迅饶 sunfull
专注自控 物物互联

【操作说明】

迅饶网关/触摸屏 JS 脚本编辑器

操作说明

目录

1、JS 脚本编辑器介绍

迅饶的网关、触摸屏中都带有可做逻辑处理的 JS 脚本编辑器，用户可以通过编辑脚本语言实现一些逻辑控制。脚本编辑器内置一些常用函数，用户可以选择一些函数编辑语言，在编辑完成后，点击“语法检查”，会自动检查语法。假如有语法错误，会提示具体哪一行语法有问题。

打开相应的网关或者触摸屏的配置软件，在最上方的菜单栏中有一栏为“Java 脚本”或者视图栏下的“JS 脚本编辑器”即为相应的进入脚本编辑界面按钮。

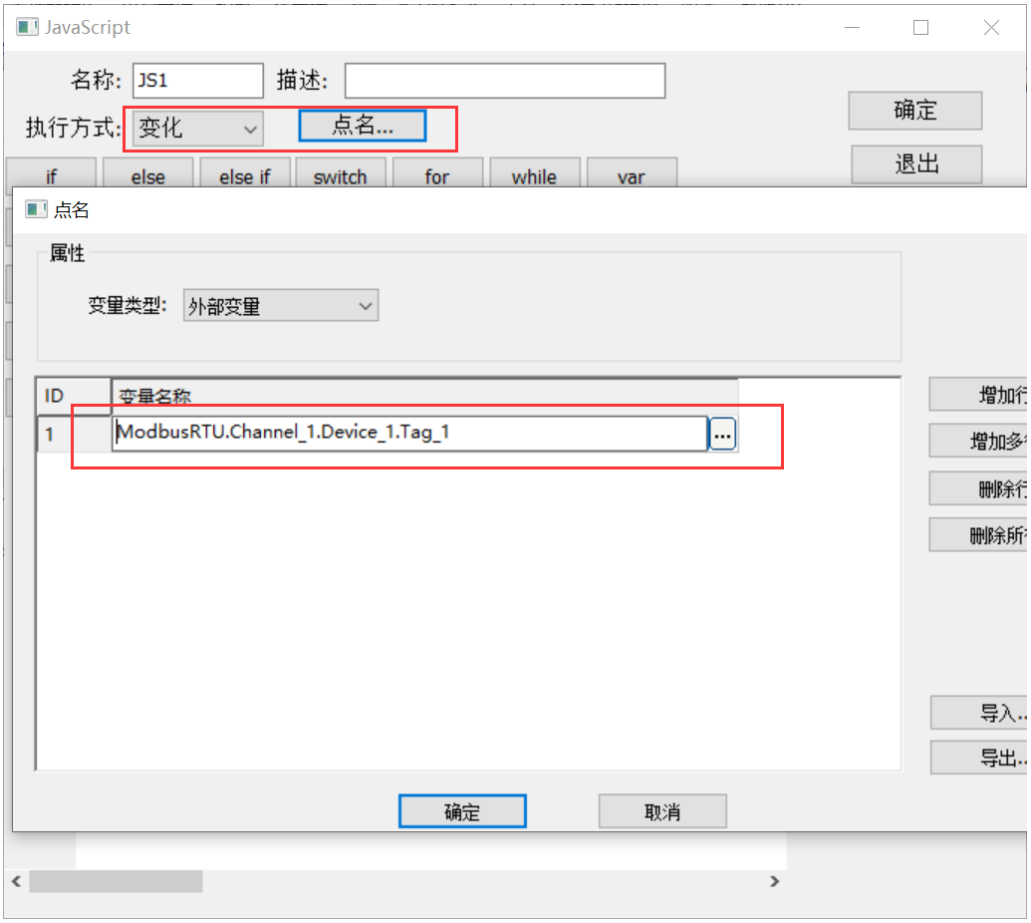
2、JS 脚本的执行模式

2.1、循环模式

脚本选择执行方式为“循环”，即表示此脚本将会以设定的执行周期进行周期性循环执行。

注：此脚本将会在设备上电的时候一直周期性的循环执行。

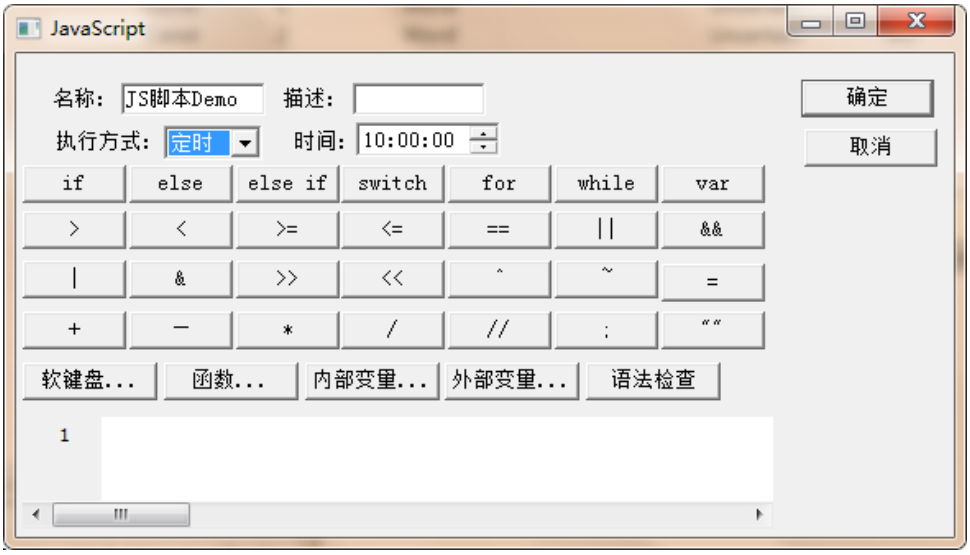
2.2、变化模式



脚本选择执行方式为“变化”，即表示此脚本将会在设定的任意点名值发生变化的时候执行。

注：此脚本只在绑定的点名值发生变化的时候执行一次。

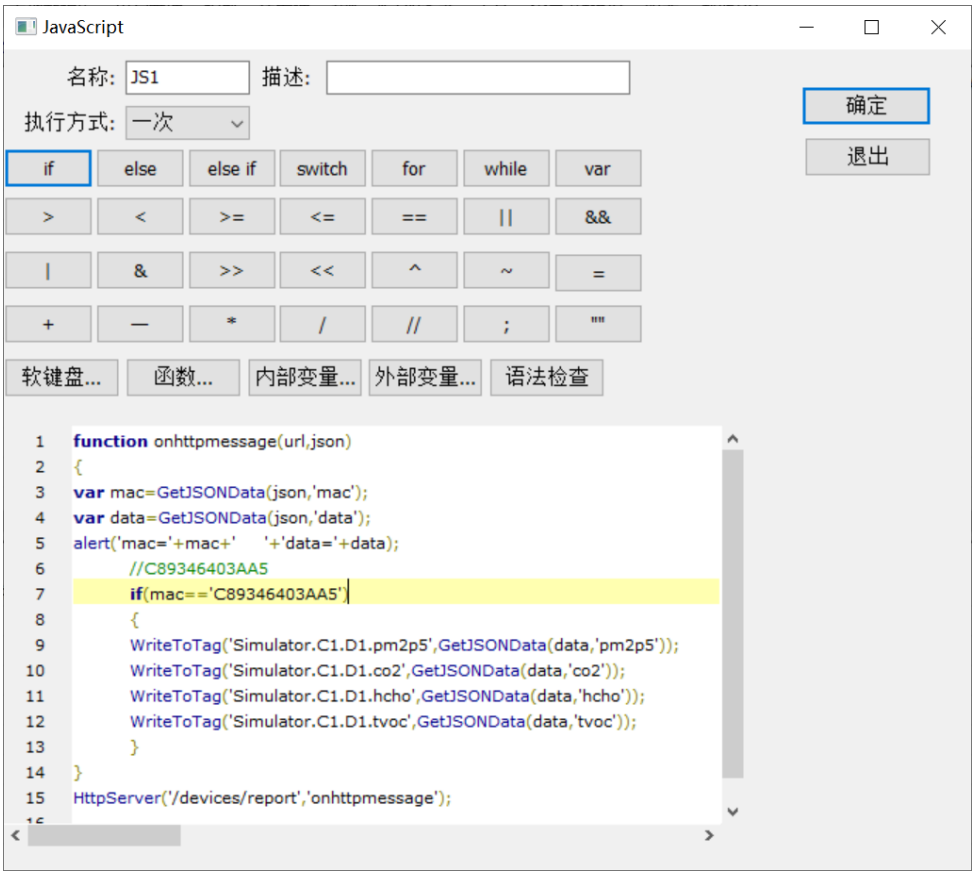
2.3、定时模式



脚本选择执行方式为“定时”，即表示此脚本将会在设定的时间执行脚本的脚本内容。

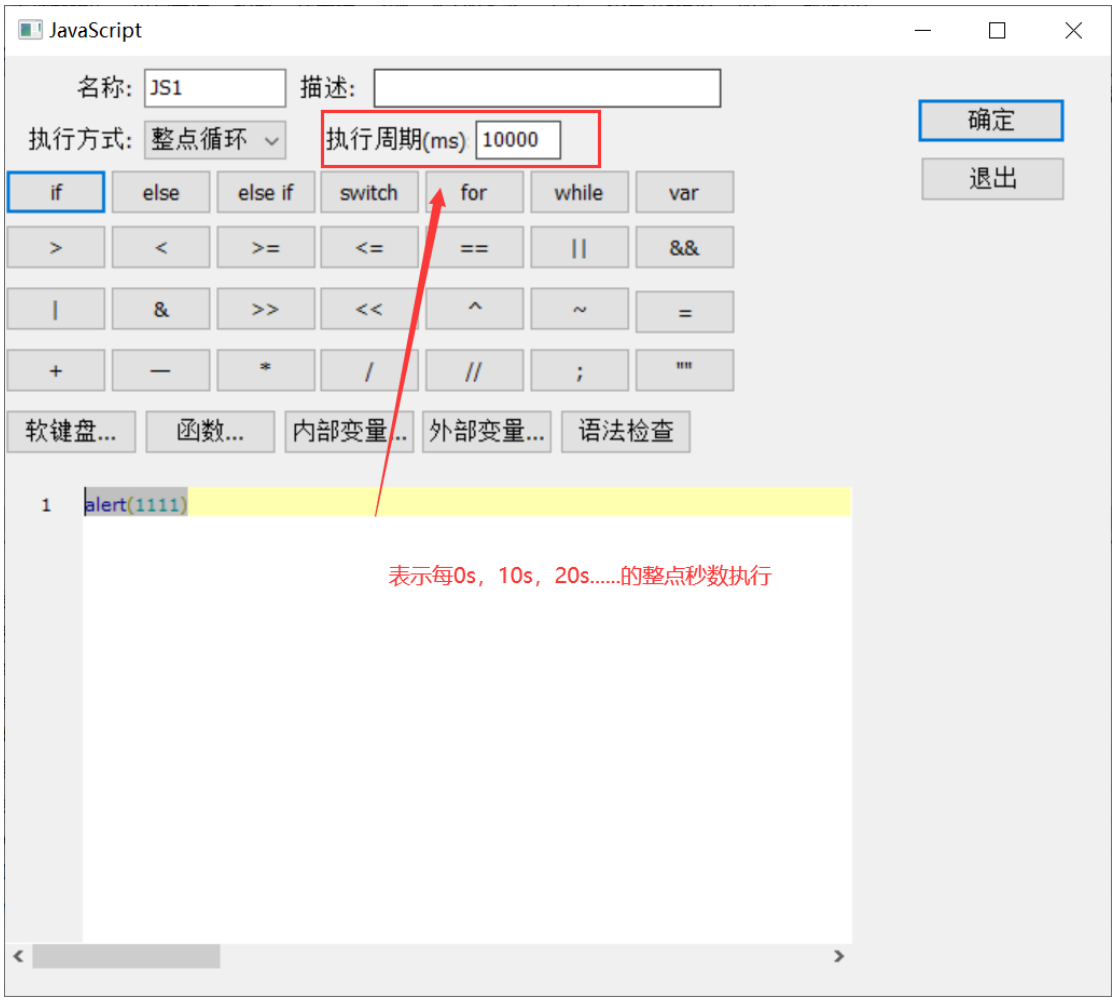
注：此脚本只在设定的时间执行一次。

2.4、一次模式



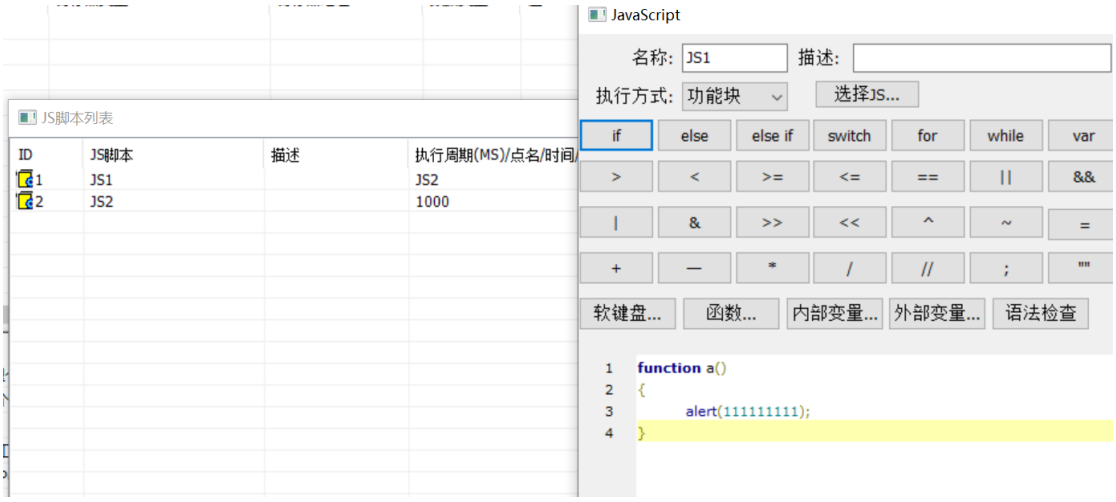
脚本选择执行方式为“一次”，即表示此脚本将会在网关启动的时候执行一次脚本的脚本内容。

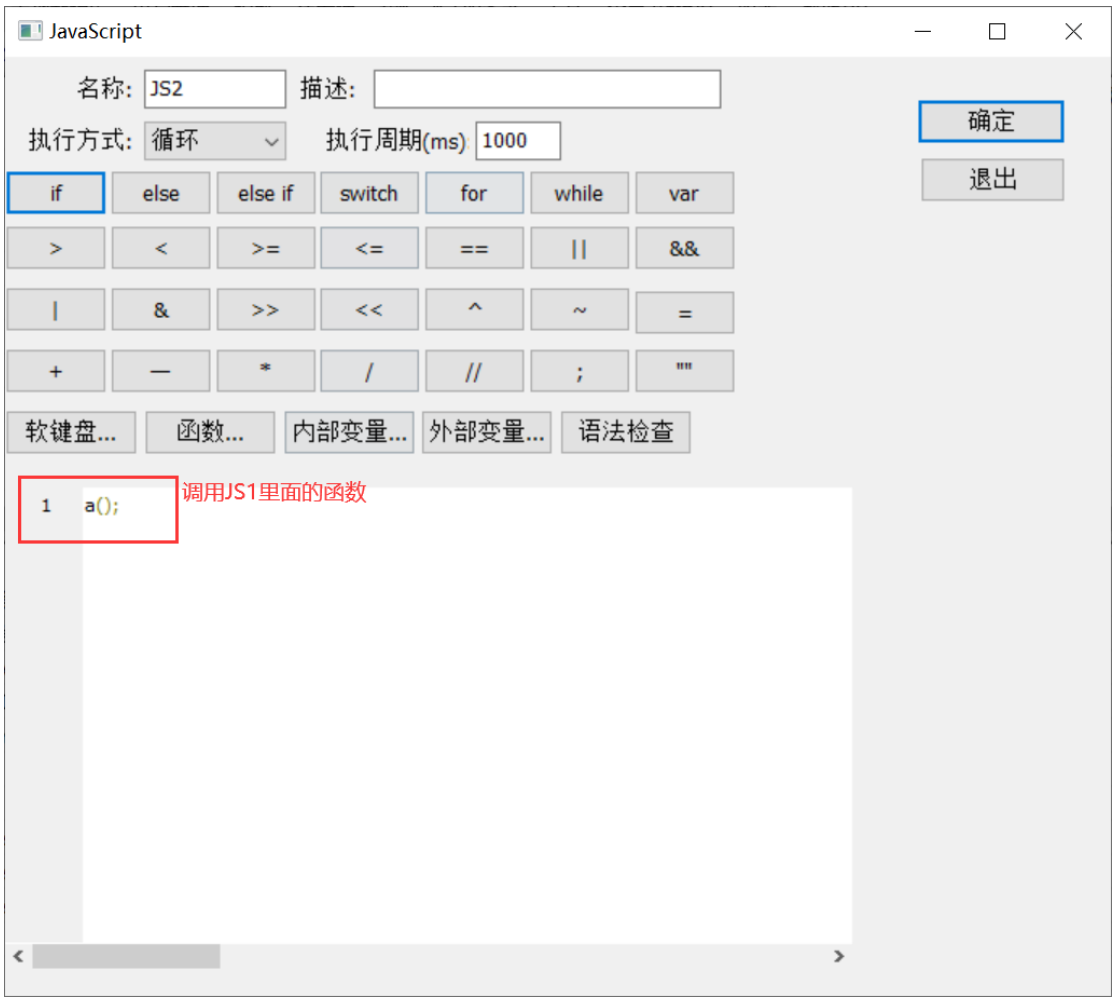
2.5、整点循环模式

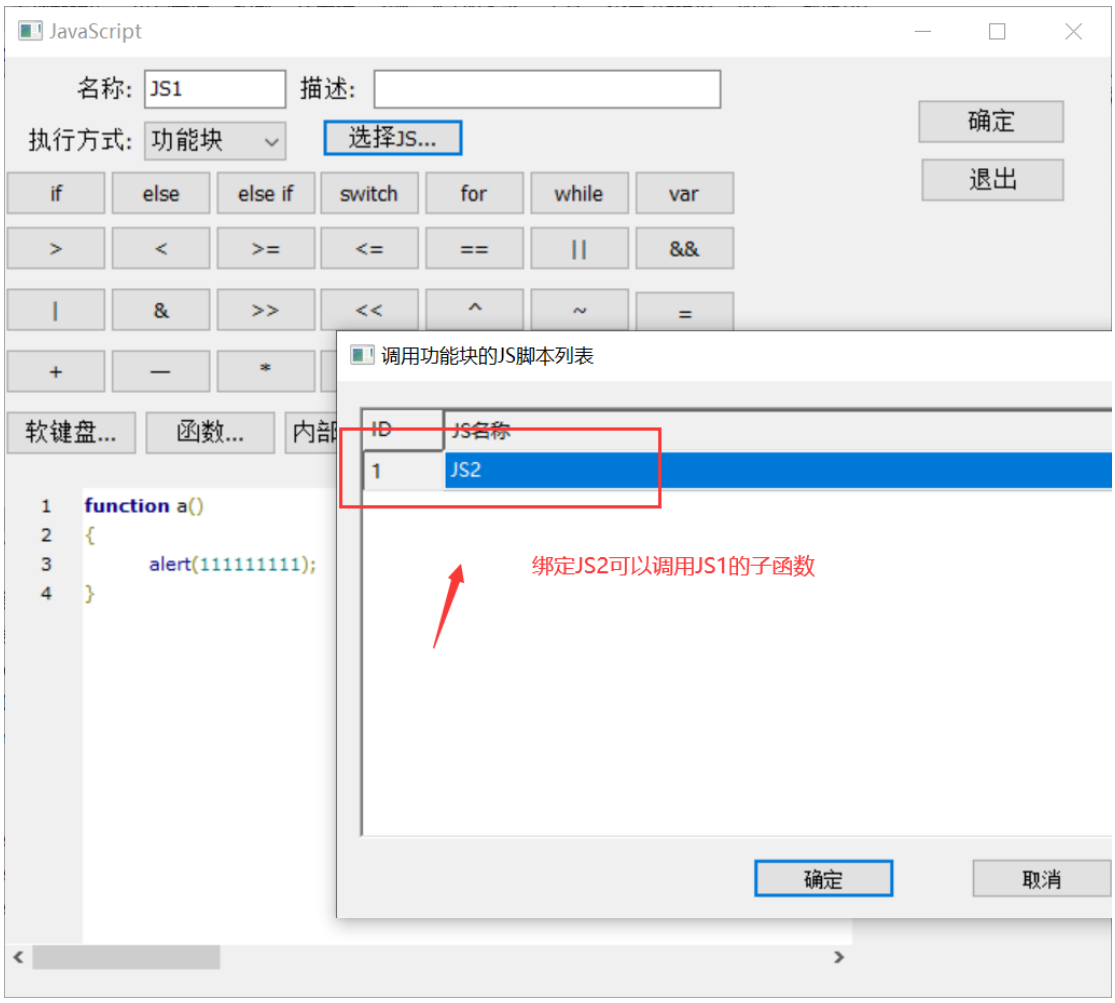


脚本选择执行方式为“整点循环”，即表示此脚本将会每 0s，10s，20s……的整点时间的时候执行脚本内容。

2.6、功能块模式


JS脚本列表
ID
JS脚本
描述
执行周期(MS)/点名/时间/
1
JS1
JS2
2
JS2
1000
JavaScript
名称: JS1
描述:
执行方式: 功能块
选择JS...
if
else
else if
switch
for
while
var
>
<
>=
<=
==
||
&&
|
&
>>
<<
^
~
=
+
—
*
/
//
;
""
软键盘...
函数...
内部变量...
外部变量...
语法检查
1 function a()
2 {
3 alert(111111111);
4 }


JavaScript
名称: JS2
描述:
执行方式: 循环
执行周期(ms) 1000
确定
退出
if
else
else if
switch
for
while
var
>
<
>=
<=
==
||
&&
|
&
>>
<<
^
~
=
+
—
*
/
//
;
""
软键盘...
函数...
内部变量...
外部变量...
语法检查
1 a();
调用JS1里面的函数



脚本选择执行方式为“功能块”，即表示此脚本为一个子程序可供其他脚本调用。

3、常用函数介绍

3.1 ReadFromTag (x)

3.1.1 函数功能介绍

ReadFromTag(x)函数是读取网关或者触摸屏中通讯点的当前值，将网关或者触摸屏中建立的设备通讯点转换到 JS 脚本编辑器的内部中，便于进行逻辑编写和传值等操作。

参数：x 只能填写网关的内部变量/外部变量点名称；不能直接填写用户自定义变量和数值。

3.1.2 函数操作举例

ReadFromTag（）函数使用 Demo：

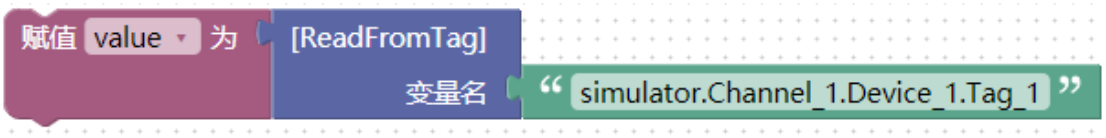
var Value = ReadFromTag('simulator.Channel_1.Device_1.Tag_1');

解析：

①读取外部变量 simulator.Channel_1.Device_1.Tag_1 的值；

②将读取到的值赋给 JS 脚本编辑器的自定义的变量 Value；

3.1.3 图形编程举例

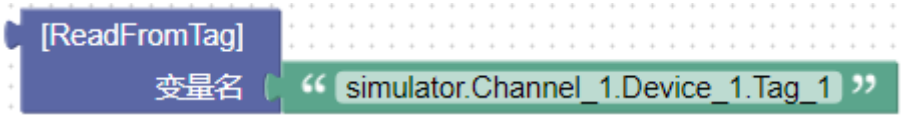


步骤：

1. 选额工具箱-->变量-->创建变量，输入变量名“value”，点击确定。

2. 选择工具箱-->变量-->“赋值...为”，在选择下拉框中的 value 变量。

3. 选择工具箱-->取值-->ReadFromTag，点击文本-->  ，从 X2view 中拷贝点名到文本框中,挂接在 ReadFromTag 块上，如下：



4. 拖动 3 中的运算符块，然后挂接到 1 中变量块的右侧。

说明：ReadFromTag 有多个块，方便用户通过下拉框的形式选择参数。

3.2 MoveValue(x,y)

3.2.1 函数功能介绍

MoveValue(x,y)函数是将网关或触摸屏中的外部变量点或者是内部变量点进行传值操作，实现点与点之间数据的交互。

参数：x，y 只能填写网关的内部变量/外部变量点名称；不能直接填写用户自定义变量和数值。

3.2.2 函数操作举例

MoveValue()函数使用 Demo1：

MoveValue('simulator.Channel_1.Device_1.Tag_1','simulator.Channel_1.Device_1.Tag_8');

解析：①变量 x: simulator.Channel_1.Device_1.Tag_1;

②变量 y: simulator.Channel_1.Device_1.Tag_8;

③将变量 x 的值赋给变量 y;

MoveValue()函数使用 Demo2：

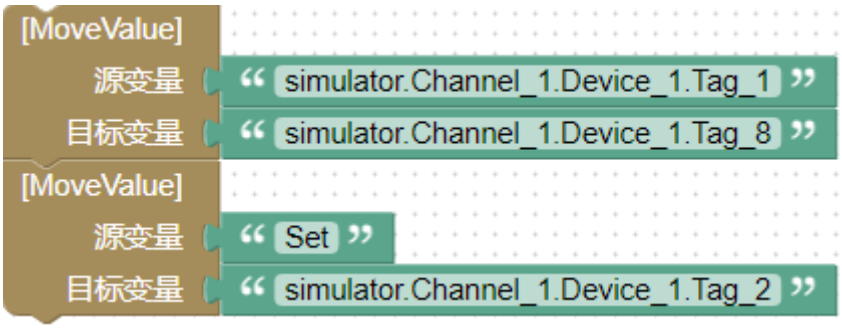
MoveValue('Set','simulator.Channel_1.Device_1.Tag_2');

解析：①变量 x: Set;

②变量 y: simulator.Channel_1.Device_1.Tag_2;

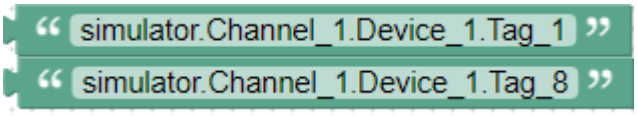
③将变量 x 的值赋给变量 y;

3.2.3 图形编程举例



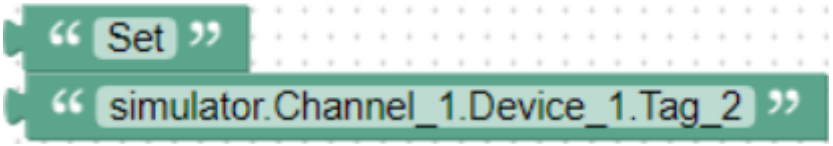
步骤：

1. 选择工具箱-->赋值-->MoveValue，点击文本-->  ，创建两字符串变量



，挂接到 MoveValue 上。

2. 选择工具箱-->赋值-->MoveValue，点击文本-->  ，创建两字符串变量



3. 将 1、2 中的模块自上而下连接在一起。

说明：MoveValue 有多个块，方便用户通过下拉框的形式选择参数。

3.3 WriteToTag (x,y)

3.3.1 函数功能介绍

WriteToTag(x,y)函数是将一个实际的数值或者是 JS 脚本中通过逻辑处理过后的数值赋给网关或者触摸屏中的某一个变量，用于对网关或者触摸屏中的变量赋值，会执行对设备一个写操作。

参数：

x 只能填写网关的内部变量/外部变量点名称；不能直接填写用户自定义变量和数值。

y 不能直接填写网关的内部变量/外部变量点名称，可以为数值，自定义变量。

3.3.2 函数操作举例

WriteToTag()函数使用 Demo1：

WriteToTag('simulator.Channel_1.Device_1.Tag_1',100);

解析：

①变量 x: simulator.Channel_1.Device_1.Tag_1;

②将数值 100 写给变量 x。

WriteToTag()函数使用 Demo2：

var Value=3;

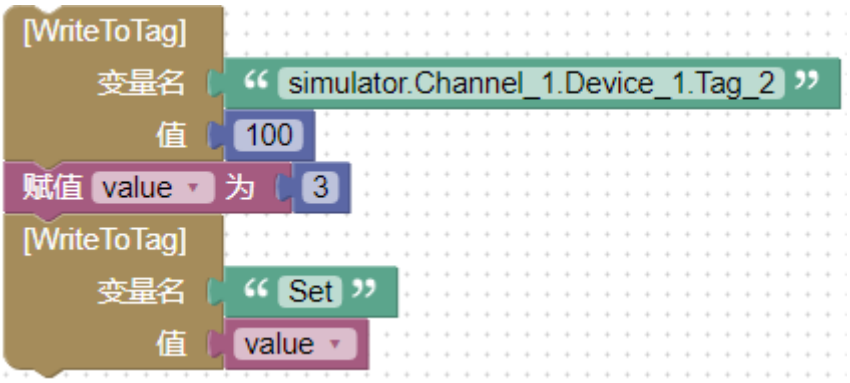
WriteToTag('Set',Value);

解析：

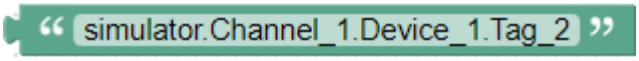
①变量 x: Set;

②将 Value 的值写给变量 x。

3.3.3 图形编程步骤

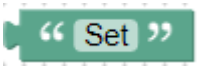
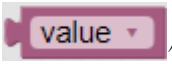


步骤：

1. 选择工具箱-->赋值->WriteToTag，点击文本-->  ，创建第一个参数  ；点击数学-->  ，创建第二个参数  ，然后挂接到 WriteToTag 即可

2. 选择工具箱-->变量--> “赋值...为”，选择下拉框中的 value 变量(如果 value 变量不存在，仿照 3.1.3 创建一个)；点击数学-->  ，创建参数  ，挂接到变量 value 上。

3. 将 value 块挂在 WriteToTag 块下方。

4. 选择工具箱-->赋值->WriteToTag，点击文本-->  ，创建第一个参数  ，点击变量-->  作为第二个参数，挂在 WriteToTag 上。

5. 将 4 中的 WriteToTag 块挂接到 value 下

说明：WriteToTag 有多个块，方便用户通过下拉框的形式选择参数

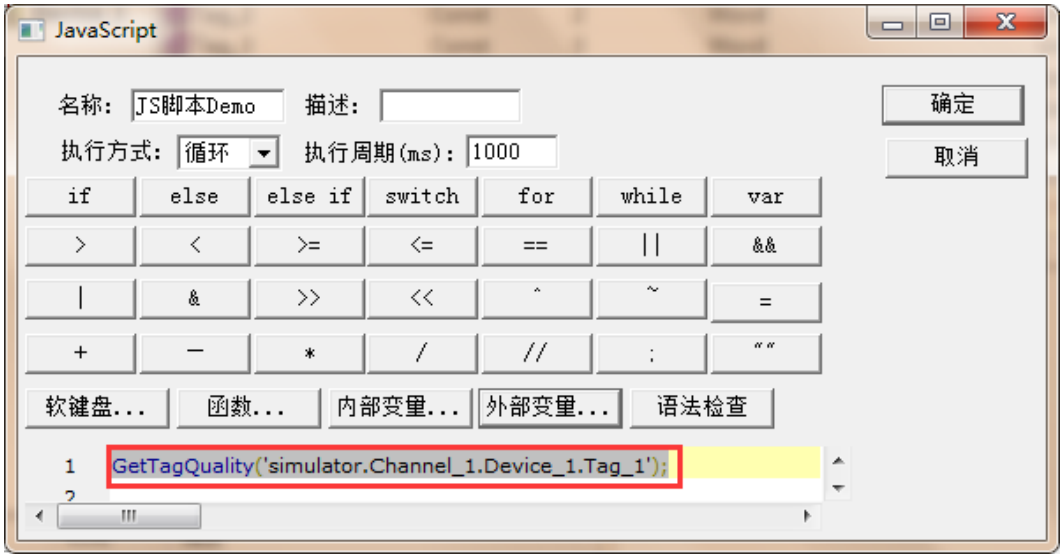
3.4 GetTagQuality(x)

3.4.1 函数功能介绍

GetTagQuality()函数是从变量点位里取质量戳。返回的结果符合 OPC 规范。即 Good 返回 192，Bad 返回 0，Uncertain 为 64，表示值未赋值，尚未被更新。

参数：x 只能填写网关的内部变量/外部变量点名称；不能直接填写用户自定义变量和数值。

3.4.2 函数操作举例



GetTagQuality()函数使用 Demo：

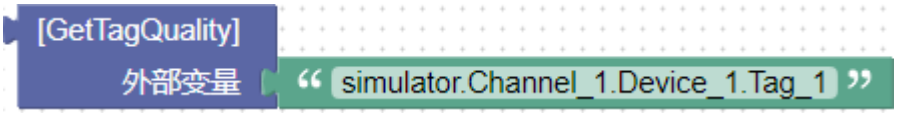
GetTagQuality('simulator.Channel_1.Device_1.Tag_1');

解析：

①变量 x: simulator.Channel_1.Device_1.Tag_1;

②读取变量 x 的质量戳。

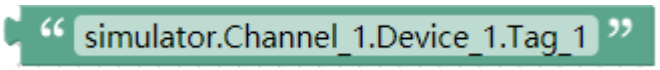
3.4.3 图形编程举例



步骤：

1. 选择工具箱-->取值-->GetTagQuality，点击文本-->  ，

2. 从 X2View 中拷贝点名粘贴到文本框中，如下：



，挂接在 GetTagQuality 块右侧。

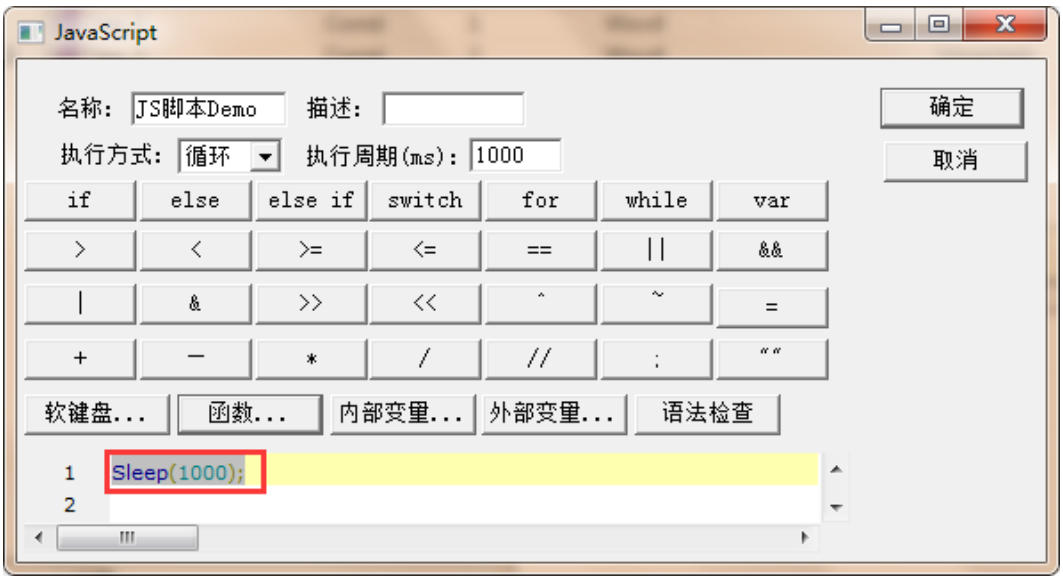
3.5 Sleep(x)

3.5.1 函数功能介绍

Sleep(x)函数，用于延迟，单位毫秒。

参数:x 可以为数值、自定义变量，不能直接填写用户自定义变量和数值。

3.5.2 函数操作举例



Sleep()函数使用 Demo：

```
Sleep(1000);
```

解析：

延时 1000ms。

3.5.3 图形编程步骤

步骤：

1. 选择工具箱-->定时器-->Sleep,点击数学--> 

,创建参数 

，然后挂接到 Sleep 右侧。

3.6 SetTimingGroup(x,y)

3.6.1 函数功能介绍

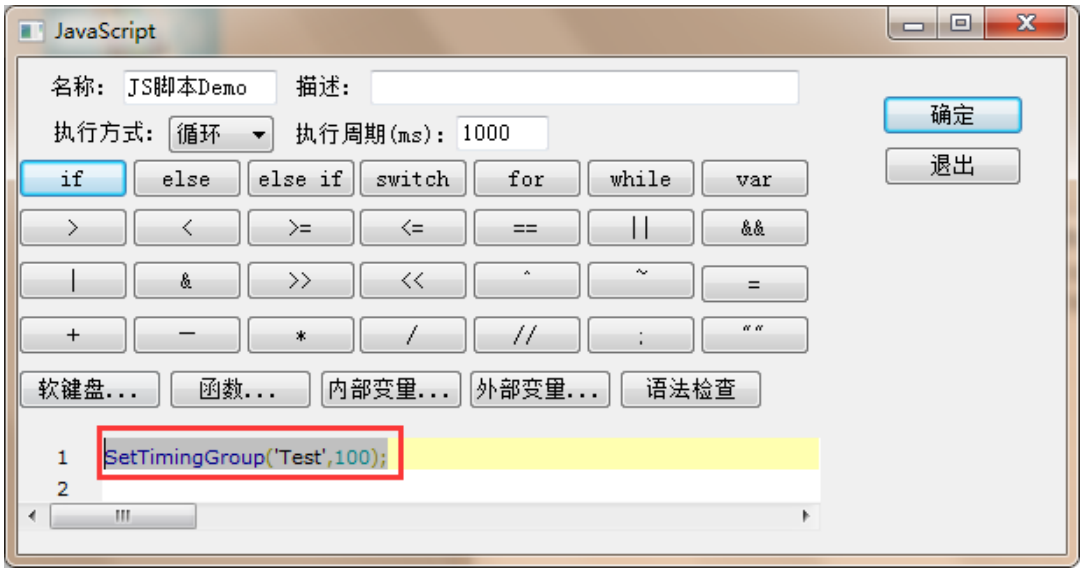
SetTimingGroup()函数，是对触摸屏/组态网关中设定的定时器组进行设置，可以通过这个函数操作你设备中建立的定时器组。

参数：

x 只能为定时器组名称；不能直接填写用户自定义变量和数值。

y 不能直接填写网关的内部变量/外部变量点名称，可以为数值，自定义变量。

3.6.2 函数操作举例



SetTimingGroup()函数使用 Demo：

SetTimingGroup('Test',100);

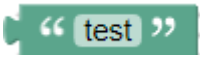
解析：

①定时器组：Test

②将数值 100 写入定时器组，即定时器组设定值为 100

3.6.3 图形编程举例

步骤：

1. 选择工具箱-->定时器-->SetTimingGroup，点击文本-->  ，创建  。

2. 通过工具箱-->数学-->  ，创建  。

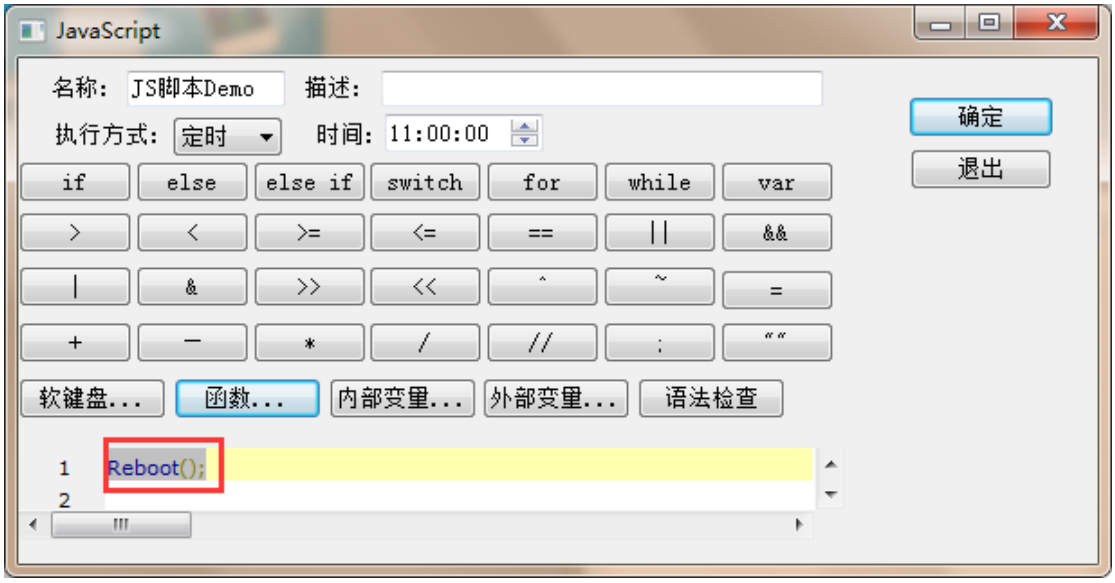
3. 将 1、2 中创建的块挂接到 SetTimingGroup 上。

3.7 Reboot()

3.7.1 函数功能介绍

Reboot()函数，是对触摸屏/组态网关进行定时的系统重启或者是间隔性循环重启。

3.7.2 函数操作举例



Reboot()函数使用 Demo：

Reboot();

解析：

直接调用 Reboot()函数内部不需要填写任何的参数。

3.7.3 图形编辑步骤

步骤

1. 工具箱-->系统--> 

。

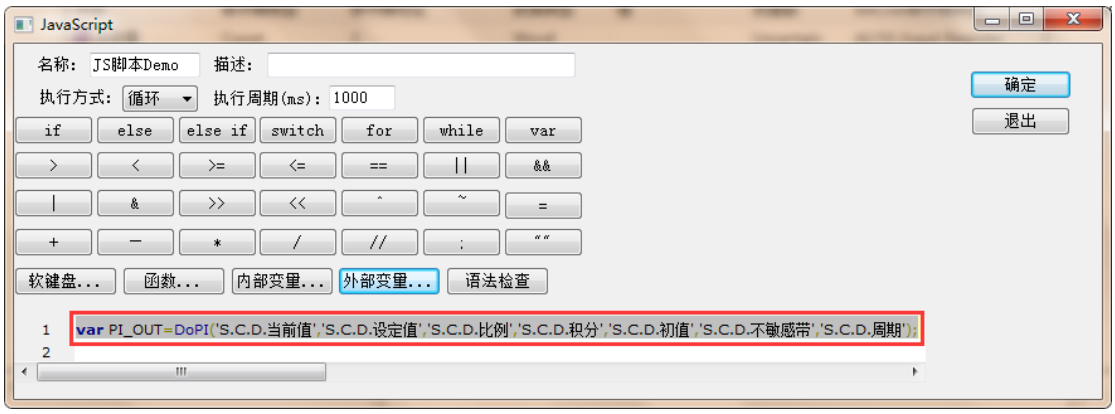
3.8 DoPI()

3.8.1 函数功能介绍

DoPI ()函数是 PI 操作指令，设定相应的参数可以进行 PI 运算处理。

参数：只能填写网关的内部变量/外部变量点名称；不能直接填写用户自定义变量和数值。

3.8.2 函数操作举例



DoPI ()函数使用 Demo：

var PI_OUT=DoPI('S.C.D.当前值','S.C.D.设定值','S.C.D.比例','S.C.D.积分','S.C.D.初值','S.C.D.不敏感带','S.C.D.周期');

解析：

①变量 A：S.C.D.当前值；

②变量 B：S.C.D.设定值；

③变量 C：S.C.D.比例；

④变量 D：S.C.D.积分；

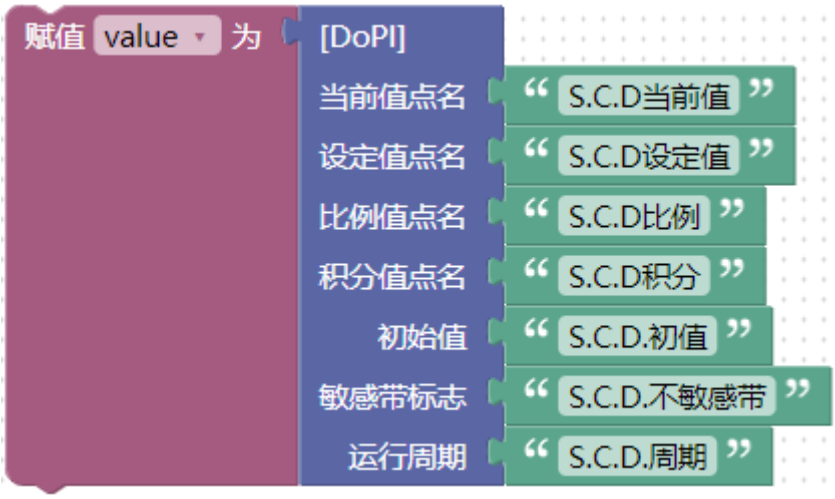
⑤变量 E：S.C.D.初值；

⑥变量 F：S.C.D.不敏感带；

⑦变量 G：S.C.D.周期；

⑧将 DoPI( )执行后的输出赋值给 JS 脚本编辑器中定义的 PI_OUT。

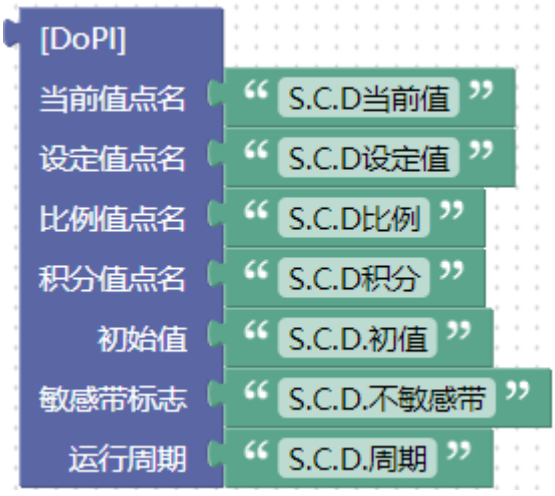
3.8.3 图形编程举例



步骤：

1. 选择工具箱-->变量-->创建变量，输入 PI_OUT，点击确认。

2. 选择工具箱-->控制-->DoPI，点击文本-->  ,创建 7 个变量，挂在 DoPI 模块右侧，如下：



3. 选择工具箱-->变量-->“赋值...为”， 点击下拉框选择 PI_OUT 变量，将 DoPI 模块挂在右侧。

说明：DoPI 有多个块，方便用户通过下拉框的形式选择参数。

3.9 SendWeChat(x,y)

3.9.1 函数功能介绍

SendWeChat(x,y)函数是微信报警接口的调用函数。可以通过 SendWeChat（x,y）函数实现数据定时的推送到微信上。

参数：

x 只能为用户组名称；不能直接填写用户自定义变量和数值。

y 不能直接填写网关的内部变量或外部变量点名称，可以为数值，自定义变量和编辑的文本。

3.9.2 函数操作举例

SendWeChat(x,y)函数使用 Demo：

var value=ReadFromTag('S.C1.D1.当前产量');

SendWeChat('Alarm','当前产量：'+value);

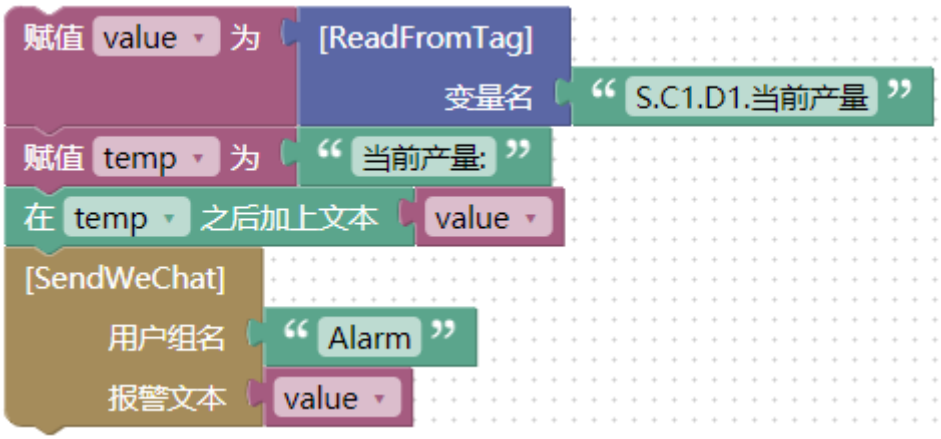
解析：

①变量 A：S.C1.D1.当前产量；

②用户组：Alarm

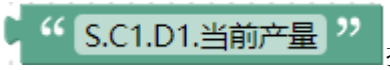
③报警推送文本：’当前产量：’+value（此格式为字符串拼接，也可自定义为所需的报警文本）

3.9.2 图形编程举例



步骤：

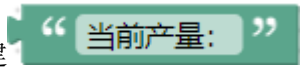
1. 选择工具箱-->变量-->创建变量，输入 value,点击确认。

2. 选择工具箱-->取值-->ReadFromTag，点击文本-->  ，创建  挂接在 ReadFromTag 右侧，如下：

3. 选择工具箱-->变量-->“赋值...为”，从下拉框中选择 value 变量，将步骤 3 中的模块挂接在右侧，如下：

4. 创建变量 temp,方法同创建 value。

5. 选择工具箱-->变量-->“赋值...为”，从下拉框中选择 temp 变量；点击文本 --> 

，创建 

，挂接到变量 temp 右侧，如下：

6. 选择工具箱-->文本--> 

，点击 item，在下拉框中选择 temp,然后将 value 变量挂在右侧，如下：

7. 选择工具箱-->报警-->SendWeChat,通过工具箱-->文本--> 

，创建 

，将“Alarm”和 temp 变量挂接在 SendWeChat 上，如下：

8. 将 4、6、7、8 中块自上而下连接在一起。

3.10 SendEMail(x,y)

3.10.1 函数功能介绍

SendEMail(x,y)函数是邮件报警接口的调用函数。可以通过 SendEMail（x,y）函数实现数据定时的推送到指定邮箱。

参数：

x 只能为邮箱地址；不能直接填写用户自定义变量和数值。

y 不能直接填写网关的内部变量或外部变量点名称，可以为数值，自定义变量和编辑的文本。

3.10.2 函数操作举例

SendEMail(x,y)函数使用 Demo：

var value=ReadFromTag('S.C1.D1.当前产量');

SendEMail('17740825170@163.com','当前产量：'+value);

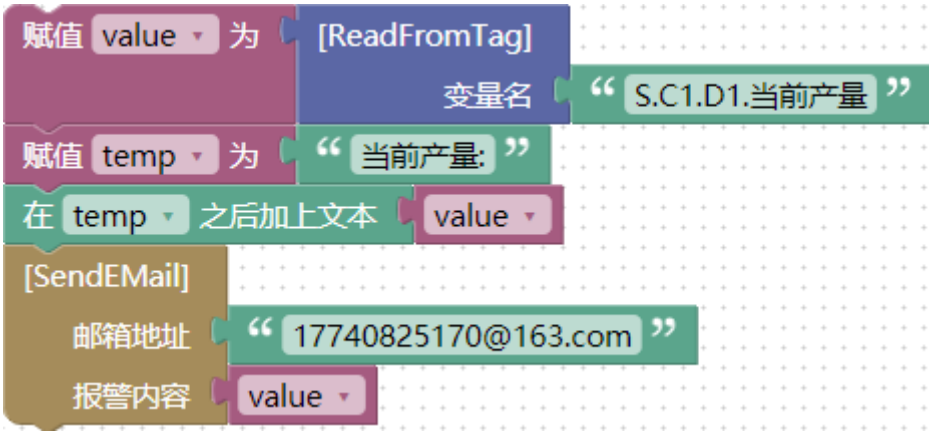
解析：

①变量 A: S.C1.D1.当前产量；

②邮箱：17740825170@163.com

③报警推送文本：'当前产量：'+value（此格式为字符串拼接，也可自定义为所需的报警文本）

3.10.3 图形编程举例



步骤：

参考 3.9.3 章节

3.11 SendPhone(x,y)

3.11.1 函数功能介绍

SendPhone（x, y）函数是电话报警接口的调用函数。可以通过 SendPhone（x, y）函数实现报警信息电话语音播报给指定手机的用户。

参数：

x 只能为电话号码；不能直接填写用户自定义变量和数值。

y 不能直接填写网关的内部变量或外部变量点名称，可以为数值，自定义变量和编辑的文本。

3.11.2 函数操作举例

SendPhone(x,y)函数使用 Demo：

var value=ReadFromTag('S.C1.D1.当前产量');

SendPhone('17740825170','当前产量：'+value);

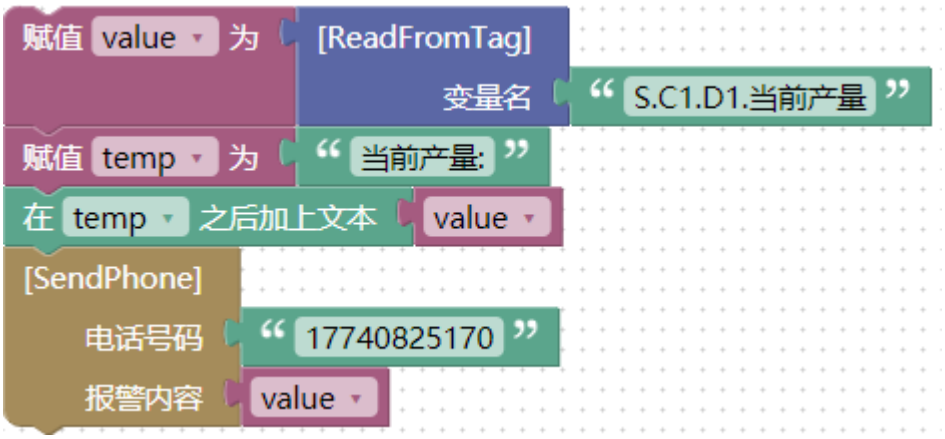
解析：

①变量 A: S.C1.D1.当前产量；

②手机号码：17740825170

③报警推送文本：'当前产量：'+value（此格式为字符串拼接，也可自定义为所需的报警文本）

3.11.3 图形编程举例



步骤：

参考 3.9.3 章节

3.12 SendSMSText(x,y)

3.12.1 函数功能介绍

SendSMSText(x,y)函数是短信报警接口的调用函数。可以通过 SendSMSText（x,y）函数实现数据以短信形式定时的推送到指定手机用户。

参数：

x 只能为电话号码；不能直接填写用户自定义变量和数值。

y 不能直接填写网关的内部变量或外部变量点名称，可以为数值，自定义变量和编辑的文本。

3.12.2 函数操作举例

SendSMSText(x,y)函数使用 Demo：

var value=ReadFromTag('S.C1.D1.当前产量');

SendSMSText('17740825170','当前产量：'+value);

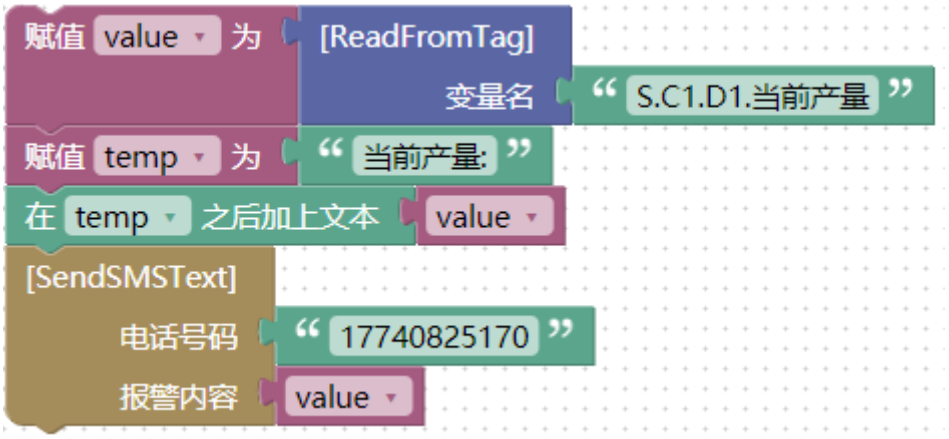
解析：

①变量 A: S.C1.D1.当前产量；

②手机号码：17740825170

③报警推送文本：'当前产量：'+value（此格式为字符串拼接，也可自定义为所需的报警文本）

3.12.3 图形编程举例



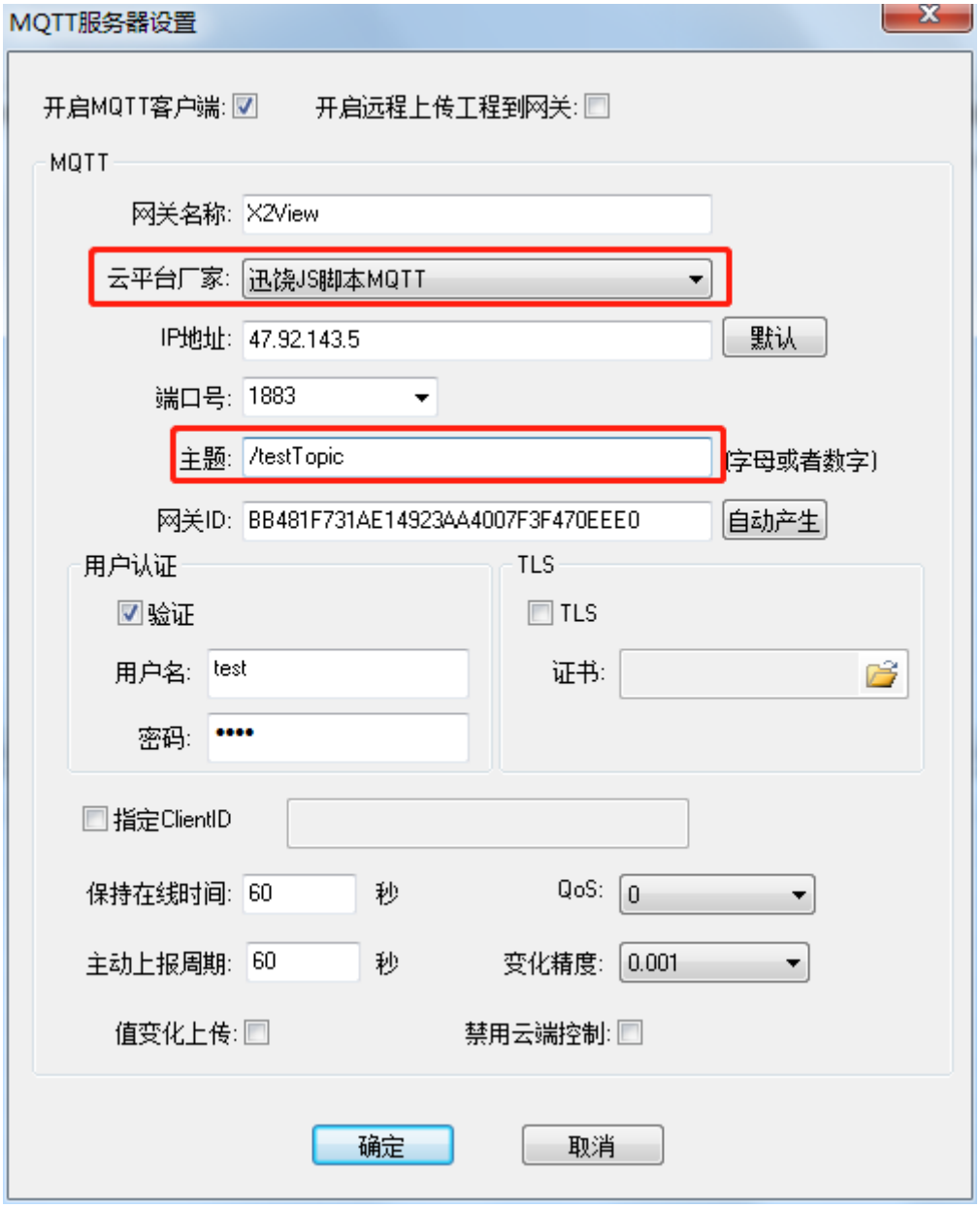
步骤：

参考 3.9.3 章节

3.13 GetMQTTTopic()、GetMQTTMessage()、MQTTPublish(x,y)

对于 GetMQTTTopic()、GetMQTTMessage()、MQTTPublish(x,y)为 MQTT 相关的函数，使用上述函数，必须填写 MQTT 参数，连接信息如下，云平台厂家必须选择“迅饶 JS 脚本 MQTT”，订阅的主题为用户填写的主题，此处以“/testTopic”，如需要订阅多个主题，可用逗号“，”隔开，比如：aaa,bbb。表示同时订阅 aaa 和 bbb 主题。

举例，其他用户名、密码等参数根据连接 MQTT 实际需要填写。



3.13.1 函数功能介绍

GetMQTTTopic()函数获取订阅的 MQTT 主题，无参数，返回值 String。当 MQTT 服务器推送数据到网关订阅的主题时，获取当前订阅主题，返回值为 String。

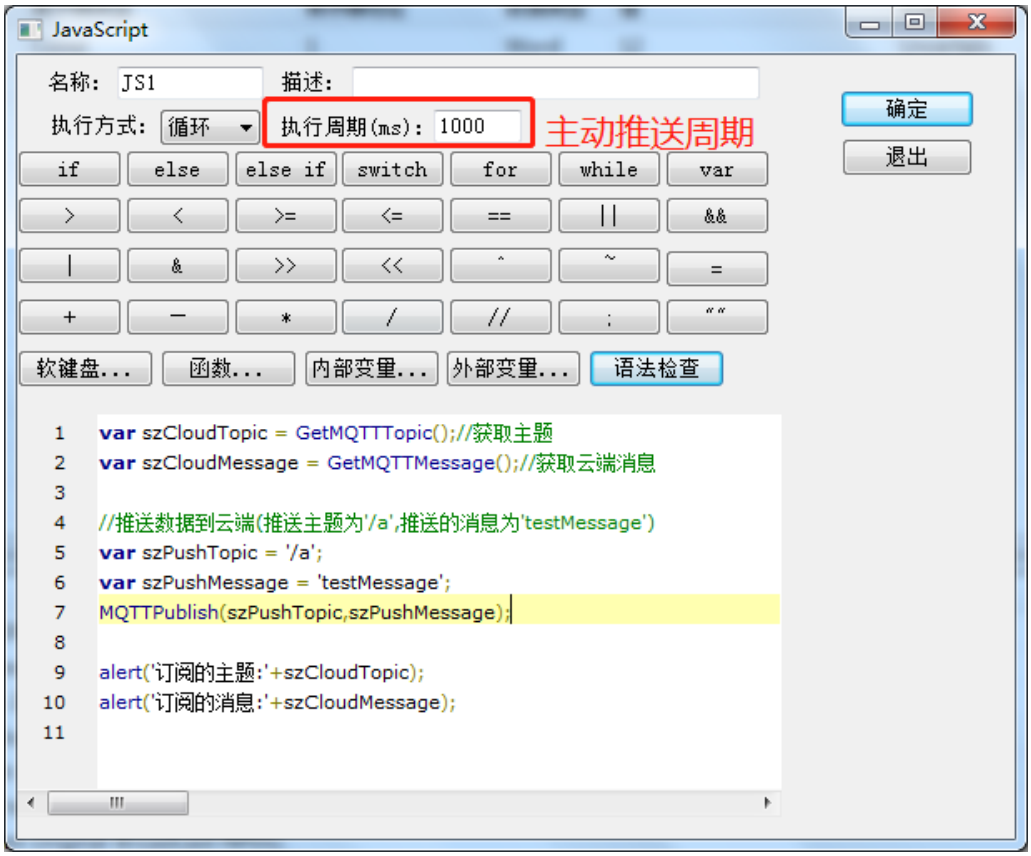
GetMQTTMessage()函数获取订阅的MQTT主题的消息，无参数，返回值String。当 MQTT 服务器推送数据到网关订阅的主题时，获取当前订阅主题的消息，返回值为 String。

MQTTPublish(x,y)函数根据用户定义主题推送消息到 MQTT 平台，无返回值。参数：

x 为推送的主题；字符串。

y 为推送的消息内容；字符串

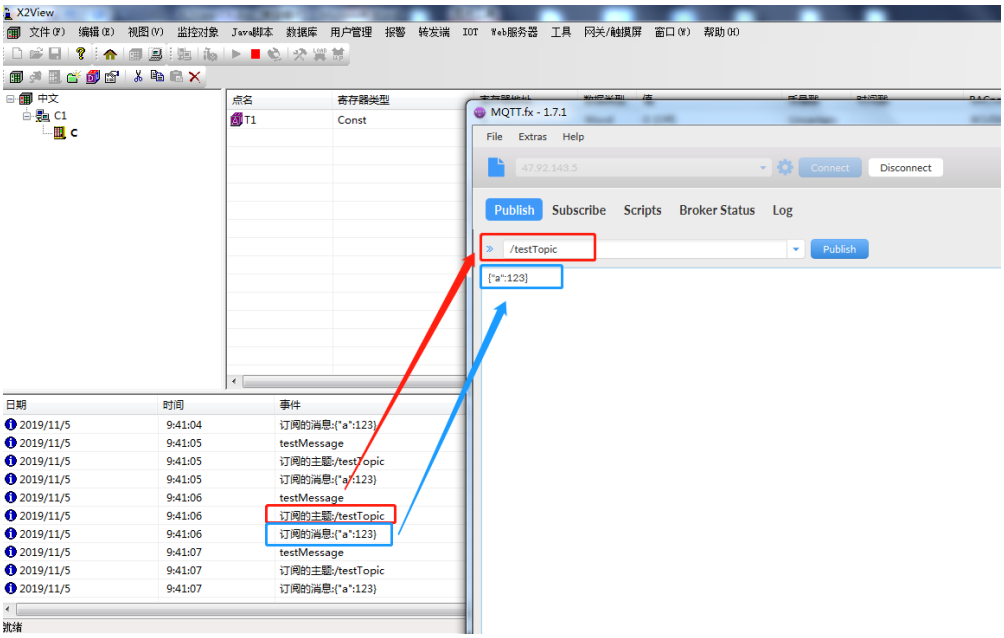
3.13.2 函数操作举例



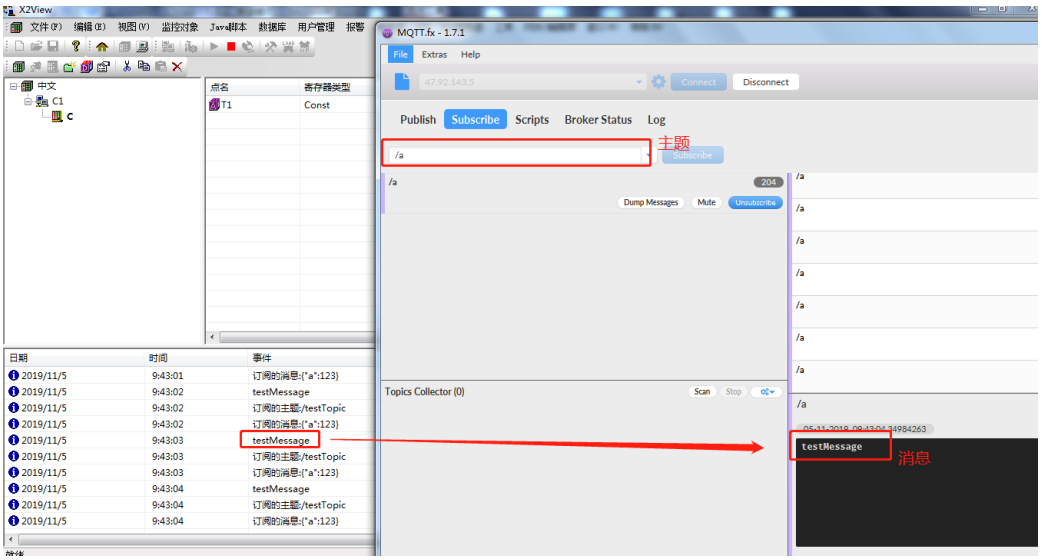
函数使用 Demo：

```
var szCloudTopic = GetMQTTTopic();//获取主题
var szCloudMessage = GetMQTTMessage();//获取云端消息
//推送数据到云端(推送主题为'/a',推送的消息为'testMessage')
var szPushTopic = '/a';
var szPushMessage = 'testMessage';
MQTTPublish(szPushTopic,szPushMessage);
```

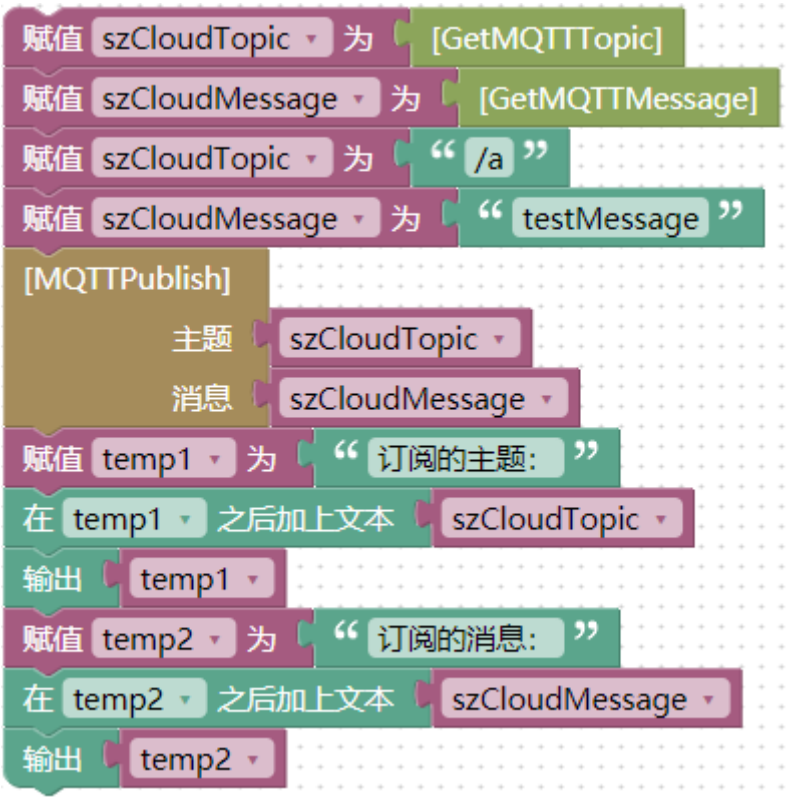
订阅云端的主题和消息实际运行效果：



推送数据到云端实际运行效果：



3.13.3 函数编程举例

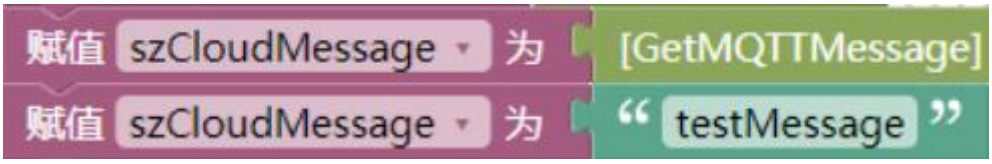


步骤：

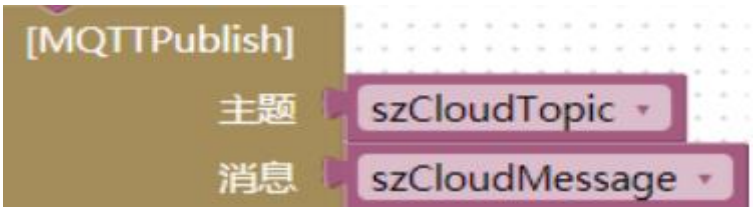
1. 选择工具箱-->变量-->创建变量，分别创建 szCloudTopic,szCloudMessage 变量。

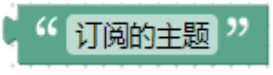
2. 选择工具箱-->MQTT-->GetMQTTTopic，挂接到变量 szCloudTopic 上，也可以文本-->" "，创建" /a "，挂在变量 szCloudTopic 上，如下：

3. 选择工具箱-->MQTT-->GetMQTTMessage，将其挂接到变量 szCloudMessage 上，也可以工具箱-->文本-->" "，创建" testMessage "，挂在变量 szCloudMessage 上，如下：



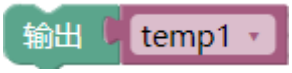
4. 选择工具箱-->MQTT-->MQTTPublish，将变量 szCloudTopic,szCloudMessage 分别挂在主题和消息位置，如下：



5. 选择工具箱-->变量-->创建临时变量 temp1，选择变量-->“赋值 temp1 为”，通过文本-->  ，创建  ，然后挂接到 temp1 右侧，如下：

6. 选择工具箱-->文本-->  ，点击 item 选择下拉框中的 temp1,然后将 szCloudTopic 挂在右侧，如下：

7. 选择工具箱-->文本-->  ，将 temp1 变量挂在右侧，如下：



8. temp2 操作方式同 temp1。

3.14 ReadFromTagEx(x,y)

3.14.1 函数功能介绍

ReadFromTagEx(x,y)函数可以读取迅饶其他设备内的通讯变量数据。

参数：

x 为对应设备的 IP 地址；不能直接填写用户自定义变量和数值。

y 为对应点名，点名结构遵循迅饶命名规则（驱动名称+通道名称+设备名称+组名+标签名称）。

3.14.2 函数操作举例

ReadFromTagEx(x,y)函数使用 Demo：

var Value=ReadFromTagEx("192.168.1.68",'s.C1.D1.40001');

解析：

①变量 x：192.168.1.68；

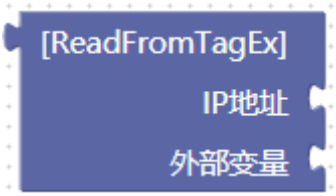
②变量 y：s.C1.D1.40001

3.14.3 图形编程举例

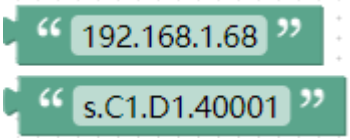
步骤：

1. 选择工具箱-->变量-->创建变量，输入 Value,点击确认。

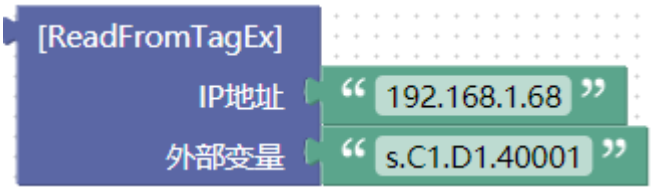
2. 选择工具箱-->取值-->ReadFromTagEx，创建模块，如下：



3. 选择工具箱-->文本-->  ，点击两次，创建两个字符串块，如下：



4. 将 3 中的模块挂在 2 中的 ReadFromTagEx 块的右侧，如下：



5. 选择工具箱-->变量-->“赋值...为”，从下拉框中选择 Value，将 ReadFromTagEx 模块挂在 Value 变量右侧,如下：

3.15 WriteToTagEx(x,y,z)

3.15.1 函数功能介绍

WriteToTagEx(x,y,z)函数可以控制迅饶其他设备内的通讯变量数据。

参数：

x 为对应设备的 IP 地址；不能直接填写用户自定义变量和数值。

y 为对应点名，点名结构遵循迅饶命名规则（驱动名称+通道名称+设备名称+组名+标签名称）。

z 不能直接填写网关的内部变量/外部变量点名称，可以为数值，自定义变量。

3.15.2 函数操作举例

WriteToTagEx(x,y,z)函数使用 Demo：

WriteToTagEx("192.168.1.68",'s.C1.D1.40001',100);

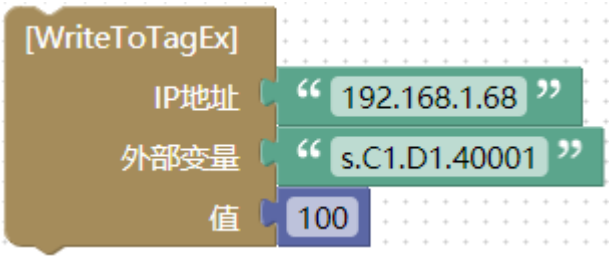
解析：

①变量 x：192.168.1.68；

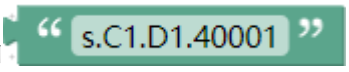
②变量 y：s.C1.D1.40001；

③变量 z：数值 100。

3.15.3 图形编程举例



步骤：

1. 工具箱-->赋值-->WriteToTagEx，通过文本-->  ，创建  和  ，通过数学-->  ，创建  ，然后挂接在 WriteToTagEx 上。

说明：WriteToTagEx 有多个块，方便用户通过下拉框的形式选择参数。

3.16 SendTextToPort(a, b, c, d, e, f, g)

3.16.1 函数功能介绍

SendTextToPort (a, b, c, d, e, f, g)函数是用户用来操作网关串口的方法，通过此方法，用户自己就可以编程通讯，读取一些仪表设备的数据。

参数：

a 为对应网关的 COM 口 Number 号，不能与驱动里正常通讯使用的串口号冲突。

b 为 com 口通讯使用的波特率，比如 38400, 19200, 9600, 4800, 2400, 1200 等

c 为 com 口通讯使用的数据位（8 或者 7）。

d 为 com 口通讯使用的停止位（1 或者 2）。

e 为 com 口通讯使用的校验位（0, 1, 2 分别代表无校验，奇校验，偶校验）。

f 为 com 口通讯等待超时时间（单位 ms）。

g 为 com 口发送内容，内容可以是字符串，或者用十进制或者十六进制数组表示。

3.16.2 函数操作举例

SendTextToPort(a,b,c,d,e,f,g)函数使用 Demo：

var mes=SendTextToPort(3,9600,8,1,0,1000,send);解析：

①变量 a:com3。

②变量 b：9600 波特率。

③变量 c：8 数据位。

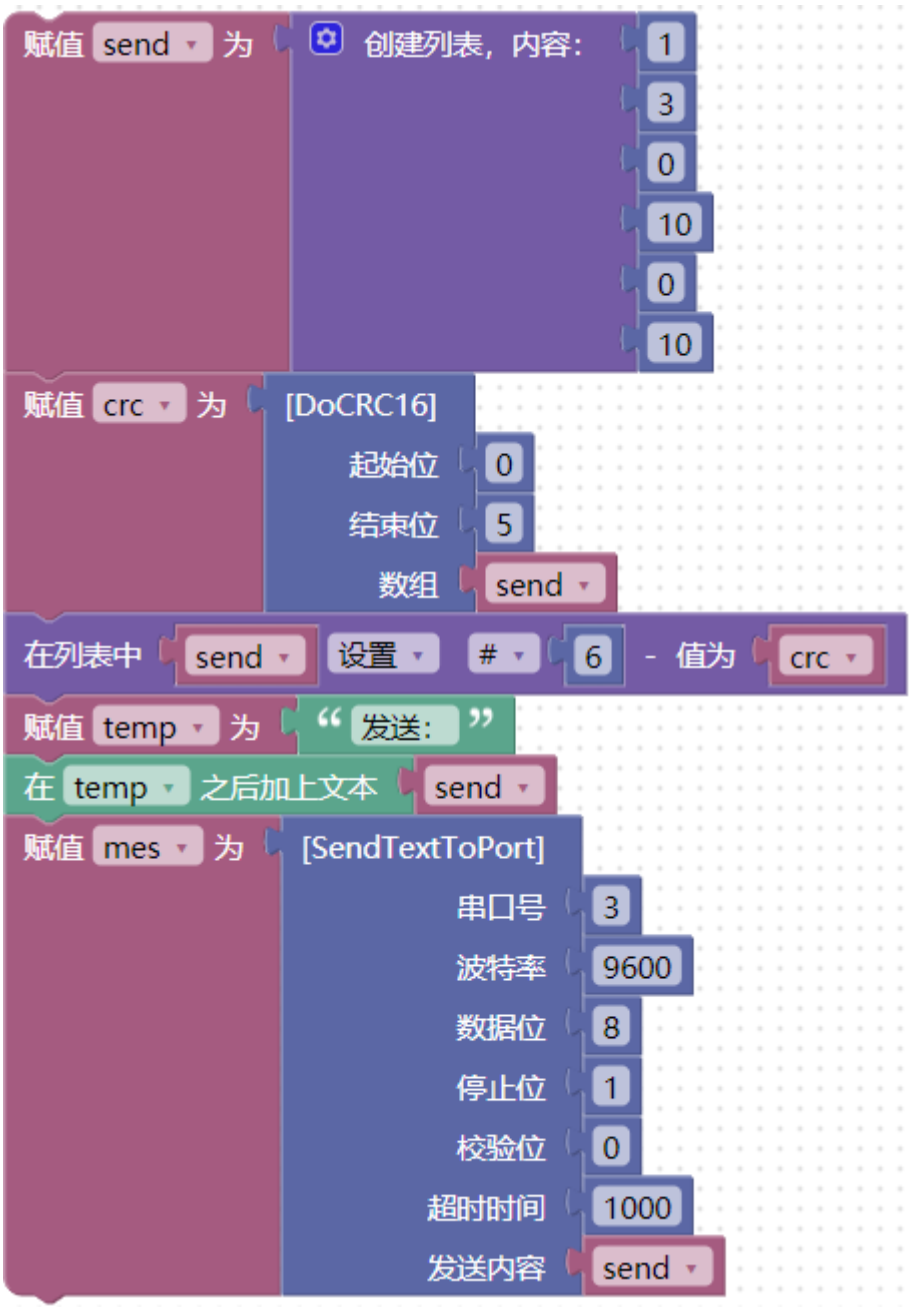
④变量 d：1 停止位。

⑤变量 e：无校验。

⑥变量 f：超时时间 1000ms。

⑦变量 g：发送数组 send。

3.16.3 图形编程举例



步骤：

1. 选择工具箱-->变量-->创建变量，输入 send 确定。

2. 选择工具箱-->列表--> 

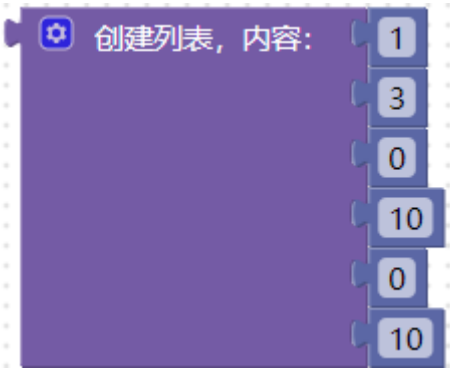
，点击设置图标

，

将项目拖入右侧的列表中，连拖 6 次，创建 6 个元素的数组，如下：

通过数学--> 

，给列表赋值，点击设置图标关闭小窗体，如下：



通过变量-->“赋值...为”，将该列表挂在 send 变量上，如下：

（可以通过“赋值...为”选择需要的变量）

3. 选择工具箱-->变量-->创建变量，输入 crc，点击确认。

4. 选择工具箱-->校验-->DoCRC16 函数，点击数学--> 

，创建 

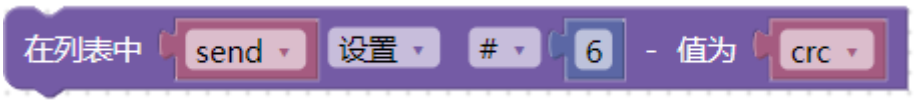
和 

，将这两个块和 send 变量挂在 DoCRC16 上，然后点击变量-->“赋值...为”，选择下拉框中的 crc 变量进行赋值，如下：

5. 选择工具箱-->列表--> 

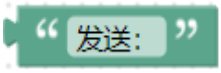
，点击 item，在下拉框中选择 send 变量,点击数学--> 

，将第一个参数设置为 6（代表数组第 5 个元素，从 0 开始）；点击变量-->crc，将 crc 设置为数组的第 5 个元素的值，如下：



。

6. 选择工具箱-->变量-->创建变量，输入 temp，点击确认。

7. 选择工具箱-->变量--> “赋值...为”，选择 temp,点击文本--> 
,创建 

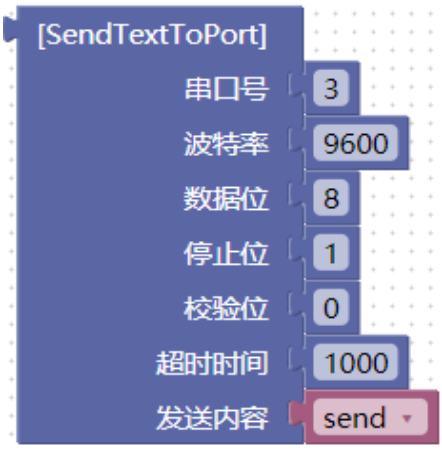
，对 temp 进行赋值，如下：

8. 选择工具箱-->文本--> 

，点击变量-->send,然后将 send 变量拖入该块中，如下：

9. 选择工具箱-->报文-->SendTextToPort，点击数学--> 

，创建 6 个整数，将 send 变量一起挂在 SendTextToPort，如下:



10. 选择工具箱-->变量-->创建变量，输入 msg,点击确认；点击变量-->“赋值...为”，选择 msg,然后将 SendTextToPort 块挂在 msg 变量上。

11. 将 2、4、5、7、8、9、10 自上而下堆叠在一起

3.17 MakeFloat(a, b, c, d)

3.17.1 函数功能介绍

MakeFloat(a,b,c,d),MakeDouble(a,b,c,d,e,f,g,h),MakeLong(a,b,c,d),MakeWord(a,b)等都是数据处理函数，这里只介绍 MakeFloat()。

参数：

a,b,c,d 依次为 IEE754 单精度 32 位 Float 数据的 4 个 Byte 字节。需要注意 4 个字节传参的顺序，JS 内部编译默认按照低字节在前，高字节在后顺序解析。

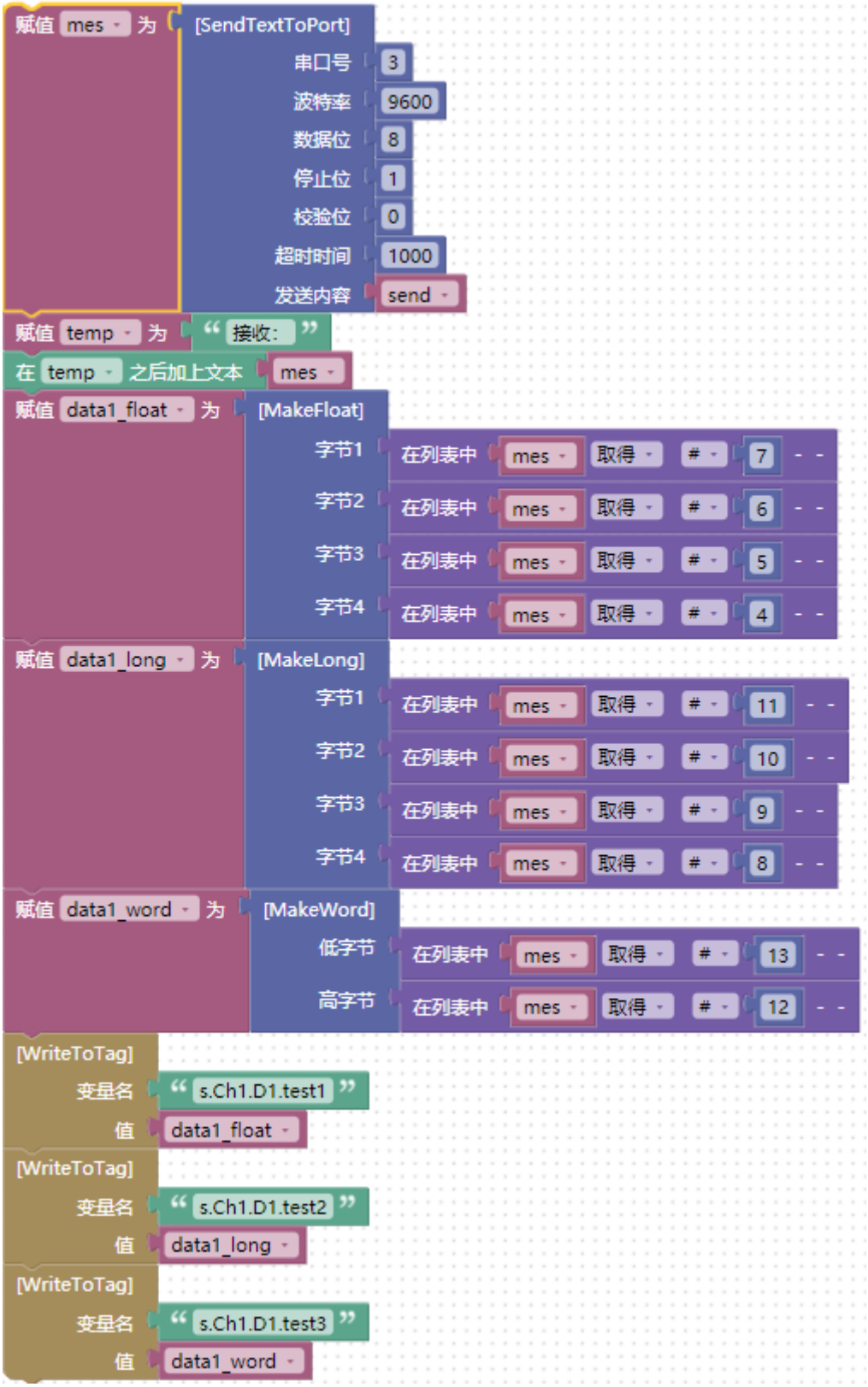
3.17.2 函数操作举例

MakeFloat(a,b,c,d)函数使用 Demo：

var data1_float=MakeFloat(mes[6],mes[5],mes[4],mes[3]);

解析：将 mse[]数组的 3~6 号元素按照单精度 Float 算法计算出当前值。

3.17.3 图形编程举例



步骤：

1. SendTextToPort 用法参照 3.16.3。

2. 选择工具箱-->字节-->MakeFloat，创建 MakeFloat 块

点击列表--> 

，读取数组的某一项值，点击 item,从下拉框中选择 mes，创建 4 个这样的块，分别读取第 7、6、5、4 项（数组索引对应 6、5、4、3）如下：

3. MakeLong、MakeWord 与 MakeFloat 操作相同

4. WriteToTag 参考 3.3.3

3.18 ExecuteSQL (x)

3.18.1 函数功能介绍

对于 ExecuteSQL(x)为执行 MySQL，SQLServer 语句的脚本函数，使用上述函数，可以根据自身编写 MySQL，SQLServer 语句进行操作数据库，配合 JS 脚本使用，可以实现循环执行语句、定时、变化执行 SQL 语句。

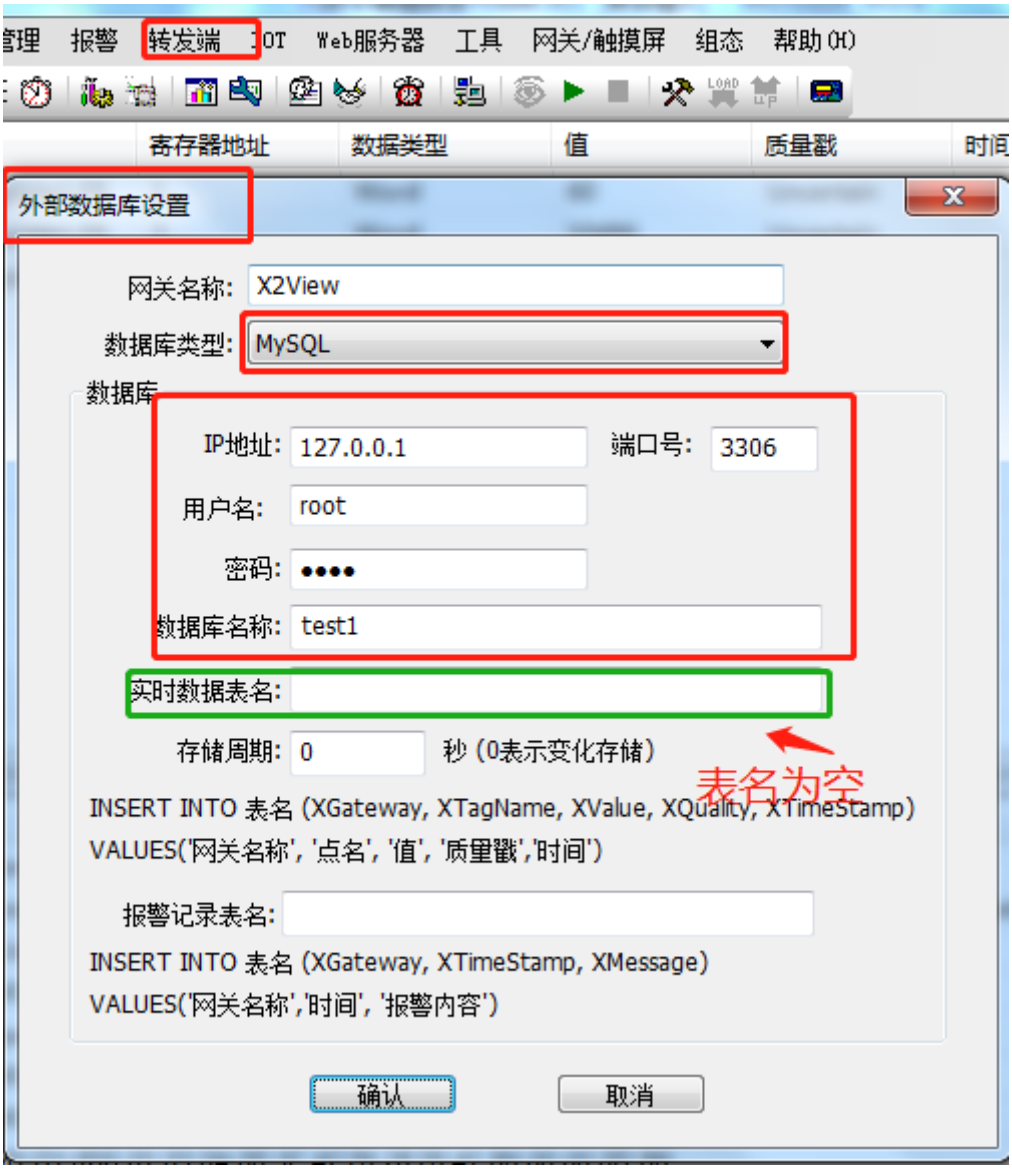
x 为需要执行的 SQL 语句，字符串。

如 UPDATE test SET V1=11,V2=22,V3=33 where id = 1。

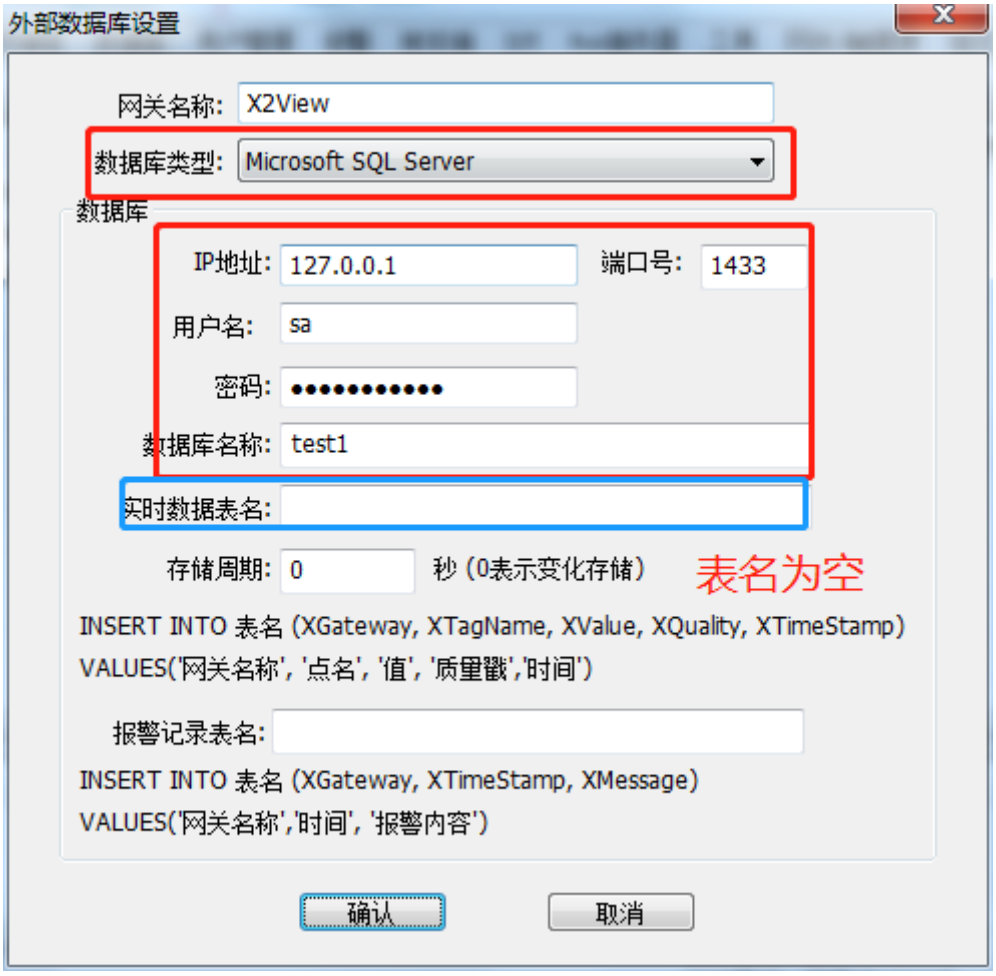
【注意】使用此功能之前，必须先配置好 X2View 转发端的 MySQL 或者 SQLServer 设置，进行连接 MySQL 或者 SQLServer 的参数配置，而且操作数据库之前，首先要在 MySQL 或者 SQLServer 中把需要操作的字段定义好，不然命令会执行失败。

如下图所示：

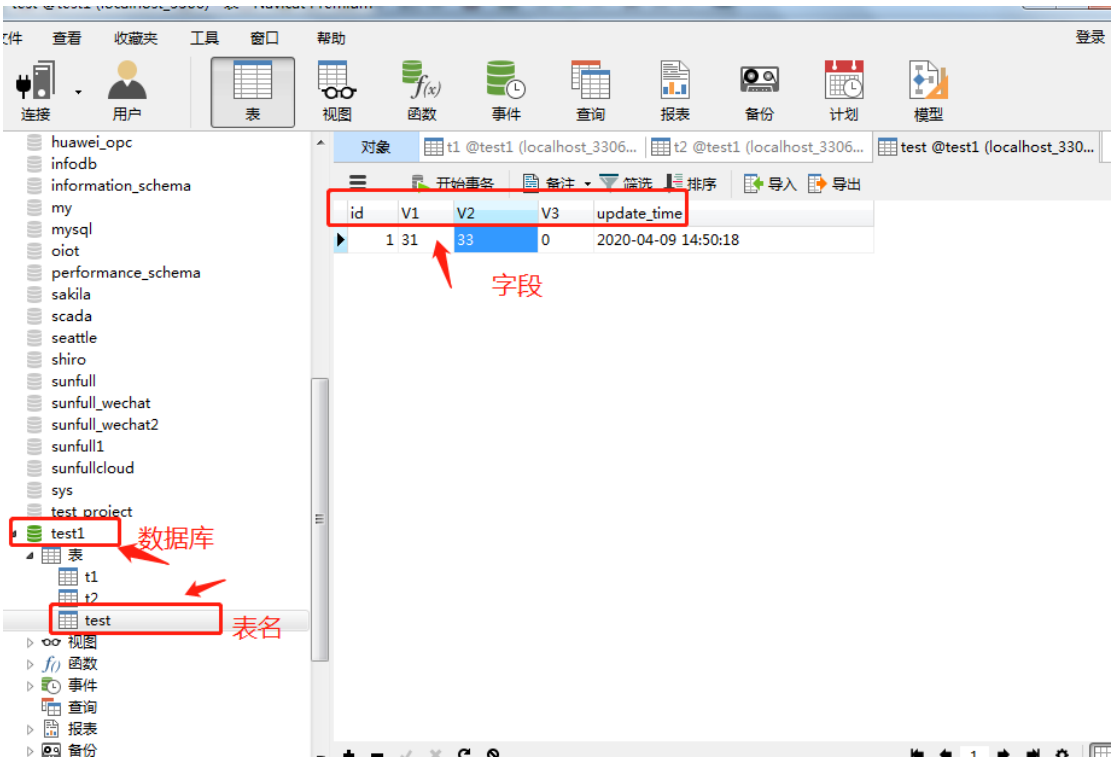
MySQL X2View 设置：



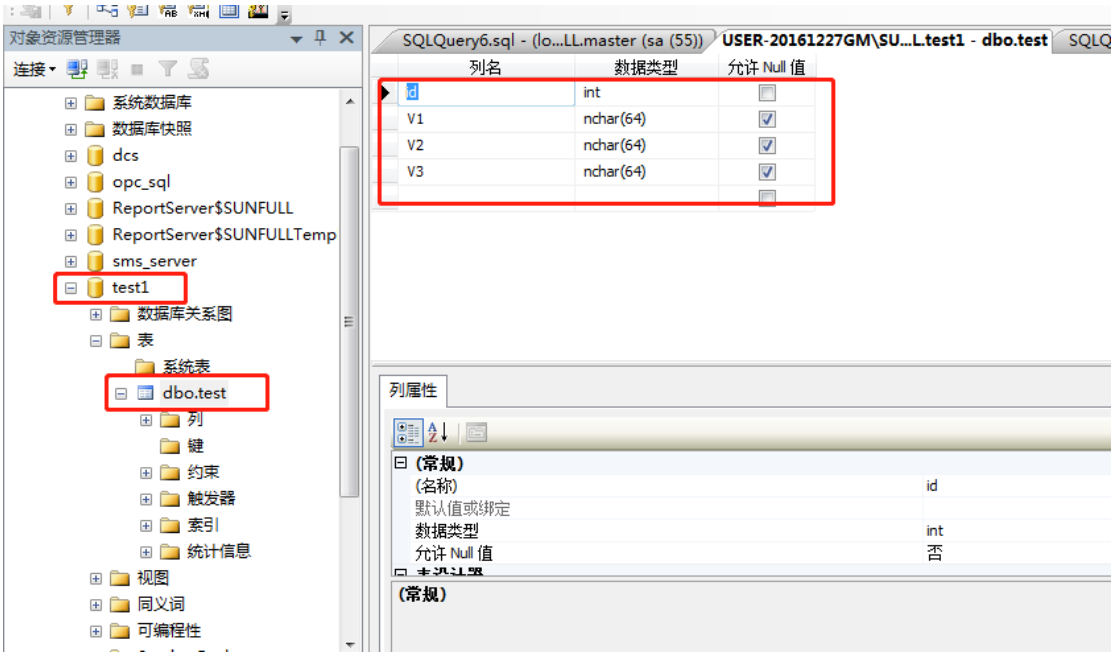
SQLServer X2View 设置



MySQL 字段设置：



SQLServer 字段设置：

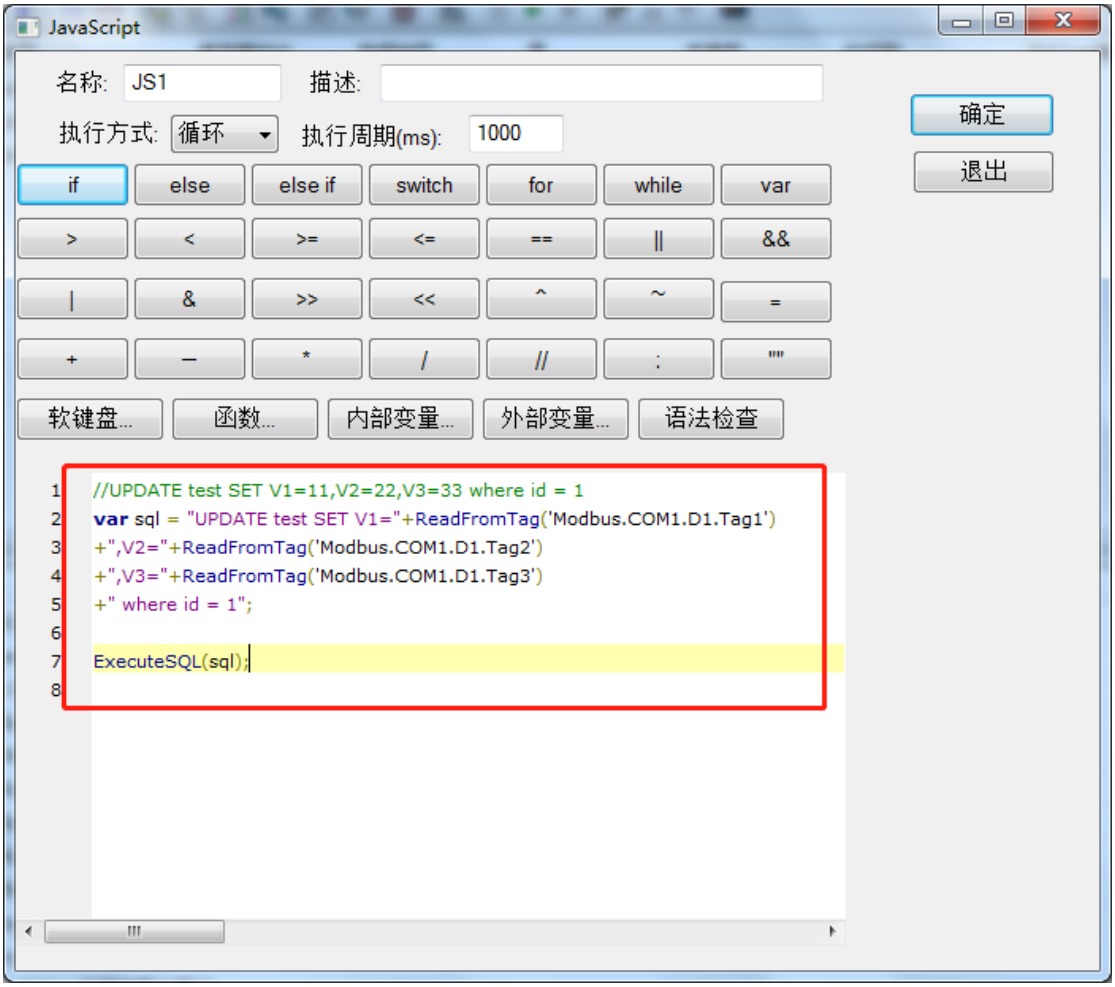


以下脚本以 MySQL 举例。

对应填写需要操作的数据库 IP 地址，端口号，数据库名称，数据库表名为空。

此处以操作数据库名 test1 为例.

3.18.2 函数操作举例



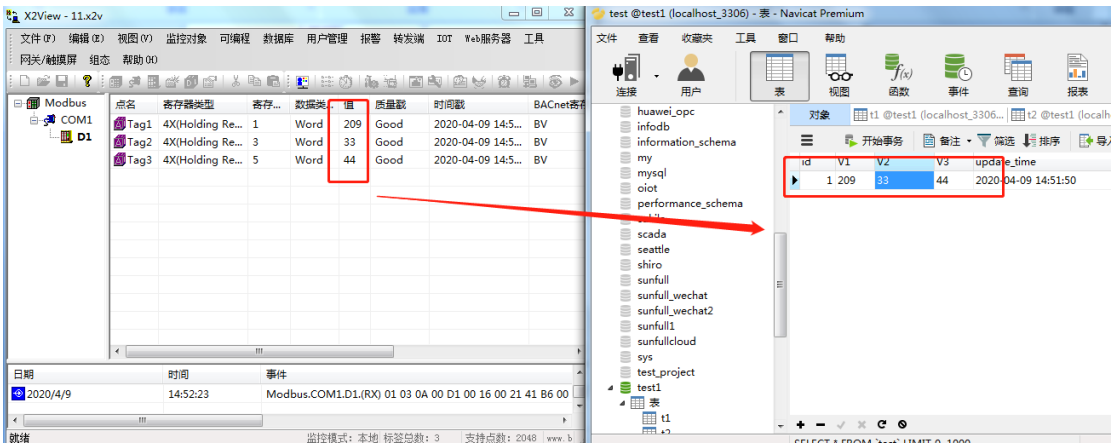

```
//UPDATE test SET V1=11,V2=22,V3=33 where id = 1
var sql = "UPDATE test SET V1="+ReadFromTag('Modbus.COM1.D1.Tag1')
+",V2="+ReadFromTag('Modbus.COM1.D1.Tag2')
+",V3="+ReadFromTag('Modbus.COM1.D1.Tag3')
+" where id = 1";
ExecuteSQL(sql);
```

解释：循环，每秒操作数据库 test1,数据库表 test，给 V1,V2,V3 字段更新数据。

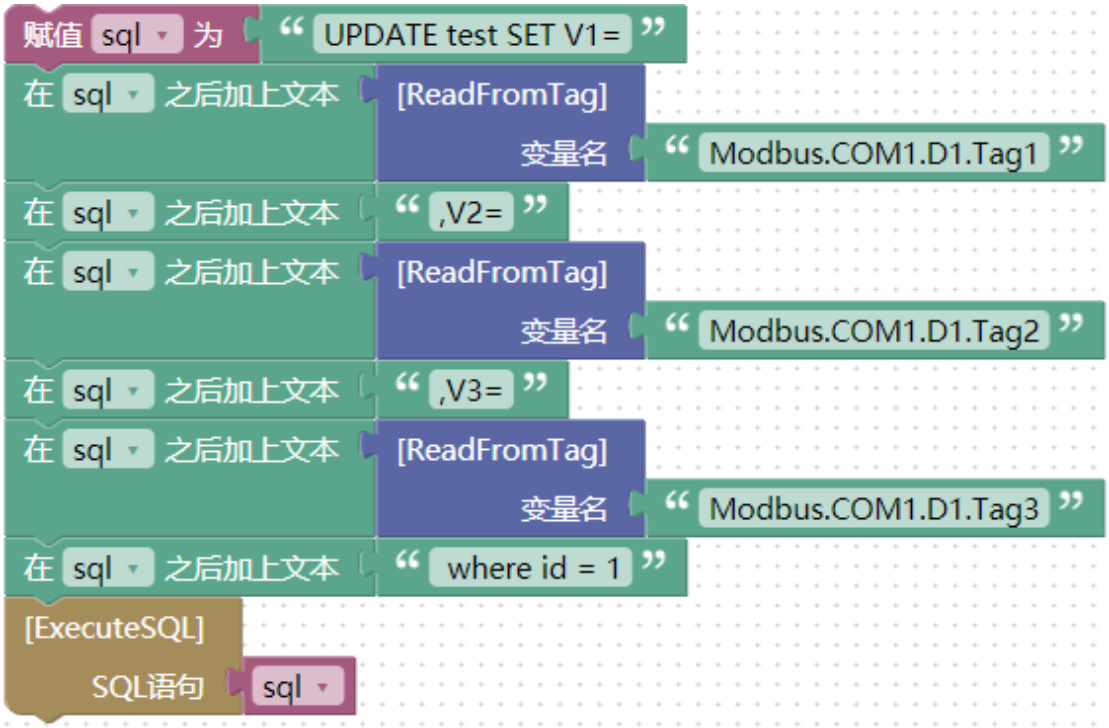
【注意】此处关键在于 sql 语句的拼接。

MySQL 数据库设置如下图：

实际结果：



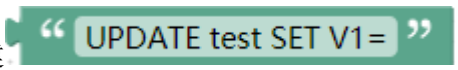
3.18.3 图形编程举例

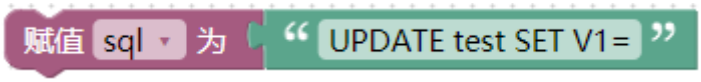


步骤：

1. ReadFromTag 参照 3.1.3。

2. 选择工具箱-->变量-->创建变量，输入 sql，点击确认。

3. 选择工具箱-->文本-->  ，创建  ，初始化 sql，如下：



4. 选择工具箱-->文本-->  ，点击 item，在下拉框中选择 sql 变量，然后将 ReadFromTag 挂接到其后面，如下：

5. 选择工具箱-->文本-->  ，点击 item，在下拉框中选择 sql 变量，然后在后面填入字符串“,V2=”，如下：

6. 重复 4、5 步骤，完成 sql 语句拼接。

7. 选择工具箱-->工具-->ExecuteSQL，将 sql 变量拖入 ExecuteSQL 的参数中。

3.19 SendTextToTCP(a, b, c, d)

3.19.1 函数功能介绍

SendTextToTCP (a, b, c, d)函数是用户用来操作网关网口的方法，通过此方法，用户自己就可以编程通讯，读取一些 TCP 通讯设备的数据。

参数：

a 为对应 TCP 设备的 IP 地址。

b 为对应 TCP 通讯设备的端口号。

c 为通讯等待超时时间（单位 ms）。

d 为发送内容，内容可以是字符串，或者用十进制或者十六进制数组表示。

3.19.2 函数操作举例


```
//ModbusTCP 通讯
var Arr_Read=new Array();     //定义数组
Arr_Read[0]=00;                          //包头起始
Arr_Read[1]=01;
Arr_Read[2]=00;
Arr_Read[3]=00;
Arr_Read[4]=00;
Arr_Read[5]=06;                          //包头结尾
Arr_Read[6]=01;                          //地址站号
Arr_Read[7]=03;                          //Modbus 功能码
Arr_Read[8]=00;                          //起始地址高字节
Arr_Read[9]=00;                          //起始地址低字节
Arr_Read[10]=00;                         //数据长度高字节
Arr_Read[11]=02;                         //数据长度低字节
var Arr_Received=SendTextToTCP('192.168.1.189',502,100,Arr_Read); //发送数据请求，接收报文为 Arr_Received 十进制数组中
WriteToTag('Simulator.Channel_1.TCP.40001',MakeWord(Arr_Received[10],Arr_Received[9]));
WriteToTag('Simulator.Channel_1.TCP.40002',MakeWord(Arr_Received[12],Arr_Received[11]));
```

SendTextToTCP (a,b,c,d)函数使用 Demo：

```
var mes=SendTextToTCP (a,b,c,d);
```

解析：

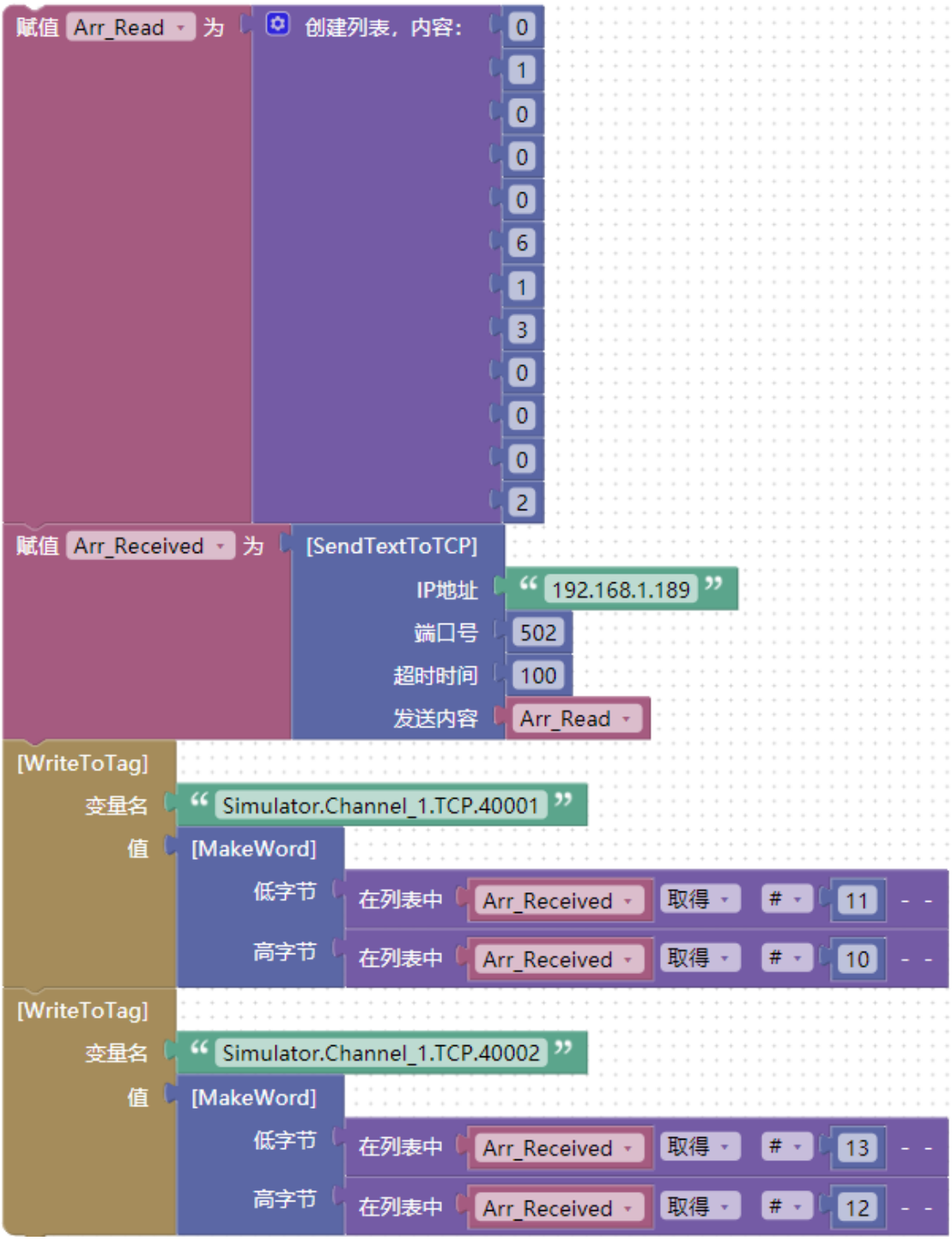
①变量 a：IP 地址’192.168.1.189’

②变量 b：端口号 502

③变量 c：超时时间 100ms

④变量 d：发送数组 Arr_Read

3.19.3 图形编程举例



步骤：

1. 数组创建方式参照 3.16.3

2. WriteToTag 参照 3.3.3

3. MakeWord 使用参照 3.17.3

4. SendTextToTCP 使用方式参照 3.16.3

3.20 SendTextToUDP(a, b, c, d)

3.20.1 函数功能介绍

SendTextToUDP (a,b,c,d)函数是用户用来操作网关网口的方法，通过此方法，用户自己就可以编程通讯，读取一些 UDP 通讯设备的数据。

参数：

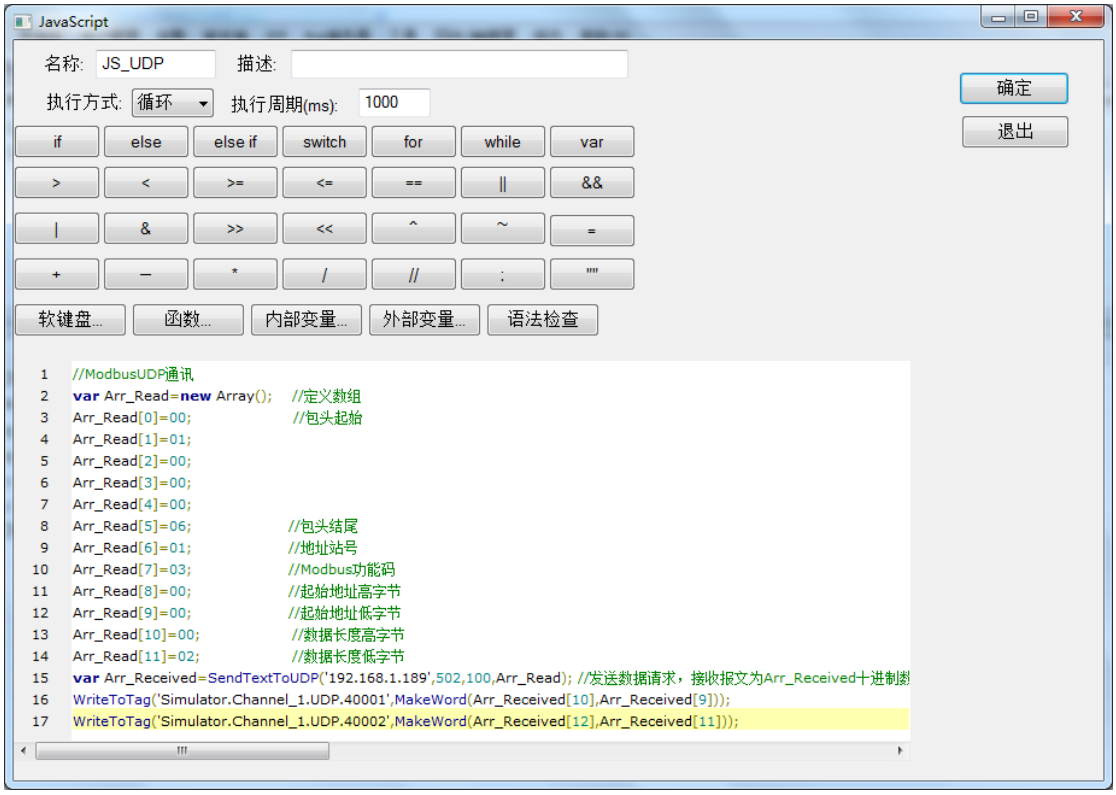
a 为对应 UDP 设备的 IP 地址。

b 为对应 UDP 通讯设备的端口号。

c 为通讯等待超时时间（单位 ms）。

d 为发送内容，内容可以是字符串，或者用十进制或者十六进制数组表示。

3.20.2 函数操作举例




```
//ModbusUDP 通讯
var Arr_Read=new Array();     //定义数组
Arr_Read[0]=00;                          //包头起始
Arr_Read[1]=01;
Arr_Read[2]=00;
Arr_Read[3]=00;
Arr_Read[4]=00;
Arr_Read[5]=06;                          //包头结尾
Arr_Read[6]=01;                          //地址站号
Arr_Read[7]=03;                          //Modbus 功能码
Arr_Read[8]=00;                          //起始地址高字节
Arr_Read[9]=00;                          //起始地址低字节
Arr_Read[10]=00;                         //数据长度高字节
Arr_Read[11]=02;                         //数据长度低字节
var Arr_Received=SendTextToUDP('192.168.1.189',502,100,Arr_Read); //发送数据请求，接收报文为 Arr_Received 十进制数组中
WriteToTag('Simulator.Channel_1.UDP.40001',MakeWord(Arr_Received[10],Arr_Received[9]));
WriteToTag('Simulator.Channel_1.UDP.40002',MakeWord(Arr_Received[12],Arr_Received[11]));
```

SendTextUDP (a,b,c,d)函数使用 Demo：

var mes=SendTextUDP (a,b,c,d);解析：

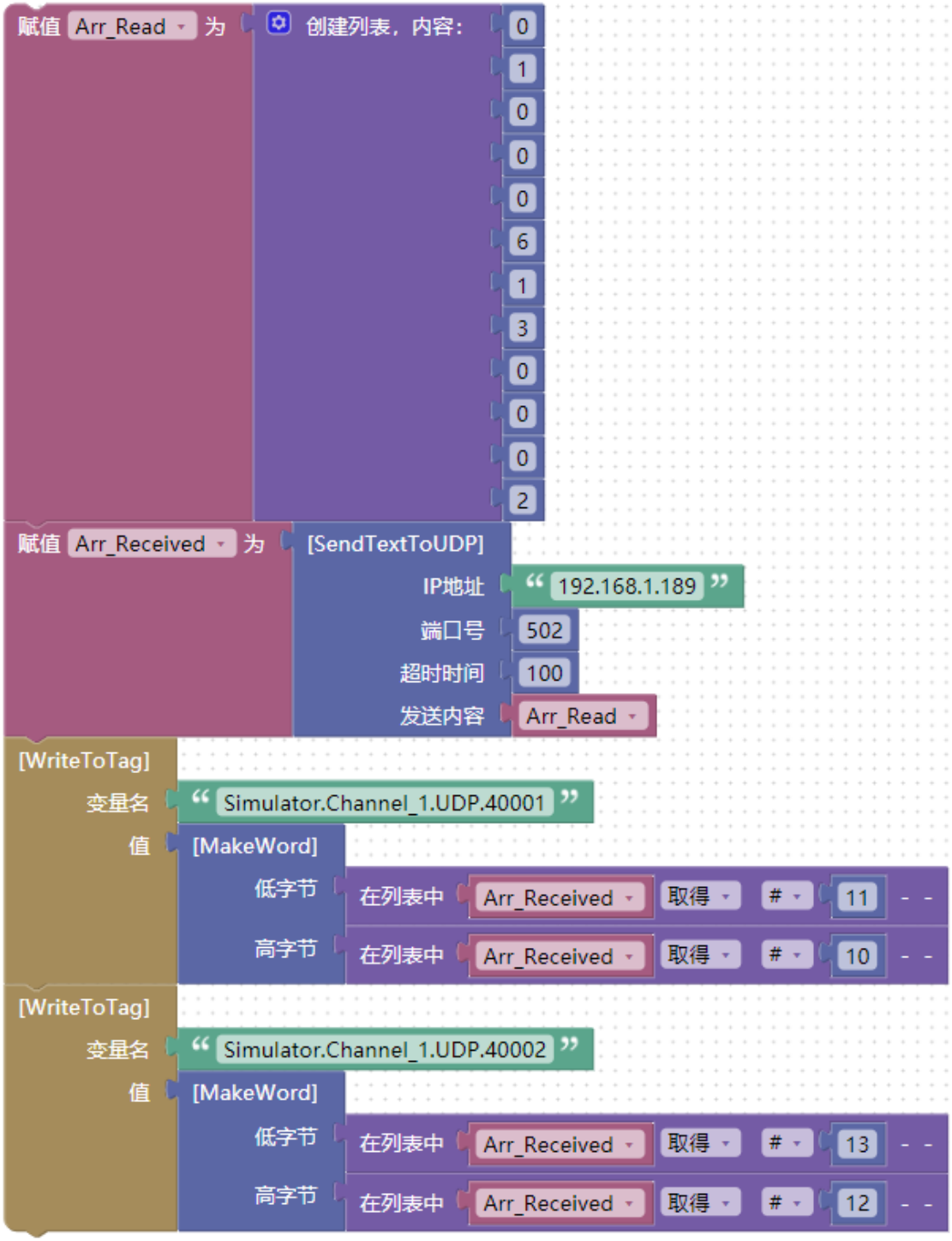
①变量 a：IP 地址’192.168.1.189’

②变量 b：端口号 502

③变量 c：超时时间 100ms

④变量 d：发送数组 Arr_Read

3.20.3 图形编程举例



步骤：

(1) 数组创建方式参照 3.16.3

(2) WriteToTag 参照 3.3.3

(3) MakeWord 使用参照 3.17.3

(4) SendTextToUDP 使用方式参照 3.16.3

3.21 HttpGet(a,b)

3.21.1 函数功能介绍

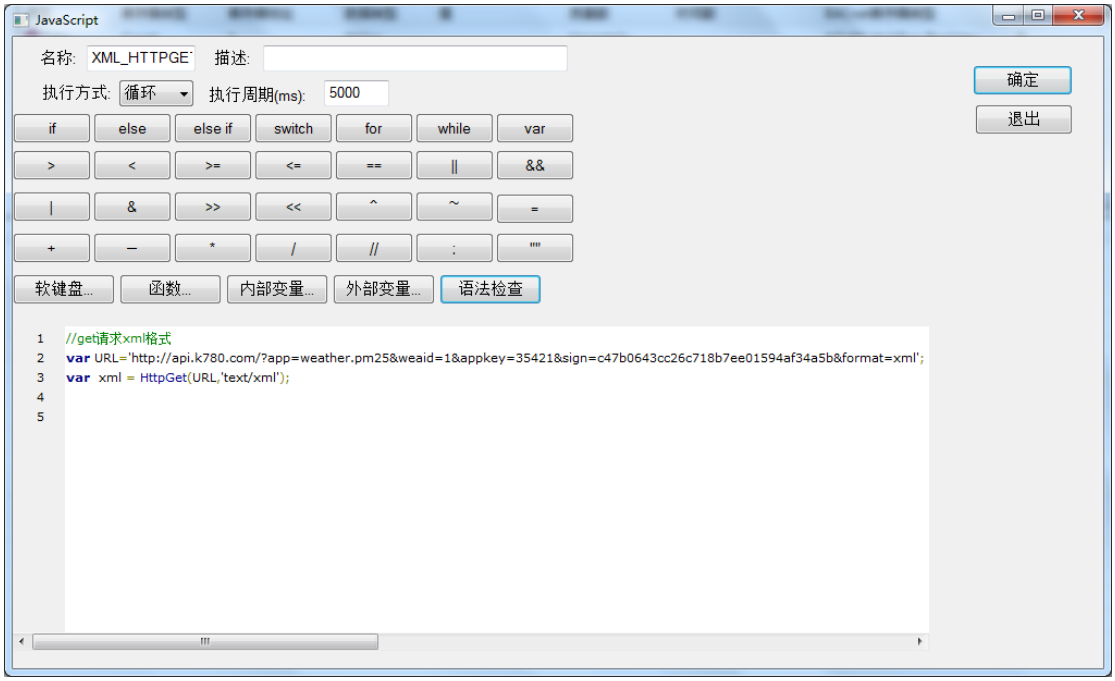
HttpGet(a,b)函数是通过网关中的 JS 脚本实现对外部 HTTP 接口进行 GET 请求，从而获取相应的接口数据。

参数：

a 为 HTTP 服务对应的 URL。

b 为 HTTP 服务对应的请求类型 XML 或者 JSON。

3.21.2 函数操作举例



//get 请求 xml 格式
//定义 GET 对象的 URL
var
URL='http://api.k780.com/?app=weather.today&weaid=1&appkey=10003&sign=b59bc3ef6191eb9f747dd4e83c99f2a4&format=xml';

```
//发送 GET 请求获取 xml
var xml = HttpGet(URL,'text/xml');
```

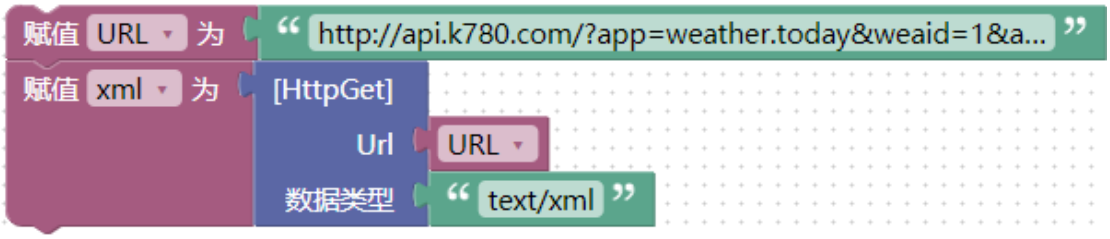
HttpGet(a,b)函数使用 Demo：

var mes=HttpGet(a,b);解析：

①变量 a：对应 HTTP 服务器 Get 请求的 URL

②变量 b：数据类型 XML 或者 JSON

3.21.3 图形编程举例



步骤：

1. 选择工具箱-->变量-->创建变量,输入 URL,点击确认。

2. 选择工具箱-->变量-->“赋值...为”，点击下拉框，选择 URL 变量，点击文本-->“”，创建“http://api.k780.com/?app=weather.today&weaid=1&a...”，挂接到 URL 变量上。

3. 选择工具箱-->HTTP-->HttpGet，创建 HttpGet 块；点击文本“”，创建“text/xml”，挂接到 HttpGet 数据类型参数上，将变量 URL 挂接到 Url 参数。

3.22 HttpPost(a, b, c)

3.22.1 函数功能介绍

HttpPost(a,b,c)函数是通过网关中的 JS 脚本实现对外部 HTTP 接口进行 Post 请求，从而获取相应的接口数据。

参数：

a 为 HTTP 服务对应的 URL。

b 为请求类型 XML 或者 JSON。

c 为发送内容

3.22.2 函数操作举例


```
//定义请求参数
var request
='<soap:Envelope><soap:Body><GetFirmWareInfo/></soap:Body></soap:Envelope>';

//Post 请求 xml 格式
var  xml = HttpPost('http://192.168.1.244/soap/GetFirmWareInfo', 'text/xml',
request);
alert(xml);
```

HttpPost(a,b,c)函数使用 Demo：

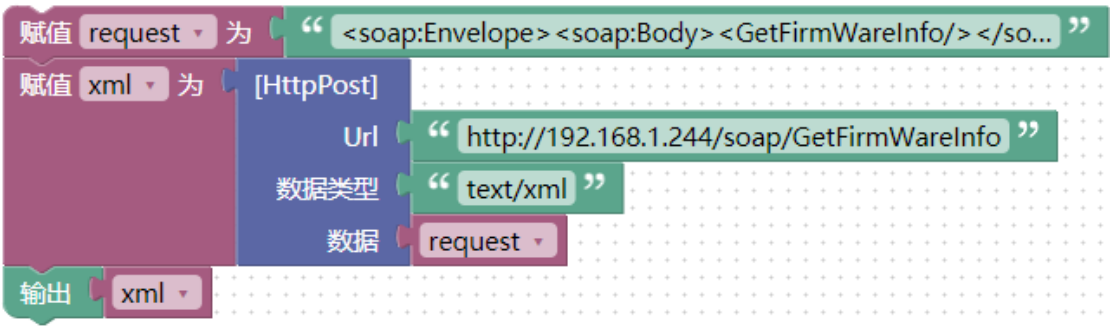
var mes=HttpPost(a,b,c);解析：

①变量 a：'http://192.168.1.244/soap/GetFirmWareInfo'

②变量 b：'text/xml'

③变量 c：
'<soap:Envelope><soap:Body><GetFirmWareInfo/></soap:Body></soap:Envelope>'

3.22.3 图形编程举例



步骤：

1. 选择工具箱-->变量-->创建变量，输入 request，点击确认。

2. 选择工具箱-->变量-->“赋值...为”，点击下拉框，选择 request 变量，点击文本-->“”，创建“<soap:Envelope><soap:Body><GetFirmWareInfo/></so...”，挂接到 request 变量上。

3. 选择工具箱-->HTTP-->HttpPost，创建 HttpPost 块；点击文本“”，创建“text/xml”，挂接到 HttpPost 数据类型参数上，将变量 request 挂接到数据参数。

3.23 HttpsPost(a, b, c)

3.23.1 函数功能介绍

HttpsPost(a,b,c)函数是通过网关中的 JS 脚本实现对外部 HTTPS 接口进行 Post 加密请求，从而获取相应的接口数据。

参数：

a 为 HTTPS 服务对应的 URL。

b 为请求类型 XML 或者 JSON。

c 为发送内容

3.23.2 函数操作举例


```
//定义请求参数
var request
='<soap:Envelope><soap:Body><GetFirmWareInfo/></soap:Body></soap:Envelope>';

//Post 请求 xml 格式
var  xml = HttpsPost('http://192.168.1.244/soap/GetFirmWareInfo', 'text/xml',
request);
alert(xml);
```

HttpsPost(a,b,c)函数使用 Demo：

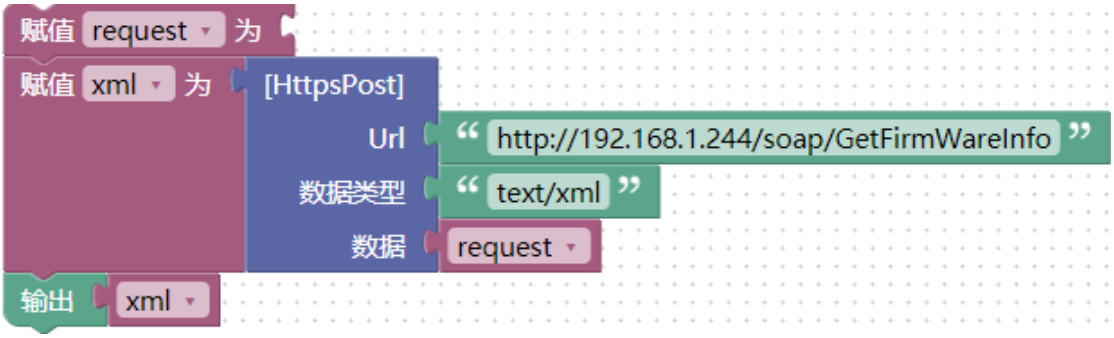
var mes=HttpsPost(a,b,c);解析：

①变量 a：'http://192.168.1.244/soap/GetFirmWareInfo'

②变量 b：'text/xml'

③变量 c：
'<soap:Envelope><soap:Body><GetFirmWareInfo/></soap:Body></soap:Envelope>'

3.23.3 图形编程举例



1. 选择工具箱-->变量-->创建变量，输入 request，点击确认。

2. 选择工具箱-->变量-->“赋值...为”，点击下拉框，选择 request 变量，点击文本-->“ ”，创建“<soap:Envelope><soap:Body><GetFirmWareInfo/></so...”，挂接到 request 变量上。

3. 选择工具箱-->HTTP-->HttpsPost，创建 HttpsPost 块；点击文本“ ”，创建“text/xml”，挂接到 HttpsPost 数据类型参数上，将变量 request 挂接到数据参数。

3.24 GetXMLAttrib(a, b, c)

3.24.1 函数功能介绍

GetXMLAttrib(a,b,c)函数是通过网关中的 JS 脚本实现对 XML 文件中对应属性获取。

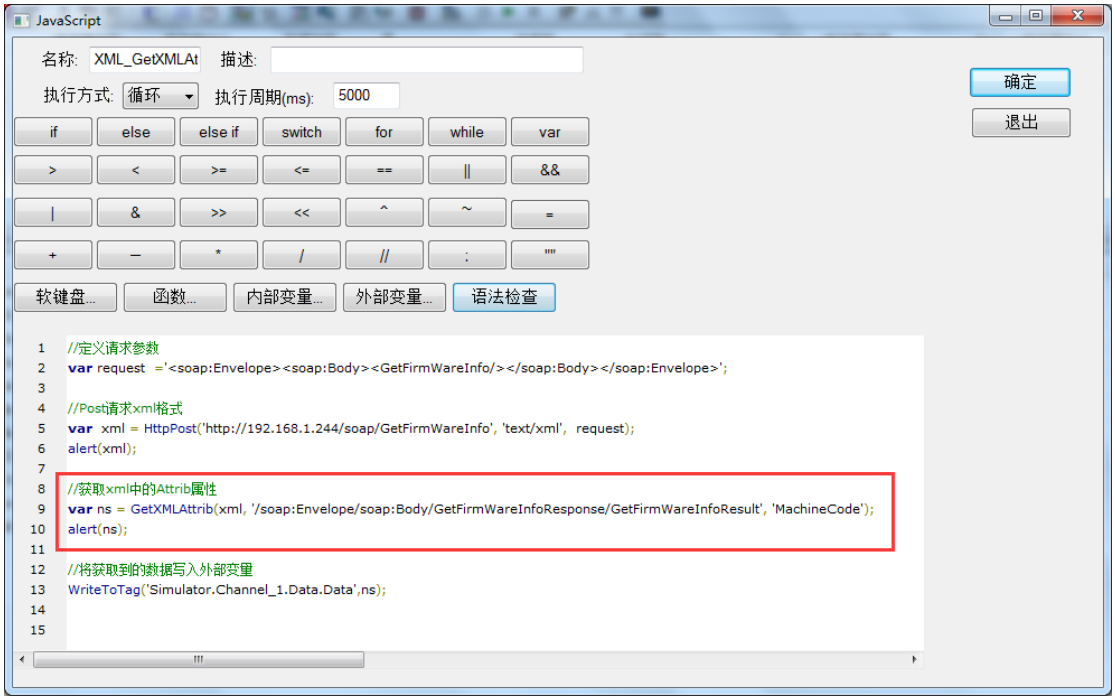
参数：

a 为 XML 格式的文本

b 为 XML 格式节点

c 为属性名称

3.24.2 函数操作举例




```
//获取 xml 中的 Attribute 属性
var ns = GetXMLAttrib(xml,
'/soap:Envelope/soap:Body/GetFirmWareInfoResponse/GetFirmWareInfoResult',
'MachineCode');
alert(ns);
```

GetXMLAttrib(a,b,c)函数使用 Demo：

var mes=GetXMLAttrib(a,b,c);解析：

①变量 a：xml（HTTP 请求后获取到的 XML）

②变量 b：

’/soap:Envelope/soap:Body/GetFirmWareInfoResponse/GetFirmWareInfoResult’

③变量 c：’MachineCode’

3.24.3 图形编程举例

3.25 GetXMLData(a, b)

3.25.1 函数功能介绍

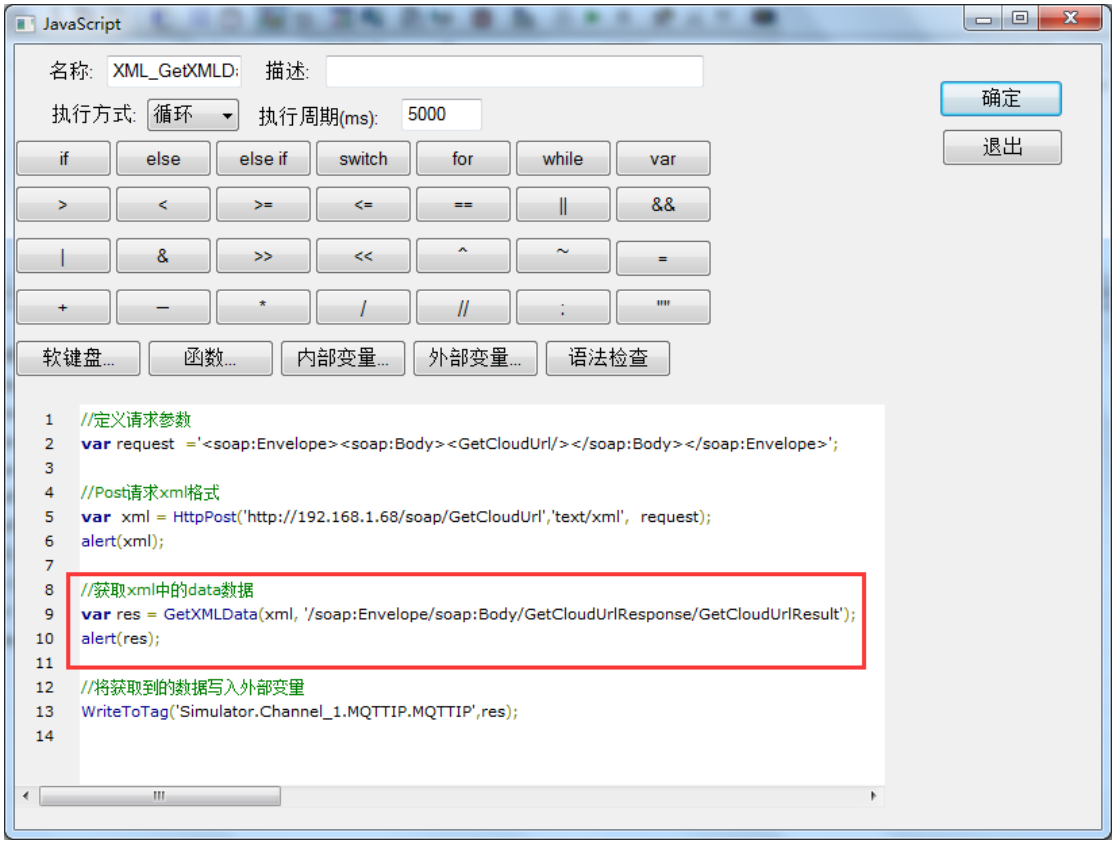
GetXMLData(a,b)函数是通过网关中的 JS 脚本实现对 XML 文件中对应数据获取。

参数：

a 为 XML 格式的文本

b 为 XML 格式节点

3.25.2 函数操作举例




```
//获取 xml 中的 data 数据
var res = GetXMLData(xml,
'/soap:Envelope/soap:Body/GetCloudUrlResponse/GetCloudUrlResult');
alert(res);
```

GetXMLData(a,b)函数使用 Demo：

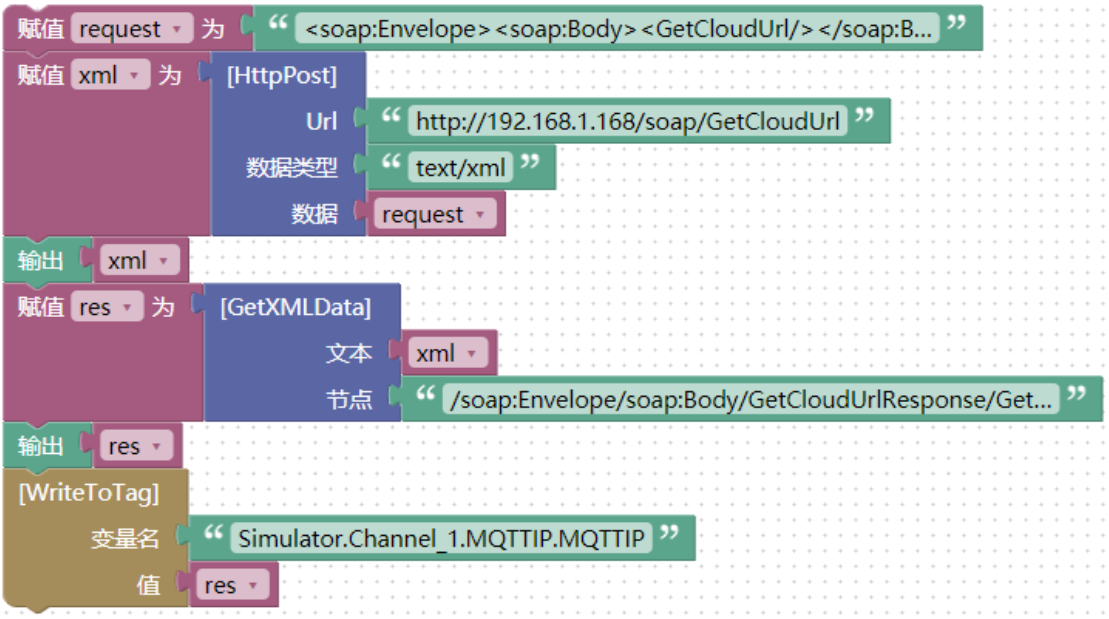
var mes=GetXMLData(a,b);解析：

①变量 a：xml（HTTP 请求后获取到的 XML）

②变量 b：

'/soap:Envelope/soap:Body/GetCloudUrlResponse/GetCloudUrlResult'

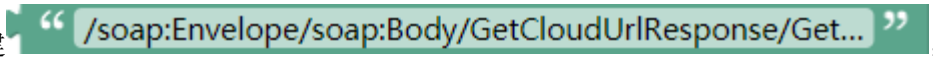
3.25.3 图形编程举例



步骤：

1. request 变量、xml 变量创建方法和 HttpPost 用法参照 3.22.2。

2. 选择工具箱-->变量-->创建变量，输入 res，点击确定。

3. 选择工具箱-->Http-->GetXMLData，创建 GetXMLData 块，点击文本-->  ，创建  ，将该块和 xml 变量挂接到 GetXMLData 块上。

4. WriteToTag 使用方式参照 3.3.3。

3.26 DoCRC16(a, b, c)

3.26.1 函数功能介绍

DoCRC16(a,b,c)函数是通过网关中的JS脚本实现对指定报文数组进行CRC16校验，输出为个 2 字节的 CRC16 校验。

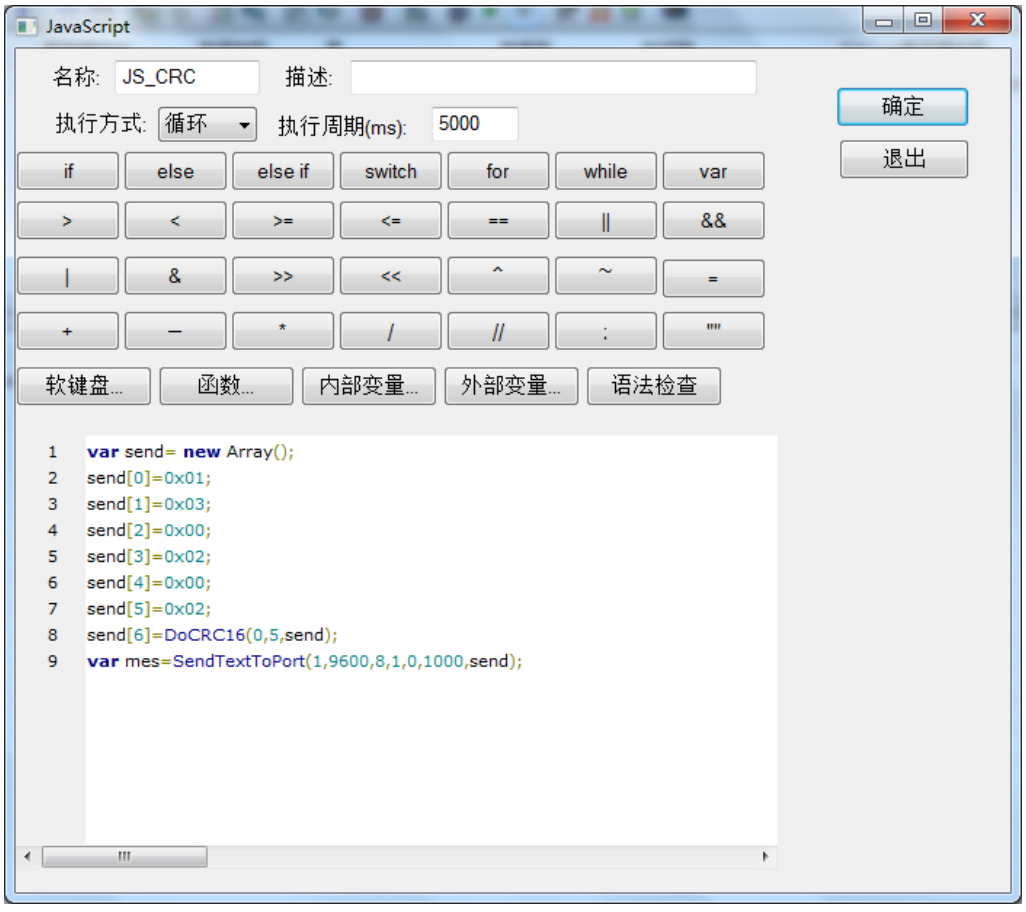
参数：

a 为起始位

b 为结束位

c 为校验的数组，可以是十进制数组或者十六进制数组

3.26.2 函数操作举例




```
var send= new Array();
send[0]=0x01;
send[1]=0x03;
send[2]=0x00;
```

```
send[3]=0x02;
send[4]=0x00;
send[5]=0x02;
send[6]=DoCRC16(0,5,send);     //CRC16 校验
var mes=SendTextToPort(1,9600,8,1,0,1000,send);
```

DoCRC16(a,b,c)函数使用 Demo：

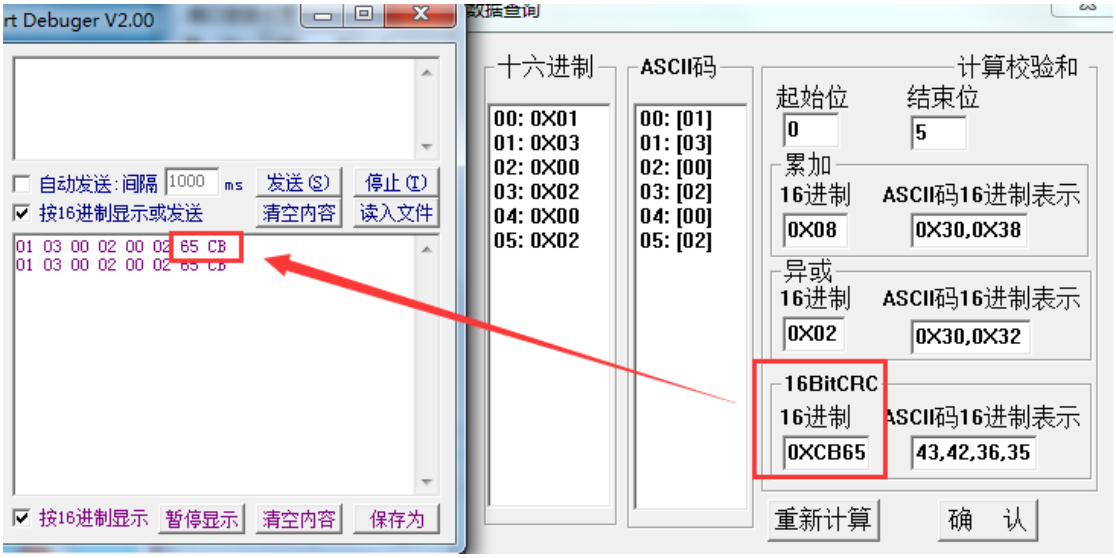
```
var mes=DoCRC16(a,b,c);
```

解析：

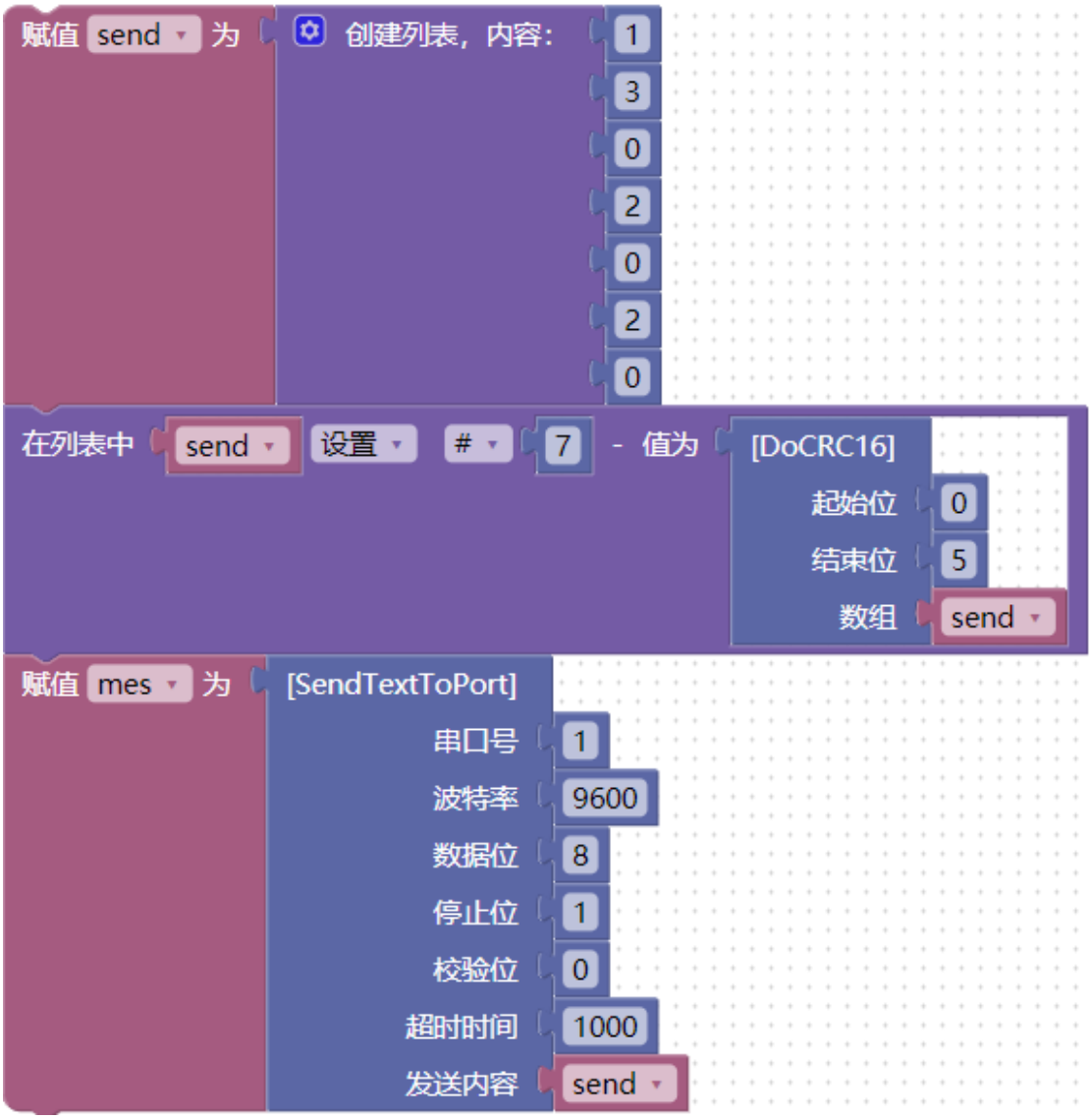
①变量 a：起始位 0

②变量 b：结束位 5

③变量 c：校验的数组 send



3.26.3 图形编程举例



步骤：

1. send 变量、数组创建方式参照 3.16.3。

2. 选择工具箱-->校验-->DoCRC16，创建 DoCRC16 块；点击数学-->  ,创建起始位和阶数位参数块，将 send 变量挂接到 DoCRC16 的数组参数位置。

3. 选择工具箱-->列表--> 

，

点击 item，在下拉框中选择 send，点击数学--> 

，创建参数 

（对应数组索引为 6），然后将 DoCRC16 块拖入最后一个参数中，如下：

4. SendTextToPort 用法参照 3.16.3。

3.27 CheckSum(a, b, c)

3.27.1 函数功能介绍

CheckSum(a,b,c)函数是通过网关中的 JS 脚本实现对指定报文数组进行累加和校验，输出为一个字节的累加和校验。

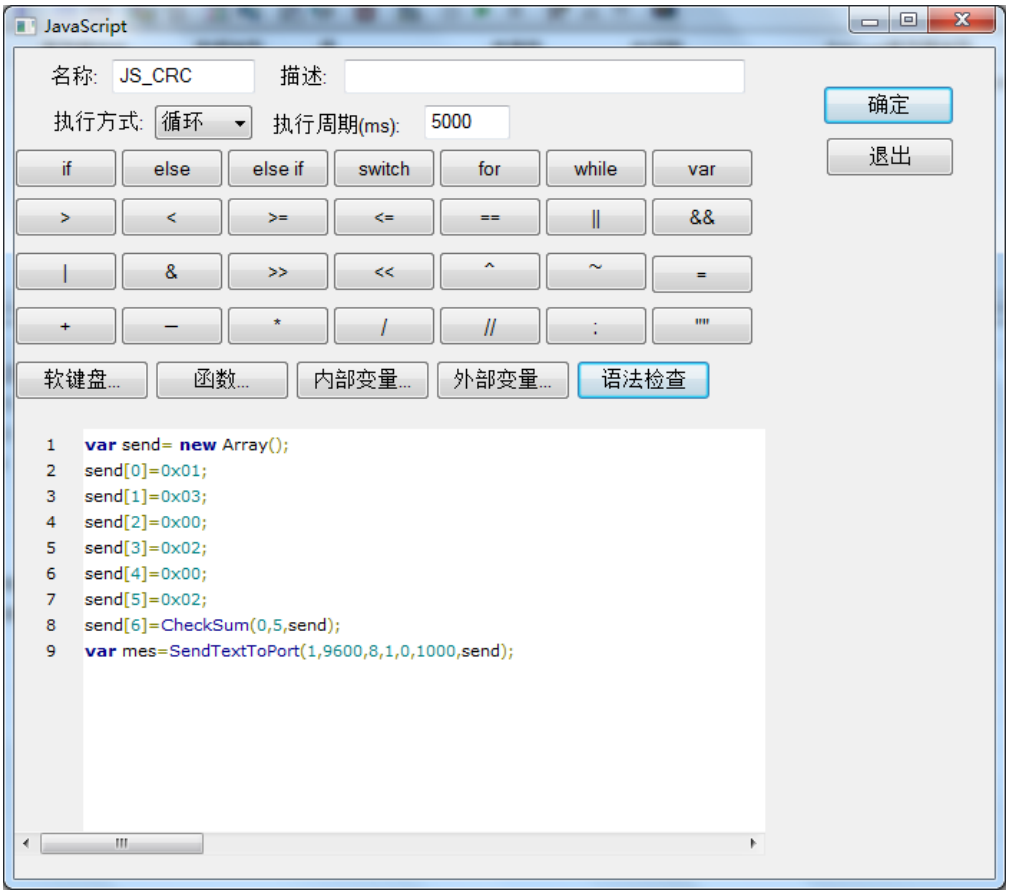
参数：

a 为起始位

b 为结束位

c 为校验的数组，可以是十进制数组或者十六进制数组

3.27.2 函数操作举例




```
var send= new Array();
send[0]=0x01;
send[1]=0x03;
send[2]=0x00;
send[3]=0x02;
send[4]=0x00;
send[5]=0x02;
send[6]=CheckSum(0,5,send);     //累加和校验
var mes=SendTextToPort(1,9600,8,1,0,1000,send);
```

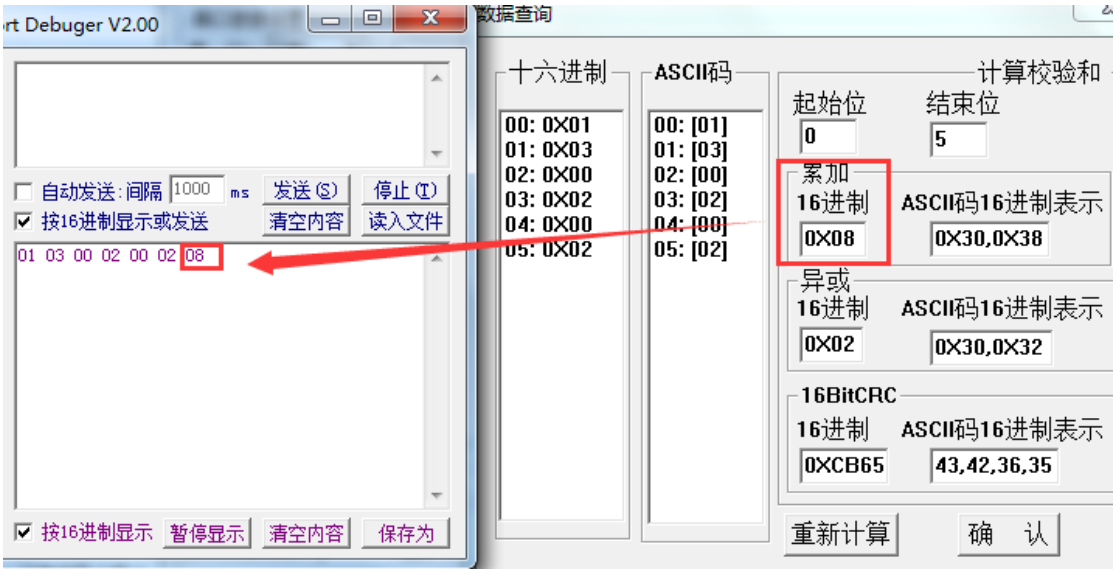
CheckSum(a,b,c)函数使用 Demo：

var mes=CheckSum(a,b,c);解析：

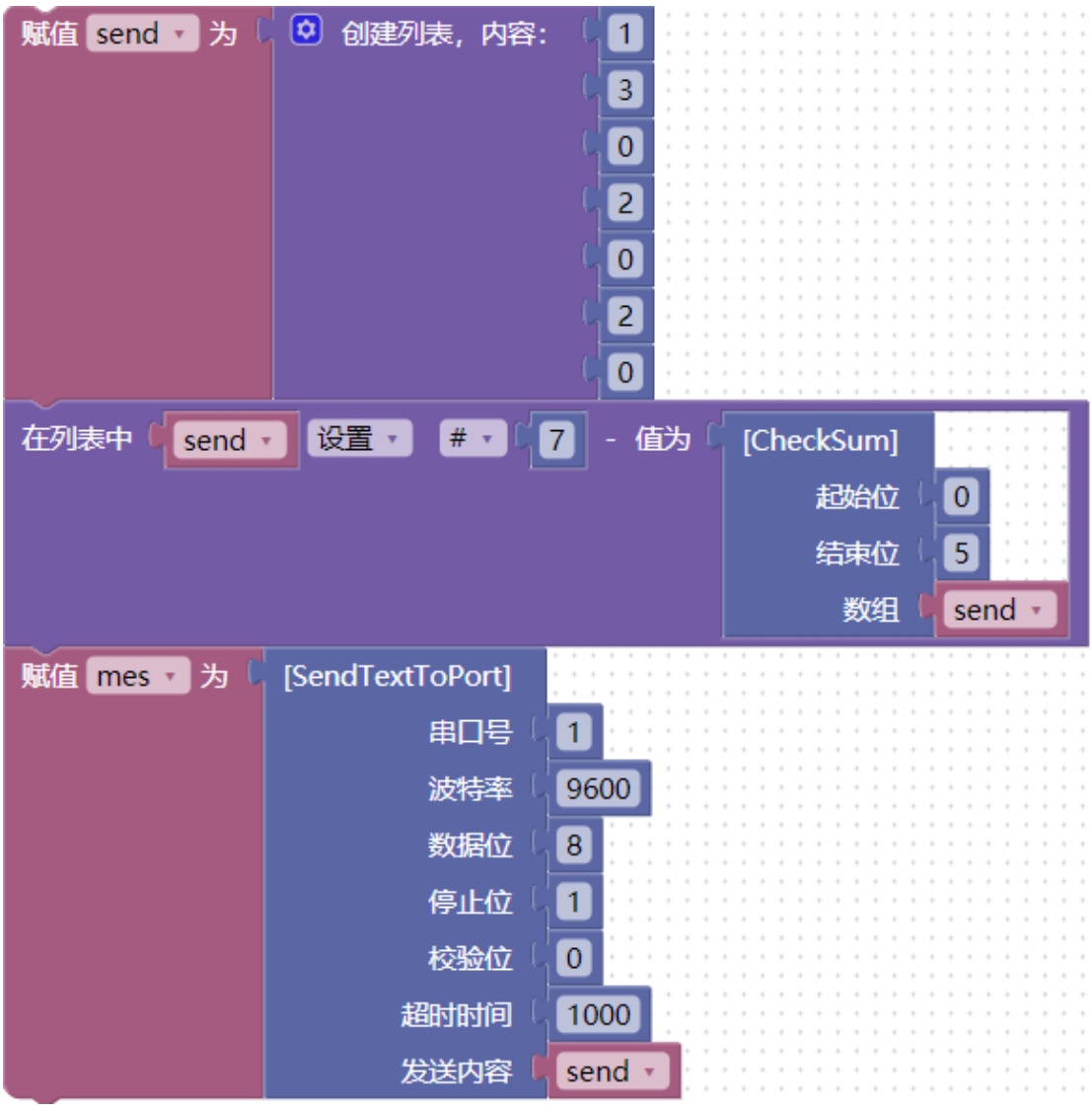
①变量 a：起始位 0

②变量 b：结束位 5

③变量 c：校验的数组 send


rt Debuger V2.00
自动发送: 间隔 1000 ms
发送 (S)
停止 (T)
按16进制显示或发送
清空内容
读入文件
01 03 00 02 00 02 08
按16进制显示
暂停显示
清空内容
保存为
数据查询
十六进制
00: 0X01
01: 0X03
02: 0X00
03: 0X02
04: 0X00
05: 0X02
ASCII码
00: [01]
01: [03]
02: [00]
03: [02]
04: [00]
05: [02]
计算校验和
起始位
0
结束位
5
累加
16进制
0X08
ASCII码16进制表示
0X30,0X38
异或
16进制
0X02
ASCII码16进制表示
0X30,0X32
16BitCRC
16进制
0XCB65
ASCII码16进制表示
43,42,36,35
重新计算
确 认

3.27.3 图形编程举例


赋值 send 为
创建列表，内容:
1
3
0
2
0
2
0
在列表中 send 设置 # 7 - 值为 [CheckSum]
起始位 0
结束位 5
数组 send
赋值 mes 为
[SendTextToPort]
串口号 1
波特率 9600
数据位 8
停止位 1
校验位 0
超时时间 1000
发送内容 send

步骤：

1. send 变量、数组创建方式参照 3.16.3。

2. 选择工具箱-->校验-->CheckSum，创建 CheckSum 块；点击数学--> 

,创建起始位和阶数位参数块，将 send 变量挂接到 CheckSum 的数组参数位置。

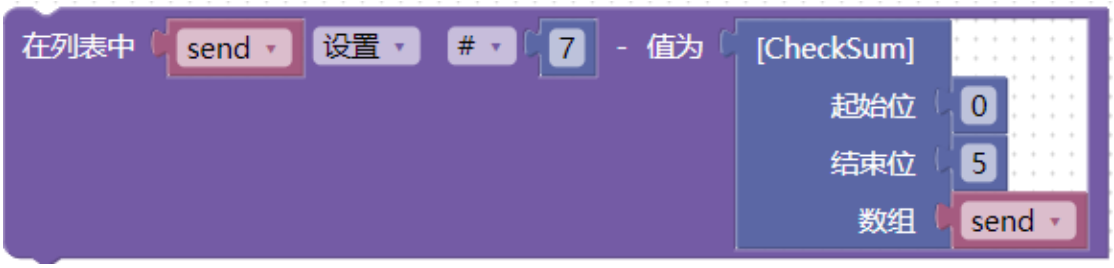
3. 选择工具箱-->列表--> 

，

点击 item，在下拉框中选择 send，点击数学--> 

，创建参数 

（对应数组索引为 6），然后将 CheckSum 块拖入最后一个参数中，如下：



5. SendTextToPort 用法参照 3.16.3。

3.28 XORCheckSum(a, b, c)

3.28.1 函数功能介绍

XORCheckSum(a,b,c)函数是通过网关中的 JS 脚本实现对指定报文数组进行异或和校验，输出为一个字节的异或和校验。

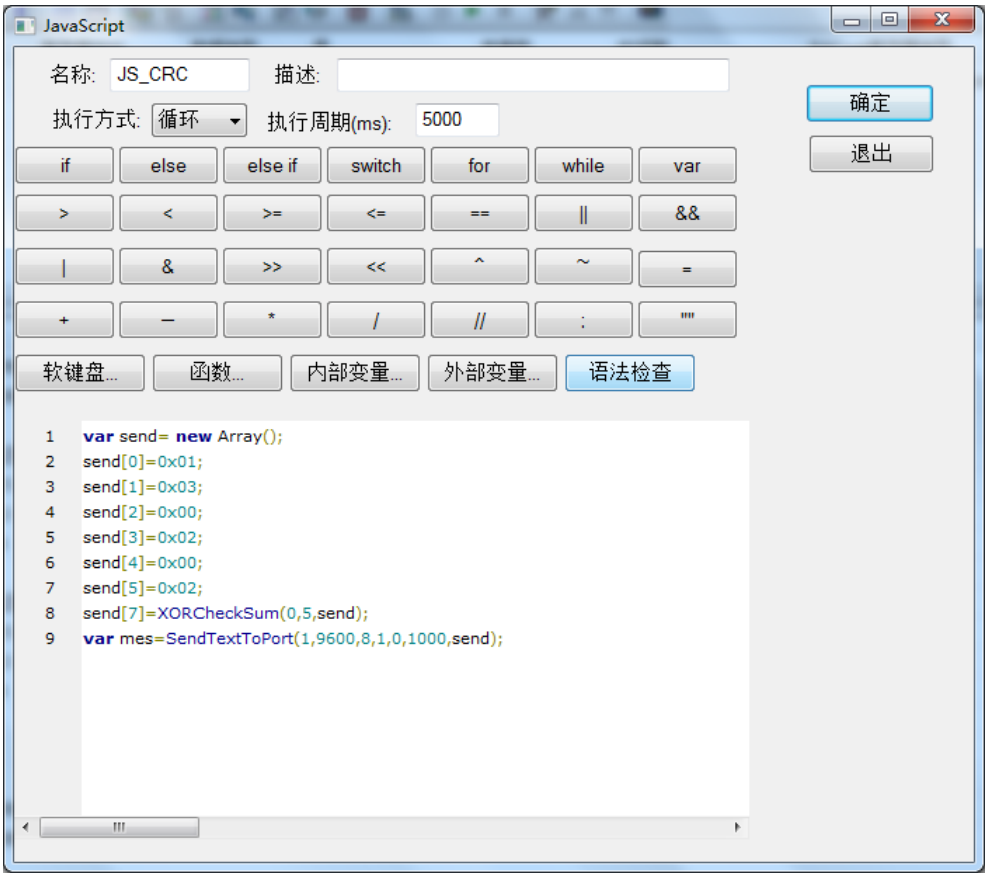
参数：

a 为起始位

b 为结束位

c 为校验的数组，可以是十进制数组或者十六进制数组

3.28.2 函数操作举例




```
var send= new Array();
send[0]=0x01;
send[1]=0x03;
send[2]=0x00;
send[3]=0x02;
send[4]=0x00;
send[5]=0x02;
send[6]=XORCheckSum(0,5,send);     //异或校验
var mes=SendTextToPort(1,9600,8,1,0,1000,send);
```

XORCheckSum(a,b,c)函数使用 Demo：

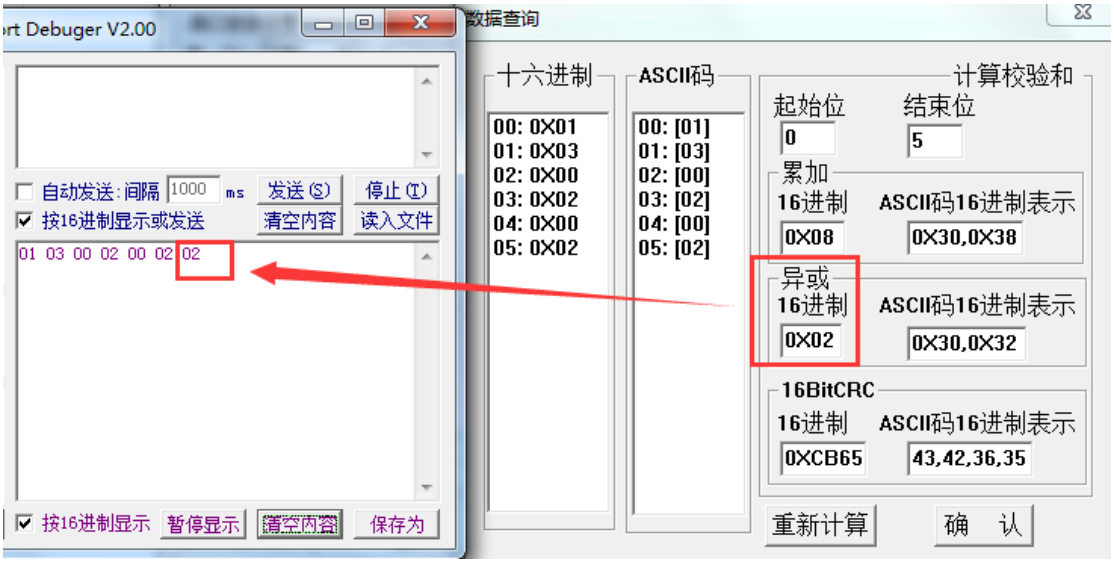
```
var mes=XORCheckSum(a,b,c);
```

解析：

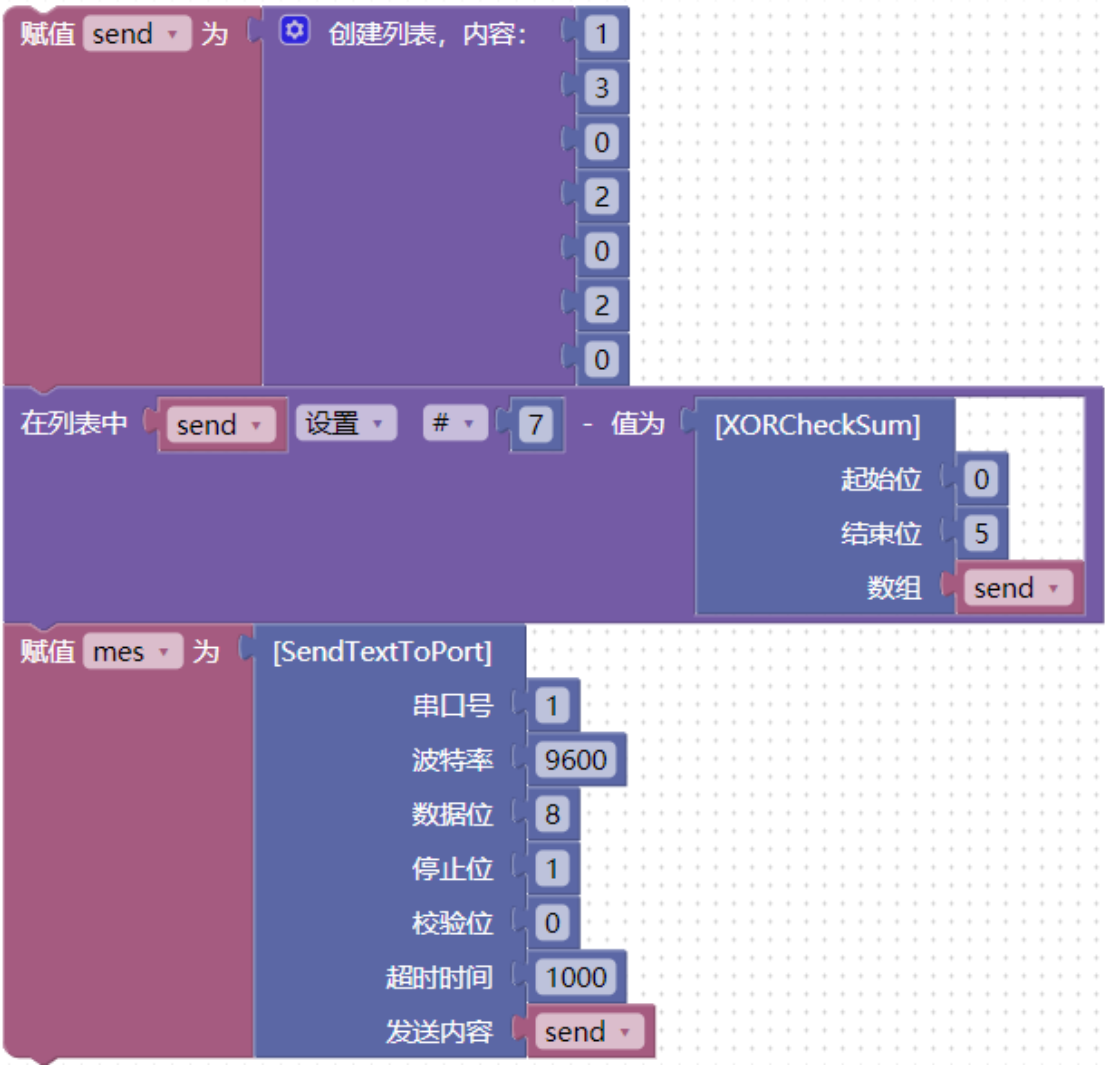
①变量 a：起始位 0

②变量 b：结束位 5

③变量 c：校验的数组 send



3.28.3 图形编程举例



步骤：

1. send 变量、数组创建方式参照 3.16.3。

2. 选择工具箱-->校验-->XORCheckSum，创建 XORCheckSum 块；点击数学--> 

,创建起始位和阶数位参数块，将 send 变量挂接到 XORCheckSum 的数组参数位置。

3. 选择工具箱-->列表--> 

，

点击 item，在下拉框中选择 send，点击数学--> 

，创建参数 

（对应数组索引为 6），然后将 XORCheckSum 块拖入最后一个参数中，如下：

4. SendTextToPort 用法参照 3.16.3。

3.29 GetJSONData(a, b)

3.29.1 函数功能介绍

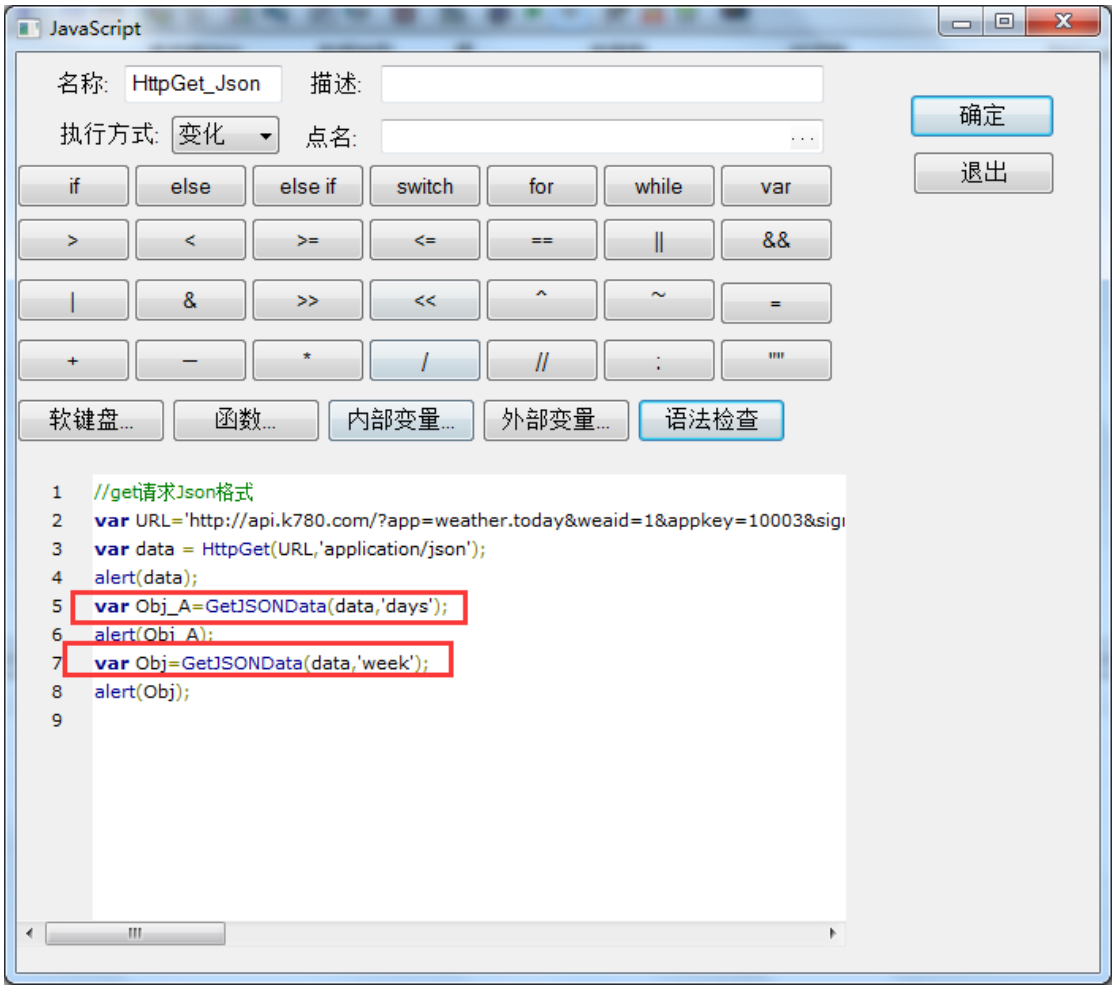
GetJSONData(a,b)函数是通过网关中的 JS 脚本实现在指定 Json 串中获取所需的属性值。

参数：

a 为 Json 格式的文本

b 为 Json 的 Key

3.29.2 函数操作举例




```
//get 请求 Json 格式
//定义 GET 对象的 URL
var
URL='http://api.k780.com/?app=weather.today&weaid=1&appkey=10003&sign=b59bc3ef6
191eb9f747dd4e83c99f2a4&format=json';
//发送 GET 请求获取 JSON
var data = HttpGet(URL,'application/json');
alert(data);

//获取 JSON 串中的 day
var Obj_A=GetJSONData(data,'days');
alert(Obj_A);

//获取 JSON 串中的 week
var Obj=GetJSONData(data,'week');
alert(Obj);
```

GetJSONData(a,b)函数使用 Demo：

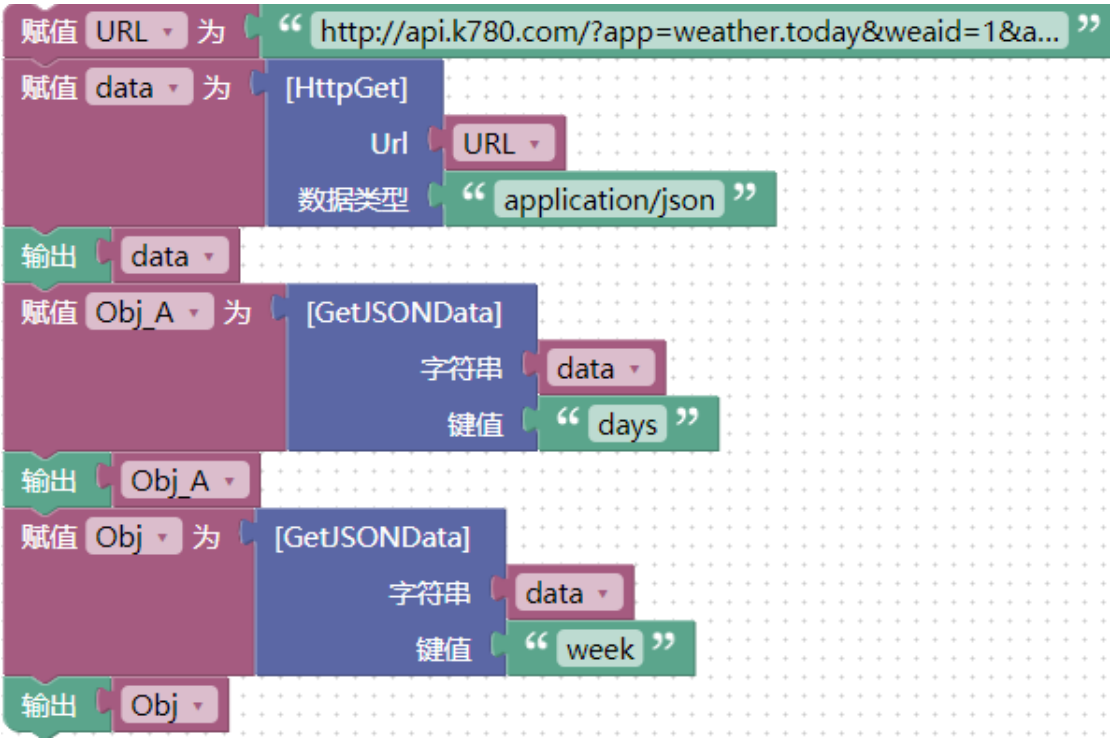
```
var mes=GetJSONData(a,b);
```

解析：

①变量 a：对应的 Json 串

②变量 b：对应 Json 串中的 key（字符串）

3.29.3 图形编程举例



步骤：

1. URL 变量、data 变量、HttpGet 用法参照 3.21.2

2. 选择工具箱-->取值-->GetJSONData,创建 GetJSONData 块，点击文本 --> " " ，创建 " days " ，挂接到 GetJSONData 块键值参数位置，然后将变量 data 挂接到字符串参数位置。

3. 选择工具箱-->取值-->GetJSONData,创建 GetJSONData 块，点击文本-->，创建 

，挂接到 GetJSONData 块键值参数位置，然后将变量 data 挂接到字符串参数位置。

4. 选择工具箱-->文本-->,创建输出块，将变量 Obj 挂到输出块上。

3.30 ReceiveTextFromPort(a,b,c,d,e,f)

3.30.1 函数功能介绍

ReceiveTextFromPort (a,b,c,d,e,f)函数是用户用来操作网关串口的方法，通过此方法，接收一些串口通讯设备主动发送过来的数据。

参数：

a 为对应网关的 COM 口 Number 号,不能与驱动里正常通讯使用的串口号冲突。

b 为 com 口通讯使用的波特率，比如 38400,19200,9600,4800,2400,1200 等

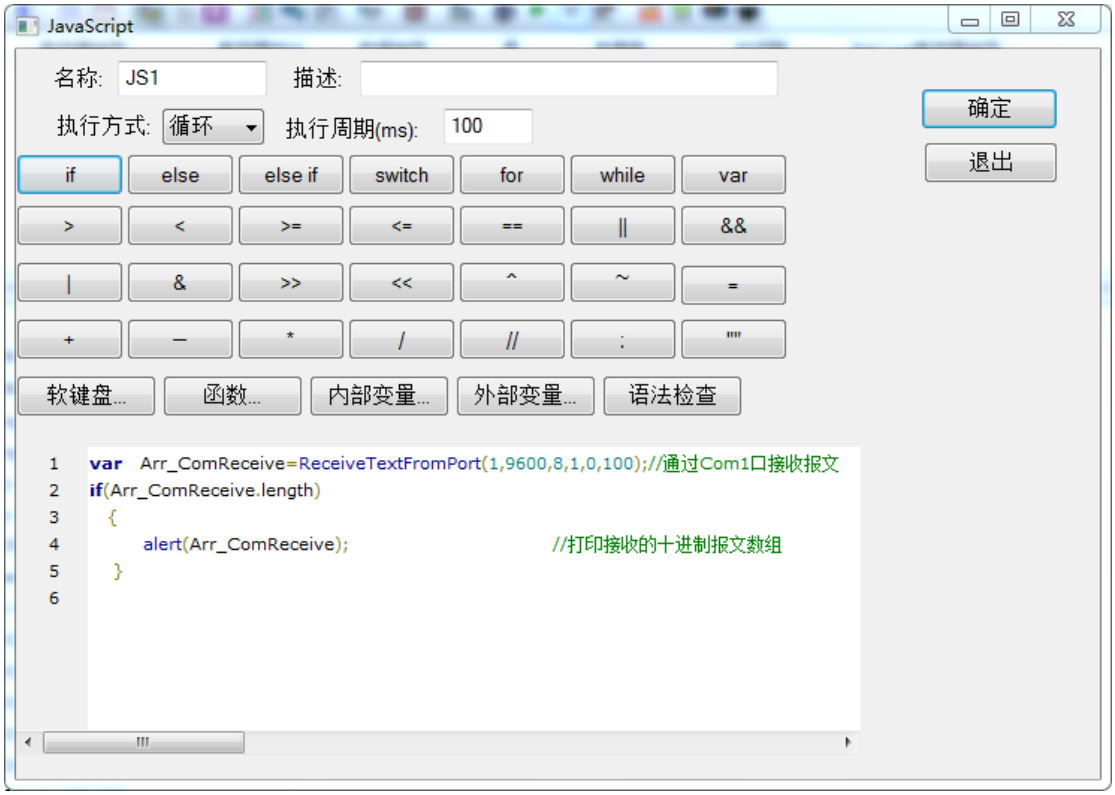
c 为 com 口通讯使用的数据位（8 或者 7）。

d 为 com 口通讯使用的停止位（1 或者 2）。

e 为 com 口通讯使用的校验位（0,1,2 分别代表无校验，奇校验，偶校验）。

f 为 com 口通讯等待超时时间（单位 ms）。

3.30.2 函数操作举例



var Arr_ComReceive = ReceiveTextFromPort (1,9600,8,1,0,100);

函数使用 Demo：

ReceiveTextFromPort(a,b,c,d,e,f)

解析：

①变量 a:com1。

②变量 b：9600 波特率。

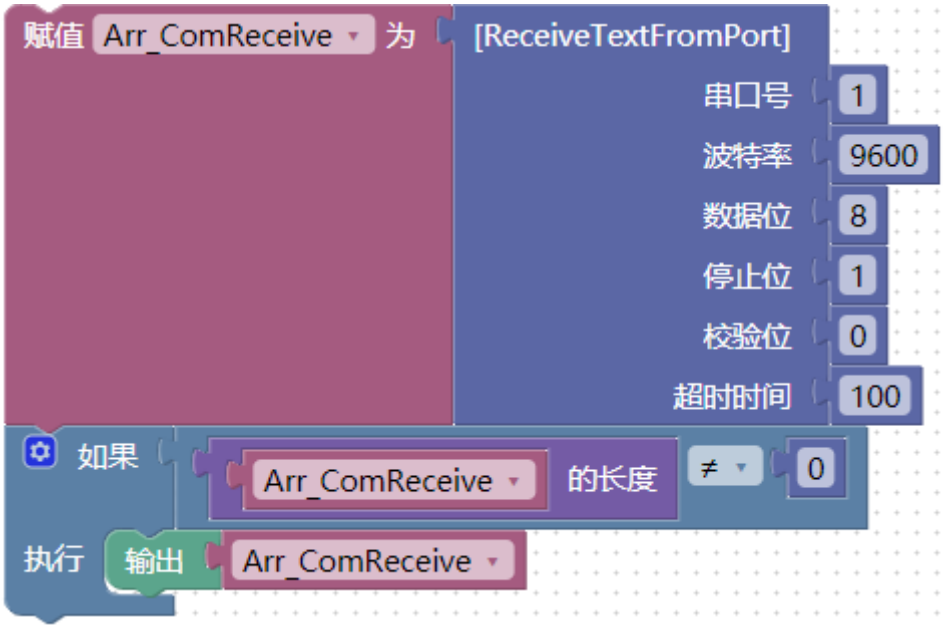
③变量 c：8 数据位。

④变量 d：1 停止位。

⑤变量 e：无校验。

⑥变量 f：超时时间 100ms。对于发送频率较快的设备 ，超时时间可以设置到 0。

3.30.3 图形编程举例

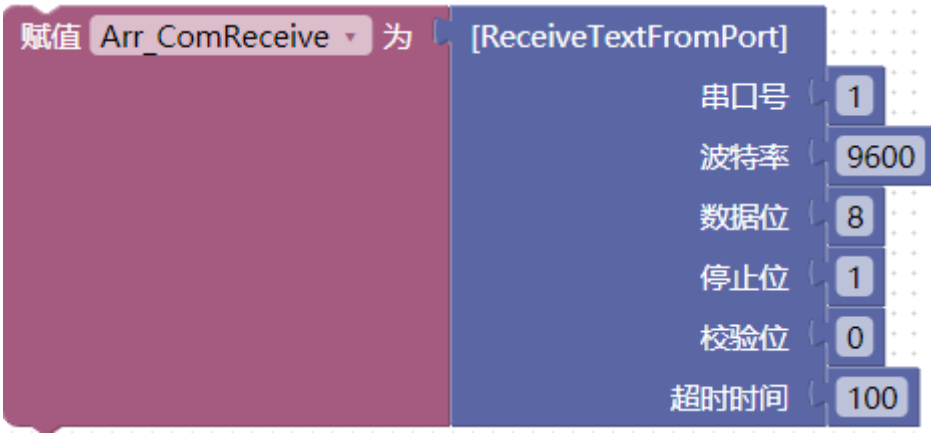


步骤：

1. 选择工具箱-->变量-->创建变量，输入 Arr_ComReceive，点击确认。

2. 选择工具箱-->报文-->ReceiveTextFromPort，创建 ReceiveTextFromPort 块；点击数学-->0,分别创建串口号、波特率、数据位、停止位、校验位和起始时间块挂接在 ReceiveTextFromPort 上面。

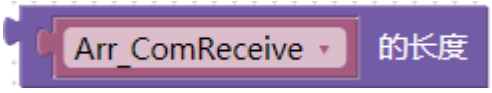
3. 选择工具箱-->变量-->“赋值...为”，点击打开下拉框，选择 Arr_ComReceive 变量，将 ReceiveTextFromPort 块挂接到 Arr_ComReceive 上，如下：



4. 选择工具箱-->逻辑--> 

，创建关系运算表达式块；点击列表 --> 

，将变量 Arr_ComReceive 拖入，如下：

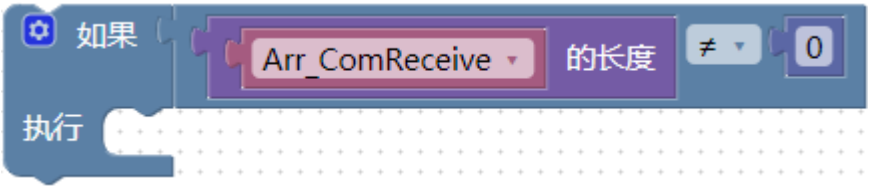


，然后将该块拖入表达式块第一个参数，如下：

，将逻辑判断修改为不等于 0。

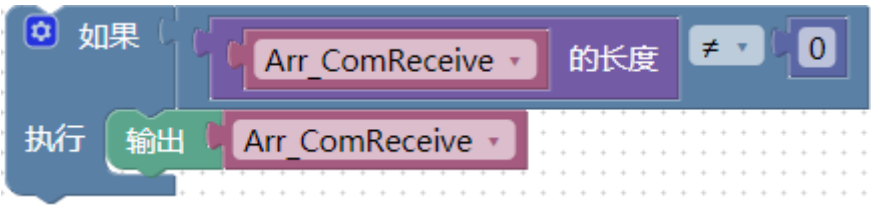
5. 选择工具箱-->逻辑--> 

，创建条件块；将步骤 4 中的关系运算符块拖入条件块，如下：



6. 选择工具箱-->文本--> 

，创建输出块，点击变量-->Arr_ComReceive，将变量 Arr_ComReceive 拖入输出块，如下：



7. 将 3、6 中的块自上而下链接在一起。

3.31 ReceiveTextFromTCP(a, b)

3.31.1 函数功能介绍

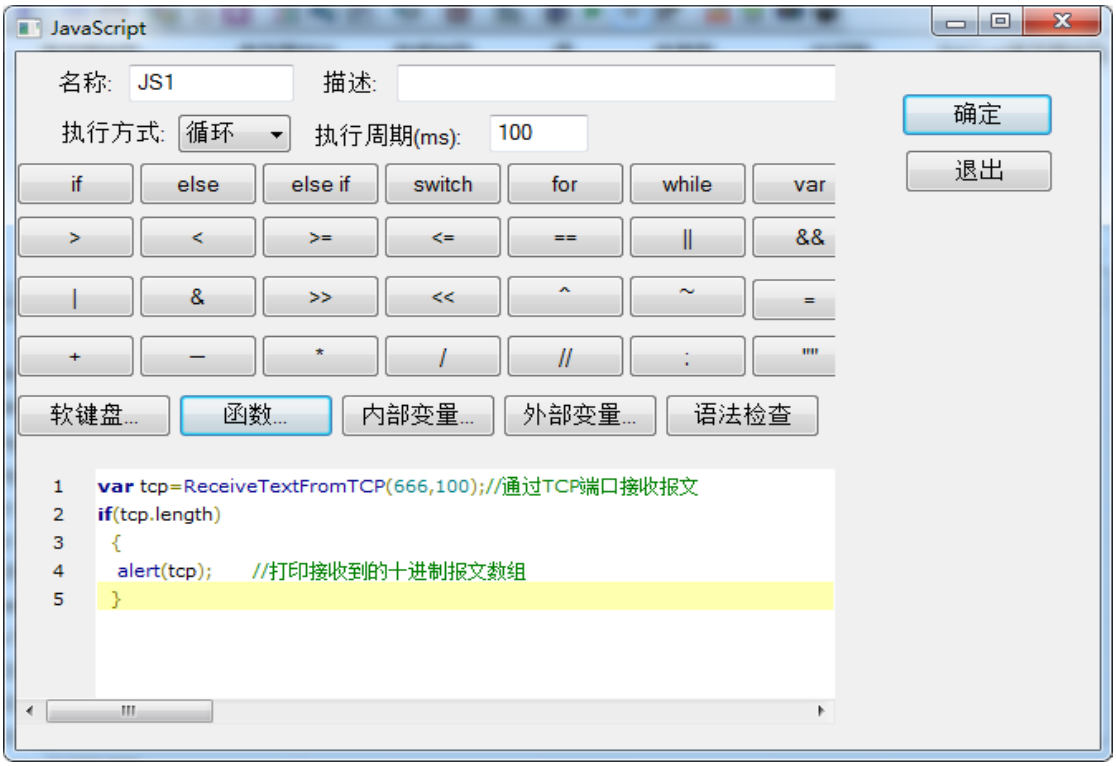
ReceiveTextFromTCP(a,b)函数是用户用来操作网关网口的方法，通过此方法，用户自己就可以编程通讯，接收一些 TCP 通讯设备主动发送过来的数据。

参数：

a 为网关自身开启的 TCP Server 端口号。

b 为接收 TCP 报文数据的超时时间。

3.31.2 函数操作举例



ReceiveTextFromTCP(a,b)函数使用 Demo：

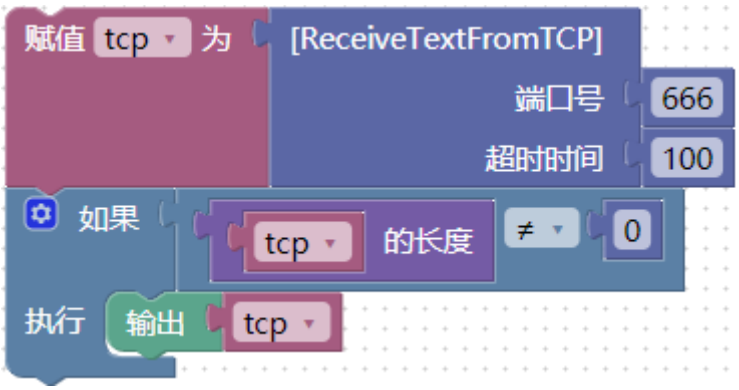
```
var tcp=ReceiveTextFromTCP(666,100);
```

解析：

①变量 a：网关自身开启的 TCP 服务端口号

②变量 b：超时时间 100ms

3.31.3 图形编程举例



步骤：

请参考 3.30.3 中 ReceiveTextFromPort 的操作方法。

3.32 ReceiveTextFromUDP(a, b)

3.32.1 函数功能介绍

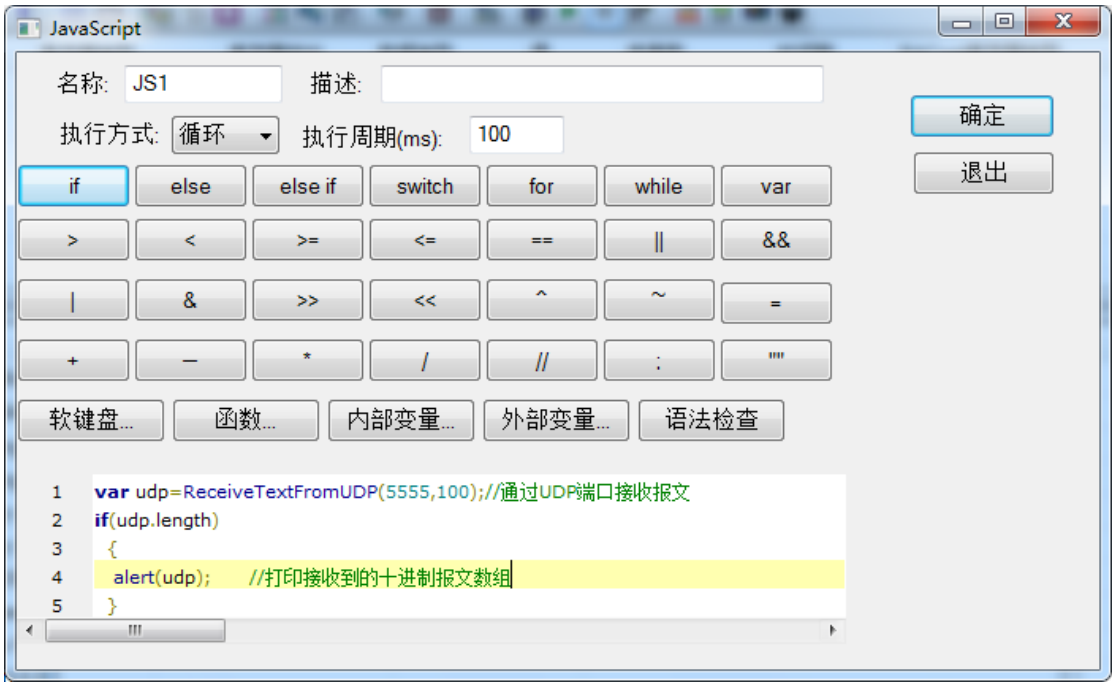
ReceiveTextFromUDP(a,b)函数是用户用来操作网关网口的方法，通过此方法，用户自己就可以编程通讯，接收一些 UDP 通讯设备主动发送过来的数据。

参数：

a 为网关自身开启的 UDP Server 端口号。

b 为接收 UDP 报文数据的超时时间。

3.32.2 函数操作举例



ReceiveTextFromUDP(a,b)函数使用 Demo：

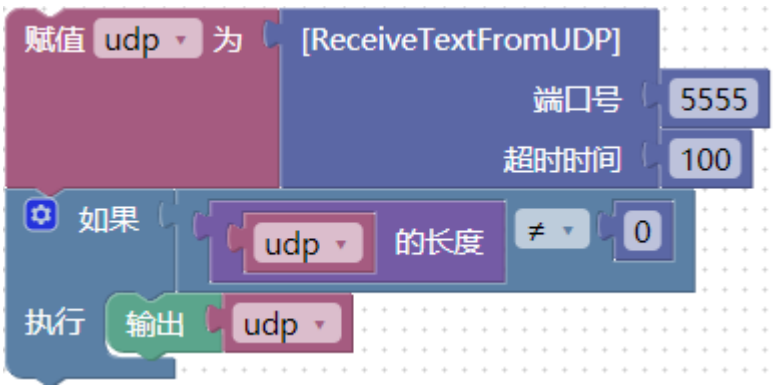
```
var udp=ReceiveTextFromUDP(5555,100);
```

解析：

①变量 a：网关自身开启的 UDP 服务端口号

②变量 b：超时时间 100ms

3.32.3 图形编程举例



步骤：

请参考 3.30.3 中 ReceiveTextFromPort 的操作方法。

3.33 OnMQTTMessage(szTopic,szMessage)

3.33.1 函数功能介绍

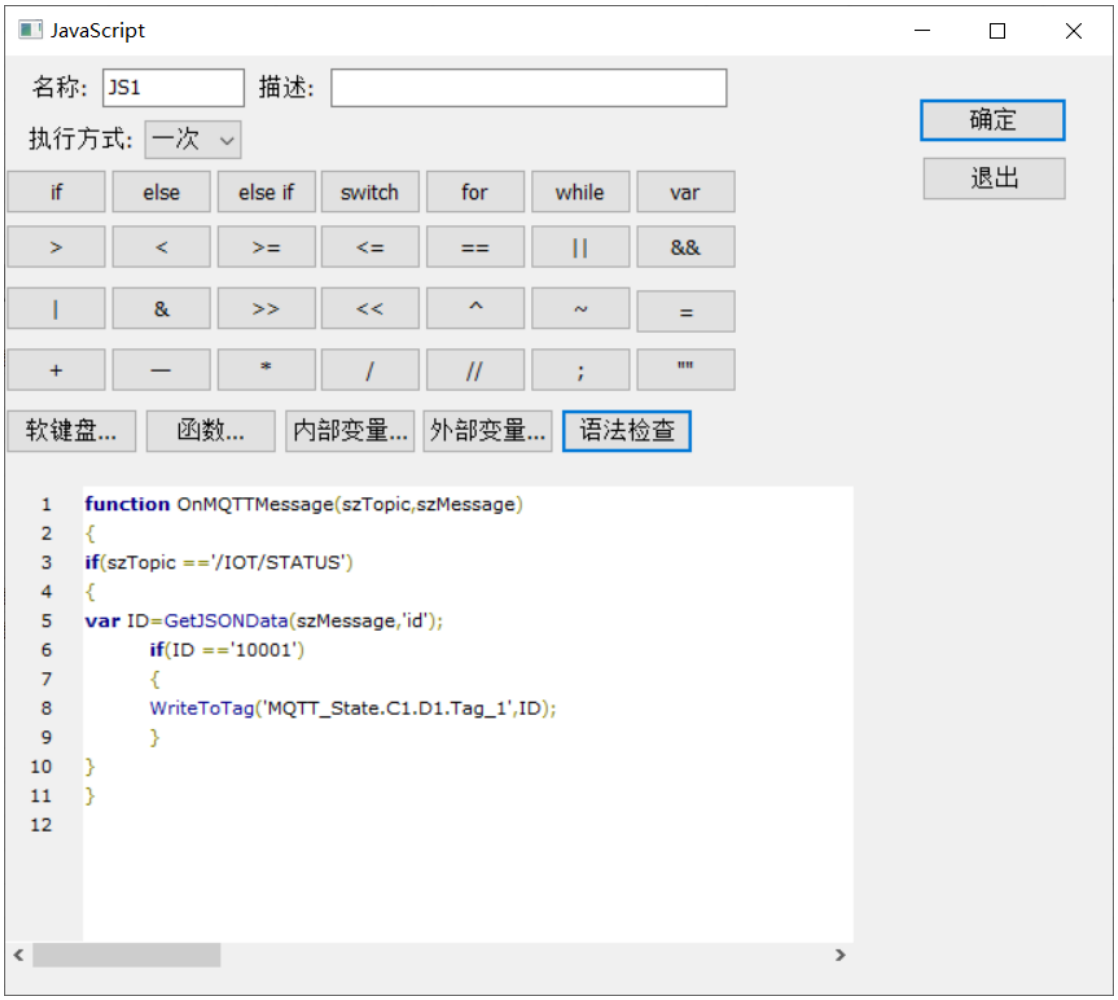
function OnMQTTMessage(szTopic,szMessage)是配合 IOT 中的 JS 脚本 MQTT 插件使用，用于订阅对应的 MQTT 服务器数据包，然后根据主题或者消息日志内容来解析自己需要的数据。

参数：

szTopic 为订阅的主题。

szMessage 为订阅到的数据包。

3.33.2 函数操作举例




```
function OnMQTTMessage(szTopic,szMessage)
{
if(szTopic ==’/IOT/STATUS’)
{
var ID=GetJSONData(szMessage,’id’);
    if(ID ==’10001’)
    {
    WriteToTag(’MQTT_State.C1.D1.Tag_1’,ID);
    }
}
}
```

3.34 onhttpmessage(url,json)、HttpServer(a,b)

3.34.1 函数功能介绍

onhttpmessage(url,json)是用于定义 http 的回调函数， HttpServer(a,b)函数用户开启网关的 http 服务端口号默认为 80，无须更改，参数 a 对应的 URL 方法，参数 b 为回调函数名称，两者配合使用，实现接收对 http 客户端 post 过来的数据，然后根据具体的数据内容来解析自己需要的数据。

参数：

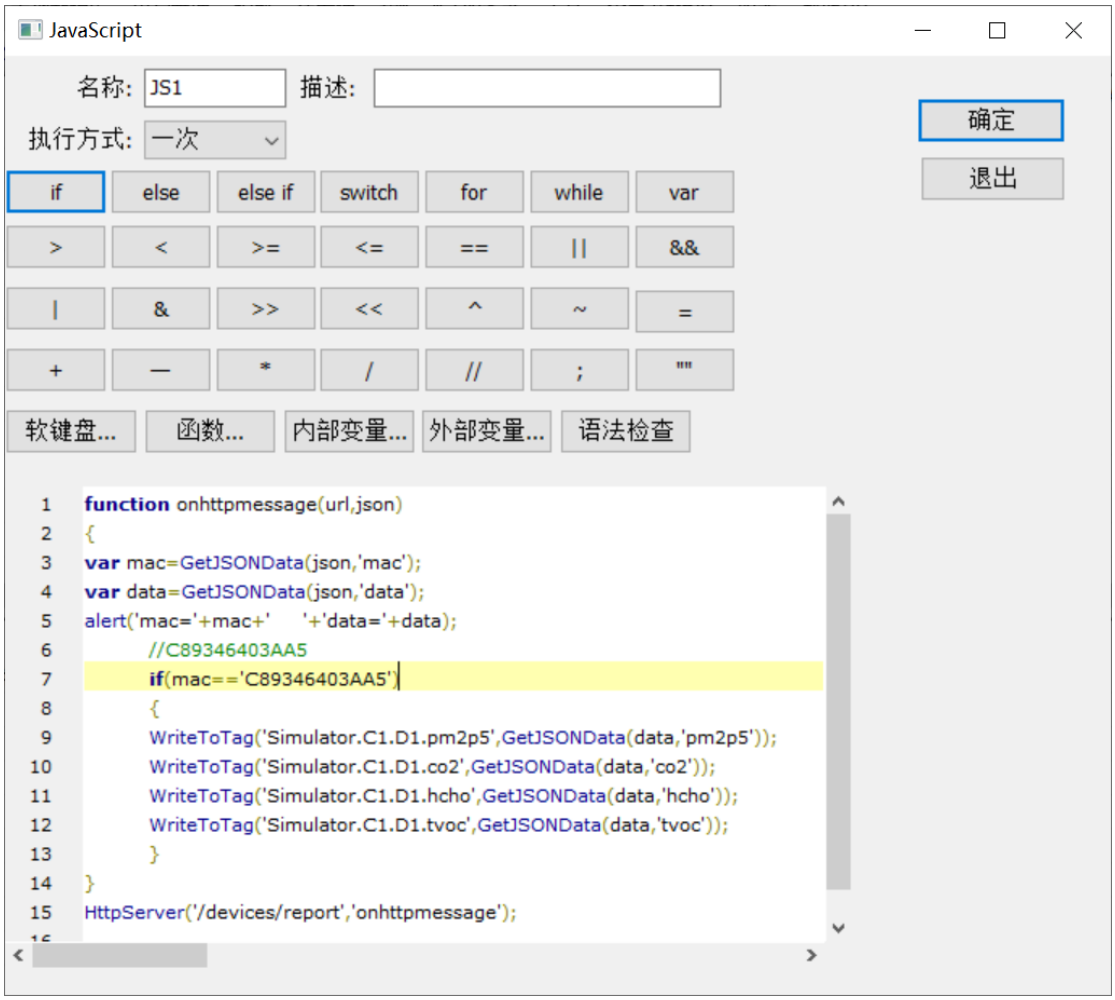
url 为回调 http 客户端的 URL 方法。

json 为回调 http 客户端推送的 json 包。

a 为 http 的 URL 方法。

b 为回调函数名称。

3.34.2 函数操作举例



onhttpmessage(url,json)函数使用 Demo：

```
function onhttpmessage(url,json)
{
var mac=GetJSONData(json,'mac');
var data=GetJSONData(json,'data');
alert('mac='+mac+'      '+'data='+data);
    //C89346403AA5
    if(mac=='C89346403AA5')
    {
    WriteToTag('Simulator.C1.D1.pm2p5',GetJSONData(data,'pm2p5'));
    WriteToTag('Simulator.C1.D1.co2',GetJSONData(data,'co2'));
    WriteToTag('Simulator.C1.D1.hcho',GetJSONData(data,'hcho'));
    WriteToTag('Simulator.C1.D1.tvoc',GetJSONData(data,'tvoc'));
    }
}
HttpServer('/devices/report','onhttpmessage');
```

3.35 NTPDate(x)

3.35.1 函数功能介绍

NTPDate(x);函数，用于同步时间。

参数:x 为字符串，可以为服务器 IP 地址或者域名。

3.5.2 函数操作举例

NTPDate()函数使用 Demo：

NTPDate("pool.ntp.org");

解析：

每天定时同步公网服务时钟。

3.36 SleepEx(x,y)

3.36.1 函数功能介绍

SleepEx(x,y)函数，用于函数异步延迟，单位秒。当 x=1 时，延迟 y 秒时，返回值为 1，进行逻辑判断。

参数:x 为点名，不能直接填写用户自定义变量和数值。

y 为 10 进制数值

3.36.2 函数操作举例

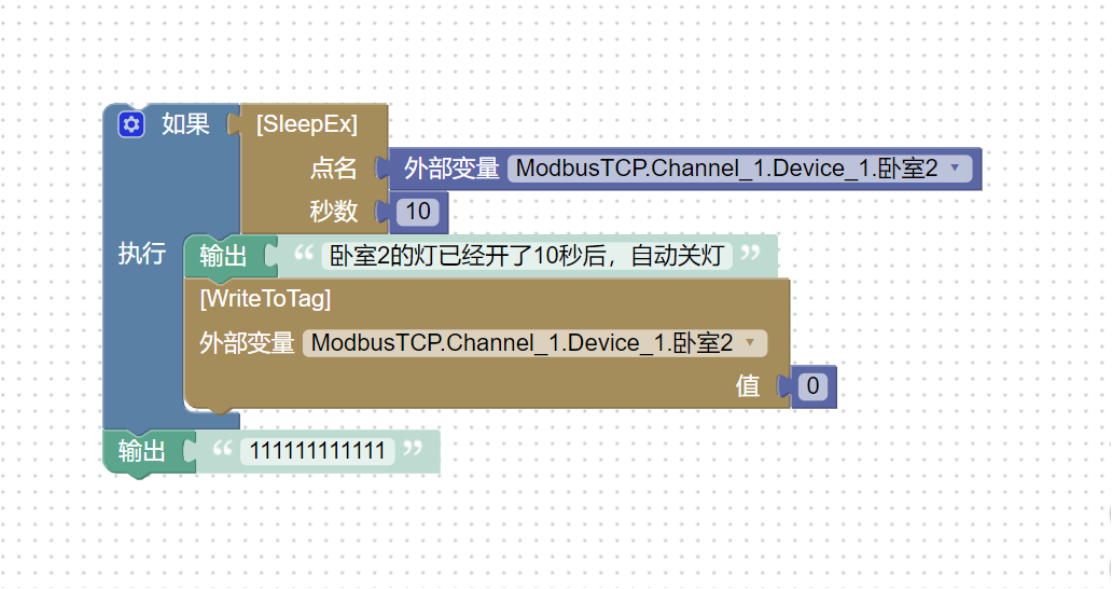
Sleep()函数使用 Demo：

```
//ModbusTCP.Channel_1.Device_1.卧室 2 值为 1 时，执行 SleepEx，延迟 10s，关灯

var vFlag = SleepEx('ModbusTCP.Channel_1.Device_1.卧室 2',10);

if(vFlag){

alert('卧室 2 的灯已经开了 10 秒后，自动关灯');

WriteToTag('ModbusTCP.Channel_1.Device_1.卧室 2',0);

}

alert(111111111111)
```

解析：

ModbusTCP.Channel_1.Device_1.卧室 2 值为 1 时，执行 SleepEx，延迟 10s，关灯

3.36.3 图形编程步骤



3.37 DoHVAC (a,b,c)

3.37.1 函数功能介绍

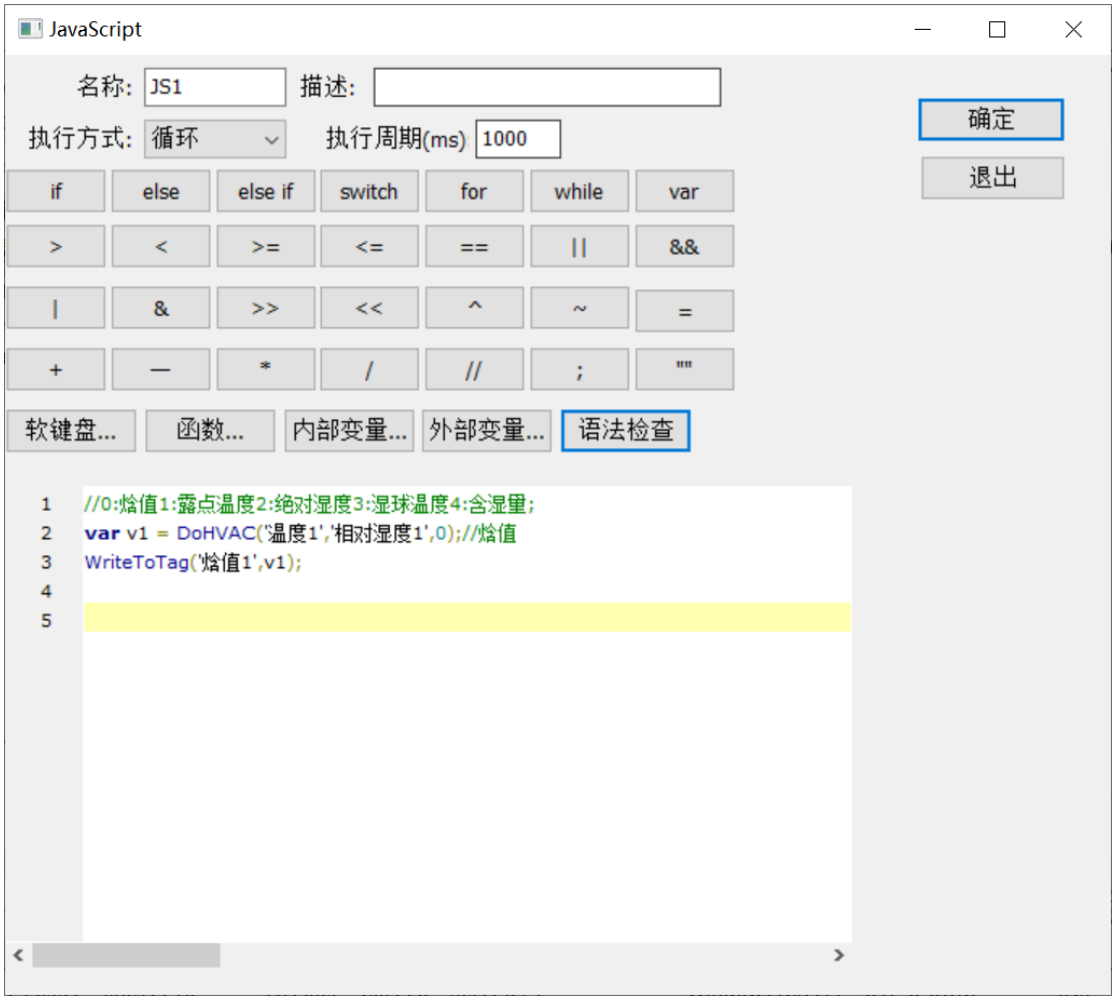
DoHVAC(a,b,c)函数，用户计算露点焓值。

参数:a 为点名，温度点名。

b 为点名，相对湿度点名

c 为 10 进制数值，0:焓值 1:露点温度 2:绝对湿度 3:湿球温度 4:含湿量;

3.37.2 函数操作举例



函数使用 Demo：

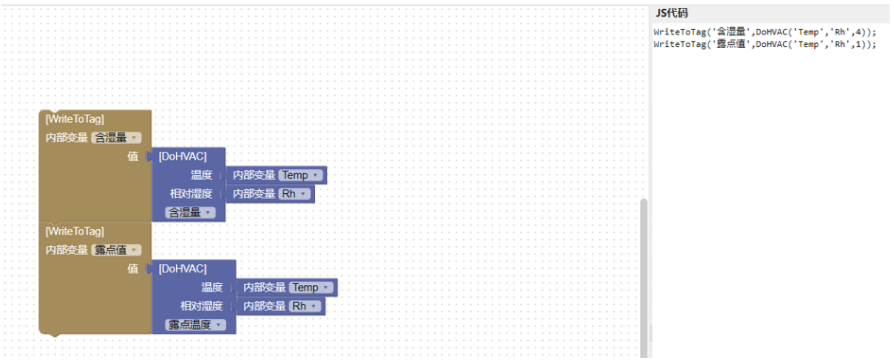
```
//0:焓值 1:露点温度 2:绝对湿度 3:湿球温度 4:含湿量;

var v1 = DoHVAC('温度 1','相对湿度 1',0);//焓值

WriteToTag('焓值 1',v1);
```

解析：

计算焓值，并写给“焓值 1”。

3.37.3 图形编程步骤

四、图形编辑器 DoHVAC 的使用，后边连接的是变量名不能使用 ReadFromTag 函数，因为 ReadFromTag 读取的是一个数值。↵



4 常用 JS 脚本 Demo

4.1 逻辑判断执行赋值操作

Demo 程序：

```
var Tag_1=ReadFromTag('simulator.Channel_1.Device_1.Tag_1');
if(Tag_1==1)
        {
        WriteToTag('simulator.Channel_1.Device_1.Tag_8',100);
        WriteToTag('simulator.Channel_1.Device_1.Tag_7',100);
        }
else
    {
        WriteToTag('simulator.Channel_1.Device_1.Tag_8',0);
        WriteToTag('simulator.Channel_1.Device_1.Tag_7',0);
```

```
}
```

Demo 说明：

读取外部变量 simulator.Channel_1.Device_1.Tag_1 的值，然后进行判断，如果值为 1 将其它的点位赋值 100，如果为 0 将其他点位赋值 0。

4.2 群控/总控/联控

Demo 程序一（使用变化模式）：

```
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_1');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_2');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_3');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_4');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_5');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_6');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_7');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_8');
```

Demo 程序一说明：

通过控制内部变量 Start_Stop 的值控制其他的点，脚本的执行只会在内部变量 Start_Stop 的值变化后对所有的点位进行操作 ，此方

法实现群控时，当其中的某个的设备需要单独控制也可以进行当个设备的控制操作。

Demo 程序二（使用循环模式）：

```
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_1');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_2');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_3');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_4');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_5');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_6');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_7');
MoveValue('Start_Stop','simulator.Channel_1.Device_1.Tag_8');
```

Demo 程序二说明：

将内部变量 Start_Stop 作为一个总控制点，将总控制点的数值赋给其他的点位进行群控，脚本的执行是按照设定的周期进行循环的执行的。此方法实现群控时，当其中的某个的设备需要单独控制时写入操作是无效的，默认其他点位的状态只和总控制点保持一致，分控制点的操作无效。

4.3 两个从站设备之间数据通讯

Demo 程序：

```
MoveValue('simulator.Channel_1.Device_1.Tag_1','simulator.Channel_2.Device_1.Tag_1');

MoveValue('simulator.Channel_1.Device_1.Tag_2','simulator.Channel_2.Device_1.Tag_2');

MoveValue('simulator.Channel_1.Device_1.Tag_3','simulator.Channel_2.Device_1.Tag_3');

MoveValue('simulator.Channel_1.Device_1.Tag_4','simulator.Channel_2.Device_1.Tag_4');

MoveValue('simulator.Channel_1.Device_1.Tag_5','simulator.Channel_2.Device_1.Tag_5');

MoveValue('simulator.Channel_1.Device_1.Tag_6','simulator.Channel_2.Device_1.Tag_6');

MoveValue('simulator.Channel_1.Device_1.Tag_7','simulator.Channel_2.Device_1.Tag_7');

MoveValue('simulator.Channel_1.Device_1.Tag_8','simulator.Channel_2.Device_1.Tag_8');
```

Demo 程序说明：

通过 JS 脚本内的 MoveValue( )函数将 Channel.Device_1 设备的数据赋给 Channe2.Device_1 设备，这样就实现了两个从站设备之间的通讯。

4.4 JS 操作串口读取 modbus RTU 仪表数据

Demo 程序：

```
var send= new Array();                    //定义 send 数组
send[0]=0x01;                             //站号
send[1]=0x03;                             //功能码
send[2]=0x00;                             //起始地址高字节
```

```
send[3]=0x0A;                                    //起始地址低字节
send[4]=0x00;                                    //长度高字节
send[5]=0x0A;                                    //长度低字节
var crc = DoCRC16(0,5,send);                     //CRC 校验
send[6]=crc;                                     //数组拼接完成
alert('发送:'+send);
var mes=SendTextToPort(3,9600,8,1,0,1000,send);  //发送请求报文,返回 10 进制数组
alert('接收:'+mes);                              //打印数组
var data1_float=MakeFloat(mes[6],mes[5],mes[4],mes[3]);   //算 32 位单精度 Float
var data1_long=MakeLong(mes[10],mes[9],mes[8],mes[7]);  //算 32 位长整数 Long
var data1_word=MakeWord(mes[12],mes[11]);        //算 16 位 Word
WriteToTag('s.Ch1.D1.test1',data1_float);        //写给内部变量 1
WriteToTag('s.Ch1.D1.test2',data1_long);         //写给内部变量 2
WriteToTag('s.Ch1.D1.test3',data1_word);         //写给内部变量 3;
```

Demo 程序说明：

通过 JS 脚本内的 SendTextToPort ( )函数操作串口发送 Modbus 命令，得到设备回应之后，取相应字节进行处理计算出当前值，并写给网关内部变量。

4.5 JS 操作网口读取 modbusTCP 数据

Demo 程序：

```
var Arr_Read=new Array();                    //定义数组
Arr_Read[0]=00;                              //包头起始
Arr_Read[1]=01;
Arr_Read[2]=00;
Arr_Read[3]=00;
Arr_Read[4]=00;
Arr_Read[5]=06;                              //包头结尾
Arr_Read[6]=01;                              //地址站号
Arr_Read[7]=03;                              //Modbus 功能码
Arr_Read[8]=00;                              //起始地址高字节
Arr_Read[9]=00;                              //起始地址低字节
Arr_Read[10]=00;                             //数据长度高字节
Arr_Read[11]=02;                             //数据长度低字节
var Arr_Received=SendTextToTCP('192.168.1.189',502,100,Arr_Read);
//发送数据请求，接收报文为 Arr_Received 十进制数组中
WriteToTag('Simulator.Channel_1.TCP.40001',MakeWord(Arr_Received[
10],Arr_Received[9]));
WriteToTag('Simulator.Channel_1.TCP.40002',MakeWord(Arr_Received[
12],Arr_Received[11]));
```

Demo 程序说明：

通过 JS 脚本内的 SendTextToTCP()函数操作网口发送 Modbus 命令，得到设备回应之后，取相应字节进行处理计算出当前值，并写给网关外部变量。

4.6 JS 操作网口读取 modbusUDP 数据

Demo 程序：

```
var Arr_Read=new Array();              //定义数组
Arr_Read[0]=00;                        //包头起始
Arr_Read[1]=01;
Arr_Read[2]=00;
Arr_Read[3]=00;
Arr_Read[4]=00;
Arr_Read[5]=06;                        //包头结尾
Arr_Read[6]=01;                        //地址站号
Arr_Read[7]=03;                        //Modbus 功能码
Arr_Read[8]=00;                        //起始地址高字节
Arr_Read[9]=00;                        //起始地址低字节
Arr_Read[10]=00;                       //数据长度高字节
Arr_Read[11]=02;                       //数据长度低字节
var Arr_Received=SendTextToUDP('192.168.1.189',502,100,Arr_Read);
//发送数据请求，接收报文为 Arr_Received 十进制数组中
WriteToTag('Simulator.Channel_1.UDP.40001',MakeWord(Arr_Received[
10],Arr_Received[9]));
WriteToTag('Simulator.Channel_1.UDP.40002',MakeWord(Arr_Received[
12],Arr_Received[11]));
```

Demo 程序说明：

通过 JS 脚本内的 SendTextToUDP()函数操作网口发送 Modbus 命令，得到设备回应之后，取相应字节进行处理计算出当前值，并写给网关外部变量。

4.7 发送 POST 请求并获取 XML 中的 Attribute 属性

Demo 程序：

```
//定义请求参数
var request ='<soap:Envelope><soap:Body><GetFirmWareInfo/></soap:Body></soap:Envelope>';
//Post 请求 xml 格式
var  xml = HttpPost('http://192.168.1.244/soap/GetFirmWareInfo', 'text/xml',  request);
//获取 xml 中的 Attribute 属性
var ns = GetXMLAttrib(xml, '/soap:Envelope/soap:Body/GetFirmWareInfoResponse/GetFirmWareInfoResult', 'MachineCode');
//将获取到的数据写入外部变量
WriteToTag('Simulator.Channel_1.GetXMLAttrib.MachineCode',ns);
```

Demo 程序说明：

通过 JS 脚本内的 HttpPost()函数发送 Post 请求获取相应的 XML，然后通过 JS 脚本中的 GetXMLAttrib 函数获取 XML 中的所需属性，并写给网关外部变量。

4.8 发送 POST 请求并获取 XML 中的 Data 数据

Demo 程序：

```
//定义请求参数
var request ='<soap:Envelope><soap:Body><GetCloudUrl/></soap:Body></soap:Envelope>';
```

```
//Post 请求 xml 格式
var  xml = HttpPost('http://192.168.1.68/soap/GetCloudUrl','text/xml', request);
//获取 xml 中的 data 数据
var res = GetXMLData(xml, '/soap:Envelope/soap:Body/GetCloudUrlResponse/GetCloudUrlResult');
//将获取到的数据写入外部变量
WriteToTag('Simulator.Channel_1.GetXMLData.MQTTIP',res);
```

Demo 程序说明：

通过 JS 脚本内的 HttpPost()函数发送 Post 请求获取相应的 XML，然后通过 JS 脚本中的 GetXMLData 函数获取 XML 中的所需数据，并写给网关外部变量。

4.9 发送 GET 请求并 GetJSONData 获取 Json 中的属性数据

Demo 程序：

```
//定义 GET 对象的 URL
var URL='http://api.k780.com/?app=weather.today&weaid=1&appkey=10003&sign=b59bc3ef6191eb9f747dd4e83c99f2a4&format=json';
//发送 GET 请求获取 JSON
var data = HttpGet(URL,'application/json');
alert(data);

//获取 JSON 串中的 day
var Obj_A=GetJSONData(data,'days');
alert(Obj_A);
```

```
//获取 JSON 串中的 week
var Obj=GetJSONData(data,'week');
alert(Obj);
```

Demo 程序说明：

通过 JS 脚本内的 HttpGet()函数发送 Get 请求获取相应的 Json 串，然后通过 JS 脚本中的 GetJSONData 函数获取 JSON 中的所需数据,并写给网关外部变量。