**新大陆物联网云平台**

**TCP透传设备接入协议**

| **版本** | **日期** | **修订内容** | **创建/修订者** |
| --- | --- | --- | --- |
| v1.0 | 2019-05-19 | 初始版本 | cs |

# 说明

便于用户能够快速地接入设备到NLECloud平台，故编写此说明文档，

TCP透传具有较高自由度，它允许用户自由编写设备终端协议脚本解析代码，完成传感数据上报或控制。

# 接入流程

* 1. 访问平台<http://www.nlecloud.com> 注册帐号
  2. 平台以项目为单位，一个项目下可以有多个设备，每个设备可以包含多个传感器，所以依次添加项目、设备、传感器，其中在添加设备时在“通讯协议”一栏**选择“TCP透传”**



添加TCP透传终端设备示意图

* 1. 操作以上功能时顺便记录下：项目ID(projectid)、设备标识(device)、设备传输密钥(key)等信息
  2. 编写针对您的设备终端的Lua协议解析脚本（可以从平台上下载模板后修改），并上传解析脚本



上传TCP透传Lua解析脚本入口示意图



上传Lua解析脚本示意图

* 1. 设备终端发送TCP连接报文到newgateway.nlecloud.com或IP 120.77.58.34，端口5300进行握手，

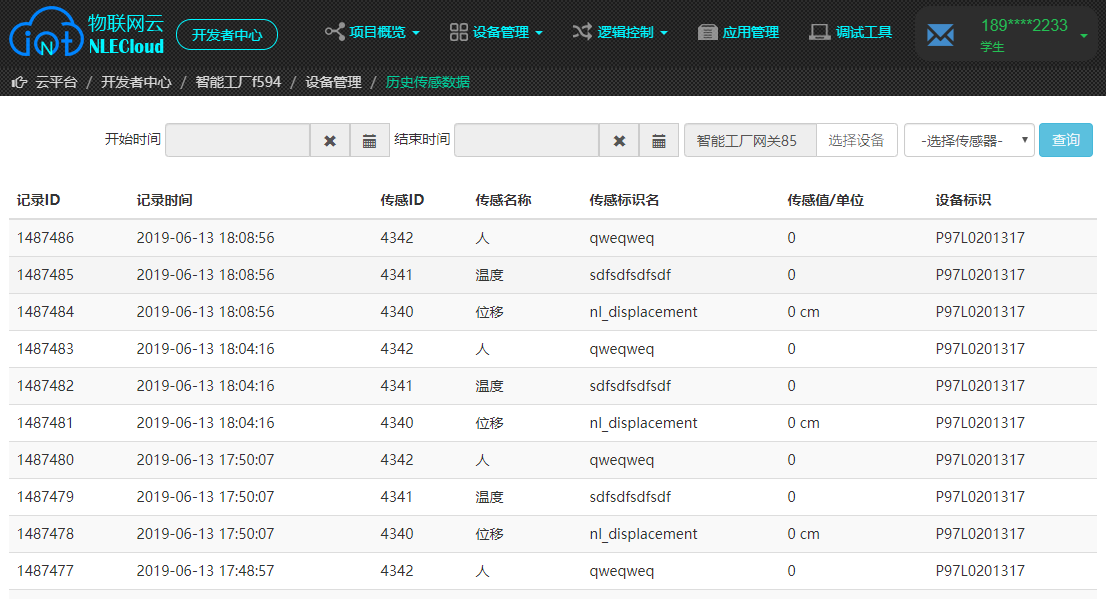
握手报文格式如下：

\*projectId,device,key\*

|  |  |  |  |
| --- | --- | --- | --- |
| **JSON键** | **JSON值** | **说明** | **消息示例** |
| projectid | 项目ID | 创建项目时生成的产品唯一性数字ID | 5685 |
| device | 设备标识 | 在添加设备时，用户创建设备时指定的唯一字符串标识 | PF12345678 |
| Key | 传输密钥 | 在平台上添加设备时自动生成的一串字符串（32位长度的字符串），该值在全局内具备唯一性； | 9861d43a0733415ab5424ee7d0f1c685 |

* 1. 设备握手成功后，NLECloud接入服务会加载先前平台上传的指定脚本，然后根据脚本内的定时下发数据设置，定时下发数据给设备。

进入项目下的“设备管理》历史传感数据”，查看传感数据情况，如下图所示：



# LUA脚本编写说明

用户必须在Lua脚本中实现：

1）定时下发数据任务初始化函数：device\_timer\_init(dev)【可选】

2）设备上传数据解析函数（包括心跳等）：data\_receive\_analyze(dev)

3）平台下发控制命令解析函数：device\_cmd\_analyze(cmd, dev)

# 编写定时下发数据任务初时化函数device\_timer\_init(dev)（可选）

-- 功能：周期性向设备下发数据（可选，此函数为平台函数函数，不可以在该脚本内被调用）

-- 参数：dev devdata 设备对象，dev提供一下几个函数：

-- dev:Task(interval,name,data)添加定时下发数据

-- 参数：interval number 数据下发的时间间隔（秒），最小不能小于3秒

-- name string 名称（须保证唯一性）

-- data string 数据（二进制数据），使用lua转义字符串

-- 返回：成功返回true，否则返回false

-- 示例：local ok = dev:Task(10,"timing","\36\01\15\16")

-- dev:Send(data)下发数据到设备

-- 参数：data string 数据（二进制数据），使用lua转义字符串

-- 返回：无

-- 示例：dev:Send("\26\2\0\150\0\37\206\89")

-- 返回：无

function device\_timer\_init(dev)

-- 添加用户自定义代码, 例如： --

local array = {0x24, 0x01, 0x0F, 0x10} -- 需要下发的数据

local datastr = ary\_to\_bytestr(array) -- 转为lua转义字符串"\36\01\15\16"

dev:Task(10, 'timing', datastr) -- 启动间隔10秒轮询向终端发送数据

end

# 编写设备上传数据解析函数data\_receive\_analyze(dev)

-- 功能：解析设备上传的数据

-- 参数：dev devdata 设备对象，dev提供一下几个函数：

-- dev:Task(interval,name,data)添加定时下发数据

-- 参数：interval number 数据下发的时间间隔（秒），最小不能小于3秒

-- name string 名称（须保证唯一性）

-- data string 数据（二进制数据），使用lua转义字符串

-- 返回：成功返回true，否则返回false

-- 示例：local ok = dev:Task(10,"timing","\36\01\15\16")

-- dev:Send(data)下发数据到设备

-- 参数：data string 数据（二进制数据），使用lua转义字符串

-- 返回：无

-- 示例：dev:Send("\26\2\0\150\0\37\206\89")

-- dev:Size()获取设备上传的数据字节数大小

-- 返回：返回设备数据字节数大小

-- 示例：local sz = dev:Size()

-- dev:Byte(pos)获取pos对应位置的设备数据（字节）

-- 参数：pos number 指定的获取位置，取值范围[1,dev:Size()+1)

-- 返回：成功返回设备数据（int），否则返回nil

-- 示例：local data = dev:Byte(1)

-- dev:Bytes(pos,count)获取从pos开始，count个设备数据

-- 参数：pos number 指定的获取起始位置，取值范围[1,dev:Size()+1)

-- count number 指定的获取数据总数，取值范围[0,dev:Size()+1-pos]

-- 返回：成功返回设备数据（string），否则返回nil

-- 示例：local datas = dev:Bytes(1,dev:Size())

-- 返回：存在以下两种情况

-- 1、传感数据上报解析返回，格式如下：

-- {

-- "t" : 3, -- 固定值3

-- "datatype" : 1,

-- "msgid" : 123,

-- "datas" :

-- {

-- "temperature" : 27.3 | "27.3" | {…},-- 传感标识名 : 布尔值、数值、字符串、json字符串

-- "light" : 4256 | "4256" | {…} , -- 传感标识名 : 布尔值、数值、字符串、json字符串

-- }

-- }

-- 2、命令响应解析返回，格式如下:

-- {

-- "t": 6, -- 固定值6

-- "cmdid": 123, -- 命令下发时携带的命令编号，原样还回

-- "status": 0, -- 0：处理成功；1：处理失败；

-- "data": 1 -- 数字、浮点、字符串、json字符串

-- }

function data\_receive\_analyze(dev)

local s = dev:Size()

local datastr = dev:Bytes(1, s) -- 通过dev:Bytes(1, s)获取原始以“\”转义的的byte string

-- 示例1：传感数据上报解析 ======================

local result = {} -- 定义一个table，用于存放返回数据

local t = {datastr:byte(1, datastr:len())} -- 将上述的datastr逐个插入临时table中

local val = (t[2] + t[3]) / 2 + 4 -- 示范取数据流的“第2位+第3位然后除2加4”这么个计算公式

add\_val(result, 'temperature', val) -- 插入到result中

-- ...

dev:Send("\34\23\24\56") -- 向设备发送响应

return to\_str(result) -- 转换为平台可识别的字符串格式

-- 示例2：命令响应解析 ======================

local cmdresp = {datastr:byte(1, datastr:len())}

local result = {}

result.t = 6

result.cmdid = math.floor(math.random() \* 10000) -- 随机给cmdid附个值

if(cmdresp[1] == 0x00) -- 取设备响应的第一位，等于0x00时代表成功

then

result.status = 0

result.data = cmdresp[2] -- 成功将设备响应的第二个字符附给

else

result.status = 1

result.data = 0

end

-- ...

return to\_str(result) -- 转换为平台可识别的字符串格式

end

# 编写平台下发控制命令解析函数device\_cmd\_analyze(cmd, dev)

-- 功能：解析平台下发的命令

-- 参数：cmd

-- cmdid ：命令编号

-- apitag：执行器标识

-- data： 命令内容

-- dev devdata 设备对象，同上

-- 返回：终端设备所能处理的自定义命令格式。

function cmd\_send\_analyze(cmd , dev)

-- 示例：比如要向设备发送[0x01, 0x24, 0x0F, 0x10] 这串命令，0x01代表地址，对应平台的传感标识是风扇fan======================

local array = {}

if(cmd.apitag == "fan")

then

array[1] = 0x01

array[2] = cmd.data

array[3] = 0x0F

array[4] = 0x10

end

local str = ary\_to\_bytestr(array) -- 转为lua转义字符串"\36\01\15\16"

dev:Send(str)

end

# 常用函数定义

-- 功能：将二进制数组转为带“\”的字符串

-- 参数：o table数组

-- 返回：返回序列化string

-- 示例：local str = ary\_to\_bytestr({0x24, 0x01, 0x0F, 0x10})

-- 结果："\36\1\15\16"

function ary\_to\_bytestr(ary)

{

local result = ""

for i = 1, #ary do

result = result .. "\\" .. string.format(ary[i])

end

return result

}

-- 功能：将object序列化成字符串

-- 参数：o boolean|number|string|table

-- 返回：返回序列化string

-- 示例：local str = to\_str({x=100})

-- 结果：{"x":100}

function to\_str(o)

local i=1

local t={}

local f

f=function(x)

local y=type(x)

if y=="number" then

t[i]=x

i=i+1

elseif y=="boolean" then

t[i]=tostring(x)

i=i+1

elseif y=="string" then

t[i]="\""

t[i+1]=x

t[i+2]="\""

i=i+3

elseif y=="table" then

t[i]="{"

i=i+1

local z=true

for k,v in pairs(x) do

if z then

z=false

t[i]="\""

t[i+1]=k

t[i+2]="\""

t[i+3]=":"

i=i+4

f(v)

else

t[i]=","

t[i+1]="\""

t[i+2]=k

t[i+3]="\""

t[i+4]=":"

i=i+5

f(v)

end

end

t[i]="}"

i=i+1

else

t[i]="nil"

i=i+1

end

end

f(o)

return table.concat(t)

end

-- 功能：添加传感值到table中

-- 参数： t table

-- apitag string 传感标识

-- value boolean|number|string|table 布尔值、数值、字符串、json

-- 返回：成功返回true，否则返回false

-- 示例：local ok = add\_val(table,"temperature",34)

function add\_val(t, apitag, value)

if type(t) ~= 'table' then

return false

elseif type(apitag) ~= 'string' then

return false

else

local type = type(value)

if type ~= 'boolean' and type ~= 'number' and type ~= 'string' and type ~= 'table' then

return false

end

end

if t.datas == nil then

t.datas = {}

end

t.datas[apitag] = value

if t.t == nil then

t.t = 3

end

if t.datatype == nil then

t.datatype = 1

end

if t.msgid == nil then

t.msgid = math.floor(math.random() \* 10000)

end

return true

end