enable sequential delivery of RPC calls using “ACTORS_RPC_SEQUENTIAL” configuration parameter.
  - SENT - ThingsBoard performed attempt to send the RPC to device.
  - DELIVERED - device confirmed that the RPC was delivered; This is the last step of processing for one-way RPC;
  - SUCCESSFUL - ThingsBoard received reply for the two-way RPC;
  - TIMEOUT - ThingsBoard transport layer (MQTT/CoAP/LwM2M, etc) detected timeout of the RPC delivery; The timeout is controlled using one of the corresponding configuration parameters: MQTT_TIMEOUT (10 seconds by default), COAP_TIMEOUT (10 seconds by default), LWM2M_TIMEOUT (120 seconds by default) By default, platform will not retry delivery of the RPC, and the state will change to FAILED. You may configure number of retries in the RPC body. The maximum number of retries is controlled by “ACTORS_RPC_MAX_RETRIES” configuration parameter (5 by default).
  - EXPIRED - The RPC was not delivered or platform did not receive the reply from device within configured expiration time;
  - FAILED - failed to deliver the RPC during configurable number of retries or device firmware does not support such a command.

# 用例
# Use Case

- 我们考虑以下用例：假设我们有一台风力涡轮机的资产，上面有风向传感器。我们需要根据风的方向，通过旋转系统改变的风力涡轮机的方向。具体的，我们假设风向传感器与旋转系统都连接到了ThingsBoard，通过向Rotating System发起RPC请求来实现。这里，RPC调用有两个键值对：
  - 方法：左旋或右旋；
  - 参数：值（旋转角度）。
- Let’s assume the following use case: we have the following devices connected to ThingsBoard: wind direction sensor and rotating system., all of which belong to our asset, namely, the wind turbine. You want to initiate an RPC request to the Rotating System and change the direction of the Wind Turbine according to the direction of the wind. Here, the RPC call will have two key-value pairs:
  - method: spinLeft or spinRight;
  - params: value.

# 用例
# Use Case

- 根据以上的描述，我们来进行建模：作为资产的风力涡轮机安装了两个设备：风向传感器和旋转系统。资产风力涡轮机与设备风向传感器、旋转系统的关系为包含关系；同时，存在从旋转系统到风向传感器的使用关系。
- We model the system based on the above description: The Wind Turbine as an asset has two devices installed: Wind Direction Sensor and Rotating System. From the relation perspective, the Wind Turbine contains the Wind Direction Sensor and Rotating System. Besides, there exists a relation of using from Rotating System to Wind Direction Sensor.



# 用例
# Use Case

- 我们根据消息流创建如下节点：
  - 节点A：Message Type Switch节点。根据消息类型路由传入消息。
  - 节点 B：保存时间序列节点。将来自风向传感器和旋转系统的消息遥测存储到数据库中。
  - 节点 C：相关属性。加载相关风向传感器的源遥测windDirection，并将其保存到名为windDirection的消息元数据中。
  - 节点 D：更改发起者节点。将发起者从Devices Wind Direction Sensor and Rotating System更改为相关的Asset Wind Turbine，提交的消息将作为来自 Asset 的消息进行处理。
  - 节点 E：保存时间序列节点。将来自Asset Wind Turbine的消息遥测数据存储到数据库中。
  - 节点 F：转换脚本。将原始消息转换为RPC请求消息。
  - 节点 G：过滤脚本节点。检查传入消息的msgType是否为 RPC 消息。
  - 节点H：RPC调用请求节点。获取消息有效负载并将其作为对旋转系统的响应发送。

# 用例
# Use Case

- We create the following nodes according to the message flow:
  - Node A: Message Type Switch node. Routes incoming messages based on the message type.
  - Node B: Save Timeseries node. Stores messages telemetry from Wind Direction Sensor and Rotating System into the database.
  - Node C: Related attributes. Loads the source telemetry windDirection of the related Wind Direction Sensor and save it into the Message metadata with the name windDirection.
  - Node D: Change originator node. Change the originator from Devices Wind Direction Sensor and Rotating System to the related Asset Wind Turbine and the submitted message will be processed as a message from Asset.
  - Node E: Save Timeseries node. Stores messages telemetry from Asset Wind Turbine into the database.
  - Node F: Transformation Script. Transform an original message into RPC request message.
  - Node G: Filter Script node. Checks if msgType of incoming message is RPC message.
  - Node H: RPC call request node. Takes the message payload and sends it as a response to the Rotating System.

# 用例
# Use Case

- 最后的规则链如下图所示：The figure below demonstrates the final rule chain: