使用说明及测试指导

HIS 脚本

Rev1.0

版本历史：

2017-06-19 初稿

2017-08-24 增加了 flash 参数说明及使用案例

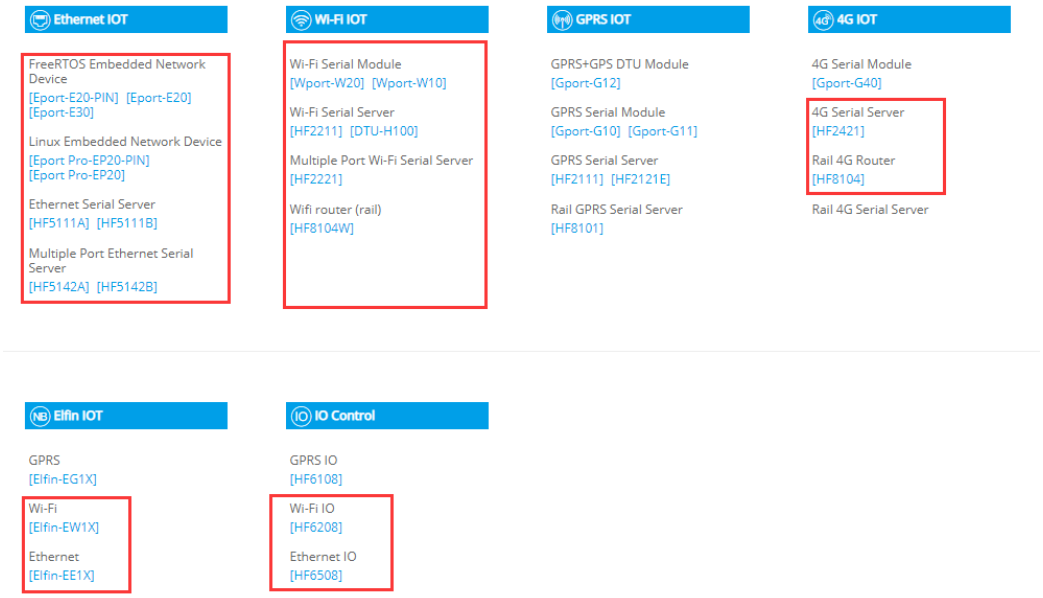
2018-11-14 增加更多例子

1. 脚本介绍

汉枫 I.O.T 脚本（HF IOT Script，缩写 HIS。用于汉枫 IOT 工控设备的数据转换，把脚本下载到工控设备后，可自动完成下列功能：

- 自动定时发数据给 UART 或 Socket，作为 Modbus 主站功能。
- 从 UART 或 Socket 收到数据后，按脚本转换后转发

目前支持的产品列表如下红色类型，下文以 Eport-E10 或者 HF2211 等产品为例介绍。



2. 语法介绍

2.1. 数据类型

HIS 脚本支持三种数据类型：整数、浮点数、字符串。

变量使用时，自动定义和赋值。如：

a=10

定义一个整数 a 并赋值为 10

f=1.04

定义一个浮点数 f 并赋值为 1.04

s=”1234”　　或　　s=[0x31,0x32,0x33,0x34]

定义一个字符串 s 并赋值

注：HIS 语法是大小写敏感的。

2.2. 运算符

HIS 支持运算符及运算符的优先级如下：

优先级从高到低	运算符
0	(,) , [,] , .
1	* , /
2	+ , -
3	>> , << , & , |
4	< , <= , > , >= , == , !=
5	&& , ||
6	=

注：脚本不支持 ++, --, +=, -=等操作

2.3. 数据运算

整数和浮点数支持“+,-,*,/”运算，整数支持位操作，如：<<,>>,&,|。

比较运算时，源操作数和目的操作数的数据类型必须同类（整数和浮点数视为同类，字符串只能和字符串进行比较）。

赋值操作时，同时会涉及到数据类型的转换。如：

a 为整数，f 为浮点数，s 为字符串

a=f　　把 f 赋值给 a，同时，a 的类型自动转为浮点数（如果: f=a，不涉及类型转换）

a=s　　a 会转为字符串，并赋值为 s 的内容

s=f　　s 转为浮点数，并赋值

字符串的加法运算：

如：s1=”123”，s2=”456”

s3=s1+s2　　结果 s3=”123456”

字符串比较运算：字符串只支持”==”和”!=”运算

字符串函数，列举如下：

如：s1=”12345678”，a1=0x010203

a=s1.length()　　结果 a 为整数 8

b=s1.charAt(2)　　结果 b 为整数 2，参数为第几个字符

c=s1.subString(1,5)　结果 c 为字符串,”2345”

d=s1.replaceAt(1,’5’)　结果 d 为字符串”15345678”

注：subString()函数，前一个参数为开始位置，后一个为结束位置，位置从零开始计数。字串中包括开始位置的数据，但不包括结束位置的数据。

d=a1.toString(0,3)　　结果 d 为字符串：[0x03,0x02,0x01,0x00]

e=a1.toString(2,0)　　结果 e 为字符串：[0x01,0x02,0x03]

f=a1.prtString()　　结果 f 为字符串：” 66051”，(0x010203 等于十进制的 66051)

g=s1.stringHex()　　结果 g 为字符串：[0x12,0x34,0x56,0x78]

h=s2.hexString()　　结果 h 为字符串：” 12345678”

i=s1.indexOf("234")　结果 i 为整数：1

注：toString ()函数的源操作数为整数，2 个参数分别为整数中 4 个字节的位置（所以参数的取值为>=0，<=3）。例子 1，d 为从 0 字节到 3 字节，例子 2，e 为从 2 字节到 0 字节

2.4. IF 语句

例如：

```
IF (a>1)
    ……
ELSE
    ……
END
```

可以没有 ELSE，如：

```
IF (a>1)
    ……
END
```

2.5. FOR 语句

例如：

```
FOR i,0,5,1   // i 为变量，后面 3 个参数依次是：初始值，结束值和步进值
    k=i
END
```

注：初始值，结束值和步进值都必须为整数，当变量等于结束值时结束循环，步进值可以是正数和负数

2.6. WHILE 语句

例如：

```
i=0
WHILE (i<10)
    i=i+1                不支持++,--,+=,-=等操作
END
```

2.7. 保留字

HIS 的保留字都为大写的字符，列举如下：

"TIMER","END","MAC","SOCKBEAT","SYSTIME","RECV","CONN","DISCONN","SOCK","UART","SEND","WAIT","IF","ELSE","FOR","WHILE","INT","FLOAT","INPUT","FUNCTION","RETURN","TRUE","FALSE","STRING","CALCRC16","RTTIME"，" RESET"

2.8. 外部接口

外部接口为脚本与工控设备交互的接口，分为 2 部分，分别为从访问工控设备部分和从工控设备输入部分。

2.9. 访问工控设备部分

保留字 MAC，为读取工控设备的 MAC 地址，如：

```
工控设备的 MAC 地址为" ACCF23001234"
mac= MAC              定义变量 mac 并赋值为" ACCF23001234"
```

SYSTIME 为读取系统时间的秒数，如：

```
t= SYSTIME
```

SOCKBEAT 为读取某个 Socket 的心跳字符串，如：

```
bt= SOCKBEAT(netp)      读取名为 netp 的 Socket 的心跳字符串
```

SEND 向工控设备某接口发送数据，如：

```
SEND(UART,uart0," 1234" )
SEND(SOCK,netp," 3456" )
```

注：第一个参数为工控设备接口类型，第二个为工控设备接口名称，第三个为数据内容。

读写设备 GPIO，如：

```
P10= GETHW(GPIO, 10)        // 读取 GPIO Pin10 的值
SETHW(GPIO, OUT, 10, 1)          // 设置 GPIO Pin10 为输出高
SETHW(GPIO, IN, 10)          // 设置 GPIO Pin10 为输入
```

Close Socket，如：

```
DISCONN(SOCK, netp)              // netp 断开连接
```

保留字 WANIP，为读取工控设备的 WAN 口 IP 地址，如：

工控设备的 WAN 口 IP 地址为”192.168.11.100”

ip= WANIP 定义变量 ip 并赋值为”192.168.11.100”

保留字 CALCRC16，计算数据串的 CRC16 值，如：

```
str=[0x01,0x03,0x10,0x01,0x00,0x03]

crc=CALCRC16(str)              //crc 等于 str 的 CRC16 值，crc=0x50cb
```

RTTIME 为读取实时时间的秒数，如：

```
t= RTTIME                              //t=1505972386，2017/9/21 13:39:46
```

RETEN //重启指令。

2.10. 从工控设备输入部分

从工控设备输入部分接口以程序块表示。列举如下：

- 定时接口

```
TIMER HeartBeat 1000
    SEND(UART,uart0,”1234”)
END
```

定时接口名称为“HeartBeat”，定时 1000ms，END 为程序块结束标记。

- 接收接口

```
RECV SOCK netp
    len=INPUT.length()
    IF (len>10)
        tmp=INPUT.subString(3,10)
    ELSE
        tmp=”ERR”
    END
    SEND(UART,uart0,tmp)
    RETURN(FALSE)
END
```

注：Socket netp 接收数据是自动调用，INPUT 为输入的参数，RETURN(FALSE)说明本函数结束时不需要按原来的 ROUT 定义转发（通常本函数已调用的 SEND 函数发送）。RETURN(TRUE)时，需要按 ROUT 定义转发，转发内容为 OUTPUT 的字符串。

- RECV 接口为串口接收时：

```
RECV UART uart0
    ……
END
```


- Socket 连接接口：

```
CONN SOCK netp
    ……
END
```

当 Socket netp 建立连接时，自动调用

- Socket 断开连接接口：

```
DISCONN SOCK netp
    ……
END
```

当 Socket netp 断开连接时，自动调用

2.11. 函数

例如：

```
FUNCTION func1 (arg1, arg2)
    tmp1= arg1
    tmp2= arg2
    RETURN(tmp1)
END
```

函数调用：

```
TIMER xxx 1000
    ……
    tmp= func1(“1234”, “abcd”)
    ……
END
```

2.12. 变量作用域

所有变量分为全局变量和局部变量，全局变量为程序块外定义的变量，全局变量在工控设备运行期间一直有效，可以被其他程序块访问，也可以被工控设备外部读取，如网页等。

局部变量以程序的外部接口运行为单位，一个外部接口运行期间，产生的中间变量都为局部变量，同名的变量视为同一个变量。如：

```
a=""
b=”1234”
TIMER HB 1000
    j=10
    ret= func(a);          #这个访问不会出错，TIMER 内的 j 可以被 func 访问
END

CONN SOCK netp
```



```
    ret=func(b)          #这个访问会出错，因为 func 内的 j 没有定义
END

FUNCTION func (arg)
    b=arg
    c=a
    IF (j==2)
        b=”abcd”
    END
END
```

上面程序中，a,b 为全局变量，arg 为局部变量，j 为局部变量，函数 func 被 TIMER 调用，此时 func 的局部变量与 TIMER 一致，所以可以调用的 j（不推荐用这种写法，因为其他的外部接口访问 func 时可能出错）。一个外部接口程序块运行结束时，会释放全部局部变所有上面程序中，两个 ret 变量为不同的变量。

2.13. Flash 参数说明

如果同一个脚本，但是针对不同的模块有部分参数需要不同，这种情况下可以使用 Flash 参数。即对于这些参数脚本在初始化时会写入 Flash 中，通过 IOTService 工具可以对这些参数进行修改，修改成功后，脚本会优先使用这些修改后的值。从而实现不同模块使用同一个脚本文件，但这些参数具有不同的初始值。

Flash 参数在脚本中定义为常量，不允许修改，即脚本中对这些参数的赋值操作时，会报错。

Flash 参数的定义：

```
FLASH(NUM)          iFls=10
FLASH(STRSTR)  ssFls=  “123456”              // 在工具中显示为 String
FLASH(STRHEX)  shFls=[0x01,0x02,0x03]        // 在工具中显示为 16 进制
FLASH(BOOL)         bFls=1                   // 取值为 0 或 1
```

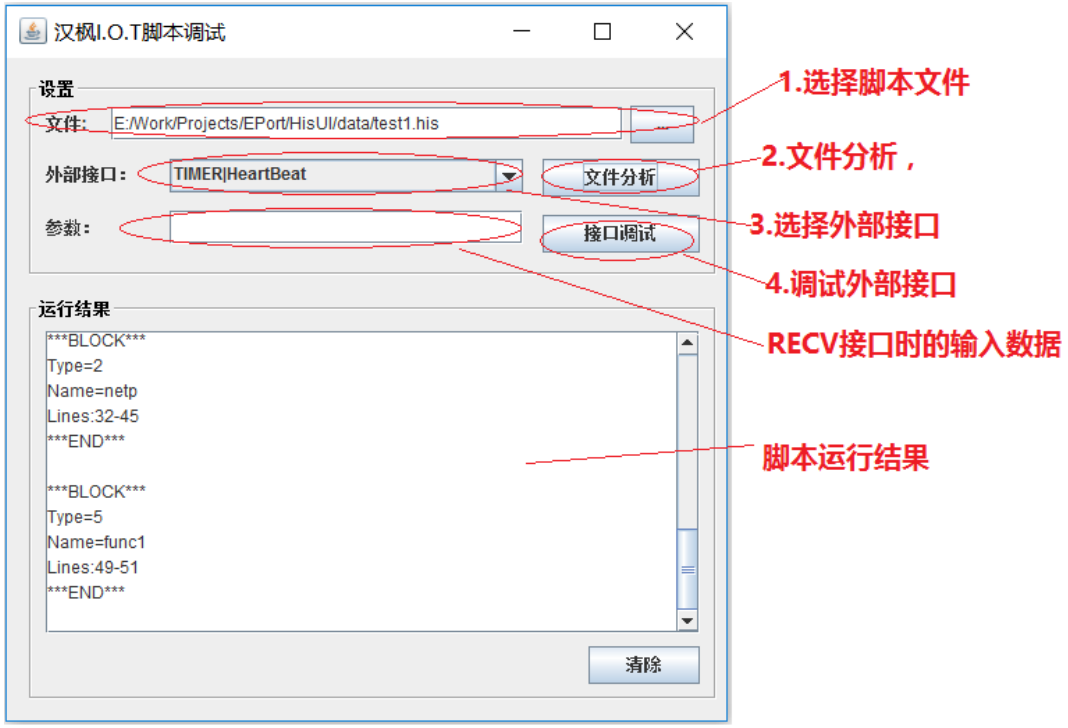
FLASHMAGIC = 200 //如果脚本的 FLASHMAGIC 和内部保存的默认值相同，则后续的 FLASH 参数不会做初始化的动作，如果不一致，则执行 FLASH 参数初始化动作且更新内部保存的默认值，比如

FLASHMAGIC = 200

FLASH(NUM) iFls=10 //如果内部保存的 FLASHMAGIC 是 200，则不会执行 iFls=10 的动作，

3. HISUI 说明

HisUI 为 HIS 脚本的可视化调试工具，可用于方便的调试脚本的功能，运行 His_Check_Tools 目录下的“HisUI.vbs”后加载文件进行测试。



注：本工具及后文介绍工具运行需要 JRE 运行环境，运行如下的指令确认 JRE 环境。

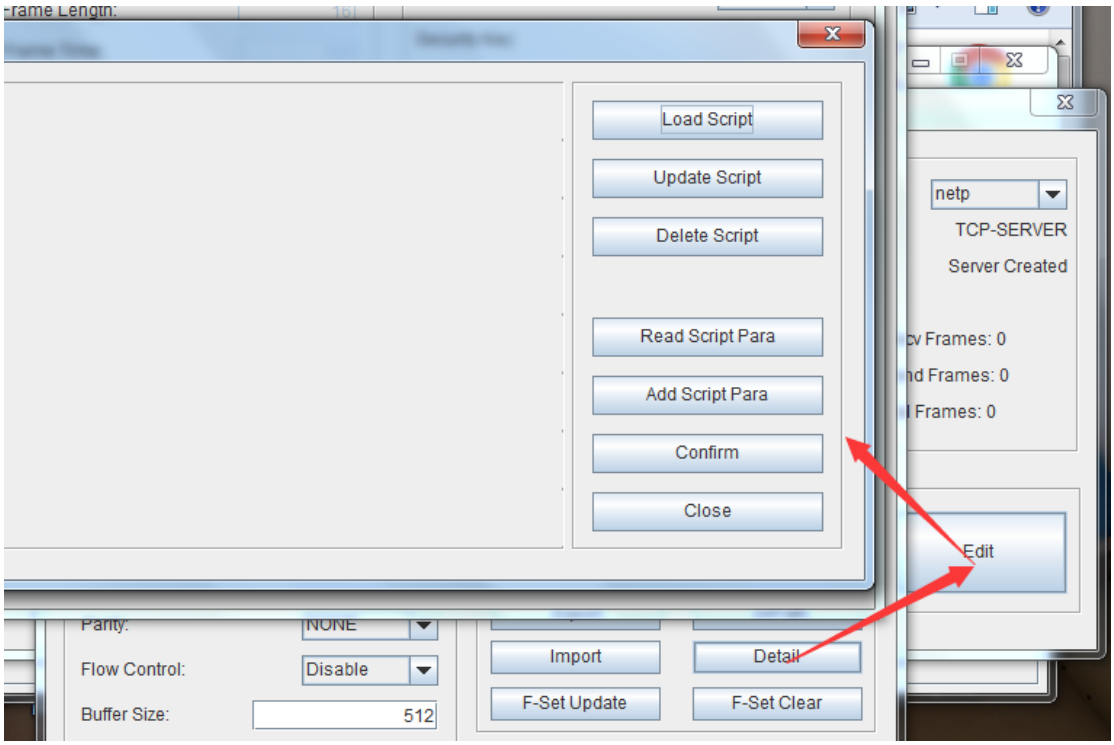
```
C:\Users\Sam>java -version
java version "1.8.0_91"
Java(TM) SE Runtime Environment (build 1.8.0_91-b15)
Java HotSpot(TM) Client VM (build 25.91-b15, mixed mode, sharing)
```

如未安装环境，可从如下链接中安装 JRE 环境。

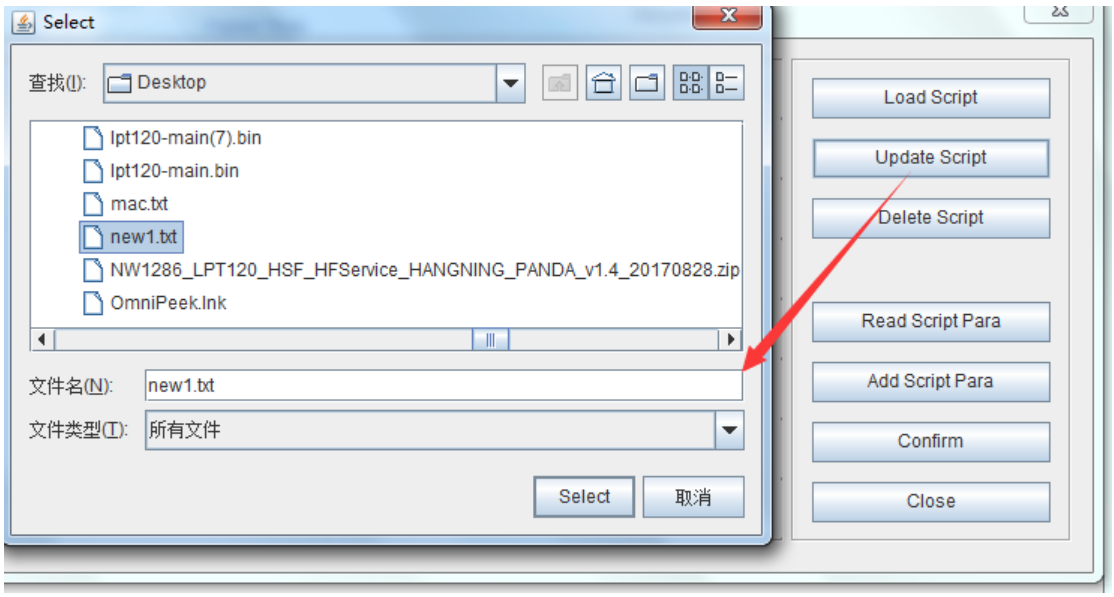
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

4. IOTSERVICE 工具升级脚本方法

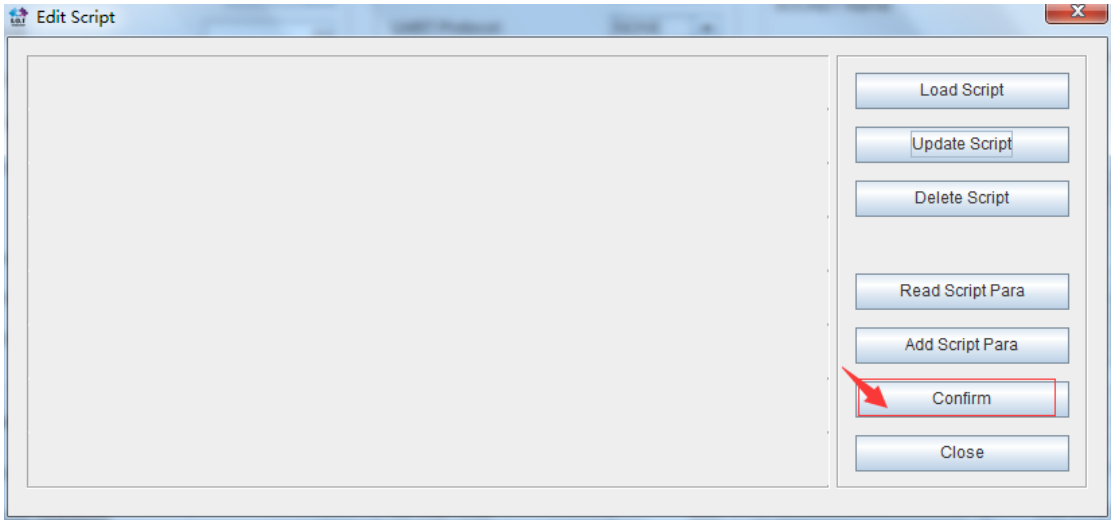
1、进入模块编辑界面，点击“高级设置”>“编辑脚本”。



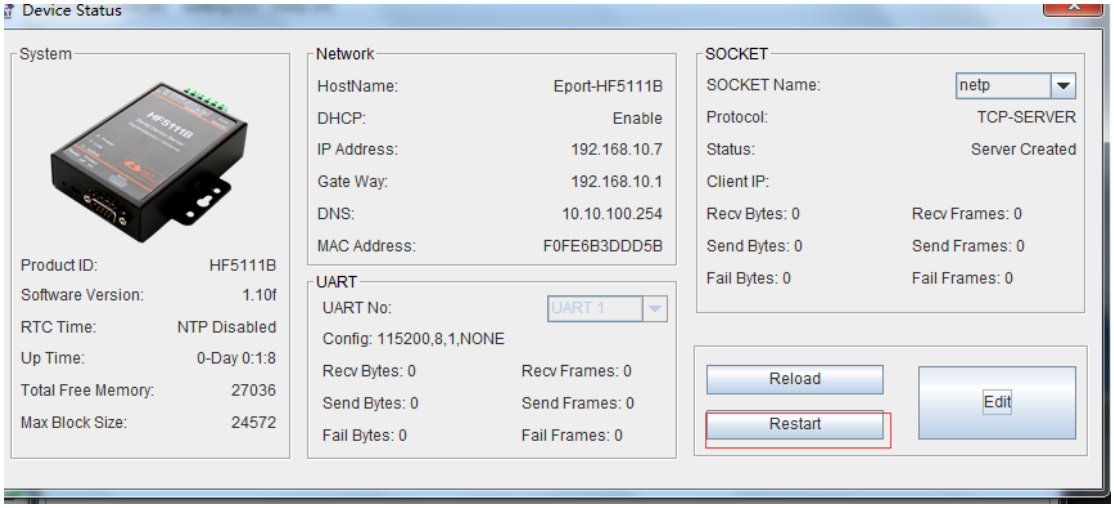
2、点击“更新脚本”进入选中导入的脚本文件 new1.txt。



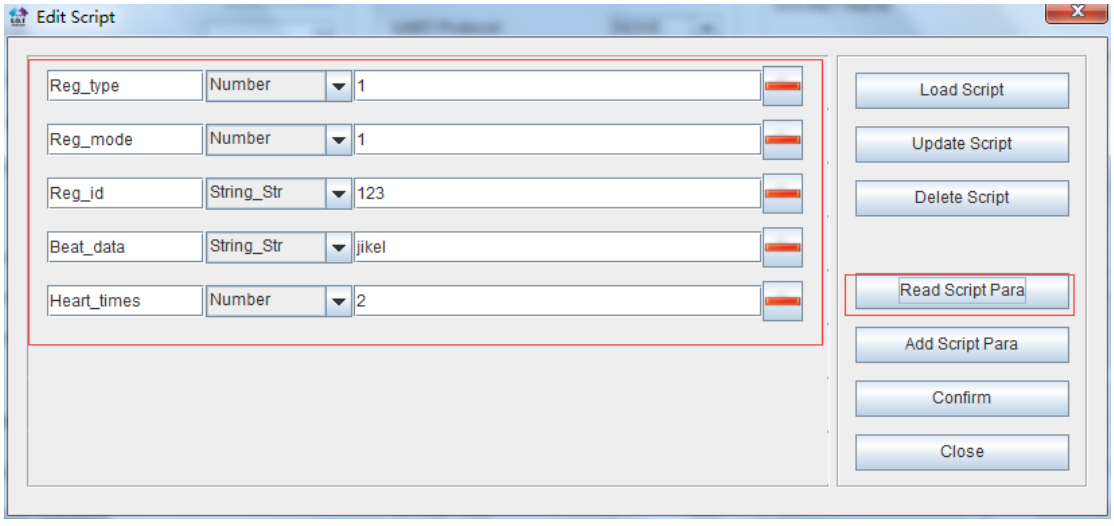
3、导入之后点击“确认”。



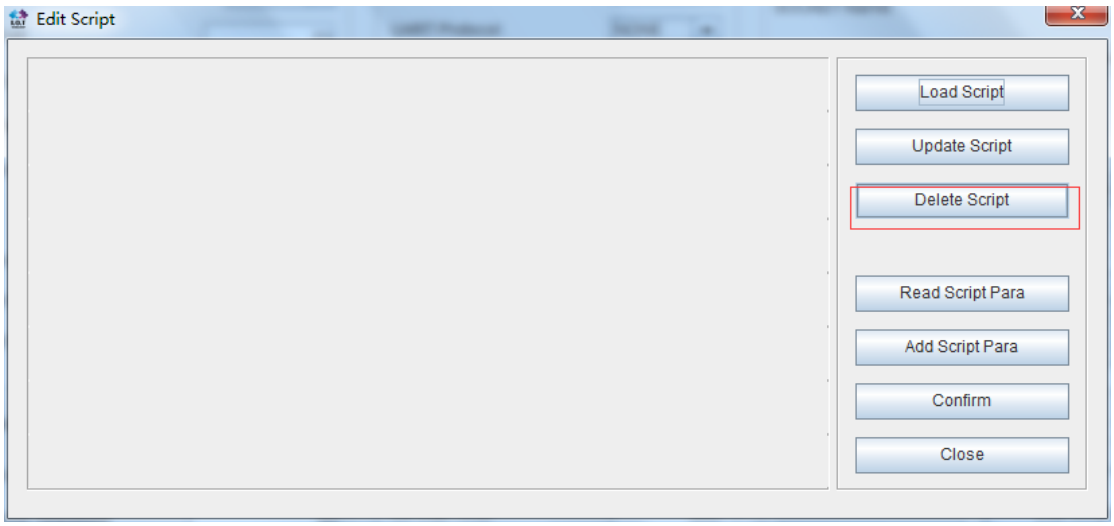
4、确认脚本文件导入之后，点击“重新启动”，因为脚本导入之后设备重启生效。



5、然后通过读取脚本参数来读取脚本里面的 FLASH 变量，然后可以通过工具直接对变量的值进行修改。



6、如果模块不需要脚本了，可以通过“删除脚本”来将脚本删除掉。



可进入 Cli 如下的目录检查脚本内容。

```
EPORT/SYS>Script
Script file:
SETHW(GPIO, OUT, 2, 1)

low1=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x30,0x0D,0x0A]
low2=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x30,0x00]
high1=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x31,0x0D,0x0A]
high2=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x31,0x00]

RECV SOCK netp


        #设置GPIO1电平为高

        IF (INPUT==high1)
                SETHW(GPIO, OUT, 2, 1)
                RETURN(FALSE)
        END

        IF (INPUT==high2)
                SETHW(GPIO, OUT, 2, 1)
                RETURN(FALSE)
        END

        #设置GPIO1电平为低
        IF (INPUT==low1)
                SETHW(GPIO, OUT, 2, 0)
                RETURN(FALSE)
        END

        IF (INPUT==low2)
                SETHW(GPIO, OUT, 2, 0)
                RETURN(FALSE)
        END

        OUTPUT=INPUT
        RETURN(TRUE)

END
```

5. 脚本控制 E10 产品 GPIO

5.1. 功能需求

netp Socket 通道接收到如下特定网络数据时，控制 GPIO2 输出高低电平，其他数据做默认透传功能。

接收到“255:W:OSLED:1\0”或者“255:W:OSLED:1\r\n ”数据时，GPIO2 输出高电平，对应如下十六进制数据：

32 35 35 3A 57 3A 4F 53 4C 45 44 3A 31 00

32 35 35 3A 57 3A 4F 53 4C 45 44 3A 31 0D 0A

接收到“255:W:OSLED:0\0”或者“255:W:OSLED:0\r\n ”数据时，GPIO2 输出低电平，对应如下十六进制数据：

32 35 35 3A 57 3A 4F 53 4C 45 44 3A 30 00

32 35 35 3A 57 3A 4F 53 4C 45 44 3A 30 0D 0A

5.2. 功能实现

测试脚本在 HIS_EXAMPLE 目录下“E10_SCRIPT_Beijing-const_20170614.txt”文件。脚本内容如下

```
SETHW(GPIO, OUT, 2, 1)  //初始化 GPIO2 输出高电平

low1=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x30,0x0D,0x0A]
low2=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x30,0x00]
high1=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x31,0x0D,0x0A]
high2=[0x32,0x35,0x35,0x3A,0x57,0x3A,0x4F,0x53,0x4C,0x45,0x44,0x3A,0x31,0x00]

RECV SOCK netp     //接收到 netp socket 的数据
        #设置 GPIO1 电平为高
        IF (INPUT==high1)    //比较接收到的数据和 high1
                SETHW(GPIO, OUT, 2, 1)
                RETURN(FALSE)
        END

        IF (INPUT==high2)
                SETHW(GPIO, OUT, 2, 1)
                RETURN(FALSE)
        END

        #设置 GPIO1 电平为低
        IF (INPUT==low1)
                SETHW(GPIO, OUT, 2, 0)
                RETURN(FALSE)
        END

        IF (INPUT==low2)
                SETHW(GPIO, OUT, 2, 0)
                RETURN(FALSE)
        END

        OUTPUT=INPUT
        RETURN(TRUE)

END
```

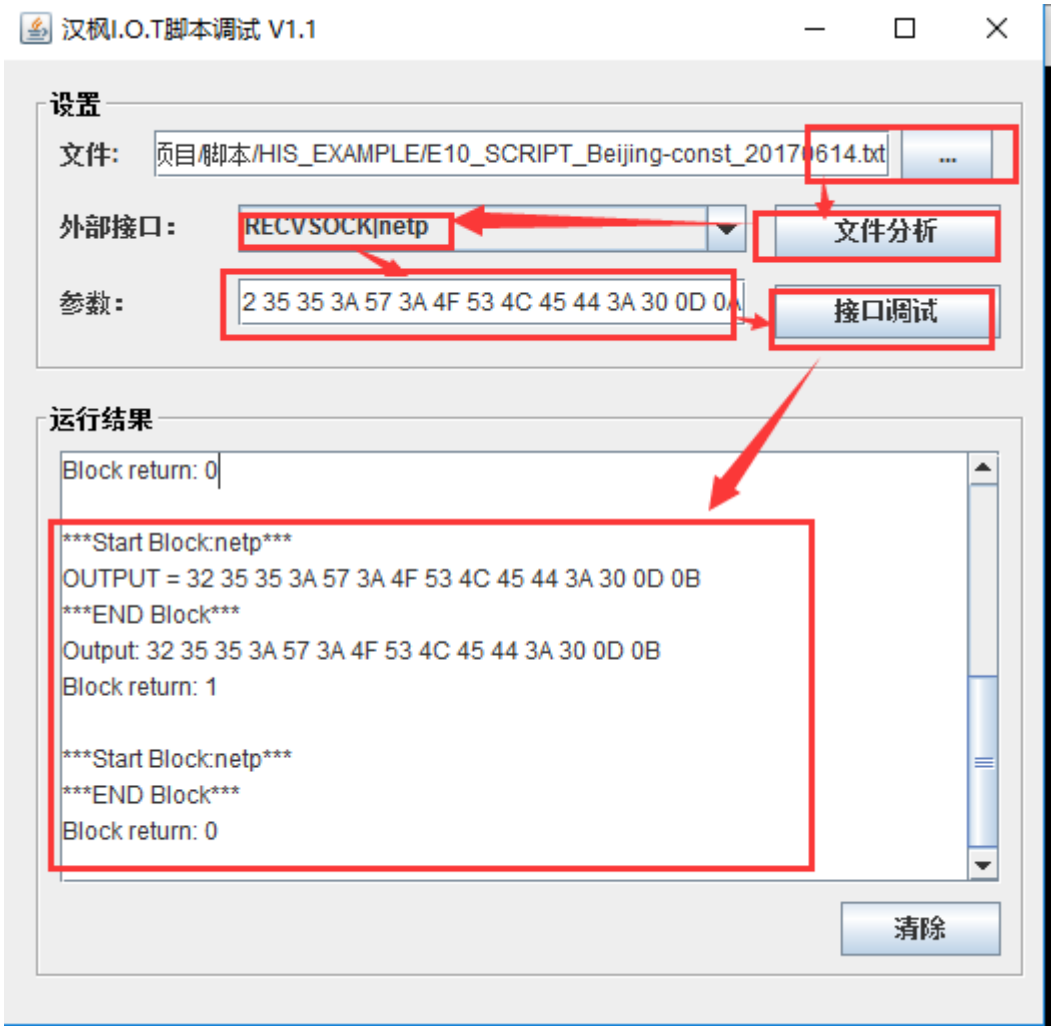

5.3. HisUI 工具测试

Step1：打开 HisUI 工具，并加载脚本文件

Step2：点击【文件分析】，选择脚本测试的接口。

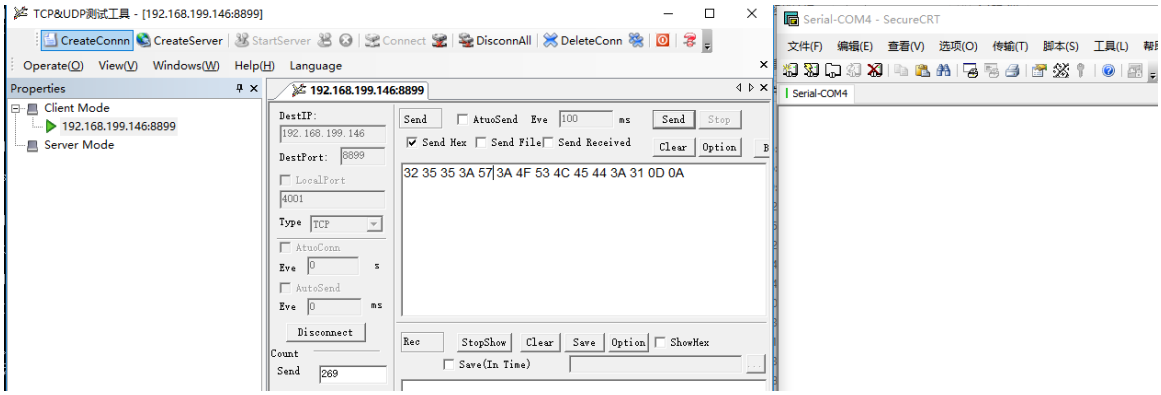
Step3：输入测试的参数，并点击【接口调试】进行测试，参数内容如果以空格隔开，则认为是 16 进制格式（如下图），否则以 ASCII 码格式识别数据内容。

Step4：根据输入的不同参数，测试功能是否实现，如下样例中，Block 如果 return 1，则数据正常透传下发，如果 return 0 则数据不下发。

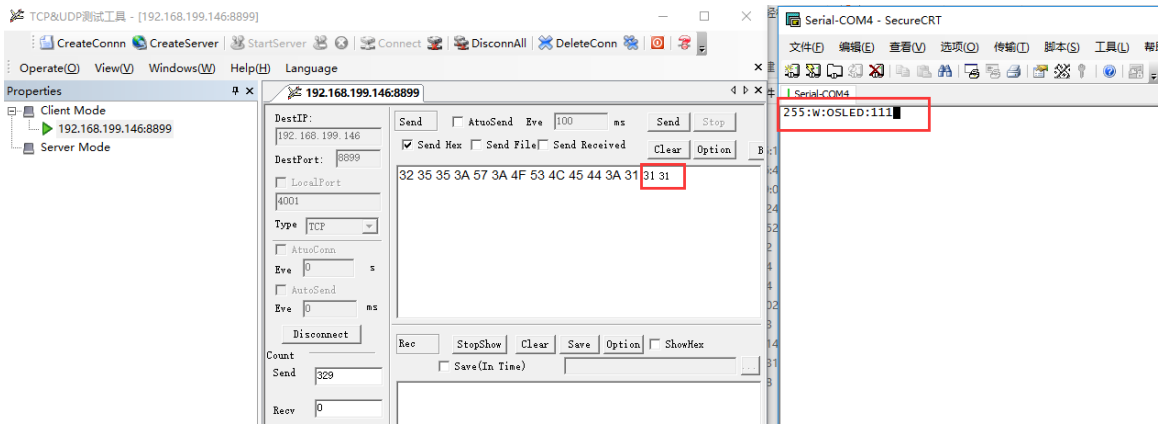


5.4. E10 产品测试

与 E10 产品 netpsocket 建立 TCP 连接，发送测试数据，如下测试数据可直接让 GPIO2 引脚输出高电平，并且数据并不从串口输出。



稍稍改动数据内容，可看到从串口输出对应数据。



附录A.　心跳包功能案例

A.1.功能需求

netp Socket 通道有有网络连接时向串口发送” netp connected “，网络连接断开时向串口发送” netp disconnected “。

netp Socket 通道链接正常时，定时 30 秒发送心跳数据“heartbeat data”。

A.2.功能实现

测试脚本在 HIS_EXAMPLE 目录下“E10_E30_HF5111B_SCRIPT_example_heartbeat.txt”文件。脚本内容如下

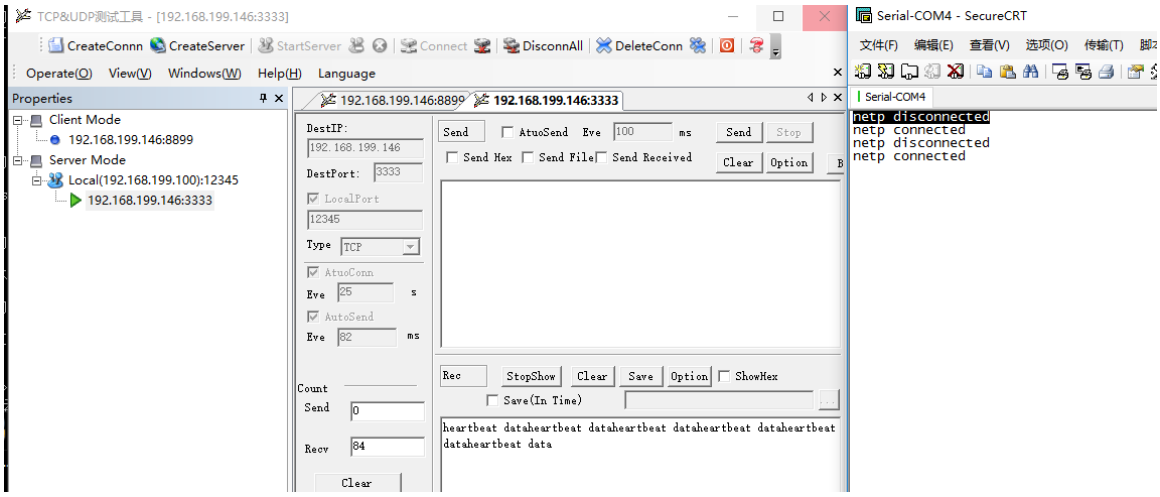
```
# “netp” 通道连接标志,0-未连接,1-连接成功
connFlag = 0
# “netp” 通道接收到数据的时间
recvTime = 0

# “netp” 通道心跳数据
space = [0x20]
beatData = "heartbeat" + space + "data"

connStr = "netp connected\r\n"

# “netp” 通道连接成功
CONN SOCK netp
 connFlag=1
 recvTime=SYSTIME
 SEND(UART, uart0, connStr)
END

# “socka" 通道断开连接
DISCONN SOCK netp
 connFlag=0
 SEND(UART, uart0, "netp disconnected\r\n")
END

RECV SOCK netp
 recvTime=SYSTIME
 OUTPUT=INPUT
 RETURN(TRUE)
END

TIMER HeartBeat 1000
 t=SYSTIME
 t=t-recvTime
```



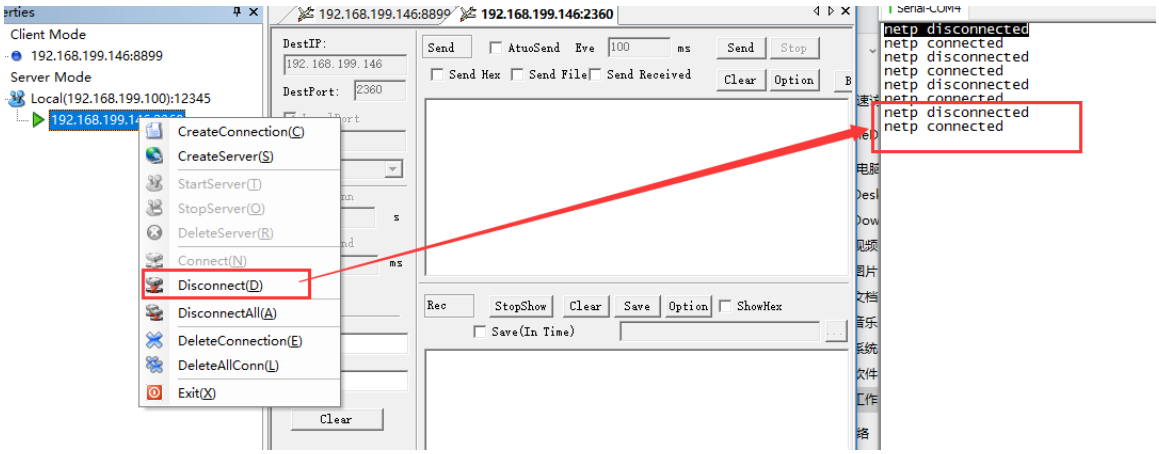
```
    IF (connFlag == 1)
        IF (t > 30000)
            SEND(SOCK, netp, beatData)
            recvTime=SYSTIME
        END
    END
END
```

A.3.产品测试

实际测试的效果



手动点击 Disconnect 之后可看到串口输出对应数据。



附录B.注册包功能案例

B.1. 功能需求

每次 netp 通道连接建立或者发送数据的时候，在数据头部增加额外的 id 信息或者 MAC 地址信息以供服务器区分设备。

B.2. 功能实现

测试脚本在 HIS_EXAMPLE 目录下“E10_E30_HF5111B_SCRIPT_example_login.txt”文件。脚本内容如下，默认脚本并没有启用注册包功能，可修改脚本 REGEN 和 REGTCP 入口参数，完成此案例的实际测试。

```
#----配置参数----------------------------------------
# 注册包类型，0-不开启注册包，1-id 模式，2-MAC 模式
REGEN = 0
# 注册包方式，1-first 连接时发送，2-every 每次发送数据时
REGTCP = 1
# 注册包 ID，0-65535，十六进制数组形式
REGID = [0x04, 0x57, 0xFB, 0xA8]
#-------------------------------------------------------

# 获取设备的 MAC 地址
mac = ""
macStr = MAC
mac = macStr.stringHex()

# “netp” 通道连接成功回调
CONN SOCK netp
 IF (REGTCP == 1)
      IF (REGEN == 1)
           SEND(SOCK, netp, REGID)
      END

      IF (REGEN == 2)
           SEND(SOCK, netp, mac)
      END
 END
END

# 串口接收到数据回调
RECV UART uart0
 flag = 0
 IF (REGTCP == 2)
      IF (REGEN == 1)
```



```
            OUTPUT = REGID + INPUT
            flag = 1
        END

        IF (REGEN == 2)
            OUTPUT = mac + INPUT
            flag = 1
        END
    END

    IF (flag == 0)
        OUTPUT = INPUT
    END
    RETURN(TRUE)
END
```


附录C.FLASH 参数使用案例

C.1. 功能需求

将心跳数据变量 beatData 定义成 flash 类型参数，通过 IOTService 工具（关于 IOTService 工具使用参考 IOTServices 使用说明手册）可以对这个参数进行修改，修改成功后，脚本会优先使用这些修改后的值。从而实现不同模块使用同一个脚本文件，但这些参数具有不同的初始值。netp Socket 通道链接正常时，定时 500 毫秒发送心跳数据。

C.2. 功能实现

测试脚本是将“E10_E30_HF5111B_SCRIPT_example_heartbeat.txt 文件里面的 beatData 变量定义为 FLASH(STRSTR) 类型变量及增加 FLASHMAGIC 变量定义（FLASHMAGIC 说明见 2.13 Flash 参数说明）。脚本内容如下，

```
#心跳包脚本示例
#  “netp”  通道连接标志,0-未连接,1-连接成功
connFlag = 0
#  “netp”  通道接收到数据的时间
recvTime = 0

#  “netp”  通道心跳数据
FLASHMAGIC = 200
FLASH(STRSTR)   beatData = "BEAT_DATA"
#  “netp”  通道连接成功
CONN SOCK netp
 connFlag=1
 recvTime=SYSTIME
END

#  “socka" 通道断开连接
DISCONN SOCK netp
 connFlag=0
END


RECV SOCK netp
 recvTime=SYSTIME
 OUTPUT=INPUT
 RETURN(TRUE)
END

TIMER HeartBeat 1000
 t=SYSTIME
 t=t-recvTime
 IF (connFlag == 1)
       IF (t > 500)
```

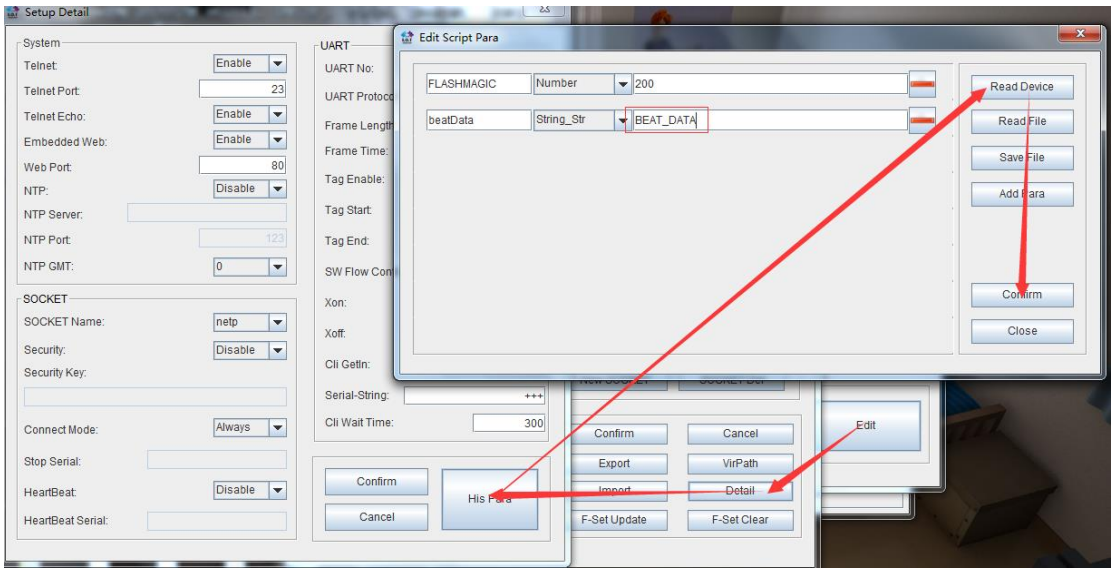


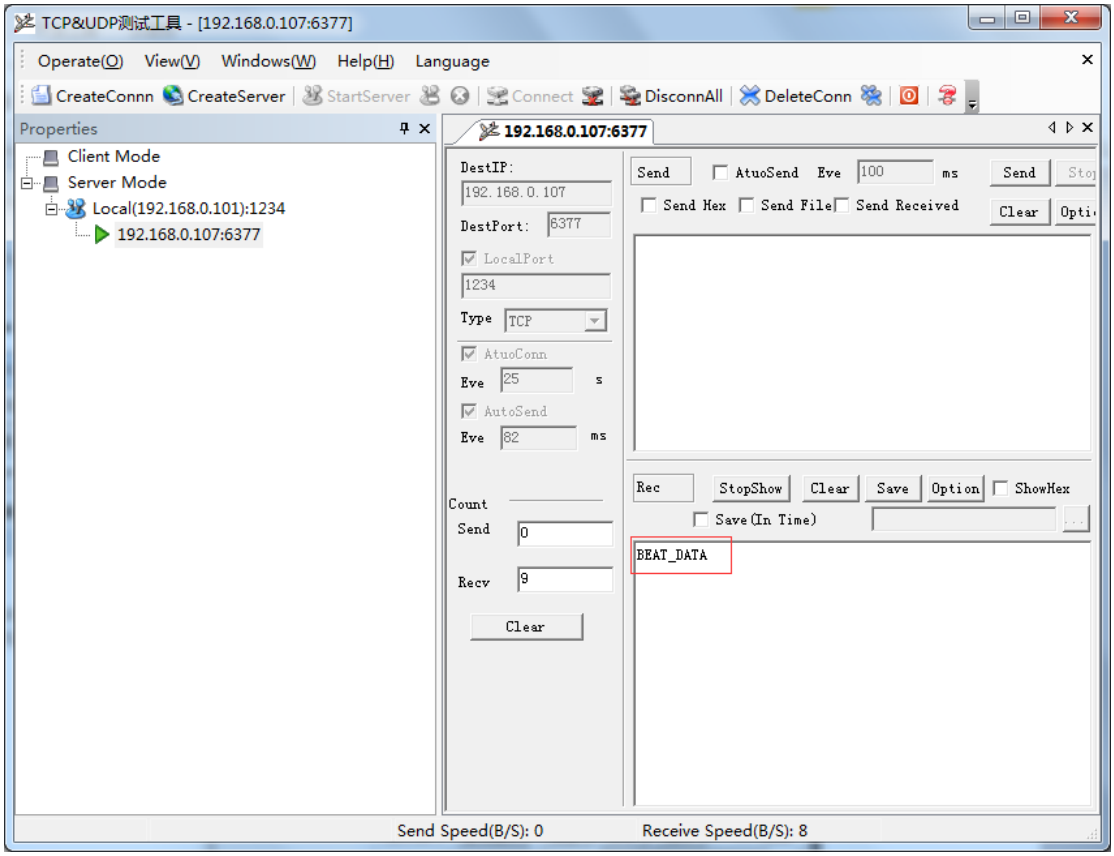
```
            SEND(SOCK, netp, beatData)
            recvTime=SYSTIME
        END
    END
    END
```

C.3. 产品测试

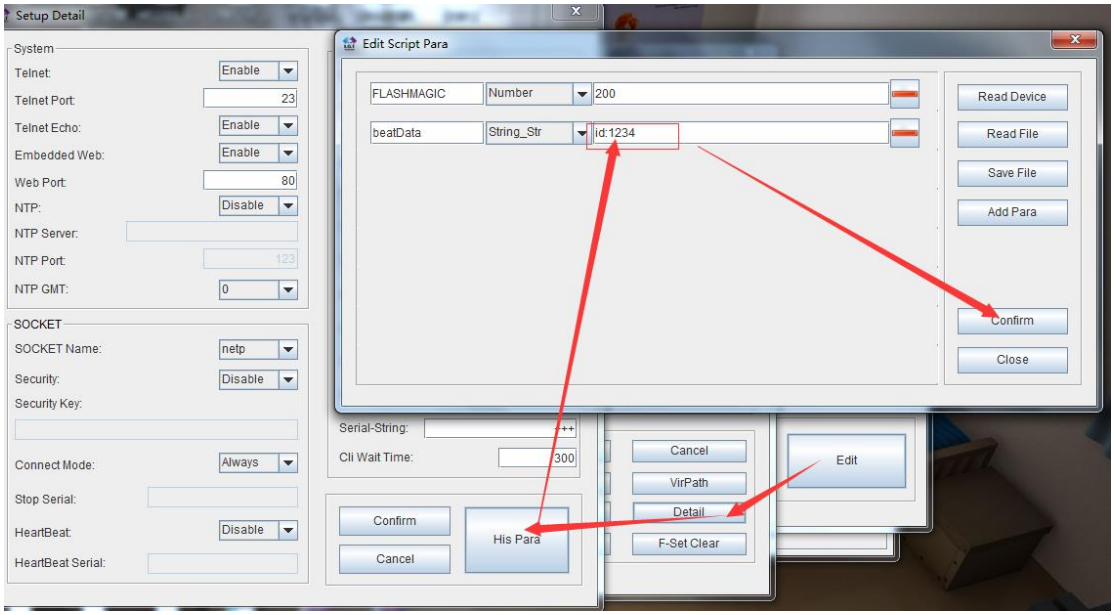
实际测试：

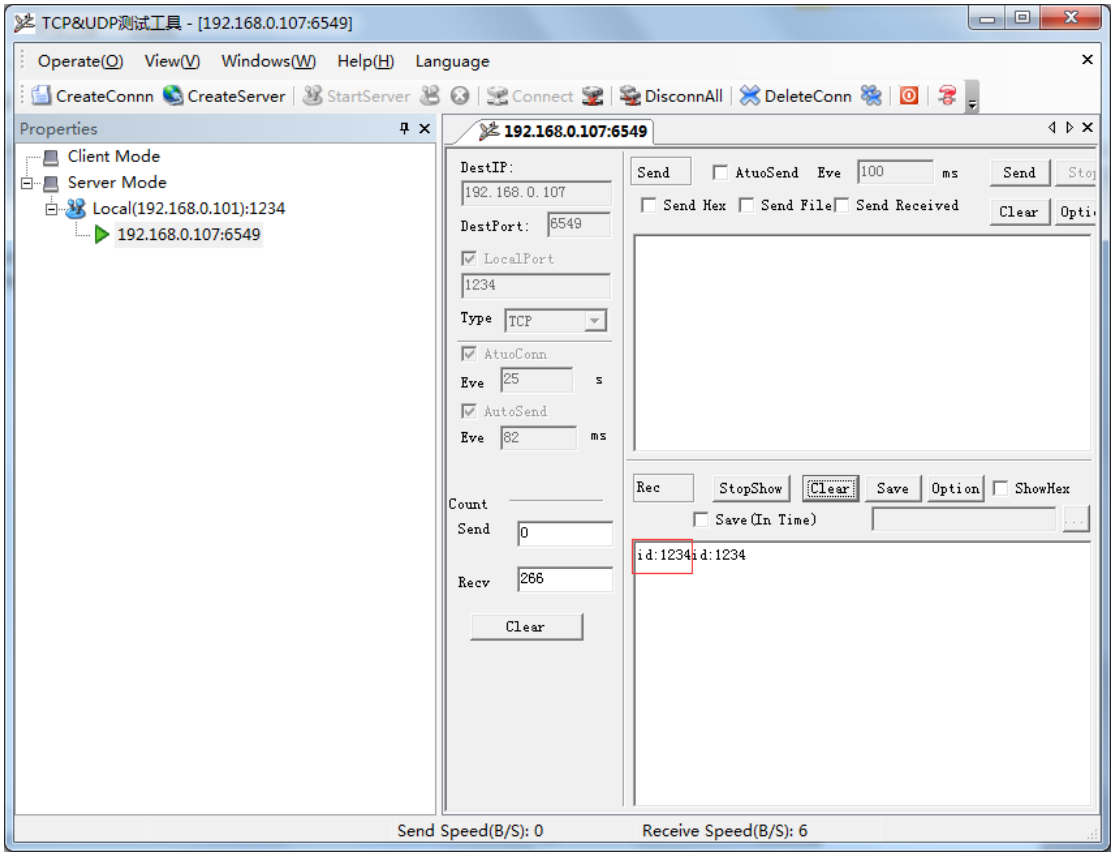
1.通过 IOTService 工具点击“读取设备参数”，获取到 FLASH 参数,参数为默认值时测试效果如图所示：





2.通过 IOTService 工具把心跳数据的参数的值修改为 id:1234,点击确认，点击模块重启之后测试效果如下：





C.4. 其他说明

测试说明脚本里面可以将需要修改的参数定义为 flash 参数，通过 IOTService 工具来修改这些参数的值来实现同一个脚本参数的初始化值不同，从而达到我们可以任意修改参数值的目的。

附录D.　HF6508 的 AI 状态检测报警测试案例

D. 1. 功能需求

HF6508 每隔一秒主动下发查询 AI 状态的指令（相关指令介绍查看 6508 使用手册），检测 AI1-AI8 的状态值，将其回复的数据超过设定的阈值之后将其上报到服务器，否则不上报，上报数据可以加上帧头等以作为区分设备。AI 状态查询介绍如图所示：

4.2.3. AI 状态查询 功能码 04

查询指令：　01 04 00 00 00 08 f1 cc

返回（例）：　01 04 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 2c

第 1-4 路测试 0-5V 电压量，用户使用不要超过此电压值。

第 5-8 路测试 0-25mA 电流值，用户使用不要超过此电流值。

第 4 个字节是返回的总数据字节，每 2 个字节代表一个数值，由低到高排序，例如第 4 第 5 字节表示 AI1 的值，这里是 0x0000，具体换算公式为首先将 0x0000 转换为 10 进制数 value。通过（value / 1024）* 5 计算公式得到模拟量的值。

注：这里计算公式需要修正一下，设备内部已经接好了 200Ω 电阻，所以这里的计算公式应该为：{[（value/1024）*5]/200}*1000

D. 2. 功能实现

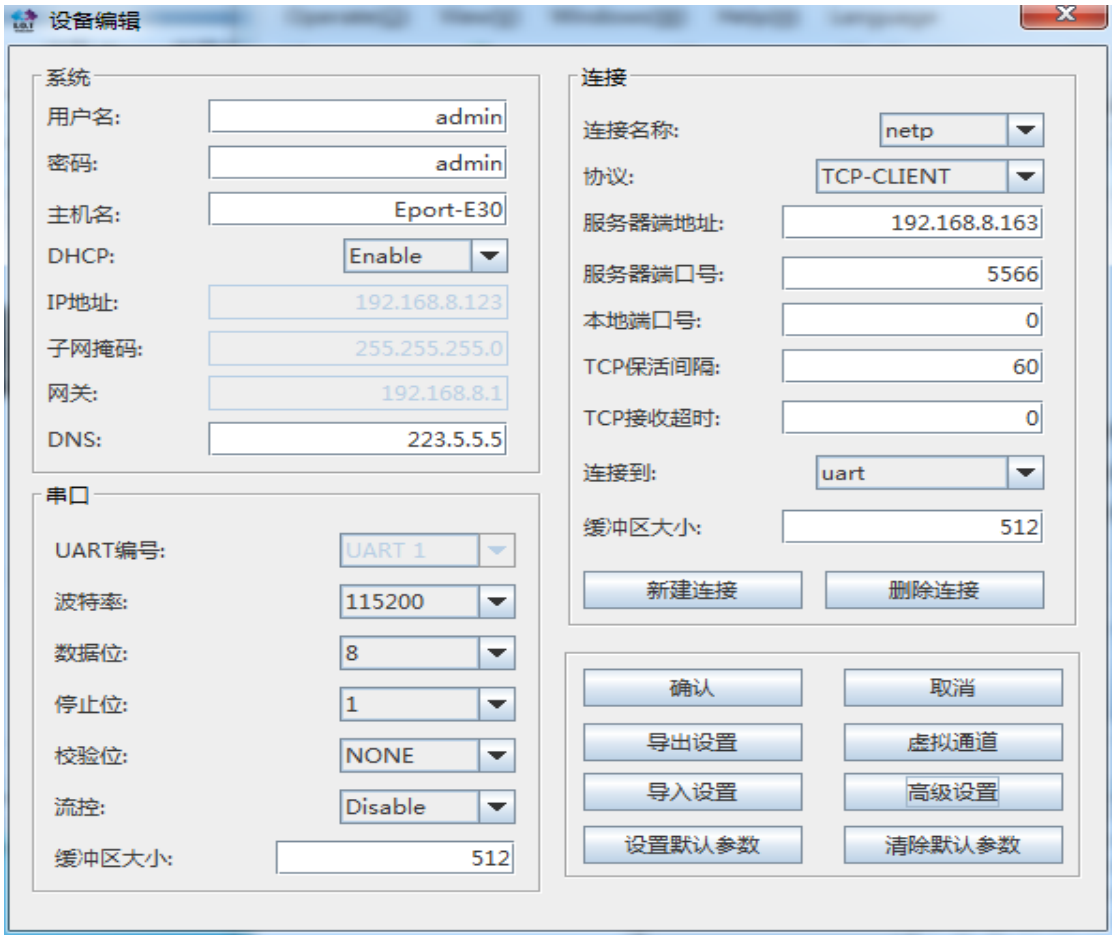
脚本是 AI_TEST.txt 文档，上报的数据加了 EA 和设备 mac，作为服务器区分设备，上报的数据格式为：‘EA+MAC+数据”，脚本里面的阈值和下发时长都可以按照自己需求去改动，这里默认模拟量的阈值是 10mA，所以算下来的 value 值是 409（{[（409/1024）*5]/200}*1000=10mA），查询指令下发的时间间隔 1s（秒），内容如下：

```
cmd1=[0x01,0x04,0x00,0x00,0x00,0x08,0xF1,0xCC]
check1=[0x01,0x04]
DATA_check1=[0x00]
yuzhi_mA=409
n=3
zhengtou=[0xEA]
mac=MAC
mac_1=mac.stringHex()
RECV UART uart0
      DATA=INPUT
      DATA_S=zhengtou+mac_1+DATA
    DATA_check1=DATA.subString(0,2)
  IF(DATA_check1==check1)
     #检查 8 路电流值
           FOR n,3,19,2
           DATA1=DATA.charAt(n)
             DATA2=DATA.charAt(n+1)
             IF(DATA1<0)
               DATA1=(DATA1+256)*256
             ELSE
```

```
            DATA1=DATA1*256
          END
        IF(DATA2<0)
          DATA2=DATA2+256
        END
        DATA3=DATA1+DATA2
        IF(DATA3>yuzhi_mA)
         SEND(SOCK,netp,DATA_S)
        END
    END
  END
END

TIMER cmd 1000

    SEND(UART,uart0,cmd1)

END
```

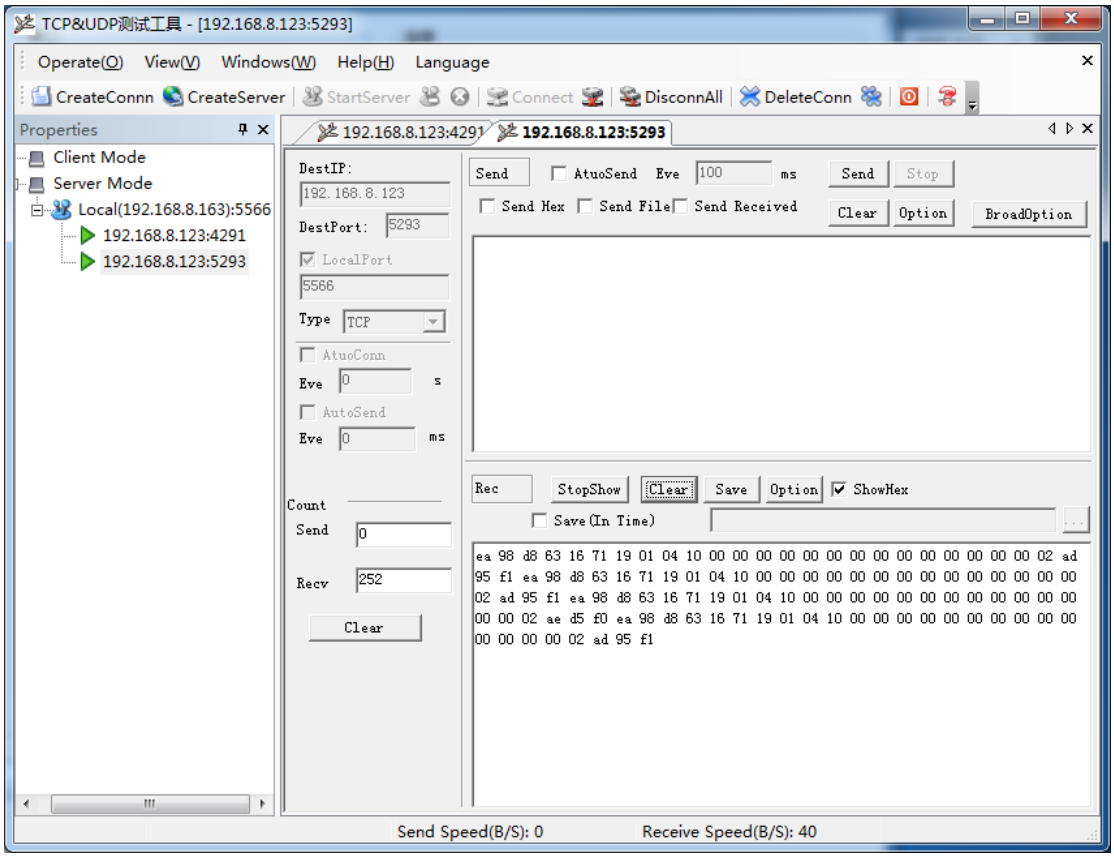
D. 3. 测试结果

1. 配置设备为客户端连接服务器，如图所示：



2. 6508 设备 AI1 接上输出 16mA 的电流设备,超过阈值，所以上报到服务器端。

注：帧头 ea,MAC：98 d8 63 16 71 19，数据：01 04 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 ad 95 f1



附录E. 设备连接断开重启测试案例

E. 1. 功能需求

检测网络连接状态，如果断开设备重启，每隔一秒检测一下连接状态。

E. 2. 功能实现

脚本内容如下：

```
connflg=0
CONN SOCK netp
      connflg=1
END
DISCONN SOCK netp

       connflg=0
END
TIMER time    1000
      IF (connflg == 0)
                RESET
```



```
    END
END
```

