基于 VGUS 的 Lua 脚本使用说明

2023-10-31

在 VUGS 组态软件中增加 Lua 脚本代码，目的是方便用户在界面中实现各种用户自定义的逻辑功能。

首先，在 Lua 脚本的加持下，VGUS 串口屏能实现哪些功能。最基本的就是支持添加用户自定义的逻辑功能，实现逻辑控制，同时减少通讯的频率；支持自定义串口协议，串口协议选择更加灵活；集成 Modbus 协议，使用更加方便。

本文主要讲述了在 VGUS 串口屏下，如何使用 Lua 脚本的回调函数、API 接口函数以及注意事项。首先介绍了四个回调函数，回调函数是不同事件的程序入口，也就是用户 Lua 代码的入口。然后介绍了各类 API 接口函数，API 接口函数作为 Lua 代码与 VGUS 进行数据交换的桥梁，支持读写寄存器、变量存储器等，开放了定时器、串口、Modbus 等相关接口功能。最后介绍了常用的 Lua 脚本编辑工具、Lua 脚本文件下载和 Lua 调试注意事项。

目 录

1 基本回调函数说明

回调函数是指在 Lua 脚本文件中，VGUS 主动调用的函数。这类函数是我们提前定义好的函数名称，作为各类事件的入口函数。开机时会调用初始化回调函数，有按钮操作时，会调用触摸回调函数。用户在对应的回调函数中编写自己的 Lua 脚本逻辑代码，串口屏会根据回调函数类型，在不同的时间点执行对应回调函数里面的用户 Lua 脚本程序。回调函数不支持相互调用和嵌套。

1.1 初始化回调函数 callback_init

callback_init()

说明：在显示开机画面前调用，可用于相关变量初始化操作，本回调函数仅执行一次。

1.2 定时器超时回调函数 callback_timer

callback_timer(timer_id)

timer_id：表示超时定时器 ID。ID 值为 0~31。

说明：根据开启定时器的模式不同，可能周期执行，也可以只执行一次。

1.3 触控回调函数 callback_touch

callback_touch(pic_id,key_code,touch_state)

pic_id：表示触摸控件所在页面。

key_code：触摸控件的按键键码。可在上位机指定，同一个页面按键键码不可重复。

没有设置按键键码触摸控件，有按钮操作时不会触发此回调函数。

touch_state：表示触摸状态。0x01 表示第一次按下；0x03 表示持续按下；0x02 表示抬起。

一般情况下，仅需处理按钮抬起状态即可。特殊应用场合才需要处理第一次按下或持续按下。

说明：按钮状态返回和增量调节控件，按下到持续按下的触发间隔为 500ms，后续持续按下的触发间隔为 1 个 VGUS 周期。滑动调节和转动调节，按下到持续按下以及后续持续按下的触发间隔为 1 个 VGUS 周期。其他触摸控件只触发按下和抬起。

弹窗显示时，弹窗中按钮触发该回调函数，pic_id 参数为弹窗页面。

1.4 串口回调函数 callback_uart

callback_uart(com_num,recv_len)

com_num：串口编号。

recv_len：对应串口接收缓冲区可读取的字节数。缓冲区大小 4096 字节。

说明：该回调函数仅对自定义串口协议有效。只有所选串口的工作模式为自定义串口协议时，串口缓冲区中有接收到数据会触发该回调函数。需要在回调函数中及时将对应串口接收缓存区的数据读出。

2 API 接口函数说明

API 接口函数是 Lua 脚本与 VGUS 沟通的桥梁。每个 API 接口函数都代表一个具体的功能。合理的使用相关 API 接口函数，结合 Lua 脚本实现的逻辑，达到控制 VGUS 的目的。

API 接口函数已实现如下功能：

(1) 开启/停止软件定时器；

(2) 串口工作模式配置，读取/发送串口数据；

(3) 获取/设置寄存器的值；

(4) 获取/设置变量存储器的值；

(5) 写曲线缓存区；

(6) 扩展指令功能；

(7) Modbus 主机/从机协议；

2.1 定时器相关

2.1.1 开启定时器vgus_timer_start

vgus_timer_start(timer_id,tmr_mode, count_mode, timeout)

timer_id：表示定时器 ID。ID 值为 0~31。

tmr_mode：定时模式：0：单次模式；1：周期模式。

count_mode：计数方式：0：向上计数；1：向下计数。

timeout：定时器超时时间，单位为 ms，最小值为 20。

返回值：无。

2.1.2 获取定时器当前计数值vgus_timer_getvlue

vgus_timer_getvlue(timer_id)

timer_id：表示定时器 ID

返回值：对应定时器 ID 的当前计数值。

2.1.3 停止定时器vgus_timer_stop

vgus_timer_stop(timer_id)

timer_id：表示定时器 ID

返回值：无。

2.2 串口相关

2.2.1 设置串口工作模式com_set_work_mode

com_set_work_mode(com_num,work_protocol,baudrate,format)

com_num：串口编号。编号范围 0-1。

work_protocol：工作协议。

0：VGUS 指令集协议。

1：自定义串口协议。

2：ModbusRTU 从机。

3：ModbusASCII 从机。

4：ModbusRTU 主机。

5：ModbusASCII 主机。

6：下载调试协议（仅串口 1 支持）。

baudrate：串口波特率。

com_format：串口格式。

0：7N1（7 数据位，无校验，1 停止位）；

1：7E1（7 数据位，偶校验 EVEN，1 停止位）；

2：7O1（7 数据位，奇校验 ODD，1 停止位）；

3：7N2（7 数据位，无校验，2 停止位）；

4：8N1（8 数据位，无校验，1 停止位）；

5：8E1（8 数据位，偶校验 EVEN，1 停止位）；

6：8O1（8 数据位，奇校验 ODD，1 停止位）；

7：8N2（8 数据位，无校验，2 停止位）。

返回值：无。

说明：

(1) 串口工作模式可以不设置，默认工作模式如下：

串口 0：VGUS 协议，组态软件中屏参配置所设置的波特率，串口格式为 8N1。

串口 1：自定义串口协议，波特率默认为 115200，串口格式默认为 8N1。

(2) 调试下载协议说明：串口的数据发送功能可正常使用。数据发送功能指通过 print、com_data_send 函数发送的数据以及 Lua 的调试信息通过该串口输出，用于打印相关调试信息。串口接收到数据后，不通过串口回调函数通知 Lua 脚本，而是内部解析是否有 8505 指令用于联机和启动串口文件下载。

(3) 串口 0 支持更改工作协议，除了下载调试协议不支持，其它协议均支持。

(4) 串口 1 支持自定义串口协议或下载调试协议。

(5) 使用 ModbusRTU 协议（从机或主机）时，串口格式必须选择 8 数据位，否则将默认配置为 8E1 模式。

(6) 该 API 接口函数仅支持在初始化回调函数（callback_init）中使用。

2.2.2 设置Lua调试串口com_set_debug_print

com_set_debug_print(com_num)

com_num：串口编号。编号范围 0-1。

返回值：无。

说明：该 API 接口函数仅在实际串口屏中有效。**虚拟串口屏**的调试信息统一在“**输出窗口**”输出。默认调试串口为串口 1。Lua 脚本的调试信息（print 打印信息或错误信息）将通过所设置的串口输出。需要注意硬件上串口 1（丝印 J20）是否引出，否则需要使用串口 0 调试。串口 0 为用户主串口。此时协议数据和 Lua 调试信息都从该串口输出。

2.2.3 获取串口接收缓冲区待读取字节数com_data_len

com_data_len(com_num)

com_num：串口编号。编号范围 0-1。

返回值：所选串口接收缓冲区待读取字节数。0 表示无数据可读。

说明：所选串口使用自定义串口协议时，该 api 可用。

2.2.4 读取串口接收缓冲区数据com_data_read

com_data_read(com_num, read_len, read_table)

com_num：串口编号。编号范围 0-1。

read_len：读取长度，单位字节。单次最大 1024。

read_table：保存读取的数据。字节数组，从 1 开始索引。

返回值：实际读取到的字节数。0 表示无数据可读。

说明：所选串口使用自定义串口协议时，该 api 可用。

在调用该 API 接口函数时，可以将读取长度统一默认用单次最大 1024，然后根据函数返回值来确认实际读取到的字节数。这样就可以不用事先去获取缓冲区中的字节数。

2.3.5 通过串口发送数据com_data_send

com_data_send(com_num ,send_len, send_table)

com_num：串口编号。编号范围 0-1。

send_len：发送长度，单位字节。单次最大 1024。

send_table：待发送数据。字节数组，从 1 开始索引。

返回值：无。

2.3 寄存器相关

2.3.1 读寄存器vgus_reg_read

vgus_reg_read(reg_addr, read_len, read_table)

reg_addr：寄存器地址。

read_len：读取字节数量。

read_table：读取数据，字节数组，从 1 开始索引。

返回值：成功返回实际读取的字节数量，失败返回 nil。

2.3.2 写寄存器vgus_reg_write

vgus_reg_write(reg_addr, write_len, write_table)

reg_addr：寄存器地址。

read_len：写入字节数量。

write_table: 写入数据，字节数组，从 1 开始索引。

返回值：成功返回实际写入的字节数量，失败返回 nil。

2.4 变量存储器相关

2.4.1 读变量存储器vgus_vp_read

vgus_vp_read(vp_addr, read_len, read_table)

vp_addr：变量存储器地址。

read_len：读取数量，单位字（双字节）。单次最大 1024。

read_table：保存读取的数据，字数组，从 1 开始索引。

返回值：成功返回实际读取的字数量，失败返回 nil。

2.4.2 写变量存储器vgus_vp_write

vgus_vp_write(vp_addr, write_len, write_table)

reg_addr：变量存储器地址。

write_len：写数量，单位字（双字节）。单次最大 1024。

write_table: 待写入数据，字数组，从 1 开始索引。

返回值：成功返回实际写入的字数量，失败返回 nil。

2.4.3 以数值方式写变量存储器vgus_vp_var_write

vgus_vp_var_write(vp_addr, var_type,var_value)

vp_addr：变量存储器地址。

var_type：数值类型。

var_value：待写入数值。

返回值：成功返回实际写入的数值，失败返回 nil。

数值类型说明：

0：整数(两字节)：-32768 到 32767

1：长整数(4 字节)：-2147483648 到 2147483647

2：VP*高字节：0 到 255

3：VP*低字节：0 到 255

4：超长整数(8 字节)：-9223372036854775808 到 9223372036854775807

5：无符号整数(2 字节)：0 到 65536

6：无符号长整数(4 字节)：0 到 4294967295

用法说明：

可结合数据变量使用，方便快捷写入数值到变量存储器。

“数值类型”需要和数据变量控件的“变量类型”相匹配。

2.4.4 以数值方式读变量存储器vgus_vp_var_read

vgus_vp_var_read(vp_addr, var_type)

vp_addr：变量存储器地址。

var_type：数值类型。

返回值：成功返回读到的数值，失败返回 nil。

说明：数值类型具体定义见 vgus_vp_var_write。

2.4.5 以字符串方式写变量存储器vgus_vp_string_write

vgus_vp_string_write(vp_addr,str_value)

vp_addr：变量存储器地址。

str_value：待写入字符串。

返回值：成功返回实际写入的字节数，失败返回 nil。

说明：一般情况下，实际写入的字节数为字符串长度加 2（添加结束符 0xFFFF）。

使用文本变量控件显示写入的非英文字符串，需要 Lua 文件的编码方式与文本变量控件的编码方式一致。

2.4.6 以字符串读变量存储器vgus_vp_string_read

vgus_vp_string_read(vp_addr)

vp_addr：变量存储器地址。

返回值：成功返回读到的字符串，失败返回 nil。

说明：单次读取字符串的最大长度为 2048 字节，遇到 0xff 或 0x00 提前结束。

2.4.7 以屏蔽方式写变量存储器vgus_vp_mask_write

vgus_vp_mask_write(vp_addr, And_Mask,Or_Mask)

vp_addr：变量存储器地址。

And_Mask：AND 屏蔽。

Or_Mask：OR 屏蔽。

返回值：成功返回对应地址的计算结果，失败返回 nil。

说明：该功能用于设置或清除变量存储器中的特定比特。

功能算法：

$$\text{结果} = (\text{当前内容 AND And\_Mask}) \text{ OR } (\text{Or\_Mask AND } \overline{\text{And\_Mask}})$$

如果 Or_Mask 值为零，那么结果是当前内容和 And_Mask 的简单逻辑 AND（与）。

如果 And_Mask 值为零，结果等于 Or_Mask 值。

注意：当前内容为变量存储器对应地址的数据。

2.5 写曲线缓存区相关

2.5.1 写曲线缓存区数据vgus_curve_write

vgus_curve_write(CH_Mode, write_len, write_table)

CH_Mode：通道模式。

write_len：曲线数据长度，单位字（双字节）。单次最大 128。

write_table: 待写入曲线数据表，字数组，从 1 开始索引。

返回值：成功返回实际写入的字数量，失败返回 nil。

说明：关于曲线通道和曲线数据的定义及说明，请参考开发指南 4.3.1“曲线缓冲区写指令 0x84”相关说明。

2.6 扩展指令相关

2.6.1 直接写显存0x85 00指令功能vgus_memory_write

vgus_memory_write(X, Y, DATA_len, DATA)

X：起始位置的 X 坐标。

Y：起始位置的 Y 坐标。

DATA_len: 显存数据长度，单位字（双字节）。单次最大 128。

DATA: 待写入显存数据，字数组，从 1 开始索引。

返回值：无。

说明：关于直接写显存，请参考《VGUS 串口屏用户开发指南》2.4 扩展指令 0x85。

2.6.2 连续播放音频文件0x85 03指令功能vgus_play_audio

vgus_play_audio(Mode, DATA_len, DATA)

Mode：播放模式：0 循环播放，1 顺序播放，其他停止播放。

DATA_len: 播放音频个数，单位字（双字节）。单次最大 128。

DATA: 待播放音频文件名编号，字数组，从 1 开始索引。

返回值：无。

说明：关于连续播放音频文件，请参考《VGUS 串口屏用户开发指南》2.4 扩展指令 0x85。

2.7 Modbus 主机相关

通过 com_set_work_mode 可以将串口 0 配置为 Modbus 主机，支持 RTU 或 ASCII。此时，屏作为 Modbus 主机，通过以下功能 API 触发。屏内部会按照 Modbus 协议打包并发送数据，然后将最终的结果返回。Modbus 功能码对应的 API 接口函数，执行后会返回错误码，错误码定义如下：

0：无错误。

1：非法寄存器地址。

2：非法参数。

3：接收数据错误。

4：应答超时错误。

5：主机忙。

6：执行功能错误。

2.7.1 获取Modbus异常码modbus_get_exception_code

modbus_get_exception_code()

返回值：Modbus 异常码。

说明：当 Modbus 主机接收到执行功能错误时，可以通过该 API 接口函数获取具体的异常码。异常码具体含义，请查看 Modbus 相关规范。

2.7.2 设置应答超时时间modbus_set_timeout

modbus_set_timeout(timeout)

timeout：从机应答超时时间。单位 ms，默认值为 100。

返回值：无。

2.7.3 读线圈（01）modbus_fun_01

modbus_fun_01(slave, addr, quantity, coils)

slave：从机 ID。

addr：读线圈地址。范围 0x0000-0xFFFF。

quantity：读线圈数量。范围 1-2000。

coils: 读取的线圈数据，字节数组，从 1 开始索引。8 个线圈一个字节。

返回值：成功返回 0，错误返回错误码。

2.7.4 读离散输入（02）modbus_fun_02

modbus_fun_02(slave, addr, quantity, input)

slave：从机 ID。

addr：读离散输入地址。范围 0x0000-0xFFFF。

quantity：读离散输入数量。范围 1-2000。

input: 读取的离散输入数据，字节数组，从 1 开始索引。8 个离散输入一个字节。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.7.5 读保持寄存器（03）modbus_fun_03

modbus_fun_03(slave, addr, quantity, regs)

slave：从机 ID。

addr：读保持寄存器地址。范围 0x0000-0xFFFF。

quantity：读保持寄存器数量。范围 1-125。

regs: 读取的保持寄存器数据，字数组（双字节），从 1 开始索引。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.7.6 读输入寄存器（04）modbus_fun_04

modbus_fun_04(slave, addr, quantity, input_regs)

slave：从机 ID。

addr：读输入寄存器地址。范围 0x0000-0xFFFF。

quantity：读输入寄存器数量。范围 1-125。

input_regs: 读取的输入寄存器数据，字数组（双字节），从 1 开始索引。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.7.7 写单个线圈（05）modbus_fun_05

modbus_fun_05(slave, addr, status)

slave：从机 ID。

addr：写单个线圈地址。范围 0x0000-0xFFFF。

status：线圈值。范围 0-1。0 为 OFF，1 位 ON。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.7.8 写单个寄存器（06）modbus_fun_06

modbus_fun_06(slave, addr, reg)

slave：从机 ID。

addr：写保持寄存器地址。范围 0x0000-0xFFFF。

reg：保持寄存器值。范围 0x0000-0xFFFF。

仅支持无符号短整形数据，传入负数会返回非法参数错误。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.7.9 写多个线圈（15）modbus_fun_15

modbus_fun_15(slave, addr, quantity,coils)

slave：从机 ID。

addr：写线圈地址。范围 0x0000-0xFFFF。

quantity：写线圈数量。范围 1-1968。

coils: 待写入的线圈数据，字节数组，从 1 开始索引。8 个线圈一个字节。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.7.10 写多个寄存器（16）modbus_fun_16

modbus_fun_16(slave, addr, quantity,regs)

slave：从机 ID。

addr：写保持寄存器地址。范围 0x0000-0xFFFF。

quantity：写保持寄存器数量。范围 1-120。

regs: 待写入的保持寄存器数据，字数组（双字节），从 1 开始索引。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.7.11 屏蔽写寄存器（22）modbus_fun_22

modbus_fun_22(slave, addr, And_Mask,Or_Mask)

slave：从机 ID。

addr：屏蔽写寄存器地址。范围 0x0000-0xFFFF。

And_Mask：AND 屏蔽。范围 0x0000-0xFFFF。

Or_Mask：OR 屏蔽。范围 0x0000-0xFFFF。

返回值：成功返回 0，错误返回 Modbus 错误码。

2.8 Modbus 从机相关

通过 com_set_work_mode 可以将串口 0 配置为 Modbus 从机，支持 RTU 或 ASCII。

Modbus 从机模式，仅需通过 Modbus_set_salveaddr 设置从机地址。

Modbus 从机模式，通过支持的功能码自动获取/设置相关数据，无需 Lua 脚本参与。

Lua 脚本可用于实现屏内的相关逻辑。

2.8.1 设置Modbus从机设备地址modbus_set_salveaddr

modbus_set_salveaddr(salve)

slave：作为 Modbus 从机（RTU 或 ASCII）使用时的设备 ID。范围 1-255。默认值 1。

返回值：无。

注意：该 API 需要在初始化回调中调用。仅设置一次。

2.8.2 Modbus从机支持的功能码

表 2-1：Modbus 从机支持的功能码

Modbus 功能码	地址范围	功能	单次读写数据长度	说明	VGUS 指令功能对应关系
03	0x0000-0xEFFF	读变量存储器	1-125	-	与83指令功能一致
	0xF000-0xF0FF	读取寄存器	1-125	地址低字节表示待读取寄存器地址。返回数据高位填充 0,低字节为有效寄存器数据。	与81指令功能一致
	0xF100-0xFFFF	非法地址	-	-	-
06	0x0000-0xEFFF	写单个变量存储器	1	-	与82指令功能一致
	0xF000-0xF0FF	写单个寄存器	1	地址低字节表示待读取寄存器地址。发送数据高位填充 0,低字节为有效寄存器数据。	与80指令功能一致
	0xF100-0xF1FF	写 1 个数据到单个曲线缓存区	1	地址低字节表示通道模式 后续数据格式定义与 84 指令一致。	与84指令功能一致

	0xF200-0xFFFF	非法地址	-	-	-
16	0x0000-0xEFFF	写多个变量存储器	1-123	-	与82指令功能一致
	0xF000-0xF0FF	写多个寄存器	1-123	地址低字节表示待读取寄存器地址。发送数据高位填充 0,低字节为有效寄存器数据。	与80指令功能一致
	0xF100-0xF1FF	写多个数据到曲线缓存区	1-123	地址低字节表示通道模式。后续数据格式定义与 84 指令一致	与84指令功能一致
	0xF200-0xF2FF	扩展指令功能	1-123	地址低字节表示扩展指令 0x85 的具体功能码。	与85指令功能一致
	0xF300-0xFFFF	非法地址	-	-	-
22	0x0000-0xEFFF	设置或清除变量存储器中指定地址的特定比特	1	-	-
	0xF000-0xFFFF	非法地址	-	-	-
01	0x000-0x1FF	读线圈	-	-	-
05	0x000-0x1FF	写单个线圈	-	-	-
15	0x000-0x1FF	写多个线圈	-	-	-

2.8.3 Modbus从机22功能码说明

该功能码用于设置或清除变量存储器中指定地址的特定比特。

地址范围：0x0000-0xEFFF。

功能算法：

$$结果 = (当前内容\ \text{AND}\ \text{And\_Mask})\ \text{OR}\ (\text{Or\_Mask}\ \text{AND}\ \overline{\text{And\_Mask}})$$

如果 Or_Mask 值为零，那么结果是当前内容和 And_Mask 的简单逻辑 AND（与）。如果 And_Mask 值为零，结果等于 Or_Mask 值。

注：当前内容为变量存储器对应地址的数据。

2.8.4 Modbus从机位操作区（线圈）说明

在变量存储器中开辟一块区域，用于位操作区。可结合位变量图标使用。

位操作区地址与变量存储器地址对应关系如表 2 所示：

使用 01、05、22 功能码时，读取或写入的地址需要使用表 2-2 的地址。

线圈地址范围：0-512

表 2-2：位操作区地址与变量存储器地址对应关系

变量存储器地址	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
0x0100	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0101	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0x0102	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32

0x0103	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48
0x0104	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65	64
0x0105	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81	80
0x0106	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97	96
0x0107	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113	112
0x0108	143	142	141	140	139	138	137	136	135	134	133	132	131	130	129	128
0x0109	159	158	157	156	155	154	153	152	151	150	149	148	147	146	145	144
0x010A	175	174	173	172	171	170	169	168	167	166	165	164	163	162	161	160
0x010B	191	190	189	188	187	186	185	184	183	182	181	180	179	178	177	176
0x010C	207	206	205	204	203	202	201	200	199	198	197	196	195	194	193	192
0x010D	223	222	221	220	219	218	217	216	215	214	213	212	211	210	209	208
0x010F	255	254	253	252	251	250	249	248	247	246	245	244	243	242	241	240
0x0110	271	270	269	268	267	266	265	264	263	262	261	260	259	258	257	256
0x0111	287	286	285	284	283	282	281	280	279	278	277	276	275	274	273	272
0x0112	303	302	301	300	299	298	297	296	295	294	293	292	291	290	289	288
0x0113	319	318	317	316	315	314	313	312	311	310	309	308	307	306	305	304
0x0114	335	334	333	332	331	330	329	328	327	326	325	324	323	322	321	320
0x0115	351	350	349	348	347	346	345	344	343	342	341	340	339	338	337	336
0x0116	367	366	365	364	363	362	361	360	359	358	357	356	355	354	353	352
0x0117	383	382	381	380	379	378	377	376	375	374	373	372	371	370	369	368
0x0118	399	398	397	396	395	394	393	392	391	390	389	388	387	386	385	384
0x0119	415	414	413	412	411	410	409	408	407	406	405	404	403	402	401	400
0x011A	431	430	429	428	427	426	425	424	423	422	421	420	419	418	417	416
0x011B	447	446	445	444	443	442	441	440	439	438	437	436	435	434	433	432
0x011C	463	462	461	460	459	458	457	456	455	454	453	452	451	450	449	448
0x011D	479	478	477	476	475	474	473	472	471	470	469	468	467	466	465	464
0x011E	495	494	493	492	491	490	489	488	487	486	485	484	483	482	481	480
0x011F	511	510	509	508	507	506	505	504	503	502	501	500	499	498	497	496

2.8.5 Modbus从机指令示例

配置为 Modbus 从机模式：

com_set_work_mode(0,2,115200,4)　--将串口 0 设置为 ModbusRTU 从机模式

com_set_work_mode(0,3,115200,4)　--将串口 0 设置为 ModbusASCII 从机模式

配置为 Modbus 从机设备地址：

modbus_set_salveaddr(0x01)　--配置 Modbus 从机设备地址为 0x01

以下示例指令无特殊说明则使用 RTU 模式。使用 0x01 作为从机设备地址。

(1) 使用 03 指令读变量存储器

从变量存储器 0x0020 地址开始读取 1 个数据。

发送（0x）：01 03 00 20 00 01 85 C0

成功应答（0x）：01 03 02 12 34 B5 33

变量存储器 0x0020 地址的数据为 0x1234。

上述功能使用 ASCII 模式的发送和应答：

发送（0x）：3A 30 31 30 33 30 30 32 30 30 30 30 31 44 42 0D 0A

成功应答（0x）：3A 30 31 30 33 30 32 31 32 33 34 42 34 0D 0A

(2) 使用 03 指令读寄存器

从寄存器 0x03 地址开始读取 2 个变量数据（获取页面编号）。

发送（0x）：01 03 F0 03 00 02 07 0B

成功应答（0x）：01 03 04 00 00 00 02 7B F2

寄存器 0x03 0x04 地址的数据为 0x00 0x02。（读取数据的低字节为寄存器的有效数据）

上述功能使用 ASCII 模式的发送和应答：

发送（0x）：3A 30 31 30 33 46 30 30 33 30 30 30 32 30 37 0D 0A

成功应答（0x）：3A 30 31 30 33 30 34 30 30 30 30 30 30 30 32 46 36 0D 0A

(3) 使用 06 指令写单个变量存储器

将变量存储器 0x0020 地址的数据写为 0x5566。

发送（0x）：01 06 00 20 55 66 37 7A

成功应答（0x）：01 06 00 20 55 66 37 7A

(4) 使用 06 指令写单个寄存器

控制蜂鸣器鸣叫 200ms。

发送（0x）：01 06 F0 02 00 14 1B 05

成功应答（0x）：01 06 F0 02 00 14 1B 05

(5) 使用 06 指令写 1 个数据到单个曲线缓存区

曲线缓冲区通道 0 写入数据 0x0060。

发送（0x）：01 06 F1 01 00 60 EA DE

成功应答（0x）：01 06 F1 01 00 60 EA DE

(6) 使用 16 指令写多个数据到变量存储器

将字符“hello word”写入 0x0200 为起始地址的变量存储器（0xFFFF 为结束符）。

发送（0x）：01 10 02 00 00 06 0C 68 65 6C 6C 6F 20 77 6F 72 64 FF FF F9 95

成功应答（0x）：01 10 02 00 00 06 41 B3

(7) 使用 16 指令写多个数据到寄存器

切换到 2 号页面。

发送（0x）：01 10 F0 03 00 02 04 00 00 00 02 36 7F

成功应答（0x）：01 10 F0 03 00 02 82 C8

(8) 使用 16 指令写多个数据到曲线缓存区

通道 0，通道 1，通道 7 分别写入两个数据。

发送（0x）：01 10 F1 83 00 06 0C 00 10 00 20 00 30 00 40 00 50 00 60 C2 D1

成功应答（0x）：01 10 F1 83 00 06 83 1F

0x83 表示数据顺序为：（通道 0+通道 1＋通道 7）＋…＋（通道 0＋通道 1＋通道 7）。

通道 0 数据：0x0010 0x0040

通道 1 数据：0x0020 0x0050

通道 7 数据：0x0030 0x0060

(9) 使用 16 指令实现直接写显存功能（VGUS 扩展指令功能 8500）

以（100,100）为起始坐标，设置 4 个像素点为红色（0xF800）。

发送（0x）：01 10 F2 00 00 06 0C 00 64 00 64 F8 00 F8 00 F8 00 F8 00 F7 8D

成功应答（0x）：01 10 F2 00 00 06 72 B3

数据 0xF2 0x00 表示直接写显存指令。

(10) 使用 16 指令实现连续播放音频文件功能（VGUS 扩展指令功能 8503）

连续播放 1 次编号为 1,2,3 的音频文件。

发送（0x）：01 10 F2 03 00 04 08 01 00 01 00 02 00 03 00 34 98

成功应答（0x）：01 10 F2 03 00 04 03 72

数据 0xF2 0x03 表示连续播放音频文件指令。

(11) 使用 22 指令设置或清除变量存储器中指定地址的特定比特。

以屏蔽写方式修改变量存储器地址 0x0020 数据。

当前内容（地址 0x0020 数据）=0x0012

And_Mask=0x00F2

Or_Mask=0x0025

结果=0x0017

发送（0x）：01 16 00 20 00 F2 00 25 17 E9

成功应答（0x）：01 16 00 20 00 F2 00 25 17 E9

(12) 使用 01 指令读线圈

从线圈地址 0 开始，读取 10 个线圈。

发送（0x）：01 01 00 00 00 0A BC 0D

成功应答（0x）：01 01 02 5A 01 42 9C

根据应答数据，得到 10 个线圈数据对应关系如表 2-3 所示。

表 2-3：读线圈地址与数据对应关系

线圈地址	7	6	5	4	3	2	1	0	-	-	-	-	-	-	9	8
数据	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	1
数据（0x）	0x5A								0x01							

(13) 使用 05 指令写单个线圈

将线圈地址 2 写为“ON”。

发送（0x）：01 05 00 02 FF 00 2D FA

成功应答（0x）：01 05 00 02 FF 00 2D FA

(14) 使用 15 指令写多个线圈

从线圈地址 0 开始，写 10 个线圈。写入数据为 0xA3 0x02。

线圈数据对应关系如表 2-4 所示。

表 2-4：写多个线圈地址与数据对应关系

线圈地址	7	6	5	4	3	2	1	0	-	-	-	-	-	-	9	8
数据	1	0	1	0	0	0	1	1	0	0	0	0	0	0	1	0
数据（0x）	0xA3								0x02							

发送（0x）：01 0F 00 00 00 0A 02 A3 02 1C 09

成功应答（0x）：01 0F 00 00 00 0A D5 CC

2.9 Lua 调试相关

2.9.1 获取Lua内存池使用百分比getmemusage

getmemusage()

返回值：Lua 内存池使用百分比。

说明：**SDWb 系列仅 2023.10.10 及以后固件支持该 API。**

1．默认固件 Lua 内存池总大小为 512KB，getmemusage()返回值为 25，表示当前已使用（512*25%）KB。

2．Lua 存在垃圾回收机制，但不是立即回收，使用 getmemusage()前调用 collectgarbage("collect") 回收内存，更加准确的反馈当前内存池的使用情况。详细信息请查看 Lua 资料的“垃圾回收”章节。

3．不建议频繁调用该 API，会降低整体的运行效率。建议在调试阶段调用，查看 Lua 内存使用情况，避免使用过渡、内存泄漏等造成的未知问题。

3 Lua 脚本使用注意事项

本节主要讲述开发过程中需要注意的相关事项。从脚本文件的编辑、下载、调试、以及运行环境要求等方面进行相关注意事项的说明。

3.1 Lua 脚本学习

可以通过以下网站学习 Lua 脚本。仅需了解一些基本概念即可，后续可结合相关例程进行学习和验证。

https://www.runoob.com/lua/lua-basic-syntax.html

3.2 SDWb 串口屏 Lua 脚本命名要求

SDWb 串口屏使用的 Lua 脚本文件与工程文件相对独立。可以单独保存，建议保存在工程文件根目录。

Lua 脚本文件命名要求如下：

(1) 命名格式参考：SDWb_LUAxxxx.lua。

(2) “SDWb_LUA”为下载时的识别字符，不允许更改。

(3) “xxx”为自定义字符，可以是编号，产品型号等。仅支持英文字符。

(4) “.lua”为 lua 文件后缀，方便脚本编辑软件自动识别文件格式。

(5) 文件名称总长度不能超过 60 字节。

3.3 Lua 脚本编辑及编译工具

Lua 脚本文件下载之前，需要通过编辑工具检查是否存在语法错误。

注意：编辑器只能检测基本的语法错误，具体的业务逻辑需要到屏中调试。

3.3.1 Lua脚本编辑器推荐1：Notepad++（8.14版本）

Notepad++编辑器界面如图3-1所示。

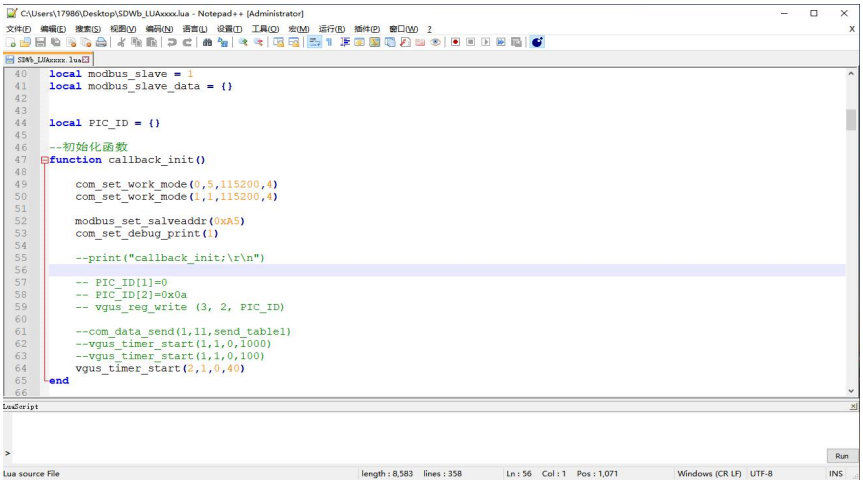

图 3-1：Notepad++界面

安装包：Notepad++_8.1.4.zip，解压后安装默认方式安装即可。

支持语法高亮，语法提示，语法折叠，多行注释，多种插件。

安装 lua 插件：插件->插件管理->搜索“LuaScript”->点击“安装”即可。

语法检查：插件->LuaScript->Execute Current File。如果有语法错误，底部会有红色字体提示。

3.3.2 Lua脚本编辑器推荐2：LuaEditor（6.30版本）

LuaEditor 编辑器界面如图 3-2 所示。

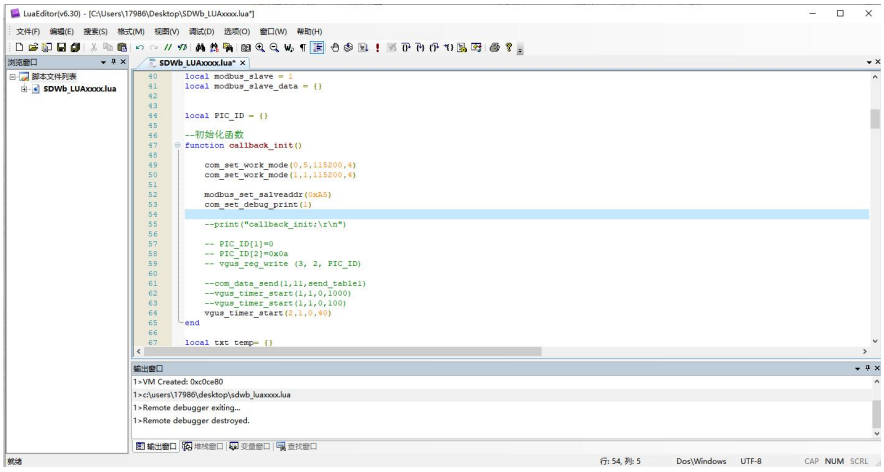

图 3-2：LuaEditor 界面

安装包：LuaEditorPro_v6.30.zip，解压后点击 LuaEditor.exe 即可运行，无需安装。

支持语法高亮，语法提示，语法折叠，多行注释等功能。

语法检查：点击“调试”->“编译脚本”进行语法检查。

3.3.3 Lua脚本编译工具

Lua 脚本的编译不是必选项，未编译的 Lua 脚本文件可直接下载到 SDWb 系列串口屏运行。

Lua 脚本的编译有以下好处：编译时会删除注释，有利于减小文件大小；编译后的文件不具备可读性，一定程度上保护 Lua 代码中的逻辑功能。

VGUS 开发工具（20231018 及以后版本的开发工具）提供 Lua 代码编译器。可在 VGUS 开发工具中的“工具”栏录中找到并打开该工具。

分别设置待编译 Lua 文件的路径、编译后的文件名以及编译后的文件保存路径，点击编译即可进行编译。Lua 编辑器界面如图 3-3 所示。编译后的文件名也要符合命名要求。

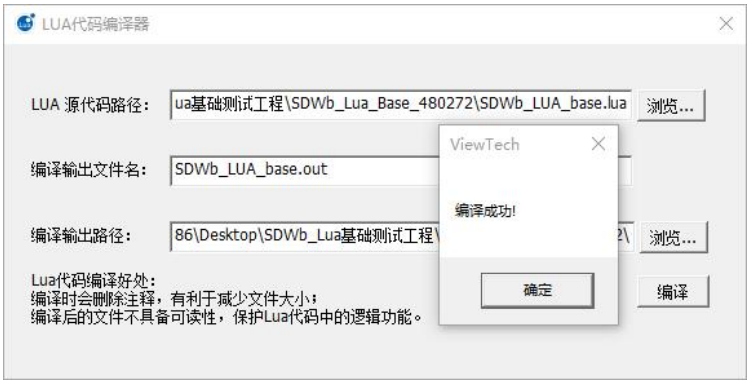

图 3-3：Lua 代码编译器界面

注意：编译后的 Lua 代码需要 SDWb 系列 2023.10.10 及以后固件才能运行。

3.4 SDWb 串口屏 Lua 脚本固件版本要求

版本固件：仅 SDWb 系列串口屏提供 Lua 版本固件。

SDWbLua 版本固件命名：SDWbLua_Firmwarexxx.bin。

(1)“SDWbLua_Firmware”为固件识别字符，不可更改。

(2)“xxx”为扩展信息，支持修改，一般为固件日期。

固件更新限制：目前存在两类固件。

SDWb 串口屏标准固件 SDWb_Firmwarexxx.bin。（标准固件不支持 Lua 脚本功能）

SDWb 串口屏 Lua 固件 SDWbLua_Firmwarexxx.bin。

当前为标准固件，只能更新标准固件。

当前为 Lua 固件，能更新到 Lua 固件或标准固件。支持回退到标准固件，工程文件需要重新下载。

回退到标准固件后不支持重新更新到 Lua 固件。

3.5 SDWb 串口屏 Lua 脚本文件下载

3.5.1 使用 TF 卡下载

将 SDWb_LUAxxxx.lua 文件复制到 TF 卡根目录。先插 TF 卡再上电，串口屏识别到文件后自动开始下载。下载的 Lua 脚本文件需要先在编辑器中检查，确保没有语法错误，再下载到串口屏中调试。

下载时，TF 卡根目录**只允许有一个符合命名要求**的 Lua 脚本文件，否则不能确定哪个 Lua 脚本文件将被下载到屏里，造成实际运行结果与预期不一致的问题。

注意：下载时，可以下载工程文件但是无法下载 Lua 脚本文件，可能是当前固件不是 Lua 固件。

3.5.2 使用串口下载

详细串口下载使用说明请参考《串口在线下载文件功能说明》。

通过串口下载 Lua 脚本文件，可以避免频繁插拔 TF 卡及上电，提高开发效率。

3.6 SDWb 串口屏 Lua 脚本调试

3.6.1 Lua 脚本调试的错误提示及调试信息

Lua 脚本的错误提示：

(1) 语法错误：在 Lua 编辑器中有基本的语法提示，点击编译或运行时，会提示相关语法错误。

(2) 运行错误：该错误发生在运行过程中，如参数类型不匹配，未定义函数，内存不足等错误。

VGUS 支持在运行过程中输出错误或者其他调试信息。可以使用 print 输出自定义的调试信息。

3.6.2 使用虚拟串口屏调试

调试文件：将符合命名要求的 Lua 脚本文件放置于工程根目录下，仅可放置一个符合命名要求的 Lua 脚本文件。命名要求详见 3.2 章节。

配置串口：Lua 脚本支持使用两路串口，分别为串口 0 和串口 1，需要在“屏参配置”的“虚拟串口屏 COM 配置”中指定到具体的串口。为了方便与串口调试助手对接，建议使用 VSPD 软件生成所需的虚拟串口，相关用法请参考官网“资料下载”的“工具软件”选项。

运行虚拟串口屏时，Lua 脚本文件将被自动加载，加载结果可以在工程“输出窗口”查看。

使用 print 打印信息以及 Lua 脚本的调试信息（一般为错误信息，如参数类型不匹配，未定义函数，

内存不足等错误）固定在“输出窗口”输出。使用 com_set_debug_print 设置无效。

如图 3-3 所示。“输出窗口”显示了 Lua 脚本文件的加载成功以及初始化回调函数中 print("callback_init;\r\n")语句的执行结果。

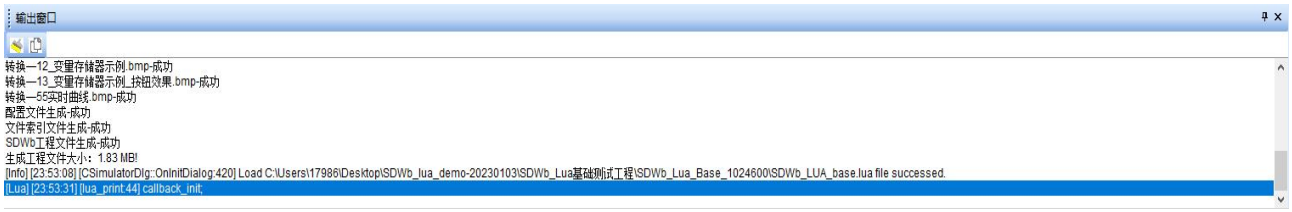

图 3-4：VGUS 输出窗口

3.6.2 使用 SDWb 串口屏调试

参照 3.5 的说明，下载 Lua 脚本文件到 SDWb 串口屏。

SDWb 串口屏支持两路串口，可以指定调试信息（print 输出信息或 Lua 调试信息）从哪路串口输出。

使用 API 接口函数 com_set_debug_print(com_num)设置调试串口。

默认使用串口 1 作为调试串口，这样可以和串口 0 的协议数据分开，方便调试。

注意：使用串口 1，要先确认所用 SDWb 串口屏硬件上是否有引出该路串口（丝印 J20）。