RT-THREAD ONENET 用户手册

RT-THREAD 文档中心

WWW.RT-THREAD.ORG

Friday 28th September, 2018

版本和修订

Date	Version	Author	Note
2018-07-17	v0.1.0	zylx	初始版本
2018-07-27	v0.1.1	zylx	添加 mqtt 上传功能

目录

第 1 章

OneNET 软件包介绍

OneNET 平台是中国移动基于物联网产业打造的生态平台，具有高并发可用、多协议接入、丰富 API 支持、数据安全存储、快速应用孵化等特点，同时，OneNET 平台还提供全方位支撑，加速用户产品的开发速度。

OneNET 平台是一个基于物联网产业特点打造的生态环境，可以适配各种网络环境和协议类型，现在支持的协议有 LWM2M（NB-IOT）、EDP、MQTT、HTTP、MODBUS、JT\T808、TCP 透传、RGMP 等。用户可以根据不同的应用场景选择不同的接入协议。

该组件包是 RT-Thread 系统针对 OneNET 平台连接的适配，通过这个组件包可以让设备在 RT-Thread 上非常方便的连接 OneNet 平台，完成数据的发送、接收、设备的注册和控制等功能。

1.1 文件目录结构

```
OneNET
|   README.md                       // 软件包使用说明
|   SConscript                      // RT-Thread 默认的构建脚本
├---docs
|   └---figures                     // 文档使用图片
|   |   api.md                      // API 使用说明
|   |   introduction.md             // 软件包详细介绍
|   |   principle.md                // 实现原理
|   |   README.md                   // 文档结构说明
|   |   samples.md                  // 软件包示例
|   |   user-guide.md               // 使用说明
|   |   port.md                     // 移植说明文档
|   └---version.md                  // 版本
├---ports                           // 移植文件
|   └---rt_ota_key_port.c           // 移植文件模板
```

```
├───samples                    // 示例代码
│   └───onenet_sample.c        // 软件包应用示例代码
├───inc                        // 头文件
└───src                        // 源文件
```

1.2 OneNET 软件包功能特点

RT-Thread OneNET 软件包功能特点如下：

断线重连

RT-Thread OneNET 软件包实现了断线重连机制，在断网或网络不稳定导致连接断开时，会维护登陆状态，重新连接，并自动登陆 OneNET 平台。提高连接的可靠性，增加了软件包的易用性。

自动注册

RT-Thread OneNET 软件包实现了设备自动注册功能。不需要在 web 页面上手动的一个一个创建设备，输入设备名字和鉴权信息。当开启设备注册功能后，设备第一次登陆 OneNET 平台时，会自动调用注册函数向 OneNET 平台注册设备，并将返回的设备信息保存下来，用于下次登陆。

自定义响应函数

RT-Thread OneNET 软件包提供了一个命令响应回调函数，当 OneNET 平台下发命令后，RT-Thread 会自动调用命令响应回调函数，用户处理完命令后，返回要发送的响应内容，RT-Thread 会自动将响应发回 OneNET 平台。

自定义 topic 和回调函数

RT-Thread OneNET 软件包除了可以响应 OneNET 官方 topic 下发的命令外，还可以订阅用户自定义的 topic，并为每个 topic 单独设置一个命令处理回调函数。方便用户开发自定义功能。

上传二进制数据

RT-Thread OneNET 软件包除了上传数字和字符串外，还支持二进制文件上传。当启用了 RT-Thread 的文件系统后，可以直接将文件系统内的文件以二进制的方式上传至云端。

第 2 章

OneNET 示例应用程序

2.1 准备工作

2.1.1 在 OneNET 云上注册账号

设备接入 OneNET 云之前，需要在平台注册用户账号，OneNET 云平台地址：`https://open.iot.10086.cn`

创建产品

账号注册登录成功后，点击**开发者中心**进入开发者中心界面；

点击**创建产品**，输入产品基本参数，页面最下方设备接入协议选择 `MQTT` 协议，如下图所示：

图 **2.1:** *onenet*

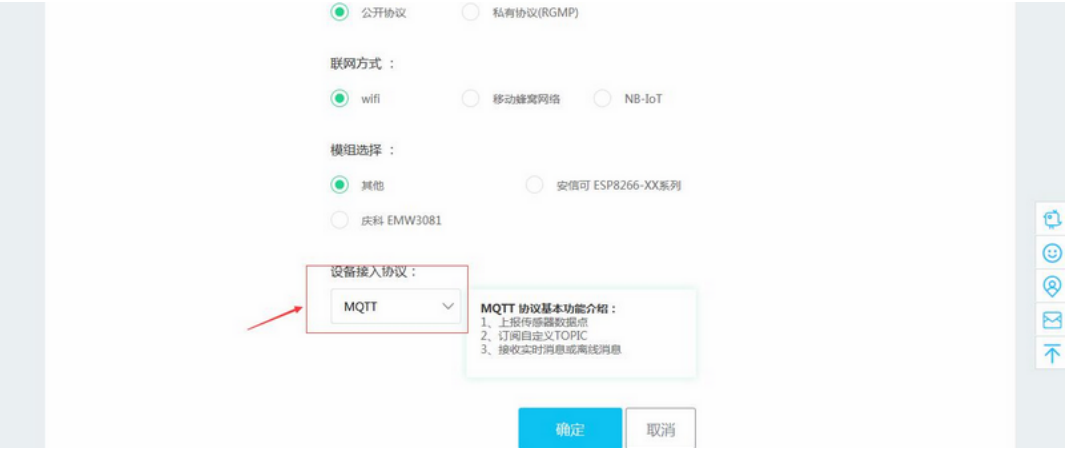

图 2.2: *onenet_create_product*

产品创建成功之后，可以在开发者中心左侧**产品概况**中查看产品基础信息（如产品 ID，接入协议，创建时间，产品 APIkey 等，后面有用）。

接入设备

在开发者中心左侧**设备管理**中点击添加设备按钮添加设备，设备名称我们填入`test1`。鉴权信息是为了区分每一个不同的设备，如果创建了多个设备，要确保每个设备的鉴权信息都不一样，我们这里填入`201807171718`, 填完之后点击接入设备

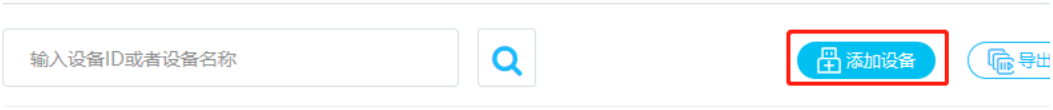

图 2.3: *onenet_add_device*

图 2.4: *onenet_create_device*

添加 APIkey

接入设备之后，可以看到设备管理的界面多了一个设备，设备的右边有一些操作设备的按钮，点击查看详情按钮

图 2.5: *onenet_info*

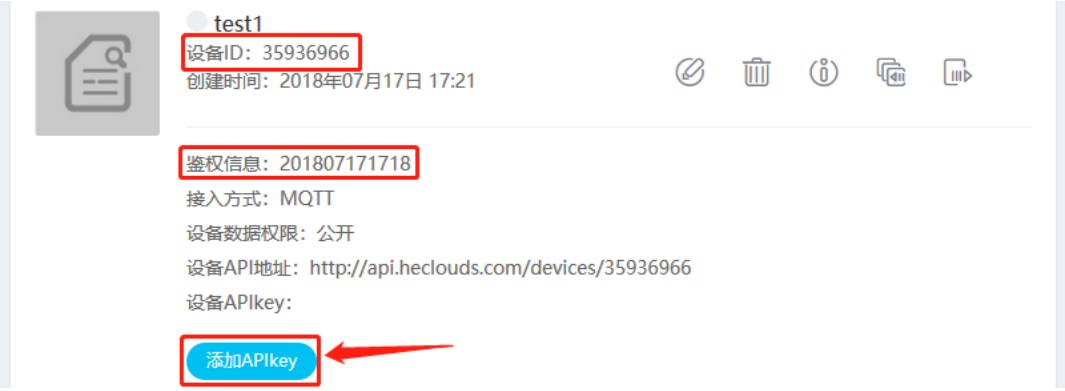

图 2.6: *onenet_add_apikey*

此设备的相关信息就都显示出来了，比如：设备 ID、鉴权信息、设备 APIkey，这些信息需要记下，在 ENV 配置时会用到。

点击按钮添加 APIkey，APIKey 的名称一般和设备相关联，我们这里填入`test_APIKey`，关联设备写入我们刚刚创建的设备`test1`。

图 2.7: *onenet7*

开启 onenet 软件包

打开 env 工具输入 menuconfig 按照下面的路径开启 onenet 软件包

```
RT-Thread online packages
    IoT - internet of things  --->
        IoT Cloud  --->
            [*] OneNET: China Mobile OneNet cloud SDK for RT-Thread
```

进入 onenet 软件包的配置菜单按下图所示配置，里面的信息依据自己的产品和设备的实际情况填写

```
--- OneNET: China Mobile OneNet cloud SDK for RT-Thread
    [ ]   Enable OneNET sample
    [*]   Enable support MQTT protocol
    [ ]   Enable OneNET automatic register device (NEW)
    (35936966) device id
    (201807171718) auth info
    (H3ak5Bbl0NxpW3QVVe33InnPxOg=) api key
    (156418) product id
    (dVZ=ZjVJvGjXIUDsbropzg1a8Dw=) master/product apikey (NEW)
        version (latest)  --->
```

Enable OneNET sample：开启 OneNET 示例代码

Enable support MQTT protocol：开启 MQTT 协议连接 OneNET 支持

Enable OneNET automatic register device：开启 OneNET 自动注册设备功能

device id：配置云端创建设备时获取的 设备ID

auth info：配置云端创建产品时 用户自定义的鉴权信息 (每个产品的每个设备唯一)

api key：配置云端创建设备时获取的 APIkey

product id：配置云端创建产品时获取的 产品ID

master/product apikey：配置云端创建产品时获取的 产品APIKey

2.2 示例文件介绍

利用 ENV 生成工程后，我们可以在工程的 onenet 目录下看到onenet_sample.c文件，该文件是 **OneNET** 软件包的示例展示，主要是展示用户如何使用 **OneNET** 软件包上传数据和接收命令。

2.3 运行示例

在使用 OneNET 软件包之前必须要先调用onenet_mqtt_init这个命令进行初始化，初始化完成后设备会自动连接 OneNET 平台。

```
msh />onenet_mqtt_init
[D/ONENET] (mqtt_connect_callback:85) Enter mqtt_connect_callback!
[D/[MQTT] ] ipv4 address port: 6002
[D/[MQTT] ] HOST = '183.230.40.39'
[I/ONENET] RT-Thread OneNET package(V0.2.0) initialize success.
msh />[I/[MQTT] ] MQTT server connect success
[D/ONENET] (mqtt_online_callback:90) Enter mqtt_online_callback!
```

2.3.1 上传数据

初始化完成后，用户可以调用onenet_upload_cycle这个命令周期性的往云平台上传数据。输入这个命令后，设备会每隔 5s 向数据流 temperature 上传一个随机值。并将上传的数据打印到 shell 窗口。

```
msh />onenet_upload_cycle
msh />[D/ONENET] (onenet_upload_data:106) buffer : {"temperature":32}
[D/ONENET] (onenet_upload_data:106) buffer : {"temperature":51}
```

我们打开 OneNET 平台，在设备管理的界面点击数据流管理按钮进入数据流界面。

图 2.8: *onenet_datastream*

点击temperature数据流右边的小箭头显示数据流信息，我们就可以看到刚刚上传的数据了。

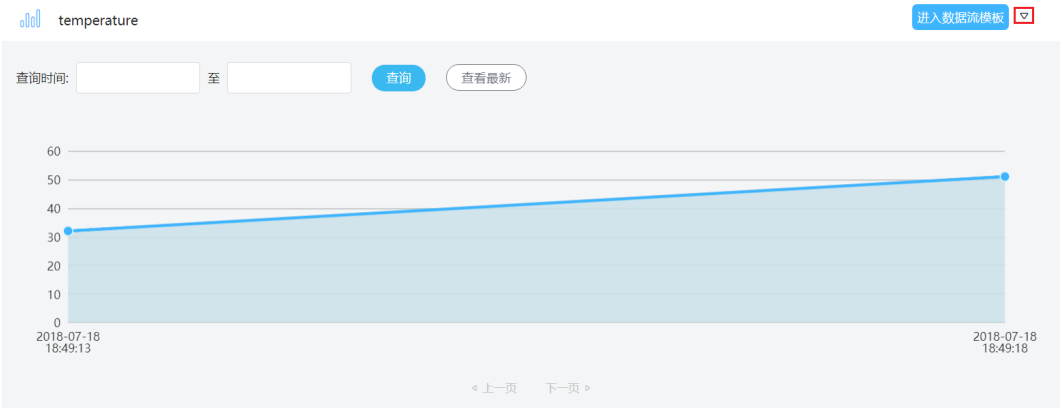

图 2.9: *onenet_datapoints*

如果用户想往别的数据流发送信息，可以使用以下 API 往云平台上传数据。

onenet_mqtt_publish_digit onenet_mqtt_publish_string

命令格式如下所示

```
onenet_mqtt_publish_digit 数据流名称 要上传的数据

onenet_mqtt_publish_string 数据流名称 要上传的字符串
```

输入命令后没有返回错误信息就表示上传成功。

示例如下

```
msh />onenet_mqtt_publish_digit test 1
msh />onenet_mqtt_publish_string test 1
msh />onenet_mqtt_publish_digit test 2
msh />onenet_mqtt_publish_string test 1
```

在数据流管理页面，我们可以看到多出来个 test 数据流，里面的数据就是我们刚刚上传的数据。

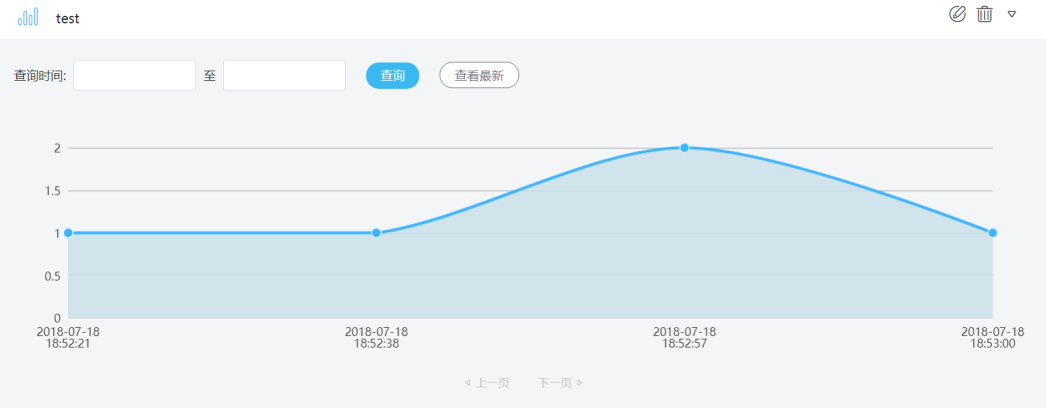

图 2.10: *onenet_upload_dp*

2.3.2 接收命令

在初始化时，命令响应回调函数默认指向了空，想要接收命令，必须设置命令响应回调函数，在 shell 中输入命令`onenet_set_cmd_rsp`, 就把示例文件里的命令响应回调函数挂载上了，这个响应函数在接收到命令后会把命令打印出来。

```
msh />onenet_set_cmd_rsp
```

我们点击设备管理界面的发送命令按钮。

图 2.11: *onenet_cmd*

在弹出来的窗口里输出 hello rt-thread!，然后点击发送命令。

图 2.12: *onenet_hello_rtthread*

就可以在 shell 中看到云平台下发的命令了。

```
msh />onenet_set_cmd_rsp
msh />[D/ONENET] (mqtt_callback:60) topic $creq/6db0c1b2-9a7e-5e4a-8897-
    bf62d4a3461f
receive a message
[D/ONENET] (mqtt_callback:62) message length is 18
[D/ONENET] (onenet_cmd_rsp_cb:107) recv data is hello rt-thread!
```

第 3 章

OneNET 工作原理

OneNET 软件包数据的上传和命令的接收是基于 MQTT 实现的，OneNET 的初始化其实就是 MQTT 客户端的初始化，初始化完成后，MQTT 客户端会自动连接 OneNET 平台。数据的上传其实就是往特定的 topic 发布消息。当服务器有命令或者响应需要下发时，会将消息推送给设备。

获取数据流、数据点，发布命令则是基于 HTTP Client 实现的，通过 POST 或 GET 将相应的请求发送给 OneNET 平台，OneNET 将对应的数据返回，这样，我们就能在网页上或者手机 APP 上看到设备上传的数据了。

下图是应用显示设备上传数据的流程图

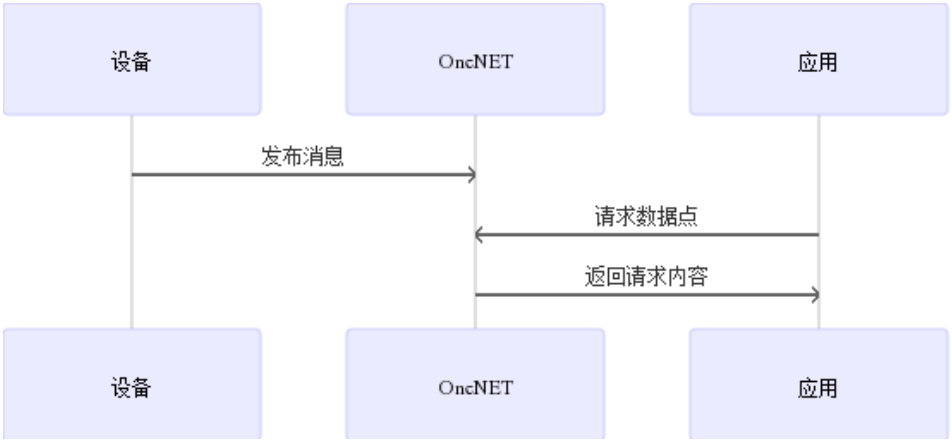

图 3.1: *onenet_upload*

下图是应用下发命令给设备的流程图

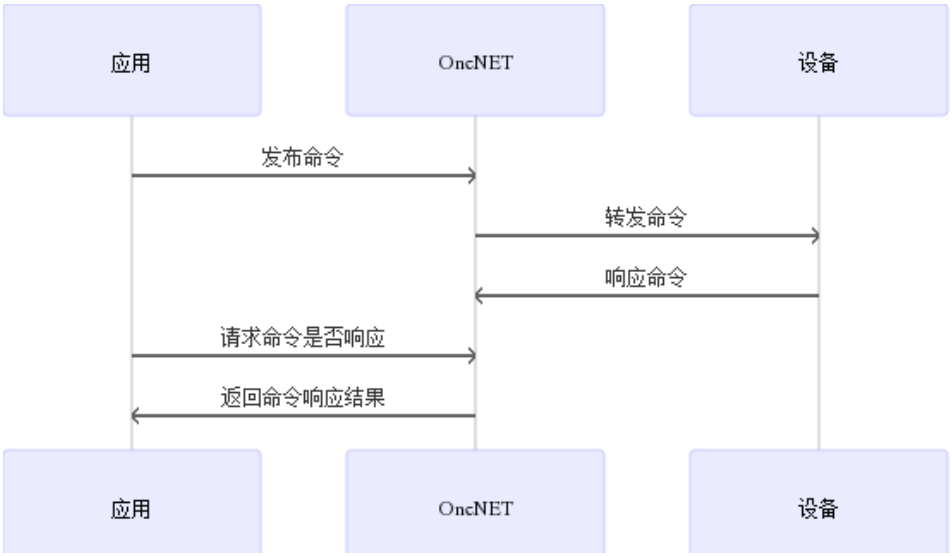

图 3.2: *onenet_send_cmd*

第 4 章

OneNET 包移植说明

本文主要介绍拿到 OneNET 软件包后，需要做的移植工作。

OneNET 软件包已经将硬件平台相关的特性剥离出去，因此 OneNET 本身的移植工作非常少，如果不启用自动注册功能就不需要移植任何接口。

如果启用了自动注册，用户需要新建 **onenet_port.c**，并将文件添加至工程。**onenet_port.c** 主要是实现开启自动注册后，获取注册信息、获取设备信息和保存设备信息等功能。接口定义如下所示：

```
/* 检查是否已经注册 */
rt_bool_t onenet_port_is_registed(void);
/* 获取注册信息 */
rt_err_t onenet_port_get_register_info(char *dev_name, char *auth_info);
/* 保存设备信息 */
rt_err_t onenet_port_save_device_info(char *dev_id, char *api_key);
/* 获取设备信息 */
rt_err_t onenet_port_get_device_info(char *dev_id, char *api_key, char *
    auth_info);
```

4.1 获取注册信息

rt_err_t onenet_port_get_register_info(char *ds_name, char* auth_info)

开发者只需要在该接口内，实现注册信息的读取和拷贝即可。

```
onenet_port_get_register_info(char *dev_name, char *auth_info)
{
```

```
    /* 读取或生成设备名字和鉴权信息 */

    /* 将设备名字和鉴权信息分别拷贝到 dev_name 和 auth_info 中*/
}
```

4.2 保存设备信息

rt_err_t onenet_port_save_device_info(char *dev_id, char* api_key)

开发者只需要在该接口内，将注册返回的设备信息保存在设备里即可。

```
onenet_port_save_device_info(char *dev_id, char *api_key)
{
    /* 保存返回的 dev_id 和 api_key */

    /* 保存设备状态为已注册状态 */
}
```

4.3 检查是否已经注册

rt_bool_t onenet_port_is_registed(void)

开发者只需要在该接口内，返回本设备是否已经在 OneNET 平台注册即可。

```
onenet_port_is_registed(void)
{
    /* 读取并判断设备的注册状态 */

    /* 返回设备是否已经注册 */
}
```

4.4 获取设备信息

rt_err_t onenet_port_get_device_info(char *dev_id, char* api_key, char *auth_info)

开发者只需要在该接口内，读取并返回设备信息即可

```
onenet_port_get_device_info(char *dev_id, char *api_key, char *auth_info)
{
    /* 读取设备id，api_key和鉴权信息 */

    /* 将设备id，api_key和鉴权信息分别拷贝到 dev_id，api_key 和 auth_info
        中*/
}
```

第 5 章

OneNET 软件包使用指南

5.1 准备工作

5.1.1 OneNET 注册及 ENV 配置

这一部分的内容请阅读软件包中示例说明文档，完成 OneNET 平台的注册和 ENV 的配置。

5.1.2 OneNET port 接口移植（仅适用于开启自动注册功能）

不启用自动注册不需要移植接口，如需启用自动注册功能，请详细阅读软件包中的移植说明文档，完成移植工作。

5.2 开始使用

5.2.1 OneNET 初始化

在 ENV 里面已经配置好了连接云平台需要的各种信息，直接调用`onenet_mqtt_init`函数进行初始化即可，设备会自动连接 OneNET 平台。

5.2.2 推送数据

当需要上传数据时，可以按照数据类型选择对应的 API 来上传数据。代码示例如下：

```
char str[] = { "hello world" };

/* 获得温度值 */
```

```
temp = get_temperature_value();
/* 将温度值上传到 temperature 数据流 */
onenet_mqtt_upload_digit("temperature",temp);

/* 将hello world上传到 string 数据流 */
onenet_mqtt_upload_string("string",str);
```

除了支持上传数字和字符串外，软件包还支持上传二进制文件。

可以通过onenet_mqtt_upload_bin或onenet_mqtt_upload_bin_by_path来上传二进制文件。代码示例如下

```
uint8_t buf[] = {0x01, 0x02, 0x03};

/* 将根目录下的1.bin文件上传到 bin 数据流 */
onenet_mqtt_upload_bin_by_path("bin", "/1.bin");
/* 将 buf 中的数据上传到 bin 数据流 */
onenet_mqtt_upload_bin(("bin", buf, 3);
```

5.2.3 命令接收

OneNET 支持下发命令，命令是用户自定义的。用户需要自己实现命令响应回调函数，然后利用onenet_set_cmd_rsp_cb将回调函数装载上。当设备收到平台下发的命令后，会调用用户实现的命令响应回调函数，等待回调函数执行完成后，将回调函数返回的响应内容再发给云平台。保存响应的内存必须是动态申请出来的，在发送完响应后，程序会自动释放申请的内存。代码示例如下：

```
static void onenet_cmd_rsp_cb(uint8_t *recv_data, size_t recv_size,
    uint8_t **resp_data,
size_t *resp_size)
{
  /* 申请内存 */

  /* 解析命令 */

  /* 执行动作 */

  /* 返回响应 */

}

int main()
```

```
{
    /* 用户代码 */

    onenet_mqtt_init();

    onenet_set_cmd_rsp_cb(onenet_cmd_rsp_cb);

    /* 用户代码 */

}
```

5.2.4 信息获取

数据流信息获取

用户可以通过onenet_http_get_datastream来获取数据流的信息，包括数据流 id，数据流最后更新时间，数据流单位，数据流当前值等等，获取的数据流信息会保存在传入的 datastream 结构体指针所指向的结构体中。代码示例如下

```
struct rt_onenet_ds_info ds_temp;

/* 获取到 temperature 数据流的信息后保存到 ds_temp 结构体中 */
onenet_http_get_datastream("temperature",ds_temp);
```

数据点信息获取

数据点信息可以通过以下 3 个 API 来获取

cJSON *onenet_get_dp_by_limit(char* ds_name, size_t limit);

cJSON *onenet_get_dp_by_start_end(char* ds_name, uint32_t start, uint32_t end, size_t limit);

cJSON *onenet_get_dp_by_start_duration(char* ds_name, uint32_t start, size_t duration, size_t limit);

这三个 API 返回的都是 cJSON 格式的数据点信息，区别只是查询的方法不一样，下面通过示例来讲解如何使用这 3 个 API。

```
/* 获取 temperature 数据流的最后10个数据点信息 */
dp = onenet_get_dp_by_limit("temperature",10);
```

```
/* 获取 temperature 数据流2018年7月19日14点50分0秒到2018年7月19日14点55分
    20秒的前10个
数据点信息 */
/* 第二、三个参数是Unix时间戳 */
dp = onenet_get_dp_by_start_end("temperature",1531983000, 1531983320, 10)
    ;

/* 获取 temperature 数据流2018年7月19日14点50分0秒往后50秒内的前10个数据
    点信息 */
/* 第二个参数是Unix时间戳 */
dp = onenet_get_dp_by_start_end("temperature",1531983000, 50, 10);
```

5.3 注意事项

- 设置命令响应回调函数之前必须要先调用onenet_mqtt_init函数，在初始化函数里会将回调函数指向 RT_NULL。
- 命令响应回调函数里存放响应内容的 buffer 必须是 malloc 出来的，在发送完响应内容后，程序会将这个 buffer 释放掉。

第 6 章

OneNET API

6.1 初始化

6.1.1 OneNET 初始化

int onenet_mqtt_init(void);

OneNET 初始化函数，需要在使用 OneNET 功能前调用。

参数	描述
无	无
返回	**描述**
0	成功
-1	获得设备信息失败
-2	mqtt 客户端初始化失败

6.1.2 设置命令响应函数

void onenet_set_cmd_rsp_cb(void(*cmd_rsp_cb)(uint8_t recv_data, size_t recv_size, uint8_t **resp_data, size_t *resp_size));

设置命令响应回调函数。

参数	描述
recv_data	接收到的数据

参数	描述
recv_size	数据的长度
resp_data	响应数据
resp_size	响应数据的长度
返回	**描述**
无	无

6.2 数据上传

6.2.1 mqtt 上传数据到指定主题

rt_err_t onenet_mqtt_publish(const char *topic, const uint8_t* msg, size_t len);

利用 mqtt 向指定 topic 发送消息。

参数	描述
topic	主题
msg	要上传的数据
len	数据长度
返回	**描述**
0	上传成功
-1	上传失败

6.2.2 mqtt 上传字符串到 OneNET

rt_err_t onenet_mqtt_upload_string(const char *ds_name, const char* str);

利用 mqtt 向 OneNET 平台发送字符串数据。

参数	描述
ds_name	数据流名称
str	要上传的字符串
返回	**描述**
0	上传成功

参数	描述
-5	内存不足

6.2.3 mqtt 上传数字到 OneNET

rt_err_t onenet_mqtt_upload_digit(const char *ds_name, const double digit);

利用 mqtt 向 OneNET 平台发送数字数据。

参数	描述
ds_name	数据流名称
digit	要上传的数字
返回	**描述**
0	上传成功
-5	内存不足

6.2.4 mqtt 上传二进制文件到 OneNET

rt_err_t onenet_mqtt_upload_bin(const char *ds_name, const uint8_t bin, size_t len);

利用 mqtt 向 OneNET 平台发送二进制文件。会动态申请内存来保存二进制文件，使用前请确保有足够的内存。

参数	描述
ds_name	数据流名称
bin	二进制文件
len	二进制文件大小
返回	**描述**
0	上传成功
-1	上传失败

6.2.5 mqtt 通过路径上传二进制文件到 OneNET

rt_err_t onenet_mqtt_upload_bin_by_path(const char *ds_name, const char bin_path);

利用 mqtt 向 OneNET 平台发送二进制文件。

参数	描述
ds_name	数据流名称
bin_path	二进制文件路径
返回	**描述**
0	上传成功
-1	上传失败

6.2.6 http 上传字符串到 OneNET

rt_err_t onenet_http_upload_string(const char *ds_name, const char str);

利用 http 向 OneNET 平台发送字符串数据，不推荐使用，推荐使用 mqtt 上传。

参数	描述
ds_name	数据流名称
str	要上传的字符串
返回	**描述**
0	上传成功
-5	内存不足

6.2.7 http 上传数字到 OneNET

rt_err_t onenet_http_upload_digit(const char *ds_name, const double digit);

利用 http 向 OneNET 平台发送数字数据，不推荐使用，推荐使用 mqtt 上传。

参数	描述
ds_name	数据流名称
digit	要上传的数字

参数	描述
返回	**描述**
0	上传成功
-5	内存不足

6.3 信息获取

6.3.1 获取数据流信息

rt_err_t onenet_http_get_datastream(const char *ds_name, struct rt_onenet_ds_info* datastream);

从 OneNET 平台获取指定数据流信息，并将信息保存在 datastream 结构体中。

参数	描述
ds_name	数据流名称
datastream	保存数据流信息的结构体
返回	**描述**
0	成功
-1	获取响应失败
-5	内存不足

6.3.2 获取最后 N 个数据点信息

cJSON *onenet_get_dp_by_limit(char* ds_name, size_t limit);

从 OneNET 平台获取指定数据流的 n 个数据点信息。

参数	描述
ds_name	数据流名称
limit	要获取的数据点个数
返回	**描述**
cJSON	数据点信息
RT_NULL	失败

6.3.3 获取指定时间内的数据点信息

cJSON *onenet_get_dp_by_start_end(char* ds_name, uint32_t start, uint32_t end, size_t limit);

从 OneNET 平台获取指定数据流指定时间段内的 n 个数据点信息。时间参数需要填入 Unix 时间戳。

参数	描述
ds_name	数据流名称
start	开始查询的时间
end	结束查询的时间
limit	要获取的数据点个数
返回	**描述**
cJSON	数据点信息
RT_NULL	失败

6.3.4 获取指定时间 n 秒的数据点信息

cJSON *onenet_get_dp_by_start_duration(char* ds_name, uint32_t start, size_t duration, size_t limit);

从 OneNET 平台获取指定数据流指定时间后 n 秒内的 n 个数据点信息。时间参数需要填入 Unix 时间戳。

参数	描述
ds_name	数据流名称
start	开始查询的时间
duration	要查询的秒数
limit	要获取的数据点个数
返回	**描述**
cJSON	数据点信息
RT_NULL	失败

6.4 设备管理

6.4.1 注册设备

rt_err_t onenet_http_register_device(const char *dev_name, const char* auth_info);

向 OneNET 平台注册设备，并返回设备 id 和 apikey。设备 id 和 apikey 会调用 `onenet_port_save_device_info`交由用户处理。

参数	描述
dev_name	设备名字
auth_info	鉴权信息
返回	**描述**
0	注册成功
-5	内存不足

6.4.2 保存设备信息

rt_err_t onenet_port_save_device_info(char *dev_id, char* api_key);

保存注册后返回的设备信息，需要用户实现。

参数	描述
dev_id	设备 id
api_key	设备 apikey
返回	**描述**
0	成功
-1	失败

6.4.3 获取设备注册信息

rt_err_t onenet_port_get_register_info(char *dev_name, char* auth_info);

获取注册设备需要的信息，需要用户实现。

参数	描述
ds_name	指向存放设备名字的指针
auth_info	指向存放鉴权信息的指针
返回	**描述**
0	成功
-1	失败

6.4.4 获取设备信息

rt_err_t onenet_port_get_device_info(char *dev_id, char api_key, char *auth_info);

获取设备信息用于登陆 OneNET 平台，需要用户实现。

参数	描述
dev_id	指向存放设备 id 的指针
api_key	指向存放设备 apikey 的指针
auth_info	指向存放鉴权信息的指针
返回	**描述**
0	成功
-1	失败

6.4.5 设备是否注册

rt_bool_t onenet_port_is_registed(void);

判断设备使用已经注册，需要用户实现。

参数	描述
无	无
返回	**描述**
RT_TURE	已经注册
RT_FALSE	未注册