

# 基于创易栈 ME3616 模块的 NB-IoT 上手玩教程

——据某谣言说：要想项目过得去，板子总得带点绿。

So，这篇教程的目标就是：点灯。

本篇教程的主要目的，是代码级实现 ME3616 模块向电信 IoT 平台上报数据，并且实现在电信 IoT 平台发送指令，点亮插在模块上的 STM32 Nucleo 板子上的一个 LED。

————— 高手现在可以抛下一个鄙视的眼神并关闭这篇文章的分界线 —————

## 一、完成上面这个目标，你需要先准备的一点操作和了解一点内容

关注创易栈公众号：emakerzone

关注 STM32 单片机公众号：STM32\_STM8\_MCU

完整的模块参考资料，请到网盘下载：

---

百度网盘链接：[https://pan.baidu.com/s/1AEB-O01\\_wHUoAr77ebV6bw](https://pan.baidu.com/s/1AEB-O01_wHUoAr77ebV6bw) 密码：gqtn

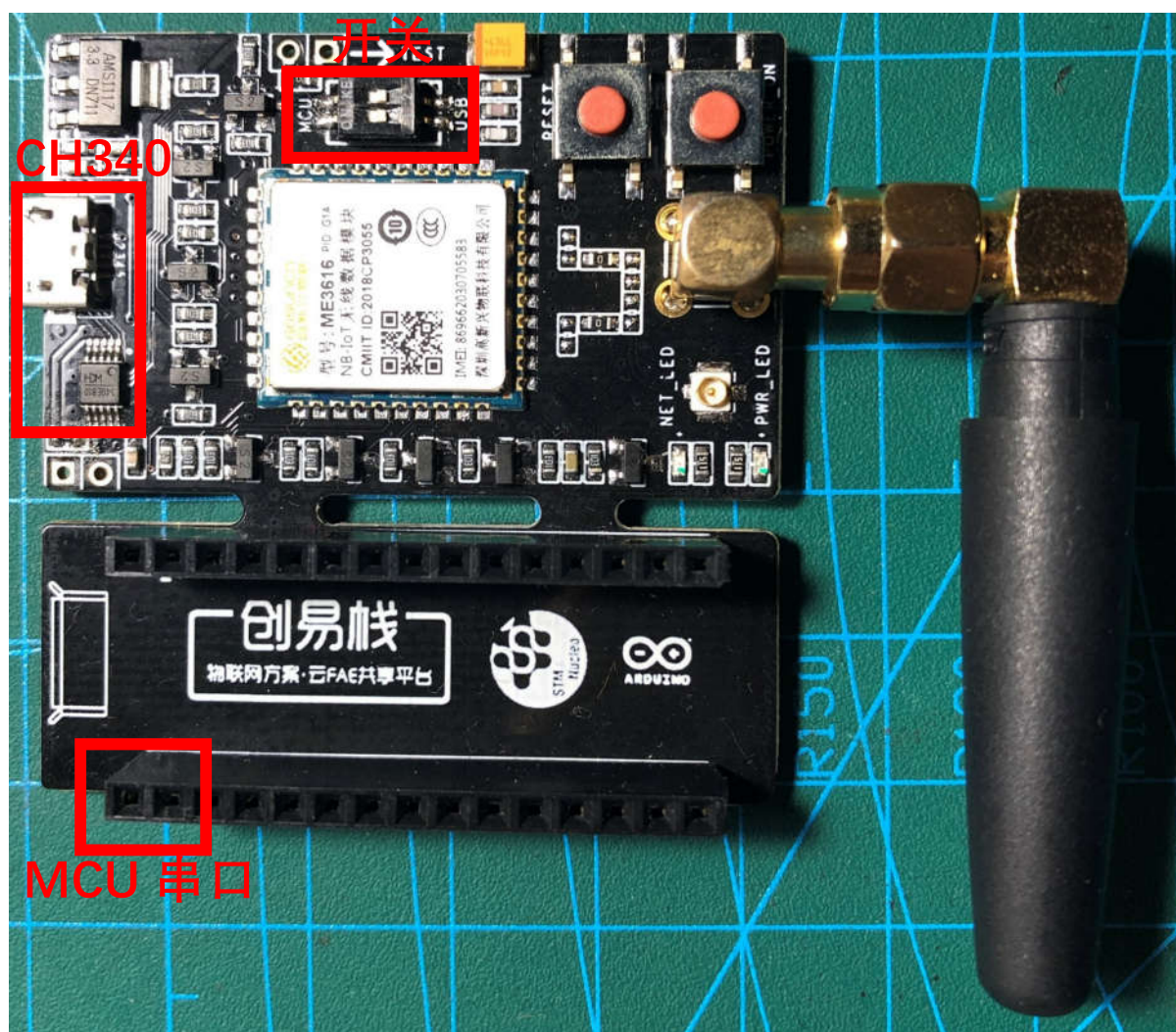
微云网盘链接：<https://share.weiyun.com/5rygWq>

---

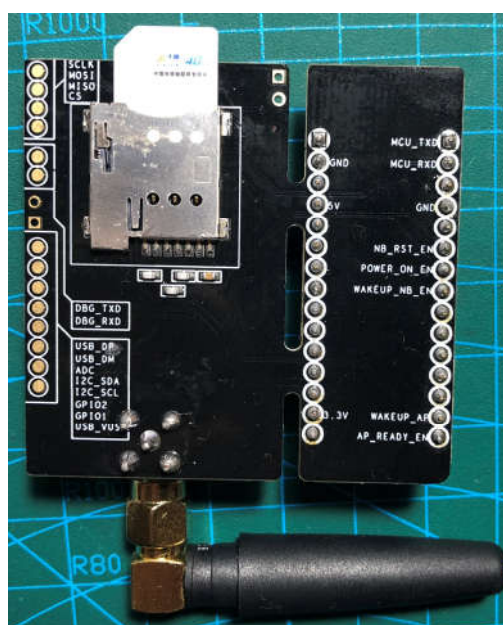
### ◆ 模块的硬件

如下图所示，ME3616 模块的串口，通过一个**选路开关**：一边通向板载的 CH340 串口，用作外部或手工调试；另一边通向 MCU 的 PA9、PA10（即 D0、D1，常用于 USART1 串口）。

串口的电路特性决定了，串口是**不可以**多路收发并联在一起使用的，所以使用的时候，根据需要把选路开关拨在对应的位置上。

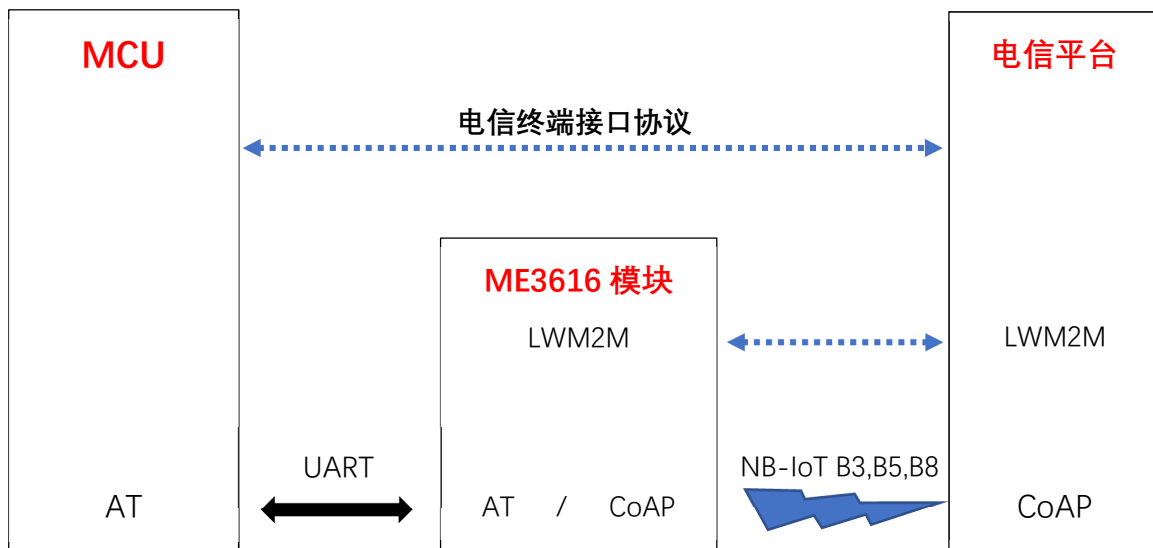


模块上电后，并不是立即就自动上报通信的，需要先手动拉低电源/复位引脚（按一下按钮）。



插入 SIM 卡，请注意观卡槽内触点的位置，SIM 卡缺口朝外，向内推到底。

## ◆ 通信逻辑简图



- MCU 始终使用 AT 指令，通过串口与模块通信。
- MCU 需要实现电信终端接口协议，其内容作为 payload，通过 AT 发送到模块。
- 模块与电信平台间，默认使用 NB-IoT Band 5，以 CoAP 作为其协议。
- 在 CoAP 以上，使用 LWM2M 作为其上层协议。
- 电信终端接口协议的消息格式说明，请查阅 <https://www.easy-iot.cn/document>

## ◆ 模块的准备和测试

1. 插上 SIM 卡，调整好选路开关
2. 插上 MicroUSB 线连上电脑，打开串口工具（建议使用 sscom），波特率 115200/8/1，无校验。

电脑上如果没有安装过 CH340 的串口驱动，这个时候需要安装一下，直到串口工具显示 CH340 串口。

3. 按一下电源/复位按钮，几秒钟之后串口打印出如下内容，说明 SIM 卡与模块工作正常

---

```

*MATREADY: 1
+CFUN: 1
+CPIN: READY
+IP: 10.180.200.143
+IP: 240e:e8:f1f1:26c5:1:2:41ee:3d67
  
```

---

- 如果出现----assert----、auto reboot 之类的内容，一般是因为电源按钮按的时间太短，等待一下或者重新按一下按钮就应该正常。
- 如果没有出现任何内容，请检查驱动是否安装正常，串口工具是否正确显示 CH340 串口，以及 USB 线与模块的开关及连接等等。

#### 4. 请输入并发送以下指令，检查模块的信息，其中 IMEI 和 IMSI 请保存下来，后面会用到：

发送指令前，请勾选串口工具中的：☒ 加回车换行

查询模块信息：ATI

请确认 SwRevision 结尾 为 **0B06** 或以上。

---

Manufacture: GOSUNCNWELINK

Model: ME3616

SwRevision: ME3616G1AV0.0B06

HwRevision: ME3616-G\_MB\_A

SVN: 65

IMEI: 86966203070xxxx <-保存这个号码

---

查询网络附着状态：AT+CEREG?

---

+CEREG: 0,1

---

查询物联网卡号 (IMSI)：AT+CIMI

---

46011300950xxxx <-保存这个号码

---

查询信号强度：AT+CESQ

---

+CESQ: 41,0,255,255,28,66

---

具体指令结果及其含义，请查阅 AT 命令手册。

若指令回复均满足手册要求，则模块的状态是 OK 的。否则请按照手册说明检查。

## ◆ 下载准备好的演示代码

---

<https://github.com/ssSimonn/ME3616-NB-IOT>

---

代码使用的开发平台为：

- **IAR 8.32**
- **Keil MDK 5.26.2 使用 Compiler 6.10.1**
  - Pack: Keil.STM32L4xx\_DFP.2.0.0
  - Pack: Keil.STM32L0xx\_DFP.2.0.0
- **STM32 CubeMX 4.27**
  - STM32Cube FW\_L4 V1.13.0
  - STM32Cube FW\_L0 V1.10.0

建议使用 STM32L432KC Nucleo 或 STM32L031K6 Nucleo 这两块板子，演示工程已经做好，只需要更改项目文件夹下，Drivers/ME3616/SRC/ Me3616\_app.c 中的 client\_imei、client\_imsi 号后，编译下载，就完成了。下载后，可以 Debug 运行，也可以直接复位 nucleo 运行。





## 二，电信平台上的操作

### 1、注册并开通电信 Easy IoT 平台账号

请到 <https://www.easy-iot.cn>，根据提示，完成注册账号的步骤。



### 2、在电信平台中，创建并设置产品的信息。

- 从左侧进入  产品开发，右上角 
- 请按照以下信息设置（因为演示代码就是按照这个写的）

基本信息	传感器	设备消息	设备指令	
名称: Sensor_1	显示名称: 传感器_1	读写类型: 读写	数据类型: Byte(1字节)	显示Logo: 
TLVType: 1	最小值: -128	最大值: 127	步进值: 1	数据单位: Degree
名称: Sensor_2	显示名称: 传感器_2	读写类型: 读写	数据类型: Int(4字节)	显示Logo: 
TLVType: 2	最小值: 0	最大值: 9999	步进值: 1	数据单位: Round
名称: Led_Green	显示名称: 指示灯	读写类型: 读写	数据类型: Bool(布尔,1...	显示Logo: 
TLVType: 3	取值: off(0)-on(1)-1			

此处定义传感器的类型和数据

基本信息	传感器	<u>设备消息</u>	设备指令
------	-----	-------------	------

消息代码: 1

消息ID: msg\_1

消息参数:

传感器\_1

传感器\_2

指示灯

+

此处定义模块发送至平台的消息结构

基本信息	传感器	设备消息	<u>设备指令</u>
------	-----	------	-------------

指令代码: 2

指令ID: CMD\_1

消息参数:

指示灯

+

响应参数:

执行结果

指示灯

+

此处定义平台发送至设备的命令结构

- 点击右上角保存。至此，平台侧的设置完成。

### 三、代码写入并观察结果

- 如果你正在使用 L432kc Nucleo 或 L031 Nucleo，直接打开对应的项目文件（IAR 或 Keil）
- 如果你使用其他的 MCU，请跳到下一节，代码移植的部分。

1. 更改代码中的 imei 及 imsi 号码，改为你的模块实际的数值（位于 me3616\_app.c 中）

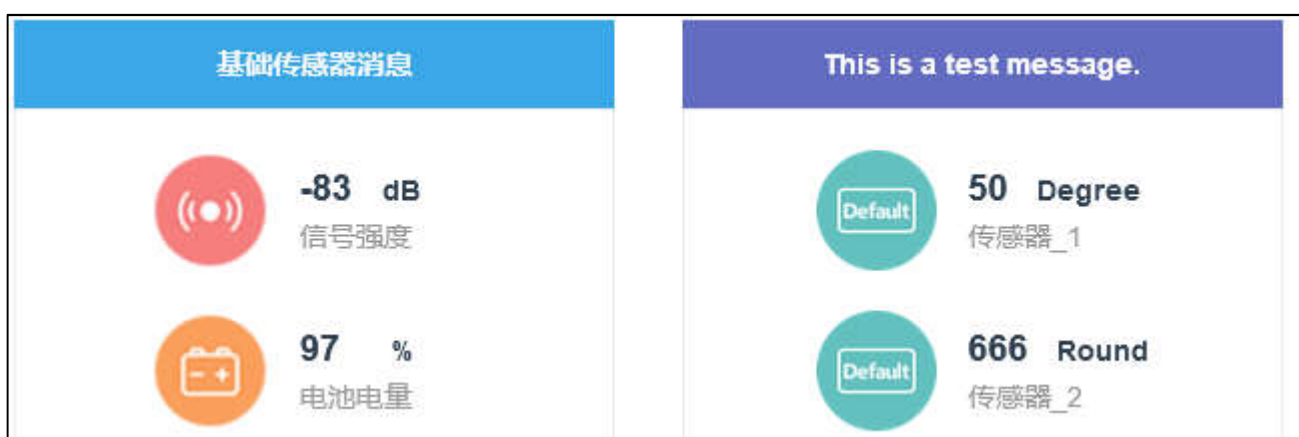
```
char * client_imei = "86966203070xxxx";
char * client_imsi = "46011300950xxxx";
```

2. 检查 ME3616 的串口选路开关。电脑连接好 Nucleo 的 ST-Link 虚拟串口，打开串口工具（连接 ST-Link 虚拟串口，115200/8/1 无校验）。
3. 编译，正常应为 0 Error 0 Warning。调试运行，或代码写入 MCU 后，按 Nucleo 的复位键。可以看到串口打印的 log：

```
[2][APP]: Start of App ME3616A
[2980][MCU_AT]: MATREADY Below:
[2982][Rx]: *MATREADY: 1
[2985][MCU_AT]: Sys_State Changed. New state is:
[2989][MCU_AT]: 40000101
[2992][MCU_AT]: CFUN Below:
[2994][Rx]: +CFUN: 1
[2996][MCU_AT]: Sys_State Changed. New state is:
[3001][MCU_AT]: 40000503
[4143][MCU_AT]: CPIN Below:
[4145][Rx]: +CPIN: READY
[4147][MCU_AT]: Sys_State Changed. New state is:
[4152][MCU_AT]: 40000703
[10015][MCU_AT]: IP Below:
[10017][Rx]: +IP: 10.47.33.172
[10020][MCU_AT]: Sys_State Changed. New state is:
[10024][MCU_AT]: 40000f03
[14041][MCU_AT]: IP Below:
[14043][Rx]: +IP: 240e:e8:f0f4:952b:1:1:c4a7:5a5f
[14048][MCU_AT]: Sys_State Changed. New state is:
[14052][MCU_AT]: 40001f03
```

看到以上内容，说明代码已经正常运行

4. 当出现 M2MSEND 发送后，回到电信平台的网页，查看设备信息，传感器的数值应该变为如下状态：





- 其中，信号强度从通过 AT 从模块读取。电量数字取自 me3616 模块的 ADC 值。
- 上传的传感器的值，位于 me3616\_app.c 中：

```
AddInt8(msg, SENSOR_1_TLV_PARAMID, 50);
AddInt32 (msg, SENSOR_2_TLV_PARAMID, 666);
```

5. 点击电信平台->设备管理中的 指令图标，向模块发送指令，如下图：

6. 最长等待 90 秒后（可修改代码），指令被模块接收，可以观察到 Nucleo 板的绿色 LED 点亮。

```
[101702][MCU_AT]: M2MCLI Below:
[101704][Rx]: +M2MCLI:register update success
[102720][MCU_AT]: M2MCLIRECV Below:
[102723][Rx]: +M2MCLIRECV:01F2000A000E0200040300010116
[102728][APP]: coap hex input 28, to binary 14.

[102732][APP]: new message from static buffer at 0x20000d66.
[102738][APP]: prepare deserialize buffer 0x20000d58, length: 14
[102744][APP]: message deserialize succ, tlv count: 1
[102749][APP]: recv cmd.
[102752][APP]: found cmdid 2 handler at 0x80078d9, process it.
[102758][APP]: coap input process finished, ret 4.

[106568][APP]: new message from static buffer at 0x20000ee8.
[106573][APP]: set msg 0x20000ee8 msgid: 2, type: 243.
[106579][APP]: message using static buffer, using static push messages.
[106585][APP]: prepare serialize message 0x20000ee8, sensor count: 2
[106592][APP]: message type is CMT_USER_CMD_RSP.

[106597][Tx]: AT+M2MCLISEND=01F3000E000E020008000001000300010120
[106704][Rx]: AT+M2MCLISEND=01F3000E000E020008000001000300010120
[106709][MCU_AT]: AT OK Confirmed.
[106713][Rx]: OK
[106714][MCU_AT]: M2MCLI Below:
[106717][Rx]: +M2MCLI:notify success
[106721][MCU_AT]: Sys_State Changed. New state is:
[106725][MCU_AT]: 400f1f07
```

- 模块在每次 update 的时候获得平台命令，具体请查看模块文档关于 PSM、eDRX 等模式。

7. 至此，点灯动作完成，演示结束。

## 四、主要代码及移植注意事项

◆ 代码除了 STM32 CubeMX 生成的部分，需要自行加入另外两部分，均为开源：

1、包含 ME3616 指令集、AT 发送/接收、MCU 串口驱动以及应用代码的：

me3616.c、me3616-if.c、me3616-app.c

2、电信开源 SDK，完成电信平台侧的协议：

easyiot.c

◆ AT 指令发送由 ME3616\_Send\_AT\_Command ()完成，参数主要有：

```
bool ME3616_Send_AT_Command(Me3616_DeviceType * Me3616, AT_CMD_t at_cmd, AT_Action_t at_action, bool override, char * pch)
```

➤ 目标 Me3616 实例

➤ 目标 AT 指令

可用指令位于 me3616.h 的 enum AT\_CMD\_t 中。

➤ 扩展指令格式

可选择扩展 (AT\_Action\_Base)、Set (AT\_Action\_Set)、Read (AT\_Action\_Read)、Test (AT\_Action\_Test) 四种。

➤ override

是否强制发送。若该值为 true，将忽略上一次指令的状态（包括上一条指令是否有回复），直接发送。有可能会造成模块响应错误等问题，用户需要自行处理错误。

若该值为 false，发送动作会等待在上一条指令状态为 ATOK 或 AT\_None 的状态下才会发送，否则取消 AT 指令发送，设置 AT 状态为超时 timeout，并返回 false。用户需要使用 Get\_AT\_State() / Set\_AT\_State()来检查和修改上一条指令的状态。

➤ 指令内容

若扩展指令格式为 AT\_Action\_Set，此处为参数的内容。

➤ 返回值

若模块回复 OK / ERROR，则返回 true。否则返回 false，并设置 AT 状态为超时 timeout。

详细的 AT 指令内容，请参考《模组 AT 指令手册》

◆ 在 AT 指令发送后，收到回复 OK/Error 前，模块向 MCU 发回的内容都交 Command\_Response()处理。

◆ 收到 OK/Error 后，模块再向 MCU 发回的内容称为“主动上报”，交由对应的 Callback 函数执行。

◆ 移植方面，主要关注的地方有：

◆ 请根据目标 MCU 的资源，配置 me3616.h

两个串口：MCU 与 me3616 的通信串口，以及 MCU 通过 ST-Link 的打 log 串口。

调整发送/接收等各种 Buffer 的大小。

◆ 根据串口通信的方式，配置 me3616\_if.c

本 DEMO 使用 DMA+CM 方式接收和发送。

接收内容为以字符开头（需要去掉字符串前面的 CR LF），'\n\0'结尾的字符串，每接收一个字符串执行一次 RxHandler()。

◆ 根据目标 MCU 的资源，配置 easyiot.h

修改#define MESSAGE\_MAX\_TLV，根据实际的传感器数量调整。

◆ 根据目标 MCU 的资源，配置 me3616\_app.c

修改 EasyIoT SDK 所需要的 buff 大小（me3616\_app.c）。

◆ 请按照以下内容修改 easyiot.c 的 CoapOutput()函数

```

1464: // CoAP 数据输出，此处的实现为 BC95，其他模组需要根据AT指令集做对应修改
1465: int CoapOutput(uint8_t *inBuf, uint16_t inLength)
1466: {
1467:     // int i, all_length;
1468:     // char headbuf[16];
1469:
1470:     if (!gl_nb_out) {
1471:         Logging(LOG_WARNING, "nb serial output cb is null, pls use setNbSerialOutputCb set
1472:         return -1;
1473:     }
1474:
1475:     // memset(headbuf, 0, sizeof(headbuf));
1476:     // sprintf(headbuf, "AT+NMGS=%d,", inLength);
1477:     // all_length = strlen(headbuf);
1478:
1479:     gl_nb_out((uint8_t*)inBuf, inLength);
1480:
1481:     for (i = 0; i != inLength; ++i) {
1482:         // sprintf(headbuf, "%02X", inBuf[i]);
1483:         // gl_nb_out((uint8_t*)headbuf, 2);
1484:         // all_length += 2;
1485:     }
1486:     //
1487:     // gl_nb_out((uint8_t*)"\\r\\n", 2);
1488:     // return all_length + 2;
1489:     return inLength;
1490: } // end CoapOutput

```

- ◆ 由于 STM32L031 内核为 Cortex-M0+，基于 ARMv6-M，并不支持非对齐操作，需要对以下函数稍作调整，确保静态分配模式下的 message 对齐：

```

1570: // ASCII HEX格式的CoAP数据输入处理，首先调用 a2b_hex，然后直接 CoapInput
1571: // 类同 CoapHexInput，只是并没有在函数里直接使用一块较大的内存空间，使用了指定的buffer处理
1572: int CoapHexInputStatic(const char* data, uint8_t* inBuf, uint16_t inMaxLength)
1573: {
1574:     int ret;
1575:     int ret_copy;
1576:
1577:     __attribute__((aligned(4))) struct Messages* msg;
1578:
1579:     ret = a2b_hex(data, (char*)inBuf, inMaxLength);
1580:     if (ret < 0) {
1581:         Logging(LOG_WARNING, "ascii to binary hex failed.\n");
1582:         return -1;
1583:     }
1584:     Logging(LOG_TRACE, "coap hex input %d, to binary %d.\r\n", strlen(data), ret);
1585:
1586:     //make ret_copy aligned(4) to message
1587:     ret_copy = ret;
1588:     if(ret_copy % 4 != 0) ret_copy += (4 - ret_copy % 4);
1589:
1590:     msg = NewMessageStatic(inBuf + ret_copy, inMaxLength - ret_copy);
1591:     ret = CoapInput(msg, inBuf, ret);
1592:     if (ret < 0) {
1593:         Logging(LOG_WARNING, "coap input process failed.\n");
1594:         return -1;
1595:     }
1596:     Logging(LOG_TRACE, "coap input process finished, ret %d.\r\n", ret);
1597:
1598:     return ret;
1599: } « end CoapHexInputStatic »

```

```

188: // 静态初始化 Message 结构体，使用此函数构造的 Message 在随后的 addtlv中，亦不会动态分配内存
189: struct Messages* NewMessageStatic(uint8_t * buf, uint16_t inMaxLength)
190: {
191:     __attribute__((aligned(4))) struct Messages * msg;
192:
193:     if (!buf) {
194:         Logging(LOG_WARNING, "new message from static, but buffer is null.\n");
195:         return NULL;
196:     }
197:     if (inMaxLength < sizeof(struct Messages)) {
198:         Logging(LOG_WARNING, "new message from static, but buffer too small.\n");
199:         return NULL;
200:     }
201:
202:     msg = (struct Messages *)buf;
203:     memset(msg, 0, sizeof(struct Messages));
204:
205:     msg->sbuf_use = 1;
206:     msg->sbuf = buf;
207:     msg->sbuf_offset = sizeof(struct Messages);
208:     msg->sbuf_maxlength = inMaxLength;
209:     Logging(LOG_TRACE, "new message from static buffer at 0x%p.\r\n", msg);
210:
211:     return msg;
212: } « end NewMessageStatic »

```

```

56: struct Messages {
57:
58:     /* 静态内存分配区，在 MessageMalloc函数中，实现了一个简单的静态内存分配机制，即只分配，不释放。
59:     uint8_t * sbuf;
60:     // dtag_mid, 在上行数据中使用 dtag 语义，用于在一个较短的时间内，去除重复上行的数据
61:     // 在下行指令中，取其mid语义，上行的指令响应必须使用同样的mid值，用以关联指令的执行结果。
62:     uint16_t dtag_mid;
63:
64:     // msgid, 消息ID，即为整个消息的Type，在EasyIoT平台定义消息时，此值会被定义
65:     // 在导出的头文件中，亦有此值的宏定义，代码中建议使用宏定义。
66:     uint8_t msgid;
67:
68:     // 当前Message对象中的TLV个数总和
69:     uint8_t tlv_count;
70:
71:
72:
73:     // 内存buffer起始地址，一旦赋值，将不再变动，具体内存使用，由 sbuf_offset 值确定
74:     uint16_t sbuf_offset;
75:     // 原始 sbuf 的最大长度
76:     uint16_t sbuf_maxlength;
77:     // 是否使用静态内存分配机制
78:     uint8_t sbuf_use;
79:     /* 静态内存分配区 */
80:
81:     // enum TlvValueType 值，用于区分是何种类型数据
82:     uint8_t msgType;
83:
84:     // TLV对象指针数组
85:     struct TLV* tlvs[MESSAGE_MAX_TLV];
86: } « end Messages » ;

```