VisualHMI - 寄存器操作

本章节以DCBUS为例，介绍寄存器的读写API使用。HMI数据类型包括bit、int16/uint16、int32/uint32、int64/uint64、float、double、string，批量写短整型等

> 适用范围：VisualHMI - HMI&M系列&Dx系列
>
> 例程下载链接：[ViusalHMI - 寄存器读写(点击下载)]
>
> 其他参考链接：[ViusalHMI - 一主多从(点击跳转)]

1. get_bit/ set_bit

1.1.API说明

1.1.set_bit(vtype, addr, value， count)

多位寄存器批量写入函数，`set_bit(vtype, addr, value, count)` 是 HMI 系统提供的**批量位寄存器写入函数**，用于**一次性设置连续多个位寄存器（bit）或单个位寄存器（bit）的状态**，适用于 Modbus RTU的线圈（Coils）、FX2N 的 X/Y/S/M 等位寄存器。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：**如 Modbus RTU 的`VT_0x`（线圈）、FX3U的`VT_M`（内部继电器）等
`addr`	number	寄存器地址
`value`	number	**16 位无符号整数值** 其低 `value` 表示目标位状态（bit0 → 地址 `addr`，bit1 → `addr+1`，...）
`count`	number	**选填，写入位数** 取值范围 1 ~ 16

1.2.get_bit (vtype, addr)

位寄存器读取函数。`get_bit(vtype, addr)` 是 HMI 系统提供的**位读取函数**，用于读取单个**位地址**的状态**（布尔量：0 或 1）。该函用于读取** Modbus 线圈/输入。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：**如 Modbus RTU 的`VT_0x`（线圈）、FX3U的`VT_M`（内部继电器）等
`addr`	number	**寄存器地址**
返回值	number	0 或 1，表示该位的当前状态

> 🔑 `get_bit(vtype, addr)` **是从 HMI 的本地缓存（内存镜像）中读取数据，而非实时向 PLC/设备发起串口请求。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，ModbusRTU为例，自动从 PLC 读取 `VT_0`、`VT_1x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_bit(VT_1x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

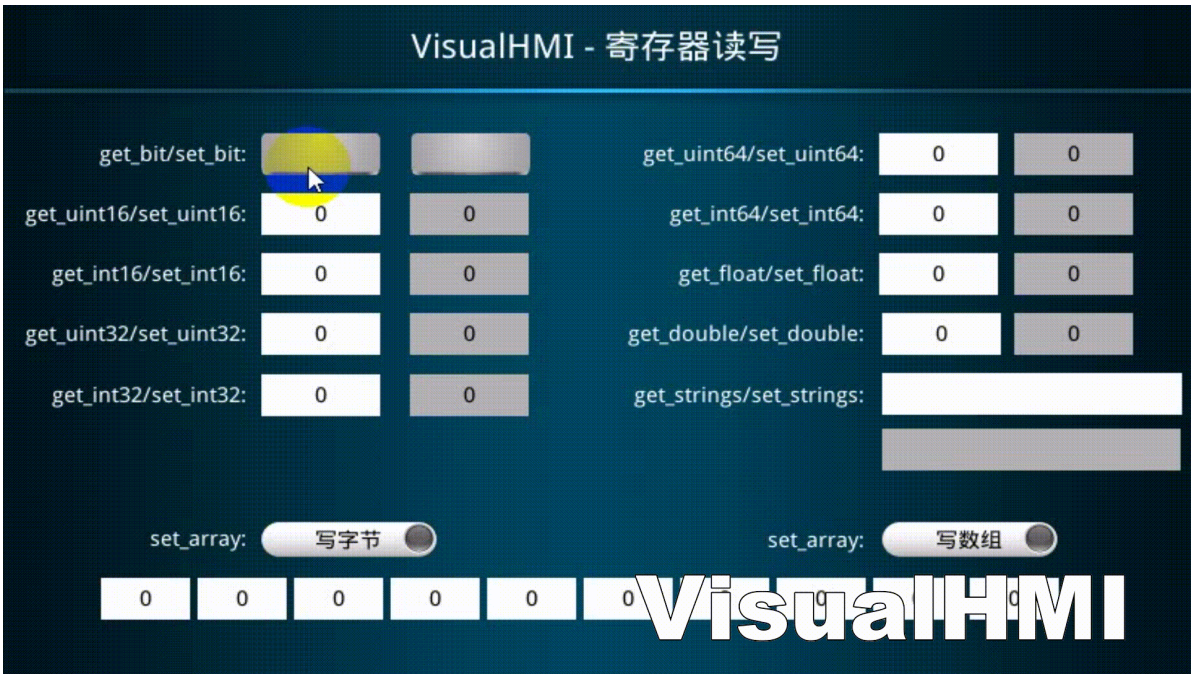
1.2.LUA脚本

点击线圈0x0000位状态指示灯，触发on_update回调，读取线圈0x0000，并设置线圈0x0001。

```
function on_update(slave,vtype,addr)
    if vtype == VT_0x
    then
        if addr == 0x0000
        then
            set_bit(VT_0x, 0x0001, get_bit(VT_0x, 0x0000))
        end
    end
end
```

1.3.运行预览

运行虚拟屏，点击线圈0x0000，线圈0x0001实时变化，如下所示：



2. get_uint16/set_uint16

2.1. API说明

2.2.1.set_uint16(vtype, addr, value)

16位无符号整型寄存器写入函数，`set_uint16(vtype, addr, value)` 是 HMI 系统提供的**16 位无符号整数写入函数**，用于向指定**地址**写入一个 **0～65535** 范围的数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型**
`addr`	number	**寄存器地址**
`value`	number	**写入值：**取值范围 0 ～ 65535（超出将自动截断为 `value & 0xFFFF`）

✅ **核心用途**：实现对 Modbus 保持寄存器、FX2N D 寄存器、HMI 内部变量等 16 位数据区的写入。

2.2.2.get_uint16(vtype, addr)

16位无符号整型寄存器读取函数，`get_uint16(vtype, addr)` 是 HMI 系统提供的**16 位无符号整数读取函数**，用于向指定**地址**获取一个 **0～65535** 范围的数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：**如 `VT_4x`（Modbus RTU 保持寄存器）、`VT_D`（FX3U D寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
返回值	number	0 ～ 65535 的整数值（若通信失败，可能返回 0 或旧值）

> 🔑 `get_uint16(vtype, addr)` **是从 HMI 的本地缓存（内存镜像）中读取数据，而非实时向 PLC/设备发起串口请求。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_uint16(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

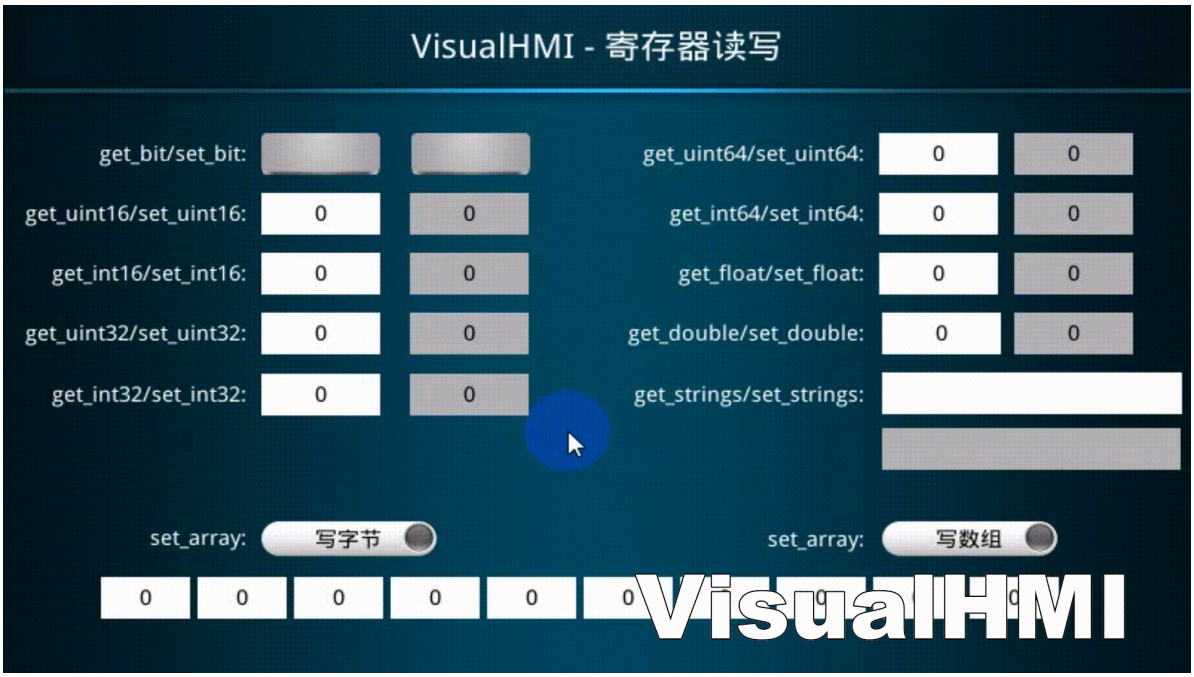
2.2.LUA脚本

点击数值控件0x1000(LW 内存变量)，触发on_update回调，读取内存变量0x1000，并设置内存变量0x1001。

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1000
        then
            set_uint16(vtype, addr+1, get_uint16(vtype, addr))
        end
    end
end
```

2.3.运行预览

运行虚拟屏，点击寄存器0x1000(LW 内存变量)数值，弹出键盘输入100，寄存器0x1001(LW 内存变量)实时变化，如下所示：



3. get_int16/set_int16

3.1. API说明

3.1.1.set_int16(vtype, addr, value)

16位有符号整型寄存器写入函数，`set_int16(vtype, addr, value)` 是 HMI 系统提供的**16 位有符号整数写入函数**，用于向指定**地址**写入一个**-32768 ~ +32767** 范围的整数值

📊 **参数说明**

参数	类型	说明
`vtype`	number	变量类型：**如** `VT_4x` **（Modbus RTU 保持寄存器）、** `VT_D` **（FX3U D寄存器）、** `VT_LW` **（内部变量）等**
`addr`	number	**寄存器地址**
`value`	number	**写入值：**取值范围**-32768 ~ +32767** （超出将自动截断为 `value & 0xFFFF`）

✅ **核心用途**：实现对支持有符号数的 Modbus 寄存器、PLC 数据区或 HMI 内部变量的**带符号整型写入**

3.1.2.get_int16(vtype, addr)

16位有符号整型寄存器读取函数，`get_int16(vtype, addr)` 是 HMI 系统提供的**16 位有符号整数读取函数**，用于从指定**地址**获取一个 **-32768 ~ +32767** 范围的数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：**如 `VT_4x`（Modbus RTU 保持寄存器）、`VT_D`（FX3U D寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
返回值	number	**-32768 ~ +32767** 的整数值（若通信失败，可能返回 0 或旧值）

> 🔑 `get_int16(vtype, addr)` **是从 HMI 的本地缓存（内存镜像）中读取数据，而非实时向 PLC/设备发起串口请求。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_int16(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定

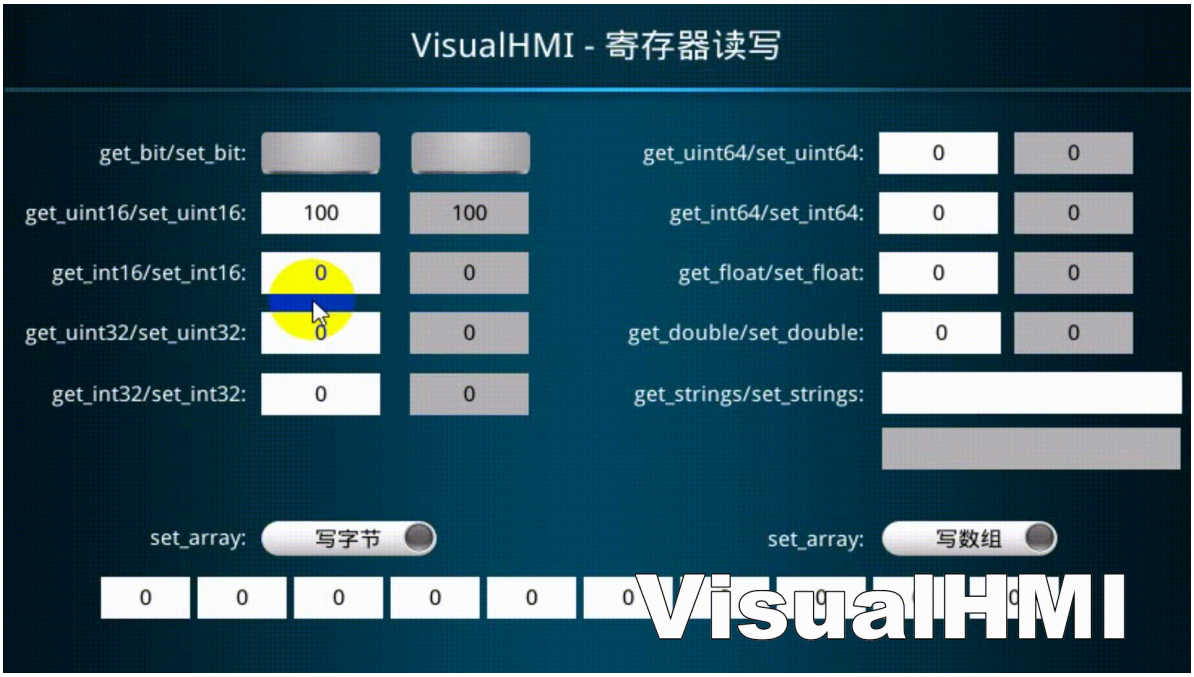
3.2.LUA脚本

点击数值控件0x1002(LW 内存变量)，触发on_update回调，读取内存变量0x1002，并设置内存变量0x1003。

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1002
        then
            set_int16(vtype, addr+1, get_int16(vtype, addr))
        end
    end
end
```

3.3.运行预览

运行虚拟屏，点击寄存器0x1002(LW 内存变量)数值，弹出键盘输入-100，寄存器0x1003(LW 内存变量)实时变化，如下所示：



4. get_uint32/set_uint32

4.1. API说明

4.1.1.set_uint32(vtype, addr, value)

32位无符号整型寄存器写入函数，`set_uint32(vtype, addr, value)` 是 HMI 系统提供的**32 位无符号整数写入函数**，用于向指定**地址**写入一个 **0 ~ 4,294,967,295** 范围内的整数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：** 如 `VT_4x`（Modbus RTU 保持寄存器）、`VT_D`（FX3U D寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
`value`	number	**写入值:**取值范围 0 ~ 4294967295（超出将自动截断为 `value & 0xFFFFFFFF`）

✅ **核心用途**：实现对 Modbus 保持寄存器、FX2N D 寄存器等区域的**32 位无符号整型数据写入。**

4.1.2.get_uint32(vtype, addr)

32位无符号整型寄存器读取函数，`get_uint32(vtype, addr)` 是HMI 系统提供的**32 位无符号整数读取函数**，用于从指定地址读取一个 **0 ~ 4,294,967,295** 范围的整数值

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_3x`（输入寄存器）、`VT_4x`（保持寄存器）、`VT_LW`（内部变量）等

参数	类型	说明
`addr`	number	**寄存器地址**
返回值	number	0 ~ 4294967295 的 32 位无符号整数（若通信失败，可能返回 0 或旧值）

> 🔑 `get_uint32()` **仅从 HMI 内存缓存中读取数据，不会向 PLC/设备发送任何串口指令。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_uint32(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

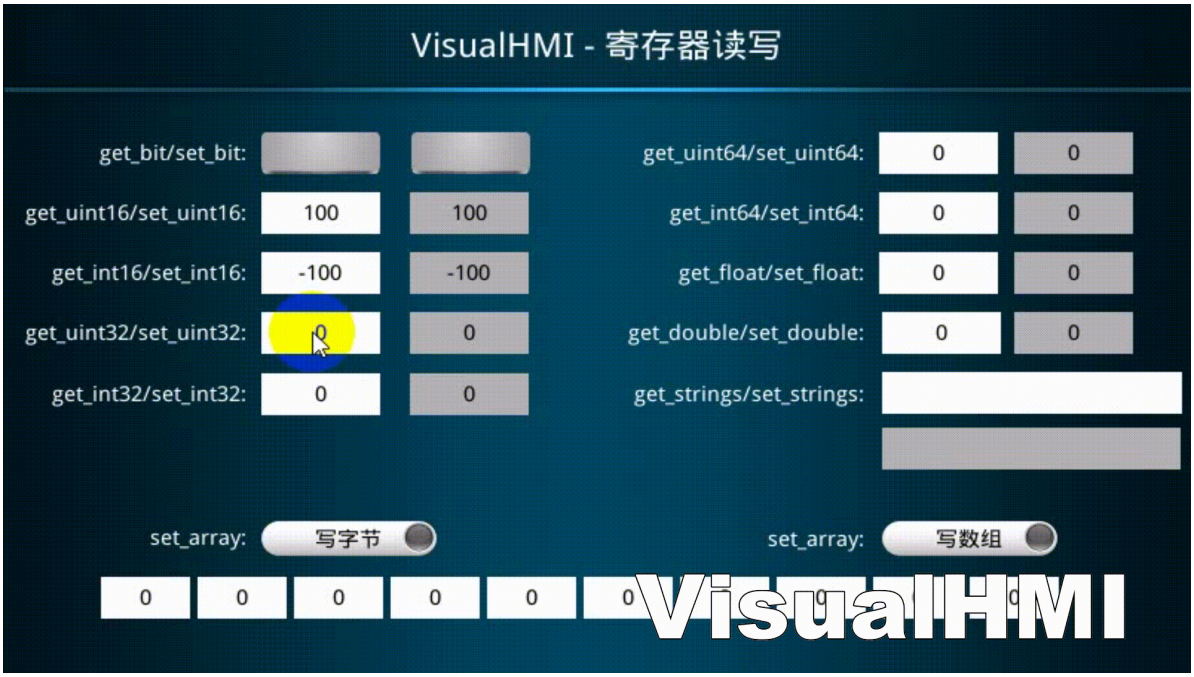
4.2.LUA脚本

点击数值控件0x1004(LW 内存变量)，触发on_update回调，读取内存变量0x1004，并设置内存变量0x1006

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1004
        then
            set_uint32(vtype, addr+2, get_uint32(vtype, addr))
        end
    end
end
```

4.3.运行预览

运行虚拟屏，点击寄存器0x1004(LW 内存变量)数值，弹出键盘输入70000，寄存器0x1006(LW 内存变量)实时变化，如下所示：



5. get_int32/set_int32

5.1. API说明

5.1.1.set_int32(vtype, addr, value)

32位有符号整型寄存器写入函数，`set_int32(vtype, addr, value)` 是 HMI 系统提供的**32 位有符号整数写入函数**，用于向指定地址的**双字地址**写入一个 ** **-2,147,483,648 ~ +2,147,483,647** ** 范围内的整数值。

📊 参数说明

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_4x`（保持寄存器）、`VT_D`（PLC 数据寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
`value`	number	写入值，取值范围 **-2,147,483,648 ~ +2,147,483,647** （超出将自动截断为 `value & 0xFFFFFFFF`）

✅ **核心用途**：实现对 Modbus 保持寄存器、FX2N D 寄存器等区域的**32 位有符号整型数据写入。**

5.1.2.get_int32(vtype, addr)

32位有符号整型寄存器读取函数，`get_int32(vtype, addr)` 是HMI 系统提供的**32 位有符号整数读取函数**，用于从指定地址读取一个 **-2,147,483,648 ~ +2,147,483,647** 范围内的整数值

📊 **参数说明**

参数	类型	说明
`vtype`	number	变量类型，如 `VT_3x`（输入寄存器）、`VT_4x`（保持寄存器）、`VT_LW`（内部变量）等

参数	类型	说明
`addr`	number	寄存器地址
返回值	number	**-2,147,483,648 ~ +2,147,483,647** 的 32 位有符号整数（若通信失败，可能返回 0 或旧值）

> 🔑 `get_int32()` **仅从 HMI 内存缓存中读取数据，不会向 PLC/设备发送任何串口指令。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_int32(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

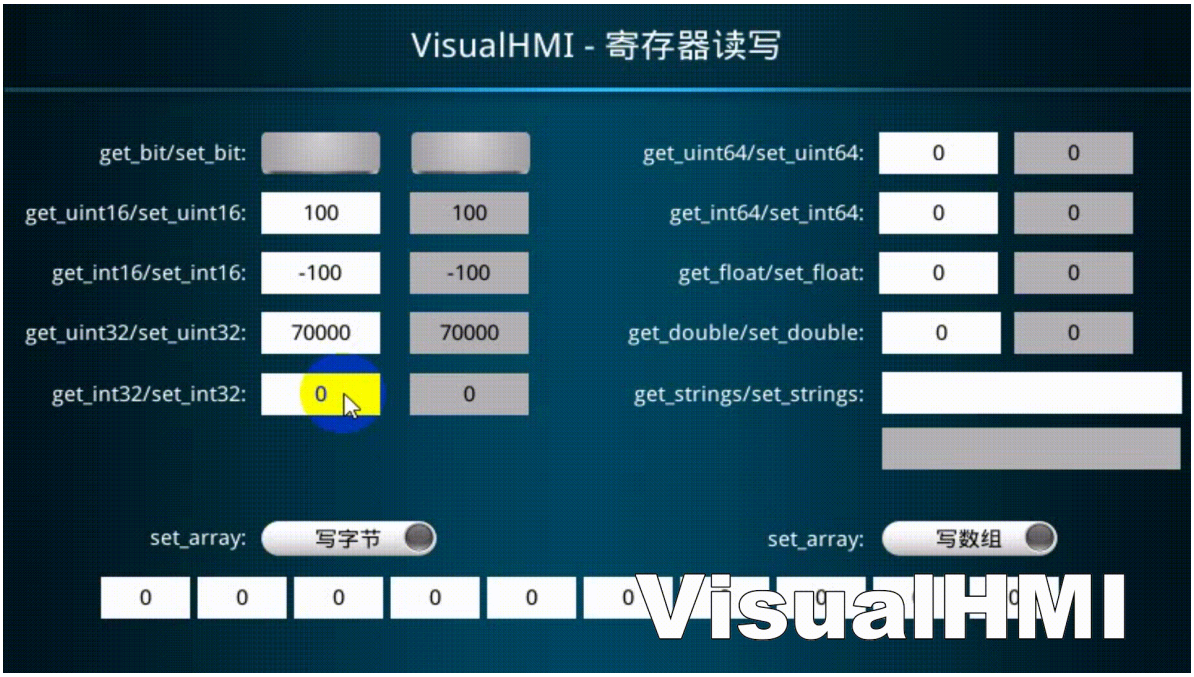
5.2.LUA脚本

点击数值控件0x1008(LW 内存变量)，触发on_update回调，读取内存变量0x1008，并设置内存变量0x100A

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1008
        then
            set_int32(vtype, addr+2, get_int32(vtype, addr))
        end
    end
end
```

5.3.运行预览

运行虚拟屏，点击寄存器0x1008(LW 内存变量)数值，弹出键盘输入-70000，寄存器0x100A(LW 内存变量)实时变化，如下所示：



6. set_uint64/get_uint64

6.1. API说明

6.1.1.set_uint64(vtype, addr, value)

64位无符号整型寄存器写入函数，`set_uint64(vtype, addr, value)` 是 HMI 系统提供的**64 位无符号整数写入函数**，用于**地址**写入一个 **0 ~ $2^{64}-1$** 范围内的整数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型*:**如 `VT_4x`（保持寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
`value`	number	写入值，取值范围**0 ~ $2^{64}-1$**

✅ **核心用途**：实现对支持 64 位数据的设备或 HMI 内部变量的**超大无符号整型写入**。

6.1.2.get_uint64 (vtype, addr)

64位有符号整型寄存器读取函数，`get_uint64(vtype, addr)` 是HMI 系统提供的**64 位有符号整数读取函数**，用于从指定地址读取一个**0 ~ $2^{64}-1$** 范围内的整数值。

📊 参数说明

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_3x`（输入寄存器）、`VT_4x`（保持寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
返回值	number	**0 ~ $2^{64}-1$** 的 64 位有符号整数，（若通信失败，可能返回 0 或旧值）

> 🔑 `get_uint64()` **仅从 HMI 内存缓存中读取数据，不会向 PLC/设备发送任何串口指令。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_uint64(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

6.2.LUA脚本

点击数值控件0x100C(LW 内存变量)，触发on_update回调，读取内存变量0x100C，并设置内存变量0x1010

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1000
        then
            set_uint64(vtype, addr+4, get_uint64(vtype, addr))
        end
    end
end
```

7. set_int64/get_int64

7.1. API说明

7.1.1.set_int64(vtype, addr, value)

64位有符号整型寄存器写入函数，`set_int64(vtype, addr, value)` 是 HMI 系统提供的**64 位有符号整数写入函数**，用于**地址**写入一个 **-2^{63} ~ $2^{63}-1$**范围的整数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：**如 `VT_4x`（保持寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
`value`	number	写入值，取值范围 **-2^{63} ~ $2^{63}-1$**

✅ **核心用途**：实现对支持 64 位数据的设备或 HMI 内部变量的**超大无符号整型写入。**

7.1.2.get_int64 (vtype, addr)

64位有符号整型寄存器读取函数，`get_int64(vtype, addr)`是HMI 系统提供的**64 位有符号整数读取函数**，用于从指定地址读取一个 **-2^{63} ~ $2^{63}-1$** 范围内的整数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_3x`（输入寄存器）、`VT_4x`（保持寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
返回值	number	**-2^{63} ~ $2^{63}-1$**的 64 位无符号整数，（若通信失败，可能返回 0 或旧值）

> 🔑 `get_int64()` **仅从 HMI 内存缓存中读取数据，不会向 PLC/设备发送任何串口指令。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_int64(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

7.2.LUA脚本

点击数值控件0x1014(LW 内存变量)，触发on_update回调，读取内存变量0x1014，并设置内存变量0x1018

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1014
        then
            set_int64(vtype, addr+4, get_int64(vtype, addr))
        end
    end
end
```

8. get_float/set_float

8.1. API说明

8.1.1.set_float(vtype, addr, value)

单精度浮点数寄存器写入函数，`set_float(vtype, addr, value)` 是 HMI 系统提供的**单精度浮点数（IEEE 754 32 位）写入函数**，用于向指定**地址**写入实数值。

📊 **参数说明**

参数	类型	说明

参数	类型	说明
`vtype`	number	**变量类型：**如 `VT_4x`（保持寄存器）、`VT_D`（PLC 数据寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
`value`	number	浮点写入值，如 `1.234`、`-45.6`、`3.14159e5`

✅ **核心用途**：实现对 Modbus 保持寄存器、PLC 数据区或 HMI 内部变量的**浮点型参数设置与指令下发。**

8.1.2.get_float (vtype, addr)

单精度浮点数寄存器读取函数，`get_float(vtype, addr)` 是 HMI 系统提供的**单精度浮点数（IEEE 754 32 位）读取函数**，用于从指定**地址**读取一个带小数的实数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_3x`（输入寄存器）、`VT_4x`（保持寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
返回值	number	IEEE 754 单精度浮点数

> 🔑 `get_float` **仅从 HMI 内存缓存中读取数据，不会向 PLC/设备发送任何串口指令。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_float(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定

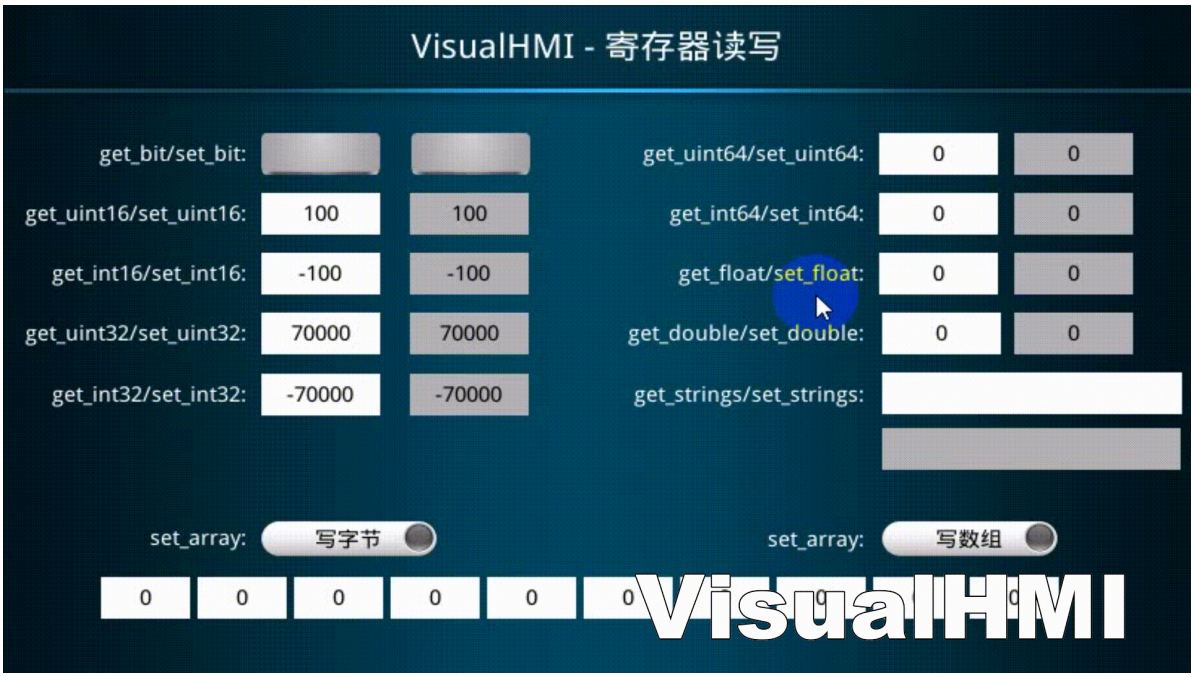
8.2.LUA脚本

点击数值控件0x101C(LW 内存变量)，触发on_update回调，读取内存变量0x101C，并设置内存变量0x101E

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x101C
        then
            set_float(vtype, addr+2, get_float(vtype, addr))
        end
    end
end
```

8.3.运行预览

运行虚拟屏，点击寄存器0x101C(LW 内存变量)数值，弹出键盘输入-1.23，寄存器0x1020(LW 内存变量)实时变化，如下所示：



9. get_double/set_double

9.1. API说明

9.1.1.set_double(vtype, addr, value)

双精度浮点数寄存器写入函数，`set_double(vtype, addr, value)` 是 HMI 系统提供的**双精度浮点数（IEEE 754 64 位）写入函数**，用于向指定**地址**写入一个高精度实数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_LW`（内部变量）、`VT_RW`（掉电保存）等
`addr`	number	**寄存器地址**
`value`	number	双精度浮点值，如 `3.141592653589793`、`-1.23e-10`

✅ **核心用途**：实现对 Modbus 保持寄存器、PLC 数据区或 HMI 内部变量的**双精度型参数设置与指令下发。**

9.1.2.get_double (vtype, addr)

双精度浮点数寄存器读取函数，`get_double(vtype, addr)` 是 HMI 系统提供的**双精度浮点数（IEEE 754 64 位）读取函数**，用于从指定**地址**读取一个高精度实数值。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_LW`（内部变量）、`VT_RW`（掉电保存）等
`addr`	number	**寄存器地址**
返回值	number	IEEE 754 双精度浮点数（Lua 原生 number 类型）

> 🔑 `get_double` **仅从 HMI 内存缓存中读取数据，不会向 PLC/设备发送任何串口指令。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_double (VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的** `get_xxx()` **是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

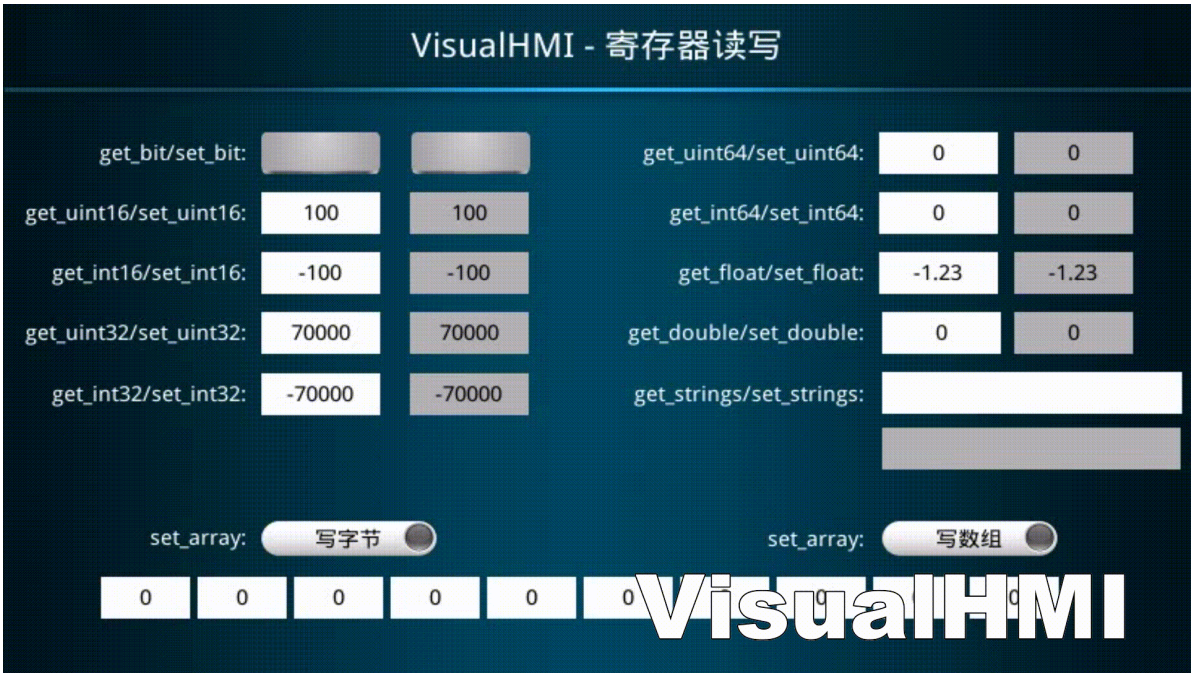
9.2.LUA脚本

点击数值控件0x1020(LW 内存变量)，触发on_update回调，读取内存变量0x1020，并设置内存变量0x1024

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1020
        then
            set_double(vtype, addr+4, get_double(vtype, addr))
        end
    end
end
```

9.3.运行预览

运行虚拟屏，点击寄存器0x1020(LW 内存变量)数值，弹出键盘输入-12345.6789，寄存器0x1024(LW 内存变量)实时变化，如下所示：



10. get_string/set_string

10.1. API说明

10.1.2.set_string(vtype, addr, strings)

字符串寄存器写入函数，`set_string(vtype, addr, str)` 是 HMI 系统提供的**字符串写入函数**，用于向指定**地址序列**写入一个**'UTF-8' 或'GBK' 编码**的字符串。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：**如 `VT_LW`（内部变量）、`VT_4x`（保持寄存器）等
`addr`	number	**寄存器地址**
`str`	string	要写入的字符串，如 `"https://www.gz-dc.com/"`，最大2K字节

✅ **核心用途**：实现对 HMI 内部变量或支持字符串的设备寄存器的**文本数据写入。**

10.1.3.get_string(vtype, addr，len)

字符串变量读取函数，`get_string(vtype, addr, len)` 是 HMI 系统提供的**字符串读取函数**，用于从指定存储区的**地址**中读取一段以空字符 `\0` 结尾的文本数据

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型：**如 `VT_LW`（内部变量）、`VT_RW`（掉电保存）等
`addr`	number	**寄存器地址**
`len`	number (可选)	**最大读取字节数:**默认 128 字节，最大支持 2048 字节（2KB）

参数	类型	说明
返回值	string	得到的字符串（自动在 `\0` 处截断）

> 🔑 `get_string` **仅从 HMI 内存缓存中读取数据，不会向 PLC/设备发送任何串口指令。**

🔴 **工作机制：**

1. **HMI 后台任务** 按照工程中配置的**轮询周期**，自动从 PLC 读取 `VT_3x`、`VT_4x` 等通信变量；
2. 读取到的数据被**缓存在 HMI 内存**中，形成“寄存器镜像”；
3. 脚本调用 `get_string(VT_3x, 0x1000)` 时，**直接返回本地缓存值**，**不产生新的串口帧**；
4. 因此，**脚本层的 `get_xxx()` 是零通信开销的。**

> ✅ **优势**：避免脚本逻辑导致通信风暴，保证系统实时性与稳定性

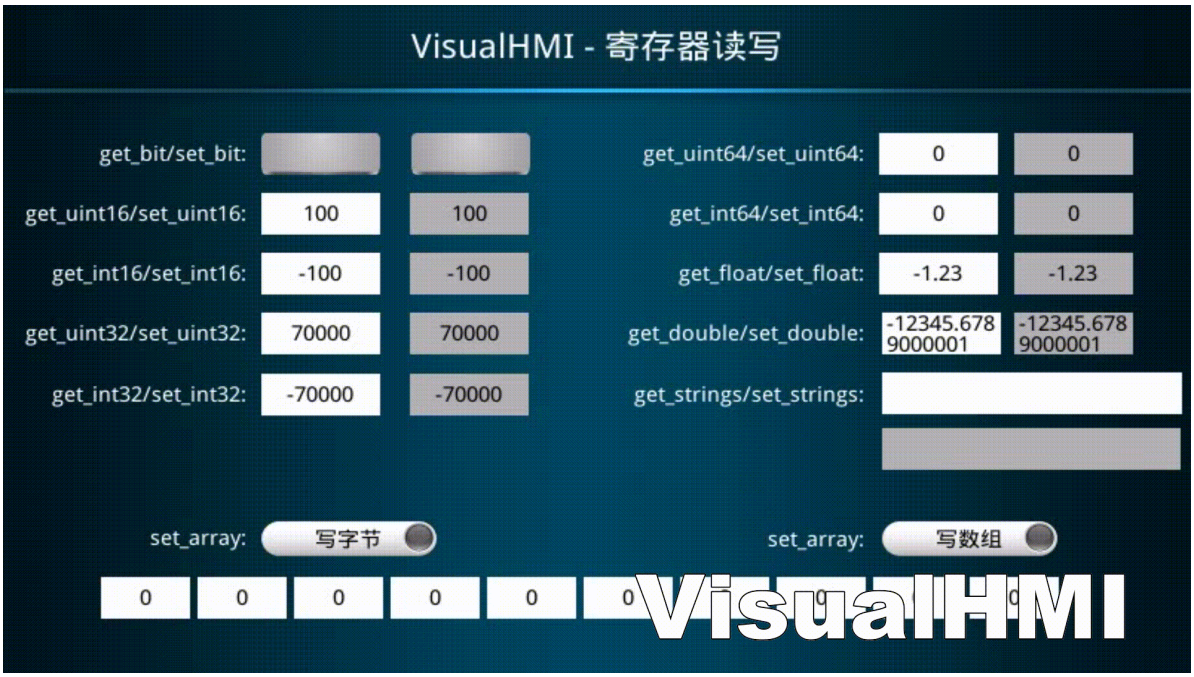
10.2.LUA脚本

点击数值控件0x101C(LW 内存变量)，触发on_update回调，读取内存变量0x101C，并设置内存变量0x1020

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1030
        then
            set_string(vtype, addr+16, get_string(vtype, addr))
        end
    end
end
```

10.3.运行预览

运行虚拟屏，点击寄存器0x1030(LW 内存变量)数值，弹出键盘输入"大彩abc123“，寄存器0x1040(LW 内存变量)实时变化，如下所示：



11. set_uint16_ex

11.1. API说明

11.1.1.set_uint16_ex(vtype, addr, value1,value2, ..., value120)

批量无符号16位寄存器写入函数,`set_uint16_ex(vtype, addr, value1, value2, ..., valueN)` 是 HMI 系统提供的**批量 16 位无符号整数写入函数**，用于**一次性向连续多个寄存器**（最多 120 个）。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_4x`（保持寄存器）、`VT_D`（PLC 数据寄存器）、`VT_LW`（内部变量）等
`addr`	number	**寄存器地址**
`value1` ~ `valueN`	number	1 到 120 个无符号 16 位整数值，每个取值范围 0 ~ 65535
最大数量	—	120 个寄存器

✅ **核心用途**：高效批量设置设备参数。

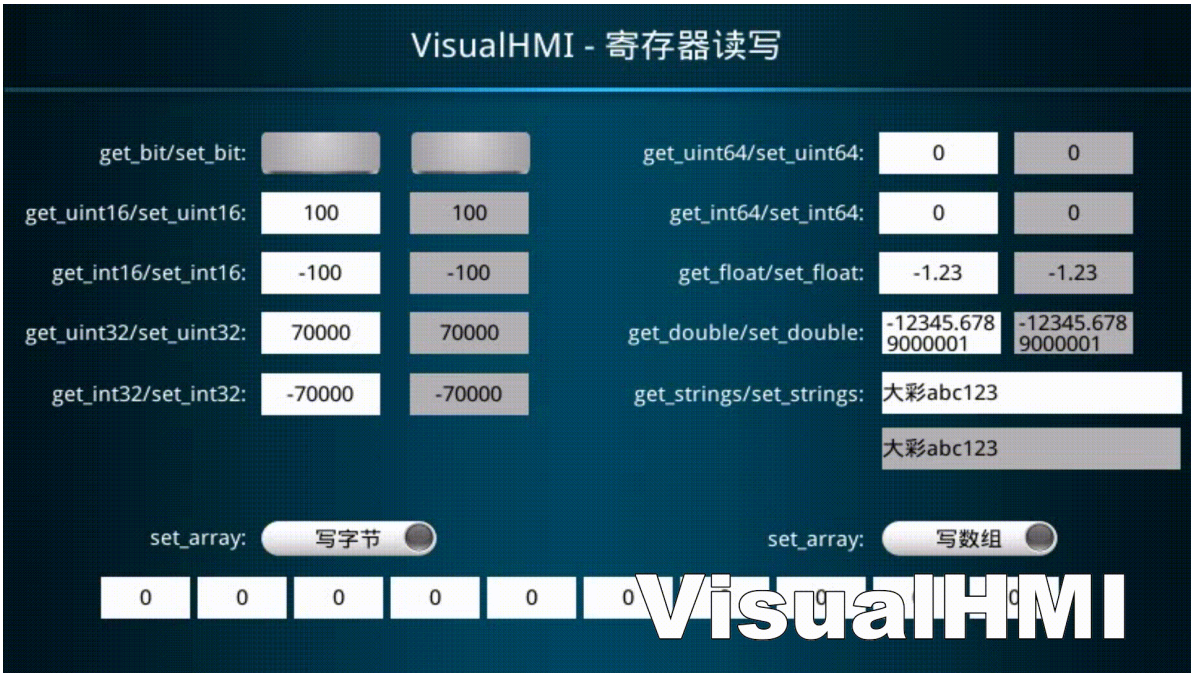
11.2.LUA脚本

点击数值控件0x1050(LW 内存变量)，触发on_update回调，批量设置0x1060~0x1069(LW 内存寄变量)存器的值

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1050 and get_uint16(vtype, addr) % 2 == 0
        then
             set_uint16_ex(vtype, 0x1060,1,2,3,4,5,6,7,8,9,10)
        end
    end
end
```

11.3.运行预览

运行虚拟屏，点击寄存器0x1050(LW 内存变量)数值，批量设置0x1060~0x1069(LW 内存寄变量)的值依次为1~10,如下所示：



12. set_array

12.1. API说明

12.1.1.set_array(vtype, addr, buff)

批量寄存器写入函数,`set_array(vtype, addr, buff)` 是 HMI 系统提供的**高效批量寄存器写入函数**，用于将一个 **16 位字（word）数组**一次性写入指定起始地址的连续寄存器区域。

与 `set_uint16_ex` 不同，`set_array` 接收一个 **Lua 表（table）作为缓冲区**，而非可变参数列表，更适合程序化生成和动态数据处理。

📊 **参数说明**

参数	类型	说明
`vtype`	number	**变量类型:**如 `VT_4x`（保持寄存器）、`VT_D`、`VT_LW` 等
`addr`	number	**寄存器地址**
`buff`	table	**字（word）数组:**每个元素为 0 ~ 65535 的整数
最大长度	—	120 个字（即最多写入 120 个连续 16 位寄存器）

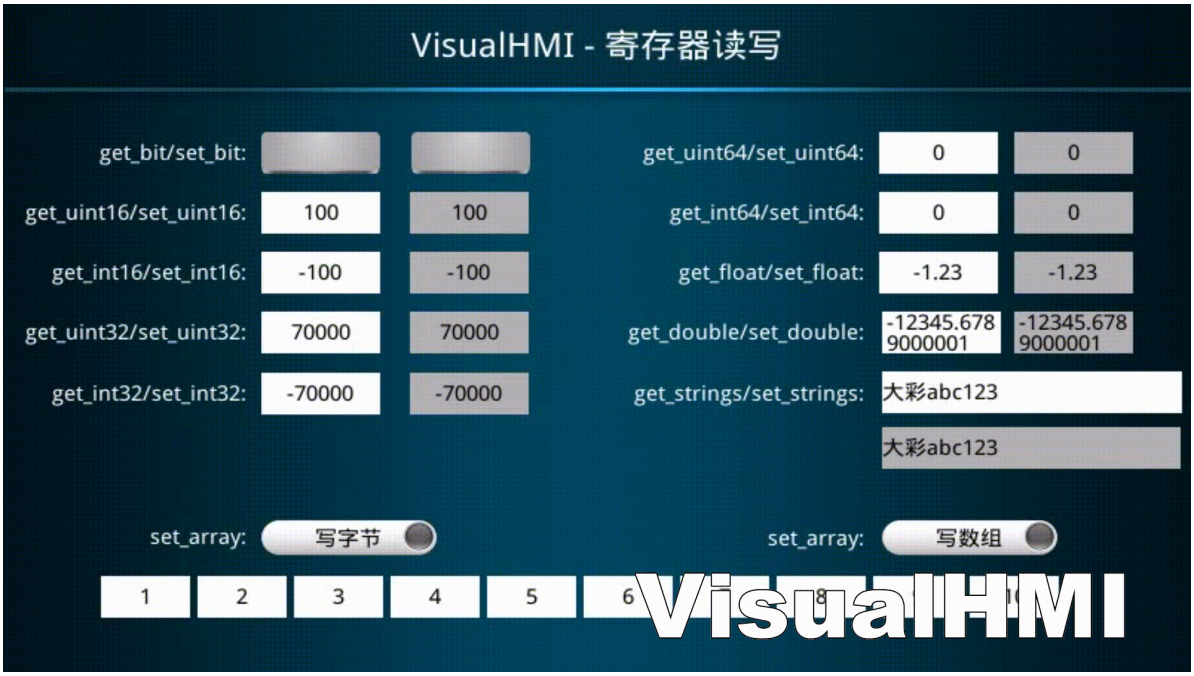
✅ **核心用途**：高效写入结构化数据块、通信协议帧、设备配置数组、HMI 内部状态缓存等。

12.2.LUA脚本

点击数值控件0x1051(LW 内存变量)，触发on_update回调，批量设置0x1060~0x1069(LW 内存寄变量)。

```
function on_update(slave,vtype,addr)
    if vtype == VT_LW
    then
        if addr == 0x1050 and get_uint16(vtype, addr) % 2 == 0
```

```
        then
            local ex_buff = {}
            for i = 1, 10
            do
                ex_buff[i] = i*10
            end
            set_array(vtype, 0x1060, ex_buff)
        end
    end
end
```

12.3.运行预览

运行虚拟屏，点击寄存器0x1050(LW 内存变量)数值，批量设置0x1060~0x1069(LW 内存寄变量)值为依次为10,20,...,90,100