

# TCP 透传-快速接入文档

修订记录

日期	修订版本	修改描述	作者
2017-02-09	V1.0	添加	XiaoBo
2017-02-14	V1.1	添加部分 Lua 函数说明	XiaoBo
2017-03-29	V1.2	支持同一连接传输多个设备的数据 支持 API 下发命令	XiaoBo
2017-04-01	V1.3	Lua 支持获取时间戳、年月日、时分秒	XiaoBo

缩略语清单

缩略语	英文全名	中文解释

## 1 说明

便于用户能够快速地接入设备到 ONENET 平台，故编写此说明文档。

## 2 名词解析

PID：产品 ID，创建产品时 OneNET 生成的产品唯一性数字标识

AuthCode：设备鉴权码，在设备注册时，用户创建设备时指定的唯一字符串标识

ParserName：用户自定义解析脚本的名称，用户上传脚本时指定的唯一字符串标识

## 3 接入流程

### 第 1 步创建产品

登录 ONENET 平台进入开发者中心，选择公共协议产品，根据实际情况创建自己的产品（此文档以 **dtu-test** 为例），如下图所示。



1 个公开协议产品

0 个私有协议产品

[创建产品](#)

dtu\_test

其他

设备接入方式：TCP透传

创建时间：2017-04-01 13:09



0 台

接入设备



0 个

生成应用



1 个

API Key



0 个


触发器数

## 第 2 步创建设备

进入 dtu-test 产品的“[设备管理](#)”，进行添加设备，（此文档以“test”举例说明），如下图所示：








OneNET

开发者中心



产品概况

设备管理

数据流模板

APIKey管理

触发器管理

设备管理

编辑设备信息

设备接入协议: TCP透传协议

设备名称: test

鉴权信息: test

数据保密性: ☒ 私有 ☐ 公开

同时，也可以进入 dtu-test 产品的“[在线调试》API 调试工具](#)”，参照文档中心下的“[应用开发创建设备](#)”说明，创建设备，（此文档以 test\_ds 举例说明）

请求内容如下：

```
{
  "title": "test",
  "auth_info": "test"
}
```

返回结果：

```
{
  "ermo": 0,
  "data": {
    "device_id": "4658765"
  },
  "error": "succ"
}
```

OneNET

产品概况

设备管理

数据流模板

APIKey管理

触发器管理

应用管理

第三方开发平台

在线调试 1

### 在线调试

数据模拟器
地图数据模拟器
API调试工具 2

• ApiURL地址

http://api.heclouds.com/devices

请求方法：

POST

请求参数：

+ 添加

请求内容

```
{
  "title": "test",
  "auth_info": "test"
}
```

设置设备名称和鉴权码

• APIKey

V=IQW-R7k=PBSSr-whcPM=

执行请求 3

#### 返回结果

```
{
  "errno": 0,
  "data": {
    "device_id": "4658765"
  },
  "error": "succ"
}
```

最后一次请求时间：2017-2-9 15:46:35

### 第 3 步上传解析脚本

进入 dtu-test 产品的“脚本管理”页面，上传 LUA 解析脚本文件，此文档以“modbus.lua”举例说明，如下图所示：

OneNET

产品概况

设备管理

数据流模板

APIKey管理

触发器管理

应用管理

[开发者中心](#)

### 脚本管理

关于脚本

[脚本语法说明](#)

脚本是用于解析从设备上传的数据，从平台给设备下发命令，通过数据脚本解析后，通过设备上传的数据能够在平台展示；并可以被统计、分析。

最多可以上传不超过10个数据脚本 [上传脚本](#)

为了保证脚本运行的稳定和正确性，我们也提供本地脚本验证工具，通过脚本验证工具，你可以直观的查看脚本解析后的脚本数据。点击这里可以下载[本地脚本调试工具](#)。

还没有脚本！

立即上传脚本>



OneNET

[开发者中心](#)


[产品概况](#)
[设备管理](#)
[数据流模板](#)
[APIKey管理](#)

### 上传脚本

脚本的单个执行时间限制为2ms，下发数据量为 1KB，所有定时器的频率之和不大于1Hz，运行内存小于100KB。

脚本名称:  请输入设备版本，只支持数字、英文

脚本文件:  [+重新添加脚本文件](#)

点击保存后，数据将通过该脚本进行解析



OneNET

[开发者中心](#)


[产品概况](#)
[设备管理](#)
[数据流模板](#)
[APIKey管理](#)
[触发器管理](#)
[应用管理](#)

### 脚本管理

关于脚本 [脚本语法说明](#)

脚本是用于解析从设备上传的数据，从平台给设备下发命令，通过数据脚本解析后，通过设备上传的数据能够在平台展示；并可以被统计、分析。

最多可以上传不超过10个数据脚本 [上传脚本](#)

为了保证脚本运行的稳定和正确性，我们也提供本地脚本验证工具，通过脚本验证工具，你可以直观的查看脚本解析后的脚本数据。点击[这里](#)可以下载[本地脚本调试工具](#)。

脚本数量: 1个

输入脚本名称搜索，不支持模糊查询，需输入全名

脚本名称	脚本文件名	操作
modbus 上传时间: 2017-04-01	modbus.lua	<input type="button" value="编辑"/> <input type="button" value="详情"/> <input type="button" value="删除"/>

## 第 4 步

设备上电，发送登录报文到 OneNET 接入服务器（[见文档中心 FAQ](#)）进行鉴权。

登录报文格式如下：

**\*PID#AuthCode#ParserName\***

## 第 5 步查看设备数据点

设备登录成功后，OneNET 接入服务会加载登录报文中指定的脚本，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 dtu-test 产品下的“[设备管理](#)》[设备数据](#)”，查看数据点情况，如下图所示：



产品概况

设备管理

数据流模板

APIKey管理

触发器管理

应用管理

设备管理

设备数据

设备信息

数据展示

5条设备数据总数

0条昨日新增

0条最近7日新增

数据流展示

ds4	更新时间:2017-02-09 16:24:23	
ds3	更新时间:2017-02-09 16:24:17	
ds2	更新时间:2017-02-09 16:24:11	
ds1	更新时间:2017-02-09 16:24:16	

其中，数据流名称是在解析脚本中指定的。

## 4 LUA 脚本编写说明

### 4.1 设置定时下发设备的数据

用户需实现 Lua 函数 `device_timer_init(dev)`，以完成数据定时下发设备的设置，`device_timer_init()` 无返回值。

`dev` 为一个 `user_data` 类型的值，提供了以下几个函数：

- 1) `dev:add(interval, name, data)`

添加定时下发数据。

```

@param    interval number    数据下发的时间间隔（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

@return   成功返回 true，否则返回 false

@notice   定时数据下发的平均频率不超过 1，及 1/interval_1+...+1/interval_n<=1

@example  dev:add(10,"test","\1\0\150\0\37\253\29")
  
```

- 2) `dev:timeout(sec)`

设置下发数据的设备响应超时时间（秒）。

```

@param    sec      number    响应超时时间（秒）
                                如果值为 0，表示不检测设备响应超时

@return   无

@example  dev:timeout(3)
  
```

- 3) `dev:response()`

设备响应成功。



```
@param 无
@return 无
@example dev:response()
```

#### 4) dev:send(data)

下发数据到设备。

```
@param data string 数据（二进制数据），使用 lua 转义字符串
@return 无
@example dev:send("\2\2\0\150\0\37\206\89")
```

## 4.2 解析设备上传数据

用户需实现 Lua 函数 device\_data\_analyze(dev)，以完成对设备上传数据的解析，device\_data\_analyze() 有 2 个返回值 size,json。

其中，size 表示已解析设备上传数据的字节数，json 表示解析后的数据点集合，格式如下：

```
[
  {
    "i": "dsname1",          // 数据流或数据流模板名称 1
    "a": 1234567890,         // 毫秒级时间戳，距离（00:00:00 UTC, January 1, 1970）的毫秒
                              // 如果值为 0，表示使用当前时间
    "v": 123 | "123" | {...} // 布尔值、数值、字符串、json
    "b": "0A0B0C0D..."    // 二进制数据（16 进制字符串），与 v 互斥，不同时存在
    "d": xxx | "xxx" | {...} // 用于描述 b（可选）；布尔值、数值、字符串、json
    "c": "authcode1"        // 用于标识数据点归属（设备 AuthCode）
                              // 如果值为 "" 或不存在，表示数据点归属建立 TCP 连接的设备
  }
  ...
  {
    "i": "dsnamen",         // 数据流或数据流模板名称 n
    "a": 1234567890,         // 毫秒级时间戳，距离（00:00:00 UTC, January 1, 1970）的毫秒
                              // 如果值为 0，表示使用当前时间
    "v": 123 | "123" | {...} // 布尔值、数值、字符串、json
    "b": "0A0B0C0D..."    // 二进制数据（16 进制字符串），与 v 互斥，不同时存在
    "d": xxx | "xxx" | {...} // 用于描述 b（可选）；布尔值、数值、字符串、json
    "c": "authcode1"        // 用于标识数据点归属（设备 AuthCode，可选）
                              // 如果值为 "" 或不存在，表示数据点归属建立 TCP 连接的设备
  }
]
```

dev 为一个 user\_data 类型的值，提供了以下几个函数：

#### 1) dev:add(interval, name, data)

添加定时下发数据。

```
@param interval number 数据下发的时间间隔（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
```

@return 成功返回 true，否则返回 false

@notice 定时数据下发的平均频率不超过 1，及  $1/\text{interval\_1} + \dots + 1/\text{interval\_n} \leq 1$

@example `local ok = dev:dd(10,"test","\1\1\0\150\0\37\253\29")`

2) `dev:timeout(sec)`

设置下发数据的设备响应超时时间（秒）。

@param sec number 响应超时时间（秒）

如果值为 0，表示不检测设备响应超时

@return 无

@example `dev:timeout(3)`

3) `dev:response()`

设备响应成功。

@param 无

@return 无

@example `dev:response()`

4) `dev:send(data)`

下发数据到设备。

@param data string 数据（二进制数据），使用 lua 转义字符串

@return 无

@example `dev:send("\2\2\0\150\0\37\206\89")`

5) `dev:size()`

获取设备数据大小（字节数）。

@param 无

@return 返回设备数据大小（字节数）

@example `local sz = dev:size()`

6) `dev:byte(pos)`

获取 pos 对应位置的设备数据（字节）。

@param pos number 指定的获取位置，取值范围  $[1, \text{dev:size}()+1)$

@return 成功返回设备数据（int），否则返回 nil

@example `local data = dev:byte(1)`

7) `dev:bytes(pos, count)`

获取从 pos 开始，count 个设备数据。

@param pos number 指定的获取起始位置，取值范围  $[1, \text{dev:size}()+1)$

count number 指定的获取数据总数，取值范围  $[0, \text{dev:size}()+1-\text{pos}]$

@return 成功返回设备数据（string），否则返回 nil

@example `local datas = dev:bytes(1,devsize())`

## 4.3 Lua 工具函数

1) `u2f(u)`

将 32 为整数内存转换为浮点数内存，并返回浮点数值（不同于值转换）；其类似于 C/C++ 的强制内存转换，例如：`*(float*)&u`。

@param u number

@return 成功返回浮点数值，否则返回 nil

@example `local f = u2f(u)`

2) `time()`

获取时间戳，距离（00:00:00 UTC, January 1, 1970）的毫秒数

@return 返回时间戳

@example local t = time()

3) year(t)

获取年，距离 1900 的年数

@param t number 时间戳，距离（00:00:00 UTC, January 1, 1970）的秒数

@return 返回年

@example local y = year(t)+1900

4) month(t)

获取月（0-11）

@param t number 时间戳，距离（00:00:00 UTC, January 1, 1970）的秒数

@return 返回月

@example local m = month(t)+1

5) day(t)

获取日（1-31）

@param t number 时间戳，距离（00:00:00 UTC, January 1, 1970）的秒数

@return 返回日

@example local d = day(t)

6) hour(t)

获取时（0-23）

@param t number 时间戳，距离（00:00:00 UTC, January 1, 1970）的秒数

@return 返回时

@example local h = hour(t)

7) minute(t)

获取分（0-59）

@param t number 时间戳，距离（00:00:00 UTC, January 1, 1970）的秒数

@return 返回分

@example local m = minute(t)

8) second(t)

获取秒（0-59）

@param t number 时间戳，距离（00:00:00 UTC, January 1, 1970）的秒数

@return 返回秒

@example local s = second(t)

9) to\_hex(s)

将 bytes string 转换为 hex string。

@param s string bytes string

@return 返回 hex string，类似"0A0B0COD..."

@example local hex = to\_hex(s)

10) to\_str(o)

将 lua 对象序列化成字符串。

@param o boolean|number|string|table

@return 返回序列化 string

@example local str = to\_str(o)

11) add\_val(t, i, a, v)

添加值数据点到 table 中。

```
@param  t    table
        i    string          数据流或数据流模板名称
        a    number          毫秒级时间戳, 距离(00:00:00 UTC, January 1, 1970)的毫秒;
                                如果值为 0, 表示使用当前时间
        v    boolean|number|string|table 布尔值、数值、字符串、json
        c    string          用于标识数据点归属(设备 AuthCode, 可选)
                                如果值为 “” 或 nil, 表示数据点归属建立 TCP 连接的设备
```

@return 成功返回 true, 否则返回 false

@example `local ok = add_val(t,"dsname",0,100,"dev")`

## 12) add\_bin(t, i, a, b, d)

添加二进制数据点到 table 中。

```
@param  t    table
        i    string          数据流或数据流模板名称
        a    number          毫秒级时间戳, 距离(00:00:00 UTC, January 1, 1970)的毫秒;
                                如果值为 0, 表示使用当前时间
        b    string          二进制数据(hex string), 类似"0A0B0C0D..."
        d    boolean|number|string|table 用于描述 b(可选), 布尔值、数值、字符串、json
        c    string          用于标识数据点归属(设备 AuthCode, 可选)
                                如果值为 “” 或 nil, 表示数据点归属建立 TCP 连接的设备
```

@return 成功返回 true, 否则返回 false

@example `local ok = add_bin(t,"dsname",0,"0A0B0C0D...",{...},"dev")`

## 13) to\_json(t)

将 table 序列化成 json 字符串。

@param t table 通过 add\_val、add\_bin 构建起来的 table

@return 返回序列化 json 字符串

@example `local json = to_json(t)`